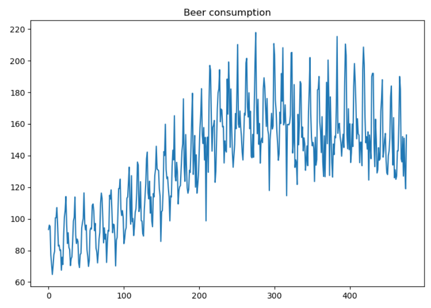
Statistics for Data Science - Assignment 1

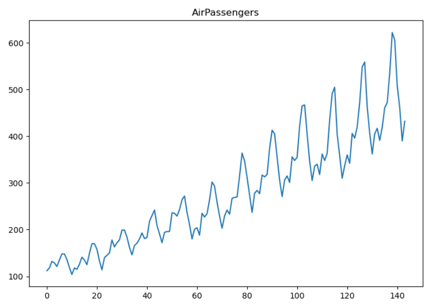
Bouruc Petru-Liviu

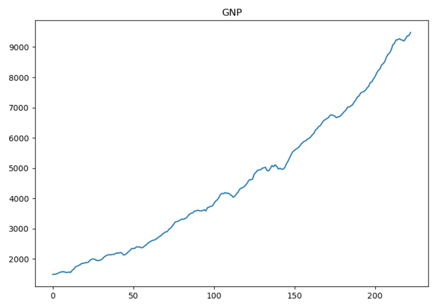
27/05/2023

1 Exercise 1

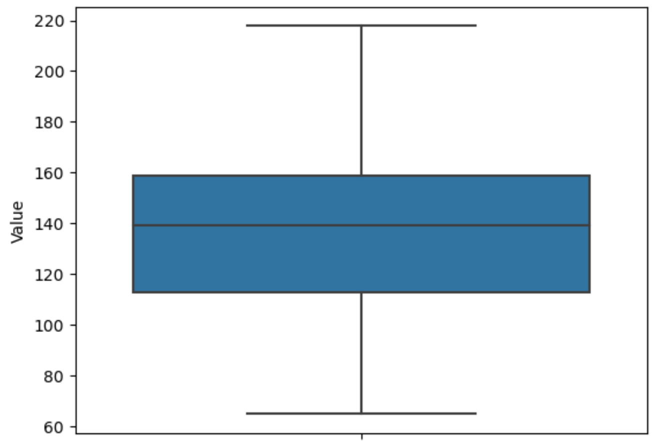
For the first exercise, I choose as dataset the Monthly Beer Production in Austria Time Series, from years 1956 to 1995. In the given .csv, the data is univariate, such that for each given time there is a single numerical observation, representing the quantity of beer produced in that specific month. As for the plot, we can see that the data is not similar with the one from the laboratory (AirPassangers and gnp):



