# JawadAhmed_20P-0165_AI_Project

April 30, 2023

## 1 Weather Forecasting Using ARIMA, Exponential Smoothing and Support Vector Regressiong Model

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
from scipy.stats import zscore
from sklearn.preprocessing import LabelEncoder
from statsmodels.tsa.arima.model import ARIMA
import pmdarima as pm
import seaborn as sns
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score, accuracy
from statsmodels.tsa.seasonal import seasonal_decompose

from sklearn.model_selection import train_test_split
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

In [2]:
```
istanbul_dataset = pd.read_csv('Istanbul Weather Data.csv')
```

In [3]:
```
istanbul_dataset.shape
```

Out[3]:
```
(3896, 12)
```

In [4]:
```
istanbul_dataset.head()
```

Out[4]:

| | DateTime | Condition | Rain | MaxTemp | MinTemp | SunRise | SunSet | MoonRise | MoonSet | AvgWind | AvgHumidity | AvgPressure |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 02.09.2019 | Partly cloudy | 0.0 | 27 | 22 | 06:32:00 | 19:37:00 | 9:52:00 | 21:45:00 | 23 | 66 | 1012 |
| 1 | 01.09.2019 | Partly cloudy | 0.0 | 27 | 22 | 06:31:00 | 19:38:00 | 8:37:00 | 21:13:00 | 21 | 66 | 1011 |
| 2 | 31.08.2019 | Patchy rain possible | 0.5 | 26 | 22 | 06:30:00 | | | | | | |
| 3 | 30.08.2019 | Partly cloudy | 0.0 | 27 | 22 | 06:29:00 | | | | | | |
| 4 | 29.08.2019 | Partly cloudy | 0.0 | 27 | 23 | 06:27:00 | | | | | | |

```
2  19:40:00  7:21:00  20:40:00       22           63         1015
3  19:42:00   6:4:00   20:5:00       20           64         1016
4  19:43:00  4:47:00  19:26:00       24           61         1015
```

```
In [5]: istanbul_dataset.tail()
```

```
Out[5]:         DateTime      Condition  Rain  MaxTemp  MinTemp   SunRise    SunSet  \
3891  05.01.2009       Overcast  4.32        5        3  08:29:00  17:50:00
3892  04.01.2009           Mist  2.91        5        3  08:29:00  17:49:00
3893  03.01.2009       Overcast  0.08        5        3  08:29:00  17:48:00
3894  02.01.2009       Overcast  4.48        4        1  08:29:00  17:48:00
3895  01.01.2009  Partly cloudy  0.23        5        2  08:29:00  17:47:00

      MoonRise   MoonSet  AvgWind  AvgHumidity  AvgPressure
3891   0:41:00   1:57:00       15           97         1015
3892   0:15:00  12:48:00        9           94         1014
3893  11:52:00       NaN       16           94         1021
3894  11:30:00  23:43:00       12           89         1021
3895   11:9:00  22:39:00       10           90         1027
```

```
In [6]: istanbul_dataset.describe()
```

```
Out[6]:               Rain      MaxTemp      MinTemp      AvgWind  AvgHumidity  \
count  3896.000000  3896.000000  3896.000000  3896.000000  3896.000000
mean      0.946794    18.084189    13.774897    16.989220    71.414784
std       2.558308     7.613318     6.865021     7.950417     9.483500
min       0.000000    -3.000000    -5.000000     2.000000    40.000000
25%       0.000000    12.000000     8.000000    11.000000    65.000000
50%       0.010000    18.000000    14.000000    16.000000    71.000000
75%       0.720000    25.000000    20.000000    22.000000    78.000000
max      42.000000    37.000000    26.000000    56.000000    97.000000

       AvgPressure
count  3896.000000
mean   1015.281314
std       6.284232
min     992.000000
25%    1011.000000
50%    1015.000000
75%    1019.000000
max    1038.000000
```

```
In [7]: istanbul_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3896 entries, 0 to 3895
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
```

```
 0   DateTime     3896 non-null   object
 1   Condition    3896 non-null   object
 2   Rain         3896 non-null   float64
 3   MaxTemp      3896 non-null   int64
 4   MinTemp      3896 non-null   int64
 5   SunRise      3896 non-null   object
 6   SunSet       3896 non-null   object
 7   MoonRise     3764 non-null   object
 8   MoonSet      3765 non-null   object
 9   AvgWind      3896 non-null   int64
 10  AvgHumidity  3896 non-null   int64
 11  AvgPressure  3896 non-null   int64
dtypes: float64(1), int64(5), object(6)
memory usage: 365.4+ KB
```

```
In [8]: # Convert the DateTime column to a datetime object
        istanbul_dataset['DateTime'] = pd.to_datetime(istanbul_dataset['DateTime'], format='%d

        # Set the DateTime column as the index
        istanbul_dataset.set_index('DateTime', inplace=True)

        # Drop unnecessary columns
        istanbul_dataset.drop(['SunRise', 'SunSet', 'MoonRise', 'MoonSet'], axis=1, inplace=Tru
```

```
In [9]: istanbul_dataset.head()
```

Out[9]:

| DateTime | Condition | Rain | MaxTemp | MinTemp | AvgWind | AvgHumidity | AvgPressure |
|---|---|---|---|---|---|---|---|
| 2019-09-02 | Partly cloudy | 0.0 | 27 | 22 | 23 | 66 | 1012 |
| 2019-09-01 | Partly cloudy | 0.0 | 27 | 22 | 21 | 66 | 1011 |
| 2019-08-31 | Patchy rain possible | 0.5 | 26 | 22 | 22 | 63 | 1015 |
| 2019-08-30 | Partly cloudy | 0.0 | 27 | 22 | 20 | 64 | 1016 |
| 2019-08-29 | Partly cloudy | 0.0 | 27 | 23 | 24 | 61 | 1015 |

