MANU PRASANNAKUMAR
20305536
M.Sc. Computer Science - Data Science

Final Assignment

CS7CS4-202122 Machine Learning

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

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

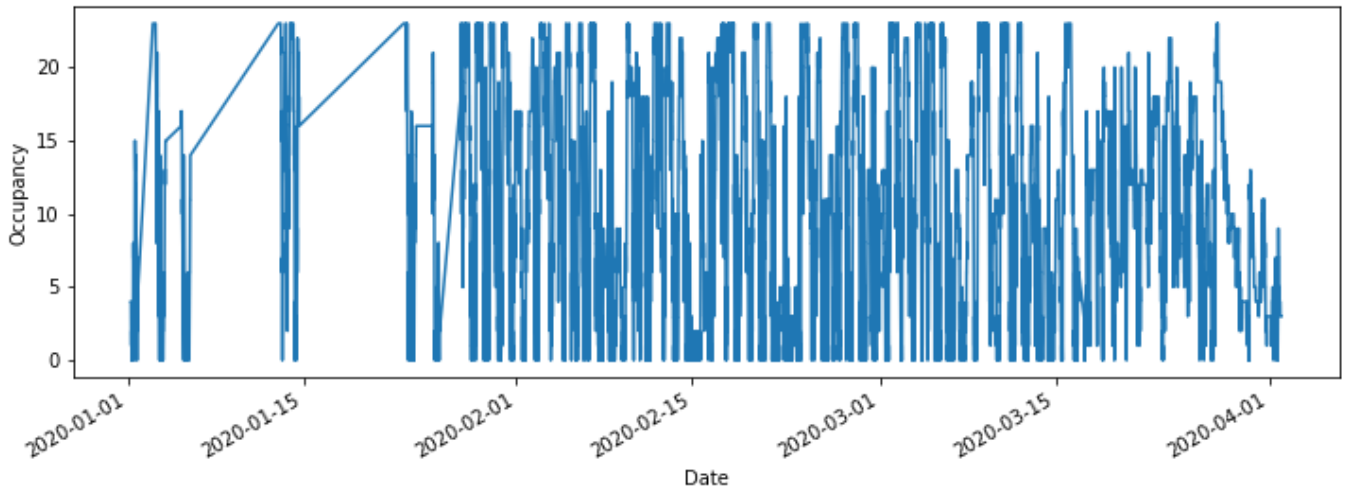
1

Dataset: Dublin bikes 2020 Q1 usage data at https://data.gov.ie/dataset/dublinbikes-api
The dataset is the Dublin bikes data for the 2020 Q1 consisting of records for each 5-minute time step from different Dublin bike stations. It consists of 11 columns and 2228278 rows (records). The column details are as follows:

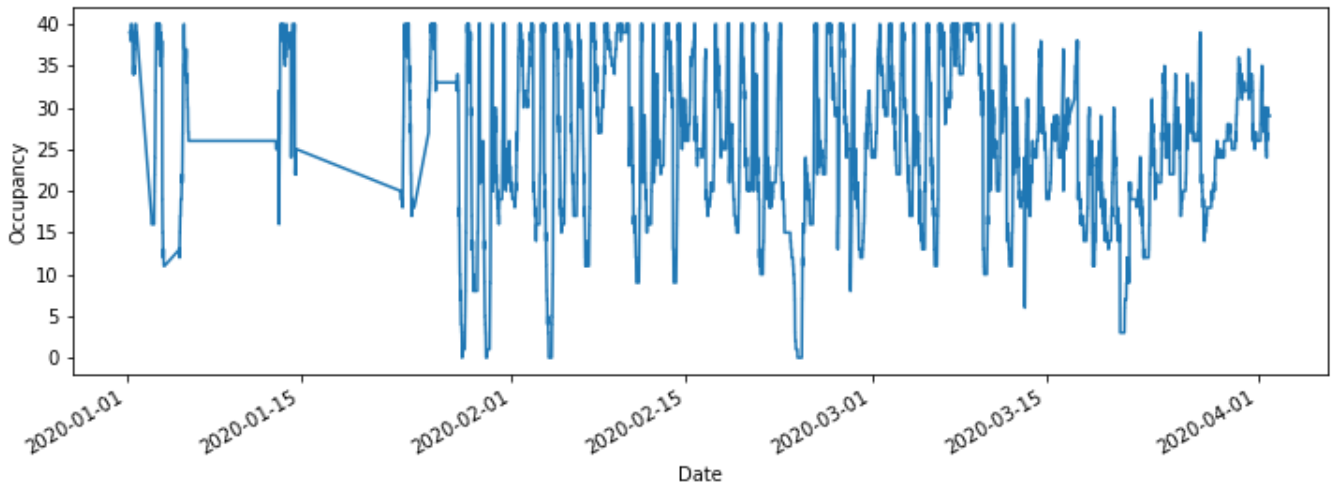
STATION ID	Id of the station for which data is recorded
TIME	Time stamp of the record
LAST UPDATED	Last Updated Time
NAME	Name of station
BIKE STANDS	Total number of bike stands in the station
AVAILABLE BIKE STANDS	Number of available bike stands at that time
AVAILABLE BIKES	Number of available bike at that time
STATUS	Status of the station
ADDRESS	Address of the station
LATITUDE	Latitude of the station
LONGITUDE	Longitude of the station

For this study, we have restricted our analysis to stations - "PRINCES STREET / O'CONNELL STREET" and 'CHARLEVILLE ROAD'. We have selected these two stations because “PRINCES STREET / O'CONNELL” is station right at the city centre and 'CHARLEVILLE ROAD' is station which is furthest from the city centre. This enables us to analyse and incorporate different behaviours such as a highly busy region near the city centre and a comparatively less busy region from outside the city centre. This will enable our model to be more versatile in identifying all kinds of trends in busy and less busy regions, such as spikes during weekdays because of daily commutes, holiday variations, etc. So, the data is filtered for the above two stations (Station ids - 33 and 107). The target column we are trying to predict is the “Available Bike Stands” column, as this column can be fairly assumed to be representing the occupancy (the number of bikes in use at a time). The plots of this column at different time steps for two stations are shown below:

Station 33(PRINCES STREET / O'CONNELL STREET)



Station 107('CHARLEVILLE ROAD)



From the plots, we can clearly see that the city centre station is busier than the other station. We can also identify some peak usages and low usages at different times. We need to build models to recognize these patterns.

Feature Engineering

We are removing columns such as 'STATION ID', 'LAST UPDATED', 'NAME', 'BIKE STANDS', 'AVAILABLE BIKES', 'STATUS', 'ADDRESS', 'LATITUDE', 'LONGITUDE' as these columns are of not providing us any information required for predicting the occupancy of the stations. For incorporating the different behavioural patterns from each station (because of their locations), the two station ids are converted using one hot encoding into two new columns - STATION_33 and STATION_107. This is achieved by using get_dummies function from the pandas library. So, the dataset will now have a value of 1 for STATION_33 and 0 for STATION_107 columns corresponding to station 33 records and vice versa for station 107 records. Since this is a time series problem, the time column which we have retained is important to us. We are generating a lot of new features (columns) from this column. The day of the month (1-31), day of the week (0-6) and hour of the day (0-23) are separated as three separate columns from the time column using attributes from the pandas datetime object. These will help us in identifying patterns arising during different weekdays, hours and days of the month. Since this is a time series data, the future data always has some relationship with past data and to incorporate this in our models we have generated some lag features. We have generated three hourly lag features- hourly_1, hourly_2 and hourly_3 which represents the number of available bikes stands at 1 hour before, 2 hours before and 3 hours before. This will help us in identifying the trends and preaks which occur hourly in the data. Three daily lag features – daily_1, daily_2 and daily_3, which represents the number of available bike stands at 1 day before, 2 days before and 3 days before, are generated. These features will help the model in capturing the daily trends and peaks. We have also generated three weekly lag features- weekly_1, weekly_2 and weekly_3, which represents the number of available bike stands at 1 week before, 2 weeks before and 3 weeks before. These features will help the model in capturing the trends and peaks occurring weekly. The null values which were generated for the new lag features because of the absence of past data were imputed with the mean values of that column for that station. All the columns except Available Bike Stands, Station_33 and Station_107 are normalized to values between 0 to 1. Normalization improves the model accuracy by making sure that no variable will dominate in the model equation because of the higher range of values that variable has. The Available Bike Stands column is excluded as it is the target variable and, Station_33 and Station_107 are excluded as these are one hot encoded column with values of either 0 or 1 and hence does not need normalization. The final processed dataset with all engineered features is shown below:

date	AVAILABLE BIKE STANDS	STATION_33	STATION_107	Day of the Month	Day of the Week	Hour of the Day	hourly_1	hourly_2	hourly_3	daily_1	daily_2	daily_3	weekly_1	weekly_2	weekly_3
2020-01-01 06:25:02	4	1	0	0.032258	0.333333	0.26087	0.263747	0.263777	0.263921	0.267793	0.264859	0.270317	0.271092	0.268319	0.277182
2020-01-01 06:30:02	4	1	0	0.032258	0.333333	0.26087	0.263747	0.263777	0.263921	0.267793	0.264859	0.270317	0.271092	0.268319	0.277182

Machine Learning Methodology

The future prediction with the above engineered features is achieved using the below machine learning techniques.

