```
import pandas as pd
```

```
path = "/content/drive/MyDrive/NFLX.csv"
```

```
df = pd.read_csv(path)
```

```
df
```

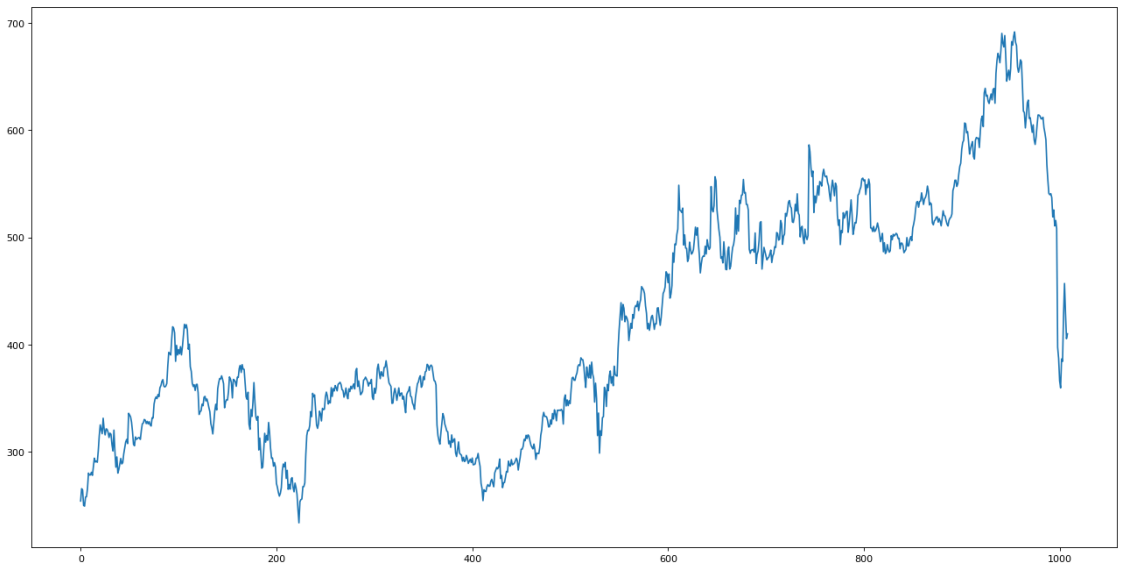
	Date	Open	High	Low	Close	Adj Close	Volume
0	2018-02-05	262.000000	267.899994	250.029999	254.259995	254.259995	11896100
1	2018-02-06	247.699997	266.700012	245.000000	265.720001	265.720001	12595800
2	2018-02-07	266.579987	272.450012	264.329987	264.559998	264.559998	8981500
3	2018-02-08	267.079987	267.619995	250.000000	250.100006	250.100006	9306700
4	2018-02-09	253.850006	255.800003	236.110001	249.470001	249.470001	16906900
...	...	...	...	...	...	...	...
1004	2022-01-31	401.970001	427.700012	398.200012	427.140015	427.140015	20047500
1005	2022-02-01	432.959991	458.480011	425.540009	457.130005	457.130005	22542300
1006	2022-02-02	448.250000	451.980011	426.480011	429.480011	429.480011	14346000
1007	2022-02-03	421.440002	429.260010	404.279999	405.600006	405.600006	9905200
1008	2022-02-04	407.309998	412.769989	396.640015	410.170013	410.170013	7782400

1009 rows × 7 columns

```
df = df[df['Close'] != 0]   #removing all zero values
```

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import math
```

```
figure(figsize=(20,10),dpi=80)
time_x = df.index
time_y = df['Close']
plt.plot(time_x,time_y)
plt.show()
```




```
! pip install EMD-signal
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting EMD-signal
  Downloading EMD_signal-1.4.0-py3-none-any.whl (77 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 77.4/77.4 kB 2.5 MB/s eta 0:00:00
Requirement already satisfied: scipy>=0.19 in /usr/local/lib/python3.9/dist-packages (from EMD-signal) (1.10.1)
Collecting pathos>=0.2.1
  Downloading pathos-0.3.0-py3-none-any.whl (79 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 79.8/79.8 kB 3.9 MB/s eta 0:00:00
Collecting tqdm==4.64.*
  Downloading tqdm-4.64.1-py2.py3-none-any.whl (78 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 78.5/78.5 kB 5.4 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.12 in /usr/local/lib/python3.9/dist-packages (from EMD-signal) (1.22.4)
Collecting pox>=0.3.2
  Downloading pox-0.3.2-py3-none-any.whl (29 kB)
Collecting dill>=0.3.6
  Downloading dill-0.3.6-py3-none-any.whl (110 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 110.5/110.5 kB 6.6 MB/s eta 0:00:00
Collecting ppft>=1.7.6.6
  Downloading ppft-1.7.6.6-py3-none-any.whl (52 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 52.8/52.8 kB 4.4 MB/s eta 0:00:00
Collecting multiprocess>=0.70.14
  Downloading multiprocess-0.70.14-py39-none-any.whl (132 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 132.9/132.9 kB 7.1 MB/s eta 0:00:00
Installing collected packages: tqdm, ppft, pox, dill, multiprocess, pathos, EMD-signal
  Attempting uninstall: tqdm
    Found existing installation: tqdm 4.65.0
    Uninstalling tqdm-4.65.0:
      Successfully uninstalled tqdm-4.65.0
Successfully installed EMD-signal-1.4.0 dill-0.3.6 multiprocess-0.70.14 pathos-0.3.0 pox-0.3.2 ppft-1.7.6.6 tqdm-4.64.1
```

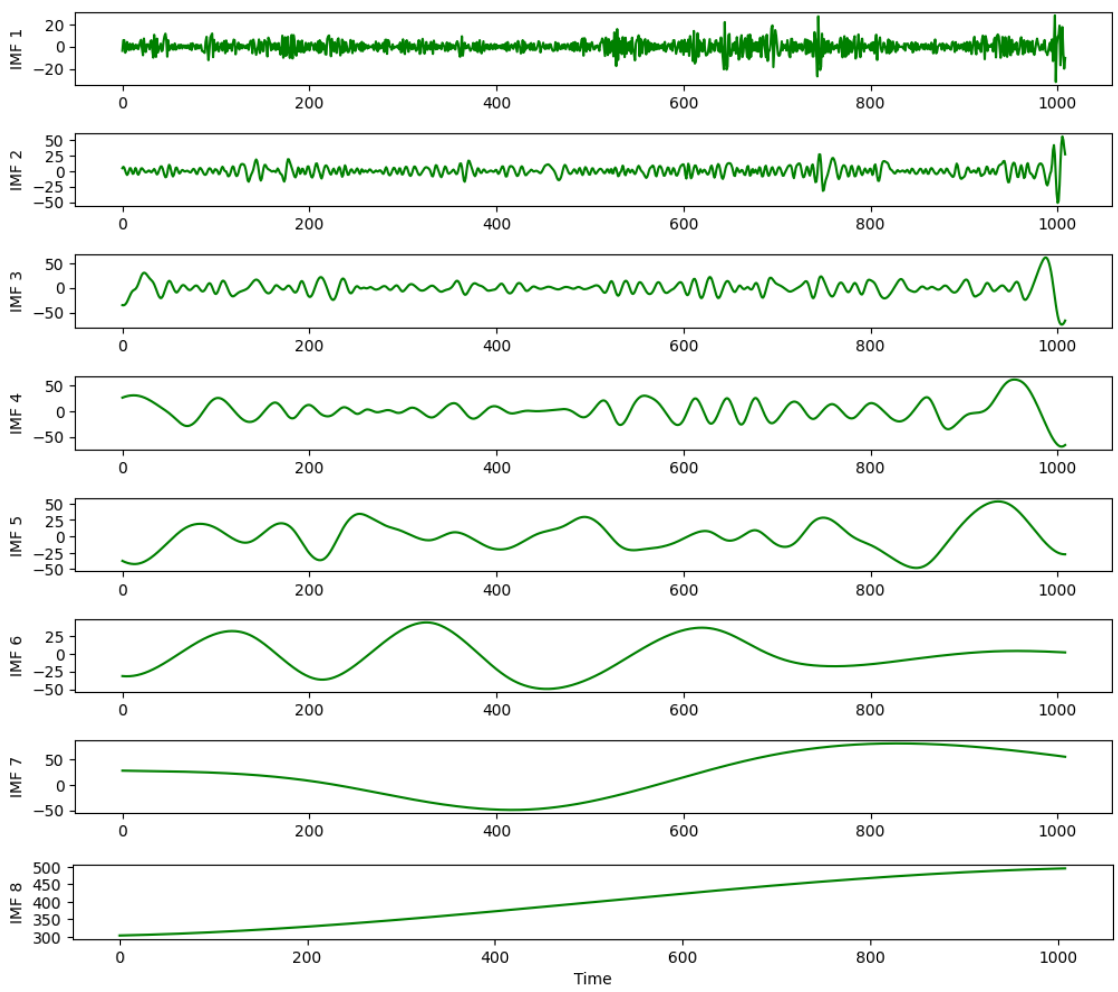
```
Signal = df['Close'].to_numpy()
T = df.index.to_numpy()
from PyEMD import EMD
emd = EMD()
IMFs = emd(Signal)
nIMFs = len(IMFs)
```

```
plt.figure(figsize=(12,9))
plt.subplot(nIMFs+10, 1, 1)
plt.plot(T, Signal, 'r')
```

```
[<matplotlib.lines.Line2D at 0x7f3f72409b50>]
```


```
for n in range(nIMFs):
  plt.figure(figsize=(12,9))
  plt.subplot(nIMFs+1, 1,2)
  plt.plot(T, IMFs[n], 'g')
  plt.ylabel("IMF %i" %(n+1))
  plt.locator_params(axis='y', nbins=5)


plt.xlabel("Time")
plt.show()
```




```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

```
def create_dataset(dataset, look_back):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return np.array(dataX), np.array(dataY)
```

```
def perform_LSTM(dataset, look_back, layer=4):

  dataset = dataset.astype('float32')
  dataset = np.reshape(dataset, (-1, 1))

  # Normalize the data -- using Min and Max values in each subsequence to normalize the values
  scaler = MinMaxScaler()
  dataset = scaler.fit_transform(dataset)

  # Split data into training and testing set
  train_size = int(len(dataset) * 0.9)
  test_size = len(dataset) - train_size
  train, test = dataset[0:train_size, :], dataset[train_size:, :]

  trainX, trainY = create_dataset(train, look_back)
  testX, testY = create_dataset(test, look_back)

  trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
  testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

  # create and fit the LSTM network
```

```
  model = Sequential()
  model.add(LSTM(layer, input_shape=(1, look_back)))
  model.add(Dense(1))
  model.compile(loss='mean_squared_error', optimizer='adam')
  model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)

  # make predictions
  trainPredict = model.predict(trainX)
  testPredict = model.predict(testX)

  # invert predictions
  trainPredict = scaler.inverse_transform(trainPredict)
  trainY = scaler.inverse_transform([trainY])
  testPredict = scaler.inverse_transform(testPredict)
  testY = scaler.inverse_transform([testY])
  testing_error = np.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
  return testPredict, testY, testing_error
```

```
IMF_predict_list = []
error_list = []
for IMF in IMFs:
  IMF_predict, IMF_test, testing_error = perform_LSTM(IMF, 10, layer=4)
  error_list.append(testing_error)
  IMF_predict_list.append(IMF_predict)
```

```
Epoch 73/100
897/897 - 2s - loss: 5.7449e-06 - 2s/epoch - 2ms/step
Epoch 74/100
897/897 - 3s - loss: 5.8542e-06 - 3s/epoch - 3ms/step
Epoch 75/100
897/897 - 2s - loss: 6.7704e-06 - 2s/epoch - 2ms/step
Epoch 76/100
897/897 - 2s - loss: 6.1915e-06 - 2s/epoch - 2ms/step
Epoch 77/100
897/897 - 2s - loss: 8.8543e-06 - 2s/epoch - 2ms/step
Epoch 78/100
897/897 - 2s - loss: 6.2455e-06 - 2s/epoch - 2ms/step
Epoch 79/100
897/897 - 2s - loss: 7.5746e-06 - 2s/epoch - 2ms/step
Epoch 80/100
897/897 - 2s - loss: 7.4911e-06 - 2s/epoch - 2ms/step
Epoch 81/100
897/897 - 2s - loss: 7.3224e-06 - 2s/epoch - 3ms/step
Epoch 82/100
897/897 - 2s - loss: 7.4952e-06 - 2s/epoch - 3ms/step
Epoch 83/100
897/897 - 2s - loss: 5.9705e-06 - 2s/epoch - 2ms/step
Epoch 84/100
897/897 - 2s - loss: 1.3746e-05 - 2s/epoch - 2ms/step
Epoch 85/100
897/897 - 2s - loss: 4.9681e-06 - 2s/epoch - 2ms/step
Epoch 86/100
897/897 - 2s - loss: 1.0908e-05 - 2s/epoch - 2ms/step
Epoch 87/100
897/897 - 2s - loss: 9.7880e-06 - 2s/epoch - 2ms/step
Epoch 88/100
897/897 - 2s - loss: 5.5792e-06 - 2s/epoch - 3ms/step
Epoch 89/100
897/897 - 3s - loss: 5.9313e-06 - 3s/epoch - 3ms/step
Epoch 90/100
897/897 - 2s - loss: 8.3074e-06 - 2s/epoch - 2ms/step
Epoch 91/100
897/897 - 2s - loss: 4.5972e-06 - 2s/epoch - 2ms/step
Epoch 92/100
897/897 - 2s - loss: 5.3493e-06 - 2s/epoch - 2ms/step
Epoch 93/100
897/897 - 2s - loss: 6.8116e-06 - 2s/epoch - 2ms/step
Epoch 94/100
897/897 - 2s - loss: 7.1901e-06 - 2s/epoch - 2ms/step
Epoch 95/100
897/897 - 2s - loss: 6.8687e-06 - 2s/epoch - 2ms/step
Epoch 96/100
897/897 - 3s - loss: 6.3345e-06 - 3s/epoch - 3ms/step
Epoch 97/100
897/897 - 2s - loss: 4.8727e-06 - 2s/epoch - 2ms/step
Epoch 98/100
897/897 - 2s - loss: 7.0100e-06 - 2s/epoch - 2ms/step
Epoch 99/100
897/897 - 2s - loss: 6.4929e-06 - 2s/epoch - 2ms/step
Epoch 100/100
897/897 - 2s - loss: 1.0533e-05 - 2s/epoch - 2ms/step
29/29 [==============================] - 1s 2ms/step
3/3 [==============================] - 0s 5ms/step
```

```
final_prediction = []
for i in range(len(IMF_predict_list[0])):
  element = 0
  for j in range(len(IMF_predict_list)):
    element += IMF_predict_list[j][i]
  final_prediction = final_prediction + element.tolist()
```

```
SP = time_y.astype('float32')
SP = np.reshape(SP.to_numpy(), (-1, 1))
```

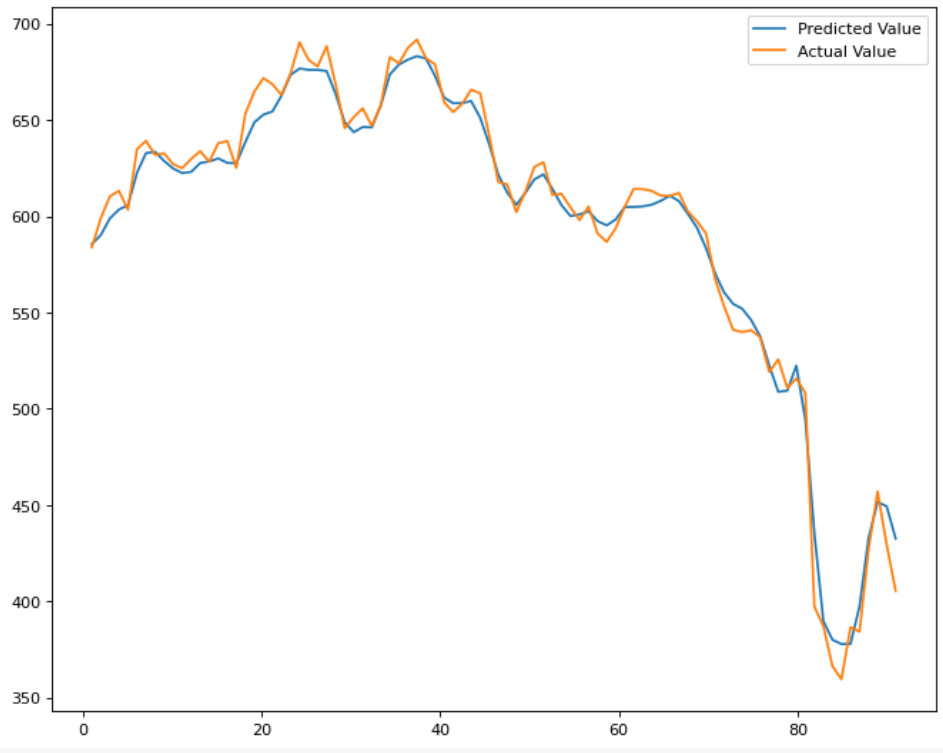
```
train_size = int(len(SP) * 0.9)
test_size = len(SP) - train_size
SP_train, SP_test = SP[0:train_size], SP[train_size:]
```

```
SP_testX, SP_testY = create_dataset(SP_test, 10)
```

```
math.sqrt(mean_squared_error(SP_testY.tolist(), final_prediction))
```

```
9.37214873115156
```

```
figure(figsize=(10, 8), dpi=80)
x = np.linspace(1, len(final_prediction)+1, len(final_prediction), endpoint=True)
# plot lines
plt.plot(x, final_prediction, label = "Predicted Value")
plt.plot(x, SP_testY.tolist(), label = "Actual Value")
plt.legend()
plt.show()
```



Colab paid products - Cancel contracts here