Experiment 6: Time Series

Aim:
To use time series analysis on a dataset and test the accuracy of the model.

Problem Statement:
Choose a classification dataset of your choice from any of the following Repository Links, download it:

1. Kaggle: https://www.kaggle.com/
2. UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/index.php

Perform Linear Regression on the chosen dataset.

Your notebook should contain:

1. Basic EDA

[***Hint***: Follow the steps in Titanic notebook uploaded on moodle under Expt 3 reference material]

Tool/Language:
Programming language: Python
Libraries: numpy, pandas, sklearn, matplotlib, seaborn

Code with visualisation graphs:

1) **Dataset Chosen:** Air Passengers
2) **Dataset Description:** Air Passengers per month. Workshop dataset.
3) **Code:**

```
from datetime import datetime
import numpy as np
import pandas as pd
import matplotlib.pylab as ply
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import acf, pacf
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima_model import ARIMA
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 10, 6
import io
plt.style.use('ggplot')

# Reading the dataset
df = pd.read_csv(io.BytesIO(uploaded['AirPassengers.csv']))
df.head()
```

	Month	#Passengers
0	1949-01	112
1	1949-02	118
2	1949-03	132
3	1949-04	129
4	1949-05	121

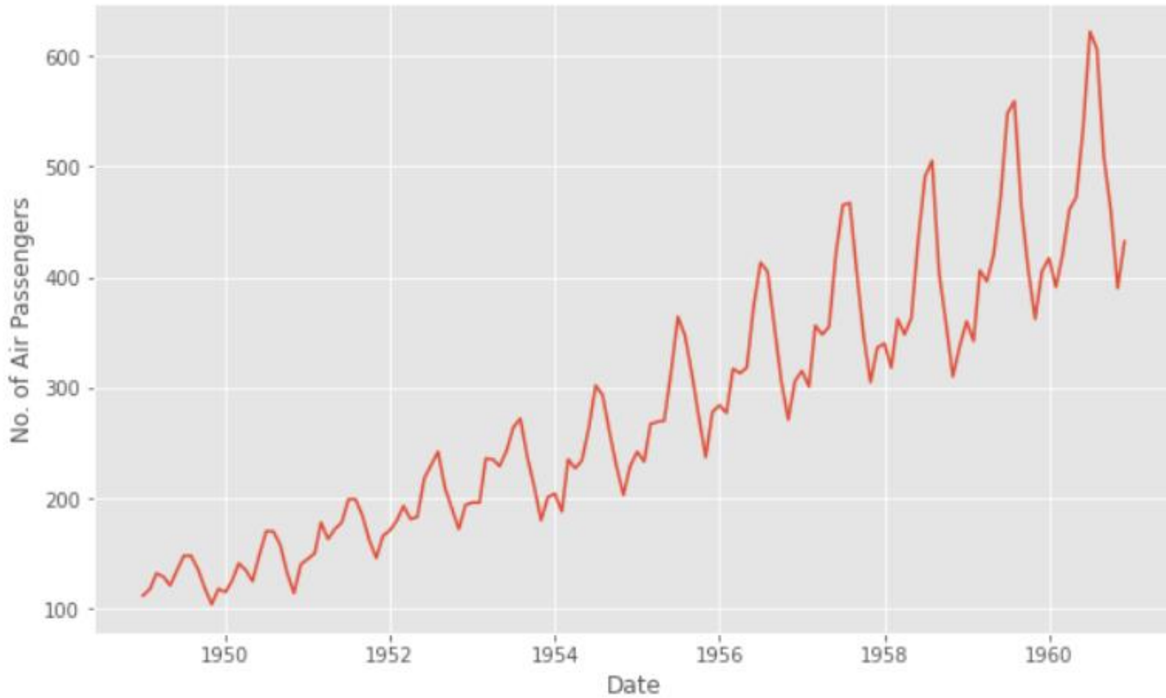
```
df.info()
```

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

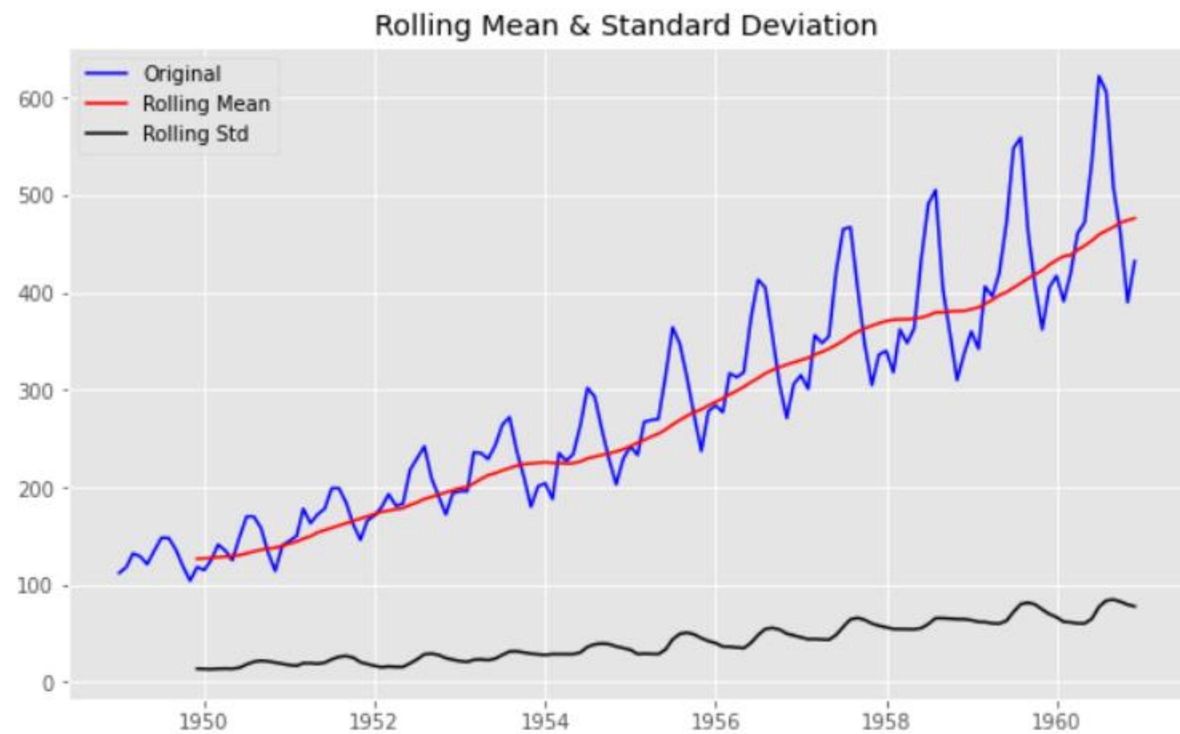
```
df['Month'] = pd.to_datetime(df['Month'],infer_datetime_format=True) # Convert fr
om String to Datetime
indexedDataset = df.set_index(['Month'])
indexedDataset.head()
```

	#Passengers
Month	
1949-01-01	112
1949-02-01	118
1949-03-01	132
1949-04-01	129
1949-05-01	121

```
plt.xlabel('Date')
plt.ylabel('No. of Air Passengers')
plt.plot(indexedDataset)
```




```
"""# **Rolling Mean & Standard Deviation**"""

# Determine Rolling Statistics
rolmean = indexedDataset.rolling(window=12).mean() # Window Size 12 denotes 12 mo
nths
rolstd = indexedDataset.rolling(window=12).std()
print(rolmean,rolstd)

# Plot Rolling Statistics
orig = plt.plot(indexedDataset, color='blue', label='Original')
mean = plt.plot(rolmean, color='red', label='Rolling Mean')
std = plt.plot(rolstd, color='black', label='Rolling Std')
plt.legend(loc='best')
plt.title('Rolling Mean & Standard Deviation')
plt.show(block=False)
```

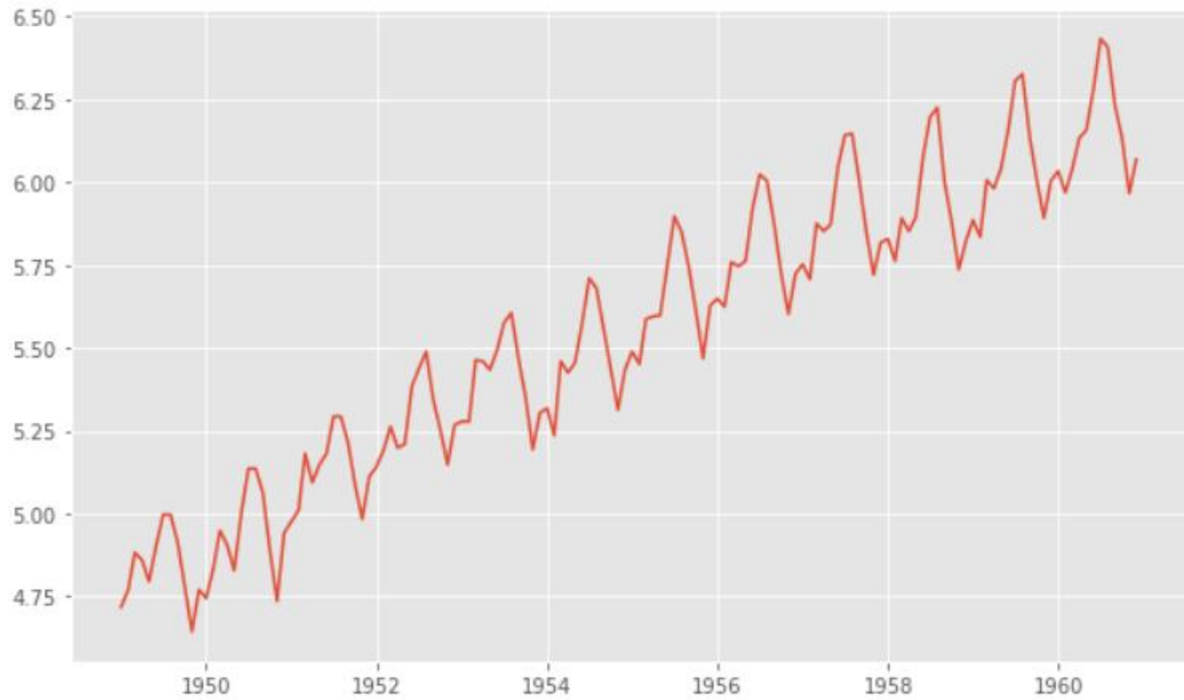



```
"""# **Augmented Dickey-Fuller Test**"""

# Perform Augmented Dickey-Fuller test:
print('Results of Dickey Fuller Test:')
dftest = adfuller(indexedDataset['#Passengers'], autolag='AIC')

dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-
value','#Lags Used','Number of Observations Used'])
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value

print(dfoutput)
```

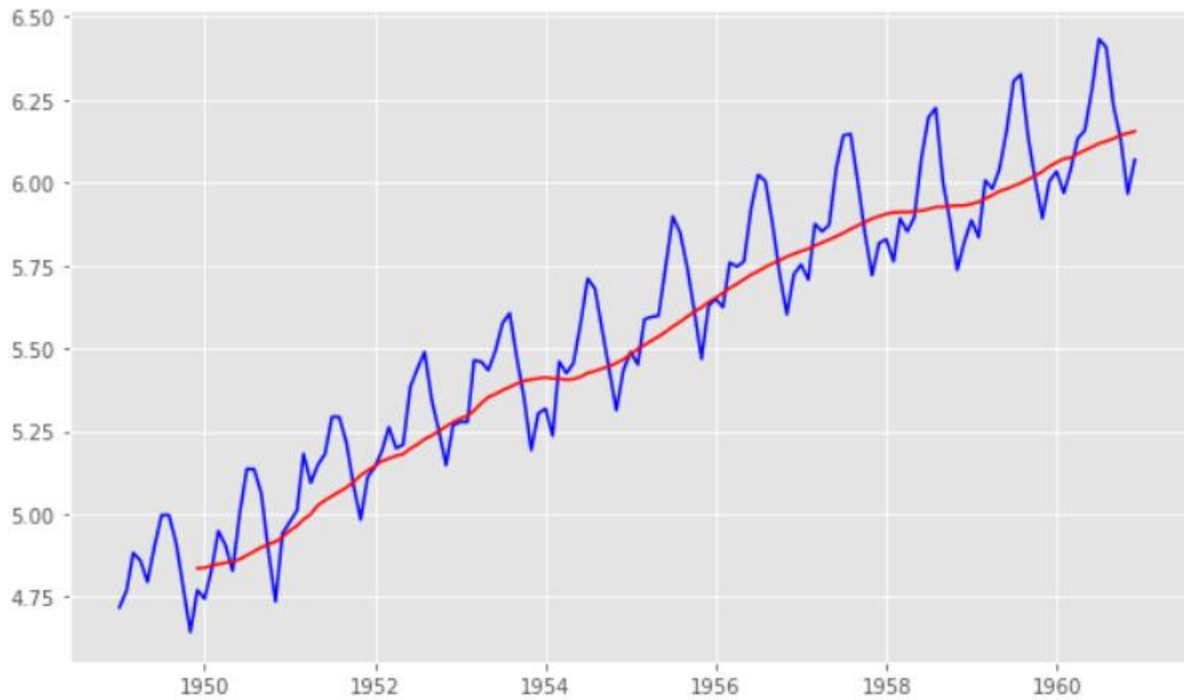
```
Results of Dickey Fuller Test:
Test Statistic                   0.815369
p-value                          0.991880
#Lags Used                      13.000000
Number of Observations Used    130.000000
Critical Value (1%)             -3.481682
Critical Value (5%)             -2.884042
Critical Value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

```
"""# **Log Scale Transformation**"""
```

```
# Estimating Trend
indexedDataset_logScale = np.log(indexedDataset)
plt.plot(indexedDataset_logScale)
```

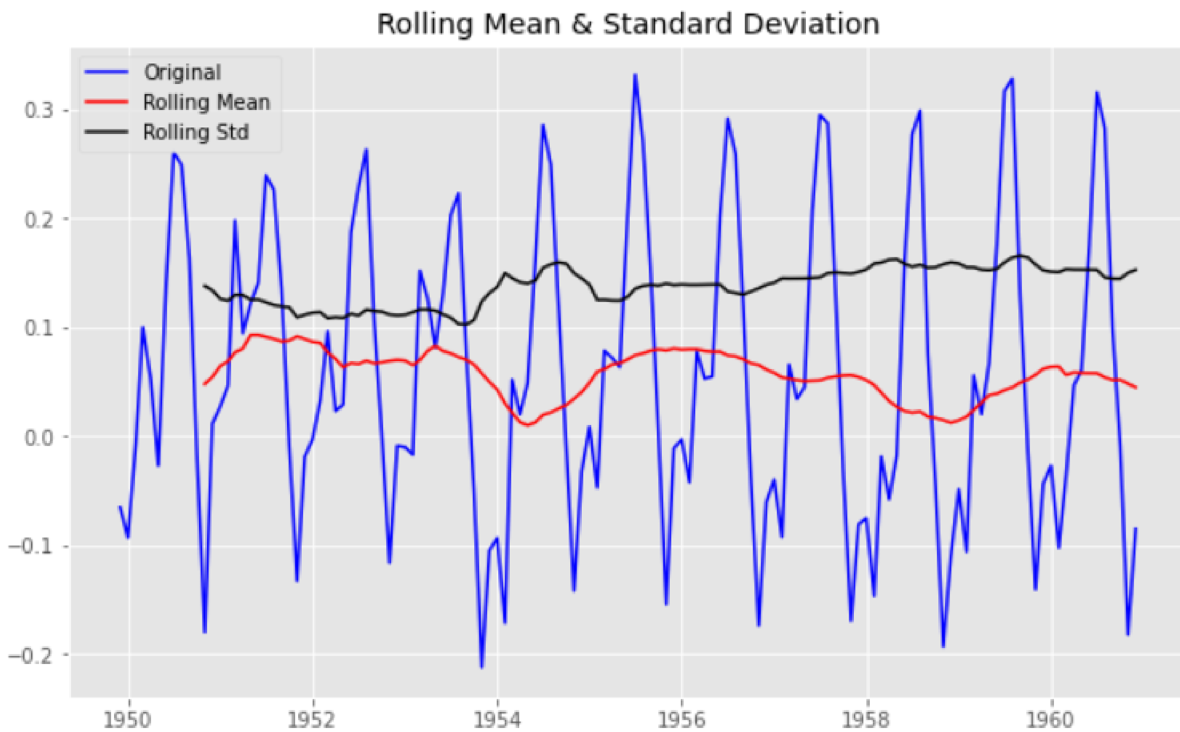



```
# Stationery Series
movingAverage = indexedDataset_logScale.rolling(window=12).mean()
movingSTD = indexedDataset_logScale.rolling(window=12).std()
plt.plot(indexedDataset_logScale, color='blue')
plt.plot(movingAverage, color='red')
```



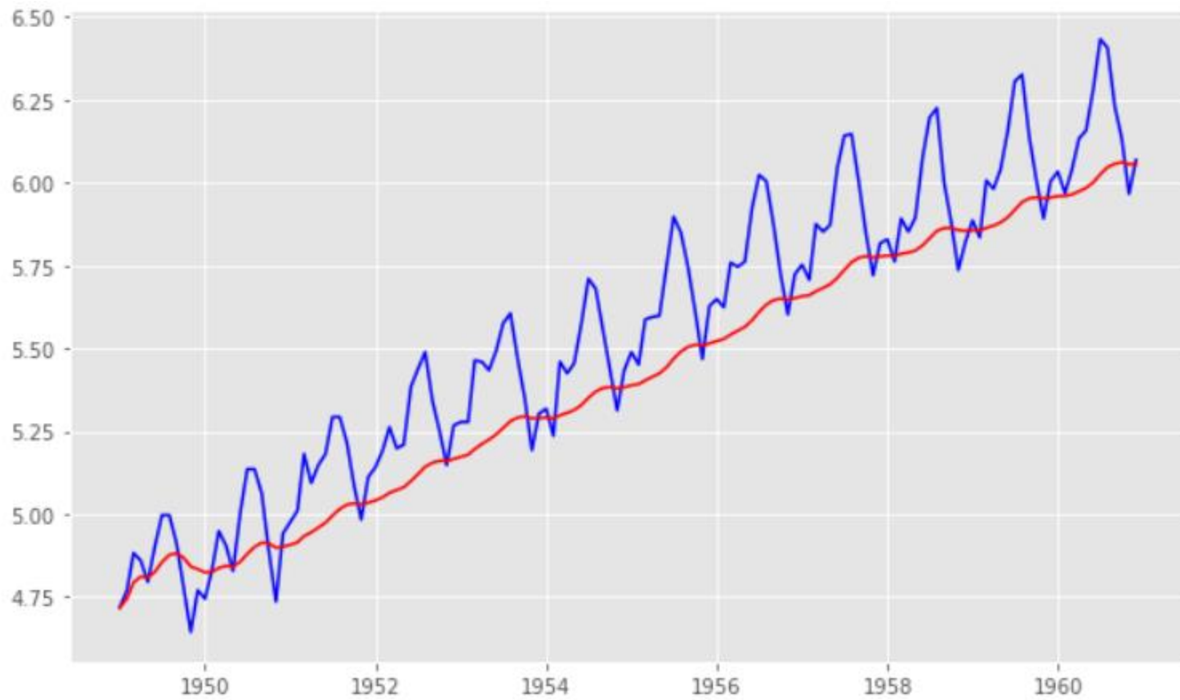

```
datasetLogScaleMinusMovingAverage = indexedDataset_logScale - movingAverage
datasetLogScaleMinusMovingAverage.head(12)

# Remove NaN values
datasetLogScaleMinusMovingAverage.dropna(inplace=True)
datasetLogScaleMinusMovingAverage.head(10)

def test_stationarity(timeseries):
    # Determine Rolling Statistics
    movingAverage = timeseries.rolling(window=12).mean()
    movingSTD = timeseries.rolling(window=12).std()

    # Plot Rolling Statistics
    orig = plt.plot(timeseries, color='blue', label='Original')
    mean = plt.plot(movingAverage, color='red', label='Rolling Mean')
    std = plt.plot(movingSTD, color='black', label='Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)

    # Perform Dickey-Fuller test:
    print('Results of Dickey Fuller Test:')
    dftest = adfuller(timeseries['#Passengers'], autolag='AIC')
```

```
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)

test_stationarity(datasetLogScaleMinusMovingAverage)
```

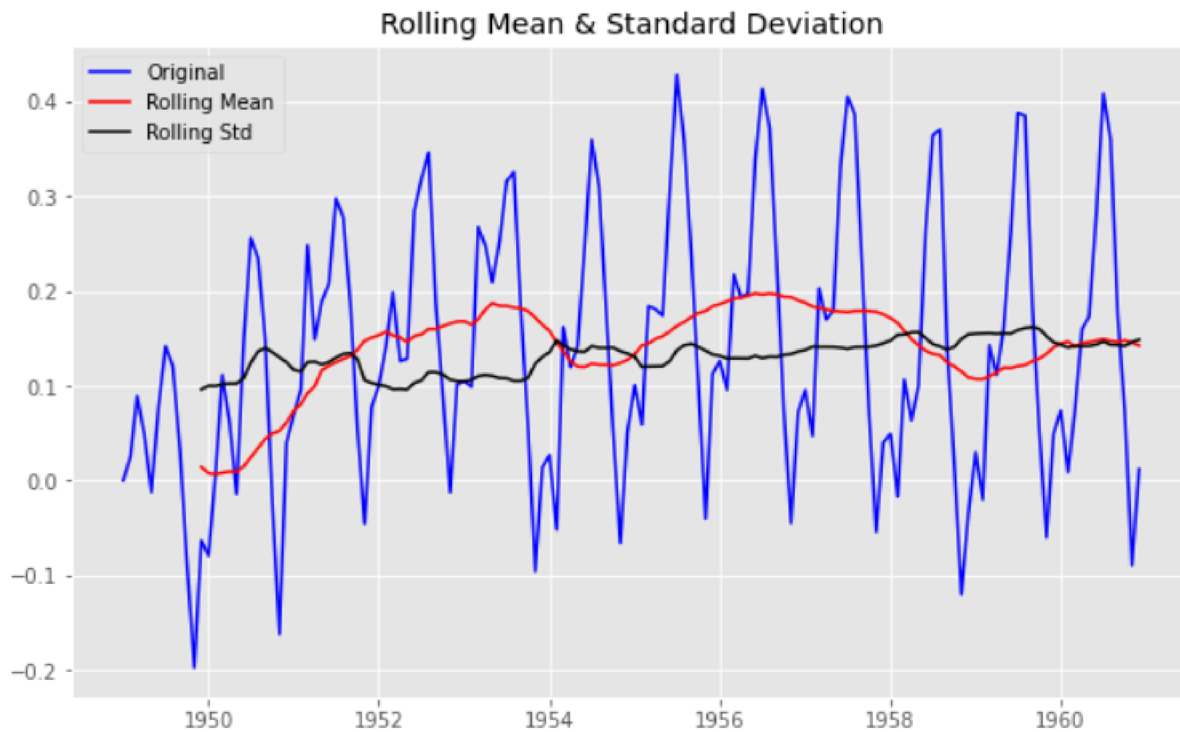



```
Results of Dickey Fuller Test:
Test Statistic                  -3.162908
p-value                          0.022235
#Lags Used                      13.000000
Number of Observations Used    119.000000
Critical Value (1%)             -3.486535
Critical Value (5%)             -2.886151
```

```
"""# **Exponential Decay Transformation**"""

exponentialDecayWeightedAverage = indexedDataset_logScale.ewm(halflife=12, min_periods=0, adjust=True).mean()
plt.plot(indexedDataset_logScale, color='blue')
plt.plot(exponentialDecayWeightedAverage, color='red')
```

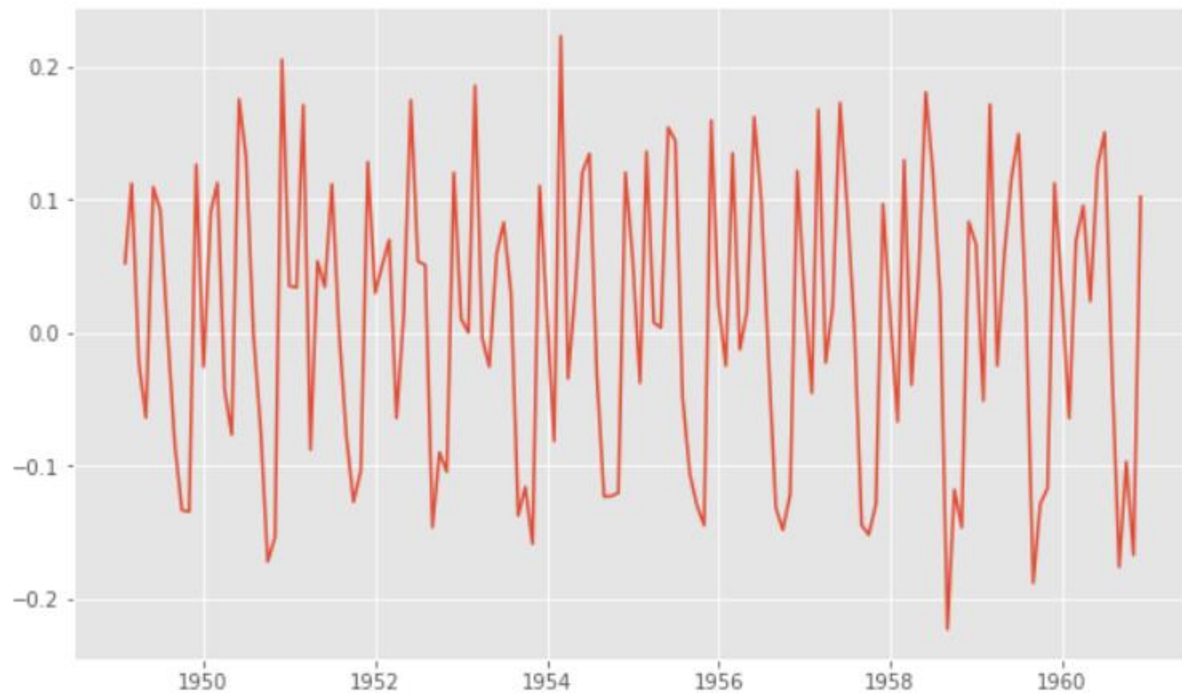



```
datasetLogScaleMinusExponentialMovingAverage = indexedDataset_logScale - exponentialDecayWeightedAverage
test_stationarity(datasetLogScaleMinusExponentialMovingAverage)
```

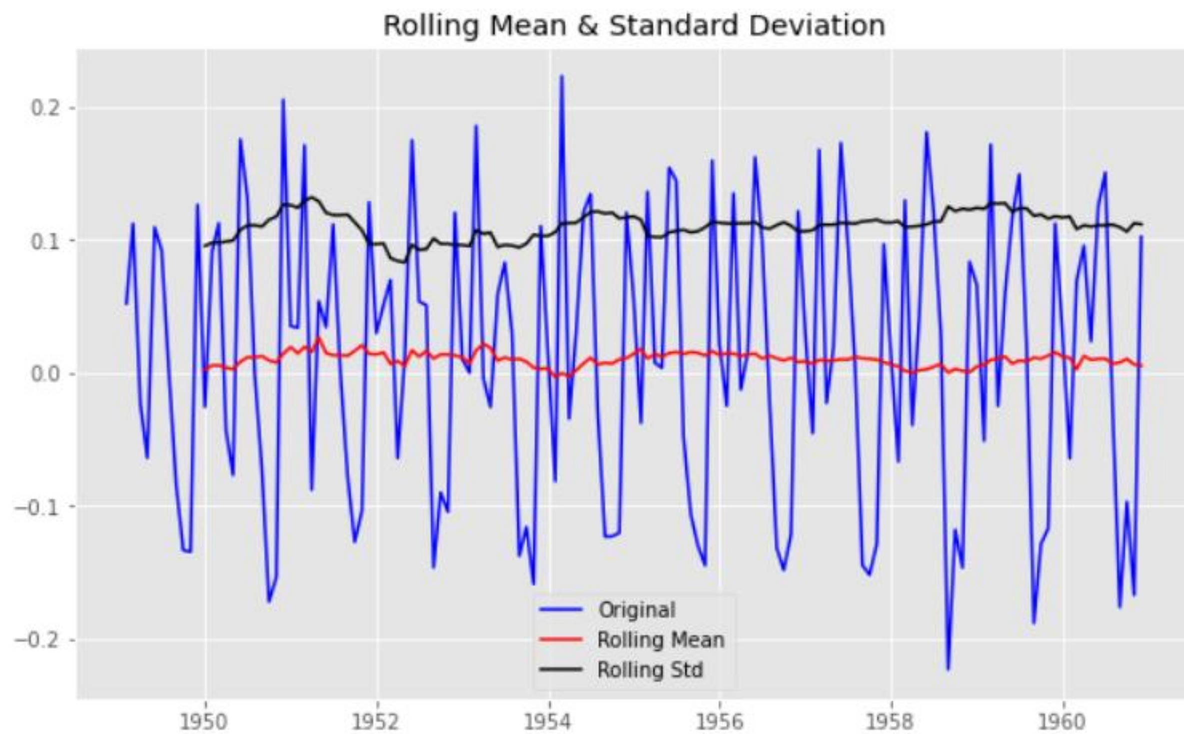



```
Results of Dickey Fuller Test:
Test Statistic                  -3.601262
p-value                          0.005737
#Lags Used                      13.000000
Number of Observations Used    130.000000
Critical Value (1%)             -3.481682
Critical Value (5%)             -2.884042
Critical Value (10%)            -2.578770
dtype: float64
```

```
"""# **Time Shift Transformation**"""

datasetLogDiffShifting = indexedDataset_logScale - indexedDataset_logScale.shift(
)
plt.plot(datasetLogDiffShifting)
```



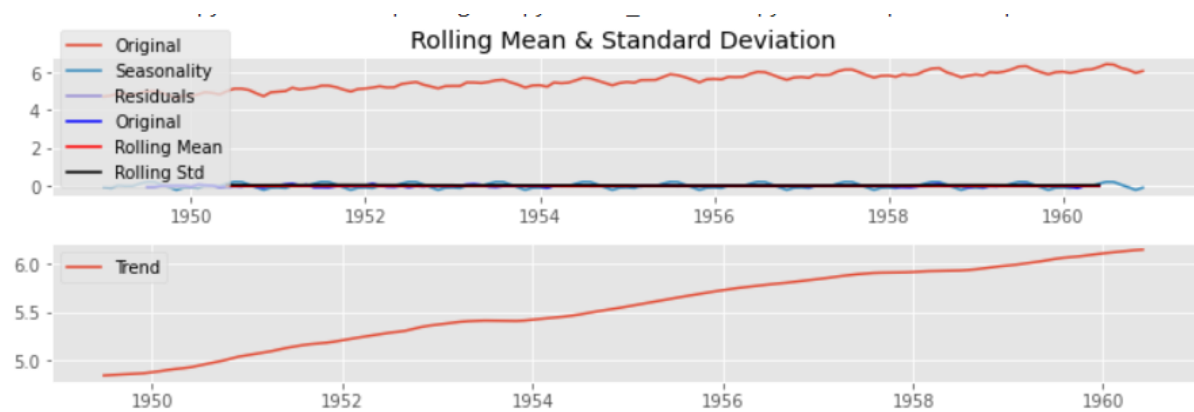

```
datasetLogDiffShifting.dropna(inplace=True)
test_stationarity(datasetLogDiffShifting)
```




```
Results of Dickey Fuller Test:
Test Statistic                  -2.717131
p-value                          0.071121
#Lags Used                      14.000000
Number of Observations Used    128.000000
Critical Value (1%)             -3.482501
Critical Value (5%)             -2.884398
Critical Value (10%)            -2.578960
dtype: float64
```

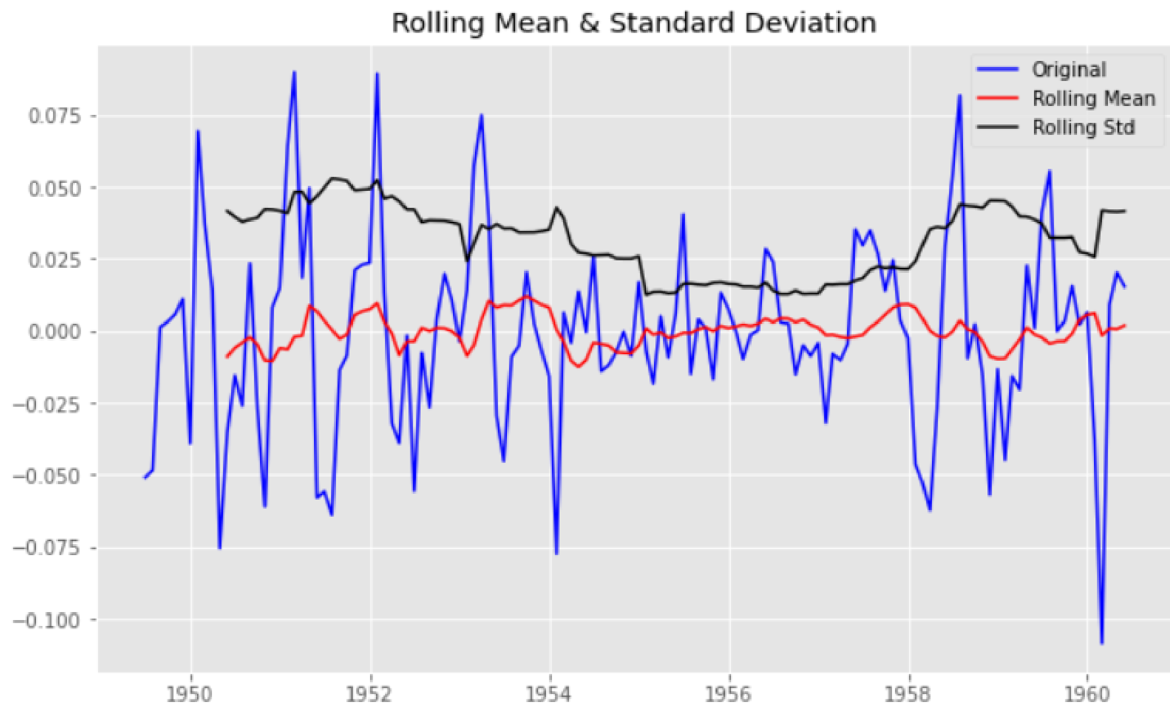
```
"""# **Decomposition using Log Scale Transformation**"""

decomposition = seasonal_decompose(indexedDataset_logScale)

trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

plt.subplot(411)
plt.plot(indexedDataset_logScale, label='Original')
plt.legend(loc='best')

plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend(loc='best')

plt.subplot(411)
```

```
plt.plot(seasonal, label='Seasonality')
plt.legend(loc='best')

plt.subplot(411)
plt.plot(residual, label='Residuals')
plt.legend(loc='best')

plt.tight_layout()

#there can be cases where an observation simply consisted of trend & seasonality.
 In that case, there won't be
#any residual component & that would be a null or NaN. Hence, we also remove such
 cases.
decomposedLogData = residual
decomposedLogData.dropna(inplace=True)
test_stationarity(decomposedLogData)
```




```
Results of Dickey Fuller Test:
Test Statistic                  -6.332387e+00
p-value                          2.885059e-08
#Lags Used                       9.000000e+00
Number of Observations Used      1.220000e+02
Critical Value (1%)             -3.485122e+00
Critical Value (5%)             -2.885538e+00
Critical Value (10%)            -2.579569e+00
dtype: float64
```

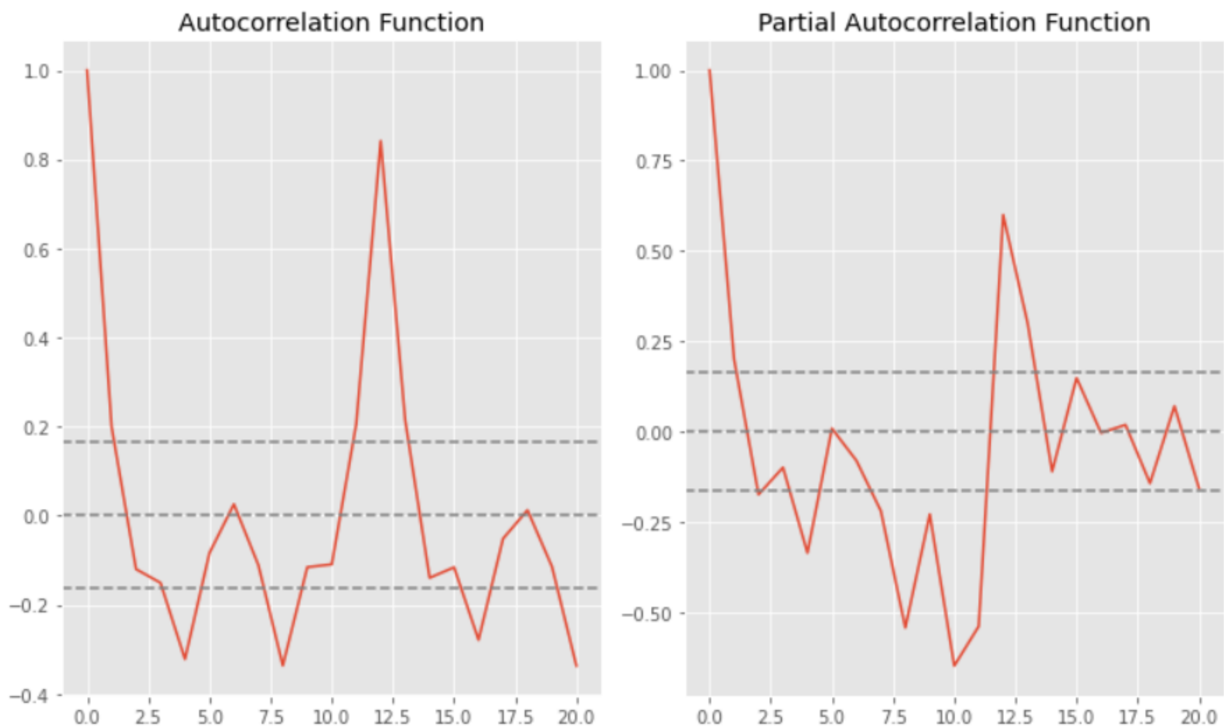
```
decomposedLogData = residual
decomposedLogData.dropna(inplace=True)
test_stationarity(decomposedLogData)
```



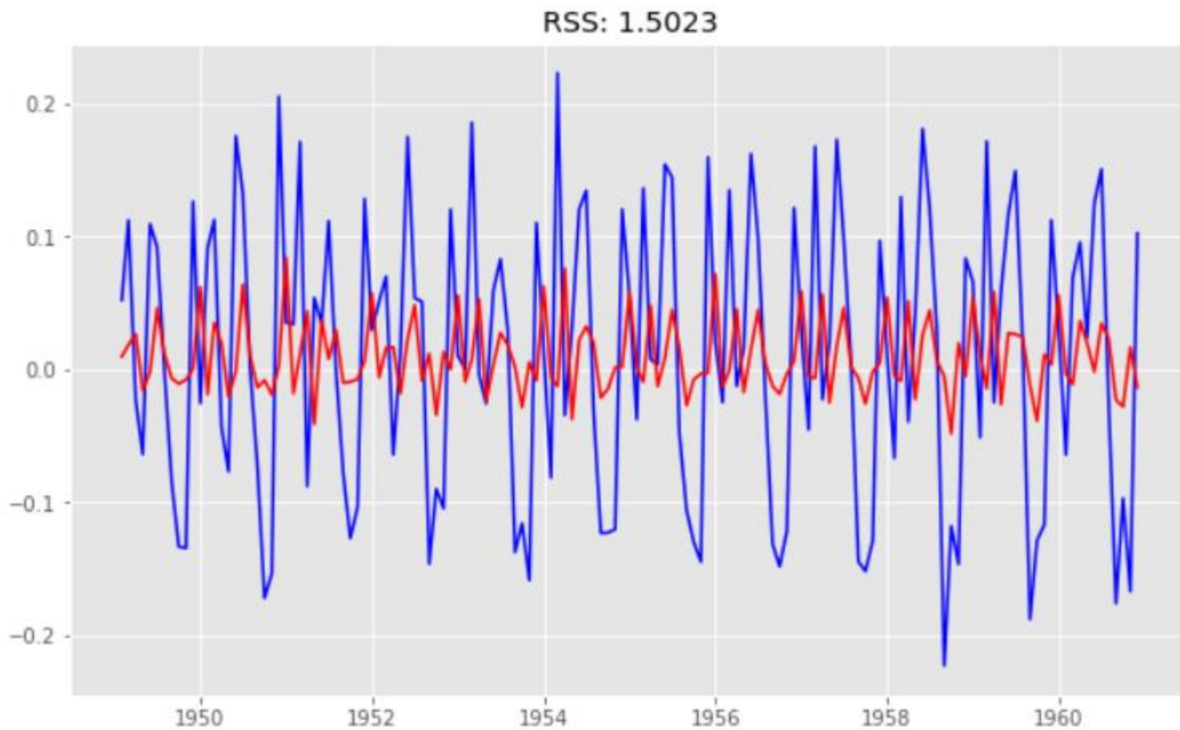

```
Results of Dickey Fuller Test:
Test Statistic                 -6.332387e+00
p-value                         2.885059e-08
#Lags Used                      9.000000e+00
Number of Observations Used     1.220000e+02
Critical Value (1%)            -3.485122e+00
Critical Value (5%)            -2.885538e+00
Critical Value (10%)           -2.579569e+00
dtype: float64
```

```
"""# **Plotting ACF & PACF**"""

# ACF & PACF plots

lag_acf = acf(datasetLogDiffShifting, nlags=20)
lag_pacf = pacf(datasetLogDiffShifting, nlags=20, method='ols')

# Plot ACF:
plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0, linestyle='--', color='gray')
plt.axhline(y=-1.96/np.sqrt(len(datasetLogDiffShifting)), linestyle='--
', color='gray')
plt.axhline(y=1.96/np.sqrt(len(datasetLogDiffShifting)), linestyle='--
', color='gray')
plt.title('Autocorrelation Function')
```

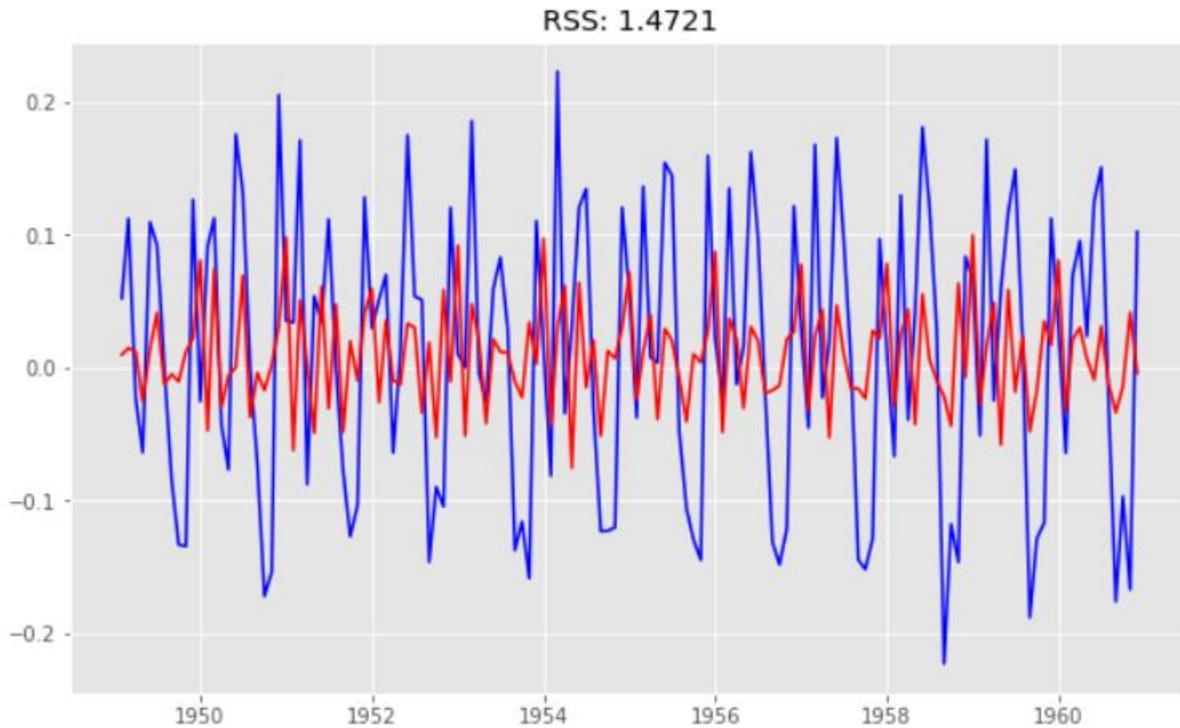
```
# Plot PACF
plt.subplot(122)
plt.plot(lag_pacf)
plt.axhline(y=0, linestyle='--', color='gray')
plt.axhline(y=-1.96/np.sqrt(len(datasetLogDiffShifting)), linestyle='--
', color='gray')
plt.axhline(y=1.96/np.sqrt(len(datasetLogDiffShifting)), linestyle='--
', color='gray')
plt.title('Partial Autocorrelation Function')

plt.tight_layout()
```



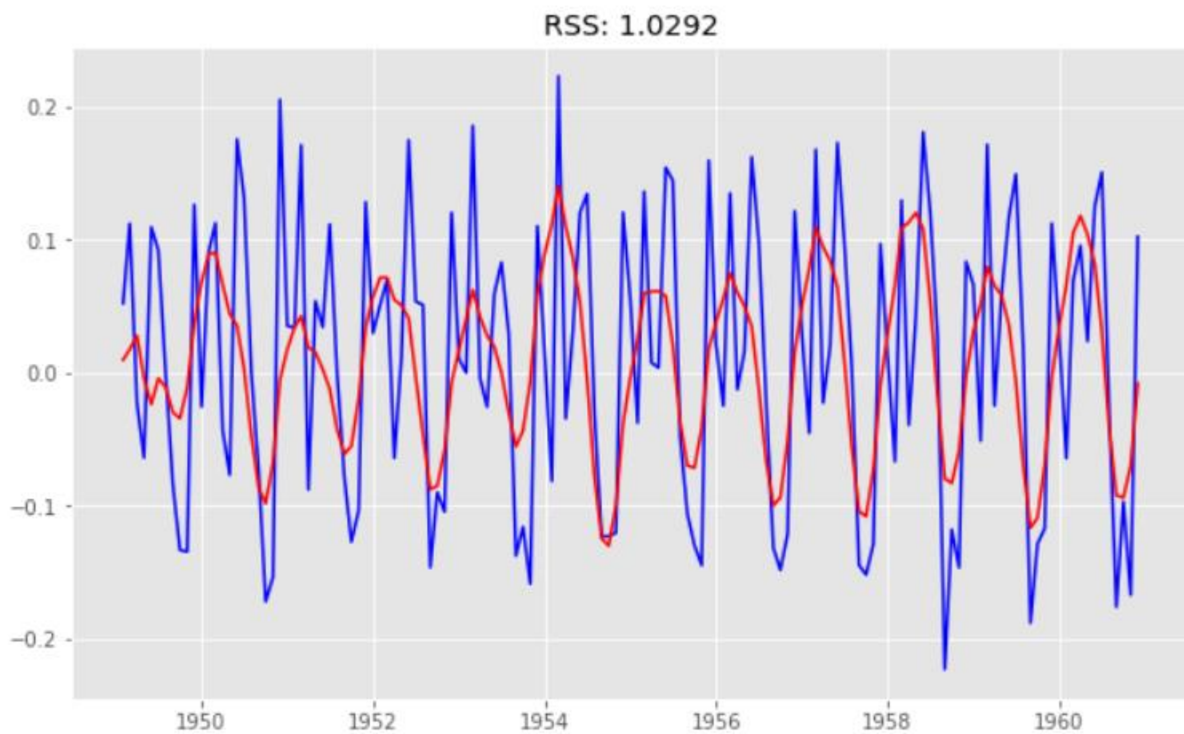

```

"""# **AR Model**"""

# AR Model
# Making Order=(2,1,0) gives RSS=1.5023
model = ARIMA(indexedDataset_logScale, order=(2,1,0))
results_AR = model.fit(disp=-1)
plt.plot(datasetLogDiffShifting, color='blue')
plt.plot(results_AR.fittedvalues, color='red')
plt.title('RSS: %.4f'%sum((results_AR.fittedvalues - datasetLogDiffShifting['#Pas
sengers'])**2))
print('Plotting AR model')
```




```
"""# **MA Model**"""

# MA Model
model = ARIMA(indexedDataset_logScale, order=(0,1,2))
results_MA = model.fit(disp=-1)
plt.plot(datasetLogDiffShifting, color='blue')
plt.plot(results_MA.fittedvalues, color='red')
plt.title('RSS: %.4f'%sum((results_MA.fittedvalues - datasetLogDiffShifting['#Pas
sengers'])**2))
print('Plotting MA model')
```




```
"""# **ARIMA Model**"""

# AR+I+MA = ARIMA model
model = ARIMA(indexedDataset_logScale, order=(2,1,2))
results_ARIMA = model.fit(disp=-1)
plt.plot(datasetLogDiffShifting, color='blue')
plt.plot(results_ARIMA.fittedvalues, color='red')
plt.title('RSS: %.4f'%sum((results_ARIMA.fittedvalues - datasetLogDiffShifting['#
Passengers'])**2))
print('Plotting ARIMA model')
```




```
"""# **Predictions & Reverse Transformation**"""

predictions_ARIMA_diff = pd.Series(results_ARIMA.fittedvalues, copy=True)
```

```
Month
1949-02-01    0.009580
1949-03-01    0.017491
1949-04-01    0.027670
1949-05-01   -0.004521
1949-06-01   -0.023889
dtype: float64
```

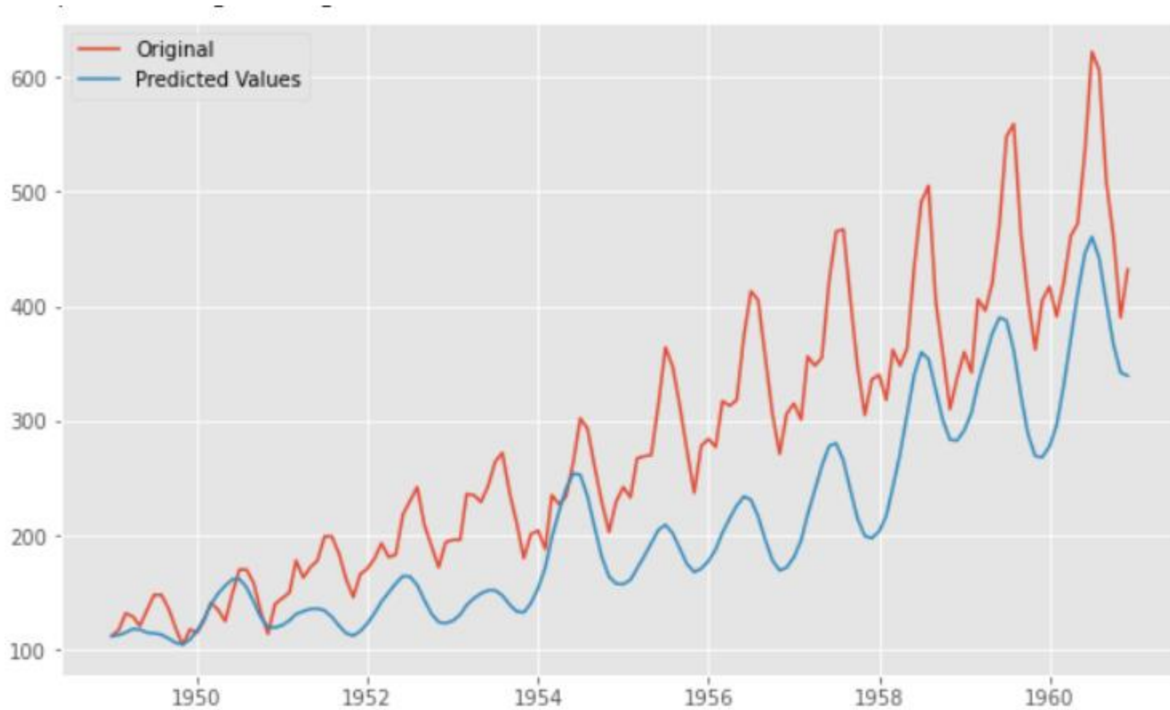
```
print(predictions_ARIMA_diff.head())

# Convert to Cumulative Sum
predictions_ARIMA_diff_cumsum = predictions_ARIMA_diff.cumsum()
print(predictions_ARIMA_diff_cumsum)
```

```
Month
1949-02-01    0.009580
1949-03-01    0.027071
1949-04-01    0.054742
1949-05-01    0.050221
1949-06-01    0.026331
                ...
1960-08-01    1.372554
1960-09-01    1.280204
1960-10-01    1.186191
1960-11-01    1.116267
1960-12-01    1.108140
Length: 143, dtype: float64
```

```
predictions_ARIMA_log = pd.Series(indexedDataset_logScale['#Passengers'].iloc[0],
 index=indexedDataset_logScale.index)
predictions_ARIMA_log = predictions_ARIMA_log.add(predictions_ARIMA_diff_cumsum,
fill_value=0)
predictions_ARIMA_log.head()
```

```
Month
1949-01-01    4.718499
1949-02-01    4.728079
1949-03-01    4.745570
1949-04-01    4.773241
1949-05-01    4.768720
dtype: float64
```

```
# Inverse of log is exp.
predictions_ARIMA = np.exp(predictions_ARIMA_log)
plt.plot(indexedDataset, label='Original')
plt.plot(predictions_ARIMA, label='Predicted Values')
plt.legend(loc='best')
```

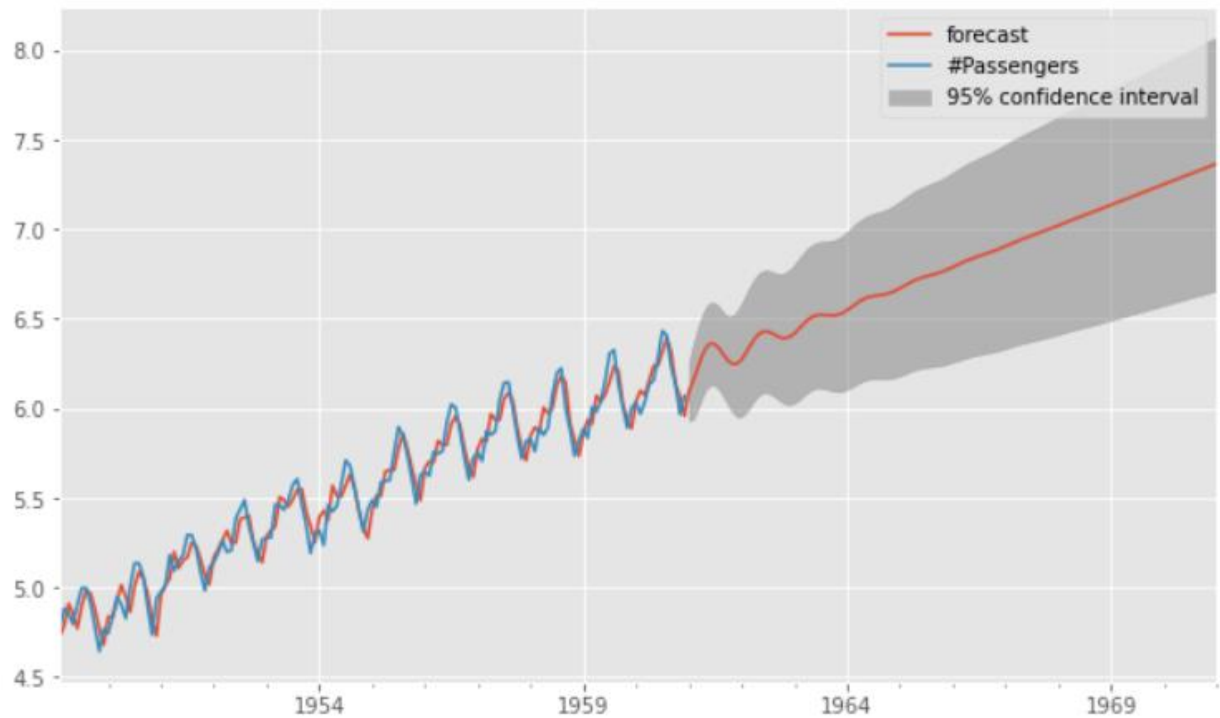



```
indexedDataset_logScale
```

	#Passengers
Month	
1949-01-01	4.718499
1949-02-01	4.770685
1949-03-01	4.882802
1949-04-01	4.859812
1949-05-01	4.795791
...	...
1960-08-01	6.406880
1960-09-01	6.230481
1960-10-01	6.133398
1960-11-01	5.966147
1960-12-01	6.068426

144 rows × 1 columns

```
# Forecasting Next 10 yrs
results_ARIMA.plot_predict(1,264)
```



Conclusion:
We learnt the basics of time series analysis. We also learnt ways to achieve stationary time series by using different transformation techniques on the dataset. We also learnt evaluation of AR, MA and ARIMA model parameters using ACF & PACF.