Time_Series_AirPassenger

October 30, 2017

1 Steps to Tackle a Time Series Problem (with Codes in Python)

Note: These are just the codes from article

1.1 Loading and Handling TS in Pandas

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pylab as plt
        %matplotlib inline
        from matplotlib.pylab import rcParams
        rcParams['figure.figsize'] = 15, 6
```

```
In [2]: #Note: aim is not to teach stock price forecasting. It's a very complex domain and I hav
        data = pd.read_csv('AirPassengers.csv')
        print data.head()
        print '\n Data Types:'
        print data.dtypes
```

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

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

Reading as datetime format:

```
In [3]: dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m')
        # dateparse('1962-01')
        data = pd.read_csv('AirPassengers.csv', parse_dates='Month', index_col='Month',date_pars
        print data.head()
```

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

```
In [4]: #check datatype of index
        data.index
```

```
Out[4]: 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=u'Month', length=144, freq=None)
```

```
In [5]: #convert to time series:
        ts = data['#Passengers']
        ts.head(10)
```

```
Out[5]: 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
        Name: #Passengers, dtype: int64
```

1.1.1 Indexing TS arrays:

```
In [6]: #1. Specific the index as a string constant:
        ts['1949-01-01']
```

```
Out[6]: 112
```

```
In [7]: #2. Import the datetime library and use 'datetime' function:
        from datetime import datetime
        ts[datetime(1949,1,1)]
```

```
Out[7]: 112
```

2 Get range:

In [8]:
```
#1. Specify the entire range:
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
Name: #Passengers, dtype: int64
```

In [9]:
```
#2. Use ':' if one of the indices is at ends:
ts[:'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
```

Note: ends included here

In [10]:
```
#All rows of 1962:
ts['1949']
```

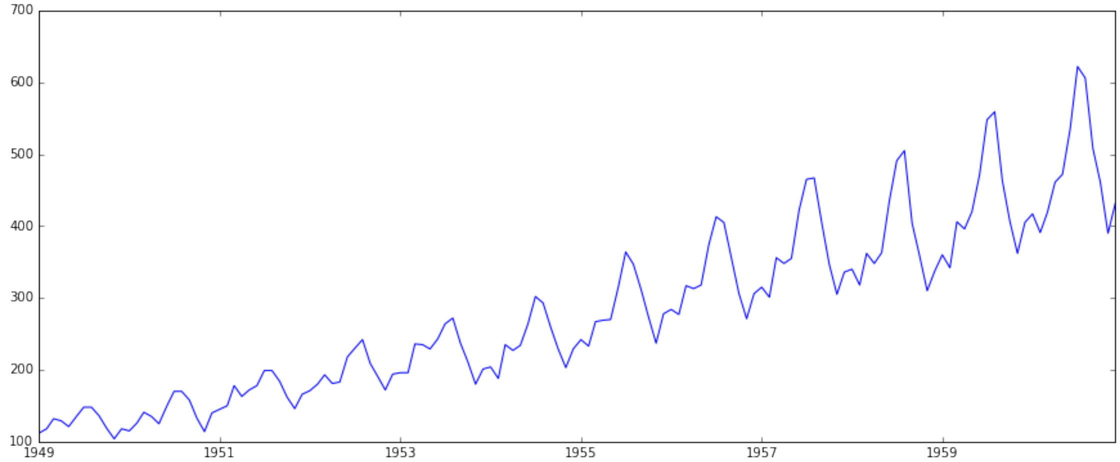
Out[10]:
```
Month
1949-01-01    112
1949-02-01    118
1949-03-01    132
1949-04-01    129
1949-05-01    121
1949-06-01    135
1949-07-01    148
1949-08-01    148
1949-09-01    136
1949-10-01    119
1949-11-01    104
1949-12-01    118
Name: #Passengers, dtype: int64
```

3 Checking for stationarity

3.1 Plot the time-series

In [11]:
```
plt.plot(ts)
```

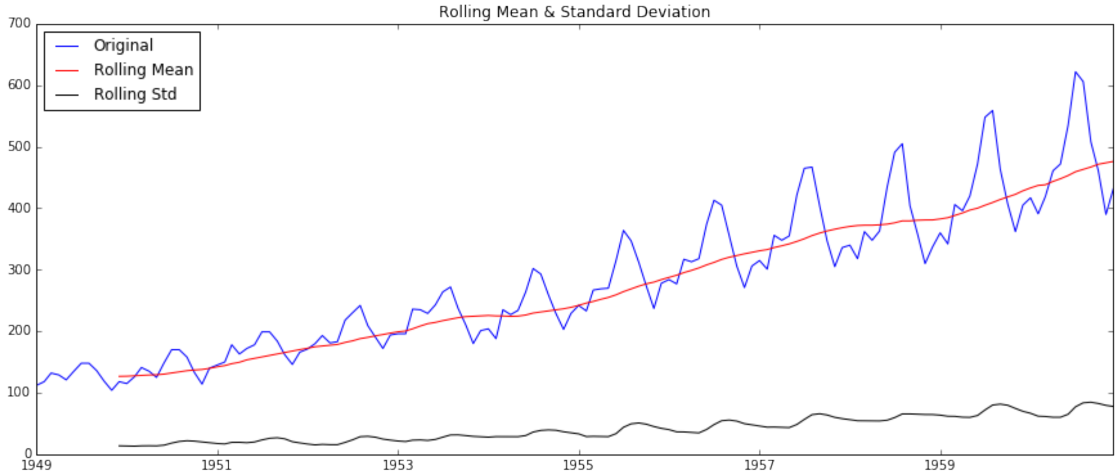
Out[11]: [<matplotlib.lines.Line2D at 0x106405e10>]



3.1.1 Function for testing stationarity

In [12]:
```
from statsmodels.tsa.stattools import adfuller
def test_stationarity(timeseries):

    #Determing rolling statistics
    rolmean = pd.rolling_mean(timeseries, window=12)
    rolstd = pd.rolling_std(timeseries, window=12)

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

In [13]:
```
test_stationarity(ts)
```



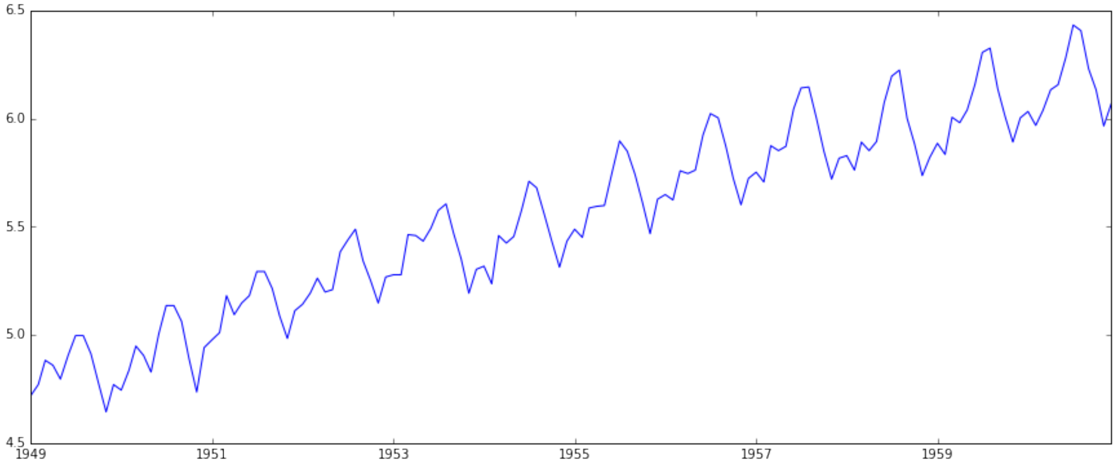

```
Results of Dickey-Fuller Test:
Test Statistic                   0.815369
p-value                          0.991880
#Lags Used                      13.000000
Number of Observations Used    130.000000
Critical Value (5%)             -2.884042
Critical Value (1%)             -3.481682
Critical Value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

4 Making TS Stationary

4.1 Estimating & Eliminating Trend

In [14]:
```
ts_log = np.log(ts)
plt.plot(ts_log)
```

Out[14]: [<matplotlib.lines.Line2D at 0x10e105210>]

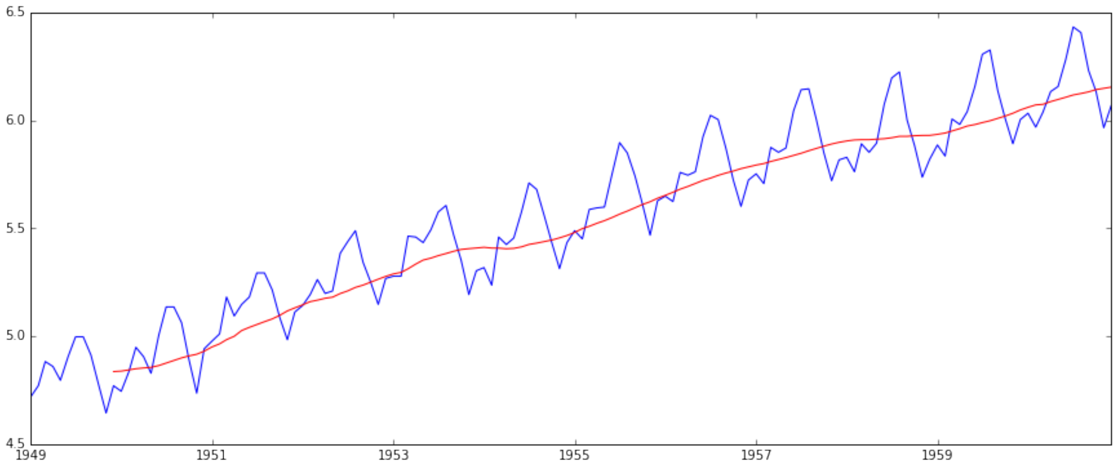


4.2 Smoothing:

4.2.1 Moving average

In [15]:
```
moving_avg = pd.rolling_mean(ts_log,12)
plt.plot(ts_log)
plt.plot(moving_avg, color='red')
```

Out[15]: [<matplotlib.lines.Line2D at 0x10e847b90>]



In [16]:
```
ts_log_moving_avg_diff = ts_log - moving_avg
ts_log_moving_avg_diff.head(12)
```

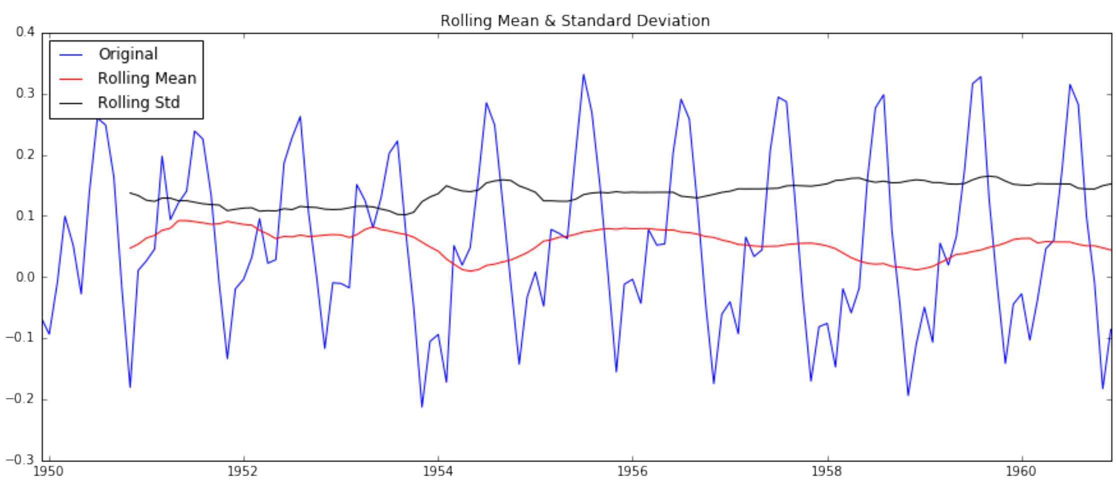
Out[16]:
```
Month
1949-01-01         NaN
```

