FACULTY OF SCIENCE AND TECHNOLOGY

MASTER THESIS

Study programme / specialisation:

Master of Science in Computer Science, specialisation in Data Science

The spring semester, 2022

Open / Confidential

Author: Dawit Habtemariam Kidane

Dawit H. Kidane
(signature author)

Course coordinator:

Supervisor(s): Professor Reggie Davidrajuh

Co-supervisor: Daniel Barati

Thesis title:

Forecasting bicycle traffic in cities

Credits (ECTS): 30

Keywords:

-Machine learning

-Data Engineering

-Bike theft forecasting

-Bike traffic forecasting

Pages: ....52...........

+ appendix: ......9.........

Stavanger, 15/06/2022
date/year

Forecasting Bicycle Traffic in Cities

Bike Theft & Bike Traffic Predictions

Dawit Habtemariam Kidane - June 15, 2022

Abstract

In this project the task is to predict bicycle theft and bicycle traffic in a city using machine learning methods. The project proposal was given in collaboration with BikeFinder AS, a Petter Stordalen's "Strawberry Million" award winning company established in 2015. Bicycle theft is a problem in many places around the world and one of the objectives in this thesis is to help preventing it, based on data science analysis and machine learning methods applied on existing data. Predicting bicycle traffic as well as analyzing the factors that might affect traffic is another important goal for this thesis. However, throughout the project it is expected to work on various other steps such as gathering the relevant data, pre-processing, evaluating and comparing methods and results. It is also important to optimize and improve the performance of the methods to achieve as accurate results as possible. Lastly, interpreting the results, and solving the questions asked in the thesis.

The project has been solved by first, gathering BikeFinder theft and traffic data, Stavanger weather conditions data, Rogaland Police District bike theft reports data and data from the bike counting sensors in the city of Stavanger. Secondly, various steps of preprocessing has been done on the data according to the use cases. Afterwards, machine learning method evaluations and comparisons, using a neutral and larger dataset, Chicago crime dataset was accomplished. Thereafter, applying the best performing methods on the theft and traffic datasets, as well as forecasting bike theft and traffic has been achieved. Finally, results interpretation and discussion on the findings of the project.

The findings in this project reflects that bike theft and bike traffic can be predicted using machine learning methods on BikeFinder data. Furthermore, other factors such as weather conditions do affect bike traffic as well as improves the performances of bike traffic predictions. The results of the project provide useful insight to multiple parties and can be used to help preventing bike theft as well as providing suggestions for city planning improvements.

1. Introduction

This master thesis is about forecasting bike theft and bike traffic using machine learning and data engineering applied to different existing datasets. The objective is to use machine learning techniques to solve real life problems such as, bike theft by predicting potential theft risks in a given place and time. Also another objective is to analyze bike traffic and attempt to predict the traffic flow based on other factors such as weather. Lastly, evaluating BikeFinder dataset on how well it can perform with those type of analysis is another objective.

The history of bicycles go all the way back to the 19th century, or at least the first verifiable claim for a practically used bicycle belongs to Karl von Drais. The idea was a human powered vehicle, although it was pedal-less in the beginning, but it still served its purpose. By the early 21st century, more than 1 billion were in existence. Bicycles became a huge part of the human race throughout history and inspired a lot of other inventions for a long time. (*Mirrorpix, 2017*)

Ever since bicycles first invention, bicycles were constantly developed in different shapes and forms. Several major improvements has been done to bicycles throughout history, whether it is mechanically or even other major changes such as the addition of motors or electricity. However, the traditional idea of a simple man powered bike is still surviving and used daily by people of nearly all ages. Bicycles are used for many purposes such as a form of transportation vehicle, racing sports, exercising or even as a form of entertainment. This simple two-wheeled vehicle invention survived through centuries and still going strong, currently with a higher production rate than automobile. This is not coincidence, due to accessibility and simplicity of bicycles it is perhaps the most common choice of transportation for people all over the world.*[4]*

Today bicycles come in a wide range of categories and varieties which results in a huge price gaps between different bikes. Although bikes can be one of the most affordable transport vehicle options to own for many people, at the same time it should come as a surprise to find bikes that exceed the prices of automobiles. Even though bikes are not a motor driven automobiles, but they do have a lot in common. Bikes share the road with cars in some cases, used as a substitute for cars and are also in most cases parked and kept out doors. However, given how simple bikes are in terms of security compared to automobiles they become a more vulnerable target for theft. Bike thefts can be as simple as grabbing a bike and leaving within seconds, or in other cases breaking a lock and maybe disassemble the bike.

Either way, stealing a bike is not that challenging of a task to be done by the average person. London, for example, is considered the number one hotspot for bike theft. According to an article from *Cycling Weekly* magazine, between 2017 and 2021 around 162,943 bicycles were registered stolen in London. The real number could possibly be higher since it is unlikely that every bike owner reports a bike theft, this add up to around one bike is stolen every 16 minutes. The stolen bikes were worth around £93 million combined. *[1]*

As a result there are hundreds of police reports yearly about bike theft according to dataset from the police, even in cities as Stavanger, relatively smaller in size and population. Today, there are several options a person can choose between in order to prevent themselves from going through such losses. Some maybe less convenient options than others. For instance, avoiding to leave bikes out doors as much as possible. Other options could be some sort of an investment like an insurance, or even a more advanced option like installing a tracker such as BikeFinder, that will be introduced at section *2.1.1*. Regardless of what the choice is, there are no guarantees that a person will 100% avoid a loss. However, what can be done is decrease the chances as much as it possibly can. A great alternative that one might think of in this situation is probably looking into the future to avoid being at the wrong place in the wrong time. However, this is unfortunately not entirely possible, but the next best thing might just be predicting it, what if we can predict the wrong place and time to be at a certain place? Machine learning is the answer.

Since the term "Machine learning" was reportedly first introduced by Arthur Samuel in 1952 the term and the idea behind it has been revolutionary. The beginnings saw IBMs computer checkers playing program that learns and adapts playing chess based on experience. Eventually, "Deep Blue" was created, a computer that managed to beat the world chess champion Garry Kasparov in *May 3–11, 1997*. (*Garry Kasparov, 2017*)

One of the main uses that Machine learning can provide is predicting, future events. In this thesis the objective is to utilize some suitable machine learning techniques that learns the existing data and based on that, it should forecast bike traffic and bike theft.

1.2 Problem definition

BikeFinder possesses rich data sets containing location data of its customers biking routes. The idea for this master thesis is to use the data for analyses on biking habits and discover what valuable insights we can gather. In this project, preprocessing will also be a vital part of the project. As the nature of BikeFinder data is sensitive, an important step of the preprocessing stages is anonymazation of the location data so that no user can be directly or indirectly identified. A general analysis using data science methods needs to be done to familiarize with and gather insights about the data.

Analyzing and predicting bike traffic provides valuable information in many aspects for a city. Predicting where and when bikers will bike in, can give an insight to the city, as far as various city planning is concerned. Bike traffic and its correlation with weather is another insightful piece of information that may save the city or public transportation companies lots of resources. The companies can for instance have less routes when it is expected that bikers in an area will be biking at a specific day and time. Predicting bike traffic can be utilized and benefited by several other sides and companies, an example could be a sports company targeting bikers with advertisements through billboards in the predicted routes. This leads to the following questions:

- Biking patterns - Where and when do bikers ride? It would be interesting to restrict the analysis to one city and generate a heat map of the density of biking routes over time.

- Correlation with weather - Can we correlate the location data with weather data? To what extent does the type of weather and temperature influence the biking traffic?

- Bike traffic prediction - Is it possible to predict the bike traffic in a city?

- Theft prediction - Discover whether or not the current theft report dataset from

BikeFinder is rich enough to predict thefts. Is it possible to use other open data sets or to generate synthetic data as an input to the prediction algorithm?

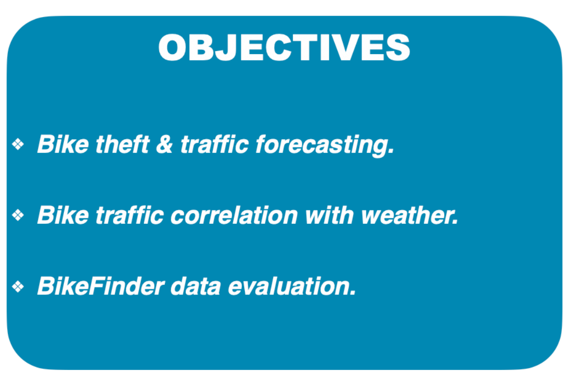

Fig:1. Main objectives

The motivation for this project is to provide insights to multiple parties. Providing the city insights about the routes traffic on different routes across the city so that they can plan better. The results in this project could say something about where a new route should be build. Additionally, in the case of theft prediction, both BikeFinder users and local police may be interested in where and when there is an increased chance of theft. As a feature of the BikeFinder app, this insight could be used to identify users when they are parking in locations with a high risk for theft.

Possible outcomes:

- Using BikeFinder data with combination of weather and public transportation data, bike theft and bike traffic predictions provide results with high accuracy. Based on the results in this project the objectives are achieved and a deeper understanding of the biking behaviors reached. The project focuses on the city of Stavanger.

- The gathered data is not suitable to achieve the objectives of this project. Analyze and explain why that is. Seek an alternative solution, perhaps a different data and compare it with the original data. Use similar data to the original ones and create a general solution that can simply be adapted to answer the questions in the project.

2. Literature Review and Formulation of the problem

Gathering datasets and predicting bike traffic and bike theft using machine learning are the main objectives. However, a number of pre processing stages should be done, such as data cleaning, evaluation, anonymazation and data engineering. Furthermore evaluating, tuning and improving the application is also necessary. The following sub-sections will extensively explain the literature review as well as defining the problem.

2.1 Literature review

2.1.1 BikeFinder

BikeFinder AS is a Stavanger based company that produces BikeFinder trackers. The idea with a BikeFinder tracker is to track a bicycles position if it was to get stolen. The tracker is installed in the bicycles handlebar. When the bike is moving, the tracker sends position signals to the BikeFinder system through satellite. These positions can then be tracked by the user through the BikeFinder app. The BikeFinder user can then locate their stolen bicycles and for example contact the police. If the bicycle is not found and all the insurance requirements were full filled, then the user can be covered by insurance. *[5]*

When a theft occurs, the user can report the theft through the app by clicking on a button. The report is then registered in the database with report time and the device id of the tracker. When a theft report is reported the user is contacted by the BikeFinder support team and then both collaborates to find the bicycles. *[5]*

2.1.2 Machine Learning

Machine learning is a type of artificial intelligence that allows software applications to become more accurate at predicting outcomes without being explicitly programmed to do so. Machine learning algorithms use historical data as input to predict new output values. The idea is to give the computer the capability of learning and improving by identifying patterns based on past experiences, similar to human beings. A number of jobs that required human resources due to the capabilities of adapting and the requirement of less general solutions in the past, now can be achieved by computers with the help of machine learning.*[3]*

There are two areas of machine learning, Supervised learning and unsupervised learning. Supervised learning uses the input data as well as the output data to train the model and then predict the output when it is given new data. Some popular examples of supervised machine learning algorithms are: Linear regression for regression problems, Random forest for classification and regression problems and Support vector machines for classification problems. *[3]*

Unsupervised learning in the other hand finds unknown patterns in data.
In unsupervised learning, the algorithm tries to learn some inherent structure to the data with only input data. Two common unsupervised learning algorithms are clustering and dimensionality reduction. In clustering, we attempt to group data points into meaningful clusters such that elements within a given cluster are similar to each other but dissimilar to those from other clusters. Clustering is useful for tasks such as market segmentation. Dimension reduction models reduce the number of variables in a dataset by grouping similar or correlated attributes for better interpretation (and more effective model training).*[3]*

2.1.3 Scikit-learn

Scikit-learn is an open source software with machine learning library for the Python programming language. It includes several regression, classification and clustering algorithms such as SVM, random forests, gradient boosting and k-means. Scikit-learn is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. Using Scikit-learn tools we get a more accurate implementation of machine learning algorithms as well as various features to analyze our results.*[12] [13]*

2.1.4 Machine learning evaluation methods

In this thesis, one of the objectives is to evaluate the methods used to reach the optimal results. Machine learning methods evaluations can be achieved by tuning the parameters properly. However it can be a difficult task to guess the most suitable parameter for each model. Therefore, a good metric to compare the performances of a model is by comparing the results of several parameters. In this project, Root Mean Square Error (RMSE) for regression and Silhouette score for clustering are used.

- ### 2.1.4.1 Root Mean Square Error (RMSE)

$$\text{Root mean squared error (RMSE)} = \sqrt{\frac{\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2}{N}}$$

Fig:2. RMSE formula

Root-mean-square error also known as *RMSE* is one of the most commonly used metric for regression tasks. This is defined as the square root of the average squared distance between the actual score and the predicted score as shown in *Fig:2*. Where *Yi* is the actual result for the *i-th* data point, and *Yi-hat* is the predicted value for the *i-th* data point. "One intuitive way to understand this formula is that it is the Euclidean distance between the vector of the true scores and the vector of the predicted scores, averaged by N, where N is the number of data points." (*Alice Zheng, 2015*)

- ### 2.1.4.2 Silhouette score

$$a(\bar{x}_i) = \frac{1}{n(j)} \sum_t d(\bar{x}_i, \bar{x}_t) \;\; \forall\, \bar{x}_t \in K_j$$

Fig:3. Silhouette score formula

Silhouette score is used to evaluate clustering algorithm performances, with the formula shown in *Fig:3*."The most common method to assess the performance of a clustering algorithm without knowledge of the ground truth is the *silhouette score*. It provides both a per-sample index and a global graphical representation that shows the level of internal coherence and separation of the clusters." (*Giuseppe Bonaccorso, 2019*)

3. Method and Design

This section includes the approached ideas and models that can possibly solve the problem, as well as a discussion about other possible alternatives. The platform that will be used is Jupyter Notebook and the programming language is Python.

"For data analysis and interactive computing and data visualization, Python will inevitably draw comparisons with other open source and commercial programming languages and tools in wide use, such as R, MATLAB, SAS, Stata, and others. In recent years, Python's improved support for libraries (such as pandas and scikit-learn) has made it a popular choice for data analysis tasks. Combined with Python's overall strength for general-purpose software engineering, it is an excellent option as a primary language for building data applications." *[9]*

Jupyter Notebook platform is a suitable choice for data analysis, the possibility to run each line independently and visualize the data and the figures is very useful. *[9]*

3.1 Overall Design

In order to approach this project it is important to have a well modeled and structured design from the very beginning. The design of this project can possibly be divided into smaller segments as there are several independent steps that needs to be taken to solve it. This makes the project easily debuggable and simpler to modify both during the project or in future developments.

The project preferably should be split into several modules. In most cases, the steps taken to solve the assignment can be treated independently and the results are compared at the end. Each objective can have two modules, one for the cleaning and preparation steps and the other for the machine learning and analyzing steps.

3.2 Modular Approach

To begin with, the project can be split into two main parts, namely bike theft predictions and bike traffic predictions. First step in this project is gathering the data, after that cleaning and preparing them accordingly is a crucial part. Lastly, performing predictions using machine learning methods on the data to predict and evaluate how these data perform should lead to answering the objectives in these project. The project will focus on a single city, the city of Stavanger and therefore all the data involved should be within Stavanger.

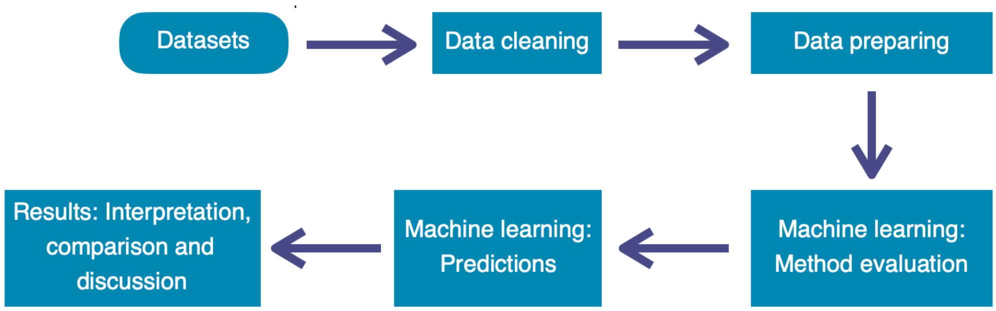

Fig:4. Project stages

Gathering the relevant data is a crucial part of this project as mentioned earlier. A major part of the project is to use and evaluate BikeFinder data to conclude if it is rich enough to be used for predicting future theft and traffic behavior. The BikeFinder data will be gathered from BikeFinder AS, it is expected to be two datasets, one with bike theft reports and the other with bike position points. As mentioned earlier BikeFinder data is sensitive as far as protecting their customers information, thus it is important to anonymize the data. One issue to figure out is then, how to anonymize the data and still give accurate answers. Therefore, for theft data one way to anonymize the position of the theft is to change the position points randomly. This means that rather than using the exact point the bike was parked at before it got stolen, instead have a random position in that area but not exact. This can possibly be done by for instance randomizing the longitude and latitude with a certain range, this way the area is preserved as well as the exact position is unknown.

As for the position data to be used for bike traffic predictions, it is perhaps crucial to use exact points for the bike movement but at the same time maintain the privacy of BikeFinder users. One possible way to solve this could be by checking how often a data is given for a specific bike, to determine whether it was at rest or on the move. The bike being at rest is assumed to probably be at home, work, etc. The idea here is if the bike was at rest then anonymize the data position same way as it was done with the theft data, otherwise use the real data. After observing the data the limit can be set to check how long the previous position was sent prior to the current one thereby determine whether the anonymazation should take place.

In order to conclude whether BikeFinder data perform well or not, it needs to be compared to the performances of other data for the same city. Theft datasets for bicycles should be gathered for the city of Stavanger, and that is possible through a collaboration with Rogaland Police District. As part of this project Rogaland Police District should be contacted and requested to provide the relevant theft data.

Furthermore, position datasets for bike traffic can be gathered from the website of Stavanger commune. There are sensors spread across the city of Stavanger used to count bicycles that happen to pass through those. This is a still sensor somehow different from the moving position data provided by BikeFinder. However, this can be an interesting combination, perhaps the results might be useful to the city for Stavanger municipality to set those sensors in other areas. Weather dataset is another relevant factor to determine and answer whether it affects bike traffic. To gather weather data, online resources can be used such as "*seklima.met.no*". However, data such as weather data, the frequency is mostly taken in an interval of 1 hour minimum, this leads to the next steps data cleaning and preparing.

Since the goal in this project is to focus on the city of Stavanger and given that BikeFinder data contains data across the world it is then reasonable to filter the data as the first step. Limiting the data to Stavanger by filtering data by only taking longitude and latitude within certain range only into the next steps. Limiting the data as first step is reasonable to avoid unnecessary running time and computer resources consumption. Furthermore removing duplicates, handling empty values, re-formatting the data structure, merging and/or splitting the data are all to be done within this part. This part of the project will be adjusted multiple time based on the requirements the further the project advances through the machine learning part.

The field of Machine learning provides a wide range of possibilities and options to choose from, there are several ways one can go about to solve a problem rather than just one. The methods will be implemented using "*scikit-learn*" machine learning library that provides a wide range of machine learning methods. Furthermore, using multiple machine learning methods, as well as evaluating the performances of these using methods such us silhouette score and comparing the *RMSE* of the results to determine what is the suitable approach to take is important part too. Additionally, other things to consider is optimizing the performance of the method as much as possible, this includes the choices of what part of the data to include and exclude.

Making some of the choices should be justified by performing multiple tests or evaluations such as correlation checks. The nature of the data in principle is a spatial data expected as a response variable in the form of longitude and latitude position data. Usually for a single response variable the approach is more straightforward than in Spatial data. In this project one way response variables could be treated as is by handling longitude and latitude values separately and then combining those. However, this could be simpler in clustering as using two variables to perform clustering can be straightforward. Results from clustering can also be interpreted as grouping the areas according to, most likely to be risky for theft or in the other case most likely high traffic.

Date and time are major input data in this project as the aim in this project is being able to predict when and where something happens. However, date and time if used as strings in the standard form the results would not make much sense. Therefore, date and values are expected to be handled before being used. Some possible ways to handle date and time values could be such as splitting them up into several categorical and continuous values. An example could be considering year as a continuous separate value and day of the week a categorical value in the interval 1-7 and for the time the fraction of a day could be taken.

Based on the results one can give several recommendations to when and where in a city there will most likely going either to be bike theft and help prevent it or predict traffic and help the city plan better. This project can also possibly be used for other purposes as well, given how similar situations can be prevented in a larger scale as far as crime is considered in general, or for instance traffic within cars.

3.4 Design Alternatives

There are several possible ways to solve this project, some of which has been suggested in the beginning of the project and others have been considered during the project. The first choice to be done was whether the analysis should be limited to a single city or a larger area. The analysis could have included the entire country of Norway or even globally since BikeFinder data features data from many different countries. Such analysis could have possibly compared the risks involved of biking in some countries than others. This sort of information would have come in handy for travelers. An example would be, getting to predict what season of the year is the most risky for theft in a specific country compared to other and maybe help making decisions based on that. However, doing the research in a smaller area would give a more detailed insight, especially in the beginning before expanding the project further. Focusing on one city and doing it properly in details is the preferred approach. Another reason is focusing on a familiar city such as Stavanger gives a proper insight on the results based on real life observations, given that Stavanger is where BikeFinder AS is based as well as where this project is taking place.

BikeFinder data could be a little smaller in number if limited to one city currently as the company is still growing. That raises the idea of possibly combining it with data obtained from Rogaland Police District for theft or the position data from recorded through the city sensors. Furthermore, compare the results to data from other cities to evaluate the results. However, although the number of data is increasing which is a great thing, but it can't give an accurate answer on the objective of this project to determine whether BikeFinder data is rich enough as it is currently.

Anonymazation of BikeFinder data as mentioned is a central part of the project therefore several techniques were considered such as data masking or data pseudonymization. Some techniques of which can hide the actual data from outside the development environment of the project and others that can possibly modify the data in a less controlled way. However, the approached solution is more suitable to the data gathered from Rogaland Police District and the position data from the city. The theft data from Rogaland Police District was "zoomed out" meaning that instead of the actual position points it contains the area name, similarly to the sensor position data from the city. Generalizing the data where it could be a possible place for a BikeFinder user stationary positions or theft position would make a better comparison than other methods as far as result evaluation is concerned.

4. Implementation

There are seven modules all together implemented in this project:-

- *bf_theft_preprosess.ipynb:* Here BikeFinder theft data is first uploaded and then exported after the cleaning and preparation steps are completed.

- *bf_traffic_preprocess.ipynb:* Here BikeFinder position data and the weather data are uploaded then exported after the cleaning, merging and preparation steps are completed.

- *police_preprocess.ipynb:* Here the theft data from Rogaland Police District is first uploaded and then exported after the cleaning and preparation steps are completed.

- *city_counter_preprocess.ipynb:* Here the city bike counting sensors and the weather data are uploaded and then exported after the cleaning, merging and preparation steps are completed.

- *theft_predictions.ipynb:* Using the data exported from *bf_theft_preprosess.ipynb* and *police_preprocess.ipynb*, here the machine learning and evaluation parts take place on the theft data.

- *traffic_predictions.ipynb:* Using the data exported from *bf_traffic_preprosess.ipynb* and *city_counter_preprocess.ipynb,* here the machine learning and evaluation parts take place on the traffic data.

- *chicago_crime_predictions.ipynb:* In this file data cleaning, preparation as well as machine learning and evaluation parts take place on the Chicago crime dataset.