Linear Regression

The first choice for model building is a linear regression model because it is the simplest model which can capture all the relationship between the target variable and the features if the features and target are having a linear relationship. In linear regression, the target variable is equated as a linear combination of the feature variables. The objective of the linear model is to find the best fitting straight line which has the minimum mean of squared errors (difference of target value and the value predicted by the line of all the training data points). The model equation of the linear regression with three features (x_1, x_2 and x_3) is: $y(\theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$.
Linear regression is divided into two types:

a. Ridge Regression

In ridge regression, the model equation which is shown above has an additional term. This is the regularization term which is calculated as the square of absolute sum of all the coefficients. As the coefficients become larger, this term also become larger increasing the error of the model. So, the model must minimize the size of the coefficients to reduce this error, and hence will prevent overfitting that can be caused by simply increasing the coefficient values. This model is implemented using Ridge function from the sklearn library. There is a hyperparameter which we can control to increase and reduce the weight of this regularization term. The hyperparameter is alpha=1/2*C. Increasing C will reduce the regularization term and hence model will be able to generate larger coefficients for the features. Decreasing C will increase the regularization term and force the model to have smaller coefficients.

b. Lasso Regression

In Lasso regression, the model equation which is shown above has a regularization term which will be the absolute sum of coefficients. As the coefficients become larger, this term also become larger increasing the error of the model. So, the model must minimize the size of the coefficients to reduce this error, and hence will prevent overfitting that can be caused by simply increasing the coefficient values. This model is implemented using Lasso function from the sklearn library. There is a hyperparameter which we can control to increase and reduce the weight of this regularization term. The hyperparameter is alpha=1/2*C. Increasing C will reduce the regularization term and hence model will be able to generate larger coefficients for the features. Decreasing C will increase the regularization term and force the model to have smaller coefficients.

Random Forest Regression

Random forest is an ensemble machine learning algorithm which means that the final model will consist of a group of models rather than one single model. We have selected random forest as the next choice for model building because we have already tried a linear model and we are checking if the features have a nonlinear relationship with the target. If there are non-linear relationships, we can improve the results we got from the linear model, as the random forests can capture the non-linear relationships unlike the linear models. In random forest, a group of decision trees will be trained on the data, wherein each model will be trained on a random sample of data points from the data and random sample of features. Each decision tree will be having some number of final classes (leaf nodes) to which the tree will assign a new data point on classification based on the decision conditions of that leaf node. In case of regression, the decision tree will classify the points to the final leaf nodes based on the decision conditions and then calculate the value of the new point as the average value of the other points in the same class. The same calculation is done by all the decision trees in the ensemble and the final prediction of the random forest regression will be the average of the results from all the decision trees. This is implemented by using the RandomForestRegressor function from the sklearn library. One important hyperparameter we have used in this analysis is the max_depth. It refers to the maximum depth each decision tree can grow. Limiting this hyperparameter, we can avoid overfitting. We selected the best value for this using cross validation. Another hyperparameter we have used to avoid overfitting of the random forest is the min_impurity_decrease. This parameter determines at what threshold of impurity decrease should the decision trees stop dividing further into nodes. Using this we can stop the decision trees from growing to the extend that it overfits and we can decide what is the minimum impurity decrease we are expecting while separating into classes.

Evaluation

Baseline Model

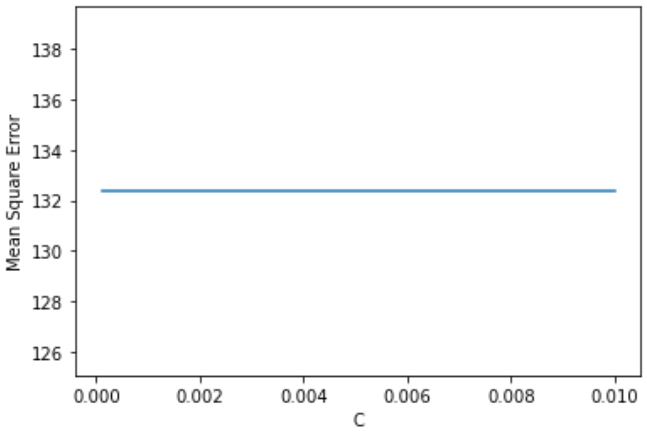
Since we are analysing time series data, we have used a naive model which predicts the value of the series one hour before as the current value, as the baseline model. For creating this we will divide our data into two time series, one each for each station selected. Then for each of these time series, we create a new column which is the predicted value, and this column will be filled with available bike stand values exactly one hour before the current time step. The mean squared errors of this model for both stations is calculated separately and the two mean squared errors are averaged to find the overall mean squared error of our baseline model.

```
Mean squared error for baseline model: 12.289777649436903
```

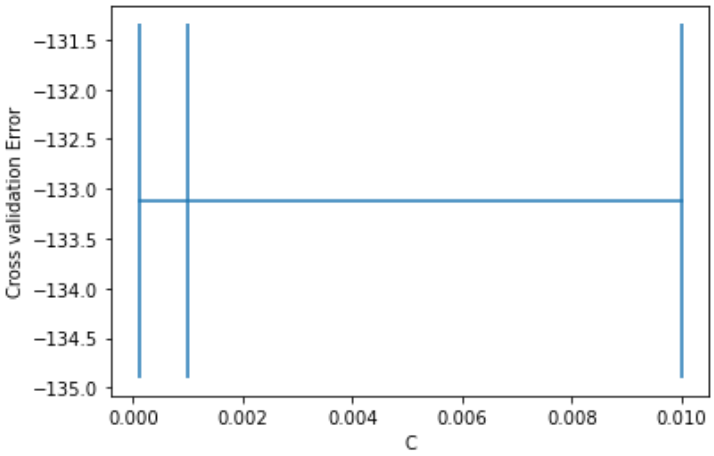
Linear Regression Model

To select correct value for hyperparameter C, we have performed cross validation for different values of C. We have plotted the mean squared error of test data vs C and cross validation error vs C. The plots are shown below:

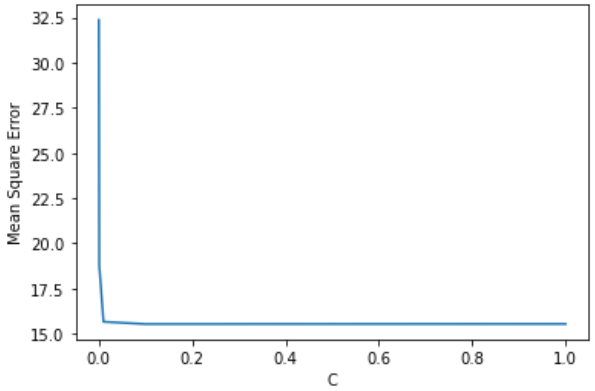
Lasso Regression (test data)



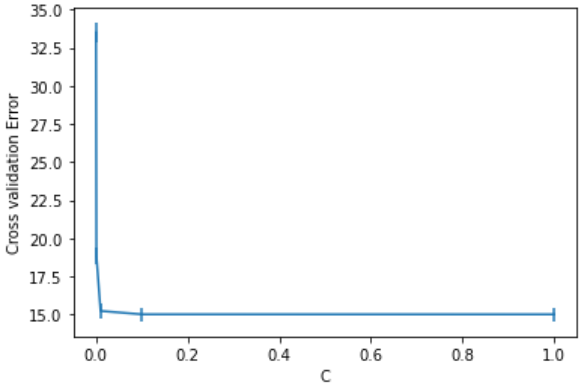
Lasso Regression (cross validation train data)



Ridge Regression (test data)

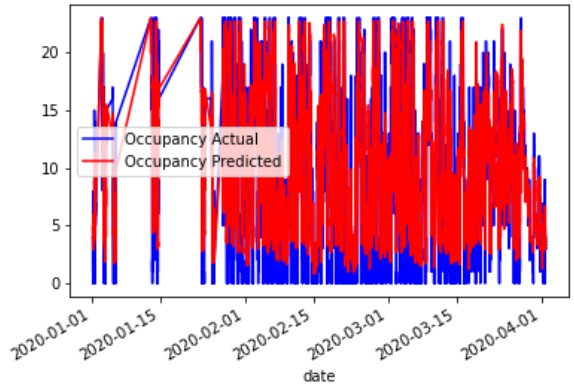


Ridge Regression (cross validation train data)

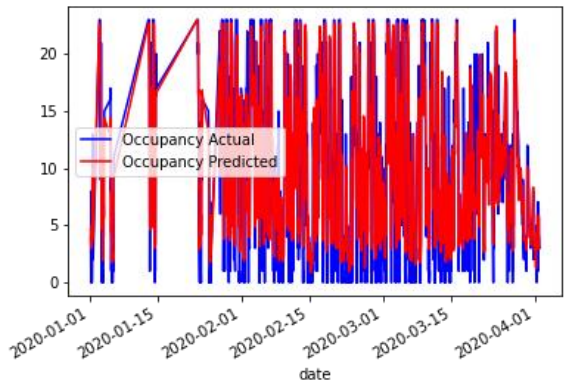


From the above plots, it is clear that the ridge regression has a much lower cross validation error on the train data and mean squared error on the test data, than the lasso regression. The mean squared error of test data as well as the cross-validation error on train data reduces drastically till C = 0.001 and stays almost the same above that. So, we have selected ridge regression model with C = 0.001 as our final hyperparameter. The plots of predictions of the model and actual data are shown below:

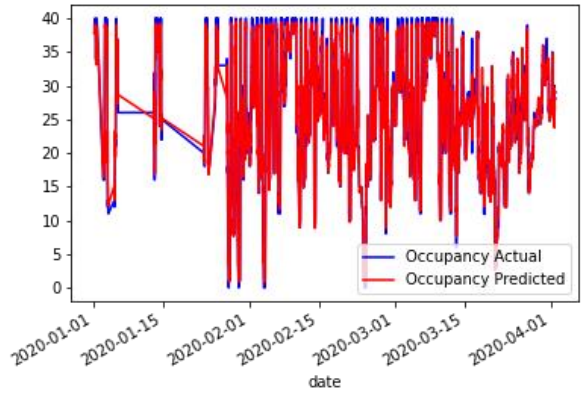
Train Data (Station 33)



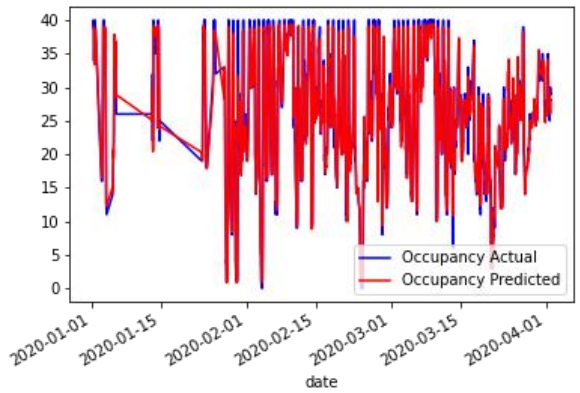
Test Data (Station 33)



Train Data (Station 107)



Test Data (Station 107)



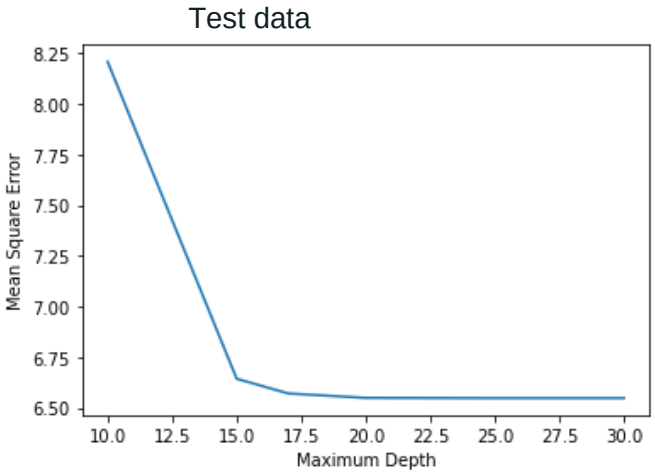
We have created a function pred_future which takes the time and station id as input and shows the predictions for occupancies at 10 minutes,30 minutes and 1 hour from that time. The output for time- '2020-03-01 10:50:02' and station 33 is shown below:

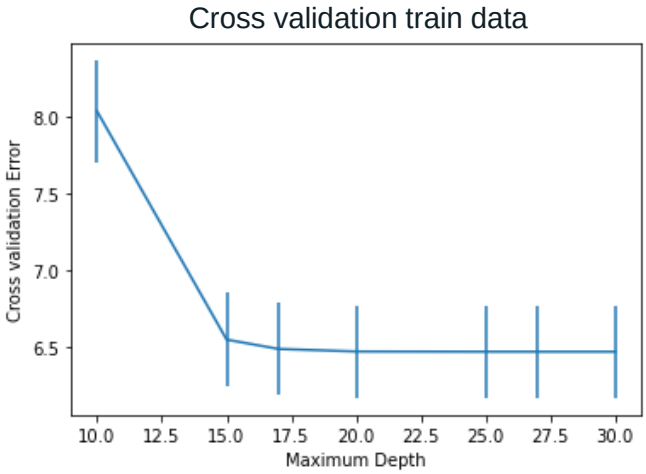
```
The occupancy of the station 33 for time: 2020-03-01 10:50:02 at 10 minutes futu
re is 8, 30 minutes future is 7 and 1 hour future is 6.
Mean squared error of test data on Ridge Regression is 18.772332
Mean squared error of train data on Ridge Regression is 18.496171
```

So, even the best hyperparameter has mean squared errors higher than our baseline model. We can clearly see from the plots that model has performed well for station 107 but failed to capture most of the high and low values for station 33. This means that we need to select another model to capture the non-linear relationships between the features and hence we moved on to Random Forest Regression.

