

Time series modeling of elevator traffic

Viljami Pirrtimaa

School of Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 02.09.2020

Supervisor

Professor Sven Bossuyt

Advisors

D.Sc. (Tech.) Juho Kokkala

D.Sc. (Tech.) Juha-Matti Kuusinen

Copyright © 2020 Viljami Pirttimaa



Author Viljami Pirttimaa

Title Time series modeling of elevator traffic

Master programme Mechanical Engineering

Code ENG25

Supervisor Professor Sven Bossuyt

Advisors D.Sc. (Tech.) Juho Kokkala, D.Sc. (Tech.) Juha-Matti Kuusinen

Date 02.09.2020

Number of pages 86

Language English

Abstract

Large changes in people flow can happen in buildings throughout their life cycles. This thesis studies long-term forecasting of elevator traffic data using time series modeling. The models can also be used for automatic detection of anomalies, which makes it possible to quickly react to unexpected problems. Long-term forecasts can be used for planning changes to elevator systems before their capacity is exceeded and problems in people flow arise.

This thesis includes a literature review of time series characteristics and models. Year-ahead forecasts were made with one traditional and two machine learning-based models, which were chosen based on the literature review. The forecasts obtained with the models were compared to a simple baseline model, which is based on the observations from the previous year. Three models were used for anomaly detection. Their results were evaluated by comparing their findings to known events during the test period.

The forecasting methods modeled the complex seasonality in the time series well. Predicting the long-term trend in the time series was not as successful with any of the models. Based on the results, XGBoost machine learning model turned out to be the best-performing method. However, the differences between the models were small and only data from a single building was used, which makes it impossible to generalize the result. The anomaly detection models found several days with less or more traffic than normal. For some of these days, an explanation supporting the findings of the models could be found.

People flow data can give invaluable insights into how different types of buildings are used and how people flow changes throughout the lifecycle of each building. Long-term forecasts are helpful especially when elevator modernizations are planned. Anomaly detection is useful for servicing the elevators when problems are detected. Based on the results of this thesis, people flow data should be collected also in the future, as it can be a valuable asset.

Keywords People flow, time series, forecasting, anomaly detection, machine learning

Tekijä Viljami Pirttimaa

Työn nimi Hissiliikenteen aikasarjamallintaminen

Maisteriohjelma Konetekniikka

Koodi ENG25

Työn valvoja Professori Sven Bossuyt

Työn ohjaajat TkT Juho Kokkala, TkT Juha-Matti Kuusinen

Päivämäärä 02.09.2020

Sivumäärä 86

Kieli Englanti

Tiivistelmä

Rakennusten ihmisvirroissa voi tapahtua suuriakin muutoksia rakennuksen elinkaaren aikana. Tämä diplomityö tutkii, voidaanko hissiliikenteen ihmisvirtoja ennustaa pitkälle tulevaisuuteen käyttämällä aikasarjojen mallinnusmenetelmiä. Mallit mahdollistavat myös odottamattomien muutosten automaattisen tunnistamisen, joka nopeuttaa ongelmien korjaamista. Pitkän ajan ennusteilla voidaan suunnitella muutoksia hissijärjestelmiin ennen kuin niiden kapasiteetti ylittyy ja ongelmia ihmisvirroissa alkaa esiintyä.

Tutkimukseen sisältyy kirjallisuuskatsaus aikasarjoihin ja niiden mallinnusmenetelmiin. Vuoden mittaisia ennusteita tehtiin yhdellä perinteisellä ja kahdella koneoppimiseen perustavalla mallinnusmenetelmällä, jotka valittiin kirjallisuuskatsauksen perusteella. Mallien ennusteita verrattiin yksinkertaiseen verrokkimalliin, joka perustuu edellisvuoden arvoihin. Odottamattomia tapahtumia tunnistettiin kolmella mallilla. Niiden tuloksia arvioitiin vertaamalla mallien löydöksiä tunnettuihin tapahtumiin testijakson aikana.

Ennustusmallit onnistuivat mallintamaan aikasarjojen monimutkaiset kausivaihtelut hyvin. Sen sijaan pitkän aikavälin trendien ennustamisessa oli ongelmia kaikilla malleilla. Tulosten perusteella XGBoost-koneoppimismalli osoittautui parhaaksi ennustusmenetelmäksi. Koska erot mallien välillä olivat pieniä ja mallinukseen käytettiin vain yhden rakennuksen dataa, ei tuloksia voida pitää yleispätevinä. Odottamattomien tapahtumien tunnistamiseen käytetyt mallit löysivät useita päiviä, joiden liikenne oli pienempää tai suurempaa kuin normaalisti. Osalle näistä päivistä pystyttiin löytämään selitys, joka tukee mallien löydöksiä.

Aikasarjamallinnuksen avulla datasta voidaan oppia, miten erilaisten rakennusten ihmisvirrat muuttuvat niiden elinkaaren aikana. Pitkän ajan ennusteista on hyötyä etenkin hissien modernisoinnin suunnittelussa, ja poikkeamien tunnistamisesta muun muassa hissien huoltamisessa. Tulosten perusteella voidaan todeta, että hissiliikenteen dataa kannattaa kerätä myös tulevaisuudessa, sillä se voi osoittautua erittäin arvokkaaksi.

Avainsanat Aikasarja, hissiliikenne, ennustaminen, poikkeamien tunnistaminen, koneoppiminen

Preface

I want to thank Professor Sven Bossuyt for supervising my thesis. He handled the practical arrangements from the school's side so I could focus on working on the thesis. I also want to thank my advisors Juha-Matti Kuusinen and Juho Kokkala. Juha-Matti took care of the arrangements from KONE's side and made me feel at home while working in his team. Juho guided me through the whole process starting from the planning of the topic all the way to the final proofreading. He gave me some direction to keep the modeling on the right track. His comments on the text were more thorough and extensive than I could ever had hoped for.

I will also thank my family and friends who have helped me over the years in my studies. Lastly, I want to thank my girlfriend who has cheered me up and motivated me to finish this thesis in a timely manner.

Graduation feels like I am leaving a big part of my life in the past, but it also gives me a great sense of accomplishment. Fortunately, I have had the chance to already familiarize myself with the working life, so I feel excited about what is waiting for me in the future.

Espoo, 02.09.2020

Viljami Pirrtimaa

Contents

Abstract	3
Abstract (in Finnish)	4
Preface	5
Contents	6
Symbols and abbreviations	8
1 Introduction	10
1.1 Objectives	11
1.2 Structure	11
2 Literature review	12
2.1 Time series data	12
2.2 Properties of time series data	13
2.3 Stationarity and unit-root tests	17
2.4 Transformations and seasonal adjustment	19
2.4.1 Removing heteroscedasticity	19
2.4.2 Removing trend	21
2.4.3 Removing seasonality	23
2.5 Time series decomposition	23
2.6 Imputation	27
2.7 Time series models	28
2.7.1 Simple models	29
2.7.2 Model parsimony	30
2.8 Stochastic models	30
2.8.1 Exponential smoothing (ETS)	31
2.8.2 Autoregressive models (AR)	32
2.8.3 Moving average models (MA)	32
2.8.4 ARMA, ARIMA and SARIMA models	33
2.8.5 Box-Jenkins methodology	33
2.8.6 TBATS	34
2.9 Machine learning	34
2.9.1 Neural networks	35
2.9.2 RNN	37
2.9.3 Long short-term memory (LSTM)	37
2.9.4 Decision tree models	37
2.9.5 Random forests	40
2.9.6 Gradient boosting	40
2.10 Forecasting	41
2.10.1 Producing forecasts with a model	41
2.10.2 Evaluating forecast accuracy	42

2.11	Anomaly detection	44
3	Materials and methods	46
3.1	Elevator-specific terms and statistics	46
3.2	Data	46
3.3	Implications for modeling	50
3.4	Smart seasonal differencing and smart lag	51
3.5	Imputing missing values in data	51
3.6	Forecasting	56
3.6.1	Forecasting experiment	56
3.6.2	LSTM	57
3.6.3	ARIMA with smart seasonal differencing (ASSD)	59
3.6.4	XGBoost	60
3.7	Anomaly detection	62
3.7.1	<i>AnomalyDetection</i> R package	62
3.7.2	Rolling average and standard deviation	63
3.7.3	ARIMA with smart seasonal differencing (ASSD)	63
3.7.4	tsoutliers	64
4	Results	65
4.1	Forecasting	65
4.1.1	LSTM	68
4.1.2	ARIMA with smart seasonal differencing	70
4.1.3	XGBoost	72
4.2	Anomaly detection	74
5	Discussion	78
5.1	Forecasting	78
5.2	Anomaly detection	79
6	Conclusions	81
	References	82

Symbols and abbreviations

Symbols

α	level smoothing parameter in an ETS model
b_t	trend equation of an ETS model at time t
β	trend smoothing parameter an ETS model
c	constant level
d	order differencing in an ARIMA model
ε_t	Stochastic error term at time t
ϕ_i	i^{th} order weight of an AR model
γ	seasonal smoothing parameter for an ETS model
G_i	Gini impurity for the i^{th} node of a decision tree
h	forecast horizon
k	number of predictors in a model, a constant used for calculation
l_t	level equation of an ETS model at time t
λ	parameter used in Box-Cox transformation
λ_k	k^{th} frequency in harmonic regression
m	number of observations per seasonal period
n	length of a rolling window
r_k	sample autocorrelation for lag k
σ_t	rolling standard deviation at time t
S_t	seasonal component at time t
s_t	seasonal equation of an ETS model at time t
t	time-step at time t
T	length of a time series
T_t	trend component at time t
TD_t	deterministic component of a time series at time t
θ_i	i^{th} order weight of a MA model
p	order of an autoregressive model
q	order of a moving average model
R_t	remainder component at time t
y_t	value of time series y at time t
\hat{y}_t	predicted value of time series y at time t
\bar{y}	mean value of time series y
y'	differenced time series y
z_t	stochastic component of a time series at time t

Operators

$S(y_t)$	Smart lag value of time series y at time t
$L(y_t)$	Smart lead value of time series y at time t

Abbreviations

ACF	Autocorrelation Function
ADF	Augmented Dickey-Fuller
AIC	Akaike Information Criterion
AR	Auto Regressive
ASSD	ARIMA with smart seasonal differencing
BIC	Schwarz's Bayesian Information Criterion
CART	Classification and Regression Tree
CT	Call Time
DCS	Destination Control System
DNN	Deep Neural Network
DOP	Destination Operating Panel
ESD	Extreme Studentized Deviate (ESD) test
ETS	Exponential Smoothing
GPU	Graphics Processing Unit
KPSS	Kwiatkowski-Phillips-Schmidt-Shin
LOCF	Last Observation Carried Forward
LOESS	Locally Estimated Scatterplot Smoothing
LSTM	Long Short-Term Memory
MA	Moving Average
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MBE	Mean Bias Error
MLE	Maximum Likelihood Estimate
NaN	Not a Number
NN	Neural Network
NOCB	Next Observation Carried Backward
OLS	Ordinary Least Squares
PACF	Partial Autocorrelation Function
PP	Phillips-Perron
RMS	Root Mean Square
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
ReLU	Rectified Linear Unit
S-H-ESD	Seasonal Hybrid ESD algorithm
SARIMA	Seasonal Autoregressive Integrated Moving Average
STL	Seasonal and Trend decomposition using Loess
TPU	Tensor Processing Unit
TTD	Time To Destination
WT	Waiting Time

1 Introduction

In modern office buildings it is common to have many tenants which might change throughout the lifetime of the building. This can have significant impact on the people flow of the building and the performance of the elevator system. If there is a considerable increase in building population, elevator usage can exceed the original design capacity of the elevator system. This will cause long waiting times and poor performance during peak traffic times like lunch time [1].

Elevator performance can be improved with modernization. A common approach is to change from a conventional elevator control system to a destination control system (DCS). This can increase the maximum handling capacity by 30 % [1]. Performance can be further improved by modernizing the elevator drives to increase their speed.

Ordinarily the modernization process is started after the elevator performance has become inadequate. The time frame from noticing the poor performance to addressing the issue with modernizing can be undesirably long. First the tenants complain to the building owner, then the building owner complains to the sales department of the elevator company and they inform the department which investigates the problem. If the modernization could be planned before the problems start, it could be done at the most convenient time like during other renovation. This could save costs and improve the daily experience of the tenants.

Elevator systems already collect usage data which includes the number of calls and waiting time for each call. This data is invaluable in troubleshooting the cause of poor performance and planning modernizations. This data could also be used for predicting long-term future traffic demand and automatic detection of temporary and persistent changes in demand and performance. If problems are expected in the future, discussions about modernization can be started with the building owner well before problems arise. In case the people flow deteriorates unexpectedly, the problem can be analyzed before the tenants complain to the building owner.

Sometimes poor performance can be caused by something as simple as wrong configuration or operation mode after routine service of the elevators. If such issue could be automatically detected, it could be resolved even before the tenants complain about poor performance.

Usage patterns of office buildings are highly seasonal [2]. During weekends and holidays there is generally almost no traffic. This can make before-after comparisons difficult after modernization. Using the collected usage data, the seasonality can be modeled, and its effect can be subtracted from the data. This would make before-after analyses comparable and the effect of the modernization would be clearer.

Some research, e.g. [3], has been done related to short-term (5-15 minutes ahead) predictions of elevator traffic demand. These predictions are useful in optimization of elevator control systems. However, little research is available on long-term predictions of elevator traffic. One possible reason for this is the limited availability of suitable data. Long-term predictions are common in other fields such as electricity demand which has been extensively researched [4, 5]. Applying similar methods to elevator data should yield interesting results.

1.1 Objectives

The main objective of this thesis is to develop a suitable time series model for elevator traffic data. This model should provide long-term forecasts of key people flow metrics such as number of calls and average waiting time. A good time series model also makes it possible to automatically detect anomalies like exceptional events and persistent changes in the data. While there are many use cases for such a model, studying them is out of scope of this thesis.

The time series models are developed by comparing many existing methods using real elevator traffic data provided by KONE Corporation. Comparison between the models is made by measuring their prediction accuracy and anomaly detection capability. The models are validated using known events and persisting changes in the real-world data. Prediction accuracy is evaluated by comparing the forecasts to out-of-sample test and validation data.

Another goal for the model is to extract the seasonality in the elevator traffic. By filtering out the seasonality in the data, a better understanding of the trend can be achieved. Deseasonalized data is also useful for comparing the elevator performance at different points in time.

While the data processing, forecasts and anomaly detection will not be automated in this thesis, it should be easy to apply the model to the data from different buildings. This would enable complete automation of the process in the future. The model parameters will likely be different in different types of buildings, but there are methods to automatically optimize them based on measured data. This means that methods which require manual tuning of parameters should be avoided.

1.2 Structure

This thesis is divided to the following chapters:

- Chapter 1 presents the background and main objectives for this thesis.
- Chapter 2 contains a brief literature review of the time series modeling methods considered in this thesis.
- Chapter 3 describes the data and modeling methods used for the time series analysis.
- Chapter 4 outlines the results obtained with the developed models.
- Chapter 5 contains a discussion about the applicability of the chosen methods and improvement ideas for future research.
- Chapter 6 summarizes the outcome of this thesis.

2 Literature review

This chapter contains a literature review of the topics that are included in the case study part of this thesis. First is a brief introduction to time series data and its properties. Second part presents different transformation methods for time series data and imputation of missing values. Time series modeling and forecasting methods are described in the third part. The last part introduces anomaly detection. A high-level overview of the topics is presented in this thesis. For a more thorough theoretical explanation, see e.g. [6–12].

2.1 Time series data

Time series data consists of continuous or discrete-time observations of a process or a phenomenon over time. Time series in engineering applications are always discrete-time because continuous measurement is not technically possible. Often the data is sampled at regular intervals to simplify the acquisition and analysis of the data. Regularly sampled time series data is denoted by y_t , $t = 1, 2, 3 \dots$. The data is indexed by t which is the point in time when that data point was sampled [6].

Unlike cross-sectional datasets, time series data has a defined time order. Information about the measured phenomenon is not only contained in the values themselves but also in the way they are arranged [6]. Discrete time series is almost always a representation of a continuous process. This means that every observation is correlated to the previous observations in some way. This dependence is measured by autocorrelation. Autocorrelation is a useful property for forecasting, i.e., predicting future values of a time series based on the current and previous values [7].

In many time series modeling methods time series are thought as realizations of a stochastic process [8]. This means that there is some expected value and a probability distribution for every point in time as well as a joint probability distribution across all the samples. The measured time series is interpreted as one realization of that stochastic process. If the expected value, i.e., mean and the probability distribution can be modeled, future values can be forecasted with the conditional distribution $p(y_t | y_1, \dots, y_{t-1})$. This distribution also gives confidence intervals for the forecast.

Time series data can be divided in two different categories, univariate and multivariate. Univariate data has only one variable that is changing over time. Multivariate time series has multiple variables that are sampled at the same points in time. The different variables in a multivariate time series might have some correlation which can help in the modeling process. Although the data used in this thesis is multivariate, only univariate methods are used in this thesis. Researching the modeling of elevator traffic data using multivariate methods could be a good direction for future research.

Time series modeling has multiple use cases in engineering. Often fitting a suitable model to the data helps understanding the process that generated the time series data. If there is a trend, seasonality or sudden level shifts, they might be explainable by some external influence on the process. A time series model can also be used for filtering out noise in the observations [6]. This is especially useful if the

measurements are used for controlling the process that generates the data. One of the most widely used applications for time series modeling is forecasting. Modeling the current and previous observations of the time series, a prediction of future values can be achieved.

Time series are often visualized with a line plot whose x-axis represents time. One of the oldest time series datasets in existence is the number sunspots per year shown in Figure 1. The dataset starts from the year 1700 and it is still updated by the Royal Observatory of Belgium [13]. The data shows some typical properties of time series data which are discussed in the next Section 2.2.

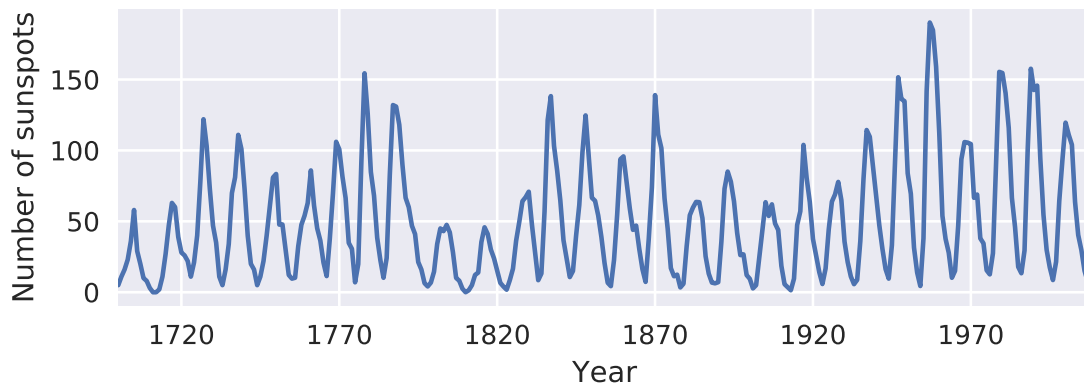


Figure 1: Sunspot data (1700-2008) from the National Geophysical Data Center.

2.2 Properties of time series data

The most typical features for real-world time series data are trend, seasonality and autocorrelation. All three of them can be observed in the airline passenger data shown in Figure 2. The example data shows a clear increasing trend which is approximately linear. Trends can also be decreasing and follow logarithmic, exponential or some other curves. Trend is always assumed to be smooth and continuous which means that quick fluctuations should be interpreted as random noise or sudden level shifts.

The air passenger data is collected at a monthly interval. The peaks and troughs are clearly 12 months apart indicating a yearly seasonality. The data shows that there is more traffic in the summer months compared to the winter. The amount of seasonal variation seems to increase according to the trend. This suggests that the airline passenger dataset could be decomposed to multiplicative components $y_t = T_t \times S_t \times R_t$, where T_t is the trend, S_t is the season and R_t is a remainder component. This decomposition is shown in Figure 3 which shows a smooth increasing trend and a clear seasonality. Some seasonal pattern can also be seen in the beginning and the end of the residual part. This indicates that the seasonality is not quite perfectly multiplicative. Time series decomposition is described in more detail in Section 2.5.

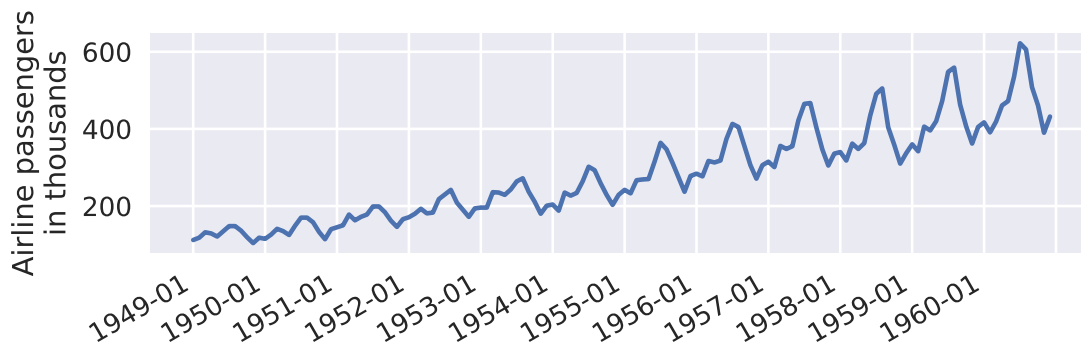


Figure 2: Monthly airline passengers from 1949 to 1960. Dataset E in [10].

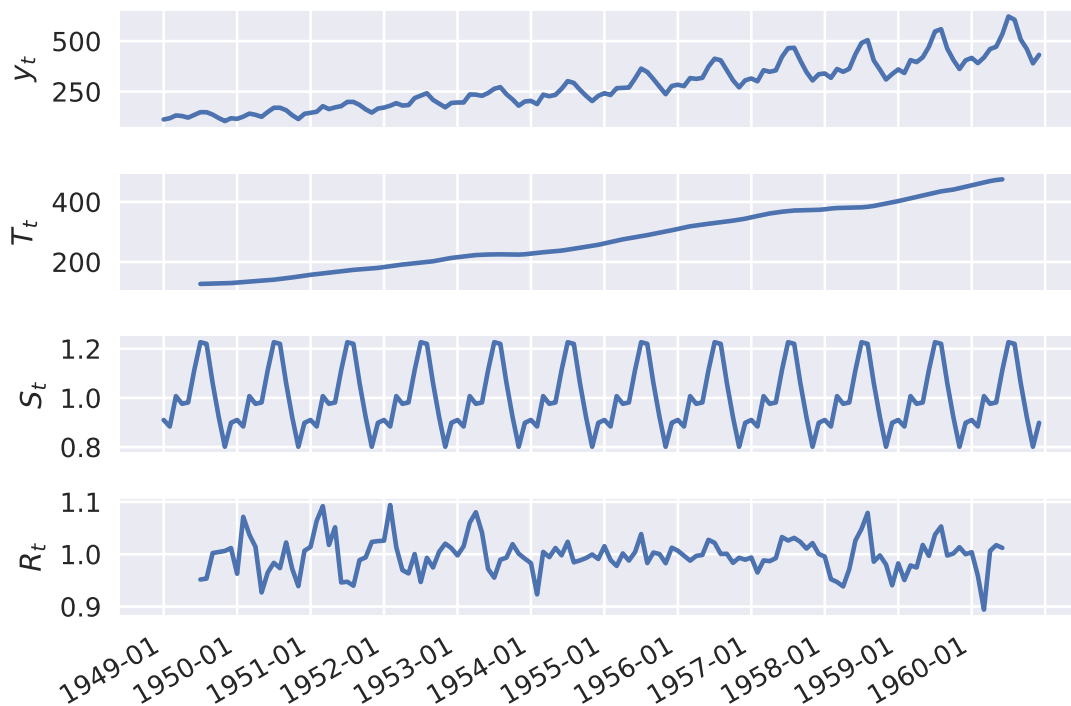


Figure 3: Multiplicative components of the airline passenger data.

Some datasets may also contain multiple seasonalities. For example, electricity usage has both yearly and weekly seasonalities. Less electricity is used on weekends when most factories are closed. Heating and cooling increase the electricity demand in the winter and summer months [5]. Having multiple seasonalities makes data analysis more difficult as most methods are designed for single seasonality. Overcoming the complexity caused by weekly and yearly seasonalities is discussed in more detail in Section 3.5.

Autocorrelation measures the correlation of a time series to its past values. It is analogous to ordinary correlation which measures the linear relationship between two different variables. Autocorrelation coefficient r_1 measures the correlation between the current value y_t and the value lagged by one observation y_{t-1} . Autocorrelation is a theoretical feature of the stochastic process that generated the time series data. It cannot be calculated with the sampled data but it can be estimated using the samples. Sample autocorrelation for lag k is calculated as follows:

$$r_k = \frac{\sum_{t=k+1}^T (y_t - \bar{y})(y_{t-k} - \bar{y})}{\sum_{t=1}^T (y_t - \bar{y})^2}, \quad (1)$$

where T is the length and \bar{y} the mean of the time series [9]. Autocorrelation function (ACF) plot for the sunspot data is shown in Figure 4. ACF plot of the airline passenger data is shown in Figure 5. The autocorrelation coefficients for each of the lags are represented by the blue dots in both figures. The first autocorrelation with lag zero is always 1 as the time series perfectly correlates with itself. All values inside the blue area are statistically indistinguishable from zero. In an ACF plot of white noise 95 % of the coefficients should be inside $\pm 2/\sqrt{T}$, where T is the length of the time series [9].

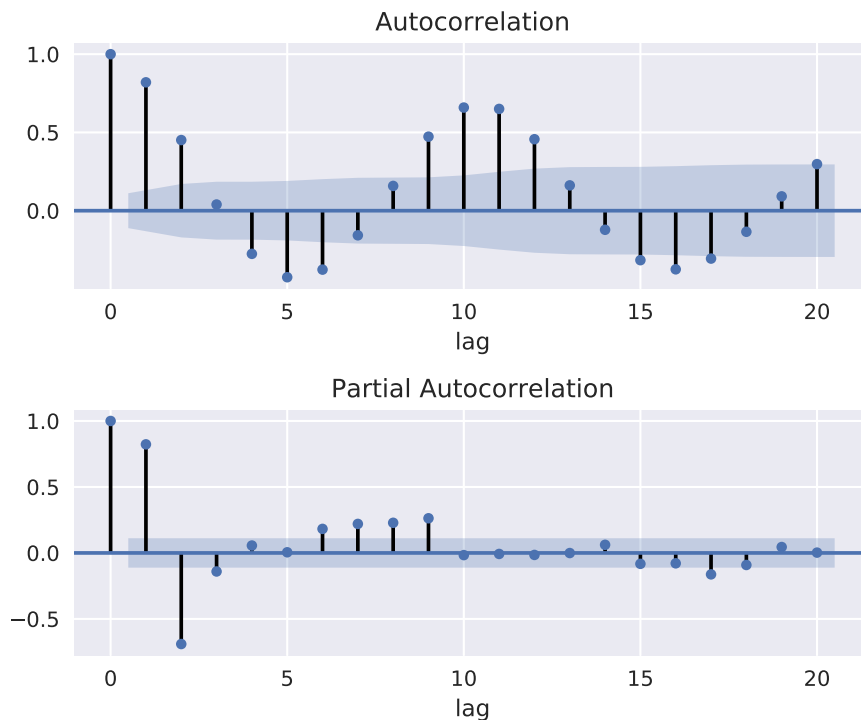


Figure 4: ACF and PACF plots of the sunspots data in Figure 1.