4.1 Implementation flow

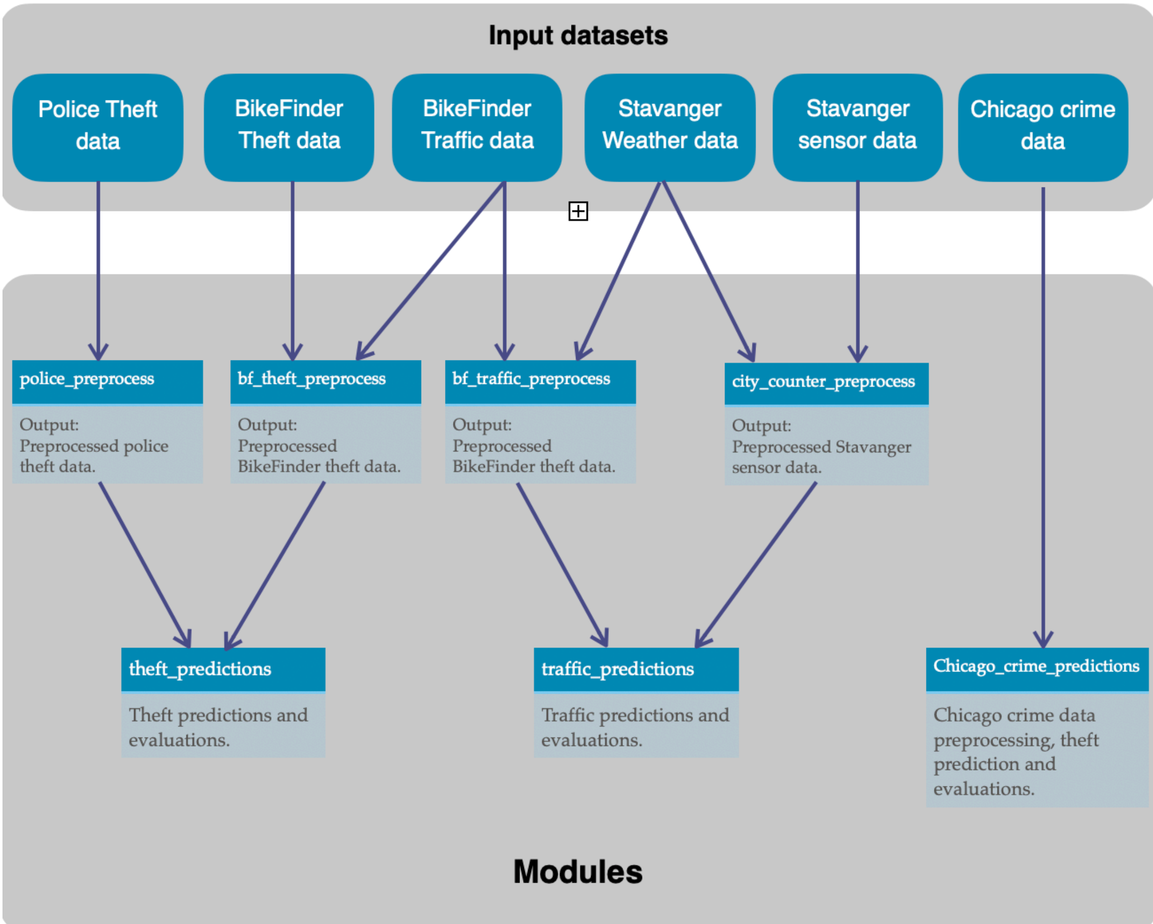

Fig:5. Project flow chart

This Project consists of 7 modules, all in a separate *ipynb* files. It also consists of 6 datasets as mentioned in chapter 4. All the inputs with the exception of Chicago crime dataset, are first processed in the 4 preprocessing modules respectively and then directly used on the prediction modules. Chicago crime data being a bonus addition to the project, everything is used in a single module for this dataset through the whole process.

4.2 Data Extraction

BikeFinder dataset was extracted and provided by BikeFinder AS. A total of two datasets were provided, one that contains thefts reports and another that contains bike positions. These are independent, so it is expected to combine those to have a complete theft dataset. For the Rogaland Police District theft dataset, Rogaland Police District were contacted and requested to provide the desired datasets. Due to security policies it was only possible to get area name for locations instead of latitude and longitude exact positions. The dataset was extracted and provided by Rogaland Police District. Stavanger bike counter sensor data was directly loaded from Stavanger municipality website. The Stavanger bike counter data however was separate for each year, all six from 2017 to 2022 are loaded separately. Weather data filtered and downloaded from *Norsk Klimaservicesenter* website.

4.3 Data Cleaning

Data cleaning is the first step of the implementation. Cleaning the data in this project is mostly about trimming down the data by eliminating undesired data. This part will feature the steps taken to clean the following datasets:

- Theft datasets: BikeFinder theft data and Rogaland Police District theft data
- Traffic datasets: BikeFinder traffic data, Stavanger city bike sensors data and weather datasets.
- Chicago crime dataset.

4.3.1 Theft data cleaning

- 4.3.1.1 BikeFinder theft data cleaning

The gathered BikeFinder Theft dataset consists of 1008 rows and 2 columns, the columns are the following:

deviceid	timestamp

- d*eviceid*: The id of the tracker installed in the stolen bike.
- *timestamp*: The time the theft report was sent by the user.

First step was to check for duplicates using the method *.duplicated()* for *pandas DataFrame,* which then detected 32 duplicates, 32 rows were an exact copy of other rows across both columns. This means the device and timestamp were identical, this could be because the data got stored twice due to a bug, or BikeFinder users reporting multiple times within a second. It can also be due to extracting the same rows from the database multiple times, either way it would not be useful data for this project. Duplicates were then removed and the remaining 976 rows are again checked for duplicates this time only across the d*eviceid* column. As a result 385 duplicate devices are detected. This means there are 591 unique devices with 976 theft reports in total. However, this time this can either be the same case or that simply the same bike got stolen multiple times, which then the data is certainly useful in this project.

In order to make sure this data is valid, defining the following rule might be necessary: In this case only keep data from the same device when it is registered 24 hours after the report registered by the same device prior to it. This is done by first sorting the data by *timestamp* and then by the *deviceid*, such as all the reports from one device is grouped together while sorted from oldest to most recent report. Then a new DataFrame *bf_theft_data_results* is initialized, this will be used to store the results.

```python
# In this section only one theft report per tracker in one day should be taken into account
mins=1440 #minutes in a day
counter=0

# New Dataframe for rsults
bf_theft_data_results = pd.DataFrame(columns=['deviceId','timestamp'])

# Iterate through every row
for i in range(len(bf_Theft_data_dup_droped_sorted)):

    # If Device exists from before in the result dataframe, go in and compare the date differences
    if str(bf_Theft_data_dup_droped_sorted.iloc[i]["deviceId"]) in str(bf_theft_data_results.deviceId):

        #Take the Last date of the existing Device and calculate the differences in the date with the current one
        res=(pd.Timedelta((bf_theft_data_results.loc[bf_theft_data_results["deviceId"]
                                    ==bf_Theft_data_dup_droped_sorted.iloc[i]["deviceId"]]).iloc[-1]['timestamp']
             -bf_Theft_data_dup_droped_sorted.iloc[i]["timestamp"]).seconds/ 60.0)

        #If the differences is less than 1440 mins then store it in the result data frame
        if (res>mins):

            bf_theft_data_results = bf_theft_data_results.append({'deviceId': bf_Theft_data_dup_droped_sorted.iloc[i]["deviceId"]
                                    ,'timestamp': bf_Theft_data_dup_droped_sorted.iloc[i]["timestamp"]}, ignore_index=True)

    #else add it as new value
    else:
        bf_theft_data_results = bf_theft_data_results.append({'deviceId': bf_Theft_data_dup_droped_sorted.iloc[i]["deviceId"]
                                ,'timestamp': bf_Theft_data_dup_droped_sorted.iloc[i]["timestamp"]}, ignore_index=True)


#Print results
bf_theft_data_results
```

Fig:6. Bike theft cleaning code

Thereafter, as shown in *Fig:6*, the code iterates through the 591 data rows in *bf_Theft_data_dup_dropped_sorted* while checking whether the device id exist in *bf_theft_data_results* or not. If the device id does not already exist in *bf_theft_data_results*, then it gets directly stored at *bf_theft_data_results*. Whereas if the device id exist, then the time difference is calculated. The time from the current data row in *bf_Theft_data_dup_dropped_sorted* is subtracted from the last row added to *bf_theft_data_results* for this specific device in minutes. Thereafter, the time difference is checked whether it is more than 1440 minutes (1 day) or less. If the value is larger than 1440 then the current data row is added to *bf_theft_data_results*, otherwise it is ignored. After this process there is 591 rows left remaining. The data is now ready for the next step, data preparation (section: *4.4.1.1*).

- 4.3.1.2 Rogaland Police District theft data cleaning

Theft data from Rogaland Police District consists of 1686 rows and 15 columns. However, most of those columns are not relevant to this project, those columns has the same values through all rows. Columns such as crime type, law chapters, police district , etc, this could be because the dataset was extracted based on those columns among different crime data rows. Therefore, only relevant data among those happen to be police zone, theft date, day of the week and time.

Police zone	Date	Day	Time

- *Police zone*: An area name a bike was stolen at, in Stavanger.
- *Date*: The date the theft accord.
- *Day*: Day of the week the theft took place.
- *Time*: Hour and minute the theft was reported, assumed to be the theft time in this project.

The data provided by Rogaland Police District is mainly from 2019 to 2021 (with 12 reports from 2018) and it does not include the exact latitude and longitude values, instead it has a police zone column which is the area name. After detecting and removing 13 duplicates in addition to two rows due to them being the only ones from 2011 and 2015, 1671 rows are left. In this case, it is not necessary to check whether a report is registered multiple times with different timestamps for the same theft, as it was done with BikeFinder theft reports.

With BikeFinder theft reports a BikeFinder user is able to register reports on their own by just clicking on the report button in the BikeFinder app, thus less controlled. However, in this case the Police register each case as a crime case in a more controlled manner. Therefore, highly unlikely two reports are registered for the same theft with different timestamps. This might raise the question, how come there were duplicates in this case? The reason for this might possibly be that it was an error when retrieving the data. However, detecting whether a theft report was reported multiple times with different timestamps for a single theft, would not be possible in this case anyway. Reason being, there is no unique id attached to each theft report per stolen bike for the Rogaland Police District dataset, thus assuming each report to be independent. The data is now ready for the next step, data preparation (section: *4.4.1.2*).

4.3.2 Traffic data cleaning

- 4.3.2.1 BikeFinder traffic data cleaning

BikeFinder traffic dataset consists of 19833415 rows and 5 columns, the columns are the following:

deviceid	packetType	latitude	longitude	timestamp

- d*eviceid*: The id of the tracker installed in the stolen bike.
- *packetType*: The packet type of the information sent from the tracker (GSM, INI and GPS).
- latitude: The latitude position value.
- *longitude*: The longitude position value.
- timestamp: The time a position was sent from the tracker.

Given how large the data is, first step should be limiting the data to Stavanger only to avoid using unnecessary computer resources on the other steps. This is possible to do as first step, opposed to the BikeFinder theft data since here the position data are included. Null values for latitude and longitude columns are checked and 79817 rows were removed due to not containing either or both values. Thereafter, the latitude and longitude columns are converted to *float* type. Finally, the positions are limited in the ranges (5.585986955209788, 5.773063826295662) for longitude and (58.9180072658198, 58.98768986749389) for latitude, with 794971 rows remaining.

Next step, duplicates are checked and removed leaving 792444 data rows. Using the code *stavanger_position_data['deviceId'].value_counts()*, *stavanger_position_data* being the data frame containing the current dataset, it shows that the 792444 position data are from only 655 trackers. The amount of data the trackers registered varies from 1 position to 19562 positions (rows) per tracker. However, all the data can be relevant because one bike can't be in several locations at once. The focus here is how crowded a location can be which makes it less of an importance which bike it is but more important how many bikes in a location. The 792444 rows are to be used for further preparation (section: *4.4.2.1*).

- 4.3.2.2 Stavanger city sensor traffic data cleaning

Data from Stavanger bike counting sensors, are loaded separately as mentioned in section: *4.2* and merged together. The data consists of 728912 rows and 13 columns. However, most of those columns are not relevant to this project, columns such as station id is not relevant when there is station name. Removed columns such as, Average vehicle length, lane name or even average temperature is going to be eliminated because it was only introduced in later years and is not included in the earlier datasets, thus might be biased to use. Therefore, only relevant data among those happen to be station name, date, time, count.

Station_Name	Date	Time	Count

- *Station_Name*: The area name a bike counter sensor is placed in Stavanger.
- *Date*: The date the counter was used.
- *Time*: Time is a one hour interval.
- *Count*: Number of bikes passed through within the one hour interval.

After removing null values there is 728909 rows remaining. The data should be merged with weather data and further prepared in the next step (section: *4.4.2.2*).

4.4 Data preparing

After cleaning the data individually in the previous step now in this section the data is prepared for the machine learning part. The data is visualized, adjusted and merged depending on the desired objectives.

4.4.1 Theft Data Preparation

- 4.4.1.1 BikeFinder theft data preparation

The BikeFinder theft data as mentioned in the previous step consists of two columns the device id and the timestamp, however an important piece of information is missing. The BikeFinder theft data is missing the theft location. A solution to this is getting the position data from BikeFinder traffic data. Thereby, a number of choices and assumption must be made, however the following are the two main assumptions:

- First assumption is that, the BikeFinder user reported the bike theft immediately after it got stolen, thus that is the time it got stolen.

- Second assumption is that, the latest position of the bike prior to the theft report is where the bike got stolen from.

Taking these assumptions into consideration the next step is to merge the BikeFinder theft and traffic data. This is done by first sorting the position data by device id as well as by

```python
# Merge the latitude and longitude of the devices with the latest time right before sending the notification

counter=0

#new dataframe to store results in
final_theft_data = pd.DataFrame(columns=['deviceId','packetType','latitude','longitude','timestamp'])

# iterate through the theft report data
for i in range(len(bf_theft_data_results)):

    # get all the rows for a theft data device from the traffic data
    latest=bf_position_data.loc[bf_position_data['deviceId'] ==
                                str(bf_theft_data_results.iloc[i]["deviceId"])].sort_values(by='timestamp').reset_index(drop=True)

    #check if there is any or thaat the time of the latest one is before the theft report time
    if not latest.empty and not(latest.loc[latest['timestamp'] < (bf_theft_data_results.iloc[i]["timestamp"])]).empty:

        #if there existed any data and the position before theft report exists, take the latest data before theft report
        temp=(latest.loc[latest['timestamp'] < (bf_theft_data_results.iloc[i]["timestamp"])]).iloc[-1]

        #append the values to the new dataframe
        final_theft_data = final_theft_data.append({'deviceId': temp["deviceId"],'packetType': temp["packetType"]
                ,'latitude': temp["latitude"],'longitude': temp["longitude"]
                ,'timestamp': bf_theft_data_results.iloc[i]["timestamp"]}, ignore_index=True)


        counter = counter + 1

counter
```

Fig:7. Bike theft preparation code

the timestamp such that all devices are grouped together and sorted by the timestamp, oldest being first and latest at the bottom.

Thereafter, a new DataFrame *final_theft_data* is initialized with columns *device_id*, *packetType*, *latitude*, *longitude* and *timestamp* to store the results at. Second step is to iterate through the BikeFinder theft data in *bf_theft_data_results* and check whether there is a traffic data with the same device id if so these will be stored in a temporary dataframe *temp*. The dataframe is then checked first if it is empty or it does not include data points before the report. That is to ensure if there exists any position data for a specific tracker prior to the theft report. If the check passes the latest data prior to the theft report from a specific device is then stored in the result dataframe *final_theft_data* . The process is done for each row in the theft data until 429 rows left in *final_theft_data*.

After limiting the data to Stavanger only by taking data within the range of *(5.585986955209788, 5.773063826295662)* for longitude and *(58.9180072658198, 58.98768986749389)* for latitude, now there are 31 data rows left. Next step is anonymizing the position points, this is done by randomizing the value of the latitude and longitude between the intervals *(x-0.00500, x+0.00500)*. Based on observations the new position data will still be within the same area but large enough that it is not possible to identify any exact addresses. Visualizing the data using the Folium library for Python gives the result in *Fig:8*:

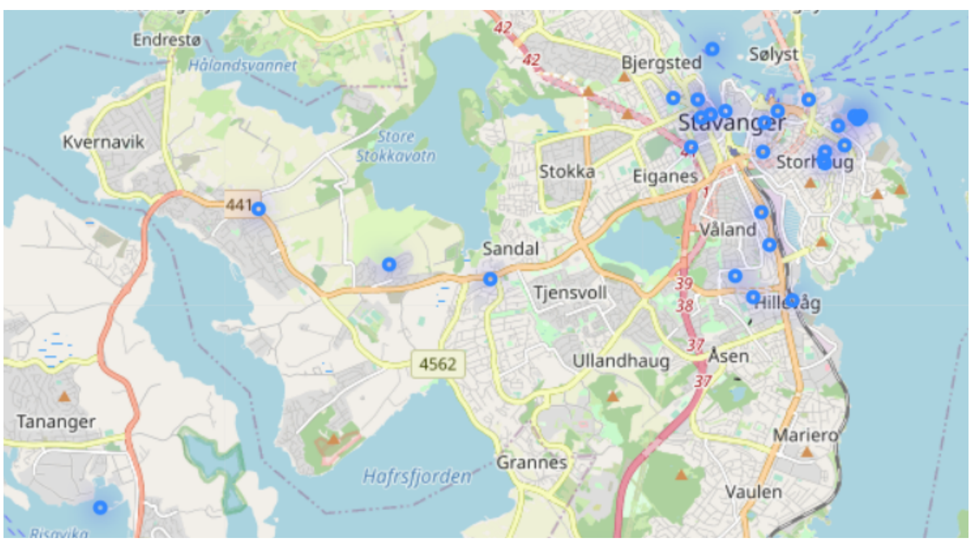

Fig:8. Visualizing BikeFinder theft data

Furthermore, other adjustments were done to the dataset such as maintaining a similar date and time format. Another major thing that was done in this part is finding a way to use date and time data for the purpose of using them as input parameters to predict the cases. Date data are split into different columns, year, month and day separately. Time is split into hour, minutes and seconds.

```python
# convert date_time column to datetime type
final_bf_stavanger_theft_data.theft_time = pd.to_datetime(final_bf_stavanger_theft_data.theft_time)

#split theft into separate columns

# extract hours
hours = final_bf_stavanger_theft_data.theft_time.dt.hour
# extract minutes
mins = final_bf_stavanger_theft_data.theft_time.dt.minute
# extract seconds
sec = final_bf_stavanger_theft_data.theft_time.dt.second

# extract month
year = final_bf_stavanger_theft_data.theft_time.dt.year

# extract month
months = final_bf_stavanger_theft_data.theft_time.dt.month

# extract day of a month
day_of_month = final_bf_stavanger_theft_data.theft_time.dt.day

time_data = pd.DataFrame({
    'year' : year,
    'month' : months,
    'day_of_month' : day_of_month,
    'hour' : hours,
    'minutes' : mins,
    'seconds' : sec

})
final_bf_stavanger_theft_data = pd.concat([final_bf_stavanger_theft_data, time_data], axis = 1)

final_bf_stavanger_theft_data = final_bf_stavanger_theft_data[['latitude', 'longitude','year','month','day_of_month','hour','minutes','seconds']]
final_bf_stavanger_theft_data
```

Fig:9. Splitting the date and time

- 4.4.1.2 Rogaland Police District theft data preparation

The theft dataset from Rogaland Police District required similar steps to the BikeFinder dataset to adjust the date and time data. However, several steps were skipped in comparison. Rogaland Police District dataset was already limited to Stavanger as well as it was not required to be merged with other datasets as it included the report time and position. Although, many steps were skipped but one new issue was, missing longitude and latitude data.

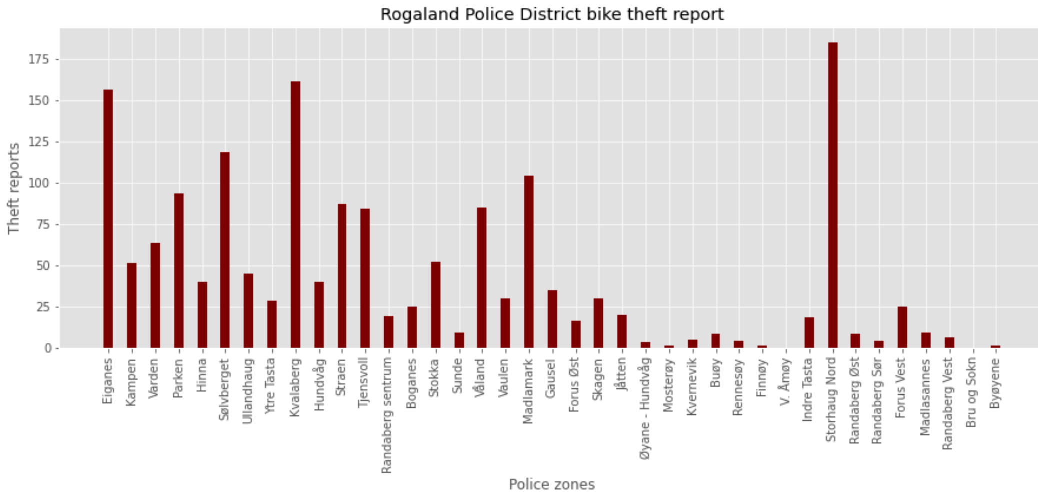

Fig:10. Rogaland Police District theft reports by police zones

To find these points *geopy* library is used, using this library a string of an area name is given as an input and latitude and longitude are returned as outputs. However, some areas were not detected using *geopy*, 10 of the 39 areas required to be manually stored. This is done by iterating through all the possible values, and store the results in a new dataframe. Thereafter, add the 10 position values to the dataframe manually. Afterwards, the new dataframe is merged with the theft data on '*Police zone*' column. There are 1669 theft reports left across 39 areas after removing the missing values.

Finally, time is handled in a similar way as it is done with the BikeFinder theft data shown in *Fig:9*. However, time for this data does not include minutes and seconds, it only includes hours.

4.4.2 Traffic data preparation

- 4.4.2.1 BikeFinder traffic data preparation

After the cleaning process of BikeFinder traffic data, time format should be adjusted. Date and Time is converted into *Pandas datetime* type and then as shown in *Fig:9,* time is split into hours, minutes and seconds. Date is split into year, month and day and then these are merged with the position data. The Date and Time Timestamp column is removed and only the following are taken to the next step:

latitude	longitude	year	month	day_of_month	hour	minutes	seconds

Thereafter, the weather will also be merged with the BikeFinder traffic data by time. However, a different version with weather data is considered since the time for weather conditions data is registered in an hourly interval. Therefore the second version of BikeFinder traffic data is rounded to the nearest hour and then only the hours are extracted. The weather condition data used contains rain and temperature data as well as hours. The weather data and BikeFinder data are then merged on hours, day, month and year. Now there are two versions of the dataset that will be used for the machine learning part, one without weather conditions data and other with weather condition data including minutes and seconds included.

- ### 4.4.2.2 Stavanger city sensor traffic data preparation

The traffic data obtained from Stavanger municipality website exist as separate datasets based on a calendar year. Therefore, first step was to merge all the data from 2017 to 2022 in a single dataframe. The datasets are missing geo-location data in the form of longitudes and latitudes, these were obtained using *geopy* library similarly to the Rogaland Police District theft dataset.

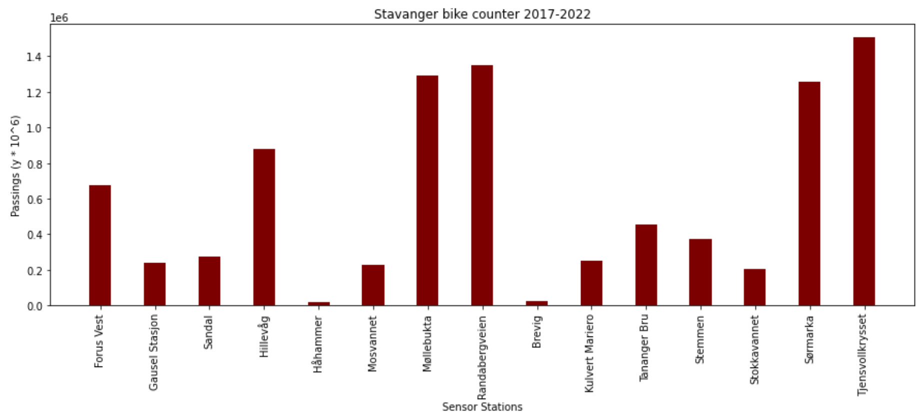

Fig:11. Visualization of the bike traffic by sensors

Furthermore the city sensor time and date data are split into hours, year, month and day. Finally, similar to BikeFinder weather data the data are split into two versions, one merged with weather conditions data and other left as it is.