```
        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
        Name: #Passengers, dtype: float64
```

```
In [17]: ts_log_moving_avg_diff.dropna(inplace=True)
         ts_log_moving_avg_diff.head()
```

```
Out[17]: 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
         Name: #Passengers, dtype: float64
```

```
In [18]: test_stationarity(ts_log_moving_avg_diff)
```



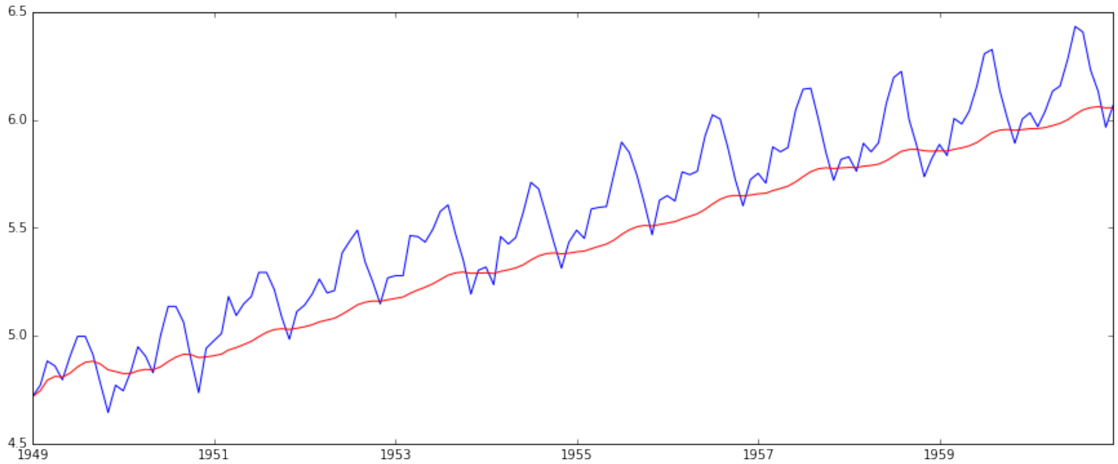

```
Results of Dickey-Fuller Test:
Test Statistic                  -3.162908
p-value                          0.022235
#Lags Used                      13.000000
Number of Observations Used    119.000000
```

```
Critical Value (5%)           -2.886151
Critical Value (1%)           -3.486535
Critical Value (10%)          -2.579896
dtype: float64
```

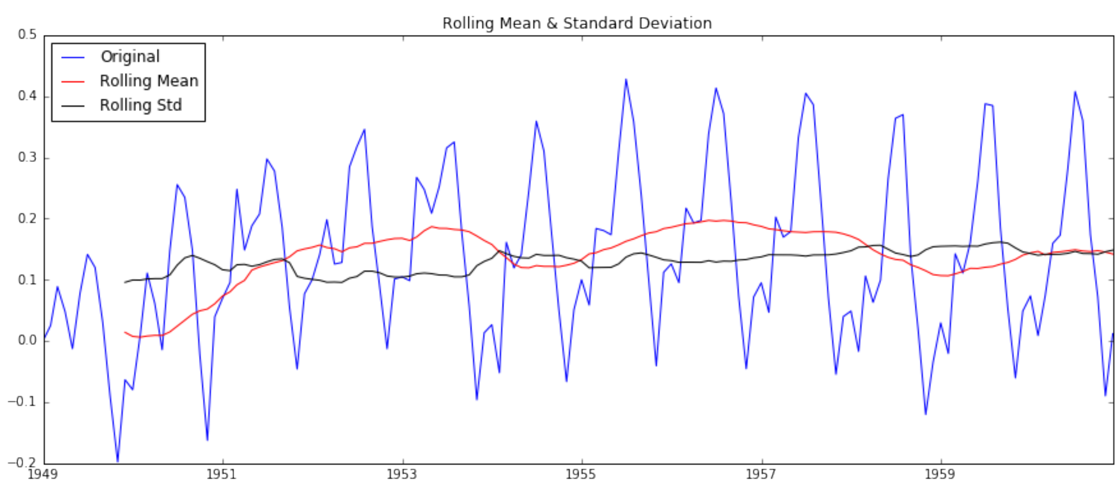
4.2.2 Exponentially Weighted Moving Average

In [19]:
```python
expwighted_avg = pd.ewma(ts_log, halflife=12)
plt.plot(ts_log)
plt.plot(expwighted_avg, color='red')
# expwighted_avg.plot(style='k--')
```

Out[19]:
```
[<matplotlib.lines.Line2D at 0x10ebfa150>]
```



In [20]:
```python
ts_log_ewma_diff = ts_log - expwighted_avg
test_stationarity(ts_log_ewma_diff)
```



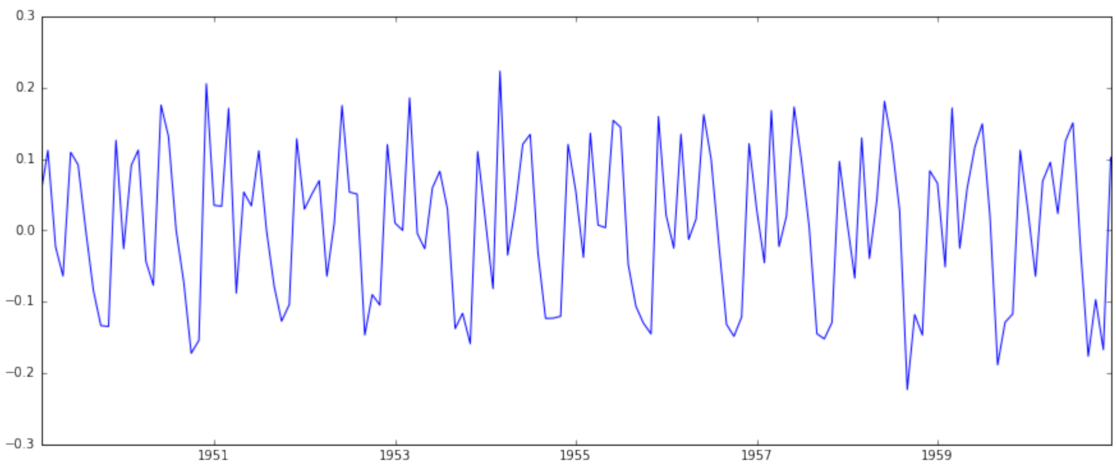
```
Results of Dickey-Fuller Test:
Test Statistic                  -3.601262
p-value                          0.005737
#Lags Used                      13.000000
Number of Observations Used    130.000000
Critical Value (5%)             -2.884042
Critical Value (1%)             -3.481682
Critical Value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

