Import Basic Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models  import Sequential
from tensorflow.keras.layers import Dense, LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

```
# connect the google drive with the colab
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
df = pd.read_csv('/content/drive/MyDrive/dataanalytics/a10 .csv')
df
```

```
           date      value
0    1991-07-01   3.526591
1    1991-08-01   3.180891
2    1991-09-01   3.252221
3    1991-10-01   3.611003
4    1991-11-01   3.565869
..          ...        ...
199  2008-02-01  21.654285
200  2008-03-01  18.264945
201  2008-04-01  23.107677
202  2008-05-01  22.912510
203  2008-06-01  19.431740

[204 rows x 2 columns]
```

```
# head() it displays the top rows
df.head()
```

```
         date     value
0  1991-07-01  3.526591
1  1991-08-01  3.180891
2  1991-09-01  3.252221
3  1991-10-01  3.611003
4  1991-11-01  3.565869
```

```
# tail() it displays the bottom  rows
df.tail()
```

```
           date      value
199  2008-02-01  21.654285
```

```
200  2008-03-01  18.264945
201  2008-04-01  23.107677
202  2008-05-01  22.912510
203  2008-06-01  19.431740
```

```python
# it displays numerical data each column
df.describe()
```

```
            value
count  204.000000
mean    10.694430
std      5.956998
min      2.814520
25%      5.844095
50%      9.319345
75%     14.289964
max     29.665356
```

```python
df.dtypes
```

```
date      object
value    float64
dtype: object
```

```python
df['date'] = df['date'].astype('datetime64')
df = df.sort_values('date')
df.set_index('date', inplace=True)
df
```

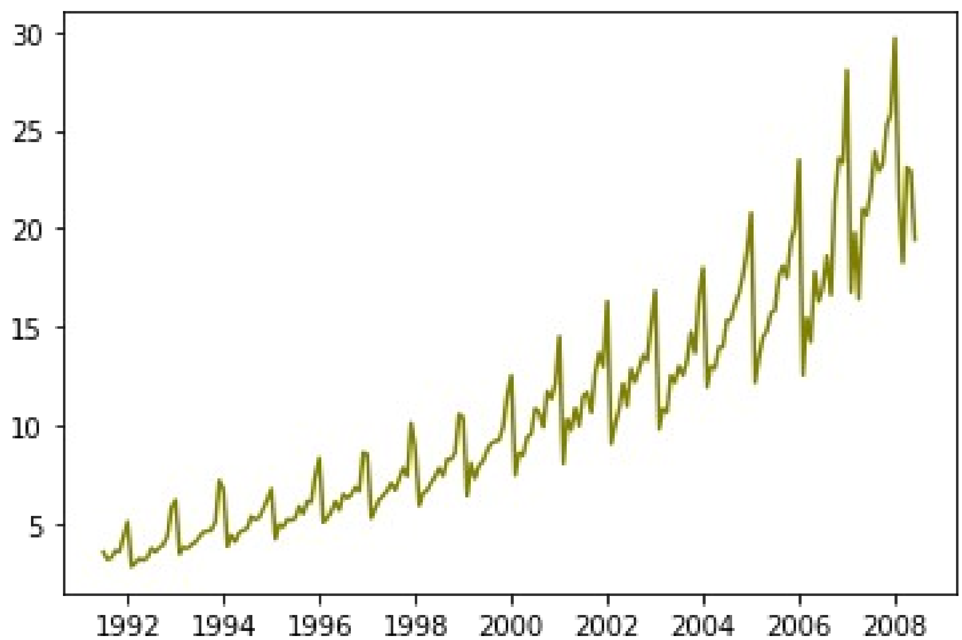
```
                value
date
1991-07-01   3.526591
1991-08-01   3.180891
1991-09-01   3.252221
1991-10-01   3.611003
1991-11-01   3.565869
...               ...