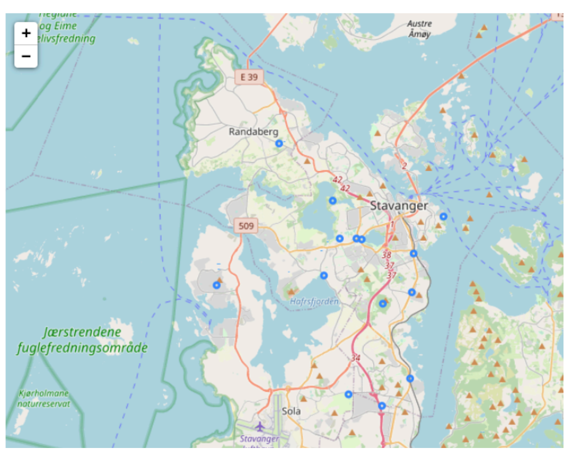

Fig:12. Bike counter sensors around Stavanger

4.4.3 Chicago crime data cleaning & preparation

Chicago crime data consists of 7486655 rows and 22 columns. Firs step is cleaning the data by checking for duplicates, empty values and removing columns that are not of use for this project. There were no duplicates detected, this shows that the data was well controlled when added to *Kaggle*. *[6]*

```
Data columns (total 23 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   ID                    945 non-null    int64
 1   Case Number           945 non-null    object
 2   Date                  945 non-null    datetime64[ns]
 3   Block                 945 non-null    object
 4   IUCR                  945 non-null    object
 5   Primary Type          945 non-null    object
 6   Description           945 non-null    object
 7   Location Description  945 non-null    object
 8   Arrest                945 non-null    bool
 9   Domestic              945 non-null    bool
 10  Beat                  945 non-null    int64
 11  District              945 non-null    float64
 12  Ward                  945 non-null    float64
 13  Community Area        945 non-null    float64
 14  FBI Code              945 non-null    object
 15  X Coordinate          945 non-null    float64
 16  Y Coordinate          945 non-null    float64
 17  Year                  945 non-null    int64
 18  Updated On            945 non-null    object
 19  Latitude              945 non-null    float64
 20  Longitude             945 non-null    float64
 21  Location              945 non-null    object
 22  init                  945 non-null    float64
dtypes: bool(2), datetime64[ns](1), float64(8), int64(3), object(9)
memory usage: 164.3+ KB
```

Fig:13. Chicago crime columns

Date and time values were split into year, month, day, hour, minutes and second similarly to the other datasets. The heat map of crimes in the city of Chicago for the first 100 rows of the dataset is shown in *Fig:14*.

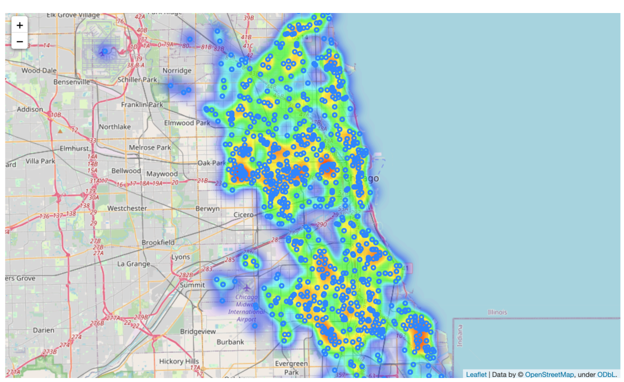

Fig:14. Heat map of the first 100 points

4.5 Machine learning

4.5.1 Method evaluation on Chicago crime data

The evaluation of the method choices is done using Chicago crime data. Reason for this choice is that, BikeFinder datasets are to be compared to the city sensor data and the police data. It would be appropriate to use the same methods for both, and determine the method by testing on a neutral dataset, in this case Chicago crime dataset.

- 4.5.1.1 Regression

To determine which regression method to use between KNN Regression, Random Forest Regression and Decision Tree Regression methods, *RMSE* value will be used. The implementation of the chosen method will be shown in details when used on the theft and traffic datasets.

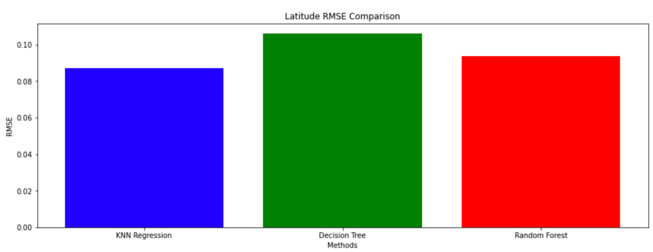

Fig:15. Latitude RMSE comparison

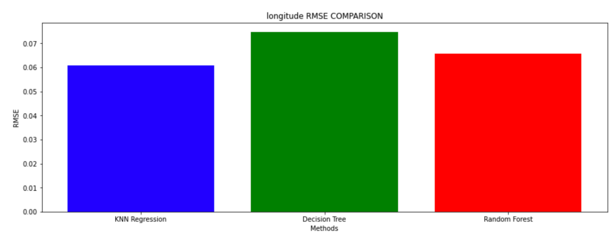

Fig:16. Longitude RMSE comparison

After applying predictions with all three methods on the Chicago dataset, KNN seem to perform better on both latitude and longitude values. Therefore, KNN Regression will be used for predictions.

- 4.5.1.2 Clustering

To determine which clustering method to use between Kmeans and Hierarchcal clustering methods, Silhouette score will be used. The implementation of the chosen method will be shown in details when used on the theft and traffic datasets.

When clustering based on the latitude and longitude, the rest of the data are not going to be used, as the goal is to find out which areas are the most dangerous. In this case, the cluster that includes most position points. Starting with the Hierarchcal method, the dendogram help to observe how the clusters are built. Furthermore, using for-loops to determine and select the best numbers of clusters to use based on the results of the Silhouette score.

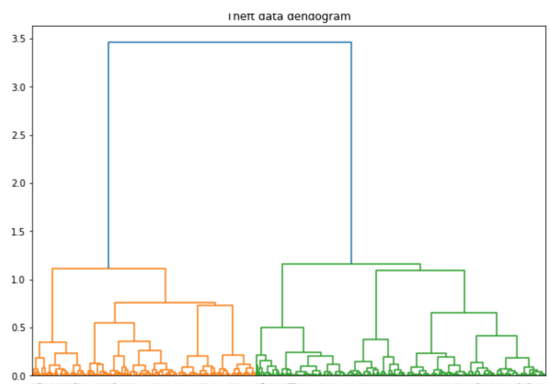

Fig:17. Dendrogram for first 100 rows

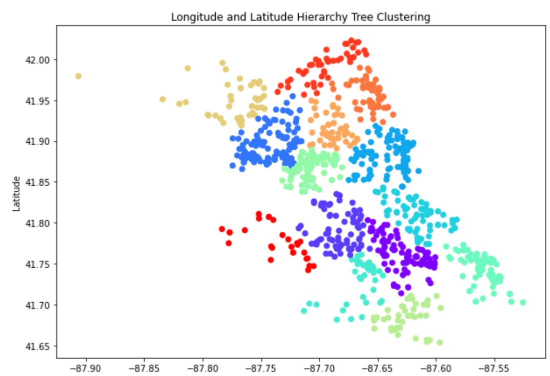

Fig:18. Hierarchal cluster

The scatter plots created to observe the results before and after clustering, as well as printing the data based on each cluster.

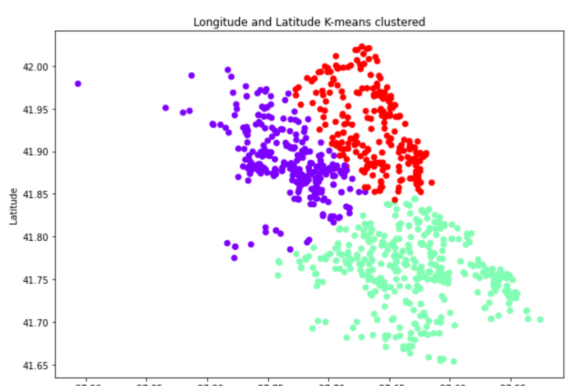

Fig:19. Kmeans clustering

Kmeans Clustering algorithm, the steps to determine the optimal numbers of clusters used here was identical to the one used for the *Hierarchcal* method. The scatter plots were also created in an identical way. The only difference is algorithm used. Based on comparison between different clustering methods used (Hierarchy and Kmeans), they appear to be very similar based on the data results.

The results were identical after observing through both scatter plots and printing the results for every cluster. The plots shows that the points are well grouped based on the given variables. However, using the Silhouette Coefficient for the given methods and their optimal numbers of clusters respectively, the Silhouette Score for *KMeans* clustering was 0.427 and Silhouette Score for Hierarchy Tree clustering 0.387. Therefore, *KMeans* clustering is to be used onto the next step.

4.5.2 KNN regression

All the datasets after the preparation step are now ready to be used for KNN regression using the *Scikit* library, to predict and forecast theft and traffic. The steps taken for all the data is similar to help for comparison. The latitude and longitude data are predicted separately. First the data will be split into train and test data to test the performance by comparing predictions versus the actual data and calculating the *RMSE* value. The data is randomly sorted and split into test and train data, where 20 percent of the data is for testing and 80 percent for training. First goes the latitude data is predicted, therefore it is added to the *y_train* and *y_test* as response variable while both latitude and longitude are removed from *x_train* and *x_test* data as predictors. Then the data are scaled between 0 and 1 to avoid bias results. Next step is to set the model for the KNN Regressor.

```
#Performing KNN and picking the model with the best results
import sklearn.neighbors
from sklearn.neighbors import KNeighborsRegressor
from numpy import sqrt
from sklearn.metrics import mean_squared_error
best_k_latitude = 0
smallest_error = 0

rmse_values_latitude = []
for K in range(10):
    K = K+1
    KNN = sklearn.neighbors.KNeighborsRegressor(n_neighbors = K)

    KNN.fit(x_train_latitude, y_train_latitude)
    pred = KNN.predict(x_test_latitude)
    rmse = sqrt(mean_squared_error(y_test_latitude,pred))
    rmse_values_latitude.append(rmse)


    if best_k_latitude == 0 or smallest_error > rmse:
        best_k_latitude = K
        smallest_error = rmse
        best_predictions_latitude= pred

print('k = ' , best_k_latitude , ', gives the smallest rmse value:', smallest_error)
```

```
k =  10 , gives the smallest rmse value: 0.015846624197014787
```

Fig:20. Choosing best-k and performing KNN

KNN Regressor requires to add a parameter *K* that is the number of neighbors to use, by default it is 5 with sckitlearn. However, to optimize the method as much as possible it has been implemented to check all possibilities from 1 to 10 in a for loop and stop the best result in a variable best_k to be used for the forecasting later and the best prediction is also stored. The check take place in the if loop that checks whether the current RMSE value is smaller than the best one, if it is smaller than the current values K, predictions and the RMSE values are stored and so on. At the end the best predictions and the rise is taken to the next step.All the RMSE values are also stored to be used in a plot in the next step.

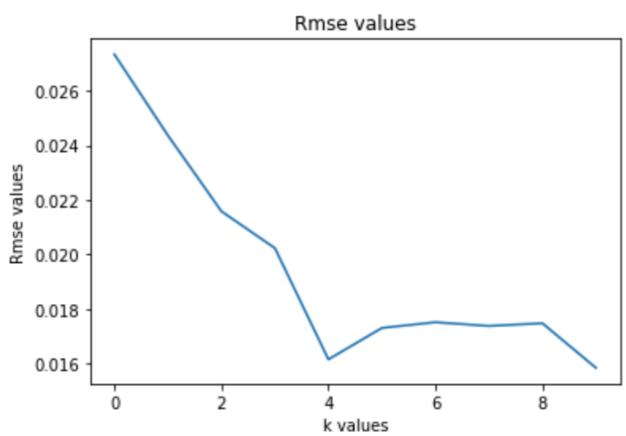

Fig:21. RMSE results for KNN

After that the same steps are done for the longitude values. However, the difference now is that the longitude values are added to *y_train* and *y_test* while both latitude and longitude are removed from *x_train* and *x_test*. When both longitude and latitude are done, the results are added to a new dataframe results as well as the actual test values and both plotted in one plot. The same steps are done to all the data, however traffic data its down twice one with weather and other without.

Next comes forecasting, the period to forecast is for July 2022. Instead of forecasting for a specific time testing the data for each day might be interesting to see and weather somedays are more likely to affect theft in a specific area, weekends for instance. A dataframe with year and month of July 2022 is generated as well as day 1-31, however time is generated randomly.

```
import random

# days of the month july
days_next_month = list(range(1, 32))

# generate a list of the month of july 31 times
month = [7] * 31

# generate a list of the year 2022 31 times
year = [2022] * 31

# generate random numbers for hour, minutes and second
hours = []
mins = []
sec = []

for i in range(0,31):
    hours.append(random.randint(0,23))
    mins.append(random.randint(0,59))
    sec.append(random.randint(0,59))


bf_theft_forecast = pd.DataFrame({'days_next_month': days_next_month,'month': month,'year': year,'hour': hours,'mins': mins,'sec': sec})
#bf_theft_forecast

```

Fig:22. Creating forecasting data

This data is then used to perform KNN regression on, same way it was done previously. The results are then listed and plotted on a map as it will be shown in the results section. The center of the theft positions is also calculated by taking the average values of the longitude results and latitude results.

4.5.3 Clustering

When clustering the rest of the data are not going to be used as longitude and latitude are the only values that are going to affect the results. Therefore Longitude and latitudes are extracted in an array. Next step is to cluster the data using *KMeans* algorithm with *Scikit learn* library. Similar to the KNN Regressor, the goal here is to optimize the result therefore picking the best parameter is crucial. In this case the number of clusters chosen can affect the results.

```
from sklearn.cluster import KMeans #for performing Kmeans
from sklearn.metrics import silhouette_samples, silhouette_score #for silhouette

range_n_clusters = [3,4,5,6,7,8,9,10,11,12,13,14,15]

print("*****Checking for the optimal number of clusters for theft and getting its results.********\n")


best_k=0
largest_silhouette_av = 0
k_theft_cluster_result = 0

for n_clusters in range_n_clusters:


    kmeans_k_theft = KMeans(n_clusters=n_clusters)
    k_theft_clusters=kmeans_k_theft.fit(K_theft)


    k_theft_silhouette_avg = silhouette_score(K_theft, k_theft_clusters.labels_)


    if best_k == 0 or largest_silhouette_av < k_theft_silhouette_avg:
        best_k = n_clusters
        largest_silhouette_av = k_theft_silhouette_avg
        k_theft_cluster_result = k_theft_clusters.labels_


print('n = ' , best_k , ', gives the largest silhouette_avg value:', largest_silhouette_av,"\n")

#**************************************************************************************************************
```

```
*****Checking for the optimal number of clusters for theft and getting its results.********

n =  4 , gives the largest silhouette_avg value: 0.6112164640598284
```

Fig:23. Performing KMeans clustering and choosing best n

This is done by checking different possible options for by iterating with 3-15 clusters. Less than 3 is considered too little to show much information, the default number by *Scikit learn* is 8. The check is quiet similar to the RMSE check implemented on the KNN Regressor, but instead its the *Silhouette* score is checked here, the largest the score the better the results. The results are then stored back in the dataframe and plotted, as well as it is now possible to get the values based on the cluster number. By counting how many positions are in a cluster, one can for instance avoid the areas around a location that belongs to a cluster with the most points. As the chances are higher in the surrounding areas for a bike to get stolen for example.

4.6 Additional feature

Additional step taken in this project was to perform theft prediction using a different dataset. The data from both BikeFinder and Rogaland Police District had some limitations mainly because they were relatively small. In order to achieve results with high level of accuracy and at the same time insightful, a large dataset are required. Therefore, a dataset on crime in Chicago was gathered, the dataset consists of around 7 million rows and 22 columns. It is important to point out that this dataset is not solely for bike theft nor is it only about theft, the dataset is about crime in general in the city of Chicago. The idea for using this is first and foremost to test the performance of the algorithm on a larger dataset. Furthermore, the idea of the data is quite similar and the purpose remains the same, in both cases a crime is reported during a specific time at a specific place. However, the Chicago data gives a more accurate information also because the occurred time is not merged based on assumptions nor is that the position guessed using *GeoLocator*.

This addition can also serve as a reference point to what BikeFinder theft data could be used as in the future with a larger dataset. Therefore, given those circumstances this seem to be an informative addition to the project, as well as an insight to future development of this project with larger data, perhaps focusing on crime in general.

This project can be useful for many different parties therefore, a feature such as finding the center location of the predictions in a map like shown in the image below, would come in handy. The black ring circle in the map represents is the center of the predictions computed by calculating the average of longitude and latitude values.

Combining this feature with a filter to use data only during certain date and time, would be a great addition to city. Having this feature can help the city place their resources such as police, ambulances or other control forms in the center of possible incidents. As for traffic, this could possibly be a cheat code for business especially moving businesses such as food trucks or advertisements, to be placed in a suitable place for that specific day.

5. Testing, Analysis and Results

5.1 Sample runs

5.1.1 BikeFinder theft data

BikeFinder theft data was tested using K-Nearest Neighbors Regression method. 20% of the data were used to be tested and 80% to be trained. *Fig:24* and *Fig:25* are plots of the actual test data vs the predicted data, *Fig:26* shows the results on a table. Clustering was also applied on the latitude and longitude values of the BikeFiner data using *KNN* clustering method, result shown in *Fig:27*.

RMSE results:

Latitude *RMSE*: 0.015846624197014787

Longitude *RMSE*: 0.014206798522780102

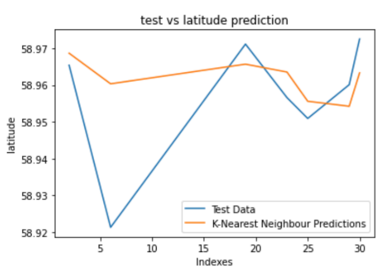

Fig:24. BikeFinder tests prediction for latitude

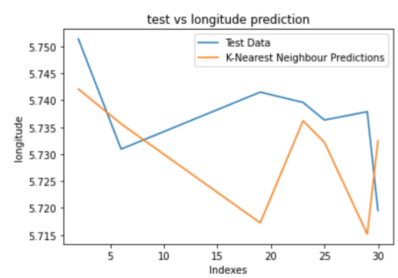

Fig:25. BikeFinder tests prediction for longitude

	Test_Data_latitude	Test_Data_longitude	Predictions_longitude	Predictions_latitude
3	58.921084	5.722016	5.725313	58.953525
4	58.973281	5.728444	5.725383	58.960778
5	58.969685	5.724326	5.711445	58.969499
10	58.972692	5.759425	5.705600	58.961922
17	58.955814	5.733725	5.714883	58.961933
28	58.972072	5.755443	5.732712	58.967898
30	58.975083	5.720755	5.737882	58.965536

Fig:26. BikeFinder tests and prediction data

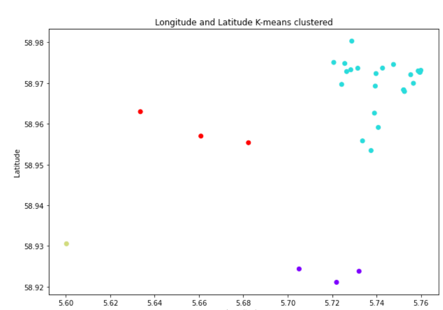

Fig:27. BikeFinder clustering results

5.1.2 Police theft data

Police theft data was tested using K-Nearest Neighbors Regression method. 20% of the data were used to be tested and 80% to be trained same as BikeFinder data. *Fig:28* and *Fig:29* are plots of the actual test data vs the predicted data, *Fig:30* shows the results on a table. Clustering was also applied on the latitude and longitude values of the Police theft data using *KNN* clustering method, result shown in *Fig:31*.

RMSE results:

Latitude rmse: 0.028786957648243457

Longitude rmse: 0.026234226313309236

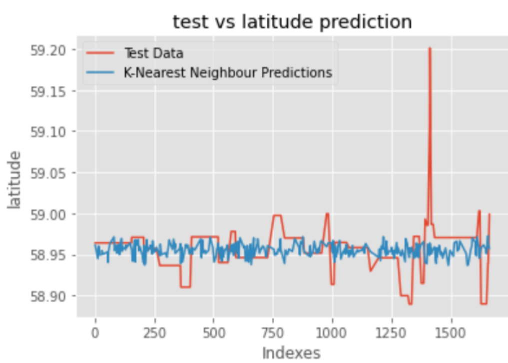

Fig:28. Police tests prediction for latitude

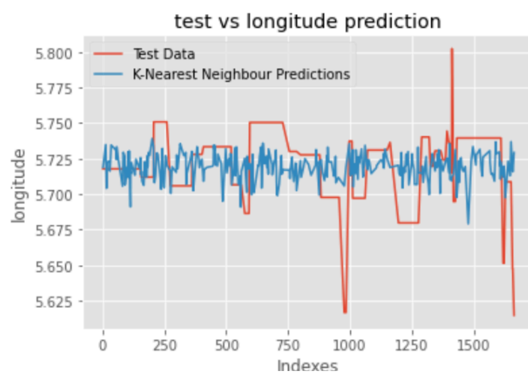

Fig:29. Police tests prediction for longitude

	Test_Data_latitude	Test_Data_longitude	Predictions_longitude	Predictions_latitude
2	58.964115	5.717764	5.718923	58.961423
13	58.964115	5.717764	5.734851	58.945005
16	58.964115	5.717764	5.704107	58.960250
17	58.964115	5.717764	5.717465	58.958242
18	58.964115	5.717764	5.722464	58.949728

Fig:30. Police tests and prediction data

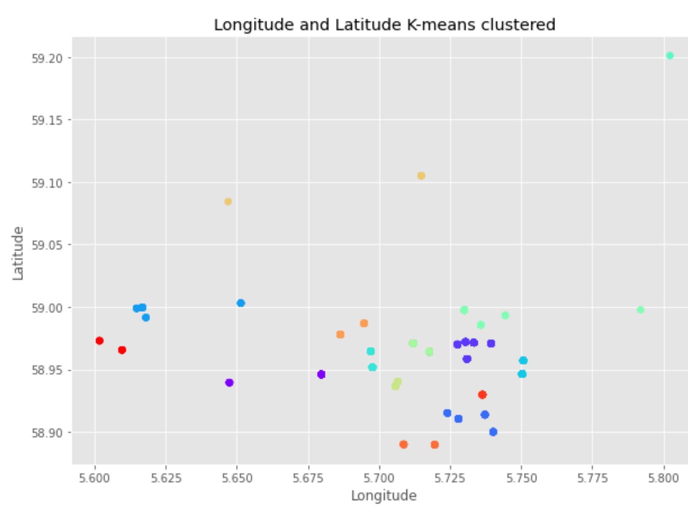

Fig:31. Police clustering results

5.1.3 BikeFinder traffic data

BikeFinder traffic data tested using K-Nearest Neighbors Regression, without weather conditions data included. The data was split 80% to 20% here also.

RMSE results:

Latitude rmse: 0.01207376927744944

Longitude rmse: 0.03305405084070961

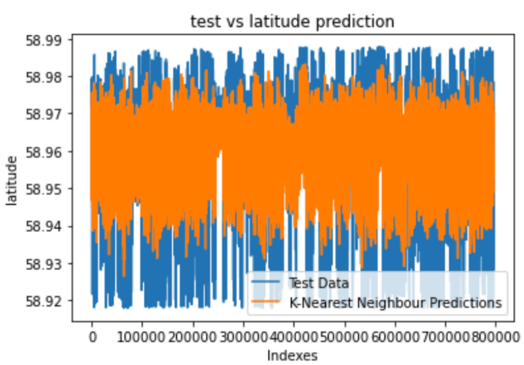

Fig:32. BikeFinder traffic, test vs prediction for latitude

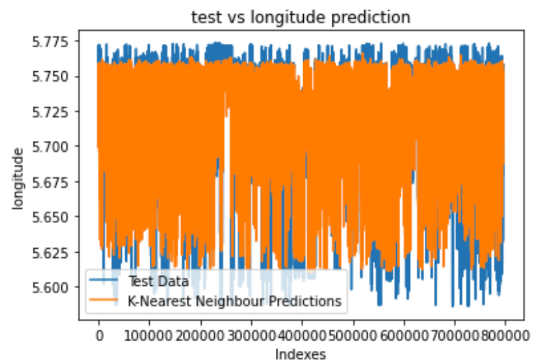

Fig:33. BikeFinder traffic, test vs prediction for longitude

BikeFinder traffic data tested using K-Nearest Neighbors Regression, with weather conditions data included. The data was split 80% to 20% here also.

RMSE results:

Latitude rmse: 0.011122154548081006

Longitude rmse: 0.029283858994079943

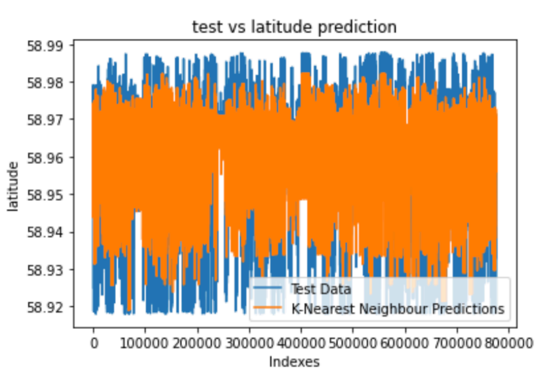

Fig:34. BikeFinder traffic, test vs prediction for latitude with weather

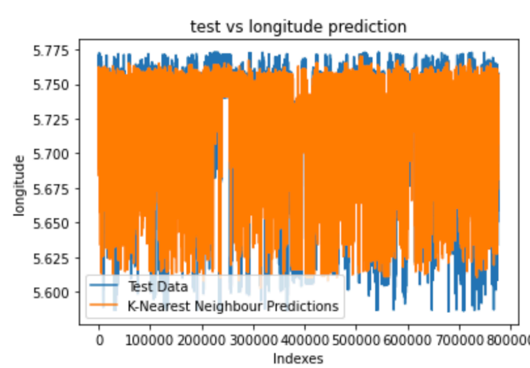

Fig:35. BikeFinder traffic, test vs prediction for longitude with weather

5.1.4 Stavanger traffic data

Stavanger traffic data tested using K-Nearest Neighbors Regression, without weather conditions data included. The data was split 80% to 20% here also.

RMSE results:

Latitude rmse: 0.06437473931032374

Longitude rmse: 0.06563877833974363

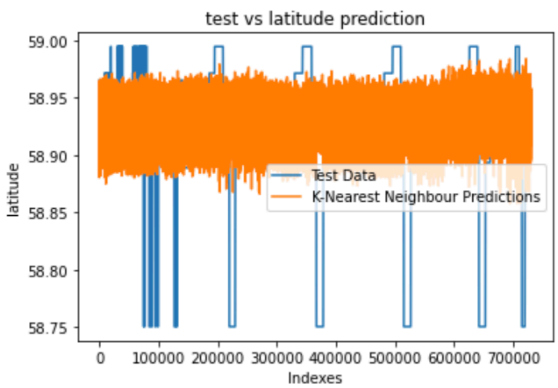

Fig:36. Stavanger traffic, test vs prediction for latitude

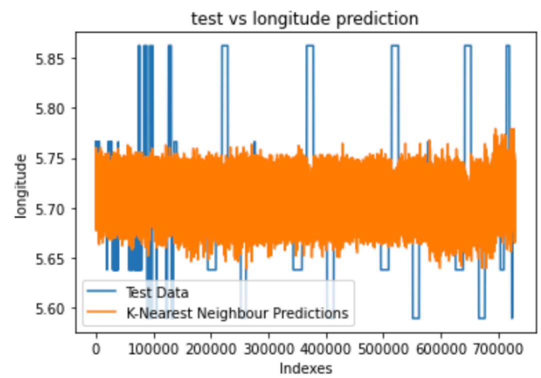

Fig:37. Stavanger traffic, test vs prediction for longitude

Stavanger traffic data tested using K-Nearest Neighbors Regression, with weather conditions data included. The data was split 80% to 20% here also.

RMSE results:

Latitude rmse: 0.06416158084185032

Longitude rmse: 0.0654394075032857

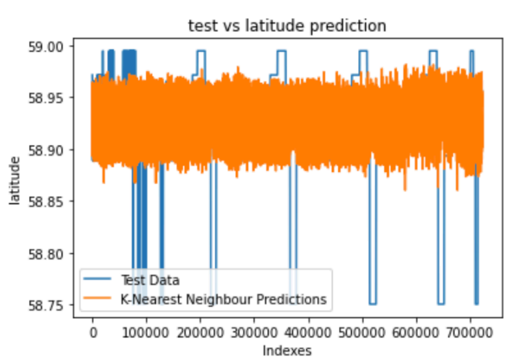

Fig:38. Stavanger traffic, test vs prediction for latitude w/weather

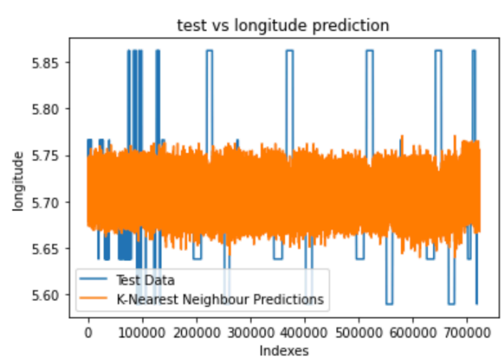

Fig:39. Stavanger traffic, test vs prediction for longitude w/weather

5.1.5 Theft forecasting results

Forecasting for theft for July 2022 using BikeFinder data (One prediction per day)

	forecast_prediction_longitude	forecast_prediction_latitude	days_next_month	month	year	hour	mins	sec
0	5.688758	58.959198	1	7	2022	7	20	6
1	5.686742	58.956437	2	7	2022	0	49	25
2	5.688806	58.958736	3	7	2022	8	9	20
3	5.702684	58.954610	4	7	2022	8	51	18
4	5.686742	58.961342	5	7	2022	0	53	29
5	5.692514	58.955946	6	7	2022	6	22	21
6	5.686742	58.961045	7	7	2022	7	56	55
7	5.699728	58.962528	8	7	2022	3	40	17
8	5.701814	58.959738	9	7	2022	17	50	23
9	5.700856	58.963727	10	7	2022	2	48	26
10	5.705751	58.956609	11	7	2022	16	9	25
11	5.694438	58.959027	12	7	2022	0	2	28
12	5.698941	58.956501	13	7	2022	11	56	9
13	5.692699	58.960223	14	7	2022	15	31	52
14	5.698128	58.956960	15	7	2022	10	6	50
15	5.694044	58.957261	16	7	2022	6	6	34
16	5.697797	58.956960	17	7	2022	12	25	44
17	5.701514	58.959534	18	7	2022	6	12	0
18	5.701429	58.959315	19	7	2022	4	23	2
19	5.694651	58.958677	20	7	2022	2	37	56
20	5.694044	58.956960	21	7	2022	8	1	44
21	5.695960	58.958631	22	7	2022	19	19	38
22	5.718485	58.954614	23	7	2022	14	37	0
23	5.700444	58.959153	24	7	2022	18	22	1
24	5.693029	58.959087	25	7	2022	10	19	16
25	5.708455	58.957241	26	7	2022	9	51	28
26	5.708455	58.958313	27	7	2022	4	55	21
27	5.708455	58.957628	28	7	2022	5	36	24
28	5.708455	58.958313	29	7	2022	4	49	16
29	5.689745	58.958745	30	7	2022	6	12	57
30	5.714018	58.958631	31	7	2022	19	2	36

Fig:40. BikeFinder theft, forecasting results

Forecasting for theft for July 2022 using Rogaland Police District data (One prediction per day)

0	5.716065	58.958962	1	7	2022	4
1	5.716065	58.958962	2	7	2022	2
2	5.714975	58.961393	3	7	2022	0
3	5.717419	58.948483	4	7	2022	20
4	5.716065	58.958962	5	7	2022	4
5	5.714975	58.961393	6	7	2022	1
6	5.714975	58.961393	7	7	2022	1
7	5.707728	58.951008	8	7	2022	21
8	5.716065	58.958962	9	7	2022	3
9	5.711566	58.948549	10	7	2022	23
10	5.716492	58.946200	11	7	2022	16
11	5.716978	58.963255	12	7	2022	0
12	5.718395	58.959557	13	7	2022	10
13	5.712693	58.955126	14	7	2022	18
14	5.716989	58.960842	15	7	2022	20
15	5.725517	58.959407	16	7	2022	4
16	5.722206	58.965787	17	7	2022	15
17	5.719499	58.960743	18	7	2022	8
18	5.716994	58.962700	19	7	2022	10
19	5.727054	58.966589	20	7	2022	23
20	5.724522	58.965890	21	7	2022	12
21	5.717934	58.968660	22	7	2022	18
22	5.724522	58.965890	23	7	2022	12
23	5.728597	58.962618	24	7	2022	7
24	5.727054	58.966589	25	7	2022	23
25	5.727616	58.953034	26	7	2022	12
26	5.725017	58.945420	27	7	2022	11
27	5.727210	58.955281	28	7	2022	8
28	5.721790	58.937486	29	7	2022	14
29	5.719878	58.970760	30	7	2022	0

Fig:41. Police theft, forecasting results

The fitted K-Nearest Neighbor models with BikeFinder theft data and Rogaland PoliceDistrict theft data has been used to forecast theft using future date and time. The data to be forecasted are dates for July 2022 and randomized time of the day.

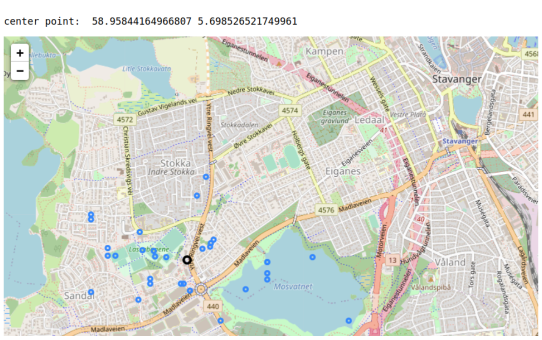

Fig:42. BikeFinder theft, forecasting on map

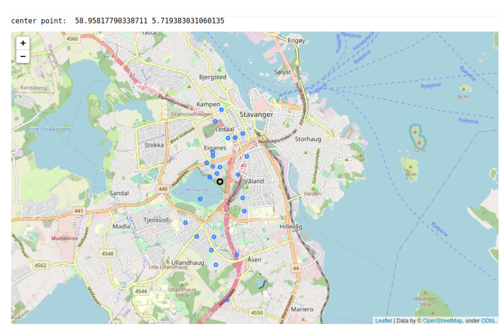

Fig:43. Police theft, forecasting on map

5.1.6 Traffic forecasting results

Forecasting for traffic for July 2022 using BikeFinder data (One prediction per day)	Forecasting for forecasting for July 2022 using Stavanger city sensor data (One prediction per day)

0	5.737045	58.971963	1	7	2022	6
1	5.727579	58.972740	2	7	2022	18
2	5.733163	58.974140	3	7	2022	23
3	5.719846	58.973188	4	7	2022	16
4	5.718437	58.973774	5	7	2022	10
5	5.719081	58.954335	6	7	2022	18
6	5.719081	58.954335	7	7	2022	18
7	5.719081	58.954335	8	7	2022	18
8	5.742005	58.955535	9	7	2022	20
9	5.725607	58.955269	10	7	2022	10
10	5.729521	58.956813	11	7	2022	1
11	5.714911	58.953893	12	7	2022	23
12	5.720622	58.953309	13	7	2022	15
13	5.720622	58.953309	14	7	2022	15
14	5.729098	58.957762	15	7	2022	21
15	5.687267	58.959741	16	7	2022	19
16	5.720169	58.965826	17	7	2022	1
17	5.715420	58.949888	18	7	2022	8
18	5.637653	58.945678	19	7	2022	12
19	5.715420	58.949888	20	7	2022	8
20	5.705182	58.962379	21	7	2022	14
21	5.708588	58.956849	22	7	2022	13
22	5.708588	58.956849	23	7	2022	13
23	5.705182	58.962379	24	7	2022	14
24	5.705133	58.950242	25	7	2022	15
25	5.700963	58.939196	26	7	2022	17
26	5.734280	58.951731	27	7	2022	3
27	5.730901	58.958562	28	7	2022	4
28	5.743921	58.957029	29	7	2022	20
29	5.706182	58.953702	30	7	2022	1
30	5.723786	58.961584	31	7	2022	13

Fig:44. BikeFinder traffic, forecasting results

	forecast_prediction_longitude	forecast_prediction_latitude	day	month	year	hour
0	5.703351	58.933101	1	7	2022	16
1	5.723357	58.914726	2	7	2022	14
2	5.691319	58.946540	3	7	2022	9
3	5.680768	58.960530	4	7	2022	21
4	5.695583	58.954926	5	7	2022	8
5	5.687088	58.956484	6	7	2022	6
6	5.704546	58.929589	7	7	2022	18
7	5.693485	58.961417	8	7	2022	23
8	5.703241	58.943591	9	7	2022	6
9	5.704546	58.929589	10	7	2022	18
10	5.692455	58.961332	11	7	2022	17
11	5.698820	58.959513	12	7	2022	21
12	5.743163	58.896246	13	7	2022	8
13	5.663229	58.957205	14	7	2022	2
14	5.694200	58.915735	15	7	2022	11
15	5.693504	58.947904	16	7	2022	6
16	5.702467	58.928994	17	7	2022	10
17	5.717492	58.892731	18	7	2022	2
18	5.710929	58.923003	19	7	2022	16
19	5.698279	58.938347	20	7	2022	11
20	5.699547	58.944063	21	7	2022	6
21	5.678650	58.960834	22	7	2022	23
22	5.701083	58.958030	23	7	2022	18
23	5.714076	58.959381	24	7	2022	7
24	5.700337	58.955050	25	7	2022	17
25	5.700337	58.955050	26	7	2022	17
26	5.708501	58.953963	27	7	2022	13
27	5.714572	58.934564	28	7	2022	17
28	5.704385	58.943234	29	7	2022	18
29	5.710200	58.942654	30	7	2022	6
30	5.689151	58.957500	31	7	2022	19

Fig:45. Stavanger traffic, forecasting results

The fitted K-Nearest Neighbor models with BikeFinder traffic data and Stavanger city sensors data has been used to forecast traffic using future date and time. The data to be forecasted are dates for July 2022 and randomized time of the day.

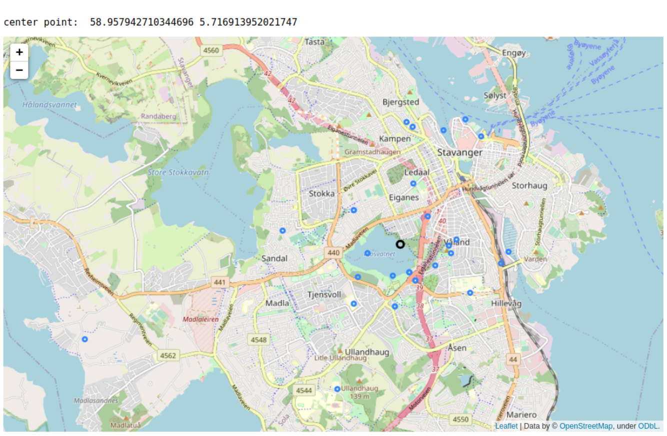

Fig:46. BikeFinder traffic, forecasting on map

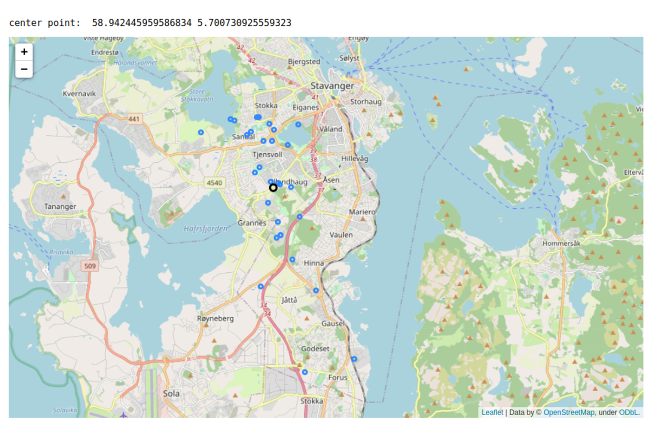

Fig:47. Stavanger traffic, forecasting on map

5.2 Data used

A number of datasets are used in this project, the main ones are the BikeFinder data as the project revolves around it. Two BikeFinder datasets were gathered one for theft reports that consisted of 1,008 rows and 2 columns. The other with position data that consisted of 19,833,415 rows and 5 columns, both datasets contained data from 2019 to 2022. In order to get theft positions it was required to merge position data with theft data. After limiting the data to Stavanger, eliminating duplicates and removed missing information, theft data contained 31 cases while position data became 794,971 rows.

Theft data from Rogaland Police District was a single dataset that consisted of 1686 rows and 15 columns in total after removing duplicates it was reduced to 1673 rows. It was challenging to gather this dataset as it required to send multiple emails, long waiting time for responses and even meeting up in person at the Police station in order to gather the dataset. As a result it was gathered at late stage into the thesis time, which then resulted in the idea to find a similar dataset such as the dataset for crime in Chicago from *Kaggle* to work with in the meantime.

Chicago crime dataset with 6.99m rows and 22 columns is the largest dataset in this project. The dataset contains data about crimes in general registered in Chicago, the dataset is from 2001 to present. According to the dataset description in kaggle approximately 10 people are shot on an average day in Chicago, which gives an idea on how the dataset is this large.

Weather data was gathered from "*Norsk KLIMASERVICE SENTER*" website, the data consists of around 47,242 rows of hourly weather data from 2017 up to now, with columns such as rain, temperature and date. *[11]* The location of this measurement is Stavanger-Våland, as it is there where the sensor is placed. The data was limited to 2017 - 2022 in order to contain data within the range of BikeFinder position data as well as the data city bike counter sensor.

The data for bike counter sensors from the city of Stavanger is gathered from the Stavanger municipality website that contains a set of datasets. The datasets are a registry of how many bikes has passed through the sensors that are placed around the city. A total of six datasets were obtained with data from 15 sensors and merged together resulting in 723121 rows and 10 columns.*[14]*

5.3 Result Analysis

The BikeFinder data has a great potential to be used in many ways to gain insight on bike traffic and bike theft. The advantage of the BikeFinder data is that it is not gathered from a stationary position such as the city bike counter sensors. This makes BikeFinder data more attractive to perform such researches and analyses as it gives a more insightful and realistic data on bike movements. However, The city bike counting sensors have the advantage on the amount of different bikes it registers. BikeFinder data in the other hand only includes bikes with BikeFinder tracker installed. In some cases, there is not a big number of those in a single city, a single bike can send many positions which might then result in a bias outcome.

BikeFinder theft data does have the potential to be used as predictor data, however as it is currently the amount of data for the city of Stavanger is too small to be a reliable predictor. A data as small as this can cause overfitting. When a model is overfitted, it is not reliable to use for forecasting as it is not going to work accurately when other different data are to be predicted. Thus, the model is not exposed to enough adversity to learn from. However, it can be seen in the results from sections *5.1.1* and *5.1.2* that BikeFinder data has lower *RMSE* values although the data size is significantly less than Rogaland Police District data. Another reason that plays a big role other than overfitting is that BikeFinder data has two more columns that Rogaland Police District data don't, namely minutes and seconds. This two values can affect the results positively as well to certain degree, it helps to get a more accurate outcome.

Based on the results of traffic forecasting from BikeFinder at section *5.1.6* and the *Fig:12 & Fig:47* the sensor placements for Stavanger city bike counting sensors it is possible to see that the sensors are placed around the city of Stavanger perhaps to count how many bike in and out the city. However, if the city is not counting most of the movements inside the city as it can be seen from the forecasting results at section *5.1.6.* The city sensors results look different in comparison to BikeFinder traffic forecasting results. There is more predictions around the city center including areas such as Bjergstad, Kampen and Eiganes where possibly many bikers might live and for instance use bikes as a transportation option in the city. The city bike counting sensors surrounding Stavanger will fail to take those into consideration. Thus, it might affect the city judgments on what routes need improvements or needs to be added. The sensors might also fail to count bikes traveling in and outside the city obviously if a biker uses other transportation means to move the bike in and out the city such as trains or busses. A suggestion is to have the sensors closer to the city center. Placing them at places where people around the city center live, between the city center and routes such as Storhaug, Bjergstad, Eiganes, Våland Hundvågtunnelen. This way the city would have a better insight on how bike traffic is.

As shown *RMSE* comparison results in sections *5.1.3* and *5.1.4*, for both BikeFinder traffic data and Rogaland Police District data, the *RMSE* value is lower when weather conditions data is included. This indicates that the predictions are more accurate when weather condition data is given, thus the weather does affect bike traffic. From the results at *5.1.4* bike traffic forecast for July 2022, if it was taken several predictions per day and time it can give an insight to for instance transportation companies. If it were to be expected much traffic around certain area a time of the year, month or day, the company can for example have less busses around that area.

6. Discussion

6.1 Originality of this work

In this thesis bike traffic and bike theft prediction for the city of Stavanger was achieved. BikeFinder data has been explored, evaluated and for the first time used for prediction purposes. BikeFinder location data were successfully anonymized to protect the BikeFinder users privacy, but at the same time the data was used as desired to achieve the objectives for theft predictions and traffic predictions. Forecasting theft with both BikeFinder data and Rogaland Police District data were achieved to help bikers avoid those place. In this thesis it was also given suggestions on where would the ideal place be to have control over possible theft was also given to sides such as the police.

Traffic predictions using weather data was also achieved and concluded that weather conditions does affect traffic, based on the data used. The city stationary sensor data and BikeFinder traffic data both can be useful predictors. The city sensors data provides data of a wider range of bikes that can result in less bias results. BikeFinder traffic data can give more detailed bike movements as well as it can provide insightful information to the city on where to place their sensors.

This thesis for the most part had freedom of choices, there wasn't a specific way the project needed to be approached, as long as it served its purpose. This means all the "hows" were up to me to decide, such as which data to use on top of BikeFinder data and methods to perform forecasting and evaluations. However, my supervisors provided very valuable guidance through the entire process and redirected me in time if the project were to go in the wrong direction. The project was taken step by step, each problem was solved through several tests with different models. Each choice has been made after comparing the test results.

This is the first project that involved BikeFinder data to be used to forecast potential theft and/or traffic, such feature would be a great addition for BikeFinder AS users, perhaps on the BikeFinder app. The project was successfully completed despite facing several challenges in gathering police theft dataset which required several attempts and almost half the thesis period. Also, failed attempts in gathering public transportation data after several requests, despite that it is safe to say the project achieved the objectives.

6.2 Further work

There are several possibilities for the growth of this project depending on the objectives to be achieved. With more resources and datasets accessibility larger datasets can be used to achieve more accurate results. Different datasets also can be added such as data about public transportation. Data like public transportation can provide insight to public transportation companies. Based on the correlation between how busy public transportation gets and bike traffic in some areas. This sort of information can save the companies money if they use the results to allocate their resources based on traffic predictions. The company can for instance have less busses in an area when bikes are expected to be used more.

This project can also be extended to more than just bikes. The project can focus on crimes generally and for instance add data about holidays, festivals etc. In this case the police can use this information to allocate their resources accordingly. Another party that can use this, hospitals and health care organizations, for instance an ambulance can be placed at areas were crime is expected during a certain time.

Traffic predictions can also be used for more than just bikes, it can be used for cars or other places where one is trying to avoid waiting time. The opposite is also possible, some companies might need to know where there is traffic to target potential customers.

The project would be a very useful tool if a User Interface were to be developed. A user can the either insert the date and time to predict and then the predictions displays on a map. A possibility to insert a dataset with date and time in a specific format as input, would also be an option for other more advanced features. Another possibility, instead of making the user insert anything, pre forecast for the day or the week, similar to weather forecasting. Also, maybe display the results on a map using a sliding switch that works as a moving window.

The further the project expands with larger data, the computing time as well as resources used can be very large, that being said this gives the opportunity to put a use for distributed processing systems such as *Hadoop* and *Spark*.

7. Acknowledgments

I would like to appreciate my supervisor Professor Reggie Davidrajuh and co-supervisor Daniel Barati, for the valuable guidance, frequent feedbacks and encouragements throughout the master thesis. I would also like to thank my family for all the support.

References

[1]. A bike is stolen every 16 minutes in London and there's a 98 per cent chance you'll never see it again. Available online: https://www.cyclingweekly.com/news/a-bike-is-stolen-every-16-minutes-in-london-and-theres-a-98-per-cent-chance-youll-never-see-it-again (accessed on 12 June 2022).

[2]. Alice Zheng (2015), Evaluating Machine Learning Models. O'Reilly Media, Inc..

[3]. Andreas C. Müller & Sarah Guido (2016), Introduction to Machine Learning with Python: A Guide for Data Scientists. O'Reilly Media, Inc.

[4]. Bicycle vs car production, Available online: https://www.worldometers.info/bicycles (accessed on 12 June 2022).

[5]. BikeFinder, Available online: https://www.bikefinder.com (accessed on 12 June 2022).

[6]. Chicago Crime, Available online: https://www.kaggle.com/datasets/chicago/chicago-crime?select=crime (accessed on 12 June 2022).

[7]. Garry Kasparov (2017), Deep Thinking: Where Machine Intelligence Ends and Human Creativity Begins. John Murray Publishers Ltd.

[8]. Giuseppe Bonaccorso (2019), Hands-On Unsupervised Learning with Python. Packt Publishing.

[9]. McKinney, W. (2017), Python for data analysis : data wrangling with Pandas, NumPy, and IPython. Beijing, O'Reilly.

[10]. Mirrorpix (2017), The Bicycle: 200 Years on Two Wheels. United Kingdom, The History Press Ltd..

[11]. NORSK KLIMA SERVICE SENTER, Available online: https://seklima.met.no/observations (accessed on 12 June 2022).

[12]. Richert, W. (2015), Building machine learning systems with Python : get more from your data through creating practical machine learning systems with Python. Birmingham, Packt Publishing.

[13]. Scikit-Learn. Available online: https://scikit-learn.org (accessed on 12 June 2022).

[14]. Stavanger Sykkeldata, Available online: https://open.stavanger.kommune.no/dataset/sykkeldata (accessed on 12 June 2022).

Appendix-A

A1: Complete code:

BikeFinder traffic data pre-processing

bf_traffic_preprocess.ipynb

Dawit H. Kidane, 15.june.2022

Importing Libraries

In [ ]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from datetime import timedelta
import folium
from folium import plugins
from folium.plugins import MarkerCluster
import warnings
warnings.filterwarnings("ignore")
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Loading and exploring the Bikefinder traffic data

In [ ]:
```
#Load the BikeFinder bike position data
bf_position_data = pd.read_csv('final_deviceLocationData.csv')
bf_position_data.shape
```

In [ ]:
```
#check for undefined latitude and/or longitude values
print((bf_position_data['latitude']=='undefined').sum())
print((bf_position_data['longitude']=='undefined').sum())

#remove undefined latitude and/or longitude values
bf_position_data = bf_position_data[bf_position_data.latitude != 'undefined']
bf_position_data = bf_position_data[bf_position_data.longitude != 'undefined']

print((bf_position_data['latitude']=='undefined').sum())
print((bf_position_data['longitude']=='undefined').sum())

# convert longitude and latitude values from string to float
bf_position_data['longitude'] = bf_position_data['longitude'].astype(float)
bf_position_data['latitude'] = bf_position_data['latitude'].astype(float)
```

In [ ]:
```
#Limit the data to Stavanger only
stavanger_position_data = bf_position_data[bf_position_data['longitude'].between

stavanger_position_data = stavanger_position_data[stavanger_position_data['latit
stavanger_position_data
```

In [ ]:
```
#check for duplicates
print(stavanger_position_data.duplicated().value_counts())

#drop duplicates
stavanger_position_data = stavanger_position_data.drop_duplicates()
stavanger_position_data
```

[redacted]

In [ ]:
```
#Convert Time format to 'month/day/year hour:minute:second'

stavanger_position_data['timestamp'] = pd.to_datetime(stavanger_position_data['t
stavanger_position_data['timestamp'] = pd.to_datetime(stavanger_position_data['t
stavanger_position_data['timestamp'][0]
```

In [ ]:
```
#Trackers that has over a 1000 positions registered
stavanger_position_data['deviceId'].value_counts()#.loc[lambda x : x>10]
```

In [ ]:
```
#anonymizations
stavanger_position_data['longitude']=random.uniform((stavanger_position_data['lo
stavanger_position_data['latitude']=random.uniform((stavanger_position_data['lat
```

In [ ]:
```
#split traffic date and time data into separate columns

# extract hours
hours = stavanger_position_data.timestamp.dt.hour
# extract minutes
mins = stavanger_position_data.timestamp.dt.minute
# extract seconds
sec = stavanger_position_data.timestamp.dt.second

# extract month
year = stavanger_position_data.timestamp.dt.year

# extract month
months = stavanger_position_data.timestamp.dt.month

# extract day of a month
day_of_month = stavanger_position_data.timestamp.dt.day

time_data = pd.DataFrame({
    'year' : year,
    'month' : months,
    'day_of_month' : day_of_month,
    'hour' : hours,
    'minutes' : mins,
    'seconds' : sec

})
final_bf_stavanger_position_data = pd.concat([stavanger_position_data, time_data

final_bf_stavanger_position_data = final_bf_stavanger_position_data[['latitude',
final_bf_stavanger_position_data
```

In [ ]:
```
#export preprocessed data to  be used for machine learning part.
final_bf_stavanger_position_data.to_csv('bf_traffic_preprocessed.csv', index=Fal
```

BikeFinder traffic data with weather data

In [ ]:
```
#importing weather data
weather = pd.read_excel('table.xlsx')
with_weather_data = stavanger_position_data
```

In [ ]:
```
with_weather_data
```

[redacted]

```
In [ ]: with_weather_data['hours'] = with_weather_data.timestamp.dt.hour
```

```
In [ ]: with_weather_data
```

```
In [ ]: weather['Tid(norsk normaltid)'] = pd.to_datetime(weather['Tid(norsk normaltid)']
        weather['Tid(norsk normaltid)'] = pd.to_datetime(weather['Tid(norsk normaltid)']
        weather['Tid(norsk normaltid)'][0]
```

```
In [ ]: weather.rename(columns = {'Tid(norsk normaltid)':'timestamp'}, inplace = True)
```

```
In [ ]: weather=weather.dropna()
        weather['hours'] = weather.timestamp.dt.hour.astype(int)
        weather['year'] = weather.timestamp.dt.year.astype(int)
        weather['month'] = weather.timestamp.dt.month.astype(int)
        weather['day'] = weather.timestamp.dt.day.astype(int)
        weather
```

```
In [ ]: with_weather_data['hours'] = with_weather_data.timestamp.dt.hour
        with_weather_data['year'] = with_weather_data.timestamp.dt.year
        with_weather_data['month'] = with_weather_data.timestamp.dt.month
        with_weather_data['day'] = with_weather_data.timestamp.dt.day
        with_weather_data
```

```
In [ ]: final_with_weather=pd.merge(with_weather_data, weather, on=['year','month','day'
        final_with_weather
```

```
In [ ]: final_with_weather.dropna(inplace=True)
        final_with_weather=final_with_weather.reset_index(drop=True)
        final_with_weather
```

```
In [ ]: final_with_weather = final_with_weather[['latitude', 'longitude','year','month',
        final_with_weather['Nedbør (1 t)'].unique()
        final_with_weather = final_with_weather[final_with_weather['Nedbør (1 t)'] != '-
        final_with_weather = final_with_weather[final_with_weather['Lufttemperatur'] !=
        final_with_weather
```

```
In [ ]: #export preprocessed data to  be used for machine learning part.
        final_with_weather.to_csv('bf_traffic_weather_preprocessed.csv', index=False)
```

```
In [ ]: 
```

BikeFinder theft data pre-processing

bf_theft_preprocess.ipynb

Dawit H. Kidane, 15.june.2022

Importing Libraries

In [ ]:
```
#Importing required libraries
import random
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from datetime import timedelta
import folium
from folium import plugins
from folium.plugins import MarkerCluster
import warnings
warnings.filterwarnings("ignore")
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Loading and exploring the Bikefinder theft data

In [ ]:
```
#Load the BikeFinder theft Data
bf_Theft_data = pd.read_csv('BikeFinder_theft_Data.csv')

#Rename BikeFinder theft data columns to match BikeFinder position data
bf_Theft_data.rename(columns = {'Time':'timestamp','Device':'deviceId'}, inplace = True)

#check for duplicates
print('checking for duplicate rows :\n',bf_Theft_data.duplicated().value_counts(),'\n')

#drop duplicates
bf_Theft_data_dup_droped = bf_Theft_data.drop_duplicates()

#check for unique devices
print('checking device duplicates :\n',bf_Theft_data_dup_droped.duplicated("deviceId").value_counts(),'\n')

#Convert theft dataset Time format to 'month/day/year hour:minute:second'
bf_Theft_data_dup_droped['timestamp'] = pd.to_datetime(bf_Theft_data_dup_droped['timestamp']).dt.strftime('%m/%
bf_Theft_data_dup_droped['timestamp'] = pd.to_datetime(bf_Theft_data_dup_droped['timestamp'], format='%m/%d/%Y
print('checking new time format :\n',bf_Theft_data_dup_droped['timestamp'][0])

#Sort by Time first and then by Device id so that each device is grouped together and sorted by time
bf_Theft_data_dup_droped_sorted = bf_Theft_data_dup_droped.sort_values(by='timestamp').reset_index(drop=True)
bf_Theft_data_dup_droped_sorted = bf_Theft_data_dup_droped_sorted.sort_values(by='deviceId').reset_index(drop=T

#show the dataset
bf_Theft_data_dup_droped_sorted
```

In [ ]:
```
# In this section only one theft report per tracker in one day should be taken into account
mins=1440 #minutes in a day
counter=0

# New Dataframe for rsults
bf_theft_data_results = pd.DataFrame(columns=['deviceId','timestamp'])

# Iterate through every row
for i in range(len(bf_Theft_data_dup_droped_sorted)):

    # If Device exists from before in the result dataframe, go in and compare the date differences
    if str(bf_Theft_data_dup_droped_sorted.iloc[i]["deviceId"]) in str(bf_theft_data_results.deviceId):

        #Take the Last date of the existing Device and calculate the differences in the date with the current c
        res=(pd.Timedelta((bf_theft_data_results.loc[bf_theft_data_results["deviceId"]
                              ==bf_Theft_data_dup_droped_sorted.iloc[i]["deviceId"]]).iloc[-1]['timestam
             -bf_Theft_data_dup_droped_sorted.iloc[i]["timestamp"]).seconds/ 60.0)

        #If the differences is less than 1440 mins then store it in the result data frame
        if (res>mins):

            bf_theft_data_results = bf_theft_data_results.append({'deviceId': bf_Theft_data_dup_droped_sorted.i
                                 ,'timestamp': bf_Theft_data_dup_droped_sorted.iloc[i]["timestamp"]}, ignore

    #else add it as new value
    else:
        bf_theft_data_results = bf_theft_data_results.append({'deviceId': bf_Theft_data_dup_droped_sorted.iloc[
                             ,'timestamp': bf_Theft_data_dup_droped_sorted.iloc[i]["timestamp"]}, ignore_inc


#Print results
bf_theft_data_results
```

In [ ]:
```
#Load the BikeFinder bike position data
bf_position_data = pd.read_csv('final_deviceLocationData.csv')

print('checking bikfinder traffic dataset size :\n',bf_position_data.shape,'\n')
```

In [ ]:
```
#check for undefined latitude and/or longitude values
print('nr. of dupliicates for latitude:\n',(bf_position_data['latitude']=='undefined').sum(),'\n')
print('nr. of dupliicates for longitude:\n',(bf_position_data['longitude']=='undefined').sum(),'\n')

#remove undefined latitude and/or longitude values
bf_position_data = bf_position_data[bf_position_data.latitude != 'undefined']
bf_position_data = bf_position_data[bf_position_data.longitude != 'undefined']

# convert longitude and latitude values from string to float
bf_position_data['longitude'] = bf_position_data['longitude'].astype(float)
bf_position_data['latitude'] = bf_position_data['latitude'].astype(float)

#Limit the data to Stavanger only
bf_position_data = bf_position_data[bf_position_data['longitude'].between(5.585986955209788, 5.773063826295662)

bf_position_data=bf_position_data[bf_position_data['latitude'].between(58.9180072658198, 58.98768986749389)].re
#show the dataset
bf_position_data
```

In [ ]:
```
#Convert traffic dataset Time format to 'month/day/year hour:minute:second'

bf_position_data['timestamp'] = pd.to_datetime(bf_position_data['timestamp']).dt.strftime('%m/%d/%Y %H:%M:%S')

bf_position_data['timestamp'] = pd.to_datetime(bf_position_data['timestamp'], format='%m/%d/%Y %H:%M:%S')

print('checking new time format :\n',bf_position_data['timestamp'][0])
```

In [ ]:
```
# Merge the latitude and longitude of the devices with the latest time right before sending the notification

counter=0

#new dataframe to store results in
bf_stavanger_theft_data = pd.DataFrame(columns=['deviceId','packetType','latitude','longitude','timestamp'])

# iterate through the theft report data
for i in range(len(bf_theft_data_results)):

    # get all the rows with the specific device id
    latest = bf_position_data.loc[bf_position_data['deviceId'] ==
                           str(bf_theft_data_results.iloc[i]["deviceId"])].sort_values(by='timestamp').reset_ir

    # if it is not empty or there exist latest data that is before the theft report then continue
    if not latest.empty and not(latest.loc[latest['timestamp'] < (bf_theft_data_results.iloc[i]["timestamp"])])

        #get the latest data before the theft reprot
        temp = (latest.loc[latest['timestamp'] < (bf_theft_data_results.iloc[i]["timestamp"])]).iloc[-1]

        #store the values for the latest data before theft in the new dataframe
        bf_stavanger_theft_data = bf_stavanger_theft_data.append({'deviceId': temp["deviceId"],'packetType': te
                  ,'latitude': temp["latitude"],'longitude': temp["longitude"]
                  ,'timestamp': bf_theft_data_results.iloc[i]["timestamp"]}, ignore_index=True)

        #counts how many rows are detected
        counter = counter + 1

counter
```

In [ ]:
```
#reset indexes
bf_stavanger_theft_data = bf_stavanger_theft_data.reset_index(drop=True)
#anonymizations
bf_stavanger_theft_data['longitude']=random.uniform((bf_stavanger_theft_data['longitude'])-0.00500, (bf_stavang
bf_stavanger_theft_data['latitude']=random.uniform((bf_stavanger_theft_data['latitude'])-0.00500, (bf_stavanger
```

In [ ]:
```
#plot a scatter plot of the results
plt.scatter(x=bf_stavanger_theft_data['longitude'].astype(float), y=bf_stavanger_theft_data['latitude'].astype
plt.xlabel('longitude')
plt.xlabel('latitude')

plt.show()
```

In [ ]:
```
#https://python-visualization.github.io/folium/modules.html#module-folium.map

#show the results in map, theft reports in Stavanger

#remove nan values.
bf_stavanger_theft_data = bf_stavanger_theft_data[bf_stavanger_theft_data['latitude'].notna()]
bf_stavanger_theft_data = bf_stavanger_theft_data[bf_stavanger_theft_data['longitude'].notna()]

#plotting the points in a map using folium library
fig_1 = folium.Map([59,5.6], zoom_start=11)
for index, row in bf_stavanger_theft_data.iterrows():
    folium.CircleMarker([row['latitude'], row['longitude']],
                        radius=3,
                        popup=row['deviceId'],
                       ).add_to(fig_1)
fig_1
```

In [ ]:
```
#plot the center of the points by calculating the average of the latitude and longitude values.

lat = []
long = []
for index, row in bf_stavanger_theft_data.iterrows():
    lat.append(row["latitude"])
    long.append(row["longitude"])

lat1=sum(lat)/len(lat)
lat2=sum(long)/len(long)
folium.CircleMarker([lat1,lat2],
                        radius=5,
                        popup="CENTER LOCATION",
                        color='black',
                       ).add_to(fig_1)
fig_1
```

In [ ]:
```
#split date and time values.

final_bf_stavanger_theft_data = bf_stavanger_theft_data[['latitude','longitude','timestamp']]
final_bf_stavanger_theft_data.rename(columns = {'timestamp':'theft_time'}, inplace = True)

# convert date_time column to datetime type
final_bf_stavanger_theft_data.theft_time = pd.to_datetime(final_bf_stavanger_theft_data.theft_time)

#split theft into separate columns

# extract hours
hours = final_bf_stavanger_theft_data.theft_time.dt.hour
# extract minutes
mins = final_bf_stavanger_theft_data.theft_time.dt.minute
# extract seconds
sec = final_bf_stavanger_theft_data.theft_time.dt.second

# extract month
year = final_bf_stavanger_theft_data.theft_time.dt.year

# extract month
months = final_bf_stavanger_theft_data.theft_time.dt.month

# extract day of a month
day_of_month = final_bf_stavanger_theft_data.theft_time.dt.day

time_data = pd.DataFrame({
    'year' : year,
    'month' : months,
    'day_of_month' : day_of_month,
    'hour' : hours,
    'minutes' : mins,
    'seconds' : sec

})
final_bf_stavanger_theft_data = pd.concat([final_bf_stavanger_theft_data, time_data], axis = 1)

final_bf_stavanger_theft_data = final_bf_stavanger_theft_data[['latitude', 'longitude','year','month','day_of_r
final_bf_stavanger_theft_data
```

In [ ]:
```
#export preprocessed data to  be used for machine learning part.
final_bf_stavanger_theft_data.to_csv('bf_theft_preprocessed.csv', index=False)
```

Rogaland Police District theft data pre-processing

police_preprocess.ipynb

Dawit H. Kidane, 15.june.2022

Importing Libraries

In [ ]:
```
#Importing required libraries

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from datetime import timedelta
import folium
from folium import plugins
from folium.plugins import MarkerCluster
from geopy.geocoders import Nominatim
import warnings
warnings.filterwarnings("ignore")
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Loading and exploring the police theft data

In [ ]:
```
#Load the Police theft Data
Theft_data = pd.read_excel('Tyveri_av_sykkel_2019_til_2021.xlsx')
Theft_data

#sort the data by date and time
Theft_data = Theft_data.sort_values(by="Gj dato start").reset_index(drop=True)

#check for duplicates
print('checking for duplicate rows :\n',Theft_data.duplicated().value_counts(),'\n')

#drop duplicates
Theft_data=Theft_data.drop_duplicates().reset_index(drop=True)

#take only data after 31-12-2017
Theft_data=(Theft_data.loc[Theft_data['Gj dato start'] > ('2017-12-31')]).reset_index(drop=True)

#converting time and date type
pd.to_datetime(Theft_data['Gj kl start'],format= '%H:%M' ).dt.time
pd.to_datetime(Theft_data['Gj dato start']).dt.strftime('%m/%d/%Y')


#combining date and time
Theft_data['timestamp'] = pd.to_datetime(pd.to_datetime(Theft_data['Gj dato start']).dt.strftime('%m/%d/%Y').as

#chaninging time and date format
Theft_data['timestamp'] = pd.to_datetime(Theft_data['timestamp']).dt.strftime('%m/%d/%Y %H:%M')

Theft_data['timestamp'] = pd.to_datetime(Theft_data['timestamp'], format='%m/%d/%Y %H:%M')
Theft_data['timestamp'][0]
```

In [ ]:
```
#check for unique police zone values
Theft_data.Politisone.unique()
```

In [ ]:
```
# assigning latitude and longitude values to the areas and add the rest manually

results = pd.DataFrame(columns=['Politisone','longitude','latitude'])

for i in ['Eiganes', 'Kampen', 'Varden', 'Parken', 'Hinna',
       'Sølvberget', 'Ullandhaug', 'Ytre Tasta', 'Kvalaberg', 'Hundvåg',
       'Straen', 'Tjensvoll', 'Randaberg sentrum',
       'Boganes', 'Stokka', 'Sunde', 'Våland', 'Vaulen',
       'Madlamark','Gausel', 'Forus Øst', 'Skagen', 'Jåtten', 'Øyane - Hundvåg',
       'Mosterøy', 'Kvernevik', 'Buøy', 'Rennesøy',
       'Finnøy', 'V. Åmøy']:

    loc = Nominatim(user_agent="GetLoc")
    getLoc = loc.geocode(i+ ", Rogaland" )

    results = results.append({'longitude': getLoc.longitude
                              ,'latitude': getLoc.latitude,
                             'Politisone':i}, ignore_index=True)

results = results.append({'longitude': 5.694794
                        ,'latitude': 58.986799,
                       'Politisone':'Indre Tasta'}, ignore_index=True)

results = results.append({'longitude': 5.739472
                        ,'latitude': 58.970624,
                       'Politisone':'Storhaug Nord'}, ignore_index=True)

results = results.append({'longitude': 5.651457
                        ,'latitude': 59.003017,
                       'Politisone':'Randaberg Øst'}, ignore_index=True)

results = results.append({'longitude': 5.618087
                        ,'latitude': 58.991531,
                       'Politisone':'Randaberg Sør'}, ignore_index=True)

results = results.append({'longitude': 5.708706
                        ,'latitude': 58.889985,
                       'Politisone':'Forus Vest'}, ignore_index=True)

results = results.append({'longitude': 5.647453
                        ,'latitude': 58.939494,
                       'Politisone':'Madlasannes'}, ignore_index=True)

results = results.append({'longitude': 5.614829
                        ,'latitude': 58.998939,
                       'Politisone':'Randaberg Vest'}, ignore_index=True)

results = results.append({'longitude': 5.668205
                        ,'latitude': 59.045169,
                       'Politisone':'Bru og Sokn'}, ignore_index=True)

results = results.append({'longitude': 5.792006
                        ,'latitude': 58.997676,
                       'Politisone':'Byøyene'}, ignore_index=True)


results
```

In [ ]:
```
#merge the latitude and longitude on the theft data by police zone names
Theft_data=pd.merge(Theft_data, results, on="Politisone", how="right")
```

In [ ]:
```
#check for column information
Theft_data.info()
```

In [ ]:
```
#drop null values
Theft_data.dropna(inplace=True)
Theft_data.info()
Theft_data=Theft_data.reset_index(drop=True)
```

In [ ]:
```
#show the results in map
fig_1 = folium.Map([59,5.6], zoom_start=11)
for index, row in Theft_data.iterrows():
    folium.CircleMarker([row['latitude'], row['longitude']],
                       radius=3,
                       popup=row['Politisone'],
                      ).add_to(fig_1)
fig_1
#heatmap
dfmatrix = Theft_data[['latitude', 'longitude']].values
# plot heatmap
fig_1.add_child(plugins.HeatMap(dfmatrix, radius=15))
fig_1
```

In [ ]:
```
# convert date_time column to datetime type
final_theft = Theft_data
final_theft.rename(columns = {'timestamp':'theft_time'}, inplace = True)

final_theft.theft_time = pd.to_datetime(final_theft.theft_time)

hours = final_theft.theft_time.dt.hour

year = final_theft.theft_time.dt.year

# extract month
months = final_theft.theft_time.dt.month

# extract day of a month
day_of_month = final_theft.theft_time.dt.day
```

In [ ]:
```
features = pd.DataFrame({
    'year' : year,
    'month' : months,
    'day_of_month' : day_of_month,
    'hour' : hours

})
features = pd.concat([final_theft, features], axis = 1)

final = features[['latitude', 'longitude','year','month','day_of_month','hour']]
final
```

In [ ]:
```
#export preprocessed data to  be used for machine learning part.
final.to_csv('police_theft_preprocessed.csv', index=False)
```

Stavanger bike counter, traffic data pre-processing

city_counter_preprocess.ipynb

Dawit H. Kidane, 15.june.2022

Importing Libraries

```
In [ ]: import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from datetime import timedelta
import folium
from folium import plugins
from folium.plugins import MarkerCluster
from geopy.geocoders import Nominatim
import warnings
warnings.filterwarnings("ignore")
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Loading and exploring the city bike counter data

City bike data import from : https://open.stavanger.kommune.no/dataset/bysykler-stavanger/resource/987ad1f2-99a6-4695-9924-3a943c4f5e0a

Source description from website:

Sykkeldata

Data fra sykkelsensorer i Stavanger kommune. Oppdateres daglig. Se datasettet "Lokalisering sykkeltellere Stavanger" for å finne plasseringen av tellerne. Knyttes sammen vha feltet "Station_id" (Navnefeltet kan også brukes). Bike data - Data from bike counting sensors in Stavanger municipality. Updated daily. See the dataset "Lokalisering sykkeltellere Stavanger" to find the locations of the sensors. Use with the field "Station_id" (the name field can also be used)

https://open.stavanger.kommune.no/dataset/sykkeldata

Bike counting stations 2017

URL: https://opencom.no/dataset/90cef5d5-601e-4412-87e4-3e9e8dc59245/resource/7472d940-285f-457c-baf2-b92565a6947d/download/sykkeldata2017-1.csv

```
In [ ]: url="https://opencom.no/dataset/90cef5d5-601e-4412-87e4-3e9e8dc59245/resource/7472d940-285f-457c-baf2-b92565a69
bike_counting_2017 = pd.read_csv(url)

print(bike_counting_2017['Station_Name'].unique())
bike_counting_2017.info()
```

```
In [ ]: bike_counting_2017['Date'] = pd.to_datetime(bike_counting_2017['Date']).dt.strftime('%Y-%m-%d')

bike_counting_2017['Date'] = pd.to_datetime(bike_counting_2017['Date'], format='%Y-%m-%d')

bike_counting_2017
```

Bike counting stations 2018

URL: https://opencom.no/dataset/90cef5d5-601e-4412-87e4-3e9e8dc59245/resource/18b5a612-e9f9-4d53-9134-3ea5f162f956/download/sykkeldata2018.csv

```
In [ ]: url="https://opencom.no/dataset/90cef5d5-601e-4412-87e4-3e9e8dc59245/resource/18b5a612-e9f9-4d53-9134-3ea5f162f
bike_counting_2018 = pd.read_csv(url)

print(bike_counting_2018['Station_Name'].unique())
bike_counting_2018.info()
```

Bike counting stations 2019

URL: https://opencom.no/dataset/90cef5d5-601e-4412-87e4-3e9e8dc59245/resource/36477654-14cf-405c-8f23-ba6fbe674d94/download/sykkeldata_2019.csv

