Assignment -18 (Forecasting) Cocacola-Sales-Rawdata

1)Forecast the CocaCola prices data set. Prepare a document for each model explaining. How many dummy variables you have created and RMSE value for each model. Finally which model you will use for Forecasting.

In [1]:

```
# import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.formula.api as smf
```

In [2]:
```
df = pd.read_excel('CocaCola_Sales_Rawdata.xlsx')
df
```

Out[2]:

	Quarter	Sales
0	Q1_86	1734.827000
1	Q2_86	2244.960999
2	Q3_86	2533.804993
3	Q4_86	2154.962997
4	Q1_87	1547.818996
5	Q2_87	2104.411995
6	Q3_87	2014.362999
7	Q4_87	1991.746998
8	Q1_88	1869.049999
9	Q2_88	2313.631996
10	Q3_88	2128.320000
11	Q4_88	2026.828999
12	Q1_89	1910.603996
13	Q2_89	2331.164993
14	Q3_89	2206.549995
15	Q4_89	2173.967995
16	Q1_90	2148.278000
17	Q2_90	2739.307999
18	Q3_90	2792.753998
19	Q4_90	2556.009995
20	Q1_91	2480.973999
21	Q2_91	3039.522995
22	Q3_91	3172.115997
23	Q4_91	2879.000999
24	Q1_92	2772.000000
25	Q2_92	3550.000000
26	Q3_92	3508.000000
27	Q4_92	3243.859993
28	Q1_93	3056.000000
29	Q2_93	3899.000000
30	Q3_93	3629.000000
31	Q4_93	3373.000000
32	Q1_94	3352.000000
33	Q2_94	4342.000000
34	Q3_94	4461.000000
35	Q4_94	4017.000000
36	Q1_95	3854.000000
37	Q2_95	4936.000000
38	Q3_95	4895.000000
39	Q4_95	4333.000000
40	Q1_96	4194.000000
41	Q2_96	5253.000000

In [3]:
```
1 df.describe()
```

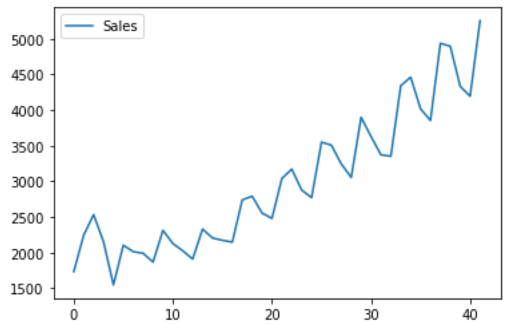
Out[3]:

	Sales
count	42.000000
mean	2994.353308
std	977.930896
min	1547.818996
25%	2159.714247
50%	2782.376999
75%	3609.250000
max	5253.000000

Visualization

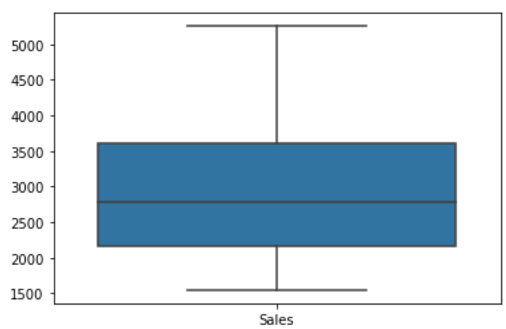
In [4]:
```
1 df.plot()
```

Out[4]: <AxesSubplot:>



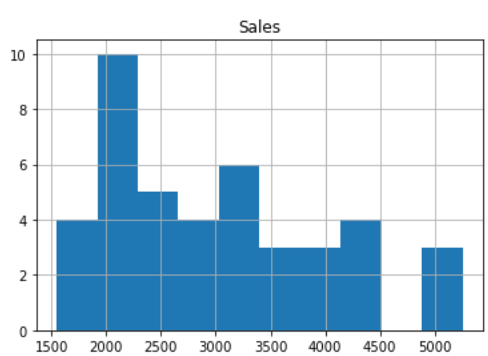
In [5]:
```
1 import seaborn as sns
2 sns.boxplot(data=df)
```

Out[5]: <AxesSubplot:>



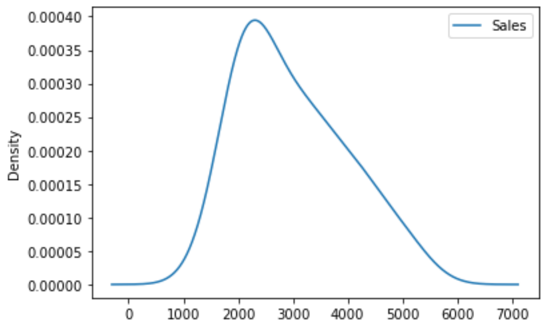
In [6]:
```
1 df.hist()
```

Out[6]: array([[<AxesSubplot:title={'center':'Sales'}>]], dtype=object)



In [7]:
```
df.plot(kind="kde")
```

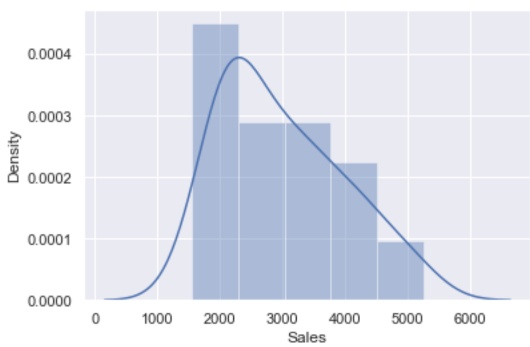
Out[7]: <AxesSubplot:ylabel='Density'>



In [8]:
```
np.array(df["Sales"])
```

Out[8]:
```
array([1734.82699966, 2244.96099854, 2533.80499268, 2154.96299744,
       1547.81899643, 2104.41199493, 2014.36299896, 1991.74699783,
       1869.04999924, 2313.63199615, 2128.31999969, 2026.82899857,
       1910.60399628, 2331.16499329, 2206.54999542, 2173.96799469,
       2148.27799988, 2739.30799866, 2792.7539978 , 2556.00999451,
       2480.97399902, 3039.522995  , 3172.11599731, 2879.00099945,
       2772.        , 3550.        , 3508.        , 3243.85999298,
       3056.        , 3899.        , 3629.        , 3373.        ,
       3352.        , 4342.        , 4461.        , 4017.        ,
       3854.        , 4936.        , 4895.        , 4333.        ,
       4194.        , 5253.        ])
```

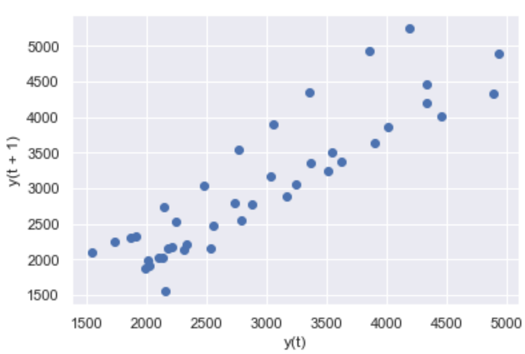
In [9]:
```
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
sns.set_theme()
rf = sns.distplot(df['Sales'],kde=True)
```



In [10]:
```
from pandas.plotting import lag_plot
import warnings
warnings.filterwarnings('ignore')
lag_plot(df['Sales'])
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a single row if you
intend to specify the same RGB or RGBA value for all points.
```

Out[10]: <AxesSubplot:xlabel='y(t)', ylabel='y(t + 1)'>



Data Preprocessing

In [11]:
```
df.head()
```

Out[11]:

	Quarter	Sales
0	Q1_86	1734.827000
1	Q2_86	2244.960999
2	Q3_86	2533.804993
3	Q4_86	2154.962997
4	Q1_87	1547.818996

In [12]:
```
len(df)
```

Out[12]: 42

In [13]:
```
df['quarter'] = 0
for i in range(42):
    p=df['Quarter'][i]
    df['quarter'][i]=p[0:2]
```

In [14]:
```
df
```

Out[14]:

	Quarter	Sales	quarter
0	Q1_86	1734.827000	Q1
1	Q2_86	2244.960999	Q2
2	Q3_86	2533.804993	Q3
3	Q4_86	2154.962997	Q4
4	Q1_87	1547.818996	Q1
5	Q2_87	2104.411995	Q2
6	Q3_87	2014.362999	Q3
7	Q4_87	1991.746998	Q4
8	Q1_88	1869.049999	Q1
9	Q2_88	2313.631996	Q2
10	Q3_88	2128.320000	Q3
11	Q4_88	2026.828999	Q4
12	Q1_89	1910.603996	Q1
13	Q2_89	2331.164993	Q2
14	Q3_89	2206.549995	Q3
15	Q4_89	2173.967995	Q4
16	Q1_90	2148.278000	Q1
17	Q2_90	2739.307999	Q2
18	Q3_90	2792.753998	Q3
19	Q4_90	2556.009995	Q4
20	Q1_91	2480.973999	Q1
21	Q2_91	3039.522995	Q2
22	Q3_91	3172.115997	Q3
23	Q4_91	2879.000999	Q4
24	Q1_92	2772.000000	Q1
25	Q2_92	3550.000000	Q2
26	Q3_92	3508.000000	Q3
27	Q4_92	3243.859993	Q4
28	Q1_93	3056.000000	Q1
29	Q2_93	3899.000000	Q2
30	Q3_93	3629.000000	Q3
31	Q4_93	3373.000000	Q4
32	Q1_94	3352.000000	Q1
33	Q2_94	4342.000000	Q2
34	Q3_94	4461.000000	Q3
35	Q4_94	4017.000000	Q4
36	Q1_95	3854.000000	Q1
37	Q2_95	4936.000000	Q2
38	Q3_95	4895.000000	Q3
39	Q4_95	4333.000000	Q4
40	Q1_96	4194.000000	Q1
41	Q2_96	5253.000000	Q2

