In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from datetime import datetime
pd.options.display.float_format = '{:.2f}'.format

from itertools import combinations
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
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
```

In [2]:

```
data = pd.read_csv('AirPassengers.csv')
```

In [3]:

```
data.head()
```

Out[3]:

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

In [4]:

```
data.shape
```

Out[4]:

```
(144, 2)
```

In [5]:

```
data.columns
```

Out[5]:

```
Index(['Month', '#Passengers'], dtype='object')
```

In [6]:

```
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
```

In [10]:

```
data.describe()
```

Out[10]:

	#Passengers
count	144.00
mean	280.30
std	119.97
min	104.00
25%	180.00
50%	265.50
75%	360.50
max	622.00

In [11]:

```
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()
```

Out[11]:

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

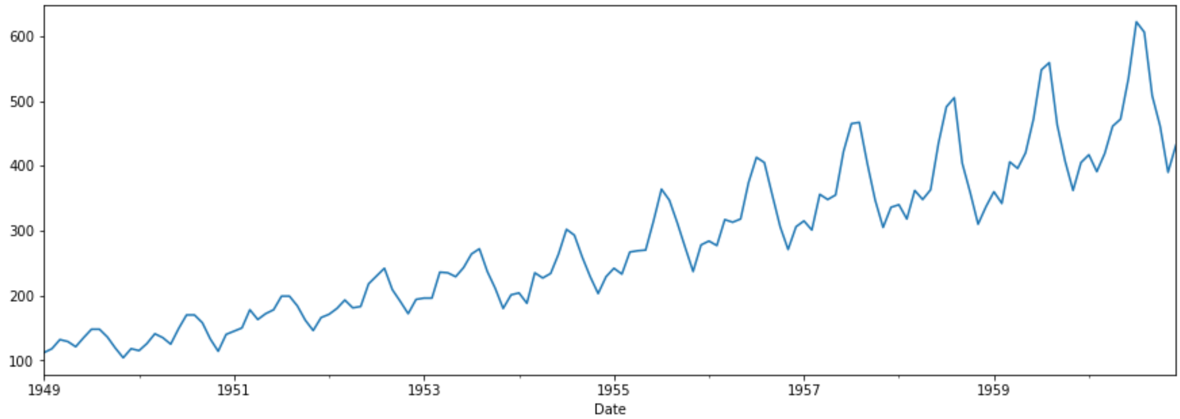
In [12]:

```
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
```

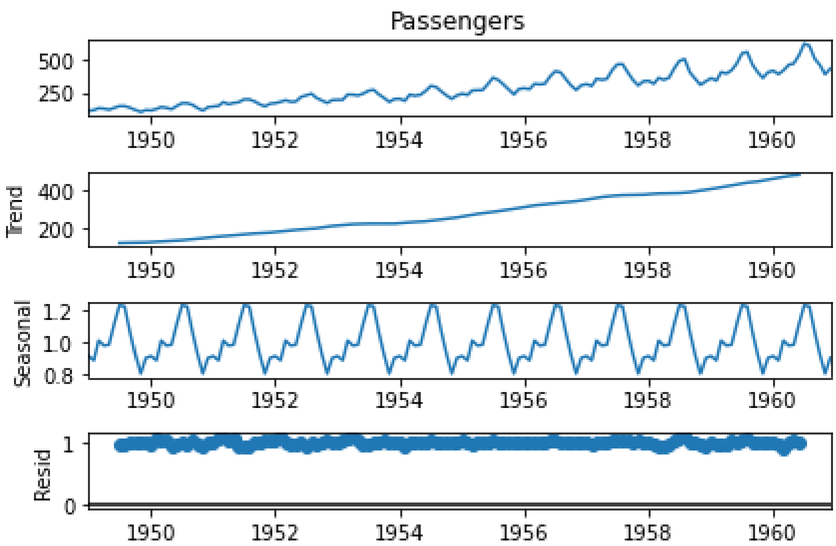
In [13]:

```
plt.figure(figsize = (15,5))
data['Passengers'].plot();
```



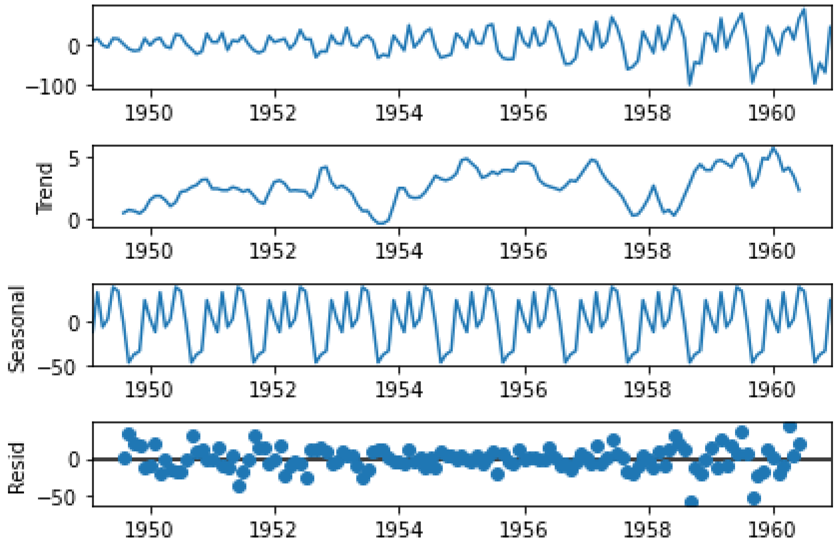
In [14]:

```
dec = sm.tsa.seasonal_decompose(data['Passengers'],period = 12, model = 'multiplicative').p
plt.show()
```



In [16]:

```
data_diff = data.diff()
data_diff = data_diff.dropna()

dec = sm.tsa.seasonal_decompose(data_diff,period = 12).plot()
plt.show()
```



In [17]:

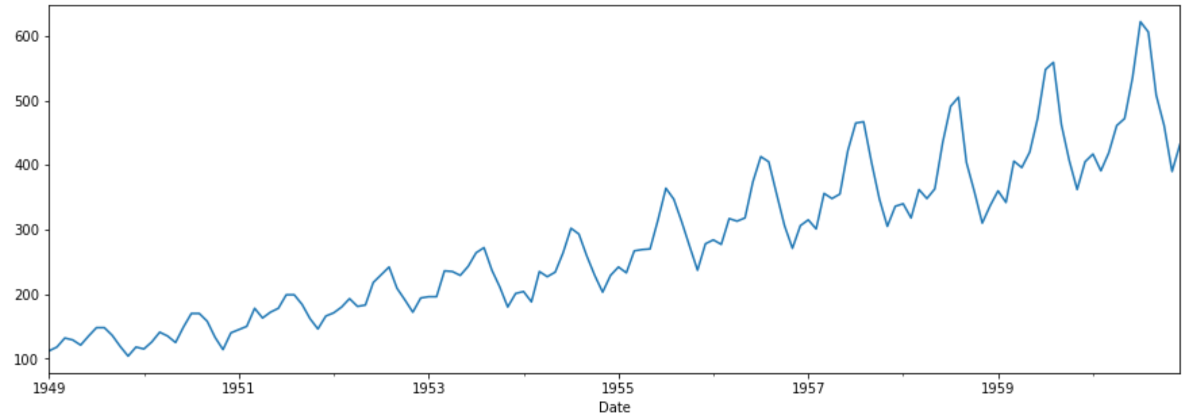
```
def test_stationarity(timeseries):
    #Determing rolling statistics
    MA = timeseries.rolling(window=12).mean()
    MSTD = timeseries.rolling(window=12).std()

    #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','Numbe
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)
```

In [18]:

```
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_va
        smt.graphics.plot_acf(y, lags=lags, ax=acf_ax)
        smt.graphics.plot_pacf(y, lags=lags, ax=pacf_ax)
        plt.tight_layout()
```

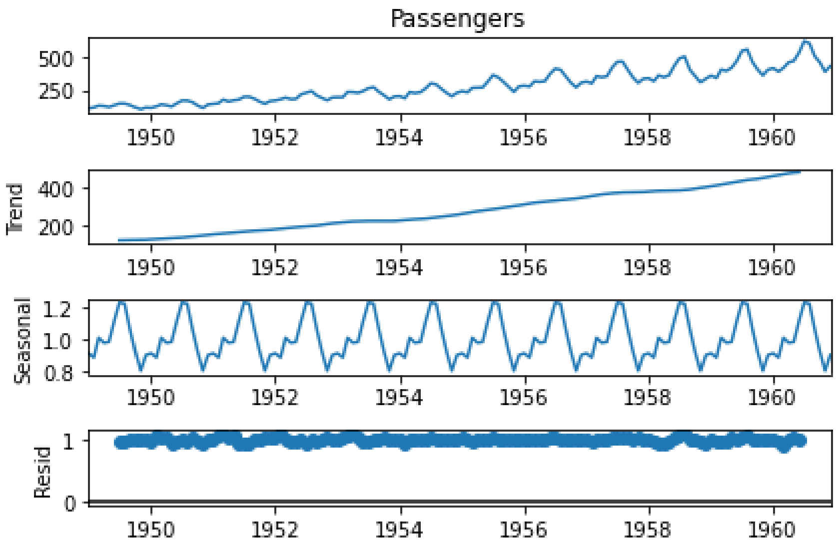
In [19]:

```
plt.figure(figsize = (15,5))
data['Passengers'].plot();
```



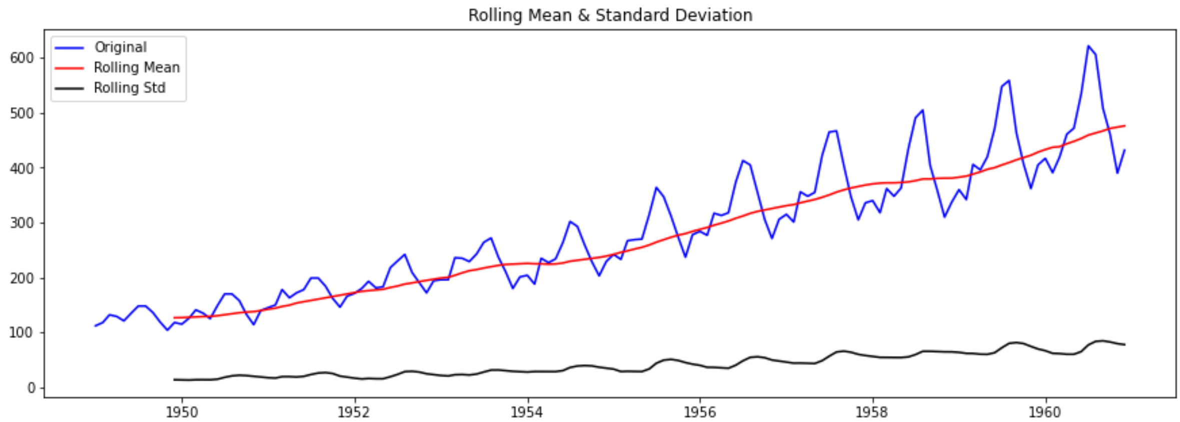
In [20]:

```
dec = sm.tsa.seasonal_decompose(data['Passengers'],period = 12, model = 'multiplicative').p
plt.show()
```



In [21]:

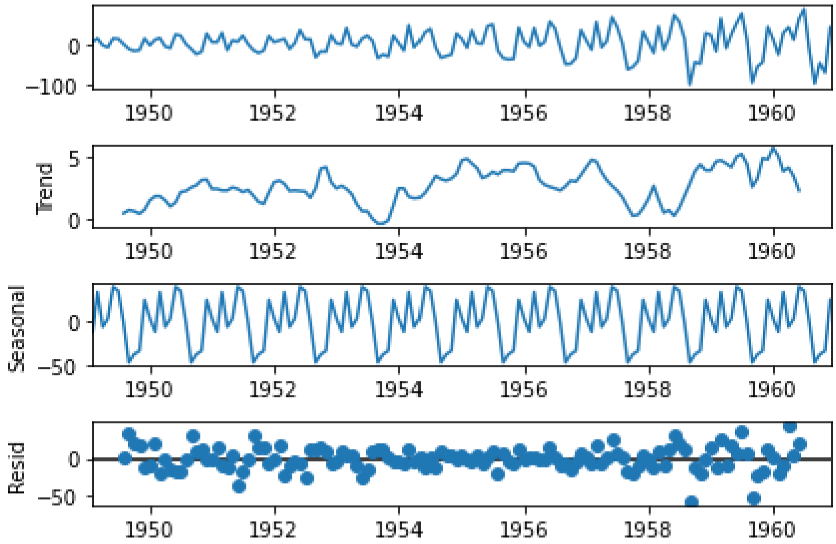
```
test_stationarity(data['Passengers'])
```




```
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
```

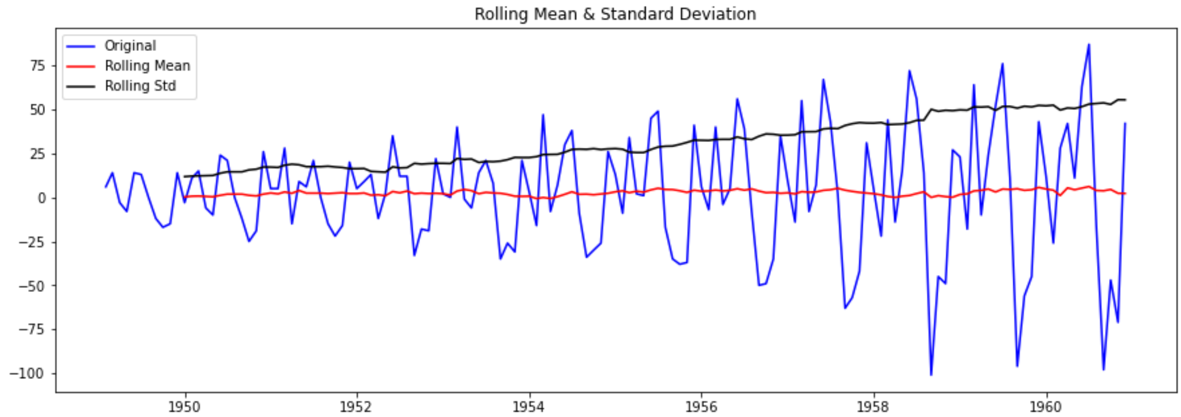
In [22]:

```
data_diff = data.diff()
data_diff = data_diff.dropna()

dec = sm.tsa.seasonal_decompose(data_diff,period = 12).plot()
plt.show()
```



In [23]:

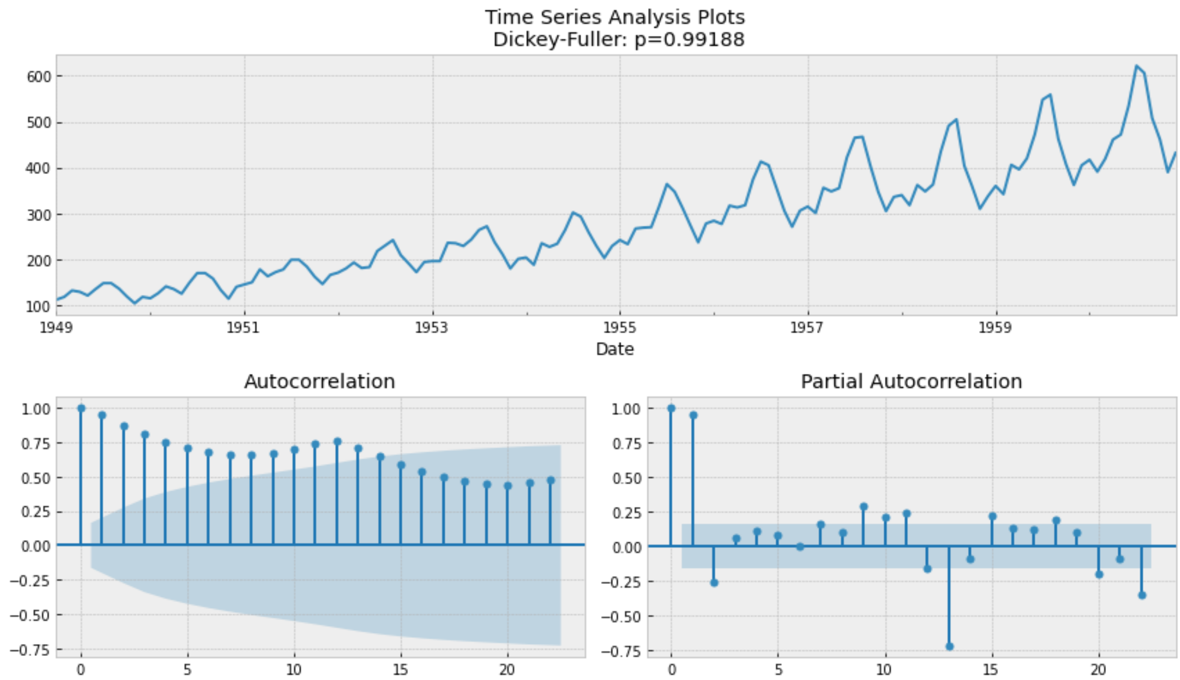
```
test_stationarity(data_diff)
```




```
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
```

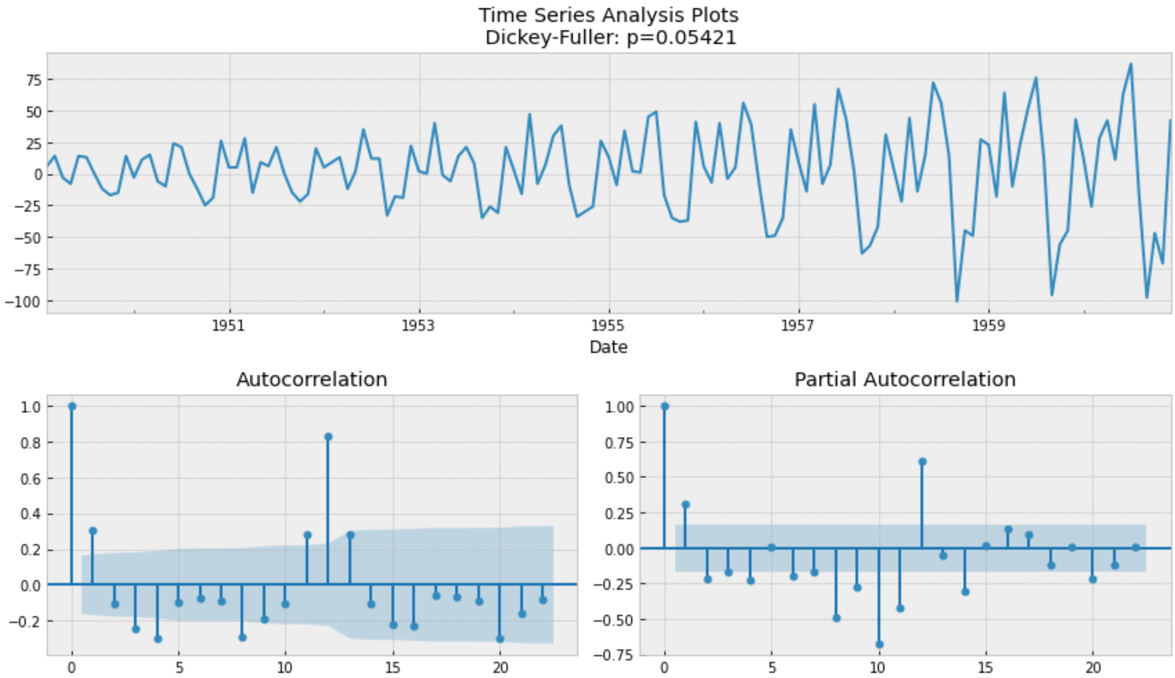
In [24]:

```
tsplot(data['Passengers'])
```



In [25]:

```
tsplot(data_diff['Passengers'])
```



In [26]:

```
model = ARIMA(data['Passengers'],order = (2,1,2))
model_fit = model.fit()
print(model_fit.summary())
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\arima_model.py:47
2: FutureWarning:
statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.ARIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (note the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 release.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace framework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until they are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARMA',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARIMA',
                        FutureWarning)

  warnings.warn(ARIMA_DEPRECATION_WARN, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:524: ValueWarning: No frequency information was provided, so inferred freq
uency MS will be used.
  warnings.warn('No frequency information was'
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:524: ValueWarning: No frequency information was provided, so inferred freq
uency MS will be used.
  warnings.warn('No frequency information was'

                             ARIMA Model Results
==========================================================================
====
Dep. Variable:          D.Passengers   No. Observations:
143
Model:                ARIMA(2, 1, 2)   Log Likelihood                -66
6.022
Method:                      css-mle   S.D. of innovations             2
4.714
Date:               Wed, 14 Sep 2022   AIC                           134
4.043
Time:                       10:06:41   BIC                           136
1.820
Sample:                   02-01-1949   HQIC                          135
1.267
                        - 12-01-1960
==========================================================================
============
                         coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------
------------
const                  2.5311      0.708      3.574      0.000       1.143
3.919
ar.L1.D.Passengers     1.6477      0.033     49.933      0.000       1.583
1.712
```

```
ar.L2.D.Passengers    -0.9094      0.033    -27.880      0.000      -0.973
-0.845
ma.L1.D.Passengers    -1.9098      0.065    -29.515      0.000      -2.037
-1.783
ma.L2.D.Passengers     0.9997      0.068     14.809      0.000       0.867
1.132
                                    Roots
==========================================================================
===
                  Real          Imaginary           Modulus         Freque
ncy
--------------------------------------------------------------------------
---
AR.1            0.9059           -0.5281j            1.0486           -0.0
840
AR.2            0.9059           +0.5281j            1.0486            0.0
840
MA.1            0.9552           -0.2965j            1.0002           -0.0
479
MA.2            0.9552           +0.2965j            1.0002            0.0
479
--------------------------------------------------------------------------
---
```

In [27]:

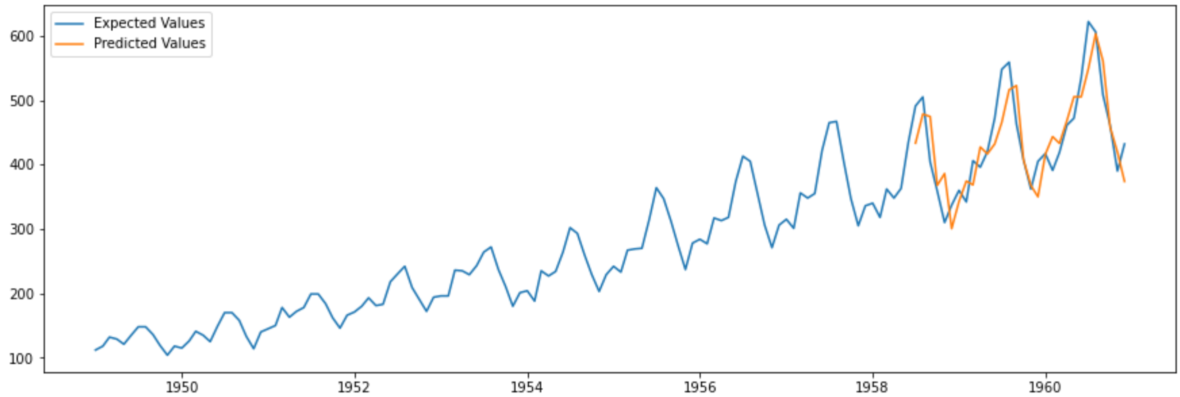
```
size = int(len(data) - 30)
train, test = data['Passengers'][0:size], data['Passengers'][size:len(data)]

print('\t ARIMA MODEL : In- Sample Forecasting \n')

history = [x for x in train]
predictions = []

for t in range(len(test)):

    model = ARIMA(history, order=(2,1,2))
    model_fit = model.fit(disp = 0)

    output = model_fit.forecast()
    yhat = output[0]
    predictions.append(float(yhat))

    obs = test[t]
    history.append(obs)

    print('predicted = %f, expected = %f' % (yhat, obs))
```

```
         ARIMA MODEL : In- Sample Forecasting 


C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\arima_model.py:47
2: FutureWarning: 
statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.ARIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (note the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 release.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace framework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until they are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARMA',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARIMA',
                        FutureWarning)

  warnings.warn(ARIMA_DEPRECATION_WARN, FutureWarning)

predicted = 433.256972, expected = 491.000000
predicted = 478.355854, expected = 505.000000
predicted = 474.552701, expected = 404.000000
predicted = 367.687051, expected = 359.000000
predicted = 386.046122, expected = 310.000000
predicted = 300.551728, expected = 337.000000
predicted = 342.709246, expected = 360.000000
predicted = 374.434495, expected = 342.000000
predicted = 368.418676, expected = 406.000000
predicted = 427.293772, expected = 396.000000
predicted = 416.580357, expected = 420.000000
predicted = 431.952427, expected = 472.000000
predicted = 465.574801, expected = 548.000000
```

```
predicted = 516.133868, expected = 559.000000
predicted = 522.642349, expected = 463.000000
predicted = 407.122831, expected = 407.000000
predicted = 367.581910, expected = 362.000000
predicted = 349.941636, expected = 405.000000
predicted = 415.817585, expected = 417.000000
predicted = 443.407937, expected = 391.000000
predicted = 432.877393, expected = 419.000000
predicted = 467.788485, expected = 461.000000
predicted = 505.289402, expected = 472.000000
predicted = 505.208366, expected = 535.000000
predicted = 548.678029, expected = 622.000000
predicted = 603.217422, expected = 606.000000
predicted = 560.803257, expected = 508.000000
predicted = 458.420863, expected = 461.000000
predicted = 419.481597, expected = 390.000000
predicted = 373.834768, expected = 432.000000
```

In [28]:

```
predictions_series = pd.Series(predictions, index = test.index)
fig,ax = plt.subplots(nrows = 1,ncols = 1,figsize = (15,5))

plt.subplot(1,1,1)
plt.plot(data['Passengers'],label = 'Expected Values')
plt.plot(predictions_series,label = 'Predicted Values');
plt.legend(loc="upper left")
plt.show()
```



In [29]:

```
error = np.sqrt(mean_squared_error(test,predictions))
print('Test RMSE: %.4f' % error)
```

```
Test RMSE: 42.5173
```

In [31]:

```
#our program ends here
#another method
#out of sampling forecasting
```

In [32]:

```
from pandas.tseries.offsets import DateOffset
future_dates = [data.index[-1] + DateOffset(weeks = x) for x in range(0,49)]

# New dataframe for storing the future values
df1 = pd.DataFrame(index = future_dates[1:],columns = data.columns)

forecast = pd.concat([data,df1])
forecast['ARIMA_Forecast_Function'] = np.NaN
forecast['ARIMA_Predict_Function'] = np.NaN
forecast.head()
```

Out[32]:

	Passengers	ARIMA_Forecast_Function	ARIMA_Predict_Function
1949-01-01	112	NaN	NaN
1949-02-01	118	NaN	NaN
1949-03-01	132	NaN	NaN
1949-04-01	129	NaN	NaN
1949-05-01	121	NaN	NaN

In [33]:

```
ARIMA_history_f = [x for x in train]
f1 = []

for t in range(len(df1)):

    model = ARIMA(ARIMA_history_f, order = (2,1,2))
    model_fit = model.fit(disp=0)

    output = model_fit.forecast()[0][0]

    ARIMA_history_f.append(output)
    f1.append(output)

for i in range(len(f1)):
    forecast.iloc[144 + i,1] = f1[i]
forecast.tail()
```

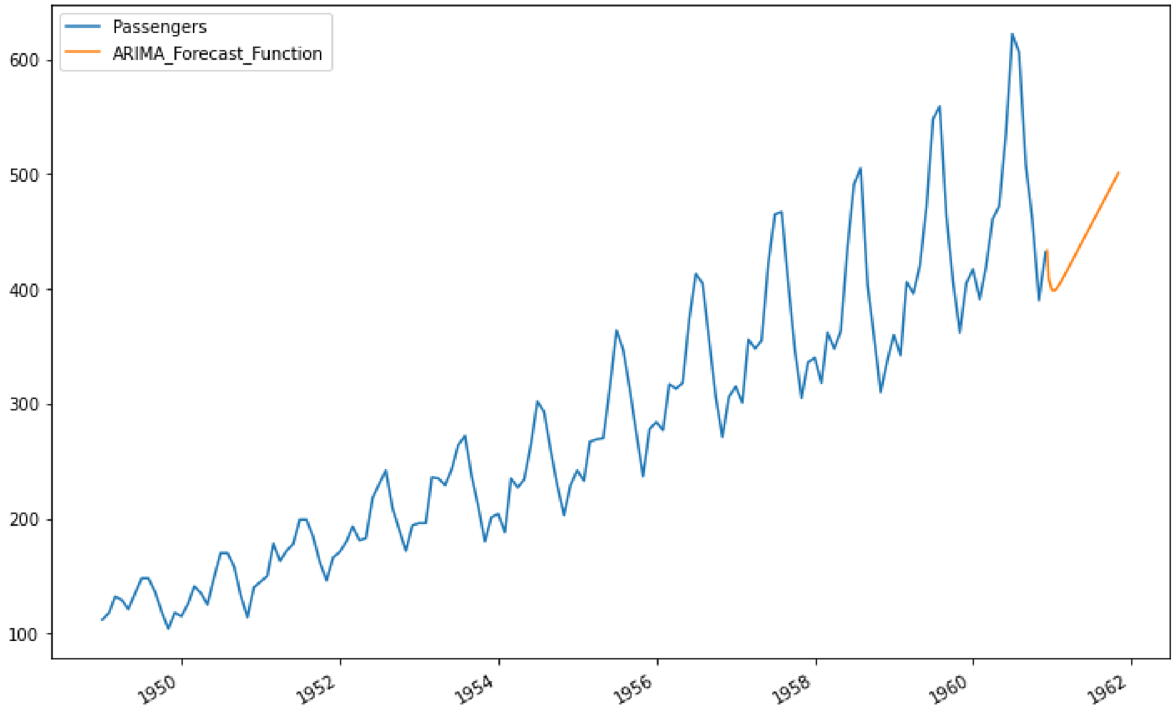
```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\arima_model.py:47
2: FutureWarning:
statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.ARIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (note the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 release.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace framework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until they are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARMA',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARIMA',
                        FutureWarning)

  warnings.warn(ARIMA_DEPRECATION_WARN, FutureWarning)
```

Out[33]:

	Passengers	ARIMA_Forecast_Function	ARIMA_Predict_Function
1961-10-05	NaN	490.80	NaN
1961-10-12	NaN	493.31	NaN
1961-10-19	NaN	495.81	NaN
1961-10-26	NaN	498.32	NaN
1961-11-02	NaN	500.83	NaN

In [34]:

```
forecast[['Passengers','ARIMA_Forecast_Function']].plot(figsize = (12,8));
```



In [35]:

```
ARIMA_history_p = [x for x in train]
f2 = []

for t in range(len(df1)):

    model = ARIMA(ARIMA_history_p, order = (2,1,2))
    model_fit = model.fit(disp=0)

    output = model_fit.predict(start = len(ARIMA_history_p),end = len(ARIMA_history_p),typ

    ARIMA_history_p.append(output)
    f2.append(output)

for i in range(len(f2)):
    forecast.iloc[144 + i,2] = f2[i]
forecast.tail()
```

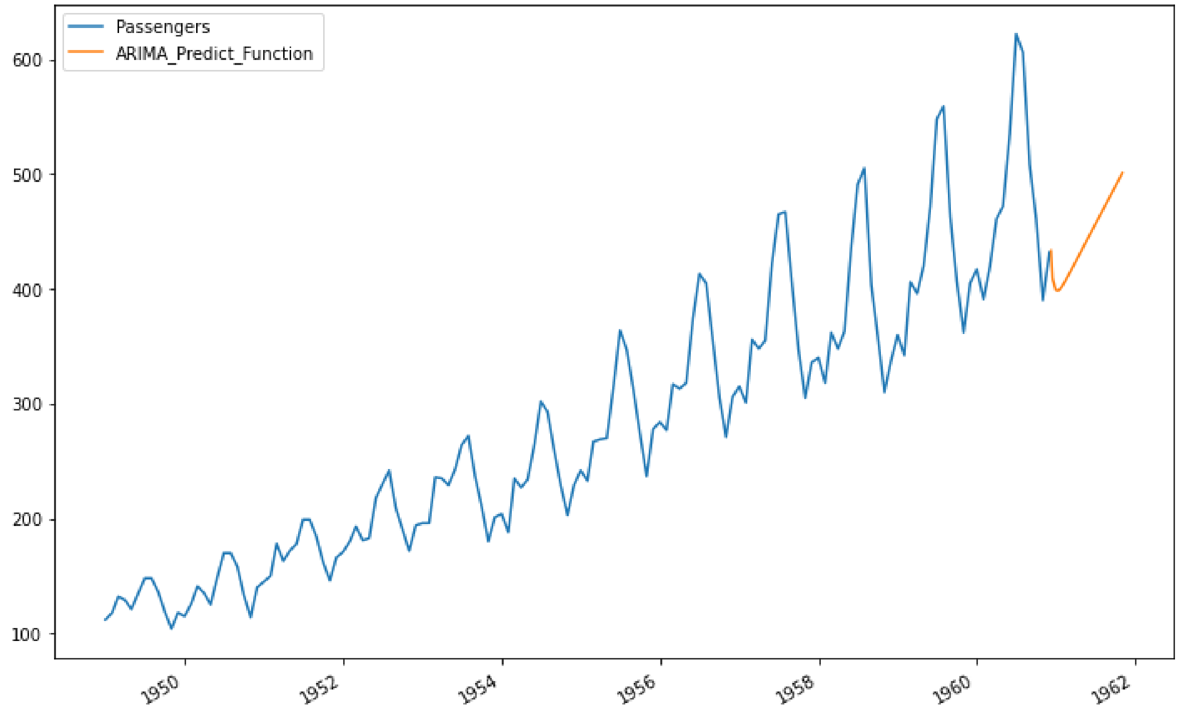
```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\arima_model.py:47
2: FutureWarning:
statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.ARIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (note the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 release.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace framework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until they are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARMA',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARIMA',
                        FutureWarning)

  warnings.warn(ARIMA_DEPRECATION_WARN, FutureWarning)
```

Out[35]:

	Passengers	ARIMA_Forecast_Function	ARIMA_Predict_Function
1961-10-05	NaN	490.80	490.80
1961-10-12	NaN	493.31	493.31
1961-10-19	NaN	495.81	495.81
1961-10-26	NaN	498.32	498.32
1961-11-02	NaN	500.83	500.83

In [36]:

```
forecast[['Passengers','ARIMA_Predict_Function']].plot(figsize = (12,8));
```



In [37]:

```
sum(f1) == sum(f2)
```

Out[37]:

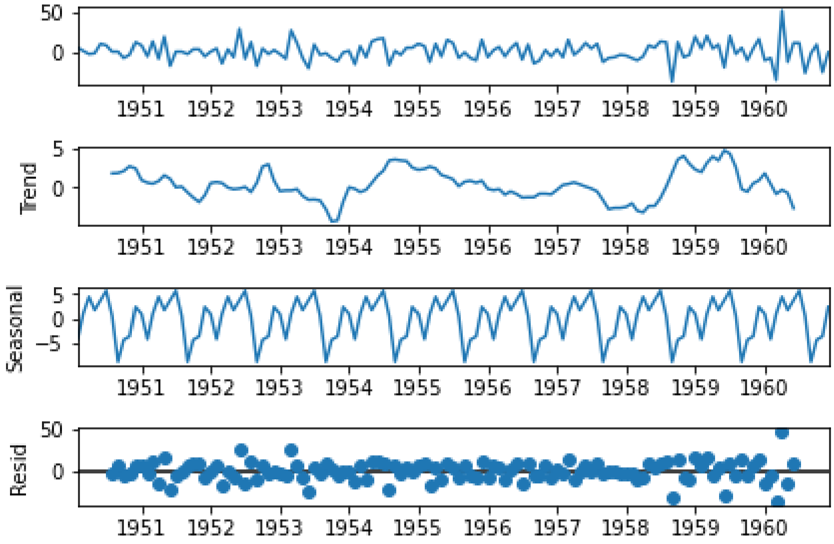
```
True
```

In [38]:

```
#sarimax model
#arima model ends here
```

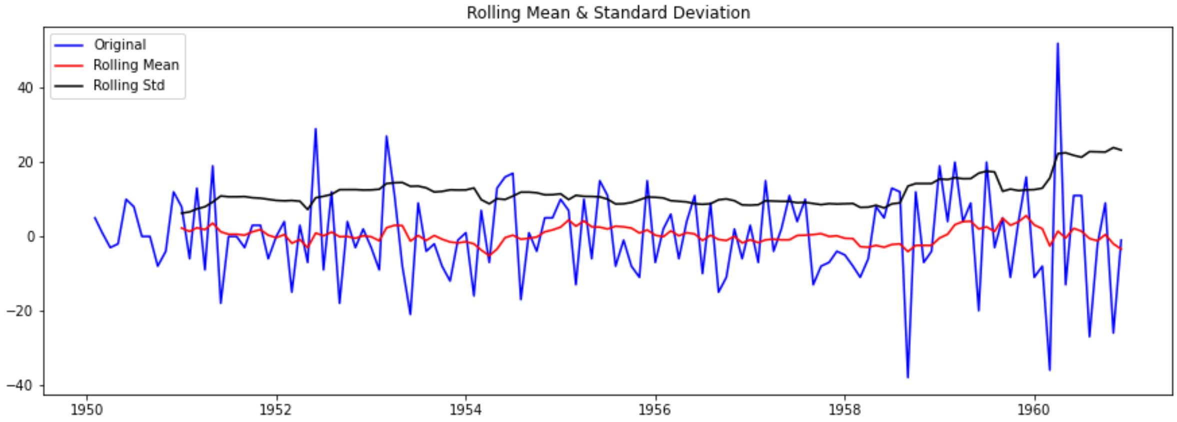
In [39]:

```
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()
```



In [40]:

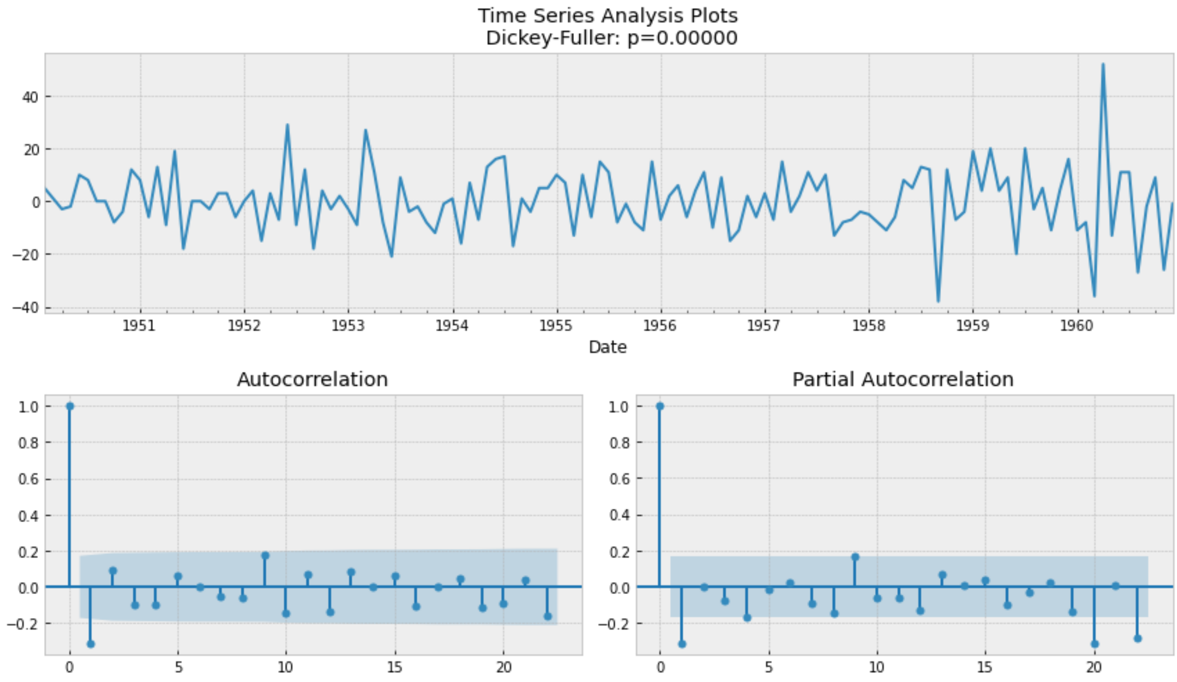
```
test_stationarity(data_diff_seas['Passengers'])
```




```
Results of Dickey-Fuller Test:
Test Statistic                -15.60
p-value                         0.00
#Lags Used                      0.00
Number of Observations Used   130.00
Critical Value (1%)            -3.48
Critical Value (5%)            -2.88
Critical Value (10%)           -2.58
dtype: float64
```

In [41]:

```
tsplot(data_diff_seas['Passengers'])
```



In [42]:

```
model = sm.tsa.statespace.SARIMAX(data['Passengers'],order = (2,1,2),seasonal_order = (0,1,
model_fit = model.fit()
print(model_fit.summary())
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:524: ValueWarning: No frequency information was provided, so inferred freq
uency MS will be used.
  warnings.warn('No frequency information was'
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:524: ValueWarning: No frequency information was provided, so inferred freq
uency MS will be used.
  warnings.warn('No frequency information was'
```

```
                                      SARIMAX Results                                      
==========================================================================
==================
Dep. Variable:                         Passengers   No. Observations:                    
144
Model:             SARIMAX(2, 1, 2)x(0, 1, [1], 12)   Log Likelihood                
-503.968
Date:                            Wed, 14 Sep 2022   AIC                           
1019.935
Time:                                    10:14:46   BIC                           
1037.186
Sample:                                01-01-1949   HQIC                          
1026.945
                                     - 12-01-1960                                         
Covariance Type:                              opg                                         
==========================================================================
====
                 coef    std err          z      P>|z|      [0.025      0.
975]
--------------------------------------------------------------------------
----
ar.L1          0.3966      0.422      0.940      0.347      -0.430      
1.223
ar.L2          0.3538      0.317      1.115      0.265      -0.268      
0.976
ma.L1         -0.7648      0.432     -1.769      0.077      -1.612      
0.083
ma.L2         -0.2060      0.414     -0.497      0.619      -1.018      
0.606
ma.S.L12      -0.1033      0.112     -0.921      0.357      -0.323      
0.117
sigma2       127.2934     14.232      8.944      0.000      99.400      15
5.187
==========================================================================
=========
Ljung-Box (L1) (Q):                   0.02   Jarque-Bera (JB):                
11.10
Prob(Q):                              0.88   Prob(JB):                         
0.00
Heteroskedasticity (H):               2.57   Skew:                             
0.05
Prob(H) (two-sided):                  0.00   Kurtosis:                         
4.42
```

```
==============================================================================
=========

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

In [43]:

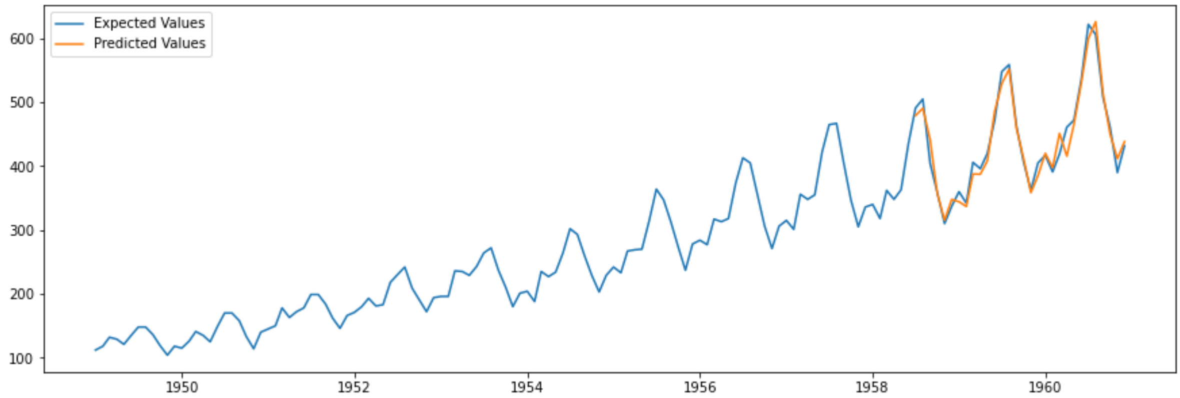
```
size = int(len(data) - 30)
train, test = data['Passengers'][0:size], data['Passengers'][size:len(data)]

print('\t SARIMA MODEL : In - Sample Forecasting \n')

history = [x for x in train]
predictions = []

for t in range(len(test)):

    model = sm.tsa.statespace.SARIMAX(history,order = (2,1,2),seasonal_order = (0,1,1,12))
    model_fit = model.fit()

    output = model_fit.forecast()
    yhat = output[0]
    predictions.append(float(yhat))

    obs = test[t]
    history.append(obs)

    print('predicted = %f, expected = %f' % (yhat, obs))
```

```
         SARIMA MODEL : In - Sample Forecasting

predicted = 479.084514, expected = 491.000000
predicted = 490.553512, expected = 505.000000
predicted = 441.276126, expected = 404.000000
predicted = 357.270516, expected = 359.000000
predicted = 315.251199, expected = 310.000000
predicted = 347.831240, expected = 337.000000

C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\base\model.py:566: Co
nvergenceWarning: Maximum Likelihood optimization failed to converge. Check
mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "

predicted = 344.251813, expected = 360.000000
predicted = 336.835601, expected = 342.000000
predicted = 387.593322, expected = 406.000000
predicted = 387.333485, expected = 396.000000
predicted = 408.192790, expected = 420.000000
predicted = 485.988157, expected = 472.000000
predicted = 529.031342, expected = 548.000000
predicted = 551.914007, expected = 559.000000
predicted = 459.061267, expected = 463.000000
predicted = 411.970100, expected = 407.000000
predicted = 358.421156, expected = 362.000000
predicted = 384.945724, expected = 405.000000
predicted = 420.143813, expected = 417.000000
predicted = 397.755392, expected = 391.000000
predicted = 451.335510, expected = 419.000000
predicted = 415.675504, expected = 461.000000
predicted = 465.295980, expected = 472.000000
predicted = 529.835404, expected = 535.000000
predicted = 599.299659, expected = 622.000000
predicted = 626.292199, expected = 606.000000
predicted = 513.891979, expected = 508.000000
predicted = 450.136741, expected = 461.000000
```

```
predicted = 411.653929, expected = 390.000000
predicted = 438.411433, expected = 432.000000
```

In [44]:

```
predictions_series = pd.Series(predictions, index = test.index)
fig,ax = plt.subplots(nrows = 1,ncols = 1,figsize = (15,5))

plt.subplot(1,1,1)
plt.plot(data['Passengers'],label = 'Expected Values')
plt.plot(predictions_series,label = 'Predicted Values');
plt.legend(loc="upper left")
plt.show()
```



In [45]:

```
error = np.sqrt(mean_squared_error(test,predictions))
print('Test RMSE: %.4f' % error)
```

```
Test RMSE: 16.9251
```

In [46]:

```
#so as we can see , using arima the rmse was 42.5173 but using sarimax you can see the
#the rmse is 16.9251 , just awesome
```

In [ ]:

```

```