ACF plot of the sunspot data in Figure 4 shows oscillating behavior with nearly constant amplitude. The oscillation happens around zero which indicates that there

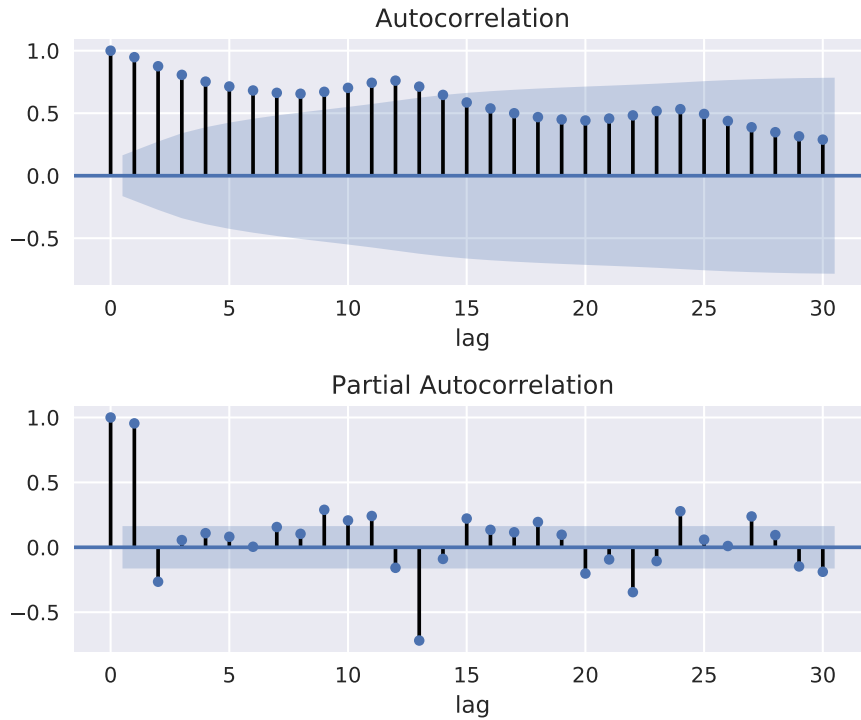


Figure 5: ACF and PACF plots of the airline passenger data in Figure 2.

is no long-term trend in the data. The frequency of the oscillation in the ACF plot is about 11 years which corresponds to the cyclic behavior in Figure 1. ACF plot of the airline passenger data in Figure 5 also has a cyclic behavior that corresponds to the yearly seasonality in the data. The autocorrelation coefficients are positive and decreasing steadily which is caused by the trend in the data. Values with small lags are highly correlated, but the trend causes the correlation to decrease with larger lags.

Figures 4 and 5 contain also partial autocorrelation function (PACF) plots. The purpose of the PACF plot is to show the remaining correlation between different lags after the effects of the other lags have been subtracted [11]. If there is correlation between y_t and y_{t-1} , there is also correlation between y_{t-1} and y_{t-2} . ACF plot does not show if there is correlation between y_t and y_{t-2} when conditioning on y_{t-1} . Partial autocorrelation shows the correlation between y_t and y_{t-k} after removing the effects of lags 1 to $k-1$ [9]. Because there are no observations between y_t and y_{t-1} , the partial autocorrelation coefficient is always the same as the autocorrelation coefficient for lag 1. Values inside the blue area are insignificantly different from zero with 95 % confidence level.

PACF plot for the sunspot data in Figure 4 shows significant correlation for lags 1, 2, 6, 8 and 9. Lag 17 is slightly outside the blue area but it is still likely caused by random variation. Significant lags for the airline passenger data are 1, 2, 9, 10, 11 and 13 according to Figure 5. ACF and PACF plots are important when selecting parameters for some time series modeling methods. Interpreting them is discussed in

more detail in Section 2.8 where AR and MA models are introduced.

2.3 Stationarity and unit-root tests

Most statistical forecasting methods assume time series to be stationary. Stationarity means that the statistical properties of the process do not change over time. More specifically for a stationary time series y_t , the distribution of (y_t, \dots, y_{t+s}) does not depend on t [9]. This implies that the mean and standard deviation of the time series stay constant over time. A classic example of a stationary time series is white noise. It contains all frequencies with the same amplitude and as such it looks statistically the same at all points in time. White noise is independent and identically distributed (i.i.d.), which means that it does not have autocorrelation. However, stationary time series can have autocorrelation, as long as it does not change over time.

Variance that changes over time is one of the common features in real-world time series that make them non-stationary. In literature, it is called heteroscedasticity. This behavior is clearly visible in the observed data y_t in Figure 2.

Stationarity of time series can be tested with multiple methods. The simplest method is to divide the time series to two or more parts and calculate the mean and variance for each of the sections. If the statistical properties for all the parts are similar, the time series can probably be assumed to be stationary. However, this naïve method does not detect things like cyclic behavior. To understand methods designed for this task, a brief overview of unit root processes is required.

Without going to too much mathematical detail about unit root processes, time series can be divided to three categories. Those for which all roots are under 1 are stationary and their stochastic behavior stays the same over time. Random noise might nudge the mean and variation temporarily, but they tend to get back to the original level. Processes which contain at least one root that is larger than one, are explosive in their nature [12]. Their variance increases over time and it tends to infinity.

An edge case between these processes is a unit root process which has at least one root equal to one but no roots more than one. Their level drifts randomly with the noise and their variance tends to infinity [12]. Unit root processes can be modeled with a random walk model in which each time period adds a random noise to the previous value. Over long time periods the accumulated error has a mean of zero, but its variance increases. A random nudge caused by the error term causes a persisting change in the level which does not tend back to the original level.

As an example the first order autoregressive process, $y_t = a_1 y_{t-1} + \varepsilon_t$ has a unit root when $a_1 = 1$ [11]. In that case the model becomes a random walk whose level fluctuates randomly with the ε_t term. As the effect of past random errors is not diminished by a_1 , the errors accumulate, and the stochastic properties of the time series change over time. This is illustrated in Figure 6 which shows a stationary and unit-root first order autoregressive processes. The error term ε_t is the same for both processes. It is sampled from a normal distribution with zero mean and standard deviation of 1. At $t = 20$ the error term was set to -10 to nudge both time series strongly downwards. The effect of the nudge is permanent for the unit root process

while the stationary process quickly recovers back to around 0.

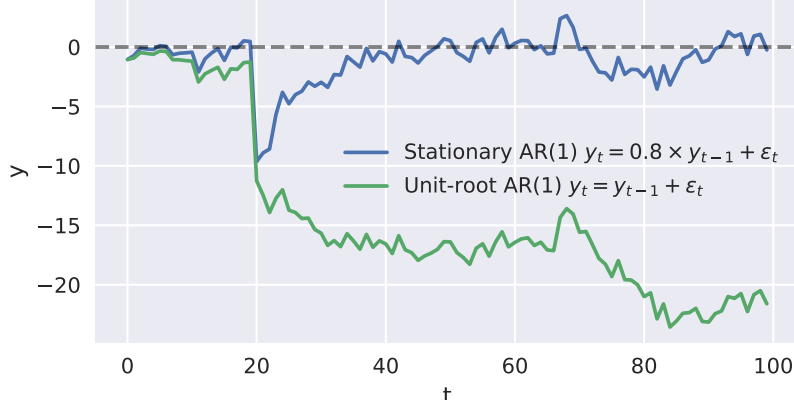


Figure 6: Stationary autoregressive AR(1) process compared to an autoregressive process that contains an unit-root. At $t = 20$ the error term ε_{20} is set to -10 to show how differently the time series behave.

Processes which have a root over one are easy to distinguish as their behavior is chaotic and they tend to infinity. Determining whether a process is a non-stationary unit root process or a stationary process with all roots less than one is more difficult. Multiple statistical methods called unit root tests have been developed for this task.

Each time series to be tested is assumed to contain three parts:

$$y_t = TD_t + z_t + \varepsilon_t,$$

where TD_t is the deterministic part, z_t is the stochastic component and ε_t is the stationary error process. If it can be shown that the stochastic component contains a unit root, it can be concluded that the time series is not stationary [12].

One of the most widely used unit root tests is called augmented Dickey–Fuller test (ADF) which was first defined in [14]. It is based on a null hypothesis which states the time series contains a unit root, implying it is non-stationary. If the hypothesis can be statistically rejected, the time series can be assumed stationary. There are three variations of the ADF test which account for different types of deterministic behavior in the data. First one of them is for stationary time series with no deterministic part. The second one accounts for a constant level and it is used for level stationary data. The third alternative also adds a constant trend suitable for trend stationary data.

Another common unit root test is Phillips-Perron (PP) test [15] which is more robust to detecting heteroscedasticity. It adds a correction factor for autocorrelated residual errors to the ADF test. It also has the same three variations for the deterministic part of the time series. PP test results are interpreted like ADF test results and the critical values are the same.

A common shortcoming for most unit root tests is their lack of statistical power to prove stationary time series near unit root as being stationary. This means that the null hypothesis is not rejected for many time series that are stationary. One

cause for this is the fact that unit root tests do not handle structural changes in the data like a change in trend or level [12].

In addition to these unit root tests a different type of statistical test called Kwiatkowski-Phillips-Schmidt-Shin (KPSS) is introduced in [16]. It inverts the null hypothesis of the ADF meaning that the time series is assumed stationary unless otherwise statistically proven. It tends to work better for time series that are near unit root. Interpreting KPSS test results is inverted compared to ADF. KPSS test statistic indicates stationarity if it is larger than the critical value in Table 1 in [16].

While the implementation details of these tests are not relevant in the scope of this thesis, these tests can be quite easily used. They are implemented in multiple software packages like the *forecast* [17] package for R and *statsmodels* [18] library for Python. It is important to keep in mind that the tests are statistical and as such their results might contradict each other. A good approach to analyzing stationarity of a time series is to use multiple tests and compare their results critically. Failing one of the tests does not automatically mean the time series is non-stationary, but if most of the tests indicate so, that is probably the case. A good approach is to utilize the opposing hypothesis of the ADF and KPSS tests. If their results contradict, the strength of the evidence, i.e. p -values of the tests, can be compared.

Many real-world time series datasets contain features that make them non-stationary. Methods that require stationarity can still be utilized, as there are multiple ways to transform the data so that it becomes weakly stationary. These methods are described in the next Section 2.4.

2.4 Transformations and seasonal adjustment

As discussed earlier, many real-world datasets contain features that make them non-stationary and difficult to model. These issues can often be overcome with the help of transformations. Different transformations can be utilized for handling different properties, and they can be combined to enable the modeling of almost any time series. Most common transformations and adjustments are discussed in this section. Airline passenger data from Section 2.2 is used as an example dataset to look at the effect of transformations in [7]. It is also used in Figure 7, which shows the data with different transformations. Table 1 summarizes ADF and KPSS stationary test results of the transformed data.

2.4.1 Removing heteroscedasticity

A common issue with datasets is heteroscedasticity, i.e., variance that changes over time. This behavior can be seen in the first panel of Figure 7. A widely used method for minimizing heteroscedasticity is a simple logarithmic transform $w_t = \log(y_t)$, where w_t is the transformed series [7]. As can be seen in the second panel of Figure 7, it stabilizes the variance while leaving the trend and seasonality intact. Logarithmic transform changes the scale of the data completely, but it is not an issue. Transformed data can be scaled back to the original with the inverse transform $y_t = e^{w_t}$. The trend still persists after the transformation, which causes the data to be non-stationary

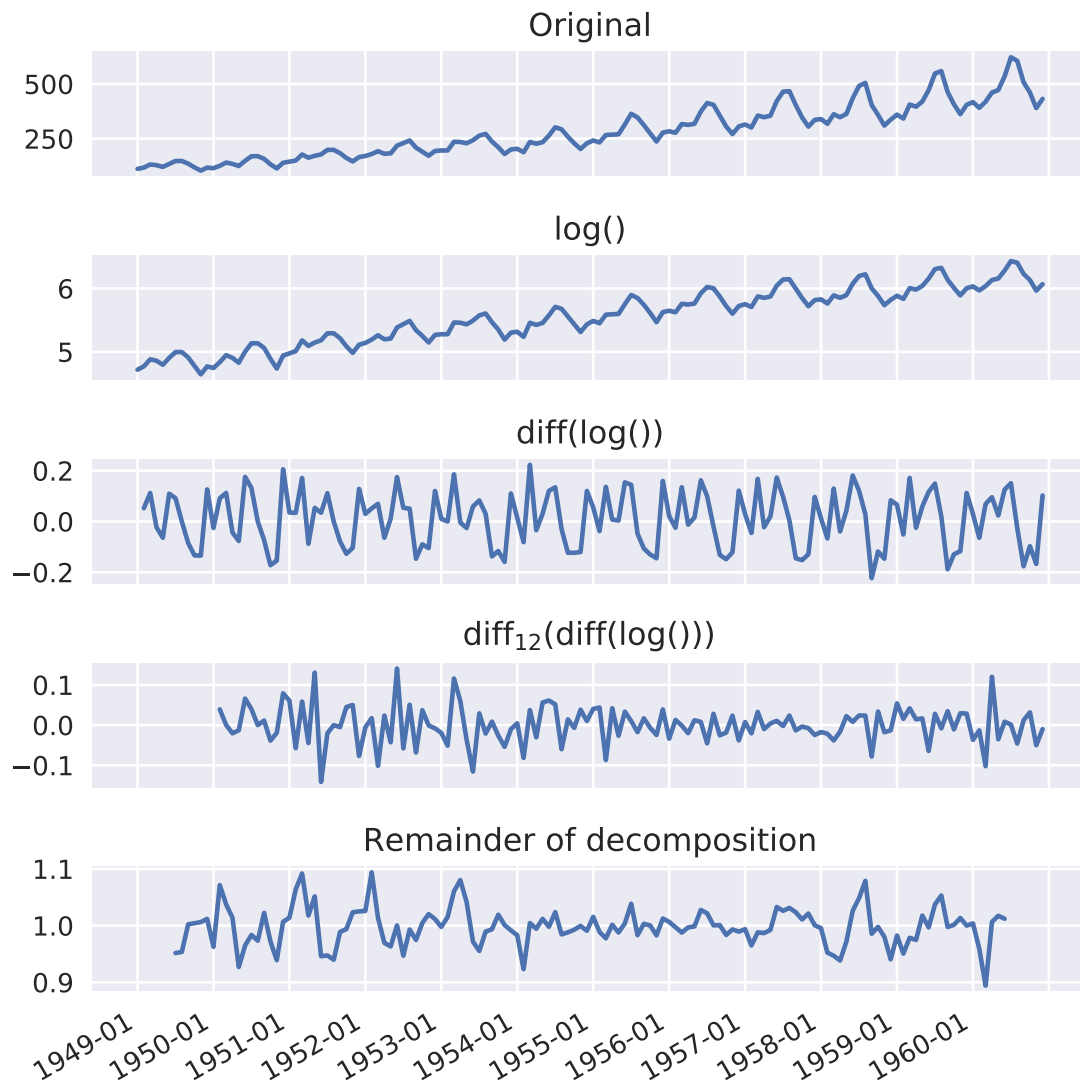


Figure 7: Different transformations applied to the airline passenger data.

Table 1: ADF and KPSS test results for the airline passenger data with different transformations. 5 % critical level is -2.89 for ADF and 0.46 for KPSS.

Transformation	Stationary (ADF/KPSS)	ADF Statistic	KPSS Statistic
no transformation	No/No	0.82	1.65
log()	No/No	-1.72	1.67
diff(log())	No/Yes	-2.72	0.04
diff ₁₂ (diff(log()))	Yes/Yes	-4.44	0.07
decomposition remainder	Yes/Yes	-7.42	0.04

according to both the ADF and KPSS tests. Logarithmic transform is also useful for constraining modeled values to be positive as the back-transformation produces only non-negative values. It is important to keep in mind that log transformation cannot be used if the data contains negative values.

Sometimes more adjustability is useful for minimizing heteroscedasticity in a time series. Box-Cox transform, first introduced in [19], can be used for producing a more optimal transformation between logarithmic and linear transformation of the data. Box-Cox transformation depends on the parameter λ and it is defined as follows:

$$w_t = \begin{cases} \log(y_t) & \text{if } \lambda = 0; \\ (y_t^\lambda - 1)/\lambda & \text{otherwise.} \end{cases} \quad (2)$$

In most cases the λ value is between 0 and 0.5, but it should be chosen so that heteroscedasticity is minimized [9]. With $\lambda = 0$ the transformation is equal to the logarithmic transform. With $\lambda = 1$ the transformed time series becomes $w_t = y_t - 1$ which is the original shifted down by one unit. Back-transforming is done with:

$$y_t = \begin{cases} e^{w_t} & \text{if } \lambda = 0; \\ (\lambda w_t + 1)^{1/\lambda} & \text{otherwise.} \end{cases} \quad (3)$$

If Box-Cox transformed data is used for making point forecasts which are then back-transformed back to the original scale, an issue arises in some cases [9]. Before the transformation, the point forecast is produced by taking the expected value $E(w_t)$ of the predicted distribution at each time step. Back-transforming the values of the point forecast does not give the expected values of the back-transformed distributions. If f is the back-transformation, i.e. $y_t = f(w_t)$, then $E(y_t) \neq f(E(w_t))$. The back-transformed point forecasts are often close to the median of the forecast distribution, rather than the mean [9]. This produces some bias to the forecasts which can be corrected by using the following bias-adjusted back-transformation instead of Equation 3:

$$y_t = \begin{cases} e^{w_t} \left[1 + \frac{\sigma_h^2}{2} \right] & \text{if } \lambda = 0; \\ (\lambda w_t + 1)^{1/\lambda} \left[1 + \frac{\sigma_h^2(1-\lambda)}{2(\lambda w_t + 1)^2} \right] & \text{otherwise,} \end{cases} \quad (4)$$

where σ_h^2 is the h -step forecast variance [9]. Using the median forecasts produced by Equation 3 is acceptable in simple cases, but if multiple forecasted components are added together, the bias-adjusted back-transform must be used [9]. In addition to logarithmic and Box-Cox transformations, also square roots and cube roots can be utilized if they produce better results with the dataset.

2.4.2 Removing trend

Removing the trend from a time series can be achieved with multiple methods. One of the most straightforward methods is to fit a linear line or other suitable smooth function to the dataset and remove it from the data. Ordinary least squares (OLS)

is a commonly utilized method for fitting functions to data. It minimizes the sum of the squares of the differences between the dataset and the fitted function. A linear trendline and detrended data are shown in Figure 8. While the trend in the data does not perfectly fit a linear line, the detrended data looks good. It is evenly distributed around zero, but the seasonality is still left untouched. A more complex function like a quadratic or polynomial would have produced a better fit, but it runs the risk of overfitting. Overfitting means that with enough variables, almost a perfect fit to the data could be achieved. However, such a model would not generalize well, and it would not work well with future data.

Another approach to removing trend is differencing. Differencing means subtracting the previous value from every value so that the differenced time series becomes $d_t = y_t - y_{t-1}$. Differencing can be repeated multiple times if the trend persists after one differentiation. In most cases 1-2 differencing operations are sufficient to make a time series stationary [10]. The example data was differenced only once in the third panel of Figure 7, but the result still looks good. The ADF test of the data indicates non-stationarity while KPSS test classifies it stationary as shown in Table 1. This disparity in the test results is caused by the remaining seasonal pattern in the data. It could be reduced by differencing it again. Differencing a dataset until it becomes stationary was recommended by Box and Jenkins in [10] and it is a valid strategy in most cases. However, there exists some more direct methods for addressing seasonality.

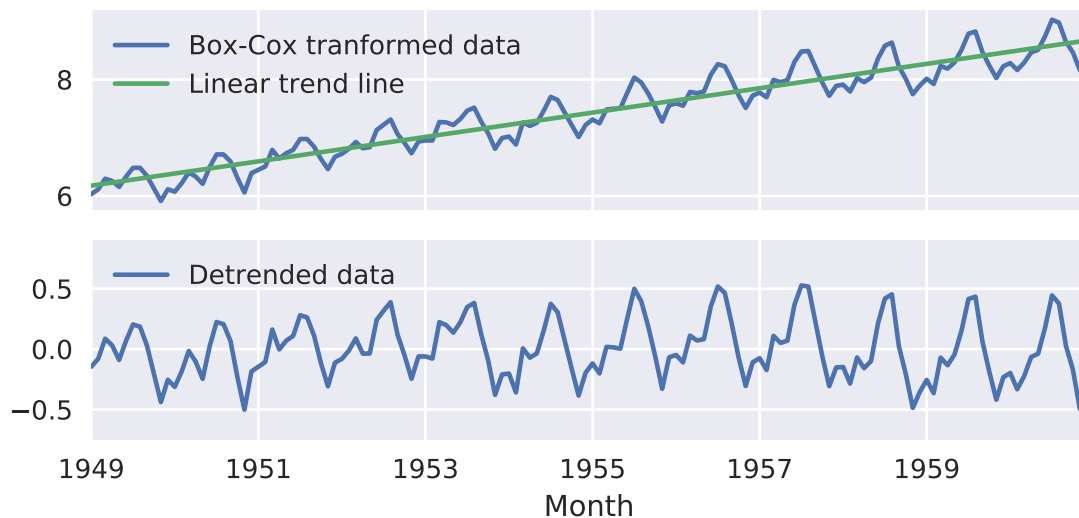


Figure 8: Linear trend line fitted using OLS. The dataset was Box-Cox transformed with $\lambda = 0.1$ to remove the heteroscedasticity.

2.4.3 Removing seasonality

A better approach to reducing seasonality is seasonal differencing which is illustrated in the fourth panel of Figure 7. It means subtracting the corresponding value from the previous season from every value according to $d_t = y_t - y_{t-s}$, where s is the length of the season [9]. Seasonal differencing with a 12 month period made the data look stationary which is confirmed by the ADF and KPSS test results shown in Table 1. In fact, both tests indicate stationarity even with 1 % critical level.

The last panel of Figure 7 shows the remainder component R_t from the multiplicative decomposition of Figure 3 as a comparison to the other methods. It is the remaining error factor after dividing out the effect of the extracted trend and seasonality components. According to the ADF and KPSS test results in Table 1, the remainder component is stationary with even better statistical evidence. Time series decomposition is presented in the next Section 2.5.

2.5 Time series decomposition

Time series decomposition is used for extracting trend and one or more seasonality components from a time series. As no real-world dataset can be perfectly decomposed, the remaining error is captured in the remainder component. If the decomposition is successful, the remainder should be a stationary time series which can be modeled with an appropriate model. Even if the remainder is non-stationary, transformations from the previous Section 2.4 can be used for making it suitable for modeling. By combining the models for the trend, seasonality and the remainder component, the time series can be understood and forecasted [9].

Additive decomposition is shown in Equation 5 and multiplicative decomposition in Equation 6:

$$y_t = T_t + S_t + R_t, \quad (5)$$

$$y_t = T_t \times S_t \times R_t, \quad (6)$$

where T_t is the trend, S_t the seasonality and R_t the remainder component. Multiplicative decomposition is chosen if the amplitude of the seasonality follows the level of the trend. Another valid approach is to remove heteroscedasticity with a logarithmic or Box-Cox transform and use the additive decomposition instead [7]. In fact, these approaches are related because by taking logarithms Equation 6 corresponds to

$$\log(y_t) = \log(T_t) + \log(S_t) + \log(R_t), \quad (7)$$

which is equivalent to Equation 5. Decompositions between multiplicative and additive can be achieved with a Box-Cox transformation followed by the additive decomposition and back-transformation of the components. $\lambda = 0$ is equivalent to multiplicative decomposition and $\lambda = 1$ is equivalent to additive decomposition [9].

Multiplicative decomposition for the airline passenger dataset was shown in Figure 3 in the previous section. Classical additive decomposition for Box-Cox transformed data is presented in Figure 10.

Classical decomposition is achieved with the steps visualized in Figure 9. First the data is transformed to be more compatible with the decomposition model if it is necessary. In this case a Box-Cox transform with $\lambda = 0.1$ was used for removing the increasing amplitude of the seasonality. The second step is to apply a rolling average filter to the data to smooth out noise and seasonality. Rolling average with a lagging window is defined as:

$$y_t = \frac{1}{n} (y_{t-n} + y_{t-n+1} + \cdots + y_t), \quad (8)$$

where n is the length of the rolling window. The window can be also centered around the current time step t , which ensures that the smoothed time series does not lag behind the original data. Window length of 2 times the season length is usually appropriate [9]. In this case a two-year window, i.e. 24-month window, was used which is visualized in red in the first panel of Figure 9. A centered window, i.e. averaging the values from the preceding and following years, was used so that the smoothed data does not lag behind the original data. This leaves the first and last year of data without values as they cannot be calculated. This was fixed by extrapolating the missing values using splines fitted with OLS as shown in Figure 9.

The second panel of Figure 9 shows the detrended data and the season component calculated from it. The detrended data is calculated by subtracting the smoothed trend component from the transformed data. The resulting time series oscillates around zero but is has no discernable trend. The seasonality component is calculated by averaging the corresponding observations from each season [9]. In this case, the average value for each month was calculated. It is important that the seasonality component does not affect the level of the time series over time. This is achieved by ensuring the sum of the seasonal component is zero over each season. To accomplish this, k is subtracted from each seasonal component S_i :

$$k = \sum_{i=1}^m S_i, \quad (9)$$

where m is the number of observations per seasonal period. The result in this case was an average value for every month of the year, which was then repeated through-out the dataset.

The next steps are deseasonalizing the transformed data and finding the trend component. These steps are visualized in the third panel of Figure 9. The deseasonalized data is produced by subtracting the seasonality component produced in the previous steps from the original data. This leaves a dataset that shows the trend with some noise. The trend component is then estimated with a rolling average whose window size can be chosen independently from the season. If forecasts are made based on the decomposition, it could be beneficial to model the trend component with an OLS-fitted function instead, as it can be easily extrapolated into the future. In this case a similar rolling average with a 24-month window was chosen.

The last panel of Figure 9 shows the remainder component of the decomposition. It is calculated by subtracting the trend and seasonality components from the transformed original data. Some seasonal pattern is still visible in the beginning and



Figure 9: Different steps used in classical decomposition of the airline passenger data. The dataset was Box-Cox transformed with $\lambda = 0.1$ to remove the heteroscedasticity.

the end of the remainder, which indicates that the magnitude of the seasonality was not quite flattened out with the Box-Cox transformation.

Figure 10 shows the original airline passenger data and back-transformed com-

ponents of the decomposition. As the inverse Box-Cox transform does not work with negative values, the seasonality component was produced by subtracting back-transformed trend component from back-transformed sum of the trend and seasonality components. The remainder was recalculated by subtracting the trend and seasonality components from the original data.

