A Python Package for Nextbike Data: Transformation, Duration Prediction and Direction Classification

Lukas Humpe 7363827
Philipp Page 7367085
Michael-The-Lan Bui 7364116
Tim Schäfer 7369518

Executive Summary

This report covers an exploratory and predictive analysis of a nextbike data set for Mannheim. In the following, the data is visualized, which illustrates consumer behaviour, such as rental duration and popular places. The result of the exploratory analysis shows that there is an increase in driving behaviour at city events as well as increased usage at certain stations near to popular places in the city. Finally, the duration of bike rentals as well as the direction is predicted. Direction prediction indicates whether the trip goes towards a university or away from a university. The direction classification accuracy is ~52%. The duration model achieves a mean absolute error of ~1400 seconds on the holdout set.

Table of Contents

List of Figures

List of Tables

1. Introduction

Today, where the idea of sustainability has already arrived in society, people think and act much more consciously and deal differently with commodities. To achieve sustainability, however, it also requires environmentally conscious action. Especially, the exhaust emissions of cars are a critical issue for society, so that the reduction of these is one of the major goals to achieve sustainability. In big cities, cars are more of a nuisance than a help, which means that bicycles are becoming more fashionable. Firstly this is due to the fact that flexibility can be gained. In addition, the carbon footprint is much lower compared to other popular alternatives. However, the purchase of a bicycle is often connected to high monetary means, so that a purchase with a moderate use is superfluous. Many ventures saw the opportunity to rent bikes and offer consumers access to flexibility for little money. It is a concept of the Sharing Economy (SE) in which users with an account can borrow bicycles for as long as they like and park them in valid zones without the need to own the bikes themselves or taking care of maintenance for them. Since this form of rental can result in a more conscious consumption and the sharing of consumer goods enables a more effective resource allocation research in this area can be considered highly relevant.

As nextbike is very interested in predicting bicycle usage behaviour, this report takes a closer look at the circumstances. The city of Mannheim is analysed in more detail based on given data. For the analysis, a data exploration and prediction is presented. The focus of data exploration is the visualization of past data to derive patterns and trends. The prediction part considers the forecasting of bike rental duration as well as a classification of direction. The report is divided into six chapters. The first chapter considers the concretization of the project goals and describes the used methods. The following chapter deals with the data exploration that reflects the behaviour of users. In the third chapter, predictions are made on the behaviour of borrowers under different aspects. The fourth chapter discusses the results by underlying the limits and the scope of this report. Finally, the results are presented in a summary.

2. Project Goals

The procedure of identifying and formulate the underlying project goals has been divided into two parts. In the first section, the problem is identified and described. Subsequently, the method, which leads to different functional and non-functional requirements, is outlined.

2.1. Problem Description

A bicycle rental company faces many management challenges. In the foreground are the optimal utilization of bicycles and a differentiation from the competition, which the company wants to offer to the customers. The SE concept behind bicycles brings, on the one hand the flexibility that bicycles can be found practically everywhere, but on the other hand, the flexible allocation may cause, that the demand in hot spots is not met. To face these problematic issues an understanding of rental patterns is required so that, based on the result, a validated prediction for bike duration and direction for efficient re-allocation of bikes can be made. Thus, to enable optimal use with the goal of customer satisfaction.

This analysis is therefore to support companies facing the problem allocating bicycles to locations so that the demand can be satisfied in the best possible way. To achieve an optimal utilization of bicycles the following questions are to be answered. Where are most bicycles parked? When are the trips made? Where do the trips take place? To answer these questions, this report makes an initial recommendation on how nextbike can better allocate its bicycles to meet the needs and demand for bicycles to maximize revenue.

2.2. Method

The method used in this project is inductively derived from the problem description to assure that business requirements are met in the most precise manner. This goes in line with the Cross Industry Standard Process for Data Mining (CRISP-DM) framework which defines business and data understanding as initial steps in the life-cycle of a data mining project (Wirth, 2000). All further chapters of this report can be mapped to this framework. The following presents functional as well as non-functional requirements for the developed Python package and therefore deals with the phase of business and data understanding.

Before being able to address the business problems in detail it is necessary to pre-process the data and translate it to a usable format, which forms the first functional requirement. Hence, the package implements a robust and automated pre-processing strategy. Resilience against corrupted input data and output format validation is used to assure the consistency of the data. These kind of validation steps are crucial since

the resulting data set forms the basis for every following step of the analysis. This whole process was accompanied by a visualization of relevant metrics to identify e.g. inconsistencies or irrelevant data points.

The second functional requirement is the ability to conduct trip duration prediction and direction classification indicating whether a trip is expected to go towards a university or away from a university. To realize this requirement, the package implements the respective predictive models utilizing the Python `scikit-learn` package (Pedregosa et al., 2020). Additionally, models can be automatically saved to disk after training for later usage, in cases where the prediction of new data is needed. To assess the performance of the trained models it is also possible to access different metrics regarding their training performance.

Even though the package itself does not implement functions in regards to visualization, a transformed data set is used to visualize relevant metrics as well as the underlying dynamics of the data in a Jupyter Notebook since this is more interactive. All relevant findings and plots are included in this report.

Apart from the functional requirements, i.e. those reflecting immediate business value, the following three non-functional requirements have been identified:

1. Implementation of a self-descriptive and easy to use command-line-interface (CLI) allowing access to the functionality the package exposes.

2. Installation and configuration without the need to setup complex environments or edit the source code.

3. Extensibility and maintainability to support further development and simplify utilisation and contribution by other researchers and the open source community.

These non-functional requirements have been kept in mind during the whole development process and helped with technical design decisions. For example, the structure of sub-packages has been kept as easy as possible (not more than 2 layers; minimal and only non-circular dependencies between packages). Moreover, all imports use absolute paths[1] and each sub-package exposes its interface explicitly through its `__init__.py`. The automated and easy installation is realized through the `pip setuptools` (Python Packaging Authority, 2006) which specify installation instructions and dependencies in a `setup.py` file. The final package structure is illustrated in Appendix A.1, Figure 15.

[1]This is also recommended by the PEP-8 style guide (van Rossum et al., 2001)

3. Data Exploration

As the business understanding has been clarified in Section 2, the next step according to the CRISP-DM process model is data understanding and preparation. This section does provide a profound data understanding by outlining relevant meta-information about the data set as well as explaining the pre-processing procedure in detail.

3.1. Meta Information

The data was obtained from and provided by the nextbike application programming interface (API). It contains information about booked bicycles of the company regarding the city of Mannheim and the surrounding areas. The file of the data set is in .csv format and has a size of 338 MB. It contains 2,722,759 data points. Each row of the data set represents an event related to a booking. A booking contains information like date or time, bike number, or coordinates of a parked bike. The bookings refer to the year 2019 and start from 20/1/2019 and go until 31/12/2019. However, the data of complete July is missing. Table 7 in the Appendix A.2 shows the available information contained input data set which has been used as fundamental basis for the pre-processing part.

3.2. Pre-Processing

The pre-processing procedure can be divided into two major phases, data cleansing and data transformation. While the former of both deals with the raw data set, the latter is a translation into a new data format which is suitable for further processing, e.g. during the predictive analysis.

3.2.1. Data Cleansing

A first observation during data cleansing was that the raw data set contained many duplicate rows (1,087,228 duplicates of 2,722,759 total instances). To achieve better performance, these have been dropped by the minimal unique feature combination, `b_number` and `datetime`. Though, a drop over all features yields the same result.

After dropping redundant rows, the columns have been inspected, which revealed inconsistencies and redundancies as well. In contrast to the expectation that `p_uid` and/or `p_number` are unique identifiers for the place name `p_name`, they do not have the same distinct number of occurrences and are therefore ignored in further steps. Besides this fact, the columns `p_place_type` and `p_spot` explain each other perfectly. More precisely, `p_place_type` is always equal to `12` if `p_bike` is `True` and else `0`. Unfortunately, this is not the case for the column `p_spot`, which is slightly different, and therefore cannot be consistently interpreted. In consequence, only `p_place_type` is considered in further steps.

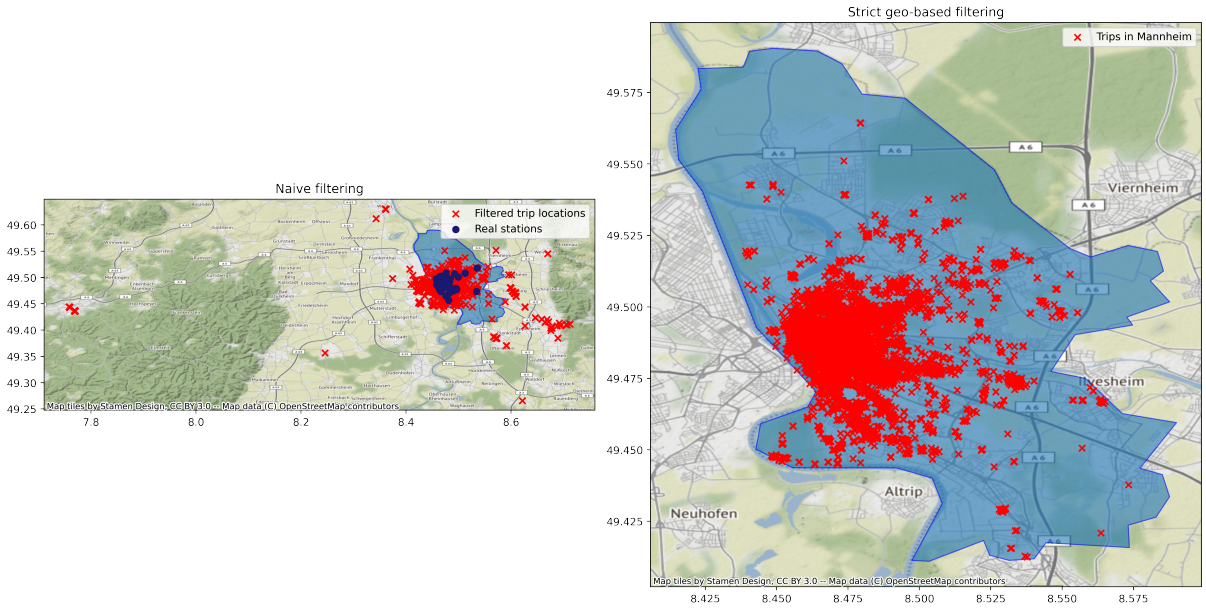

Figure 1: Naive Filtering Strategy (left) vs. Geo-Based Filtering Strategy (right)

Besides these technical inconsistencies, content-related issues have been identified as well. While inspecting the coordinates of bikes it became clear that the data set did not only contain trips within Mannheim but also trips in different cities in Germany as well as in different countries.

