

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Ticket Amount and Type Prediction with Time Series Forecasting

João Carlos Machado Rocha Pires



Master in Informatics and Computing Engineering

Supervisor: Prof. Luís Paulo Reis

October 13, 2023

Ticket Amount and Type Prediction with Time Series Forecasting

João Carlos Machado Rocha Pires

Master in Informatics and Computing Engineering

Approved in oral examination by the committee:

President: Prof. André Restivo

Referee: Prof. José Manuel Torres

Referee: Prof. Luís Paulo Reis

October 13, 2023

Abstract

At Infineon, every day, a large number of tickets are submitted through a specific platform and then handled by a specific team, manually and based entirely on previous experience with historical tickets. Tickets can be related to project rollouts (e.g. oneSAP), releases/applications go-live (monthly), change requests (tunings), feature bug fixes (issues/problems), and operational incidents, among others. Tickets are then ordered by priority and available resources and then handled by the specific team that attends to the requests. Although this process works, it is not effective in a way that human decisions do not always represent the best decision and can be biased. To identify which resources should be available to support the resolution of tickets, avoiding major impacts and incidents, as well as to take proactive actions and prioritize in the best possible way, this Master Thesis focuses on obtaining a Time Series Forecasting model capable of predicting and forecasting the volume of the tickets and their types, especially for project rollouts, as well as to recognize patterns and trends (e.g., end of the fiscal year or holiday season). Several Time Series Forecasting models were described as the result of the state-of-the-art research made, being then used and compared, with Lasso and Ridge Regression achieving the best results in well-known metrics. To visualize the conclusions obtained from the best-performed models, a visual interface (dashboard) displays the results. This solution should support the team that handles the tickets in the resolution of them, having a major impact in terms of performance, efficiency, and planning.

Acknowledgements

I would like to thank Professor Luís Paulo Reis and Emerson Ribeiro, the supervisors of this Master Thesis, from FEUP and Infineon, respectively, for their endless support throughout the duration of this Master Thesis.

I am also grateful for everyone at Infineon that helped me along this Master Thesis journey, especially the ones I have been working closely with, Alvin Shen, Anabela Moreira, Marlon Dopker, Nuno Nunes, and Vasco Soares.

Last but definitely not least, I am forever thankful for the unconditional love and support given by my Parents, Brother, Grandparents, and Girlfriend throughout my academic path and every day. Without you, I would be nothing.

João Carlos Machado Rocha Pires

*“Anyone who has never made a mistake
has never tried anything new.”*

Albert Einstein

Contents

Abstract	i
Acknowledgements	ii
Abbreviations	x
1 Introduction	1
1.1 Motivation	1
1.2 Objectives	1
1.3 Structure	2
2 Time Series Forecasting	3
2.1 Time Series	3
2.2 Time Series Forecasting	4
2.3 Time Series Forecasting Models and Techniques	5
2.3.1 Lasso Regression	5
2.3.2 Ridge Regression	6
2.3.3 ARIMA	7
2.3.4 SARIMAX	9
2.3.5 Artificial Neural Networks	10
2.3.6 LSTM	14
2.3.7 LSTMX	15
2.3.8 DeepAR	16
2.3.9 Prophet	18
2.3.10 Summary	21
2.3.11 Other Techniques	23
2.4 Data Preparation in Time Series Forecasting	24
2.4.1 Machine learning supervised analysis for enhancing incident management process	24
2.4.2 Effective feature pre-processing for time series forecasting	25
3 Scope and Preliminary Work	27
3.1 Technology Decision Matrix	27
4 Data	31
4.1 Data Understanding	31
4.1.1 Incidents	32
4.1.2 Tasks	34
4.2 Data Preparation	37

4.2.1	Data selection and cleaning	38
4.2.2	Data construction	40
4.2.3	Data integration and formatting	41
4.3	Data Visualization	43
5	Models	48
5.1	Train and test data sets	48
5.2	Lasso Regression	49
5.3	Ridge Regression	51
5.4	ARIMA	53
5.5	SARIMAX	54
5.6	RNN	56
5.7	LSTM	59
5.8	LSTMX	61
5.9	DeepAR	62
5.10	Prophet	71
6	Evaluation	74
6.1	Historical Evaluation	74
6.1.1	Lasso Regression	75
6.1.2	Ridge Regression	76
6.1.3	ARIMA	76
6.1.4	SARIMAX	77
6.1.5	RNN	78
6.1.6	LSTM	78
6.1.7	LSTMX	79
6.1.8	DeepAR	79
6.1.9	Prophet	80
6.2	Best models	81
6.3	Live Evaluation	82
6.3.1	Future Data Sets	82
6.3.2	Analysis of the results	83
7	Conclusions and Future Work	85
7.1	Satisfaction of goals	85
7.2	Future work	86
7.2.1	Academic future work	86
7.2.2	Company future work	86
	References	87
A	Preliminary Work	93
A.1	Work Plan	93
A.2	Communication Plan	94
A.3	Risk Management	94
B	Evaluation	96

List of Figures

2.1	Example of a Perceptron. [61]	11
2.2	ANN and Feedforward Network.	11
2.3	Direct and Indirect Feedback Network.	12
2.4	Lateral and Complete Feedback Network.	13
2.5	LSTM architecture. Source: [3]	15
2.6	DeepAR architecture for training (left) and prediction (right) stages.	18
3.1	Average score by topic and technology for the Database.	29
3.2	Average score by topic and technology for the AI.	29
3.3	Average score by topic and technology for the Dashboard.	30
4.1	Incidents distribution per year.	33
4.2	Incidents distribution in 2020.	34
4.3	Incidents distribution in 2021.	34
4.4	Incidents distribution in 2022.	35
4.5	Incidents distribution per source.	35
4.6	Tasks distribution per year.	36
4.7	Tasks distribution in 2020.	37
4.8	Tasks distribution in 2021.	37
4.9	Tasks distribution in 2022.	38
4.10	Tasks distribution per type.	38
4.11	Drop-downs to select the prediction interval for the incidents and tasks.	44
4.12	Example of the dashboard with the prediction for the next seven days.	44
4.13	Example of a window that appears on mouse click on a stacked bar in the tasks prediction graphic.	45
4.14	Set of options available for each graphic.	46
4.15	Hide (top) vs. Show (bottom) option for the graphic's caption.	46
4.16	Maximum Zoom In (top) vs. Maximum Zoom Out (bottom) option for the graphics.	47
4.17	Graphic seen in full-screen mode.	47
A.1	Gantt diagram for this Master Thesis.	93

List of Tables

2.1	Summary of the Regression Models.	22
2.2	Summary of the Neural Networks models.	23
2.3	Summary of the Advanced Models.	26
6.1	Best results obtained for the Lasso Regression model on the Tasks data set.	75
6.2	Best results obtained for the Lasso Regression model on the Incidents data set.	76
6.3	Best results obtained for the Ridge Regression model on the Tasks data set.	76
6.4	Best results obtained for the Ridge Regression model on the Incidents data set.	76
6.5	Best results obtained for the ARIMA model on the Tasks data set.	77
6.6	Best results obtained for the ARIMA model on the Incidents data set.	77
6.7	Best results obtained for the SARIMAX model on the Tasks data set.	77
6.8	Best results obtained for the SARIMAX model on the Incidents data set.	78
6.9	Best results obtained for the RNN model on the Tasks data set.	78
6.10	Best results obtained for the RNN model on the Incidents data set.	78
6.11	Best results obtained for the LSTM model on the Tasks data set.	79
6.12	Best results obtained for the LSTM model on the Incidents data set.	79
6.13	Best results obtained for the LSTMX model on the Tasks data set.	79
6.14	Best results obtained for the LSTMX model on the Incidents data set.	80
6.15	Best results obtained for the DeepAR model on the Tasks data set.	80
6.16	Best results obtained for the DeepAR model on the Incidents data set.	80
6.17	Best results obtained for the Prophet model on the Tasks data set.	80
6.18	Best results obtained for the Prophet model on the Incidents data set.	81
6.19	Best-performing models for the Tasks data set.	81
6.20	Best-performing models for the Incidents data set.	82
6.21	"Live Evaluation" for the Tasks data set for the prediction interval of 7 days.	83
6.22	"Live Evaluation" for the Tasks data set for the prediction interval of 30 days.	83
6.23	"Live Evaluation" for the Incidents data set for the prediction interval of 7 days.	84
6.24	"Live Evaluation" for the Incidents data set for the prediction interval of 30 days.	84
A.1	Communication plan.	94
A.2	Risk management.	95
B.1	Results obtained for the Lasso Regression model on the Tasks data set.	96
B.2	Results obtained for the Lasso Regression model on the Incidents data set.	97
B.3	Results obtained for the Ridge Regression model on the Tasks data set.	98
B.4	Results obtained for the Ridge Regression model on the Incidents data set.	99
B.5	Results obtained for the ARIMA model on the Tasks data set.	100
B.6	Results obtained for the ARIMA model on the Incidents data set.	101
B.7	Results obtained for the SARIMAX model on the Tasks data set.	102

B.8	Results obtained for the SARIMAX model on the Incidents data set.	103
B.9	Results obtained for the RNN model on the Tasks data set.	104
B.10	Results obtained for the RNN model on the Incidents data set.	105
B.11	Results obtained for the LSTM model on the Tasks data set.	106
B.12	Results obtained for the LSTM model on the Incidents data set.	107
B.13	Results obtained for the LSTMX model on the Tasks data set.	108
B.14	Results obtained for the LSTMX model on the Incidents data set.	109
B.15	Results obtained for the DeepAR model on the Tasks data set.	109
B.16	Results obtained for the DeepAR model on the Incidents data set.	110
B.17	Results obtained for the Prophet model on the Tasks data set.	110
B.18	Results obtained for the Prophet model on the Incidents data set.	111

Listings

4.1	JSON example for the incidents.	45
4.2	JSON example for the tasks.	45

Abbreviations and Symbols

AI	Artificial intelligence
AMP	Automatic mixed precision
ANN	Artificial neural network
API	Application programming interface
AR	Autoregressive
ARIMA	Autoregressive integrated moving average
ARIMAX	Autoregressive integrated moving average with exogenous variables
ARMA	Autoregressive moving average
CNN	Convolutional neural network
CPU	Central processing unit
DNN	Deep neural network
FC	Fully-connected
GPU	Graphics processing unit
HICP	Hybrid infineon cloud platform
HPU	Hydraulic power unit
HTML	HyperText markup language
ID	Identifier
IPU	Intelligence processing unit
IT	Information technology
JSON	JavaScript object notation
LLM	Large language models
LSTM	Long short-term memory
LSTMX	Long short-term memory with exogenous variables
MA	Moving average
MAE	Mean absolute error
MAP	Maximum a posteriori
MAPE	Mean absolute percentage error
MCMC	Markov chain monte carlo
ML	Machine learning
MSE	Mean squared error
MVC	Model-view-controller
NLP	Natural language processing
NN	Neural network
REGEX	Regular expression
RMSE	Root mean squared error
RNN	Recurrent neural network
RSS	Residual sum of squares
SARIMA	Seasonal autoregressive integrated moving average

SARIMAX	Seasonal autoregressive integrated moving average with exogenous variables
SNN	Simulated neural network
SQL	structured query language
SVM	Support vector machine
TPU	Tensor processing unit
UI	User interface
XLA	Accelerated linear algebra
XML	Extensible markup language

Chapter 1

Introduction

1.1 Motivation

At Infineon, every day, a large amount of SAP [47] tickets are submitted through a specific platform. Tickets are either incidents or tasks and can refer to project rollouts (e.g., oneSAP), releases/applications go-live (monthly), change requests (tunings), feature bug fixes (issues/problems), and operational incidents, among others. Tickets are handled by a specific team (SAP Basis), who manually orders them by priority and available resources (technical and human resources) and based entirely on previous experience with historical tickets.

The handling process is not effective since human decisions are not always the best and can be biased [14]. The decisions rely on identifying which resources should be available to support the resolution of tickets, avoiding major impacts and incidents, as well as taking proactive actions and prioritizing in the best possible way.

1.2 Objectives

This Master Thesis focuses on the creation of an AI model capable of predicting and forecasting the volume of the tickets and their types (only applicable to tasks since incidents do not have an associated type), especially for project rollouts, as well as to recognize patterns and trends (e.g., end of the fiscal year or holiday season).

The AI model is the best performing one among a set of models that are trained using historical tickets and tuned using hyperparameter optimization, with the training being repeated frequently to improve the models' accuracy and keep them up-to-date.

Each model is trained and used for forecasting tickets for each of the desired prediction intervals (7, 30, 60, 90, 365 days). Common known metrics are used to assess models' accuracy.

A dashboard is also available to display the results obtained from the built model. The dashboard is composed of two main parts. One refers to the prediction of the overall number of incidents, and another to the prediction of the overall number of tasks, with its distribution per type.

The user can select the prediction horizon and visually adapt the graphics to better analyze the predictions.

For the specific case of predictions for the next 365 days, it's also expected to have a visual clue on the possible existing patterns and trends.

This solution supports the team that handles the tickets in the resolution of them, having a major impact in terms of performance, efficiency, and planning.

1.3 Structure

This document is structured as follows: Chapter 2 includes state-of-the-art research about Time Series, its forecasting techniques, methods, and models, and Data Preparation in Time Series Forecasting. Each model is analyzed in depth in this Chapter, and some examples of their usage are presented.

Chapter 3 describes the problem and the preliminary work made during the preparation of this Master Thesis.

Then, in Chapter 4, the Data Understanding and Data Preparation processes are described in detail, in what constitutes two of the most important steps in a Time Series Forecasting project like this Master Thesis. This chapter also describes the process behind the creation of the visual interface to display the predictions obtained, as well as the dashboard components and functionality.

This chapter is followed by Chapter 5, in which the concrete implementations of all the models used in the project are presented and justified.

Afterward, the results of the models obtained with known metrics are analyzed in depth in Chapter 6, which includes two approaches (Historical and Live Evaluation) and a new method to split the train and test data.

Last but not least, Chapter 7 contains the conclusions for this Master Thesis, including the satisfaction of goals and future work.

Chapter 2

Time Series Forecasting

In this chapter, an introduction and analysis is presented to the state-of-the-art for Time Series Forecasting. The chapter is divided into several parts, with the first one being the analysis of the Time Series concept, followed by the most commonly used forecasting techniques, methods, and models, as well as the Data Preparation stage specially designed for problems similar to the one developed under this Master Thesis.

To search for the above-mentioned topics and related work, both *Scopus* [12], an abstract and citation database, and *Elicit* [41], an AI-powered research assistant that uses LLM, were used.

Mendeley [11] was used to collect all the documents that were read and analyzed and manage the corresponding references.

The literature about the above-mentioned topics and the related work search was done after identifying the following information needs:

- "time series"
- "time series forecasting"
- "time series forecasting" AND "ticket"
- "time series forecasting" AND "amount"
- "ticket analysis" OR "incident analysis"
- "machine learning" AND "ticket management"

The selection criteria were based on the number of citations, relevance, and similarity of the retrieved documents.

2.1 Time Series

The definition of Time Series, according to the Cambridge Business English Dictionary [10], is "a list of numbers relating to a particular activity, which is recorded at regular periods of time and then studied. Time series are typically used to study, for example, sales, orders, income, etc."

In the same direction, the Encyclopedia of Biometrics [1] defines Time Series as "a sequence of data points, measured typically at successive times, spaced at (often uniform) time intervals."

According to Margherita Gerolimetto, in the paper titled "Introduction to time series analysis" [18], time series are "any (finite or infinite) sequence of observations indexed by an ordered set ('time')."

In the same paper, a distinction is made between continuous and discrete Time Series. The first are those where the observations can be obtained at any point in time. The second refers to all whose observation can only be obtained in a limited set of time values.

For the discrete Time Series, there is also a distinction between the levels and flow types. While the levels are Time Series registered instantaneously, flows are cumulated Time Series over a time interval.

In a nutshell, Time Series is a list of data points whose occurrence is time-dependent, with the interval of time between successive data points being equal most of the time.

An example of a Time Series that everyone had contact with was the daily cases of COVID-19. In this case, the data points were the number of infections, which were associated with the day on which the same number was registered.

In the scope of this Master Thesis, the Time Series, as analyzed in depth in Chapter 4, is a list of data points, each representing a ticket and its characteristics, with the time component being the day in which each ticket was submitted. The Time Series used in this Master Thesis is a discrete and levels-type one.

2.2 Time Series Forecasting

A common usage of a Time Series is to forecast future occurrences of similar data points that make up the Time Series.

As mentioned in the Encyclopedia of Biometrics [1], "learning on time series data attempts to understand the relationship between data and time, e.g., making forecasts. The order of the data points along the time axis is an important factor to be considered in the learning methods."

This means that models are used to understand and extract the relationship between data and time, but also that, for the purpose of this forecasting, the data points must be ordered chronologically so that, in case there are any patterns and/or relations, these can be extracted and used to predict similar occurrences in the future. However, while this is a mandatory step for traditional models, new ones are tolerant of the non-ordering of the data by date, with some requiring that the data is not chronologically ordered at all, as explained later in this Master Thesis.

According to the paper "Introduction to time series analysis" [18], Time Series Forecasting has two main objectives:

1. Develop models for describing the behavior of individual or multiple time series.
2. Propose a methodology for specifying, estimating, and validating (assessing) an appropriate model for specific data.

In the same paper, multiple Time Series Forecasting problems are described. For this Master Thesis, the problem that is being addressed is forecasting within a time interval.

2.3 Time Series Forecasting Models and Techniques

As mentioned before, in order to forecast future occurrences of a Time Series, several models can be used.

In this section, a theoretical explanation is provided for each of the selected models used in the scope of this Master Thesis. Each of the subsections is dedicated to a model, containing its presentation and the references to related selected documents. Each document is summarized and compared against the goals of this Master Thesis.

2.3.1 Lasso Regression

In 1996, Robert Tibshirani coined the Lasso Regression term [75], whose origin goes back to the geophysics area in which this kind of regression was first introduced.

The Lasso Regression stands for Least Absolute Shrinkage and Selection Operator and is a popular regularization technique used in ML projects for prediction and feature selection tasks. It is characterized by being a model that achieves high accuracy with simplicity and a lower number of parameters. It is also a model that achieves good results for high-dimensional data sets (i.e., a high number of features).

It is based on the basic linear regression formula, where y is the dependent variable (target) and $x_1, x_2, \dots, x_p$ are the independent variables. The β values are the coefficients to be predicted, and the ε is the error.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p + \varepsilon$$

The additional penalty introduced by Lasso Regression is obtained with the L1 regularization, which sums the absolute values of the coefficients to be predicted and multiplies the sum by a regularization parameter (λ) that controls the level of regularization applied to the model.

$$L1 = \lambda * (|\beta_0| + |\beta_1| + |\beta_2| + \dots + |\beta_p|)$$

The practical usage of the L1 regularization technique is that it shrinks some coefficients to zero when λ is high enough, automatically doing the feature selection task. Thus, it is crucial to define the tuning parameter (λ) correctly.

When equal to 0, the tuning parameter makes the Lasso Regression a Linear Regression, and when close to infinity, it eliminates more features. Increasing the tuning parameter means an increase in bias, while a decrease reflects a variance increase.

On the theoretical side, L1 regularization is used to avoid over-fitting the data.

The objective of the Lasso Regression is to find the coefficient values that minimize the L1 regularization term and, at the same time, minimize the sum of the squared difference between the observed values and the predictions (RSS).

$$RSS = \sum_{i=1}^n (y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij})$$

In a nutshell, Lasso Regression tries to minimize $RSS + L1$.

To find the best coefficient values, it is common practice to use an optimization algorithm such as the Coordinate Descent in parallel, which updates each coefficient iteratively while keeping the remaining unchanged.

An automated k-fold cross-validation approach is also a common practice [42]. In this approach, the data set is divided randomly into k sub-samples of the same size, using k-1 of those for the model construction and the remaining one for validation purposes. This procedure is made k times, obtaining k number of λ and selecting the best one to be used in the model.

While Lasso Regression has some advantages, it also has some disadvantages, like the lower accuracy and loss of information when in a data set there are multiple variables with some correlation between them. In the presence of correlated variables, Lasso Regression retains one of them and shrinks the coefficients of the others to zero.

2.3.2 Ridge Regression

In 1970, in a paper entitled "Ridge Regression: Biased Estimation for Nonorthogonal Problems" [21], Arthur Hoerl and Robert Kennard introduced the Ridge Regression model, which is similar to the Lasso Regression.

Both take as starting point the Linear Regression, but while Lasso uses the L1 regularization technique, the Ridge Regression uses the L2 regularization, which is particularly important when the data set has multicollinearity.

Multicollinearity in a data set means that at least two independent variables have a correlation, making it difficult to determine the individual effect of each to determine the dependent variable. In this specific case, variances are large, and least-squares are unbiased, meaning the difference between predicted and observed values is significant.

Like Lasso Regression, Ridge Regression also addresses the issue of over-fitting in Linear Regression, which comes from the fact that variables can be highly correlated and prone to over-fitting while making the model complex at the same time.

The L2 regularization term is introduced by Ridge Regression as a penalty term to the error function, which shrinks the coefficients towards zero, thus reducing their magnitudes. While doing this, the coefficients cannot become too large, which also reduces the complexity of the model.

$$L2 = \lambda * (\beta_0 + \beta_1 + \beta_2 + \dots + \beta_p)^2$$

Here, the λ is the penalty term, reducing the magnitude of the coefficients with an increase in its value. It also shrinks the parameters, preventing multicollinearity and reducing the model complexity.

With the application of the L2 regularization, Ridge Regression achieves a balance between the importance of individual data points and the capture of underlying patterns in data.

Ridge Regression is also effective against outliers, despite not being as effective as Lasso Regression. In Linear Regression, the coefficients can be highly influenced by outliers, while in Ridge Regression, with the L2 regularization, the coefficients are shrunk, thus reducing the influence of the outliers.

In the same direction as Lasso Regression, the objective of Ridge Regression is to minimize $RSS + L2$.

Usually, Ridge Regression achieves better prediction results and accuracy, but it also has some disadvantages.

One of the main disadvantages results in the obtained coefficients being biased. As the L2 regularization shrinks the coefficients toward zero, that means that the obtained coefficients only provide a reduced version of the original magnitude, thus not indicating the relationship between a feature and the outcome variables.

Another practical disadvantage is that the application of this model to a large data set can become computationally expensive. It is a model that is hard to optimize, and the interpretation of the results is also difficult because of the modification made to the coefficients.

2.3.3 ARIMA

ARIMA stands for Autoregressive Integrated Moving Average and is a generalization of the Autoregressive Moving Average (ARMA) model [38].

ARMA was first introduced in 1951 in a thesis entitled "Hypothesis testing in time series analysis" [8], of the authorship of Peter Whittle.

ARMA models are used in time series analysis whenever the time series is stationary [64]. Being stationary means that its statistical properties do not depend on time t , and it may be only stationary with respect to one characteristic and not to another. A stationary time series is far more easy to predict than a non-stationary time series due to its known statistical properties.

The time series that are used in ARMA models are generated by passing white noise (i.e., stationary time series or stationary random process with zero autocorrelation, which means it is an experiment that produces as output functions of a single continuous variable, which is time, in this case) through a recursive and a non-recursive linear filter, consecutively. A linear filter is a filter whose output is a linear function of the input.

This kind of model results from the combination of an autoregressive (AR) model [37] with a moving average (MA) model [36].

On the one hand, an AR model is a model in which each value to be predicted in a time series depends on the previous one(s). This model assumes that there is a strong correlation between the previous values and the future ones. In the most simple case, the AR model equation is the following.

$$y_t = c + \phi_1 y_{t-1} + \varepsilon_t$$

In the above equation, y_t is a variable to be predicted at time t , y_{t-1} is the value on the immediate before time, c is a constant, ε_t is the error, and ϕ_1 is a value between -1 and 1 that represents the relevance of the previous value in the prediction of the current one.

To generalize the AR equation, one can consider P as the number of previous values to be used in the calculation of y_t . With that, the equation for AR is the following.

$$y_t = (\sum_{i=1}^P \phi_i y_{t-i}) + c + \varepsilon_t$$

The value for P ranges from 1 to the number of available previous values.

On the other hand, a MA model is a model that takes into account previous errors to predict future values. This model assumes that there is a strong correlation between the previous errors and the future values. In the most simple case, the MA model can be defined with the following equation.

$$y_t = c + \theta_1 \varepsilon_{t-1} + \varepsilon_t$$

In the above equation, y_t , c , and ε_t have the same meaning as for the AR equation. ε_{t-1} represents the error from the previous time ($t-1$), and θ_1 is the value that represents the relevance of that error for the calculation of y_t .

Again, like for the AR model, to generalize MA, one can consider Q as the number of previous errors to be considered in the calculation of y_t . That said, the general equation of MA is the following.

$$y_t = (\sum_{j=1}^Q \theta_j \varepsilon_{t-j}) + c + \varepsilon_t$$

Q can take values from 1 to the number of previous values.

By joining the equations of the AR and MA models, in the simple case, it is possible to obtain the equation for the ARMA model, which is the following.

$$y_t = c + \phi_1 y_{t-1} + \theta_1 \varepsilon_{t-1} + \varepsilon_t$$

While the above equation takes into consideration only the last predicted value, which is the simplest case, the model can be more complex and also consider a variable number of previous values and errors. Considering the P value from AR and the Q value from MA, ARMA can be defined as ARMA(P,Q). That said, the general formula for ARMA(P,Q) is the following.

$$y_t = (\sum_{i=1}^P \phi_i y_{t-i}) + (\sum_{j=1}^Q \theta_j \varepsilon_{t-j}) + c + \varepsilon_t$$

The ARIMA model, contrary to the ARMA model, can also be used with non-stationary time series [5]. This is only possible due to the differencing that is applied to non-stationary time series, which converts them into stationary time series, being thus compatible with the ARMA approach.

The integrated component of the ARIMA model extends the ARMA model and is the one responsible for the differencing. That said, ARIMA is defined as ARIMA(P,D,Q), where P and Q

have the same meaning as for the ARMA model, and D is the number of times that the difference is taken until the time series becomes stationary.

Considering $By_t = y_{t-1}$, where B is the Backshift Operator, the first order difference is defined as

$$y'_t = y_t - y_{t-1} = (1 - B)y_t$$

.

Thus, the general d -th order difference equation is the following.

$$y'_t = (1 - B)^d y_t$$

With this, the ARIMA(P,D,Q) model can be generally defined as follows.

$$y'_t = (\sum_{i=1}^P \phi_i y'_{t-i}) + (\sum_{j=1}^Q \theta_j \epsilon_{t-j}) + c + \epsilon_t$$

The ARIMA model is known to be computationally expensive and for not have a clear methodology to determine the values of P , D , and Q .

The paper [19] published in 2015 by the authors A. Gordiievych and I. Shubin introduces the usage of ARIMA models to predict the air ticket prices, taking into account extensions that accommodate preferences such as the number of stops in the itinerary or the specific airline to be used.

The document lacks important information regarding the setup made to the models used and the evaluation of them. However, it shows the importance of analyzing the data domain and its characteristics very carefully, which is the main acknowledgment, alongside the usage of the classic ARIMA models that performed well for the scope of the project.

The paper [6], published in 2021 by the authors Q. Chi, M. Liu, and B. Yang, also focuses on the prediction of air ticket prices using an ARIMA model. However, the goal of this work is not to create a model that can predict future prices but rather to create a model that fits into the existing data, thus being able to explain the variation and fluctuation of the prices over the years.

Apart from the main acknowledgment that is the usage of an ARIMA model not only to predict but also to adapt to the historical data, this paper also highlights the importance of the peaks in the prices, the majority related to vacations and other festivities. This seasonality is taken into consideration in the SARIMAX model that is presented next.

2.3.4 SARIMAX

SARIMA stands for Seasonal Autoregressive Integrated Moving Average and is an extension of the ARIMA model, including the seasonal component [4].

The SARIMA model is the result of the combination of the ARIMA model with a seasonal component that contains the AR and MA components lagged by the frequency of the seasonality (e.g., hourly - 24, monthly - 12).

That said, the general simple formula of SARIMA can be defined as follows.

$$y'_t = ARIMA + \gamma_1 y'_{t-s_1} + \eta_1 \varepsilon_{t-s_1} = \phi_i y'_{t-i} + \theta_j \varepsilon_{t-j} + c + \varepsilon_t + \gamma_1 y'_{t-s_1} + \eta_1 \varepsilon_{t-s_1}$$

In the above equation, y'_{t-s_1} represents the lagged seasonal value (time $t - s_1$, where s_1 is the seasonality), γ_1 represents the relevance of that value for the calculation of y'_t , ε_{t-s_1} represents the lagged seasonal error (time $t - s_1$), and η_1 the relevance of the lagged seasonal error to the calculation of y'_t . All the remaining components have the same meaning as in the ARIMA equation.

To generalize the simple equation that defines SARIMA, there is the need to consider three new parameters, P' , Q' , D' , and s .

P' is the number of lagged seasonal values used in the calculation of y'_t and can take values from -1 to 1, while Q' is the number of lagged seasonal errors and can take values from -1 to 1 and D' the number of lagged times that the difference is taken until the time series becomes stationary, both used in the same calculation and s is the seasonality frequency.

That said, the general formula for SARIMAX(P,D,Q)(P',D',Q')s is the following.

$$y'_t = (\sum_{i=1}^{P'} \phi_i y'_{t-i}) + (\sum_{j=1}^{Q'} \theta_j \varepsilon_{t-j}) + c + \varepsilon_t + (\sum_{k=1}^{P'} \gamma_k y'_{t-sk}) + (\sum_{l=1}^{Q'} \eta_l \varepsilon_{t-sl})$$

The SARIMAX is an extension of the SARIMA model to include exogenous variables, i.e., external data that adds value to the time series.

In the simple case, the equation for SARIMAX is the following.

$$y'_t = SARIMA + \beta_1 x_{1_t} = \phi_i y'_{t-i} + \theta_j \varepsilon_{t-j} + c + \varepsilon_t + \gamma_1 y'_{t-s_1} + \eta_1 \varepsilon_{t-s_1} + \beta_1 x_{1_t}$$

In the above equation, x_{1_t} is an exogenous variable on the time t , and β_1 is the relevance of this variable to the calculation of y'_t .

By generalizing the above equation to include a dynamic number of exogenous variables, the SARIMAX equation is as follows.

$$y'_t = (\sum_{i=1}^{P'} \phi_i y'_{t-i}) + (\sum_{j=1}^{Q'} \theta_j \varepsilon_{t-j}) + c + \varepsilon_t + (\sum_{k=1}^{P'} \gamma_k y'_{t-sk}) + (\sum_{l=1}^{Q'} \eta_l \varepsilon_{t-sl}) + (\sum_{n=1}^m \beta_n x_{n_t})$$

In the above equation, m is the number of exogenous variables.

