Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CS7CS4 Machine Learning Final Assignment

I have read, and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at: http://www.tcd.ie/calendar

I have also completed the Online Tutorial on avoiding plagiarism 'Ready, Steady, Write', located at http://tcd-ie.libguides.com/plagiarism/ready-steady-write

Student Name & Number	Aditya Kumar Shrivastava TCD ID: 19323354
Module	CS7CS4- MACHINE LEARNING
Name of Demonstrator	Professor Douglas Leith
Title of Assignment	Final Assignment
Date	04-Jan-2022

1. Evaluating the feasibility of predicting bike station occupancy 10 minutes, 30 minutes and 1 hour into the future.

(a) Initial Design Choices

Bike stations selected to study: Station 19: Herbert Place & Station 10: Dame Street

I chose to study bike stations 10 and 19 because station 10 is located in the city whereas station 19 is located in a residential area and is on a popular commuter route. This results in different bike availability and usage patterns which can be observed in figures 1(a) & 1(b). Figures 1(a) and 1(b) below show the number of available bikes between Monday 10-Feb to Monday 18-Feb for both stations.

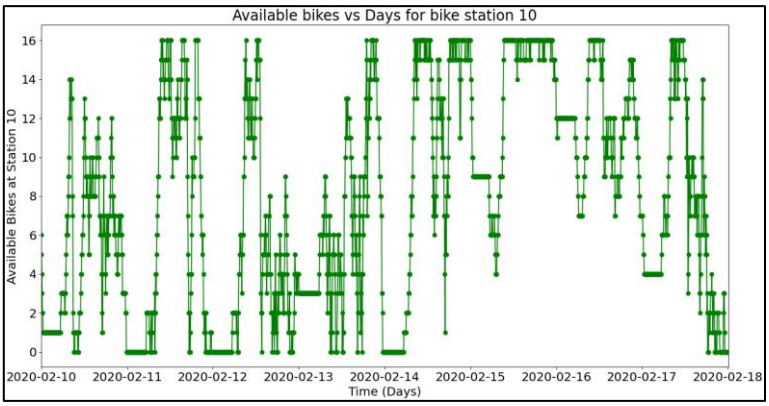


Figure 1(a): Number of available bikes between 10-Feb and 18-Feb for station 10

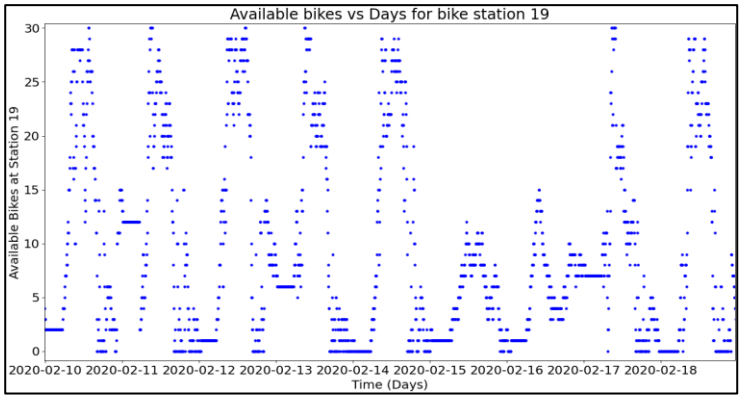


Figure 1(b): Number of available bikes between 10-Feb and 18-Feb for station 19

(ii) Selecting the date range for the data.

The "Dublinbikes 2020 Q1 usage data" dataset is measured between 01-Jan-2020 and 01-Apr-2020. On observing the dataset, I noticed that the dataset had missing measurements for various dates in Jan 2020. From figure 2(a), we can observe missing time measurements for the number of available bikes in Jan 2020.

Missing data points may lead to problems in analysing the data and catching the trends and seasonality. Based on my feature engineering technique explained in section (b), the input to the implemented models will contain data from previous weeks, days and timestamps. Missing data points in my method of implementing features and models may not lead to the best possible predictions or the number of available bikes.

Therefore, I selected the data range between Wednesday 29 Jan 2020 and Wednesday 1 Apr. The trimmed dataset will have no missing data points or measurements. The trimmed dataset for station 10 can be observed in Figure 2b.

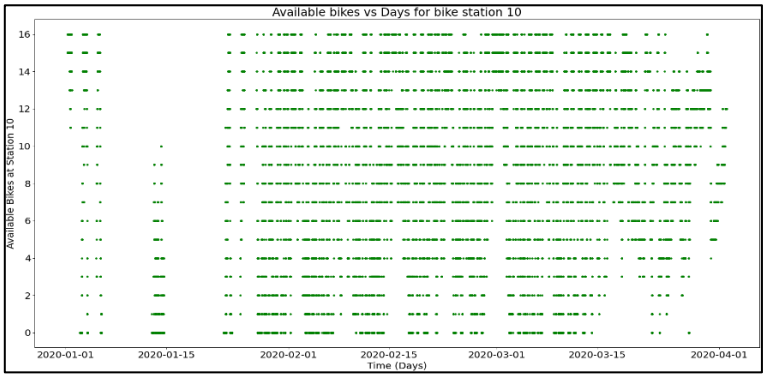


Figure 2(a): Number of available bikes vs time for station 19- Missing Points.

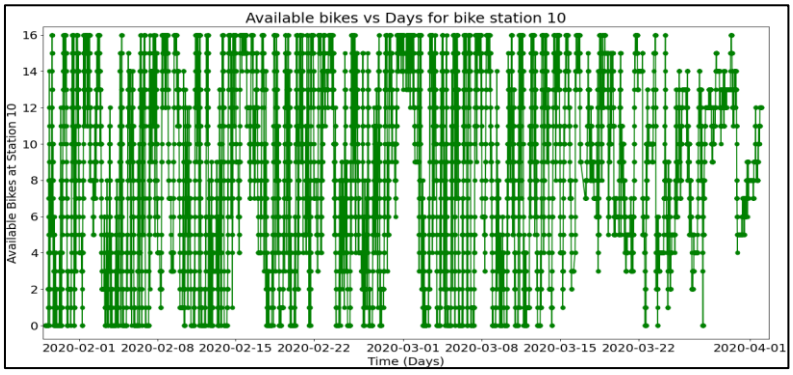


Figure 2(b): Number of available bikes vs time station 19. No missing data points

(b) Feature Engineering

Section (b) contains how the features were engineered to predict bike station occupancy

Observation on trends and seasonality:

We can observe several things from figures 1(a) & 2(b). There is a short-term correlation between the number of bikes at nearby times, i.e., if the number of bikes is high at 01-Feb 13:05, then it is likely that the number of bikes is also high at nearby times. There is also a regular pattern on weekdays and weekends. In Figures 1(a) & (b), we can observe the difference in bikes on the weekends (15 and 16 Feb) and the weekdays. We can also observe a regular pattern of bikes available on each weekday in figure 1(a) and 1(b). The weekly and daily seasonality for station 10 can be observed in figures 3(a, b) and 4(a, b).

There is a short-term trend and a daily as well as weekly seasonality in the dataset for the number of available bikes. On observing the trend and seasonality, I decided to model input features based on short-term trends, the daily seasonality, and the weekly seasonality.

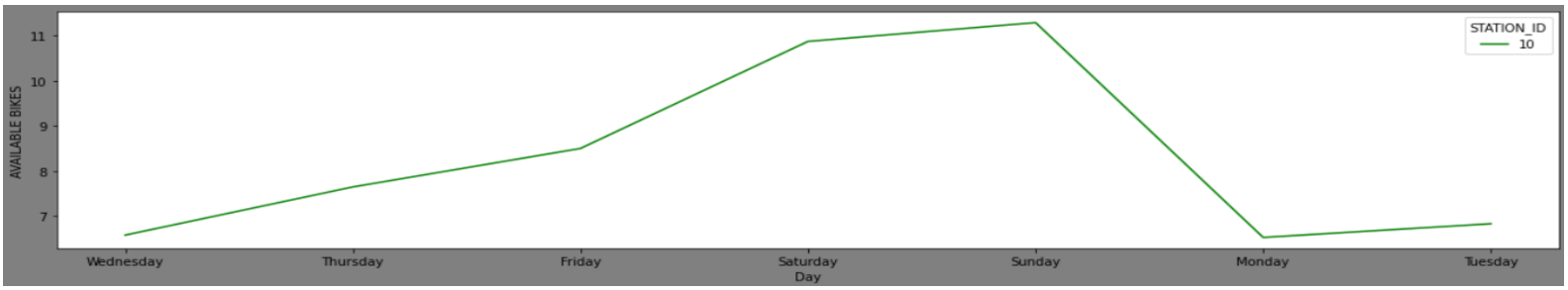


Figure 3(a): Weekly Seasonality - Pattern of available bikes each day of the week for station 10

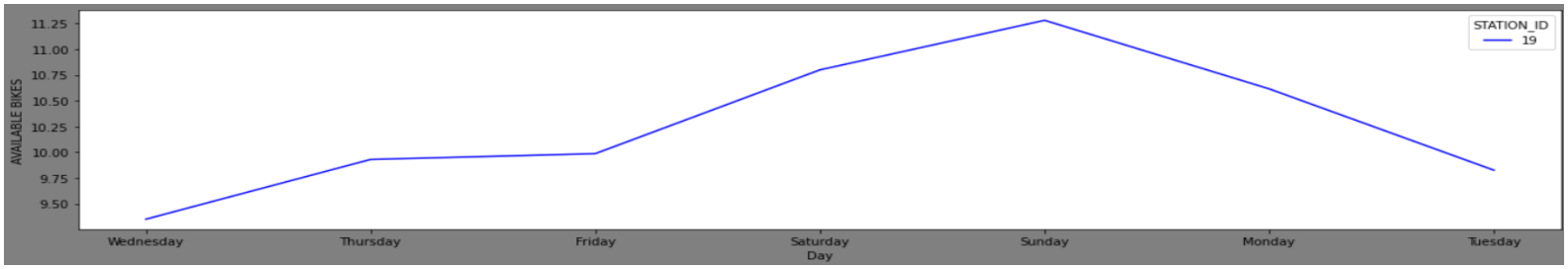


Figure 3(b): Weekly Seasonality - Pattern of available bikes each day of the week for station 19

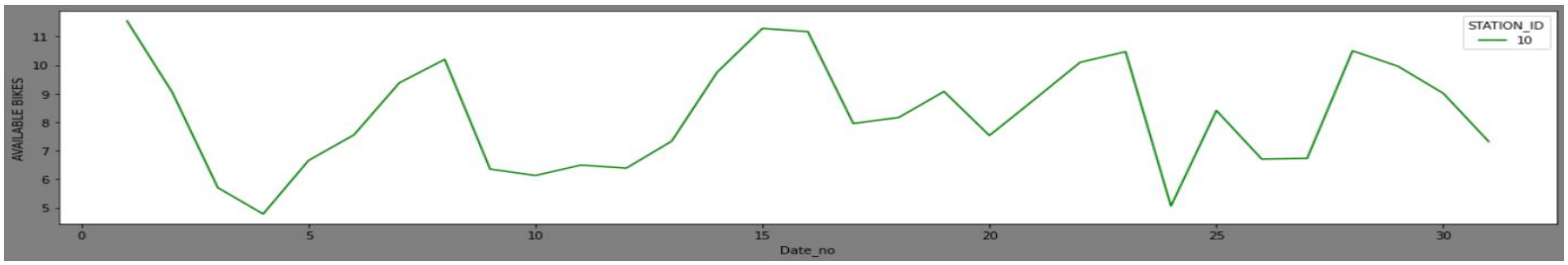


Figure 4(a): Daily Seasonality- Pattern of available bikes each date of a month for station 10

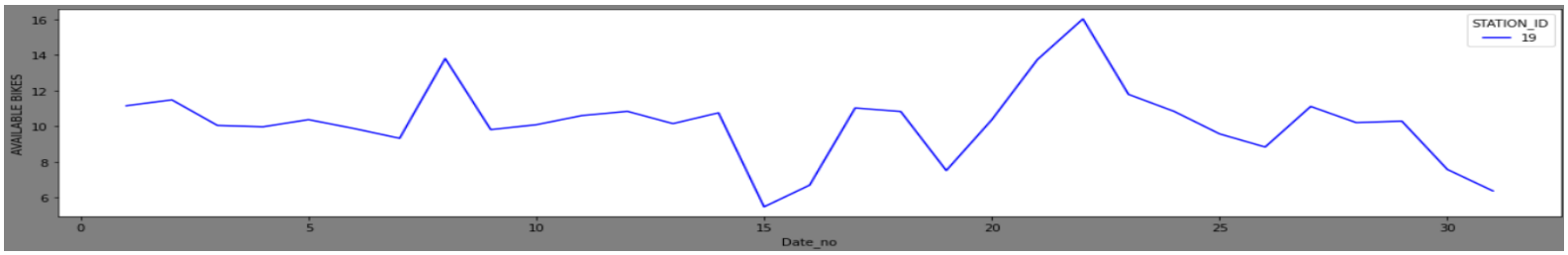


Figure 4(b): Daily Seasonality- Pattern of available bikes each date of a month for station 10

Process of engineering features:

1. The dataset used to predict bike station occupancy at a certain time into the future is a time series where each term has a timestamp.

2. To predict the bike station occupancy, i.e. the number of bikes available at a station, the original dataset is trimmed down to contain the target value $y^{(k)}$ which is the number of available bikes at a station. The timestamp is used as an index in the dataframe that holds $y^{(k)}$.

3. k is the number of the current measurement and the time $t^{(k)}$ of the measurement can be measured by $k * timeSamplingInterval$. The number of available bikes is measured regularly every 5 minutes. Therefore, our time sampling interval is 5 minutes. This implies that $t^{(k)} = k * 5mins$ and our data sequence is $y^{(k)}$ where k=1,2,…

4. The task hand is to predict the bike station occupancy **q-steps** ahead into the future. If we have data up to time $k-1$, the task is to predict $y^{(k-1+q)}$ which would be a time $t^{(k-1+q)} = (k-1+q) * 5mins$. The inputs to perform a q-step ahead prediction would look like $[y^{(1)}, y^{(2)} \ldots y^{(k-1)}]$.
   For example, to predict bike occupancy 10 minutes ahead into the future, step-size **q=2**. If k=3 and q=2, the inputs would be $[y^{(1)}, y^{(2)}]$ measured at times $[t^{(1)} = 5mins,\ t^{(2)} = 10mins]$ and the output would be $y^{(4)}$ measured at time $t^{(4)} = 20\ mins$

5. We can calculate the step sizes q by the formula $q = \frac{time\ you\ wish\ to\ predict\ ahead}{timeSamplingInterval}$. Therefore, to predict bike station occupancy 10 minutes, 30 minutes and 1 hour into the future, the step sizes would be 2, 6, and 12.

6. The input features XX are modelled using $y^{(k)}$. To model the input features, I have used the short-term trends, the daily patterns, and the weekly patterns available in the data.
   a. If we want to predict $y^{(k-1+q)}$ using short term trends using the two most recent values, the feature vector would contain values $[y^{(k-2-q)}, y^{(k-1-q)}]$. We can pass a parameter 'lag' to determine how many recent values we would like to consider. E.g. If lag is 4, then the feature vector will consider the four most recent time measurements, and the feature vector would look like $[y^{(k-4-q)}, y^{(k-3-q)}, y^{(k-2-q)}, y^{(k-1-q)}]$.
   To model short-term features in my code, a new column is appended to the bikes_dataframe and this column stores the measurement of the previous timestamp by shifting the data ahead in a loop that runs between 0 and lag.

```
for data_point in range(0, lag):
        available_bikes_df.loc[:, f"Bikes_{data_point+1}_point_before"] = available_bikes_df.loc[:, "AVAILABLE BIKES"].shift(data_point + 1)
```

   b. Along with short-term trends, the feature vector also includes measurements from the previous day(s) and previous week(s) to capture the seasonality. Suppose we want to include measurements from the two most recent days and two most recent weeks to predict $y^{(k-1+q)}$, the feature vector will contain values $[y^{(k-2d)}, y^{(k-1d)}, y^{(k-2w)}, y^{k-1w}]$, where *d=number_of_measurements_per_day* and *w=number_of_measurements_per_week*. We can pass the **'lag'** parameter to indicate how many previous days and weeks we would like to consider in the feature vector.
   To model daily and weekly features in code, new columns are appended to the bikes_dataframe. These columns stores the measurement of the previous day(s) and week(s) by shifting the number of available bikes ahead by d*(lag+1) for days and week*(lag+1) for weeks in a loop that runs between 0 and lag.

```
num_samples_per_day = math.floor(24 * 60 * 60 / time_sampling_interval_dt); num_samples_per_week = math.floor(7 * 24 * 60 * 60 / time_sampling_interval_dt)
```

```
for day in range(0, lag):
        available_bikes_df.loc[ :, f"Bikes_{day+1}_day_ago" ] = available_bikes_df.loc[:, "AVAILABLE BIKES"].shift(num_samples_per_day * (day + 1))
for week in range(0, lag):
        available_bikes_df.loc[:, f"Bikes_{week+1}_week_ago"] = available_bikes_df.loc[:, "AVAILABLE BIKES"].shift(num_samples_per_week * (week + 1))
```

7. If we are predicting the bike station occupancy **q-steps** ahead into the future for an input feature vector X where the most recent point was measured at e.g. *2020-01-29 00:20:02*, the output vector will contain the number of bikes available at time $(q_{steps} * 5mins) + (2020/01/29\ 00{:}20{:}02)$. This has been modelled in the code by shifting the measurement of each timestamp by -q_steps and storing this measurement in a new column. This ensures that for each measurement at a timestamp, the new column stores the number of available bikes q_steps ahead.

```
available_bikes_df.loc[:, f"bikes_avail_{q}_mins_ahead"] = available_bikes_df.loc[:, "AVAILABLE BIKES"].shift(-q_step_size)
```

8. We select input features to predict the number of available bikes as a **sliding window**. To illustrate how the sliding window works in this project, suppose we are predicting bike station occupancy 10 minutes ahead into the future for point 'a' at timestamp 2020-02-05 00:25:02, and we are using the previous 3 measurements in our input feature vector. Our input feature vector would contain the bikes available at 2020-02-05 00:15:02, 2020-02-05 00:10:02 and 2020-02-05 00:5:02 respectively. Now, if we want to predict the next point 'b' at timestamp 2020-02-05 00:35:02, we step forward through the data and our input feature vector would contain points measured at 2020-02-05 00:25:02, 2020-02-05 00:20:02 & 2020-02-05 00:15:02 respectively. As we move on to predict point 'b' from point 'a', our input features slide ahead by 5 minutes in terms of their timestamp. The same concept applies to sliding input feature measurements next day or next week. Therefore, when predicting bike station occupancy in the future, our input features slide forward each time we want to predict the next point into the future.

9. The dataframe is finally split into feature vector XX and output vector yy.

```
XX = df_features.drop([f"bikes_avail_{q}_mins_ahead"], axis=1);  yy = df_features[[f"bikes_avail_{q}_mins_ahead"]]
```

Input and output vectors if lag=1 and q_step = 2 to predict 10 minutes ahead into the future

```
>>> XX_ridge
                     Bikes_1_point_before  Bikes_1_day_ago  Bikes_1_week_ago
TIME
2020-02-05 00:00:02                   0.0              0.0               0.0
2020-02-05 00:05:02                   0.0              0.0               0.0
2020-02-05 00:10:02                   0.0              0.0               0.0
2020-02-05 00:15:02                   0.0              0.0               0.0
2020-02-05 00:20:02                   0.0              0.0               0.0
```

```
>>> yy_ridge
                     bikes_avail_10.0_mins_ahead
TIME
2020-02-05 00:00:02                          0.0
2020-02-05 00:05:02                          0.0
2020-02-05 00:10:02                          0.0
2020-02-05 00:15:02                          0.0
2020-02-05 00:20:02                          0.0
```

(c) Machine Learning Methodology:

This section describes the machine learning methods implemented to predict bike station occupancy 10 minutes, 30 minutes and 1 hour into the future. Furthermore, we select critically analyse and select features. Following feature selection, we choose the best versions of the ridge and kNN model by performing cross validation to tune model hyperparameters.
Following modelling the feature vectors, I chose to implement two machine learning approaches to predict bike station occupancy. These approaches are:

1. Ridge Regression, and
2. K-Nearest Neighbours for regression

I chose Ridge regression and kNN for regression because the way in which Ridge and kNN make predictions are vastly different.

- kNN is an instance-based model and makes predictions directly based on the training data. For regression problems, kNN calculates the weighted mean of the outputs $y^{(i)}$ for the k closest training points and uses the weighted mean as the prediction. kNN is implemented using sklearn's KNeighborsRegressor, and the target value is predicted by local interpolation of the target values associated with the nearest neighbours in the training set.

- Ridge regression makes predictions by applying a linear model hθ(x) to an input feature vector (parameters θ) with a quadratic penalty (L2 regularisation) added to the cost function. Ridge then applies the least-squares optimisation to select parameters or weights that minimise the cost function. Ridge regression is implemented using sklearn's Ridge, where the model solves a regression model where the loss function is the linear least-squares function and regularisation is given by the l2-norm.

On selecting the 2 models, we decide what features are important by fitting the ridge regression model and plotting the weight value given to each feature in sorted order. We use sklearn's RidgeCV that performs efficient Leave-One-Out Cross-Validation to figure out feature importance to fit the feature vector.

Feature Selection

To choose the features we use in our models, we perform 4 experiments. The experiments are as follows:

1. **Experiment 1:** Deciding the lag value by performing lag cross-validation (CV). *(click here to see the meaning of lag)*
   To decide the best value of the 'lag' parameter for the ridge (C=1) and kNN (k=1) models to predict bike station occupancy 10,30 and 60 minutes into the future, we train each model for each step ahead prediction (see point 5 in feature engineering) for a range of lag values between 1 to 6. For each step ahead prediction, we calculate and plot the negative mean error and the standard deviation for the range of lag values. The negative mean error is just the positive mean squared error with a negative sign. Sckit-learn uses the negative MSE as it tries to improve model performance by solving a maximisation problem and tries to maximise the negative MSE(bringing the negative MSE to 0). The plots for each model and step ahead prediction can be observed in figures A(1,2,3) and B(1,2,3) below:

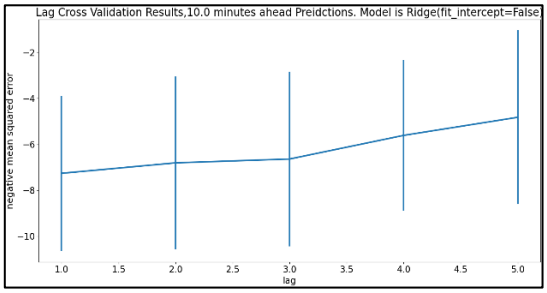


Figure A1: Ridge regression lag CV, step size = 2. Station 19

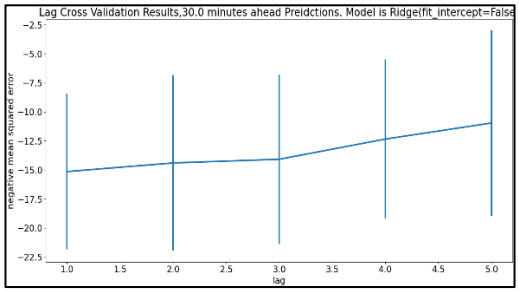
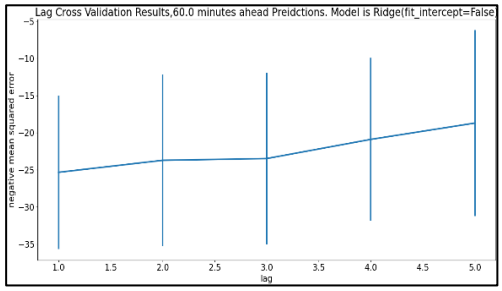


Figure A2: Ridge regression lag CV, step size = 6. Station 19



A3: Ridge regression lag CV, step size = 12. Station 19

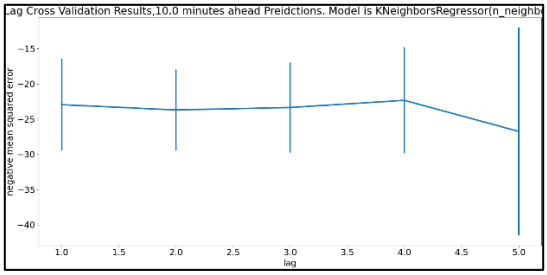


Figure B1: kNN lag CV, step size = 2. Station 19

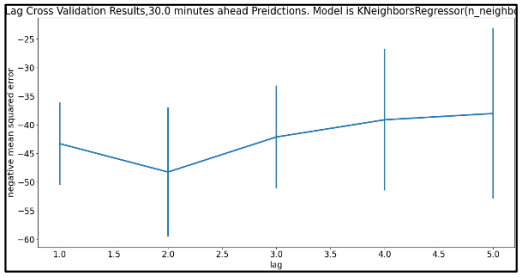


Figure A2: kNN lag CV, step size = 6. Station 19

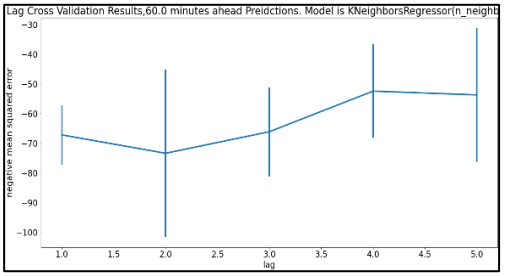


Figure A3: kNN lag CV, step size = 12. Station 19

From figures A1, A2, & A3, I chose the lag value to be 2 for the ridge model. This is so because even though we have a more negative MSE (further away from 0), having a lower lag value ensures that we have a larger number of timestamps/measurements for which we can predict bike station occupancy. For example, suppose we are predicting bike station occupancy 60 minutes into the future, and our feature vector is using measurements from the past 4 weeks, i.e. lag = 4. This would mean that the first point we would be able to predict bike station occupancy for would be = timestamp_of_first_measurement (Wednesday 29 Jan 2020)+ 4_weeks, equal to 26th Feb 2020.
Since the lag was higher (4), we had fewer points to measure our models' performance. When we have lower points to evaluate our model on, we may get over-optimistic results which is why we see a steady increase in the negative MSE (heading closer to 0) in figures A1, A2, & A3. Having a higher value of lag, we also have more input features, and this may be computationally expensive when we scale up the input dataset. Therefore it makes sense here to choose a lower value of lag.

