

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]: 1 # import Libraries
        2 import pandas as pd
        3 import numpy as np
        4 import seaborn as sns
        5 import matplotlib.pyplot as plt
        6 import statsmodels.formula.api as smf
```

```
In [2]: 1 df = pd.read_excel('CocaCola_Sales_Rawdata.xlsx')
        2 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()
```

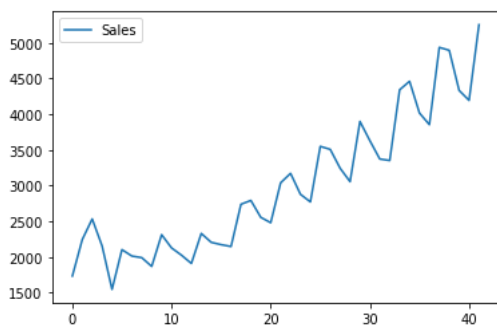
```
Out[3]:
```

	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

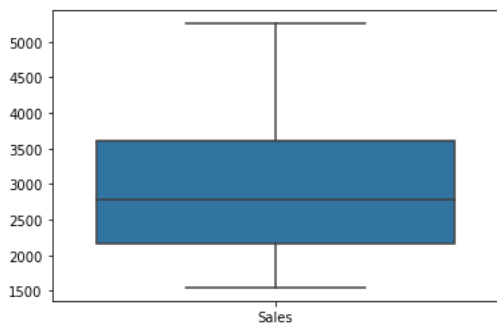
```
In [4]: 1 df.plot()
```

```
Out[4]: <AxesSubplot:>
```



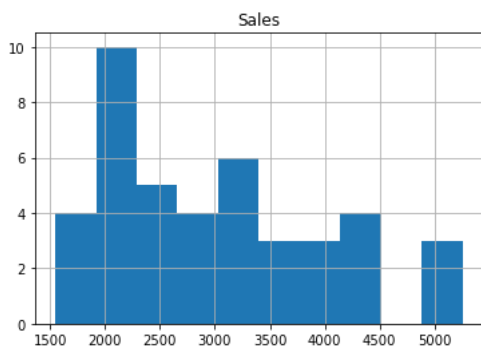
```
In [5]: 1 import seaborn as sns  
2 sns.boxplot(data=df)
```

```
Out[5]: <AxesSubplot:>
```



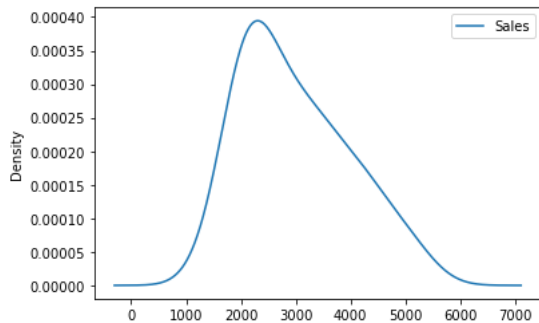
```
In [6]: 1 df.hist()
```

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



```
In [7]: 1 df.plot(kind="kde")
```

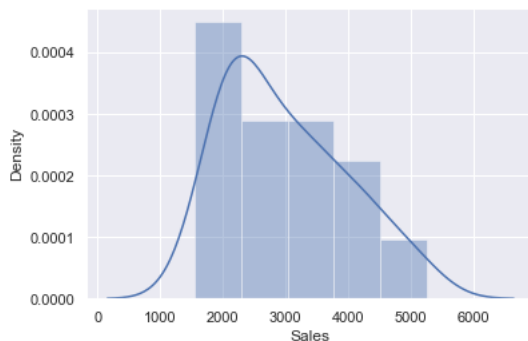
```
Out[7]: <AxesSubplot:ylabel='Density'>
```



```
In [8]: 1 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. ])
```

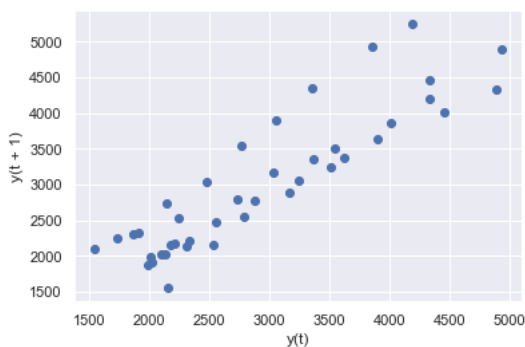
```
In [9]: 1 import seaborn as sns
2 import warnings
3 warnings.filterwarnings('ignore')
4 sns.set_theme()
5 rf = sns.distplot(df['Sales'],kde=True)
```



```
In [10]: 1 from pandas.plotting import lag_plot
2 import warnings
3 warnings.filterwarnings('ignore')
4 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]: 1 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]: 1 len(df)

Out[12]: 42

In [13]:

```
1 df['quarter'] = 0
2 for i in range(42):
3     p=df['Quarter'][i]
4     df['quarter'][i]=p[0:2]
```

In [14]:

1 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]:

1 df['quarter'].value_counts()

Out[15]:

```
Q1    11
Q2    11
Q3    10
Q4    10
Name: quarter, dtype: int64
```

In [16]:

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

In [17]: 1 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]: 1 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]: 1 cc['t'] = np.arange(1,43)
2 cc['t_squared'] = cc['t']**2
3 cc["Sales_log"] = np.log(df['Sales'])
```

```
In [20]: 1 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]: 1 train = cc.head(32)
2 test = cc.tail(10)
```

```
In [22]: 1 df['Sales'].plot()
```

```
Out[22]: <AxesSubplot:>
```



Model

```
In [23]: 1 from sklearn.metrics import mean_squared_error
```

```
In [24]: 1 # Linear Model
2 linear_model = smf.ols("Sales~t",data=train).fit()
3 linear_pred = pd.Series(linear_model.predict(test['t']))
4 linear_rmse = np.sqrt(mean_squared_error(np.array(test['Sales']),np.array(linear_pred)))
5 linear_rmse
```

```
Out[24]: 752.9233932767132
```

```
In [25]: 1 # Quadratic model
2 quad_model = smf.ols("Sales~t+t_squared",data=train).fit()
3 quad_pred = pd.Series(quad_model.predict(test[['t','t_squared']]))
4 quad_rmse = np.sqrt(mean_squared_error(np.array(test['Sales']),np.array(quad_pred)))
5 quad_rmse
```

```
Out[25]: 457.7357355407399
```

```
In [26]: 1 # Exponential model
2 exp_model = smf.ols("Sales_log~t",data=train).fit()
3 exp_pred = pd.Series(exp_model.predict(test['t']))
4 exp_rmse = np.sqrt(mean_squared_error(np.array(test['Sales']),np.array(exp_pred)))
5 exp_rmse
```

```
Out[26]: 4387.940544839098
```



```
In [27]: 1 data = {"MODEL":pd.Series(["rmse_linear","rmse_exp","rmse_quad"]), "RMSE_Values":pd.Series([linear_rmse,exp_rmse,quad_rmse,])
2 table_rmse=pd.DataFrame(data)
3 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]: 1 data = pd.read_excel('CocaCola_Sales_Rawdata.xlsx',header=0,index_col=0, parse_dates=True)
2 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]: 1 #separate out a validation dataset
2 split_point = len(data) - 7
3 dataset_cc, validation_cc = data[0:split_point], data[split_point:]
4 print('Dataset_cc %d, Validation_cc %d' % (len(dataset_cc), len(validation_cc)))
```

Dataset_cc 35, Validation_cc 7

```
In [30]: 1 dataset_cc.to_csv('dataset_cc.csv', header=False)
2 validation_cc.to_csv('validation_cc.csv', header=False)
```

```
In [31]: 1 from pandas import read_csv
2         from sklearn.metrics import mean_squared_error
3         from math import sqrt
4
5         train = read_csv('dataset_cc.csv', header=None, index_col=0, parse_dates=True, squeeze=True)
6
7         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]: 1 from pandas import read_csv
2 from sklearn.metrics import mean_squared_error
3 from math import sqrt
4
5 train = read_csv('dataset_cc.csv', header=None, index_col=0, parse_dates=True, squeeze=True)
6
7 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]: 1 X = train.values
2 X = X.astype('float32')
3 train_size = int(len(X) * 0.50)
4 train, test = X[0:train_size], X[train_size:]
```

Validation

```
In [34]: 1 history = [x for x in train]
2 predictions = list()
3 for i in range(len(test)):
4     yhat = history[-1]
5     predictions.append(yhat)
6     # observation
7     obs = test[i]
8     history.append(obs)
9     print('>Predicted=%.3f, Expected=%.3f' % (yhat, obs))
10 # report performance
11 rmse = sqrt(mean_squared_error(test, predictions))
12 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]: 1 data = {"MODEL":pd.Series(["rmse_linear","rmse_exp","rmse_quad","RMSE_ARIMA"]), "RMSE_Values":pd.Series([linear_rmse,exp_rmse])
2 table_rmse=pd.DataFrame(data)
3 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 [ ]: 1
```

```
In [ ]: 1
```

```
In [ ]: 1
```

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]: 1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 %matplotlib inline
5 from matplotlib.pyplot import rcParams
6 from datetime import datetime
7 import warnings
8 warnings.filterwarnings('ignore')
9
```

```
In [37]: 1 data = pd.read_excel("Airlines+Data.xlsx")
          2 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]: 1 data.shape
```

```
Out[38]: (96, 2)
```

```
In [39]: 1 data['Month']=pd.to_datetime(data['Month'], infer_datetime_format=True)
          2 data=data.set_index(['Month'])
```

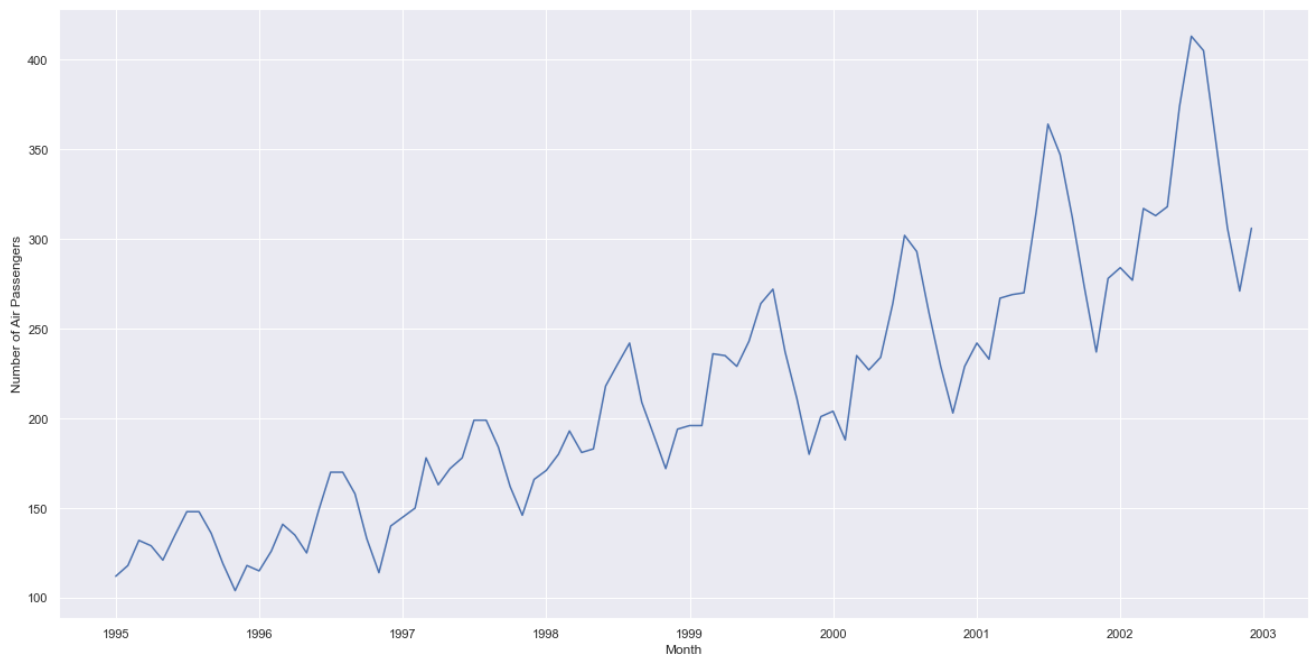
```
In [40]: 1 data.head()
```

```
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]: 1 plt.figure(figsize=(20,10))
          2 plt.xlabel("Month")
          3 plt.ylabel("Number of Air Passengers")
          4 plt.plot(data)
```

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



```
In [42]: 1 rolmean=data.rolling(window=12).mean()
2         rolstd=data.rolling(window=12).std()
3         print(rolmean.head(15))
4         print(rolstd.head(15))
```