2.3.5 Artificial Neural Networks

Artificial Neural Networks (ANN), also known as Neural Networks (NN) or Simulated Neural Networks (SNN), are networks of layers with nodes whose structure resemble the human brain, with the nodes being the artificial representation of neurons [23]. They were first introduced in 1958 by Frank Rosenblatt [44], whose work is based on the one from Warren S. McCulloch and Walter Pitts (1943) [35], through its simplest form called Perceptron.

A Perceptron is an artificial neuron, or node, with one or more inputs, the corresponding weights, a bias (also known as threshold), and an output, calculated through the usage of an activation function that is linear, as represented in figure 2.1.

Each input x_i is multiplied by the corresponding weight w_i , and the sum of these multiplications is compared against the threshold. If higher, the node is activated (output set to 1). Otherwise, the node is not activated, and the output is set to 0. The following equation summarizes these calculations.

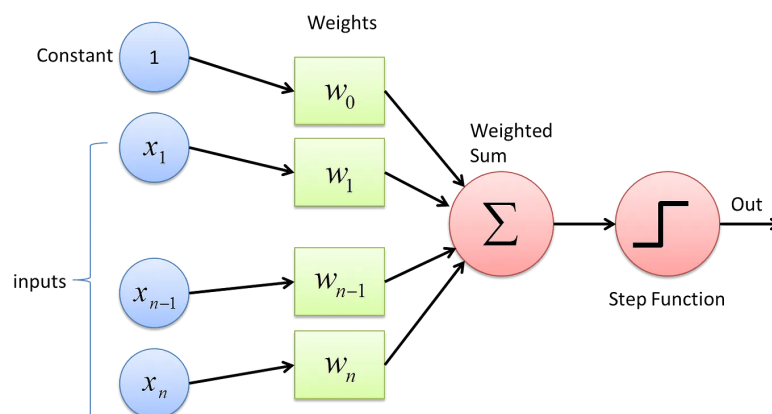


Figure 2.1: Example of a Perceptron. [61]

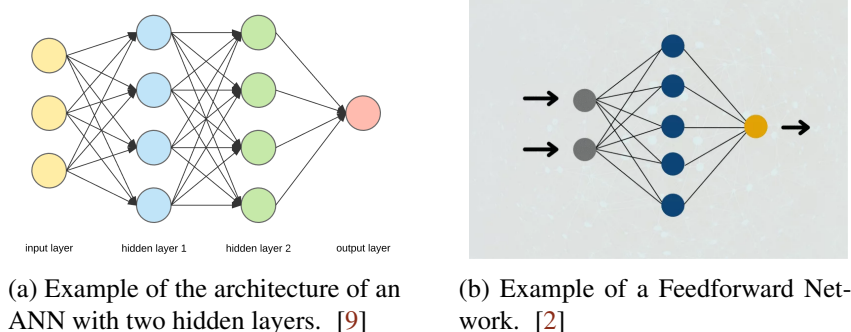
$$f(x) = \begin{cases} 1, & \text{if } (\sum w_i x_i) + bias \geq 0 \\ 0, & \text{otherwise.} \end{cases}$$

In the above equation, $f(x)$ is the activation function that determines the output of the node, which can be either 0 (not activated) or 1 (activated), w_i is the weight i , and x_i is the value i . As it is easily recognizable, higher weights have more influence on the node's output.

ANN are networks of nodes, with one input layer that contains one or several nodes, an output layer with one or several nodes, and one or more hidden layers in the between with several nodes, as seen in figure 2.2a. Each node is connected to other nodes and has an associated weight and threshold (also known as bias). The threshold of each node is used to validate the data that is in the output. If the output value is above the node's threshold, then the node is activated, and the data is sent to the next layer of the network. Otherwise, the data is not passed to the network in the same logic as the one for the Perceptron.

The biggest difference between the Perceptron and a node in an ANN is that, while the Perceptron uses a linear activation function, the node of an ANN uses a sigmoid activation function.

The node's output, except for the nodes in the output layer, is the input for other nodes. If those nodes are located just in the following layers, the network is called Feedforward Network.



(a) Example of the architecture of an ANN with two hidden layers. [9]

(b) Example of a Feedforward Network. [2]

Figure 2.2: ANN and Feedforward Network.

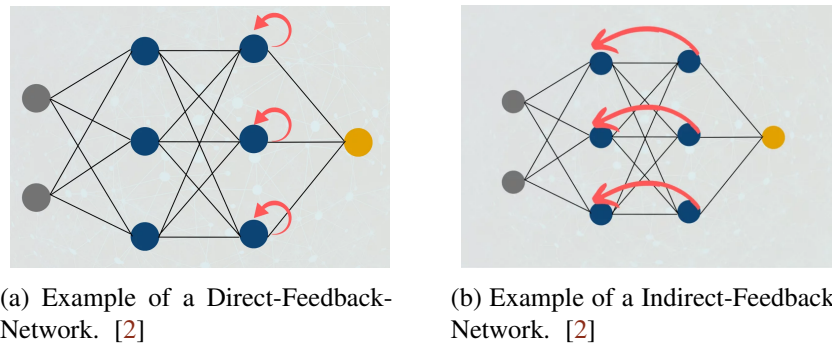


Figure 2.3: Direct and Indirect Feedback Network.

Feedforward networks were introduced in 1965 in the work of Alexey Grigorevich Ivakhnenko and Valentin Lapa [13]. Figure 2.2b shows an example of a Feedforward Network.

However, Feedforward Networks are not the only kind of networks. Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN) are also two types of ANN.

RNNs were first introduced in 1986 by David Rumelhart [34], and they differ from Feedforward Networks because the nodes of a RNN can also connect its output to the input of nodes in the same or previous layers, and not just to the following layers.

RNNs are used for NLP and time series forecasting projects since, in these areas, the sequential relationship that is part of the nature of a RNN is of such importance.

There are four types of RNN:

- Direct-Feedback-Network - the output of a neuron is simultaneously its input (figure 2.3a).
- Indirect-Feedback-Network - the output of a neuron is the input of a neuron in a previous layer (figure 2.3b).
- Lateral-Feedback-Network - the output of a neuron is the input of a neuron in the same layer (figure 2.4a).
- Complete-Feedback-Network - the output of a neuron is the input of all the neurons in the network (figure 2.4b).

Like in every prediction model, the goal of a RNN is to minimize the difference between the predicted values and the observed ones. To do so, the training of a RNN consists of obtaining the best weight values in the network that make this difference as little as possible.

The training process of a RNN includes two main steps: forward propagation and backpropagation. In the first one, forward propagation, the input sequence is passed to the network, and the output is computed every time step, being then compared to the actual output with a loss function to assess the error. This error is then backpropagated through the network, and the weights are updated using an optimization algorithm.

RNNs have a set of problems, like the possibility of the backpropagation leading to a small gradient that can lead to slow or even impossible training. Another problem is that RNNs have

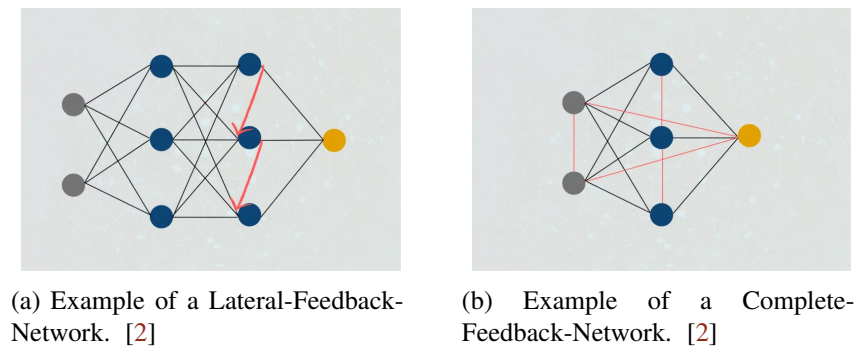


Figure 2.4: Lateral and Complete Feedback Network.

a short-term memory in regards to retaining information from previous layers into the current neuron.

The first works on CNNs date back to 1980 and 1989, when Kunihiko Fukushima [17] and Yann LeCun [27], respectively, introduced this type of ANN that is mainly used in classifications and computer vision tasks because of their better performance with media (audio and image) [22].

These kinds of ANNs have three types of layers:

- Convolutional layer,
- Pooling layer,
- and Fully-connected (FC) layer.

The convolutional layer is the first in a CNN, followed by other convolutional or pooling layers. The FC layer is always the last in a CNN. The complexity of a CNN increases over its layers, with the first ones having a lower complexity when compared to the last.

Starting with the convolutional layer, which is the one responsible for the majority of the computation, this layer contains the input data, a filter, and a feature map. The application of the filter, which contains weights, to the input data is what gives the name to the layer, as this process is known as a convolution.

The convolution process produces as output an array with the dot products of the filter and input, which is the feature map. Once finished, a Rectified Linear Unit (ReLU) transformation is applied to the feature map, and that introduces non-linearity to the CNN.

The pooling layer, like the convolutional layer, applies a filter to the input, but the differences are that the filter does not contain weights, and the goal is to reduce the dimension of the input. This reduction is achieved by using an aggregation function that produces an output array. This aggregation can be achieved with an average function (average of the input) or with a maximum function (maximum of the input).

The reduction that is made in this layer has as its main disadvantage the loss of information, but it also has some known advantages such as complexity reduction, improved efficiency, and limits the overfitting risk.

Last but not least, the nodes in the FC layer have as input the output of a node of a previous layer. It is the final layer of a CNN and has as its main goal the classification task based on the processing made in the previous layers. Softmax is the activation function used in this layer, most of the time, producing a probability between 0 and 1.

ANNs are also the basis for Deep Learning Algorithms, which use Deep Neural Networks (DNN). If a Neural Network has 3 layers (including the input and output layer), it is called ANN. Otherwise, it is a DNN.

The document [62] published in 2014 by the authors M. Sodanil and P. Chatthong proposes the usage of an ANN to predict the amount of solid waste in Bangkok.

The ANN is built using a backpropagation algorithm, i.e., a RNN, and the performance vector is used to evaluate the model. The performance vector is obtained with the actual and predicted values, the MSE, and a custom correlation.

The optimum number of neurons was chosen by trial and error in both the input and hidden layers. For the output layer, only one neuron, the prediction for the next time window, was used, as expected.

The sigmoid function was also used as an activation function in the hidden layer, while the linear function was the one used in the output layer. Due to the nature of the RNN architecture, the weights and biases were adjusted dynamically during the training process.

The main acknowledgments of this paper are the usage of a RNN, the fact that the optimal number of neurons was obtained by trial and error, and the usage of the sigmoid and linear functions in the hidden and output layers, respectively.

2.3.6 LSTM

LSTM stands for Long Short-Term Memory, and this kind of network was invented by Sepp Hochreiter and Jürgen Schmidhuber in 1997 [20].

LSTM networks are a sub-type of Recurrent Neural Networks (RNN), introduced in the previous section, whose origin is motivated by the resolution of the problems that RNNs have.

The main purpose of a LSTM network is to retain the information from previous layers for as long as possible in a cell state, also called long-term memory. The information from a previous layer, which was already taken into account by the RNNs, is stored in the hidden state, representing the short-term memory.

Every time step, the input of the current neuron, $x(t)$, is also augmented with the previous hidden state, $h(t-1)$, and the previous cell state, $c(t-1)$, as it is possible to see in figure 2.5.

Both the input of the current neuron and the previous hidden state pass through a forget gate, in which this information is multiplied by a value from 0 to 1 to decide which part of it is relevant to be stored in the cell state. If this value is 0, it means all the information is discarded, and if it is 1, it means that all the information is kept. Any value in the between results on only part of the information to be stored in the cell state. The result of this combination of input, hidden state, and value is then multiplied by the cell state value.

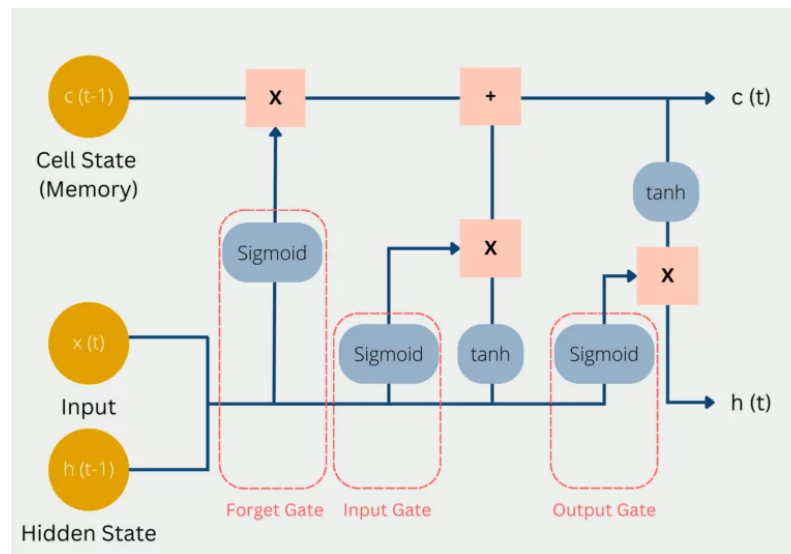


Figure 2.5: LSTM architecture. Source: [3]

The input gate assesses how important is the input of the current neuron by multiplying it with the previous hidden state and with the weight matrix from the last execution. The result of this calculation is then added to the cell state, which becomes the new value to be used in the next execution.

Last but not least, the output gate calculates from the hidden state the output of the current execution, using a sigmoid function, multiplying it for the cell state that was previously activated with the tanh function.

2.3.7 LSTMX

LSTMX networks are for LSTM as SARIMAX is for SARIMA. LSTMX stands for Long Short-Term Memory with Exogenous variables and is an extension of the LSTM model to include external data to the time series or any other data used in the model.

The inclusion of exogenous variables in LSTMX networks depends on the architecture selected by the developer and differs from project to project, depending on the special needs of the data.

In this document [24], published in 2021, the authors K. Imanaka and C. Sakama focus on the usage of neural networks to predict air ticket demand based on past ticket sales.

There is a strong relation between the goals of this document and the ones of this Master Thesis, with the main difference being the area on which the data relies.

From the models used and evaluated, the authors concluded that the LSTMX and SARIMAX models were the best performed, with LSTMX achieving a better RMSE metric result and capturing dynamics in short periods better.

From the modifications made to the original models, two are particularly important. The separate processing of the time series data and non-time series data in the LSTM block and the

introduction of the AR component apart from the neural network, which captures the short and long-term patterns in the time series data.

One of the conclusions of this document was that the month of departure date and the day of the week are effective in predicting the fluctuation. Another conclusion was that the fluctuation prediction would be more accurate if longer-term training data of several years were used.

Both conclusions were taken into account for this Master Thesis, with both being handled in the best way, as described in Chapter 4.

2.3.8 DeepAR

DeepAR is a forecasting method developed by Amazon in 2019 [45] that uses autoregressive recurrent neural networks (RNN), which learn from all the time series that are available in a specific data domain, instead of the classical algorithms that learn from just one time series at a time.

This is precisely one of the main advantages of DeepAR compared to other models. Despite the known differences in magnitudes of related time series, DeepAR can fit them in a single global module.

Moreover, DeepAR has another set of advantages:

- Once the model captures the seasonality in the time series and the dependencies between covariates, the amount of feature engineering needed decreases.
- The model produces probabilistic forecasts through Monte Carlo samples. These can thus be used for the computation of consistent quantile estimates.
- Since the model learns from similar time series, it can generate predictions for new time series that are similar to the ones that the model was trained with but not necessarily equal, which means that they can have little to no historical data associated.
- DeepAR does not assume Gaussian Noise but is compatible with numerous alternatives. This makes it easier for the developer to select the one that fits better the statistical characteristics of the data.

The goal of DeepAR is to model the following conditional distribution.

$$P(z_{i,t_0:T} | z_{i,1:t_0-1}, x_{i,1:T})$$

In the above distribution, $z_{i,t_0:T} = [z_{i,t_0}, z_{i,t_0+1}, \dots, z_{i,T}]$ represents the future time series, where $z_{i,t}$ is the value of a time series i at time t and t_0 is the time point from which $z_{i,t}$ is unknown during prediction; $z_{i,1:t_0-1} = [z_{i,1}, \dots, z_{i,t_0-2}, z_{i,t_0-1}]$ represents the past time series; and $x_{i,1:T}$ is the list of covariates that are assumed to be known for all the time points.

It is also assumed that the model's distribution ($Q_{\Theta}(z_{i,t_0:T} | z_{i,1:t_0-1}, x_{i,1:T})$) consists of the product of likelihood factors, parametrized by the output ($h_{i,t}$) of an autoregressive recurrent neural network.

$$Q_{\Theta}(z_{i,t_0:T} | z_{i,1:t_0-1}, x_{i,1:T}) = \prod_{t=t_0}^T Q_{\Theta}(z_{i,t} | z_{i,1:t-1}, x_{i,1:T}) = \prod_{t=t_0}^T l(z_{i,t} | \Phi(h_{i,t}, \Theta))$$

In the model's distribution formula, $l(z_{i,t} | \Phi(h_{i,t}, \Theta))$, l stands for the likelihood and is a fixed distribution whose parameters rely on a function $(\Phi(h_{i,t}, \Theta))$ that depends itself on the network's output $(h_{i,t})$ and on the parameters of the model (Θ) .

The likelihood function $(l(z|\phi))$ should match the statistical properties of the data, determining the "noise model". The parameters (ϕ) , like the mean and variance, of the probability distribution for the next time point are predicted by the nature of the network. The choice of the likelihood function is up to the developer, depending on the specificity of the data, but one thing to have in mind is that the "samples from the distribution can cheaply be obtained, and the log-likelihood and its gradients wrt. the parameters can be evaluated" [45].

$$h_{i,t} = k(h_{i,t-1}, z_{i,t-1}, x_{i,t}, \Theta)$$

In the output equation, k is a function implemented by a multi-layer recurrent neural network with LSTM cells.

The model is autoregressive since the input of the current observation considers the last time step $(z_{i,t-1})$ and is recurrent since the previous output of the network $(h_{i,t-1})$ is used as input in the following time step.

All the past time series information is passed to the future time series by the initial state, i.e., h_{i,t_0-1} , which is the output of an encoder network.

With a data set of multiple time series and the corresponding covariates, the parameters (θ) of the model can be obtained by maximizing the log-likelihood (ι) .

$$\iota = \sum i = 1^N \sum t = t_0^T \log \iota(z_{i,t} | \phi(h_{i,t}))$$

In the above equation, N is the number of available time series, T is the number of time steps, and the remaining symbols have the same meaning as in the previous equations. Another thing to have in mind is that since $h_{i,t}$ is a deterministic function of the input, all the needed quantities for the calculations are available and do not need to be inferred. It is also possible to use a stochastic gradient descent to optimize it.

Having the model parameters (Θ) , joint samples $(\tilde{z}_{i,t_0:T} Q_{\Theta}(z_{i,t_0:T} | z_{i,1:t_0-1}, x_{i,1:T}))$ can be obtained through ancestral sampling. To do so, the first step is to obtain h_{i,t_0-1} by calculating $h_{i,t}$ for t from 1 to t_0 . Next, for t from t_0 to T , $\tilde{z}_{i,t} k(\cdot | \phi(\tilde{h}_{i,t}, \theta))$ is sampled. Here, $\tilde{h}_{i,t} = k(h_{i,t-1}, \tilde{z}_{i,t-1}, x_{i,t}, \Theta)$ and $\tilde{h}_{i,t_0-1} = h_{i,t_0-1}$ and $\tilde{z}_{i,t_0-1} = z_{i,t_0-1}$. These samples are useful to compute quantiles, for example.

Figure 2.6 summarizes the architecture of DeepAR, both for the training and prediction stages.

In the paper "A Temperature Time Series Forecasting Model Based on DeepAR" by the authors Yanping Lao and Chao Liang [29], the main goal is to forecast the weather for the next 24 hours with limited numerical data and with the usage of a simplified model.

The model that was built is based on the DeepAR presented before, with some changes. While DeepAR mainly uses LSTM layers to extract temporal features, the authors decided to stack those

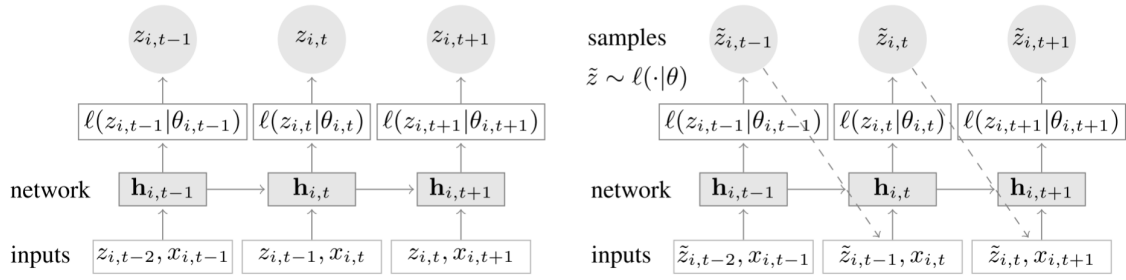


Figure 2.6: DeepAR architecture for training (left) and prediction (right) stages.

layers and set multiple cells in each layer. Then, an LSTM layer processes input sequences one by one at each time step, which is not efficient in terms of the training time. To accelerate the process, the authors went for the usage of stacked temporal convolutional layers and LSTM layers for both the encoder and decoder.

Another change made to the model was aimed at making it possible to be trained either with a single-variable input or multi-variable input. For that, the second dimension of the convolution kernel was adjusted. This adjustment consisted of the usage of LSTM layers to extract high-level time features, taking as input the output of convolutional layers with which they were stacked.

In terms of data preparation, the authors decided to estimate the missing values using a linear function in case they were within a time window of 3 hours. For missing values in a time window higher than 6 hours, the corresponding day was discarded.

The main conclusion is that the developed model and DeepAR perform better than traditional models, with the developed model achieving better results than DeepAR.

2.3.9 Prophet

Prophet is a forecasting model developer by Facebook with implementations in Python and R. It was first introduced in 2017 in the paper "Forecasting at scale" [67].

The motivation for the Prophet's creation relies on the complexity of existing time series forecasting models and has as its main goal the democratization and ease of usage of a general time series forecasting model that can be used in a huge variety of projects. It is also intended that Prophet can be used by people with little knowledge of time series forecasting, which is, most of the time, a problem in such kinds of projects due to the lower level of skilled people in the area in a business context.

Prophet is a decomposable time series model with three main components: trend, seasonality, and holidays. The general equation for Prophet is the following.

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t$$

In the above equation, $g(t)$ is the trend function that models non-periodic changes in the time series values. $s(t)$ represents the seasonality, and $h(t)$ the effects of holidays, which might be

as diverse as one can imagine. ε_t is the error of the model, which is assumed to be normally distributed.

The $g(t)$ component has two implementations, one non-linear (saturating growth) and another linear (piecewise).

Starting with the non-linear implementation, attending to Facebook's business, the growth modeling refers most of the time to the evolution of the number of users. This growth is modeled using the following logistic growth model.

$$g(t) = \frac{C}{1 + \exp(-k(t-m))}$$

In the above equation, C is the carrying capacity (e.g., for Facebook, it is the number of people with Internet access), k is the growth rate, and m is an offset parameter.

Since C is not a constant, as the carrying capacity may change over time, there is the need to map its value to a function that is time-dependent, $C(t)$. The same happens for the growth rate (k), since it depends on a huge variety of factors that make it not constant over time.

To model the changes in the growth rate, consider s_j as the point in which the growth rate changes, where j can take values from 1 to the number of change points. Then, consider δ as the vector of rate adjustments, where δ_j is the rate change that occurs in point s_j .

With that, the rate at time t is the base rate k plus all the rate changes until that point, which traduces in the following expression.

$$k + \sum_{j:t > s_j} \delta_j$$

A cleaner representation of it can be achieved with a vector $a(t)$ defined as follows.

$$a_j(t) = \begin{cases} 1, & \text{if } t \geq s_j \\ 0, & \text{otherwise.} \end{cases}$$

Substituting the above vector in the expression above, the rate at time t is

$$k + a(t)^T \delta$$

The offset parameter, m , must also be adjusted with the rate change. The adjustment at the point of change j can be computed with the following formula.

$$\gamma_j = (s_j - m - \sum_{l < j} \gamma_l) \left(1 - \frac{k + \sum_{l < j} \delta_l}{k + \sum_{l \leq j} \delta_l}\right)$$

Finally, the logistic growth model equation is the following.

$$g(t) = \frac{C(t)}{1 + \exp(-(k + a(t)^T \delta)(t - (m + a(t)^T \gamma)))}$$

The logistic growth model above is a single type of sigmoid curve but can be extended to other curves without difficulties.

For the linear implementation, the trend model equation is the following.

$$g(t) = (k + a(t)^T \delta)t + (m + a(t)^T \gamma)$$

In the above equation, all the components have the same meaning as for the non-linear implementation, except the γ_j whose value is $-s_j \delta_j$ to make the function continuous.

The $s(t)$ component represents the seasonality and is represented by a standard Fourier series using the following formula.

$$s(t) = \sum_{n=1}^N (a_n \cos(\frac{2\pi nt}{P}) + b_n \sin(\frac{2\pi nt}{P}))$$

In the above formula, P is the regular period of the time series (e.g., $P=365$ for yearly data or $P=7$ for weekly data), and N is an integer higher than 0 and whose value depends on the time series, being calculated, most of the times, with a hyperparameter optimization technique. a_n and b_n are parameters that need to be estimated.

To estimate these parameters (β), a matrix of seasonality vectors $X(t)$ is constructed.

$$X(t) = [\cos(\frac{2\pi 1t}{P}), \dots, \sin(\frac{2\pi Nt}{P})]$$

That said,

$$s(t) = X(t)\beta$$

An increase in the value of N means that more seasonal patterns can be fitted, although this might lead to overfitting of the data to the model. According to Prophet's paper, the values 10 and 3 were the best for N in time series with $P = 365$ and $P = 7$, respectively.

The third and last component is the holiday effect, $h(t)$. Although there are recurrent holidays and their impact on time series forecasting cannot be discarded, it is also difficult to model since most of the holidays are not periodic, with some changing from one year to another.

Prophet supports the addition of custom holidays through a list inputted by the user that can contain general or country-specific holidays. The inclusion of the holidays in the model is made with the assumption that the holidays' effect is independent.

Considering the holiday i , D_i is the list of past and future dates for that holiday. The model adds an indicator function that represents whether time t is during the holiday i . Each holiday is also assigned with a parameter k_i , which represents the change in the forecasting.

On the same direction as with the seasonality, the matrix of regressors for the holidays, $Z(t)$ is the following, where L is the number of holidays.

$$Z(t) = [1(t D_1), \dots, 1(t D_L)]$$

With that, the equation of the holidays is the following.

$$h(t) = Z(t)k$$

Since the effects of holidays are usually visualized also in the surrounding days, Prophet considers a time window of days, where a day is treated as a "holiday."

This model has several advantages compared to other well-known time series forecasting models:

- Flexibility - seasonality can be considered in different periods, giving the opportunity for the developer to take different conclusions about the trends.
- The model does not require the data to be regularly spaced in time or even account for missing values, like ARIMA models.
- It fits fast to the data, which allows the developer to test multiple parameter configurations.
- These parameters can be easily understood by the developer, which eases and democratizes the usage of the model for time series forecasting projects.

Prophet works best with time series that have strong seasonal effects and several seasons of historical data [68].

The work carried out and described in this paper [76], from the authors Yadwad S, Kumari V, Lakshmi S, published in 2021, has two main goals:

- Forecast the count of customer trouble tickets for the coming days using structural time series analysis.
- Use a range of warnings obtained from the historical data several days prior to the failure to formulate a pattern that can effectively predict what happens to the equipment the following day.

The main difference between this approach and this Master Thesis is that these authors use both logs and tickets as data sources, while for the scope of this Master Thesis, only the information about tickets is used as data source. Also, the prediction as a result of the work developed by these authors is limited to the next 9 days, which is significantly lower than the maximum prediction period of this Master Thesis (up to 365 days).

However, there are a few acknowledgments taken from this document. The authors successfully used the Lasso and Ridge regression models, as well as Prophet, which achieved better results than the frequently used ARIMA models.

2.3.10 Summary

In this section, there is a table for each group of models that summarizes their characteristics, advantages, and disadvantages.

Lasso and Ridge Regression, ARIMA, and SARIMAX belong to the Regression Models group. Their main characteristics, advantages, and disadvantages are described in table 2.1.

Model	Characteristics	Advantages	Disadvantages
Lasso	- Based on linear regression - L1 regularization	- High accuracy with simplicity and low number of parameters - Good with high-dimensional data sets - Automatically does feature selection - Avoids data overfitting - Effective against outliers	Lower accuracy and loss of information with data sets with strong correlation in the features
Ridge	- Based on linear regression - L2 regularization	- Good for data sets with multicollinearity - Avoids data overfitting - Reduced complexity - Effective against outliers	- Biased coefficients - Computationally expensive for large data sets - Hard to optimize and to interpret the results
ARIMA	- Generalization of the ARMA model - Considers previous values and previous errors	Applies differencing that makes any time series a stationary one	- Computationally expensive - Does not have a clear methodology to determine the values of P, D, and Q
SARIMAX	Extension of ARIMA to include seasonality and exogenous variables	- Captures the seasonality in a data set - Allows the time series to include external information (covariates)	- Same as for ARIMA - Does not have a clear methodology to determine the values of P', D', and Q'

Table 2.1: Summary of the Regression Models.

Model	Characteristics	Advantages	Disadvantages
ANN	• Networks of nodes with weights and thresholds • Basis for Deep Neural Networks	• Captures sequential relationship in a Time Series (RNN) • Good performance with media (CNN) • Reduced complexity (CNN) • Improved efficiency (CNN) • Over-fitting risk limited (CNN)	• Backpropagation can lead to a small gradient or impossible training (RNN) • Short-term memory (RNN) • Loss of information (CNN) • Does not have a clear methodology to determine the number of nodes and layers
LSTM	• Based on RNN • Fixes the RNN main problem with long-term memory	• Solves the short-term memory of RNN • Captures long-term relationships	• More complex than RNN • Low performance with data sets where data is not sequential
LSTMX	• Extension of LSTM to include exogenous variables	• Allows the time series to include external information (covariates)	• Same as for LSTM • Does not have a clear methodology where to include the exogenous variables in the network

Table 2.2: Summary of the Neural Networks models.

