Training models below using RandomForest Algorithm

In [1]:

```
# Import pandas, numpy
import pandas as pd
import numpy as np
import pymysql

from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.model_selection import cross_validate
from sklearn.model_selection import cross_val_score

# Import the model we are using
from sklearn.ensemble import RandomForestRegressor

import pickle
```

In [2]:

```
# connect to db on rds
conn = pymysql.connect(
    host="dublin-bikes.c96ersz2ktrh.us-east-1.rds.amazonaws.com",
    port=int(3306),
    user="root",
    passwd="dublin_bikes_root",
    db="dublin_bikes")
```

In [3]:

```
# Getting all the stations
df_static = pd.read_sql_query("SELECT * FROM static_bike_details order by number asc", conn)
df_static
```

Out[3]:

	number	contract_name	name	address	lat	lng	banking	bonus	bike_stands	date_created
0	2	dublin	BLESSINGTON STREET	Blessington Street	53.356769	-6.26814	True	False	20	2020-02-15 15:58:21
1	3	dublin	BOLTON STREET	Bolton Street	53.351182	-6.269859	False	False	20	2020-02-15 15:58:31
2	4	dublin	GREEK STREET	Greek Street	53.346874	-6.272976	False	False	20	2020-02-15 15:51:28
3	5	dublin	CHARLEMONT PLACE	Charlemont Street	53.330662	-6.260177	False	False	40	2020-02-15 15:51:26
4	6	dublin	CHRISTCHURCH PLACE	Christchurch Place	53.343368	-6.27012	False	False	20	2020-02-15 15:51:10
...	...	...	...	...	...	...	...	...	...	...
105	113	dublin	MERRION SQUARE SOUTH	Merrion Square South	53.338614	-6.248606	True	False	40	2020-02-15 15:51:22
106	114	dublin	WILTON TERRACE (PARK)	Wilton Terrace (Park)	53.333653	-6.248345	True	False	40	2020-02-15 15:51:21
107	115	dublin	KILLARNEY STREET	Killarney Street	53.354845	-6.247579	False	False	30	2020-02-15 15:51:34
108	116	dublin	BROADSTONE	Broadstone	53.3547	-6.272314	True	False	30	2020-02-27 12:11:43
109	117	dublin	HANOVER QUAY EAST	Hanover Quay East	53.343653	-6.231755	False	False	40	2020-02-15 15:51:35

110 rows × 10 columns

In [4]:

```
#This function is used repeatedly to compute all metrics
def printMetrics(testActualVal, predictions):
    #classification evaluation measures
    print('\n===============================================================================')
    print("MAE: ", metrics.mean_absolute_error(testActualVal, predictions))
    #print("MSE: ", metrics.mean_squared_error(testActualVal, predictions))
    print("RMSE: ", metrics.mean_squared_error(testActualVal, predictions)**0.5)
    print("R2: ", metrics.r2_score(testActualVal, predictions))
```

