

DLMDSME01

Automation of Standby Duty Planning for Rescue Drivers via a Forecasting Model

GitHub: https://github.com/silenNH/CaseStudyModelEngineering.git

Niels Humbeck

32003045

28.11.2021

Köln, Germany

# Table of contents

[1 Table of contents 2](#_Toc88992093)

[2 Image directory 2](#_Toc88992094)

[3 Tables 3](#_Toc88992095)

[4 Introduction: 4](#_Toc88992096)

[5 Business understanding 5](#_Toc88992097)

[6 Data Acquisition & Understanding 6](#_Toc88992098)

[6.1 Datatype, missing values and outliers 6](#_Toc88992099)

[6.2 Data wrangling & analysis 7](#_Toc88992100)

[6.3 Calculating the base line success KPIs 9](#_Toc88992101)

[7 Model Training and Evaluation 10](#_Toc88992102)

[7.1 Model requirements and library selection 10](#_Toc88992103)

[7.2 Model Training and evaluation 11](#_Toc88992104)

[8 Deployment 12](#_Toc88992105)

[9 Discussion and Conclusion 13](#_Toc88992106)

[10 Library 15](#_Toc88992107)

[11 Appendix – Code 16](#_Toc88992108)

[11.1 Data Acquisation & exploaration 16](#_Toc88992109)

[11.2 Model Training and Evaluation 36](#_Toc88992110)

[11.3 Deployment 45](#_Toc88992111)

# Image directory

[Figure 1: The Teams Data Science Process (TSDP) 4](#_Toc88992079)

[Figure 2: Boxplots for outlier analysis 7](#_Toc88992080)

[Figure 3: Histogram of standby drivers needed 7](#_Toc88992081)

[Figure 4: Histogram of further drivers needed [exceeding duty and standby] 7](#_Toc88992082)

[Figure 5: Visualization and data representation 8](#_Toc88992083)

[Figure 6: Seasonality component shows a weekly seasonality of the variable calls 8](#_Toc88992084)

[Figure 7: Month\_plot calls shows a seasonality over the year 9](#_Toc88992085)

[Figure 8: Month\_plot n\_sick shows a seasonality over the year 9](#_Toc88992086)

[Figure 9: Month\_plot drivers\_at\_work shows a seasonality over the year 9](#_Toc88992087)

[Figure 10: Linear correlation between calls and drivers\_atwork (linear coef. of 4,82 calls per driver shift) 9](#_Toc88992088)

[Figure 11: KPIs independence of interval width 12](#_Toc88992089)

[Figure 12: Model result over complete period at a interval\_width of 97.5% (number of drivers at work [black], predicted number of drivers on duty [blue], number of predicted drivers on duty + standby [red]) 12](#_Toc88992090)

[Figure 13: Prophets Diagnostic Cross Validation (Mean absolute percentage error) 12](#_Toc88992091)

# Tables

[Table 1: Base line KPIs 10](#_Toc88992092)

# Introduction:

Up to 80-85% of the data science projects are never making it into production. From the 15- 20% of completed data science projects it is assumed that only 8% generates values to the company and 92% not (Thomas 2020). One of the reasons is that not the right use cases for data since projects were found (Guasduff 2019). Another reason is the lack of professional model and feature engineering during the data science project. This use case is focusing on the proper model and feature engineering of a given real world problem: Automation of Standby Duty Planning for Rescue Drivers via a Forecasting Model

In the following work data science project follows the Teams Data Science Process methodology. In the first chapter the used methodology for the given case study in model engineering is explained. The applied framework is called “Teams Data Science Process (TDSP)”. TDSP is a framework providing a structural methodology to conduct data science projects. Based on the framework the 5 chapter provides basic understanding of the business. The focus is set here especially on the problem description, the goal of the data science project, the measures how to quantify the success of the project and the benefits. The 6 chapter is about data acquisition & understanding. After the quality of the data is accessed including its properties, data wrangling and data visualization and representation is done. In the 7 the model training and evaluation is discussed with an focus of the model selection, the evaluation based on the identified KPIs. Chapter 8 describes the deployment process into production and in chapter 9 the work is discussed and a conclusion made.

The core concept of the TDSP is depicted in the figure below (see Figure 1).

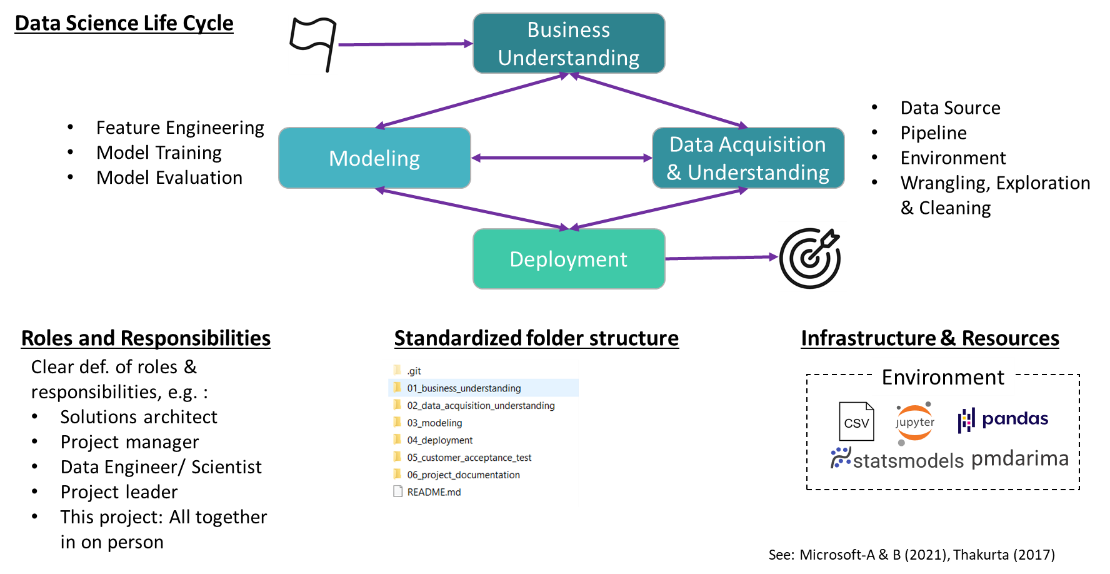


Figure 1: The Teams Data Science Process (TSDP)

The TDSP contains a data science life cycle, the standardized roles and responsibilities, the standardized folder structure and the infrastructure & resources (Microsoft-A, 2021). The data science life cycle described in a flexible and iterative manner a methodic to conduct data science project. The very first step is to understand the business and its needs. Together with the subject matter experts – who has the needed domain knowledge – the data scientist describes the problem and outlines the goal of the data science projects. It is important to define in the beginning quantifiable success measures. In the next step the data is acquired and analyzed: Which data sources are available, creating data pipelines, accessing the quality of the data, cleaning the data, exploring hidden patterns in the data. Based on the given data a model is trained in order to serve the business needs. Feature engineering is conducted before model training to provide meaningful input values. After the model is trained an evaluation is conducted. If the evaluation meet the measurable success criteria, the model is deployed in production. All of this steps are iterative and interconnected. For example a first trained model is shown to the subject matter expert to get a feedback (Microsoft-B, 2021).

The TDSP outlines the key personal roles and associated tasks. Basic roles in a data science project are: solutions architect, project manager, data engineer, data scientist, project lead (Thakurta, 2017). Besides roles and responsibilities the framework advocates clear folder structure as well as the use of a version control software to enable the team work. In the given project git hub is used as the version control system. The used folder structure is depicted in Figure 1. Furthermore TDSP gives advises for the infrastructure to use in a data science project.

# Business understanding

Berlins red-cross is a charity-oriented organization providing ambulance transport services. The organization incorporates 51000 members, 2500 volunteer workers and 1000 full time employees. Operating ambulance transports is an ethical sensible environment since lifes depends on the reliability and availability of these transports. That’s why sufficient capacities of ambulance transport is eminent to the business success.

The business faced in the past difficulties with insufficient planning of the standby/duty planning of rescue drivers. The number of rescue drivers needed highly depends on the amount of emergency calls received per day. For each day a predefined number of rescue driver as well as standby rescue drivers is on duty. Short-term sickness of rescue drivers as well as unusual high amounts of emergency calls results in an unusual high demand of rescue drivers which can exceed the amount of available rescue drivers. Unusual low short-term sickness of rescue drivers as well as unusual low amounts of emergency calls – on the other side - results in an unusual low demand of rescue drivers. In this case the amount of planned standby drivers are not needed. The goal of the given data science project is to create a model which predicts on the 15th for the upcoming month the demand of rescue drivers (inc. Standby divers) for the next month. The prediction is influenced by seasonal patterns. The success of developed model is measured by:

* percentage of standbys being activated is higher than in the current approach of keeping 90 drivers on hold
* situations with not enough standbys should occur less often than in the current approach.

A successfully deployed prediction model in the production has several benefits:

* Improved prediction of demand of rescue drivers results in less cost for providing idle capacities
* Higher reliability in duty planning returns in higher free time quality of employees
* Increase in trust in the capabilities of Berliner Red-Cross to cope with the

# Data Acquisition & Understanding

## Datatype, missing values and outliers

The data is provided in a csv file and contains daily datasets starting from the 2016-14-01 till 2019-05-27. The dataset contains following columns:

* date: entry date [Datetime index]
* n\_sick: number of drivers called sick on duty [integer]
* calls: number of emergency calls [float]
* n\_duty: number of drivers on duty available [integer]
* n\_sby: number of standby resources available [integer]
* sby\_need: number of standbys, which are activated on a given day [float]
* dafted: number of additional drivers needed due to not enough standbys [float]

It seems reasonable, that the number of sick drivers (n\_sick), the drivers on duty (n\_duty) and the number of standby resources available (n\_sby) are integer values since there are no half drivers. The columns calls, standby drivers needed (sby\_need) as well as the column dafted (number of additional drivers needed due to not enough standbys) are float values. In this use case no float values are needed. The float values were transformed into int64 datatype.

The data was inspected for **missing data**. All datasets were valid and didn’t contain missing information. Thus no data imputation is needed. In the Figure 2 are the distribution of all variables shown in a boxplot chart. The variables calls, n\_sick, n\_duty contains no unreasonable **outliers**. The variable n\_sby is a fixed number.

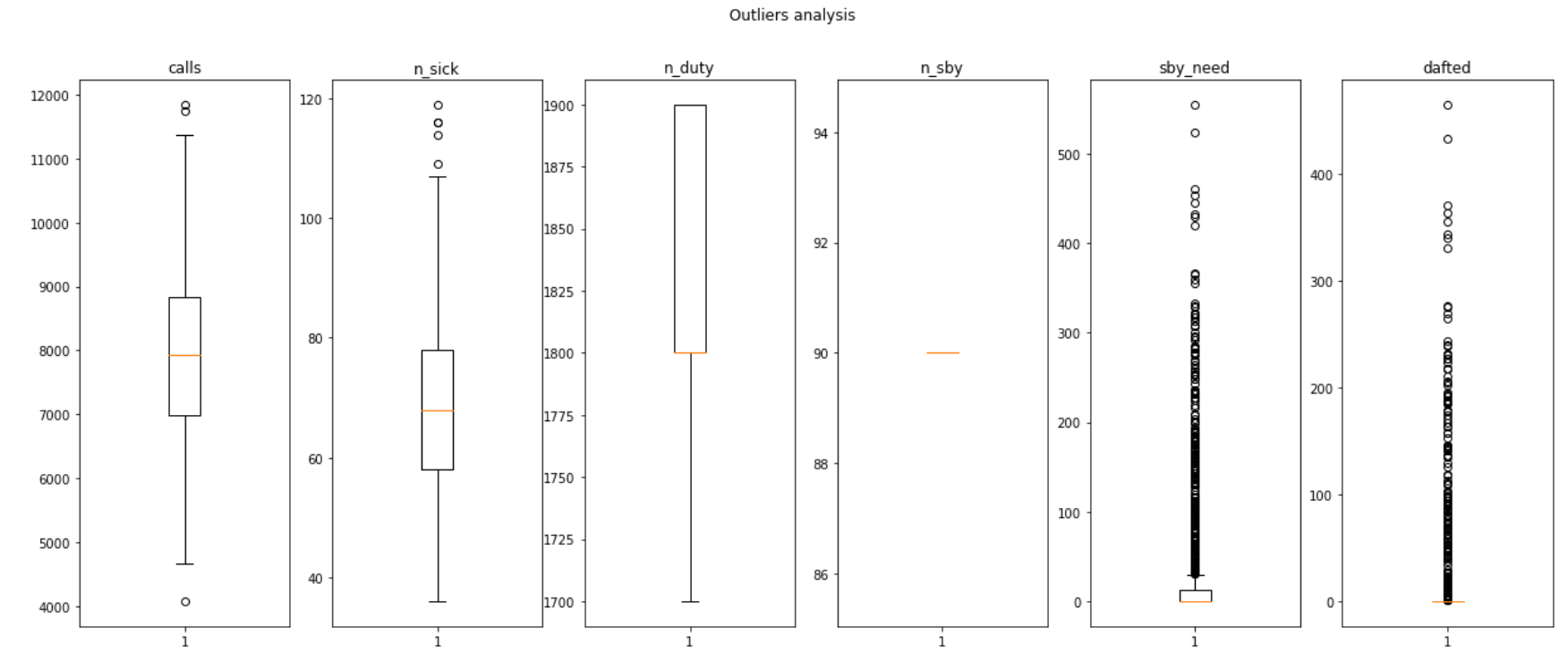


Figure 2: Boxplots for outlier analysis

The variables sby\_need and dafted show a high number of outliers. This is reasonable since sby\_need is in most cases zero unless standbys need to be activated. Same applies for the variable dafted when more than the duty + standby drivers are needed (see Figure 3 and Figure 4). All together no unreasonable outliers were detected.

|  |  |
| --- | --- |
| Figure 3: Histogram of standby drivers needed | Figure 4: Histogram of further drivers needed [exceeding duty and standby] |

## Data wrangling & analysis

For further analysis data wrangling is needed. Data can be reshaped and transformed to get valuable insights, e.g. with the Pandas function resample the average value of one month can be calculated. The Pandas DataFrame is indexed with the date column to facilitate further analysis. In some cases new columns were created by applying calculation, e.g. creating the column drivers\_atwork by adding the number drivers on duty (n\_duty) and the number of standbys needed (sby\_need).

The Figure 5 depicts the variables graphical over the given time period. The variable drivers\_atwork were created by adding n\_duty and sby\_need. The variables calls and n\_sick seems to have a seasonal as well as a trend component. The trend is positive which means over the time more calls and more drivers are sick. The variable n\_duty is jump-fixed with a trend upwards. The variable drivers\_atwork has as a lower boundary n\_duty. Beside this base line there are days where more driver needs to be at work. There seems to be a correlation between the number of calls and the days where more drivers are needed at work than the base line n\_duty and this pattern seems to have a seasonal and a trend component.

The variables sby\_need and dafted are in the most cases zero when the number of drivers on duty are sufficient. The relationship between between sby\_need & dafted can be explained as follows max(sby\_need-n\_sby;0). The pattern seems to be seasonal similar to the variable calls.

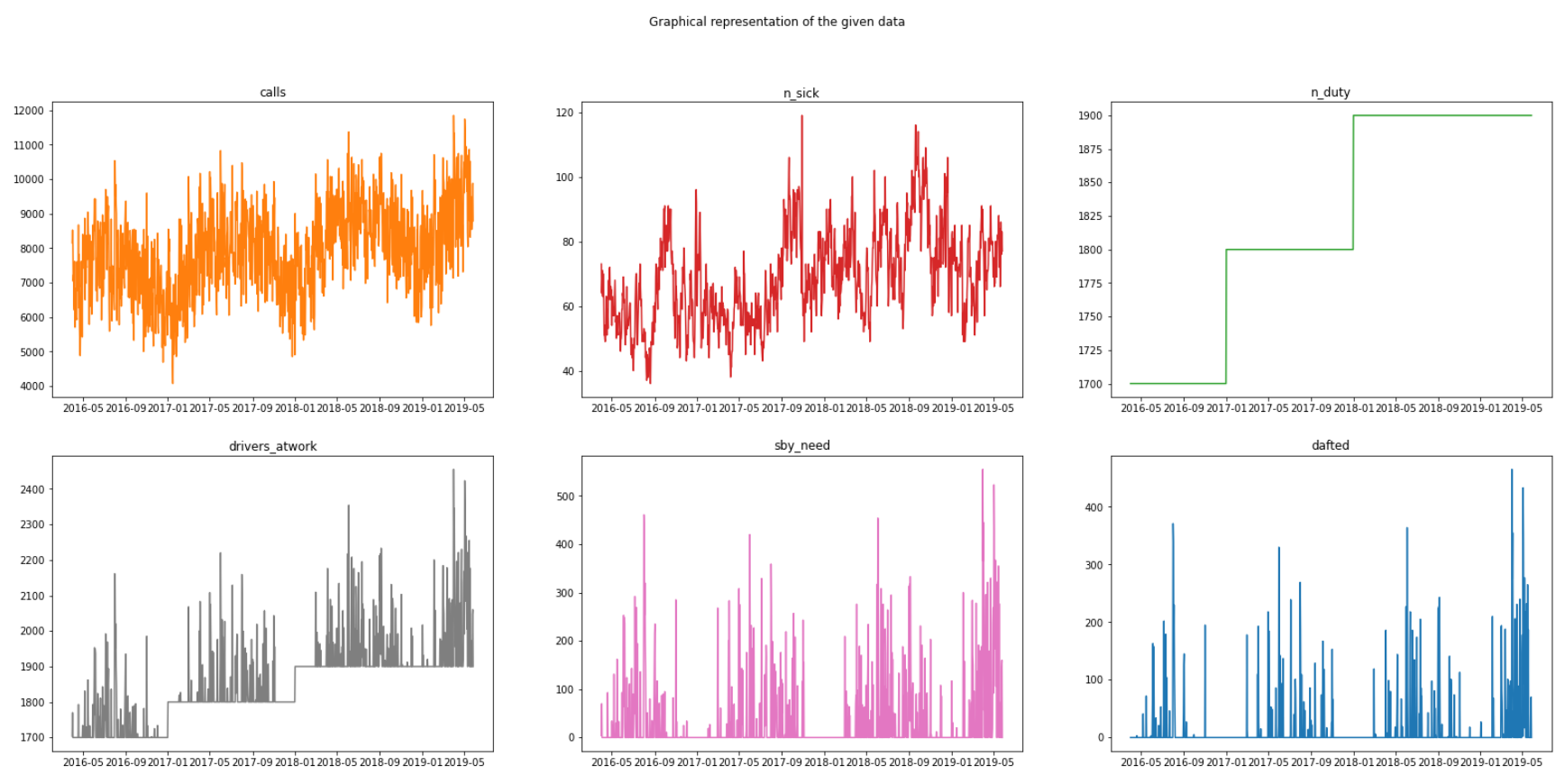


Figure 5: Visualization and data representation

The **seasonality** of the variables calls, n\_sick and drivers\_atwork were analyzed. There is a weekly trend for the variable calls as shown in Figure 6. This applies for n\_sick and drivers\_atwork as well.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Figure 6: Seasonality component shows a weekly seasonality of the variable calls

Besides the weekly seasonality exists a clear yearly seasonality for the variable calls and the variables driver\_atwork (see Figure 7 and Figure 9). The variable n\_sick seems to have a annual pattern as well where the most numbers of absenteeism occur in September and November (see Figure 8). The month plots were created by resampling the data.

|  |  |  |
| --- | --- | --- |
| Calls | N\_sick | Drivers\_at\_work |
| Ein Bild, das Antenne enthält.  Automatisch generierte Beschreibung  Figure 7: Month\_plot calls shows a seasonality over the year | Ein Bild, das Objekt, Antenne enthält.  Automatisch generierte Beschreibung  Figure 8: Month\_plot n\_sick shows a seasonality over the year | Ein Bild, das Objekt, Antenne enthält.  Automatisch generierte Beschreibung  Figure 9: Month\_plot drivers\_at\_work shows a seasonality over the year |

A **correlation analysis** has been conducted. The variables calls and drivers\_atwork are highly correlated (0,7 Pearson correlation). One distorting factor for the correlation analysis is the lower boundary of the number of drivers\_atwork (1700 , 1800, 1900). A reasonable guess is that less than the lower boundary of drivers\_atwork are needed for some periods. From a logical perspective there is a **causation** between the number of calls and the resulting number of drivers needed. An estimated guess is that there is a linear relationship between the amount calls and the resulting number of drivers\_atwork. Calculation of the linear coefficient describing best the relationship between calls and drivers at work: 4,82 calls per driver shift (with a reasonable MSE of 196). The result is shown in the Figure 10 (yellow calls/4,82 and black drivers\_atwork).

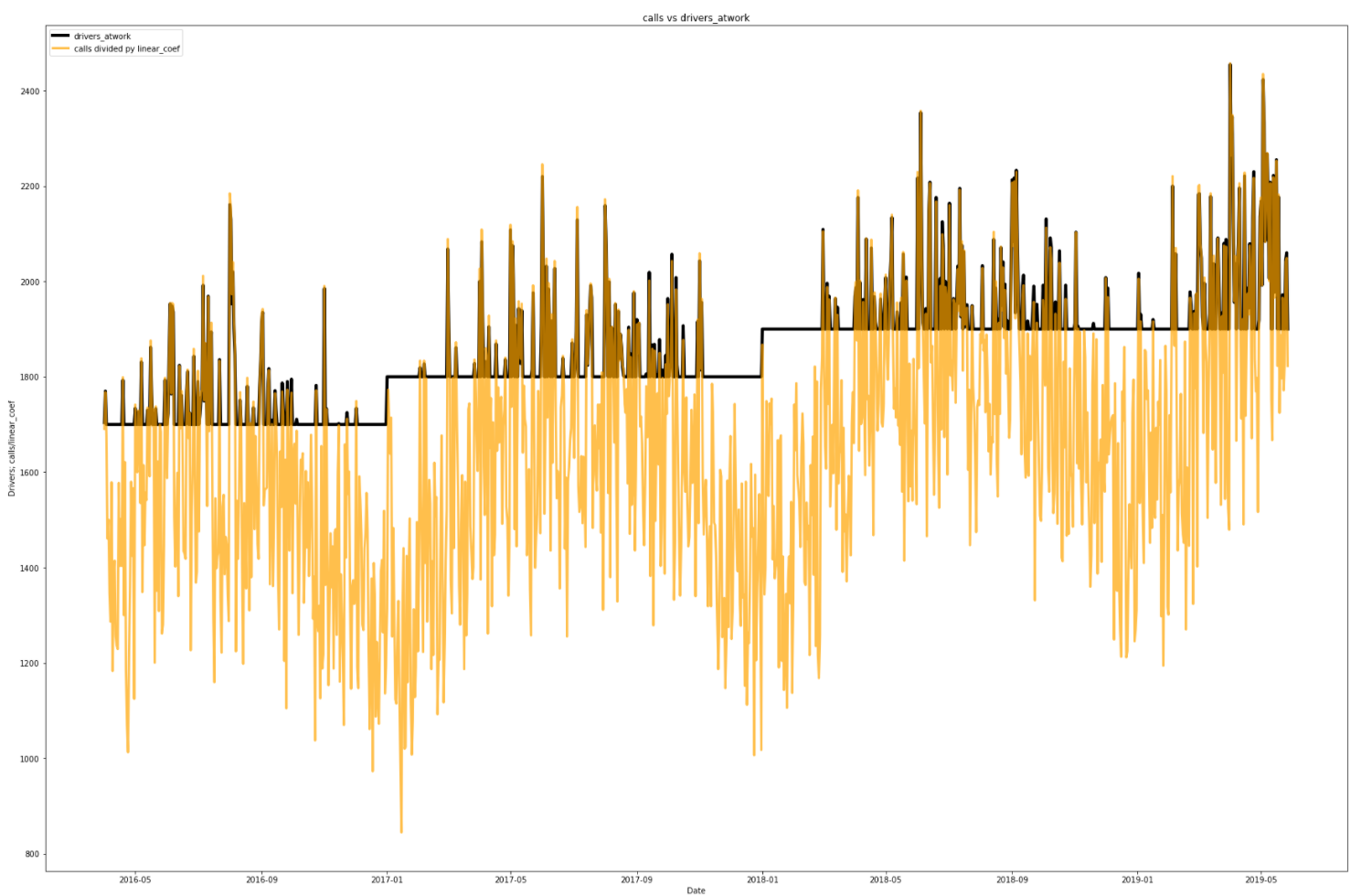


Figure 10: Linear correlation between calls and drivers\_atwork (linear coef. of 4,82 calls per driver shift)

## Calculating the base line success KPIs

In order to evaluate the prediction model KPI (Key Performance Indicators) needs to be defined and calculated for the current status. The business provided two major KPIs the prediction model needs to improve:

* Sby\_utilization: Percentage of standbys being activated is higher than in the current approach of keeping 90 drivers on hold
* Sby\_exceeded: Situations with not enough standbys should occur less often than in the current approach.

The Table 1 shows the base line KPIs for the whole dataset and for the last 365 days. The prediction model will be evaluated for the last 365 days thus the relevant KPIs are Sby\_utilization is 28,61% and Sby\_exceeded is 22,47%

Table 1: Base line KPIs

|  |  |  |
| --- | --- | --- |
|  | All data points | Last 365 days |
| Sby\_utilization | 20.43 % | 28.61 % |
| Sby\_exceeded | 14.84 % | 22.47 % |

# Model Training and Evaluation

## Model requirements and library selection

As in chapter 6 shown the given data is time series of a daily dataset with a weekly and annual seasonality. Besides the seasonality a trend is given. The prediction model should be capable to process weekly and annual seasonality as well as trends. Three possible time series predictions models were identified being capable of performing the given task. The first is SARIMAX models. The SARIMAX model is a extension of the well known ARIMA (autoregressive integrated moving average) model which is able to directly handle seasonal data. As in chapter 6 described the data contain a weekly and an annual seasonality. SARIMAX can handle to multiple seasonality with exogenous variables. Through one hot encoding each weekday is transformed to a own column and will be fed to the model as the exogenous variable. The annual seasonality is described in the period parameter m (Brownlee 2019). Unfortunately the used pmdarima.auto\_arima tool for finding the right parameter (p, d, q, P, D, Q) is unable to process such long seasonal periods sufficiently (Hyndman 2010). A work around is to transform the annual seasonality information with Fourier and feed it as a exogenous variable to the auto\_arima model with a weekly seasonality (m=7) (see Stackoverflow 2021). This approach is successful.

Another well-developed open source library is Facebooks Prophet. Prophet is able to process large datasets of data of hourly, daily or weekly observations over a period from at least one year. It can consider special days – e.g. holidays – in its predictions and trend changes. Prophet is easy to use and delivers accurate results in high quality. An additive regression model is used with four main components: yearly seasonality is modeled by a Fourier series, weekly seasonality b dummy variables (one hot encoding), user provided list of holidays and a module to detect trend changes (see Taylor 2017A and Taylor 2017B).

A third module is the exponential smoothing state space model TBATS which seems to be outdated..

In the following model training Facebooks Prophet was chosen since the predictions models were the most accurate once for the given needs with the easiest to customize.

## Model Training and evaluation

The dataset were split in a train and a test dataset. Since the annual seasonality is important the smallest test dataset is one year (365 days). The dataset contains in total 1152 rows. Thus the test data contains 31,7% of all values which is slightly higher than usual but still acceptable.

The given dataset needs to be transformed in a Prophet compatible format as a Pandas DataFrame with two columns “ds” for dates and “y” for the value on which Prophet will fit the model. As in chapter 6.2 explained exist a causation between the calls and the resulting number of drivers\_atwork. Therefore the variable calls is the “y” column.

There are several important parameter in the Prophet model. We have a linear growth model (default growth=’linear’). The daily, weekly and annual seasonality is analyzed automatically without further settings. The interval\_width provides an estimated guess based on historical data on how much & of the further datapoints will be between the lower and the upper predicted boundaries (“yhat\_upper” & “yhat\_lower”). Between the lower and upper bound is the prediction yhat of the calls (see (Prophet 2021 and Merwe 2018). For our given business problem following approach was chosen: The number of drivers on duty is estimated by the prediction variable “yhat” (predicted amount of calls) divided by the linear coefficient (4,82 calls per driver) as explained in chapter 6.2. The number of standby drivers (n\_sby) is predicted by the delta of calls “yhat\_upper” and “yhat” divided by the linear coefficient (4,82 calls per driver) as explained in chapter 6.2. The model was tested for different levels of interval\_width from 0.7 till 0.99. At interval\_width of 0.975 the KPIs indicator reaches for Sby\_exceeded 6,8% and for Sby\_utilization 44,4%. The KPI Sby\_utilization decreased with increasing interval\_width since the likelihood for exceeding the number of drivers in standby were minimized. Thus the percentage of sby\_exceeded increased as depicted in .

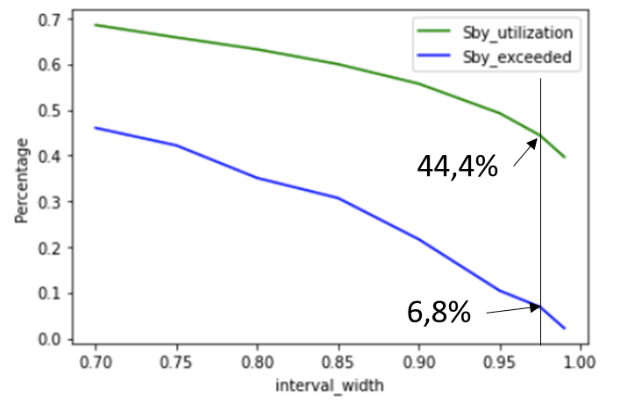


Figure 11: KPIs independence of interval width

The Figure 12 shows the model predictions over complete period at a interval\_width of 97.5% (number of drivers at work [black], predicted number of drivers on duty [blue], number of predicted drivers on duty + standby [red])

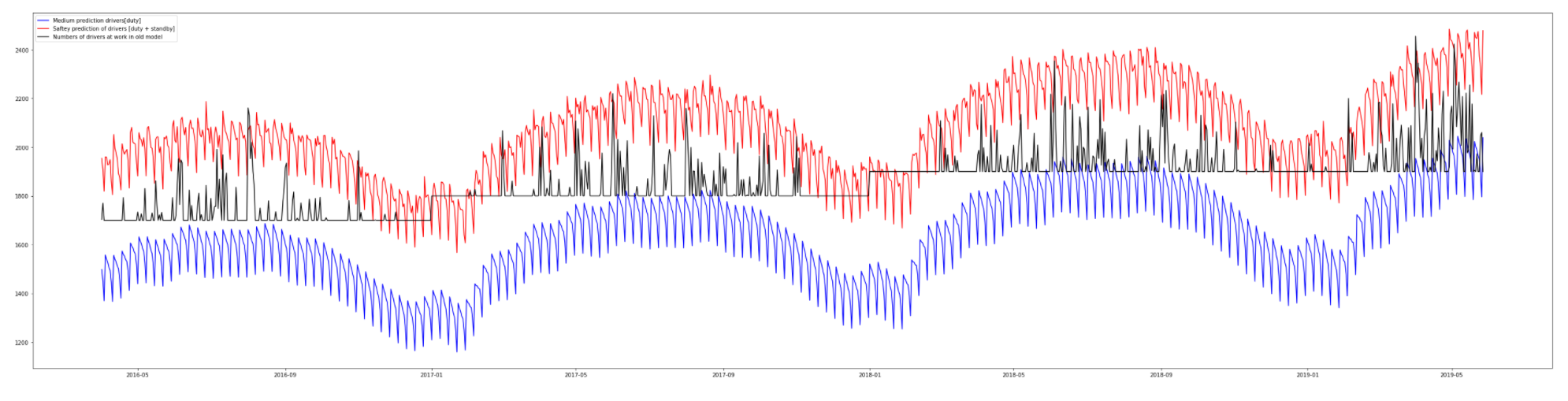


Figure 12: Model result over complete period at a interval\_width of 97.5% (number of drivers at work [black], predicted number of drivers on duty [blue], number of predicted drivers on duty + standby [red])

The root mean square error (RMSE) is 1012 and the Figure 13 shows the result of the cross validation. The mean absolute percentage error is between 8 and 12%.

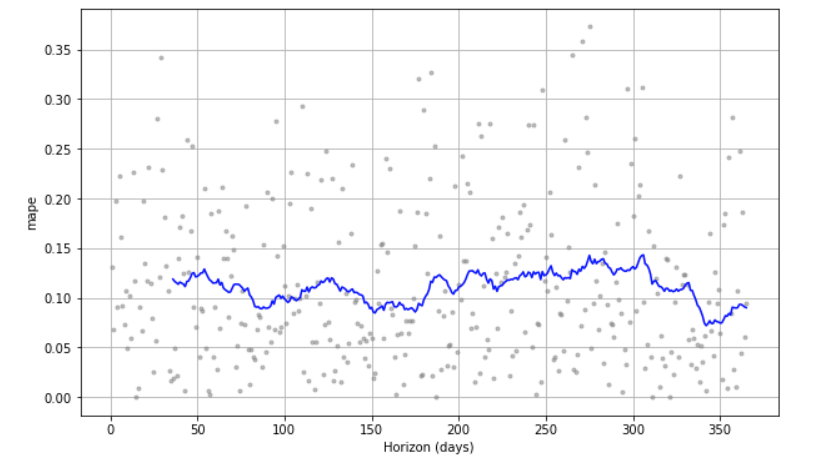


Figure 13: Prophets Diagnostic Cross Validation (Mean absolute percentage error)

# Deployment

A Jupyter notebook is created for the business to predict the needed drivers on duty (pred\_n\_duty) and the needed standby drivers (pred\_n\_sby) for the next 31 days. The model is trained by the whole datasets gathered till the date of use to improve the performance. The parameter are set as in chapter 7.2 described. An csv file is generated with the date of prediction and the pred\_n\_duty as well as pred\_n\_sby to support the fast adoption. An on the job training for at least two HR members is planned to facilitate an easy and safe start of the predictions. The HR member will have a key account partner where questions and possible issues can be answered/ resolved within 2 hours.

# Discussion and Conclusion

The aim of the project was to increase Sby\_utilization (Percentage of standbys being activated is higher than in the current approach of keeping 90 drivers on hold) and to decrease the amount of Sby\_exceeded (Situations with not enough standbys should occur less often than in the current approach). The described model increased the utilization of standby drivers from 28.61 % to 44.4%. The number of situations where the number of standbys were exceeded decreases from 22.47 % to 6.8%. Both KPIs were improved significantly.

Berliner Red-Cross provides services critical for social welfare. A key requirement for the success and the trust of the customers is a high reliability of the transport services. If not enough drivers are on duty or on standby the reliability will be impacted negatively. That’s why the given features benefiting the decrease of Sby\_exceeded over the utilization of standbys (in case of constant Sby\_exceeded a increase of Sby\_utilization to 55,8% is realistic). The parameter interval\_width is a key feature to influence the trade off between those two KPIs. The higher the interval\_width the unlikelier the amount of drivers on duty + standbys will be exceeded based on the historical dataset. With an increased interval\_width the amount of driver shifts increase as well (especially of the standby drivers) which will have a slightly negative cost effect. Since the business of Berliner Red-Cross is a critical infrastructure the reliability more important than saving money.

The mean average percentage of error for the predictions of the numbers of drivers on duty n\_duty calculated in the cross validation is between 8 and 12%. The planning based only on the number of drivers would be less accurate. Besides the number of drivers on duty the number of drivers on standby n\_sby is predicted. In the test time window of one year on 22 days the number of standbys and on duty drivers were exceeded. In the base model the number of on duty and standby drivers were exceeded at 88 days. This shows that the model is highly reliable and fulfills the business requirements to improve the utilization of standbys as well as reducing the percentage of days where the amount of drivers needed exceeds the amount of on duty drivers plus the amount of standby drivers.

# Library

* Microsoft-A (2021), Was ist der Team Data Science-Prozess (TDSP)?, <https://docs.microsoft.com/de-de/azure/architecture/data-science-process/overview>, last access: 18.11.2021 at 21:20
* Microsoft-B (2021), Lebenszyklus des Team Data Science-Prozesses, <https://docs.microsoft.com/de-de/azure/architecture/data-science-process/lifecycle>, last access: 18.11.2021 at 21:20
* Thakurta, D., McGehee, H. (2017), Team Data Science Process: Roles and tasks, <https://github.com/Azure/Microsoft-TDSP/blob/master/Docs/roles-tasks.md>, last access: 18.11.2021 at 21:21
* Deutesches Rotes Kreuz – Berlin (2021), <https://www.drk-berlin.de/>, last access 19.11.2021 at 17:58
* Brownlee, J. (2019), “ A gentle introduction to SARIMA for Time Series Forecasting in Python”, <https://machinelearningmastery.com/sarima-for-time-series-forecasting-in-python/>, last access 27.11.2021 at 17:48
* Hndman, R. (2010), Forecasting with ling seasonal periods, <https://robjhyndman.com/hyndsight/longseasonality/>, last access at 27.11.2021 at 17:50
* Stackoverflow 2021, “Forecast time series with multiple seasonality by using aut\_arima(SARIMAX) and Fourier terms, <https://stackoverflow.com/questions/68923679/forecasting-time-series-with-multiple-seasonaliy-by-using-auto-arimasarimax-an>, last access 27.11.2021 at 18:21
* Taylor, S., Letham, B. (2017A), Forecasting at Scale, <https://doi.org/10.7287/peerj.preprints.3190v>
* Taylor, S., Letham, B. (2017B), Prohept: forecasting at scale – A Blog, <https://research.fb.com/blog/2017/02/prophet-forecasting-at-scale/>, last access at 27.11.2021 at 18:30
* Prophet (2021), “Uncertaintiy Intervals”, <https://facebook.github.io/prophet/docs/uncertainty_intervals.html>, last access 27.11.2021 at 20:41
* Merwe, R. (2018), Implementing Facebook Prophet efficiently, <https://towardsdatascience.com/implementing-facebook-prophet-efficiently-c241305405a3>, last access 27.11.2021 at 20:42
* Thomas (2020), 10 reasons why data science projects fails, https://fastdatascience.com/why-do-data-science-projects-fail/ , last access at 29.05.2021 at 5:18 pm
* Guasduff, L. (2019), 3 Barriers to AI [/ML] Adoption, https://www.gartner.com/smarterwithgartner/3-barriers-to-ai-adoption/, last access 29.05.2021 at 6:26pm

# Appendix – Code

**ALL CODE IS AVAILABLE AT; https://github.com/silenNH/CaseStudyModelEngineering.git**

## Data Acquisation & exploaration

Agenda

1. General Data Properties
2. Access the quality of the data
   1. Checking incorrect value types
   2. Missing values and empty data & incorrect or invalid values
   3. Outliers and non relevant data
3. Data Wrangling & Analysis

General Data Properties

import pandas as pd  
import numpy as np   
import matplotlib.pyplot as plt  
  
  
from statsmodels.tsa.seasonal import seasonal\_decompose  
from statsmodels.graphics.tsaplots import month\_plot, quarter\_plot  
from statsmodels.tsa.stattools import adfuller  
  
%matplotlib inline

# Read in the given data and set the index column to the date column and parse the dates  
df=pd.read\_csv("../01\_data\_source/sickness\_table.csv", index\_col="date", parse\_dates=True )   
df.index.freq="D" #Set the frequence to Daily  
df.index

DatetimeIndex(['2016-04-01', '2016-04-02', '2016-04-03', '2016-04-04',  
 '2016-04-05', '2016-04-06', '2016-04-07', '2016-04-08',  
 '2016-04-09', '2016-04-10',  
 ...  
 '2019-05-18', '2019-05-19', '2019-05-20', '2019-05-21',  
 '2019-05-22', '2019-05-23', '2019-05-24', '2019-05-25',  
 '2019-05-26', '2019-05-27'],  
 dtype='datetime64[ns]', name='date', length=1152, freq='D')

df.head()

Unnamed: 0

n\_sick

calls

n\_duty

n\_sby

sby\_need

dafted

date

2016-04-01

0

73

8154.0

1700

90

4.0

0.0

2016-04-02

1

64

8526.0

1700

90

70.0

0.0

2016-04-03

2

68

8088.0

1700

90

0.0

0.0

2016-04-04

3

71

7044.0

1700

90

0.0

0.0

2016-04-05

4

63

7236.0

1700

90

0.0

0.0

df.describe()

Unnamed: 0

n\_sick

calls

n\_duty

n\_sby

sby\_need

dafted

count

1152.000000

1152.000000

1152.000000

1152.000000

1152.0

1152.000000

1152.000000

mean

575.500000

68.808160

7919.531250

1820.572917

90.0

34.718750

16.335938

std

332.698061

14.293942

1290.063571

80.086953

0.0

79.694251

53.394089

min

0.000000

36.000000

4074.000000

1700.000000

90.0

0.000000

0.000000

25%

287.750000

58.000000

6978.000000

1800.000000

90.0

0.000000

0.000000

50%

575.500000

68.000000

7932.000000

1800.000000

90.0

0.000000

0.000000

75%

863.250000

78.000000

8827.500000

1900.000000

90.0

12.250000

0.000000

max

1151.000000

119.000000

11850.000000

1900.000000

90.0

555.000000

465.000000

df.info()

<class 'pandas.core.frame.DataFrame'>  
DatetimeIndex: 1152 entries, 2016-04-01 to 2019-05-27  
Freq: D  
Data columns (total 7 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Unnamed: 0 1152 non-null int64   
 1 n\_sick 1152 non-null int64   
 2 calls 1152 non-null float64  
 3 n\_duty 1152 non-null int64   
 4 n\_sby 1152 non-null int64   
 5 sby\_need 1152 non-null float64  
 6 dafted 1152 non-null float64  
dtypes: float64(3), int64(4)  
memory usage: 72.0 KB

Columns descriptions

* date: entry date
* n\_sick: number of drivers called sick on duty
* calls: number of emergency call
* n\_duty: number of drivers on duty available
* n\_sby: number of standby resources available
* sby\_need: number of standbys, which are activated on a given day
* dafted: number of additional drivers needed due to not enough standbys

Additional informations

* Business claims, that having a daily fixed amount of standbys (n\_sby = 90) is not efficient because there are days with too many standbys followed by days with not enough standbys. The business aims at a more dynamical standby allocation, which takes seasonal patterns into account.
* Most important, the model should minimize dates with not enough standby drivers at hand!

Access the quality of the data

* Missing values and empty data
* Data imputation
* Incorrect types
* incorrect or invalid values
* outliers and non relevant data
* statistical sanitization

A: Checking incorrect value types

# What kind of variable types are given in the data frame  
df.dtypes

Unnamed: 0 int64  
n\_sick int64  
calls float64  
n\_duty int64  
n\_sby int64  
sby\_need float64  
dafted float64  
dtype: object

It seems reaonable, that the number of sick drivers (n\_sick), the drivers on duty (n\_duty), number of standby resources available (n\_sby) are integer values since there are no half drivers. The columns calls, standby drivers needed (sby\_need) as well as the column dafter (number of additional drivers needed due to not enough standbys) are float values. In this use case no float values are needed. The float values will be transformed to int64:

df["calls"] = df["calls"].apply(np.int64)  
df["sby\_need"] = df["sby\_need"].apply(np.int64)  
df["dafted"] = df["dafted"].apply(np.int64)

df.dtypes

Unnamed: 0 int64  
n\_sick int64  
calls int64  
n\_duty int64  
n\_sby int64  
sby\_need int64  
dafted int64  
dtype: object

# How much data lines includes the dataframe  
print( f"The dataframe contains: {len(df)} datasets. One dataset is one day resulting in data for roundabout {len(df)/365} years")

The dataframe contains: 1152 datasets. One dataset is one day resulting in data for roundabout 3.1561643835616437 years

B: Missing values and empty data & incorrect or invalid values

# Are values values missing:  
df.isnull().sum()

Unnamed: 0 0  
n\_sick 0  
calls 0  
n\_duty 0  
n\_sby 0  
sby\_need 0  
dafted 0  
dtype: int64

As displayed there are no missing values. And we have seen above all columns contain only integer value thus the values are valid. Thus no data imputation is needed

C: Outliers and non relevant data

fig, (fig1, fig2, fig3, fig4, fig5, fig6) = plt.subplots(1, 6)  
fig.suptitle('Outliers analysis')  
fig.set\_figheight(8)  
fig.set\_figwidth(21)  
fig1.boxplot(df["calls" ])  
fig1.set\_title('calls')  
fig2.boxplot(df["n\_sick"])  
fig2.set\_title('n\_sick')  
fig3.boxplot(df["n\_duty"])  
fig3.set\_title('n\_duty')  
fig4.boxplot(df["n\_sby"])  
fig4.set\_title('n\_sby')  
fig5.boxplot(df["sby\_need"])  
fig5.set\_title('sby\_need')  
fig6.boxplot(df["dafted"])  
fig6.set\_title('dafted');

png

png

df["sby\_need"].plot.hist(bins=50,edgecolor="k", title="Histogram of standby drivers needed").autoscale(enable=True,axis="both", tight=True)

png

png

df["dafted"].plot.hist(bins=50,edgecolor="k", title="Histogram of further drivers needed [exceeding duty and standby]").autoscale(enable=True,axis="both", tight=True)

png

png

Column "calls": - The columns calls has two small outliers around 11800.This seems to me reasonable

Column "n\_sick": - The column n\_sick has 4 outliers around 110-120. This seems to me reasonable as well

Column "n\_duty": - no major outliers can be detected

column "n\_sby": - its a fixed number --> no outliers

columns sby\_need & dafted: - The column sby\_need describes the number of standbys activated on a given day. The column "dafted" describes the number of additional drivers needed due to not enough standbys. These values are equal zero when enough drivers are on duty and no additional drivers are needed. This is the case for the most times (see Histogram). In some cases the number of drivers on duty is not sufficient and thus sby\_needed is greater then 0 and if this number exeeds 90 the number duafted is greater then 0 as well. - The boxplots diagrams show a lot of outliers here. This is reasonable since just in a few cases standby drivers are needed as well as dafted drivers.

All together no unreasonable outliers are detected in the dataset

Data Wrangling & Analysis

Data Wrangling

* Reshaping and transforming structures
* Indexing data for quick access (already done in chapter one)
* Merging, combining and joining data

Analysis

* Exploration
* Visualization and representation
* Correlation vs Causation analysis
* Statistical analysis

A: Merging, combining and joining data

* the column of the number of drivers working at the given date is not calculated yet
* drivers\_atwork= n\_duty + sby\_need

df["drivers\_atwork"]=df["n\_duty"]+df["sby\_need"]

B: Visaulization and representation

Basic visualization

fig, ((fig1, fig2, fig3), (fig4, fig5, fig6))= plt.subplots(2, 3)  
fig.suptitle('Graphical representation of the given data')  
fig.set\_figheight(12)  
fig.set\_figwidth(27)  
fig1.plot(df["calls" ], "tab:orange")  
fig1.set\_title('calls')  
fig2.plot(df["n\_sick"], "tab:red")  
fig2.set\_title('n\_sick')  
fig3.plot(df["n\_duty"], "tab:green")  
fig3.set\_title('n\_duty')  
fig4.plot(df["drivers\_atwork"], "tab:grey")  
fig4.set\_title('drivers\_atwork')  
fig5.plot(df["sby\_need"], "tab:pink")  
fig5.set\_title('sby\_need')  
fig6.plot(df["dafted"], "tab:blue")  
fig6.set\_title('dafted');  
  
#n\_sby is not considered here since its a fixed value and thus boring

png

png

Column "calls": - the calls seems to have a trend (more calls with increasing times) as well as a seasonal (seasonal fluctuation) component - there is definitely some noice in the data - Analysis of trend and sesonality is needed!

Column "n\_sick": - the calls seems to have a weak trend (more calls with increasing times) as well as a weak seasonal (seasonal fluctuation) component --> closer look is necessary - Analysis of trend and sesonality is needed!

Column "n\_duty": - the number of planned duty drivers is jump-fixed

column "drivers\_atwork": - the buttom line is the number of drivers on duty - besides the base line, there are days where more drivers needs to be at work - the pattern of days where more drivers has to be at work seems to be seasonal --> closer look is needed - it seems that the amount of drivers at work is correlating with the number of calls --> closer look is needed

columns sby\_need & dafted: - base line is 0 - sby\_need and dafted are connected --> max(sby\_need-n\_sby;0) - The peaks of the data seems to be seasonal

Analyzing Trends & Seasonalities

Calls, n\_sick & drivers\_atwork seems to have a seasonal and a trend component. Further evaluation are conducted in the following.

def test\_adfuller(series,title=''):  
 """  
 A Augmented Dickey-Fuller Test-Report is created based on the given time series and an optional title  
 """  
 print(f'DF-Test: {title}')  
 result = adfuller(series.dropna(),autolag='AIC')  
 labels = ['ADF test statistic','p-value','# lags used','# observations']  
 out = pd.Series(result[0:4],index=labels)  
  
 for key,val in result[4].items():  
 out[f'critical value ({key})']=val  
 print(out.to\_string())   
   
 if result[1] <= 0.05:  
 print("Strong evidence against the null hypothesis")  
 print("Reject of the null hypothesis")  
 print("Data is stationary")  
 else:  
 print("Weak evidence against the null hypothesis")  
 print("Fail to reject the null hypothesis")  
 print("Data is non-stationary")

#calls  
results=seasonal\_decompose(df["calls"], model="add")  
results.plot();

png

png

results.seasonal.head(59)

date  
2016-04-01 -34.035301  
2016-04-02 -212.149504  
2016-04-03 -637.308136  
2016-04-04 312.846232  
2016-04-05 299.685953  
2016-04-06 201.328810  
2016-04-07 69.631946  
2016-04-08 -34.035301  
2016-04-09 -212.149504  
2016-04-10 -637.308136  
2016-04-11 312.846232  
2016-04-12 299.685953  
2016-04-13 201.328810  
2016-04-14 69.631946  
2016-04-15 -34.035301  
2016-04-16 -212.149504  
2016-04-17 -637.308136  
2016-04-18 312.846232  
2016-04-19 299.685953  
2016-04-20 201.328810  
2016-04-21 69.631946  
2016-04-22 -34.035301  
2016-04-23 -212.149504  
2016-04-24 -637.308136  
2016-04-25 312.846232  
2016-04-26 299.685953  
2016-04-27 201.328810  
2016-04-28 69.631946  
2016-04-29 -34.035301  
2016-04-30 -212.149504  
2016-05-01 -637.308136  
2016-05-02 312.846232  
2016-05-03 299.685953  
2016-05-04 201.328810  
2016-05-05 69.631946  
2016-05-06 -34.035301  
2016-05-07 -212.149504  
2016-05-08 -637.308136  
2016-05-09 312.846232  
2016-05-10 299.685953  
2016-05-11 201.328810  
2016-05-12 69.631946  
2016-05-13 -34.035301  
2016-05-14 -212.149504  
2016-05-15 -637.308136  
2016-05-16 312.846232  
2016-05-17 299.685953  
2016-05-18 201.328810  
2016-05-19 69.631946  
2016-05-20 -34.035301  
2016-05-21 -212.149504  
2016-05-22 -637.308136  
2016-05-23 312.846232  
2016-05-24 299.685953  
2016-05-25 201.328810  
2016-05-26 69.631946  
2016-05-27 -34.035301  
2016-05-28 -212.149504  
2016-05-29 -637.308136  
Freq: D, Name: seasonal, dtype: float64

results.seasonal.plot(figsize=(50,8));

png

png

test\_adfuller(df["calls"], "ADF Test for calls on a daily basis")

DF-Test: ADF Test for calls on a daily basis  
ADF test statistic -2.761936  
p-value 0.063920  
# lags used 19.000000  
# observations 1132.000000  
critical value (1%) -3.436140  
critical value (5%) -2.864097  
critical value (10%) -2.568131  
Weak evidence against the null hypothesis  
Fail to reject the null hypothesis  
Data is non-stationary

calls\_m =df["calls"].resample(rule="MS").mean()

results=seasonal\_decompose(calls\_m, model="add")  
results.plot();

png

png

month\_plot(calls\_m);

png

png

test\_adfuller(calls\_m, "ADF Test for calls on a month basis")

DF-Test: ADF Test for calls on a month basis  
ADF test statistic 0.399761  
p-value 0.981486  
# lags used 10.000000  
# observations 27.000000  
critical value (1%) -3.699608  
critical value (5%) -2.976430  
critical value (10%) -2.627601  
Weak evidence against the null hypothesis  
Fail to reject the null hypothesis  
Data is non-stationary

calls\_w =df["calls"].resample(rule="W").mean()  
results=seasonal\_decompose(calls\_w, model="add")  
results.plot();

png

png

test\_adfuller(calls\_w, "ADF Test for calls on a weekly basis")

DF-Test: ADF Test for calls on a weekly basis  
ADF test statistic -2.811003  
p-value 0.056719  
# lags used 10.000000  
# observations 155.000000  
critical value (1%) -3.473259  
critical value (5%) -2.880374  
critical value (10%) -2.576812  
Weak evidence against the null hypothesis  
Fail to reject the null hypothesis  
Data is non-stationary

# n\_sick  
results=seasonal\_decompose(df["n\_sick"], model="add")  
results.plot();

png

png

results.seasonal.plot(figsize=(50,8));

png

png

test\_adfuller(df["n\_sick"], "ADF Test for n\_sick on a daily basis")

DF-Test: ADF Test for n\_sick on a daily basis  
ADF test statistic -3.374317  
p-value 0.011866  
# lags used 21.000000  
# observations 1130.000000  
critical value (1%) -3.436150  
critical value (5%) -2.864101  
critical value (10%) -2.568134  
Strong evidence against the null hypothesis  
Reject of the null hypothesis  
Data is stationary

n\_sick\_m =df["n\_sick"].resample(rule="MS").mean()  
results=seasonal\_decompose(n\_sick\_m, model="add")  
results.plot();

png

png

month\_plot(n\_sick\_m);

png

png

test\_adfuller(n\_sick\_m, "ADF Test for n\_sick on a monthly basis")

DF-Test: ADF Test for n\_sick on a monthly basis  
ADF test statistic -1.795900  
p-value 0.382499  
# lags used 2.000000  
# observations 35.000000  
critical value (1%) -3.632743  
critical value (5%) -2.948510  
critical value (10%) -2.613017  
Weak evidence against the null hypothesis  
Fail to reject the null hypothesis  
Data is non-stationary

n\_sick\_w =df["n\_sick"].resample(rule="W").mean()  
results=seasonal\_decompose(n\_sick\_w, model="add")  
results.plot();

png

png

test\_adfuller(n\_sick\_w, "ADF Test for n\_sick on a weekly basis")

DF-Test: ADF Test for n\_sick on a weekly basis  
ADF test statistic -3.796061  
p-value 0.002948  
# lags used 1.000000  
# observations 164.000000  
critical value (1%) -3.470866  
critical value (5%) -2.879330  
critical value (10%) -2.576255  
Strong evidence against the null hypothesis  
Reject of the null hypothesis  
Data is stationary

#drivers\_atwork

results=seasonal\_decompose(df["drivers\_atwork"], model="add")  
results.plot();

png

png

results.seasonal.plot(figsize=(50,12))

<AxesSubplot:xlabel='date'>

png

png

test\_adfuller(df["drivers\_atwork"], "ADF Test for drivers\_atwork on a daily basis")

DF-Test: ADF Test for drivers\_atwork on a daily basis  
ADF test statistic -1.843787  
p-value 0.358949  
# lags used 21.000000  
# observations 1130.000000  
critical value (1%) -3.436150  
critical value (5%) -2.864101  
critical value (10%) -2.568134  
Weak evidence against the null hypothesis  
Fail to reject the null hypothesis  
Data is non-stationary

drivers\_atwork\_m=df["drivers\_atwork"].resample(rule="MS").mean()  
results=seasonal\_decompose(drivers\_atwork\_m, model="add")  
results.plot();

png

png

month\_plot(drivers\_atwork\_m);

png

png

test\_adfuller(drivers\_atwork\_m, "ADF Test for drivers\_atwork on a monthly basis")

DF-Test: ADF Test for drivers\_atwork on a monthly basis  
ADF test statistic -0.556377  
p-value 0.880538  
# lags used 10.000000  
# observations 27.000000  
critical value (1%) -3.699608  
critical value (5%) -2.976430  
critical value (10%) -2.627601  
Weak evidence against the null hypothesis  
Fail to reject the null hypothesis  
Data is non-stationary

drivers\_atwork\_w=df["drivers\_atwork"].resample(rule="W").mean()  
results=seasonal\_decompose(drivers\_atwork\_w, model="add")  
results.plot();

png

png

test\_adfuller(drivers\_atwork\_w, "ADF Test for drivers\_atwork on a weekly basis")

DF-Test: ADF Test for drivers\_atwork on a weekly basis  
ADF test statistic -1.183565  
p-value 0.680626  
# lags used 4.000000  
# observations 161.000000  
critical value (1%) -3.471633  
critical value (5%) -2.879665  
critical value (10%) -2.576434  
Weak evidence against the null hypothesis  
Fail to reject the null hypothesis  
Data is non-stationary

month\_plot(drivers\_atwork\_m);

png

png

Correlation and Causation analysis

df\_calls\_driveratwork=df[["calls", "drivers\_atwork"]]

df\_calls\_driveratwork.corr(method='pearson')

calls

drivers\_atwork

calls

1.000000

0.704728

drivers\_atwork

0.704728

1.000000

#adjust the dataframe by the values where the base line is hit  
df\_adjusted = df.loc[df["drivers\_atwork"] != 1770.0]  
df\_adjusted = df\_adjusted.loc[df\_adjusted["drivers\_atwork"] != 1700.0]  
df\_adjusted = df\_adjusted.loc[df\_adjusted["drivers\_atwork"] != 1800.0]  
df\_adjusted = df\_adjusted.loc[df\_adjusted["drivers\_atwork"] != 1900.0]  
df\_adjusted.head()

Unnamed: 0

n\_sick

calls

n\_duty

n\_sby

sby\_need

dafted

drivers\_atwork

date

2016-04-01

0

73

8154

1700

90

4

0

1704

2016-04-19

18

57

8676

1700

90

93

3

1793

2016-05-01

30

54

8400

1700

90

34

0

1734

2016-05-04

33

58

8340

1700

90

26

0

1726

2016-05-07

36

57

8868

1700

90

131

41

1831

Min\_MSEError=10\*\*1000  
x\_Value=0  
for x in np.linspace(2, 6, num=1000):  
 ErrorSUM=0  
 #print(x)  
 for counter1 in range(0,len(df\_adjusted)):  
 a=float(df\_adjusted.iloc[counter1, 7]) # drivers\_atwork  
 b=float(df\_adjusted.iloc[counter1, 2]) # calls  
 ErrorSUM=ErrorSUM + (a-(b/x))\*\*2  
 if Min\_MSEError > (ErrorSUM/len(df\_adjusted)):  
 Min\_MSEError=ErrorSUM/len(df\_adjusted)  
 x\_Value=x  
   
print ("Min\_MSEError", Min\_MSEError)  
print("Best linear\_coef value: ", x\_Value)  
  
  
fig = plt.figure(figsize=(30,20))  
ax = fig.add\_subplot(111)  
ax.set\_title('calls vs drivers\_atwork')  
plt.plot(df["drivers\_atwork"],color='black',label='drivers\_atwork', lw=4)  
plt.plot(df["calls"]/x\_Value,color='orange',label="calls divided py linear\_coef", alpha=0.7, lw=3)  
ax.set\_xlabel('Date')  
ax.set\_ylabel('Drivers; calls/linear\_coef')  
ax.legend(loc='upper left')  
plt.show()  
  
#plt.figure(figsize=(30,30))  
#labels=["drivers\_atwork","calls divided py linear\_coef"]  
#plt.plot(df["drivers\_atwork"], label="drivers\_atwork")  
#plt.plot(df["calls"]/x\_Value, label="calls divided py linear\_coef")  
#plt.legend()  
#plt.show()

Min\_MSEError 196.22728826407905  
Best linear\_coef value: 4.822822822822823

png

png

plt.hist(df\_adjusted["sby\_need"])

(array([88., 76., 43., 33., 29., 20., 4., 3., 3., 2.]),  
 array([ 2. , 57.3, 112.6, 167.9, 223.2, 278.5, 333.8, 389.1, 444.4,  
 499.7, 555. ]),  
 <BarContainer object of 10 artists>)

png

png

Calculating the base line success KPIs to compare the model with status quo

* Percentage of standbys being activated is higher than in the current approach of keeping 90 drivers on hold
* Situations with not enough standbys should occur less often than in the current approach.

A: Whole dataset

# Percentage of standbys being activated is higher than in the current approach of keeping 90 drivers on hold  
df["sby\_activated"]=df["sby\_need"]-df["dafted"]  
df["percentage\_sby\_activated"]=df["sby\_activated"]/df["n\_sby"]  
percentage\_standby\_activated=df["percentage\_sby\_activated"].mean()  
print(f'{np.round(percentage\_standby\_activated\*100,2)} % of the standby drivers are activated over the whole dataset before optimisation')

20.43 % of the standby drivers are activated over the whole dataset before optimisation

# Situations with not enough standbys should occur less often than in the current approach.  
print(f'In {np.round(len(df[df["sby\_need"]>90])/len(df)\*100,2)} % of the days the amount of 90 standby drivers were exceeded')

In 14.84 % of the days the amount of 90 standby drivers were exceeded

B: The last 365 to compare with test data set

# Percentage of standbys being activated is higher than in the current approach of keeping 90 drivers on hold  
df["sby\_activated"]=df["sby\_need"]-df["dafted"]  
df["percentage\_sby\_activated"]=df["sby\_activated"]/df["n\_sby"]  
percentage\_standby\_activated=df["percentage\_sby\_activated"].iloc[len(df)-365:].mean()  
print(f'{np.round(percentage\_standby\_activated\*100,2)} % of the standby drivers are activated over the whole dataset before optimisation')

28.61 % of the standby drivers are activated over the whole dataset before optimisation

df["datfted\_for\_calc"]=df["dafted"].apply(lambda x:x if x>0 else np.nan)  
print(f'In {np.round(df["datfted\_for\_calc"].iloc[len(df)-365:].count()/365\*100,2)} % of the days the amount of 90 standby drivers were exceeded')

In 22.47 % of the days the amount of 90 standby drivers were exceeded

df.head()

Unnamed: 0

n\_sick

calls

n\_duty

n\_sby

sby\_need

dafted

drivers\_atwork

sby\_activated

percentage\_sby\_activated

datfted\_for\_calc

date

2016-04-01

0

73

8154

1700

90

4

0

1704

4

0.044444

NaN

2016-04-02

1

64

8526

1700

90

70

0

1770

70

0.777778

NaN

2016-04-03

2

68

8088

1700

90

0

0

1700

0

0.000000

NaN

2016-04-04

3

71

7044

1700

90

0

0

1700

0

0.000000

NaN

2016-04-05

4

63

7236

1700

90

0

0

1700

0

0.000000

NaN

C: Calculating the consumed FTE (full time equivilant) shifts as a third comparison parameter (not the leading indicator ... but never the less interesting)

print(f'In the last 365 days {df["n\_duty"].iloc[len(df)-365:].sum() + df["sby\_need"].iloc[len(df)-365:].sum()} FTE shifts were needed')  
print(f'In the last 365 days {df["n\_sby"].iloc[len(df)-365:].sum() - df["sby\_activated"].iloc[len(df)-365:].sum()} FTE shifts were hold on standby and were not needed')

In the last 365 days 713377 FTE shifts were needed  
In the last 365 days 23451 FTE shifts were hold on standby and were not needed

np.round(df["datfted\_for\_calc"].iloc[len(df)-365:].count(),2)

82

## Model Training and Evaluation

import pandas as pd  
import numpy as np  
from fbprophet import Prophet  
import matplotlib.pyplot as plt  
from statsmodels.tools.eval\_measures import rmse  
from fbprophet.diagnostics import cross\_validation,performance\_metrics  
from fbprophet.plot import plot\_cross\_validation\_metric

# Read in the given data and set the index column to the date column and parse the dates  
df=pd.read\_csv("../../02\_data\_acquisition\_understanding/01\_data\_source/sickness\_table.csv", index\_col="date", parse\_dates=True )   
df["drivers\_atwork"]=df["n\_duty"]+df["sby\_need"]  
df.index.freq="D" #Set the frequence to Daily  
df.index

DatetimeIndex(['2016-04-01', '2016-04-02', '2016-04-03', '2016-04-04',  
 '2016-04-05', '2016-04-06', '2016-04-07', '2016-04-08',  
 '2016-04-09', '2016-04-10',  
 ...  
 '2019-05-18', '2019-05-19', '2019-05-20', '2019-05-21',  
 '2019-05-22', '2019-05-23', '2019-05-24', '2019-05-25',  
 '2019-05-26', '2019-05-27'],  
 dtype='datetime64[ns]', name='date', length=1152, freq='D')

# Prepare the Data for Prophet  
# Create the column "date"  
df["date"]=df.index  
# Create new DataFrames with the column "date"  
df\_new=pd.DataFrame(df["date"])  
# Create the column y="calls"  
df\_new["y"]=df["calls"]  
# Delete the index  
df\_new.reset\_index(drop=True, inplace=True)  
# Rename the columns to the Prophet specifics  
df\_new.rename(columns={"date": "ds", "y": "y"}, inplace=True)  
# Show the head of the DataFrame  
df\_new.head()

ds

y

0

2016-04-01

8154.0

1

2016-04-02

8526.0

2

2016-04-03

8088.0

3

2016-04-04

7044.0

4

2016-04-05

7236.0

# Split the Data in a test and a split data set  
train = df\_new.iloc[:len(df\_new)-365]  
test = df\_new.iloc[len(df\_new)-365:]  
test.set\_index(test["ds"], inplace=True)  
# Get the time window for the predicted times   
df\_new.iloc[len(df\_new)-365:]

ds

y

787

2018-05-28

8862.0

788

2018-05-29

8226.0

789

2018-05-30

8064.0

790

2018-05-31

7392.0

791

2018-06-01

10752.0

...

...

...

1147

2019-05-23

8544.0

1148

2019-05-24

8814.0

1149

2019-05-25

9846.0

1150

2019-05-26

9882.0

1151

2019-05-27

8790.0

365 rows × 2 columns

test

ds

y

ds

2018-05-28

2018-05-28

8862.0

2018-05-29

2018-05-29

8226.0

2018-05-30

2018-05-30

8064.0

2018-05-31

2018-05-31

7392.0

2018-06-01

2018-06-01

10752.0

...

...

...

2019-05-23

2019-05-23

8544.0

2019-05-24

2019-05-24

8814.0

2019-05-25

2019-05-25

9846.0

2019-05-26

2019-05-26

9882.0

2019-05-27

2019-05-27

8790.0

365 rows × 2 columns

#Create the Prohpet-Model,with an interval\_width of 97,5% (97,5% of the train values are in the predicted range)   
m = Prophet(interval\_width=0.975, daily\_seasonality=True,seasonality\_mode="multiplicative")  
# Train the Prophet-Model  
m.fit(train)  
# Create future datafame with xx days (periods & freq) as basis for prediction  
future = m.make\_future\_dataframe(periods=365,freq='D')  
# Make predictions for the created future dataframe  
forecast = m.predict(future)  
#plot the predictions   
m.plot(forecast, figsize=(20, 6));  
forecast.set\_index(forecast["ds"], inplace=True)  
forecast["n\_RealCalls"]=df["calls"]

png

png

fig = plt.figure(figsize=(50,12))  
ax = fig.add\_subplot(1, 1, 1)  
ax.plot(forecast["ds"].iloc[len(forecast)-365:], forecast["yhat"].iloc[len(forecast)-365:], label='Medium prediction calls', c="blue")  
ax.plot(forecast["ds"].iloc[len(forecast)-365:], forecast["yhat\_upper"].iloc[len(forecast)-365:], label='Saftey prediction of calls', c="red")  
ax.plot(forecast["ds"].iloc[len(forecast)-365:],forecast["n\_RealCalls"].iloc[len(forecast)-365:],label="Real amount of Calls", c="black")  
ax.legend(loc='upper left')

<matplotlib.legend.Legend at 0x24e571128c8>

png

png

#ratio calls/ drivers:   
#adjust the dataframe by the values where the base line is hit  
df\_adjusted = df.loc[df["drivers\_atwork"] != 1770.0]  
df\_adjusted = df\_adjusted.loc[df\_adjusted["drivers\_atwork"] != 1700.0]  
df\_adjusted = df\_adjusted.loc[df\_adjusted["drivers\_atwork"] != 1800.0]  
df\_adjusted = df\_adjusted.loc[df\_adjusted["drivers\_atwork"] != 1900.0]  
df\_adjusted.head()  
  
Min\_MSEError=10\*\*1000  
x\_Value=0  
for x in np.linspace(2, 6, num=1000):  
 ErrorSUM=0  
 #print(x)  
 for counter1 in range(0,len(df\_adjusted)):  
 a=float(df\_adjusted.iloc[counter1, 7]) # drivers\_atwork  
 b=float(df\_adjusted.iloc[counter1, 2]) # calls  
 ErrorSUM=ErrorSUM + (a-(b/x))\*\*2  
 if Min\_MSEError > (ErrorSUM/len(df\_adjusted)):  
 Min\_MSEError=ErrorSUM/len(df\_adjusted)  
 x\_Value=x  
   
print ("Min\_MSEError", Min\_MSEError)  
print("Best linear\_coef value: ", x\_Value)  
  
  
fig = plt.figure(figsize=(30,20))  
ax = fig.add\_subplot(111)  
ax.set\_title('calls vs drivers\_atwork')  
plt.plot(df["drivers\_atwork"],color='black',label='drivers\_atwork', lw=4)  
plt.plot(df["calls"]/x\_Value,color='orange',label="calls divided py linear\_coef", alpha=0.7, lw=3)  
ax.set\_xlabel('Date')  
ax.set\_ylabel('Drivers; calls/linear\_coef')  
ax.legend(loc='upper left')  
plt.show()

Min\_MSEError 196.22728826407905  
Best linear\_coef value: 4.822822822822823

png

png

#List of interval\_width to be evaluated  
liste=[0.7,0.75,0.80, 0.85, 0.9,0.95,0.975, 0.99]  
  
data=[]  
for i in liste:  
 #Create the Prohpet-Model,with an interval\_width of 95% (95% of the train values are in the predicted range)   
 m = Prophet(interval\_width=i, daily\_seasonality=True,seasonality\_mode="multiplicative")  
 # Train the Prophet-Model  
 m.fit(train)  
 # Create future datafame with xx days (periods & freq) as basis for prediction  
 future = m.make\_future\_dataframe(periods=365,freq='D')  
 # Make predictions for the created future dataframe  
 forecast = m.predict(future)  
 #plot the predictions   
 #m.plot(forecast, figsize=(20, 6));  
 forecast.set\_index(forecast["ds"], inplace=True)  
 forecast["n\_RealCalls"]=df["calls"]  
  
  
 df\_eval=pd.DataFrame(forecast["ds"].iloc[len(forecast)-365:])  
 df\_eval["pred\_n\_duty"]=forecast["yhat"].iloc[len(forecast)-365:]/x\_Value  
 df\_eval["pred\_n\_sby"]=forecast["yhat\_upper"].iloc[len(forecast)-365:]/x\_Value-forecast["yhat"].iloc[len(forecast)-365:]/x\_Value  
 df\_eval["real\_drivers\_atwork"]=df["drivers\_atwork"].iloc[len(forecast)-365:]  
 df\_eval["Delta"]=df\_eval["pred\_n\_duty"]-df\_eval["real\_drivers\_atwork"]  
 df\_eval["Delta2"]=df\_eval["pred\_n\_duty"] + df\_eval["pred\_n\_sby"] -df\_eval["real\_drivers\_atwork"]  
  
 df\_eval["pred\_sbyNeed"]= df\_eval["Delta"].apply(lambda x: -x if x < 0 else 0)  
 df\_eval["pred\_dafted"]= df\_eval["Delta2"].apply(lambda x: -x if x < 0 else np.nan)  
 df\_eval["pred\_utilizationSby"]=df\_eval["pred\_sbyNeed"]/df\_eval["pred\_n\_sby"]  
 df\_eval["pred\_utilizationSby"]= df\_eval["pred\_utilizationSby"].apply(lambda x: x if x < 1 else 1)  
  
 df\_eval["real\_n\_sby"]=df["n\_sby"].iloc[len(forecast)-365:]  
 df\_eval["real\_sby\_need"]=df["sby\_need"].iloc[len(forecast)-365:]  
 df\_eval["real\_dafted"]=df["dafted"].iloc[len(forecast)-365:]  
 utilization=df\_eval["pred\_utilizationSby"].mean()  
 Overexceed=df\_eval["pred\_dafted"].count()/365   
 FTE\_shifte\_needed=df\_eval["pred\_n\_duty"].sum() + df\_eval["pred\_sbyNeed"].sum()  
 FTE\_standbyShifts\_withoutDuty\_needed=df\_eval["pred\_n\_sby"].sum() - df\_eval["pred\_sbyNeed"].sum() + df\_eval["pred\_dafted"].iloc[len(df)-365:].sum()  
   
   
   
 data.append([i, utilization, Overexceed,FTE\_shifte\_needed,FTE\_standbyShifts\_withoutDuty\_needed])  
  
   
 #print(f'Schritt-{i}:In the last 365 days {df\_eval["pred\_n\_duty"].sum() + df\_eval["pred\_sbyNeed"].sum()} FTE shifts we needed')  
 #print(f'Schritt-{i}:In the last 365 days {df\_eval["pred\_n\_sby"].sum() - df\_eval["pred\_sbyNeed"].sum() + df\_eval["pred\_dafted"].iloc[len(df)-365:].sum()} FTE shifts were hold on standby and were not needed')  
 #print(f"Schritt-{i}: The utilization of the standby drivers is: {utilization}, and the number of standby drivers is exeeded in: {Overexceed} 5 over one year")  
  
cols=['interval\_width', 'Utilization\_ofStandby\_%', 'Standby\_exceeded\_%', 'FTE\_shifte\_needed', 'FTE\_standbyShifts\_withoutDuty\_needed']  
df\_result= pd.DataFrame(data, columns=cols)  
df\_result.set\_index("interval\_width", inplace=True)  
df\_result

Utilization\_ofStandby\_%

Standby\_exceeded\_%

FTE\_shifte\_needed

FTE\_standbyShifts\_withoutDuty\_needed

interval\_width

0.700

0.681972

0.454795

714844.426749

2016.720380

0.750

0.659824

0.432877

714844.426749

10340.131240

0.800

0.630474

0.358904

714844.426749

19844.493298

0.850

0.599755

0.306849

714844.426749

30698.786910

0.900

0.555172

0.230137

714844.426749

45843.909901

0.950

0.495772

0.117808

714844.426749

67919.806416

0.975

0.442822

0.065753

714844.426749

88130.147749

0.990

0.393779

0.030137

714844.426749

111117.289235

#df\_eval.to\_csv("PredictionResults.csv", sep=";", decimal=",")  
#df\_eval

#df\_result["Utilization\_ofStandby\_%"].plot(figsize=(12,8), grid =True)  
#df\_result["Standby\_exceeded\_%"].plot()

#fig, ax1 = plt.subplots()  
#ax2 = ax1.twinx()  
#ax1.plot(df\_result.index,df\_result["Utilization\_ofStandby\_%"], 'g-')  
#ax1.plot( df\_result.index,df\_result["Standby\_exceeded\_%"], 'g-')  
#ax2.plot( df\_result.index,df\_result["FTE\_standbyShifts\_withoutDuty\_needed"], 'r-')  
#  
#ax1.set\_xlabel('X data')  
#ax1.set\_ylabel('Y1 data', color='g')  
#ax2.set\_ylabel('Y2 data', color='b')  
#plt.show()

fig, ax1 = plt.subplots()  
ax1.plot(df\_result.index,df\_result["Utilization\_ofStandby\_%"], 'g-', label="Sby\_utilization")  
ax1.plot( df\_result.index,df\_result["Standby\_exceeded\_%"], 'b-', label="Sby\_exceeded")  
  
ax1.set\_xlabel('interval\_width')  
ax1.set\_ylabel('Percentage', color='black')  
plt.legend()  
plt.show()

png

png

#Create the Prohpet-Model,with an interval\_width of 97,5% (97,5% of the train values are in the predicted range)   
m = Prophet(interval\_width=0.975, daily\_seasonality=True,seasonality\_mode="multiplicative")  
# Train the Prophet-Model  
m.fit(train)  
# Create future datafame with xx days (periods & freq) as basis for prediction  
future = m.make\_future\_dataframe(periods=365,freq='D')  
# Make predictions for the created future dataframe  
forecast = m.predict(future)  
#plot the predictions   
forecast.set\_index(forecast["ds"], inplace=True)  
forecast["n\_RealCalls"]=df["calls"]  
forecast["drivers\_atwork\_old"]=df[["drivers\_atwork"]]  
  
fig = plt.figure(figsize=(50,12))  
ax = fig.add\_subplot(1, 1, 1)  
ax.plot(forecast["ds"], forecast["yhat"]/x\_Value, label='Medium prediction drivers[duty]', c="blue")  
ax.plot(forecast["ds"], forecast["yhat\_upper"]/x\_Value, label='Saftey prediction of drivers [duty + standby]', c="red")  
ax.plot(forecast["ds"],forecast["drivers\_atwork\_old"],label="Numbers of drivers at work in old model", c="black")  
ax.legend(loc='upper left')

<matplotlib.legend.Legend at 0x24e56f6d7c8>

png

png

Calculation of the root mean square error

pred=forecast["yhat"].iloc[len(forecast)-365:].values  
try:  
 rmse\_value=rmse(pred,test['y'].values)  
except:   
 rmse\_value=0  
print(f' The root mean square error is: {rmse\_value}')

The root mean square error is: 1012.0483883515906

Prophets Diagnostic Crossvalidation

# Initial 1 years training period  
initial = 1 \* 365.25  
initial = str(initial) + ' days'  
# Fold every year  
period = 1 \* 365  
period = str(period) + ' days'  
# Forecast 1 year into the future  
horizon = 365  
horizon = str(horizon) + ' days'  
  
df\_cv = cross\_validation(m, initial=initial, period=period, horizon = horizon)  
plot\_cross\_validation\_metric(df\_cv, metric='mape');

INFO:fbprophet:Making 1 forecasts with cutoffs between 2017-05-27 00:00:00 and 2017-05-27 00:00:00  
  
  
  
 0%| | 0/1 [00:00<?, ?it/s]  
  
  
C:\Users\niels\anaconda3\envs\tsa\_course\_env\_V3\lib\site-packages\fbprophet\plot.py:526: FutureWarning: casting timedelta64[ns] values to int64 with .astype(...) is deprecated and will raise in a future version. Use .view(...) instead.  
 x\_plt = df\_none['horizon'].astype('timedelta64[ns]').astype(np.int64) / float(dt\_conversions[i])  
C:\Users\niels\anaconda3\envs\tsa\_course\_env\_V3\lib\site-packages\fbprophet\plot.py:527: FutureWarning: casting timedelta64[ns] values to int64 with .astype(...) is deprecated and will raise in a future version. Use .view(...) instead.  
 x\_plt\_h = df\_h['horizon'].astype('timedelta64[ns]').astype(np.int64) / float(dt\_conversions[i])

png

png

## Deployment

# Import the needed libraries   
## Attention:For the installation of Prophet please follow the instructions written here: https://stackoverflow.com/questions/53178281/installing-fbprophet-python-on-windows-10  
import pandas as pd  
import numpy as np  
from fbprophet import Prophet  
import matplotlib.pyplot as plt

# Read in the given data and set the index column to the date column and parse the dates  
df=pd.read\_csv("../02\_data\_acquisition\_understanding/01\_data\_source/sickness\_table.csv", index\_col="date", parse\_dates=True )   
df["drivers\_atwork"]=df["n\_duty"]+df["sby\_need"]  
df.index.freq="D" #Set the frequence to Daily  
df.index

DatetimeIndex(['2016-04-01', '2016-04-02', '2016-04-03', '2016-04-04',  
 '2016-04-05', '2016-04-06', '2016-04-07', '2016-04-08',  
 '2016-04-09', '2016-04-10',  
 ...  
 '2019-05-18', '2019-05-19', '2019-05-20', '2019-05-21',  
 '2019-05-22', '2019-05-23', '2019-05-24', '2019-05-25',  
 '2019-05-26', '2019-05-27'],  
 dtype='datetime64[ns]', name='date', length=1152, freq='D')

# Prepare the Data for Prophet  
# Create the column "date"  
df["date"]=df.index  
# Create new DataFrames with the column "date"  
df\_new=pd.DataFrame(df["date"])  
# Create the column y="calls"  
df\_new["y"]=df["calls"]  
# Delete the index  
df\_new.reset\_index(drop=True, inplace=True)  
# Rename the columns to the Prophet specifics  
df\_new.rename(columns={"date": "ds", "y": "y"}, inplace=True)  
# Show the head of the DataFrame  
df\_new.head()

ds

y

0

2016-04-01

8154.0

1

2016-04-02

8526.0

2

2016-04-03

8088.0

3

2016-04-04

7044.0

4

2016-04-05

7236.0

#Define the number of days the model should predict the workforce plan  
numberofdaysforprediction=31  
  
#Create the Prohpet-Model,with an interval\_width of 95% (95% of the train values are in the predicted range)   
m = Prophet(interval\_width=0.975, daily\_seasonality=True,seasonality\_mode="multiplicative")  
# Train the Prophet-Model  
m.fit(df\_new)  
# Create future datafame with xx days (periods & freq) as basis for prediction  
future = m.make\_future\_dataframe(periods=numberofdaysforprediction,freq='D')  
# Make predictions for the created future dataframe  
forecast = m.predict(future)  
#plot the predictions   
m.plot(forecast, figsize=(20, 6));  
forecast.set\_index(forecast["ds"], inplace=True)

png

png

# Generating the prdiction of drivers on duty and drivers on standby for the next 31 days  
x\_Value=4.82  
df\_predresult=pd.DataFrame(forecast["ds"].iloc[len(forecast)-numberofdaysforprediction:])  
df\_predresult["pred\_n\_duty"]=forecast["yhat"].iloc[len(forecast)-numberofdaysforprediction:]/x\_Value  
df\_predresult["pred\_n\_sby"]=forecast["yhat\_upper"].iloc[len(forecast)-numberofdaysforprediction:]/x\_Value-forecast["yhat"].iloc[len(forecast)-numberofdaysforprediction:]/x\_Value  
  
#Round the values and cast to intger:   
df\_predresult["pred\_n\_duty"]=df\_predresult["pred\_n\_duty"].apply(lambda x: np.around(x))  
df\_predresult["pred\_n\_duty"]=df\_predresult["pred\_n\_duty"].apply(lambda x: int(x))  
df\_predresult["pred\_n\_sby"]=df\_predresult["pred\_n\_sby"].apply(lambda x: np.around(x))  
df\_predresult["pred\_n\_sby"]=df\_predresult["pred\_n\_sby"].apply(lambda x: int(x))  
#drop the column "ds" since the index is already the date   
df\_predresult.drop("ds", inplace=True, axis=1)

# Show the predictions for the next 31 days  
df\_predresult

pred\_n\_duty

pred\_n\_sby

ds

2019-05-28

2071

437

2019-05-29

2052

418

2019-05-30

2025

445

2019-05-31

2007

440

2019-06-01

1975

455

2019-06-02

1880

438

2019-06-03

2104

439

2019-06-04

2109

439

2019-06-05

2091

425

2019-06-06

2063

444

2019-06-07

2045

410

2019-06-08

2010

438

2019-06-09

1913

428

2019-06-10

2134

434

2019-06-11

2136

426

2019-06-12

2113

426

2019-06-13

2081

444

2019-06-14

2059

419

2019-06-15

2020

499

2019-06-16

1918

443

2019-06-17

2135

439

2019-06-18

2134

446

2019-06-19

2108

421

2019-06-20

2073

429

2019-06-21

2048

430

2019-06-22

2007

426

2019-06-23

1904

419

2019-06-24

2121

442

2019-06-25

2120

451

2019-06-26

2095

443

2019-06-27

2061

436

# Create an extract as csv --> the separator is a Semikolon and the decimals is a comma  
df\_predresult.to\_csv("Workforceplaning.csv", decimal=",",sep=";")