Then, ANN, LSTM, and LSTMX are in the Neural Networks group. Their main characteristics, advantages, and disadvantages are described in table 2.2.

Lastly, DeepAR and Prophet are in the Advanced Models group. Their main characteristics, advantages, and disadvantages are described in table 2.3.

2.3.11 Other Techniques

The paper "Improving IT Support by Enhancing Incident Management Process with Multi-modal Analysis" [33], published in 2019 by the authors A. Mandal, S. Agarwal, N. Malhotra, G.

Sridhara, A. Ray and D. Swarup, has as a unique goal the demonstration of the advantages and process of using multi-modal ticket analysis rather than text-based analysis only.

The main difference is the fact the data source used in this approach is composed not only of text but also of other data formats, such as documents and images.

Although, at the time being, the ticket data used in this Master Thesis does not include any format other than text, the procedure described in this paper to extract text from screenshots and enrich the ticket analysis might be interesting and useful in the future.

Another technique that is particularly interesting is the one pointed out in the paper "Ticket price forecasting based on machine learning" [28]. This document, published in 2017 by the authors Y. Li, Z. Li, and S. Qin, presents the usage of a combined model that predicts air ticket prices for nine cities in China.

To predict the price of tickets, the authors use a time series algorithm, while a random forest algorithm is used to predict the residual error. The error of the combined model is significantly smaller than the one of the individual models. However, as pointed out by the authors, the running time is relatively long.

The main acknowledgment is the combination of two models, one of which not being a time series algorithm, to predict the price of air tickets.

2.4 Data Preparation in Time Series Forecasting

In this section, an analysis is made of the selected documents that represent an important step in the life cycle of a project like this Master Thesis, which is the Data Preparation. Each section summarizes each of the analyzed documents, as well as the extracted acknowledgments, which are properly described.

2.4.1 Machine learning supervised analysis for enhancing incident management process

In this document [39], published in 2020 by the authors A. Mustapha, S.A. Mostafa, M.H. Hassan, M.A. Jubair, S.H. Khaleefah, and M.H. Hassan, the goals rely on using machine learning algorithms to decide which are the best features to use in the prediction related to incident management.

The ML algorithms used were Trees Decision Stump, Bayes Net, Naive Bayes, Rules One R, and Rules Zero R.

The main difference between this approach and this Master Thesis is similar to one of the differences noted in some of the analyzed documents: the authors use logs as data sources instead of the ticket information.

There is, however, a common topic with this Master Thesis, which is the desire to select the relevant features among a set of various available.

The main acknowledgment of this document was that the best-performing algorithm for the above-mentioned task was Bayes Net.

2.4.2 Effective feature pre-processing for time series forecasting

In this document [77], published in 2006 by the authors J.H. Zhao, Z. Dong, and Z. Xu, it is shown the importance of feature pre-processing as part of a time series forecasting project.

The data set used is a highly volatile one and contains the electricity prices in Australia.

The simplest form of SVM, Linear Regression, using Vapnik's ϵ -insensitive loss function, is used to predict the upcoming prices of electricity in Australia, while the MAPE is the selected measure to assess both the prediction accuracy and the feature set quality.

For this specific data set and model, the best-performing combination among the 15 techniques was the wrapper model with genetic search.

The main conclusion of this paper is that "(...) including irrelevant features in the forecasting model can significantly decrease the forecasting accuracy."

The main acknowledgment is not the best-performing technique but rather the importance of doing a properly structured and cautious data pre-processing.

Model	Characteristics	Advantages	Disadvantages
DeepAR	Amazon model that uses RNN	- Learns from multiple time series - Once seasonality and dependencies are captured, feature engineering needs decrease - Produces probabilistic forecasts (Monte Carlo samples) that can be used to compute consistent quantile estimates - Can be trained and used with different (but still similar) time series - Does not assume Gaussian Noise: allows the selection of the one that fits better the statistical characteristics	- Does not have a clear methodology to decide on the likelihood function - Complex to understand - Requires a huge amount of data to be trained with - Loses performance with the increase of the prediction interval
Prophet	Meta model to democratize and ease the usage of a time series forecasting model	- Trend variations, seasonality, and holidays (with time window) - Flexibility (seasonality considered in different periods) - Data points do not need to be equally spaced in time - No need to deal with missing values - Fits fast to the data	- Does not have a clear methodology to calculate N - Performance decreases with time series with little to no seasonality effect and without several seasons of historical data

Table 2.3: Summary of the Advanced Models.

Chapter 3

Scope and Preliminary Work

This Master Thesis is essentially divided into two parts, with the second based on the first one, both contributing to the main goal, which is to support the Infineon team that handles the tickets in their resolution, i.e., on the preparation of the necessary manpower and software/hardware resources.

The first part is related to the creation of an AI model capable of outputting two kinds of time series predictions, one for the volume of tickets (incidents and tasks) and another one for the volume of tickets per type (only applicable to tasks, since incidents do not have an associated type). This latter one is especially important for project rollouts.

The time series predictions are aimed for the next 7, 30, 60, 90, or 365 days. For the specific case of 365 days, it should be possible to recognize patterns and trends that might occur by analyzing the produced graphic. For example, the end of the fiscal year and the holiday season are two complete opposites in terms of the expected volume of tickets, and both represent a clear pattern and trend amongst the volume of tickets during a year.

The training of the AI model should be done on a regular basis, with the frequency still being discussed internally at Infineon, to make sure the accuracy is high and the model is up-to-date. The training is based on historical tickets, meaning that the volume of data to use on the training will increase over time.

The second part of this work is related to the creation of a dashboard/visual interface to display the results obtained from the built model.

The dashboard is composed of two main parts, with each one referring to one of the components of the AI model, i.e., one for the prediction of the volume of tickets and another for the prediction of the volume of tickets and its distribution per type.

3.1 Technology Decision Matrix

Considering the previously described solution and the deliverables for this Master Thesis, a working plan, a communication plan, and a risk management matrix were elaborated. These plans

and matrix are described in detail in the appendix A, while the Technology Decision Matrix, which lead to the selection of the tools to be used, is described next.

The technology decision matrix process started with the definition of the topics in which each technology was going to be evaluated.

Then, for each topic, a set of questions was raised, each one with an associated weight (from 1 to 10), depending on the relevance of the decision process, and each technology received a score (from 1 to 5). The technologies were divided into three groups: one for the database tools, another for the AI model, and another one for the dashboard.

The topics and the questions, with the corresponding weight (inside brackets), were the following:

- Accessibility
 - Is the access to the tool open to everyone? [10]
 - How easy is it to obtain access to the tool? [7]
 - Is there any license/cost to use the tool? [8]
 - Are there any limitations in terms of usability? [8]
- Security
 - Is the tool recommended to deal with/store the type of data used in this project? [10]
 - Is there any security guideline available to be used in this tool? [9]
- Ease of use
 - Is the tool easy to use / intuitive? [7]
 - Is there training available to get familiar with the tool? [6]
- Scalability
 - Is the tool scalable in terms of the amount of data? [10]
 - Does the performance decrease with the amount of data? [9]
- Support
 - Is there a team or a responsible for the tool at Infineon? [6]
 - What kind of support is available? [6]
- Maturity
 - Are there concrete examples of implementation/use cases at Infineon? [5]
 - Are there known failures/limitations of the tool? [10]
- Documentation

- Is there enough documentation available regarding the usage of the tool? [8]
- Does the tool offer the possibility to document what is developed? [7]
- Linkage with other tools
 - What is the level of support for the integration with other technologies? [10]

The final score for each technology was obtained by summing the multiplications of the score in each question by the corresponding weight.

The average score by topic for each group and technology can be seen in the figures 3.1, 3.2, 3.3.

Oracle SQL was the selected tool for the database, Jupyter Notebook (HICP) was the selected tool for the AI-related tasks, and SAPUI5 was the selected tool to build the dashboard.

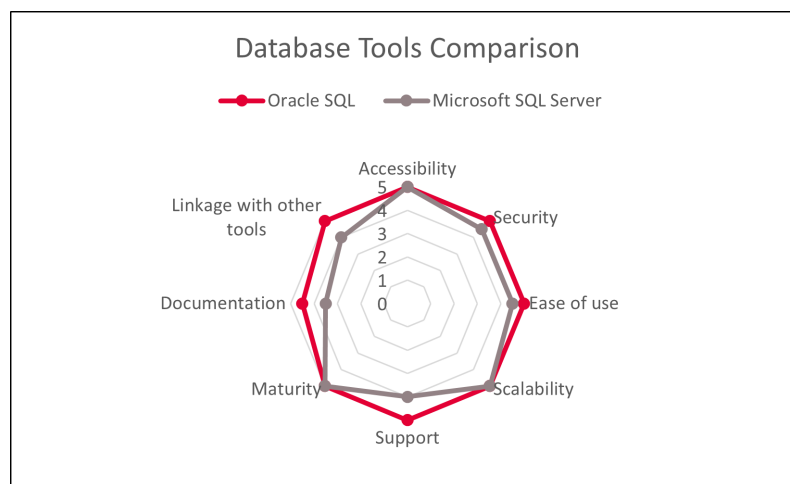


Figure 3.1: Average score by topic and technology for the Database.

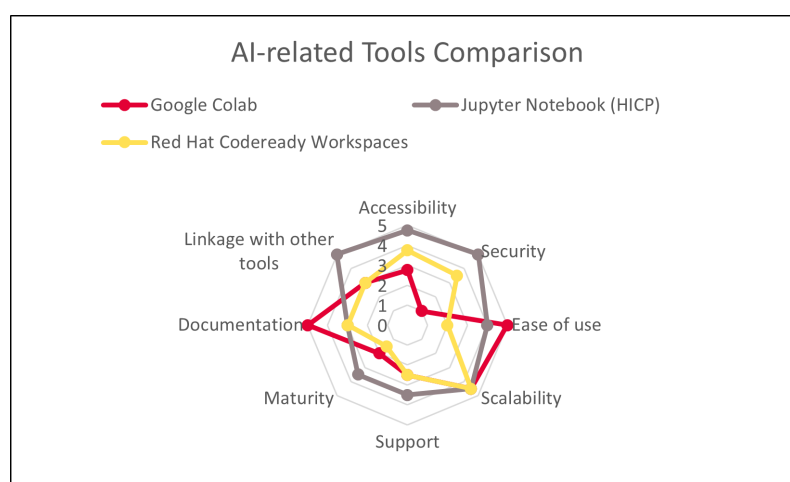


Figure 3.2: Average score by topic and technology for the AI.

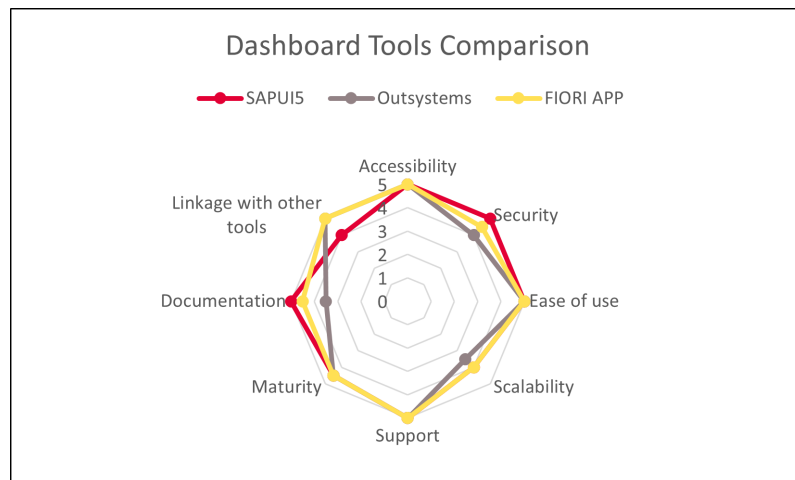


Figure 3.3: Average score by topic and technology for the Dashboard.

Chapter 4

Data

In this chapter, the data understanding and data preparation processes are described in detail, in what constitutes one of the most important parts of a Time Series Forecasting project like the one from this Master Thesis.

Also in this chapter, the last section presents the Data Visualization component, which was built to show the predictions obtained.

4.1 Data Understanding

The data understanding process is the first to be made on a Machine Learning project. For this Master Thesis, the data is divided into two data sets, both referring to tickets.

One of the data sets is constituted with tasks, which are a type of tickets that represent requests made by Infineon employees. Those requests can be related to the need to be granted a role to access an SAP system, a change in an SAP system that can only be performed by a group of users, or the creation of an account for the requester to be able to access some SAP system, to mention a few cases.

The other data set refers to incidents, which are also a type of tickets that group all the problems and errors the Infineon employees face, like an authorization error, a system that is failing upon a specific request, or any other error that happens in an SAP system, to mention some of the possible cases.

Due to data privacy concerns, there is no possibility of showcasing any data example, neither for an incident nor for a task.

Both data sets are stored in individual databases, which were available for read-only access during the development of this Master Thesis.

The data was collected and converted into a Python data frame, one for the tasks and another for the incidents.

The information that follows is regarding the available data until the 31st of July of 2023. One must keep in mind that the data increases in volume every day, with new tickets being submitted,

thus the results and conclusions obtained could be different if the data used was from another time period.

For the incidents, a total of 33,613 tickets were available, with a dimensionality of 71, which includes all tickets' characteristics. For the tasks, a total of 91,443 tickets were available with a dimensionality of 38.

After the data collection was completed, the next step was to describe and explore the data, assessing its quality. For that, a couple of graphics were produced in order to better visualize the data evolution and distribution.

4.1.1 Incidents

To start the data understanding process relative to the incidents, an analysis of the available features must be made in order to assess what kind of information is available. To ease the analysis, the related features were grouped by their semantics, obtaining the following four groups:

- **IDs and other numerical characteristics:** *SUPPORT GROUP ID, TICKET ID, PROTOCOL ID, PROTOCOL ID WC, DAY ID CLOSED DATE, DAY ID LAST RESOLVED*. These features denote IDs that are assigned during the submission of the incident by the requester.
- **Time-related characteristics:** *CLOSED DATE, DURATION, END-TO-END RESOLUTION TIME, LAST RESOLVED TIME, TICKET AGING, LAST MODIFIED DATE, TICKET CREATE TIME, SOLVING RATE (24 HOURS), SOLVING RATE (4 DAYS), FIRST STATUS START TIME, FIRST CLOSED START TIME, COMPLETION DATE, FIRST INVESTIGATION START TIME*. These features are automatically created upon the change in the status of the incident, e.g., the *END-TO-END RESOLUTION TIME* is set after the incident is marked as completed and represents the time it took since the submission to the resolution of the incident.
- **Personal features:** *CUSTOMER DEPARTMENT, COMPANY, CUSTOMER, ORGANIZATION, DIRECT CONTACT COMPANY, DIRECT CONTACT OFFICE, CUSTOMER COMPANY, CUSTOMER OFFICE, CREATED BY (NAME), CREATED BY (DEPARTMENT), CREATED BY (SITE), CREATED BY (REGION), CUSTOMER FIRST NAME, CUSTOMER LAST NAME, DIRECT CONTACT LAST NAME, DIRECT CONTACT FIRST NAME, DIRECT CONTACT DEPARTMENT, IT LINE, REGION, SITE, PERSON DEPARTMENT, OWNER GROUP, RESOLVED ASSIGNEE, CUSTOMER REQUESTER SITE, CREATED BY, SERVICE IT LINE, OWNER*. These features are related to the requester and assignee characteristics, including personal information such as the name and contacts, and professional information such as the department and location.
- **Incidents intrinsic characteristics:** *IS WEB TICKET, CREATED VIA XML GATEWAY, MODEL VERSION, IMPACT, NOTES, OPERATION CATEGORIZATION TIER 1, OPERATION CATEGORIZATION TIER 2, OPERATION CATEGORIZATION TIER 3, PRIORITY,*

PRODUCT CATEGORIZATION TIER 1, PRODUCT CATEGORIZATION TIER 2, PRODUCT CATEGORIZATION TIER 3, PRODUCT NAME, SOURCE, SUMMARY, RESOLUTION METHOD, TICKET TYPE, URGENCY, IS OPEN, CLOSING STATUS, STATUS, BASIC DATA, FIRST LEVEL CREATED, LAST PRIORITY, FIRST STATUS, FIRST DRAFT TO REVIEW, FIRST LEVEL. These features characterize each incident on a set of attributes that differ incidents from each other. Most of these features are related to categorizations of the incidents and metadata generated upon their submission, as well as free-text fields.

Next, an analysis was made on the number of incidents over time and also on the distribution of incidents per source and per year.

The first graphic, 4.1, shows the distribution of incidents per year. With a quick analysis of the graphic, it is possible to conclude that the number of incidents is only relevant since 2018, as before (in 2016 and 2017), it is residual and not significant. The main conclusion that can be taken is that the submission of incidents only became a recurrent behavior from Infineon employees since 2018. Before that, the insignificant number of submitted incidents denotes that either the submission system or the process was different.

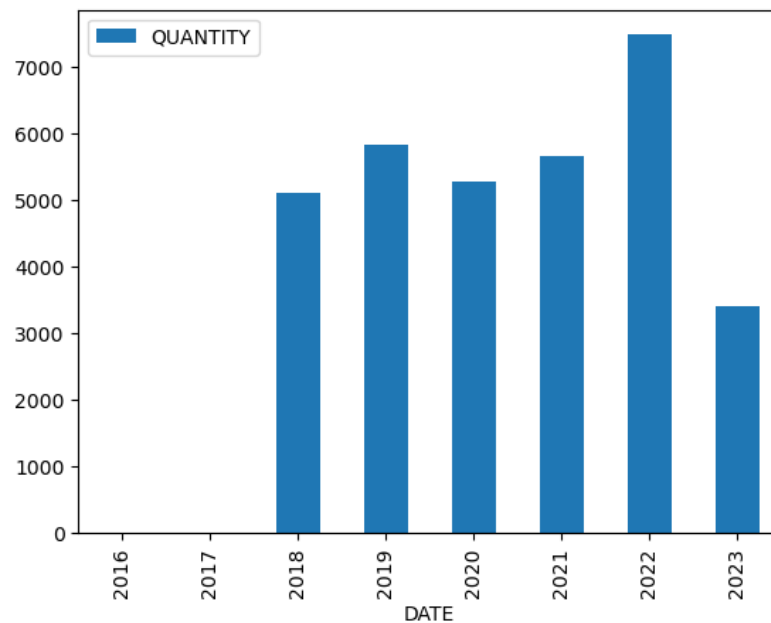


Figure 4.1: Incidents distribution per year.

Then, the following three graphics, 4.2, 4.3, and 4.4, show the evolution of the number of incidents in the last three completed years. As it is possible to observe from a quick analysis of the graphics, there are some recurrent peaks at the beginning of the year, on Easter, on Summer holidays, and by the end of the year.

The last graphic, 4.5, shows the distribution of incidents per source. Due to data privacy concerns, the sources were masked into a numerical representation. As it is possible to see, there are

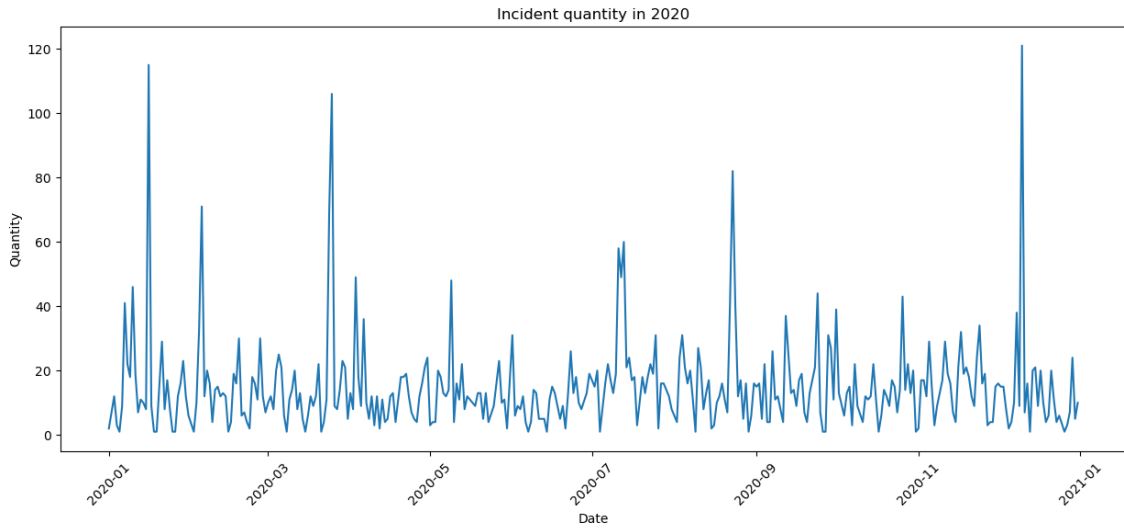


Figure 4.2: Incidents distribution in 2020.

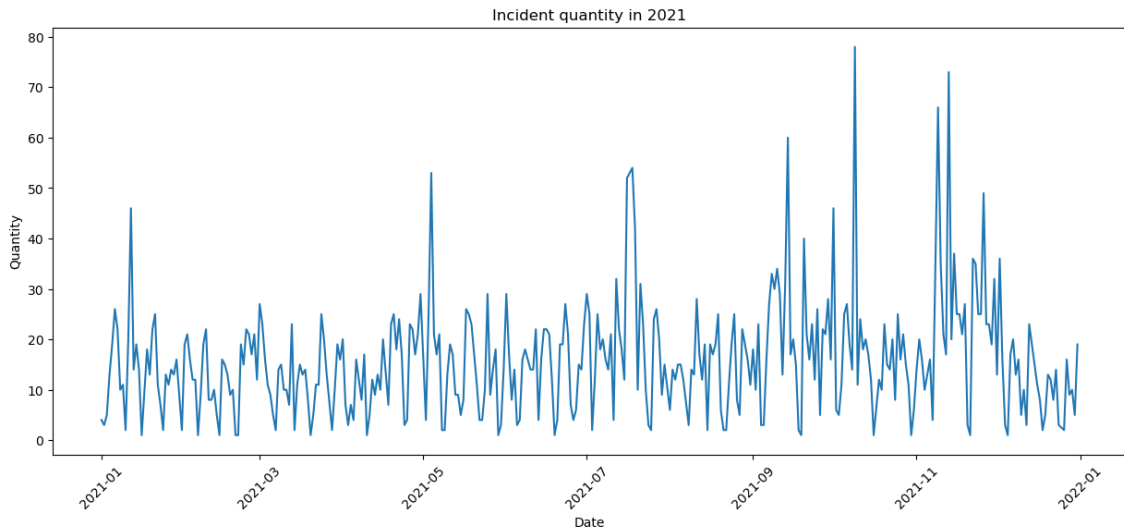


Figure 4.3: Incidents distribution in 2021.

a few sources with higher representativity (3, 6, and 7), while the others have a residual representation.

4.1.2 Tasks

The features of the data set with the tasks can be grouped according to a semantics evaluation similar to the one made for the incidents, resulting in a grouping in the same number and kind as the one used before.

- **IDs and other numerical characteristics:** *DAY ID, TICKET ID, TASK ID, INSTANCE NUMBER, SEQUENCE, PROTOCOL ID, PROTOCOL ID WC, SUPPORT GROUP ID,*

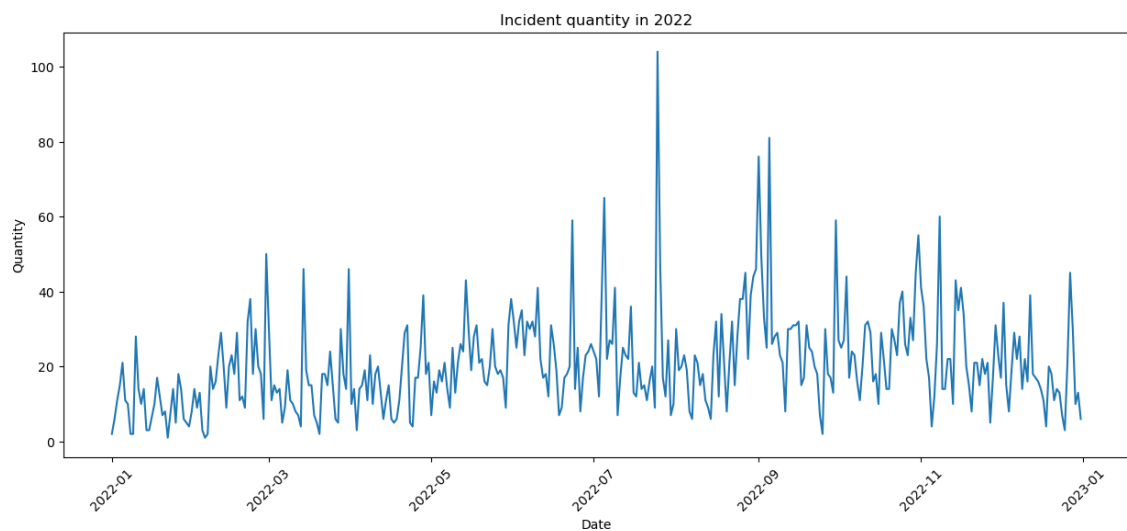


Figure 4.4: Incidents distribution in 2022.

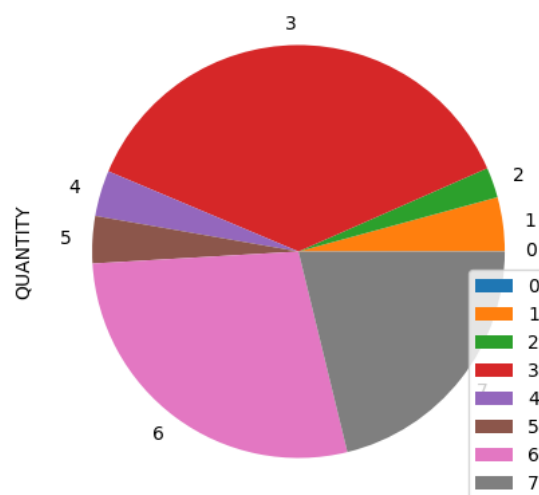


Figure 4.5: Incidents distribution per source.

ASSIGNEE LOGIN ID. Again, these features represent unique IDs that are assigned automatically upon the submission of a ticket by the requester.

- **Time-related characteristics:** *ACTIVATE TIME*, *CLOSED DATE*, *END-TO-END RESOLUTION TIME*, *CREATE DATE*, *SCHEDULED END DATE*. These features denote time characteristics that are automatically generated during the different statuses that a ticket has, from its submission until its resolution.
- **Personal features:** *RESOLVED BY*, *ASSIGNEE DEPARTMENT*, *ASSIGNEE REGION*, *ASSIGNEE SITE*, *CREATED BY*, *SITE*, *REGION*, *IT LINE*, *ASSIGNEE*. As for the incidents, these features are related to the characteristics of the requester and assignee, including the name and professional information about both, like the department or location.

- **Tasks intrinsic characteristics:** *TICKET TYPE, PRIORITY, STATUS (duplicated), STATUS REASON, SUMMARY, PRODUCT NAME, TASK NAME, IS OPEN, OPERATION CATEGORIZATION TIER 1, OPERATION CATEGORIZATION TIER 2, OPERATION CATEGORIZATION TIER 3, PRODUCT CATEGORIZATION TIER 1, PRODUCT CATEGORIZATION TIER 2, PRODUCT CATEGORIZATION TIER 3*. These features are metadata and other information that is either provided by the requester using a set of predefined options or free-text fields or generated automatically by the ticketing system.

Next, an analysis was made regarding the evolution and distribution of certain task characteristics over time.

The first graphic, 4.6, shows the distribution of the tasks per year. As one can see by analyzing the graphic, only from 2018 onwards, like for the incidents, the number of tasks surpassed 6000. The year 2023 was still ongoing by the time of the analysis, which is the reason why it still had less than 4,000 tasks submitted. The main conclusion that can be taken is the same as the one taken for the incidents regarding the usage of the actual ticketing system.

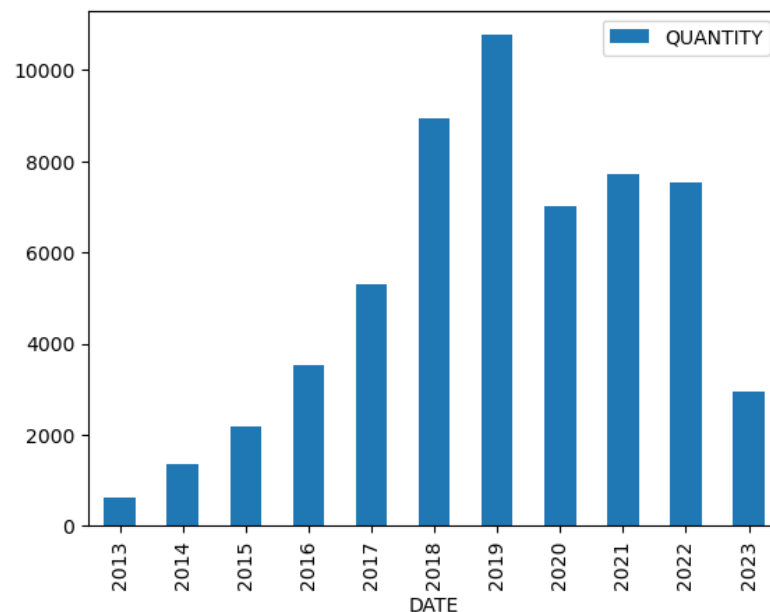


Figure 4.6: Tasks distribution per year.

The following graphics, 4.7, 4.8, and 4.9, show the distribution of tasks in the last three completed years. As it is possible to recognize after analyzing the graphics, there is no clear pattern in the years and from year to year. There are some random peaks and no constant periods in which the number of tasks remains the same.

The last graphic, 4.10, shows the distribution of the tasks per type in all years. Due to data privacy concerns, the task types were masked into a numerical representation. As it is possible to recognize from the graphic, there are no dominant types in terms of representativity, being the types' representation mostly balanced, with some types under-represented.

