Final Assignment predicting the occupancy of two Dublin bike stations

Part 1 of 2

Submission Date: 23/December/2021

Name: Valerie Brosnan

Student ID: 15316435

Datafile: Dublin Bikes open Data Q1 USAGE

Coding environment: Jupyter Notebook Python Cc

Code attached: Submitted you will find four jupyter notebooks:

1. processData
2. Functions
3. ridgeRegression
4. knnF

Please note in order to run the code processData must be run first then all notebooks can be run independently. Functions was used to explore the different trends in the data. To build the models Ridge regression and knnf can be run.

Processing of Dataset:

The datafile was loaded from the Dublin bikes 2020 Q1 usage data. This data included data from the 1/1/2020 to 1/4/2020.The locations selected were Pearse Street id number 32 and Portobello Road id number 43. These two stations were chosen as it was expected they would both have different trends and occupancy patterns. Pearse street being in the city centre was expected to be less predictable and it would have more regular use than the portobello station which is in a more residential/ commuters' area outside the city centre. I expected the two stations to be an interesting comparison as they are being used by different categories of users, they would both have different occupancy trends.

Data Reduction:

On initial inspection of the data, it was evident that the data recorded in January was quite sparse and incomplete see figures 1and 2. The effects of covid from March 11th can also be seen in figure 1 and 2 on the initial plot and the availability considerably drops and I considered this to not be a true depiction of the dataset. The data in January was further inspected in order to check if the number of available stands was recorded and perhaps the data could be calculated for the available bikes and thus the occupancy. However, no data was recorded during this interval. It would be possible to interpolate this data from values from the remainder of dataset however as there was such a large gap, this may introduce a bias in the training set and thus the data up to January 28th was omitted. The data used for training was then from January 28th to the 11th of March.

I also considered thinning out the data points to reduce having continuations of the same bike availability. The reason this was not applied is as the data is time series, I worried about the implications of this on my model. If recurring elements were removed and then returning to the time stamp on the previous day/ week wouldn't be possible if the data had been removed. If modelling the data again, thinning out the features would be something I would explore more in depth as I found the predictions appeared to follow a 'mode' of the dataset.

Plots of original Raw Data

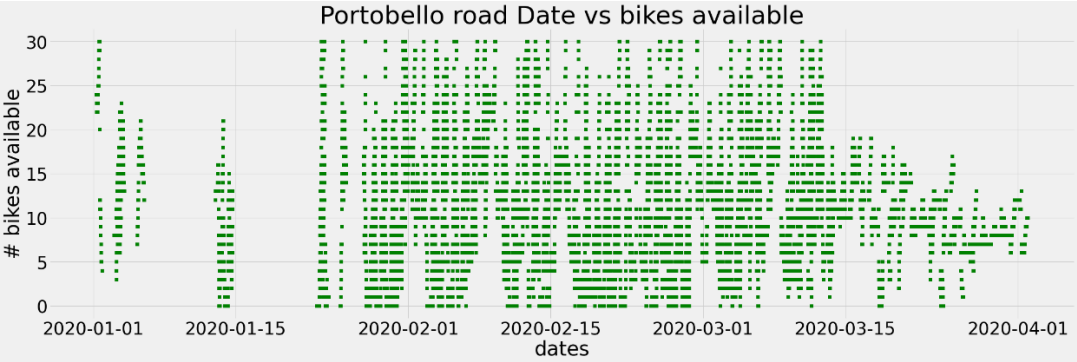

Figure 1: Graph of bike availability versus Date Portobello Road

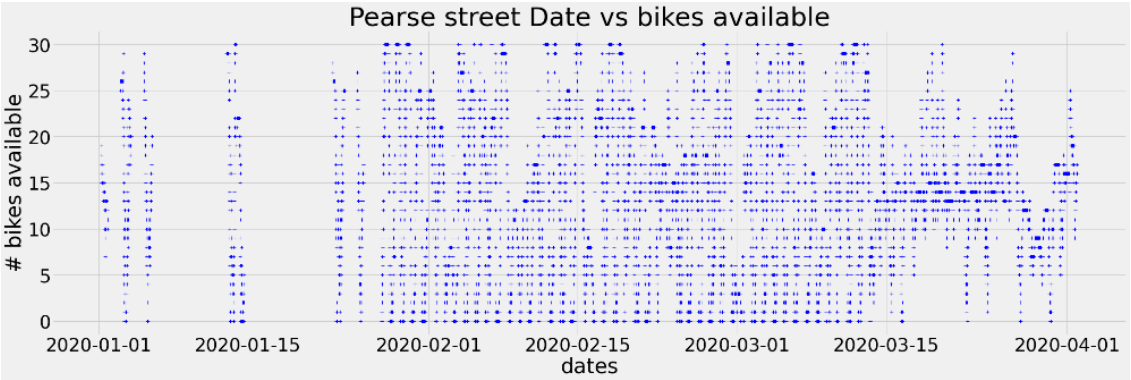

Figure 2: Graph of bike availability versus date Pearse street

The dataset was then reduced and to assist identifying trends the data was split into weekdays and weekends. As expected, both models follow different trends. The capacity of both stations is the same. Both datasets have a cyclical trend with a notable difference at the weekend. The weekday trend for both datasets is similar with the weekend pattern providing an interesting comparison. While both stations follow a cyclical pattern at midnight each weekday pearse streets occupancy is at maximum, while for the station on portobello road it tends to be at a minimum. The weekend usage of the pearse street station is much less predictable than that of portobello road.

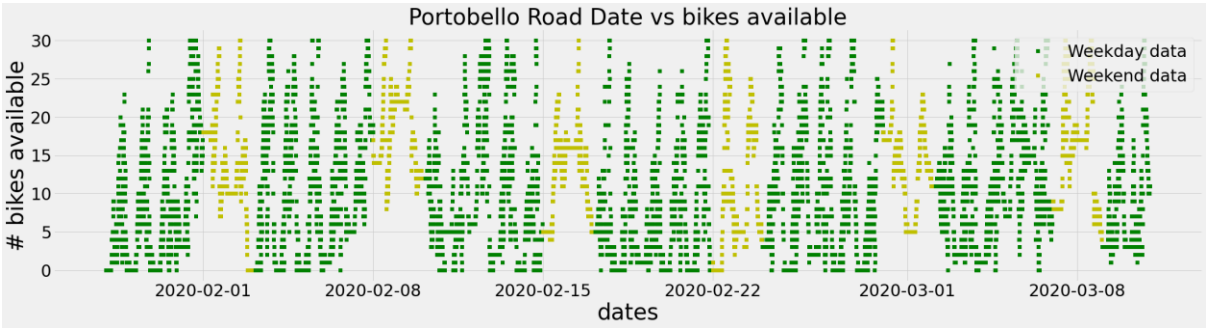

Figure 3: Portobello Road bike availability versus Dates reduced data

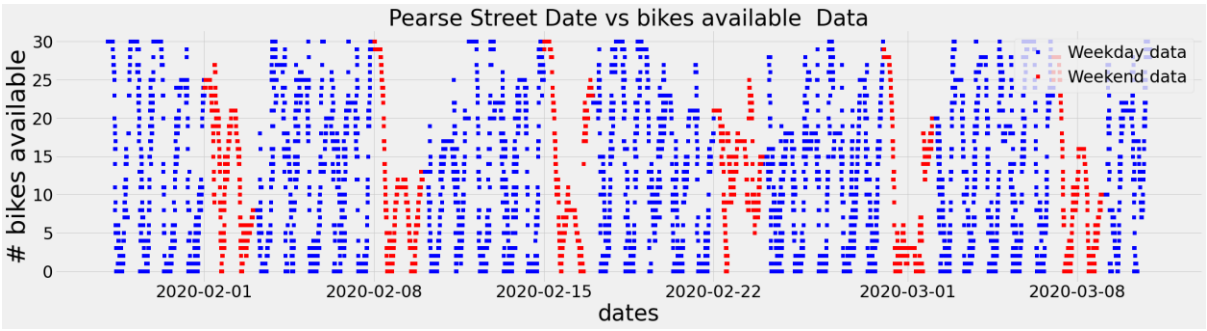

Figure 4: Pearse Street bike availability versus date weekday and weekend data

Building the model:

The data was modelled using short term trends, daily trends, weekly trends and a combination of these. When selecting from within these trends rather than looking at the entire dataset, the previous three points where selected to truncate the data. All four models were built considering each trend individually and considering all trends together.

Trend used:	10 Min Prediction	30 Min Prediction	1 Hour Prediction
Short term	0.0121	0.0121	0.0121
Daily	0.0016	0.0016	0.0016
Weekly	0.0007	0.0009	0.001

Table 1: Examining different trends Pearse data

Column1	10 Minutes MSE	20 Minutes MSE	1 Hour MSE
Weekly Feature	7.13	7.38	7.93
Combined features	5.28	5.29	5.3
Baseline	9.98	9.99	9.99

Table 2: Pearse model comparison

Note: The portobello dataset on testing demonstrated the same result that the combined features were best for performance. The Pearse dataset is used here to demonstrate the results.

This was simply modelled to compute the mean squared of the different trends to see which would be most appropriate by selecting the trend which minimizes the mse. It was immediately evident the weekly parameters were optimal this can be seen in table 1. On deeper inspection it was selected to model the data by building the model from a combination of all features and compare it with when weekly features were used. As seen in table 2 the combined features performance was best. The models selected were K nearest neighbour regressor and ridge regression. In ridge regression it could apply the weights to the parameters as to identify key features, while KNN will use the neighbouring elements to predict the target vector both models seemed appropriate for this data

Ridge Regression was selected so that the 'importance' of each feature could be identified. Ridge regression is a form of linear regression which helps to reduce overfitting by enforcing the simplest model possible. The model enforces L2 regularisation resulting in parameters which have little importance in determining the output tend towards zero. It achieves this by adding the highlighted additional coefficient to the loss function: $J(\emptyset) = \frac{1}{m}\sum_{i=0}^{m}(\emptyset^T x^{(i)} - y^{(i)})^2 + \frac{1}{C}\sum_{j=1}^{n}\emptyset_j^2$. Thus, the lower the parameter C the greater the applied penalty. This model was selected as it could apply an interesting insight into the features of the model, to identify key features. In this case the combined features were selected to be used where the model could apply weight to each feature as appropriate, whereas the other selected knn will tell us little about the features.

KNN

KNN was selected as it models its predictions directly from the input data. The output is predicted based on the existing training points around the input. This model was selected – although it is not desired for large datasets, for the reason predicts directly from the input data, I thought it would be appropriate for the model. There are two options to weight the neighbouring points – uniform and gaussian. If a uniform weight is applied all neighbouring training points are weighted equally $w = 1$, whereas if distance parameter is selected for weight $w = e^{-\gamma d(x^i,x)^2}$, so points further from the query point could be weighted less. The predicted output can then be calculated by

$$Y_{pred} = \sum_i w^i y^i \Big/ \sum_i w^i$$

Baseline

The baseline model selected always predicts the mean value of the station selected. It is used as a comparison to test how well the models behave compared to a very simple model.

The two metrics used to evaluate the performance of the regression models are the r2 score and the root mean squared error. Tuning parameters is carried out by varying the hyperparameter and selecting the lowest value for the RMSE $\sqrt{\frac{1}{m}\sum_{i=0}^{m}(\theta^T x^{(i)} - y^{(i)})^2}$.

**Please note: Throughout the report rmse is referenced as mse.

The r2 score: The optimal value for the r2 score is 1 when the model predicts perfectly. The r2 value of a model is 0 when it predicts the mode value. It quantifies the correlation between the predicted and actual output.

Ridge Regression:

Initially a range of c ($\alpha = {}^{1}/_{2c}$) was selected when training the ridge regression model, five-fold cross validation was then used in order to identify the value of C that would optimise the performance by resulting in the lowest mean squared error. The data was modelled considering all the short term, daily and weekly trend to build the feature vectors:

$$[y^{(k-3w)}, y^{(k-2w)}, y^{(k-w)}, y^{(k-3d)}, y^{(k-2d)}, y^{(k-d)}, y^{(k-3-q)}, y^{(k-2-q)}, y^{(k-1-q)}]$$

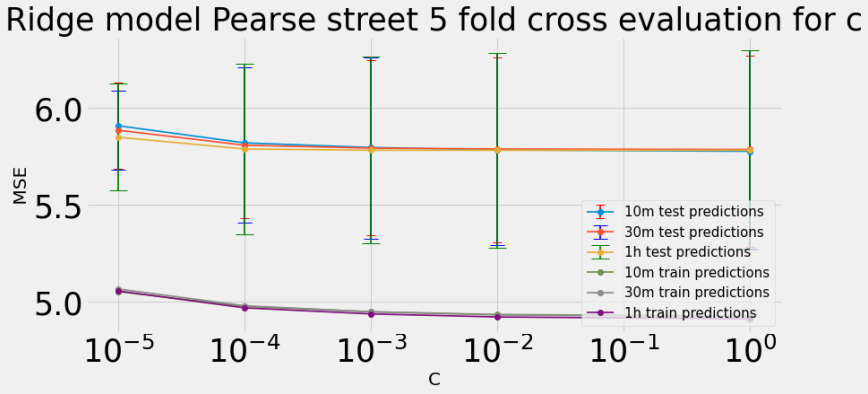

Figure 4: 5-fold cross validation to select optimal c of mse versus c

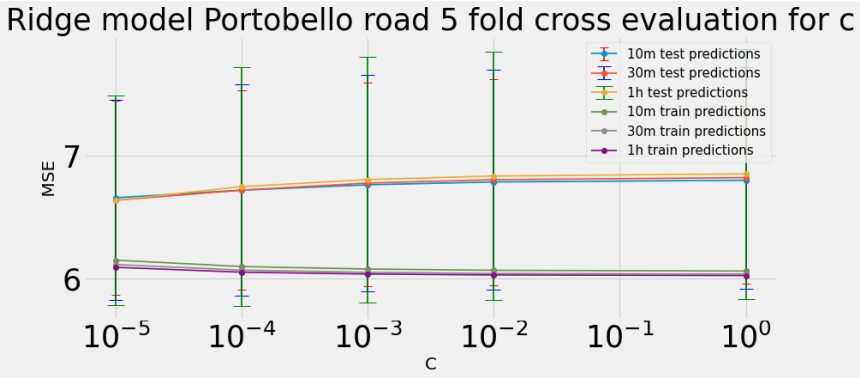

Figure 5: 5-fold cross validation for c versus mse

What is immediately evident from the graph is that overfitting is occurring for the model on both datasets figure 4 while initially varying the increasing c results in a decreased error further increases have a negligible effect on the error while causing the standard deviation to increase. The model is

preforming better on the training data than on the test data demonstrating overfitting occurs for the model. For the portobello dataset in figure 5 the mean squared error observed is higher than that seen for the Pearse street data. The portobello and Pearse street data follow a different trend as the value of c is increased. In figure 5 for a low value of c, when the penalty applied is at its largest the mean squared error of the testing data is at its lowest. The opposite is seen in figure 4 when a small value of c has a greater error in both training and testing data. Often graphically local minimum or difficult to identify therefore grid search with 5-fold cross validation was used to identify the optimal results for each prediction. The results can be found in table XX in evaluation of results:

The test data predicted available bikes was plotted against time for the test data using the optimal parameters.

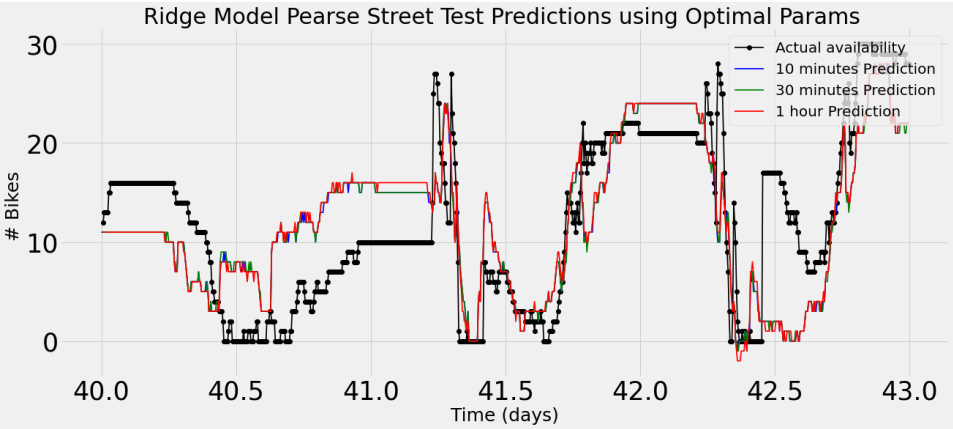

Figure 6: Pearse Street Data ridge model

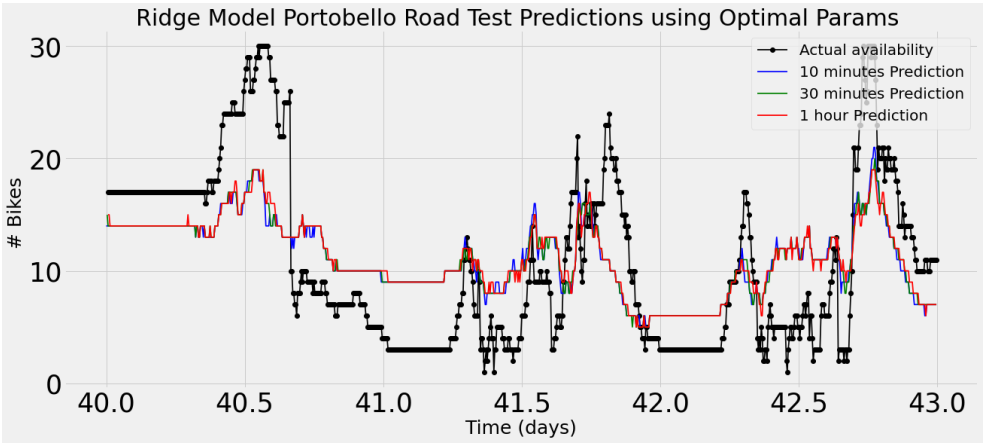

Figure 7: Portobello Road Data Ridge model

Figure 6 reflects the Pearse street data. The parameters were weighted as shown in table 3.

Pearse	$y^{(k-3w)}$	$y^{(k-2w)}$	$y^{(k-w)}$	$y^{(k-3d)}$	$y^{(k-2d)}$	$y^{(k-d)}$	$y^{(k-3-q)}$	$y^{(k-2-q)}$	$y^{(k-1-q)}$
10min	0.0508	0.1189	0.114	0.0508	0.0862	0.0472	0.0544	0.0518	0.0508
30 min	0.0567	0.1186	0.113	0.0567	0.0846	0.0465	0.0513	0.0499	0.0504
1 hour	0.0663	0.1208	0.116	0.0663	0.0823	0.0459	0.0443	0.0439	0.0457

Table 3: Pearse data model coefficients

As expected the previous weeks were most heavily weighted. In order to achieve a lower error the features should have been fit to polynomial features to optimise the model. The previous weeks have the most influence on the model. What its evident on both figures 6 and 7 is that the there is not a huge deviation in the prediction in 10 minutes ,20 minutes and 30 minutes. The model does not respond well when the availability of bikes rapidly changes.

The parameters for the portobello road were weighted as shown in table 4 and as seen for the Pearse street station the previous weeks are assigned the majority of the weight. I modelled on the entire dataset to observe weekly trend, the entire dataset was used in the model. This could also contribute to the contribution of the weekly data as the weekday and weekend data should be modelled separately to improve performance. Table 4 demonstrates that the previous week and two weeks previous have the largest weight, and greatest influence on each prediction. Figure 7 demonstrates that the ridge model does respond more to the changes in the actual data however everything appears 'scaled' to the mode of the data as it never predicts full /empty bike availability.

Portobello	$y^{(k-3w)}$	$y^{(k-2w)}$	$y^{(k-w)}$	$y^{(k-3d)}$	$y^{(k-2d)}$	$y^{(k-d)}$	$y^{(k-3-q)}$	$y^{(k-2-q)}$	$y^{(k-1-q)}$
10min	0.0468	0.3909	0.325	0.0468	0.2428	-.1153	-0.0868	0.031	0.0468
30 min	0.0677	0.397	0.335	0.0677	0.2332	-.1086	-0.0301	-.0231	-.0084
1 hour	0.0758	0.4041	0.315	0.0758	0.2518	-.1093	-0.0780	-.0409	0.0356

Table 4: Portobello Road Ridge model coefficients

KNN model

The second model investigated was knn. In order to select the optimal number of neighbours cross validation was used in order to select the optimal number of neighbours. The range of neighbours selected to investigate was from 1-20. Another factor to select between is the weight parameter to apply to neighbour values. As the data was not thinned out, applying the distance weight parameter may assign excess weight to the neighbouring elements which are the same value.

Portobello model.

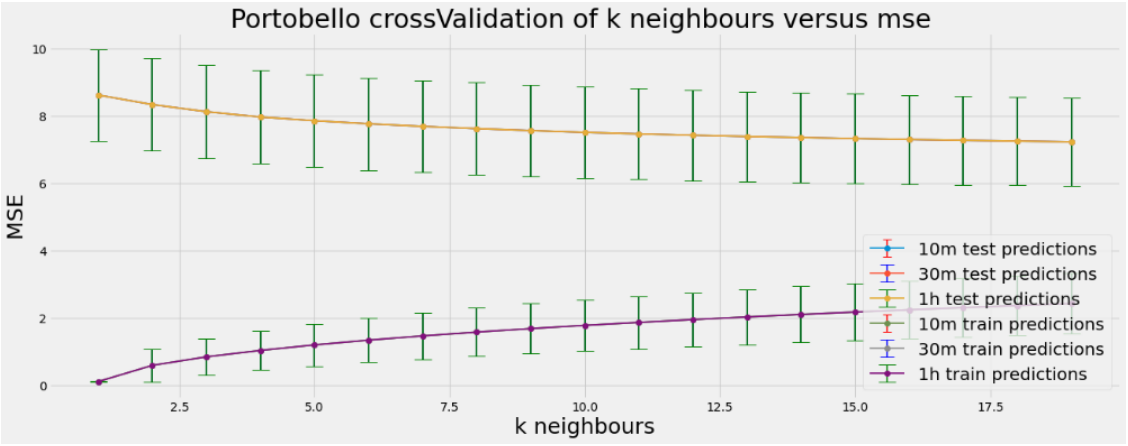

Figure 8: 5fold cross validation uniform weight parameter for knn

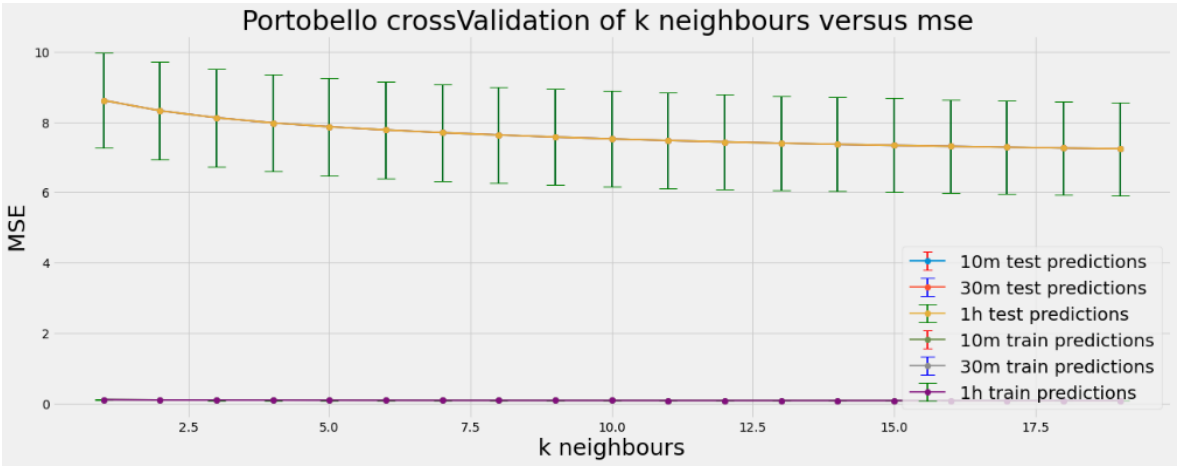

Figure 9: 5fold cross validation distance weight parameter for knn

Figures 8 and 9 demonstrate that again the KNN model overfits to the test data. They also show as the value for the number of neighbours increase the mse initially decreases before it begins to plateaue. There is no difference in mean squared error between the 10 minute ,30 minute and 1 hour predictions. As K increases the overfitting decreases however then begins to plateau. When a unifrom weight is applied the difference between the mse of the test and training data decreases. However this is because the model becomes 'worse' at predicting on the test data. For increasing k inboth figure 8 and 9 the mse decresases. Grid search using 5 fold cross validation was carried out to identify the optimal number of neighbours and weighting parameters. Test predicted mean squared error is slightly lower for the portobello model when the weight parameter unifrom was used. This is not as expected but as the difference is so small between the two and they are evidently poor models I wouldn't consider this significantly. The optimal model selected was thus k=19 with a uniform distance weight parameter.

This found that the model preforms significantly better on the training data. The model is overfitting to the training data this can be seen by the high r2 value on training and the low r2 on testing and reinforced with the large deviation in mse between test and training data. In order to avoid overfitting more data could be added. Other methods to attempt to reduce this was using 5 fold cross validation when training the model.

The same process was then calculated on the pearse dataset in order to identify the optimal hyperparameters. Similar results were observed as seen in figures 9 and 10 with a lower initial MSE.

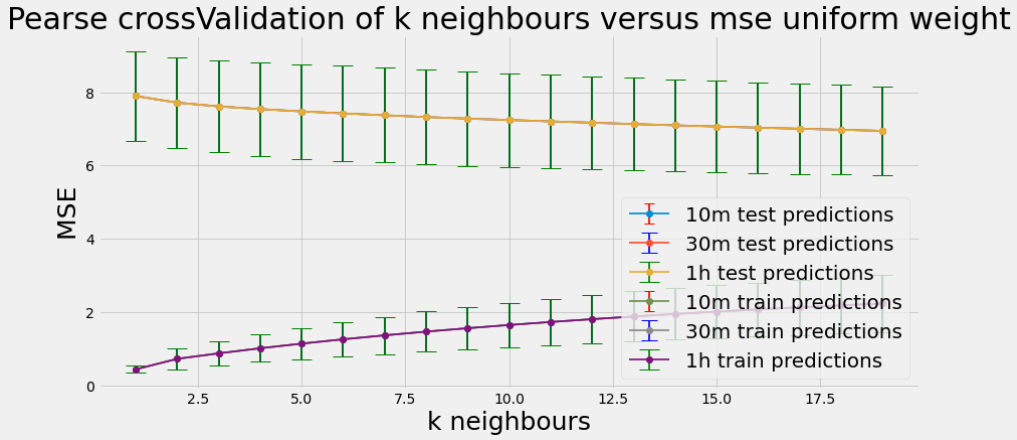

Figure 10: 5 fold crossvalidation on pearse data

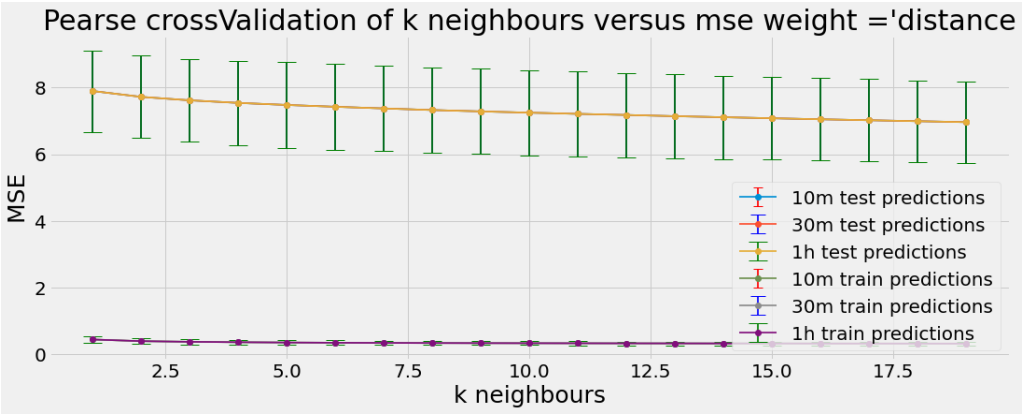

Figure 11: 5 fold crossvalidation on pearse data

The models again overfit to the training data, when a weight = disitance there is very little fluctuation in the error as k increases. All future time predictions result in the same value which again is a bad indication of the model. When a unifrom weight is assigned as seen on the portobello dataset with a increasing number of neighbours the models test error increases. In order to distinguish the best parameters. Grid search was again applied. To identify the optimal parameters, visually it is easier to identify in figures 8-10 than when analysing the ridge plots as there is no difference between the future predictions. A larger K is favoured as (although marginal) the mse is lower. The resulting optimal values were plotted for both datasets.

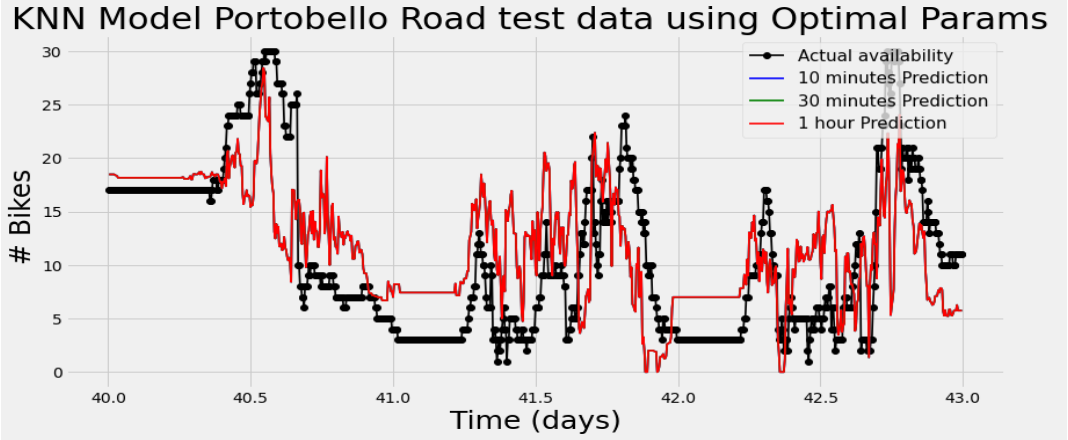

Figure 11: Knn predictions k =19

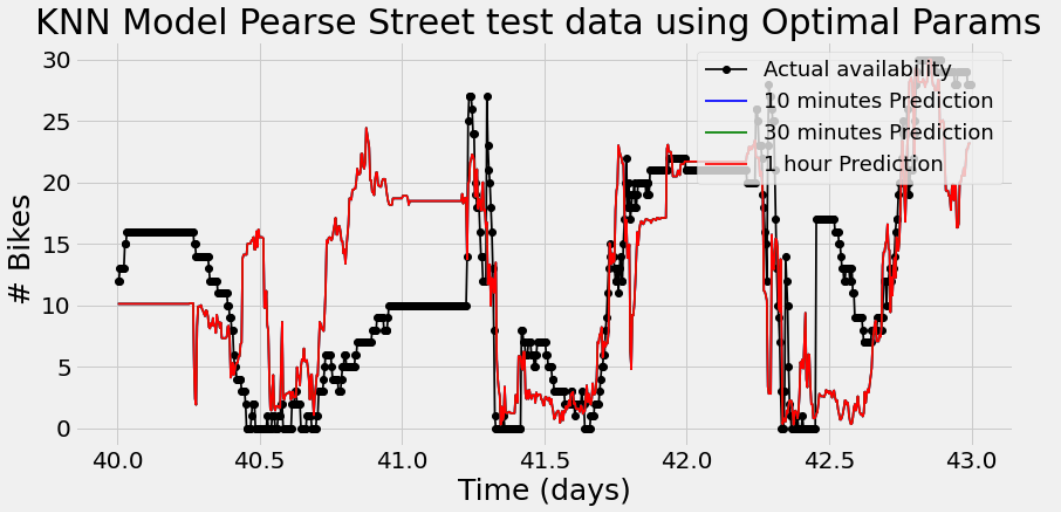

Figure 12: KNN predictions k =19

Comparison between ridge predictions:

Obviously independent of how long in the future you are requesting the availability the knn model predicts the same value, however the difference is also not large for future predictions between the time interval for the ridge model. For the Pearse street station the knn model is clearly a worse fit and the graphs is scattered and unpredictable. The model tends to 'overreact' to outliers0 event of a small magnitude. For portobello road unlike with the ridge model the graph does not appear to be a scaled version of the input. The knn model does manage to predict full and minimum occupancy. However similarly to the knn model for the Pearse street data the graph is hectic and does not reflect the true availability. Visually it is evident that neither model preforms very well.

Evaluation of models:

Pearse Street:

Time prediction	Model	MSE_{train}	MSE_{test}	$R2_{train}$	$R2_{test}$
10 minutes	Ridge c =10^{-4}	5.0543	6.1424	0. 7583	0.4899
10 minutes	KNN	3.0767	7.0376	0.9105	0.3313
10 minutes	Baseline	10.2842	8.6374	0	0
30 minutes	Ridge c = 1	5.050	6.2025	0.7588	0.4806
30 minutes	KNN	3.0767	7.0376	0.9105	0.3313
30 minutes	Baseline	10.2842	8.6374	0	0
1 hour	Ridge c =1	5.035	6.2330	0.7605	0.476
1 hour	KNN	3.0767	7.0376	0.9105	0.3313
1 hour	Baseline	10.2842	8.6374	0	0

Table 5: Results for Pearse street data

The increase in the MSE from the training to test data and the decrease in the r2 score indicates that all of the models overfit to the training dataset. In training the knn appears to be the better model for all time predictions with the low training error and a r2 value which would nearly be considered ideal. However, on the test data which is a better representation of how the model preforms this is not the case and the ridge model is found to be optimal. Both models outperform the baseline

model which predicts the mode of the dataset. The best model is the Ridge model for each prediction as it results in both the lowest error and the highest r2 value.

Portobello Road:

Time prediction	Model	MSE_{train}	MSE_{test}	$R2_{train}$	$R2_{test}$
10 minutes	Ridge c=1	6.4433	5.6999	0. 3471	0.4473
10 minutes	KNN	3.5233	6.1319	0.8046	0.3607
10 minutes	Baseline	7.9668	7.6715	0	0
30 min	Ridge c=1	6.4289	5.6722	0.3492	0.4529
30 minutes	KNN	3.5233	6.1319	0.8046	0.3607
30 minutes	Baseline	7.9668	7.6715	0	0
1 hour	Ridge c=1	6.4433	5.7153	0.3495	0.4547
1 hour	KNN	3.5233	6.1319	0.8046	0.3607
1 hour	Baseline	7.9668	7.6715	0	0

Table 6: Results for portobello road data

The ideal performing classifier for the portobello road station is the ridge model again with a c parameter of 1. This is the only model who's performance is better on the test data than on training data, indicating the model is not overfitting the data. The same is seen for the r2 score as it improves from the test to training set, However the r2 values are still low indicating a poor correlation between the predicted y and actual value. All models designed preform better than the baseline demonstrating they preform better than a model which knows continually predicts the mode availability. As seen in the cases of the Pearse Street station the knn model preforms better on test data. This is reflected with a high value for the r2 score and a low training error, but this performance is not seen on the testing data.

Conclusion and Discussion:

The optimal models for each dataset was found to be ridge regression. The regularisation incorporated in the ridge model contributes to its success. In order to further improve this model further feature analysis using polynomial features should have been carried out. The ridge model for the Portobello Road station is the only model which was found to not overfit the data with testing metrics outperforming the training values. Knn was selected as I thought using neighbouring training points would help with the prediction. On reflection it was not a good model as finding the nearest neighbour is computationally expensive and this model took much longer than the ridge and baseline model to train. Rather than select KNN a decision tree may have been used to increase the computation time. In my initial processing of the dataset, I separated days and weekends this is something I should have further explored when modelling the data.

On reflection of the assignment, I do not think the models selected were optimal for the time series prediction, this is also reflected in the metrics. I would explore further training a neural net model as well as adding polynomial features or investigating applying the kernel trick to the features. Another way to approach the model would be to include a feedback model. This could be approached by training a recurrent neural network with a LSTM block after the convolutional layers.

Code Appendix

Please note that not all functions used to plot the data are not included to minimise the length of the code. These can be found in the jupyter notebook attached -Also note code is not sceenshot is simply cells copied and pasted.

PROCESS DATA

```
import pandas as pd
import numpy as np
import math , sys
import matplotlib.pyplot as plt


from sklearn.linear_model import Ridge
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold

from sklearn.metrics import mean_squared_error
plt.rc('font',size=18)
plt.rcParams['figure.constrained_layout.use'] = True
plt.rcParams.update({'font.size':22})
plt.style.use('fivethirtyeight')
plt.rcParams['lines.linewidth'] = 1.7
import datetime
pearseID = 32
portobelloID =43
df =pd.read_csv("dublinbikes_20200101_20200401.csv")
df.head()

pearseRaw = df[df.iloc[:,0]==pearseID]
pearseRaw.head()
portobelloRaw =df[df.iloc[:,0]==portobelloID]
portobelloRaw.head()
portobelloRaw.to_csv("portobelloRaw.csv",index=False)
pearseRaw.to_csv("pearseRaw.csv",index=False)
pearse=pd.read_csv("portobelloRaw.csv",usecols=[1,6],parse_dates=[1])
portobello = pd.read_csv("pearseRaw.csv",usecols=[1,6],parse_dates=[1])

## convert the data to the correct type
pearse.iloc[:,0] = pd.to_datetime(pearse.iloc[:,0])
pearse.iloc[:,1] = pearse.iloc[:,1].astype(str).astype(int)

portobello.iloc[:,0] = pd.to_datetime(portobello.iloc[:,0])
portobello.iloc[:,1] = portobello.iloc[:,1].astype(str).astype(int)
plt.plot(pearse.iloc[:,0],pearse.iloc[:,1],'+',color='blue')
```

```
def removeData(df):
   start = pd.to_datetime("28-01-2020",format='%d-%m-%Y')
   end = pd.to_datetime("11-03-2020", format='%d-%m-%Y')
   temp =df[df.iloc[:,0]>=start]
   df = temp[temp.iloc[:,0]<=end]

   return df
pearseRed = removeData(pearse)
portobelloRed = removeData(portobello)

def applyFeatures(df):
    df['dayNum'] = df.iloc[:,0].dt.dayofweek
    df['dataType'] = np.where(df['dayNum']<=4, 'wday',
(np.where(df['dayNum']==5,'Saturday','Sunday')))

    return df


def seperateData(df):
    weekDay = df[df.iloc[:,3]=='wday']

    weekEnd = df[df.iloc[:,3] != 'wday']

    return(weekDay,weekEnd)
pearseWithDays = applyFeatures(pearseRed)
pearseWithDays.head()
portobelloWithDays = applyFeatures(portobelloRed)
pearse1,pearse2 = seperateData(pearseWithDays)
pearse1.head()
portobello1,portobello2 = seperateData(portobelloWithDays)
pearseWithDays.head()


def convertDays(df):
    convertSec = 1000000000

    tFull =
((pd.DatetimeIndex(df.iloc[:,0]).astype(np.int64))/convertSec).values
    dt = tFull[1] -tFull[0]
    print(dt)

    t = (tFull-tFull[0]) /60 /60/24
    #df.iloc[:,4]=t.tolist()
    return t ,dt
pearse1.to_csv("pearseWeekdays.csv",index=False)
pearse2.to_csv("pearseWeekEnds.csv",index=False)
pearseWithDays.to_csv("pearseProcessed.csv",index = False)
portobello1.to_csv("portobelloWeekdays.csv",index=False)
```

```
portobello2.to_csv("portobelloWeekEnd.csv",index=False)
portobelloWithDays.to_csv("portobelloProcessed.csv",index=False)

pearse1.iloc[:,1]
```

Ridge regression model

```
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
from sklearn.linear_model import Ridge
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import mean_squared_error
from sklearn.dummy import DummyRegressor
from sklearn.model_selection import GridSearchCV

plt.rc('font',size=18)
plt.rcParams['figure.constrained_layout.use'] = True
plt.rcParams.update({'font.size':22})
plt.style.use('fivethirtyeight')
plt.rcParams['lines.linewidth'] = 1.7
#%matplotlib inline
pearse =pd.read_csv("pearseProcessed.csv",usecols=[0,1,3],parse_dates=[0])
portobello =
pd.read_csv("portobelloProcessed.csv",usecols=[0,1,3],parse_dates=[0])

def convertDays(df):
    convertSec = 1000000000
    # tFull =
pd.array(pd.DatetimeIndex(df.iloc[:,0]).astype(np.int64))/convertSec
    tFull =
((pd.DatetimeIndex(df.iloc[:,0]).astype(np.int64))/convertSec).values
    dt = tFull[1] -tFull[0]
    print(dt)

    t = (tFull-tFull[0]) /60 /60/24
    #df.iloc[:,4]=t.tolist()
    return t ,dt

secondsPearse,dt = convertDays(pearse)
pearse['days']=secondsPearse.tolist()
portobello['days']= secondsPearse.tolist()
```

```
def BuildModel(q,y,t):
    lag=3
    stride=1

    w=math.floor(7*24*60*60/dt) # number of samples per week
    len =y.size-w-lag*w-q
    XX=y[q:q+len:stride]
    for i in range(1,lag):
        X=y[i*w+q:i*w+q+len:stride]
        XX=np.column_stack((XX,X))

    d=math.floor(24*60*60/dt) # number of samples per day

    for i in range(0,lag):
        X=y[i*d+q:i*d+q+len:stride]
        XX=np.column_stack((XX,X))

    for i in range(0,lag):
        X=y[i:i+len:stride]
        XX=np.column_stack((XX,X))

    yy=y[lag*w+w+q:lag*w+w+q+len:stride]
    tt=t[lag*w+w+q:lag*w+w+q+len:stride]

    yy.reset_index(drop=True, inplace=True)
    tt.reset_index(drop=True, inplace=True)
    return tt,yy,XX
qVal=[2,6,12]
stride = 1
lag = 3
#y =result.iloc[:,1]
y= portobello.iloc[:,1]
t=portobello.iloc[:,3]
alphaRange = [.00001, 0.0001,0.001, 0.01,1]

def crossEvalAlpha(q,y,t):
    tt,yy,XX = BuildModel(q,y,t)
    train, test =
train_test_split(np.arange(0,yy.size),test_size=0.2,shuffle=False)
    MSE =[]; meanE=[];stdE=[]
    temp =[]; meanETrain =[]; stdETrain=[]
    cv = TimeSeriesSplit(n_splits=5)
    for a in alphaRange:
        for train,test in cv.split(XX):
            model
=Ridge(fit_intercept=False,alpha=1/(2*a)).fit(XX[train],yy[train])
            yPred = model.predict(XX[test])
```

```
            yPredTrain = model.predict(XX[train])
            yp2 =np.rint(yPred)
            MSE.append(mean_squared_error(yp2,yy[test]))
            temp.append(mean_squared_error(yPredTrain,yy[train]))
            #print(np.sqrt(MSE))
        meanE.append(np.array(np.sqrt(MSE)).mean())
        meanETrain.append(np.array(np.sqrt(temp)).mean())
        #print(meanE)
        stdE.append(np.array(np.sqrt(MSE)).std())
        stdETrain.append(np.array(np.sqrt(temp)).std())
    return meanE,stdE,meanETrain,stdETrain


def
printCrossEval(m10,std10,m30,std30,mean1,std1,m0,s0,m1,s1,m2,s2,alpha,TITLE):
    plt.title(TITLE,size=35)
    plt.rcParams['figure.figsize']=[9,7]
    plt.xlabel("C",size=20)
    plt.ylabel(" MSE ",size=20)
    plt.errorbar(alpha,m10,yerr=std10,fmt="-o",ecolor="r",capsize=5)
    plt.errorbar(alpha,m30,yerr=std30,fmt="-o",ecolor="b",capsize=8)
    plt.errorbar(alpha,mean1,yerr=std1,fmt="-o",ecolor="g",capsize=10)
    ## plot training results
    plt.errorbar(alpha,m0,fmt="-o",ecolor="r",capsize=5)
    plt.errorbar(alpha,m1,fmt="-o",ecolor="b",capsize=8)
    plt.errorbar(alpha,m2,fmt="-o",ecolor="g",capsize=10)
    plt.xscale('log')
    plt.legend(["10m test predictions","30m test predictions","1h test
predictions","10m train predictions","30m train predictions","1h train
predictions "],loc='lower right',fontsize=15)
    plt.rcParams['figure.figsize']=[12,6]

def gridridge(q,y,t):
  tt,yy,XX = BuildModel(q,y,t)
  train, test =
train_test_split(np.arange(0,yy.size),test_size=0.2,shuffle=False)

  model = Ridge()
  cv =TimeSeriesSplit(n_splits=5)
  param = {
      'alpha':[.00001, 0.0001,0.001, 0.01,1], #np.arange(0, 1.001, 0.01)
      'fit_intercept':[True,False],
      'normalize':[True,False],
  'solver':['auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga']
        }

  search = GridSearchCV(Ridge(), param, scoring='r2', n_jobs=-1, cv=cv)
  result = search.fit(XX[train], yy[train])
```

```
  print('Best Model: \n')
  best=result.best_params_

  model=Ridge(**result.best_params_).fit(XX[train], yy[train])
  yTrain = yy[train]
  yTest = yy[test]
  yTemp = np.rint(model.predict(XX[train]))
  yPred = np.rint(model.predict(XX[test]))

  #metrics
  mse_tr=metrics.mean_squared_error(yTrain, yTemp)
  mse_te=metrics.mean_squared_error(yTest, yPred)
  r2_tr=metrics.r2_score(yTrain, yTemp)
  r2_te=metrics.r2_score(yTest, yPred)

  print('Training set mse: ',mse_tr )
  print('Test set mse: ', mse_te)
  print('Training set r2: ', r2_tr)
  print('Test set r2: ', r2_te)
  print(model.intercept_, model.coef_)

  return mse_tr,mse_te,r2_tr,r2_te, best,tt,yTrain,yTest,yTemp,yPred

meanE0,stdE0,m0,s0 = crossEvalAlpha(qVal[0],y,t)
meanE1,stdE1,m1,s1 = crossEvalAlpha(qVal[1],y,t)
meanE2,stdE2,m2,s2 = crossEvalAlpha(qVal[2],y,t)
print(meanE0)
print("----------------------------------------")
title = "Ridge model Portobello road 5 fold cross evaluation for c"
printCrossEval(meanE0,stdE0,meanE1,stdE1,meanE2,stdE2,m0,s0,m1,s1,m2,s2,alphaR
ange,title)

mse_tr_q1,mse_te_q1,r2_tr_q1,r2_te_q1,best_q1,tt1,y_train,y_test,y_hat_q1,y_pr
ed_q1=gridridge(qVal[0],y,t)
mse_tr_q2,mse_te_q2,r2_tr_q2,r2_te_q2,best_q2,tt2,y_train,y_test,y_hat_q2,y_pr
ed_q2=gridridge(qVal[1],y,t)
mse_tr_q3,mse_te_q3,r2_tr_q3,r2_te_q3,best_q3,tt3,y_train,y_test,y_hat_q3,y_pr
ed_q3=gridridge(qVal[2],y,t)
q1_df=pd.DataFrame.from_dict(best_q1, orient='index',columns=['q=2'])
q2_df=pd.DataFrame.from_dict(best_q2, orient='index',columns=['q=6'])
q3_df=pd.DataFrame.from_dict(best_q3, orient='index',columns=['q=12'])
df_best_rd=pd.concat([q1_df,q2_df,q3_df],axis=1)

m1_q1=pd.DataFrame(data=[mse_tr_q1,mse_te_q1,r2_tr_q1,r2_te_q1],
                   index=['mse_train','mse_test','r2_train','r2_test'],columns
=['q=2'])
m2_q2=pd.DataFrame(data=[mse_tr_q2,mse_te_q2,r2_tr_q2,r2_te_q2],
```

```
                    index=['mse_train','mse_test','r2_train','r2_test'],columns
=['q=6'])
m3_q3=pd.DataFrame(data=[mse_tr_q3,mse_te_q3,r2_tr_q3,r2_te_q3],
                    index=['mse_train','mse_test','r2_train','r2_test'],columns
=['q=12'])
df_metrics_rd=pd.concat([m1_q1,m2_q2,m3_q3],axis=1)

df_params=pd.concat([df_best_rd,df_metrics_rd],axis=0)

df_params


## bASELIN MODEL
for i in qVal:
    tt,yy,XX = BuildModel(i,y,t)
    train, test =
train_test_split(np.arange(0,yy.size),test_size=0.2,shuffle=False)
    MSE =[]; meanE=[];stdE=[]
    dummyReg = DummyRegressor(strategy="mean")
    cv = TimeSeriesSplit(n_splits=5)
    for train,test in cv.split(XX):
        dummyReg.fit(XX[train],yy[train])
        yPred =dummyReg.predict(XX)
        MSE.append(mean_squared_error(yPred,yy))
    meanE.append(np.array(np.sqrt(MSE)).mean())
    print("q",i,"Mean squared error",meanE)



bikes = pearse.iloc[:,1]
time = pearse.iloc[:,3]
meanE0,stdE0,m0,s0 = crossEvalAlpha(qVal[0],bikes,t)
meanE1,stdE1,m1,s1 = crossEvalAlpha(qVal[1],bikes,t)
meanE2,stdE2,m2,s2 = crossEvalAlpha(qVal[2],bikes,t)

print("----------------------------------------")
title = "Ridge model Pearse street 5 fold cross evaluation for c"
printCrossEval(meanE0,stdE0,meanE1,stdE1,meanE2,stdE2,m0,s0,m1,s1,m2,s2,alphaR
ange,title)


mse_tr_q1,mse_te_q1,r2_tr_q1,r2_te_q1,best_q1,tt1,y_train,y_test,y_hat_q1,y_pr
ed_q1=gridridge(qVal[0],bikes,time)
mse_tr_q2,mse_te_q2,r2_tr_q2,r2_te_q2,best_q2,tt2,y_train,y_test,y_hat_q2,y_pr
ed_q2=gridridge(qVal[1],bikes,time)
mse_tr_q3,mse_te_q3,r2_tr_q3,r2_te_q3,best_q3,tt3,y_train,y_test,y_hat_q3,y_pr
ed_q3=gridridge(qVal[2],bikes,time)
```

```
q1_df=pd.DataFrame.from_dict(best_q1, orient='index',columns=['q=2'])
q2_df=pd.DataFrame.from_dict(best_q2, orient='index',columns=['q=6'])
q3_df=pd.DataFrame.from_dict(best_q3, orient='index',columns=['q=12'])
df_best_rd=pd.concat([q1_df,q2_df,q3_df],axis=1)

m1_q1=pd.DataFrame(data=[mse_tr_q1,mse_te_q1,r2_tr_q1,r2_te_q1],
                   index=['mse_train','mse_test','r2_train','r2_test'],columns
=['q=2'])
m2_q2=pd.DataFrame(data=[mse_tr_q2,mse_te_q2,r2_tr_q2,r2_te_q2],
                   index=['mse_train','mse_test','r2_train','r2_test'],columns
=['q=6'])
m3_q3=pd.DataFrame(data=[mse_tr_q3,mse_te_q3,r2_tr_q3,r2_te_q3],
                   index=['mse_train','mse_test','r2_train','r2_test'],columns
=['q=12'])
df_metrics_rd=pd.concat([m1_q1,m2_q2,m3_q3],axis=1)

df_params=pd.concat([df_best_rd,df_metrics_rd],axis=0)
df_params
def Baseline(bikes,time):
    for i in qVal:
        tt,yy,XX = BuildModel(i,bikes,time)
        train, test =
train_test_split(np.arange(0,yy.size),test_size=0.2,shuffle=False)
        MSE =[]; meanE=[];r2Val =[]
        dummyReg = DummyRegressor(strategy="mean")
        cv = TimeSeriesSplit(n_splits=5)
        for train,test in cv.split(XX):
            dummyReg.fit(XX[train],yy[train])
            yPred =dummyReg.predict(XX[train])
            MSE.append(mean_squared_error(yPred,yy[test]))
        meanE.append(np.array(np.sqrt(MSE)).mean())
        print("q",i,"Mean squared error",meanE)

Baseline(portobello.iloc[:,1],pearse.iloc[:,3])
```

KNN Model:

```
import pandas as pd
import numpy as np
import math , sys
import matplotlib.pyplot as plt
from sklearn.neighbors  import KNeighborsRegressor
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.metrics import mean_squared_error ,r2_score
from sklearn.dummy import DummyRegressor

plt.rc('font',size=18)
plt.rcParams['figure.constrained_layout.use'] = True
plt.rcParams.update({'font.size':22})
plt.style.use('fivethirtyeight')
plt.rcParams['lines.linewidth'] = 1.7
pearse =pd.read_csv("pearseProcessed.csv",usecols=[0,1,3],parse_dates=[0])
portobello =
pd.read_csv("portobelloProcessed.csv",usecols=[0,1,3],parse_dates=[0])
def convertDays(df):
    convertSec = 1000000000
    # tFull =
pd.array(pd.DatetimeIndex(df.iloc[:,0]).astype(np.int64))/convertSec
    tFull =
((pd.DatetimeIndex(df.iloc[:,0]).astype(np.int64))/convertSec).values
    dt = tFull[1] -tFull[0]
    print(dt)

    t = (tFull-tFull[0]) /60 /60/24
    #df.iloc[:,4]=t.tolist()
    return t ,dt
secondsPearse,dt = convertDays(pearse)
pearse['days']=secondsPearse.tolist()
portobello['days']= secondsPearse.tolist()
lag = 3
y = portobello.iloc[:,1]
t = portobello.iloc[:,3]
qVal=[2,6,12]
def BuildModel(q,y,t):
    stride=1
    q=6
    w=math.floor(7*24*60*60/dt) # number of samples per week
    len =y.size-w-lag*w-q
    XX=y[q:q+len:stride]
    for i in range(1,lag):
        X=y[i*w+q:i*w+q+len:stride]
        XX=np.column_stack((XX,X))

    d=math.floor(24*60*60/dt) # number of samples per day

    for i in range(0,lag):
        X=y[i*d+q:i*d+q+len:stride]
        XX=np.column_stack((XX,X))

    for i in range(0,lag):
        X=y[i:i+len:stride]
```

```
        XX=np.column_stack((XX,X))

    yy=y[lag*w+w+q:lag*w+w+q+len:stride]
    tt=t[lag*w+w+q:lag*w+w+q+len:stride]

    yy.reset_index(drop=True, inplace=True)
    tt.reset_index(drop=True, inplace=True)
    return tt,yy,XX
def KnnModel(q,bikes,time,weight):
    tt,yy,XX = BuildModel(q,bikes,time)
    train, test = train_test_split(np.arange(0,yy.size),test_size=0.2)
    mse =[];std=[];temp=[]
    print("MSE HERE SHOULD BE EMPTY",mse)
    for k in range(1,20):
        model =
KNeighborsRegressor(n_neighbors=k,weights=weight).fit(XX[train],yy[train])
        yPred = model.predict(XX[test])
        temp.append(mean_squared_error(yPred,yy[test]))
        mse.append(np.array(temp).mean())
        std.append(np.array(temp).std())
    #print("MSE".)
    return mse ,std
def
printCrossEval(m10,std10,m30,std30,mean1,std1,m0,s0,m1,s1,m2,s2,alpha,TITLE):
    plt.title(TITLE,size=30)
    plt.rcParams['figure.figsize']=[15,6]
    plt.rc('xtick',labelsize=20)
    plt.rc('ytick',labelsize=20)
    plt.xlabel("k neighbours",size=25)
    plt.ylabel(" MSE ",size=25)
    plt.errorbar(alpha,m10,yerr=std10,fmt="-o",ecolor="r",capsize=5)
    plt.errorbar(alpha,m30,yerr=std30,fmt="-o",ecolor="b",capsize=8)
    plt.errorbar(alpha,mean1,yerr=std1,fmt="-o",ecolor="g",capsize=10)
    ## plot training results
    plt.errorbar(alpha,m0,yerr=s0,fmt="-o",ecolor="r",capsize=5)
    plt.errorbar(alpha,m1,yerr=s1,fmt="-o",ecolor="b",capsize=8)
    plt.errorbar(alpha,m2,yerr=s2,fmt="-o",ecolor="g",capsize=10)
    plt.legend(["10m test predictions","30m test predictions","1h test
predictions","10m train predictions","30m train predictions","1h train
predictions "],loc='lower right',fontsize=20)
 weight ='uniform'

title= "Portobello crossValidation of k neighbours versus mse "

meanE0,stdE0,m0,s0  =   KnnModel1(qVal[0],y,t,weight)
meanE1,stdE1,m1,s1  = KnnModel1(qVal[1],y,t,weight)
meanE2,stdE2,m2,s2= KnnModel1(qVal[2],y,t,weight)
kRange = range(1,20)
```

```
printCrossEval(meanE0,stdE0,meanE1,stdE1,meanE2,stdE2,m0,s0,m1,s1,m2,s2,kRange
,title)
weightD ='distance'
title= "Portobello crossValidation of k neighbours versus mse "

meanE0,stdE0,m0,s0  =   KnnModel1(qVal[0],y,t,weightD)
meanE1,stdE1,m1,s1  = KnnModel1(qVal[1],y,t,weightD)
meanE2,stdE2,m2,s2= KnnModel1(qVal[2],y,t,weightD)
kRange = range(1,20)
printCrossEval(meanE0,stdE0,meanE1,stdE1,meanE2,stdE2,m0,s0,m1,s1,m2,s2,kRange
,title)


def gridknn(q,y,t):
  tt,yy,XX = BuildModel(q,y,t)
  train, test =
train_test_split(np.arange(0,yy.size),test_size=0.2,shuffle=False)

  cv =TimeSeriesSplit(n_splits=5)
  param = { 'n_neighbors' :range(1,20), #[2,3,4,5,6], #[3,5,7,9,11,13,15]
              'weights' : ['uniform','distance'],
              'metric' : ['minkowski','euclidean','manhattan']}

  search = GridSearchCV(KNeighborsRegressor(), param, scoring='r2', n_jobs=-1,
cv=cv)
  result = search.fit(XX[train], yy[train])
  print('\n Best Model: \n')
  best=result.best_params_

  rd=KNeighborsRegressor(**result.best_params_).fit(XX[train], yy[train])
  y_train = yy[train]
  y_test = yy[test]
  y_hat = rd.predict(XX[train])
  y_pred = rd.predict(XX[test])

  #metrics
  mse_tr=metrics.mean_squared_error(y_train, y_hat)
  mse_te=metrics.mean_squared_error(y_test, y_pred)
  r2_tr=metrics.r2_score(y_train, y_hat)
  r2_te=metrics.r2_score(y_test, y_pred)
  print(" Q value", q)
  print('Training set mse: ',math.sqrt(mse_tr))
  print('Test set mse: ', math.sqrt(mse_te))
  print('Training set r2: ', r2_tr)
  print('Test set r2: ', r2_te)

  return mse_tr,mse_te,r2_tr,r2_te, best,tt,y_train,y_test,y_hat,y_pred
y = portobello.iloc[:,1]
```

```
t = portobello.iloc[:,3]

mse_tr_q1,mse_te_q1,r2_tr_q1,r2_te_q1,best_q1,tt1,y_train,y_test,y_hat_q1,y_pr
ed_q1=gridknn(qVal[0],y,t)
mse_tr_q2,mse_te_q2,r2_tr_q2,r2_te_q2,best_q2,tt2,y_train,y_test,y_hat_q2,y_pr
ed_q2=gridknn(qVal[1],y,t)
mse_tr_q3,mse_te_q3,r2_tr_q3,r2_te_q3,best_q3,tt3,y_train,y_test,y_hat_q3,y_pr
ed_q3=gridknn(qVal[2],y,t)

q1_df=pd.DataFrame.from_dict(best_q1, orient='index',columns=['q=2'])
q2_df=pd.DataFrame.from_dict(best_q2, orient='index',columns=['q=6'])
q3_df=pd.DataFrame.from_dict(best_q3, orient='index',columns=['q=12'])
df_best_rd=pd.concat([q1_df,q2_df,q3_df],axis=1)

m1_q1=pd.DataFrame(data=[mse_tr_q1,mse_te_q1,r2_tr_q1,r2_te_q1],
                   index=['mse_train','mse_test','r2_train','r2_test'],columns
=['q=2'])
m2_q2=pd.DataFrame(data=[mse_tr_q2,mse_te_q2,r2_tr_q2,r2_te_q2],
                   index=['mse_train','mse_test','r2_train','r2_test'],columns
=['q=6'])
m3_q3=pd.DataFrame(data=[mse_tr_q3,mse_te_q3,r2_tr_q3,r2_te_q3],
                   index=['mse_train','mse_test','r2_train','r2_test'],columns
=['q=12'])
df_metrics_rd=pd.concat([m1_q1,m2_q2,m3_q3],axis=1)

df_params=pd.concat([df_best_rd,df_metrics_rd],axis=0)
df_params

plt.plot(t[-len(y_test):],y_test,'-o',color='black')
plt.plot(tt1[-len(y_pred_q1):],y_pred_q1,color='blue')
plt.plot(tt2[-len(y_pred_q2):],y_pred_q2,color='green')
plt.plot(tt3[-len(y_pred_q3):],y_pred_q3,color='red')
plt.title("KNN Model Portobello Road test data using Optimal Params ",size=30)
plt.legend(["Actual availability","10 minutes Prediction","30 minutes
Prediction","1 hour Prediction"],loc='upper right',fontsize=15)
#plt.xlim((4*7,4*7+5))
plt.xlabel('Time (days)',size=25)
plt.ylabel("# Bikes ",size=25)
plt.rcParams['figure.figsize']=[12,6]

bikes = pearse.iloc[:,1]
time = pearse.iloc[:,3]
weight ='uniform'
title= "Pearse crossValidation of k neighbours versus mse uniform weight"

meanE0,stdE0,m0,s0  =   KnnModel1(qVal[0],bikes,time,weight)
meanE1,stdE1,m1,s1  = KnnModel1(qVal[1],bikes,time,weight)
meanE2,stdE2,m2,s2= KnnModel1(qVal[2],bikes,time,weight)
```

```
kRange = range(1,20)
printCrossEval(meanE0,stdE0,meanE1,stdE1,meanE2,stdE2,m0,s0,m1,s1,m2,s2,kRange
,title)
weight ='distance'
title= "Pearse crossValidation of k neighbours versus mse weight ='distance"

meanE0,stdE0,m0,s0  =   KnnModel1(qVal[0],bikes,time,weight)
meanE1,stdE1,m1,s1  = KnnModel1(qVal[1],bikes,time,weight)
meanE2,stdE2,m2,s2= KnnModel1(qVal[2],bikes,time,weight)
kRange = range(1,20)
printCrossEval(meanE0,stdE0,meanE1,stdE1,meanE2,stdE2,m0,s0,m1,s1,m2,s2,kRange
,title)

mse_tr_q1,mse_te_q1,r2_tr_q1,r2_te_q1,best_q1,tt1,y_train,y_test,y_hat_q1,y_pr
ed_q1=gridknn(qVal[0],bikes,time)
mse_tr_q2,mse_te_q2,r2_tr_q2,r2_te_q2,best_q2,tt2,y_train,y_test,y_hat_q2,y_pr
ed_q2=gridknn(qVal[1],bikes,time)
mse_tr_q3,mse_te_q3,r2_tr_q3,r2_te_q3,best_q3,tt3,y_train,y_test,y_hat_q3,y_pr
ed_q3=gridknn(qVal[2],bikes,time)

q1_df=pd.DataFrame.from_dict(best_q1, orient='index',columns=['q=2'])
q2_df=pd.DataFrame.from_dict(best_q2, orient='index',columns=['q=6'])
q3_df=pd.DataFrame.from_dict(best_q3, orient='index',columns=['q=12'])
df_best_rd=pd.concat([q1_df,q2_df,q3_df],axis=1)

m1_q1=pd.DataFrame(data=[mse_tr_q1,mse_te_q1,r2_tr_q1,r2_te_q1],
                   index=['mse_train','mse_test','r2_train','r2_test'],columns
=['q=2'])
m2_q2=pd.DataFrame(data=[mse_tr_q2,mse_te_q2,r2_tr_q2,r2_te_q2],
                   index=['mse_train','mse_test','r2_train','r2_test'],columns
=['q=6'])
m3_q3=pd.DataFrame(data=[mse_tr_q3,mse_te_q3,r2_tr_q3,r2_te_q3],
                   index=['mse_train','mse_test','r2_train','r2_test'],columns
=['q=12'])
df_metrics_rd=pd.concat([m1_q1,m2_q2,m3_q3],axis=1)

df_params=pd.concat([df_best_rd,df_metrics_rd],axis=0)
df_params
plt.plot(t[-len(y_test):],y_test,'-o',color='black')
plt.plot(tt1[-len(y_pred_q1):],y_pred_q1,color='blue')
plt.plot(tt2[-len(y_pred_q2):],y_pred_q2,color='green')
plt.plot(tt3[-len(y_pred_q3):],y_pred_q3,color='red')
plt.title("KNN Model Pearse Street test data using Optimal Params ",size=30)
plt.legend(["Actual availability","10 minutes Prediction","30 minutes
Prediction","1 hour Prediction"],loc='upper right',fontsize=18)
#plt.xlim((4*7,4*7+5))
plt.xlabel('Time (days)',size=25)
plt.ylabel("# Bikes ",size=25)
```

```
plt.rcParams['figure.figsize']=[12,6]
```