In [15]:
```
df['quarter'].value_counts()
```

Out[15]:
```
Q1    11
Q2    11
Q3    10
Q4    10
Name: quarter, dtype: int64
```

In [16]:
```
df_dummies=pd.DataFrame(pd.get_dummies(df['quarter']),columns=['Q1','Q2','Q3','Q4'])
cc=pd.concat([df,df_dummies],axis= 1)
```

In [17]:
```
df.head()
```

Out[17]:

	Quarter	Sales	quarter
0	Q1_86	1734.827000	Q1
1	Q2_86	2244.960999	Q2
2	Q3_86	2533.804993	Q3
3	Q4_86	2154.962997	Q4
4	Q1_87	1547.818996	Q1

In [18]:
```
cc
```

Out[18]:

	Quarter	Sales	quarter	Q1	Q2	Q3	Q4
0	Q1_86	1734.827000	Q1	1	0	0	0
1	Q2_86	2244.960999	Q2	0	1	0	0
2	Q3_86	2533.804993	Q3	0	0	1	0
3	Q4_86	2154.962997	Q4	0	0	0	1
4	Q1_87	1547.818996	Q1	1	0	0	0
5	Q2_87	2104.411995	Q2	0	1	0	0
6	Q3_87	2014.362999	Q3	0	0	1	0
7	Q4_87	1991.746998	Q4	0	0	0	1
8	Q1_88	1869.049999	Q1	1	0	0	0
9	Q2_88	2313.631996	Q2	0	1	0	0
10	Q3_88	2128.320000	Q3	0	0	1	0
11	Q4_88	2026.828999	Q4	0	0	0	1
12	Q1_89	1910.603996	Q1	1	0	0	0
13	Q2_89	2331.164993	Q2	0	1	0	0
14	Q3_89	2206.549995	Q3	0	0	1	0
15	Q4_89	2173.967995	Q4	0	0	0	1
16	Q1_90	2148.278000	Q1	1	0	0	0
17	Q2_90	2739.307999	Q2	0	1	0	0
18	Q3_90	2792.753998	Q3	0	0	1	0
19	Q4_90	2556.009995	Q4	0	0	0	1
20	Q1_91	2480.973999	Q1	1	0	0	0
21	Q2_91	3039.522995	Q2	0	1	0	0
22	Q3_91	3172.115997	Q3	0	0	1	0
23	Q4_91	2879.000999	Q4	0	0	0	1
24	Q1_92	2772.000000	Q1	1	0	0	0
25	Q2_92	3550.000000	Q2	0	1	0	0
26	Q3_92	3508.000000	Q3	0	0	1	0
27	Q4_92	3243.859993	Q4	0	0	0	1
28	Q1_93	3056.000000	Q1	1	0	0	0
29	Q2_93	3899.000000	Q2	0	1	0	0
30	Q3_93	3629.000000	Q3	0	0	1	0
31	Q4_93	3373.000000	Q4	0	0	0	1
32	Q1_94	3352.000000	Q1	1	0	0	0
33	Q2_94	4342.000000	Q2	0	1	0	0
34	Q3_94	4461.000000	Q3	0	0	1	0
35	Q4_94	4017.000000	Q4	0	0	0	1
36	Q1_95	3854.000000	Q1	1	0	0	0
37	Q2_95	4936.000000	Q2	0	1	0	0
38	Q3_95	4895.000000	Q3	0	0	1	0
39	Q4_95	4333.000000	Q4	0	0	0	1
40	Q1_96	4194.000000	Q1	1	0	0	0
41	Q2_96	5253.000000	Q2	0	1	0	0

In [19]:
```
cc['t'] = np.arange(1,43)
cc['t_squared'] = cc['t']**2
cc["Sales_log"] = np.log(df['Sales'])
```

In [20]:
```
cc.head()
```

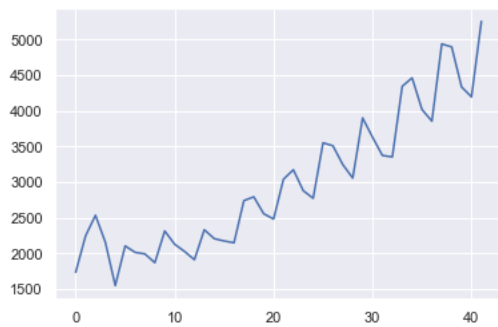
Out[20]:

	Quarter	Sales	quarter	Q1	Q2	Q3	Q4	t	t_squared	Sales_log
0	Q1_86	1734.827000	Q1	1	0	0	0	1	1	7.458663
1	Q2_86	2244.960999	Q2	0	1	0	0	2	4	7.716443
2	Q3_86	2533.804993	Q3	0	0	1	0	3	9	7.837477
3	Q4_86	2154.962997	Q4	0	0	0	1	4	16	7.675529
4	Q1_87	1547.818996	Q1	1	0	0	0	5	25	7.344602

In [21]:
```
train = cc.head(32)
test = cc.tail(10)
```

In [22]:
```
df['Sales'].plot()
```

Out[22]: <AxesSubplot:>



Model

In [23]:
```
from sklearn.metrics import mean_squared_error
```

In [24]:
```
# Linear Model
linear_model = smf.ols("Sales~t",data=train).fit()
linear_pred = pd.Series(linear_model.predict(test['t']))
linear_rmse = np.sqrt(mean_squared_error(np.array(test['Sales']),np.array(linear_pred)))
linear_rmse
```

Out[24]: 752.9233932767132

In [25]:
```
# Quadratic model
quad_model = smf.ols("Sales~t+t_squared",data=train).fit()
quad_pred = pd.Series(quad_model.predict(test[['t','t_squared']]))
quad_rmse = np.sqrt(mean_squared_error(np.array(test['Sales']),np.array(quad_pred)))
quad_rmse
```

Out[25]: 457.7357355407399

In [26]:
```
# Exponential model
exp_model = smf.ols("Sales_log~t",data=train).fit()
exp_pred = pd.Series(exp_model.predict(test['t']))
exp_rmse = np.sqrt(mean_squared_error(np.array(test['Sales']),np.array(exp_pred)))
exp_rmse
```

Out[26]: 4387.940544839098

In [27]:
```
data = {"MODEL":pd.Series(["rmse_linear","rmse_exp","rmse_quad"]),"RMSE_Values":pd.Series([linear_rmse,exp_rmse,quad_rmse,])
table_rmse=pd.DataFrame(data)
table_rmse.sort_values(['RMSE_Values'])
```

Out[27]:

	MODEL	RMSE_Values
2	rmse_quad	457.735736
0	rmse_linear	752.923393
1	rmse_exp	4387.940545

Using ARIMA model

In [28]:
```
data = pd.read_excel('CocaCola_Sales_Rawdata.xlsx',header=0,index_col=0, parse_dates=True)
data.head()
```

Out[28]:

	Sales
Quarter	
Q1_86	1734.827000
Q2_86	2244.960999
Q3_86	2533.804993
Q4_86	2154.962997
Q1_87	1547.818996

In [29]:
```
#separate out a validation dataset
split_point = len(data) - 7
dataset_cc, validation_cc = data[0:split_point], data[split_point:]
print('Dataset_cc %d, Validation_cc %d' % (len(dataset_cc), len(validation_cc)))
```

```
Dataset_cc 35, Validation_cc 7
```

In [30]:
```
dataset_cc.to_csv('dataset_cc.csv', header=False)
validation_cc.to_csv('validation_cc.csv', header=False)
```

In [31]:
```
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from math import sqrt

train = read_csv('dataset_cc.csv', header=None, index_col=0, parse_dates=True, squeeze=True)

train
```

Out[31]:
```
0
Q1_86    1734.827000
Q2_86    2244.960999
Q3_86    2533.804993
Q4_86    2154.962997
Q1_87    1547.818996
Q2_87    2104.411995
Q3_87    2014.362999
Q4_87    1991.746998
Q1_88    1869.049999
Q2_88    2313.631996
Q3_88    2128.320000
Q4_88    2026.828999
Q1_89    1910.603996
Q2_89    2331.164993
Q3_89    2206.549995
Q4_89    2173.967995
Q1_90    2148.278000
Q2_90    2739.307999
Q3_90    2792.753998
Q4_90    2556.009995
Q1_91    2480.973999
Q2_91    3039.522995
Q3_91    3172.115997
Q4_91    2879.000999
Q1_92    2772.000000
Q2_92    3550.000000
Q3_92    3508.000000
Q4_92    3243.859993
Q1_93    3056.000000
Q2_93    3899.000000
Q3_93    3629.000000
Q4_93    3373.000000
Q1_94    3352.000000
Q2_94    4342.000000
Q3_94    4461.000000
Name: 1, dtype: float64
```

