# DATA SCIENCE USE CASES IN THE MANUFACTURING INDUSTRY
FROM THEORY TO PRACTICE

Diego Alejandro Arenas Contreras

A Thesis Submitted for the Degree of EngD
at the
University of St Andrews

2022

Data Science use cases in the Manufacturing Industry

From theory to practice.

Diego Alejandro Arenas Contreras

University of
St Andrews

This thesis is submitted in partial fulfilment for the degree of
Doctor of Engineering (EngD)
at the University of St Andrews

February 2022

Data Science use cases in the Manufacturing Industry

From theory to practice

Diego Alejandro Arenas Contreras

Abstract

One of the main challenges organisations face today is supporting business decisions from the massive volumes of data they are continuously collecting. The problem for organisations is how to become a *data-driven* organisation using the data they collect to generate insights and repeatable solutions connecting information needs with usable data products.

Our objectives during the doctorate were to research and implement high quality technological and methodological solutions following best practices from academia and industry and, at the same time, build internal capacity for the organisation from experience.

We implemented a series of data-related projects. The projects can be classified into two types. There are foundational projects that build infrastructure and processes to analyse data and applied data projects. Our methods included practices from software engineering, data science, and data engineering. We designed and built data solutions based on the principles of *scalability*, *automation*, *encapsulation*, and *abstraction*.

We followed the principles mentioned above from the design phases of the projects; this allowed us to achieve good integration with the current systems and infrastructure of the organisation. We operationalised the technologies we explored for each project using a use-case driven approach. Users and stakeholders were involved early on on the projects, and we maintained excellent and continuous communication with them.

The foundational projects implemented data architectures rather than implementing a specific ad-hoc solution so that the projects adjusted well to changing requirements and were generalisable to be reused entirely or components of the solutions in future projects. We used the foundational projects in the applied data projects.

We deployed an estimation model to quantify the number of technicians needed to support an on-site project. Using an API to query the model, we used a microservice architecture exposing the final model to be consumed. We designed and implemented the analysis of estimating the lifespan of batteries using survival analysis and spectral clustering techniques. We ranked specific machines from best to worst performance based on their fuel consumption to optimise resources on project sites. We designed and implemented a Python custom package to facilitate the exploration of databases for data science and data engineering projects. We designed and implemented a microservices architecture to support data streaming analytics. We made recommendations on using a machine learning framework to track and monitor machine learning models, wrote guidelines for best practices, and delivered internal tutorials about the use and benefits of these kinds of solutions. We imple-

mented a data-driven architecture to support the analysis of telemetry data from multiple data sources. We implemented an alarm system on top of the solution using the analytical database of the project. Finally, we designed and implemented a custom Python package to handle repeatable data engineering tasks for the data engineering team.

Data science and data engineering are new and essential roles in companies that aim to become data-driven organisations. We believe that using software engineering and software development techniques contributes significantly to this organisational change and accelerates internal innovation using data.

We promptly provided data and information to the stakeholders to support their information needs and decision-making processes.

Dedication

To my parents María Eliana and Raúl.

Declarations

Candidate's declaration

I, Diego Alejandro Arenas Contreras, do hereby certify that this thesis, submitted for the degree of EngD, which is approximately 49,000 words in length, has been written by me, and that it is the record of work carried out by me, or principally by myself in collaboration with others as acknowledged, and that it has not been submitted in any previous application for any degree. I confirm that any appendices included in my thesis contain only material permitted by the "Assessment of Postgraduate Research Students" policy.

I was admitted as a research student at the University of St Andrews in October 2016.

I received funding from an organisation or institution and have acknowledged the funder(s) in the full text of my thesis.

21.10.2022

Date Signature of candidate

Supervisor's declaration

I hereby certify that the candidate has fulfilled the conditions of the Resolution and Regulations appropriate for the degree of EngD in the University of St Andrews and that the candidate is qualified to submit this thesis in application for that degree. I confirm that any appendices included in the thesis contain only material permitted by the 'Assessment of Postgraduate Research Students' policy.

Date　　　　Signature of supervisor

Permission for publication

In submitting this thesis to the University of St Andrews we understand that we are giving permission for it to be made available for use in accordance with the regulations of the University Library for the time being in force, subject to any copyright vested in the work not being affected thereby. We also understand, unless exempt by an award of an embargo as requested below, that the title and the abstract will be published, and that a copy of the work may be made and supplied to any bona fide library or research worker, that this thesis will be electronically accessible for personal or research use and that the library has the right to migrate this thesis into new electronic forms as required to ensure continued access to the thesis. I, Diego Alejandro Arenas Contreras, confirm that my thesis does not contain any third-party material that requires copyright clearance. The following is an agreed request by candidate and supervisor regarding the publication of this thesis:

Printed copy

No embargo on print copy.

Electronic copy

No embargo on electronic copy.

21.10.2022

Date Signature of candidate

Date Signature of supervisor

Underpinning Research Data or Digital Outputs

Candidate's declaration

I, Diego Alejandro Arenas Contreras, hereby certify that no requirements to deposit original research data or digital outputs apply to this thesis and that, where appropriate, secondary data used have been referenced in the full text of my thesis.

21.10.2022

Date Signature of candidate

General acknowledgements

Firstly, I would like to thank my academic supervisor, Dr Simon Dobson, for his endless and continuous academic and personal support during the EngD. For his invaluable guidance through the process of researching and applying science at the same time.

I would like to thank my industrial supervisor Elizabeth Hollinger for her guidance and facilitation during the projects presented in this thesis. Also, to Steven Faull, my first industrial supervisor, for his support during my first year at Aggreko.

To my colleagues at Aggreko, Raymond Callahan, Collin Parry, John Stewart, Soner Candan, and Lyndsey McKirdy, with whom I had the pleasure of working on several data-related projects and collaborating personally and through them to all my colleagues at Aggreko that it would be difficult to name them all.

To my friends and colleagues Simon C. Smith, Felipe Espic, Lissette Áviles, Pablo Escárate, Paulina Bravo, Pamela Villamar, and Alejandro Gutierrez for our endless conversations, support, and their friendship in the last five years.

Finally, last but not least, to my extended family, for providing their support during my studies abroad.

Funding

This work was supported by Aggreko PLC.
This work was supported by The Datalab in Scotland, REG-17465.

Contents

1 Introduction 17

1.1 Structure 20

1.2 Side projects 22

1.3 The organisation 23

I Foundational Projects 24

2 Automated Exploratory Data Analysis 25

2.1 Introduction 25

2.2 Background 26

2.3 Related Work 27

2.3.1 Data Exploration in Data Mining Methodologies 27

2.4 Design 28

2.4.1 Assumptions 28

2.4.2 Exploratory Data Analysis Tasks 28

2.4.3 SQL code generation 29

2.4.4 Data Governance and Data Quality 30

2.4.5 Candidate Key Search 30

2.4.6 Performance 33

2.4.7 Data Discovery 34

2.5 Evaluation 34

2.5.1 Support 35

2.6 Conclusions 35

2.6.1 Future work 35

3 A Streaming Analytics Architecture 37

3.1 Introduction 37

3.2 Problem Statement 38

3.3 Background 39

3.4 Design 40

3.4.1 The data 43

3.4.2 Architecture set up 43

3.4.3 Components of the Architecture 44

3.4.4 Connecting the architecture 47

3.5 Results 47

3.6 Evaluation 48

4 Machine Learning models governance 49
4.1 Introduction 49
4.1.1 Benefits of ML Governance 50
4.1.2 Contributions 50
4.1.3 Structure of the project 51
4.1.4 Terms, Definitions, and Acronyms 51
4.2 Problem Statement 52
4.3 Background 53
4.3.1 Challenges in ML model management 54
4.4 Literature Review 54
4.5 Technology Review 56
4.5.1 MLFlow 57
4.5.2 Azure Machine Learning (AML) 58
4.5.3 Tensorflow Extended (TFX) 58
4.5.4 Other platforms 59
4.5.5 Interpretability 59
4.5.6 Coding Guidelines 60
4.5.7 DataOps 64
4.6 Evaluation 64
4.6.1 Security 65
4.6.2 Cost 65
4.6.3 Recommendation 65

5 Analytics repository for telemetry data from IoT projects 68
5.1 Background 68
5.1.1 Data Silos 69
5.1.2 Description of the Problem 69
5.1.3 Proactive and Predictive Alarms 70
5.2 Problem Statement 71
5.3 Architecture Design 71
5.3.1 Architecture and components 72
5.4 Benefits of a single data repository 75
5.5 Data Model Design 75
5.5.1 The data 76
5.5.2 The flow of the data 77
5.5.3 Reverse Engineering 77
5.5.4 Analysis of the data source 78
5.6 Implementation 78
5.6.1 Data Engineering 79
5.6.2 Data Science 80
5.7 Deployment 85
5.8 Discussion 85
5.8.1 Further analysis 86

6 Aggreko Data Engineering Library 87
6.1 Introduction 87
6.2 Background 88
6.2.1 A brief history of the information systems 88
6.3 Description of the Problem 89

6.3.1 Benefits of a custom Python library . . . 90
6.4 Design . . . 90
6.4.1 Analysis of the notebooks . . . 91
6.4.2 Team coordination . . . 91
6.4.3 Domain-Driven Design . . . 92
6.4.4 Code structure . . . 93
6.5 Implementation . . . 94
6.5.1 Testing . . . 96
6.5.2 Deployment . . . 96
6.6 Discussion . . . 97

II Applied Data Projects 98

7 Manning Optimisation 99
7.1 Background . . . 99
7.1.1 The current model . . . 100
7.1.2 The Regression Models . . . 100
7.1.3 Benefits of an estimation model for manning optimisation . . . 102
7.2 Problem Statement . . . 102
7.3 Planning . . . 102
7.3.1 Methodology . . . 103
7.4 Design . . . 104
7.4.1 Software Architecture . . . 104
7.4.2 Data . . . 107
7.4.3 API . . . 107
7.5 Implementation . . . 108
7.5.1 Deployment . . . 109
7.6 Evaluation . . . 109
7.7 Phase 2 . . . 111
7.7.1 Planning . . . 111
7.7.2 Modelling . . . 113
7.8 Conclusions . . . 115
7.9 Further development . . . 115

8 External Fuel Tank Battery Analysis 117
8.1 Introduction . . . 117
8.2 Background . . . 117
8.3 Problem Statement . . . 118
8.4 Methodology . . . 118
8.4.1 Project Plan . . . 119
8.4.2 Data Exploration . . . 120
8.4.3 The data . . . 121
8.5 Data Analysis . . . 122
8.5.1 Survival Analysis . . . 123
8.5.2 Spectral Clustering . . . 124
8.6 Conclusion . . . 127
8.6.1 Further analysis . . . 127

9 Fuel consumption rate 128
9.1 Introduction . . . 128
9.2 Methodology . . . 129
9.2.1 Guidelines . . . 129
9.3 Experimental questions . . . 131
9.3.1 Filter differential pressure . . . 132
9.3.2 Impact of low-quality oil . . . 132
9.3.3 Impact of fuel quality . . . 133
9.3.4 Vibration measurement . . . 133
9.3.5 Determine better or worst performing assets . . . 133
9.3.6 Sensors added value . . . 133
9.4 Implementation . . . 134
9.4.1 Data Exploration . . . 134
9.4.2 Data preprocessing . . . 135
9.4.3 Data Analysis . . . 135
9.5 Results . . . 137
9.6 Further development . . . 138

10 Conclusions 139

A Appendix: Survey questions 143

B Appendix: Infrastructure configuration 145
B.1 Docker . . . 145
B.2 Kafka Server . . . 145

C Appendix: Other contributions 147

Acronyms

AML Azure Machine Learning. 52, 57, 58

CD/CI Continuous Development & Continuous Integration. 54

EngD Engineering Doctorate. 22, 139, 147

IaaS Infrastructure as a Service. 42, 43

IoT Internet of Things. 20, 68

ML Machine Learning. 49, 54, 57

ONNX Open Neural Network Exchange Format. 57

PaaS Platform as a Service. 42, 43

PFA Portable Format Analytics. 57

PMML Predictive Model Markup Language. 57

RL Reinforcement Learning. 55, 63

SaaS Software as a Service. 42, 43

SCADA Supervisory Control And Data Acquisition. 69

SD Software Development. 54

XAI Explainable Artificial Intelligence. 56

Glossary

Data Analyst A person with knowledge about data manipulation for business processes. Often creating reports for managers. They often have skills of the SQL language. They use to work mostly with structured data.. 26

Data Engineer A person working with data transformations and able to integrate and combine data sources. Creation of data processes or data pipelines for the organisation with focus on the efficiency of the data processes.. 26

Data Lake A central data repository that stores data source in original format that allows data transformations and the combination of data sets from multiple sources to present a unified version of the data.. 39

Data Scientist A person working on data analysis with enough knowledge in statistics, maths and data modelling, able to use advanced algorithms and interpret the results providing insights from the advanced data analysis. They are able to work with structured, semi-structured, and unstructured data. They are able to design and process massive volumes of data for their analysis.. 26

Power Solutions A line of business of the company that has long term projects with customers.. 132

Rental Solutions A line of business of the company that deals with short-term projects.. 99

Chapter 1

Introduction

Your technology stack is already obsolete. Today Data Technologies offer a myriad of options to drive data analytics projects. There is a constant evolution in the types of data processing technologies to solve the same problems. Innovation happens at a quick pace. It is necessary to continue experimenting and testing the new technologies available in the market.

This thesis document presents a series of data-related projects executed during the author's engineering doctorate programme at Aggreko. The common denominator of the information needs that triggered the data science and data engineering projects described in this document is the aim to become a *data-driven* organisation. It is believed that the work presented in this thesis has contributed to achieving this status.

This document presents the following specific contributions:

- The deployment of a model to estimate the number of technicians required to support unplanned failures of machines in a given project site.
- The analysis of the lifespan of batteries of the sensor attached to external fuel tanks.
- The estimation of fuel consumption to rank best and worst-performing assets regarding their fuel consumption under different workloads.

There are also general contributions with tools and processes that were designed and built to complete the specific contributions. The general contributions are:

- An open-source Python library to automate the exploration of databases.
- A streaming analytics architecture.
- A machine learning governance framework recommendation to track the development and deployment of machine learning models.
- The design and implementation of an analytics data repository for telemetry data.
- An internal Python library for data engineering tasks.

The design and implementation of the data science and data engineering projects presented in this thesis follow some basic principles from software engineering. These principles may sound like common sense, but they represent an invaluable contribution in the history of computer science, software development, and now becoming part to the new fields of data science and data engineering. The software engineering practices contribute significantly to the success of the individual projects presented in this document.

What is the path to becoming a data-driven organisation? Many companies follow the example of big tech companies like Facebook, Amazon, Google or LinkedIn, incorporating their data tools and practices. But we need to recognise that most of the companies in the world will not necessarily handle the same volumes of data as the big tech companies. This is not considered when selecting the technology stack or the toolset used by a team to develop products. Not only in volume but the number of concurrent users.

It can be argued that a data-driven organisation uses facts from data to support its decisions. The technological solutions implemented in a data-driven organisation must provide timely access to the information from data. Considering the *access to information* premise in mind when designing technological solutions can significantly impact the results and contribution of the delivered projects. In general, it should be avoided to implement technologies just because they seem helpful without a use case to evaluate the implementations. But first, it is necessary to find the *right question* and then find the *right* technologies to solve it.

Note on the language used in this document. The work presented in this EngD thesis was completed by the author, but in several paragraphs the author will use the terms “we” and “ours” to refer to the tasks as a team effort even when the tasks were completed by the author unless is stated differently. This voice felt natural when describing the projects involving the supervision by Prof. Simon Dobson, the collaboration with data professional colleagues and the involvement of stakeholders.

Challenges with technology stacks

A challenge that companies face when choosing their technology stack is evaluating the benefits and costs of adding a new technology compared with their current technology stack. This balance is often a constraint for the data teams when deciding on the use of new technologies. Sometimes technologies are appealing to use, but it is necessary to evaluate the impact in the long term and match it with the organisation data strategy. *Does the new tool contribute to achieving the strategic goals of the organisation efficiently?* How the new technology can *amplify* the work of the data analysts, or how it can optimise costs and resources used to add value to the business units.

For example, the use of multiple programming languages within a team could be an advantage because it grants flexibility to solve problems based on the skills of the analysts, but this also becomes an issue when we think about maintaining the solutions. Having multiple programming languages can be a bottleneck to further developing new solutions or maintaining existing ones. Standardising the analyst’s development tools is a productivity gain over time.

The current technology stack of the organisation[1] tools and processes are the starting point for any of the evaluation of technologies for the EngD projects. There was special consideration to not creating technical debt[2] [52] nor the intention to create isolated solutions that are separated from the organisation's data processes.

The approach

The solution to finding the *right* technologies for a particular data problem is to establish a process rather than assessing tools and frameworks individually. The suggested process uses software engineering practices applied to data science as principles when designing and planning solutions to data problems.

The first principle of software engineering is to think in terms of *Scalability* [47]. *Scalability* is one of the principles we followed across the different projects presented in this thesis. An idea behind each of the implemented projects was that they would be able to scale out their current demand for processing power or data storage. Sometimes it is necessary to refactor the solution to make it scalable when the throughput requirements increase. Scalability matters because data teams must prepare organisations for the increasing volumes of data they are collecting. The technological solution developed today is desired to handle a growing amount of data in five or ten years.

Another pattern that emerges from studying data-driven organisations and their leaders is identifying and automating repeatable processes. Automating tasks can free up resources in terms of time, knowledge and costs. The time spent automating a solution can have a considerable return on investment. This automation principle is applied whenever possible in the EngD projects presented in this document, as seen in the following chapters. Automating repeatable work frees time to develop new projects and ensures automated processes include quality testing.

Along with the principle of Scalability, it is necessary to consider the *extensibility* principle. Extending the solution should be easy enough for the data team to extend the project after the deployment is completed. The analyst should consider the extensibility of the solution from the design phase.

Highly interrelated with the scalability and extensibility principle, there are the principles of *abstraction* and *encapsulation*. These principles are transversal to the designs and implementations found in this thesis document. They naturally appeared in the data science and data engineering designs formulated in the multiple projects built in the last four years.

The concepts of *abstraction* and *encapsulation* are not new to software engineering. On the contrary, software development practices have been advocating for them for many decades as best practices to follow. And now, these best software engineering practices can be part of the new role of data scientists [58]. There is a transition from *data insights* to *data products*, making necessary the use of software development techniques to be able to deliver the value from the insights with quality and sustainability [43].

[1] The technology stack includes some constraints on the use of key technologies like Microsoft Azure as the default cloud provider, Databricks as the central platform to process data, and using Python as the standard language for data analysis.

[2] Technical debt is understood as increasing the cost and effort of future changes in the software due to poor design in the early stages of projects.

Complexity in software should be abstracted and encapsulated into functions and modules to test them and ensure their quality and then used in the code.

The progress in big data analytics is given by the capacity of the new solutions to encapsulate and abstract complex processes for the analyst to make use of them without the need to implement the complexity themselves. A good example is the abstraction and encapsulation of parallel and distributed processing on clusters. This technology has been available for many years to programmers, but they had to implement it from scratch. When a solution abstracts and encapsulates the complexity of distributed computing processing, it gets private companies' interest to leverage advanced ways to process big data volumes.

The fundamental principles followed on each one of the projects presented in this document are: *scalability*, *automation*, *encapsulation*, and *abstraction*.

A secondary set of principles were followed in the projects, such as *integration* with existing systems—the operationalisation of the solutions and being *use-case driven* instead of testing technologies for the sake of testing them.

Integration with existing systems is essential. Designing big data architectures can bring a lot of disruptive innovation to the organisation, but this will not be sustainable if every new system brings disruptive changes to the ecosystem of tools that the organisation currently uses. Any new technological solution should consider the current systems and evaluate the best ways to integrate with the existing solution within the organisation.

Any new technological implementation should be tested along with a use case. The implementation should have a clear, tangible goal and evidence of accomplishment. The implementation of projects should be use case driven.

1.1 Structure

This thesis document is divided into two parts. The first part contains five, and the second part has three chapters. Each chapter presents the design and implementation of a project. The projects presented in Part I are considered foundation projects. They built capacity within the organisation to implement the projects presented in Part II. The projects in Part II describe applied projects with narrowed objectives and often make use of the projects presented in Part I.

The EngD projects are highly related. The foundational projects of Part I made it possible to work on the applied EngD projects of Part II of this thesis. Data exploration activities played a significant role across the different projects.

Part I presents an automated data exploration tool, a streaming analytics data architecture, a machine learning governance framework selection, an analytics repository for Internet of Things (IoT) projects, and the implementation of a data engineering library.

Chapter 2 presents an automated tool to explore the content of databases. This tool is used extensively in other projects presented in this thesis. The tool helped with the data exploration of databases that there was no prior knowledge about their content, it was used to find relationships between tables in databases, and a feature implemented in the tool was used to find the candidate primary keys of two hundred and twenty tables that otherwise would have required many hours of manual labour.

The extensive use of the automated tool for exploratory analysis across the different projects presented in this document highlighted the relevance of the data exploration phase in data science projects. How quick an organisation can be aware of its data? This question is relevant because it represents an advantage over competitors and leads to improved processes to have quick responses for ad-hoc analysis. The *analytical question* sometimes is redefined based on the data available, and the data available can only be determined by the exploratory data analysis.

The tool was designed to abstract the repeatable tasks that analysts have to perform to explore the content of a database. It encapsulates different queries into functions, and it generates the necessary SQL code to query the database.

The tool is based on the code from the MSc dissertation project [54] that automatically searches for relationships between relational tables.

In Chapter 3 presents a streaming data processing architecture. Using the principles of scalability and extensibility to design a solution able to scale out depending on the demand. This was an assessment of the organisation's big data capabilities to respond to increasing demand for information needs.

The design was modularised, implementing an event-driven data architecture. This was the first EngD project, and the software engineering principles mentioned above can be seen applied to the design and implementation of this project.

In Chapter 4 multiple frameworks for machine learning governance are evaluated. A recommendation was made based on criteria established at the beginning of the EngD project. A set of recommendations were made as best practices for data science and machine learning projects. The use of interpretable machine learning in the data science projects was recommended for the analysts to incorporate it into the machine learning development process.

In Chapter 5 an analytical database is implemented to store and analyse telemetry data from remote devices deployed around the world. Data transformation processes were built to ingest the data, and proactive alarms were developed using the data from the database. The components of the architecture were modularised using the design principles of abstraction and encapsulation.

In Chapter 6 presents the design and implementation of a data engineering library. Encapsulating repeatable data transformation tasks and wrapping them into functions in a custom Python library. We created modules for the different components. Each module contained scripts that abstracted the complexity of data transformation tools. We designed the library to be extendable.

We created the processes to deploy the library to clusters of data processing. We created guidelines and recommendations to develop and extend the library. We held hands-on sessions with the data engineering team to work with Git repositories and collaborative work.

In Part II, we present the deployment of a forecast model to estimate the number of experts necessary to provide support to run a project site, an analysis of the lifespan of batteries attached to sensors, and a model to rank generators based on their fuel consumption rates.

In Chapter 7 we present a data science project where we transferred a solution from spreadsheets to a three-tier architecture solution in the cloud. We deployed an application to estimate the number of experts and crew members to run and maintain the machine deployed to a project site given the initial conditions of the project and its location.

We explored and analysed the data sources to update the forecasting model.

In Chapter 8 a data science project to understand the lifespan of batteries attached to fuel level sensors is described. Using analysis such as *survival analysis* and *spectral clustering* to provide information to the stakeholders to make decisions on the maintenance of the machines.

We explored the data source and found the tables that contained the data we needed to implement the analysis.

In Chapter 9 we explained the process to prioritise question to design experiments. We presented a framework to design and run experimental designs. We advised for the design of experiments and wrote guidelines for other team members to start using statistical analysis to answer questions for which there is not enough data.

We selected six analytical questions by process of voting and elimination. We prioritised the implementation of the projects. We implemented a ranking of the generators present in a project site based on their fuel consumption performance. This ranking will help to prioritise the order of the generators that will be connected to the grid under variable demand using the most efficient generators.

Finally, Chapter 10 presents the conclusions.

1.2 Side projects

I had the privilege to work on interesting side projects during my Engineering Doctorate (EngD). The side projects were a way to continue with my learning path and professional and personal self-development [91], [84]. I often used the gained knowledge from these side projects in the projects presented in the chapters of this thesis.

These side projects greatly enriched the potential contributions of each EngD project. The experience acquired by joining teams and development groups provided the experience to tackle and lead some of the data science and data engineering projects of the EngD.

In 2018, two position papers were written by Diego on Data for Good. *The Case for Data For Good*[3] and *Scalable Digital Volunteering: A Data for Social Good Marketplace*[4] presenting the need to scale the volunteering work using digital

[3] Downloadable at https://darenasc.github.io/files/TheCaseForDataForGood.pdf (accessed 25 October 2021).

[4] Downloadable at https://darenasc.github.io/files/ScalableDigitalVolunteering.pdf (accessed 25 October 2021).

and collaborative platforms. The other paper explains the design of a platform for collaboration on data science projects for the common good.

The design of that platform data science platform for Data for Good led to the collaboration with a working group about that was about to start working on a blueprint for a Data Safe Haven. The overlapping of Data Safe Haven ideas and a data platform for processing and storing data for data science projects led me to join the group. Diego was invited as a Visiting Researcher at the Alan Turing Institute, we he contributed to the work published on this paper on *Design choices for productive, secure, data-intensive research at scale in the cloud* [86].

Around the same time, Diego joined the development group of a new and promising machine learning library written in the Julia[5] programming language. He was one of the first maintainers of the library *MLJ: A Julia package for composable machine learning* [93]. The experience from the work on this open-source project was key for the designs of the libraries presented in Chapters 2 and 6.

1.3 The organisation

Aggreko is a manufacturing company. It was funded in 1962 with headquarters in Glasgow, Scotland. Aggreko has a presence around the world. We will refer to Aggreko as *the company* or *the organisation* in the next chapters. The company manufactures equipment for power generation such as generators, loadbanks, transformers, and equipment for energy distribution—also heating and chilling equipment, dehumidifiers, and boilers.

The equipment is available for hire. The hiring could last for a few days or could be multi-year projects that require electric power generation. The equipment is shipped almost anywhere on the planet, and multiple industries use them.

The hired equipment is remotely monitored. There are many sensors attached to the machines that are deployed to the project sites worldwide. The data collected from these sensors are the raw material for most of the projects presented in this thesis.

The starting point of the Engineering Doctorate journey was a welcoming Information Technology team that serves the company's information needs. All the projects presented in this thesis were implemented as part of the team of Advanced Analytics at Aggreko. They were related to telemetry data from the generators and equipment or customers of the company. The projects are a selection of data science and data engineering projects.

In the beginning, there was a stable on-premise data warehouse system and scheduled data transformation processes to transfer data from the operational systems to the data warehouse. The organisation was taking the first steps towards a data-driven company. Five years later, we can see the great progress that a fantastic group of people have made. It is on the way to a promising future that I feel proud to have contributed to.

[5] Official website of the Julia programming language, https://julialang.org (accessed 25 October 2021).

Part I

Foundational Projects

Chapter 2

Automated Exploratory Data Analysis

Summary

An automated data profiling tool is presented in this chapter that explores the content of databases and creates a data catalogue containing the data source's metadata. Another contribution of this work is an algorithm to find candidate primary keys from tables in $O(n^k)$ where n is the total number of columns in the table, and k is the number of columns in the primary key.

2.1 Introduction

Companies often have many operational information systems, collecting and storing data for multiple purposes. How can users make sense of all the data available at a company level? In principle, the answer is straightforward: given access to the schemata for the databases, users can formulate queries against multiple data sources: this is, after all, the original promise of relational databases. In reality, however, schemata are often nor available across the entire organisation (or indeed at all) simply because the data sources grow too fast, or change too rapidly, to be documented. In many ways this is a triumph: it is now easy to store huge data volumes, and so organisations do so – often without any idea of the purpose to which the data may later be put. We see this tendency growing very strongly alongside the "Internet of Things", with companies retaining data that they believe (or hope) that it will one day be valuable. If several groups have this idea simultaneously, it does not take long before the proliferation of undocumented databases can overwhelm the capabilities of analysts. This is a shame, as it frustrates a company's ability to extract value from its data and reduces the return on data investment.

An open-source tool called *Automated Exploratory Data Analysis (AEDA)*[1] was developed that can perform schema extract at data-science scales by finding candidate keys in sets of tables. The process is fully automated and database agnostic, based on metadata collection from a target database.

The AEDA library was developed in Python and was presented at the PyData

[1] GitHub repository of the AEDA tool, https://github.com/darenasc/aeda (accessed 18 August 2022).

London Meetup[2], PyData Edinburgh Meetup[3], and at PyData Global in 2021[4].

This tool can be useful for companies that rely on external data sources. Most of the EngD projects in this thesis used a telemetry database on MySQL designed by a third party. The organisation had access to this external database but no knowledge of its contents. AEDA proved helpful in identifying the relevant tables for different sensors and processes.

Three contributions are presented in this chapter: a standard automated process to explore databases; a brute force algorithm to find candidate keys in tables; augmented with a process for data discovery of similar information in multiple systems.

The origin of the algorithm to find candidate keys in tables was presented as a challenge by the company's Data Engineering Lead (RC) to the author of this thesis. At the time, there was an Oracle database with over 120 tables and cryptic table and column names using groups of chars without human readable meaning. The database did not have primary keys on the tables, and they were needed to implement a Change Data Capture strategy to load them into the centralised data repository of the organisation. The challenge was presented on a Friday afternoon during a social activity with the IT team of the company, and the algorithm was developed over that weekend. It was tested the following Monday with good results.

The rest of this chapter is structured as follows. We describe the commercial context for our work in section 2.2, and survey related work in section 2.3. Our design is presented in section 2.4 and evaluated against a real commercial scenario in section 2.5. Finally, section 2.6 presents our conclusions and future directions.

2.2 Background

In this chapter we will use the term *Analyst* to refer to a Data Analyst, Data Scientist or Data Engineer. The Data Analysts often are part of business units creating reports for managers using SQL language to query databases and data warehouses. The Data Scientists have background and knowledge of maths and statistics. They can model the data to create predictive models and can describe datasets using statistics. The Data Engineers create the data transformations to provide data to the organisation. They care for efficient data processes. They create the data transformation tasks to move data from the data sources of operational systems to centralised data repositories where data scientists and data analysts can access the data.

The databases used in companies on industrial environments can easily contain hundreds of tables per system. This represents a challenge to any experienced or new analyst working on an analytics problem.

[2] Video of the lightning talk: *Automated Exploratory Data Analysis on Databases* at the 58th PyData London Meetup on September 3, 2019, https://www.youtube.com/watch?v=vvvBWQLFtok (accessed 18 August 2022).

[3] Event description of talk: *PyData Edinburgh: Automated Exploratory Data Analysis of Databases* on November 4, 2021, https://opentechcalendar.co.uk/event/11493-automated-exploratory-data-analysis-of-databases, (accessed 18 August 2022).

[4] Video of the lightning talk: *Automating the Exploration of Databases for Data Science with AEDA* at PyData Global on October 30, 2021, https://www.youtube.com/watch?v=PsCKG8EZIfw&t=3314s (accessed 18 August 2022).

The ever-increasing number of data sources available makes two problems particularly difficult due the amount of queryable data. The first problem is to make sense of the data in an environment of information overload, and the second problem is searching related information regarding a specific piece of information that is of interest: the classic "searching for a needle in a haystack".

Database vendors will often offer good usage statistics about the data stored in their database engines, but will lack of information regarding other systems. This becomes a problem for a company with systems using multiple database engines. There is a lack of generalisation in the problem of providing summarised descriptive statistics about the data.

Companies launch data governance and data management initiatives to take control of the large amounts of data available and to make it available to the stakeholders involved. Data quality initiatives are also an important part of the assessment [24] in these practices.

It is necessary to know how the data in different systems relate to each other because it all belongs to the same company. And a feature that not many data tools offers is how data from external sources relates with the data within the company. Both challenges are met with the profiling process of data exploration and its byproducts.

The processes of Change Data Capture (CDC) [35] and Extract-Transform-Load (ETL) [36] are also affected by the knowledge the analyst has of the datasets. In order to ingest data from one system to another, the analysts require a level of understanding of the data.

A tool that performs reverse engineering on databases [20], [8] and generates SQL queries to extract descriptive statistics from the data in a given database is presented in this chapter. The aim of the tool is to minimise the time an analyst spend querying, *a priori* unknown, databases in order to understanding its content.

2.3 Related Work

Exploratory Data Analysis [3] is a central activity in any data related process. It is a series of steps the analyst needs to perform in order the understand the data.

The exploration of database tables, is often performed using the ANSI SQL Standard [9] or simply the SQL language. The analyst will perform a series of SQL queries to aggregate data and compute descriptive statistics.

A data professional will consume most of the time preparing the data, understanding the format, visually inspecting the variables and ranges of the numerical data [88] before performing any analysis. There is room for automation as most of the descriptive statistics tasks are repeatable [30], [3].

There is research around data discovery in big data sets [57] and finding related tables [44]. Exploratory systems has been suggested but mainly for data visualization [55], [75]. We propose a common approach for structured data using a direct queryable metadata database to find relationships among data.

2.3.1 Data Exploration in Data Mining Methodologies

The data exploration phase is common to the most popular and used data mining methodologies and frameworks such as KDD [13] CRISP-DM [15], [21], [31] and

SEMMA.

KDD stands for Knowledge Discovery in Databases. The first three phases in the KDD process are data selection, pre processing and data transformation. A requirement for for this phases is certain familiarity with the data. A common practice to gain familiarity with the data is for the analyst to start querying the database.

CRISP-DM stands for CRoss-Industry Standard Process for Data Mining. Initially developed by a group of companies as part of a European Union project, it provided guidelines to work with data mining at industrial level. The second and third of the six phases of CRISP-DM: Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, Deployment involves data exploratory work.

SEMMA stands for Sample, Explore, Modify, Model, Assess. It was developed by SAS Institute and the Explore phase is relevant because allow to modify and model the data.

Data exploration, data discovery is at the core of the analyst tasks and we present a way to automate these tasks.

2.4 Design

Can we design a tool that helps the analyst with repetitive tasks at the same time that provide insightful information about data quality and data distributions of, *a priori*, unknown data sources?

There are many potential applications of the profiling tool presented in this paper, we will present two use cases and list the rest. The first use case, is the automation of the exploratory data analysis (EDA). The second use case is a sound approach to identify candidate keys on tables when no schema or information is provided.

The processing of the queries for rapid EDA are pushed to the source RDBMS.

The processing of candidate keys searches, use an heuristic explained in Section 2.4.5 that uses local resources for initial fast computations and remote resources for validation.

2.4.1 Assumptions

Assumption 1. The server hosting the source database is capable of processing SQL queries against the source database.

2.4.2 Exploratory Data Analysis Tasks

When an analyst is exposed to a new database without any prior knowledge about its content, the analyst will query the tables and datasets in order to learn and extract knowledge for the analytics purpose.

We have defined a series tasks that any analyst would perform in an exploratory data analysis. We numbered the tasks to reference them later. The analyst would like to know:

T0 the server name or server address, table catalogue, table schema, table names, column names, ordinal position of the columns, and column types.

T1 number of tables in the database.
T2 number of rows per table.
T3 number of columns per table.
T4 number of unique values per column.
T5 number of null values per column.
T6 frequency number per data value per column.
T7 timewise aggregation of the data.
T8 univariate summary statistics for all the numeric data types.

The data collection in **T0** task enables the computation of the rest of the tasks.

T1, **T2**, and **T3** allow the analyst to estimate the size of the project in terms of the volumes of data. The results from **T2** help prioritising the processing of the rest of tasks starting with the less populated tables and, optionally, distribute the processing of the bigger tables.

The data types from **T0** trigger further explorations based on the data types. There are three main types a) discrete types, b) continuous types, and c) time types. We want to compute **T6** for a), **T8** for b) and **T7** for c). For now, open text types are out of the scope of this tool.

The results from **T4** is compared with a maximum unique values *threshold*, if the number of unique values is less than the threshold it will trigger the **T6** task for those columns.

T5 reports whether or not a column can be considered as a part of a candidate key. It is also used to assess the quality of the data in the tables.

T6 is providing the *domain order* of each column, this information helps the analyst understand the content of the database. It is also used for data discovery as the domain order can be queried against the other databases profiled. The sum of the frequencies will indicate the level of similarity between the columns.

The aggregated values from **T7** can be used to visualise trends in the data.

Finally, the univariate summary statistics from **T8** such as mean, standard deviation, variance, maximum, minimum, percentiles (0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.975, 0.99), interquartile range, range, skewness and kurtosis, will help to assess the distributions of the numerical data.

With this approach the analyst can cover different data types for discrete, numerical, and temporal data.

2.4.3 SQL code generation

Many relational database management systems (RDBMS) use an internal database called the *INFORMATION_SCHEMA* database as a way to store the metadata about the databases in the system.

The profiling tool executes **T0** leveraging on the INFORMATION_SCHEMA database whenever is available, and implements ad-hoc scripts for systems using a different method to store internal metadata such as Oracle, Delta tables, or Apache Hive tables.

The SQL queries for **T2**, **T4** and **T5** only differ on the table and column names. These queries are generated using the metadata database with the data collected in **T0**. Also, **T4** and **T5** go together in a single query as they have the same target.

Similarly, the SQL queries for **T6**, **T7**, and **T8** can be generated using the metadata of the source database.

2.4.4 Data Governance and Data Quality

The metadata database is a single repository that can be used for Data Governance. It contains relevant information such as the number of *NULL* values per column **T5**, that combined with the total number of rows per table **T2** gives a metric of the percentage of completeness of each column, highlighting any issues with the data. The calculation can be computed at table, database, server, and company level.

The level of completeness provides a good assessment of the status of the data and can be used to inform stakeholders and make decisions based on the quality of the data.

The profiling tool can advise and generate warning alarms regarding the completeness level of the source database.

2.4.5 Candidate Key Search

The implementation of a candidate key finder is a *brute force* algorithm with a series of heuristics in order to reduce the search space and to improve the performance of the searches.

To search candidate keys for a given table, the algorithm assumes no prior knowledge about the table. And it will always converge if the algorithm has enough time and resources to process. It assumes that the given table has a primary key. But even if this is not the case the algorithm has a mechanism to stop the searches if suspects that.

The complexity of the algorithm is $O(n^k)$, where n is the total number of columns, and k is the number of columns in the primary key of the table. In the best scenario, the algorithm will provide a solution in linear time $O(n)$ when $k = 1$, meaning the primary key is a single column. The exploration of the solution discards unsuitable candidate keys early in the process in linear time. Only the most likely candidate keys will be evaluated in the full table.

Limitations. When a table does not have a primary key, the algorithm has a stop mechanism, and it will search in reasonable spaces defined by the analyst. The algorithm will inform the analyst that the table is presenting an unusual behaviour for what it is expected.

The cadidate primary key search algorithm

As presented in Algorithm 1, the algorithm receives three parameters: a *table_name*, a *k_max* integer number, and a *threshold* of accuracy as a real number between 0 and 1.

The primary key could be a single column or a combination of columns in the table. The *k_max* parameter is the maximum number of columns to use to calculate the combinations without repetition using all the columns from the table. The default value is five $k\_max = 5$, this means that the algorithm will stop after testing all the combinations of five columns as candidates for the table, whether it finds candidate keys or not.

Algorithm 1 Candidate primary key search

```
Input $table\_name$, $k\_max$, $threshold$
Output log file with candidate primary keys.
$D_1 \leftarrow get\_top\_100\_rows(table\_name)$
$D_2 \leftarrow get\_top\_10000\_rows(table\_name)$
$D_3 \leftarrow get\_top\_1000000\_rows(table\_name)$
$columns \leftarrow get\_column\_names(table\_name)$
for $k = 1$ to $k\_max$ do
    $candidate\_list \leftarrow get\_combinations\_without\_repetition(columns, k)$
    for $candidate\_key$ in $candidate\_list$ do
        $temp\_result \leftarrow get\_count\_unique\_values(candidate\_key, D_1)$
        if $temp\_result/total\_rows(D_1) \geq threshold$ then
            $candidates\_1h \leftarrow candidate\_key$
        end if
    end for
    if $length(candidates\_1h) > 0$ then
        for $candidate\_key$ in $candidates\_1h$ do
            $temp\_result \leftarrow get\_count\_unique\_values(candidate\_key, D_2)$
            if $temp\_result/total\_rows(D_2) \geq threshold$ then
                $candidates\_1k \leftarrow candidate\_key$
            end if
        end for
        if $length(candidates\_1k) > 0$ then
            for $candidate\_key$ in $candidates\_1k$ do
                $temp\_result \leftarrow get\_count\_unique\_values(candidate\_key, D_3)$
                if $temp\_result/total\_rows(D_3) \geq threshold$ then
                    $final\_candidates \leftarrow candidate\_key$
                end if
            end for
            if $length(final\_candidates) > 0$ then
                for $candidate\_key$ in $final\_candidates$ do
                    $temp\_result \leftarrow get\_count\_unique\_values(candidate\_key, table\_name)$
                    if $temp\_result/total\_rows(table\_name) \geq threshold$ then
                        $candidate\_primary\_keys \leftarrow candidate\_key$
                    end if
                end for
            end if
        end if
    end if
end for
return $candidate\_primary\_keys$
```

The *threshold* is the minimum percentage of acceptable accuracy for any candidate to be reported as a candidate key. The default value is 99.999%. For example, for a table T with N^T number of rows, and a candidate c with N_c^T number of rows in the set formed by *ck*, N_c^T divided by N^T must be greater or equal to 0.99999 for the algorithm to report it as a finding or a candidate key.

The output of the algorithm is a list of one or more recommended candidate keys formed by columns of the table. It is for the analyst to decide what candidate key to choose.

The algorithm will create three sampled datasets, D_1, D_2, and D_3 with 100, 10,000 and 1,000,000 rows respectively from the table that is being processed and it will keep them in memory. It will also create a list of *usable* columns from the table. If **T5** has been computed it will use this information to discard all columns containing *NULL* values because a primary key can not have *NULL* values on it; if **T4** has been computed it will discard all columns with single values, otherwise, it will return a list of all columns in the table. The total number of usable columns is n.

As it is presented in Algorithm 1, the algorithm will start iterating from $k = 1$ to $k = k\_max$. It will compute a list of all the combinations without repetition of k columns in n. For $k = 1$ is the same as the list of all usable columns. Then per *candidate_key* in the list of *candidate_list* it will compute the number of distinct values using the *candidate_key* in the D_1 dataset, if the number of distinct rows divided by the total number of rows in the sampled dataset is greater or equal to the *threshold* of accuracy, then the candidate will be temporarily stored in a new list of candidates to be tested in the D_2 dataset.

If one or more candidates surpass the threshold using D_1, then that reduced number of candidate keys will be evaluated in D_2, if one or more candidates pass this second evaluation, they will be stored in a new list of candidates and then will be evaluated in D_3. The candidates that pass the third evaluation in D_3 will be stored in a list of *final_candidates* and will be evaluated in the full table in the database.

If a final candidate passes the evaluation using all the data in the table, then it will be reported as a candidate key by the algorithm and made available to the analyst.

If zero candidates pass the evaluation of a sampled dataset (D_1, D_2, or D_3), the algorithm will increase $k = k+1$ and will get a new list of combinations of k columns with its new value, and start to test the new candidates in D_1.

If zero candidates are found after the last iteration when $k = k_{max}$, the algorithm can stop there or decrease the *threshold* parameter by 1% and restart the evaluations. This stop is optional and the analyst can define a maximum number of iterations to decrease the *threshold*.

If accuracy of 100% is achieved during an iteration, the algorithm will stop at the end of the run, after having tested all the candidates with the same number of k columns.

Example of search candidate primary key algorithm

The steps will be followed with an imaginary example to demonstrate how the Algorithm 1 works. A table with 20 million rows and 42 columns; and that composed primary has two columns.

Inputs. The table name is needed to generate SQL code to query the source database to collect the column names and sample data and test the quality of the candidate primary keys, *table_name* = *test_table*. The second parameter is *k_max* that, for the example, will be set to 5 (five), meaning that the algorithm will stop after testing combinations without repetition of 5 columns in the 42 columns of the source table for candidates for the composed primary key. A *threshold* = 0.99999 means that the candidate composed primary keys are required to explain the uniqueness of 99.999% or more of the total number of rows in the table. This parameter represents a condition of sufficiency.

Processing. Lines 1 to 3 generate three sample datasets that will be used incrementally. Line 4 gets a list of all the column names of the table. The combination without repetition when $k = 1$ is the same list of unique columns (combinations only are computed when k is greater than one), so the list of candidate keys is equal to the list of the 42 unique columns. Then, for each of the 42 columns, the algorithm will compute the number of unique values the column can produce and compare it with the total number of records in the table. The algorithm must complete the evaluation of all the combinations before deciding to continue evaluating the combinations in larger sample datasets or to stop the exploration of k combinations and move to $k + 1$ combinations and repeat the process; in line 13, the algorithm is evaluating if there are any candidates, if it doesn't find any then it will move to $k + 1$ combinations. For this example, let's say that none of the 42 columns was sufficient to generate 100 unique records so that the algorithm moves to the next round of evaluation of combinations of columns. It will compute the combination without repetition of 2 columns using the 42 available columns, equal to 861 combinations using two columns at a time. Between lines 7 and 12, the algorithm will evaluate the 861 combinations. For the sake of this example, let's say it found 5 of the 861 combinations that could produce 100 unique rows. Now the algorithm will move the 5 potential candidates to evaluate them in a larger sample dataset of 10 thousand records. Only 5 evaluations are necessary and happen in RAM. Let's say that 3 out of 5 candidates can produce 9,990 or more unique values. The next step is evaluating the 3 potential candidates in a bigger sample data set between lines 21 and 26. If any of the 3 candidates surpasses the *threshold* it will be sent in a query to the source database to be evaluated with all the data in the table. For the example, let's say 2 of the 3 candidates generate more than 99.999% of unique records using all the data in the table.

Output. All the intermediate and final results are logged to a file available for the analyst to explore and decide by visual inspection which combination of columns will define as the primary key for that table.

2.4.6 Performance

The use of sampled datasets increases the performance of the search. Often the use of 100 rows will act as a filter to not continuing testing a candidate. As D_1 is kept in memory the evaluation of a great number of candidates can be performed in seconds rather than testing it against the full table in the database. Only the candidates that have passed the funnel process will be tested using all the data. A task like this will take hours, even for a trained analyst, testing possible combinations of columns.

The evaluations on D_1, D_2, D_3 happens in local memory, which means less usage of network or remote processing in the host database. Only a reduced number of queries will be sent to the database that should be able to respond.

The time to move the sampled data sets depends on the network velocity. Once the data is loaded into local memory the first pass takes the order of seconds and the second and third passes can take minutes for a single table. The evaluation of the *final_candidates* may take up to hours depending on the size of the table and the processing power of the host database server.

The algorithm described in in 2.4.5 can be parallelised and distributed as each run is testing different columns. The number of simultaneous evaluations is given by the number of cores in the local machine. For the number of queries sent to the database we wouldn't advice to overload the database with multiple queries, it is planned to improve the algorithm to timing the latency of the responses from the database in order to establish the right number of simultaneous queries to send to the database without overloading it.

2.4.7 Data Discovery

Discovering where and how data is stored in databases is a challenge when there is no schema to query the provenance of the data. The analyst will see a graphical user interface of an operational system showing specific data, but it is difficult to find the tables without a relational schema of the database.

The frequency metadata captured by the profiling tool provide a simple way to interact and search for similar data sets in columns from different tables and systems. **T6** provides a key to join different columns using a one-to-one join because all values will be unique.

The column's similarity feature allows to discover potential *interrelation* between tables.

2.5 Evaluation

We ran the profiling tool in a MySQL v5.7 server on a database of with 251 tables, more than 4.3K columns, and more than 4 billion rows in total that took less than 12 hours to run.

Of the 251 tables, 250 tables have a *primary key* of a single column or composed by two or more columns. 91.2% of the candidates suggested by the algorithm were the or part of a *primary key*. For tables with more than 1 million rows, the algorithm's precision was 100%, all the suggestions given by the algorithm were among or were the primary key of the table.

We ran another search in a database with 120 tables that took in total 34 hours. 93% of the tasks took less than one minute to finish, and a 0.625% of the tasks took more than 10 minutes to complete accounting for 31 hours of the 34 hours total.

For example, for one table with 61 columns and 285,956 records. The algorithm took an hour and thirty five minutes on the first pass to resolve that there were 349 candidate keys out of 521,855 combinations of potential candidate keys formed by four columns. The 349 candidates were reduced to 96 candidates, with an accuracy of 100%, in less than two minutes on the second pass. Finally, it was resolved in less than one minute that there were nine potential candidate keys for that table, and the selection was left to the analyst to decide.

A different table in the same database had 10,702,693 rows and 48 columns. It took the algorithm less than six hours for find 341 candidate key formed by five columns among the 1,712,304 possible combinations of five columns in the first pass. And then, it took only seconds to find the final ten candidate keys.

The table that took more time was a table with 56 columns and 79,092,409 records. It took 12 hours and 30 minutes to determine 71 candidates out of 3,819,816 combinations of five columns. And then, ten minutes more to determine 2 (two) potential primary keys for the table.

2.5.1 Support

The process is database agnostic and the library currently supports the database engines of: SQLite, MySQL, PostgreSQL, Oracle, MS SQL Server and Azure SQL Server, Databricks Delta Tables, and Apache Hive tables. There is no indication that it could not support other RDBMS not listed in the tests.

The MySQL, PostgreSQL, and MS SQL Server engines use the `INFORMATION_SCHEMA` database. Whereas Oracle has different data structure to pull the data from that is also supported. Databricks Delta tables and Hive tables are analysed using the `spark.sql()` context and the SQL queries are generated similarly to the rest of the database engines.

2.6 Conclusions

Exploring tables and databases is a process that can be automated due to the type of analysis of descriptive statistics. Data exploration is a time consuming task for most analysts and gains in time dedicated to analysis can benefit companies, specially when using large data sets.

We presented a process to automate the exploration of databases. An algorithm to identify candidates keys on tables. And a way to find related data across multiple systems using aggregations of the data and assuring closure using the domain values from the metadata.

This tool is useful to populate datalakes and data warehouses. Identify changing in the data and gain understanding of *all* the data companies collect.

A consulting company uses AEDA in Chile[5] to profile the databases of their clients as one of the first tasks in their projects to have rapid access to the metadata. They then provide search functionality at domain value level to their customers to the data catalogue generated with the tool and provide data quality assessment reports. The most notable instance of a database profiled is a SAP HANA instance with more than 152 thousand tables and more than 1.7 million columns.

2.6.1 Future work

This library has the potential to become a simple tool to gain insights from databases. Future work on this library includes the computation of mutual information between

[5] Stratify is the company that is currently using the AEDA library on their projects, https://stratify.cl (accessed 18 August 2022).

columns from different tables. These tables can come from various source systems but can be matched using metadata and a simple similarity calculation using the domain values and their frequencies. This information potentially will indicate that two tables might be related.

Future work for this library includes providing a user interface that makes it easy to visualise summary information from the metadata database and search functionality to explore the data using ElasticSearch for full-text indexation and a Grafana dashboard to visualise the collected metadata and provide data quality reports off the shelf after running the exploration.

Chapter 3

A Streaming Analytics Architecture

Summary

This chapter describes the design and implementation of an experimental data architecture to process data on the move while ensuring the collection of data and delivery to the architecture components that will process it. It contains all the elements of modern data architecture for fast, reliant, and efficient data processing.

3.1 Introduction

With the introduction of the Apache Hadoop project based on the design of two papers published by Google about their Google, File System [25], and the MapReduce algorithm [34] in 2003 and 2004, respectively. For the first time, companies could process enormous volumes of data on standard machines. Creating clusters of commodity machines made it possible to process massive volumes of data for companies without supercomputing or oversized hardware. It started as a cost-effective way to process enormous volumes of data in batches but is moving towards a real-time analytics approach adding value.

The launching of Apache Hadoop as an open-source project in 2006 allowed companies to start processing data they had been collecting for years. Apache Hadoop is a distributed system that abstracts the complexity of distributing data storage in shards using different machines within a cluster. Shards are parts of data files. A file can be composed of a series of shards, and shards can be replicated, saving a copy of it in the storage available of multiple machines. The distribution of the storage allows processing the files locally using the *MapReduce* framework. The principle and advantage behind Hadoop is local access to the data. Data is processed where it is stored, reducing network traffic and potential bottlenecks produced by moving data around the network. By default, Hadoop replicates the shards of each file three times.

Big Data Technologies allow companies to process more data than ever before, helping organisations identify business opportunities and optimise business operations.

We can see a shift around the type of data processed at organisations from using almost strictly structured data to the use of semi-structured and unstructured data. Nowadays, new data sources are added regularly and in different data formats. Cheaper storage is one of the reasons why companies are collecting and wanting to use more data than in the past. Better throughput in network bandwidth impacts the development of new technologies and frameworks (software) that better use the available hardware, commodity hardware.

This new stage in the era of big data has affected how companies process and analyse data. *Streaming analytics* is now an important aspect of the enterprise data architecture. Moving from a *batch processing* approach to a *streaming* one, where data is processed as soon it is produced.

This chapter presents the results of a minimum working experiment of a *streaming analytics architecture*. The data architecture is deployed as a set of *micro-services*. Micro-services allows connection and disconnection of the *data services*, minimising the impact and performance to other components of the architecture.

At the time of the design and implementation of this project, there were no plans of having a data lake or central repository.

The suggested architecture has a *central transport layer*, a *persistent data layer*, a *monitoring system*, and an *exploratory & modelling component*. This experiment intends to find a sound data architecture capable of processing and delivering value from telemetry data an organisation is collecting from devices worldwide. The telemetry data for this experiment has been simulated.

The problem statement of the experiment is presented in Section 3.2. In Section 3.3 several concepts and terms are presented to go through this chapter. The solution design is described in Section 3.4. Data requirements are in Subsection 3.4.1. Results of the experiment are in Section 3.5. The evaluation of the results are in Section 3.6.

The data science architecture with streaming analytics was presented as a generic solution for an organisation at the Edinburgh Docker Meetup in the January event of 2017[1].

3.2 Problem Statement

The telemetry data for this project is coming from assets or machines deployed on project sites in different locations around the world. These machines provide temporary electric power, heating, and cooling solutions to customers from multiple industries and sectors such as mining, oil & gas, construction, and many others.

The sensors are attached to the machines measuring a wide range of signals and levels necessary to monitor the devices' workload properly. The data challenge is collecting, storing, processing, and analysing the data produced by the sensors. We

[1] Edinburgh Docker Meetup event page, https://www.meetup.com/Docker-Edinburgh/events/kxsmtlywcbzb/ (accessed 09 October 2021).

need to create analytical capabilities in the organisation with a sound *data architecture* supported by a sound *technological infrastructure* to support the information needs of the business. The question to be answered, to handle and process *all* the data adding value to the company, is:

How can we design a technological infrastructure to support the collection and processing of the ever-increasing telemetry and operational data?

One of the main goals of the IT Team of the organisation is to **enable advanced analytics capabilities** to implement machine learning models for preventive maintenance, scoring of business proposals, cross-selling to current customers, up-selling of solutions based on the history of transactions, etc. This goal is considered for this proposal of a data architecture implementation.

3.3 Background

Data-driven organisations prepare their growth in data maturity on multiple fronts. Getting ready is a three-fold effort. It is about people, technology and culture. Within the culture, we can find the processes that are used by the teams.

People refer to the collaborators of the organisation and their development and contributing skills to the team. It defines the technical skills of the data professionals that work with data. Deploying systems in production environments is very relevant. The *last mile* matters when it comes to the delivery of solutions.

The organisation has an on-premise data warehousing platform. It is used for reporting to business users. As is often the case with data warehousing technologies, it is very strict with data format from the inputs. The system grants little flexibility on what data should be stored, and any changes will take time before the load and transformation processes of the data sources are modified. The data architecture that we will suggest in this project will work with the data warehouse. Still, we will also recommend the potential incorporation of a Data Lake to the technology stack of the organisation.

We will define a few concepts before describing some of the design criteria used for the data architecture. We will introduce a series of concepts such as data lake, microservice architecture, and enterprise architectures such as lambda architecture and kappa architecture.

We will use the concepts of abstraction and encapsulation to design the data architecture.

Microservice architecture is based on the idea that each system functionality should be implemented independently and deployed as a service. Each service interacts and communicates with the other services. Each service always performs the same task that is self-contained with all the dependencies needed to execute it.

3.4 Design

Designing an enterprise *data architecture* is a non-trivial task which needs to consider many different factors.

A constraint for this EngD project was the use of Microsoft Azure as the cloud provider used by the organisation. The design will adapt to this constraint.

The first factor is the purpose of the solution, the end goal of having a new tech stack implemented, and the consequences of not having it. Understanding how a new architecture will change processes by making them more efficient or disrupting the teams' working methods. At the same time, it is necessary to understand the volumes of data required to be handled by the data architecture.

We also need to think about the performance of the data architecture. We need to consider that the system should be scalable and handle the current and future workload.

Additionally, there are some factors that we need to consider, such as the current technology stack and the integration with existing technologies. These two factors can help us recommend the best approach suitable to the organisation's needs and capabilities.

We can reduce the data architecture options by asking some questions. The answers to these questions will condition the responses of follow up questions. What questions should we ask before starting a project like this?

The first question is *why* we are starting a project like this? *What* is it that we want to achieve? This helps to keep the end always in plain sight to make crucial design choices along the way. We started from the base of having a stable data warehousing platform that has been in evolving development for the last 15 years. Batch processing is a well known way to process data for the organisation, but at the time of the implementation of this project, more and more generators and other machines were fitted with sensors it was expected to have more incoming telemetry data than in the previous years. There was also a need for more *real-time* analysis such as the one for predictive alarms based on recent data. Batch processing takes some time between when the data is produced until it is consumed by a model to generate some results. This challenge in velocity can be solved with a streaming processing architecture that is able to process the data as it is received in a landing area or platform.

We can continue with the following questions: are there efficiencies in costs, efforts, people, technology? How does this integrate with the current technological stack? Will this system require more or fewer people to be maintained? Will this solution enable other projects within the organisation? If not, how can we adapt and support the current and future efforts of a data-driven organisation? Does the new architecture require new skills and capabilities from the tech team? If so, what are the current gaps? It is essential to prepare the change to bring new technologies to the organisation and avoid underutilising new technologies and resources. Is this technology the right fit for the demand we have or that we will have?

A streaming architecture would extend the current batch processing strategy of the organisation. Incorporating streaming processing would require implementing and learning a new type of platform designed to handle queues of messages instead of tables and bounded data sets. The data in a streaming architecture is unbounded, and that is one of the advantages of using it, the fact that it can handle the incoming data with minimal delays in the data workflows.

After considering the factors mentioned above, we suggested a *streaming analytics* data architecture to process incoming data in real-time or near real-time, with a batch processing layer. This architecture is known as the *Lambda Architecture*[2]. This type of architecture has been proposed in [63] for real-time predictive maintenance, which is one of the goals of the organisation. The other option was to implement a more efficient batch processing workflow to process more incoming data but this is not what we are seeing when the demand for processing real-time data increases, we see a transition towards streaming platforms that can handle the data analysis and processing of a continuous flux of streams of data.

We recommend to separate the *computing* from the *storage* [76] of the data. Also, the decision of building something *on-premise* or *in the cloud* should be addressed.

Building an analytic architecture on-premise means that the machines have to be sized in advance, acquired, and maintained by an on-premise team. This also means that we have a fixed amount of resources allocated to the project. To perform a correct sizing process, the analyst must know the computing demand of the system in advance. The result can be a group of powerful but underutilised machines.

Deploying the system to the cloud means that the cloud provider performs the maintenance of the machines. The system then is highly scalable, allowing to add or remove resources at will. The time to the first test is shorter because the team can start using the available resources sooner than with an on-premise approach. The use of the cloud is often a pay-for-use model, and this approach will be cheaper than acquiring and maintaining all the machines before starting coding.

If the system's analytic demand and use cases are unknown at the beginning of the project, we recommend implementing the system in the cloud. Once the system requirements have been sized and tested, an on-premise deployment could be considered based on several factors like cost, team familiarity with the technology, and time available for a transition.

The proposed architecture is designed as a set of independent services, *microservices*, providing batch and real-time analysis in a *scalable* and *flexible* way. It is scalable because the platforms can balance the load and increase their processing power on demand. It is flexible because there are alternatives for each service or component of the architecture. A microservice architecture has several advantages for data science environments.

A data service can be defined as an independent system that interacts with other data services to provide specific functionality to the ecosystem that other services are not providing. A service can be composed of other services. For example, a Hadoop cluster providing a data lake service can be formed by independent machines providing processing power to the cluster.

We suggest a *streaming processing* approach because it would allow us to ingest and process telemetry data as soon as it is produced. With an ecosystem of *data*

[2] A repository dedicated to the Lambda Architecture, http://lambda-architecture.net (accessed 10 November 2021).

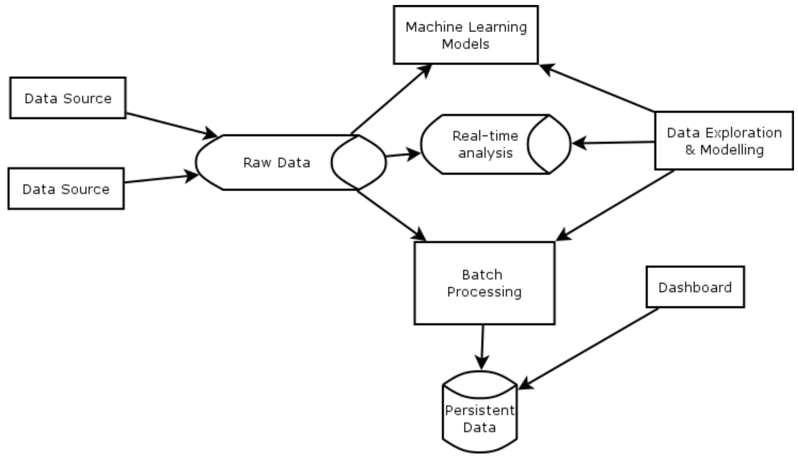


Figure 3.1: Components of a streaming and batch processing architecture.

services for several reasons: it would facilitate rapid prototyping connecting components to the architecture as they are needed; it will provide isolation among the components; it offers flexibility to upgrade, change and modify the data platforms in use, with minimum impact to the rest of the services in the system; it provides a clear division of the processing from the storage which helps users to access the data and use it for different purposes.

The designed solution serves two other critical criteria considered best practices in the industry: 1) developing a unique data repository for data users and 2) moving the company towards a self-service data culture.

The reasons mentioned above apply to creating a data science environment for the organisation.

The experiment is a minimal realistic scenario where streaming analytics is applied. Figure 3.1 presents the components of the architecture. Raw data is ingested from a streaming platform used as a central transport layer. The platform stores and classifies the incoming data and delivers the data to the other components of the architecture. The other components in the experiment are: a set of machine learning models deployed as services allowing to query them or consuming them via REST APIs sending a data tuple and the model will reply with the score or the results processed in an isolated machine or hardware; we suggest a batch processing platform to analyse and monitor big volumes of data. This will be a version of a data lake. There is a persistent layer to store metadata and significant datasets. And a component for data scientists to query, exploring and model the data.

We will simulate the telemetry data for the experiment. This will allow to deploy and test the experiment in environments in the cloud like Microsoft Azure[3] or other Infrastructure as a Service (IaaS), Platform as a Service (PaaS), or Software as a Service (SaaS) providers and not exposing the actual data.

The objectives of the experiment are:

- Test the fitness of a streaming data analytics architecture for the company's

[3] Official website of Microsoft Azure, https://portal.azure.com (accessed 10 October 2021).

data.

- Test scalability of the system and estimate the data volume demand.
- Test a data architecture where the data is processed/consumed as collected/produced.
- Evaluate costs and technologies to implement this type of architecture.

The milestones of the experiment are:

- Simulate telemetry data to use it for testing the solution.
- Deploy a streaming analytics platform.
- Deploy a batch processing platform.
- Deploy machine learning models as services.
- Deploy a real-time analytics platform.
- Deploy a data visualisation tool or dashboard.

3.4.1 The data

The data architecture should handle data from at least 3,000 devices per day and scale up until around 20,000 devices per day. The baseline of the experiment will simulate sending data every 30 minutes, and we will estimate the data demand for the next two years.

3.4.2 Architecture set up

We used Microsoft Azure resources to deploy the working experiment. It is implemented as IaaS because it provides flexibility and control to deploy the platforms required for the experiment. IaaS is also cheaper than PaaS and SaaS for testing and non-production environments.

The storage, processing power, and bandwidth are provided as IaaS by Microsoft Azure. The experiment is running in *containers* using the Docker platform. The Docker platform is running in virtual machines (VMs) with Ubuntu Server 17.04.

Containers. It is a technology that uses the Linux Capabilities[4] isolating the processes running in Linux. Each container has access to CPU and memory, I/O resources, file systems, and network resources. Containers interact directly with the Linux kernel providing fast access to the machine resources. Containers are one order of magnitude faster than VMs, so they are recommendable for rapid prototyping.

Docker is a company that encapsulated the interaction with some of the Linux capabilities providing an API to work with containers.

In terms of billing costs, it is expected that IaaS will be cheaper than a PaaS and SaaS. This report is not going deeper on this matter. Still, it can be mentioned that, for example, using IaaS, one can install any open source big data platform like Apache Kafka, paying £105 per month for a VM. In contrast, a similar PaaS

[4] Linux capabilities manual page, http://man7.org/linux/man-pages/man7/capabilities.7.html (accessed 10 October 2021).

technology like Event Hub by Microsoft will cost USD 733.13 per day[5]. Of course, there are differences between the resource configuration of both implementations. The resources of the VMs are managed by an end-user, whereas the cloud provider manages the PaaS resources. Both are managing a fixed amount of available resources.

This project aims to determine the need for a big data infrastructure for the organisation. We should avoid the risks of the scenarios of over and under-provisioning resources. It is necessary to find the right amount of resources to provision at a reasonable price. How much is a reasonable price should be determined by evaluating the alternatives available.

3.4.3 Components of the Architecture

The tools selected for the experiment are:

- Apache Kafka 2.11
- Python 3.x
- Jupyter Notebook
- Hadoop 2.7.2
- Pentaho 7.1
- PostgreSQL 9.6

Transport Data Layer

There are many options in streaming processing platforms. Among the candidates for this component of the architecture we can find *Spark Streaming* [46], the Spark library for streaming processing, process data in *mini-batches*. *Apache Storm* [48], developed at Twitter like its successor *Apache Heron* [51]. The Heron platform is at *incubation*[6] phase at the Apache Foundation but already has shown better performance than Storm. *Apache Kafka* [41] and *Apache Flink* [50] implement stream processing as a *record-at-a-time* approach, instead of mini-batch. Apache Flink is also an incubating project with a promising future.

The platforms in Azure providing streaming processing are the ones for Internet of Things like *Event Hub* and *Stream Analytics*[7].

We choose Apache Kafka[8] as the transport data layer platform because it can handle high volumes of real-time data feeds with high throughput. It can deal with periodic data loads from online and offline systems, and it offers a partitioned, distributed, real-time processing environment. Also, it guarantees fault tolerance in the presence of machine failures. Kafka reports performances of handling 20.000 messages per second [41] and our expectation is a much less threshold of 11.11

[5] Azure Event-hub pricing, https://azure.microsoft.com/en-us/pricing/details/event-hubs/ (accessed 12 January 2017).

[6] Official documentation website of Heron, http://incubator.apache.org/projects/heron.html, (accessed 10 October 2021).

[7] Stream Analytics pricing page, https://azure.microsoft.com/en-us/pricing/details/stream-analytics/ (accessed 10 October 2021).

[8] Official webpage of Apache Kafka, http://kafka.apache.org/ (accessed 10 October 2021).

messages per second simulating a fleet of 20,000 machines sending data every 30 seconds.

Kafka is a mature and stable technology widely used by tech companies like Netflix, Spotify, Twitter, PayPal, Uber, Airbnb, and LinkedIn[9] among others.

We will use the *kafka-python*[10] library to connect services to the Kafka server and vice-versa.

In Kafka, the replication factor controls how many times the data will be copied in other servers. The partition count impacts the maximum level of parallelism to consume the data.

Persistent Data Layer

We selected the *Postgres* [7] database because of its good performance and its open-source database and does not require licence fees. Is one of the most widely used databases[11] in the market. It is simple and easy to use. A close alternative is the MySQL[12] database.

Postgres has an official Docker image[13] at the Docker Hub, which makes it suitable for the experiment using containers.

We will use the *psycopg2*[14] Python library in client programs to collect and send data to the database. We can deploy the persistent layer of the experiment with a single line of code using Docker.

Data Exploration & Modeling layer

The *Jupyter Notebook* with the base installation[15] container is used for the experiment. Data exploration can be performed using containers with notebooks created on demand by data scientists and removed after use. The base installation has Python 2.7 and 3.6 kernels by default, but the analyst can install multiple kernels to work with Spark or R, for example.

A Jupyter notebook can be created with a single line of code. The analyst will then go to the URL that appears in the terminal and work using its web browser to process the data on the server side.

Batch Processing

Most of the companies use Hadoop as the preferred storage system for their data-lakes. That is why the experiment considers a small Hadoop cluster that can scale-out in datanodes. The replication factor will be three.

[9] Official website of Kafka users, https://kafka.apache.org/powered-by (accessed 12 January 2021).

[10] Github repository of the kafka-python library, https://github.com/dpkp/kafka-python (accessed 10 October 2021).

[11] https://db-engines.com/en/ranking

[12] https://www.mysql.com/

[13] Official Docker hub page of the PostgreSQL database, https://hub.docker.com/_/postgres/ (accessed 10 October 2021).

[14] http://initd.org/psycopg/

[15] Github repository of Docker Jupyter Notebooks, https://github.com/jupyter/docker-stacks/tree/master/base-notebook (accessed 10 October 2021).

The Hadoop cluster is deployed in containers. The cluster, by default, uses three containers, one container as the master node and two containers as data nodes. The cluster can be re-sized at will, scaling in or out. Scaling in takes more time than scaling out because of the re-balancing of the data stored.

Monitoring

The tool *kafka-manager*[16], was developed by Yahoo! engineers. It is used to monitor the performance of the Kafka server and its brokers. It allows checking the load, the number of messages sent to Kafka and throughput.

To visualise the data from the database, we have two options: 1) *Pentaho* Business Intelligence Server Community Edition[17], and 2) *Microsoft Power BI* is also connected to the Postgres database.

Machine Learning Models Deployment

There are usually three ways how companies deploy machine learning models in production. 1) Deploying the ML model in a lightweight web server as a REST API service, where it can receive GET messages and reply to the *score* of the model calculated on the server side. 2) Deploying it as a PMML (Predictive Model Markup Language), an XML standard to exchange and deploy ML models, so the model can be trained with one tool and deployed with a different one. And 3) using the deployment tools provided by the framework where the model was trained.

Inference is the process of predicting a value using a machine learning model. The inference task often occurs as an offline process, but this should be an online task for a streaming analytics architecture.

There are multiple options to deploy a machine learning model as a service. A simple approach is to deploy it using a lightweight webserver. The webserver will receive GET calls from clients, and it will return a JSON file with the score.

The machine learning model can be stored in XML in the web server using the PMML standard.

The deployment of machine learning models is further explored in Chapter 4 about Machine Learning Governance.

Telemetry Data

The telemetry data was simulated using Python scripts running in containers. Each container represented an asset or group of assets sending telemetry data. The analyst can create any number of assets with a bash script that receives the number of assets to be created. Each container simulating assets is automatically deleted after a successful run.

Two parameters are required to simulate an asset: `delay`, which is the number of seconds between samples and `limit`, which is the number of data samples to be

[16] Github repository of the Kafka Manager tool developed by Yahoo! engineers, https://github.com/yahoo/CMAK (accessed 10 October 2021).

[17] http://www.pentaho.com/

sent by the asset. The simulated assets send data every 30 minutes reporting variables like voltage, amperage, temperature, pressure, frequency, fuel level, battery, hours of continuous power, and RPM. The `delay` parameter allows to speed up the experiments instead of waiting 30 minutes to receive a data point.

For example, to simulate two weeks of data, the `limit` parameter should be equal to 672. If the `delay` parameter is 5 seconds, then the results will be available after 56 minutes.

The values of the variables of the assets are initialised using a Gaussian distribution. The analyst can modify this in the future to take any desired shape and distribution. An analyst could add random failures of assets to the simulation.

3.4.4 Connecting the architecture

As can be seen in Figure 3.1, Python scripts simulate telemetry data that are constantly pushed to the Kafka server. The Kafka server works as a transport layer, keeping the data for some time, seven days by default. Kafka delivers the data to the consumer components like the Hadoop cluster and the Postgres database.

Data exploration and modelling can be performed using notebooks accessing a database, file system, or Hadoop cluster.

The system is monitored using Pentaho connected to Postgres. Machine learning models are pushed to a lightweight webserver.

3.5 Results

The experiment runs in virtual machines (VMs) with Ubuntu Server 17.04 on Microsoft Azure. We used Docker version 1.13.0 installed on the VMs. One VM called `dockerdev` with four cores, 28GB of RAM, and 48GB of SSD storage, the other one called `docker-server` was half the size of the first one.

We performed two types of tests. 1) How the system supports the amount of incoming data flow, and 2) How the system supports the volume of stored data. These are two different problems concerning the design.

We tested several configurations to evaluate the performance of the system. We set the baseline as the current working conditions at the company by July 2017.

The system was tested using 100, 500, 1,500, 5,000, 10,000, and 20,000 simulated assets. For one day, one week, one month and one year of data.

When the simulation reached over 700 containers started to struggle during the first attempts, the host machine with 14GB of RAM, so the size had to be extended to 28GB.

The next problem was the number of devices in the network with a maximum of 1,024. The solution was to simulate a small fleet per container instead to use a single container per asset. To have a single container per asset is still possible when the number of assets to simulate is less than 1,024.

A simulation of 1 day of data from 1,500 devices, representing the current load of the telemetry systems, took 4 minutes using a delay parameter of 5 seconds.

Simulating 60 days of 100 assets, were 2,880 data points of telemetry data per asset. The database was 56MB and had 288.000 data points after the simulation.

Simulating **20,000 assets** required 500 containers with a pool of 40 simulated assets in each container.

One day of data from 20,000 assets sending data every 30 minutes produce 960,000 messages. The Kafka server and the Postgres database received all the messages successfully. The Kafka server had three brokers and 15 partitions for the topic. The size of the database was 188MB for one day of data.

3.6 Evaluation

A microservices architecture allows isolating storage from processing. This facilitates scaling the system out.

With this kind of load on a single machine Kafka server, a single node can handle the amount of data produced in one year.

The separation of architecture components in data services allows scaling out independently. If we were required to scale out the database by adding a new node to the cluster, this would not impact the other data services or even upgrade the database. Apache Kafka was central to the architecture, and it can scale out depending on the demand. More machines can be connected to the server, having zero impact on the other services.

The database can scale using shards and partitioning the data geographically.

Chapter 4

Machine Learning models governance

Summary

This chapter presents the assessment of platforms and technologies used for machine learning model management.

4.1 Introduction

With the increasing adoption of Machine Learning (ML) by data-driven organisations, the challenge is how to organise and manage the machine learning models developed by the data science teams.

With the availability of big data sets in organisations, it makes sense to start creating machine learning models that use the installed data infrastructure.

The administration of machine learning models and the processes related to them is known as machine learning governance.

A machine learning model management tool has to provide support for the phases of *training* and *serving* of the models. *Training* a machine learning model is the process of fitting data to a machine learning algorithm to determine the parameters that the model will use. We will understand a *machine learning model* or simply *model* as a trained algorithm that has estimated the parameters used by the method. *Serving* a machine learning model is the process of *scoring* the new data, feeding the data to the trained model to get an estimation from the trained model.

There has been an increasing demand over time to train more machine learning models in the by the data science team of the organisation. It is for this reason that this project suggests a way to get ahead of the demand and plan to govern the current and future data products related to machine learning models.

In this chapter, it is described the methodology used to evaluate some of the software used for ML governance and, additionally, this chapter includes recommendations to incorporate good practices from DevOps and adding explainability to the machine learning model processes for the benefit of the users of the ML models.

4.1.1 Benefits of ML Governance

Incorporating ML governance is recommendable to improve the communication between the data science and data engineering teams. Often, it is the data science responsibility to train and create the machine learning models and then once the mode has acceptable performance, they are handed to the data engineering team to deploy them or, it is reported sometimes, to rewrite the models in a more efficient platform or language and deploy the improved version of the model.

Monitoring multiple machine learning models in a production environment can be a challenging task. Keeping track of the models in production manually requires writing complex tailored code to maintain information about the models available to the data scientists. The complexity of maintaining multiple machine learning models over time increases with the number of models an organisation has.

Promoting machine learning models to the production environment means that once a machine learning model is trained, it needs to be deployed to be consumed. Deploying a machine learning model is often a manual task performed by a data scientist or a data engineer. Once the model is deployed, there may exist other machine learning models that are candidates to replace the one in production, but that at the time of training, they under performed compared with the model promoted to the production environment. The candidate models could continue scoring new data, and their performance can be compared against the production model. If a model candidate shows better performance than the production model, it could replace it. Promoting a machine learning model seems a trivial task, but it could involve many hours of effort. Evaluating and comparing the models, then replacing and versioning the newly promoted machine learning model, and the implementation of the process will be different depending on the type of model and language used for the model.

With an ML governance tool, it becomes transparent all the changes and versions of the models used in the production environment and independent of the number of models in use.

We reviewed a set of tools and platforms during this project to recommend a sound and usable solution to the Data Science and Data Engineering teams for ML model management. A DevOps approach was used and documented in a git repository. As byproducts of this project, a set of guidelines to work in ML projects and explainable ML were documented. In Sections 4.4 and 4.5 we present the literature review and technology review respectively.

4.1.2 Contributions

The contributions of this project are:

1. A recommendation of tools for machine learning models governance. From tracking the training and deployment of models.

2. Guidelines for coding and development and deployment of machine learning models with DevOps approach.

3. Adding explainability to the training phase of the machine learning models.

4.1.3 Structure of the project

This project was developed in three stages using Azure DevOps Projects and its git repository feature to make it reproducible and repeatable. The code, configuration files, and documentation are part of a git repository created for this project.

The first stage consisted of researching material about model management and how the industry is deploying machine learning models. A list of platforms was collected from publications and presentations at data science conferences. At the same time, a survey was sent to the team members of the data science team to include their perceptions and recommendations into this project. The questions of the survey are shown in Appendix A. The outputs from the first stage were the literature review presented in Section 4.4 of this chapter. A list of potential tools was selected to be reviewed for model management in the following stages. The platforms were sorted by potential usefulness to the data science team, so the searching for a tool could stop when a platform or set of platforms would fulfil the initial requirements. We created an evaluation criteria for the tools based on the literature available with elements from the survey to the team.

The second stage involved preparing the infrastructure and code examples to use in the experiments to test the machine learning model management platforms. It was planned the use of Terraform[1] to deploy virtual machines in the cloud provider from the source code of the repository. The set-up included an Apache Kafka[2] cluster for the staging area[3] to send messages to published APIs of the machine learning models created as part of this project, the scored results will also be stored in the Kafka cluster. Kafka made it simple to stress test the ML model governance platforms. This part was not used because the evaluation process focused on the features of the platforms rather than on response times or stress tests.

The third and final stage was the evaluation of the selected platforms based on the evaluation criteria generated in the first stage. The outputs from this stage were the recommendation of the *MLFlow* platform for internal use and a workflow guideline for machine learning model development and deployment.

The project team met every two weeks, and we maintained communication with the senior analysts of the data science team to keep them informed about the progress on this project. We prepared a tutorial of the tools in the middle of the project to demonstrate the functionalities of some of the tools reviewed.

4.1.4 Terms, Definitions, and Acronyms

Library or platform. A *library* can be defined as a set of functionalities wrapped up in a single package (the terms library and package are exchangeable); libraries abstract code complexity and provide a simpler way of doing specific tasks. A *platform* is often a set of libraries together proving an End-to-End solution, but sometimes a single library is complete enough to be called a platform.

Flexibility is a crucial concept used to evaluate the platforms. It refers to the ability to connect to and from other platforms. It means the opposite to have hard constraints of languages, or technologies, or locked-in technologies that in general

[1] Official Terraform website, https://www.terraform.io (accessed 14 October 2021).

[2] Official Apache Kafka website, http://kafka.apache.org (accessed 14 October 2021).

[3] Staging area is temporal storage used as a buffer to receive the incoming messages and then deliver them to other systems.

should be avoided due to the risks of depending on specific providers.

The general recommendation will be to use *MLFlow* for the training phase of the models and to use Azure Machine Learning (AML) to deploy them when this is required.

In this chapter, the problem statement is described in Section 4.2. Section 4.3 covers the background about the technologies and ideas supporting this project. Literature review is in Section 4.4. Section 4.5 will review the selected technologies involved in the project. Evaluation is in Section 4.6.

4.2 Problem Statement

The administration of machine learning models is not a solved problem. There are no mature or turnkey solutions. Often, big data technology companies, such as Google, Facebook, Airbnb, Uber, etc., build their platforms for end-to-end machine learning processes.

The increasing number of machine learning models in production environments can be challenging to maintain without the right policies and governance procedures. An ML model can be expensive to maintain due to the effort and resources required to do so. An ML model may deviate from the expected behaviour, something called *concept drift*, [28] and that could be unnoticed without monitoring it.

ML models are not isolated projects. Often, an initial inference experiment evolves into an ML model deployed in a production environment. This official model would need to be monitored. In some cases, the analyst will evaluate the model against new data after deployment, feeding back the evaluation of the models. At the same time, the analysts can train other models on the same data. These other models are known as candidate models that could eventually replace the official model based on their performance. If there are three candidate models, the analyst would like to track the performance of these candidates and compare them against the official one. If a candidate model becomes the official one, the recently replaced model could be observed for some time in case it surpasses the performance of the new official model. Eventually, adding new data sources and using feature engineering in the modelling will impact the deployed models. The data science and data engineering teams aim to track the changes in the models, versioning them and allowing easy substitution of the official model in the production environment. All this work applies to handling just one model. One machine learning model solves a specific business need. Tracking five or more versions deployed for one business process could become a maintenance problem considering the effort and time available from a small analytics team. Some companies have reported having dozens or hundreds of ML models in production. How to handle this problem is the aim of this project.

The question is, **how can we successfully deploy, versioning, evaluate, and monitor the performance of the machine learning models deployed in production environments to help the business in an efficient and cost-saving manner?** We want to ensure that the number of ML models deployed is not a bottleneck. At the same time, the proposed solution reduces and mitigates the *technical debt* [52] or the future cost of additional work due to poor design decisions made early on projects.

4.3 Background

One of the main challenges in data-driven organisations is maintaining control over the organisation's data sets and databases[4]. The solution to this issue is to implement policies of *data governance* [38], [37] to control and provide information to stakeholders and analysts regarding the data sources available within an organisation. Data governance involves a series of best practices of handling and collecting information about data sets in the form of metadata such as *data quality management* and *data provenance* [22] among others. *Data quality management* maintains information about the quality of the data sets and makes it available to analysts and stakeholders to decide, for example, whether or not to use specific data for a given task or implement policies to improve the quality of the data. *Data provenance* [22] or data lineage documents the origin and the transformations applied to a piece of data so that traceability can be visible to the users and analysts.

Similar issues to data governance are what data science teams face when they start developing machine learning models. Governance over the ML models is a new problem for data science teams. For example, maintaining an inventory of the machine learning models can be difficult if there are no standardised practices and repeatable processes. ML model governance provides visibility and consistency to training ML models. It grants access to stakeholders and analysts to the models. Visibility adds a layer of transparency, increasing the confidence in the models. Having a repeatable process to train and deploy ML models allows analysts to improve the models iterating on the development without the cost of manual maintenance of the code.

We can recognise three main phases when data scientists develop a machine learning model: 1) data preparation, 2) training and evaluation, and 3) deployment. All three stages are essential for the success of a machine learning project.

Data preparation ensures that we are working with the correct data. The *data types* becomes relevant because they define what kinds of ML techniques can be used on a particular data set. Data transformation and feature engineering are included at this stage.

In the *training and evaluation* phase, we are interested in tracking the metrics and parameters of the trained models. This phase includes a series of tasks such as data modelling, algorithm selection, model training, model evaluation, and model selection, all of which are based on the selected metrics to measure performance.

The *deployment* phase is how we will serve the model. There are mainly two ways of serving models in production: 1) in batch processing, applying the model at once on the new data; or 2) deploying the model as a web service and making it available for real-time scoring, often through a RESTful API.

For decades, the deployment of machine learning models and the serving part of those models received less attention than the development or training part. More

[4] A technology constraint for this EngD project was the use of a specific cloud platform to access and train ML models in a given cloud provider. This selection also determines the default relational database engines used by the organisation.

recently, there has been a surging interest in model management and consuming and making these models available to users. The training of machine learning models represents a small percentage of the effort [52] to deliver a machine learning model in an organisation. Most of the time is consumed in other tasks related to the validation and deployment of the models.

4.3.1 Challenges in ML model management

Developing machine learning is not only about training and deployment. There are more challenges associated with machine learning management. Some of the challenges in managing machine learning models are:

- Data science teams often struggle to cope with reproducibility, fairness, model evaluation, and keeping the domain knowledge when a teammate leaves the company.
- The number of ML models can be hard to track without a central repository or a model registry.
- Scalability and responsiveness of the models can be a problem when demand increases.
- The diversity of software tools that data scientists use daily can present some challenges when and handing over the models due to the different backgrounds and expertise of the analysts involved.

The nature of ML is different from Software Development (SD), but the adoption of some of the SD techniques is a good fit for best practices in ML development. We are talking about code versioning, Continuous Development & Continuous Integration (CD/CI), and DevOps. ML development is a dynamic process with training and retraining models with an interactive evaluation of results where management should be flexible and responsive. We explore the emerging culture of DataOps briefly in Section 4.5.7.

At the organisation, the current way of deploying ML models is using scheduled runs of notebooks. This process scores the new data, but it does not track performance metrics, model parameters, or the model version used for the scoring. The notebook is scheduled to run and scoring new data. This is a sound solution for inference, but it might become complex to handle it manually with many ML models in the production environment.

4.4 Literature Review

Some of the challenges of serving ML models are pointed out in [82], [73], development challenges such as how to validate the models, when is the right time to retrain the models; data management challenges such as the lack of declarative language for ML pipelines, querying model metadata; or engineering challenges such using more than one programming language to build the models and pipelines, heterogeneous skill level from users, how to ensure backward compatibility of trained models.

In addition to the traditional offline scoring using batch analytics and streaming real-time serving of models, new architectures are proposed to serve ML models in

[73], [62], and new tools addressing the serving challenges regarding deep learning models or serving reinforcement learning models [80], [70].

The problem of ML models in production is a new concern for the research community. It requires a multidisciplinary approach, *SysML* [89] a new research conference was created to address these and other problems at the intersection of Systems and Machine Learning research.

Big tech companies often develop their tools such as Facebook's FBLearner [56], Uber's Michelangelo [69], or Airbnb's Bighead[5]. These examples are closed source, but the features are similar for any end-to-end solution: flexibility, experiment tracking, reproducibility, and model deployment.

Clipper [65] was one of the first open-source prediction serving systems launched in 2016. Clipper[6] is a distributed serving system that uses Docker containers and a Redis database for persistent storage. Models are registered and deployed in isolated containers. The models are queried using a single point of interaction via an API call. Clipper can orchestrate containers using Kubernetes[7]. Clipper can track the data that has been scored and also combine predictions from different models. Clipper can deploy Python and Spark models.

Tensorflow Extended (TFX) [64] is a general-purpose ML platform. It supports many different data tasks of ML, not only the serving part. TFX is an open-source platform developed by Google. It was made for continuous training and serving, with built-in tools for model validation and data validation to ensure reliability and scalability of the models. TFX makes use of other Google tools such as Apache Beam[8] to schedule and execute processing, and needs Apache Airflow[9] or Kubernetes to deploy ML models or any custom workflow tool for serving purposes. TFX provides a high-level API to define models and data processing pipelines, from data validation to serving the models. TFX adds an interface layer between other Google libraries for data transformation, data validation, training, model evaluation and validation, and serving, making it an integrated ecosystem of libraries to work with data.

Spark MLLib [60] is a distributed machine learning library available in Apache Spark as part of the API of the system. It is written in Scala and uses native C++ algebra libraries to improve performance at runtime. MLLib includes Scala, Java, and Python APIs. It supports several methods and algorithms used for machine learning. MLLib integrates very well with standard Spark functionalities, so performance and scalability are at the core of the development.

MLFlow [85] is presented as an open-source platform to handle end-to-end machine learning lifecycles. It is language and library agnostic. It is a high-level API implemented in Python to track the training of ML models and their deployment in production environments. It supports many deployment tools for ML models. It covers three main challenges: experimentation, reproducibility, and model deployment.

Ray [80] is a platform for Reinforcement Learning (RL) tasks. Ray is a dis-

[5] Slides of a presentation of the Bighead framework, https://www.slideshare.net/databricks/bighead-airbnbs-endtoend-machine-learning-platform-with-krishna-puttaswamy-and-andrew-hoh (accessed 14 October 2021).

[6] Official website, http://clipper.ai (accessed 14 October 2021).

[7] Official Kubernetes website, https://kubernetes.io (accessed 14 October 2021).

[8] Official Apache Beam website, https://beam.apache.org (accessed 14 October 2021).

[9] Official Apache Airflow website, https://airflow.apache.org (accessed 14 October 2021).

tributed framework for training agents and interacts with simulations of environments. Tune [78] is a scalable framework for hyperparameter search focused on deep learning and deep reinforcement learning.

An alternative to using already made frameworks is to adopt a standard for development and then standardise the deployment using a standard format. OpenML [49] for model exchange, it defines standards, so it makes it easy for different libraries to report and define machine learning tasks. Facilitates the benchmarking of algorithms and models. Other initiatives include Predictive Model Markup Language (PMML)[10] and Portable Format for Analytics (PFA)[11]. Also, Neural Network Exchangeable format (ONNX) for cross platform deployment of neural networks models.

Finally, we believe that explainable machine learning [87] or Explainable Artificial Intelligence (XAI) [66], [59] is a concept that requires exploration by organisation working with machine learning models [67]. We want to facilitate the introduction explainability to the machine learning development process to the organisation. This would help users understand specific scoring the data and data scientists to understand and tune the ML models created. There are a few libraries for this task [61], [71], [81], [90] that we wanted to explore in this project.

Concept drift [28], the issue that the data may change over time, impacting the performance of the machine learning models, may become a problem in the long run. It would be ideal that the recommended platform for model management can handle this problem. A solution inspired by software development is model assertions [77].

4.5 Technology Review

We wanted the selected platform to handle four types of machine learning models: classification, regression, anomaly detection, and recommendation systems.

We established initial criteria to rank the set of selected platforms. We rated them based on two main factors: 1) their alignment with the current technology stack of the organisation and 2) with the skills set of the collaborators of the data teams at the organisation. In our case, we wanted platforms that integrate well with the cloud provider of the organisation, and analysts could use with Python programming language and Databricks (a commercial version of the Apache Spark project), the main two tools used for data science and machine learning by the organisation's data teams.

We shortlisted MLFlow, Tensorflow Extended, Kubeflow, Databricks, and the Azure Machine Learning service. The initial list of platforms considered more tools like Clipper and Ray.

For machine learning interpretability the libraries LIME[12] [61], SHAP[13] [71], and Aequitas[14] [81] were selected.

[10] Official website, http://dmg.org/pmml/v4-4-1/GeneralStructure.html (accessed 14 October 2021).

[11] Official website of PFA, http://dmg.org/pfa/ (accessed 14 October 2021).

[12] Official LIME's GitHub repository, https://github.com/marcotcr/lime (accessed 15 October 2021).

[13] Official SHAP's GitHub repository, https://github.com/slundberg/shap (accessed 15 October 2021).

[14] Official Aequitas's GitHub repository, https://github.com/dssg/aequitas (accessed 15 October

The main objective is to achieve governance of the machine learning models deployed in production environments. Secondary objectives are 1) to add explainability to the modelling process, 2) use Azure DevOps capabilities for data science projects, and 3) embrace Infrastructure as Code (IaC) whenever is possible.

In ML we can distinguish two main phases, training and inference. We include data preparation tasks as part of the training phase. Inference is about scoring new data with the trained model, also known as *serving* the model. There are many ways to *serve* machine learning models from in-house web service deployments to scheduled deployments on big data clusters.

An option that we don't recommend for this project is to use exchangeable formats to deploy the models. For example, using industry standards such as Predictive Model Markup Language (PMML) and Portable Format Analytics (PFA). The same idea applies to neural networks models with the Open Neural Network Exchange Format (ONNX) project. We don't recommend standardising formats because, in our case, it is not required to transfer the models to third party platforms using different environments or technology stack.

MLFlow was our selected platform for many reasons. It is a product developed at Databricks, it is open-source, and therefore analysts can make checks or modifications on the code. MLFlow is flexible and language agnostic. At the time of this project, it was in beta release, meaning that some functionalities may break in the future. We also recommend the use of AML because it is the closest Microsoft solution for end-to-end ML model management. TFX was selected because it seemed an excellent general solution ML. Kubeflow was chosen because it manages ML models in Kubernetes, a technology that is gaining interest from industry[15].

4.5.1 MLFlow

MLFlow is an open-source platform developed at Databricks to handle the machine learning lifecycle.

MLFlow deploys an internal server to store information about the models. The user just needs to add calls to the MLFlow API into the training code of the models. The models are stored with their dependencies. This allows repeatability and reproducibility of the machine learning projects.

MLFlow [85] uses the concepts of *experiment* and *run*. An experiment contains zero or many runs. A run is a single execution of the training code so that the user can call the MLFlow API to log metrics, parameters, notes, the ML model itself, files, etc., to the server. When an analyst trains an ML model, will start a run of MLFlow, log relevant information about the model to the server, and attach the run to an experiment to review the results later.

MLFlow offers three APIs that can work independent of each other.

1. *MLFlow Tracking*: allow to logging metrics, parameters of the experiments and runs, basically lightweight data.

2021).

[15] Official Kubernetes website of industry adopters, https://kubernetes.io/partners/#kcsp (accessed 15 October 2021).

2. *MLFlow Projects*: is an encapsulation of the project, it saves the description of the libraries and code of the ML project for reproducibility.
3. *MLFlow Models*: allow to deploy the ML models to downstream tools, MLFlow call it *flavors*, and each different tool is a different flavor.

Analysts can deploy the models stored in MLFlow in several downstream platforms for serving. MLFlow provides integration to deploy the models to H2O, Spark, MLeap, Tensorflow, etc.

MLFlow presents interesting features such as:

- It works with any machine learning library and is language-agnostic due to its RESTful API and includes a Python API.
- It is designed to scale out to large organisations.
- It allows to deploy ML models to different tools such as AML.

4.5.2 Azure Machine Learning (AML)

AML[16] is a service provided by Microsoft Azure. It requires having an AML workspace in an Azure subscription. The workspace will be the place to log experiments, metrics, models, create model images, create container compute targets, and deploy ML models.

AML relies heavily on Microsoft's Python SDK, specifically on the library `azureml`. `azureml` can control other Azure resources such as Azure Databricks (ADB), Azure Container Instances (ACI), Azure Kubernetes Service (AKS), Azure Machine Learning Studio, Virtual Machines, etc. The `azureml` library is closed source, the API and functions are well documented, but the user cannot access the source code.

The use of AML is constrained to the use of Python libraries. ML models can be trained using the user's local resources *compute targets* in Azure using ADB or AKS, for example, and models can be deployed in containers to ACI or AKS from the model registry.

The training and deployment of models can be done via Python notebooks or Python scripts. The monitoring can be done via a web interface in the Azure portal, making it more straightforward for the data engineering team to overview the models and their deployment.

AML was launched by Microsoft and had good documentation, and it is very stable. The *azureml*[17] library and the supporting documentation is constantly evolving.

AML allows the deployment of ML models to several targets. The main two are ACI and AKS.

4.5.3 Tensorflow Extended (TFX)

The *TFX* [64] library was recently launched at the time of the implementation of this project by Google. It is a general-purpose machine learning library. It

[16] Official AML website, https://docs.microsoft.com/en-us/azure/machine-learning/ (accessed 15 October 2021).

[17] Official azureml website, https://docs.microsoft.com/en-gb/python/api/overview/azure/ml (accessed 15 October 2021).

encompasses tasks from data transformation to model deployment and is meant to handle the entire process of ML. *TFX* is wrapping up a set of useful Google libraries for ML. This high-level abstraction makes it possible to use these sets of, in the past independent, libraries together.

TFX is highly based on the *tensorflow* [53] library. *tensorflow* is a computing graph programming paradigm by Google. It is a great resource for neural networks models. For traditional machine learning models it requires some adaption to the *tensorflow* syntax.

TFX presents an entirely new set of tools compared to the current data science ecosystem of the organisation. To deploy the models will imply using Apache Airflow or Kubernetes workflows; neither of them is part of the current technology stack and is not planned to incorporate them at the moment. This was enough reason to stop exploring the use of *TFX* as a platform for ML model management.

The *TFX* platform is different from the *tensorflow* library. The latter might be desirable to use in future projects involving deep learning challenges.

4.5.4 Other platforms

Clipper [65] was evaluated but it was still in Alpha release. The functionality it offers is one of the best ones available in the market, but we think it needs a company behind to be trusted by the industry. The feedback and model ensembling features are unique among the tools evaluated.

ModelDB [62] seems to be a good tool. It has good functionality for tracking the performance of the ML models. But at the time of the evaluation, its repository did not have changes in the last four months. It was recommended as an option for model management in Microsoft ML documentation. One of the co-creators in 2016 was Matei Zaharia, the creator of Spark and founder of MLFlow. It is fair to say that most of the functionality of ModelDB was transferred to MLFlow.

Kubeflow[18]. We stopped the evaluation after evaluating TFX. Kubeflow would have been the next one to evaluate. Kubeflow was added because it makes deployments of ML models using Kubernetes (a container orchestration platform). The organisation had no developments using Kubernetes, and this could have represented the incorporation of a new technology to maintain in the organisation's technology stack.

4.5.5 Interpretability

Subjects such as interpretability and explainability are gaining importance among the ML community. It is expected that models provide explanations of the results they produce. There is a more significant concern in this regard when algorithms are scoring people. We'd argue that adding explainability to the results of predictive models used in the manufacturing industry can add value to the business. For example, explaining why an anomaly is detected could help better understand how customers are using the device and, at the same time, will drive insights about the design of the mechanisms the user uses.

We expect that the ML models deployed in a production environment are interpretable. A human should understand the impact the input variables have on the

[18] Official Kubeflow website, https://www.kubeflow.org (accessed 15 October 2021).

result from a model. This is not a compulsory requirement, but it is a desirable one. Analysts could provide explanations to the technical teams of why an anomaly has been detected.

The use of an interpretability framework could benefit the team as we deliver and support other business areas; this could engage even more the users of the ML models. It also could help to understand why some assets present potential failures.

As the number of machine learning models grows, it will be more critical to explain their scoring. We should avoid implementing models as *black boxes*. With zero or minimal knowledge of how the model or the algorithm works and then use them in production, this will potentially increase the hidden technical debt [52] of the machine learning process.

We surveyed three libraries: LIME [61], SHapley Additive exPlanations (SHAP) [71], and Aequitas [81]. The Sequential Feature Explanation (SFE) [90] for time series data was included in the readings but not tested due to time constraints in the project.

LIME [61] can be used for the explainability of any classifier. LIME creates local decision tree models to explain how the studied model makes decisions. The SHAP library has a more general approach but only provides single point explanations and contains the LIME algorithm with other algorithms to explain neural networks (DeepLIFT). Aequitas serves a different purpose. It can tell if a feature has misrepresented values if there is bias in the data.

4.5.6 Coding Guidelines

The recommendations presented in this chapter come from internal practices of the data science team of the organisation, for example, in the use of a *Take on* document at the beginning of the projects. Another example from experience is framing the data science question, as it has been the standard practice for each EngD project that started framing the problem in a question that needs further exploration and experimentation. The documentation process and folder structure for data science projects come from two primary sources: the Team Data Science Process[19] created at Microsoft and the Cookiecutter Data Science by DrivenData[20].

As a first step, completing the following checklist at the beginning of any project is recommended:

- Define the goal of the project. Frame the objective of the project as an open question to be answered with the project.
- Identify the data sources for the project. Assess the existence of the necessary data internally or externally (requires data collection).
- Is the data available validated by the stakeholders of the project? This should be a milestone to move forward with the project. Analysts should do no work with data that is not approved or inconsistent with the operational systems or business reports unless there is explicit approval from the project's stakeholders.

[19] Official documentation website: *What is the Team Data Science Process?*, https://docs.microsoft.com/en-us/azure/architecture/data-science-process/overview (accessed 17 August 2022).

[20] Official documentation https://drivendata.github.io/cookiecutter-data-science/ (accessed 17 August 2022).

- Create a code repository for the project with folders: *documentation*, *notebooks*, *code* and *tests*. Add the files: *README.md*, *charter.md*, and *exit_report.md.*

As part of the evaluation of tools, we considered it relevant to recommend good practices to follow in developing machine learning models. We recommended a machine learning process to develop models. The process is presented in Figure 4.1. We explained what tasks should happen in each one of the phases of the machine learning process:

- **Exploration**. Description and profiling of the data to be used in the project. Data provenance is desirable (know from where the data is coming from and the process until the current state).
- **Preparation**. Data cleaning and data transformation of the variables. This is a two-fold process to prepare the data for the training step of the machine learning models and create a repeatable process to be embedded in the pipeline of the project, so when new data arrives, it can be processed by this code and then scored by the model.
- **Training**. Depending on the type of problem, the analyst can select the algorithms and techniques to approach the problem. This is an iterable phase and may require going back to data preparation to get better data.
- **Evaluation**. Once the models are trained, there must be a way to decide on better models. This is done with performance metrics depending on the type of problem. There are many performance metrics to choose from. For classification problems, it is useful to have the confusion matrix and the model's lift curve and the accuracy and recall of the models. For regression, a loss function would be sufficient.
- **Selection**. Model selection is based on the metrics computed during the evaluation of the models. The analyst will want to select a model for good performance but might include other criteria such as interpretability (if the problem requires it), computation power needed to score new data or retrain the model, etc.
- **Deployment**. The model deployment consists in making the model available for scoring new data. This is called serving the model. There are two main ways to deploy ML models. Batch scoring, where new data is scored in batches at once by a version of the model, this process is offline, and the user gathers the data first and then applies the model to the data and gets the results. The other way is online scoring, usually deploying a web server wrapping up the model with a RESTful API to communicate with it. The user will send the data as a message to the server, the server will process this incoming new data and return the result of the model.
- **MLFlow**. This is the tool for tracking performance metrics, model parameters, files and the ML models. It is a centralised server able to store this information about the models.

Git

We strongly recommended familiarity with *Git* for versioning code. Any approach to DevOps or DataOps approach should ensure the skill development of code versioning of the team members. We prepared an internal tutorial about code versioning for the data science team as part of the activities of this project where we introduced the philosophy of code versioning, presented the most common commands used to work with Git and presented a workflow for collaborative work on code repositories. We shared resources to learn more about the work with Git.

The structure of the repository may very from project to project but some folders and documents should be expected to have consistency in the development of machine learning processes:

- A *code/* folder containing all the source code used for the project. In this folder we can create *experiments/*.
- A *data preparation/* folder.
- A *training/* folder.
- A *deployment/* folder.
- A *documentation/* folder with all the relevant information for users and stakeholdes.
- A *notebooks/* folder. Most of the exploratory work happens in notebooks so this way we can track the interactive part of the process. It is separated from the *code/* folder because the work in notebooks is often more exploratory and open ended, if a notebook becomes the final version it should be moved or created in the *code/* folder.

Machine learning development

We added recommendations and ideas to the development guidelines. We thought of the scenario when an analyst starts a new machine learning project. We added checklists to follow to gain insights from the data sources of the project [52], [82], [68].

We suggested to create a *data dictionary* for the data used in the project. The data dictionary should be updated every time new data is added to the project. This could lead to a data catalogue or to feed the enterprise data management strategy.

There are several steps for data exploration and this can be carried on in a modular way. The key to *data exploration* is to start answering simple questions such as *what*, *how many*, *how much*, *where*, *when* and interrogate the data. We made the distinction of analysing discrete and continuous variables, and we also made suggestions about univariate and multivariate analysis.

For univariate analysis we recommended to consider separate analysis for continuous variables, discrete variables and time series data. Continuous variables are often associated with numeric columns and/or real numbers. Discrete variables are associated with string values or they are also known as categorical variables. They take a finite number of values. And time series data is often columns of the *time*, *date* or *timestamp* data type. For multivariate analysis we recommended to run a correlation analysis and analysing the dependent variables against the target variable, the variable that we want to predict.

For example, in univariate analysis the aim is to summarise the content of the variables to gain understanding about the problem. For continuous variables in the data sets the analyst could plot a histogram and a boxplot of the variable. A graphical analysis helps to understand the distribution of the data. Also computing some statistics could help too, statistics such as the mean, median, variance, standard deviation, maximum value, minimum value, percentiles 1%, 2.5%, 5%, 10%, 25%, 75%, 90%, 95%, 97.5%, and 99%. The covariance matrix to check linear association between variables. A covariance equal zero if the variables are independent.

For discrete or categorical features we recommended to compute the frequency to understand the distribution of the values of the variable, what are the most frequent values, how many unique values the variable contains.

For time series data we recommended to plot the tendencies of the variable. Group the data by day or month and plot the tendency of the the data and counting the number of records that are grouped per time unit. Identify when was the last data point in the data set, what is the span of time that the variable contains data, etc.

After the data exploration phase a summary report should be created and shared with the stakeholders. Early findings should be shared early in the process.

Data modelling

There are several ways of classifying the type of problems in machine learning. One is to divide the problems and methods in supervised learning, unsupervised learning, online learning, and RL. Often, companies work with the first two types of ML. Deep learning techniques can be classified under supervised, unsupervised and RL methods.

In supervised learning, we find two main subtypes of problems: classification and regression problems. Classification is when the target variable has discrete values. Regression is when the target variable is continuous. We find three main techniques used in unsupervised learning: clustering analysis, correlation analysis, and association rules.

It is crucial to frame the machine learning project as one of these types of problems. Once we have formulated the problem, we can suggest techniques used to solve these subtypes of problems. Some problems will instinctively be classified into a category, but we could frame it in a different class to solve it. For example, the prediction of the contract value is a regression problem. Still, we can frame it as a classification problem by predicting the contract value will be greater or less than a threshold. We want to do this because a binary problem is often easier to model and solve.

Regarding the target variable in supervised learning problems, it should be clear what it represents, what it measures, and the domain of the values it can take.

Documentation

The documentation is crucial for any project. It facilitates communication with users and stakeholders. It makes more straightforward the maintainability of the project.

We suggested writing all the documentation of the project as *markdown* files and as part of the repository. A machine learning or data science project documentation should include information about the system architecture, data dictionaries, reports about data exploration and modelling, planning docs, information obtained from a business owner or client about the project, the presentations prepared to share information about the project.

4.5.7 DataOps

DataOps is a concept derived from merging DevOps and the roles of data professionals such as data scientists and data engineers. DevOps is the concept of putting together software developers and the people from operations who care for the software once it is deployed in production.

DataOps is characterized by developing data projects in an agile way. Using task management systems to track requirements and maintaining a backlog of tasks sorted by priority. Using *git* repositories for code versioning. Testing and deliverables are different in ML projects if we compare them to software development. The testing phase will often test the deployment of a web server from where the final model will be served or test infrastructure deployment for processing data in batch mode.

The concepts involved in DataOps are attractive to implement in agile teams because it allows shortening the delivery times of data products.

4.6 Evaluation

In practice, all the evaluated technologies made use of container technologies in one way or another. The most common use was to put the models into containers with all the required libraries to query the models and maintain communication through an API.

Clipper would be useful when real-time serving is critical, where one problem is the interactive serving demand of different workloads.

Standardising the deployment with OpenML, PMML, PFA, or ONNX add additional overhead to the development that can be avoided if these steps of compatibility or exchangeable formats are not a requisite in the first place.

TFX is highly centred around the TensorFlow library for graph computation. TensorFlow is used by models using neural networks, but using it with traditional machine learning algorithms adds an extra difficulty because the analyst should adapt the algorithms to the TensorFlow syntax. The use of TFX would require to include at least two new tools to the current toolbox: Apache Beam to coordinate executions of tasks and Kubernetes or Apache Airflow to serve ML models. TFX controls and provides APIs to execute, but it doesn't offer infrastructure for processing.

MLLib is commonly used to train ML models and the exporting the models to a format that the analyst can use to serve the models such as MLeap[21] or Python functions if the model was trained with the PySpark API.

Concept drift [28] is something relevant to consider. The issue is that the models will decay their performance over time after being deployed in production. There are alternatives to avoid this problem. A closer look into this problem would imply a

[21] Official documentation of MLeap, https://combust.github.io/mleap-docs/ (accessed 15 October 2021).

Service	Price
AKS	~$100.317/month[26]
Container Registry	~$18.676/month
Block Blob storage	~$0.0644/month
Key Vault	$0.03/10,000 transactions

Table 4.1: Table with Azure services and price for AML

small project. From this project, alternatives are suggested, such as periodic retraining of the ML models. Also, building a periodically scheduled pipeline can evaluate the performance of the model on new data. MLFlow can help with monitoring candidate models.

4.6.1 Security

MLFlow has no security implemented as part of the platform. It depends on the security of the infrastructure where it is installed. The recommendation is to use the MLFlow version that comes with ADB. That way, the security relies on the access ADB in Azure.

AML has high and several levels of security. It requires a valid account with access to AML *workspace* in Azure; then, when every new logging in, it will ask for a code provided by the `azureml` library that should be input to Azure, only then the user will be able to use the /textitAML and Azure resources.

4.6.2 Cost

A version of MLFlow comes with ADB that is already part of the organisation's tools to be used at no extra cost. MLFlow is open source, and it could be installed in a standalone server to start using it.

The pricing of AML is per use. It uses Azure Kubernetes Services (AKS)[22] to deploy ML models and additionally Azure Container Registry[23], Block Blob storage[24], and Azure Key Vault[25]. The Table 4.1 contains estimated prices for AML usage.

4.6.3 Recommendation

After evaluating different tools for machine learning management and libraries and platforms for specific ML tasks, the proposal uses two platforms together. The platforms have minimal overlapping, and the combined benefits from both tools and the main gain are the flexibility in the development and deployment of ML models.

MLFlow can be used at the training phase, and the analyst could use it for tracking metrics, parameters, files, and ML models logging the information about the candidate models, and **AML** can be used to deploy and serve the selected models to Azure infrastructure which can be ACI (Azure Container Instances), AKS

[22] https://azure.microsoft.com/en-us/pricing/details/machine-learning-service/

[23] https://azure.microsoft.com/en-us/pricing/details/container-registry/

[24] https://azure.microsoft.com/en-us/pricing/details/storage/blobs/

[25] https://azure.microsoft.com/en-us/pricing/details/key-vault/

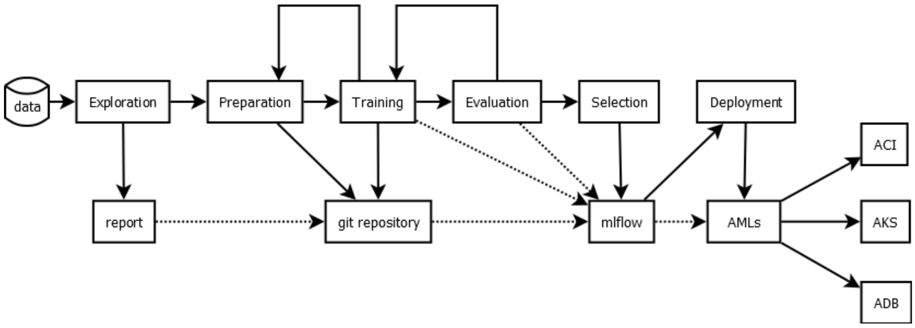


Figure 4.1: Workflow of a machine learning project using `MLFlow` and `AML`.

(Azure Kubernetes Service), ADB, or VMs (data science virtual machines).

We present our suggested process to develop and deploy machine learning models in Figure 4.1 using *MLFlow* and *AML*. The training phase involves data preparation, training, evaluation, and selection of the models and relevant information be tracked with **MLFlow**. Then, the selected model can be deployed using AML to AKS or ACI.

Some of the reasons for this recommendation are:

- MLFlow is a flexible library and programming language agnostic platform. It has a well documented and simple Python API.
- MLFlow is open source and developed by Databricks. It integrates very well with Azure Databricks (ADB) that provides an MLFlow instance with its services.
- A migration of MLFlow servers will be transparent, as the user needs to change the URL of the new server and the logs will be sent to the new server.
- MLFlow Models allow to deploy the models in Azure infrastructure and many other downstream platforms for serving models.
- AML has the UI inside Azure, making it easy to monitor by the Data Engineering team.
- AML uses Microsoft's Python SDK to control the aspects of deployments. Deployments could be controlled by code in the repository.
- AML integrates very well with other Azure services.
- MLFlow won't impact the way we write code as a few lines should be added to track metrics, parameters and the models (as we saw in the MLFlow demo)
- AML will extend a little bit the code as a call to the API should be added to:
    - register the model
    - create an image for the model
    - optionally create a Compute resource to batch process new data
    - optionally deploy the model to AKS or ACI

Some of the reasons for not using MLFlow alone or AML alone:

- MLFlow depends on other platforms to deploy the models. It is an external library to manage the lifecycle, but it doesn't offer processing or deployment capabilities but connections and integration to the most used platforms for deployment. An alternative would be to use Clipper.ai for deployment together with MLFlow
- AML is constrained to Python models, so flexibility is lost by using it for the training phase.
- AML is restricted to Azure platforms being ACI or AKS the main ones.

MLFlow could be beneficial to compare and track relevant data of ML models at the training phase. The analyst can use it to standardize the way of model evaluation and selection. The use of AML could be beneficial to deploy ML models to AKS or ACI. Currently, the five models in production can be managed by scheduling ADB notebooks. AML could be used in the future as a sound solution to scale the deployment of ML models.

Chapter 5

Analytics repository for telemetry data from IoT projects

Summary

This project presents the design and implementation of an analytics database for telemetry data. This database should receive data from generators and other types of devices deployed in site projects worldwide. A use case for this project is to implement a set of alarms to prevent failures in machines and a machine learning model.

5.1 Background

IoT projects are factors of the increasing volume of data generated around the world. The promise of remote monitoring of assets becomes a burden when installing, transmitting, and then maintaining the solutions.

This chapter describes an end-to-end proof of concept project to receive and analyse telemetry data coming from sensors from machines deployed worldwide in remote project sites.

For any IoT project, we need many components in place before analysing any byte of data. The sensors are deployed and configured to collect the data. They are connected to a system that receives the data in the same location as the sensors. The system that controls and collects data from the sensors requires to be operative 24/7 in the project site; most likely, it will require additional storage and backup systems to operate. The data from the monitoring system will require a network to transmit the data, 2G, 3G, 4G, 5G, satellite, internet, etc. The type of network imposes constraints on speed and volume that can be transmitted and cost. On the other side, it is required to have a data broker, a platform able to receive *all* the incoming data packages and assure there is no missing information, and it is able to process them. After the data broker, a permanent storage platform like a relational database or No-SQL database is required to serve the analysis. A data warehouse or a data lake solution. A platform or framework to support the data the *data pipelines* or the data movements between the remote sites with the sensors and the monitoring system and the repository where the data will serve the analysis. Finally, a platform to provide access and computing power to process and build analysis using the sensor data. Additionally, some solutions such as predictive maintenance machine learning models will be built and deployed to infrastructure monitoring the new data.

Data problems may arise if the business wants to analyse tendencies, comparisons between different locations, project sites, or the same asset model deployed to other customers. When the data is in data silos, data integration and time opportunity are the main risks for successfully implementing this kind of project.

The unique nature of the operations of an organisation generates value for this data. There is no doubt that there is potential value in analysing its own data. There are many ways to learn and improve from analysing this data collected from the fleet's operation and multiple customers.

5.1.1 Data Silos

Companies with multiple data sources sooner or later face the problem of managing their data. It makes sense to build an ad-hoc solution when the data is only used by one area within the organisation. That is how *data silos* are created inside organisations. Problems start to arise when the data is required by other departments in the organisation.

One of the problems of maintaining *siloed* data sets is the high cost to integrate and analyse data. Any analytical question will become in a one-shot analysis. Any updates for that analysis will require collecting the data again and the many people involved in the process. A slightly different analysis using the same data sources, let's say to add an extra column from the data sources, will require to modify every data source and reload the data for analysis. This process is not sustainable over time due to the number of resources needed to make it work.

The solution is a central data repository. A solution that can collect and integrate the data from multiple sources into a data model accessible to analysts and developers to build analysis and machine learning models using this data. We will explain our solution and architecture in this Chapter, but for more details, we presented a similar architecture in Chapter 3.

5.1.2 Description of the Problem

The projects of the Power Solution line of business stores their data locally in industrial computers installed on the project sites with the monitored assets. They use Supervisory Control And Data Acquisition (SCADA) [16] systems to monitor the operations in the project site. SCADA systems are control architectures that provide a standardised framework for high-level supervision of the machines and sensors. SCADA systems communicate with the sensors and interface with the controllers and actuators of the industrial system. The SCADA system in the project site collects and stores the data locally.

Data locality is optimal from a project site perspective because it minimises the data that needs to be transited outside the project site and efficiently maintains the system. From a company-wide perspective, it would be better to have visibility of the *all* the projects, if possible, and asset performance from a single data repository.

Currently, the Rental Solutions line of business provides a service where rented assets can transmit telemetry data to a central database making it simple to monitor and analyse them. The *in-transit* data is managed by a third-party provider and making it available to the organisation granting access to a telemetry database. The data from that telemetry database is ingested in the data warehouse and data lake of the company from where it can be analysed by analysts and data scientists. This process allows the analytics team to compute analysis and create proactive and predictive alarms.

The telemetry database used by Rental Solutions described in the paragraph above was used for the projects described in Chapters 7, 8, and 9.

This project implements a proof of concept for the Power Solutions line of business. Enabling the storage, analysis, and computation of the data from assets that in the past were controlled locally from within the project sites. It presents the design and implementation of a database to store the messages generated in the SCADA systems and a pipeline to update the data models, and a way to run proactive alarms using the database and framework.

5.1.3 Proactive and Predictive Alarms

For this project, a *proactive alarm* is a single threshold trigger. It is defined by a *signal* to monitor, a *threshold* value, and a *time interval*, and an *instruction* for above or below the threshold. If a *given* signal remains above or below the *threshold* for more time than the defined *time interval* then the proactive alarm should be triggered and inform other systems and the relevant people about its occurrence.

For this project, a *predictive alarm* is a multi-variate analysis created with a machine learning algorithm or heuristic to spot anomalies or unexpected behaviour of the signal. Predictive alarms often take more time to develop and validate. They require a training dataset from where to draw conclusions about the signal.

Both proactive and predictive alarms are based on domain knowledge from experts and implemented by data scientists. People who monitor the machines' performance led and prioritised the development of new proactive and predictive alarms.

The alarms created are based on the needs of the business and stakeholders. Proactive alarms are less expensive to implement than predictive alarms. Proactive alarms can prevent failures before the predictive alarms. Both types of alarms may present a high rate of false positives in the early stages of development. Predictive alarms will require more supervision and monitoring after deployment as the results can be dynamic. Proactive alarms are simpler as they only need *what if* scenario analysis that can be computed from historic data.

So far, most of the alarms of the company are for rented assets. This is not the case for assets deployed in long term projects, where a customer will use the machines for months or multi-year projects.

The definition of success for this project is:

1. The implementation of a database that can receive all the data generated in project sites.

2. The implementation of the ingestion process to collect and transform the incoming data into a table readable format for data analysis.

3. A set of proactive alarms is implemented on top of the analytics database.
4. A set of predictive alarms can be implemented on top of the analytics database.

5.2 Problem Statement

There are many options in the market to implement architectures to analyse sensor data. Most of the solutions will include a database for persistent storage and the use of the SQL language to query data sources and in-transit data; some programming language such as Python, R or Julia for pre-processing or transforming the data; some of the popular libraries for machine learning and data engineering of these programming languages; platforms to process massive volumes of data such as Apache Hadoop, Apache Spark or similar; and the decision of hosting the solution in a cloud platform, in an on-premise infrastructure, as an hybrid solution, or as a multi-tenancy solution.

The question is, *can we design and implement an analytical database for time series data processing and monitoring alarms that integrates with the current data pipelines of the company?*

5.3 Architecture Design

This project was implemented using data from a mining project in Africa with around twenty machines deployed on site. The project started with the design of the database and the data transformation processes to move the data into the database. This EngD project assumed that other on-site projects would send messages in a similar format and syntax containing similar information.

This solution considered multiple architecture components and had to consider some constraints[1]. First, we want a modular solution to improve the building components independently from other components in the architecture. A similar approach to modular design is explained in Chapter 7 to design the deployment of an estimation model.

In both designs considered the principle *Plan to throw one [design] away* from the Mythical Man-Month book [10]. Without assuming that a first design could be changed in the future would have put more pressure on the delivery, but as we were expecting an evaluation and potential refactoring or redesign, it helped the quality of the solution delivered. Developing this mindset was helpful because we re-design the database and the data pipeline to update the data model.

Scalability and *maintainability* were also considered for this EngD project. The solution should be able to add data from more project sites without impacting the data pipelines' performance or the data processing tasks. And it should be clear how and what to modify in the project for future requirements.

The next step in the design is to choose the technology stack for the project. As it is a new project implementing something that is not in the current stack, we need

[1] A technology constraints for this project was the use of a big data processing platform in the cloud in the cloud provider of the organisation. This constraint dictated the selection of Databricks on Azure as the platform to implement this EngD project.

to be careful with the technology selection. We will explain the design decisions in this chapter.

There are many ways to implement the proof of concept. We are using a combination of multiple technologies and frameworks.

At the beginning of the project, we considered the current technology stack for data analytics projects and tried to adapt it in the best way possible to the known practices. We added a couple of technologies to grow the potential and impact for the team. For example, for data processing, we used Databricks, which is the chosen solution for data pipelines and data processing tasks, and we added Delta tables as permanent storage.

For the data model design, we considered the existing data model for rental solutions. The idea was to make the data models reusable and that algorithms that run in rental models could run for power solution models and vice versa.

We designed a modular solution separating data collection, computation, permanent storage, analysis, and data pipelines to process and update the data models. We created the data model from scratch based on the messages generated by the SCADA[2] system and the processes for updating it.

An initial data mapping between the rental solutions data model and the message tags used by the SCADA system that covered fields used by the machine learning models for rental solutions was attempted. The mapping was unfinished but it was sufficient to start the project with a basic idea of the required fields from the SCADA system. In any case, there is no missing data because the data model was able to contain all the existing messages from the SCADA system. And a mechanism to detect changes or new message tags that would be not loaded to the database was added.

We reverse-engineered the tag messages from the SCADA system to design the data model for the database, and we aimed to store 100% of the messages received.

After exploring the data, we discarded the alternative solution of storing the SCADA messages directly into the telemetry database. It was not feasible for multiple reasons. The messages don't have all the columns that the rental data model requires; we didn't have access to the metadata or how technicians configured the assets to send the data, so all the complementary information would not be present in the database, only the signals. The sample rates were different, and there would have been no context or record to store when it is changed. Also, the units of the signals were inferred by their ranges, but it is a barrier to store the data in the same data model.

5.3.1 Architecture and components

To implement proactive and predictive alarms, we need a data model. To update that data model we needed data processes to move the data from the landing area, transform it, and load it to the data model. To have a landing area, we needed a data broker to receive the data messages and store them until they can be moved to a permanent storage layer. Additionally, we required mechanisms to identify changes in tag messages and keep the history of the alarms sent to users.

[2] Supervisory control and data acquisition (SCADA), is a control system architecture used to operate the machines and plants on-site.

The tags used in the messages from the SCADA system are configured on the project site, and we have little or no influence on the format. We could make suggestions to implement them in the future. There was no standardised way for the syntax of the tags, but there was some sort of consistency in the tag patterns.

The first step was to create and connect a data broker solution in order to receive the data from the project site. We used an Azure IoT Hub.

We planned the project in five sequential phases that are dependant on previous ones:

1. Exploration of the data generated by the SCADA system.
2. Design of an analytical database that allows queries and data modelling.
3. Design of the data ingestion process.
4. Implementation of the proactive alarms.
5. Implementation of the predictive alarms.

Scalability is an important criterion for the design. Ideally, a project site will start sending data to the central data repository with no impact on performance for other project sites. Adding new project sites should be transparent with minimal configuration on their side. It is expected that the solution will have similar performance for one project site as for hundreds of them, considering that the projects and data collection have reasonable configurations like the ones described for this project.

An alternative architecture that we didn't test was implementing a time-series database to analyse and create alarms on top of it and a data visualisation tool to monitor it. A technology stack like this would have made sense as the data from the projects are of the time-series type, but we did not have the time to test it. The suggestion was to start testing a platform like InFluxDB[3] with a tool like Grafana[4] to visualise the data. Both tools are not in the company's technology stack, which was the main reason for not testing them in the first place. The testing of this tool is suggested as further analysis for this project. The alternatives in Azure is Azure Times Series Insights[5]. The Azure option is paid, whereas InfluxDB and Grafana are open-source projects.

The requirements for this project are similar to the ones presented in Chapter 3 about a streaming analytics architecture. In fact, the purpose of that experiment was to collect data from multiple assets around the world and being able to analyse and query the data and also generate machine learning models that use the data. The streaming analytics architecture suggested three years prior to the start of this project still makes sense, and the recommendations still hold. We will be using the same components.

[3] Official website of Influxdata, creators of InfluxDB, https://www.influxdata.com (accessed 12 August 2021).

[4] Official Grafana website, https://grafana.com (accessed 12 August 2021).

[5] Official Azure website of Time Series Insights, https://azure.microsoft.com/en-us/services/time-series-insights/, (accessed 12 August 2021).

We need a *data broker* to receive the incoming data. A *Data storage* to save the files and documents generated by the processing. A *database* to query the data for analysis. And of course, a *data processing* platform.

Data Broker

A data broker is a system that receives input data and delivers it to other systems using a certain logic that can be programmed in the broker. A data broker system allows scaling out the solution. If there is an increasing demand from project sites to send their data, the broker system will scale adequately to meet the demand.

A data broker will work as an infinite queue of messages. In our case, each message is a data file transmitted from a project site. The data broker will deliver the messages to the corresponding storage accounts. For this project, a given data broker, an Azure IoT Hub[6], stores the data in a given blob storage location in a cloud provider.

Data Storage

The next component of the architecture is where the persistent data will be stored. There are two instances where storage is required. One is to store the incoming files from the project sites, and the other is to store the content of the tables of the analytical database.

The storage is in a cloud provider. The blob storage selected for this project scales with the demand. It allows to connect and mount the storage from the processing environment.

The platform for data storage is Azure Blob Storage account[7]. Technology used in other data related projects in the company.

Data Processing

The data processing environment allows to explore and analyse the content of the files received from the project sites. It is the computing engine that data scientists can use to run analysis on the data. It is also used to design the ingestion process for when the data needs to be loaded to the database in an incremental way.

The data processing platform is Azure Databricks. Databricks is a solution provided by the company with the same name. It offers an enhanced version of the Apache Spark[45] platform. Azure Databricks is the current data platform used by the data science and data engineering teams at the company.

Database

The database engine selected for this project is the Delta storage solution. Designed by Databricks and open-sourced and donated to the Linux Foundation[8] in 2019.

[6] Official Azure IoT Hub website, https://azure.microsoft.com/en-gb/services/iot-hub/ (accessed 25 August 2021).

[7] Official Azure Blog Storage service website, https://azure.microsoft.com/en-gb/services/storage/blobs/ (accessed 25 August 2021).

[8] https://www.linuxfoundation.org

Delta storage[92] is a file system that provides ACID transactions. An ACID transaction is a common feature found in relational databases. It is the bare minimum for a system to be called a relational database, but it is not something that distributed file systems offered until now. What was common in file systems is eventual consistency. That the system will *eventually* be in sync in all its partitions and replicas.

Eventual consistency follows the CAP theorem[19],[17]. That means that you can expect two out of three of the following requisites: Consistency, Availability and Partition tolerance at any given time and that the third will *eventually* be fulfilled.

The Delta Project was pretty new, just launched and open-sourced when the project described in this chapter started. The decision to use it was because this is a proof of concept project, so we are allowed to test new and potentially useful new platforms and also because it presented some features that can be beneficial for data processing projects. For example, Delta has an off the shelf way to update data incrementally that makes the data integration process efficient. It also abstracts the handling of the distribution of the data in the clusters.

5.4 Benefits of a single data repository

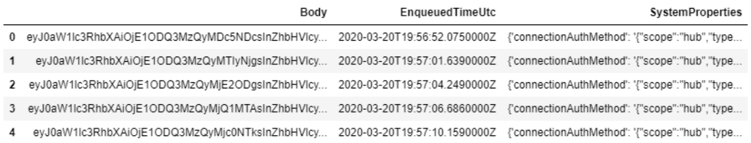
There are multiple benefits in transmitting the data to a data repository and, for example, generating statistics across different project sites. Aggregating data from dozens of sites using hundreds of machines would allow a better understanding of how the assets are being used and, in the future, to profile the project sites and customers by actual usage of the assets. The comparison of workloads among assets and customers would allow identifying differences in geographic locations and workloads of the assets. It would enable the generation of knowledge for preemptive maintenance of the machines.

Accessing the data generated on remote project sites would allow the company to profile the usage of the machines, the type of project sites, and finally, the type of customers. This understanding would allow to prevent failures and improve the maintenance cycles for each machine type and model. Alarms could notify about critical events before reaching critical levels in the workloads. The analysis of project sites represents new knowledge for the company that so far has more knowledge about the machines used in the rental business, focusing more on short term contracts that are more numerous.

The advantages of having the data available in the Datalake are many. The data will be ready for analysis; analysts will have a single source of truth to generate results; solutions can be scalable. Adding more data sources is transparent for the users, and more data in the system can only enrich the analysis.

5.5 Data Model Design

Designing an end-to-end solution means thinking ahead and about the different components of the solution. It also means to prepare processes for when the assumptions considered at the beginning of the project may change—creating policies to deal with expected and unexpected changes.



	Body	EnqueuedTimeUtc	SystemProperties
0	eyJ0aW1lc3RhbXAiOjE1ODQ3MzQyMDc5NDcsInZhbHVlcy...	2020-03-20T19:56:52.0750000Z	{'connectionAuthMethod': '{"scope":"hub","type...
1	eyJ0aW1lc3RhbXAiOjE1ODQ3MzQyMTIyNjgsInZhbHVlcy...	2020-03-20T19:57:01.6390000Z	{'connectionAuthMethod': '{"scope":"hub","type...
2	eyJ0aW1lc3RhbXAiOjE1ODQ3MzQyMjE2ODgsInZhbHVlcy...	2020-03-20T19:57:04.2490000Z	{'connectionAuthMethod': '{"scope":"hub","type...
3	eyJ0aW1lc3RhbXAiOjE1ODQ3MzQyMjQ1MTAsInZhbHVlcy...	2020-03-20T19:57:06.6860000Z	{'connectionAuthMethod': '{"scope":"hub","type...
4	eyJ0aW1lc3RhbXAiOjE1ODQ3MzQyMjc0NTksInZhbHVlcy...	2020-03-20T19:57:10.1590000Z	{'connectionAuthMethod': '{"scope":"hub","type...

Figure 5.1: A message generated by the SCADA system in the project site landed in the IoT Hub and moved to permanent storage in a Delta table in Azure Databricks. The *Body* field contains the data from the sensors encoded in base64 format. The *EnqueuedTimeUtc* is a timestamp generated by the SCADA system, and *SystemProperties* contains details of the host system's configuration collecting the data on the project site.

We wanted to use the most straightforward data model that can work and support the project's requirements. It also needs to scale well if the demand grows.

The data is collected in SCADA systems on the project site. Then that data collected is transmitted to a remote repository using standard 2G, 3G, 4G networks or satellite connections from the project site.

Until now, the SCADA data resides only on the project sites. It has not been integrated nor combined with other data sources into a central data repository.

The process to design the data model was analysing the tags used in the messages. We reviewed each tag to understand the patterns used to classify the data. We identify different patterns, and we were able to separate the data used for different contexts. We also identified the aliases of the machines from the tags, so we had to use an additional data source to match up the machines' unique asset codes. This issue was a recommendation to incorporate the actual asset codes or add a data source with this information for each project.

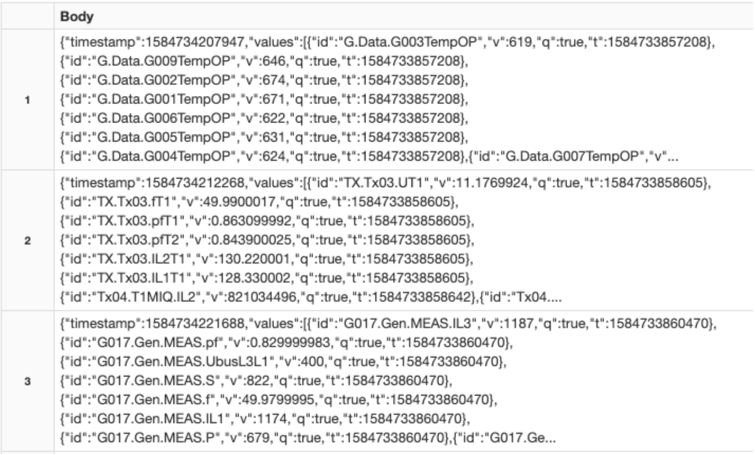
5.5.1 The data

The data from the project site comes in messages formatted as JSON[9] files. Each file contains three fields: *Body*, *EnqueuedTimeUtc*, and *SystemProperties*. The data from the sensors is in the *body* field and is encoded in `base64` as can be seen in Figure 5.1. The EnqueuedTimeUtc is a timestamp generated by the SCADA system and SystemProperties contains details of the setup of the host system.

The decoded body message has two fields. A *timestamp* and a *values* field. The *values* field contains a list of dozens and sometimes hundreds of data structures with four key-value pairs: *id*, *v*, *q*, and *t* as can be seen in Figure 5.2.

The values of the *id* key in the *values* are known as *tags* in the SCADA system and identify a unique pair sensor in an asset. The collection of all the id fields provides a data domain for the data modelling. The design of the database was based completely on the information gathered from the id field. We also had a list of all the IDs or tags used in the project, around three thousand ids, but not all of the possible IDs were reported from the project site. The alias of the asset is included in the list of tags, that means that the same sensor can be repeated for each asset in the list. The *v* field contains the actual value measured by the sensor. The data type of v could be an integer, real or boolean. The *q* is a boolean variable

[9] Official JSON website, https://www.json.org/json-en.html (accessed 12 August 2021).



	Body
1	{"timestamp":1584734207947,"values":[{"id":"G.Data.G003TempOP","v":619,"q":true,"t":1584733857208}, {"id":"G.Data.G009TempOP","v":646,"q":true,"t":1584733857208}, {"id":"G.Data.G002TempOP","v":674,"q":true,"t":1584733857208}, {"id":"G.Data.G001TempOP","v":671,"q":true,"t":1584733857208}, {"id":"G.Data.G006TempOP","v":622,"q":true,"t":1584733857208}, {"id":"G.Data.G005TempOP","v":631,"q":true,"t":1584733857208}, {"id":"G.Data.G004TempOP","v":624,"q":true,"t":1584733857208},{"id":"G.Data.G007TempOP","v"...
2	{"timestamp":1584734212268,"values":[{"id":"TX.Tx03.UT1","v":11.1769924,"q":true,"t":1584733858605}, {"id":"TX.Tx03.fT1","v":49.9900017,"q":true,"t":1584733858605}, {"id":"TX.Tx03.pfT1","v":0.863099992,"q":true,"t":1584733858605}, {"id":"TX.Tx03.pfT2","v":0.843900025,"q":true,"t":1584733858605}, {"id":"TX.Tx03.IL2T1","v":130.220001,"q":true,"t":1584733858605}, {"id":"TX.Tx03.IL1T1","v":128.330002,"q":true,"t":1584733858605}, {"id":"Tx04.T1MIQ.IL2","v":821034496,"q":true,"t":1584733858642},{"id":"Tx04....
3	{"timestamp":1584734221688,"values":[{"id":"G017.Gen.MEAS.IL3","v":1187,"q":true,"t":1584733860470}, {"id":"G017.Gen.MEAS.pf","v":0.829999983,"q":true,"t":1584733860470}, {"id":"G017.Gen.MEAS.UbusL3L1","v":400,"q":true,"t":1584733860470}, {"id":"G017.Gen.MEAS.S","v":822,"q":true,"t":1584733860470}, {"id":"G017.Gen.MEAS.f","v":49.9799995,"q":true,"t":1584733860470}, {"id":"G017.Gen.MEAS.IL1","v":1174,"q":true,"t":1584733860470}, {"id":"G017.Gen.MEAS.P","v":679,"q":true,"t":1584733860470},{"id":"G017.Ge...

Figure 5.2: Example of a decoded *body* part of three messages. Each record contains data from several sensors, *id* is the name of the sensor, *v* is the value the sensor is registering, *t* is the Unix time stamp at the time of the data measurement.

that for most of the data had the value *True*, but we don't know what it represents. And t is a timestamp in Unix time format.

5.5.2 The flow of the data

The data broker, an Azure IoT Hub instance, would receive a message every sixty seconds from the SCADA system, containing data from all the devices connected to the project site. The SCADA system is configured to collect a new data point only if it is different from the previous data point collected. The data received is stored in an Azure Datalake Storage account. The data received includes performance measurements from generators, transformers, weather stations, etc., that are allocated on the project sites of the customer.

The data is sampled for each sensor based on their *sample rate* parameter. The sample rate tells the system how often, in seconds, to get a new reading from the sensor. The sample rate can condition the project's viability as collecting too much data could significantly increase the network and storage costs.

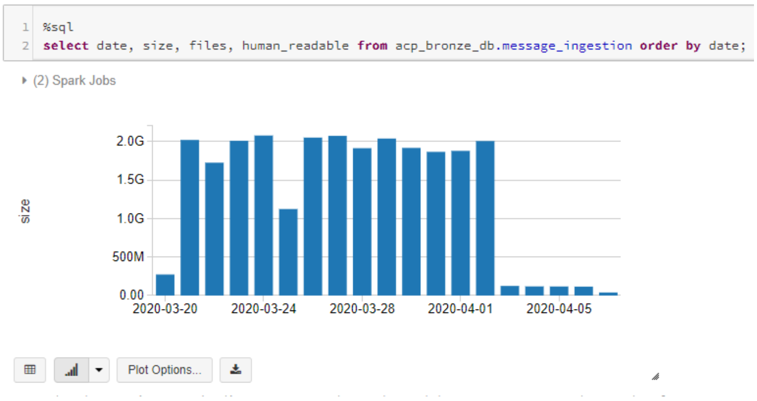
In Figure 5.3 we can see that on the first days of the project, we received messages of around 80MB because the sample rate was one second. That volume of data received decreased significantly when the expert user changed the sample rate to sixty seconds.

The analytical database will be updated every thirty minutes with the new messages since the last update. The alarms triggers will run every thirty minutes, fifteen minutes apart from the most recent database update.

5.5.3 Reverse Engineering

The analysis was carried out using Python in a Databricks environment. To design the database, we analysed the structure of all the id values in the collected messages plus the list of tags generated from the configuration of the SCADA system.

We reverse-engineered the design of the database from the SCADA id messages. Reverse engineering is a technique where we try to understand how a system works from the system's outputs. In this case, we had the tags of the messages containing



```
%sql
select date, size, files, human_readable from acp_bronze_db.message_ingestion order by date;
```

Figure 5.3: Volume of data received per day. During the first days of the project the sample rate was set up to one second, that changed later to sixty seconds.

information about the system that we want to store in a database. We tried to understand what is being measured by analysing the tags or ids. This will give us a good understanding of what tables are necessary for the database. The analysis of the frequency of the messages tells us how often the sensors are sampled, which is an essential factor for partitioning the tables.

5.5.4 Analysis of the data source

The database design was an iterative process. A complete revision of the messages led to the design of the first version of the database. We classified the tags into subjects that represented entities for an entity relation relationship model that translated to the database tables. The early design of the database had six tables to store all the data from the project site: `e_meter`, `g_data`, `meas`, `t1miq`, `t2miq`, and `tx`.

Each table had at least two index columns: the asset alias and the timestamp of the measurement. The columns of the table correspond with the sensor names, so each sensor had a column to report its values. The `meas` table is the measurements table, and it has an extra index column which with asset type as it stores data of the different kind of assets. i.e., generators, transformers, etc. The other tables were asset or theme-specific, so they do not require to include the type of asset.

Not all the sensors report data in every message, so the percentage of NULL values in the columns varies. We observed a pattern of some sensors constantly reporting values, whereas other sensors were sending fewer data translating this issue in columns with sparse data. The analysis of sparse data led to a better design of the database as we used to split some of the tables, a technique is known as *vertical partitioning* in the field of databases.

The eleven tables of the final data model are shown in Table 5.1.

5.6 Implementation

We will define some of the terms to explain the implementation process better. The data broker receives one *file* every sixty seconds. We also denominate these files as *messages*. Each message contains a *body* field and within the body there

Table 5.1: Table with table names and number of columns of the tables in the database.

Table name	Number of columns
alm	156
e_meter	26
g_data_asset	9
g_data_sites_raw	6
meas	135
rw	11
scada	7
stat	128
tmiq	14
tx	47
weather	7

is a filed called *values*. Inside the values field, there is a list of data structures containing four key-value items that we are going to call *sensor reading*. So each tuple $(id, value, timestamp)$ is a sensor reading.

From the analysis of the syntax and structure of the *id*s, we generated the database design to store all the information from the messages. We created a database in Databricks, and we created all the eleven tables as Delta tables.

We used the same patterns of the *id*s to map the sensor readings to their corresponding table. We parse each id to get three pieces of information. We got the name of the table, the alias of the asset that can be mapped to their unique code and the sensor that is being reported. The parsing maps each id to a table in the database and also gets the column name.

5.6.1 Data Engineering

We created different scripts and mechanisms to keep the project updated with minimal manual intervention from an analyst to maintain the project's data. We created scripts to create and recreate the database and the tables of the project. This would be useful; for example, if we would require to move the project to a different environment, this script would facilitate the creation of the necessary data structures. Also, for testing purposes to test the solution in a new environment.

We created two different sets of scripts for the data ingestion. A bulk insertion script optimised to deal with large volumes of data. It can populate an empty database in minutes with all the messages stored in the storage account until that date. This script is optimised to transfer and load larger volumes of data.

We had a second script for incremental updates of the data. Once there is data in the database, we needed a mechanism to keep the database updated. This process would run every thirty minutes and takes all the last received messages from the data storage account and process them to insert them respectively into the database tables.

The bulk insertion could run after creating the data structures, making available a full analytical environment in minutes, ready to query the database. The bulk insertion process is efficient from the processing time point of view. It is possible to insert all the messages using the incremental updates script, but this is not optimised for large volumes of data. Both scripts are optimised for their main task.

We created a dashboard to monitor the data ingestion process. The dashboard uses a table that contains the number of messages received per day and their weight, as can be seen in Figure 5.4. The table with the data for the dashboard is updated once a day in the mornings. The number of messages received per day should be around 1,440 as the messages are transmitted every sixty seconds.

During the project, there were sporadic data gaps. Data would stop arriving at the data broker for different reasons. We could spot these data gaps using the data ingestion dashboard by looking at the number of messages received per day. We also implemented a proactive alarm to notify if we have not received new messages in the last 24 hours.

Identification of changes in the data

From the data exploration phase of the ids of the messages, we were able to classify and map each id to a specific column and table of the data model. This exhaustive process gave us the design of the database. But we are conscious that all the data available then *is not* all the data there is. We added a mechanism to the incremental update process to print out the unknown *id*s. We identify unknown ids as soon as they are parsed to send them to the database, but there is no proper data structure to support them.

Unidentified ids are presented in a counter dictionary to know how many instances of that id are in the data. The analyst then can consult with the business experts on the meaning of the new tags and if there should be new data structures or modifying existing ones. Those changes could be added to the next iteration of the project.

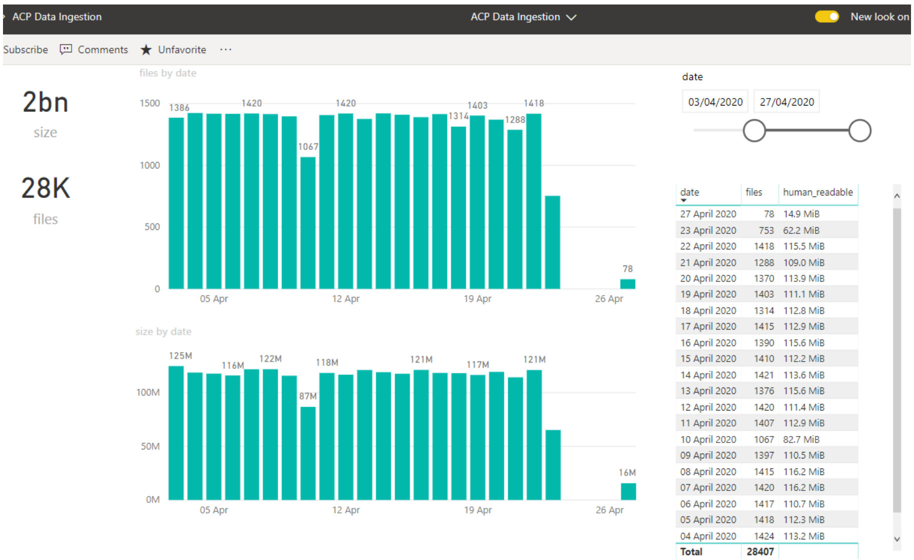
Data Profiling

We ran a data profiling script to understand the status of the tables and how they have been populated over time. We reused the code from the profiling tool described in Chapter 2. We used the results of the data profiling to improve the design of the database. The percentage of null values per column guided the re-design and vertical partitioning of the tables.

To determine thresholds for the proactive alarms, we run an exploratory data analysis shared with the expert users to determine the levels at which we should report alarms. We profiled the variables using statistics: mean, standard deviation, minimum, maximum, deciles, quartiles, and percentiles .1 and .99. We plotted the histograms of the sensors under different *running* conditions when the machine is running with load, when it is running without payload, and when it is on stand-by. In Figure 5.5 are histograms of the variables of assets running with a load.

5.6.2 Data Science

This section describes the implementation of the proactive alarms using the data from the analytical database designed for this project. We will explain how the



date	files	human_readable
27 April 2020	78	14.9 MiB
23 April 2020	753	62.2 MiB
22 April 2020	1418	115.5 MiB
21 April 2020	1288	109.0 MiB
20 April 2020	1370	113.9 MiB
19 April 2020	1403	111.1 MiB
18 April 2020	1314	112.8 MiB
17 April 2020	1415	112.9 MiB
16 April 2020	1390	115.6 MiB
15 April 2020	1410	112.2 MiB
14 April 2020	1421	113.6 MiB
13 April 2020	1376	115.6 MiB
12 April 2020	1420	111.4 MiB
11 April 2020	1407	112.9 MiB
10 April 2020	1067	82.7 MiB
09 April 2020	1397	110.5 MiB
08 April 2020	1415	116.2 MiB
07 April 2020	1420	116.2 MiB
06 April 2020	1417	110.7 MiB
05 April 2020	1418	112.3 MiB
04 April 2020	1424	113.2 MiB
Total	28407	

Figure 5.4: Power BI dashboard of the data ingestion process for the project. It shows the number of messages per day and their weight. We can see a gap in the data in the last days of April 2020 where data stop coming to the Datalake because of a problem in the project site.

proactive alarms were defined, a method to standardise the definition of proactive alarms, the algorithm to execute the checks for proactive alarms in the data, and finally, the workflow from when data is processed to the end when a proactive alarm is reported.

We added a table to the database to store the proactive alarms that will be reported. We also designed a process to coordinate the assigning unique ids of alarms. The coordination is necessary because there are other systems in the company that are reporting alarms to users.

The proactive alarms need to be reported to the Alarms Management System (AMS) of the company. The technicians will investigate and follow up on the development of the alarms from that system. The scope of this part of the project is to spot and send the alarms to AMS.

Alarm id allocation

The AMS requires a unique id for each alarm received. This is prone to id conflicts if more than one process generates proactive alarms. For this reason, we designed and created an alarm id allocation system that returns unique ids on request. We identified the alarm types we are generating and assigned a range of numeric ids per alarm type. This idea is based on the IPv4 address allocation [6].

We created a registry of alarm types used by the data science team. Each alarm type has assigned a range of ids between fifty and hundred thousand. There is a mechanism to allocate more ids for an alarm type if it is near its upper limit of ids. We created a function that receives the type of alarm and an integer that indicates the number of requested ids. The process will check if the alarm type is registered, return a list with the generated ids, and update a status table in the database with the last generated id. We thought of implementing the id request system as a REST API service. Still, all the alarms are working in the same database system,

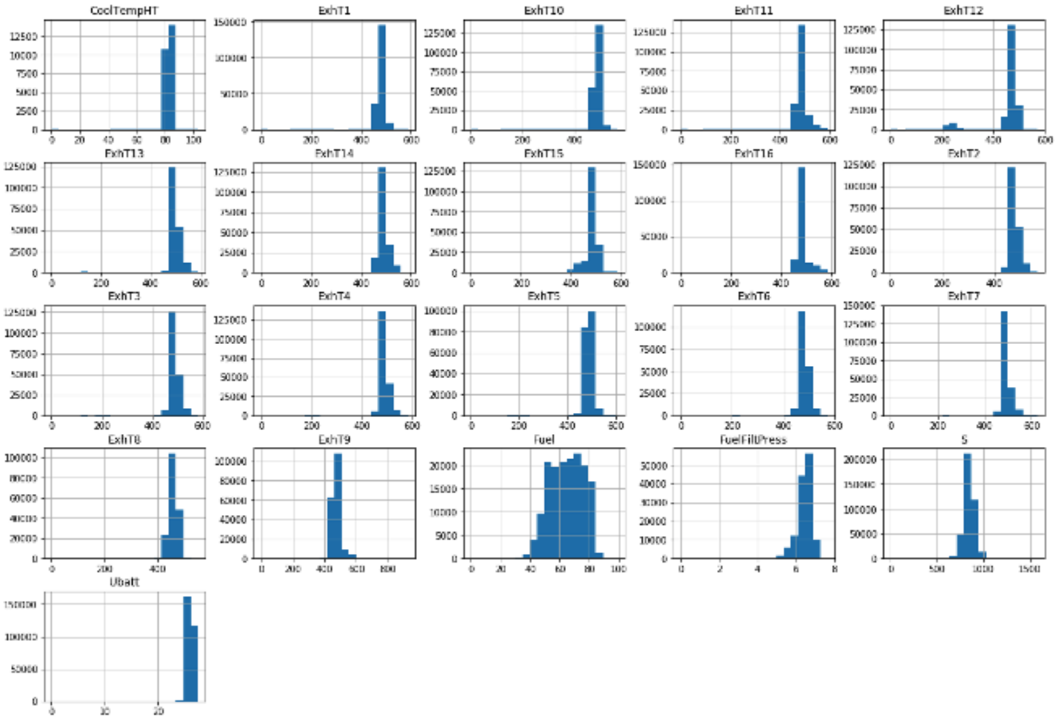


Figure 5.5: Histograms of 21 different sensors with data collected only when a machine is *running*. These results from the data exploration phase were shared with experts monitoring the workload of the machines to determine thresholds for proactive alarms.

so we decided to keep the id allocation system in the same environment and not add additional complexity in the maintenance of the alarm system.

Proactive alarm definition

We proposed a proactive alarm template to standardise the definition of proactive alarms. This is a proof of concept project, and there is no precedence for an artifact like this among the internal resources of the data science projects. The template facilitates the exchange of information between the expert users and the development team and makes the definition of the alarms available to anyone in the development team.

The fundamental elements of a proactive alarm definition include a high andor a low threshold value and a time. The system should inform the proactive alarm if the measured signal is above or below the threshold value for more time than the defined time span.

The following information defines a proactive alarm:

- *engine_type*: The assets are often groups by engine type. This way, we apply the alarm to all the assets with that specific engine type. It is expected that the thresholds are similar for assets with the same engine type.
- *customer_level*: A proactive alarm can be tailored to a specific customer in the case there is an interest to monitor a particular project site. It can be

defined for all the customers, or customers in a region, or specific customers. By default, the alarm applies to all the customers.

- *column_name*: This is the name of the sensor that maps to a column in the database.
- *low_value*: A lower threshold that is being monitored. It can be NULL if the alarm only uses the upper threshold.
- *high_value*: An upper threshold that is being monitored. It can be NULL if the alarm only uses the lower threshold.
- *gen_running*: The machines have three statuses. Running, stand by and turned off. For example, if the alarm is defined only for when the asset is running, it will ignore data when the asset is on stand-by.
- *units*: These are the units of the upper and lower thresholds. It is important because the sensor could be configured to transmit data using different units. For example, the temperature units could be sent in Celsius or Fahrenheit. This field is used for unit conversion if needed.
- *time_delay_seconds*: Minimum number of seconds to be checked to notify the alarm.
- *description*: The content from this field is sent as a message to the technician who will receive the notification of the alarm.

If the signal returns to normal levels below (or above) threshold, it reset the time counter that evaluates the time delay in seconds. The alarms will be notified only if all the signal values have been consistently above or below the threshold for more than the time delay in seconds.

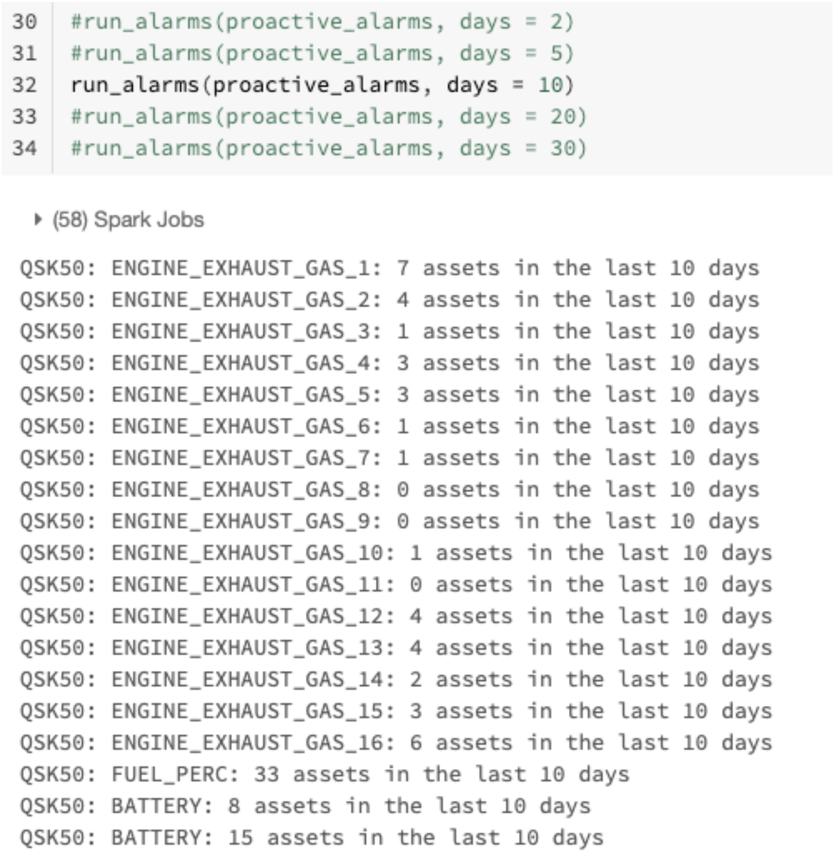
Proactive alarms

The expert users in charge of remote monitoring of the assets defined a list of 25 proactive alarms. By default, each proactive alarm is tested on the last two weeks of available data for that sensor.