A first approach to handle this issue was filtering out all trips which did not use bikes owned by the city of Mannheim or did not end or start at nextbike stations in Mannheim. Therefore, the official bike status data set by Mannheim has been used (nextbike GmbH, 2019). Even though this approach increased the consistency of the data dramatically, it could not filter out all trips outside of Mannheim. One explanation is that users are allowed to drive to other cities with a bike according to the nextbike policy (VRN, 2020) or that some bikes have been re-allocated by the provider to other cities for e.g. maintenance.

To address this problem, a more strict approach has been applied, namely filtering based on the geo-boundary of Mannheim. This has been realized with the Python packages `shapely` (Gillies et al., 2013) and `geopandas` (GeoPandas developers, 2019) which provide convenience functions to deal with geo-data. The consistent data set, at least concerning the geographic location of bikes, has then been created by removing all trips whose coordinates are outside of Mannheim. Figure 1 illustrates the resulting bike locations of both approaches. One observes that only the second approach yields a completely consistent result according to the bike location.

A problem, also resulting from the removal of trips based on the location, is a mismatch between the number of bookings of type `start` and `end` in the data set (see Table 7, column `trip`). This means that there might be start bookings without a corresponding end booking and vice versa. Therefore, Algorithm 1 has been designed which fixes this inconsistency by removing end trips without a start trip and start trips without an end

trip. Please note that rows where the `trip` column equals “first” or “last” can be safely omitted since they have no additional value for calculating e.g. trip duration (cf. Section 3.2.2). The actual implementation of this algorithm in Python uses exclusively built-in `numpy` and `pandas` functions which ensures a high performance since these libraries implement their major algorithms in the C programming language.

Algorithm 1 Pseudo code for a cleansing algorithm to remove the mismatch between start and end trips consistently.

Require: Original *data* without *data.trip* = "first" or *data.trip* = "last" rows

```
1:  function FixStartEndBookingMismatch(data)
2:      data ← Sort(data) by data.b_number, data.datetime
3:      trips ← data.trip
4:      bike_numbers ← data.b_number
5:      delete_indices ← ∅
6:      for i < Size(trips) − 1 do
7:          if trips[i] = "start" and trips[i + 1] = "end"
              and bike_numbers[i] != bike_numbers[i + 1] then
8:              Add(delete_indices, i)
9:              Add(delete_indices, i + 1)
10:         end if
11:         if trips[i] = trips[i + 1] then
12:             if trips[i] = "start" then
13:                 Add(delete_indices, i)
14:             else
15:                 Add(delete_indices, i + 1)
16:             end if
17:         end if
18:     end for
19:     if trips[Size(trips) − 1] = "start" then
20:         Add(delete_indices, Size(trips) − 1)
21:     end if
22:     data ← Drop(data, delete_indices)
23:     return data
24: end function
```

Ensure: *data* is consistent concerning start and end bookings

Algorithm 1 covers two cases, a “special” case in the if clause in line seven and a “regular” case in the if clause in line 11. The regular case simply checks if two consecutive rows have the same trip type. This would mean that one of them has no corresponding end or start booking. For example, if two consecutive start trips occur, that would mean that the affected bike is rented again before the previous rental finished. Here, the first start trip has to be omitted because the rows are sorted by `datetime` within each group of bike numbers. For consecutive end trips, the procedure is similar, but instead of omitting the first end trip, the second one is omitted because a booking which has already ended cannot end again without a start trip in-between.

The regular case only fixes the mismatch between start and end trips but misses a semantically important aspect. A trip of bike `A` should not end with a "start" booking while a trip of bike `B` should not start with an "end" booking. This is caught by the special case in line seven which checks exactly this condition and marks both entries for removal. Remember that bookings are first sorted by `b_number` and then, within each group of bike numbers, by `datetime`. This ensures that the last booking of one group of bikes is really the last booking according to time.

Unfortunately, due to the comparative structure of the algorithm, it cannot catch if the last record of the data set ends with a start trip. This is covered in line 19.

3.2.2. Data Transformation

The cleaned data set from Section 3.2.1 is the input for the data transformation. Since it only contains bookings of type "start" and "end" it is straightforward to calculate actual trips containing information like e.g. start location, end location and duration. One possibility is to split the data set into two data sets containing only start and respectively end bookings. This is always possible because the number of rows is even. Then, one can calculate the trip information by using the corresponding start and end bookings of the two data sets. Table 1 documents the calculation of the transformed data set. `start_df` is the data set containing only trips of type "start" and `end_df` the data set containing trips of type "end". All calculations are element-wise vector operations.

Column Name	dtype	Calculation
`bike_number`	int64	Take `b_number` column of `start_df`
`start_time`	datetime64[ns]	Take `datetime` column of `start_df`
`weekend`	bool	`True` if `datetime` in `start_df` is at weekend, else `False`
`start_position`	geometry	Take `geometry` column of `start_df`
`start_position_name`	object	Take `p_name` column of `start_df`
`duration`	int64	Difference of `datetime` in `end_df` and `start_df` in seconds
`end_time`	datetime64[ns]	Take `datetime` column of `end_df`
`end_position`	geometry	Take `geometry` column of `end_df`
`end_position_name`	object	Take `p_name` column of `end_df`
`is_station`	bool	`True` if `p_place_type` equals `12` in `start_df` and `end_df`, else `False`

Table 1: Calculation of the Transformed Data Set.

In addition to the features specified in task 1c of the semester project, the dummy variable `is_stations` has been included as well. The team identified it as a promising feature for later analysis since it distinguishes free-floating trips from trips that end and start at fixed stations.

3.3. Implementation

The relevant part of the developed Python package is the `preprocessing` sub-package (cf. Appendix A.1, Figure 15). It contains four files each exposing one class written in the

object-oriented class syntax of Python using type hinting (Python Software Foundation, 2020b). It has been proven useful to use this programming style for this project since responsibilities are clear cut. Thanks to type hinting, each method is also able to define the expected type for each parameter which helps the developers to write clean code by being explicit about type choices.

In addition to the class syntax and type hinting, the `preprocessing` package implements an abstract base class using the built-in `abc` package in `AbstractValidator.py` (Python Software Foundation, 2020a). It forces child classes to implement a validation method which can be used as a post-hook after execution of business logic. Both, the `Preprocessor` class and the `Transformer` class implement this method to validate whether the pre-processing and the transformation yield a consistent data set. This technique enables the usage of user-friendly error messages, e.g. when using the CLI. Further technical details of the `Preprocessor` and `Transformer` class are not discussed in detail for the sake of brevity.

Since the geo-filtering is a bottle-neck regarding performance, it has been tested whether the filtering is faster when using multi-processing. However, on a test machine with an Intel Core i9 processor with 8 physical cores, the overhead for creating 8 new processes and passing the sub-data frame to it is heavier than the performance gain. Therefore, the CLI uses the single-process `Preprocessor`. Though, the implementation for the multi-processing version can be found in the `ParallelPreprocessor` class for reference.

4. Descriptive Analytics

The following section aims to illustrate differences of demand in regards to region, time and other event based data. It therefore contributes to the phase of data understanding and modeling. In order to get a first impression of consumers' driving behaviour, the average duration per month, per weekday and per hour was visualised. A comparison between weekend and weekday clearly shows in Figure 2 that the average driving time at the weekend was longer than during the week.

Looking at the duration distribution in Figure 16 in Appendix A.1, it can be seen that there are many bookings longer than 24 hours. Using the z-score, bookings over 10,000 seconds are marked as outliers and can be removed (Kannan et al., 2015). In addition, there are many bookings less than 180 seconds, as shown in Figure 16 in Appendix A.1. After consulting the official FAQ from VRN-Nextbike (VRN, 2020) it became clear that these were bookings in which the customer was not able to successfully rent the bike.

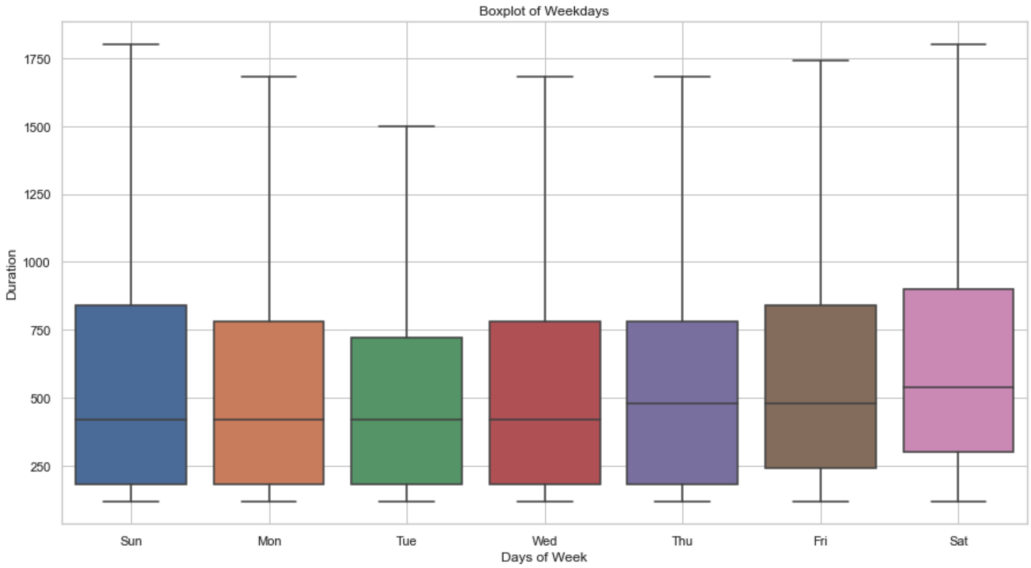

Figure 2: Statistical Quantities of Duration Divided into Weekdays

It was particularly noticeable that journeys last longer in the morning than during the rest of the day. The boxplot shown in Figure 17 in Appendix A.1 illustrates the results. With regard to the duration at weekdays, it can be stated that the longest trips on average take place on Saturday. The average journey length increases slightly from Tuesday to Saturday on a daily basis. The longest trips take place between Friday and Sunday. When comparing the duration aggregated by month, Figure 18 in Appendix A.1 shows that the spring months of April and May have a longer duration than the summer months.

In order to visually display the assignment of started trips to postal codes in the summer months, a polygon was laid over the city of Mannheim which then was divided by postal code. The started and ended trips are summed up and assigned to the postal codes.

Figure 3 shows the result whereby the polygon is subdivided in his postal codes to get a clear overview regarding the quantity of bookings with a geographical assignment.