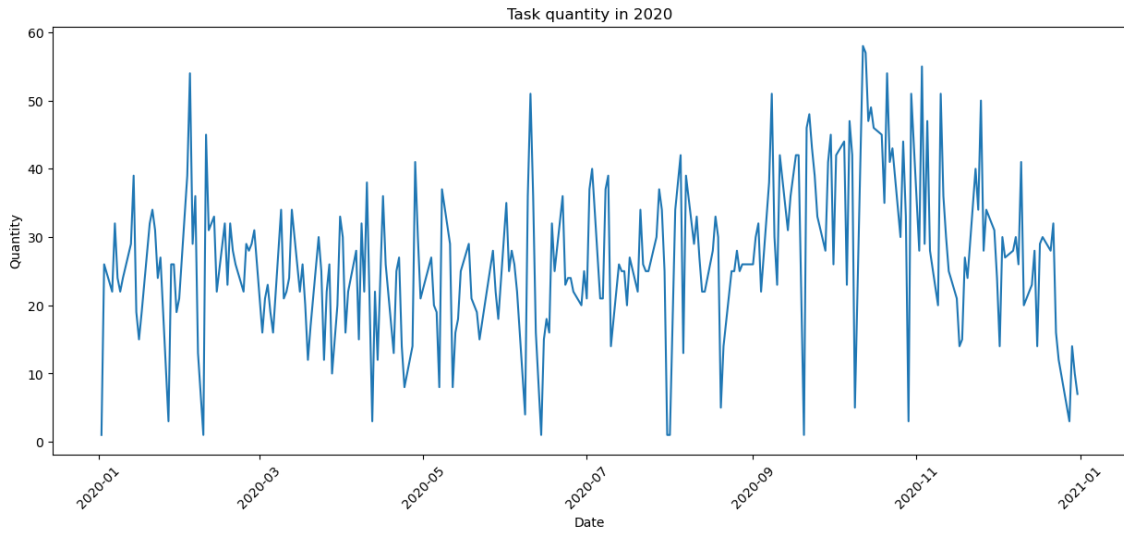


Figure 4.7: Tasks distribution in 2020.

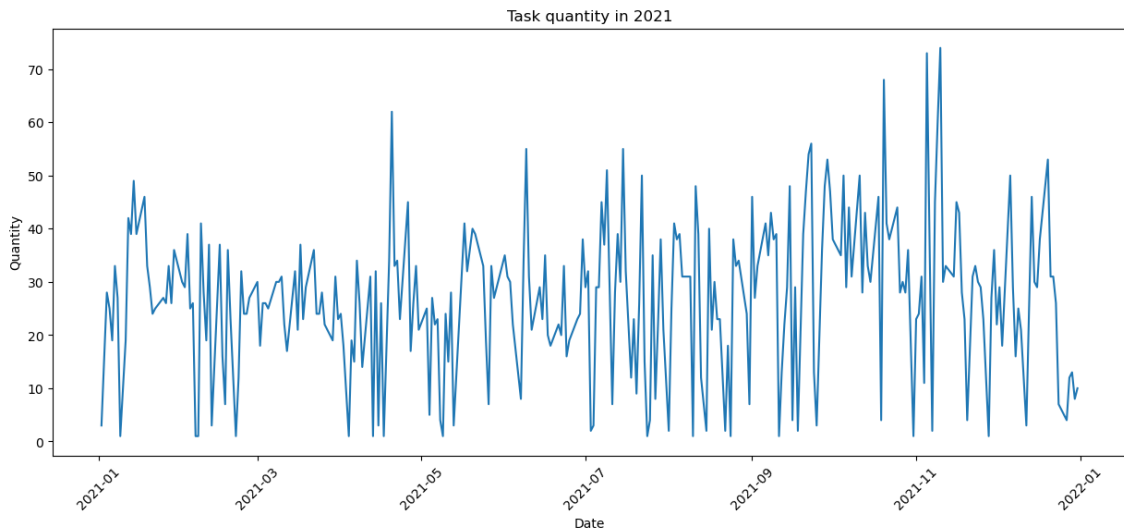


Figure 4.8: Tasks distribution in 2021.

4.2 Data Preparation

According to the paper "Effective feature pre-processing for time series forecasting" [77], from the authorship of J.H. Zhao, Z. Dong, and Z. Xu, "feature pre-processing [data preparation] is an important part of time series forecasting."

The importance of this process resides in the fact that the accuracy of the models will be, in a way, as high as good as the data they are trained with is.

To obtain a good data set from the original data described in the previous section, some transformations were performed, namely the selection of the data to be used from the one available, the data cleaning stage, the construction of data to extend the original one, the integration of data and its formatting, which was the last step before proceeding to the modeling.

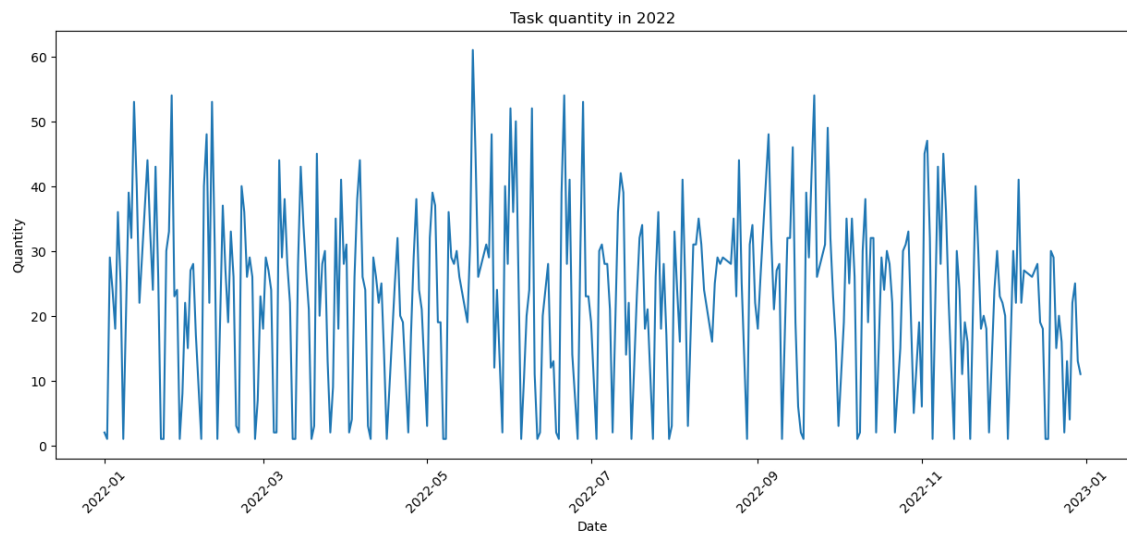


Figure 4.9: Tasks distribution in 2022.

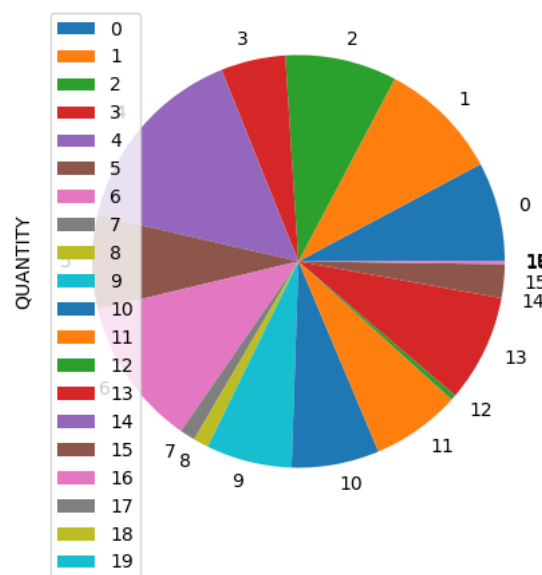


Figure 4.10: Tasks distribution per type.

In the next sections, each one of these transformations is described in detail, both for the incidents and for the tasks.

4.2.1 Data selection and cleaning

On the data selection and cleaning, starting with the incidents, the first step was to drop all the columns that contained either personal or company information. The *SITE* feature was the only one kept, as an exception, since it was used later on to obtain information regarding the public holidays.

Then, all the columns with IDs that are unique and not relevant were removed, followed by the removal of semantically irrelevant features. Semantically irrelevant features are all those that do not relate to the subject of this Master Thesis.

The remaining columns were analyzed in detail to correct missing or incorrect format problems. Starting with the *MODEL VERSION*, *RESOLUTION METHOD*, and *TICKET AGING*, these features have an average of 99% of missing values. Thus, the corresponding columns were removed from the data set.

The features *LAST RESOLVED TIME*, *COMPLETION DATE*, *LAST MODIFIED DATE*, and *CLOSED DATE* contain the same values, semantically speaking. After an analysis of the number of missing values, the column *CLOSED DATE* was the one kept, due to an almost close to 0% of missing values, removing the other ones from the data set.

In the same direction, the *END-TO-END RESOLUTION TIME* and *DURATION* features represent the same information needs. The *END-TO-END RESOLUTION TIME* was removed since it contains a significant number of missing values (8%) compared to the *DURATION* (0%).

With the *TICKET CREATE TIME* and *DURATION*, it is possible to obtain the *CLOSED DATE*, meaning that the latter could be removed.

The analysis of the incidents assumes that only the incidents that were successfully closed are relevant. To achieve this, the incidents were filtered on the *IS OPEN* feature, only keeping the ones with a value equal to 'NO'.

The same process was applied to the *CLOSING STATUS* feature, filtering the entire data set by the value 'CLOSED' in this column and dropping the feature *STATUS* afterward, since its value is the same for all the closed incidents.

For the *OPERATION CATEGORIZATION TIER 1*, *OPERATION CATEGORIZATION TIER 2*, and *OPERATION CATEGORIZATION TIER 3* features, since the average of missing values was higher than 90%, the three columns were dropped from the original data set.

After a manual analysis of a representative number of incidents, the *SUMMARY* and *NOTES* features were removed because both contain unstructured and diverse data, including confidential information such as access credentials and other systems' critical information. The treatment of these two features could be made in another project, out of the scope of this Master Thesis, or as an extension of it, using Natural Language Processing to extract only useful information from it.

The last step for the incidents regarding data selection and cleaning was to remove all the incidents earlier than 2018 from the data set due to their lower representativity.

For the tasks, a similar procedure to one of the incidents was made. First, the columns with unique and irrelevant IDs as well as the ones containing personal information and the ones semantically irrelevant to the scope of this Master Thesis were removed.

Again, the *SITE* was the only feature to be kept to be able to obtain public holiday information later on.

The data set was also filtered on the *STATUS* of each task, with all the entries with a status different from 'CLOSED' to be discarded. The *STATUS REASON* was also used to filter all the

tasks with a value different from 'SUCCESS', removing those. The reason behind this is the same as the one presented for the incidents.

With the *END-TO-END RESOLUTION TIME* and *CREATE DATE* features, it is possible to obtain the *CLOSED DATE* one. However, it was the *END-TO-END RESOLUTION* feature to be discarded due to its high number of missing values. The same happened for the *ACTIVATE TIME* and *CREATE DATE* features, which are semantically identical, with the first one being removed due to the significant number of missing values when compared to the latter. At a later stage, the *CLOSED DATE* was also removed since the time information it adds to each data point is not relevant to the scope of the project.

Since the *SUMMARY* and *TASK NAME* features contain important information, these could not be discarded in an initial stage. However, after a comparison between both, the values each one had were similar. Thus, only the *SUMMARY* was kept. Later on, for the same reason mentioned on the removal of the *SUMMARY* and *NOTES* features for the incidents, the *SUMMARY* feature for the tasks was also removed.

All the tasks prior to 2018 were discarded due to their low representativity.

4.2.2 Data construction

For the *PRODUCT CATEGORIZATION TIER 1*, *PRODUCT CATEGORIZATION TIER 2*, and *PRODUCT CATEGORIZATION TIER 3* of the incidents data set, the missing values problem was overcome by concatenating the values of each feature into a single feature (*PRODUCT CATEGORIZATION TIER*), dropping then the original columns from the incidents data set. This column was then converted from its nominal representation to a discrete numerical one.

The *SITE* feature was used on the same data set to obtain the coordinates (latitude and longitude) with the GeoPy Python client. Then, with the coordinates, the addresses and the country codes were obtained, for each entry, getting lastly the information regarding whether or not it was a public holiday on the days and locations of the submitted incidents.

Moreover, a new feature was created for each incident to denote the day of the week on which it was submitted. This feature contains values from 1 to 7, where 1 represents Sunday and 7 represents Saturday, with all the values in between representing the remaining weekdays, by order.

In the same direction as what was made for the incidents data set, on the tasks data set, the features *OPERATION CATEGORIZATION TIER 1*, *OPERATION CATEGORIZATION TIER 2*, and *OPERATION CATEGORIZATION TIER 3* were concatenated into a single feature (*OPERATION CATEGORIZATION TIER*), to overcome the missing values problem. The exact same approach was taken for the *PRODUCT CATEGORIZATION TIER 1*, *PRODUCT CATEGORIZATION TIER 2*, and *PRODUCT CATEGORIZATION TIER 3* features, dropping the original columns that became unnecessary.

After this concatenation of each pair of three columns into one, it was possible to observe in the data set that there were just a few combinations of the occurrences of the *OPERATION*

CATEGORIZATION TIER and *PRODUCT CATEGORIZATION TIER* values. To reduce the dimensionality of the data set and capture the relationship among the features, these columns were thus concatenated once more into a single column named *CATEGORIZATION TIER*.

From the *SUMMARY* feature, before its removal, with the usage of a REGEX expression, the *TASK TYPE* was obtained and stored in an individual feature. The *SUMMARY* field follows the pattern '*SUMMARY [TASK TYPE]*', which eased the extraction of the *TASK TYPE* using the following REGEX expression

$$'(.*)\[(.+)\]'$$

All the tasks without an associated task type were discarded due to the fact the task type is strictly necessary for the scope of the prediction to be made. The task type was converted from its nominal representation to a discrete numerical one.

The *SITE* feature was used to obtain the public holidays, following the same logic as the one for the incidents. Also, the day of the week was created for each task to denote this extra information, following the same logic used for the incidents.

4.2.3 Data integration and formatting

On the data integration and formatting, the first thing to do for the incidents was to convert the features with date and time into date features since the time is not a piece of relevant information for the scope of this Master Thesis. Afterward, the entries in the data set were ordered chronologically, from the oldest incident to the newest.

An assessment was made for the nominal (categorical) columns to check the number of unique values each contained. A conversion was then made between the original value and a discrete numerical representation.

After the data processing for the incidents described above was concluded, the data was separated into two distinct data sets. One data set contains only future covariates, while another one contains future and past covariates, with both data sets being used later on during the modeling stage, depending on the characteristics of each model used.

According to the Darts documentation [7], which is a library in Python specifically designed for Time Series Forecasting, covariates "refer to external data that can be used as inputs to models to help improve forecasts." Covariates distinguish from the target variable in a way that while the latter is the one to be forecasted, the first is not predicted at all.

The Darts documentation also enumerates three kinds of covariates:

- **Past covariates**, which are data only known in the past. For the scope of this Master Thesis, an example of the past covariates is the ticket type, which is only known for the past entries and not for the ones being forecasted.
- **Future covariates**, which are data known from the past to the future. The weekday and holiday information is an example of future covariates.

- *Static covariates*, which are data that are constant over time, like IDs. In this Master Thesis, static covariates are not used.

The past covariates for the incidents are the *DURATION*, *IMPACT*, *PRODUCT CATEGORIZATION TIER*, *PRODUCT NAME*, *PRIORITY*, *SOURCE*, *TICKET TYPE*, *URGENCY*, *FIRST LEVEL*, *FIRST LEVEL CREATED*, *CREATED VIA XML GATEWAY*, *IS WEB TICKET*, while the *DATE*, *WEEKDAY*, and *HOLIDAYS* are the future covariates.

With this information in mind, two data sets were created from the one obtained until this moment.

One of the data sets, from now on named as past data set, was obtained by grouping the original data set by *DATE* in order to obtain a data set with just one entry per date, choosing then the mode (*PRODUCT CATEGORIZATION TIER*, *PRODUCT NAME*, *SOURCE*, *TICKET TYPE*, *FIRST LEVEL*, *FIRST LEVEL CREATED*, *WEEKDAY*), the mean (*DURATION*, *IMPACT*, *PRIORITY*, *URGENCY*) or the sum (*CREATED VIA XML GATEWAY*, *IS WEB TICKET*, *HOLIDAYS*) as the aggregation method.

The future data set was updated to include missing entries, i.e. entries that contain all the missing combinations of the feature's values. To do so, and since it was not feasible to do it otherwise with the technology resources available, some columns were discarded (*PRODUCT NAME*, *PRODUCT CATEGORIZATION TIER*, *DURATION*, *SOURCE*, *FIRST LEVEL*, *FIRST LEVEL CREATED*, *CREATED VIA XML GATEWAY*, *IS WEB TICKET*). Then, for the date interval of the data set (i.e., for the interval from the oldest incident to the newest), all the missing combinations of values from the features *IMPACT*, *TICKET TYPE*, *PRIORITY*, and *URGENCY* were created and added to the data set as individual entries, for which the weekday and holidays information was calculated right after.

The data preparation for the incidents resulted in two data sets: the past data set with 1.997 entries and a dimensionality of 16, while the future data set had 547.900 entries and a dimensionality of 8.

The first thing to do for the tasks data set regarding data integration and formatting was also the conversion from date and time to time of all the features that contained the time information, as well as the chronological ordering of the tasks from the oldest to the newest.

The nominal features were also converted into discrete numerical features.

For the task types extracted from the *SUMMARY* field, a conversion was made from the values obtained with the regular expression to a smaller set manually defined by the SAP Basis team at Infineon. Each new task type results in the grouping of previous task types with similar meanings into a more general group. The total number of task types after this transformation was 18.

After the data processing for the tasks described above was concluded, the data set was divided into two in a similar process to the one made for the incidents. While the future data set was obtained by copying the data set available at this stage, the past data set was obtained with the grouping of the data by the pair *DATE* and *TASK TYPE* in order to obtain a data set with just one

entry per date-task type combination, choosing then the mode (*CATEGORIZATION TIER*, *PRODUCT NAME*, *PRIORITY*, *TICKET TYPE*, *WEEKDAY*) or the sum (*HOLIDAYS*) as the aggregation method.

For the future data set, a similar process to the one carried on for the incidents was put into practice. The missing combinations of the features *TASK TYPE*, *TICKET TYPE*, and *PRIORITY* were added to the data set as new entries after removing the features *PRODUCT NAME* and *CATEGORIZATION TIER* due to technological limitations. The holidays and weekday information were calculated and added to these new entries.

For the tasks, the past covariates are the *TICKET TYPE*, *PRIORITY*, *PRODUCT NAME*, *CATEGORIZATION TIER*, while the future covariates are the *DATE*, *TASK TYPE*, *WEEKDAY*, *HOLIDAYS*.

This process resulted in a future data set with 809.882 entries and a dimensionality of 7, while the past data set had 13.069 entries and a dimensionality of 9.

4.3 Data Visualization

To visualize the predictions obtained for the incidents and tasks, a dashboard was built. SAP UI5, which stands for SAP User Interface for HTML5, was used for that.

SAP UI5 is a JavaScript framework and UI library that helps developers create cross-platform enterprise-ready web applications in an efficient way [48].

Microsoft Visual Studio was used to develop the dashboard, using the SAP Fiori Tool extension pack [46].

With this pack, the creation of the project structure for the dashboard was eased and made according to the latest guidelines and recommendations from SAP.

The development using SAP UI5 uses the MVC design pattern to separate the data from user interactions and visual interface. MVC stands for Model-View-Controller, and it is a well-known design pattern that was introduced in the late 70s by Trygve Reenskaug [43].

The implementation was based on the existing example provided by the official documentation for stacked columns [49].

Starting with the model component, which is the simplest one, contains the device model, which is used to identify the device from which the dashboard is being accessed.

The view component contains the code for the visual interface, which is divided into three main components: the settings box, the incidents graphic, and the tasks graphic.

The settings box contains two drop-downs to select the prediction intervals, one for the incidents and another for the tasks, containing five options each. Each option refers to one of the supported prediction intervals, i.e., 7, 30, 60, 90, and 365 days, as shown in the figure 4.11.

The options and the handlers to refresh the data in the graphics are defined in the controller component.

The incidents graphic, which shows the prediction for the selected interval, is a bar chart, with the x values being the dates and the y values being the quantity predicted for that day.

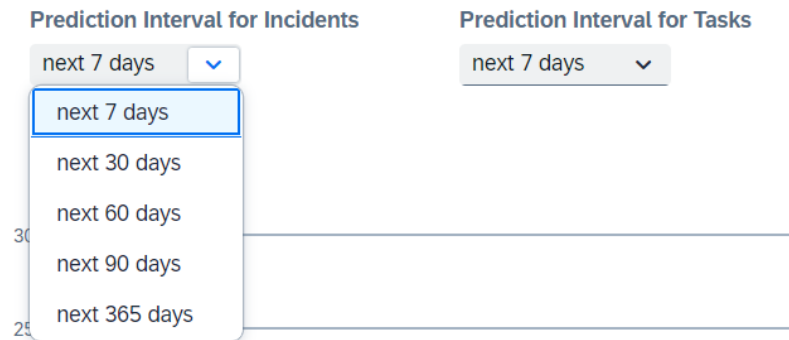


Figure 4.11: Drop-downs to select the prediction interval for the incidents and tasks.

The tasks graphic, which also shows the prediction for the selected interval, is a stacked bar chart, with each bar representing the total quantity of tasks predicted for that specific date and each sub-bar representing the prediction for the corresponding task type.

The x-axis contains each date from the selected date interval, while the y-axis contains the predicted quantity per date. The caption for the graphic contains the mapping between the colors and the task type. The colors are defined automatically, but the different task types are defined in the controller component.

Figure 4.12 shows an example of the dashboard with the prediction interval for the next seven days. Due to the confidentiality of the data, some information was masked.



Figure 4.12: Example of the dashboard with the prediction for the next seven days.

As it is possible to assess from figure 4.12, for the task prediction, the stacked bars for each predicted type have a unique color, which can be interpreted using the caption on the right side of the graphic.

However, although it is possible to see the total quantity predicted for a specific date, the same does not happen if the user wants to see the quantity for a date but only for a specific task type, which only appears if the stacked bar is large enough.

To solve this problem, a mouse click on a stacked bar opens a window with the information

about the quantity predicted for that date and task type, as shown in figure 4.13. Again, due to the confidentiality of the data, some information was masked.

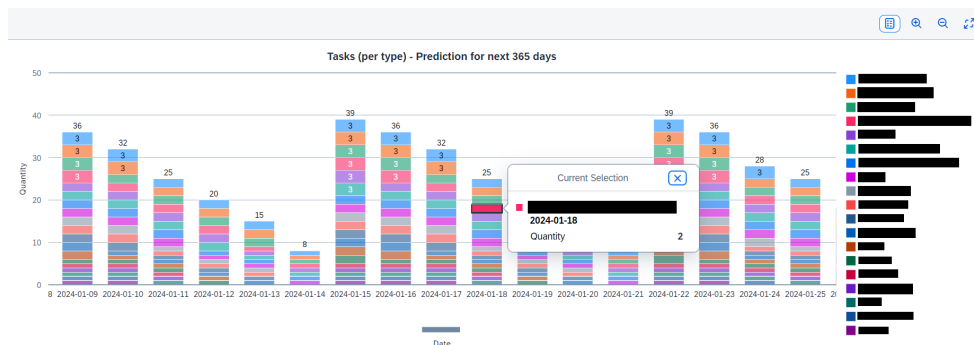


Figure 4.13: Example of a window that appears on mouse click on a stacked bar in the tasks prediction graphic.

Lastly, the controller component contains the initialization and setup of the dashboard with the data load. The data source is a JSON file for the incidents and another for the tasks. Both files are generated as the last step in the Jupyter Notebook, with the predictions for the incidents and for the tasks using the best-performing model for each. Its format differs for the incidents and for the tasks.

For the incidents, the format is the following:

```
1 {
2   "DATE": <DATE>,
3   "QUANTITY": <QUANTITY>
4 }
```

Listing 4.1: JSON example for the incidents.

Both *<DATE>* and *<QUANTITY>* are placeholders for the dates and quantities, respectively. For the tasks, the format is the following:

```
1 {
2   "DATE": <DATE>,
3   "TASKTYPE": <TASK_TYPE>,
4   "QUANTITY": <QUANTITY>
5 }
```

Listing 4.2: JSON example for the tasks.

<DATE>, *<TASK_TYPE>*, and *<QUANTITY>* are placeholders for the dates, task types, and quantities, respectively.

For visualization purposes, each graphic also includes three options, as shown in figure 4.14. The first one is used to hide/show the caption of the graphic (figure 4.15). The second one (set

of zoom in, zoom out) is used to zoom into the graphic, thus seeing each bar larger or thinner (figure 4.16). The third and last one is used to display the graphic in full-screen mode, thus being able to use the full width and height of the screen to see the graphic’s data (figure 4.17).



Figure 4.14: Set of options available for each graphic.

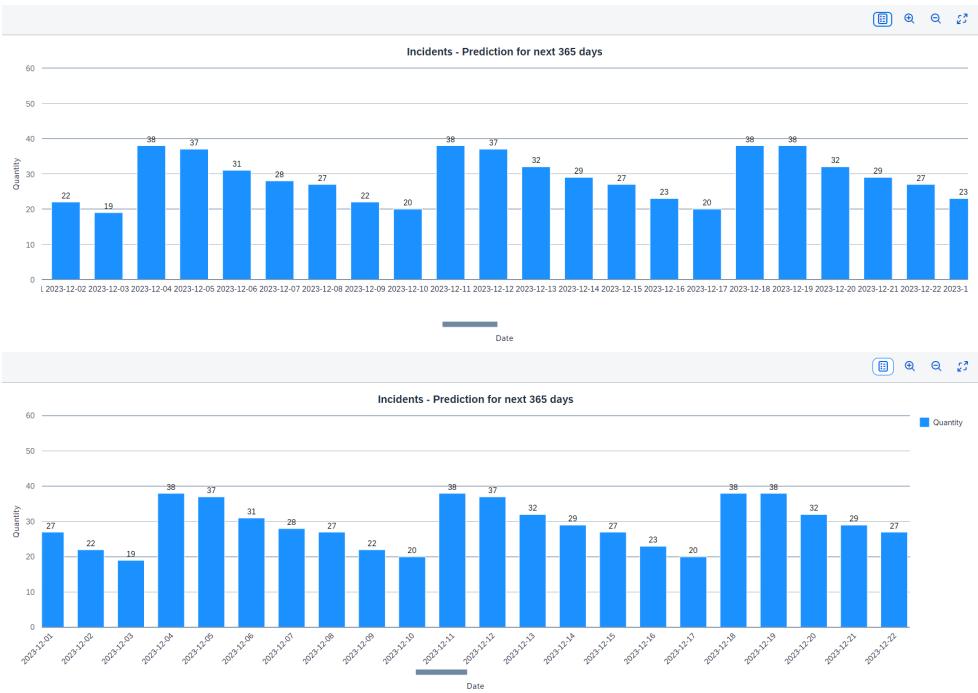


Figure 4.15: Hide (top) vs. Show (bottom) option for the graphic’s caption.

The properties of the dashboard, such as the axis for each graphic, the graphic’s title, and other characteristics, such as the place where the values are shown (on top of each bar), are also defined in the controller component.

The controller component also defines the handler function for the time interval drop-down. On a change of value made by the user, the data is filtered by date to show only the predictions within the selected time interval. The graphic’s title is also changed to reflect the new selection.

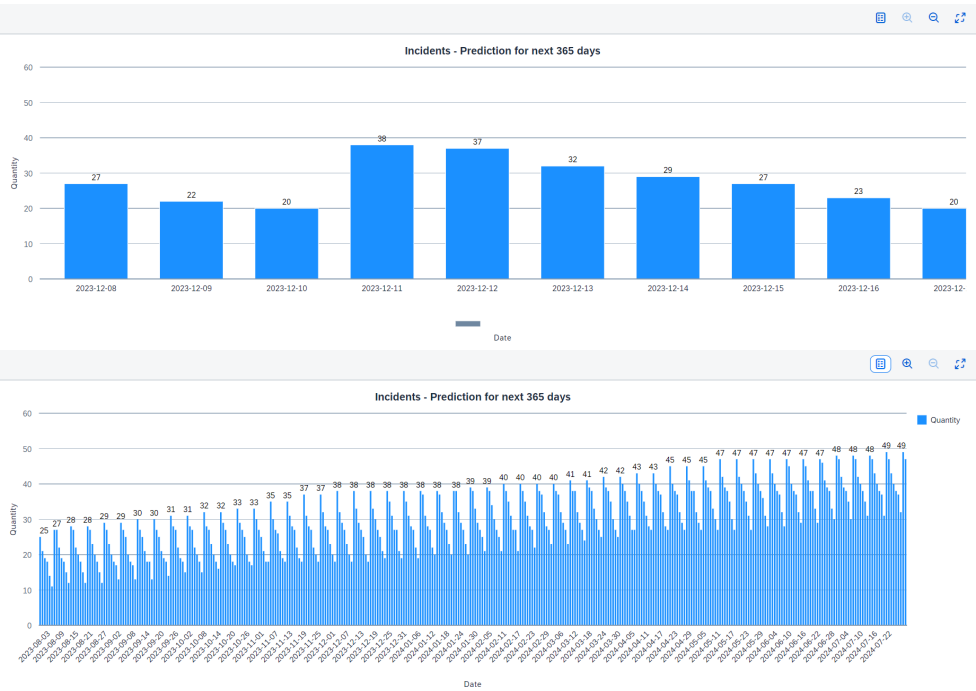


Figure 4.16: Maximum Zoom In (top) vs. Maximum Zoom Out (bottom) option for the graphics.

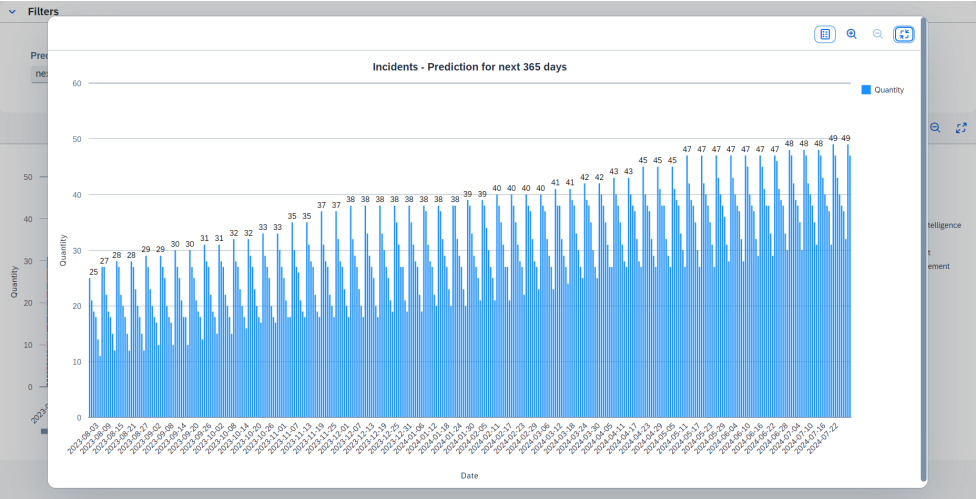


Figure 4.17: Graphic seen in full-screen mode.

Chapter 5

Models

With the conclusion of the data understanding and data preparation stages, the next step was to train and test the models. For that, all the models presented and discussed in Chapter 2 were used, both for the tasks and incidents.

In the first section, an explanation is made of the preparation of the train and test data sets, which is common to all the models. Then, in the following sections, each model's implementation is explained in detail.

5.1 Train and test data sets

In ML projects, it is common to divide the data set using an 80-20 distribution, where 80% of the data is used to train the models and the remaining 20% is used to test them.

However, that was not the procedure used for the division of the data set into train and test data for this Master Thesis.

Considering the five prediction intervals (7, 30, 60, 90, 365 days), an algorithm was applied to the tasks and incidents data sets, and, for each one, it was applied equally to the past data set and to the future data set mentioned in Chapter 4.

The algorithm consists of the following:

- For each prediction interval, six training data sets, and one testing data set are generated.
- The testing data set is composed of the data from the original data set whose *DATE* attribute is within the time interval between the last prediction interval days and the current date (e.g., for the prediction interval equal to 7, the test data set is composed by all the tasks/incidents from the last seven days, excluding the current day).
- The six training data sets are the following:
 1. One with all the data that is not part of the testing data set.
 2. One with the last 7 days of the data that is not part of the testing data set.
 3. One with the last 30 days of the data that is not part of the testing data set.

4. One with the last 60 days of the data that is not part of the testing data set.
5. One with the last 90 days of the data that is not part of the testing data set.
6. One with the last 365 days of the data that is not part of the testing data set.

With this, the purpose is to test, for each prediction interval, what is the best data interval to achieve the best results. This means that each model was trained and tested 30 times for the incidents data set and another 30 for the tasks data set, using either the past or future data set depending on the compatibility with the used models.

For most of the models, a hyperparameter optimization technique was used to get the best parameters since a manual selection of the parameters would require a try-and-repeat procedure until the best value was found. The details of the applied hyperparameter optimization technique are described together with each model implementation.

5.2 Lasso Regression

The Lasso implementation from the Python *sklearn.linear_model* library was used for the Lasso Regression [53].

This implementation receives a few parameters:

- *alpha* - float that represents the tuning parameter. This parameter is optional, and the default value is 1.0.
- *fit_intercept* - boolean that, once true, calculates the intercept for this model. If set to False, no intercept will be used in calculations (i.e., data is expected to be centered). This parameter is optional, and the default value is True.
- *precompute* - boolean or pre-computed Gram matrix to speed up calculations. This parameter is optional, and the default value is False.
- *copy_X* - boolean to define if X should be copied (True) or overwritten (False). This parameter is optional, and the default value is True.
- *max_iter* - maximum number of iterations. This parameter is optional, and the default value is 1000.
- *tol* - float that represents the tolerance for the optimization: if the updates are smaller than *tol*, the optimization code checks the dual gap for optimality and continues until it is smaller than *tol*. This parameter is optional, and the default value is 1e-4.
- *warm_start* - boolean that, when set to True, reuses the solution of the previous call to fit as initialization. Otherwise, just erase the previous solution. This parameter is optional, and the default value is False.

- *positive* - boolean that, when set to True, forces the coefficients to be positive. This parameter is optional, and the default value is False.
- *random_state* - integer that represents the seed of the pseudo-random number generator that selects a random feature to update. Used when *selection* equal to 'random.' When used, it produces the same output across multiple function calls. This parameter is optional, and the default value is None.
- *selection* - 'cyclic' or 'random' values. If set to 'random,' a random coefficient is updated every iteration rather than looping over features sequentially by default. This (setting to 'random') often leads to significantly faster convergence, especially when *tol* is higher than 1e-4. This parameter is optional, and the default value is 'cyclic.'

All the parameters, except the *alpha*, were kept with their default value. For the *alpha*, since it represents a tuning parameter that must be carefully chosen, as explained in Chapter 2, a hyperparameter optimization technique was used with the help of the *GridSearchCV* function from the *sklearn.model_selection* library [51].

According to the official documentation of *GridSearchCV*, the function receives the following parameters:

- *estimator* - instance of the used model, which is an instance of the Lasso implementation presented before. It is a mandatory parameter.
- *param_grid* - dictionary with the keys being the model parameters and the values being the intervals for the corresponding model parameter. It is also a mandatory parameter.
- *scoring* - defines the score to be used while searching for the best model parameters among the ones passed. This parameter is optional, and the default value is None.
- *n_jobs* - number of jobs to run in parallel. This parameter is optional, and the default value is None.
- *refit* - if set to True, allows the instance of the *GridSearchCV* to be callable to make predictions and also to return the best model parameters combination. This parameter is optional, and the default value is True.
- *cv* - determines the cross-validation splitting strategy. If the estimator is a classifier and *y* is either binary or multi-class, *StratifiedKfold* [55] is used. In all the other cases, *KFold* [52] is used. The splits are the same across multiple function calls. This parameter is optional, and the default value is None, which means that it uses the 5-fold cross-validation.
- *verbose* - refers to the quantity of messages. This parameter is optional, and the default is 0.
- *pred_dispatch* - controls the number of jobs that get dispatched during parallel execution. This parameter is optional, and the default value is $2 * n_jobs$.

- *error_score* - value to assign to the score if an error occurs in estimator fitting. This parameter is optional, and the default value is 'raise', meaning that any error is raised when it occurs.
- *return_train_score* - if True, the *cv_results_* attribute will include the training scores. This parameter is optional, and the default value is False.

In the *estimator*, as mentioned before, an instance of Lasso without any parameter was passed. For the *param_grid*, a dictionary with only one pair key value was used. The key is the alpha parameter that Lasso receives, and the values range from 0 to 5 with steps equal to 0.1. For the *scoring* parameter, the mean squared error was the selected metric. This metric was selected because, from the ones available [50], this was one of the used afterward in the evaluation of the model. All the remaining values were kept as default.

The future data set was the one used, both for the tasks and incidents, since the Lasso Regression model and its implementation do not support out-of-the-box past covariates.

The best value for the tuning parameter was found, both for the tasks and the incidents. The results can be seen in Chapter 6.

5.3 Ridge Regression

The Ridge implementation from the Python *sklearn.linear_model* library was used for the Ridge Regression [54].

This implementation receives a few parameters:

- *alpha* - float that multiplies the L2 term. This parameter is optional, and the default value is 1.0.
- *fit_intercept* - boolean that, once true, calculates the intercept for this model. If set to False, no intercept will be used in calculations (i.e., data is expected to be centered). This parameter is optional, and the default value is True.
- *copy_X* - boolean to define if X should be copied (True) or overwritten (False). This parameter is optional, and the default value is True.
- *max_iter* - maximum number of iterations for conjugate gradient solver. This parameter is optional, and the default value depends on the solver.
- *tol* - float that determines the precision of the solution by specifying the different convergence criteria depending on the solver. This parameter is optional, and the default value is 1e-4.
- *solver* - solver to use in the computational routines. This parameter accepts the following values:

1. `'auto'` - chooses the solver automatically depending on the type of data.
2. `'svd'` - stands for Singular Value Decomposition, and it is the most stable solver at the cost of being slower. X is decomposed to compute the Ridge coefficients.
3. `'cholesky'` - uses the standard `scipy.linalg.solve` [59] function to obtain a closed-form solution.
4. `'lsqr'` - uses the dedicated regularized least-squares routine `scipy.sparse.linalg.lsqr` [57]. It is the fastest and uses an iterative procedure.
5. `'sparse_cg'` - uses the conjugate gradient solver as found in `scipy.sparse.linalg.cg` [56]. As an iterative algorithm, this solver is more appropriate than `'cholesky'` for large-scale data (possibility to set the parameters `tol` and `max_iter`).
6. `'sag'` and `'saga'` - `'sag'` uses a Stochastic Average Gradient descent, and `'saga'` uses its improved, unbiased version named SAGA. Both methods also use an iterative procedure and are often faster than other solvers when both the number of samples and features are large.
7. `'lbfgs'` - uses L-BFGS-B algorithm implemented in `scipy.optimize.minimize` [58]. It can be used only when the parameter `positive` is set to `True`.

All solvers except `'svd'` support both dense and sparse data. However, only `'lsqr'`, `'sag'`, `'sparse_cg'`, and `'lbfgs'` support sparse input when the parameter `fit_intercept` is `True`. This parameter is optional, and the default value is `'auto'`.

- `positive` - boolean that, when set to `True`, forces the coefficients to be positive. In this case, the only solver that supports it is the `'lbfgs'`. This parameter is optional, and the default value is `False`.
- `random_state` - integer, used only when the solver is `'sag'` or `'saga'`, to shuffle the data. This parameter is optional, and the default value is `None`.

All the parameters, except the `alpha`, were kept with their default value. For the `alpha`, since it must be carefully chosen as explained in Chapter 2, the same hyperparameter optimization technique used in Lasso Regression was used.

Regarding the parameters for the `GridSearchCV` function, the ones used in Lasso Regression were the same used here, except, of course, for the `estimator`, which received an empty instance of the Ridge regression presented before.

In the same direction as in the Lasso Regression, the future data set was the one used to train and test the model, both for the incidents and for the tasks, since the Ridge Regression model and its implementation do not support out-of-the-box past covariates.

The best value for `alpha` was found, both for the tasks and incidents, and the results can be seen in Chapter 7.

5.4 ARIMA

The ARIMA model was implemented with the usage of the ARIMA implementation from the Python *statsmodels* library [65].

The implementation receives the following parameters:

- *endog* - the time series to train the model with. It is a mandatory parameter, and there is no default value for it.
- *exog* - an array with the exogenous variables used. It is an optional parameter, and the default value is None.
- *order* - the values for the autoregressive, difference, and moving average components, i.e., (P,D,Q). D must be an integer, but P and Q can be either an integer or a list of integers. This parameter is optional, and the default value is (0,0,0).
- *seasonal_order* - on the same direction as the previous parameter, this one refers to the seasonal values, with the fourth value being the seasonality. This value can be an integer, and all the others follow the same logic as for the *order* parameter. It is an optional parameter, and its default value is (0,0,0,0).
- *trend* - either a string or an iterable that defines the deterministic trend. If it is a string, it can take as values 'n' (no trend), 'c' (constant trend), 'l' (linear trend), and 'ct' (both trends). If it is an iterable, it defines a polynomial. It is an optional parameter, and the default value is 'c' for models without integration and 'n' for models with integration.
- *enforce_stationarity* - boolean that, if set to true, requires the autoregressive parameters to correspond to a stationary process. It is an optional parameter, and the default value is True.
- *enforce_invertibility* - boolean that, if set to true, requires the moving average parameters to correspond to a stationary process. It is an optional parameter, and the default value is True.
- *concentrate_scale* - boolean that, if set to True, concentrates the scale, i.e., the variance of the error term, out of the likelihood. By doing so, the number of parameters is reduced in one. This is only applicable when considering estimation by numerical maximum likelihood. It is an optional parameter, and the default value is False.
- *trend_offset* - integer that represents the start time offset in which trend values are considered. It is an optional parameter, and the default value is 1.
- *dates* - array of dates with time, only used in case the *exog* and *endog* parameters do not include an index. The parameter is optional, and the default value is None.
- *freq* - string that represents the time series frequency. Only used if the *exog* and *endog* parameters do not include an index. It is an optional parameter, and the default value is None.

- *missing* - string that represents the handling that should be made to missing values. It can receive the value 'none', which is the default, if no checking is done to missing values; 'drop', if any observations with missing values are dropped; or 'raise', if an error is raised in case missing values are found. It is an optional parameter.
- *validate_specification* - boolean whose information is not available in the official documentation. It is an optional parameter, and the default value is True.

For the incidents, all the parameters, except the *endog* and *order*, were kept with their default value. The *endog* was set with the time series, i.e. the date and quantity per date information, and the *order* was set with the values defined for it.

For the tasks, the same parameters as for the incidents were used, but the *exog* was set to be the task type. This made the implementation of ARIMA, in fact, a sort of an implementation of ARIMAX since the task type was the exogenous variable used.

The hyperparameter optimization was applied to the *order*, but since the used ARIMA implementation is not compatible with the *GridSearchCV* function used for the Ridge and Lasso Regressions, the hyperparameter optimization was achieved by pre-defining a list with all the combinations of the values for P, D, and Q.

The values for P and Q were either 1 or 2, while for D were either 0 or 1. To compare the model's performance depending on the parameters, the MAE metric was used.

Unlike the Lasso and Ridge Regression, the past data set was the one used for the incidents and for the tasks. Since no covariate was used, except the task type, which is considered a future covariate, it was irrelevant which data set was used. However, the decision to proceed with the past data set was because it contains fewer data entries as the data is grouped by date (for the incidents) and by the group data-task type (for the tasks).

The best values for P, D, and Q, as well as the obtained metrics, both for the tasks and for the incidents, can be seen in Chapter 6.

5.5 SARIMAX

The SARIMAX model was implemented with the usage of the SARIMA implementation from the same Python library used for the ARIMA implementation, *statsmodels* [66].

The implementation receives the following parameters:

- *endog*, *exog*, *order*, *seasonal_order*, *trend*, *enforce_stationarity*, *enforce_invertibility*, *concentrate_scale*, *trend_offset*, *dates*, *freq*, *missing*, *validate_specification* - these parameters are the exact same used in the ARIMA implementation, from the same Python library.
- *measurement_error* - boolean that, if set to True, assumes that the exogenous variables were observed with error. This parameter is optional, and the default value is False.

- *time_varying_regression* - boolean that, if set to True, assumes that the exogenous variables coefficients are allowed to vary over time. This parameter is optional, and the default value is False.
- *mle_regression* - boolean that, if set to True, makes the model estimate the regression coefficients of the exogenous variables as part of the maximum likelihood estimation or through the Kalman filter. If the *time_varying_regression* value is True, this parameter must be set to False. It is an optional parameter whose default value is True.
- *simple_differencing* - boolean that represents the usage (True) or not (False) of partially conditional maximum likelihood estimation. In other words, if the value of this parameter is set to True before the estimation, the differencing is performed, discarding the initial rows and resulting in a smaller state-space formulation. If the value of this parameter is set to False, all the data points can be used in the estimation since the full SARIMAX model is put in the state-space form. This parameter is optional, and its default value is False.
- *hamilton_representation* - boolean that, if set to True, makes the model use the Hamilton representation of an ARMA process. If set to False, the model uses the Harvey representation. This parameter is optional, and its default value is False.
- *use_exact_diffuse* - boolean that, if set to True, makes the model use exact diffuse initialization for non-stationary states. If set to false, instead of the exact diffuse initialization, use an approximate diffuse initialization. This parameter is optional, and its default value is False.
- ***kwargs* - this parameter represents any keyword arguments that may be used to provide default values for state-space matrices or for Kalman filtering options. It is an optional parameter with no default value associated.

All the parameters, except the *endog*, *exog*, *order*, and *seasonal_order*, were kept with their default value. For the *endog*, the time series, i.e., the dates and the quantities per date, were used.

For the *exog*, the value differs for the incidents and tasks. For the incidents, it contains the *TICKET TYPE*, *PRIORITY*, *IMPACT*, *URGENCY*, *WEEKDAY*, and *HOLIDAYS*, which is a mix of past and future covariates. For the tasks, it contains the *TASK TYPE*, *WEEKDAY*, *HOLIDAYS*, *TICKET TYPE*, and *PRIORITY*, which are also a mix of past and future covariates. The *order* and *seasonal_order* parameters were set with the values defined for them.

On the same logic as the one applied in the ARIMA implementation, since the SARIMAX implementation used is not compatible with the *GridSearchCV* function used in the Lasso and Ridge Regressions implementations, a "manual" hyperparameter optimization technique was used.

Again, the values for the *order* and *seasonal_order* resulted in all the possible combinations between the different values set for P, Q, D, p, q, d, s. For P, Q, p, and q, the possible values were either 1 or 2. For D and d, the possible values were 0 or 1. Finally, for s, the possible values were 7 (weekly seasonality) or 12 (monthly seasonality).

To assess which model performed better and compare the different values used for each parameter, the MAE metric was used.

The future data set, as it was used for the Lasso and Ridge Regressions implementations, was the selected one, both for the tasks and incidents, to train and test the SARIMAX model.

The best values for P, Q, D, p, q, d, s, as well as the obtained metrics, both for the tasks and for the incidents, can be seen in Chapter 6.

5.6 RNN

The RNN model was implemented with the usage of some *TensorFlow* Python libraries [69].

Before the definition of the network, the target variable, i.e., the quantity, was reshaped so that its values would be in the interval from 0 to 1. The remaining features were also expanded from a 2-dimensional to a 3-dimensional matrix, using the *NumPy* *expand_dims* function [40].

For the network itself, the *Sequential* implementation [73] was used, which receives the following parameters:

- *layers* - layers of the network. This parameter is optional, and the default value is None.
- *name* - name of the network. This parameter is optional, and the default value is None.

The *name* was kept with the default value, but the *layers* parameter was assigned with two implementations also from *TensorFlow*, *SimpleRNN* [74] and *Dense* [70].

The *SimpleRNN* represents the input and the only hidden layer of the network. Its implementation receives the following parameters:

- *units* - integer that represents the dimensionality of the output space. It is a mandatory parameter.
- *activation* - activation function. This parameter is optional, and its default value is '*tanh*' (i.e., hyperbolic tangent).
- *use_bias* - if set to True (default), the layer uses a bias vector. This parameter is optional.
- *kernel_initializer* - initializer for the *kernel* weights matrix, which is used for the linear transformation of the inputs. This parameter is optional, and its default value is '*glorot_uniform*'.
- *recurrent_initializer* - initializer for the *recurrent_kernel* weights matrix, which is used for the linear transformation of the recurrent state. This parameter is optional, and its default value is '*orthogonal*'.
- *bias_initializer* - initializer for the bias vector. This parameter is optional, and its default value is '*zeros*'.
- *kernel_regularizer* - regularizer function applied to the *kernel* weights matrix. This parameter is optional, and its default value is None.

- *recurrent_regularizer* - regularizer function applied to the *recurrent_kernel* weights matrix. This parameter is optional, and its default value is None.
- *bias_regularizer* - regularizer function applied to the bias vector. This parameter is optional, and its default value is None.
- *activity_regularizer* - regularizer function applied to the output of the layer, i.e., its activation. This parameter is optional, and its default value is None.
- *kernel_constraint* - constraint function applied to the *kernel* weights matrix. This parameter is optional, and its default value is None.
- *recurrent_constraint* - constraint function applied to the *recurrent_kernel* weights matrix. This parameter is optional, and its default value is None.
- *bias_constraint* - constraint function applied to the bias vector. This parameter is optional, and its default value is None.
- *dropout* - float that takes values between 0 and 1, representing the fraction of the units to drop for the linear transformation of the inputs. This parameter is optional, and its default value is 0.0.
- *recurrent_dropout* - float that takes values between 0 and 1, representing the fraction of the units to drop for the linear transformation of the recurrent state. This parameter is optional, and its default value is 0.0.
- *return_sequences* - boolean that, if set to True, makes the layer return the last output in the output sequence or the full sequence. This parameter is optional, and its default value is False.
- *return_state* - boolean that, if set to True, makes the layer return the last state in addition to the output. This parameter is optional, and its default value is False.
- *go_backwards* - boolean that, if set to True, processes the input sequence backward and returns the reversed sequence. This parameter is optional, and its default value is False.
- *stateful* - boolean that, if set to True, makes the last state for each sample at index *i* in a batch to be used as the initial state for the sample of index *i* in the following batch. This parameter is optional, and its default value is False.
- *unroll* - boolean that, if set to True, makes the network be unrolled. Otherwise, a symbolic loop will be used. Unrolling can speed up a RNN, although it tends to be more memory-intensive. Unrolling is only suitable for short sequences. This parameter is optional, and its default value is False.
- ***kwargs* - this parameter represents any keyword arguments. It is an optional parameter with no default value associated.

All the parameters were kept with their default value, except the *units*, *activation* and *input_shape*.

The *units* were set to either 25 or 50, which was made with a manual hyperparameter optimization technique. The MAE was the selected metric to compare the model's performance with the different *units* values.

The *activation* was set to '*relu*', i.e., the rectified linear unit was the selected activation function. Last but not least, the *input_shape* was set to a tuple of two elements, the first equal to the number of features in the training data (7 for the incidents and 6 for the tasks) and the second equal to 1.

The *Dense* represents the output layer of the network, and its implementation receives the following parameters:

- *units*
- *activation*
- *use_bias*
- *kernel_initializer*
- *bias_initializer*
- *kernel_regularizer*
- *bias_regularizer*
- *activity_regularizer*
- *kernel_constraint*
- *bias_constraint*
- ***kwargs*

All the above parameters have the same meaning and default values as for the *SimpleRNN* implementation.

All the values were kept by default, except the *units*, which was set to 1, representing the dimensionality of the output of the network, i.e., the target variable value that is forecasted.

The model was compiled with the *compile* method from *TensorFlow* [72], which receives the following parameters:

- *optimizer* - the optimizer to use. This parameter is optional, and its default value is '*rm-sprop*'.
- *loss* - the loss function. This parameter is optional, and its default value is *None*.

- *metrics* - list of metrics to be evaluated by the model during training and testing. This parameter is optional, and its default value is None.
- *loss_weights* - list or dictionary specifying scalar coefficients to weight the loss contributions of different model outputs. This parameter is optional, and its default value is None.
- *weighted_metrics* - list of metrics to be evaluated and weighted by *sample_weight* or *class_weight* during training and testing. This parameter is optional, and its default value is None.
- *run_eagerly* - boolean that, if set to True, will make the model's logic not be wrapped. This parameter is optional, and its default value is False.
- *steps_per_execution* - The number of batches to run. This parameter is optional, and its default value is 1.
- *jit_compile* - boolean that, if set to True, compile the model training step with XLA. XLA is an optimizing compiler for machine learning. This parameter is optional, and its default value is None.
- *pss_evaluation_shards* - sets the number of shards to split the data set into. This parameter is optional, and its default value is 0.
- ***kwargs* - this parameter represents any keyword arguments supported for backward compatibility only. It is an optional parameter with no default value associated.

All the default values were kept by default, except for the *optimizer* and *loss* parameters. For the *optimizer*, Adam was the selected one. Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments. For the *loss*, the MAE was the selected metric.

The model was trained with 10 epochs, a batch size of 32, and a validation split of 0.1, i.e., 10% of the data was used at each epoch to assess the model's performance.

Lastly, the target variable values were converted again to the original interval of values instead of the 0 to 1 interval.

The future data set, as for the Lasso and Ridge Regression and SARIMAX implementations, was the selected one, both for the tasks and incidents, to train and test the RNN model.

The best values for the nodes and the obtained metrics, both for the tasks and for the incidents, can be seen in Chapter 6.

5.7 LSTM

For the LSTM implementation, some of the *TensorFlow* Python libraries used in the RNN implementation were also used.

As for the RNN implementation, the first thing to do in the LSTM implementation was to reshape the target variable so that its values could be between 0 and 1, as well as to expand the remaining features from a 2-dimensional to a 3-dimensional matrix.

For the network itself, the *Sequential* implementation was used, with only the *layers* parameter being assigned to two implementations, *LSTM* [71] and *Dense*.

The *LSTM* represents the input and the only hidden layer of the network. Its implementation receives the following parameters:

- *units*
- *activation*
- *recurrent_activation* - activation function to use for the recurrent step. This parameter is optional, and its default value is 'sigmoid'.
- *use_bias*
- *kernel_initializer*
- *recurrent_initializer*
- *bias_initializer*
- *unit_forget_bias* - boolean that, if set to True (default), If True, adds 1 to the bias of the forget gate at initialization. If the value of this parameter is True, the *bias_initializer* will automatically have 'zeros' as value. This parameter is optional.
- *kernel_regularizer*
- *recurrent_regularizer*
- *bias_regularizer*
- *activity_regularizer*
- *kernel_constraint*
- *recurrent_constraint*
- *bias_constraint*
- *dropout*
- *recurrent_dropout*
- *return_sequences*
- *return_state*

- *go_backwards*
- *stateful*
- *time_major* - shape format of the inputs and outputs tensors. If the value is True, the inputs and outputs will be in shape *[timesteps, batch, feature]*. If the value is False, the same shape will be *[batch, timesteps, feature]*. This parameter is optional, and its default value is False.
- *unroll*
- ***kwargs*

All the parameters without description in the list above have the same meaning and default values as the ones of the *SimpleRNN* implementation.

Except for the *units*, *activation*, and *input_shape* parameters, all the others were kept with their default values. The *units* parameter, like for the LSTM, was set to either 25 or 50, in what consisted of the manual hyperparameter optimization applied to the model. The MAE metric was used to assess the model's performance and decide the best value for the *units*.

The *activation* function was *'relu'*, and the *input_shape* was set to a tuple with two values, with the first being the number of features in the training data (1 for the incidents and 2 for the tasks) and the second one the value 1, as for the RNN implementation.

The *Dense* represents the output layer of the network, and its implementation is the exact same as the one used for the RNN, i.e., the *units* parameter was set to 1, and all the other were kept with their default values.

The model was also compiled with the Adam optimizer and MAE as the loss metric.

The training of the model was made in 10 epochs, with a batch size of 32 and a validation split of 0.1.

By the end, the target variable was also reconverted to the original scale of values.

For the LSTM implementation, the past data set was used, both for the tasks and for the incidents, since the training data only contained the *DATE*, *QUANTITY*, and *TASKTYPE*, with this last one being only applicable to the tasks data set.

The best values for the nodes, as well as the obtained results for the metrics, both for the tasks and for the incidents, can be seen in Chapter 6.

5.8 LSTMX

The implementation for the LSTMX model was the exact same one used for the LSTM model, with the only difference being the data set used, both for the tasks and for the incidents.

Instead of the past data set, the future data set was the one used since the model was trained and tested with all the data available in this data set.

With this, the *input_shape* parameter of the *LSTM* implementation was set to a tuple with two elements, with the first one being the number of features in the training data (7 for the incidents and 6 for the tasks), and the second one equal to 1.

The best values for the nodes, as well as the obtained results for the metrics, both for the tasks and for the incidents, can be seen in Chapter 6.

5.9 DeepAR

The DeepAR model was implemented with the usage of the DeepAR implementation from the *PyTorch Forecasting* Python library [15] as an adapted version of the implementation made by Cory Maklin [32].

For this implementation, the past data set was used, both for the tasks and for the incidents. However, there was the need to do some pre-processing in order for it to be compatible with the model's implementation.

The first thing to do was to shuffle the training data so that it would not be ordered by date, as recommended in the official documentation of DeepAR [60].

Then, all the columns except the *DATE* and *QUANTITY* had to be converted to the String type.

A new column was created, *TIME INDEX*, which was obtained by getting the minimum *DATE* value available in the training data set and calculating the difference between each row's *DATE* value and this minimum.

Then, the data set was converted to a Time Series data set with the usage of the *TimeSeries-DataSet* implementation also from the *PyTorch Forecasting* Python library [16].

This implementation receives the following parameters:

- *data* – data frame with sequence data, where each row can be identified with *time_idx* and the *group_ids*. This parameter is mandatory and has no default value.
- *time_idx* – integer column denoting the time index, which is used to determine the sequence of samples. This parameter is mandatory and has no default value.
- *target* – column denoting the target or list of columns denoting the target variables. This parameter is mandatory and has no default value.
- *group_ids* – list of column names identifying a time series. This parameter is mandatory and has no default value.
- *weight* – column name for weights. This parameter is optional, and the default value is None.
- *max_encoder_length* – maximum length to encode. This parameter is optional, and the default value is 30.
- *min_encoder_length* – minimum allowed length to encode. This parameter is optional, and the default value is the same as the *max_encoder_length*.
- *min_prediction_idx* – minimum *time_idx* from where to start predictions. This parameter is optional, and the default value is None.

- *max_prediction_length* – maximum prediction/decoder length. This parameter is optional, and the default value is 1.
- *min_prediction_length* – minimum prediction/decoder length. This parameter is optional, and the default value is *max_prediction_length*.
- *static_categoricals* – list of categorical variables that do not change over time. This parameter is optional, and the default value is an empty list.
- *static_reals* – list of continuous variables that do not change over time. This parameter is optional, and the default value is an empty list.
- *time_varying_known_categoricals* – list of categorical variables that change over time and are known in the future. This parameter is optional, and the default value is an empty list.
- *time_varying_known_reals* – list of continuous variables that change over time and are known in the future. This parameter is optional, and the default value is an empty list.
- *time_varying_unknown_categoricals* – list of categorical variables that change over time and are not known in the future. This parameter is optional, and the default value is an empty list.
- *time_varying_unknown_reals* – list of continuous variables that change over time and are not known in the future. This parameter is optional, and the default value is an empty list.
- *variable_groups* – dictionary mapping a name to a list of columns in the data. The name should be present in a categorical or real class argument to be able to encode or scale the columns by group. This will effectively combine categorical variables and is particularly useful if a categorical variable can have multiple values at the same time. This parameter is optional, and the default value is an empty dictionary.
- *constant_fill_strategy* – dictionary of column names with constants to fill in missing values if there are gaps in the sequence (by default forward fill strategy is used). The values will be only used if the parameter *allow_missing_timesteps* is set to True. This parameter is optional, and the default value is an empty dictionary.
- *allow_missing_timesteps* – boolean that, if set to True, allows missing time steps, which are automatically filled up. Missing values refer to gaps in the *time_idx* parameter. This parameter is optional, and the default value is False.
- *lags* – dictionary of variable names mapped to a list of time steps by which the variable should be lagged. Lags can be useful to indicate seasonality to the models. A lagged variable has to appear in the time-varying variables. This parameter is optional, and the default value is an empty dictionary.