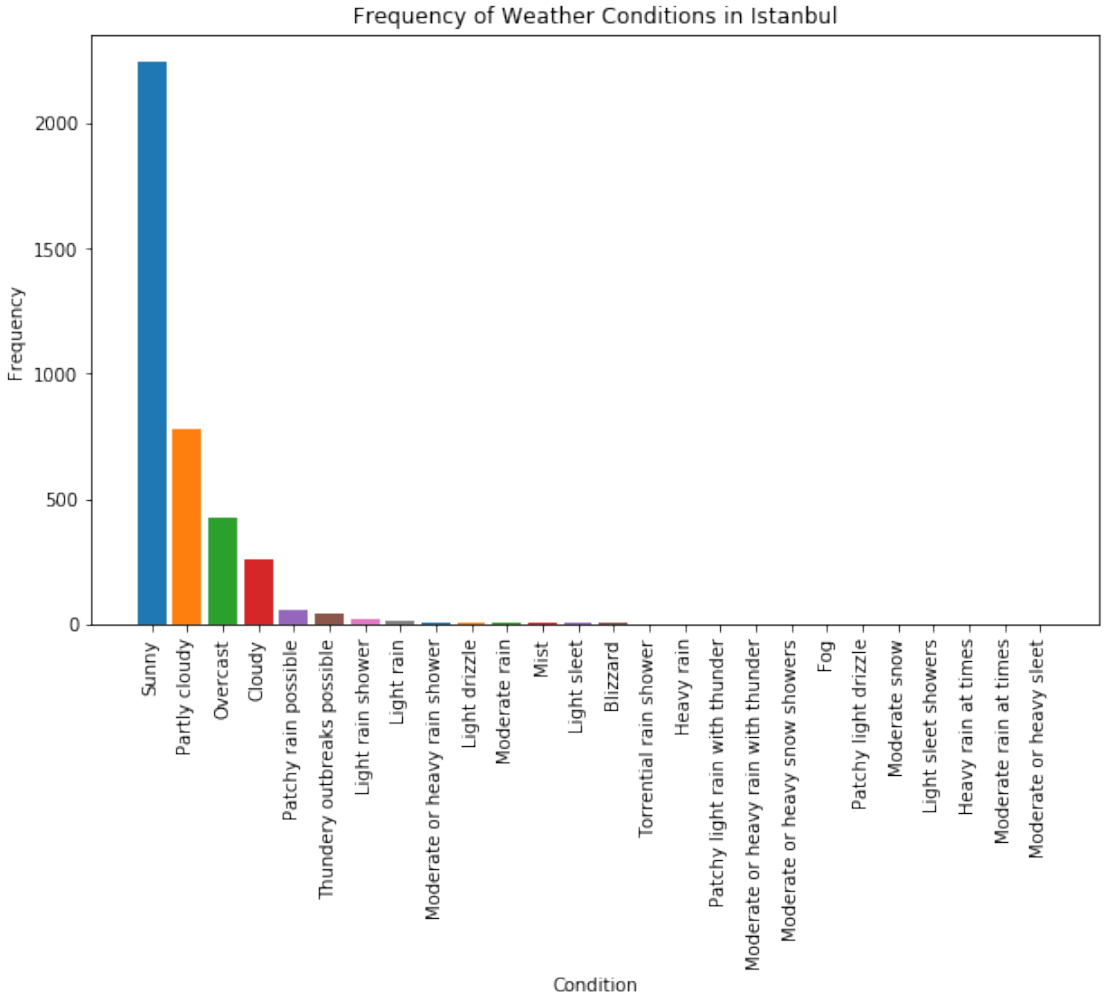
```
In [10]: missing_values_count = istanbul_dataset.sum()
```

```
In [11]: missing_values_count
```

```
Out[11]: Condition      Partly cloudyPartly cloudyPatchy rain possible...
         Rain                                                    3688.71
```

```
MaxTemp                                                 70456
MinTemp                                                 53667
AvgWind                                                 66190
AvgHumidity                                            278232
AvgPressure                                           3955536
dtype: object
```

In [12]:
```
# Set the figure size
plt.figure(figsize=(10, 6))

# Plot the frequency of each condition value
condition_counts = istanbul_dataset['Condition'].value_counts()
colors = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brown'
plt.bar(condition_counts.index, condition_counts.values, color=colors)

# Set the x-tick labels to rotate vertically
plt.xticks(rotation=90)

# Set the title and axis labels
plt.title('Frequency of Weather Conditions in Istanbul')
plt.xlabel('Condition')
plt.ylabel('Frequency')

# Show the plot
plt.show()
```

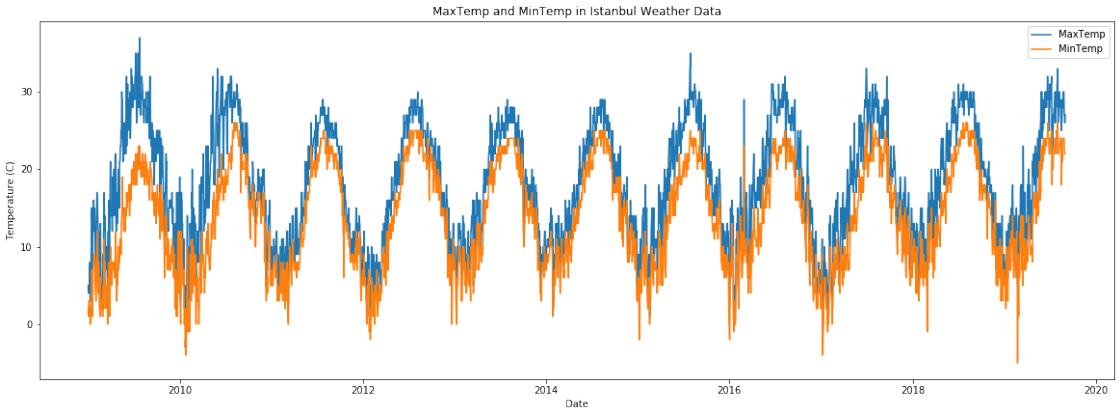



### 1.0.1 Visualizing the Max and Min Temperature

```
In [13]: # Set the figure size
         plt.figure(figsize=(20, 15))

         # Create a subplot for MaxTemp and MinTemp
         plt.subplot(2, 1, 1)

         # Plot MaxTemp over time
         plt.plot(istanbul_dataset.index, istanbul_dataset['MaxTemp'], label='MaxTemp')

         # Plot MinTemp over time
         plt.plot(istanbul_dataset.index, istanbul_dataset['MinTemp'], label='MinTemp')

         # Set the title and axis labels
         plt.title('MaxTemp and MinTemp in Istanbul Weather Data')
```

```
        plt.xlabel('Date')
        plt.ylabel('Temperature (C)')

        # Add a legend
        plt.legend()

        # Save the graph as a PNG file
        plt.savefig('istanbul_temps.png')
```

```
/home/jawad_ahmed/anaconda3/lib/python3.7/site-packages/matplotlib/cbook/__init__.py:2064: Futu
  x[:, None]
/home/jawad_ahmed/anaconda3/lib/python3.7/site-packages/matplotlib/axes/_base.py:248: FutureWa
  x = x[:, np.newaxis]
/home/jawad_ahmed/anaconda3/lib/python3.7/site-packages/matplotlib/axes/_base.py:250: FutureWa
  y = y[:, np.newaxis]
```




