microsoft-stock-data

August 11, 2023

Microsoft Stock Data

[ ]:
```
#import Library

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import time
```

[ ]:
```
#Load DataSet
from google.colab import files
upload=files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving MSFT.csv to MSFT.csv
```

[ ]:
```
df=pd.read_csv('MSFT.csv')
```

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

[ ]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1986-03-13	0.088542	0.101563	0.088542	0.097222	0.061434	1031788800
1	1986-03-14	0.097222	0.102431	0.097222	0.100694	0.063628	308160000
2	1986-03-17	0.100694	0.103299	0.100694	0.102431	0.064725	133171200
3	1986-03-18	0.102431	0.103299	0.098958	0.099826	0.063079	67766400
4	1986-03-19	0.099826	0.100694	0.097222	0.098090	0.061982	47894400

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9083 entries, 0 to 9082
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       9083 non-null   object
 1   Open       9083 non-null   float64
```

```
 2   High       9083 non-null   float64
 3   Low        9083 non-null   float64
 4   Close      9083 non-null   float64
 5   Adj Close  9083 non-null   float64
 6   Volume     9083 non-null   int64
dtypes: float64(5), int64(1), object(1)
memory usage: 496.9+ KB
```

[ ]:
```
df.set_index('Date',drop=True,inplace=True)
```

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

[ ]:
```
                Open      High       Low     Close  Adj Close      Volume
Date
1986-03-13  0.088542  0.101563  0.088542  0.097222   0.061434  1031788800
1986-03-14  0.097222  0.102431  0.097222  0.100694   0.063628   308160000
1986-03-17  0.100694  0.103299  0.100694  0.102431   0.064725   133171200
1986-03-18  0.102431  0.103299  0.098958  0.099826   0.063079    67766400
1986-03-19  0.099826  0.100694  0.097222  0.098090   0.061982    47894400
```

We'll use Only Close Features

[ ]:
```
df=df[['Close']]
```

[ ]:
```
df
```

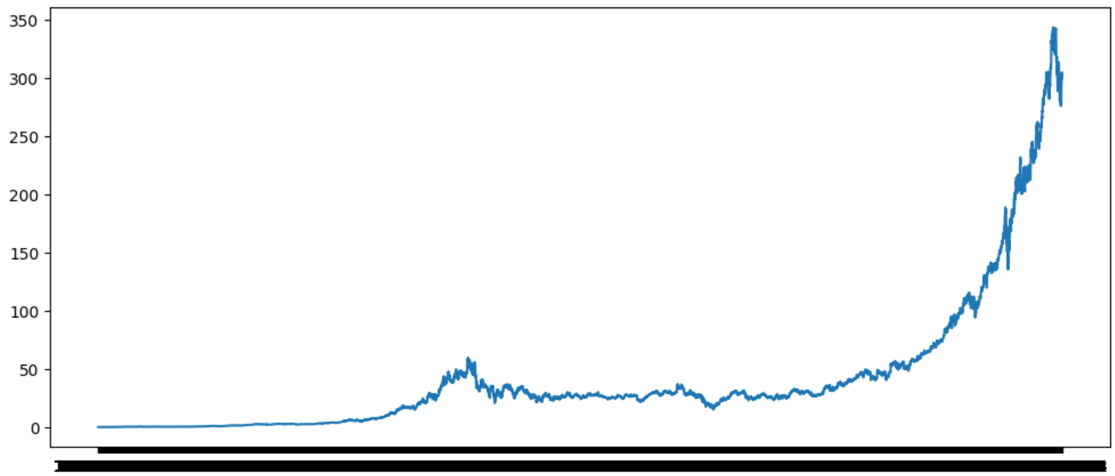
[ ]:
```
                 Close
Date
1986-03-13    0.097222
1986-03-14    0.100694
1986-03-17    0.102431
1986-03-18    0.099826
1986-03-19    0.098090
…                  …
2022-03-18  300.429993
2022-03-21  299.160004
2022-03-22  304.059998
2022-03-23  299.489990
2022-03-24  304.100006

[9083 rows x 1 columns]
```

[ ]:
```
df.describe()
```

[ ]:
```
             Close
count  9083.000000
mean     41.335628
std      59.714567
```

