**1. Exploring and analyse the time series data.

1. Forecasting using ARIMA, ARIMAX**

In [1]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O
 (e.g. pd.read_csv)
import numpy as np
import datetime as dt
import matplotlib.pyplot as plt

from subprocess import check_output
print(check_output(["ls", "../input"]).decode("utf8"
))

# Any results you write to the current directory are s
aved as output.
```

```
AirPassengers.csv
```

In [2]:

```
dateparse = lambda d : dt.datetime.strptime(d, '%Y-%
m')
data = pd.read_csv("../input/AirPassengers.csv", inde
x_col='Month',date_parser=dateparse)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 144 entries, 1949-01-01 t
o 1960-12-01
Data columns (total 1 columns):
#Passengers    144 non-null int64
dtypes: int64(1)
memory usage: 2.2 KB
```

In [3]:

```
data.head()
```

Out[3]:

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

In [4]:

```
data.describe()
```

Out[4]:

	#Passengers
count	144.000000
mean	280.298611
std	119.966317
min	104.000000
25%	180.000000
50%	265.500000
75%	360.500000
max	622.000000

In [5]:

```
#x = [dt.datetime.strptime(d,'%Y-%m') for d in data.index]
#data.set_index = [ str(d.year)+"-"+str(d.month)+"-"+str(d.day) for d in x]
data.index
```

Out[5]:

```
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', '1949-10-0
```

```
1',
               ...
               '1960-03-01', '1960-04-0
1', '1960-05-01', '1960-06-01',
               '1960-07-01', '1960-08-0
1', '1960-09-01', '1960-10-01',
               '1960-11-01', '1960-12-0
1'],
              dtype='datetime64[ns]', na
me='Month', length=144, freq=None)
```

In [6]:

```
data['#Passengers']['1949-01-01']
```

Out[6]:

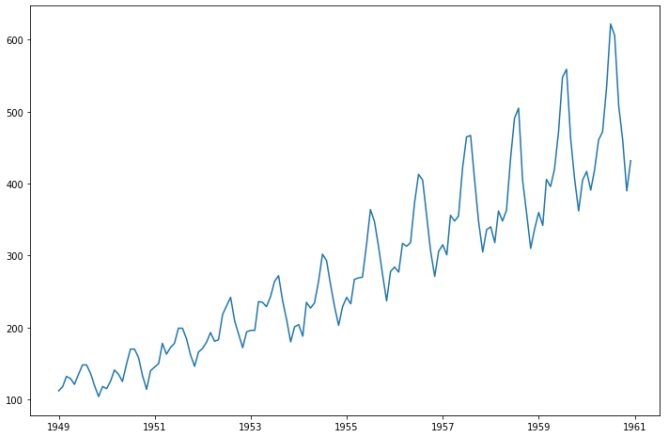
```
112
```

In [7]:

```
plt.figure(figsize=(12,8))
plt.plot(data['#Passengers'])
```

Out[7]:

```
[<matplotlib.lines.Line2D at 0x7f489551c
3c8>]
```



In [8]:

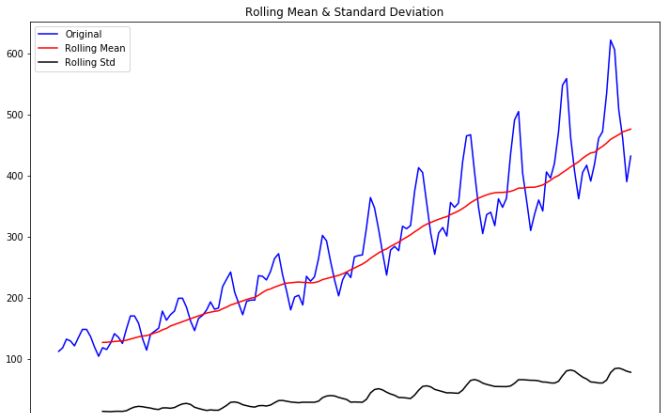
```
from statsmodels.tsa.stattools import adfuller
def test_stationarity(timeseries):
    plt.figure(figsize=(12,8))
    #Determing rolling statistics
```

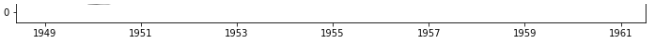
```
    rolmean = timeseries.rolling(window=12).mean()
    rolstd = timeseries.rolling(window=12).std()

