Prediction of weather parameters using Random Forest ML model

Data Preparation

In [1]:
```python
# Import pandas, numpy, matplotlib, seaborn libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime
import csv
# Using sklearn to split data into training and testing sets,train classifier and regressor models
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
#pickle package saves and loads sklearn models
import pickle

# hide ipykernel warnings
import warnings
warnings.filterwarnings('ignore')

%matplotlib inline
```

Import data

Import weather dynamic data from csv into dataframe

In [2]:
```python
# Read in data as pandas dataframe and display first 5 rows
weatherDynamic = pd.read_csv('dWeatherD.csv')
weatherDynamic.reset_index(drop=True)
weatherDynamic = weatherDynamic[['main_temp','main_feels_like',\
                              'main_pressure','main_humidity','main_temp_min', 'main_temp_max', \
                               'wind_speed', 'wind_deg','data_entry_timestamp', 'weather_main']]
weatherDynamic.reset_index(drop=True)

weatherDynamic.tail(5)
```

Out[2]:

	main_temp	main_feels_like	main_pressure	main_humidity	main_temp_min	main_temp_max	wind_speed	wind_deg	data_entry_timestamp	weather_mai
3437	10	7	1018	66	8	12	2.6	150.0	8/24/2020 10:40	Clouc
3438	11	9	1017	58	9	12	1.0	NaN	9/24/2020 10:50	Clouc
3439	11	9	1017	58	9	12	1.0	NaN	10/24/2020 0:00	Clouc
3440	11	9	1017	58	9	12	1.0	NaN	11/24/2020 11:10	Clouc
3441	11	7	1017	62	10	12	4.1	200.0	12/24/2020 11:20	Clouc

check for shape

In [3]:
```python
print('The shape of our features is:', weatherDynamic.shape,'\n', weatherDynamic.dtypes)
```

```
The shape of our features is: (3442, 10)
 main_temp                int64
main_feels_like          int64
main_pressure            int64
main_humidity            int64
main_temp_min            int64
main_temp_max            int64
wind_speed             float64
wind_deg               float64
data_entry_timestamp    object
weather_main            object
dtype: object
```

check for Null entries

In [4]:
```python
weatherDynamic.isnull().sum()
```

Out[4]:
```
main_temp                 0
main_feels_like           0
main_pressure             0
main_humidity             0
main_temp_min             0
main_temp_max             0
wind_speed                0
wind_deg                112
data_entry_timestamp      0
weather_main              0
dtype: int64
```

Fill Null entries

In [5]:
```python
weatherDynamic.loc[weatherDynamic['wind_deg'].isnull(),('wind_deg')] = weatherDynamic['wind_deg'].ffill()
```

In [6]:
```python
weatherDynamic.isnull().sum()
```

Out[6]:
```
main_temp               0
main_feels_like         0
main_pressure           0
main_humidity           0
main_temp_min           0
main_temp_max           0
wind_speed              0
wind_deg                0
data_entry_timestamp    0
weather_main            0
dtype: int64
```

Process and normalise dates

Observation state that results for date 28 February and 24 March are patial. Hence, they can be dropped for data consistency. Its upto discration of user whether to incorporate it for training or not.

dublin weather

In [7]:
```python
#DATETIME DATA

# Select columns containing datetime data
continous_date_columns = weatherDynamic[['data_entry_timestamp']].columns

# Assign object type datetime to columns enlisted in continous_date_columns
for column in continous_date_columns:
    weatherDynamic[column] = pd.to_datetime(weatherDynamic[column])

# end_date = pd.to_datetime('2020-03-24')
# start_date = pd.to_datetime('2020-02-28')

# # Ommit partial data for dates 28-2-2020 and 24-3-2020
# mask = (weatherDynamic['data_entry_timestamp'].dt.date > start_date) & (weatherDynamic['data_entry_timestamp'].dt.
date < end_date)
# weatherDynamic = weatherDynamic.loc[mask]

# for column in continous_date_columns:
#     weatherDynamic[column] = pd.to_datetime(weatherDynamic[column])

weatherDynamic.tail()
```