We attain a similar plot with the same trend observed in Figures A1, A2, A3 when performing lag cross-validation (ridge model) for 30 & 60 minutes ahead predictions for station 19. On performing lag cross-validation for station 10, ridge regression model, We observe a very similar plot with the same trend observed in Figures A1, A2, A3 for 10, 30 & 60 minutes ahead predictions. Therefore the lag value of the features used in the ridge regression model is taken to be 2 for station 10 and station 19. From figures B1, B2, & B3, I chose the lag value to be 2 for the kNN model for the same reasons mentioned for the ridge regression lag choice. Having a smaller value of lag would allow us to test the model's predictions on more points, ensuring a more robust model evaluation. As mentioned above, while performing this experiment, similar trends and plots detected in figures B1,B2 B3 were observed when performing lag cross-validation (kNN model) for 30 & 60 minutes ahead predictions for stations 19 and 10, 30 & 60 minutes ahead predictions for station 10. Therefore the lag value for kNN models for stations 10 and 19 is also lag=2 *[Note: I was unable to paste the plots for station 10 due to space constraints but the plots can be found in my code submission folder path: Plots/LagCrossValidation/Station10]*

2. **Experiment 2:** After selecting the lag value to be 2 for both the models for stations 10 and 19, we progressively add and remove features and fit a ridge regression model and kNN (KNeighborRegressor) model to see the impact on predictions for both stations. To see the impact of features, we vary the types of input features that a model can utilise. We then observe the performance of the model on varying the features. The performance of a model is observed by analysing the plots showing the predicted bikes vs the actual number of bikes as well as interpreting the model's mean squared error values. The input features used, fall under 3 main categories, which are:
    i. Features based on short term trends in the input data
    ii. Features based on the daily seasonality in the input data
    iii. Features based on the weekly seasonality in the input data

➢ Table 1 below shows the variation of input features used to fit a ridge regression model are represented in:

	short term trends	daily seasonality	weekly seasonality
Features included in Run 1	☑	☒	☒
Features included in Run 2	☒	☑	☒
Features included in Run 3	☒	☒	☑
Features included in Run 4	☑	☑	☒
Features included in Run 5	☑	☒	☑
Features included in Run 6	☒	☑	☑
Features included in Run 7	☑	☑	☑

Table 1: Adding/Removing features to observe their impact

➢ Tables 2(a) shows the mean squared error for a ridge regression model (C=0.05) trained on each set of input features for station 19 i.e. each Run mentioned in table 1 above.
Table 2(b) shows the mean squared error for a kNN regression (k=1) model trained on each set of input features for station 19 i.e. each Run mentioned in table 1 above

Ridge Regression Model, alpha=10	**Mean Square Errors for the set of features used in each run in table 1 above.**						
Station 19	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Run 6**	**Run 7**
10 minutes ahead predictions	7.6598	49.7843	53.7952	7.3454	6.8043	46.4198	6.7535
30 minutes ahead predictions	15. 9125	50.9148	55.0598	15.1138	14.3059	47.8639	14.1618
60 minutes ahead predictions	25.9285	54.0988	57.6340	24.7894	23.0450	51.3874	22.8919

Table 2(a)

KNN Regression Model. Number of neighbours = 1	**Mean Square Errors for the set of features used in each run in table 1 above.**						
Station 19	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Run 6**	**Run 7**
10 minutes ahead predictions	9.4113	56.357	63.5964	6.8012	6.1498	30.8470	7.4997
30 minutes ahead predictions	21.4126	60.8004	64.7071	14.5828	12.6928	32.4394	14.5664
60 minutes ahead predictions	38.3944	63.2184	66.1063	23.7288	19.5390	36.6588	20.8756

Table 2(b)

➢ From tables 1, 2(a), and 2(b), we can observe that both models have the minimum mean squared error when the set of input features consists of the short term trends, daily seasonality, and weekly seasonality i.e., Run 7 from table 1. This is the case for 10, 30 and 60 minutes ahead predictions. On comparing run 5 that is tested with features capturing short-term trends and weekly seasonality, with run 7, we notice that there is only a small decrease in the mean square error when the input feature vector captures the short-term trends, daily and the weekly seasonality. Even though the improvement in the model is small in run 7, it is important as capturing the trends and seasonalities available may help the model scale well when implemented with a larger dataset of bikes. Capturing the short term trends, daily & weekly seasonality is important as it improves a model's predictions and brings the predictions closer to the actual outputs.

- The above step of experiment 2 is performed for station 10 and the same results are obtained as that for station 19. Station 10 has the best performance when the input feature vector contains the set of features mentioned in run 7 of table 1. Therefore for stations 10 and 19, we can conclude that capturing the short term trends, daily & weekly seasonality is important as it improves a model's predictions and brings the

predictions closer to the actual outputs.

- To illustrate and observe the difference in predictions while using several different sets of input features, I have plotted the predictions vs expected outputs for each run as well as step ahead prediction and analysed them. Due to space constraints on the report, my observations are illustrated through an example where we observe the plots for 60 minutes ahead predictions between run 5 and run 7 for station 19 (refer table 1).
    - Figures 5(a) & 5(b) show the plot for the number of available bikes vs the predicted number of available bikes when predicting 60 minutes in the future using the ridge regression model for station 19. Figure 5(a) shows the plot for the ridge regression model trained using the set of features outlined in *Run 5 of table 1* whereas figure 5(b) shows the plot for the ridge regression model trained using the set of features outlined in *Run 7 of table 1*.
        - From figures 5(a) & 5(b) we can observe that the ridge model trained using the set of features in Run 7 (see figure 5b) is predicting bike station occupancy closer to the actual bike station occupancy. The evidence for this can be noticed in the outlined areas in red in figures 5(a) and 5(b).

    - Figures 6(a) & 6(b) show the plot for the number of available bikes vs the predicted number of available bikes for station 19 when predicting 60 minutes in the future using a kNN regression (KNeighborsRegressor) model. Figure 6(a) shows the plot for the kNN model trained using the set of features outlined in *Run 5 of table 1,* whereas figure 6(b) shows the plot for the kNN model trained using the set of features outlined in *Run 7 of table 1*.
        - From figures 6(a) & 6(b) we can observe that the kNN trained using the set of features in Run 7 (see figure 6b) is predicting bike station occupancy closer to the actual bike station occupancy. The evidence for this can be noticed in the outlined areas in red in figures 6(a) and 6(b).

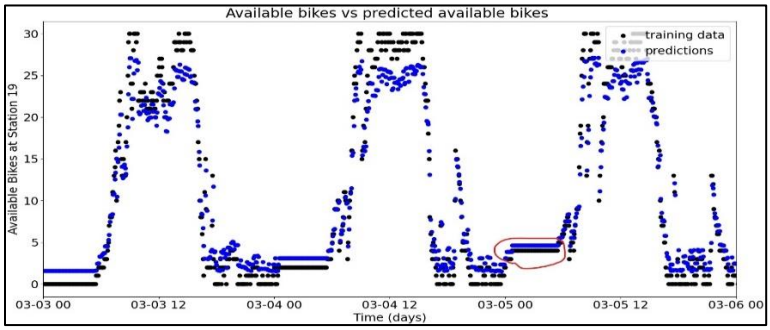


*Figure 5(a): **Run 5**, 60 minutes ahead predictions using **Ridge Regression. Station 19***

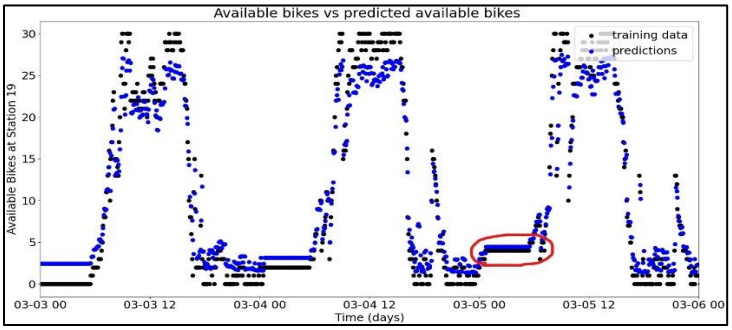


*Figure 5(b): **Run 7**, 60 minutes ahead predictions using **Ridge Regression. Station 19***

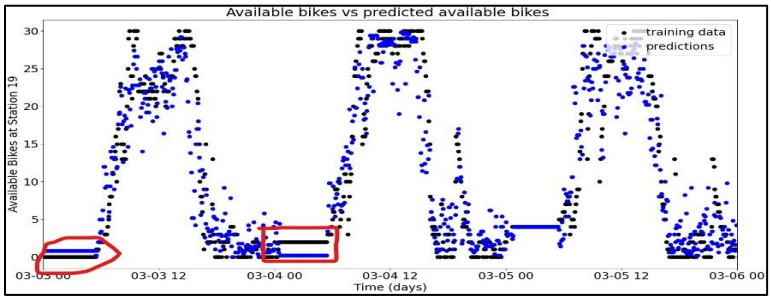


*Figure 6(a): **Run 5**, 60 minutes ahead predictions using **KNeighborsRegressor. Station 19***

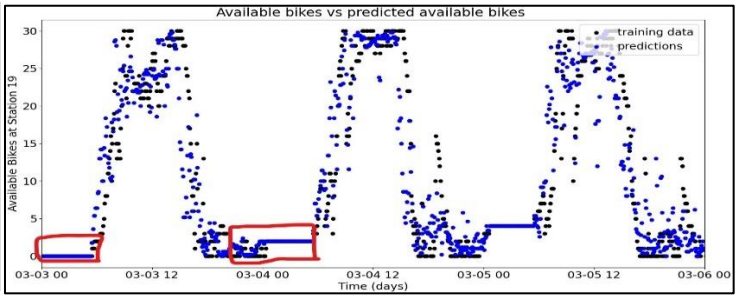


*Figure 6(b): **Run 7**, 60 minutes ahead predictions using **KNeighborsRegressor . Station 19***

By observing the prediction plots and the mean squared error scores for each set of runs and features(table1) for stations 10 and 19, I chose the features used in the models to be composed of the short-term features, daily, and weekly seasonality.

3. **Experiment 3:** The input features chosen from experiment 2 are fit using a ridge regression model, and the weights given to each feature are analysed. This experiment was performed for both stations 10 and 19. Following this, a plot is created that charts what each feature signifies and its weight.

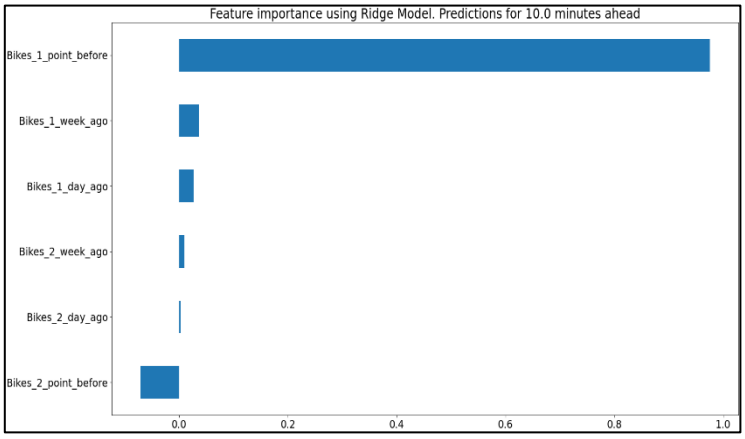


*Figure 7(a): Feature importance Ridge Model 10 minutes ahead preds. **Station 19***

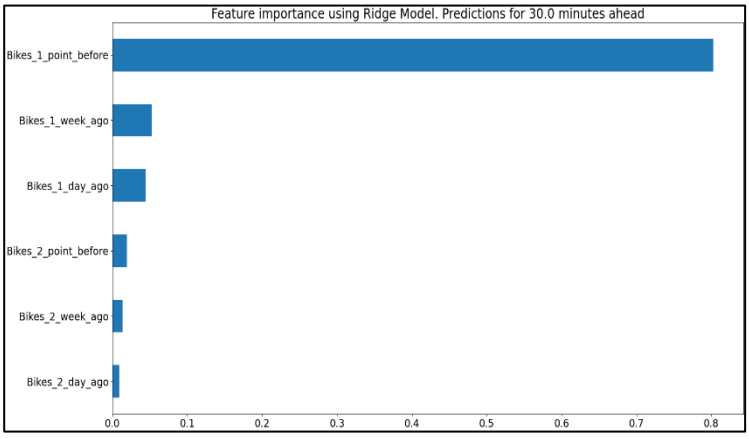


*Figure 7(b): Feature importance Ridge Model 30 minutes ahead preds. **Station 19***

