

TIME SERIES MODELING & ANALYSIS

Instructor Name: Reza Jafari

Lab#: 6

Submitted by Dinesh Kumar Padmanabhan

Date: 23-Oct-2020

ABSTRACT

In LAB #6, the main purpose was to write python program that applies the time series decomposition methods discussed in the lecture on the real dataset. Different decomposition methods discussed in the lecture viz Moving average, Weighted moving average, Classical Decomposition, X11 decomposition, SEATS (Seasonal Extraction in ARIMA time series) decomposition and STL (Seasonal and trend decomposition using Loess). using 'Air passengers' datasets, we plotted various time series plots and made a calculation to estimate trend cycle, seasonality and finally the strength of the trend.

INTRODUCTION

Time series data can exhibit a variety of patterns and it is useful in forecasting to split a time series into several components, each representing an underlying pattern category such as:

Trend: pattern exists when there is a long-term increase or decrease in the data.

Cyclic: pattern exists when data exhibits rise and falls that are not of fixed period.

Seasonal: pattern exits when a series is influenced by seasonal factors(e.g., the quarter of the year, the month, or day of the week)

For this lab the 'Air passengers.csv' dataset will be used. The main purpose of this LAB is to write a python program that applies the time series decomposition methods discussed in the lecture on the dataset. Different decomposition methods discussed in the lecture viz Moving average, Weighted moving average, Classical Decomposition, X11 decomposition, SEATS (Seasonal Extraction in ARIMA time series) decomposition and STL (Seasonal and trend decomposition using Loess). A moving average of order m (m-MA) which estimates the trend-cycle at time t by averaging values of times series within k periods of t can be written as:

$$\hat{T}_t = \frac{1}{m} \sum_{j=-k}^k y_{t+j}$$

where m = 2k + 1

A time series decomposition can be used to measure the strength of trend . The rubric to measure the strength of the trend is given as :

$$F_T = \max\{0, 1 - \frac{Var(R_t)}{Var(T_t + R_t)}\}$$

METHOD, THEORY & PROCEDURES

Method:

1. Programming Language: Python

Libraries used: Some basic libraries used for analysis & model building are mentioned below

<u>library(Numpy)</u> - large collection of high-level mathematical functions to operate on these arrays.

library (Pandas) - For Data manipulation and analysis

<u>library(Matplotlib)</u> – is a system for declaratively creating graphics

library(Math) -To Compute mathematical calculations

<u>library</u> (statsmodels) – Import statistical models

Theory:

The main purpose of this LAB is to write a python program that applies the time series decomposition methods.

Procedure:

I shall be looking at the results of various time series decomposition methods and time series plots and infer about it in my analysis. And through my exploration I shall try to identify which methods perform better and draw inferences.

The Dataset will be explored in following stages:

- 1. Data Exploration (EDA) looking at the models and making inferences about the data.
- 2. **Data Visualization** Plotting different time series plots for the regression method and forecast accuracy.
- 3. **Testing** Running Autocorrelation, Pearson correlation and ADF test to identify the correlation and stationarity.

ANSWERS TO QUESTIONS

```
File - Homework6
 1 ssh://ubuntu@54.172.90.76:22/usr/bin/python3 -u /home/
   ubuntu/ML2-AWS/QUIZ/Homework6.py
 2 Enter the order of moving average: 1
 4 Sorry!! order 1,2 will not be accepted
 5 Enter the order of moving average: 3
 6 3
 7 Result of 3-MA is: [1297.4126666666666, 1426.757000000000003
    1458.9603333333334, 1445.925, 1393.615, 1303.
   8996666666665, 1345.865, 1438.073, 1561.1366666666665, 1569
   .122333333335, 1558.486, 1486.0756666666666, 1390.
   140666666667, 1396.385999999997, 1475.838666666665, 1578.
   341666666665, 1576.525666666664, 1557.292666666665, 1482
   .3606666666667, 1378.9676666666667, 1388.6633333333333,
   1473.6846666666668, 1595.418333333333, 1606.079, 1600.
   349999999997, 1520.256, 1403.7700000000000, 1393.
   322333333335, 1460.688666666667, 1570.6616666666669, 1584
   .3266666666666, 1585.212333333336, 1513.2750000000003,
   1400.707, 1393.9533333333336, 1474.0683333333334, 1585.
   0293333333332, 1598.897333333333, 1581.552, 1499.
   954666666668, 1377.7676666666666, 1366.709333333335, 1435
   .1323333333332, 1545.161666666666, 1545.916, 1527.
   429333333333, 1439.005666666667, 1322.116, 1323.
   8936666666666, 1406.1360000000002, 1529.488333333335, 1536
   .903333333334, 1524.610999999999, 1446.7763333333332,
   1340.5346666666667, 1319.756, 1385.718333333333, 1494.
   5206666666666, 1526.204666666665, 1527.336, 1458.046, 1351
   .527333333332, 1348.629666666667, 1412.199333333332,
   1520.679, 1519.43, 1506.534999999999, 1431.3796666666667,
   1338.033999999999, 1344.535666666667, 1409.1016666666667
     1508.9793333333334, 1516.3346666666666, 1510.
   087333333332, 1430.9796666666664, 1312.0396666666666, 1295
   .6776666666667, 1358.885333333334, 1462.7483333333332,
   1466.0076666666666, 1457.4776666666667, 1377.277666666668
    , 1264.1866666666667, 1249.5973333333334, 1317.
   625666666668, 1424.846333333332, 1443.670333333333, 1441
   .6416666666667, 1364.763, 1248.9536666666665, 1246.
   8613333333333, 1323.171333333335, 1435.762, 1439.
   1166666666666, 1425.210333333332, 1341.264, 1231.
   681666666666, 1236.077999999997, 1310.5253333333333, 1417
   .727333333335, 1420.1850000000002, 1319.883333333334,
   1206.652, 1063.763, 1027.329, 1115.1773333333333, 1242.
   370999999999, 1360.078666666668, 1354.646333333334, 1282
   .8086666666668, 1176.2076666666665, 1180.40266666666666,
   1249.3570000000002, 1359.9930000000002, 1372.936333333333
   , 1373.8576666666668, 1304.8303333333333, 1185.
```

Page 1 of 241

```
#1- Using the Python program to load the 'opsd_germany_daily-data.csv'. You need
user to input the order
# of moving average. If m is odd, then then the software must ask a user to enter
m=1,2 will not be
order. In the latter case
use numpy and pandas
Enter the order of moving average: 1
Sorry!! order 1,2 will not be accepted
Enter the order of moving average: 3
Result of 3-MA is: [1297.41266666666666, 1426.7570000000003, 1458.960333333334,
1396.385999999997, 1475.8386666666665, 1578.341666666665, 1576.5256666666664,
1581.552, 1499.9546666666668, 1377.7676666666666, 1366.709333333335,
1526.204666666665, 1527.336, 1458.046, 1351.527333333332, 1348.629666666667, 1412.19933333332, 1520.679, 1519.43, 1506.534999999999, 1431.3796666666667,
1317.625666666668, 1424.846333333333, 1443.670333333333, 1441.6416666666667,
1242.370999999999, 1360.0786666666668, 1354.646333333334, 1282.808666666668, 1176.2076666666665, 1180.4026666666666, 1249.3570000000002, 1359.9930000000002,
```

```
1224.4396666666669, 1342.1370000000002, 1347.37700000000002, 1336.3026666666667,
1079.122333333335, 1202.604, 1320.2420000000002, 1322.0493333333334, 1251.619,
1165.1506666666667, 1164.22166666666666, 1235.263333333333, 1345.690333333332,
1288.8586666666667, 1284.9086666666667, 1219.258333333333, 1116.552333333333, 1113.837333333332, 1178.3410000000001, 1280.374, 1291.0383333333334,
1211.0476666666666, 1297.5896666666665, 1397.760666666668, 1398.682666666668,
1387.580333333333, 1301.345, 1182.64, 1187.367333333334, 1274.277, 1391.289,
1414.454333333331, 1400.839, 1319.4523333333334, 1204.9646666666667, 1172.789333333334, 1153.924, 1259.713333333334, 1296.55566666666664,
1270.205, 1360.82266666666667, 1478.12, 1485.101333333333, 1475.7946666666667,
1408.7076666666665, 1430.709333333332, 1437.2096666666669, 1418.9766666666667,
1575.564999999998, 1495.289, 1370.814, 1361.4070000000002, 1420.2356666666667,
1522.704, 1509.65666666666668, 1464.9606666666666, 1367.658999999999, 1242.156,
1135.878333333333, 1082.623, 1144.448, 1225.067, 1291.308333333334,
```

```
1446.107666666668, 1437.086666666668, 1367.04399999999, 1254.689333333333,
1247.937, 1319.5686666666668, 1425.492, 1439.9306666666664, 1327.144,
1287.315333333333, 1225.8256666666666, 1133.946666666667, 1111.921, 1163.82,
1257.809, 1375.8063333333332, 1381.048666666666, 1377.0163333333333,
1376.0990000000002, 1380.246333333335, 1319.45, 1202.926333333336,
1189.0106666666668, 1243.451666666668, 1352.769666666668, 1359.690333333335,
1359.5176666666666, 1285.767, 1176.070333333333, 1180.8466666666666,
1215.0046666666667, 1332.0633333333333, 1334.825, 1313.0240000000001,
1239.3043333333333, 1144.759, 1175.365333333334, 1263.92066666666666,
1174.8220000000001, 1176.034333333333, 1259.9063333333334, 1392.388,
```

```
1326.3666666666666, 1443.4056666666668, 1447.6703333333335, 1433.0476666666666,
1399.466333333333, 1281.911, 1265.238, 1346.5736666666667, 1473.2646666666667,
1463.694333333331, 1381.264333333333, 1387.215999999997, 1443.751,
1454.03999999997, 1440.67, 1412.882666666666, 1333.889000000001,
1273.38566666666666, 1200.747999999999, 1112.466333333335, 1114.234666666665,
1230.122999999999, 1360.47, 1372.087000000002, 1310.63333333333332,
```

```
1301.6716666666669, 1264.253, 1183.2986666666666, 1145.743333333333, 1186.97,
1374.6026666666667, 1394.04966666666666, 1385.832999999999, 1296.759, 1175.741,
1383.958333333333, 1302.3496666666667, 1183.3476666666666, 1186.7079999999999,
1334.046333333335, 1451.5130000000001, 1459.159333333333, 1436.8686666666665,
1408.186333333333, 1514.896333333334, 1504.6580000000001, 1492.8456666666666,
1171.892333333335, 1112.119, 1099.501, 1119.6546666666666, 1156.67,
1213.713333333334, 1267.985999999999, 1382.310666666665, 1422.81766666666666,
```

```
1283.663999999998, 1260.0783333333331, 1333.89199999998, 1444.7946666666667,
1249.5113333333331, 1298.46, 1378.1906666666666, 1414.0363333333335,
1195.606, 1093.425, 1085.356, 1146.162333333334, 1256.7093333333355,
1272.617333333334, 1262.4006666666667, 1188.241, 1083.845, 1094.467666666667, 1181.157, 1293.681333333333, 1288.2716666666665, 1153.606, 1040.104333333332,
901.7226666666666, 972.467, 1107.884666666668, 1242.7233333333334,
1252.3010000000002, 1188.578, 1085.853666666666, 1091.278333333334,
1249.1116666666667, 1249.943, 1240.419999999998, 1163.653666666668,
1141.1766666666665, 1125.914, 1190.054666666667, 1299.4226666666666,
```

```
1130.7696666666668, 1224.451333333334, 1346.906666666665, 1366.0133333333333,
1456.192, 1439.3526666666667, 1362.01766666666666, 1245.7043333333334, 1253.1193333333333, 1328.155, 1456.718, 1468.9170000000001, 1464.8300000000002,
1495.761666666665, 1435.152, 1348.718666666666, 1335.042666666667,
1333.014000000001, 1402.2630000000001, 1502.124333333334, 1497.9530000000002,
1460.518, 1417.2516666666668, 1321.001666666668, 1203.4596666666666,
1326.872, 1186.982, 1183.41366666666666, 1262.649999999999, 1379.311333333333,
1281.785333333333, 1233.1136666666666, 1159.38033333333, 1161.277333333332,
```

```
1273.339, 1225.5443333333335, 1131.1076666666665, 1084.6180000000002,
1361.596333333333, 1284.96633333333, 1182.204, 1171.6719999999998,
1497.009666666668, 1416.074000000003, 1315.928333333335, 1307.948, 1382.64466666668, 1472.128333333333, 1475.7030000000002, 1462.1090000000002,
1567.857333333334, 1490.582, 1387.7576666666664, 1410.104333333333,
1202.4516666666666, 1336.4006666666667, 1464.469333333335, 1452.89266666666666,
1395.987333333335, 1520.960333333332, 1531.7456666666667, 1518.0503333333334,
```

```
1378.6846666666668, 1463.0106666666663, 1586.875, 1593.1403333333335,
1436.310333333331, 1448.422333333332, 1364.7196666666669, 1240.113, 1232.4580000000003, 1317.481666666668, 1432.438, 1434.7603333333334,
1287.586999999998, 1178.303666666667, 1040.395333333333, 1018.468,
1173.184333333334, 1259.995333333333, 1385.60666666666, 1387.447333333333,
1226.1323333333332, 1340.471333333332, 1347.158999999999, 1332.7526666666665,
1324.594666666669, 1344.0776666666668, 1354.161, 1290.63, 1188.400333333335,
1194.90466666666666, 1270.3136666666667, 1376.5870000000002, 1379.069666666667,
1379.101333333333, 1302.865333333332, 1190.0206666666666, 1186.0496666666666,
1372.736999999999, 1380.051999999997, 1370.216333333333, 1288.39566666666666,
```

```
1399.373999999998, 1389.0406666666665, 1311.996666666667, 1190.9503333333332,
1236.127333333331, 1185.0606666666665, 1191.2096666666666, 1310.497, 1374.046,
1538.769666666667, 1521.4843333333336, 1438.2236666666668, 1323.419999999999,
1168.126333333334, 1075.206333333333, 1098.4446666666668, 1184.151333333335, 1317.154, 1352.8463333333332, 1338.33633333334, 1281.187, 1190.095333333334,
1482.264666666667, 1491.97, 1480.3076666666668, 1399.5486666666666, 1288.101,
1286.5020000000002, 1363.041333333333, 1483.9156666666668, 1520.3280000000002,
1407.648333333333, 1401.2150000000001, 1458.869, 1529.4633333333334, 1510.174, 1473.0043333333333, 1393.015, 1260.785666666669, 1248.4576666666665,
1314.612333333333, 1429.9710000000002, 1438.4806666666666, 1418.236333333333, 1311.4620000000002, 1197.908999999999, 1218.674, 1310.3496666666667,
1380.824333333332, 1402.2533333333333, 1393.564333333335, 1311.11566666666666,
1194.685333333333, 1180.7373333333333, 1254.564, 1361.214, 1383.6026666666667,
1023.7706666666667, 1002.1550000000001, 1102.5776666666666, 1259.7886666666666,
1202.0326666666667, 1118.4073333333333, 1055.263, 1159.84366666666666, 1239.397,
1363.047333333332, 1292.75766666666666, 1178.091333333335, 1178.4356666666665,
1267.769, 1150.9956666666667, 1149.270333333333, 1244.466333333333,
1095.425, 1147.0806666666667, 1241.720666666668, 1364.7263333333333,
```

```
1222.544, 1099.9743333333333, 1121.356333333333, 1203.6766666666667,
1102.015333333333, 1088.24766666666666, 1167.5376666666666, 1301.7073333333335,
1156.444333333336, 1245.384666666668, 1359.047333333334, 1360.7810000000002, 1340.5810000000001, 1256.6786666666667, 1143.552333333333, 1157.494333333333,
1354.314333333335, 1261.330333333336, 1148.599333333333, 1150.638333333333,
1394.665, 1381.859666666665, 1308.694, 1201.742333333334, 1211.70833333333, 1302.502, 1431.836333333334, 1431.23333333336, 1418.8976666666665,
1243.5433333333333, 1334.999, 1462.8426666666667, 1484.2250000000001,
1323.901, 1231.8786666666667, 1248.6850000000002, 1343.230333333333,
```

```
1418.4276666666667, 1312.3773333333334, 1192.719333333335, 1190.367333333334,
1215.7556666666667, 1192.5283333333332, 1258.9636666666665, 1379.866, 1403.527, 1400.632666666666, 1330.405, 1229.725999999999, 1215.649, 1267.9146666666666,
1213.010333333334, 1320.017, 1230.0196666666668, 1180.081, 1102.723, 1092.7226666666666, 1129.544333333333, 1202.18333333332, 1302.30533333333, 1305.865999999998, 1296.6603333333335, 1222.833666666667, 1100.9273333333333,
1200.7686666666666, 1099.889666666667, 1098.5863333333334, 1172.805,
1108.637, 1185.415, 1289.5766666666668, 1301.7820000000002, 1302.332,
1146.0773333333334, 1138.9643333333333, 1207.294333333333, 1305.733,
1101.420333333333, 1116.139333333333, 1182.106666666666, 1269.425, 1268.527333333334, 1272.0846666666666, 1213.2810000000002, 1122.644, 1128.825,
1277.139333333335, 1165.960333333332, 1160.590333333333, 1238.5569999999998,
1275.7846666666667, 1238.114333333334, 1153.48933333334, 1125.1136666666669,
```

```
1307.7146666666665, 1202.2196666666666, 1191.6033333333332, 1268.738,
1448.225999999999, 1454.4170000000001, 1438.9316666666666, 1365.54866666666666,
1037.013333333333, 1049.5316666666665, 1081.865333333332, 1095.830333333333,
1495.95, 1550.6140000000003, 1558.334999999998, 1491.682, 1382.1599999999999,
1649.413333333333, 1638.918, 1549.408000000001, 1420.7036666666667,
1264.561666666667, 1347.9440000000002, 1465.1363333333331, 1471.2266666666667,
1239.4453333333333, 1239.438333333333, 1327.692, 1445.1506666666667,
```

```
1423.7856666666667, 1344.9926666666668, 1232.846333333333, 1229.6756666666665,
1466.791333333333, 1383.5559999999998, 1258.447333333333, 1257.042666666667,
1518.506333333335, 1514.990333333334, 1425.301333333336, 1310.1236666666666, 1310.450333333332, 1401.9996666666666, 1526.726666666667, 1539.671999999998,
1409.257999999998, 1502.53333333335, 1631.736, 1642.348333333334,
1496.3190000000002, 1621.7396666666666, 1637.584, 1619.665, 1517.802, 1389.514,
1587.135333333334, 1507.508333333334, 1395.870000000001, 1405.71966666666666,
1407.241666666668, 1414.353, 1507.256, 1633.0846666666666, 1645.976999999999,
```

```
1504.485333333333, 1498.992999999997, 1434.162666666664, 1330.1760000000002,
1333.604333333334, 1404.838666666665, 1508.8310000000001, 1515.8136666666667,
1418.613666666669, 1427.466999999999, 1413.982666666668, 1326.6783333333333,
1408.559333333334, 1409.439, 1330.360333333333, 1219.581333333333, 1220.526999999998, 1305.5103333333334, 1414.8473333333332, 1351.8583333333333,
1220.5310000000002, 1225.212333333334, 1308.875333333332, 1424.1986666666664,
1426.828666666665, 1414.370333333335, 1329.308666666668, 1211.56766666666666,
1384.765, 1301.1719999999998, 1192.4053333333334, 1192.07966666666668, 1269.209,
1222.7563333333333, 1319.2753333333333, 1434.481333333332, 1436.764, 1418.063, 1337.1376666666665, 1231.655, 1235.526, 1317.4506666666666, 1423.60566666666666,
1455.686, 1477.402, 1464.4986666666666, 1375.262333333333, 1258.582333333335,
```

```
1546.764333333333, 1553.0510000000002, 1536.526333333332, 1454.201333333334, 1347.527666666668, 1347.609000000002, 1421.311, 1526.993333333333,
1251.045333333332, 1241.747333333332, 1321.8436666666666, 1439.2549999999999,
1247.475999999999, 1331.7596666666666, 1439.9396666666669, 1441.393,
1427.095333333334, 1350.4696666666666, 1248.096666666666, 1263.5490000000002,
1193.381999999998, 1230.003333333333, 1318.521333333334, 1430.033333333333,
1296.468333333332, 1411.054333333335, 1415.1346666666668, 1399.215999999997,
```

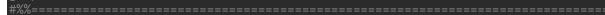
```
1342.5149999999999, 1337.1466666666665, 1267.819, 1167.9843333333333,
1396.055666666664, 1408.276333333333, 1406.348333333334, 1329.0766666666668,
1410.8806666666667, 1293.8120000000001, 1294.9996666666666, 1379.0323333333333,
1588.059333333334, 1511.488, 1399.854, 1391.363999999998, 1460.313333333335,
1561.9450000000004, 1673.221666666667, 1671.900000000003, 1654.287333333334,
1617.6483333333333, 1602.220333333334, 1507.6956666666665, 1385.9486666666664,
1385.924, 1478.318333333334, 1607.1636666666666, 1628.242999999997,
```

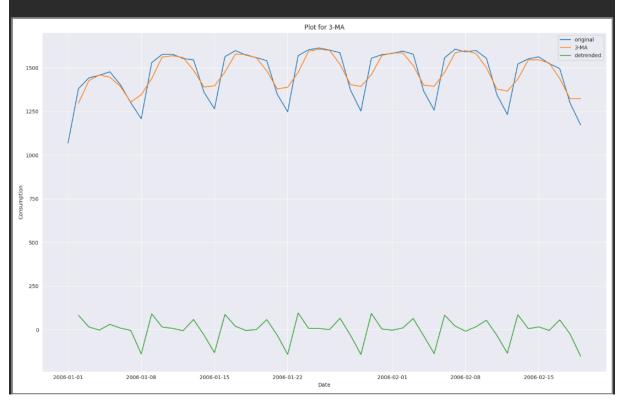
```
1342.604, 1228.52066666666666, 1230.292666666665, 1310.4476666666667,
1210.05566666666666, 1293.336999999998, 1411.938, 1426.175, 1424.314666666664,
1303.942, 1240.5973333333332, 1154.823333333335, 1149.8866666666666, 1211.325,
1320.899333333335, 1190.420333333332, 1069.019666666664, 1137.2626666666665, 1265.665666666668, 1385.734333333333, 1390.161333333335, 1306.2946666666667,
1420.102999999998, 1408.846333333332, 1318.600333333333, 1206.4853333333333,
1252.820333333333, 1358.486666666667, 1365.155, 1350.1096666666665, 1272.547, 1174.027333333332, 1175.6973333333333, 1253.121333333333, 1354.5613333333333,
1221.2696666666668, 1334.256, 1358.3906666666664, 1368.08466666666666,
1195.170333333332, 1196.9246666666666, 1281.707999999999, 1404.172333333332,
1217.0739833333334, 1203.7101, 1321.1433100000002, 1372.09699, 1467.7411266666668,
1397.50576, 1287.4313366666665, 1285.68704, 1366.10945, 1479.34781,
1490.8367766666668, 1482.0275833333335, 1396.9954866666667, 1272.37951666666666,
```

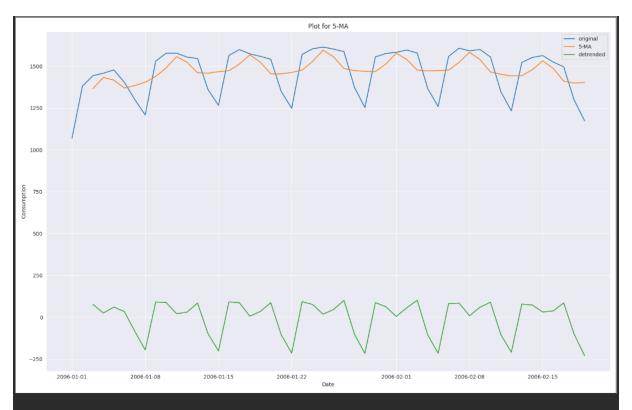
```
1419.481943333334, 1313.8943466666667, 1269.0179, 1242.29263, 1286.38148, 1329.15513, 1418.187343333333, 1410.0507, 1316.60407, 1328.10051, 1415.0553766666665, 1532.9309633333332, 1550.250689999998, 1555.176646666668, 1488.1624633333333, 1382.7904766666668, 1375.5965733333333, 1452.6013466666666, 1559.44092, 1571.4073133333331, 1562.99542, 1498.95715666666666, 1396.7137666666667, 1401.16385, 1470.0228633333334, 1566.9313166666668, 1572.337866666667, 1556.2148733333333, 1481.0234966666667, 1379.12699, 1403.810666666665, 1497.8855866666665, 1607.1809766666665, 1612.132683333333, 1604.65977, 1533.94967, 1438.359393333334, 1447.6979633333333, 1533.94967, 1438.359393333334, 1447.6979633333333, 1526.5439066666665, 1637.8087533333335, 1634.5944333333332, 1449.2045233333336, 1526.5439066666665, 1637.8087533333335, 1645.2704633333333, 1621.7933, 1532.1175999999998, 1419.00940333333332, 1419.59567666666666, 1487.4640833333333, 1574.9529266666668, 1555.51335, 1501.0985233333333, 1405.2602433333334, 1279.0553233333333, 1175.0705099999998, 1127.71917, 1168.44704, 1231.30724, 1286.2974733333333, 1270.133493333333, 1205.883793333333]
```

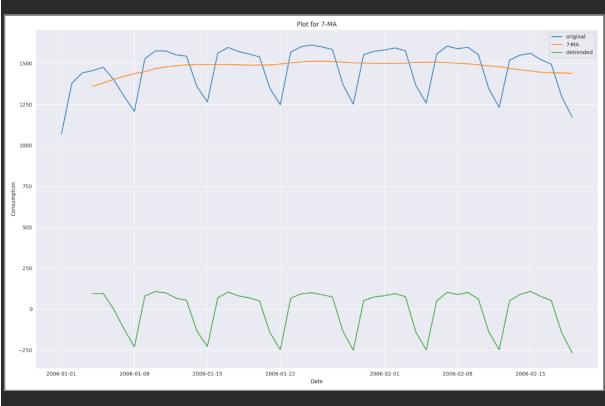
the detrended data on

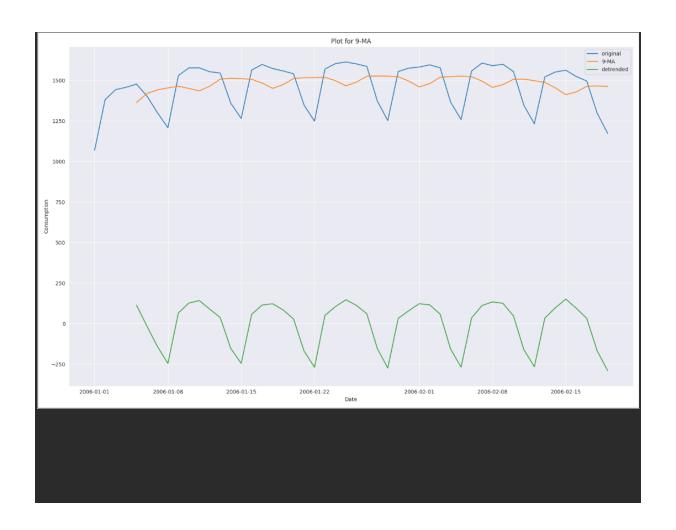
the same graph. Add an appropriate title, x-label, y-label, and legend to the graph.







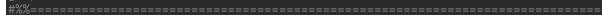


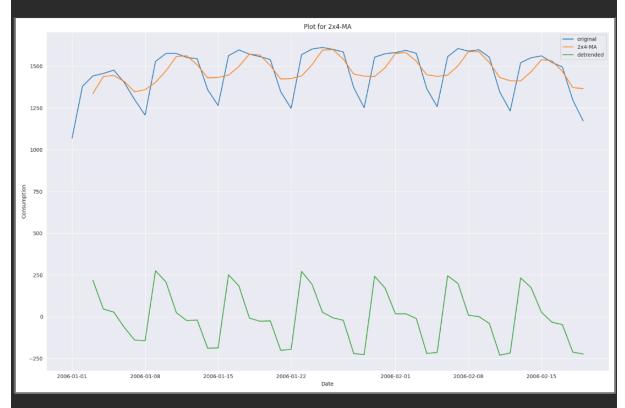


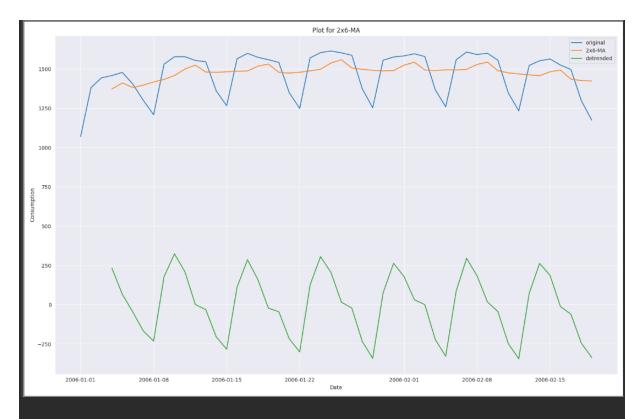
#3. Using the function developed in the previous step plot the estimated cycletrend versus the original

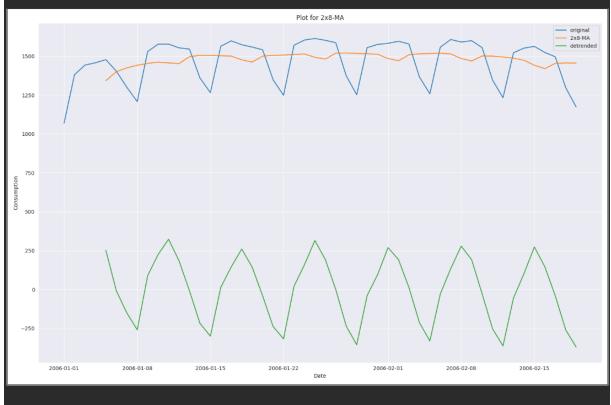
dataset (plot only the first 100 samples) for 3-MA, 5-MA, 7-MA, and 9-MA. Plot the detrended data on

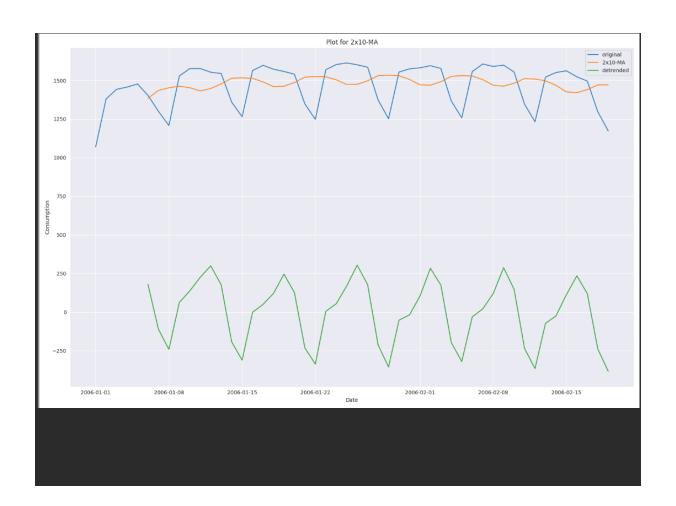
the same graph. Add an appropriate title, x-label, y-label, and legend to the graph.











Original DataSet

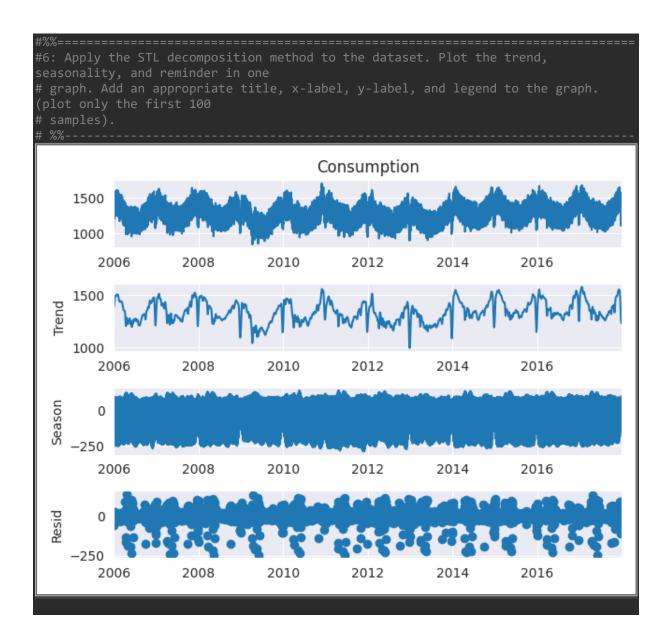
the p-value is less than 5%, we strong evidence against the null hypothesis, so we reject the null hypothesis.

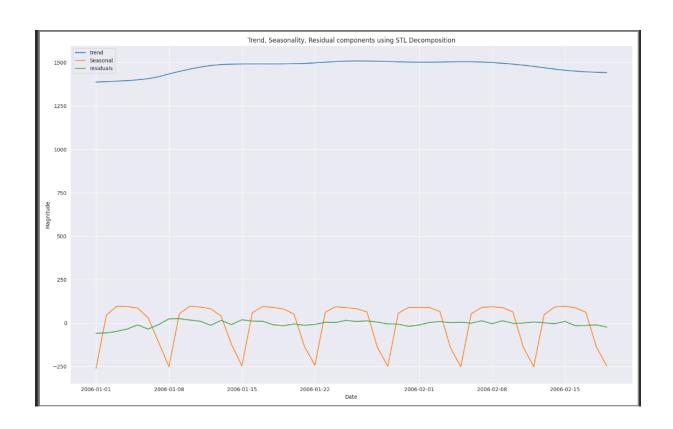
As per the test results above, the p-value is 0.0 which is < 0.05 therefore we reject the null in favour of the alternative hypothesis that the time series is stationary.

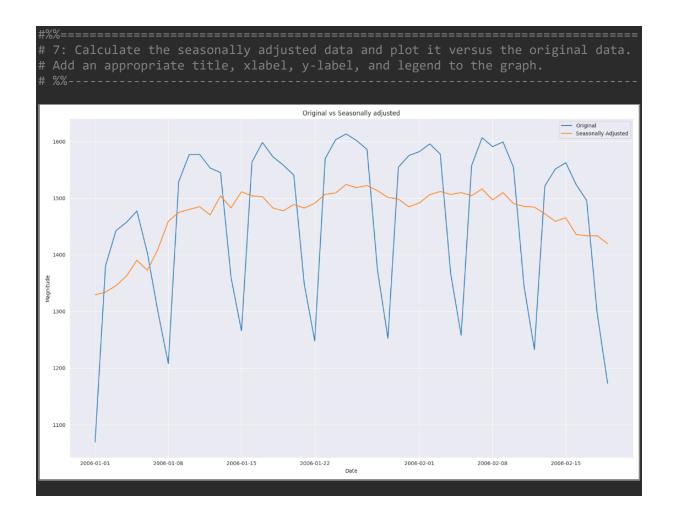
Detrended Dataset

the p-value is less than 5%, we strong evidence against the null hypothesis, so, we reject the null hypothesis.

As per the test results above, the p-value is 0.01 which is <0.05 therefore we reject the null in favour of the alternative hypothesis that the time series is stationary.







#%%====================================
8: Calculate the strength of trend using the following equation and display the following message on the # console:
The strength of trend for the minimum temperature data set is # %%
Strength of trend for opsd germany dataset is 0.8855098504197492
#%%====================================
9: Calculate the strength of seasonality using the following equation and display the following message # on the console:
The strength of seasonality for the minimum temperature data set is # %%
Strength of seasonality for opsd germany dataset is 0.9209108297499213
#%====================================
Running the example plots the observed, trend and seasonal time series. We can see that there is strong trend and seasonality information extracted from the series does seem reasonable.

CONCLUSION

In this lab, we discovered time series decomposition and how to decompose time series data with Python. Specifically, we learned the following:

- a. The structure of decomposing time series into level, trend, seasonality, and noise.
- b. How to automatically decompose a time series dataset with Python.
- c. How to decompose an additive or multiplicative model and plot the results.

Running the example plots the observed, trend and seasonal time series. We can see that the trend and seasonality information extracted from the series does seem reasonable.

CHALLENGE

Calculations was little tricky to understand in the beginning, after lot of clarifications it provided clarity.

APPENDIX

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import STL
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
sns.set style('darkgrid')
df = pd.read_csv('opsd_germany_daily-data.csv', index_col='Date',
Year = pd.date_range(start='2006-01-01', end='2017-12-31', freq='D')
y = df['Consumption'].astype(float)
def plot_ma(y, k, trend, detrend, ma_order, folding_order):
    plt.figure(figsize=(16,10))
    plt.plot(np.array(Year[:50]), np.array(y[:50]), label='original')
    if ma order%2 != 0:
        plt.plot(np.array(Year[k:50]), np.array(trend[:50-k]), label='{}-
MA'.format(ma_order))
        plt.title('Plot for {}-MA'.format(ma_order))
        plt.plot(np.array(Year[k:50]), np.array(trend[:50 - k]), label='{}x{}-
MA'.format(folding_order, ma_order))
        plt.title('Plot for {}x{}-MA'.format(folding_order, ma_order))
    plt.plot(np.array(Year[k:50]), np.array(detrend[:50-k]), label='detrended')
    plt.xlabel('Date')
    plt.ylabel('Consumption')
    plt.legend()
    plt.show()
def ADF Cal(x):
    result = adfuller(x)
    print('ADF Statistic: %f' %result[0])
    print('p-value: %f' %result[1])
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))
m = int(input('Enter the order of moving average: '))
    print('Sorry!! order 1,2 will not be accepted')
    m = int(input('Enter the order of moving average: '))
if m % 2 == 0:
    n = int(input('Enter the order of moving average: '))
    while n < 2 or n % 2 != 0 or n >= m:
```

```
print('Sorry!! folding order should not be odd or not less than 2')
        n = int(input('Enter the order of moving average: '))
def cal_moving_average(col, ma_order, folding_order):
    ma = []
    k = int(np.ceil((ma_order - 1) / 2))
    for t in range(0, len(col) - ma order + 1):
        temp = np.sum(y[t:ma_order + t])
        ma.append(temp / ma_order)
    if folding_order > len(ma):
    elif folding_order != 0:
        k1 = int(np.ceil((ma_order - 1) / 2) + ((folding_order - 1) / 2))
        folding_ma = []
        for t in range(0, len(ma) - folding_order + 1):
            a = np.sum(y[t:folding_order + t])
            folding_ma.append(a / folding_order)
        print("Result of {}x{}-MA is: {}".format(folding order, ma order,
folding_ma))
        detrended = np.subtract(list(y.iloc[k1:-k1]), folding ma)
        plot_ma(y, k1, ma, detrended, ma_order, folding_order)
        return detrended, folding ma
        print("Result of {}-MA is: {}".format(ma_order, ma))
        detrended = np.subtract(list(y.iloc[k:-k]), ma)
        plot_ma(y, k, ma, detrended, ma_order, folding_order)
        return detrended, ma
if m % 2 != 0:
    res ma = cal moving average(col=y, ma order=m, folding order=0)
    res ma = cal moving average(col=y, ma order=m, folding order=n)
detrended_3, ma_3 = cal_moving_average(col=y, ma_order=3, folding_order=0)
detrended_5, ma_5 = cal_moving_average(col=y, ma_order=5, folding_order=0)
detrended_7, ma_7 = cal_moving_average(col=y, ma_order=7, folding_order=0)
detrended 9, ma 9 = cal moving average(col=y, ma order=9, folding order=0)
detrended_2x4, ma_2x4 = cal_moving_average(col=y, ma_order=4, folding_order=2)
detrended_2x6, ma_2x6 = cal_moving_average(col=y, ma_order=6, folding_order=2)
detrended_2x8, ma_2x8 = cal_moving_average(col=y, ma_order=8, folding_order=2)
detrended_2x10, ma_2x10 = cal_moving_average(col=y, ma_order=10, folding_order=2)
ADF_Cal(y)
ADF Cal(detrended 3)
STL = STL(y)
res = STL.fit()
fig = res.plot()
plt.show()
T = res.trend
S = res.seasonal
```

```
R = res.resid
plt.figure(figsize=(16,10))
plt.plot(T[:50], label='trend')
plt.plot(S[:50], label='Seasonal')
plt.plot(R[:50], label='residuals')
plt.xlabel('Date')
plt.ylabel('Magnitude')
plt.title('Trend, Seasonality, Residual components using STL Decomposition')
plt.legend()
plt.show()
adjusted seasonal = y-S
plt.figure(figsize=(16,10))
plt.plot(y[:50], label='Original')
plt.plot(adjusted_seasonal[:50], label='Seasonally Adjusted')
plt.xlabel('Date')
plt.ylabel('Magnitude')
plt.title('Original vs Seasonally adjusted')
plt.legend()
plt.show()
# Measuring strength of trend and seasonality
F = np.maximum(0,1-np.var(np.array(R))/np.var(np.array(T+R)))
print('Strength of trend for opsd germany dataset is', F)
FS = np.maximum(0, 1-np.var(np.array(R))/np.var(np.array(S+R)))
print('Strength of seasonality for opsd germany dataset is', FS)
```

	REFER	ENCES		
https://otexts.com/fpp2/#				
		40		