```
min        0.090278
25%        4.075195
50%       26.840000
75%       39.937500
max      343.109985
```

[ ]:
```
plt.figure (1,figsize=(12,5))
plt.plot(df.Close);
```



Calculate the percentage Change **The Reason for using pct_change Instead of the prices is the beniefit of normalization as we can measure all variables a comparable metric.Also returns have more manageable statistics properties than prices such as stationarity,as in most cases we don't statiobnary prices but we can have stationary returns.

A Stationary time series is one where statistical properties such as mean ,varience ,correlation ,etc are constant over Time**

[ ]:
```
df['returns']=df.Close.pct_change()
```

[ ]:
```
134.75-132.89-1
```

[ ]: 0.8600000000000136

Calculate the log returns

***Several benefits of using log returns, both theoretic and algorithmic.

First, log-normality: if we assume that prices are distributed log normally (which, in practice, may or may not be true for any given price series), then log(1 + r_i) is conveniently normally distributed, because:

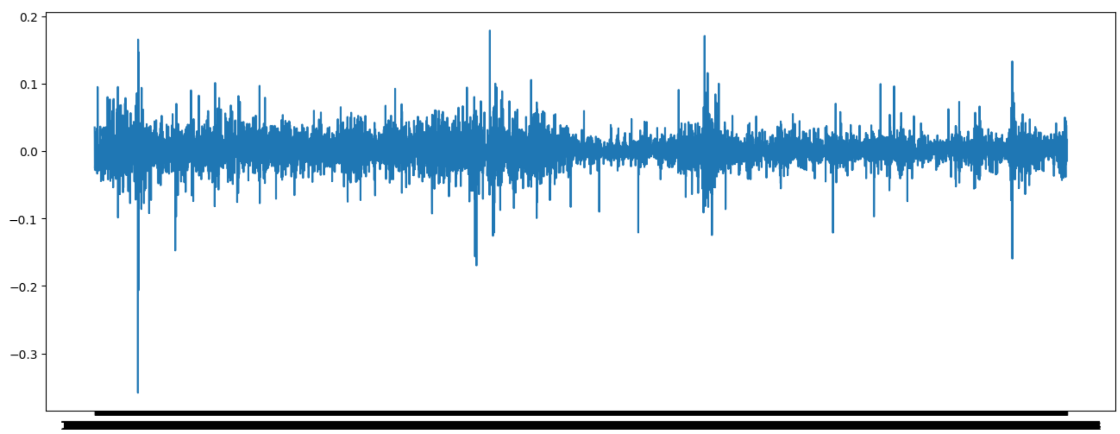
`1 + r_i = \frac{p_i}{p_j} = \exp^{\log(\frac{p_i}{p_j})}`

This is handy given much of classic statistics presumes normality***

[ ]:
```
df['log_returns']=np.log(1+df['returns'])
```

[ ]:
```
plt.figure(1,figsize=(16,6))
plt.plot(df.log_returns)
```

[ ]: [<matplotlib.lines.Line2D at 0x7bd5d531b220>]



[36]:
```
#Drop NULL
df.dropna(inplace=True)
x=df[['Close','log_returns']].values
```

[37]:
```
x
```

[37]:
```
array([[ 1.00694000e-01,  3.50891917e-02],
       [ 1.02431000e-01,  1.71031861e-02],
       [ 9.98260000e-02, -2.57607307e-02],
       …,
       [ 3.04059998e+02,  1.62464831e-02],
       [ 2.99489990e+02, -1.51440491e-02],
       [ 3.04100006e+02,  1.52756198e-02]])
```

[52]:
```
#Scaling
from sklearn.preprocessing import StandardScaler
```

[54]:
```
scaler=StandardScaler()
x_scaler=scaler.fit_transform(x)
```

[55]:
```
x_scaler[:5]
```

[55]:
```
array([[-0.69062809,  1.59840864],
       [-0.690599  ,  0.75786948],
       [-0.69064263, -1.24528795],
```

```
        [-0.6906717 , -0.86126061],
        [-0.69071531, -1.2987997 ]])
```

[95]:
```
y=[x[0] for x in x_scaler ]
```

[101]:
```
y[:5]
```

[101]:
```
[-0.6906280902266285,
 -0.6905990010829361,
 -0.6906426264250884,
 -0.6906716988220084,
 -0.6907153074173882]
```

[58]:
```
#Train Test Split
```

[97]:
```
split=int(len(x_scaler)*0.8)
print(split)
```

```
7265
```

[98]:
```
x_train=x_scaler[:split]
x_test=x_scaler[split: len(x_scaler)]
y_train=y[:split]
y_test=y[split: len(y)]
```

[100]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_scaler,y,train_size=0.
↪75,random_state=0)
```

[102]:
```
len(x_train)==len(y_train)
len(x_test)==len(y_test)
```

[102]:
```
True
```

Labeling *we want to pridict the stock price at a future time .

as we going to use LSTM*

[161]:
```
n=3
xtrain=[]
ytrain=[]
xtest=[]
ytest=[]
for i in range(n,len(x_train)):
    xtrain.append(x_train[i-n:i,:x_train.shape[1]]) #pridiction next record
    ytrain.append(y_train[i])
for i in range(n,len(x_test)):
    ytest.append(y_test[i])
```

```
    xtest.append(x_test[i-n:i,:x_test.shape[1]]) #pridiction next record
```

[162]:
```
df.head()
```

[162]:
```
                Close   returns  log_returns
Date
1986-03-14  0.100694  0.035712     0.035089
1986-03-17  0.102431  0.017250     0.017103
1986-03-18  0.099826 -0.025432    -0.025761
1986-03-19  0.098090 -0.017390    -0.017543
1986-03-20  0.095486 -0.026547    -0.026906
```

[163]:
```
xtrain[0]
```

[163]:
```
array([[-0.21972049,  0.13279207],
       [-0.49710982,  0.74827861],
       [ 1.62074973,  0.58533444]])
```

[164]:
```
ytrain[0]
```

[164]:
```
-0.28310702240970176
```

[165]:
```
val=np.array(ytrain[0])
val=np.c_[val,np.zeros(val.shape)]
```

[166]:
```
scaler.inverse_transform(val)
```

[166]:
```
array([[2.44349990e+01, 8.86161074e-04]])
```

[167]:
```
xtrain,ytrain=[np.array(xtrain),np.array(ytrain)]
```

[168]:
```
xtrain=np.reshape(xtrain ,(xtrain.shape[0],xtrain.shape[1],xtrain.shape[2]))
xtest=np.array(xtest)
ytest=np.array(ytest)
xtest=np.reshape(xtest ,(xtest.shape[0],xtest.shape[1],xtest.shape[2]))
```

[169]:
```
print(xtrain.shape)
print(ytrain.shape)
print("************************")
print(xtest.shape)
print(ytest.shape)
```

```
(6808, 3, 2)
(6808,)
************************
(2268, 3, 2)
(2268,)
```

[170]:
```
##LSTM Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM,Dense
```

[171]:
```
model=Sequential()
```

[172]:
```
model.add(LSTM(4,input_shape=(xtrain.shape[1],xtrain.shape[2])))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
```

[173]:
```
model.
 ↪fit(xtrain,ytrain,validation_data=(xtest,ytest),epochs=50,batch_size=16,verbose=1)
```

```
Epoch 1/50
426/426 [==============================] - 5s 5ms/step - loss: 1.0259 -
val_loss: 0.9458
Epoch 2/50
426/426 [==============================] - 2s 4ms/step - loss: 1.0205 -
val_loss: 0.9433
Epoch 3/50
426/426 [==============================] - 2s 5ms/step - loss: 1.0196 -
val_loss: 0.9428
Epoch 4/50
426/426 [==============================] - 2s 4ms/step - loss: 1.0195 -
val_loss: 0.9430
Epoch 5/50
426/426 [==============================] - 2s 5ms/step - loss: 1.0194 -
val_loss: 0.9424
Epoch 6/50
426/426 [==============================] - 2s 5ms/step - loss: 1.0191 -
val_loss: 0.9433
Epoch 7/50
426/426 [==============================] - 2s 4ms/step - loss: 1.0195 -
val_loss: 0.9424
Epoch 8/50
426/426 [==============================] - 2s 4ms/step - loss: 1.0191 -
val_loss: 0.9421
Epoch 9/50
426/426 [==============================] - 2s 4ms/step - loss: 1.0192 -
val_loss: 0.9420
Epoch 10/50
426/426 [==============================] - 2s 4ms/step - loss: 1.0191 -
val_loss: 0.9420
Epoch 11/50
426/426 [==============================] - 2s 4ms/step - loss: 1.0189 -
val_loss: 0.9420
Epoch 12/50
```

```
426/426 [==============================] - 2s 5ms/step - loss: 1.0189 -
val_loss: 0.9419
Epoch 13/50
426/426 [==============================] - 2s 5ms/step - loss: 1.0190 -
val_loss: 0.9417
Epoch 14/50
426/426 [==============================] - 2s 4ms/step - loss: 1.0188 -
val_loss: 0.9423
Epoch 15/50
426/426 [==============================] - 2s 4ms/step - loss: 1.0189 -
val_loss: 0.9418
Epoch 16/50
426/426 [==============================] - 2s 4ms/step - loss: 1.0189 -
val_loss: 0.9415
Epoch 17/50
426/426 [==============================] - 3s 6ms/step - loss: 1.0188 -
val_loss: 0.9420
Epoch 18/50
426/426 [==============================] - 4s 9ms/step - loss: 1.0189 -
val_loss: 0.9417
Epoch 19/50
426/426 [==============================] - 3s 6ms/step - loss: 1.0187 -
val_loss: 0.9419
Epoch 20/50
426/426 [==============================] - 3s 6ms/step - loss: 1.0187 -
val_loss: 0.9414
Epoch 21/50
426/426 [==============================] - 3s 6ms/step - loss: 1.0188 -
val_loss: 0.9413
Epoch 22/50
426/426 [==============================] - 3s 7ms/step - loss: 1.0187 -
val_loss: 0.9413
Epoch 23/50
426/426 [==============================] - 3s 7ms/step - loss: 1.0186 -
val_loss: 0.9414
Epoch 24/50
426/426 [==============================] - 2s 5ms/step - loss: 1.0184 -
val_loss: 0.9411
Epoch 25/50
426/426 [==============================] - 2s 6ms/step - loss: 1.0184 -
val_loss: 0.9416
Epoch 26/50
426/426 [==============================] - 2s 6ms/step - loss: 1.0185 -
val_loss: 0.9412
Epoch 27/50
426/426 [==============================] - 3s 7ms/step - loss: 1.0182 -
val_loss: 0.9411
Epoch 28/50
```

```
426/426 [==============================] - 3s 7ms/step - loss: 1.0182 -
val_loss: 0.9409
Epoch 29/50
426/426 [==============================] - 3s 6ms/step - loss: 1.0183 -
val_loss: 0.9409
Epoch 30/50
426/426 [==============================] - 3s 6ms/step - loss: 1.0183 -
val_loss: 0.9407
Epoch 31/50
426/426 [==============================] - 3s 6ms/step - loss: 1.0181 -
val_loss: 0.9417
Epoch 32/50
426/426 [==============================] - 4s 9ms/step - loss: 1.0182 -
val_loss: 0.9407
Epoch 33/50
426/426 [==============================] - 3s 6ms/step - loss: 1.0182 -
val_loss: 0.9408
Epoch 34/50
426/426 [==============================] - 3s 6ms/step - loss: 1.0180 -
val_loss: 0.9410
Epoch 35/50
426/426 [==============================] - 3s 6ms/step - loss: 1.0181 -
val_loss: 0.9407
Epoch 36/50
426/426 [==============================] - 3s 8ms/step - loss: 1.0180 -
val_loss: 0.9408
Epoch 37/50
426/426 [==============================] - 3s 8ms/step - loss: 1.0180 -
val_loss: 0.9407
Epoch 38/50
426/426 [==============================] - 3s 6ms/step - loss: 1.0179 -
val_loss: 0.9413
Epoch 39/50
426/426 [==============================] - 2s 6ms/step - loss: 1.0177 -
val_loss: 0.9415
Epoch 40/50
426/426 [==============================] - 2s 6ms/step - loss: 1.0178 -
val_loss: 0.9407
Epoch 41/50
426/426 [==============================] - 3s 8ms/step - loss: 1.0177 -
val_loss: 0.9406
Epoch 42/50
426/426 [==============================] - 3s 6ms/step - loss: 1.0177 -
val_loss: 0.9411
Epoch 43/50
426/426 [==============================] - 2s 6ms/step - loss: 1.0174 -
val_loss: 0.9411
Epoch 44/50
```

```
426/426 [==============================] - 3s 6ms/step - loss: 1.0175 -
val_loss: 0.9411
Epoch 45/50
426/426 [==============================] - 3s 7ms/step - loss: 1.0175 -
val_loss: 0.9408
Epoch 46/50
426/426 [==============================] - 4s 9ms/step - loss: 1.0173 -
val_loss: 0.9416
Epoch 47/50
426/426 [==============================] - 3s 6ms/step - loss: 1.0172 -
val_loss: 0.9410
Epoch 48/50
426/426 [==============================] - 3s 6ms/step - loss: 1.0170 -
val_loss: 0.9420
Epoch 49/50
426/426 [==============================] - 2s 6ms/step - loss: 1.0167 -
val_loss: 0.9410
Epoch 50/50
426/426 [==============================] - 4s 9ms/step - loss: 1.0169 -
val_loss: 0.9411
```

[173]:
```
<keras.callbacks.History at 0x7bd561e87b80>
```

[174]:
```python
model.summary()
```

```
Model: "sequential_2"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_10 (LSTM)              (None, 4)                 112

 dense_1 (Dense)             (None, 1)                 5

=================================================================
Total params: 117
Trainable params: 117
Non-trainable params: 0
_________________________________________________________________
```

[175]:
```python
#invert pridiction
trainprediction=model.predict(xtrain)
testprediction=model.predict(xtest)
```

```
213/213 [==============================] - 1s 2ms/step
71/71 [==============================] - 0s 4ms/step
```

[156]:
```python
from sklearn.metrics import mean_squared_error
```

[176]:
```python
print(mean_squared_error(trainprediction,ytrain))
```

```
1.0163514446549322
```

[177]:
```python
ytrain
```

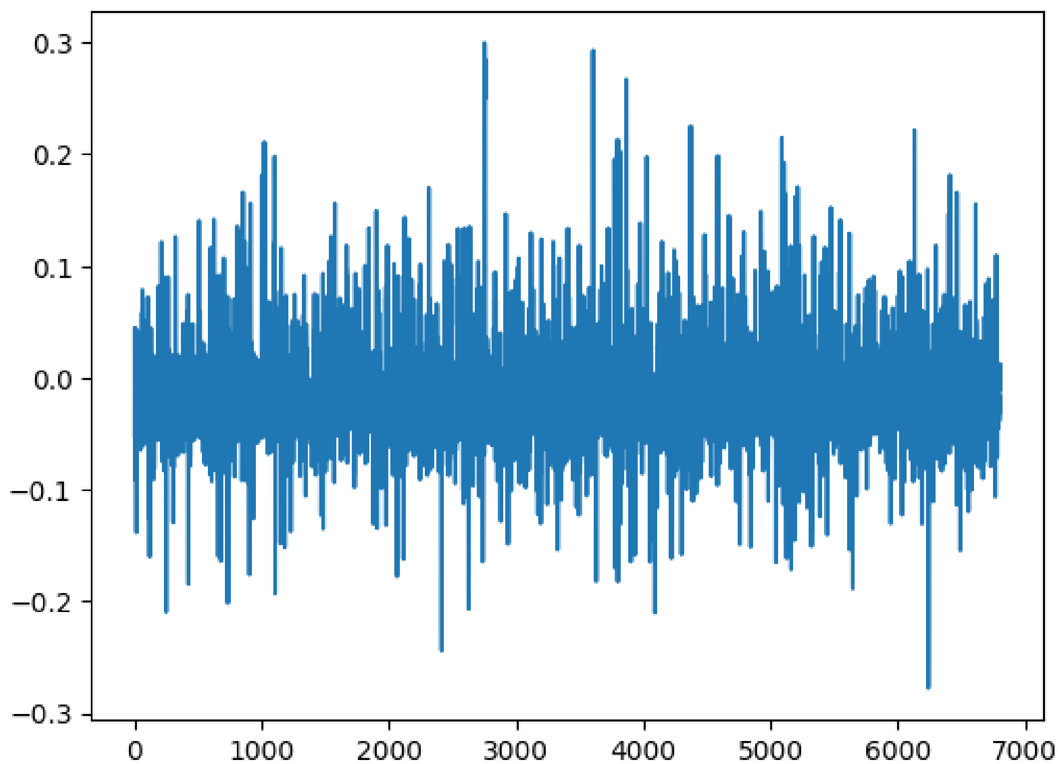
[177]:
```
array([-0.28310702, -0.29424361,  0.24985899, …, -0.27297519,
       -0.01093009, -0.52144498])
```

[178]:
```python
trainprediction
```

[178]:
```
array([[ 0.04358709],
       [-0.05365378],
       [-0.02182905],
       …,
       [ 0.01246042],
       [-0.00134046],
       [-0.00943948]], dtype=float32)
```

[180]:
```python
plt.plot(trainprediction)
```

[180]:
```
[<matplotlib.lines.Line2D at 0x7bd56a2ab4c0>]
```



[ ]:

```

```