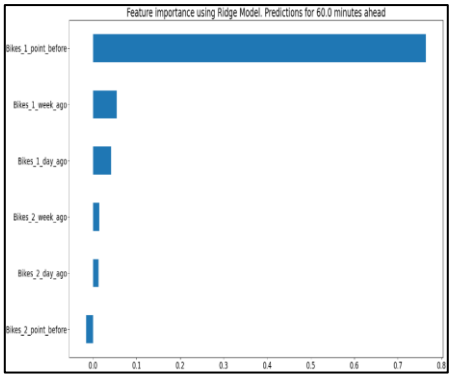


Figure 7(c): FI Ridge,60 minutes ahead preds. ***Station 19***

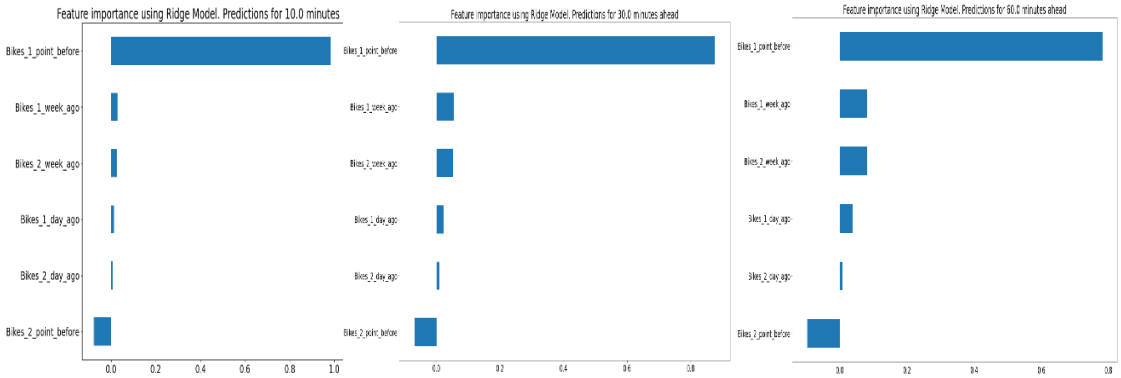


Figure 7(d,e,f): Feature importance Ridge Model for 10,30,60 minutes ahead preds. ***Station 11***

From the feature importance (FI) plots in figure 7(a,b,c) for station 19 and figure 7(d,e,f) for station 11, we can observe that for the 10,30 and 60 minutes ahead predictions, the number of available bikes measured at the previous point/timestamp is the most important. This means that the model utilises the short-term trends the most to predict the number of available bikes in the future. From the FI plots, we can also observe that the measurements from the previous one week and day are important but not as significant as those from the previous point. The measurements from 2 weeks and days ago have a small, allocated weight which means that the model is utilising these measurements and can extract some information from the daily and weekly seasonalities.

We can also observe that the number of available bikes 2 points before contributes slightly to making predictions using the ridge model. The number of available bikes 2 points before is assigned a very small negative weight in figures 7(a,c,d,e,f), whereas the number of available bikes 2 points before is assigned a small positive weight in figure 7(c) which signifies the FI for the set of input features used to prediction bike station occupancy 30 minutes into the future for station 10

By observing the feature importance plots in figures 7(a,b,c,d,e,f) in Experiment 3, I chose the input feature vector to contain all the features seen in figure 7. This is so because all the features observed in figure 7 contribute towards making more correct predictions, and this can also be observed in figures 5(b) & 6(b) when compared with figures 5(a) & 6(a).

Choosing the best version of ridge and kNN model by tuning and choosing model hyperparameters through k-Fold cross validation:

In the following experiments, I choose the metric to be mean squared error for model tuning using cross-validation. This is so because MSE helps us understand how close our predictions are to the actual expected output.

1. Selecting the maximum order of the polynomial to use and augment/ting the input features with polynomial features.
   To choose q, we scan across a range of values for q (1 to 5) with a fixed value of C (C=1.0) for both the stations. We then perform 10 fold cross-validation for each value of q and plot the distribution of the negative mean squared error and q value. Figure 8(a) visualises this plot for station 10 and 8(b) for station 19 for 10 minutes ahead predictions.

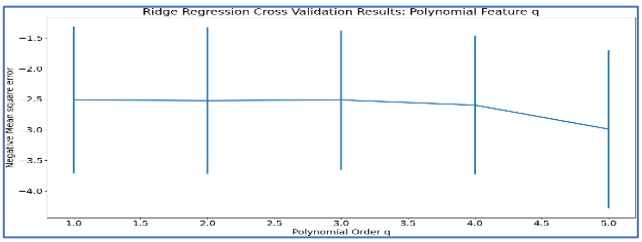


Figure 8 (a): Plotting q values vs neg MSE for 10 min ahead preds ridge model. Station 10

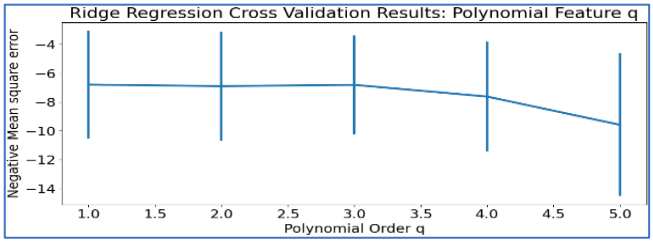


Figure 8 (b): Plotting q values vs neg MSE for 10 min ahead preds ridge model. Station 19

From figures 8(a) & 8(b), we choose the polynomial features value q = 1 since we attain a negative MSE closest to 0 on the plot, and q =1 would provide us with the simplest form of input features. If I decide to choose q>1, it would unnecessarily make the input features more complex.

I chose not to augment input features with polynomial features for a kNN regressor since kNN can capture non-linear patterns. We similarly choose the polynomial order value for the model used to predict 30 minutes and 60 minutes ahead into the future. Please refer to table 3 below to see the q values for these models

2. Selecting the weight C given to the penalty parameter in the ridge (L2) cost function.

To choose the value for the penalty parameter C, we scan across a range of values for C (0.00001 to 50) with chosen q value (q=1) for ridge. We then perform cross-validation for each value of C and plot the MSE and C value distribution. Figure 9(a) visualises this plot for station 10 and 9(b) for station 19

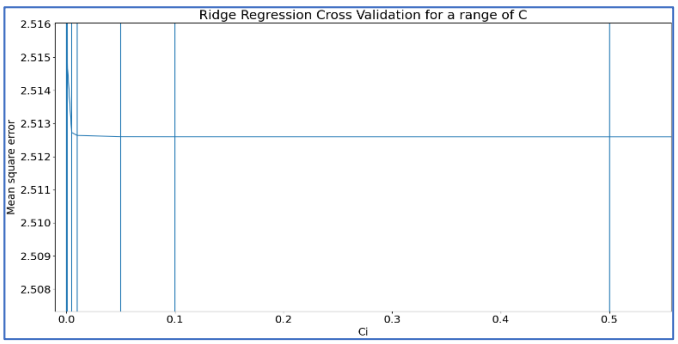


Figure 9(a): Plotting C values vs MSE for ridge model. Station 10

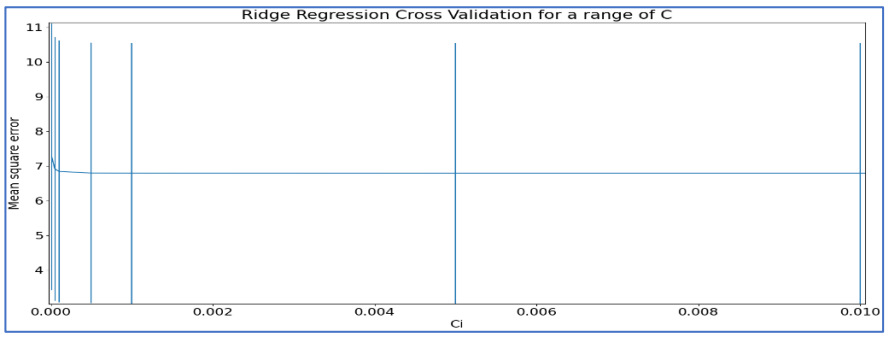


Figure 9(b): Plotting C values vs MSE for ridge model. Station 19

I chose a C value of 0.05 for station 10 and C=0.010 for station 19, since in Figure 9(a), at C = 0.05 and 9(b) at C=0.010, the MSE is the lowest. It would make sense to choose a value of C>=0.05 for station 10 and C>=0.010 for station 19 since the MSE is low and uniform, but I recommend a value of C=0.05 for station 10 and C=0.010 for station 19 so that we use the 'simplest model' available and avoid overfitting. We similarly choose the C value for the ridge model used to predict 30 minutes and 60 minutes ahead into the future. Please refer to table 3 below to see the q values for these models

3. Selecting the number of neighbours k in the kNN regression model known as KNeighborsRegressor in sklearn

To choose the value for k, which is the number of neighbours, we perform 10 fold cross-validation to tune and k. To choose the value for k, we scan across a range of values for k (1 to 100) and train the model with uniform weight distribution, which means that each neighbour data point has equal weight. We then perform cross-validation for each value of k and plot the MSE and k value distribution. Figure 10(a) visualises this plot for stations 10 and 10(b) for station 19. Based on the cross-validation data from Figure 11(a) and 11(b), a k value of 20 should be used to train a model using a kNN classifier such that it neither under nor overfits the data. I would choose a k value of 20 for training a kNN classifier since in Figure 10(a) & 10(b), at k =20, the MSE score is closest to 0. It would make sense to choose a value of k>=20 since the MSE score is close to 0, but I recommend a value of k=20 to use the 'simplest model' available and avoid overfitting. We similarly choose the k value for the kNN model used to predict 30 minutes and 60 minutes ahead into the future. Please refer to table 3 below to see the q values for these models

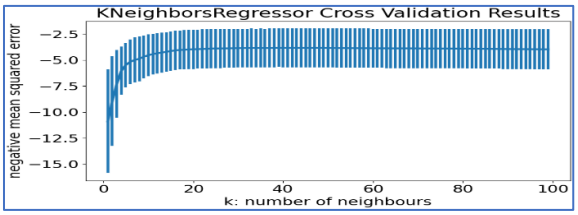


Figure 10(a): Plotting k values vs MSE for kNN model. Station 10

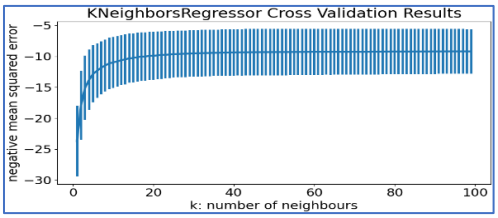


Figure 10(b): Plotting k values vs MSE for kNN model. Station 19

Table 3: Hyperparameter values for the ridge and kNN model used to predict 10, 30 & 60 minutes into the future

	10 minutes ahead predictions		30 minutes ahead predictions		60 minutes ahead predictions	
	Station 10	**Station 19**	**Station 10**	**Station 19**	**Station 10**	**Station 19**
Polynomial Order Value q	1	1	1	1	1	1
C value Ridge Regression Model	0.05	0.01	0.05	0.01	0.05	0.05
K value kNN Model	20	20	30	30	40	60

(d) Evaluation

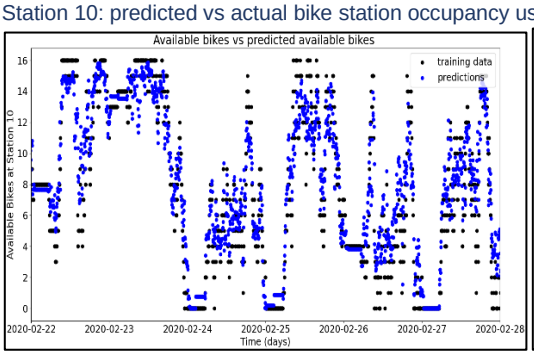
The evaluation section covers model assessment where we estimate our model's prediction error on test data, i.e., new & previously unseen data. We compare our models with each other and to the baseline estimator that predicts the number of bikes available to be the same as the previous day. We then discuss the comparison of the models with the help of predictions vs expected output plots and performance metrics. The models are evaluated using 2 performance metrics with explanations. The performance metrics used are the mean squared error and the R2 score.

Following the feature selection and model hyperparameter selection in section (c) for bike stations 10 and 19, we train a ridge regression and kNN model with the selected hyperparameters. These models are trained to predict bike station occupancy 10,30 & 60 minutes into the future. To train the models, the input features are split into training and test sets i.e hold-out data is used to train and test the model. The training data is 70 percent of the selected engineered features (refer to section (b) and (c)), whereas the test data is extracted from 30 percent of the selected engineered features. I choose the 70-30 split since it strikes a good balance to have adequate data to train the models and test them. See code below:

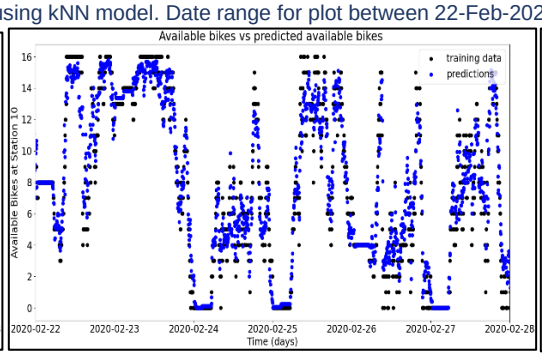
```
train_indices = 0.70 * df_features.shape[0]
X_train = df_features[: int(train_indices)].drop([f"bikes_avail_{q}_mins_ahead"], axis=1); y_train = df_features[: int(train_indices)];  y_train = y_train[[f"bikes_avail_{q}_mins_ahead"]]
X_test = df_features[int(train_indices) :].drop([f"bikes_avail_{q}_mins_ahead"], axis=1); y_test = df_features[int(train_indices) :];  y_test = y_test[[f"bikes_avail_{q}_mins_ahead"]]
```

<u>Figures 11 to 15 below visualise the predicted vs actual bike station occupancy for stations 10 & 19 for the ridge, kNN and baseline models predicting 10,30 & 60 minutes ahead. Table 3 below shows the performance metrics for the ridge, kNN and baseline models.</u>

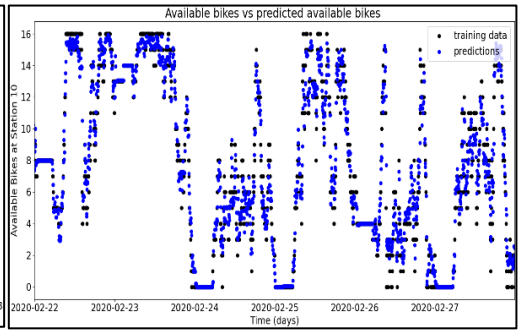
Station 10: predicted vs actual bike station occupancy using kNN model. Date range for plot between 22-Feb-2020 to 28-Feb-2020



11(a): Station 10, KNeighborRegressor, 10 min ahead preds

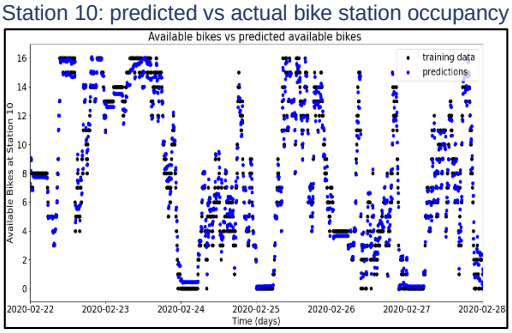


11(b): Station 10, KNeighborRegressor, 10 min ahead preds

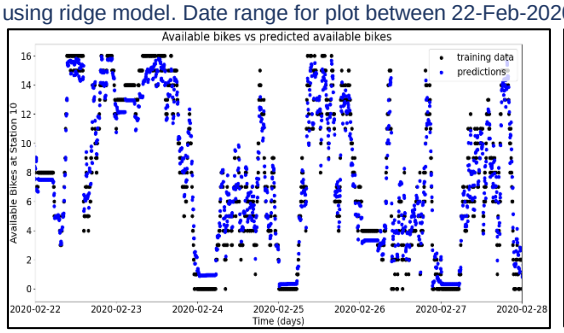


11(c): Station 10, KNeighborRegressor, 10 min ahead preds

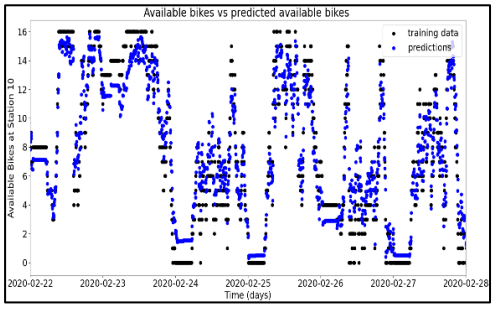
Station 10: predicted vs actual bike station occupancy using ridge model. Date range for plot between 22-Feb-2020 to 28-Feb-2020



12(a): Station 10, ridge, 10 min ahead preds

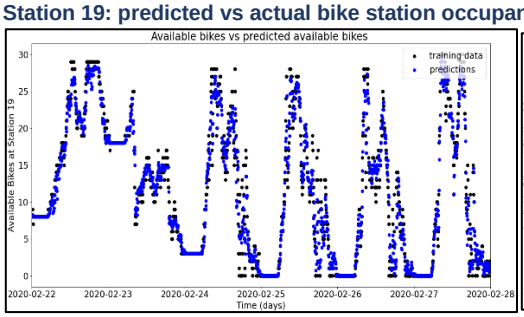


12(b): Station 10, ridge, 10 min ahead preds

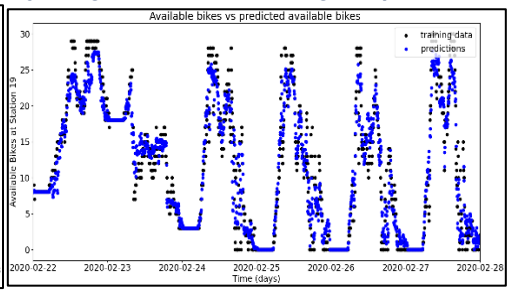


12(c): Station 10, ridge, 10 min ahead preds

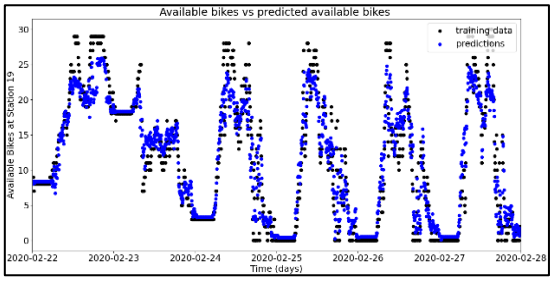
Station 19: predicted vs actual bike station occupancy using kNN model. Date range for plot between 22-Feb-2020 to 28-Feb-2020



13(a): Station 19, KNeighborRegressor, 10 min ahead preds

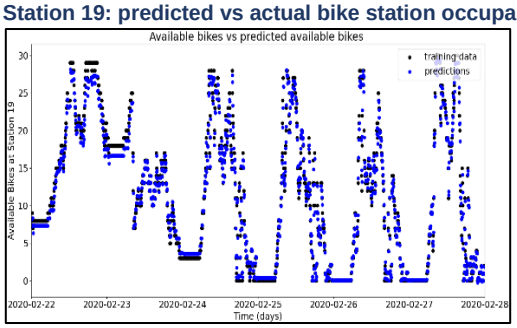


13(b): Station 19, KNeighborRegressor, 30 min ahead preds

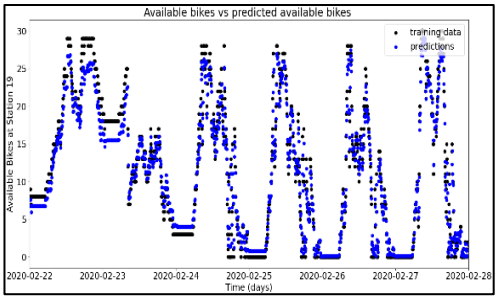


13(c): Station 19, KNeighborRegressor, 60 min ahead preds

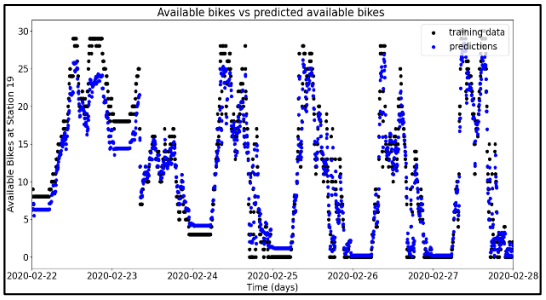
Station 19: predicted vs actual bike station occupancy using ridge model. Date range for plot between 22-Feb-2020 to 28-Feb-2020



14(a): Station 19, ridge, 10 min ahead preds

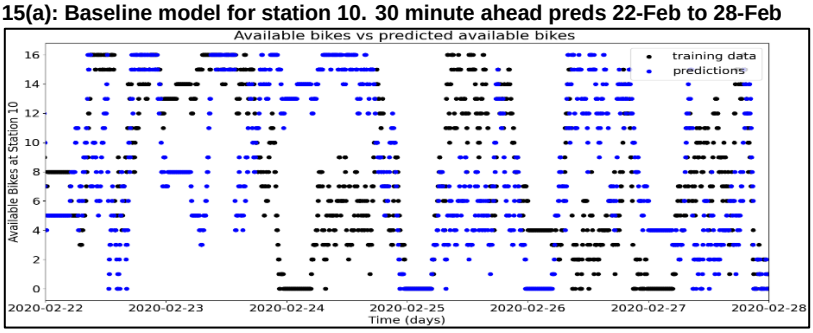


14(b): Station 19, ridge, 30 min ahead preds

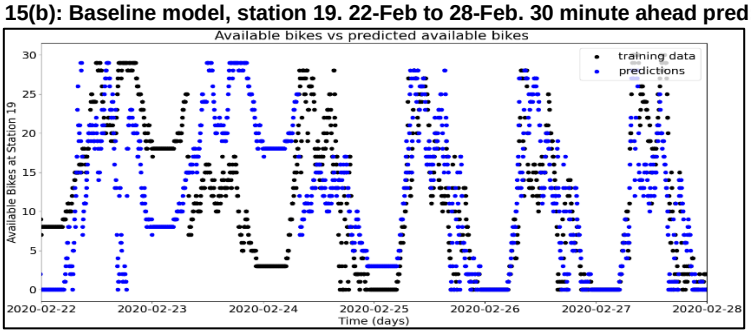


14(c): Station 19, ridge, 60 min ahead preds

15(a): Baseline model for station 10. 30 minute ahead preds 22-Feb to 28-Feb



15(b): Baseline model, station 19. 22-Feb to 28-Feb. 30 minute ahead pred:



Model Performance Metrics	10 minutes ahead predictions						30 minutes ahead predictions						60 minutes ahead predictions					
	Station 10			Station 19			Station 10			Station 19			Station 10			Station 19		
	Ridge	kNN	BL	Ridge	kNN	BL	Ridge	kNN	BL	Ridge	kNN	BL	Ridge	kNN	BL	Ridge	kNN	BL
mean squared error (MSE)	1.228	2.288	33.049	2.445	8.696	66.726	3.004	4.490	33.056	5.876	16.569	66.732	5.299	7.580	33.067	10.521	21.293	66.749
r2 score	0.916	0.844	-0.237	0.915	0.700	0.039	0.796	0.695	0.237	0.797	0.429	0.039	0.640	0.485	-0.238	0.638	0.267	0.039

Table 3

Ridge regression, kNN and baseline models are trained on the training data. These models are tested by making predictions on the test data and their performance metrics are stored in Table 3. All 3 models are used to predict bike station occupancy 10,30 and 60 minutes into the future for stations 10 & 19.

Baseline Model Working Methodology: The baseline model predicts that the number of bikes' x' minutes ahead into the future will be the same as the number of bikes' x' minutes ahead on the previous day. The plot for the baseline predictions can be observed in figures 15(a) & 15(b). The plots for the baseline model predicting 10,30 and 60 minutes into the future are all very similar, so I only show the plot for 30 minutes ahead predictions. Table 3 shows that the performance of the baseline model for predicting 10,30 & 60 minutes ahead is almost the same.

What each performance metric represents: I chose the performance metrics to be MSE and r2 because the 2 metrics help us get a more complete view of a models' performance in comparison to using only 1 metric. MSE helps us understand how close our predictions are to the actual expected output, the lower the MSE, the better are the model's predictions. The idea behind the r2 score is that it helps us understand how much better is our model doing than just predicting a constant. The r2 score helps us see how well a model fits the input feature vector. r2=1 when the model predicts perfectly, and r2 = 0 when the model's prediction is no better than predicting the mean value[1]. I did not choose RMSE as an evaluation metric since it is just the square root of the MSE. Both, the RMSE and MSE help us understand how close our predictions are to the actual output.

Discussion and Model Comparison: On training all the 3 tuned models (with the optimal C value, k value and polynomial order), we predict the number of available bikes 10,30 & 60 minutes ahead for the entire set of input features for both stations. The plots in figures 11 to 15 to show predicted vs actual bike station occupancy between 22-Feb-2020 to 28-Feb-2020. I chose this date range so we can clearly observe the differences between the predictions made by each model.

From figures 11,12,13,14 and table 3, we can observe that as we predict further into the future for both stations., the predictions to become less accurate for each model, the MSE increases and the r2 score decreases. We can observe that figures 11(a), 12(a), 13(a) & 14(a) have more accurate predictions when compared with figures 11(c), 12(c), 13(c) & 14(c) [Observe the predictions between 22-Feb and 23-Feb in the figures mentioned above to notice the decrease in accuracy of the predictions.]
This is because when we are predicting 10 minutes ahead into the future, there is a lot of overlap and a lot common between input features and the output measurements. The inputs and outputs are very correlated since we use the output i.e., the number of available bikes to model the feature vector. When we are

predicting closer into the future, the correlation between the inputs and outputs is higher than when we are predicting further into the future. In the case of predicting 10 minutes ahead into the future, we may have over-optimistic results since the inputs and outputs are highly correlated, but this is not the case when we are predicting 60 minutes in the future as the inputs and output points are less correlated.

Comparing the Ridge and KNN models through figures 11 & 12 for station 10, figures 13 & 14 for station 19 and table 3, we can observe that the Ridge regression model outperforms the kNN regression model. Ridge regression has more accurate predictions than KNN when predicting bike station occupancy 10, 30 and 60 minutes ahead for both stations. This can be observed by the plots showing predictions vs the actual outputs above. When comparing figures 11(a) with 12(a), 11(b) with 12(b) &, 11(c) with 12(c) between 22-Feb and 23-Feb, we can clearly observe that for station 10, the ridge regression model's predictions are closer to the actual expected output, and the ridge model fits the input feature vector better when compared to the kNN regressor. Ridge's better performance here is also reflected in table 3 where it constantly has a lower MSE and a higher r2 score than the kNN regressor. The ridge model performs better than kNN, but kNN's performance is comparable to the ridge regression model.
For station 19, we can observe through figures 13 & 14 and table 3 that the ridge regression model performs significantly better than the kNN regressor. When comparing figures 13(a) with 14(a), 13(b) with 14(b) &, 13(c) with 14(c) between 26-Feb and 28-Feb, we can clearly observe that the ridge regression model's predictions are closer to the actual expected output and the ridge model fits the input feature vector better when compared to the kNN regressor. Ridge's better performance is significantly reflected in table 3 where it constantly has a much lower MSE and a significantly higher r2 score than the kNN regressor. As we predict further into the future, the difference in the MSE and r2 score for ridge and knn grows, with kNN having a significant decrease in r2 scores. Lower r2 scores for kNN can be observed in table 3 for 60 minute ahead predictions which signify that the kNN model does not fit the feature vector well. The ridge model performs much better than the kNN model.

On comparing the ridge and kNN model with the baseline, we can clearly observe that the kNN and ridge regression models are more accurate in predicting bike station occupancy. In table 3, for both stations, the ridge and kNN models always have a much lower MSE when compared to the baseline, which means that our models are making better predictions than the baseline model. The ridge and kNN models also fit the feature vector better than the baseline model, and this can be observed from the r2 scores in table 3. On comparing figures 11(b) & 12(b) with 15(a) for station 10 and figures 13(b) & 14(b) with 15(b) for station 19, we can clearly see that the predictions of the kNN and ridge models are much more accurate than the baseline. Since the selected ridge and kNN models had better predictions and performance than the baseline model, it is worth spending the time and resources into training, fine-tuning and testing these models.

I believe that we are noticing overoptimistic results when predicting bike station occupancy 10 minutes in the future for station 10 and 19 since there are overlapping elements in the feature vector when predicting 10 minutes ahead due to the sliding window. To tackle this problem, I would set the training data points further apart for the feature vector for predicting bike station occupancy 10 minutes ahead into the future. I would reduce the overlap by constructing the input feature vector to not have any measurements of the previous 2 points if it does not drastically hinder the model's performance.

In conclusion, it is observed that the ridge regression model is the best model for predicting bike station occupancy 10, 30 and 60 minutes into the future for bike stations 10 and 19. The kNN regression model performed well in predicting bike station occupancy for station 10 but experienced a significant drop in prediction accuracy for station 19. Nonetheless, on comparing the kNN regressor with the baseline model that predicts the next point to be the same as the previous day, we can say that kNN beat the baseline and it is still a viable option, however, it is not the best model suited to our current problem and use case. Therefore, we can conclude that the fine tuned ridge regression model is the best fit for out problem and use case to predict bike station occupancy 10, 30 and 60 minutes in the future.

2.(i) What is a ROC curve. How can it be used to evaluate the performance of a classifier.

A receiver operating characteristic curve or ROC curve plots a classifier's true positive rate and false-positive rate. An ROC curve is formed by varying the decision threshold β or decision boundary of a classifier model between 0 and 1. We vary the value of the β between 0 and 1 since that is the range of values the confidence function (sigmoid) can take. As β is varied between 0 and 1, we calculate the training data's true and false positive rates. By varying the threshold β, we get a set of true positive TP and false-positive FP rates that are traced out on a ROC curve. An example of a ROC curve can be seen in the figure below. As an example, for a specific value of β, the true positive rate may be 0.75, and the false positive rate may be 0.25. An ideal classifier would be one where we have a threshold value that maximises TP rate (TP rate=100%) and minimises FP rate (FP rate = 0%) and this would be the top left corner in the roc curve. However, depending on the business case, our need for a specific true positive and false positive rate can change. We can also use the ROC curve to compare the performance of different classifiers, as observed in the figure. The 45-degree line in figure 16 is a random dummy classifier that choses uniformly at random between y = +1 and y = -1

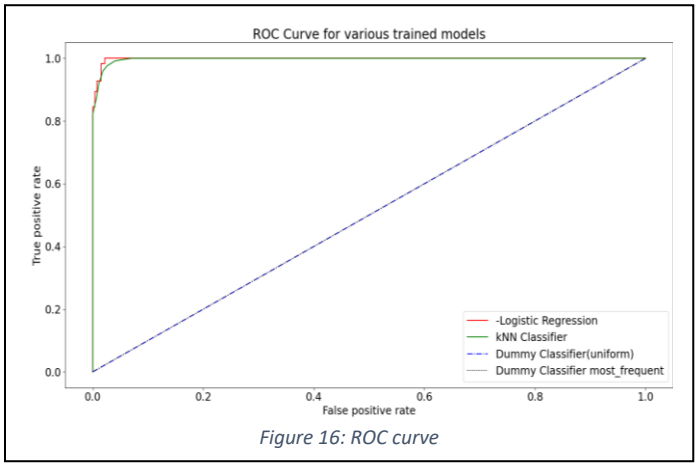


Figure 16: ROC curve

Please Note: The confidence function is a sigmoid or logistic function that maps $\theta^T x$ (from the linear model y = $sign(\theta^T x)$) to a confidence value between 0 and 1. The greater the confidence value is i.e., closer to 1, the more confident a classifier's prediction is

ROC curves can be used to evaluate classifier performance in 2 ways. To evaluate classifier performance, we can observe the ROC curve for a classifier. If the ROC curve of a classifier is close to the 45 degree line, it would mean that our classifier is performing well at classifying the data accurately. If the ROC curve of a classifier is below the 45 degree line, it would mean that our classifier is worse than a random classifier, which signifies that the classifier is terrible. Ideally, we would want a classifier that maximises TP rate while minimising FP rate at the same time (we want our classifier to operate on a threshold close to the top left corner of the ROC curve), but this would also depend on the business problem and how much we care about the false and true positives

Another way of evaluating model performance through the ROC curve is by looking at the area under the curve or AUC of the ROC curve. Looking at the AUC leads to loss of information but helps bring down the ROC curve for a classifier to one value. If the classifier we are analysing is a perfect classifier and is at the coordinates (0,1), then the AUC is 1. If our classifier is just random, the AUC would be half or 0.5; this point would be the AUC of the blue line in figure 16. We can evaluate the performance of a classifier by observing the AUC. If the classifier AUC is >0.5 and less than 1, it implies that our classifier is making better predictions than the baseline. If the AUC is under 0.5 (below the blue line in figure 16), we should flip the model. (i.e. if the model predicts +1, we should change it to -1 and vice versa).

2(ii) Give two examples of situations when a linear regression would give inaccurate predictions. Explain your reasoning. [5 marks]

Examples of situations where linear regression would give inaccurate predictions

● Example 1: Linear regression would give inaccurate predictions when the input data is of non-linear in shape. Linear regression assumes that the relationship between the dependent (output) and independent (input) variables is linear. If there is a non-linear relationship between the input and output variable, then the linear regression model will not fit the non-linear data well. Linear regression can be used on data that only fits a straight line plot. If there is a curved or quadratic relationship between the dependent (output) and independent (input) variables, then the linear regression model will fit the input data badly. For example, a linear regression model would fit the data in figure 17(a) well since the input and output variables have a linear relationship. However, a linear regression model would not fit the data in figure 17(b) well since the input and output variables have a non-linear relationship

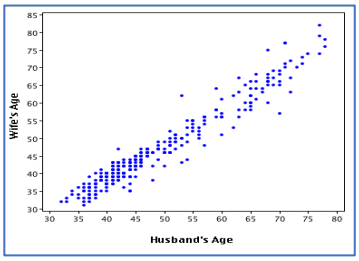


Figure 17(a): Linear relationship between x & y

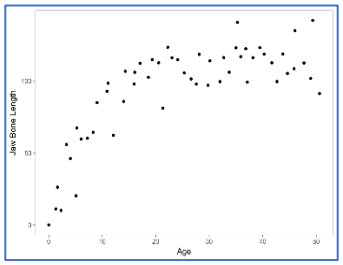


Figure 17(b): Nonlinear relationship between x & y

● Example 2: Linear regression would give inaccurate predictions when the input variables are highly correlated with each other. Linear regression assumes that independent variables are not highly correlated with each other. When independent variables are highly correlated, linear regression will have a problem identifying which input variables contribute well to predict the dependent variable and which input variables don't contribute well towards predicting the value of the dependent variable.
For example, we are trying to predict salaries using the age and experience of a person, we can observe that the age and experience are highly corelated. If we change the coefficient of one input variable (say age), it should also be accompanied by a shift in the coefficient of the dependent input variable (experience).
If the linear regression model wants to improve prediction accuracy by changing only one feature coefficient (say age), it would not be able to improve the models fit since it would also have to change the coefficient of the dependent input feature (experience)

2(iii) Discuss three pros/cons of an SVM classifier vs a neural net classifier. [5 marks]

1. The weights of neural networks are hard to interpret since neural networks work in a black box fashion as they have multiple hidden layers, nodes and different layers can have different activation functions. The parameters of SVM's are easier to interpret since it does not work in a black-box fashion. The model equation ($h\theta(x) = \text{sign}(\theta^T x)$), the cost function (hinge-loss) and how parameters are selected (select θ that minimise cost function) are clearly laid out and it is easier to interpret the parameters of SVM.

2. An advantage of neural networks over SVM's is that NN's can have more than 1 output, whereas SVM's in their most basic form can have only 1 output. Neural networks can perform multiclass classification, whereas SVM's cannot perform multiclass classification.

3. Neural networks can be slow to train since the cost function is non-convex in weights and there can be huge number of parameters that need to learned depending on the number of hidden layers and the nodes in each hidden layer.
   In contrast, SVM's are faster to train since the hinge loss function used in SVM's is convex. SVM's learn faster as well since they usually would have a lesser number of parameters when compared to a neural network with many hidden layers and nodes.

Please Note: I have assumed that we are discussing a non-kernalised SVM classifier above.

2(iv) Describe the operation of a convolutional layer in a conv Net. Give a small example to illustrate. [5 marks]

Operation of a convolutional layer in a convNet: A convolutional layer in a convNet takes the output of its previous layer as its input. Each node in a convolutional layer uses a kernel and convolves this kernel onto the input matrix to produce an output. The number of channels in each kernel would be the same as that of the input to that conv layer. When a kernel is convolved onto the input image or the output of the previous convolutional layer, we can apply padding =' same' to ensure that we pad the original input and then apply the kernel so that the output is the same as the input. Suppose we want the output image to be smaller than the input. In that case, we can also choose the padding to be 'valid'. For each convolutional layer, we can also choose to select the stride, which dictates how we move the kernel along when applying it to the input of the convolutional layer. In each convolutional layer, we can apply several filters to the input of the conv layer to dictate if we want a larger/smaller/same number of input channels. In each convolutional layer, the goal is to learn the output parameters and kernel weights. The output from the last convolutional layer is a matrix which is flattened into a vector x which is then fed to a linear model. We use training data to learn the kernel weights and output parameters in each convolutional layer so that we have the best set of input features. Having the best set of input features in the feature vector ensures that the model is better at classifying images.

To illustrate, let us use a small example demonstrating the operation of the first convolutional layer:
The input to the first convolutional layer is an input image of shape 5 X 5 X 1, where the height and width of the image is 4 X 4 and the image has only 1 channel (the image is formed from only 1 shade of pixels, grey-scale images). The shape of the kernel is 3X3, the padding used is 'valid' meaning that we apply the kernel directly to the input and our output image from the 1st conv layer is smaller than the input image. A stride of 1 is used and the activation function used is 'relu'. This kernel has 2 filters, and we stack the outputs together.
The first convolutional layer takes the input image in the form of a matrix and convolves the matrix and the kernel to produce the output as a matrix of shape 2 X 2 X 2. We have 2 output channels since we use 2 filters. This output matrix is then fed to the next convolutional layer. Through the 1st convolutional layer, the model has to learn 18 kernel weights and 8 output parameters for the output matrix which is performed using stochastic gradient descent. The same process is carried out for the other convolutional layers.

2(v) In k-fold cross-validation a dataset is resampled multiple times. What is the idea behind this resampling i.e. why does resampling allow us to evaluate the generalisation performance of a machine learning model? Give a small example to illustrate. [5 marks]

Resampling in k-fold cross-validation allows us to evaluate the generalisation performance of a machine learning model because we can get the average estimate of the cost function J(θ). Once we attain the average value of J(θ), we can use the model to estimate the average spread of values [1] which allows us to evaluate the generalisation performance of a machine learning model. Averaging a machine learning model over a spread of values also helps smooth out the noise experienced while training the model. This also contributes to evaluating the generalisation performance of a machine learning model.

A small example to illustrate the idea behind resampling: Suppose we choose k=5 and perform 5 fold cross-validation (CV). To perform 5 fold CV, we split the training dataset into 5 equally sized unique groups. We then hold out the 1st group as test data and use the rest as training data on which the model is trained. We then calculate the cost function J(θ) for the 1st group as we test our predictions on the 1st group. We repeat the same procedure for groups 2,3,4 & 5 and calculate the cost function J(θ) for each held-out group. On performing this process for 5 folds, we have 5 estimates of the cost function attain the average cost function J(θ) by averaging out all the k folds of the cost function J(θ). By averaging out the

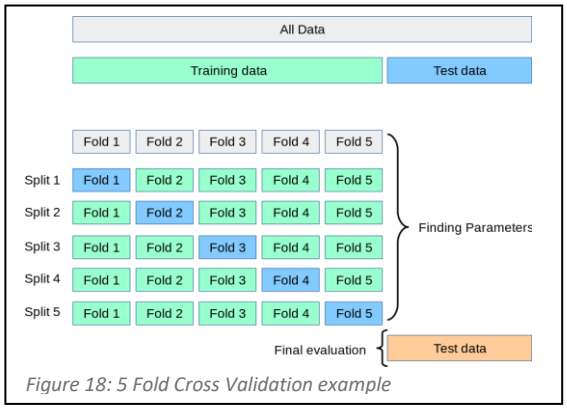


Figure 18: 5 Fold Cross Validation example

cost function, we get an idea of how consistently and robustly the models predictions are and if it generalises well or not. This example is visualised through figure 18.

Bibliography

[1] Leith, D 2021, Lecture Week 3: Evaluating Performance, lecture notes, Machine Learning CS7IS4-202122, Trinity College Dublin.

[2] Leith, D 2021, Lectures Week 1 to Week 12: All lectures, lecture notes, Machine Learning CS7IS4-202122, Trinity College Dublin.

[3] scikit-learn. 2022. *3.1. Cross-validation: evaluating estimator performance*. [online] Available at: <https://scikit-learn.org/stable/modules/cross_validation.html> [Accessed 4 January 2022].

[4] scikit-learn. 2022. *sklearn.neighbors.KNeighborsRegressor*. [online] Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html> [Accessed 4 January 2022].

Appendix:

Bike Prediction Code:

```
import math
from operator import index
from turtle import color

import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from matplotlib.style import available
from sklearn import metrics
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression, Ridge, RidgeCV
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import (
    GridSearchCV,
    KFold,
    TimeSeriesSplit,
    cross_val_score,
    train_test_split,
)
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.svm import SVR

DATASET_PATH = "Data/dublinbikes_20200101_20200401.csv"
SELECTED_STATIONS = [19, 10]

def plot_station_data(time_full_days, y_available_bikes, station_id):
    # plot number of available bikes at station id vs number of days
    plt.rc("font", size=18)
    plt.rcParams["figure.constrained_layout.use"] = True
    plt.figure(figsize=(8, 8), dpi=80)
```

```
    c = "red"
    if station_id == 10:
        c = "green"
    elif station_id == 19:
        c = "blue"
    # plt.plot(time_full_days, y_available_bikes, '-o', color=c)
    plt.scatter(time_full_days, y_available_bikes, color=c, marker=".")
    plt.xlabel("Time (Days)")
    plt.ylabel(f"Available Bikes at Station {station_id}")
    plt.title(f"Available bikes vs Days for bike station {station_id}")
    # plt.xlim(pd.to_datetime('2020-02-10'), pd.to_datetime('2020-02-19'))
    plt.show()


def plot_preds(time_full_days, y_available_bikes, time_preds_days, y_pred, station_id):
    plt.scatter(time_full_days, y_available_bikes, color="black")
    plt.scatter(time_preds_days, y_pred, color="blue")
    plt.xlabel("Time (days)")
    plt.ylabel(f"Available Bikes at Station {station_id}")
    plt.title(f"Available bikes vs predicted available bikes ")
    plt.legend(["training data", "predictions"], loc="upper right")
    # plt.xlim(pd.to_datetime('2020-03-03'), pd.to_datetime('2020-03-06'))
    plt.xlim(pd.to_datetime("2020-03-21"), pd.to_datetime("2020-03-30"))
    plt.show()


def regression_evaluation_metircs(ytest, ypred):
    mean_abs_err = metrics.mean_absolute_error(ytest, ypred)
    mean_sq_err = metrics.mean_squared_error(ytest, ypred)
    root_mean_sq_err = np.sqrt(mean_sq_err)
    median_abs_err = metrics.median_absolute_error(ytest, ypred)
    r2Score = metrics.r2_score(ytest, ypred)

    scores_dict = {
        "mean_sq_err": mean_sq_err,
        "root_mean_sq_err": root_mean_sq_err,
        "mean_abs_err": mean_abs_err,
        "r2Score": r2Score,
        "median_abs_err": median_abs_err,
    }

    print(f"Mean Squared Error: {mean_sq_err} ")
    print(f"Root Mean Squared Error: {root_mean_sq_err}")
    print(f"Mean_Absolute_Error: {mean_abs_err}")
```

```
    print(f"R2 Score: {r2Score}")
    print(f"Median Absolute Error: {median_abs_err}")

    return scores_dict


def feature_engineering(
    df_station,
    lag,
    q_step_size,
    time_sampling_interval_dt,
    short_term_features_flag,
    daily_features_flag,
    weekly_features_flag,
):
    available_bikes_df = df_station[["AVAILABLE BIKES"]]
    available_bikes_df = available_bikes_df.dropna()

    time_sampling_interval_dt_mins = time_sampling_interval_dt / 60

    # Setting outputs for step ahead predictions
    q = q_step_size * time_sampling_interval_dt_mins

    available_bikes_df.loc[:, f"bikes_avail_{q}_mins_ahead"] = available_bikes_df.loc[
        :, "AVAILABLE BIKES"
    ].shift(-q_step_size)

    if short_term_features_flag:
        for data_point in range(0, lag):
            available_bikes_df.loc[
                :, f"Bikes_{data_point+1}_point_before"
            ] = available_bikes_df.loc[:, "AVAILABLE BIKES"].shift(data_point + 1)

    # number of samples per day
    num_samples_per_day = math.floor(24 * 60 * 60 / time_sampling_interval_dt)
    if daily_features_flag:
        for day in range(0, lag):
            available_bikes_df.loc[
                :, f"Bikes_{day+1}_day_ago"
            ] = available_bikes_df.loc[:, "AVAILABLE BIKES"].shift(
                num_samples_per_day * (day + 1)
            )
```

```
    # number of samples per week
    num_samples_per_week = math.floor(7 * 24 * 60 * 60 / time_sampling_interval_dt)
    if weekly_features_flag:
        for week in range(0, lag):
            available_bikes_df.loc[
                :, f"Bikes_{week+1}_week_ago"
            ] = available_bikes_df.loc[:, "AVAILABLE BIKES"].shift(
                num_samples_per_week * (week + 1)
            )

    df_features = available_bikes_df.copy()
    df_features = df_features.drop(["AVAILABLE BIKES"], axis=1)

    df_features = df_features.dropna()
    available_bikes_df = available_bikes_df.dropna()

    # Setting up Hold Out train and test data for predicting available bikes 10 minutes ahead
    XX = df_features.drop([f"bikes_avail_{q}_mins_ahead"], axis=1)
    yy = df_features[[f"bikes_avail_{q}_mins_ahead"]]

    train_indices = 0.70 * df_features.shape[0]
    X_train = df_features[: int(train_indices)].drop(
        [f"bikes_avail_{q}_mins_ahead"], axis=1
    )
    y_train = df_features[: int(train_indices)]
    y_train = y_train[[f"bikes_avail_{q}_mins_ahead"]]

    X_test = df_features[int(train_indices) :].drop(
        [f"bikes_avail_{q}_mins_ahead"], axis=1
    )
    y_test = df_features[int(train_indices) :]
    y_test = y_test[[f"bikes_avail_{q}_mins_ahead"]]

    return XX, yy, X_train, y_train, X_test, y_test, df_features


def lagCrossValidation(
    df_station, time_sampling_interval_dt, df_total_station_data, station_id
):
    mean_error = []
    std_error = []
    lag_range = list(range(1, 6))
```

```
    test_model_ridge = Ridge(fit_intercept=False)
    test_model_kNNR = KNeighborsRegressor(n_neighbors=100)
    models = [test_model_ridge, test_model_kNNR]

    step_size = [2, 6, 12]
    for model in models:
        print(f"Model is {model}")
        for q_value in step_size:
            print(f"Step size is {q_value}")
            mean_error = []
            std_error = []
            for lag_value in lag_range:
                XX_CV, yy_CV, X_train, y_train, X_test, y_test, _ = feature_engineering(
                    df_station=df_station,
                    lag=lag_value,
                    q_step_size=q_value,
                    time_sampling_interval_dt=time_sampling_interval_dt,
                    short_term_features_flag=True,
                    daily_features_flag=True,
                    weekly_features_flag=True,
                )
                scores = cross_val_score(
                    model, XX_CV, yy_CV, cv=10, scoring="neg_mean_squared_error"
                )
                mean_error.append(np.array(scores).mean())
                std_error.append(np.array(scores).std())
                # model.fit(X_train, y_train)
                # plot_preds(df_total_station_data.index, df_total_station_data['AVAILABLE BIKES'], XX_CV.index,  model.predict(XX_CV), station_id)
            plt.rc("font", size=18)
            plt.rcParams["figure.constrained_layout.use"] = True
            plt.errorbar(lag_range, mean_error, yerr=std_error, linewidth=3)
            plt.xlabel("lag")
            plt.ylabel("negative mean squared error")
            plt.title(
                f"Lag Cross Validation Results,{q_value*(time_sampling_interval_dt/60)} minutes ahead Preidctions. Model is {model}"
            )
            plt.show()

def featureImportance(
    df_station, time_sampling_interval_dt, lag_value, df_total_station_data, station_id
):
    test_model_ridge = Ridge(fit_intercept=False)
```

```
step_size = [2, 6, 12]
for q_value in step_size:
    print(f"Step size is {q_value}")
    XX_CV, yy_CV, X_train, y_train, X_test, y_test, _ = feature_engineering(
        df_station=df_station,
        lag=lag_value,
        q_step_size=q_value,
        time_sampling_interval_dt=time_sampling_interval_dt,
        short_term_features_flag=True,
        daily_features_flag=True,
        weekly_features_flag=True,
    )

    test_model_ridge = RidgeCV().fit(X_train, y_train)
    # test_model_kNN = KNeighborsRegressor().fit(X_train, y_train)
    print(
        f"Best alpha value using sklearn Ridge Cross Validation: {test_model_ridge.alpha_}"
    )
    print(
        f"Best negative mse score using sklearn Ridge Cross Validation: {mean_squared_error(yy_CV, test_model_ridge.predict(XX_CV) )}"
    )

    weights = test_model_ridge.coef_.reshape(lag_value * 3)
    weights_with_labels = pd.Series(weights, index=X_train.columns.to_numpy())
    imp_weights_with_labels = weights_with_labels.sort_values()

    # plotting feature importance
    plt.rc("font", size=18)
    plt.rcParams["figure.constrained_layout.use"] = True
    plt.figure(figsize=(8, 8), dpi=80)
    imp_weights_with_labels.plot(kind="barh")
    plt.title(
        f"Feature importance using Ridge Model. Predictions for {q_value*time_sampling_interval_dt/60} minutes ahead"
    )
    plt.show()
    plot_preds(
        df_total_station_data.index,
        df_total_station_data["AVAILABLE BIKES"],
        XX_CV.index,
        test_model_ridge.predict(XX_CV),
        station_id,
    )
```

```
        # plot_preds(df_total_station_data.index, df_total_station_data['AVAILABLE BIKES'], XX_CV.index,  test_model_kNN.predict(XX_CV), station_id)


def PolynomialOrderCrossValidation(XX_ridge, yy_ridge, XX_kNR, yy_kNR):
    mean_error_ridge = []
    std_error_ridge = []
    mean_error_kNNR = []
    std_error_kNNR = []
    scaler = StandardScaler()
    XX_scaled_ridge = scaler.fit_transform(XX_ridge)
    # XX_scaled_kNR = scaler.fit_transform(XX_kNR)
    q_range = [1, 2, 3, 4, 5]
    for q in q_range:
        # Ridge
        Xpoly_ridge = PolynomialFeatures(q).fit_transform(XX_scaled_ridge)
        model_ridge = Ridge()
        scores_ridge = cross_val_score(
            model_ridge, Xpoly_ridge, yy_ridge, cv=10, scoring="neg_mean_squared_error"
        )
        mean_error_ridge.append(np.array(scores_ridge).mean())
        std_error_ridge.append(np.array(scores_ridge).std())

        # KNeighborsRegressor
        # Xpoly_kNR = PolynomialFeatures(q).fit_transform(XX_scaled_kNR)
        # model_kNNR = KNeighborsRegressor()
        # scores_kNNR = cross_val_score(
        #     model_kNNR, Xpoly_kNR, yy_kNR, cv=10, scoring="neg_mean_squared_error"
        # )
        # mean_error_kNNR.append(np.array(scores_kNNR).mean())
        # std_error_kNNR.append(np.array(scores_kNNR).std())

    plt.rc("font", size=18)
    plt.rcParams["figure.constrained_layout.use"] = True
    plt.errorbar(q_range, mean_error_ridge, yerr=std_error_ridge, linewidth=3)
    plt.xlabel("Polynomial Order q")
    plt.ylabel("Negative Mean square error")
    plt.title("Ridge Regression Cross Validation Results: Polynomial Feature q")
    plt.show()

    # plt.rc("font", size=18)
    # plt.rcParams["figure.constrained_layout.use"] = True
    # plt.errorbar(q_range, mean_error_kNNR, yerr=std_error_kNNR, linewidth=3)
    # plt.xlabel("Polynomial Order q")
```

```
    # plt.ylabel("Negative Mean square error")
    # plt.title("KNeighborsRegressor Cross Validation Results: Polynomial Feature q")
    # plt.show()


def RidgeAlphaValueCrossValidation(XX, yy):
    mean_error = []
    std_error = []
    C_range = [
        0.00001,
        0.00005,
        0.0001,
        0.0005,
        0.001,
        0.005,
        0.01,
        0.05,
        0.1,
        0.5,
        1,
        5,
        10,
        50,
        100,
        500,
    ]
    for C in C_range:
        ridge_model = Ridge(alpha=1 / (2 * C))
        scores = cross_val_score(
            ridge_model, XX, yy, cv=10, scoring="neg_mean_squared_error"
        )
        mean_error.append(np.array(scores).mean())
        std_error.append(np.array(scores).std())
    plt.rc("font", size=18)
    plt.rcParams["figure.constrained_layout.use"] = True
    plt.errorbar(C_range, mean_error, yerr=std_error)
    plt.xlabel("Ci")
    plt.ylabel("Negative Mean square error")
    plt.title("Ridge Regression Cross Validation for a range of C")
    plt.show()

def RidgeAlphaValueCrossValidation_method1(XX, yy):
    mean_error = []
```

```
std_error = []
XX_local = XX.copy()
yy_local = yy.copy()
XX_local = XX_local.reset_index(drop=True)
yy_local = yy_local.reset_index(drop=True)
XX_local.index = XX_local.index.astype(int, copy=False)
yy_local.index = yy_local.index.astype(int, copy=False)
C_range = [
    0.00001,
    0.00005,
    0.0001,
    0.0005,
    0.001,
    0.005,
    0.01,
    0.05,
    0.1,
    0.5,
    1,
    5,
    10,
    50,
]
for C in C_range:
    ridge_model = Ridge(alpha=1 / (2 * C))
    temp = []
    kf = KFold(n_splits=10)
    for train, test in kf.split(XX_local):
        ridge_model = Ridge(alpha=1 / (2 * C)).fit(
            XX_local.iloc[train], yy_local.iloc[train]
        )
        ypred = ridge_model.predict(XX_local.iloc[test])
        temp.append(mean_squared_error(yy_local.iloc[test], ypred))
    mean_error.append(np.array(temp).mean())
    std_error.append(np.array(temp).std())
plt.rc("font", size=18)
plt.rcParams["figure.constrained_layout.use"] = True
plt.errorbar(C_range, mean_error, yerr=std_error)
plt.xlabel("Ci")
plt.ylabel("Mean square error")
plt.title("Ridge Regression Cross Validation for a range of C")
# plt.xlim((0, 200))
```

```
    plt.show()


def KNeighborsRegressor_k_value_CV(XX, yy, num_neigbor_range):
    mean_error = []
    std_error = []
    num_neighbours = list(range(1, num_neigbor_range))
    for k in num_neighbours:
        model = KNeighborsRegressor(n_neighbors=k, weights="uniform")
        scores = cross_val_score(model, XX, yy, cv=10, scoring="neg_mean_squared_error")
        mean_error.append(np.array(scores).mean())
        std_error.append(np.array(scores).std())
    plt.rc("font", size=18)
    plt.rcParams["figure.constrained_layout.use"] = True
    plt.errorbar(num_neighbours, mean_error, yerr=std_error, linewidth=3)
    plt.xlabel("k: number of neighbours")
    plt.ylabel("negative mean squared error")
    plt.title("KNeighborsRegressor Cross Validation Results")
    plt.show()


def ridgeRegression(X_train, y_train, X_test, y_test, C_value):
    ridgeRegression_model = Ridge(alpha=1 / (2 * C_value), fit_intercept=False)
    ridgeRegression_model.fit(X_train, y_train)
    print("Ridge Regression Model Parameters")
    print(ridgeRegression_model.intercept_, ridgeRegression_model.coef_)
    ypred = ridgeRegression_model.predict(X_test)
    ridge_metrics_scores = regression_evaluation_metircs(y_test, ypred)
    return ridgeRegression_model, ridge_metrics_scores


def kNearestNeighborsRegression(X_train, y_train, X_test, y_test, num_neighbors_k):
    kNR_model = KNeighborsRegressor(n_neighbors=num_neighbors_k, weights="uniform")
    kNR_model.fit(X_train, y_train)
    ypred = kNR_model.predict(X_test)
    ridge_metrics_scores = regression_evaluation_metircs(y_test, ypred)
    return kNR_model, ridge_metrics_scores


def baselineModel(yy, station_id, pred_point_value):
    yy_baseline_true = yy.copy()
    yy_baseline_true = yy_baseline_true.drop(
        yy_baseline_true.index[range(pred_point_value)]
    )


    yy_baseline_pred = yy.copy()
```

```
    yy_baseline_pred.loc[:, f"pred_1_day_before bikes"] = yy_baseline_pred.loc[
        :, yy_baseline_pred.columns[0]
    ].shift(pred_point_value)
    yy_baseline_pred = yy_baseline_pred[yy_baseline_pred.columns[1]]
    yy_baseline_pred = yy_baseline_pred.dropna()

    baseline_metrics_scores = regression_evaluation_metircs(
        yy_baseline_true, yy_baseline_pred
    )

    print("Plotting Baseline Regression Predictions")
    plot_preds(
        yy_baseline_true.index,
        yy_baseline_true[yy_baseline_true.columns[0]],
        yy_baseline_pred.index,
        yy_baseline_pred,
        station_id,
    )

def exam_2021(df_station, station_id):
    start_date = pd.to_datetime("29-01-2020", format="%d-%m-%Y")
    df_station = df_station[df_station.TIME > start_date]

    # Finding the time interval between each measurement
    time_full_seconds = (
        pd.array(pd.DatetimeIndex(df_station.iloc[:, 1]).astype(np.int64)) / 1000000000
    )
    time_full_seconds = time_full_seconds.to_numpy()
    time_sampling_interval_dt = time_full_seconds[1] - time_full_seconds[0]
    print(
        f"data sampling interval is {time_sampling_interval_dt} secs or {time_sampling_interval_dt/60} minutes"
    )

    # Extracting all data for available bikes into df_total_station_data
    df_station = df_station.set_index("TIME")
    df_total_station_data = df_station[["AVAILABLE BIKES"]]
    df_total_station_data = df_total_station_data.dropna()

    # Plotting avaliable bikes vs time of our dataset
    plot_station_data(
        df_total_station_data.index,
        df_total_station_data["AVAILABLE BIKES"],
```

```
    station_id,
)

# Setting the step sizes
q_step_size_10_min_ahead_preds = 2
q_step_size_30_min_ahead_preds = 6
q_step_size_1_hr_ahead_preds = 12
q_step_size_list = [
    q_step_size_10_min_ahead_preds,
    q_step_size_30_min_ahead_preds,
    q_step_size_1_hr_ahead_preds,
]

# Performing cross-validation to select the number of prior points, days and weeks to consider.
lagCrossValidation(
    df_station, time_sampling_interval_dt, df_total_station_data, station_id
)

# lag_value_ridge is 4 and lag_value_kNNR is 2
lag_value_ridge = int(
    input(
        "Please choose the desired 'lag' value for the Ridge Regression model:    "
    )
)
lag_value_kNNR = int(
    input(
        "Please choose the desired 'lag' value for the KNeighborsRegressor model:    "
    )
)

# Feature Importance for Ridge Regression
featureImportance(
    df_station,
    time_sampling_interval_dt,
    lag_value_ridge,
    df_total_station_data,
    station_id,
)

# Calculating features for step ahead predictions and extracting these features into seperate dataframes
(
    XX_ridge,
```

```
    yy_ridge,
    X_train_ridge,
    y_train_ridge,
    X_test_ridge,
    y_test_ridge,
    df_features_2_step_ahead_ridge,
) = feature_engineering(
    df_station=df_station,
    lag=lag_value_ridge,
    q_step_size=12,
    time_sampling_interval_dt=time_sampling_interval_dt,
    short_term_features_flag=True,
    daily_features_flag=True,
    weekly_features_flag=True,
)

(
    XX_kNR,
    yy_kNR,
    X_train_kNR,
    y_train_kNR,
    X_test_kNR,
    y_test_kNR,
    df_features_2_step_ahead_kNR,
) = feature_engineering(
    df_station=df_station,
    lag=lag_value_kNNR,
    q_step_size=12,
    time_sampling_interval_dt=time_sampling_interval_dt,
    short_term_features_flag=True,
    daily_features_flag=True,
    weekly_features_flag=True,
)

# Cross Validation
PolynomialOrderCrossValidation(XX_ridge, yy_ridge, XX_kNR, yy_kNR)
polynomial_order_ridge = int(
    input("Please enter the polynomial order 'q' value for Ridge Regression :    ")
)
# polynomial_order_kNR = int(input("Please enter the polynomial order 'q' value for KNeighborsRegressor :    "))

XX_poly_ridge = XX_ridge
```

```
XX_poly_kNR = XX_kNR
if polynomial_order_ridge > 1:
    XX_poly_ridge = PolynomialFeatures(polynomial_order_ridge).fit_transform(
        XX_poly_ridge
    )
    X_train_ridge = PolynomialFeatures(polynomial_order_ridge).fit_transform(
        X_train_ridge
    )
    X_test_ridge = PolynomialFeatures(polynomial_order_ridge).fit_transform(
        X_test_ridge
    )

# if polynomial_order_kNR > 1:
#     XX_poly_kNR = PolynomialFeatures(polynomial_order_kNR).fit_transform(
#         XX_poly_kNR
#     )
#     X_train_kNR = PolynomialFeatures(polynomial_order_ridge).fit_transform(
#         X_train_kNR
#     )
#     X_test_kNR = PolynomialFeatures(polynomial_order_ridge).fit_transform(
#         X_test_kNR
#     )

# RidgeAlphaValueCrossValidation(XX_poly_ridge, yy_ridge)
RidgeAlphaValueCrossValidation_method1(XX_poly_ridge, yy_ridge)
C_value_ridge = float(
    input("Please choose the desired 'C' value for the Ridge Regression model:    ")
)

KNeighborsRegressor_k_value_CV(XX_poly_ridge, yy_ridge, 100)
k_value = int(
    input(
        "Please enter the number of neighnours 'k' value for the KNeighborsRegressor model:    "
    )
)

# ----------------------------------------Ridge Regression----------------------------------------
model_ridgeReg, scores_ridgeReg = ridgeRegression(
    X_train_ridge, y_train_ridge, X_test_ridge, y_test_ridge, C_value_ridge
)
ypred_full_ridge = model_ridgeReg.predict(XX_poly_ridge)
ypred_test_ridge = model_ridgeReg.predict(X_test_ridge)
```

```
    print("Plotting Ridge Regression Predictions")
    plot_preds(
        df_total_station_data.index,
        df_total_station_data["AVAILABLE BIKES"],
        XX_poly_ridge.index,
        ypred_full_ridge,
        # X_test_ridge.index,
        # ypred_test_ridge,
        station_id,
    )
    print(scores_ridgeReg)

    # ----------------------------------------KNeigborsRegressor-------------------------------------
    model_kNR, scores_kNR = kNearestNeighborsRegression(
        X_train_kNR, y_train_kNR, X_test_kNR, y_test_kNR, k_value
    )
    ypred_full_kNR = model_kNR.predict(XX_poly_kNR)

    print("Plotting Ridge Regression Predictions")
    plot_preds(
        df_total_station_data.index,
        df_total_station_data["AVAILABLE BIKES"],
        XX_poly_kNR.index,
        ypred_full_kNR,
        station_id,
    )
    print(scores_kNR)

    # ------------------------------------------------Baseline-------------------------------------------------
    baselineModel(yy_ridge, station_id, 288)

def main():
    df_dublin_bikes = pd.read_csv(DATASET_PATH)
    df_dublin_bikes["TIME"] = pd.to_datetime(df_dublin_bikes["TIME"])
    df_dublin_bikes.rename(columns={"STATION ID": "STATION_ID"}, inplace=True)
    for station_id in SELECTED_STATIONS:
        df_station = df_dublin_bikes.loc[df_dublin_bikes["STATION_ID"] == station_id]
        exam_2021(df_station, station_id)

    print("End of Final Assignment")
```

```
if __name__ == "__main__":
    main()
```