Out[7]:

	main_temp	main_feels_like	main_pressure	main_humidity	main_temp_min	main_temp_max	wind_speed	wind_deg	data_entry_timestamp	weather_mai
3437	10	7	1018	66	8	12	2.6	150.0	2020-08-24 10:40:00	Clouc
3438	11	9	1017	58	9	12	1.0	150.0	2020-09-24 10:50:00	Clouc
3439	11	9	1017	58	9	12	1.0	150.0	2020-10-24 00:00:00	Clouc
3440	11	9	1017	58	9	12	1.0	150.0	2020-11-24 11:10:00	Clouc
3441	11	7	1017	62	10	12	4.1	200.0	2020-12-24 11:20:00	Clouc

In [8]:
```python
print('The shape of our features is:', weatherDynamic.shape,'\n', weatherDynamic.dtypes)
```

```
The shape of our features is: (3442, 10)
 main_temp                        int64
main_feels_like                  int64
main_pressure                    int64
main_humidity                    int64
main_temp_min                    int64
main_temp_max                    int64
wind_speed                     float64
wind_deg                       float64
data_entry_timestamp    datetime64[ns]
weather_main                    object
dtype: object
```

In [9]:
```python
weatherDynamic.head()
```

Out[9]:

	main_temp	main_feels_like	main_pressure	main_humidity	main_temp_min	main_temp_max	wind_speed	wind_deg	data_entry_timestamp	weather_main
0	12	7	987	87	11	14	7.2	250.0	2020-02-28 15:00:00	Clouds
1	12	7	987	87	11	14	7.2	250.0	2020-02-28 15:10:00	Clouds
2	11	6	987	87	11	12	7.2	250.0	2020-02-28 15:20:00	Clouds
3	12	5	987	71	11	12	8.2	250.0	2020-02-28 15:30:00	Clouds
4	12	5	987	71	11	12	8.2	250.0	2020-02-28 15:40:00	Clouds

In [10]:
```python
print('The shape of our features is:', weatherDynamic.shape,'\n', weatherDynamic.dtypes)
```

```
The shape of our features is: (3442, 10)
 main_temp                        int64
main_feels_like                  int64
main_pressure                    int64
main_humidity                    int64
main_temp_min                    int64
main_temp_max                    int64
wind_speed                     float64
wind_deg                       float64
data_entry_timestamp    datetime64[ns]
weather_main                    object
dtype: object
```

Resolve date into time components

In [11]:
```python
#Drop 'last_update' feature
weatherDynamic['year'] = weatherDynamic.data_entry_timestamp.dt.year
weatherDynamic['Day'] = weatherDynamic.data_entry_timestamp.dt.day
weatherDynamic['month'] = weatherDynamic.data_entry_timestamp.dt.month
weatherDynamic['hour'] = weatherDynamic.data_entry_timestamp.dt.hour
weatherDynamic['minute'] = weatherDynamic.data_entry_timestamp.dt.minute
weatherDynamic = weatherDynamic.drop(['data_entry_timestamp'], axis=1)
```

Set Data types prior to encoding

In [12]:
```python
weatherDynamic.reset_index(drop=True)
```

Out[12]:

	main_temp	main_feels_like	main_pressure	main_humidity	main_temp_min	main_temp_max	wind_speed	wind_deg	weather_main	year	Day	month
0	12	7	987	87	11	14	7.2	250.0	Clouds	2020	28	2
1	12	7	987	87	11	14	7.2	250.0	Clouds	2020	28	2
2	11	6	987	87	11	12	7.2	250.0	Clouds	2020	28	2
3	12	5	987	71	11	12	8.2	250.0	Clouds	2020	28	2
4	12	5	987	71	11	12	8.2	250.0	Clouds	2020	28	2
...	...	...	...	...	...	...	...	...	...	...	...	...
3437	10	7	1018	66	8	12	2.6	150.0	Clouds	2020	24	8
3438	11	9	1017	58	9	12	1.0	150.0	Clouds	2020	24	9
3439	11	9	1017	58	9	12	1.0	150.0	Clouds	2020	24	10
3440	11	9	1017	58	9	12	1.0	150.0	Clouds	2020	24	11
3441	11	7	1017	62	10	12	4.1	200.0	Clouds	2020	24	12

