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 [19]:
```
df = pd.read_sql_query("SELECT * FROM weather_details order by date_created asc", conn)
df
```

Out[19]:

	sr_no	weather	weather_description	temperature	feels_like	temp_min	temp_max	pressure	humidity	visibility	speed	sunrise	suns

0	1	Clouds	scattered clouds	4	-1	3	4	1012	74	10000	4.6	2020-02-20 07:34:19	202 02- 17:43:
1	2	Clouds	few clouds	4	-2	3	4	1014	80	10000	5.7	2020-02-20 07:34:19	202 02- 17:43:
2	3	Clouds	scattered clouds	4	-2	3	5	1015	80	10000	6.2	2020-02-20 07:34:19	202 02- 17:43:
3	4	Clouds	broken clouds	6	0	6	7	1014	75	10000	6.7	2020-02-21 07:32:10	202 02- 17:45:
4	5	Clouds	broken clouds	8	0	7	8	1011	75	10000	8.7	2020-02-21 07:32:10	202 02- 17:45:
...	...	...	...	...	...	...	...	...	...	...	...	...	
182	183	Clouds	overcast clouds	7	2	6	7	1025	56	10000	4.1	2020-03-21 06:24:14	202 03- 18:39:
183	184	Clouds	overcast clouds	5	0	3	6	1026	65	10000	4.1	2020-03-22 06:21:47	202 03- 18:41:
184	185	Clouds	few clouds	1	-2	0	2	1027	91	10000	0.5	2020-03-22 06:21:47	202 03- 18:41:
185	186	Clouds	scattered clouds	8	2	7	8	1027	70	10000	6.2	2020-03-22 06:21:47	202 03- 18:41:
186	187	Clouds	few clouds	9	4	8	10	1026	63	10000	5.1	2020-03-22 06:21:47	202 03- 18:41:

187 rows × 14 columns

In [20]:
```
#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))
```

In [21]:
```
# 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)
```

In [22]:
```
df.dtypes
```

Out[22]:
```
sr_no                          int64
weather                       object
weather_description           object
temperature                   object
feels_like                    object
temp_min                      object
temp_max                      object
pressure                      object
humidity                      object
visibility                    object
speed                         object
sunrise                datetime64[ns]
sunset                 datetime64[ns]
date_created           datetime64[ns]
day_no                         int64
month                          int64
hours                          int64
minutes                        int64
time_of_day_Morning            uint8
time_of_day_Night              uint8
day_Monday                     uint8
day_Saturday                   uint8
day_Sunday                     uint8
day_Thursday                   uint8
day_Tuesday                    uint8
day_Wednesday                  uint8
dtype: object
```

In [23]:
```
df["temperature"]=df["temperature"].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_Wednesda
X = features
y = df.temperature
features
```

Out[23]:

	day_no	month	hours	minutes	time_of_day_Morning	time_of_day_Night	day_Monday	day_Saturday	day_Sunday	day_Thursday	day_T
0	20	2	19	13	0	0	0	0	0	1	
1	20	2	22	21	0	1	0	0	0	1	
2	20	2	23	13	0	1	0	0	0	1	
3	21	2	3	13	0	1	0	0	0	0	
4	21	2	7	13	1	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...
182	21	3	23	13	0	1	0	1	0	0	
183	22	3	3	13	0	1	0	0	1	0	
184	22	3	7	13	1	0	0	0	1	0	
185	22	3	11	13	1	0	0	0	1	0	
186	22	3	15	13	0	0	0	0	1	0	

187 rows × 12 columns

Train model for temperature

In [24]:
```
train_features, test_features, train_labels, test_labels = train_test_split(X, y,  test_size=0.3, random
```

In [25]:
```
# Instantiate model with 1000 decision trees
rf = RandomForestRegressor(n_estimators = 1000)
# Train the model on training data
rf.fit(train_features, train_labels);

# Printing Train features
predictions = rf.predict(train_features)
printMetrics(train_labels, predictions)

# Printing Test features
predictions = rf.predict(test_features)
printMetrics(test_labels, predictions)
```

```
================================================================================
MAE:  0.5529692307692308
RMSE:  0.7440918160880623
R2:  0.9431471621785836

================================================================================
MAE:  1.6542280701754386
RMSE:  2.14653448568748
R2:  0.48559803934437495
```

In [26]:
```
# Predict temperature
datapoint = pd.DataFrame({
    'day_no': [4],'month': [4],'hours': [16], 'minutes': [55] ,
    'time_of_day_Morning': [0], 'time_of_day_Night': [0],
    'day_Monday': [0], 'day_Saturday': [0], 'day_Sunday' : [1], 'day_Thursday':[0],
    'day_Tuesday':[0], 'day_Wednesday':[0]})
test_predictions = rf.predict(datapoint)
test_predictions
```

Out[26]:
```
array([6.644])
```

In [27]:
```
test_predictions = rf.predict([[4,4,16,55,0,0,0,0,1,0,0,0]])
test_predictions
```

Out[27]:
```
array([6.644])
```

In [28]:
```
# Serialize model object into a file called model.pkl on disk using pickle
with open("temperature.pkl", 'wb') as handle:
     pickle.dump(rf, handle, pickle.HIGHEST_PROTOCOL)
```

Train model for feels_like

In [29]:
```
df["feels_like"]=df["feels_like"].astype(int)
y = df.feels_like
train_features, test_features, train_labels, test_labels = train_test_split(X, y,  test_size=0.3, random
```

In [30]:
```
# Instantiate model with 1000 decision trees
rf = RandomForestRegressor(n_estimators = 1000)
# Train the model on training data
rf.fit(train_features, train_labels);

# Printing Train features
predictions = rf.predict(train_features)
printMetrics(train_labels, predictions)

# Printing Test features
predictions = rf.predict(test_features)
printMetrics(test_labels, predictions)
```

```
================================================================================
MAE:  0.6683
RMSE:  0.9091773669726851
R2:  0.9264122774921512

================================================================================
MAE:  1.9158596491228073
RMSE:  2.4802934178432383
R2:  0.43798846249015844
```

In [31]:
```
# Predict feels_like
datapoint = pd.DataFrame({
    'day_no': [3],'month': [4],'hours': [16], 'minutes': [55] ,
    '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]})
test_predictions = rf.predict(datapoint)
test_predictions
```

Out[31]:
```
array([-0.457])
```

In [32]:
```
# Serialize model object into a file called model.pkl on disk using pickle
with open("feels_like.pkl", 'wb') as handle:
     pickle.dump(rf, handle, pickle.HIGHEST_PROTOCOL)
```

Train model for temp_max

In [33]:
```
df["temp_max"]=df["temp_max"].astype(int)
y = df.temp_max
train_features, test_features, train_labels, test_labels = train_test_split(X, y,  test_size=0.3, random
```

In [34]:
```
# Instantiate model with 10000 decision trees
rf = RandomForestRegressor(n_estimators = 1000)
# Train the model on training data
rf.fit(train_features, train_labels);

# Printing Train features
predictions = rf.predict(train_features)
printMetrics(train_labels, predictions)

# Printing Test features
predictions = rf.predict(test_features)
printMetrics(test_labels, predictions)
```

```
================================================================================
MAE:  0.5522076923076923
RMSE:  0.7451918028782409
R2:  0.9411318974407226

================================================================================
MAE:  1.689877192982456
RMSE:  2.2164018058358192
R2:  0.46865611229109805
```

In [35]:
```
# Predict temp_max
datapoint = pd.DataFrame({
    'day_no': [3],'month': [4],'hours': [16], 'minutes': [55] ,
    '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]})
test_predictions = rf.predict(datapoint)
test_predictions
```

Out[35]:
```
array([7.656])
```

In [36]:
```
# Serialize model object into a file called model.pkl on disk using pickle
with open("temp_max.pkl", 'wb') as handle:
     pickle.dump(rf, handle, pickle.HIGHEST_PROTOCOL)
```