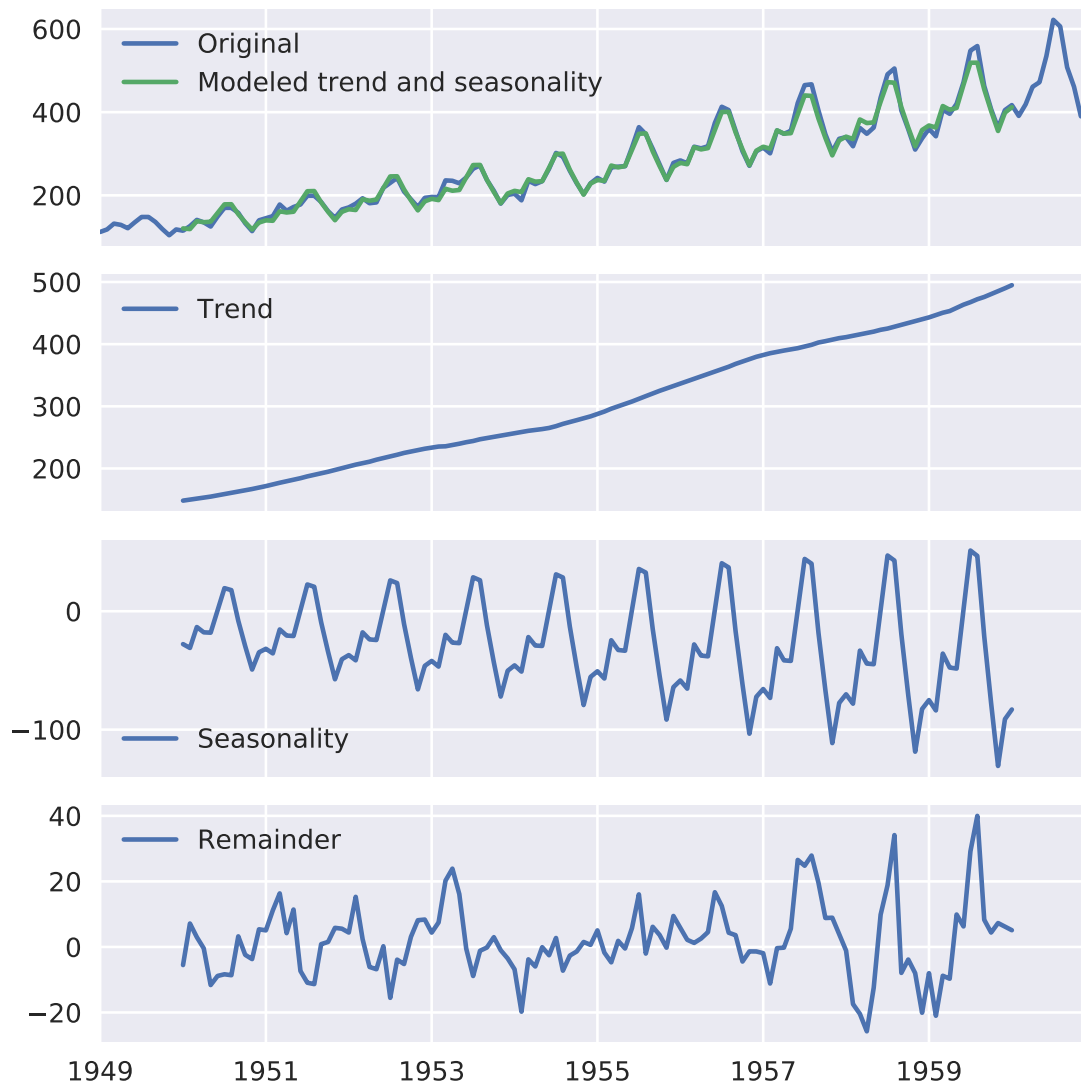


Figure 10: Back-transformed components from the decomposition shown in Figure 9.

Back-transforming the seasonality adds back the heteroscedasticity seen in the original data. The green plot in the first panel of Figure 10 shows the trend and seasonality components overlaid on top of the original data. Apart from a few small random fluctuations and slightly too small seasonal variation in the end, the modeled values follow the observed values well. As can be seen from the scale of the remainder

component, the errors are quite small compared to the scale of the original data. ADF and KPSS tests indicate that the remainder component of the classical decomposition is stationary.

While the classical decomposition is easy to understand and implement, it is not ideal for all datasets. Estimating the trend using a rolling average tends to smooth out rapid fluctuations in the data [9]. Calculating the seasonal component by averaging the values from every season does not allow the season to change over time. Outliers in the data can be very detrimental in calculating the season because there is usually quite few datapoints to average.

A widely used method for decomposition is Seasonal and Trend decomposition using LOESS (STL) which was first introduced in [20]. LOESS stands for locally estimated scatterplot smoothing, and it is a more advanced method for estimating non-linear relationships. STL decomposition overcomes many of the shortcomings of classical decomposition. It handles sudden changes in trend better, it supports changing seasonality and it has a robust method for handling outliers in the data [9].

While the implementation details of STL decomposition are not relevant in this thesis, it can be easily utilized with software packages like *forecast* [17] for R and *statsmodels* [18] for Python. The *forecast* package includes a variation called *mstl* which supports multiple seasonalities. This is especially useful for daily data which can have both weekly and yearly seasonalities.

2.6 Imputation

Real-world time series datasets may contain missing values that can make modeling and forecasting more difficult. Some modeling methods cannot be used with incomplete data. Missing data can be caused by human or technical errors that are unavoidable in some cases. The issues caused by missing data can be mitigated by filling the missing values with estimated values. This process is called imputation and it can be implemented in several ways.

Imputation is a common task in many fields of statistics and many implementations exist in statistical software packages like R. However, most of the methods cannot be directly applied to univariate time series data as they rely on the correlation of multiple variables in a cross-sectional dataset [21]. A univariate time series has only one variable which has a time dependence to itself.

The simplest methods for time series imputation include filling the missing values with some constant value, e.g., zero or the average of the available data. While this makes it possible to use modeling methods that require complete data, the imputed values might bias the model significantly. Filling the missing values with a constant ignores the time dependent nature of the data completely and thus does not utilize all the information available. In the terminology of Moritz et al. [21]., these types of imputation methods are called univariate algorithms.

Univariate time series algorithms are specifically designed for imputing time series data and they account for the time dependence of the data [21]. One of the simplest such approaches is last observation carried forward (LOCF), i.e., copying the last known value to the missing values. Another similar approach is next observation

carried backward (NOCB) which means copying the next known value to the missing values. These methods work better than the univariate algorithms if the level of the time series changes over time and the time series has autocorrelation. However, they do not properly account for trend or seasonality.

Interpolating the missing values using surrounding data is a simple method that accounts for a trend in the data. Handling seasonality requires more advanced methods like deseasonalizing the data using decomposition, imputing the values with linear interpolation and adding the seasonality back in. Another approach is to group the data according to the seasonality and interpolate using the corresponding values from each season.

The third category of methods is called multivariate algorithms with lagged data [21]. Imputation methods not directly suitable for univariate data can be utilized by creating multiple variables from different lags of the time series. This way the time dependence of the data can be utilized with algorithms that are not designed for time series data.

Moritz et al. [21] found univariate time series algorithms to perform the best. Multivariate algorithms with lagged data performed moderately as well but the simplest univariate methods showed poor performance. Bokde et al. [22] developed a new method to compete with the simpler univariate time series methods. It is based on Pattern Sequence Forecasting (PSF) which was first proposed by Martínez-Álvarez et al. [23]. In the first step, different types of patterns are searched for from the time series. In the second step, missing gaps of data are forecasted with the PSF method using the data before the gap and backcasted using the data after the gap. Imputed values are the average of the forecast and the backcast. This functionality is implemented in the *imputePSF* R package [24].

According to Bokde et al. the *imputePSF* method performs equally well or better compared other univariate time series imputation methods. It works the best with cyclic or seasonal data which has little noise. *imputePSF* was not tested in this thesis due to incompatibility issues. As the other available methods did not produce adequate results, a new imputation method was developed in this thesis which is described in Section 3.5. It is a combination of existing techniques that aims to model the weekly and yearly seasonality in the data, and account for the holidays using a calendar.

2.7 Time series models

Time series modeling is utilized in many fields as it can help in understanding unexplainable processes. By studying time series data, a suitable mathematical model can be chosen to describe it. Mathematical models have parameters that need to be estimated. Recorded past observations of the time series can be used for the estimation of model parameters. If the model fits the past data adequately, it can give useful insights to the underlying process that generated the data [25].

Time series modeling should always be started by exploring the data. Often visualizing the data with a plot and checking the minimum, maximum and average values are essential first steps. A histogram can be used for analyzing the distribution

of the data and it usually shows if there are clear outliers. It is also important to check if there are missing values in the data as they can bias the model if they are not handled correctly.

Making sure the minimum and maximum values seem sensible is important. Technical errors and limitations may cause peculiar values in the dataset. For example, a failed sensor reading might be recorded as -1 . If the measured value should be non-negative by its nature, this should cause some concern. In this case it would be appropriate to check how many samples were recorded as -1 and then mark them as missing values. If a modeling method that requires complete data is chosen, the missing values should be imputed before the modeling as discussed in Section 2.6.

Plotting the data can quickly reveal the common time series features discussed in Section 2.2. Detecting all the features is essential when selecting the modeling method applied to the data. Some of the simplest methods are discussed in the following Section 2.7.1.

2.7.1 Simple models

The simplest time series model is random i.i.d. noise at a constant level $y_t = c + \varepsilon_t$. The mean level c can be calculated by averaging the samples in the dataset. ε_t is modeled by calculating its variance using sample standard deviation. The random component ε_t is evident in all real-world time series data and it is caused by measurement errors and unexplainable factors in the process. After successful application of more complicated models, the remainder should always be i.i.d. noise, which indicates that the other models were successful in capturing all the deterministic behavior in the data.

As discussed in Section 2.3 unit root processes can be modeled with a random walk model. Random walk is a cumulative sum of i.i.d. noise i.e. $y_t = y_{t-1} + \varepsilon_t$. Random-walk models can be converted to random noise by simply differencing them once i.e. $y'_t = y_t - y_{t-1}$ [26].

If the data shows some characteristics that makes it non-stationary, transformations, detrending and deseasonalization described in Section 2.4 can be also added to the model. A simple model that captures trend and seasonality can be achieved with time series decomposition presented in Section 2.5. If the seasonality component S_t is assumed to stay constant over time, it can be simply extended into the future. Another approach to estimate seasonality is harmonic regression. Even complex seasonalities can be modeled as a sum of sine waves:

$$S_t = a_0 + \sum_{j=1}^k (a_j \cos(\lambda_j t) + b_j \sin(\lambda_j t)),$$

where a_0, a_1, \dots, a_k and b_0, b_1, \dots, b_k are parameters to estimate and $\lambda_0, \lambda_1, \dots, \lambda_k$ are fixed frequencies that correspond to the seasonality [7]. Harmonic regression also works with multiple seasonalities as the frequency components λ_k can be chosen to model multiple frequencies. With fewer estimated parameters the estimated

seasonality is smoother. For complex seasonalities, the number of parameters k can be increased to accomplish satisfactory performance.

With the model of the seasonal component, the time series can be deseasonalized. To complete the model, the trend component can be estimated by fitting a suitable function with the OLS method. Trend that is modeled with a function can be easily extrapolated into the future, to generalize the model and enable forecasts.

While this type of model is easily explainable and it can perform well, it might still leave some of the available data unutilized. As stated earlier, many real-world time series contain autocorrelation. As can be seen in the remainder component R_t in Figure 10, there is still some seasonality and autocorrelation that is not captured by this simple model. The remainder component could be modeled with, e.g. some of the stochastic models introduced in Section 2.8, to make sure that all deterministic information in the data is utilized by the model.

2.7.2 Model parsimony

Following sections introduce more sophisticated models that have more parameters to estimate. While choosing a suitable model, the principle of parsimony should be kept in mind. With an infinite number of parameters, a model can perfectly fit any time series data. However, such a model would not generalize the data as it would have been overfit. Future measured values of the time series would probably not fit the model very well and forecasts made with the model would not be accurate. Overfit model has captured the random noise component in addition to the deterministic components which causes problems with unseen data [27].

The principle of model parsimony states that the model with least parameters to estimate, which is still suitable to represent the underlying process should be chosen [25]. Finding the balance between a good fit to the data and model parsimony is not a simple task. Several modeling error metrics that penalize for a large number of model parameters have been developed to help in model selection. Some of them are presented in Section 2.10.2.

2.8 Stochastic models

Many stochastic models are still widely used for time series modeling. The most common are exponential smoothing (ETS) from the late 1950s [28, 29] and ARIMA from the early 1970s [10]. Traditional ETS models produce only point forecasts, but there is also a stochastic interpretation which produces prediction intervals [9]. Many stochastic models utilize the autocorrelation found in almost every time series. The basic concept is to model the current value as a linear combination of the past values. While these stochastic models are very suitable for many time series, they have some inherent drawbacks. They usually do not support non-linear relationships between the lagged values of the time series. They also assume that the time series is stationary and follows a known probability distribution [25]. Transformations and seasonal adjustment are often required before applying stochastic models to real-world datasets.

2.8.1 Exponential smoothing (ETS)

Exponential smoothing (ETS) uses an exponentially decreasing weighted average to model the time series. The most recent observation is assigned the largest weight and the effect of higher lags tapers off. Simple ETS model is described by the following equation:

$$\hat{y}_{t+1|t} = \alpha y_t + (1 - \alpha)\hat{y}_{t|t-1},$$

where $0 \leq \alpha \leq 1$ is the smoothing parameter, $\hat{y}_{t+1|t}$ is the next and $\hat{y}_{t|t-1}$ the previous time step of the exponentially smoothed time series [9]. Choosing a high value for α focuses on the newest observations while a low value assigns more equal weight to older observations. The process needs an estimated initial value for $\hat{y}_{-1} = \ell_0$ to get the calculation started. For the simple ETS method, only the smoothing parameter α and the starting value ℓ_0 need to be estimated. It can be done by hand but a more robust method is to minimize the sum of squared modeling error $\sum_{t=1}^T (y_t - \hat{y}_{t|t-1})^2$ [9].

If the time series contains a trend or seasonality, so called Holt and Holt-Winters methods are required. They add additional parameters to model the trend and seasonality. The best way to formalize the Holt's linear method is to use the component form notation for h -step-ahead forecast:

Forecast equation	$\hat{y}_{t+h t} = \ell_t + hb_t$
Level equation	$\ell_t = \alpha y_t + (1 - \alpha)(\ell_{t-1} + b_{t-1})$
Trend equation	$b_t = \beta(\ell_t - \ell_{t-1}) + (1 - \beta)b_{t-1},$

where ℓ_t is the estimate of the level of the time series and b_t is the estimate for the trend at time t . $0 \leq \alpha \leq 1$ is the smoothing parameter for the level and $0 \leq \beta \leq 1$ is the smoothing parameter for the trend. [9] The forecast equation with $h = 1$ is used for producing the next exponentially smoothed estimates for ℓ_t and b_t until the end of the time series is reached. Parameters α , β , ℓ_0 and b_0 are estimated from the data in a similar fashion to the simple ETS method.

Holt-Winters' seasonal method adds a third smoothing equation for the seasonal component s_t . The smoothing parameter for seasonality is γ and the frequency of the season is m . For monthly data $m = 12$ as there are 12 observations per season. Daily data can be modeled with $m = 7$ to capture the seasonality caused by day of week. [9] At least one complete season of data is required to initialize the seasonal component. To estimate the seasonal smoothing factor γ properly, the length of the time series should be at least $2m$ [30].

The seasonal component can be either additive or multiplicative depending on the nature of the time series. Like in time series decomposition, additive seasonality is suitable if the seasonal variations do not depend on the level of the time series. Multiplicative seasonal component is required if the seasonal variation changes in proportion with the overall level of the time series. The component for representation

of the additive method is:

$$\begin{aligned}\hat{y}_{t+h|t} &= \ell_t + hb_t + s_{t+h-m(k+1)} \\ \ell_t &= \alpha(y_t - s_{t-m}) + (1 - \alpha)(\ell_{t-1} + b_{t-1}) \\ b_t &= \beta(\ell_t - \ell_{t-1}) + (1 - \beta)b_{t-1} \\ s_t &= \gamma(y_t - \ell_{t-1} - b_{t-1}) + (1 - \gamma)s_{t-m},\end{aligned}$$

where the integer k is chosen such that the seasonal indices for the next step come from the previous season. The level equation ℓ_t is seasonally adjusted by subtracting the seasonal component from the observed data before applying the exponentially decreasing weighted average of past observations. [9] The multiplicative version is similar but the deseasonalizing is done by dividing the observations by the seasonal component.

In most cases, minimizing the error of one-step-ahead forecasts is used for optimizing the model [31]. This is a complex non-linear problem which requires numerical methods. Fortunately, most software packages designed for time series analysis include implementations for optimizing the parameters.

2.8.2 Autoregressive models (AR)

Autoregressive (AR) models use a linear combination of past observations to model the time series. The number of lags p can be chosen by comparing the modeling errors with multiple values. Analyzing a PACF plot of the data is also useful as the optimal value for p is often the number of significant lags in the PACF plot [7]. The weights associated with each of the lags ϕ_1, \dots, ϕ_p can be estimated from the data with maximum likelihood estimation (MLE). MLE maximizes the probability of the observed values fitting the model [9]. AR(p) i.e., autoregressive model of order p is defined as:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \dots + \phi_p y_{t-p} + \varepsilon_t,$$

where c is a constant level, and ε_t the remaining error term [9]. ε_t is assumed to be i.i.d. noise. AR models can be fitted to many stationary datasets, but they are often combined with a moving average model which is presented in the next section.

2.8.3 Moving average models (MA)

Instead of past observations, moving average (MA) models use a weighted average of past modeling errors to produce the time series model. $\theta_1, \dots, \theta_q$ are the weights for the past modeling errors and q is the order of the MA(q) model. Moving average model of order q is defined as:

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \dots + \theta_q \varepsilon_{t-q},$$

where $\varepsilon_{t-1} \dots \varepsilon_{t-q}$ are the past errors and c is a constant level [9]. MA models should not be confused with moving average smoothing that was used in the time series decomposition in Section 2.5. Similarly to AR models, the weights θ can be determined by minimizing the sum of squared modeling errors over the whole time

series. Choosing the number of lags q can be done by testing the performance with different values or by looking at an ACF plot of the data. The number of significant lags in the ACF plot is often a good starting point for q [7]. The next section presents models that combine AR and MA models which work great in tandem.

2.8.4 ARMA, ARIMA and SARIMA models

ARMA models are a combination of autoregressive and moving average models. They are suitable for many stationary time series. A related model is called ARIMA, where I stands for integrated. It simply means differencing which was introduced in Section 2.4. The order of differencing is denoted by d . ARIMA(p, d, q) model is defined as:

$$y'_t = c + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t,$$

where y'_t is the time series differenced d times. Differencing enables the modeling of non-stationary time series which become stationary with the differencing. The value of d should be chosen so that the time series becomes stationary, which is often achieved with $d = 2$ or lower [9]. AR and MA models can be interpreted as being ARIMA(p, d, q) models with the unneeded parts of the model left out by setting p, d or q to zero as needed.

While adding differencing to the ARMA model can help with non-stationarity and trends, ARIMA models do not handle seasonality. Seasonal ARIMA (SARIMA) was developed to handle time series with seasonality. In addition to the (p, d, q) terms that work with the previous observations, SARIMA models add $(P, D, Q)_m$ terms that work with observations one season length m away from the current value. The seasonal terms are similar to the non-seasonal terms except that they work with observations from the previous season. Non-seasonal and seasonal terms are added together to produce the SARIMA model.

2.8.5 Box-Jenkins methodology

Choosing the order of the ARIMA(p, d, q) can be complicated as there are three parameters to optimize. In addition to the quality of fit, also model parsimony needs to be kept in mind to prevent overfitting. Box and Jenkins developed a simple three step process to guide in the selection process. This so-called Box-Jenkins methodology was first presented in [10]. The three steps are called model identification, parameter estimation and diagnostic checking.

In the first step, model identification, an estimate of the needed parts of ARIMA model is made. ACF and PACF plots and unit root tests can indicate if the autoregressive part, moving average part or differencing is needed. When a tentative model is chosen, the parameters p, d, q are estimated according to the chosen model. Initial estimates for p and q can be made based on the ACF and PACF plots. The number of differencing operations d is increased until the data becomes stationary. The last step, diagnostic checking includes statistical tests that measure the performance of the tentative model. Suitable diagnostic metrics are Akaike Information Criterion (AIC)

and Bayesian Information Criterion (BIC) which are presented in the Section 2.10.2. These steps are repeated iteratively until the diagnostics indicate that the model is satisfactory. [25]

Many software packages include automatic functions for finding optimal ARIMA model parameters, e.g. *auto.arima* [32] in the *forecast* [17] package for R. They use a stepwise search algorithm to test different p, d, q parameters within predefined ranges and compare their performance with different error metrics that take model parsimony into account [32]. Weights and variance of the error term are also automatically fitted using MLE. Fitting dozens of models automatically and comparing them with a performance metric is usually a robust method for coming up with an optimal model. However, as the number of combinations is vast, the automatic algorithms do not test all combinations. This means that they do not always find the global optimum model.

2.8.6 TBATS

TBATS, presented in [33], is a quite new method which combines many of the methods described in the previous sections. It is a completely automated method which uses harmonic regression with an exponential smoothing state space model and a Box-Cox transformation. The modeling errors from the ETS model are fed into an ARMA model to capture the remaining deterministic information. Opposed to normal harmonic regression, TBATS allows the seasonality to change over time. Similarly to harmonic regression, complex seasonal patterns with multiple frequencies are supported [9]. TBATS is implemented in the *forecast* [17] R package.

2.9 Machine learning

Machine learning has gained popularity in the past decade as more data and computing power has become available. Machine learning models can be thought as extremely complex non-linear functions that are fitted with large amount of data. They do not assume any known distribution or features in the data, but their data-driven approach can learn complex features. Their non-linear nature gives an advantage in comparison to the linear methods presented in the previous section. However, machine learning methods are complex and often not very interpretable or explainable. Sections 2.9.1-2.9.6 introduce different machine learning methods following Géron [34].

Most machine learning methods are not designed for time series data, but some of them have been successfully applied to time series data with some data transformations, e.g. [3, 35]. To capture the time series nature of the data, it is common to provide a series of lagged values as input to the model, i.e. $\hat{y}_t = f(y_{t-1}, \dots, y_{t-n})$ [9]. Some models can benefit from engineered features such as day of week or a Boolean variable indicating whether the day is a holiday. Such features can be easily computed from the observation date and they are also known for future predictions. It is common practice to scale the input values within $[-1, 1]$ or $[0, 1]$. This is done so that the scale difference of different variables does not cause bias to the model.

The model could learn to overcome the bias, but the training is faster if the all the features have standard scale [26]. Another benefit of scaling the input values is that it enables the generalization of models to different datasets. It is common to use an existing model that was trained with a similar dataset as a starting point and continue training it with an application-specific dataset. This way the training time can be made significantly shorter, and the amount of data required is greatly reduced [36].

All machine learning methods require training data to fit the model. In order to test the performance of the model, a separate test dataset is used. Depending on the model and the amount of data available, the dataset is split to training and test data. For example, 80 % of the data is used for the training process and the last 20 % is used for testing the model. It is important to test with data that the model has not seen in the training phase in order to ensure that the model generalizes and is not overfit. Another concern is that the model is manually tuned based on the prediction performance of the test dataset, which might also cause overfitting. In order to validate that the model has generalized to the data, a separate validation set is commonly held out until the development of the model is finished. If the model performs well with data not used during its development, it likely performs well with future real-world data. This split to training, test and validation datasets is also utilized with the other time series modeling methods used in this thesis.

2.9.1 Neural networks

Nowadays, neural networks are one of the most popular machine learning methods. They mimic the working principle of the human brain in order to learn from a large set of data. A simple neural network is the basis of neural network (NN) and deep learning-based models. Such an architecture is presented in Figure 11. It consists of multiple layers which have one or more neurons. All of the neurons are linked with weights that are presented by blue lines in Figure 11. The outputs from each of the layers are used as inputs for the next layer until the last layer gives out the desired output. The inputs and outputs of a single neuron are real numbers that get multiplied by the weights connecting the neurons.

The first layer is the input layer which has as many neurons as there are input variables in the model. Figure 11 has 7 inputs which could be the last 7 observations of a time series for example. The last layer of the neural network is always the output of the model. The number of neurons in the last layer corresponds to the desired amount of output variables. For example in Figure 11 the single output could be a prediction for the next value of the time series.

The layers between the input and output layers are called hidden layers. The number of hidden layers and the number of neurons in them can be chosen arbitrarily. Neural networks which have more than one hidden layer are called deep neural networks (DNN). More layers and neurons enable the model to learn more complex patterns at the cost of increasing the training time and possibly causing overfitting.

Each node in the layer gets its inputs from the nodes of the previous layer. The input is the sum of the incoming connections multiplied by the weights that

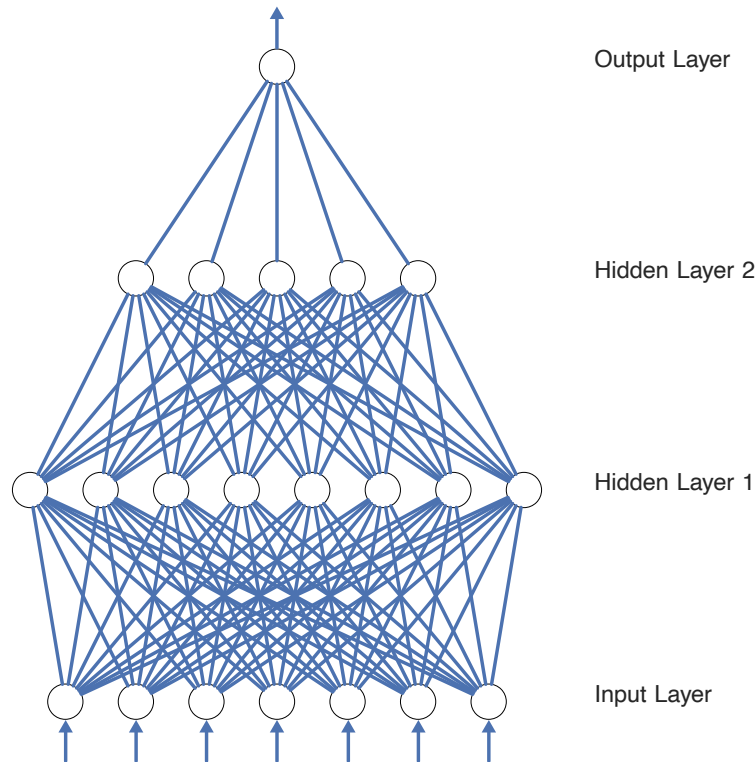


Figure 11: A neural network with 7 input neurons, 2 hidden layers with 8 and 5 neurons and a single output neuron.

correspond to the connections between the nodes. The result of the weighted sum is fed into a non-linear activation function which adds non-linearity and scales the values. A traditional activation function is sigmoid $s(z) = \frac{1}{1+e^{-z}}$, which keeps the value between zero and one [9]. A simpler activation function that is faster to compute is rectified linear unit (ReLU) which means simply replacing negative values with 0.

