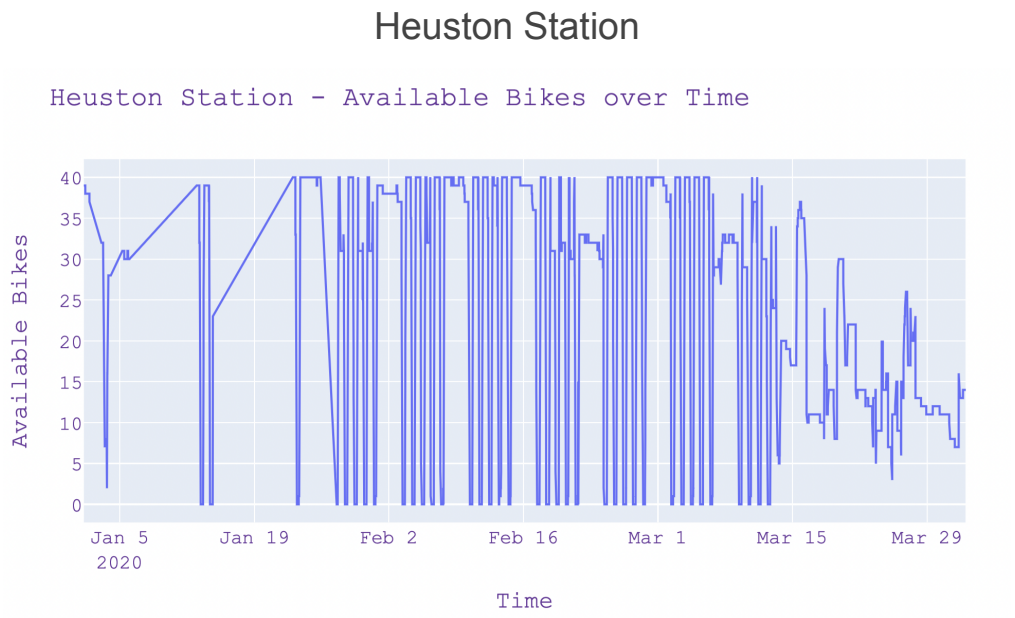
Bike Availability Report

Introduction

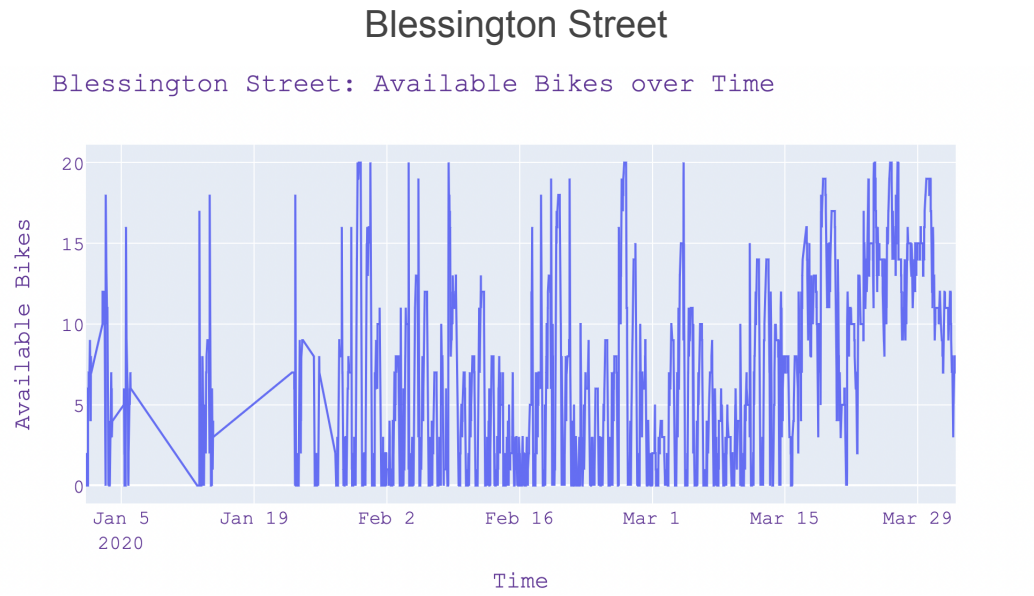
The stations I chose to analyse are Heuston Station Central and Blessington Street.
I chose Heuston Station Central because it is located in an isolated area where the only usage will come from commuters coming to and from Heuston Station. I expect this unique situation will reflect in its usage over the first quarter of 2020.
Blessington Street is an interesting challenge because it's location is in a residential area which makes predicting patterns of usage harder than the commuter-centric Heuston Station.

Understanding the Datasets

A Cursory Look



Above is the graph of bike availability over time at Heuston Station. January and the second half of March immediately stands out as an outlier. Decreased bike usage in January could be attributed to the unfavourable weather conditions, while March's behaviour is not so easily rationalised.



The above visualisation has similar outliers in January but not in March. It also suggests that Blessington Street has no apparent pattern to it unlike Heuston Station.

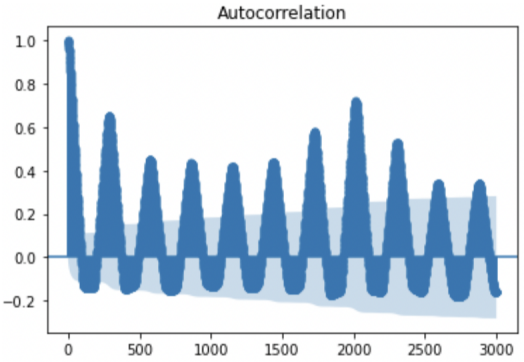
Feature Engineering

ACF Plot

An ACF plot describes how well a given sample in the time series correlates to past samples, or lags. Noting which samples correlate with statistical significance provides insight on what samples should be included when crafting our models' feature vector.

Heuston Station

The following is the ACF plot of Heuston Station's bike availability uses 3000 lags, which equates to around 10 days.



This graph clearly shows high correlation between the immediate past and the upcoming value. Besides that, other lags with the highest correlation are 100 lags (8 hours) ago, 300 lags (1 day) ago, 1750 lags (6 days) ago, 2000 lags (1 week) ago, and 2250 lags (8 days) ago.

This results in the following feature vector:
[Current, 10 minutes ago, 8 hours ago, 1 day ago, 6 days ago, 7 days ago, 8 days ago]
And the Ridge model gave the following weighting:

```
In [121]: print(model.coef_)

          [[ 1.40317461 -0.422213   -0.00685433  0.00818751  0.00239942  0.01001927
            -0.00296969]]
```

This model resulted RMSLE scores for 10 minute, 30 minute, and 60 minute predictions:

0.034301692373866103
0.343732351261730945
0.71868018563628303

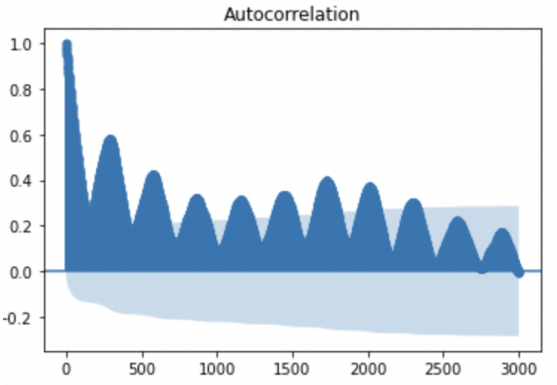
I decided to prune some low-weighted features off and include only the 5 most weighted features. This resulted in the following feature vector:

[Current, 10 minutes ago, 8 hours ago, 1 day ago, 7 days ago]
This new feature vector produced a model with the following RMSLE for 10min, 30min, and 60min predictions.

[0.026927460589364755]
[0.14131896201220145]
[0.5066254256879464]

This is an improvement over the previous feature vector and became my feature vector for both Ridge Regression and kNN.

Blessington Street



Unlike Heuston Station, the Blessington Street's ACF graph shows that no values have a negative correlation with future values. There is also not such a strong correlation between values in the distant past such as one week ago compared to Heuston Station.
The lags with the highest correlation are 1 lags (5 minutes) ago, 250 lags (1 day) ago, 575 lags (2 days) ago, 1750 lags (6 days) ago and 2000 lags (1 week) ago.

Fitting a model with this feature vector
[Current, 10 minutes ago, 1 day ago, 6 days ago, 7 days ago]

produced a model with RMSLE for 10min, 30 min, 60min predictions of:
[0.13620136988220005]
[0.42538673612077105]
[1.6649270359024613]

And parameter weighting of:
[1.11023045, -0.12769062, 0.01353657, -0.00160685, 0, 0.00130112]

The lag from 6 days before the target value is receiving a weight of 0 so it is an obvious candidate to cut from the feature vector.

Methodology

Ridge Regression

Ridge regression is a suitable method for predicting values in a time series because it is especially proficient in dealing with multicollinearity in the data, which is present in these time series'.
The cost function used in Ridge regression is

$$\mathbf{Min(||Y - X(theta)||^2 + \lambda||theta||^2)}$$

Where lambda is the penalty term which decreases the magnitude of the coefficients as lamda increases. This shrinkage prevents multicollinearity and reduces the complexity of the model.

Hyperparameter Tuning

Ridge Regression uses regularisation to create variance in its estimates in order to avoid overfitting the model to the training data.
I used the forward chaining to tune the Ridge Regression's alpha parameter, which affects the model's regularisation strength.
Forward chaining is used instead of k-fold for cross validation because order is important in time series.
So instead of doing

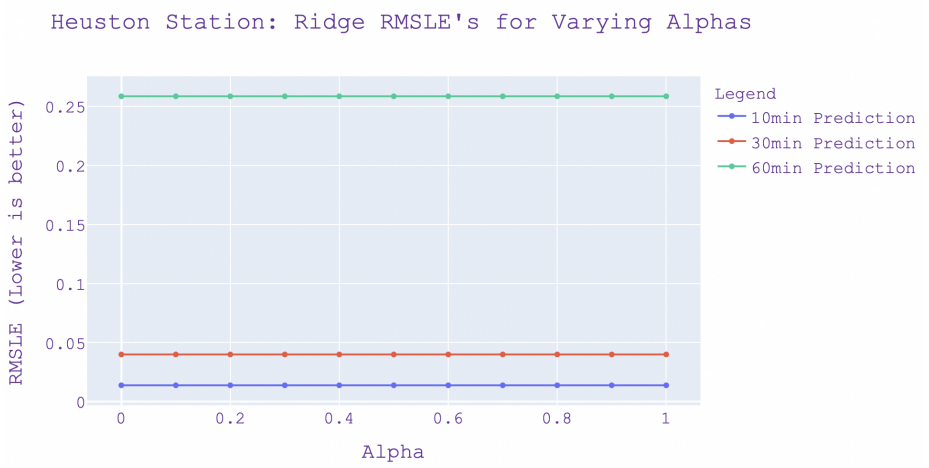
- fold 1 : training [1 2 3], test [4]
- fold 2 : training [1 2 4], test [3]
- fold 3 : training [1 3 4], test [2]
- fold 4 : training [2 3 4], test [1]

I use forward chaining which looks like:

- fold 1 : training [1], test [2]
- fold 2 : training [1 2], test [3]
- fold 3 : training [1 2 3], test [4]
- fold 4 : training [1 2 3 4], test [5]

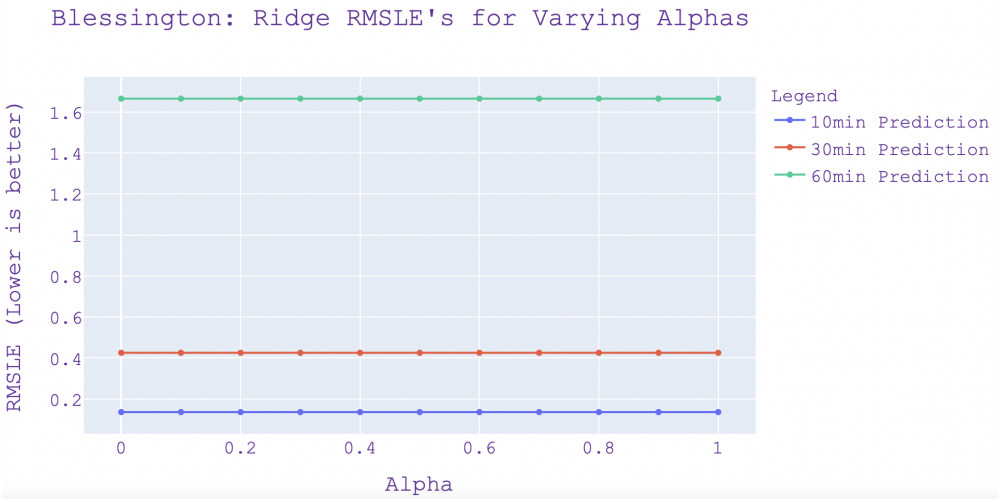
At each fold I incremented through an array of 20 alpha values ranging from 0 to 2 and determined which alpha value minimised the RMSLE

Heuston Station



As you can see above, there is no change in RMSLE as alpha changes. This could be due to the fact that the model's predictions are converted to integers so the variance enforced by the alpha value is rounded out at the end.
With this in mind I decided to use alpha = 1.0 for the Heuston Station dataset to reduce model complexity.

Blessington Street



There is no change in the model's RMSLE as the alpha value increases for Blessington Street, which I suspect is due to the same reason discussed above.

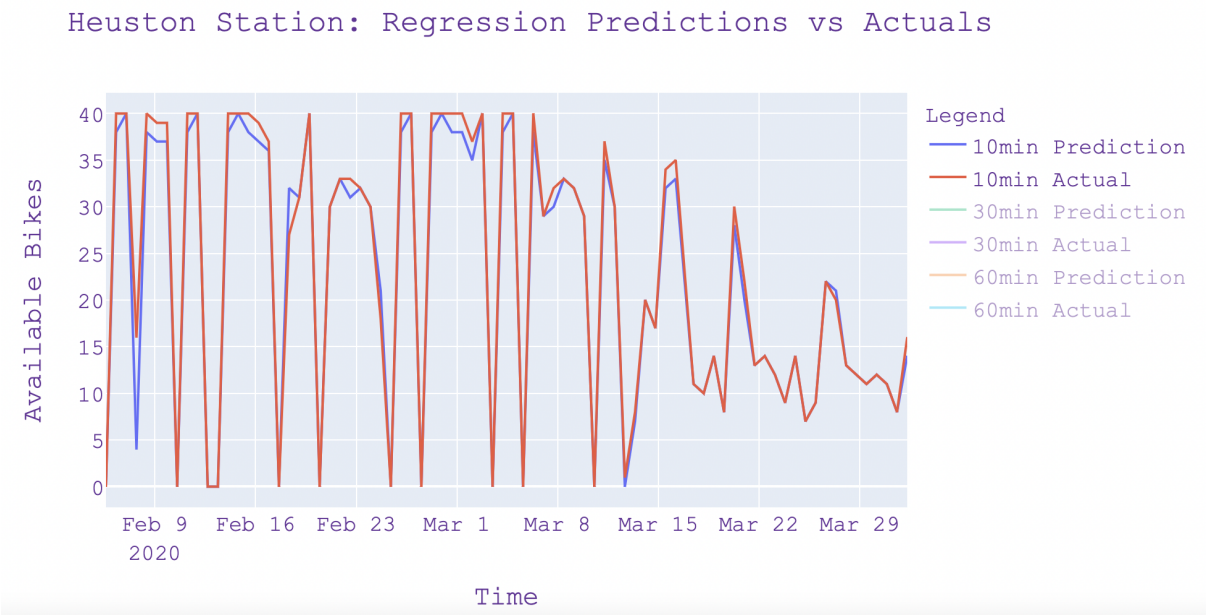
Evaluation

I use Root Mean Square Log Error (RMSLE) when evaluating these models because it offers multiple advantages over the more common Root Mean Square Error (RMSE) in this instance. For one, RMSLE performs much better in the event of outliers than Root Mean Square Error (RMSE) does. RMSLE also calculates the relative error between the prediction and actual value, whereas RMSE does not. This is useful in the case of predicting the number of available bikes because as a user looking for an available bike before they arrive at the station, the difference between 1 bike and 0 is far more significant than the difference between 40 bikes and 39. Thus prioritising the relative error penalises mistakes more when the true number of available bikes is low which is important.

The following is a comparison of predicted and actual values for 10, 30, and 60 minute predictions. All predictions were made on unseen data using the forward chaining method discussed earlier. This gives confidence that the predictions are not the result of overfitting.

Heuston Station

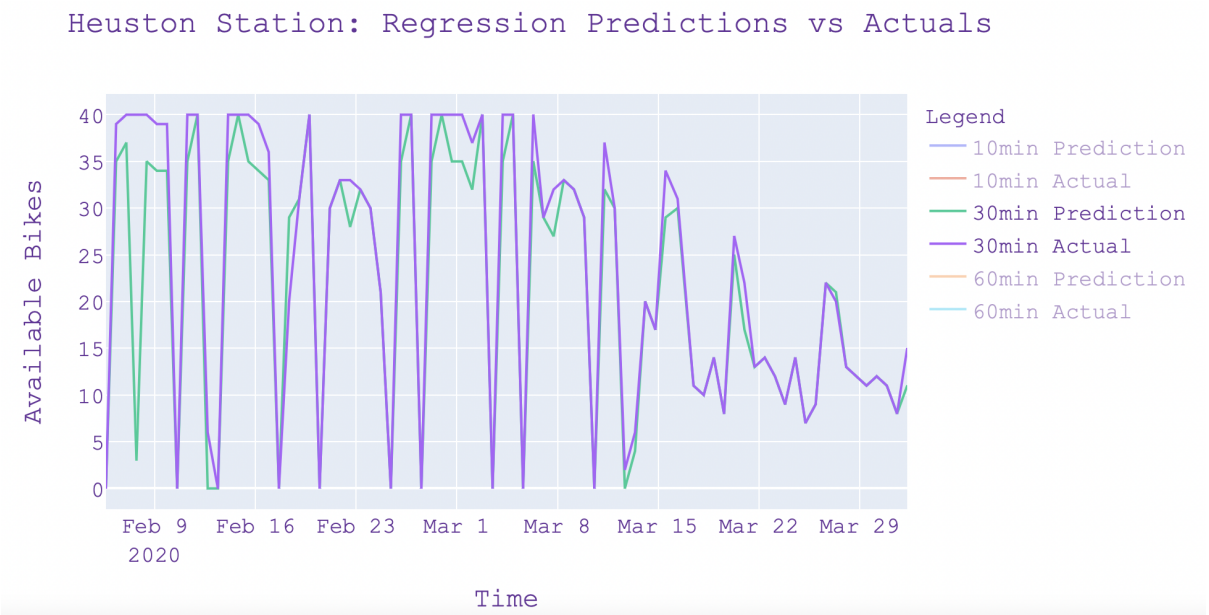
10min Predictions



The RMSLE of these 10min predictions was 0.0269

An RMSLE of 0.0269 means that actual values will mostly be close to plus or minus 2.69% of the predicted value - a little less if it is an overestimation and a little more if it is an underestimation. This confirms what the above graph seems to show, which is that 10 minute predictions of Heuston Station's bike availability is extremely accurate with this Ridge Regression model.

30min Predictions

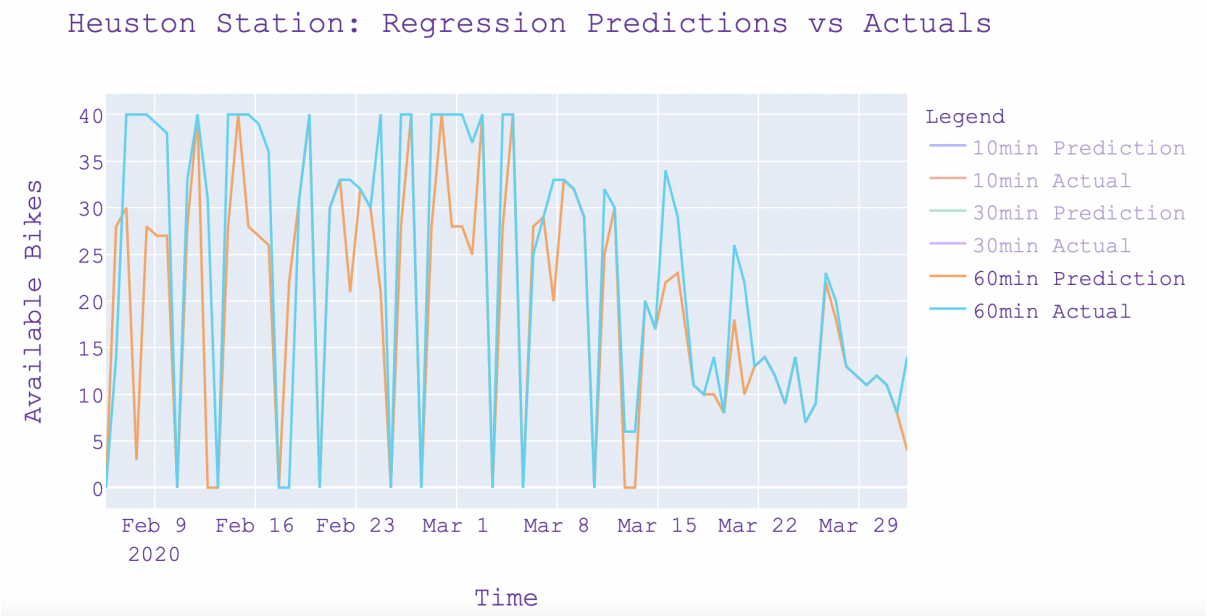


The RMSLE of 30min predictions 0.1413

Predictions 30 minutes ahead have a relative error of 14% which is significantly higher than predictions 10 minutes ahead. This is likely due to the cumulative nature of errors in k-step ahead predictions. In k-step ahead, predictions rely on previous predictions which means the error of one prediction is compounded in the following prediction that grows in a snowball effect.

In a practical application, 14% relative error is likely still acceptable for predicting bike availability since it will rarely deter a user from going to a bike station. For instance, you can see in the graph above that there are only two occasions where the model incorrectly predicts that there will be no more bikes available for the user to take when there actually were.

60min Predictions

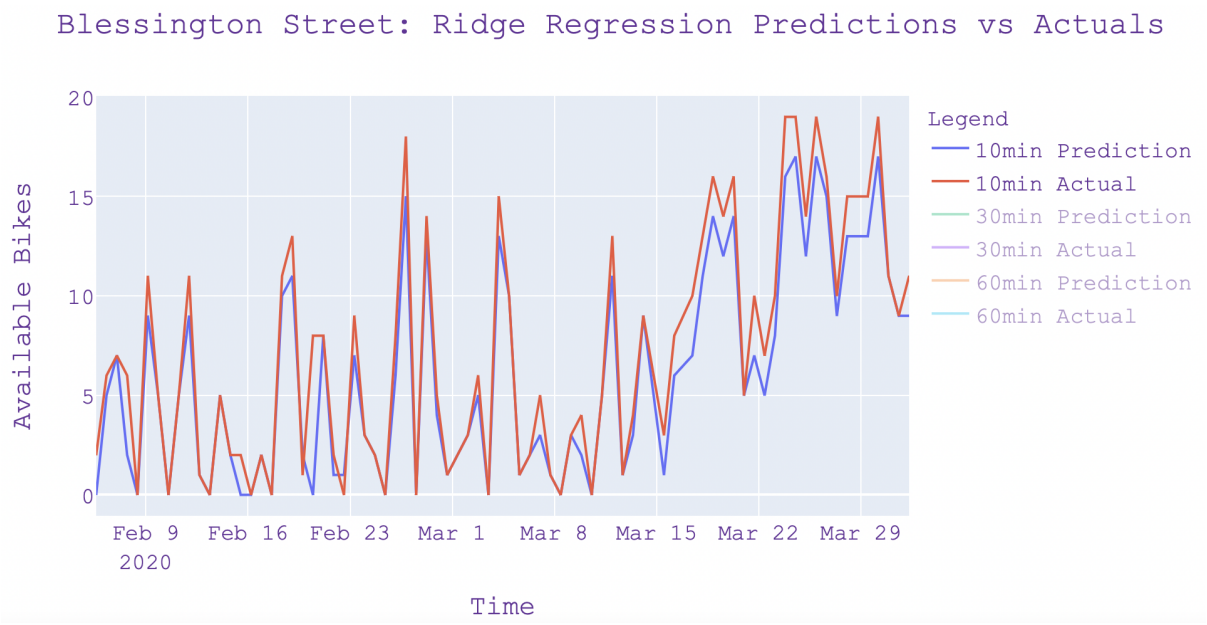


The RMSLE of 60min predictions 0.4015

These predictions are also functionally sufficient like the 30 minute predictions, but the snowballing nature of errors in q-step ahead as discussed earlier makes it difficult to accurately predict so far in the future.

Blessington Street

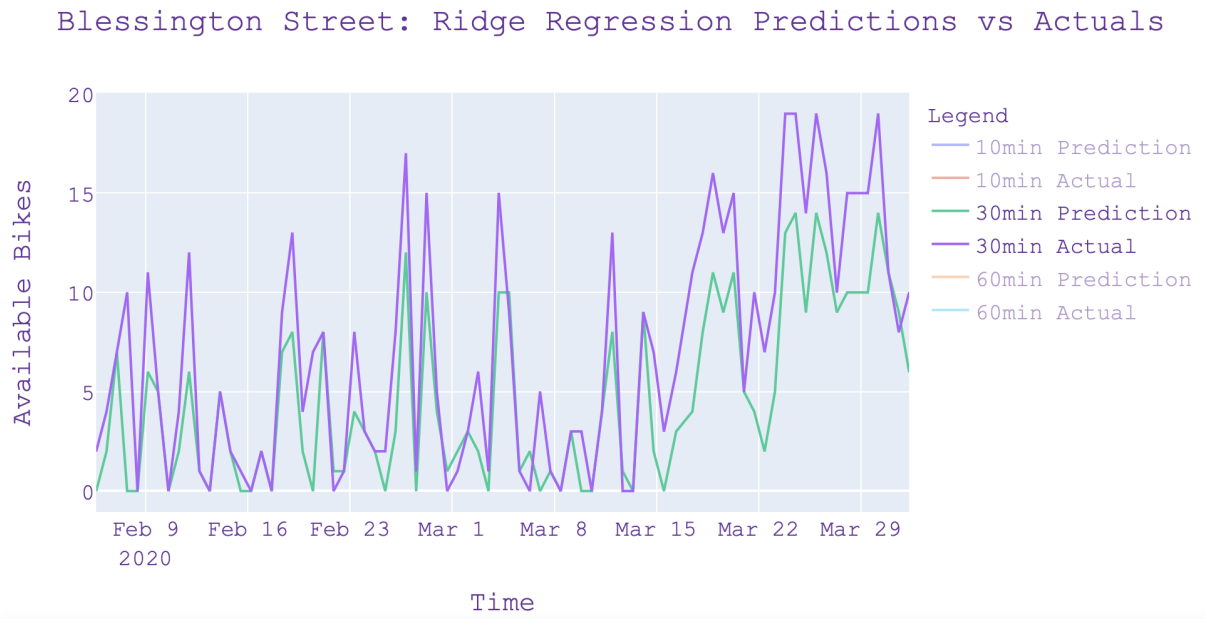
10min Predictions



RMSLE of these 10min predictions 0.1362

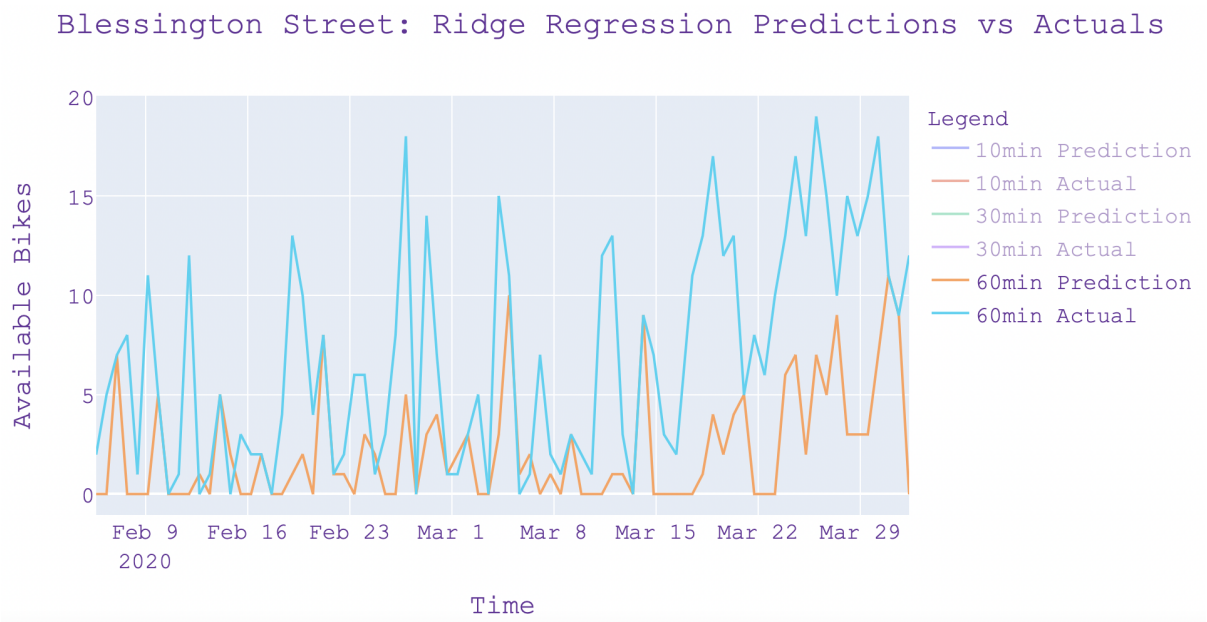
An RMSLE of 0.1362 for 10min predictions is worse than the model trained on the Heuston Station dataset, but it is still sufficiently accurate for the purpose of determining if a bike will be available for use in the next 10 minutes.

30min Predictions



RMSLE of 30min predictions [0.4254]
The relative error rate increased significantly compared to the 10 minute predictions as expected. The predictions are still accurate enough to serve their function, but there is much uncaptured information still in the initial dataset.

60min Predictions



RMSLE of 60min predictions 1.6649

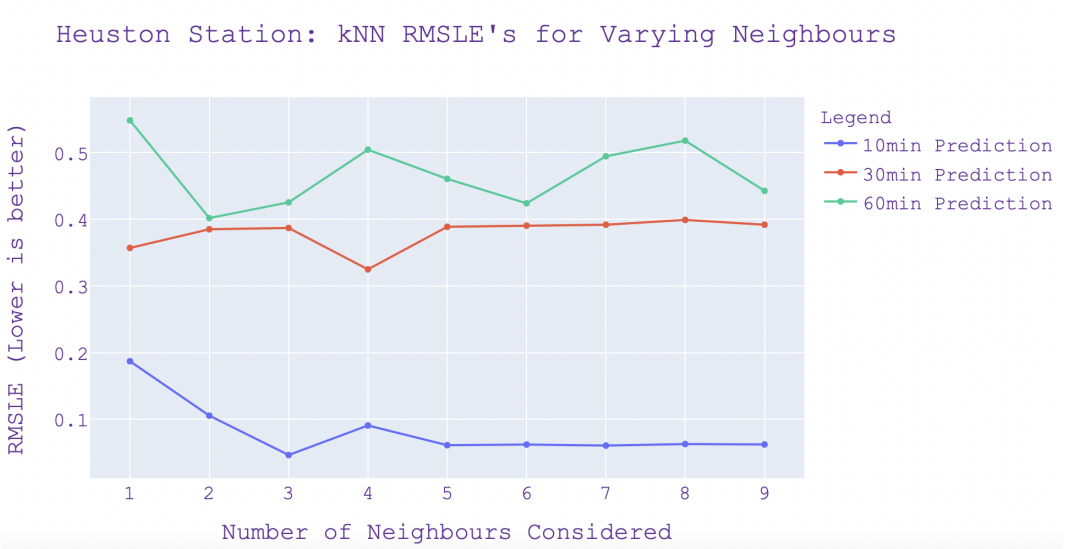
An RMSLE of 1.6649 shows that the Ridge model performs very poorly with 60 minute predictions on the Blessington Street data. As can be seen in the above graph, there are number instances when the model incorrectly predicts that bike availability will drop to 0 which is a big problem for the usability of the predictions.

k-Nearest Neighbour

Hyperparameter Tuning

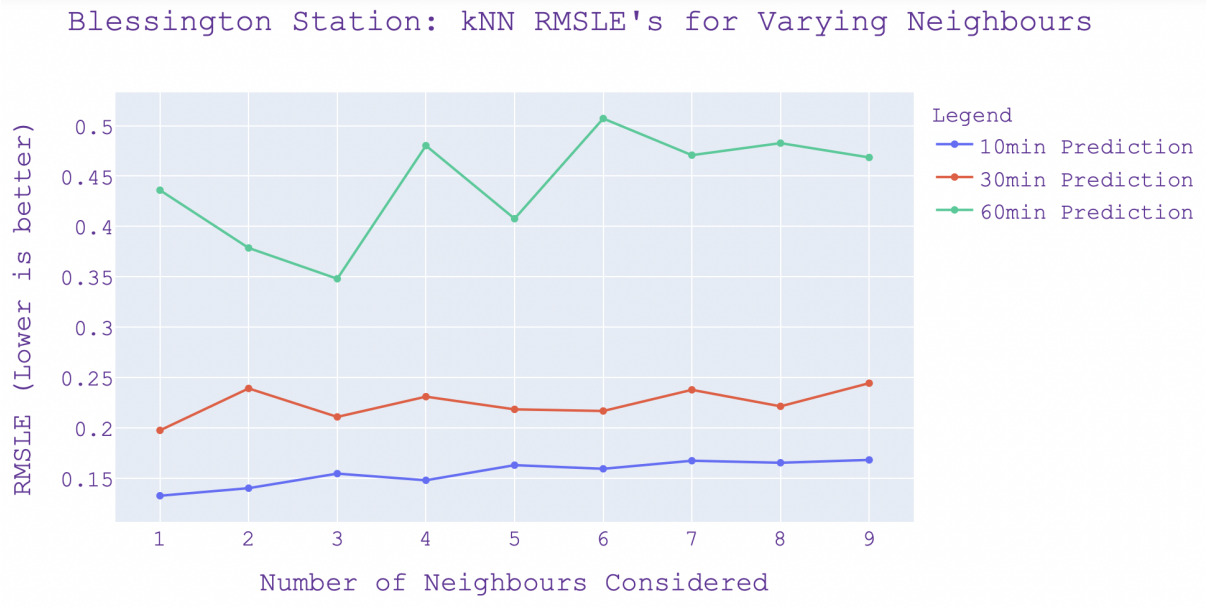
The optimal k-value is determined using the same forward-chaining method used to determine the Ridge model's optimal alpha value.

Heuston Station



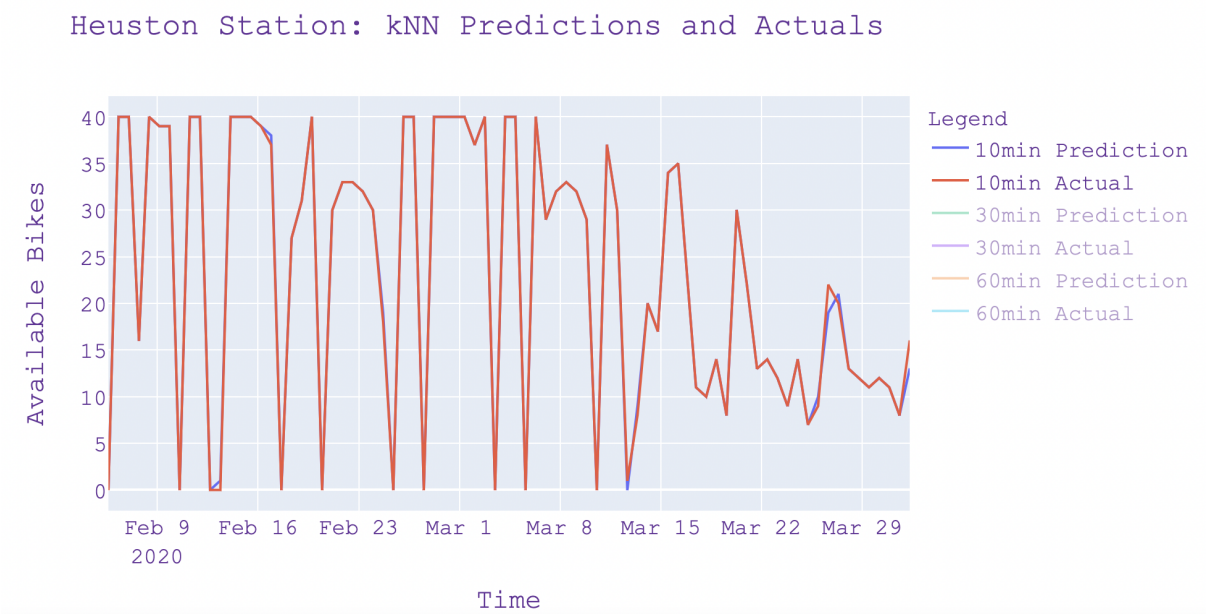
This graph shows that [10, 30, 60] minute ahead predictions perform best when considering the [3, 4, 2] closest neighbours. I decided to use k = 3 as the optimal tuning because while k = 2 optimises 60 minute predictions, it impacts 10 minute predictions too heavily. And while k = 4 optimises 30 minute predictions, it impacts both 10 minute and 60 min predictions heavily.

Blessington Street



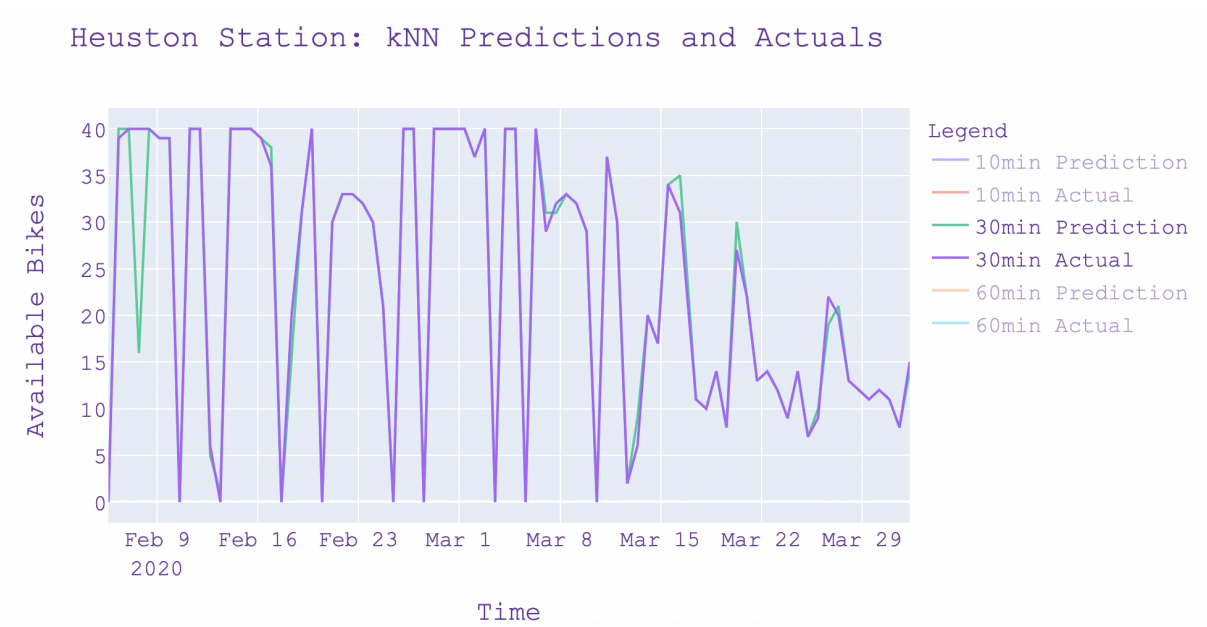
This graph shows that 30, and 60 minute ahead predictions all perform best when considering the 3 closest neighbours while 10 minute predictions prefer 1 neighbour. This led me to use k = 3 as the optimal tuning because while k = 1 risks underfitting and is insufficient for 30 and 60 minute predictions.

Evaluation



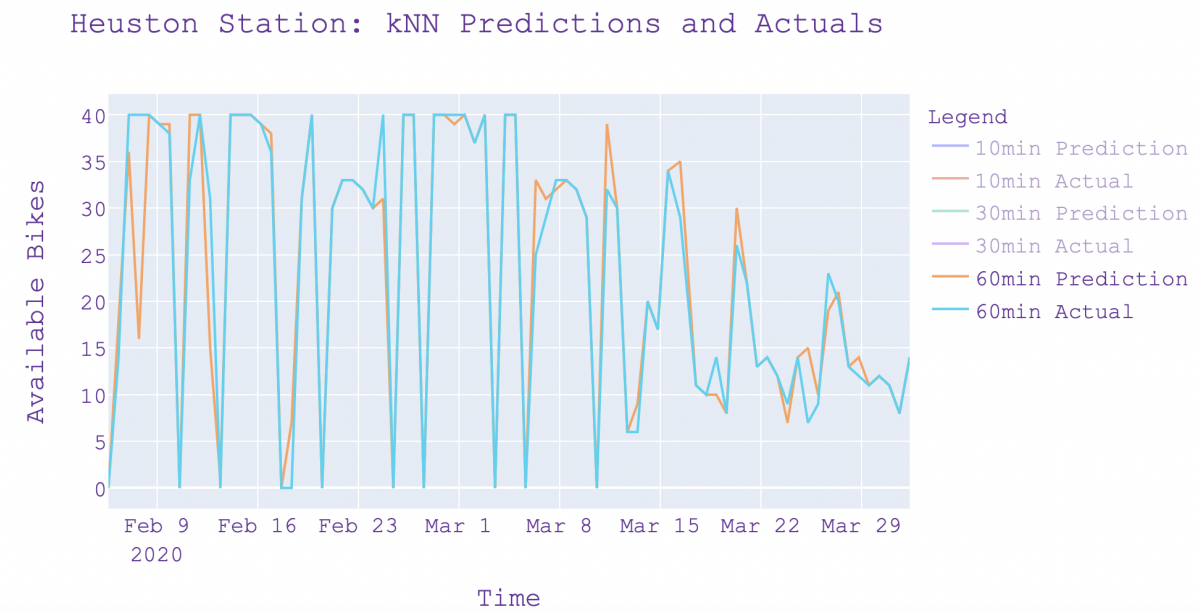
RMSLE = 0.013212

It can be said given this RMSLE that the kNN model is a very accurate predictor of available bikes 10 minutes ahead.



RMSLE = 0.01922

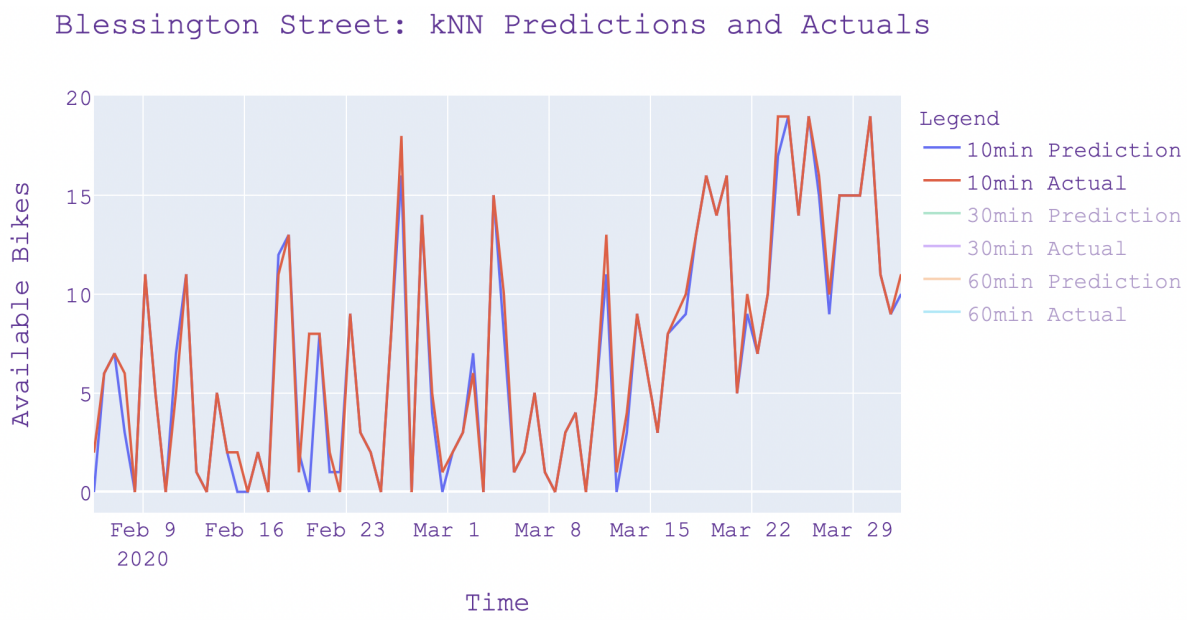
Where the Ridge model RMSLE ballooned for 30 minute predictions, kNN's RMSLE remains very similar. This suggests that the models have better captured the dataset than the Ridge model.



RMSLE = 0.09128

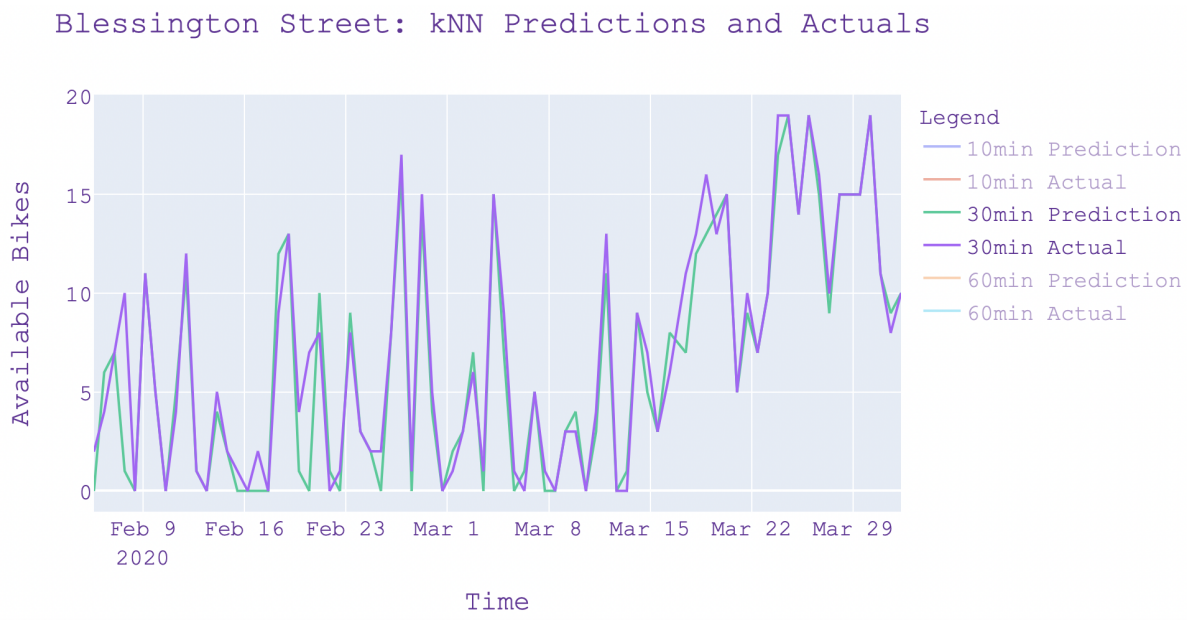
Again the kNN model performs surprisingly well, maintaining a low RMSLE for 60 minute predictions.

Blessington Station



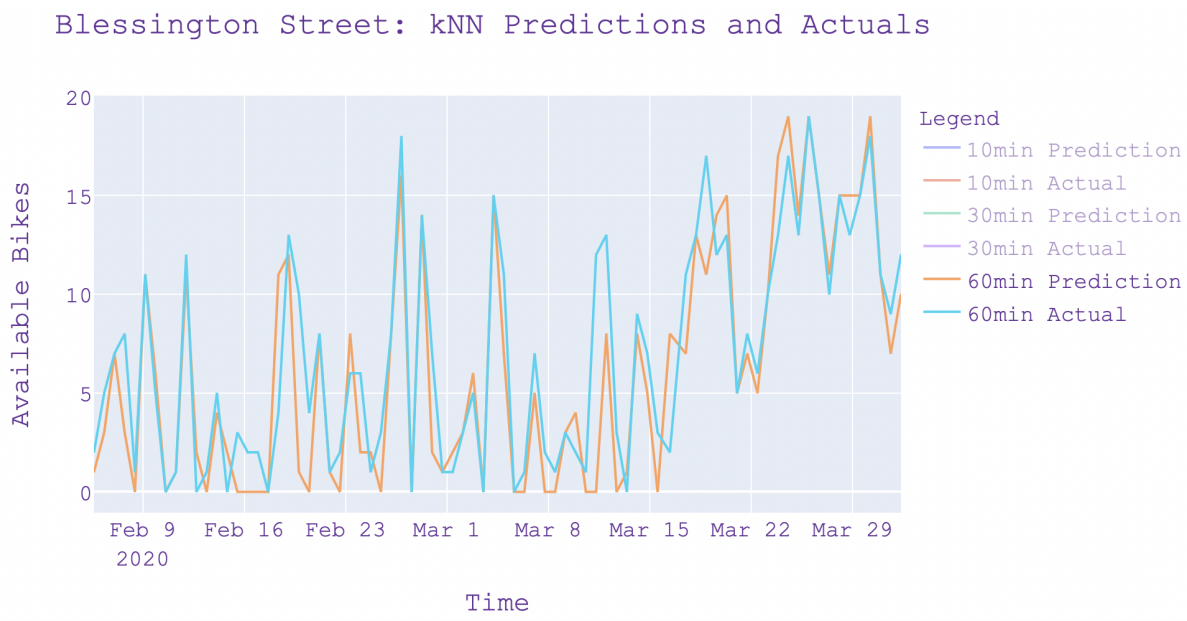
RMSLE = 0.1117

Though still better than the Ridge model, the kNN model's performance is worse for Blessington Street.



RMSLE = 0.2624

3o minute predictions of Blessington Street continue the trend of outperforming the Ridge model but doing worse than Heuston Station.



RMSLE = 0.5196428

kNN predictions for 60 minutes ahead perform substantially better than the Ridge model alternative.

Results

The kNN models outperformed the Ridge models on both datasets. Both models performed better on Heuston Station than Blessington Street. The Heuston Station overperformance could be explained by the respective locations of the bike stations. Heuston Station is a commuter hub and has a strong pattern during the workdays while Blessington Street is a station surrounded by houses and so is subject to a more erratic and unpredictable pattern.

Appendix

[Jupyter Notebook file attached]

```
#! pip install pandas-datareader
#! pip install pmdarima
#! pip install plotly
#! pip install dash-bootstrap-components
#! pip install sklearn
import numpy as np
from pandas_datareader import DataReader # pip install pandas-datareader
from pandas_datareader import data
from datetime import datetime
from pmdarima.arima import *
from pmdarima import preprocessing
from scipy import stats
from scipy.stats import skew
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from sklearn.linear_model import Ridge
from sklearn.linear_model import RidgeCV
from statsmodels import api as sm
from statsmodels.tsa.seasonal import seasonal_decompose

import pandas as pd
import pmdarima as pm
import plotly.graph_objects as go
import plotly.express as px
from plotly.tools import mpl_to_plotly
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_log_error



hs_df = pd.read_csv('dataset1.csv') #heuston station
ph_df = pd.read_csv('dataset2.csv') #portobello harbour
def testLevelStationarity(ts):
    adf_result = sm.tsa.stattools.adfuller(ts)
    kpss_result = sm.tsa.stattools.kpss(ts)
    print(f'ADF p-value: {adf_result[1]}')
    print(f'KPPS p-value: {kpss_result[1]}')

    d = ndiffs(ts, test='kpss')
    d += ndiffs(ts, test='adf')
    d += ndiffs(ts, test='pp')

    return int(d/3)
```

```
test = sm.tsa.stattools.adfuller(hs_df['AVAILABLE BIKES'])
print(test[0])
print(test[1])

test = sm.tsa.stattools.adfuller(ph_df['AVAILABLE BIKES'])
print(test[0])
print(test[1])

min_5 = 1 #2 x 5 minute samples = 10 minutes
min_30 = 6 #6 x 5 minute samples = 30 minutes
hour_1 = min_30 * 2
day_1 = min_30*24*2 #1 day worth of 5 min samples

week_1 = day_1*7 #1 week worth of 5 min samples
hs_vis = px.scatter(title="s")
hs_vis.update_layout(
    title="Heuston Station - Available Bikes over Time",
    xaxis_title="Time",
    yaxis_title="Available Bikes",
    legend_title="Legend",
    font=dict(
        family="Courier New, monospace",
        size=18,
        color="RebeccaPurple"
    )
)
hs_vis.add_trace(
    go.Scatter(name='Heuston Station',
          x=hs_df['TIME'],
          y=hs_df['AVAILABLE BIKES']))

hs_vis = px.scatter(title="s")
hs_vis.update_layout(
    title="Blessington Street: Available Bikes over Time",
    xaxis_title="Time",
    yaxis_title="Available Bikes",
    legend_title="Legend",
    font=dict(
        family="Courier New, monospace",
        size=18,
        color="RebeccaPurple"
    )
)
hs_vis.add_trace(
    go.Scatter(name='Portobello Harbour',
          x=ph_df['TIME'],
          y=ph_df['AVAILABLE BIKES']))
hs_vis.show()
```

```
import datetime as dt
from sklearn.model_selection import train_test_split
<h1> Heuston Station: Ridge Regression </h1>
<h2> ACF Plot </h2>
from statsmodels.graphics.tsaplots import plot_acf

plot_acf(hs_df['AVAILABLE BIKES'].values, lags=3000)
<h2> Model Fit + Predictions </h2>
X = pd.to_datetime(hs_df['TIME'])
temp = X
X = X.map(dt.datetime.toordinal)
X = X.to_frame()

y = hs_df['AVAILABLE BIKES'].values
y = y.reshape(-1,1)
alphas = 0

rmsle10 = []
rmsle30 = []
rmsle60 = []
while alphas < 1:
    test_size = 0.01
    y_preds10 = []
    y_actuals10 = []
    y_preds30 = []
    y_actuals30 = []
    y_preds60 = []
    y_actuals60 = []


    while test_size < 0.8:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= test_size, shuffle =
False)

        # define model
        model = Ridge(alpha=alphas)



        from numpy import array


        i = week_1
        j = 0
        rows = int (X_train.size - week_1)

        X_list = [[0]*5]*rows
        y_list = []
```

```
    # X_list[j] = [X_train.TIME.iloc[i], X_train.TIME.iloc[i-min_30],
X_train.TIME.iloc[i-day_1+min_30], X_train.TIME.iloc[i-week_1+min_30]]

    #create feature vector of sliding 5 min samples
    while i < y_train.size:
        X_list[j] = [y_train[i-min_5],y_train[i-min_5*2], y_train[i-hour_1*8],
y_train[i-day_1+min_5], y_train[i-week_1+min_5]]
        y_list.append(y_train[i])
        i+= 1
        j+= 1
    X_list = array(X_list)
    X_list  = X_list[:, :, 0]

    i = 0

    X_list = X_list.tolist()
    model.fit(X_list,y_list)



    while i < 12:
    # fit model
        temp = [y_list[-min_5],y_list[-min_5*2], y_list[-hour_1*8], y_list[-day_1+min_5],
y_list[-week_1+min_5]]
        temp = np.array(temp).reshape(1, -1)



        # # make a prediction
        yhat = model.predict(temp)
        yhat = int(yhat[0])

        if i == 1:
            y_preds10.append(yhat)
            y_actuals10.append(y_test[i][0])
        elif i == 4:
            y_preds30.append(yhat)
            y_actuals30.append(y_test[i][0])
        elif i == 11:
            y_preds60.append(yhat)
            y_actuals60.append(y_test[i][0])



        X_list.append([y_list[-1],y_list[-min_5*2], y_list[-hour_1*8],
y_list[-day_1+min_5],y_list[-day_1*6+min_5], y_list[-week_1+min_5]])

        y_list.append(yhat)
```

```
            i += 1
        test_size += 0.01
        #mean_squared_error(y_actuals, y_preds)

    rmsle10.append(mean_squared_log_error(y_actuals10, y_preds10))
    rmsle30.append(mean_squared_log_error(y_actuals30, y_preds30))
    rmsle60.append(mean_squared_log_error(y_actuals60, y_preds60))
#     print("Number of close neighbours used: ", neigh)
#     print("RMSLE of predictions 10 minutes ahead: ", rmsle10)
#     print("RMSLE of predictions 30 minutes ahead: ", rmsle30)
#     print("RMSLE of predictions 60 minutes ahead: ", rmsle60)
#     print("\n")

    alphas += .1
<h1> Portobello Harbour: Ridge Regression</h1>
<h2> ACF Plot </h2>
from statsmodels.graphics.tsaplots import plot_acf

plot_acf(ph_df['AVAILABLE BIKES'].values, lags=3000)

from statsmodels.tsa.stattools import adfuller

result = adfuller(ph_df['AVAILABLE BIKES'].values)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))
<h2> Model Fit + Predictions </h2>
X = pd.to_datetime(ph_df['TIME'])
temp = X
X = X.map(dt.datetime.toordinal)
X = X.to_frame()

y = ph_df['AVAILABLE BIKES'].values
y = y.reshape(-1,1)
alphas = 0

rmsle10 = []
rmsle30 = []
rmsle60 = []
while alphas < 1:
    test_size = 0.01
    y_preds10 = []
    y_actuals10 = []
    y_preds30 = []
    y_actuals30 = []
```

```
    y_preds60 = []
    y_actuals60 = []


    while test_size < 0.8:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= test_size, shuffle =
False)

        # define model
        model = Ridge(alpha=alphas)



        from numpy import array


        i = week_1
        j = 0
        rows = int (X_train.size - week_1)

        X_list = [[0]*5]*rows
        y_list = []
        # X_list[j] = [X_train.TIME.iloc[i], X_train.TIME.iloc[i-min_30],
X_train.TIME.iloc[i-day_1+min_30], X_train.TIME.iloc[i-week_1+min_30]]

        #create feature vector of sliding 5 min samples
        while i < y_train.size:
            X_list[j] = [y_train[i-min_5],y_train[i-min_5*2], y_train[i-day_1+min_5],
y_train[i-week_1+min_5]]
            y_list.append(y_train[i])
            i+= 1
            j+= 1
        X_list = array(X_list)
        X_list  = X_list[:, :, 0]


        X_list = X_list.tolist()
        model.fit(X_list,y_list)

        i = 0
        while i < 12:
        # fit model
            temp = [y_list[-min_5],y_list[-min_5*2], y_list[-day_1+min_5], y_list[-week_1+min_5]]
            temp = np.array(temp).reshape(1, -1)



            # # make a prediction
            yhat = model.predict(temp)
```

```
            yhat = int(yhat[0])

            if i == 1:
                y_preds10.append(yhat)
                y_actuals10.append(y_test[i][0])
            elif i == 4:
                y_preds30.append(yhat)
                y_actuals30.append(y_test[i][0])
            elif i == 11:
                y_preds60.append(yhat)
                y_actuals60.append(y_test[i][0])


            X_list.append([y_list[-1],y_list[-min_5*2], y_list[-day_1+min_5],
y_list[-week_1+min_5]])

            y_list.append(yhat)




            i += 1
        test_size += 0.01
        #mean_squared_error(y_actuals, y_preds)

    rmsle10.append(mean_squared_log_error(y_actuals10, y_preds10))
    rmsle30.append(mean_squared_log_error(y_actuals30, y_preds30))
    rmsle60.append(mean_squared_log_error(y_actuals60, y_preds60))
#     print("RMSLE of predictions 10 minutes ahead: ", rmsle10)
#     print("RMSLE of predictions 30 minutes ahead: ", rmsle30)
#     print("RMSLE of predictions 60 minutes ahead: ", rmsle60)
#     print("\n")

    alphas += .1

xaxis = [0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]
hs_vis = px.scatter(title="s")
hs_vis.update_layout(
    title="Blessington: Ridge RMSLE's for Varying Alphas",
    xaxis_title="Alpha",
    yaxis_title="RMSLE (Lower is better)",
    legend_title="Legend",
    font=dict(
        family="Courier New, monospace",
        size=18,
        color="RebeccaPurple"
    )
)
```

```
hs_vis.add_trace(
   go.Scatter(name='10min Prediction',
         x=xaxis,
         y=rmsle10))
hs_vis.add_trace(
   go.Scatter(name='30min Prediction',
         x=xaxis,
         y=rmsle30))
hs_vis.add_trace(
   go.Scatter(name='60min Prediction',
         x=xaxis,
         y=rmsle60))
<h2> Evaluation </h2>
<h3> Heuston Station</h3>
X = pd.to_datetime(hs_df['TIME'])
X = X.tolist()
y = hs_df['AVAILABLE BIKES'].values
y = y.reshape(-1,1)
alphas = 0

rmsle10 = []
rmsle30 = []
rmsle60 = []

test_size = 0.01
y_preds10 = []
y_actuals10 = []
x_axis10 = []

y_preds30 = []
y_actuals30 = []
x_axis30 = []


y_preds60 = []
y_actuals60 = []
x_axis60 = []



while test_size < 0.7:
   X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= test_size, shuffle = False)

   # define model
   model = Ridge(alpha=1)
```

```
    from numpy import array


    i = week_1
    j = 0
    rows = int (len(X_train) - week_1)

    X_list = [[0]*5]*rows
    y_list = []

    #create feature vector of sliding 5 min samples
    while i < y_train.size:
        X_list[j] = [y_train[i-min_5],y_train[i-min_5*2], y_train[i-hour_1*8],
y_train[i-day_1+min_5], y_train[i-week_1+min_5]]
        y_list.append(y_train[i])
        i+= 1
        j+= 1
    X_list = array(X_list)
    X_list  = X_list[:, :, 0]

    i = 0

    X_list = X_list.tolist()
    model.fit(X_list,y_list)



    while i < 12:
    # fit model
        temp = [y_list[-min_5],y_list[-min_5*2], y_list[-hour_1*8], y_list[-day_1+min_5],
y_list[-week_1+min_5]]
        temp = np.array(temp).reshape(1, -1)



        # # make a prediction
        yhat = model.predict(temp)
        yhat = int(yhat[0])

        if i == 1:
            y_preds10.append(yhat)
            y_actuals10.append(y_test[i][0])
            x_axis10.append(X_test[i])

        elif i == 4:
            y_preds30.append(yhat)
            y_actuals30.append(y_test[i][0])
            x_axis30.append(X_test[i])
```

```
        elif i == 11:
            y_preds60.append(yhat)
            y_actuals60.append(y_test[i][0])
            x_axis60.append(X_test[i])



        X_list.append([y_list[-1],y_list[-min_5*2], y_list[-hour_1*8],
y_list[-day_1+min_5],y_list[-week_1+min_5]])

        y_list.append(yhat)




        i += 1
    test_size += 0.01
    #mean_squared_error(y_actuals, y_preds)

rmsle10.append(mean_squared_log_error(y_actuals10, y_preds10))
rmsle30.append(mean_squared_log_error(y_actuals30, y_preds30))
rmsle60.append(mean_squared_log_error(y_actuals60, y_preds60))
print(rmsle10)
print(rmsle30)
print(rmsle60)
hs_vis = px.scatter(title="s")
hs_vis.update_layout(
    title="Heuston Station: Ridge Regression Predictions vs Actuals",
    xaxis_title="Time",
    yaxis_title="Available Bikes",
    legend_title="Legend",
    font=dict(
        family="Courier New, monospace",
        size=18,
        color="RebeccaPurple"
    )
)
hs_vis.add_trace(
    go.Scatter(name='10min Prediction',
           x=x_axis10,
          y=y_preds10))
hs_vis.add_trace(
    go.Scatter(name='10min Actual',
           x=x_axis10,
          y=y_actuals10))

hs_vis.add_trace(
    go.Scatter(name='30min Prediction',
```

```
            x=x_axis30,
           y=y_preds30))
hs_vis.add_trace(
    go.Scatter(name='30min Actual',
            x=x_axis30,
           y=y_actuals30))

hs_vis.add_trace(
    go.Scatter(name='60min Prediction',
            x=x_axis60,
           y=y_preds60))
hs_vis.add_trace(
    go.Scatter(name='60min Actual',
            x=x_axis60,
           y=y_actuals60))
RMSLE
print("RMSLE of 10min predictions", rmsle10)
print("RMSLE of 30min predictions", rmsle30)
print("RMSLE of 60min predictions", rmsle60)

print(mean_squared_error(y_actuals10, y_preds10))
print(mean_squared_error(y_actuals30, y_preds30))
print(mean_squared_error(y_actuals60, y_preds60))
<h2> Blessington Street </h2>
X = pd.to_datetime(ph_df['TIME'])
X = X.tolist()
y = ph_df['AVAILABLE BIKES'].values
y = y.reshape(-1,1)
alphas = 0

rmsle10 = []
rmsle30 = []
rmsle60 = []

test_size = 0.01
y_preds10 = []
y_actuals10 = []
x_axis10 = []

y_preds30 = []
y_actuals30 = []
x_axis30 = []



y_preds60 = []
y_actuals60 = []
x_axis60 = []
```

```
while test_size < 0.8:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= test_size, shuffle = False)

    # define model
    model = Ridge(alpha=1)



    from numpy import array


    i = week_1
    j = 0
    rows = int (len(X_train) - week_1)

    X_list = [[0]*6]*rows
    y_list = []

    #create feature vector of sliding 5 min samples
    while i < y_train.size:
        X_list[j] = [y_train[i-min_5],y_train[i-min_5*2],
y_train[i-day_1+min_5],y_train[i-day_1*2+min_5], y_train[i-week_1+min_5]]
        y_list.append(y_train[i])
        i+= 1
        j+= 1
    X_list = array(X_list)
    X_list  = X_list[:, :, 0]

    i = 0

    X_list = X_list.tolist()
    model.fit(X_list,y_list)



    while i < 12:
    # fit model
        temp = [y_list[-min_5],y_list[-min_5*2], y_list[-day_1+min_5], y_list[-day_1*2+min_5],
y_list[-week_1+min_5]]
        temp = np.array(temp).reshape(1, -1)



        # # make a prediction
        yhat = model.predict(temp)
        yhat = int(yhat[0])

        if i == 1:
```

```
            y_preds10.append(yhat)
            y_actuals10.append(y_test[i][0])
            x_axis10.append(X_test[i])

        elif i == 4:
            y_preds30.append(yhat)
            y_actuals30.append(y_test[i][0])
            x_axis30.append(X_test[i])


        elif i == 11:
            y_preds60.append(yhat)
            y_actuals60.append(y_test[i][0])
            x_axis60.append(X_test[i])



        X_list.append([y_list[-1],y_list[-min_5*2],
y_list[-day_1+min_5],y_list[-day_1*2+min_5],y_list[-week_1+min_5]])

        y_list.append(yhat)




        i += 1
    test_size += 0.01
    #mean_squared_error(y_actuals, y_preds)

rmsle10.append(mean_squared_log_error(y_actuals10, y_preds10))
rmsle30.append(mean_squared_log_error(y_actuals30, y_preds30))
rmsle60.append(mean_squared_log_error(y_actuals60, y_preds60))
print(rmsle10)
print(rmsle30)
print(rmsle60)
print(model.coef_)
hs_vis = px.scatter(title="s")
hs_vis.update_layout(
    title="Blessington Street: Ridge Regression Predictions vs Actuals",
    xaxis_title="Time",
    yaxis_title="Available Bikes",
    legend_title="Legend",
    font=dict(
        family="Courier New, monospace",
        size=18,
        color="RebeccaPurple"
    )
)
```

```
hs_vis.add_trace(
    go.Scatter(name='10min Prediction',
               x=x_axis10,
              y=y_preds10))
hs_vis.add_trace(
    go.Scatter(name='10min Actual',
               x=x_axis10,
              y=y_actuals10))

hs_vis.add_trace(
    go.Scatter(name='30min Prediction',
               x=x_axis30,
              y=y_preds30))
hs_vis.add_trace(
    go.Scatter(name='30min Actual',
               x=x_axis30,
              y=y_actuals30))

hs_vis.add_trace(
    go.Scatter(name='60min Prediction',
               x=x_axis60,
              y=y_preds60))
hs_vis.add_trace(
    go.Scatter(name='60min Actual',
               x=x_axis60,
              y=y_actuals60))
<h1> kNN </h1>
<h2> Heuston Station </h2>
from sklearn.neighbors import KNeighborsRegressor


X = pd.to_datetime(hs_df['TIME'])
temp = X
X = X.map(dt.datetime.toordinal)
X = X.to_frame()

y = ph_df['AVAILABLE BIKES'].values
y = y.reshape(-1,1)
neigh = 1

rmsle10 = []
rmsle30 = []
rmsle60 = []
while neigh < 10:
    test_size = 0.01
    y_preds10 = []
    y_actuals10 = []
    y_preds30 = []
```

```
    y_actuals30 = []
    y_preds60 = []
    y_actuals60 = []


    while test_size < 0.8:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= test_size, shuffle =
False)

        # define model
        model = KNeighborsRegressor(n_neighbors=neigh)



        from numpy import array


        i = week_1
        j = 0
        rows = int (X_train.size - week_1)

        X_list = [[0]*5]*rows
        y_list = []
        # X_list[j] = [X_train.TIME.iloc[i], X_train.TIME.iloc[i-min_30],
X_train.TIME.iloc[i-day_1+min_30], X_train.TIME.iloc[i-week_1+min_30]]

        #create feature vector of sliding 5 min samples
        while i < y_train.size:
            X_list[j] = [y_train[i-min_5],y_train[i-min_5*2], y_train[i-hour_1*8],
y_train[i-day_1+min_5], y_train[i-week_1+min_5]]
            y_list.append(y_train[i])
            i+= 1
            j+= 1
        X_list = array(X_list)
        X_list  = X_list[:, :, 0]

        i = 0

        X_list = X_list.tolist()
        model.fit(X_list,y_list)



        while i < 12:
        # fit model
            temp = [y_list[-min_5],y_list[-min_5*2], y_list[-hour_1*8], y_list[-day_1+min_5],
y_list[-week_1+min_5]]
            temp = np.array(temp).reshape(1, -1)
```

```
            # # make a prediction
            yhat = model.predict(temp)
            yhat = int(yhat[0])

            if i == 1:
                y_preds10.append(yhat)
                y_actuals10.append(y_test[i][0])
            elif i == 4:
                y_preds30.append(yhat)
                y_actuals30.append(y_test[i][0])
            elif i == 11:
                y_preds60.append(yhat)
                y_actuals60.append(y_test[i][0])


            X_list.append([y_list[-1],y_list[-min_5*2], y_list[-hour_1*8], y_list[-day_1+min_5],
y_list[-week_1+min_5]])

            y_list.append(yhat)




            i += 1
        test_size += 0.01
        #mean_squared_error(y_actuals, y_preds)

    rmsle10.append(mean_squared_log_error(y_actuals10, y_preds10))
    rmsle30.append(mean_squared_log_error(y_actuals30, y_preds30))
    rmsle60.append(mean_squared_log_error(y_actuals60, y_preds60))
#     print("Number of close neighbours used: ", neigh)
#     print("RMSLE of predictions 10 minutes ahead: ", rmsle10)
#     print("RMSLE of predictions 30 minutes ahead: ", rmsle30)
#     print("RMSLE of predictions 60 minutes ahead: ", rmsle60)
    print("\n")
    neigh += 1

xaxis = [1,2,3,4,5,6,7,8,9]
hs_vis = px.scatter(title="s")
hs_vis.update_layout(
    title="Heuston Station: kNN RMSLE's for Varying Neighbours",
    xaxis_title="Number of Neighbours Considered",
    yaxis_title="RMSLE (Lower is better)",
    legend_title="Legend",
    font=dict(
        family="Courier New, monospace",
        size=18,
```

```
        color="RebeccaPurple"
    )
)
hs_vis.add_trace(
    go.Scatter(name='10min Prediction',
           x=xaxis,
           y=rmsle10))
hs_vis.add_trace(
    go.Scatter(name='30min Prediction',
           x=xaxis,
           y=rmsle30))
hs_vis.add_trace(
    go.Scatter(name='60min Prediction',
           x=xaxis,
           y=rmsle60))
<h2> kNN: Heuston Station </h2>
from sklearn.neighbors import KNeighborsRegressor


X = pd.to_datetime(ph_df['TIME'])
temp = X
X = X.map(dt.datetime.toordinal)
X = X.to_frame()

y = ph_df['AVAILABLE BIKES'].values
y = y.reshape(-1,1)
neigh = 1

rmsle10 = []
rmsle30 = []
rmsle60 = []
while neigh < 10:
    test_size = 0.01
    y_preds10 = []
    y_actuals10 = []
    y_preds30 = []
    y_actuals30 = []
    y_preds60 = []
    y_actuals60 = []


    while test_size < 0.8:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= test_size, shuffle =
False)

        # define model
        model = KNeighborsRegressor(n_neighbors=neigh)
```

```
        from numpy import array


        i = week_1
        j = 0
        rows = int (X_train.size - week_1)

        X_list = [[0]*5]*rows
        y_list = []
        # X_list[j] = [X_train.TIME.iloc[i], X_train.TIME.iloc[i-min_30],
X_train.TIME.iloc[i-day_1+min_30], X_train.TIME.iloc[i-week_1+min_30]]

        #create feature vector of sliding 5 min samples
        while i < y_train.size:
            X_list[j] = [y_train[i-min_5],y_train[i-min_5*2],
y_train[i-day_1+min_5],y_train[i-day_1*2+min_5], y_train[i-week_1+min_5]]
            y_list.append(y_train[i])
            i+= 1
            j+= 1
        X_list = array(X_list)
        X_list  = X_list[:, :, 0]

        i = 0

        X_list = X_list.tolist()
        model.fit(X_list,y_list)



        while i < 12:
        # fit model
            temp = [y_list[-min_5],y_list[-min_5*2], y_list[-day_1+min_5],y_list[-day_1*2+min_5],
y_list[-week_1+min_5]]
            temp = np.array(temp).reshape(1, -1)



            # # make a prediction
            yhat = model.predict(temp)
            yhat = int(yhat[0])

            if i == 1:
                y_preds10.append(yhat)
                y_actuals10.append(y_test[i][0])
            elif i == 4:
                y_preds30.append(yhat)
                y_actuals30.append(y_test[i][0])
            elif i == 11:
```

```
            y_preds60.append(yhat)
            y_actuals60.append(y_test[i][0])


        X_list.append([y_list[-1],y_list[-min_5*2],
y_list[-day_1+min_5],y_list[-day_1*2+min_5], y_list[-week_1+min_5]])

        y_list.append(yhat)




        i += 1
      test_size += 0.01
      #mean_squared_error(y_actuals, y_preds)

   rmsle10.append(mean_squared_log_error(y_actuals10, y_preds10))
   rmsle30.append(mean_squared_log_error(y_actuals30, y_preds30))
   rmsle60.append(mean_squared_log_error(y_actuals60, y_preds60))
#    print("Number of close neighbours used: ", neigh)
#    print("RMSLE of predictions 10 minutes ahead: ", rmsle10)
#    print("RMSLE of predictions 30 minutes ahead: ", rmsle30)
#    print("RMSLE of predictions 60 minutes ahead: ", rmsle60)
   print("\n")
   neigh += 1

xaxis = [1,2,3,4,5,6,7,8,9]
hs_vis = px.scatter(title="s")
hs_vis.update_layout(
   title="Blessington Station: kNN RMSLE's for Varying Neighbours",
   xaxis_title="Number of Neighbours Considered",
   yaxis_title="RMSLE (Lower is better)",
   legend_title="Legend",
   font=dict(
      family="Courier New, monospace",
      size=18,
      color="RebeccaPurple"
   )
)
hs_vis.add_trace(
   go.Scatter(name='10min Prediction',
        x=xaxis,
        y=rmsle10))
hs_vis.add_trace(
   go.Scatter(name='30min Prediction',
        x=xaxis,
        y=rmsle30))
hs_vis.add_trace(
```

```
    go.Scatter(name='60min Prediction',
           x=xaxis,
           y=rmsle60))
from sklearn.neighbors import KNeighborsRegressor


X = pd.to_datetime(hs_df['TIME'])
X = X.tolist()
y = hs_df['AVAILABLE BIKES'].values
y = y.reshape(-1,1)
neigh = 3

rmsle10 = []
rmsle30 = []
rmsle60 = []
test_size = 0.01
y_preds10 = []
y_actuals10 = []
x_axis10 = []

y_preds30 = []
y_actuals30 = []
x_axis30 = []


y_preds60 = []
y_actuals60 = []
x_axis60 = []



while test_size < 0.8:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= test_size, shuffle = False)

    # define model
    model = KNeighborsRegressor(n_neighbors=neigh)



    from numpy import array


    i = week_1
    j = 0
    rows = int (len(X_train) - week_1)

    X_list = [[0]*5]*rows
    y_list = []
```

```
    #create feature vector of sliding 5 min samples
    while i < y_train.size:
        X_list[j] = [y_train[i-min_5],y_train[i-min_5*2], y_train[i-hour_1*8],
y_train[i-day_1+min_5], y_train[i-week_1+min_5]]
        y_list.append(y_train[i])
        i+= 1
        j+= 1
    X_list = array(X_list)
    X_list  = X_list[:, :, 0]

    i = 0

    X_list = X_list.tolist()
    model.fit(X_list,y_list)



    while i < 12:
        temp = [y_list[-min_5],y_list[-min_5*2], y_list[-hour_1*8], y_list[-day_1+min_5],
y_list[-week_1+min_5]]
        temp = np.array(temp).reshape(1, -1)



        # # make a prediction
        yhat = model.predict(temp)
        yhat = int(yhat[0])

        if i == 1:
            y_preds10.append(yhat)
            y_actuals10.append(y_test[i][0])
            x_axis10.append(X_test[i])
        elif i == 4:
            y_preds30.append(yhat)
            y_actuals30.append(y_test[i][0])
            x_axis30.append(X_test[i])



        elif i == 11:
            y_preds60.append(yhat)
            y_actuals60.append(y_test[i][0])
            x_axis60.append(X_test[i])

        X_list.append([y_list[-1],y_list[-min_5*2], y_list[-hour_1*8], y_list[-day_1+min_5],
y_list[-week_1+min_5]])

        y_list.append(yhat)
```

```
        i += 1
    test_size += 0.01
    #mean_squared_error(y_actuals, y_preds)

rmsle10.append(mean_squared_log_error(y_actuals10, y_preds10))
rmsle30.append(mean_squared_log_error(y_actuals30, y_preds30))
rmsle60.append(mean_squared_log_error(y_actuals60, y_preds60))

print(rmsle10)
<h2> Evaluation </h2>
print(rmsle10)
print(rmsle30)
print(rmsle60)
hs_vis = px.scatter(title="s")
hs_vis.update_layout(
    title="Heuston Station: kNN Predictions and Actuals",
    xaxis_title="Time",
    yaxis_title="Available Bikes",
    legend_title="Legend",
    font=dict(
        family="Courier New, monospace",
        size=18,
        color="RebeccaPurple"
    )
)
hs_vis.add_trace(
    go.Scatter(name='10min Prediction',
           x=x_axis10,
           y=y_preds10))
hs_vis.add_trace(
    go.Scatter(name='10min Actual',
           x=x_axis10,
           y=y_actuals10))

hs_vis.add_trace(
    go.Scatter(name='30min Prediction',
           x=x_axis30,
           y=y_preds30))
hs_vis.add_trace(
    go.Scatter(name='30min Actual',
           x=x_axis30,
           y=y_actuals30))

hs_vis.add_trace(
    go.Scatter(name='60min Prediction',
           x=x_axis60,
```

```
        y=y_preds60))
hs_vis.add_trace(
    go.Scatter(name='60min Actual',
           x=x_axis60,
           y=y_actuals60))
<h2> kNN: Blessington Street </h2>
from sklearn.neighbors import KNeighborsRegressor


X = pd.to_datetime(ph_df['TIME'])
X = X.tolist()
y = ph_df['AVAILABLE BIKES'].values
y = y.reshape(-1,1)
neigh = 3

rmsle10 = []
rmsle30 = []
rmsle60 = []
test_size = 0.01
y_preds10 = []
y_actuals10 = []
x_axis10 = []

y_preds30 = []
y_actuals30 = []
x_axis30 = []


y_preds60 = []
y_actuals60 = []
x_axis60 = []



while test_size < 0.8:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= test_size, shuffle = False)

    # define model
    model = KNeighborsRegressor(n_neighbors=neigh)



    from numpy import array


    i = week_1
    j = 0
    rows = int (len(X_train) - week_1)
```

```
    X_list = [[0]*5]*rows
    y_list = []

    #create feature vector of sliding 5 min samples
    while i < y_train.size:
        X_list[j] = [y_train[i-min_5],y_train[i-min_5*2], y_train[i-hour_1*8],
y_train[i-day_1+min_5], y_train[i-week_1+min_5]]
        y_list.append(y_train[i])
        i+= 1
        j+= 1
    X_list = array(X_list)
    X_list  = X_list[:, :, 0]

    i = 0

    X_list = X_list.tolist()
    model.fit(X_list,y_list)



    while i < 12:
        temp = [y_list[-min_5],y_list[-min_5*2], y_list[-hour_1*8], y_list[-day_1+min_5],
y_list[-week_1+min_5]]
        temp = np.array(temp).reshape(1, -1)



        # # make a prediction
        yhat = model.predict(temp)
        yhat = int(yhat[0])

        if i == 1:
            y_preds10.append(yhat)
            y_actuals10.append(y_test[i][0])
            x_axis10.append(X_test[i])
        elif i == 4:
            y_preds30.append(yhat)
            y_actuals30.append(y_test[i][0])
            x_axis30.append(X_test[i])



        elif i == 11:
            y_preds60.append(yhat)
            y_actuals60.append(y_test[i][0])
            x_axis60.append(X_test[i])

        X_list.append([y_list[-1],y_list[-min_5*2], y_list[-hour_1*8], y_list[-day_1+min_5],
y_list[-week_1+min_5]])
```

```
        y_list.append(yhat)



        i += 1
    test_size += 0.01
    #mean_squared_error(y_actuals, y_preds)

rmsle10.append(mean_squared_log_error(y_actuals10, y_preds10))
rmsle30.append(mean_squared_log_error(y_actuals30, y_preds30))
rmsle60.append(mean_squared_log_error(y_actuals60, y_preds60))

print(rmsle10)
<h2> Evaluation </h2>
print(rmsle10)
print(rmsle30)
print(rmsle60)
hs_vis = px.scatter(title="s")
hs_vis.update_layout(
    title="Blessington Street: kNN Predictions and Actuals",
    xaxis_title="Time",
    yaxis_title="Available Bikes",
    legend_title="Legend",
    font=dict(
        family="Courier New, monospace",
        size=18,
        color="RebeccaPurple"
    )
)
hs_vis.add_trace(
    go.Scatter(name='10min Prediction',
           x=x_axis10,
           y=y_preds10))
hs_vis.add_trace(
    go.Scatter(name='10min Actual',
           x=x_axis10,
           y=y_actuals10))

hs_vis.add_trace(
    go.Scatter(name='30min Prediction',
           x=x_axis30,
           y=y_preds30))
hs_vis.add_trace(
    go.Scatter(name='30min Actual',
           x=x_axis30,
           y=y_actuals30))
```

```
hs_vis.add_trace(
    go.Scatter(name='60min Prediction',
               x=x_axis60,
              y=y_preds60))
hs_vis.add_trace(
    go.Scatter(name='60min Actual',
               x=x_axis60,
              y=y_actuals60))
```