4.3 Eliminating Trend and Seasonality

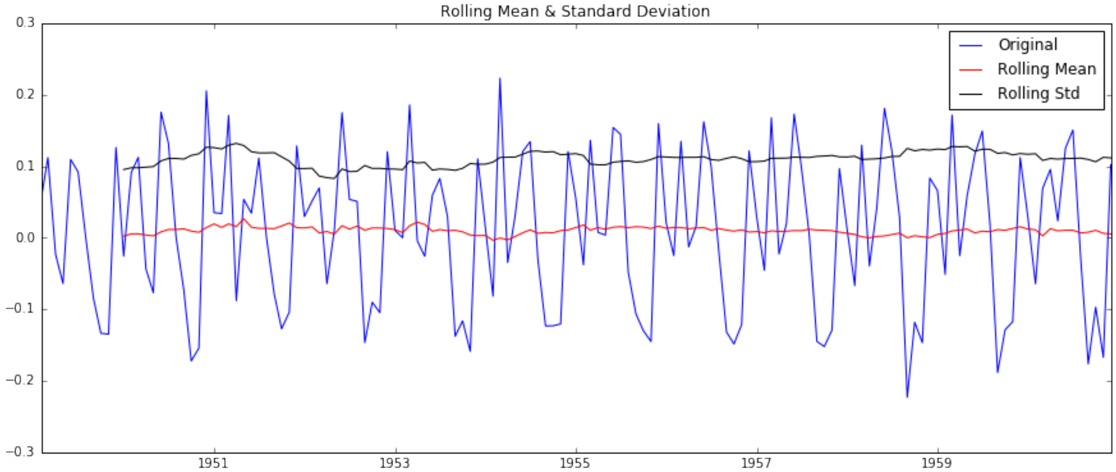
4.3.1 Differencing:

In [21]:
```
#Take first difference:
ts_log_diff = ts_log - ts_log.shift()
plt.plot(ts_log_diff)
```

Out[21]: [<matplotlib.lines.Line2D at 0x10ec4f250>]



In [22]:
```
ts_log_diff.dropna(inplace=True)
test_stationarity(ts_log_diff)
```

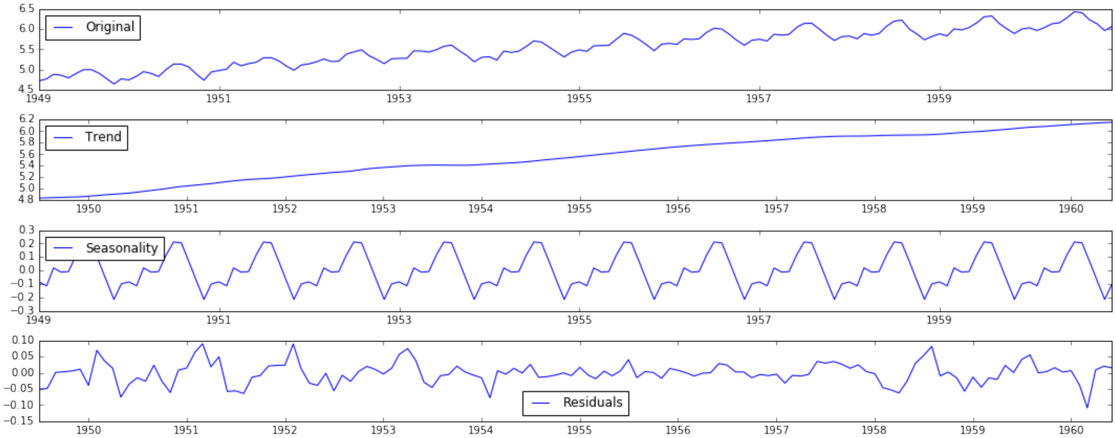



```
Results of Dickey-Fuller Test:
Test Statistic                  -2.717131
p-value                          0.071121
#Lags Used                      14.000000
Number of Observations Used    128.000000
Critical Value (5%)             -2.884398
Critical Value (1%)             -3.482501
Critical Value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

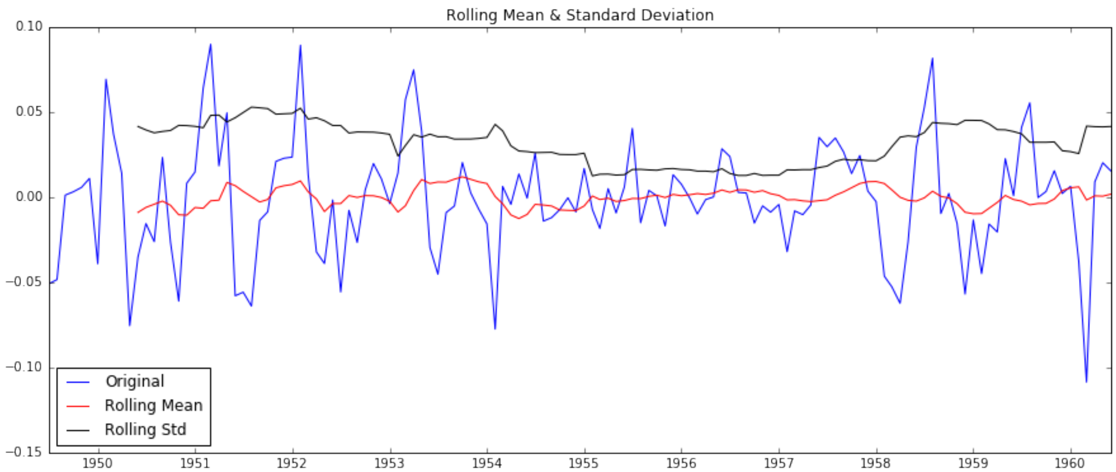
4.3.2 Decomposition:

```
In [23]: from statsmodels.tsa.seasonal import seasonal_decompose
         decomposition = seasonal_decompose(ts_log)

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

         plt.subplot(411)
         plt.plot(ts_log, label='Original')
         plt.legend(loc='best')
         plt.subplot(412)
         plt.plot(trend, label='Trend')
         plt.legend(loc='best')
         plt.subplot(413)
         plt.plot(seasonal,label='Seasonality')
         plt.legend(loc='best')
         plt.subplot(414)
         plt.plot(residual, label='Residuals')
```

```
plt.legend(loc='best')
plt.tight_layout()
```



In [24]:
```
ts_log_decompose = residual
ts_log_decompose.dropna(inplace=True)
test_stationarity(ts_log_decompose)
```



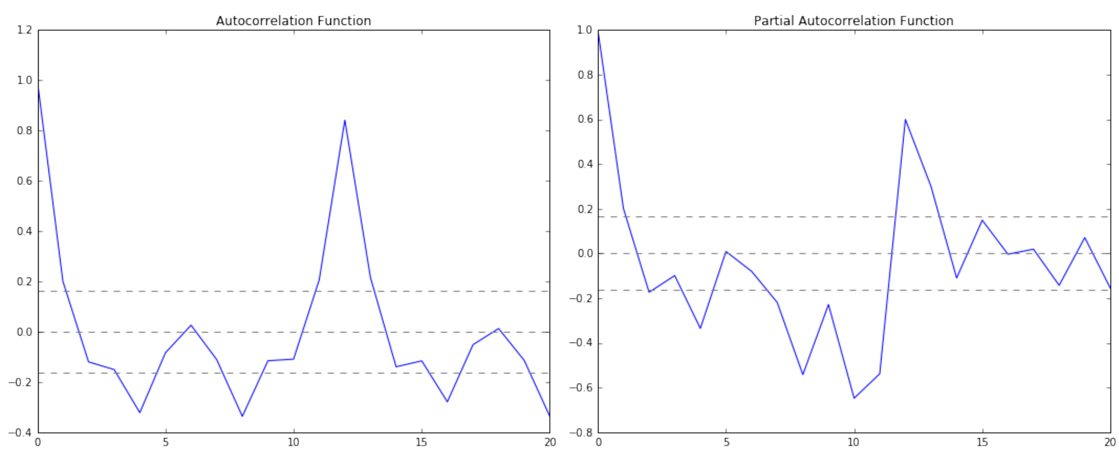

```
Results of Dickey-Fuller Test:
Test Statistic                -6.332387e+00
p-value                        2.885059e-08
#Lags Used                     9.000000e+00
Number of Observations Used    1.220000e+02
Critical Value (5%)           -2.885538e+00
Critical Value (1%)           -3.485122e+00
Critical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

5 Final Forecasting

```
In [25]: from statsmodels.tsa.arima_model import ARIMA
```

5.0.1 ACF & PACF Plots

```
In [26]: #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')

         #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()
```