The weights start with random values that are optimized in the training process to minimize the modeling error [9]. The training process utilizes an algorithm called backpropagation to update the weights based on gradient descent. A gradient of the cost function, i.e. the modeling error, is calculated with the training data. All the weights are shifted slightly towards the direction of greatest descent to make the cost function smaller. This process is repeated for the desired amount of iterations, called epochs. The training process should find a local minimum of the cost function and produce a model that generalizes to new data. This can be achieved by training the model until the modeling error stops decreasing.

The weights are altered according to a parameter called learning rate. If the learning rate is too high, the model training might be erratic and jump between multiple local minima. Too small learning rate might not find an optimal local minimum and the learning requires more epochs.

2.9.2 RNN

The simple neural network architecture that was presented in the previous section can work with very short and simple time series. Feeding lagged values of the time series to the input neurons of the neural network enables the model to learn simple time dependence of the observations. However, the model has no memory of the past time steps which means that it does not know anything about the time series from lags higher than the number of input neurons.

Recurrent neural networks (RNN) were developed to enable the modeling of longer sequential data like time series [27]. They are very similar to the architecture presented earlier, but the input of each neuron consists of a sum of the current input and the output of the neural network from the previous time step. The output from the previous time step contains the outputs of all the previous time steps due to the recursive nature of the learning process. This means that the model has a memory of the whole dataset. However, the effect of past observations starts to decay quite quickly and the effective memory depth of one RNN neuron is about 10 time steps [34]. The next Section 2.9.3 presents LSTM neural networks which have a significantly better memory compared to RNNs.

Each neuron in the RNN has two sets of weights, one for the current input value and another for the output of the previous time step. The weights are learned similar to normal neural networks, but the recursive nature of the model needs to be unrolled through time. The weights are adjusted starting from the latest time step propagating backwards towards the first time step. Like regular neural networks, also RNNs can contain multiple layers and an arbitrary number of neurons per layer. If the time series contains seasonality, it is beneficial to also include inputs from the previous season.

2.9.3 Long short-term memory (LSTM)

Long short-term memory (LSTM) neural networks are an adaptation of RNNs that can be used to model longer sequences. They were first proposed in [37] and they have since been further improved. RNNs with long-term memory have proven so useful that regular RNNs are rarely used anymore [34].

The main idea behind LSTM neurons is that they have a long-term and a short-term state. At every time step, some of the unnecessary information is forgotten in the long-term state and some new information is learned. The short-term state is produced with the long-term state and it represents the modeled value for the current time step.

Implementation details of RNN and LSTM models are out-of-scope of this thesis, but most machine learning frameworks like *keras* [38] have implemented LSTM neurons which makes them easy to use.

2.9.4 Decision tree models

Decision tree-based methods are commonly used machine learning algorithms in real-world applications [27]. They are flexible, fast and easy to interpret which makes

them a desirable option for many use cases.

Decision trees can be used for both classification and regression. They divide the dataset to a tree-like structure based on the input variables. The resulting structure is non-linear, and it does not assume any known distribution in the data. One of the main advantages of decision trees is their easy interpretability [27]. A decision tree used for classifying iris plant species based on petal width and length is shown in Figure 12. It shows clear rules for dividing the data to come up with the three different classes. Resulting decision boundaries are shown in Figure 13.

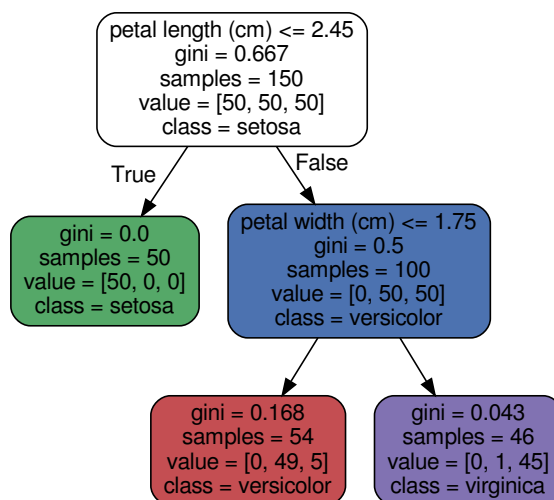


Figure 12: A decision tree that classifies iris plant species based on petal width and length. Adapted from [34].

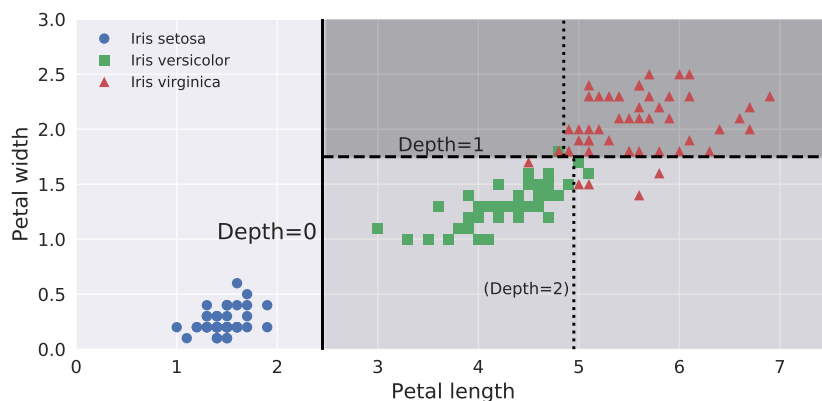


Figure 13: Decision boundaries for the different iris species. Adapted from [34].

Gini parameter in all the nodes indicates impurity of the class division, i.e., it measures how successful the model was in the classification. It is calculated with

$G_i = 1 - \sum_{k=1}^n p_{i,k}^2$, where $p_{i,k}$ is the ratio of class k samples among the training samples in the i^{th} node. Gini impurity of zero indicates that all the training samples in that node are in the same class, i.e., classified correctly. The samples parameter tells how many training samples of each category were classified to each node. The class parameter indicates which class the node predicts. The class of each node is determined by counting the classes of the training samples in each node and choosing the class with the most samples.

One of the most popular algorithms used for building the tree structure is the classification and regression tree (CART). It is a so-called greedy algorithm as it tries to find the optimal splitting point at each splitting depth. The algorithm tries to minimize the sample impurity, i.e. gini impurity, when splitting the nodes. This does not guarantee that the resulting tree structure is optimal, just that every split in the tree minimizes sample impurity. This is implemented by finding the optimal splitting point for every input variable and choosing the one that produces the purest subclasses weighted by their size [39].

If the algorithm is not regularized, it can make so many splits that every sample in the training data is categorized correctly. However, as the data is commonly noisy, finding a perfect fit is usually impossible without overfitting the training data. To prevent overfitting, it is common to limit the depth of the tree and set a minimum number of training samples per node. If a split is made only to separate few training samples from the whole set, it is likely reacting to noise, and thus overfitting the model.

Decision trees can be used for regression in a way that is very similar to classification. Decision tree used for regressing a noisy quadratic function is shown in Figure 14 and the results of the regression in Figure 15. The regression is a stepwise function that minimizes the sum of squared error. Increasing the depth of the tree, increases the resolution of the end result. Again, if the decision tree is left unregularized, it easily overfits and picks up noise and outliers. This is one of the drawbacks of decision trees which can be overcome with ensemble methods discussed in the next section.

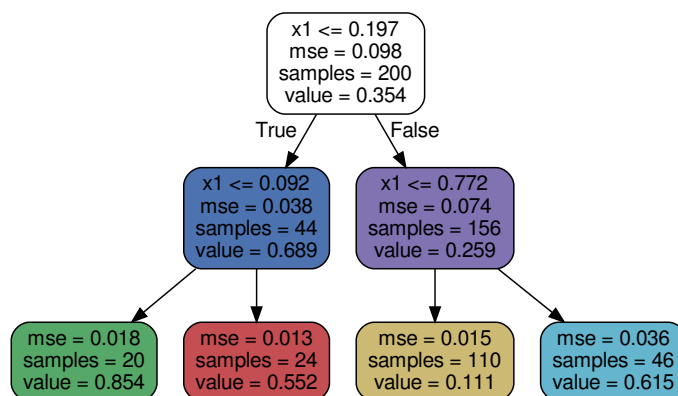


Figure 14: A decision tree that is used for regression. Adapted from [34].

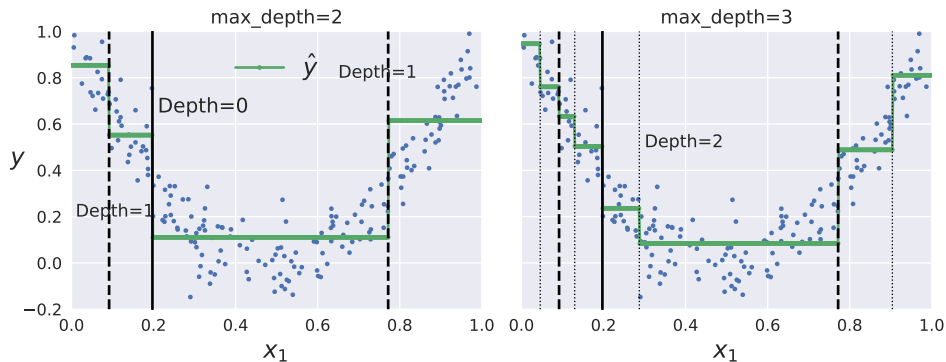


Figure 15: Two decision trees with different depths fitted on noisy quadratic data. Adapted from [34].

2.9.5 Random forests

Aggregating the result of multiple models and using it for predictions is called ensemble learning. Training the same model multiple times with a random subset of the training data is called bagging which stands for bootstrap aggregating. Ensemble methods are a common way to improve prediction performance as it simple to combine different models that were tested during the development of the model [27].

A commonly used ensemble method is random forest which uses multiple decision trees with bagging. The decision trees are trained in parallel to produce the ensemble model. The final prediction in classification tasks is made by choosing the class that was voted by most of the trees. Regression tasks are predicted by averaging the result of the parallel trees. This way the effect of noise and outliers is mitigated. Using multiple parallel decision trees enables getting probabilities for the different classes or regression outputs. By using a weighted average or weighted votes, the effect of confident predictions can be made higher than uncertain predictions [34].

While the interpretability of random forest models is not as great as with single decision trees, they have a nice feature which helps in selecting the input variables. Random forests can show the relative importance of each input variable. This is done by calculating how much the tree nodes using that input variable to split the data reduce the impurity of the subclasses. This way the model can be first trained with a large number of input variables and then the most useful ones can be chosen for the final model. In case of time series data, the inputs can be lagged values and calculated values like the day of week and the day of year [34]. Random forests are widely used because they are very versatile and work with many types of data. An implementation of random forests is included in for example the *scikit-learn* [40] library.

2.9.6 Gradient boosting

Boosting is an ensemble method in which multiple models are arranged sequentially as opposed to bagging where they are in parallel. Each successive model tries to

correct the errors of their predecessor. The later models in the sequence focus only on the parts of the data that the earlier models failed to model.

Overfitting is a concern with this method, so regularization is required. One suitable method is to keep track of the modeling error as more models are added to the sequence. If the modeling error stops decreasing significantly, the training should be stopped. This is called early stopping [27].

A widely used modeling technique called gradient boosting was first introduced in [41] and further improved in [42]. It uses sequential decision trees that correct the residual errors of the previous trees. Early stopping and adjusting the weight of each sequential decision tree are valid methods for regularizing gradient boosting models.

Gradient boosting is implemented in the *scikit-learn* [40] python library. A more advanced version called Extreme Gradient Boosting is implemented in a python library called *XGBoost* [43]. It has a similar API to *scikit-learn* but it has been optimized to be faster and more scalable. It includes an automatic implementation of early stopping which makes it simple to use.

2.10 Forecasting

Forecasting is one of the most useful use cases for time series models. It can give insight into the future of processes and systems which can be invaluable when making decisions. The following sections focus on how to produce forecasts with time series models and how to validate their performance.

2.10.1 Producing forecasts with a model

Producing forecasts with a model is usually done one time step at a time depending on the model. For simple models, the prediction for the next observation is predicted by simply feeding the model with the past observations and setting the error term ε_t to zero as the random error cannot be predicted in advance. If a long-term forecast is required, the predictions are added to the end of the observed data and the model is applied again. As the forecast horizon, i.e. the number of time steps predicted gets larger, the uncertainty of the forecast increases. This is caused by the modeling error adding up when the predicted values from earlier time steps are added as inputs to the time series model. Stochastic models can predict probability distributions for future time steps using the past observations as input. Expected value of the predicted distribution is the point forecast and the error can be derived from the distribution. Some models utilize information that is known in advance, e.g. holidays and the days of week, which can be calculated using a calendar [9].

The uncertainty of the forecast is almost as important as the forecast itself. Without knowing the prediction intervals, the value of the forecast is limited. Prediction intervals can be directly calculated for some of the simpler linear stochastic models. However, more complex models cannot output any statistical measure of their certainty. In such cases bagging is a useful tool. By making hundreds of forecasts with random subsets of the data, the distribution can be estimated [9].

2.10.2 Evaluating forecast accuracy

An important measure of performance is to calculate the forecasting error with a test set of data. The test set is usually taken from the end of the dataset before fitting the model. Making predictions on the test set simulates forecasting in real production environment where the future values are unknown. Comparing the forecast with the actual observations, a clear picture of the performance of the model can be seen [27].

Different error metrics are used for comparing the forecasted values and actual observations. There is no one perfect error measure that can be used as is. Different error metrics reveal different things about the model performance. To get a good understanding, multiple error metrics need to be calculated and compared critically. Some error metrics suitable for evaluating point forecasts are listed below.

Error metrics that measure absolute error penalize both positive and negative errors, but they do not reveal if the model predicts too high or low on average. Mean errors show bias, but positive and negative errors cancel each other out. Percentage errors are scale independent which is helpful for data that has a trend or large fluctuations in level. This way the error metric does not bias the parts of data with higher level [9]. Some of the most common error metrics are presented in the following list:

Root mean squared error (RMSE) is defined as:

$$RMSE = \sqrt{\frac{1}{T} \sum_{t=1}^T (\hat{y}_t - y_t)^2},$$

where \hat{y}_t is the predicted value, y_t the observed value and T the length of the time series. It has the same scale as the data and positive and negative errors do not cancel each other out. These properties make RMSE a good starting point for evaluating the magnitude of forecasting errors. Sometime RMSE is normalized by dividing it by the mean value of the observations \bar{y} . This indicates the scale of the error in relation to the scale of the time series. This makes it easier to compare the performance between datasets with different scales.

Mean absolute percentage error (MAPE) is defined as:

$$MAPE = \frac{1}{T} \sum_{t=1}^T \left(100 \left| \frac{\hat{y}_t - y_t}{y_t} \right| \right).$$

MAPE is scale independent, as every error is divided by y_t . This is useful as it ties the magnitude of the errors to the overall level of the time series. However, if the time series goes near zero, the relative error might become very large and unstable. As the absolute values of the relative errors are averaged, positive and negative errors do not cancel out.

Mean bias error (MBE) is defined as:

$$MBE = \frac{1}{T} \sum_{t=1}^T (\hat{y}_t - y_t).$$

It shows if the forecasted values have bias, i.e., they are on average too high or too low. As positive and negative errors cancel each other out, MBE is not useful for estimating the magnitude of the error. However, having significant bias in the forecasts will cause systematic errors in the forecasts. This makes MBE an important metric.

Akaike information criterion (AIC) is defined as:

$$\text{AIC} = -2\log(L) + 2k,$$

where L is the likelihood of the data and k is the number of parameters in the model. k includes σ^2 , i.e., the variance of the residuals as an estimable parameter. AIC error penalizes models with a large number of parameters because they are likely to cause overfitting. Choosing the model with the lowest AIC will probably work the best with out-of-sample predictions [9].

Schwarz's Bayesian Information Criterion (BIC) is defined as:

$$\text{BIC} = -2\log(L) + k\log(T).$$

It is similar to AIC but it favors even less model parameters. Choosing a model based on BIC is either the same model as would be chosen by AIC or a model with fewer parameters [9].

Prediction intervals can be quite simply evaluated for forecasting methods that produce them. If 95 % prediction intervals are calculated for step-ahead forecasts, approximately 95 % of the observed values should be inside the prediction intervals. More formal statistical tests exist for measuring the forecasting performance of stochastic models, but they are not used in this thesis. A commonly used metric is logarithmic scoring rule [44].

When comparing multiple models, it should be relatively simple to choose the best-performing one based on the above error metrics. However, it is important to keep in mind that the error metrics themselves could favor some models more than others. The achievable accuracy is largely dependent on the data which means that the performance is difficult to judge just with the calculated error metrics. To get an understanding of the performance, the models can be compared against a benchmark. A simple benchmark is a so-called naïve model. It predicts all future values as the last known value. For seasonal time series the corresponding value from the last available season is used as the prediction. If the more complicated model does not outperform the naïve model, it probably is not working adequately.

Cross-validation is a good method for estimating forecasting error for forecast horizons significantly shorter than the test set. An illustration of cross-validation is shown in Figure 16. The error calculation advances one time step at a time and the errors are averaged from every time step. The first forecast horizon of data is predicted using the training data. Then the forecast error from the first predictions is calculated. Then the same process is repeated one time step further so that the first value of the test set is added to the training set and the forecast horizon is shifted by one. Averaging the errors for each time step tested gives a reliable estimate of the performance of the model.

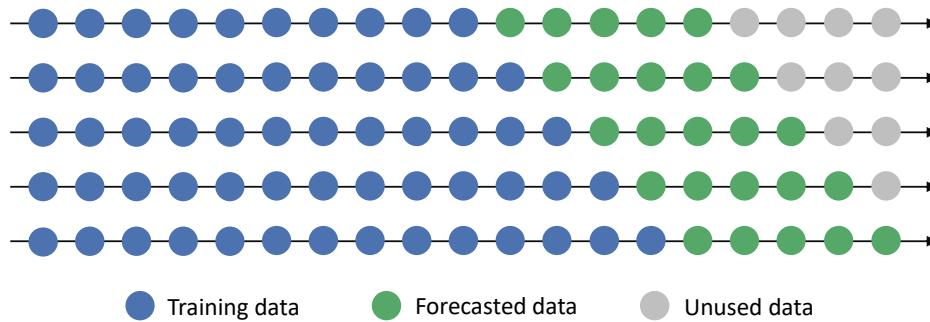


Figure 16: Illustration of cross-validation with a forecast horizon of 5 time steps. Adapted from [9].

2.11 Anomaly detection

Time series data is not only useful for simulation, filtering and forecasting. It can also reveal sudden changes in the phenomenon or process the time series measures. While time series analysis does not give a clear answer to what caused the change, it is useful to automatically detect such events so they can be further investigated manually.

An example of a clear anomaly in time series data is shown in Figure 17. It shows the number of elevator calls in an office building during the first half of 2020. The global COVID-19 pandemic caused a drastic change in the working environment as remote working became recommended. It caused the number of elevator calls to plummet overnight. This anomaly can be thought of as a persisting structural change of the time series. Another type of anomaly is a point outlier which affects only one or a few observations [45]. They can be caused by technical errors or unexpected events in the process of the time series measures. Automatically detecting different types of anomalies can be implemented in several ways.

A simple method is to compare each observation to a rolling average and standard deviation of the time series. If the new observation is for example more than 2 standard deviations away from the rolling average, it is likely an outlier. Anomaly detection in time series with strong seasonality can be implemented with a seasonal rolling average, e.g., comparing new observations to the previous observations from the same day of week. Holidays and other explainable anomalies will probably cause false detections, but they can be quite easily filtered out with a calendar.

The same time series models used for forecasting can also be utilized for anomaly detection. If an observed value differs from a forecasted value significantly, i.e., it falls outside of the prediction intervals, it is likely an outlier. Estimation based methods compare each observation with observations preceding and following it. Prediction based models compare the newest observations only to the previous observations which makes them compatible for detecting anomalies as soon as they happen [45]. Literature has many examples of different time series models used for anomaly detection. STL decomposition was used in [46] and ARIMA models with exogenous variables in [47].

Some R packages have been developed to make anomaly detection easier. Twitter's

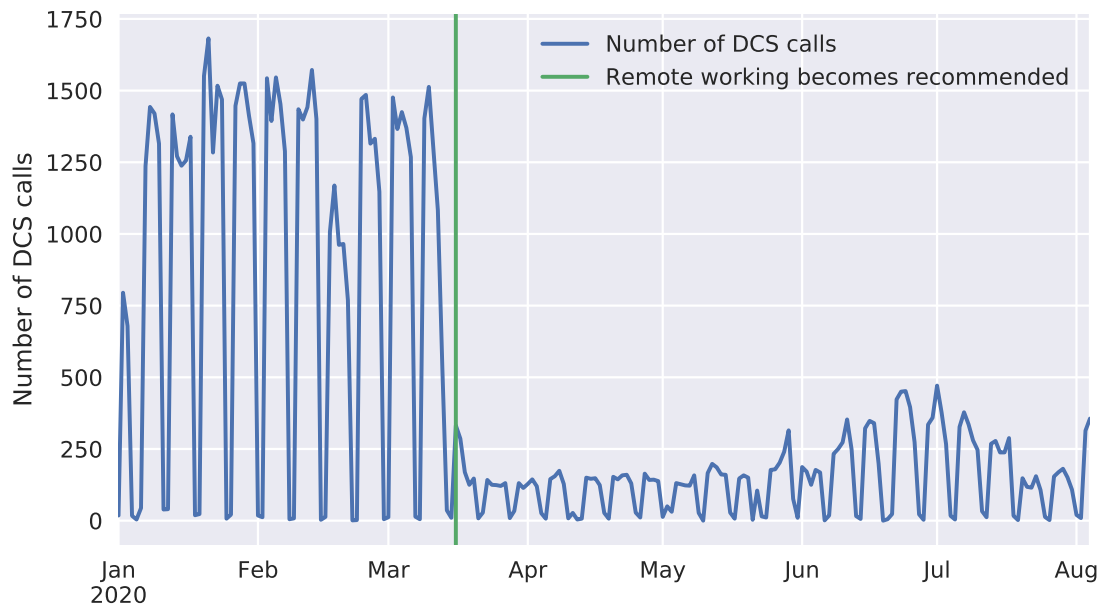


Figure 17: Anomaly in people flow caused by the COVID-19 pandemic.

open source R package called *AnomalyDetection* [48] uses Seasonal Hybrid ESD (S-H-ESD) algorithm, which is based on the generalized Extreme Studentized Deviate (ESD) test. ESD test is a statistical method which finds up to r outlier observations one at a time from a dataset. The upper bound r can be chosen arbitrarily [30]. *tsoutliers* R package [49] follows the methods developed in [50]. It detects outliers by checking the residuals of an ARIMA model. After finding all possible outliers, a new ARIMA model is fitted using a regression equation that accounts for the possible outliers. Possible outliers that did not significantly affect the fitting of the new ARIMA model are removed from the list of potential outliers. This process continues by repeating the same steps until all remaining outliers are significant or the maximum number of iterations is reached [51].

3 Materials and methods

Elevator-specific terms and statistics used in this thesis are defined in Section 3.1. Datasets used for the time series modeling are presented and explored in Section 3.2. Section 3.3 outlines implications for modeling based on the data exploration. More thorough descriptions of the modeling and experimentation methods are in Sections 3.4-3.7.

3.1 Elevator-specific terms and statistics

Different elevator-specific terms and statistics are used for evaluating elevator traffic performance. The terms and metrics used in this thesis are described in the following list:

Landing call is an elevator call made with conventional up and down buttons in an elevator lobby [52].

DCS call is an elevator call made with destination control system (DCS). With a DCS system, the user selects the destination floor using a destination operating panel (DOP) when they make the call in the elevator lobby [52].

Call time (CT) indicates the time from a landing call to the arrival of an elevator to the elevator lobby where the call was made from. The elevator is considered to arrive when it starts to decelerate before stopping.

Waiting time (WT) indicates the time from when a passenger either registers a DCS call, or joins a queue, until the responding elevator begins to open its doors at the boarding floor [53].

Time to destination (TTD) indicates the time from when a passenger either registers a DCS call, or joins a queue, until the responding elevator begins to open its doors at the destination floor [53].

3.2 Data

The main dataset for the experiments was collected from a Finnish office building. The elevator system in the building is a so-called hybrid system where the entrance floors use a destination control system (DCS) and the upper floors have conventional up and down landing call buttons.

The data was acquired from an elevator monitoring system database which collects data from the elevator group. The database contains dozens of different metrics, but the following were chosen as variables of interest for the time series modeling:

- Number of landing calls per day
- Daily average landing call time
- Daily maximum 15-minute average landing call time

- Number of DCS calls per day
- Daily average DCS waiting time
- Daily maximum 15-minute average DCS waiting time
- Daily average DCS time to destination
- Daily maximum 15-minute average DCS time to destination

Landing calls and DCS calls are not directly comparable, thus they are handled separately. The data was aggregated to daily totals, averages and maximums because daily resolution is sufficient for long-term modeling. Landing call data is shown in Figure 18 and DCS call data in Figure 19.

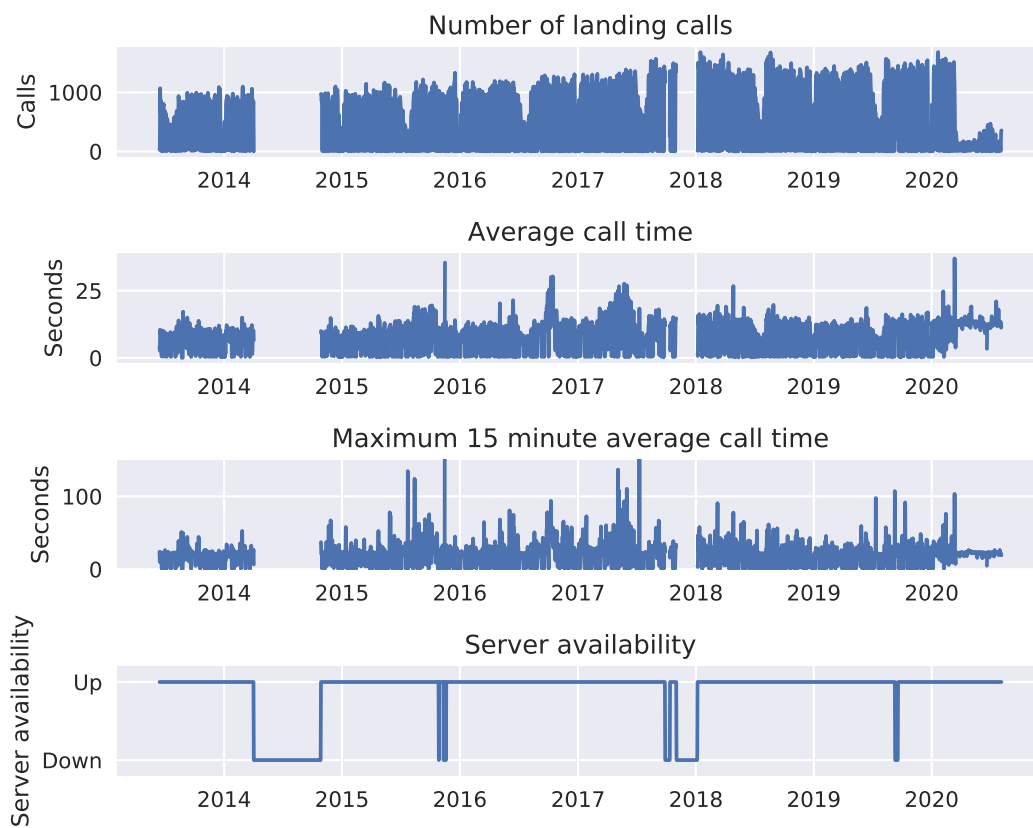


Figure 18: Landing call data

Average waiting times are used in the design and dimensioning of elevator systems so its effect on user experience is well understood. Maximum 15-minute average reflects elevator traffic during peak traffic like lunch time. Number of calls correlates with the total population of the building and helps explain the trends in waiting times and times to destination.

The database was configured to store data for the last 500 days. All 500 days were fetched, and new data was appended to it as it became available. There is also

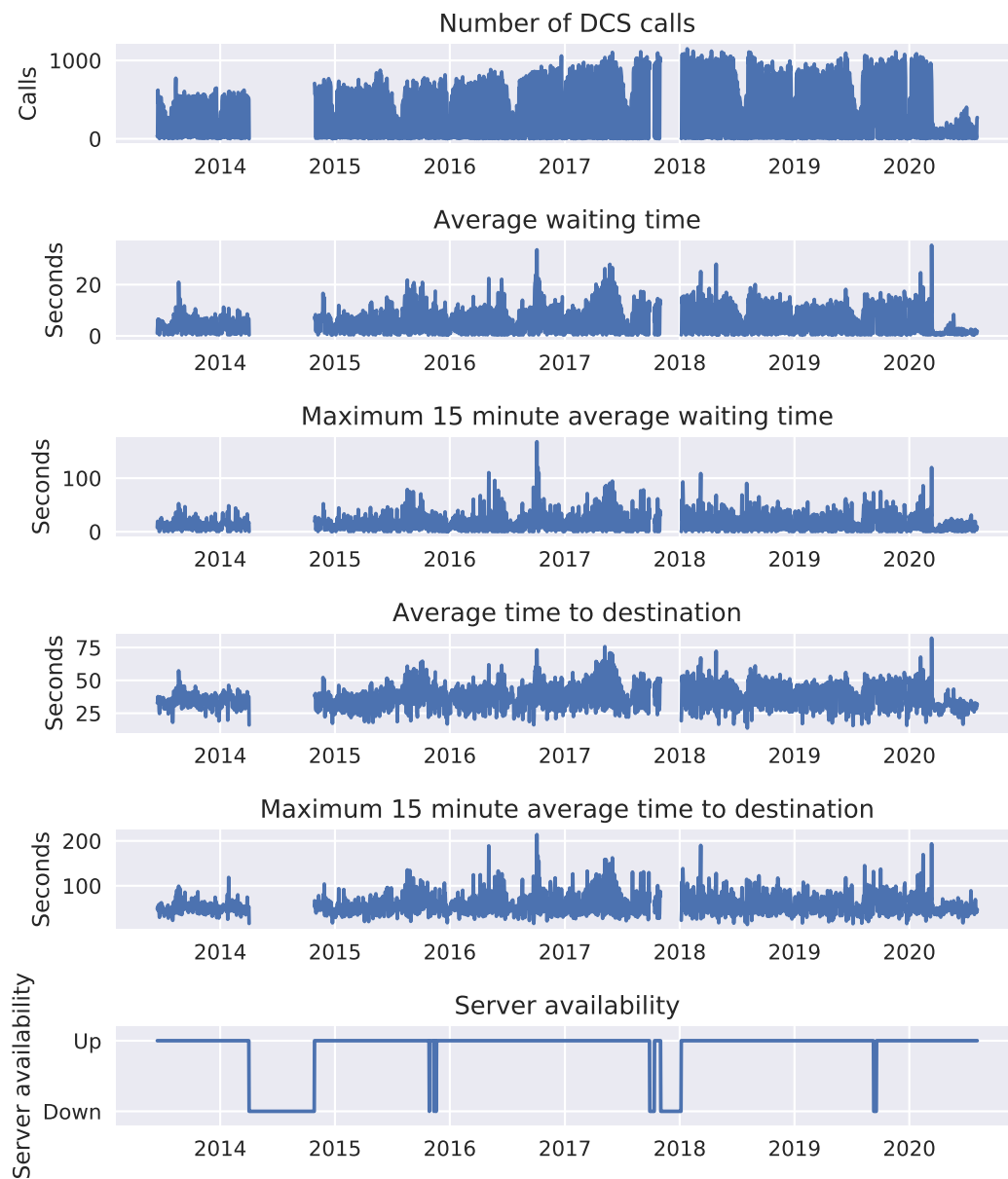


Figure 19: DCS call data

an archive of old data that starts from mid-June 2013. This archived data was also added to the current dataset. New data is manually added to the archive which has led to some missing periods of data. Data availability is depicted in the server availability plots in Figures 18 and 19.

The longest missing period is from April to end of October in 2014. For models that cannot handle missing data properly, all the data before November 2014 should be left out. Another large missing period in the data is from November 2017 to the start of January 2018. In addition, there are some smaller, 2-7-day periods of missing data which can be reasonably imputed by methods described in Section 3.5.

The dataset spans more than 6 years, although some longer periods of data are

missing. Three full years have little missing data which should be sufficient for most time series modeling methods. At least two years are needed as training data to distinguish the yearly season from noise [54]. This still leaves a full year of data to measure forecast accuracy with out-of-sample data. Some neural network-based models require very large datasets to work optimally, which might limit the choice of modeling methods.

As can be expected, the data shows a strong weekly seasonality. During weekends there is little to no traffic. Fridays are slightly quieter than other working days and their variance is significantly higher. Weekly seasonality is visualized in Figure 20.

The data also shows a clear yearly seasonality which is shown in Figure 21. Summer and Christmas holidays show a significant decrease in traffic. Traffic starts to decrease week by week during mid-June and hits a minimum in the end of July. Number of calls rises back to a normal level during the first weeks of August when most people come back from their holidays. Christmas holidays show a sudden drop of traffic during the public holidays and gradual increase during the turn of the year. Other parts of year show quite steady traffic with a slight increasing trend throughout the years.

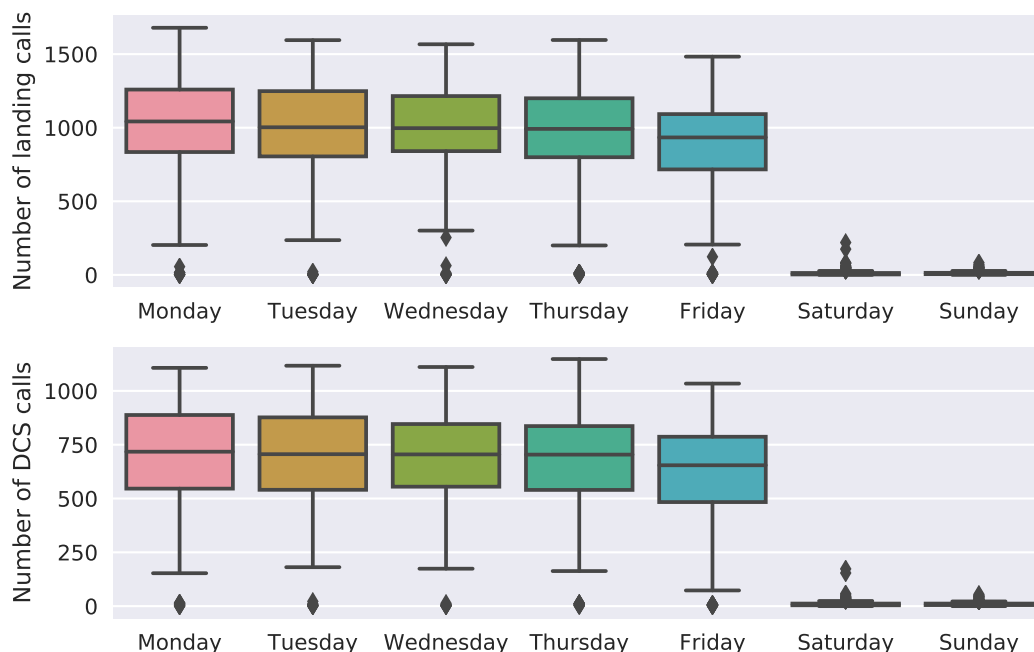


Figure 20: Number of landing and DCS calls per day of week. The box represents upper and lower quartiles and the horizontal line inside the box is the median. The whiskers show highest and lowest values excluding outliers.

Especially, the maximum landing call times show some exceptionally high peaks. On these days the maximum landing call time was more than 100 seconds, which is more than twice as much as normal days. Also, the DCS waiting times and times to destination show some outliers. Looking at the database revealed no clear reason

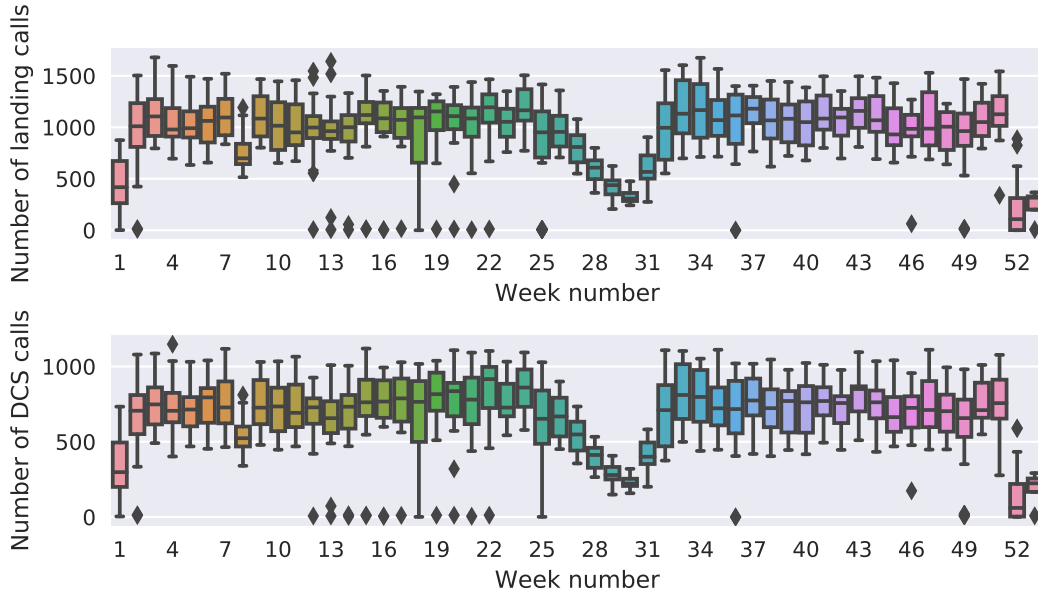


Figure 21: Daily number of landing and DCS calls grouped by week number.

for these exceptional days. However, it showed that the large maximum 15-minute average waiting times were often caused by a few extremely long waiting times. These could be caused by servicing or cleaning of the elevators, and can be probably quite safely ignored. To prevent biasing of the time series model, the outliers can be limited to a maximum value or replaced with imputed values [9].

3.3 Implications for modeling

As discussed in the previous section, the datasets used for the modeling have complex seasonalities. This is something that must be considered with all the modeling methods. One way of deseasonalizing time series is differencing. Many of the methods used in this thesis utilize the observations from the previous year to model the complex seasonality.

An imputation method presented in Section 3.5 was developed in order to fill the missing values in the data. It accounts for the complex seasonality by utilizing the weekly and yearly seasonality present in the data. It also handles holidays separately to make the imputed data as realistic as possible. Imputation is important as many of the forecasting and anomaly detection methods assume complete data to work correctly.

Different anomaly detection methods were experimented with, in order to find days with exceptional elevator traffic. Detecting persisting structural changes in the time series was also tried, but it is not the main focus of the thesis. The detected anomalies were evaluated by comparing them to known events that happened during the recording of time series.

The next sections describe how the forecasting and anomaly detection methods used in this thesis were implemented. For some methods, existing implementations

found in different software packages are used. Other methods were developed in this thesis and they are described in more thorough detail.

3.4 Smart seasonal differencing and smart lag

This section defines a so-called smart lag operator $S(y_t)$ which is utilized in the forecasting and anomaly detection methods described in the following sections. The datasets used in this thesis consists of daily observations that have strong yearly and weekly seasonality. To make the time series stationary, the seasonalities need to be removed by decomposition or differencing. Simple seasonal differencing works with time series that have only one seasonality of a constant period [9]. To handle both the weekly and yearly seasonalities as well as the effect of holidays, smart seasonal differencing based on the so-called smart lag operator was developed in this thesis.

The smart lag operator $S(y_t)$ gives the value y_{t-k} from the previous year that corresponds to time t . It accounts for weekly seasonality by selecting an observation from the same day of week. Yearly seasonality is considered by using the observation from the same week of year. If the observation from the same week of year is missing or it is a holiday, the observation is chosen from the previous or following weeks of year. Observations are searched up to 3 weeks preceding and following the week that corresponds to the current observation. First the previous week is checked, then the following week and so on. If a suitable observation is not found within ± 3 weeks, smart lag returns a missing value. If y_t is an observation from a holiday, $S(y_t)$ is chosen by selecting the same holiday's observation from the previous year using a calendar. If y_t is not a holiday, it is also ensured using the ± 3 week tolerance that $S(y_t)$ is not an observation from a holiday. Using these rules to select k also handles leap years without problems. In order to mitigate outliers, the value of the smart lag is clipped to 90 % quantile of the whole time series. This is done to ensure that the smart lag returned is a sensible presentation of the whole time series.

Smart seasonal differenced time series is defined as $y'_t = y_t - S(y_t)$. Smart seasonal differencing is inverted by adding the value from the previous year based on the same rules, i.e., $y_t = y'_t + S(y_t)$. However, the differencing loses the first year of data as the values from the year before it are not available. This means the inverse operation needs to be given the first year of the undifferenced data which is added to the first year of differenced data to get back the original values. If a forecast is made using smart seasonally differenced data, it can be inverse differenced using the same method.

3.5 Imputing missing values in data

The datasets in this thesis contain some periods of missing data as was discussed in Section 3.2. While this situation is not ideal, it is quite typical. To use methods that require complete data, the missing values need to be imputed based on the available data. However, it is important to keep in mind that imputing does not add any new information to the dataset.

Distinguishing between days whose data is missing and days that had no traffic is required so valid days are not filled with imputed data. Using the database, days with partially or completely missing data were marked as invalid. This ensures that small gaps in the data during the day will not skew the daily totals, averages and maximums.

Missing values can be imputed in several ways. The data has high yearly and weekly seasonality as shown in Section 3.2. It means that the simplest methods like carrying over the last known value or filling missing values with the mean of the data are not suitable. However, similar approaches can be adapted to account for the seasonality to make them work. Three such methods are compared in this thesis. One of them is *na_seasplit()* function from *imputeTS* R package [24]. It has some shortcomings with default settings which are corrected in the second method which is based on *na_seasplit()*. The third method is developed in this thesis with the observations made from the first two methods and it provided the basis for the smart lag operator defined earlier.

na_seasplit() splits the data to year-long periods. It uses linear interpolation by default to impute values. The imputed value is the average of the same day from the preceding and following year. If the value for previous or next year is not available, last observation carried forward (LOCF) or next observation carried backward (NOCB) is used instead. If values from both the previous and following year are unavailable, the missing value is not imputed. *na_seasplit()* matches the days from different years based on the number of days from the beginning of the year. This causes problems as the average is calculated from different days of week. If LOCF is used instead of interpolation, the imputed values are still from a wrong day of week. This issue is caused by the length of the weekly seasonality not being a multiple of the yearly season. Handling complex seasonality is widely covered topic in literature, e.g. De Livera et al. [33].

There are 52 weeks in a year and 7 days in a week. However, $7 \times 52 = 364$ does not add up to 365 days in a year. This means that the weekly seasonality shifts by one day in a regular year and two days in a leap year. For example the 100th day in 2012 was a Monday. If its value was to be filled with the value of the 100th day of 2011, it would be severely underestimated as that day was a Sunday. This can be corrected by imputing the value with the 101st day of 2011 but then holidays would be misaligned by one day. Leap days complicate the issue even more as the day offset increases by one after the leap day. Public holidays with different dates year-to-year like Easter make the seasonality adjustment even more difficult.

To correct the above issue, LOCF was chosen as the algorithm for *na_seasplit()* in the second method. Imputed values were shifted backwards by one day to make the weekly seasonality correct. Holidays were handled with a country-specific calendar in the *fbprophet* library [55]. Because the amount of traffic on a holiday does not presumably depend on the week of day, the one-day shift was not used for them and unshifted values were used instead. The value for each day preceding a holiday is the same as the value of the holiday because of the one-day shift. To correct this, days preceding holidays were filled with the average of the same day of week from the preceding and following weeks. This way the holidays were put back to correct

dates and the days preceding holidays were imputed as regular days. This works well for holidays that happen on the same date every year but holidays like Easter are still problematic.

The third method, called seasonal method, tries to combine the best aspects of the first two methods and overcome their problems. Linear interpolation is better than LOCF, as it can account for a trend in the data. Handling holidays with a calendar is a good idea but it should also work with Easter. This is achieved by utilizing the smart lag operator $S(y_t)$ defined in Section 3.4.

In order to use linear interpolation to impute missing values, in addition to the value from the previous year, also the value from the following year is needed. Following year's value is acquired with a smart lead operator $L(y_t) = y_{t+k}$, which follows a similar logic as the smart lag operator $S(y_t)$. If the corresponding values for both the preceding and following years are available, the imputed value is calculated as their average. If only one of the values is available, the imputed value is that value, i.e., LOCF or NOCB is used. If both smart lag and smart lead are not available, the value is left temporarily unimputed. After the smart lag and lead based imputation, any missing values are imputed with a rolling mean of the same days of week to make sure there are no missing values left after the imputation.

All three methods were compared by first filling in the DCS call dataset with the seasonal method to mitigate the bias of missing data. Then three months of existing data was removed from the filled test dataset. The data was deleted from June 2017 to the end of August 2017. This period was chosen, as imputing the values for the summer holiday period tests the ability to account for yearly seasonality. Year 2017 was chosen because its data is not actually missing and there is a large increase in the number of calls from 2016 to 2017. The increase in traffic tests the ability to account for a trend in the data. DCS call dataset was chosen as it contains times to destination in addition to waiting times and number of calls. Times to destination differ from the other data columns as they can never be zero.

All three methods were used to fill the deleted period of data. Imputation error was calculated for all three methods by comparing the imputed values with the actual values. Root mean square (RMS) errors for all three of the methods are summarized in Table 2. Comparison of the seasonal method and default `na_seasplit()` is shown in Figure 22. Seasonal method is compared against the corrected `na_seasplit()` method in Figure 23.

`na_seasplit()` with default parameters performed very poorly. Imputed values for the weekends were significantly higher than the actual values. This is caused by the linear interpolation and offset in the weekly seasonality. Because 2016 was a leap year, values for Sundays are imputed as the average of a Friday from last year and a Monday from the following year. Misaligned weekly seasonality shows as large spikes in the error plot in Figure 22.

The corrected `na_seasplit()` method performs very similarly to the seasonal method. However, it clearly predicts too low values for the number of calls as can be seen in Figure 23. This is caused by the LOCF algorithm that does not account for the trend in the data. In times to destination the corrected `na_seasplit()` performs even slightly better than the seasonal method based on the RMS error.

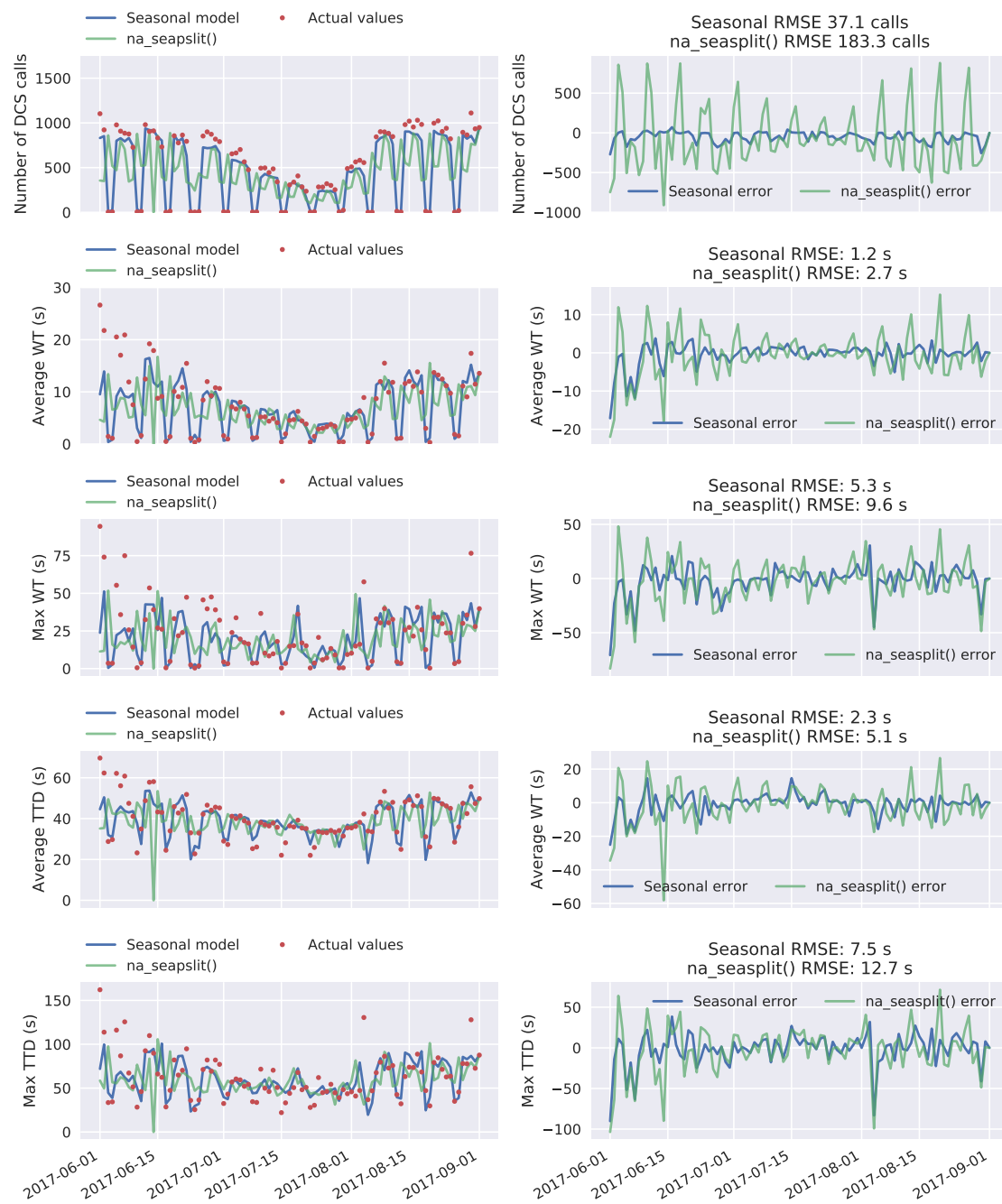


Figure 22: Imputation performance of `na_seasplit()` with default parameters compared against the seasonal method.

Overall the seasonal method is performing the best as it addresses the shortcomings of the other two methods. While its performance could probably be improved, it is adequate for imputing missing data to make more forecasting methods available. One way to improve it would be to fit a linear or polynomial trend to every available

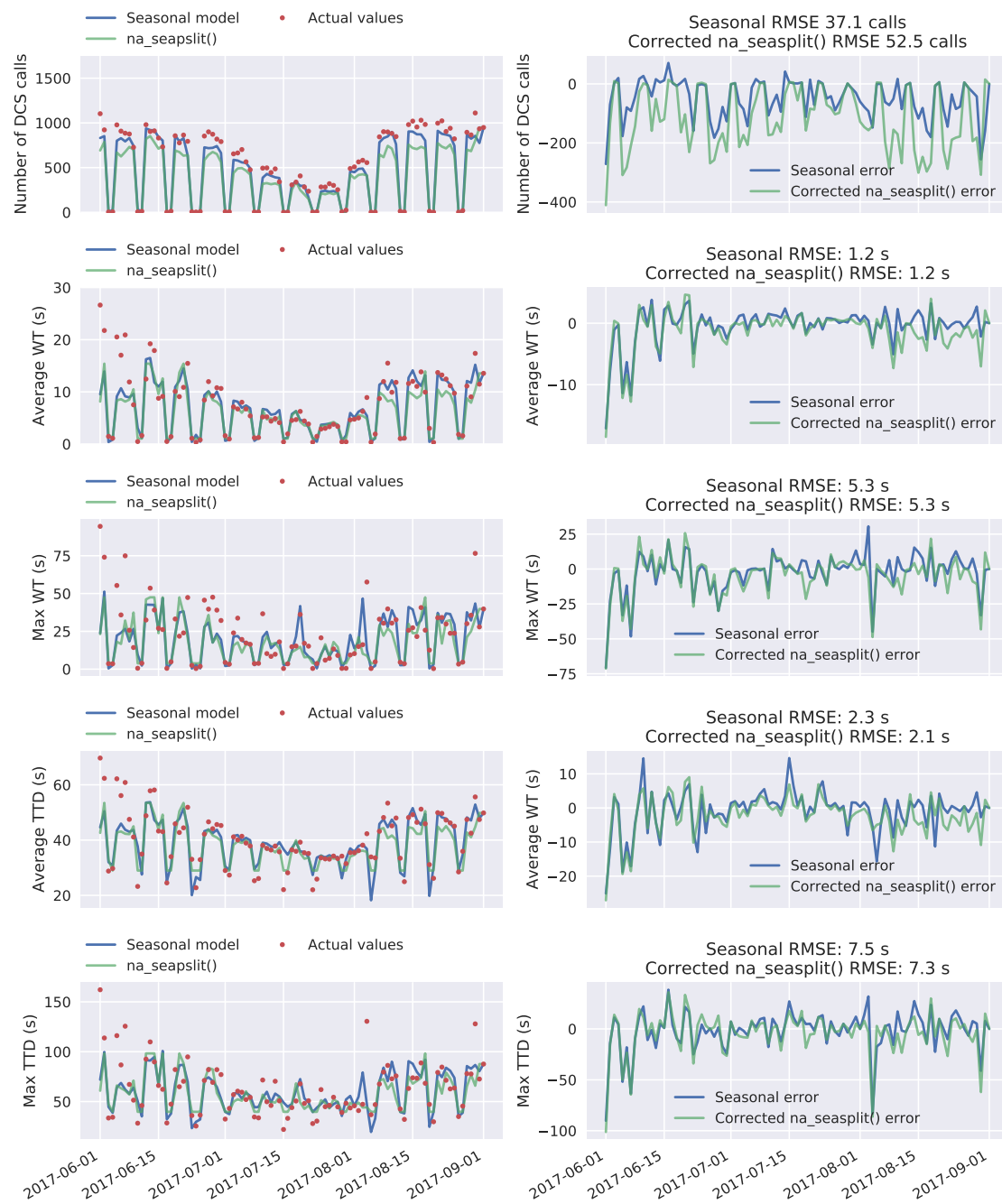


Figure 23: Imputation performance of `na_seasplit()` with corrections and optimized parameters compared against the seasonal method.

year's days that correspond to the day to be imputed. This would improve the ability to account for trends, but it would complicate the calculation a lot more. As imputation is not the focus of this thesis, this functionality was not implemented.

Table 2: RMS errors of the three different imputation methods. WT stands for waiting time and TTD for time to destination.

	na_seasplit()	Corrected na_seasplit()	Seasonal
Number of calls	183.3	52.5	37.0
Average WT (s)	2.7	1.2	1.2
Maximum WT (s)	9.6	5.3	5.3
Average TTD (s)	5.1	2.1	2.3
Maximum TTD (s)	12.8	7.3	7.5

3.6 Forecasting

Section 3.6.1 outlines the data and methods used for the forecasting experiments conducted in this thesis. Sections 3.6.2-3.6.4 present the implementation details of the forecasting methods used in the experiments.

3.6.1 Forecasting experiment

All three methods were trained using the DCS and landing call datasets from 2015 to 2018. Year 2019 was left as a test dataset which was not given to the model during the training. The models were developed and optimized by calculating different error metrics of the year-ahead forecast for the test dataset. The error metrics of all methods are compared to a seasonal naïve forecast, i.e., predicting each day to be the same as its smart lag.

To validate the developed models, the data from the beginning of 2020 was left as a validation set. The forecasting performance of the models was validated by training them with the data from 2015-2019 and calculating the error metrics for the forecasts of the beginning of 2020. 2020 is not an ideal validation set as the elevator traffic changed overnight mid-March. Most employees in the office started to work remotely due to the global COVID-19 pandemic which caused the traffic to drop to significantly. As the forecasts produced in this thesis are year-ahead, they cannot react to the sudden change. However, the first two and a half months of 2020 should be comparable to the training data and as such they can validate at least short-term performance. While the forecasts were made for the whole year 2020, the error metrics for the validation are calculated using the first 72 days. Using the days after the remote working started would not benefit the validation process, as none of the methods have a way to adapt to the situation.

Some of the methods require complete data to function properly. Training data whose missing values were imputed with the seasonal method developed in this thesis was used for them. The error metrics were calculated with unimputed data discarding the few missing values. Models that could handle missing values were trained on the original data with the missing datapoints removed.

ARIMA with smart seasonal differencing presented in Section 3.6.3 produced 80 % prediction intervals which are shown in the plots of the results. The other

two methods do not support prediction intervals natively, which is why they are not presented. Prediction intervals could be produced for all the methods with for example bagging [9].

Error metrics chosen to measure forecasting performance are RMS error, bias (MBE) and normalized RMS error (NRMSE). Normalized RMS error was calculated by dividing the RMS error with the mean of the test dataset. It relates the magnitude of the errors to the magnitude of the time series. It makes it easier to compare the performance of the forecasts of different time series with different scales.

3.6.2 LSTM

keras [38] framework with *Tensorflow* [56] backend was used to implement the LSTM neural network model. The number of time steps used as input to the model and the number of time steps forecasted at a time were experimented with during the development of the model. In the end, 120 time steps was found to be an appropriate input. The model performed the best when it predicted one time step at a time, so one output was chosen. As neural networks can find complex dependencies among multiple variables, date features were added as input variables in addition to past values. Date features consisted of the following:

- Boolean indicator is today a holiday
- Boolean indicator is tomorrow a holiday
- Day of week
- Day of month
- Day of year
- Week of year
- Year

LSTM neural networks remember the most prominent features of past time steps, but the depth of the memory is limited. In order to ensure the model knows what happened one year ago, smart lag values of the input time steps were added as additional variables. The date features and smart lag of the day to be forecasted are also known before-hand, so they were added as additional columns to the input. All the input variables were scaled to a range between 0-1 to ensure they work correctly with the activation functions used in the LSTM layers. The scaling was done using *RobustScaler* from the *scikit-learn* [40] Python package to mitigate the effect of outlier values. Altogether the input data consisted of 120×17 values, where $S(y_t)$ is the smart lag and $D(y_t)$ are the 7 date features for y_t :

120 time-steps	{	y_{t-120}	$D(y_{t-120})$	$S(y_{t-120})$	$D(y_t)$	$S(y_t)$
		y_{t-119}	$D(y_{t-119})$	$S(y_{t-119})$	0	0
		y_{t-118}	$D(y_{t-118})$	$S(y_{t-118})$	0	0
	
		y_{t-1}	$D(y_{t-1})$	$S(y_{t-1})$	0	0
		17 features				

The output of the model is y_t , which was added as an input to the prediction of the next time step during the forecasting of the year-long test data. LSTM neural network could learn to handle missing values but to simplify the training process, imputed values were used.

Model chosen for the forecasting is shown in Figure 24. It was designed to follow a simple encoder-decoder structure, implemented with a bottleneck. The idea behind encoder-decoder structure is to use the first part of the neural network find useful input features from a large amount of input data, which act as inputs for the model doing the prediction [57]. Adding a bottleneck in the middle of the model, forces the model to find optimal features before the bottleneck which are fed into the layers after the bottleneck [58].

The first LSTM layer has 256 neurons and it outputs 256 outputs for every 120 input time steps. Default *tanh* activation function was used for every LSTM layer in the model. The second LSTM layer has 150 neurons and it returns only 150 outputs causing a bottleneck to the model. RepeatVector layer is used to adapt the output shape of the second LSTM layer to the input shape of the third LSTM layer. If the number of outputs is n , RepeatVector copies the 150 input values n times to produce an output of size $(n, 150)$. In this model, the number of outputs of the whole model is one, which means that RepeatVector does nothing in this case. The third LSTM layer has input shape of $(n, 150)$, but as $n = 1$, it has effectively a similar structure as the second LSTM layer. This means that the bottleneck structure was not actually utilized in the final model as the amount of inputs is not increased after the intended bottleneck. The last layer is a regular neural network layer with 150 inputs and $n = 1$ outputs. It is used only for generating the desired number of time steps of predictions from the LSTM layer before it. A linear activation is used for the last layer to ensure the predictions are not constrained to any limits. After producing the forecast, the values were scaled back to the original scale using the scaling parameters calculated from the training data.

The training of the model used a maximum of 1000 epochs with early stopping in case the modeling error stopped decreasing for 20 epochs. In practice about 300 epochs were used for training each dataset. The final model has over 700 000 trainable parameters, which made the training process extremely slow. While the computations times of the different methods were not accurately measured, training the LSTM model took several hours on a standard laptop. *Tensorflow* supports graphics processing unit (GPU) and tensor processing unit (TPU) computation, which would speed up the process significantly.

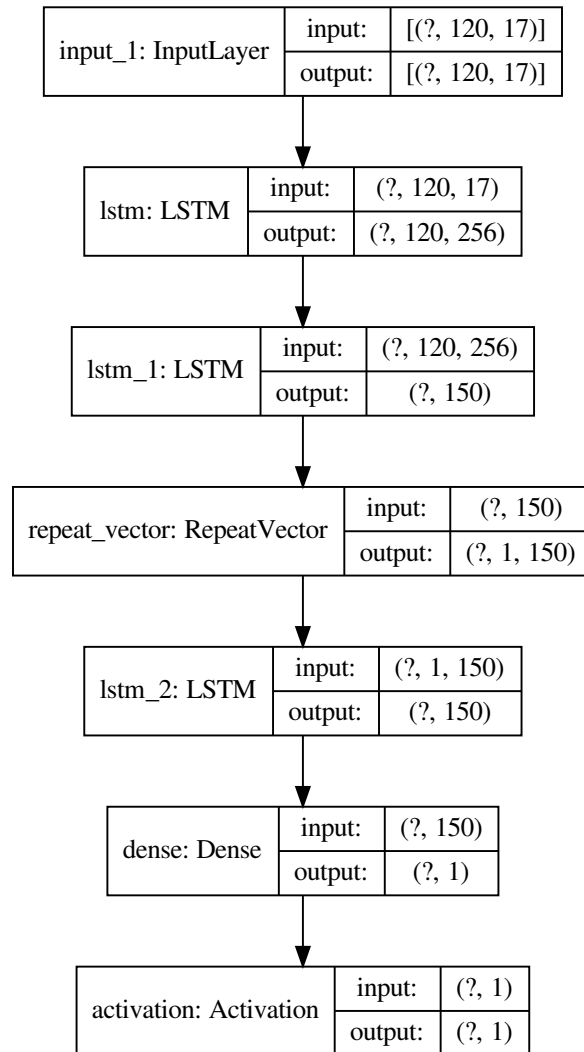


Figure 24: LSTM model used for forecasting. The length of the time series is unknown to the model when it is compiled. This is represented by the question mark in the input and output dimensions.

3.6.3 ARIMA with smart seasonal differencing (ASSD)

Smart seasonal differencing algorithm described in Section 3.4 was used for removing the complex seasonality before applying an ARIMA model. The smart seasonally differenced data had still some characteristics of non-stationarity. After one additional simple differencing operation, the data was stationary according to unit-root tests.

As the I term in ARIMA handles simple differencing, the smart seasonally differenced data was fed straight into an automated implementation of ARIMA in the *pmdarima* [59] Python library. ARIMA model implemented in *pmdarima* assumes

complete data, so imputed data was used for the training and forecasting. p , d and q parameters of the ARIMA model were automatically chosen by *pmdarima* based on AIC. These parameters were optimized for each dataset to get optimal performance.

Both ARIMA and SARIMA models are supported by *pmdarima*. Because the seasonality is already handled by the smart seasonal differencing, ARIMA model was chosen. SARIMA models were also tested and they showed lower AIC in the training phase, but they performed worse on out-of-sample test data.

After fitting the model, a year-ahead forecast with prediction intervals was produced with the training data. The forecast was inverse smart seasonal differenced using the last year of training data to transform it back to the original scale.

3.6.4 XGBoost

XGBoost [60] Python package is an open source project based on the methods introduced in [43]. It is an optimized implementation of gradient boosting introduced in Section 2.9.6.

XGBoost is not a traditional time series method and as such, it does not automatically utilize the time series characteristics of the data. To overcome this limitation, different input features were calculated from the dataset. In addition to past observations, day of week, day of month, day of year, month and holidays were used as additional inputs. To include autoregression, the observation from 7 days ago and several observations from one year ago were used. As the model is given the day of week as a separate input feature, the seasonal lagged values from one year ago were not corrected for day of week. However, maximum call time, maximum waiting time and maximum travel time datasets performed better when they were given a smart lag from the previous year in addition to the other lag features, so it was utilized. XGBoost algorithm can show the importance of each input feature after the training process. Input features used in forecasting the number of landing calls, ranked by their importance, are shown in Figure 25.

The value from the previous day was omitted from the input features as it caused problems with year-long forecasts made one day at a time. As the model is trained with step-ahead forecasts, the fit was good with the training data, where the lagged value is calculated from the training data. When year-long step-ahead forecasts were made with that model, it failed to adapt to summer holidays because the lagged feature is based on the forecasted value from the previous step. The model relied heavily on the lagged value, trusting that it was correct, which caused it to fail with out-of-sample data. During the training, the model did not have to learn the summer holidays, as it could see them with one day's delay using the observation from the day before.

Leaving out the lagged values from the past few days forced the model to focus on other features, such as the day of week and day of year. Even though the importance of the value from the previous week is the highest, the model still managed to capture the effect of the summer holidays quite well.

Although the model learned the yearly season fairly well, the largest errors happened during the summer holidays and the winter holiday week. Additional input

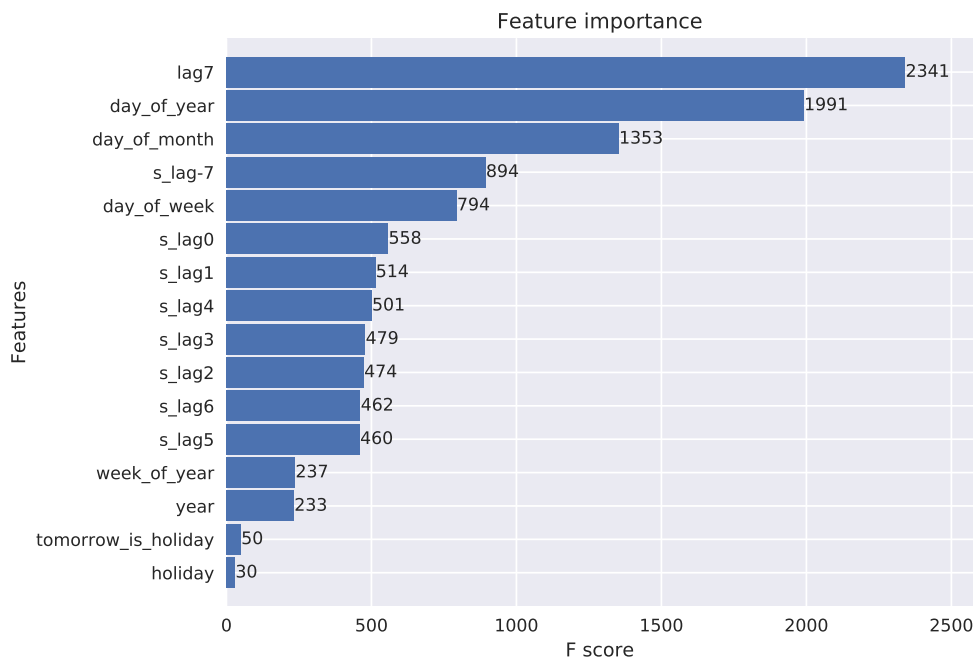


Figure 25: Feature importance for forecasting the number of landing calls using a *XGBoost* regressor. `lag7` is the observation from one week ago and `s_lag0` is the observation from $52 \times 7 = 364$ days ago.

features were tested to see if they could improve the performance during problematic periods of the year. Adding a *summer_holiday* feature which is 1 for weeks of year 27-31 and 0 otherwise, showed slight improvement during the summer period. During week 8 it is common to have winter holidays, as children have no school. During the winter holiday week there is reduced traffic which is a lot higher than during official holidays when almost no one is at work. A feature called *winter_holiday* was added to improve the modeling performance during the 8th week of the year. In the end, these two additional features were removed as they could have also been added to the other models, which makes comparing the models unfair. Adding seasonal features is easy with domain knowledge of the building and unofficial national holidays but automating the process in the future would be a lot more difficult.

XGBoost model is designed to handle missing data by deciding a default splitting direction for each node if data is unavailable. This made it possible to use unimputed datasets. When calculating the lag features, missing values were left as is. Time steps whose data was not available, were removed altogether from the training data as gaps in the data does not affect the model. Forecasts were made one day at a time. Date and holiday-related features were calculated for each day based on the dates of the forecast horizon. Lagged variables were calculated using the training data and the forecasted values from previous steps of the forecasting process. The output of the forecast was limited between zero and $1.3 \times Q_{90\%}$, where $Q_{90\%}$ is the 90 % quantile of the training data. This lets the forecast to be higher than the training data but prevents outliers.

3.7 Anomaly detection

Because finding explanations to anomalies is easier for the past few years, 2015-2016 was chosen as a training dataset and 2017 to June 2020 was used as a test dataset. Methods that do not require separate training data, were given the whole dataset from 2015 onwards, but only the anomalies during the test period are shown in the results.

Anomalies existed in all the time series of the landing and DCS call datasets, but only number of calls are presented in the results. This decision was done to ensure the results are as interpretable as possible. Time series that measured call, waiting and travel times are more susceptible to random events that cause anomalous observations which are very difficult to explain afterwards. Number of calls is a fairly stable metric and also easy to understand.

Some of the anomaly detection methods required complete data, so values imputed with the seasonal method were used with them. Anomalies detected from imputed data were removed afterwards to ensure all anomalies were based on real observed data.

As was expected, all the methods labeled holidays as anomalies. Days preceding holidays and Fridays following a holiday were also commonly labeled as anomalies. Anomalies were removed from the results if they happened during any of the following:

- Holiday based on a calendar
- Day before a holiday
- Friday after a holiday
- Weeks 1, 52 and 53 to filter out unofficial holidays around the New Year's Day
- Week 8 to filter out the winter holidays

As discussed in Section 3.6.4, some of the features used for filtering are very specific and difficult to automate. However, they are necessary to filter out false positives in automatic anomaly detection.

The results were evaluated by combining the results of the different methods and comparing the days they detected as anomalies. Explanations for the anomalies were searched by reading old news and asking employees of the building, if they had old calendar events in their calendars that might explain the anomalies.

3.7.1 *AnomalyDetection* R package

pyculiarity [61] Python package was used instead of the *AnomalyDetection* [48] R package. It is a ported version of the R package and the results should be the same as with the original R package. The library advised to impute missing values so imputed datasets were used.

3.7.2 Rolling average and standard deviation

Rolling average and standard deviation can be used for anomaly detection. If any observation differs more than $1.96 \times \sigma_t$ from the rolling average, it does not fit the model with 95 % probability, which makes it a possible anomaly. To account for the weekly seasonality, the rolling average and standard deviation are calculated using the values from the same days of week preceding and following the day to be evaluated. While this method seems simple at first, it had some problems that needed to be fixed with more complicated methods.

A centered window that averages the past and following weeks, detects single anomalous days effectively. However, it is not very effective in detecting sudden level shifts that continue in the future. A lagging window that calculates the average of past weeks is more effective for such anomalies.

To find both outliers and level shifts, a centered window with 11 weeks and a lagging window with 7 weeks were chosen. Both windows include the day to be evaluated, so trends caused by yearly seasonality do not cause false detection. Random outlier values in the data will increase the rolling standard deviation significantly which makes almost all observations fit inside the 95 % prediction intervals. To mitigate this issue, the values of the time series were limited to the 98 % quantile of the whole dataset, when the rolling average and standard deviation was calculated.

Many weekends were detected as anomalies if they had few more observations than usual. For example, 5 elevator calls on a Saturday is 67 % higher than 3 elevator calls on an average Saturday, but it is not likely to be an outlier as the absolute difference is very small. These false detections were filtered out if $abs(y_t - \bar{y}_t) / \bar{y} < 0.2$, i.e., the absolute difference between the observed value and the rolling average was less than 20 % of the mean of the whole time series.

Another problem with the anomalies detected with the lagging window was that it easily picked up days during the decreasing traffic on the start of the summer holiday season. The observed values during June are a lot smaller than the 6 weeks before it, which is detected as an anomaly. This was fixed by filtering out anomalies detected with the lagging window, if the observation differed less than 10 % from the centered rolling average. Even with this filter in place, this method detected a few days during the summer holidays as anomalously low, even though they seem normal when inspected manually. This method could probably be improved by removing the yearly seasonality with time series decomposition.

3.7.3 ARIMA with smart seasonal differencing (ASSD)

ARIMA with smart seasonal differencing was used for forecasting in Section 3.6.3. The same model can also be used for anomaly detection by comparing the observed values to the prediction intervals of step-ahead predictions. Missing values were imputed and the whole time series was smart seasonally differenced.

ARIMA model was trained with the training data using *pmdarima*. Anomalies were detected by doing step-ahead predictions and updating weights of the ARIMA model using the observed value after each prediction. If the observed value was not inside the 95 % prediction intervals of the corresponding prediction, it was marked

as an anomaly. If the observed value was higher than the average of the same days of week preceding and following it, the day was marked as higher than typical. If the observed was lower than the rolling average, it was marked lower than typical.

3.7.4 `tsoutliers`

tsoutliers [51] R package was also tested, but its results are not shown due to poor performance. The data used in this thesis did not suit the ARIMA model used in *tsoutliers*, which caused problems in fitting the model. Sometimes the automated ARIMA model could not find suitable parameters to model the non-stationary data. This probably could have been corrected by using transformations like smart seasonal differencing. However, the anomaly detection was very slow for datasets larger than a year, which made experiments difficult. Detecting anomalies one year at a time was not successful either, as the model detected only holidays and some extreme outliers in the data. Longer input periods showed some improvement but with some datasets the model did not converge after tens of minutes of calculation.

4 Results

This chapter presents the results of the experiments. An overview of the forecasting results is shown in Section 4.1. The test dataset shows year-ahead performance and the validation data proves the models work with unseen future data. The results of each method are discussed in more detail and selected plots are shown in Sections 4.1.1-4.1.3. The plots were chosen to show the strengths and weaknesses of each method. Plots of both test and validation datasets are shown. As plots of all the time series and methods could not be included due to there being too many of them, error metrics found in the tables should be used for comparing the models. Results of the different anomaly detection methods are summarized in Section 4.2.

4.1 Forecasting

Error metrics of the forecasts produced with all the methods are shown in Table 3 for the landing call data and Table 4 for the DCS call data. The results are calculated separately for the test and validation datasets. The tables include error metrics from the three forecasting methods presented in Sections 3.6.2-3.6.4 and a baseline metric called Smart lag. Smart lag is used as the implementation of the so-called seasonal naïve method discussed in Section 2.10.2.

Table 3: Test and validation error metrics of the landing call data. Error metrics of the best performing method are marked with bold for each dataset. ARIMA with smart seasonal differencing method is abbreviated to ASSD in the table.

Method	Test (2019)			Validation (1-3/2020)		
	RMSE	Bias	NRMSE	RMSE	Bias	NRMSE
AVG. CALL TIME (s)						
LSTM	2.6	0.62	0.30	4.6	-2.4	0.42
ASSD	2.4	-0.65	0.28	5.1	-3.9	0.45
XGBoost	2.4	0.94	0.28	4.8	-2.6	0.43
Smart lag	2.2	0.41	0.26	5.0	-3.0	0.44
MAX CALL TIME (s)						
LSTM	11	1.3	0.55	13	-6.3	0.49
ASSD	12	1.1	0.58	14	-7.8	0.51
XGBoost	13	4.9	0.65	13	-5.5	0.49
Smart lag	11	0.74	0.52	15	-7.9	0.54
NUMBER OF CALLS						
LSTM	164	12	0.21	147	-20	0.16
ASSD	186	11	0.24	195	-140	0.21
XGBoost	149	32	0.19	177	-88	0.19
Smart lag	175	9.2	0.22	172	-110	0.19

Table 4: Test and validation error metrics of the DCS call data. Error metrics of the best performing method are marked with bold for each dataset.

Method	Test (2019)			Validation (1-3/2020)		
	RMSE	Bias	NRMSE	RMSE	Bias	NRMSE
AVG. WAITING TIME (s)						
LSTM	2.6	-0.52	0.34	4.6	-1.8	0.47
ASSD	2.7	-0.45	0.36	4.0	-2.5	0.45
XGBoost	2.7	1.3	0.36	4.6	-1.6	0.50
Smart lag	2.3	0.46	0.31	4.2	-1.6	0.46
MAX WAITING TIME (s)						
LSTM	14	4.0	0.68	18	-7.8	0.72
ASSD	13	-0.32	0.66	15	-6.2	0.60
XGBoost	12	4.2	0.59	15	-2.6	0.61
Smart lag	11	0.71	0.54	16	-4.8	0.64
AVG. TIME TO DESTINATION (s)						
LSTM	5.8	2.4	0.15	11	-7.2	0.24
ASSD	5.5	-1.0	0.14	7.2	-4.2	0.17
XGBoost	6.7	2.9	0.17	6.7	-2.2	0.16
Smart lag	6.2	0.72	0.16	7.6	-3.1	0.18
MAX TIME TO DESTINATION (s)						
LSTM	17	-1.3	0.29	25	-7.1	0.37
ASSD	19	-1.9	0.31	21	-8.0	0.32
XGBoost	18	5.6	0.29	24	-5.8	0.35
Smart lag	17	1.1	0.28	25	-8.1	0.36
NUMBER OF CALLS						
LSTM	115	17	0.21	173	-129	0.28
ASSD	138	4.4	0.26	133	-93	0.21
XGBoost	119	61	0.22	128	-86	0.20
Smart lag	129	17	0.24	119	-77	0.19

The results achieved with all three methods are similar. To compare the different methods, they were ranked based on RMSE with each test and validation dataset to find the best method overall. Average ranks and normalized RMS errors are shown in Table 5. Of the three developed methods, LSTM was the best with the test data and XGBoost was the best with the small validation dataset.

As the test dataset is quite similar to the year before it, the seasonal naïve method outperformed the developed methods in every dataset apart from the number of calls and average time to destination. The validation dataset is only 72 days long, which makes it not suitable for evaluating the forecasting performance in terms of yearly seasonality. All the methods perform slightly worse with the validation datasets compared to the test datasets. Negative biases with the validation dataset, suggest

Table 5: Average ranks and NRMSE of the forecasting methods. The averages are calculated over all the time series for each method. The results of the best performing method are marked in bold for the test and validation datasets which are calculated separately.

Method	Test (2019)		Validation (1-3/2020)	
	Avg. rank	Avg. NRMSE	Avg. rank	Avg. NRMSE
LSTM	2.4	0.34	2.9	0.39
ASSD	3.0	0.35	2.4	0.37
XGBoost	2.6	0.34	2.1	0.37
Smart lag	1.8	0.32	2.6	0.38

that all the methods failed to account for the increasing trend. On average, XGBoost performed the best in this regard, showing bias closest to zero.

Based on the results obtained, XGBoost performed well on the test dataset and it was the best model on the validation dataset. As the difference in performance between the models is very small, it is not possible to conclude which model is certainly the best. To visualize how similar the forecasts of each method were, average time to destination for the validation dataset is shown in Figure 26 for all the methods.

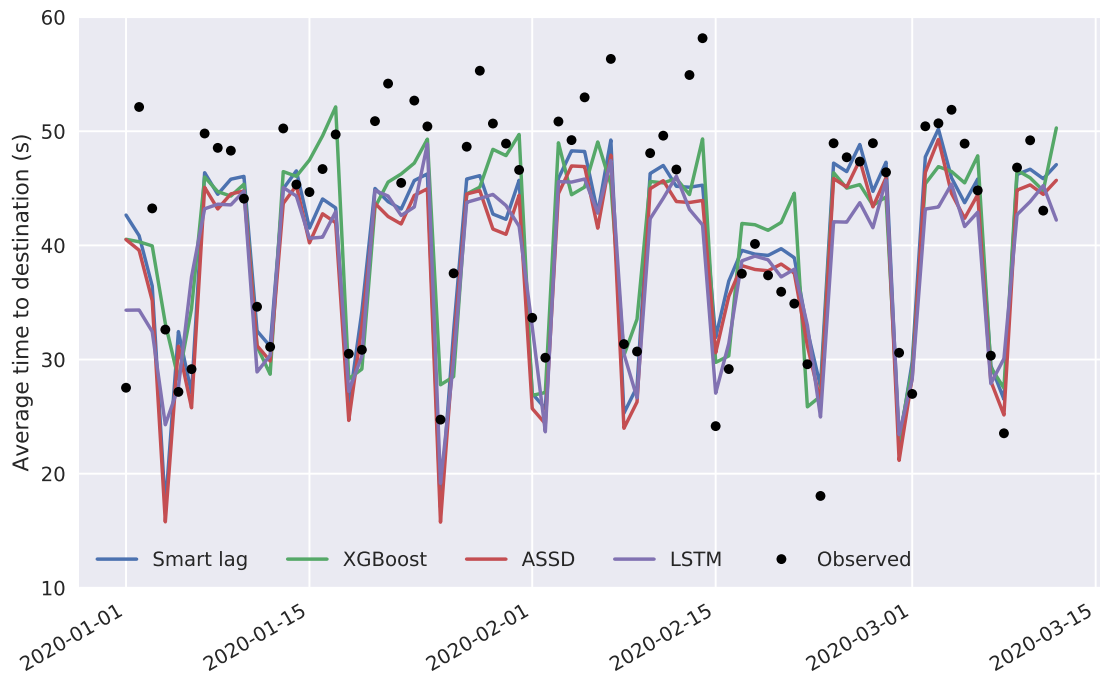


Figure 26: Average time to destination forecasted with all methods for the validation period.