Train model for available_bikes per station

In [19]:

```
for num in df_static['number']:
    df = pd.read_sql_query("SELECT * FROM dynamic_bike_details where number = "+str(num)+" order by date

    # Getting weather
    temperature_list = []
    weather_list = []
    for val in df['date_created']:
        df2 = pd.read_sql_query('SELECT * FROM weather_details where date_created <= "'+str(val)+'" orde
        if not df2.empty:
            temperature_list.append(df2.iloc[0]['temperature'])
            weather_list.append(df2.iloc[0]['weather_description'])
        else:
            temperature_list.append("")
            weather_list.append("")
    df['temperature'] = temperature_list
    df['weather_description'] = weather_list

    # Drop blank values
    df['temperature'].replace('', np.nan, inplace = True)
    df.dropna(subset=['temperature'], inplace = True)

    # Create dummies for weather_description
```

```
weather_description_dummies = pd.get_dummies(df['weather_description'], prefix='weather_description'
df = pd.concat([df, weather_description_dummies], axis=1)
df = df.drop('weather_description', axis = 1)

# Extract new features from date_created and create dummies wherever required
df['day_no'] = df.date_created.dt.day
df['month'] = df.date_created.dt.month
df['hours'] = df.date_created.dt.hour
df['minutes'] = df.date_created.dt.minute

time_of_day = []
for hr in df['hours']:
    if hr >= 0 and hr <=3:
        time_of_day.append("Night")
    elif hr >= 4 and hr <=11:
        time_of_day.append("Morning")
    elif hr >= 12 and hr <=20:
        time_of_day.append("Afternoon")
    else:
        time_of_day.append("Night")
df['time_of_day'] = time_of_day

time_of_day_dummies = pd.get_dummies(df['time_of_day'], prefix='time_of_day', drop_first=True)
df = pd.concat([df, time_of_day_dummies], axis=1)
df = df.drop('time_of_day', axis = 1)

df['day'] = df.date_created.dt.strftime("%A")
day_dummies = pd.get_dummies(df['day'], prefix='day', drop_first=True)

df = pd.concat([df, day_dummies], axis=1)
df = df.drop('day', axis = 1)

df["temperature"]=df["temperature"].astype(int)
df["available_bikes"]=df["available_bikes"].astype(int)
features = df[['day_no','month', 'hours','minutes','time_of_day_Morning', 'time_of_day_Night',
                   'day_Monday', 'day_Saturday', 'day_Sunday','day_Thursday','day_Tuesday','day_Wedr
```

```
                       'day_Monday', 'day_Saturday', 'day_Sunday', 'day_Thursday', 'day_Tuesday', 'day_Wed
                  'temperature',
    'weather_description_drizzle',
    'weather_description_few clouds',
    'weather_description_light intensity drizzle',
    'weather_description_light intensity drizzle rain',
    'weather_description_light intensity shower rain',
    'weather_description_light rain',
    'weather_description_moderate rain',
    'weather_description_overcast clouds',
    'weather_description_scattered clouds',
    'weather_description_shower rain']]

    X = features
    y = df.available_bikes

    train_features, test_features, train_labels, test_labels = train_test_split(X, y,  test_size=0.3, ra

    # Instantiate model with 10 decision trees
    rf = RandomForestRegressor(n_estimators = 10)
    # Train the model on training data
    rf.fit(train_features, train_labels);

    predictions = rf.predict(test_features)
    printMetrics(test_labels, predictions)
    print("\n\n")

    # Serialize model object into a file called model.pkl on disk using pickle
    file_name = "available_bikes_" + str(num) + ".pkl"
    with open(file_name, 'wb') as handle:
         pickle.dump(rf, handle, pickle.HIGHEST_PROTOCOL)
#     break
```

```
==============================================================================
MAE:  0.45439560439560434
```

```
RMSE:  0.915900445667337
R2:  0.9686902712988777
```

Test available_bikes model

In [22]:

```
with open('available_bikes/available_bikes_2.pkl', 'rb') as handle:
    model = pickle.load(handle)
datapoint = pd.DataFrame({'day_no': [3], 'month': [4], 'hours':[16],'minutes':[30],
                         "time_of_day_Morning": [0], "time_of_day_Night": [0],
                         'day_Monday': [0], 'day_Saturday': [0], 'day_Sunday' : [0], 'day_Thursday':[0],
                         'day_Tuesday':[0], 'day_Wednesday':[0],
                         'temperature':[6],
'weather_description_drizzle':[0],
'weather_description_few clouds':[0],
'weather_description_light intensity drizzle':[0],
'weather_description_light intensity drizzle rain':[0],
'weather_description_light intensity shower rain':[0],
'weather_description_light rain':[0],
'weather_description_moderate rain':[0],
'weather_description_overcast clouds':[0],
'weather_description_scattered clouds':[0],
'weather_description_shower rain':[0]})

test_predictions = model.predict(datapoint)
test_predictions
```

Out[22]:

```
array([2.9])
```

Train model for available_bike_stands per station

In [13]:

```
for num in df_static['number']:
    df = pd.read_sql_query("SELECT * FROM dynamic_bike_details where number = "+str(num)+" order by date

    # Getting weather
    temperature_list = []
    weather_list = []
    for val in df['date_created']:
        df2 = pd.read_sql_query('SELECT * FROM weather_details where date_created <= "'+str(val)+'" orde
        if not df2.empty:
            temperature_list.append(df2.iloc[0]['temperature'])
            weather_list.append(df2.iloc[0]['weather_description'])
        else:
            temperature_list.append("")
            weather_list.append("")
    df['temperature'] = temperature_list
    df['weather_description'] = weather_list

    # Drop blank values
    df['temperature'].replace('', np.nan, inplace = True)
    df.dropna(subset=['temperature'], inplace = True)

    # Create dummies for weather_description
    weather_description_dummies = pd.get_dummies(df['weather_description'], prefix='weather_description'
    df = pd.concat([df, weather_description_dummies], axis=1)
    df = df.drop('weather_description', axis = 1)

    # Extract new features from date_created and create dummies wherever required
    df['day_no'] = df.date_created.dt.day
    df['month'] = df.date_created.dt.month
    df['hours'] = df.date_created.dt.hour
    df['minutes'] = df.date_created.dt.minute
```

```
time_of_day = []
for hr in df['hours']:
    if hr >= 0 and hr <=3:
        time_of_day.append("Night")
    elif hr >= 4 and hr <=11:
        time_of_day.append("Morning")
    elif hr >= 12 and hr <=20:
        time_of_day.append("Afternoon")
    else:
        time_of_day.append("Night")
df['time_of_day'] = time_of_day

time_of_day_dummies = pd.get_dummies(df['time_of_day'], prefix='time_of_day', drop_first=True)
df = pd.concat([df, time_of_day_dummies], axis=1)
df = df.drop('time_of_day', axis = 1)

df['day'] = df.date_created.dt.strftime("%A")
day_dummies = pd.get_dummies(df['day'], prefix='day', drop_first=True)

df = pd.concat([df, day_dummies], axis=1)
df = df.drop('day', axis = 1)

df["temperature"]=df["temperature"].astype(int)
df["available_bike_stands"]=df["available_bike_stands"].astype(int)
features = df[['day_no','month', 'hours','minutes','time_of_day_Morning', 'time_of_day_Night',
                   'day_Monday', 'day_Saturday', 'day_Sunday','day_Thursday','day_Tuesday','day_Wedr
              'temperature',
'weather_description_drizzle',
'weather_description_few clouds',
'weather_description_light intensity drizzle',
'weather_description_light intensity drizzle rain',
'weather_description_light intensity shower rain',
'weather_description_light rain',
'weather_description_moderate rain',
'weather_description_overcast clouds',
    'weather_description_scattered clouds',
    'weather_description_shower rain']]

    X = features
    y = df.available_bike_stands

    train_features, test_features, train_labels, test_labels = train_test_split(X, y,  test_size=0.3, ra

    # Instantiate model with 10 decision trees
    rf = RandomForestRegressor(n_estimators = 10)
    # Train the model on training data
    rf.fit(train_features, train_labels);

    predictions = rf.predict(test_features)
    printMetrics(test_labels, predictions)
    print("\n\n")

    # Serialize model object into a file called model.pkl on disk using pickle
    file_name = "available_bike_stands_" + str(num) + ".pkl"
    with open(file_name, 'wb') as handle:
        pickle.dump(rf, handle, pickle.HIGHEST_PROTOCOL)
#       break
```

```
===============================================================================
MAE:  0.47655677655677653
RMSE:  0.9581680857689221
R2:  0.9657285706562504
```

Test available_bike_stands model

In [17]:

```
with open('available_bike_stands_2.pkl', 'rb') as handle:
    model = pickle.load(handle)
datapoint = pd.DataFrame({'day_no': [3], 'month': [4], 'hours':[16],'minutes':[30],
                         "time_of_day_Morning": [0], "time_of_day_Night": [0],
                         'day_Monday': [0], 'day_Saturday': [0], 'day_Sunday' : [0], 'day_Thursday':[0],
                         'day_Tuesday':[0], 'day_Wednesday':[0],
                         'temperature':[6],
'weather_description_drizzle':[0],
'weather_description_few clouds':[0],
'weather_description_light intensity drizzle':[0],
'weather_description_light intensity drizzle rain':[0],
'weather_description_light intensity shower rain':[0],
'weather_description_light rain':[0],
'weather_description_moderate rain':[0],
'weather_description_overcast clouds':[0],
'weather_description_scattered clouds':[0],
'weather_description_shower rain':[0]})

test_predictions = model.predict(datapoint)
test_predictions
```

Out[17]:

```
array([18.3])
```

In [ ]: