Time Series Analysis

Auto-Regressive Integrated Moving Averages (ARIMA forecasting)

In [1]:

```
import pandas as pd
import numpy as np
from datetime import datetime
import matplotlib.pylab as plt
%matplotlib inline
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 15, 6
```

In [2]:

```
# Function to determin rolling statistics and perform Dickey-Fuller test
from statsmodels.tsa.stattools import adfuller
def test_stationarity(timeseries):

    #Determing rolling statistics (Window = 12 months in a year)
    rolmean = timeseries.rolling(window=12).mean()
    rolstd = timeseries.rolling(window=12).std()

    #Plot rolling statistics:
    orig = plt.plot(timeseries, 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)

    #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)
```

```
C:\Users\Ricardo\Anaconda3\lib\site-packages\statsmodels\compat\pandas.p
y:56: FutureWarning: The pandas.core.datetools module is deprecated and
will be removed in a future version. Please use the pandas.tseries modul
e instead.
  from pandas.core import datetools
```

Preparing Dataset

In [3]:

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

In [4]:

```
data.head()
```

Out[4]:

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

In [5]:

```
data.dtypes
```

Out[5]:

```
Month          object
#Passengers     int64
dtype: object
```

In [6]:

```
dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m')
data = pd.read_csv('AirPassengers.csv', parse_dates=['Month'], index_col='Month',dat
e_parser=dateparse)
data.head()
```

Out[6]:

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

In [7]:

```
data.index
```

Out[7]:

```
DatetimeIndex(['1949-01-01', '1949-02-01', '1949-03-01', '1949-04-01',
               '1949-05-01', '1949-06-01', '1949-07-01', '1949-08-01',
               '1949-09-01', '1949-10-01',
               ...
               '1960-03-01', '1960-04-01', '1960-05-01', '1960-06-01',
               '1960-07-01', '1960-08-01', '1960-09-01', '1960-10-01',
               '1960-11-01', '1960-12-01'],
              dtype='datetime64[ns]', name='Month', length=144, freq=Non
e)
```

In [8]:

```
# You can easily filter information having your time-series aa an index
ts = data['#Passengers']
ts['1949']
# Example 1: ts['1949-01']
# Example 2: ts['1949-01-01':'1949-05-01']
```

Out[8]:

```
Month
1949-01-01    112
1949-02-01    118
1949-03-01    132
1949-04-01    129
1949-05-01    121
1949-06-01    135
1949-07-01    148
1949-08-01    148
1949-09-01    136
1949-10-01    119
1949-11-01    104
1949-12-01    118
Name: #Passengers, dtype: int64
```

In [9]:

```
ts['1949-01-01':'1949-05-01']
```

Out[9]:

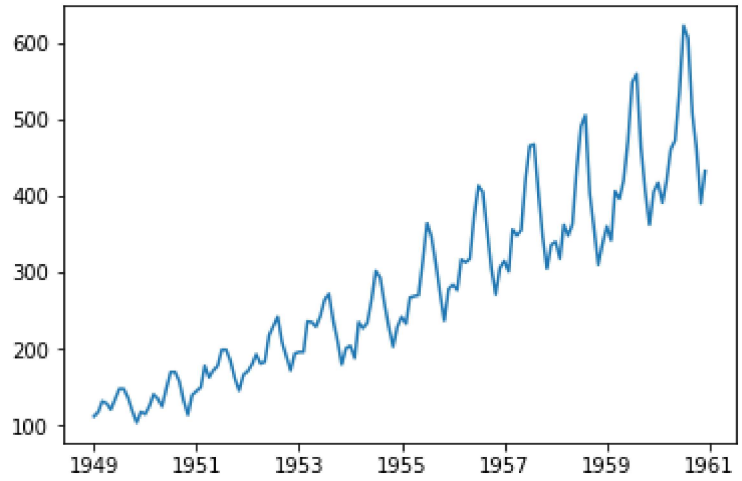
```
Month
1949-01-01    112
1949-02-01    118
1949-03-01    132
1949-04-01    129
1949-05-01    121
Name: #Passengers, dtype: int64
```

In [10]:

```
plt.plot(ts)
```

Out[10]:

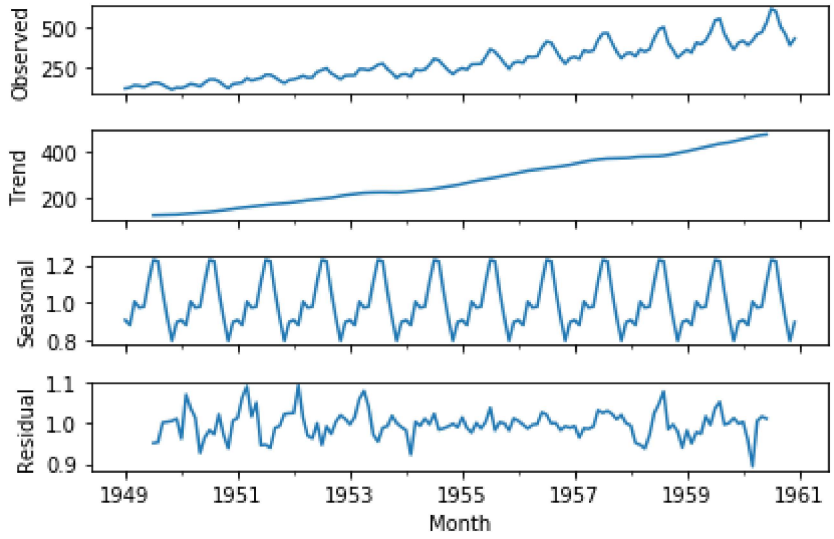
```
[<matplotlib.lines.Line2D at 0x276b6db9fd0>]
```



Multiplicative Model Decomposition Plot

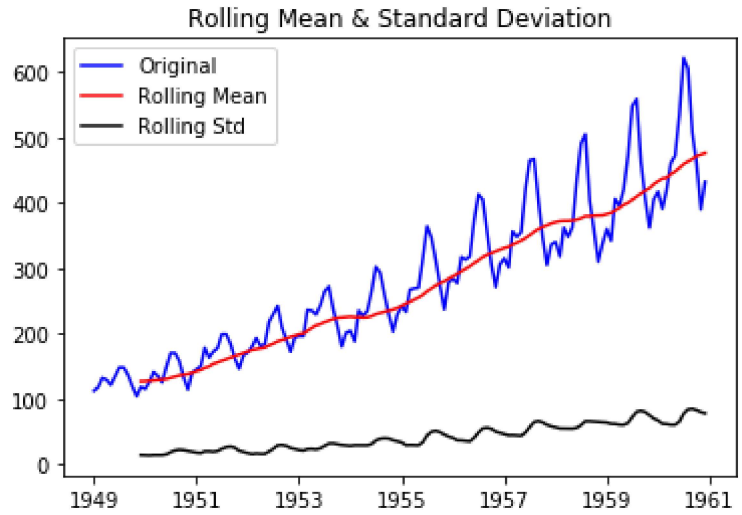
In [21]:

```
from matplotlib import pyplot
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(ts, model='multiplicative')
result.plot()
pyplot.show()
```



In [12]:

```
# rolling statistics and perform Dickey-Fuller test
test_stationarity(ts)
```




```
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
```

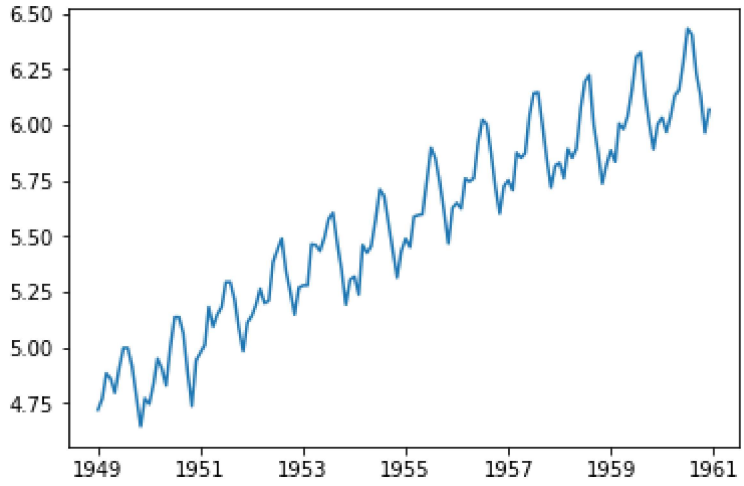
Estimating & Eliminating Trend

In [13]:

```
#Estimating trend
ts_logScale = np.log(ts)
plt.plot(ts_logScale)
```

Out[13]:

```
[<matplotlib.lines.Line2D at 0x276b7056278>]
```



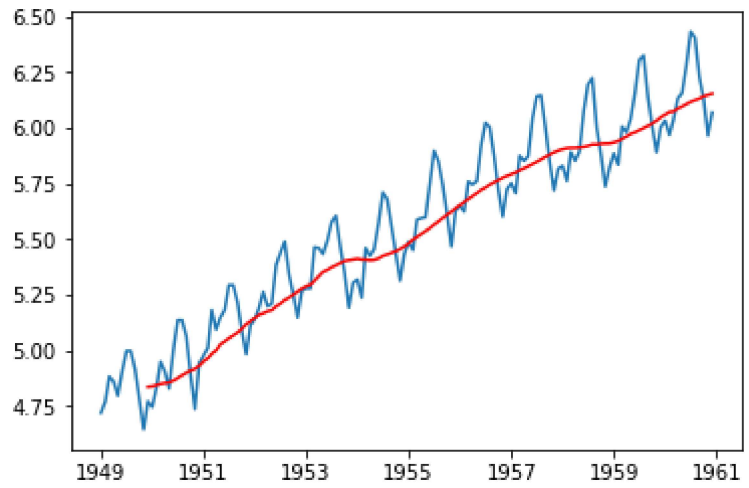
Moving Average

In [22]:

```
movingAverage = ts_logScale.rolling(window = 12).mean()
movingSTD = ts_logScale.rolling(window = 12).std()
plt.plot(ts_logScale)
plt.plot(movingAverage, color='red')
```

Out[22]:

```
[<matplotlib.lines.Line2D at 0x276b75d6cc0>]
```



In [23]:

```
ts_log_minus_movingAverage = ts_logScale - movingAverage
ts_log_minus_movingAverage.head(14)
```

Out[23]:

```
Month
1949-01-01         NaN
1949-02-01         NaN
1949-03-01         NaN
1949-04-01         NaN
1949-05-01         NaN
1949-06-01         NaN
1949-07-01         NaN
1949-08-01         NaN
1949-09-01         NaN
1949-10-01         NaN
1949-11-01         NaN
1949-12-01   -0.065494
1950-01-01   -0.093449
1950-02-01   -0.007566
Name: #Passengers, dtype: float64
```

In [24]:

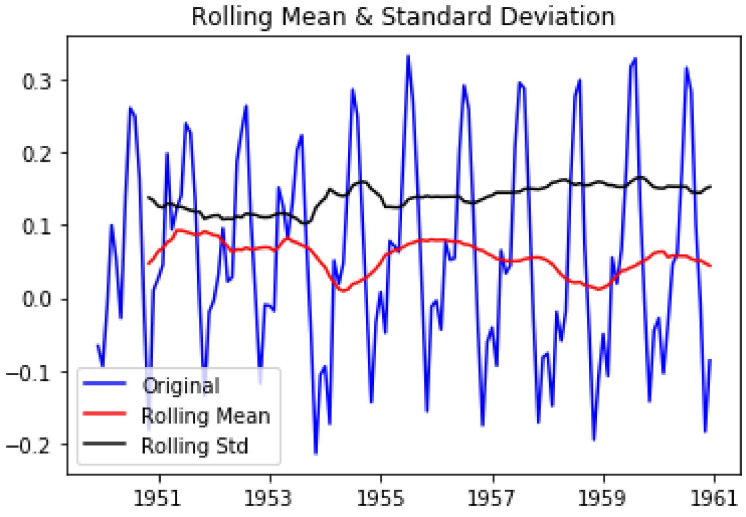
```
#Remove the NaN values
ts_log_minus_movingAverage.dropna(inplace=True)
ts_log_minus_movingAverage.head(14)
```

Out[24]:

```
Month
1949-12-01   -0.065494
1950-01-01   -0.093449
1950-02-01   -0.007566
1950-03-01    0.099416
1950-04-01    0.052142
1950-05-01   -0.027529
1950-06-01    0.139881
1950-07-01    0.260184
1950-08-01    0.248635
1950-09-01    0.162937
1950-10-01   -0.018578
1950-11-01   -0.180379
1950-12-01    0.010818
1951-01-01    0.026593
Name: #Passengers, dtype: float64
```

In [25]:

```
# rolling statistics and perform Dickey-Fuller test
test_stationarity(ts_log_minus_movingAverage)
```




```
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
Critical Value (10%)            -2.579896
dtype: float64
```

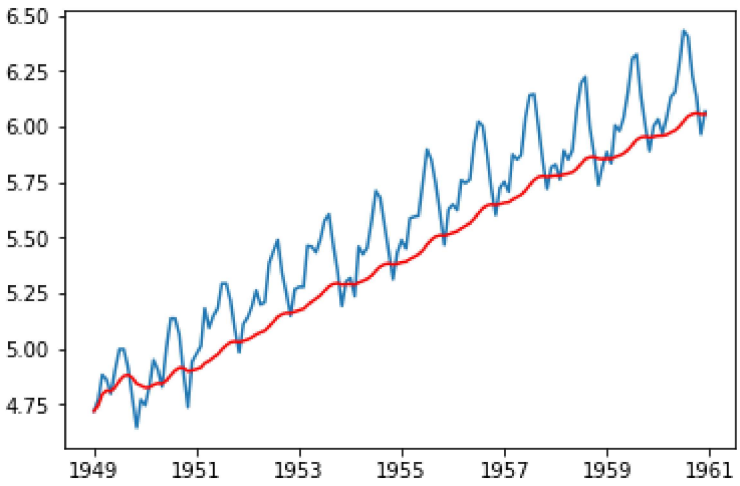
Exponentially weighted moving average

In [26]:

```
expwighted_avg = ts_logScale.ewm(halflife=12, min_periods=0, adjust=True).mean()
plt.plot(ts_logScale)
plt.plot(expwighted_avg, color='red')
```

Out[26]:

```
[<matplotlib.lines.Line2D at 0x276b75eb550>]
```

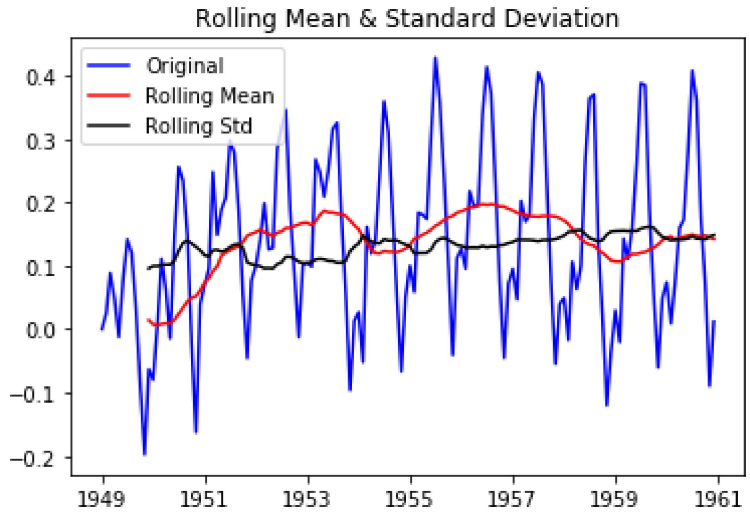


In [27]:

```
ts_log_ewm_diff = ts_logScale - expwighted_avg
```

In [28]:

```
# rolling statistics and perform Dickey-Fuller test
test_stationarity(ts_log_ewm_diff)
```




```
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
```

This TS has even lesser variations in mean and standard deviation in magnitude. Also, the test statistic is smaller than the 1% critical value, which is better than the previous case. Note that in this case there will be no missing values as all values from starting are given weights. So it'll work even with no previous values.
Source: https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/ (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/)

Eliminating Trend and Seasonality

The simple trend reduction techniques discussed before don't work in all cases, particularly the ones with high seasonality. Lets discuss two ways of removing trend and seasonality:

1. Differencing – taking the differece with a particular time lag
2. Decomposition – modeling both trend and seasonality and removing them from the model.

Differencing

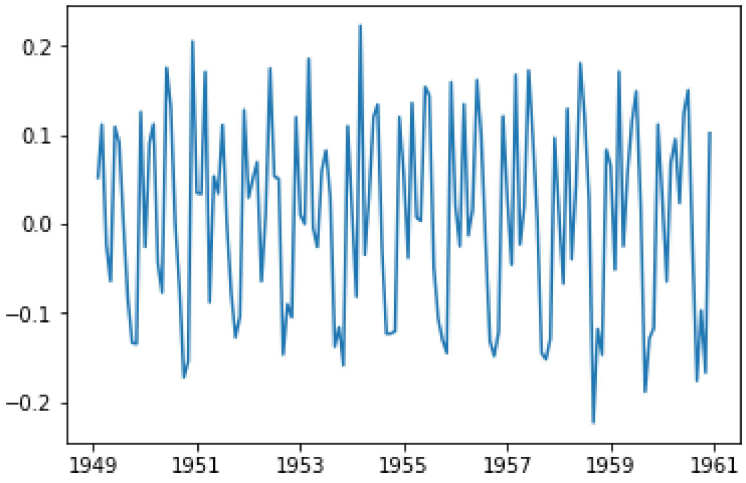
One of the most common methods of dealing with both trend and seasonality is differencing. In this technique, we take the difference of the observation at a particular instant with that at the previous instant. This mostly works well in improving stationarity. First order differencing can be done in Pandas as:

In [29]:

```
ts_log_diff = ts_logScale - ts_logScale.shift()
plt.plot(ts_log_diff)
```

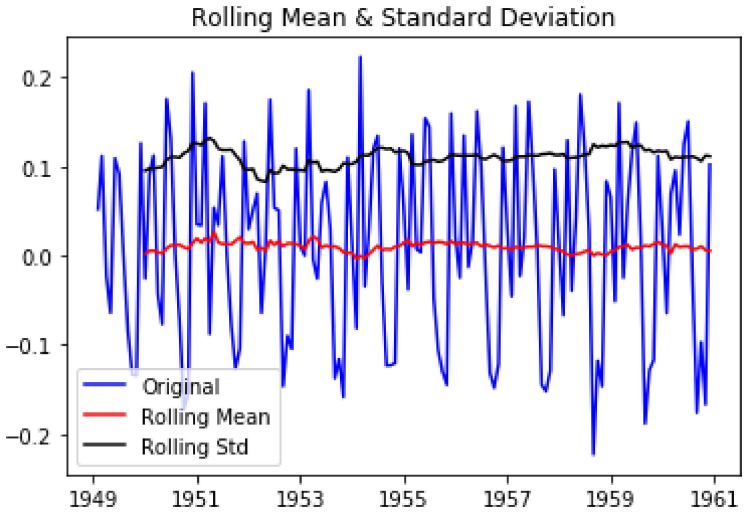
Out[29]:

```
[<matplotlib.lines.Line2D at 0x276b74dc0b8>]
```



In [30]:

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




```
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
```

Decomposing

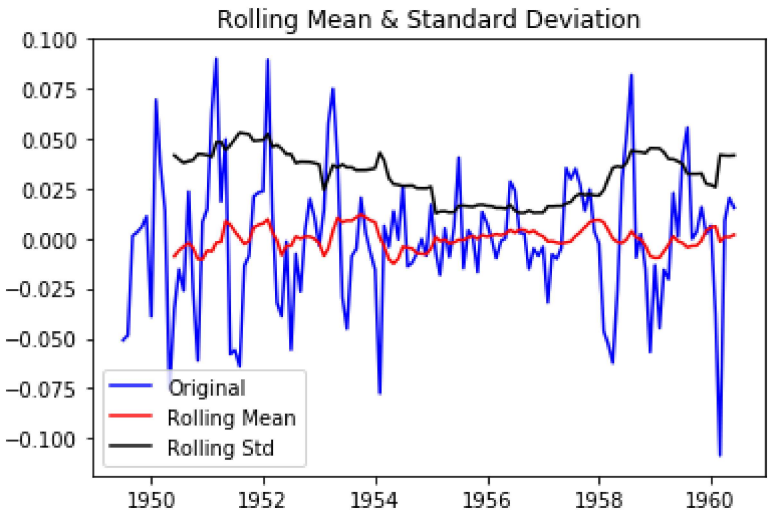
In [32]:

```
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(ts_logScale)

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

In [33]:

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




```
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
```

The Dickey-Fuller test statistic is significantly lower than the 1% critical value. So this TS is very close to stationary.

Forecasting a Time Series

ARIMA stands for **Auto-Regressive Integrated Moving Averages.** The ARIMA forecasting for a stationary time series is nothing but a linear (like a linear regression) equation. The predictors depend on the parameters (p,d,q) of the ARIMA model:

1. Number of AR (Auto-Regressive) terms (p): AR terms are just lags of dependent variable. For instance if p is 5, the predictors for x(t) will be x(t-1)….x(t-5).

2. Number of MA (Moving Average) terms (q): MA terms are lagged forecast errors in prediction equation. For instance if q is 5, the predictors for x(t) will be e(t-1)….e(t-5) where e(i) is the difference between the moving average at ith instant and actual value.

3. Number of Differences (d): These are the number of nonseasonal differences, i.e. in this case we took the first order difference. So either we can pass that variable and put d=0 or pass the original variable and put d=1. Both will generate same results.

Source: https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/ (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/)

We use two plots to determine the value of ‘p’ and ‘q’.

1. Autocorrelation Function (ACF): It is a measure of the correlation between the the TS with a lagged version of itself. For instance at lag 5, ACF would compare series at time instant ‘t1’…’t2’ with series at instant ‘t1-5’…’t2-5’ (t1-5 and t2 being end points).

2. Partial Autocorrelation Function (PACF): This measures the correlation between the TS with a lagged version of itself but after eliminating the variations already explained by the intervening comparisons. Eg at lag 5, it will check the correlation but remove the effects already explained by lags 1 to 4.

In [35]:

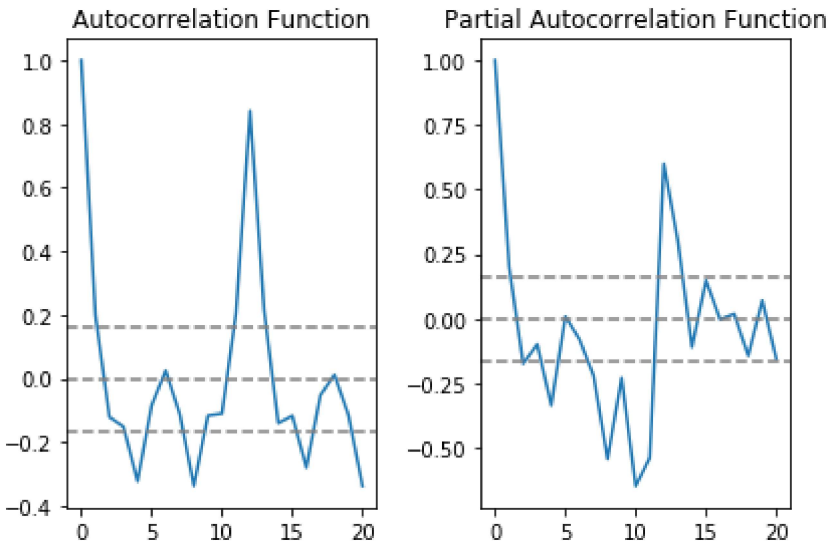
```
#ACF and PACF plots:
from statsmodels.tsa.stattools import acf, pacf

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

In [37]:

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



In this plot, the two dotted lines on either sides of 0 are the confidence interevals. These can be used to determine the ‘p’ and ‘q’ values as:

1. p – The lag value where the PACF chart crosses the upper confidence interval for the first time. If you notice closely, in this case p=2.
2. q – The lag value where the ACF chart crosses the upper confidence interval for the first time. If you notice closely, in this case q=2.

In [50]:

```
# Load the ARIMA model
from statsmodels.tsa.arima_model import ARIMA
```

The p,d,q values can be specified using the order argument of ARIMA which take a tuple (p,d,q).

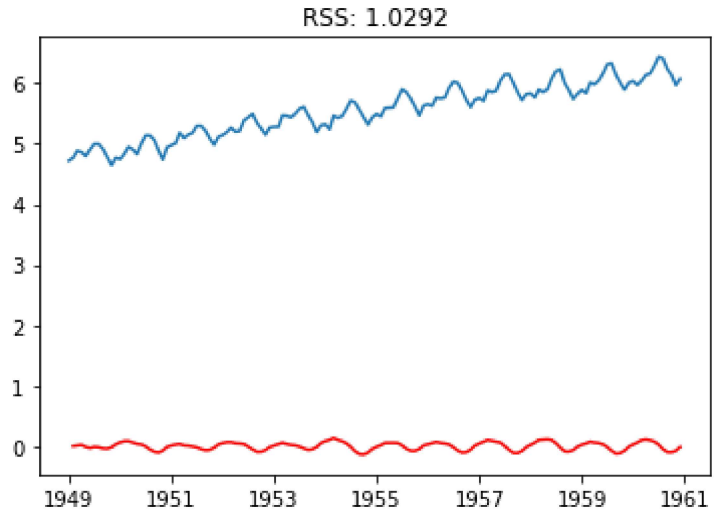
In [53]:

```
# Combined Model
model = ARIMA(ts_logScale, order=(2, 1, 2))
results_ARIMA = model.fit(disp=-1)
plt.plot(ts_logScale)
plt.plot(results_AR.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((results_ARIMA.fittedvalues-ts_log_diff)**2))
```

```
C:\Users\Ricardo\Anaconda3\lib\site-packages\statsmodels\tsa\kalmanf\kal
manfilter.py:646: FutureWarning: Conversion of the second argument of is
subdtype from `float` to `np.floating` is deprecated. In future, it will
be treated as `np.float64 == np.dtype(float).type`.
  if issubdtype(paramsdtype, float):
C:\Users\Ricardo\Anaconda3\lib\site-packages\statsmodels\tsa\kalmanf\kal
manfilter.py:650: FutureWarning: Conversion of the second argument of is
subdtype from `complex` to `np.complexfloating` is deprecated. In futur
e, it will be treated as `np.complex128 == np.dtype(complex).type`.
  elif issubdtype(paramsdtype, complex):
C:\Users\Ricardo\Anaconda3\lib\site-packages\statsmodels\tsa\kalmanf\kal
manfilter.py:577: FutureWarning: Conversion of the second argument of is
subdtype from `float` to `np.floating` is deprecated. In future, it will
be treated as `np.float64 == np.dtype(float).type`.
  if issubdtype(paramsdtype, float):
```

Out[53]:

```
Text(0.5,1,'RSS: 1.0292')
```



AR Model (2,1,0): RSS = 1.5023

MA Model (0,1,2): RSS = 1.4721

Combined Model (2,1,2): RSS = 1.0292

Taking it back to original scale

In [55]:

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

Out[55]:

```
Month
1949-02-01    0.009580
1949-03-01    0.017491
1949-04-01    0.027670
1949-05-01   -0.004521
1949-06-01   -0.023890
dtype: float64
```

In [57]:

```
# Calculate the cumulative sum
predictions_ARIMA_diff_cumsum = predictions_ARIMA_diff.cumsum()
predictions_ARIMA_diff_cumsum.head()
```

Out[57]:

```
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
dtype: float64
```

In [59]:

```
# First element is base number itself and from thereon the values cumulatively added
predictions_ARIMA_log = pd.Series(ts_logScale.iloc[0], index=ts_logScale.index)
predictions_ARIMA_log = predictions_ARIMA_log.add(predictions_ARIMA_diff_cumsum,fill
_value=0)
predictions_ARIMA_log.head()
```

Out[59]:

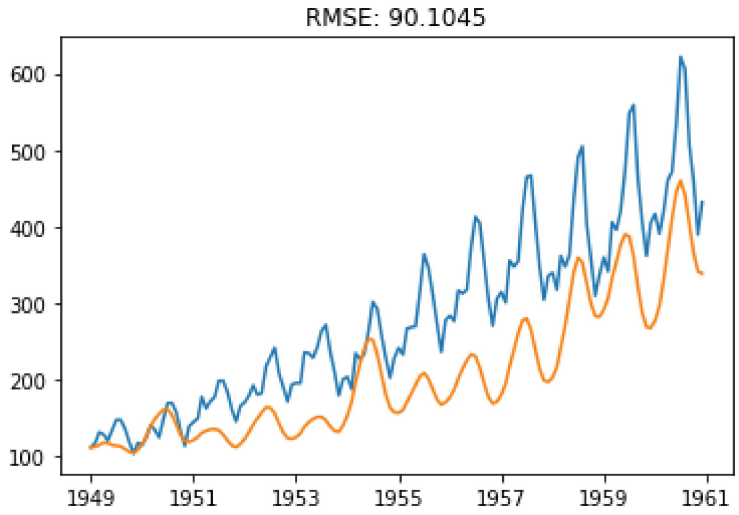
```
Month
1949-01-01    4.718499
1949-02-01    4.728079
1949-03-01    4.745570
1949-04-01    4.773241
1949-05-01    4.768720
dtype: float64
```

In [60]:

```
# Take the exponent and compare with the original series.
predictions_ARIMA = np.exp(predictions_ARIMA_log)
plt.plot(ts)
plt.plot(predictions_ARIMA)
plt.title('RMSE: %.4f'% np.sqrt(sum((predictions_ARIMA-ts)**2)/len(ts)))
```

Out[60]:

```
Text(0.5,1,'RMSE: 90.1045')
```



In [84]:

```
print('Number of rows: %d' % ts_logScale.shape[0])
print('Number of years: %d' % int(ts_logScale.shape[0]/12))
print('Number of years to predict: %d' % int(10))
print('Number of months in the serie + months to predict: %d' % int(ts_logScale.shap
e[0]+120))
```

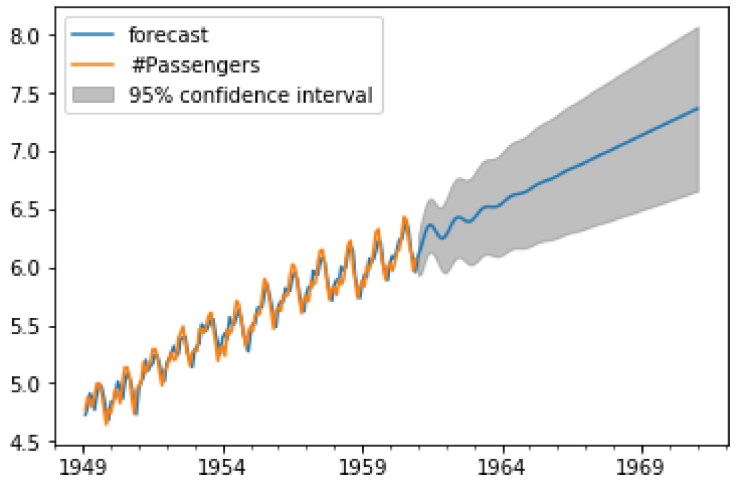
```
Number of rows: 144
Number of years: 12
Number of years to predict: 10
Number of months in the serie + months to predict: 264
```

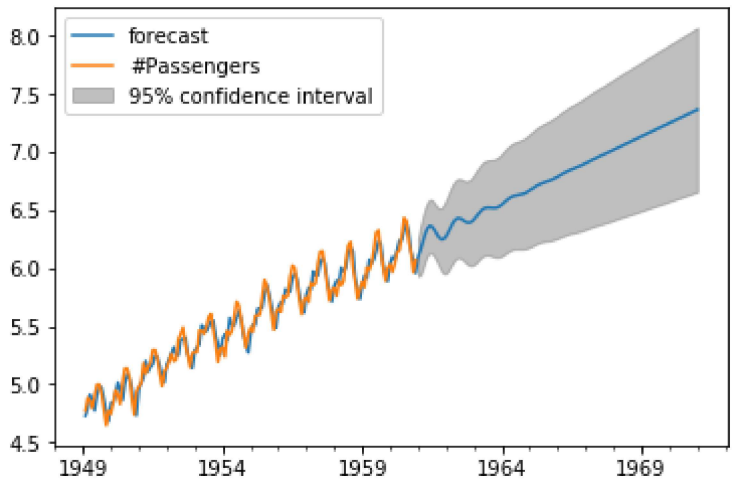
In [74]:

```
results_ARIMA.plot_predict(1,264)
```

```
C:\Users\Ricardo\Anaconda3\lib\site-packages\statsmodels\tsa\kalmanf\kal
manfilter.py:577: FutureWarning: Conversion of the second argument of is
subdtype from `float` to `np.floating` is deprecated. In future, it will
be treated as `np.float64 == np.dtype(float).type`.
  if issubdtype(paramsdtype, float):
```

Out[74]:





In [85]:

```
# The prediction in a table
results_ARIMA.forecast(steps = 120)
```

Out[85]:

```
(array([6.0955341 , 6.15281431, 6.22442995, 6.29241123, 6.34164724,
        6.36359353, 6.35784664, 6.33139277, 6.29597514, 6.26447697,
        6.24738322, 6.2502518 , 6.27275854, 6.30940333, 6.3515151 ,
        6.38988664, 6.41727379, 6.43011041, 6.4290667 , 6.41842468,
        6.40456138, 6.39403606, 6.39183082, 6.40019511, 6.41833705,
        6.4429542 , 6.46937445, 6.49293988, 6.51024114, 6.51989022,
        6.52267564, 6.52112373, 6.51864516, 6.51853684, 6.52311923,
        6.53322366, 6.54812877, 6.56591565, 6.58410313, 6.60036539,
        6.61313261, 6.62192772, 6.62737778, 6.63093119, 6.63438682,
        6.63937981, 6.64696505, 6.65739844, 6.67015329, 6.68414254,
        6.69806627, 6.71077899, 6.72157727, 6.73033976, 6.73749803,
        6.74386371, 6.75037318, 6.7578259 , 6.76668682, 6.77699824,
        6.78841321, 6.80032854, 6.81207193, 6.82308757, 6.83307131,
        6.84202487, 6.85022306, 6.85811216, 6.86617336, 6.87479149,
        6.88416302, 6.89426384, 6.90487889, 6.91567937, 6.92632194,
        6.93654137, 6.94621263, 6.95536933, 6.96417804, 6.97288009,
        6.98171967, 6.99087873, 7.00043527, 7.01035362, 7.02050589,
        7.0307156 , 7.04080964, 7.05066385, 7.06023078, 7.0695442 ,
        7.07870158, 7.08783151, 7.09705631, 7.10646022, 7.11607109,
        7.12585893, 7.13574994, 7.14565076, 7.15547525, 7.1651666 ,
        7.17470933, 7.18412902, 7.19348117, 7.20283338, 7.21224609,
        7.22175731, 7.23137492, 7.24107775, 7.25082423, 7.2605656 ,
        7.27025962, 7.27988103, 7.28942652, 7.29891327, 7.30837222,
        7.31783843, 7.32734135, 7.33689756, 7.34650778, 7.3561583 ]),
 array([0.08384711, 0.10749462, 0.11568695, 0.11702775, 0.11703496,
        0.11744018, 0.1176225 , 0.11778713, 0.12024162, 0.12736039,
        0.13870955, 0.15118786, 0.16157805, 0.16834385, 0.17177318,
        0.17311971, 0.17358721, 0.17385447, 0.17430205, 0.17543319,
        0.17788134, 0.18195709, 0.18726203, 0.19283378, 0.19769228,
        0.20130622, 0.20369037, 0.20519814, 0.2062525 , 0.2072098 ,
        0.20836952, 0.21000307, 0.21229724, 0.21524458, 0.21860158,
        0.22198235, 0.22503042, 0.22755272, 0.22954783, 0.23114774,
        0.23253801, 0.2339051 , 0.23541285, 0.23718287, 0.23926513,
        0.24161512, 0.24410345, 0.2465633 , 0.24885229, 0.25089601,
        0.25269705, 0.25431599, 0.25584283, 0.25737199, 0.2589836 ,
        0.2607282 , 0.26261518, 0.26461059, 0.26664867, 0.26865449,
        0.27056817, 0.27236064, 0.27403669, 0.27562741, 0.27717796,
        0.27873493, 0.28033557, 0.28199984, 0.28372648, 0.28549475,
        0.28727175, 0.2890231 , 0.29072301, 0.29236047, 0.29394004,
        0.29547839, 0.29699832, 0.29852239, 0.30006743, 0.30164089,
        0.30323975, 0.30485229, 0.30646217, 0.30805345, 0.30961501,
        0.31114298, 0.31264084, 0.31411756, 0.31558463, 0.31705285,
        0.31852966, 0.3200176 , 0.32151416, 0.32301296, 0.32450593,
        0.3259857 , 0.32744751, 0.32889024, 0.33031622, 0.33173028,
        0.33313818, 0.33454502, 0.33595403, 0.33736594, 0.33877905,
        0.34019   , 0.34159482, 0.34299013, 0.34437393, 0.34574608,
        0.34710804, 0.34846239, 0.34981198, 0.35115923, 0.35250551,
        0.35385092, 0.35519444, 0.35653432, 0.35786863, 0.35919584]),
 array([[5.93119677, 6.25987142],
        [5.94212872, 6.3634999 ],
        [5.9976877 , 6.4511722 ],
        [6.06304106, 6.5217814 ],
        [6.11226292, 6.57103155],
        [6.13341502, 6.59377205],
        [6.12731077, 6.58838251],
        [6.10053423, 6.5622513 ],
        [6.06030591, 6.53164438],
        [6.01485518, 6.51409875],
        [5.9755175 , 6.51924893],
```

```
[5.95392904, 6.54657456],
[5.95607137, 6.5894457 ],
[5.97945544, 6.63935121],
[6.01484585, 6.68818436],
[6.05057824, 6.72919504],
[6.07704911, 6.75749846],
[6.08936191, 6.77085892],
[6.08744095, 6.77069245],
[6.07458194, 6.76226742],
[6.05592035, 6.75320241],
[6.03740673, 6.7506654 ],
[6.02480399, 6.75885766],
[6.02224785, 6.77814237],
[6.03086729, 6.8058068 ],
[6.04840126, 6.83750715],
[6.07014865, 6.86860024],
[6.09075892, 6.89512085],
[6.10599367, 6.91448861],
[6.11376648, 6.92601396],
[6.11427889, 6.93107239],
[6.10952527, 6.93272218],
[6.1025502 , 6.93474011],
[6.09666521, 6.94040847],
[6.094668  , 6.95157046],
[6.09814625, 6.96830107],
[6.10707725, 6.9891803 ],
[6.11992051, 7.01191078],
[6.13419765, 7.03400861],
[6.14732415, 7.05340664],
[6.15736648, 7.06889873],
[6.16348216, 7.08037328],
[6.16597706, 7.08877849],
[6.16606131, 7.09580106],
[6.16543578, 7.10333785],
[6.16582288, 7.11293675],
[6.16853108, 7.12539901],
[6.17414325, 7.14065363],
[6.18241177, 7.15789481],
[6.19239539, 7.1758897 ],
[6.20278916, 7.19334339],
[6.2123288 , 7.20922917],
[6.22013454, 7.22302   ],
[6.22589994, 7.23477959],
[6.22989949, 7.24509656],
[6.23284583, 7.2548816 ],
[6.23565689, 7.26508947],
[6.23919868, 7.27645312],
[6.24406502, 7.28930861],
[6.25044511, 7.30355138],
[6.25810935, 7.31871707],
[6.26651149, 7.33414559],
[6.2749699 , 7.34917397],
[6.28286778, 7.36330736],
[6.28981249, 7.37633013],
[6.29571444, 7.38833529],
[6.30077543, 7.39967069],
[6.30540263, 7.41082168],
[6.31007968, 7.42226704],
[6.31523206, 7.43435092],
[6.32112073, 7.44720531],
[6.32778898, 7.4607387 ],
```

```
       [6.33507226, 7.47468552],
       [6.34266339, 7.48869536],
       [6.35021004, 7.50243384],
       [6.35741436, 7.51566837],
       [6.36410662, 7.52831863],
       [6.3702762 , 7.54046246],
       [6.37605668, 7.5522994 ],
       [6.3816748 , 7.56408538],
       [6.38738067, 7.57605867],
       [6.39337921, 7.58837824],
       [6.39978046, 7.60109008],
       [6.40657996, 7.61412729],
       [6.41367162, 7.62734017],
       [6.42088657, 7.64054464],
       [6.42804487, 7.65357442],
       [6.43500475, 7.66632295],
       [6.44169627, 7.67876529],
       [6.44813203, 7.69095637],
       [6.45439491, 7.70300825],
       [6.46060854, 7.71505449],
       [6.46690014, 7.72721248],
       [6.47336646, 7.73955398],
       [6.48005115, 7.75209103],
       [6.4869387 , 7.76477916],
       [6.49396461, 7.77753528],
       [6.50103774, 7.79026378],
       [6.50806735, 7.80288314],
       [6.51498719, 7.81534601],
       [6.5217705 , 7.82764816],
       [6.52843283, 7.8398252 ],
       [6.53502338, 7.85193897],
       [6.54160829, 7.86405846],
       [6.54825134, 7.87624083],
       [6.55499716, 7.88851746],
       [6.56186137, 7.90088848],
       [6.56882946, 7.91332604],
       [6.57586372, 7.92578474],
       [6.58291574, 7.93821547],
       [6.58994035, 7.95057888],
       [6.5969073 , 7.96285476],
       [6.60380763, 7.97504541],
       [6.61065382, 7.98717271],
       [6.61747412, 7.99927031],
       [6.62430338, 8.01137349],
       [6.63117304, 8.02350966],
       [6.63810314, 8.03569199],
       [6.64509815, 8.04791741],
       [6.6521474 , 8.06016921]]))
```

In [ ]:

```
```