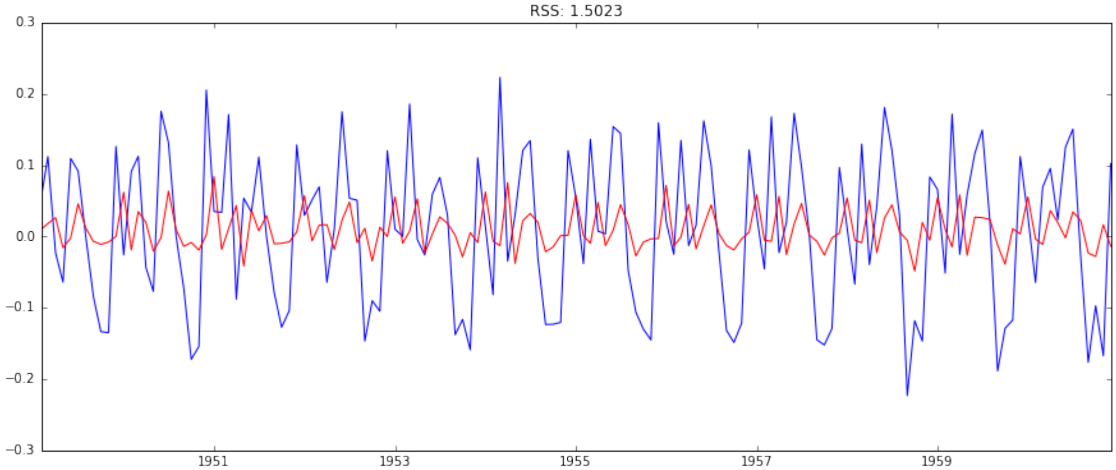


5.0.2 AR Model:

```
In [27]: #MA model:
         model = ARIMA(ts_log, order=(2, 1, 0))
```

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

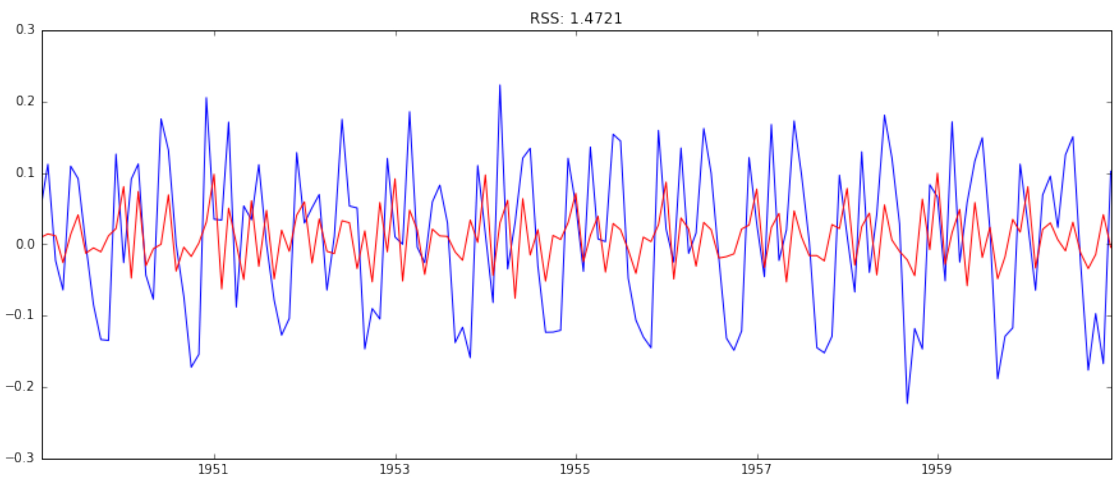
Out[27]: <matplotlib.text.Text at 0x1103cae50>



5.0.3 MA Model

In [28]:
```
model = ARIMA(ts_log, order=(0, 1, 2))
results_MA = model.fit(disp=-1)
plt.plot(ts_log_diff)
plt.plot(results_MA.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((results_MA.fittedvalues-ts_log_diff)**2))
```

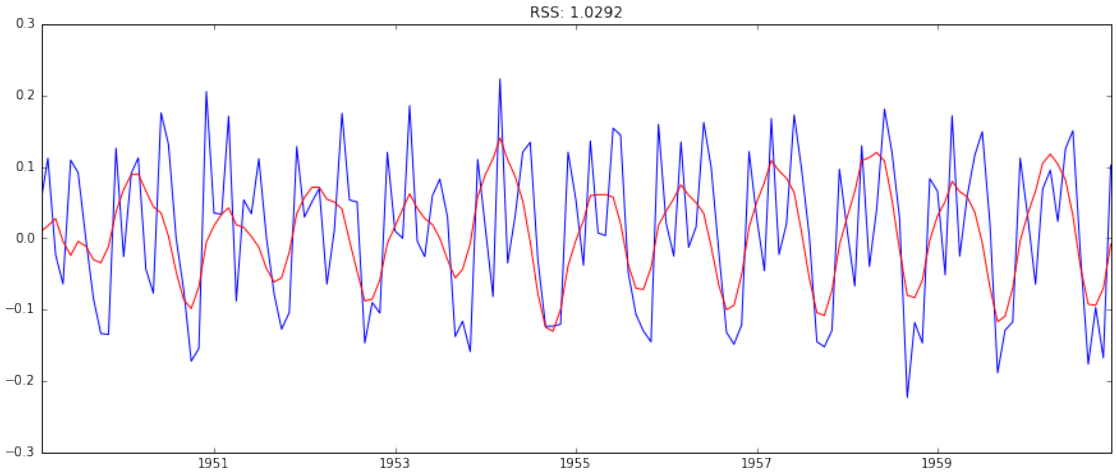
Out[28]: <matplotlib.text.Text at 0x1106b3e50>



5.0.4 ARIMA Model:

```
In [29]: model = ARIMA(ts_log, order=(2, 1, 2))
         results_ARIMA = model.fit(disp=-1)
         plt.plot(ts_log_diff)
         plt.plot(results_ARIMA.fittedvalues, color='red')
         plt.title('RSS: %.4f'% sum((results_ARIMA.fittedvalues-ts_log_diff)**2))
```

```
Out[29]: <matplotlib.text.Text at 0x11077b550>
```



