air-passengers-forecasting-arima-vs-sarima

September 29, 2021

1 ARIMA vs SARIMA

1.1 IMPORT THE NECESSARY LIBRARIES

[46]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from datetime import datetime
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from itertools import combinations

from datetime import datetime
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import acf, pacf
from statsmodels.tsa.arima_model import ARIMA as ARIMA
import statsmodels.api as sm
import statsmodels.tsa.api as smt
pd.options.display.float_format = '{:.2f}'.format
```

1.2 IMPORT THE DATASET

[48]:
```
data = pd.read_csv('AirPW.csv')
data.head()
```

[48]:

	Month	#Passengers
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

1.3 CHECK FOR MISSING VALUES AND BASIC INFO

[3]:
```
data.isnull().sum()
```

[3]:
```
Month          0
#Passengers    0
dtype: int64
```

[49]:
```
data.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
```

[50]:
```
data['Month'] = pd.to_datetime(data['Month'], infer_datetime_format = True)
data['Month'] = data['Month'].dt.to_period('M')

data['Month'] = data['Month'].astype(str)
data['Month'] = pd.to_datetime(data['Month'])

data['Date'] = pd.to_datetime(data['Month'],format='%Y-%m-%d')
data = data.drop(columns = 'Month')
data = data.set_index('Date')
data = data.rename(columns = {'#Passengers':'Passengers'})
data.head()
```

[50]:

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

[51]:
```
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   Passengers   144 non-null    int64
```

```
dtypes: int64(1)
memory usage: 2.2 KB
```

- NO MISSING VALUES
- CONVERT THE MONTH COLUMN TO DATETIME DATATYPE AND ASSIGN IT AS INDEX

[6]:
```
#data['Date'] = pd.to_datetime(data['Month'])
#data = data.drop(columns = 'Month')
#data = data.set_index('Date')
#data = data.rename(columns = {'#Passengers':'Passengers'})
#data.head()
```

1.4 FUNCTIONS FOR TIMESERIES ANALYSIS

[7]:
```
def test_stationarity(timeseries):
    #Determing rolling statistics
    MA = timeseries.rolling(window=12).mean()
    MSTD = timeseries.rolling(window=12).std()

    #Plot rolling statistics:
    plt.figure(figsize=(15,5))
    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(MA, color='red', label='Rolling Mean')
    std = plt.plot(MSTD, 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, 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)
```

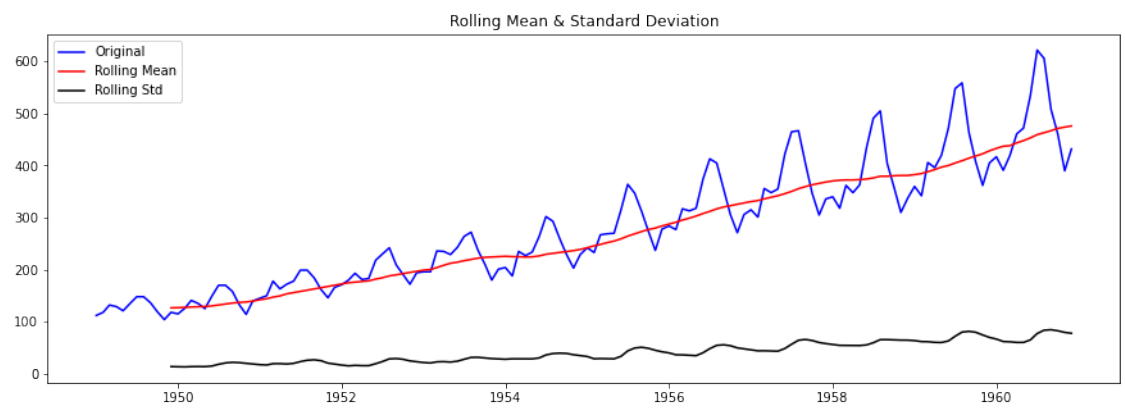
[8]:
```
def tsplot(y, lags=None, figsize=(12, 7), style='bmh'):
    if not isinstance(y, pd.Series):
        y = pd.Series(y)

    with plt.style.context(style):
        fig = plt.figure(figsize=figsize)
        layout = (2, 2)
        ts_ax = plt.subplot2grid(layout, (0, 0), colspan=2)
        acf_ax = plt.subplot2grid(layout, (1, 0))
        pacf_ax = plt.subplot2grid(layout, (1, 1))
```

```
        y.plot(ax=ts_ax)
        p_value = sm.tsa.stattools.adfuller(y)[1]
        ts_ax.set_title('Time Series Analysis Plots\n Dickey-Fuller: p={0:.5f}'.
↪format(p_value))
        smt.graphics.plot_acf(y, lags=lags, ax=acf_ax)
        smt.graphics.plot_pacf(y, lags=lags, ax=pacf_ax)
        plt.tight_layout()
```

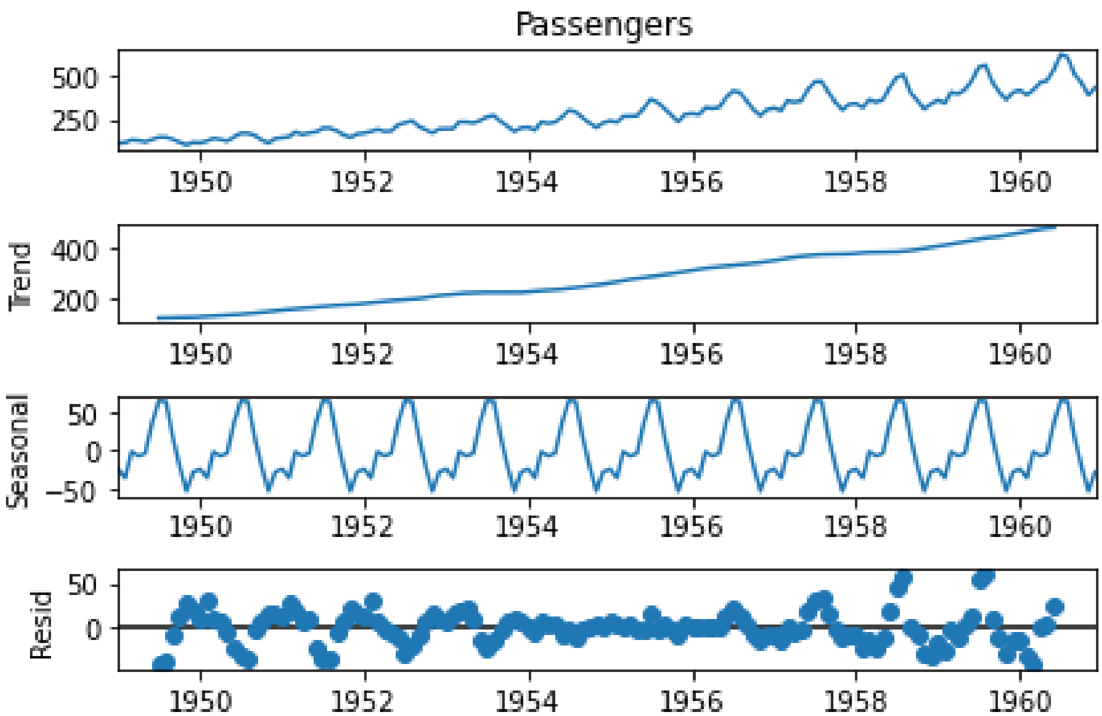
1.4.1 DATA

[9]:
```
test_stationarity(data['Passengers'])
```




```
Results of Dickey-Fuller Test:
Test Statistic                   0.82
p-value                          0.99
#Lags Used                      13.00
Number of Observations Used    130.00
Critical Value (1%)             -3.48
Critical Value (5%)             -2.88
Critical Value (10%)            -2.58
dtype: float64
```

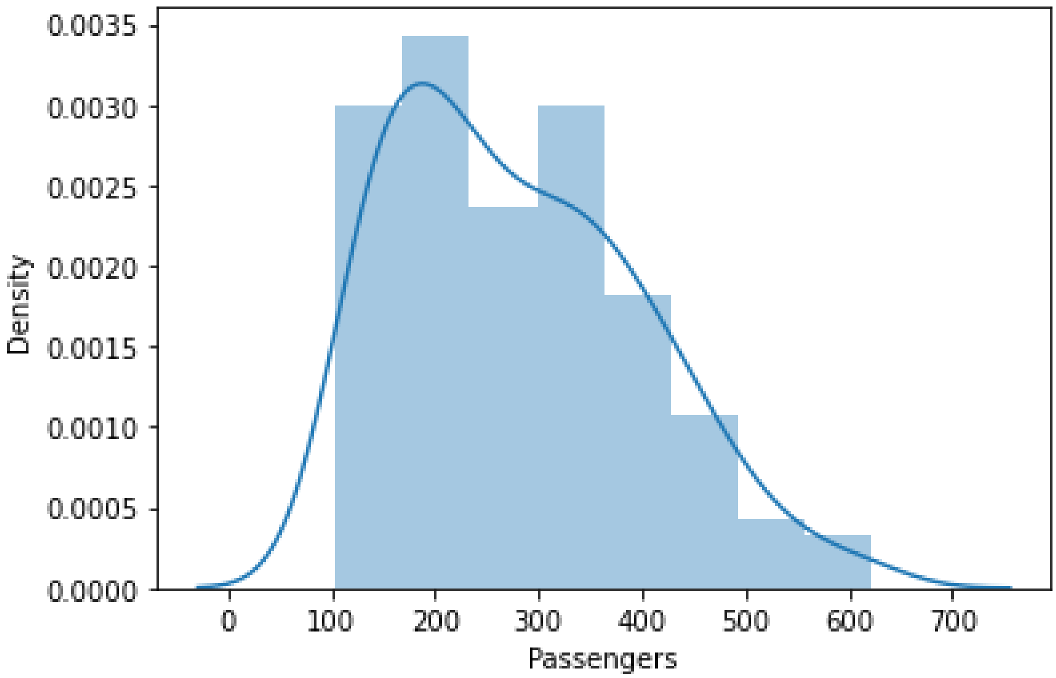
[10]:
```
dec = sm.tsa.seasonal_decompose(data['Passengers'],period = 12).plot()
plt.show()
```



[11]:
```
sns.distplot(data['Passengers'])
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[11]:
```
<AxesSubplot:xlabel='Passengers', ylabel='Density'>
```



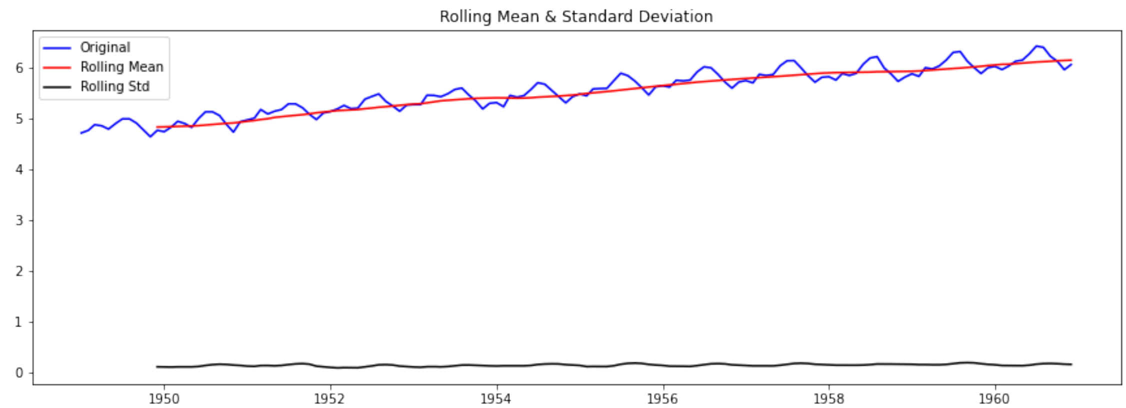
- DATA IS NOT STATIONARY AS THE TEST STATISTIC VALUE IS MORE THAN ANY OF THE CRITICAL VALUE
- ALSO THE P-Value IS NOT LESS THAN 0.05
- DATA HAS AN INCREASING TREND
- DATA IS ALSO SEASONAL WITH A PATTERN OF 1 YEAR

1.4.2 LOG DATA

[12]:
```
log_data = np.log(data)
log_data.head()
```

[12]:
```
            Passengers
Date
1949-01-01        4.72
1949-02-01        4.77
1949-03-01        4.88
1949-04-01        4.86
1949-05-01        4.80
```

[13]:
```
test_stationarity(log_data[ 'Passengers'])
```

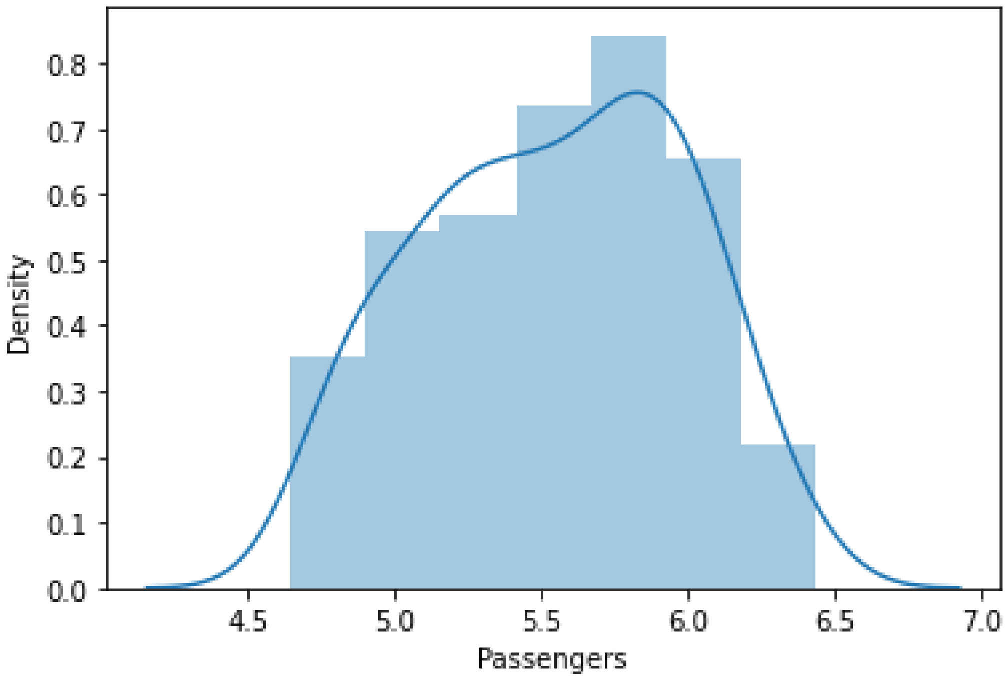



```
Results of Dickey-Fuller Test:
Test Statistic                 -1.72
p-value                         0.42
#Lags Used                     13.00
Number of Observations Used   130.00
Critical Value (1%)            -3.48
Critical Value (5%)            -2.88
Critical Value (10%)           -2.58
dtype: float64
```

[14]:
```python
sns.distplot(log_data['Passengers'])
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[14]:
```
<AxesSubplot:xlabel='Passengers', ylabel='Density'>
```

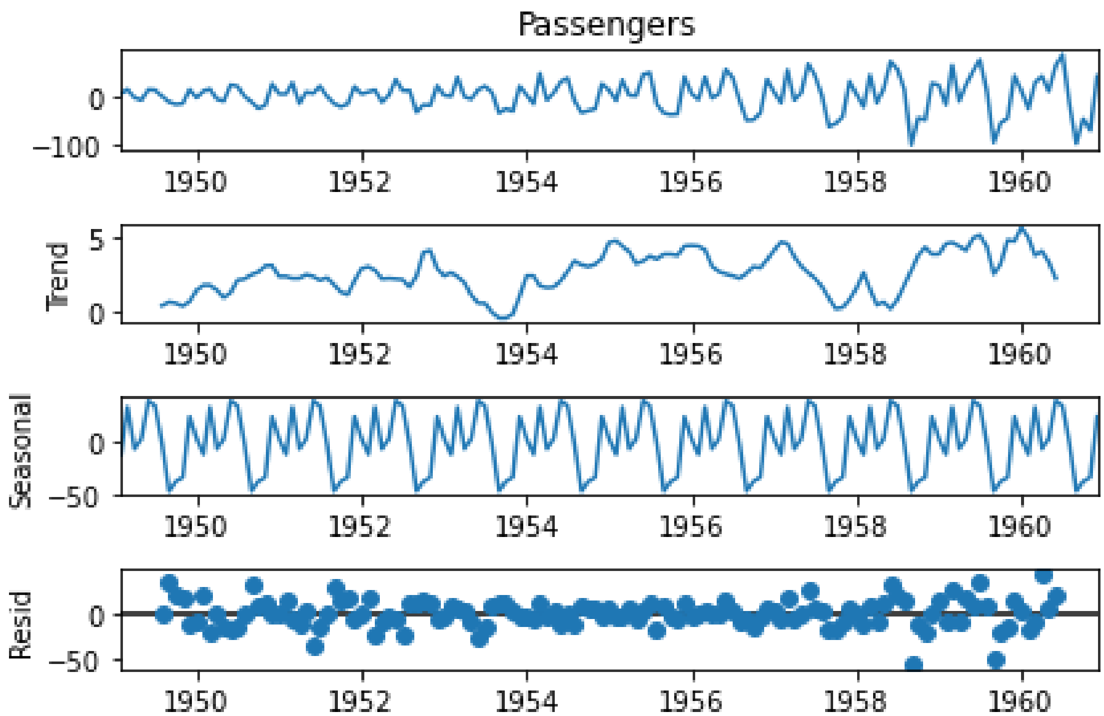


- LOG DATA ALSO HAS THE SAME ATTRIBUTES AS THAT OF DATA
- ONLY THE DATA DISTRIBUTION IS SLIGHTLY BETTER THAN PREVIOUS

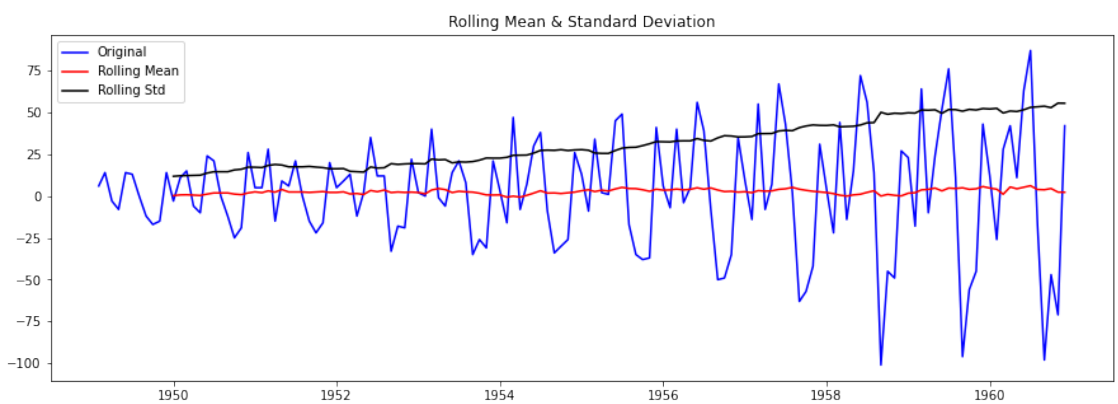
1.5 DIFFERENCING

1] DATA

[15]:
```
data_diff = data['Passengers'].diff()
data_diff = data_diff.dropna()
dec = sm.tsa.seasonal_decompose(data_diff,period = 12).plot()
plt.show()
```



[16]:
```
test_stationarity(data_diff)
```



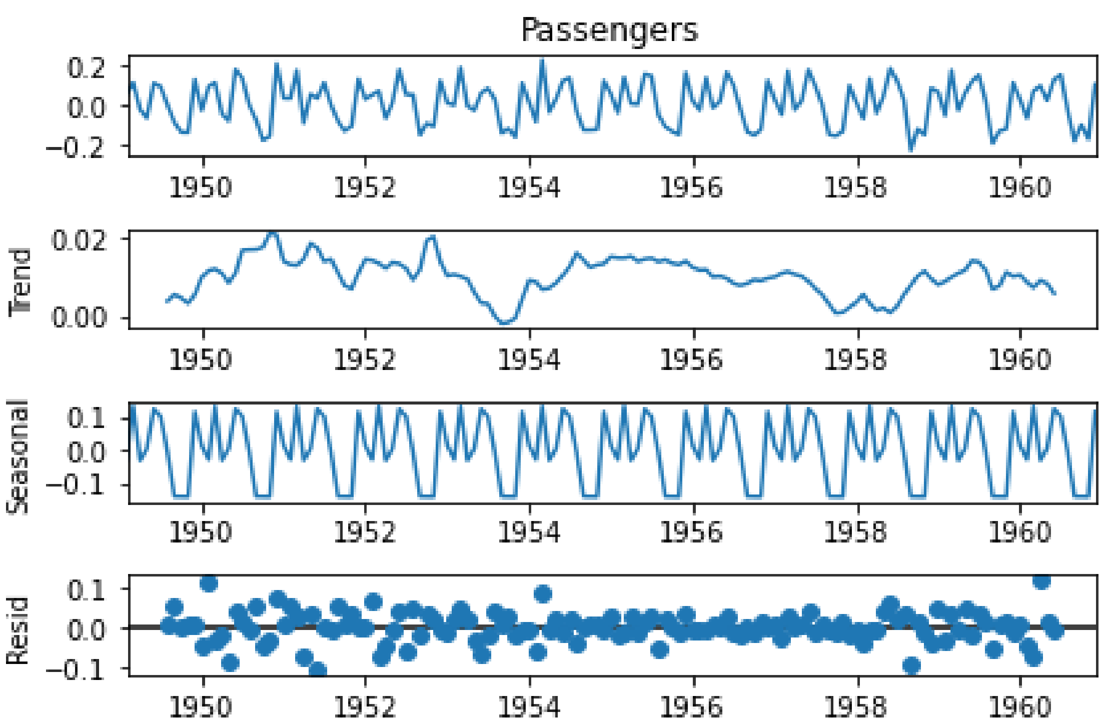

```
Results of Dickey-Fuller Test:
Test Statistic                  -2.83
p-value                          0.05
#Lags Used                      12.00
Number of Observations Used    130.00
Critical Value (1%)             -3.48
Critical Value (5%)             -2.88
```

```
Critical Value (10%)           -2.58
dtype: float64
```

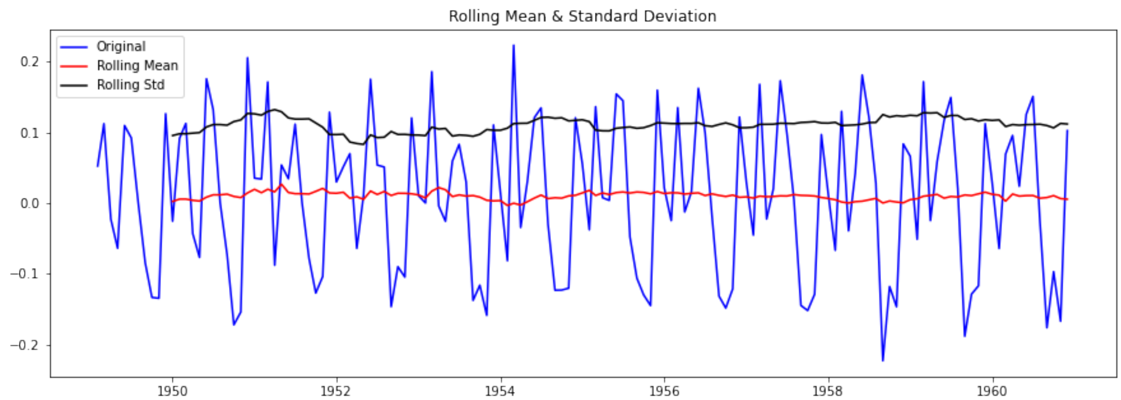
- TREND HAS DIED DOWN AND IS CONSTANT
- TEST STATISTIC < CRITICAL VALUE(10%) –> DATA IS 90% SURELY STATIONARY
- P-Value = 0.05
- ROLLING IS ALSO CONSTANT
- HENCE DATA IS STATIONARY
- HOWEVER SEASONALITY IS STILL PRESENT

2] LOG DATA

[17]:
```
log_data_diff = log_data['Passengers'].diff()
log_data_diff = log_data_diff.dropna()
dec = sm.tsa.seasonal_decompose(log_data_diff,period = 12)
dec.plot()
plt.show()
```



[18]:
```
test_stationarity(log_data_diff)
```




```
Results of Dickey-Fuller Test:
Test Statistic                  -2.72
p-value                          0.07
#Lags Used                      14.00
Number of Observations Used    128.00
Critical Value (1%)             -3.48
Critical Value (5%)             -2.88
Critical Value (10%)            -2.58
dtype: float64
```

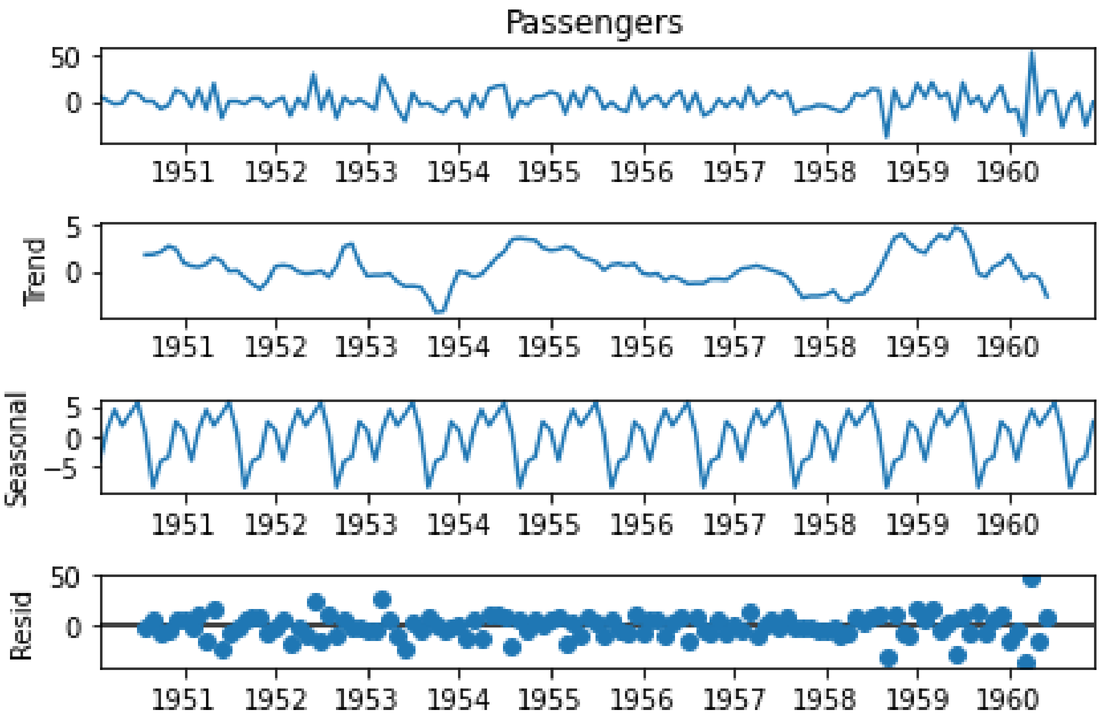
- TREND HAS DIED DOWN AND IS CONSTANT
- TEST STATISTIC < CRITICAL VALUE(10%) –> DATA IS 90% SURELY STATIONARY
- P-Value = 0.05
- ROLLING IS ALSO CONSTANT
- HENCE DATA IS STATIONARY
- HOWEVER SEASONALITY IS STILL PRESENT

1.6 FROM THE ABOVE TESTS, WE CAN CHOOSE ANY OF THE DATA ABOVE FOR SELECTING THE ORDER OF SARIMA

2 SARIMA [(p,d,q)x(P,D,Q,s)]

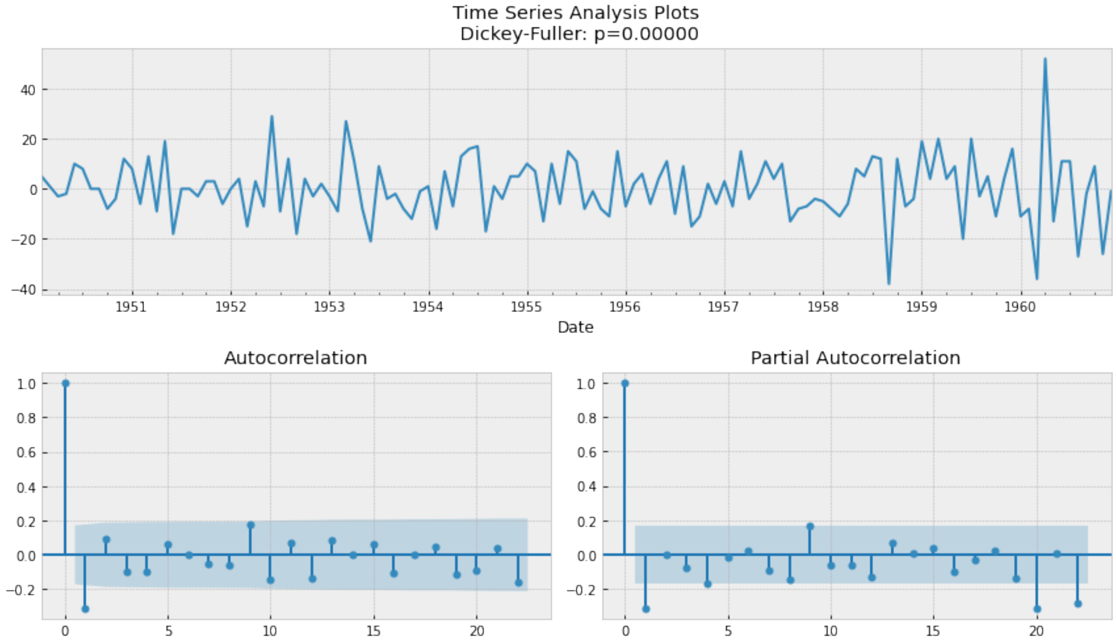
1] DATA

[19]:
```
data_diff_seas = data_diff.diff(12)
data_diff_seas = data_diff_seas.dropna()
dec = sm.tsa.seasonal_decompose(data_diff_seas,period = 12)
dec.plot()
plt.show()
```



- SEASONAL DIFFERENCE WITH A SEASONAL PERIOD(s) OF 12
- SINCE OUR DATA IS MONTHLY DATA AND FROM THE PLOTS,WE OBSERVE THAT A YEARLY PATTERN IS PRESENT
- WE USE THIS OPERATION ON THE PREVIOUSLY DIFFERENCED DATA SO THAT WE DO NOT HAVE TO DEAL WITH TREND & STATIONARITY AGAIN

[20]:
```
tsplot(data_diff_seas)
```



- SARIMA MODEL ORDER [(p,d,q)x(P,D,Q,s)]
- (p,d,q) = THIS ORDER IS INHERITED FROM OUR ABOVE ARIMA MODEL
- (P,D,Q,s) = THIS IS ORDER IS SELECTED USING THE SAME TECHNIQUE USED FOR ARIMA
- s = SEASONAL ORDER = ONLY ADDITIONAL PARAMETER
- WE AGAIN SELECT THE MODEL WITH LEAST AIC SCORE

[21]:
```
model = sm.tsa.statespace.SARIMAX(data['Passengers'],order =␣
↪(2,1,2),seasonal_order = (1,1,2,12))
results = model.fit()
print(results.summary())
```

```
/opt/conda/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:524:
ValueWarning: No frequency information was provided, so inferred frequency MS
will be used.
  warnings.warn('No frequency information was'
/opt/conda/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:524:
ValueWarning: No frequency information was provided, so inferred frequency MS
will be used.
  warnings.warn('No frequency information was'

RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            8     M =           10
```

```
At X0         0 variables are exactly at the bounds

At iterate    0    f=  3.51684D+00    |proj g|=  9.17899D-02
 This problem is unconstrained.


At iterate    5    f=  3.50599D+00    |proj g|=  1.13879D-02

At iterate   10    f=  3.50179D+00    |proj g|=  9.12514D-03

At iterate   15    f=  3.49420D+00    |proj g|=  9.01407D-03

At iterate   20    f=  3.49288D+00    |proj g|=  4.89332D-03

At iterate   25    f=  3.49256D+00    |proj g|=  2.65990D-03

At iterate   30    f=  3.47276D+00    |proj g|=  2.77433D-02

At iterate   35    f=  3.46708D+00    |proj g|=  5.14822D-03

At iterate   40    f=  3.46539D+00    |proj g|=  8.39646D-04

At iterate   45    f=  3.46530D+00    |proj g|=  2.32677D-03

At iterate   50    f=  3.46524D+00    |proj g|=  1.96666D-04

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    8     50     64      1     0     0   1.967D-04   3.465D+00
  F =   3.4652352723483677

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT
```

```
/opt/conda/lib/python3.9/site-packages/statsmodels/base/model.py:566:
ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check
mle_retvals
```

```
  warnings.warn("Maximum Likelihood optimization failed to "
                                     SARIMAX Results
================================================================================
==========
Dep. Variable:                           Passengers   No. Observations:
144
Model:             SARIMAX(2, 1, 2)x(1, 1, 2, 12)   Log Likelihood
-498.994
Date:                          Sun, 26 Sep 2021   AIC
1013.988
Time:                                  07:48:33   BIC
1036.989
Sample:                              01-01-1949   HQIC
1023.334
                                   - 12-01-1960
Covariance Type:                            opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.6119      0.384      1.595      0.111      -0.140       1.364
ar.L2          0.2192      0.249      0.881      0.378      -0.269       0.707
ma.L1         -1.0744      0.426     -2.521      0.012      -1.910      -0.239
ma.L2          0.0917      0.319      0.287      0.774      -0.534       0.718
ar.S.L12       0.9833      0.097     10.162      0.000       0.794       1.173
ma.S.L12      -1.2945      0.305     -4.251      0.000      -1.891      -0.698
ma.S.L24       0.3758      0.152      2.477      0.013       0.078       0.673
sigma2       107.6977     20.092      5.360      0.000      68.319     147.077
================================================================================
===
Ljung-Box (L1) (Q):                   0.04   Jarque-Bera (JB):
15.54
Prob(Q):                              0.84   Prob(JB):
0.00
Heteroskedasticity (H):               2.72   Skew:
-0.03
Prob(H) (two-sided):                  0.00   Kurtosis:
4.69
================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

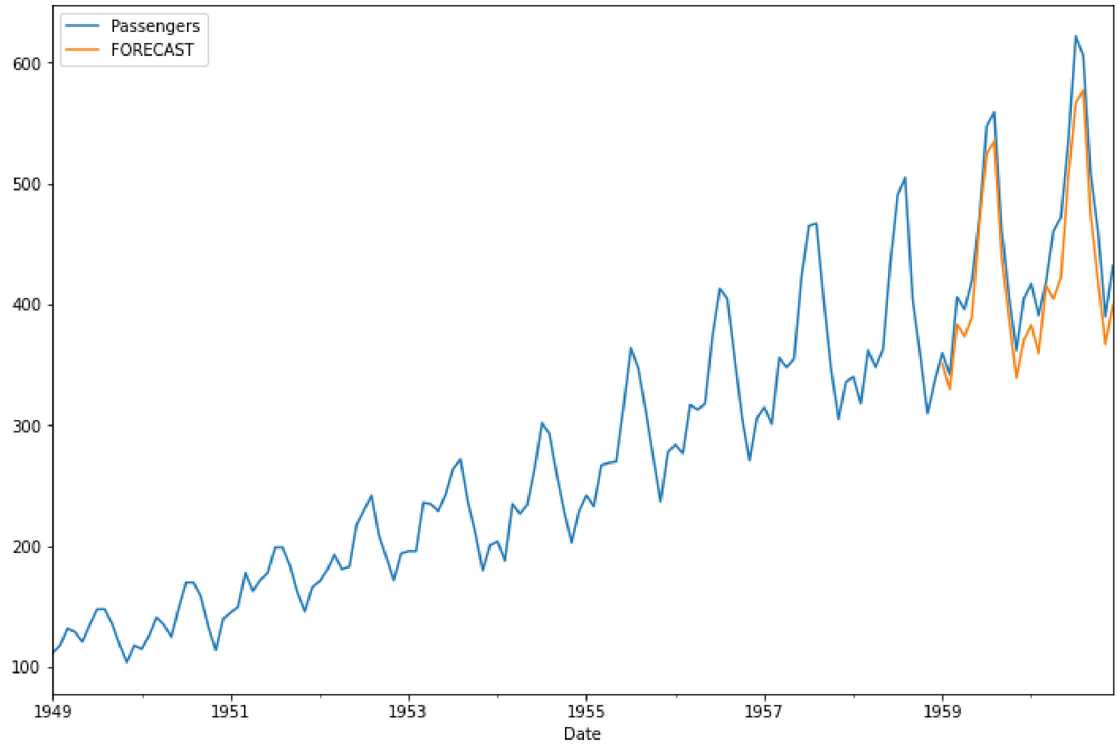
[28]:

```
#data = data.reset_index()

data['FORECAST'] = results.predict(start = 120,end = 144,dynamic = True)
```

```
data[['Passengers','FORECAST']].plot(figsize = (12,8))
```

[28]: <AxesSubplot:xlabel='Date'>



[29]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 144 entries, 1949-01-01 to 1960-12-01
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Passengers  144 non-null    int64
 1   FORECAST    24 non-null     float64
dtypes: float64(1), int64(1)
memory usage: 3.4 KB
```

[30]:
```
np.any(np.isnan(data))
np.all(np.isfinite(data))
```

[30]: False

[31]:
```
exp = [data.iloc[i,0] for i in range(120,len(data))]
pred = [data.iloc[i,1] for i in range(120,len(data))]
```

```
print(mean_absolute_error(exp,pred))
```

```
27.662716979860505
```

[32]:
```
def mean_absolute_percentage_error(exp, pred):
    return np.mean(np.abs((exp - pred) /np.abs(exp))) * 100
print('Mean absolute percentage error: ',mean_absolute_percentage_error(data.
 ↪Passengers,data.FORECAST))
```

```
Mean absolute percentage error:  6.073880840287297
```

[33]:
```
data = data.drop(columns = 'FORECAST')
```

- PREDICTED PLOTS ARE GREAT
- ERROR HAS ALSO REDUCED ALOT
- HENCE WE ACCEPT THIS MODEL AND FORECAST FOR 2 MORE YEARS

3 FORECASTING

3.0.1 ADD DATES TO OUR DATAFRAME FOR OUR FORECASTING PURPOSE

[57]:
```
from pandas.tseries.offsets import DateOffset
future_dates = [data.index[-1] + DateOffset(months = x)for x in range(0,37)]
df = pd.DataFrame(index = future_dates[1:],columns = data.columns)
```

[34]:
```
start_index = '1961-01-01'
end_index = '1963-12-01'
forecast = results.predict(start=start_index, end=end_index)
#forecast = forecast.astype(str)
```

```
/opt/conda/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:132:
FutureWarning: The 'freq' argument in Timestamp is deprecated and will be
removed in a future version.
  date_key = Timestamp(key, freq=base_index.freq)
```

[35]:
```
test = pd.read_csv('Test2.csv')
test.head()
```

[35]:

	Month
0	1961-01-01
1	1961-02-01
2	1961-03-01
3	1961-04-01
4	1961-05-01

[36]:
```
test = test.set_index(forecast.index)
```

[37]:
```python
test['Passengers'] = forecast
test.head()
```

[37]:

	Month	Passengers
1961-01-01	1961-01-01	447.77
1961-02-01	1961-02-01	420.76
1961-03-01	1961-03-01	463.17
1961-04-01	1961-04-01	488.67
1961-05-01	1961-05-01	505.92

[38]:
```python
test['Passengers'] = test['Passengers'].round(decimals = 0)
```

[39]:
```python
test['Passengers'] = test['Passengers'].astype(int)
```

[40]:
```python
test['Passengers'].head()
```

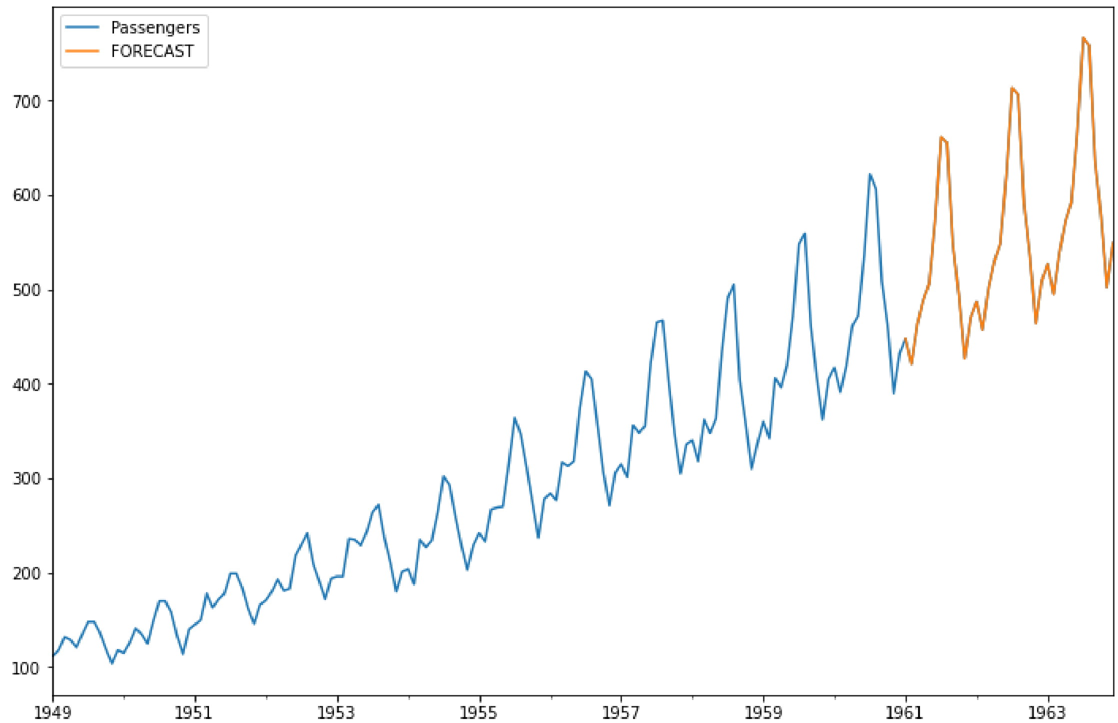
[40]:
```
1961-01-01    448
1961-02-01    421
1961-03-01    463
1961-04-01    489
1961-05-01    506
Freq: MS, Name: Passengers, dtype: int64
```

[41]:
```python
test.to_csv('Predictions.csv', header=True, index=False)
```

3.1 FINAL PLOT

[44]:
```python
forecast = pd.concat([data,test])
forecast['FORECAST'] = results.predict(start = 144,end = 300,dynamic = True)
forecast[['Passengers','FORECAST']].plot(figsize = (12,8))
```

[44]: <AxesSubplot:>


Passengers
FORECAST
700
600
500
400
300
200
100
1949
1951
1953
1955
1957
1959
1961
1963