```
In [ ]: url="https://opencom.no/dataset/90cef5d5-601e-4412-87e4-3e9e8dc59245/resource/36477654-14cf-405c-8f23-ba6fbe674
bike_counting_2019 = pd.read_csv(url)

print(bike_counting_2019['Station_Name'].unique())
bike_counting_2019.info()
```

Bike counting stations 2020

URL: https://opencom.no/dataset/90cef5d5-601e-4412-87e4-3e9e8dc59245/resource/4952514a-0590-4381-9583-0048a10f3f87/download/sykkeldata_2020.csv

```
In [ ]: url="https://opencom.no/dataset/90cef5d5-601e-4412-87e4-3e9e8dc59245/resource/4952514a-0590-4381-9583-0048a10f3
bike_counting_2020 = pd.read_csv(url)

print(bike_counting_2020['Station_Name'].unique())
bike_counting_2020.info()
```

Bike counting stations 2021

URL: https://opencom.no/dataset/90cef5d5-601e-4412-87e4-3e9e8dc59245/resource/d86e8405-fc7a-47e7-a67c-ec156a3a1e87/download/sykkeldata.csv

```
In [ ]: url="https://opencom.no/dataset/90cef5d5-601e-4412-87e4-3e9e8dc59245/resource/d86e8405-fc7a-47e7-a67c-ec156a3a1
bike_counting_2021 = pd.read_csv(url)

print(bike_counting_2021['Station_Name'].unique())
bike_counting_2021.info()
```

Bike counting stations 2022

URL: https://opencom.no/dataset/90cef5d5-601e-4412-87e4-3e9e8dc59245/resource/8f3d84b5-c3b8-41b2-8ffd-4dee1b6ffc86/download/sykkeldata.csv

```
In [ ]: url="https://opencom.no/dataset/90cef5d5-601e-4412-87e4-3e9e8dc59245/resource/8f3d84b5-c3b8-41b2-8ffd-4dee1b6ff
bike_counting_2022 = pd.read_csv(url)

print(bike_counting_2022['Station_Name'].unique())
bike_counting_2022.info()
```

```
In [ ]: frames = [bike_counting_2017,
          bike_counting_2018,
          bike_counting_2019,
          bike_counting_2020,
          bike_counting_2021,
          bike_counting_2022]

Merged_counting_stations_17_22 = pd.concat(frames)
```

```
In [ ]: print(Merged_counting_stations_17_22['Station_Name'].unique())
Merged_counting_stations_17_22.info()
```

```
In [ ]: totalcases = np.array([Merged_counting_stations_17_22.loc[Merged_counting_stations_17_22['Station_Name'] == 'Fo
                       Merged_counting_stations_17_22.loc[Merged_counting_stations_17_22['Station_Name'] == 'Ga
                       Merged_counting_stations_17_22.loc[Merged_counting_stations_17_22['Station_Name'] == 'Sa
                       Merged_counting_stations_17_22.loc[Merged_counting_stations_17_22['Station_Name'] == 'Hi
                       Merged_counting_stations_17_22.loc[Merged_counting_stations_17_22['Station_Name'] == 'Hå
                       Merged_counting_stations_17_22.loc[Merged_counting_stations_17_22['Station_Name'] == 'Mo
                       Merged_counting_stations_17_22.loc[Merged_counting_stations_17_22['Station_Name'] == 'Mø
                       Merged_counting_stations_17_22.loc[Merged_counting_stations_17_22['Station_Name'] == 'Ra
                       Merged_counting_stations_17_22.loc[Merged_counting_stations_17_22['Station_Name'] == 'Br
                       Merged_counting_stations_17_22.loc[Merged_counting_stations_17_22['Station_Name'] == 'Ku
                       Merged_counting_stations_17_22.loc[Merged_counting_stations_17_22['Station_Name'] == 'Ta
                       +Merged_counting_stations_17_22.loc[Merged_counting_stations_17_22['Station_Name'] == 'T
                       Merged_counting_stations_17_22.loc[Merged_counting_stations_17_22['Station_Name'] == 'St
                       Merged_counting_stations_17_22.loc[Merged_counting_stations_17_22['Station_Name'] == 'St
                       Merged_counting_stations_17_22.loc[Merged_counting_stations_17_22['Station_Name'] == 'Sø
                       Merged_counting_stations_17_22.loc[Merged_counting_stations_17_22['Station_Name'] == 'Tj
```

```
In [ ]: fig = plt.figure(figsize = (15, 5))

# creating the bar plot
plt.bar(['Forus Vest','Gausel Stasjon', 'Sandal', 'Hillevåg', 'Håhammer', 'Mosvannet', 'Møllebukta',
       'Randabergveien','Brevig','Kulvert Mariero','Tananger Bru', 'Stemmen', 'Stokkavannet', 'Sørmarka',
       'Tjensvollkrysset'], totalcases, color ='maroon',
        width = 0.4)

plt.xlabel("Sensor Stations")
plt.xticks(rotation = 90)
plt.ylabel("Passings (y * 10^6)")
plt.title("Stavanger bike counter 2017-2022")
plt.show()
```

```
In [ ]: Merged_counting_stations_17_22.loc[Merged_counting_stations_17_22['Station_Name'] == 'Forus Vest', 'Count'].sum
```

```
In [ ]: Merged_counting_stations_17_22
Merged_counting_stations_17_22['Date'] = pd.to_datetime(Merged_counting_stations_17_22['Date']).dt.strftime('%Y

Merged_counting_stations_17_22['Date'] = pd.to_datetime(Merged_counting_stations_17_22['Date'], format='%Y-%m-%

Merged_counting_stations_17_22
```

```
In [ ]: Merged_counting_stations_17_22.columns
```

Geolocator

```
In [ ]: results = pd.DataFrame(columns=['name','longitude','latitude'])

for i in ['Forus Vest','Gausel Stasjon', 'Sandal', 'Hillevåg', 'Mosvannet', 'Møllebukta',
       'Randabergveien','Tananger Bru', 'Stemmen', 'Stokkavannet', 'Sørmarka',
       'Tjensvollkrysset']:

    loc = Nominatim(user_agent="GetLoc")
    getLoc = loc.geocode(i+ ", Rogaland" )

    results = results.append({'longitude': getLoc.longitude
                             ,'latitude': getLoc.latitude,
                            'name':i}, ignore_index=True)

results = results.append({'longitude': 5.766132388103561
                        ,'latitude': 58.96522697040621,
                       'name':'Brevig'}, ignore_index=True)

results = results.append({'longitude': 5.741497499503618
                        ,'latitude': 58.93473267193572,
                       'name':'Kulvert Mariero'}, ignore_index=True)

results = results.append({'longitude': 5.672781147867864
                        ,'latitude': 58.94165180079404,
                       'name':'Håhammer'}, ignore_index=True)

results.rename(columns = {'name':'Station_Name'}, inplace = True)
results
```

```
In [ ]: #show the results in map
fig_1 = folium.Map([59,5.6], zoom_start=11)
for index, row in results.iterrows():
    folium.CircleMarker([row['latitude'], row['longitude']],
                        radius=3,
                        popup=row['Station_Name'],
                       ).add_to(fig_1)
fig_1
```

```
In [ ]: Merged_counting_stations_17_22=pd.merge(Merged_counting_stations_17_22, results, on="Station_Name", how="left")
```

```
In [ ]: Merged_counting_stations_17_22['temp_time']= Merged_counting_stations_17_22['Time'].str.split(':')
Merged_counting_stations_17_22['temp_date']= Merged_counting_stations_17_22['Date'].astype(str).str.split('-')
```

```
In [ ]: Merged_counting_stations_17_22["hours"] = Merged_counting_stations_17_22["temp_time"].str[0]
Merged_counting_stations_17_22["day"] = Merged_counting_stations_17_22["temp_date"].str[2]
Merged_counting_stations_17_22["month"] = Merged_counting_stations_17_22["temp_date"].str[1]
Merged_counting_stations_17_22["year"] = Merged_counting_stations_17_22["temp_date"].str[0]
```

```
In [ ]: Merged_counting_stations_17_22['Tid(norsk normaltid)'] = Merged_counting_stations_17_22["day"]+'.'+Merged_count
```

```
In [ ]: Merged_counting_stations_17_22
```

```
In [ ]: del Merged_counting_stations_17_22["Station_id"]
del Merged_counting_stations_17_22["Station_Uptime"]
del Merged_counting_stations_17_22["Lane_Name"]
del Merged_counting_stations_17_22["Average_Speed"]
del Merged_counting_stations_17_22["Average_Temperature"]
Merged_counting_stations_17_22.dropna(inplace=True)
Merged_counting_stations_17_22.info()
```

Merge Traffic data with weather data

Weather data from https://seklima.met.no/observations/

```
In [ ]: weather = pd.read_excel('table.xlsx')
weather.head(5)
```

```
In [ ]: with_weather=pd.merge(Merged_counting_stations_17_22, weather, on="Tid(norsk normaltid)", how="left")
```

```
In [ ]: #check for undefined latitude and/or longitude values
print((Merged_counting_stations_17_22['latitude']=='').sum())
print((Merged_counting_stations_17_22['longitude']=='').sum())

#remove undefined latitude and/or longitude values
Merged_counting_stations_17_22 = Merged_counting_stations_17_22[Merged_counting_stations_17_22.latitude != 'und
Merged_counting_stations_17_22 = Merged_counting_stations_17_22[Merged_counting_stations_17_22.longitude != 'un

print((Merged_counting_stations_17_22['latitude']=='undefined').sum())
print((Merged_counting_stations_17_22['longitude']=='undefined').sum())

# convert longitude and latitude values from string to float
Merged_counting_stations_17_22['longitude'] = Merged_counting_stations_17_22['longitude'].astype(float)
Merged_counting_stations_17_22['latitude'] = Merged_counting_stations_17_22['latitude'].astype(float)
```

```
In [ ]: Merged_counting_stations_17_22 = Merged_counting_stations_17_22.reset_index(drop=True)
```

```
In [ ]: final_stavanger_position_data = Merged_counting_stations_17_22[['latitude', 'longitude','year','month','day','h
final_stavanger_position_data
```

```
In [ ]: #export preprocessed data to  be used for machine learning part.
final_stavanger_position_data.to_csv('city_traffic_preprocessed.csv', index=False)
```

```
In [ ]: with_weather.dropna(inplace=True)
```

```
In [ ]: #check for undefined latitude and/or longitude values
print((with_weather['latitude']=='').sum())
print((with_weather['longitude']=='').sum())

#remove undefined latitude and/or longitude values
with_weather = with_weather[with_weather.latitude != 'undefined']
with_weather = with_weather[with_weather.longitude != 'undefined']

print((with_weather['latitude']=='undefined').sum())
print((with_weather['longitude']=='undefined').sum())

# convert longitude and latitude values from string to float
with_weather['longitude'] = with_weather['longitude'].astype(float)
with_weather['latitude'] = with_weather['latitude'].astype(float)
```

```
In [ ]: with_weather=with_weather.reset_index(drop=True)
final_traffic=with_weather
```

```
In [ ]: final_traffic.rename(columns = {'Tid(norsk normaltid)':'theft_time'}, inplace = True)
final_traffic
```

```
In [ ]: final = final_traffic[['latitude', 'longitude','year','month','day','hours','Nedbør (1 t)','Lufttemperatur']]
#making sure all empty data are removed
final['Nedbør (1 t)'].unique()
final = final[final['Nedbør (1 t)'] != '-']
final = final[final['Lufttemperatur'] != '-']
final
```

```
In [ ]: #export preprocessed data to  be used for machine learning part.
final.to_csv('city_traffic_weather_preprocessed.csv', index=False)
```

```
In [ ]:
```

Chicago crime data, preprocessing & predictions

theft_predictions.ipynb

Dawit H. Kidane, 15.june.2022

Importing Libraries

```
#Importing required libraries
from math import sqrt
from sklearn.metrics import mean_absolute_error, mean_squared_error  #for calculation of errors
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import sklearn.neighbors
from sklearn.neighbors import KNeighborsRegressor
from numpy import sqrt
import random
from sklearn.cluster import KMeans #for performing Kmeans
from sklearn.metrics import silhouette_samples, silhouette_score #for silhouette
from sklearn import tree
from sklearn.ensemble import RandomForestRegressor
from scipy.cluster.hierarchy import dendrogram, linkage #for the dendogram
from sklearn.cluster import AgglomerativeClustering #for performing AgglomerativeClustering
from sklearn.cluster import KMeans #for performing Kmeans
from scipy.cluster.hierarchy import dendrogram, linkage

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from datetime import timedelta
import folium
from folium import plugins
from folium.plugins import MarkerCluster
import warnings
warnings.filterwarnings("ignore")
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Loading and exploring the Chicago crime data

https://www.kaggle.com/datasets/chicago/chicago-crime?select=crime

```
#Load the preprocessed BikeFinder theft Data
data = pd.read_csv('Crimes_-_2001_to_Present.csv')
data.shape

#check for duplicates
data.duplicated().value_counts()

#use 100000 rows
data_100000=data.head(100000)

print(data_100000.columns)
print(data_100000.info())

# convert date_time column to datetime type
data_100000.Date = pd.to_datetime(data_100000.Date)
```

```
print(data_100000.columns)
data_100000.info()
```

```
#split theft into separate columns

# extract hours
hours = data_100000.Date.dt.hour
# extract minutes
mins = data_100000.Date.dt.minute
# extract seconds
sec = data_100000.Date.dt.second

# extract month
year = data_100000.Date.dt.year

# extract month
months = data_100000.Date.dt.month

# extract day of a month
day_of_month = data_100000.Date.dt.day

time_data = pd.DataFrame({
    'year' : year,
    'month' : months,
    'day_of_month' : day_of_month,
    'hour' : hours,
    'minutes' : mins,
    'seconds' : sec

})
final_chicago_theft_data = pd.concat([data_100000, time_data], axis = 1)

final_chicago_theft_data = final_chicago_theft_data[['Latitude', 'Longitude','year','month','day_of_month','hou
```

```
#drop empty rows
final_chicago_theft_data=final_chicago_theft_data.dropna().reset_index(drop=True)
final_chicago_theft_data
```

```
#https://python-visualization.github.io/folium/modules.html#module-folium.map
#show in map first 100 points

data_100 =final_chicago_theft_data.head(100)
data_100 = data_100[data_100['Latitude'].notna()]
data_100 = data_100[data_100['Longitude'].notna()]

fig_1 = folium.Map([41.8616504,-87.6779599], zoom_start=11)
for index, row in data_100.iterrows():
    folium.CircleMarker([row['Latitude'], row['Longitude']],
                        radius=3,
                        popup=row['year'],
                       ).add_to(fig_1)
fig_1
```

```
#heatmap
dfmatrix = data_100[['Latitude', 'Longitude']].values
# plot heatmap
fig_1.add_child(plugins.HeatMap(dfmatrix, radius=15))
fig_1
```

Predictions

```
#Latitude prediction

train , test = train_test_split(final_chicago_theft_data, test_size = 0.2)

x_train_latitude = train.drop(['Latitude','Longitude'], axis=1)
y_train_latitude = train['Latitude']

x_test_latitude = test.drop(['Latitude','Longitude'], axis = 1)
y_test_latitude = test['Latitude']
```

```
#scalling the training values between 0 and 1, to avoid bias results

scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled_latitude = scaler.fit_transform(x_train_latitude)
x_train_latitude = pd.DataFrame(x_train_scaled_latitude)

x_test_scaled_latitude = scaler.fit_transform(x_test_latitude)
x_test_latitude = pd.DataFrame(x_test_scaled_latitude)
```

Testing predictions with different alogoritms

```
#tree latitude
clf = tree.DecisionTreeRegressor()
clf.fit(x_train_latitude, y_train_latitude)
pred_tree = clf.predict(x_test_latitude)

rmse_latitude_tree = sqrt(mean_squared_error(y_test_latitude,pred_tree))
print("rmse_latitude_tree",rmse_latitude_tree)

#RF latitude
RF = RandomForestRegressor()
RF.fit(x_train_latitude, y_train_latitude)
pred_RF = RF.predict(x_test_latitude)

rmse_latitude_RF = sqrt(mean_squared_error(y_test_latitude,pred_RF))
print("rmse_latitude_RF",rmse_latitude_RF)
```

```
#Performing KNN and picking the model with the best results

best_k_latitude = 0
rmse_latitude_KN = 0

rmse_values_latitude = []
for K in range(10):
    K = K+1
    KNN = sklearn.neighbors.KNeighborsRegressor(n_neighbors = K)

    KNN.fit(x_train_latitude, y_train_latitude)
    pred = KNN.predict(x_test_latitude)
    rmse = sqrt(mean_squared_error(y_test_latitude,pred))
    rmse_values_latitude.append(rmse)


    if best_k_latitude == 0 or rmse_latitude_KN > rmse:
        best_k_latitude = K
        rmse_latitude_KN = rmse
        best_predictions_latitude= pred

print('k = ' , best_k_latitude , ', gives the smallest rmse value:', rmse_latitude_KN)
```

```
#compare Latitiude rmse
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_axes([0,0,2,1])
Methods = ['KNN Regression', 'Decision Tree', 'Random Forest']
plt.title("Latitude RMSE Comparison")
plt.xlabel("Methods")
plt.ylabel("RMSE")
RMSE_latitude = [rmse_latitude_KN,rmse_latitude_tree,rmse_latitude_RF]
ax.bar(Methods,RMSE_latitude,color=['blue','green','red'])
plt.show()
```

```
#plotting the rmse values against k values
rmse_plots = pd.DataFrame(rmse_values_latitude)
plt.title('Rmse values')
plt.xlabel('k values')
plt.ylabel('Rmse values')
plt.plot(rmse_plots)
```

```
#longitude

x_train_longitude = train.drop(['Latitude','Longitude'], axis=1)
y_train_longitude = train['Longitude']

x_test_longitude = test.drop(['Latitude','Longitude'], axis = 1)
y_test_longitude = test['Longitude']
```

```
#scalling the training values between 0 and 1, to avoid bias results

scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled_longitude = scaler.fit_transform(x_train_longitude)
x_train_longitude = pd.DataFrame(x_train_scaled_longitude)

x_test_scaled_longitude = scaler.fit_transform(x_test_longitude)
x_test_longitude = pd.DataFrame(x_test_scaled_longitude)
```

```
#tree longitude

clf = tree.DecisionTreeRegressor()
clf = clf.fit(x_train_longitude, y_train_longitude)
pred_tree = clf.predict(x_test_longitude)

rmse_longitude_tree = sqrt(mean_squared_error(y_test_longitude,pred_tree))
print("rmse_longitude_tree",rmse_longitude_tree)

#RF longitude
RF = RandomForestRegressor()
RF.fit(x_train_longitude, y_train_longitude)
pred_RF = RF.predict(x_test_longitude)

rmse_longitude_RF = sqrt(mean_squared_error(y_test_longitude,pred_RF))
print("rmse_longitude_RF",rmse_longitude_RF)
```

```
#Performing KNN and picking the model with the best results

best_k_longitude = 0
rmse_longitude_KN = 0

rmse_values_longitude = []
for K in range(10):
    K = K+1
    KNN = sklearn.neighbors.KNeighborsRegressor(n_neighbors = K)

    KNN.fit(x_train_longitude, y_train_longitude)
    pred_longitude = KNN.predict(x_test_longitude)
    rmse = sqrt(mean_squared_error(y_test_longitude,pred_longitude))
    rmse_values_longitude.append(rmse)


    if best_k_longitude == 0 or rmse_longitude_KN > rmse:
        best_k_longitude = K
        rmse_longitude_KN = rmse
        best_predictions_longitude= pred_longitude

print('k = ' , best_k_longitude , ', gives the smallest rmse value:', rmse_longitude_KN)
```

```
#compare Longitude rmse
fig = plt.figure()
ax = fig.add_axes([0,0,2,1])
Methods = ['KNN Regression', 'Decision Tree','Random Forest']
plt.title("longitude RMSE COMPARISON")
plt.xlabel("Methods")
plt.ylabel("RMSE")
RMSE_longitude = [rmse_longitude_KN,rmse_longitude_tree,rmse_longitude_RF]
ax.bar(Methods,RMSE_longitude,color=['blue','green','red'])
plt.show()
```

```
#plotting the rmse values against k values
rmse_plots = pd.DataFrame(rmse_values_longitude)
plt.title('Rmse values')
plt.xlabel('k values')
plt.ylabel('Rmse values')
plt.plot(rmse_plots)
```

```
results = pd.DataFrame()

results['Test_Data_latitude']=y_test_latitude
results['Test_Data_longitude']=y_test_longitude
#print(y_test.shape)

results['Predictions_longitude']=best_predictions_longitude
results['Predictions_latitude']=best_predictions_latitude

#Sorting them by based on the keys from the test data
results = results.sort_index()
results
```

```
plt.plot(results['Test_Data_latitude'],)
plt.plot(results['Predictions_latitude'])
plt.title('test vs latitude prediction')
plt.xlabel('Indexes')
plt.ylabel('latitude')
plt.legend(['Test Data','K-Nearest Neighbour Predictions'])
```

```
plt.plot(results['Test_Data_longitude'],)
plt.plot(results['Predictions_longitude'])
plt.title('test vs longitude prediction')
plt.xlabel('Indexes')
plt.ylabel('longitude')
plt.legend(['Test Data','K-Nearest Neighbour Predictions'])
```

Clustering

Hierarchy Tree Clustering

```
#Sources:
#https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.axes.Axes.axhline.html
#https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.figure.html
#https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.normalize.html

df_hierarchy=final_chicago_theft_data.head(100)

# assigning the latitude and longitude column to HT
HT = df_hierarchy.iloc[:, 0:2].values
#print(HT)

#creating Dendograms for both latitude and longitude values combined
plt.figure(figsize=(10, 7))
plt.title("Theft data dendogram")
z = linkage(HT)
dendogram = dendrogram(z)
```

```
#Choosing The Optimal Number Of Clusters
#Sources:
#https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html

range_n_clusters = [ 3,4,5, 6,7,8,9,10,11,12,13,14,15]

print("********Checking for the optimal number of clusters for latitude and longitude combined**********")

best_n=0
largest_silhouette_av = 0
HT_cluster_result = 0
for n_clusters in range_n_clusters:

    # clustering for latitude longitude values combines
    HT_cluster = AgglomerativeClustering(n_clusters=n_clusters, affinity='euclidean', linkage='ward')
    HT_cluster_res=HT_cluster.fit_predict(HT)

    HT_silhouette_avg = silhouette_score(HT, HT_cluster_res)
    #print("For n_clusters =", n_clusters,"The average silhouette_score is :", HT_silhouette_avg)

    if best_n == 0 or largest_silhouette_av < HT_silhouette_avg:
        best_n = n_clusters
        largest_silhouette_av = HT_silhouette_avg
        HT_cluster_result = HT_cluster_res


print('n = ' , best_n , ', gives the largest silhouette_avg value:', largest_silhouette_av)

# Adding the clustering values to the dataset
df_hierarchy['HT_cluster']=HT_cluster_result
```

```
# Scatter plott for latitude and longitude values
plt.figure(figsize=(10, 7))
plt.title("Longitude and Latitude Hierarchy Tree Clustering")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.scatter(df_hierarchy['Longitude'], df_hierarchy['Latitude'], c=df_hierarchy['HT_cluster'], cmap='rainbow')
```

KMeans Clustering

```
#Preparing the data

df_kmeans=final_chicago_theft_data.head(100)

# assigning the latitude and longitude column to KC
KC = df_kmeans.iloc[:, 0:2].values
```

```
range_n_clusters = [3,4,5,6,7,8,9,10,11,12,13,14,15]

print("*****Checking for the optimal number of clusters for latitude and longitude, and getting the results.***


best_k=0
largest_silhouette_av = 0
KC_cluster_result = 0

for n_clusters in range_n_clusters:


    kmeans_KC = KMeans(n_clusters=n_clusters)
    KC_clusters=kmeans_KC.fit(KC)


    KC_silhouette_avg = silhouette_score(KC, KC_clusters.labels_)

    #print("For n_clusters =", n_clusters,"The average silhouette_score is :", KC_silhouette_avg)

    if best_k == 0 or largest_silhouette_av < KC_silhouette_avg:
        best_k = n_clusters
        largest_silhouette_av = KC_silhouette_avg
        KC_cluster_result = KC_clusters.labels_


print('n = ' , best_k , ', gives the largest silhouette_avg value:', largest_silhouette_av,"\n")

#*****************************************************************************************************************
```

```
#Sources:
#https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

#Adding the clustering values to the dataframe as new columns
df_kmeans['KC_clusters']=KC_cluster_result
```

```
#Sources:
# https://matplotlib.org/3.1.0/gallery/subplots_axes_and_figures/subplots_demo.html

# Scatter plott for Latitude and Longitude values
plt.figure(figsize=(10, 7))
plt.title("Longitude and Latitude K-means clustered")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.scatter(df_kmeans['Longitude'], df_kmeans['Latitude'], c=df_kmeans['KC_clusters'], cmap='rainbow')
```

```
#Sources:
#https://medium.com/@ODSC/assessment-metrics-for-clustering-algorithms-4a902e00d92d

#EVALUATION for:



# Silhouetter Score evaluation for Kmeans clustering
score1 = silhouette_score(KC, KC_cluster_result, metric='euclidean')
print('Silhouette Score for Kmeans clustering by positions: %.3f' % score1)

#*******************************************************************************************************

# Silhouetter Score evaluation for Hierarchy Tree clustering
score2 = silhouette_score(HT, HT_cluster_result, metric='euclidean')
print('Silhouette Score for Hierarchy Tree clustering by positions: %.3f' % score2)
```

Theft predictions

theft_predictions.ipynb

Dawit H. Kidane, 15.june.2022

Importing Libraries

In [ ]:
```python
#Importing required libraries
from math import sqrt
from sklearn.metrics import mean_absolute_error, mean_squared_error  #for calculation of errors
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import sklearn.neighbors
from sklearn.neighbors import KNeighborsRegressor
from numpy import sqrt
import random
from sklearn.cluster import KMeans #for performing Kmeans
from sklearn.metrics import silhouette_samples, silhouette_score #for silhouette


import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from datetime import timedelta
import folium
from folium import plugins
from folium.plugins import MarkerCluster
import warnings
warnings.filterwarnings("ignore")
warnings.simplefilter(action='ignore', category=FutureWarning)
```

BikeFinder theft data

In [ ]:
```python
#Load the preprocessed BikeFinder theft Data
final_bf_stavanger_theft_data = pd.read_csv('bf_theft_preprocessed.csv')
final_bf_stavanger_theft_data
```

In [ ]:
```python
#Latitude prediction for BikeFinder data

#splitting the data into training and testing parts
train , test = train_test_split(final_bf_stavanger_theft_data, test_size = 0.2)

#removing the latitude and longitude values as predictors and adding latitude as response value
x_train_latitude = train.drop(['latitude','longitude'], axis=1)
y_train_latitude = train['latitude']

x_test_latitude = test.drop(['latitude','longitude'], axis = 1)
y_test_latitude = test['latitude']
```

In [ ]:
```python
#scalling the training values between 0 and 1, to avoid bias results

scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled_latitude = scaler.fit_transform(x_train_latitude)
x_train_latitude = pd.DataFrame(x_train_scaled_latitude)

x_test_scaled_latitude = scaler.fit_transform(x_test_latitude)
x_test_latitude = pd.DataFrame(x_test_scaled_latitude)
```

In [ ]:
```python
#Performing KNN and picking the model with the best results

best_k_latitude = 0
smallest_error = 0

rmse_values_latitude = []
#iterate through different k and get the results with the least RMSE value
for K in range(10):
    K = K+1
    KNN = sklearn.neighbors.KNeighborsRegressor(n_neighbors = K)

    KNN.fit(x_train_latitude, y_train_latitude)
    pred = KNN.predict(x_test_latitude)
    rmse = sqrt(mean_squared_error(y_test_latitude,pred))
    rmse_values_latitude.append(rmse)


    if best_k_latitude == 0 or smallest_error > rmse:
        best_k_latitude = K
        smallest_error = rmse
        best_predictions_latitude= pred

print('k = ' , best_k_latitude , ', gives the smallest rmse value:', smallest_error)
```

In [ ]:
```python
#plotting the rmse values against k values
rmse_plots = pd.DataFrame(rmse_values_latitude)
plt.title('Rmse values')
plt.xlabel('k values')
plt.ylabel('Rmse values')
plt.plot(rmse_plots)
```

In [ ]:
```python
#Longitude prediction for BikeFinder data

x_train_longitude = train.drop(['latitude','longitude'], axis=1)
y_train_longitude = train['longitude']

x_test_longitude = test.drop(['latitude','longitude'], axis = 1)
y_test_longitude = test['longitude']
```

In [ ]:
```python
#scalling the training values between 0 and 1, to avoid bias results

scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled_longitude = scaler.fit_transform(x_train_longitude)
x_train_longitude = pd.DataFrame(x_train_scaled_longitude)

x_test_scaled_longitude = scaler.fit_transform(x_test_longitude)
x_test_longitude = pd.DataFrame(x_test_scaled_longitude)
```

In [ ]:
```python
#Performing KNN and picking the model with the best results

best_k_longitude = 0
smallest_error = 0

rmse_values_longitude = []
for K in range(10):
    K = K+1
    KNN = sklearn.neighbors.KNeighborsRegressor(n_neighbors = K)

    KNN.fit(x_train_longitude, y_train_longitude)
    pred_longitude = KNN.predict(x_test_longitude)
    rmse = sqrt(mean_squared_error(y_test_longitude,pred_longitude))
    rmse_values_longitude.append(rmse)


    if best_k_longitude == 0 or smallest_error > rmse:
        best_k_longitude = K
        smallest_error = rmse
        best_predictions_longitude= pred_longitude

print('k = ' , best_k_longitude , ', gives the smallest rmse value:', smallest_error)
```

In [ ]:
```python
#plotting the rmse values against k values
rmse_plots = pd.DataFrame(rmse_values_longitude)
plt.title('Rmse values')
plt.xlabel('k values')
plt.ylabel('Rmse values')
plt.plot(rmse_plots)
```

In [ ]:
```python
results = pd.DataFrame()

results['Test_Data_latitude']=y_test_latitude
results['Test_Data_longitude']=y_test_longitude
#print(y_test.shape)

results['Predictions_longitude']=best_predictions_longitude
results['Predictions_latitude']=best_predictions_latitude

#Sorting them by based on the keys from the test data
results = results.sort_index()
results
```

In [ ]:
```python
#test vs latitude prediction

plt.plot(results['Test_Data_latitude'],)
plt.plot(results['Predictions_latitude'])
plt.title('test vs latitude prediction')
plt.xlabel('Indexes')
plt.ylabel('latitude')
plt.legend(['Test Data','K-Nearest Neighbour Predictions'])
```

In [ ]:
```python
#test vs longitude prediction

plt.plot(results['Test_Data_longitude'],)
plt.plot(results['Predictions_longitude'])
plt.title('test vs longitude prediction')
plt.xlabel('Indexes')
plt.ylabel('longitude')
plt.legend(['Test Data','K-Nearest Neighbour Predictions'])
```

Forecasting with BikeFinder theft data

In [ ]:
```python
#forecasting values

# days of the month july
days_next_month = list(range(1, 32))

# generate a list of the month of july 31 times
month = [7] * 31

# generate a list of the year 2022 31 times
year = [2022] * 31

# generate random numbers for hour, minutes and second
hours = []
mins = []
sec = []

for i in range(0,31):
    hours.append(random.randint(0,23))
    mins.append(random.randint(0,59))
    sec.append(random.randint(0,59))


bf_theft_forecast = pd.DataFrame({'days_next_month': days_next_month,'month': month,'year': year,'hour': hours,
#bf_theft_forecast
```

In [ ]:
```python
final_bf_stavanger_theft_data_forecast = final_bf_stavanger_theft_data[['latitude','longitude','year','month','
#final_bf_stavanger_theft_data_forecast
```

In [ ]:
```python
#latitude
x_train_latitude = final_bf_stavanger_theft_data_forecast.drop(['latitude','longitude'], axis=1)
y_train_latitude = final_bf_stavanger_theft_data_forecast['latitude']

x_test_latitude = bf_theft_forecast
```

In [ ]:
```python
#scalling the training values between 0 and 1, to avoid bias results

scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled_latitude = scaler.fit_transform(x_train_latitude)
x_train_latitude = pd.DataFrame(x_train_scaled_latitude)

x_test_scaled_latitude = scaler.fit_transform(x_test_latitude)
x_test_latitude = pd.DataFrame(x_test_scaled_latitude)
```

In [ ]:
```python
#Performing KNN and picking the model with the best results

KNN = sklearn.neighbors.KNeighborsRegressor(n_neighbors = best_k_latitude)

KNN.fit(x_train_latitude, y_train_latitude)
forecast_prediction_latitude = KNN.predict(x_test_latitude)
```

In [ ]:
```python
#longitude
x_train_longitude = final_bf_stavanger_theft_data_forecast.drop(['latitude','longitude'], axis=1)
y_train_longitude = final_bf_stavanger_theft_data_forecast['longitude']

x_test_longitude = bf_theft_forecast
```

In [ ]:
```python
#scalling the training values between 0 and 1, to avoid bias results

scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled_longitude = scaler.fit_transform(x_train_longitude)
x_train_longitude = pd.DataFrame(x_train_scaled_longitude)

x_test_scaled_longitude = scaler.fit_transform(x_test_longitude)
x_test_longitude = pd.DataFrame(x_test_scaled_longitude)
```

In [ ]:
```python
#Performing KNN and picking the model with the best results
KNN = sklearn.neighbors.KNeighborsRegressor(n_neighbors = best_k_longitude)

KNN.fit(x_train_longitude, y_train_longitude)
forecast_prediction_longitude = KNN.predict(x_test_longitude)
```

In [ ]:
```python
results = pd.DataFrame()

results['forecast_prediction_longitude']=forecast_prediction_longitude
results['forecast_prediction_latitude']=forecast_prediction_latitude

#Sorting them based on the keys from the test data
results = results.sort_index()
results


plt.plot(results['forecast_prediction_latitude'])
plt.title('forecast_prediction_latitude')
plt.xlabel('Indexes')
plt.ylabel('latitude')
plt.legend(['Test Data','K-Nearest Neighbour Predictions'])
```

In [ ]:
```python
plt.plot(results['forecast_prediction_longitude'])
plt.title('forecast_prediction_longitude')
plt.xlabel('Indexes')
plt.ylabel('longitude')
plt.legend(['Test Data','K-Nearest Neighbour Predictions'])
```

In [ ]:
```python
results = pd.concat([results, bf_theft_forecast], axis = 1)

results
```

In [ ]:
```python
#show the results in map
fig_1 = folium.Map([59,5.6], zoom_start=11)
for index, row in results.iterrows():
    folium.CircleMarker([row['forecast_prediction_latitude'], row['forecast_prediction_longitude']],
                        radius=3,
                        popup=row['days_next_month'],
                       ).add_to(fig_1)
fig_1
```

In [ ]:
```python
lat = []
long = []

for index, row in results.iterrows():
    lat.append(row["forecast_prediction_latitude"])
    long.append(row["forecast_prediction_longitude"])

lat1=sum(lat)/len(lat)
lat2=sum(long)/len(long)
folium.CircleMarker([lat1,lat2],
                        radius=5,
                        popup="CENTER LOCATION",
                        color='black',
                       ).add_to(fig_1)

print('center point: ',lat1,lat2)
fig_1
```

Clustering with BikeFinder theft data

In [ ]:
```python
#Preparing the data

df_kmeans = final_bf_stavanger_theft_data

K_theft=df_kmeans.iloc[:, 0:2].astype(float).values
```

In [ ]:
```python
range_n_clusters = [3,4,5,6,7,8,9,10,11,12,13,14,15]

print("*****Checking for the optimal number of clusters for latitude and longitude, and getting the results.***


best_k=0
largest_silhouette_av = 0
k_theft_cluster_result = 0

for n_clusters in range_n_clusters:


    kmeans_k_theft = KMeans(n_clusters=n_clusters)
    k_theft_clusters=kmeans_k_theft.fit(K_theft)


    k_theft_silhouette_avg = silhouette_score(K_theft, k_theft_clusters.labels_)


    if best_k == 0 or largest_silhouette_av < k_theft_silhouette_avg:
        best_k = n_clusters
        largest_silhouette_av = k_theft_silhouette_avg
        k_theft_cluster_result = k_theft_clusters.labels_


print('n = ' , best_k , ', gives the largest silhouette_avg value:', largest_silhouette_av,"\n")

#*****************************************************************************************************************
```

In [ ]:
```python
#Sources:
#https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html


#Adding the clustering values to the dataframe as new columns
df_kmeans['k_clusters']=k_theft_cluster_result
```

In [ ]:
```python
#Sources:
# https://matplotlib.org/3.1.0/gallery/subplots_axes_and_figures/subplots_demo.html

# Scatter plott for Latitude and Longitude values
plt.figure(figsize=(10, 7))
plt.title("Longitude and Latitude K-means clustered")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.scatter(df_kmeans['longitude'], df_kmeans['latitude'], c=df_kmeans['k_clusters'], cmap='rainbow')
```

In [ ]:
```python
df_kmeans['k_clusters'].value_counts()
```

Police theft data

In [ ]:
```python
#Load the preprocessed Police theft Data
final_police_stavanger_theft_data = pd.read_csv('police_theft_preprocessed.csv')
final_police_stavanger_theft_data
```

In [ ]:
```python
#Latitude prediction for BikeFinder data

#splitting the data into training and testing parts
train , test = train_test_split(final_police_stavanger_theft_data, test_size = 0.2)

#removing the latitude and longitude values as predictors and adding latitude as response value
x_train_latitude = train.drop(['latitude','longitude'], axis=1)
y_train_latitude = train['latitude']

x_test_latitude = test.drop(['latitude','longitude'], axis = 1)
y_test_latitude = test['latitude']
```

In [ ]:
```python
#scalling the training values between 0 and 1, to avoid bias results

scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled_latitude = scaler.fit_transform(x_train_latitude)
x_train_latitude = pd.DataFrame(x_train_scaled_latitude)

x_test_scaled_latitude = scaler.fit_transform(x_test_latitude)
x_test_latitude = pd.DataFrame(x_test_scaled_latitude)
```

In [ ]:
```python
best_k_latitude = 0
smallest_error = 0

rmse_values_latitude = []
#iterate through different k and get the results with the least RMSE value
for K in range(10):
    K = K+1
    KNN = sklearn.neighbors.KNeighborsRegressor(n_neighbors = K)

    KNN.fit(x_train_latitude, y_train_latitude)
    pred = KNN.predict(x_test_latitude)
    rmse = sqrt(mean_squared_error(y_test_latitude,pred))
    rmse_values_latitude.append(rmse)


    if best_k_latitude == 0 or smallest_error > rmse:
        best_k_latitude = K
        smallest_error = rmse
        best_predictions_latitude= pred

print('k = ' , best_k_latitude , ', gives the smallest rmse value:', smallest_error)
```

In [ ]:
```python
#plotting the rmse values against k values
rmse_plots = pd.DataFrame(rmse_values_latitude)
plt.title('Rmse values')
plt.xlabel('k values')
plt.ylabel('Rmse values')
plt.plot(rmse_plots)
```

In [ ]:
```python
#Longitude prediction for police data

x_train_longitude = train.drop(['latitude','longitude'], axis=1)
y_train_longitude = train['longitude']

x_test_longitude = test.drop(['latitude','longitude'], axis = 1)
y_test_longitude = test['longitude']
```

In [ ]:
```python
#scalling the training values between 0 and 1, to avoid bias results

scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled_longitude = scaler.fit_transform(x_train_longitude)
x_train_longitude = pd.DataFrame(x_train_scaled_longitude)

x_test_scaled_longitude = scaler.fit_transform(x_test_longitude)
x_test_longitude = pd.DataFrame(x_test_scaled_longitude)
```

In [ ]:
```python
#Performing KNN and picking the model with the best results

best_k_longitude = 0
smallest_error = 0

rmse_values_longitude = []
for K in range(10):
    K = K+1
    KNN = sklearn.neighbors.KNeighborsRegressor(n_neighbors = K)

    KNN.fit(x_train_longitude, y_train_longitude)
    pred_longitude = KNN.predict(x_test_longitude)
    rmse = sqrt(mean_squared_error(y_test_longitude,pred_longitude))
    rmse_values_longitude.append(rmse)


    if best_k_longitude == 0 or smallest_error > rmse:
        best_k_longitude = K
        smallest_error = rmse
        best_predictions_longitude= pred_longitude

print('k = ' , best_k_longitude , ', gives the smallest rmse value:', smallest_error)
```

In [ ]:
```python
#plotting the rmse values against k values
rmse_plots = pd.DataFrame(rmse_values_longitude)
plt.title('Rmse values')
plt.xlabel('k values')
plt.ylabel('Rmse values')
plt.plot(rmse_plots)
```

In [ ]:
```python
results = pd.DataFrame()

results['Test_Data_latitude']=y_test_latitude
results['Test_Data_longitude']=y_test_longitude
#print(y_test.shape)

results['Predictions_longitude']=best_predictions_longitude
results['Predictions_latitude']=best_predictions_latitude

#Sorting them by based on the keys from the test data
results = results.sort_index()
results
```

In [ ]:
```python
#test vs latitude prediction
plt.plot(results['Test_Data_latitude'],)
plt.plot(results['Predictions_latitude'])
plt.title('test vs latitude prediction')
plt.xlabel('Indexes')
plt.ylabel('latitude')
plt.legend(['Test Data','K-Nearest Neighbour Predictions'])
```

In [ ]:
```python
#test vs longitude prediction
plt.plot(results['Test_Data_longitude'],)
plt.plot(results['Predictions_longitude'])
plt.title('test vs longitude prediction')
plt.xlabel('Indexes')
plt.ylabel('longitude')
plt.legend(['Test Data','K-Nearest Neighbour Predictions'])
```

Forecasting with Police theft data

In [ ]:
```python
# days of the month july
days_next_month = list(range(1, 32))

# generate a list of the month of july 31 times
month = [7] * 31

# generate a list of the year 2022 31 times
year = [2022] * 31

# generate random numbers for hour
hours = []

for i in range(0,31):
    hours.append(random.randint(0,23))


police_theft_forecast = pd.DataFrame({'days_next_month': days_next_month,'month': month,'year': year,'hour': ho
#police_theft_forecast
```

In [ ]:
```python
final_police_stavanger_theft_data_forecast = final_police_stavanger_theft_data[['latitude','longitude','year','
```

In [ ]:
```python
#latitude
x_train_latitude = final_police_stavanger_theft_data_forecast.drop(['latitude','longitude'], axis=1)
y_train_latitude = final_police_stavanger_theft_data_forecast['latitude']

x_test_latitude = police_theft_forecast
```

In [ ]:
```python
#scalling the training values between 0 and 1, to avoid bias results

scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled_latitude = scaler.fit_transform(x_train_latitude)
x_train_latitude = pd.DataFrame(x_train_scaled_latitude)

x_test_scaled_latitude = scaler.fit_transform(x_test_latitude)
x_test_latitude = pd.DataFrame(x_test_scaled_latitude)
```

In [ ]:
```python
KNN = sklearn.neighbors.KNeighborsRegressor(n_neighbors = best_k_latitude)

KNN.fit(x_train_latitude, y_train_latitude)
forecast_prediction_latitude = KNN.predict(x_test_latitude)
```

In [ ]:
```python
#longitude
x_train_longitude = final_police_stavanger_theft_data_forecast.drop(['latitude','longitude'], axis=1)
y_train_longitude = final_police_stavanger_theft_data_forecast['longitude']

x_test_longitude = police_theft_forecast
```

In [ ]:
```python
#scalling the training values between 0 and 1, to avoid bias results

scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled_longitude = scaler.fit_transform(x_train_longitude)
x_train_longitude = pd.DataFrame(x_train_scaled_longitude)

x_test_scaled_longitude = scaler.fit_transform(x_test_longitude)
x_test_longitude = pd.DataFrame(x_test_scaled_longitude)
```

In [ ]:
```python
KNN = sklearn.neighbors.KNeighborsRegressor(n_neighbors = best_k_longitude)

KNN.fit(x_train_longitude, y_train_longitude)
forecast_prediction_longitude = KNN.predict(x_test_longitude)
```

In [ ]:
```python
results = pd.DataFrame()

results['forecast_prediction_longitude']=forecast_prediction_longitude
results['forecast_prediction_latitude']=forecast_prediction_latitude

#Sorting them by based on the keys from the test data
results = results.sort_index()
results


plt.plot(results['forecast_prediction_latitude'])
plt.title('forecast_prediction_latitude')
plt.xlabel('Indexes')
plt.ylabel('latitude')
plt.legend(['Test Data','K-Nearest Neighbour Predictions'])
```

In [ ]:
```python
plt.plot(results['forecast_prediction_longitude'])
plt.title('forecast_prediction_longitude')
plt.xlabel('Indexes')
plt.ylabel('longitude')
plt.legend(['Test Data','K-Nearest Neighbour Predictions'])
```

In [ ]:
```python
results = pd.concat([results, police_theft_forecast], axis = 1)

results
```

In [ ]:
```python
#show the results in map
fig_1 = folium.Map([59,5.6], zoom_start=11)
for index, row in results.iterrows():
    folium.CircleMarker([row['forecast_prediction_latitude'], row['forecast_prediction_longitude']],
                        radius=3,
                        popup=row['days_next_month'],
                       ).add_to(fig_1)
fig_1
```

In [ ]:
```python
lat = []
long = []

for index, row in results.iterrows():
    lat.append(row["forecast_prediction_latitude"])
    long.append(row["forecast_prediction_longitude"])

lat1=sum(lat)/len(lat)
lat2=sum(long)/len(long)
folium.CircleMarker([lat1,lat2],
                        radius=5,
                        popup="CENTER LOCATION",
                        color='black',
                       ).add_to(fig_1)

print('center point: ',lat1,lat2)
fig_1
```

Clustering with police theft data

In [ ]:
```python
#Preparing the data

df_kmeans = final_police_stavanger_theft_data

K_theft=df_kmeans.iloc[:, 0:2].astype(float).values

range_n_clusters = [3,4,5,6,7,8,9,10,11,12,13,14,15]

print("*****Checking for the optimal number of clusters for latitude and longitude, and getting the results.***


best_k=0
largest_silhouette_av = 0
k_theft_cluster_result = 0

for n_clusters in range_n_clusters:


    kmeans_k_theft = KMeans(n_clusters=n_clusters)
    k_theft_clusters=kmeans_k_theft.fit(K_theft)


    k_theft_silhouette_avg = silhouette_score(K_theft, k_theft_clusters.labels_)


    if best_k == 0 or largest_silhouette_av < k_theft_silhouette_avg:
        best_k = n_clusters
        largest_silhouette_av = k_theft_silhouette_avg
        k_theft_cluster_result = k_theft_clusters.labels_


print('n = ' , best_k , ', gives the largest silhouette_avg value:', largest_silhouette_av,"\n")

#*****************************************************************************************************************
```

In [ ]:
```python
#Sources:
#https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html


#Adding the clustering values to the dataframe as new columns
df_kmeans['k_clusters']=k_theft_cluster_result

#Sources:
# https://matplotlib.org/3.1.0/gallery/subplots_axes_and_figures/subplots_demo.html

# Scatter plott for Latitude and Longitude values
plt.figure(figsize=(10, 7))
plt.title("Longitude and Latitude K-means clustered")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.scatter(df_kmeans['longitude'], df_kmeans['latitude'], c=df_kmeans['k_clusters'], cmap='rainbow')
```

Traffic predictions

traffic predictions.ipynb

Dawit H. Kidane, 15.june.2022

Importing Libraries

```python
#Importing required libraries
from math import sqrt
from sklearn.metrics import mean_absolute_error, mean_squared_error  #for calculation of errors
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import sklearn.neighbors
from sklearn.neighbors import KNeighborsRegressor
from numpy import sqrt
import random
from sklearn.cluster import KMeans #for performing Kmeans
from sklearn.metrics import silhouette_samples, silhouette_score #for silhouette
import random

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from datetime import timedelta
import folium
from folium import plugins
from folium.plugins import MarkerCluster
import warnings
warnings.filterwarnings("ignore")
warnings.simplefilter(action='ignore', category=FutureWarning)
```

BikeFinder traffic data without weather

```python
#Load the BikeFinder bike position data
final_bf_stavanger_position_data = pd.read_csv('bf_traffic_preprocessed.csv')
final_bf_stavanger_position_data
```

```python
#Latitude prediction for BikeFinder data

#splitting the data into training and testing parts
train , test = train_test_split(final_bf_stavanger_position_data, test_size = 0.2)

#removing the latitude and longitude values as predictors and adding latitude as response value
x_train_latitude = train.drop(['latitude','longitude'], axis=1)
y_train_latitude = train['latitude']

x_test_latitude = test.drop(['latitude','longitude'], axis = 1)
y_test_latitude = test['latitude']
```

```python
#scalling the training values between 0 and 1, to avoid bias results

scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled_latitude = scaler.fit_transform(x_train_latitude)
x_train_latitude = pd.DataFrame(x_train_scaled_latitude)

x_test_scaled_latitude = scaler.fit_transform(x_test_latitude)
x_test_latitude = pd.DataFrame(x_test_scaled_latitude)
```

```python
#Performing KNN and picking the model with the best results

best_k_latitude_no_weather = 0
smallest_error = 0

rmse_values_latitude = []
for K in range(10):
    K = K+1
    KNN = sklearn.neighbors.KNeighborsRegressor(n_neighbors = K)

    KNN.fit(x_train_latitude, y_train_latitude)
    pred = KNN.predict(x_test_latitude)
    rmse = sqrt(mean_squared_error(y_test_latitude,pred))
    rmse_values_latitude.append(rmse)


    if best_k_latitude_no_weather == 0 or smallest_error > rmse:
        best_k_latitude_no_weather = K
        smallest_error = rmse
        best_predictions_latitude= pred

print('k = ' , best_k_latitude_no_weather , ', gives the smallest rmse value:', smallest_error)
```

```python
#plotting the rmse values against k values
rmse_plots = pd.DataFrame(rmse_values_latitude)
plt.title('Rmse values')
plt.xlabel('k values')
plt.ylabel('Rmse values')
plt.plot(rmse_plots)
```

```python
#Longitude prediction for BikeFinder data

x_train_longitude = train.drop(['latitude','longitude'], axis=1)
y_train_longitude = train['longitude']

x_test_longitude = test.drop(['latitude','longitude'], axis = 1)
y_test_longitude = test['longitude']
```

```python
#scalling the training values between 0 and 1, to avoid bias results

scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled_longitude = scaler.fit_transform(x_train_longitude)
x_train_longitude = pd.DataFrame(x_train_scaled_longitude)

x_test_scaled_longitude = scaler.fit_transform(x_test_longitude)
x_test_longitude = pd.DataFrame(x_test_scaled_longitude)
```

```python
#Performing KNN and picking the model with the best results

best_k_longitude_no_weather = 0
smallest_error = 0

rmse_values_longitude = []
for K in range(10):
    K = K+1
    KNN = sklearn.neighbors.KNeighborsRegressor(n_neighbors = K)

    KNN.fit(x_train_longitude, y_train_longitude)
    pred_longitude = KNN.predict(x_test_longitude)
    rmse = sqrt(mean_squared_error(y_test_longitude,pred_longitude))
    rmse_values_longitude.append(rmse)


    if best_k_longitude_no_weather == 0 or smallest_error > rmse:
        best_k_longitude_no_weather = K
        smallest_error = rmse
        best_predictions_longitude= pred_longitude

print('k = ' , best_k_longitude_no_weather , ', gives the smallest rmse value:', smallest_error)
```

```python
#plotting the rmse values against k values
rmse_plots = pd.DataFrame(rmse_values_longitude)
plt.title('Rmse values')
plt.xlabel('k values')
plt.ylabel('Rmse values')
plt.plot(rmse_plots)
```

```python
results = pd.DataFrame()

results['Test_Data_latitude']=y_test_latitude
results['Test_Data_longitude']=y_test_longitude
#print(y_test.shape)

results['Predictions_longitude']=best_predictions_longitude
results['Predictions_latitude']=best_predictions_latitude

#Sorting them by based on the keys from the test data
results = results.sort_index()
results
```

```python
plt.plot(results['Test_Data_latitude'],)
plt.plot(results['Predictions_latitude'])
plt.title('test vs latitude prediction')
plt.xlabel('Indexes')
plt.ylabel('latitude')
plt.legend(['Test Data','K-Nearest Neighbour Predictions'])
```

```python
plt.plot(results['Test_Data_longitude'],)
plt.plot(results['Predictions_longitude'])
plt.title('test vs longitude prediction')
plt.xlabel('Indexes')
plt.ylabel('longitude')
plt.legend(['Test Data','K-Nearest Neighbour Predictions'])
```

BikeFinder traffic data with weather

```python
#Load the BikeFinder with weather position data
final_with_weather = pd.read_csv('bf_traffic_weather_preprocessed.csv')
final_with_weather
```

```python
#latitude

train , test = train_test_split(final_with_weather, test_size = 0.2)

x_train_latitude = train.drop(['latitude','longitude'], axis=1)
y_train_latitude = train['latitude']

x_test_latitude = test.drop(['latitude','longitude'], axis = 1)
y_test_latitude = test['latitude']
```

```python
#scalling the training values between 0 and 1, to avoid bias results

scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled_latitude = scaler.fit_transform(x_train_latitude)
x_train_latitude = pd.DataFrame(x_train_scaled_latitude)

x_test_scaled_latitude = scaler.fit_transform(x_test_latitude)
x_test_latitude = pd.DataFrame(x_test_scaled_latitude)
```

```python
#Performing KNN and picking the model with the best results

best_k_latitude = 0
smallest_error = 0

rmse_values_latitude = []
for K in range(10):
    K = K+1
    KNN = sklearn.neighbors.KNeighborsRegressor(n_neighbors = K)

    KNN.fit(x_train_latitude, y_train_latitude)
    pred = KNN.predict(x_test_latitude)
    rmse = sqrt(mean_squared_error(y_test_latitude,pred))
    rmse_values_latitude.append(rmse)


    if best_k_latitude == 0 or smallest_error > rmse:
        best_k_latitude = K
        smallest_error = rmse
        best_predictions_latitude= pred

print('k = ' , best_k_latitude , ', gives the smallest rmse value:', smallest_error)
```

```python
#plotting the rmse values against k values
rmse_plots = pd.DataFrame(rmse_values_latitude)
plt.title('Rmse values')
plt.xlabel('k values')
plt.ylabel('Rmse values')
plt.plot(rmse_plots)
```

```python
#longitude

x_train_longitude = train.drop(['latitude','longitude'], axis=1)
y_train_longitude = train['longitude']

x_test_longitude = test.drop(['latitude','longitude'], axis = 1)
y_test_longitude = test['longitude']
```

```python
#scalling the training values between 0 and 1, to avoid bias results

scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled_longitude = scaler.fit_transform(x_train_longitude)
x_train_longitude = pd.DataFrame(x_train_scaled_longitude)

x_test_scaled_longitude = scaler.fit_transform(x_test_longitude)
x_test_longitude = pd.DataFrame(x_test_scaled_longitude)
```

```python
#Performing KNN and picking the model with the best results

best_k_longitude = 0
smallest_error = 0

rmse_values_longitude = []
for K in range(10):
    K = K+1
    KNN = sklearn.neighbors.KNeighborsRegressor(n_neighbors = K)

    KNN.fit(x_train_longitude, y_train_longitude)
    pred_longitude = KNN.predict(x_test_longitude)
    rmse = sqrt(mean_squared_error(y_test_longitude,pred_longitude))
    rmse_values_longitude.append(rmse)


    if best_k_longitude == 0 or smallest_error > rmse:
        best_k_longitude = K
        smallest_error = rmse
        best_predictions_longitude= pred_longitude

print('k = ' , best_k_longitude , ', gives the smallest rmse value:', smallest_error)
```

```python
#plotting the rmse values against k values
rmse_plots = pd.DataFrame(rmse_values_longitude)
plt.title('Rmse values')
plt.xlabel('k values')
plt.ylabel('Rmse values')
plt.plot(rmse_plots)
```

```python
results = pd.DataFrame()

results['Test_Data_latitude']=y_test_latitude
results['Test_Data_longitude']=y_test_longitude
#print(y_test.shape)

results['Predictions_longitude']=best_predictions_longitude
results['Predictions_latitude']=best_predictions_latitude

#Sorting them by based on the keys from the test data
results = results.sort_index()
results
```

```python
plt.plot(results['Test_Data_latitude'],)
plt.plot(results['Predictions_latitude'])
plt.title('test vs latitude prediction')
plt.xlabel('Indexes')
plt.ylabel('latitude')
plt.legend(['Test Data','K-Nearest Neighbour Predictions'])
```

```python
plt.plot(results['Test_Data_longitude'],)
plt.plot(results['Predictions_longitude'])
plt.title('test vs longitude prediction')
plt.xlabel('Indexes')
plt.ylabel('longitude')
plt.legend(['Test Data','K-Nearest Neighbour Predictions'])
```

Forecasting with BikeFinder traffic data

```python
# days of the month july
days_next_month = list(range(1, 32))

# generate a list of the month of july 31 times
month = [7] * 31

# generate a list of the year 2022 31 times
year = [2022] * 31

# generate random numbers for hour, minutes and second
hours = []
mins = []
sec = []

for i in range(0,31):
    hours.append(random.randint(0,23))
    mins.append(random.randint(0,59))
    sec.append(random.randint(0,59))


bf_traffic_forecast = pd.DataFrame({'day': days_next_month,'month': month,'year': year,'hour': hours})
#bf_traffic_forecast
```

```python
final_bf_stavanger_traffic_data_forecast = final_with_weather[['latitude','longitude','year','month','day','hou
#final_bf_stavanger_theft_data_forecast
```

```python
#latitude
x_train_latitude = final_bf_stavanger_traffic_data_forecast.drop(['latitude','longitude'], axis=1)
y_train_latitude = final_bf_stavanger_traffic_data_forecast['latitude']

x_test_latitude = bf_traffic_forecast
```

```python
#scalling the training values between 0 and 1, to avoid bias results

scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled_latitude = scaler.fit_transform(x_train_latitude)
x_train_latitude = pd.DataFrame(x_train_scaled_latitude)

x_test_scaled_latitude = scaler.fit_transform(x_test_latitude)
x_test_latitude = pd.DataFrame(x_test_scaled_latitude)
```

```python
#Performing KNN and picking the model with the best results

KNN = sklearn.neighbors.KNeighborsRegressor(n_neighbors = best_k_latitude_no_weather)#best_k_latitude)

KNN.fit(x_train_latitude, y_train_latitude)
forecast_prediction_latitude = KNN.predict(x_test_latitude)
```

```python
#longitude
x_train_longitude = final_bf_stavanger_traffic_data_forecast.drop(['latitude','longitude'], axis=1)
y_train_longitude = final_bf_stavanger_traffic_data_forecast['longitude']

x_test_longitude = bf_traffic_forecast
```

```python
#scalling the training values between 0 and 1, to avoid bias results

scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled_longitude = scaler.fit_transform(x_train_longitude)
x_train_longitude = pd.DataFrame(x_train_scaled_longitude)

x_test_scaled_longitude = scaler.fit_transform(x_test_longitude)
x_test_longitude = pd.DataFrame(x_test_scaled_longitude)
```

```python
#Performing KNN and picking the model with the best results

KNN = sklearn.neighbors.KNeighborsRegressor(n_neighbors = best_k_longitude_no_weather)#best_k_longitude)

KNN.fit(x_train_longitude, y_train_longitude)
forecast_prediction_longitude = KNN.predict(x_test_longitude)
```

```python
results = pd.DataFrame()

results['forecast_prediction_longitude']=forecast_prediction_longitude
results['forecast_prediction_latitude']=forecast_prediction_latitude

#Sorting them by based on the keys from the test data
results = results.sort_index()
results


plt.plot(results['forecast_prediction_latitude'])
plt.title('forecast_prediction_latitude')
plt.xlabel('Indexes')
plt.ylabel('latitude')
plt.legend(['Test Data','K-Nearest Neighbour Predictions'])
```

```python
plt.plot(results['forecast_prediction_longitude'])
plt.title('forecast_prediction_longitude')
plt.xlabel('Indexes')
plt.ylabel('longitude')
plt.legend(['Test Data','K-Nearest Neighbour Predictions'])
```

```python
results = pd.concat([results, bf_traffic_forecast], axis = 1)

results
```

```python
#show the results in map
fig_1 = folium.Map([59,5.6], zoom_start=11)
for index, row in results.iterrows():
    folium.CircleMarker([row['forecast_prediction_latitude'], row['forecast_prediction_longitude']],
                        radius=3,
                        popup=row['day'],
                       ).add_to(fig_1)
fig_1
```

```python
lat = []
long = []

for index, row in results.iterrows():
    lat.append(row["forecast_prediction_latitude"])
    long.append(row["forecast_prediction_longitude"])

lat1=sum(lat)/len(lat)
lat2=sum(long)/len(long)
folium.CircleMarker([lat1,lat2],
                        radius=5,
                        popup="CENTER LOCATION",
                        color='black',
                       ).add_to(fig_1)

print('center point: ',lat1,lat2)
fig_1
```

Stavanger bike counter, traffic data without weather

```python
#Load the preprocessed Police theft Data
final_stavanger_position_data = pd.read_csv('city_traffic_preprocessed.csv')
final_stavanger_position_data
```

```python
#latitude

train , test = train_test_split(final_stavanger_position_data, test_size = 0.2)

x_train_latitude = train.drop(['latitude','longitude'], axis=1)
y_train_latitude = train['latitude']

x_test_latitude = test.drop(['latitude','longitude'], axis = 1)
y_test_latitude = test['latitude']
```

```python
#scalling the training values between 0 and 1, to avoid bias results

scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled_latitude = scaler.fit_transform(x_train_latitude)
x_train_latitude = pd.DataFrame(x_train_scaled_latitude)

x_test_scaled_latitude = scaler.fit_transform(x_test_latitude)
x_test_latitude = pd.DataFrame(x_test_scaled_latitude)
```

```python
#Performing KNN and picking the model with the best results

best_k_latitude_no_weather = 0
smallest_error = 0

rmse_values_latitude = []
for K in range(10):
    K = K+1
    KNN = sklearn.neighbors.KNeighborsRegressor(n_neighbors = K)

    KNN.fit(x_train_latitude, y_train_latitude)
    pred = KNN.predict(x_test_latitude)
    rmse = sqrt(mean_squared_error(y_test_latitude,pred))
    rmse_values_latitude.append(rmse)


    if best_k_latitude_no_weather == 0 or smallest_error > rmse:
        best_k_latitude_no_weather = K
        smallest_error = rmse
        best_predictions_latitude= pred

print('k = ' , best_k_latitude_no_weather , ', gives the smallest rmse value:', smallest_error)
```

```python
#plotting the rmse values against k values
rmse_plots = pd.DataFrame(rmse_values_latitude)
plt.title('Rmse values')
plt.xlabel('k values')
plt.ylabel('Rmse values')
plt.plot(rmse_plots)
```

```python
#longitude

x_train_longitude = train.drop(['latitude','longitude'], axis=1)
y_train_longitude = train['longitude']

x_test_longitude = test.drop(['latitude','longitude'], axis = 1)
y_test_longitude = test['longitude']
```

```python
#scalling the training values between 0 and 1, to avoid bias results

scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled_longitude = scaler.fit_transform(x_train_longitude)
x_train_longitude = pd.DataFrame(x_train_scaled_longitude)

x_test_scaled_longitude = scaler.fit_transform(x_test_longitude)
x_test_longitude = pd.DataFrame(x_test_scaled_longitude)
```

```python
#Performing KNN and picking the model with the best results

best_k_longitude_no_weather = 0
smallest_error = 0

rmse_values_longitude = []
for K in range(10):
    K = K+1
    KNN = sklearn.neighbors.KNeighborsRegressor(n_neighbors = K)

    KNN.fit(x_train_longitude, y_train_longitude)
    pred_longitude = KNN.predict(x_test_longitude)
    rmse = sqrt(mean_squared_error(y_test_longitude,pred_longitude))
    rmse_values_longitude.append(rmse)


    if best_k_longitude_no_weather == 0 or smallest_error > rmse:
        best_k_longitude_no_weather = K
        smallest_error = rmse
        best_predictions_longitude= pred_longitude

print('k = ' , best_k_longitude_no_weather , ', gives the smallest rmse value:', smallest_error)
```

```python
#plotting the rmse values against k values
rmse_plots = pd.DataFrame(rmse_values_longitude)
plt.title('Rmse values')
plt.xlabel('k values')
plt.ylabel('Rmse values')
plt.plot(rmse_plots)
```

```python
results = pd.DataFrame()

results['Test_Data_latitude']=y_test_latitude
results['Test_Data_longitude']=y_test_longitude
#print(y_test.shape)

results['Predictions_longitude']=best_predictions_longitude
results['Predictions_latitude']=best_predictions_latitude

#Sorting them by based on the keys from the test data
results = results.sort_index()
results
```

```python
plt.plot(results['Test_Data_latitude'],)
plt.plot(results['Predictions_latitude'])
plt.title('test vs latitude prediction')
plt.xlabel('Indexes')
plt.ylabel('latitude')
plt.legend(['Test Data','K-Nearest Neighbour Predictions'])
```

```python
plt.plot(results['Test_Data_longitude'],)
plt.plot(results['Predictions_longitude'])
plt.title('test vs longitude prediction')
plt.xlabel('Indexes')
plt.ylabel('longitude')
plt.legend(['Test Data','K-Nearest Neighbour Predictions'])
```

Stavanger bike counter, traffic data with weather

```python
#Load the preprocessed Police theft Data
final_stavanger_position_data_weather = pd.read_csv('city_traffic_weather_preprocessed.csv')
final_stavanger_position_data_weather
```

```python
#latitude

train , test = train_test_split(final_stavanger_position_data_weather, test_size = 0.2)

x_train_latitude = train.drop(['latitude','longitude'], axis=1)
y_train_latitude = train['latitude']

x_test_latitude = test.drop(['latitude','longitude'], axis = 1)
y_test_latitude = test['latitude']
```

```python
#scalling the training values between 0 and 1, to avoid bias results

scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled_latitude = scaler.fit_transform(x_train_latitude)
x_train_latitude = pd.DataFrame(x_train_scaled_latitude)

x_test_scaled_latitude = scaler.fit_transform(x_test_latitude)
x_test_latitude = pd.DataFrame(x_test_scaled_latitude)
```

```python
best_k = 0
smallest_error = 0

rmse_values_latitude = []
for K in range(10):
    K = K+1
    KNN = sklearn.neighbors.KNeighborsRegressor(n_neighbors = K)

    KNN.fit(x_train_latitude, y_train_latitude)
    pred = KNN.predict(x_test_latitude)
    rmse = sqrt(mean_squared_error(y_test_latitude,pred))
    rmse_values_latitude.append(rmse)


    if best_k == 0 or smallest_error > rmse:
        best_k = K
        smallest_error = rmse
        best_predictions_latitude= pred

print('k = ' , best_k , ', gives the smallest rmse value:', smallest_error)
```

```python
#plotting the rmse values against k values
rmse_plots = pd.DataFrame(rmse_values_latitude)
plt.title('Rmse values')
plt.xlabel('k values')
plt.ylabel('Rmse values')
plt.plot(rmse_plots)
```

```python
#longitude

x_train_longitude = train.drop(['latitude','longitude'], axis=1)
y_train_longitude = train['longitude']

x_test_longitude = test.drop(['latitude','longitude'], axis = 1)
y_test_longitude = test['longitude']
```

```python
#scalling the training values between 0 and 1, to avoid bias results

scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled_longitude = scaler.fit_transform(x_train_longitude)
x_train_longitude = pd.DataFrame(x_train_scaled_longitude)

x_test_scaled_longitude = scaler.fit_transform(x_test_longitude)
x_test_longitude = pd.DataFrame(x_test_scaled_longitude)
```

```python
best_k = 0
smallest_error = 0

rmse_values_longitude = []
for K in range(10):
    K = K+1
    KNN = sklearn.neighbors.KNeighborsRegressor(n_neighbors = K)

    KNN.fit(x_train_longitude, y_train_longitude)
    pred_longitude = KNN.predict(x_test_longitude)
    rmse = sqrt(mean_squared_error(y_test_longitude,pred_longitude))
    rmse_values_longitude.append(rmse)


    if best_k == 0 or smallest_error > rmse:
        best_k = K
        smallest_error = rmse
        best_predictions_longitude= pred_longitude

print('k = ' , best_k , ', gives the smallest rmse value:', smallest_error)
```

```python
#plotting the rmse values against k values
rmse_plots = pd.DataFrame(rmse_values_longitude)
plt.title('Rmse values')
plt.xlabel('k values')
plt.ylabel('Rmse values')
plt.plot(rmse_plots)
```

```python
results = pd.DataFrame()

results['Test_Data_latitude']=y_test_latitude
results['Test_Data_longitude']=y_test_longitude
#print(y_test.shape)

results['Predictions_longitude']=best_predictions_longitude
results['Predictions_latitude']=best_predictions_latitude

#Sorting them by based on the keys from the test data
results = results.sort_index()
results
```

```python
plt.plot(results['Test_Data_latitude'],)
plt.plot(results['Predictions_latitude'])
plt.title('test vs latitude prediction')
plt.xlabel('Indexes')
plt.ylabel('latitude')
plt.legend(['Test Data','K-Nearest Neighbour Predictions'])
```

```python
plt.plot(results['Test_Data_longitude'],)
plt.plot(results['Predictions_longitude'])
plt.title('test vs longitude prediction')
plt.xlabel('Indexes')
plt.ylabel('longitude')
plt.legend(['Test Data','K-Nearest Neighbour Predictions'])
```

Forecasting with Stavanger bike counter data

```python
# days of the month july
days_next_month = list(range(1, 32))

# generate a list of the month of july 31 times
month = [7] * 31

# generate a list of the year 2022 31 times
year = [2022] * 31

# generate random numbers for hour, minutes and second
hours = []
mins = []
sec = []

for i in range(0,31):
    hours.append(random.randint(0,23))
    mins.append(random.randint(0,59))
    sec.append(random.randint(0,59))


stavanger_traffic_forecast = pd.DataFrame({'day': days_next_month,'month': month,'year': year,'hour': hours})
#stavanger_traffic_forecast
```

```python
final_stavanger_traffic_data_forecast = final_stavanger_position_data[['latitude','longitude','year','month','
#final_stavanger_traffic_data_forecast
```

```python
#latitude
x_train_latitude = final_stavanger_traffic_data_forecast.drop(['latitude','longitude'], axis=1)
y_train_latitude = final_stavanger_traffic_data_forecast['latitude']

x_test_latitude = stavanger_traffic_forecast
```

```python
#scalling the training values between 0 and 1, to avoid bias results

scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled_latitude = scaler.fit_transform(x_train_latitude)
x_train_latitude = pd.DataFrame(x_train_scaled_latitude)

x_test_scaled_latitude = scaler.fit_transform(x_test_latitude)
x_test_latitude = pd.DataFrame(x_test_scaled_latitude)
```

```python
KNN = sklearn.neighbors.KNeighborsRegressor(n_neighbors = best_k_latitude_no_weather)#best_k_latitude)

KNN.fit(x_train_latitude, y_train_latitude)
forecast_prediction_latitude = KNN.predict(x_test_latitude)
```

```python
#longitude
x_train_longitude = final_stavanger_traffic_data_forecast.drop(['latitude','longitude'], axis=1)
y_train_longitude = final_stavanger_traffic_data_forecast['longitude']

x_test_longitude = stavanger_traffic_forecast
```

```python
#scalling the training values between 0 and 1, to avoid bias results

scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled_longitude = scaler.fit_transform(x_train_longitude)
x_train_longitude = pd.DataFrame(x_train_scaled_longitude)

x_test_scaled_longitude = scaler.fit_transform(x_test_longitude)
x_test_longitude = pd.DataFrame(x_test_scaled_longitude)
```

```python
KNN = sklearn.neighbors.KNeighborsRegressor(n_neighbors = best_k_longitude_no_weather)#best_k_longitude)

KNN.fit(x_train_longitude, y_train_longitude)
forecast_prediction_longitude = KNN.predict(x_test_longitude)
```

```python
results = pd.DataFrame()

results['forecast_prediction_longitude']=forecast_prediction_longitude
results['forecast_prediction_latitude']=forecast_prediction_latitude

#Sorting them by based on the keys from the test data
results = results.sort_index()
results


plt.plot(results['forecast_prediction_latitude'])
plt.title('forecast_prediction_latitude')
plt.xlabel('Indexes')
plt.ylabel('latitude')
plt.legend(['Predicted Data','K-Nearest Neighbour Predictions'])
```

```python
plt.plot(results['forecast_prediction_longitude'])
plt.title('forecast_prediction_longitude')
plt.xlabel('Indexes')
plt.ylabel('longitude')
plt.legend(['Predicted Data','K-Nearest Neighbour Predictions'])
```

```python
results = pd.concat([results, stavanger_traffic_forecast], axis = 1)

results
```

```python
#show the results in map
fig_1 = folium.Map([59,5.6], zoom_start=11)
for index, row in results.iterrows():
    folium.CircleMarker([row['forecast_prediction_latitude'], row['forecast_prediction_longitude']],
                        radius=3,
                        popup=row['day'],
                       ).add_to(fig_1)
fig_1
```

```python
lat = []
long = []

for index, row in results.iterrows():
    lat.append(row["forecast_prediction_latitude"])
    long.append(row["forecast_prediction_longitude"])

lat1=sum(lat)/len(lat)
lat2=sum(long)/len(long)
folium.CircleMarker([lat1,lat2],
                        radius=5,
                        popup="CENTER LOCATION",
                        color='black',
                       ).add_to(fig_1)

print('center point: ',lat1,lat2)
fig_1
```