In [32]:
```
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from math import sqrt

train = read_csv('dataset_cc.csv', header=None, index_col=0, parse_dates=True, squeeze=True)

train
```

Out[32]:
```
0
Q1_86    1734.827000
Q2_86    2244.960999
Q3_86    2533.804993
Q4_86    2154.962997
Q1_87    1547.818996
Q2_87    2104.411995
Q3_87    2014.362999
Q4_87    1991.746998
Q1_88    1869.049999
Q2_88    2313.631996
Q3_88    2128.320000
Q4_88    2026.828999
Q1_89    1910.603996
Q2_89    2331.164993
Q3_89    2206.549995
Q4_89    2173.967995
Q1_90    2148.278000
Q2_90    2739.307999
Q3_90    2792.753998
Q4_90    2556.009995
Q1_91    2480.973999
Q2_91    3039.522995
Q3_91    3172.115997
Q4_91    2879.000999
Q1_92    2772.000000
Q2_92    3550.000000
Q3_92    3508.000000
Q4_92    3243.859993
Q1_93    3056.000000
Q2_93    3899.000000
Q3_93    3629.000000
Q4_93    3373.000000
Q1_94    3352.000000
Q2_94    4342.000000
Q3_94    4461.000000
Name: 1, dtype: float64
```

In [33]:
```
X = train.values
X = X.astype('float32')
train_size = int(len(X) * 0.50)
train, test = X[0:train_size], X[train_size:]
```

Validation

In [34]:

```
history = [x for x in train]
predictions = list()
for i in range(len(test)):
    yhat = history[-1]
    predictions.append(yhat)
# observation
    obs = test[i]
    history.append(obs)
    print('>Predicted=%.3f, Expected=%.3f' % (yhat, obs))
# report performance
rmse = sqrt(mean_squared_error(test, predictions))
print('RMSE: %.3f' % rmse)
```

```
>Predicted=2148.278, Expected=2739.308
>Predicted=2739.308, Expected=2792.754
>Predicted=2792.754, Expected=2556.010
>Predicted=2556.010, Expected=2480.974
>Predicted=2480.974, Expected=3039.523
>Predicted=3039.523, Expected=3172.116
>Predicted=3172.116, Expected=2879.001
>Predicted=2879.001, Expected=2772.000
>Predicted=2772.000, Expected=3550.000
>Predicted=3550.000, Expected=3508.000
>Predicted=3508.000, Expected=3243.860
>Predicted=3243.860, Expected=3056.000
>Predicted=3056.000, Expected=3899.000
>Predicted=3899.000, Expected=3629.000
>Predicted=3629.000, Expected=3373.000
>Predicted=3373.000, Expected=3352.000
>Predicted=3352.000, Expected=4342.000
>Predicted=4342.000, Expected=4461.000
RMSE: 434.401
```

In [35]:

```
data = {"MODEL":pd.Series(["rmse_linear","rmse_exp","rmse_quad","RMSE_ARIMA"]),"RMSE_Values":pd.Series([linear_rmse,exp_rmse
table_rmse=pd.DataFrame(data)
table_rmse.sort_values(['RMSE_Values'])
```

Out[35]:

	MODEL	RMSE_Values
3	RMSE_ARIMA	434.400665
2	rmse_quad	457.735736
0	rmse_linear	752.923393
1	rmse_exp	4387.940545

The least RMSE values has the RMSE_ARMIA model and we can final this model

In [ ]:

In [ ]:

In [ ]:

2)Forecast the Airlines Passengers data set. Prepare a document for each model explaining.How many dummy variables you have created and RMSE value for each model.Finally which model you will use for Forecasting.

In [36]:

```
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
%matplotlib inline
from matplotlib.pylab import rcParams
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')

```

In [37]:
```
data = pd.read_excel("Airlines+Data.xlsx")
data
```

Out[37]:

	Month	Passengers
0	1995-01-01	112
1	1995-02-01	118
2	1995-03-01	132
3	1995-04-01	129
4	1995-05-01	121
...	...	...
91	2002-08-01	405
92	2002-09-01	355
93	2002-10-01	306
94	2002-11-01	271
95	2002-12-01	306

96 rows × 2 columns

In [38]:
```
data.shape
```

Out[38]: (96, 2)

In [39]:
```
data['Month']=pd.to_datetime(data['Month'], infer_datetime_format=True)
data=data.set_index(['Month'])
```

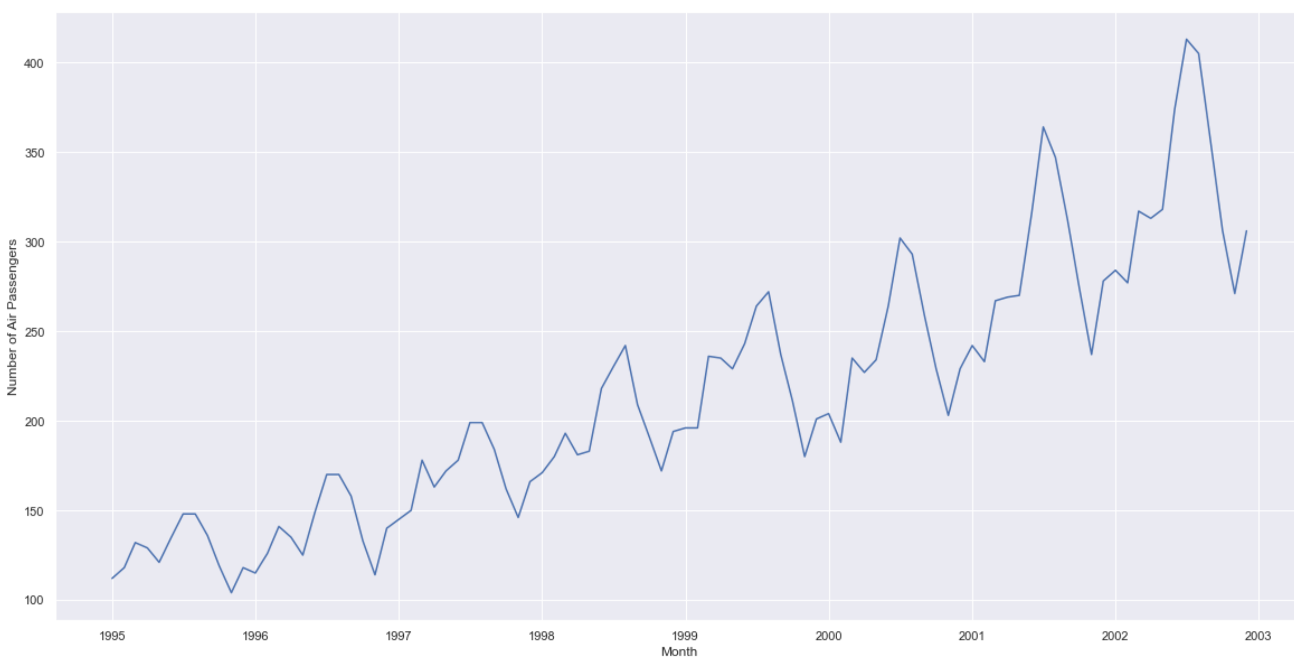
In [40]:
```
data.head()
```

Out[40]:

Month	Passengers
1995-01-01	112
1995-02-01	118
1995-03-01	132
1995-04-01	129
1995-05-01	121

In [41]:
```
plt.figure(figsize=(20,10))
plt.xlabel("Month")
plt.ylabel("Number of Air Passengers")
plt.plot(data)
```

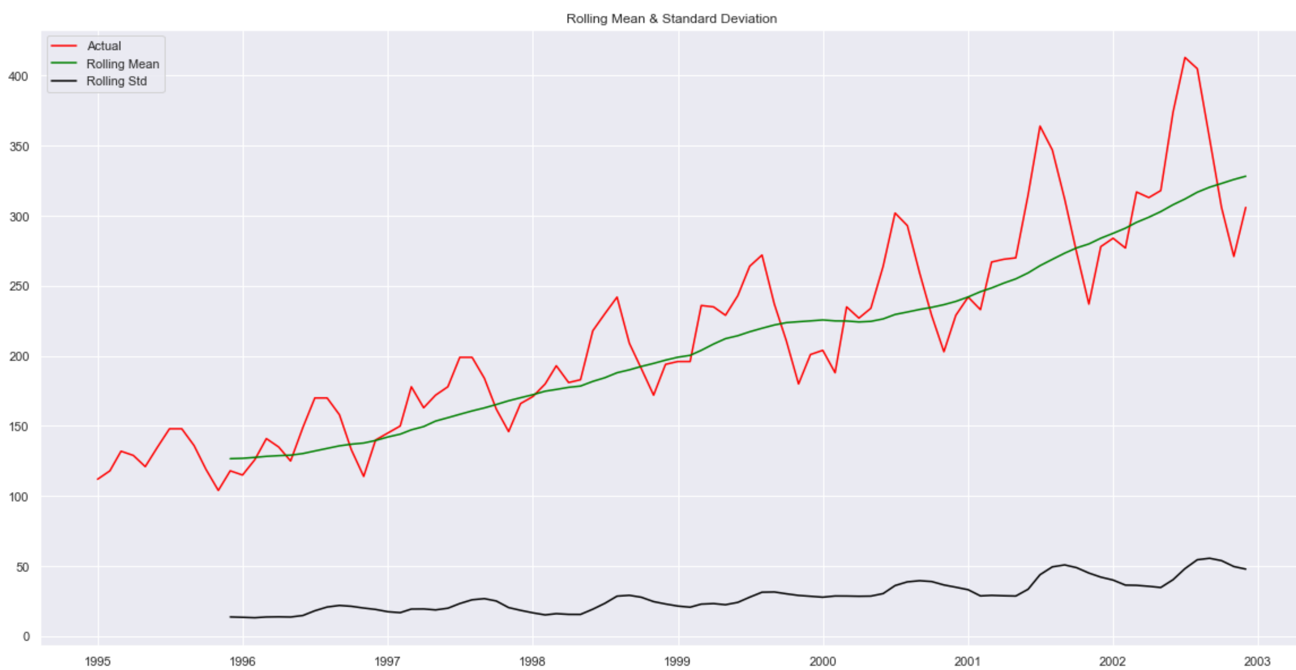
Out[41]: [<matplotlib.lines.Line2D at 0x1cf63deba60>]



In [42]:
```
rolmean=data.rolling(window=12).mean()
rolstd=data.rolling(window=12).std()
print(rolmean.head(15))
print(rolstd.head(15))
```

```
            Passengers
Month
1995-01-01         NaN
1995-02-01         NaN
1995-03-01         NaN
1995-04-01         NaN
1995-05-01         NaN
1995-06-01         NaN
1995-07-01         NaN
1995-08-01         NaN
1995-09-01         NaN
1995-10-01         NaN
1995-11-01         NaN
1995-12-01  126.666667
1996-01-01  126.916667
1996-02-01  127.583333
1996-03-01  128.333333
            Passengers
Month
1995-01-01         NaN
1995-02-01         NaN
1995-03-01         NaN
1995-04-01         NaN
1995-05-01         NaN
1995-06-01         NaN
1995-07-01         NaN
1995-08-01         NaN
1995-09-01         NaN
1995-10-01         NaN
1995-11-01         NaN
1995-12-01   13.720147
1996-01-01   13.453342
1996-02-01   13.166475
1996-03-01   13.686977
```

In [43]:
```
plt.figure(figsize=(20,10))
actual=plt.plot(data, color='red', label='Actual')
mean_6=plt.plot(rolmean, color='green', label='Rolling Mean')
std_6=plt.plot(rolstd, color='black', label='Rolling Std')
plt.legend(loc='best')
plt.title('Rolling Mean & Standard Deviation')
plt.show(block=False)
```

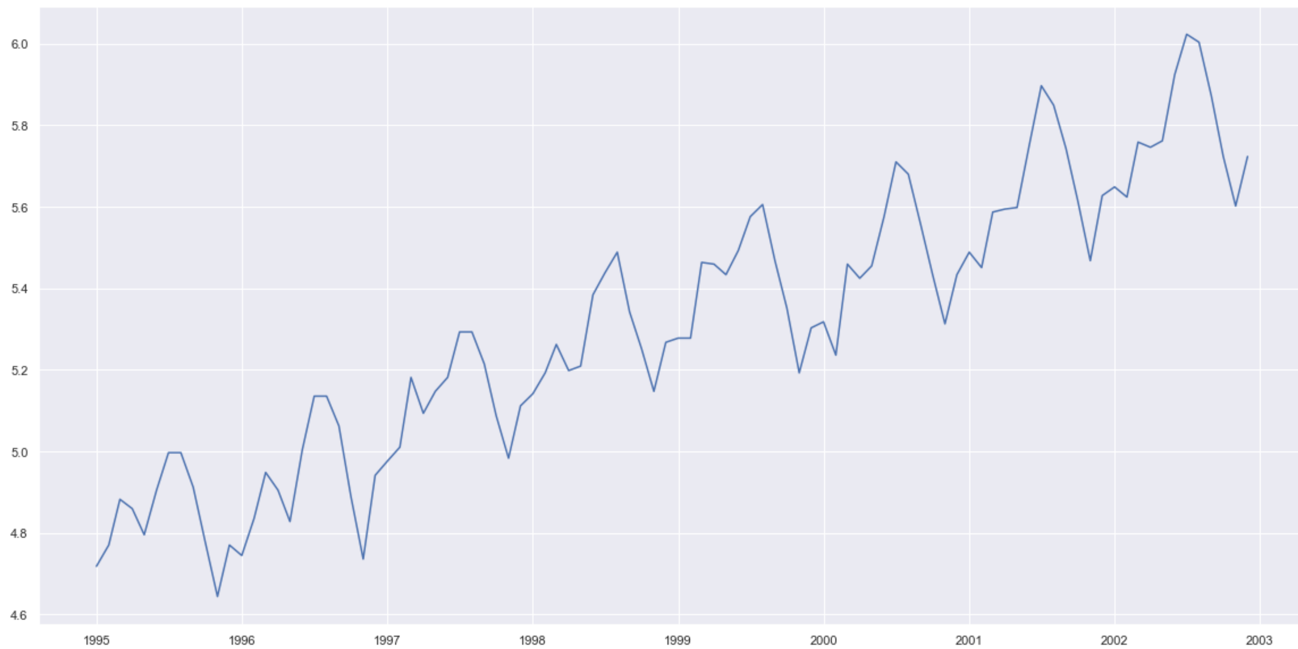


In [44]:
```
from statsmodels.tsa.stattools import adfuller
print('Dickey-Fuller Test: ')
dftest=adfuller(data['Passengers'], autolag='AIC')
dfoutput=pd.Series(dftest[0:4], index=['Test Statistic','p-value','Lags Used','No. of Obs'])
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value
print(dfoutput)
```

```
Dickey-Fuller Test: 
Test Statistic           1.340248
p-value                  0.996825
Lags Used               12.000000
No. of Obs              83.000000
Critical Value (1%)     -3.511712
Critical Value (5%)     -2.897048
Critical Value (10%)    -2.585713
dtype: float64
```

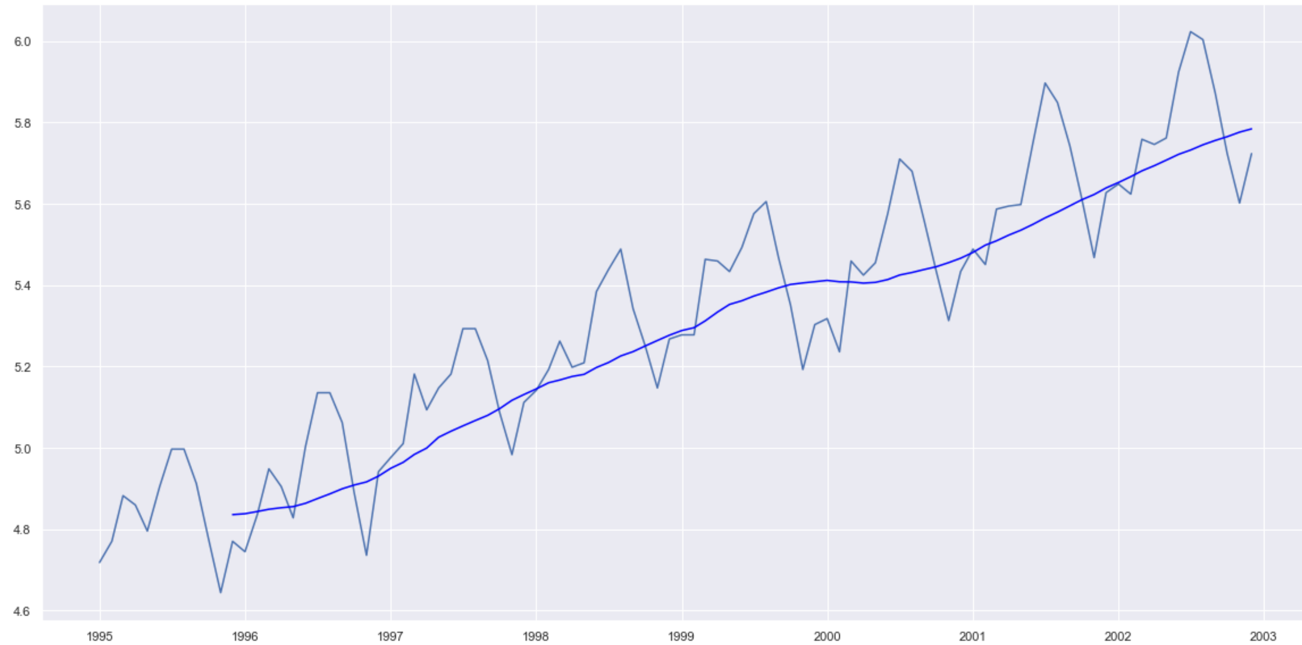
In [45]:
```
plt.figure(figsize=(20,10))
data_log=np.log(data)
plt.plot(data_log)
```

Out[45]: [<matplotlib.lines.Line2D at 0x1cf6463d070>]



In [46]:
```
plt.figure(figsize=(20,10))
MAvg=data_log.rolling(window=12).mean()
MStd=data_log.rolling(window=12).std()
plt.plot(data_log)
plt.plot(MAvg, color='blue')
```

Out[46]: [<matplotlib.lines.Line2D at 0x1cf646dbdc0>]



In [47]:
```
data_log_diff=data_log-MAvg
data_log_diff.head(12)
```

Out[47]:

	Passengers
Month	
1995-01-01	NaN
1995-02-01	NaN
1995-03-01	NaN
1995-04-01	NaN
1995-05-01	NaN
1995-06-01	NaN
1995-07-01	NaN
1995-08-01	NaN
1995-09-01	NaN
1995-10-01	NaN
1995-11-01	NaN
1995-12-01	-0.065494

In [48]:
```
data_log_diff=data_log_diff.dropna()
data_log_diff.head()
```

Out[48]:

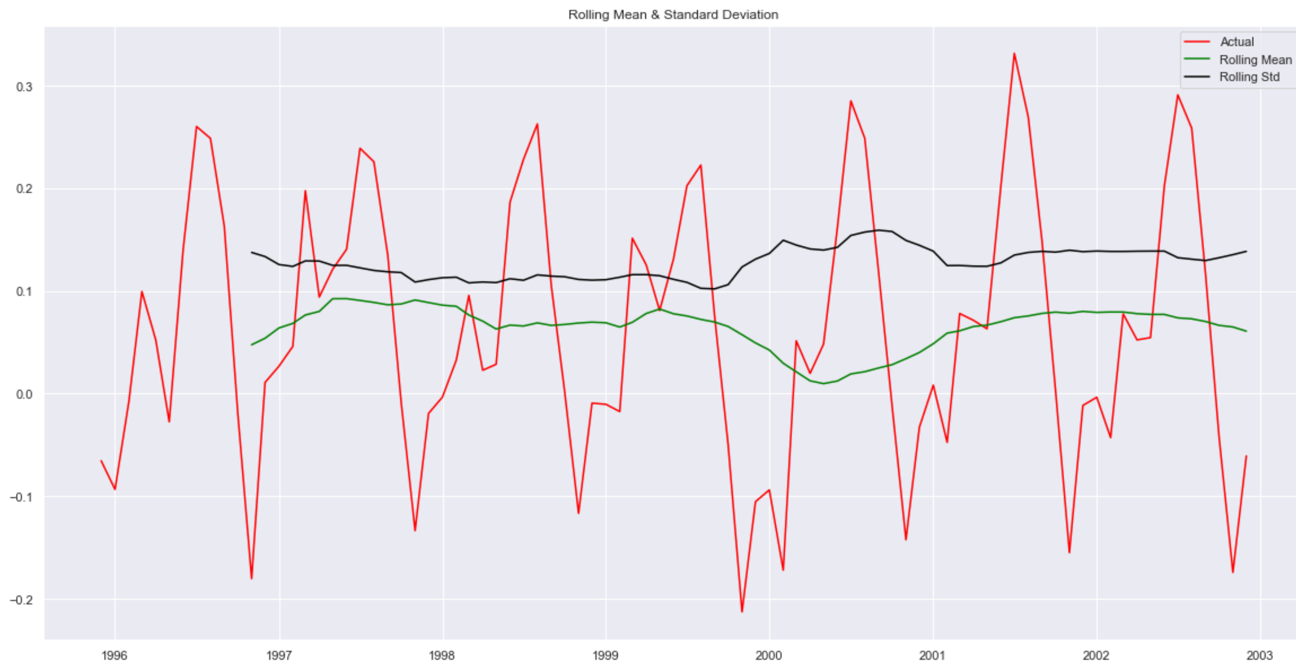
	Passengers
Month	
1995-12-01	-0.065494
1996-01-01	-0.093449
1996-02-01	-0.007566
1996-03-01	0.099416
1996-04-01	0.052142

In [49]:

```
def stationarity(timeseries):

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

    plt.figure(figsize=(20,10))
    actual=plt.plot(timeseries, color='red', label='Actual')
    mean_6=plt.plot(rolmean, color='green', label='Rolling Mean')
    std_6=plt.plot(rolstd, color='black', label='Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)

    print('Dickey-Fuller Test: ')
    dftest=adfuller(timeseries['Passengers'], autolag='AIC')
    dfoutput=pd.Series(dftest[0:4], index=['Test Statistic','p-value','Lags Used','No. of Obs'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)
```

In [50]:

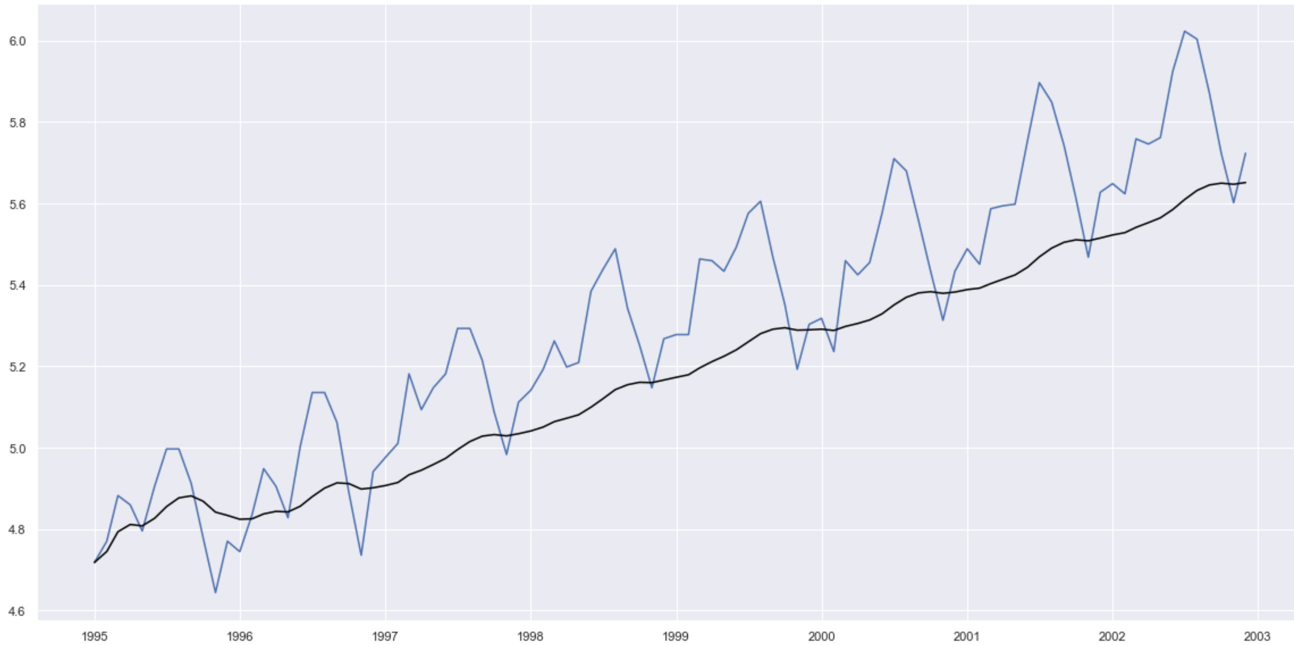
```
stationarity(data_log_diff)
```



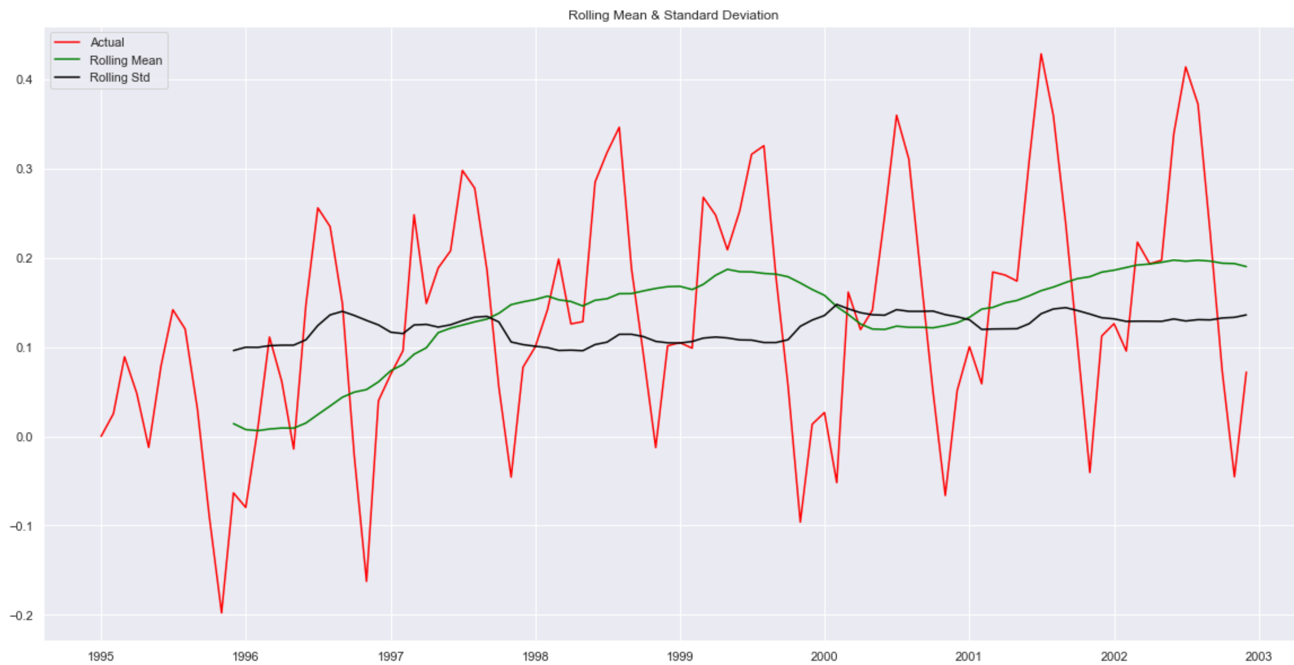

```
Dickey-Fuller Test: 
Test Statistic          -1.910930
p-value                  0.326937
Lags Used               12.000000
No. of Obs              72.000000
Critical Value (1%)     -3.524624
Critical Value (5%)     -2.902607
Critical Value (10%)    -2.588679
dtype: float64
```

In [51]:
```
plt.figure(figsize=(20,10))
exp_data=data_log.ewm(halflife=12, min_periods=0, adjust=True).mean()
plt.plot(data_log)
plt.plot(exp_data, color='black')
```

Out[51]: [<matplotlib.lines.Line2D at 0x1cf64a7c6d0>]



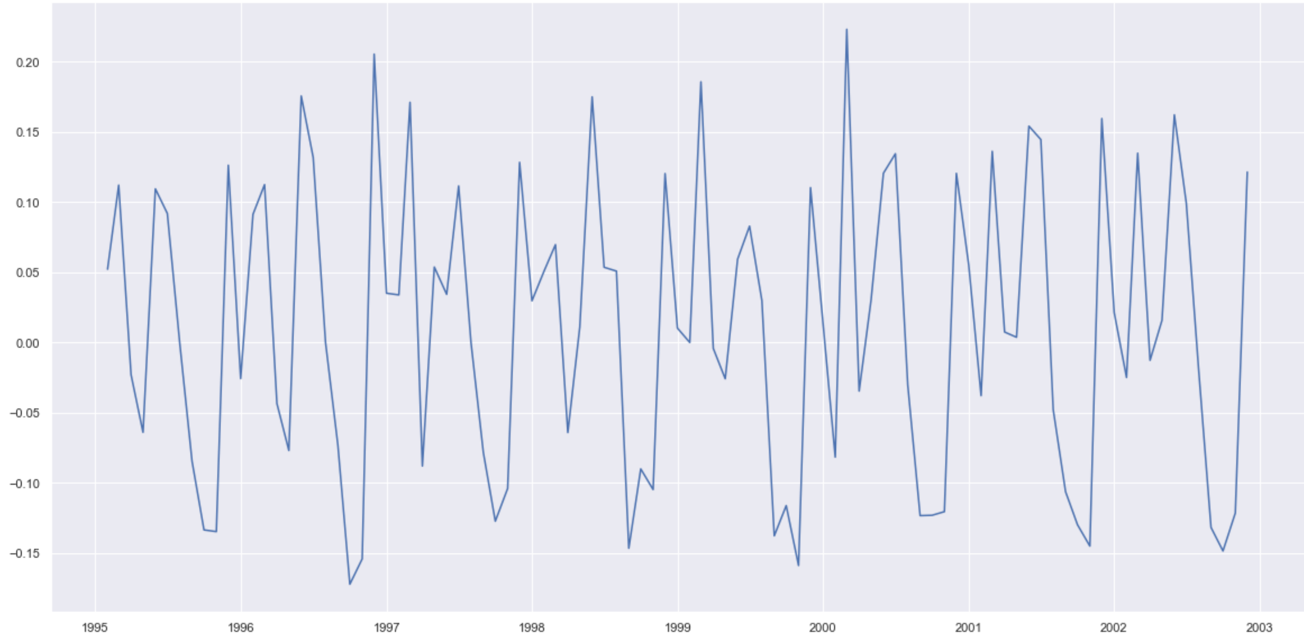
In [52]:
```
exp_data_diff=data_log-exp_data
stationarity(exp_data_diff)
```



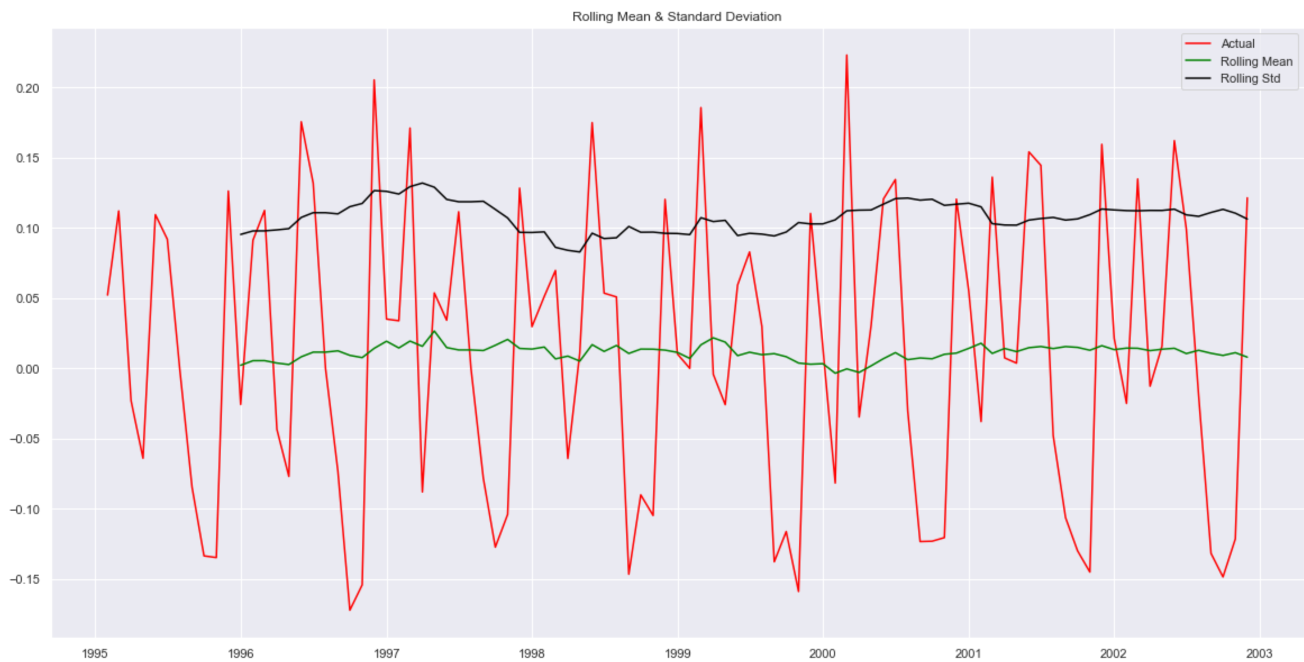

```
Dickey-Fuller Test:
Test Statistic          -2.835036
p-value                  0.053441
Lags Used               12.000000
No. of Obs              83.000000
Critical Value (1%)     -3.511712
Critical Value (5%)     -2.897048
Critical Value (10%)    -2.585713
dtype: float64
```

In [53]:
```
plt.figure(figsize=(20,10))
data_shift=data_log-data_log.shift()
plt.plot(data_shift)
```

Out[53]: [<matplotlib.lines.Line2D at 0x1cf64d5c280>]



In [54]:
```
data_shift=data_shift.dropna()
stationarity(data_shift)
```

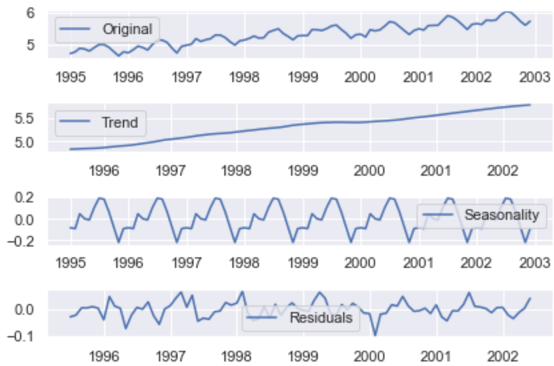



```
Dickey-Fuller Test:
Test Statistic           -2.670823
p-value                   0.079225
Lags Used                12.000000
No. of Obs               82.000000
Critical Value (1%)      -3.512738
Critical Value (5%)      -2.897490
Critical Value (10%)     -2.585949
dtype: float64
```

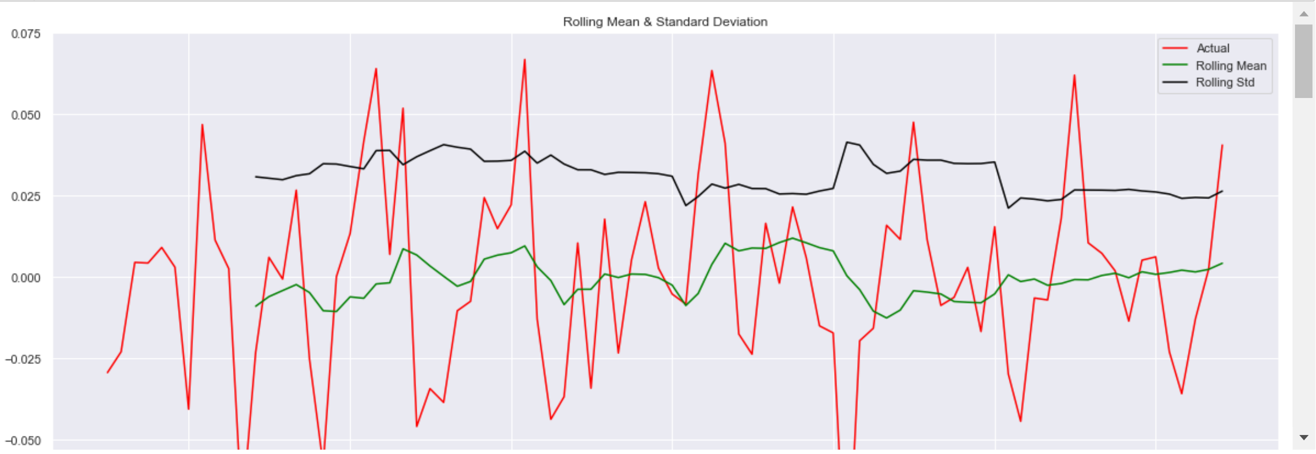
In [71]:
```
!pip install = statsmodels --upgrade
```

```
WARNING: Ignoring invalid distribution -tatsmodels (c:\users\admin\anaconda3\lib\site-packages)
ERROR: Invalid requirement: '='
Hint: = is not a valid operator. Did you mean == ?
WARNING: Ignoring invalid distribution -tatsmodels (c:\users\admin\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -tatsmodels (c:\users\admin\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -tatsmodels (c:\users\admin\anaconda3\lib\site-packages)
```

In [73]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
decomp=seasonal_decompose(data_log)

trend=decomp.trend
seasonal=decomp.seasonal
residual=decomp.resid

plt.subplot(411)
plt.plot(data_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 [75]:
```
decomp_data=residual
decomp_data=decomp_data.dropna()
stationarity(decomp_data)
```

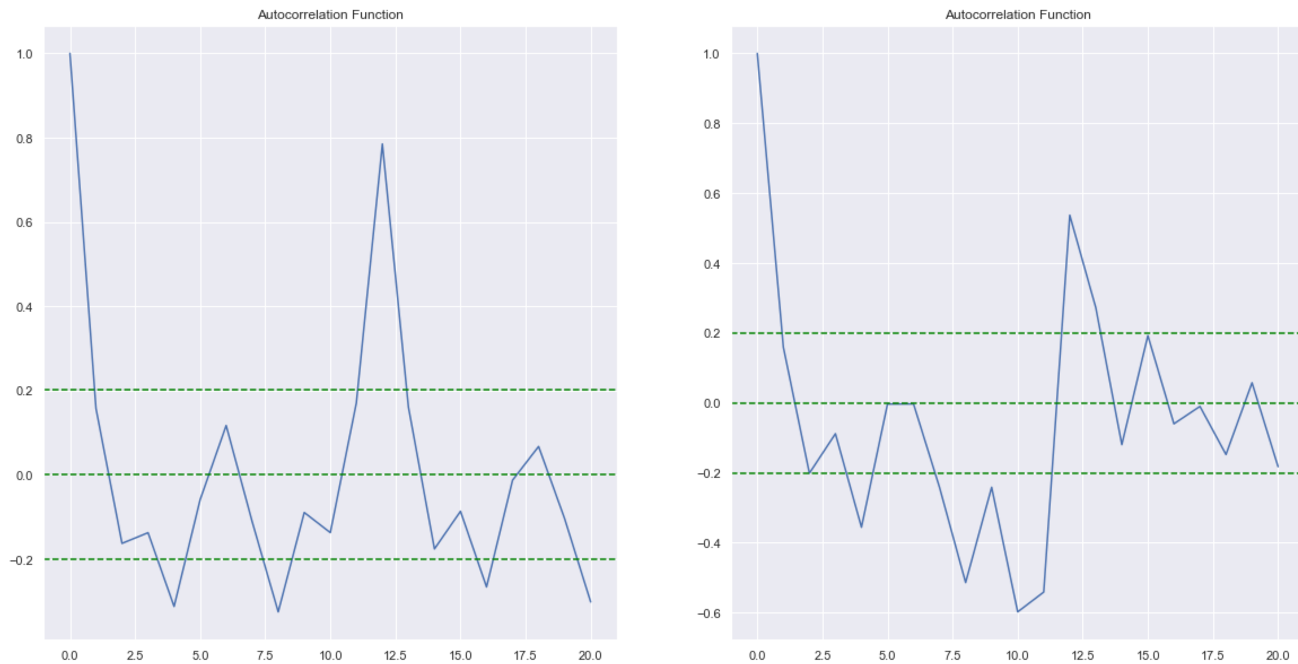


In [76]:

```
from statsmodels.tsa.stattools import acf, pacf

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

plt.figure(figsize=(20,10))
plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--',color='green')
plt.axhline(y=-1.96/np.sqrt(len(data_shift)),linestyle='--',color='green')
plt.axhline(y=1.96/np.sqrt(len(data_shift)),linestyle='--',color='green')
plt.title('Autocorrelation Function')

plt.subplot(122)
plt.plot(lag_pacf)
plt.axhline(y=0,linestyle='--',color='green')
plt.axhline(y=-1.96/np.sqrt(len(data_shift)),linestyle='--',color='green')
plt.axhline(y=1.96/np.sqrt(len(data_shift)),linestyle='--',color='green')
plt.title('Autocorrelation Function')
```

Out[76]:

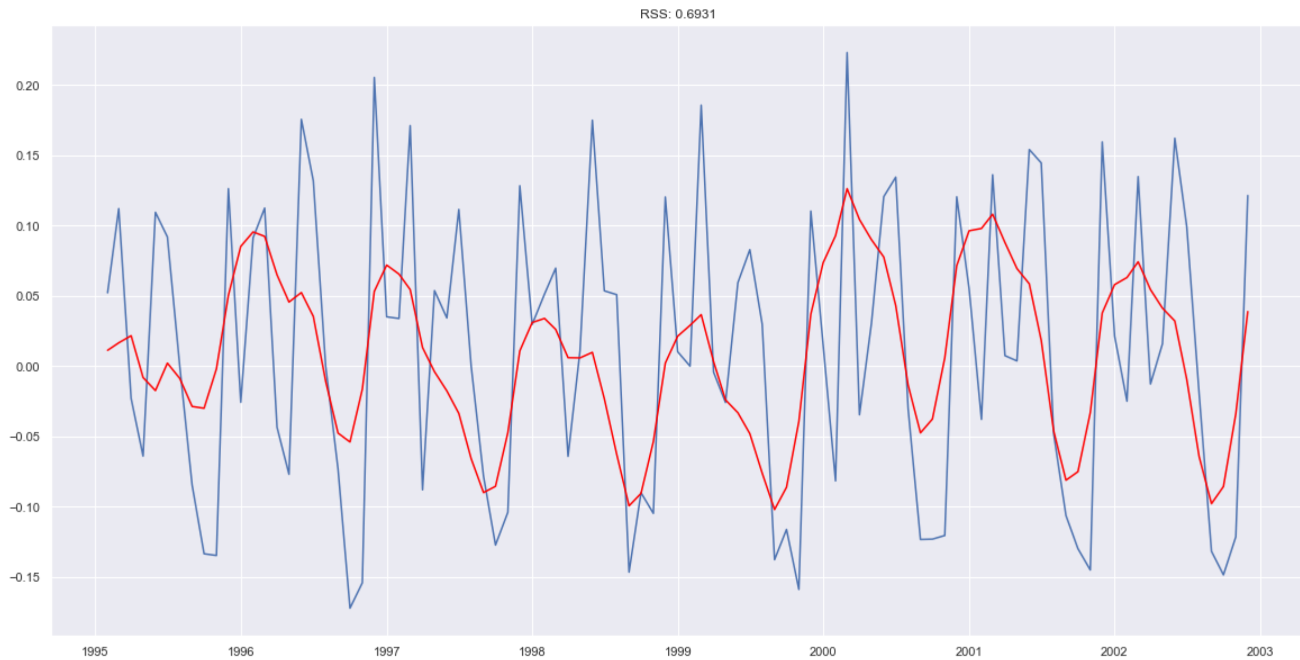
```
Text(0.5, 1.0, 'Autocorrelation Function')
```



ARIMA model

In [77]:
```
from statsmodels.tsa.arima_model import ARIMA

plt.figure(figsize=(20,10))
model=ARIMA(data_log, order=(2,1,2))
results=model.fit(disp=-1)
plt.plot(data_shift)
plt.plot(results.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((results.fittedvalues-data_shift['Passengers'])**2))
print('plotting ARIMA model')
```

```
plotting ARIMA model
```



In [78]:
```
predictions=pd.Series(results.fittedvalues, copy=True)
print(predictions.head())
```

```
Month
1995-02-01    0.011261
1995-03-01    0.016602
1995-04-01    0.021662
1995-05-01   -0.008096
1995-06-01   -0.017394
dtype: float64
```

In [79]:
```
predictions_cum_sum=predictions.cumsum()
print(predictions_cum_sum.head())
```

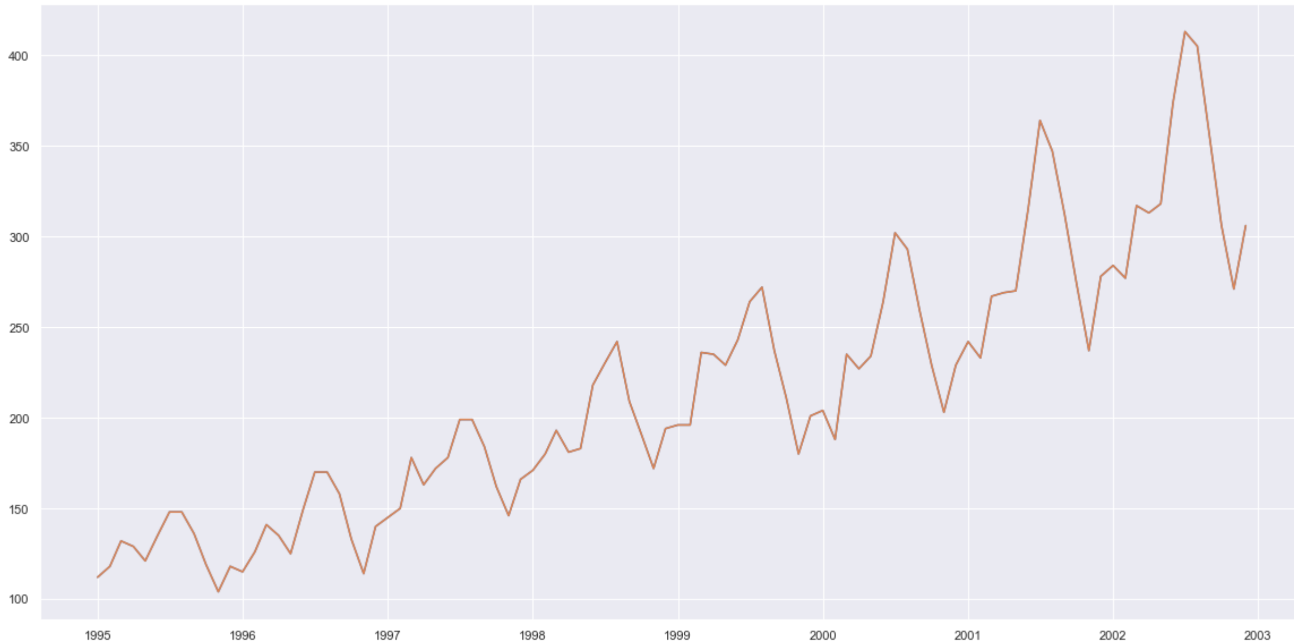
```
Month
1995-02-01    0.011261
1995-03-01    0.027863
1995-04-01    0.049525
1995-05-01    0.041428
1995-06-01    0.024034
dtype: float64
```

In [80]:
```
predictions_log=pd.Series(data_log['Passengers'])
predictions_log=predictions_log.add(predictions_cum_sum,fill_value=0)
predictions_log.head()
```

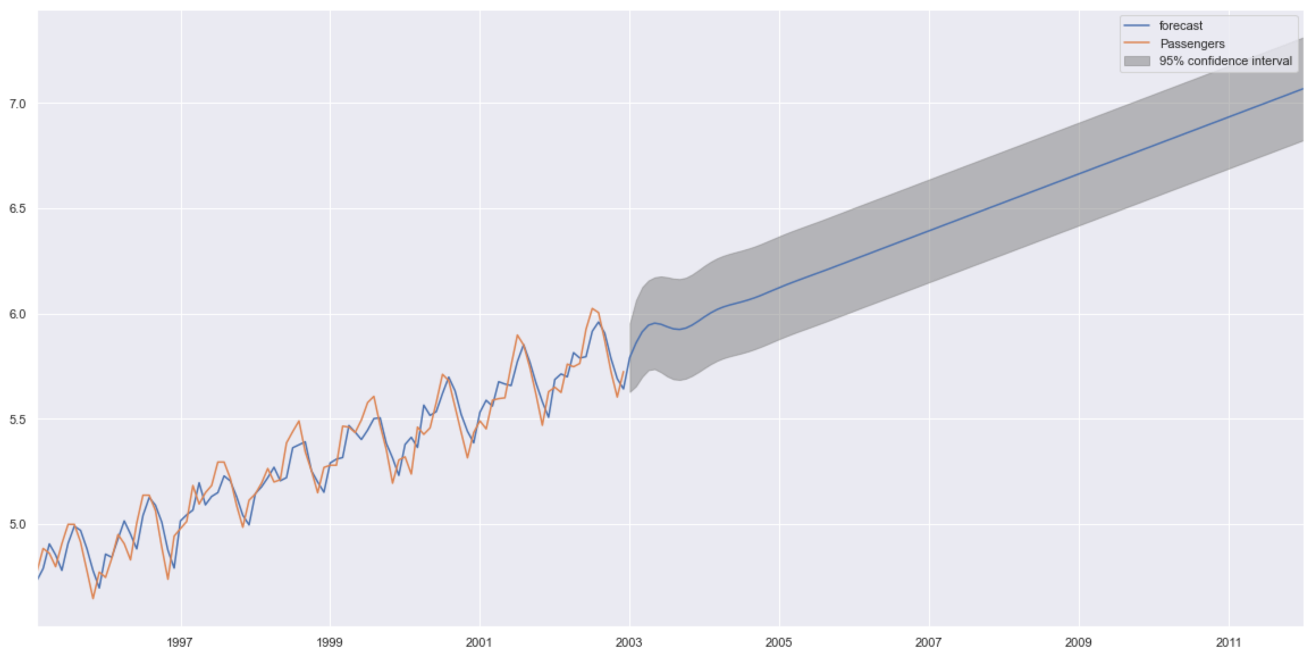
Out[80]:
```
Month
1995-01-01    4.718499
1995-02-01    4.781946
1995-03-01    4.910665
1995-04-01    4.909337
1995-05-01    4.837219
dtype: float64
```

In [81]:
```
predictions_ARIMA=np.exp(pd.Series(data_log['Passengers']))
plt.figure(figsize=(20,10))
plt.plot(data)
plt.plot(predictions_ARIMA)
```

Out[81]: [<matplotlib.lines.Line2D at 0x1cf6a379880>]



In [82]:
```
rcParams['figure.figsize']=20,10
results.plot_predict(1,204)
x=results.forecast(steps=120)
```



In [83]:
```
results.forecast(steps=120)
```

Out[83]:
```
(array([5.79002956, 5.85832606, 5.91213824, 5.94370241, 5.9535906 ,
        5.94822761, 5.93647701, 5.92650259, 5.92369917, 5.92996826,
        5.94416472, 5.9632639 , 5.98372467, 6.00261562, 6.01825815,
        6.03034271, 6.03963296, 6.04745467, 6.05517127, 6.06379517,
        6.07380483, 6.08516093, 6.09746165, 6.11015519, 6.12273469,
        6.13486621, 6.14643321, 6.15750821, 6.16827998, 6.17896877,
        6.18975641, 6.20074664, 6.21195806, 6.22334274, 6.23481806,
        6.24629929, 6.25772369, 6.26906178, 6.28031623, 6.29151218,
        6.30268397, 6.31386308, 6.32507017, 6.33631239, 6.34758514,
        6.35887665, 6.37017321, 6.38146355, 6.39274125, 6.4040052 ,
        6.41525841, 6.42650603, 6.43775332, 6.44900416, 6.46026039,
        6.4715218 , 6.48278685, 6.49405337, 6.50531943, 6.51658374,
        6.52784586, 6.53910609, 6.55036516, 6.56162392, 6.57288308,
        6.58414302, 6.59540381, 6.60666527, 6.61792708, 6.62918891,
        6.64045051, 6.65171177, 6.66297272, 6.67423344, 6.68549407,
        6.69675473, 6.7080155 , 6.7192764 , 6.73053742, 6.74179851,
        6.75305963, 6.76432071, 6.77558175, 6.78684274, 6.79810368,
        6.8093646 , 6.82062553, 6.83188646, 6.84314742, 6.85440839,
        6.86566938, 6.87693038, 6.88819137, 6.89945236, 6.91071334,
```

In [ ]:
```

```