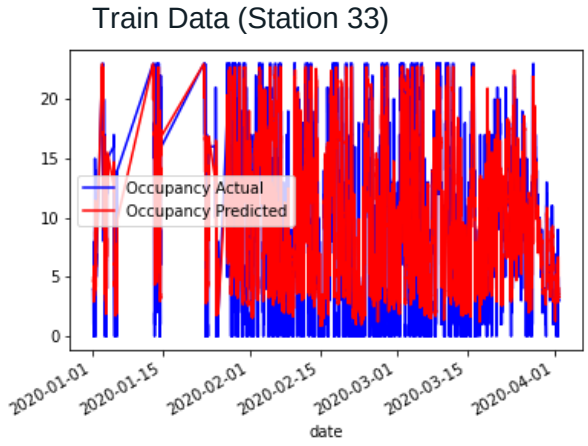
Random Forest Regressor

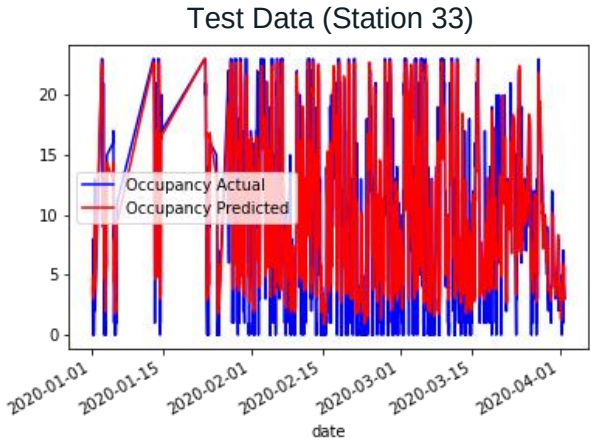
To select correct value for hyperparameter max_depth, we have performed cross validation for different values of max_depth. We have plotted the mean squared error of test data vs max_depth and cross validation error vs max_depth. The plots are shown below:

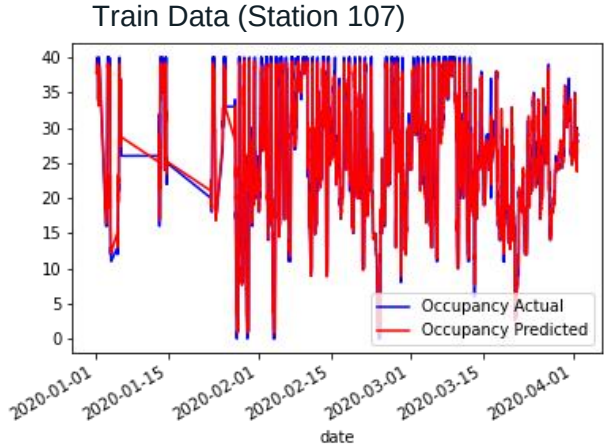


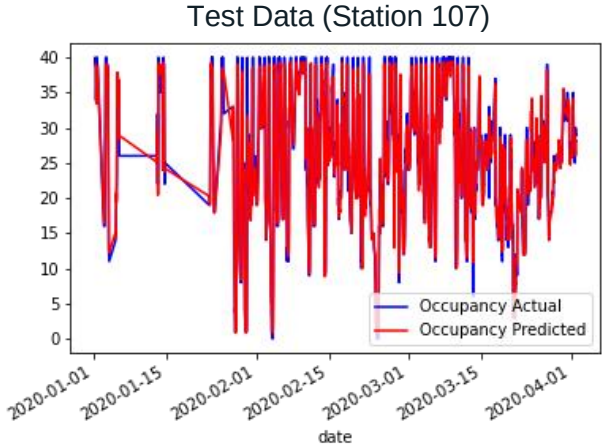


From the above plots, it is clear that the mean squared error of test data as well as the cross-validation error on train data reduces drastically till max_depth = 15 and stays almost the same above that. So, we have selected random forest regression model with max_depth = 15 as our final hyperparameter. We have selected one more hyperparameter of min_impurity_decrease= 0.008. This is because without this parameter, the train data error was going close to zero and the test data error was still significant, indicating overfitting. So, we increased the min_impurity_decrease parameter, so that the minimum impurity decreases for splitting of nodes increases thereby reducing the number of nodes and nodes with a smaller number of data points. This will prevent overfitting to some data points. The plots of predictions of the model and actual data are shown below:









We have created a function pred_future_rf which takes the time and station id as input and shows the predictions for occupancies at 10 minutes,30 minutes and 1 hour from that time. The output for time- '2020-03-01 10:50:02' and station 33 is shown below:

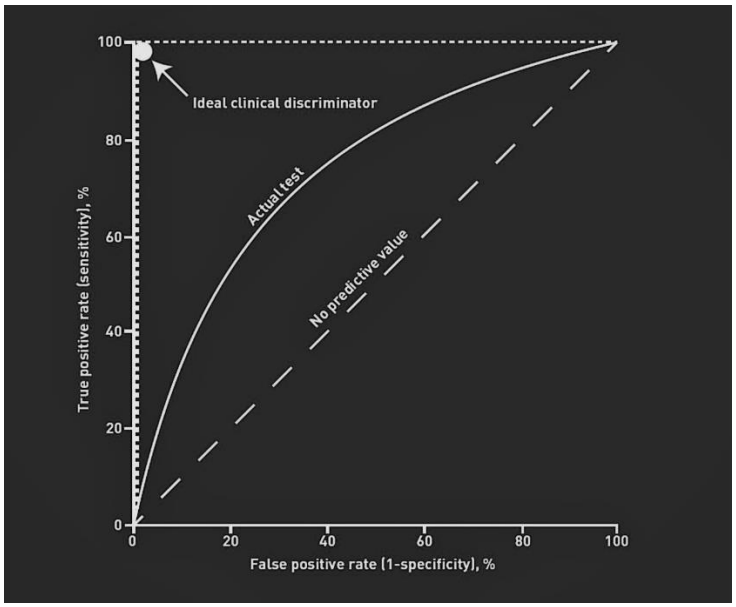
```
The occupancy of the station 33 for time: 2020-03-01 10:50:02  at 10 minutes future
is 5, 30 minutes future is 5 and 1 hour future is 5.
Mean squared error of test data on Random Forest Regression is 6.643016
Mean squared error of train data on Random Forest Regression is 5.206504
```

So, our random forest regression model has mean squared errors lower than our baseline model and linear model. We can clearly see from the plots that model has performed well for station 107 and improved for station 33.

Conclusion

Since the random forest regression model has the least mean squared errors without overfitting and the predictions seem to have captured most of the trends and seasonality of the time series data, we are selecting the random forest regression model as our final model.

2 (1) ROC stands for Receiver Operating characteristics. ROC curve is a graph with False Positive Rate (FPR) on the X axis and True Positive Rate (TPR) on the Y axis, plotted for different classification thresholds of a classification model. FPR = FP/(FP+TN) and TPR = TP/(TP+FN), where FP represents the number of data points falsely recognized as positive class by the classifier, TP represents the number of data points correctly recognized as positive class by the classifier, FN represents the number of data points falsely recognized as negative class by the classifier and TN represents the number of data points correctly recognized as negative class by the classifier. ROC is used to evaluate the performance of classifiers by calculating the area under the ROC curve (AUC). For a prediction engine classifying the data points randomly, the ROC curve will be a straight line at 45 degrees from the origin. So, the AUC value for random classifier is 0.5. So, any good classifier should have their AUC value between 0.5 and 1. The higher the AUC value better is the model's ability to differentiate positive and negative classes and hence better performance as a classifier. Examples are shown below:



(2) Linear regression works by trying to find the best fit line that has the least errors with the data points. The relationship between the input variables and the target is hence modelled with a linear equation. But this is not always the case in real life. Most data in real life does not have linear relationship with the target we are trying to predict. For example the target variable might be the square of sum of the input variables. In these cases, linear regression will model a linear equation between the input variables and target which will not capture the complete relationship between input variables and target and hence the linear regression would give inaccurate predictions.

Linear regression fails in scenarios when the training data has outliers in it. For example, a training dataset of age and salary, there might be some data points with very high salaries at a low age or people with very high ages and low salaries, but these are extreme cases. But, when a linear regression model is trained on this, the model tries to fit the line through these data point as well and thereby change the slope of the regression line and hence cause more errors when predicting on the normal data points. Thus, linear regression would give inaccurate predictions in this case.

(3) SVM classifiers identify the decision boundary with the help of support vectors (the data points from each class closest to the decision boundary being evaluated). So, the SVM classifier can train using subset of the complete data and get good results if the classes are well separated. It is hence memory efficient and computationally efficient. This is not the case for neural networks. Neural networks train on the input data in batches and hence the decision boundary it learns depends on the sequence in which the data was trained. So, the neural network will need to train on the whole dataset to reliably identify the decision boundary. Hence, the neural networks are not computationally efficient.

The number of parameters of an SVM classifier increases with the size of the input. The neural network parameters do not change with the increase in input. So, a neural network with the same number of parameters as an SVM will have a higher complexity than the SVM and will be able to perform better classification.

When the amount of data is very high, the training time for SVM is very high and it will not be able to give accurate results if the data includes noisy data. The neural networks on the other hand works well as the amount of train data increases. The predictions become more accurate and can handle noisy data as well.

Neural Networks use gradient descent as the optimization technique. Hence, they are sensitive to initial randomization of weight matrix. There is a chance that the neural network can get stuck at a local minimum because of the selection of initial weights. They will not be able to reach the global minima in such cases. SVM will reach the global minimum and does not depend on their initial configuration. Hence, SVMs are more reliable in such scenarios.

(4) Convolution layer is a layer made up of convolution filters (kernels). This layer is used as part of a neural network and can identify features from the input. Each convolution filter has a matrix of numerical values, which is convolved over (a sliding dot product of input matrix and kernel matrix) the input to generate new values(features) corresponding to the input. An example of convolution operation is shown below:
We have a grayscale image with the input vectors as shown below:

$$\begin{matrix}1&1&1&0&0\\0&1&1&1&0\\0&0&1&1&1\\0&0&1&1&0\\0&1&1&0&0\end{matrix} \quad \text{Kernel(filter)} = \begin{matrix}1&0&1\\0&1&0\\1&0&1\end{matrix}$$

So, the input image is of shape 5*5 and on applying convolution operation on that with a single filter of shape 3*3 as shown above, we take the dot product of the sub matrix of first 3 rows and first 3 column from the input image with the Kernel matrix resulting in a value of 4. Then, same operation is performed again for the sub matrix starting from second to fourth rows and columns of the input matrix and the kernel resulting in 3. This is continued to till the end of rows and columns of the input matrix. Here we used a stride of 1 (we moved the sub matrix windows by 1 column or row). So, continuing the same operation on the entire input matrix, the resulting output(feature) matrix will be of shape 3*3. Output = $\begin{matrix}4&3&4\\2&4&3\\2&3&4\end{matrix}$

Similarly, there will be many different filters (different matrix values) in one convolution layer. For example, if there are 5 3*3 filters applied on the input image, then the output from this layer will be of shape 3*3*5, where each filter would have found some defining features from the image like edges, sharpness, etc. In cases of neural networks, these kernels will be initialized with random values and the values will be learnt through backpropagation.

(5) In k-fold cross validation, the dataset available to us is divided into k (parameter specified) folds and in each run one of these folds is selected as the test set and the remaining folds are used for training. K such models will be trained by resampling the same data k times with the required hyperparameters to the model with different fold selected as the test set. This resampling allows us to evaluate the model performance on different subsets of the data and thereby avoid the model from overfitting to any specific subset of the data we used for training. This helps us evaluate the generalization performance of the model, because each run the model had used different train and test sets and hence is showing results free which are not overfitting to the data.

For example, we have an income dataset of 100 records with two columns age and income. When we use k-fold cross validation with k as 5 on this, the data set is split into 5 folds with each fold having 20 records in it. So, fold 1, fold 2, fold 3, fold 4 and fold 5 each have 20 records in it. Now, we are training a ridge regression model with C as 0.001 to evaluate its generalization performance. In first run, the fold 1 records will be kept aside as test data and the other folds are used for training. We can see the performance of this model while testing on fold 1 with any metrics such as mean squared error. The same process is repeated with fold 2, 3, 4 and 5 each used as test sets for 4 different runs, and we will get 4 more models and their evaluation scores (mean squared error). Now, if the model evaluation scores are good and does not show much variance for all the 5 models, this hyperparameter (C = 0.001) is appropriate. But, if the evaluation scores are showing a huge variance for the 5 models, this hyperparameter is not generalizing well and is working only for subsets of data, which is overfitting to the data. In that case, we need to change the hyperparameter to get better cross validation results, so that we can have model which generalizes well.

APPENDIX

Code:

```
import pandas as pd
import numpy as np
import math, sys
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import sklearn.linear_model as linear_model
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import OneHotEncoder
from sklearn.dummy import DummyRegressor
import time
from sklearn.svm import SVR
from sklearn.preprocessing import normalize



df = pd.read_csv(r'dublinbikes_20200101_20200401.csv',parse_dates=[1])


stations_selected = ["PRINCES STREET / O'CONNELL STREET",'CHARLEVILLE ROAD']
df1 = df[df['NAME'].isin(stations_selected)]
indices_original= df1.index
df1 = df1.iloc[:,[0,1,5]]


print('Plot of Station 33')
stations_selected = ["PRINCES STREET / O'CONNELL STREET"]
df_test = df[df['NAME'].isin(stations_selected)]
```

```
df_test = df_test.iloc[:,[1,5]]
df_test.index = df_test['TIME']
df_test = df_test.iloc[:,1]
plt.figure(figsize=(12,4))
df_test.plot()
plt.xlabel('Date')
plt.ylabel('Occupancy')
plt.show()

print('Plot of Station 107')
stations_selected = ['CHARLEVILLE ROAD']
df_test = df[df['NAME'].isin(stations_selected)]
df_test = df_test.iloc[:,[1,5]]
df_test.index = df_test['TIME']
df_test = df_test.iloc[:,1]
plt.figure(figsize=(12,4))
df_test.plot()
plt.xlabel('Date')
plt.ylabel('Occupancy')
plt.show()

unique_values = pd.DataFrame(df1['STATION ID'].unique())
encoded_values = pd.get_dummies(unique_values[0], prefix="STATION")
encoded_arr = pd.merge(left=unique_values,right=encoded_values,left_index=True,right_index = True)
encoded_arr = encoded_arr.rename(columns={0:"STATION ID"})
df2 = pd.merge(left=df1,right=encoded_arr)

df2['date'] = pd.to_datetime(df2['TIME'],format='%d-%m-%Y %H:%M')
df2.index = df2['date']
```

```
def generate_feature(station_id,date,n,unit):
    value = None
    new_date = date - pd.to_timedelta(n,unit=unit)
    #print('old date : {} and new date : {}'.format(date,new_date))
    data = df2[(df2['STATION ID'] == station_id)& (df2.index.date == new_date.date())& (abs((df2.index - new_date).total_seconds())<151)]
    #data = df2[(df2['STATION ID'] == station_id)& (df2['date'] == new_date)]
    if not data.empty:
        value = data['AVAILABLE BIKE STANDS'][0]
    return value


def impute_values(record,column_name):
    if np.isnan(record[column_name]):
        df_filtered = df2[df2['STATION ID'] == record['STATION ID']]
        mean_value = df_filtered[column_name].mean()
        return mean_value
    else:
        return record[column_name]


print('Feature Engineering in progress (Takes about an hour)')
df2["Day of the Month"] = df2.index.day
df2["Day of the Week"] = df2.index.weekday
df2["Hour of the Day"] = df2.index.hour
df2['hourly_1'] = df2.apply( lambda x: generate_feature(x['STATION ID'],x['date'],1,'h'), axis = 1)
df2['hourly_1'] = df2.apply( lambda x: impute_values(x,'hourly_1'), axis = 1)
df2['hourly_2'] = df2.apply( lambda x: generate_feature(x['STATION ID'],x['date'],2,'h'), axis = 1)
df2['hourly_2'] = df2.apply( lambda x: impute_values(x,'hourly_2'), axis = 1)
df2['hourly_3'] = df2.apply( lambda x: generate_feature(x['STATION ID'],x['date'],3,'h'), axis = 1)
df2['hourly_3'] = df2.apply( lambda x: impute_values(x,'hourly_3'), axis = 1)
```

```
df2['daily_1'] = df2.apply( lambda x: generate_feature(x['STATION ID'],x['date'],1,'d'), axis = 1)
df2['daily_1'] = df2.apply( lambda x: impute_values(x,'daily_1'), axis = 1)
df2['daily_2'] = df2.apply( lambda x: generate_feature(x['STATION ID'],x['date'],2,'d'), axis = 1)
df2['daily_2'] = df2.apply( lambda x: impute_values(x,'daily_2'), axis = 1)
df2['daily_3'] = df2.apply( lambda x: generate_feature(x['STATION ID'],x['date'],3,'d'), axis = 1)
df2['daily_3'] = df2.apply( lambda x: impute_values(x,'daily_3'), axis = 1)
df2['weekly_1'] = df2.apply( lambda x: generate_feature(x['STATION ID'],x['date'],1,'W'), axis = 1)
df2['weekly_1'] = df2.apply( lambda x: impute_values(x,'weekly_1'), axis = 1)
df2['weekly_2'] = df2.apply( lambda x: generate_feature(x['STATION ID'],x['date'],2,'W'), axis = 1)
df2['weekly_2'] = df2.apply( lambda x: impute_values(x,'weekly_2'), axis = 1)
df2['weekly_3'] = df2.apply( lambda x: generate_feature(x['STATION ID'],x['date'],3,'W'), axis = 1)
df2['weekly_3'] = df2.apply( lambda x: impute_values(x,'weekly_3'), axis = 1)
df2.drop(['STATION ID','TIME', 'date' , ], axis=1,inplace=True)


df2['date'] = df2.index
df2.index = indices_original
df_to_normalize=df2.iloc[:,3:15]
df_normalized = normalize(df_to_normalize,norm='max', axis=0)
df_normalized_df= pd.DataFrame(df_normalized,index= indices_original)
df_rest=df2.iloc[:,0:3]
df_new = pd.concat([df_rest, df_normalized_df], axis=1,join='inner')
df_new.index = df2['date']
df_new.columns= [ 'AVAILABLE BIKE STANDS', 'STATION_33', 'STATION_107',
      'Day of the Month', 'Day of the Week', 'Hour of the Day', 'hourly_1',
      'hourly_2', 'hourly_3', 'daily_1', 'daily_2', 'daily_3', 'weekly_1',
      'weekly_2', 'weekly_3']
df2 = df_new
df3 = df2.dropna()
print('Feature Engineering completed')
```

```
X = df3.iloc[:, 1:]
y=df3.loc[:, 'AVAILABLE BIKE STANDS']
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.10, random_state=42)

def naive_baseline_model(series,station_id):
    values = pd.DataFrame(series.values)
    dataframe = pd.concat([values.shift(12), values], axis=1)
    dataframe.columns = ['t-1', 't+1']

    # split into train and test sets
    X = dataframe.values
    train_size = int(len(X) * 0.66)
    train, test = X[1:train_size], X[train_size:]
    train_X, train_y = train[:,0], train[:,1]
    test_X, test_y = test[:,0], test[:,1]

    predictions = list()
    for x in test_X:
        yhat = x
        predictions.append(yhat)
    test_score = mean_squared_error(test_y, predictions)
    return test_score

def naive_baseline(series_1,series_2):
    mean_error = (naive_baseline_model(series_1,33) + naive_baseline_model(series_2,107))/2
    print('Mean squared error for baseline model: {}'.format(mean_error))

series_1 = y[X['STATION_33']==1]
```

```
series_2 = y[X['STATION_107']==1]

naive_baseline(series_1,series_2)


print("Lasso Linear Regression")

mean_err=[]

std_err=[]

scores_mean=[]

scores_std=[]

C=[0.0001,0.001,0.01]

for c in C:

    model = linear_model.Lasso(alpha=1/(2*c))

    model.fit(X_train, y_train)

    ypred = model.predict(X_test)

    mse = mean_squared_error(y_test, ypred)

    mean_err.append(np.array(mse).mean())

    std_err.append(np.array(mse).std())

    cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)

    scores = cross_val_score(model, X_train, y_train, scoring='neg_mean_squared_error', cv=cv, n_jobs=-1)

    #print('Cross Validation Scores Mean: {}'.format(np.array(scores).mean()))

    scores_mean.append(np.array(scores).mean())

    scores_std.append(np.array(scores).std())

    #print('Cross Validation Scores Standard Deviation: {}'.format(np.array(scores).std()))

plot1 = plt.figure(1)

plt.errorbar( C,mean_err, yerr=std_err)

plt.xlabel('C')

plt.ylabel('Mean Square Error')

plot2 = plt.figure(2)

plt.errorbar( C,scores_mean, yerr=scores_std)

plt.xlabel('C')
```

```
plt.ylabel('Cross validation Error')
plt.show()


print("Ridge Linear Regression")
mean_err=[]
std_err=[]
scores_mean=[]
scores_std=[]
C=[0.0001,0.001,0.01,0.1,1]
for c in C:
    model = linear_model.Ridge(alpha=1/(2*c))
    model.fit(X_train, y_train)
    ypred = model.predict(X_test)
    mse = mean_squared_error(y_test, ypred)
    mean_err.append(np.array(mse).mean())
    std_err.append(np.array(mse).std())
    cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
    scores = cross_val_score(model, X_train, y_train, scoring='neg_mean_squared_error', cv=cv, n_jobs=-1)
    #print('Cross Validation Scores Mean: {}'.format(np.array(scores).mean()))
    scores_mean.append(np.array(scores).mean()*-1)
    scores_std.append(np.array(scores).std())
    #print('Cross Validation Scores Standard Deviation: {}'.format(np.array(scores).std()))
plot1 = plt.figure(1)
plt.errorbar( C,mean_err, yerr=std_err)
plt.xlabel('C')
plt.ylabel('Mean Square Error')
plot2 = plt.figure(2)
plt.errorbar( C,scores_mean, yerr=scores_std)
plt.xlabel('C')
```

```
plt.ylabel('Cross validation Error')

plt.show()


c=0.001

linear_reg_model = linear_model.Ridge(alpha=1/(2*c)).fit( X_train, y_train)

y_pred = linear_reg_model.predict(X_test)

y_train_pred = linear_reg_model.predict(X_train)

print("Mean squared error of test data on Ridge Regression is %f"%(mean_squared_error(y_test,y_pred)))

print("Mean squared error of train data on Ridge Regression is
%f"%(mean_squared_error(y_train,y_train_pred)))


DayOfMonth = np.sort(X_test['Day of the Month'].unique())

DayOfWeek = np.sort(X_test['Day of the Week'].unique())

HourOfDay = np.sort(X_test['Hour of the Day'].unique())


# Plot of train predictions station 33

print('Plot of train predictions station 33')

df_plot = pd.DataFrame()

df_plot_pred = pd.DataFrame()

df_plot['Occupancy'] = y_train[X_train['STATION_33']==1]

df_plot_pred['Occupancy'] = y_train_pred[X_train['STATION_33']==1]

df_plot.index = y_train[X_train['STATION_33']==1].index

df_plot_pred.index = y_train[X_train['STATION_33']==1].index

fig, ax = plt.subplots()

df_plot.plot(c='blue', ax = ax)

df_plot_pred.plot(c='red', ax = ax)

ax.legend( ['Occupancy Actual', 'Occupancy Predicted'])

plt.show()


# Plot of train predictions station 107
```

```
print('Plot of train predictions station 107')
df_plot = pd.DataFrame()
df_plot_pred = pd.DataFrame()
df_plot['Occupancy'] = y_train[X_train['STATION_107']==1]
df_plot_pred['Occupancy'] = y_train_pred[X_train['STATION_107']==1]
df_plot.index = y_train[X_train['STATION_107']==1].index
df_plot_pred.index = y_train[X_train['STATION_107']==1].index
fig, ax = plt.subplots()
df_plot.plot(c='blue', ax = ax)
df_plot_pred.plot(c='red', ax = ax)
ax.legend( ['Occupancy Actual', 'Occupancy Predicted'])
plt.show()

# Plot of test predictions station 33
print('Plot of test predictions station 33')
df_plot = pd.DataFrame()
df_plot_pred = pd.DataFrame()
df_plot['Occupancy'] = y_test[X_test['STATION_33']==1]
df_plot_pred['Occupancy'] = y_pred[X_test['STATION_33']==1]
df_plot.index = y_test[X_test['STATION_33']==1].index
df_plot_pred.index = y_test[X_test['STATION_33']==1].index
fig, ax = plt.subplots()
df_plot.plot(c='blue', ax = ax)
df_plot_pred.plot(c='red', ax = ax)
ax.legend( ['Occupancy Actual', 'Occupancy Predicted'])
plt.show()

# Plot of test predictions station 107
print('Plot of test predictions station 107')
```

```
df_plot = pd.DataFrame()

df_plot_pred = pd.DataFrame()

df_plot['Occupancy'] = y_test[X_test['STATION_107']==1]

df_plot_pred['Occupancy'] = y_pred[X_test['STATION_107']==1]

df_plot.index = y_test[X_test['STATION_107']==1].index

df_plot_pred.index = y_test[X_test['STATION_107']==1].index

fig, ax = plt.subplots()

df_plot.plot(c='blue', ax = ax)

df_plot_pred.plot(c='red', ax = ax)

ax.legend( ['Occupancy Actual', 'Occupancy Predicted'])

plt.show()


df2.index = pd.to_datetime(df2.index,format='%Y-%m-%d %H:%M')


def gen_Value(station_id,date,n,unit,column_name):

    value = None

    new_date = date - pd.to_timedelta(n,unit=unit)

    station_name = 'STATION_'+str(station_id)

    #print('old date : {} and new date : {}'.format(date,new_date))

    if station_id ==107:

        data = df2[(df2['STATION_107'] == 1)& (df2.index.date == new_date.date())& (abs((df2.index -
new_date).total_seconds())<151)]

    elif station_id == 33:

        data = df2[(df2['STATION_33'] == 1)& (df2.index.date == new_date.date())& (abs((df2.index -
new_date).total_seconds())<151)]

    if data.empty:

        pass

    else:

        value = data[column_name][0]

    if value == None:
```

```
        df_filtered = df2[df2[station_name] == 1]
        mean_value = df_filtered[column_name].mean()
        value = mean_value
    return value


def pred_future(date,station_id):
    df_new = pd.DataFrame(data= [[0,0,0,0,0,0,0,0,0,0,0,0,0,0]],columns=['STATION_33',
'STATION_107','Day of the Month','Day of the Week','Hour of the Day', 'hourly_1', 'hourly_2', 'hourly_3',
       'daily_1', 'daily_2', 'daily_3', 'weekly_1', 'weekly_2', 'weekly_3'])
    time = pd.to_datetime(date,format='%Y-%m-%d %H:%M')
    time.replace(second=0)


    if station_id == 33:
        df_new['STATION_33'] = 1
    elif station_id == 107 :
        df_new['STATION_107'] = 1




    # 10 minute future
    new_time = time + pd.to_timedelta(10,unit='minutes')
    #df_new['Weekday'] = generate_weekdays(new_time)
    #df_new['Weekend'] = generate_weekend(new_time)
    df_new[''Day of the Month''] = DayOfMonth[new_time.day]
    df_new[''Day of the Week''] = DayOfWeek[new_time.weekday()]
    df_new[''Hour of the Day''] = HourOfDay[new_time.hour]
    df_new['hourly_1'] = gen_Value(station_id,new_time,1,'h','hourly_1')
    df_new['hourly_2'] = gen_Value(station_id,new_time,2,'h','hourly_2')
    df_new['hourly_3'] = gen_Value(station_id,new_time,3,'h','hourly_3')
    df_new['daily_1'] = gen_Value(station_id,new_time,1,'d','daily_1')
```

```
df_new['daily_2'] = gen_Value(station_id,new_time,2,'d','daily_2')
df_new['daily_3'] = gen_Value(station_id,new_time,3,'d','daily_3')
df_new['weekly_1'] = gen_Value(station_id,new_time,1,'W','weekly_1')
df_new['weekly_2'] = gen_Value(station_id,new_time,2,'W','weekly_2')
df_new['weekly_3'] = gen_Value(station_id,new_time,3,'W','weekly_3')
bikes_available_in_10minutes = linear_reg_model.predict(df_new)

# 30 minute future
new_time = time + pd.to_timedelta(30,unit='minutes')
#df_new['Weekday'] = generate_weekdays(new_time)
#df_new['Weekend'] = generate_weekend(new_time)
df_new["Day of the Month"] = DayOfMonth[new_time.day]
df_new["Day of the Week"] = DayOfWeek[new_time.weekday()]
df_new["Hour of the Day"] = HourOfDay[new_time.hour]
df_new['hourly_1'] = gen_Value(station_id,new_time,1,'h','hourly_1')
df_new['hourly_2'] = gen_Value(station_id,new_time,2,'h','hourly_2')
df_new['hourly_3'] = gen_Value(station_id,new_time,3,'h','hourly_3')
df_new['daily_1'] = gen_Value(station_id,new_time,1,'d','daily_1')
df_new['daily_2'] = gen_Value(station_id,new_time,2,'d','daily_2')
df_new['daily_3'] = gen_Value(station_id,new_time,3,'d','daily_3')
df_new['weekly_1'] = gen_Value(station_id,new_time,1,'W','weekly_1')
df_new['weekly_2'] = gen_Value(station_id,new_time,2,'W','weekly_2')
df_new['weekly_3'] = gen_Value(station_id,new_time,3,'W','weekly_3')
bikes_available_in_30minutes = linear_reg_model.predict(df_new)

# 1 hour future
new_time = time + pd.to_timedelta(1,unit='h')
#df_new['Weekday'] = generate_weekdays(new_time)
#df_new['Weekend'] = generate_weekend(new_time)
```

```
    df_new["Day of the Month"] = DayOfMonth[new_time.day]
    df_new["Day of the Week"] = DayOfWeek[new_time.weekday()]
    df_new["Hour of the Day"] = HourOfDay[new_time.hour]
    df_new['hourly_1'] = gen_Value(station_id,new_time,1,'h','hourly_1')
    df_new['hourly_2'] = gen_Value(station_id,new_time,2,'h','hourly_2')
    df_new['hourly_3'] = gen_Value(station_id,new_time,3,'h','hourly_3')
    df_new['daily_1'] = gen_Value(station_id,new_time,1,'d','daily_1')
    df_new['daily_2'] = gen_Value(station_id,new_time,2,'d','daily_2')
    df_new['daily_3'] = gen_Value(station_id,new_time,3,'d','daily_3')
    df_new['weekly_1'] = gen_Value(station_id,new_time,1,'W','weekly_1')
    df_new['weekly_2'] = gen_Value(station_id,new_time,2,'W','weekly_2')
    df_new['weekly_3'] = gen_Value(station_id,new_time,3,'W','weekly_3')
    bikes_available_in_1hour = linear_reg_model.predict(df_new)

    return
round(bikes_available_in_10minutes[0]),round(bikes_available_in_30minutes[0]),round(bikes_available_in_1
hour[0])

station = 33
time = '2020-03-01 10:50:02'
x,y,z = pred_future(time,station)
print('Linear Regression Prediction:')
print(f"The occupancy of the station {station} for time: {time}  at 10 minutes future is {x}, 30 minutes future is
{y} and 1 hour future is {z}.")

print("Random Forest Regression")
mean_err=[]
std_err=[]
scores_mean=[]
scores_std=[]
max_depth=[10,15,17,20,25,27,30]
```

```
for m in max_depth:
    model = RandomForestRegressor(max_depth=m, random_state=0, min_impurity_decrease= 0.008)
    model.fit(X_train, y_train)
    ypred = model.predict(X_test)
    mse = mean_squared_error(y_test, ypred)
    mean_err.append(np.array(mse).mean())
    std_err.append(np.array(mse).std())
    cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
    scores = cross_val_score(model, X_train, y_train, scoring='neg_mean_squared_error', cv=cv, n_jobs=-1)
    scores_mean.append(np.array(scores).mean()*-1)
    scores_std.append(np.array(scores).std())
plot1 = plt.figure(1)
plt.errorbar( max_depth,mean_err, yerr=std_err)
plt.xlabel('Maximum Depth')
plt.ylabel('Mean Square Error')
plot2 = plt.figure(2)
plt.errorbar( max_depth,scores_mean, yerr=scores_std)
plt.xlabel('Maximum Depth')
plt.ylabel('Cross validation Error')
plt.show()


regr = RandomForestRegressor(max_depth=15, random_state=0, min_impurity_decrease= 0.008) #,
min_impurity_decrease= 0.005, max_features = 10 )
regr.fit(X_train, y_train)
y_pred = regr.predict(X_test)
y_train_pred = regr.predict(X_train)
print("Mean squared error of test data on Random Forest Regression is
%f"%(mean_squared_error(y_test,y_pred)))
print("Mean squared error of train data on Random Forest Regression is
%f"%(mean_squared_error(y_train,y_train_pred)))
```

```
# Plot of train predictions station 33
print('Plot of train predictions station 33')
df_plot = pd.DataFrame()
df_plot_pred = pd.DataFrame()
df_plot['Occupancy'] = y_train[X_train['STATION_33']==1]
df_plot_pred['Occupancy'] = y_train_pred[X_train['STATION_33']==1]
df_plot.index = y_train[X_train['STATION_33']==1].index
df_plot_pred.index = y_train[X_train['STATION_33']==1].index
fig, ax = plt.subplots()
df_plot.plot(c='blue', ax = ax)
df_plot_pred.plot(c='red', ax = ax)
ax.legend( ['Occupancy Actual', 'Occupancy Predicted'])
plt.show()

# Plot of train predictions station 107
print('Plot of train predictions station 107')
df_plot = pd.DataFrame()
df_plot_pred = pd.DataFrame()
df_plot['Occupancy'] = y_train[X_train['STATION_107']==1]
df_plot_pred['Occupancy'] = y_train_pred[X_train['STATION_107']==1]
df_plot.index = y_train[X_train['STATION_107']==1].index
df_plot_pred.index = y_train[X_train['STATION_107']==1].index
fig, ax = plt.subplots()
df_plot.plot(c='blue', ax = ax)
df_plot_pred.plot(c='red', ax = ax)
ax.legend( ['Occupancy Actual', 'Occupancy Predicted'])
plt.show()

# Plot of test predictions station 33
```

```
print('Plot of test predictions station 33')

df_plot = pd.DataFrame()

df_plot_pred = pd.DataFrame()

df_plot['Occupancy'] = y_test[X_test['STATION_33']==1]

df_plot_pred['Occupancy'] = y_pred[X_test['STATION_33']==1]

df_plot.index = y_test[X_test['STATION_33']==1].index

df_plot_pred.index = y_test[X_test['STATION_33']==1].index

fig, ax = plt.subplots()

df_plot.plot(c='blue', ax = ax)

df_plot_pred.plot(c='red', ax = ax)

ax.legend( ['Occupancy Actual', 'Occupancy Predicted'])

plt.show()


# Plot of test predictions station 107

print('Plot of test predictions station 107')

df_plot = pd.DataFrame()

df_plot_pred = pd.DataFrame()

df_plot['Occupancy'] = y_test[X_test['STATION_107']==1]

df_plot_pred['Occupancy'] = y_pred[X_test['STATION_107']==1]

df_plot.index = y_test[X_test['STATION_107']==1].index

df_plot_pred.index = y_test[X_test['STATION_107']==1].index

fig, ax = plt.subplots()

df_plot.plot(c='blue', ax = ax)

df_plot_pred.plot(c='red', ax = ax)

ax.legend( ['Occupancy Actual', 'Occupancy Predicted'])

plt.show()


def pred_future_rf(date,station_id):

    df_new = pd.DataFrame(data= [[0,0,0,0,0,0,0,0,0,0,0,0,0,0]],columns=['STATION_33',
'STATION_107','Day of the Month','Day of the Week','Hour of the Day', 'hourly_1', 'hourly_2', 'hourly_3',
```

```
   'daily_1', 'daily_2', 'daily_3', 'weekly_1', 'weekly_2', 'weekly_3'])
time = pd.to_datetime(date,format='%Y-%m-%d %H:%M')
time.replace(second=0)

if station_id == 33:
    df_new['STATION_33'] = 1
elif station_id == 107 :
    df_new['STATION_107'] = 1



# 10 minute future
new_time = time + pd.to_timedelta(10,unit='minutes')
#df_new['Weekday'] = generate_weekdays(new_time)
#df_new['Weekend'] = generate_weekend(new_time)
df_new[''Day of the Month''] = DayOfMonth[new_time.day]
df_new[''Day of the Week''] = DayOfWeek[new_time.weekday()]
df_new[''Hour of the Day''] = HourOfDay[new_time.hour]
df_new['hourly_1'] = gen_Value(station_id,new_time,1,'h','hourly_1')
df_new['hourly_2'] = gen_Value(station_id,new_time,2,'h','hourly_2')
df_new['hourly_3'] = gen_Value(station_id,new_time,3,'h','hourly_3')
df_new['daily_1'] = gen_Value(station_id,new_time,1,'d','daily_1')
df_new['daily_2'] = gen_Value(station_id,new_time,2,'d','daily_2')
df_new['daily_3'] = gen_Value(station_id,new_time,3,'d','daily_3')
df_new['weekly_1'] = gen_Value(station_id,new_time,1,'W','weekly_1')
df_new['weekly_2'] = gen_Value(station_id,new_time,2,'W','weekly_2')
df_new['weekly_3'] = gen_Value(station_id,new_time,3,'W','weekly_3')
bikes_available_in_10minutes = regr.predict(df_new)
```

```
# 30 minute future
new_time = time + pd.to_timedelta(30,unit='minutes')
#df_new['Weekday'] = generate_weekdays(new_time)
#df_new['Weekend'] = generate_weekend(new_time)
df_new["Day of the Month"] = DayOfMonth[new_time.day]
df_new["Day of the Week"] = DayOfWeek[new_time.weekday()]
df_new["Hour of the Day"] = HourOfDay[new_time.hour]
df_new['hourly_1'] = gen_Value(station_id,new_time,1,'h','hourly_1')
df_new['hourly_2'] = gen_Value(station_id,new_time,2,'h','hourly_2')
df_new['hourly_3'] = gen_Value(station_id,new_time,3,'h','hourly_3')
df_new['daily_1'] = gen_Value(station_id,new_time,1,'d','daily_1')
df_new['daily_2'] = gen_Value(station_id,new_time,2,'d','daily_2')
df_new['daily_3'] = gen_Value(station_id,new_time,3,'d','daily_3')
df_new['weekly_1'] = gen_Value(station_id,new_time,1,'W','weekly_1')
df_new['weekly_2'] = gen_Value(station_id,new_time,2,'W','weekly_2')
df_new['weekly_3'] = gen_Value(station_id,new_time,3,'W','weekly_3')
bikes_available_in_30minutes = regr.predict(df_new)

# 1 hour future
new_time = time + pd.to_timedelta(1,unit='h')
#df_new['Weekday'] = generate_weekdays(new_time)
#df_new['Weekend'] = generate_weekend(new_time)
df_new["Day of the Month"] = DayOfMonth[new_time.day]
df_new["Day of the Week"] = DayOfWeek[new_time.weekday()]
df_new["Hour of the Day"] = HourOfDay[new_time.hour]
df_new['hourly_1'] = gen_Value(station_id,new_time,1,'h','hourly_1')
df_new['hourly_2'] = gen_Value(station_id,new_time,2,'h','hourly_2')
df_new['hourly_3'] = gen_Value(station_id,new_time,3,'h','hourly_3')
df_new['daily_1'] = gen_Value(station_id,new_time,1,'d','daily_1')
```

```
    df_new['daily_2'] = gen_Value(station_id,new_time,2,'d','daily_2')

    df_new['daily_3'] = gen_Value(station_id,new_time,3,'d','daily_3')

    df_new['weekly_1'] = gen_Value(station_id,new_time,1,'W','weekly_1')

    df_new['weekly_2'] = gen_Value(station_id,new_time,2,'W','weekly_2')

    df_new['weekly_3'] = gen_Value(station_id,new_time,3,'W','weekly_3')

    bikes_available_in_1hour = regr.predict(df_new)



    return
round(bikes_available_in_10minutes[0]),round(bikes_available_in_30minutes[0]),round(bikes_available_in_1
hour[0])



station = 33

time = '2020-03-01 10:50:02'

x,y,z = pred_future_rf(time,station)

print('Random Forest Regression Prediction:')

print(f"The occupancy of the station {station} for time: {time} at 10 minutes future is {x}, 30 minutes future is
{y} and 1 future is {z}.")
```