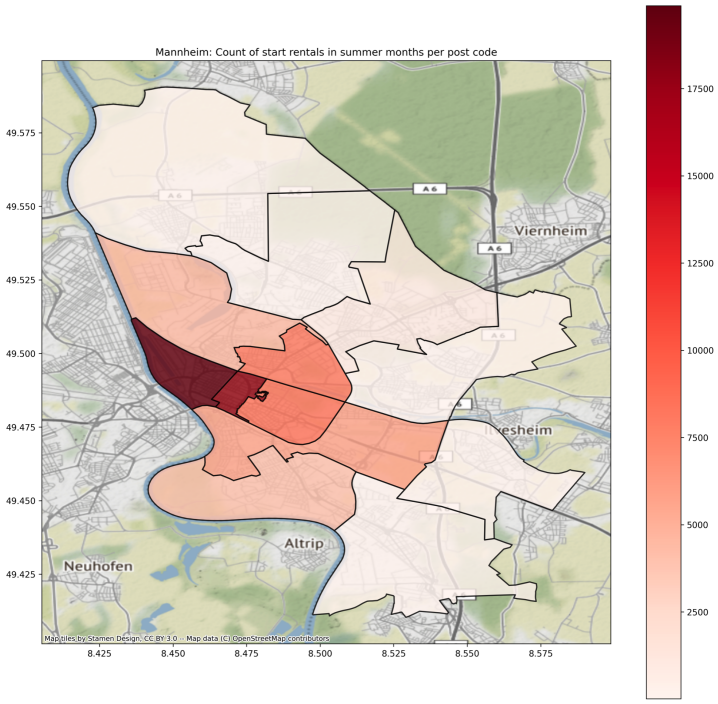

Figure 3: Representation of the Start of Rent Per Postal Code

The results clearly show that most of the started and ended trips start from the city centre. In contrast, the use of bicycles on the outskirts of the city is much lower.

The heat map in Figure 4 shows where the bikes will be handed in during the year in the city of Mannheim. It can be clearly seen that the bikes are handed in more often in the city centre, as expected. Nevertheless, there are some hot spots outside the city, such as the "Duale Hochschule" in the east of Mannheim.

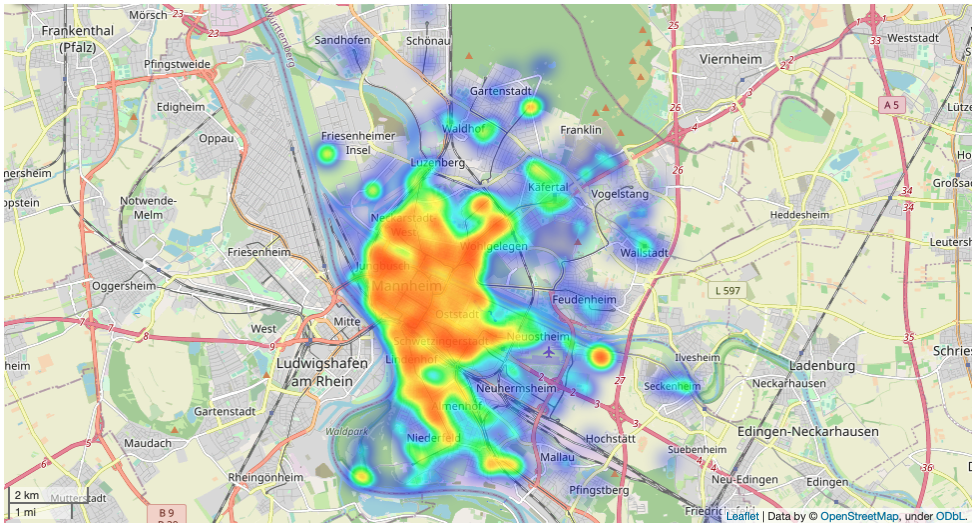

Figure 4: Heat map of the End Trips of the Whole Period 2019

To see if certain major events have an impact on the bike drop-off or bike rental location, a dynamic plot was created that reflects the change in bike location on the exact dates. It can be seen that major events have little or no apparent effect on the trips. An exception is a Metallica concert on 25/08/2019 in the "Maimarkthalle" as shown in the Figure 5. Bicycles will almost rarely be parked at the "Maimarktgelände" in 2019. However, there is an increased pile-up at the event.

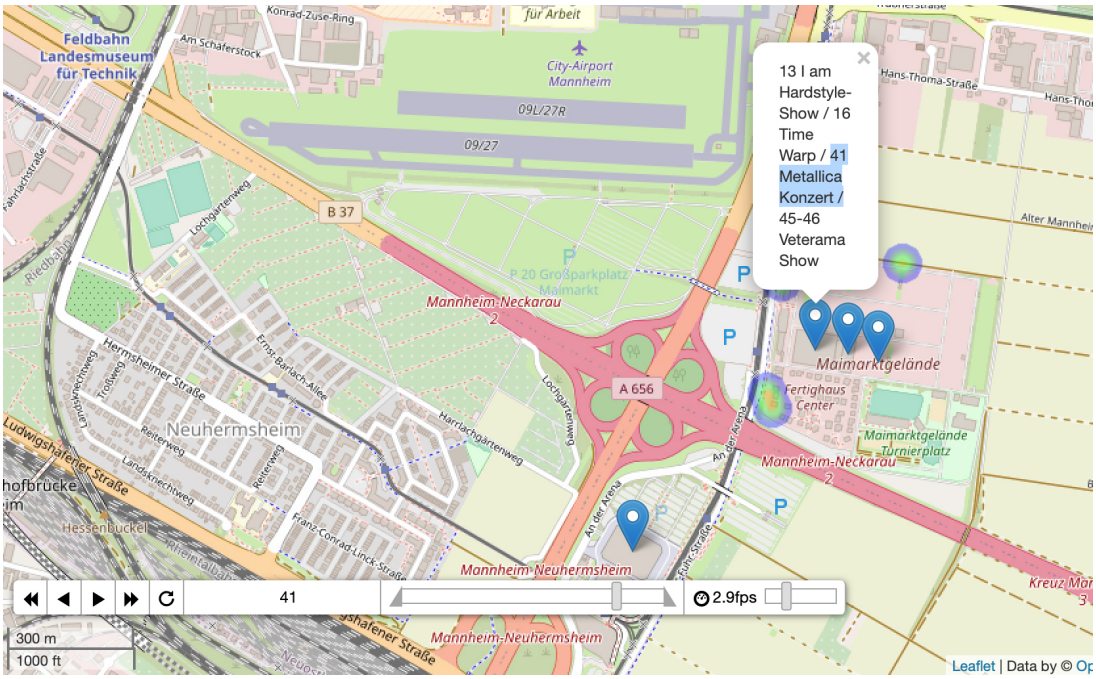

Figure 5: End of Trips Before a Metallica Concert

Since the length of the trips is predicted subsequently, it is advantageous to take a closer look at the duration of the trip and its distribution. In Figure 6, the distribution of the duration of the journey was visualized over the entire period. By comparing the distribution of trip lengths per months Figure 18 and the normal distribution with mean and standard deviation it can be stated that the distribution is not normally distributed and there are many short trips where the high quantity correlates with the duration, so this decreases with longer trips.

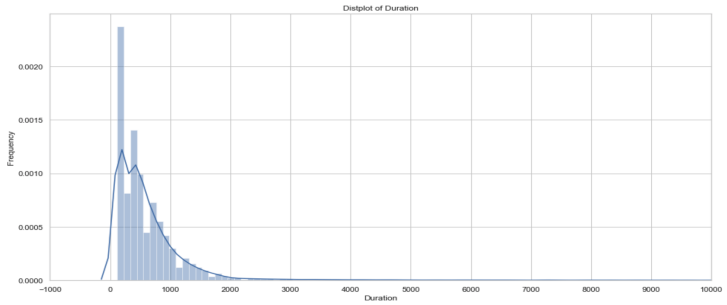

Figure 6: Distribution of the Rental Duration

For a better understanding of the duration distribution, each month was considered

individually and the duration distribution was compared to the normal distribution. The visualization is shown in 7 for January and in Appendix A.1 Figure 19, Figure 20 for the following months. This was compared with the mean and standard deviation of each month. July is excluded because no data is available for that month.

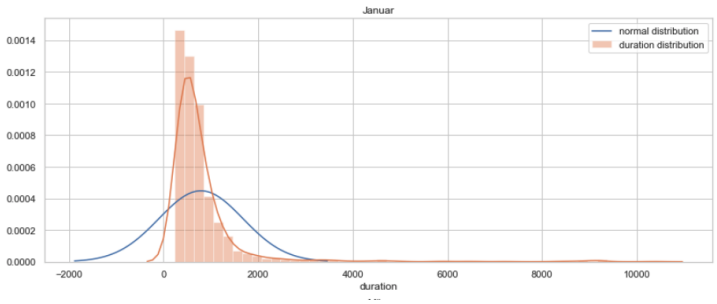

Figure 7: Distribution of the Duration in January Compared to the Normal Distribution

In order to get a better insight into the movements at stations, the 03.06.2020 was chosen. Therefore the number of stations in the morning, at noon and in the evening was considered. A slight shift of the bicycles in Figure 8, Figure 9, Figure 10 from the city centre to the outside and then back to the city centre can be seen. However, since only a certain point in time is considered here, no conclusions can be drawn about the population. Therefore in Appendix A.1 the Figure 21 and Figure 22 are more meaningful. There the bicycle movement of start and end stations of the whole period was visualized. From this figure, it is possible to see which of the stations are most frequented.

