

# Report INF161:

## Predicting available city-bikes

### Contents

1.	Introduction .....	1
2.	Preparing the data .....	2
2.1	stations.csv .....	2
2.2	weather.csv .....	4
2.3	trips.csv.....	5
2.4	Additional changes .....	6
2.5	Additional notes.....	7
3.	Exploring the data.....	7
3.1	Splitting data .....	7
3.2	Checking current value and target value .....	8
3.3	Correlations.....	8
3.4	Graphing comparisons .....	9
3.5	Using a model to find useful variables.....	11
4.	Modelling.....	13
4.1	Testing hyperparameters .....	13
4.1.1	Random Forest Regression model.....	13
4.1.2	Support Vector Regressor .....	13
4.1.3	Multinomial Naïve Bayes.....	14
4.1.4	Ridge Regression.....	14
5.	Deployment.....	15
5.1	train.py .....	15
5.2	Predict.py .....	16
5.3	Results .....	16
6.	Sources for the data .....	17

### 1. Introduction

In this project, I attempt to take 3 datasets containing information on weather, available city-bikes, and registered city-bike trips in Bergen, and use this to train a machine learning model to

correctly predict the number of bikes at a station an hour ahead of time. The model has 9 target stations that it predicts for, those being Møllendalsplass, Grieghallen, Akvariet, Studentboligene, Dreggsallmenningen Sør, Florida Bybanestopp, Torgallmenningen, Damsgårdsveien 71 and Høyteknologisenteret. During the project I will go through 4 important steps in the data-science process, by preparing data, exploring it, building models on the data, and then deploying the results.

## 2. Preparing the data

In this step I describe how I prepared a pipeline, which will take the three sets of raw data (stations.csv, weather.csv, and trips.csv) and combine them in a way that makes them useful for training a model to predict free bikes at a station ahead of time. The pipeline has to be possible to use on the raw datasets whatever their size, since I will split them into training, validation and test data before running them through the pipeline, in order to avoid data-leakage.

### 2.1 stations.csv

I used Stations.csv as a starting point, as it contains the variables that will become the target variables.

	station	longitude	latitude	timestamp	skipped_updates	free_bikes	free_spots
0	Gågaten	5.318798	60.395070	2024-10-26 13:09:58+00:00	6	7	3
1	Bjørnsongsgate	5.342082	60.373986	2025-03-16 10:23:21+00:00	11	0	19
2	Haukeland sykehus	5.356595	60.373638	2025-04-23 09:36:49+00:00	1	3	16
3	Krohnviken	5.331021	60.378107	2024-10-09 18:24:57+00:00	0	4	15
4	Nykirkekaien	5.314548	60.397057	2024-09-29 09:23:49+00:00	3	3	7
5	Møllendalsplass	5.352078	60.379895	2025-02-26 06:49:03+00:00	32	1	16
6	Solheimsvik	5.335575	60.377082	2024-09-06 07:49:09+00:00	1	17	2
7	Torget	5.325284	60.395878	2024-09-17 14:23:31+00:00	1	15	8
8	Laksevåg v/Kirkebukten	5.300108	60.386069	2025-03-30 13:22:24+00:00	4	22	3
9	Busstasjonen 1 Nord	5.333817	60.388910	2024-11-06 12:26:36+00:00	0	3	21

Figure 1: Stations.csv converted to a dataframe

In this dataset, I determined that free-bikes, station and timestamp were the useful variables for our model, with free\_bikes being used to create the target variable.

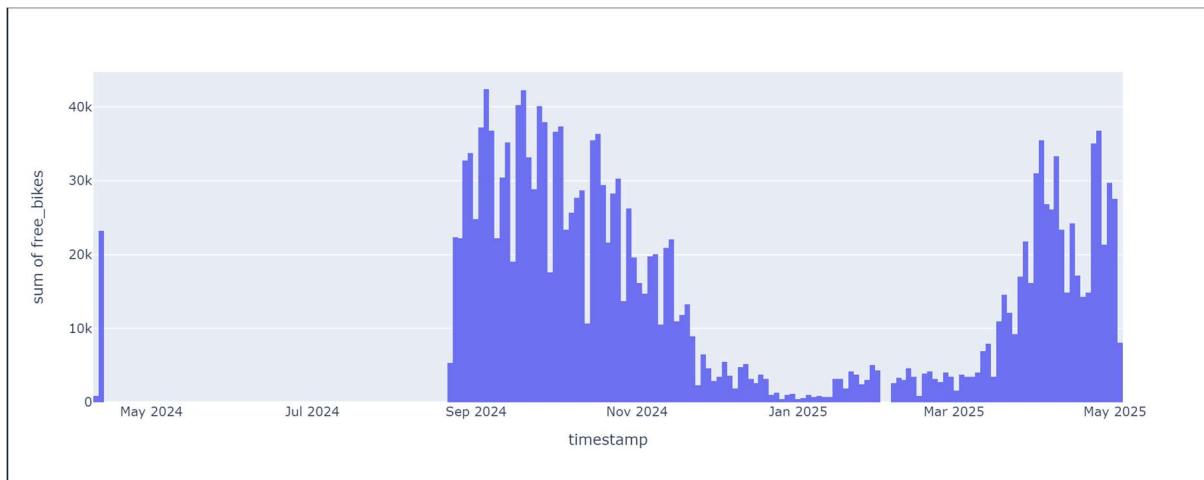


Figure 2: Shape of the dataset, represented using the sum of free bikes over time

Looking at the overall shape of the dataset, I noticed that a large chunk of data was missing over a large time interval. A large missing chunk of data could indicate that it's Missing Not At Random (MNAR), so I decided to only use data from September 2024 and onward.

