AUTOMATION OF STANDBY DUTY PLANNING FOR RESCUE DRIVERS VIA A FORECASTING

CASE STUDY

Author: Jelson Lino
Tutor: Dr. Sahar Qaadan

Matriculation Number: 92120090

DLMDSME01 – Model Engineering
Master of Science in Data Science
IU International University of Applied Sciences
29 February 2024

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1. 1. Business Understanding

1. 1. 1. Business Problem

The current standby-duty plan for Berlin's red-cross rescue service struggles with inefficiencies, leading to situations where there are not enough standby drivers or too many standby drivers are kept on hold. The HR planning department wants to improve the current planning logic by incorporating predictive models to estimate the daily number of standby rescue drivers more accurately.

1. 1. 2. Project Objectives

This project aims to develop a predictive system to estimate the number of standby rescue drivers required, improve the efficiency of the standby-duty plan by increasing the percentage of standby drivers being activated compared to the current approach of keeping 90 drivers on hold, reduce situations where there are not enough standby drivers or excess standby drivers, ensure that the new planning model can incorporate seasonal patterns and other factors affecting the number of required standby drivers, and meet the deadline for finishing the duty plan on the 15th of the current month for the upcoming month.

1. 1. 3. Success Criteria

The success criteria of this project include increasing the percentage of standby drivers being activated compared to the current approach, reducing the number of situations where there are not enough standby drivers or excess standby drivers, developing a predictive system with a reasonable level of accuracy in estimating the daily number of standby drivers required, and meeting the deadline for finishing the duty plan on the 15th of the current month for the upcoming month consistently.

Project Plan

The execution of this project followed the following steps:

- Understand the current standby-duty plan and its shortcomings.
- Understand and analyze the data provided.
- Develop predictive models to estimate the daily number of standby drivers required.
- Provide recommendations on the implementation of the selected predictive system into the planning process.

1. 1. 4. Resources

A Windows laptop with RAM and processor of 8.00 GB and Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz, respectively, was used to run Python and its libraries and frameworks to process the data and train and evaluate the machine learning algorithms. In addition, Brisqi was used as a project management tool.

2. DATA UNDERSTANDING

The data was stored in a csv file. Pandas was used to load and understand the data. There were eight columns. None of the columns had any nun values nor what could be considered outliers. One of the columns had object-type data stored in it, while the remaining columns stored numerical data. The following description defines the nature of each column in the dataset.

• date: entry date

• n_sick: number of drivers called sick on duty

• calls: number of emergency calls

• 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

To improve readability and facilitate analysis, the columns were renamed based on the description given above. There was an Unnamed column, which was simply an index column and not relevant to data modeling. Therefore, it was removed from the data. The date column stores object data. However, the information stored in it is date data. Therefore, the format of this column was changed from object to datetime.

3. DATA PREPARATION

The seasonality claimed by the HR department was checked by plotting the data.

Figure 1: Activated Standby Drivers

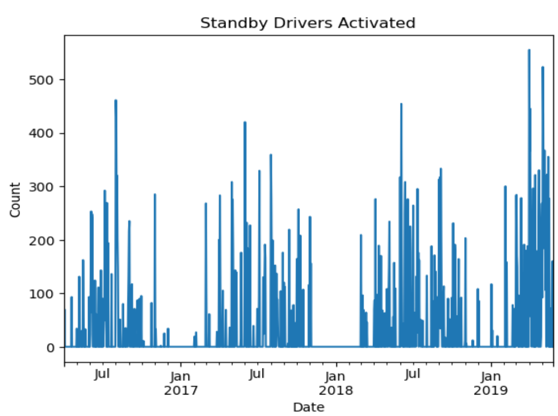


Figure 1 indicates a seasonal pattern in the number of activated standby drivers. The peak season is in the Summer, while the Winter registers the lowest numbers of activated standby drivers.

The data also suggest that there are days in which no standby driver is activated. To further understand the data, Figure 2 shows the relationship between the target variable and all the independent variables.

Figure 2: Relationship Between the Target Feature and Other Variables

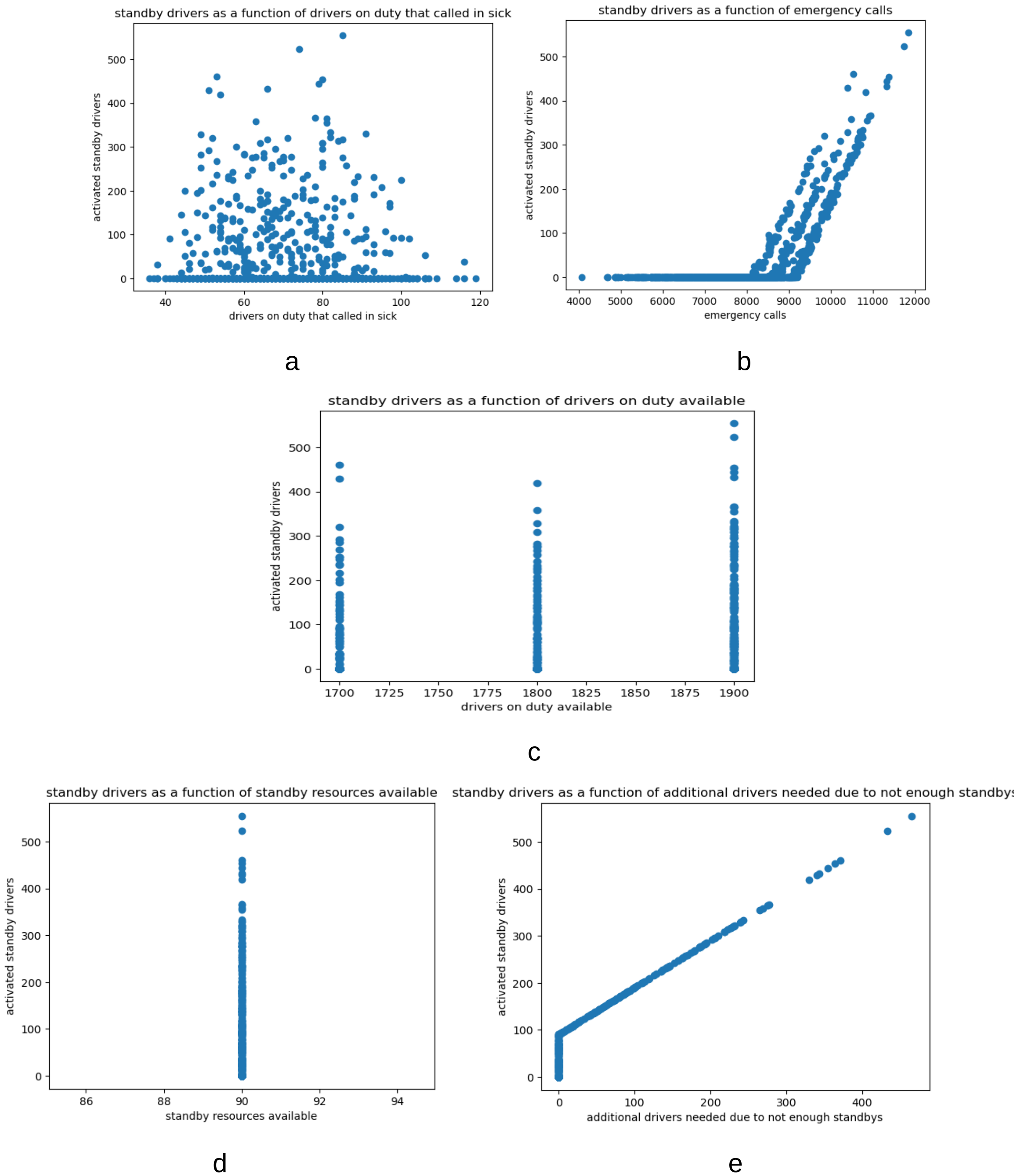


Figure 2a does not show a clear mathematical relationship between the target variable and sick drivers. Nevertheless, this variable still has an impact on the target variable. Figure 2b shows that there is a threshold after which the target variable and emergency calls are linearly related. The three lines that appear after the threshold are due to the three unique values found in the number of available drivers, as shown in Figure 2c. Figure 2c suggests that the number of drivers available increased twice since 2016. Figure 2d shows that the number of standby resources available is always 90. Thus, this column should not be used in the training of the machine learning models. The constant value of this column is the reason why there is a threshold of 90 in Figure 2e. Figure 2e shows a linear relationship between the target variable and the number of additional drivers needed due to insufficient standby drivers after the threshold of 90 activated standby drivers is reached. This is because additional drivers are only needed when the 90 standby drivers cannot respond to all the calls that the on-duty drivers are unable to respond to. The following figures show the seasonal pattern of each feature.

Figure 3: Features Over Time

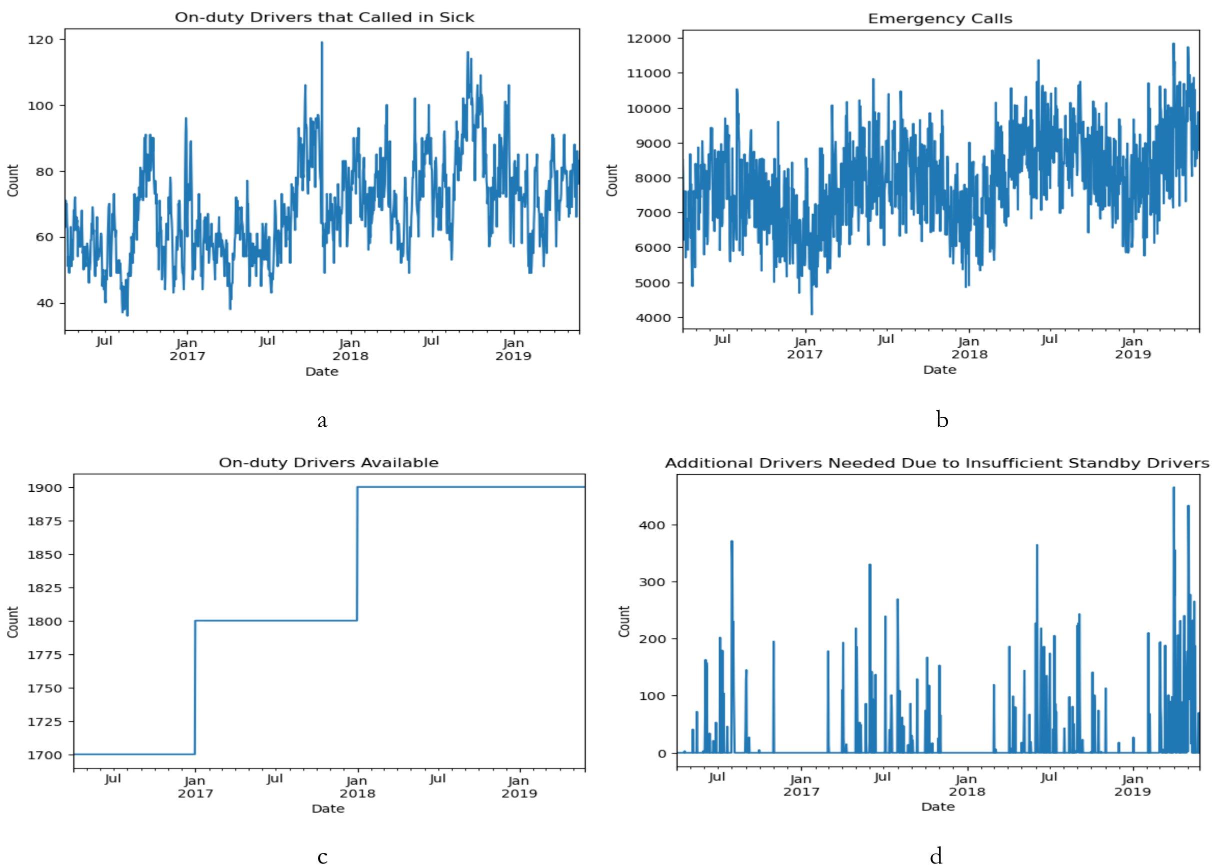
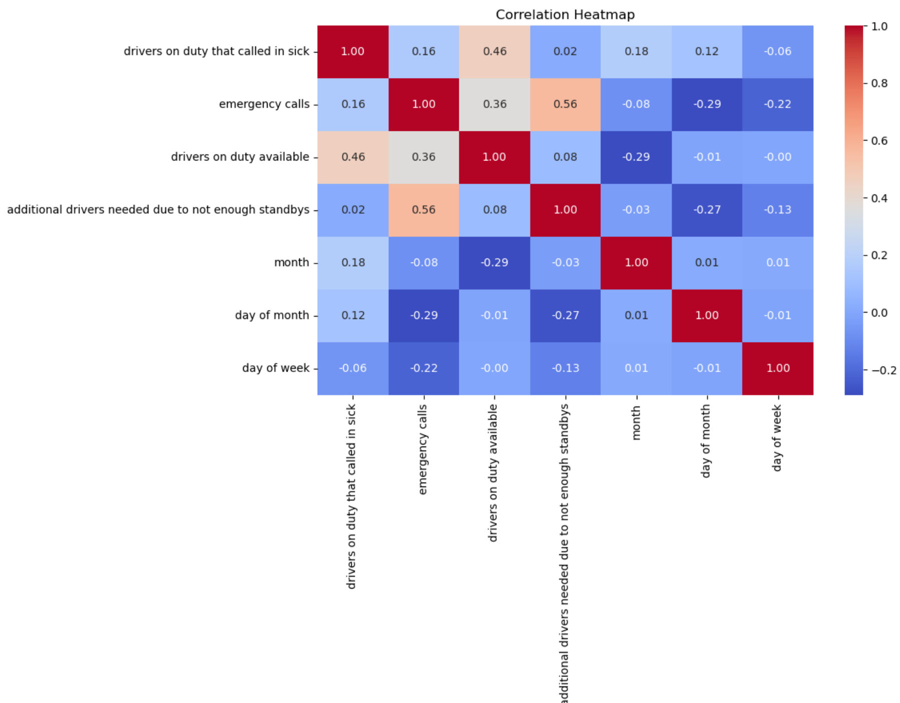


Figure 3 shows a clear seasonal pattern on most variables. The fact that, regardless of the number of on-duty drivers calling in sick, the number of on-duty drivers available is almost always constant raises concern about the quality and veracity of the data. However, we will assume that when n on-duty drivers call in sick, n standby drivers are put on duty to replace those who called in sick, and n non-standby drivers are put on standby to replace those who are now on duty.

A date is a data point that does not repeat itself; it only appears once in the dataset. Because we are interested in building a model that can grasp the seasonal aspect of the data, three more variables were created from the date column; they are month, day of month, and day of week. The values of these variables can appear more than once and can be extracted from any given month. Thus, given the date whose activated standby drivers one wants to predict, the values of these columns are extracted and fed into the predictive system. Although the data plots shown above show the nature of each variable, to further understand the relationship between variables, a correlation heatmap is shown below.

Figure 4: Correlation Heatmap



There is a moderate correlation between emergency calls and additional drivers needed due to not enough standby drivers. This correlation is expected because the higher the volume of calls, the higher the number of drivers activated and the higher the number of additional drivers needed due to not enough standby drivers. Because the variable standby resources available is always 90, the higher the number of calls, the higher the number of additional standby drivers needed (once the threshold of calls that the on-duty and the 90 standby drives can bear is reached). After the threshold of 90, these two variables are expected to be directly proportional. Nevertheless, the objective is to predict the number of standby drivers activated so that the standby drivers are set based on the prediction and ideally the number of additional drivers is kept at zero.

In addition, this variable (additional drivers needed due to not enough standby drivers) only exists because the number of standby drivers available is always 90. If we set the number of standby drivers available based on the prediction for activated standby drivers, then we will be able to keep additional drivers at zero (or close to zero). Therefore, the variable additional drivers needed due to not enough standby drivers was dropped from the analysis.

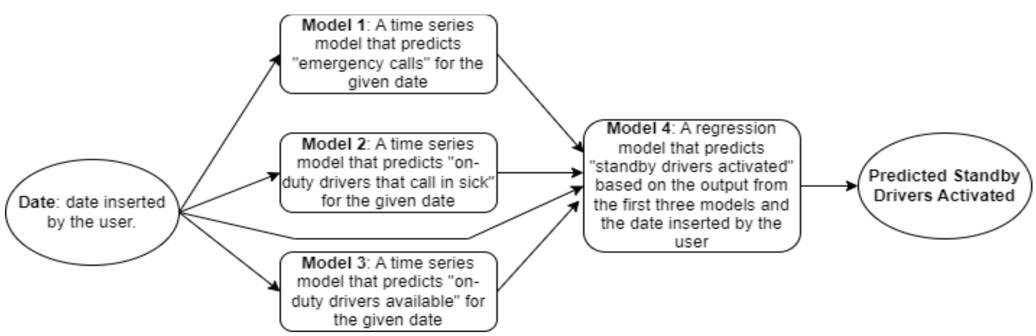
Based on the understanding we have of the data, the number of additional drivers needed due to not enough standbys + the number of standby resources available is equal to the number of standbys activated, when the number of standbys activated is equal to or greater than 90. When the number of standby drivers activated is less than 90 (or the number of standby resources available), the number of additional drivers needed due to not enough standby drivers is zero. Because of this, added to the fact that it is a constant, the column standby resources available was dropped from the analysis.

4. MODELING

As demonstrated, there are temporal patterns in the data and there are independent features that do have some impact on the target feature. For example, the number of calls on a given day does have an impact on the number of standby drivers activated. Therefore, to predict the number of standby drivers activated, we need to first predict all the variables that have an impact on our target feature (standby drivers activated).

Based on Figure 13, to predict the target feature, the values of six independent variables are required (considering that additional drivers needed due to insufficient standby drivers is dropped). Three of those variables are obtained from the current date, that is the date whose standby drivers activated is being predicted. The variables obtained from the date are month, day of month, and day of week. The variables on-duty drivers that called in sick, emergency calls, and on-duty drivers that are available have temporal patterns, as shown in Figure 3. In addition, due to the nature of the variable being predicted (a numerical continuous variable), a supervised regression model is appropriate for this project. The three variables whose values cannot be extracted from date are predicted separately using different time series models. This means that the entire prediction system works based on four supervised regression models. Figure 5 illustrates how the system works.

Figure 5: Standby Drivers Activated Prediction System



Three time-series models (models 1, 2, and 3) predict on-duty drivers that called in sick, emergency calls, and on-duty drivers that are available. The output from these models and the intended month, day of the week, and day of the month are fed into the main model (model 4) that predicts the number of standby drivers activated. The available data can be used to run the four models of the entire system. In this report, model 4 is presented first. However, during operation, the order presented in Figure 5 is kept.

4. 1. Model 4

As shown in Figure 5, model 4 is the main model of the system. In this section, the training and evaluation of model 4 are presented. Because the target variable (standby drivers activated) is a numerical continuous variable, regression is the appropriate machine learning method to be used in this project. In addition, the target variable shows a temporal pattern. Thus, a multivariate time series model could be used. However, due to extremely poor performance, multivariate time series models were put aside.

4. 1. 2. Splitting Data Vertically and Horizontally

The data was split into the target variable and independent variables. All variables, except standby drivers activated, were selected as independent variables, while the variable standby drivers activated was selected as the target feature. Furthermore, the data was also split into training and test sets, with 80% of the data being used for training and the remaining being used for testing.

A baseline model was developed, as shown below, against which all subsequent models were compared. Seven supervised regression models were trained using the same data. The objective was to evaluate various models and select the best one. The metrics used to evaluate the models were mean-squared error (MSE) and r-squared. MSE represents the average squared difference between the predicted values and the actual values. Lower MSE indicates better predictive performance, with values closer to zero suggesting a better fit. R2, also known as the coefficient of determination, measures the proportion of the variance in the target variable that is predictable from the independent variables. It ranges from 0 to 1, where 1 indicates a perfect fit and 0 indicates no linear relationship between the variables. In this project, any model with values worse than those of the baseline model was considered unsuitable. Models performing better than the baseline were compared against each other. Figure 15 shows the models and their metrics.

Figure 6a: Model Evaluation - MSE

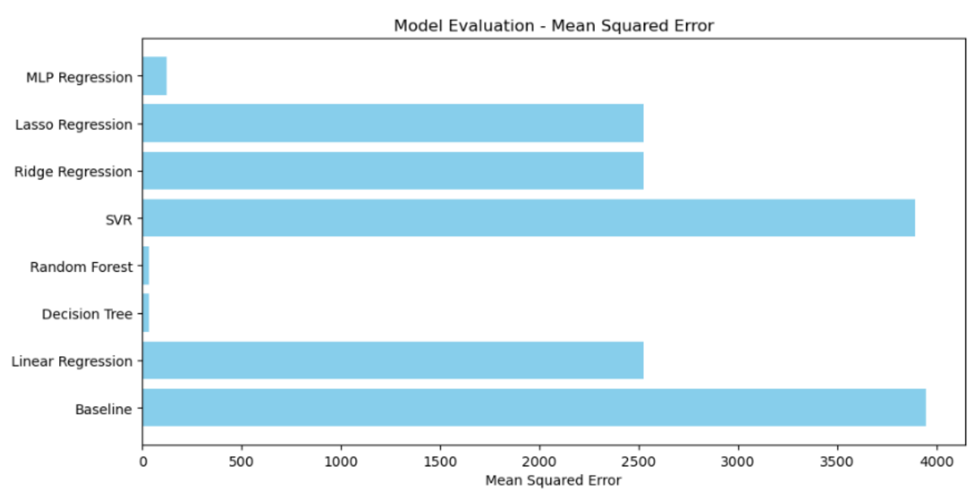
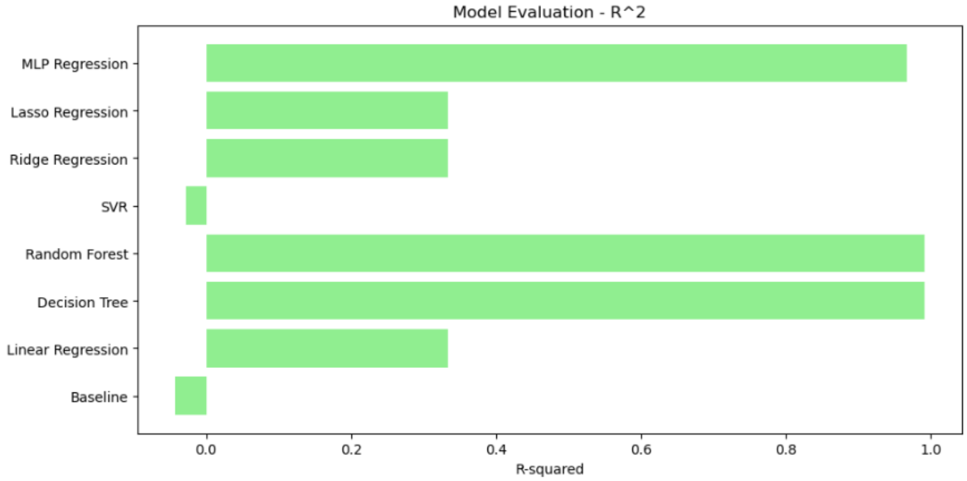


Figure 6b: Model Evaluation - R2

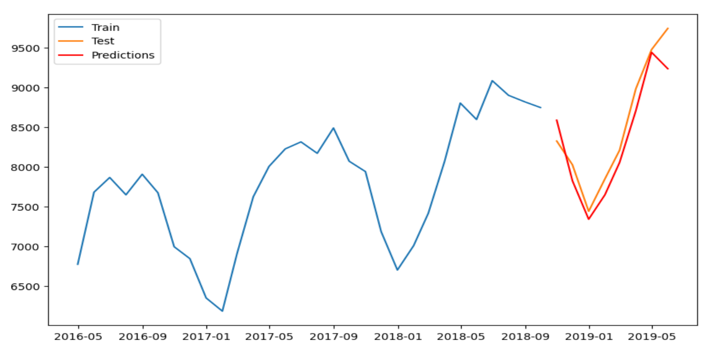


Figures 6a and 6b show that the decision tree model is the best-performing model, as it has the best value in both metrics, and is closely followed by the random forest model. SVR (support vector regressor) is the worst-performing model. Therefore, the decision tree model is used as model 4 of Figure 5.

4. 2. Model 1

The data on the column emergency calls was split into 80% and 20% for training and testing, respectively, while the Date column was used as the index. Then, a SARIMAX model was trained. The model achieved an MSE of 254.5966. Figure 16 shows the performance of this model compared with the actual values.

Figure 7: Emergency Calls



4. 3. Models 2 & 3

Models 2 and 3 were built using a convolutional neural network designed based on Keras with a TensorFlow backend. 916 data points were used as training data, while 115 and 114 data points were used as validation and testing data, respectively. In both models, 7 previous values are used to predict the next value. Models 2 and 3 had MSE values of 41.30 and 0.641, respectively. Figures 8a and 8b show the performance of models 2 and 3, respectively.

Figure 8a: Model 2 - Drivers That Call in Sick

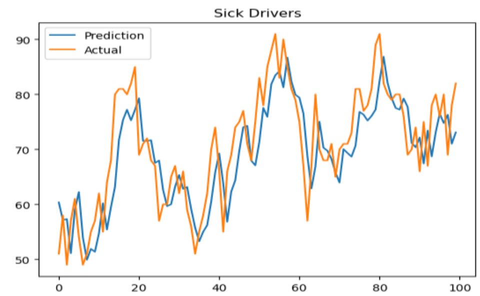
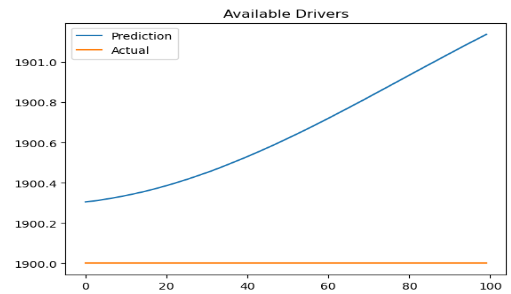


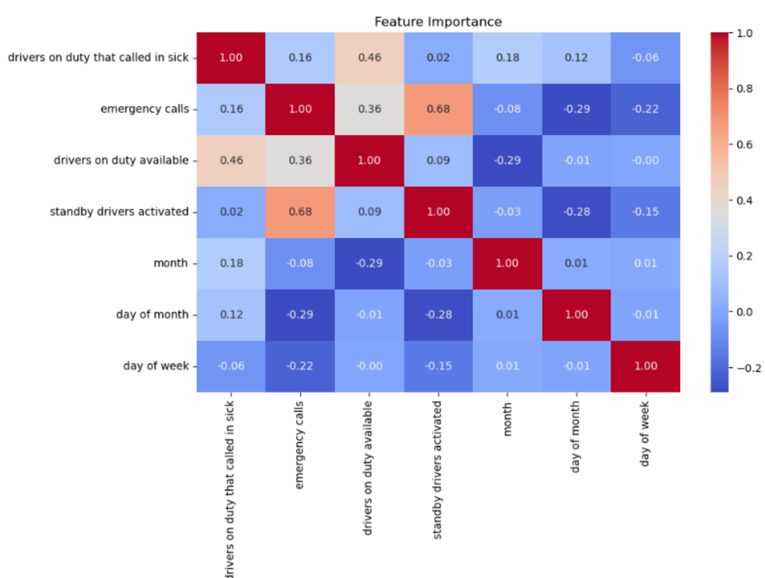
Figure 8b: Model 3 - Drivers Available



5. EVALUATION

MSE indicates the average squared difference between the predicted and actual values. A lower MSE suggests better predictive performance. All seven machine learning models considered for model 4 have an MSE lower than that of the baseline model. However, the lowest MSE is that of the decision tree model. This means that the decision tree model has the smallest deviation from the true values. The MSE of the decision tree model is 32.5; this implies that the mean absolute deviation from the true values is approximately 5.7. Considering that the values of the variable that model 4 predicts range from zero to more than 500, an error of +/- 5.7 is reasonably acceptable. In practice, it means that when the model predicts, for example, 500 standby drivers activated, there could be an error of at least 5.7, making the true value between 494 and 506. However, as part of the data fed into model 4 is already a prediction from three models, the difference between the predicted and true values of model 4 can slightly increase because the output from the three models could already be slightly deviated from the true values. Models 1, 2, and 3 have MSE values of 254.49, 41.30, and 0.6411, respectively. As shown in Figure 9, the variable emergency calls is the most important variable in the prediction of standby drivers activated. This variable is predicted by model 1 and has an MSE of 254.59, which is acceptable, given that the values predicted can be higher than 9500. An MSE of 254.59 implies that the mean absolute difference between the predicted and true values is only 15.96.

Figure 9: Feature Importance



The results indicate that this system can help the HR department plan more accurately by minimizing situations where too many drivers are kept on hold. In addition, the number of additional drivers needed due to insufficient standby drivers will also be minimized; however, there is no guarantee that this number will always be zero.

6. DEPLOYMENT

As shown in Figure 5, the entire system uses four models. The system can be used by the HR team as a web app deployed on Azure or any other suitable cloud service provider. The web app can be used in two manners. The HR team can access the web app via URL (http triggered) and make predictions as often as necessary or the program can be set to automatically run (time triggered) and produce a report with the predictions on the 14th day of every month.

The first approach allows for experimentation, and, in case of any unexpected change or issue, the HR team can rerun the program as many times as necessary without involving the analytics/IT team. In the second approach, the process is completely automated, which means that the HR team does not interact directly with the program as the team simply receives an automatic report; this process allows for a completely automated scheduling service. However, in case of any unexpected change or issue, the analytics/IT team will have to respond to the change/issue.

Whatever the option used, the model's performance will have to be monitored regularly and the system should be retrained at least once every two months (because in two months enough new data is produced to retrain the system). However, if recent predictions deviate significantly from the true values, the model must be retrained immediately, regardless of whether or not two months have already passed from the last retraining.

7. CONCLUSION

In this project, a system was developed that predicts the number of standby drivers activated and automates and optimizes the standby-duty plan of the Red Cross in Berlin by increasing the percentage of standby drivers being activated, thus, reducing situations where there are too many or too few standby drivers. The system, which is composed of four supervised regression algorithms (one SARIMAX, two CNNs, and one Decision Tree), incorporates seasonal patterns. The system can be deployed on Azure, or any other cloud service provider, as a web app to allow the HR team to open it from their browser. Further studies can be conducted, once additional data is available, to improve the mean squared error of the SARIMAX model and the first CNN model. The predicted and actual values of every month should be recorded and saved for performance monitoring and retraining of the entire system.

8. APPENDIX 1: GitHub Repository Structure

The GitHub repository for this project should contain the following items:

1. The main files: those are the files that contain the python program. For example, app.py, models.py, and requirements.txt. The file app.py contains models 1, 2, 3, and 4, and the lines of code that instruct the web app how to use the models. The file models.py is used to train the models and process the data used to train the models.
2. Configuration files: Those are the files required for configuring the deployment environment. The specific files depend on the cloud service provider on which the web app is deployed. In the case of Azure, the configuration files include function.json, host.json, and local.settings.json.
3. Documentation file: this is a README.md file that describes the project and the process of setting it up locally and on the selected cloud service provider.
4. CI/CD: any file necessary for automated deployment to facilitate retraining and update of the system.

9. APPENDIX 2: Source Code

DATA UNDERSTANDING, CLEANING, AND PROCESSING

```
# Library Import
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
import copy
import seaborn as sns
from tensorflow.keras.models import load_model
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.metrics import RootMeanSquaredError
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error
from sklearn.neural_network import MLPRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
```

```python
from sklearn.svm import SVR
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from statsmodels.tsa.statespace.sarimax import SARIMAX
import warnings
warnings.filterwarnings('ignore')
```

```
WARNING:tensorflow:From C:\Users\JC\anaconda3\Lib\site-packages\keras\src\losses.p
y:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use
tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

```python
# import data
df = pd.read_csv("sickness_table.csv")
df.head()
```

	Unnamed: 0	date	n_sick	calls	n_duty	n_sby	sby_need	dafted
0	0	2016-04-01	73	8154.0	1700	90	4.0	0.0
1	1	2016-04-02	64	8526.0	1700	90	70.0	0.0
2	2	2016-04-03	68	8088.0	1700	90	0.0	0.0
3	3	2016-04-04	71	7044.0	1700	90	0.0	0.0
4	4	2016-04-05	63	7236.0	1700	90	0.0	0.0

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1152 entries, 0 to 1151
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  1152 non-null   int64
 1   date        1152 non-null   object
 2   n_sick      1152 non-null   int64
 3   calls       1152 non-null   float64
 4   n_duty      1152 non-null   int64
 5   n_sby       1152 non-null   int64
 6   sby_need    1152 non-null   float64
 7   dafted      1152 non-null   float64
dtypes: float64(3), int64(4), object(1)
memory usage: 72.1+ KB
```

Column Description:

- date: entry date

- n_sick: number of drivers called sick on duty
- calls: number of emergency calls
- 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

```python
# change column names to improve readability
df.rename(
    columns={"date": "Date",
             "n_sick": "drivers on duty that called in sick",
             "calls": "emergency calls",
             "n_duty": "drivers on duty available",
             "n_sby": "standby resources available",
             "sby_need": "standby drivers activated",
             "dafted": "additional drivers needed due to not enough standbys"},
    inplace=True,
)
```

The project aims at minimizing dates with not enough standby drivers while having only standbys that will be used. Thus, the target variable for our model shall be number_of_standbys_activated

```python
df.head()
```

	Unnamed: 0	Date	drivers on duty that called in sick	emergency calls	drivers on duty available	standby resources available	standby drivers activated	additional drivers needed due to not enough standbys
0	0	2016-04-01	73	8154.0	1700	90	4.0	0.0
1	1	2016-04-02	64	8526.0	1700	90	70.0	0.0
2	2	2016-04-03	68	8088.0	1700	90	0.0	0.0
3	3	2016-04-04	71	7044.0	1700	90	0.0	0.0

	Unnamed: 0	Date	drivers on duty that called in sick	emergency calls	drivers on duty available	standby resources available	standby drivers activated	additional drivers needed due to not enough standbys
4	4	2016-04-05	63	7236.0	1700	90	0.0	0.0

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1152 entries, 0 to 1151
Data columns (total 8 columns):
 #   Column                                                Non-Null Count  Dtype
---  ------                                                --------------  -----
 0   Unnamed: 0                                            1152 non-null   int64
 1   Date                                                  1152 non-null   object
 2   drivers on duty that called in sick                   1152 non-null   int64
 3   emergency calls                                       1152 non-null   float64
 4   drivers on duty available                             1152 non-null   int64
 5   standby resources available                           1152 non-null   int64
 6   standby drivers activated                             1152 non-null   float64
 7   additional drivers needed due to not enough standbys  1152 non-null   float64
dtypes: float64(3), int64(4), object(1)
memory usage: 72.1+ KB
```

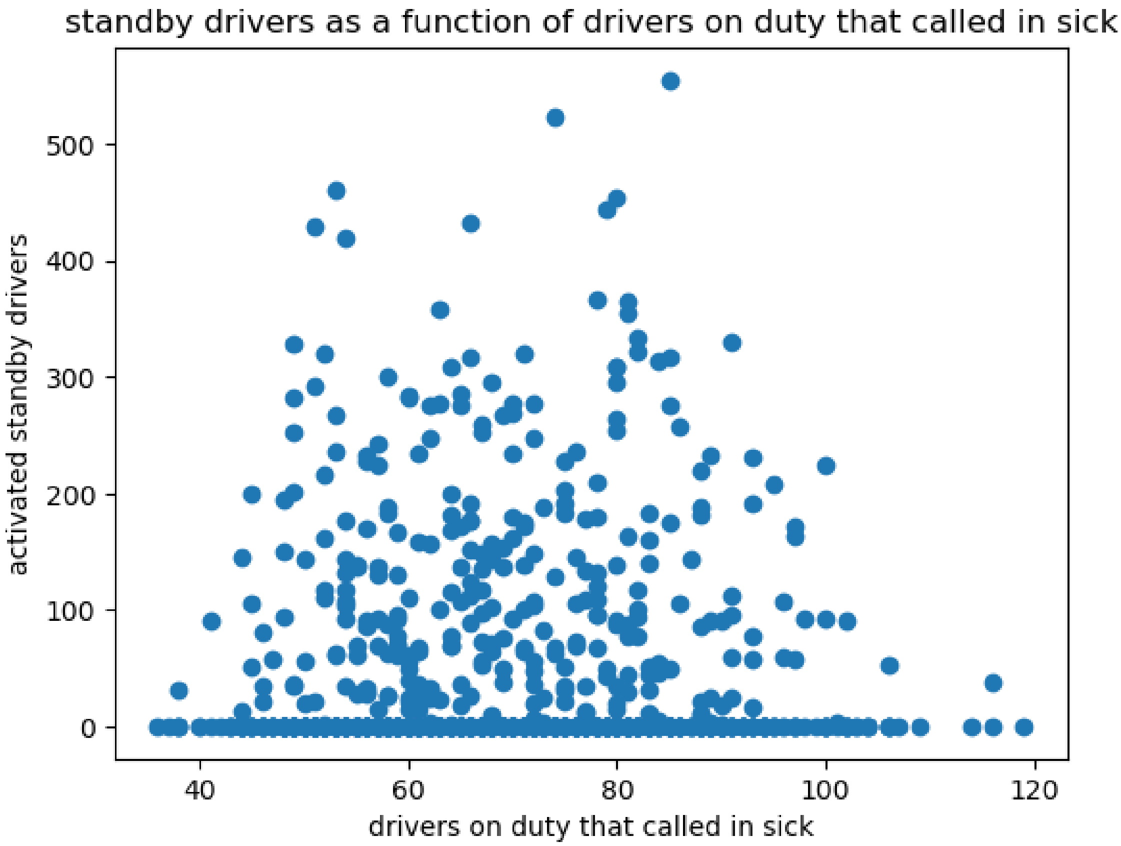
```
# drop Unnamed column
df.drop(df.filter(regex="Unnamed"),axis=1, inplace=True)
```

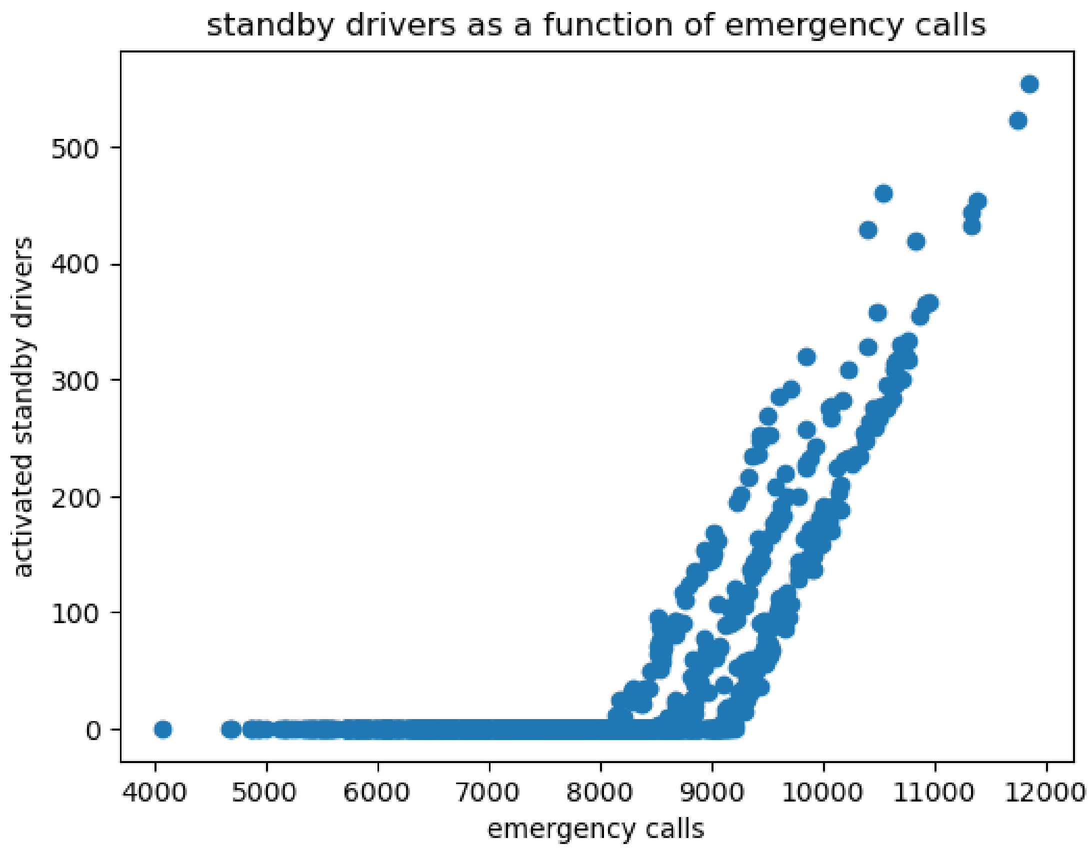
```
df.info()
```

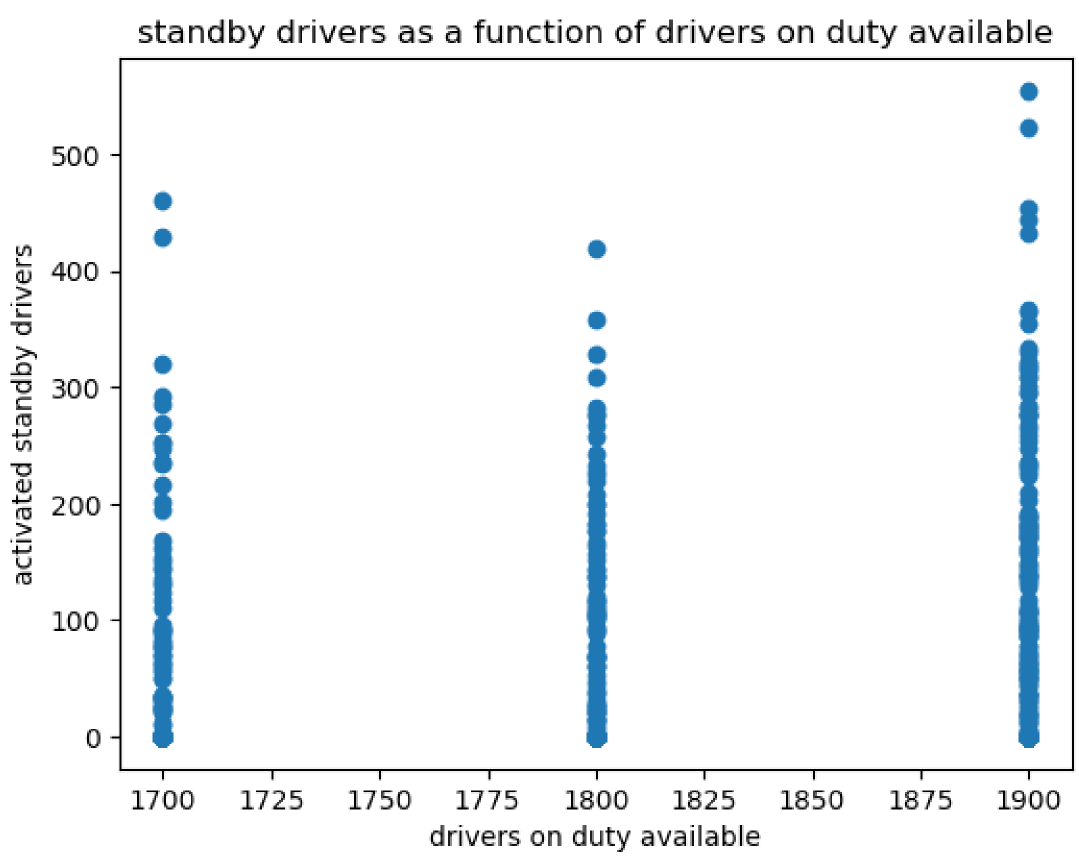
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1152 entries, 0 to 1151
Data columns (total 7 columns):
 #   Column                                                Non-Null Count  Dtype
---  ------                                                --------------  -----
 0   Date                                                  1152 non-null   object
 1   drivers on duty that called in sick                   1152 non-null   int64
 2   emergency calls                                       1152 non-null   float64
 3   drivers on duty available                             1152 non-null   int64
 4   standby resources available                           1152 non-null   int64
 5   standby drivers activated                             1152 non-null   float64
 6   additional drivers needed due to not enough standbys  1152 non-null   float64
dtypes: float64(3), int64(3), object(1)
memory usage: 63.1+ KB
```

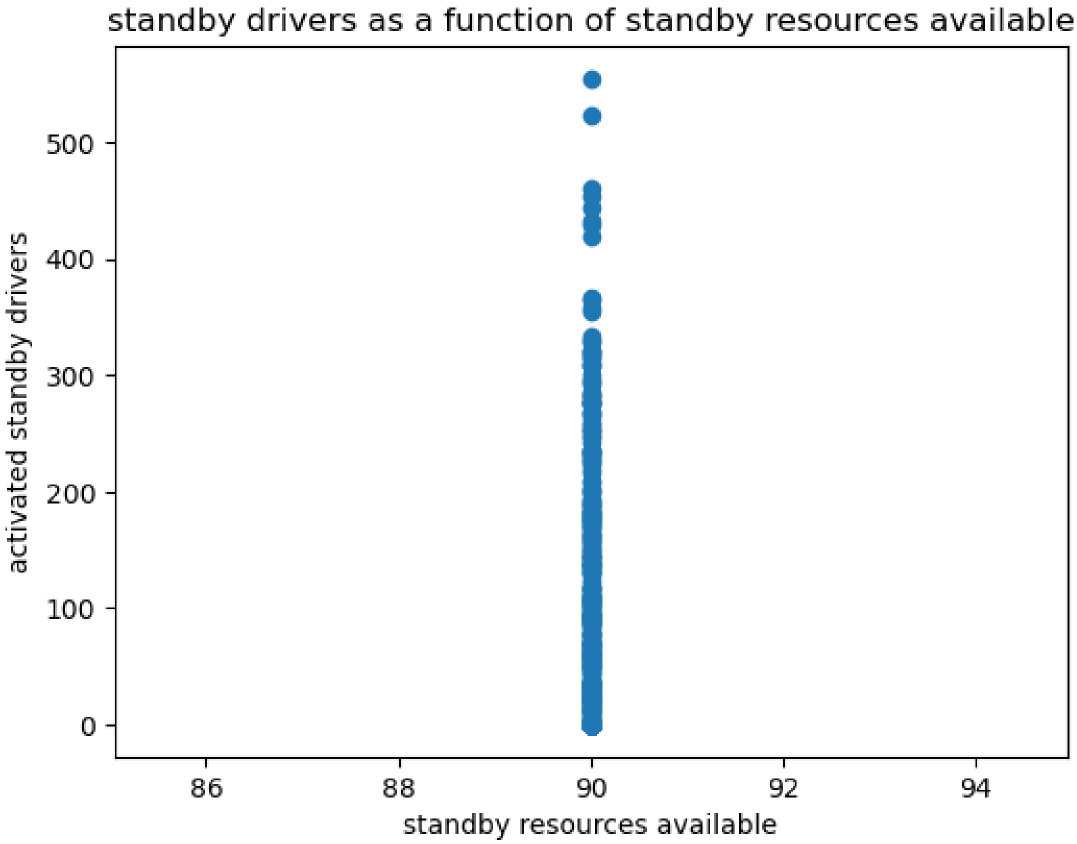
There are no null values in the data.

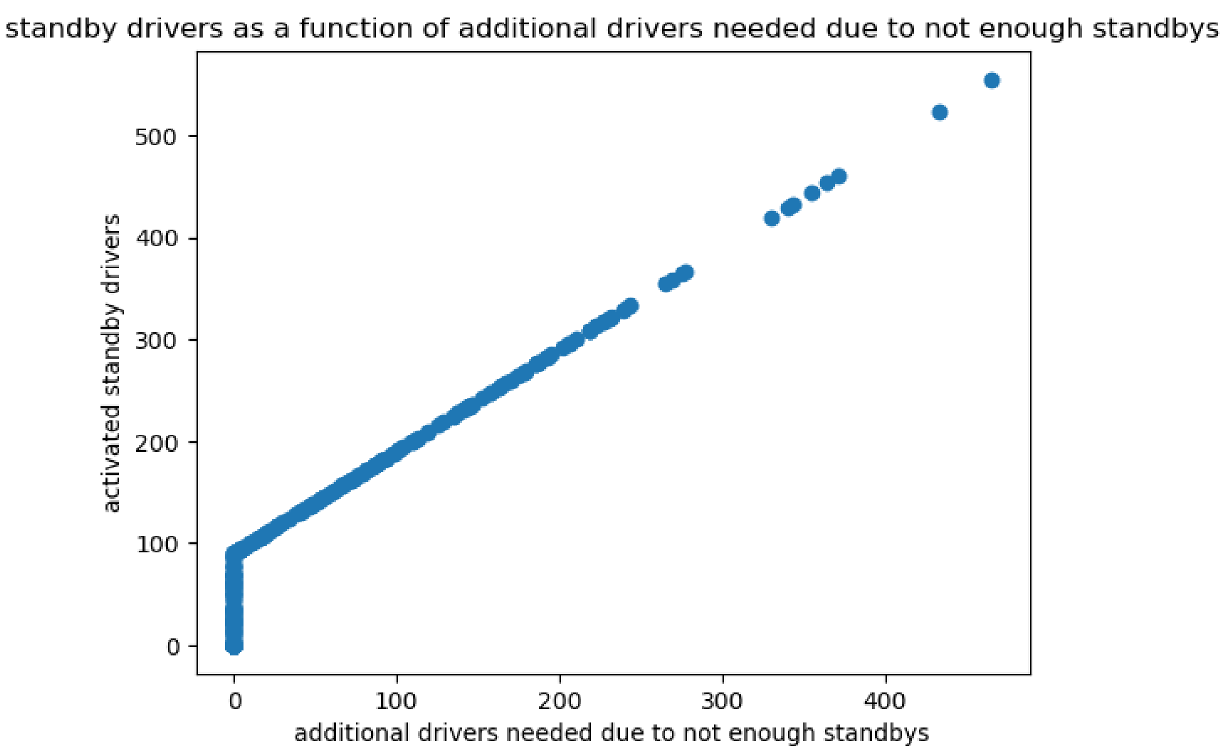
```python
# Select numerical columns
df_numerical_columns = df.drop(columns=['Date','standby drivers activated'])

# Relationship between the target variables and the numerical variables

for label in df_numerical_columns.columns:
  plt.scatter(df_numerical_columns[label], df["standby drivers activated"])
  plt.title(f'standby drivers as a function of {label}')
  plt.ylabel("activated standby drivers")
  plt.xlabel(label)
  plt.show()
```




standby drivers as a function of emergency calls
activated standby drivers
emergency calls
0
100
200
300
400
500
4000
5000
6000
7000
8000
9000
10000
11000
12000

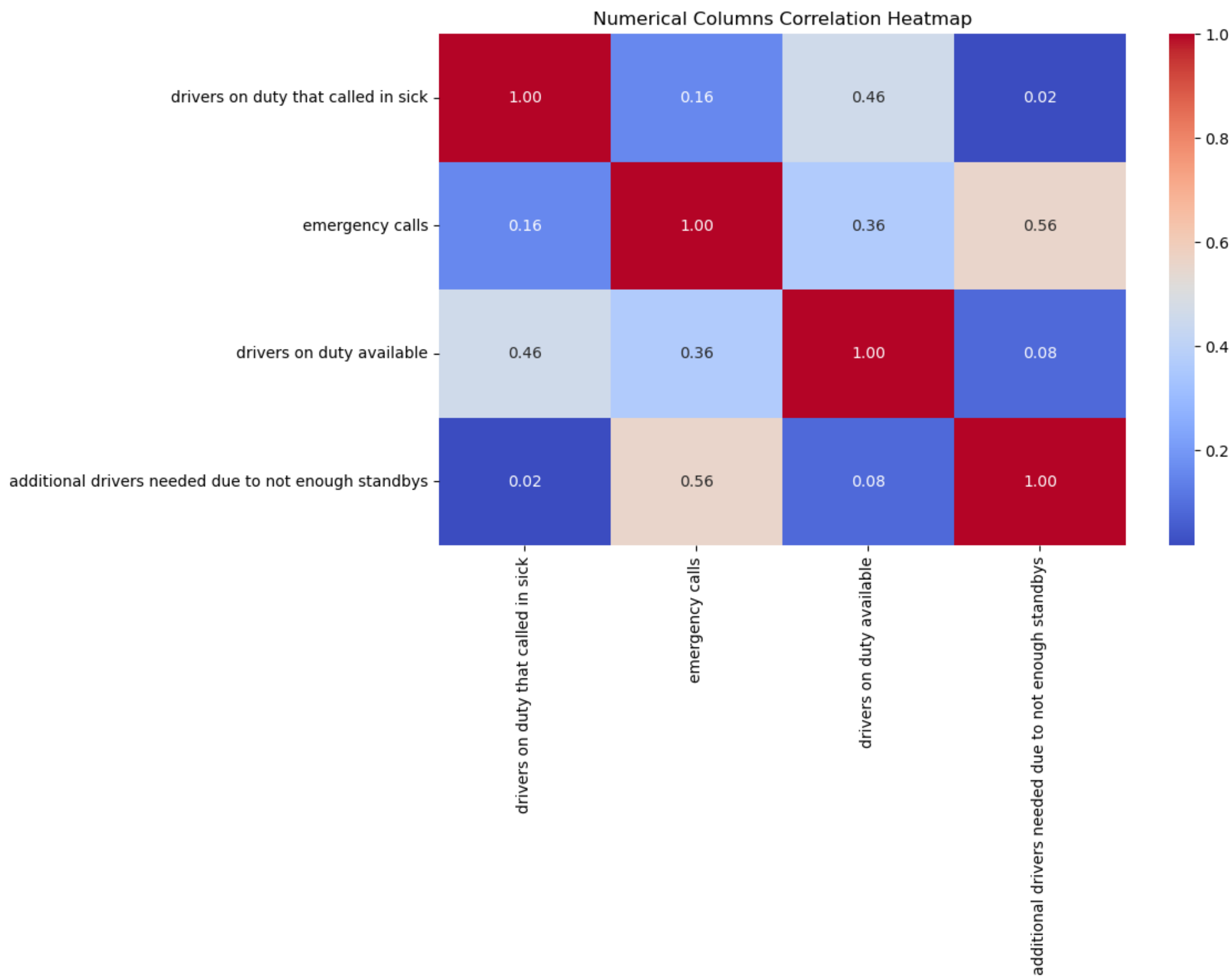

standby drivers as a function of drivers on duty available
activated standby drivers
drivers on duty available
0
100
200
300
400
500
1700
1725
1750
1775
1800
1825
1850
1875
1900






```
# Calculate correlation matrix
correlation_matrix = df_numerical_columns.drop(columns=['standby resources availab
le']).corr()
```

```
# Create heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Numerical Columns Correlation Heatmap')
plt.show()
```



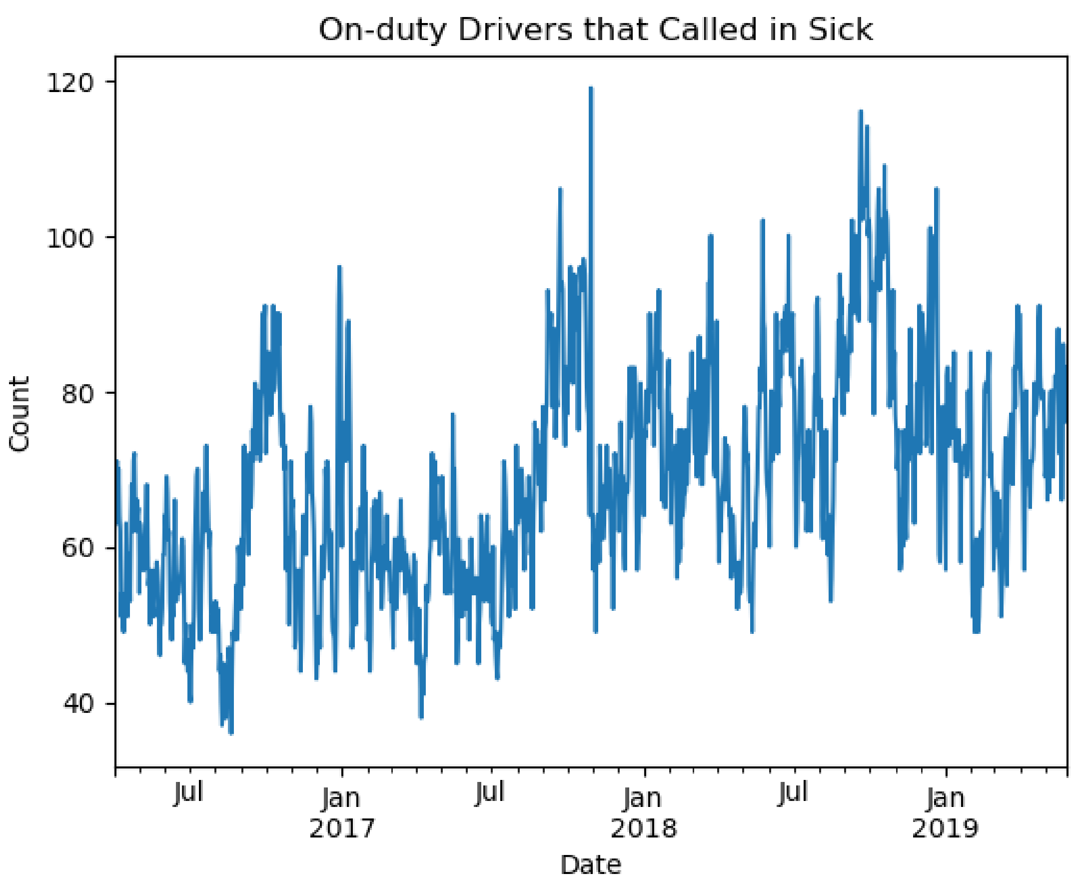
There is a moderate correlation between the number of emergency calls and the number of additional drivers needed due to not enough standby drivers. This correlation is expected because the higher the volume of calls, the higher the number of drivers needed and the higher the number of additional drivers needed.

```
# Convert 'Date' from string to date
df['Date'] = pd.to_datetime(df['Date'])

# set date as index
df.index = pd.to_datetime(df['Date'])
df.drop(df.filter(regex="Date"),axis=1, inplace=True)

df.info()

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1152 entries, 2016-04-01 to 2019-05-27
```

```
Data columns (total 6 columns):
 #   Column                                          Non-Null Count  Dtype
---  ------                                          --------------  -----
 0   drivers on duty that called in sick             1152 non-null   int64
 1   emergency calls                                 1152 non-null   float64
 2   drivers on duty available                       1152 non-null   int64
 3   standby resources available                     1152 non-null   int64
 4   standby drivers activated                       1152 non-null   float64
 5   additional drivers needed due to not enough standbys  1152 non-null   float64
dtypes: float64(3), int64(3)
memory usage: 63.0 KB
```

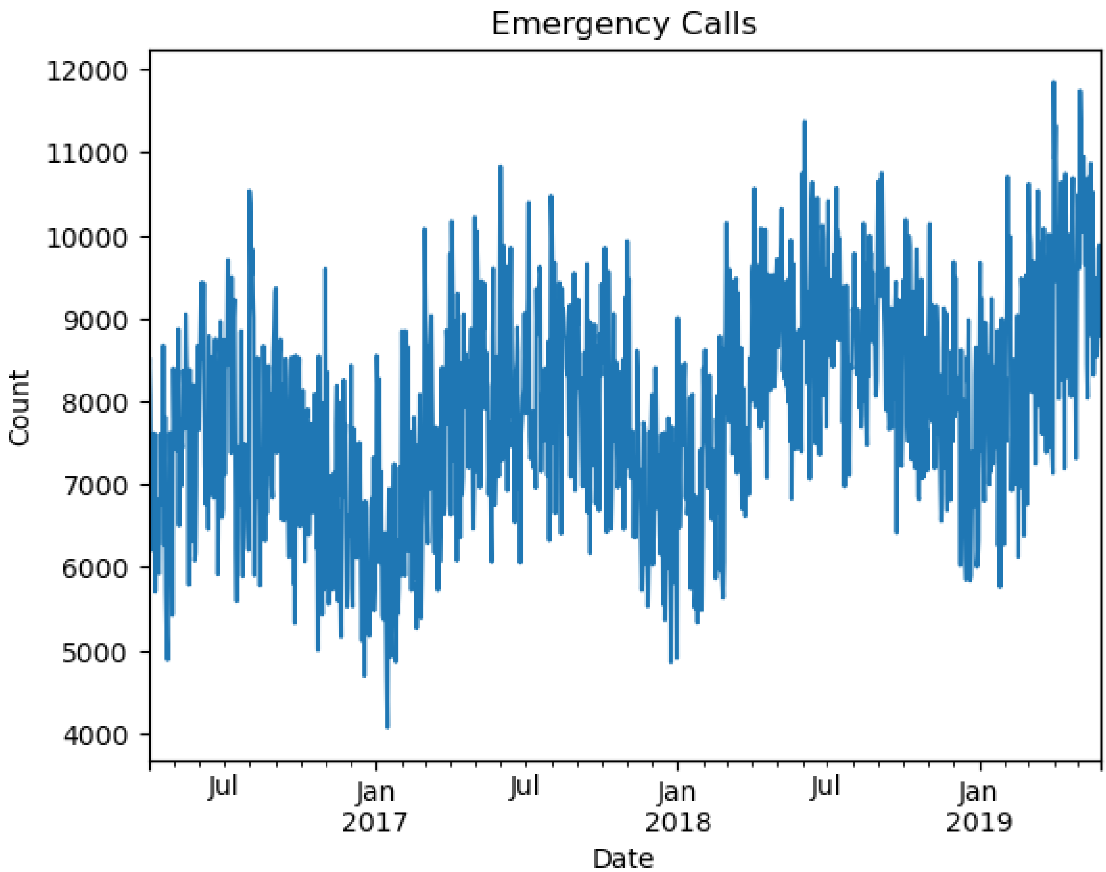
```
# Plot 'on-drivers that called in sick' over time
df['drivers on duty that called in sick'].plot()
plt.title(' On-duty Drivers that Called in Sick')
plt.ylabel('Count')
```

```
Text(0, 0.5, 'Count')
```



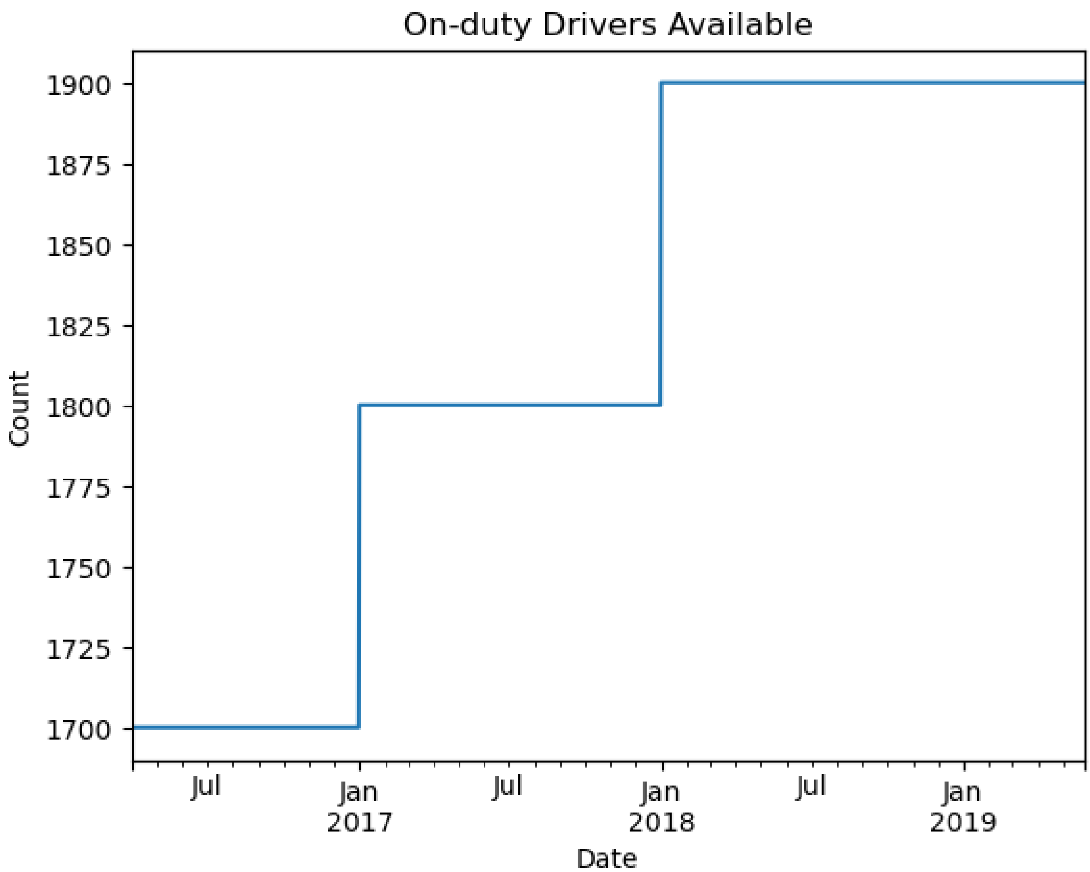

```
# Plot 'emergency calls' over time
df['emergency calls'].plot()
plt.title('Emergency Calls')
plt.ylabel('Count')
```

```
Text(0, 0.5, 'Count')
```



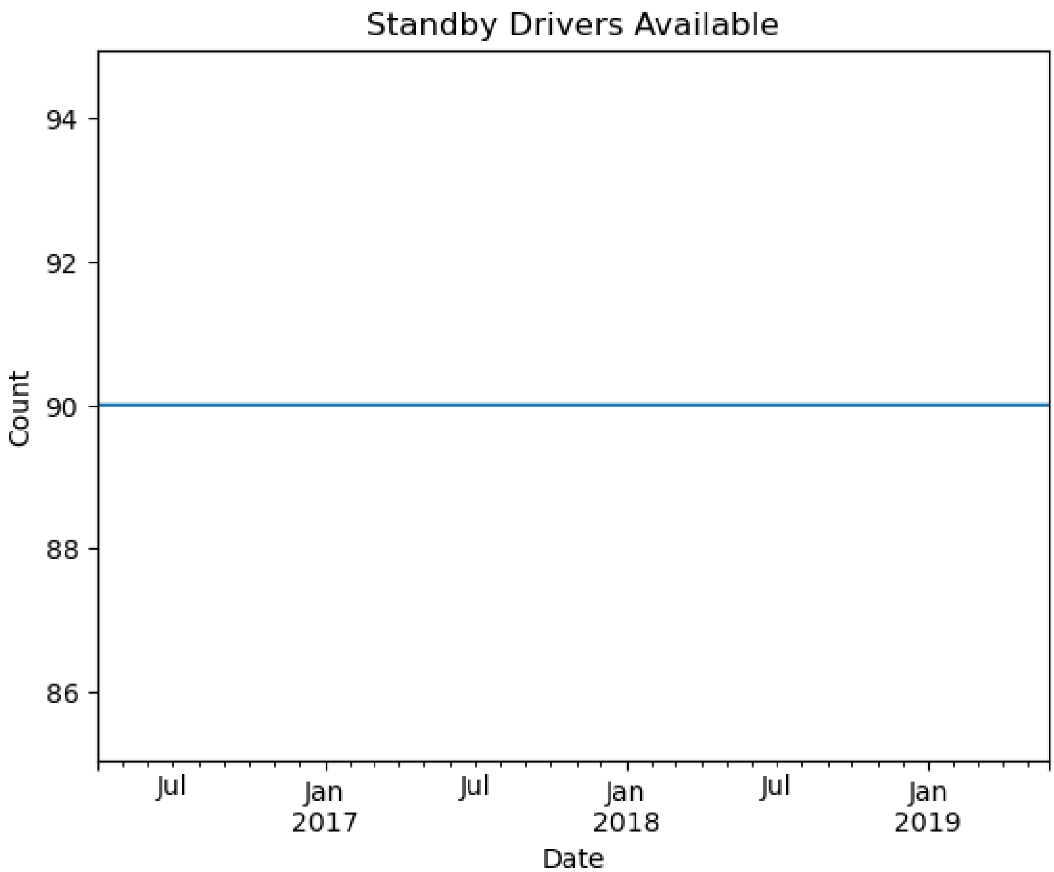

```
# Plot 'on-drivers available' over time
df['drivers on duty available'].plot()
plt.title('On-duty Drivers Available')
plt.ylabel('Count')
```

```
Text(0, 0.5, 'Count')
```



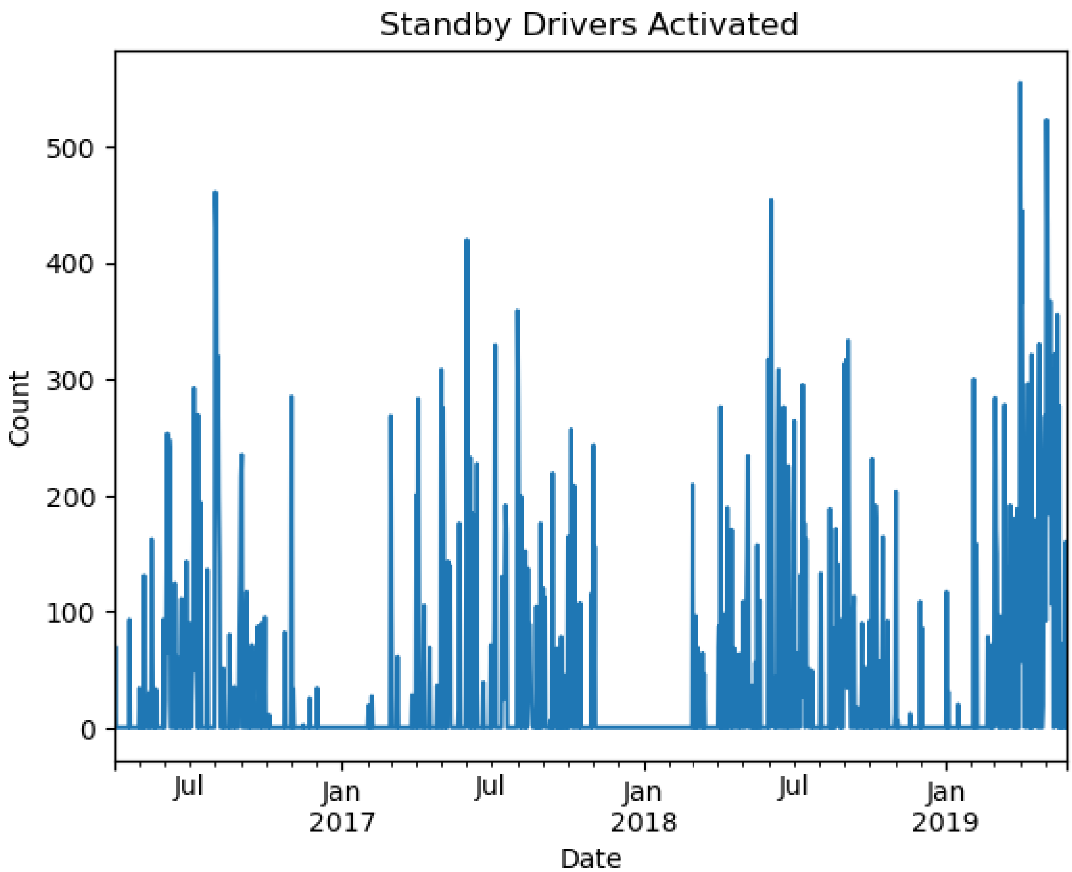

```
# Plot 'standby drivers available' over time
df['standby resources available'].plot()
plt.title('Standby Drivers Available')
plt.ylabel('Count')
```

```
Text(0, 0.5, 'Count')
```



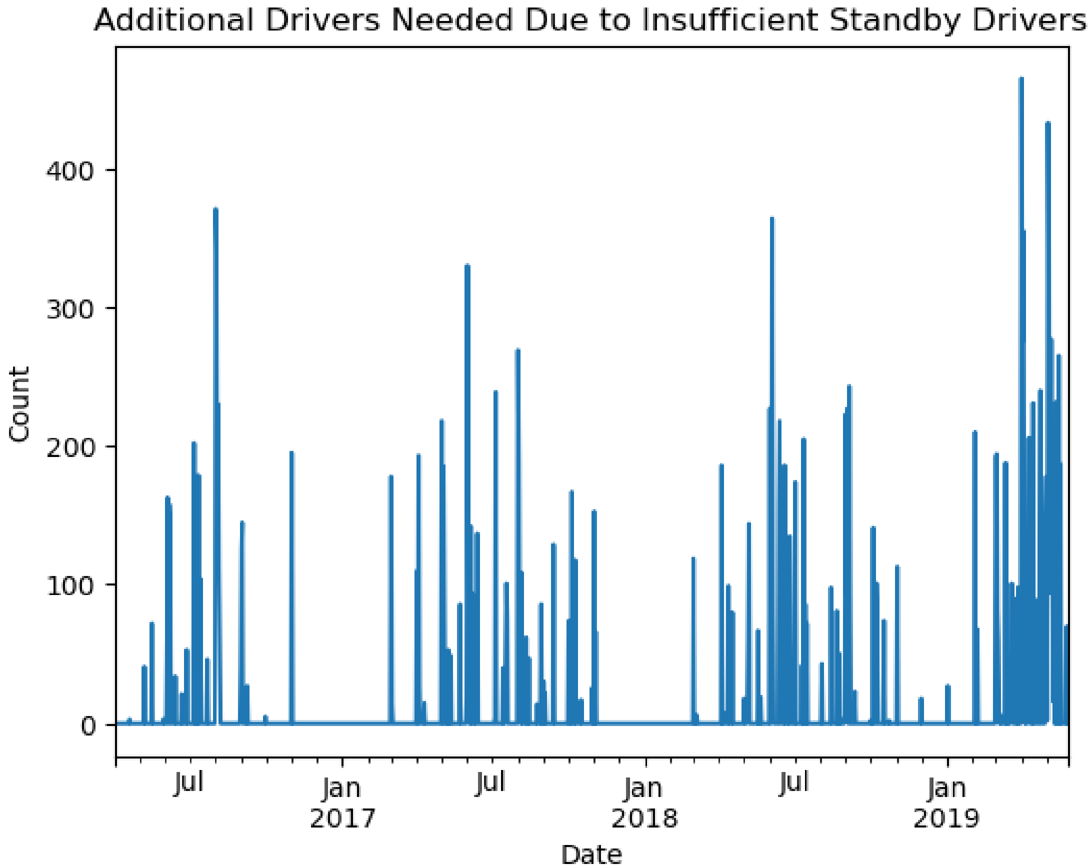

```
# Plot 'standbys activated' over time
df['standby drivers activated'].plot()
plt.title('Standby Drivers Activated')
plt.ylabel('Count')
```

```
Text(0, 0.5, 'Count')
```

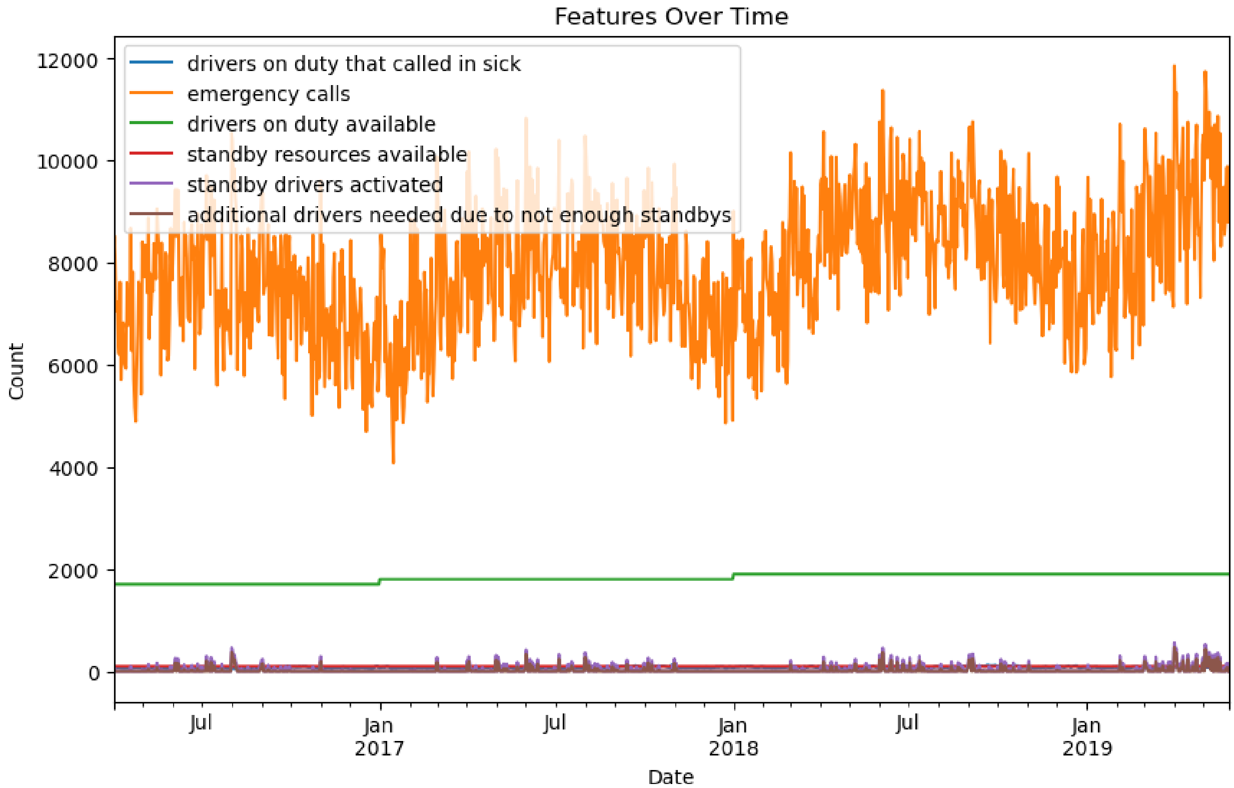



```python
# Plot 'additional drivers needed due to not enough standbys' over time
df['additional drivers needed due to not enough standbys'].plot()
plt.title('Additional Drivers Needed Due to Insufficient Standby Drivers')
plt.ylabel('Count')
```

```
Text(0, 0.5, 'Count')
```




```
# Plot all variables over time
plt.figure(figsize=(10, 6))
for label in df.columns:
    df[label].plot()
    plt.title('Features Over Time')
    plt.ylabel('Count')
    plt.legend()
```



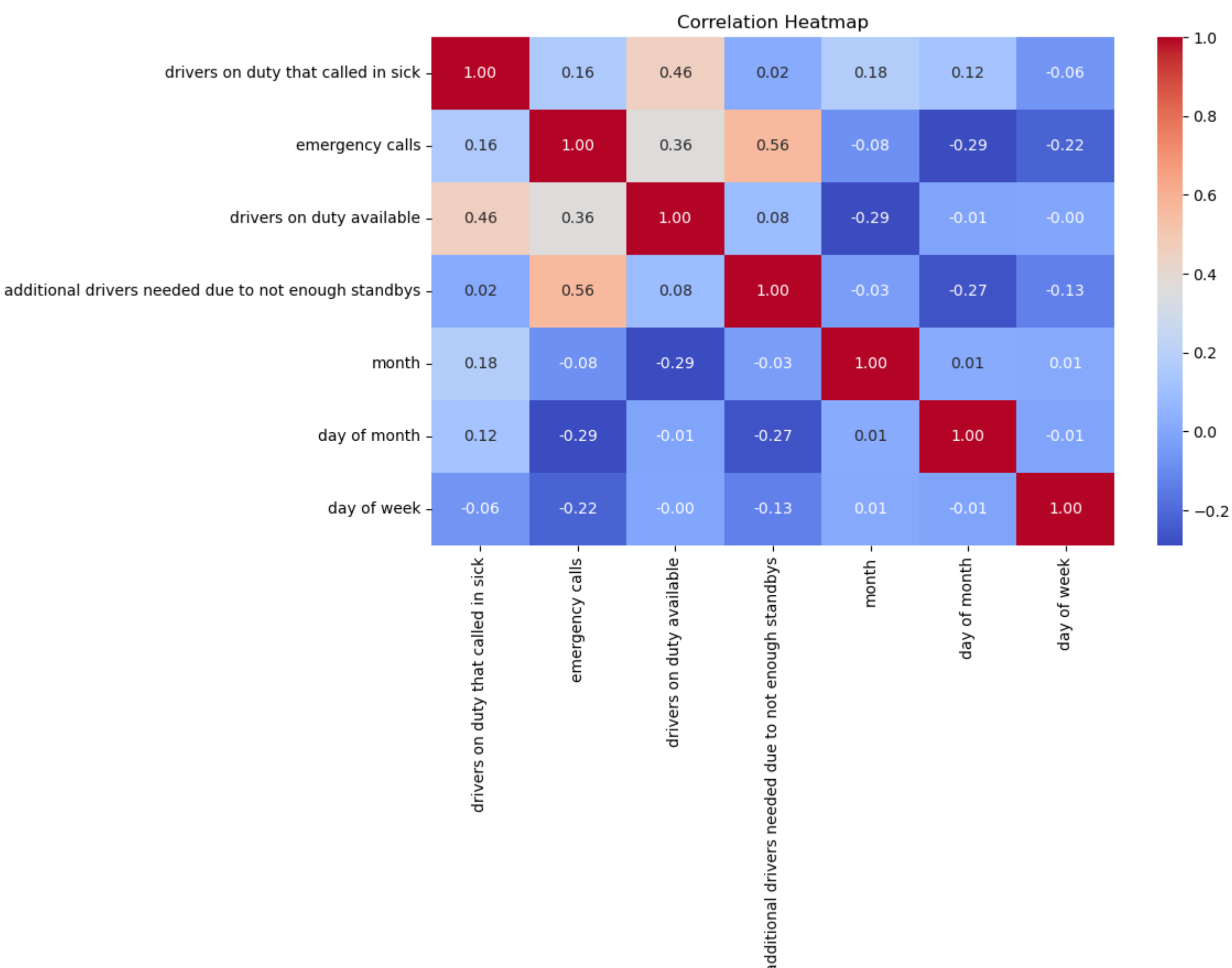
The fact that, regardless of the number of on-duty drivers calling in sick, the number of on-duty drivers available is almost always constant raises concern about the quality and veracity of the data. However, we will assume that when x on-duty drivers call in sick, x standby drivers are put on duty to replace those that called in sick and x non-standby drivers are put on standby to replace those that are now on duty.

```
# Create a new column called "month"
df['month'] = df.index.month

# Create a new column 'day' with the day of the month from the index
df['day of month'] = df.index.day

# Create a new column 'day of week' with the day of the week from the index (Monda
y is 0, Tuesday is 1, etc)
df['day of week'] = df.index.dayofweek

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1152 entries, 2016-04-01 to 2019-05-27
Data columns (total 9 columns):
 #   Column                                                 Non-Null Count  Dtype
---  ------                                                 --------------  -----
 0   drivers on duty that called in sick                    1152 non-null   int64
 1   emergency calls                                        1152 non-null   float64
 2   drivers on duty available                              1152 non-null   int64
 3   standby resources available                            1152 non-null   int64
 4   standby drivers activated                              1152 non-null   float64
```

```
 5   additional drivers needed due to not enough standbys  1152 non-null   float64
 6   month                                                 1152 non-null   int64
 7   day of month                                          1152 non-null   int64
 8   day of week                                           1152 non-null   int64
dtypes: float64(3), int64(6)
memory usage: 90.0 KB
```

```python
# Calculate correlation matrix
correlation_matrix = df.drop(columns=['standby resources available','standby drive
rs activated']).corr()

# Create heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```



There is a moderetely strong correlation between additional drivers needed due to not enough standby drivers and emergency calls. This is so because, considering that standby resources available is always 90, the higher the number of calls, the higher the number of additional standby drivers are needed.

After the threshold of calls that the on-duty and standby drivers can bear is reached, These two variables are expected to be directly proportional. Nevertheless, we want to predict the number of standby drivers activated so that we can set the predicted number of drivers on standby and ideally keep the additional drivers required to zero.

In addition, this variable (additional drivers needed due to not enough standby drivers) only exists because the number of standby drivers available is always 90. If we set the number of standby drivers available based on the prediction for activated standby drivers, then we will be able to keep additional drivers to zero (or close to zero).

Thus, additional standby drivers is not a variable that we should use in the modeling process. Therefore, we should drop it.

Based on the understanding we have of the data, we expect that the number of additional drivers needed due to not enough standbys + the number of standby resources available to be equal to the number of standbys activated, when the number of standbys activated is equal to or greater than 90. When the number of standby activated is less than 90 (or the number of standby resources available), the number of additional drivers needed due to not enough standby drivers is expected to be zero.

If these assuptions/expectations are true, we will drop the number of additional drivers needed due to not enough standbys and the number of standby resources available because apart from being a strongly correlated variable and a constant, respectively, their value/number is already embedded in the target variable.

```
df[['standby drivers activated','additional drivers needed due to not enough stand
bys','standby resources available']].head()
```

	standby drivers activated	additional drivers needed due to not enough standbys	standby resources available
Date			
2016-04-01	4.0	0.0	90
2016-04-02	70.0	0.0	90
2016-04-03	0.0	0.0	90
2016-04-04	0.0	0.0	90

	standby drivers activated	additional drivers needed due to not enough standbys	standby resources available
Date			
2016-04-05	0.0	0.0	90

```
# Filter rows where 'number_of_standbys_activated' is greater than 90
filtered_df = df[df['standby drivers activated'] >= 90]
filtered_df['sum of additional and available standby drivers'] = filtered_df['additional drivers needed due to not enough standbys'] + filtered_df['standby resources available']

# Display only selected columns for the filtered DataFrame
result = filtered_df[['sum of additional and available standby drivers',
                      'standby drivers activated',
                      'additional drivers needed due to not enough standbys',
                      'standby resources available']]
result.head()
```

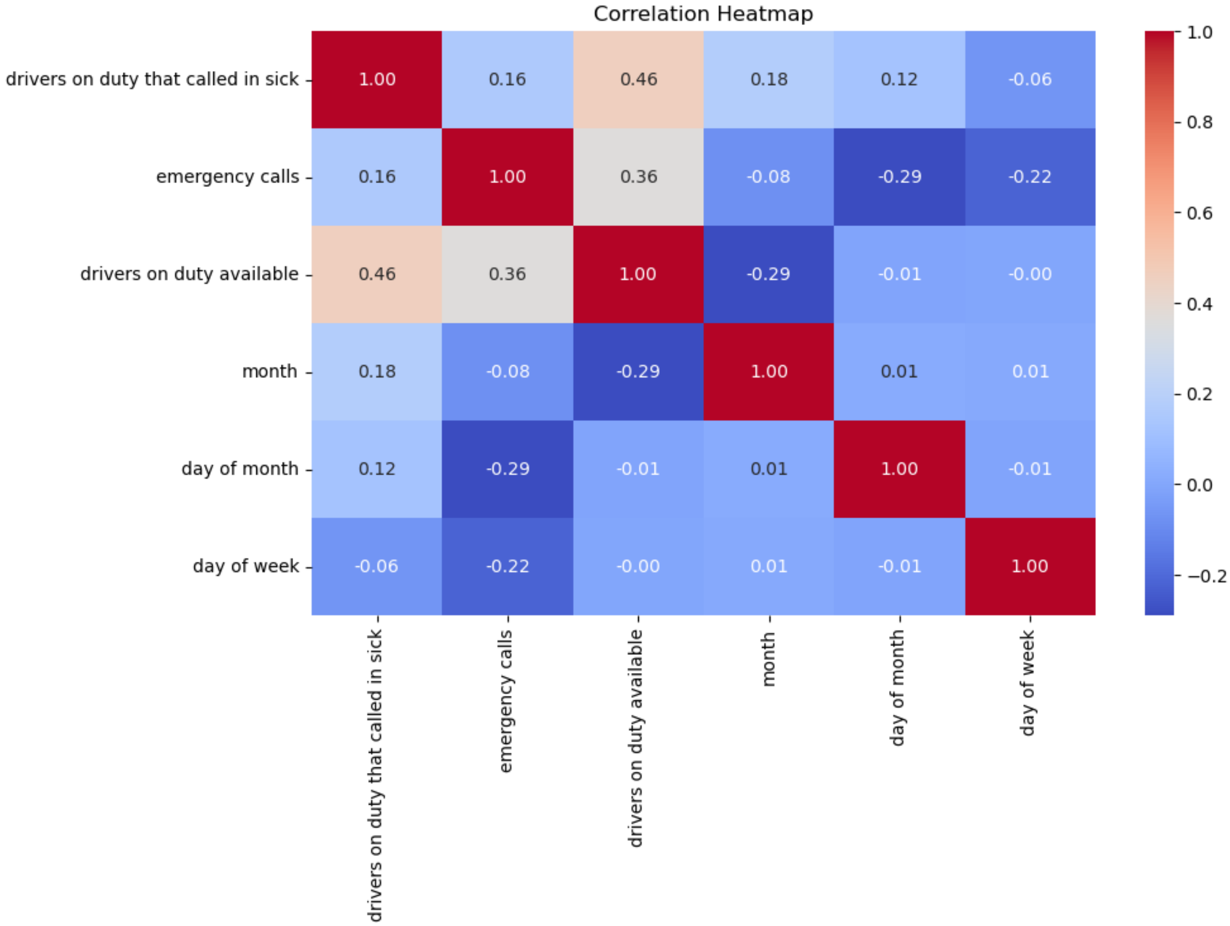
	sum of additional and available standby drivers	standby drivers activated	additional drivers needed due to not enough standbys	standby resources available
Date				
2016-04-19	93.0	93.0	3.0	90
2016-05-07	131.0	131.0	41.0	90
2016-05-16	162.0	162.0	72.0	90
2016-05-30	93.0	93.0	3.0	90
2016-06-03	154.0	154.0	64.0	90

The expectations/assumptions hold true. Thus, the two columns mentioned above shall be dropped from the dataset.

```
# Drop additional drivers needed due to not enough standbys and standby resources available
df = df.drop(columns=['additional drivers needed due to not enough standbys','stan
dby resources available'])
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1152 entries, 2016-04-01 to 2019-05-27
Data columns (total 7 columns):
 #   Column                             Non-Null Count  Dtype
---  ------                             --------------  -----
 0   drivers on duty that called in sick 1152 non-null   int64
 1   emergency calls                    1152 non-null   float64
 2   drivers on duty available          1152 non-null   int64
 3   standby drivers activated          1152 non-null   float64
 4   month                              1152 non-null   int64
 5   day of month                       1152 non-null   int64
 6   day of week                        1152 non-null   int64
dtypes: float64(2), int64(5)
memory usage: 72.0 KB
```

```
# Calculate correlation matrix
correlation_matrix2 = df.drop(columns=['standby drivers activated']).corr()

# Create heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix2, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```



MODEL BUILDING

Building the main model

We are interested in predicting the number of standby drivers activated. As demonstrated, there are temporal patterns in the data and there are independent features that do have some impact on the target feature. We will use time series models to predict the number of on-duty drivers that called in sick and the number of on-duty drivers that are available. The output from these models and the intended month, day of the week, and day of the month shall be fed into a regression model that will predict the number of standby drivers activated.

```
# Split the data into features and target variable
X = df.drop(["standby drivers activated"], axis=1)
y = df['standby drivers activated']

# Split the data into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_st
ate=42)

# Baseline model: Use mean of training labels as prediction
baseline_prediction = np.mean(y_train)
```

```python
# Evaluate baseline model
baseline_predictions = np.full_like(y_test, baseline_prediction)
baseline_mse = mean_squared_error(y_test, baseline_predictions)
baseline_r2 = r2_score(y_test, baseline_predictions)

print("Baseline Mean Squared Error:", baseline_mse)
print("Baseline R^2 Score:", baseline_r2)
```

```
Baseline Mean Squared Error: 3948.0596391092417
Baseline R^2 Score: -0.04265361361883846
```

```python
# Train a linear regression model
model_lr = LinearRegression()
model_lr.fit(X_train, y_train)

# Make predictions
y_pred_lr = model_lr.predict(X_test)

# Evaluate the model
mse_lr = mean_squared_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)

print("\nLinear Regression Mean Squared Error:", mse_lr)
print("Linear Regression R^2 Score:", r2_lr)
```

```
Linear Regression Mean Squared Error: 2524.2737605261527
Linear Regression R^2 Score: 0.33335780136046145
```

```python
# Instantiate the decision tree regressor
decision_tree = DecisionTreeRegressor(random_state=42)

# Train the decision tree model
decision_tree.fit(X_train, y_train)

# Make predictions
y_pred_dt = decision_tree.predict(X_test)

# Evaluate the decision tree model
mse_dt = mean_squared_error(y_test, y_pred_dt)
r2_dt = r2_score(y_test, y_pred_dt)

print("Decision Tree Mean Squared Error:", mse_dt)
print("Decision Tree R^2 Score:", r2_dt)
```

```
Decision Tree Mean Squared Error: 32.59307359307359
Decision Tree R^2 Score: 0.9913924081530768
```

```python
# Instantiate the Random Forest regressor
random_forest = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the Random Forest model
random_forest.fit(X_train, y_train)
```

```python
# Make predictions
y_pred_rf = random_forest.predict(X_test)

# Evaluate the Random Forest model
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print("Random Forest Mean Squared Error:", mse_rf)
print("Random Forest R^2 Score:", r2_rf)
```

```
Random Forest Mean Squared Error: 32.90772510822511
Random Forest R^2 Score: 0.9913093109941451
```

```python
# Instantiate the SVR model
svr = SVR(kernel='rbf')  # You can specify different kernels like 'linear', 'poly'
, 'rbf', etc.

# Train the SVR model
svr.fit(X_train, y_train)

# Make predictions
y_pred_svr = svr.predict(X_test)

# Evaluate the SVR model
mse_svr = mean_squared_error(y_test, y_pred_svr)
r2_svr = r2_score(y_test, y_pred_svr)

print("SVR Mean Squared Error:", mse_svr)
print("SVR R^2 Score:", r2_svr)
```

```
SVR Mean Squared Error: 3894.1987192886922
SVR R^2 Score: -0.02842933946466175
```

```python
# Instantiate the Ridge Regression model
ridge_model = Ridge(alpha=1.0)  # You can adjust the regularization strength by ch
anging alpha

# Train the Ridge Regression model
ridge_model.fit(X_train, y_train)

# Make predictions
y_pred_ridge = ridge_model.predict(X_test)

# Evaluate the Ridge Regression model
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)

print("Ridge Regression Mean Squared Error:", mse_ridge)
print("Ridge Regression R^2 Score:", r2_ridge)
```

```
Ridge Regression Mean Squared Error: 2524.2740713712647
Ridge Regression R^2 Score: 0.33335771926854585
```

```python
# Instantiate the Lasso Regression model
lasso_model = Lasso(alpha=1.0)  # You can adjust the regularization strength by ch
anging alpha

# Train the Lasso Regression model
lasso_model.fit(X_train, y_train)

# Make predictions
y_pred_lasso = lasso_model.predict(X_test)

# Evaluate the Lasso Regression model
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
r2_lasso = r2_score(y_test, y_pred_lasso)

print("Lasso Regression Mean Squared Error:", mse_lasso)
print("Lasso Regression R^2 Score:", r2_lasso)
```

```
Lasso Regression Mean Squared Error: 2525.815583388028
Lasso Regression R^2 Score: 0.33295061724333985
```

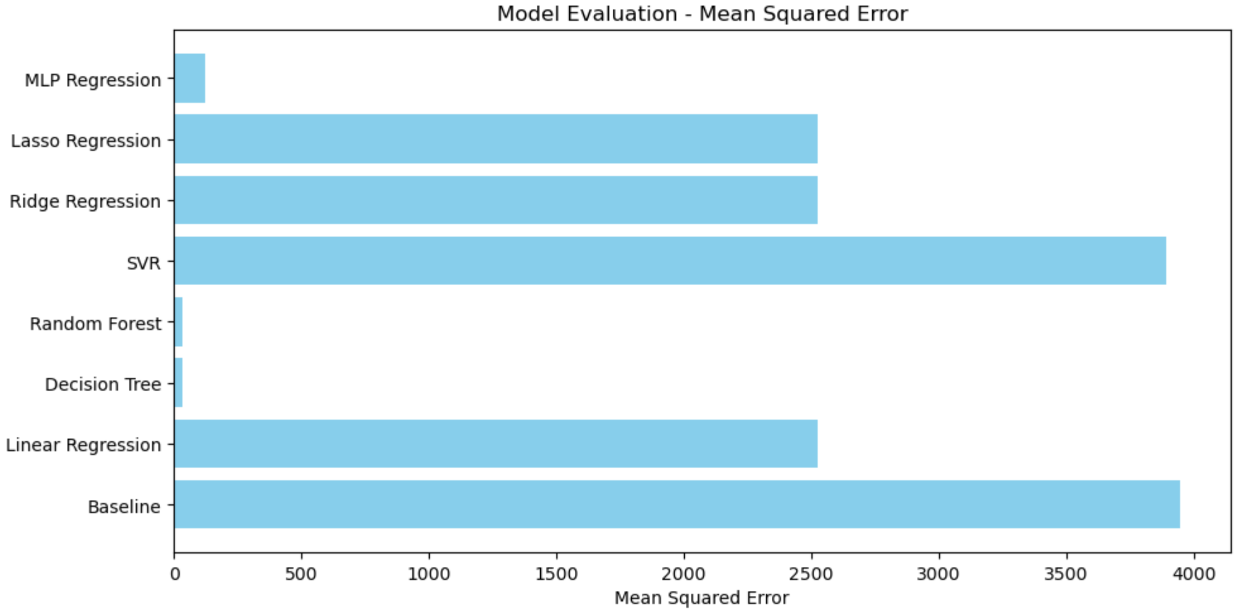
```python
# Instantiate the MultiLayerPerceptronRegressor model
mlp_model = MLPRegressor(hidden_layer_sizes=(100, 50), activation='relu', solver='
adam', random_state=42)

# Train the MLPRegressor model
mlp_model.fit(X_train, y_train)

# Make predictions
y_pred_mlp = mlp_model.predict(X_test)

# Evaluate the MLPRegressor model
mse_mlp = mean_squared_error(y_test, y_pred_mlp)
r2_mlp = r2_score(y_test, y_pred_mlp)

print("MLP Regression Mean Squared Error:", mse_mlp)
print("MLP Regression R^2 Score:", r2_mlp)
```

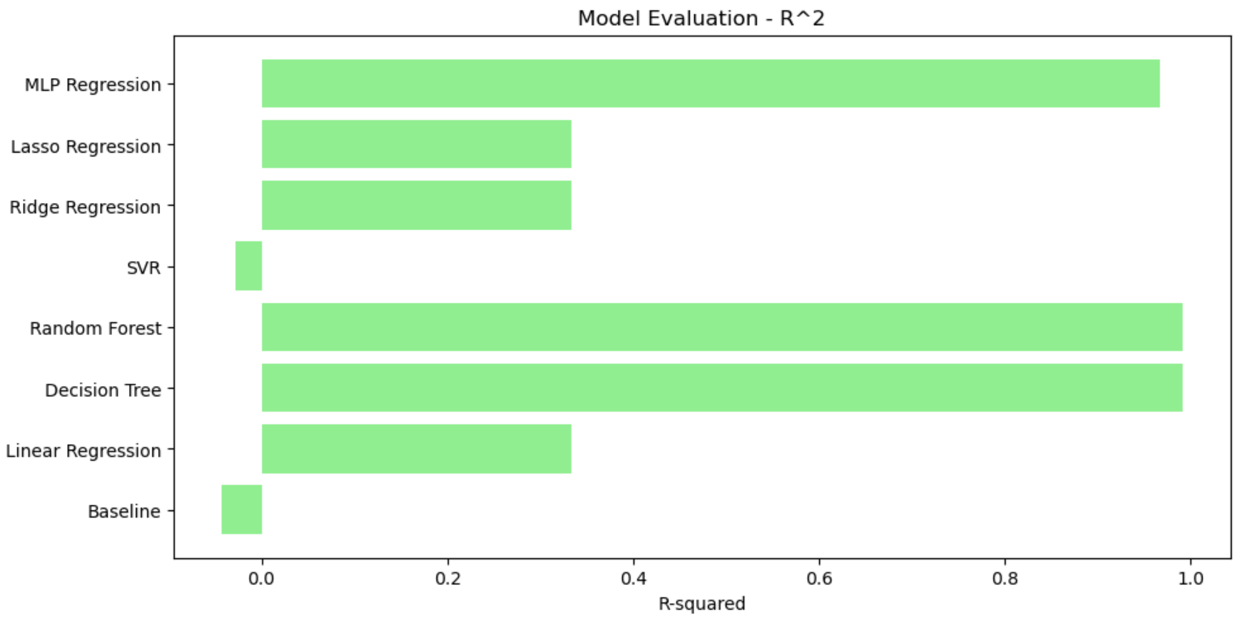
```
MLP Regression Mean Squared Error: 124.11091074358653
MLP Regression R^2 Score: 0.9672232181362078
```

```python
model_evaluation = pd.DataFrame({"mse":[baseline_mse, mse_lr, mse_dt, mse_rf, mse_
svr, mse_ridge, mse_lasso, mse_mlp],
                                 "r2":[baseline_r2, r2_lr, r2_dt, r2_rf, r2_svr, r
2_ridge, r2_lasso, r2_mlp],
                                 "label":["Baseline", "Linear Regression", "Decisi
on Tree", "Random Forest",
                                          "SVR", "Ridge Regression", "Lasso Regres
sion", "MLP Regression"]})

model_evaluation.head(10)
```

	mse	r2	label
0	3948.059639	-0.042654	Baseline
1	2524.273761	0.333358	Linear Regression
2	32.593074	0.991392	Decision Tree
3	32.907725	0.991309	Random Forest
4	3894.198719	-0.028429	SVR
5	2524.274071	0.333358	Ridge Regression
6	2525.815583	0.332951	Lasso Regression
7	124.110911	0.967223	MLP Regression

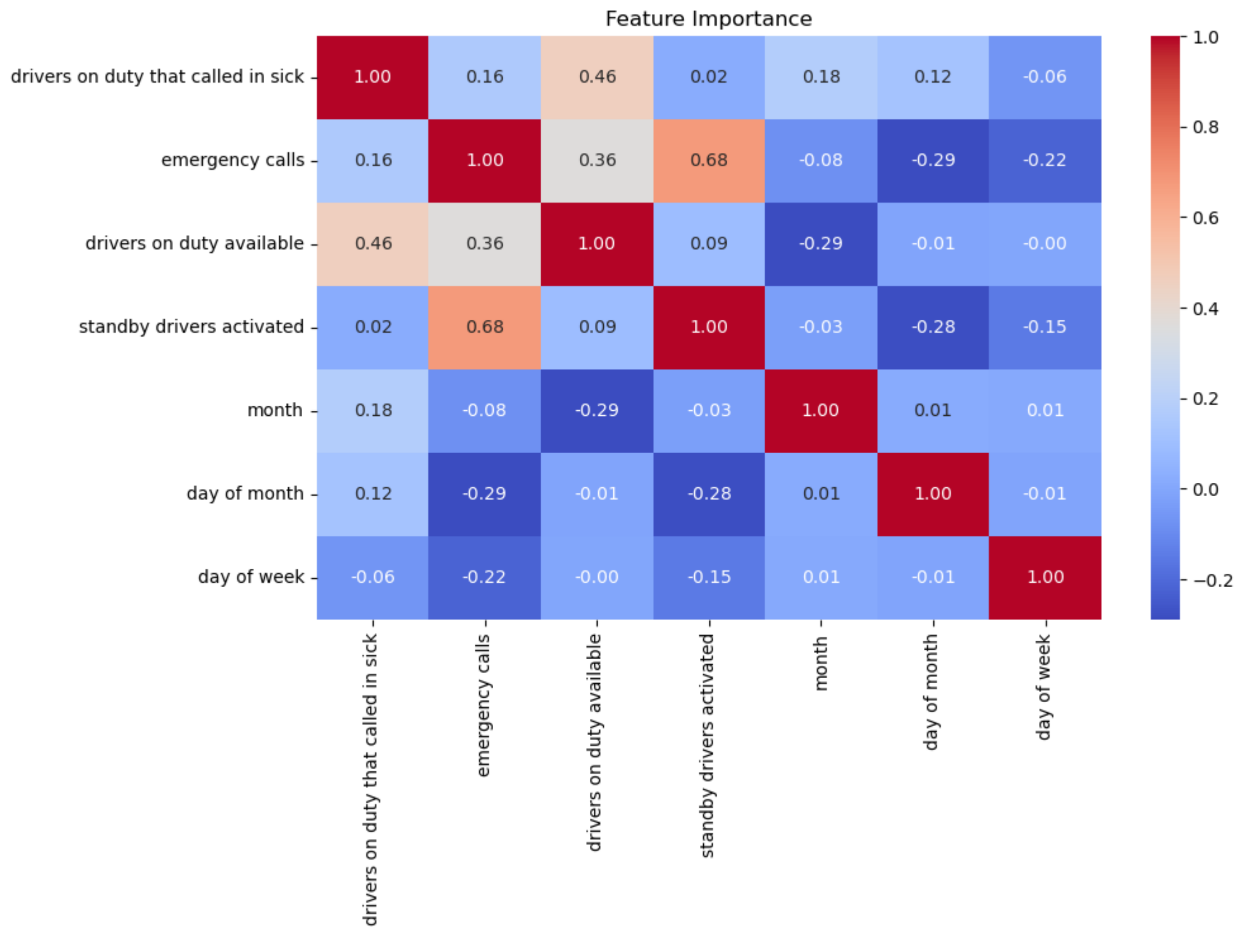
```python
# Plotting
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 10))

# MSE plot
ax1.barh(model_evaluation['label'], model_evaluation['mse'], color='skyblue')
ax1.set_xlabel('Mean Squared Error')
ax1.set_title('Model Evaluation - Mean Squared Error')

# R^2 plot
ax2.barh(model_evaluation['label'], model_evaluation['r2'], color='lightgreen')
ax2.set_xlabel('R-squared')
ax2.set_title('Model Evaluation - R^2')

plt.tight_layout()
plt.show()
```





Based on the results, Randon Forest is the best performing model. Therefore, we will Random Forest to predict the number of standby drivers activated.

```
# Calculate correlation matrix
correlation_matrix3 = df.corr()

# Create heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix3, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Feature Importance')
plt.show()
```

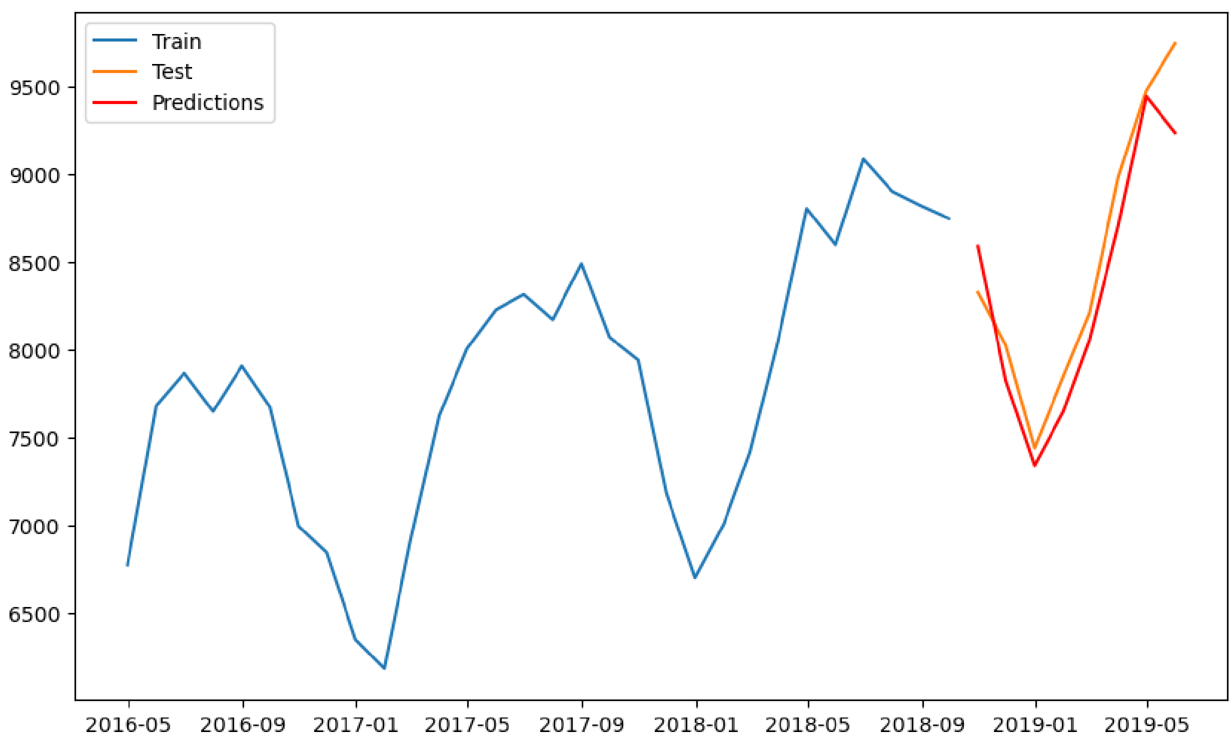


Predicting Emergency Calls

```
# Create a dataframe to store emergency calls data
df_calls = pd.DataFrame()
df_calls['emergency calls'] = df['emergency calls']


# Resample data from daily to monthly
df_calls = df_calls.resample('M').mean()

# Split data into train and test sets
train_size = int(len(df_calls) * 0.8)
train, test = df_calls.iloc[:train_size], df_calls.iloc[train_size:]

# Define and fit SARIMA model
order = (1, 1, 1)  # (p, d, q) parameters for non-seasonal components
seasonal_order = (1, 1, 1, 12)  # (P, D, Q, S) parameters for seasonal components
model_call = SARIMAX(train, order=order, seasonal_order=seasonal_order)
model_fit = model_call.fit()

# Make predictions
predictions_call = model_fit.predict(start=len(train), end=len(train) + len(test)
- 1, dynamic=False)
```

```
# Evaluate the model
mse = mean_squared_error(test, predictions_call)
rmse = np.sqrt(mse)
print('RMSE:', rmse)

# Plot results
plt.figure(figsize=(10, 6))
plt.plot(train.index, train, label='Train')
plt.plot(test.index, test, label='Test')
plt.plot(test.index, predictions_call, label='Predictions', color='red')
plt.legend()
plt.show()

# Forecast future emergency calls
forecast_steps = 12  # Example: forecast for 12 steps (months)
future_forecast = model_fit.forecast(steps=forecast_steps)
print('Future forecast:', future_forecast)
```

```
RMSE: 254.59663198738528
```

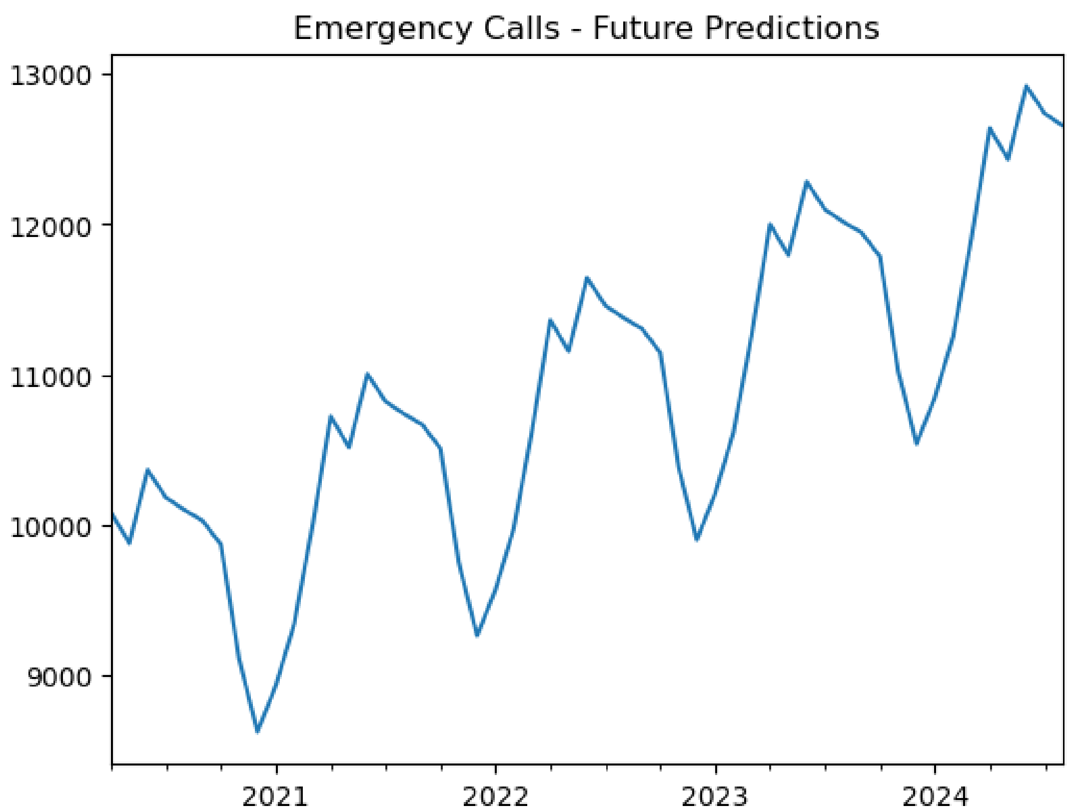



```
Future forecast: 2018-10-31    8589.855737
2018-11-30    7826.914237
2018-12-31    7342.432273
2019-01-31    7649.996874
2019-02-28    8057.104412
2019-03-31    8708.189330
2019-04-30    9443.257825
2019-05-31    9236.528630
2019-06-30    9726.206897
2019-07-31    9541.641249
```

```
2019-08-31    9459.179647
2019-09-30    9387.941398
Freq: M, Name: predicted_mean, dtype: float64
```

```
# Make predictions
predictions_call = model_fit.predict(start=48, end=100, dynamic=False)
predictions_call.plot()
plt.title('Emergency Calls - Future Predictions')
```

```
Text(0.5, 1.0, 'Emergency Calls - Future Predictions')
```



Data & Fuction Preparation For Predicting Sick and Available On-duty Drivers

```
# Creating a dataframe to store the data needed in the next time series models
df_on_duty_drivers = pd.DataFrame()
df_on_duty_drivers['drivers on duty that called in sick'] = df['drivers on duty th
at called in sick']
df_on_duty_drivers['drivers on duty available'] = df['drivers on duty available']

# Creating new columns based on date
df_on_duty_drivers['Seconds'] = df_on_duty_drivers.index.map(pd.Timestamp.timestam
p)

day = 60*60*24
```

```
year = 365.2425*day

df_on_duty_drivers['Day sin'] = np.sin(df_on_duty_drivers['Seconds'] * (2* np.pi /
day))
df_on_duty_drivers['Day cos'] = np.cos(df_on_duty_drivers['Seconds'] * (2 * np.pi
/ day))
df_on_duty_drivers['Year sin'] = np.sin(df_on_duty_drivers['Seconds'] * (2 * np.pi
/ year))
df_on_duty_drivers['Year cos'] = np.cos(df_on_duty_drivers['Seconds'] * (2 * np.pi
/ year))

# Creating two new dataframes
df_sick_drivers = df_on_duty_drivers.drop(['Seconds',
                              'drivers on duty available'], axis=1)

df_drivers_available = df_on_duty_drivers.drop(['Seconds',
                                   'drivers on duty that called in sick'], axis=1)

df_on_duty_drivers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1152 entries, 2016-04-01 to 2019-05-27
Data columns (total 7 columns):
 #   Column                               Non-Null Count  Dtype
---  ------                               --------------  -----
 0   drivers on duty that called in sick  1152 non-null   int64
 1   drivers on duty available            1152 non-null   int64
 2   Seconds                              1152 non-null   float64
 3   Day sin                              1152 non-null   float64
 4   Day cos                              1152 non-null   float64
 5   Year sin                             1152 non-null   float64
 6   Year cos                             1152 non-null   float64
dtypes: float64(5), int64(2)
memory usage: 72.0 KB
```

```
# Defining a funtion to extract X and y from df

def df_to_X_y(df, window_size=7):
  df_as_np = df.to_numpy()
  X = []
  y = []
  for i in range(len(df_as_np)-window_size):
    row = [r for r in df_as_np[i:i+window_size]]
    X.append(row)
    label = [df_as_np[i+window_size][0]]
    y.append(label)
  return np.array(X), np.array(y)

# Defining a funtion to plot results

def plot_predictions(model, X, y, start=0, end=100,):
  predictions = model.predict(X).flatten()
  df = pd.DataFrame(data={'Predictions': predictions, 'Actuals':y.flatten()})
```

```python
  mse_value =  mean_squared_error(predictions, y)
  plt.plot(df['Predictions'][start:end], label='Prediction')
  plt.plot(df['Actuals'][start:end], label='Actual')
  plt.title('Sick Drivers')
  plt.legend()
  return df, mse_value

# Defining a function to plot results

def plot_predictions2(model, X, y, start=0, end=100,):
  predictions = model.predict(X).flatten()
  df = pd.DataFrame(data={'Predictions': predictions, 'Actuals':y.flatten()})
  mse_value =  mean_squared_error(predictions, y)
  plt.plot(df['Predictions'][start:end], label='Prediction')
  plt.plot(df['Actuals'][start:end], label='Actual')
  plt.title('Available Drivers')
  plt.legend()
  return df, mse_value
```

Predicting the Number of On-duty Sick Drivers

```python
WINDOW_SIZE = 7
X1, y1 = df_to_X_y(df_sick_drivers, WINDOW_SIZE)
X1.shape, y1.shape
```

```
((1145, 7, 5), (1145, 1))
```

```python
X_train1, y_train1 = X1[:916], y1[:916]
X_val1, y_val1 = X1[916:1031], y1[916:1031]
X_test1, y_test1 = X1[1031:], y1[1031:]
X_train1.shape, y_train1.shape, X_val1.shape, y_val1.shape, X_test1.shape, y_test1
.shape
```

```
((916, 7, 5), (916, 1), (115, 7, 5), (115, 1), (114, 7, 5), (114, 1))
```

```python
model_sick = Sequential()
model_sick.add(InputLayer((7, 5)))
model_sick.add(Conv1D(32, kernel_size=2))
model_sick.add(Flatten())
model_sick.add(Dense(8, 'relu'))
model_sick.add(Dense(1, 'linear'))

model_sick.summary()
```

```
WARNING:tensorflow:From C:\Users\JC\anaconda3\Lib\site-packages\keras\src\backend.
py:873: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_d
efault_graph instead.

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 6, 32)             352
```

```
 flatten (Flatten)            (None, 192)               0

 dense (Dense)                (None, 8)                 1544

 dense_1 (Dense)              (None, 1)                 9

=================================================================
Total params: 1905 (7.44 KB)
Trainable params: 1905 (7.44 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
cp = ModelCheckpoint('model_sick/', save_best_only=True)
model_sick.compile(loss=MeanSquaredError(), optimizer=Adam(learning_rate=0.001), m
etrics=[RootMeanSquaredError()])

model_sick.fit(X_train1, y_train1, validation_data=(X_val1, y_val1), epochs=30, ca
llbacks=[cp])
```

```
Epoch 1/30
WARNING:tensorflow:From C:\Users\JC\anaconda3\Lib\site-packages\keras\src\utils\tf
_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.c
ompat.v1.ragged.RaggedTensorValue instead.

17/29 [================>.............] - ETA: 0s - loss: 1589.3279 - root_mean_squ
ared_error: 39.8664 INFO:tensorflow:Assets written to: model_sick\assets


INFO:tensorflow:Assets written to: model_sick\assets


29/29 [==============================] - 7s 92ms/step - loss: 1007.5209 - root_mea
n_squared_error: 31.7415 - val_loss: 289.3402 - val_root_mean_squared_error: 17.01
00
Epoch 2/30
17/29 [================>.............] - ETA: 0s - loss: 121.8203 - root_mean_squa
red_error: 11.0372INFO:tensorflow:Assets written to: model_sick\assets


INFO:tensorflow:Assets written to: model_sick\assets


29/29 [==============================] - 2s 58ms/step - loss: 107.9577 - root_mean
_squared_error: 10.3903 - val_loss: 105.3863 - val_root_mean_squared_error: 10.265
8
Epoch 3/30
26/29 [=========================>....] - ETA: 0s - loss: 81.0069 - root_mean_squar
ed_error: 9.0004INFO:tensorflow:Assets written to: model_sick\assets


INFO:tensorflow:Assets written to: model_sick\assets
```

```
29/29 [==============================] - 1s 50ms/step - loss: 82.6192 - root_mean_
squared_error: 9.0895 - val_loss: 97.5151 - val_root_mean_squared_error: 9.8750
Epoch 4/30
16/29 [===============>..............] - ETA: 0s - loss: 79.8727 - root_mean_squar
ed_error: 8.9372INFO:tensorflow:Assets written to: model_sick\assets


INFO:tensorflow:Assets written to: model_sick\assets


29/29 [==============================] - 1s 47ms/step - loss: 80.1275 - root_mean_
squared_error: 8.9514 - val_loss: 94.8405 - val_root_mean_squared_error: 9.7386
Epoch 5/30
29/29 [==============================] - 0s 12ms/step - loss: 77.1599 - root_mean_
squared_error: 8.7841 - val_loss: 95.1087 - val_root_mean_squared_error: 9.7524
Epoch 6/30
24/29 [=======================>......] - ETA: 0s - loss: 73.6555 - root_mean_squar
ed_error: 8.5823INFO:tensorflow:Assets written to: model_sick\assets


INFO:tensorflow:Assets written to: model_sick\assets


29/29 [==============================] - 2s 62ms/step - loss: 74.3245 - root_mean_
squared_error: 8.6212 - val_loss: 90.1267 - val_root_mean_squared_error: 9.4935
Epoch 7/30
28/29 [===========================>..] - ETA: 0s - loss: 71.8477 - root_mean_squar
ed_error: 8.4763INFO:tensorflow:Assets written to: model_sick\assets


INFO:tensorflow:Assets written to: model_sick\assets


29/29 [==============================] - 2s 63ms/step - loss: 71.2917 - root_mean_
squared_error: 8.4434 - val_loss: 87.0311 - val_root_mean_squared_error: 9.3290
Epoch 8/30
16/29 [===============>..............] - ETA: 0s - loss: 73.3907 - root_mean_squar
ed_error: 8.5668INFO:tensorflow:Assets written to: model_sick\assets


INFO:tensorflow:Assets written to: model_sick\assets


29/29 [==============================] - 1s 48ms/step - loss: 68.5371 - root_mean_
squared_error: 8.2787 - val_loss: 85.2897 - val_root_mean_squared_error: 9.2352
Epoch 9/30
29/29 [==============================] - 0s 12ms/step - loss: 66.2957 - root_mean_
squared_error: 8.1422 - val_loss: 89.1088 - val_root_mean_squared_error: 9.4397
Epoch 10/30
25/29 [========================>.....] - ETA: 0s - loss: 64.0031 - root_mean_squar
ed_error: 8.0002INFO:tensorflow:Assets written to: model_sick\assets
```

```
INFO:tensorflow:Assets written to: model_sick\assets


29/29 [==============================] - 1s 40ms/step - loss: 63.9110 - root_mean_
squared_error: 7.9944 - val_loss: 76.9277 - val_root_mean_squared_error: 8.7708
Epoch 11/30
28/29 [===========================>..] - ETA: 0s - loss: 60.9844 - root_mean_squar
ed_error: 7.8092INFO:tensorflow:Assets written to: model_sick\assets


INFO:tensorflow:Assets written to: model_sick\assets


29/29 [==============================] - 2s 54ms/step - loss: 61.1866 - root_mean_
squared_error: 7.8222 - val_loss: 74.3113 - val_root_mean_squared_error: 8.6204
Epoch 12/30
19/29 [==================>...........] - ETA: 0s - loss: 56.4444 - root_mean_squar
ed_error: 7.5129INFO:tensorflow:Assets written to: model_sick\assets


INFO:tensorflow:Assets written to: model_sick\assets


29/29 [==============================] - 1s 44ms/step - loss: 57.9791 - root_mean_
squared_error: 7.6144 - val_loss: 73.3855 - val_root_mean_squared_error: 8.5665
Epoch 13/30
17/29 [================>.............] - ETA: 0s - loss: 55.9602 - root_mean_squar
ed_error: 7.4807INFO:tensorflow:Assets written to: model_sick\assets


INFO:tensorflow:Assets written to: model_sick\assets


29/29 [==============================] - 2s 56ms/step - loss: 55.7177 - root_mean_
squared_error: 7.4644 - val_loss: 70.6127 - val_root_mean_squared_error: 8.4031
Epoch 14/30
16/29 [===============>..............] - ETA: 0s - loss: 56.0677 - root_mean_squar
ed_error: 7.4878INFO:tensorflow:Assets written to: model_sick\assets


INFO:tensorflow:Assets written to: model_sick\assets


29/29 [==============================] - 2s 77ms/step - loss: 53.8377 - root_mean_
squared_error: 7.3374 - val_loss: 68.4784 - val_root_mean_squared_error: 8.2752
Epoch 15/30
27/29 [==========================>...] - ETA: 0s - loss: 50.6146 - root_mean_squar
ed_error: 7.1144INFO:tensorflow:Assets written to: model_sick\assets
```

```
INFO:tensorflow:Assets written to: model_sick\assets


29/29 [==============================] - 2s 71ms/step - loss: 52.0947 - root_mean_
squared_error: 7.2177 - val_loss: 66.3029 - val_root_mean_squared_error: 8.1427
Epoch 16/30
16/29 [==============>..............] - ETA: 0s - loss: 46.3289 - root_mean_squar
ed_error: 6.8065INFO:tensorflow:Assets written to: model_sick\assets


INFO:tensorflow:Assets written to: model_sick\assets


29/29 [==============================] - 1s 43ms/step - loss: 51.0028 - root_mean_
squared_error: 7.1416 - val_loss: 64.6316 - val_root_mean_squared_error: 8.0394
Epoch 17/30
18/29 [================>............] - ETA: 0s - loss: 49.7288 - root_mean_squar
ed_error: 7.0519INFO:tensorflow:Assets written to: model_sick\assets


INFO:tensorflow:Assets written to: model_sick\assets


29/29 [==============================] - 1s 49ms/step - loss: 51.1250 - root_mean_
squared_error: 7.1502 - val_loss: 63.4216 - val_root_mean_squared_error: 7.9638
Epoch 18/30
16/29 [==============>..............] - ETA: 0s - loss: 50.1275 - root_mean_squar
ed_error: 7.0801INFO:tensorflow:Assets written to: model_sick\assets


INFO:tensorflow:Assets written to: model_sick\assets


29/29 [==============================] - 1s 49ms/step - loss: 48.8415 - root_mean_
squared_error: 6.9887 - val_loss: 62.3435 - val_root_mean_squared_error: 7.8958
Epoch 19/30
29/29 [==============================] - 0s 5ms/step - loss: 46.9775 - root_mean_s
quared_error: 6.8540 - val_loss: 64.1359 - val_root_mean_squared_error: 8.0085
Epoch 20/30
24/29 [=======================>......] - ETA: 0s - loss: 47.9477 - root_mean_squar
ed_error: 6.9244INFO:tensorflow:Assets written to: model_sick\assets


INFO:tensorflow:Assets written to: model_sick\assets


29/29 [==============================] - 1s 43ms/step - loss: 46.0237 - root_mean_
squared_error: 6.7841 - val_loss: 60.7134 - val_root_mean_squared_error: 7.7919
Epoch 21/30
19/29 [=================>..........] - ETA: 0s - loss: 48.8084 - root_mean_squar
ed_error: 6.9863INFO:tensorflow:Assets written to: model_sick\assets
```

```
INFO:tensorflow:Assets written to: model_sick\assets


29/29 [==============================] - 1s 41ms/step - loss: 45.2089 - root_mean_
squared_error: 6.7238 - val_loss: 60.4991 - val_root_mean_squared_error: 7.7781
Epoch 22/30
27/29 [==========================>...] - ETA: 0s - loss: 43.9228 - root_mean_squar
ed_error: 6.6274INFO:tensorflow:Assets written to: model_sick\assets


INFO:tensorflow:Assets written to: model_sick\assets


29/29 [==============================] - 2s 57ms/step - loss: 45.3793 - root_mean_
squared_error: 6.7364 - val_loss: 59.4480 - val_root_mean_squared_error: 7.7103
Epoch 23/30
13/29 [===========>..................] - ETA: 0s - loss: 40.7348 - root_mean_squar
ed_error: 6.3824INFO:tensorflow:Assets written to: model_sick\assets


INFO:tensorflow:Assets written to: model_sick\assets


29/29 [==============================] - 2s 66ms/step - loss: 44.1789 - root_mean_
squared_error: 6.6467 - val_loss: 58.9951 - val_root_mean_squared_error: 7.6808
Epoch 24/30
16/29 [==============>...............] - ETA: 0s - loss: 38.5057 - root_mean_squar
ed_error: 6.2053INFO:tensorflow:Assets written to: model_sick\assets


INFO:tensorflow:Assets written to: model_sick\assets


29/29 [==============================] - 1s 53ms/step - loss: 43.1252 - root_mean_
squared_error: 6.5670 - val_loss: 58.5900 - val_root_mean_squared_error: 7.6544
Epoch 25/30
27/29 [==========================>...] - ETA: 0s - loss: 40.9683 - root_mean_squar
ed_error: 6.4006INFO:tensorflow:Assets written to: model_sick\assets


INFO:tensorflow:Assets written to: model_sick\assets


29/29 [==============================] - 1s 46ms/step - loss: 43.0936 - root_mean_
squared_error: 6.5646 - val_loss: 58.2909 - val_root_mean_squared_error: 7.6348
Epoch 26/30
18/29 [================>.............] - ETA: 0s - loss: 39.8607 - root_mean_squar
ed_error: 6.3135INFO:tensorflow:Assets written to: model_sick\assets


INFO:tensorflow:Assets written to: model_sick\assets
```

```
29/29 [==============================] - 1s 52ms/step - loss: 42.2221 - root_mean_
squared_error: 6.4979 - val_loss: 58.1424 - val_root_mean_squared_error: 7.6251
Epoch 27/30
18/29 [=================>............] - ETA: 0s - loss: 46.1316 - root_mean_squar
ed_error: 6.7920INFO:tensorflow:Assets written to: model_sick\assets


INFO:tensorflow:Assets written to: model_sick\assets


29/29 [==============================] - 1s 48ms/step - loss: 42.0107 - root_mean_
squared_error: 6.4816 - val_loss: 57.8014 - val_root_mean_squared_error: 7.6027
Epoch 28/30
29/29 [==============================] - 0s 4ms/step - loss: 41.8476 - root_mean_s
quared_error: 6.4690 - val_loss: 57.9077 - val_root_mean_squared_error: 7.6097
Epoch 29/30
20/29 [===================>..........] - ETA: 0s - loss: 40.9531 - root_mean_squar
ed_error: 6.3995INFO:tensorflow:Assets written to: model_sick\assets


INFO:tensorflow:Assets written to: model_sick\assets


29/29 [==============================] - 1s 42ms/step - loss: 42.0190 - root_mean_
squared_error: 6.4822 - val_loss: 57.6350 - val_root_mean_squared_error: 7.5918
Epoch 30/30
29/29 [==============================] - 0s 5ms/step - loss: 42.0783 - root_mean_s
quared_error: 6.4868 - val_loss: 58.0227 - val_root_mean_squared_error: 7.6173
```

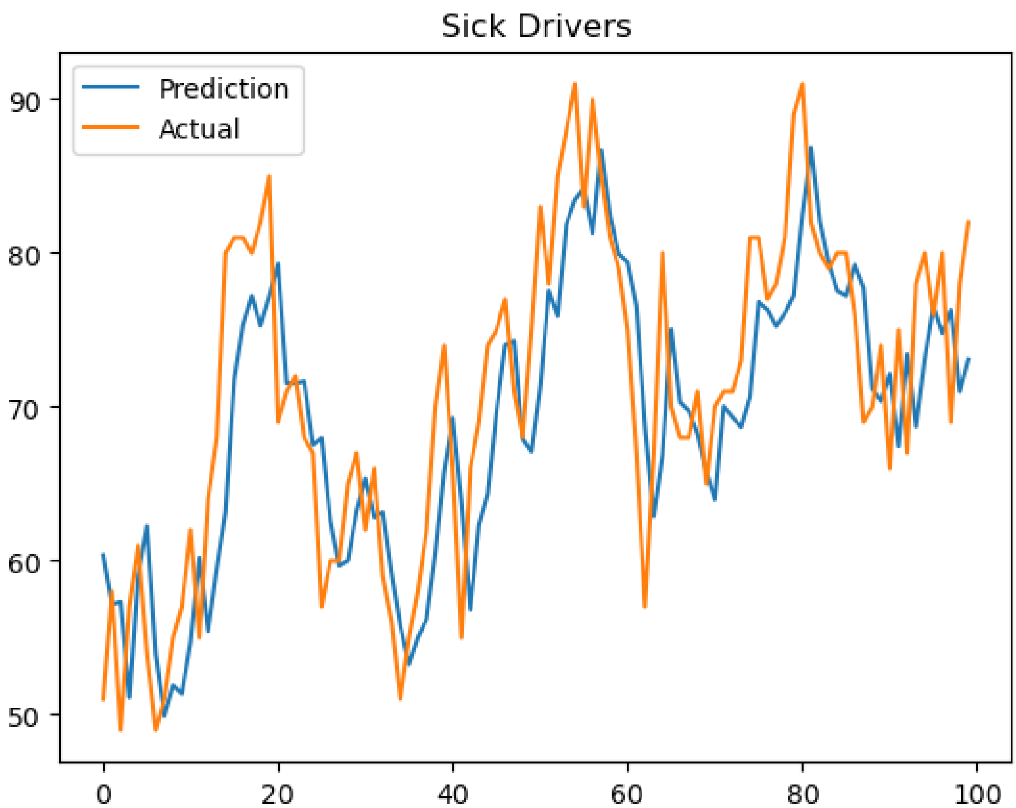
```
<keras.src.callbacks.History at 0x1cf66255250>
```

```
plot_predictions(model_sick, X_test1, y_test1)
```

```
4/4 [==============================] - 0s 4ms/step
```

```
(     Predictions  Actuals
 0      60.352203     51.0
 1      57.144787     58.0
 2      57.327007     49.0
 3      51.103374     57.0
 4      59.064884     61.0
 ..           ...      ...
```

```
109    74.590141     86.0
110    83.732262     81.0
111    76.882034     76.0
112    76.458420     83.0
113    82.523857     77.0

[114 rows x 2 columns],
41.30176027910758)
```



Predicting On-duty Drivers Available

```
WINDOW_SIZE = 7
X2, y2 = df_to_X_y(df_drivers_available, WINDOW_SIZE)
X2.shape, y2.shape
```

```
((1145, 7, 5), (1145, 1))
```

```
X_train2, y_train2 = X2[:916], y2[:916]
X_val2, y_val2 = X2[916:1031], y2[916:1031]
X_test2, y_test2 = X2[1031:], y2[1031:]
X_train2.shape, y_train2.shape, X_val2.shape, y_val2.shape, X_test2.shape, y_test2
.shape
```

```
((916, 7, 5), (916, 1), (115, 7, 5), (115, 1), (114, 7, 5), (114, 1))
```

```
model_available = Sequential([
    InputLayer((7, 5)),
    Conv1D(32, kernel_size=2),
```

```
    Flatten(),
    Dense(8, activation='relu'),
    Dense(1, activation='linear')
])

model_available.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv1d_1 (Conv1D)           (None, 6, 32)             352       
                                                                 
 flatten_1 (Flatten)         (None, 192)               0         
                                                                 
 dense_2 (Dense)             (None, 8)                 1544      
                                                                 
 dense_3 (Dense)             (None, 1)                 9         
                                                                 
=================================================================
Total params: 1905 (7.44 KB)
Trainable params: 1905 (7.44 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Setting up model checkpoint to save the best model during training
# `save_best_only=True` ensures only the best model based on validation loss will
be saved
cp = ModelCheckpoint('model_available/', save_best_only=True)

# Compiling the model
from tensorflow.keras.optimizers import Adamax



model_available.compile(loss=MeanSquaredError(), optimizer=Adam(learning_rate=0.00
01), metrics=[RootMeanSquaredError()])
#model2.compile(loss=MeanSquaredError(), optimizer = Adamax(learning_rate=0.002, b
eta_1=0.9, beta_2=0.999), metrics=[RootMeanSquaredError()])

model_available.fit(X_train2, y_train2, validation_data=(X_val2, y_val2), epochs=3
0, callbacks=[cp])
```

```
Epoch 1/30
19/29 [==================>...........] - ETA: 0s - loss: 6607701.5000 - root_mean_
squared_error: 2570.5449 INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 2s 46ms/step - loss: 6277540.5000 - root_
mean_squared_error: 2505.5020 - val_loss: 5872978.5000 - val_root_mean_squared_err
```

```
or: 2423.4229
Epoch 2/30
27/29 [==========================>...] - ETA: 0s - loss: 4559578.0000 - root_mean_
squared_error: 2135.3169INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 1s 45ms/step - loss: 4523698.5000 - root_
mean_squared_error: 2126.8987 - val_loss: 4163560.7500 - val_root_mean_squared_err
or: 2040.4805
Epoch 3/30
19/29 [==================>...........] - ETA: 0s - loss: 3359710.0000 - root_mean_
squared_error: 1832.9512INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 1s 36ms/step - loss: 3162176.0000 - root_
mean_squared_error: 1778.2509 - val_loss: 2890349.0000 - val_root_mean_squared_err
or: 1700.1027
Epoch 4/30
24/29 [=======================>......] - ETA: 0s - loss: 2346650.2500 - root_mean_
squared_error: 1531.8781INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 1s 43ms/step - loss: 2301234.5000 - root_
mean_squared_error: 1516.9821 - val_loss: 2299720.0000 - val_root_mean_squared_err
or: 1516.4828
Epoch 5/30
20/29 [===================>..........] - ETA: 0s - loss: 2021842.2500 - root_mean_
squared_error: 1421.9150INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 1s 39ms/step - loss: 1994511.8750 - root_
mean_squared_error: 1412.2719 - val_loss: 2116252.7500 - val_root_mean_squared_err
or: 1454.7346
Epoch 6/30
15/29 [==============>...............] - ETA: 0s - loss: 1861314.2500 - root_mean_
squared_error: 1364.2999INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets
```

```
29/29 [==============================] - 2s 57ms/step - loss: 1827306.8750 - root_
mean_squared_error: 1351.7792 - val_loss: 1930000.5000 - val_root_mean_squared_err
or: 1389.2446
Epoch 7/30
17/29 [================>.............] - ETA: 0s - loss: 1693091.3750 - root_mean_
squared_error: 1301.1885INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 2s 56ms/step - loss: 1659863.7500 - root_
mean_squared_error: 1288.3571 - val_loss: 1744524.5000 - val_root_mean_squared_err
or: 1320.8044
Epoch 8/30
17/29 [================>.............] - ETA: 0s - loss: 1530097.8750 - root_mean_
squared_error: 1236.9712INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 1s 42ms/step - loss: 1494254.7500 - root_
mean_squared_error: 1222.3971 - val_loss: 1562819.0000 - val_root_mean_squared_err
or: 1250.1276
Epoch 9/30
20/29 [===================>..........] - ETA: 0s - loss: 1354031.0000 - root_mean_
squared_error: 1163.6284INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 2s 55ms/step - loss: 1332499.2500 - root_
mean_squared_error: 1154.3394 - val_loss: 1386232.6250 - val_root_mean_squared_err
or: 1177.3838
Epoch 10/30
20/29 [===================>..........] - ETA: 0s - loss: 1199177.3750 - root_mean_
squared_error: 1095.0696INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 1s 36ms/step - loss: 1176238.2500 - root_
mean_squared_error: 1084.5452 - val_loss: 1216253.5000 - val_root_mean_squared_err
or: 1102.8389
Epoch 11/30
23/29 [======================>.......] - ETA: 0s - loss: 1040293.1250 - root_mean_
squared_error: 1019.9476INFO:tensorflow:Assets written to: model_available\assets
```

```
INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 1s 42ms/step - loss: 1026737.3125 - root_
mean_squared_error: 1013.2805 - val_loss: 1055127.8750 - val_root_mean_squared_err
or: 1027.1942
Epoch 12/30
19/29 [==================>...........] - ETA: 0s - loss: 908495.6875 - root_mean_s
quared_error: 953.1504INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 1s 39ms/step - loss: 885765.1250 - root_m
ean_squared_error: 941.1509 - val_loss: 903834.9375 - val_root_mean_squared_error:
950.7023
Epoch 13/30
19/29 [==================>...........] - ETA: 0s - loss: 775654.1250 - root_mean_s
quared_error: 880.7123INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 1s 42ms/step - loss: 754298.6875 - root_m
ean_squared_error: 868.5037 - val_loss: 763933.7500 - val_root_mean_squared_error:
874.0330
Epoch 14/30
19/29 [==================>...........] - ETA: 0s - loss: 651471.8125 - root_mean_s
quared_error: 807.1381INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 1s 37ms/step - loss: 633477.5625 - root_m
ean_squared_error: 795.9130 - val_loss: 636464.1250 - val_root_mean_squared_error:
797.7870
Epoch 15/30
20/29 [===================>..........] - ETA: 0s - loss: 538384.5000 - root_mean_s
quared_error: 733.7469INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 1s 46ms/step - loss: 524190.4688 - root_m
ean_squared_error: 724.0100 - val_loss: 522193.2500 - val_root_mean_squared_error:
722.6294
Epoch 16/30
```

```
18/29 [================>............] - ETA: 0s - loss: 442682.5000 - root_mean_s
quared_error: 665.3439INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 1s 37ms/step - loss: 426943.9688 - root_m
ean_squared_error: 653.4095 - val_loss: 421371.1250 - val_root_mean_squared_error:
649.1310
Epoch 17/30
21/29 [====================>.........] - ETA: 0s - loss: 351610.6562 - root_mean_s
quared_error: 592.9677INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 1s 42ms/step - loss: 341935.1562 - root_m
ean_squared_error: 584.7522 - val_loss: 334090.2812 - val_root_mean_squared_error:
578.0054
Epoch 18/30
20/29 [===================>..........] - ETA: 0s - loss: 277779.4062 - root_mean_s
quared_error: 527.0479INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 1s 48ms/step - loss: 268968.4375 - root_m
ean_squared_error: 518.6216 - val_loss: 260114.1719 - val_root_mean_squared_error:
510.0139
Epoch 19/30
26/29 [=========================>....] - ETA: 0s - loss: 210332.7188 - root_mean_s
quared_error: 458.6205INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 2s 61ms/step - loss: 207738.5469 - root_m
ean_squared_error: 455.7834 - val_loss: 198463.5625 - val_root_mean_squared_error:
445.4925
Epoch 20/30
17/29 [================>............] - ETA: 0s - loss: 166321.2188 - root_mean_s
quared_error: 407.8250INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 1s 53ms/step - loss: 157255.6406 - root_m
```

```
ean_squared_error: 396.5547 - val_loss: 148578.4375 - val_root_mean_squared_error:
385.4587
Epoch 21/30
18/29 [=================>............] - ETA: 0s - loss: 122953.3438 - root_mean_s
quared_error: 350.6470INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 1s 46ms/step - loss: 116722.8750 - root_m
ean_squared_error: 341.6473 - val_loss: 108956.0234 - val_root_mean_squared_error:
330.0849
Epoch 22/30
26/29 [=========================>....] - ETA: 0s - loss: 86167.7109 - root_mean_sq
uared_error: 293.5434INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 1s 39ms/step - loss: 84892.9922 - root_me
an_squared_error: 291.3640 - val_loss: 78218.7578 - val_root_mean_squared_error: 2
79.6762
Epoch 23/30
20/29 [===================>..........] - ETA: 0s - loss: 63759.3008 - root_mean_sq
uared_error: 252.5060INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 1s 51ms/step - loss: 60463.3320 - root_me
an_squared_error: 245.8929 - val_loss: 54988.2812 - val_root_mean_squared_error: 2
34.4958
Epoch 24/30
19/29 [==================>...........] - ETA: 0s - loss: 44634.5000 - root_mean_sq
uared_error: 211.2688INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 1s 34ms/step - loss: 42182.9258 - root_me
an_squared_error: 205.3848 - val_loss: 37837.4258 - val_root_mean_squared_error: 1
94.5184
Epoch 25/30
15/29 [==============>...............] - ETA: 0s - loss: 31306.5820 - root_mean_sq
uared_error: 176.9367INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets
```

```
29/29 [==============================] - 1s 49ms/step - loss: 28824.1191 - root_me
an_squared_error: 169.7767 - val_loss: 25498.6367 - val_root_mean_squared_error: 1
59.6829
Epoch 26/30
29/29 [==============================] - ETA: 0s - loss: 19307.9688 - root_mean_sq
uared_error: 138.9531INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 2s 61ms/step - loss: 19307.9688 - root_me
an_squared_error: 138.9531 - val_loss: 16812.0508 - val_root_mean_squared_error: 1
29.6613
Epoch 27/30
29/29 [==============================] - ETA: 0s - loss: 12680.3525 - root_mean_sq
uared_error: 112.6071INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 1s 37ms/step - loss: 12680.3525 - root_me
an_squared_error: 112.6071 - val_loss: 10850.5654 - val_root_mean_squared_error: 1
04.1660
Epoch 28/30
16/29 [===============>..............] - ETA: 0s - loss: 8954.5312 - root_mean_squ
ared_error: 94.6284INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 1s 48ms/step - loss: 8169.2515 - root_mea
n_squared_error: 90.3839 - val_loss: 6867.4761 - val_root_mean_squared_error: 82.8
702
Epoch 29/30
19/29 [==================>...........] - ETA: 0s - loss: 5534.3374 - root_mean_squ
ared_error: 74.3931INFO:tensorflow:Assets written to: model_available\assets


INFO:tensorflow:Assets written to: model_available\assets


29/29 [==============================] - 1s 46ms/step - loss: 5174.2319 - root_mea
n_squared_error: 71.9321 - val_loss: 4256.5918 - val_root_mean_squared_error: 65.2
426
Epoch 30/30
28/29 [===========================>..] - ETA: 0s - loss: 3245.2603 - root_mean_squ
ared_error: 56.9672INFO:tensorflow:Assets written to: model_available\assets
```

```
INFO:tensorflow:Assets written to: model_available\assets
```

```
29/29 [==============================] - 1s 50ms/step - loss: 3226.4072 - root_mea
n_squared_error: 56.8015 - val_loss: 2583.5054 - val_root_mean_squared_error: 50.8
282
```

```
<keras.src.callbacks.History at 0x1cf6add6f10>
```

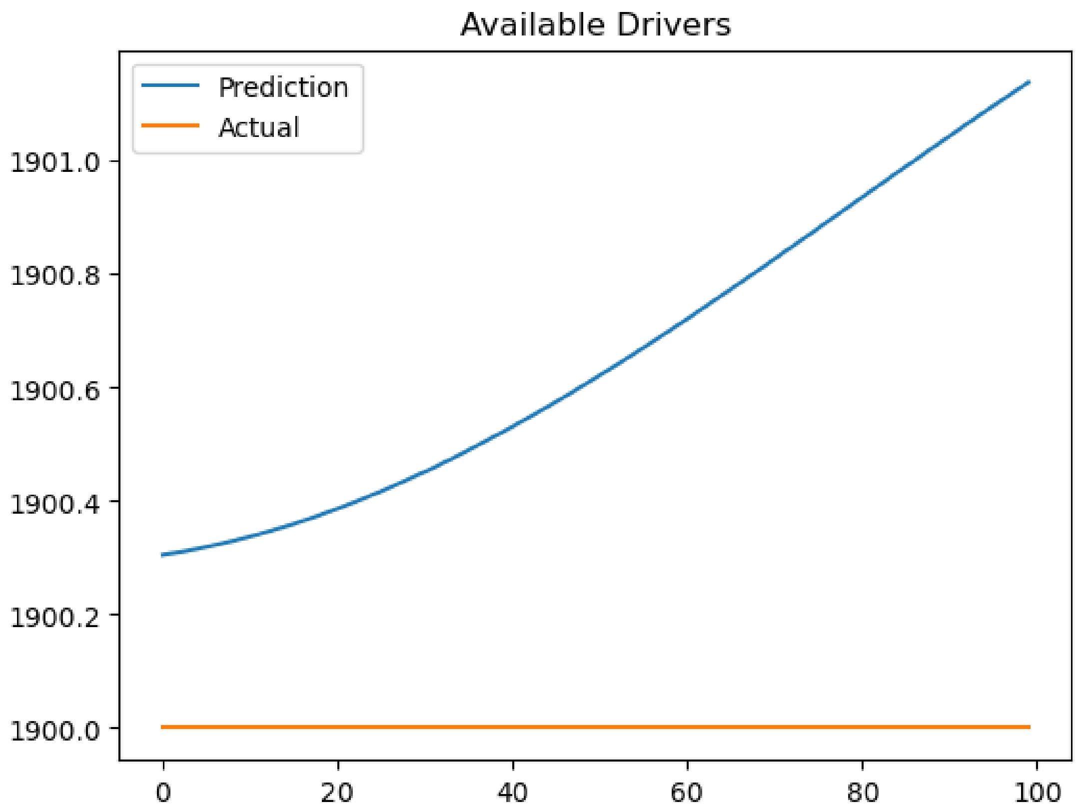
```
plot_predictions2(model_available, X_test2, y_test2)
```

```
4/4 [==============================] - 0s 3ms/step
```

```
4/4 [==============================] - 0s 3ms/step
```

Out[120]:

```
(      Predictions  Actuals
 0     1900.305298   1900.0
 1     1900.307739   1900.0
 2     1900.309937   1900.0
 3     1900.312866   1900.0
 4     1900.315796   1900.0
 ..            ...      ...
 109  1901.237183   1900.0
 110  1901.246216   1900.0
 111  1901.255615   1900.0
 112  1901.264771   1900.0
 113  1901.274048   1900.0

 [114 rows x 2 columns],
 0.6132620592650614)
```


Available Drivers
Prediction
Actual
1901.0
1900.8
1900.6
1900.4
1900.2
1900.0
0
20
40
60
80
100