Smallest normalized RMS errors were achieved with the number of landing and DCS calls and the average time to destination datasets. Their NRMSE was in the 0.15-0.25 range. As can be expected, maximum call times and maximum waiting times are the most difficult to forecast. Their NRMSE was 0.5-0.7 with most methods. Surprisingly, NRMSE of maximum time to destination was about 0.3, which is slightly better than the NRMSE of average waiting time, which intuitively should be easier to predict.

4.1.1 LSTM

Forecasted average time to destination for the year 2019 are shown in Figure 27. The residuals look normally distributed and the bias is low considering the scale of the data. There are a few sporadic erroneous predictions which show as large positive and negative peaks in the residuals. The beginning of the year is constantly predicted higher than the observed values, but the summer holiday period and the end of the year matches reality well.



Figure 27: LSTM forecast of the average time to destination for the test dataset.

The number of DCS calls for the beginning of the year 2020 is shown in Figure 28. The beginning of the year is predicted lower than the observed values but otherwise the forecast looks good. The third week of February has decreased amount of calls due to the winter holidays. LSTM failed to model this as its predictions are very similar to the surrounding weeks. This was also the case with the test dataset and the number of landing calls.

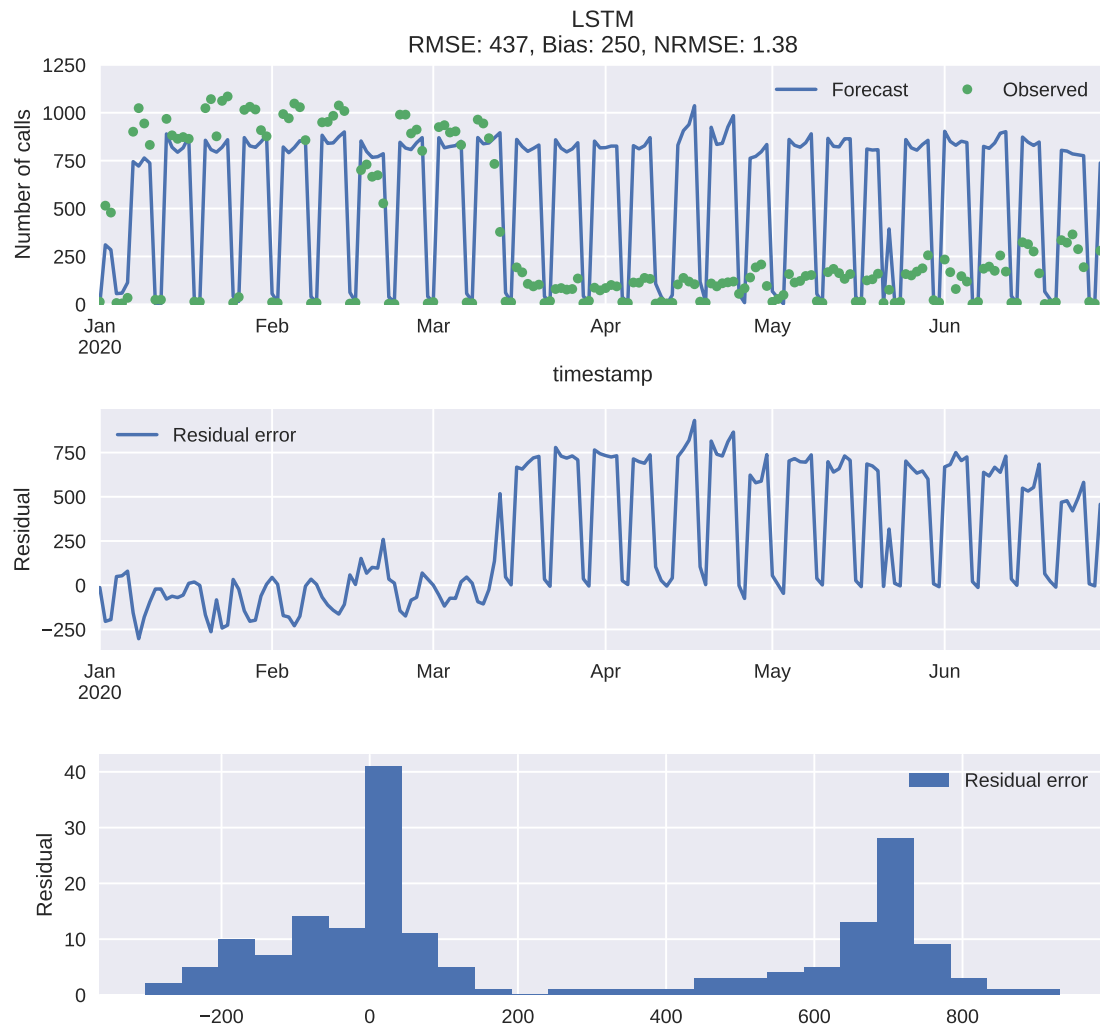


Figure 28: Number of DCS calls forecasted with LSTM for the first half of 2020. The predictions are close to the observed values until March 13th when the office moved to remote working due to the global COVID-19 pandemic.

When the office started working remotely, the number of calls plummeted. The forecasted values continue similar to the beginning of the year, as the sudden shift in level could not have been predicted based on the training data from 2015-2019.

4.1.2 ARIMA with smart seasonal differencing

ARIMA with smart seasonal differencing performed well with the average time to destination dataset. Forecast of the test dataset is shown in Figure 29. The predictions follow the observed values very well throughout the year and only 3.4 % of the observations are outside the 80 % prediction intervals. The model performs well during holidays in a calendar, e.g., May 1st. The forecast also follows the observed values well during the winter holiday week and the summer holiday period. The residuals seem to be normally distributed, and the bias is very low.

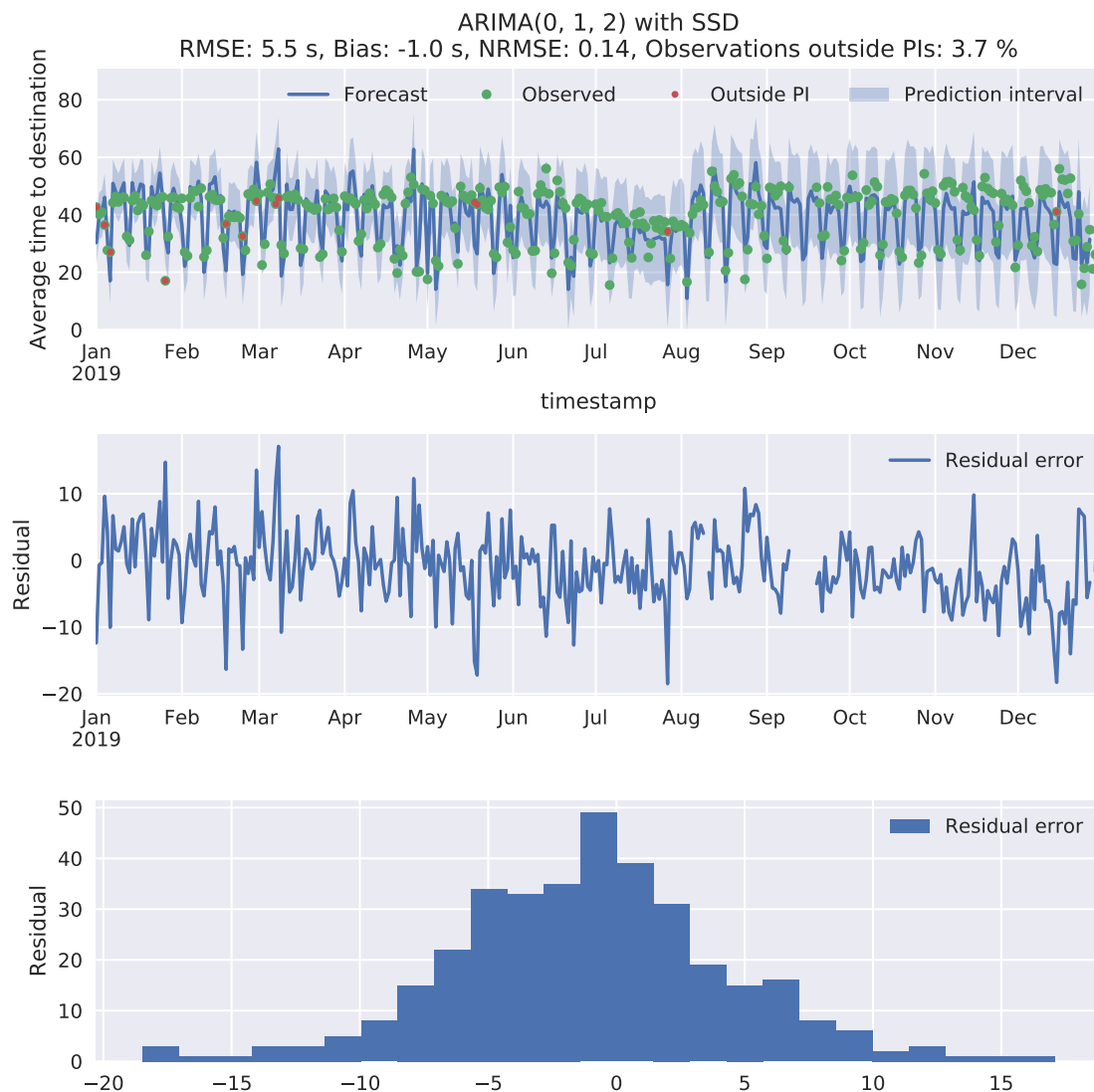


Figure 29: Year-ahead forecast of the average DCS time to destination for the test dataset.

ARIMA with smart seasonal differencing was the worst performing method for the number of landing calls dataset, both in the test and validation datasets. Forecast

for the test dataset is shown in Figure 30. Even though this was the worst method, the forecast looks good. It managed to account for the winter and summer holidays well and the level of the forecast is close to the observations during the whole year.

Even though the forecast looks good overall, there are some extremely high outliers on December 5th, 23rd, 30th and 31st. They are days preceding a holiday which has caused people to work from home. As the days are not in the holiday calendar, they were modeled as regular days, which caused the forecast to be a lot higher than the observed value. Interestingly, December 5th is an exception as it was forecasted to be quiet, but in reality it had quite normal traffic.

Number of calls for Friday February 1st was forecasted quite low even though there were no holidays in February or late January. After investigating the issue, a reasonable explanation was found. Due to the smart seasonal differencing, the prediction is largely based on Friday February 2nd 2018, which had unusually low amount of landing calls. That day, there were multiple strikes including bus drivers. It is likely that some employees had to work from home because public transportation was not available. Such issues could be solved by not using abnormal days as smart lag values. Detecting abnormal days could be achieved by utilizing the anomaly detection methods described in Section 2.11.

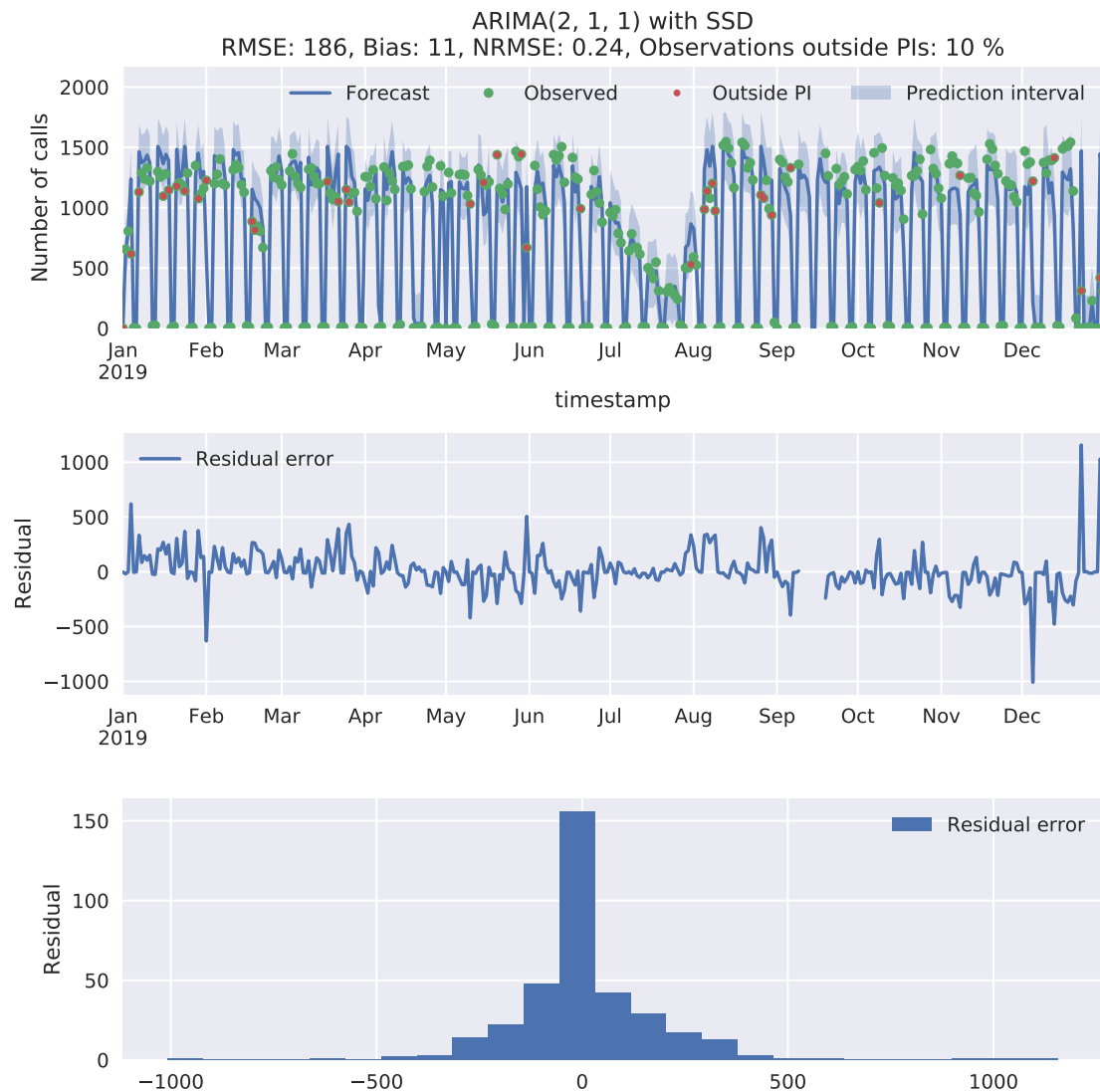


Figure 30: Year-ahead forecast of the number of landing calls for the test dataset.

4.1.3 XGBoost

XGBoost was the best-performing method with the number of landing calls test dataset. Its forecasting performance is shown in Figure 31. Most of the residuals are near zero but the distribution is quite flat and wide. Largest systematic errors happen on the winter holiday week and the last weeks of the summer holiday season. Otherwise the predictions follow the observations well. Some large errors happen occasionally which show up as large peaks in the residuals. Their cause is harder to explain than the errors in the ARIMA with smart seasonal differencing method.

Forecast of the average call time for the validation dataset is shown in Figure 32. Apart from few outlier observations, the forecast follows the observed data well. As can be seen from the bias, the forecasts are constantly slightly lower than the actual

values. The distribution of the residuals is clearly skewed towards negative values.

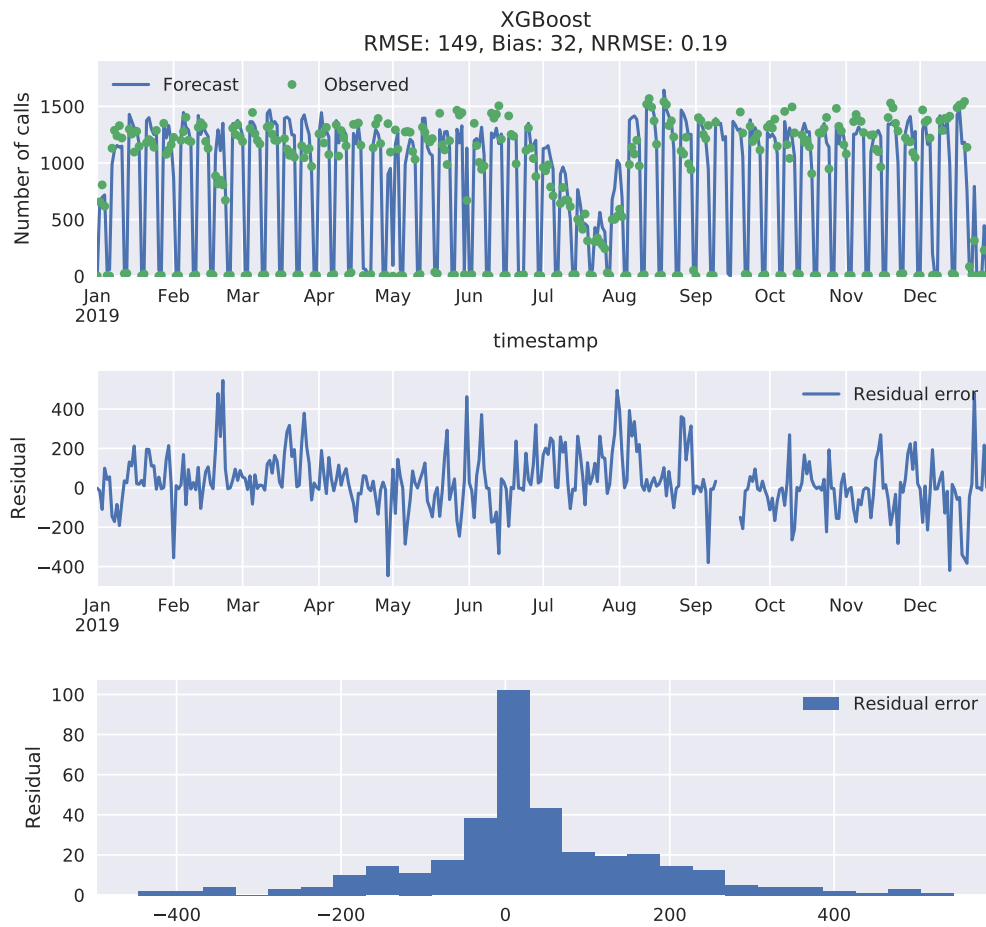


Figure 31: Year-ahead forecast of the number of landing calls for the test dataset.

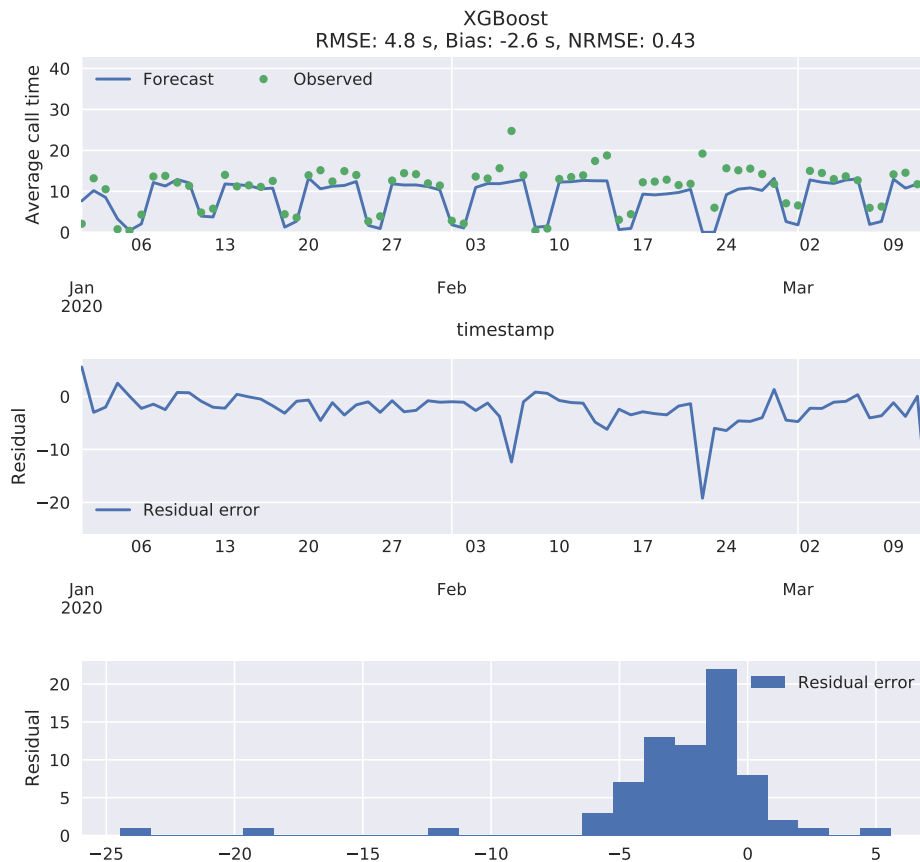


Figure 32: Forecast of the average call time for the validation dataset.

4.2 Anomaly detection

Detected anomalies for the anomaly detection test period from 2017 to the end of June 2020 are shown in Table 6. As discussed in Section 3.7, only the number of landing and DCS calls were used for anomaly detection. ARIMA with smart seasonal differencing marked a lot more days as anomalies compared to the other two methods. To make the results easier to compare and fit them into a single page, anomalies detected only by ASSD are not shown. Explanations for the anomalies are included in Table 6 for the days that could be explained.

To show a more visual presentation of the detected anomalies, the results for the year 2018 are plotted in Figure 33. It clearly shows the ASSD method is quite sensitive and it marks fairly ordinary traffic as anomalous. Further analysis of the results is included in Section 5.2.

For some of the anomalies, a plausible explanation of their cause was found by manually searching for relevant events during the days marked as anomalies. This included searching for news that happened on the dates of the anomalies as well as asking for employees in the office to check for old calendar events. The plausible explanations for the anomalies found with these methods produced a so-called ground truth.

Table 6: Abnormal days in terms of the number of landing and DCS calls detected with the different methods. H means higher than expected and L lower than expected traffic. AD stands for the *AnomalyDetection* and ASSD for the ARIMA with seasonal differencing method. Days that were detected only by the Smart seasonal ARIMA are not shown to fit the result to a single page. Summer holidays marked in the explanation column are not anomalies, but they explain why those days were caught by the different methods.

Date	DOW	Landing calls			DCS calls			Explanation
		Rolling	AD	ASSD	Rolling	AD	ASSD	
2017-01-26	Thu	-	-	-	-	H	-	A large internal event
2017-01-30	Mon	-	H	-	-	-	-	
2017-03-01	Wed	-	-	-	H	-	-	
2017-03-02	Thu	H	-	-	-	-	H	
2017-03-16	Thu	-	H	H	-	H	H	
2017-07-03	Mon	-	-	-	-	L	-	(Summer holidays)
2017-08-01	Tue	-	L	-	-	L	-	(Summer holidays)
2017-09-20	Wed	L	L	-	L	-	-	Bus strike
2017-09-21	Thu	L	-	-	L	L	-	Bus strike
2017-10-13	Fri	-	H	H	-	-	-	
2017-10-23	Mon	-	-	H	-	H	H	
2017-10-31	Tue	-	H	-	-	H	-	
2018-02-02	Fri	L	L	L	L	L	L	Bus strike
2018-04-03	Tue	-	-	-	L	-	-	
2018-04-07	Sat	H	-	-	H	-	-	
2018-04-20	Fri	L	-	-	L	-	-	
2018-05-15	Tue	-	-	-	H	-	-	
2018-05-16	Wed	L	-	L	-	-	-	
2018-06-08	Fri	-	-	-	-	L	-	
2018-06-13	Wed	-	H	-	-	-	-	
2018-07-05	Thu	L	-	-	-	-	-	(Summer holidays)
2018-07-09	Mon	L	-	-	-	-	-	(Summer holidays)
2018-08-10	Fri	-	-	-	-	H	-	
2018-09-25	Tue	-	L	-	-	L	-	
2018-10-17	Wed	-	-	-	L	-	-	An out-of-office event
2018-10-19	Fri	L	L	L	L	L	L	An out-of-office event
2019-01-18	Fri	-	-	-	-	L	-	
2019-03-05	Tue	-	H	-	-	H	-	
2019-03-29	Fri	L	-	-	L	-	-	
2019-04-25	Thu	-	-	-	H	-	-	A large internal event
2019-06-24	Mon	H	-	-	-	-	-	
2019-07-25	Thu	-	L	-	-	-	-	(Summer holidays)
2019-08-27	Tue	-	L	-	-	-	-	
2019-08-29	Thu	-	-	-	-	L	-	
2019-10-02	Wed	-	-	-	-	H	-	
2019-10-10	Thu	-	H	H	-	-	-	A large internal event
2019-11-07	Thu	-	-	-	-	H	-	
2019-11-08	Fri	-	H	-	-	-	-	
2020-01-22	Wed	-	L	-	-	-	-	
2020-01-24	Fri	-	-	-	-	H	-	
2020-03-13	Fri	L	L	L	-	L	L	Remote working (COVID-19)
2020-03-16	Mon	L	L	L	L	-	L	Remote working (COVID-19)
2020-03-17	Tue	L	-	L	L	-	L	Remote working (COVID-19)
2020-03-18	Wed	L	-	L	L	-	L	Remote working (COVID-19)
2020-05-29	Fri	-	H	-	-	-	H	

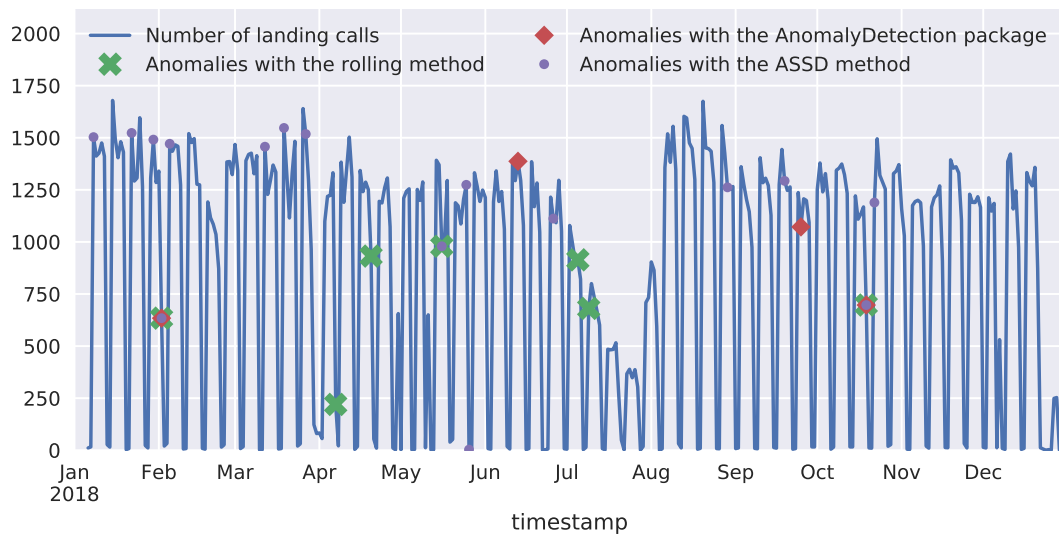


Figure 33: Anomalies in the number of landing calls during 2018 detected with the different methods.