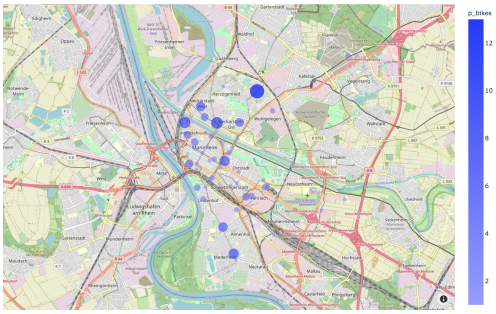

Figure 8: Bikes at Stations at 6am

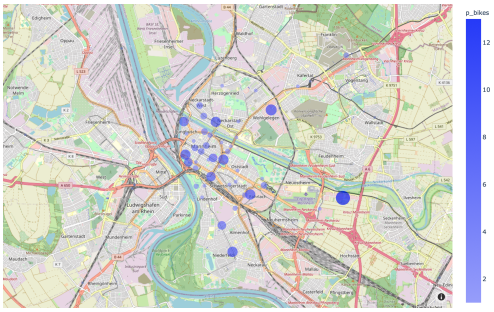

Figure 9: Bikes at Stations at 1pm

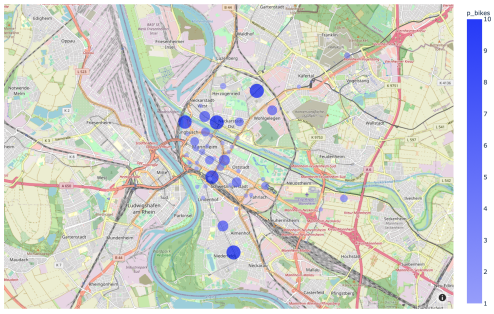

Figure 10: Bikes at Stations at 8pm

5. Predictive Analytics

The following two sub-chapters (5.1, 5.2) will comment on the steps undertaken during feature engineering, modelling, evaluation as described in the CRISP-DM framework. It is important to mention that the iterative nature of this process is not reflected in this report as this is beyond the scope and the purpose of this project document. Each of them covers a specific predictive target. In section Duration Prediction prediction of the continuous target booking duration is examined. The following section, Direction Classification, deals with the classification of the nominal target feature direction which represents whether trips go towards or away from the university. At the end of each sub-chapter, it will be commented on real test-set performance briefly. Possible future enhancements and limitations will not be discussed in this chapter but dealt with in Section 6. The third sub-chapter (5.3 Implementation) goes in line with the phase of deployment and comments on choices regarding the architecture and programming style.

5.1. Duration Prediction

One major challenge in prediction is to find the right model which is able to fit the underlying distribution of the target variable and its relationship with explanatory features. For this purpose, one can examine the statistical properties of the data by looking at descriptive statistics as well as plots which enable a rough overview on the underlying dynamics that are present. As already discussed in Section 4, the raw data contained many trips which showed a duration of exactly 120 or 180 seconds that could be identified as faulty bookings. As these faulty bookings skew the true distribution of booking duration, which can easily be seen by looking at Figure 6 in the Appendix A.1, a dedicated feature was created in which observations with bookings with a shorter or equal duration of 180 seconds and same start and end positions were labelled as false. This was done, instead of dropping false trips, in order to ensure that when predicting unseen data the model would not predict a longer duration for a booking that is most probably going to be false. Since the information on duration is not present in a real prediction scenario a random forest classifier (RFC) was trained, using only information that would be available at the start of the booking, to predict whether an observation is a false booking. This allowed for the usage of this feature which was originally only known from using duration information. The classifier hereby reached a classification validation accuracy of around 91.3% and was trained and evaluated with the same train-validation split used for the duration prediction model so that the duration prediction model could be trained with the deduced feature from duration information and validation be made with the predicted feature of false bookings. Table 2 summarizes all features that are available at prediction time and can therefore be used for feature engineering or prediction. Apart from particularly short or faulty bookings there were also few particularly long bookings in the data. Even though it

Feature Name	Description
Start Time	Timestamp containing information in the format YYYY-MM-DD HH:MM:SS.
Weekend	Binary variable indicating whether booking was conducted on the weekend.
Start Position Name	Name of the start position if booked from a station, else it is the bike number.
Is Station	Binary variable indicating whether booking started and ended at a non-floating station.
False Booking	Binary variable indicating whether a booking was false.

Table 2: Available Features for Model Training in Duration Prediction.

is recommended to remove these observations when they can be classified as outliers it is difficult to find a threshold since it is practically possible that bookings exceed a duration of 8000 seconds. In addition, these bookings were not completely rare. Based on these findings and the fact that only data from one year was provided these bookings were not removed from the data and were included, because a reasoned and detailed handling of outlier removal would have been beyond the scope of this report.

After the data was investigated in regard to its validity and distribution, possible features were investigated and engineered.The final data structure and used features as a result of the the engineering process are shown in Table 8 in Appendix A.2. Information on start time was used to extract the hour of the day, the day of the week and the week of the year. Instead of also extracting month of the year it was decided to use season as the next larger resolution. Firstly, as previous analysis has shown, one month was missing from the data set which could result in problems regarding the prediction of this month. Secondly, this was done because only one year of data was available. Therefore higher resolutions deliver a more robust explanatory power since the average influence of these variables can be calculated on a larger number of observations and therefore show less random variance. Even though these arguments could also be made for week of the year, the combination of season and calendar week allows to balance out the trade off between higher precision due to higher temporal granularity and error associated to random variance. As already shown in Section 4 all the time related features selected show differences in booking duration and were therefore used as explanatory features.

After the extraction of time related features the column start time was dropped and all the resulting features were transformed using a form of sine-cosine transformation to map these periodical features on a two dimensional circle as shown by Kaleko (2017). Doing this ensures that distances between points in time are kept intact, e.g. if the feature hour would be one-hot-encoded then 11pm would be one step away from 0am as well as 5am even though in a temporal sphere the distances differ. Since, however, these transformations produce less interpretable results regarding the importance of features, season was used as a one-hot-encoded variable since distances are very small and interpretability was therefore chosen over the tiny-marginal precision improvement which could have been achieved by the transformation of this particular feature. Apart from these time related features, weekend and is station were used as is. In addition, fixed stations from the

column start position name were utilized as one-hot-encoded features assuming that e.g. the trip duration will be longer when a trip is started further away from the city centre. It is important to mention that for both one-hot-encoded features no category was dropped since the final method used does not encounter issues with multicollinearity.

Another useful feature when dealing with data that contains time stamps is to look for autocorrelation as this is often present in time series data. As one, however, can see by looking at Figure 11 it becomes clear that there is no significant autocorrelation present for booking duration. Therefore lagged values of the target variable were not included as features.

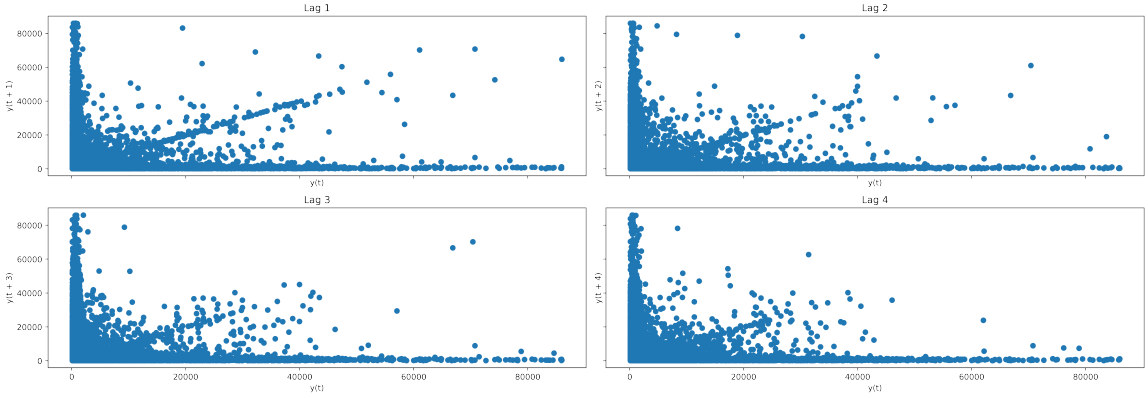

Figure 11: Fourth Order Autocorrelation of Booking Duration

After all features were engineered train-validation-split was conducted using scikit-learn's built in method. It is important to mention that scaling the target feature was tested but did not change predictive accuracy and was therefore dropped during the modelling process.

After all desired features have been engineered a baseline model was trained. Due to the high number of binary variables, that came from the one-hot-encoding of the start stations, scikit-learn's build-in LassoLarsCV model was used as it also conducts feature selection (Hastie et al., 2009). The model was trained with scikit-learn's default hyper-parameters [2] and reached a validation mean absolute error (MAE) of 1014.19 seconds. Figure 12 shows the predictions compared to the actual values from model validation as well as the resulting residuals. One can see that the baseline model does not exert the ability to catch the complex relationships between booking duration and explanatory variables since predictions are always close to the value of the faulty bookings. The model therefore does lack the ability to predict a majority of bookings especially when booking duration is long. Due to the fact that time features were transformed with the sine-cosine transformation a model that would catch a polynomial relationship without additional engineering and examination was needed, as the interpretation of these features is difficult. For this purpose random forest regression (RF) was chosen as it allows for polynomial and

[2] for details consult (Pedregosa et al., 2020)

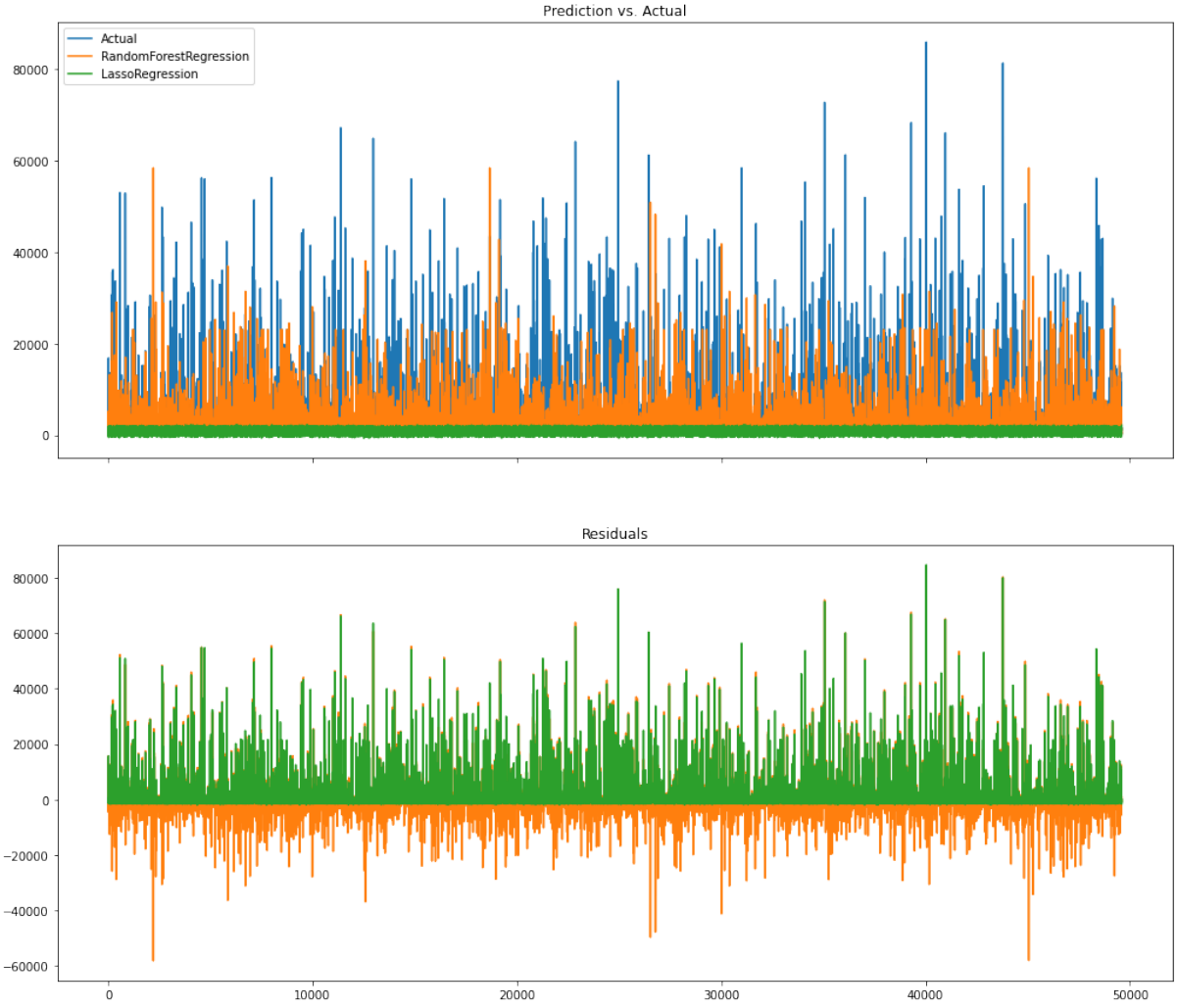

Figure 12: Prediction and Residuals of Random Forest and Lasso Regression

non-polynomial relationships as well as it reduces issues with multicollinearity (Dormann et al., 2013). The model was trained with scikit-learn's default hyper-parameters. The resulting MAE for the same validation data was 961.79 seconds. When comparing the predictions and residuals of both models looking at Figure 12, the first thing that catches the eye is that the RF model did better in predicting higher values as well as catching the dynamics that can be found in the data. This can also be seen by looking at the residuals which resemble a normal distribution more than the residuals of the linear model which indicates that there is less of a systematic error present in the model. In general, it can therefore be said that the RF model is more suitable for the purpose of this package and was therefore considered as a the final model for deployment. Figure 13 shows the feature importance of each explanatory variable. It gets obvious that the sine-cosine transformed time related variables exert the biggest influence on booking duration. With day of the week having the largest influence. The seasons as well as station and weekend information are of less importance. The feature false bookings feature which was itself predicted by the random forest classifier is second most important after the sine-cosine transformed time

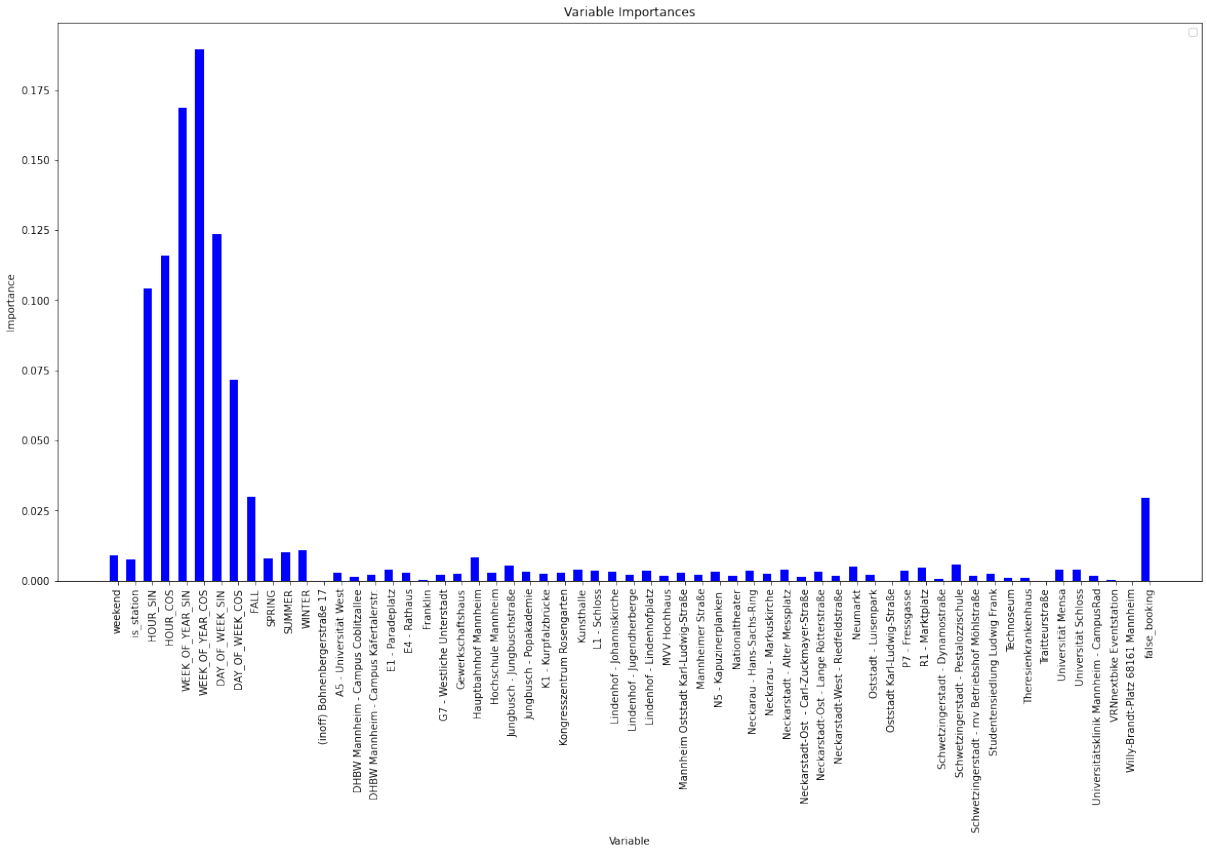

Figure 13: Feature Importances of Random Forest Regression

related features. Model performance on the test set was not as good as on the validation set. It amounted to 1413.76 seconds. This however can be explained by the fact that the test data consisted of bookings during the missing month of July, since it can be expected that dynamics in July are different to the the other month e.g. most number of bookings, fewer false bookings, lower average booking duration than other warm month. In addition, data was limited to one year of time, since therefore the model cannot learn from robust averages of month or in this case seasons missing July means that the average influence of the respective season is biased. This is especially true because July has low average duration compared to the other warmer month.

5.2. Direction Classification

To efficiently re-allocate bikes one needs to know where the majority of bikes will be available at certain points in time. In order to gain first insights for this it is helpful to know how many bikes will be ridden from certain areas to other areas. Even though, the most exhaustive approach would be to predict these arrivals and departures per station, it was decided, as a first attempt, to predict whether certain trips are leading towards or away from university related stations. For this purpose, a classification model was trained predicting four different classes.

	Random Forest				Logistic Regression			
	Precision	**Recall**	**F1-Score**	**Support**	**Precision**	**Recall**	**F1-Score**	**Support**
false	0.78	0.86	0.82	10680	0.00	0.18	0.00	39
from_university	0.33	0.44	0.38	6141	0.00	0.05	0.00	108
not_university	0.84	0.66	0.74	30371	1.00	0.48	0.65	49449
to_university	0.13	0.32	0.19	2404	0.00	0.00	0.00	0
accuracy			0.66	49596			0.48	49596
macro avg	0.52	0.57	0.53	49596	0.25	0.18	0.16	49596
weighted avg	0.73	0.66	0.68	49596	0.99	0.48	0.65	49596

Table 4: Validation Accuracy of Baseline and Final Model

As previous analysis has shown, there were many false bookings present in the data, the first class therefore represents bookings that were not successfully started. The next class deals with trips that were not university related at all and the last two classes embody trips that arrive or depart from university stations. A list of selected stations that were seen as university station is shown in Table 3. The data was labeled as follows: First, any trip starting at a university station got the label from_university. In the second run, any trip ending at a university station got the label to_university. Overwriting the first label if start and end were at the university, circumvent the issue of trips that are both departing and arriving in this same area.[3] Third, any trip fulfilling the criteria of a false booking as described in Section 5.1 got the label false outvoting the previous labels also controlling for trips that either start or end at the university. Finally, any trip not having a label to this point got the label not_university. Since the results in this prediction scenario are only of use if predictions can be made independently of trips i.e. that re-allocation can be planned in advance and is not related to individual trips, start and end position of trips could not be used as features in prediction. Apart from time related features, information on whether a trip would start and end at a fixed station was included as well, as this enables control on whether bikes need to be only collected from fixed stations or from non-station related areas. Since this classification task deals with the prediction of a categorical target, the time related features were not sine-cosine transformed nor were they one-hot-encoded to at least keep ordinal aspect of the features.

Station Name
DHBW Mannheim - Campus Coblitzallee
A5 - Universität West
L1 - Schloss
DHBW Mannheim - Campus Käfertalerstr
Universität Schloss
Universität Mensa
Universitätsklinik Mannheim - CampusRad
Hochschule Mannheim

Table 3: University Stations

As for the duration prediction a simplistic baseline model was trained for later evaluation. For this scikit-learn's logistic regression model was utilized with default parameters. The final model was a random forest classifier (RFC) again with no hyper-parameter tuning. Table 4 shows the output of scikit-learn's classification report. It gets clear that the RFC model did provide better predictions considering the average accuracy increase of 18%. Another thing that catches the eye is that the baseline model does lack the ability to predict the university related classes at all, which can be seen by zero precision and recall

[3]This goes in hand with the goal since rides ending at the university will mean that bikes remain at the university which then would have to be re-allocated from there if needed.

for these classes. This is also reflected by the F1-Scores, which generally should be used for model comparison, in which the baseline model performed poorly among all classes except not_university. In general, it can therefore be said that the trade of between complexity and predictive performance is justified. Even though a 66% accuracy is not bad performance considering that four classes were predicted, looking at the recall of the RFC model it gets clear that this classifier also has problems with predicting the university related classes. This however can be attributed to the imbalance of class counts. Both not university and false accounted for around 71% of the data and only 17% were away from the university and 12% to the university. Which explains better precision for the not university related classes.