2008-02-01  21.654285
2008-03-01  18.264945
2008-04-01  23.107677
2008-05-01  22.912510
2008-06-01  19.431740

[204 rows x 1 columns]
```

```python
df.shape
```

```
(204, 1)
```

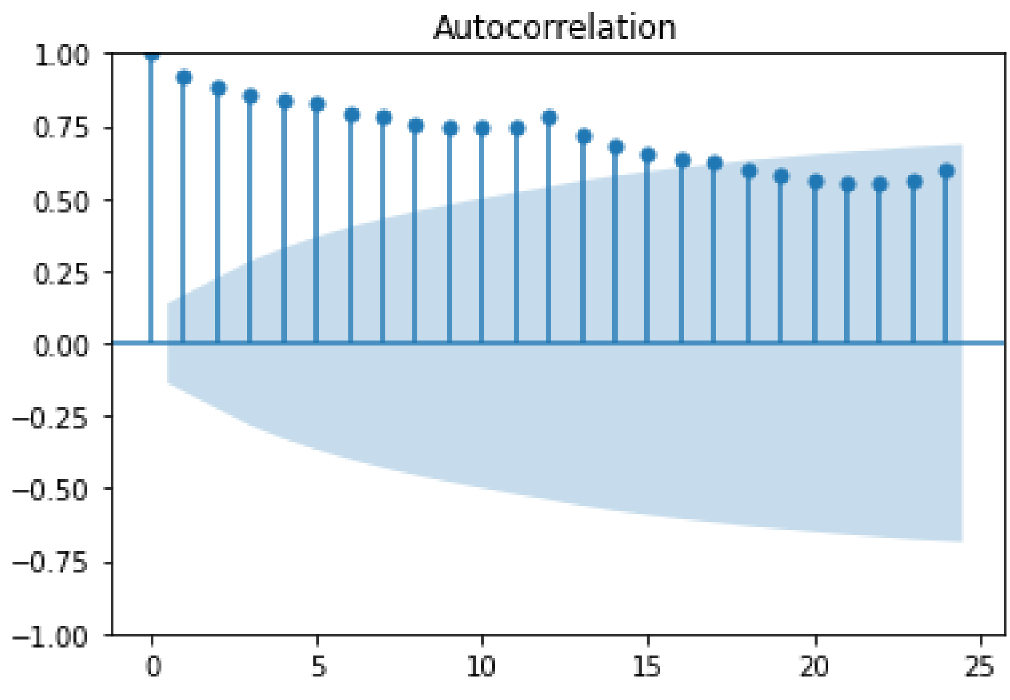
```python
plt.plot(df,color='olive')
```

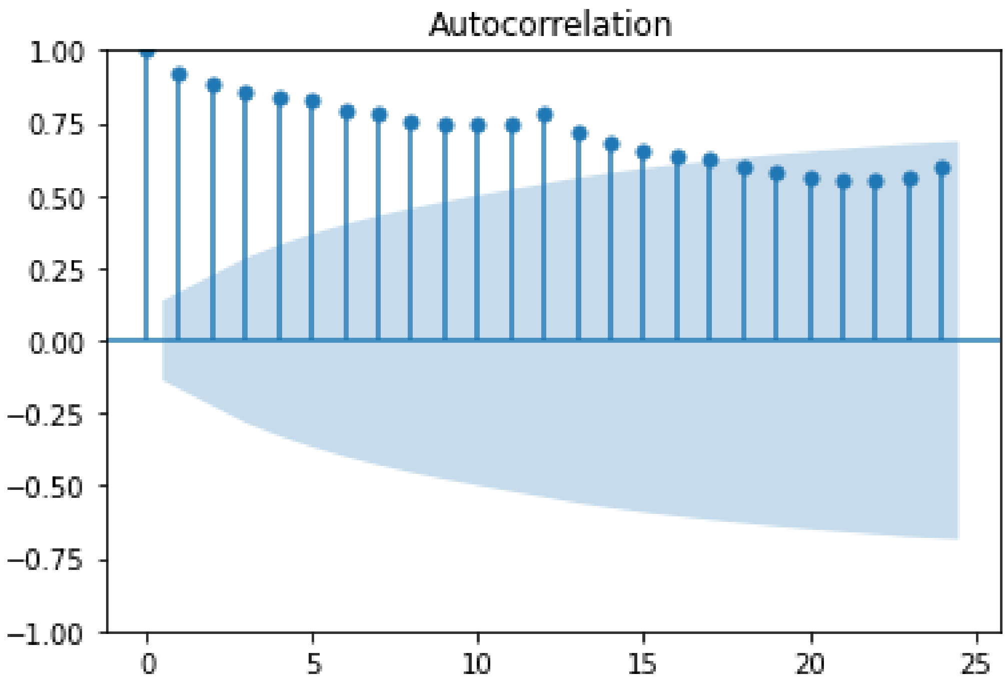
```
[<matplotlib.lines.Line2D at 0x7f79144baf10>]
```



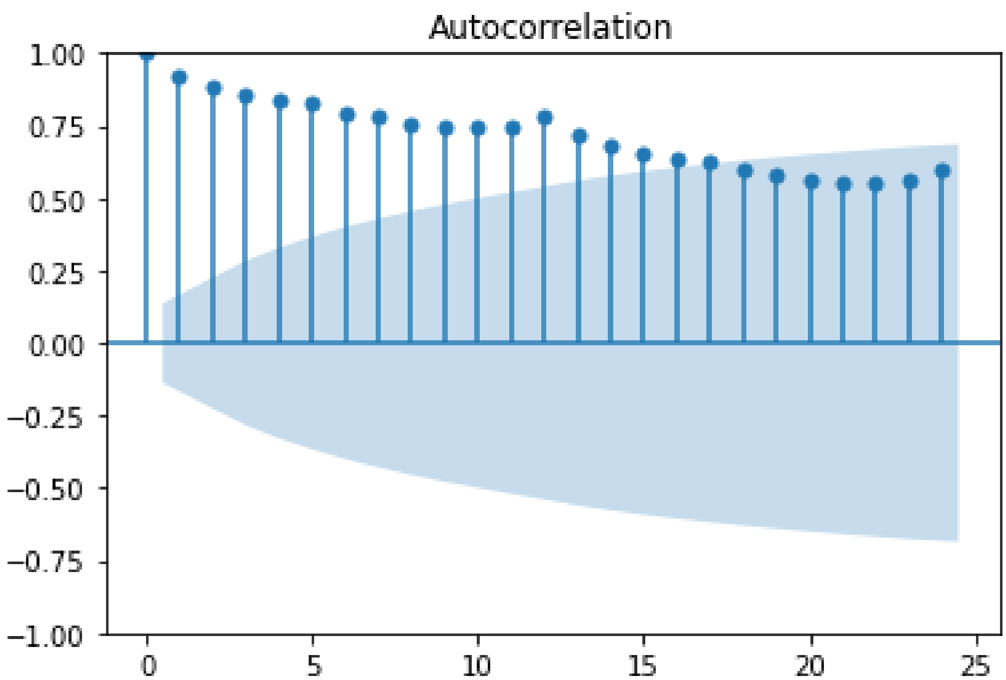

```
from statsmodels.graphics.tsaplots import plot_pacf ,plot_acf
```

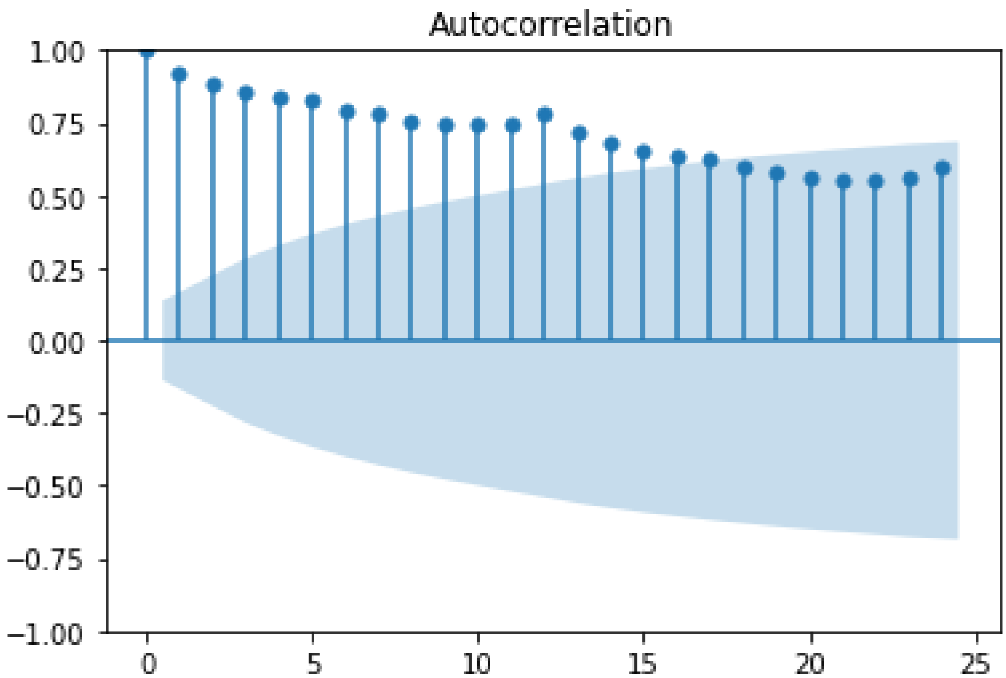
```
plot_acf(df)
```




Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25

```
plot_acf(df)
```


Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25

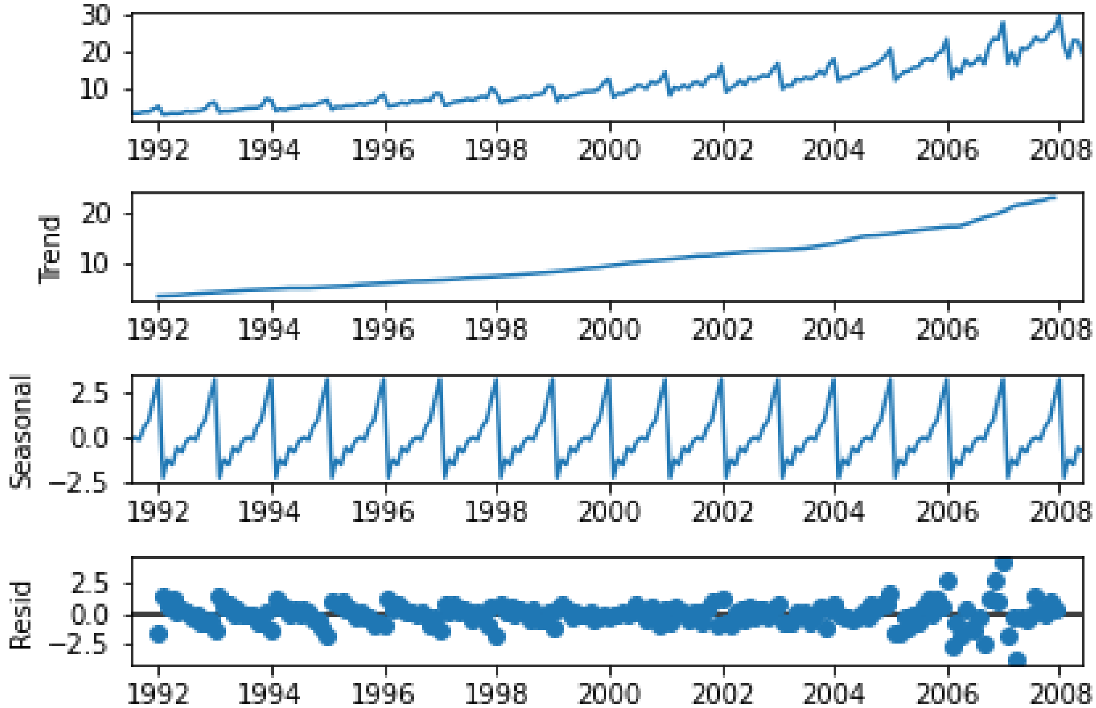


ETS decomposition on the data

```
# For liner data we us additive model

from statsmodels.tsa.seasonal import seasonal_decompose

result = seasonal_decompose(df, model='additive')
res = pd.DataFrame()
res['Trend'] = result.trend
res['Seaonality'] = result.seasonal
res['Residual'] = result.resid
res['Observed'] = result.observed
res['Actual'] = df.iloc[:, 0]
res.head()
```

date	Trend	Seaonality	Residual	Observed	Actual
1991-07-01	NaN	-0.227809	NaN	3.526591	3.526591
1991-08-01	NaN	-0.023116	NaN	3.180891	3.180891
1991-09-01	NaN	-0.149022	NaN	3.252221	3.252221
1991-10-01	NaN	0.569161	NaN	3.611003	3.611003
1991-11-01	NaN	0.966836	NaN	3.565869	3.565869

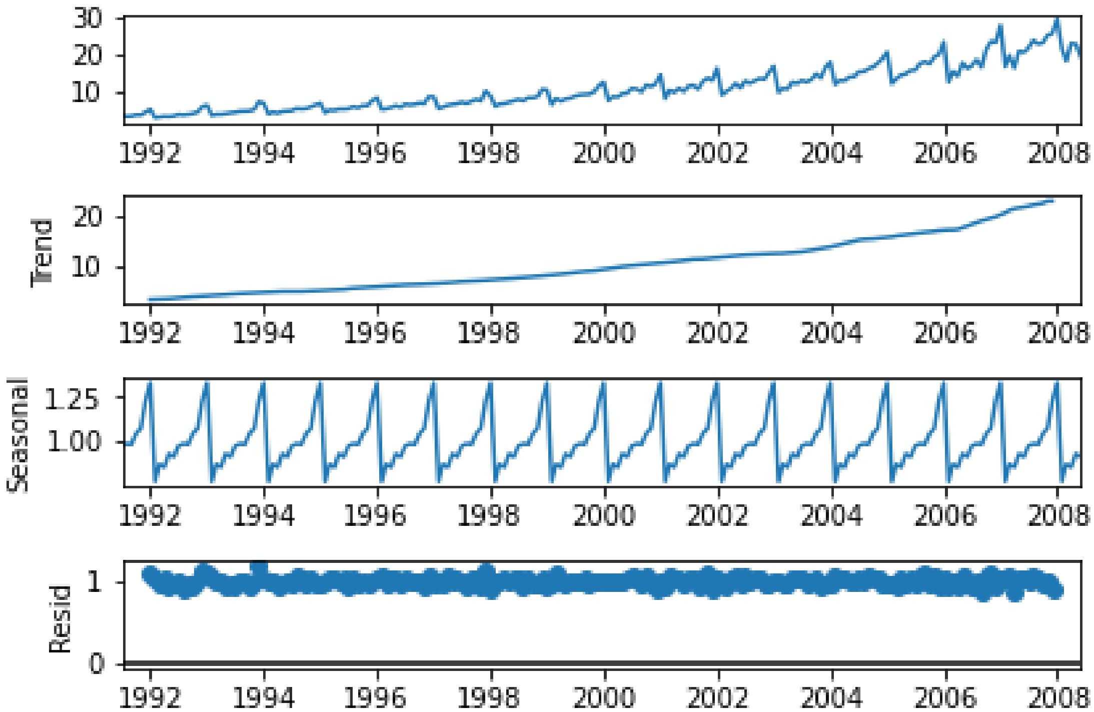
```
import matplotlib.pyplot as plt
result.plot()
plt.show()
```




```
# For Non liner data we us multiplicative model

from statsmodels.tsa.seasonal import seasonal_decompose

result = seasonal_decompose(df, model='multiplicative')
res = pd.DataFrame()
res['Trend'] = result.trend
res['Seaonality'] = result.seasonal
res['Residual'] = result.resid
res['Observed'] = result.observed
res['Actual'] = df.iloc[:, 0]
res.head()
```

date	Trend	Seaonality	Residual	Observed	Actual
1991-07-01	NaN	0.978509	NaN	3.526591	3.526591
1991-08-01	NaN	0.989722	NaN	3.180891	3.180891
1991-09-01	NaN	0.986418	NaN	3.252221	3.252221
1991-10-01	NaN	1.045509	NaN	3.611003	3.611003
1991-11-01	NaN	1.075573	NaN	3.565869	3.565869

```
import matplotlib.pyplot as plt
result.plot()
plt.show()
```



LSTM

```
# convert a dataframe into array
dataset=df.values
dataset[:5]
```

```
array([[3.526591],
       [3.180891],
       [3.252221],
       [3.611003],
       [3.565869]])
```

```
# Normalize the data using MinMaxScaler (0to1)
scaler = MinMaxScaler(feature_range=(0,1))
dataset = scaler.fit_transform(dataset)
dataset[:5]
```

```
array([[0.02651951],
       [0.01364468],
       [0.01630121],
       [0.02966325],
       [0.02798233]])
```

```
# Split data into train and test
train_size = int(len(dataset)*0.8)
train = dataset[:train_size]
test = dataset[train_size:]

# change at data into x,y
def create_dataset(dataset, lag):
```

```
    datax = []
    datay = []
    for i in range(len(dataset)-lag-1):
        a = dataset[i:i+lag, 0]
        datax.append(a)
        datay.append(dataset[i+lag, 0])
    return np.array(datax), np.array(datay)

# set a lag value & create train data and test data
lag = 3
trainx, trainy = create_dataset(train, lag)
testx, testy = create_dataset(test, lag)

# Our data have proper shape to pass in LSTM
trainx = np.reshape(trainx, (trainx.shape[0], 1,
                             trainx.shape[1]))
testx = np.reshape(testx, (testx.shape[0], 1, testx.shape[1]))

trainx.shape
```

```
(159, 1, 3)
```

```
# creat a model
model = Sequential()
model.add(LSTM(4, input_shape=(1, lag)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainx, trainy, epochs=100, batch_size=1, verbose=2)
```

```
Epoch 1/100
159/159 - 2s - loss: 0.0322 - 2s/epoch - 11ms/step
Epoch 2/100
159/159 - 0s - loss: 0.0124 - 209ms/epoch - 1ms/step
Epoch 3/100
159/159 - 0s - loss: 0.0097 - 232ms/epoch - 1ms/step
Epoch 4/100
159/159 - 0s - loss: 0.0074 - 199ms/epoch - 1ms/step
Epoch 5/100
159/159 - 0s - loss: 0.0057 - 203ms/epoch - 1ms/step
Epoch 6/100
159/159 - 0s - loss: 0.0046 - 216ms/epoch - 1ms/step
Epoch 7/100
159/159 - 0s - loss: 0.0040 - 234ms/epoch - 1ms/step
Epoch 8/100
159/159 - 0s - loss: 0.0037 - 303ms/epoch - 2ms/step
Epoch 9/100
159/159 - 0s - loss: 0.0035 - 288ms/epoch - 2ms/step
Epoch 10/100
159/159 - 0s - loss: 0.0034 - 308ms/epoch - 2ms/step
Epoch 11/100
159/159 - 0s - loss: 0.0034 - 289ms/epoch - 2ms/step
```

```
Epoch 12/100
159/159 - 0s - loss: 0.0035 - 290ms/epoch - 2ms/step
Epoch 13/100
159/159 - 0s - loss: 0.0034 - 277ms/epoch - 2ms/step
Epoch 14/100
159/159 - 0s - loss: 0.0033 - 294ms/epoch - 2ms/step
Epoch 15/100
159/159 - 0s - loss: 0.0034 - 294ms/epoch - 2ms/step
Epoch 16/100
159/159 - 0s - loss: 0.0035 - 288ms/epoch - 2ms/step
Epoch 17/100
159/159 - 0s - loss: 0.0034 - 220ms/epoch - 1ms/step
Epoch 18/100
159/159 - 0s - loss: 0.0034 - 234ms/epoch - 1ms/step
Epoch 19/100
159/159 - 0s - loss: 0.0033 - 227ms/epoch - 1ms/step
Epoch 20/100
159/159 - 0s - loss: 0.0033 - 214ms/epoch - 1ms/step
Epoch 21/100
159/159 - 0s - loss: 0.0033 - 200ms/epoch - 1ms/step
Epoch 22/100
159/159 - 0s - loss: 0.0033 - 215ms/epoch - 1ms/step
Epoch 23/100
159/159 - 0s - loss: 0.0033 - 215ms/epoch - 1ms/step
Epoch 24/100
159/159 - 0s - loss: 0.0034 - 223ms/epoch - 1ms/step
Epoch 25/100
159/159 - 0s - loss: 0.0032 - 212ms/epoch - 1ms/step
Epoch 26/100
159/159 - 0s - loss: 0.0033 - 200ms/epoch - 1ms/step
Epoch 27/100
159/159 - 0s - loss: 0.0033 - 205ms/epoch - 1ms/step
Epoch 28/100
159/159 - 0s - loss: 0.0032 - 202ms/epoch - 1ms/step
Epoch 29/100
159/159 - 0s - loss: 0.0032 - 219ms/epoch - 1ms/step
Epoch 30/100
159/159 - 0s - loss: 0.0033 - 207ms/epoch - 1ms/step
Epoch 31/100
159/159 - 0s - loss: 0.0032 - 196ms/epoch - 1ms/step
Epoch 32/100
159/159 - 0s - loss: 0.0033 - 217ms/epoch - 1ms/step
Epoch 33/100
159/159 - 0s - loss: 0.0032 - 210ms/epoch - 1ms/step
Epoch 34/100
159/159 - 0s - loss: 0.0032 - 227ms/epoch - 1ms/step
Epoch 35/100
159/159 - 0s - loss: 0.0032 - 216ms/epoch - 1ms/step
Epoch 36/100
159/159 - 0s - loss: 0.0031 - 209ms/epoch - 1ms/step
```

```
Epoch 37/100
159/159 - 0s - loss: 0.0032 - 198ms/epoch - 1ms/step
Epoch 38/100
159/159 - 0s - loss: 0.0033 - 211ms/epoch - 1ms/step
Epoch 39/100
159/159 - 0s - loss: 0.0032 - 203ms/epoch - 1ms/step
Epoch 40/100
159/159 - 0s - loss: 0.0032 - 197ms/epoch - 1ms/step
Epoch 41/100
159/159 - 0s - loss: 0.0033 - 212ms/epoch - 1ms/step
Epoch 42/100
159/159 - 0s - loss: 0.0032 - 201ms/epoch - 1ms/step
Epoch 43/100
159/159 - 0s - loss: 0.0032 - 209ms/epoch - 1ms/step
Epoch 44/100
159/159 - 0s - loss: 0.0033 - 205ms/epoch - 1ms/step
Epoch 45/100
159/159 - 0s - loss: 0.0032 - 203ms/epoch - 1ms/step
Epoch 46/100
159/159 - 0s - loss: 0.0031 - 211ms/epoch - 1ms/step
Epoch 47/100
159/159 - 0s - loss: 0.0032 - 207ms/epoch - 1ms/step
Epoch 48/100
159/159 - 0s - loss: 0.0032 - 204ms/epoch - 1ms/step
Epoch 49/100
159/159 - 0s - loss: 0.0031 - 209ms/epoch - 1ms/step
Epoch 50/100
159/159 - 0s - loss: 0.0032 - 218ms/epoch - 1ms/step
Epoch 51/100
159/159 - 0s - loss: 0.0032 - 202ms/epoch - 1ms/step
Epoch 52/100
159/159 - 0s - loss: 0.0031 - 222ms/epoch - 1ms/step
Epoch 53/100
159/159 - 0s - loss: 0.0032 - 208ms/epoch - 1ms/step
Epoch 54/100
159/159 - 0s - loss: 0.0032 - 208ms/epoch - 1ms/step
Epoch 55/100
159/159 - 0s - loss: 0.0031 - 213ms/epoch - 1ms/step
Epoch 56/100
159/159 - 0s - loss: 0.0032 - 199ms/epoch - 1ms/step
Epoch 57/100
159/159 - 0s - loss: 0.0032 - 229ms/epoch - 1ms/step
Epoch 58/100
159/159 - 0s - loss: 0.0032 - 226ms/epoch - 1ms/step
Epoch 59/100
159/159 - 0s - loss: 0.0032 - 219ms/epoch - 1ms/step
Epoch 60/100
159/159 - 0s - loss: 0.0031 - 205ms/epoch - 1ms/step
Epoch 61/100
159/159 - 0s - loss: 0.0031 - 210ms/epoch - 1ms/step
```

```
Epoch 62/100
159/159 - 0s - loss: 0.0031 - 225ms/epoch - 1ms/step
Epoch 63/100
159/159 - 0s - loss: 0.0032 - 205ms/epoch - 1ms/step
Epoch 64/100
159/159 - 0s - loss: 0.0031 - 313ms/epoch - 2ms/step
Epoch 65/100
159/159 - 0s - loss: 0.0032 - 293ms/epoch - 2ms/step
Epoch 66/100
159/159 - 0s - loss: 0.0030 - 295ms/epoch - 2ms/step
Epoch 67/100
159/159 - 0s - loss: 0.0032 - 301ms/epoch - 2ms/step
Epoch 68/100
159/159 - 0s - loss: 0.0031 - 287ms/epoch - 2ms/step
Epoch 69/100
159/159 - 0s - loss: 0.0032 - 291ms/epoch - 2ms/step
Epoch 70/100
159/159 - 0s - loss: 0.0031 - 270ms/epoch - 2ms/step
Epoch 71/100
159/159 - 0s - loss: 0.0031 - 278ms/epoch - 2ms/step
Epoch 72/100
159/159 - 0s - loss: 0.0031 - 291ms/epoch - 2ms/step
Epoch 73/100
159/159 - 0s - loss: 0.0031 - 249ms/epoch - 2ms/step
Epoch 74/100
159/159 - 0s - loss: 0.0031 - 213ms/epoch - 1ms/step
Epoch 75/100
159/159 - 0s - loss: 0.0030 - 220ms/epoch - 1ms/step
Epoch 76/100
159/159 - 0s - loss: 0.0031 - 206ms/epoch - 1ms/step
Epoch 77/100
159/159 - 0s - loss: 0.0032 - 217ms/epoch - 1ms/step
Epoch 78/100
159/159 - 0s - loss: 0.0031 - 210ms/epoch - 1ms/step
Epoch 79/100
159/159 - 0s - loss: 0.0031 - 198ms/epoch - 1ms/step
Epoch 80/100
159/159 - 0s - loss: 0.0031 - 208ms/epoch - 1ms/step
Epoch 81/100
159/159 - 0s - loss: 0.0030 - 207ms/epoch - 1ms/step
Epoch 82/100
159/159 - 0s - loss: 0.0031 - 209ms/epoch - 1ms/step
Epoch 83/100
159/159 - 0s - loss: 0.0031 - 222ms/epoch - 1ms/step
Epoch 84/100
159/159 - 0s - loss: 0.0032 - 204ms/epoch - 1ms/step
Epoch 85/100
159/159 - 0s - loss: 0.0031 - 200ms/epoch - 1ms/step
Epoch 86/100
159/159 - 0s - loss: 0.0031 - 197ms/epoch - 1ms/step
```

```
Epoch 87/100
159/159 - 0s - loss: 0.0030 - 216ms/epoch - 1ms/step
Epoch 88/100
159/159 - 0s - loss: 0.0031 - 219ms/epoch - 1ms/step
Epoch 89/100
159/159 - 0s - loss: 0.0031 - 205ms/epoch - 1ms/step
Epoch 90/100
159/159 - 0s - loss: 0.0032 - 213ms/epoch - 1ms/step
Epoch 91/100
159/159 - 0s - loss: 0.0032 - 216ms/epoch - 1ms/step
Epoch 92/100
159/159 - 0s - loss: 0.0031 - 217ms/epoch - 1ms/step
Epoch 93/100
159/159 - 0s - loss: 0.0031 - 203ms/epoch - 1ms/step
Epoch 94/100
159/159 - 0s - loss: 0.0031 - 215ms/epoch - 1ms/step
Epoch 95/100
159/159 - 0s - loss: 0.0031 - 217ms/epoch - 1ms/step
Epoch 96/100
159/159 - 0s - loss: 0.0031 - 206ms/epoch - 1ms/step
Epoch 97/100
159/159 - 0s - loss: 0.0031 - 231ms/epoch - 1ms/step
Epoch 98/100
159/159 - 0s - loss: 0.0031 - 206ms/epoch - 1ms/step
Epoch 99/100
159/159 - 0s - loss: 0.0031 - 201ms/epoch - 1ms/step
Epoch 100/100
159/159 - 0s - loss: 0.0031 - 226ms/epoch - 1ms/step

<keras.callbacks.History at 0x7f790f7e7070>
```

```
# make a prediction on train and test data
trainpredict = model.predict(trainx)
testpredict = model.predict(testx)
```

```
5/5 [==============================] - 1s 2ms/step
2/2 [==============================] - 0s 5ms/step
```

```
# Now we get predicted value in sacled form we want in actual value so
we use inverse_transform
trainpredict = scaler.inverse_transform(trainpredict)
trainy = scaler.inverse_transform([trainy])
testpredict = scaler.inverse_transform(testpredict)
testy = scaler.inverse_transform([testy])
```

```
mean_squared_error(trainy[0], trainpredict[:,0])
```

```
2.190742639424695
```

```
mean_squared_error(testy[0], testpredict[:,0])
```

```
11.535367674320788
```

```python
trainpredictplot=np.empty_like(dataset)
trainpredictplot[:,:] = np.nan
trainpredictplot[lag:len(trainpredict)+lag,:] = trainpredict

testpredictplot=np.empty_like(dataset)
testpredictplot[:,:] = np.nan
testpredictplot[len(trainpredict) + 2*lag+1:len(dataset)-1, :] =
testpredict

dataset = scaler.inverse_transform(dataset)

#  Visualize Between predicted and actual value
plt.plot(dataset,color='gray', label="Actual data")
plt.plot(trainpredictplot ,color='purple',label="train_predict value")
plt.plot(testpredictplot ,color='cyan', label="test_predict value")
plt.legend()
plt.show()
```