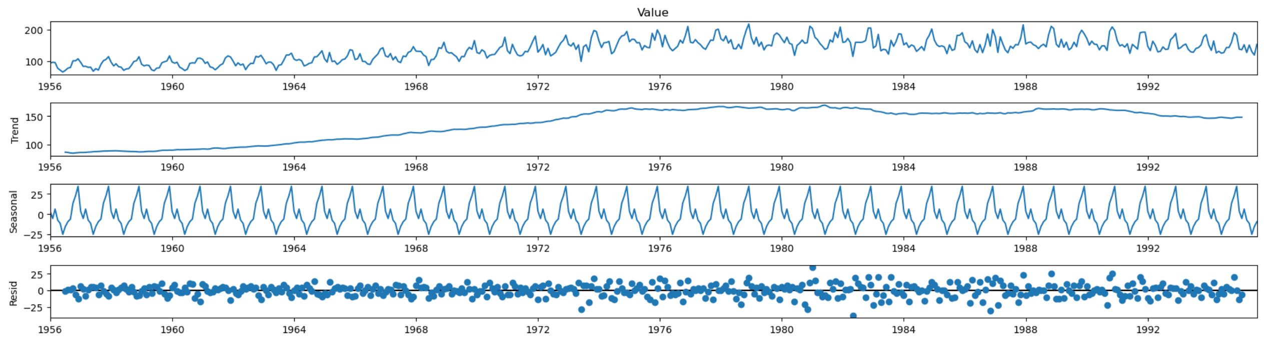




Firstly, we look at our data to see if there are unusual or extreme observations in the dataset. There are no missing values or 0 values in our time series, as well as our data does not contain outliers. This can also be seen when plotting the box plot:

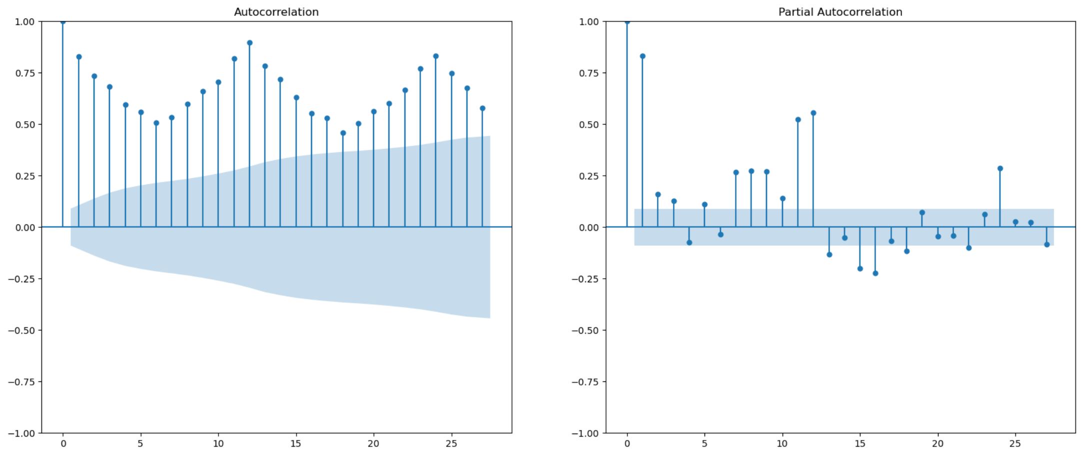


Among the first things we can observe about data is that the production increases over time, until 1974. From that year onwards, it stays the same, with a slight decrease at the end. So, our mean values for the data varies over time, indicating that there are multiple trends, our data being non-stationarity. We can also plot the seasonal decomposition for our time series to have a better look:



As we can see from the plot, there are spikes every year at the end, in December. This is an indicator that the series in seasonal.

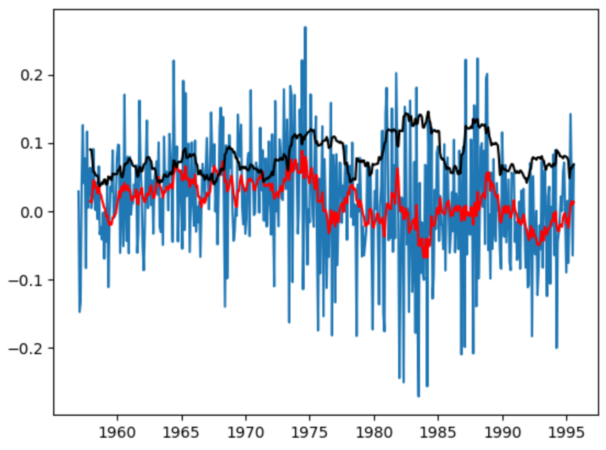
Now, to have a better understanding of our time series, we can take a look at the ACF and PACF plot:



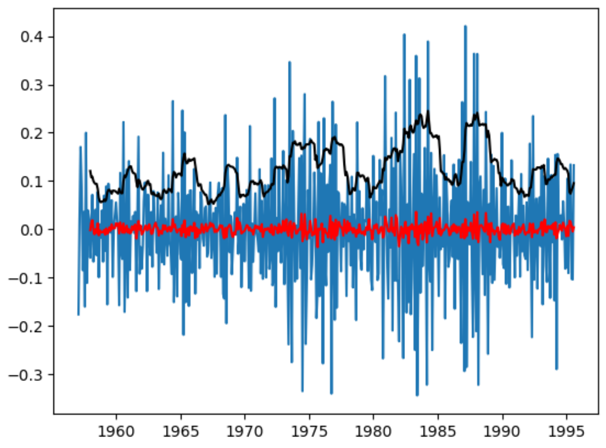
To better enforce the observation made by looking at the trend graph (that our time series is non-stationary), I run the ADF-test. It returned me a value of -2.282, which compared with the critical values, we can say that, with a significance level of 10%, our data is indeed *not stationary*.

To make the series stationary, I first logged the series than I applied differentiation with a period of 12 (seasonal differenciating) because as mentioned before and as we can see in ACF, there is a repetition every 12 lags.

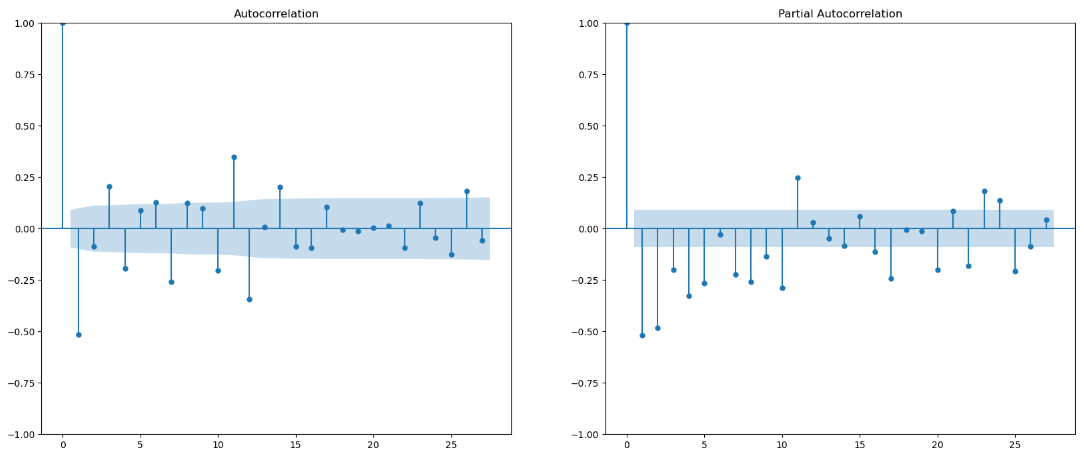
Running the ADF test again, it gave me a p-value of 0.065, which is much smaller than the previous value, but still not enough to tell that our data is stationary.



From the plot above and the p-value we can see that our data is still not stationary, so I differenciated it one more time. After that, ADF test gave me a very small p-value: 1.3e-17. Also, from the plotting of the mean value we can see that our data is now stationary.



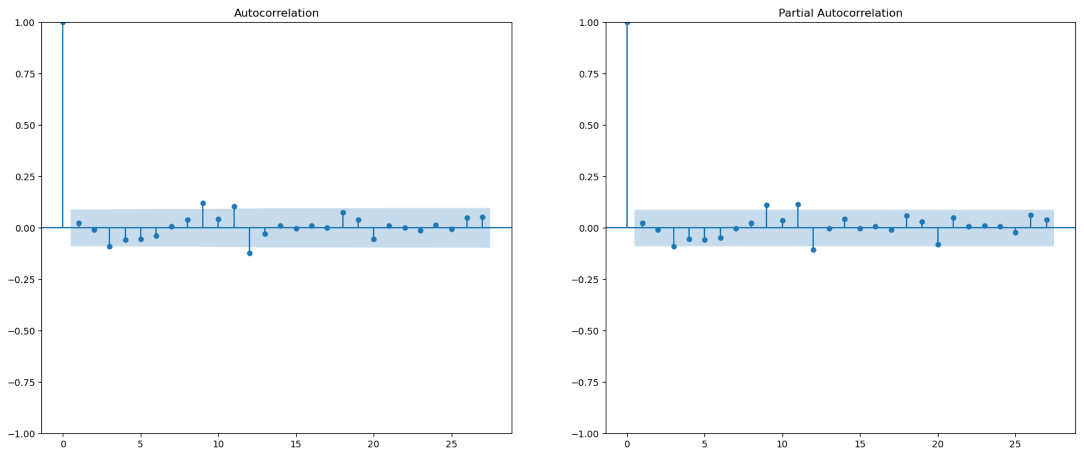
All this being done, we can look at the ACF and PACF plots for the differenciated data in order to make observations about the possible values of p and q for the forecasting model. More than that, for this time series I chose to use the SARIMA model (Seasonal ARIMA), because our data is seasonal. Seasonal differenciatig can achieve stationarity as it is similar to ordinary differenciating, but instead of substracting consecutive terms, it looks for the terms from the previous season.



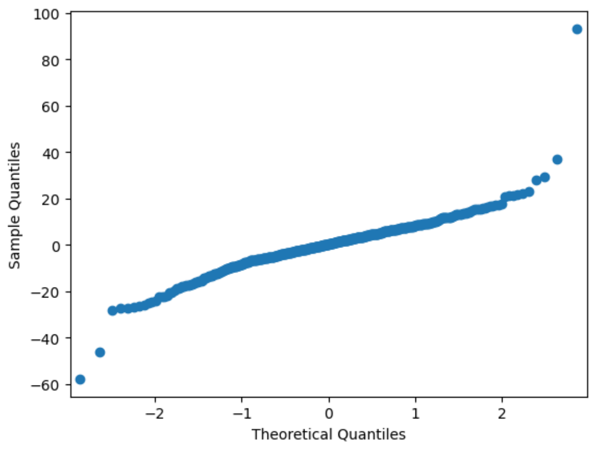
In our SARIMA(p,d,q)x(P,D,Q)[S] model, we are dealing with some parameters: p can be determined from the PACF plot and represents the AR model lags, d is the degree of differencing and q which can be determined from the ACF plot and is the size of the moving average window (order). P, D, Q are the same, but on the seasonal level.

Looking at the differenced data, ACF suggests that we could use values 1, 2 or 3 as possible q parameters for the MA model, and value 1 for the seasonal component. PACF suggests that we can use values 4, 6 or 7 as possible p parameters for AR model and the values 1 or 2 for the seasonal component. After trying with these values, I changed them to other close values and analyzed the results (AIC results) in order to set the best parameters for the forcasting. As for the d and D parameters, it is enough set them to value 1, because after 1 differencing on the seasonal level and the regular level the data becomes stationary. To use SARIMA model we need another parameter s: the determination of seasonality. We will set this parameter $s = 12$ as that is our seasonality. The best model was that with the values $(6, 1, 3)(1, 1, 2, 12)$.

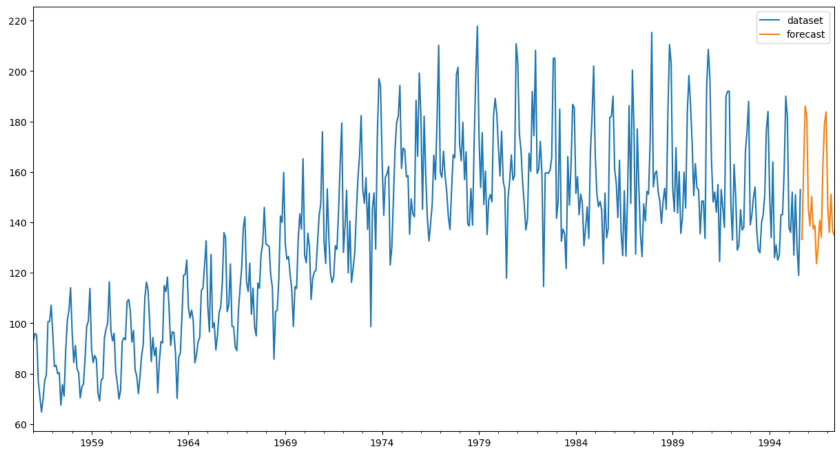
To see how our model performs, we can plot the residual distribution and see if the ACF and PACF haven't significant terms.



Another method to check if it is okay, is to observe the Q-Q Plot and see if the data is approximately linear, which it is for our plot.

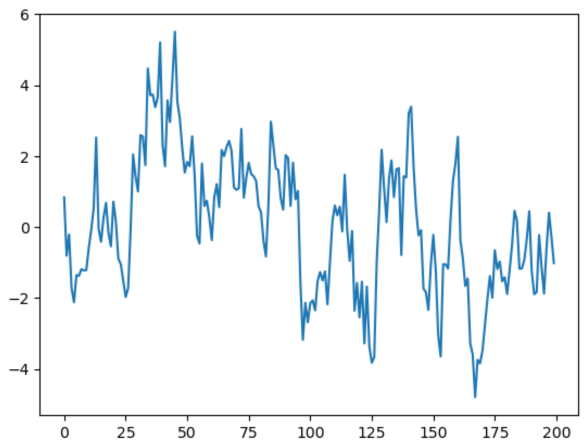


In the end, let's take a look at the forecast for the next 20 values in the time series:

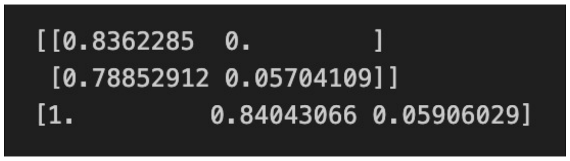


2 Exercise 2

In the exercise 2 we had to implement the Durbin-Levinson algorithm where we replace the population moments by sample moments and compare the results with the ones from the PACF algorithm from the existing packages. In order to solve this exercise, I generated 200 samples with the AR(2) model. As for the parameters, I set the $\phi = (1, -0.9)$, as said in the assignment.



As we can see, the values from the first diagonal in our D-L algorithm result are very close to the values given by PACF (2nd and 3rd values from the output).



```
[[0.8362285  0.        ]
 [0.78852912 0.05704109]]
[1.         0.84043066 0.05906029]
```