Figure 14 shows the composition of classes at different temporal resolutions and their points in time and the associated predictive accuracy. Firstly, it gets obvious that time

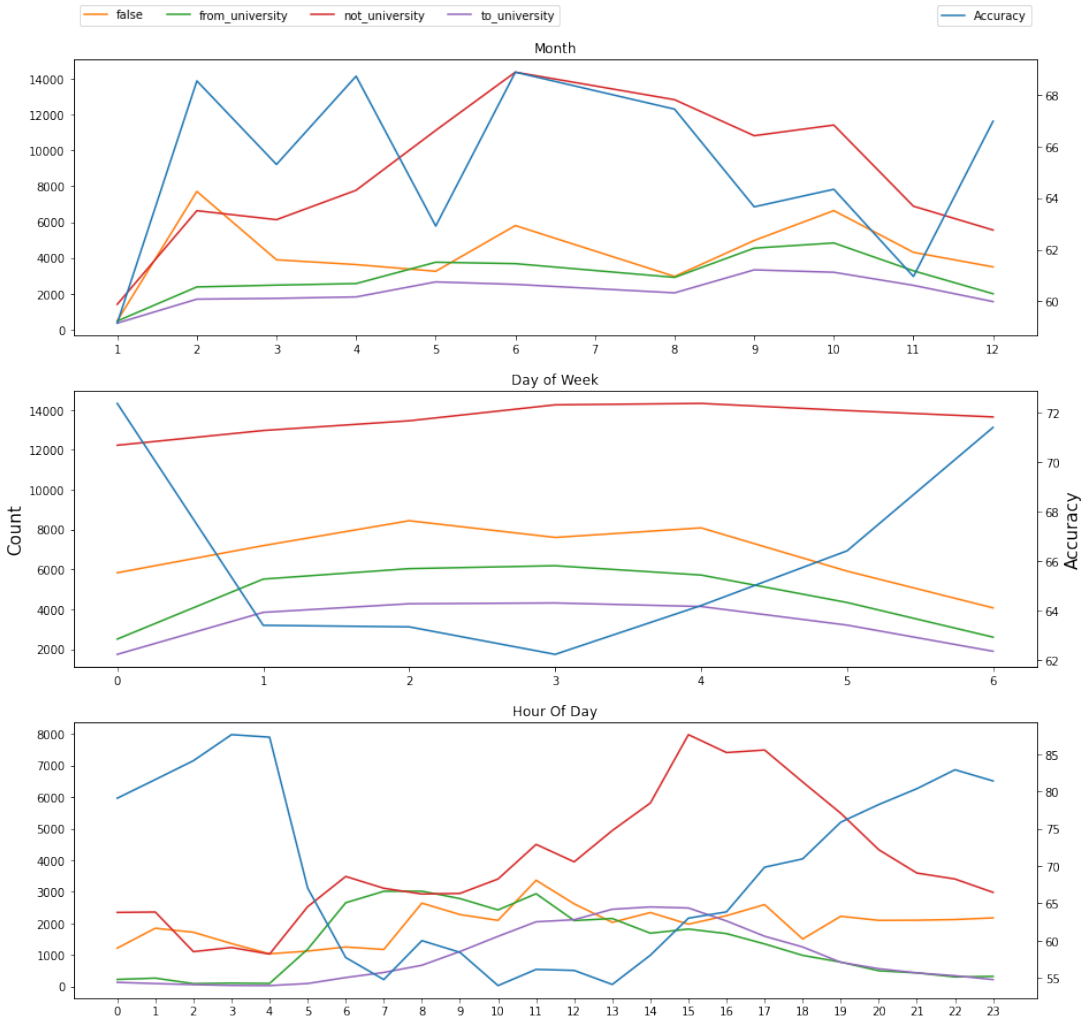

Figure 14: Class Counts and Accuracy at Different Temporal Resolutions

related features have explanatory power since average composition is changing constantly when changing the point in time. Looking at the monthly changes, however, no clear pattern can be found since the accuracy is oscillating strongly. Looking at the day of

week or the hour of day it seems that in times where there are less number of bookings accuracy is generally higher. Secondly, one can see that at both of these resolutions' predictive accuracy is higher when trips that are university related are low. This might be due to the fact that university related trips are usually low at nights and early during mornings as well as during weekends and that the classifier exerts more ability to predict the not university related trips therefore the effects of lower number of bookings as well as most trips belong to the better predictable classes add up and result in higher overall predictive performance.

This hypothesis can also be confirmed by looking at the overall test performance in Table 5. The test data consisted solely of bookings from the missing month July. First of all, it has to be said that this month contained about double as many bookings compared to the least busy month which is December, which contained one of the best accuracy at the same time.

	Precision	**Recall**	**F1-Score**	**Support**
false	0.27	0.03	0.05	10300
from_university	0.18	0.10	0.13	2914
not_university	0.54	0.92	0.68	17737
to_university	0.12	0.01	0.02	2197
accuracy			0.51	33148
macro avg	0.28	0.27	0.22	33148
weighted avg	0.40	0.51	0.39	33148

Table 5: Test Accuracy of Final Model

The overall accuracy fell by 15%. It can also be seen that the classes representing trips towards or away from the university were difficult to predict. Another thing that catches the eye is that that the false category of false booking was predicted very poorly compared to validation results. This can be explained that were fewer false bookings compared to the other month. In addition trips from and towards the university were particularly low and trips that were not university related were highest in July.

5.3. Implementation

All functionality regarding the training and prediction with machine learning models was developed inside the sub module `models` (cf. Appendix A.1, Figure 15). The implementation was conducted in a way to fulfil both, the functional and non-functional requirements as described in Section 2.2. For the sake of extensibility and maintainability an abstract base class `Model` was defined mandating methods that are necessary for training and prediction, so that each developer knows which methods he has to implement for a new model. This class should also ensure that later integration with the pre-processing

steps as well as into the CLI could be done using standard methods and that extending the model pool by other predictors would work seamlessly. Each predictive target was handled by its own separate model class. Feature engineering process as well as helpful feature engineering methods were put into a standalone utility script allowing for later use in other models, trying to reduce redundancy by extracting methods that could be used in multiple prediction scenarios. In order to achieve a reasonable level of modularity IO related functionality was kept inside the `io` sub-module allowing easy extensibility without changing any business logic. Table 6 shows the methods that are mandatory when implementing new prediction models for the package.

Name of Method	Functional Requirement
load_from_csv	Load data from a raw file and pass it over to load_from_transformer.
load_from_transformer	Transform data using package's transform and pre-processing functionality and conduct feature engineering using custom methods. Save the results to the needed class attributes.
train	Train a machine learning model using data stored in class attributes and save the models and if needed transformers to disk. Finally update the class attributes
predict	Predict either new data or data that is already stored in class, save predictions to disc and update class attributes.
training_score	If class instance was used for training and prediction of the training data return the training score.

Table 6: Mandatory Methods for Prediction Models

The first two methods aim to handle the data import and transformation of data for training or prediction. In order to save time when the data set is already transformed both a .csv import and an import of already transformed data set are mandated. Apart from simple transformations, in these methods additional feature engineering steps that go beyond the transformation steps and differ from model to model should be called. Train and predict are basically wrapper functions for scikit-learn's model library that interact seaming less with the provided data and CLI and provide basic training and prediction functionality. The last method allows for a training score of a previously trained model after the same data which was used for training is predicted. It is important to mention that a train-validation split was not implemented since models and features were already selected before hand. In addition, these classes and their implemented methods do not aim to support easy modelling but rather a standardised template for deployment and quick utilization purposes.

6. Limitations

A fundamental limit can be set on the credibility and informative value of the results which is caused through the amount of data. For the scope of the use case, only one data set from nextbike of Mannheim from 2019 was analysed. So it is for example difficult to use the means of month and calendar weeks. In addition, the consideration of one data set of the city leads to a limited result that, for example, detected patterns of driving behaviour resulting from the data exploration and supervised analysis are only valid for the city of Mannheim.

Furthermore, the influences of weather was not considered and how the weather could affect the rental behaviour. With the help of weather data, rental patterns can be examined from a different point of view, so that the interpretation of the results from predictive and descriptive analysis are more accurate. This however was non only beyond the scope of this analysis but also due to the fact that the time span of the test data was not explicitly specified. Even though this could have been solved by using a dynamic weather API there is no publicly available free API for the city Mannheim allowing to collect weather data dynamically.

Furthermore, a grid search could have been conducted, but a lack of additional data this would have resulted in an ultimate over-fitting on either the training or the validation set. An exhaustive grid-search would therefore not have resulted in a validated and accurate performance.

In addition, other more complex models such as boosted trees or neural networks were not considered as circumstances, such as amount of data, did result in these models to be in suitable. In addition random forest models proved to be easy to train and to tune (Hastie et al., 2009). In addition an intensive examination of model alternatives was beyond the scope of this project but should be of interest for future work.

The contemplation of free floating stations or stations from competitors for a more efficient re-allocation suggestion would have been useful to identify better or more important areas. But the analysis would disregard the framework of this report, so this approach provides another scope for a different scope of further analysis.

7. Summary and Outlook

This report dealt with the development of a usable self-contained nextbike package which fulfils the descriptive and predictive requirements defined in Sections 4 and 5. Meta information of the package describes its functionality and method briefly.

The results provide a first draft and serve as a guide for nextbike's management to allocate bicycles more efficiently and effectively in order to optimize sales and customer needs. Based on driving behaviour patterns, it can be stated that the city centre of Mannheim tends to show a high level of traffic of started and finished trips. It is, therefore, advantageous for nextbike to keep the quantity of their bikes at hot spots, such as the main station or university, high in order to meet the demand of customers and to achieve an optimal level of bike usage.

The results of data exploration and predictive analysis give first insights into bike rental patterns and also provide a basis for further analysis. An outlook for further analysis, predictive or descriptive, can be carried out under other aspects. These include, for example, clustering booking behaviour, clustering booking duration, classification of booking behaviour, predicting bike rental demand, predicting bike availability per station or predicting bike availability within discretized areas.

A. Appendices

A.1. Figures

```
$WORKING_DIRECTORY/
    nextbike/
        cli/
            predict/
                commands.py
            train/
                commands.py
            transform/
                commands.py
            cli.py
        io/
        models/
            DestinationModel.py
            DurationModel.py
            Model.py
            util.py
        preprocessing/
            AbstractValidator.py
            ParallelPreprocessor.py
            Preprocessor.py
            Transformer.py
    setup.py
```

Figure 15: Directory Tree (shortened for brevity) of the Nextbike Package.

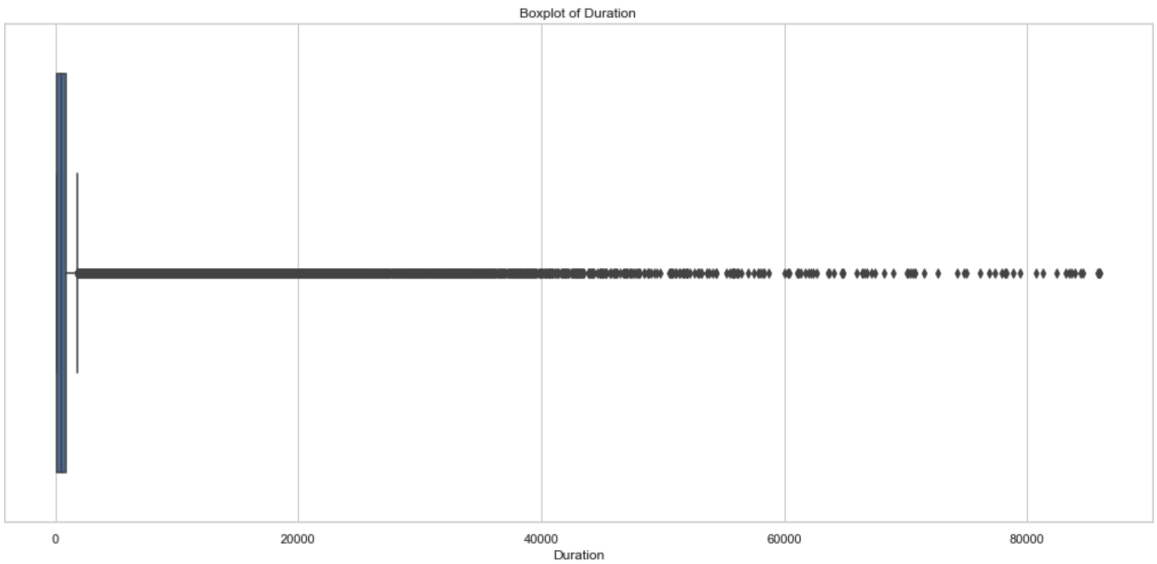

Figure 16: Statistical Quantities of Bike Trip Duration

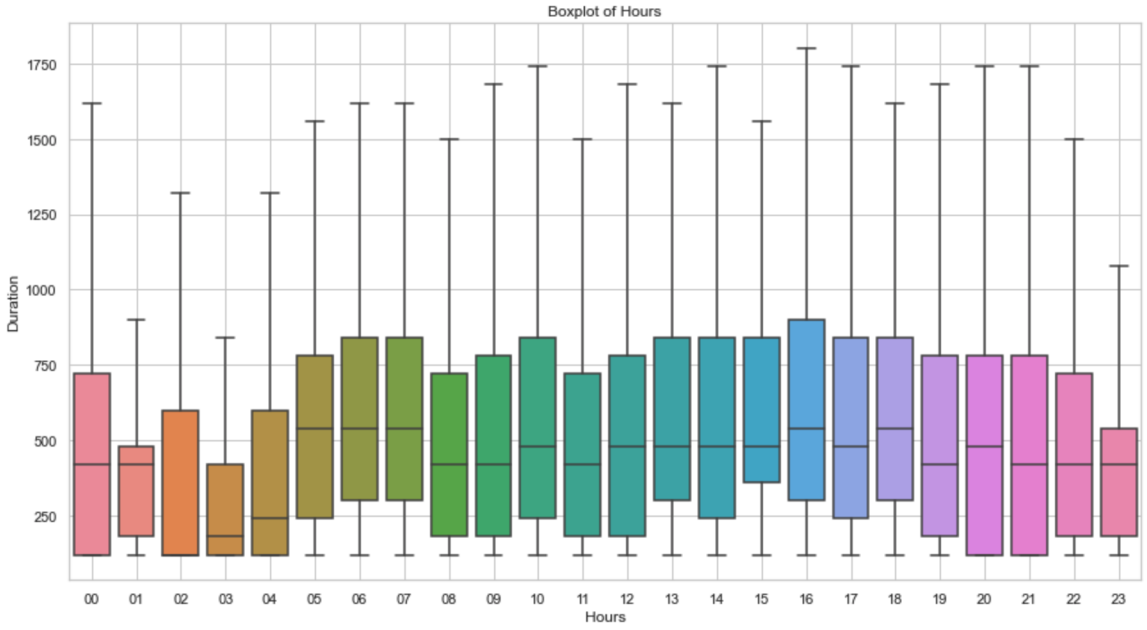

Figure 17: Statistical Quantities of Duration Divided Into Hours

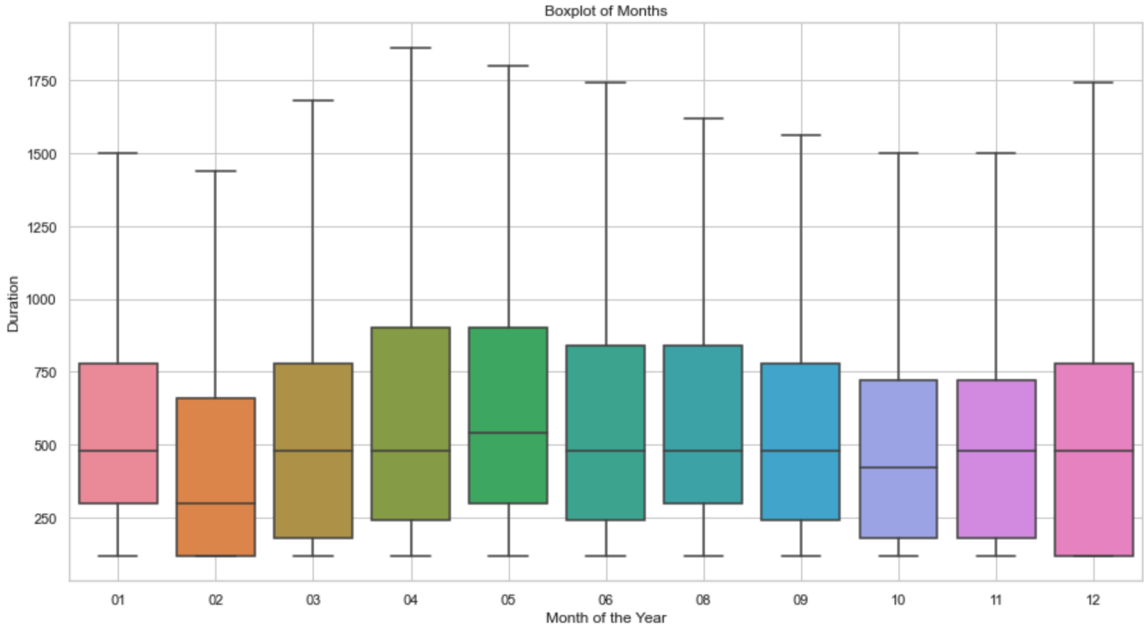

Figure 18: Statistical Quantities of Duration Divided Into Months

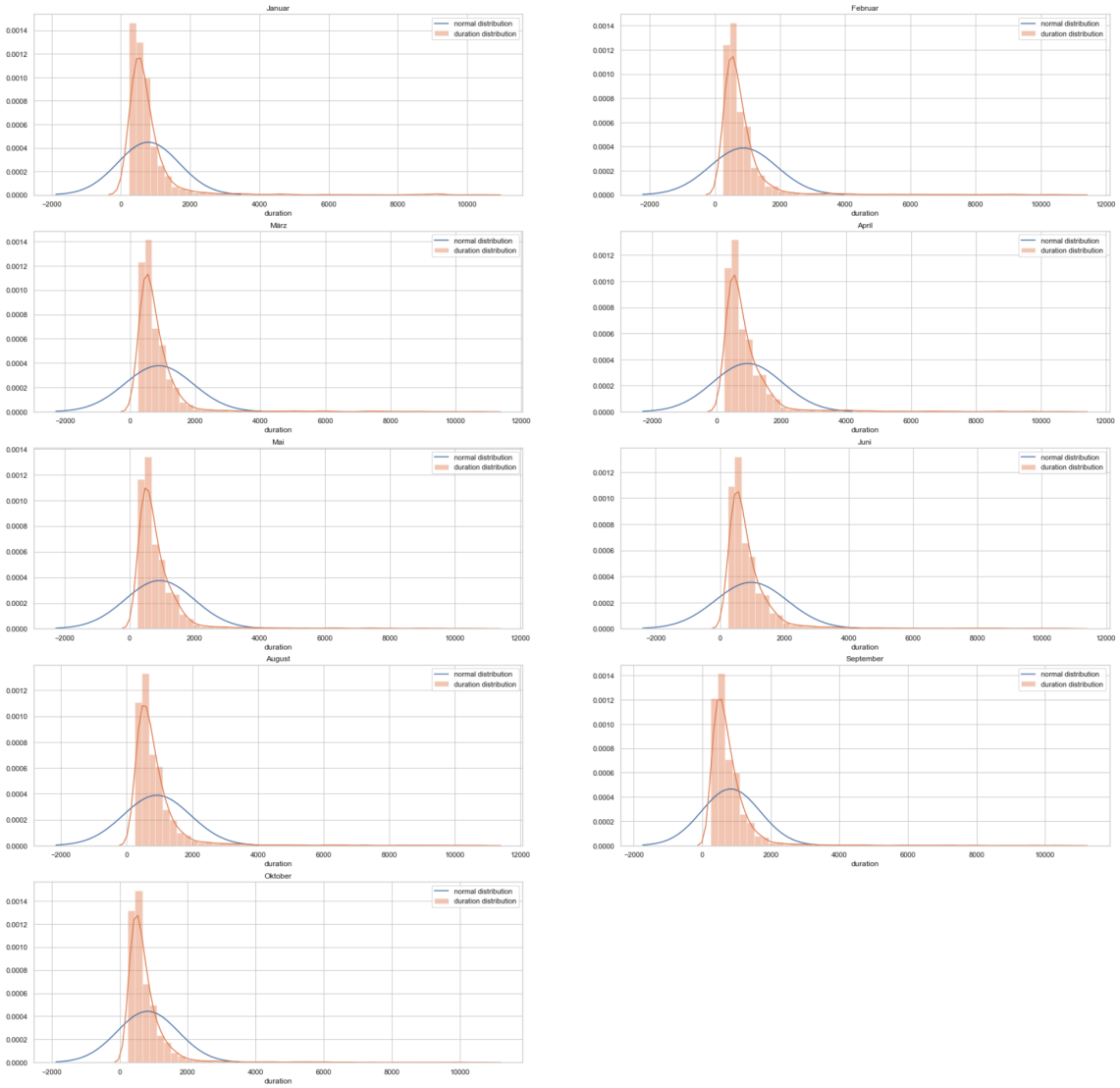

Figure 19: Distribution the of Duration from January to October

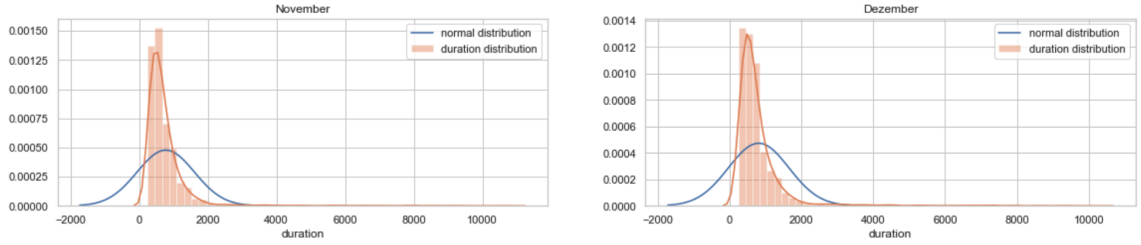

Figure 20: Distribution of the Duration of November and December

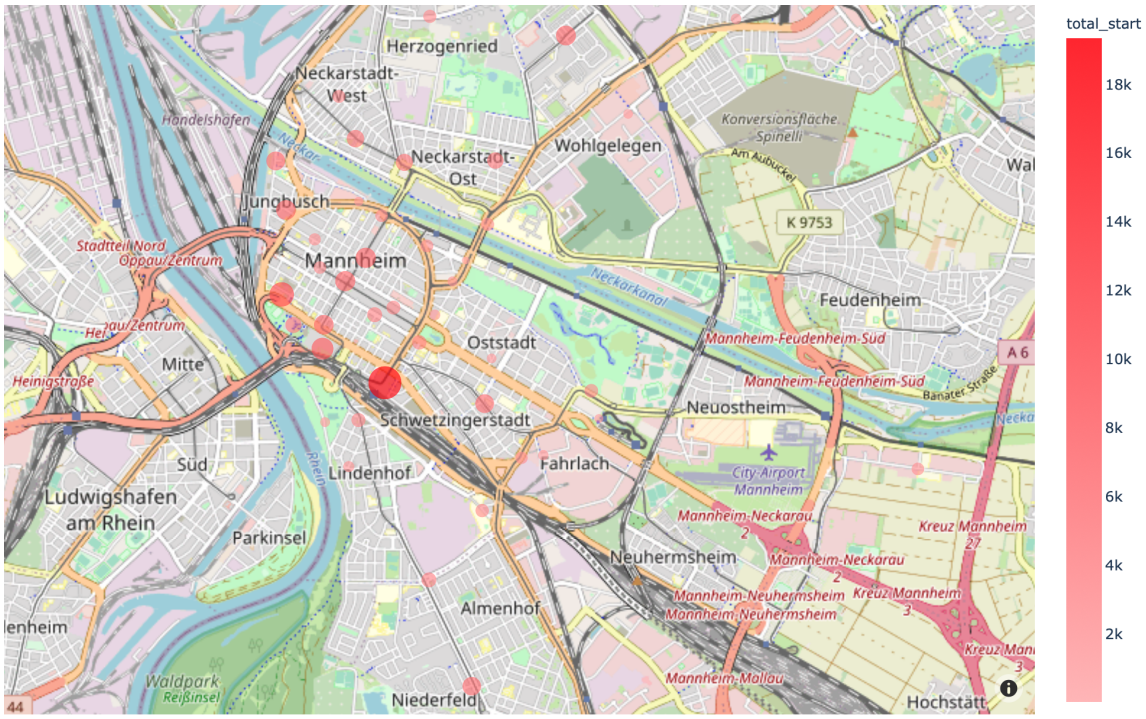

Figure 21: Started Bike Bookings of each Station

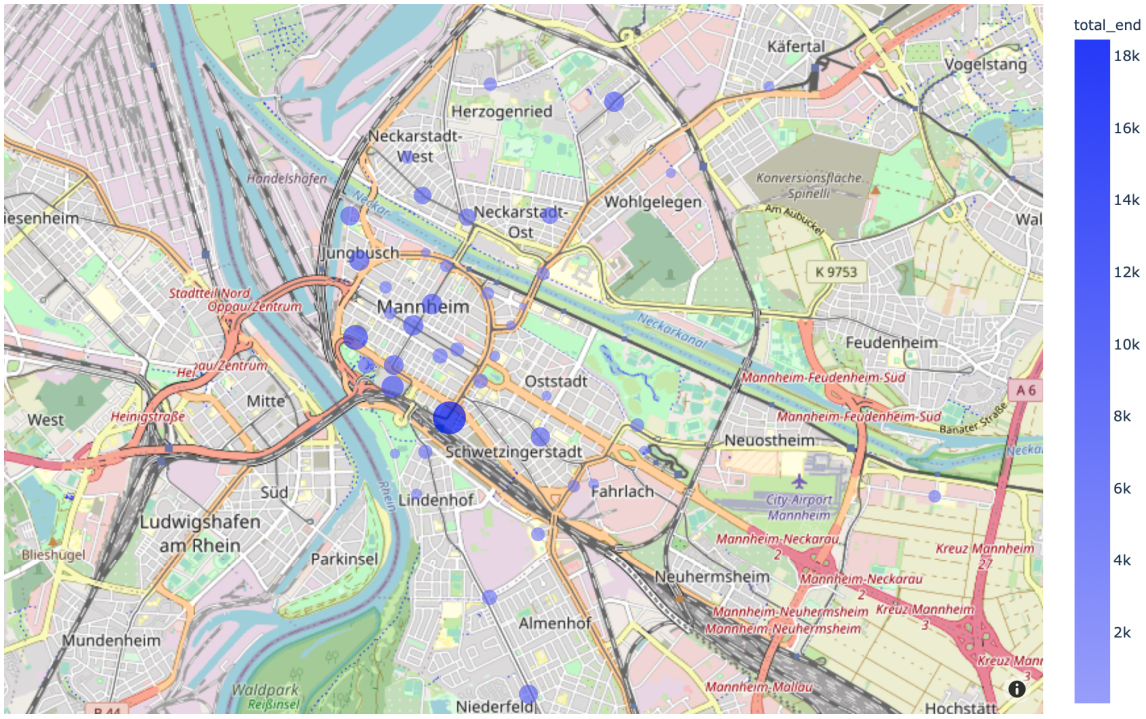

Figure 22: Ended Bike Bookings of each Station

A.2. Tables

Columns Name	dtype	Description
p_spot	bool	Whether it is a spot or not.
p_place_type	int64	It is 0 or 12.
datetime	datetime64[ns]	The date of booking.
b_number	int64	Number of a bike.
trip	object	Declares the type of the booking.
p_uid	int64	ID of place.
p_bikes	int64	Number of bikes at a place.
p_lat	float64	Latitude of a bike at a place.
p_lng	float64	Longitude of a bike at a place.
p_number	float64	Number of a place.
p_bike	bool	Whether bike is in station or not.
p_name	object	Station name.
b_bike_type	int64	Type of a bike.
geomtery	geometry	Coordinates of a bike.

Table 7: Features of Raw Data

Feature Name	dtype	Description
weekend	bool	Binary variable indicating whether booking was conducted on the weekend.
is_station	bool	Binary Variable indicating whether booking started and ended at a non-floating station.
hour_sin	float64	Sine transformed hour of the day
hour_cos	float64	Cosine transformed hour of the day
week_of_year_sin	float64	Sine transformed week of the year
week_of_year_cos	float64	Cosine transformed hour of the day
day_of_week_sin	float64	Cosine transformed hour of the day
day_of_week_cos	float64	Cosine transformed day of the week
season	bool	Binary variable indicating whether the trip lies in the specific season.[1]
false_booking	bool	Binary Variable indicating whether a booking was false.
station	bool	Binary variable indicating whether a the trip was started at the specific station.[2]

[1] Present as one-hot-encoded feature for every season.
[2] Present as one-hot-encoded feature for every non-floating station.

Table 8: Features Used for Prediction

B. References