5.0.5 Convert to original scale:

```
In [30]: predictions_ARIMA_diff = pd.Series(results_ARIMA.fittedvalues, copy=True)
         print predictions_ARIMA_diff.head()
```

```
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
```

```
In [31]: predictions_ARIMA_diff_cumsum = predictions_ARIMA_diff.cumsum()
         print predictions_ARIMA_diff_cumsum.head()
```

```
Month
1949-02-01    0.009580
1949-03-01    0.027071
1949-04-01    0.054742
1949-05-01    0.050221
```

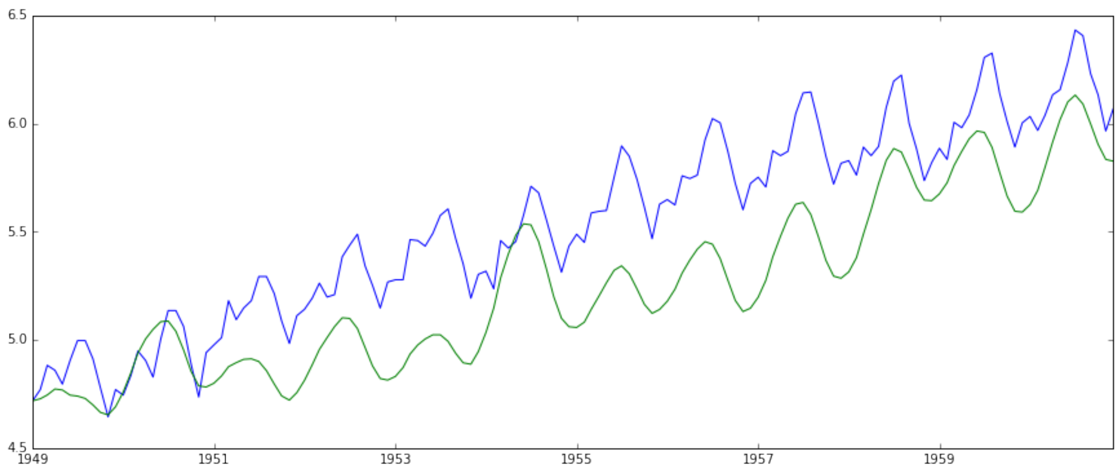
```
1949-06-01    0.026331
dtype: float64
```

In [32]:
```
predictions_ARIMA_log = pd.Series(ts_log.ix[0], index=ts_log.index)
predictions_ARIMA_log = predictions_ARIMA_log.add(predictions_ARIMA_diff_cumsum,fill_va
predictions_ARIMA_log.head()
```

Out[32]:
```
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
```

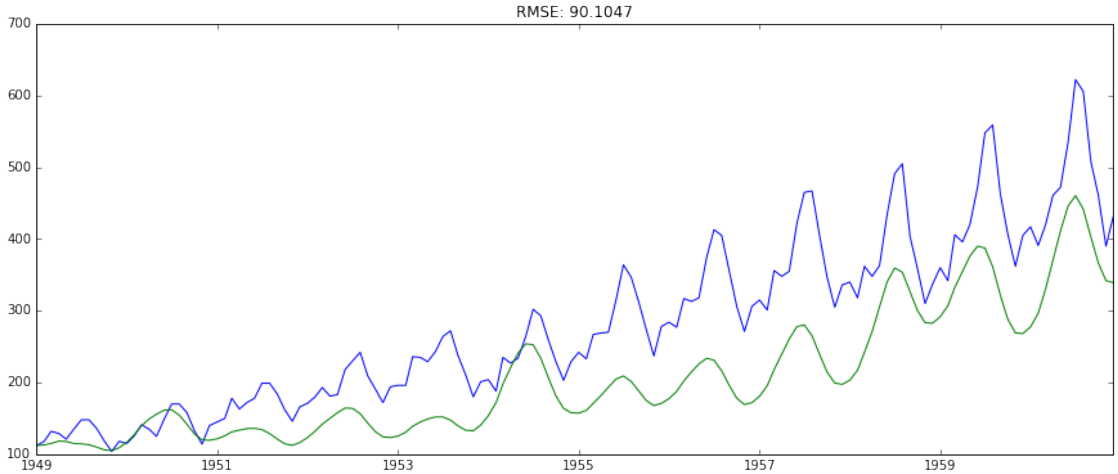
In [33]:
```
plt.plot(ts_log)
plt.plot(predictions_ARIMA_log)
```

Out[33]:
```
[<matplotlib.lines.Line2D at 0x1106756d0>]
```



In [34]:
```
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[34]:
```
<matplotlib.text.Text at 0x110a6a550>
```



RMSE: 90.1047
700
600
500
400
300
200
100
1949
1951
1953
1955
1957
1959