Automation of Standby Duty Planning for Rescue Driver via Forecasting Model

Author:	Tony Wang
Email:	tony-sibo.wang@iu-study.org
Type of Work:	Case Study
Matriculation No:	32113891
Study:	Data Science (M. Sc)
Course:	Case Study: Model Engineering
Submission Date:	01.11.2023
Prof. in charge:	Sahar Qaadan

Table of Content

1. Introduction

The CRISP-DM methodology which stands for Cross-Industry Standard Process for Data Mining was pioneered in the late 1990s by different member companies in the European Union. It laid the groundwork for systematic data analysis and created a guideline for working with projects in the field of data science. Despite ever evolving methods and approaches in the field, CRISP-DM remains a foundational standard, offering essential insights to data science intricacies (Chapman, 1999).

CRISP-DM aims to guide the developers in their endeavors by following predefined steps and achieving milestones. It is possible to revise and repeat certain steps, should the desired outcome not be achieved initially (IBM, 2021).

This goal of this case study is to apply the CRISP-DM methodology to a practical data science application. In the following case, an application for automation of standby duty planning for rescue drivers for Berlin's Red Cross should be developed. The data science application should support the HR department in their planning efforts and increasing duty efficiency. Every step will be addressed and elucidated, along with its underlying intentions.

2. Applying CRISP-DM Methodology

2.1. Business Understanding

In context of CRISP-DM, the phase Business Understanding plays a pivotal role since it builds to foundation of the entire work (Shearer, 2000). Through this structured approach, the project gains a clear direction, fostering efficient implementation and optimal outcomes. All further steps are based on the decisions made in this first step which makes it crucial to understand the Red Cross' endeavor.

The current problem that the Red Cross in Berlin is facing, lies in the planning of standby drivers. "Due to short-term sickness of rescue drivers or an unusual amount of emergency calls often more drivers are needed than initially expected" (IU, 2023). For this reason, every day there are 90 standby drivers kept on hold and activated when needed. However due to seasonal patterns, even with additional standby drivers, it sometimes does not suffice and drivers on their day off have to called and employed. Goal of this project is to predict the number of standby drivers needed for each day and increasing the overall efficiency of planning standby drivers. By doing so, the overall number of standby drivers planned will also be reduced.

The project aim is to estimate the amount of daily standby rescue drivers needed. Also, we want to achieve a higher efficiency rate, and reduce the number of overall standby drivers needed.

2.2. Data Understanding

Once the business objectives are clear, this stage involves assessing the existing data. Initial data quality checks such as identifying missing values and assessing data quality are performed. Utilizing visualizations like scatterplots and histograms proves immensely valuable, offering developers and project collaborators profound insights (IBM, 2021).

In the following an exploratory data analysis is performed on the existing variables.

2.2.1 General Information

The dataset given in the task consists of a csv-file with 1152 rows and 7 columns. The individual columns consist of the following variables (IU, 2023):

- **date**: The entry data set is ordered after its date. The date is entered in the following format: yyyy-mm-dd. The earliest entry is 2016-04-01 and its last entry is 2019-05-27. Every individual day between its earliest and latest entry is present with no missing days in-between.
- **n_sick**: This variable represents the number of drivers who called sick before or during their duty on that specific date. This variable ranges from 36 to 119.
- **n_duty**: This variable represents the overall number of drivers on duty on each specific date. At the end of year, the number of drivers on duty is increased by 100, with the exception of 2018, where n_duty was not increased for 2019. This variable ranges from 1700 to 1900.
- **calls**: This variable describes the number of emergency calls, that the rescue drivers receive. This number ranges from 4074 up to 11850.
- **n_sby**: This variable represents the number of standby drivers on each duty who can be called into duty, when needed. This number is consistently at 90 for all dates.
- **sby_need**: On each day, the number of actual standby drivers who are called into duty is different. On most days, this number is 0 and on some occasions this number can reach to a maximum of 555.
- **dafted**: This variable represents the number of additional drivers needed due to not having enough standby drivers. This number is calculated by subtracting sby_need and n_sby. When the result is supposed to be negative, the number is entered as 0.

The entire dataset doesn't have any missing or invalid values. Since the task is to support HR in an improved planning of standby drivers, the variable sby_need seems to be the target variable of the prediction. Since n_sby always assumes the same value, namely of 90, and dafted is the resulting subtraction of sby_need and n_sby, these two variables do not need to be observed any further and can be dropped as redundant information.

2.2.2 Data Visualisation by Year

The following graphs depict the most relevant variables in the dataset, ordered by their values averaged over the four years. All variables in the data set show an increasing trend with each continuous year. While the average of n_sick, n_duty and n_calls show an almost linear increasing with each year, the average of sby_needed increases almost tremendously. Noticeable is the fact that, in 2018 even though the average amount of n_sick is higher than 2019, the additional standby drivers needed is lower in 2018 than 2019. A first conjecture could be that the amount of additional standby drivers needed (sby_need) is largely dependent on the number of emergency calls.