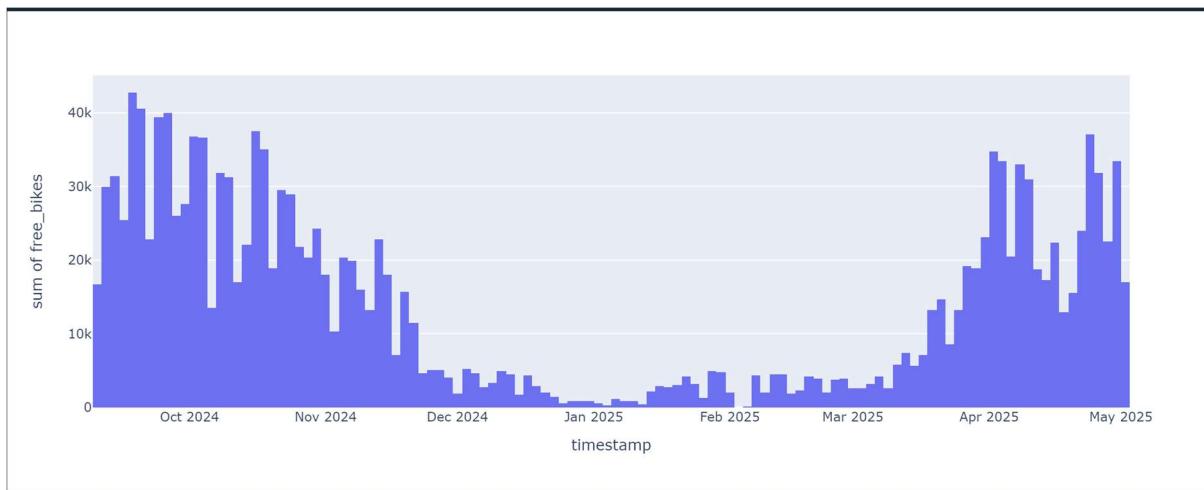


Figure 3: Shape of the dataset from after removing data from before September 2024

Because we are predicting for the next whole hour, I kept only the last measurements made before the next hour for each of the target stations and made a new column with each of these timestamps rounded up to the next whole hour (because a measurement is representative of the number of bikes at a station until the next measurement is made). I then made these rounded timestamps the index of the dataframe, while each target station received its own column with the corresponding number of free bikes for each timestamp as values. I also kept the non-rounded timestamps, for later use.

I then filled in rows for all the missing timestamps, so that there was 1 row for each hour in the time interval the dataset covers. These rows had no values, but because a number of bikes is valid until the next measurement, I could simply forward fill the last measured value to fill in the missing ones.

## 2.2 weather.csv

The weather dataset was already formatted in a practical way, so I didn't need to change the shape.

	timestamp	temperature	precipitation	wind_speed
0	2023-12-31 23:00:00+00:00	3.0	0.0	17.9
1	2024-01-01 00:00:00+00:00	2.8	0.0	13.2
2	2024-01-01 01:00:00+00:00	2.6	0.0	10.5
3	2024-01-01 02:00:00+00:00	2.7	0.0	18.9
4	2024-01-01 03:00:00+00:00	2.1	0.0	18.4

