Gaayathri AG

Reg No: 24MSD7007

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.tsa.stattools import acf

df = pd.read_csv("IPG2211A2N lab1.csv")
df
```

```
    observation_date  IPG2211A2N
0         1939-01-01      3.3336
1         1939-02-01      3.3591
2         1939-03-01      3.4354
3         1939-04-01      3.4608
4         1939-05-01      3.4608
..               ...         ...
967       2019-08-01    112.3547
968       2019-09-01    103.1333
969       2019-10-01     93.8206
970       2019-11-01     99.9782
971       2019-12-01    109.9263

[972 rows x 2 columns]
```

1. Check whether the data is a clean data.

```
df.isnull()
```

```
     observation_date  IPG2211A2N
0               False       False
1               False       False
2               False       False
3               False       False
4               False       False
..                ...         ...
967             False       False
968             False       False
969             False       False
970             False       False
971             False       False

[972 rows x 2 columns]
```

There are null datas so not clean

2. Print the dtypes of the given data

```
df.dtypes
```

```
observation_date     object
IPG2211A2N          float64
dtype: object
```

3.Change the dtype of first column to datetime64 and rename the second column as 'Production of electric utilities'.

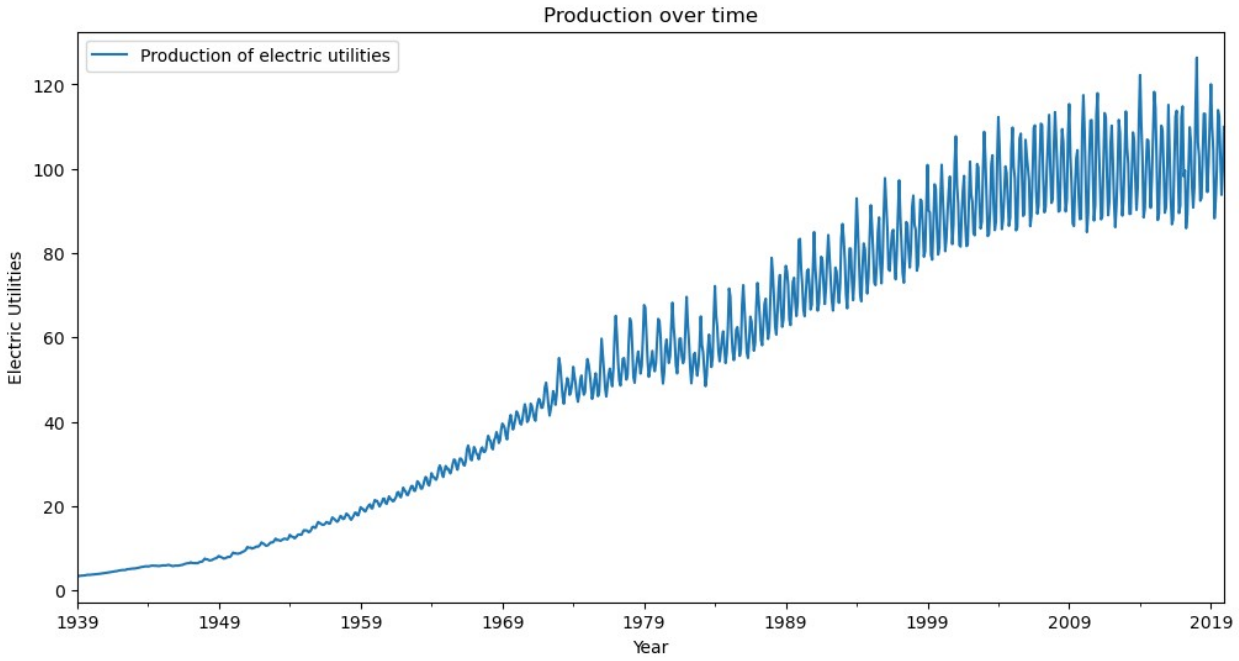
```
df["observation_date"] = pd.to_datetime(df["observation_date"])
df.rename(columns = {"IPG2211A2N": "Production of electric
utilities"},inplace = True)
df
```

```
    observation_date  Production of electric utilities
0         1939-01-01                            3.3336
1         1939-02-01                            3.3591
2         1939-03-01                            3.4354
3         1939-04-01                            3.4608
4         1939-05-01                            3.4608
..               ...                               ...
967       2019-08-01                          112.3547
968       2019-09-01                          103.1333
969       2019-10-01                           93.8206
970       2019-11-01                           99.9782
971       2019-12-01                          109.9263

[972 rows x 2 columns]
```

4.Plot the time series data with suitable labels and title. What do you conclude from the plotted data.

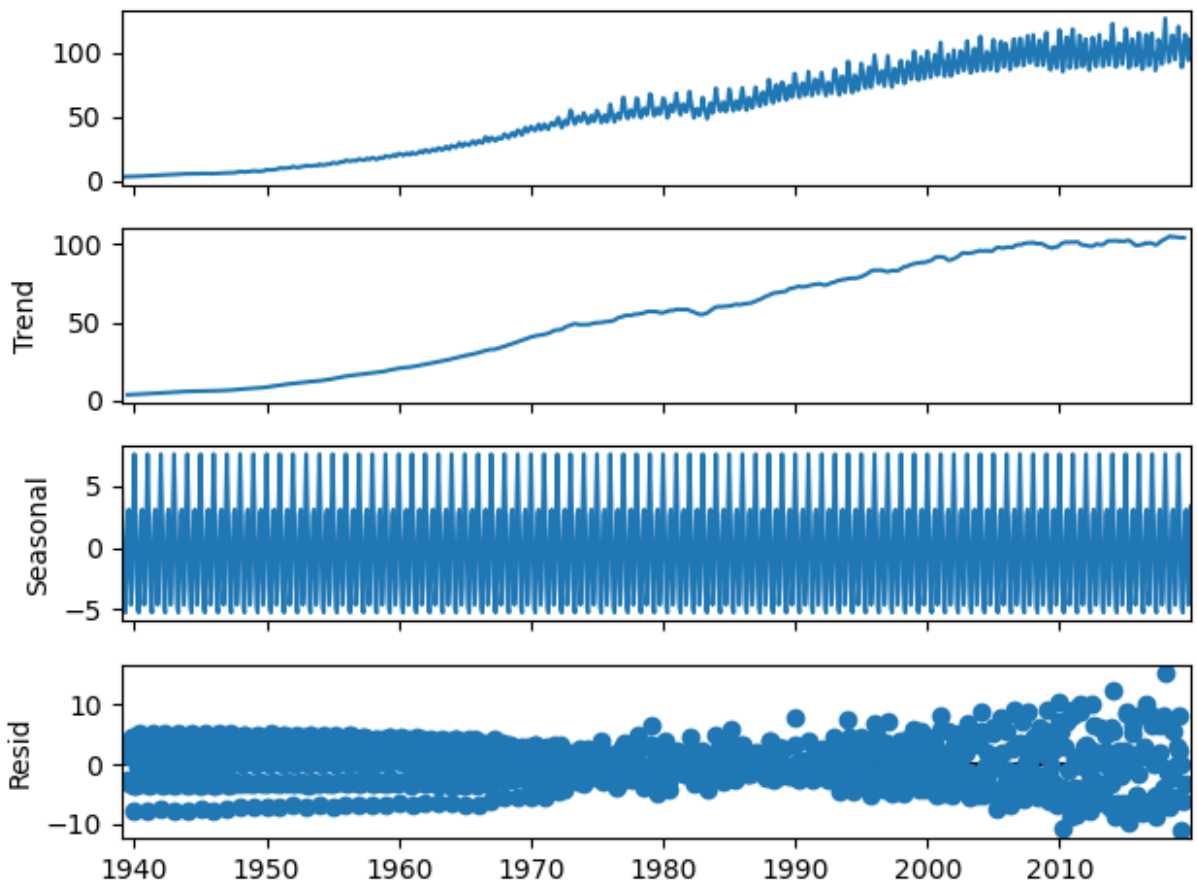
```
df.set_index("observation_date").plot(figsize=(12,6), title =
"Production over time" , xlabel = "Year", ylabel = "Electric
Utilities")
plt.show()
```



The time series data shows an upward trend and the production is increasing over time. The fluctuations imply seasonality. Over time the fluctuations become more pronounced, reflecting increasing variability.

5.Decompose the time series model using an appropriate model form with a valid explanation. Plot all the decomposed components of the chosen model.

```
decompose_df =
sm.tsa.seasonal_decompose(df.set_index("observation_date"),model =
"additive",period = 12)
decompose_df.plot()
plt.show()
```



6.Calculate the expected value and variance of the time series. Next, calculate the same for both for t+1. Is the data stationary?

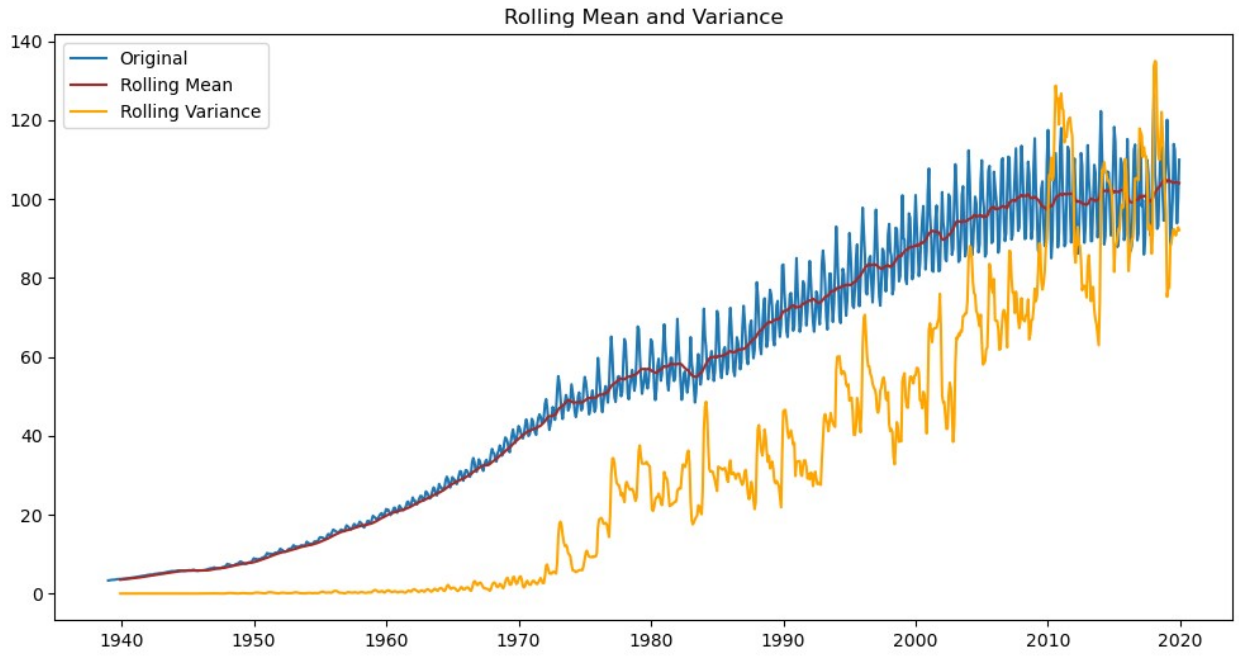
```
expected_value = df["Production of electric utilities"].mean()
variance_value = df["Production of electric utilities"].var()
print(f" Mean: {expected_value}")
print(f"Variance: {variance_value}")
```

```
 Mean: 54.003225925925925
Variance: 1224.499384082603
```

A time series is stationary if its statistical properties (mean, variance, autocovariance) do not change over time. If the mean changes over time or If variance fluctuates significantly, it indicates heteroscedasticity (non-stationarity). Autocorrelation measures how the current value is related to past values, but it doesn't directly confirm stationarity. Plotting these helps visually detect trends and seasonality, which are signs of non-stationarity.

```
rolling_mean = df["Production of electric
utilities"].rolling(window=12).mean()
rolling_var = df["Production of electric
utilities"].rolling(window=12).var()
```

```
plt.figure(figsize=(12,6))
plt.plot(df["observation_date"], df["Production of electric
utilities"], label="Original")
plt.plot(df["observation_date"], rolling_mean, label="Rolling Mean",
color='brown')
plt.plot(df["observation_date"], rolling_var, label="Rolling
Variance", color='orange')
plt.legend()
plt.title("Rolling Mean and Variance")
plt.show()
```



The mean is not constant; it shows an upward trend, indicating a non-stationary process. Here variance is increasing over time, which shows changing variance.

7.Calculate the autocovariance between the time series and its lag (t+1).

```
acf_values = acf(df["Production of electric utilities"], nlags=1,
fft=False)
autocov_t1 = acf_values[1] * variance_value
print("Autocovariance (t+1):", {autocov_t1})

Autocovariance (t+1): {1206.956149640936}
```

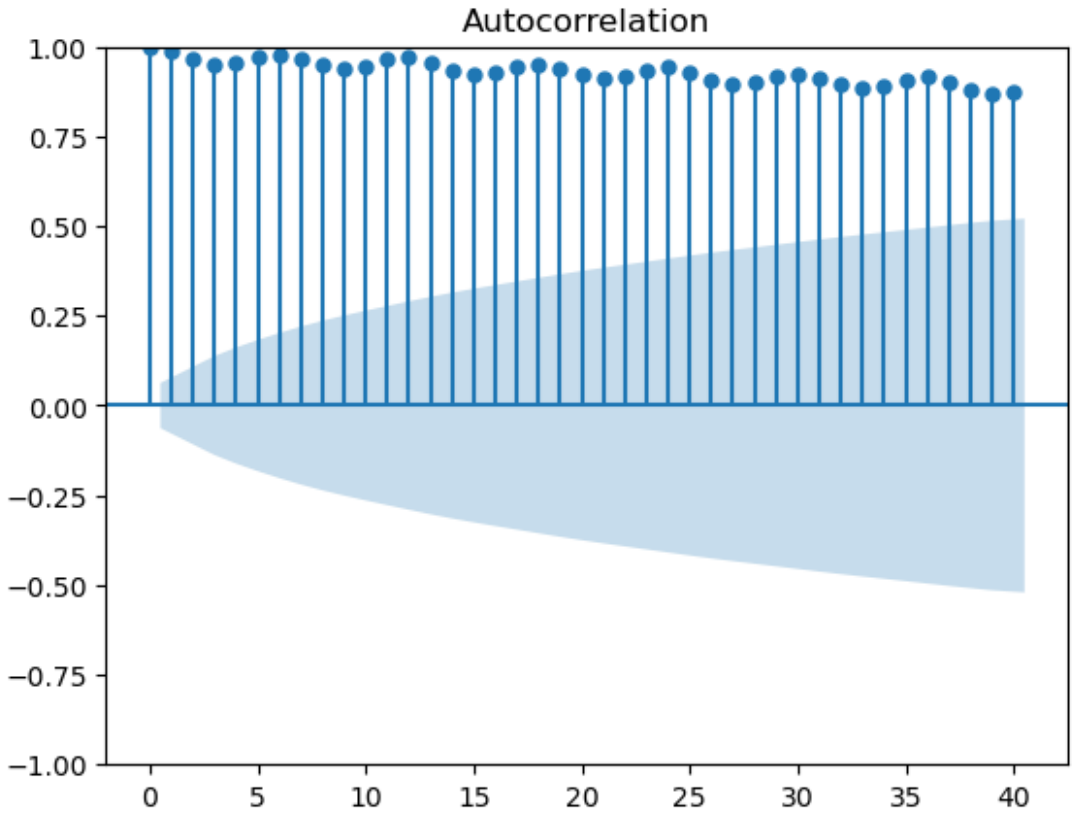
8.Calculate the autocorrelation between the time series and its lag (t+1).

```
autocorr_t1 = acf_values[1]
print("Autocorrelation (t+1):", {autocorr_t1})

Autocorrelation (t+1): {0.9856731373900931}
```

9. Create ACF plot. What do you observe from the plot.

```
sm.graphics.tsa.plot_acf(df["Production of electric utilities"],
lags=40)
plt.show()
```



The autocorrelation values are high for all lags, showing a strong connection between past and present values. Since the ACF does not drop quickly to zero, it indicates a trend in the data, confirming that the series is non-stationary. The repeated high correlations also suggest seasonal patterns, meaning electricity production follows a periodic cycle.