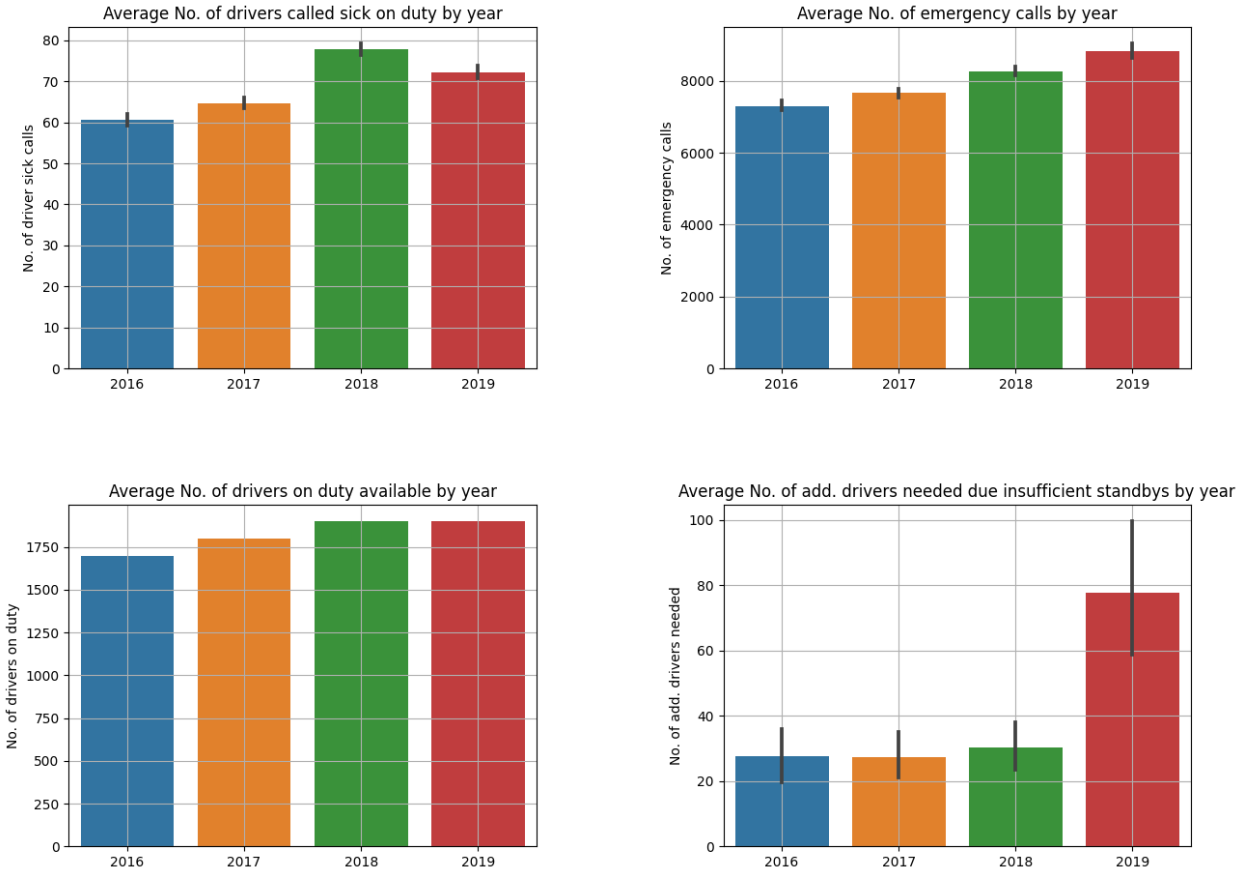

Figure 1: Variable Visualisation by Year

2.2.3 Data Visualisation by Month

The following graphs depict the same variables as in chapter 2.2.2. but in this graph their averages are ordered by month. In the first subplot, it is noticeable that in most months, the average number of driver sick calls is at about 60. September and October form an obvious exception with the average of n_sick being over 80.

The average number of emergency calls increase in the summer months, with May and June being the peaking months. The winter months, especially January and December are about 1000 emergency calls lower on average.

In this visualization n_duty has no relevance since, that number is only changed at the start of the year in January.

Eye-catchingly, even though September and October have a much higher driver sick call on average, the average of sby_need in those months are not particularly high. Instead, just like the increasing number of emergency calls in the summer months, the average of sby_need rises particularly high in those same summer months. This endorses the conjecture stated in 2.2.2. The two variables sby_need and n_calls seem to have a strong correlation.

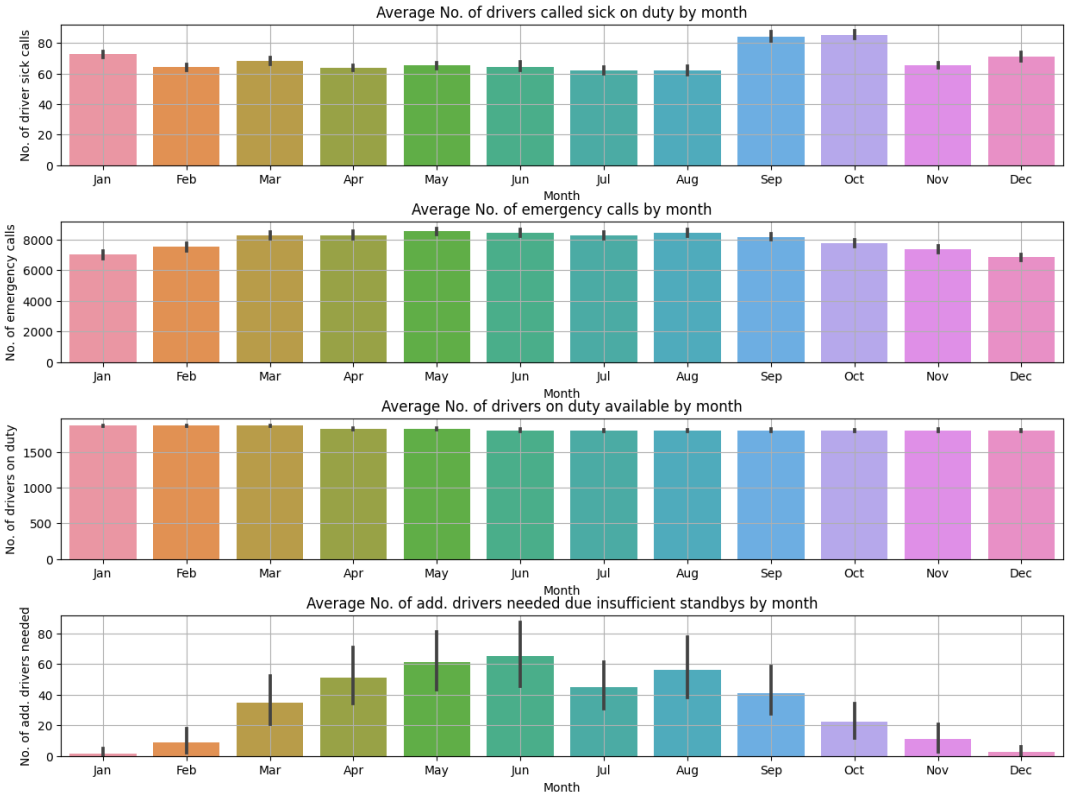

Figure 2: Variable Visualisation by Month

2.2.4 Data Visualisation by Weekday

Since the human working cycle is orientated around the work week, it also makes sense to observe the variables trend for each weekday. Immediately, we can identify that the average of n_sick is almost equal for all weekdays, apart from Saturday being slightly lower and Monday being slightly higher.

The average number of emergency calls show a decreasing trend with Monday being the highest number with around 8200 average emergency calls. On Sundays, on the number of emergency calls on average is at around 7200.

Similar to the last graph, the average of drivers on duty stays the same and only changes at the end of each year.

Noticeably, the average of sby_need also follows the same trend just like emergency calls. This once again supports the initial conjecture.