Train model for temp_min

In [37]:
```
df["temp_min"]=df["temp_min"].astype(int)
y = df.temp_min
train_features, test_features, train_labels, test_labels = train_test_split(X, y,  test_size=0.3, random
```

In [38]:
```
# Instantiate model with 1000 decision trees
rf = RandomForestRegressor(n_estimators = 1000)
# Train the model on training data
rf.fit(train_features, train_labels);

# Printing Train features
predictions = rf.predict(train_features)
printMetrics(train_labels, predictions)

# Printing Test features
predictions = rf.predict(test_features)
printMetrics(test_labels, predictions)
```

```
================================================================================
MAE:  0.5787
RMSE:  0.7814724710934627
R2:  0.941139942000057

================================================================================
MAE:  1.7853157894736844
RMSE:  2.2084640967574427
R2:  0.5209386267912208
```

In [39]:
```
# Predict temp_min
datapoint = pd.DataFrame({
    'day_no': [3],'month': [4],'hours': [16], 'minutes': [55] ,
    '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]})
test_predictions = rf.predict(datapoint)
test_predictions
```

Out[39]:
```
array([6.071])
```

In [40]:
```
# Serialize model object into a file called model.pkl on disk using pickle
with open("temp_min.pkl", 'wb') as handle:
     pickle.dump(rf, handle, pickle.HIGHEST_PROTOCOL)
```

Train model for weather description

In [41]:
```
# Import LabelEncoder
from sklearn import preprocessing
```

In [42]:
```
df = pd.read_sql_query("SELECT * FROM weather_details order by date_created asc", conn)
df
```

Out[42]:

	sr_no	weather	weather_description	temperature	feels_like	temp_min	temp_max	pressure	humidity	visibility	speed	sunrise	suns
0	1	Clouds	scattered clouds	4	-1	3	4	1012	74	10000	4.6	2020-02-20 07:34:19	202 02- 17:43:
1	2	Clouds	few clouds	4	-2	3	4	1014	80	10000	5.7	2020-02-20 07:34:19	202 02- 17:43:
2	3	Clouds	scattered clouds	4	-2	3	5	1015	80	10000	6.2	2020-02-20 07:34:19	202 02- 17:43:
3	4	Clouds	broken clouds	6	0	6	7	1014	75	10000	6.7	2020-02-21 07:32:10	202 02- 17:45:
4	5	Clouds	broken clouds	8	0	7	8	1011	75	10000	8.7	2020-02-21 07:32:10	202 02- 17:45:
...	...	...	...	...	...	...	...	...	...	...	...	...	

182	183	Clouds	overcast clouds	7	2	6	7	1025	56	10000	4.1	2020-03-21 06:24:14	202 03- 18:39:
183	184	Clouds	overcast clouds	5	0	3	6	1026	65	10000	4.1	2020-03-22 06:21:47	202 03- 18:41:
184	185	Clouds	few clouds	1	-2	0	2	1027	91	10000	0.5	2020-03-22 06:21:47	202 03- 18:41:
185	186	Clouds	scattered clouds	8	2	7	8	1027	70	10000	6.2	2020-03-22 06:21:47	202 03- 18:41:
186	187	Clouds	few clouds	9	4	8	10	1026	63	10000	5.1	2020-03-22 06:21:47	202 03- 18:41:

187 rows × 14 columns

In [43]:
```
# 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
df['day'] = df.date_created.dt.strftime("%A")

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
df
```

Out[43]:

	sr_no	weather	weather_description	temperature	feels_like	temp_min	temp_max	pressure	humidity	visibility	speed	sunrise	suns
0	1	Clouds	scattered clouds	4	-1	3	4	1012	74	10000	4.6	2020-02-20 07:34:19	202 02- 17:43:
1	2	Clouds	few clouds	4	-2	3	4	1014	80	10000	5.7	2020-02-20 07:34:19	202 02- 17:43:
2	3	Clouds	scattered clouds	4	-2	3	5	1015	80	10000	6.2	2020-02-20 07:34:19	202 02- 17:43:
3	4	Clouds	broken clouds	6	0	6	7	1014	75	10000	6.7	2020-02-21 07:32:10	202 02- 17:45:
4	5	Clouds	broken clouds	8	0	7	8	1011	75	10000	8.7	2020-02-21 07:32:10	202 02- 17:45:
...	...	...	...	...	...	...	...	...	...	...	...	...	
182	183	Clouds	overcast clouds	7	2	6	7	1025	56	10000	4.1	2020-03-21 06:24:14	202 03- 18:39:
183	184	Clouds	overcast clouds	5	0	3	6	1026	65	10000	4.1	2020-03-22 06:21:47	202 03- 18:41:
184	185	Clouds	few clouds	1	-2	0	2	1027	91	10000	0.5	2020-03-22 06:21:47	202 03- 18:41:
185	186	Clouds	scattered clouds	8	2	7	8	1027	70	10000	6.2	2020-03-22 06:21:47	202 03- 18:41:

186	187	Clouds	few clouds	9	4	8	10	1026	63	10000	5.1	2020-03-22 06:21:47	202 03- 18:41:

187 rows × 20 columns

In [44]:
```
le = preprocessing.LabelEncoder()

# Converting string labels into numbers.
day_encoded=le.fit_transform(df['day'])
df['day_encoded'] = day_encoded
df[['day', 'day_encoded']].head(50)
```

Out[44]:

	day	day_encoded
0	Thursday	4
1	Thursday	4
2	Thursday	4
3	Friday	0
4	Friday	0
5	Friday	0
6	Friday	0
7	Friday	0
8	Friday	0
9	Saturday	2
10	Saturday	2
11	Saturday	2
12	Saturday	2

13	Saturday	2
14	Saturday	2
15	Sunday	3
16	Sunday	3
17	Sunday	3
18	Sunday	3
19	Sunday	3
20	Sunday	3
21	Monday	1
22	Monday	1
23	Monday	1
24	Monday	1
25	Monday	1
26	Monday	1
27	Tuesday	5
28	Tuesday	5
29	Tuesday	5
30	Tuesday	5
31	Tuesday	5
32	Tuesday	5
33	Wednesday	6
34	Wednesday	6
35	Wednesday	6

36	Wednesday	6
37	Wednesday	6
38	Wednesday	6
39	Thursday	4
40	Thursday	4
41	Thursday	4
42	Thursday	4
43	Thursday	4
44	Thursday	4
45	Friday	0
46	Friday	0
47	Friday	0
48	Friday	0
49	Friday	0

In [45]:
```
time_of_day_encoded=le.fit_transform(df['time_of_day'])
df['time_of_day_encoded'] = time_of_day_encoded
df[['time_of_day', 'time_of_day_encoded']].head(50)
```

Out[45]:

	time_of_day	time_of_day_encoded
0	Afternoon	0
1	Night	2
2	Night	2
3	Night	2

4	Morning	1
5	Morning	1
6	Afternoon	0
7	Afternoon	0
8	Night	2
9	Night	2
10	Morning	1
11	Morning	1
12	Afternoon	0
13	Afternoon	0
14	Night	2
15	Night	2
16	Morning	1
17	Morning	1
18	Afternoon	0
19	Afternoon	0
20	Night	2
21	Night	2
22	Morning	1
23	Morning	1
24	Afternoon	0
25	Afternoon	0
26	Night	2

27	Night	2
28	Morning	1
29	Morning	1
30	Afternoon	0
31	Afternoon	0
32	Night	2
33	Night	2
34	Morning	1
35	Morning	1
36	Afternoon	0
37	Afternoon	0
38	Night	2
39	Night	2
40	Morning	1
41	Morning	1
42	Afternoon	0
43	Afternoon	0
44	Night	2
45	Night	2
46	Morning	1
47	Morning	1
48	Afternoon	0
49	Afternoon	0

In [46]:
```
weather_description_encoded=le.fit_transform(df['weather_description'])
df['weather_description_encoded'] = weather_description_encoded
```

In [47]:
```
df["weather_description"]=df["weather_description"].astype('category')
features = df[['day_no','month','hours','minutes','day_encoded', 'time_of_day_encoded']]
X = features
y = df.weather_description_encoded
features
```

Out[47]:

	day_no	month	hours	minutes	day_encoded	time_of_day_encoded
0	20	2	19	13	4	0
1	20	2	22	21	4	2
2	20	2	23	13	4	2
3	21	2	3	13	0	2
4	21	2	7	13	0	1
...	...	...	...	...	...	...
182	21	3	23	13	2	2
183	22	3	3	13	3	2
184	22	3	7	13	3	1
185	22	3	11	13	3	1
186	22	3	15	13	3	0

187 rows × 6 columns

Train using Naive Bayes model

In [48]:
```
train_features, test_features, train_labels, test_labels = train_test_split(X, y,  test_size=0.3, random
```

In [49]:
```
#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB

#Create a Gaussian Classifier
model = GaussianNB()

# Train the model using the training sets
model.fit(train_features,train_labels)

predictions = model.predict(train_features)
print("Accuracy:",metrics.accuracy_score(train_labels, predictions))
```

```
Accuracy: 0.3
```

In [50]:
```
predictions = model.predict(test_features)
predictions
```

Out[50]:
```
array([0, 6, 8, 0, 9, 9, 0, 9, 5, 0, 0, 9, 8, 9, 0, 9, 0, 0, 4, 9, 0, 4,
       6, 0, 8, 8, 0, 9, 8, 0, 0, 0, 5, 9, 6, 6, 8, 4, 5, 6, 0, 0, 8, 8,
       0, 0, 0, 0, 9, 9, 8, 0, 4, 4, 0, 8, 0])
```

In [51]:
```
print("Accuracy:",metrics.accuracy_score(test_labels, predictions))
```

```
Accuracy: 0.21052631578947367
```

As accuracy was not good, we tried using RandomForrestClassifier

In [52]:
```
#Import Random Forest Model
from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=1000)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(train_features,train_labels)

predictions = clf.predict(train_features)
print("Accuracy:",metrics.accuracy_score(train_labels, predictions))
```

```
Accuracy: 1.0
```

In [53]:
```
predictions = clf.predict(test_features)
print("Accuracy:",metrics.accuracy_score(test_labels, predictions))
```

```
Accuracy: 0.3508771929824561
```

In [54]:
```
predictions
```

Out[54]:
```
array([0, 0, 0, 9, 9, 0, 0, 9, 0, 0, 0, 9, 0, 0, 0, 9, 5, 0, 0, 9, 0, 2,
       6, 0, 9, 0, 0, 6, 0, 0, 0, 9, 5, 2, 0, 6, 0, 0, 2, 6, 0, 0, 8, 0,
       2, 0, 9, 0, 2, 9, 0, 5, 0, 9, 0, 0, 0])
```

In [55]:
```
# print(df[['weather_description', 'weather_description_encoded']])
d = dict()
for index, row in df.iterrows():
    d[row['weather_description_encoded']] = row['weather_description']
print(d)
```

```
{9: 'scattered clouds', 2: 'few clouds', 0: 'broken clouds', 6: 'light rain', 5: 'light intensity showe
r rain', 7: 'moderate rain', 8: 'overcast clouds', 10: 'shower rain', 3: 'light intensity drizzle', 1:
'drizzle', 4: 'light intensity drizzle rain'}
```

In [56]:
```
clf.predict([[3,4,18,16,1,2]])
```

Out[56]:
```
array([9])
```

In [57]:
```
# Serialize model object into a file called model.pkl on disk using pickle
with open("weather_description.pkl", 'wb') as handle:
     pickle.dump(clf, handle, pickle.HIGHEST_PROTOCOL)
```

In [ ]:
```

```