Figure 4: Weather.csv as a dataframe

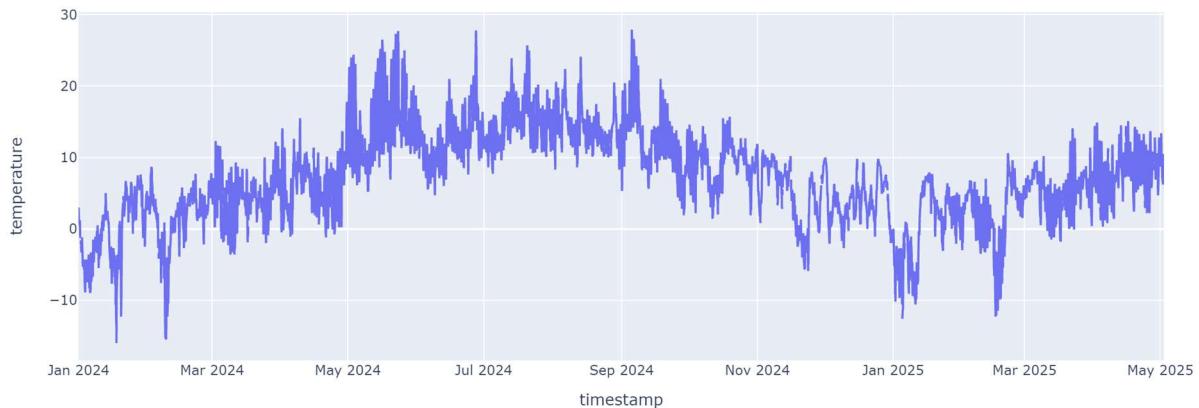


Figure 5: Temperature over time

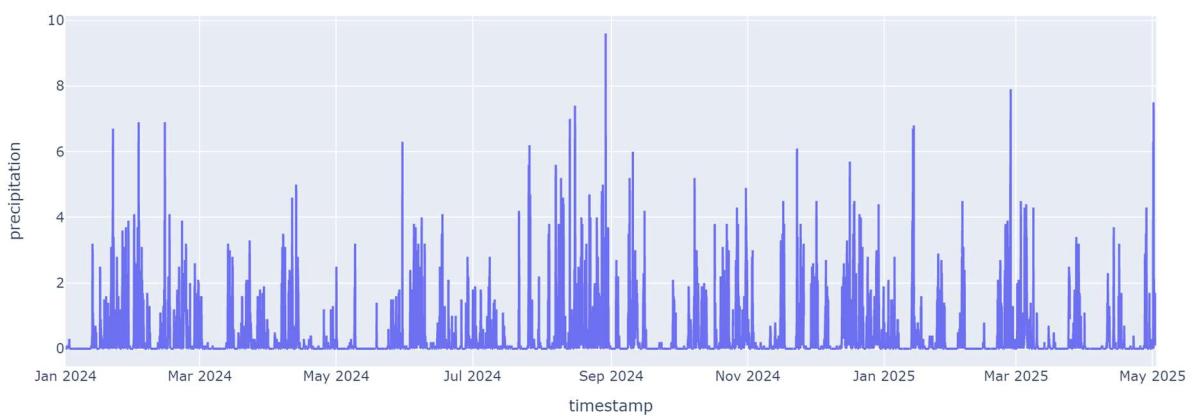
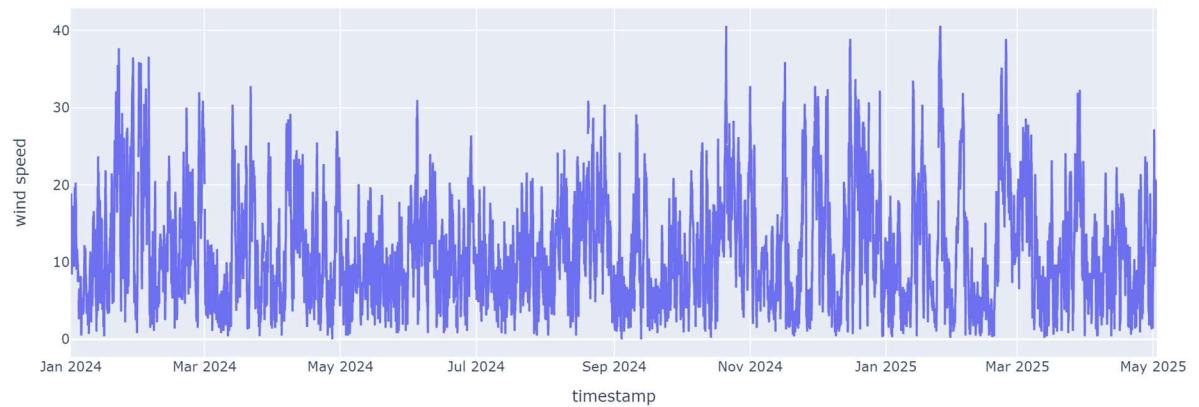


Figure 6: Precipitation over time (mm)



*Figure 7: Wind speed over time*

The absence of large gaps of measurements in our dataset suggests that there are no MNAR type missing data in this dataset.

I deemed temperature, precipitation and wind speed to be possibly useful variables, so I just merged the weather dataframe with the stations dataframe (on their shared timestamps) without changing it.

## 2.3 trips.csv

Because the trips dataset was in a different format than the other datasets, I decided to create a new variable from the existing data in the dataset.

	started_at	ended_at	start_station_name	start_station_latitude	start_station_longitude	end_station_name	end_station_latitude	end_station_longitude
0	2023-01-01 04:22:50.614000+00:00	2023-01-01 04:33:19.884000+00:00	Torget	60.395878	5.325284	Takhagen på Nordnes	60.398865	
1	2023-01-01 04:31:33.724000+00:00	2023-01-01 04:33:21.525000+00:00	Kong Oscars gate	60.393323	5.330654	Kong Oscars gate	60.393323	
2	2023-01-01 04:33:41.500000+00:00	2023-01-01 04:44:24.625000+00:00	Kong Oscars gate	60.393323	5.330654	Skutevikstorget	60.402229	
3	2023-01-01 05:55:30.388000+00:00	2023-01-01 05:59:17.644000+00:00	John Lunds plass	60.388247	5.324558	Skur 11	60.396384	
4	2023-01-01 05:59:33.946000+00:00	2023-01-01 06:03:47.999000+00:00	Skur 11	60.396384	5.324169	Hans Hauges gate	60.401906	

*Figure 8: trips.csv as a dataframe, not the right format to combine with the other datasets*



Figure 9: The total amount of trips for a few of the target stations over time

No big gaps between measurements in the dataset suggests that there is no MNAR type missing data here.

I decided to use the amount of total trips per hour as a measurement for overall traffic, as an hour with more trips (busier hour) or less trips could show useful patterns in the data when compared with other variables, and free bikes (more traffic means more changes in free bikes, and more bikes being in use means less bikes would be available). I used the starting time for each trip to count the trips per hour (started\_at column) and added a column with traffic at each hour to the combined station and weather dataset. I also added rows for all missing timestamps, and filled them with the value 0, as an hour with no measured trips has 0 trips per hour, so 0 traffic.

## 2.4 Additional changes

After combining the three datasets I made some additional changes to the dataset before it was completed.

- Adding a column for target values.
  - Because the goal was to make one prediction per row, I had to multiply each row by 9 (one for each target station). I then had to add a column for each target station with each row of the column containing either 1 if were predicting for that station in said row, or 0 if not (one hot encoding).
  - The target feature row was filled in using the number of free bikes for the currently selected station, one hour ahead of time.
- Moving current number of free bikes into a single row.
  - I decided using other target stations as training data for predicting number of bikes for a target station would be messy and could pull attention away from the variables I was interested in predicting for, so I created a single column containing only the number of free bikes for the station currently selected in the row.
  - In my opinion, in order for the number of free bikes at other stations to be useful variables, I would have to use the data from all the stations in Bergen (which

would be much more time consuming), and the bikes would have to exist in a closed system (which they don't, as bikes can be added to stations by the company, or removed for maintenance for example).

- Forward filling values in the combined dataset, so that values that are missing at random take the last measurement. I think forward fill is the best imputing method in this case, as weather is the only part of the dataset that can still be missing values, and weather changes over time, and will thus probably be more similar to the last measured value for weather.
- Creating separate columns for the hour, month and day of the week.
  - Because timestamp is equivalent to a string and I will be using regression models, I had to make new columns with parts of the timestamp represented as integers. I did this instead of just not using timestamp because I think the day of the week, month of the year and hour of the day could have an effect on the number of free bikes.
- Dropping rows that still contain null values
  - Because I use forward fill to impute missing values, some of the first rows of the combined dataset could still be missing values, as they have no values to fill from. I decided to simply remove these rows, as I didn't think another imputing method would be appropriate, and they would most likely be a tiny fraction of the data anyway.

## 2.5 Additional notes

Because I use the stations dataframe as a starting point, the other dataframes are adjusted to match its shape when they are merged with it. I also don't use any imputing methods that take other values before fitting the other datasets to stations. This means that I only need to split the stations dataframe into training, validation and test data before running it through the pipeline, while I can run the other two dataframes through unparted without any risk of data leakage. This solves an issue of the time-periods of the dataframes not matching up with each other after splitting them into train, val and test data.

# 3. Exploring the data

## 3.1 Splitting data

Before exploring the data, it had to be split into training, validation and test data, and then passed separately through the pipeline. I only explored the training data, to avoid data-leakage and looking at future data.

Because the data is sequential (linearly time-based), it has to be split chronologically, with the training data being the oldest and the test data being the most recent. This is in part because training the model on more recent data could give a misleading idea of how well it generalises on new data, since the data might have long term trends that our model does not pick up on.

### 3.2 Checking current value and target value

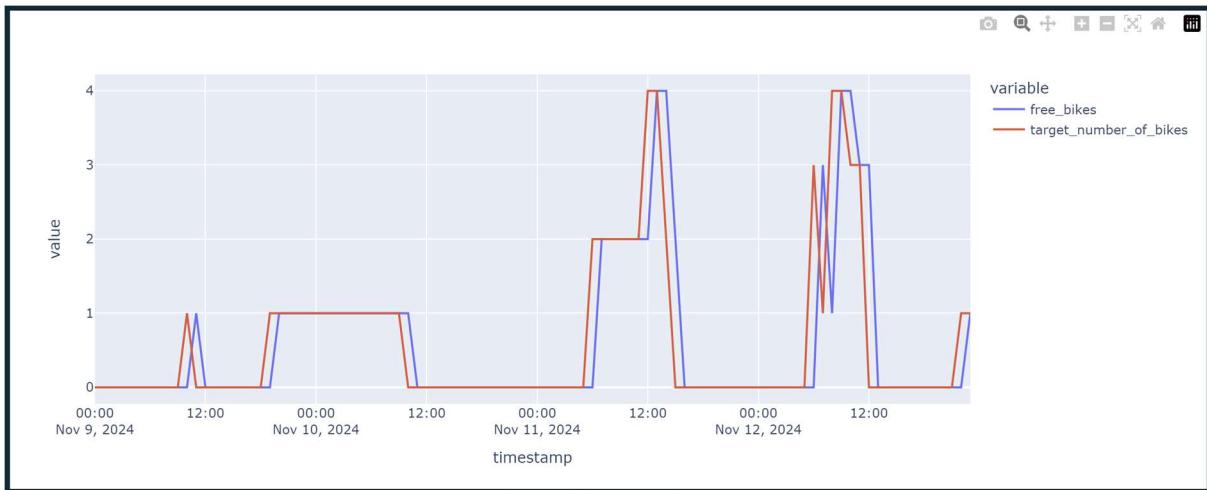


Figure 10: Number of free bikes at Akvariet and target number of free bikes

Graphing out the current number of free bikes for a station compared to the target variable of the station can show us whether our target variables have been filled in correctly (it appears they were)

### 3.3 Correlations

I started by looking at how correlated each variable was with the target variable, to get an idea of which variables might be more important.

target_number_of_bikes	1.000000
free_bikes	0.976039
temperature	0.378670
traffic	0.201226
selected_Dreggsallmenningen_Sør	0.189336
month	0.142533
selected_Torgallmenningen	0.099473
selected_Damsgårdveien71	0.051715
selected_Studentboligene	0.017803
hour	-0.004764
selected_Grieghallen	-0.005282
precipitation	-0.008534
day_of_week	-0.010898
selected_Møllendalsplass	-0.018108
wind_speed	-0.041985
selected_Florida	-0.059009
selected_Høyteknologisenteret	-0.084641
selected_Akvariet	-0.191286

Figure 11: Correlations between each variable and the target variable (target\_number\_of\_bikes)

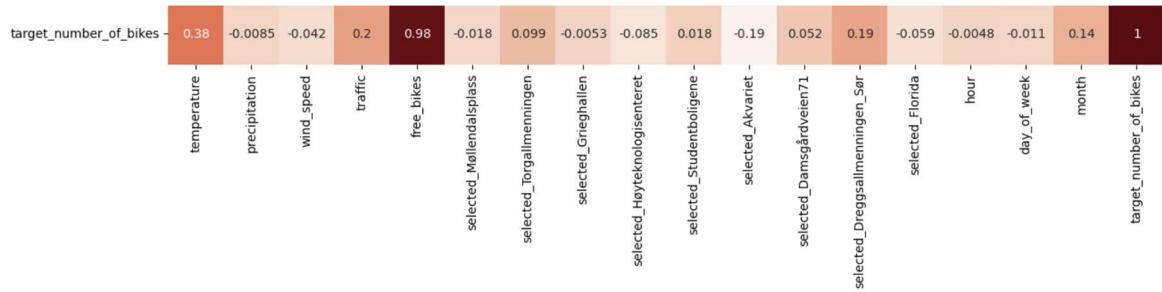


Figure 12: Heatmap of the correlations between each variable and the target variable (target\_number\_of\_bikes)

Here we can see that free\_bikes has a large correlation with the target number of bikes, which makes sense, as there will probably be a similar number of free bikes at a station from hour to hour. Temperature, traffic and month also have stronger correlations, suggesting that maybe how busy an hour is, and the time of year have an effect on free bikes.

selected\_Dreggsallmenningen\_Sør also has a stronger correlation, suggesting that this station generally has a higher number of free bikes (because the station being selected is increasing its value (0-1) and a higher correlation means that values increase at similar rates, so as the stations value increases (it is selected) target\_number\_of\_bikes also increases (more free bikes))

### 3.4 Graphing comparisons

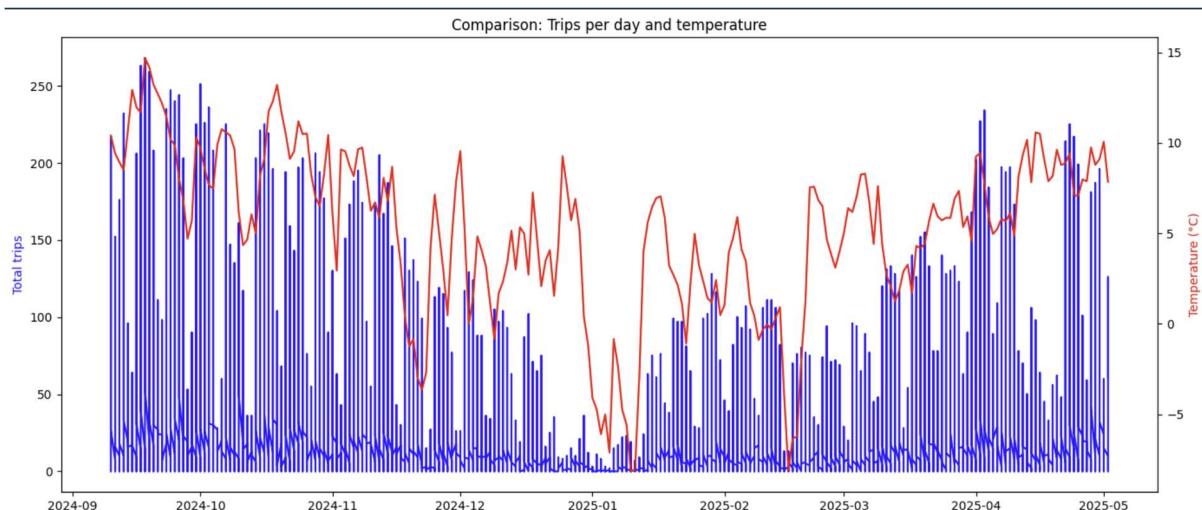


Figure 13: Total trips per day over time compared with temperature

Graphing out total trips per day over time compared with temperatures shows that they follow a similar pattern, suggesting these two variables may be connected.

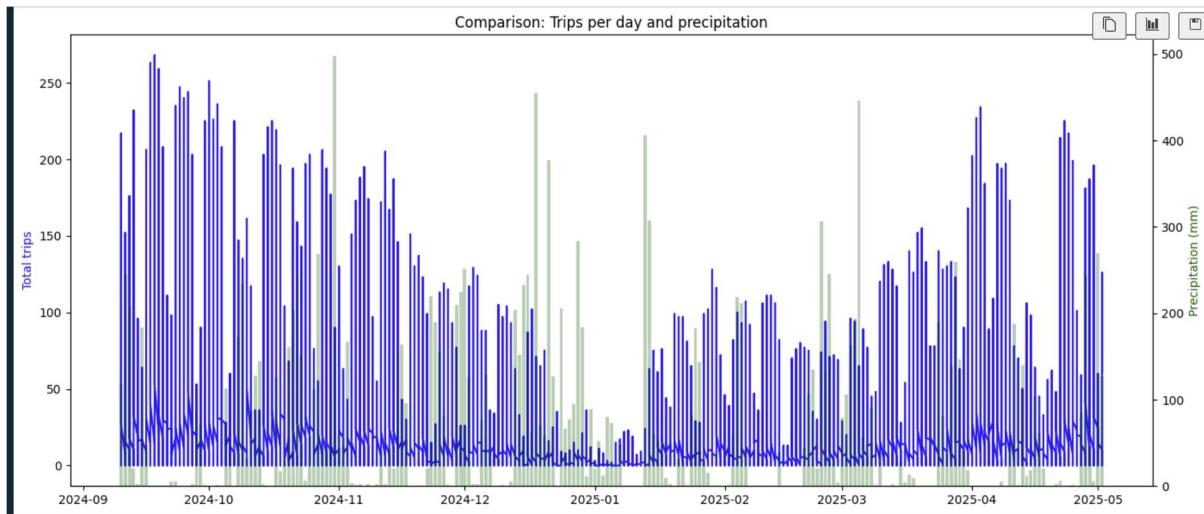


Figure 14: Trips per day compared to precipitation per day (mm)

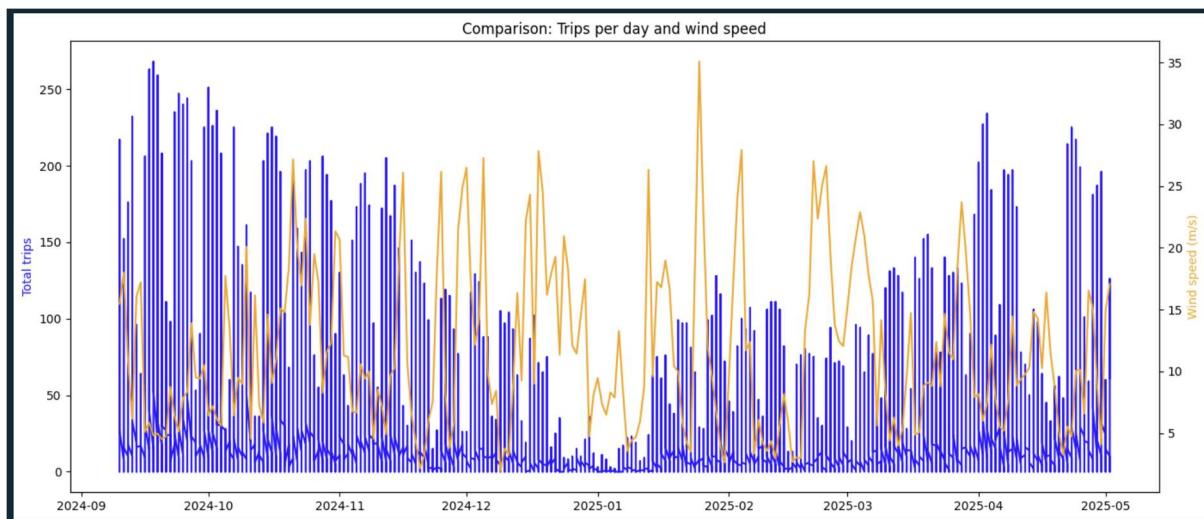


Figure 15: Trips per day and wind speed (m/s)

Wind speed and precipitation seem less connected with the number of total trips per day, indicated that these variables may be less important.

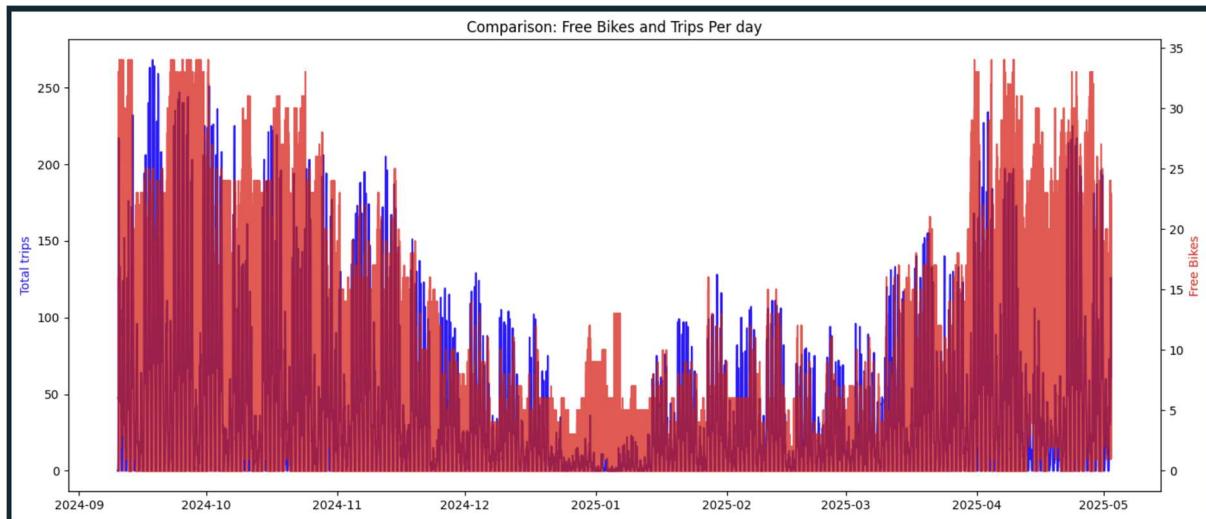


Figure 16: Traffic (trips per day) and free bikes per day over a longer period

The similar shapes of the traffic and free bikes show that traffic could be a useful variable in predicting free bikes for the next hour.

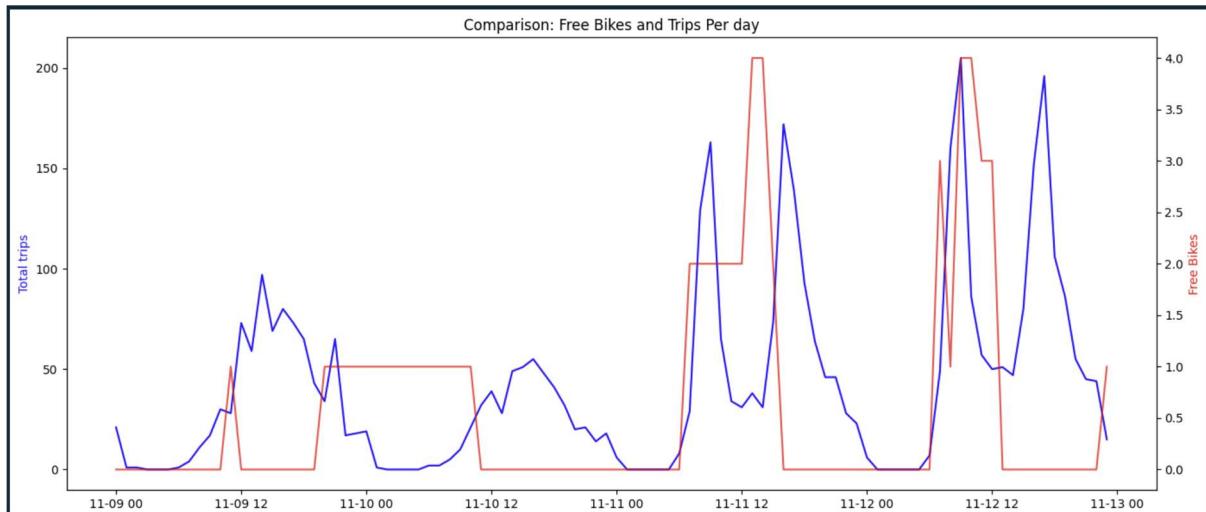


Figure 17: Traffic (trips per day) and free bikes per day over a shorter period

Over a shorter period, the similarity in shape is less clear, though there appears to be some similarity still.

### 3.5 Using a model to find useful variables

```
Random Forest Root Mean Squared Error: Train RMSE: 0.5460387742955196 Val RMSE: 1.4952488031613271
```

Figure 18: Root Mean Squared Error of our random forest model (training data and validation data)

	feature names	feature importances
4	free_bikes	0.953650
3	traffic	0.010606
0	temperature	0.007252
2	wind_speed	0.007119
14	hour	0.005224
15	day_of_week	0.002869
16	month	0.002843
1	precipitation	0.002191
8	selected_Høgtekknologisenteret	0.001934
13	selected_Florida	0.001121
6	selected_Torgallmenningen	0.001011
5	selected_Møllendalsplass	0.000904
11	selected_Damsgårdveien71	0.000838
9	selected_Studentboligene	0.000730
12	selected_Dreggsallmenningen_Sør	0.000716
10	selected_Akvariet	0.000497
7	selected_Grieghallen	0.000496

Figure 19: Feature Importances, showing how much each feature influenced the random forest predictions

We can also train a random forest model and examine the feature importances.

Though the model appears to be quite overfit (much higher validation RMSE than training RMSE) it will do for our purposes.

In the case of feature importances, random forest appears the most important still, while precipitation, month and day of week appear less important.

We can try to train the model without free\_bikes first to see the effect:

```
Random Forest Root Mean Squared Error: Train RMSE: 0.9281355492248645 Val RMSE: 3.2718359721214068
```

Figure 20: RMSEs for a random forest model trained without the free\_bikes variable

Removing free\_bikes lead to far higher RMSE scores, while still keeping a large gap between training and validation RMSEs, suggesting that free\_bikes is an important variable, and that it's not largely responsible for the model being overfit.

We can also remove precipitation, month and day of week and train another model

```
Random Forest Root Mean Squared Error: Train RMSE: 0.5500894543164168 Val RMSE: 1.4892375536905755
```

Figure 21: RMSEs for a random forest model trained without the precipitation, month and day\_of\_week variable

Removing these variables didn't significantly decrease the training time of the model, and slightly increased the RMSE scores. Due to there being no upside to removing these variables, I decided to keep them

## 4. Modelling

After having processed and explored our data, I can train some models on it and find the best model.

### 4.1 Testing hyperparameters

#### 4.1.1 Random Forest Regression model

I decided to use a Random Forest Regression model, due to its effectiveness on larger scale datasets. I tested the model for different amount of estimators (100 – 900) giving these results:

	train_rmse	val_rmse
900	0.536063	1.487196
700	0.536504	1.487396
800	0.536071	1.487470
400	0.538752	1.487631
300	0.538652	1.487751
600	0.537165	1.487897
500	0.537836	1.487966
200	0.540454	1.488226
100	0.546039	1.495249

Figure 22: RMSE scores for a random forest with 100-900 estimators

It appears a more complex tree gave a better RMSE score, though all the scores indicate that the model is quite overfit, as the training RMSE is much lower than the validation RMSE.

#### 4.1.2 Support Vector Regressor

I also tried a Support Vector Regressor, a type of Support Vector Machine, due to it being effective in time-series predictions. Trying different kernel functions gave these results:

	train_rmse	val_rmse
linear	1.494926	1.388929
rbf	1.488527	1.389487
poly	3.223781	3.220925
sigmoid	1825.477649	1825.828500

Figure 23: RMSE scores for an SVR model with different Kernel functions

In this case the linear kernel function was the most accurate, while the sigmoid function was decidedly worst. The model linear kernel gave a better validation RMSE score than the random forest, and appears less overfit (though the training RMSE is higher than the validation RMSE, which could indicate underfitting).

#### 4.1.3 Multinomial Naïve Bayes

I tried using a Multinomial Naïve Bayes model, because it's very efficient to train. Trying a range of alphas gave the results:

	train_rmse	val_rmse
0.01	8.015032	8.073036
1.01	8.054665	8.118808
2.01	8.079546	8.140878
3.01	8.103102	8.160315
4.01	8.115301	8.178076
5.01	8.127687	8.191677
6.01	8.141661	8.206948
7.01	8.150101	8.216041
8.01	8.159277	8.225782
9.01	8.167190	8.231619

Figure 24: RMSE scores for a MNB model with different Alphas

The results of this model were much worse than the previous 2, suggesting that our data is not as well suited for this type of model. (MNB is better suited for classification tasks, with discrete variables instead of continuous ones.)

#### 4.1.4 Ridge Regression

Finally, I tried training a Ridge Regression model, as they are effective at preventing overfitting on training data. I also trained this model with a range of alphas:

	<b>train_rmse</b>	<b>val_rmse</b>
9.01	1.487469	1.383540
8.01	1.487469	1.383540
7.01	1.487469	1.383540
6.01	1.487469	1.383540
5.01	1.487469	1.383540
4.01	1.487469	1.383541
3.01	1.487469	1.383541
2.01	1.487469	1.383541
1.01	1.487469	1.383541
0.01	1.487469	1.383541

Figure 25: RMSE scores for a Ridge Regression model with different alphas

The ridge regression model displays similar RMSE scores to the SVR model, and the different alphas make little difference.

## 5. Deployment

The deployment of the models will happen in 2 scripts, one where the best model is determined, its ability to generalise is tested, and it is saved to a pickle file. The other will use the model to predict the amount of bikes at all the target stations for the hour following the last measurement in the dataset.

### 5.1 train.py

The train.py script handles the model selection and evaluation, with a function “pipeline” that handles the processing of data, “model\_selection” that determines the best of the 4 models discussed above (and a Linear Regression model used as a baseline), and “evaluation” that evaluates the model returned from the model\_selection function on the test data.

The pipeline works as described in the Preparing the Data section and is run on the stations data after it is split into training, validation and test data (as well as weather data and trips data, though these are not split beforehand, as previously mentioned). The three processed datasets are then separated into X\_train and y\_train, X\_val and y\_val, and X\_test and y\_test.

The model selection function trains and saves the 4 different models with different hyperparameters, ranks each model by best RMSE, and then compares the models, returning the model with the lowest validation RMSE.

The evaluation function then uses test data on the best model to get an expected RMSE score, and saves the model using pickle.

## 5.2 Predict.py

The predict.py file runs the un-split dataset through the pipeline and then uses the 9 last rows and the model saved to pickle to predict the expected number of free bikes for the next hour. It then prints the results in a table.

## 5.3 Results

The best model overall turned out to be the Ridge Regression Model, which generalised quite well when compared to the validation RMSE

```
| Expected RMSE: 1.38928948270387 | Validation RMSE: 1.3835399847074081 | Best Model: Ridge Regressor |
```

Figure 26: Results of the train.py script, with the best model overall being the Ridge Regression model.

The final predictions turned out like this:

Siste tidsstempel i data:	2025-05-02 17:49:25+02:00	
Neste hele klokkeetime:	2025-05-02 18:00:00+02:00	
Predikerer for tidsstempel:	2025-05-02 19:00:00+02:00	
<hr/>		
Stasjon	Nåværende Sykler	Predikerte Sykler
<hr/>		
Møllendalsplass	4	4
Torgallmenningen	20	20
Grieghallen	16	16
Høyteknologisenteret	24	23
Studentboligene	8	8
Akvariet	1	1
Damsgårdsveien 71	13	13
Dreggsallmenningen Sør	17	17
Florida Bybanestopp	21	20

Figure 27: Table with the final prediction, and relevant timestamps

Overall, the RMSE score could be better, probably by tuning the hyperparameters in the model some more. I think this model (given some more tuning) could be useful in a real life scenario, where you'd want to know ahead of time if there are any bikes available when you have to leave.

## 6. Sources for the data

Weather : <https://open-meteo.com>

Stations and trips: <https://bergenbysykkel.no/en/open-data>