3442 rows × 14 columns

In [13]:
```python
print('The shape of our features is:', weatherDynamic.shape)
print(weatherDynamic.dtypes)
```

```
The shape of our features is: (3442, 14)
main_temp            int64
main_feels_like      int64
main_pressure        int64
main_humidity        int64
main_temp_min        int64
main_temp_max        int64
wind_speed         float64
wind_deg           float64
weather_main        object
year                 int64
Day                  int64
month                int64
hour                 int64
minute               int64
dtype: object
```

In [14]:
```python
# Assign data type category to columns listed in categorical_columns
categorical_columns = weatherDynamic[['year','month', 'weather_main']].columns
for column in categorical_columns:
    weatherDynamic[column] = weatherDynamic[column].astype('category')
```

In [15]:
```python
print('The shape of our features is:', weatherDynamic.shape)
print(weatherDynamic.dtypes)
```

```
The shape of our features is: (3442, 14)
main_temp             int64
main_feels_like       int64
main_pressure         int64
main_humidity         int64
main_temp_min         int64
main_temp_max         int64
wind_speed          float64
wind_deg            float64
weather_main       category
year               category
Day                   int64
month              category
hour                  int64
minute                int64
dtype: object
```

In [16]:
```python
weatherDynamic.head()
```

Out[16]:

	main_temp	main_feels_like	main_pressure	main_humidity	main_temp_min	main_temp_max	wind_speed	wind_deg	weather_main	year	Day	month	ho
0	12	7	987	87	11	14	7.2	250.0	Clouds	2020	28	2	
1	12	7	987	87	11	14	7.2	250.0	Clouds	2020	28	2	
2	11	6	987	87	11	12	7.2	250.0	Clouds	2020	28	2	
3	12	5	987	71	11	12	8.2	250.0	Clouds	2020	28	2	
4	12	5	987	71	11	12	8.2	250.0	Clouds	2020	28	2	

Encoding

Dictionary vectorization

In [17]:
```python
def encode(weatherDynamic, col):
    dfMLweather_enc = pd.get_dummies(weatherDynamic, columns = col, prefix_sep='_')
    dfMLweather_enc.head()
    dfMLweather_enc.to_csv("WeatherGetDummiesCatagorical.csv")
    return dfMLweather_enc
```

Training and Testing Sets

In [18]:
```python
def trainTestData(features, target):
    # Split the data into training and testing sets
    train_features, test_features, train_labels, test_labels = train_test_split(features, target, test_size = 0.25,
                                                                                random_state = 42)
    return train_features, test_features, train_labels, test_labels
```

Training the Forest

In [19]:
```python
def trainRandomForest(train_features, train_labels, classify = False):
    # Instantiate model
    rf = RandomForestRegressor(n_estimators= 100, random_state=42) if not classify else\
        RandomForestClassifier(n_estimators= 100, random_state=42)

    # Train the random forest model
    rf.fit(train_features, train_labels)

    return rf
```

Make Predictions on Test Data

In [20]:
```python
def predictRandomForest(rf, test_features):
    # Use the forest's predict method on the test data
    predictions = rf.predict(test_features)

    return predictions
```