- *add_relative_time_idx* – boolean that, if set to True, adds a relative time index as a feature (i.e. for each sampled sequence, the index will range from *encoder_length* to *prediction_length*). This parameter is optional, and the default value is False.
- *add_target_scales* – boolean that, if set to True, adds scales for the target to static real features (i.e. add the center and scale of the not normalized time series as features). This parameter is optional, and the default value is False.
- *add_encoder_length* – boolean or string that indicates whether or not to add decoder length to a list of static real variables. This parameter is optional, and the default value is 'auto', i.e. the value is only True if the value of *min_encoder_length* is different from the value of *max_encoder_length*.
- *target_normalizer* – transformer that take *group_ids*, *target* and *time_idx* to normalize targets. It can take the values *TorchNormalizer*, *GroupNormalizer*, *NaNLabelEncoder*, *EncoderNormalizer*, *None*. This parameter is optional, and, by default, an appropriate normalizer is chosen automatically.
- *categorical_encoders* – dictionary of *scikit-learn* label transformers. This parameter is optional, and, by default, it is set effectively to *sklearn*'s *LabelEncoder()* [25].
- *scalers* – dictionary of *scikit-learn* scalers. The possible values are *EncoderNormalizer*, *GroupNormalizer* or *scikit-learn*'s *StandardScaler()*, *RobustScaler()*, *None*. This parameter is optional, and the default value is the *sklearn*'s *StandardScaler()* [26].
- *randomize_length* – if set to None or False, it will not randomize lengths. If it is a tuple, it represents the beta distribution concentrations from which probabilities are sampled that are used to sample new sequence lengths with a binomial distribution. If set to True, defaults to (0.2, 0.05), i.e. approx. 1/4 of samples around minimum encoder length. This parameter is optional, and the default value is False.
- *predict_mode* – boolean that, if set to True, will iterate over each time series once (only the last provided samples). Effectively, this will choose for each time series identified by *group_ids* the last *max_prediction_length* samples as prediction samples and everything previous up to *max_encoder_length* samples as encoder samples.

The *data* parameter was set with the training data set. The *time_idx* was set to the column *TIME INDEX*, which contains the time index. The *target* was set to the column *QUANTITY* and the *group_ids* was set to the columns *PRODUCT CATEGORIZATION TIER*, *PRODUCT NAME*, *SOURCE*, *TICKET TYPE*, *FIRST LEVEL*, *FIRST LEVEL CREATED*, *WEEKDAY*, *DURATION*, *IMPACT*, *PRIORITY*, *URGENCY*, *CREATED VIA XML GATEWAY*, *IS WEB TICKET*, *HOLIDAYS* for the incidents, and to the columns *CATEGORIZATION TIER*, *PRODUCT NAME*, *PRIORITY*, *TICKET TYPE*, *WEEKDAY*, *HOLIDAYS* for the tasks.

The *max_encoder_length* was set to the maximum value of the time index. The *min_encoder_length* was set to 1. The *max_prediction_length* was set to the prediction interval.

The *static_categoricals* were set to the columns *PRODUCT CATEGORIZATION TIER*, *PRODUCT NAME*, *SOURCE*, *TICKET TYPE*, *FIRST LEVEL*, *FIRST LEVEL CREATED*, *DURATION*, *IMPACT*, *PRIORITY*, *URGENCY*, *CREATED VIA XML GATEWAY*, *IS WEB TICKET* for the incidents, and to the columns *CATEGORIZATION TIER*, *PRODUCT NAME*, *PRIORITY*, *TICKET TYPE* for the tasks.

The *time_varying_known_reals* was set to the columns *WEEKDAY*, *HOLIDAYS*, both for the tasks and for the incidents, while the *time_varying_unknown_reals* was set to the target variable.

Finally, the *allow_missing_timesteps* parameter was set to True, and all the others were kept with their default value.

Two recommendations from the official documentation are that the model should be trained with at least 300 observations in the data set and that large prediction intervals, i.e., greater than four hundred, make the model slow and less accurate and, thus should be avoided.

Regarding the first recommendation, with the approach made in this Master Thesis, the training process follows the recommendation for the cases in which the training data set contains all the data or the data from the last 365 days. For this reason, and for this model only, instead of the 30 training cycles described before, only 10 training cycles were made, both for the tasks and for the incidents.

In other words, the model was trained only with all the data available or the data from the past 365 days in each of the prediction intervals.

Regarding the second recommendation, it is not a problem since the prediction intervals for this Master Thesis are not greater than four hundred (the maximum is 365).

According to the same documentation, "the method works with little or no hyperparameter tuning on a wide variety of data sets." For that reason, there was no hyperparameter optimization technique applied to this model.

Then, the training and testing data were divided into chunks of data of size 128 each. To have reproducible results, a seed was set (42).

An instance of *DeepAR* from the *PyTorch Forecasting* library was implemented. This instance receives the following parameters:

- *dataset* – time series data set. This parameter is mandatory.
- *allowed_encoder_known_variable_names* – list of known variables that are allowed in the encoder. This parameter is optional and defaults to all.
- ***kwargs* – additional arguments such as hyperparameters for model. This parameter is optional and has no default value.

The *allowed_encoder_known_variable_names* parameter was kept with the default value, but the *dataset* and ***kwargs* parameters were set with some values.

For the *dataset*, the previously created instance of *TimeSeriesDataSet* was passed. For the ***kwargs*, the following parameters were set:

- *learning_rate*=0.7
- *log_interval*=10
- *log_val_interval*=1
- *hidden_size*=30
- *rnn_layers*=2
- *loss*=*MultivariateNormalDistributionLoss(rank=30)*

To train and test the model, a new instance of a *Trainer* [31] was created. This instance receives the following parameters:

- *accelerator* – supports passing different accelerator types ('cpu', 'gpu', 'tpu', 'ipu', 'hpu', 'mps', 'auto') as well as custom accelerator instances. This parameter is optional, and the default value is 'auto'.
- *strategy* – supports different training strategies with aliases as well as custom strategies. This parameter is optional, and the default value is 'auto'.
- *devices* – the devices to use. It can be set to a positive number, a sequence of device indices, the value -1 to indicate all available devices should be used, or 'auto' for automatic selection based on the chosen accelerator. This parameter is optional, and the default value is 'auto'.
- *num_nodes* – number of GPU nodes for distributed training. This parameter is optional, and the default value is 1.
- *precision* – double precision, full precision, 16bit mixed precision or bfloat16 mixed precision. It can be used on CPU, GPU, TPUs, HPUs, or IPUs. This parameter is optional, and its default value is '32-true'.
- *logger* – logger (or iterable collection of loggers) for experiment tracking. If set to True, it uses the default *TensorBoardLogger* if it is installed, otherwise *CSVLogger*. If set to False, it will disable logging. If multiple loggers are provided, local files (checkpoints, profiler traces, etc.) are saved in the *log_dir* of the first logger. This parameter is optional, and its default value is True.
- *callbacks* – add a callback or list of callbacks. This parameter is optional, and its default value is None.
- *fast_dev_run* – runs *n* if set to *n* (integer), else 1 if set to True, batch(es) of training, validation, and test to find any bugs. This parameter is optional, and its default value is False.

- *max_epochs* – stop training once this number of epochs is reached. This parameter is optional, and is disabled by default. If both *max_epochs* and *max_steps* are not specified, it defaults to 1000.
- *min_epochs* – force training for at least these many epochs. This parameter is optional, and is disabled by default.
- *max_steps* – stop training after this number of steps. This parameter is optional, and is disabled by default. If *max_steps* = -1 and *max_epochs* = None, it will default to 1000.
- *min_steps* – force training for at least this number of steps. This parameter is optional, and is disabled by default.
- *max_time* – stop training after this amount of time has passed. This parameter is optional, and is disabled by default.
- *limit_train_batches* – how much of training data set to check. This parameter is optional, and the default value is 1.0.
- *limit_val_batches* – how much of validation data set to check. This parameter is optional, and the default value is 1.0.
- *limit_test_batches* – how much of test data set to check. This parameter is optional, and the default value is 1.0.
- *limit_predict_batches* – how much of prediction data set to check. This parameter is optional, and the default value is 1.0.
- *overfit_batches* – over-fit a fraction of training/validation data (float) or a set number of batches (integer). This parameter is optional, and the default value is 0.0.
- *val_check_interval* – how often to check the validation set. If it receives a float in the range [0.0, 1.0], it checks after a fraction of the training epoch. If an integer is passed, it checks after a fixed number of training batches. This parameter is optional, and the default value is 1.0.
- *check_val_every_n_epoch* – performs a validation loop every after every *N* training epochs. If set to None, validation will be done solely based on the number of training batches, requiring *val_check_interval* to be an integer value. This parameter is optional, and the default value is 1.0.
- *num_sanity_val_steps* – sanity check runs *n* validation batches before starting the training routine. This parameter is optional, and the default value is 2.
- *log_every_n_steps* – how often to log within steps. This parameter is optional, and the default value is 50.

- *enable_checkpointing* – if set to True, enables check-pointing. It will configure a default *ModelCheckpoint* callback if there is no user-defined *ModelCheckpoint* in callbacks. This parameter is optional, and the default value is True.
- *enable_progress_bar* – whether to enable to progress bar by default. This parameter is optional, and the default value is True.
- *enable_model_summary* – whether to enable model summarizing by default. This parameter is optional, and the default value is True.
- *accumulate_grad_batches* – accumulates gradients over *k* batches before stepping the optimizer. This parameter is optional, and the default value is 1.
- *gradient_clip_val* – the value at which to clip gradients. If set to None, disables gradient clipping. If using Automatic Mixed Precision (AMP), the gradients will be unscaled before. This parameter is optional, and the default value is None.
- *gradient_clip_algorithm* – the gradient clipping algorithm to use. If set to 'value', it will clip by value. If set to 'norm', it will clip by norm. This parameter is optional, and the default value is 'norm'.
- *deterministic* – if set to True, sets whether *PyTorch* operations must use deterministic algorithms. If set to 'warn', it uses deterministic algorithms whenever possible, throwing warnings on operations that don't support deterministic mode. This parameter is optional, and the default value is False.
- *benchmark* – the value (True or False) to set *torch.backends.cudnn.benchmark* to. The value for *torch.backends.cudnn.benchmark* set in the current session will be used (False if not manually set). If *deterministic* is set to True, this will default to False. This parameter is optional, and the default value is None.
- *inference_mode* – whether to use *torch.inference_mode()* or *torch.no_grad()* during evaluation (validate/test/predict). This parameter is optional, and the default value is True.
- *use_distributed_sampler* – whether to wrap the *DataLoader*'s sampler with *torch.utils.data.DistributedSampler*. If not specified, this is toggled automatically for strategies that require it. By default, it will add *shuffle=True* for the train sampler and *shuffle=False* for validation/test/predict samplers. To disable this logic, it can be set as False, and the own distributed sampler can be added in the *dataloader* hooks. If True and a distributed sampler was already added, Lightning will not replace the existing one. For iterable-style data sets, this is not automatically done. This parameter is optional, and the default value is True.
- *profiler* – to profile individual steps during training and assist in identifying bottlenecks. This parameter is optional, and the default value is None.

- *detect_anomaly* – Enable anomaly detection for the *autograd* engine. This parameter is optional, and the default value is False.
- *barebones* – Whether to run in '*barebones mode*', where all features that may impact raw speed are disabled. This is meant for analyzing the Trainer's overhead and is discouraged during regular training runs. The following parameters are deactivated: *enable_checkpointing*, *logger*, *enable_progress_bar*, *log_every_n_steps*, *enable_model_summary*, *num_sanity_val_steps*, *fast_dev_run*, *detect_anomaly*, *profiler*, *log()*, *log_dict()*. This parameter is optional, and the default value is False.
- *plugins* – plugins allow modification of core behavior like *ddp* and *amp*, and enable custom lightning plugins. This parameter is optional, and the default value is None.
- *sync_batchnorm* – synchronize batch norm layers between process groups/whole world. This parameter is optional, and the default value is False.
- *reload_dataloaders_every_n_epochs* – set to a non-negative integer to reload dataloaders every *n* epochs. This parameter is optional, and the default value is 0.
- *default_root_dir* – Default path for logs and weights when no *logger/ckpt_callback* passed. This parameter is optional, and the default value is *os.getcwd()*.

The *max_epochs* parameter was set to 10, the *enable_model_summary* parameter set to True, and the *gradient_clip_val* set to 0.1.

For the *callbacks* parameter, since there is no need to keep the training process in case the loss in the testing data set is less than 0.0001, an instance of *EarlyStopping* from the *Pytorch Lightning* [30] was passed. This instance receives the following parameters:

- *monitor* – quantity to be monitored. This parameter is optional, and the default value is '*val_loss*'.
- *min_delta* – minimum change in the monitored quantity to qualify as an improvement, i.e. an absolute change of less than *min_delta*, will count as no improvement. This parameter is optional, and the default value is 0.0.
- *patience* – number of validation epochs with no improvement after which training will be stopped. This parameter is optional, and the default value is 3.
- *verbose* – verbosity mode. This parameter is optional, and the default value is False.
- *mode* – one of '*auto*', '*min*', '*max*'. In '*min*' mode, training will stop when the quantity monitored has stopped decreasing; in '*max*' mode it will stop when the quantity monitored has stopped increasing; in '*auto*' mode, the direction is automatically inferred from the name of the monitored quantity. This parameter is optional, and the default value is '*auto*'.

- *strict* – whether to crash the training if the monitor is not found in the validation metrics. This parameter is optional, and the default value is True.

The parameters *monitor*, *min_delta*, *patience*, *verbose*, and *mode* were set with the values 'val_loss', 10^{-4} , 10, False and 'min', respectively. The *strict* parameter was left with the default value.

The *limit_train_batches* parameter was set to 50, the *enable_checkpointing* to True, and the *logger* to an instance of *TensorBoardLogger*, also from the *PyTorch Lightning* Python library.

This instance receives the following parameters:

- *save_dir* – the path to the save directory. This parameter is mandatory and has no default value.
- *name* – the experiment name. If it is set as an empty string, then no per-experiment sub-directory is used. This parameter is optional, and the default value is 'lightning_logs'.
- *version* – the experiment version. If the version is not specified, the logger inspects the save directory for existing versions and then automatically assigns the next available version. If it is a string, then it is used as the run-specific sub-directory name. This parameter is optional, and the default value is None.
- *log_graph* – adds the computational graph to *TensorBoard*. This parameter is optional, and its default value is False.
- *default_hp_metric* – enables a placeholder metric with key *hp_metric* when *log_hyperparams* is called without a metric (otherwise calls to *log_hyperparams* without a metric are ignored). This parameter is optional, and the default value is True.
- *prefix* – string to put at the beginning of metric keys. This parameter is optional, and the default value is an empty string.
- *sub_dir* – sub-directory to group *TensorBoard* logs. If a *sub_dir* argument is passed, then logs are saved in */save_dir/name/version/sub_dir/*. This parameter is optional, and the default value is None, which means that no sub-directory is used.
- ***kwargs* – additional keyword arguments. This parameter is optional and has no default value.

All the parameters were kept with their default value, except the *save_dir* parameter, which was set to the directory in which the Jupyter Notebook was running.

Finally, the model was trained on the DeepAR instance created, with the training and testing chunks, obtaining the best model.

The results obtained for the selected metrics, both for the tasks and for the incidents, can be seen in Chapter 6.

5.10 Prophet

The Prophet model was implemented with the usage of the Prophet implementation from a library with the same name as the model [63], which follows the *sklearn* library model API.

The implementation receives the following parameters, which are all optional:

- *growth* - a string that can take as values *'linear'*, *'logistic'*, or *'flat'*, which are used to specify a linear, logistic, or flat trend, respectively. This parameter is optional, and the default value is *'linear'*.
- *changepoints* - a list of dates with potential changepoints, which are automatically selected if they are not specified. This parameter is optional.
- *n_changepoints* - integer that represents the number of potential changepoints to include. Only used if the changepoints are automatically selected, in which case the *n_changepoints* potential changepoints are selected uniformly from the first *changepoint_range* proportion of the history. This parameter is optional, and the default value is 25.
- *changepoint_range* - float that represents the percentage of history in which trend changepoints will be estimated. This parameter is optional, and the default value is 0.8 (i.e., 80%) and is only used if the *changepoints* list is not specified.
- *yearly_seasonality* - this parameter indicates if the model should fit the yearly seasonality. It can take the values *'auto'*, True, False, or a number of Fourier terms to generate. This parameter is optional, and the default value is *'auto'*.
- *weekly_seasonality* - this parameter goes in the same direction as the previous one but is used to indicate if the model should fit the weekly seasonality. It can take the same values as the *yearly_seasonality*, and the default value is *'auto'* too. This parameter is optional.
- *daily_seasonality* - like the last two parameters, this parameter aims to indicate if the model should fit the daily seasonality. The possible values are the same as the last two, and the default value is also *'auto'*. This parameter is optional.
- *holidays* - is a data frame in Python with the holidays. It should contain a column with the holiday's name, a column with the date, and, optionally, a range of days surrounding the holiday, as well as the specification for the prior scale for that holiday. This parameter is optional, and the default value is None.
- *seasonality_mode* - represents the seasonality mode, which can either be *'additive'* (default) or *'multiplicative'*. This parameter is optional.
- *seasonality_prior_scale* - this parameter modules the strength of the seasonality model. It is a float where larger values allow the model to fit larger seasonal fluctuations, and smaller values dampen the seasonality. It can also be specified for individual seasonalities. This parameter is optional, and the default value is 10.0.

- *holidays_prior_scale* - this parameter is also a float that modules the strength of the holiday components model unless overridden in the *holidays*. This parameter is optional, and the default value is also 10.0.
- *changepoint_prior_scale* - this parameter is a float that modules the flexibility of the automatic changepoint selection. For larger values, more changepoints will be allowed, while the opposite happens for small values. This parameter is optional, and the default value is 0.05.
- *mcmc_samples* - this integer parameter makes the model do a full Bayesian inference with the specified number of MCMC samples if the value is greater than 0. If it is 0, the model will do MAP estimation. This parameter is optional, and the default value is 0.
- *interval_width* - this parameter is a float that represents the width of the uncertainty intervals provided for the forecast. If the value of *mcmc_samples* is zero, this will be only the uncertainty in the trend using the MAP estimate of the extrapolated generative model. If the value of *mcmc_samples* is greater than zero, this will be integrated over all model parameters, which will include uncertainty in seasonality. This parameter is optional, and the default value is 0.8.
- *uncertainty_samples* - this parameter is an integer that represents the number of simulated draws used to estimate uncertainty intervals. If this parameter is set to 0 or False, it will disable uncertainty estimation and speed up the calculation. This parameter is optional, and the default value is 1000.
- *stan_backend* - finally, this parameter is a String which can take the values defined in the *StanBackendEnum*. The default value is 'None', which will try to iterate over all available backends and find the working one. This parameter is optional.

All the parameters were kept with their default value, unless the *holidays* parameter, which was filled with a data frame containing the holidays and corresponding dates for the time interval of the data set.

The model expects a data set with a *ds* column and a *y* column to be trained with. The *ds* column should include all the available dates in the format date or date time, while the *y* should include the values known for the target variable.

However, more than these columns, in the implementation made for this Master Thesis, some regressors were also used.

The regressors in Prophet are the exogenous variables (or covariates), which, according to the model's specificity, can only be future covariates since they have to be known both at training and prediction time.

Thus, the future data set was the one used, and all the features other than the date and quantity were added as regressors.

No hyperparameter optimization was necessary since no parameter was used other than the *holidays*.

The results for each metric, prediction, and training interval, both for the tasks and incidents, can be seen in Chapter 6.

Chapter 6

Evaluation

To evaluate the performance of the used models, two approaches were used.

The first one, named "Historical Evaluation," refers to the common practice of using historical data to train and test each model. The division of the data by train and test was already described in Chapter 5.

The second one, named "Live Evaluation," refers to the usage of the whole data set to train the best-performing model, forecasting then for the supported time intervals and comparing the predicted values with the observed ones in the following days. Due to feasibility and time constraints, for this approach, it was only possible to evaluate the predictions in the time intervals of 7 and 30 days.

The "Live Evaluation" can be seen as a concrete case of the "Historical Evaluation", in which, for instance, the last 7 or 30 days' information is forgotten. Then, the remaining data set is used to train the best-performing model, and the results of the predictions made afterward are compared against the information that was forgotten in the beginning.

In the first two sections, the "Historical Evaluation" approach is explained, and the obtained results are presented, followed by the analysis of the best-performing models. Then, in the last section, the "Live Evaluation" approach is explained, and the results are analyzed.

For simplification purposes, only the best results are presented in this chapter. The complete results can be seen in the Appendix B.

6.1 Historical Evaluation

For the historical evaluation, the metrics used were the Mean Absolute Error (MAE), the Mean Squared Error (MSE), the Root Mean Squared Error (RMSE), and the Mean Absolute Percentage Error (MAPE).

The MAE calculates the average of the absolute difference between the predicted and the observed values. Its formula is the following

$$\frac{\sum_{i=1}^n |p_i - a_i|}{n},$$

where n is the number of observed/predicted values, p_i is the predicted value for time= i , and a_i is the actual value for time= i .

The MSE calculates the average of the power to two differences between the actual and predicted values. Its formula is the following

$$\frac{\sum_{i=1}^n (p_i - a_i)^2}{n},$$

where n , p_i , and a_i have the same meaning as in the MAE formula.

The RMSE is the root of the MSE, with its formula being the following

$$\sqrt{\frac{\sum_{i=1}^n (p_i - a_i)^2}{n}},$$

with n , p_i , and a_i having naturally the same meaning as for the MAE and MSE.

The MAPE is the average of the absolute percentage of error, with its formula being the following

$$\frac{\sum_{i=1}^n \left| \frac{a_i - p_i}{a_i} \right|}{n},$$

with n , p_i , and a_i having naturally the same meaning as for the MAE, MSE, and RMSE.

As mentioned in Chapter 5, for each model, the training and testing stages were repeated 60 times, 30 for the tasks and 30 for the incidents.

In the following subsections, each model's results are presented, and the best results for each type of ticket and the corresponding models' configuration are highlighted. The selection of the best result is based on the lowest MAE, which was the selected metric for the hyperparameter optimization. For simplification purposes, the tables of the best results for each model and data set do not include all the metrics but only the MAE and MAPE.

6.1.1 Lasso Regression

Starting with the Lasso Regression model and with the Tasks data set, table 6.1 contains the best results obtained for each forecasting interval. As it is possible to recognize, the best training interval differs depending on the prediction interval. The best values for alpha are either 0 or 0.01.

TRAIN. INT	PRED. INT	ALPHA	MAE	MAPE
30	7	0	0,098	2,068
90	30	0,01	0,093	2,348
60	60	0,01	0,092	2,148
All	90	0	0,11	2,530
7	365	0,01	0,102	2,138

Table 6.1: Best results obtained for the Lasso Regression model on the Tasks data set.

For the Incidents data set, table 6.2 contains the best results obtained for each forecasting interval. Here, the training intervals 30 and All are the best for the prediction intervals of 30, 90, and 60, 365, respectively. The best alpha values are either 0.01, 0.02, or 0.

TRAIN. INT	PRED. INT	ALPHA	MAE	MAPE
60	7	0,01	0,134	1,417
30	30	0,02	0,106	1,849
All	60	0	0,136	1,863
30	90	0,01	0,125	1,560
All	365	0	0,134	1,694

Table 6.2: Best results obtained for the Lasso Regression model on the Incidents data set.

6.1.2 Ridge Regression

For the Ridge Regression model and starting with the Tasks data set, table 6.3 contains the best results obtained for each forecasting interval. As it is possible to recognize from a quick analysis, the best alpha value is always 0.01, and the training interval of 30 days is the one that gives the best results for three of the five prediction intervals.

TRAIN. INT	PRED. INT	ALPHA	MAE	MAPE
30	7	0,01	0,098	2,068
90	30	0,01	0,098	2,483
60	60	0,01	0,097	2,257
30	90	0,01	0,099	2,279
30	365	0,01	0,106	2,225

Table 6.3: Best results obtained for the Ridge Regression model on the Tasks data set.

For the Incidents data set, table 6.4 contains the best results obtained for each forecasting interval. As it is possible to observe, again, the best value for alpha is 0.01, and both the training intervals 30 and All perform well for two prediction intervals each.

TRAIN. INT	PRED. INT	ALPHA	MAE	MAPE
7	7	0,01	0,139	1,466
30	30	0,01	0,122	2,127
All	60	0,01	0,136	1,863
30	90	0,01	0,136	1,698
All	365	0,01	0,134	1,694

Table 6.4: Best results obtained for the Ridge Regression model on the Incidents data set.

6.1.3 ARIMA

For the ARIMA model and starting with the Tasks data set, table 6.5 contains the best results obtained for each forecasting interval. The best training intervals differ for each prediction interval, as well as the values for P and D, which range from 1 to 2 and from 0 to 1, respectively. Only the best Q value is constant and equal to 1.

TRAIN. INT	PRED. INT	(P, D, Q)	MAE	MAPE
7	7	(1, 1, 1)	1,762	0,605
7	30	(2, 1, 1)	1,706	0,635
365	60	(1, 0, 1)	1,736	0,636
30	90	(2, 1, 1)	1,446	0,631
All	365	(1, 1, 1)	1,867	0,586

Table 6.5: Best results obtained for the ARIMA model on the Tasks data set.

For the Incidents data set, table 6.6 contains the best results obtained for each forecasting interval. Here, the best training interval changes quite often with the prediction interval, as well as the P, D, and Q values, which range from 1 to 2, 0 to 1, and 1 to 2, respectively.

TRAIN. INT	PRED. INT	(P, D, Q)	MAE	MAPE
365	7	(1, 0, 1)	12,571	0,658
7	30	(1, 1, 2)	8,1	0,495
60	60	(2, 1, 1)	10,6	0,884
90	90	(1, 0, 1)	10,544	0,770
365	365	(1, 0, 2)	9,584	0,588

Table 6.6: Best results obtained for the ARIMA model on the Incidents data set.

6.1.4 SARIMAX

For the SARIMAX model and starting with the Tasks data set, table 6.7 contains the best results obtained for each forecasting interval. Except for the prediction interval of 60 days, all the others perform best when trained with the whole data set. Also, the best values for P, D, Q, p, d, q, and s are constant.

TRAIN. INT	PRED. INT	(P, D, Q, p, d, q, s)	MAE	MAPE
All	7	(1, 0, 1, 1, 0, 1, 12)	1,929	0,507
All	30	(1, 0, 1, 1, 0, 1, 12)	1,594	0,589
90	60	(1, 0, 1, 1, 0, 1, 12)	1,576	0,671
All	90	(1, 0, 1, 1, 0, 1, 12)	1,485	0,638
All	365	(1, 0, 1, 1, 0, 1, 12)	1,741	0,717

Table 6.7: Best results obtained for the SARIMAX model on the Tasks data set.

For the Incidents data set, table 6.8 contains the best results obtained for each forecasting interval. As one can observe, while the values for P, D, Q, p, d, q, and s are constant, the same does not happen for the training intervals, which differ quite a lot depending on the prediction interval.

TRAIN. INT	PRED. INT	(P, D, Q, p, d, q, s)	MAE	MAPE
30	7	(1, 0, 1, 1, 0, 1, 12)	12,286	0,557
90	30	(1, 0, 1, 1, 0, 1, 12)	8,367	0,522
60	60	(1, 0, 1, 1, 0, 1, 12)	9,25	0,918
90	90	(1, 0, 1, 1, 0, 1, 12)	9,789	0,823
365	365	(1, 0, 1, 1, 0, 1, 12)	8,652	0,593

Table 6.8: Best results obtained for the SARIMAX model on the Incidents data set.

6.1.5 RNN

For the RNN model and starting with the Tasks data set, table 6.9 contains the best results obtained for each forecasting interval. Except for the prediction interval of 30 days, the best number of nodes is 25. Also, both the training intervals 60 and 90 days are the best-performing ones.

TRAIN. INT	PRED. INT	NODES	MAE	MAPE
90	7	25	82,096	2,569
60	30	50	20,629	0,570
90	60	25	33,324	0,793
60	90	25	89,63	2,144
365	365	25	113,263	1,329

Table 6.9: Best results obtained for the RNN model on the Tasks data set.

For the Incidents data set, table 6.10 contains the best results obtained for each forecasting interval. Unlike the Tasks data set, the best number of nodes differs depending on the prediction interval. Also, the best training interval is different from all the prediction intervals.

TRAIN. INT	PRED. INT	NODES	MAE	MAPE
All	7	25	1,089	11,485
90	30	50	1,956	34,231
7	60	25	1,793	24,492
60	90	25	1,921	24,016
30	365	50	0,255	3,227

Table 6.10: Best results obtained for the RNN model on the Incidents data set.

6.1.6 LSTM

For the LSTM model and starting with the Tasks data set, table 6.11 contains the best results obtained for each forecasting interval. The best number of nodes is always 25, and the whole data set is the best for training purposes in three out of five prediction intervals.

For the Incidents data set, table 6.12 contains the best results obtained for each forecasting interval. The best number of nodes is 50 for the shortest three prediction intervals, while for the

TRAIN. INT	PRED. INT	NODES	MAE	MAPE
365	7	25	18,568	0,581
30	30	25	19,434	0,537
All	60	25	23,064	0,549
All	90	25	23,598	0,565
All	365	25	84,813	0,995

Table 6.11: Best results obtained for the LSTM model on the Tasks data set.

remaining is 25. The training interval is quite irregular, with the 90-day interval being the best for the prediction intervals of 7 and 60 days.

TRAIN. INT	PRED. INT	NODES	MAE	MAPE
90	7	50	1522,153	0,992
7	30	50	783,127	0,776
90	60	50	1733,307	0,575
30	90	25	2138,954	0,645
365	365	25	2100,745	0,636

Table 6.12: Best results obtained for the LSTM model on the Incidents data set.

6.1.7 LSTMX

For the LSTMX model and starting with the Tasks data set, table 6.13 contains the best results obtained for each forecasting interval. For two prediction intervals, the best number of nodes is 50, while for the others is 25. The same happens for the training intervals, where the 60-day interval is the best for two of the five prediction intervals.

TRAIN. INT	PRED. INT	NODES	MAE	MAPE
60	7	25	19,605	0,614
365	30	50	20,619	0,570
90	60	25	24,948	0,594
60	90	25	22,862	0,547
30	365	50	50,418	0,592

Table 6.13: Best results obtained for the LSTMX model on the Tasks data set.

For the Incidents data set, table 6.14 contains the best results obtained for each forecasting interval. Except for the prediction interval of 30 days, all the others perform best with 25 nodes. The training interval is, however, always 90 days.

6.1.8 DeepAR

For the DeepAR model and starting with the Tasks data set, table 6.15 contains the best results obtained for each forecasting interval. For the shorter prediction intervals, the data from the last year gives better results, while the whole data set performs better for larger prediction intervals.

TRAIN. INT	PRED. INT	NODES	MAE	MAPE
90	7	25	688,698	0,449
90	30	50	595,729	0,590
90	60	25	2668,85	0,885
90	90	25	1626,501	0,491
90	365	25	1816,548	0,550

Table 6.14: Best results obtained for the LSTMX model on the Incidents data set.

TRAIN. INT	PRED. INT	MAE	MAPE
365	7	0,479	0,162
365	30	0,689	0,315
All	60	0,293	0,169
All	90	0,256	0,089
All	365	5,231	1,209

Table 6.15: Best results obtained for the DeepAR model on the Tasks data set.

For the Tasks data set, table 6.16 contains the best results obtained for each forecasting interval. Unlike the Tasks data set, the best training interval is irregular when compared to the prediction interval. For the predictions for the next 7, 60, and 90 days, the data from the last year gives better results, while for the other prediction intervals, the whole data set performs better.

TRAIN. INT	PRED. INT	MAE	MAPE
365	7	2,083	0,208
All	30	3,143	1,319
365	60	2,202	0,563
365	90	2,22	1,117
All	365	11,146	1,115

Table 6.16: Best results obtained for the DeepAR model on the Incidents data set.

6.1.9 Prophet

For the Prophet model and starting with the Tasks data set, table 6.17 contains the best results obtained for each forecasting interval. The best prediction intervals range between 90 and 365 days.

TRAIN. INT	PRED. INT	MAE	MAPE
365	7	1,91	0,603
365	30	1,618	0,612
90	60	1,569	0,622
90	90	1,496	0,626
365	365	1,79	0,576

Table 6.17: Best results obtained for the Prophet model on the Tasks data set.

For the Incidents data set, table 6.18 contains the best results obtained for each forecasting interval. Except for the prediction interval of 365 days, all the others perform best when trained with the data from the last 365 days.

TRAIN. INT	PRED. INT	MAE	MAPE
365	7	11,882	498,047
365	30	9,922	194,217
365	60	10,027	87,242
365	90	9,65	1089,358
60	365	9,194	2,486

Table 6.18: Best results obtained for the Prophet model on the Incidents data set.

6.2 Best models

In this section, for each prediction interval and for the tasks and incidents, the best models and the corresponding results are presented.

Starting with the Tasks data set, the Lasso Regression model was the best-performing for all the prediction intervals, except for the prediction interval of 90 days, in which the Ridge Regression performed better, as it is possible to see in the table 6.19.

The MAPE, if rounded to the nearest integer, is 2, which means that every day, the best-performing model might have a forecasting error of at most 2%, more or less. For the scope of this Master Thesis, this is a good result since the preparation and planning that needs to be made by the SAP Basis team at Infineon is not influenced by this short error margin.

TRAIN. INT	PRED. INT	MAE	MAPE	Model
30	7	0,098	2,068	LASSO
90	30	0,093	2,348	LASSO
60	60	0,092	2,148	LASSO
30	90	0,099	2,279	RIDGE
7	365	0,102	2,138	LASSO

Table 6.19: Best-performing models for the Tasks data set.

For the Incidents data set, the Lasso Regression model was the best-performing for all the prediction intervals, as it is possible to see in table 6.20.

In the same direction as for the Tasks, if the MAPE is rounded to the nearest integer, its value ranges from 1 to 2, which means that every day, the best-performing model might have a forecasting error of at most 1 or 2%, more or less. Again, this is a good result considering the scope of this Master Thesis.

TRAIN. INT	PRED. INT	MAE	MAPE	Model
60	7	0,134	1,417	LASSO
30	30	0,106	1,849	LASSO
All	60	0,136	1,863	LASSO
30	90	0,125	1,560	LASSO
All	365	0,134	1,694	LASSO

Table 6.20: Best-performing models for the Incidents data set.

6.3 Live Evaluation

For the "Live Evaluation" approach, the first step consisted of the creation of four data sets, two for the Tasks and another two for the Incidents, named Future Data Sets, which are presented in the first sub-section.

The second sub-section contains the analysis of the results obtained in this kind of evaluation approach.

6.3.1 Future Data Sets

The Future Data Sets consist of data sets with the same or part of the features that the Tasks and Incidents data set contain but without any value in the *QUANTITY* columns (target variable) of each. The dates of each data set are also future dates.

With these Future Data Sets, it is possible to forecast the number of tasks and incidents using the best-performing model.

Starting with the Tasks, the first data set is the one to be used in all the models that are only compatible with future covariates. It is obtained by creating all the combinations of dates (considering the desired prediction interval), unique ticket types, unique task types, and unique priorities.

These combinations are then augmented with the holidays and weekday features. This process results in a data set with seven columns: *DATE*, *WEEKDAY*, *HOLIDAYS*, *TICKET TYPE*, *TASK TYPE*, *PRIORITY*, *QUANTITY*.

The second data set is obtained by generating as many rows as the number of combinations of days in the prediction interval and unique task types, augmenting them with the weekday and holidays information.

This last data set contains five columns: *DATE*, *WEEKDAY*, *HOLIDAYS*, *TASK TYPE*, *QUANTITY*, and is used in all the models compatible with both past and future covariates.

For the Incidents, a similar process is applied. Starting with the data set that is used in the models that are compatible with only future covariates, it is generated from the combinations of the dates in the prediction interval, unique ticket types, unique priorities, unique urgency, and unique impact.

The resulting data set is then augmented with the inclusion of the holidays and weekday information. This data set contains the following columns: *DATE*, *WEEKDAY*, *HOLIDAY*, *TICKET TYPE*, *PRIORITY*, *URGENCY*, *IMPACT*, *QUANTITY*.

Finally, the second data set for the Incidents is obtained by just augmenting the dates for the prediction interval with the holidays and weekday information, resulting in a data set with four columns: *DATE*, *WEEKDAY*, *HOLIDAYS*, *QUANTITY*. This data set is used in all the models compatible with both past and future covariates.

Note that, both for the Tasks and Incidents data set, the information about the holidays is obtained using the following procedure:

1. For each row and for each of the observed *SITE* in the training data set, obtain the information regarding whether or not it is a public holiday in any of the locations and, simultaneously, in the current day.
2. The *HOLIDAYS* value will result in the sum of the number of holidays, i.e., the total number of holidays in all the observed *SITE* in the current day.

Since the best results were obtained with Lasso and Ridge Regressions, which are both incompatible with past covariates, the first data set for the Tasks and for the Incidents was used to make the predictions for future dates.

Considering that each of the used data sets contains more than one entry for each pair date-task type (tasks) and date (incidents), after the prediction, the data sets had to be grouped by date-task type (tasks) and by date (incidents) to compare the forecasted values against the observed ones.

6.3.2 Analysis of the results

Due to feasibility and time constraints, it was only possible to evaluate the predictions made for the time intervals of 7 and 30 days, i.e. use the data until the 31st of July of 2023 to predict the next 7 days (from the 1st of August of 2023 to the 7th of August of 2023) and the next 30 days (from the 1st of August of 2023 to the 30th of August of 2023).

For the prediction intervals mentioned above, considering the results presented before, the Lasso Regression was the best-performing model both for the Tasks and for the Incidents data set.

Starting with the tasks and with the shortest prediction interval (7 days), table 6.21 shows the results obtained.

MAE	MSE	RMSE	MAPE
0,935	0,874	0,935	0,93465

Table 6.21: "Live Evaluation" for the Tasks data set for the prediction interval of 7 days.

Still for the tasks, but for the prediction interval of 30 days, table 6.22 shows the results obtained.

MAE	MSE	RMSE	MAPE
0,953	0,908	0,953	0,95293

Table 6.22: "Live Evaluation" for the Tasks data set for the prediction interval of 30 days.

For the incidents and for the shortest prediction interval (7 days), table 6.23 shows the results obtained.

MAE	MSE	RMSE	MAPE
0,987	0,984	0,992	0,97638

Table 6.23: "Live Evaluation" for the Incidents data set for the prediction interval of 7 days.

For the incidents and for the prediction interval of 30 days, table 6.24 shows the results obtained.

MAE	MSE	RMSE	MAPE
0,996	1,001	1	0,98794

Table 6.24: "Live Evaluation" for the Incidents data set for the prediction interval of 30 days.

In terms of the MAE, the results obtained in the "Live Evaluation" are worse than those obtained in the "Historical Evaluation". However, comparing the MAPE, the results are better.

Rounding the MAPE to the nearest integer, both for the tasks and incidents and both for the prediction interval of 7 and 30 days, the error is 1%, more or less, which means that the prediction is accurate considering the scope of This Master Thesis.

Chapter 7

Conclusions and Future Work

In this Chapter, the satisfaction of the goals of this Master Thesis is discussed in the first section, while the second section contains the future work that was identified, both on the academic side of the Master Thesis and on the company (Infineon) side, including the next steps that are already planned.

7.1 Satisfaction of goals

Regarding the goals of this Master Thesis, all of them were successfully met.

Prior to the start of the development, the work and communication plan were elaborated, as well as the risk management and the technology decision matrix.

Then, the state-of-the-art research was made to Time Series, to its forecasting methodologies and techniques, as well as to well-known methods, from the traditional and simple to the most recent and advanced ones. This research also included the analysis of related projects, resulting in a complete description of the concepts and models.

Moving to the development, the first step was to collect the data and perform an exploratory analysis to assess both the data quality and its characteristics.

Then, the data preparation, which was one the longest steps in this Master Thesis, took place, with the complete preparation of the tasks and incidents data sets that included the data selection and cleaning, the data construction and its integration and formatting.

The resulting data sets were then used to train and test several models, using a distribution of the data between train and test unusual and not typical in this kind of project, but achieving good results, which were analyzed in depth.

The best-performing models (Lasso and Ridge Regressions) were then used to forecast tasks and incidents, feeding the built dashboard that shows the predictions in a user-friendly interface with some useful actions available to the end-user.

The main goal of this Master Thesis which was to build a system that included both a Time Series Forecasting model and a Dashboard to visualize the forecasts was achieved with success,

being now a useful tool to be used by the SAP Basis team at Infineon, supporting it in the process of handling tickets more effectively and proactively.

7.2 Future work

Regarding the future work, it is divided into two main areas: the academic side and the company (Infineon) side. Each of the following two sub-sections describes in detail the future work in these areas.

7.2.1 Academic future work

Starting with the academic future work, the work developed under this Master Thesis can be improved in the following two topics:

- Extend the usage of the hyperparameter optimization techniques in a way that more parameters are tested and evaluated.
- Train and test the data sets with more Time Series Forecasting models and, for the ones already in use, try different configurations, namely for the Neural Networks.

7.2.2 Company future work

On the company's future work side, the next steps consist of the following:

- Deploy the developed code in the company's cloud, making the dashboard accessible to the SAP Basis team.
- Automate the training and testing cycles on a regular basis so that the forecasts can be made with the best-performing model, which itself can be different from time to time, depending on the obtained results.
- Extend this solution to other areas/departments with similar use cases, adapting the system as needed.

References

- [1] Time series. In Stan Z. Li and Anil Jain, editors, *Encyclopedia of Biometrics*, pages 1340–1340, Boston, MA, 2009. Springer US.
- [2] What are recurrent neural networks?, 12 2021. Available at <https://databasecamp.de/en/ml/recurrent-neural-network>, last accessed in September 2023.
- [3] Long short-term memory networks (lstm)- simply explained!, 6 2022. Available at <https://databasecamp.de/en/ml/lstms>, last accessed in September 2023.
- [4] Brendan Artley. Time series forecasting with arima, sarima and sarimax, 4 2022. Available at <https://towardsdatascience.com/time-series-forecasting-with-arima-sarima-and-sarimax-ee61099e78f6>, last accessed in September 2023.
- [5] Neha Bora. Understanding arima models for machine learning, 11 2021. Available at <https://www.capitalone.com/tech/machine-learning/understanding-arima-models/>, last accessed in September 2023.
- [6] Q. Chi, M. Liu, and B. Yang. Prediction and analysis of air ticket based on arima model. *Advances in Intelligent Systems and Computing*, 1304:128–135, 2021.
- [7] Darts. Covariates in darts, 2023. Available at <https://unit8co.github.io/darts/userguide/covariates.html>, last accessed in September 2023.
- [8] Field Nathan David and Peter Whittle. Hypothesis-testing in time series analysis., 1952. Available at <https://api.semanticscholar.org/CorpusID:125371196>, last accessed in September 2023.
- [9] Arden Dertat. Applied deep learning - part 1: Artificial neural networks, 8 2017. Available at <https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6#106c>, last accessed in September 2023.
- [10] Cambridge Business English Dictionary. Cambridge dictionary - time series, 2023. Available at <https://dictionary.cambridge.org/dictionary/english/time-series>, last accessed in September 2023.
- [11] ELSEVIER. Mendeley, 2023. Available at <https://www.mendeley.com/>, last accessed in September 2023.
- [12] ELSEVIER. Scopus, 2023. Available at <https://www.scopus.com/home.uri>, last accessed in September 2023.

- [13] Stanley J. Farlow. The gmdh algorithm of ivakhnenko. *The American Statistician*, 35(4):210–215, 1981.
- [14] Hunt Steven Fitter, Fawn. Sap insights - how ai can end bias, July 2022. Available at <https://insights.sap.com/how-ai-can-end-bias/>, last accessed in September 2023.
- [15] PyTorch Forecasting. Pytorch forecasting deepar, 2023. Available at https://pytorch-forecasting.readthedocs.io/en/stable/api/pytorch_forecasting.models.deepar.DeepAR.html, last accessed in September 2023.
- [16] PyTorch Forecasting. Pytorch forecasting timeseriesdataset, 2023. Available at https://pytorch-forecasting.readthedocs.io/en/stable/api/pytorch_forecasting.data.timeseries.TimeSeriesDataSet.html, last accessed in September 2023.
- [17] Kunihiro Fukushima and Sei Miyake. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition*, 15:455–469, 1982.
- [18] Margherita Gerolimetto. Introduction to time series analysis, 2010. Available at <https://api.semanticscholar.org/CorpusID:17139013>, last accessed in September 2023.
- [19] A. Gordiievych and I. Shubin. Forecasting of airfare prices using time series. *2015 Information Technologies in Innovation Business Conference, ITIB 2015 - Proceedings*, pages 68–71, 2015.
- [20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.
- [21] Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 42(1):80–86, 2000.
- [22] IBM. Convolutional neural networks, 2023. Available at <https://www.ibm.com/topics/convolutional-neural-networks>, last accessed in September 2023.
- [23] IBM. What are neural networks?, 2023. Available at <https://www.ibm.com/topics/neural-networks>, last accessed in September 2023.
- [24] K. Imanaka and C. Sakama. Predicting air ticket demand using deep neural networks. *Proceedings - 2021 IEEE International Conference on Big Data, Big Data 2021*, pages 2901–2908, 2021.
- [25] Scikit learn. Sklearn labelencoder, 2023. Available at <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html>, last accessed in September 2023.
- [26] Scikit learn. Sklearn standardscaler, 2023. Available at <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>, last accessed in September 2023.
- [27] Yann Lecun, Leon Bottou, Y. Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278 – 2324, 12 1998.

- [28] Y. Li, Z. Li, and S. Qin. Ticket price forecasting based on machine learning. *2017 7th International Workshop on Computer Science and Engineering, WCSE 2017*, pages 465–469, 2017.
- [29] Yanping Liao and Chao Liang. A temperature time series forecasting model based on deepar. *2021 7th International Conference on Computer and Communications, ICC 2021*, pages 1588–1593, 2021.
- [30] PyTorch Lightning. Pytorch lightning earlystopping, 2023. Available at https://pytorch-lightning.readthedocs.io/en/0.9.0/api/pytorch_lightning.callbacks.early_stopping.html#pytorch_lightning.callbacks.early_stopping.EarlyStopping, last accessed in September 2023.
- [31] PyTorch Lightning. Pytorch lightning trainer, 2023. Available at <https://lightning.ai/docs/pytorch/stable/common/trainer.html#trainer-class-api>, last accessed in September 2023.
- [32] C. Maklin. Deepar forecasting algorithm, July 2015. Available at <https://medium.com/@corymaklin/deepar-forecasting-algorithm-6555efa63444>, last accessed in September 2023.
- [33] A. Mandal, S. Agarwal, N. Malhotra, G. Sridhara, A. Ray, and D. Swarup. Improving it support by enhancing incident management process with multi-modal analysis. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11895 LNCS:431–446, 2019.
- [34] James L. McClelland and David E. Rumelhart. Schemata and sequential thought processes in pdp models. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Psychological and Biological Models*, pages 7–57, 1987.
- [35] Warren S Mcculloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *BULLETIN OF MATHEMATICAL BIOPHYSICS*, 5, 1943.
- [36] Viktor Mehandzhiyski. What is a moving average model?, 10 2021. Available at <https://365datascience.com/tutorials/time-series-analysis-tutorials/moving-average-model/>, last accessed in September 2023.
- [37] Viktor Mehandzhiyski. What is an arma model?, 4 2023. Available at <https://365datascience.com/tutorials/time-series-analysis-tutorials/arma-model/>, last accessed in September 2023.
- [38] Viktor Mehandzhiyski. What is an autoregressive model?, 4 2023. Available at <https://365datascience.com/tutorials/time-series-analysis-tutorials/autoregressive-model/>, last accessed in September 2023.
- [39] A. Mustapha, S.A. Mostafa, M.H. Hassan, M.A. Jubair, S.H. Khaleefah, and M.H. Hassan. Machine learning supervised analysis for enhancing incident management process. *International Journal of Emerging Trends in Engineering Research*, 8:199–204, 2020.
- [40] Numpy. Numpy expand dimension, 2023. Available at https://numpy.org/doc/stable/reference/generated/numpy.expand_dims.html, last accessed in September 2023.

- [41] Ought. Elicit: The ai research assistant, 2023. Available at <https://elicit.org>, last accessed in September 2023.
- [42] J. Ranstam and J. A. Cook. Lasso regression. *British Journal of Surgery*, 105:1348, 9 2018.
- [43] Trygve Reenskaug. The model-view-controller (mvc) its past and present. 2003.
- [44] F Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain 1. *Psychological Review*, 65:19–27.
- [45] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, 2020.
- [46] SAP. Developing apps with sap fiori tools, 2023. Available at <https://sapui5.hana.ondemand.com/sdk/#/topic/a460a7348a6c431a8bd967ab9fb8d918>, last accessed in September 2023.
- [47] SAP. Sap, 2023. Available at <https://www.sap.com/about.html>, last accessed in September 2023.
- [48] SAP. Sapui5, 2023. Available at <https://community.sap.com/topics/ui5>, last accessed in September 2023.
- [49] SAP. Sapui5: Stacked columns, 2023. Available at <https://sapui5.hana.ondemand.com/#/entity/sap.viz.ui5.controls.VizFrame/sample/sap.viz.sample.StackedColumn/>, last accessed in September 2023.
- [50] scikit-learn developers. Metrics and scoring: Quantifying the quality of predictions, 2023. Available at https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter, last accessed in September 2023.
- [51] scikit-learn developers. Sklearn gridsearchcv, 2023. Available at https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html, last accessed in September 2023.
- [52] scikit-learn developers. Sklearn kfold implementation, 2023. Available at https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html, last accessed in September 2023.
- [53] scikit-learn developers. Sklearn lasso implementation, 2023. Available at https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html, last accessed in September 2023.
- [54] scikit-learn developers. Sklearn ridge implementation, 2023. Available at https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html, last accessed in September 2023.
- [55] scikit-learn developers. Sklearn stratifiedkfold implementation, 2023. Available at https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html, last accessed in September 2023.
- [56] scipy developers. Scipy cg implementation, 2023. Available at <https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.cg.html>, last accessed in September 2023.

- [57] scipy developers. Scipy lsqr implementation, 2023. Available at <https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.lsqr.html>, last accessed in September 2023.
- [58] scipy developers. Scipy minimize implementation, 2023. Available at <https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html>, last accessed in September 2023.
- [59] scipy developers. Scipy solve implementation, 2023. Available at <https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.solve.html>, last accessed in September 2023.
- [60] Amazon Web Services. Deepar forecasting algorithm, 2023. Available at <https://docs.aws.amazon.com/sagemaker/latest/dg/deepar.html>, last accessed in September 2023.
- [61] Sagar Sharma. What the hell is perceptron?, 9 2017. Available at <https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53>, last accessed in September 2023.
- [62] M. Sodanil and P. Chatthong. Artificial neural network-based time series analysis forecasting for the amount of solid waste in bangkok. *2014 9th International Conference on Digital Information Management, ICDIM 2014*, pages 16–20, 2014.
- [63] Facebook Open Source. Prophet, 2023. Available at https://facebook.github.io/prophet/docs/quick_start.html, last accessed in September 2023.
- [64] Statistics.com. Autoregression and moving average (arma) models, 2023. Available at <https://www.statistics.com/glossary/autoregression-and-moving-average-arma-models/>, last accessed in September 2023.
- [65] statsmodels. Arima statsmodels, 2023. Available at <https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima.model.ARIMA.html>, last accessed in September 2023.
- [66] statsmodels. Sarimax statsmodels, 2023. Available at <https://www.statsmodels.org/stable/generated/statsmodels.tsa.statespace.sarimax.SARIMAX.html>, last accessed in September 2023.
- [67] Sean J. Taylor and Benjamin Letham. Forecasting at scale. *PeerJ Preprints*, 5, 9 2017.
- [68] Facebook’s Core Data Science team. Prophet, 2023. Available at <https://facebook.github.io/prophet/>, last accessed in September 2023.
- [69] tensorflow. Tensorflow, 2023. Available at https://www.tensorflow.org/api_docs/python/tf, last accessed in September 2023.
- [70] tensorflow. Tensorflow dense, 2023. Available at https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense, last accessed in September 2023.
- [71] TensorFlow. Tensorflow lstm, 2023. Available at https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM, last accessed in September 2023.

- [72] TensorFlow. Tensorflow model compile, 2023. Available at https://www.tensorflow.org/api_docs/python/tf/keras/Model#compile, last accessed in September 2023.
- [73] tensorflow. Tensorflow sequential, 2023. Available at https://www.tensorflow.org/api_docs/python/tf/keras/Sequential, last accessed in September 2023.
- [74] tensorflow. Tensorflow simplernn, 2023. Available at https://www.tensorflow.org/api_docs/python/tf/keras/layers/SimpleRNN, last accessed in September 2023.
- [75] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58:267–288, 1996.
- [76] S.A. Yadwad, V.V. Kumari, and S.V. Lakshmi. Service outages prediction through logs and tickets analysis. *International Journal of Advanced Computer Science and Applications*, 12:177–183, 2021.
- [77] J.H. Zhao, Z. Dong, and Z. Xu. Effective feature preprocessing for time series forecasting. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4093 LNAI:769–781, 2006.

Appendix A

Preliminary Work

A.1 Work Plan

Considering all the deliverables and important dates of this Master Thesis, a Gantt diagram was elaborated prior to the start of the work to highlight the main tasks and corresponding duration, as shown in the figure A.1.

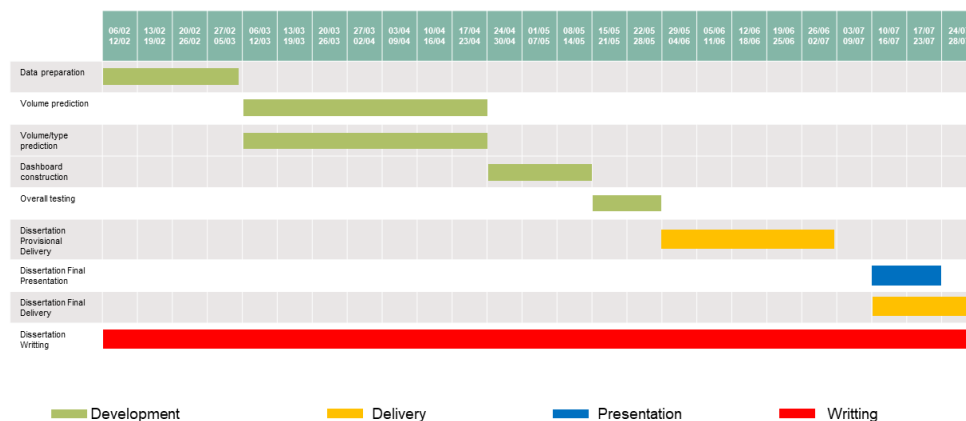


Figure A.1: Gantt diagram for this Master Thesis.

The data preparation, which includes data collection, data pre-processing, and data transformation, was the first task of this work, with a planned duration of four weeks. It was then followed by the planned seven weeks to work on the AI model.

Then, another three weeks were planned to be used on the building of the dashboard, followed by the overall testing of the AI model and the dashboard, which was planned to took the last two weeks before the provisional delivery.

The dissertation writing took place during the entire duration of this Master Thesis, finishing with the final delivery.

The plan suffered some changes during the course of the Master Thesis due to numerous factors.

A.2 Communication Plan

For this Master Thesis, considering it was developed within a company environment (Infineon) and had to follow some well-established stages designed for projects of this kind, a communication plan was also created. This plan established the periodicity, type, and elements of regular meetings to be carried out during the entire Master Thesis duration.

The table A.1 shows the three main regular meetings planned. The daily stand-up meeting was aimed at the discussion of potential issues or delays. The sprint planning took place every two weeks to plan the next activities to be included in the upcoming sprint. The general update was a meeting with the FEUP Supervisor, Prof. Luís Paulo Reis, whose main goal was to share progress and upcoming milestones, being held every 2 or 3 weeks, depending on the needs.

Who	Why	What	When	How	By Whom	Level of communication detail
Developer (João Pires)	Discuss potential issues or delays	Daily stand-up	Daily	Webex meeting	Scrum Master	Help/Support needed
Developer (João Pires) + Project Owner	Sprint planning	Plan next activities	Every 2 weeks	Face-to-face meeting	Scrum Master	Task level
Supervisors (FEUP and Infineon)	Share progress and upcoming milestones	General update	Every 2 or 3 weeks	Zoom meeting	Developer (João Pires)	General status, achievement, risk and next steps

Table A.1: Communication plan.

A.3 Risk Management

The risk management was an activity made during the Master Thesis definition and includes the risks that might have occurred during the development, as well as the corresponding probability and impact.

The table A.2 presents the list of risks identified, the cause and effect of each one, as well as the probability of happening and the impact it could have on the overall work.

Both the probability and impact were classified on a scale with three levels: Low, Medium, and High.

Risk	Cause	Effect	Probability	Impact
Data quality	Data with missing information (null values) and information not updated	AI model predictions affected by the poor quality of data	Low	High
Accuracy of predictions affected by incomplete data	Incomplete access to data due to authorizations	Might lead to accuracy of AI model being low	Low	Medium
Model training is slower than the data update cycle	Data is updated faster than the time required to train the model	AI model is not always up-to-date, which might affect the prediction	Low	Low

Table A.2: Risk management.

Appendix B

Evaluation

TRAIN. INT	PRED. INT	ALPHA	MAE	MSE	RMSE	MAPE
All	7	0	0,112	0,037	0,193	2,356
7	7	0	0,117	0,046	0,213	2,457
30	7	0	0,098	0,038	0,194	2,068
60	7	0	0,101	0,037	0,193	2,132
90	7	0	0,102	0,037	0,193	2,135
365	7	0	0,106	0,037	0,193	2,219
All	30	0	0,107	0,033	0,181	2,709
7	30	0,01	0,168	0,047	0,216	4,259
30	30	0	0,107	0,033	0,182	2,715
60	30	0	0,101	0,032	0,18	2,552
90	30	0,01	0,093	0,032	0,18	2,348
365	30	0	0,101	0,033	0,18	2,556
All	60	0	0,109	0,035	0,187	2,543
7	60	0,01	0,123	0,038	0,195	2,868
30	60	0,01	0,093	0,035	0,186	2,171
60	60	0,01	0,092	0,035	0,187	2,148
90	60	0,01	0,094	0,035	0,186	2,184
365	60	0	0,104	0,035	0,186	2,413
All	90	0	0,11	0,035	0,188	2,530
7	90	0,01	0,437	0,278	0,527	10,036
30	90	0,01	0,091	0,037	0,192	2,086
60	90	0,01	0,094	0,035	0,187	2,161
90	90	0,01	0,096	0,035	0,187	2,208
365	90	0	0,105	0,035	0,187	2,422
All	365	0	0,114	0,038	0,194	2,401
7	365	0,01	0,102	0,038	0,194	2,138
30	365	0,01	0,104	0,037	0,194	2,184
60	365	0,01	0,103	0,038	0,196	2,173
90	365	0	0,111	0,038	0,195	2,333
365	365	0	0,114	0,038	0,194	2,391

Table B.1: Results obtained for the Lasso Regression model on the Tasks data set.

TRAIN. INT	PRED. INT	ALPHA	MAE	MSE	RMSE	MAPE
All	7	0	0,154	0,077	0,278	1,621
7	7	0	0,139	0,08	0,282	1,465
30	7	0	0,142	0,08	0,283	1,499
60	7	0,01	0,134	0,081	0,284	1,417
90	7	0	0,151	0,078	0,279	1,590
365	7	0	0,161	0,077	0,277	1,697
All	30	0	0,124	0,048	0,219	2,165
7	30	0,01	0,115	0,048	0,218	2,010
30	30	0,02	0,106	0,05	0,223	1,849
60	30	0	0,132	0,049	0,22	2,312
90	30	0	0,142	0,049	0,222	2,476
365	30	0	0,135	0,049	0,22	2,359
All	60	0	0,136	0,061	0,247	1,863
7	60	0,02	0,172	0,093	0,304	2,356
30	60	0	0,142	0,063	0,251	1,940
60	60	0	0,159	0,062	0,25	2,165
90	60	0	0,15	0,061	0,248	2,055
365	60	0	0,146	0,061	0,248	1,998
All	90	0	0,141	0,065	0,256	1,758
7	90	0	0,278	0,15	0,387	3,472
30	90	0,01	0,125	0,071	0,266	1,560
60	90	0	0,136	0,067	0,258	1,702
90	90	0	0,142	0,066	0,256	1,774
365	90	0	0,149	0,065	0,256	1,868
All	365	0	0,134	0,063	0,251	1,694
7	365	0,01	1,569	3,321	1,822	19,813
30	365	0	0,15	0,065	0,255	1,894
60	365	0	0,153	0,062	0,249	1,932
90	365	0	0,175	0,065	0,254	2,215
365	365	0,01	0,148	0,063	0,252	1,864

Table B.2: Results obtained for the Lasso Regression model on the Incidents data set.

TRAIN. INT	PRED. INT	ALPHA	MAE	MSE	RMSE	MAPE
All	7	0,01	0,112	0,037	0,193	2,356
7	7	0,01	0,102	0,039	0,198	2,153
30	7	0,01	0,098	0,038	0,194	2,068
60	7	0,01	0,101	0,037	0,193	2,132
90	7	0,01	0,102	0,037	0,193	2,134
365	7	0,01	0,106	0,037	0,193	2,219
All	30	0,01	0,107	0,033	0,181	2,709
7	30	0,01	0,192	0,055	0,234	4,871
30	30	0,01	0,107	0,033	0,182	2,715
60	30	0,01	0,101	0,032	0,18	2,552
90	30	0,01	0,098	0,032	0,18	2,483
365	30	0,01	0,101	0,033	0,18	2,556
All	60	0,01	0,109	0,035	0,187	2,543
7	60	0,01	0,17	0,049	0,221	3,954
30	60	0,01	0,099	0,035	0,186	2,301
60	60	0,01	0,097	0,035	0,186	2,257
90	60	0,01	0,099	0,035	0,186	2,305
365	60	0,01	0,104	0,035	0,186	2,413
All	90	0,01	0,11	0,035	0,188	2,530
7	90	0,01	0,52	0,378	0,615	11,937
30	90	0,01	0,099	0,038	0,195	2,279
60	90	0,01	0,1	0,035	0,187	2,285
90	90	0,01	0,102	0,035	0,187	2,336
365	90	0,01	0,105	0,035	0,187	2,422
All	365	0,01	0,114	0,038	0,194	2,401
7	365	0,01	0,908	1,088	1,043	19,103
30	365	0,01	0,106	0,038	0,196	2,225
60	365	0,01	0,109	0,039	0,197	2,302
90	365	0,01	0,111	0,038	0,195	2,333
365	365	0,01	0,114	0,038	0,194	2,391

Table B.3: Results obtained for the Ridge Regression model on the Tasks data set.

TRAIN. INT	PRED. INT	ALPHA	MAE	MSE	RMSE	MAPE
All	7	0,01	0,154	0,077	0,278	1,621
7	7	0,01	0,139	0,079	0,282	1,466
30	7	0,01	0,142	0,08	0,283	1,499
60	7	0,01	0,142	0,08	0,282	1,501
90	7	0,01	0,151	0,078	0,279	1,590
365	7	0,01	0,161	0,077	0,277	1,697
All	30	0,01	0,124	0,048	0,219	2,165
7	30	0,01	0,132	0,048	0,219	2,315
30	30	0,01	0,122	0,05	0,223	2,127
60	30	0,01	0,132	0,049	0,22	2,312
90	30	0,01	0,141	0,049	0,222	2,476
365	30	0,01	0,135	0,049	0,22	2,359
All	60	0,01	0,136	0,061	0,247	1,863
7	60	0,01	0,406	0,26	0,51	5,541
30	60	0,01	0,142	0,063	0,251	1,939
60	60	0,01	0,159	0,062	0,25	2,165
90	60	0,01	0,15	0,061	0,248	2,055
365	60	0,01	0,146	0,061	0,248	1,998
All	90	0,01	0,141	0,065	0,256	1,758
7	90	0,01	0,272	0,146	0,382	3,407
30	90	0,01	0,136	0,072	0,268	1,698
60	90	0,01	0,136	0,067	0,258	1,701
90	90	0,01	0,142	0,066	0,256	1,774
365	90	0,01	0,149	0,065	0,256	1,868
All	365	0,01	0,134	0,063	0,251	1,694
7	365	0,01	2,511	8,415	2,901	31,707
30	365	0,01	0,15	0,065	0,255	1,894
60	365	0,01	0,153	0,062	0,249	1,932
90	365	0,01	0,175	0,065	0,254	2,214
365	365	0,01	0,151	0,062	0,25	1,911

Table B.4: Results obtained for the Ridge Regression model on the Incidents data set.

TRAIN. INT	PRED. INT	(P, D, Q)	MAE	MSE	RMSE	MAPE
All	7	(1, 0, 1)	1,952	6,381	2,526	0,631
7	7	(1, 1, 1)	1,762	7,571	2,752	0,687
30	7	(2, 0, 2)	1,857	6,714	2,591	0,647
60	7	(1, 1, 2)	1,905	6,333	2,517	0,629
90	7	(1, 1, 2)	1,905	6,333	2,517	0,629
365	7	(1, 0, 1)	1,952	6,381	2,526	0,631
All	30	(1, 0, 1)	1,718	5,153	2,27	0,635
7	30	(2, 1, 1)	1,706	5,118	2,262	0,633
30	30	(1, 0, 1)	1,718	5,153	2,27	0,635
60	30	(1, 0, 1)	1,718	5,153	2,27	0,635
90	30	(1, 0, 1)	1,718	5,153	2,27	0,635
365	30	(1, 0, 1)	1,718	5,153	2,27	0,635
All	60	(1, 0, 1)	1,739	5,587	2,364	0,636
7	60	(1, 0, 1)	1,739	5,587	2,364	0,636
30	60	(1, 1, 1)	1,739	5,587	2,364	0,636
60	60	(1, 0, 1)	1,739	5,587	2,364	0,636
90	60	(1, 0, 1)	1,739	5,587	2,364	0,636
365	60	(1, 0, 1)	1,736	5,579	2,362	0,636
All	90	(1, 0, 1)	1,645	4,822	2,196	0,605
7	90	(1, 0, 1)	1,447	5,262	2,294	0,632
30	90	(2, 1, 1)	1,446	5,26	2,294	0,632
60	90	(1, 1, 1)	1,447	5,262	2,294	0,632
90	90	(1, 0, 1)	1,645	4,822	2,196	0,605
365	90	(1, 0, 1)	1,645	4,822	2,196	0,605
All	365	(1, 1, 1)	1,867	7,109	2,666	0,621
7	365	(1, 0, 1)	1,867	7,118	2,668	0,621
30	365	(2, 0, 2)	1,867	7,118	2,668	0,621
60	365	(1, 0, 1)	1,867	7,109	2,666	0,621
90	365	(1, 0, 1)	1,867	7,109	2,666	0,621
365	365	(1, 1, 1)	1,867	7,109	2,666	0,621

Table B.5: Results obtained for the ARIMA model on the Tasks data set.

TRAIN. INT	PRED. INT	(P, D, Q)	MAE	MSE	RMSE	MAPE
All	7	(1, 1, 1)	13	393	19,824	0,495
7	7	(2, 0, 1)	14,429	535	23,13	0,577
30	7	(1, 0, 2)	15	508,714	22,555	0,563
60	7	(1, 0, 2)	13	392,143	19,803	0,494
90	7	(1, 0, 1)	12,857	368,571	19,198	0,479
365	7	(1, 0, 1)	12,571	356	18,868	0,471
All	30	(1, 0, 1)	9	156,533	12,511	0,590
7	30	(1, 1, 2)	8,1	159,567	12,632	0,595
30	30	(1, 1, 2)	12,2	222,667	14,922	0,703
60	30	(1, 0, 1)	12,933	242,667	15,578	0,734
90	30	(1, 1, 2)	11,1	195,3	13,975	0,659
365	30	(2, 1, 2)	10,867	191,333	13,832	0,652
All	60	(1, 0, 1)	10,65	496,117	22,274	0,537
7	60	(2, 1, 2)	10,65	512,25	22,633	0,546
30	60	(1, 1, 2)	10,933	499	22,338	0,539
60	60	(2, 1, 1)	10,6	495,867	22,268	0,537
90	60	(1, 0, 2)	11,45	488,117	22,093	0,533
365	60	(1, 1, 1)	11,8	489,467	22,124	0,534
All	90	(1, 1, 1)	10,622	386,244	19,653	0,488
7	90	(2, 0, 1)	10,789	418,456	20,456	0,507
30	90	(2, 0, 2)	10,6	401,156	20,029	0,497
60	90	(1, 0, 1)	10,611	386,056	19,648	0,487
90	90	(1, 0, 1)	10,544	375,856	19,387	0,481
365	90	(1, 1, 2)	10,633	386,722	19,665	0,488
All	365	(2, 1, 1)	9,764	214,701	14,653	0,453
7	365	(2, 1, 1)	13,452	285,452	16,895	0,522
30	365	(2, 0, 2)	10,507	220,595	14,852	0,459
60	365	(1, 0, 2)	10,501	220,129	14,837	0,458
90	365	(1, 1, 1)	10,181	216,4	14,711	0,454
365	365	(1, 0, 2)	9,584	225,216	15,007	0,464

Table B.6: Results obtained for the ARIMA model on the Incidents data set.

TRAIN. INT	PRED. INT	(P, D, Q, p, d, q, s)	MAE	MSE	RMSE	MAPE
All	7	(1, 0, 1, 1, 0, 1, 12)	1,929	6,833	2,614	0,638
7	7	(1, 0, 1, 1, 0, 1, 12)	2,214	10,595	3,255	0,795
30	7	(1, 0, 1, 1, 0, 1, 12)	1,952	7,095	2,664	0,651
60	7	(1, 0, 1, 1, 0, 1, 12)	1,976	7,119	2,668	0,652
90	7	(1, 0, 1, 1, 0, 1, 12)	1,952	6,905	2,628	0,642
365	7	(1, 0, 1, 1, 0, 1, 12)	1,952	6,905	2,628	0,642
All	30	(1, 0, 1, 1, 0, 1, 12)	1,594	5,065	2,25	0,589
7	30	(1, 0, 1, 1, 0, 1, 12)	2,012	8,8	2,966	0,777
30	30	(1, 0, 1, 1, 0, 1, 12)	1,671	5,165	2,273	0,595
60	30	(1, 0, 1, 1, 0, 1, 12)	1,653	5,053	2,248	0,589
90	30	(1, 0, 1, 1, 0, 1, 12)	1,594	5,053	2,248	0,589
365	30	(1, 0, 1, 1, 0, 1, 12)	1,594	5,041	2,245	0,588
All	60	(1, 0, 1, 1, 0, 1, 12)	1,601	5,361	2,315	0,591
7	60	(1, 0, 1, 1, 0, 1, 12)	1,739	5,554	2,357	0,602
30	60	(1, 0, 1, 1, 0, 1, 12)	1,647	6,038	2,457	0,628
60	60	(1, 0, 1, 1, 0, 1, 12)	1,592	6,141	2,478	0,633
90	60	(1, 0, 1, 1, 0, 1, 12)	1,576	5,516	2,349	0,600
365	60	(1, 0, 1, 1, 0, 1, 12)	1,641	5,266	2,295	0,586
All	90	(1, 0, 1, 1, 0, 1, 12)	1,485	4,993	2,234	0,548
7	90	(1, 0, 1, 1, 0, 1, 12)	2,112	9,271	3,045	0,748
30	90	(1, 0, 1, 1, 0, 1, 12)	1,879	8,007	2,83	0,695
60	90	(1, 0, 1, 1, 0, 1, 12)	1,499	4,865	2,206	0,542
90	90	(1, 0, 1, 1, 0, 1, 12)	1,578	4,654	2,157	0,530
365	90	(1, 0, 1, 1, 0, 1, 12)	1,572	4,624	2,15	0,528
All	365	(1, 0, 1, 1, 0, 1, 12)	1,741	7,42	2,724	0,583
7	365	(1, 0, 1, 1, 0, 1, 12)	2,306	12,321	3,51	0,751
30	365	(1, 0, 1, 1, 0, 1, 12)	1,988	9,51	3,084	0,660
60	365	(1, 0, 1, 1, 0, 1, 12)	2,019	9,948	3,154	0,675
90	365	(1, 0, 1, 1, 0, 1, 12)	1,75	7,958	2,821	0,604
365	365	(1, 0, 1, 1, 0, 1, 12)	1,778	7,022	2,65	0,567

Table B.7: Results obtained for the SARIMAX model on the Tasks data set.

TRAIN. INT	PRED. INT	(P, D, Q, p, d, q, s)	MAE	MSE	RMSE	MAPE
All	7	(1, 0, 1, 1, 0, 1, 12)	13,714	537,429	23,183	0,522
7	7	(1, 0, 1, 1, 0, 1, 12)	20,714	705,286	26,557	0,598
30	7	(1, 0, 1, 1, 0, 1, 12)	12,286	427,429	20,674	0,465
60	7	(1, 0, 1, 1, 0, 1, 12)	20,571	633,429	25,168	0,566
90	7	(1, 0, 1, 1, 0, 1, 12)	15,286	456,714	21,371	0,481
365	7	(1, 0, 1, 1, 0, 1, 12)	15,571	613	24,759	0,557
All	30	(1, 0, 1, 1, 0, 1, 12)	10,3	187,433	13,691	0,675
7	30	(1, 0, 1, 1, 0, 1, 12)	17,667	497,733	22,31	1,099
30	30	(1, 0, 1, 1, 0, 1, 12)	15,633	360,1	18,976	0,935
60	30	(1, 0, 1, 1, 0, 1, 12)	12,067	283,6	16,84	0,830
90	30	(1, 0, 1, 1, 0, 1, 12)	8,367	168,567	12,983	0,640
365	30	(1, 0, 1, 1, 0, 1, 12)	8,4	161,133	12,694	0,626
All	60	(1, 0, 1, 1, 0, 1, 12)	11,25	515,917	22,714	0,568
7	60	(1, 0, 1, 1, 0, 1, 12)	16,467	559,667	23,657	0,591
30	60	(1, 0, 1, 1, 0, 1, 12)	11,283	491,55	22,171	0,554
60	60	(1, 0, 1, 1, 0, 1, 12)	9,25	521,35	22,833	0,571
90	60	(1, 0, 1, 1, 0, 1, 12)	10,2	512,1	22,63	0,566
365	60	(1, 0, 1, 1, 0, 1, 12)	10,15	516,883	22,735	0,568
All	90	(1, 0, 1, 1, 0, 1, 12)	9,878	422,722	20,56	0,453
7	90	(1, 0, 1, 1, 0, 1, 12)	16,878	627,033	25,041	0,552
30	90	(1, 0, 1, 1, 0, 1, 12)	10,522	455,122	21,334	0,470
60	90	(1, 0, 1, 1, 0, 1, 12)	9,889	422,4	20,552	0,453
90	90	(1, 0, 1, 1, 0, 1, 12)	9,789	419,344	20,478	0,452
365	90	(1, 0, 1, 1, 0, 1, 12)	10,478	458,1	21,403	0,472
All	365	(1, 0, 1, 1, 0, 1, 12)	8,94	195,668	13,988	0,414
7	365	(1, 0, 1, 1, 0, 1, 12)	34,951	1824,863	42,718	1,266
30	365	(1, 0, 1, 1, 0, 1, 12)	11,575	262,808	16,211	0,480
60	365	(1, 0, 1, 1, 0, 1, 12)	10,329	218,323	14,776	0,438
90	365	(1, 0, 1, 1, 0, 1, 12)	9,803	206,734	14,378	0,426
365	365	(1, 0, 1, 1, 0, 1, 12)	8,652	217,677	14,754	0,437

Table B.8: Results obtained for the SARIMAX model on the Incidents data set.

TRAIN. INT	PRED. INT	NODES	MAE	MSE	RMSE	MAPE
All	7	25	456,007	208631,774	456,762	14,271
7	7	50	418,367	175667,923	419,128	13,093
30	7	50	191,799	37455,305	193,534	6,003
60	7	25	250,448	63598,774	252,188	7,838
90	7	25	82,096	7547,159	86,874	2,569
365	7	25	306,878	94917,849	308,087	9,604
All	30	50	111,502	13382,758	115,684	3,082
7	30	50	1323,72	1753148,81	1324,065	36,591
30	30	25	148,762	23323,026	152,719	4,112
60	30	50	20,629	875,605	29,591	0,570
90	30	25	338,821	116091,71	340,722	9,366
365	30	25	75,298	6505,58	80,657	2,081
All	60	50	40,519	2178,897	46,679	0,965
7	60	50	472,592	224581,484	473,9	11,251
30	60	25	744,85	556154,381	745,758	17,732
60	60	25	37,443	2796,348	52,881	0,891
90	60	25	33,324	1573,133	39,663	0,793
365	60	50	192,801	38373,868	195,892	4,590
All	90	50	247,172	62267,507	249,535	5,913
7	90	50	1003,232	1008224,245	1004,104	24,000
30	90	25	106,172	12854,116	113,376	2,540
60	90	25	89,63	9108,159	95,437	2,144
90	90	25	288,704	86003,3	293,263	6,906
365	90	25	327,839	108771,29	329,805	7,843
All	365	50	475,609	231063,644	480,691	5,582
7	365	25	5025,823	25266529,31	5026,582	58,990
30	365	25	1022,777	1054466,693	1026,872	12,005
60	365	50	1881,818	3550357,031	1884,239	22,087
90	365	25	620,884	391909,844	626,027	7,287
365	365	25	113,263	15236,1	123,435	1,329

Table B.9: Results obtained for the RNN model on the Tasks data set.

TRAIN. INT	PRED. INT	NODES	MAE	MSE	RMSE	MAPE
All	7	25	1,089	1,272	1,128	11,485
7	7	50	9,835	96,819	9,84	103,698
30	7	50	29,151	849,888	29,153	307,357
60	7	25	8,174	66,904	8,18	86,185
90	7	25	8,239	67,969	8,244	86,869
365	7	50	7,369	54,395	7,375	77,700
All	30	25	3,537	12,561	3,544	61,889
7	30	25	3,348	11,266	3,356	58,598
30	30	25	8,836	78,13	8,839	154,632
60	30	50	7,536	56,851	7,54	131,886
90	30	50	1,956	3,88	1,97	34,231
365	30	25	12,006	144,208	12,009	210,113
All	60	25	5,329	28,463	5,335	72,778
7	60	25	1,793	3,284	1,812	24,492
30	60	25	11,085	122,956	11,089	151,402
60	60	25	5,403	29,255	5,409	73,786
90	60	50	11,125	123,824	11,128	151,936
365	60	25	7,249	52,617	7,254	99,005
All	90	50	7,424	55,191	7,429	92,822
7	90	25	3,994	16,025	4,003	49,934
30	90	25	6,36	40,52	6,366	79,515
60	90	25	1,921	3,763	1,94	24,016
90	90	50	12,187	148,602	12,19	152,374
365	90	50	9,3	86,556	9,304	116,271
All	365	25	4,73	22,451	4,738	59,743
7	365	50	4,66	21,792	4,668	58,857
30	365	50	0,255	0,138	0,372	3,227
60	365	50	6,051	36,683	6,057	76,416
90	365	25	2,959	8,831	2,972	37,375
365	365	25	9,396	88,363	9,4	118,668

Table B.10: Results obtained for the RNN model on the Incidents data set.

TRAIN. INT	PRED. INT	NODES	MAE	MSE	RMSE	MAPE
All	7	25	31,662	1637,442	40,465	0,991
7	7	50	1946,616	3789939,564	1946,777	60,922
30	7	50	22,112	1129,432	33,607	0,692
60	7	25	21,944	1109,686	33,312	0,687
90	7	50	23,121	728,1	26,983	0,724
365	7	25	18,568	827,1	28,759	0,581
All	30	25	51,831	3543,882	59,531	1,433
7	30	50	342,633	118235,242	343,854	9,471
30	30	25	19,434	924,213	30,401	0,537
60	30	50	28,882	1135,669	33,7	0,798
90	30	25	21,7	1232,205	35,103	0,600
365	30	25	20,716	880,087	29,666	0,573
All	60	25	23,064	1349,574	36,737	0,549
7	60	25	29,163	2046,243	45,235	0,694
30	60	50	30,133	2103,259	45,861	0,717
60	60	50	23,791	1225,875	35,012	0,566
90	60	25	24,251	1325,563	36,408	0,577
365	60	25	59,235	4771,672	69,077	1,410
All	90	25	23,598	1093,415	33,067	0,565
7	90	25	184,161	34939,915	186,922	4,406
30	90	50	26,473	1188,085	34,469	0,633
60	90	25	39,386	2612,146	51,109	0,942
90	90	25	40,795	2726,922	52,22	0,976
365	90	25	40,386	2698,297	51,945	0,966
All	365	25	84,813	12761,1	112,965	0,995
7	365	50	189,422	41572,507	203,893	2,223
30	365	50	212,002	50368,953	224,43	2,488
60	365	25	105,675	16739,003	129,379	1,240
90	365	25	118,624	19704,101	140,371	1,392
365	365	25	202,652	46660,945	216,011	2,379

Table B.11: Results obtained for the LSTM model on the Tasks data set.

TRAIN. INT	PRED. INT	NODES	MAE	MSE	RMSE	MAPE
All	7	25	1540,446	3498197,151	1870,347	1,004
7	7	25	21076,352	445338943,8	21103,055	13,734
30	7	25	11595,367	135577015,2	11643,754	7,556
60	7	25	1523,42	3446031,009	1856,349	0,993
90	7	50	1522,153	3442174,45	1855,31	0,992
365	7	25	1761,028	4226457,525	2055,835	1,148
All	30	25	1094,625	1873091,721	1368,609	1,084
7	30	50	783,127	1273871,486	1128,659	0,776
30	30	50	12306,76	152128432	12334,036	12,190
60	30	25	1748,893	3733652,472	1932,266	1,732
90	30	50	52423,615	2748899541	52429,949	51,925
365	30	50	812,197	1326925,744	1151,923	0,804
All	60	25	3000,71	20280977,78	4503,441	0,996
7	60	25	91918,752	8460238409	91979,554	30,496
30	60	50	5216,787	38489019,93	6203,952	1,731
60	60	50	3762,888	25435154,33	5043,328	1,248
90	60	50	1733,307	11280124,08	3358,59	0,575
365	60	25	1979,964	14733086,35	3838,37	0,657
All	90	25	3379,469	20008419,1	4473,077	1,020
7	90	50	10401,151	116758785,9	10805,498	3,139
30	90	25	2138,954	12662998,69	3558,511	0,645
60	90	50	5382,132	37551304,73	6127,912	1,624
90	90	50	24307,909	599424435,5	24483,146	7,335
365	90	25	3304,924	19510278,06	4417,044	0,997
All	365	25	2751,104	12526479,83	3539,277	0,833
7	365	25	28536,878	819315921,6	28623,695	8,644
30	365	50	10754,721	119620936,5	10937,136	3,258
60	365	25	11912,648	145904995,3	12079,114	3,609
90	365	50	9429,27	92804879,2	9633,529	2,856
365	365	25	2100,745	8956621,556	2992,762	0,636

Table B.12: Results obtained for the LSTM model on the Incidents data set.

TRAIN. INT	PRED. INT	NODES	MAE	MSE	RMSE	MAPE
All	7	50	24,098	1218,783	34,911	0,754
7	7	25	34,396	1938,035	44,023	1,076
30	7	25	24,942	1259,307	35,487	0,781
60	7	25	19,605	650,455	25,504	0,614
90	7	25	31,946	1660,066	40,744	1,000
365	7	25	31,061	1601,363	40,017	0,972
All	30	25	34,613	2054,27	45,324	0,957
7	30	25	82,338	7631,955	87,361	2,276
30	30	25	36,578	2918,208	54,02	1,011
60	30	25	34	2012,252	44,858	0,940
90	30	25	33,014	3823,676	61,836	0,913
365	30	50	20,619	1087,376	32,975	0,570
All	60	25	24,962	1662,04	40,768	0,594
7	60	50	67,696	5836,325	76,396	1,612
30	60	50	214,873	47402,884	217,722	5,115
60	60	25	48,042	3569,284	59,743	1,144
90	60	25	24,948	1655,066	40,682	0,594
365	60	25	42,433	3038,41	55,122	1,010
All	90	25	55,886	4183,011	64,676	1,337
7	90	25	37,469	2575,942	50,754	0,896
30	90	25	30,276	1443,739	37,997	0,724
60	90	25	22,862	1295,208	35,989	0,547
90	90	25	60,348	4684,549	68,444	1,444
365	90	25	23,457	1385,653	37,224	0,561
All	365	25	73,72	11041,979	105,081	0,865
7	365	25	99,943	15511,722	124,546	1,173
30	365	50	50,418	5845,376	76,455	0,592
60	365	25	50,673	6917,866	83,174	0,595
90	365	25	83,321	12516,54	111,877	0,978
365	365	25	119,162	19669,728	140,249	1,399

Table B.13: Results obtained for the LSTMX model on the Tasks data set.

TRAIN. INT	PRED. INT	NODES	MAE	MSE	RMSE	MAPE
All	7	25	1442,744	3206729,301	1790,734	0,940
7	7	25	1305,324	2829817,594	1682,206	0,851
30	7	50	2197,117	5954020,503	2440,086	1,432
60	7	25	1346,49	2938856,433	1714,309	0,877
90	7	25	688,698	1203437,725	1097,013	0,449
365	7	25	1713,151	4061192,094	2015,24	1,116
All	30	25	968,164	1608653,005	1268,327	0,959
7	30	25	1061,217	1802642,319	1342,625	1,051
30	30	25	24549,546	603349119,7	24563,166	24,316
60	30	50	5450,14	30378208,52	5511,643	5,398
90	30	50	595,729	686152,973	828,344	0,590
365	30	25	852,764	1398505,964	1182,584	0,845
All	60	25	2959,459	20040084,27	4476,615	0,982
7	60	50	4989,615	36171361,9	6014,263	1,655
30	60	50	40170,541	1624910251	40310,176	13,327
60	60	25	2671,941	18400364,1	4289,565	0,886
90	60	25	2668,85	18392122,12	4288,604	0,885
365	60	25	2984,573	20184517,66	4492,718	0,990
All	90	25	3320,747	19615849,16	4428,978	1,002
7	90	50	5509,624	33804208,68	5814,139	1,663
30	90	50	3223,078	18978878,19	4356,475	0,973
60	90	50	6746,386	54094911,53	7354,924	2,036
90	90	25	1626,501	9357251,834	3058,963	0,491
365	90	25	3310,516	19550442,62	4421,588	0,999
All	365	25	2930,747	13588593,21	3686,271	0,888
7	365	25	5876,529	37801059,43	6148,257	1,780
30	365	25	9248,364	90528788,82	9514,662	2,802
60	365	50	2819,289	12923816,14	3594,971	0,854
90	365	25	1816,548	7453539,44	2730,117	0,550
365	365	25	3284,608	15808100,79	3975,94	0,995

Table B.14: Results obtained for the LSTMX model on the Incidents data set.

TRAIN. INT	PRED. INT	MAE	MSE	RMSE	MAPE
365	7	0,479	6,917646363	0,503	0,162
All	7	5,783	24,0483856	6,105	2,211
365	30	0,689	8,298731457	0,728	0,315
All	30	3,143	17,7290764	3,186	1,319
All	60	0,293	5,417309646	0,335	0,169
365	60	2,202	14,83855198	2,248	0,563
All	90	0,256	5,059648585	0,283	0,089
365	90	2,22	14,90057214	2,331	1,117
All	365	5,231	22,87124833	5,323	1,209
365	365	45,36257813	198,3366066	46,16039063	10,48429688

Table B.15: Results obtained for the DeepAR model on the Tasks data set.

TRAIN. INT	PRED. INT	MAE	MSE	RMSE	MAPE
365	7	2,083	14,434202	2,187	0,208
All	90	3,212	17,92150524	3,591	0,321
All	30	4,184	20,45469323	4,27	0,418
365	90	4,533188337	25,29313777	5,068081979	0,453036568
365	30	5,905	24,30109668	5,908	0,591
All	60	7,915	28,13292733	7,922	0,791
All	365	11,146	33,38583532	11,39	1,115
365	60	11,17066802	39,70481259	11,18054732	1,116361138
365	365	15,73067161	47,1183933	16,07503585	1,573631692
All	7	56,919	75,44482222	58,605	5,692

Table B.16: Results obtained for the DeepAR model on the Incidents data set.

TRAIN. INT	PRED. INT	MAE	MSE	RMSE	MAPE
All	7	1,937	6,489	2,547	0,626
7	7	3,052	16,027	4,003	0,986
30	7	2,005	6,588	2,567	0,648
60	7	1,925	6,68	2,585	0,622
90	7	1,956	6,61	2,571	0,632
365	7	1,91	6,621	2,573	0,617
All	30	1,655	5,04	2,245	0,612
7	30	1,863	8,389	2,896	0,689
30	30	1,754	5,397	2,323	0,648
60	30	1,708	5,165	2,273	0,631
90	30	1,626	5,057	2,249	0,601
365	30	1,618	5,023	2,241	0,598
All	60	1,689	5,471	2,339	0,618
7	60	6,446	52,768	7,264	2,358
30	60	1,632	5,881	2,425	0,597
60	60	1,574	5,8	2,408	0,576
90	60	1,569	5,737	2,395	0,574
365	60	1,637	5,433	2,331	0,599
All	90	1,64	4,779	2,186	0,603
7	90	17,173	378,122	19,445	6,313
30	90	1,549	5,747	2,397	0,569
60	90	1,536	5,717	2,391	0,565
90	90	1,496	4,886	2,21	0,550
365	90	1,576	4,649	2,156	0,579
All	365	1,883	6,983	2,642	0,626
7	365	40,915	2194,952	46,85	13,606
30	365	2,411	11,737	3,426	0,802
60	365	2,611	13,121	3,622	0,868
90	365	2,079	9,79	3,129	0,691
365	365	1,79	7,042	2,654	0,595

Table B.17: Results obtained for the Prophet model on the Tasks data set.

TRAIN. INT	PRED. INT	MAE	MSE	RMSE	MAPE
All	7	12,902	431,297	20,768	2,486
7	7	15,732	559,144	23,646	1089,358
30	7	17,165	718,814	26,811	593,048
60	7	18,922	716,263	26,763	194,217
90	7	14,696	548,795	23,426	87,242
365	7	11,882	461,386	21,48	32,882
All	30	14,68	303,457	17,42	3,250
7	30	18,507	414,706	20,364	2315,795
30	30	15,076	389,001	19,723	498,047
60	30	10,367	200,08	14,145	192,568
90	30	14,037	270,964	16,461	88,959
365	30	9,922	177,844	13,336	11,171
All	60	13,892	578,121	24,044	0,962
7	60	93,96	11272,252	106,171	1,212
30	60	14,788	706,166	26,574	0,988
60	60	17,548	653,555	25,565	0,679
90	60	13,423	556,167	23,583	0,919
365	60	10,027	533,357	23,095	0,650
All	90	11,734	419,031	20,47	0,539
7	90	88,831	9832,173	99,157	4,077
30	90	27,582	1173,263	34,253	1,266
60	90	14,386	579,831	24,08	0,660
90	90	10,826	470,82	21,698	0,497
365	90	9,65	420,002	20,494	0,443
All	365	9,695	211,796	14,553	0,449
7	365	1277,069	2160247,049	1469,778	59,206
30	365	17,601	548,278	23,415	0,816
60	365	9,194	239,868	15,488	0,426
90	365	17,176	416,593	20,411	0,796
365	365	17,849	448,942	21,188	0,828

Table B.18: Results obtained for the Prophet model on the Incidents data set.