VSEVOLOD SYRTSOV
18323202

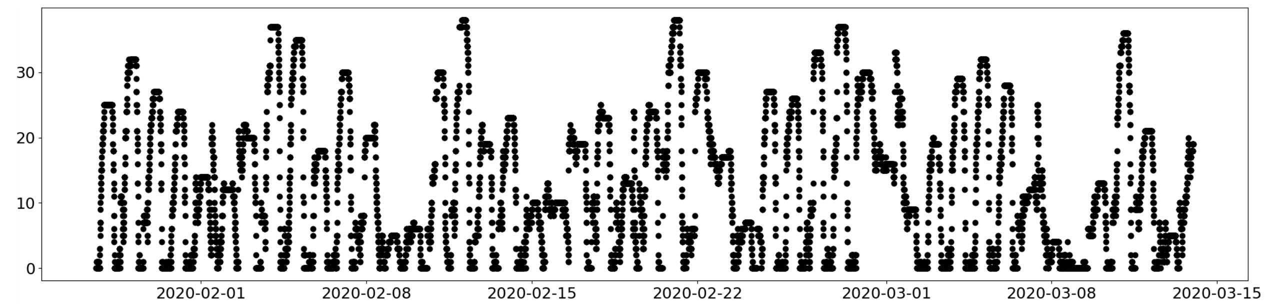
QUESTION ONE
I have a few ideas as to how to feasibly obtain some sort of prediction for future states that a bike station will be in. This looks like it can be approached from a time-series forecasting approach, and as such I will examine this problem by tweaking characteristics of the method, selecting feature ranges that return the best prediction, and adequately selecting which parts of the data to train and test my method with.

STATIONS. The two stations I chose are at Kilmainham Gaol - *ID 97* - and at Charlemont Place - *ID 5.* Both of the stations have a max capacity of 40 bikes, which I chose specifically to have a larger range to assess, which would result in smaller % change over time. The former I chose because it is one of the furthest stations from the centre, and the latter because it is a station located within the city centre with a max capacity.

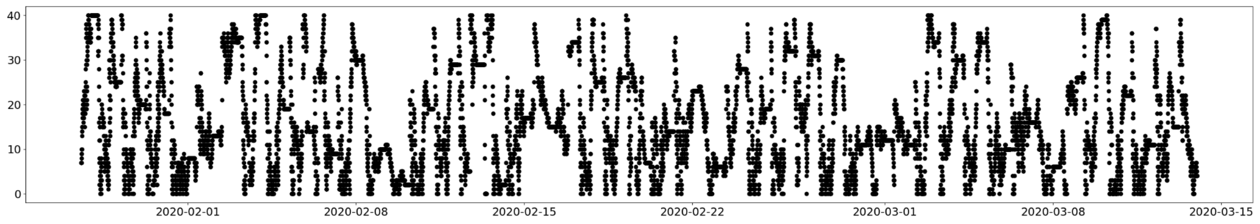
DATASET. I think that choosing a large period of time, such as a month, will be enough to satisfy all testing requirements required to make a deduction. 1 month contains 4-5 weeks, that would be 4-5 occurrences of a certain day within a month, a decent spread for k-fold splitting testing and training data, as well as trying to perform seasonal time-series forecasting - later found that seasonal time series in this assignment is not a viable solution.

MACHINE LEARNING MODEL SELECTIONS AND JUSTIFICATION. I'm sure there are many ways well outside of the scope of the module that are equipped to deal with this problem, however I am most interested in examining how well different regression models perform in this scenario. To observe the feasibility of predicting the status in the referenced future times, I elect to train *Linear Regression* and *kNN Regression* models. Considering that time-series forecasting uses a relatively small feature vector for making predictions, I believe that trying to eliminate multicollinearity, outliers, or any possible noise in a feature window of 3-6 features is trivial. If one, or even two(or more) of the features is/are noisy, then that would create a feature vector with a relatively high amount of noise so much so that the feature vector is effectively not very descriptive of the nature of the data in the first place. Given that I am going to train models on multiple areas of the data (which should have some correlation given the shape of the graph below), I would rather focus on creating an argument conducive to the assignment than deal with carefully selecting hyperparameters at the possible expense of getting rid of actual features. The kNN regression model I feel will be quite well suited to the task, as the entire feature vector will be used as it's neighbours, and given my assumption that the feature vectors will most likely not contain noise, is also suited to datasets with low signal-to-noise ratios. In this context, the cost function for the linear regression model, the Root Mean Squared Error(RMSE) is computationally feasible on any dataset, I don't think any more complexity needs to be invited for such a small dataset. The kNN algorithm does not have a cost function, in the sense that no parameters are minimized during training. Instead it approximated the association between independent variables in the dataset, and with such a small dataset, associations are very quick to compute, making it an adequate application for the problem. This also presents the opportunity to compare the effectiveness of a parametric and non-parametric model in this context.

Below are the graphs for Kilmainham Gaol and Charlemont place, with the amount of available bikes graphed on the y-axis:



Kilmainham



Charlemont

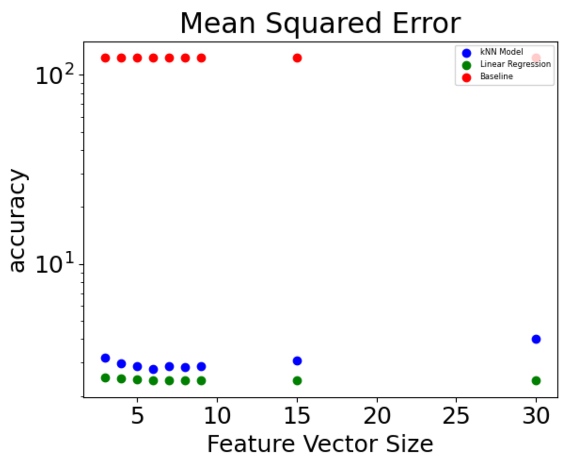
I purposefully truncated a 3-month period from Jan 1 to Mar 31 to be from Jan 27 to Mar 13, as the data outside of that was unusually sparse.

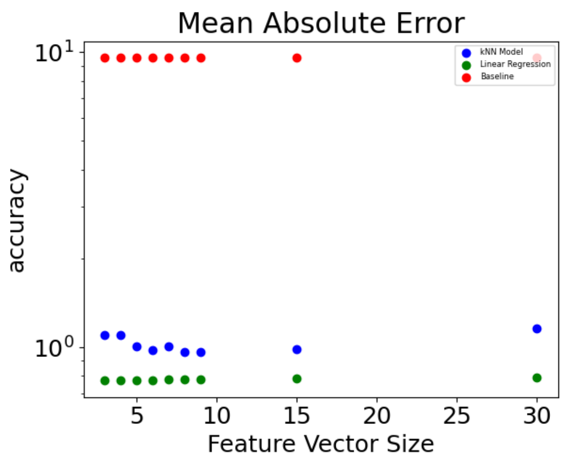
FEATURE ENGINEERING. I first of all selected two stations that would have different types of usage, with one being in the city centre and the other being out as far as possible from the centre. I began by examining the station at Kilmainham as from a glance it seems that it has more distinct peak times than the other station.

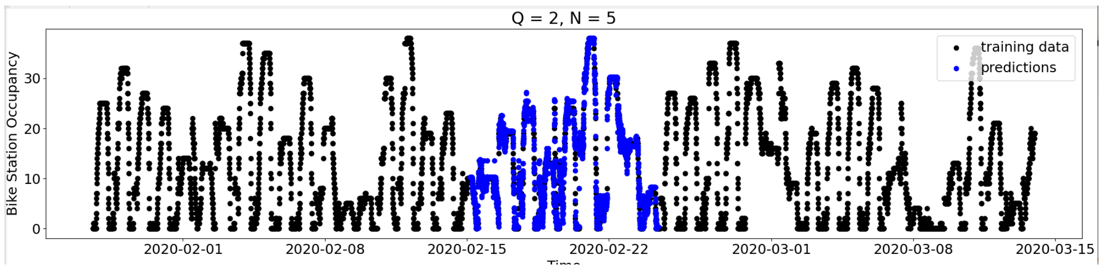
Kilmainham Gaol - no. 97

10 mins

Choosing features for this prediction proved to be relatively easy - I decided to try to predict q=2 steps ahead by using a short term trend analysis. My choice of feature vector size varied from 3-9,15,30 data points. The kNN model performs slightly worse across the two metrics I have chosen to use to evaluate my models. Below is a graph of the results of the Mean Squared Error(MSE) and Mean Absolute Error(MAE) of the kNN Model, Linear Regression Model, a mean baseline regression model. I chose these metrics because MSE effectively describes accuracy of the models, penalising heavily for larger inaccuracies, whereas the MAE I feel is good for evaluating the pragmatic use of a model for the task, for example if one person needed to know if there would be a bike available at a station, there would be no cause for concern if 17 bikes are predicted to be available when there only is 15 or 16, and can also be used to express a % of the total available bikes. Clearly the regression models heavily outperform the baseline model and indicate that it is feasible to predict bike station occupancy 10 minutes into the future with some small error. More data points in short term trend analysis proves to decrease accuracy. Below that is a graph of the general shape of the predictions that the best performing configuration has, in one of its folds - clearly very similar to the shape of the real data. **The best performing model is a Linear Regression model using a feature vector with 6 data points**.

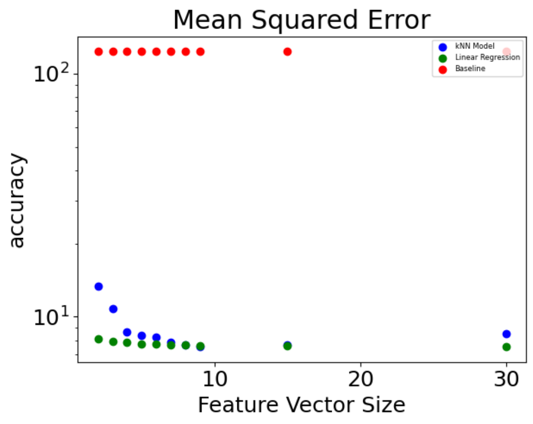


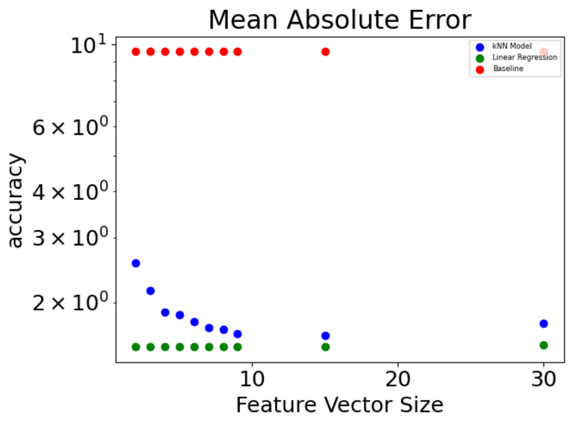




30 mins

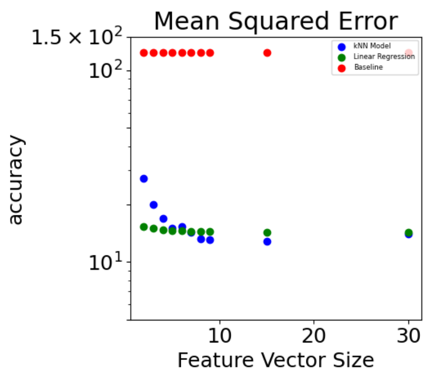
For this prediction I decided to use short term trend prediction once again. The results were more interesting, as the MSE and MAE began to converge at some point, as well as the kNN model performing about as good as the linear regression model. From a feasibility standpoint, the models perform relatively well once again in predicting future occupancy.

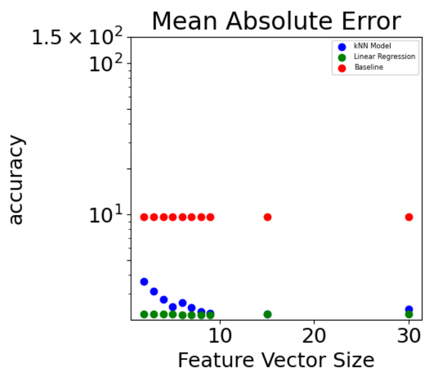




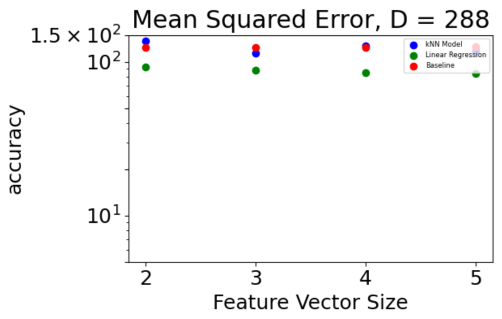
1 hr

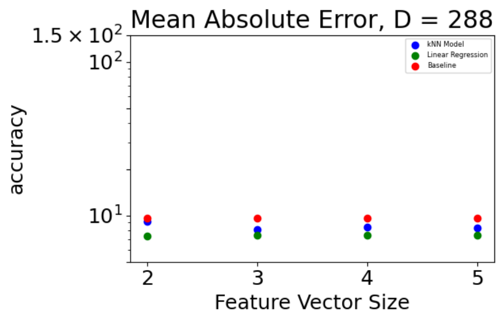
My approach for this scenario was rooted in avoiding creating a model that can predict an hour into the future, and to rather expend all possible configurations in an effort to find one where the metrics display performance that is at best equal to the models' performances in previous scenarios, and at worst better than the baseline regression. From examining short term trend predictions below are the metrics' results. Sure, they perform better than baseline, but clearly worse than what they were predicting for shorter time periods in the future. Across all models, the linear model seems to remain calmly at one level, whereas the kNN model approaches the linear model's representative metric value as the neighbours increase, and if increasing a little more, loses its predictive ability, for short term trends.



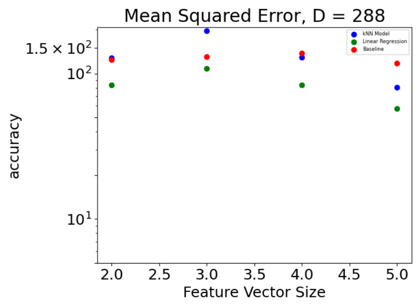


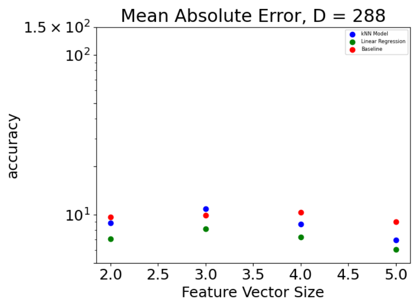
I was also interested in how well the models perform when using seasonal data in the feature vector. Long story short they do not perform well at all. Below are the graphs for seasonal predictions using the last 5 days. Since the metrics show that it does perform that much better than the baseline regression, it is not feasible in predicting occupancy an hour ahead.





…and for when using weekly predictions, using the last 2-5 weeks. It is clearly not the best for trying to predict 1 hour in the future.



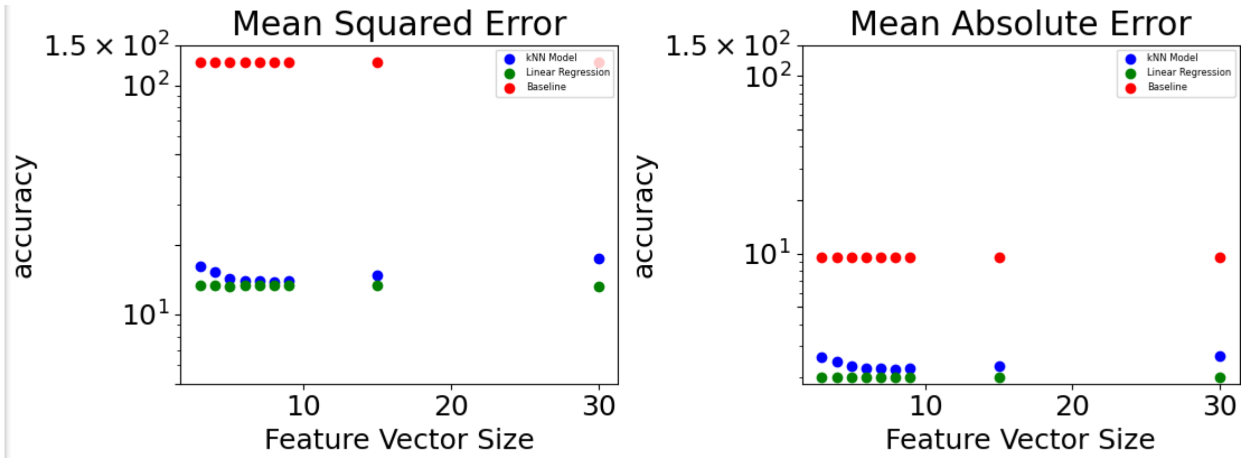


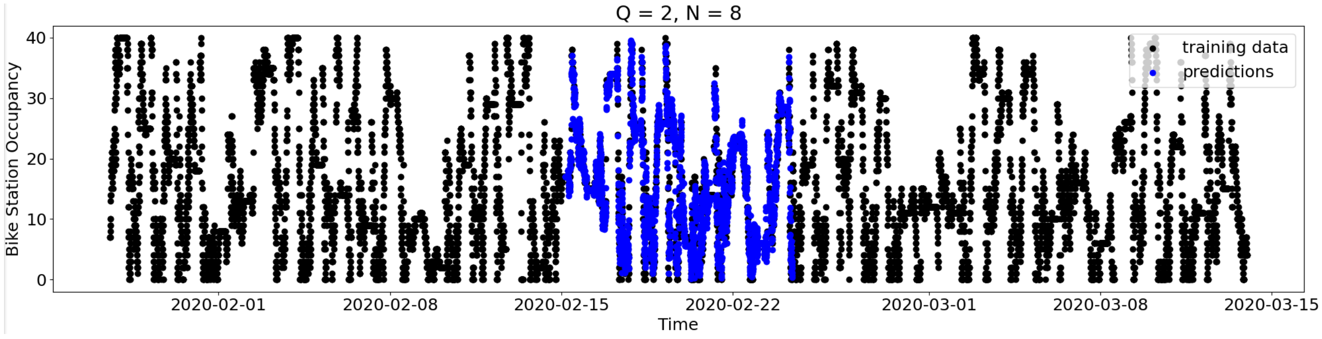
Charlemont Place - no. 5

I will take the same agnostic approach to discovering if it is feasible to predict the occupancy of this bike station as I did when examining the one at Kilmainham.