It is important to note that finding reliable explanations for past events is quite difficult. This means some of the days for which no explanation was found probably would actually have an explanation. On the other hand, anomalies missed by all the methods were not considered at all. Validating whether all the anomalies are actually caused by the reasons mentioned in their explanations is also difficult. These considerations should be kept in mind when evaluating the results.

ARIMA with smart seasonal differencing method found a lot more anomalies than the other methods when 95 % prediction intervals were used. If wider prediction intervals were used, some of the anomalous days detected by the other methods were missed by ASSD. Thus, it seems that ARIMA with smart seasonal differencing is not suitable for anomaly detection without further development.

Both the rolling and the *AnomalyDetection* method marked a few days during the summer holidays as abnormally low traffic, even though the traffic was normal for the holiday period. Such false detections could be quite simply filtered out in a similar manner to the other holidays described in Section 3.7. However, it would require some country and site-specific knowledge of the summer holiday season. Features for holiday-based filtering could be automatically estimated using historical data, but that was left out-of-scope of this thesis.

The effect of holidays is shown in Figure 34. The holidays decrease traffic not only during the holiday, but also before and after the holiday. The day preceding a holiday, some of the employees like to leave early. If the day after the holiday is a Friday, some of the employees work remotely or take the day off. This would cause a lot of false detections, which is why the additional holiday filters were added to all the methods.

It is fairly difficult to evaluate the anomaly detection performance of the different methods in Table 6, as explanations for all the days could not be found. However, it

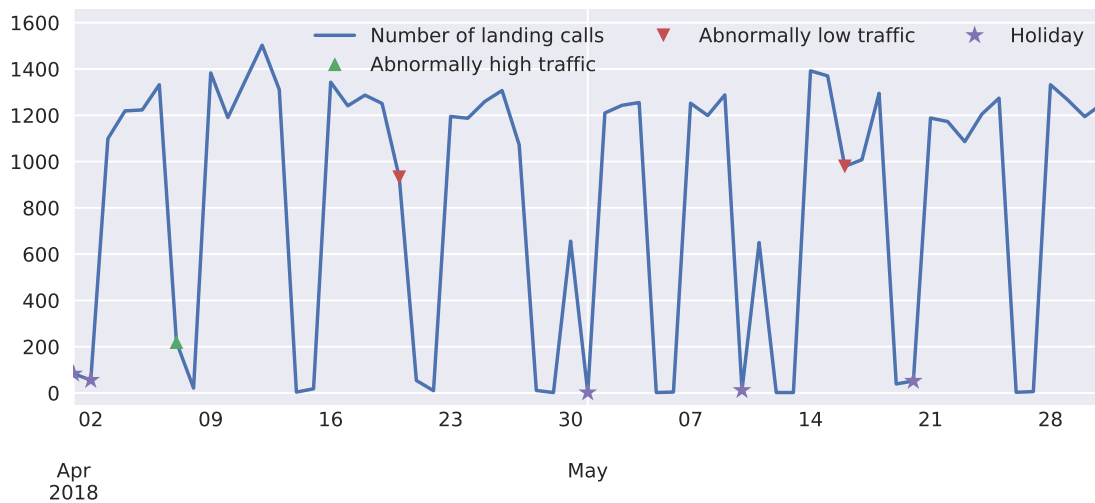


Figure 34: Anomalies in the number of landing calls during April and May 2018 detected with the rolling method. The days preceding and following holidays also show reduced traffic, but they were filtered out with a calendar.

is likely that the days detected with most of the methods were abnormal in reality. This was also verified by manually looking at the data. Even days that were detected by only one of the methods can be anomalies. For example, April 25th 2019 was the date for a large internal event. During such days, some of the employees in the office gather to the cafeteria. An increase in traffic during the hours surrounding the event was also visible in unaggregated data, which adds plausibility to the explanation.

The implementation of ASSD used in this thesis had clearly the worst performance of the three anomaly detection methods tested. Ranking the other two is a difficult task based on the results produced. A simple metric is to look at how many percent of the detections have an explanation and how many percent of the explained days were found by each method. While some of the days without an explanation probably could be explained in some way, this gives at least some idea to how the methods compare to each other.

49 % of the detections made by the rolling method have a ground truth explanation. Only 29 % of the detections made by the *AnomalyDetection* method could be explained. The rolling method found 83 % of the explained days while *AnomalyDetection* found only 67 %. The days explained as summer holidays were not counted as explainable. Each day was regarded as a detected anomaly if it was marked as an anomaly based on either the landing or DCS call data. This was done because some anomalies might affect only one of the datasets. These results indicate the rolling method performs the best of the three. It is also the simplest and easiest to interpret.

5 Discussion

This chapter includes a discussion of the results. Section 5.1 reviews which of the goals set for forecasting were met and which require more work in the future. Section 5.2 examines the validity of the anomaly detection results and outlines a few ideas for future development.

5.1 Forecasting

One of the goals for the models developed in this thesis was to model the complex seasonality present in the data. Based on the forecasts produced by all the models, this goal was met. The year-ahead predictions followed the weekly and yearly seasonalities well, and even holidays were modeled correctly in most cases.

Another goal was to produce long-term forecasts of elevator usage and people flow performance. Such forecasts would be useful in ensuring optimal people flow in buildings throughout their life cycles. Large negative biases in the forecasts of the validation set indicate none of the methods managed to predict the increasing trend. Modeling the relatively small trend component in the datasets was shadowed by the complex seasonality, which took a lot of time and effort to model successfully.

The failure to predict the trend in the chosen models was caused by multiple reasons. The trend in elevator traffic data is non-deterministic and modeling it is difficult, especially with a very limited amount of data. The ARIMA model used in the ASSD method was given smart seasonally differenced data as its input. The ARIMA model assumed its input data to be stationary or stationary after differencing, i.e., it has no trend, or the trend is constant. As the forecasts produced for the test dataset did not seem to clearly suffer from the lack of a trend component, it was not added during the development. XGBoost and LSTM models were given the year and month of the observations as input features. They could have learned the trend, but both underperformed with the validation dataset.

Based on the validation results, the trend component should have been researched more and possibly modeled separately. However, due to lack of time and data it was not done in this thesis. If the models would have been tuned to forecast the validation datasets, it would be uncertain if they would work any better with unseen future data.

Working with daily data added complexity to the models as they had to cope with multiple seasonality. Using weekly or monthly aggregated data could have simplified the modeling task and possibly enabled the models to learn the trend component better. Another option would have been to extract the trend using time series decomposition and forecast it separately. This would possibly be a quick and straight-forward way to improve the developed models and make them more usable for predicting future traffic demand. Due to time constraints, such experiments were left out of this thesis, but it would be a good starting point for future research.

While based on the results of the experiment, XGBoost was concluded as the best-performing method, the results cannot be generalized. The test and validation datasets were quite short and only from a single office building. In addition, random

variation was not formally analyzed. Based on these experiments, no method should be left out of consideration in the future.

In early stages of developing the models, a few methods not mentioned in the results were also tested. Facebook’s automated *fbprophet* [55] forecasting package was applied to some of the datasets. It performed poorly even when it was given holidays and weekends as external regressors. Applying logarithmic transformation to the time series before feeding them to *fbprophet* made the results better. In the end, *fbprophet* was left out of the chosen methods, as it is fully automated, and its performance is difficult to improve.

mstl decomposition, found in the *forecast* [17] R package, was used for decomposing the time series. *mstl* divided the time series to a trend, weekly and yearly seasonality components. Forecasts were made to deseasonalized data using ARIMA and ETS models. This method performed quite well but *mstl* failed to model the complex seasonality, especially holidays. With more development, this method could perform well. This could also be a good direction for future research.

The datasets modeled in this thesis are multivariate, but only univariate methods were used. There is clear correlation between the time series as can be seen in Figure 35. Using multivariate models that could utilize these correlations, could provide better forecasting and anomaly detection performance in future research.

5.2 Anomaly detection

Finding anomalous days within the time series is interesting, as it can give insights into how the building is used. However, most anomalies are caused by relatively short events that might be difficult to detect from daily aggregated data. Using, e.g. hourly data, would make it significantly easier to detect sudden changes in people flow and make the results more interpretable. Combining elevator availability statistics to the other elevator traffic metrics could also be beneficial. This would enable for the model to know whether the anomaly was caused by some external event or if one elevator was out-of-use due to cleaning or service for example.

While finding single outliers in the time series is interesting, finding sudden persisting changes in people flow would be more useful. Such a detector could notify e.g. if a new tenant in an office building is causing people flow performance issues or if an elevator has been left to wrong operating mode after service.

The rolling method and the *AnomalyDetection* are not suitable for detecting persisting structural changes in a time series. Such a change happened on March 13th 2020 when most of the employees in the office started working remotely due to the global COVID-19 pandemic. All methods detected anomalies when the change happened, but only ASSD continued to detect anomalies afterwards. While detecting persisting changes was not the main goal for the anomaly detection in this thesis, it would be a good feature in the future. *tsoutliers* R package includes a detector for structural changes but the method was not directly suitable for the datasets used, as discussed in Section 3.7.4. Other ways of detecting structural change anomalies in elevator traffic would be a good topic for future research.

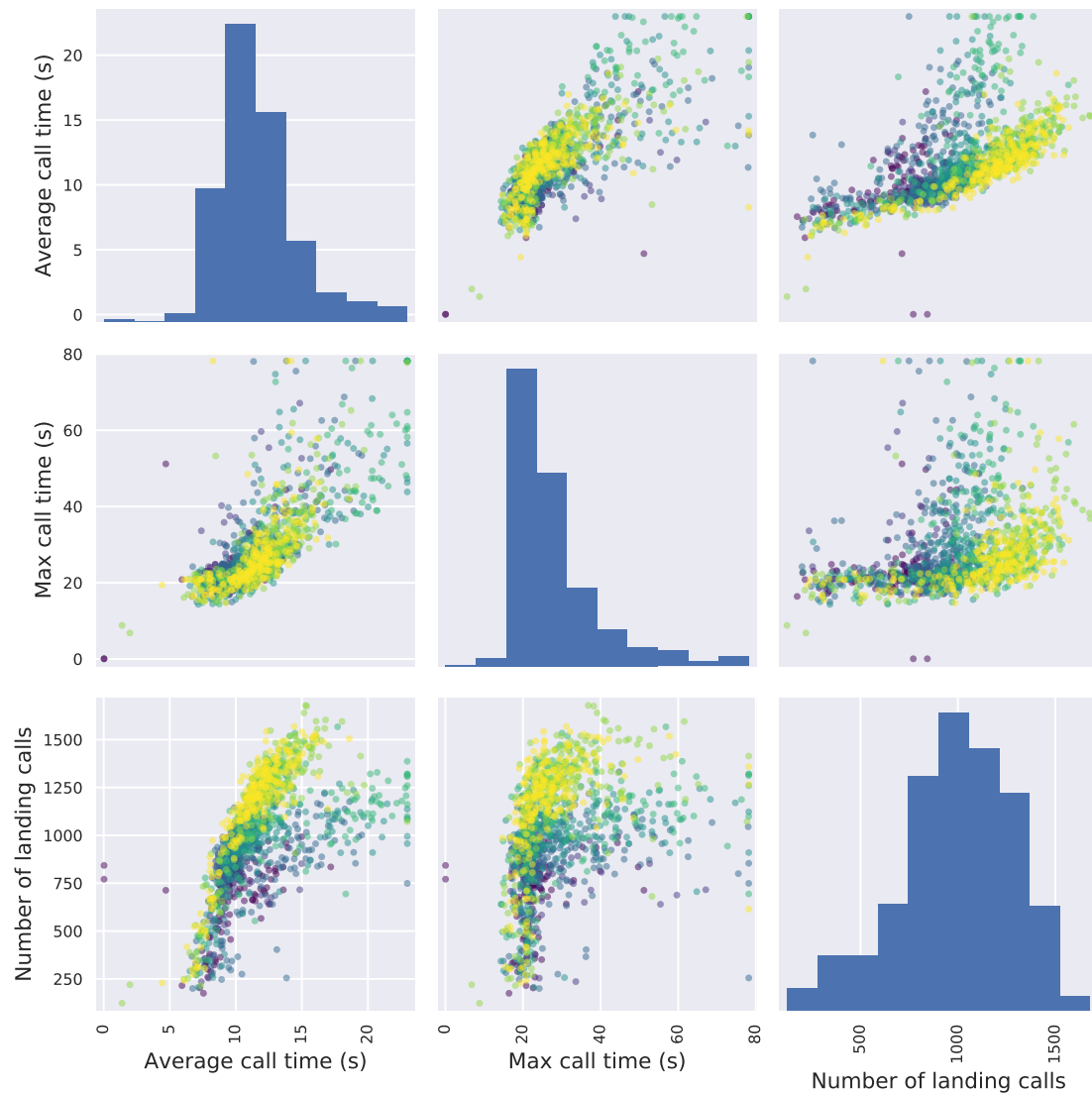


Figure 35: Correlation plot of the landing call dataset. Only days with more than 100 landing calls are shown and the observed values are limited to 99 % quantiles in order show the outliers while keeping the scale sensible. The year of each observation is represented by the color of each point. Year 2020 is not shown as it contains anomalous observations due to COVID-19.

6 Conclusions

The elevator traffic datasets used in this thesis were thoroughly explored to find out their characteristic features. They show strong weekly and yearly seasonalities, which complicate the modeling process. Due to technical and human errors, some of the data is missing. External factors affect the behavior of office employees, which show up as anomalies in people flow data. All these factors were considered when the modeling methods used were developed.

The time series models developed in this thesis were successful in capturing the complex multiple seasonality present in elevator traffic datasets. In addition, a new imputation method was developed based on the strengths and weaknesses of existing implementations. It imputes missing data considering the seasonality and effects caused by holidays. The imputed data was used for forecasting and anomaly detection models that require complete data.

The modeling methods applied to the datasets were developed based on the information gathered with a literature review on time series data and modeling. While the methods applied were mostly existing implementations found in various software packages, the input data required transformations and processing to make it work with the models. Smart lag operator and smart seasonal differencing were developed in this thesis to help the models cope with the complex seasonality.

Year-ahead forecasts produced with all the developed models follow out-of-sample observations well, proving that long-term predictions are possible. Two of the models, i.e. ASSD and XGBoost, outperformed the baseline model with validation data. The models did not completely capture trends present in the datasets. Modeling the trends separately from the seasonality would be a good direction for future research. Although the models had some weaknesses, the results show that it is possible to work with elevator traffic time series aggregated to daily level, even with long forecast horizons.

Four different anomaly detection algorithms were applied to some of the time series datasets. Two of them were successful in detecting clear anomalies in elevator traffic. For some of these anomalies, an explanation was found to help validate the results obtained. Finding short-term anomalies could possibly benefit from using hourly datasets and adding elevator availability statistics as external variables. Future research could focus on detecting persisting long-term changes in people flow data which would be helpful in detecting problems automatically.

The findings and results of this thesis are a good foundation for future developments in time series modeling of elevator traffic data. It is clear that people flow data should be recorded and archived for future use. With some more work in the future, time series analysis could be automated fleet-wide, which would yield invaluable insights into the usage of the buildings throughout their life cycles. People flow data is something few organizations can acquire, which makes it very valuable. Data gathered today will give a competitive advantage in the future, when it can be fully utilized.

References

- [1] Marja-Liisa Siikonen and Janne Sorsa. People Flow Analysis in Lift Modernization. In *8th Symposium on Lift & Escalator Technologies/CIBSE*, pages 89–98, May 2018.
- [2] Marja-Liisa Siikonen and J. Leppälä. Elevator traffic pattern recognition. In *Proceedings of IFSA '91 Brussels*, pages 195–198. The International Fuzzy Systems Association, July 1991.
- [3] Xiaoke Yan, Yu Liu, Zongyuan Mao, Zhong-Hua Li, and Hong-Zhou Tan. SVM-based Elevator Traffic Flow Prediction. In *2006 6th World Congress on Intelligent Control and Automation*, volume 2, pages 8814–8818, June 2006. doi: 10.1109/WCICA.2006.1713703.
- [4] Erkan Erdogdu. Electricity demand analysis using cointegration and ARIMA modelling: A case study of Turkey. *Energy Policy*, 35(2):1129–1146, February 2007. ISSN 0301-4215. doi: 10.1016/j.enpol.2006.02.013.
- [5] Ergun Yukseltan, Ahmet Yucekaya, and Ayse Humeyra Bilge. Forecasting electricity demand for Turkey: Modeling periodic variations and demand segregation. *Applied Energy*, 193:287–296, May 2017. ISSN 0306-2619. doi: 10.1016/j.apenergy.2017.02.054.
- [6] Raquel Prado and Mike West. *Time Series: Modeling, Computation, and Inference*. CRC Press LLC, Philadelphia, PA, United States, 2010. ISBN 978-1-4398-8275-7.
- [7] Peter J. Brockwell and Richard A. Davis. *Introduction to Time Series and Forecasting*. Springer Texts in Statistics. Springer, New York, second edition, 2002. ISBN 978-0-387-95351-9.
- [8] Peter J. Brockwell. *Time Series: Theory and Methods*. Springer Series in Statistics. Springer, second edition, 1991.
- [9] R.J. Hyndman and G. Athanasopoulos. *Forecasting: Principles and practice*, 2nd edition, OTexts: Melbourne, Australia. <https://otexts.com/fpp2/>, 2018.
- [10] George E. P. Box, Gwilym M. Jenkins, and Gregory C. Reinsel. *Time Series Analysis*. John Wiley & Sons, Ltd, fourth edition, 2008. ISBN 978-1-118-61919-3. doi: 10.1002/9781118619193.
- [11] Robert H. Shumway and David S. Stoffer. *Time Series Analysis and Its Applications: With R Examples*. Springer, April 2017. ISBN 978-3-319-52452-8.
- [12] Evžen Kočenda and Alexandr Černý. *Elements of Time Series Econometrics: An Applied Approach*. Karolinum Press, Prague, Czech Republic, second edition, 2015. ISBN 978-80-246-3198-1.

- [13] Royal Observatory of Belgium. SILSO | World Data Center for the production, preservation and dissemination of the international sunspot number. <http://sidc.oma.be/silso/>, April 2020.
- [14] David A. Dickey and Wayne A. Fuller. Distribution of the Estimators for Autoregressive Time Series With a Unit Root. *Journal of the American Statistical Association*, 74(366):427–431, 1979. ISSN 0162-1459. doi: 10.2307/2286348.
- [15] Peter C. B. Phillips and Pierre Perron. Testing for a Unit Root in Time Series Regression. *Biometrika*, 75(2):335–346, 1988. ISSN 0006-3444. doi: 10.2307/2336182.
- [16] Denis Kwiatkowski, Peter C. B. Phillips, Peter Schmidt, and Yongcheol Shin. Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root? *Journal of Econometrics*, 54(1):159–178, October 1992. ISSN 0304-4076. doi: 10.1016/0304-4076(92)90104-Y.
- [17] Rob J. Hyndman and Yeasmin Khandakar. Automatic Time Series Forecasting: The forecast Package for R. *Journal of Statistical Software*, 27(3), 2008. ISSN 1548-7660. doi: 10.18637/jss.v027.i03.
- [18] Skipper Seabold and Josef Perktold. Statsmodels: Econometric and statistical modeling with Python. In *9th Python in Science Conference*, 2010.
- [19] G. E. P. Box and D. R. Cox. An Analysis of Transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, 26(2):211–252, 1964. ISSN 0035-9246.
- [20] R. B. Cleveland, William S. Cleveland, Jean E. McRae, and Irma J. Terpenning. STL: A seasonal-trend decomposition procedure based on loess. *Journal of Official Statistics*, 6(1):3–33, 1990.
- [21] Steffen Moritz, Alexis Sardá, Thomas Bartz-Beielstein, Martin Zaefferer, and Jörg Stork. Comparison of different Methods for Univariate Time Series Imputation in R. *arXiv:1510.03924 [cs, stat]*, October 2015.
- [22] Neeraj Bokde, Marcus W. Beck, Francisco Martínez Álvarez, and Kishore Kulat. A novel imputation methodology for time series based on pattern sequence forecasting. *Pattern Recognition Letters*, 116:88–96, December 2018. ISSN 01678655. doi: 10.1016/j.patrec.2018.09.020.
- [23] F. Martínez-Álvarez, A. Troncoso, J. C. Riquelme, and J. S. Aguilar-Ruiz. LBF: A labeled-based forecasting algorithm and its application to electricity price time series. In *2008 Eighth IEEE International Conference on Data Mining*, pages 453–461, 2008.

- [24] Steffen Moritz and Thomas Bartz-Beielstein. imputeTS: Time Series Missing Value Imputation in R. *The R Journal*, 9(1):207, 2017. ISSN 2073-4859. doi: 10.32614/RJ-2017-009.
- [25] Ratnadip Adhikari and R. K. Agrawal. An Introductory Study on Time Series Modeling and Forecasting. *arXiv:1302.6613 [cs, stat]*, February 2013.
- [26] Avishek Pal and PKS Prakash. *Practical Time Series Analysis*. Packt Publishing, 1st edition, 2017. ISBN 978-1-78829-022-7.
- [27] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, Cambridge, 2019. ISBN 978-1-108-38069-0. doi: 10.1017/9781108380690.
- [28] Robert Goodell Brown. *Statistical Forecasting for Inventory Control*. McGraw-Hill, New York, 1959.
- [29] Charles C. Holt. Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting*, 20(1):5–10, January 2004. ISSN 0169-2070. doi: 10.1016/j.ijforecast.2003.09.015.
- [30] Nathanael A. Heckert, James J. Filliben, C. M. Croarkin, B. Hembree, William F. Guthrie, P. Tobias, and J. Prinz. *Handbook 151: NIST/SEMATECH e-Handbook of Statistical Methods*. National Institute of Standards and Technology, November 2002.
- [31] Arnau Alabort. *Forecasting of Location Occupancy by Means of Cell Phone Network Data*. Master’s Thesis, Aalto University, Espoo, 2019.
- [32] Rob J. Hyndman. Auto.arima function | R Documentation. <https://www.rdocumentation.org/packages/forecast/versions/8.12/topics/auto.arima>, March 2020.
- [33] Alysha M. De Livera, Rob J. Hyndman, and Ralph D. Snyder. Forecasting Time Series With Complex Seasonal Patterns Using Exponential Smoothing. *Journal of the American Statistical Association*, 106(496):1513–1527, 2011. ISSN 0162-1459.
- [34] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O’Reilly Media, second edition, October 2019. ISBN 978-1-4920-3264-9.
- [35] G. Peter Zhang. Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing*, 50:159–175, January 2003. ISSN 0925-2312. doi: 10.1016/S0925-2312(01)00702-0.
- [36] Dipanjan Sarkar. *Hands-On Transfer Learning with Python*. Packt Publishing, Birmingham, UK, 1st edition, 2018. ISBN 978-1-78883-130-7.

- [37] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735.
- [38] François Chollet et al. Keras. <https://keras.io>, 2015.
- [39] Krzysztof Grałbczewski. *Meta-Learning in Decision Tree Induction*, volume 498 of *Studies in Computational Intelligence*. Springer International Publishing, Cham, 2014. ISBN 978-3-319-00960-5. doi: 10.1007/978-3-319-00960-5.
- [40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.
- [41] Leo Breiman. Arcing the edge. Technical Report 486, Statistics Department, University of California, 1997.
- [42] Jerome H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, 29(5):1189–1232, 2001. ISSN 0090-5364.
- [43] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939785.
- [44] Frank Lad, Giuseppe Sanfilippo, and Gianna Agro. Completing the logarithmic scoring rule for assessing probability distributions. *AIP Conference Proceedings*, 1490:13–30, October 2012. doi: 10.1063/1.4759585.
- [45] Ane Blázquez-García, Angel Conde, Usue Mori, and Jose A. Lozano. A review on outlier/anomaly detection in time series data. *arXiv:2002.04236 [cs, stat]*, February 2020.
- [46] Jordan Hochenbaum, Owen S. Vallis, and Arun Kejariwal. Automatic Anomaly Detection in the Cloud Via Statistical Learning. *arXiv:1704.07706 [cs]*, April 2017.
- [47] Hermine N. Akouemo and Richard J. Povinelli. Time series outlier detection and imputation. In *2014 IEEE PES General Meeting | Conference Exposition*, pages 1–5, July 2014. doi: 10.1109/PESGM.2014.6939802.
- [48] Twitter. AnomalyDetection. <https://github.com/twitter/AnomalyDetection>, June 2020.
- [49] Javier López-de-Lacalle. tsoutliers: Detection of Outliers in Time Series. <https://CRAN.R-project.org/package=tsoutliers>, February 2019.

- [50] Chung Chen and Lon-Mu Liu. Joint Estimation of Model Parameters and Outlier Effects in Time Series. *Journal of the American Statistical Association*, 88(421):284–297, 1993. ISSN 0162-1459. doi: 10.2307/2290724.
- [51] Javier Lopez-de-Lacalle. tsoutliers R Package for Detection of Outliers in Time Series. <https://jalobe.com/doc/tsoutliers.pdf>, 2019.
- [52] R. Peters. Understanding the benefits and limitations of destination dispatch. In *Elevator Technology 16, Proceedings of Elevcon Helsinki 2006*, pages 258–269. The International Association of Elevator Engineers, June 2006.
- [53] G. Barney. Towards agreed traffic design definitions. *Elevator World*, 53(2), February 2005.
- [54] Seasonal decomposition of short time series | Rob J Hyndman. <https://robjhyndman.com/hyndsight/tslm-decomposition/>, March 2020.
- [55] facebook. Prophet, a forecasting tool available in Python and R. <https://facebook.github.io/prophet/>, January 2020.
- [56] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv:1603.04467 [cs]*, March 2016.
- [57] Matteo Maggilo and Gerasimos Spanakis. Autoregressive Convolutional Recurrent Neural Network for Univariate and Multivariate Time Series Prediction. *Computational Intelligence*, page 6, 2019.
- [58] Yu Gu, Zhen-Hua Ling, and Li-Rong Dai. Speech Bandwidth Extension Using Bottleneck Features and Deep Recurrent Neural Networks. In *Interspeech 2016*, pages 297–301, September 2016. doi: 10.21437/Interspeech.2016-678.
- [59] Taylor G. Smith et al. pmdarima: ARIMA estimators for Python. <http://www.alkaline-ml.com/pmdarima>, June 2017.
- [60] XGBoost Python Package. <https://xgboost.readthedocs.io/en/latest/python/index.html>, August 2020.
- [61] Nicolas Steven Miller. Pycularity. <https://github.com/zrnsn/pycularity>, July 2020.