Dormann, C. F., Elith, J., Bacher, S., Buchmann, C., Carl, G., Carré, G., Marquéz, J. R. G., Gruber, B., Lafourcade, B., Leitão, P. J., Münkemüller, T., McClean, C., Osborne, P. E., Reineking, B., Schröder, B., Skidmore, A. K., Zurell, D. & Lautenbach, S. (2013). Collinearity: a review of methods to deal with it and a simulation study evaluating their performance. *Ecography*, *36*(1), 27–46.

GeoPandas developers. (2019). GeoPandas 0.7.0 — GeoPandas 0.7.0 documentation. https://geopandas.org/index.html

Gillies, S., Bierbaum, A. & Lautaportti, K. (2013). Shapely — Shapely 1.8dev documentation. https://shapely.readthedocs.io/en/latest/index.html

Hastie, T., Tibshirani, R. & Friedman, J. (2009). *Springer Series in Statistics The Elements of Statistical Learning* (Vol. 27). New York, Springer.

Kaleko, D. (2017). Feature Engineering - Handling Cyclical Features. http://blog.davidkaleko.com/feature-engineering-cyclical-features.html

Kannan, K. S., Manoj, K. & Arumugam, S. (2015). Labeling Methods for Identifying Outliers. *International Journal of Statistics and Systems(IJSS)*, *10*(2), 231–238.

nextbike GmbH. (2019). Status der verfügbaren VRNNextbike Fahrräder an den Stationen in Mannheim. https://mannheim.opendatasoft.com/explore/dataset/free_bike_status/information/

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. & Duchesnay, E. (2020). scikit-learn - API Reference. https://scikit-learn.org/stable/modules/classes.html#module-sklearn.ensemble

Python Packaging Authority. (2006). Easily download, build, install, upgrade, and uninstall Python packages. https://pypi.org/project/setuptools/

Python Software Foundation. (2020a). abc — Abstract Base Classes — Python 3.7.7 documentation. https://docs.python.org/3.7/library/abc.html

Python Software Foundation. (2020b). PEP 484 – Type Hints | Python.org. https://www.python.org/dev/peps/pep-0484/

van Rossum, G., Warsaw, B. & Coghlan, N. (2001). PEP 8 – Style Guide for Python Code. https://www.python.org/dev/peps/pep-0008/#id23

VRN. (2020). Häufig gestellte Fragen und Antworten. https://www.vrnnextbike.de/de/mannheim/faq/

Wirth, R. (2000). CRISP-DM : Towards a Standard Process Model for Data Mining. *Proceedings of the Fourth International Conference on the Practical Application of Knowledge Discovery and Data Mining*, (24959), 29–39.