View Results

In [21]:
```python
def result(predictions,test_labels, targetC, classify = False):

    # Save results to CSV

    if not classify:
        with open('resultWeather.csv','a') as resultCSV:
            resultCSV.write(targetC+",")
            resultCSV.write(str(metrics.mean_absolute_error(test_labels, predictions))+",")
            resultCSV.write(str(metrics.mean_squared_error(test_labels, predictions))+",")
            resultCSV.write(str(metrics.mean_squared_error(test_labels, predictions, squared=False))+",")
            resultCSV.write(str(metrics.r2_score(test_labels, predictions))+"\n")
            resultCSV.close()
        print('\n==========================================================================')
        print(targetC)
        print("MAE: ",metrics.mean_absolute_error(test_labels, predictions))
        print("MSE: ",metrics.mean_squared_error(test_labels, predictions))
        print("RMSE: ",metrics.mean_squared_error(test_labels, predictions, squared=False))
        print("R2: ",metrics.r2_score(test_labels, predictions),"\n")
        resultCSV.close()

        plt.plot(test_labels[:10])
        plt.plot(predictions[:10])

        plt.xlabel('Time')
        plt.ylabel('Value')

        plt.show()

    else:
        with open('resultWeather.csv','a') as resultCSV:
            resultCSV.write("\n"+targetC+"\n")
            resultCSV.write(metrics.classification_report(test_labels, predictions)+"\n\n")
            resultCSV.close()

        print("report: \n", metrics.classification_report(test_labels, predictions))
```

Variable Importances

In [22]:
```python
def show_Importance(rf,features,feature_list):
    # Get numerical feature importances
    importances = list(rf.feature_importances_)

    # List of tuples with variable and importance
    feature_importances = [(features, round(importance, 2)) for features, importance in zip(feature_list, importance
s)]

    # Sort the feature importances by most important first
    feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)

    # Print out the feature and importances
    print('Variable IMportance')
    [print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_importances]
```

Pipeline

In [23]:
```python
# ****************************************************************************************************************
def pipeLine(dfMLweather_enc, targetCols, classify = False):
    dfML_inputPipe = dfMLweather_enc.drop(targetCols, axis = 1)

    # Saving input feature names for later use
    feature_list = list(dfML_inputPipe.columns)
    # Convert to numpy array
    features = np.array(dfML_inputPipe)

    # ***************************************************************************************************************
*
    # dfML_PipeLine = normaliseDF(dfMLweather_enc[targetCols])
    dfML_PipeLine = dfMLweather_enc[targetCols]

    # ***************************************************************************************************************
*
    if not classify:
        with open('resultWeather.csv','a') as resultCSV:
                resultCSV.write("Feature,MAE, MSE,RMSE,R2\n")

    for targetC in targetCols:

        print("Target Feature:", targetC, "\n\n")

        # Array of target variable
        target = np.array(dfML_PipeLine[targetC])

        train_features, test_features, train_labels, test_labels = trainTestData(features, target)
        rfWeatherModel = trainRandomForest(train_features, train_labels, classify)

        # save the model
        filename = targetC+'.pkl'
        pickle.dump(rfWeatherModel, open(filename, 'wb'))

        # load the model
        rfWeatherModel_loaded = pickle.load(open(filename, 'rb'))

        predictions = predictRandomForest(rfWeatherModel_loaded,test_features)
        result(predictions,test_labels,targetC,classify)
        # show_Importance(rfWeatherModel_loaded,features, feature_list)
        print("\n\n\n\n\n\n\n")
```