The analyst can modify the time range used to evaluate the proactive alarms. For example, to run *what-if* scenarios and study how many times alarms could be triggered in the last n days. Or how many assets would have been activated the alarm using the current thresholds. An example of this can be seen in figure 5.6. This information could estimate the workload for the technicians receiving the alarms.

In Algorithm 2 we present the process to generate and inform the proactive alarms. There is a pre-processing phase to make the proactive alarm process efficient. We create a data set from the database with only the relevant columns, and we filter it by customer ids and by engine types required by the proactive alarms. Each proactive alarm is processed in a subset of the data specific for that alarm. An alarm checks all the assets of that particular engine in the same run.

Any proactive alarm should be reported only once. As the check for alarms uses the last two weeks of data, the process could spot the alarm again in the next run. If an alarm is already in the database, it will not be sent again.



```
#run_alarms(proactive_alarms, days = 2)
#run_alarms(proactive_alarms, days = 5)
run_alarms(proactive_alarms, days = 10)
#run_alarms(proactive_alarms, days = 20)
#run_alarms(proactive_alarms, days = 30)
```

```
QSK50: ENGINE_EXHAUST_GAS_1: 7 assets in the last 10 days
QSK50: ENGINE_EXHAUST_GAS_2: 4 assets in the last 10 days
QSK50: ENGINE_EXHAUST_GAS_3: 1 assets in the last 10 days
QSK50: ENGINE_EXHAUST_GAS_4: 3 assets in the last 10 days
QSK50: ENGINE_EXHAUST_GAS_5: 3 assets in the last 10 days
QSK50: ENGINE_EXHAUST_GAS_6: 1 assets in the last 10 days
QSK50: ENGINE_EXHAUST_GAS_7: 1 assets in the last 10 days
QSK50: ENGINE_EXHAUST_GAS_8: 0 assets in the last 10 days
QSK50: ENGINE_EXHAUST_GAS_9: 0 assets in the last 10 days
QSK50: ENGINE_EXHAUST_GAS_10: 1 assets in the last 10 days
QSK50: ENGINE_EXHAUST_GAS_11: 0 assets in the last 10 days
QSK50: ENGINE_EXHAUST_GAS_12: 4 assets in the last 10 days
QSK50: ENGINE_EXHAUST_GAS_13: 4 assets in the last 10 days
QSK50: ENGINE_EXHAUST_GAS_14: 2 assets in the last 10 days
QSK50: ENGINE_EXHAUST_GAS_15: 3 assets in the last 10 days
QSK50: ENGINE_EXHAUST_GAS_16: 6 assets in the last 10 days
QSK50: FUEL_PERC: 33 assets in the last 10 days
QSK50: BATTERY: 8 assets in the last 10 days
QSK50: BATTERY: 15 assets in the last 10 days
```

Figure 5.6: Summary of a what-if scenario checking the proactive alarms with a 10-day window of data. QSK50 is the code of the engine model of the asset, and the results show how many assets would present an alarm for each of the proactive alarms named after the monitoring sensor.

The way to report the proactive alarms is by saving a JSON file per alarm containing the information of the alarm. The JSON files will be named as follows *0_ps_vod_proactive.json*, *1_ps_vod_proactive.json*, ..., *n_ps_vod_proactive.json* where n is total minus one number of alarms to be reported.

An email is sent to a service account at the end of the proactive alarm process. That email triggers an Azure Logic App[10] that will collect the JSON files and transfer them to AMS. The proactive alarm process will delete any existing file in that storage directory before storing new alarms in the next run.

Predictive alarms

We started the work for predictive alarms based on the analytics database using Python libraries such as fbprophet [83] and sklearn [42], but COVID-19 issues halted the work during the implementation of the project. It was expected that the technicians that will receive the alarm notification would return to the office to start monitoring these new alarms. Still, the lockdown period extended, and then there were new priorities for the business team sponsoring this project.

[10] Official website of Azure Logic Apps, https://azure.microsoft.com/en-gb/services/logic-apps/ (accessed 24 August 2021).

Algorithm 2 Execution of alarms

```
Pre-processing: Generate dataset $D$ with the last two weeks of data and only with the columns of the alarms to be tested.
for alarm in proactive_alarms do
    generate evaluation dataset $D_{alarm}$ using engine_type and customer_level
    evaluate alarm on dataset $D_{alarm}$
    add last triggered alarm per asset in dataset $D_{alarm}$ to candidate_alarms
end for
for candidate_alarm in candidate_alarms do
    if candidate_alarm has not been reported then
        save candidate_alarm in new_alarms
    end if
end for
for new_alarm in new_alarms do
    generate json message
    save new_alarm to persistent storage
    save new_alarm to database
end for
```

5.7 Deployment

We deployed the proactive alarms process to a scheduled Databricks notebook that would run every thirty minutes. We tested the system, and it was ready to go into the production environment when COVID-19 issues halted the project. This also impacted the implementation of predictive alarms based on the data of this project.

5.8 Discussion

This project implemented the experimental design proposed in Chapter 3 three years before the starting of this project.

The end-to-end pipelines and architecture design for this project marked a successful implementation of a data science project using exploratory analysis techniques to inform the design decisions for the system better. The planning for this project considered the existing processes and infrastructure of the data science and data engineering team.

This project created processes and document artifacts to establish standardised ways to communicate and configure systems. This project made precedence for future projects in the use of good practices in data science, for example, the use of a standardised way to define proactive alarms or the unique id request system to coordinate the id allocation of alarm's id.

A logical step forward in refactoring this project is to include streaming analytics methods to run the alarms. Using the tech stack suggested early in this chapter or using Apache Spark's off-the-shelf techniques. This could simplify the implementation process of alarms.

An alarm that is never triggered does not necessarily mean that the asset has been running without problems. It could mean that the threshold is at a level that

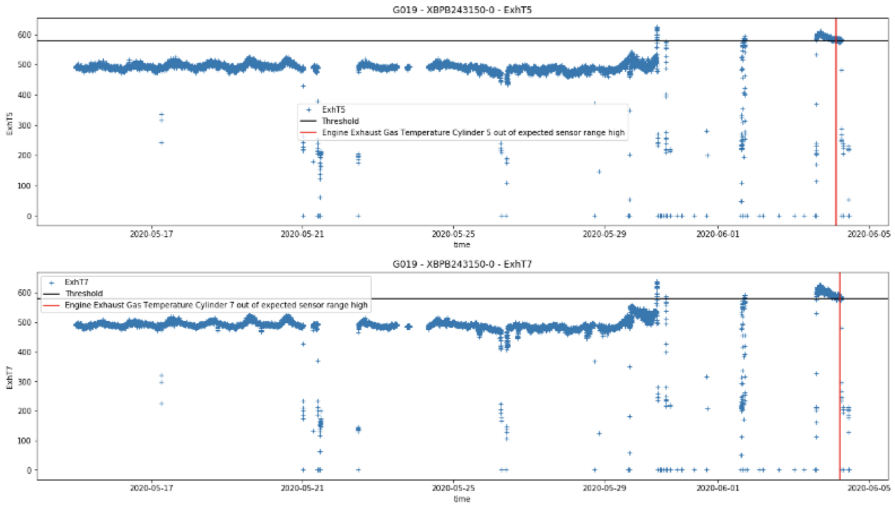


Figure 5.7: Proactive alarm identified for sensors ExhT5 and ExhT7. ExhT stands for exhaust temperature. The black line marks the threshold of the proactive alarm, and the red line marks the instant the threshold has been surpassed for more than thirty minutes which is the period defined for these alarms.

the values never reaches or that the signal is zero all the time. At the same time, an alarm that is triggered all the time does not necessarily mean that it is valid; this could lead to ignoring the warnings that are informed constantly. All these scenarios need to be tracked with a monitoring system for the alarms.

5.8.1 Further analysis

One of the analyses that can be implemented using the telemetry data is to calculate the right *sample rate* per asset. The sample rate determines every how many seconds a new measurement is collected and stored. The sample rate is a significant factor in the size of the data sets.

Two risks identified during the project are:

- The tags contained generic names or alias for the asset such as *G001* to identify one of the generators in the project site. It was required an additional mapping table with the aliases and their unique codes. This information could be standardised or passed in the messages from the project site.

- How to identify the project site from which the data is being received. This information was not part of the messages. If more project sites start transmitting data, knowing what assets are assigned to what projects will be necessary. This issue is related to the identification of the assets with their unique codes instead of generic aliases.

Chapter 6

Aggreko Data Engineering Library

Summary

The Aggreko Data Engineering Library (ADEL) is a Python custom package developed to handle everyday data engineering tasks. It encapsulates a set of utility functions for data ingestion from multiple database engines to the data lake. The scope of the library is the ingestion process from external sources into the data lake.

6.1 Introduction

The Data Engineering (DE) team is in charge of moving and transforming the data for the information systems to work inside the organisation. From the work of the DE team, the development of reports and dashboards by the Business Intelligence team and the analysis and machine learning models of the Data Science team.

The Data Engineering team builds data pipelines to move and transform the data from source systems into the data lake repository. The data lake is a central repository with a landing area for the data. It ingests data in its raw format and also computes data transformations and loads the data using a layered classification of environments. The three layers for data are bronze, silver and gold. A layer is another word for environment. The bronze layer has raw data from the source systems. The silver layer has cleaner and transformed data from the bronze layer. And, the gold layer has the cleanest data and is combined to serve reports and dashboards for the business. It is also the self-service layer of the system.

This chapter describes the implementation of a custom Python library of utility functions and operations to handle and transform data from the source systems to the raw and bronze layers of the data lake.

We start giving some context to the problem in the section 6.2. We continue with the description of the problem in section 6.3. The design of the solution is explained in section 6.4 and how we took different software development elements to work in the library. The implementation and development process are presented in section 6.5. Finally, the discussion of the project is in section 6.6.

6.2 Background

This project is in the data engineering field. To understand the context and many of the concepts mentioned in this chapter, we will present a brief history of the evolution of the information systems focusing on the data engineering techniques used over time and defining many technical terms used in this chapter.

6.2.1 A brief history of the information systems

Since the *triumph* of the entity-relationship model [1] [2] over the network or graph and set entity representation of data in the '70s, there has been an increasing desire to store and analyse more and more data in relational databases by organisations. Early on, companies made a distinction between operational and information environments. Operational environments have systems that support the daily operations of the businesses. Information environments have *copies* of the data (from the operational environments) to be analysed by analysts.

Relational databases were the preferred way to store data, and in the 80's we saw apparition of operational data stores (ODS) [12]. ODS were a copy of the operational environment. The copies were necessary to avoid a performance impact on the operational systems and allow business people and analysts to query and analyse their operations' data. Of course, the analytical power was limited to the design of the operational data model.

In the 90's, the Data Warehouse (DWH) [14] [11] was, in some sense, an evolution from the ODS. The DWH had a structured way to collect and analyse data. It also created the necessary process for data collection and data transformation to maintain the DWH updated. The data warehouse provided a stable and accessible solution for data analysis. It kept historic data, often the last five years or more, giving access to business users to query the data. The data warehouse data model is optimised for fast query answering; it is more efficient in retrieving data from the database than inserting them. The data in the DWH is highly structured and it is transformed before being loaded into the DWH. Not all the operational data make it to the DWH, only defined columns of specific tables relevant from pre-defined analysis.

With the rising of DHWs we saw the rising of the *ETL* techniques needed to *extract*, *transform* and *load* the data from data sources to the DWH. ETL is a term to wrap up the data movement tasks used to feed a data warehouse. Interestingly enough, there were data transformations before the DWH, but those were called just data processes or data movements and worked with scripts.

In the 00's we saw the apparition of massive volumes of data. For the first time, organisations were able to generate insights from vast volumes of data. The release of Apache Hadoop based on papers from Google engineers [25] [34], and later Apache Spark [45] made it possible to process massive volumes of data using clusters of commodity machines enabling the world of *big data* that is known to us today.

In 2010 a new concept was added to the ecosystem of data technologies, the Data Lake. Data lakes are enterprise-wide data repositories that ingest the operational systems in their raw formats. Data lakes have layers to access the data, refining it and combine it with the different systems. The top layer of a data lake will have an integrated version of the data from multiple systems in reports, dashboards, and ad-hoc analysis to generate insights. Data lakes use data pipelines to collect and transform data. Data pipelines are to data lakes what ETLs are to data warehouses.

A main difference between the data warehouse and the data lake is that the latter includes the data without transformation in their raw format. Storage space is cheap nowadays, so this is the way to have timely access to operational data. Data in a DWH is highly structured, and the ETL process is designed and structured. In contrast, the data ingestion processes for the data lake are adapted per data source and often occur with change data capture technologies (CDC).

Later in the 10's, we saw the use of new techniques in data engineering to catch updates of operational databases. Change Data Capture (CDC) is now broadly used to detect updates in the data sources. CDC will be reading the log of a database, and it will interpret the changes that the database reports to the database log as the events, and it will inform and sent the changes to the target database.

We see in the future hybrid approaches and multitenancy, using more than one cloud provider, data stores.

The progress in information systems over the years is that each new technology that is incorporated into the ecosystem of existing tools, and sometimes replacing some of them is that it *abstract away* the complexities of the processes and offers a more straightforward way to execute the same tasks. That was the case of the ODS, the DWH, the data lake, Apache Hadoop and Apache Spark, and CDC tools. They took a process that was doable but complex. They encapsulated away from the complicated parts for the developer to use off-the-shelf functionalities that before was limited to the development process.

The design and implementation of a data engineering library pursuits the aim of abstracting complexities and offering a straightforward way to execute data transformation tasks for the data engineering team.

6.3 Description of the Problem

When the new enterprise data lake environment of the organisation started to get populated with data from multiple data source systems, the data engineering team followed a repeatable process to create the data pipelines to keep the data lake updated from the data sources. There would be at least four notebooks with code for each new data source to be incorporated into the data lake, containing data transformations for the bulk and incremental data load.

The data sources are operational systems of the organisation. Each operational system has its database. To make the information from operational systems available to the users, it is required to extract the data from their internal databases and put them into the data lake. The databases are in multiple relational database management systems (RDBMS), i.e. Microsoft SQL Server, MySQL, Oracle, etc. Also, there could be various versions of an RBDMS.

The data transformations in the notebooks used SQL and Python language. The notebooks are separated by the domain of the data transformation tasks. First, a notebook to create the database and tables. A second notebook for a bulk load of the data. The third notebook for change data capture. And the fourth notebook for the incremental load of the data.

As these notebooks started to repeat for each data source system, we foresaw that maintaining them could become a problem. Any modification would need to be applied to each notebook of the same type. The effort to maintain the code grows linearly as a new data source is added.

The copied code to maintain a new data source would often require minimal modification. Sometimes, the data source would require a customised data transformation, and in those cases, the code is added to the particular notebook that handles that data transformation. This is a reason to keep the code separated by the source system and not having a general notebook for all the systems.

Creating similar code for each new data source system triggered the consideration of creating a custom library that could provide off-the-shelf functionality and be used and extended for more complex tasks.

The question for this project is: *can we design and implement a custom library that can handle the data ingestion process from multiple database engines into the data lake, abstracting the complexities of the process?*

6.3.1 Benefits of a custom Python library

Having a custom library would reduce the amount of code to be maintained. Reducing the chance of introducing human error into the data transformations and making the development process more efficient. Encapsulating reusable code minimises the repetition code so that any change is introduced once in the code and not maintained multiple times.

The analyst has more control over the development process and can establish a workflow to add new functionalities and maintain current ones. All the changes to the code can be traced and reversed if needed.

Testing is an essential part of software development. Still, when the code is in notebooks in a cloud environment, the testing is often overlooked because there is no simple way to test the code from notebooks. It is necessary to incorporate testing into the code to ensure its quality and maintain the system in live or production environments.

6.4 Design

In computer science, designing a custom library is a software development task. The library will be used for data engineering processes, but the implementation will be treated as a software engineering endeavour. For this reason, we looked into software engineering modelling techniques for enterprise applications using data.

In terms of requirements for this EngD project, the data transformations of the data engineering team were implemented in Azure Databricks and using the Python language.

We looked into three approaches to design and model the library and took elements from each one to model and implement the library. 1) Test-driven development (TDT) is a common technique used to design systems that handle data. 2) Event-Driven Architecture is an architecture design that uses the data flow in the system as input for the components and design. And 3) the Domain-Driven Design (DDD) approach that leverages the modelling around the knowledge about the problem. It highlights the relevance of a shared language for modelling.

An event-driven architecture is described in Chapters 3 and 5 where the design of the architecture was informed by how the data would be arriving and later processed in the system. In this case, it doesn't make sense to use this approach as the library is intended to support data engineering tasks, meaning that it would enable an event-driven architecture. Still, the approach does not make sense to *implement* the library.

We used the guidelines from DDD for the design. We added testing to the library, but after the functionality is implemented and not prior to writing it as is recommended by the TDD practices.

We decomposed the activities into three main groups to organise and design the library—first, we analysed the existing code in notebooks and its functionality. Second, the code structure of the repository. And finally, the team coordination to collaborate in the development of the library.

We used an iterative process with brainstorming sessions to check early library designs and where to incorporate the required functionalities of the library. We used the shared language to distinguish the different components and features. We established a list of what is needed to be implemented, and we followed it in sequence, implementing modules of the library one by one.

The library is intended to be used primarily in a Databricks environment but this is not a constraint for the library to work in any other environment.

6.4.1 Analysis of the notebooks

We analysed over 90 (ninety) notebooks with data transformation code for 12 (twelve) different operational source systems. We identified the different types of notebooks and the different types of tasks within the notebooks. We can classify most of the data transformation tasks into one of these four groups:

1. Creation of the database and tables.
2. Bulk load of the initial data.
3. Append new data from the source.
4. Merge data from the source.

The review of the notebooks led to the design of the library in a generative approach. We wanted to implement all the functionalities that are already provided using the notebooks, plus the flexibility of adding new functionalities without impacting existing working code.

6.4.2 Team coordination

This project would have contributions from three members of the team and more in the future as new features are required. Communication and coordination is key to the success of the project. Early in the project, we defined how to collaborate, establishing the tools and libraries for the project and defining the git workflow for development. We had hands-on sessions to show and practice the implementation and development process and document them.

We created a git repository for the library, and we used Python as the default programming language. The repository contains the library's source code, the code for testing, and documentation about the inception and design of the library.

We created a second git repository to emulate the file structure of the file system in the clusters. This would facilitate the testing of the library. The *adel-test-data* repository contained samples of the files that we would find in the cluster's file system.

We held daily calls in the mornings during the first weeks of implementation and then weekly calls to update the status of the library.

6.4.3 Domain-Driven Design

In this approach, *domain* refers to the context and knowledge around the problem where the problem. DDD uses the knowledge of the problem as a base to model the solution [26]. Our *domain* was data transformation tasks performed to ingest data into a data lake. The DDD process recommends cultivating a shared language so that the system can be modelled using plain English associated with the technical terms and functionalities in the project's scope.

We established a common language to facilitate communication and implementation. A*ubiquitous language* helps to translate domain terminology into code.

The terms we use must be unequivocally referring to a specific element in the data ingestion process. As part of the shared language, we defined two main terms: *data source* and *data engine*:

- **data source**: Refers to an internal operational system that is used to support a business process.
- **data engine**: Refers to the database management system (DBMS) where a *data source* is hosted.

These terms are embedded in the library as sets of modules that can handle *data sources* and *data engines* as they were defined. Each *data source* will have a Python module in the library, and each *data engine* as well. A Python module is a file in the library with the extension *.py*.

The standardisation of the language contributes to having better communication among the developers. There were three people involved in the development of this library. We used DevOps tools for code versioning and agile development.

We took a pragmatic approach to the implementation. We made sure that what was requested worked well, and we cut the additional functionalities *nice to have*, but that was not in the requirements. This way allowed us to deliver quality software that did what was meant to do.

We used a generative approach to the design. From the review of notebooks, we documented and created a list with requisites for the library.

One of the issues at the time of the modelling and then in the implementation was that the development would happen in local computers. In contrast, the testing and actual use of the library would occur in clusters in a cloud environment. This affected the way to test the library as some of the functionalities were meant to use local resources of the clusters. We implemented some strategies to work out these issues. We created a process to generate the library as a file to be installed in the clusters. Early on, we connected the local development environments with the clusters using the *databricks-connect* library that later on was advised no to be used, so we replaced it with the *pyspark* Python library.

We followed the single-responsibility principle (SRP) in software development [79]. The SRP dictates that every component in the software, module, class, or function should have only one responsibility. This was relevant because it solved some of the design questions we had. For example, the same function would need to be implemented for different data engines; should we use a unique function for the task, or each data engine module should have the function implemented. Using the SRP, we decided to write the function on each module to solve any particular issue locally and not affect other data engines.

We suggested a *functional programming* approach to developing the library following the SRP making use of a ubiquitous language for the project. We can naturally map the data transformations to be implemented to functions; this makes the functional programming approach sound.

As in any project requiring a design to be used in the future, requirements and needs are expected to change over time. It is essential not to commit to a specific design early in the lifetime of a project. Early design decisions often turned out to be very expensive to maintain over time. The library's design approach was based on quick iterations of ideas to select the best design that adjusts to the needs and offers a way to extend the library's functionality at the same time. Early decisions on the design can compromise the functionality of the software.

Based on the experience of the Manning Optimisation project described in Chapter 7. Quick iterations of mock-ups and prototypes with users can provide insights that otherwise would be more difficult to get. For example, know how we will use the tool and queries, what aspects are more relevant to the user and get honest feedback on the tool's usability.

This project is based on the experience of developing open source libraries [93].

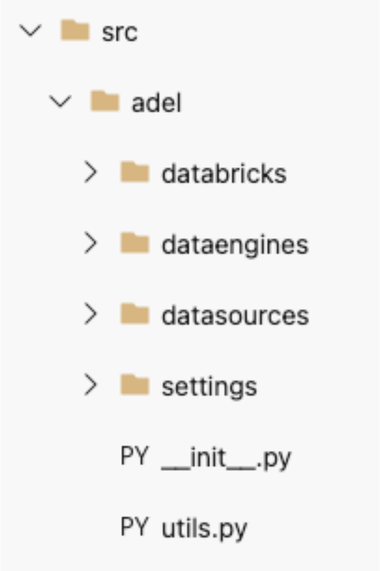
6.4.4 Code structure

The design of the library follows a pattern separating the code in three main modules: *databricks*, *datasources*, and *dataengines*. *databricks* corresponds to code to interact with Databricks' REST APIs[1] to interact with *clusters*, *libraries*, and their *file system*. The modules in *datasources* are created to handle each one of the internal systems, not their database systems but the business processes related tasks; for example, a data source system that has telemetry data on it some functions will need to be created to transform the units. There is one module per database management system in the *dataengines* component.

The breakdown of tasks then corresponded to one of the previous activities per data source or data engine. So they can be developed independently with minimal impact on the overall progress of the library.

The *data source* modules include features like getting the files with data to be ingested to the data lake, deduplication of the data that is going to be loaded, functions to populate and incremental updates of the tables in the first layers of the data lake, the raw and bronze layers.

[1] Official documentation of REST APIs, https://docs.databricks.com/dev-tools/api/latest/index.html (accessed 03 September 2021).



```
src
  adel
    databricks
    dataengines
    datasources
    settings
    PY __init__.py
    PY utils.py
```

Figure 6.1: Structure of the code in the library.

The *data engine* modules includes functionalities for five different DBMS. The features include parsing of .sql files with table structures from the data engines to create the data structures to ingest the data to the data lake, data type conversions between the data engine and the data lake table format and vice-versa, among the specific functions for each data engine.

The library has a configuration file to use parameters such as the cluster names, database names, environment names, paths in the Databricks file system and other relevant information. The option of using a table in a database was discussed and could maybe be implemented, but we are not fond of this option as it relies on an external source. This means that if the database with the configuration settings is not available, then the library will be unusable.

We added a *utils* module where we put transversely helpful utility functions. For example, functions to read in the configuration file or to change time formats. *utils* handles shared tasks among the different data sources and data engines.

We modelled the library using functions rather than an Object-Oriented approach. This approach facilitates the reuse of code and models behaviour using the arguments of the functions. We are including typing annotations to the variables used in functions. We considered that data transformation tasks can be easily translated to the implementation of functions to perform a specific data transformation task or decompose into smaller functions.

6.5 Implementation

As more than one person would be adding functionalities to the repository, we needed to organise the development process by defining the git workflow that we would follow.

The git workflow consisted in branching out from the *develop* branch. We established the *main* branch as the official version of the library. We would not modify it directly but only through updates to the *dev* branch. We created a development branch called *dev*, which we would update by creating pull requests (PR) from feature branches. We would need to create feature branches with descriptive names depending on the functionality to develop any new functionality. We created a work item for each feature that we wanted to implement in the library. This way, we can track the development process using the Azure DevOps platform, where also the repository is hosted. We created unit tests for the functionalities that we implemented and then created a pull request from the feature branch to the *dev* branch to merge our code to the library. We would merge the *dev* branch into the *main* branch less frequently, and once the *dev* branch contains enough updates to update the library unless we had to make a hotfix which we would merge as soon the fix was available.

We created two git repositories. One called *adel* for the source code of the library and one for the test data called *adel-test-data*. The *adel* repository contained the source code for the library, and we created a separate repository with testing sample data for the library. Often, the data for testing is added to the same repository of the library, but we decided to separate the repositories for two reasons. We wanted to make a lightweight library to deploy it to clusters, and we wanted to replicate the file system organisation in the clusters using sample data.

We added a *CONTRIBUTING.md* document to the documentation with development guidelines to facilitate the contribution of more people to the library. The guideline contains several sections describing the integrated development environment (IDE) Visual Studio Code[2] and how to use it. We choose this IDE for its integration with Azure, Microsoft's cloud platform, where the platforms we were working with are hosted. A list of the Python libraries used for development such as black[3], autoflake[4], pipenv[5], pytest[6], and databricks-connect[7]. The description of our git workflow where we are committing changes to the *dev* branch and creating issues for each piece of code that we are working on—the instructions to add new modules to extend the library's functionality. Instructions of how to setting up the virtual environment to work locally in the development for then push the changes to the remote repository. The documentation also contained the most common and used git commands for familiarity. Instructions on how to run tests to the code. How to install and deploy the library in Databricks.

We used a versioning system following PEP 440[8] with three digits separated by dots indicating minor changes, significant changes, and breaking changes with the release of the custom package.

[2] Official website of Visual Studio Code, https://code.visualstudio.com (accessed 30 August 2021).

[3] Official GitHub site of the black library, https://github.com/psf/black (accessed 31 August 2021).

[4] Official GitHub site, https://github.com/myint/autoflake (accessed 31 August 2021).

[5] Official documentation of pipenv, https://pipenv-fork.readthedocs.io/en/latest/ (accessed 31 August 2021).

[6] Official documentation of pytest, https://docs.pytest.org/en/stable/ (accessed 31 August 2021).

[7] Official databricks-connect documentation, https://docs.databricks.com/dev-tools/databricks-connect.html (accessed 31 August 2021).

[8] Official documentation of PEP 440, https://www.python.org/dev/peps/pep-0440/ (accessed 31 August 2021).

6.5.1 Testing

We tested the library using the *pytest* [27] Python library. We created unit tests for each module. The *adel* library is installed locally and tested. We followed good practices for *pytest* [74].

6.5.2 Deployment

The library must be installed in a Databricks cluster to be used. The functionality to deploy the library to any Databricks cluster was added to the code using the REST APIs of Databricks.

Installing the library in a cluster makes the library available to be used from notebooks so that the analyst could write `import adel` to use it.

At the beginning of the project, we had a manual process to deploy and install the library. We would generate a wheel file (.whl) from the library, and we would manually upload the file into the Databricks cluster and then installing it from there to make it available for the notebooks.

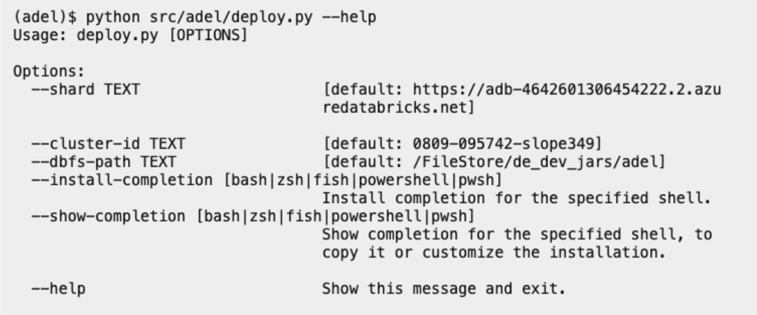
Later on, we evaluated alternatives to deploy the library to the clusters. We could use an Azure DevOps pipeline, which means adding a .yaml file to the repository triggered when we push changes to a specific branch as part of the pipeline to deploy the library to clusters. Or, we could deploy the library from the local environment using the REST APIs of Databricks. We implemented the latter, and the former is on the list of features to be implemented. With the deployment from the development environment, we have more control over which clusters the library is deployed for development, testing, or production environment.

We used three REST APIs from Databricks to implement the deployment process: the *clusters API* that handles the start and termination of clusters, the *DBFS API* that handles the Databricks files system with calls for creating directories, deleting, moving, copying or reading files, and so on, and the *libraries API* to install and uninstall libraries into the clusters.

The deployment process of the library is as follows:

1. Start the cluster and wait until it is running (clusters API).
2. Uninstall any previous version of the library (libraries API).
3. Restart the cluster and wait until it is running (clusters API).
4. Clean the dbfs-path where adel is should be copied (DBFS API).
5. Upload the latest local .whl file to the dbfs-path (DBFS API).
6. Install the library in the cluster (libraries API).

We added a deployment script that we can use from the terminal. Deploying the library becomes a matter of using the terminal locally as can be seen in the Figure 6.2 using three parameters: *shard*, *cluster-id*, and *dbfs-path.*



```
(adel)$ python src/adel/deploy.py --help
Usage: deploy.py [OPTIONS]

Options:
  --shard TEXT                    [default: https://adb-4642601306454222.2.azu
                                  redatabricks.net]

  --cluster-id TEXT               [default: 0809-095742-slope349]
  --dbfs-path TEXT                [default: /FileStore/de_dev_jars/adel]
  --install-completion [bash|zsh|fish|powershell|pwsh]
                                  Install completion for the specified shell.
  --show-completion [bash|zsh|fish|powershell|pwsh]
                                  Show completion for the specified shell, to
                                  copy it or customize the installation.

  --help                          Show this message and exit.
```

Figure 6.2: Deployment of the library from the command line. It receives the *shard*, the *clusted-id* and the *dbfs-path* as parameters.

6.6 Discussion

More than a final finished product, this project established a process to develop and further expand a data engineering library. We set the foundations to collaborate and contribute with functionalities, as we understood that this would be a learning process and new functionalities would be required in the near future.

By setting up the code structure and design of the modules, the testing process and the development process, we enabled the data engineering team to maintain and use the library.

The library design followed good practices from software engineering methods suitable to the requisites for this project. The process to work on the library follows best practices from the open-source community.

We based the design around a common language and knowledge about the data transformation processes. We followed the recommendations of the single responsibility principle to implement the functions that will handle the complexities of data transformations. We modularised the design with the distinction of data sources and data engines; we added modules to handle the REST APIs with the data lake's computing platform. We added a settings module to configure the use of the library and adding environmental variables. We added testing to the development process. And we documented contributing guidelines and the deployment of the library.

A library is a piece of software that helps to build other pieces of software. It is a building block for robust and stable code. A good library design should abstract the complexities of the development and offer a more straightforward way to implement the functionalities than doing them directly. The key to usability and design is the right level of abstraction for the data transformation tasks. We achieved a sound design by using a shared language that can translate to the implementation of functionalities.

Part II

Applied Data Projects

Chapter 7

Manning Optimisation

Summary

This project estimates the number of technicians needed to solve unplanned events on machine on a project site, given its characteristics such as the number of machines, their capacity, the air quality in the location and fuel quality.

7.1 Background

There are often two types of projects that use the machines of the organisation. One mode is short-term projects where the generators are hired for days and maybe months, and the other is long-term projects where generators and other machines are allocated for months and sometimes years to a project site location. The Rental Solutions line of business manages the long-term projects. When a long-term project is agreed to be executed, one of the necessary tasks is to estimate the number and qualification of the people that will provide support and maintain the hardware deployed on the project site. A forecast model was introduced in 2016 to estimate the number of people needed to maintain diesel machines. That model created by business analysts is the starting point for this EngD project.

The model estimation created in 2016 was based on the number and size (in Megawatts) of the machines that will be used on the project site. It is expected to prevent failures in the devices, and in the eventual case of failure, it is expected to have qualified personnel to solve the issues on site.

There are two types of asset maintenance: *planned* and *unplanned*. *Planned* maintenance is a well-known event where a service or procedure is applied upon the asset and are scheduled based on the number of running hours of the assets or other metrics. On the other hand, *unplanned* maintenance is an unexpected event of failure or similar kind of issue that often requires immediate attention to continue with the operation of the asset.

The maintenance tasks of the machines are recorded in the database as service orders. There are specific codes to identified planned and unplanned service orders.

The project stakeholders built an estimation model in 2016, a couple of years before the start of this EngD project. It was built using surveyed data from the technicians of projects in South America. The estimation model was built on a standalone spreadsheet. This format of distribution of the estimation model could be challenging to maintain and share among multiple users. The estimation model was built based on data from six projects, and on the surveys of around ninety technicians that reported estimation times to specific maintenance tasks. With this data, the team generated a mixed regression model considering the air quality, the fuel quality, as factors for the model plus the number of machines using diesel fuel on the project sites.

The estimation model created by the stakeholders applies to a subset of engine models that use fuel diesel to operate. Other devices use gas, and there are hybrid machines that use both. Diesel engine models are widely used in many projects, and that's why we started with these engine models.

In this chapter, we will use the term *solution* to wrap up the processes to deploy the forecast model serving requests from client applications. The *solution* is the forecast model deployed and working properly. We also will use the terms *estimation model*, the *forecast model* and *headcount model* interchangeable.

7.1.1 The current model

The forecast model estimates the number of unplanned maintenances per asset per week. That number is multiplied by the average number of hours required to solve the task according to the surveyed data from the technicians' responses.

The surveyed data from technicians contained the estimation of times required to solve different types of issues with the assets. The collected data was tabulated to compute average times to solve the different types of unplanned maintenance. This tabular data was added to the spreadsheet and queried using *vlookup* calls.

The model was based on data collected from six projects in South America from 2016. Estimating the number of hours is computed based on a survey sent to technicians to report their best assessment of the time needed to solve the issues surveyed.

The current model estimates the number of electricians, mechanics, and general crew needed for the project site. It considers factors such as the air quality and fuel quality on the project site and whether or not the project requires personnel to be available 24/7.

7.1.2 The Regression Models

The forecast model uses three regression models showed in 7.1, 7.2, and 7.3 trained with surveyed data from a sample of project sites.

The Excel file with the forecast model contained 22 sheets with tabular data and calculations to compute the estimation. One of the sheets was used as the user interface interacting with the models. The user could input parameters with project data on the top part of the spreadsheet and then see the results in the bottom part of the same spreadsheet. The results were calculated using nested formulas in the Excel document. The formulas jumped between sheets to get the calculations and tracked them all to understand the workings of the current forecast model.

The model uses four input variables:

- Rh/Set/Day: Running hours per set or device per day.

- SMR: Service, maintenance, and repair. Number of hours to schedule a regular maintenance.
- Redundancy: A percentage of the total power provided on the project site.
- Megawatts: The total amount of Megawatts that needs to be supplied by the project site.

Additionally, the user can input other parameters that are used as factors to present the results. The other parameters are:

- Air quality: Three levels: high, medium, and low.
- Fuel quality: Two levels: high and low quality.
- Redundancy: Percentage of the size of the project in MW that need to be additionally ensured to be present in the project site.
- 24/7: Whether or not the project site requires people working twenty four hours per day, seven days per week.

The first model 7.1 estimates the number of unplanned service orders number ten (SO10) that a piece of equipment will have per week. Where x_1 is Rh/Set/Day, x_2 is average SMR, x_3 is redundancy, and x_4 is the number of Megawatts hired.

$$\begin{aligned} SO10PerSetPerWeek = &-0.1075 + (0.00626 * x_1) \\ &+ (0.000020 * x_2) + (1.789 * x_3 + 0.001883 * x_4) \\ &- (0.000000 * x_2 * x_2) - (0.000076 * x_2 * x_3) \\ &- (0.0430 * x_3 * x_4) \end{aligned} \tag{7.1}$$

The second model 7.2 estimates the number of mechanic hours needed per service order.

$$\begin{aligned} MechanicsHoursPerSO = &-0.862 + 0.000167 * x_1 + 0.0289 * x_2 + 32.8 * x_3 \\ &- 0.0986 * x_4 + 86.0 * x_3 * x_3 + 0.000005 * x_1 * x_2 \\ &- 0.002932 * x_1 * x_3 - 1.170 * x_2 * x_3 \\ &+ 1.967 * x_3 * x_4 \end{aligned} \tag{7.2}$$

And the third regression formula 7.3 estimates the number of service crew hours required per service order. Service crew help with the operation of the project site but they are not highly specialised like the electricians or mechanics.

$$\begin{aligned} ServiceCrewHoursPerSO = &\ 0.677 + (0.0000016 * x_1) \\ &- (0.0014 * x_2) + (30.15 * x_3) - (0.2204 * x_4) \\ &+ (0.00707 * x_4 * x_4) + (0.000005 * x_1 * x_2) \\ &- (0.001189 * x_1 * x_3) - (1.143 * x_2 * x_3) \\ &+ (0.825 * x_3 * x_4) \end{aligned} \tag{7.3}$$

7.1.3 Benefits of an estimation model for manning optimisation

The limitations of a model in an Excel document are many. The need to make multiple copies of the model where each copy can be prone to introduce human error, generating different results for the same input data. It makes it difficult to update the model given that there will be multiple versions of the Excel document across the organisation. For a big organisation, this will introduce a new challenge to make sure every person uses the latest version of the spreadsheet.

The benefits of the contributions of this project are many; they are operational, financial, and reputational among others.

On the operational benefits, we can increase the forecast accuracy by using up to date data rather than static data sets. Also, we will add easiness to use and accessibility to the forecast model.

On the financial benefits, we aim to balance the ratio of *rotators* versus *local staff*. Rotators are technicians that go from one project site to the next and they are not assigned to a location for the full duration of the projects. Still, they travel based on the issues that appear on particular project sites. Local staff are people hired locally for the specific project site. We aim to optimise employee allocation. This would drive efficient cost savings for the organisation.

A forecast model like this one would help bid managers at commercial bids to right-size the budget entered for the manning of the projects.

Finally, we aim to prevent under and overstaffed sites.

7.2 Problem Statement

The question we try to answer is *can we implement and deploy the current model to site managers securely, providing the same information to each user and simplify the process of update or retrain the model based on new data and information?*

7.3 Planning

The EngD project was planned in three phases given the requirements of operationalising the existing estimation model and then updating it using up to date data.

The first phase was about making the existing model in a standalone spreadsheet, enterprise-wide available from a single user interface so that every user could have access to the same version of the model. The second phase would be about enhancing the model using up to date data and using all the data available in the databases of historic project sites. The original model had access only to a subset of the project sites data. The third phase of this project, a new project, would be to consider an estimation model for multiple project sites. So nearby project sites could rely on shared resources and technicians.

In phase one of the project, we looked at the available data sources and explored them enough to choose what data sources we could use. We also created a layered architecture to deploy this model in an application connected via API that was containerised and deployed in the cloud service of the organisation.

In phase two, we validated and checked data sources to update the model, but we did not deploy a new model. We verified the feasibility of training a new model using the data sources available at the time to update the current formula. This was done during the implementation of a enterprise-wide data lake so most of the data sources would be incorporated to the central repository in the coming months but at the time of the second phase they were not available on-demand for analysis.

This chapter reports on phases one and two. Phase three has not been planned at the time of the writing of this chapter.

1. Phase 1: Automation of current the Excel process.
2. Phase 2: Enhancing the current model with more up to date data sets and insight.
3. Phase 3: Implementing a solution that optimises the calculation for multiple sites.

7.3.1 Methodology

We maintained close communications with the stakeholders of the project. We held weekly meetings with the stakeholders where we would present early results and ask questions about the implementation and expectations of the project. We first explored the data sources and reported the findings. We built rapid prototypes and mock-ups to show the users how the interaction with the model would be using a web interface. This iterative process led to a modular design of the architecture and it the user interface was validated early in the project by the stakeholders. The user interface of the application was adjusted for user interaction with the recommendations from the stakeholders. We deployed an online mobile application that final users tested.

We created a git repository for phase one of the project. We wrote documentation about the business process. We created a knowledge base for the project so the subsequent phases could leverage the insights generated from previous stages.

We developed the solution using an agile approach. We presented early results and the progress of the project in weekly meetings with the stakeholders so we could include their feedback in the planning for the next week's tasks. We built two functional mock-ups that the user reviewed. With the feedback from these early prototypes, we could design the final user interface for the application.

The first mock-up was a monolith application using Python. An HTML page we created with boxes to collect the input the parameters of an on-site project. This helped to design the response from the estimation model and the user interface to present the results. The second mock-up implemented user feedback; the user interface was developed with PowerApps, which helped to finalise a layered design and use corporate colours in the user interface in the final version and separating the layers of the interaction of the final solution.

The deployment of the model was automated by creating a trigger that will deploy the model to Azure into a container every time a change is pushed to the *main* branch of the git repository of the project. The time between phase one and two was around eight months apart.

7.4 Design

The objective of the first phase was operationalising the existing estimation model. Once current model was deployed, we wanted to use up to date data to retrain a new model to forecast the people needed to support a project site. We wanted a modular design that would allow, for example, to replace the existing model with no impact on other system components. Ideally, the estimation model could be replaced by a new version without changing the solution's user interface. In the future, we would need to upgrade the model, and modularising the design makes the maintenance tasks and further development simpler, as the components are replaceable and upgradeable.

This project aimed since its conception to have a modular design. The design considered a microservice architecture implemented on containers. The unplanned failures estimation model can be served from a container. We used a lightweight web server enabling an API to communicate with client apps that would request a manning estimation. The front-end was implemented using PowerApps.

7.4.1 Software Architecture

There is a wide range of options to borrow from software engineering practices. We looked at some software design patterns to design our solution. One of the most common software architecture used is the *layered architecture* also known as tier-architecture. It is a good starting point for us because it allows the separation of the functionality of the system in horizontal layers that interact with each other via interfaces. It is good to represent business logic, persistent storage, and databases working together.

We combined the *layered* design with another enterprise architecture, the *microservices architecture*. The idea behind a microservice architecture is to separate functionality into components and deploy each one of them as a separate unit. A microservice architecture increases the scalability of the solution while decouples the design of the solution. The challenge with a microservice architecture is to determine the right level of component granularity, what task a component should handle, or if it should be decomposed into a more granular level of components.

For this project, we used both approaches to design the final solution. A layered architecture helps to identify and implement the functionality of the solution. Additionally, a microservice architecture allows for scalability of the deployment of the solution to almost any cloud provider using manifests for the deployment.

Other architectures we explored were the *event-driven architecture* (used in Chapters 3 and 5), *microkernel architecture* used to design real-time systems, and *space-based architectures*. The utility of these architectures were less clear in the context of this project.

We suggested a three-layered architecture with 1) the estimation model in the back-end, 2) a REST API in the middle to communicate with the model, and 3) a front-end with the user interface. The estimation model would be deployed as a service (or microservice) so that any application could connect and interact with the back-end model using its API.

A layered architecture is a level of abstraction in the design. The separation of functionality allows designing specific components that take care of particular tasks in the system. Abstraction is something desired as it helps to decouple the interactions between the system components.

A layered architecture is the opposite of having all the code implemented under a single module, so any change to specific parts of the system may risk impacting things that were not intended to be changed. Still, as all the code is in the same module, all the code is highly dependable. Adding layers of abstraction is the way to solve this kind of problem, allowing modification but only affecting the component's scope and not the overall system.

Technology stack

We used Docker[1] and PowerApps[2] as our technological stack. We used Azure DevOps for the project repository and documentation. We used Azure Container Instances[3] (ACI) for the deployment of the solution.

Docker and containers

Docker is a containerisation framework that allows to create and work with containers using their API. This makes the development simpler and ensures that all the dependencies needed are included in the container deployed as a self-isolated service. Docker has the concept of *images* and *containers*. An *image* can be understood as a template. We built images that have precisely the libraries and dependencies that we need to work with for our project. For example, for this project, we created an image that contained Python version 3. We installed the necessary Python libraries for the system to work like Flask, a lightweight web server. We choose Flask because, among the Python web frameworks, is the one that allows rapid web prototyping with minimal setup and coding. Also it was straightforward to implement a REST API on using Flask.

We used that image to deploy create the container that will be deployed with the solution. *Containers* are created from *images*. A *container* is an instance of an existing *image*. As containers are created from images, we expect the same behaviour, platform-independent, of any container created from the same image.

Terraform

Early in the project, we explored the use of Terraform[4]. Terraform is a framework that helps create machines and other resources for computing using the APIs of cloud providers. Terraform allows creating and configuring cloud services from code or description files. Creating and consuming cloud services from source code is known as Infrastructure as a Service (IaaS). The idea was to configure the deployment from the repository and then use Terraform to deploy it. The advantage of using IaaS is

[1] Official website of Docker, https://www.docker.com (accessed 15 September 2021).

[2] Official website of PowerApps, https://powerapps.microsoft.com/en-us/ (accessed 14 September 2021).

[3] Official website documentation, https://azure.microsoft.com/en-gb/services/container-instances/ (accessed 14 September 2021).

[4] Official website of Terraform, https://www.terraform.io (accessed 16 September 2021).

that it allowed us to deploy the full solution from the source code and it could be deployed to any cloud provider.

Prototyping

The estimation of people required at a project site used regression algorithms to estimate the number of unexpected failures and then computed the number of hours a technician will spend fixing unplanned maintenances. The user could query the estimation model on demand. The user has a PowerApp application where to input the size of the project in megawatts, the number of machines and type of generators that the project will use, the quality of the fuel that is going to be used in the devices, the quality of the air in the location of the project site. Given the inputs from the user, the system will respond with the estimation of people needed to work supporting the project site.

The system required to be interactive. This interactivity with the models demands a different way of batch or streaming processing architecture. The interactivity can be solved using an API to query the models while allowing the user to interact with the results.

We presented an early prototype of the solution to the stakeholders in week three of the project to collect user feedback. The prototype was built using Flask, a lightweight Python web server, and Python code in a Docker container. The benefits of presenting an early prototype were twofold: it helped to realise that the back-end of the solution needed to be modular to improve the model and make future improvements and helped with the design of the user interface to add the interactive section on the results page. The prototype implemented in Python as a monolith application had a straightforward implementation of the Excel document. The final solution used a much more decoupled design.

As in the classic software engineering book "The Mythical Man-Month" [10] the advice in chapter 11, saying "*plan to throw one [design] away*" because the first versions, often, include design criteria that may compromise functionalities or make more difficult future changes to the architecture. The early decisions condition how the software will be developed, and there is often less information to make these decisions early in the project. The same principle was applied to the design of the ADEL library as described in Chapter 6.

The solution

The final solution implemented a combination of a layered architecture with a microservices architecture. We modularised the design into layers to abstract the behaviour of the solution. The implementation of the layered architecture used a microservice architecture.

We used Python code for the back-end replicating the regression models, an API to interface between the back-end and front-end, and a front-end user interface implemented in PowerApps.

The containerisation of the solution helped to deploy the latest changes to the solution in a fast way. Future improvements to the regression models or the replacement of them would be happening by simply updating the models' code in the back-end with the new models and then deploying them. This would not impact the final user as the container is replaced with an updated version, and from now on, the API will be answering requests using a new version of the model. Of course, the new models should have been tested and validated, but the change would occur with no impact on the application's API or the user interface.

The back-end replicates the logic from the Excel file. All the models and functions are implemented in Python.

7.4.2 Data

The outcome from Phase 1 was an online solution that replicated the standalone spreadsheet current estimation model. The current estimation model was developed and shared using an Excel document. Within the spreadsheets, there was tabular data that was queried using *vlookup* functions inside the Excel file.

We added the tabular data to the project repository. We traced the formulas and links inside the Excel file and we identified the necessary tables to be included as part of the project. We created a *data* folder inside the project's repository and added five CSV files *electricians_hours.csv*, *helpers_hours.csv*, *mechanics_hours.csv*, *service_crew_hours.csv*, and *shift_patterns.csv*. The first four files contained the times per activity reported by the technicians from surveys. The shift patterns file contained the work time schedules for different countries and regions to know how many hours per week they are expected to be available depending on the shift in a particular location. Any update on the working patterns should be updated in the shift_patterns.csv file in the repository and redeploy the project. The working patterns are not expected to be updated often.

The information used in the CSV files was collected during 2016 and represented static data based on a sample of technicians and project sites. The time per activity is one of the things that would require to be updated and generating this information from up to date data from the database system that records the maintenance of the machines on the project sites. We identified the system and its database. We explored the database during the second phase of the project.

7.4.3 API

We implemented the estimation model in a Python module (a single .py file) called *headcount_model.py*. The headcount module implements functions to compute totals and generate the results. A single function is used as an entry point for the API, it receives a dictionary with the input data and returns another dictionary containing detailed information to be presented by the user interface. There is a single function that calculates each one of the detailed information. The response of the *get_estimation()* function includes the expected number of hours of planned and unplanned maintenance per mechanics, electrician, service crew and helpers; and the number of people needed per technician considering the project requires 24/7 coverage.

The API was implemented using Flask. A lightweight Python web server. The API receives a single POST message containing a JSON file with the input data for the models. The API calls the estimation model and receives a JSON formatted structure returned in the API POST call to the API. The same structure is returned to the API caller as the response of its POST API call.

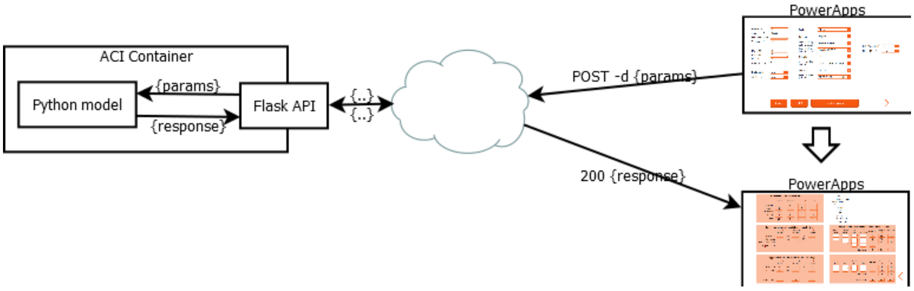


Figure 7.1: The architecture of the application. The estimation model is a Python module with a REST API that accepts a single POST message to query the estimation model. Both modules are containerised in a Docker container. The API is exposed in port 5000 and can receive calls from client applications. The API will reply to the POST call with the results of the model in a JSON file. The PowerApp in the front-end will request the results and present them back to the user in a second interactive interface.

Using an API simplifies the interaction between the front-end and the models in the back-end. Any changes to the models just need to be adjusted in the API layer and vice versa. A lightweight web server allows the application to be self-contained without the need for additional resources.

The architecture of the application is presented in Figure 7.1.

7.5 Implementation

We created a Docker image with the code of the project and pushed that image to Azure Container Services (ACS) so that we made the image available from Azure so we could create containers using the image. The Docker image will contain the headcount module, the CSV files containing the data for lookup reference values, and the REST API that queries the estimation model. The image starts the webserver in port 5000. When a container is created using that image, the container will start listening to the port's API.

The front-end of the system was implemented using PowerApps. PowerApps simplifies the development of mobile apps. The users can access our app from tablets, browsers, or chat apps such as Microsoft Teams. We expected that the final users would make use of tablets in the project sites.

The security is integrated by default as to have access to the PowerApp app, is required to have a valid corporate email account. The access to the app is only allowed using the corporate network of the organisation.

The graphic user interface has two pages. In Figure 7.2 we can see the first page for input data. Most of the input parameters are implemented as a selection list to avoid data quality problems. The domain values for the parameters are part of the repository in CSV files.

The results of the estimation model are sent to the client application. The user is presented with the estimated number of planned and unplanned hours for the maintenance of the project. In Figure 7.3 we can see the second page with the results. The user can modify parameters in the results and play with what-if scenarios. The user can test different configurations of the project depending on the people available or other factors. For example, the user can change the number of already confirmed people for the project site, and the app will calculate how many additional people are required per expertise type. This additional functionality makes the application useful as the results can be modified according to people's general availability around the globe. The user interface then offers the amount of local or global rotators necessary to support the project site maintenance of the machines.

The technology stack used in this project is the following:

- Python 3.6 or above for *flask* and *flask_restful*
- Docker
- PowerApps
- Azure DevOps
- Azure Container Registry (CR)
- Azure Container Instances (ACI)

The modular architecture allows us to update the forecast model component with no impact on the front-end. If a new parameter needs to be added or returned to the front-end, it will imply a slight change to the API, and as it is versioned, it won't impact the current implementation until the new version is updated in the client app.

7.5.1 Deployment

The project is on a git repository in Azure DevOps for automated deployment using the build options in Azure. This project included the use of continuous integration using Azure DevOps. When a commit is pushed to the *main* branch of the repository, it will automatically deploy the system as a new version.

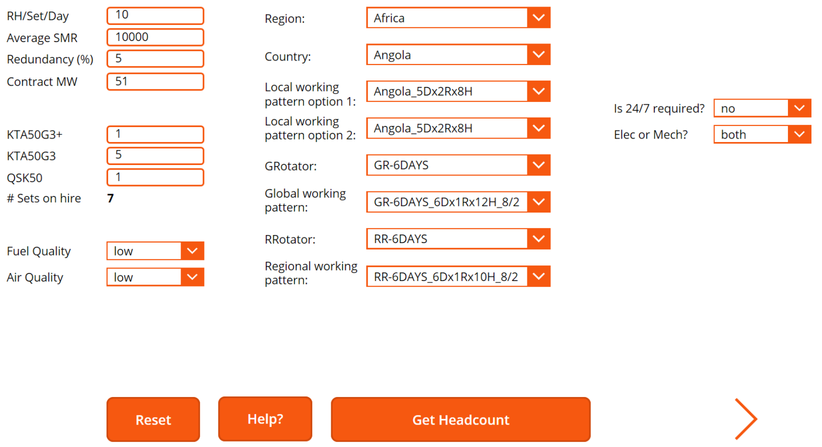
We created a pipeline in the code repository for the deployment of the project. So every time we would push code to the main branch of the repository, the deployment pipeline would be triggered, creating a new Docker image, registering the new Docker image to the container registry of Azure, creating a container from that image, and exposing it for other applications to consume it. In our case, the application was a PowerApp application that sends a JSON file with the POST call, and it expects a JSON file back from the API with the response and all the data that needs to be presented back to the user.

When a container is created, we can assign it a fully qualified domain name (FQDN) and pass the FQDN to the front-end app to access the REST API.

The back-end and the webserver are containerised in Docker using Python 3.7. We deployed a single container to Azure Container Instances (ACI) with the models and API, but it can be deployed to a Kubernetes cluster if necessary. In ACI, the front-end can query the models using an FQDN for the API.

7.6 Evaluation

We compared the results from the model in Python to the Excel files.

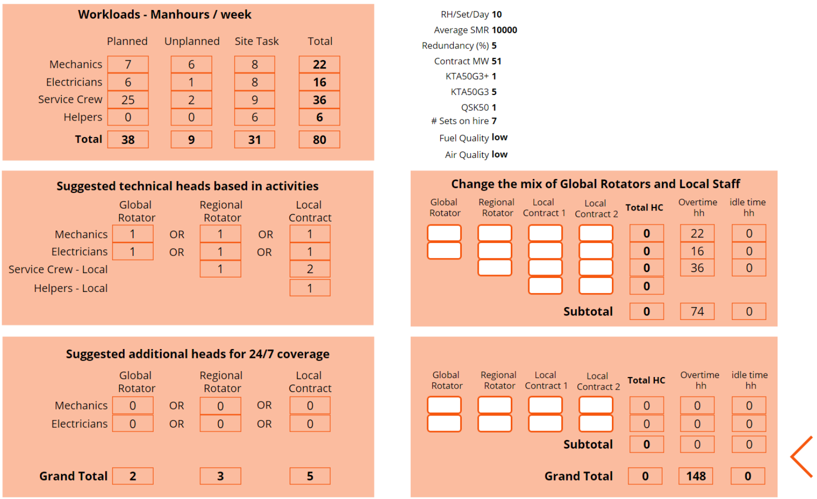


RH/Set/Day	10
Average SMR	10000
Redundancy (%)	5
Contract MW	51
KTA50G3+	1
KTA50G3	5
QSK50	1
# Sets on hire	7
Fuel Quality	low
Air Quality	low

Region:	Africa
Country:	Angola
Local working pattern option 1:	Angola_5Dx2Rx8H
Local working pattern option 2:	Angola_5Dx2Rx8H
GRotator:	GR-6DAYS
Global working pattern:	GR-6DAYS_6Dx1Rx12H_8/2
RRotator:	RR-6DAYS
Regional working pattern:	RR-6DAYS_6Dx1Rx10H_8/2

Is 24/7 required?	no
Elec or Mech?	both

Figure 7.2: Screen for input data.



Workloads - Manhours / week

	Planned	Unplanned	Site Task	Total
Mechanics	7	6	8	**22**
Electricians	6	1	8	**16**
Service Crew	25	2	9	**36**
Helpers	0	0	6	**6**
Total	**38**	**9**	**31**	**80**

Suggested technical heads based in activities

	Global Rotator		Regional Rotator		Local Contract
Mechanics	1	OR	1	OR	1
Electricians	1	OR	1	OR	1
Service Crew - Local			1		2
Helpers - Local					1

Suggested additional heads for 24/7 coverage

	Global Rotator		Regional Rotator		Local Contract
Mechanics	0	OR	0	OR	0
Electricians	0	OR	0	OR	0
Grand Total	**2**		**3**		**5**

RH/Set/Day	**10**
Average SMR	**10000**
Redundancy (%)	**5**
Contract MW	**51**
KTA50G3+	**1**
KTA50G3	**5**
QSK50	**1**
# Sets on hire	**7**
Fuel Quality	**low**
Air Quality	**low**

Change the mix of Global Rotators and Local Staff

Global Rotator	Regional Rotator	Local Contract 1	Local Contract 2	**Total HC**	Overtime hh	idle time hh
				0	22	0
				0	16	0
				0	36	0
				0		
			Subtotal	**0**	74	0

Global Rotator	Regional Rotator	Local Contract 1	Local Contract 2	**Total HC**	Overtime hh	idle time hh
				0	0	0
				0	0	0
			Subtotal	**0**	0	0
			Grand Total	**0**	**148**	**0**

Figure 7.3: Screen for results.

The security of the system is managed via Azure authentication. All the app users must have a valid account and are registered in the organisation's network. The users of the PowerApp are granted access to their nominated accounts using their corporate email accounts.

We deployed the system in Azure Container Services and granted access to a few users to test the app. The users could access the URL of the app by logging in to the organisation's network, which is something they are most of the time while they are working.

We added model assertions as testing of the model [94] for testing the results generated by the estimation model module of the system. Model assertions are used to improve machine learning models. Still, we included early on with the standalone implementation of the forecast models thinking that in the future, for phase 2, we would require techniques for testing and debugging the machine learning models that we would train using up to date data.

We received good feedback from the users. Some users asked if we could implement a forecast model like the one built for diesel assets but for gas-powered assets. We explored this option in phase 2 of the project. The answer is yes. As we can collect data for diesel, we could collect data for gas assets, changing some of the filters and queries to the same data sources, and this would enable the modelling for gas assets allowing us to estimate the number of necessary technicians to support the project sites that use gas assets. We tested creating a training data set for the gas assets in phase 2 of the project.

7.7 Phase 2

In this section, we will describe the work we did for phase 2 of the project. It was a continuation of the exploration performed in phase 1 of the project. This phase did not deliver a new estimation model but explored the data sources for the project mainly because of time constraints and changes in the roles of the stakeholders of the project at the time of the implementation of phase 2.

In phase 2 we managed to identify the necessary data sources for a new forecast model. We identified the tables where the service orders are stored in the system, and we created a simple machine learning model to estimate the number of unplanned service orders per week. The exercise of creating a machine learning model was to check if the data we collected could be used or not for a final model or more data would be necessary to complete a forecast model. We concluded that we could use the data we found to create a complete model. But adding more data sources is desirable and those data sets were not available on demand for an analyst to deploy and maintain a new machine learning model at the time of the exploration.

We required the data about the project sites to be available in the central data repository for analysts to explore and retrain models. We identified the data sources, but those systems were not yet in a centralised data repository to be used on request. It was possible to query the data from the data source, but analysts would not have direct access to it in the data lake. There were two systems that we had no access to their data on request. The technician skills database where we could find the experience and courses of the technicians and the technician application data that records the starting and finishing time of the tasks technicians performs on the maintenance of the assets. This limitation made us decide to hold the deployment of a full estimation model using updated data.

Among the many data sources explored, we identified the assets assigned to the project sites. For each asset, we had identified the engine type of the machine. We could sum up the capacity or size of the assets and determine the size of the projects. We also found data about the service orders by type, planned and unplanned. We identified the different types of generators assigned to project sites.

We created a training data set to test if a machine learning model would have predictive capabilities using this data.

7.7.1 Planning

We listed the tasks that we would need to do once we were ready to start working on phase 2 of the project. We established five high-level tasks that we needed to explore: data exploration, data modelling, deployment, evaluation, and post-analysis.

Data exploration

We identified the critical systems and databases to be explored for phase 2 to get the necessary data to build the forecast models. We needed to do a historic analysis of services orders with a focus on unplanned maintenance orders. We would require guidance from business experts to determine the type of orders that classify as unplanned. We would need to explore the tables of the database using the profiling tool explained in Chapter 2.

The analysis would include the frequency of the service orders per asset type and locations over time. We had to take into consideration potential changes in the typification of the service orders for unplanned maintenance. The stakeholders told us that in 2018 there was a change in the labelling used in the system that records the service orders, so this would impact any raw analysis unless we clean and combine the types of service orders.

Among the identified data sources for phase 2 is the system where the service orders are recorded and updated as the technicians work on the assets. A secondary system that technicians use to record all the maintenance processes is used from handhelds or tablets and helps record regular checks on the machines or results and notes from inspections to the devices. It records the actual starting and ending times of maintenance tasks. This data would help to compute the time spent per maintenance task more accurately. Another dataset is the technician's level and experience that we think we could contribute to the project to weigh the experience into the estimation model to have more accuracy in the results.

Data modelling

Once we had the data, the first step would be to create a *baseline model* using a simple predictor. We would prefer not to use the current estimation model as a baseline model as it was built using a small subset of projects and data from three years ago. A simple baseline model trained with up to date data can be compared against the results of the original estimation model but just for reference, not as a validation of the new model.

A correlation analysis will be necessary to determine useful features to be used by the model. Also we would require to apply some feature engineering operations to transform and improve the accuracy of the models using the data available. Finally, training multiple machine learning models and select a sound model to replace the current forecast model that uses surveyed data and manual inputs to generate the estimations.

Deployment

The deployment of the solution was implemented during phase 1 of the project. We developed a deployment pipeline that when we push changes to the main branch of the project's repository, the solution will be deployed to a container. The PowerApp could access it using an FQND assigned to the container.

For a replacement of the estimation model, we would need to include the data collection and transformation operations implemented in the data modelling stage to reproduce the forecast model's training.

The modularised design of the solution makes it simple to replace or update the model or any of the components of the architecture.

It will be necessary to include an additional pipeline to retrain the forecast model whenever it is needed, or concept drift [28] happens.

Evaluation

It will be necessary to test and validate the results from the new estimation model. We would grant access to a reduced number of users to get estimations and compare them with their experience.

We plan to do the exercise of computing the forecast from using both models, the original one and the one trained with up to date data and use it for comparison, not as a measure of accuracy as the models were trained using a different subset of data. The differences or similarities in the predictions would be for reference to the users to inform about any discrepancies from the previous way to estimate the number of workers in project sites.

Post analysis

The idea of a post-analysis of the project is to provide more information on what impacts the forecast model—for example, analysing what factors impact the number of service orders: location, running hours, type of personnel, usage pattern of the assets, etc. This would generate some understanding of what triggers unplanned maintenances on project sites and could mitigate some of the actionable factors. This step is different from data modelling because data modelling will indicate what currently correlates with unplanned maintenances. A post-analysis could include data not necessarily used to train the model but related to the assets' operations on the project sites.

We want to monitor the model's performance to determine when necessary to retrain it or deploy a new version of the model. Ideally, to suggest a period to retrain the model over time.

7.7.2 Modelling

We were able to create a training data set to predict unplanned maintenances. Our target variable for the model was the number of unplanned maintenances per asset and per project per week. We used this data set to develop a machine learning model to predict the number of unplanned service orders per week. This model was created only to test the available features' predictive capacity and inform about this for a future project.

In Figure 7.4 we can see the total number of service orders of a given project in the blue line and the number of SO10 in the orange line.

We gathered data about the project sites and how many assets were allocated per project site. We identified the asset type or engine types of the assets. With the identification of the assets, we were able to collect the service orders data. The service orders are where the maintenance system stores the information about the maintenances of the assets.

We were interested in two types of service orders. Service orders *number ten* (SO10) and service orders *number thirteen* (SO13). Both corresponded to unplanned

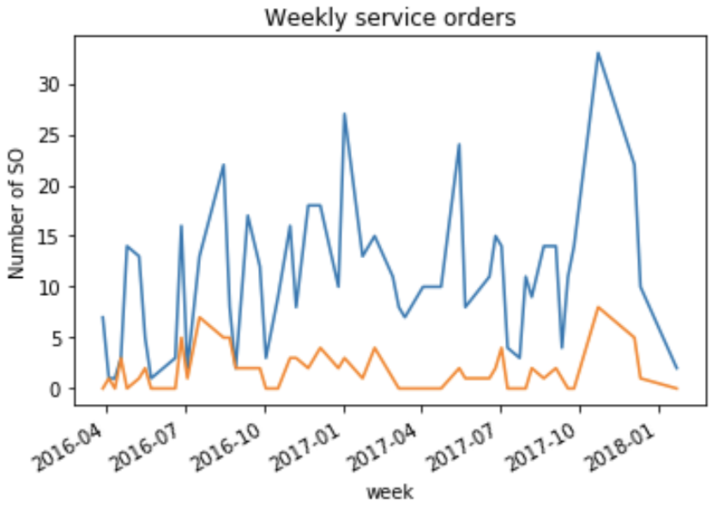


Figure 7.4: Number of service orders per week. The total number of SO is in blue and the number of SO10 is in orange.

maintenances on assets. In 2018, there was a change in the labelling of the unplanned maintenances for the assets. Before 2018 the unplanned maintenances were labelled with SO10, and after 2018 the technicians started using SO13 for unplanned maintenances. We had the starting and finishing dates of the assets in the projects to calculate the weeks hired and the number of service orders per project per type of planned and unplanned maintenances.

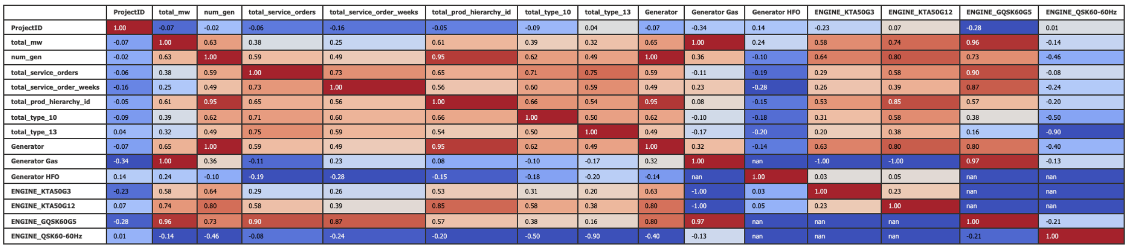
We created the training data set, created features with the available data, and created three target variables to explore what time horizon would be more beneficial for a forecasting model. The code was flexible enough to make a training data set for a given engine type. So we could test that multiple machine learning models for the different asset types.

We created three target variables to test machine learning models with different time ranges to predict the number of unplanned service orders. We also created the target variables for other asset types. The idea was to train and compare the three models. The benefit of training three models is that we can compare the predictive capacity of the features we created for the model and choose the best model to predict the future demand of technicians at project sites. The three target variables were the number of unplanned service orders 1) in the next three months, 2) in the next six months, and 3) in the next twelve months. The training data will only contain information from the time previous to the starting date of the period of the target variable.

Feature engineering

The one-hot encoding technique was used to create binary features from the project category, which indicates the type and location of the project sites by geographic zones.

Four additional features were created to represent five Megawatt (MW) sizes of projects. Less than forty MW, between forty and eighty MW, between eighty and hundred and twenty MW, and greater than a hundred and twenty MW. It was decided on the thresholds after analysing the histogram plot of the project's sizes



	ProjectID	total_mw	num_gen	total_service_orders	total_service_order_weeks	total_prod_hierarchy_id	total_type_10	total_type_13	Generator	Generator Gas	Generator HFO	ENGINE_KTA50G3	ENGINE_KTA50G12	ENGINE_GQSK60G5	ENGINE_QSK60-60Hz
ProjectID	1.00	-0.07	-0.02	-0.06	-0.16	-0.05	-0.09	0.04	-0.07	-0.34	0.14	-0.23	0.07	-0.28	0.01
total_mw	-0.07	1.00	0.63	0.38	0.25	0.61	0.39	0.32	0.65	1.00	0.24	0.58	0.74	0.96	-0.14
num_gen	-0.02	0.63	1.00	0.59	0.49	0.95	0.62	0.49	1.00	0.36	-0.10	0.64	0.80	0.73	-0.46
total_service_orders	-0.06	0.38	0.59	1.00	0.73	0.65	0.71	0.75	0.59	-0.11	-0.19	0.29	0.58	0.90	-0.08
total_service_order_weeks	-0.16	0.25	0.49	0.73	1.00	0.56	0.60	0.59	0.49	0.23	-0.28	0.26	0.39	0.87	-0.24
total_prod_hierarchy_id	-0.05	0.61	0.95	0.65	0.56	1.00	0.66	0.54	0.95	0.08	-0.15	0.53	0.85	0.57	-0.20
total_type_10	-0.09	0.39	0.62	0.71	0.60	0.66	1.00	0.50	0.62	-0.10	-0.18	0.31	0.58	0.38	-0.50
total_type_13	0.04	0.32	0.49	0.75	0.59	0.54	0.50	1.00	0.49	-0.17	-0.20	0.20	0.38	0.16	-0.90
Generator	-0.07	0.65	1.00	0.59	0.49	0.95	0.62	0.49	1.00	0.32	-0.14	0.63	0.80	0.80	-0.40
Generator Gas	-0.34	1.00	0.36	-0.11	0.23	0.08	-0.10	-0.17	0.32	1.00	nan	-1.00	-1.00	0.97	-0.13
Generator HFO	0.14	0.24	-0.10	-0.19	-0.28	-0.15	-0.18	-0.20	-0.14	nan	1.00	0.03	0.05	nan	nan
ENGINE_KTA50G3	-0.23	0.58	0.64	0.29	0.26	0.53	0.31	0.20	0.63	-1.00	0.03	1.00	0.23	nan	nan
ENGINE_KTA50G12	0.07	0.74	0.80	0.58	0.39	0.85	0.58	0.38	0.80	-1.00	0.05	0.23	1.00	nan	nan
ENGINE_GQSK60G5	-0.28	0.96	0.73	0.90	0.87	0.57	0.38	0.16	0.80	0.97	nan	nan	nan	1.00	-0.21
ENGINE_QSK60-60Hz	0.01	-0.14	-0.46	-0.08	-0.24	-0.20	-0.50	-0.90	-0.40	-0.13	nan	nan	nan	-0.21	1.00

Figure 7.5: Correlation of variables generated to train a machine learning model.

before splitting the data into train and test datasets.

Results

We trained a machine learning model using a logistic regression algorithm (Chapter 4.3.4 of [29]) and a decision tree (Chapter 14.4 of [29]). The decision tree was a decision tree regressor. We used the scikit-learn Python library [42] to train the models. The accuracy of the tree regressor was 93% which is good for a model just using project locations and project size as input variables. The explanation of the variance was 93% as well.

A k-nearest neighbour algorithm with $n = 2$, achieved 64% accuracy and 43% with $n = 3$.

7.8 Conclusions

The contribution of this project was a modularised architecture to deploy a forecast model, that could be reused to deploy other types of machine learning models that require to be served to client applications. The exploration and understanding of project site related data that can be used for the development of future projects related with project sites.

The forecast model for fuel powered asset was successfully deployed in containers and made available in the Azure cloud for users to query the model.

The project was well received by the stakeholders. We presented the results and the solution deployed in a retrospective meeting before releasing the solution for testing and validation by end users.

After the diesel model was released for testing, there was a request to implement a similar model for gas powered assets which is in standby at the time of writing this chapter.

7.9 Further development

There is a correlation between the number of hours an asset runs and the number of unplanned maintenance. This means that the lifetime of the project sites may impact the performance of the machines used by the project. There could be differences in the number of unplanned maintenances at the beginning, middle, or once the project is coming to an end. The workloads could be different over time so this could be interesting to analyse in the future. If this hypothesis is true, this could account for the manning allocation of project sites.

After the implementation of the diesel fuel model, there was interest from the business to develop new estimation models for gas engine types. This is feasible, there was enough data to make it work, but the implementation of such model was out of the scope of the manning optimisation project that dealt with diesel engine models. This could be a further development leveraging the knowledge generated during the manning optimisation project.

Another interesting analysis that can derive from this work is to determine the life cycle of machines. Analysing the maintenance logs of planned and unplanned maintenance to see if it can be optimised or help in the prediction of other maintenances.

One of the challenges for future development is to confirm the accuracy of the times reported in the maintenance log systems. Make sure the activities are mapped correctly to the tasks. Validating the accuracy of the data collected from the project sites.

Chapter 8

External Fuel Tank Battery Analysis

Summary

This project explores the lifespan of lithium batteries attached to fuel sensors on external fuel tanks. External fuel tanks refill the internal fuel tanks of generators deployed in remote project sites around the world. The fuel sensor's batteries should have a lifespan of 12 months.

8.1 Introduction

This project presents a single experimental analysis on the duration of batteries of sensors based on an information requirement from the operations team in charge of planning the maintenances of the machines.

This project reflects some of the skills expected from data professionals that sometimes will have to transform business requirements from the domain experts into an analytical approach. The use of analytical and critical thinking skills was crucial to planning the analysis for a project like this.

Our contribution in this project was to put together an analytic approach based on verbal requirements from stakeholders. We designed an analytical project where used several applied data analysis techniques to answer the question from the stakeholders.

We used the automated exploratory data analysis tool presented in Chapter 2 to explore the content of a telemetry database to identify the necessary tables for the analysis. We applied survival analysis and spectral clustering to the data available as part of our approach. The details of the design and implementation are explained later in this chapter.

8.2 Background

The Power Solutions (PS) line of business works with long-term projects. The duration of the projects can vary from a few months to several years. In contrast, the Rental Solutions (RS) line of business deals with short term projects that can last for a day to often a few weeks with a high rotation of the assets used in the projects. This analysis is in the context of the PS line of business.

Power Solutions projects often have assets powered by diesel with internal fuel tanks. The capacity of the internal fuel tanks depends on the model of the generator. Sometimes, the projects will also have external fuel tanks deployed to the project sites. The external fuel tanks can be connected to more than one asset simultaneously and will refill the internal fuel tanks when they are below a certain level, often below the 40% mark.

When a Power Solutions project ends, the assets are sent to a warehouse to be checked and prepared for their next deployment. But with multi-year projects, it is necessary to anticipate the needs of maintenance of the assets and service them on-site.

Once the assets are installed on a project site, they are not expected to be relocated during the project's duration. For this reason, it is essential to be aware of the times when specific parts of the assets require maintenance.

The fuel level of the external fuel tanks is monitored with level sensors. These level sensors have lithium batteries attached to make them work. The level sensor will stop reporting the level of the external fuel tank if its battery runs out of charge, which can cause different kinds of problems.

By default, the sensor reports the fuel level every twenty-four hours. It sends a single data point of the measurement to a telemetry database. The expected duration of a battery of a fuel level sensor is twelve months. Replacing batteries in advance is considerably cheaper than waiting until it fails which might stop the project or provoke a failure.

8.3 Problem Statement

The question for this project was *can we determine the average lifetime of the batteries of the fuel sensors of the external fuel tanks?*. Do these batteries last for twelve or more months?

8.4 Methodology

In this chapter, we will use the terms *stakeholders* and *users* interchangeably depending on the context, but they refer to the same business people interested in the project.

We held regular meetings with the stakeholders, presenting early results for discussion and guidance on the analysis. We produced results for every weekly meeting. Having early results to discuss and analyse proved helpful in understanding the problem, validating the data at the beginning of the project, and then conducting the data analysis.

We used a methodology of documentation for data science projects recommended by Microsoft[1]. The git workflow and documentation artifacts used during this project became a recommended practice informed in Section 4.5.6 of Chapter 4 about coding guidelines and documentation of data science projects.

[1] TDSP Project Structure, and Documents and Artifact Templates, https://github.com/Azure/Azure-TDSP-ProjectTemplate (accessed 21 July 2021).

The documentation of the project included:

- **Take on document**: It contains questions to be answered from the stakeholder's perspective. The idea is to communicate relevant knowledge from the stakeholders to the development team. Some of the questions are: what are the challenges they see; what are the benefits (operational, financial, reputational, and of health and safety); the current process they use to get the results the project is intending to get; and finally about the data sources that we could use to work on the project.

- **Project Charter**: It contains an overview of the project with the scope, the people and their roles, the metrics used to define success, the planning, the architecture, and contact information for any questions regarding the project.

- **Data summary report**: It contains the description of the data sources, the exploratory data analysis, and initial analysis of the target variable, in this case, the voltage of the batteries.

- **Final report**: It contains information about the machine learning models used, their implementation, a description of the solution and the results. It is a model-wise report; it tells about the final model selected in the project. If there is more than one final model to report, a final report document is created for each final model.

- **Exit report**: It contains the main results from the project. It provides an overview of what the project was about, the business domain and the business problem that it helped to solve, a high-level description of the data used and the modelling and validation method used, the solution architecture, the benefits for the company and a section with *lessons learned* during the project where one of the questions is "What is unique about project, specific challenges?". This section is meant to be shared with the broader analytics team to build knowledge from finished projects and their particular challenges.

8.4.1 Project Plan

As in many new data science projects, the data for the project was unknown to the team. For this reason, we included an initial phase for data exploration and data discovery. The data analysis phase would be based on the results of the data exploration. What would be feasible is determined by the data available. And finally, the third phase is the analysis of the results and the suggested actions for the stakeholders.

Two weeks before starting this project, there was a major upgrade in the way technicians collected data from the sensors. Only the voltage signal of the batteries, among the many signals collected from the sensors, was usable for this project. We based all the analysis on the voltage signal of the batteries. Another impasse was that we didn't have enough data to cover 12 months of data. For this reason, we waited for six months to start the data analysis phase. We completed the data exploration phase, first identifying the data sources and designed the process and analysis that we wanted to apply to the data, then we waited.

At the end of the project the exit report was completed and shared with the stakeholders. The exit report marked the completion of the project. It contained an overview of the project, the main findings from the exploration phase, the modelling and validation techniques used, relevant results, any decision criteria that was used to during the project to let everyone know what decisions were made why. This report also contained a section of *lessons learned* so all the challenges and solutions can be shared with the rest of the data science team.

8.4.2 Data Exploration

The users had insufficient information about the data sources for the project. They had access to a web platform designed for monitoring the assets to check the battery and fuel levels but did not know about the underlying database.

The first challenge was to identify the tables from the database with the data for this project. We knew that all the necessary data was in that telemetry database that was used by the web platform. We knew the name of the database, but we had no information about which tables we could use for this project.

We profiled the database using the database profiling tool presented in Chapter 2. The results were around 250 tables, around 5,900 columns and more than 800 million rows in the database. The profiling exercise generated a queryable database with metadata from the original database.

The metadata database contained table names, column names, number of rows and number of columns per table, number of unique values and number of null values per column, the frequency of data values of columns with less than five thousand unique data values[2], time and date data grouped by month, and statistics such as average, standard deviation, variance, minimum, maximum, among other statistical moments for each numeric column.

We were able to query the database with some of the unique codes of the fuel tanks that we could get from accessing the web platform. This *sherlock-esque* method helped to identify the tables first and then their relationships with other tables. In less than a day of work, we had an idea of the columns and tables that were necessary to use for the project. In contrast, a manual search for the same data would have taken an analyst days, if not weeks.

Using the metadata, we first identify the ID columns of the tables that contained information about the fuel tanks. Among those tables, we identified the parametric tables containing information about the assets like their capacity in litres.

We used the column ID to search for other tables with the same column names first, and then we used a strategy of joining columns from other tables using the data domain of the columns. The use of the data domain allowed us to identify potential relationships with other tables. The next step was to search for more tables potentially related using the data domain of the other columns in the tables already identified.

The search by data domain can be costly in time and computing if applied directly to the source database. Still, using the metadata database, it is transformed to a join query of a 1-to-1 cardinality. For example, a column in a table may contain thousands of records but only a few unique values; a similar column from a different

[2] The 5,000 unique values threshold is the default value of the profiling tool and it was expected to have less than five thousand external fuel tanks in the system.

table may have millions of records but with the same few unique values; using the metadata from both tables, unique values and frequency, a query that would produce a many-to-many query can be reduced to a 1-to-1 query using only a few kilobytes in memory compared to a cartesian product using maybe gigabytes of data.

We validated the data by comparing the values presented in the user interface of the web platform and the values that we were able to generate from queries to the source database, based on knowledge obtained from its metadata.

8.4.3 The data

We identified around five tables related to the fuel tanks of the 250 tables in the database: two operational tables and three parametric tables with information about the assets and equipment. The telemetry data is often stored for all the assets in the same table, and there aren't asset-specific tables. The telemetry tables had over 100 million rows, of which only 2 million rows were of interest for the project. The fuel data table had 30 million rows with a data point every hour for each asset reporting the fuel level and a 3 million rows table summarising the daily fuel levels. Still, we could not make use of fuel measurements for this project.

The data exploration and findings for this project turned out to be useful for other internal projects. Among those projects the *fuel consumption rate* is described in Chapter 9.

Once we identified the tables, we proceed to query the database to understand the volume of the data first and then the potential for exploration.

Identifying the exact time when batteries are replaced is essential because it marks the ending and starting point of the lifespan of the batteries that we want to measure. There was no mark in the database for battery replacements.

A normal voltage signal is around 14V. The voltage values are expected to be between 10V and 14V. In Figure 8.2 we can see the voltage signal over time. A clear break happens when the battery is running out of charge. The signal starts decaying to voltages below 12V and sometimes below 10V until it has a sudden break and goes back to the 14V level. We designed a heuristic to identify those events and mark them as battery replacement marks based on this observed behaviour.

The heuristic to determine a battery replacement was a negative voltage variation of more than 1V. For example, if between two consecutive data points we spot a change from 9V to 12V, that means a variation of -3V, which would be a battery replacement. We tested the heuristic by plotting the voltage around those data points, and the identification of those changes was consistent for all the cases. There was an edge case when the signal would remain around healthy levels but would send no voltage signal for several days; we considered it a new signal if no data was collected after seven days.

These two decision criteria were validated and approved by the stakeholders. With this definition of battery replacement, we split the voltage signal of a battery into multiple *sections*, each *section* representing a battery replacement or a different lifespan of a battery.

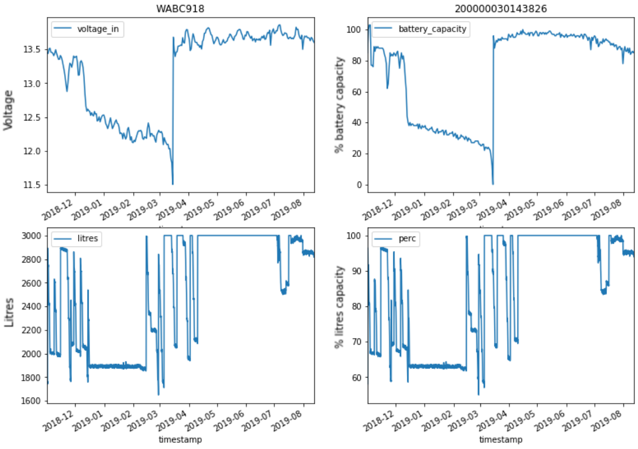


Figure 8.1: The four plots are for the same external fuel tank. From the top left and clockwise, we have the voltage signal, the battery capacity in percentage, the fuel tank capacity in percentage, and the number of litres of fuel in the tank. In all the plots, we have nine months worth of data.

8.5 Data Analysis

We filtered out the first *section* of the voltage signals because the batteries could have been working before the database started receiving the sensors' data. There was no way to determine how long the batteries ran until the first battery replacement.

With the data exploration phase, we understood the volume and scope of the data for this project. We identified the external fuel tanks and their associated devices, such as the batteries for the fuel sensors. With this understanding, we designed the analysis based on the volume of data available for the project.

We suggested two main types of analysis for this project based on determining the lifetime of a battery: Survival Analysis and Spectral Clustering.

Survival analysis is a method originally developed to estimate the lifetime of patients in healthcare, but it could be used in any domain that studies the lifetime or lifespan of a studied entity. It only requires the *duration* of the event and a mark indicating the death or survival at the time t. The result of a survival analysis tells the probability that the event death has occurred or not at the time t: $P(t) = Pr(T > t)$. It is expected that with this information, it will be possible to determine the probability of the batteries to be *alive* or with charge after 365 days.

Spectral Clustering. We wanted a way to group sections based on the shape of the curve that visually appear naturally to the observer, as can be seen in Figure 8.3, produced by the daily voltage measurements. This distinction can be detected using an unsupervised learning technique called spectral clustering. This clustering algorithm considers the similarities, calculated with a given distance function, among the sections.

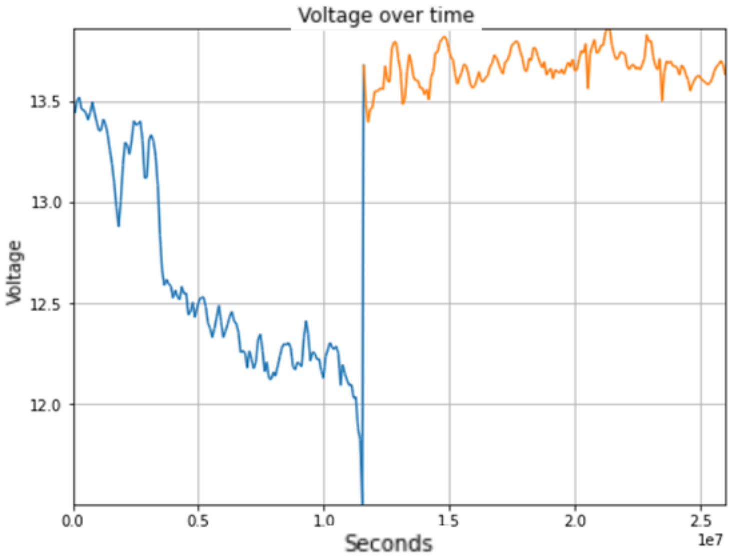


Figure 8.2: The voltage of a battery over time. The x-axis is Voltage, and the y-axis is seconds starting from zero, where zero is equal to the first timestamp of the time series. We can see in blue and orange two different sections of the same voltage signal.

8.5.1 Survival Analysis

We used the lifelines[3] Python library [95] to run the survival analysis on the batteries. The data modelling is straightforward for survival analysis. We modelled two columns T and E, where each section was a row in the dataset. T is an integer that indicates duration, in our case, the number of days that a battery has been reporting voltage data, and E is a binary or Boolean variable that indicates if the battery has died. This modelling technique allows to include sections that are still alive because the effect to be observed have not occurred for all the batteries. When a section is still alive at the measuring time, we talk about a right-censored data point.

It was important for this analysis to have some of the batteries that already have been replaced. We had to wait to collect more data before running the analysis. The data from batteries that are still running are called *right-censored*, we can not know what happened to those batteries after the instant of the last data point. The information is the current status of its lifespan, which is less than its actual lifespan.

Figure 8.4 shows the Kaplan-Meier survival curve, and we can see the y-axis is between 0 to 1 and represents the probability of a battery to be *alive* at x days, the x-axis is the number of days the battery has been reporting its voltage level. This plot was generated with nine months of data. We can see that above 60% of the batteries are alive after 250 days.

With survival analysis, we expect to know the probability of a battery to be *alive* at day n. For each time step calculates the probability of being alive at that instant, the aim is to analyse and model the *time-to-event* [33]. For example, at time zero, all the batteries will be alive or have a 100% chance of being alive. If one of the 200 batteries dies at day 50, then the probability of being alive that day will be 199/200

[3] Lifelines Python library, Official Documentation, https://lifelines.readthedocs.io/ (accessed 30 July 2021)

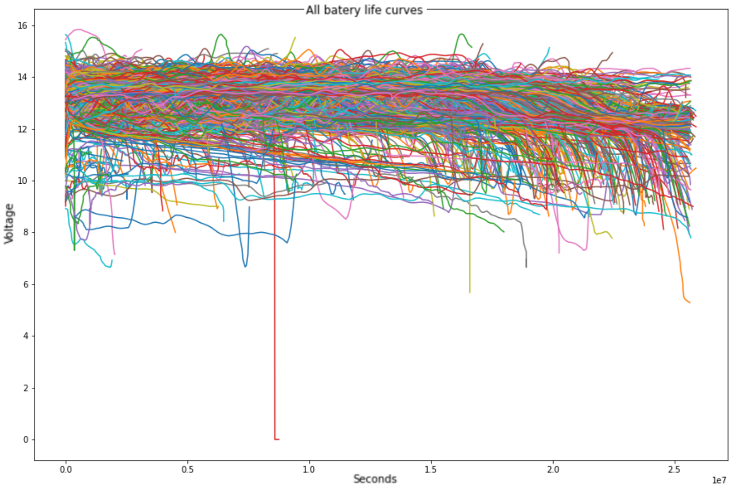


Figure 8.3: All battery life curves. The axis is Voltage vs Time in seconds, where second zero is the first timestamp data in the data set. Each curve is composed of voltage measurements collected every twenty-four hours. The plot contains nine months of voltage data from 1,300 sensors.

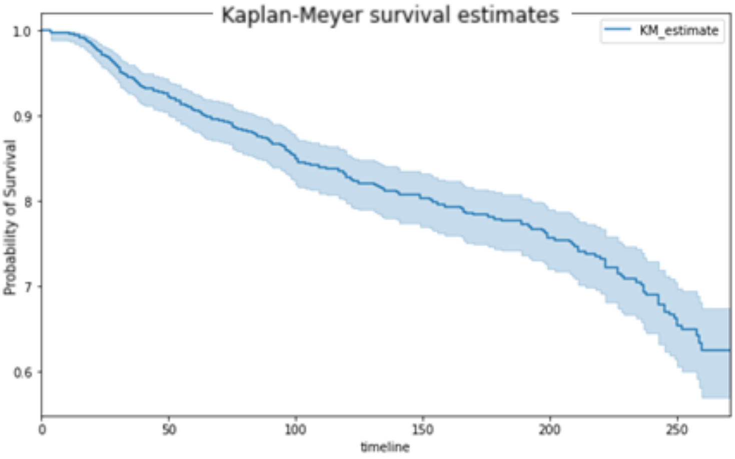


Figure 8.4: The Kaplan-Meier surviving curve.

or 99.5%, if a second battery dies at day 65, the probability of being alive that day will be 198/199 * 199/200, which is 99% and so on.

We ran the same analysis using cohorts. In Figure 8.5 we can see the batteries grouped by the month they began sending data to the database. The Nov-2018 cohort is the one with more data points.

We suggested repeating the analysis once the project has surpassed more than a year of data collection.

8.5.2 Spectral Clustering

The expected behaviour of the lithium batteries, according to the stakeholders, should remain around 14V until it decays only in the last few days of its lifespan. In contrast, an alkaline or non-lithium battery will decrease steadily over time. We wanted to identify lithium/alkaline battery swaps, for example, replacing a set of 14V lithium batteries for a set of 14V alkaline batteries on the project sites and also

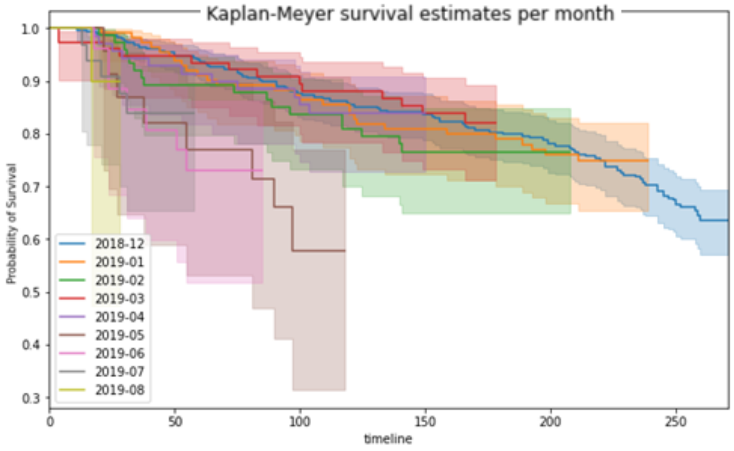


Figure 8.5: The Kaplan-Meier surviving curve by month.

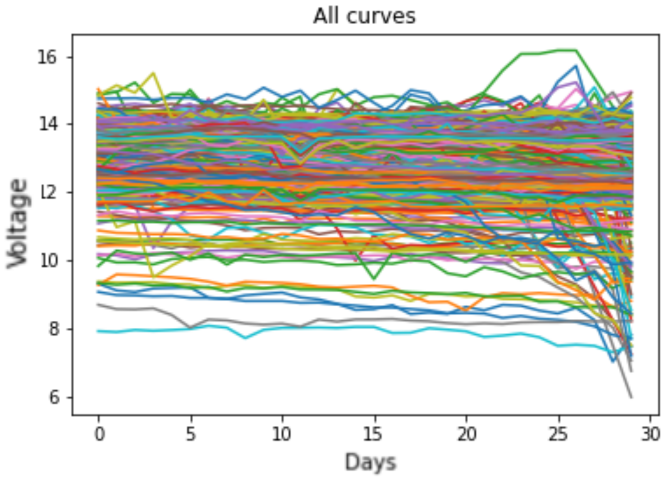


Figure 8.6: Plot of the voltage signal of the last 30 days per battery.

the malfunctioning lithium ones.

After plotting the voltages of the batteries, as can be seen in Figure 8.6, we thought the spectral clustering method could help us distinguishing well-performing batteries from poorly performing ones.

We used the last 30 days of voltage data of each section.

We used the sklearn [23] spectral clustering implementation for the analysis.

The spectral clustering computes a pairwise similarity measure using a given distance function like the euclidean distance. Then, it uses a standard clustering method like K-means to generate the groups based on the data points' similarity matrix rather than the actual data points.

The number of clusters is an input parameter for the algorithm. We ran it multiple times with a different number of clusters, between 2 and 8 groups. We visually inspected the results to determine that seven groups were a good number of clusters. We could identify two groups of failing batteries from this clustering and five groups with normal behaviour. We also decided that the 30 days period of data points of data would be reasonable, with enough anticipation for the users to take action on the results of malfunctioning batteries or for further investigation. We estimated that at least seven days of voltage data is required to spot batteries

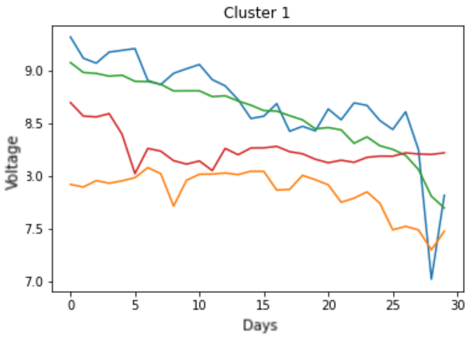


Figure 8.7: Cluster 1.

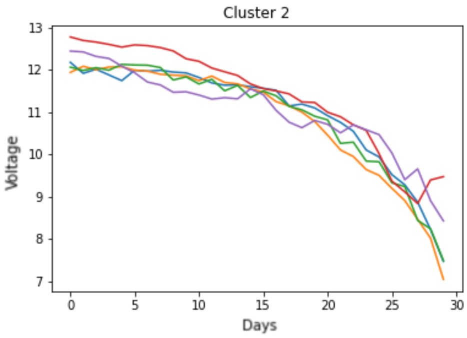


Figure 8.8: Cluster 2.

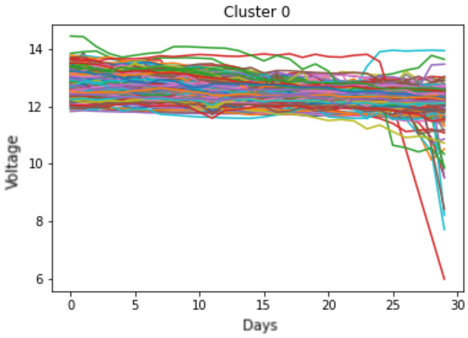


Figure 8.9: Cluster 0.

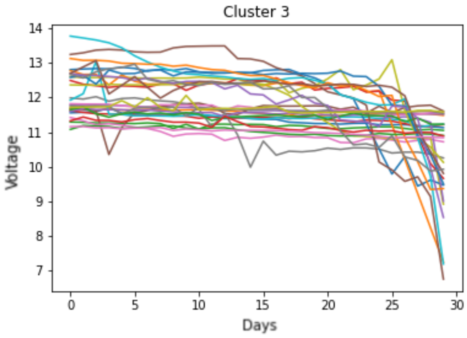


Figure 8.10: Cluster 3.

that are about to stop working.

We choose a clustering run with seven groups. Two of the clusters, cluster 1 and cluster 2 in Figures 8.7 and 8.8 contained malfunctioning signals. Cluster 1 had voltages below 10V which is abnormal and cluster 2 shows a rapid decrease from 12V to 7V in the last days. Clusters 0 and 3, in Figures 8.9 and 8.10 looks like normal behaviour around between 10V and 14V and in the last five to seven days present and abrupt decay to lower voltages. Clusters 4 and 5, in Figures 8.11 and 8.12 present low voltage signals. And finally, cluster 6 in Figure 8.13 contains all the normal and stable voltage signals.

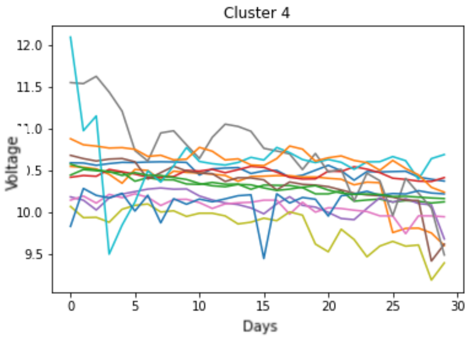


Figure 8.11: Cluster 4.

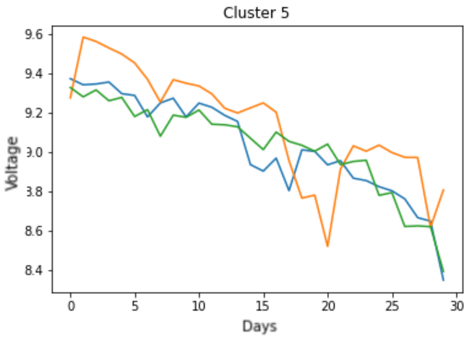


Figure 8.12: Cluster 5.

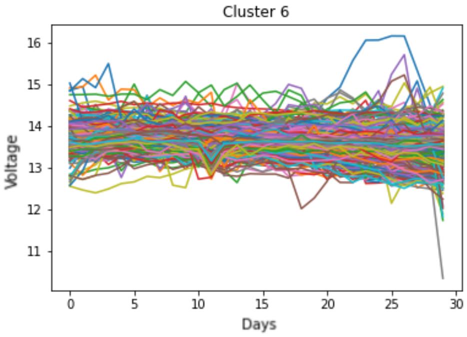


Figure 8.13: Cluster 6. Normal voltage signals.

8.6 Conclusion

The survival analysis and spectral clustering used for this specific project could be used for a wide range of projects related to duration and signal analysis in simple terms. Survival analysis could be used, for example, to study and understand the payment process of the company's customers; to understand the lifetime of generators by model type; to understand the estimation of the duration of the project sites; to study the length of projects by industry and geography. Spectral clustering could be used, for example, to detect malfunctioning of assets that should be running at similar workload levels; to understand customer behaviour and cluster them based on their purchasing behaviour.

One potential problem of using *only* survival analysis to study the lifespan of batteries is that it reports the median, which is a single value to evaluate a whole population. An analyst should extend this analysis to analyse the deciles or more percentiles of the population to generate better insights about the batteries.

Often, survival analysis is considered outside the scope of data science and machine learning, and it is ignored by analysts when this type of analysis. This is a missed opportunity. The mission of the data scientists is to suggest the best method to answer the question, independently of the field of computer science, statistics, or other data related fields.

8.6.1 Further analysis

We think that further analysis can be applied, for example, factoring in the location and weather of where the external fuel tanks are located. Another factor that could influence the lifespan of the batteries could be if the external fuel tanks are under a roof or have protection from the weather. A survey of the current projects could obtain this information, but this analysis was out of this project's scope.

We believe that the temperature and battery capacity signals are relevant, but they are out of this project's scope.

An analysis that was left out of scope was to study the correlation between data points and battery charge. The hypothesis, in this case, is that the sensor reporting more data points to the database are consuming more energy; therefore, there will be more battery consumption. An analyst could analyse this hypothesis by running the number of data points reported in a given period and the average voltage of the battery in the same period.

Chapter 9

Fuel consumption rate

Summary

This chapter presents the implementation of a fuel consumption rate calculation to prioritise the connection of generators to the power grid based on their performance. The fuel consumption rate is part of a macro project containing six questions to design experiments to answer them. The methodology and the six questions are presented in the first sections of this chapter.

9.1 Introduction

This macro project was intended to answer analytical questions for which the organisation did not or was not collecting data about the problem. The idea was to design the mechanisms to collect the correct data and develop ways to analyse it. The focus was on business hypotheses for which there was a sense of knowledge but no evidence to support decisions.

This project used collaborative design thinking [32] of projects to define the six problems to explore. The main contributions are the processes to agree and prioritise, by the ability to execute and business impact, on project ideas and the experimental analytical for one of six experimental questions. The ideas came from a pool of business hypotheses generated by the stakeholders of the project. This project is an excellent example of collaboration between business units and technical departments where the needs from the business are transformed into feasible analysis to respond to a business need or information.

Similar to the work presented in Chapter 8, we put together the analytical approach for this project using applied data analysis techniques. We designed the algorithm to produce critical insight from the data. We explored the data using the data profiling tool presented in Chapter 2 that helped us to understand the limitations of the data sets and allowed us to design mechanisms in the workflow to account for them. For example, filtering outliers in the calculations and also using linear regressions with sawtooth-shaped data.

We started this project as a set of questions with the engineering team of the company. We held a one-day workshop with the stakeholders of this project. The idea was to explain what the design of experiments could do for the engineering team.

In Section 9.2 we preset the methodology used. In Section 9.3 we describe the six experimental questions. In Section 9.4 we present the implementation of one of the six experimental questions regarding the fuel consumption rate of machines. In Section 9.5 we present our results and finally in Section 9.6 we suggest additional analysis that can be built on top of the code base of this project.

9.2 Methodology

We organised a one-day workshop with the stakeholders of the project from the engineering team. We used the session to think of questions that we wanted answers for, but we are not collecting data to support analysis.

The first exercise was to think of *observational studies* where we can use historic data to answer the analytical question. For those questions that we don't have data available, we could design *controlled experiments* to collect the necessary data to answer the analytical query.

We explain the benefits of framing the problems as designing experiments. We asked to think of unanswered questions that we may or may not have data available. We asked to put them in post-it notes freely. We collected more than thirty ideas for analysis. We read them out loud and explained the intention of the study. We grouped similar ideas, and we used one post-it note representing the grouped ideas. We then asked each participant to vote for three ideas that they considered more important. We selected the most voted ideas and then asked the participants to rank the ideas along two axes. One axis is on the importance of the organisation, and the second axis is with easiness to execute the analysis.

We ended up with a selection of six ideas that we planned to execute depending on the availability of data and other factors.

Our approach was to exhaust the possibilities of *observational studies* given we have already data we can use, and *designing controlled experiments* for the unknowns that remain unanswered.

It is planned to use the *statsmodels* [39] Python library to run the analysis. *MLFlow* [85] to keep track and compare the results from the experiments. And using a Git repository for documentation and source code of the project.

The six questions were ranked according to three criteria. 1) Data available? 2) Readiness to start? And 3) Complexity. This chapter only presents the implementation of one of the six questions which is: *Determining better or worse-performing assets in terms of fuel consumption*, as the priorities of the stakeholders changed after a few months into the experimental design project.

9.2.1 Guidelines

We developed a document with guidelines to conduct the design of experiments. This was the first time we tried to use statistics from experimental design to answer data science questions. We thought it would be helpful to document our findings and learning process to expand this project or start new experimental questions with the resources collected for this project. We added the guidelines document to the documentation of the git repository of this project.

The guideline contains notes taken from the books and resources we found to inform the development of this project. For example, we mentioned the two options we have to answer analytical questions. One is to run *controlled experiments* where an engineer can change working conditions, and we can evaluate its impact. The second option is to have *observational studies* where performance is measured but not intervened to get a response.

Before starting with the design and definition of the experiment, we set up a questionnaire for the analyst that would help clarify the objective of the analysis while helping define things for the experiments: what is the objective of the investigation? Who is responsible, stakeholders? Do we have any past data about the events in the study? How was that data collected? How were the responses measured? How much physical theory is known about the phenomenon? What are the sampling, measurement, and adjustment protocols?

We advised following a *sequential* approach where a series of small experiments is better than a single comprehensive experiment.

Process knowledge is required when selecting the ranges of operation of the variables. We explored domain driven design in Chapter 7 that can be helpful to understand how to leverage the internal knowledge about the processes to design the solutions better.

Based on the literature around design of experiments [72], [4], [5]. We established a seven-step process for each one of the questions. The steps would help track the progress of the project and mark the necessary steps to complete the analysis. The steps are:

1. Recognition of and statement of the problem.
2. Selection of the response variable
3. Choice of factors, levels, and ranges
4. Choice of experimental design
5. Performing the experiment
6. Statistical analysis of the data
7. Conclusions and recommendations

To identify the problem to be solved we need to ask questions such as is this a *screening* or a *characterisation* problem?. Is it a new system, or do we want to understand what impacts the response variable?. Is it an *optimisation* problem so that we are interested in what is the best way to handle the asset or finding the levels to optimise the response. Is it a *confirmation* problem where we put the hypothesis to the test. Or a *discovery* problem where we are interested in gaining knowledge about the system. We need to consider the *robustness* of the study, under what circumstances the response variable seriously degrades.

During the planning of the study, we need to identify what is our response variables.

Also, we need to choose our factors, define their levels and ranges. We need to classify the factors into *potential design factors* and *nuisance factors.* The potential design factors we need to classify them into *design factors* that are the ones selected to study in the experiment; *held-constant factors* that are variables that may exert some effect on the response, but for purposes of the experiment, these factors are not of interest so that they will be held at a specific level and *allowed-to-vary factors.* The nuisance factors we need to be aware of three types: *controllable*, *uncontrollable*, and *noise factors.* The blocking principle is often useful to deal with controllable nuisance factors. The blocking principles dictate the creation of homogeneous blocks so that the nuisance factors are held constant across the experiment. When the objective of the experiment is factor screening or process characterisation, it is usually best to keep the number of factor levels low.

Having the response variables and the factors and their levels, the next step is to choose the experimental design. We need to define, for example, the sample size and number of *replicates.* A replicate is a single item or element in an experiment run. We need to define a proper run order for the experimental trials and determine whether or not blocking or other randomisation restrictions are involved. We are interested in identifying *which* factors that cause this difference and in estimating the *magnitude* of the response change. Also necessary is the type of model we want to find, first-order models and interactions, second-order models for optimisation, or binomial experimental design.

We recommended running *pre-experimental runs* to identify factor levels and ranges so that we can use this information to design the experiments. For example, using a cause and effect diagram, we could identify controllable and uncontrollable factors.

Experimentation is an iterative process. To learn, we formulate a hypothesis about a system. We run experiments to investigate this hypothesis, and on the results, we develop a new hypothesis, and so on. We advised running a few trials before performing the experiment to verify that data is collected correctly and suitable for the analysis.

We could use *graphical methods* to help with the interpretation. Like the *fishbone diagram* or also known as the *Ishikawa diagram* [18] created to identify the causes of a quality problem.

We included links to resources and documentation of different statistical methods to compare variables and analyse the results.

Statistical methods can not prove that a factor (or a set of factors) has a particular effect. They only provide guidelines as to the reliability and validity of results.

9.3 Experimental questions

From the workshop we chose six analytical questions that are described below:

1. Filter differential pressure
2. Impact of low-quality oil
3. Impact of fuel quality
4. Vibration measurement
5. Determine better or worst performing assets

6. Sensors added value

9.3.1 Filter differential pressure

We wanted to determine the correct times for maintenance and replacement intervals of the *differential pressure filter.* Specifically, we want to monitor and proactively schedule the maintenance and replacement of filters measuring the oil differential pressure.

A blocked filter may increase the temperature inside the machines and cause the malfunction them. Implementing proactive filter changes improves the airflow into the containers leading to better efficiency.

The benefits of implementing proactive filter changes are operational, having a more efficient maintenance schedule. Better maintained assets may have a longer life.

Finally, this analysis can decrease the downtime of machines due to unexpected maintenances produced by differential pressure, which is a common scenario when the assets are not constantly monitored.

This project aims to find the right time for filter replacement on-air and oil filters. Sensors are measuring the pressure for air filters, and there will be necessary to start measuring the oil filter differential pressure.

The current process to obtain differential pressure filters is performing field trials on machines to determine how the filters age and measure the differential pressure. This process is followed in one of the project sites for air filters, but it would be ideal for estimating the different machines operating across the various project sites worldwide.

For oil filters, there are sensors attached in machines that work with gas, but they are not used now.

The data for air filters are stored in local SCADA systems located in the project sites. The oil filters sensors need to be attached to the machines and start the data collection.

9.3.2 Impact of low-quality oil

The question is, what is the impact of using low-quality oil on the engines? Is there degradation of parts faster than using standard or high-quality oil? Does the oil have a shorter life? Does it generate more blocked filters?

Answering this question could have an operational impact on other parts of the machine. The oil quality may be impacting the ageing process of the engine parts using the oil, leading to a faster depreciation of the machine. It also could be affecting adding more unplanned failures leading to site downtime.

The data for this project is a database with oil samples taken at intervals of around 250 hours for projects of Power Solutions.

The project in Chapter 5 implements a data architecture to collect, combine, and centralise the data stored in local SCADA systems in project sites and creates proactive alarms to notify potential malfunctioning of assets to minimise downtime on the sites.

9.3.3 Impact of fuel quality

The question of this potential experiment is *what is the impact of the fuel quality on the lifespan of an asset?*

The operational benefits of computing this analysis impact a more efficient way to replace fuel filters, increase the injector and fuel pumps lifespan, minimise the debris circulating inside the engine. Combustion byproducts can build up on pistons requiring more cleaning. Also, incomplete or poor combustion lead to lower efficiency of the generator set.

Bad fuel quality may lead to increased operational costs because the asset will consume more fuel to deliver the same power and will require maintenance sooner than using better fuel quality.

One of the challenges of answering this question is that high-quality fuel might be scarce in some regions of the planet. Fuel quality varies over time on the project site depending on the availability daily.

The current process is collecting samples on bottles of gas and diesel. There is no data reference for the weather that may impact the performance of the machines.

The data for this project are the bottle samples of fuel from project sites.

9.3.4 Vibration measurement

This question tries to determine the best approach for maintaining rotating machinery and service needs if it is better to have spot maintenances or scheduled continuous services based on vibration measurements on the machines.

The benefit of answering this question is operational as we are trying to avoid catastrophic failures or the engines or alternator on the machine.

The challenge with this project is the use of expensive equipment and extensive validation needed to understand the expected performance of each type of equipment.

There is no data collection happening to answer this question. To implement the project, sensors must be acquired and attached to the machines; multiple tests would be necessary to determine the best location inside the generator to measure vibration. Different devices would require other sensors because of the internal space distribution of the parts of the generators in the containers.

9.3.5 Determine better or worst performing assets

This question tries to determine specific fuel consumption patterns to identify better or worst-performing assets.

9.3.6 Sensors added value

Among all the sensors attached to a specific type of machine, are they being used to make decisions? Prevent failures? What is the added value of the sensors attached to devices? Are more sensors required? Different types of sensors? A reduced number of sensors would achieve the same results?

The idea is to test the hypothesis that some sensors are underused and are attached to every machine.

9.4 Implementation

We started with the fifth question from the workshop. Identifying the better or worst-performing asset has medium complexity. There was data available about the fuel consumption of the assets. We will explain in the data exploration how we narrowed the data sources to find the necessary data to answer this question about fuel consumption.

Problem Statement

Over time some assets decay their efficiency in fuel consumption. Not all generators have a workload all the time. It depends on the demand on the project site, and this demand can vary over time. When power demand increases, connecting more assets to the power grid that delivers energy from the project site is necessary. Being able to select the more fuel-efficient assets represent cost savings and better service for the customers.

The question is *can we determine the best and worst asset in terms of fuel consumption?*

9.4.1 Data Exploration

We use the data profiling tool presented in Chapter 2 to explore the telemetry database. We found 248 tables with more than 48 hundred columns and 813 million rows when we ran the profiling. It was challenging to explore this database without its schema or data dictionary, which we didn't have.

The profiling tool creates a metadata database with the information collected from the tables. We mainly used this metadata database to identify the columns and tables required to compute the analysis.

We identified four common column names across the database: *UID*, *NAME*, *UNIT_ID* and *NODE*. They have different syntaxes for the codes they use in the database, but they are connected somehow because we could see combinations of columns present in the same tables. Based on these four columns, we identified the unique values of those columns in other tables of the database, so that we extended the search for other *ID* columns so that for the operational tables, the ones with more records, we identified what the ID columns that are used in them and their data domains were. We summarised the tables indicating what information the table stored and which of the four IDs we could use to join that transactions table with the rest of the data model. We also knew the number of unique values each ID had for that table.

In one day, we explored the whole dataset using only its metadata to navigate through its content. We were able to identify relationships between the tables using the data domain of the ID columns and search them at once in all the other columns of the database but using the metadata. This search was efficient as each value is stored only once with its frequency.

We discovered that the table we needed was the *fuel_raw* table with 30 million rows. We found the other tables to relate fuel tanks with a contract and a project site.

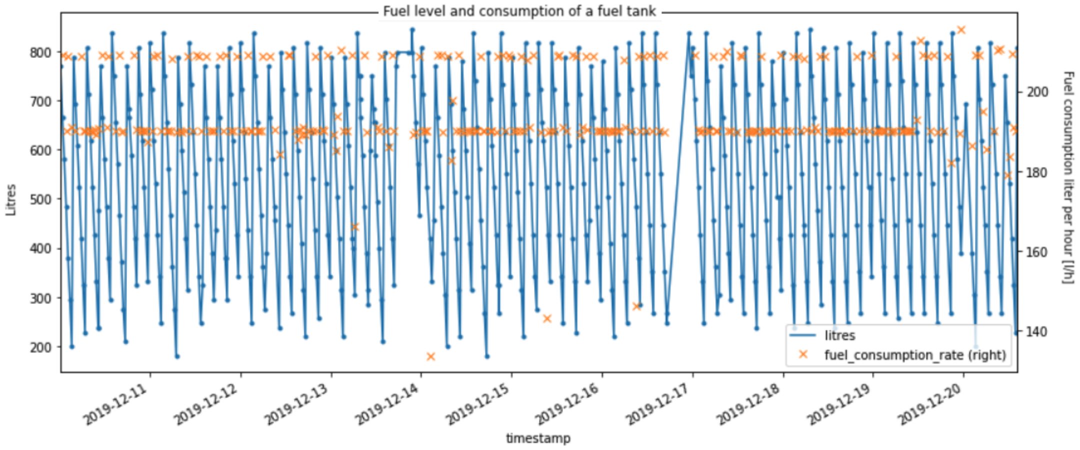


Figure 9.1: Plots of the level of litres in the tank over time in blue line and fuel consumption rate in orange.

9.4.2 Data preprocessing

The analysis of fuel consumption was made using only hired machines. Hired machines are allocated to a customer and a project site.

The generators can have three statuses: 1) Standby, 2) running, and 3) running with a load. The analysis is focused only when the assets are running with a load by indication of the stakeholders of the project.

The status of the machines is collected in the database, so we combined the fuel levels and their status only to use the fuel consumption when the engines were *running with load.*

The fuel consumption data behaves as a saw chain, as can be seen in Figure 9.1. We filtered the data points not to use the upper and lower measurements for two reasons. We could not know when the tank was refilled, so that difference in time could affect the computation. Because we needed to compute a rate, we needed the differences between the data points to be positive to calculate the slope of simple linear regression.

We computed the fuel consumption rate between all the contiguous data points and averaged them to determine the rate for the asset.

We filtered out some data points from the consumption rate. Those data points were out of range, as can be seen in Figure 9.2. We tried different methods generally used for anomaly detection. We used the support vector machine algorithm and isolation forest. We preferred isolation forest for its capacity to capture more outliers from the data.

9.4.3 Data Analysis

We wanted to find the fuel consumption rate of different assets and compare them to determine better and worst-performing assets. We also wanted to identify factors that may produce this difference.

We identified the asset types of each machine allocated to a project site.

The fuel consumption is determined by:

$$C = E * Ckwh$$

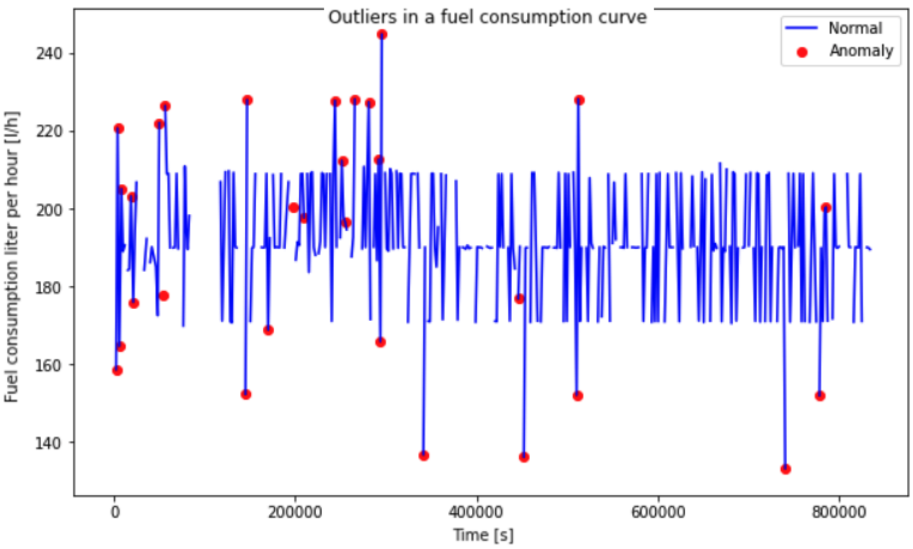
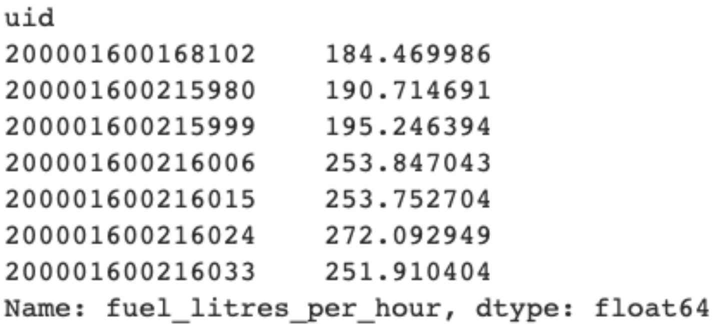


Figure 9.2: Anomaly detection on the fuel consumption rate data. The anomalies are the red dots that will be filtered out to compute the average fuel consumption.



```
uid
200001600168102    184.469986
200001600215980    190.714691
200001600215999    195.246394
200001600216006    253.847043
200001600216015    253.752704
200001600216024    272.092949
200001600216033    251.910404
Name: fuel_litres_per_hour, dtype: float64
```

Figure 9.3: Ranking of assets allocated to one project site sorted by fuel consumption per hour. uid is the unique identifier of a machine, the number on the right is fuel consumption rate.

E is the active electric energy in the output of the diesel engine in kWh.

$$E = P * h * d(kWh)$$

P is the active electric power in the output of the diesel generator in kW.

$$P = S * cosphi$$

Where S is the apparent electric power in the output of the diesel engine in kVA $S = P/cosphi$. $Cosphi$ is the power factor (usually between 0.8 and 1). h is the number of hours per day the generator runs. d is the number of days the power generator runs. And $Ckwh$ is the consumption of fuel per kWh (usual value is between 0.3 and 0.6 (l/kWh)).

We reused code from the fuel tank battery analysis presented in Chapter 8. We have already identified some dimension tables and identified some of the tables that store telemetry data.

We had the active power data and the cosine of phi from the data source.

The fuel level data was sampled every thirty minutes by default. The generator status data were reported when they changed, so that we assumed the same status until the next status data point. We

We encapsulated the most frequent queries to simplify the process of getting data from the database. To get the fuel level and status of the asset was a parametric function so that we could try different approaches and get the data We wrapped the pre-processing data tasks so that we could call the parts from the notebooks.

We performed the analysis using SQL and Python. First, using the metadata database, and once we identified the tables we needed, we used the data source database to run the queries and analysis. Using Python scripts, we queried the data and then processed it using Python commands.

The first analysis was to estimate the number of project sites and the number of machines allocated to each project site. This analysis was simple thanks to the identification of the tables with automated data profiling. Otherwise, it would have required to explore more than eight hundred tables because there is no schema or data catalogue about the data.

We created a function *get_fuel_consumption_mean(uid, days = -10)* that returned the average fuel consumption of an asset per hour considering the last ten days of fuel data. The UID is a unique identifier of the machine. With this hourly fuel consumption, we can rank the assets in a project and determine what are the assets that consumes more or less fuel while they are working with load as we can see in Figure 9.3.

We further applied the analysis to three different levels of workload. We categorised the percentage of load based on the workload as a percentage of the maximum capacity of the generator. We classified the percentage of workload into three categories: below 50% as low workload, between 50% and 80% as medium workload, and above 80% as high workload. We computed the fuel consumption rate for each of the ranges low, medium and high workload to check if, depending on the load that the asset had at the time, the consumption rate might depend on the type of workload. The fuel consumption rate was different depending on the workload, but the ranking of the assets remained in the same order.

9.5 Results

We completed the work of identifying the best and worst-performing assets given their fuel levels and statuses. The next step was to use this calculation in the project sites.

Two strategies were planned to deploy this solution, 1) as a batch processing tool where we could rank the generators offline and send or publish an updated list of the generators to the project sites for them to have the information of what assets are more recommendable to use based on the conditions demanded; and 2) as a module inside the software that controls the plant in the project sites.

We started the conversations to attach our fuel consumption calculation as a module to the controlling software of the project sites. At the time of evaluating this option, we halted the project due to a change of priorities, and the analysis was conclusive and available. Still, it was not deployed to the project site.

At the time of this project, there was no central repository for the data of project sites. In Chapter 5 we explored the construction of an analytical database that could store the data used on this project and would facilitate the deployment of the fuel performance results of the assets.

9.6 Further development

From the analysis of how the machines are working and consuming fuel, it would be trivial to determine the opposite. What are the unused assets, and why? This question could be answered and create red flags for the support that are not rotating in the fleet of machines.

Chapter 10

Conclusions

The basis for this thesis is a simple question: how does an organisation best become *data-driven*?

We believe we contributed to becoming a more data-driven organisation based on principles taken from the software engineering field of knowledge. We made specific contributions with targeted analysis to provide timely information to decision-makers. We also made more general contributions by building the tools to implement the specific contributions and writing guidelines and recommendations from the experience of the implementation of the projects.

We applied best practices from software engineering to successfully deliver data science projects based on the principles of scalability, abstraction, encapsulation, and extensibility. We encapsulated and automated repeatable tasks and created recommendations for the rest of the team to follow.

Across the multiple projects, we made recommendations to the data science and data engineering teams. For example, the design of the data engineering library showed an excellent reference point for the design of a data science library where the data scientists are encapsulating data collection and the use of algorithms tailored to the data science processes of the organisation. The execution of the EngD projects was, we believe, capacity building for the data science and data engineering teams that now can build on top of the developments achieved in the tasks of the EngD and reuse the methodology and documentation generated in the repository of the projects.

The methodology used for the projects, framing the project under a research question and computing an exploratory data analysis to understand the problem better, proved helpful for all the projects. We modelled the problem based on the data available.

Technology decisions can be costly over time if we compare them with decisions about processes. Processes can be adapted and modified over a short period. But technology decisions such as what technology stack will the company use, what programming language will be the one that newcomers should use for all their projects, and so on have a more significant impact. These decisions have a long term impact.

We often referred to the technology stack of the organisation in the chapters. We believe this is a crucial point that enables and amplifies the work of the analysts working in the organisation. The incremental gains given by the correct use of technology that a skilled analyst can achieve represents the difference between a successful delivery of a related data project or not. For this reason, any organisation needs to have a continuous process of evaluating emerging technologies. One way to do that is to incorporate the use of new technologies as part of project activities to decide the usefulness of these sometimes hyped technologies and decide by themselves, the organisation, if the incorporation of a new technology could bring benefits or not to the internal processes.

At the same time as the exploration of new technologies, there is a balance between the number of tools that an organisation uses and the number of problems for which those tools are used. An extreme case would be that only one technology is used for all the use cases and data processes the organisation has. The other extreme is to adopt a different tool for each use case. The balance should consider the cost, and required skills, of maintaining different and multiple platforms and the benefits of using that particular tool. There is the need then to use the *correct* type of technology to solve a specific kind of problem. Adopting or replacing technologies should be a thoughtful process that involves constant evaluation and training of the people using the technologies because there is a risk of under-utilising the potential of the technologies adopted.

We tried to investigate and incorporate new technologies from the design of the EngD projects so that we would learn something new simultaneously; we would assess technologies that could be useful to adopt to a data-driven organisation. We would advise allocating effort in the projects to use or learn a new tool, library, policy, etc., that is related to the project, but the organisation has not adopted that. This scouting process could become a standard practice that could spark innovation by having early access and evidence of the workings of new technologies.

Thinking in general about the data projects presented in this thesis document, we can reflect that the most successful projects were those that the stakeholders were most involved in, interested in hearing the results along with the development of the projects. In our opinion, the projects with low stakeholder involvement tended to last more due to unanswered questions or lack of guidance on the next steps of the projects. We mitigated risks and solved the problems early in the process, so no greater impact in those projects with higher stakeholder involvement.

Something that we will include from now on in our risk assessments of future projects, is to add an exit plan for when there is a change in the priorities of stakeholders, what to do if the priorities of the stakeholders change while the project is in development. It may sound non-necessary because if users and stakeholders commit time and resources to take in a data science project,t is expected that they will stay until the completion of the project. We had a couple of situations where the priorities of the stakeholders changed due to externalities, so for those cases, it would be good to have an exit plan where it is stated clear what to do depending on the stage of the project and how to maximise the contribution of the effort if a project is halted.

Working on the data projects during the EngD, we realised that no project or success story would have been possible without the collaboration and support of the colleagues and management of the organisation. The collaborative spirit prompt to solve any issue during the implementation of the projects was a constant contributing factor to the success of the performance of the EngD projects. In one of the first retrospective meetings and the beginning of the programme, one of the quotes that stuck with us was: “It is incredible how much can be accomplished if no one cares who gets the credit.” (*John Wooden*) and that still resonates with us in the way to approach new projects and collaborate with other people. This simple quote, in some sense, reflects the collaboration and teamwork of the organisation. This collaborative culture is a takeaway for us to apply to future projects and endeavours.

Most of the projects presented in this thesis used structured data and internal data sets. A recommendation for future research would be to evaluate the usage of unstructured data such as images and text from available open data sets. Incorporating this type of data could improve and present opportunities for the data science team on their machine learning models and understanding of the events they are modelling. We would recommend using a DevOps and DataOps approach to data projects to maintain close communication with the stakeholders. Finally, we would advise incorporating an interpretability step to the machine learning development process that could benefit the analyst to understand better the machine learning models and to the stakeholders to learn the reasons for the scoring results from the models.

We can classify the data projects into mainly two categories. Long term projects implementing a new architecture or solution to be stable over time and ad-hoc analysis where the focus is on an insight or gaining quick knowledge about a problem for a decision-maker. We believe that both types of projects are relevant in their own way. Long-term projects are required to scale and implement stable processes over time, optimising the efforts of the organisation, and ad-hoc analysis are important because they solve information blockers that allow organisations to support the decision-making process highly relevant for data-driven organisations. The exploration work of ad-hoc analysis can lead to long-term projects and trigger innovation faster. This thesis contains both types of projects. The projects presented in Part I represented more long term projects, and Part II included more specific projects looking for specific analysis and insights from the data.

The data science and data engineering projects require a mixed set of skills from the people implementing them. As it happens with the tools, there is no silver bullet skill or experience to solve any type of problem. The projects vary in difficulty, impact, return of investment, and the variety of users and stakeholders. The level of expertise of the team can be a relevant factor in the success of implementation. We believe that mixing the level of experience and skills can work well to develop the shared pool of skills within the team.

The broad range of exposure working with people in technology, the business, and the organisation’s management gave me a perspective to work and communicate with stakeholders. It also provided me with the formation of soft skills.

This thesis presented a series of data science and data engineering projects implemented to help become a data-driven organisation. An organisation able to make decisions based on information generated from data. The data related projects followed best practices from software engineering, making relevant the utility of these methods. A data-driven organisation can benefit significantly from a data science and data engineering team with skills from a software engineering background. A data-driven organisation also cares about the toolsets for the analysts. The set of platforms and infrastructure can expand the talent of the people working with them. The technology stack an organisation chooses to work with determines how effective and efficient that organisation can become. And together, the people and tools shape the culture of the organisation. These three components are crucial to maintain and constantly develop the data solutions in the organisation.

The automation of processes and the use of novel technologies to solve the information needs, the search for use cases before implementing new technologies, the design for scalability and extensibility, communication with the stakeholders and collaborative work with the project team are all, in general, contributing factors to becoming a data-driven organisation. Once the organisation achieves the capabilities of answering questions with insights from data analysis, it becomes a constant process and shared responsibility within the team to maintain it.

Appendix A

Appendix: Survey questions

Question of the survey to internal analysts.

1. What analytic tools for data science, machine learning do you use or plan to use to train machine learning models?
2. How would be the process(es) you would follow to deploy a ML model into production?
3. In your opinion, what are the common problems when you try to deploy a machine learning model into production?
4. What technologies, platforms, tools, or frameworks would you recommend to explore and review to deploy machine learning models in production environments? Please name all the ones you think could be relevant.
5. What are the most important features in a platform for ML model management? Mark all the ones you consider important.
   - Scalability
   - Integration with existing tools
   - Simplicity
   - Easy to deploy
   - Easy to maintain
   - Usability
   - Error checking
   - Auto training of ML models
   - Performance monitoring of the deployed ML models
   - Easy to train ML models
   - Low latency for serving models
   - Non dependent on the framework of the trained model
   - Can handle ensemble models
   - Minimal configuration
6. In a scale of 1 to 10 where 1 is not useful at all, and 10 is very useful. In your opinion, how useful is to have a tool for ML models governance?

7. What kind of ML models are you planning to deploy in production within the next 6 months or after? (You can refer to the use case, algorithm, technology, etc. or all of the above)

8. Please let me know if you have any feedback and comments.

Appendix B

Appendix: Infrastructure configuration

In a Linux/GNU server with Docker.

B.1 Docker

```
# This are the docker images to pull
docker pull postgres:9.6
docker pull kiwenlau/hadoop:1.0
docker pull jupyter/datascience-notebook
docker pull jupyter/base-notebook
docker pull python:2.7
docker pull docker.elastic.co/kibana/kibana:5.4.3
docker pull docker.elastic.co/elasticsearch/elasticsearch:5.4.3
```

B.2 Kafka Server

Edit the following lines on the In the modify the following lines on file `config/server.properties`.

```
delete.topic.enable=true
advertised.listeners=PLAINTEXT://kafka-server:9092
```

On the file `bin/kafka-run-class.sh` edit this lines:

```
-Dcom.sun.management.jmxremote \
    -Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false \
    -Dcom.sun.management.jmxremote.rmi.port=<PORT>
-Djava.rmi.server.hostname=<IP> \
    -Dcom.sun.management.jmxremote.port=<PORT>
```

Setting up the environmnet.

```
# Create a network called telemetry
docker network create telemetry
```

```
# Starts a single node Kafka server
docker run -itd --name kafka-server \
    -h kafka \
    --network=telemetry \
    -p 2181:2181 -p 9092:9092 -p 9999:9999 \
    darenas/kafka:1.3 bash
docker run -itd --name kafka \
    -h kafka \
    --network=telemetry \
    -p 2181:2181 -p 9092:9092 -p 9999:9999 -p 1098:1098 \
    darenas/kafka:1.3 bash

# start ZooKeeper
bin/zookeeper-server-start.sh config/zookeeper.properties

# start Kafka Server
bin/kafka-server-start.sh config/server.properties

# create the topic telemetry
bin/kafka-topics.sh --create --zookeeper localhost:2181 \
    --if-not-exists --replication-factor 1 \
    --partitions 1 --topic telemetry

# check the topics in kafka
bin/kafka-topics.sh --list --zookeeper localhost:2181

# start a consumer to check the incoming messages
bin/kafka-console-consumer.sh \
    --bootstrap-server localhost:9092 \
    --topic telemetry --from-beginning

# to delete the telemetry topic
bin/kafka-topics.sh --zookeeper localhost:2181 \
    --delete --topic telemetry
```

Appendix C

Appendix: Other contributions

I had the privilege to work on interesting side projects during my EngD. The side projects were a way to continue with my learning path and professional and personal self-development [91], [84]. I often used the gained knowledge from these side projects in the projects presented in the chapters of this thesis.

In 2017 I started volunteering at a data science charity based in London called DataKindUK[1] and participating in applied collaborative data science events. I developed one of my interests since I was a recent graduate, which is Data for Good or the use of data science methods for the Common Good [40].

In 2018, two position papers were written by Diego on Data for Good. *The Case for Data For Good*[2] and *Scalable Digital Volunteering: A Data for Social Good Marketplace*[3] where it is presented why is necessary to scale the volunteering work using digital and collaborative platforms. In the other paper it is explained the design of a platform for collaboration on data science projects for the common Good.

The design of that platform data science platform for Data for Good led to the collaboration with a working group about that was about to start working on a blueprint for a Data Safe Haven. The overlapping of Data Safe Haven ideas and a data platform for processing and storing data for data science projects led me to join the group. Diego was invited as a Visiting Researcher at the Alan Turing Institute, we he contributed to the work published on this paper on *Design choices for productive, secure, data-intensive research at scale in the cloud* [86].

Around the same time, Diego joined the development group of a new and promising machine learning library written in the Julia[4] programming language. He was one of the first maintainers of the library *MLJ: A Julia package for composable machine learning* [93]. The experience from the work on this open-source project was key for the designs of the libraries presented in Chapters 2 and 6.

During the EngD, I was constantly learning about new technologies and ar-

[1] Official website of DataKindUK, https://datakind.org.uk (accessed 25 October 2021).

[2] Downloadable at https://darenasc.github.io/files/TheCaseForDataForGood.pdf (accessed 25 October 2021).

[3] Downloadable at https://darenasc.github.io/files/ScalableDigitalVolunteering.pdf (accessed 25 October 2021).

[4] Official website of the Julia programming language, https://julialang.org (accessed 25 October 2021).

chitectures, practising my coding skills. A way to practise was to explain the exciting topics that I would find are of interest to people I talked to. I wrote twenty-eight blog posts about data science, machine learning and big data in a technical conference blog of a data science conference that received contributions (https://opendatascience.com/user/darenasc/). I recorded three episodes of podcasts interviewing people in the world of data science and data for Good. I wrote a series of three posts about Data Science for Good.

In 2018 and 2019, Diego co-organised with other postgraduate students from the University of Edinburgh, an annual conference called *Thinking Chile in Edinburgh*[5]. Inviting researchers living in the UK to present their research related to Latin American countries and the region. The 2018 version[6] had two tracks: 1) *Creativity, Culture and Society and Health* and 2) *Environment, and Technology for Social Development.* The 2019 version[7] had four tracks: 1) *One Health for Latin American Development*, 2) *Sustainable Cities in Latin America*, 3) *Artificial Intelligence* and 4) *Arts, Culture and Society.*

Since 2019, I started co-hosting a podcast about artificial intelligence and data science called EscuchAI[8]. The podcast aims to inform the public about topics in artificial intelligence and data science in layman terms and spread knowledge to interested people who may not have the background to get into these topics.

In 2020, Diego's interest in Data Science for Social Good took him to be one of the Technical Mentors at the Data Science for Social Good summer program[9] in the UK organised by the Alan Turing Institute and The University of Warwick. And collaborating with the organisation of 2019, 2020, and 2021 versions of the program in the UK.

[5] Video resource presenting the Conference, https://vimeo.com/319034557 (accessed 26 October 2021).

[6] Thinking Chile website 2018, https://sites.google.com/view/thinkingchile/page (accessed 26 October 2021).

[7] https://sites.google.com/view/thinking-chile-2019/page

[8] ”Escucha” means ”to listen” and with the suffix AI for artificial intelligence. The episodes are available at this link http://escuchai.com (accessed 26 October 2021). The welcome pack of the conference is available on this link https://drive.google.com/file/d/11eR7Y15iESaS72FUxZ1W4AOXpg38Pb5q/view (accessed 26 October 2021).

[9] Website of the DSSGxUK 2019 in the Alan Turing Institute website, https://www.turing.ac.uk/collaborate-turing/data-science-social-good (accessed 26 October 2021).

Bibliography

[1] Edgar F Codd. "A relational model of data for large shared data banks". In: *Communications of the ACM* 13.6 (1970), pp. 377–387.

[2] Peter Pin-Shan Chen. "The entity-relationship model—toward a unified view of data". In: *ACM transactions on database systems (TODS)* 1.1 (1976), pp. 9–36.

[3] John W Tukey. *Exploratory data analysis*. Vol. 2. Reading, Mass., 1977.

[4] George EP Box, William H Hunter, Stuart Hunter, et al. *Statistics for experimenters*. Vol. 664. John Wiley and sons New York, 1978.

[5] John Norman Richard Jeffers. *Design of experiments*. Vol. 1. Institute of Terrestrial Ecology, 1978.

[6] *Assigned numbers*. RFC 790. Sept. 1981. DOI: `10.17487/RFC0790`. URL: `https://rfc-editor.org/rfc/rfc790.txt`.

[7] Michael Stonebraker and Lawrence A Rowe. *The design of Postgres*. Vol. 15. 2. ACM, 1986.

[8] J-L Hainaut, Muriel Chandelon, Catherine Tonneau, and Michel Joris. "Contribution to a theory of database reverse engineering". In: *[1993] Proceedings Working Conference on Reverse Engineering*. IEEE. 1993, pp. 161–170.

[9] Jim Melton and Alan R Simon. *Understanding the new SQL: a complete guide*. Morgan Kaufmann, 1993.

[10] Frederick P Brooks Jr. *The mythical man-month: essays on software engineering*. Pearson Education, 1995.

[11] William H Inmon. "What is a data warehouse". In: *Prism Tech Topic* 1.1 (1995), pp. 1–5.

[12] William H Inmon, Claudia Imhoff, and Greg Battas. *Building the operational data store*. John Wiley & Sons, Inc., 1995.

[13] UM Feyyad. "Data mining and knowledge discovery: Making sense out of data". In: *IEEE expert* 11.5 (1996), pp. 20–25.

[14] Ralph Kimball. *The data warehouse toolkit: practical techniques for building dimensional data warehouses*. John Wiley & Sons, Inc., 1996.

[15] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rüdiger Wirth. "The CRISP-DM user guide". In: *4th CRISP-DM SIG Workshop in Brussels in March*. Vol. 1999. 1999.

[16] Axel Daneels and Wayne Salter. "What is SCADA?" In: (1999).

[17] Armando Fox and Eric A Brewer. “Harvest, yield, and scalable tolerant systems”. In: *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems*. IEEE. 1999, pp. 174–178.

[18] Joseph M Juran, A Blanton Godfrey, Robert E Hoogstoel, and Edward G Schilling. *Juran's quality handbook 5th ed.* 1999.

[19] Eric A Brewer. “Towards robust distributed systems”. In: *PODC*. Vol. 7. 10.1145. Portland, OR. 2000, pp. 343477–343502.

[20] Hausi A Müller, Jens H Jahnke, Dennis B Smith, Margaret-Anne Storey, Scott R Tilley, and Kenny Wong. “Reverse engineering: a roadmap”. In: *Proceedings of the Conference on the Future of Software Engineering*. 2000, pp. 47–60.

[21] Rüdiger Wirth and Jochen Hipp. “CRISP-DM: Towards a standard process model for data mining”. In: *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*. Springer-Verlag London, UK. 2000, pp. 29–39.

[22] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. “Why and where: A characterization of data provenance”. In: *International conference on database theory*. Springer. 2001, pp. 316–330.

[23] Andrew Y Ng, Michael I Jordan, and Yair Weiss. “On spectral clustering: Analysis and an algorithm”. In: *Advances in neural information processing systems*. 2002, pp. 849–856.

[24] Leo L Pipino, Yang W Lee, and Richard Y Wang. “Data quality assessment”. In: *Communications of the ACM* 45.4 (2002), pp. 211–218.

[25] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. “The Google file system”. In: *ACM SIGOPS operating systems review*. Vol. 37. 5. ACM. 2003, pp. 29–43.

[26] Eric Evans and Eric J Evans. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004.

[27] Holger Krekel, Bruno Oliveira, Ronny Pfannschmidt, Floris Bruynooghe, Brianna Laugher, and Florian Bruhin. *pytest x.y.* 2004. URL: `https://github.com/pytest-dev/pytest`.

[28] Alexey Tsymbal. “The problem of concept drift: definitions and related work”. In: *Computer Science Department, Trinity College Dublin* 106.2 (2004), p. 58.

[29] Christopher M Bishop. “Machine learning and pattern recognition”. In: *Information science and statistics. Springer, Heidelberg* (2006).

[30] Todd G Nick. “Descriptive statistics”. In: *Topics in Biostatistics*. Springer, 2007, pp. 33–52.

[31] Ana Isabel Rojão Lourenço Azevedo and Manuel Filipe Santos. “KDD, SEMMA and CRISP-DM: a parallel overview”. In: *IADS-DM* (2008).

[32] Tim Brown et al. “Design thinking”. In: *Harvard business review* 86.6 (2008), p. 84.

[33] Kitty J Jager, Paul C Van Dijk, Carmine Zoccali, and Friedo W Dekker. “The analysis of survival data: the Kaplan–Meier method”. In: *Kidney international* 74.5 (2008), pp. 560–565.

[34] Dean J Ghemawat S MapReduce. “simplified data processing on large clusters [J]”. In: *Commun. ACM* 1 (2008), pp. 107–113.

[35] JinGang Shi, YuBin Bao, FangLing Leng, and Ge Yu. “Study on log-based change data capture and handling mechanism in real-time data warehouse”. In: *2008 international conference on computer science and software engineering*. Vol. 4. IEEE. 2008, pp. 478–481.

[36] Panos Vassiliadis. “A survey of extract–transform–load technology”. In: *International Journal of Data Warehousing and Mining (IJDWM)* 5.3 (2009), pp. 1–27.

[37] Kristin Weber, Boris Otto, and Hubert Österle. “One size does not fit all—a contingency approach to data governance”. In: *Journal of Data and Information Quality (JDIQ)* 1.1 (2009), pp. 1–27.

[38] Vijay Khatri and Carol V Brown. “Designing data governance”. In: *Communications of the ACM* 53.1 (2010), pp. 148–152.

[39] Skipper Seabold and Josef Perktold. “statsmodels: Econometric and statistical modeling with python”. In: *9th Python in Science Conference*. 2010.

[40] Diego Arenas Contreras. “Strategic Business Intelligence for NGOs”. In: *CEPIS UPGRADE Vol. XII, issue No. 3, July 2011*. Council of European Professional Informatics Societies. 2011, pp. 38–42.

[41] Jay Kreps, Neha Narkhede, Jun Rao, et al. “Kafka: A distributed messaging system for log processing”. In: *Proceedings of the NetDB*. 2011, pp. 1–7.

[42] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. “Scikit-learn: Machine learning in Python”. In: *the Journal of machine Learning research* 12 (2011), pp. 2825–2830.

[43] Raymond PL Buse and Thomas Zimmermann. “Information needs for software development analytics”. In: *2012 34th International Conference on Software Engineering (ICSE)*. IEEE. 2012, pp. 987–996.

[44] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Y Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. “Finding related tables”. In: (2012).

[45] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. “Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing”. In: *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association. 2012, pp. 2–2.

[46] Matei Zaharia, Tathagata Das, Haoyuan Li, Scott Shenker, and Ion Stoica. “Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters.” In: *HotCloud* 12 (2012), pp. 10–10.

[47] David Loshin. *Big data analytics: from strategic planning to enterprise integration with tools, techniques, NoSQL, and graph*. Elsevier, 2013.

[48] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al. “Storm@ twitter”. In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM. 2014, pp. 147–156.

[49] Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. "OpenML: networked science in machine learning". In: *ACM SIGKDD Explorations Newsletter* 15.2 (2014), pp. 49–60.

[50] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. "Apache flink: Stream and batch processing in a single engine". In: *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36.4 (2015).

[51] Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M Patel, Karthik Ramasamy, and Siddarth Taneja. "Twitter heron: Stream processing at scale". In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM. 2015, pp. 239–250.

[52] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. "Hidden technical debt in machine learning systems". In: *Advances in neural information processing systems*. 2015, pp. 2503–2511.

[53] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. "Tensorflow: A system for large-scale machine learning". In: *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. 2016, pp. 265–283.

[54] Diego Arenas Contreras. "Automatic hierarchical data extraction from relational databases". MA thesis. The University of Edinburgh, 2016.

[55] Andrew Crotty, Alex Galakatos, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. "The case for interactive data exploration accelerators (IDEAs)". In: *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*. 2016, pp. 1–6.

[56] J. Dunn. *Introducing FBLearner Flow: Facebook's AI backbone*. 2016. URL: `https://code.fb.com/core-data/%20introducing-fblearner-flow-facebook-s-ai-backbone/` (visited on 05/09/2016).

[57] Raul Castro Fernandez, Ziawasch Abedjan, Samuel Madden, and Michael Stonebraker. "Towards large-scale data discovery: position paper". In: *Proceedings of the Third International Workshop on Exploratory Search in Databases and the Web*. 2016, pp. 3–5.

[58] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. "The emerging role of data scientists on software development teams". In: *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE. 2016, pp. 96–107.

[59] Zachary C Lipton. "The mythos of model interpretability". In: *arXiv preprint arXiv:1606.03490* (2016).

[60] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. "Mllib: Machine learning in apache spark". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1235–1241.

[61] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “Why should i trust you?: Explaining the predictions of any classifier”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM. 2016, pp. 1135–1144.

[62] Manasi Vartak, Harihar Subramanyam, Wei-En Lee, Srinidhi Viswanathan, Saadiyah Husnoo, Samuel Madden, and Matei Zaharia. “ModelDB: a system for machine learning model management”. In: *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*. ACM. 2016, p. 14.

[63] Yoji Yamato, Hiroki Kumazaki, and Yoshifumi Fukumoto. “Proposal of Lambda Architecture Adoption for Real Time Predictive Maintenance”. In: *Computing and Networking (CANDAR), 2016 Fourth International Symposium on*. IEEE. 2016, pp. 713–715.

[64] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, et al. “Tfx: A tensorflow-based production-scale machine learning platform”. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2017, pp. 1387–1395.

[65] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. “Clipper: A Low-Latency Online Prediction Serving System.” In: *NSDI*. 2017, pp. 613–627.

[66] Derek Doran, Sarah Schulz, and Tarek R Besold. “What does explainable AI really mean? A new conceptualization of perspectives”. In: *arXiv preprint arXiv:1710.00794* (2017).

[67] Finale Doshi-Velez and Been Kim. “Towards a rigorous science of interpretable machine learning”. In: *arXiv preprint arXiv:1702.08608* (2017).

[68] Ted Dunning and Ellen Friedman. *Machine Learning Logistics. Model Management in the Real World*. Wiley, 2017.

[69] J. Hermann and M. D. Balso. *Meet Michelangelo: Uber’s machine learning platform*. 2017. URL: `https://code.fb.com/core-data/%20introducing-fblearner-flow-facebook-s-ai-backbone/` (visited on 09/05/2017).

[70] Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E Gonzalez, Michael I Jordan, and Ion Stoica. “RLlib: Abstractions for distributed reinforcement learning”. In: *arXiv preprint arXiv:1712.09381* (2017).

[71] Scott M Lundberg and Su-In Lee. “A unified approach to interpreting model predictions”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 4765–4774.

[72] Douglas C Montgomery. *Design and analysis of experiments*. John wiley & sons, 2017.

[73] Robert Nishihara, Philipp Moritz, Stephanie Wang, Alexey Tumanov, William Paul, Johann Schleier-Smith, Richard Liaw, Mehrdad Niknami, Michael I Jordan, and Ion Stoica. “Real-time machine learning: The missing pieces”. In: *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*. ACM. 2017, pp. 106–110.

[74] Brian Okken. *Python Testing with Pytest: Simple, Rapid, Effective, and Scalable*. Pragmatic Bookshelf, 2017.

[75] Laurel Orr, Magda Balazinska, and Dan Suciu. “Probabilistic database summarization for interactive data exploration”. In: *arXiv preprint arXiv:1703.03856* (2017).

[76] Ashish Thusoo and Joydeep Sen Sarma. “Creating a Data-Driven Enterprise with DataOps”. In: (2017).

[77] Daniel Kang, Deepti Raghavan, Peter Bailis, and Matei Zaharia. “Model assertions for debugging machine learning”. In: *NeurIPS MLSys Workshop*. 2018.

[78] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. “Tune: A research platform for distributed model selection and training”. In: *arXiv preprint arXiv:1807.05118* (2018).

[79] Robert C Martin, James Grenning, and Simon Brown. *Clean architecture: a craftsman’s guide to software structure and design*. Prentice Hall, 2018.

[80] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. “Ray: A Distributed Framework for Emerging {AI} Applications”. In: *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 2018, pp. 561–577.

[81] Pedro Saleiro, Benedict Kuester, Abby Stevens, Ari Anisfeld, Loren Hinkson, Jesse London, and Rayid Ghani. “Aequitas: A Bias and Fairness Audit Toolkit”. In: *arXiv preprint arXiv:1811.05577* (2018).

[82] Sebastian Schelter, Felix Biessmann, Tim Januschowski, David Salinas, Stephan Seufert, Gyuri Szarvas, Manasi Vartak, Samuel Madden, Hui Miao, Amol Deshpande, et al. “On challenges in machine learning model management”. In: *Data Engineering* (2018), p. 5.

[83] Sean J Taylor and Benjamin Letham. “Forecasting at scale”. In: *The American Statistician* 72.1 (2018), pp. 37–45.

[84] Data Study Group team. *Data Study Group Final Report: Codecheck*. Sept. 2018. DOI: `10.5281/zenodo.1415344`. URL: `https://doi.org/10.5281/zenodo.1415344`.

[85] Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, et al. “Accelerating the Machine Learning Lifecycle with MLflow”. In: *Data Engineering* (2018), p. 39.

[86] Diego Arenas, Jon Atkins, Claire Austin, David Beavan, Alvaro Cabrejas Egea, Steven Carlysle-Davies, Ian Carter, Rob Clarke, James Cunningham, Tom Doel, et al. “Design choices for productive, secure, data-intensive research at scale in the cloud”. In: *arXiv preprint arXiv:1908.08737* (2019).

[87] J. Hermann and M. D. Balso. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*. 2019. URL: `https://christophm.github.io/interpretable-ml-book/` (visited on 04/12/2019).

[88] Michael Muller, Ingrid Lange, Dakuo Wang, David Piorkowski, Jason Tsay, Q Vera Liao, Casey Dugan, and Thomas Erickson. "How Data Science Workers Work with Data: Discovery, Capture, Curation, Design, Creation". In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems.* 2019, pp. 1–15.

[89] Alexander Ratner, Dan Alistarh, Gustavo Alonso, Peter Bailis, Sarah Bird, Nicholas Carlini, Bryan Catanzaro, Eric Chung, Bill Dally, Jeff Dean, et al. "SysML: The New Frontier of Machine Learning Systems". In: *arXiv preprint arXiv:1904.03257* (2019).

[90] Md Amran Siddiqui, Alan Fern, Thomas G Dietterich, and Weng-Keen Wong. "Sequential Feature Explanations for Anomaly Detection". In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 13.1 (2019), p. 1.

[91] Data Study Group team. *Data Study Group Final Report: Global bank.* Feb. 2019. DOI: 10.5281/zenodo.2557809. URL: https://doi.org/10.5281/zenodo.2557809.

[92] Michael Armbrust, Tathagata Das, Liwen Sun, Burak Yavuz, Shixiong Zhu, Mukul Murthy, Joseph Torres, Herman van Hovell, Adrian Ionescu, Alicja Łuszczak, et al. "Delta lake: high-performance ACID table storage over cloud object stores". In: *Proceedings of the VLDB Endowment* 13.12 (2020), pp. 3411–3424.

[93] Anthony D Blaom, Franz Kiraly, Thibaut Lienart, Yiannis Simillides, Diego Arenas, and Sebastian J Vollmer. "MLJ: A Julia package for composable machine learning". In: *arXiv preprint arXiv:2007.12285* (2020).

[94] Daniel Kang, Deepti Raghavan, Peter Bailis, and Matei Zaharia. "Model assertions for monitoring and improving ML models". In: *arXiv preprint arXiv:2003.01668* (2020).

[95] Cameron Davidson-Pilon, Jonas Kalderstam, Noah Jacobson, Sean Reed, Ben Kuhn, Paul Zivich, Mike Williamson, AbdealiJK, Deepyaman Datta, Andrew Fiore-Gartland, Alex Parij, Daniel WIlson, Gabriel, Luis Moneda, Arturo Moncada-Torres, Kyle Stark, Harsh Gadgil, Jona, JoseLlanes, Karthikeyan Singaravelan, Lilian Besson, Miguel Sancho Peña, Steven Anton, Andreas Klintberg, GrowthJeff, Javad Noorbakhsh, Matthew Begun, Ravin Kumar, Sean Hussey, and Skipper Seabold. *CamDavidsonPilon/lifelines: 0.26.0.* Version 0.26.0. May 2021. DOI: 10.5281/zenodo.4816284. URL: https://doi.org/10.5281/zenodo.4816284.