In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.impute import SimpleImputer

# Providing the path in drive
file_path = '/content/drive/MyDrive/Colab Notebooks/Advanced Data Analytics/AirP

# Loading the data
data = pd.read_csv(file_path)

# Display the first few rows of the dataset
data.head()
```

Out[ ]:

	date	value
0	1949-01-01	112
1	1949-02-01	118
2	1949-03-01	132
3	1949-04-01	129
4	1949-05-01	121

In [ ]:
```
# Exploratory analysis

# Display basic information about the dataset
data.info()

# Display statistical summary of the dataset
data.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144 entries, 0 to 143
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   date    144 non-null    object
 1   value   144 non-null    int64 
dtypes: int64(1), object(1)
memory usage: 2.4+ KB
```

Out[ ]:

	value
count	144.000000
mean	280.298611
std	119.966317
min	104.000000
25%	180.000000
50%	265.500000
75%	360.500000
max	622.000000

In [ ]:
```
# Check for missing values
missing_values = data.isnull().sum()
print(missing_values[missing_values > 0])

# Fill missing values with the mean for numerical columns and mode for categoric
for column in data.columns:
    if data[column].dtype == 'object':
        data[column].fillna(data[column].mode()[0], inplace=True)
    else:
        data[column].fillna(data[column].mean(), inplace=True)
```

```
Series([], dtype: int64)
```

In [ ]:
```
data.shape
```

Out[ ]:
```
(144, 2)
```

No Misssing Values exist

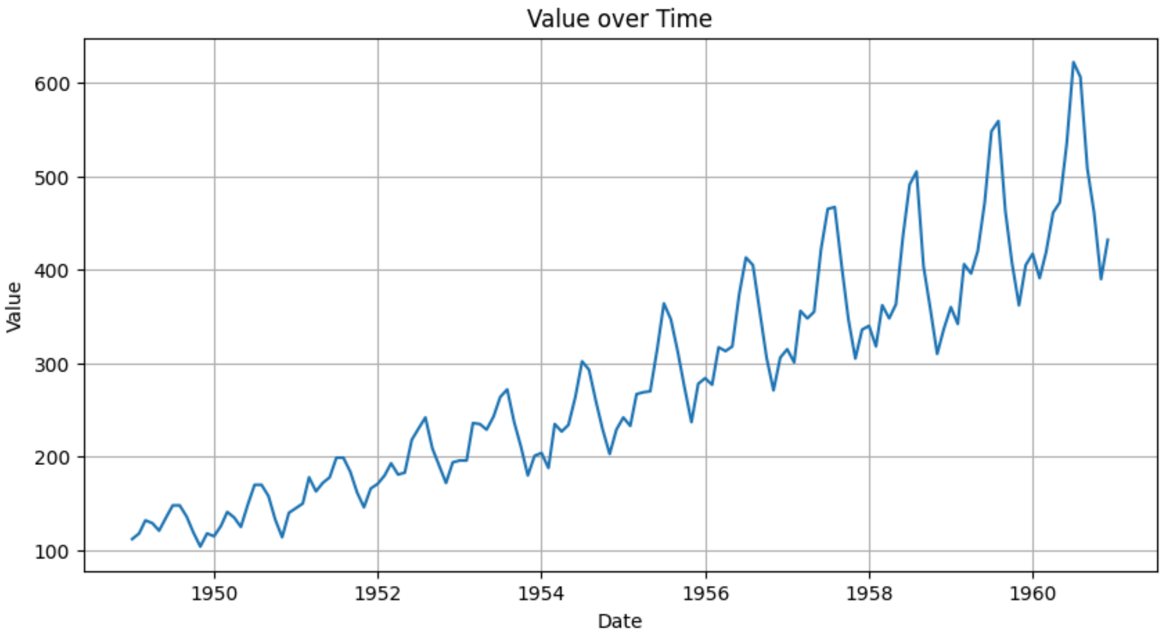
In [ ]:
```
data['date'] = pd.to_datetime(data['date'])
```

In [ ]:
```
data.set_index('date', inplace=True)
```

In [ ]:
```
data.info()
```

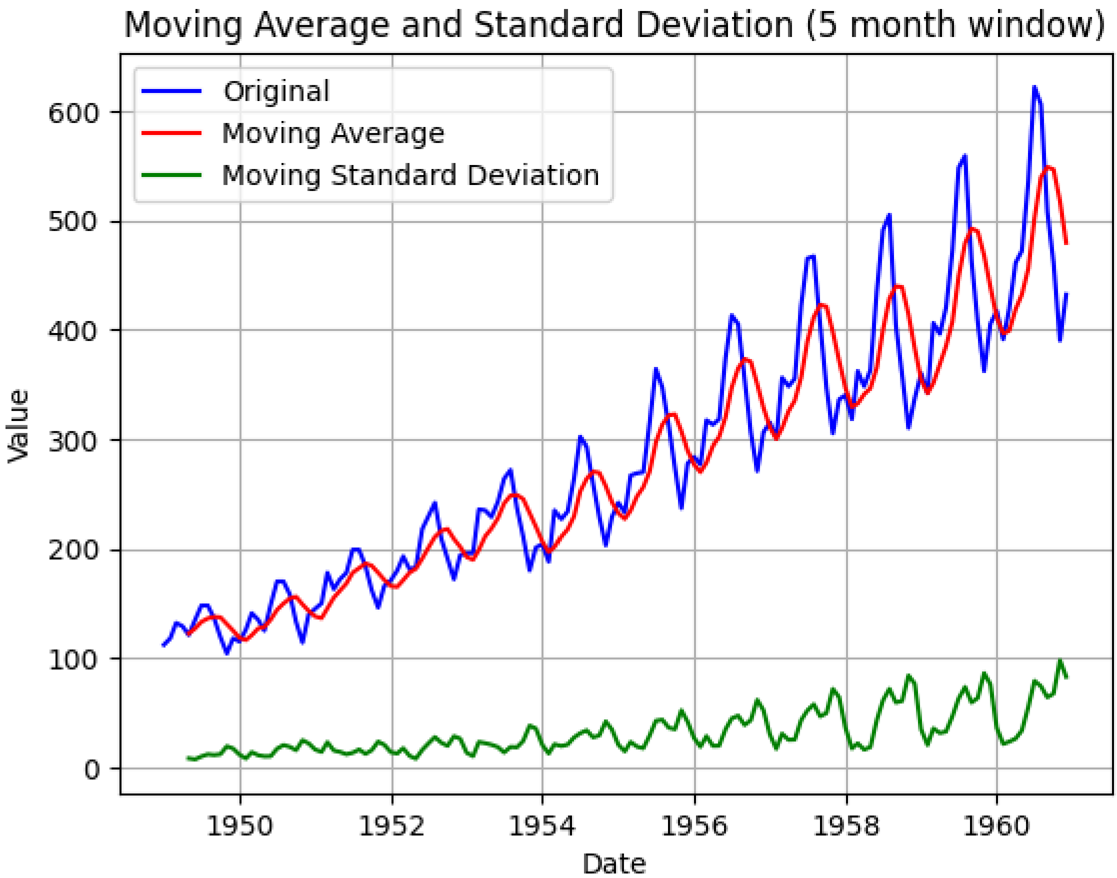
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 144 entries, 1949-01-01 to 1960-12-01
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   value   144 non-null    int64
dtypes: int64(1)
memory usage: 2.2 KB
```

In [ ]:
```
plt.figure(figsize=(10, 5))
plt.plot(data.index, data['value'])
plt.title('Value over Time')
plt.xlabel('Date')
plt.ylabel('Value')
plt.grid(True)
plt.show()
```



In [ ]:
```
# Rolling Statistic
moving_avg = data.rolling(window=5).mean()
moving_std = data.rolling(window=5).std()
```

In [ ]:
```
plt.plot(data.index, data['value'], label='Original',color="blue")
plt.plot(data.index, moving_avg, label='Moving Average',color="red")
plt.plot(data.index, moving_std, label='Moving Standard Deviation',color="green"
plt.title('Moving Average and Standard Deviation (5 month window)')
plt.xlabel('Date')
plt.ylabel('Value')
plt.legend()
plt.grid(True)
plt.show()
```



h0 - > Assume data is non stationary

h1 - > it is stationary

test statisc < critical value and p value < 0.05 - reject null

In [ ]:
```
# ADF Test
from statsmodels.tsa.stattools import adfuller

def adf_test(series):
    dftest = adfuller(series ,autolag ="AIC")
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags U
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)

adf_test(data['value'])
```

```
Test Statistic                   0.815369
p-value                          0.991880
#Lags Used                      13.000000
Number of Observations Used    130.000000
Critical Value (1%)             -3.481682
Critical Value (5%)             -2.884042
Critical Value (10%)            -2.578770
dtype: float64
```

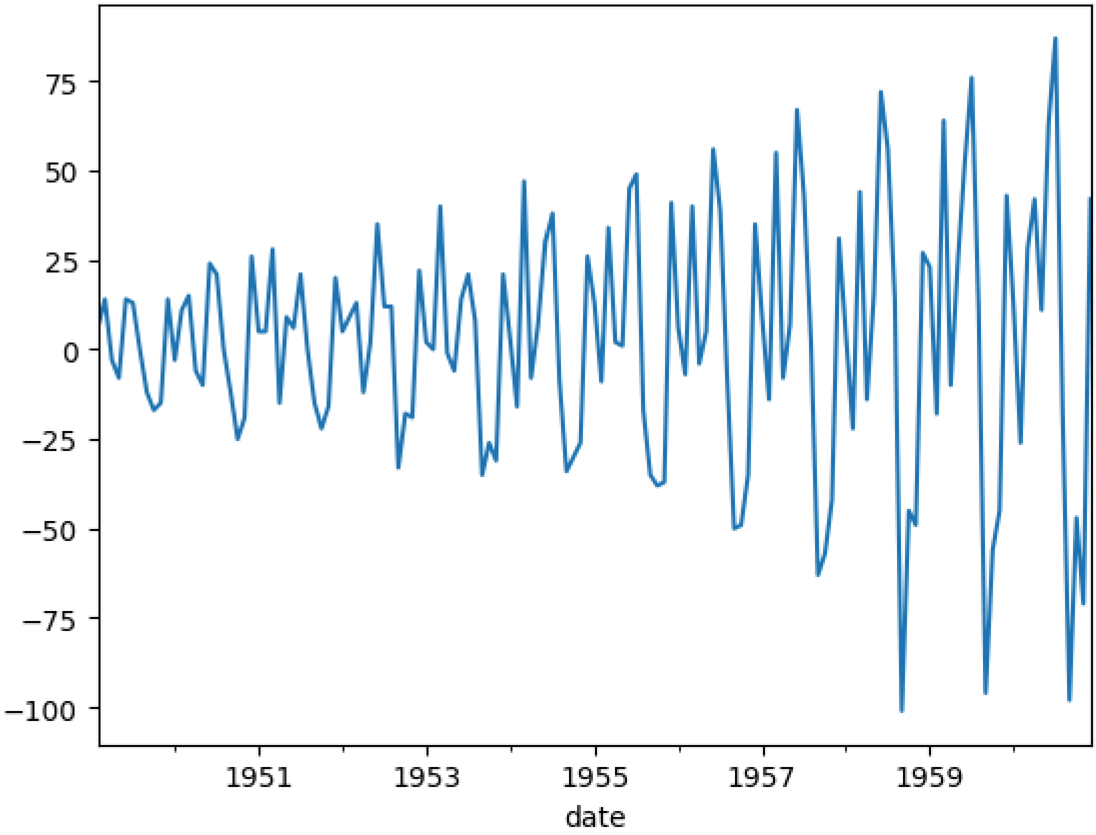
The test **statistic 0.8153** which is greater than all three critical values

and the **P value 0.991** is greater than 0.05

Hence We **accept the null hypothesis** H0 that the dataset is non Stationary

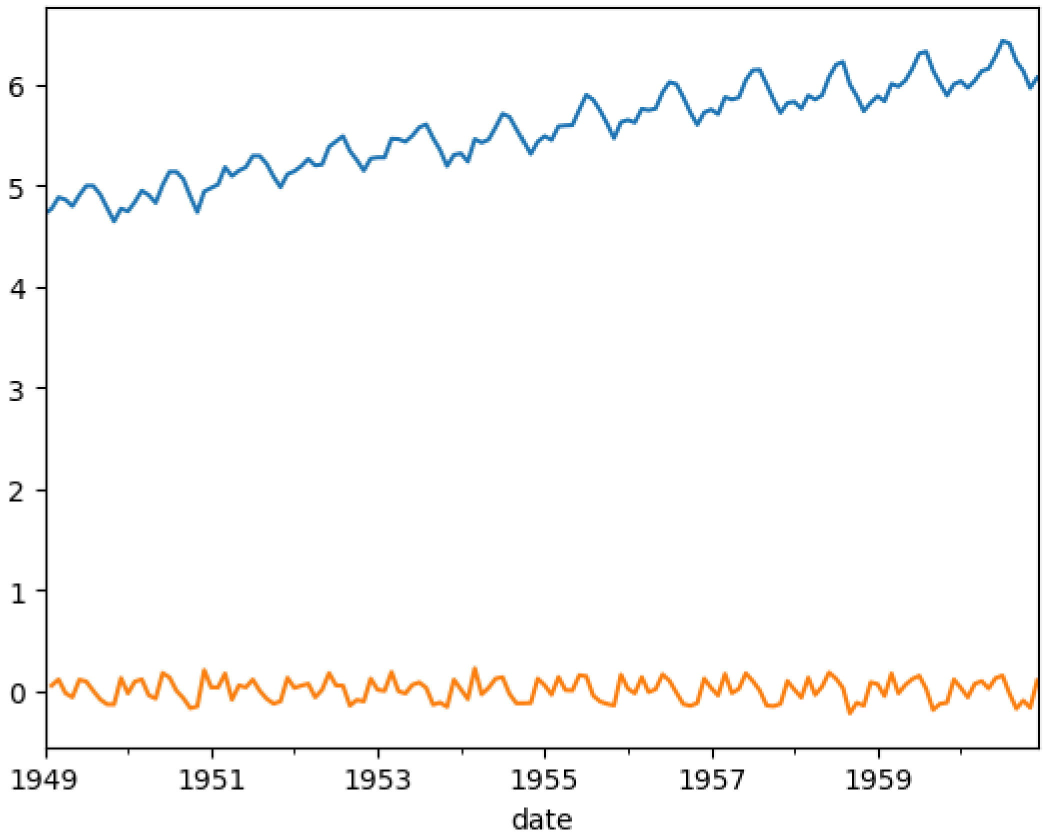
In [ ]:
```
# Making the Data Stationary
# Differencing
# Yt = Yt - Yt-1
# shift value you is given by the seasonality
shiftvalue = 1
data['value_diff'] = data['value']- data['value'].shift(shiftvalue)
data['value_diff'].dropna().plot()
```

Out[ ]: <Axes: xlabel='date'>



In [ ]:
```
# Transformation
# log transformation
data['value_log'] = np.log(data['value'])
data['value_log'].dropna().plot()
data['value_log_diff'] = data['value_log'] - data['value_log'].shift(shiftvalue)
data['value_log_diff'].dropna().plot()
```

Out[ ]: <Axes: xlabel='date'>



In [ ]:
```
data
```

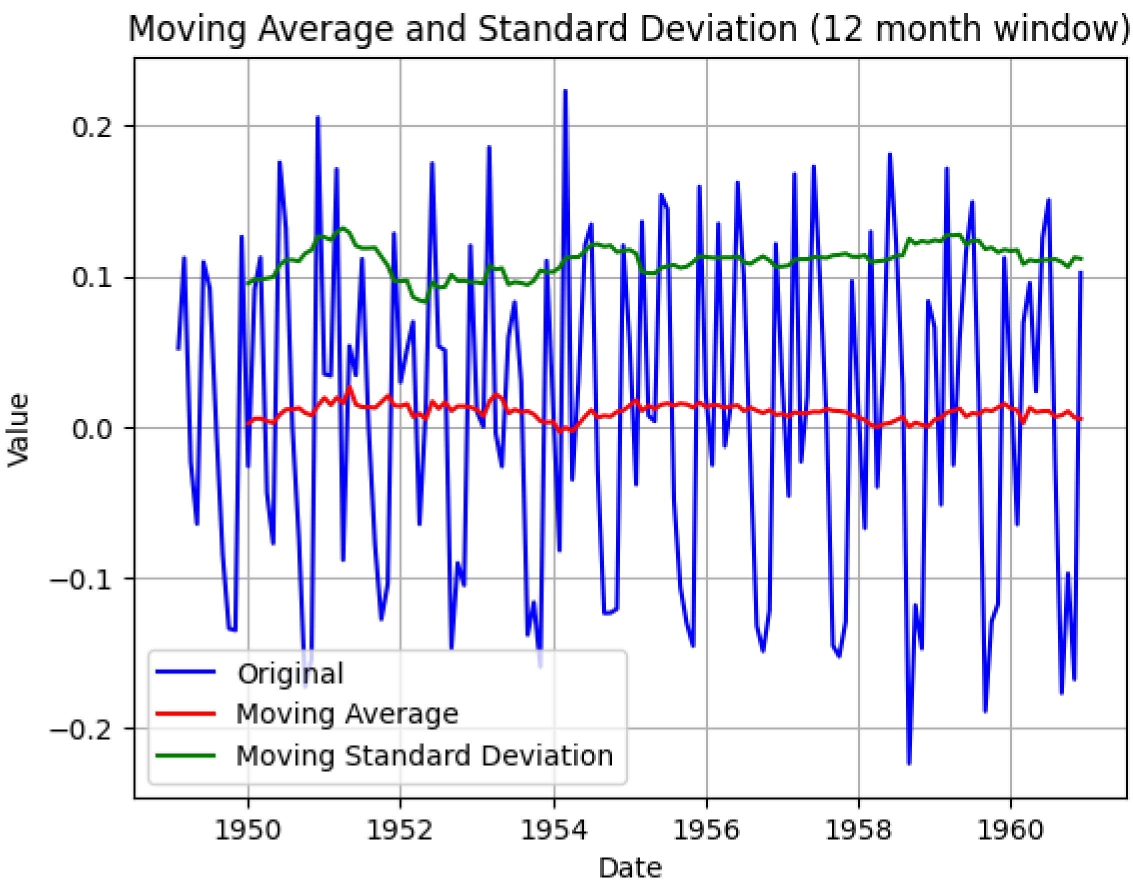
Out[ ]:

	value	value_diff	value_log	value_log_diff
date				
1949-01-01	112	NaN	4.718499	NaN
1949-02-01	118	6.0	4.770685	0.052186
1949-03-01	132	14.0	4.882802	0.112117
1949-04-01	129	-3.0	4.859812	-0.022990
1949-05-01	121	-8.0	4.795791	-0.064022
...	...	...	...	...
1960-08-01	606	-16.0	6.406880	-0.026060
1960-09-01	508	-98.0	6.230481	-0.176399
1960-10-01	461	-47.0	6.133398	-0.097083
1960-11-01	390	-71.0	5.966147	-0.167251
1960-12-01	432	42.0	6.068426	0.102279

144 rows × 4 columns

In [ ]:
```
moving_avg = data['value_log_diff'].rolling(window=12).mean()
moving_std = data['value_log_diff'].rolling(window=12).std()
```

```
In [ ]: plt.plot(data.index, data['value_log_diff'], label='Original',color="blue")
        plt.plot(data.index, moving_avg, label='Moving Average',color="red")
        plt.plot(data.index, moving_std, label='Moving Standard Deviation',color="green"
        plt.title('Moving Average and Standard Deviation (12 month window)')
        plt.xlabel('Date')
        plt.ylabel('Value')
        plt.legend()
        plt.grid(True)
        plt.show()
```



Additive Model = Trend + Seasonality + Residual

Multiplicative Model = Trend * Seasonality * Residual

Use Additive

When seasonality is consstant over time

Trend and seasonality are indipendent

Use Multiplicative

When seasonality Increases or Decreases over time

Trend and seasonality are dependent

from statsmodels.tsa.seasonal import seasonal_decompose

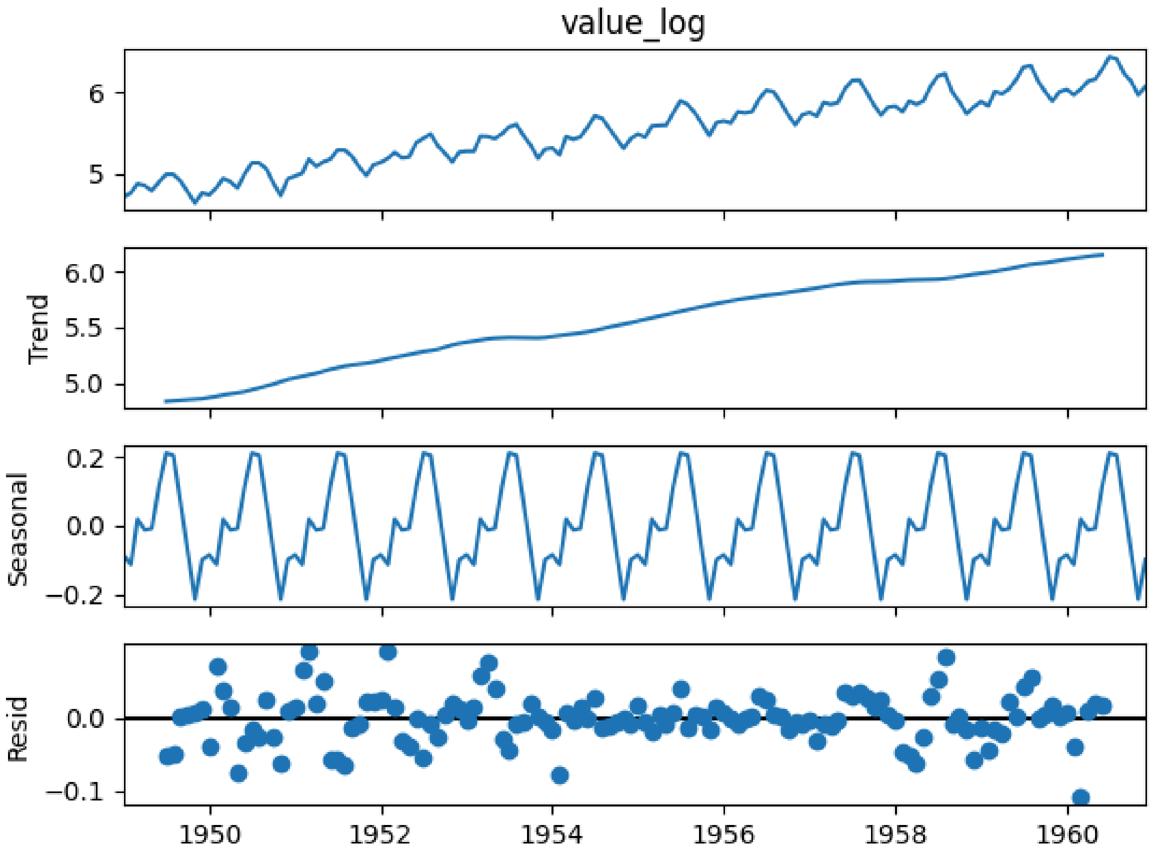
print("Additive") decomposition = seasonal_decompose(data['value_log'], model='additive') decomposition.plot() plt.show()

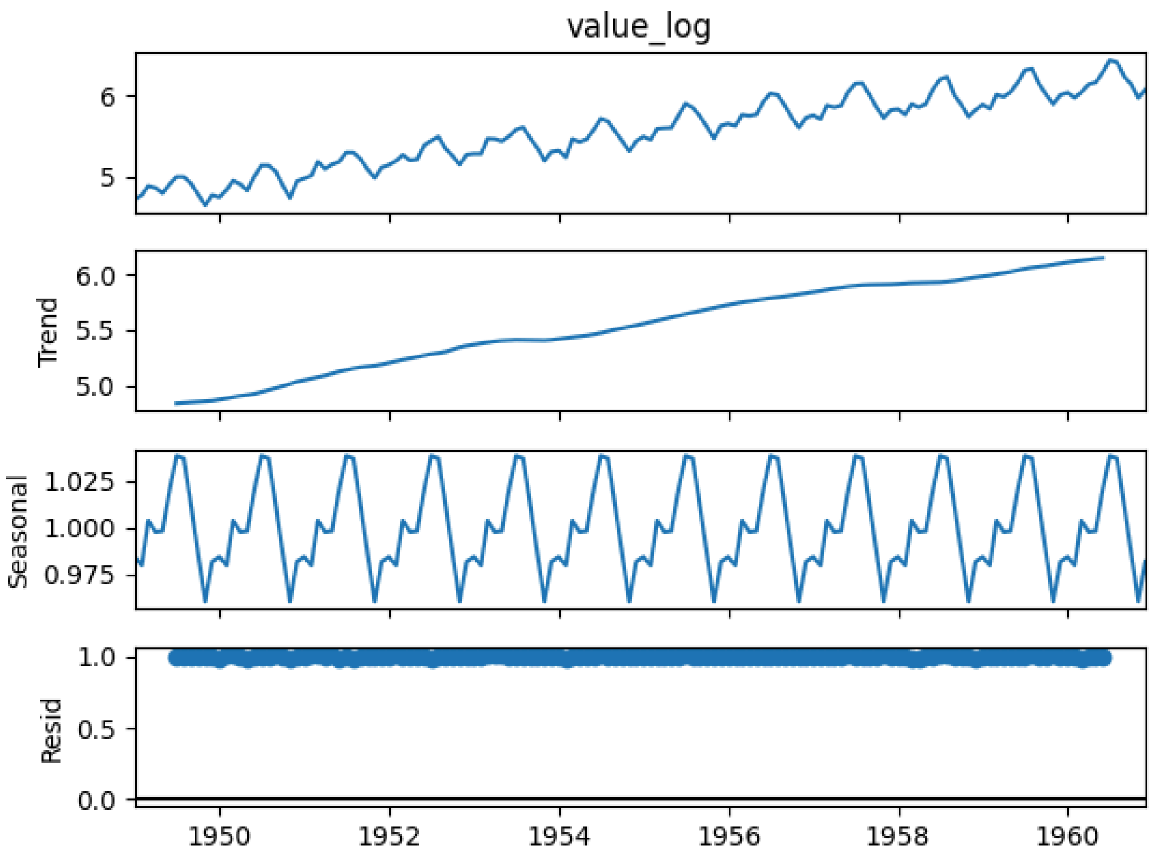
print("Multiplicative")

decomposition = seasonal_decompose(data['value_log'], model='multiplicative') decomposition.plot() plt.show()

In [ ]:

```
#Additive
from statsmodels.tsa.seasonal import seasonal_decompose
decompostion=seasonal_decompose(data['value_log'],model='additive')
decompostion.plot()
plt.show()
#multiplicative
from statsmodels.tsa.seasonal import seasonal_decompose
decompostion=seasonal_decompose(data['value_log'],model='multiplicative')
decompostion.plot()
plt.show()
```

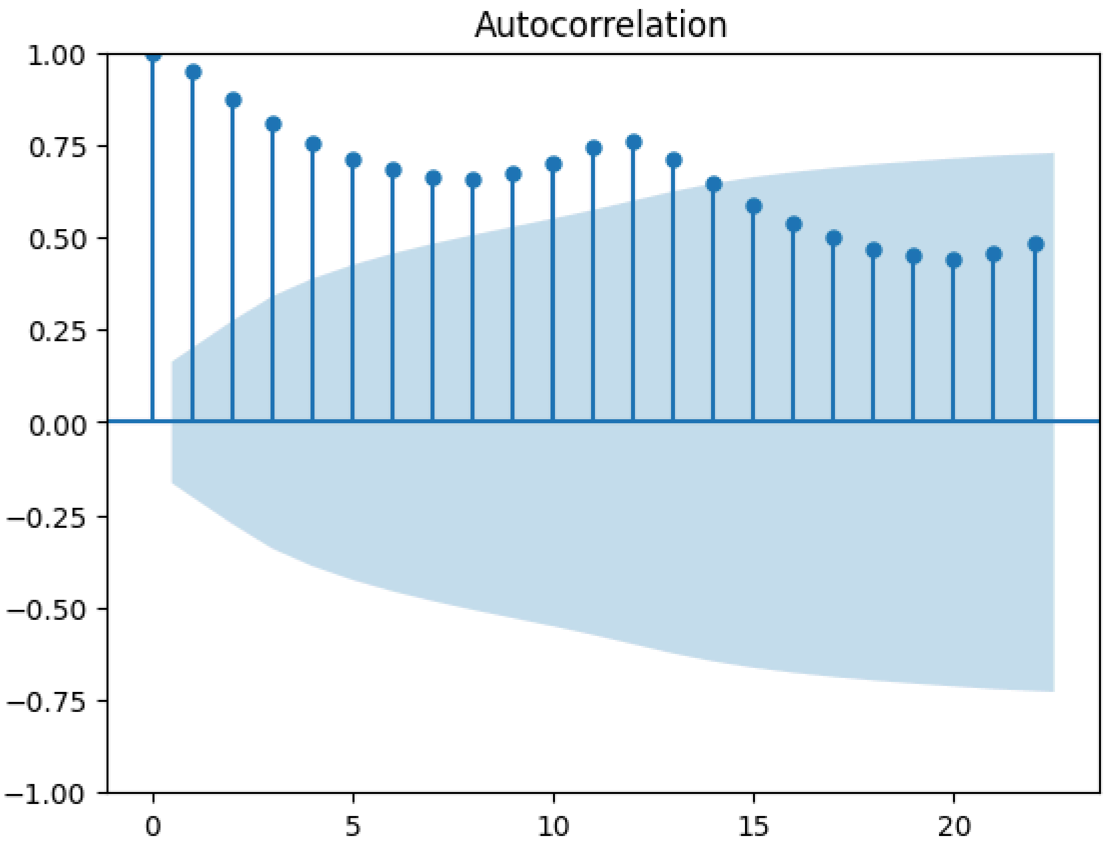


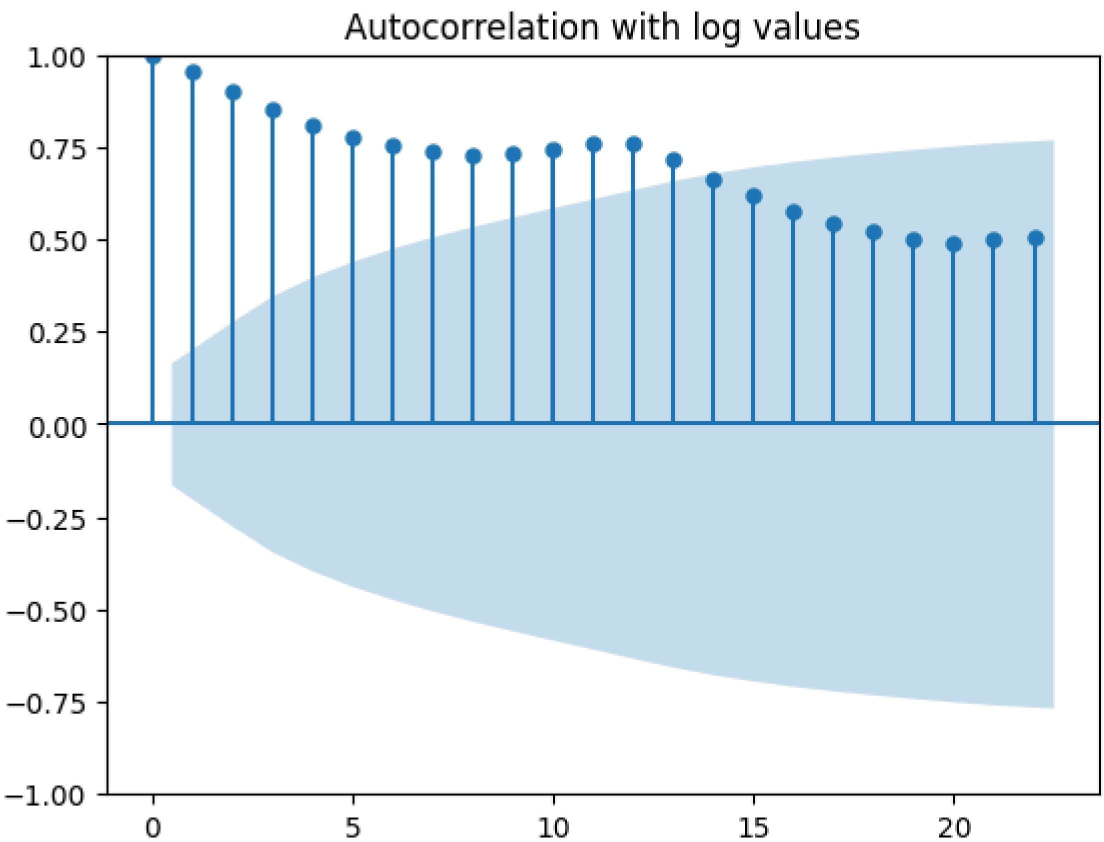


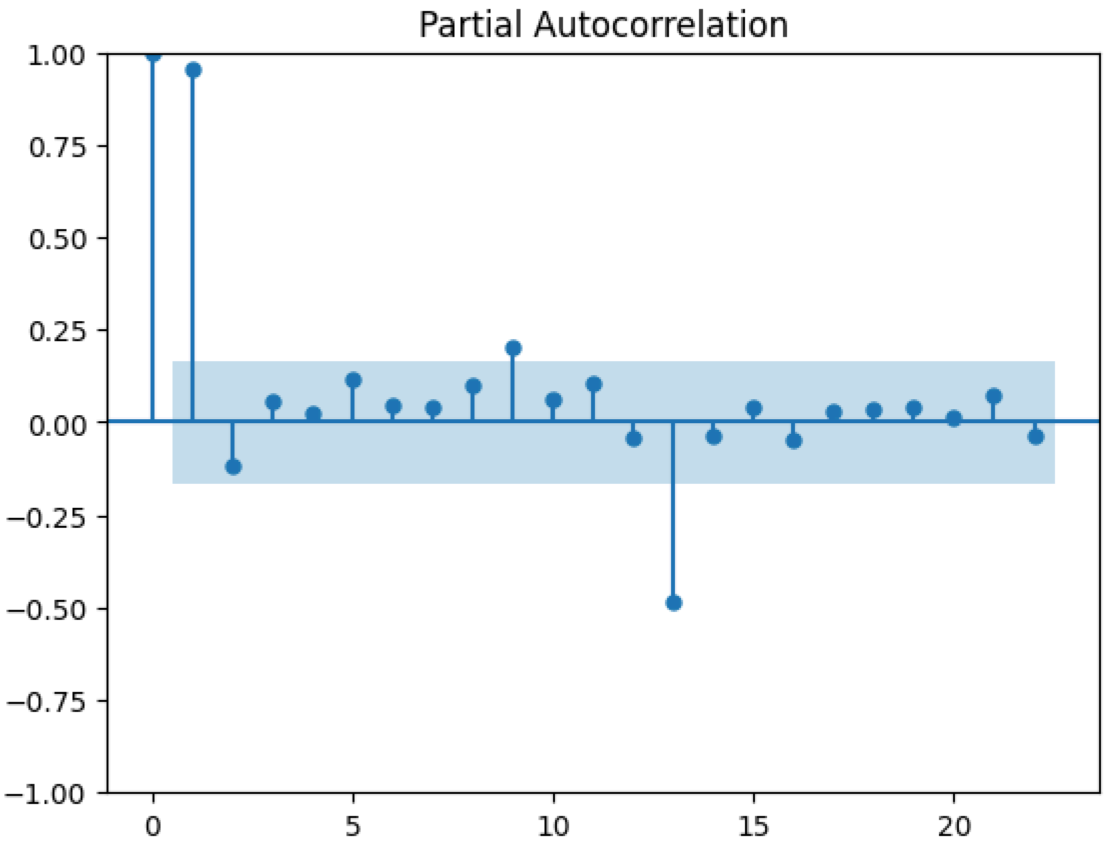
In [ ]:

```
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
plot_acf(data['value'])
plt.show()

plot_acf(data['value_log'])
plt.title("Autocorrelation with log values")
```

```
plt.show()

plot_pacf(data['value_log'])
plt.show()
```



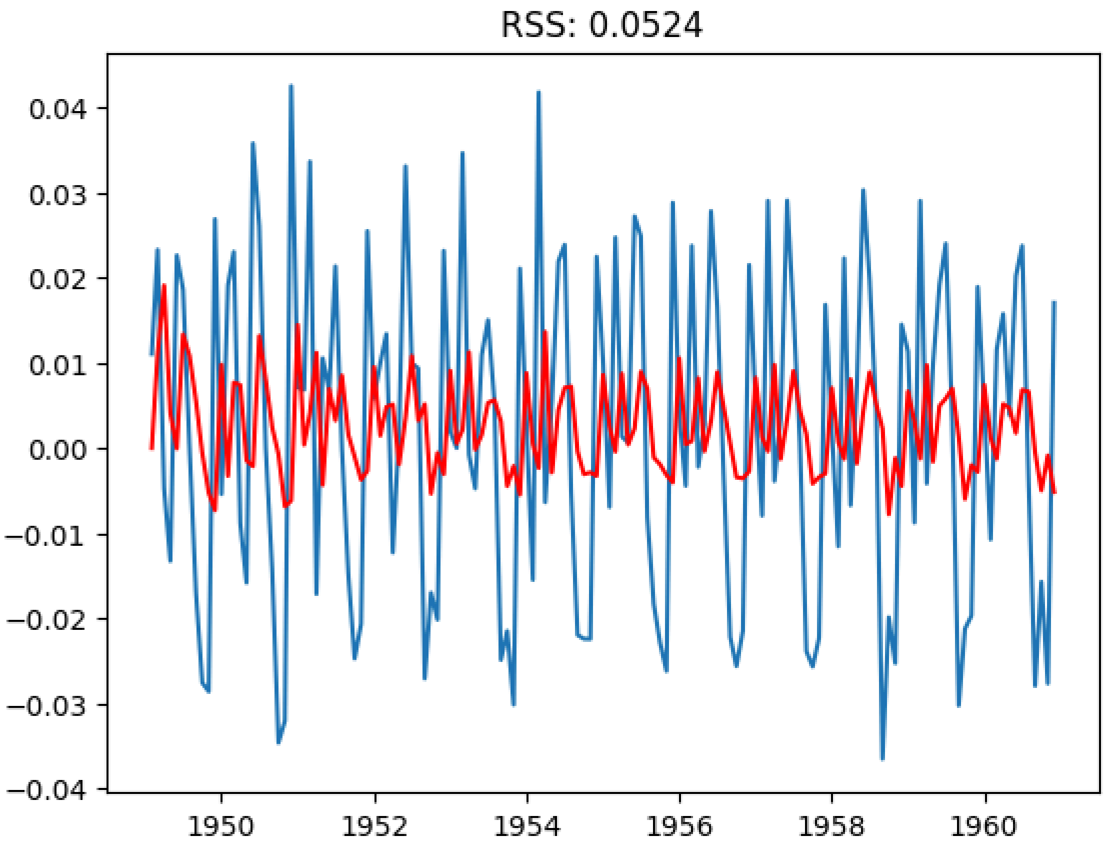




In [ ]:

```
#ARIMA
from statsmodels.tsa.arima.model import ARIMA
ts_log=np.log(data['value_log']).diff().dropna()
model1=ARIMA(ts_log,order=(0,1,2))
results_AR=model1.fit()
plt.plot(ts_log)
plt.plot(results_AR.fittedvalues,color='red')
plt.title('RSS: %.4f'% sum((results_AR.fittedvalues-ts_log)**2))
print('Plotting AR model')
```

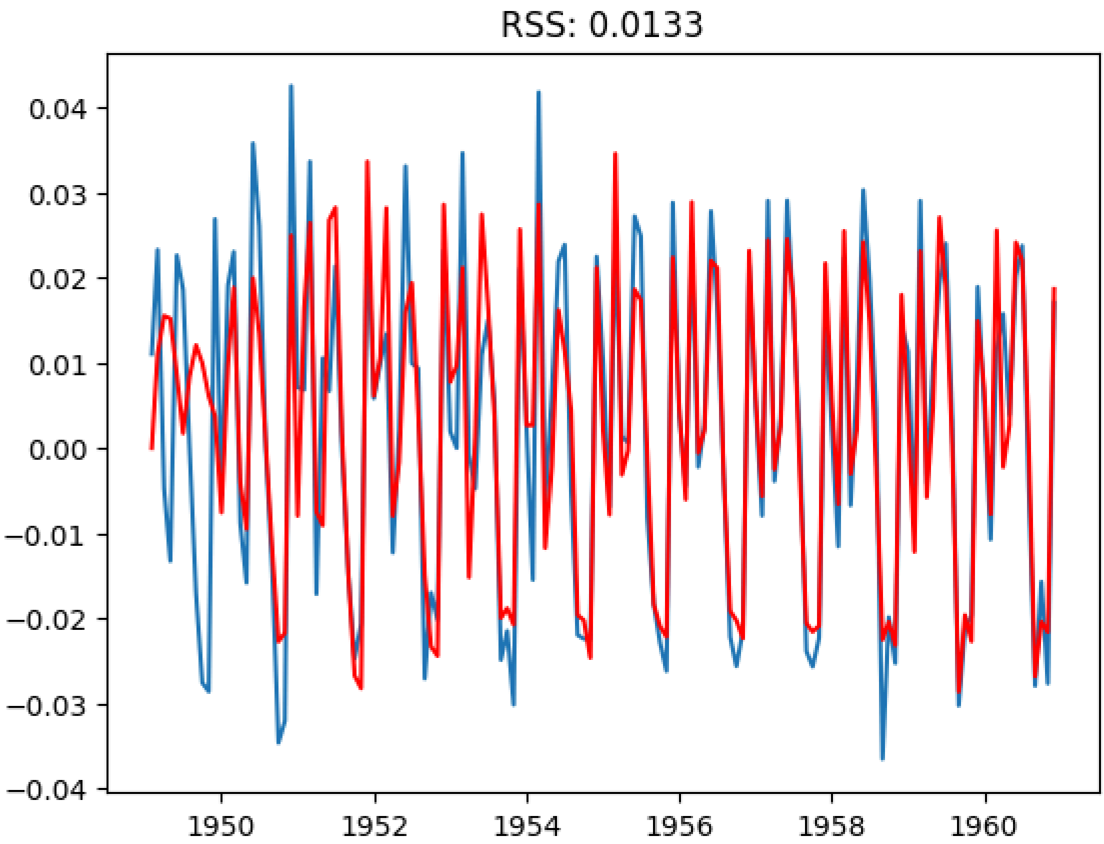
```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
Plotting AR model
```



In [ ]:
```
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX

# SARIMA model
sarima_model = SARIMAX(ts_log, order=(2,1,3), seasonal_order=(1,0,2,12))
results_SAR = sarima_model.fit()

print('Plotting SARIMA model')
plt.plot(ts_log)
plt.plot(results_SAR.fittedvalues, color='red')
plt.title('RSS: %.4f' % sum((results_SAR.fittedvalues - ts_log) ** 2))
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
Plotting SARIMA model
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

Out[ ]: `Text(0.5, 1.0, 'RSS: 0.0133')`



In [ ]:
```
# Forecast the next 12 months for both models
forecast_ARIMA = results_AR.forecast(steps=12)
forecast_SARIMA = results_SAR.forecast(steps=12)

# Convert the forecasted values back to their original scale
# forecast_ARIMA = np.exp(forecast_ARIMA)
# forecast_SARIMA = np.exp(forecast_SARIMA)
```

In [ ]:
```
# Create a DataFrame for the forecasted values
forecast_df = pd.DataFrame({
    'ARIMA Forecast': forecast_ARIMA,
    'SARIMA Forecast': forecast_SARIMA,
})

print(forecast_df)
```

	ARIMA Forecast	SARIMA Forecast
1961-01-01	0.007410	0.005366
1961-02-01	0.001805	-0.009825
1961-03-01	0.001805	0.017756
1961-04-01	0.001805	0.005608
1961-05-01	0.001805	0.005578
1961-06-01	0.001805	0.019712
1961-07-01	0.001805	0.021324
1961-08-01	0.001805	-0.000828
1961-09-01	0.001805	-0.027556
1961-10-01	0.001805	-0.017130
1961-11-01	0.001805	-0.023231
1961-12-01	0.001805	0.016047

In [ ]:

Strategic Business Insights

Based on the forecasted values, discuss potential implications for the airline's capacity planning like fleet management, optimize ticket pricing, marketing and promotions, and resource allocation.

1. Airline Vehicle Management:

Based on the expected change and number of customers they can increase the number of planes to occupy incoming custmers

2. Ticket Pricing :

- Using the forcasted value to understand the reach of the company and make pricing decisions

3. Marketing and Advertisement:

- Target marketing campaigns in the coming years with the expectation that new customers will come. Add more locations to flight travel and include vacation plans

4. Resource Allocation:

- Allocate resources such as staff, ground crew, and airport facilities based on the forecasted passenger traffic understanding peak months. .

Forcasted value gives us an insight into the future and help us to make decision which are backed by data and is the most probable outcome