TIME SERIES MODELING & ANALYSIS

Instructor Name: Reza Jafari

HW#: 9

Submitted by Dinesh Kumar Padmanabhan

Date: 20-Nov-2020

Data

The data is obtained from UCI Machine Learning Repository. It was recorded by 5 metal oxide chemical sensors located in a significantly polluted area in an Italian city, and I will analyze one of them, CO. The dataset contains 9358 instances of hourly averaged responses spreading from March 2004 to February 2005.Below is the attributes description loaded from UCI Machine Learning Repository. Attribute Information:

0 Date (DD/MM/YYYY)

1 Time (HH.MM.SS)

2 True hourly averaged concentration CO in mg/m^3 (reference analyzer)

3 PT08.S1 (tin oxide) hourly averaged sensor response (nominally CO targeted)

4 True hourly averaged overall Non Metanic HydroCarbons concentration in microg/m^3

5 True hourly averaged Benzene concentration in microg/m^3 (reference analyzer)

6 PT08.S2 (titania) hourly averaged sensor response (nominally NMHC targeted)

7 True hourly averaged NOx concentration in ppb (reference analyzer)

8 PT08.S3 (tungsten oxide) hourly averaged sensor response (nominally NOx targeted)

9 True hourly averaged NO2 concentration in microg/m^3 (reference analyzer)

10 PT08.S4 (tungsten oxide) hourly averaged sensor response (nominally NO2 targeted)

11 PT08.S5 (indium oxide) hourly averaged sensor response (nominally O3 targeted)

12 Temperature in Â°C

13 Relative Humidity (%)

14 AH Absolute Humidity

ANSWERS TO ASKED QUESTIONS

File - unknown

```
 1 C:\ProgramData\Anaconda3\python.exe "C:\Program Files\
   JetBrains\PyCharm 2019.3.1\plugins\python\helpers\pydev\
   pydevconsole.py" --mode=client --port=61794
 2
 3 import sys; print('Python %s on %s' % (sys.version, sys.
   platform))
 4 sys.path.extend(['C:\\Users\\nsree_000\\Desktop\\Python-
   Quiz', 'C:/Users/nsree_000/Desktop/Python-Quiz'])
 5
 6 Python 3.7.4 (default, Aug  9 2019, 18:34:13) [MSC v.1915
   64 bit (AMD64)]
 7 Type 'copyright', 'credits' or 'license' for more
   information
 8 IPython 7.8.0 -- An enhanced Interactive Python. Type '?'
   for help.
 9 PyDev console: using IPython 7.8.0
10
11 Python 3.7.4 (default, Aug  9 2019, 18:34:13) [MSC v.1915
   64 bit (AMD64)] on win32
12 In[2]: runfile('C:/Users/nsree_000/Desktop/Python-Quiz/TIME
    SERIES/TERM PROJECT/url.py', wdir='C:/Users/nsree_000/
   Desktop/Python-Quiz/TIME SERIES/TERM PROJECT')
13
14
15 loading dataset....
16
17
18 Download complete....
19
20
21 --------------------PREPROCESSING--------------------
22          Date      Time CO(GT)  ...      AH  Unnamed: 15
   Unnamed: 16
23 0  10/03/2004  18.00.00    2,6  ...  0,7578          NaN
            NaN
24 1  10/03/2004  19.00.00      2  ...  0,7255          NaN
            NaN
25 2  10/03/2004  20.00.00    2,2  ...  0,7502          NaN
            NaN
26 3  10/03/2004  21.00.00    2,2  ...  0,7867          NaN
            NaN
27 4  10/03/2004  22.00.00    1,6  ...  0,7888          NaN
            NaN
28
29 [5 rows x 17 columns]
30
```

File - unknown

```
<bound method DataFrame.info of             Date      Time
    CO(GT)  PT08.S1(CO)  ...  PT08.S5(O3)     T    RH      AH
0     10/03/2004  18.00.00    2,6       1360.0  ...
   1268.0  13,6  48,9  0,7578
1     10/03/2004  19.00.00      2       1292.0  ...
   972.0  13,3  47,7  0,7255
2     10/03/2004  20.00.00    2,2       1402.0  ...
   1074.0  11,9  54,0  0,7502
3     10/03/2004  21.00.00    2,2       1376.0  ...
   1203.0  11,0  60,0  0,7867
4     10/03/2004  22.00.00    1,6       1272.0  ...
   1110.0  11,2  59,6  0,7888
           ...       ...    ...          ...  ...          .
   ..   ...   ...     ...
9466         NaN       NaN    NaN          NaN
     ...          NaN   NaN   NaN      NaN
9467         NaN       NaN    NaN          NaN
     ...          NaN   NaN   NaN      NaN
9468         NaN       NaN    NaN          NaN
     ...          NaN   NaN   NaN      NaN
9469         NaN       NaN    NaN          NaN
     ...          NaN   NaN   NaN      NaN
9470         NaN       NaN    NaN          NaN
     ...          NaN   NaN   NaN      NaN

[9471 rows x 15 columns]>

 Date              object
Time              object
CO(GT)            object
PT08.S1(CO)      float64
NMHC(GT)         float64
C6H6(GT)          object
PT08.S2(NMHC)    float64
NOx(GT)          float64
PT08.S3(NOx)     float64
NO2(GT)          float64
PT08.S4(NO2)     float64
PT08.S5(O3)      float64
T                 object
RH                object
AH                object
dtype: object


Shape of the Dataset: (9471, 15)
```

```
 65
 66
 67 Date             114
 68 Time             114
 69 CO(GT)           114
 70 PT08.S1(CO)      114
 71 NMHC(GT)         114
 72 C6H6(GT)         114
 73 PT08.S2(NMHC)    114
 74 NOx(GT)          114
 75 PT08.S3(NOx)     114
 76 NO2(GT)          114
 77 PT08.S4(NO2)     114
 78 PT08.S5(O3)      114
 79 T                114
 80 RH               114
 81 AH               114
 82 dtype: int64
 83
 84       Date Time  CO(GT)  PT08.S1(CO)  ...  PT08.S5(O3)   T
      RH  AH
 85 9357  NaN  NaN     NaN          NaN  ...          NaN NaN
    NaN NaN
 86 9358  NaN  NaN     NaN          NaN  ...          NaN NaN
    NaN NaN
 87 9359  NaN  NaN     NaN          NaN  ...          NaN NaN
    NaN NaN
 88 9360  NaN  NaN     NaN          NaN  ...          NaN NaN
    NaN NaN
 89 9361  NaN  NaN     NaN          NaN  ...          NaN NaN
    NaN NaN
 90
 91 [5 rows x 15 columns]
 92 Shape of the Dataset after null value removal (9357, 15)
 93 Date                0
 94 Time                0
 95 CO(GT)           1683
 96 PT08.S1(CO)       366
 97 NMHC(GT)         8443
 98 C6H6(GT)          366
 99 PT08.S2(NMHC)     366
100 NOx(GT)          1639
101 PT08.S3(NOx)      366
102 NO2(GT)          1642
103 PT08.S4(NO2)      366
104 PT08.S5(O3)       366
```

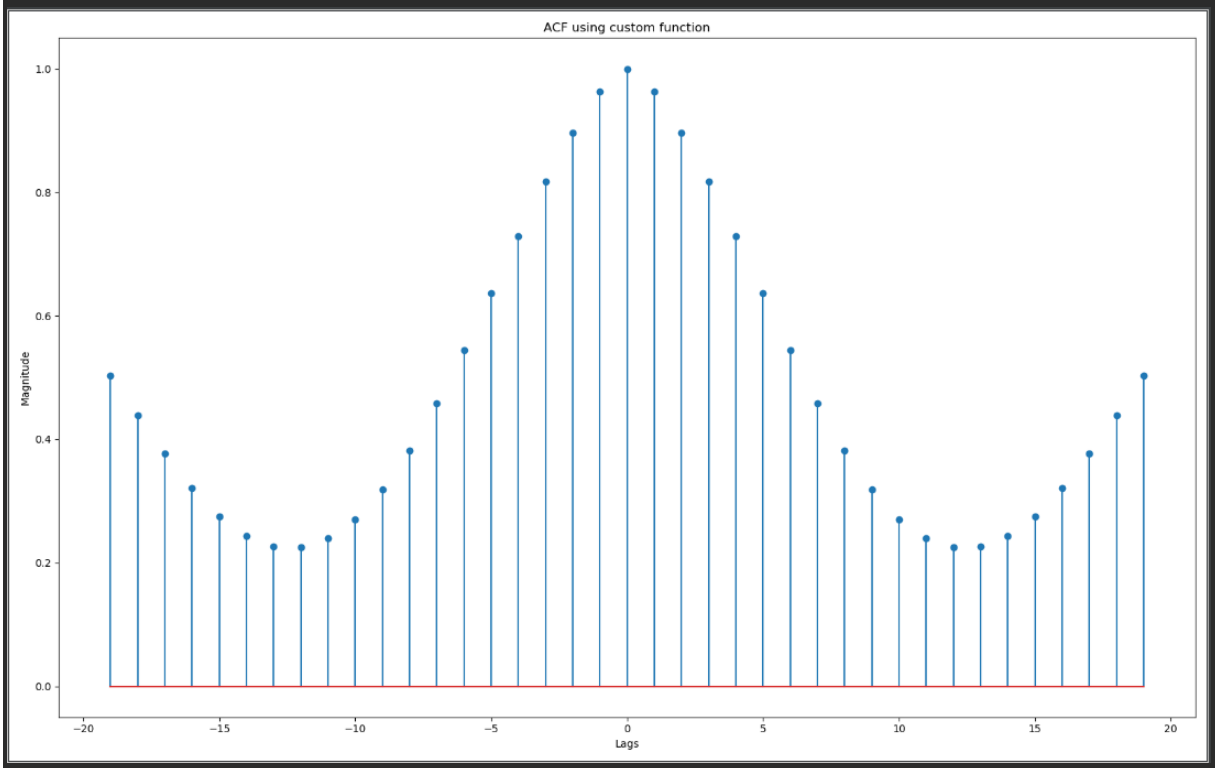
```
105 T                366
106 RH               366
107 AH               366
108 dtype: int64
109 Int64Index([   0,    1,    2,    3,    4,    5,    6,    7
    ,    8,    9,
110              ...
111             9347, 9348, 9349, 9350, 9351, 9352, 9353, 9354
    , 9355, 9356],
112            dtype='int64', length=9357)
113 Date                 0
114 Time                 0
115 CO(GT)            1683
116 PT08.S1(CO)       366
117 NMHC(GT)          8443
118 C6H6(GT)          366
119 PT08.S2(NMHC)     366
120 NOx(GT)           1639
121 PT08.S3(NOx)      366
122 NO2(GT)           1642
123 PT08.S4(NO2)      366
124 PT08.S5(O3)       366
125 T                 366
126 RH                366
127 AH                366
128 Datetime            0
129 dtype: int64
130 <bound method NDFrame.head of
    Date       Time  ...       AH             Datetime
131 2004-03-10 18:00:00  10/03/2004  18.00.00  ...  0.7578  10
    /03/2004 18.00.00
132 2004-03-10 19:00:00  10/03/2004  19.00.00  ...  0.7255  10
    /03/2004 19.00.00
133 2004-03-10 20:00:00  10/03/2004  20.00.00  ...  0.7502  10
    /03/2004 20.00.00
134 2004-03-10 21:00:00  10/03/2004  21.00.00  ...  0.7867  10
    /03/2004 21.00.00
135 2004-03-10 22:00:00  10/03/2004  22.00.00  ...  0.7888  10
    /03/2004 22.00.00
136                          ...       ...  ...     ...
                ...
137 2005-04-04 10:00:00  04/04/2005  10.00.00  ...  0.7568  04
    /04/2005 10.00.00
138 2005-04-04 11:00:00  04/04/2005  11.00.00  ...  0.7119  04
    /04/2005 11.00.00
139 2005-04-04 12:00:00  04/04/2005  12.00.00  ...  0.6406  04
```

```
/04/2005 12.00.00
2005-04-04 13:00:00  04/04/2005  13.00.00  ...  0.5139  04
/04/2005 13.00.00
2005-04-04 14:00:00  04/04/2005  14.00.00  ...  0.5028  04
/04/2005 14.00.00

[9357 rows x 16 columns]>
AFTER MEAN
 Date                0
Time                 0
CO(GT)            1683
PT08.S1(CO)        366
NMHC(GT)          8443
C6H6(GT)           366
PT08.S2(NMHC)      366
NOx(GT)           1639
PT08.S3(NOx)       366
NO2(GT)           1642
PT08.S4(NO2)       366
PT08.S5(O3)        366
T                  366
RH                 366
AH                 366
Datetime             0
dtype: int64


THE CLEANED DATASET AFTER PREPROCESSING:
                      CO(GT)  PT08.S1(CO)    NMHC(GT
)  ...     T    RH      AH
2004-03-10 18:00:00     2.6       1360.0  150.000000
     ...  13.6  48.9  0.7578
2004-03-10 19:00:00     2.0       1292.0  112.000000
     ...  13.3  47.7  0.7255
2004-03-10 20:00:00     2.2       1402.0   88.000000
     ...  11.9  54.0  0.7502
2004-03-10 21:00:00     2.2       1376.0   80.000000
     ...  11.0  60.0  0.7867
2004-03-10 22:00:00     1.6       1272.0   51.000000
     ...  11.2  59.6  0.7888
                       ...          ...         ...  ...   .
..   ...     ...
2005-04-04 10:00:00     3.1       1314.0  218.811816
     ...  21.9  29.3  0.7568
2005-04-04 11:00:00     2.4       1163.0  218.811816
     ...  24.3  23.7  0.7119
```

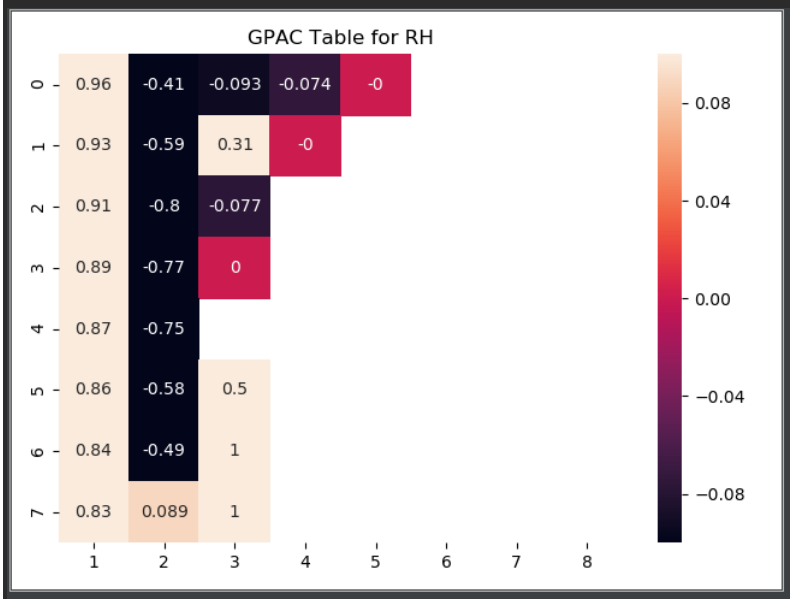
```
2005-04-04 12:00:00     2.4       1142.0  218.811816
     ...  26.9  18.3  0.6406
2005-04-04 13:00:00     2.1       1003.0  218.811816
     ...  28.3  13.5  0.5139
2005-04-04 14:00:00     2.2       1071.0  218.811816
     ...  28.5  13.1  0.5028

[9357 rows x 13 columns]

The dimension of train data is:
(7485, 13)

The dimension of test data is:
(1872, 13)

                         RH
2004-03-10 18:00:00  48.9
2004-03-10 19:00:00  47.7
2004-03-10 20:00:00  54.0
2004-03-10 21:00:00  60.0
2004-03-10 22:00:00  59.6


ADF Test Statistic : -7.39116419022283
p-value : 7.993979270511995e-11
#Lags Used : 38
Number of Observations Used : 9318
strong evidence against the null hypothesis(Ho), reject
    the null hypothesis. Data has no unit root and is
    stationary

GPAC Table:
          1        2        3        4    5   6   7   8
0  0.96257 -0.40722 -0.09333 -0.07407 -0.0 NaN NaN NaN
1  0.93151 -0.59044  0.30952 -0.00000  NaN NaN NaN NaN
2  0.91105 -0.80294 -0.07692      NaN  NaN NaN NaN NaN
3  0.89167 -0.77010  0.00000      NaN  NaN NaN NaN NaN
4  0.87332 -0.74817      inf      NaN  NaN NaN NaN NaN
5  0.85569 -0.58384  0.50000      NaN  NaN NaN NaN NaN
6  0.84191 -0.49266  1.00000      NaN  NaN NaN NaN NaN
7  0.83248  0.08936  1.00000      NaN  NaN NaN NaN NaN
```

```
#%%=============================================================================
# 1: Pick only the dependent variable. You don't need to involve features
(independent variables) for this homework.
# %%-----------------------------------------------------------------------------

Relative Humidity

#%%=============================================================================
# 2: Perform an ADF test to see if the dependent variable is stationary. (95% confidence)
# %%-----------------------------------------------------------------------------

ADF Test Statistic : -7.39116419022283
p-value : 7.993979270511995e-11
#Lags Used : 38
Number of Observations Used : 9318
strong evidence against the null hypothesis(Ho), reject the null hypothesis.
Data has no unit root and is stationary

#%%=============================================================================
# 3: If the data set is stationary skip to step 5.
# 4: If the data set is not stationary the you need to make it stationary using the techniques
learned in class. Skip to step 5, once the data set passes the ADF test.
# %%-----------------------------------------------------------------------------

N/A
```

```
#%%==========================================================================
# 5: Estimate the ACF and plot the ACF for 20 lags
# %%--------------------------------------------------------------------------
```




```
#%%==========================================================================
# 6: Display the GPAC table for k=8 and j=8 in your report.
# %%--------------------------------------------------------------------------
GPAC Table:
         1        2        3        4    5   6   7   8
0  0.96257 -0.40722 -0.09333 -0.07407 -0.0 NaN NaN NaN
1  0.93151 -0.59044  0.30952 -0.00000  NaN NaN NaN NaN
2  0.91105 -0.80294 -0.07692      NaN  NaN NaN NaN NaN
3  0.89167 -0.77010  0.00000      NaN  NaN NaN NaN NaN
4  0.87332 -0.74817      inf      NaN  NaN NaN NaN NaN
```



```
#%%=========================================================================================
# 7: Estimate the na and nb by looking at the GPAC table. You need to come up with at least one
set for na and nb (it is possible to see more than one pattern). Highlight the potential pattern(s).
# %%-------------------------------------------------------------------------------------
Potential Patterns
 na = 2
 nb = 1
```

APPENDIX

```
#Homework 8 : Partial Correlation Coefficient

import os
import pandas as pd
from requests import get
from io import BytesIO
from zipfile import ZipFile
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
from statsmodels.tsa.seasonal import STL
from functions import *
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.stattools import adfuller
from pandas.plotting import register_matplotlib_converters
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings("ignore")
register_matplotlib_converters()


#%%=============================================================================
# DATA SET DESCRIPTION
# %%------------------------------------------------------------------------------
'''
1: Objective:
To predict the Relative Humidity of a given point of time based on the all other
attributes affecting the change  in RH
'''
#%%=============================================================================
# LOAD THE DATASET
# %%------------------------------------------------------------------------------
print('\n')
print("loading dataset....")
if "AirQualityUCI" not in os.listdir():
    request = get('https://archive.ics.uci.edu/ml/machine-learning-
databases/00360/AirQualityUCI.zip')
    zip_file = ZipFile(BytesIO(request.content))
    zip_file.extractall()
print('\n')
df = pd.read_csv("AirQualityUCI.csv", sep = ';',infer_datetime_format=True)
print("Download complete....")

#%%=============================================================================
# PREPROCESSING
# %%------------------------------------------------------------------------------
print('\n')
print(20 * "-" + "PREPROCESSING" + 20 * "-")
print(df.head(5))

# Removing last two Unnamed columns
df = df.drop(['Unnamed: 15','Unnamed: 16'], axis = 1)
print('\n',df.info)
```

```
# Changing the datatype from object to float
print('\n',df.dtypes)
df['CO(GT)'] = df['CO(GT)'].str.replace(',', '.').astype(float)
df['C6H6(GT)'] = df['C6H6(GT)'].str.replace(',','.').astype(float)
df['T'] = df['T'].str.replace(',', '.').astype(float)
df['RH'] = df['RH'].str.replace(',', '.').astype(float)
df['AH'] = df['AH'].str.replace(',', '.').astype(float)

# Dimension of the Dataset
print('\n')
print('Shape of the Dataset:',df.shape)
# %%--------------------------------------------------------------------------------


print('\n')
# Estimating null values
print(df.isnull().sum())

# Remving null values
null_data = df[df.isnull().any(axis=1)]
print('\n',null_data.head())
df= df.dropna()
# print(df.head)
print('Shape of the Dataset after null value removal',df.shape)

# Replacing -200 with nan
df = df.replace(-200,np.nan)
print(df.isnull().sum())

# Appending date and time
print(df.index)
df.loc[:,'Datetime'] = df['Date'] + ' ' + df['Time']
DateTime = []
for x in df['Datetime']:
    DateTime.append(datetime.strptime(x,'%d/%m/%Y %H.%M.%S'))
datetime = pd.Series(DateTime)
df.index = datetime
# print(df.head())
# print('AFTER',df.dtypes)
df = df.replace(-200, np.nan)
print(df.isnull().sum())
print(df.head)


# %%--------------------------------------------------------------------------------
# CREATING PROCESSED DATAFRAME
# SD = df['Date']
# ST = df['Time']
S0 = df['CO(GT)'].fillna(df['PT08.S1(CO)'].mean())
S1 = df['PT08.S1(CO)'].fillna(df['PT08.S1(CO)'].mean())
S2 = df['NMHC(GT)'].fillna(df['NMHC(GT)'].mean())
S3 = df['C6H6(GT)'].fillna(df['C6H6(GT)'].mean())
S4 = df['PT08.S2(NMHC)'].fillna(df['PT08.S1(CO)'].mean())
S5 = df['NOx(GT)'].fillna(df['NOx(GT)'].mean())
S6 = df['PT08.S3(NOx)'].fillna(df['PT08.S1(CO)'].mean())
S7 = df['NO2(GT)'].fillna(df['NO2(GT)'].mean())
S8 = df['PT08.S4(NO2)'].fillna(df['PT08.S1(CO)'].mean())
S9 = df['PT08.S5(O3)'].fillna(df['PT08.S1(CO)'].mean())
S10 = df['T'].fillna(df['T'].mean())
```

```
S11 = df['RH'].fillna(df['RH'].mean())
S12 = df['AH'].fillna(df['AH'].mean())

print('AFTER MEAN\n',df.isnull().sum())
print('\n')
df = pd.DataFrame({'CO(GT)':S0,'PT08.S1(CO)':S1,'NMHC(GT)':S2, 'C6H6(GT)':S3,
'PT08.S2(NMHC)':S4, 'NOx(GT)':S5,
                   'PT08.S3(NOx)':S6, 'NO2(GT)':S7,  'PT08.S4(NO2)':S8,
'PT08.S5(O3)':S9, 'T':S10, 'RH':S11, 'AH':S12 })
print("THE CLEANED DATASET AFTER PREPROCESSING:\n", df)

# df.index = datetime
# df.to_csv("AQI.csv")

# print('CLEANDED DATASET:\n')
# print(df.isnull().sum())
# print('CLEANDED DATASET:\n',df.head)
#
%%=============================================================================
# HOMEWORK 9 - Order estimation using GPAC Table
# %%-----------------------------------------------------------------------------
# split into train and test(20%) dataset
train, test = split_df_train_test(df, 0.2)
print()
print("The dimension of train data is:")
print(train.shape)
# dimension of test data
print()
print("The dimension of test data is:")
print(test.shape)

# Created Dataframe for Dependent variable and time
df_rh = pd.DataFrame({'RH':S11})
# df.to_csv("AirQuality_processed_rh.csv")
print('\n',df_rh.head())

# DEPENDENT VARIABLE V/S TIME
plt.figure(figsize=(16,10))
plt.plot(df_rh, 'b-', label = 'RH')
plt.xlabel('Time | March 2004- February 2005')
plt.ylabel('RH')
plt.title('Time Series plot of Relative humidity')
plt.legend(loc='best')
plt.show()

# AUTO CORRELATION USING BULTIN FUNCTION
plot_acf(df_rh)
plt.xlabel('Lags')
plt.ylabel('Magnitude')
plt.title('ACF using statsmodel')
plt.show()

# AUTO CORRELATION USING CREATED FUNCTION
y = df['RH']
k = 20
acfcal = auto_corr_cal(y,k)
# acfcal = cal_auto_corr(y,k)
acfplotvals = acfcal[::-1] + acfcal[1:]
plt.figure(figsize=(16,10))
```

```
plt.stem(range(-(k - 1), k), acfplotvals)
plt.xlabel('Lags')
plt.ylabel('Magnitude')
plt.title('ACF using custom function')
plt.show()

corr = df.corr()
ax = sns.heatmap(corr, vmin=-1, vmax=1, center=0, square=True)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, horizontalalignment='right')
plt.figure(figsize=(15,10))
plt.show()

print('\n')
test_result = adfuller(df['RH'])
adfuller_test(df['RH'])

j = 8
k = 8
lags = j + k

y_mean = np.mean(train['RH'])
y = np.subtract(y_mean, df['RH'])
actual_output = np.subtract(y_mean, test['RH'])
ry = auto_corr_cal(y, lags)

# create GPAC Table
gpac_table = create_gpac_table(j, k, ry)
print()
print("GPAC Table:")
print(gpac_table.to_string())
print()
plot_heatmap(gpac_table, "GPAC Table for RH")


%%=============================================================================
# functions
# %%-----------------------------------------------------------------------------

#Autocorrelation Function
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from statsmodels.tsa.stattools import adfuller
def split_df_train_test(df, test_size, random_seed=42):

    train, test = train_test_split(df, shuffle=False, test_size=test_size,
random_state=random_seed)
    return train, test


def auto_corr(y,k):
    T = len(y)
    y_mean = np.mean(y)
    res_num = 0
    res_den = 0
```

```
    for t in range(k,T):
        res_num += (y[t] - y_mean) * (y[t-k] - y_mean)

    for t in range(0,T):
        res_den += (y[t] - y_mean)**2

    res = res_num/res_den
    return res

def auto_corr_cal(y,k):
    res = []
    for t in range(0,k):
        result = auto_corr(y,t)
        res.append(result)
    return res


def adfuller_test(RH):
    result = adfuller(RH)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations
Used']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null
hypothesis. Data has no unit root and is stationary")
    else:
        print("weak evidence against null hypothesis, time series has a unit root,
indicating it is non-stationary ")

def get_max_denominator_indices(j, k_scope):
    # create denominator indexes based on formula for GPAC
    denominator_indices = np.zeros(shape=(k_scope, k_scope), dtype=np.int64)

    for k in range(k_scope):
        denominator_indices[:, k] = np.arange(j - k, j + k_scope - k)

    return denominator_indices

def get_apt_denominator_indices(max_denominator_indices, k):
    apt_denominator_indices = max_denominator_indices[-k:, -k:]
    return apt_denominator_indices


def get_numerator_indices(apt_denominator_indices, k):
    numerator_indices = np.copy(apt_denominator_indices)
    # take the 0,0 indexed value and then create a range of values from
(indexed_value+1, indexed_value+k)
    indexed_value = numerator_indices[0, 0]
    y_matrix = np.arange(indexed_value + 1, indexed_value + k + 1)

    # replace the last column with this new value
    numerator_indices[:, -1] = y_matrix

    return numerator_indices

def get_ACF_by_index(numpy_indices, acf):
    # select values from an array based on index specified
    result = np.take(acf, numpy_indices)
```

```
    return result

def get_phi_value(denominator_indices, numerator_indices, ry, precision=5):
    # take the absolute values since when computing phi value, we use ACF and ACF
is symmetric in nature
    denominator_indices = np.abs(denominator_indices)
    numerator_indices = np.abs(numerator_indices)

    # replace the indices with the values of ACF
    denominator = get_ACF_by_index(denominator_indices, ry)
    numerator = get_ACF_by_index(numerator_indices, ry)

    # take the determinant
    denominator_det = np.round(np.linalg.det(denominator), precision)
    numerator_det = np.round(np.linalg.det(numerator), precision)

    # divide it and return the value of phi
    return np.round(np.divide(numerator_det, denominator_det), precision)

def create_gpac_table(j_scope, k_scope, ry, precision=5):
    # initialize gpac table
    gpac_table = np.zeros(shape=(j_scope, k_scope), dtype=np.float64)

    for j in range(j_scope):
        # create the largest denominator
        max_denominator_indices = get_max_denominator_indices(j, k_scope)

        for k in range(1, k_scope + 1):
            #  slicing largest denominator as required
            apt_denominator_indices =
get_apt_denominator_indices(max_denominator_indices, k)

            # for numerator replace denominator's last columnn with index starting
from j+1 upto k times
            numerator_indices = get_numerator_indices(apt_denominator_indices, k)

            # compute phi value
            phi_value = get_phi_value(apt_denominator_indices, numerator_indices,
ry, precision)
            gpac_table[j, k - 1] = phi_value

    gpac_table_pd = pd.DataFrame(data=gpac_table, columns=[k for k in range(1,
k_scope + 1)])

    return gpac_table_pd


def plot_heatmap(corr_df, title, xticks=None, yticks=None, x_axis_rotation=0,
annotation=True):
    sns.heatmap(corr_df, annot=annotation)
    plt.title(title)
    if xticks is not None:
        plt.xticks([i for i in range(len(xticks))], xticks,
rotation=x_axis_rotation)
    if yticks is not None:
        plt.yticks([i for i in range(len(yticks))], yticks)
    plt.show()
```