    #Plot rolling statistics:
    orig = plt.plot(timeseries, color='blue',label='O
riginal')
    mean = plt.plot(rolmean, color='red', label='Roll
ing Mean')
    std = plt.plot(rolstd, color='black', label = 'Ro
lling 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 St
atistic','p-value','#Lags Used','Number of Observatio
ns Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)
```

```
/opt/conda/lib/python3.6/site-packages/s
tatsmodels/compat/pandas.py:56: FutureWa
rning: The pandas.core.datetools module
is deprecated and will be removed in a f
uture version. Please use the pandas.tse
ries module instead.
  from pandas.core import datetools
```

In [9]:

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





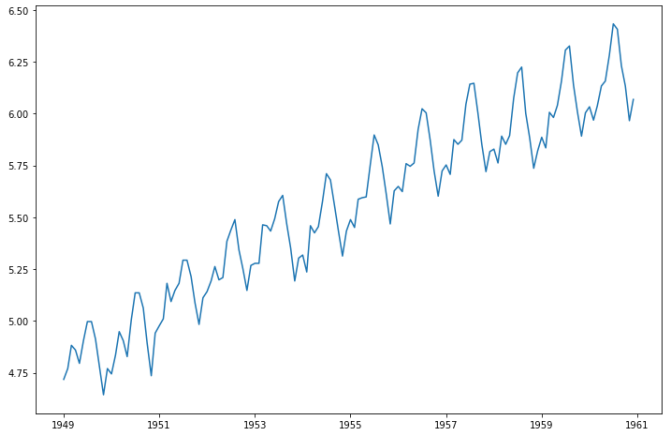

```
Results of Dickey-Fuller Test:
Test Statistic                   0.81536
9
p-value                          0.99188
0
#Lags Used                      13.00000
0
Number of Observations Used    130.00000
0
Critical Value (1%)             -3.48168
2
Critical Value (5%)             -2.88404
2
Critical Value (10%)            -2.57877
0
dtype: float64
```

In [10]:

```
ts_log = np.log(data['#Passengers'])
plt.figure(figsize=(12,8))
plt.plot(ts_log)
```

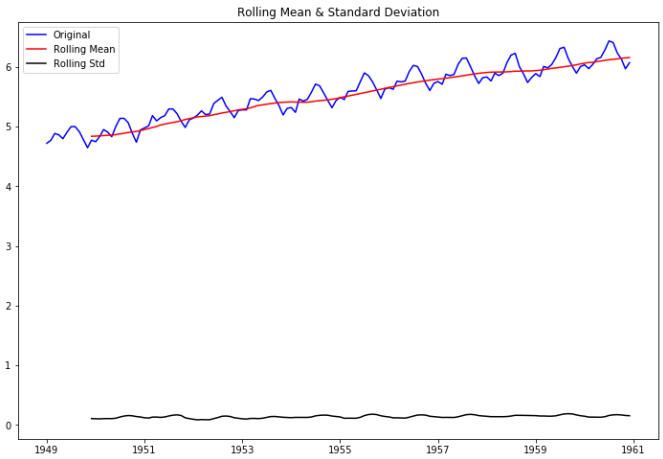
Out[10]:

```
[<matplotlib.lines.Line2D at 0x7f483bf37
550>]
```



In [11]:

```
test_stationarity(np.log(data['#Passengers']))
```



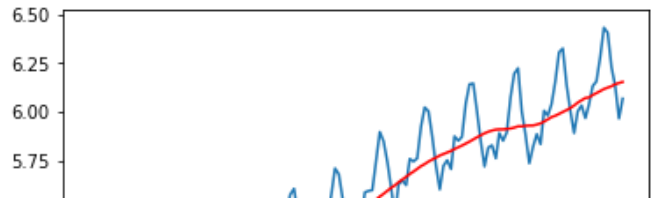

```
Results of Dickey-Fuller Test:
Test Statistic                  -1.71701
7
p-value                          0.42236
7
#Lags Used                      13.00000
0
Number of Observations Used    130.00000
0
Critical Value (1%)             -3.48168
2
Critical Value (5%)             -2.88404
2
Critical Value (10%)            -2.57877
0
dtype: float64
```

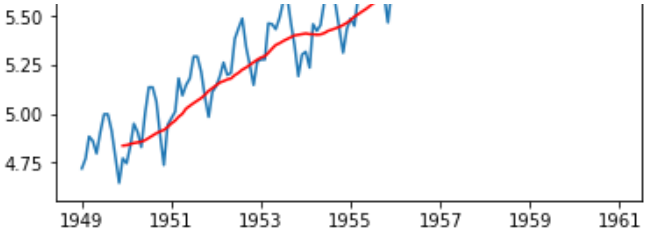
In [12]:

```
moving_avg = np.log(data['#Passengers']).rolling(12).
mean()
plt.plot(np.log(data['#Passengers']))
plt.plot(moving_avg, color='red')
```

Out[12]:

```
[<matplotlib.lines.Line2D at 0x7f4838229
7f0>]
```





In [13]:

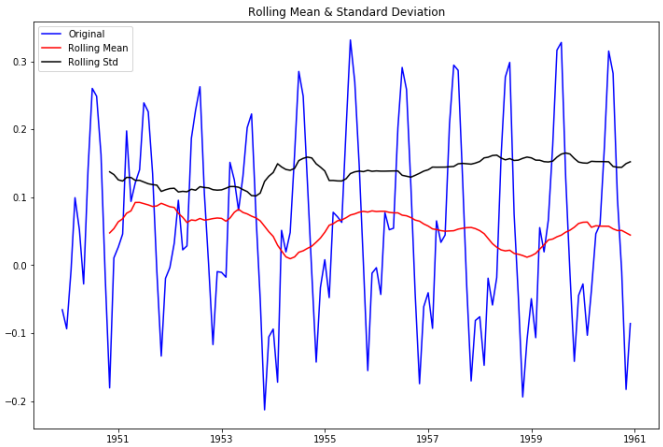
```
ts_log_moving_avg_diff = np.log(data['#Passengers'])
- moving_avg
ts_log_moving_avg_diff.head(12)
```

Out[13]:

```
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 [14]:

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



```
Results of Dickey-Fuller Test:
Test Statistic                  -3.16290
8
p-value                          0.02223
5
#Lags Used                      13.00000
0
Number of Observations Used    119.00000
0
Critical Value (1%)             -3.48653
5
Critical Value (5%)             -2.88615
1
Critical Value (10%)            -2.57989
6
dtype: float64
```

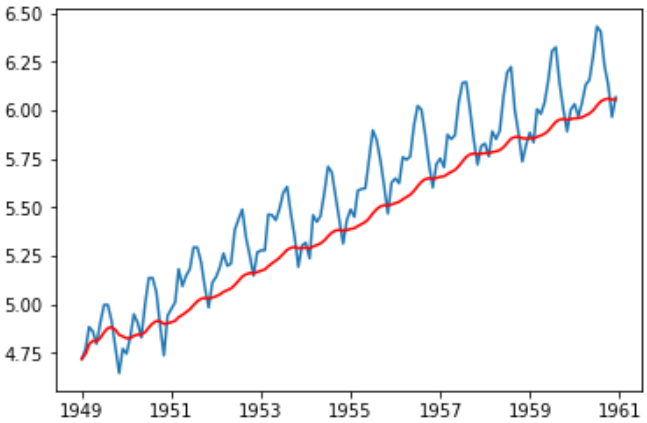
Using **Exponential Weighted Average**. Recent values given more weight.

In [15]:

```
expwighted_avg = np.log(data['#Passengers']).ewm(half
life=12).mean()
plt.plot(np.log(data['#Passengers']))
plt.plot(expwighted_avg, color='red')
```

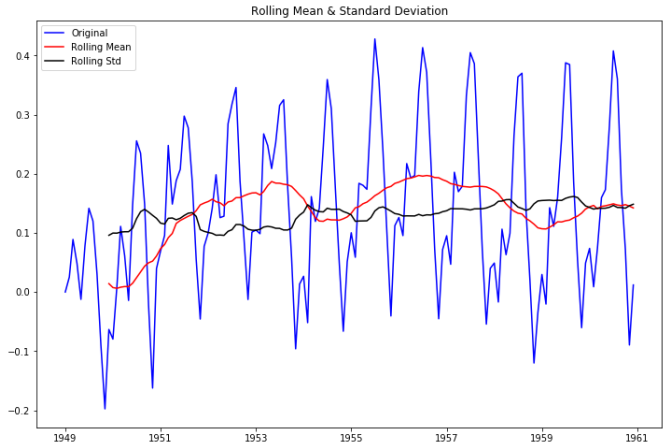
Out[15]:

```
[<matplotlib.lines.Line2D at 0x7f4838189
da0>]
```



In [16]:

```
ts_log_ewma_diff = np.log(data['#Passengers']) - expw
ighted_avg
test_stationarity(ts_log_ewma_diff)
```



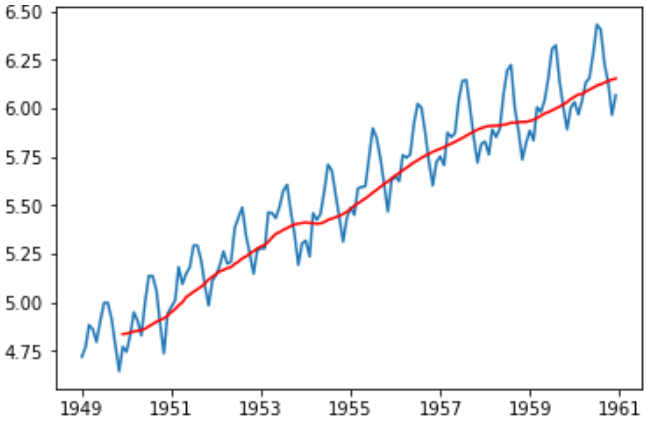

```
Results of Dickey-Fuller Test:
Test Statistic                  -3.60126
2
p-value                          0.00573
7
#Lags Used                      13.00000
0
Number of Observations Used    130.00000
0
Critical Value (1%)             -3.48168
2
Critical Value (5%)             -2.88404
2
Critical Value (10%)            -2.57877
0
dtype: float64
```

In [17]:

```
moving_avg = ts_log.rolling(12).mean()
plt.plot(ts_log)
plt.plot(moving_avg, color='red')
```

Out[17]:

```
[<matplotlib.lines.Line2D at 0x7f48380e9
0f0>]
```



In [18]:

```
ts_log_moving_avg_diff = ts_log - moving_avg
ts_log_moving_avg_diff.head(12)
```

Out[18]:

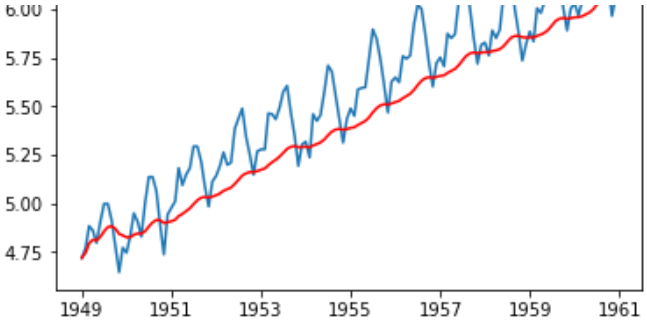
```
Month
1949-01-01         NaN
1949-02-01         NaN
1949-03-01         NaN
1949-04-01         NaN
1949-05-01         NaN
1949-06-01         NaN
1949-07-01         NaN
1949-08-01         NaN
1949-09-01         NaN
1949-10-01         NaN
1949-11-01         NaN
1949-12-01   -0.065494
Name: #Passengers,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

In [19]:

```
expwighted_avg = ts_log.ewm(halflife=12).mean()
plt.plot(ts_log)
plt.plot(expwighted_avg, color='red')
```

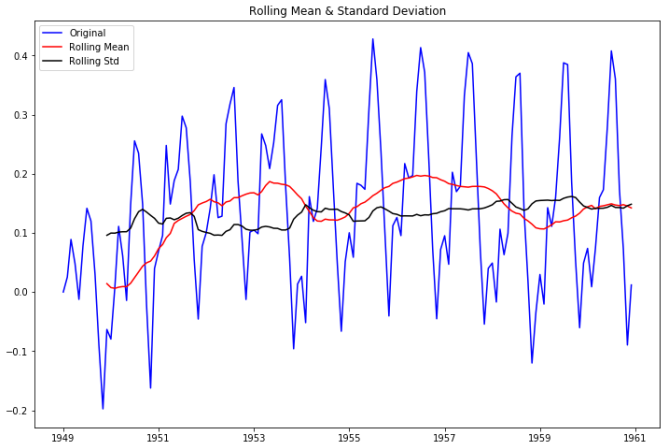
Out[19]:

```
[<matplotlib.lines.Line2D at 0x7f4838229
da0>]
```



In [20]:

```
ts_log_ewma_diff = ts_log - expwighted_avg
test_stationarity(ts_log_ewma_diff)
```



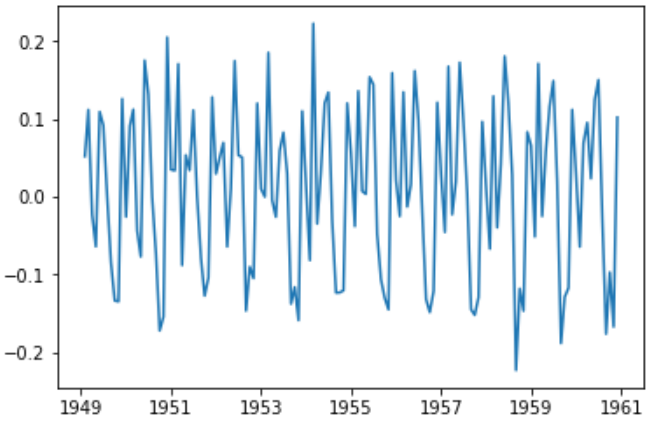

```
Results of Dickey-Fuller Test:
Test Statistic                  -3.601262
p-value                          0.005737
#Lags Used                      13.000000
Number of Observations Used    130.000000
Critical Value (1%)             -3.481682
Critical Value (5%)             -2.884042
Critical Value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

In [21]:

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

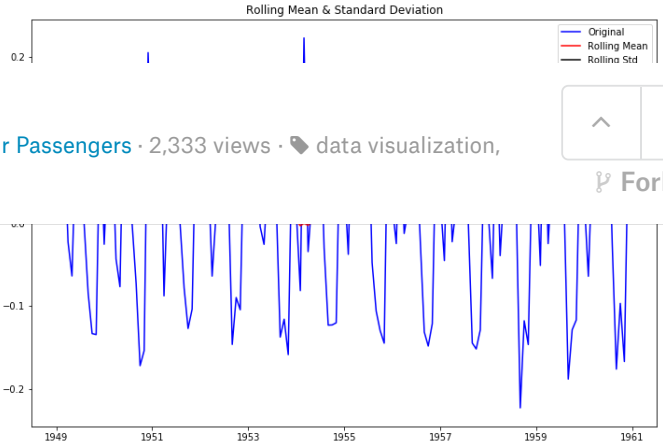
Out[21]:

```
[<matplotlib.lines.Line2D at 0x7f483bf7b
b00>]
```



In [22]:

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




```
Results of Dickey-Fuller Test:
Test Statistic                  -2.71713
1
p-value                          0.07112
1
#Lags Used                      14.00000
0
Number of Observations Used    128.00000
0
```

```
Critical Value (1%)              -3.48250
1
Critical Value (5%)              -2.88439
8
Critical Value (10%)             -2.57896
0
dtype: float64
```

In [23]:

```
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
# fit model
model = ARIMA(ts_log, order=(5,1,0))
model_fit = model.fit(disp=0)
print(model_fit.summary())
```

```
                             ARIMA Model
Results
=======================================
=======================================
Dep. Variable:          D.#Passengers
No. Observations:                   143
Model:                 ARIMA(5, 1, 0)
```

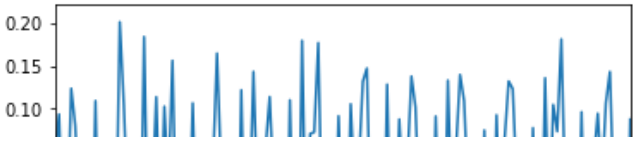


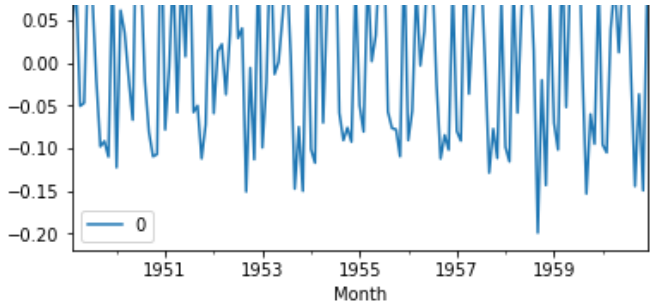
```
S.D. of innovations              0.096
Date:                Wed, 29 Nov 2017
AIC                           -248.746
Time:                         07:34:46
BIC                           -228.006
Sample:                     02-01-1949
HQIC                          -240.318
                          - 12-01-1960
=======================================
=======================================
=======
                            coef    std er
r          z      P>|z|      [0.025
0.975]
---------------------------------------
---------------------------------------
-------
const                     0.0101      0.00
6      1.682      0.095      -0.002
0.022
ar.L1.D.#Passengers       0.1951      0.08
```

```
4      2.329      0.021       0.031
0.359
ar.L2.D.#Passengers    -0.2019      0.08
2     -2.464      0.015      -0.363
-0.041
ar.L3.D.#Passengers    -0.0247      0.08
3     -0.297      0.767      -0.188
0.138
ar.L4.D.#Passengers    -0.3306      0.08
2     -4.042      0.000      -0.491
-0.170
ar.L5.D.#Passengers     0.0088      0.08
5      0.103      0.918      -0.158
0.176
                                    Root
s
=====================================
=====================================
                 Real           Imaginar
y           Modulus         Frequency
-------------------------------------
-------------------------------------
AR.1            0.8292           -0.9299
j            1.2460           -0.1341
AR.2            0.8292           +0.9299
j            1.2460            0.1341
AR.3           -0.8744           -1.0858
j            1.3941           -0.3579
AR.4           -0.8744           +1.0858
j            1.3941            0.3579
AR.5           37.6703           -0.0000
j           37.6703           -0.0000
-------------------------------------
-------------------------------------
```

In [24]:

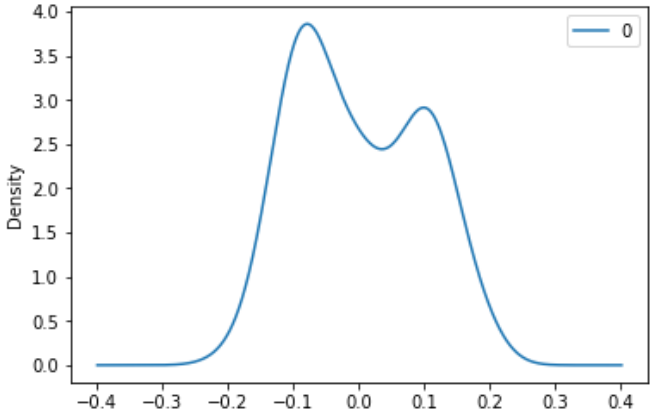
```
# plot residual errors
residuals = pd.DataFrame(model_fit.resid)
residuals.plot()
plt.show()
```





In [25]:

```
residuals.plot(kind='kde')
plt.show()
print(residuals.describe())
```




```
                0
count  143.000000
mean     0.000190
std      0.096771
min     -0.199075
25%     -0.080785
50%     -0.012261
75%      0.091857
max      0.201409
```

In [26]:

```
validation_size =int(len(ts_log)*0.66)
X = data['#Passengers'].astype('float')
train, test = X[0:validation_size], X[validation_size
:len(X)]
history = [x for x in train]
predictions = list()
```

In [27]:

```
for t in range(len(test)):
        model = ARIMA(history, order=(5,1,0))
        model_fit = model.fit(disp=0)
        output = model_fit.forecast()
        yhat = output[0]
        predictions.append(yhat)
        obs,y = test[t],test.index[t]
        history.append(obs)
        print('predicted=%f, expected=%f, month=%s' %
(yhat, obs, y))
```

```
predicted=279.908324, expected=306.00000
0, month=1956-12-01 00:00:00
predicted=346.953525, expected=315.00000
0, month=1957-01-01 00:00:00
predicted=325.749786, expected=301.00000
0, month=1957-02-01 00:00:00
predicted=305.826125, expected=356.00000
0, month=1957-03-01 00:00:00
predicted=365.160651, expected=348.00000
0, month=1957-04-01 00:00:00
predicted=330.488083, expected=355.00000
0, month=1957-05-01 00:00:00
predicted=368.172534, expected=422.00000
0, month=1957-06-01 00:00:00
predicted=417.317444, expected=465.00000
0, month=1957-07-01 00:00:00
predicted=468.888429, expected=467.00000
0, month=1957-08-01 00:00:00
predicted=462.688104, expected=404.00000
0, month=1957-09-01 00:00:00
predicted=368.300040, expected=347.00000
0, month=1957-10-01 00:00:00
predicted=340.324474, expected=305.00000
0, month=1957-11-01 00:00:00
predicted=308.878324, expected=336.00000
```


Data

Data Sources

Air Passeng...

144 x 2

Air Passengers

Number of air passengers per month

Last Updated: 2 years ago (Version 1)

About this Dataset

Context

Air Passengers per month. Workshop dataset

Run Info

Succeeded	True	Run Time	134.4 seconds
Exit Code	0	Queue Time	0 seconds
Docker Image Name	kaggle/python(Dockerfile)		
Timeout Exceeded	False	Output Size	0
		Used All Space	False
Failure Message			

Log

Download Log

```
Time  Line #  Log Message
           1  [{
           2    "data": "[NbConvertApp] Converting
              notebook __temp_notebook_source__.ipynb to
              html\n",
           3    "stream_name": "stderr",
           4    "time": 2.063298226974439
           5  },{
           6    "data": "[NbConvertApp] Support files
              will be in __results___files/\n",
           7    "stream_name": "stderr",
           8    "time": 2.2815360369859263
           9  },{
          10    "data": "[NbConvertApp] Making directory
              __results___files\n[NbConvertApp] Making
              directory __results___files\n[NbConvertApp]
              Making directory
              __results___files\n[NbConvertApp] Making
              directory __results___files\n[NbConvertApp]
```

```
Making directory
__results___files\n[NbConvertApp] Making
directory __results___files\n[NbConvertApp]
Making directory
__results___files\n[NbConvertApp] Making
directory __results___files\n[NbConvertApp]
Making directory
__results___files\n[NbConvertApp] Making
directory __results___files\n[NbConvertApp]
Making directory
__results___files\n[NbConvertApp] Making
directory __results___files\n[NbConvertApp]
Making directory
__results___files\n[NbConvertApp] Making
directory __results___files\n[NbConvertApp]
Making directory
__results___files\n[NbConvertApp] Making
directory __results___files\n[NbConvertApp]
Writing 303536 bytes to
__results__.html\n",
  "stream_name": "stderr",
  "time": 2.3168033150141127
}{
  "data": "[NbConvertApp] Converting
notebook __temp_notebook_source__.ipynb to
notebook\n",
  "stream_name": "stderr",
  "time": 1.9464787910110317
},{
  "data": "[NbConvertApp] Executing
notebook with kernel: python3\n",
  "stream_name": "stderr",
  "time": 1.9827664089971222
},{
  "data": "Fontconfig warning: ignoring
C.UTF-8: not a valid language tag\n",
  "stream_name": "stderr",
  "time": 2.975227717019152
},{
  "data": "[NbConvertApp] Writing 824531
bytes to __notebook__.ipynb\n",
  "stream_name": "stderr",
  "time": 84.99889715999598
}{
  "data": "[NbConvertApp] Converting
notebook __notebook__.ipynb to html\n",
  "stream_name": "stderr",
  "time": 2.1387842720141634
},{
  "data": "[NbConvertApp] Support files
will be in
__results___files/\n[NbConvertApp] Making
directory __results___files\n",
  "stream_name": "stderr",
  "time": 2.369405484001618
},{
  "data": "[NbConvertApp] Making directory
__results___files\n[NbConvertApp] Making
directory __results___files\n[NbConvertApp]
Making directory
__results___files\n[NbConvertApp] Making
directory __results___files\n[NbConvertApp]
Making directory
__results___files\n[NbConvertApp] Making
directory __results___files\n[NbConvertApp]
Making directory

__results___files\n[NbConvertApp] Making
```

```
directory __results___files\n[NbConvertApp]
Making directory
__results___files\n[NbConvertApp] Making
directory __results___files\n[NbConvertApp]
Making directory
__results___files\n[NbConvertApp] Making
directory __results___files\n[NbConvertApp]
Making directory
__results___files\n[NbConvertApp] Making
directory __results___files\n[NbConvertApp]
Making directory
__results___files\n[NbConvertApp] Writing
303524 bytes to __results__.html\n",
  "stream_name": "stderr",
  "time": 2.405713091022335
}

Complete. Exited with code 0.
```


Comments (2)

Sort by: All Comments | Hotness

Please sign in to leave a comment.

Clyton D... • Posted on Latest Version • a year ago 0

How do you decide the (p,q,d) for the model?

Kaiva... • Posted on Latest Version • 7 months ago 1

Check out the "ACF" and "PACF" i.e. the auto correlation and partial auto correlation for deciding the p and q values which can be determined by the checking the intersection of the line with 0.

Similar Kernels