Predictions

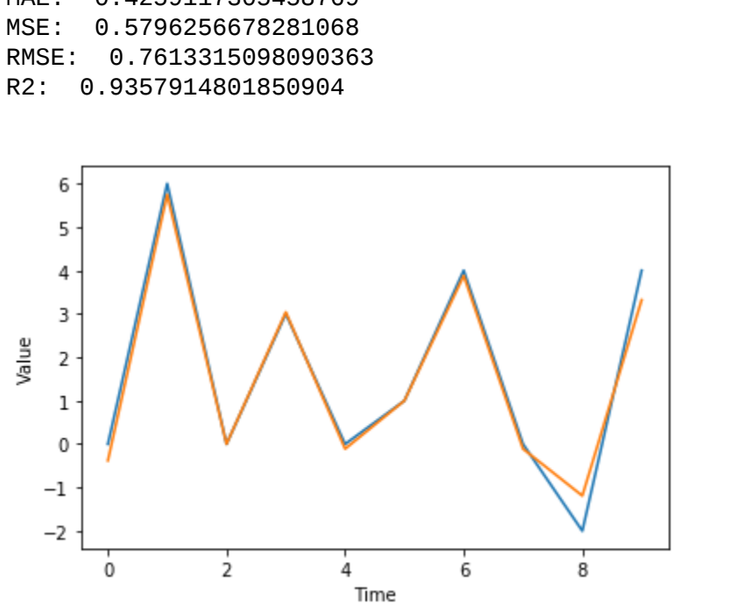
- **As it is evident from Variable Importance, catagorical features "month: are not significant NOW; given the fact that data of only 2 years is provided.**
- **Yet they are preserved because model must be trained to recognise the catagory when data of many years acumulate in future and used for prediction.**

In [24]:
```python
# COntinuous Target columns to be predicted.
# ****************************************************************************************************************
targetCols = ['main_temp','main_feels_like', 'main_pressure', 'main_humidity', \
             'main_temp_min', 'main_temp_max', 'wind_speed', 'wind_deg']
targetCatCols = ['weather_main']
encodeCols = ['year','month']

dfMLweather_enc = encode(weatherDynamic.drop(targetCatCols,axis=1),encodeCols)
pipeLine(dfMLweather_enc, targetCols)
```

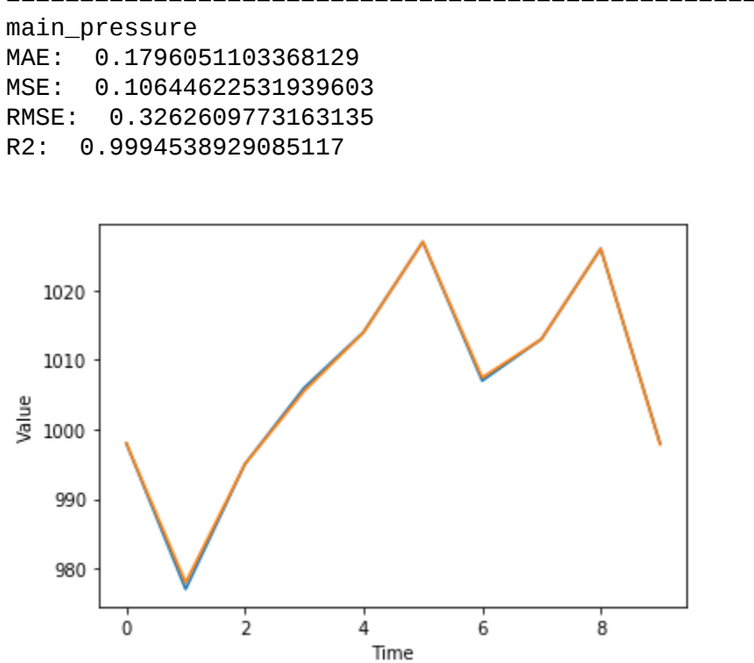
```
Target Feature: main_temp


==========================================================================
main_temp
MAE:  0.25653890824622533
MSE:  0.17324018583042974
RMSE:  0.4162213183276774
R2:  0.9827697561003097
```



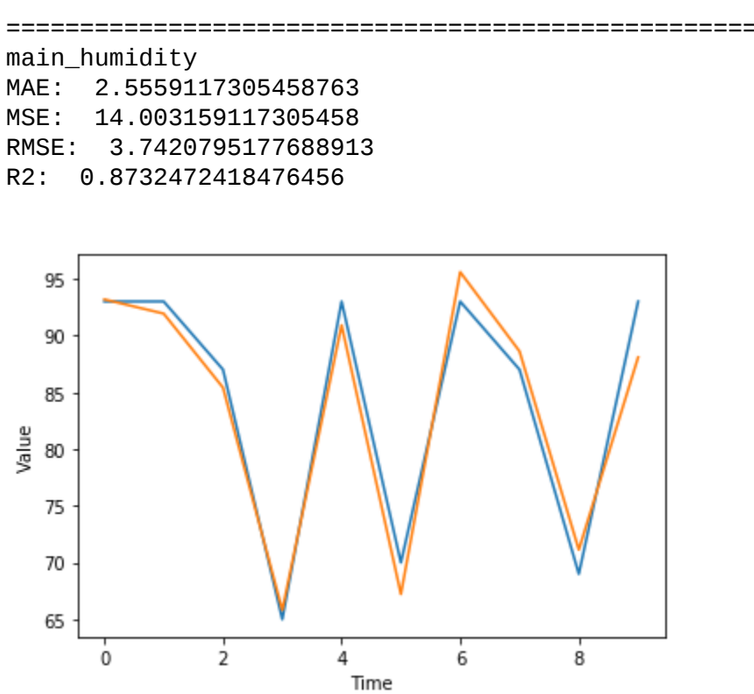

```
Target Feature: main_feels_like


==========================================================================
main_feels_like
MAE:  0.4259117305458769
MSE:  0.5796256678281068
RMSE:  0.7613315098090363
R2:  0.9357914801850904
```



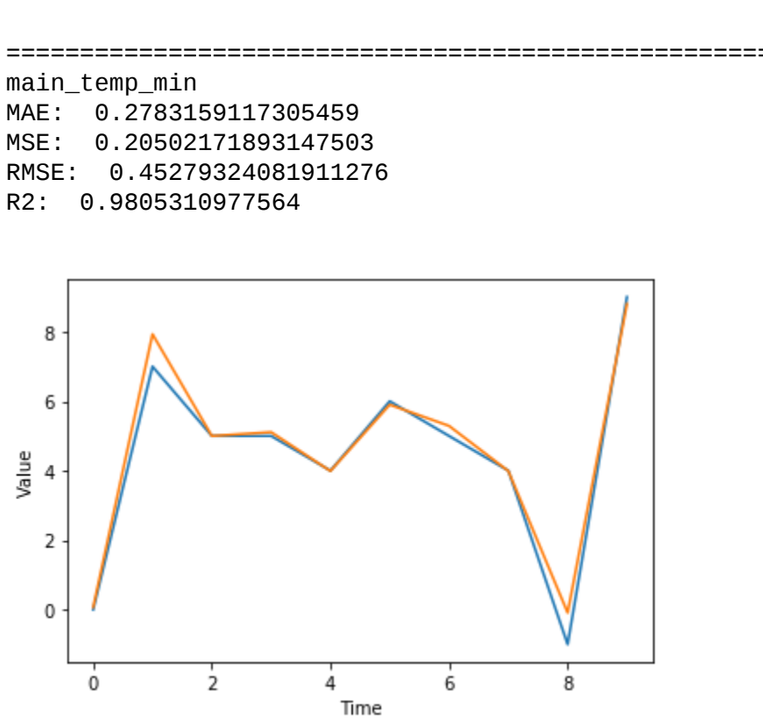

```
Target Feature: main_pressure


==========================================================================
main_pressure
MAE:  0.1796051103368129
MSE:  0.10644622531939603
RMSE:  0.3262609773163135
R2:  0.9994538929085117
```