3 Code

3.1 Exercise 1

```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import astsadata
import sklearn.metrics as sme
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.statespace import sarimax


fig, axs = plt.subplots(1, 3, figsize=(25, 5))

beerproduction = pd.read_csv('monthly-beer-production-in-austr.csv')
plt.subplot(1, 3, 1)
plt.title("Beer consumption")
plt.plot(beerproduction["Value"])
```

```
airpassengers = pd.read_csv('airpassenger.csv')
plt.subplot(1, 3, 2)
plt.title("AirPassengers")
plt.plot(airpassengers["#Passengers"])

gnp_data = astsadata.gnp
plt.subplot(1, 3, 3)
plt.title("GNP")
plt.plot(gnp_data["value"].to_numpy())


beerproduction['Month'] = pd.to_datetime(beerproduction['Month'])
beerproduction.set_index('Month', inplace=True)
beerproduction.head()

sns.boxplot(y=beerproduction['Value'])

decomposition = seasonal_decompose(beerproduction['Value'])

fig = plt.figure()
fig = decomposition.plot()
fig.set_size_inches(20, 5)

adf = adfuller(beerproduction['Value'])
print("adf: " + str(adf[0]))
print("p-value: " + str(adf[1]))
print("critical values: " + str(adf[4]))

fig,ax = plt.subplots(1,2,figsize=(20,8))
fig = plot_acf(beerproduction, ax=ax[0])
fig = plot_pacf(beerproduction, ax=ax[1])
plt.show()

diff0_beerproduction = np.log(beerproduction).diff(periods=12)
diff0_beerproduction.dropna(inplace=True)

diff_beerproduction_avg = diff0_beerproduction.rolling(12).mean()
diff_beerproduction_std_dev = diff0_beerproduction.rolling(12).std()
plt.plot(diff0_beerproduction)
plt.plot(diff_beerproduction_avg, color="red")
plt.plot(diff_beerproduction_std_dev, color ="black")
plt.show()

adf = adfuller(diff0_beerproduction['Value'])
print("adf: " + str(adf[0]))
print("p-value: " + str(adf[1]))
print("critical values: " + str(adf[4]))
```

```
diff_beerproduction = np.log(beerproduction).diff(periods=12).diff()
diff_beerproduction.dropna(inplace=True)

diff_beerproduction_avg = diff_beerproduction.rolling(12).mean()
diff_beerproduction_std_dev = diff_beerproduction.rolling(12).std()
plt.plot(diff_beerproduction)
plt.plot(diff_beerproduction_avg, color="red")
plt.plot(diff_beerproduction_std_dev, color ="black")
plt.show()

adf = adfuller(diff_beerproduction['Value'])
print("adf: " + str(adf[0]))
print("p-value: " + str(adf[1]))
print("critical values: " + str(adf[4]))

fig, (ax1,ax2) = plt.subplots(1,2, figsize=(20,8))
plot_acf(diff_beerproduction, ax=ax1)
plot_pacf(diff_beerproduction, ax=ax2)

sarima = sarimax.SARIMAX(beerproduction, order=(4,1,2),
    seasonal_order=(2,1,1,12)).fit()
sarima.summary()

res = sarima.resid
fig,ax = plt.subplots(1,2,figsize=(20,8))
fig = plot_acf(res, ax=ax[0])
fig = plot_pacf(res, ax=ax[1])
plt.show()

fig = sm.qqplot(sarima.resid)
plt.show()

pred_sarima = sarima.predict("1995-09-01", "1997-05-01")
print(pred_sarima)
pd.DataFrame({'dataset':beerproduction['Value'],'forecast':pred_sarima}).plot(figsize=(15,8))
plt.show()
```

3.2 Exercise 2

```
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima_process import ArmaProcess
from statsmodels.tsa.stattools import pacf
```

```
def durbin_levinson(data, p):
    n = len(data)
    mean = np.mean(data)
    sample_moments = np.zeros(p+1)
    phi = np.zeros((p, p))
    v = np.zeros(p)

    for h in range(p+1):
        sum = 0
        for t in range(n-h):
            sum += (data[t+h]-mean) * (data[t]-mean)
        sample_moments[h] = sum / n

    phi[0, 0] = sample_moments[1] / sample_moments[0]
    v[0] = sample_moments[0] * (1 - phi[0, 0] ** 2)

    for n in range(1, p):
        sum = 0
        for k in range(n):
            sum += phi[n-1, k] * sample_moments[n-k]
        phi[n, n] = (sample_moments[n+1] - sum) / v[n-1]
        for k in range(n):
            phi[n, k] = phi[n-1, k] - phi[n, n] * phi[n-1, k-1]
        v[n] = v[n-1] * (1 - phi[n, n]**2)

    return phi


ar_params = np.array([1, -0.9])
ar_series = ArmaProcess(ar_params).generate_sample(nsample=200)
plt.plot(ar_series)
plt.show()

print(durbin_levinson(ar_series, p=2))
print(pacf(ar_series, nlags=2))
```