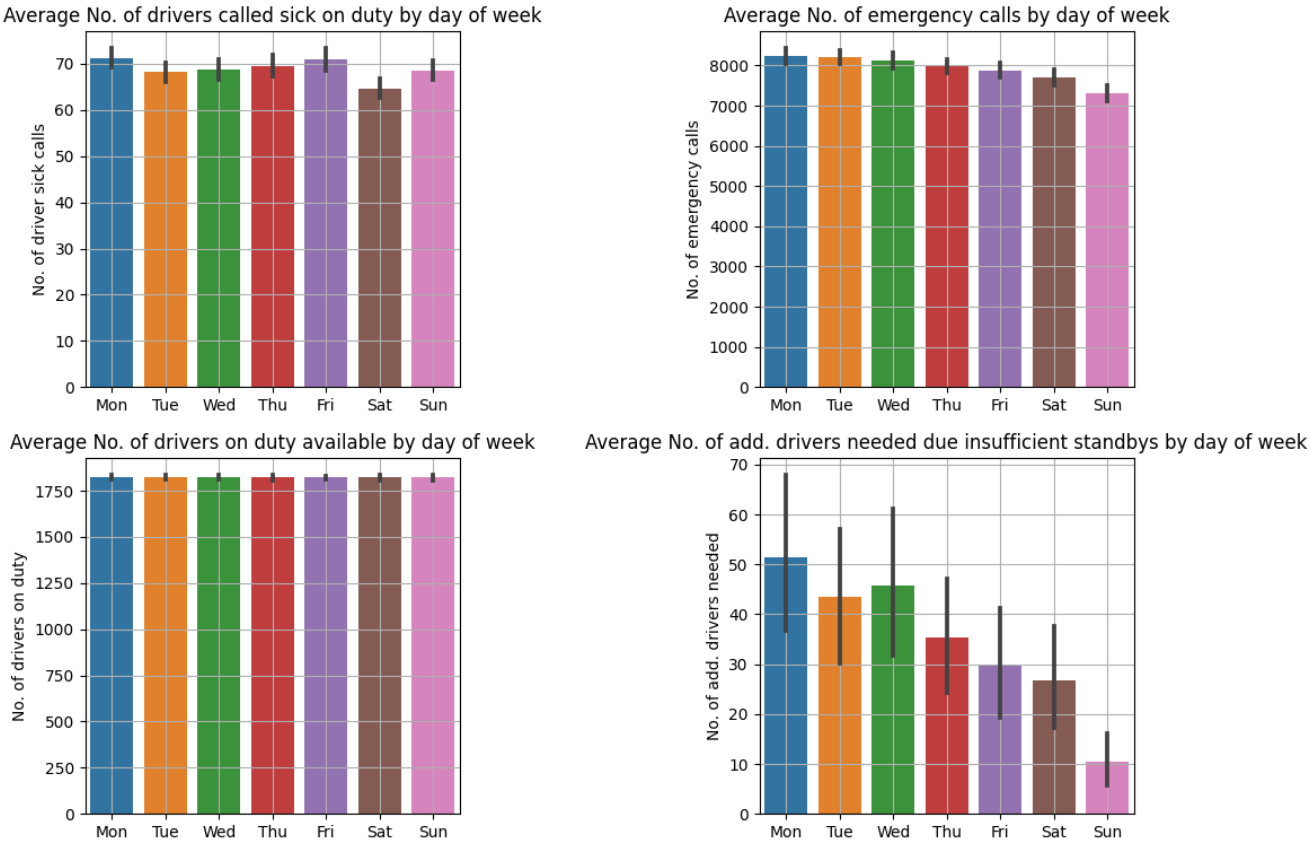

Figure 3: Variable Visualisation by Weekday

2.2.5 Data Visualisation by Month and Weekday

In the months graph, there was a prevalent difference in relevant variables between the different months. For this reason, it might be insightful to focus on the weekdays of the months with the greatest discrepancies.

January is the month with the lowest emergency calls. Although, the number of driver sick calls is relatively high, it still proves to have the lowest number of sby_need. On average, only on Wednesday and Friday, standby drivers were needed. Even on peak days, not even 50 standby drivers were needed. The average number of emergency calls does not exceed 8000 on any days in its recorded years.

June is the month with the highest average number of additional standby drivers needed and second average of highest emergency calls. In the months graph, the only month with more emergency calls appears to be May. This means that the amount of emergency calls does not directly affect the number of additional standby drivers needed but instead suggests that for example when a certain threshold was reached, then sby_need increases.

September is the month with the most driver sick calls on average. sby_need seems to be significantly lower than in June.

The initial conjecture still seems to prove itself. This reinforces itself with the following two graphs.

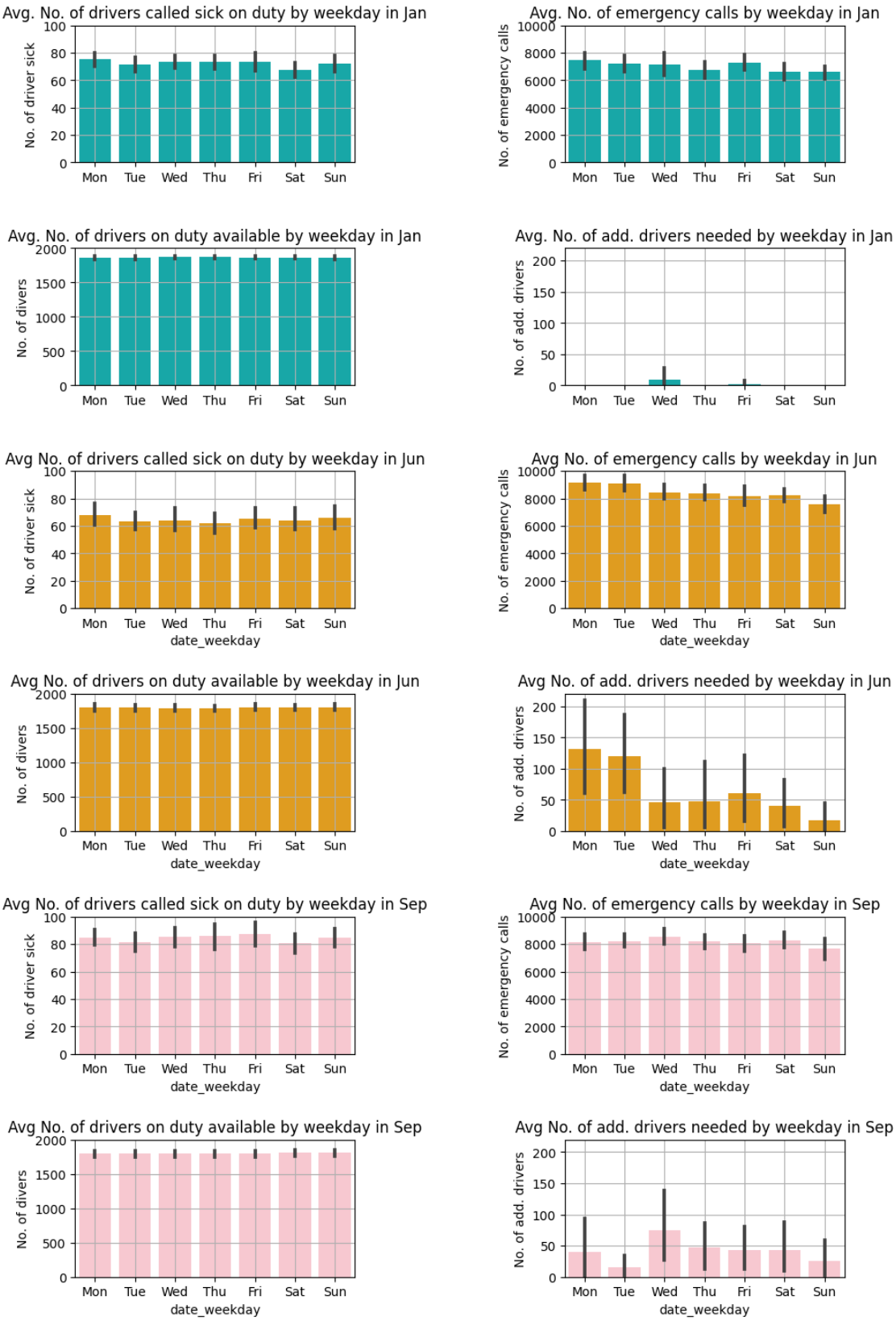

Figure 4: Variable Visualisation by Weekday (Selective Months)

2.2.6 Other Types of Visualisations

Scatterplot

Until now, we gained insight by looking into the most relevant data ordered after a certain time-series. The following scatter plot depicts all recorded instances ordered by the relation between emergency calls and driver sick calls. On the x-axis, driver sick calls are tracked, and on the y-axis, emergency calls are tracked. In addition to each point, a hue is added to indicate the number of standby drivers needed on that day. The darker and more purple the point gets; the more additional standby drivers were needed on that day.

Eye-strikingly, the majority of points with darker hue are located at the top of the graph, which indicates a higher number of emergency calls. This effect does not show itself at higher driver sick calls. The variable range of where the hue of the points begins to change color is about at 8500 to 9200. This varies for instances of different years. For 2016, the color change already begins at 8500 while for 2019, it barely starts at 9200. This could be related to the different number of drivers on duty. This indicates that a dependency between sby_need and n_calls is only existent after a certain threshold of n_calls was reached.

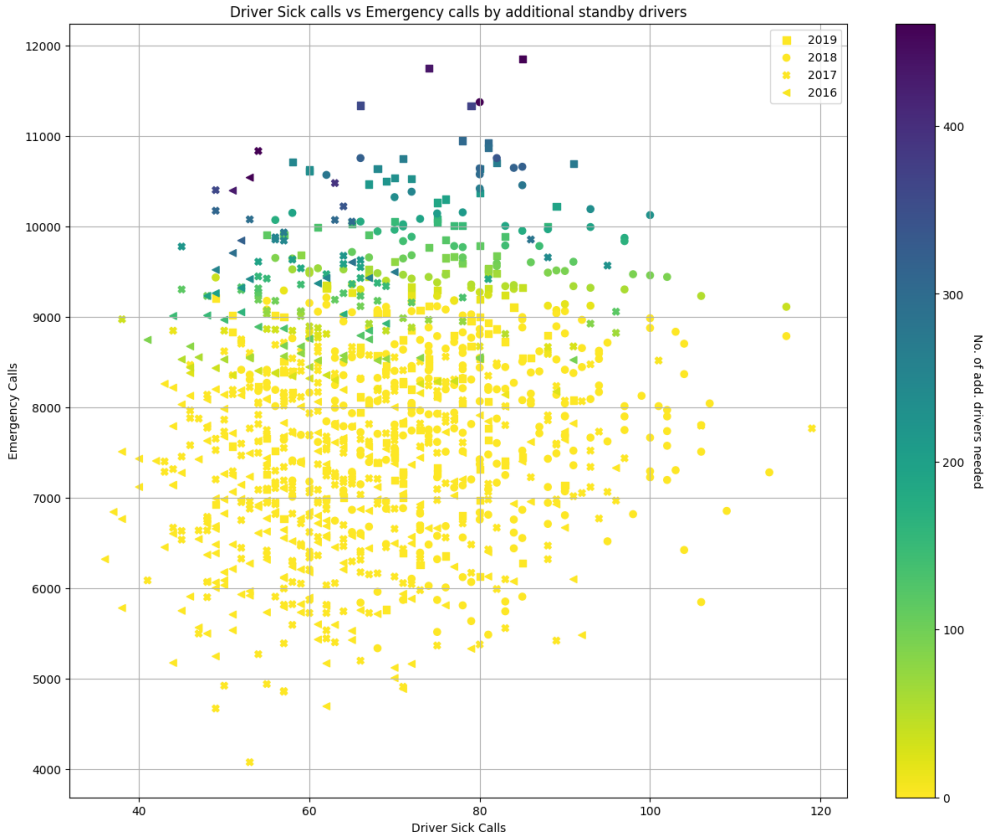

Figure 5: Scatterplot by Sick and Emergency Calls with sby_need hue

Correlation Heat Map

The following heat map depicts the mathematical correlation between the different variables. The strongest correlation of 67% is between the variable calls and sby_need. Logically, the more emergency calls the rescue drivers receive, the more occupied they are. Every rescue driver can only effectively accept a certain number of calls. If the number of emergency calls exceed the number of calls, that the rescue drivers can accept, then standby drivers will be needed. This proves that sby_need is dependent on the number of emergency calls.

Another strong correlation of 46% is between n_sick and n_duty. This of course also makes sense, due to more drivers being assigned to duty each year which once again means that proportionally more drivers will call in sick.

The last mentionable correlation of 36% is between n_duty and calls. Due to the fact of increasing emergency calls with each year, the overall number of on duty rescue drivers was increased which explains this vague correlation.

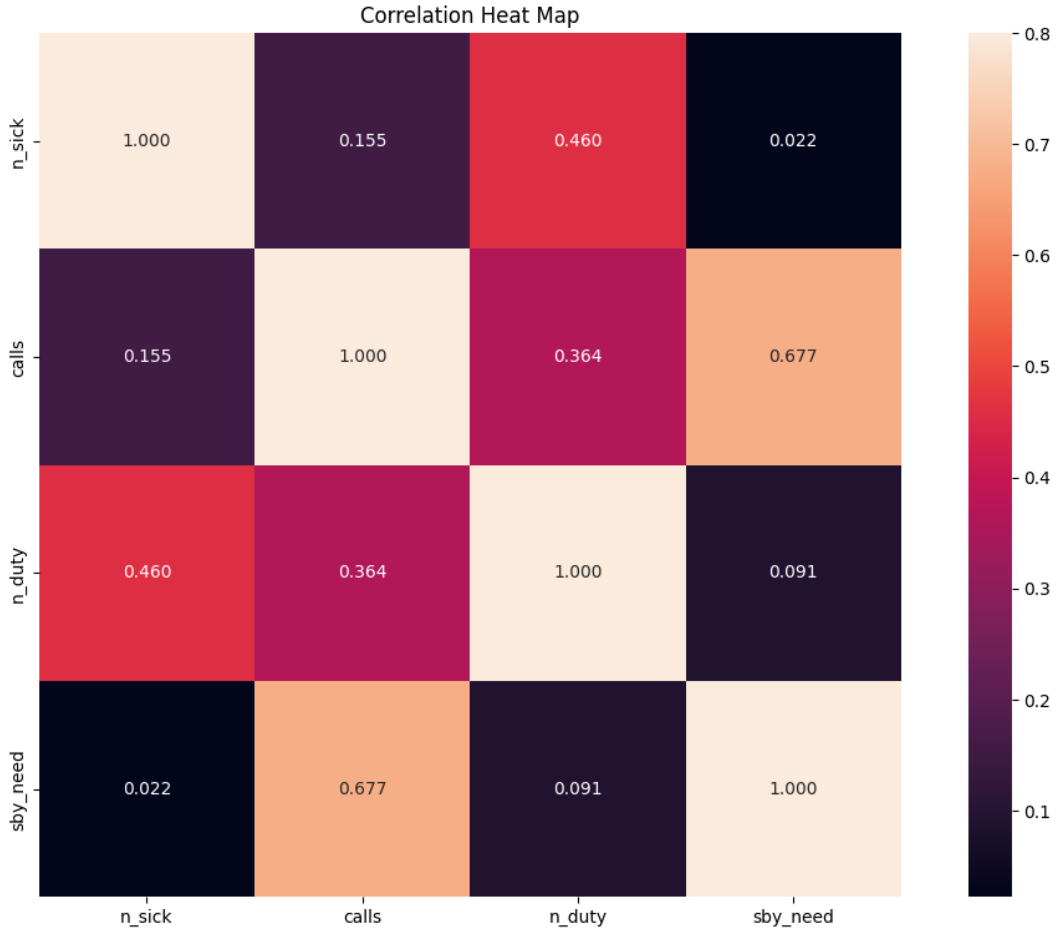

Figure 6: Correlation Heat Map of all relevant variables

2.3. Data Preparation

The next step involves selecting, formatting, and cleaning the data, ensuring their readiness for predictive modelling. Valuable insights from the previous step can be applied here. This step is pivotal, since a sensible outcome from most models relies on well-prepared data that suits the model's requirements.

The data quality of the given data set is exceptionally high with no missing or undefined values, and no missing entries.

```
<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
```

Figure 7: Variable data types

The initial data types of each variable are float64, int64 and string. All data types can be converted in int64 since all variables are integers apart from the date variable. This variable is converted into the more practical data type datetime64. Due to many existing libraries and frameworks which specialize in working with datetime64, this simplifies many previous/upcoming steps e.g., for visualization.

Next, the date variable is split into years, months and days which will be easier to work with during the upcoming steps. From the individual dates, the day of week as well as the month can also be extracted. This is relevant for the prediction as seen in Chapter 2.2.2. e.g., Mondays tend to require more sby_need due to higher emergency call rates.

Since the date variables are technically categorical variables, it is beneficial for many prediction models to convert them into binary variables due to some models not being able to process categorical variables. This is done with the so-called One-Hot-Encoding (Harris et al., 2021, p.129). The column week which contains the weekdays Monday to Sunday, will be converted into

seven individual variables which contain the binary information of each day. For instance, if the week variable indicated a Monday, the created binary variable weekday_Mon will have a 1 was an entry and all other weekday columns will have a 0 as an entry.

Furthermore, the data set is split into two portions - a training and a testing portion. The chosen ratio is 80 to 20 and usually done randomly (scikit-learn, 2023). This is important during the creation of the prediction model. In order to validate the prediction model, a portion of the data is withheld during training and used during model evaluation. This process ensures that the model didn't just memorize the testing data and results in an unbiased evaluation of the model.

Finally, for many prediction models it is beneficial for the data to be standardized (Marquardt, 2012). This means that the distribution of the variables has a mean of 0 and a standard distribution of 1. This can be implemented by utilizing the data science framework sci-kit learn.

2.4. Modelling

Once the data is prepared in the correct format, the modelling phase can begin. Here, one or more models can be selected and trained. Usually, the model with the best performance is ultimately chosen for the deployment in the actual application.

Since the target prediction variable is the number of additional standby drivers needed (sby_need) which is a continuous variable, regression analysis techniques must be utilized here. Since the given data set is relatively small (1152 entries), deep learning methods such as neural networks are ineffective due to requiring a large data set. For this reason, traditional machine learning methods are more adequate and will most likely give a better result.

In machine learning, there is a plethora of different methods and techniques with varying accuracy and success. For this case study, a handful of methods are tested out with default parameters. The ones with the highest accuracy are selected and focused on in further steps.

All models will undergo the same steps. First of all, the model is initiated with default parameters. Afterwards, the model is trained with the training data set which was split up beforehand. Subsequentially, it is then tested with the remaining testing data set to evaluate the result. The two evaluation metrics Mean Square Error and R^2-Score will provide a common ground for comparison. In Chapter 2.5., the model evaluation will be discussed more thoroughly.

The Mean Squared Error score of a model is the total average squared error of the predicted outcome and the actual outcome (Bickel et al., 2015). The smaller the score, the more accurate the model is.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2$$

Equation 1: Mean Square Error

The coefficient of determination provides a statistic of how well an observed outcome is predicted or replicated by a model (Hughes et al., 1971). It is calculated by subtracting the ratio the sum of squared residuals with the total squared residuals with 1. The resulting score has a range from 0 to 1 ($R^2 \in [0,1]$) with 1 indicating a perfect prediction.

$$R^2 = 1 - \frac{RSS}{TSS}$$

Equation 2: Coefficient of Determination

In order to be completely unbiased, all models will receive the same training and testing data set. Hence, the model evaluation will provide a direct comparison.

2.4.1 Ridge and Lasso Regression

Linear regression is used to analyze the relationship of the dependent variable towards one or more independent variables. In our case, the dependent variable is the target variable sby_need and the independent variables are the input variables. The linear regression algorithm attempts to find the best-fitting line/hyperplane that minimizes the ordinary least square between the actual and predicted values. When the regression line was found, new input variables can be inserted, and a predicted/estimated output is returned (scikit-learn, 2023).

A common technique to reduce overfitting it to add a penalty term. This process is called regularization. In (linear) regression, Ridge and Lasso are two common variants of regularized linear regression.

Ridge regression adds a penalty which is equivalent to the square of coefficients. This prevents overfitting and mitigates multicollinearity of variables in the data set.

Lasso regression adds a penalty term which is equal to the absolute of coefficients. This reduces variables with little or no impact on the model, effectively removing them.

For this model, the sci-kit learn library `sklearn.linear_model` was used.

Here are the parameters which were used in the model.

2.4.2 Support Vector Regression

Support Vector Regression is a machine learning method based on algorithm support vector machines. By finding the optimal hyperplane in a high-dimensional space that separates different classes, and when possible, maximizing the separating margin, the algorithm can apply classification to a certain dataset (scikit-learn, 2023).

In support vector regression, the task is not to find the optimal hyperplane which splits the data but instead to find the optimal which includes most points data and representing a mathematical function, usually the regression line. Unlike traditional regression, SVR does not aim to minimize errors for all data points but focuses on minimizing errors within a margin. Data points within the Margin/Epsilon-Insensitive tube are within an acceptable error and only points outside of it contribute to the loss function and are penalized. Optimally, the hyperplane maximizes the margin and minimizes the loss function for points outside of the margin tube.

In some cases, the not all data points can be covered comfortably within the margin tube due to non-linear behavior. In this case, a so-called kernel can be used to map the input data points into a higher dimensional function where the data point behavior is more linear and easier to cover with the margin tube.

For this model, the sci-kit learn library `sklearn.svm` was used.

Here are the parameters which were used in the default model.

Here are the parameters which were used in the optimized model.

2.4.3 K-Nearest Neighbour

K-Nearest Neighbour is a rather simple machine learning algorithm, in which a data point is predicted by taking the average of the nearest data points. A commonly used distance metric for regression tasks is the Euclidean distance. The number of points from which the average is taken

can be defined with the parameter k. This model is included in case, the relation between the variables cannot be covered with a regression model (scikit-learn, 2023).

For this model, the sci-kit learn library `sklearn.neighbors` was used.

Here are the parameters which were used in the KNN model.

2.4.1 Decision Forest

Decision Forests are ensemble machine learning methods for classification and regression which are based on a multitude of decision trees. The structure of a decision tree includes decision nodes, root nodes, branches, and leaves.

Decision nodes are followed by branches which represent a conditional statement (e.g., if-statement) which lead to either another decision node or a leaf node. The predicted result is represented by leaf nodes, which mark the end of a decision tree.

To balance out the decision tree's habit of overfitting, a variation, namely the random decision forest can be used, to increase the overall accuracy. A single decision tree may be a weak learner but a multitude of decision trees, namely a decision forest can enormously increase the model's performance. By utilizing the bagging method, bootstrap samples of the data are generated which are used to train and build n weak learners which in combination have a greater performance (TensorFlow, 2023).

In regression tasks, the output is the mean or average prediction generated by the individual trees.

For this model, the TensorFlow/Keras library `tensorflow_decision_forest` was used.

2.5. Evaluation

In the first step of evaluation, the models will be trained with all existing relevant variables and also evaluated based on that (IU, 2023). In the second step, only the dates are going to be used to train and evaluate the model. Here, developers and project collaborators can gauge the expected model performance, providing valuable insights into its capabilities. The model with the best performance is selected for deployment in the actual application.

Since there is no definite cut off value which determines a good R^2-Score, we are going to define that all R^2-Score that are above 75% are considered to be good (Duke University, 2023).

2.5.1 Ridge and Lasso Regression

Both variants of linear regression return a similar Mean Square Error as well as R^2-Score. Unfortunately, the R^2-Score is too low to be used in an actual operative application. Also, both algorithms return negative numbers, which is why all negative values are converted to 0.

	Ridge Regression	Lasso Regression
Mean Square Error	2544.34	2530.66
R^2-Score	0.32	0.33

Table 1: Metric Score of Ridge and Lasso Regression

The graphs below depict the actual sby_need of the testing data in orange vs. the predicted sby_need of the models in blue. While both models identify days with peaks, it fails to grasp the days with no additional standby drivers needed and still predicts there to be standby drivers needed. Overall, the model cannot capture the relation between sby_need and the other input variables.

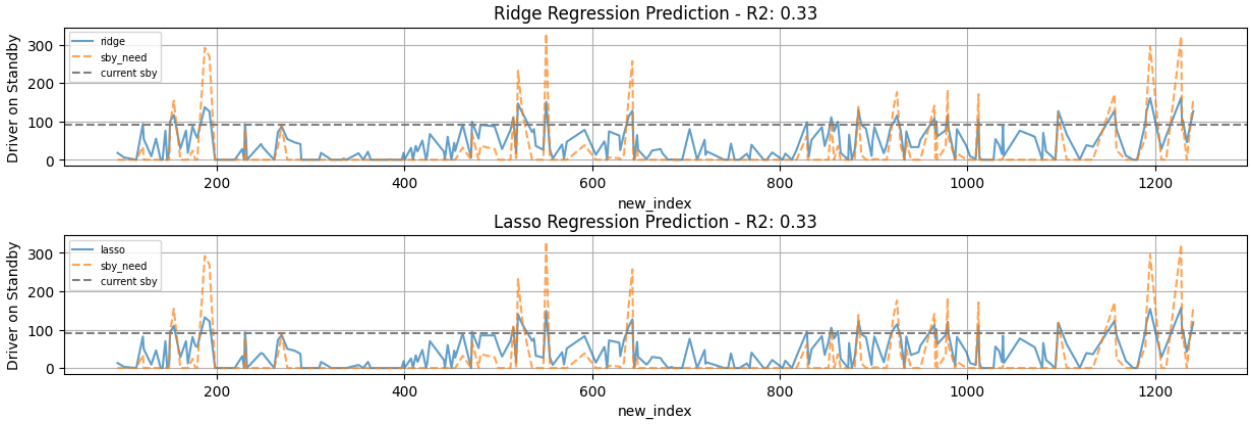

Figure 8: Graphical Overview of Ridge and Lasso Regression Fit

2.5.2 Support Vector Regression

For Support Vector Regression, one model was initiated with default parameters and one with optimized parameters found through GridSearchCV (scikit-learn, 2023). The model with default parameters has the lowest R^2-Score of all models. Thus, the default SVR fails in overall performance. On the other hand, the optimized model has a particularly high R^2-Score of 0.86, making this model the most performant model yet. Also, both algorithms return negative numbers, which is why all negative values are converted to 0.

	SVM (default)	SVM (optimized)
Mean Square Error	3363.32	527.47
R^2-Score	0.11	0.86

Table 2: Metric Score of Support Vector Regression

The graphs of the optimized model show promising results. All the peaks with high sby_need are captured very well. Compared to Ridge and Lasso Regression, which also captured the individual peaks, this model can also accurately predict a close to correct value for small and large peaks. In terms of days with no sby_need, it predicts a relatively small number compared to Ridge and Lasso Regression. This is neglectable since a certain number of standby drivers should be available every day.

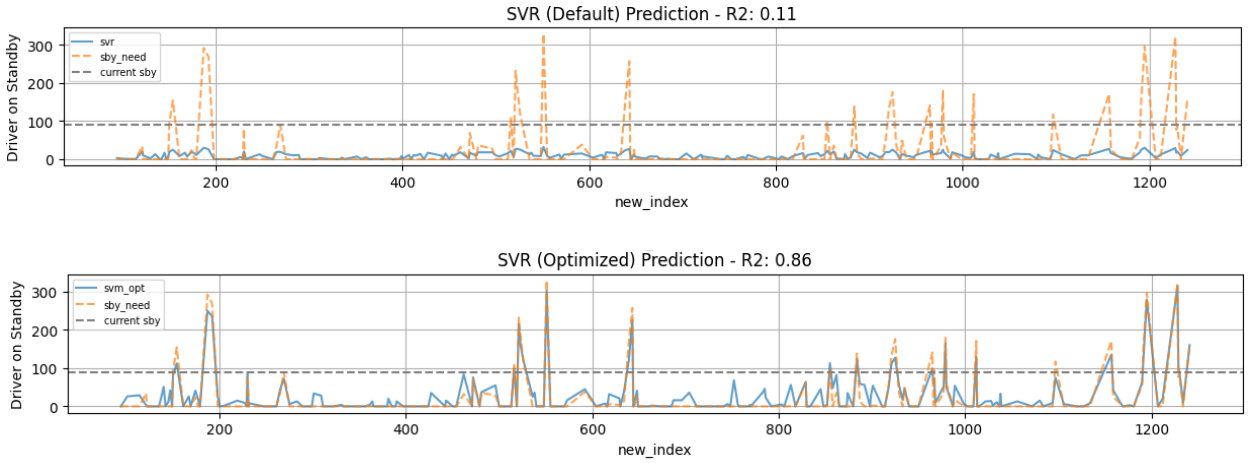

Figure 9: Graphical Overview of Support Vector Regression

2.5.3 K-Nearest-Neighbour

The K-Nearest Neighbour algorithm has a relatively low R^2-Score. This can be explained through the way the algorithm works. It forms averages between a certain number of points which does not represent the data pattern in out use-case.

	KNN
Mean Square Error	2852.91
R^2-Score	0.24

Table 3: Metric Score of K-Nearest-Neighbour

While all other algorithms and models somewhat at least captured the peaks of the test data set, the K-Nearest Neighbor algorithm fails to do so. This means that the behavior of the underlying model cannot be accurately represented with a K-Nearest-Neighbour model.

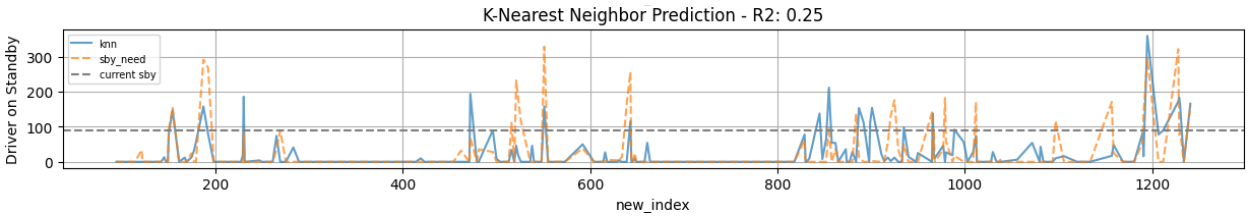

Table 4: Graphical Overview of K-Nearest-Neighbor

2.5.4 Decision Forest

At last, the TensorFlow Decision Forest shows the overall best performance of all models. With a Mean Square Error of 121.87 and an R^2-Score of 0.96, the model is very accurate in predicting the number of additional drivers needed.

	TFDF
Mean Square Error	121.87
R^2-Score	0.96

Table 5: Metric Score of K-Nearest-Neighbor

The model accurately predicts the peaks in the dataset but also predicts the days with 0 sby_need. For this reason, this model will be used and focused on in further steps.

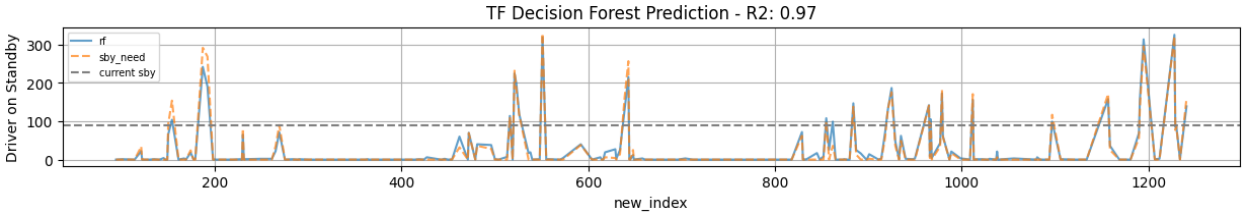

Figure 10: Graphical Overview of TensorFlow Decision Forest

2.5.5. Exclusive Prediction with Date variables

While the random forest model does extremely well with all information available during the prediction, in the reality, the two most important variables n_sick and calls are only known being reported after the shift. Without those two variables, the model behavior changes immensely, with the prediction accuracy reducing extremely. As a result, we have the following metrics:

	TFDF – Date only
Mean Square Error	4677.33
R^2-Score	-0.23

The negative coefficient of determination already reveals the most relevant fact. A negative R^2-Score means, that the prediction does not follow the trend of the data and thus does not actually predict a meaningful value for sby_need.

2.6. Deployment

In this step, the application is finally deployed to the real system. After being proofed on bugs and other errors, it will be integrated to a running system. Since the application does not require much computational power, it can run on the regular servers of the Red-Cross in Berlin. In order to make the prediction model even more accurate in the future or more robust against sudden changes and model drift, constant new data for training is added to the model.

The model either undergoes a completely new training with the new data or transfer training can be applied to the model. In case, the model's performance deteriorates, or bugs appear in the program, it is sensible to apply version control (Shearer, 2000). By doing so, it is always possible to go back to a running version. Also, a periodical monitoring system for detecting subtle model degradation is important, so changes and interventions can be made before false or suboptimal predictions affect the everyday work.

At last, the model and its corresponding code should be well documented, so that the model can be maintained and further developed even once the original developer or data science consultants leave the project. This could also be the basis for replicating the project for other Red Cross locations in Germany.

Next, an exemplary GUI as well as Git repository is shown how it could be deployed.

2.6.1. Exemplary Graphical User Interface

In everyday work, the application could be in form of a simple Graphical User Interface, as shown below. The employees of the HR Department simply have to input the startdate from when they want their prediction and the end date. This would output a plot where the predicted number of standby drivers are needed. For reference with prior years, there is an option to display the dates from years before. With a drop-down menu, the employees can simply choose a year which they want to compare.

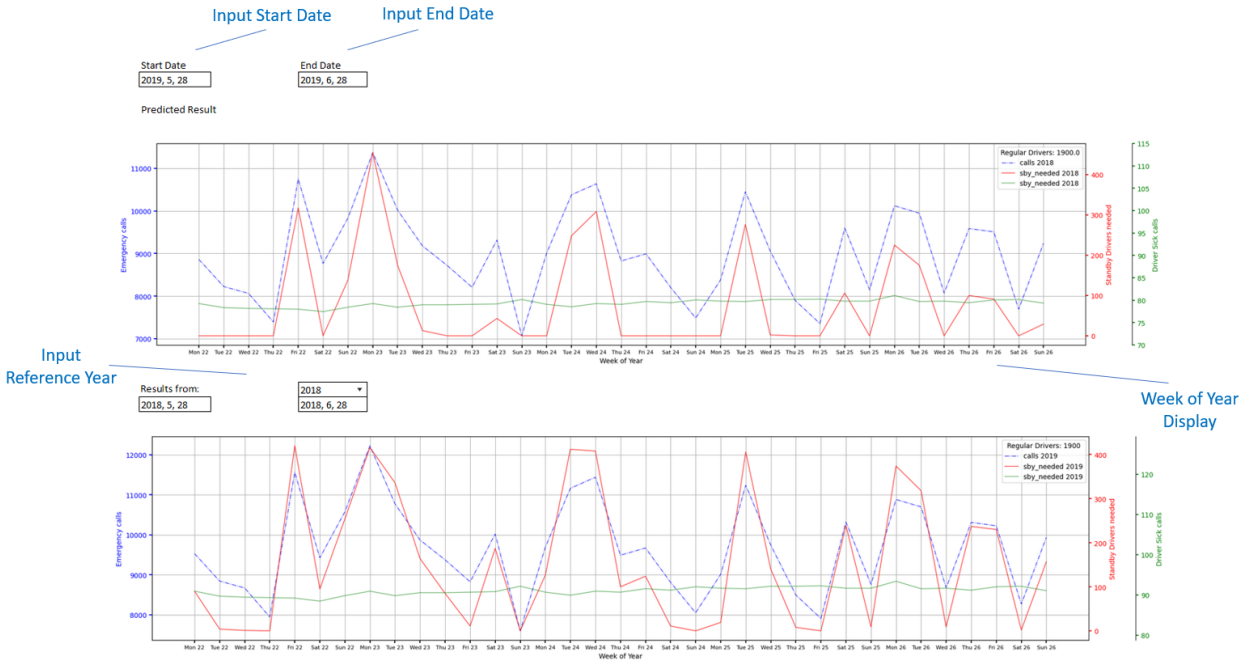

Figure 11: Exemplary Graphical User Interface

Also, a table with the definite number of standby drivers is also returned. With this they can easily manage the standby driver duty plan.

	date	n_sick	calls	n_duty	sby_need	weekofyear
0	2019-04-28	66	10463	1900	237	Sun 17
1	2019-04-29	72	8891	1900	2	Mon 18
2	2019-04-30	70	9933	1900	161	Tue 18
3	2019-05-01	68	9722	1900	111	Wed 18
4	2019-05-02	67	9439	1900	75	Thu 18

Table 6: Exemplary Table of Prediction Outcome

2.6.2 Exemplary GIT Repository

The entire project would also benefit from a private GIT repository, enabling a distributed version control system for coordinated work amongst programmers (the repository in the documentation is public). Future changes and versions can be managed through this repository by authorized personnel. The proposed structure of the GIT repository is as follows:

```
ModelEngineering
↳readme                 - short overview of the project
↳license                - license file how to use the code
↳docs
  ↳requirements         - requirements documentation
  ↳application_doc      - actual documentation of the application & code
↳data
  ↳sickness_table.csv   - unprocessed input data
  ↳data_processed.csv   - processed input data
↳src
  ↳ml                   - source code files of the machine learning
  ↳gui                  - source code files of the GUI
  ↳other                - source code files of other components of the app
↳tests                  - unit test and integration test
↳model_data
  ↳config               - configuration and hyperparameters of the ML model
  ↳model                - the actual ML model
↳results                - prediction results
↳changelog              - change log where all the changes are documented
```

3. Conclusion and Summary

3.1. Error Analysis

The current version of the model undoubtedly has a high prediction accuracy with its current available information. However, the two most important variables n_sick and calls are in most cases only known after the end of a shift where the results are reported back to HR.

In the current model, the two variables n_sick and calls have great relevance for the prediction accuracy of the model. By leaving out these two, the model accuracy drops immensely. For the current solution, the model takes historical values (multiplied with an update factor) for n_sick and calls and uses them to predict the number of sby_need. In reality, these two values are more or less unforeseeable. N_sick and calls are likely to have a seasonal influence and could be replaced by taking the exact date (with the exception for the year) for each prediction instead of the month and day of week approach. Another relevant variable that could be introduced into the model are holidays and special days in Berlin, where emergency calls have a different trend e.g., Christmas.

By implementing these changes into the next version, the prediction application could increase in usability. Depending on its performance, either the old version can be brought back, or the new version is continued to be used.

3.2. Conclusion

CRISP-DM in its individual phases acts as a bridge between developers and stakeholders, making sure that their goals and understanding are at the same level. By utilizing the CRISP-DM methodology, it is guaranteed that all important points of a data science project are taken into consideration. It emphasizes that all six phases have an equal status within the project. By leaving out even a single phase, the project could fail, and countless hours of work could be in vain.

The six steps business understanding, data understanding, data preparation, modeling, evaluation, and deployment act as a guideline throughout the entire project. In a more complex use case in real life, the entire methodology undergoes an iterative process since it's almost impossible to achieve a satisfactory result in the first try.

Through CRISP-DM, we have learned that we cannot utilize the entire information that was given to us. At first glance, it seems like using n_sick and calls as feature variables for model training and prediction would result in an easy and robust model. However, by analyzing the business use-case, it becomes clear that those two variables are only afterwards known and thus cannot simply be used in the actual prediction. In a longer project, that would result in an iterative process. In conclusion, this realization endorses the usage of such methodologies in data science applications.

Bibliography

Chapman, P. (1999). *The CRISP-DM User Guide. https://s2.smu.edu/~mhd/8331f03/crisp.pdf*

Shearer, C. (2000). *The CRISP-DM model: the new blueprint for data mining. J Data Warehousing; 5:13-22.*

IBM (2021). *CRISP-DM Help Overview. https://www.ibm.com/docs/en/spss-modeler/saas-?topic=dm-crisp-help-overview*

IU International University (2023). *Tasks for Course: DLMDSME01 – Model Engineering.*

Harris, D., Harris S. (2012). *Digital Design and Computer Architecture. ISBN 9780123978165*

scikit-learn.org (2023). *sklearn.model_selection.train_test_split. https://scikit-learn.org/stable/-modules/generated/sklearn.model_selection.train_test_split.html.*

Marquardt, D. (2012). *You should standardize the predictor Variables in Your Regression Models. https://doi.org/10.1080/01621459.1980.10477430*

Frost, J. (2023). *When do you need to Standardize the Variables in a Regression Model. https://statisticsbyjim.com/regression/standardize-variables-regression/*

Bickel, P., Doksum, K. (2015). *Mathematical Statistics: Basic Ideas and Selected Topics. Vol. I. ISBN 013850363X, 9780138503635. https://doi.org/10.1201/b18312 - MSE*

Hughes, A., Grawoig, D. (1971). *Statistics: A Foundation for Analysis. ISBN 0201030217, 9780201030211. – Coefficient of Determination*

scikit-learn.org (2023). *sklearn.linear_model.Ridge. https://scikit- learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html*

scikit-learn.org (2023). *sklearn.svm.SVR. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html*

scikit-learn.org (2023). *sklearn.neighbors.KNeighborsClassifier. https://scikit-learn.org/stable/-modules/generated/sklearn.neighbors.KNeighborsClassifier.html*

TensorFlow (2023). *TensorFlow Decision Forests. https://www.tensorflow.org/decision_forests*

Duke University (2023). *What's a good value for R-Squared? https://people.duke.edu/~rnau/-rsquared.htm*

List of Figures

List of Tables

List of Equations

Attachments

Github-Link: https://github.com/t0wgster/Repo_ModEng