### 1.0.2 Inference:

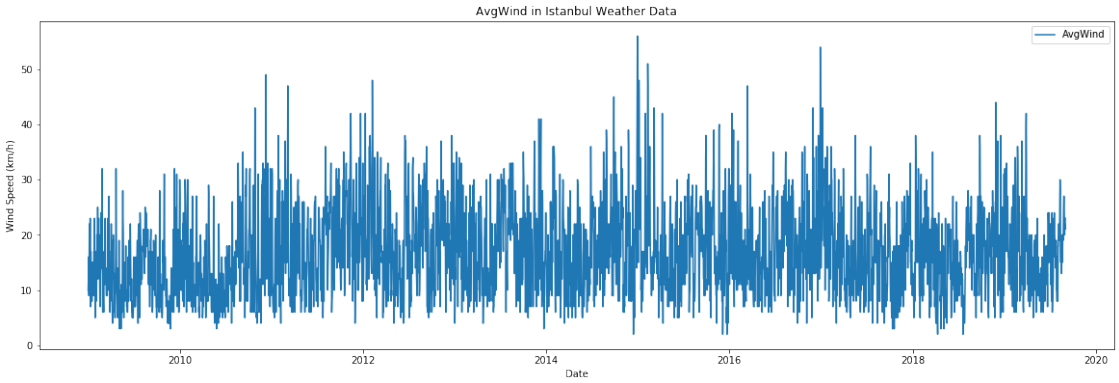
1. The graph shows that max and min temperature at the start and mid of the year the temperature is low.
2. The temperature is high usually in the time in between start and mid and end of the year.
3. The Max and Min Temperature does not seem to be stationary.
4. There is also some seasonality in the graph. Pattern are repeating.
5. The temperatures tend to be highest in the summer months (June to September) and lowest in the winter months (December to February). There are also some fluctuations within each season, but the overall pattern is quite clear.

### 1.0.3 Visualizing the Wind Speed

In [14]:
```
# Set the figure size
plt.figure(figsize=(20, 15))
# Create a subplot for AvgWind
```

```
plt.subplot(2, 1, 2)

# Plot AvgWind over time
plt.plot(istanbul_dataset.index, istanbul_dataset['AvgWind'], label='AvgWind')

# Set the title and axis labels
plt.title('AvgWind in Istanbul Weather Data')
plt.xlabel('Date')
plt.ylabel('Wind Speed (km/h)')

# Add a legend
plt.legend()

# Adjust the spacing between subplots
plt.subplots_adjust(hspace=0.4)

# Save the figure as a PNG image
plt.savefig('istanbul_weather_seasonality_alt.png')

# Show the plot
plt.show()
```

```
/home/jawad_ahmed/anaconda3/lib/python3.7/site-packages/matplotlib/cbook/__init__.py:2064: Futu
  x[:, None]
/home/jawad_ahmed/anaconda3/lib/python3.7/site-packages/matplotlib/axes/_base.py:248: FutureWa
  x = x[:, np.newaxis]
/home/jawad_ahmed/anaconda3/lib/python3.7/site-packages/matplotlib/axes/_base.py:250: FutureWa
  y = y[:, np.newaxis]
```




### 1.0.4 Inference:

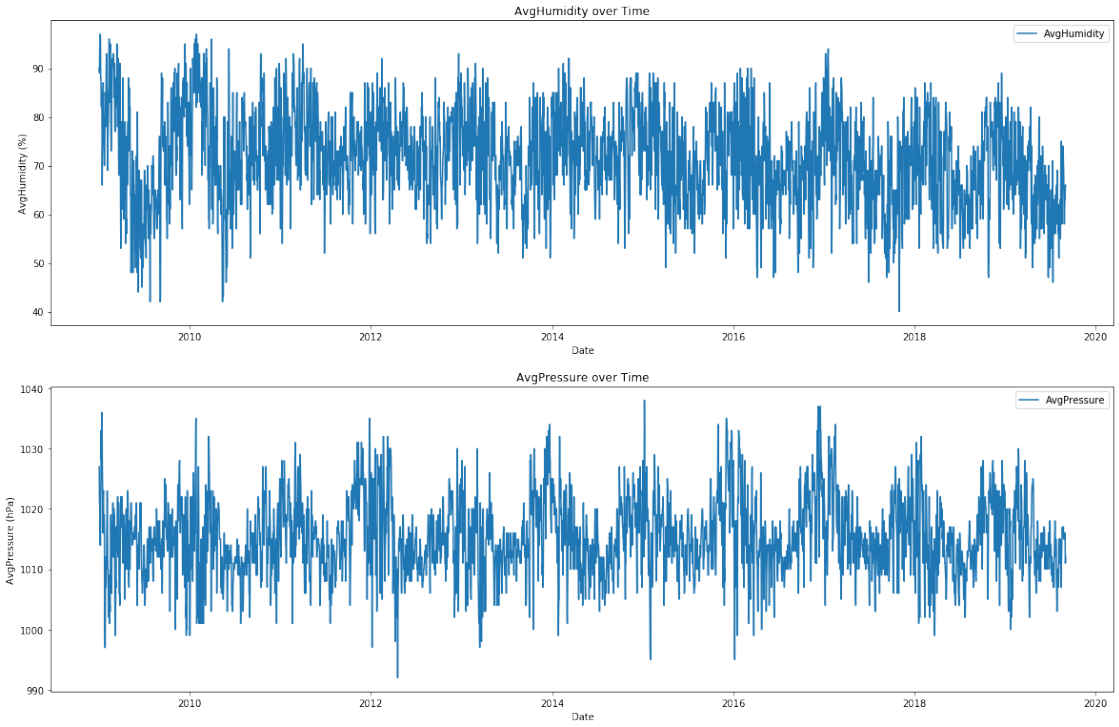
1. The wind speed seemed to higher in the winter months, particularly in December and January.

2. Wind Speed generally stays between 0 to 30 km/h.
3. Graph does not seem to have a clear seasonality.

### 1.0.5 Visualizing the Pressure and Humidity

In [15]:
```
# Set the figure size
plt.figure(figsize=(20, 20))

# Plot AvgHumidity over time
plt.subplot(311)
plt.plot(istanbul_dataset.index, istanbul_dataset['AvgHumidity'], label='AvgHumidity')
plt.title('AvgHumidity over Time')
plt.xlabel('Date')
plt.ylabel('AvgHumidity (%)')
plt.legend()

# Plot AvgPressure over time
plt.subplot(312)
plt.plot(istanbul_dataset.index, istanbul_dataset['AvgPressure'], label='AvgPressure')
plt.title('AvgPressure over Time')
plt.xlabel('Date')
plt.ylabel('AvgPressure (hPa)')
plt.legend()

# Save the figure as a PNG image
plt.savefig('istanbul_weather_humidity_pressure.png')

# Show the plot
plt.show()
```

```
/home/jawad_ahmed/anaconda3/lib/python3.7/site-packages/matplotlib/cbook/__init__.py:2064: Futu
  x[:, None]
/home/jawad_ahmed/anaconda3/lib/python3.7/site-packages/matplotlib/axes/_base.py:248: FutureWa
  x = x[:, np.newaxis]
/home/jawad_ahmed/anaconda3/lib/python3.7/site-packages/matplotlib/axes/_base.py:250: FutureWa
  y = y[:, np.newaxis]
```




### 1.0.6 Inference:

1. The AvgHumidity graph shows that there is not much seasonality in the graph
2. In the AvgHumidity there no trend seen as seen in previous one's.
3. The AvgPressure graph also have no seasonality.
4. Pressure is low at the start of the year.

### 1.0.7 Removing Outliers

```
In [16]: istanbul_dataset.describe()
```

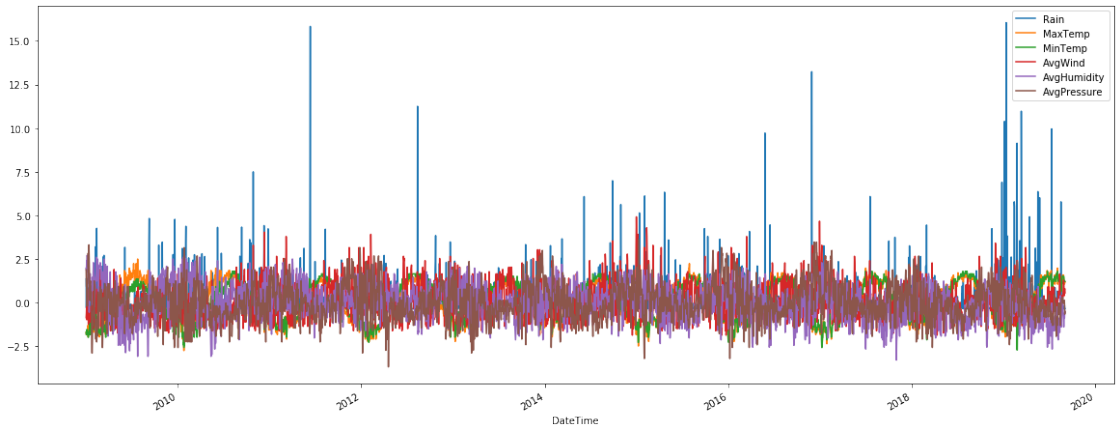
```
Out[16]:                Rain      MaxTemp      MinTemp      AvgWind  AvgHumidity  \
         count  3896.000000  3896.000000  3896.000000  3896.000000  3896.000000
         mean      0.946794    18.084189    13.774897    16.989220    71.414784
         std       2.558308     7.613318     6.865021     7.950417     9.483500
         min       0.000000    -3.000000    -5.000000     2.000000    40.000000
         25%       0.000000    12.000000     8.000000    11.000000    65.000000
         50%       0.010000    18.000000    14.000000    16.000000    71.000000
         75%       0.720000    25.000000    20.000000    22.000000    78.000000
         max      42.000000    37.000000    26.000000    56.000000    97.000000

                AvgPressure
         count  3896.000000
         mean   1015.281314
```

```
std       6.284232
min     992.000000
25%    1011.000000
50%    1015.000000
75%    1019.000000
max    1038.000000
```

In [17]:
```
numeric_cols = istanbul_dataset.select_dtypes(include=np.number).columns.tolist()
istanbul_dataset_zscore = istanbul_dataset[numeric_cols].apply(zscore)
```

In [18]:
```
# Plot the z-score normalized dataset
istanbul_dataset_zscore.plot(figsize=(20,8))
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6b7d72a320>




In [19]:
```
istanbul_dataset_zscore.describe()
```

Out[19]:

| | Rain | MaxTemp | MinTemp | AvgWind | AvgHumidity | AvgPressure |
|---|---|---|---|---|---|---|
| count | 3.896000e+03 | 3.896000e+03 | 3.896000e+03 | 3.896000e+03 | 3.896000e+03 | 3.896000e+03 |
| mean | -5.054706e-16 | -9.928175e-17 | 4.997713e-16 | -4.665444e-16 | 3.213833e-16 | 7.469669e-15 |
| std | 1.000128e+00 | 1.000128e+00 | 1.000128e+00 | 1.000128e+00 | 1.000128e+00 | 1.000128e+00 |
| min | -3.701336e-01 | -2.769738e+00 | -2.735215e+00 | -1.885580e+00 | -3.312998e+00 | -3.705195e+00 |
| 25% | -3.701336e-01 | -7.992533e-01 | -8.413140e-01 | -7.534182e-01 | -6.765021e-01 | -6.813663e-01 |
| 50% | -3.662243e-01 | -1.105953e-02 | 3.279401e-02 | -1.244396e-01 | -4.374310e-02 | |
| 75% | -8.866145e-02 | 9.084999e-01 | 9.069020e-01 | 6.303347e-01 | 6.944758e-01 | |
| max | 1.604908e+01 | 2.484887e+00 | 1.781010e+00 | 4.907389e+00 | 2.698213e+00 | |

```
50%   -4.477083e-02
75%    5.918246e-01
max    3.615653e+00
```

In [20]:
```
istanbul_dataset_zscore.head()
```

Out[20]:

| DateTime | Rain | MaxTemp | MinTemp | AvgWind | AvgHumidity | AvgPressure |
|---|---|---|---|---|---|---|
| 2019-09-02 | -0.370134 | 1.171231 | 1.198271 | 0.756130 | -0.571042 | -0.522217 |
| 2019-09-01 | -0.370134 | 1.171231 | 1.198271 | 0.504539 | -0.571042 | -0.681366 |
| 2019-08-31 | -0.174667 | 1.039866 | 1.198271 | 0.630335 | -0.887422 | -0.044771 |
| 2019-08-30 | -0.370134 | 1.171231 | 1.198271 | 0.378743 | -0.781962 | 0.114378 |
| 2019-08-29 | -0.370134 | 1.171231 | 1.343956 | 0.881926 | -1.098341 | -0.044771 |

In [21]:
```
istanbul_dataset_zscore['condition'] = istanbul_dataset['Condition']
```

In [22]:
```
istanbul_dataset_zscore.head()
```

Out[22]:

| DateTime | Rain | MaxTemp | MinTemp | AvgWind | AvgHumidity | AvgPressure | condition |
|---|---|---|---|---|---|---|---|
| 2019-09-02 | -0.370134 | 1.171231 | 1.198271 | 0.756130 | -0.571042 | -0.522217 | Partly cloudy |
| 2019-09-01 | -0.370134 | 1.171231 | 1.198271 | 0.504539 | -0.571042 | -0.681366 | Partly cloudy |
| 2019-08-31 | -0.174667 | 1.039866 | 1.198271 | 0.630335 | -0.887422 | -0.044771 | Patchy rain possible |
| 2019-08-30 | -0.370134 | 1.171231 | 1.198271 | 0.378743 | -0.781962 | 0.114378 | Partly cloudy |
| 2019-08-29 | -0.370134 | 1.171231 | 1.343956 | 0.881926 | -1.098341 | -0.044771 | Partly cloudy |

In [23]:
```
le = LabelEncoder()
istanbul_dataset_zscore['condition'] = le.fit_transform(istanbul_dataset_zscore['condi
```

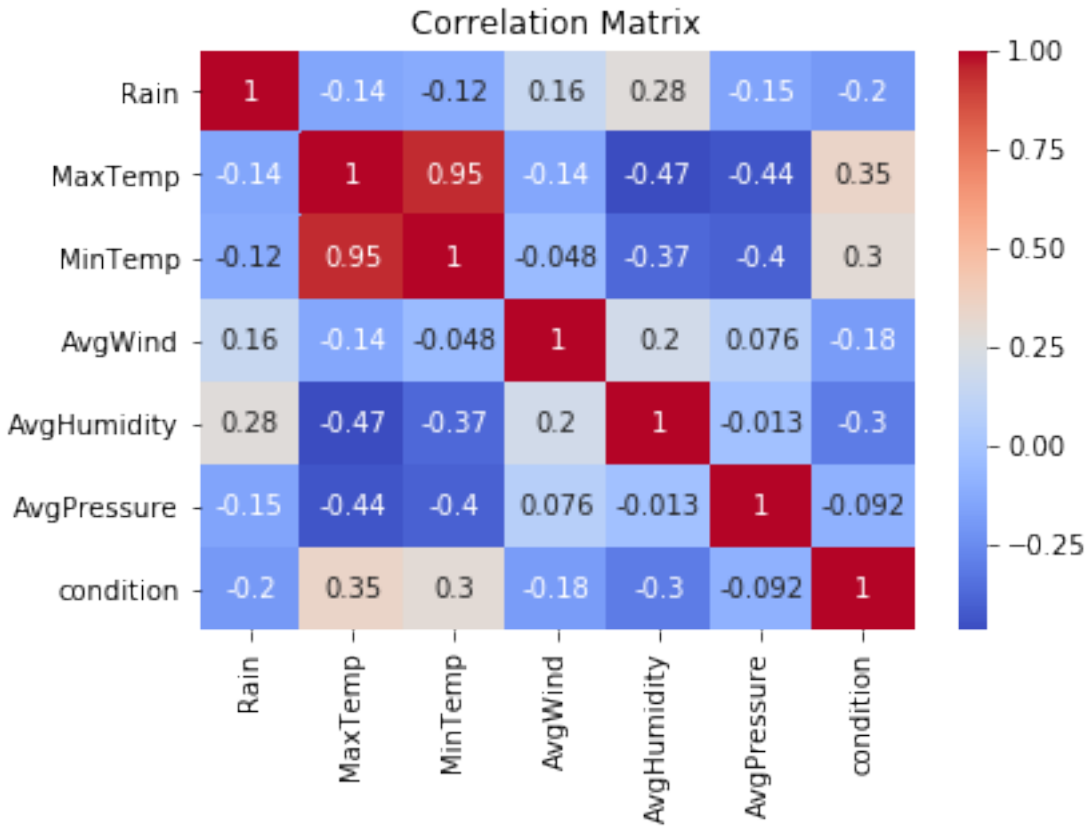
In [24]:
```
istanbul_dataset_zscore.head()
```

Out[24]:

| DateTime | Rain | MaxTemp | MinTemp | AvgWind | AvgHumidity | AvgPressure | condition |
|---|---|---|---|---|---|---|---|
| 2019-09-02 | -0.370134 | 1.171231 | 1.198271 | 0.756130 | -0.571042 | -0.522217 | |
| 2019-09-01 | -0.370134 | 1.171231 | 1.198271 | 0.504539 | -0.571042 | -0.681366 | |
| 2019-08-31 | -0.174667 | 1.039866 | 1.198271 | 0.630335 | -0.887422 | -0.044771 | |
| 2019-08-30 | -0.370134 | 1.171231 | 1.198271 | 0.378743 | -0.781962 | 0.114378 | |
| 2019-08-29 | -0.370134 | 1.171231 | 1.343956 | 0.881926 | -1.098341 | -0.044771 | |

```
2019-09-02          19
2019-09-01          19
2019-08-31          22
2019-08-30          19
2019-08-29          19
```

In [25]:
```
corr_matrix = istanbul_dataset_zscore.corr()
sns.heatmap(corr_matrix, cmap='coolwarm', annot=True)
plt.title('Correlation Matrix')
plt.show()
```




In [26]:
```
condition_save = istanbul_dataset_zscore['condition']
istanbul_dataset_zscore = istanbul_dataset_zscore.drop('condition', axis=1)
```

In [27]:
```
# Calculate the correlation matrix
corr_matrix = istanbul_dataset_zscore.corr().abs()

# Create a boolean mask to identify highly correlated features
mask = corr_matrix.mask(np.tril(np.ones([len(corr_matrix)]*2, dtype=bool))).stack().nl

# Get the column names of highly correlated features
```

```
        drop_columns = mask[mask].index.tolist()

        # # Drop the highly correlated features
        # istanbul_dataset_zscore = istanbul_dataset_zscore.drop('MaxTemp', axis=1)
```

```
In [28]: istanbul_dataset_zscore['condition'] = condition_save
```

```
In [29]: istanbul_dataset_zscore.shape
```

```
Out[29]: (3896, 7)
```

```
In [30]: istanbul_dataset_zscore.head()
```

```
Out[30]:                 Rain   MaxTemp   MinTemp   AvgWind  AvgHumidity  AvgPressure  \
        DateTime
        2019-09-02 -0.370134  1.171231  1.198271  0.756130    -0.571042    -0.522217
        2019-09-01 -0.370134  1.171231  1.198271  0.504539    -0.571042    -0.681366
        2019-08-31 -0.174667  1.039866  1.198271  0.630335    -0.887422    -0.044771
        2019-08-30 -0.370134  1.171231  1.198271  0.378743    -0.781962     0.114378
        2019-08-29 -0.370134  1.171231  1.343956  0.881926    -1.098341    -0.044771

                    condition
        DateTime
        2019-09-02         19
        2019-09-01         19
        2019-08-31         22
        2019-08-30         19
        2019-08-29         19
```

### 1.1 Augmented Dickey-Fuller (ADF) Test for Checking Data is stationary

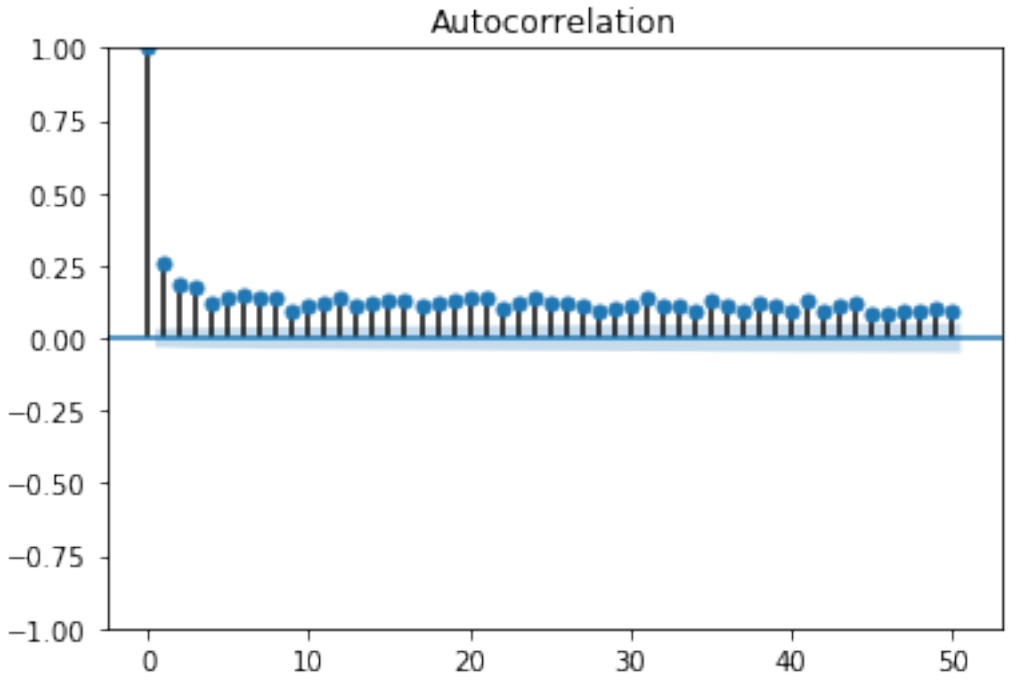
```
In [31]: from statsmodels.tsa.stattools import adfuller

        result = adfuller(istanbul_dataset_zscore['condition'])
        print('ADF Statistic: %f' % result[0])
        print('p-value: %f' % result[1])
        print('Critical Values:')
        for key, value in result[4].items():
            print('\t%s: %.3f' % (key, value))
```

```
ADF Statistic: -5.663917
p-value: 0.000001
Critical Values:
        1%: -3.432
        5%: -2.862
        10%: -2.567
```

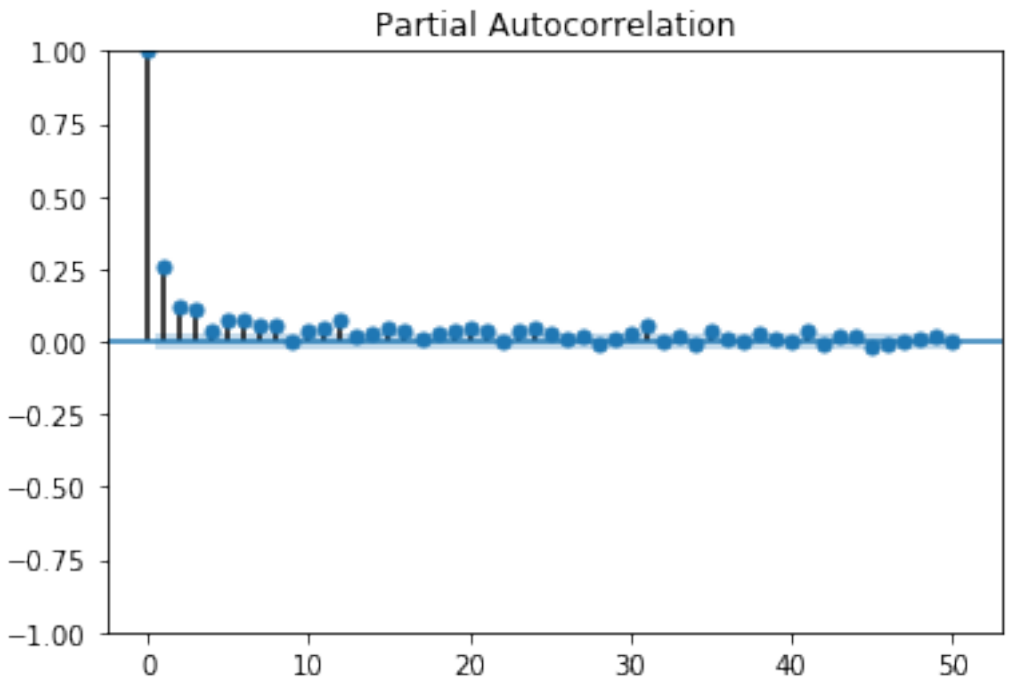
### 1.1.1 Inference:

1. Based on the value of ADF Statisics = -5.663917 s less than the critical value at the 1% significance level (-3.432) and the p-value (0.000001) is less than the significance level of 0.05, we can reject the null hypothesis that the time series is non-stationary.
2. Therefore, based on these results, we can conclude that the TARGET time series in istanbul_dataset_zscore is stationary.

In [32]:
```
# Plot ACF
plot_acf(istanbul_dataset_zscore['condition'], lags=50, alpha=0.05)
plt.show()

# Plot PACF
plot_pacf(istanbul_dataset_zscore['condition'], lags=50, alpha=0.05)
plt.show()
```




```
/home/jawad_ahmed/anaconda3/lib/python3.7/site-packages/statsmodels/graphics/tsaplots.py:353: F
  FutureWarning,
```




In [33]:
```
# Apply seasonal differencing with a lag of 12
istanbul_dataset_zscore_diff = istanbul_dataset_zscore.diff(25)

# Remove NaN values
istanbul_dataset_zscore_diff = istanbul_dataset_zscore.dropna()
```

In [34]:
```
istanbul_dataset_zscore_diff.head()
```

Out[34]:

| DateTime | Rain | MaxTemp | MinTemp | AvgWind | AvgHumidity | AvgPressure | condition |
|---|---|---|---|---|---|---|---|
| 2019-09-02 | -0.370134 | 1.171231 | 1.198271 | 0.756130 | -0.571042 | -0.522217 | 19 |
| 2019-09-01 | -0.370134 | 1.171231 | 1.198271 | 0.504539 | -0.571042 | -0.681366 | 19 |
| 2019-08-31 | -0.174667 | 1.039866 | 1.198271 | 0.630335 | -0.887422 | -0.044771 | 22 |
| 2019-08-30 | -0.370134 | 1.171231 | 1.198271 | 0.378743 | -0.781962 | 0.114378 | 19 |
| 2019-08-29 | -0.370134 | 1.171231 | 1.343956 | 0.881926 | -1.098341 | -0.044771 | 19 |

In [35]:
```
istanbul_dataset_zscore_diff.shape
```

Out[35]: (3896, 7)

In [36]:
```
# Apply seasonal differencing with a period of 12 (monthly data with yearly seasonali
seasonal_diff_istanbul_dataset = istanbul_dataset_zscore_diff.diff(periods=12)

# Drop the first 12 rows (which contain NaN values due to differencing)
seasonal_diff_istanbul_dataset = istanbul_dataset_zscore_diff.dropna()

# Print the resulting DataFrame
print(seasonal_diff_istanbul_dataset)
```

```
                Rain   MaxTemp   MinTemp   AvgWind  AvgHumidity  AvgPressure  \
DateTime
2019-09-02 -0.370134  1.171231  1.198271  0.756130    -0.571042    -0.522217
2019-09-01 -0.370134  1.171231  1.198271  0.504539    -0.571042    -0.681366
2019-08-31 -0.174667  1.039866  1.198271  0.630335    -0.887422    -0.044771
2019-08-30 -0.370134  1.171231  1.198271  0.378743    -0.781962     0.114378
2019-08-29 -0.370134  1.171231  1.343956  0.881926    -1.098341    -0.044771
...              ...       ...       ...       ...          ...          ...
2009-01-05  1.318699 -1.718813 -1.569737 -0.250235     2.698213    -0.044771
2009-01-04  0.767483 -1.718813 -1.569737 -1.005010     2.381833    -0.203920
2009-01-03 -0.338859 -1.718813 -1.569737 -0.124440     2.381833     0.910122
2009-01-02  1.381249 -1.850178 -1.861107 -0.627622     1.854534     0.910122
2009-01-01 -0.280219 -1.718813 -1.715422 -0.879214     1.959994     1.865015

            condition
DateTime
2019-09-02         19
2019-09-01         19
2019-08-31         22
2019-08-30         19
2019-08-29         19
...               ...
2009-01-05         18
2009-01-04         10
2009-01-03         18
2009-01-02         18
2009-01-01         19

[3896 rows x 7 columns]
```

## 2 Applying the ARIMA Model

In [37]:
```
train_data, test_data = train_test_split(seasonal_diff_istanbul_dataset, test_size=0.
```

In [38]:
```
train_data.shape
```

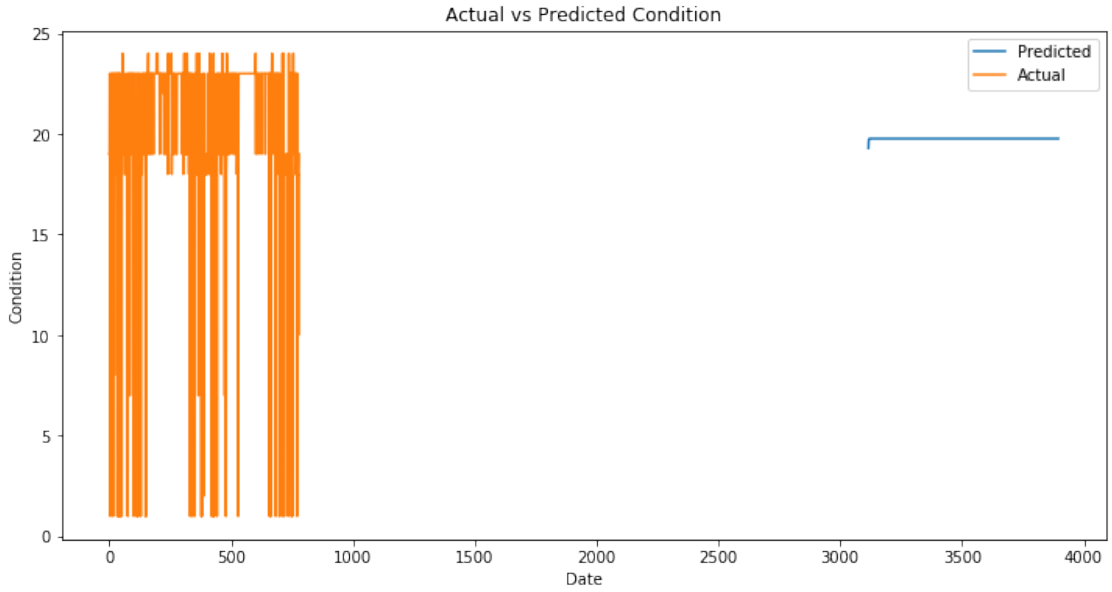
Out[38]: (3116, 7)

```
In [39]: test_data.shape

Out[39]: (780, 7)

In [40]: from sklearn.ensemble import GradientBoostingRegressor
         from statsmodels.tsa.arima.model import ARIMA
         from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

In [41]: # Define the target variable
         y_train = train_data['condition']

         # Define the features
         X_train = train_data.drop(['condition'], axis=1)

         # Define the target variable for the test set
         y_test = test_data['condition']

         # Define the features for the test set
         X_test = test_data.drop(['condition'], axis=1)

In [42]: import itertools
         import warnings
         from statsmodels.tsa.statespace.sarimax import SARIMAX

         warnings.filterwarnings("ignore")

         # Define the range of p, q, and d values to search over
         p_values = range(0, 3)
         d_values = range(0, 3)
         q_values = range(0, 3)

         # Generate all possible combinations of p, d, and q values
         pdq_values = list(itertools.product(p_values, d_values, q_values))

         # Initialize variables to store best parameters and performance
         best_pdq = None
         best_mae = float('inf')

         # Perform grid search over all combinations of p, d, and q values
         for pdq in pdq_values:
             try:
                 # Fit ARIMA model with current parameters
                 model = ARIMA(y_train, order=pdq)
                 model_fit = model.fit()

                 # Make predictions on validation set
                 y_pred = model_fit.forecast(len(y_test))

                 # Calculate performance metric
```

```
                mae = mean_absolute_error(y_test, y_pred)

                # Check if current model is better than previous best model
                if mae < best_mae:
                    best_pdq = pdq
                    best_mae = mae
            except:
                continue

    # Fit final ARIMA model with best parameters
    model = ARIMA(y_train, order=best_pdq)
    model_fit = model.fit()

    # Make predictions on test set
    y_pred = model_fit.forecast(len(y_test))

    # Calculate evaluation metrics
    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)

    # Print results
    print('Best p, d, q values:', best_pdq)
    print('MAE:', mae)
    print('RMSE:', rmse)
    print('R2 Score:', r2)
```

```
Best p, d, q values: (1, 0, 0)
MAE: 3.5203378729863837
RMSE: 5.224910673275245
R2 Score: -0.010360927879180082
```

```
In [43]: import matplotlib.pyplot as plt

    # Plot predicted and actual values
    plt.figure(figsize=(12,6))
    plt.plot(y_pred, label='Predicted')
    plt.plot(y_test.values, label='Actual')
    plt.legend()
    plt.title('Actual vs Predicted Condition')
    plt.xlabel('Date')
    plt.ylabel('Condition')
    plt.show()
```




## 3 Applyting the Exponential Smoothing Model

```
In [44]: import itertools
         from statsmodels.tsa.holtwinters import ExponentialSmoothing
         from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
         import warnings

         warnings.filterwarnings('ignore')

         # Define the range of hyperparameters to search over
         trend_values = ['add', 'mul']
         seasonal_values = ['add', 'mul', None]
         seasonal_periods_values = [7, 14]

         # Generate all possible combinations of hyperparameters
         params = list(itertools.product(trend_values, seasonal_values, seasonal_periods_values

         # Initialize variables to store best parameters and performance
         best_params = None
         best_mae = float('inf')

         # Perform grid search over all combinations of hyperparameters
         for param in params:
             try:
                 # Fit Exponential Smoothing model with current hyperparameters
                 model = ExponentialSmoothing(endog=y_train, trend=param[0], seasonal=param[1]
                 model_fit = model.fit()
```

```
                # Make predictions on validation set
                y_pred = model_fit.forecast(len(y_test))

                # Calculate performance metric
                mae = mean_absolute_error(y_test, y_pred)

                # Check if current model is better than previous best model
                if mae < best_mae:
                    best_params = param
                    best_mae = mae
            except:
                continue

        # Fit final Exponential Smoothing model with best parameters
        model = ExponentialSmoothing(endog=y_train, trend=best_params[0], seasonal=best_params
        model_fit = model.fit()

        # Make predictions on test set
        y_pred = model_fit.forecast(len(y_test))

        # Calculate evaluation metrics
        mae = mean_absolute_error(y_test, y_pred)
        rmse = np.sqrt(mean_squared_error(y_test, y_pred))
        r2 = r2_score(y_test, y_pred)

        # Print results
        print('Best parameters:', best_params)
        print('MAE:', mae)
        print('RMSE:', rmse)
        print('R2 Score:', r2)
```

```
Best parameters: ('add', 'add', 7)
MAE: 9.30364702104133
RMSE: 9.575541991817595
R2 Score: -2.393481870404652
```

```
In [45]: train_data.head()
```

Out[45]:

| DateTime | Rain | MaxTemp | MinTemp | AvgWind | AvgHumidity | AvgPressure |
|---|---|---|---|---|---|---|
| 2019-09-02 | -0.370134 | 1.171231 | 1.198271 | 0.756130 | -0.571042 | -0.522217 |
| 2019-09-01 | -0.370134 | 1.171231 | 1.198271 | 0.504539 | -0.571042 | -0.681366 |
| 2019-08-31 | -0.174667 | 1.039866 | 1.198271 | 0.630335 | -0.887422 | -0.044771 |
| 2019-08-30 | -0.370134 | 1.171231 | 1.198271 | 0.378743 | -0.781962 | 0.114378 |
| 2019-08-29 | -0.370134 | 1.171231 | 1.343956 | 0.881926 | -1.098341 | -0.044771 |

```
            condition
DateTime
2019-09-02         19
2019-09-01         19
2019-08-31         22
2019-08-30         19
2019-08-29         19
```

In [46]:
```
test_data.head()
```

Out[46]:
```
                Rain   MaxTemp   MinTemp   AvgWind  AvgHumidity  AvgPressure  \
DateTime
2011-02-19 -0.370134 -1.061985 -0.986999  0.630335     1.854534    -2.272855
2011-02-18  0.474283 -0.930619 -0.841314 -0.376031     1.010855    -0.363069
2011-02-17 -0.354496 -1.193350 -1.569737 -1.130805     0.061717     0.432676
2011-02-16 -0.334950 -1.718813 -1.569737  1.007722     0.061717     0.432676
2011-02-15 -0.331040 -1.587447 -1.424053  1.385109     0.694476    -0.522217

            condition
DateTime
2011-02-19         19
2011-02-18         19
2011-02-17         23
2011-02-16         19
2011-02-15          1
```

# 4 Applying Support Vector Regression Model

In [47]:
```
from sklearn.svm import SVR
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

In [48]:
```
# Separate X (features) and y (target) for training and test sets
X_train, y_train = train_data.drop('condition', axis=1), train_data['condition']
X_test, y_test = test_data.drop('condition', axis=1), test_data['condition']
```

In [49]:
```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid to search over
param_grid = {
    'C': [1, 10, 100, 1000],
    'gamma': [0.1, 0.01, 0.001],
    'epsilon': [0.1, 0.01, 0.001]
}

# Create an instance of SVR
svr = SVR(kernel='rbf')

# Perform a grid search with cross-validation to find the best hyperparameters
```

```
grid_search = GridSearchCV(estimator=svr, param_grid=param_grid, cv=5, scoring='neg_m
grid_search.fit(X_train, y_train)

# Print the best hyperparameters found
print(f"Best hyperparameters: {grid_search.best_params_}")

# Train a new model using the best hyperparameters found
best_svr = SVR(kernel='rbf', C=grid_search.best_params_['C'], gamma=grid_search.best_
               epsilon=grid_search.best_params_['epsilon'])
best_svr.fit(X_train, y_train)

# Predict on test set
y_pred = best_svr.predict(X_test)

# Evaluate performance
mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)

print(f"MAE: {mae:.2f}")
print(f"RMSE: {rmse:.2f}")
print(f"R2: {r2:.2f}")
```

```
Best hyperparameters: {'C': 10, 'epsilon': 0.1, 'gamma': 0.1}
MAE: 2.00
RMSE: 4.57
R2: 0.23
```

In [ ]: