University of New Brunswick

Time Series Analysis
(EE 6563)

Assignment #1

Professor:
Erik Scheme
Electrical and Computer
Engineering

Author:
Saeed Kazemi
(3713280)

February 9, 2021

1. **Download the following datasets:**

   (a) **Minimum Daily Temperatures Dataset**:This dataset describes the minimum daily temperatures over 10 years (1981-1990) in the city Melbourne, Australia. The units are in degrees Celsius and there are 3650 observations. The source of the data is credited as the Australian Bureau of Meteorology (Source).

   (b) **Monthly Sunspot Dataset**:This dataset describes a monthly count of the number of observed sunspots for just over 230 years (1749-1983). The units are a count and there are 2,820 observations. The source of the dataset is credited to Andrews & Herzberg (1985) (Source).

2. **For each dataset, visualize the raw signals, identify any trends, seasonality, and/or other components, and try to remove them. Remember that you are not limited to the tools shown in the tutorial and should explore the various concepts discussed in class. You do not need to explain the theory behind the approaches, but you should provide justification for their use, and discussion of their results.**

   Figures 1 and 2 indicate the raw signal of both data sets. Likewise, figure 3 to 4 illustrate a short period in two datasets.

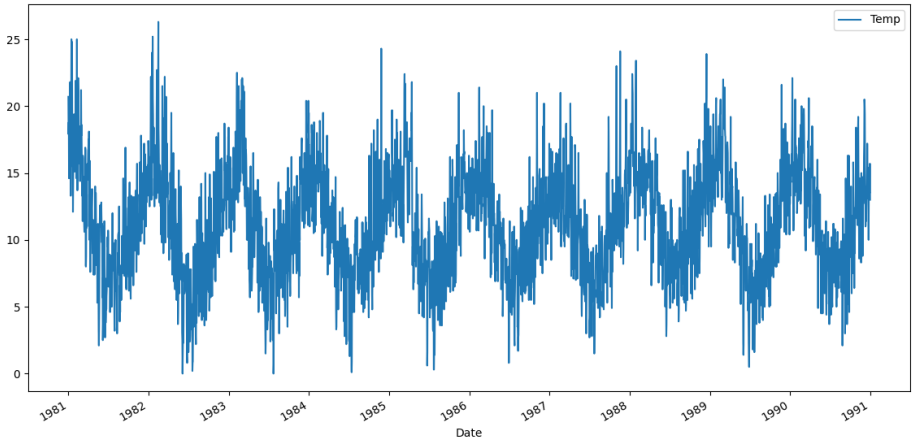

Figure 1: The raw signal of the first dataset.

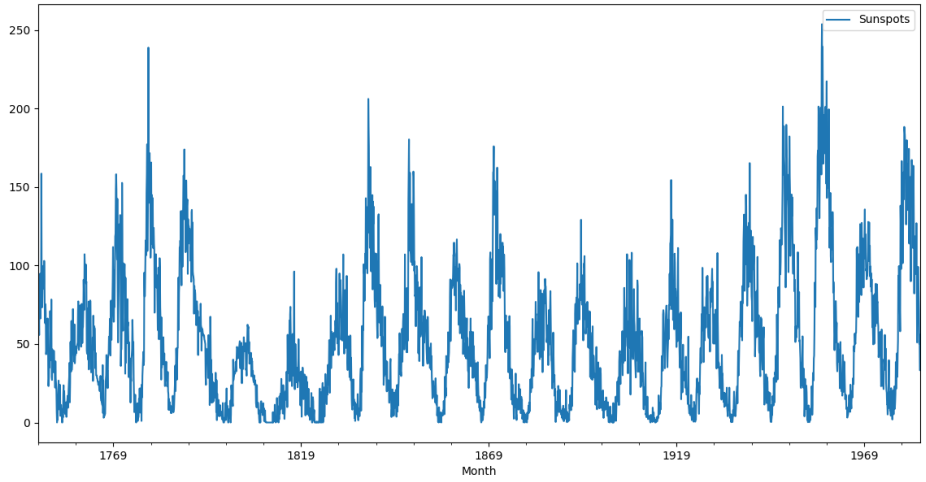

Figure 2: The raw signal of the second dataset.

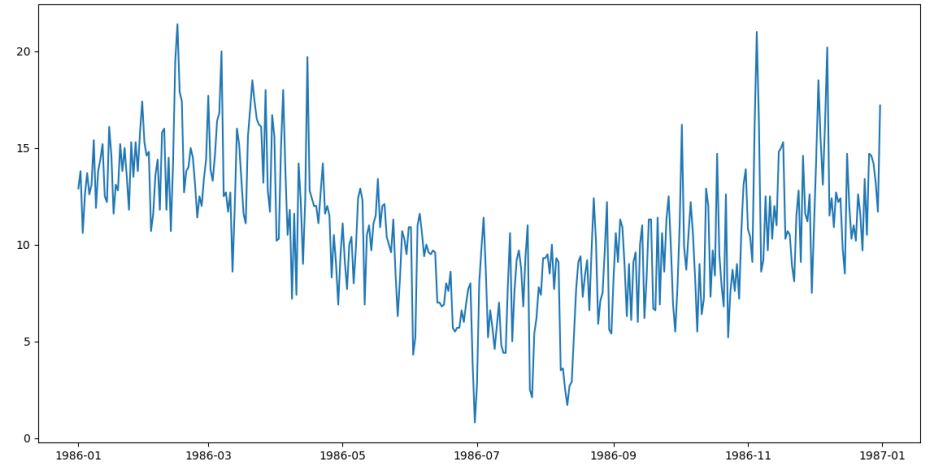

Figure 3: Visualizing a short period of the first dataset.

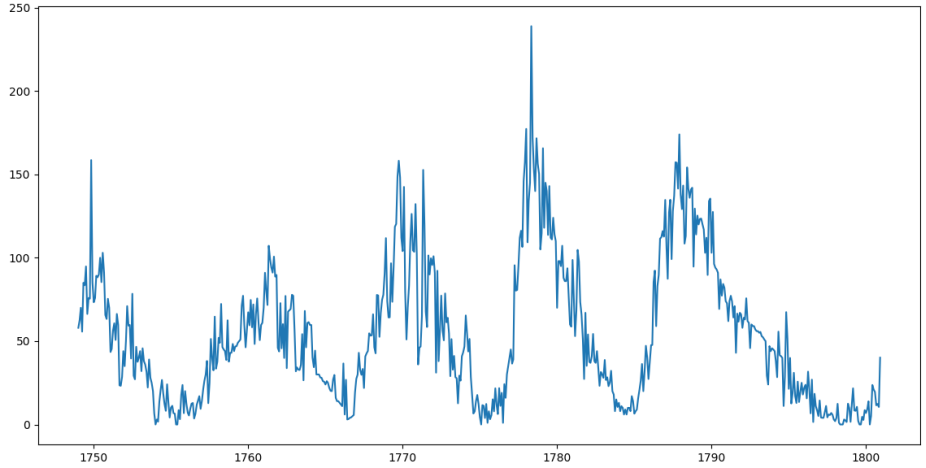

Figure 4: Visualizing a short period of the second dataset.

For intuition, Table 1 to 4 show the top raw of two datasets along with the descriptive statistics of two time-series datasets.

Table 1: The first five rows of the raw signal in the first dataset.

Date	Temp
1981-01-01	20.7
1981-01-02	17.9
1981-01-03	18.8
1981-01-04	14.6
1981-01-05	15.8

Table 2: The descriptive statistics of the first dataset.

	Temp
count	3650.000000
mean	11.177753
std	4.071837
min	0.000000
25%	8.300000
50%	11.000000
75%	14.000000
max	26.300000

Table 3: The first five rows of the raw signal in the second dataset.

Month	Sunspots
1749-01-01	58.0
1749-02-01	62.6
1749-03-01	70.0
1749-04-01	55.7
1749-05-01	85.0

Table 4: The descriptive statistics of the second dataset.

	Sunspots
count	2820.000000
mean	51.265957
std	43.448971
min	0.000000
25%	15.700000
50%	42.000000
75%	74.925000
max	253.800000

For decomposing the time series, the below methods used:

(a) *Seasonal_decompose (Figure 9 and 10)*

(b) *STL (Figure 11 and 12)*

(c) *Linear Regression method (Figure 13 and 14)*

(d) *Difference method (Figure 15 and 16)*

(e) *Fitting a polynomial (Figure 17 and 18)*

(f) *Moving Average window (Figure 19 and 20)*

Some mentioned methods used only for extracting only one type of component (trend or seasonality), while others like STL and Seasonal_decompose provided all three parts. Table 5 compares these methods together.

Also, there are two models for the reconstruction of time series, Additive and Multiplicative Model. In this assignment, the additive model was only used because the multiplicative model is not appropriate for zero and negative values.

Table 5: Comparing the implemented methods.

	Trend component	Seasonal component	Residual component
Seasonal_decompose	✓	✓	✓
STL	✓	✓	✓
Linear Regression	✓	-	-
Difference	✓	✓	✓
Fitting polynomial	✓	✓	✓
Moving average Function	✓	-	-

For finding the seasonal component, we need to find the period of the time series. For doing this, the resampling method was used to smooth the plots of two datasets. Figures 5 and 6 indicate the smooth version of the two datasets.

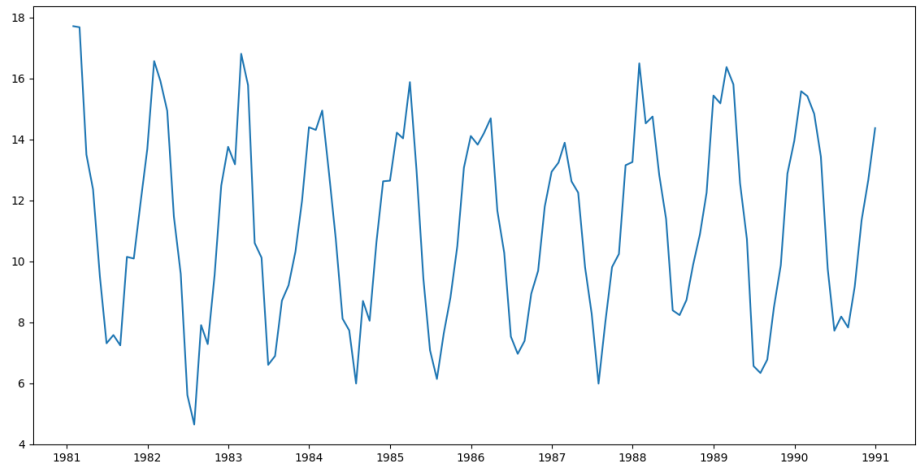

Figure 5: Resampling plot of the first dataset. The period of this dataset is 365 samples/cycle.

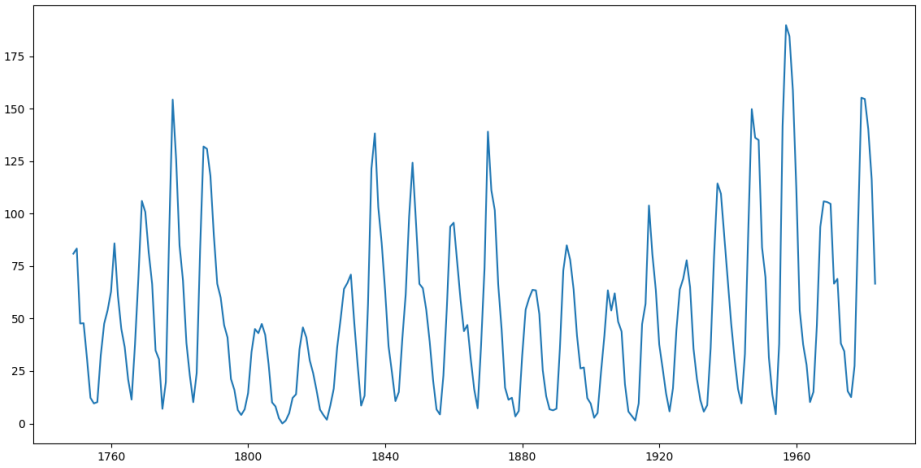

Figure 6: Resampling plot of the second dataset. The period of this dataset is 130 samples/cycle.

Furthermore, the period of the time-series data can calculate by AutoCorrelation function (ACF) (figures 7 and 8). This plot has an oscillation, indicative of a seasonal series and the peaks occurred at each period.

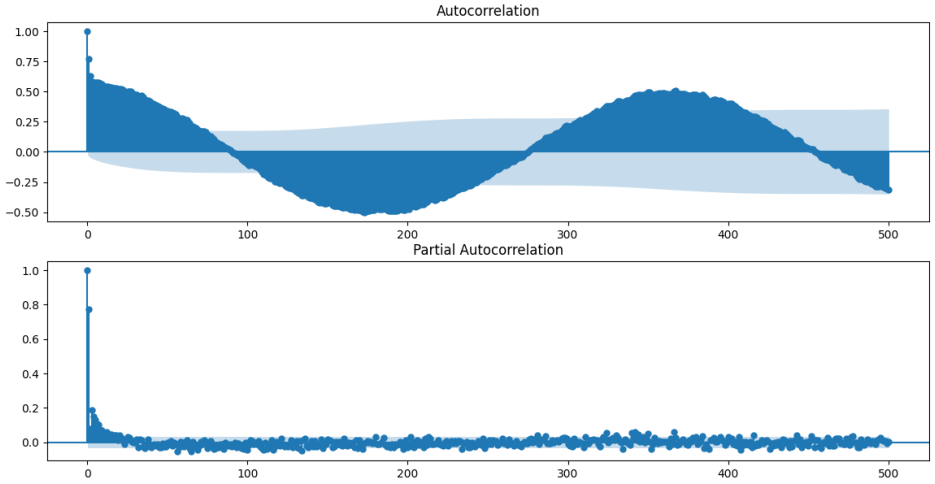

Figure 7: The ACF and Partial AutoCorrelation function (PACF) of the first dataset. The peaks occur at the lag of 365 that means the period is 365 samples/cycle.

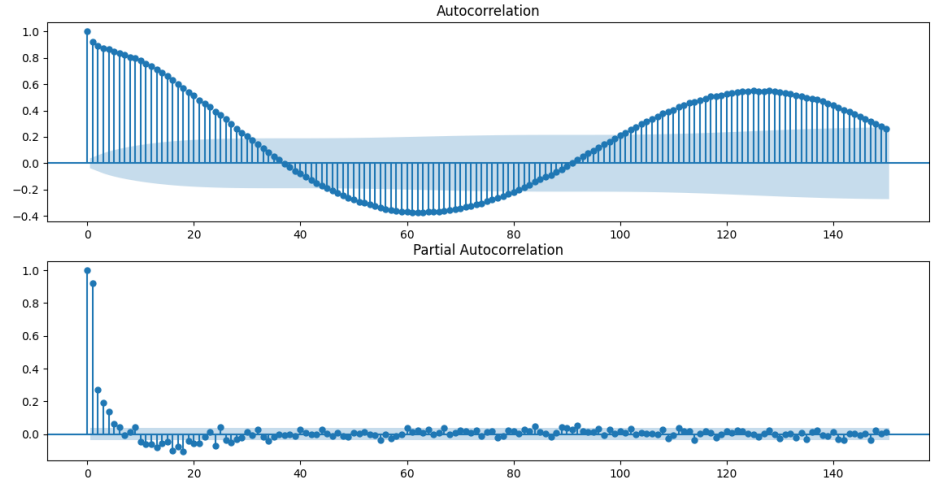

Figure 8: The ACF and PACF of the second dataset. The peaks occur at the lag of 130 that means the period is 130 samples/cycle.

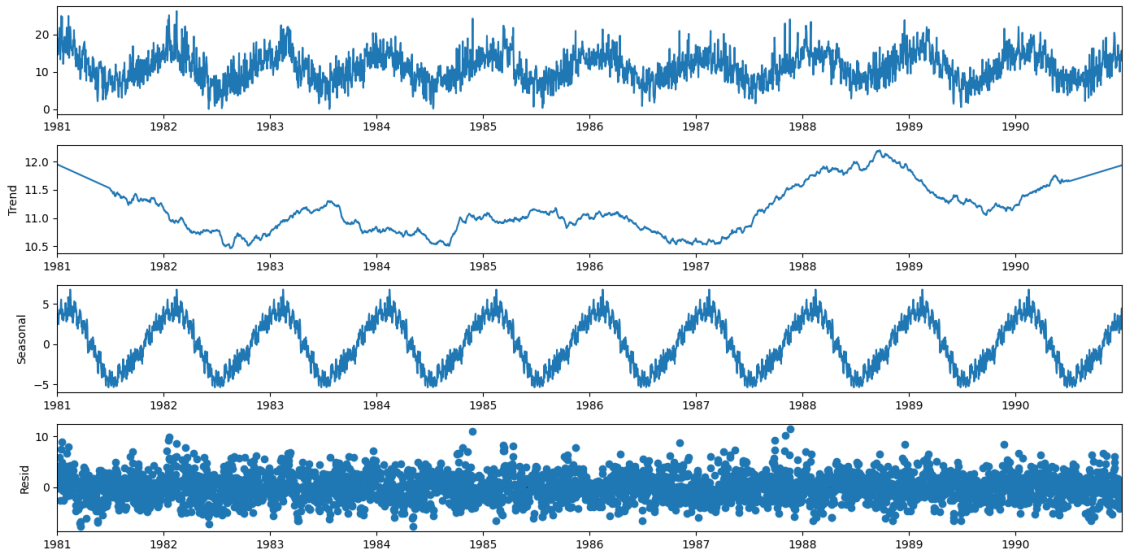

Figure 9: Decomposition of the first dataset by seasonal_decompose method

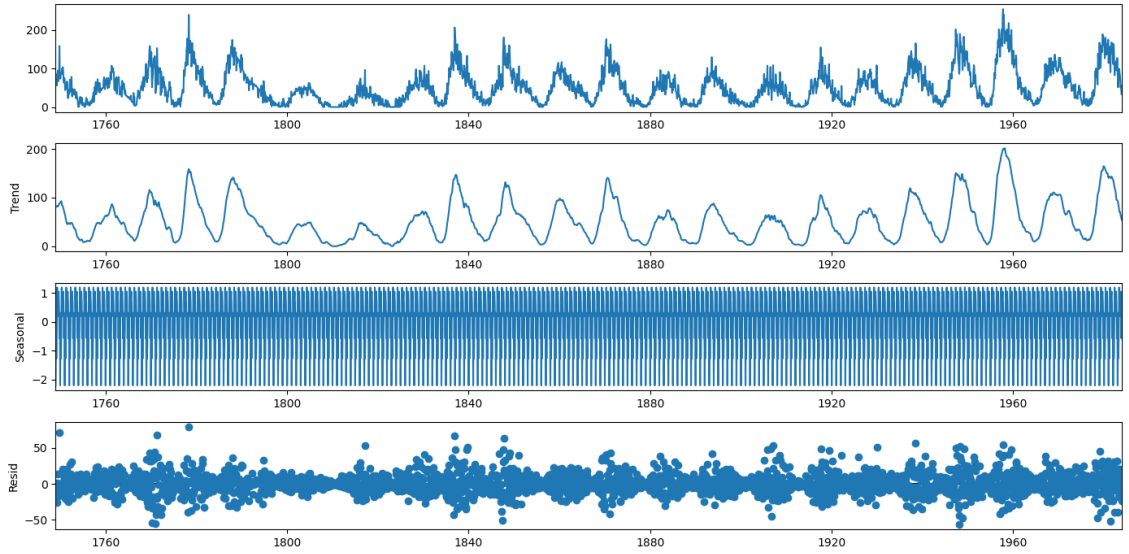

Figure 10: Decomposition of the second dataset by seasonal_decompose method

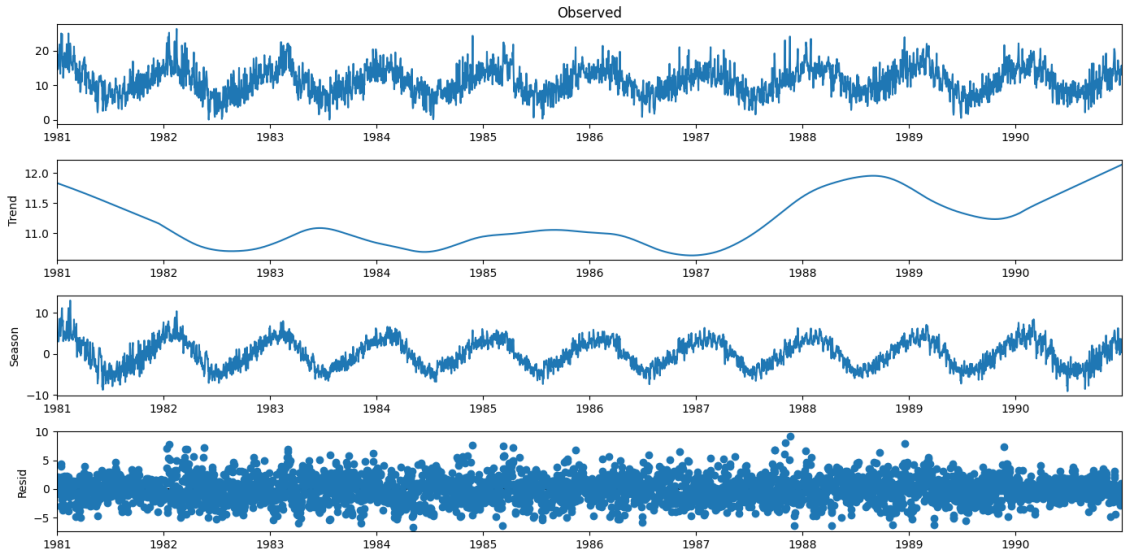

Figure 11: Decomposition of the first dataset by STL method

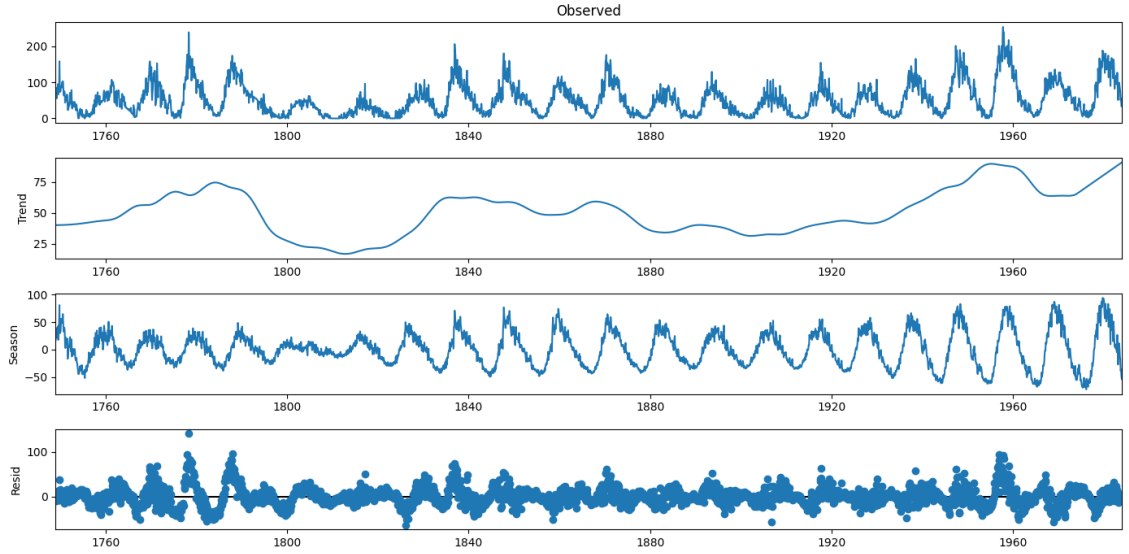

Figure 12: Decomposition of the second dataset by STL method

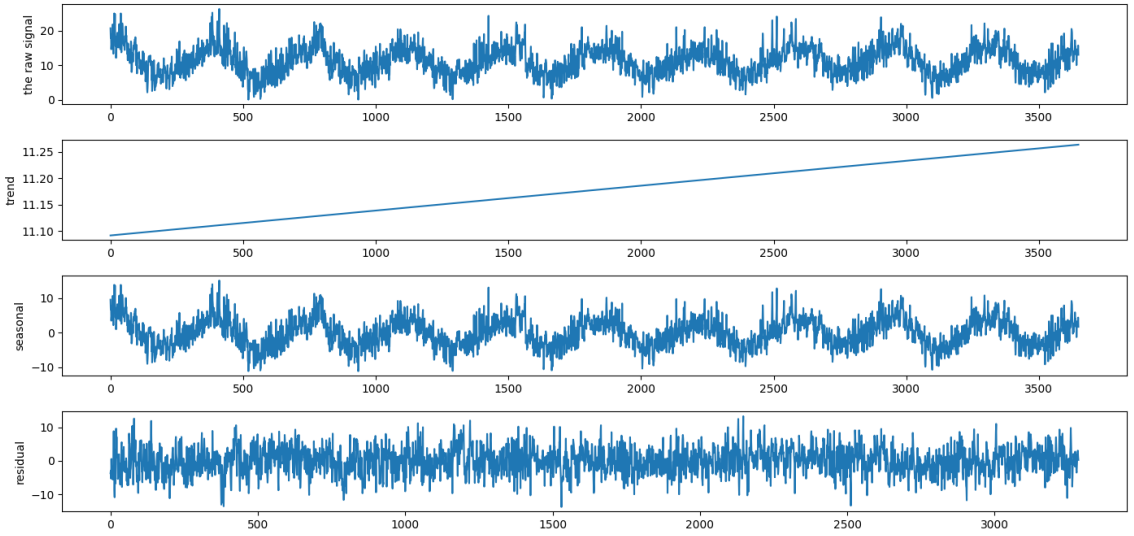

Figure 13: Decomposition of the first dataset by Linear Regression and difference method.

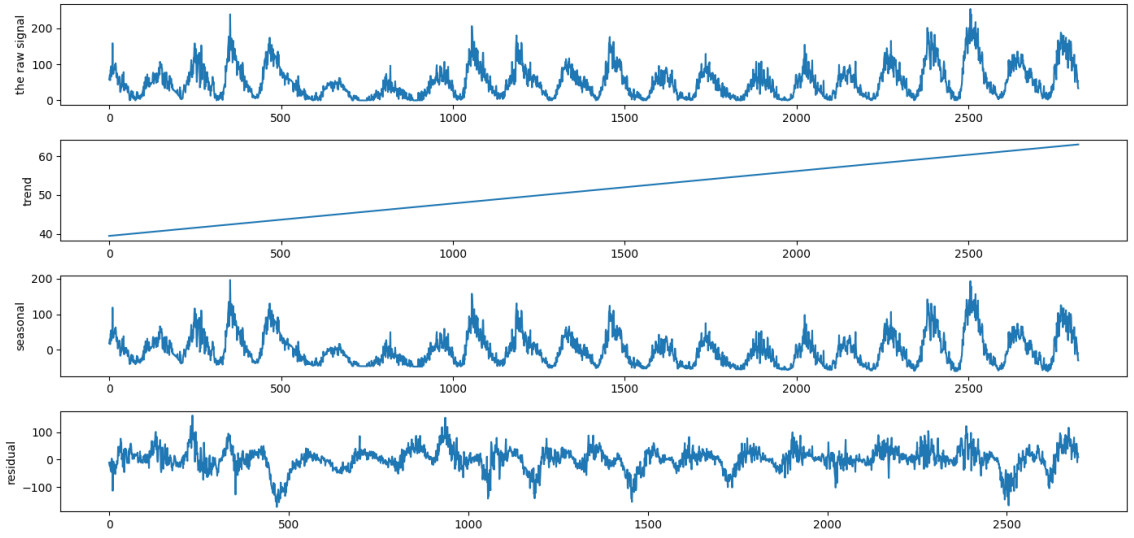

Figure 14: Decomposition of the second dataset by Linear Regression and difference method.

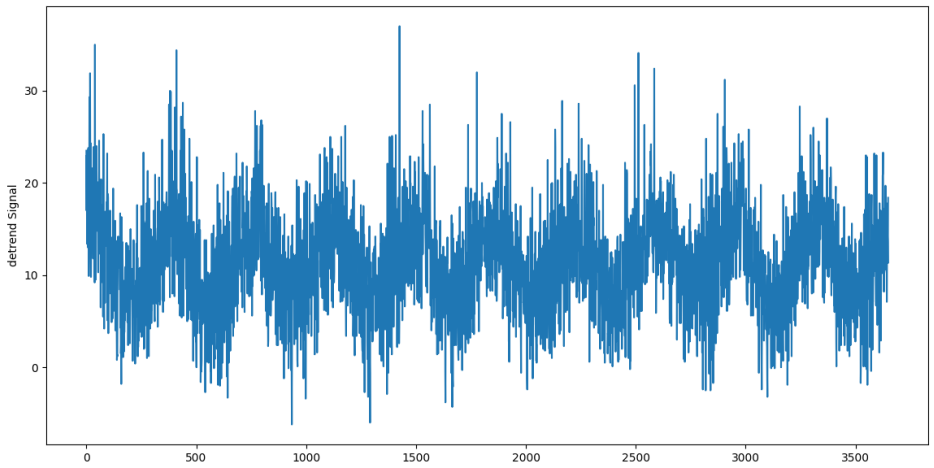

Figure 15: Detrending of the first dataset by the first-order differencing.

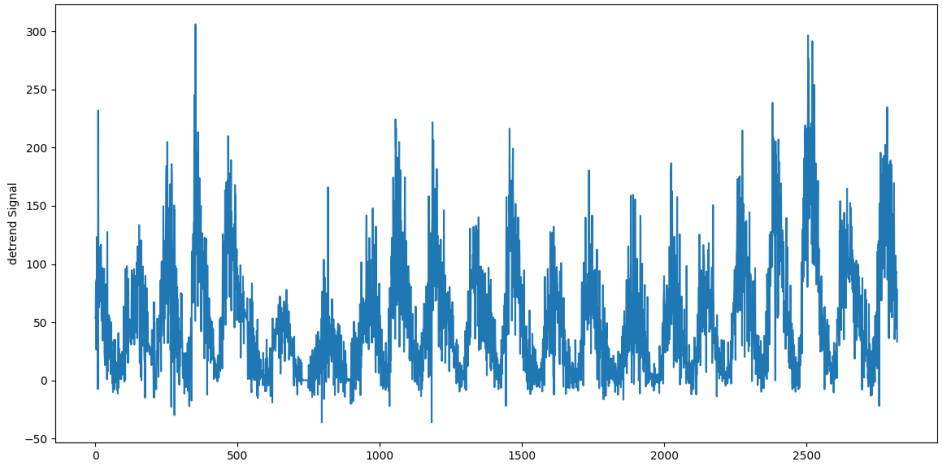

Figure 16: Detrending of the second dataset by the first-order differencing.

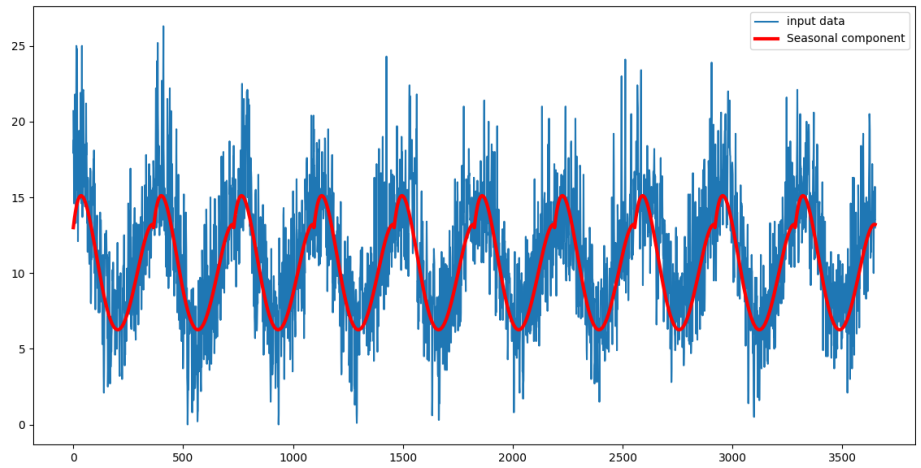

Figure 17: Seasonal component of the first dataset by fitting a polynomial (Degree of the polynomial is 4).

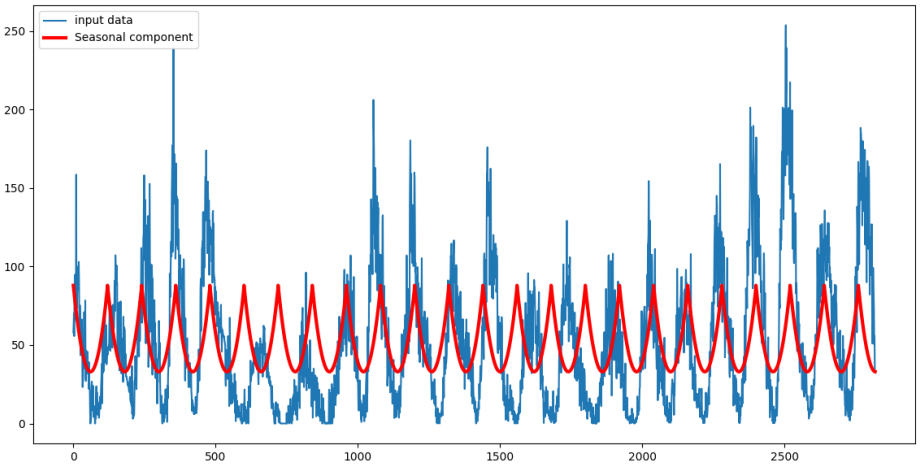

Figure 18: Seasonal component of the second dataset by fitting a polynomial (Degree of the polynomial is 2).

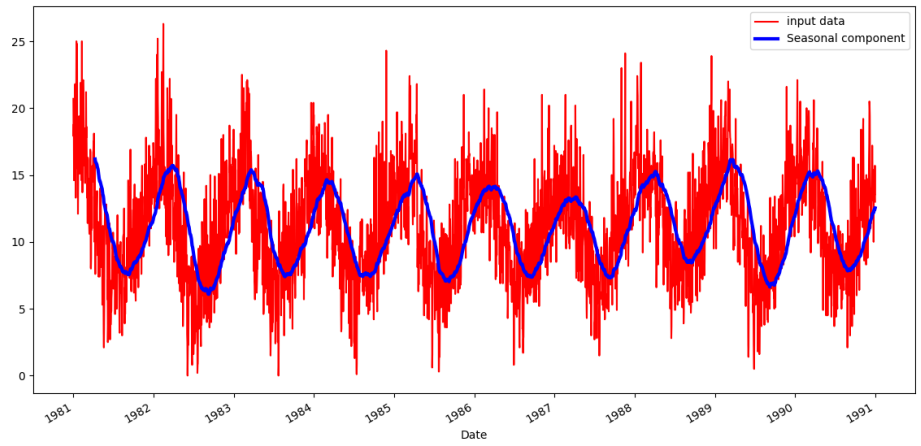

Figure 19: Seasonal component of the first dataset by Moving Average. The size of the window was set to 100.

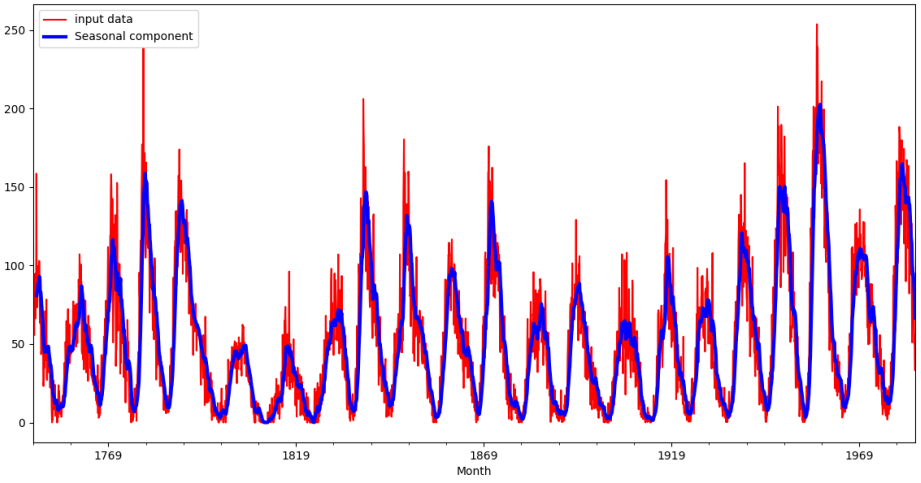

Figure 20: Seasonal component of the second dataset by Moving Average. The size of the window was set to 12.

3. **For each dataset, examine the stationarity of the residuals using the ACF and PACF functions, Lag Plots, and/or other approaches. Show your results and provide commentary about your observations.**

   Augmented Dickey-Fuller test (ADF test) shows that the time series is non-stationary or stationary. If the p-value is less than the significance level of 0.05 and the ADF test statistic is lower than one of the critical values, then the time series is stationary. For example, tables 6 and 7 shows the result of ADF test on a residual component of STL method in the two datasets.

Table 6: The result of the ADF test on the first dataset.

	0
ADF Statistic	-24.697687
p-value	0.000000
#Lags Used	3.000000
Number of Observations Used	3646.000000
Critical Value (1%)	-3.432145
Critical Value (5%)	-2.862333
Critical Value (10%)	-2.567192

Table 7: The result of the ADF test on the second dataset.

	0
ADF Statistic	-1.128610e+01
p-value	1.416821e-20
#Lags Used	2.700000e+01
Number of Observations Used	2.792000e+03
Critical Value (1%)	-3.432694e+00
Critical Value (5%)	-2.862576e+00
Critical Value (10%)	-2.567321e+00

Kwiatkowski-Phillips-Schmidt-Shin test (KPSS test) is another test for checking the stationarity of a time series. If the p-value is less than the significance level of 0.05, then the time series is not stationary. Table 8 and 9 show the result of this test on STL residual.

Table 8: The result of the KPSS test on the first dataset.

	0
KPSS Statistic	0.019544
p-value	0.100000
Lags Used	21.000000
Critical Value (10%)	0.347000
Critical Value (5%)	0.463000
Critical Value (2.5%)	0.574000
Critical Value (1%)	0.739000

Table 9: The result of the KPSS test on the second dataset.

	0
KPSS Statistic	0.007897
p-value	0.100000
Lags Used	29.000000
Critical Value (10%)	0.347000
Critical Value (5%)	0.463000
Critical Value (2.5%)	0.574000
Critical Value (1%)	0.739000

We can conclude that the series is stationary or not based on both KPSS test and ADF test [1]. Table 10 shows the possible outcomes of applying these two tests.

Table 10: The combination of the result of the KPSS test and ADF test.

KPSS test	KDF test	The combination result
non-stationary	non-stationary	The series is non-stationary.
stationary	non-stationary	Use detrending to make series stationary.
non-stationary	stationary	Use differencing to make series stationary.
stationary	stationary	The series is stationary.

ACF and PACF plots allow us to determine how correlated points are with each other. Figures 21 and 22 indicate these two plots for our datasets.

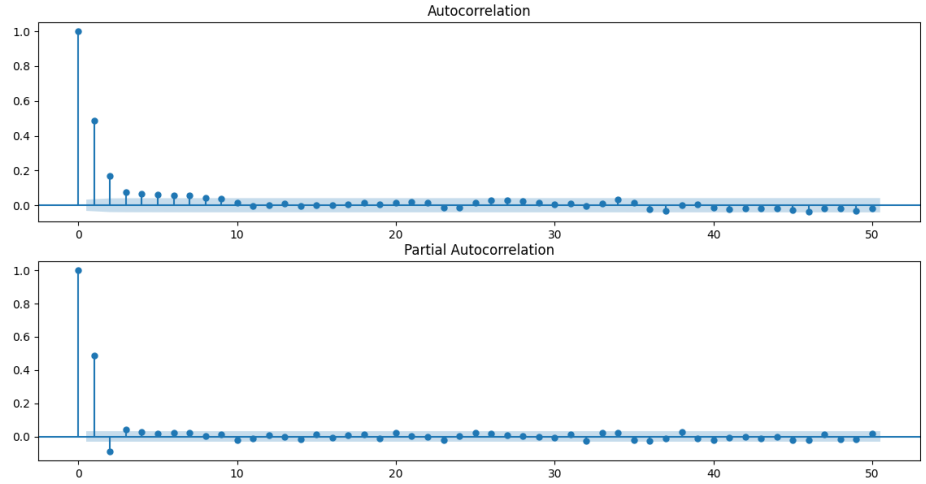

Figure 21: A plot of the ACF and PACF of the first dataset.

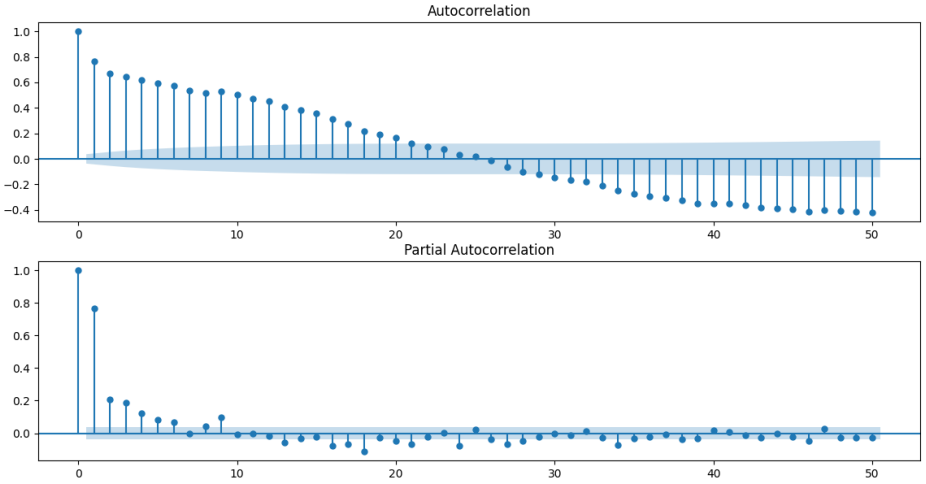

Figure 22: A plot of the ACF and PACF of the second dataset.

These two plots are used to find the q and p for the ARIMA model. For example, if ACF decays towards zero, and PACF have only q significant value then our time series is a AR(q) process.

The below figures (figures 23 and 24) show the lag plot of the two datasets for

different lags. As these plots illustrate, both datasets have diagonal shapes that indicate datasets are not a stationary time series.

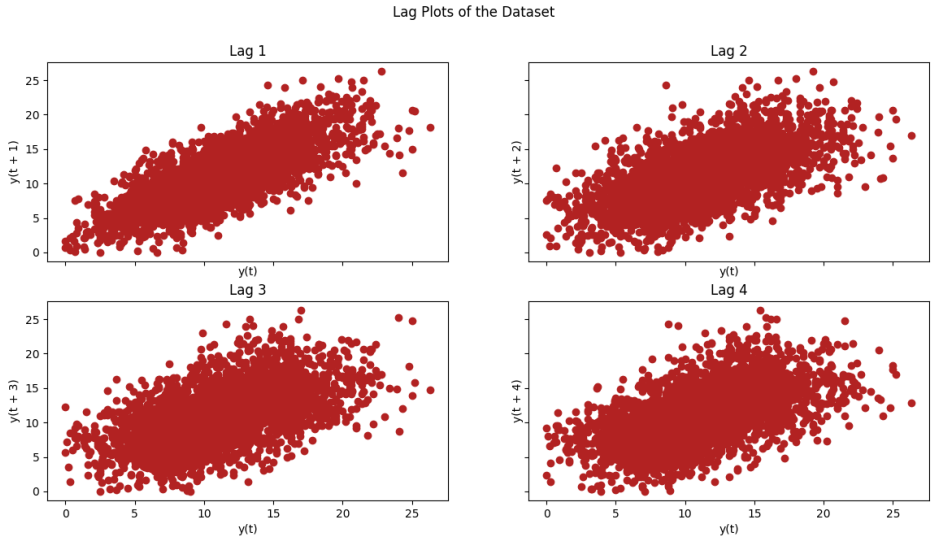

Figure 23: A different lag plot of the first dataset.

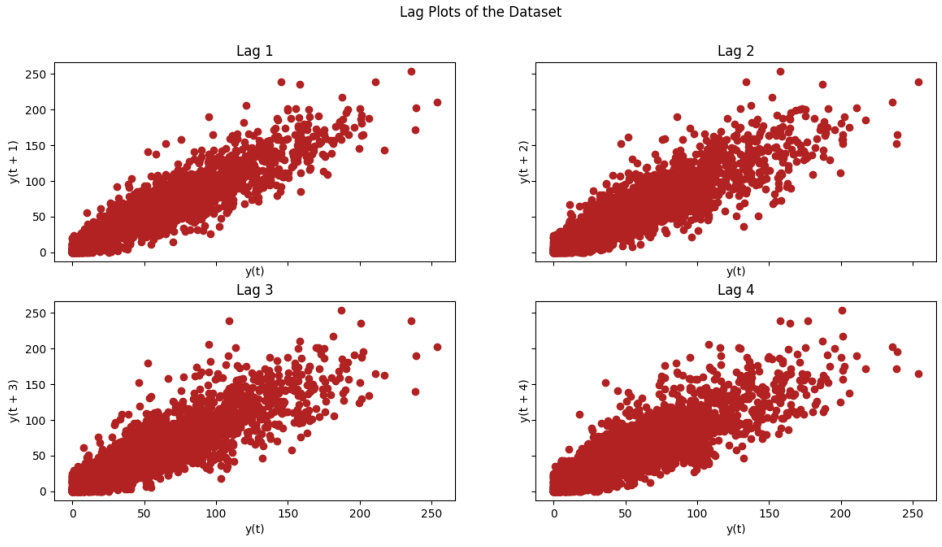

Figure 24: A different lag plot of the second dataset.

Figures 25 and 26 show the lag plot of the residual component for different lags. As figure 25 illustrates, the first dataset has round shapes for lags 2 to 4.

Therefore, the time series is stationary for lags 2 to 4 and a weak correlation on lag 1.

Figure 26 illustrates a clustered around the diagonal that turns to a round shape by increasing the lag. Hence, the dataset has a weak positive autocorrelation.

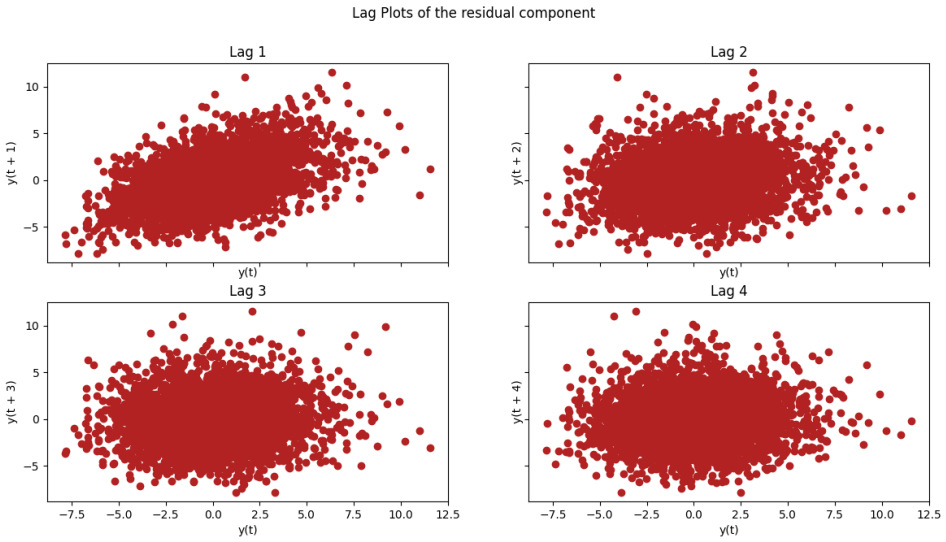

Figure 25: A different lag plot of residual component on the first dataset.

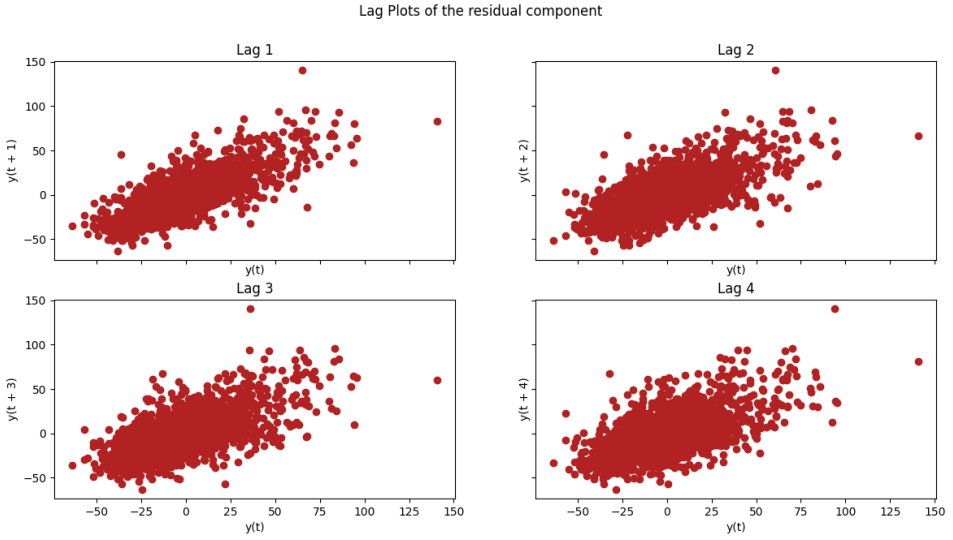

Figure 26: A different lag plot of residual component on the second dataset.

In order to turn the second dataset to a stationary dataset, we can use a first-order differencing. Tables 11 and 11 indicate the result of ADF test and KPSS test on the first-order differencing. These two tests show that the data is a stationary dataset.

Table 11: The result of the ADF test test on the 1^{st} order differensing in the second dataset.

	0
ADF Statistic	-8.647591e+00
p-value	5.219691e-14
#Lags Used	2.300000e+01
Number of Observations Used	2.795000e+03
Critical Value (1%)	-3.432692e+00
Critical Value (5%)	-2.862575e+00
Critical Value (10%)	-2.567321e+00

Table 12: The result of the KPSS test test on the 1^{st} order differensing in the second dataset.

	0
KPSS Statistic	0.007892
p-value	0.100000
Lags Used	54.000000
Critical Value (10%)	0.347000
Critical Value (5%)	0.463000
Critical Value (2.5%)	0.574000
Critical Value (1%)	0.739000

If we look at the ACF and PACF (figure 27), we can conclude that the data is probably random. Or in other words, there is no autocorrelation between samples. Likewise, figure 28 confirms this theory.

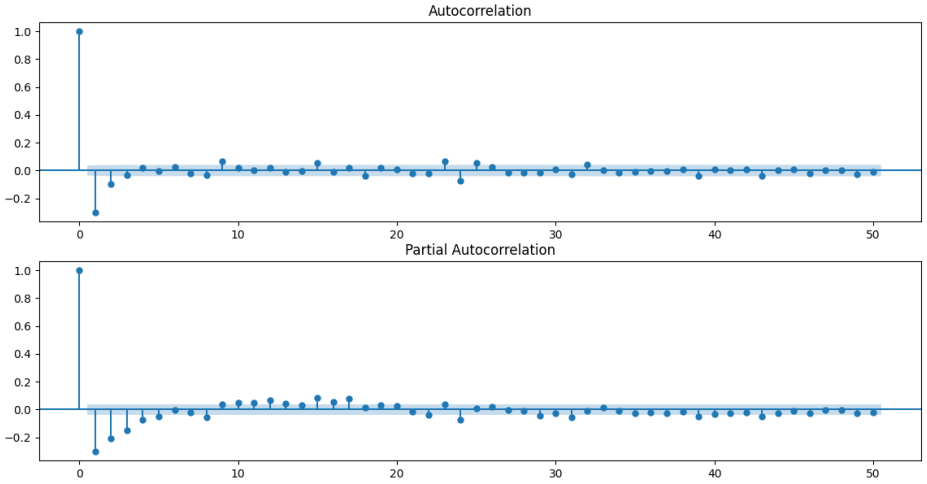

Figure 27: A plot of the ACF and PACF on the 1^{st} order differensing.

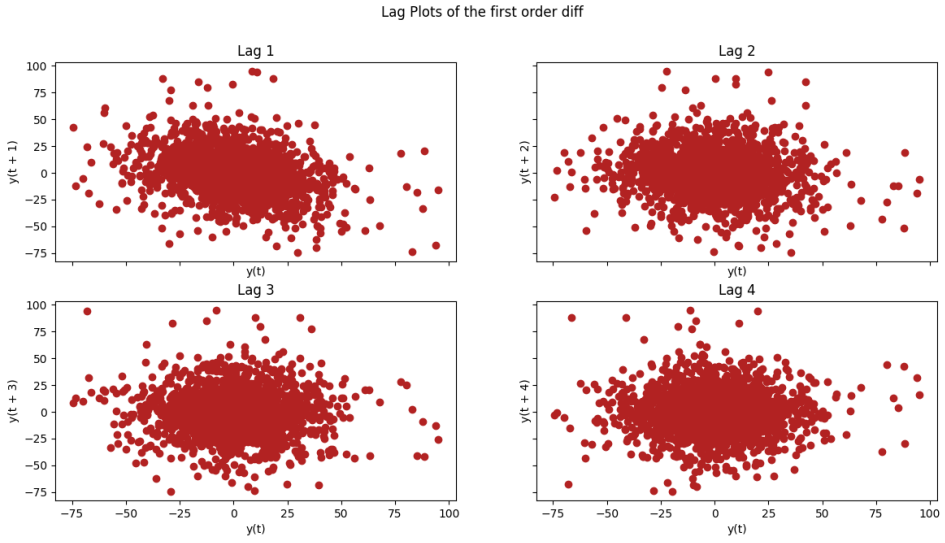

Figure 28: A different lag plot on the 1^{st} order differensing.

4. **Try modeling the residuals as an AR process. Use the tools at your disposal to decide on an appropriate order and analyse the results. What is the impact of selecting different orders on the remaining residuals?**

   For this part, we need to select the order of the autoregressive model (AR) term (p). Therefore, PACF and ACF were plotted at the first step (see figures 29 and 30).

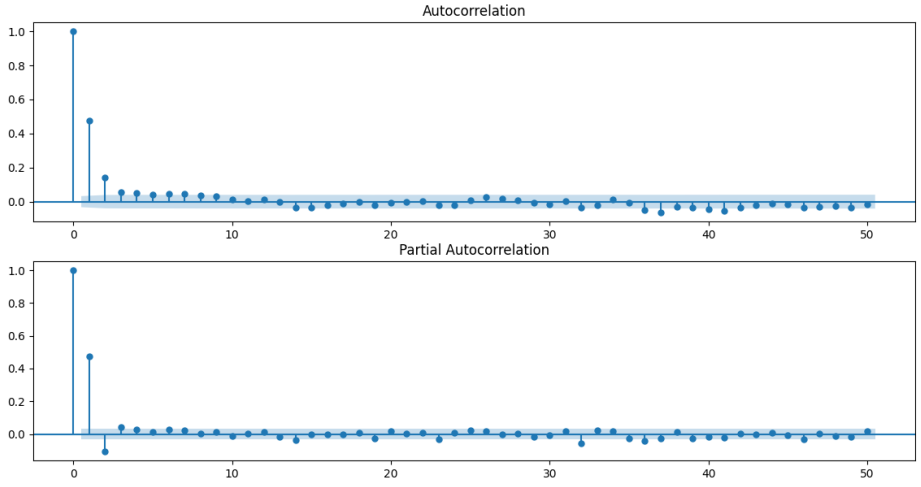

Figure 29: A plot of the PACF and ACF on the residual part of the first dataset (residual of the STL method).

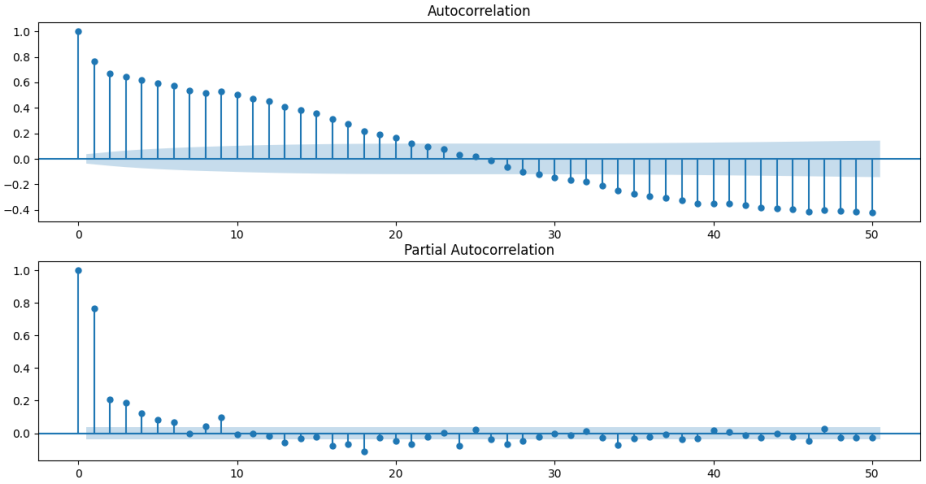

Figure 30: A plot of the PACF and ACF on the residual part of the second dataset (residual of the STL method).

Since ACF is decaying in both figures, it can conclude that these processes are an Auto-Regressive process. Also, based on PACF in the first plot (figure 29), the order of the AR model should be either 1 and 2 due to these two lags have a significant value. Likewise, for the second dataset, we should select p a value between 1 to 4 (see figure 30).

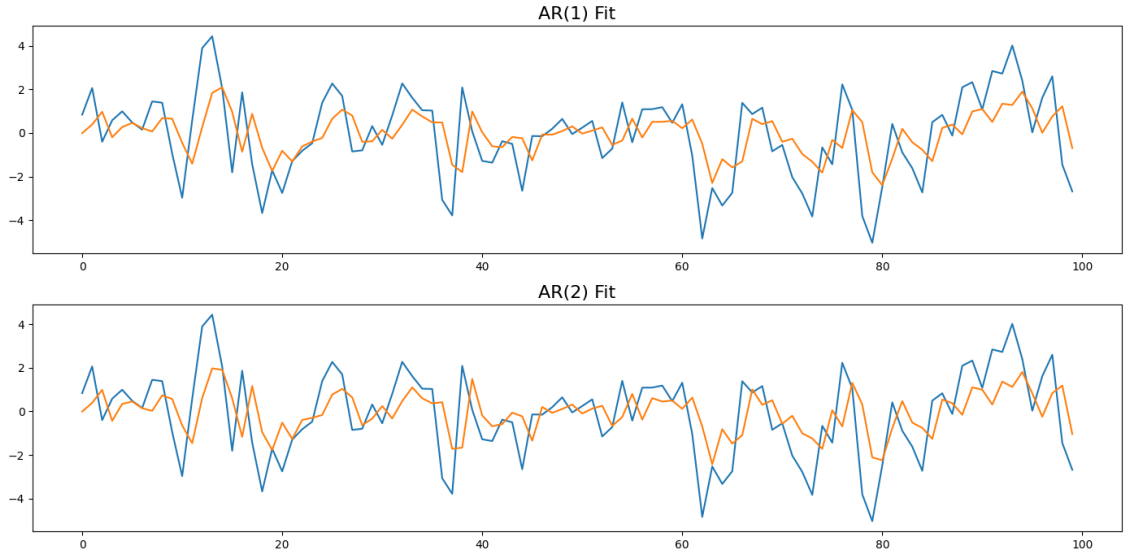

Figure 31: A part of residual signal (blue) and the prediction of AR (orange) for the first dataset.

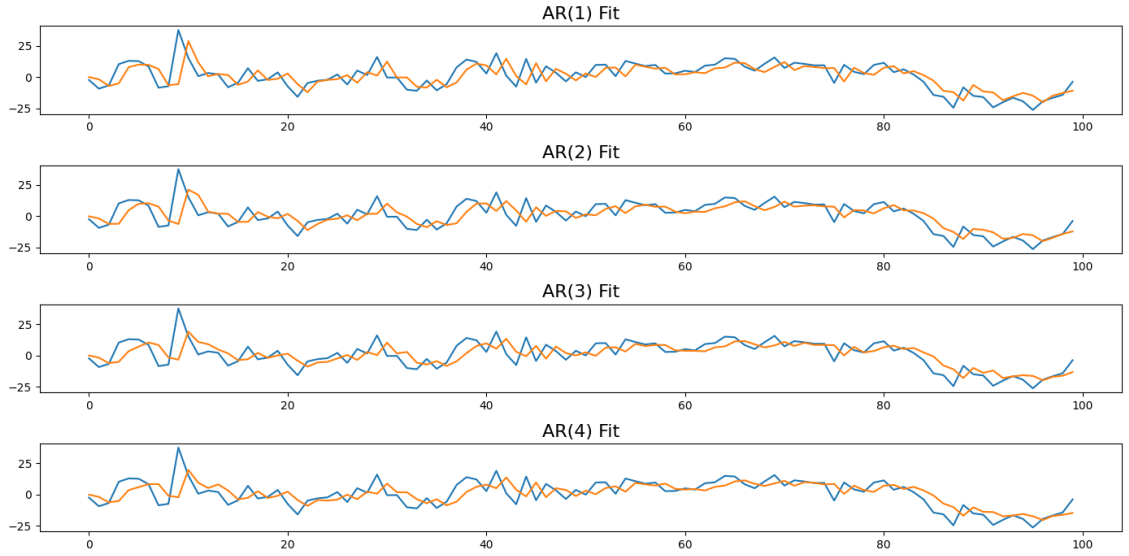

Figure 32: A part of residual signal (blue) and the prediction of AR (orange) for the second dataset.

Figures 31 and 32 indicate the output of our models on the datasets. As these two figures indicate, it is hard to select the best model based on the trained

signal. Therefore, we need some criteria to help us to find the best model. Tables 13 and 14 show these criteria on these models.

Table 13: Comparing the AR models in the first dataset.

	AR(1)	AR(2)
lag(s)	1	2
AIC	14995.448	14956.605
BIC	15014.043	14981.399
RMS error	1.149	1.157
Correlation	0.289	0.284
MPE	-1.078	-0.950
MAE	0.857	0.851

Table 14: Comparing the AR models in the second dataset.

	AR(1)	AR(2)	AR(3)	AR(4)
lag(s)	1	2	3	4
AIC	22359.771	22236.837	22136.843	22097.596
BIC	22377.589	22260.594	22166.538	22133.231
RMS error	7.777	8.629	10.069	11.047
Correlation	0.522	0.457	0.401	0.335
MPE	-0.618	0.046	0.739	1.149
MAE	5.994	6.706	7.953	9.017

Akaike Information Critera (AIC) and Bayesian information criterion (BIC) show the simplicity and goodness of an AR model. If a model has a lower AIC and BIC, it will be generally better than others. So for the first dataset, AR(2) is better than AR(1), and for the second dataset, AR(4) has a lower value among others.

5. **Summarize your findings and observations briefly in a final discussion. Submit both the developed code and your document to the Assignment 1 folder on D2L.**

   This assignment is available on my GitHub repository [2], and the below table shows some significant paths. Also, for convenience, the codes are available in Appendix.

The Important Paths:	
./EE6563/Dataset/Ass1	Assignment datasets
./EE6563/code/Ass1/Temp.py	Assignment code (the 1^{st} dataset)
./EE6563/code/Ass1/sun.py	Assignment code (the 2^{nd} dataset)
./EE6563/manuscript/src/figures	Assignment figures
./EE6563/manuscript/src/tables	Assignment tables
./EE6563/manuscript/src/Ass1.tex	Assignment documents

REFERENCES

[1] *Stationarity and detrending (ADF/KPSS) [Pleaseinsert"PrerenderUnicode–"intopreamble] statsmodels*. [Online]. Available: https://www.statsmodels.org/dev/examples/notebooks/generated/stationarity_detrending_adf_kpss.html.

[2] *SKazemii/EE6563*. [Online]. Available: https://github.com/SKazemii/EE6563.

Appendix (codes)

The script of sun.py

```
"""https://www.machinelearningplus.com/time-series/time-series-analysis-python/"""
import warnings

warnings.filterwarnings("ignore")

import math
import os

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import statsmodels.api as sm

from pandas.plotting import lag_plot
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa import seasonal, stattools
from statsmodels.tsa.ar_model import AutoReg
from statsmodels.tsa.arima_model import ARIMA


plt.rcParams["figure.figsize"] = (14, 7)
a = "Ass1_D2_"

print("[INFO] Setting directories")
project_dir = os.getcwd()
fig_dir = os.path.join(project_dir, "manuscript", "src", "figures", ...
    "Ass1")
tbl_dir = os.path.join(project_dir, "manuscript", "src", "tables", ...
    "Ass1")
data_dir = os.path.join(project_dir, "Dataset", "Ass1")
dataset_file = os.path.join(data_dir, "monthly-sunspots.csv")
# dataset_file = os.path.join(data_dir, "temperatures.csv")


print("[INFO] Reading the first dataset")
series = pd.read_csv(
    dataset_file,
```

```
    header=0,
    index_col=0,
).dropna()

series.index = pd.to_datetime(series.index)

plt.close("all")
##############################################################################
#########     Saving and showing the plot of the raw signal ##########
##############################################################################

print("[INFO] Saving and showing the plot of the first dataset")
plt.figure(0)
fig = series.plot()
plt.savefig(os.path.join(fig_dir, a + "raw_signal.png"))


plt.figure()
plt.plot(series["1749":"1800"])
plt.savefig(os.path.join(fig_dir, a + "raw_signal_1990.png"))


series_diff = series.copy(deep=True)
series_diff["Sunspots"] = series.diff().dropna()
series_diff.dropna(inplace=True)
plt.figure()
fig = plt.plot(series_diff)
plt.savefig(os.path.join(fig_dir, a + "1_diff_raw_signal.png"))


series_2diff = series.copy(deep=True)
series_2diff["Sunspots"] = series.diff().diff().dropna()
series_2diff.dropna(inplace=True)
fig = plt.plot(series_2diff)
plt.savefig(os.path.join(fig_dir, a + "2_diff_raw_signal.png"))


print("[INFO] Saving and printing the head of the first dataset")
with open(os.path.join(tbl_dir, a + "raw_signal.tex"), "w") as tf:
    tf.write(series.head(5).to_latex())


print(series.describe())
```

```
with open(os.path.join(tbl_dir, a + ...
    "raw_signal_summary_statistics.tex"), "w") as tf:
    tf.write(series.describe().to_latex())

################################################################################
#########      Decompositions: Moving Avarage function     ##########
################################################################################
print("[INFO] Saving and showing the plot of Moving Avarage function")

# r.agg, r.apply, r.count, r.exclusions, r.max, r.median, r.name, ...
    r.quantile, r.kurt, r.cov, r.corr, r.aggregate, r.std, r.skew, ...
    r.sum, r.var
r = series.rolling(window=12)

plt.figure()
axes = plt.axes()
series.plot(color="red", ax=axes)
r.mean().plot(style="b", linewidth=3, ax=axes)

plt.legend(["input data", "Seasonal component"])
plt.savefig(os.path.join(fig_dir, a + "Moving_Avrage.png"))


################################################################################
#########      Decompositions: seasonal_decompose method   ##########
################################################################################

print("[INFO] Plot the decomposition by seasonal_decompose method...")
decomposition_sd = seasonal.seasonal_decompose(
    series, model="additive", extrapolate_trend="freq" # , period=120
)

fig = decomposition_sd.plot()
plt.savefig(os.path.join(fig_dir, a + "seasonal_decompose.png"))


################################################################################
#########        Seasonal Modeling (fitting polynomial)    ##########
################################################################################
resample = series.resample("AS").mean()
plt.figure()
plt.plot(resample)
plt.savefig(os.path.join(fig_dir, a + "resample.png"))
```

```
print("[INFO] Plot the decomposition by fitting polynomial method...")
X = [i % 120 for i in range(0, len(series))]
y = series.values

degree = 2
coef = np.polyfit(X, y, degree)
print("[INFO] polynomial Coefficients are :\n%s\n" % coef)


curve = list()
for i in range(len(X)):
    value = 88
    for d in range(degree):
        value += X[i] ** (degree - d) * coef[d]
    curve.append(value)

plt.figure()
plt.plot(y)
plt.plot(curve, color="red", linewidth=3)
plt.legend(["input data", "Seasonal component"])
plt.savefig(os.path.join(fig_dir, a + "fiting_polynomial.png"))


#############################################################################
#########        Decompositions: Differencing method       ##########
#############################################################################
print("[INFO] Plot the detrending signal by Differencing method...")
diff = list()
period = 1
for i in range(period, len(X)):
    value = series.values[i] - series.values[i - period]
    diff.append(value)

plt.figure()
plt.plot(series.values[:-1] - diff)
plt.ylabel("detrend Signal")
plt.savefig(os.path.join(fig_dir, a + "one_diff.png"))


#############################################################################
#########            Decompositions: STL method            ##########
```

```
##############################################################################

print("[INFO] Plot the decomposition by STL method...")
decomposition_STL = seasonal.STL(series, period=130).fit()
fig = decomposition_STL.plot()
plt.savefig(os.path.join(fig_dir, a + "STL.png"))


##############################################################################
#########       Decompositions: LinearRegression method     ##########
##############################################################################

print("[INFO] Plot the decomposition by LinearRegression method...")
X = [i for i in range(0, len(series))]
X = np.reshape(X, (len(X), 1))
y = series.values
model = LinearRegression()
model.fit(X, y)
trend = model.predict(X)

# plot the raw signal
plt.subplot(411)
plt.plot(y)
plt.ylabel("the raw signal")

# plot trend
plt.subplot(412)
plt.plot(trend)
plt.ylabel("trend")
# detrending
detrended = [y[i] - trend[i] for i in range(0, len(series))]
# plot Detrended
plt.subplot(413)
plt.plot(detrended)
plt.ylabel("seasonal")

##############################################################################
#########        Decompositions: Differencing method        ##########
##############################################################################
"""https://machinelearningmastery.com/time-series-seasonality-with-python/"""
print("[INFO] Plot the seasonal signal by Differencing method...")

diff = list()
```

```
period = 120
for i in range(period, len(X)):
    value = detrended[i] - detrended[i - period]
    diff.append(value)

# plot Residual
plt.subplot(414)
plt.plot(diff)
plt.ylabel("residual")
plt.savefig(os.path.join(fig_dir, a + "LinearRegression_diff.png"))


################################################################################
#########             stationary test: ADF method           ##########
################################################################################

"""residual = {decomposition_sd.resid, decomposition_STL.resid, ...
    diff}"""
residual = decomposition_STL.resid

print("[INFO] ACF plot for residual component...")
plt.figure()
plot_acf(residual, lags=100)
plt.savefig(os.path.join(fig_dir, a + "ACF.png"))

print("[INFO] Results of Dickey-Fuller Test:")
result = stattools.adfuller(residual, autolag="AIC")
dfoutput = pd.Series(
    result[0:4],
    index=["ADF Statistic", "p-value", "#Lags Used", "Number of ...
        Observations Used"],
)
for key, value in result[4].items():
    dfoutput["Critical Value (%s)" % key] = value
print(dfoutput)

print("[INFO] saving Results of Dickey-Fuller Test on file...")
with open(os.path.join(tbl_dir, a + "ADF.tex"), "w") as tf:
    tf.write(dfoutput.to_latex(index=True))


print("[INFO] ACF plot for the first order diff...")
plt.figure()
```

```
plot_acf(series_diff, lags=100)
plt.savefig(os.path.join(fig_dir, a + "ACF_1_diff.png"))

print("[INFO] Results of Dickey-Fuller Test:")
result = stattools.adfuller(series_diff, autolag="AIC")
dfoutput = pd.Series(
    result[0:4],
    index=["ADF Statistic", "p-value", "#Lags Used", "Number of ...
        Observations Used"],
)
for key, value in result[4].items():
    dfoutput["Critical Value (%s)" % key] = value
print(dfoutput)

print("[INFO] saving Results of Dickey-Fuller Test on file...")
with open(os.path.join(tbl_dir, a + "ADF_1_diff.tex"), "w") as tf:
    tf.write(dfoutput.to_latex(index=True))


print("[INFO] ACF plot for the second order diff...")
plt.figure()
plot_acf(series_2diff, lags=100)
plt.savefig(os.path.join(fig_dir, a + "ACF_2_diff.png"))

print("[INFO] Results of Dickey-Fuller Test:")
result = stattools.adfuller(series_2diff, autolag="AIC")
dfoutput = pd.Series(
    result[0:4],
    index=["ADF Statistic", "p-value", "#Lags Used", "Number of ...
        Observations Used"],
)
for key, value in result[4].items():
    dfoutput["Critical Value (%s)" % key] = value
print(dfoutput)

print("[INFO] saving Results of Dickey-Fuller Test on file...")
with open(os.path.join(tbl_dir, a + "ADF_2_diff.tex"), "w") as tf:
    tf.write(dfoutput.to_latex(index=True))


################################################################################
#########            stationary test: KPSS method           ##########
################################################################################
```

```
"""
## ...
    https://www.statsmodels.org/stable/examples/notebooks/generated/stationarity_detren
"""
print("[INFO] Results of KPSS Test:")
Results = stattools.kpss(residual, regression="c", nlags="auto")
kpss_output = pd.Series(Results[0:3], index=["KPSS Statistic", ...
    "p-value", "Lags Used"])
for key, value in Results[3].items():
    kpss_output["Critical Value (%s)" % key] = value
print(kpss_output)

print("[INFO] saving Results of KPSS Test on file...")
with open(os.path.join(tbl_dir, a + "KPSS.tex"), "w") as tf:
    tf.write(kpss_output.to_latex(index=True))


print("[INFO] Results of KPSS Test:")
Results = stattools.kpss(series_diff, regression="c", nlags="auto")
kpss_output = pd.Series(Results[0:3], index=["KPSS Statistic", ...
    "p-value", "Lags Used"])
for key, value in Results[3].items():
    kpss_output["Critical Value (%s)" % key] = value
print(kpss_output)

print("[INFO] saving Results of KPSS Test on file...")
with open(os.path.join(tbl_dir, a + "KPSS_1_diff.tex"), "w") as tf:
    tf.write(kpss_output.to_latex(index=True))

print("[INFO] Results of KPSS Test:")
Results = stattools.kpss(series_2diff, regression="c", nlags="auto")
kpss_output = pd.Series(Results[0:3], index=["KPSS Statistic", ...
    "p-value", "Lags Used"])
for key, value in Results[3].items():
    kpss_output["Critical Value (%s)" % key] = value
print(kpss_output)

print("[INFO] saving Results of KPSS Test on file...")
with open(os.path.join(tbl_dir, a + "KPSS_2_diff.tex"), "w") as tf:
    tf.write(kpss_output.to_latex(index=True))

#############################################################################
#########             stationary test: PACF method          ###########
```

```
##############################################################################
""" ...
    https://towardsdatascience.com/detecting-stationarity-in-time-series-data-d29e0a21e
"""
print("[INFO] PACF plot for residual component...")
PACF_output = stattools.pacf(residual)
plt.figure()
plt.stem(PACF_output)
plt.savefig(os.path.join(fig_dir, a + "PACF.png"))

plt.figure()
fig, axes = plt.subplots(2, 1)
plot_acf(residual, lags=50, ax=axes[0])
plot_pacf(residual, lags=50, ax=axes[1])
plt.savefig(os.path.join(fig_dir, a + "PACF_ACF.png"))

plt.figure()
fig, axes = plt.subplots(2, 1)
plot_acf(series_diff, lags=50, ax=axes[0])
plot_pacf(series_diff, lags=50, ax=axes[1])
plt.savefig(os.path.join(fig_dir, a + "PACF_ACF_1_diff.png"))

plt.figure()
fig, axes = plt.subplots(2, 1)
plot_acf(series, lags=150, ax=axes[0])
plot_pacf(series, lags=150, ax=axes[1])
plt.savefig(os.path.join(fig_dir, a + "PACF_ACF_series.png"))
##############################################################################
#########             stationary test: Lag Plots            ##########
##############################################################################

print("[INFO] Lag plot for residual component...")

fig, axes = plt.subplots(2, 2, sharex=True, sharey=True, dpi=100)
for i, ax in enumerate(axes.flatten()[:4]):
    lag_plot(series, lag=i + 1, ax=ax, c="firebrick")
    ax.set_title("Lag " + str(i + 1))

fig.suptitle("Lag Plots of the Dataset")
plt.savefig(os.path.join(fig_dir, a + "Lag_Plots.png"))


fig, axes = plt.subplots(2, 2, sharex=True, sharey=True, dpi=100)
```

```
for i, ax in enumerate(axes.flatten()[:4]):
    lag_plot(residual, lag=i + 1, ax=ax, c="firebrick")
    ax.set_title("Lag " + str(i + 1))

fig.suptitle("Lag Plots of the residual component")
plt.savefig(os.path.join(fig_dir, a + "Lag_Plots_residual.png"))

fig, axes = plt.subplots(2, 2, sharex=True, sharey=True, dpi=100)
for i, ax in enumerate(axes.flatten()[:4]):
    lag_plot(series_diff, lag=i + 1, ax=ax, c="firebrick")
    ax.set_title("Lag " + str(i + 1))

fig.suptitle("Lag Plots of the first order diff")
plt.savefig(os.path.join(fig_dir, a + "Lag_Plots_1_diff.png"))

fig, axes = plt.subplots(2, 2, sharex=True, sharey=True, dpi=100)
for i, ax in enumerate(axes.flatten()[:4]):
    lag_plot(series_2diff, lag=i + 1, ax=ax, c="firebrick")
    ax.set_title("Lag " + str(i + 1))

fig.suptitle("Lag Plots of the second order diff")
plt.savefig(os.path.join(fig_dir, a + "Lag_Plots_2_diff.png"))
##############################################################################
#########         Predict Approach of Auto Regression      ##########
##############################################################################
"""
https://towardsdatascience.com/trend-seasonality-moving-average-auto-regressive-model-
"""


X = decomposition_STL.resid.values

plt.figure()
fig, axes = plt.subplots(2, 1)
plot_acf(X, lags=50, ax=axes[0])
plot_pacf(X, lags=50, ax=axes[1])
plt.savefig(os.path.join(fig_dir, a + "PACF_ACF_X.png"))


print("[INFO] spliting dataset to train and test set...")
n_predict = 14
train, test = X[1 : len(X) - n_predict], X[len(X) - n_predict :]
```

```
ar_orders = [1, 2, 3, 4]
fitted_model_dict = {}

AIC_list = list()
BIC_list = list()
RMS_list = list()
cof_list = list()
ord_list = list()
col_list = list()
mae_list = list()
mpe_list = list()
cor_list = list()


for idx, ar_order in enumerate(ar_orders):

    print("[INFO] train autoregression...")
    model = ARIMA(train, order=(ar_order, 0, 0))
    model_fit = model.fit()

    fitted_model_dict[ar_order] = model_fit

    col_list.append("AR({:1.0f})".format(ar_order))

    predictions = model_fit.predict(
        start=len(train), end=len(train) + len(test) - 1, dynamic=False
    )

    RMS_list.append("{:2.3f}".format(np.sqrt(mean_squared_error(test, ...
        predictions))))
    mae_list.append("{:2.3f}".format(np.mean(np.abs(predictions - ...
        test))))
    mpe_list.append("{:2.3f}".format(np.mean((predictions - test) / ...
        test)))
    cor_list.append("{:2.3f}".format(np.corrcoef(predictions, ...
        test)[0, 1]))

    plt.figure()
    xpos = np.arange(len(X))
    plt.plot(X[2700:], "r", linewidth=0.5)
    plt.plot(
        xpos[len(X) - n_predict - 2700 : len(X) - 2700], ...
            predictions[:], color="blue"
```

```
    )
    plt.legend(["train+Test", "Predictions"])
    plt.savefig(os.path.join(fig_dir, a + "_" + str(idx + 1) + ...
        "_AR2.png"))

for idx, ar_order in enumerate(ar_orders):
    plt.figure(0)
    plt.subplot(len(ar_orders), 1, idx + 1)
    plt.plot(train[:100])
    plt.plot(fitted_model_dict[ar_order].fittedvalues[:100])
    plt.title("AR({:1.0f}) Fit".format(ar_order), fontsize=16)


plt.tight_layout()
plt.savefig(os.path.join(fig_dir, a + "ARs models.png"))

print("[INFO] AIC and BIC of autoregression models...")
for ar_order in ar_orders:
    AIC_list.append("{:2.3f}".format(fitted_model_dict[ar_order].aic))
    cof_list.append(fitted_model_dict[ar_order].params)
    ord_list.append(ar_order)
    BIC_list.append("{:2.3f}".format(fitted_model_dict[ar_order].bic))


df = pd.DataFrame(
    np.row_stack(
        [ord_list, AIC_list, BIC_list, RMS_list, cor_list, mpe_list, ...
           mae_list]
    ),
    index=["lag(s)", "AIC", "BIC", "RMS error", "Correlation", ...
        "MPE", "MAE"],
)
df.columns = col_list
print(df)
with open(os.path.join(tbl_dir, a + "AR.tex"), "w") as tf:
    tf.write(df.to_latex(index=True))
```

The script of Temp.py

```
"""https://www.machinelearningplus.com/time-series/time-series-analysis-python/"""
import warnings
```

```
warnings.filterwarnings("ignore")

import math
import os

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import statsmodels.api as sm
import seaborn as sns

from pandas.plotting import lag_plot
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa import seasonal, stattools
from statsmodels.tsa.ar_model import AutoReg
from statsmodels.tsa.arima_model import ARIMA


plt.rcParams["figure.figsize"] = (14, 7)
a = "Ass1_D1_"

print("[INFO] Setting directories")
project_dir = os.getcwd()
fig_dir = os.path.join(project_dir, "manuscript", "src", "figures", ...
    "Ass1")
tbl_dir = os.path.join(project_dir, "manuscript", "src", "tables", ...
    "Ass1")
data_dir = os.path.join(project_dir, "Dataset", "Ass1")
# dataset_file = os.path.join(data_dir, "monthly-sunspots.csv")
dataset_file = os.path.join(data_dir, "temperatures.csv")


print("[INFO] Reading the first dataset")
series = pd.read_csv(
    dataset_file,
    header=0,
    index_col=0,
).dropna()

series.index = pd.to_datetime(series.index)
```

```
################################################################################
#########     Saving and showing the plot of the raw signal ##########
################################################################################

print("[INFO] Saving and showing the plot of the first dataset")
plt.figure(0)
fig = series.plot()
plt.savefig(os.path.join(fig_dir, a + "raw_signal.png"))

plt.figure()
plt.plot(series["1990":"1991"])
plt.savefig(os.path.join(fig_dir, a + "raw_signal_1990.png"))

plt.figure()
plt.plot(series.loc["1986"])
plt.savefig(os.path.join(fig_dir, a + "raw_signal_1986.png"))


print("[INFO] Saving and printing the head of the first dataset")
with open(os.path.join(tbl_dir, a + "raw_signal.tex"), "w") as tf:
    tf.write(series.head(5).to_latex())


print(series.describe())
with open(os.path.join(tbl_dir, a + ...
    "raw_signal_summary_statistics.tex"), "w") as tf:
    tf.write(series.describe().to_latex())

################################################################################
#########       Decompositions: Moving Avrage function      ##########
################################################################################
print("[INFO] Saving and showing the plot of Moving Avrage function")

# r.agg, r.apply, r.count, r.exclusions, r.max, r.median, r.name, ...
    r.quantile, r.kurt, r.cov, r.corr, r.aggregate, r.std, r.skew, ...
    r.sum, r.var
r = series.rolling(window=100)

plt.figure()
axes = plt.axes()
series.plot(color="red", ax=axes)
r.mean().plot(style="b", linewidth=3, ax=axes)
```

```
plt.legend(["input data", "Seasonal component"])
plt.savefig(os.path.join(fig_dir, a + "Moving_Avrage.png"))


################################################################################
#########      Decompositions: seasonal_decompose method    ##########
################################################################################

print("[INFO] Plot the decomposition by seasonal_decompose method...")
decomposition_sd = seasonal.seasonal_decompose(
    series, model="additive", extrapolate_trend="freq", period=365
)

fig = decomposition_sd.plot()
plt.savefig(os.path.join(fig_dir, a + "seasonal_decompose.png"))


################################################################################
#########        Seasonal Modeling (fiting polynomial)     ##########
################################################################################

resample = series.resample("M").mean()
plt.figure()
plt.plot(resample)
plt.savefig(os.path.join(fig_dir, a + "resample.png"))


print("[INFO] Plot the decomposition by fiting polynomial method...")
X = [i % 365 for i in range(0, len(series))]
y = series.values

degree = 5
coef = np.polyfit(X, y, degree)
print("[INFO] polynomial Coefficients are :\n%s\n" % coef)


curve = list()
for i in range(len(X)):
    value = 13
    for d in range(degree):
        value += X[i] ** (degree - d) * coef[d]
    curve.append(value)
```

```
plt.figure()
plt.plot(y)
plt.plot(curve, color="red", linewidth=3)
plt.legend(["input data", "Seasonal component"])
plt.savefig(os.path.join(fig_dir, a + "fiting_polynomial.png"))


################################################################################
#########        Decompositions: Differencing method       ##########
################################################################################
print("[INFO] Plot the detrending signal by Differencing method...")
diff = list()
period = 1
for i in range(period, len(X)):
    value = series.values[i] - series.values[i - period]
    diff.append(value)

plt.figure()
plt.plot(series.values[:-1] - diff)
plt.ylabel("detrend Signal")
plt.savefig(os.path.join(fig_dir, a + "one_diff.png"))


################################################################################
#########             Decompositions: STL method            ##########
################################################################################

print("[INFO] Plot the decomposition by STL method...")
decomposition_STL = seasonal.STL(series, period=365).fit()
fig = decomposition_STL.plot()
plt.savefig(os.path.join(fig_dir, a + "STL.png"))


################################################################################
#########       Decompositions: LinearRegression method    ##########
################################################################################

print("[INFO] Plot the decomposition by LinearRegression method...")
X = [i for i in range(0, len(series))]
X = np.reshape(X, (len(X), 1))
y = series.values
model = LinearRegression()
```

```
model.fit(X, y)
trend = model.predict(X)

# plot the raw signal
plt.subplot(411)
plt.plot(y)
plt.ylabel("the raw signal")

# plot trend
plt.subplot(412)
plt.plot(trend)
plt.ylabel("trend")
# detrending
detrended = [y[i] - trend[i] for i in range(0, len(series))]
# plot Detrended
plt.subplot(413)
plt.plot(detrended)
plt.ylabel("seasonal")

#############################################################################
#########        Decompositions: Differencing method       ##########
#############################################################################
"""https://machinelearningmastery.com/time-series-seasonality-with-python/"""
print("[INFO] Plot the seasonal signal by Differencing method...")

diff = list()
period = 365
for i in range(period, len(X)):
    value = detrended[i] - detrended[i - period]
    diff.append(value)

# plot Residual
plt.subplot(414)
plt.plot(diff)
plt.ylabel("residual")
plt.savefig(os.path.join(fig_dir, a + "LinearRegression_diff.png"))


#############################################################################
#########             stationary test: ADF method           ##########
#############################################################################
```

```
"""residual = {decomposition_sd.resid, decomposition_STL.resid, ...
    diff}"""
residual = decomposition_sd.resid

print("[INFO] ACF plot for residual component...")
plt.figure()
plot_acf(residual, lags=100)
plt.savefig(os.path.join(fig_dir, a + "ADF.png"))

print("[INFO] Results of Dickey-Fuller Test:")
result = stattools.adfuller(residual, autolag="AIC")
dfoutput = pd.Series(
    result[0:4],
    index=["ADF Statistic", "p-value", "#Lags Used", "Number of ...
        Observations Used"],
)
for key, value in result[4].items():
    dfoutput["Critical Value (%s)" % key] = value
print(dfoutput)

print("[INFO] saving Results of Dickey-Fuller Test on file...")
with open(os.path.join(tbl_dir, a + "ADF.tex"), "w") as tf:
    tf.write(dfoutput.to_latex(index=True))

##############################################################################
#########            stationary test: KPSS method           ##########
##############################################################################

print("[INFO] Results of KPSS Test:")
Results = stattools.kpss(residual, regression="c", nlags="auto")
kpss_output = pd.Series(Results[0:3], index=["KPSS Statistic", ...
    "p-value", "Lags Used"])
for key, value in Results[3].items():
    kpss_output["Critical Value (%s)" % key] = value
print(kpss_output)

print("[INFO] saving Results of KPSS Test on file...")
with open(os.path.join(tbl_dir, a + "KPSS.tex"), "w") as tf:
    tf.write(kpss_output.to_latex(index=True))


##############################################################################
#########            stationary test: PACF method           ##########
```

```
##############################################################################


print("[INFO] PACF plot for residual component...")
PACF_output = stattools.pacf(residual)
plt.figure()
plt.stem(PACF_output)
plt.savefig(os.path.join(fig_dir, a + "PACF.png"))

plt.figure()
fig, axes = plt.subplots(2, 1)
plot_acf(residual, lags=50, ax=axes[0])
plot_pacf(residual, lags=50, ax=axes[1])
plt.savefig(os.path.join(fig_dir, a + "PACF_ACF.png"))

plt.figure()
fig, axes = plt.subplots(2, 1)
plot_acf(series, lags=500, ax=axes[0])
plot_pacf(series, lags=500, ax=axes[1])
plt.savefig(os.path.join(fig_dir, a + "PACF_ACF_series.png"))

##############################################################################
#########             stationary test: Lag Plots            ##########
##############################################################################

'''(Points get wide and scattered with increasing lag -> lesser ...
    correlation)\n"'''
print("[INFO] Lag plot for residual component...")

fig, axes = plt.subplots(2, 2, sharex=True, sharey=True, dpi=100)
for i, ax in enumerate(axes.flatten()[:4]):
    lag_plot(series, lag=i + 1, ax=ax, c="firebrick")
    ax.set_title("Lag " + str(i + 1))

fig.suptitle("Lag Plots of the Dataset")
plt.savefig(os.path.join(fig_dir, a + "Lag_Plots.png"))


fig, axes = plt.subplots(2, 2, sharex=True, sharey=True, dpi=100)
for i, ax in enumerate(axes.flatten()[:4]):
    lag_plot(residual, lag=i + 1, ax=ax, c="firebrick")
    ax.set_title("Lag " + str(i + 1))
```

```
fig.suptitle("Lag Plots of the residual component")
plt.savefig(os.path.join(fig_dir, a + "Lag_Plots_residual.png"))

##############################################################################
#########         Predict Approach of Auto Regression      ##########
##############################################################################
"""
https://towardsdatascience.com/trend-seasonality-moving-average-auto-regressive-model-
"""
X = decomposition_STL.resid.values

plt.figure()
fig, axes = plt.subplots(2, 1)
plot_acf(X, lags=50, ax=axes[0])
plot_pacf(X, lags=50, ax=axes[1])
plt.savefig(os.path.join(fig_dir, a + "PACF_ACF_X.png"))


print("[INFO] spliting dataset to train and test set...")
n_predict = 14
train, test = X[1 : len(X) - n_predict], X[len(X) - n_predict :]

ar_orders = [1, 2]
fitted_model_dict = {}

AIC_list = list()
BIC_list = list()
RMS_list = list()
cof_list = list()
ord_list = list()
col_list = list()
mae_list = list()
mpe_list = list()
cor_list = list()


for idx, ar_order in enumerate(ar_orders):

    print("[INFO] train autoregression...")
    model = ARIMA(train, order=(ar_order, 0, 0))
    model_fit = model.fit()

    fitted_model_dict[ar_order] = model_fit
```

```
        col_list.append("AR({:1.0f})".format(ar_order))

        predictions = model_fit.predict(
            start=len(train), end=len(train) + len(test) - 1, dynamic=False
        )

        RMS_list.append("{:2.3f}".format(np.sqrt(mean_squared_error(test, ...
            predictions))))
        mae_list.append("{:2.3f}".format(np.mean(np.abs(predictions - ...
            test))))
        mpe_list.append("{:2.3f}".format(np.mean((predictions - test) / ...
            test)))
        cor_list.append("{:2.3f}".format(np.corrcoef(predictions, ...
            test)[0, 1]))

        plt.figure()
        xpos = np.arange(len(X))
        plt.plot(X[3500:], "r", linewidth=0.5)
        plt.plot(
            xpos[len(X) - n_predict - 3500 : len(X) - 3500], ...
                predictions[:], color="blue"
        )
        plt.legend(["train+Test", "Predictions"])
        plt.savefig(os.path.join(fig_dir, a + "_" + str(idx + 1) + ...
            "_AR2.png"))

    for idx, ar_order in enumerate(ar_orders):
        plt.figure(0)
        plt.subplot(len(ar_orders), 1, idx + 1)
        plt.plot(train[:100])
        plt.plot(fitted_model_dict[ar_order].fittedvalues[:100])
        plt.title("AR({:1.0f}) Fit".format(ar_order), fontsize=16)

    plt.tight_layout()
    plt.savefig(os.path.join(fig_dir, a + "ARs models.png"))

    print("[INFO] AIC and BIC of autoregression models...")
    for ar_order in ar_orders:
        AIC_list.append("{:2.3f}".format(fitted_model_dict[ar_order].aic))
        cof_list.append(fitted_model_dict[ar_order].params)
        ord_list.append(ar_order)
        BIC_list.append("{:2.3f}".format(fitted_model_dict[ar_order].bic))
```

```
df = pd.DataFrame(
    np.row_stack(
        [ord_list, AIC_list, BIC_list, RMS_list, cor_list, mpe_list, ...
           mae_list]
    ),
    index=["lag(s)", "AIC", "BIC", "RMS error", "Correlation", ...
       "MPE", "MAE"],
)
df.columns = col_list
print(df)
with open(os.path.join(tbl_dir, a + "AR.tex"), "w") as tf:
    tf.write(df.to_latex(index=True))
```