10 mins

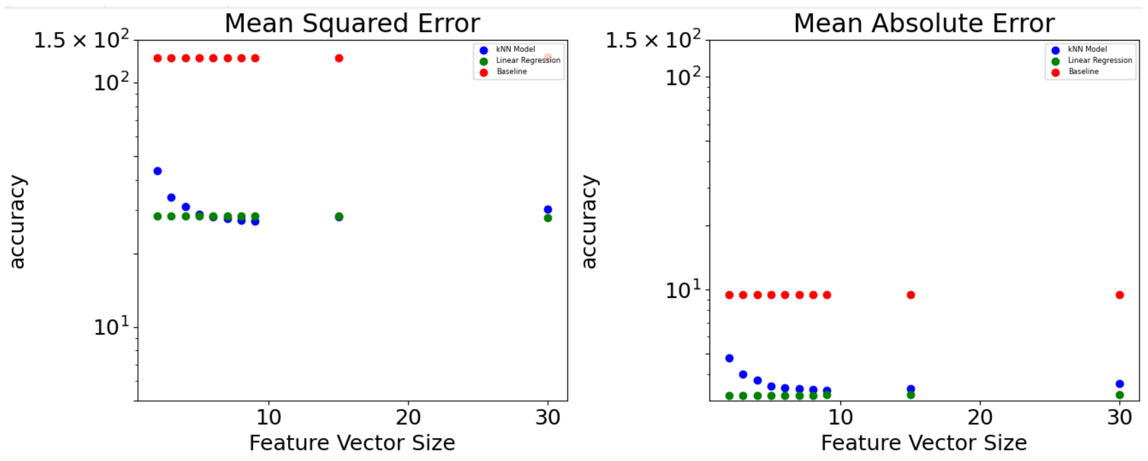
Naturally, I approached this one with the **same tools and mindset as the previous station**, and below I have captured the performance of the models when predicting 10 minutes into the future using short term analysis. First thing to note is that in general they perform slightly worse than when examining Kilmainham. Given that this station is in the city centre it is used far more often than the one on the outskirts of the city, and as such has a much more hectic shape. Still, it performs far better than the baseline regression, and that is enough to say that it is feasible to predict with some degree of persuasiveness the occupancy of the bike station 10 minutes into the future. The lower graph demonstrates the shape of the predictions for a fold, which matches pretty closely to the actual data. Once again the linear model performs better than kNN regression.





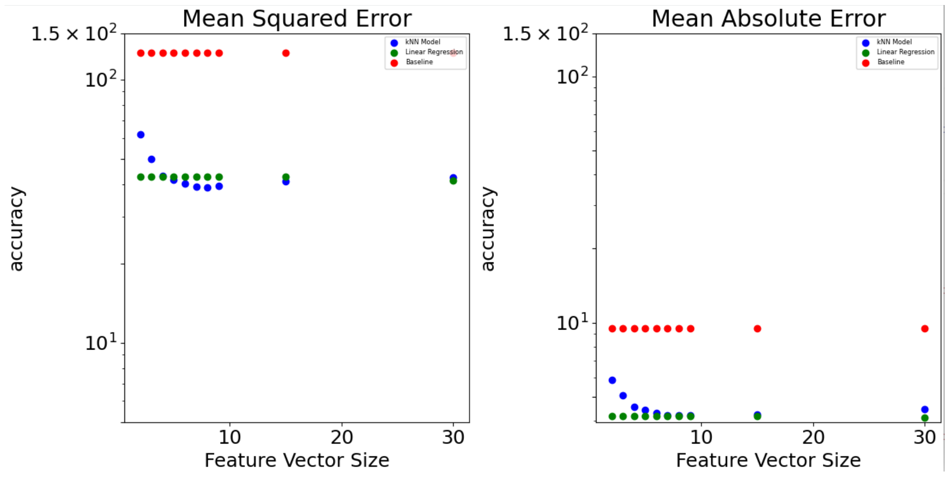
30 mins

For this prediction I decided to use short term trend prediction once again. The results were more interesting, as the MSE and MAE began to converge at some point, as well as the kNN model performing about as good as the linear regression model. From a feasibility standpoint, the models perform relatively well once again in predicting future occupancy.

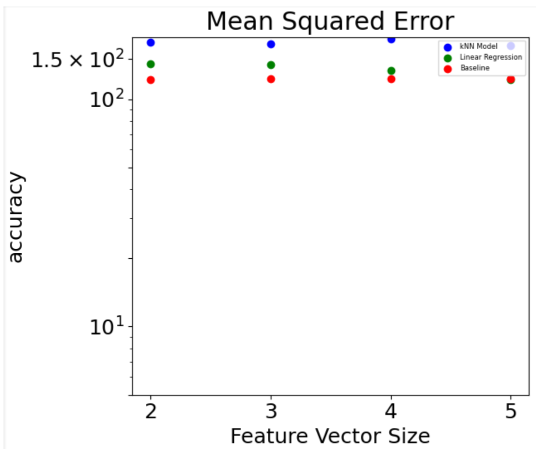


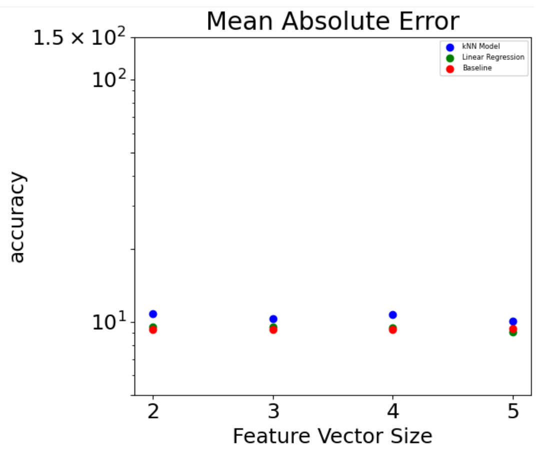
1 hr

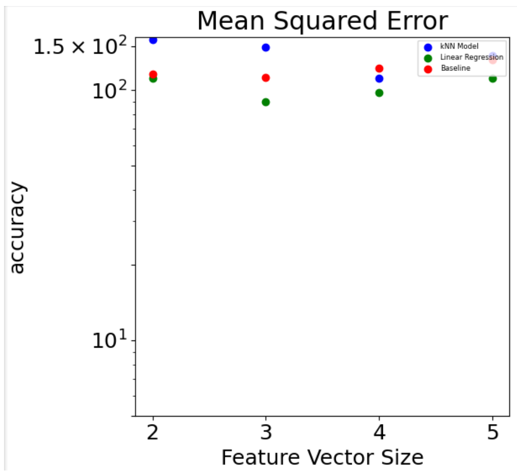
Continuing on from the previous examination of the hourly data, it seems that it is also quite difficult to determine with certainty the occupancy of the bike station an hour in the future, using the pool of configurations I have chosen to examine. The absolute best configuration seems to be 8 data points for short term trend analysis across the models, and the metrics indicate this. The city never sleeps it seems…

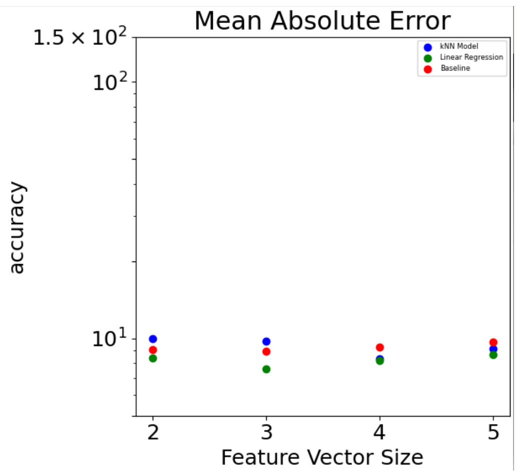


I also examined the data using seasonal analysis, to no avail. The first graph below is using daily data, and the one below that uses weekly data, with feature vectors containing 2-5 data points. The same conclusions as from the last station can be drawn from the results and associated with this station: **it is not feasible to predict 1 hour into the future using seasonal feature vectors.**









Summary

Using Linear Regression and kNN Regression models it is feasible to predict to an acceptable degree of accuracy the occupancy of bike stations 10, 30 and 60 minutes into the future, given observations and trials from two stations of differing nature. Seasonal analysis does not provide the same results as short term trend analysis does, and is no better if not worse in some scenarios than predicting using the mean.

QUESTION TWO

i. An ROC curve is a metric for evaluating the performance of a classifier first of all. It plots the rate of True Positives appearing (TP/ TP+FN) against the rate of False Positives (FP/FP+TN) in a model. A classifier that shows a curve that tends to the top left of the graph indicates that it has a high rate of True Positives throughout its performance, which indicates a well performing classifier. Can be used to also compare classifiers.

ii. A linear regression model would not work where the data does not fit the purpose of a LR model, for example where the input of a feature vector should map to some classification, rather than a continuous range, because a linear model outputs a function that matches the form of a line (y=mx+c, where x is the feature vector in this case) where the output is not normalised and is continuous as previously stated. Pure linear regression does not work as well as other models for datasets where there is high multicollinearity. For good measure, another place where a linear regression model would not work is with a non-linear dataset.

iii.

1. An SVM classifier is essentially a logistic regression model with a different cost function, which also optimises a set of parameters. An SVM in this case retains the simplicity of a logistic regression and when approaching a problem where machine learning begins to appear as a viable solution, an SVM is among the first models to consider if there is a classification problem.
2. An SVM model is capable of performing the same kernel trick that a neural net is capable of, transforming its inputs into feature-spaces of higher dimensions, while not having to trade off computational costs like a neural net would require.
3. Considering the amount of data that a neural net needs to train, due to growing complexity as it traverses its layers, an SVM is capable of being trained with smaller datasets, overall reducing the time for performing some analysis.

iv. A convolution layer in a ConvNet is composed of multiple nodes which map inputs to outputs. Each node in a convolution layer applies a kernel to some input which results in an abstracted output which can be used more conveniently by succeeding layers. The name convolution layer comes from the fact that a kernel is used to convolve the input, whereas there are other layers that apply different mathematics to alter it. A convolution layer can take input from another convolution layer, and send its output to another convolution layer, over and over resulting in the term deep learning. An example of a convolution layer would involve iterating over groups of values in an image using a convolution kernel to detect edges, which is very useful in computer vision.

v. Using k-fold cross-validation lets the model be trained more times, with more datasets, though divided. This adds more noise into the training, which averaging out eliminates to provide a general representation of what the model is supposed to perform like. Resampling the data provides a slightly more accurate method of analysing the training set as well, giving more information to work with when choosing models, configurations, etc. For example, for a dataset, using k-fold cross validation one would divide the dataset into k equal parts. If k=5 then there would exist k1..5 train/test pairs of data, where in each pair the test section will be different from all the other pairs.

APPENDIX - CODE

```
import pandas as pd
import numpy as np
import math, sys
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression
from sklearn.dummy import DummyRegressor
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error, mean_absolute_error

plt.rc('font', size=18); plt.rcParams['figure.constrained_layout.use'] =
True
# read data. column 1 is date/time, col 6 is #bikes
df = pd.read_csv("bikesdata.csv", usecols = [0,1,3,4,6], parse_dates=[1])
kilmainham = df[df['STATION ID'] == 97].copy()
charlemont = df[df['STATION ID'] == 5].copy()


start=pd.to_datetime("27-01-2020",format='%d-%m-%Y')
end=pd.to_datetime("14-03-2020",format='%d-%m-%Y')
kilmainham = kilmainham.loc[(kilmainham['TIME'] > start) &
(kilmainham['TIME'] <= end)].copy()

kilmainham_timestamps = kilmainham["TIME"].tolist()
kilmainham_occupancy = kilmainham["AVAILABLE BIKES"].tolist()
#plt.scatter(kilmainham_timestamps, kilmainham_occupancy, color='black')
#plt.show()
charlemont = charlemont.loc[(charlemont['TIME'] > start) &
(charlemont['TIME'] <= end)].copy()

charlemont_timestamps = charlemont["TIME"].tolist()
charlemont_occupancy = charlemont["AVAILABLE BIKES"].tolist()
#plt.scatter(charlemont_timestamps, charlemont_occupancy, color='black')
#plt.show()



def test_pred(timestamps, occupancy, q,d,n, plot, station):
   # num_features*d = max offset from k-q => k-q-d*n
   # => k = i+q+d*n
   i = 0
   features = []
   targets = []
   time_domain = []
   while(i+q+(d*n) < len(occupancy)):
       feature_vector = []
       for index in range(n):
           feature_vector.append(occupancy[i+(index*d)])
       features.append(feature_vector)
       targets.append(occupancy[i+q+(d*n)])
       time_domain.append(timestamps[i+(d*n)])
       i += 1
   accuracies = [0,0,0,0,0,0]
   for train, test in KFold(n_splits=5).split(features):
       knn_model =
KNeighborsRegressor(n_neighbors=n,weights='uniform').fit(np.array(features
)[train], np.array(targets)[train])
       lr_model =
LinearRegression(fit_intercept=False).fit(np.array(features)[train],
np.array(targets)[train])
       dummy_model =
DummyRegressor(strategy="mean").fit(np.array(features)[train],
np.array(targets)[train])
       dummy_preds = dummy_model.predict(np.array(features)[test])
       knn_preds = knn_model.predict(np.array(features)[test])
       lr_preds = lr_model.predict(np.array(features)[test])
       knn_mse_accuracy =
mean_squared_error(np.array(targets)[test],knn_preds)
       lr_mse_accuracy =
mean_squared_error(np.array(targets)[test],lr_preds)
       dummy_mse_accuracy =
mean_squared_error(np.array(targets)[test],dummy_preds)

       knn_mae_accuracy =
mean_absolute_error(np.array(targets)[test],knn_preds)
       lr_mae_accuracy =
mean_absolute_error(np.array(targets)[test],lr_preds)
```

```
        dummy_mae_accuracy =
mean_absolute_error(np.array(targets)[test],dummy_preds)

        # Store MSE + MAE accuracies
        accuracies[0] += knn_mse_accuracy
        accuracies[1] += lr_mse_accuracy
        accuracies[2] += dummy_mse_accuracy
        accuracies[3] += knn_mae_accuracy
        accuracies[4] += lr_mae_accuracy
        accuracies[5] += dummy_mae_accuracy


        if plot:
            plt.scatter(timestamps, occupancy, color='black')
            plt.scatter(np.array(time_domain)[test], knn_preds,
color='blue')
            plt.title("Q = {}, N = {}".format(q,n))
            plt.xlabel("Time")
            plt.ylabel("Bike Station Occupancy")
            plt.legend(["training data", "predictions"], loc="upper right")
            plt.show()
            plt.cla()
    knn_model =
KNeighborsRegressor(n_neighbors=n,weights='uniform').fit(np.array(features
), np.array(targets))

    # Get averages of accuracies from the 5 folds
    accuracies = np.array(accuracies)/5
    print("\n\n{}, q = {}, n = {}\nkNN Model MSE = {}, Linear Regression
MSE = {}, Dummy Regressor MSE - {}\nkNN Model MAE = {}, Linear Regression
MAE = {}, Dummy Regressor MAE - {} ".format(station,q,n,
round(accuracies[0],2), round(accuracies[1],2),
round(accuracies[2],2),round(accuracies[3],2), round(accuracies[4],2),
round(accuracies[5],2)))
    return accuracies

def plot_accuracies(accuracies, window, d):
    knn_mse_accuracies = [x[0] for x in accuracies]
    lr_mse_accuracies = [x[1] for x in accuracies]
    dummy_mse_accuracies = [x[2] for x in accuracies]
```

```
   knn_mae_accuracies = [x[3] for x in accuracies]
   lr_mae_accuracies = [x[4] for x in accuracies]
   dummy_mae_accuracies = [x[5] for x in accuracies]

   plt.subplot(1,2,1)
   plt.scatter(window,knn_mse_accuracies, color="blue")
   plt.scatter(window,lr_mse_accuracies, color="green")
   plt.scatter(window,dummy_mse_accuracies, color="red")
   plt.title("Mean Squared Error")
   plt.xlabel("Feature Vector Size")
   plt.ylabel("accuracy")
   plt.legend(["kNN Model", "Linear Regression", "Baseline"], loc="upper
right", prop={'size': 6})
   plt.yscale("log")
   plt.yticks([5,10,20,50,100,150])

   plt.subplot(1,2,2)
   plt.scatter(window,knn_mae_accuracies, color="blue")
   plt.scatter(window,lr_mae_accuracies, color="green")
   plt.scatter(window,dummy_mae_accuracies, color="red")
   plt.title("Mean Absolute Error")
   plt.xlabel("Feature Vector Size")
   plt.ylabel("accuracy")
   plt.legend(["kNN Model", "Linear Regression", "Baseline"], loc="upper
right", prop={'size': 6})
   plt.yscale("log")
   plt.yticks([5,10,20,50,100,150])


   plt.show()
# -- KILMAINHAM --
"""
# 10 minutes ahead - q = 2, using last 5 values
window = [3,4,5,6,7,8,9, 15,30]
accuracies = []
for n in window:
   accuracies.append(test_pred(kilmainham_timestamps,
kilmainham_occupancy, 2,1,n,False, "Kilmainham"))

# plot_accuracies(accuracies, window,1)
# 30 minutes ahead - q = 6, using last 7 values
window = [2,3,4,5,6,7,8,9, 15,30]
accuracies = []
for n in window:
   accuracies.append(test_pred(kilmainham_timestamps,
kilmainham_occupancy, 6,1,n,False, "Kilmainham"))

plot_accuracies(accuracies, window,1)

# 1 hour ahead - q = 10, using short term data
window = [2,3,4,5,6,7,8,9,15,30]
accuracies = []
for n in window:
   accuracies.append(test_pred(kilmainham_timestamps,
kilmainham_occupancy, 10,1,n,False, "Kilmainham"))

plot_accuracies(accuracies, window,1)



# 1 hour ahead - q = 10, using daily data
window = [2,3,4,5]
accuracies = []
for n in window:
   accuracies.append(test_pred(kilmainham_timestamps,
kilmainham_occupancy, 10,288,n,False, "Kilmainham"))

plot_accuracies(accuracies, window, 288)



# 1 hour ahead - q = 10, using weekly data
window = [2,3,4,5]
accuracies = []
for n in window:
   accuracies.append(test_pred(kilmainham_timestamps,
kilmainham_occupancy, 10,2016,n,False, "Kilmainham"))

plot_accuracies(accuracies, window, 288)
```

```
# -- CHARLEMONT PLACE --

# 10 minutes ahead - q = 2, using last 5 values
window = [3,4,5,6,7,8,9, 15,30]
accuracies = []
for n in window:
   accuracies.append(test_pred(charlemont_timestamps,
charlemont_occupancy, 2,1,n,True, "Charlemont"))

plot_accuracies(accuracies, window,1)

# 30 minutes ahead - q = 6, using last 7 values
window = [2,3,4,5,6,7,8,9, 15,30]
accuracies = []
for n in window:
   accuracies.append(test_pred(charlemont_timestamps,
charlemont_occupancy, 6,1,n,False, "Charlemont"))

plot_accuracies(accuracies, window,1)

# 1 hour ahead - q = 10, using short term data
window = [2,3,4,5,6,7,8,9,15,30]
accuracies = []
for n in window:
   accuracies.append(test_pred(charlemont_timestamps,
charlemont_occupancy, 10,1,n,False, "Charlemont"))

plot_accuracies(accuracies, window,1)




# 1 hour ahead - q = 10, using daily data
window = [2,3,4,5]
accuracies = []
for n in window:
   accuracies.append(test_pred(charlemont_timestamps,
charlemont_occupancy, 10,288,n,False, "Charlemont"))

plot_accuracies(accuracies, window, 288)

"""
# 1 hour ahead - q = 10, using weekly data
window = [2,3,4,5]
accuracies = []
for n in window:
   accuracies.append(test_pred(charlemont_timestamps,
charlemont_occupancy, 10,2016,n,False, "Charlemont"))

plot_accuracies(accuracies, window, 288)
```