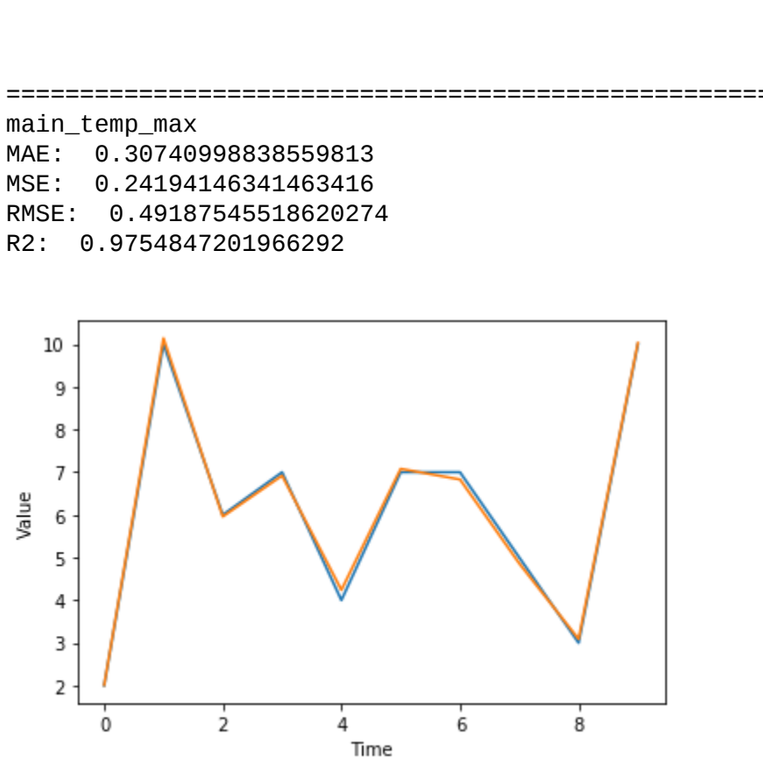

```
Target Feature: main_humidity


==========================================================================
main_humidity
MAE:  2.5559117305458763
MSE:  14.003159117305458
RMSE:  3.7420795177688913
R2:  0.8732472418476456
```



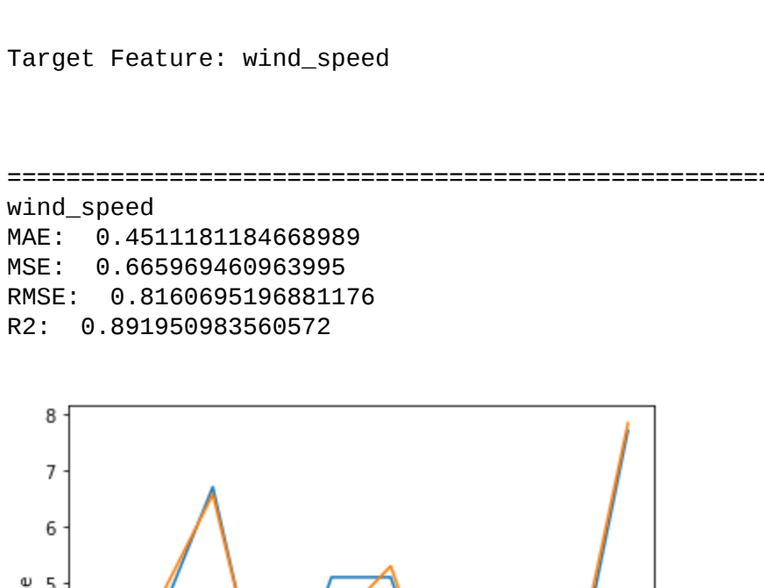

```
Target Feature: main_temp_min


==========================================================================
main_temp_min
MAE:  0.2783159117305459
MSE:  0.20502171893147503
RMSE:  0.45279324081911276
R2:  0.9805310977564
```



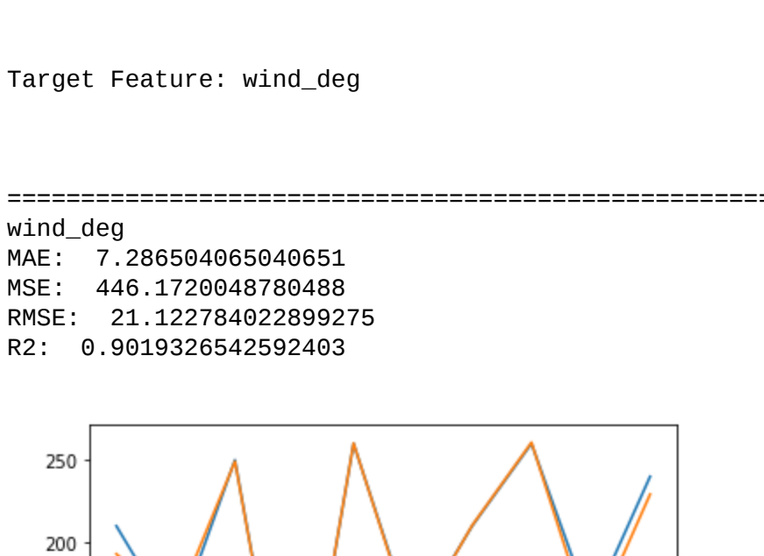

```
Target Feature: main_temp_max


==========================================================================
main_temp_max
MAE:  0.30740998838559813
MSE:  0.24194146341463416
RMSE:  0.49187545518620274
R2:  0.9754847201966292
```



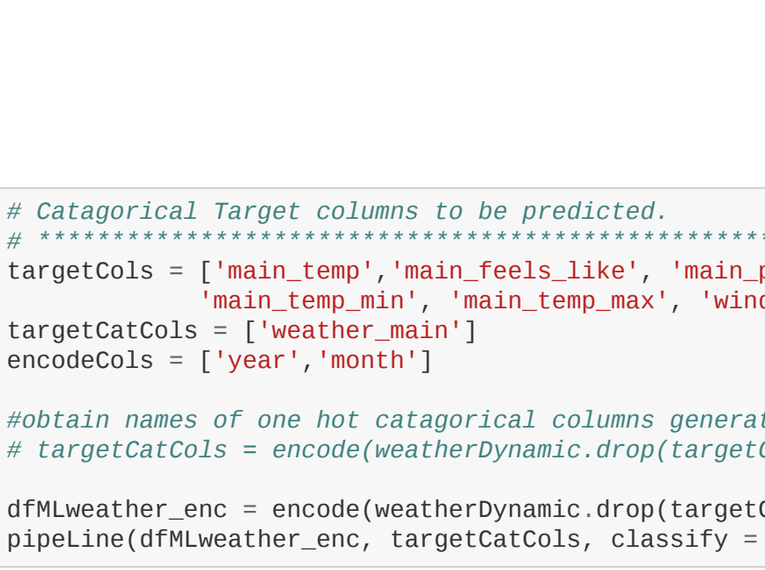

```
Target Feature: wind_speed


==========================================================================
wind_speed
MAE:  0.4511181184668989
MSE:  0.665969460963995
RMSE:  0.8160695196881176
R2:  0.891950983560572
```




```
Target Feature: wind_deg


==========================================================================
wind_deg
MAE:  7.286504065040651
MSE:  446.1720048780488
RMSE:  21.122784022899275
R2:  0.9019326542592403
```



In [25]:
```python
# Catagorical Target columns to be predicted.
# ****************************************************************************************************************
targetCols = ['main_temp','main_feels_like', 'main_pressure', 'main_humidity', \
             'main_temp_min', 'main_temp_max', 'wind_speed', 'wind_deg']
targetCatCols = ['weather_main']
encodeCols = ['year','month']

#obtain names of one hot catagorical columns generated after encoding
# targetCatCols = encode(weatherDynamic.drop(targetCols,axis=1)).columns

dfMLweather_enc = encode(weatherDynamic.drop(targetCols,axis=1),encodeCols)
pipeLine(dfMLweather_enc, targetCatCols, classify = True)
```

```
Target Feature: weather_main


report:
               precision    recall  f1-score   support

       Clear       0.00      0.00      0.00         2
      Clouds       0.90      0.96      0.93       687
     Drizzle       0.67      0.33      0.44        12
        Mist       0.00      0.00      0.00         1
        Rain       0.75      0.57      0.65       159
        Snow       0.00      0.00      0.00         0

    accuracy                           0.88       861
   macro avg       0.39      0.31      0.34       861
weighted avg       0.87      0.88      0.87       861
```