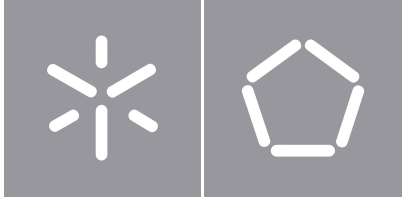

**University of Minho**
School of Engineering

Miguel Domingos Silva Pinto Oliveira

# Development of statistical cycle-time analysis tool in tire building

Masters Dissertation
Master's in Industrial Engineering and Management
Dissertation supervised by
**Ana Cristina da Silva Braga**

january 2024

# Copyright and Terms of Use for Third Party Work



## License granted to users of this work:

# Acknowledgements

I would like to express my deep gratitude and appreciation to professor Ana Braga for her insightful guidance, patience and encouragement during the extend of this dissertation. This endeavour wouldn't be possible without her support and virtues, It was an amazing experience both at academic level and personal growth. I will be forever grateful for this.

My grateful appreciation is also extended to Ricardo Rodrigues for his guidance, friendship and mentorship throughout all my childhood and early adulthood which culminated with this huge milestone. To Ana Rita Silva, for the mentorship and for granting me the opportunity to work alongside her and trust to fulfill this project within targets. To all the DEI team, directed by Tiago Batista, I deeply appreciate your friendship, patience and humanity. This crowning achievement is dedicated to each and everyone of you.

Finally, I would like to thank my family as well for paying my tuition and help me financially throughout the degree.

# Statement of Integrity

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

University of Minho, Braga, january 2024

Miguel Domingos Silva Pinto Oliveira

# Abstract


## Development of statistical cycle-time analysis tool in tire building

This dissertation covers the development of an end-to-end data science application in the tire industry and supports the increasing importance of data-driven solutions in the context of Industry 4.0. Incorporating evidence from research papers and personal correspondence, this dissertation demonstrates it is possible to create strategic value from unstructured data, which is often a neglected by-product of industrial activities.

The value proposition is to capture the real-time data from industrial programmable logic circuits (PLC) and store it in structured databases, compute industry-relevant metrics, and build a dashboard to support managerial decisions over productivity, and machine up-time and efficiency. Pinpointing a set of pertinent features of the application can (1) create an up-to-date database with support for multiple connectors, (2) develop black-box algorithms to provide statistical measures, (3) increase awareness and visibility about PLC settings. To this end, multiple technologies & methods were employed to implement a data pipeline, to ensure numerical solutions to statistical problems with accuracy, and finally, to display pertinent data in a data visualization dashboard according to a set of user requirements.



**Keywords** data-pipeline, statistics, tire-industry, database-modeling, feature-engineering, industry 4.0

# Resumo


## Desenvolvimento de uma ferramenta estatística para análise de tempos de ciclo na montagem de pneus

Esta dissertação aborda o desenvolvimento de uma aplicação com base nos principios de ciência de dados na indústria de pneus e apoia a crescente importância de soluções orientadas por dados no contexto da Indústria 4.0. Incorporando evidências de artigos cientificos, esta dissertação demonstra que é possível criar valor estratégico a partir de dados não estruturados, frequentemente negligenciados como subproduto das atividades industriais.

A proposta de valor é capturar dados em tempo real dos programmable logic circuits e armazená-los em bases de dados estruturadas, calcular métricas relevantes para a indústria e criar um painel de apoio a decisões com especial enfase em produtividade e eficiência industrial. A identificação de um conjunto de características pertinentes da aplicação permite (1) criar uma base de dados atualizada com suporte para múltiplos conectores diferentes, (2) desenvolver algoritmos 'caixa-preta' para fornecer medidas estatísticas e (3) aumentar a conscientização e visibilidade sobre as configurações do PLC. Neste sentido, foram empregues várias tecnologias e métodos para implementar um *pipeline* de dados, garantir soluções numéricas para problemas estatísticos e, finalmente, exibir dados pertinentes num painel de visualização de dados de acordo com um conjunto de requisitos do utilizador.



**Keywords** bases de dados, inferência estatistica, tire-industry, modelação de dados, indústria 4.0

# Contents

# List of Figures

# Chapter 1

# Introduction

Tire building is an important step in tire production value chain in the Lousado plant especially with the increasing production necessities and tire complexity. The investigation of the manufacturing cycle time is thus a necessary step to ensure reduced losses and increased machinery efficiency. In this work, it is provided a framework to analyse manufacturing throughput time and to compare across different machines and tire types with the main purpose of identifying improvement opportunities and taking actions to diminish their impact. As in hereof, we ought to develop a method to pursue machine optimization by identifying said opportunities to improve throughput instead of increasing the production capacity by acquiring new machines. The methods used for such tasks are data driven tools sustained on statistical methods and data analytics. Among them, the techniques used were maximum likelihood, kernel density estimation, support vector machine and bootstrap. Several tools were required due to the enormous data volume and heterogeneity between cycle phases. These methods were implemented using SQL, Python and python packages.

## 1.1 Motivation

This work was developed as an integral part of the master thesis in Industrial and Management Engineering, Minho University. The predictability and estimation of cycle times represents an important baseline in the determination of other downstream processes and requirements such as Material Requirements Planning (MRP), production planning and key performance indicators (Overall Equipment Effectiveness). Statistical analysis is a toolset based on a scientific approach which solves these problems. From multivariate regression models, machine learning, distribution modelling there are distinct solutions in the literature as theorised solutions. The Continental Lousado plant is dedicated to the manufacturing of specialised tires for the global market. The production unit in this study was the passenger tire building (PLT) and the business solution proposed is the development of a decision support system reporting tool.

## 1.2 Objectives

The main objective of this work is the development of a decision support system infrastructure with the following user requirements:

- Cycle time data pipe-lining, including extract, transform and loading workflows; handling multiple data sources; scheduling, task atomicity and dependencies between directed acyclic graphs (DAGs);
- Analysis of cycle element times using statistical methods that will be defined later to meet user technical requirements;
- Development of a support data base with support for several business intelligence connectors, linking producer processes to consumers;
- Definition of KPI's and reporting metrics;

## 1.3 Research Methodology

In order to follow through with the research objectives outlined in the section above, several studies were made with varied natures and contexts. Insofar as this dissertation is concerned, we focused in exploratory and descriptive studies to understand the dynamics of the underlying variables and their relationships with the objectives.

Consequently, the strategy used was action research methods based on a number of qualitative and quantitative data in both longitudinal and cross-section studies. The data will be manipulated in order to achieve the end proposed end results and ultimately contribute to initiatives taken to reduce cycle times. All mathematical models, exploratory analysis and descriptive studies will be implemented in Python.

## 1.4 Dissertation Structure

In the 2nd chapter it is made a literature review pertaining to concepts and methodologies used throughout the development of this dissertation. In this chapter it is introduced the motivation behind the project, its purpose and objectives, the research methodology as well as the document structure. The chapter 3 and 4 addresses the context of the project, describing the industry environment, technologies and project methodology. The chapter 5 discusses the research questions which is the analysis of the triggers in the tire building process, from product requirements to deployment. Starting from the business requirements

and available data, eloping into the formulation of concrete ideas and methods to tackle such issues through brainstorming. The stages are, first and foremost the establishment of a database of pertaining metrics to feed downstream processes and the deployment of such data using tools and data visualisation techniques allowing for rich client interaction. It also discusses the inference of cycle times of the tire building process. Once again with a similar approach, starting from business requirements to product deployment. The stages are the careful selection and processing of the data followed by the employment of several statistical inference algorithms. In chapter 6, we discuss the implementation of the web application going into details about pertaining questions about layout and user interface. Ultimately, in chapter 7 a succinct overview of the principal conclusions and findings related to this work are presented as well as future work.

# Chapter 2

# State of the Art

## 2.1 Support Vector Machine

The support vector machine (SVM) was developed in 1995 by Vladimir Vapnik based on the Vapnik-Chervonenkis theory and soon gained traction in the industry [Noble, 2006]. Traditional neural network approaches suffered severe difficulties with generalization and producing models with efficient separability of non-linear regions. SVM is a method particularly well suited to handle data-set classification by separating positive from negative values with the hyperplane with the maximum margin [Wang, 2005]. It can perform linear separation if feasible and non-linear separation with the kernel trick, implicitly mapping inputs into a higher dimensional feature space. A margin is the distance between the hyperplane and the nearest positive and negative samples. The picture in figure 1 shows a linear SVM.

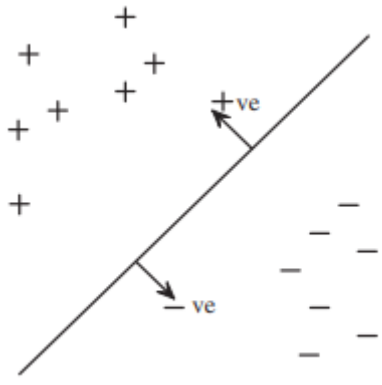


Figure 1: Linear SVM denoting the separating hyperplane (extracted from [Liu et al., 2019]).

SVM is traditionally considered to be a supervised learning model relying on previously available labeled data sets. It belongs to the class of kernel machines which are algorithms capable of operating with kernels. There can be several hyper-planes that successfully classify the data, however, the most reasonable choice must be the one that represents the largest separation or margin between the classes. Such a hyperplane

if exists is known as the maximum-margin hyperplane. Intuitively, a good class separation is achieved by the hyperplane that has the larger margin and thus offers the best generalization error [Wang, 2005]. Figure 2, both H2 and H3 are suitable solutions however H3 is the maximum margin hyper-parameter as depicted in the figure 2.

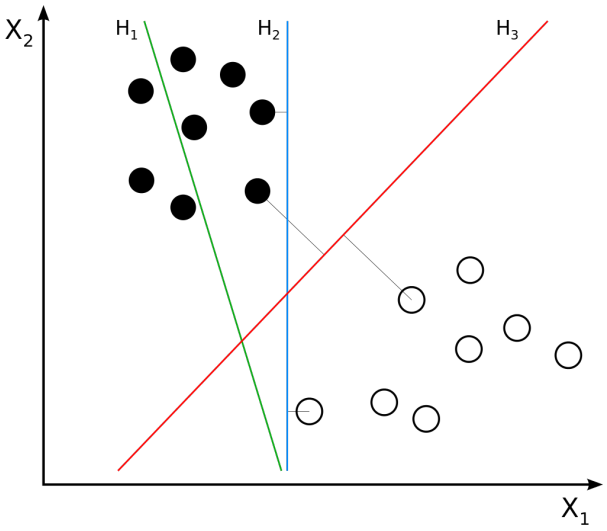


Figure 2: Comparison between hyper-planes Hua and Sun, 2001

### 2.1.1 Mathematical Formulation

Cristianini and Ricci considered the class of hyper-planes in some feature space $\mathcal{H}$ [Cristianini and Ricci, 2008]:

$$\langle w, x\rangle + b = 0 \text{ where } w \in \mathcal{H}\ , b \in \mathbb{R}$$
$$f(x) = \text{sgn}(\langle w, x\rangle + b) \tag{2.1}$$

where sgn is the sign function which returns the sign of the real number

The data is correctly classified if $y_i(\langle w, x\rangle + b) > 0$, since $(\langle w, x\rangle + b)$ should be positive when $y_i = +1$ and negative if -1. This leads to the definition of the canonical hyper-planes by setting the closest points in each side at which $y_i(\langle w, x\rangle + b) = 1$ and $y_i(\langle w, x\rangle + b) = -1$ respectively.

The optimal condition is thus to maximize the margin $\gamma$ [Wang, 2005]. The margin is computed from the two points in the canonical hyper-planes and is half of the marginal band. Let $x_1$ and $x_2$ be those points, $\langle w, x_1\rangle + b = 1$ and $\langle w, x_2\rangle + b = -1$, then $\langle w, (x_2 - x1)\rangle + b = 2$. The marginal band $(x_2 - x_1)\frac{w}{||w||} = \frac{2}{||w||}$. The margin is half of this $\gamma = \frac{1}{||w||}$ [figure 3].

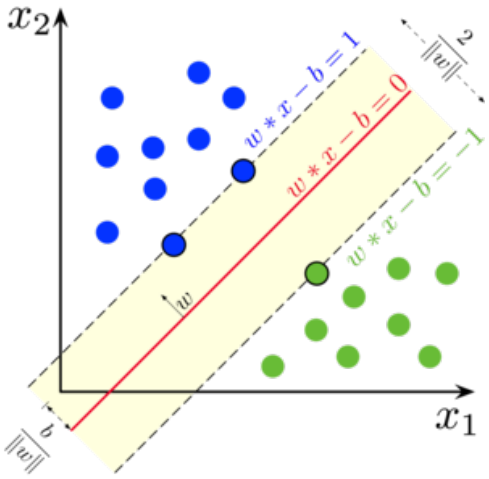


Figure 3: Canonical hyper-planes and marginal band (extracted from [Pecha and Horák, 2020])

Because the goal of the SVM is to improve generalization and learning stability, the optimal condition for the hyperplane should maximize the margin value according to [Cristianini and Ricci, 2008]. This can be posed as a quadratic constrained minimization problem:

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}||w^2|| \\ \text{subject to} \quad & y_i(\langle w, x \rangle + b) > 1 \text{ for all i = 1, ..., m} \end{aligned} \tag{2.2}$$

The solution to these problems follows the Lagrange multipliers and a few assumptions such as Karush-Kuhn-Tucker (KKT) conditions and duality.

### 2.1.2 Soft Margin Classifier

Unlike the method explained above, the soft margin classifier allows some observations to lie on the wrong side of the support vectors or even hyperplane at can be seen in figure 4.

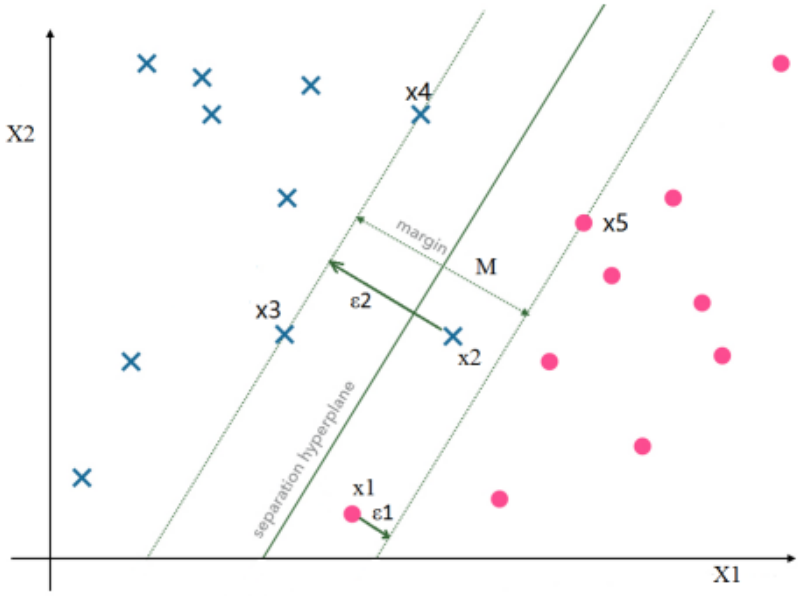


Figure 4: Soft margin regularization with observations lying outside of the hyperplane (extracted from [Pecha and Horák, 2020]).

This method is a relaxed formulation of the maximal margin formulation which allows the processing of messy data that cannot be perfectly separated with a hyperplane [Cristianini and Ricci, 2008]. The new mathematical formula accounts for a slack variable $\epsilon$.

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}||w^2|| \\ \text{subject to} \quad & y_i(\langle w, x\rangle + b) > 1(1-\epsilon) \text{ for all i = 1, ..., m} \\ & \epsilon > 0 \end{aligned} \tag{2.3}$$

### 2.1.3 Non-Linear Classifier and Kernel Trick

$$K(x, x') = \exp^{-\frac{(x-x')^2}{2\sigma^2}} \tag{2.4}$$

Gaussian Radial Basis Function (RBF) kernel is a stationary kernel, invariant to translations. For a single parameter, it exhibits isotropic property, which implies that scaling of one parameter leads to automatic scaling of all other parameters. The adjustable parameter $\sigma$ is tuned according to the nature of the problem [Kwak, 2013]. If set to a higher value, the kernel behaves almost linearly, causing an overestimation of the problem, and the nonlinear higher dimensional projection no longer holds. Similarly, if set to a very small value, the regularization function is affected and this underestimation makes the decision boundary sensitive to noise in the training data. It controls the flexibility of the classifier, as shown in the figure 5.

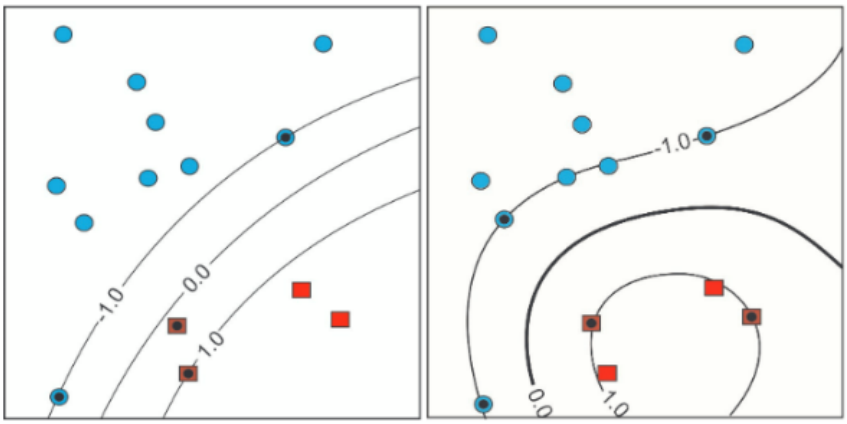


Figure 5: Gaussian kernel for $\sigma = 20$ (left) and $\sigma = 1$ (right) (extracted from [Cristianini and Shawe-Taylor, 2000])

The introduction of a kernel and the implied mapping of the input space into a feature space is known as a kernel trick or kernel substitution.

## 2.2 Maximum Likelihood Estimation

The statistical distributions of the occurrences are quite beneficial while trying to gather some relevant data for stochastic events. Although creating histograms can be useful for identifying trends in data, many observations must be made to get a smooth and accurate approximation of the distribution. So, when any limitations on the form of the distribution are known, it is generally more practical to utilize a parametric model to estimate the distribution. By defining a set of statistical parameters based on the observed data, a distribution is here estimated.

The maximum likelihood estimation (MLE) method is an optimization method relying on the likelihood function to find the parameter of parametric models or distributions that maximize the underlying likelihood probability function. That being said, it answers the question: for which parameter(s) value "does the observed data have the biggest probability?".

### 2.2.1 Mathematical Formulation

Maximum likelihood is the most common technique for estimating parameters associated with discrete stochastic variables (or probability density functions of continuous stochastic variables) based on $j$ observations independently sampled from the distribution.

The general formulation for the probability density function follows:

$$P(X < r|\theta) = \int p(x|\theta)dX \tag{2.5}$$

The goal is to find the parameter $\theta$ which maximizes the likelihood function where $X$ is the continuous stochastic variable [Eliason, 1993].

$$\mathcal{L}(\theta) = \prod_{1}^{j} P(X = x_i|\theta) \tag{2.6}$$

$$\text{argmax}_\theta \mathcal{L}(\theta|x_j) \tag{2.7}$$

### 2.2.2 Normal Distribution

In the case of normal distribution, the objective of the MLE function is to estimate the mean and variance parameters.

$$p(x|\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{2.8}$$

Where $x$ can be substituted for each $X_1$, $X_2$, ..., $X_j$ for $j$ observations. The likelihood $\mathcal{L}$ can be written as:

$$\mathcal{L}(\mu, \sigma) = p(X_1|\mu, \sigma)p(X_2|\mu, \sigma)...p(X_j|\mu, \sigma) \tag{2.9}$$

$$\mathcal{L}(\mu, \sigma) = \prod_{1}^{j} P(X = x_i|\theta) \tag{2.10}$$

According to Eliason [Eliason, 1993], this equation can be simplified with the log-likelihood function and solved afterward in the form of convex continuous optimization:

$$\frac{\partial}{\partial\mu}\mathcal{L}(\mu, \sigma) = 0 \text{ and } \frac{\partial}{\partial\sigma}\mathcal{L}(\mu, \sigma) = 0 \tag{2.11}$$

## 2.3 Kernel Density Estimation

The probability density function (pdf), shows how the entire mass with a 100% probability is distributed along the x-axis, or over the values of an $X$ random variable. However, the histogram, the oldest pdf empirical representation, is a very arbitrary structure because it depends on the initial point chosen and the number of class intervals (bins) into which a sample's range is divided. The histogram is plagued by its original sin, regardless of the class selection method used: data binning, in which the data are stripped of their individual positions and given bin (interval) locations instead.

In kernel estimation of the probability density function, these issues are not present. Kernel estimate is not a new technique, it was invented 50 years ago by Rosenblatt and Parzen [Stefanski and Carroll, 1990].

### 2.3.1 Mathematical Formulation

Let the series $X_1$, $X_2$, ..., $X_j$ be an independent sample from $j$ observations taken from the population $X$ with an unknown distribution probability $f(x)$. The kernel estimate assigns each i-th sample data point $X_1$ a function $K(X_i, t)$ called a kernel function [Stefanski and Carroll, 1990] the following way:

$$\frac{1}{n}\sum_{1}^{j} K(X_i, t) \tag{2.12}$$

This requires a normalization property:

$$\int K(X_i, t)dt = 1 \tag{2.13}$$

Examples of kernel basis functions are represented in the figure 6.

| Kernel | Definition |
|---|---|
| Epanechnikov | $K(t)=\begin{cases}\frac{3}{4\sqrt{5}}(1-\frac{1}{5}t^2) & \text{for } \lvert t\rvert<\sqrt{5}\\ 0 & \text{for } \lvert t\rvert\geq\sqrt{5}\end{cases}$ |
| Biweight | $K(t)=\begin{cases}\frac{15}{16}(1-t^2)^2 & \text{for } \lvert t\rvert<1\\ 0 & \text{for } \lvert t\rvert\geq 1\end{cases}$ |
| Triangular | $K(t)=\begin{cases}1-\lvert t\rvert & \text{for } \lvert t\rvert<1\\ 0 & \text{for } \lvert t\rvert\geq 1\end{cases}$ |
| Gaussian | $K(t)=\frac{1}{\sqrt{2\pi}}e^{-t^2/2}$ |
| Rectangular | $K(t)=\begin{cases}\frac{1}{2} & \text{for } \lvert t\rvert<1\\ 0 & \text{for } \lvert t\rvert\geq 1\end{cases}$ |

Figure 6: Examples of symmetric kernel basis functions [Dinardo and Tobias, 2001].

### 2.3.2 Parzen Windows

The pdf is estimated for each decomposed region separately called Parzen Windows. This approach associates each random sample $X_i$ with an arbitrary sequence of windows centered at $x_1$, $x_2$, etc. such that the density at each point is different and calculated independently [Terrell and Scott, 1992]. The number of windows is dependent on the sample and disposition:

$$p(x) = \frac{1}{j}\sum_{1}^{j}\frac{1}{h^2}K(\frac{x_i - x}{h}) \tag{2.14}$$

Where once again $j$ is the sample size, $h$ is the dimension of the Parzen Window, $p(x)$ is the probability estimation of the variable $x$.

## 2.4 Bootstrap Method

An important property of this method is that is more robust and thus less sensitive to assumptions than other traditional techniques [Davison and Hinkley, 1997].

In general, if we consider the bootstrap sample $x = (x_1, x_2, ..., x_j)$ that is obtained from randomly sampling j times with replacement from the original data points, and apply this procedure $B$ times then calculate the value of the target statistic $T_s(X)$ based on each sample, we will obtain B estimates $T_s(x_1), T_s(x_2), ..., T_s(x_B)$.

### 2.4.1 Confidence Intervals and Bootstrap Percentile Method

Because bootstrap is flexible and requires no assumptions about the underlying data and distribution, we can estimate the sampling distribution through the frequentist approach. This is the foundation of the procedure according to [Davison and Hinkley, 1997].
The confidence coefficient $\alpha$ corresponds to the relative frequency which the confidence region will include the true parameter value $\theta$. A confidence interval will be defined by the limits $\theta_{\alpha 1}$ and $\theta_{\alpha 2}$.

$$P(\theta_{\alpha_1} < \theta < \theta_{\alpha_2}) = \alpha \tag{2.15}$$

Where $[\theta_{\alpha_1}, \theta_{\alpha_2}]$ is the coverage interval and $\alpha_1$ and $\alpha_2$ represent the left and right tail error probabilities respectively. In the calculation, this is a case of two-sided confidence intervals.
Using this approach with the bootstrap method, we construct a confidence interval for parameter $\theta$ obtaining a bilateral bootstrap interval at (1-$\alpha$) confidence interval.

$$(P(T*_j \leq x|X_j) = \alpha/2, P(T*_j \leq x|X_j) = 1 - \alpha/2) \tag{2.16}$$

where $P(T*_j \leq x|X_j)$ represents the estimated probability quantile for the bilateral bootstrap interval $\alpha$ [Davison and Hinkley, 1997].

# Chapter 3

# Company overview

## 3.1 Company Description

Continental Mabor is part of the group Continental AG colloquially referred as Conti is a German multinational automotive parts manufacturing company specializing in tires, brake systems, interior electronics, automotive safety, and chassis components. Continental AG is structured into three divisions: automotive, tires, Contitech Continental and Finanzanalyst [2019]. Its headquarters are located in Hanover.

Continental Mabor was a result of a joint venture between Continental AG and Mabor - Manufatura Nacional de Borracha S.A. in 1989. In 1993, Continental Mabor was owned completely by the German group as a result of a takeover Mabor [2022]. It's currently one of the biggest foreign direct investments in Portugal and one of the biggest gross profit generators in the country.

Ever since its acquisition, the continuous inflow of investment, training, and restructuring projects allowed the manufacturing plant to go from 5000 tires a day in 1990 to 26000 tires a day in 1996. In 2016 was created a new operational facility CST - Commercial Specialty Tires as a result of the LousAgro project which allowed access to a new distinct tire market, the specialty tires mostly used in agricultural machinery (Agro) Mabor [2022].

### 3.1.1 Continental Mabor Organization

The internal organization of the company Continental Mabor is depicted in the following picture. It should be noted this work was developed under the guidance of the Industrial Engineering Department Mabor [2022].

Figure 7: Continental Mabor (Mabor [2022])

- Occupational safety and health
- Management
- Communication
- Solution development center
- Research & development
- Production of passenger light tires
- Quality
- Product industrialization
- Industrial engineering
- Controlling
- Information Technologies
- Logistics & operations
- Human resources

- Industrial safety
- Production of commercial truck tires
- Innovation

### 3.1.2 Tire Structure

The tire structure is projected to increase vehicle safety, stability, traction, cargo support, resistance to dynamical charges produced by the acceleration forces, and damping resulting from road irregularities. For the reasons mentioned, the tire manufacturing process has to ensure high-quality levels. Figure 8 shows the composition of the tire structure.


Figure 8: Tire composition Continental and Finanzanalyst [2019]

**Tread Components**

- The tread (1) is the most external element of the tire and has direct contact with the road surface assuring high millage. It is made of a mixture of synthetic and natural rubber. It can be further divided into the Cap, which is the outermost part of the tread and provides grip on the road surface and wear resistance, the base underneath the cap reduces rolling resistance, and the shoulder which is on the outer edges of the tread Rodgers [2020].

- Cap plies (2) is a layer capable of supporting speeds at high velocity, located directly below the tire's tread. Picture a nylon cord tightly wrapped around inelastic stitching, encapsulated in rubber Rodgers [2020]. This single cord runs seamlessly from one end to the other, spiraling around the tire's circumference without overlap.

- Steel cord belt plies (3) provide the tire with rigidity enhancing shape retention and directional stability and increasing mileage performance.

**Casing Components**

- Textile cord ply (4) is a rubberized rayon or polyester that controls the internal pressure of the tire maintaining its shape.

- Inner liner (5) is an airtight component of butyl rubber that seals the air-filled inner chamber and controls tire pressure.

- Sidewall (6) is the exterior wall of the tire connecting to the tread through the tread shoulder Rodgers [2020]. It is made of natural rubber and protects the casing against external damage such as atmospheric conditions.

**Tire Bead Components**

- Bead reinforcement (7) is made of a strong and heat-resistant synthetic fiber such as nylon and facilitates precise steering direction.

- Bead apex (8) is a stabilizing synthetic rubber

- Bead core (9) is made up of steel wire embedded in rubber. The core ensures the tire sits firmly on the wheel rim Rodgers [2020].

According to the figure, The belt can be either textile, metallic, or a combination of both and is a rubber-coated component that prevents tire expansion at high speeds. A metallic tread contains steel wiring and the textile tread has textile materials such as polyester. The metallic tread ensures directional stability while the textile one aims for tire structural reinforcement.

### 3.1.3 Tire Manufacturing Process

The tire manufacturing process is divided into 5 main stages as demonstrated in the figure 9.

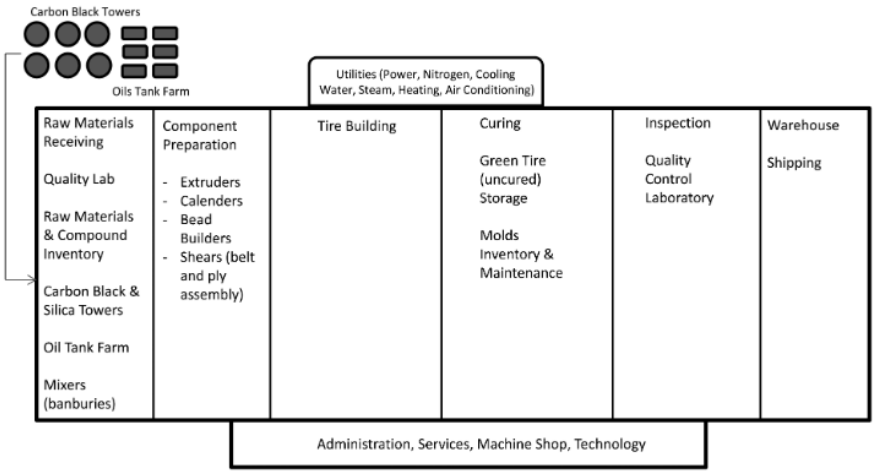


Figure 9: Tire process Rodgers [2020].

### Mixture

The process starts at the mixing stage where all the input materials are quality controlled and verified to meet the production standards. Mixing of a compound is the process of blending with all of the materials in a formulation to produce a homogeneous material Rodgers [2020]. The mixture involves mechanical stress usually through counter-rotating rotors causing pressure and sheer stress to the materials and high temperatures.

### Preparation

The preparation stage is divided into three general operations by machine specificity, namely extrusion, calendaring, and bead building. At Continental, these processes are further divided into cold preparation and hot preparation. The hot preparation is responsible for the fabrication of the tread, sidewalls, and beads. The cold preparation integrates the remaining compounds Rodgers [2020].

### Tire Building

Building is the process of assembling all the components onto a tire building drum resulting in a green tire or uncured tire. The process was divided into two different working stations, one two-stage machine and one single-stage machine where the casing belt and tread assembly were built in different machines.
The tire building operation can be broadly described as follows Rodgers [2020]:

**First stage operation - casing**

- Textile cord ply and inner liner is applied on the circular building drum

- The casing ply and multiply plies are applied to the drum
- Beads and sidewalls are applied
- Turn up and first shaping operation creating the casing

**Second stage operation - tread/belt assembly**

- The breakers and the tread are applied in a rotating drum
- The carcass is inflated and the breakers/tread conjunction is applied to the carcass producing the uncured tire
- Stitchers ensure all components are tightly sealed and the trapped air is removed and deflated

All compounds require additional steps for quality reasons such as inspection and splicing.

### Tire Curing

Tire curing or vulcanization is the fourth stage of operation and the most capital-intensive. The curing consists of two areas, the green tire storage and the press area. The green tire storage holds the uncured tires up to several shifts before they are moved to the press line Rodgers [2020]. Curing is the process of applying pressure and heat to the green tire in a mold in order to give it its final shape. Heated molds are crucial to achieve the final tire dimensions, tread pattern, and vulcanization of all rubber components. Curing cycles can be complex and require an inflow of steam, water, and nitrogen at different stages and temperatures.

### Inspection

The final inspection covers the following areas Rodgers [2020]:

- Visual check by a specialized inspector for several defects such as blisters, exposed cords, and incomplete fill
- Removal of flash if present
- Uniformity (measurements)
- Shearography and other special inspections required by the customer for specific tires

- Laboratory analysis for a) cut tire analysis and individual component measurements b) durability tests c) specific statistical quality control (SQC)
- Some specialized tires such as large commercial truck tires as well as some light truck tires are inspected through X-ray which can penetrate the rubber and analyze the steel cord structure, placement, and uniformity

# Chapter 4

# Project in Tire Industry

## 4.1 Overview

The tire industry is a critical component of the global economy as it provides essential input to a broad range of industries such as transportation, construction, and agriculture.

By market size, it is estimated to be worth around $230 billion as of 2022. The market is expected to grow at a compound annual growth rate (CAGR) of 6.1% from 2022 to 2028 Report [2012]. In recent history, the COVID-19 pandemic had a significant effect on the automotive tires market directly in the short term as the production and sales of new vehicles witnessed a decline in 2020. Moreover, owing to restrictions, vehicle owners reduced the amount of driving resulting in delayed visits for maintenance or tire replacement. However, with the projected exponential increase in vehicle sales over the forecast period, the market is expected to be revived economically.

The main drivers of demand are population growth, urbanization, increasing disposable income for new original equipment manufacturers (OEM) tires (while the opposite is true for aftermarket supply), and the growth of the automotive industry. The demand for tires is also influenced by government regulations related to fuel efficiency and environmental concerns as stated Report [2012].

On the other hand, the tire industry is highly dependent on raw materials such as natural rubber, synthetic rubber, carbon black, and other chemicals. The industry is also heavily influenced by the price of crude oil as it is a major component of synthetic rubber Report [2012].

All in all, the tire industry is expected to grow in the coming years due to factors such as the increasing demand for vehicles in emerging economies, technological advancements in tire manufacturing, and the growing popularity of electric vehicles. However, the industry faces significant challenges that need to be addressed to sustain growth and profitability. For instance, mature industries such as the tire and mobility markets face a great deal of competitiveness and everything boils down to economies of scale and efficiency to achieve the profitability margins required by investors.

## 4.2 Competitive Analysis

The tire industry is highly concentrated, with a few large companies dominating the market. The major players in the tire industry are Bridgestone, Michelin, Goodyear, Continental, and Pirelli. It is highly competitive, with companies competing on factors such as price, quality, innovation, and brand recognition Report [2012]. The industry also faces significant challenges such as rising raw material costs, changing consumer preferences, and increasing government regulations.

Tire manufacturing is a capital-intensive process that requires significant investments in equipment, facilities, and human resources. The costs of production can vary depending on factors such as location, labor costs, energy costs, and regulations

The tire industry can be analyzed using Porter's Five Forces framework, which helps to understand the competitive dynamics of the industry. The five forces are:

- **Threat of new entrants:** The tire industry is a capital-intensive industry that requires significant investments in manufacturing facilities, research and development, and marketing Research [2010]. As a result, the threat of new entrants is relatively low. Existing companies have established brands and strong distribution networks, which make it difficult for new entrants to compete. The tire industry is highly globalized, with many tire manufacturers operating in multiple countries and exporting products to other regions. International trade policies, tariffs, and regulations can have a significant impact on the industry. For example, the U.S.-China trade war led to increased tariffs on imported tires, which impacted the profitability of tire manufacturers. Advancements in technology have improved the quality, safety, and durability of tires. For example, the development of new tire compounds and tread designs can enhance performance and fuel efficiency. However, investing in new technologies can also increase production costs and impact profitability.

- **Bargaining power of suppliers:** The tire industry is heavily reliant on raw materials such as rubber, steel, and chemicals Report [2012]. The bargaining power of suppliers can be high, especially if there are limited suppliers of these raw materials. However, tire manufacturers can reduce their dependence on specific suppliers by diversifying their sourcing strategies

- **Bargaining power of buyers:** The tire industry is highly competitive, and buyers have a significant amount of bargaining power. Consumers are price-sensitive and have access to a wide range of tire brands, which means that tire manufacturers must keep prices competitive and offer high-quality products to remain competitive.

- **Threat of substitutes:** There are several substitutes for tires, including public transportation, bicycles, and walking. However, these alternatives are not practical for most consumers, which means that the threat of substitutes is relatively low Research [2010].

- **Intensity of competitive rivalry:** The tire industry is highly competitive, with several established brands competing for market share. The industry is characterized by frequent price wars, aggressive marketing, and product innovation. Companies must constantly improve their products and processes to remain competitive.

## 4.3 Waterfall Methodology

The waterfall method, also known as the waterfall model, is a traditional project management approach used in software development and other fields. It is a linear and sequential model that consists of distinct phases, with each phase being completed before moving on to the next one. The model follows a top-down approach, meaning that the process flows steadily downward, just like a waterfall, and each phase must be completed in order Adenowo and Adenowo [2013].

### 4.3.1 Requirements

This is the initial planning phase in which the team gathers as much information as needed to ensure the success of the project. This part is very crucial because the waterfall method is linear, so it is very important to detail and plan ahead with a lot of forethought to avoid problems downstream Adenowo and Adenowo [2013].

For this purpose, it is necessary to gather information about specific roles and responsibilities, team members and stakeholders, resources required, budget, timeline, and deadlines, identify the customer, project scope delimitation, goals, metrics, and deliverables, etc.

Many of these requirements were already discussed and aligned internally before the start of this internship such as the project deadlines, purpose, and goals. Some other information with flexible constraints was adapted during the project.

### 4.3.2 Project Overview

| | |
|---|---|
| **Problem** | Currently the company has no tool to control construction machines cycle times with the precision and accuracy desired. This leads to a lot of hours of work employed in traditional sampling methods via footage analysis and inaccurate results.<br>There is software developed and installed in each machine to time the cycle time elements via programmable logic circuits actuators.<br>This software needs also to be validated to ensure the data collected is representative of the current state of production. |
| **Purpose** | The implementation of such measures will help immensely the industrial engineering department by reducing hours employed in sampling and by producing data with much higher accuracy and trustworthiness than ever before. |
| **Business Case** | As the tire industry is a mature and consolidated industry with a high barrier to entry due to the high capital intensity, know-how and low product differentiation, the company strategy is to be as lean as possible and ensure the highest productivity output possible.<br>The monitorization of the cycle times and PLC controls for each machine will allow to optimize further each machine by choosing the optimal PLC programming within each machine and PLC class.<br>For example, given a class of machines, we can learn which is the fastest performer by assessing cycle times and then perform an in-depth analysis to learn the latent variables (understand if it is due to the PLC programming, mechanical pieces or other shenanigans) and use it as a reference for the remaining machines in the class. |
| **Goals** | Create a software application to monitor PLC settings and risk assessment.<br>Create software to make a statistical analysis of the cycle times. |
| **Deliverables** | Software application with PLC monitoring capacity.<br>SQL datamart for the PLC data and cycle time data with support in Excel, power bi.<br>Software application for statistical analysis of cycle times. |

## Project Scope

| | |
|---|---|
| **Project Scope** | An integral part of the project is to contact the key stakeholders for data integration (IT partners) and engineering to assist in whichever tasks are required. This includes all the PLC validation, data accesses, and servers. |
| **Outside Project Scope** | This project doesn't affect other ongoing analytical solutions, it doesn't make any changes to the data warehouses (SQL and RedShift) and it's not responsible for the maintenance or data quality in the source. This project isn't responsible for the improvement of PLC circuits, it is a tool to learn improvement opportunities. All ad hoc issues are requested to the responsible partners and they should take over the changes accordingly. |

## Gantt Diagram

The proposed duration for the internship with the aligned tasks initially predicted a kick-off in the first week of March till the end of September. The Gantt diagram is defined in the figure 10.

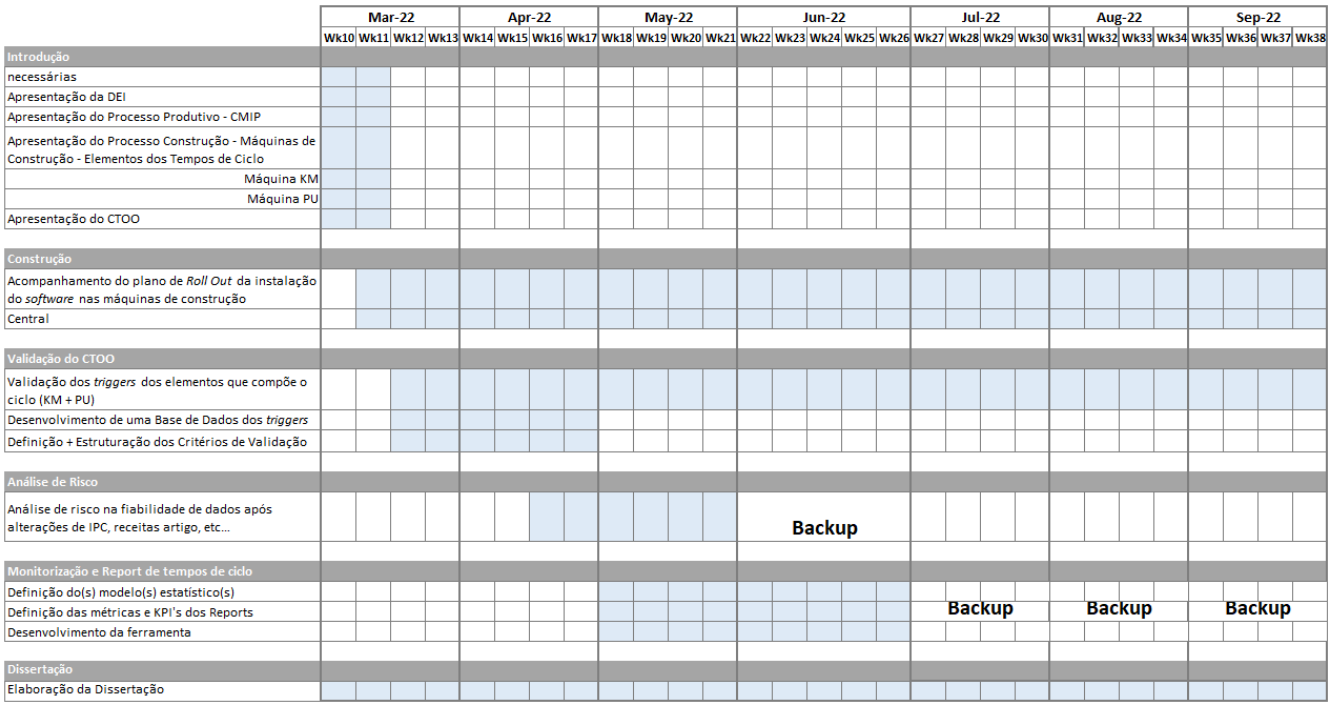


Figure 10: Internship Gantt diagram.

## Objectives

The main purpose of this project was the development of a data-driven decision support system infrastructure with two essential objectives: monitoring and control of PLC triggers and the statistical inference of cycle times (CT).

| **PLC tracking** | **CT Statistical Inference** |
|---|---|
| Define customer requirements | Define customer requirements |
| Obtain and select the required data | Obtain and select the required data |
| Development of a support database | Development of a support database |
| Selection of suitable PLC metrics | Development of a statistical inference framework |
| Deploy metrics into data-visualization techniques | Deploy metrics into data-visualization techniques |
| Develop a user interface | Develop a user interface |

## Wireframe

The proposed layout wireframe for the user interface, widgets, and graphs follows the following mock-up in figure 11.

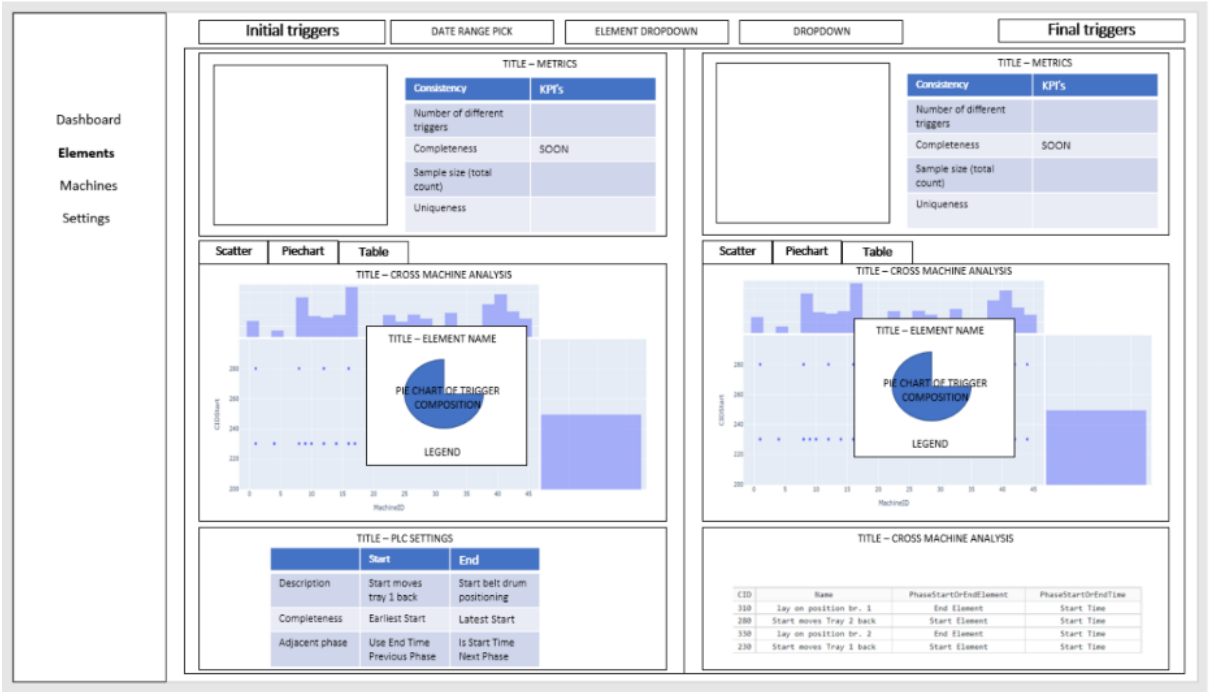


Figure 11: Initial wireframe proposal.

### 4.3.3 System architecture

#### Monolith architecture overview

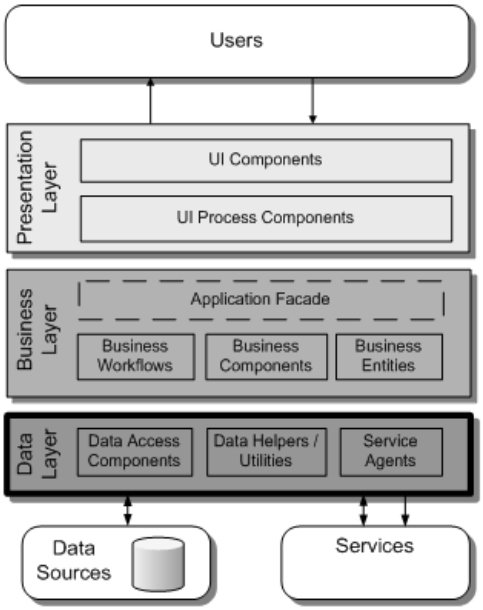


Figure 12: Data layer overview Paolozzi et al. [2012].

The data layer provides a brief overview of all the interacting components, from the back end and data sources, all the way to the business logic and front end layer where it interacts with the users. In this application, the data flow is mainly one-way since it does not collect user data besides user inputs.
For this purpose, the application is divided into 3 main layers according to the figure 12:

- **Data layer:** it provides the functionalities to interact with the databases. It has a data helper function with miscellaneous methods and decorators to provide a different range of functionalities, create facades, and extend behaviors to existing functions. Some of those purposes are to provide interfaces between the database modules and data frames and facades grouping several interface functions.

- **Business layer:** It provides the necessary tools to implement business domain logic and statistical analysis. It includes data manipulation operations such as grouping and sorting data as well as calculating the key metrics we ought to analyze. It prescribes methods for how business objects interact and enforce the routes by which business objects are accessed and updated. It includes the business rules which are operations, constraints, and definitions which rule data flows. The

application facade is the gateway that connects to the user interface. This contains the helper functions and facade class which encapsulates methods that transform the output business data into the input of data visualization functions.

- **Presentation layer:** The presentation layer includes all the functions and classes directly responsible for client interaction, interface creation, and user interactivity. It is mainly done with plotly and dash packages in python which in turn binds to a javascript layer to create the HTML/CSS/-Javascript frontend.

### 4.3.4 Technical Requirements

Libraries and relevant packages

| | |
|---|---|
| **Apache Airflow** | Apache Airflow is an open-source tool to author, schedule and monitor workflows, and it is used for scheduling and orchestration of data pipelines or workflows. These data pipelines deliver data sets that are ready for consumption either by business applications, data analysis, or big data applications. |
| **Boto3** | This is the Amazon Web Services Software Developer Kit for Python to create, configure and manage AWS services. In this case, this is the connector used for the Amazon RedShift database. |
| **Dash** | Dash is an open source framework for building data visualization interfaces built on top of flask, react and plotly. |
| **Dash Bootstrap Components** | Dash Bootstrap components is an extension of Dash library maintained by the same developers providing new widgets and binds to the powerful front end framework Bootstrap. It uses a pre-built grid system and custom themes and building blocks. |
| **Dash Table** | Dash table is a module built on top of Dash providing the functionalities of custom tables in front-end. |
| **Dash HTML Components** | Dash HTML components is an extension of Dash providing new HTML base components. |
| **Dash Tabulator** | Dash tabulator is a binding for tabulator javascript framework to build, dynamic tables and rich feature tables. |
| **Distfit** | Distfit is the module used to fit distributions and make parametric estimations using namely the Maximum Likelihood Estimator to datasets. |
| **Flask** | Flask is a Web Server Gateway Interface providing a micro-framework to connect the web applications to the server side communications. It provides effective and simple means to scale and release complex web applications. |
| **Graphviz** | Graphviz is an open-source graph visualization software used to represent structural information as graphs and networks. It is the core for the visualization of Directed Acyclic Graphs and other data/workflow graphs. |
| **Keras preprocessing** | This is the data pre-processing and data augmentation module of the Keras library. It provides several helping functions to digest data inputs. |
| **Mysql** | MySQL is the standard ODBC SQL connector |
| **Numpy** | Numpy is a mathematical and computational framework of Python and it is the core of other scientific computing environments such as machine learning and data analysis applications. |
| **Openpyxl** | Openpyxl provides an interface to the excel handlers and allows the export and formatting into xls, xlsx, xlsm formats. It also provides every functionality of excel through python binding functions. |
| **Pandas** | Pandas is a fast, powerful, flexible, and easy-to-use open-source data analysis and manipulation tool. |

Libraries and relevant packages

| | |
|---|---|
| **Pandas Profiling** | Its primary goal is to provide a one-line Exploratory Data Analysis (EDA) experience in a consistent and fast solution. |
| **Pyodbc** | Pyodbc is an open source Python module that makes accessing Open Database Connectivity ODBC databases simple |
| **PySpark** | PySpark is an interface for Apache Spark in Python. It not only allows to write Spark applications using Python APIs, but also provides the PySpark shell for interactively analyzing data in a distributed environment |
| **Requests** | Requests library for http connections. |
| **Sklearn** | Sklearn is a machine learning library with several clustering algorithms, classification, regression and support vector machines. |
| **Scipy** | SciPy is a collection of mathematical algorithms and convenience functions built on NumPy . It adds significant power to Python by providing the user with high-level commands and classes for manipulating and visualizing data. |
| **SQLAlchemy** | It provides a full suite of well-known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language. |
| **Statsmodel** | Statsmodels is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration. An extensive list of result statistics is available for each estimator. |
| **Urlib3** | urllib3 is a powerful, user-friendly HTTP client for Python. Much of the Python ecosystem already uses urllib3 |

## 4.4 Tech Stack

### 4.4.1 Amazon Redshift

Amazon Redshift is a data warehouse service in the Amazon Webservice (AWS) Cloud provided by Amazon. Amazon Redshift is a relational database management system (RDBMS) and provides the same functionalities as traditional RDBMS systems:

- Flexibility - it is easy to add, update, or delete tables, records, relationships, and other changes to the data without impacting existing applications
- ACID compliance - Atomicity, Consistency, Isolation, Durability performance. Ensures data consistency and validity regardless of errors
- Ease of use - easy to run complex SQL queries
- Collaboration - multiple people can operate and access data simultaneously
- Built-in security - role-based security ensures data access is limited to specific users

- Database normalization - relational databases employ a normalization technique reducing redundancy and improving data integrity

### Amazon Redshift Architecture

It's a collection of computing resources called nodes, which are organized into clusters. Each cluster runs an engine and contains one or several databases.

A cluster is a core infrastructure component and is provisioned with two or more compute nodes and an additional leader node that coordinates the compute nodes. The leader node handles external communication with applications, such as query editors and business intelligence (BI) tools Redshift [2022]. For that reason, the client application interacts directly only with the leader node, hence its importance in the process.

Each cluster contains one or more databases. User data is stored on the compute nodes. The SQL client communicates with the leader node which in turn runs queries with the underlying compute nodes. Each database is organized into one or more schemas Redshift [2022]. The figure 13 illustrates the RedShift architecture.

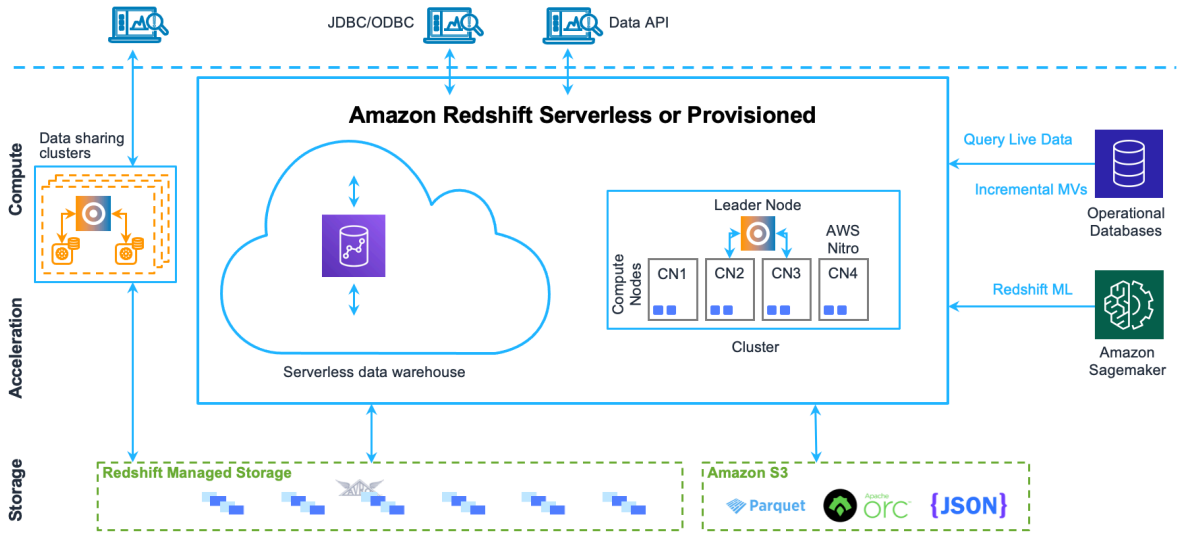


Figure 13: High level Amazon Redshift system architecture Redshift [2022].

## 4.4.2 SQLAlchemy

SQLAlchemy SQL Toolkit is a comprehensive set of tools for working with relational databases using object-oriented programming languages such as Python. Its major components are the SQLAlchemy Object Relational Mapper (ORM) and SQLAlchemy Core as depicted in figure 14.

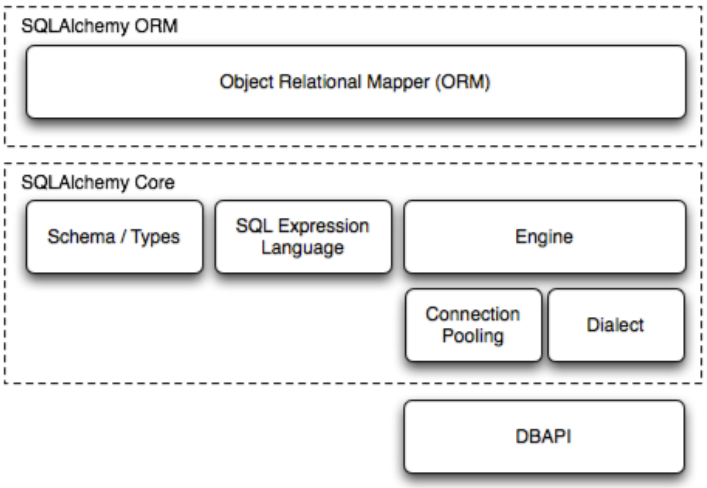


Figure 14: High level SQLAlchemy system architecture Sqlalchemy [2023].

Core contains the breadth of database integration with the most prominent part of this being the SQL Expression Language.

The SQL Expression Language is a toolkit on its own independent of the ORM package which provides a mean of constructing SQL expressions represented by composable objects which are then executed against the target database resulting in a set of transactions Sqlalchemy [2023]. Each and every interaction is achieved by passing SQL expression objects representing these statements.

The ORM is built upon Core to provide a means of working with a domain object model mapped to a database schema. ORM is a programming technique for converting data between a relational database and the heap of object-oriented programming languages Sqlalchemy [2023]. For this effect, an object-oriented implementation is created based on an underlying schema. For example, given an address book entry representing a single person along with contact, address, and name fields, the ORM equivalent would be through a Person object with each attribute holding a data item: `Person.address`, `Person.name`, etc.

### 4.4.3 Python and Jupyter Notebooks

Python is a high-level, imperative, object oriented, and functional programming language. One of its main features is to allow easy reading of the code and less verbosity when compared with other programming languages.

### 4.4.4 Jupyter Notebooks

A Jupyter Notebook (figure 15) is a rich Internet web computing environment that supports both executable code and rich text elements for note-taking. Notebook documents are human-readable documents containing analysis descriptions, figures, tables, mathematical expressions as well as executable snippets of code.

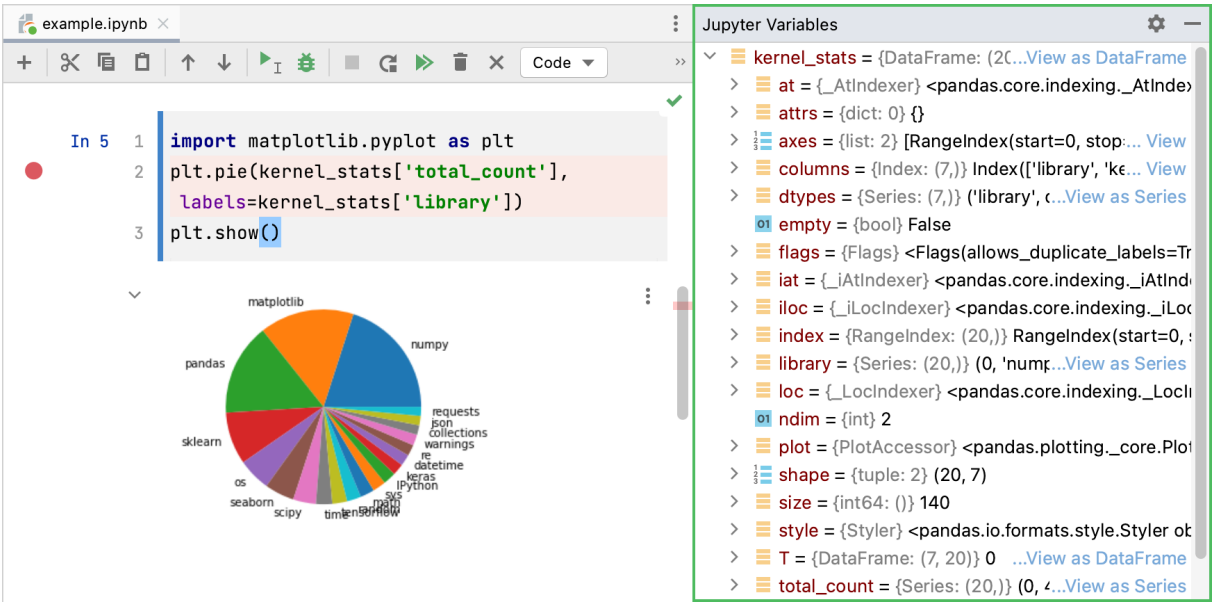


Figure 15: Jupyter Notebook.

The Jupyter Notebook App runs on a server-client application. It can be executed locally requiring no internet access. The main components are the notebook kernel or computational engine that executes the code contained in a Notebook and the notebook dashboard which is first shown when launching the application and is used to open the documents and manage kernels.

### 4.4.5 Apache Airflow

Apache Airflow is an open-source workflow management platform for data engineering pipelines. It uses directed DAGs to manage workflow orchestration. Tasks and dependencies are defined with Python and the Airflow manages the scheduling and execution. DAGs can be run procedurally on a defined schedule (hourly or daily) or based on external event triggers.

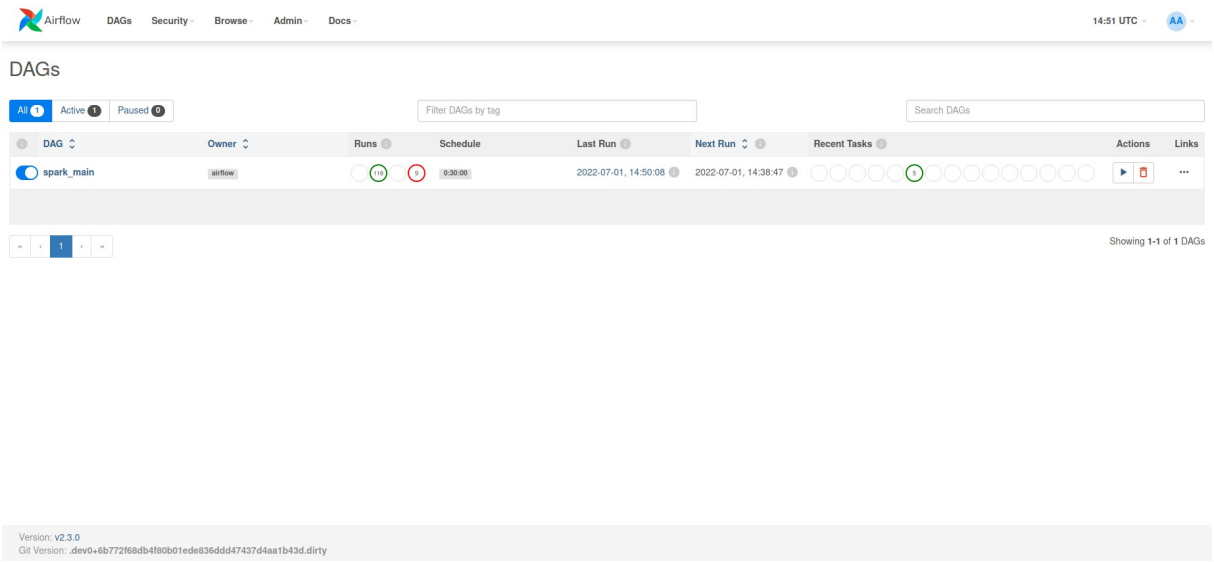


Figure 16: Apache Airflow interface

It has a very rich set of operators bundled with a very clean web application to explore DAG's definition together with their dependencies, progress, metadata and logs (figure 16). The base modules are very easy to extend so they promote the community development of extensions.

# Chapter 5

# Data Pipeline Orchestration

Data orchestration refers to the process of managing and coordinating various data-related tasks and workflows within an organization. It involves the arrangement, integration, and automation of data movement, processing, and analysis across different systems, applications, and platforms.

In today's data-driven world, organizations deal with vast amounts of data from various sources, such as databases, data warehouses, cloud services, application programming interfaces (APIs), and external data providers. Data orchestration aims to streamline and optimize these complex data processes to ensure data availability, reliability, and consistency throughout the organization Matskin et al. [2021].

Data orchestration is composed primarily by the following steps, by no particular order:

- Data Integration: Involves integrating data from disparate sources, including structures and unstructured data into a unified view. This can be achieved through data extraction, transformation, and loading (ETL) processes, where data is extracted from different sources, transformed into a consistent format and loaded into a target system.

- Data Transformation: Data often needs to be transformed or enriched to make it suitable for analysis or to meet specific business requirements. Data orchestration includes tasks such as data cleansing, normalization, aggregation, where data is processed and modified as needed.

- Data Movement and Synchronization: Data orchestration coordinates the movement of data across various systems and platforms, ensuring that data is delivered to the right place at the right time. This involves tasks such as data replication, data synchronization, and data migration, which enable data consistency and availability across different environments.

- Workflow Automation: Data orchestration involves automating the execution of data-related tasks and workflows to reduce manual effort, improve efficiency, and minimize errors. This can be achieved through workflow management systems or data orchestration tools, which enable the design, scheduling, and monitoring of data processes.

- Data Governance: Data orchestration includes enforcing data governance policies and security measures to ensure data privacy, compliance, and protection. It involves managing data access controls, data quality, data lineage, and auditing to maintain data integrity and trustworthiness.

- Data Analytics and Insights: Data orchestration plays a crucial role in facilitating data analysis and deriving meaningful insights. By orchestrating data processes, organizations can ensure that the right data is available to analysts, data scientists, and decision-makers, enabling them to derive actionable insights and make informed business decisions.

A data pipeline's architecture is made up of 4 main parts: a data source, business logic, a destination, and a scheduler (for batch).

- Data sources: Common data sources are application databases, APIs, or files from an SFTP server. In this scenario, the data sources will be the Enterprise Resource Planning backend, file sheets and machine internal database for staging.

- Business logic: Business logic is a general term that encompasses the type of transformations that need to be applied to the data inside the data pipeline. It usually involves cleaning, filtering, and applying logic to the data that is specific to the business. It will be tackled in two steps: data cleansing and data transformation.

- Data destination: Typically, the target we send the data is another database. Common data targets are databases or data storage areas that are made for analytics.

- Orchestration tool: The orchestration tool is responsible for organizing, controlling and managing workflows and data batches ensuring automated data pipelines. For this purpose, Luigi and Airflow were considered, eventually considering Apache Airflow.

## 5.1 Data Integration

### Data profiling

Data profiling is the process of analyzing and examining data from various sources to gain a better understanding of its structure, quality, and content. It involves assessing the completeness, consistency, accuracy, and integrity of the data. Data profiling aims to identify and understand data patterns, relationships, and anomalies, and to provide insights into the overall health and usability of the data.

The primary goal of data profiling is to gather metadata about the data, including statistical summaries, data types, field lengths, and value distributions. This information helps in assessing the overall quality of the data and identifying potential data quality issues. By understanding the characteristics and properties of the data, organizations can make informed decisions about data integration, migration, cleansing, and transformation processes.

Data profiling techniques typically involve examining the data at both the structural and content levels. Structural profiling focuses on the data's schema, such as column names, data types, and relationships between tables. Content profiling involves analyzing the actual values within the data, such as identifying missing values, outliers, or inconsistent formats.

Data profiling can be performed using various tools and techniques, ranging from manual analysis to automated processes. Some common techniques include statistical analysis, pattern recognition, data visualization, and data quality rules and checks. The results of data profiling provide insights into the data's strengths and weaknesses, allowing organizations to address any data quality issues and make informed decisions about data management and usage.

#### Pandas profiling

Pandas profiling is an extension to pandas data-frames which provides a one-point stop to all the requirements of data-frame profiling.

- **Profiling Overview:**

  Provides an API to access all the descriptive metadata of the data-frame, from variable type, columns, row count, duplicates, and memory usage (figures 20, 21, 22).

- **Column Analysis:**

  This involves analyzing individual columns within a data set. For example, you can profile a column

containing customer names to determine the most common names, identify any missing or inconsistent values, or identify outliers. This analysis helps in understanding the data distribution and detecting potential data quality issues (figures 17, 19).

- **Value Frequency Analysis:**

  This analysis examines the frequency of different values within a specific column. It helps in identifying duplicate or redundant values, determining cardinality (number of distinct values), and detecting potential data anomalies or errors. For example, you can identify if a product category column contains misspelled or inconsistent values (figure 18).

- **Data Type Analysis:**

  This involves analyzing the data types of columns within a data-set. It helps in ensuring that the data is stored in the appropriate format. For example, you can identify if a column that should contain dates has string values or if numeric columns contain non-numeric characters.

- **Completeness Analysis:**

  This analysis assesses the completeness of data within a data-set. It helps in identifying missing or null values within columns and understanding the extent of missing data. This insight is crucial for data quality assessment and decision-making processes (figure 17).

- **Pattern Analysis:**

  This involves identifying patterns and relationships within the data (figure 23). For example, you can analyze the correlation between different columns to determine if there are any dependencies or if one column can be derived from another. Pattern analysis helps in understanding the data structure and identifying any inconsistencies or anomalies.

- **Statistical Analysis:**

  This analysis involves calculating statistical measures such as mean, median, standard deviation, and outliers for numerical columns (figure 19). It helps in understanding the distribution of data, identifying extreme values, and detecting potential data quality issues.

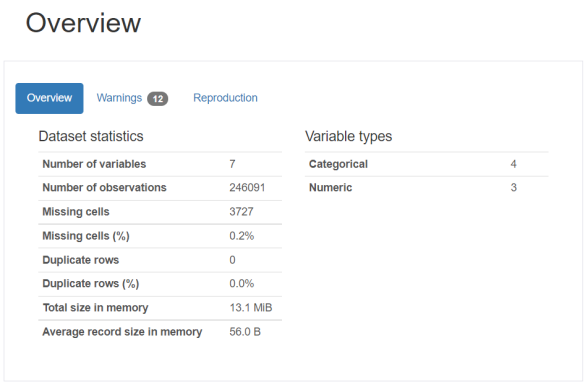


Figure 17: Completeness, data type analysis overview

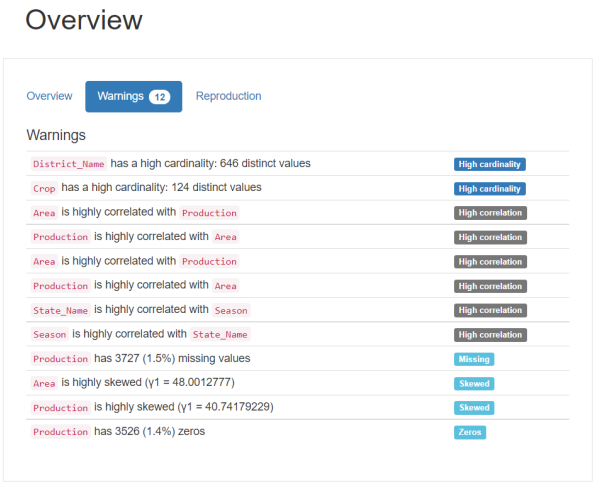


Figure 18: Pattern and value frequency overview

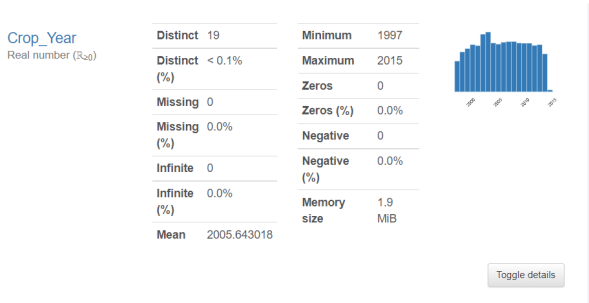


Figure 19: Numeric column-wise analysis

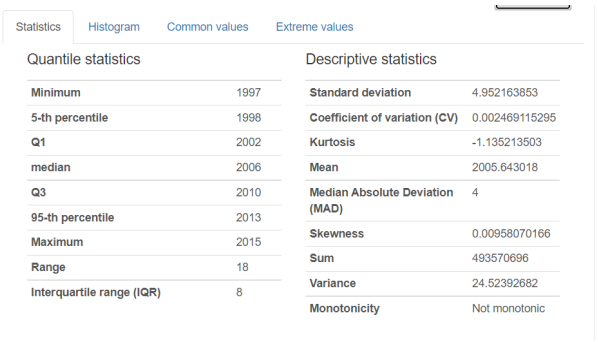


Figure 20: Numeric column-wise descriptive analysis

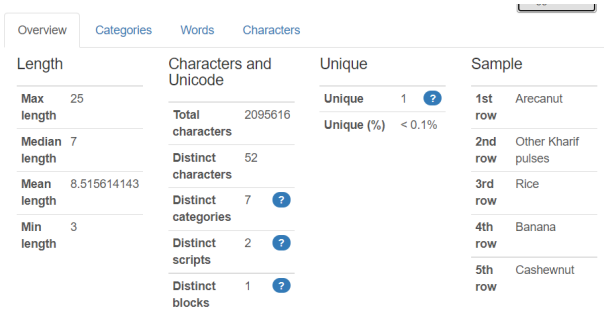


Figure 21: String column-wise descriptive analysis

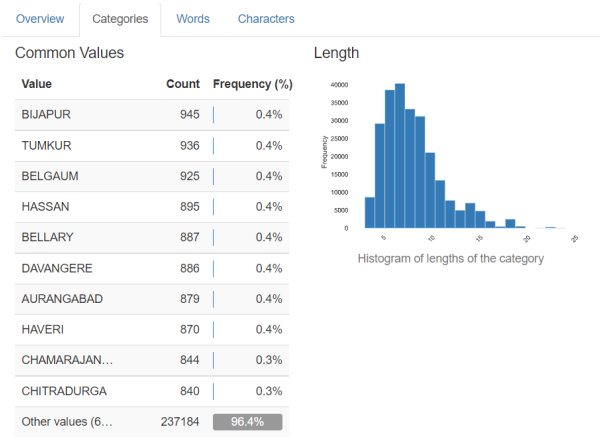


Figure 22: String column-wise descriptive analysis

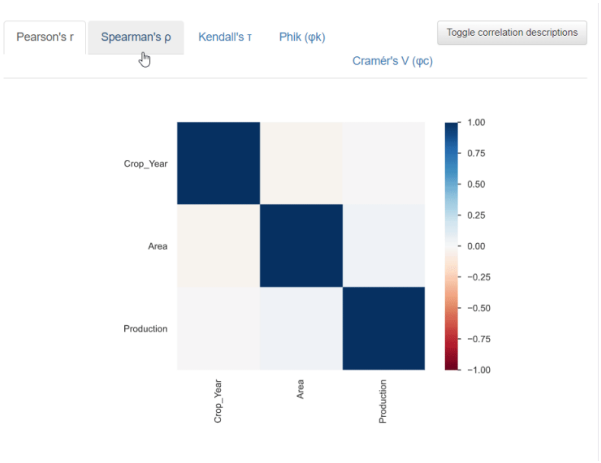


Figure 23: Variable correlation analysis

## 5.2 Database Modeling

Data modeling is the process of defining and analyzing data requirements to support business processes and problems within the scope of computer science and information systems. This involves different stakeholders such as the business stakeholders as potential users of the information system and the data modelers.

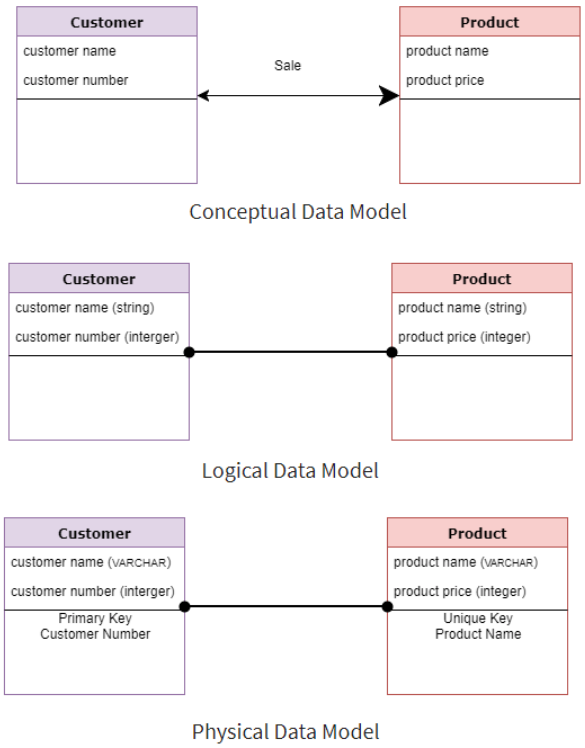


Figure 24: Database modeling

The workflow for data modeling involves the concepts in figure 24, which can be explained below:

- **Conceptual data model** is defined as an abstract model that organizes data description, data semantics, and consistency constraints of data. The data model emphasizes on what data is needed and how it should be organized instead of what operations will be performed on data. The two most common data modeling techniques used are entity-relationship (ER) diagrams for the data objects and unified modeling language (UML) to model classes and methods accross the application.
- **Logical data model** which structures multiple logical models that can be implemented in databases.
- **Physical data model** which organizes data into tables, accounts for access, performance, storage details, data elements, structures, and relationships between them.

### 5.2.1 STAR Schema

A star schema is a type of data modeling technique used in data warehousing to represent data in a structured and intuitive way. It is the fundamental schema among data mart schemes and it is also the simplest. The data is disposed into a central fact table that contains the measures of interest, surrounded

by dimension tables that describe the attributes of the measures. The start schema is necessary and the predecessor of the snowflake schema.

It is said to be a star as its physical model resembles the star shape having a fact table at its center and the dimension tables at its periphery representing the star's points. The fact table contains the measures or metrics that are of interest to the user or organization. For example, in a sales data warehouse, the fact table might contain sales revenue, units sold, and profit margins. Each record in the fact table represents a specific event or transaction, such as a sale or order. The dimension tables in a star schema contain the descriptive attributes of the measures in the fact table. Facts change regularly, and dimensions do not change, or change very slowly. In a star schema, each dimension table is joined to the fact table through a foreign key relationship. This allows users to query the data in the fact table using attributes from the dimension tables.

The star schema is a popular data modeling technique in data warehousing because it is easy to understand and query. The simple structure of the star schema allows for fast query response times and efficient use of database resources. Additionally, it can be easily extended by adding new dimension tables or measures to the fact table, making it a fast, simple scalable and flexible solution for data warehousing.

Some advantages of STAR Schema Han et al. [2012]:

- **Simpler queries** Join logic of star schema is needed to fetch data from a transactional schema that is highly normalized.
- **Simplified business report logic** In comparison to a transactional schema that is highly normalized, the star schema makes simpler common business reporting logic
- **Query performance** Because a star schema database has a small number of tables and clear join paths, queries run faster than they do against an Online Transaction Processing system. Small single-table queries, usually of dimension tables, are almost instantaneous.
- **Easy to understand and navigate** Star schema are easy for end users and applications to understand and navigate.

Some disadvantages of STAR Schema Han et al. [2012]:

- **De-normalized schema** Data integrity is not enforced well since in a highly de-normalized schema state. Data doesn't follow ACID ruleset (Atomicity, Consistency, Isolation, Durability).
- **Not suitable for very detailed data** Star schema doesn't reinforce many-to-many relationships within business entities – at least not frequently.

- **Doesn't scale very well with the size of information**

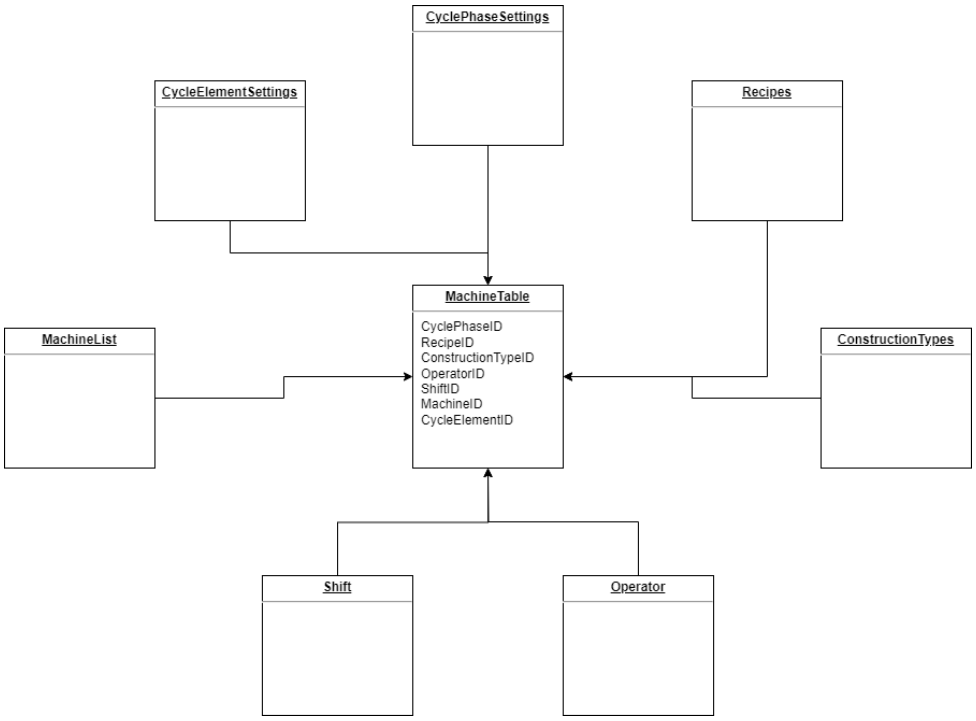


Figure 25: Early version of the proposed physical model using STAR Schema

Based on the figure 25 as a reference, it's possible to conclude the MachineTable is a fact table and all the surrounding tables are dimension tables containing the necessary measures.

### 5.2.2 Snowflake Schema

As previously stated, the Snowflake schema is an extension of the STAR schema where dimensions are further broken down into sub-dimensions. Snowflake schema is very popular in business intelligence, data warehouses, data marts, and overall SQL relational databases.

Because individual dimensions are broken down into logical sub-dimensions, this adds additional complexity to the model but can make it easier for business analysts to perform their assessments. The central fact table connects to multiple dimension tables via foreign keys like in STAR schematics.

Additionally, snowflake schema is more storage space efficient since they are more normalized than their predecessor. De-normalized data models have more redundancy and thus more duplicated or unnecessary data fields which obviously carry additional space and performance cost.

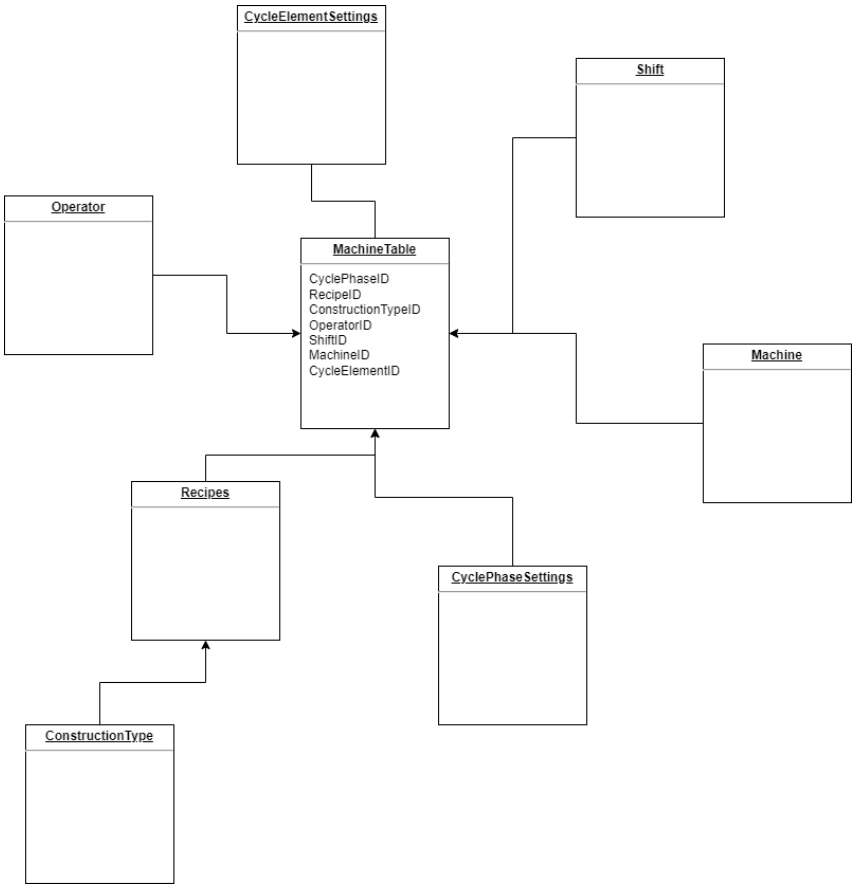


Figure 26: Proposed Snowflake schema.

In the figure 26, the snowflake schema for the current problem is proposed.

### 5.2.3 Snowflake Procedures

A procedure (or sproc) is a subroutine available to all relational database management systems (RDBMS) and thus not exclusive to snowflake databases. Such procedures are stored within the database and have certain properties that make them attractive to developers. An example of a procedure is depicted in figure 27.

For instance, stored procedures can save time, memory and processor power. Several SQL statements can be saved within a procedure and all applications which access the database can call these procedures. Finally, these may contain stored variables, flexible user-defined data types, and control flow units (such as IF, WHILE, LOOP, CASE statements).

```
// Example of a procedure
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30)
AS
SELECT * FROM Customers WHERE City = @City
GO;

EXEC SelectAllCustomers @City = 'London';
```

Figure 27: Example of a procedure

### 5.2.4 Virtual Tables

Virtual Tables (or views) are a special kind of SQL tables (figure 28. They provide a virtual environment that can perform various complex operations in memory. The definitions of the view are stored in the database in a dictionary like the procedures.

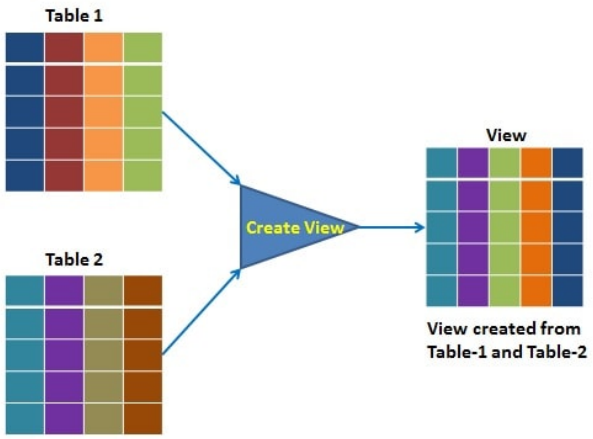


Figure 28: Virtual table resulting from the logical join between two tables

Virtual tables can be classified into:

- **Simple View** based on a single table, without any GROUP By clause or function
- **Complex View** based on multiple tables or containing GROUP BY clause or function
- **Inline View** based on a sub-query which creates a temporary table and simplifies a complex query
- **Materialized View** stores the definition as well as the data. Stores the data in physical memory.

Some key differences between virtual tables and procedures:

- Stored procedures are more flexible and accept parameters while views are static
- Views can be used to simplify complex queries (Inline Views) while procedures cant be used in queries
- Views are memory representations and therefore can't physically change any memory address while procedures can modify tables
- Procedures can contain several statements while views are limited to only one SELECT
- Materialized Views can be the target of INSERT or modification clauses whereas procedures can't

### Materialized View

Materialized view is a physical presentation of the retrieved data. This replica can be reused without executing the view again. They are snapshots of a query at a specific time hence they are very helpful to improve general dataflow and data governance. The scheme for the CID is in figure 29.

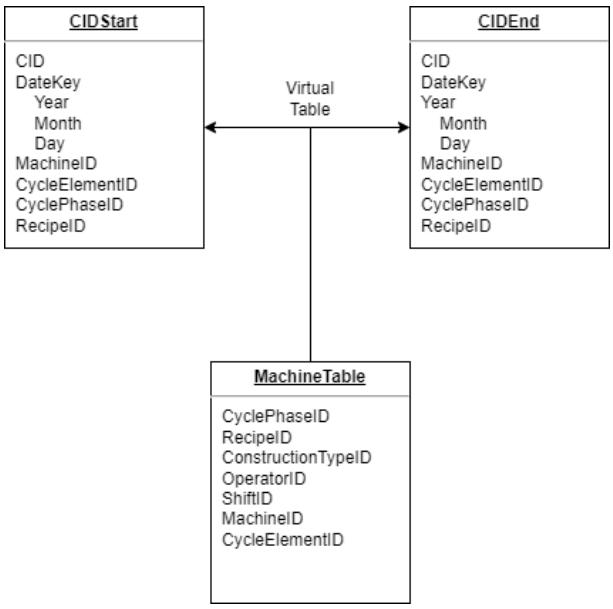


Figure 29: Materialized views CID count

A useful application of this concept is the implementation of two CID count views. The main purpose of these tables is to count and track the PLC actuator ID per foreign key of the fact table.

```
for __machine in tqdm(machines):
    query = """
            SELECT RecipeID, CyclePhaseID, MachineID, CIDStart, COUNT(CIDStart)
                as CIDStartCount, datepart(day, CreatedOn) as Day,
                datepart(month, CreatedOn) as Month, datepart(year, CreatedOn)
                as Year
            FROM {}
            GROUP BY MachineID, CyclePhaseID, CIDStart, RecipeID,
                datepart(year, CreatedOn), datepart(month, CreatedOn),
                datepart(day, CreatedOn)""".format(__machine)
    _data = pd.read_sql_query(query, self.connection)
    # ISO FORMAT
    _data["Date"] = _data["Year"].astype(str) + '-' +
        _data["Month"].astype(str) + '-' + _data["Day"].astype(str)
    _data = _data.drop(["Day", "Month", "Year"], axis=1)
    d = d.append(_data)
```

Figure 30: The figure illustrates multiple SQL queries that later turned into a single CIDStart materialized view

## 5.3 Data Transformation

### 5.3.1 Data Cleansing & Data Validation

Data cleansing, also known as data cleaning or data scrubbing, is the process of identifying and correcting or removing errors, inconsistencies, inaccuracies, and anomalies from a data-set. The goal of data cleansing is to improve data quality, ensuring that the data is accurate, reliable, and suitable for analysis or other data processing tasks.

#### Type checking

Type checking is a process performed by compilers, interpreters, and other programming tools to verify the correctness of the types used in a program. It involves analyzing the code to ensure that variables, expressions, and function calls are used consistently with their declared types.
The purpose of type checking is to catch programming errors and ensure that operations are performed on compatible data types. It helps prevent type-related bugs, such as attempting to perform arithmetic operations on incompatible types (e.g., adding a number to a string) or passing arguments of the wrong type to a function.
Type checking can be performed statically or dynamically:

- Static type checking: Static type checking is done at compile-time and involves analyzing the source code without executing it. The compiler examines the types declared for variables, functions, and expressions and checks if they are used correctly throughout the program. Static type checking can catch many errors before the program is run, which can help improve reliability and performance.

- Dynamic type checking: Dynamic type checking is performed at runtime while executing the program. The type of a variable or expression is checked when it is actually used during program execution. If a type mismatch or inconsistency is detected, an error or exception may be raised. Dynamic type checking allows for more flexibility but may result in errors occurring during program execution.

Type-checking can be enforced through type annotations in statically typed languages. These annotations explicitly specify the types of variables, function parameters, and return values. Statically typed languages, such as Java or C++, often perform extensive type-checking during compilation. In dynamically typed languages, type checking is typically looser or deferred until runtime.

One of the key aspects of using python dynamic typed variables is that it assigns by default the broadest variable type available at compilation time. When loading data into pandas, the library assesses for each column the data types and assigns integer (4 bytes), floating point, string, bool, and object. The object type is useful because it is the broadest data type available to Python which means there is virtually no chance of compiling errors or casting losses downstream due to data types, at the obvious cost of performance.

**Datetime ISO 8601 format** The datetime format used throughout the application was the International Organization for Standardization ISO 8601 following the general rules:

- Date and time values are ordered from largest to smallest unit of time: year, month, day, hour, minute, second
- Each date has a fixed number of zeros so it must be padded with leading zeros
- Representation can be formatted with a minimal number of separators (hyphen for dates and colon for time values). For example, the 6th day of the 1st month of the year 2009 may be written as "2009-01-06" in the extended format or simply as "20090106" in the basic format without ambiguity.

For this purpose, the datetime standard Python library was used.

### Type Casting

Type casting is often necessary when we want to perform operations that require operands of compatible types or when you need to store a value in a variable of a different type. For example, casting can be used implicitly while operating two variables together such as adding and integer to a floating point where the resulting variable will implicitly be of type float. This is utterly important in data driven programs such as this project where very strictly typed variables can result in data type errors downstream or overly relaxed weak typed variables can lead to memory issues.

There are two forms of type-casting:

- **Implicit type casting (coercion)** (figure 31): Implicit type casting occurs automatically by the compiler or interpreter when it is safe to convert one type to another without losing information or precision. For example, converting an integer to a float or widening the range of a numeric type. Implicit type casting is also known as type coercion.

- **Explicit type casting (conversion)** (figure 32): Explicit type casting involves manually converting a value from one type to another using casting operators or functions provided by the programming language. Explicit casting is typically required when there is a potential loss of information or precision during the conversion.

```
x = 5   # integer
y = 2.0 # float

result = x + y # The integer x is implicitly cast to float before the addition
print(result) # Output: 7.0
```

Figure 31: A simple example of implicit casting.

```
x = 5
y = 2.7

result = x + int(y) # Explicitly cast the float to an int using the int()
    function
```

Figure 32: A simple example of explicit casting.

### User-defined data types and strong typing

Custom data types, also known as user-defined data types, are data structures created by a programmer to represent specific entities or concepts within a program. They allow developers to define their own data structures that go beyond the built-in data types provided by a programming language.
Custom data types are typically created using classes, structures, or enums, depending on the programming language. They encapsulate related data and provide methods or functions to operate on that data. The figure 33 shows a custom data type as a Rectangle and its properties.

```
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def calculate_area(self):
        return self.width * self.height

    def calculate_perimeter(self):
        return 2 * (self.width + self.height)
```

Figure 33: Example of a custom data type using classes.

**Pandera** Pandera is an open-source Python library that provides a lightweight and intuitive way to validate and sanitize structured data, primarily focused on working with pandas data frames. It offers a declarative syntax for defining data validation rules and can be used in data pre-processing and data quality assurance workflows.

Pandera allows you to define schemas to specify the structure, types, and constraints of your data. These schemas act as blueprints for validating and cleaning your data. The library provides various built-in validators, such as checking for null values, ensuring data types, enforcing value ranges, and validating string patterns.

It allows for:

- **Schema definition**: Pandera allows you to define schema using a concise and expressive syntax. You can specify column names, data types, and various validation rules for each column in your data.
- **Data validation**: Once we have defined a schema, we can use it to validate the data. Pandera provides functions to validate pandas data frames against the specified schema, ensuring that the data conforms to the defined rules. It flags any discrepancies or errors encountered during the validation process.
- **Integration with pandas and dask**: Pandera seamlessly integrates with pandas, leveraging its powerful data manipulation capabilities. We can combine Pandera's schema validation with pandas' data handling operations, making it convenient for data pre-processing tasks.

```
import pandera as pa
from pandera import Column, Check, DataFrameSchema

schema = DataFrameSchema({
    "small_values": Column(float, Check.less_than(100)),
    "one_to_three": Column(int, Check.isin([1, 2, 3])),
    "phone_number": Column(str, Check.str_matches(r'^[a-z0-9-]+$')),
})

from pandera.typing import Series

class Schema(pa.DataFrameModel):

    column1: Series[int] = pa.Field(le=10)
    column2: Series[float] = pa.Field(lt=-1.2)
    column3: Series[str] = pa.Field(str_startswith="value_")

    @pa.check("column3")
    def column_3_check(cls, series: Series[str]) -> Series[bool]:
        """Check that column3 values have two elements after being split with
            '_'"""
        return series.str.split("_", expand=True).shape[1] == 2

Schema.validate(df)
```

Figure 34: Pandera checks using data frame schema.

With the operation in the figure 34, we can assert custom data-types such as "small_values" which is a type of float within the 100 range and we can also match strings using regex.

### Outlier detection

Outlier detection particularly in time series data refers to the identification of data points or patterns that deviate significantly from the expected behavior or the normal pattern of the time series. Outliers can indicate anomalous events, errors, or unusual behavior that may require further investigation or different

treatment in analysis or modeling.

Outlier detection uses several distinct methods to create a decision function or decision boundary which is then able to decide whether a new observation belongs to the same distribution as existing observations (inlier) or should belong to a different origin process (outlier).

**Percentile based method** In the percentile-based method (35), the values will be ranked and the values will be dropped or kept according to the percentiles considered. For this method, the threshold percentiles are user-defined and can be toggled to adjust the process sensitivity to outliers and are adjusted for skewed distributions.

```
if skew > 3:
    P05 = _data["Duration"].quantile(0.05)
    P60 = _data["Duration"].quantile(0.60)
    _data = _data[~((_data["Duration"] < P05) |(_data["Duration"] > P60))]

else:
    P05 = data["Duration"].quantile(0.05)
    P60 = data["Duration"].quantile(0.90)
    data = data[~((data["Duration"] < P05) |(data["Duration"] > P60))]
```

Figure 35: Percentile-based outlier detection.

**Support Vector Machine** One-Class Support Vector Machine (SVM) is an unsupervised model for anomaly or outlier detection. Unlike the regular supervised SVM, the one-class SVM does not have target labels for the model training process. Instead, it learns the boundary for the normal data points and identifies the data outside the border to be anomalies.

This problem can be thought of as a classification problem where by default 1 counts as normal data points or inliers and -1 as outliers. The algorithm will then drop the values classified as -1 and keep the inliers (figure 36).

```
# Predict the anomalies
prediction = one_class_svm.predict(X_test)# Change the anomalies' values to
    make it consistent with the true values
prediction = [1 if i==-1 else 0 for i in prediction]
```

Figure 36: Model prediction.

### Missing Data

Missing data can be categorized into three categories:

- Missing Completely at Random (MCAR)
- Missing at Random (MAR)
- Not Missing at Random (NMAR)

If the probability of missing data is the same for all classes, then the data is said to be missing completely at random. This effect implies that the causes of the missing data are unrelated to the data. A good example of why this may be happening is due to sampling and random sampling of a population where each member has an equal chance of being included in the sample. The unobserved data members are considered MCAR.

If the probability of missing is the same only in defined groups within the observed data, then the data is missing at random (MAR). An example of MAR is when a sample is taken from the population and the probability to be included depends on a given property of the data.

If the probability of missing data varies across time for reasons unknown an unrelated to the data, the data is missing not at random (NMAR). This situation is much more difficult to assess as the reasons could be several and varied. For example, this could be due to detection mechanism wear and tear over time, producing more missing data as time progresses.

The missing data properties can be solved in different ways:

- Ignore
- Discard the data rows completely
- Parameter estimation
- Imputation

**Ignore**

By ignoring the fact some data values are missing, we set up to use incomplete entries regardless in computations and calculations. It can be an option depending on the relevance of the missing values for the computations. It's a worthy option when the data available is very sparse or limited.

**Discard data**

Discarding data with missing properties means they won't be considered for calculations. This can be row-based or column-based. In the case of a dense data set, this technique has a very good positive effect because all the remaining entries are complete.

**Parameter estimation**

Parameter estimation fills unknown fields using prediction techniques based on the known values.

**Imputation**

There are several imputation techniques used to fill missing values. The most popular are:

- Average, take the mean value of all the known values
- Min, take the minimum value of all known values
- Zero, substitute empty values with zero
- Random, take a random between the min and max of known values

### 5.3.2 Aggregation functions

Aggregating data involves combining multiple rows or groups of data into a summarized format. Aggregation functions like sum, average, count, or maximum can be used to consolidate data and calculate statistics at a higher level. Aggregation functions condense multiple data points into a single value, providing insights into the overall characteristics or trends of the data-set.

Some examples of standard aggregation functions supported by both SQL and Python:

- Sum: calculates the total sum for a given group
- Count: counts the total occurrences for a given group
- Mean/Average
- Median
- Quantiles and percentiles

- Minimum/Maximum
- Standard deviation/Variance
- Mode

**Pandas aggregation functions**

These functions are usually used in the context of a group-by or aggregate clause.

- Dataframe.query: allows for boolean expressions for filtering according to the SQL terminology
- Dataframe.rolling: creates a rolling window column with lag (n)
- Dataframe.pivot: creates a pivot table
- Dataframe.melt, Dataframe.sort_values, Dataframe.sort_index: sorts and reindexes
- Dataframe.apply, Dataframe.transform, Dataframe.applymap: apply operations value by value with lambda function

**SQL aggregation functions**

These functions are usually used in the context of a HAVING, OVER, SELECT DISTINCT or PARTITION clause.

- AVG: average
- COUNT: count
- MAX, MIN
- STDEV, STDEVP, VAR, VARP
- PERCENTILE_CONT, PERCENTILE_DISC

### 5.3.3 Feature engineering & data reduction

Sometimes, data may be too large or complex for analysis. Data reduction techniques such as sampling, feature selection, or dimensionality reduction (e.g., principal component analysis) can be applied to reduce the data size while preserving its integrity Paulson et al. [2022].

### Datetime transformations

Date and time transformations involve manipulating and extracting specific information from date and time variables to derive meaningful features or facilitate analysis. These transformations allow for better understanding, interpretation, and utilization of temporal data.

**Extracting components from date time** Date time variables can be further decomposed into their constituent parts such as year, month, day, hour, minute, second, day of the week, day of the month, quarter which allows for a more granular approach with deeper insights by capturing different temporal patterns downstream.
This was the basis for a new sub-dimension table in the data model.

**Extracting components from shift** Some other useful trick is to link the shift to date time and associate each time constituent, to its shift counterpart. As a simplified example, the 17 hours of the day represent the 1 hour of shift 2.

### Sampling functions

Sampling functions are one of the backbones of the application developed as a good sampling function allows the unbiased representation of the data and extensive testing without having to cycle through the whole data-set Paulson et al. [2022]. The sampling methods used were also employed in the staging and development and still persist currently in the last release of the application to evaluate and characterize the data as well as run some computationally expensive algorithms.

**Balanced datasets** Because the data-set had multiple stratus and variables, we needed sampling functions that permitted objective assessment of a snapshot of a specific variable, all else equals. The only way to accurately compare measurements and statistics is through balanced data-sets, in order to reduce bias.
A balanced sample refers to a sample that accurately represents the proportions or distribution of different groups or categories within a population. In a balanced sample, each group or category is adequately represented, allowing for reliable statistical analysis and inference for each subgroup.
The concept of a balanced sample is particularly relevant when dealing with imbalanced data-sets, where the number of observations or instances in different groups or classes is significantly disproportionate.

Imbalanced data-sets are common in various domains, such as fraud detection, disease diagnosis, or rare event prediction.

Balanced sampling helps mitigate the issue of imbalanced data-sets and ensures that statistical analysis and modeling are not biased towards the majority group(s). It enables more accurate assessment of performance metrics, such as precision, recall, or accuracy, for different groups and improves the overall robustness of the analysis.

As an example, if we want to analyze and compare the cycle-time duration for a machine in two different periods, we ought to consider the same subgroups, same proportion of recipe types, operators, tire type, and tire diameter.

**Stratified Sampling** Stratified sampling is a sampling technique used in statistics and research to ensure that the sample reflects the characteristics or proportions of the population being studied. It involves dividing the population into homogeneous subgroups or strata based on specific variables or attributes and then selecting a proportional or representative sample from each stratum.

The process follows these steps:

- Identify the Stratification Variable: Determine the variable or attribute that will be used to divide the population into strata. This variable should be relevant and meaningful in terms of the research objectives.
- Define Strata: Create distinct and mutually exclusive strata based on the values or categories of the stratification variable. Each stratum should be internally homogeneous but have distinct characteristics compared to the other strata.
- Determine Sample Sizes: Decide on the desired sample size for each stratum. The sample sizes can be proportional to the size or proportion of each stratum in the population or can be determined based on statistical considerations.
- Combine Samples

The main advantage of stratified sampling is that it ensures that the sample includes representatives from each subgroup or stratum, thus improving the accuracy and precision of estimates for different groups within the population. By incorporating the variability within each stratum, stratified sampling allows for more accurate inferences and generalizations. The implementation can be seen in the figure 37.

```
table = "tbLive_Machine_%03d_CycleTime" % __machine

query2 = sql.select(
           sql.table(table),
           sql.func.row_number().over(order_by(sql.column("RecipeID"))).label("seqnum"),
           sql.func.count().over().label("count"),
           sql.func.count().over(partition_by(sql.column("RecipeID"))).label("cnt"))

query2 = query2.filter(sql.and_(sql.column("CyclePhaseID") == phase,
                            sql.column("CreatedOn").between('{}'.format(start_date),'{}'.for

df = pd.DataFrame(columns = ["MachineID", "RecipeID", "CreatedOn", "Duration",
    "Barcode", "OperatorID", "ShiftHistoryID"])
df = df.astype(dtype={"MachineID": "int16", "RecipeID": "int16", "CreatedOn":
    "datetime64[s]", "Duration": "float16", "OperatorID": "object",
    "ShiftHistoryID": "object"})
df = df.set_index("CreatedOn")

data = dd.read_sql_query(query2, connection_string, index_col="CreatedOn",
    npartitions = 20, meta = df)
data = data.compute()
```

Figure 37: Stratified Sampling

This example groups data by machine for a specific time frame and cycle-phase and iterates over the recipes partitions ensuring the number of recipes considered for the sample is somewhat equal, even though very likely some recipes run much more frequently due to being fast movers.

**Oversampling** Oversampling is a technique used in data pre-processing to address class imbalance in data-sets, where one or more classes have significantly fewer instances compared to other classes. The goal of oversampling is to increase the representation of the minority or underrepresented class(es) by artificially generating or duplicating samples.

The primary purpose of oversampling is to create a balanced data-set that allows machine learning models to learn from and make accurate predictions for both the majority and minority classes. By increasing the

number of instances in the minority class, oversampling helps to alleviate the bias towards the majority class and improves the performance of classifiers in predicting the minority class.
In the scenarios where the data-set is too skewed and there aren't nearly enough occurrences for a particular class, one of the solutions proposed was the oversampling for those through re-sampling and thus artificially generating and duplicating samples.

## 5.4 Performance

### 5.4.1 Wrapper Pattern

A wrapper function is a function or subroutine whose main purpose is to call a second subroutine. In object-oriented programming, the decorator pattern acts to encapsulate another object allowing a new behavior to be added incrementally without affecting the baseline behavior or this class. That being said, this allows functionality to be extended to individual components without affecting their counterparts.
This augmentation strategy was employed as a means to evaluate the application performance throughout the development stage. Performance was measured with timing (figure 38) and memory (figure 39).

```
def timing(f):
    @wraps(f)
    def wrap(*args, **kw):
        ts = time()
        result = f(*args, **kw)
        te = time()
        print ('func:%r args:[%r, %r] took: %2.4f sec' % \
          (f.__name__, args, kw, te-ts))
        return result
    return wrap
```

Figure 38: Timing decorator allows the count of elapsed time between the initial call and return of the underlying function.

```
def record_mem_usage(func):
   @wraps(func)
   def wrapper(*args, **kwargs):
      process = psutil.Process(os.getpid())
     mem_start = process.memory_info()[0]
     rt = func(*args, **kwargs)
     mem_end = process.memory_info()[0]
     diff_KB = (mem_end - mem_start) // 1000
     print('memory usage of %s: %s KB' % (func.__name__, diff_KB))
     return rt
 return wrapper
```

Figure 39: Memory usage computes the difference in memory during the program compilation.

### 5.4.2 SQL optimizations

#### SQL engine

The framework used for SQL ODBC driver handling was SQLAlchemy. SQLAlchemy uses Object Relational Mapper (ORM) logic and unit of work pattern when synchronizing changes to the database. A unit of work is started implicitly when the first SQL statement is issued against the database. All subsequent reads and writes by the same application are considered part of the same unit of work. The application must end the unit of work by issuing either a COMMIT or a ROLLBACK statement. The COMMIT statement makes permanent all changes made within a unit of work. The ROLLBACK statement removes these changes from the database. This goes even beyond the simple statements and clauses as it includes attributes assigned to objects, tracking the changes made to all rows through identity maps. This obviously involves a decent amount of bookkeeping and record control in order to keep the data flowing smoothly at a cost of some performance, especially in a lot of sequential INSERT statements (as an example).
This was the initial implementation, using SQLAlchemy and ORM module. However, SQLAlchemy also provides a built-in Core package separate of the ORM module, which includes an extensible Python-based SQL expression language, metadata and connection pool.

**Testing ORM and Core packages**

```
class Customer(Base):
    __tablename__ = "customer"id= Column(Integer, primary_key=True)
    def init_sqlalchemy(dbname = 'sqlite:///sqlalchemy.db'):
        engine = create_engine(dbname, echo=False)
        DBSession.remove()
        DBSession.configure(bind=engine, autoflush=False,
            expire_on_commit=False)
        Base.metadata.drop_all(engine)
        Base.metadata.create_all(engine)

    def test_sqlalchemy_orm(n=100000):
        init_sqlalchemy()
        t0 = time.time()
        for i in range(n):
            customer = Customer()
            customer.name = 'NAME '+ str(i)
            DBSession.add(customer)
            if i % 1000== 0:
                DBSession.flush()
        DBSession.commit()
        print"SqlAlchemy ORM: Total time for "+ str(n) + " records "+
            str(time.time() - t0) + "
        engine.execute(Customer.__table__.insert(),
            [{"name":'NAME '+ str(i)} fori inrange(n)]
        )
        print"SqlAlchemy Core: Total time for "+ str(n) + " records "+
            str(time.time() - t0) + " secs"definit_sqlite3(dbname):
        c = conn.cursor()
        c.execute("DROP TABLE IF EXISTS customer")
        c.execute("CREATE TABLE customer (id INTEGER NOT NULL, name
            VARCHAR(255), PRIMARY KEY(id))")
```

Figure 40: Testing module sourced from StackOverflow based on the insert of 10.000rows.

```
test_sqlalchemy_orm(100000)
test_sqlalchemy_core(100000)
```

Figure 41: Testing module sourced from StackOverflow based on the insert of 10.000rows.

The result obtained:

```
SqlAlchemy ORM: Total time for 100000records 16.4133379459secs
 SqlAlchemy Core: Total time for 100000records 0.568737983704secs
```

Figure 42: Test results

As it's possible to conclude by the figure 42, the difference between the modules is astonishing at least when it comes to the insertion of 10.000 rows sequentially (figure 40 and 41). On this basis, the Core engine was chosen to issue the commit statements of new data into the database.

**SQL statements over python statements**

While pandas offer an arguably much easier and more flexible alternative to SQL manipulation it comes at a huge cost of performance. After all, SQL database management systems are the primitives for data handling and are designed to efficiently store and retrieve data from the disk whereas pandas is nowhere near as capable. For very large data-sets, this carries an exponential cost.
Furthermore, SQL managers have built-in query optimizers that choose the most efficient way to compile queries and statements. Pandas does not have a query optimizer and suffers from the problems Python has (dynamically typed variables is one). While pandas can support multiple ´central processing units (CPUs) it often requires very explicit programming in order to do so, while SQL does this parallelization innately.

```
query = """SELECT RecipeID, CyclePhaseID, MachineID as Machine, CIDStart,
    CIDStartCount, Date
        FROM CIDStart
        """

query2 = """SELECT R.ID as RecipeID, R.RecipeName, CT.Name
        FROM Recipes as R, ConstructionTypes as CT
        WHERE R.TireTypeID = CT.ID AND MachineTypeID = {}""".format(machinetype)

query3 = """SELECT CID as CIDStart, Name as CIDName
        FROM CycleElementSettings
        WHERE MachineTypeID = {}""".format(machinetype)

query4 = """SELECT MachineID as Machine, Name as MachineID
        FROM MachineTable
        WHERE MachineTypeID = {}""".format(machinetype)

q1 = pd.read_sql_query(query, con= self.local_connection)
q2 = pd.read_sql_query(query2, con= self.local_connection)
q4 = pd.read_sql_query(query4, con= self.local_connection)
data = q1.merge(q2, on = "RecipeID").drop_duplicates()
data["Date"] = data["Date"].apply(lambda x: datetime.strptime(x, "%Y-%m-%d"))
data = data.drop_duplicates()
data = data.merge(q4, left_on = "Machine", right_on = "Machine")
```

Figure 43: Example of the implementation of multiple queries in Python

The figure 43 is an excellent example of several simple queries being concatenated and processed using pandas data-frames instead of using a unique complex SQL query.

### SQL procedures over inline statements

Different relational database management systems (RDBMS) have different configurations, and compilators and may operate very differently, but historically procedures are much more efficient than inline statement queries. When an SQL procedure is created, the SQL queries are separated from the procedu-

ral logic and compiled statically into sections in a package. For each section, an access plan is selected based on a DB2 optimizer.
A query plan (or query execution plan) is a sequence of steps used to access the data in a relational database. When a query is submitted, the query optimizer evaluates the best execution plan. All in all, procedures provide the following advantages:

- Pre-parsed SQL statements. The SQL statements are pre-compiled into a template and the parameters are substituted at execution time.
- Pre-generated query execution plan. The execution plan of very complicated queries is stored pre-compilation in memory, avoiding the overhead costs of the optimizer computing a new execution plan(s). This execution plan is cached and is particularly helpful in small queries performed frequently (where the execution plan would have been calculated every time).

### 5.4.3 Python Optimization

#### Global Interpreter Lock

The Global Interpreter Lock (GIL) is a mechanism used by computer language interpreters to synchronize the execution of threads so that only one native thread (per process) can run at a time. An interpreter using the GIL can only run one thread at a time, even when running on a multi-core processor. Common interpreters that use the GIL include CPython and Ruby MRI.

The use of a global interpreter lock limits the amount of parallelism reachable through the concurrency of a single interpreter process with multiple threads. If the process does not rely on outside calls, the interpreter will be locked for long periods of time and there will be no benefit from running the process on a multiprocessor machine. Also, when the single thread calls are blocking an operating system process such as disk access (such as writing data from disk), the entire process is blocked.
Python tried to solve this GIL issue by instead the multiple threads to multiple processes; making it so that each processor has a fundamental locker. It makes it hard to synchronize and communicate with threads. It is unpractical and laborious to share the memory, only can be implemented by declaring a queue.
Dask releasing the global interpreter by pandas library; Because the Dask array implements Pandas and Numpy library interface, it will allow multiple threads to run simultaneously during computation to potentially allow improvements in performance by multiple threads

### Dask Dataframes (300MB+)

By default, Dask DataFrame uses the multi-threaded scheduler. The full range Dask ecosystem can be seen in figure 44. This exposes some parallelism when pandas or the underlying NumPy operations release the global interpreter lock (GIL). Generally, pandas is more GIL bound than NumPy, so multi-core speed-ups are not as pronounced for Dask DataFrame as they are for Dask Array. This is particularly true for string-heavy Python DataFrames, as Python strings are GIL bound.

Use cases:

- Manipulating large datasets, even when those datasets don't fit in memory (above 300MB)
- Accelerating long computations by using many cores
- Distributed computing on large datasets with standard pandas operations like groupby, join, and time series computations
- Trivially parallelizable operations (such as element-wise operations, row-wise operations, loc, aggregations)
- Cleverly parallelizable operations (groupby aggregate ops, value counts, drop duplicates, merge)
- Operations requiring a shuffle

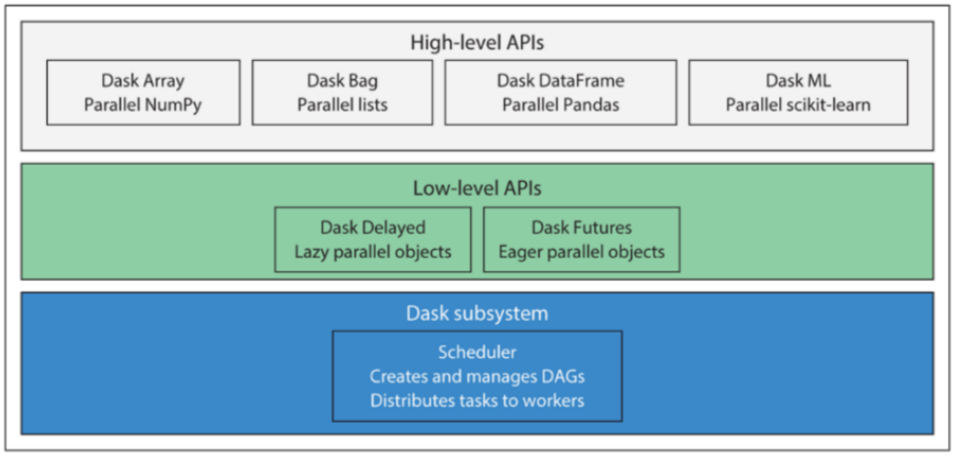


Figure 44: Dask ecosystem (Dask).

### Persistent data (<300MB)

In contexts where the data needs to be accessible throughout the program execution and it is feasible to store it outside of a physical disk using Random Access Memory resources, the best alternative would be

to store it in-memory either using an SQLite database or dask persist method.

**SQLite** A different temporary file is created each time so that, just as with the special ":memory:" string, two database connections to temporary databases each have their own private database. Temporary databases are automatically deleted when the connection that created them closes.
Even though a disk file is allocated for each temporary database, in practice the temporary database usually resides in the in-memory pager cache and hence there is very little difference between a pure in-memory database created by ":memory:" and a temporary database created by an empty filename. The sole difference is that a ":memory:" database must remain in memory at all times whereas parts of a temporary database might be flushed to disk if the database becomes large or if SQLite comes under memory pressure.
The previous paragraphs describe the behavior of temporary databases under the default SQLite configuration. An application can use the `temp_store` pragma and the `SQLITE_TEMP_STORE` compile-time parameter to force temporary databases to behave as pure in-memory databases, if desired.

**Dask persist** This turns lazy Dask collections into Dask collections with the same metadata, but now with their results fully computed or actively computing in the background. For example, a lazy dask array built up from many lazy calls will now be a dask array of the same shape, dtype, chunks, etc., but now with all of those previously lazy tasks either computed in memory as many small numpy.array (in the single-machine case) or asynchronously running in the background on a cluster (in the distributed case) (figure 45).

```
df = dd.read_csv('/path/to/*.csv')
df = df[df.name == 'Alice']
df['in-debt'] = df.balance < 0
df = df.persist() # triggers computation

df.value().min() # future computations are now fast
```

Figure 45: Example of implementation of dask persist in a dataframe.

In this example, the data is instanced by default as a collection of many lazy calls of a dask data

frame where it gathers the data frame metadata and stores the operations as future or delayed tasks. When calling the persistent method, the lazy calls will execute and be locked by the scheduler until the operations are finished and it is released. Eventually, all of the futures of this collection will be completed at which point further queries on this collection will likely be very fast. In this case, the min() computation will likely be much faster than using standard pandas library or dask compute method.
This is especially useful when loading data and using such data for a lot of sequential or complex tasks. By using `.persist()` method before passing the data to other tasks, the loading steps of the data will be run only once instead of once for every task assigned.

**Lazy evaluation for the parallelization of complex tasks**

Sometimes problems don't fit into one of the collections like dask.array or dask.dataframe. In these cases, we can parallelize custom algorithms using the simpler dask.delayed interface.
We used the dask.delayed (figure 47) function to wrap the function calls that we want to turn into tasks. None of the function calls have happened yet. Instead, the object total is a `Delayed` result that contains a task graph of the entire computation.

```
## import dependencies
from time import sleep
## calculate square of a number

def calculate_square(x):
    sleep(1)
    x= x**2
    return x

## calculate sum of two numbers
def get_sum(a,b):
    sleep(1)
    return a+b

%%time
## call functions sequentially, one after the other

## calculate square of first number
x = calculate_square(10)

## calculate square of second number
y = calculate_square(20)

## calculate sum of two numbers
z = get_sum(x,y)
print(z)
```

Figure 46: Example testing the speed of pandas sequential operations.

```
import dask
from dask import delayed

%%time
## Wrapping the function calls using dask.delayed
x = delayed(calculate_square)(10)
y = delayed(calculate_square)(20)
z = delayed(get_sum)(x, y)
print(z)
```

Figure 47: Example testing with dask parallel operations.

```
CPU times: user 2.07ms, sys:4.62ms, total:6.69ms
CPU times: user 1.72ms, sys:2.24ms, total:3.96ms
```

Figure 48: Results show a 68% gain on a simple task

### 5.4.4 Other Remarks

In most of the optimization cases, unless there is a very specific context most softwares provide already some extend of built-in optimization handles. Further optimization requires either management of CPU resources, HDD storage space, threads, processes, RAM, etc. This is not a zero-sum game whereas an increase of CPU performance may leak into additional RAM costs, as example. The best optimization strategy will always depend on the context of the problem and the resources available.

#### Time-memory tradeoff

Time–memory trade-off is a case where an algorithm or program trades increased space usage with decreased time.

The data storage consumed in performing a given task (RAM, HDD), and time refers to the time consumed in performing a given task (computation time or response time). The utility of a given space–time trade-off is affected by related fixed and variable costs (e.g., CPU speed, storage space), and is subject to diminishing returns. For example, queuing tasks add a cost in latency. Every time workers finish a task, they have to ask the scheduler what to do next and sit under-utilized, or even idle, until they get an answer.

### Recalculation and table look-ups

Another example of a common dilemma involves a lookup table which can be implemented either by addressing the entire lookup table which will consume more memory resources, or selecting the needed entries and computing the desired metrics, which will cost less memory requirements but additional computing time.

### Indexing and table scans

DBMS offers the capability of creating database index structures improving the speed of lookup operations at the cost of additional space. Without the index, table scans are necessary to locate the data. The main difference between these operations is that indexing iterates over only the index data structures and table rows are retrieved from the index search, while table scans iterate over all table rows which is much more computationally expensive.

# Chapter 6

# Implementation

Following the project's main objectives, the application was divided into 3 major modules. Each of this modules establishes different data flows and operations with common procedures and functions even though they are applied to possibly very different data (different cycle-phases and different machines).

- Triggers analysis;
- Descriptive statistics;
- Inferential statistics;

## 6.1 Introduction

The development file structure is organized by folders and within several different files according to the image below. According to figure 49, the main folders are:

- Assets: contains styling patterns and figures such as images and .css and .js files. These concern the application layout.
- Pages: contains a file per each specific application page

The main files are:

- Components: it contains the UI components built upon HTML, Dash, and the assets folder
- Graphs: in this file, there were defined functions and pieces of code related to the display of information in the pages using graphs and tables
- Database: in this file, there is a class managing context that makes all the calls and data access to the triggers database (access to the sourcing databases, queries processing, and metrics upload into a local database)

- Database CTOO: this file contains a class that creates all the connections to the data warehouse and queries mostly data about cycle times
- Statistics functions: this file has all the statistical tools used in the program
- Helper functions: a miscellaneous file with a lot of different functions, patterns and objectives
- App: it is the main file of the application connecting all the other modules.

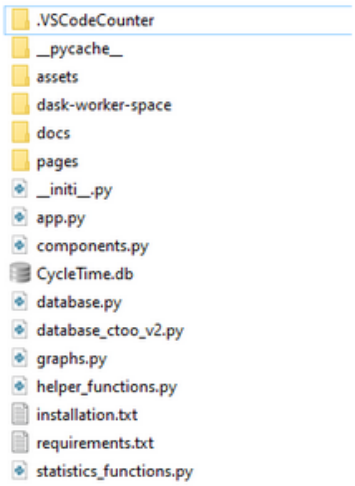


Figure 49: File structure layout.

Some other files worth mentioning are the installation text file, where general support for the installation of the application is provided along with explanations and shell commands, and the requirements file which contains all the dependencies within the project.

## 6.2 Components

### 6.2.1 Sidebar


Figure 50: Sidebar.

The sidebar (figure 50) is the main building component of the user interface as it allows the selection of different pages through *hrefs* and are organized into two collapsible modules: LOSBKM and LOSBPU which are two different machines.
The sidebar was built on top of Bootstrap sidebar with a customized layout.

### 6.2.2 Datatable class

The data table is a custom class based on the `dash.dash_table` and is an interactive table component designed for viewing and exploring large data-sets. For the purpose of visualization, edits were discarded as they were not needed at this time. The root for the tables used is `React.js` for which the python class is only a binding. The API was designed to be extensively customized through its properties and rendered with standard HTML. The creation possibilities are nearly endless, but we will focus on some key customizations of the data tables relevant to this application.

**Callbacks** Perhaps the most important feature of each component it's its ability to be responsive to page interactions. That's what callbacks are. Each page interaction generates another function in return. For

example, the simple click or manipulation of a layout component generates a trigger in another function which will then return the desired result. In the case of a data table, the simple sorting clicks by itself generates a callback on the data table function, which then updates the existing data in order to reflect the sorting desired as shown in figure 51.

```
app.layout = dbc.Container([
    dbc.Label('Click a cell in the table:'),
    dash_table.DataTable(df.to_dict('records'),[{"name": i, "id": i} for i in
        df.columns], id='tbl'),
    dbc.Alert(id='tbl_out'),
])

@callback(Output('tbl_out', 'children'), Input('tbl', 'active_cell'))
def update_graphs(active_cell):
    return str(active_cell) if active_cell else "Click the table"
```

Figure 51: A very simple callback to display the selected cell on click

**Listeners and stacking callbacks** Considering the single example listed where there is only one function and callback assigned to the table to highlight the clicked cell, in production data tables need to possess very different behaviors and be linked to different event listeners. The event listeners can be added to an event handler using the method `addEventListener()` (figure 52).

```
document.getElementById("myBtn").addEventListener("click", displayDate);
```

Figure 52: Event listener example

In this example, one element is created (a button) and the click of this button generates a callback, displayDate function. The addEventListener() method attaches an event handler to an element without overwriting existing event handlers which means several event handlers can be added (figure 53) to an element simultaneously.

```
document.getElementById("myBtn").addEventListener("click", displayDate);
document.getElementById("myBtn").addEventListener("click",
    displayLocalization);
```

Figure 53: Adding several handlers to an element without overwriting the existing context.

In this second example, two listeners are added for button click with two different callback functions. This is often useful for elements such as the data table, allowing a click to generate multiple callbacks. These callbacks can be linked to javascript code, as is the case of native sorting, or pythonic code, by performing functions linked to the python backend.

**CSS Styling** Dash data tables formatting follows the HTML table general formatting, so the easiest way to do this is by creating a CSS file and modifying it with custom formats assigning it to the desired id's classes, either using bootstrap templates, code-pen, or other online resources available. This way is by far the most creative and bound-free as the stylings are endless.

**Python Styling** Dash data tables also offer standardized pythonic styling using custom class attributes. This is the easiest way to style it since there are different custom attributes and inputs as well as predefined themes according to figure 54.

```
style_cell_conditional=[
    {
        'if': {'column_id': c},
        'textAlign': 'left'
    } for c in ['Date', 'Region']
],
style_data={
    'color': 'black',
    'backgroundColor': 'white'
},
style_data_conditional=[
    {
        'if': {'row_index': 'odd'},
        'backgroundColor': 'rgb(220, 220, 220)',
    }
],
style_header={
    'backgroundColor': 'rgb(210, 210, 210)',
    'color': 'black',
    'fontWeight': 'bold'
}
```

Figure 54: A simple example of multiple stylings using pythonic class attributes.

**Formatting** Data tables also offer formatting options, such as data validation, custom types, and formats (figure 55). The formatting options include the addition of type fields such as numeric and strings, precision or number of decimal cases, custom ascii symbols, suffix/prefix, among other configurations. This is all natively supported by the data tables without no additional required customization such as javascript functions.

```
{
'id': 'min',
'name': 'Min Temperature °(F)',
'type': 'numeric',
'format': Format(
    nully='N/A',
    precision=0,
    scheme=Scheme.fixed,
    sign=Sign.parantheses,
    symbol=Symbol.yes,
    symbol_suffix='°C'
),
# equivalent manual configuration
# 'format': {
#     'locale': {
#         'symbol': ['', '°F']
#     },
#     'nully': 'N/A'
#     'specifier': '($.0f'
# }
'on_change': {
    'action': 'coerce',
    'failure': 'default'
},
'validation': {
    'default': None
}
```

Figure 55: Field formatting for temperatures.

This simple example customizes the table with the machine minimum working temperature, adding a suffix "°C", and a precision of 0 (integer), among other features.

**Native Filtering, paging, sorting** The data table has built-in filtering, paging, and sorting with its native javascript code (figure 56). This means we only need to pass the right supported arguments with Python and it should be done without additional effort. For simple interactive tables where standard operations are required, this feature is very helpful. Any additional or custom behavior needs to be implemented on Python side.

```
dash_table.DataTable(
    id='datatable-interactivity',
    columns=[
        {"name": i, "id": i, "deletable": True, "selectable": True} for i in
            df.columns
    ],
    data=df.to_dict('records'),
    editable=True,
    filter_action="native",
    sort_action="native",
    sort_mode="multi",
    column_selectable="single",
    row_selectable="multi",
    row_deletable=True,
    selected_columns=[],
    selected_rows=[],
    page_action="native",
    page_current= 0,
    page_size= 10,
)
```

Figure 56: Using native options to customize the table. Deletable rows, paging and page size, selectability, and sorting.

However, these behaviors can also be implemented with python driven logic. An example of page sizing and paging follows in figure 57.

```
PAGE_SIZE = 5

app.layout = dash_table.DataTable(
    id='datatable-paging',
    columns=[
        {"name": i, "id": i} for i in sorted(df.columns)
    ],
    page_current=0,
    page_size=PAGE_SIZE,
    page_action='custom'
)


@callback(
    Output('datatable-paging', 'data'),
    Input('datatable-paging', "page_current"),
    Input('datatable-paging', "page_size"))
def update_table(page_current,page_size):
    return df.iloc[
        page_current*page_size:(page_current+ 1)*page_size
    ].to_dict('records')
```

Figure 57: Python driven paging.

**Exporting** In the newest versions, data tables can easily export the data to several formats upon request, such as `.xlsx` and `.csv`. However, the current design involved the addition of a button element "Export" to the layout and exporting using python pandas driven export. We can see this in action in figure 58 and 59.

```
html.Button('Export',id="download-xlsx")


@app.callback(
    Output("first_output","data"),
    [Input('download-xlsx','n_clicks'),Input("ticker", "value")],
    prevent_initial_call = True,
)
def dl_un_xlsx(n_clicks,ticker):
    while n_clicks == 1:
        df.to_excel("data_{}.xlsx".format(timestep))
```

Figure 58: Python driven export using a layout button and pandas.

Export

| State | # Solar Plants | MW | Mean MW/Plant | GWh |
|---|---|---|---|---|
| California | 289 | 4395 | 15.3 | 10826 |
| Arizona | 48 | 1078 | 22.5 | 2550 |
| Nevada | 11 | 238 | 21.6 | 557 |
| New Mexico | 33 | 261 | 7.9 | 590 |
| Colorado | 20 | 118 | 5.9 | 235 |

Figure 59: Export example with dummy data.

### 6.2.3 Graph class

The charts module inherits most of its charts from plotly.js using a simple python binding. The $dcc.Graph$ (figure 60) also supports other python figure environments such as seaborn, datashader, matplotlib and bokeh although for the purpose of streamlining the code, we will only be looking at plotly.

```
app = Dash()
app.layout = html.Div([
    dcc.Graph(figure=fig)
])
```

Figure 60: Python dash binding for Plotly graphs.

The syntax for graph creation in dash is straightforward. We create the figure using the supported libraries such as plotly figure and pass it onto the $dcc.Graph$ constructor and it will automatically convert

it to browser-interpreted code.

**Interactive Graphs** Interactive visualization is the ultimate goal of every data visualization application. For this purpose, `plotly.js` uses a vectorized backend resourcing to high-quality SVG images and WebGL for the rendering.

When it comes to interactivity, the interactions happen because every figure in dash consists of a set of attributes and properties as we have already seen. This means everything can be dynamically changed with callbacks and custom functions targeting such attributes. Starting with the basics, the dcc.Graph component has four attributes that can change through user interaction: `hoverData`, `clickData`, `selectedData`, `relayoutData`. These properties update when hovering over points, clicking on points, or selecting regions of points in a graph.

Other supported lists of interactive rich features include:

- Transitions between callbacks (figure 61)

```
'transition': {
    'duration': 500,
    'easing': 'cubic-in-out'
}
```

Figure 61: Transition example.

- Custom animations
- Cross-filtering using different graphs, buttons, dropdown or other layout elements
- Graph styling using CSS
- Zooming, selecting, hovering, clicking
- Overlapping data such as bar charts, trendlines
- Custom notes and drawings

**Sharing data between callbacks** In order to pursue complex interactions between different layout elements, it is often needed to share data between callbacks (figure 62). While it's not required, it's very

much preferred since it will improve performance, especially on large data sets. This means having one callback with multiple outputs instead of having several callbacks on the original data with one output.

```
@callback(Output('intermediate-value', 'data'), Input('dropdown', 'value'))
def clean_data(value):
    # some expensive data processing step
    cleaned_df = slow_processing_step(value)

    # more generally, this line would be
    # json.dumps(cleaned_df)
    return cleaned_df.to_json(date_format='iso', orient='split')

@callback(Output('graph', 'figure'), Input('intermediate-value', 'data'))
def update_graph(jsonified_cleaned_data):

    # more generally, this line would be
    # json.loads(jsonified_cleaned_data)
    dff = pd.read_json(jsonified_cleaned_data, orient='split')

    figure = create_figure(dff)
    return figure

@callback(Output('table', 'children'), Input('intermediate-value', 'data'))
def update_table(jsonified_cleaned_data):
    dff = pd.read_json(jsonified_cleaned_data, orient='split')
    table = create_table(dff)
    return table
```

Figure 62: Callback dependencies.

In figure 63, there are 3 callback functions but only 2 graphs are being updated and transferred to layout. The reason there is a third callback function is because the two graph callbacks depend on the same sourcing data, therefore if we want to change or query the original data, we only need to do it once before each graph updates.

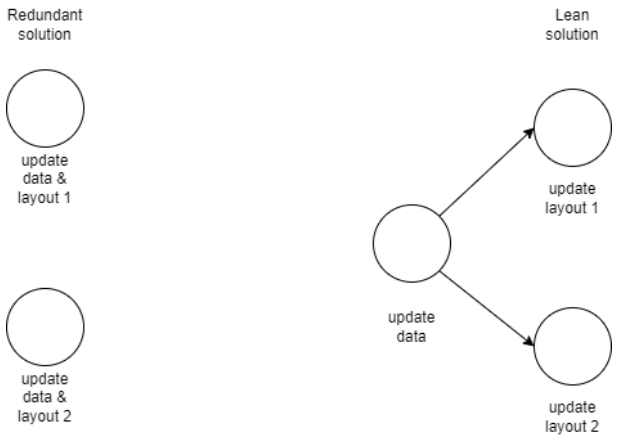


Figure 63: Understanding callbacks hierarchy and precedence in order to avoid redundant operations using data graphs.

**Caching** Another solution to share data between components and pages is to cache data on the server side. In computing, a cache is a high-speed data storage layer that stores a subset of data, typically transient in nature, so that future requests for that data are served up faster than is possible by accessing the data's primary storage location.

Caching uses Redis via Flask-Cache for storing "global variables" on the server side in a database. This data is accessed through a function (`global_store()`), the output of which is cached and keyed by its input arguments. Uses the `dcc.Store` solution to send a signal to the other callbacks when the expensive computation is complete. This signaling is performant because it allows the expensive computation to only take up one process and be performed once. Without this type of signaling, each callback could end up computing the expensive computation in parallel, locking four processes instead of one.

## 6.3 Modules

The modules are built on top of several components and consist of several callbacks. Modules were designed with the strategic goals of the application in mind and divided into three main modules:

- Descriptive statistics is the module responsible for the calculation and display of all the descriptive statistics tables and sampling.
- Triggers is the module responsible for the display and aggregation of PLC cid and triggers used in each machine and class considered.
- Inferential statistics is the module responsible for the sampling and calculation of inference metrics.

### 6.3.1 Triggers module

The triggers module does aggregations and computations per start and end cycle triggers for each machine, construction type, tire type and inch size. The triggers module is as much a support decision tool as a risk measurement since it increases the accountability and visibility of the logic circuit programming changes effected by the engineering department which otherwise would be unknown. These changes can have an impact on productivity and cycle times, thus the availability of such information to the industrial department is crucial.

#### Purpose and objectives

The main purpose of this module is to provide a holistic view of the programming cid triggers being considered for the start and end phases across each class variable. It should help the user to understand the number of occurrences of each trigger at the programmable logic circuit level and compare, for example, with the PLC charge list or other accountable information and conclude if the machine is properly programmed and considering the right trigger for the job or if it requires an engineering revision.
This module was the first to be developed and it is arguably the most important one in our system logic because it verifies before any additional computation if the data being produced is reliable. According to the data quality requirements, in order to achieve comparability between peers, the data acquisition methods must be the same between machines. For instance, if the machines are considering different triggers for the same cycle elements, can we consider the cycle elements as being the same? The answer is very likely no since the cycle phases will have different start or end triggers.

#### User inputs

From the user perspective, the first components which require user interaction are located in the topbar and are the datepicker component, cycle phases dropdown, and tire type dropdown. In addition, there are two tabs, one for the initial triggers analysis and the other for the final triggers.
The datepicker allows the user to select a date window to collect the data and perform the analysis. The cycle phases dropdown selects the cycle phase the user wants to analyze. Both of these components are mandatory. There is an additional drop-down in case the user wants to filter by tire type and it's an additional requested feature that is helpful to troubleshoot programming changes across tire types.
On an important note, these user inputs affect all the visualization elements. By querying tire type, these changes should reflect in all graphs and tables accordingly.

**Visualization elements**

For the assessment of the information, there are 3 main graphs and one accessory table. The graphs are as follows:

- Tire type sunburst chart: this chart allows the user to understand how the triggers are related to the tire type. By selecting the trigger in the analysis, the user is able to see the distribution of counts across tire types. If there is any causal relationship between the triggers and tire types (which obviously has to be determined by the engineering team), this graph will help show it.
- Machine bar chart: this machine shows the distribution of triggers across every machine. The user is able to see if the occurrence of a specific trigger is linked to a specific machine which should then be forward to the engineering once again to address the specific machine(s).
- Data table: This is a general-purpose data table whose main purpose is to support as a reference of the tables above. This table should contain all the information displayed and can be filtered or exported for better analysis or information forwarding.

The figure 64 represents a skeleton of the layout.

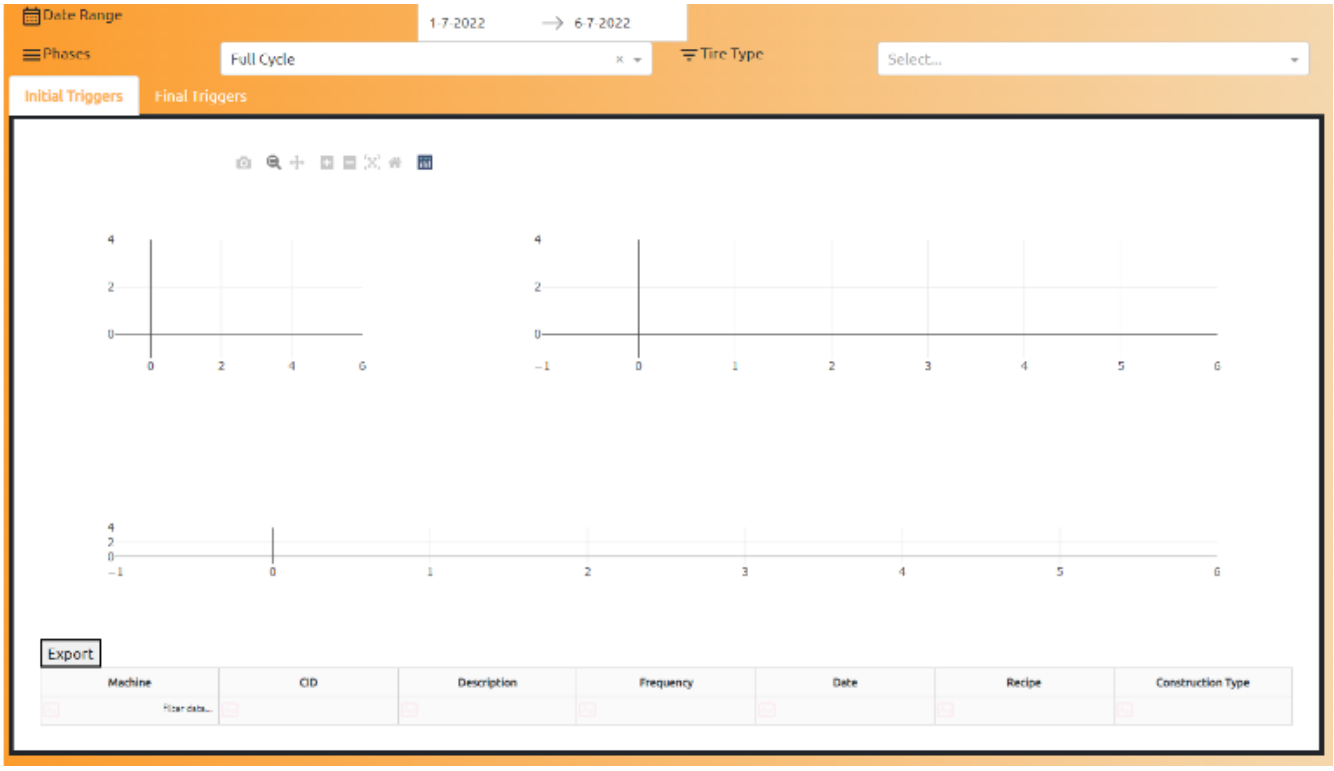


Figure 64: Page layout.

### 6.3.2 Inferential statistics

The inferential statistics module is by far the most computationally complex and heavyweight module and is thus one of the main focuses of this project. It is able to provide several cycle time estimators at different confidence levels.

#### Purpose and objectives

The main objective of this module is to be able to provide an accurate estimation of the cycle time estimators using reliable statistical methods and robust estimators.
From the user perspective, upon parameterization, it should able to return dynamically several estimators such as mean, and quartiles, among others using different methods which can then be compared or used ensemble to provide an even more robust estimator.

#### User inputs

The user inputs for this algorithm are the data range picker for which the samples are collected and the cycle phase selector. These are standard to all the modules. In addition, one key input of this module is the confidence level considered for statistical computations. The confidence level is a parameter that must be considered in every frequentist statistical analysis and refers to the probability that the real value of the original population for this statistic is within the defined confidence limits.
The predefined confidence levels are 95% and 99%.

#### Algorithm logic

The development of this algorithm posed a few challenges during the staging phases due to several factors, namely its computation complexity and lack of scalability with data set size and computational power and the fact that the calculations could not be pre-computed and stored before running (it should compute on demand). This way a key factor during development because it vastly restricted the number of estimators used and the data-set size.

**Sample size** One of the solutions to fight the scalability of the data sets is to consider a fixed-size data. The number chosen for this was 3000 samples. In order to achieve this number, a lot of runs were performed to study:

- the stability of the results from the procedures
- the total elapsed time between runs
- theoretical considerations of the algorithms used

The standard rule for every inference algorithm is the bigger and broader data set, the better results it should be able to return (in theory). However, by assessing this curve and considering the computational complexity we are able to study the amount of accuracy and stability gained per 100 additional data points per additional complexity added (algorithm duration). This was the reason why the 3000 samples were considered.

**Smoothing factor and sample size** In order to achieve better and more predictable results, we considered a higher smoothing factor in both the distribution fitting methods and the kernel density estimation. In the following graph figure 65, we can see the test runs for the distribution fit using different smoothing operators. We concluded the best approach would be to consider a smoothing factor of 4 since it provides the best results for low sample sizes (2000-3000 samples).

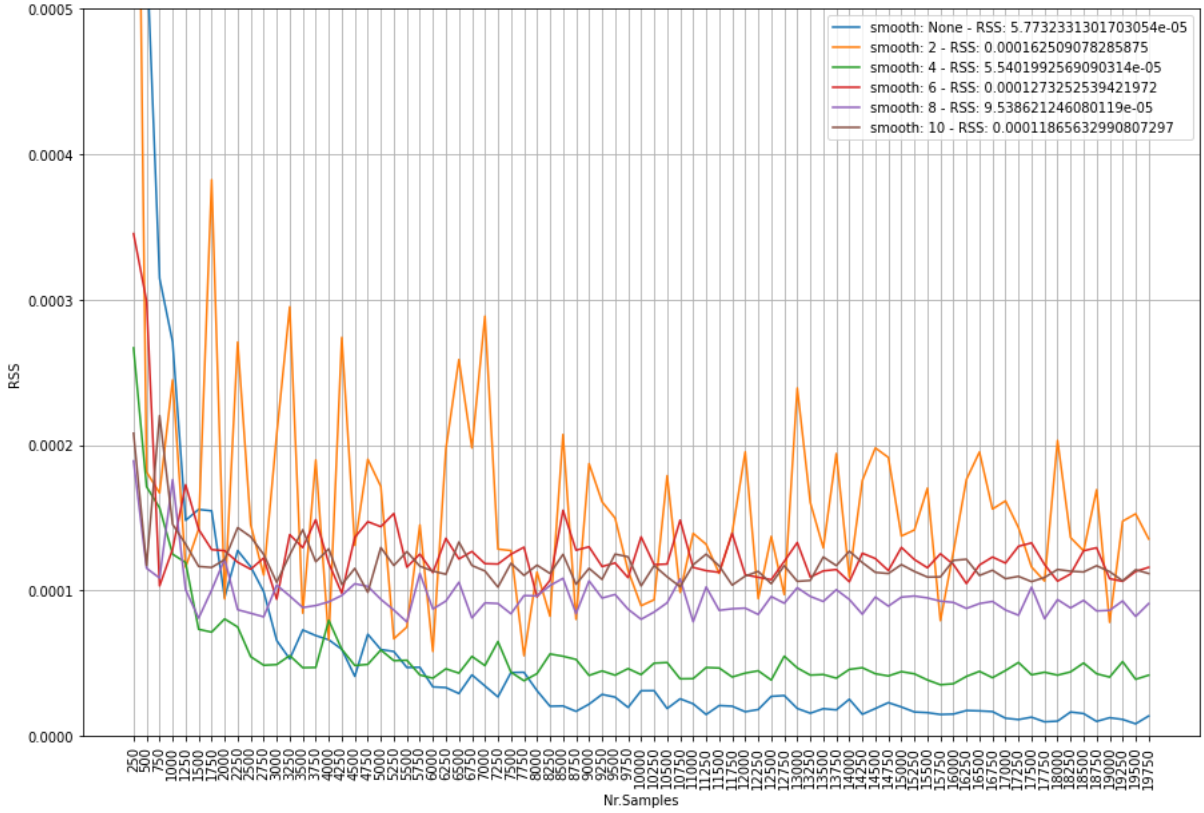


Figure 65: Smoothing factor per sample size

**Re-sampling** There are also two parameters in the table, the sample drawn and the sample used. The sample drawn is the number of samples drawn from the original data-set and the sample used is the size of the sample used to feed the algorithm. The reason why there are two separate fields for sample size is due to re-sampling methods used to ensure there is always enough data to perform computations. Therefore, if for some reason there is insufficient data being drawn from one machine (ex. date range of one day, machine has low volume due to maintenance or shift stoppage, etc) the algorithm will use re-sampling techniques to fill out the data. Thus, samples drawn between 2000-3000 samples will be accepted and considered without further re-sampling. However, for samples under 2000, the algorithm will generate enough samples to guarantee a minimum of 2000 samples.

**Choice of estimators** The procedure considers several estimators for each statistic but only displays the best-fitting ones as depicted in figure 66. When first running the program it will fit several distributions and methods and then it will evaluate the best-suited methods using the sum of squared errors metric which is the sum of the errors between the distribution of the actual data and the distribution of the estimator.

This method was requested by the users so it doesn't overcrowd and clutter the display with useless data.

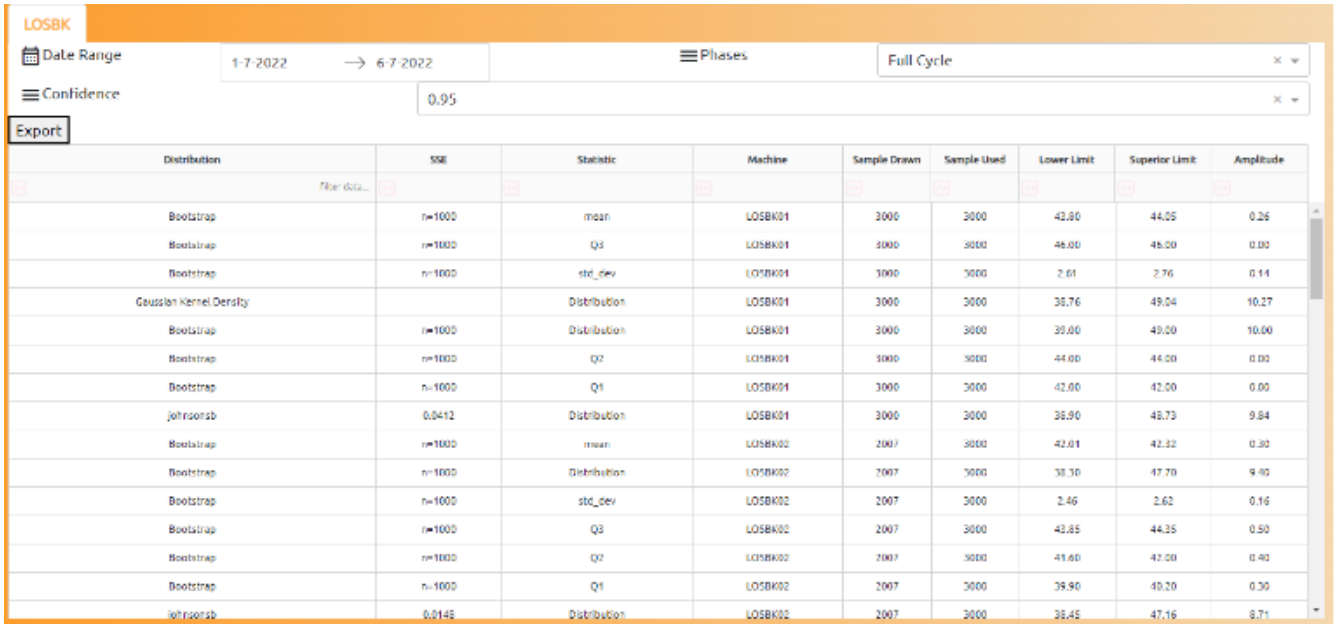


Figure 66: Page layout.

### 6.3.3 Descriptive statistics module

#### Purpose and objectives

Then descriptive statistics module collects stratified samples and aggregates several descriptive statistics per interest variable. The statistics calculated are mean, mode, standard deviation, Q1, Q2, Q3, inter-quartile range 1-3, inter-quartile range 1-2, inter-quartile range 2-3, skewness, kurtosis.

#### Algorithm logic

**Sample size** Once again, the sample size is a key variable to consider when doing dynamic studies. The population size can be as big as several thousands of lines per machine or none or close to none per machine.

For the first problem, the solution found was to draw samples. This time the algorithm is not so computationally expensive as the inferential statistics module, but the dimension of the population could still be a problem as it grows to infinity. In addition, the purpose of samples in statistics is also to be able to create a reliable inference about population statistics without running through the population. For such purpose was then considered a fractionary sampling.

The sampling function follows a hyperbolic co-secant as pictured below.

$$cschz(z) = \frac{2}{(e^{10z^{-3}} - e^{-10z^{-3}})} \tag{6.1}$$

**Stratified sampling and aggregations** In order to calculate meaningful statistics, the samples are stratified according to the different classes considered: machines, tire type, recipes, operator, shift, and rim size. The reason stratified sampling is used is to have a meaningful representation across classes and to have a balanced data set.

As an example, when calculating machine statistics, the algorithm balances out all the remaining categories of the data-set for this machine, ensuring there is a similar number of samples per recipe, operator, shift, and tire type. If this were not the case, the recipe high runners and tire types would overweight the remaining tire classes, and thus the machine statistics would be biased towards them.

# Chapter 7

# Conclusion

We have presented an end-to-end data statistical system for doing data science with relational data models. The system achieved the objectives proposed initially:

- Increase overall awareness of machine cycle time and adjacent information
- Develop a risk assessment tool to oversee the current status of validation of PLC triggers
- Develop a statistical analysis framework to infer and predict cycle times with accuracy
- A vast set of interface user requirements

Overall the system was tested in production. It was documented and its performance was tested. The current features should be reliable and work according to the description. In the ultimate analysis, these goals will align with the strategic objectives of the company.

## Prospects for future work

This tool was the first iteration of a data-driven concept to incorporate data in decision support systems. It's clearly there is still a lot of work to be done and a lot of potential to be discovered. To this end, some of the directions for future work are:

**Data validation** The current data is very unreliable and requires additional work in order to assure data quality. It also requires the implementation of fault measures and KPIs to a) prevent the occurrence of data quality concerns and b) increase the visibility and monitoring of data quality across time and machines. Quality data should be the foundation of any good data-driven module.

**Database management** The current database system is a data warehouse barely staging ready. A lot of new procedures should be implemented to clean up the data, aggregate and manipulate into higher-level metrics. The next step would be to create a reliable pipeline into a data mart with pertaining measures and views.

**Additional behaviors** The current system logic is very restrictive which can make the comparison of the data complicated. It provides a very basic framework to look into cycle times, trigger breakdown, and descriptive statistics. This could be definitely improved with new behaviors such as the longitudinal study of a class to understand the evolution of a variable across time or cross comparison across same class objects (operator, shift, machine performance comparison). The current application does not provide means to make a piece-wise analysis. That is, although the data is fed per part number, it is currently not possible to check the phase times for individual pieces.

**Rich layout dashboard** The dashboard layout has far too little and simple elements which are not so fulfilling to interact.

# Bibliography


Adetokunbo AA Adenowo and Basirat A Adenowo. Software engineering methodologies: a review of the waterfall model and object-oriented approach. *International Journal of Scientific & Engineering Research*, 4(7):427–434, 2013.

AG Continental and RBI Finanzanalyst. Continental ag. 2019.

Nello Cristianini and Elisa Ricci. *Support Vector Machines*, pages 928–932. Springer US, Boston, MA, 2008. ISBN 978-0-387-30162-4. doi: 10.1007/978-0-387-30162-4_415. URL https://doi.org/10.1007/978-0-387-30162-4_415.

Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000. doi: 10.1017/CBO9780511801389.

Dask.

Anthony Christopher Davison and David Victor Hinkley. *Bootstrap methods and their application*. Number 1. Cambridge university press, 1997.

John Dinardo and Justin Tobias. Nonparametric density and regression estimation. *Journal of Economic Perspectives*, 15:11–28, 02 2001. doi: 10.1257/jep.15.4.11.

Scott R Eliason. *Maximum likelihood estimation: Logic and practice*. Number 96. Sage, 1993.

Jiawei Han, Micheline Kamber, and Jian Pei. 4 - data warehousing and online analytical processing. In Jiawei Han, Micheline Kamber, and Jian Pei, editors, *Data Mining (Third Edition)*, The Morgan Kaufmann Series in Data Management Systems, pages 125–185. Morgan Kaufmann, Boston, third edition edition, 2012. ISBN 978-0-12-381479-1. doi: https://doi.org/10.1016/B978-0-12-381479-1.00004-6. URL https://www.sciencedirect.com/science/article/pii/B9780123814791000046.

SJ Hua and Zhirong Sun. Support vector machine approach for protein subcellular localization prediction. *Bioinformatics (Oxford, England)*, 17:721–8, 09 2001. doi: 10.1093/bioinformatics/17.8.721.

Nojun Kwak. Nonlinear projection trick in kernel methods: An alternative to the kernel trick. *IEEE transactions on neural networks and learning systems*, 24(12):2113–2119, 2013.

Guoxu Liu, Shuyi Mao, and Jae Kim. A mature-tomato detection algorithm using machine learning and color analysis. *Sensors*, 19:2023, 04 2019. doi: 10.3390/s19092023.

Continental Mabor. Continental mabor. https://www.continentalmabor.pt, 2022. Accessed: 2022-09-30.

Mihhail Matskin, Shirin Tahmasebi, Amirhossein Layegh, Amir H Payberah, Aleena Thomas, Nikolay Nikolov, and Dumitru Roman. A survey of big data pipeline orchestration tools from the perspective of the datacloud project. In *Proc. 23rd Int. Conf. Data Analytics Management Data Intensive Domains (DAMDID/RCDL 2021)*, pages 63–78, 2021.

William S Noble. What is a support vector machine? *Nature biotechnology*, 24(12):1565–1567, 2006.

Stefano Paolozzi, Pierluigi Del Nostro, Francesco Orciuoli, Pierluigi Ritrovato, and Daniele Toti. A semantic-based architecture for managing knowledge-intensive organizations: The aristotele platform. 11 2012. ISBN 978-3-642-38332-8. doi: 10.1007/978-3-642-38333-5_15.

Noah H. Paulson, Joseph Kubal, Logan Ward, Saurabh Saxena, Wenquan Lu, and Susan J. Babinec. Feature engineering for machine learning enabled early prediction of battery lifetime. *Journal of Power Sources*, 527:231127, 2022. ISSN 0378-7753. doi: https://doi.org/10.1016/j.jpowsour.2022.231127. URL https://www.sciencedirect.com/science/article/pii/S0378775322001495.

Marek Pecha and David Horák. *Analyzing l1-loss and l2-loss Support Vector Machines Implemented in PERMON Toolbox*, pages 13–23. 01 2020. ISBN 978-3-030-14906-2. doi: 10.1007/978-3-030-14907-9_2.

Amazon Redshift. Redshift. https://docs.aws.amazon.com/redshift/latest/dg/c_high_level_system_architecture.html, 2022. Accessed: 2010-09-30.

Mordor Report. Mordor inteligence report. https://www.mordorintelligence.com/industry-reports/automotive-tires-market, 2012. Accessed: 2010-09-30.

Expert Market Research. Expertmarketresearch. https://www.expertmarketresearch.com/reports/tire-market, 2010. Accessed: 2010-09-30.

Brendan Rodgers. *Tire engineering: An introduction*. CRC Press/Taylor amp; Francis Group, LLC, 2020.

Sqlalchemy. sqlalchemy. https://docs.sqlalchemy.org/en/20/intro.html, 2023. Accessed: 2010-09-30.

Leonard A Stefanski and Raymond J Carroll. Deconvolving kernel density estimators. *Statistics*, 21(2): 169–184, 1990.

George R Terrell and David W Scott. Variable kernel density estimation. *The Annals of Statistics*, pages 1236–1265, 1992.

Lipo Wang. *Support vector machines: theory and applications*, volume 177. Springer Science & Business Media, 2005.