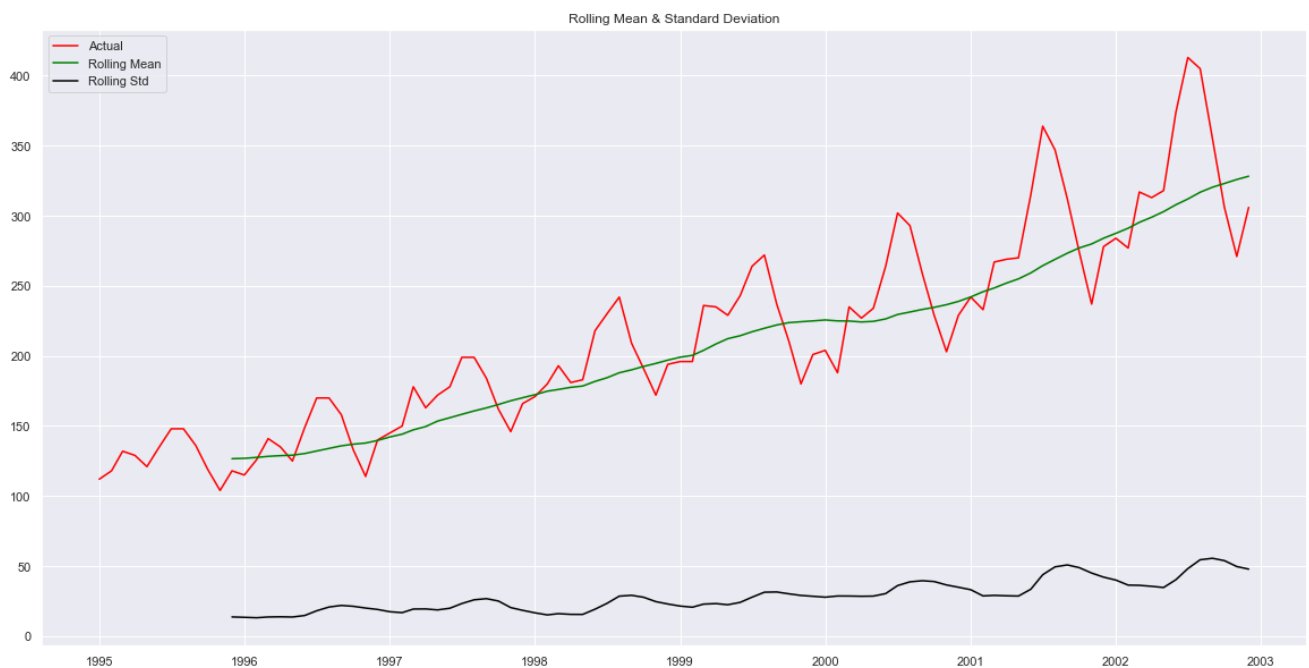
Passengers

Month	Passengers
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	Passengers
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]: 1 plt.figure(figsize=(20,10))
2         actual=plt.plot(data, color='red', label='Actual')
3         mean_6=plt.plot(rolmean, color='green', label='Rolling Mean')
4         std_6=plt.plot(rolstd, color='black', label='Rolling Std')
5         plt.legend(loc='best')
6         plt.title('Rolling Mean & Standard Deviation')
7         plt.show(block=False)
```

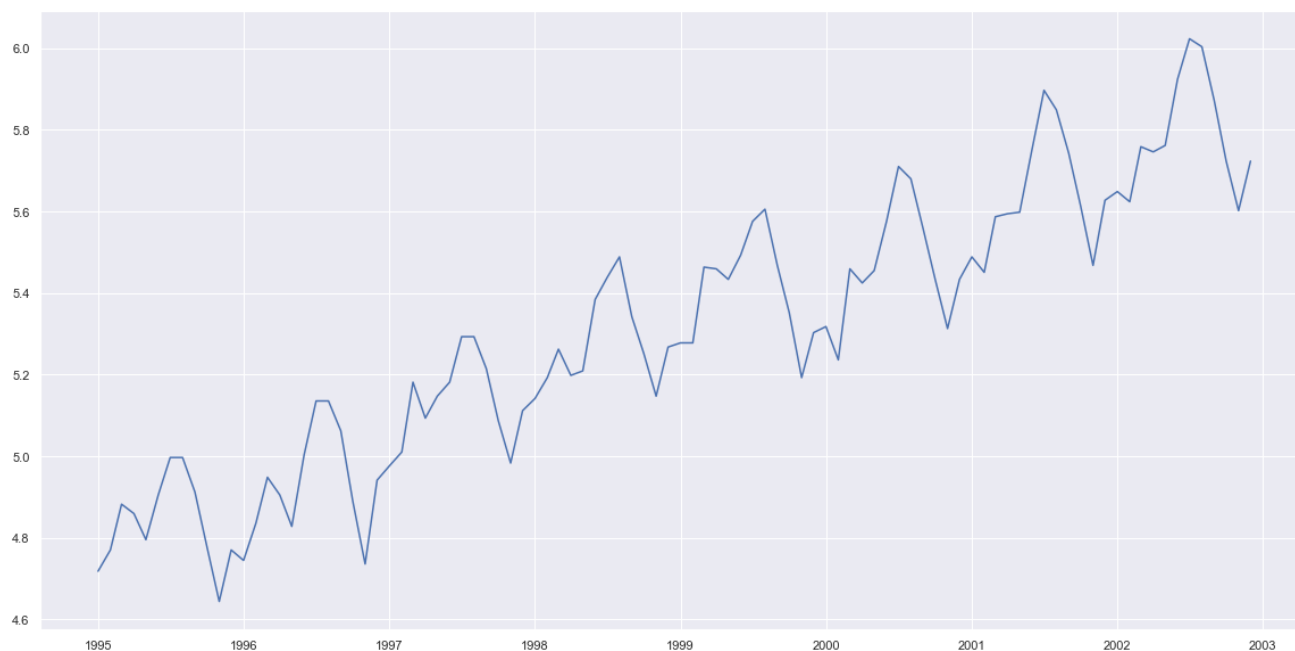


```
In [44]: 1 from statsmodels.tsa.stattools import adfuller
2 print('Dickey-Fuller Test: ')
3 dfest=adfuller(data['Passengers'], autolag='AIC')
4 dfoutput=pd.Series(dfest[0:4], index=['Test Statistic','p-value','Lags Used','No. of Obs'])
5 for key,value in dfest[4].items():
6     dfoutput['Critical Value (%)'%key] = value
7 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
```

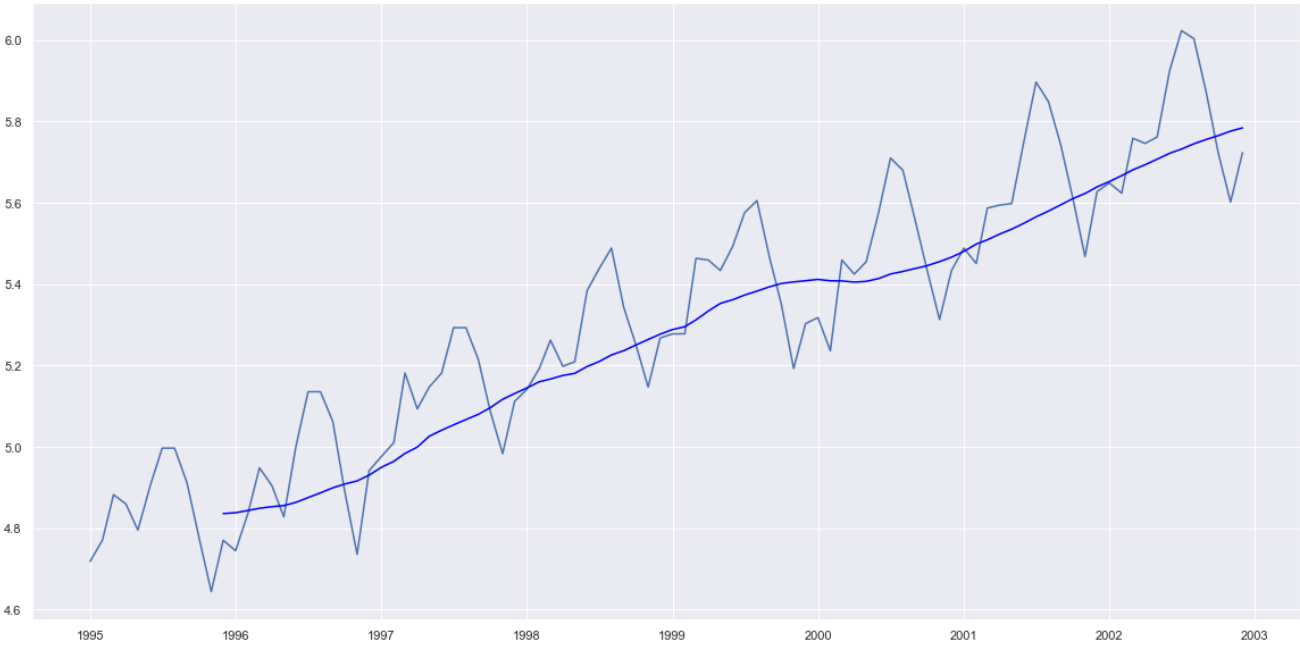
```
In [45]: 1 plt.figure(figsize=(20,10))
2 data_log=np.log(data)
3 plt.plot(data_log)
```

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



```
In [46]: 1 plt.figure(figsize=(20,10))
2 MAvg=data_log.rolling(window=12).mean()
3 MStd=data_log.rolling(window=12).std()
4 plt.plot(data_log)
5 plt.plot(MAVg, color='blue')
```

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



```
In [47]: 1 data_log_diff=data_log-MAvg
2 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]: 1 data_log_diff=data_log_diff.dropna()
2 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


```

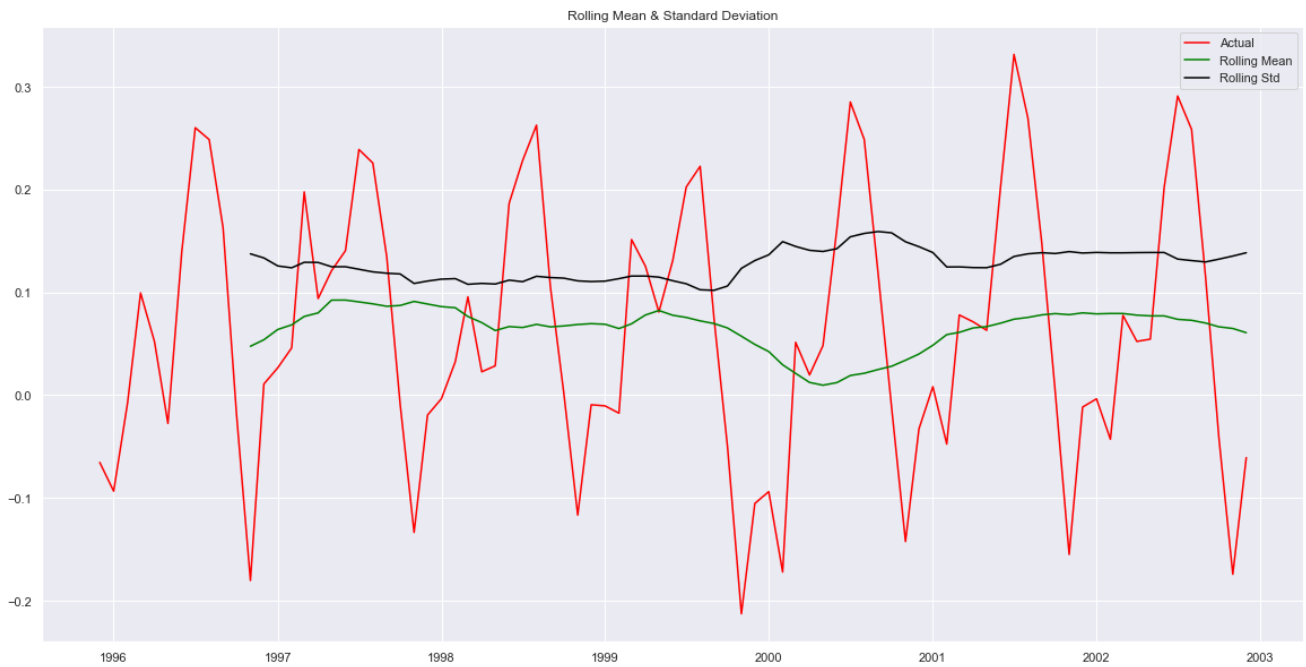
In [49]: 1 def stationarity(timeseries):
2
3     rolmean=timeseries.rolling(window=12).mean()
4     rolstd=timeseries.rolling(window=12).std()
5
6     plt.figure(figsize=(20,10))
7     actual=plt.plot(timeseries, color='red', label='Actual')
8     mean_6=plt.plot(rolmean, color='green', label='Rolling Mean')
9     std_6=plt.plot(rolstd, color='black', label='Rolling Std')
10    plt.legend(loc='best')
11    plt.title('Rolling Mean & Standard Deviation')
12    plt.show(block=False)
13
14    print('Dickey-Fuller Test: ')
15    dfctest=adfuller(timeseries['Passengers'], autolag='AIC')
16    dfoutput=pd.Series(dfctest[0:4], index=['Test Statistic', 'p-value', 'Lags Used', 'No. of Obs'])
17    for key,value in dfctest[4].items():
18        dfoutput['Critical Value (%)'%key] = value
19    print(dfoutput)

```

```

In [50]: 1 stationarity(data_log_diff)

```



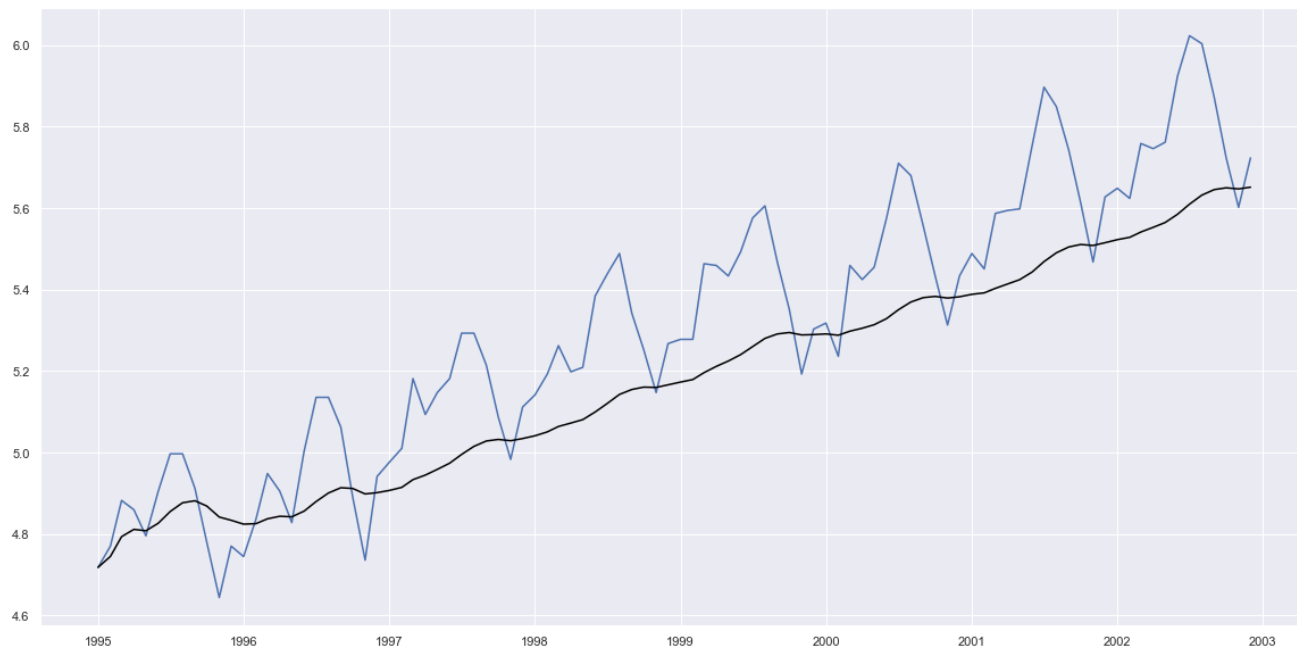
```

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

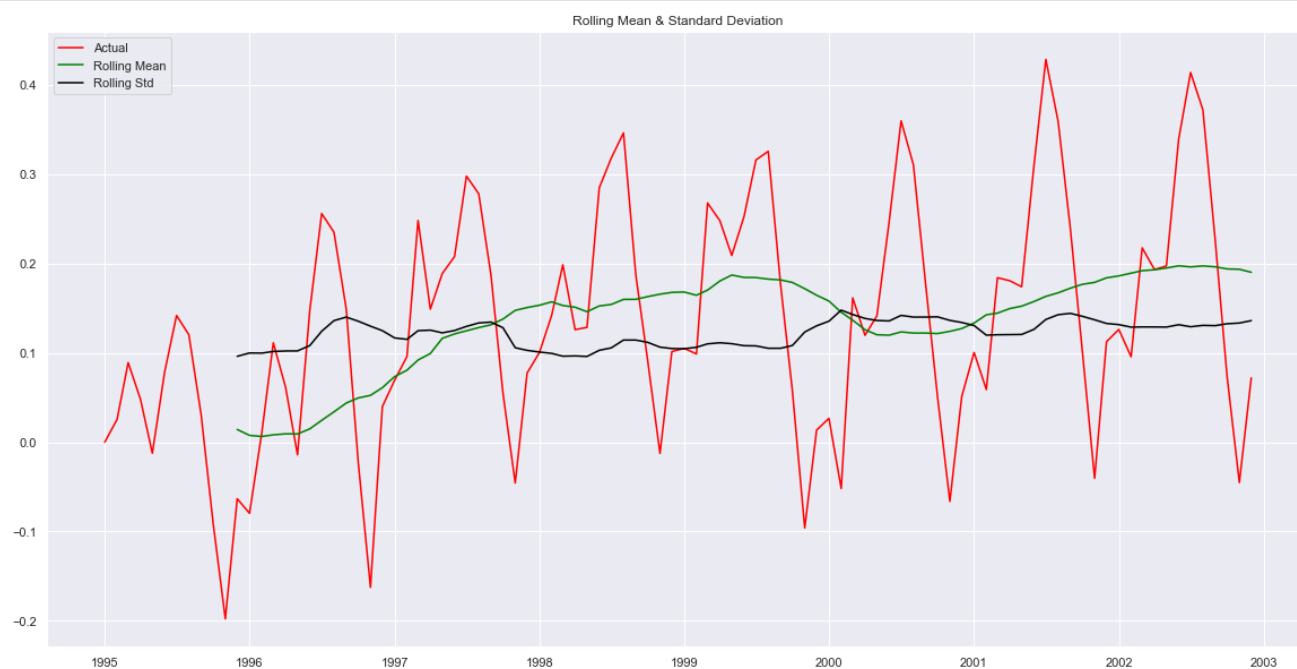
```

```
In [51]: 1 plt.figure(figsize=(20,10))
2 exp_data=data_log.ewm(halflife=12, min_periods=0, adjust=True).mean()
3 plt.plot(data_log)
4 plt.plot(exp_data, color='black')
```

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



```
In [52]: 1 exp_data_diff=data_log-exp_data
2 stationarity(exp_data_diff)
```

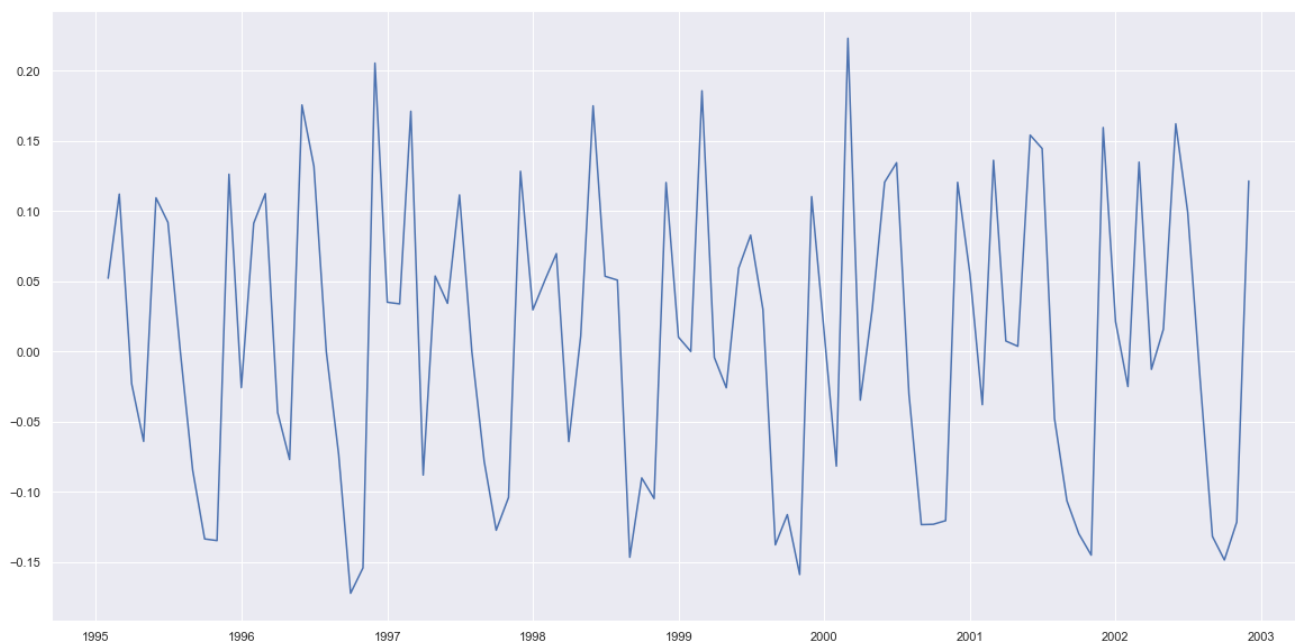


Dickey-Fuller Test:

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]: 1 plt.figure(figsize=(20,10))
2 data_shift=data_log-data_log.shift()
3 plt.plot(data_shift)
```

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



```
In [54]: 1 data_shift=data_shift.dropna()
2 stationarity(data_shift)
```

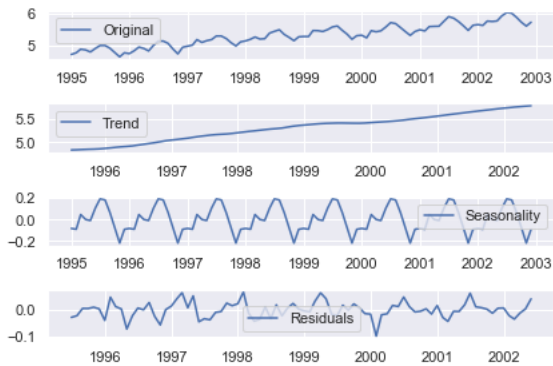


```
Dickey-Fuller Test:
Test Statistic      -2.670823
p-value              0.079225
Lags Used            12.000000
No. of Obs           82.000000
Critical Value (1%)  -3.512738
Critical Value (5%)  -2.897490
Critical Value (10%) -2.585949
dtype: float64
```

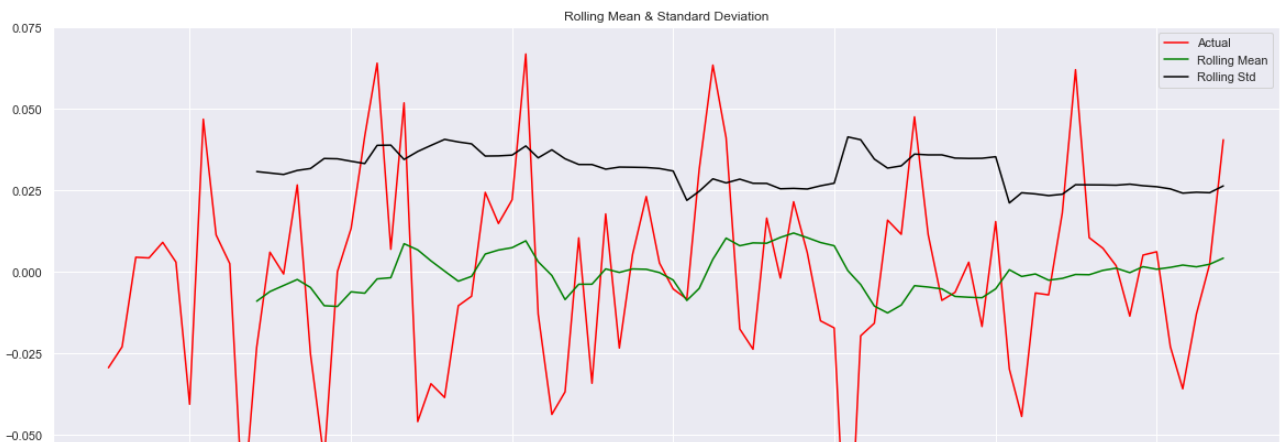
In [71]: 1 !pip install = statsmodels --upgrade

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]: 1 from statsmodels.tsa.seasonal import seasonal_decompose
 2 decomp=seasonal_decompose(data_log)
 3
 4 trend=decomp.trend
 5 seasonal=decomp.seasonal
 6 residual=decomp.resid
 7
 8 plt.subplot(411)
 9 plt.plot(data_log, label='Original')
 10 plt.legend(loc='best')
 11 plt.subplot(412)
 12 plt.plot(trend, label='Trend')
 13 plt.legend(loc='best')
 14 plt.subplot(413)
 15 plt.plot(seasonal, label='Seasonality')
 16 plt.legend(loc='best')
 17 plt.subplot(414)
 18 plt.plot(residual, label='Residuals')
 19 plt.legend(loc='best')
 20 plt.tight_layout()



In [75]: 1 decomp_data=residual
 2 decomp_data=decomp_data.dropna()
 3 stationarity(decomp_data)

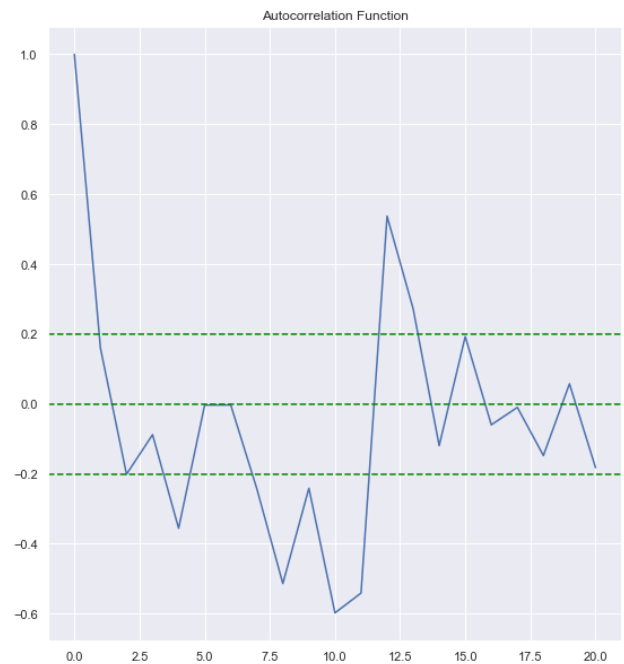
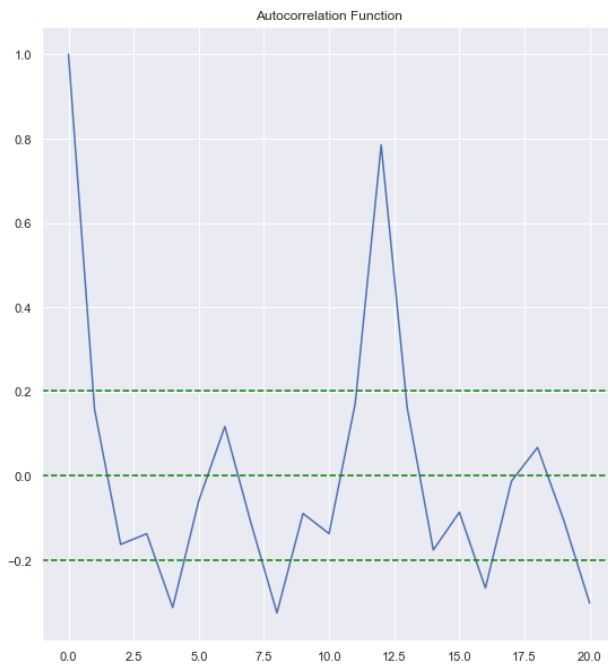


```

In [76]: 1 from statsmodels.tsa.stattools import acf, pacf
2
3 lag_acf=acf(data_shift, nlags=20)
4 lag_pacf=pacf(data_shift, nlags=20, method='ols')
5
6 plt.figure(figsize=(20,10))
7 plt.subplot(121)
8 plt.plot(lag_acf)
9 plt.axhline(y=0,linestyle='--',color='green')
10 plt.axhline(y=-1.96/np.sqrt(len(data_shift)),linestyle='--',color='green')
11 plt.axhline(y=1.96/np.sqrt(len(data_shift)),linestyle='--',color='green')
12 plt.title('Autocorrelation Function')
13
14 plt.subplot(122)
15 plt.plot(lag_pacf)
16 plt.axhline(y=0,linestyle='--',color='green')
17 plt.axhline(y=-1.96/np.sqrt(len(data_shift)),linestyle='--',color='green')
18 plt.axhline(y=1.96/np.sqrt(len(data_shift)),linestyle='--',color='green')
19 plt.title('Autocorrelation Function')

```

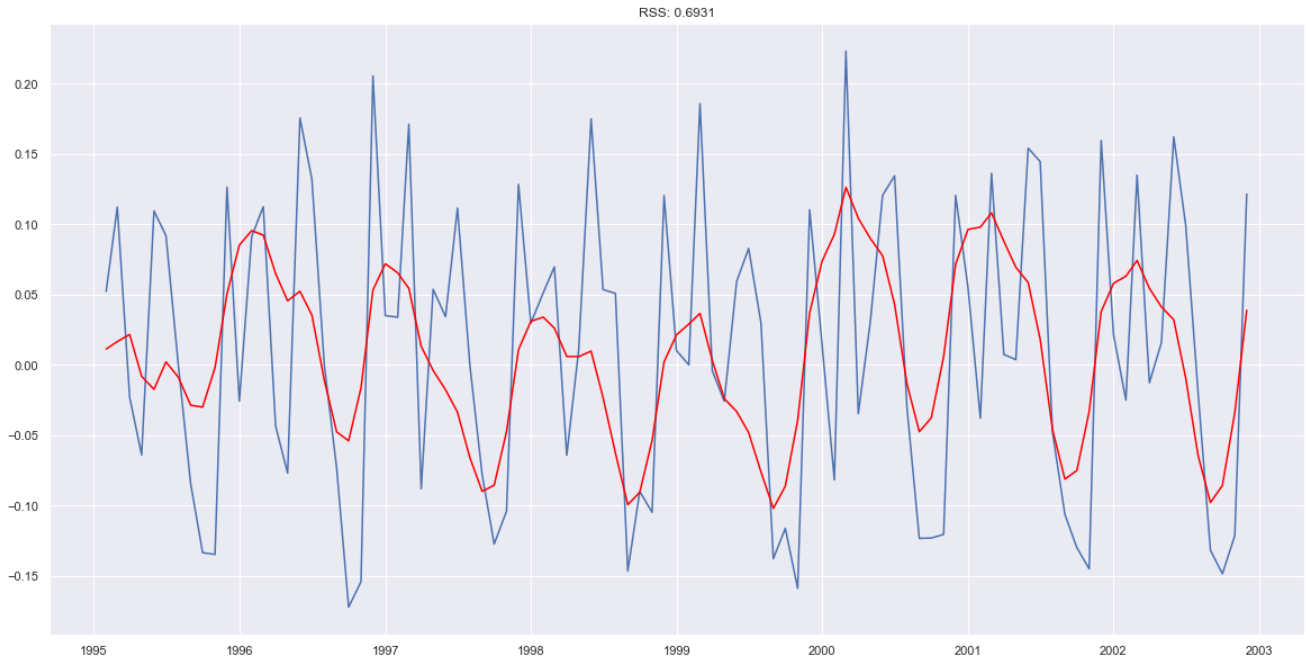
Out[76]: Text(0.5, 1.0, 'Autocorrelation Function')



ARIMA model

```
In [77]: 1 from statsmodels.tsa.arima_model import ARIMA
2
3 plt.figure(figsize=(20,10))
4 model=ARIMA(data_log, order=(2,1,2))
5 results=model.fit(dis=-1)
6 plt.plot(data_shift)
7 plt.plot(results.fittedvalues, color='red')
8 plt.title('RSS: %.4f'% sum((results.fittedvalues-data_shift['Passengers'])**2))
9 print('plotting ARIMA model')
```

plotting ARIMA model



```
In [78]: 1 predictions=pd.Series(results.fittedvalues, copy=True)
2 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]: 1 predictions_cum_sum=predictions.cumsum()
2 print(predictions_cum_sum.head())
```

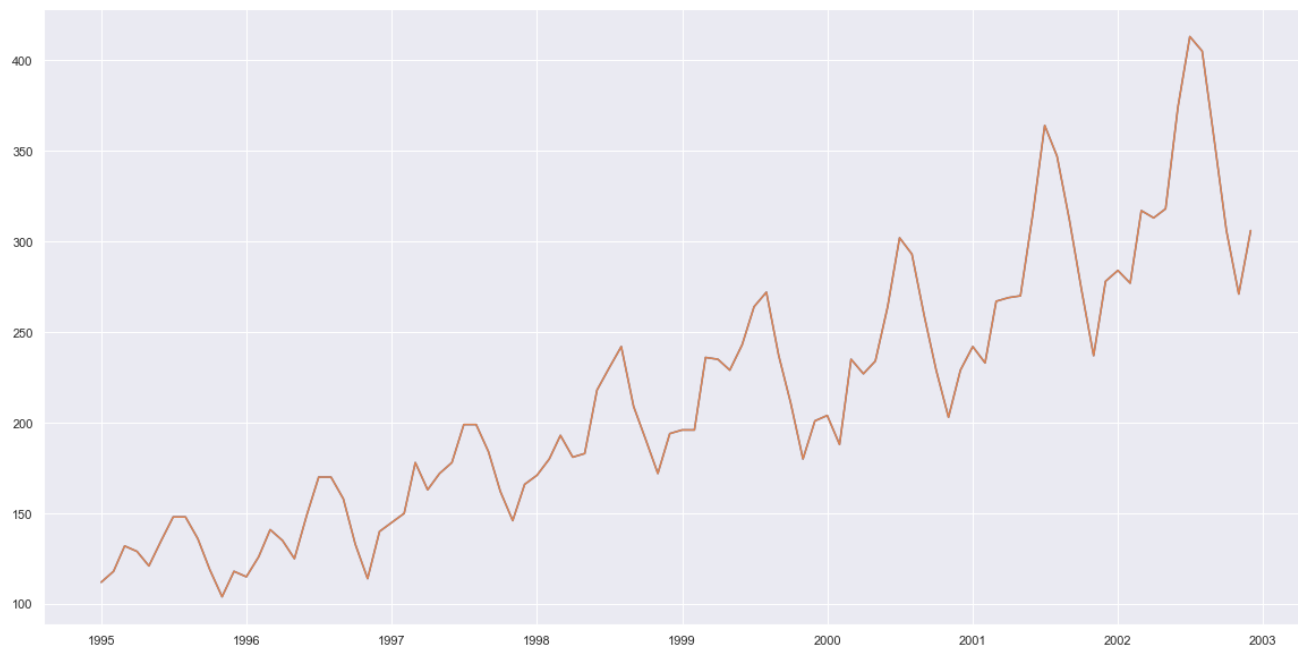
```
Month
1995-02-01    0.011261
1995-03-01    0.027863
1995-04-01    0.049525
1995-05-01    0.041428
1995-06-01    0.024034
dtype: float64
```

```
In [80]: 1 predictions_log=pd.Series(data_log['Passengers'])
2 predictions_log=predictions_log.add(predictions_cum_sum,fill_value=0)
3 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]: 1 predictions_ARIMA=np.exp(pd.Series(data_log['Passengers'])))  
2 plt.figure(figsize=(20,10))  
3 plt.plot(data)  
4 plt.plot(predictions_ARIMA)
```

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



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



```
In [83]: 1 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,
6.92197434, 6.93323534, 6.94449634, 6.95575734, 6.96701834,
6.97827934, 6.98954034, 7.00080134, 7.01206234, 7.02332334,
7.03458434, 7.04584534, 7.05710634, 7.06836734, 7.07962834,
7.09088934, 7.10215034, 7.11341134, 7.12467234, 7.13593334,
7.14719434, 7.15845534, 7.16971634, 7.18097734, 7.19223834,
7.20349934, 7.21476034, 7.22602134, 7.23728234, 7.24854334,
7.25980434, 7.27106534, 7.28232634, 7.29358734, 7.30484834,
7.31610934, 7.32737034, 7.33863134, 7.34989234, 7.36115334,
7.37241434, 7.38367534, 7.39493634, 7.40619734, 7.41745834,
7.42871934, 7.43998034, 7.45124134, 7.46250234, 7.47376334,
7.48502434, 7.49628534, 7.50754634, 7.51880734, 7.53006834,
7.54132934, 7.55259034, 7.56385134, 7.57511234, 7.58637334,
7.59763434, 7.60889534, 7.62015634, 7.63141734, 7.64267834,
7.65393934, 7.66520034, 7.67646134, 7.68772234, 7.69898334,
7.71024434, 7.72150534, 7.73276634, 7.74402734, 7.75528834,
7.76654934, 7.77781034, 7.78907134, 7.80033234, 7.81159334,
7.82285434, 7.83411534, 7.84537634, 7.85663734, 7.86789834,
7.87915934, 7.89042034, 7.90168134, 7.91294234, 7.92420334,
7.93546434, 7.94672534, 7.95798634, 7.96924734, 7.98050834,
7.99176934, 8.00303034, 8.01429134, 8.02555234, 8.03681334,
8.04807434, 8.05933534, 8.07059634, 8.08185734, 8.09311834,
8.10437934, 8.11564034, 8.12690134, 8.13816234, 8.14942334,
8.16068434, 8.17194534, 8.18320634, 8.19446734, 8.20572834,
8.21698934, 8.22825034, 8.23951134, 8.25077234, 8.26203334,
8.27329434, 8.28455534, 8.29581634, 8.30707734, 8.31833834,
8.32959934, 8.34086034, 8.35212134, 8.36338234, 8.37464334,
8.38590434, 8.39716534, 8.40842634, 8.41968734, 8.43094834,
8.44220934, 8.45347034, 8.46473134, 8.47599234, 8.48725334,
8.49851434, 8.50977534, 8.52103634, 8.53229734, 8.54355834,
8.55481934, 8.56608034, 8.57734134, 8.58860234, 8.59986334,
8.61112434, 8.62238534, 8.63364634, 8.64490734, 8.65616834,
8.66742934, 8.67869034, 8.68995134, 8.70121234, 8.71247334,
8.72373434, 8.73499534, 8.74625634, 8.75751734, 8.76877834,
8.78003934, 8.79130034, 8.80256134, 8.81382234, 8.82508334,
8.83634434, 8.84760534, 8.85886634, 8.87012734, 8.88138834,
8.89264934, 8.90391034, 8.91517134, 8.92643234, 8.93769334,
8.94895434, 8.96021534, 8.97147634, 8.98273734, 8.99399834,
9.00525934, 9.01652034, 9.02778134, 9.03904234, 9.05030334,
9.06156434, 9.07282534, 9.08408634, 9.09534734, 9.10660834,
9.11786934, 9.12913034, 9.14039134, 9.15165234, 9.16291334,
9.17417434, 9.18543534, 9.19669634, 9.20795734, 9.21921834,
9.23047934, 9.24174034, 9.25300134, 9.26426234, 9.27552334,
9.28678434, 9.29804534, 9.30930634, 9.32056734, 9.33182834,
9.34308934, 9.35435034, 9.36561134, 9.37687234, 9.38813334,
9.39939434, 9.41065534, 9.42191634, 9.43317734, 9.44443834,
9.45569934, 9.46696034, 9.47822134, 9.48948234, 9.50074334,
9.51200434, 9.52326534, 9.53452634, 9.54578734, 9.55704834,
9.56830934, 9.57957034, 9.59083134, 9.60209234, 9.61335334,
9.62461434, 9.63587534, 9.64713634, 9.65839734, 9.66965834,
9.68091934, 9.69218034, 9.70344134, 9.71470234, 9.72596334,
9.73722434, 9.74848534, 9.75974634, 9.77100734, 9.78226834,
9.79352934, 9.80479034, 9.81605134, 9.82731234, 9.83857334,
9.84983434, 9.86109534, 9.87235634, 9.88361734, 9.89487834,
9.90613934, 9.91740034, 9.92866134, 9.93992234, 9.95118334,
9.96244434, 9.97370534, 9.98496634, 10.00000000, 10.01000000,
10.02000000, 10.03000000, 10.04000000, 10.05000000, 10.06000000,
10.07000000, 10.08000000, 10.09000000, 10.10000000, 10.11000000,
10.12000000, 10.13000000, 10.14000000, 10.15000000, 10.16000000,
10.17000000, 10.18000000, 10.19000000, 10.20000000, 10.21000000,
10.22000000, 10.23000000, 10.24000000, 10.25000000, 10.26000000,
10.27000000, 10.28000000, 10.29000000, 10.30000000, 10.31000000,
10.32000000, 10.33000000, 10.34000000, 10.35000000, 10.36000000,
10.37000000, 10.38000000, 10.39000000, 10.40000000, 10.41000000,
10.42000000, 10.43000000, 10.44000000, 10.45000000, 10.46000000,
10.47000000, 10.48000000, 10.49000000, 10.50000000, 10.51000000,
10.52000000, 10.53000000, 10.54000000, 10.55000000, 10.56000000,
10.57000000, 10.58000000, 10.59000000, 10.60000000, 10.61000000,
10.62000000, 10.63000000, 10.64000000, 10.65000000, 10.66000000,
10.67000000, 10.68000000, 10.69000000, 10.70000000, 10.71000000,
10.72000000, 10.73000000, 10.74000000, 10.75000000, 10.76000000,
10.77000000, 10.78000000, 10.79000000, 10.80000000, 10.81000000,
10.82000000, 10.83000000, 10.84000000, 10.85000000, 10.86000000,
10.87000000, 10.88000000, 10.89000000, 10.90000000, 10.91000000,
10.92000000, 10.93000000, 10.94000000, 10.95000000, 10.96000000,
10.97000000, 10.98000000, 10.99000000, 11.00000000, 11.01000000,
11.02000000, 11.03000000, 11.04000000, 11.05000000, 11.06000000,
11.07000000, 11.08000000, 11.09000000, 11.10000000, 11.11000000,
11.12000000, 11.13000000, 11.14000000, 11.15000000, 11.16000000,
11.17000000, 11.18000000, 11.19000000, 11.20000000, 11.21000000,
11.22000000, 11.23000000, 11.24000000, 11.25000000, 11.26000000,
11.27000000, 11.28000000, 11.29000000, 11.30000000, 11.31000000,
11.32000000, 11.33000000, 11.34000000, 11.35000000, 11.36000000,
11.37000000, 11.38000000, 11.39000000, 11.40000000, 11.41000000,
11.42000000, 11.43000000, 11.44000000, 11.45000000, 11.46000000,
11.47000000, 11.48000000, 11.49000000, 11.50000000, 11.51000000,
11.52000000, 11.53000000, 11.54000000, 11.55000000, 11.56000000,
11.57000000, 11.58000000, 11.59000000, 11.60000000, 11.61000000,
11.62000000, 11.63000000, 11.64000000, 11.65000000, 11.66000000,
11.67000000, 11.68000000, 11.69000000, 11.70000000, 11.71000000,
11.72000000, 11.73000000, 11.74000000, 11.75000000, 11.76000000,
11.77000000, 11.78000000, 11.79000000, 11.80000000, 11.81000000,
11.82000000, 11.83000000, 11.84000000, 11.85000000, 11.86000000,
11.87000000, 11.88000000, 11.89000000, 11.90000000, 11.91000000,
11.92000000, 11.93000000, 11.94000000, 11.95000000, 11.96000000,
11.97000000, 11.98000000, 11.99000000, 12.00000000, 12.01000000,
12.02000000, 12.03000000, 12.04000000, 12.05000000, 12.06000000,
12.07000000, 12.08000000, 12.09000000, 12.10000000, 12.11000000,
12.12000000, 12.13000000, 12.14000000, 12.15000000, 12.16000000,
12.17000000, 12.18000000, 12.19000000, 12.20000000, 12.21000000,
12.22000000, 12.23000000, 12.24000000, 12.25000000, 12.26000000,
12.27000000, 12.28000000, 12.29000000, 12.30000000, 12.31000000,
12.32000000, 12.33000000, 12.34000000, 12.35000000, 12.36000000,
12.37000000, 12.38000000, 12.39000000, 12.40000000, 12.41000000,
12.42000000, 12.43000000, 12.44000000, 12.45000000, 12.46000000,
12.47000000, 12.48000000, 12.49000000, 12.50000000, 12.51000000,
12.52000000, 12.53000000, 12.54000000, 12.55000000, 12.56000000,
12.57000000, 12.58000000, 12.59000000, 12.60000000, 12.61000000,
12.62000000, 12.63000000, 12.64000000, 12.65000000, 12.66000000,
12.67000000, 12.68000000, 12.69000000, 12.70000000, 12.71000000,
12.72000000, 12.73000000, 12.74000000, 12.75000000, 12.76000000,
12.77000000, 12.78000000, 12.79000000, 12.80000000, 12.81000000,
12.82000000, 12.83000000, 12.84000000, 12.85000000, 12.86000000,
12.87000000, 12.88000000, 12.89000000, 12.90000000, 12.91000000,
12.92000000, 12.93000000, 12.94000000, 12.95000000, 12.96000000,
12.97000000, 12.98000000, 12.99000000, 13.00000000, 13.01000000,
13.02000000, 13.03000000, 13.04000000, 13.05000000, 13.06000000,
13.07000000, 13.08000000, 13.09000000, 13.10000000, 13.11000000,
13.12000000, 13.13000000, 13.14000000, 13.15000000, 13.16000000,
13.17000000, 13.18000000, 13.19000000, 13.20000000, 13.21000000,
13.22000000, 13.23000000, 13.24000000, 13.25000000, 13.26000000,
13.27000000, 13.28000000, 13.29000000, 13.30000000, 13.31000000,
13.32000000, 13.33000000, 13.34000000, 13.35000000, 13.36000000,
13.37000000, 13.38000000, 13.39000000, 13.40000000, 13.41000000,
13.42000000, 13.43000000, 13.44000000, 13.45000000, 13.46000000,
13.47000000, 13.48000000, 13.49000000, 13.50000000, 13.51000000,
13.52000000, 13.53000000, 13.54000000, 13.55000000, 13.56000000,
13.57000000, 13.58000000, 13.59000000, 13.60000000, 13.61000000,
13.62000000, 13.63000000, 13.64000000, 13.65000000, 13.66000000,
13.67000000, 13.68000000, 13.69000000, 13.70000000, 13.71000000,
13.72000000, 13.73000000, 13.74000000, 13.75000000, 13.76000000,
13.77000000, 13.78000000, 13.79000000, 13.80000000, 13.81000000,
13.82000000, 13.83000000, 13.84000000, 13.85000000, 13.86000000,
13.87000000, 13.88000000, 13.89000000, 13.90000000, 13.91000000,
13.92000000, 13.93000000, 13.94000000, 13.95000000, 13.96000000,
13.97000000, 13.98000000, 13.99000000, 14.00000000, 14.01000000,
14.02000000, 14.03000000, 14.04000000, 14.05000000, 14.06000000,
14.07000000, 14.08000000, 14.09000000, 14.10000000, 14.11000000,
14.12000000, 14.13000000, 14.14000000, 14.15000000, 14.16000000,
14.17000000, 14.18000000, 14.19000000, 14.20000000, 14.21000000,
14.22000000, 14.23000000, 14.24000000, 14.25000000, 14.26000000,
14.27000000, 14.28000000, 14.29000000, 14.30000000, 14.31000000,
14.32000000, 14.33000000, 14.34000000, 14.35000000, 14.36000000,
14.37000000, 14.38000000, 14.39000000, 14.40000000, 14.41000000,
14.42000000, 14.43000000, 14.44000000, 14.45000000, 14.46000000,
14.47000000, 14.48000000, 14.49000000, 14.50000000, 14.51000000,
14.52000000, 14.53000000, 14.54000000, 14.55000000, 14.56000000,
14.57000000, 14.58000000, 14.59000000, 14.60000000, 14.61000000,
14.62000000, 14.63000000, 14.64000000, 14.65000000, 14.66000000,
14.67000000, 14.68000000, 14.69000000, 14.70000000, 14.71000000,
14.72000000, 14.73000000, 14.74000000, 14.75000000, 14.76000000,
14.77000000, 14.78000000, 14.79000000, 14.80000000, 14.81000000,
14.82000000, 14.83000000, 14.84000000, 14.85000000, 14.86000000,
14.87000000, 14.88000000, 14.89000000, 14.90000000, 14.91000000,
14.92000000, 14.93000000, 14.94000000, 14.95000000, 14.96000000,
14.97000000, 14.98000000, 14.99000000, 15.00000000, 15.01000000,
15.02000000, 15.03000000, 15.04000000, 15.05000000, 15.06000000,
15.07000000, 15.08000000, 15.09000000, 15.10000000, 15.11000000,
15.12000000, 15.13000000, 15.14000000, 15.15000000, 15.16000000,
15.17000000, 15.18000000, 15.19000000, 15.20000000, 15.21000000,
15.22000000, 15.23000000, 15.24000000, 15.25000000, 15.26000000,
15.27000000, 15.28000000, 15.29000000, 15.30000000, 15.31000000,
15.32000000, 15.33000000, 15.34000000, 15.35000000, 15.36000000,
15.37000000, 15.38000000, 15.39000000, 15.40000000, 15.41000000,
15.42000000, 15.43000000, 15.44000000, 15.45000000, 15.46000000,
15.47000000, 15.48000000, 15.49000000, 15.50000000, 15.51000000,
15.52000000, 15.53000000, 15.54000000, 15.55000000, 15.56000000,
15.57000000, 15.58000000, 15.59000000, 15.60000000, 15.61000000,
15.62000000, 15.63000000, 15.64000000, 15.65000000, 15.66000000,
15.67000000, 15.68000000, 15.69000000, 15.70000000, 15.71000000,
15.72000000, 15.73000000, 15.74000000, 15.75000000, 15.76000000,
15.77000000, 15.78000000, 15.79000000, 15.80000000, 15.81000000,
15.82000000, 15.83000000, 15.84000000, 15.85000000, 15.86000000,
15.87000000, 15.88000000, 15.89000000, 15.90000000, 15.91000000,
15.92000000, 15.93000000, 15.94000000, 15.95000000, 15.96000000,
15.97000000, 15.98000000, 15.99000000, 16.00000000, 16.01000000,
```