IMPORT REQUIRED LIBRARIES

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df_stock = pd.read_csv('AAPL.csv')
df_stock.head()
```

Out[2]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1980-12-12	0.128348	0.128906	0.128348	0.128348	0.100178	469033600
1	1980-12-15	0.122210	0.122210	0.121652	0.121652	0.094952	175884800
2	1980-12-16	0.113281	0.113281	0.112723	0.112723	0.087983	105728000
3	1980-12-17	0.115513	0.116071	0.115513	0.115513	0.090160	86441600
4	1980-12-18	0.118862	0.119420	0.118862	0.118862	0.092774	73449600

In [3]:
```
df_stock.shape
```

Out[3]:
```
(10468, 7)
```

In [4]:
```
df_stock.describe()
```

Out[4]:

	Open	High	Low	Close	Adj Close	Volume
count	10468.000000	10468.000000	10468.000000	10468.000000	10468.000000	1.046800e+04
mean	14.757987	14.921491	14.594484	14.763533	14.130431	3.308489e+08
std	31.914174	32.289158	31.543959	31.929489	31.637275	3.388418e+08
min	0.049665	0.049665	0.049107	0.049107	0.038329	0.000000e+00
25%	0.283482	0.289286	0.276786	0.283482	0.235462	1.237768e+08
50%	0.474107	0.482768	0.465960	0.475446	0.392373	2.181592e+08
75%	14.953303	15.057143	14.692589	14.901964	12.835269	4.105794e+08
max	182.630005	182.940002	179.119995	182.009995	181.511703	7.421641e+09

In [5]:
```
df_stock.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10468 entries, 0 to 10467
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       10468 non-null  object 
 1   Open       10468 non-null  float64
 2   High       10468 non-null  float64
 3   Low        10468 non-null  float64
 4   Close      10468 non-null  float64
 5   Adj Close  10468 non-null  float64
 6   Volume     10468 non-null  int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 572.6+ KB
```

In [6]:
```
df_stock.isnull().sum()
```

Out[6]:
```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

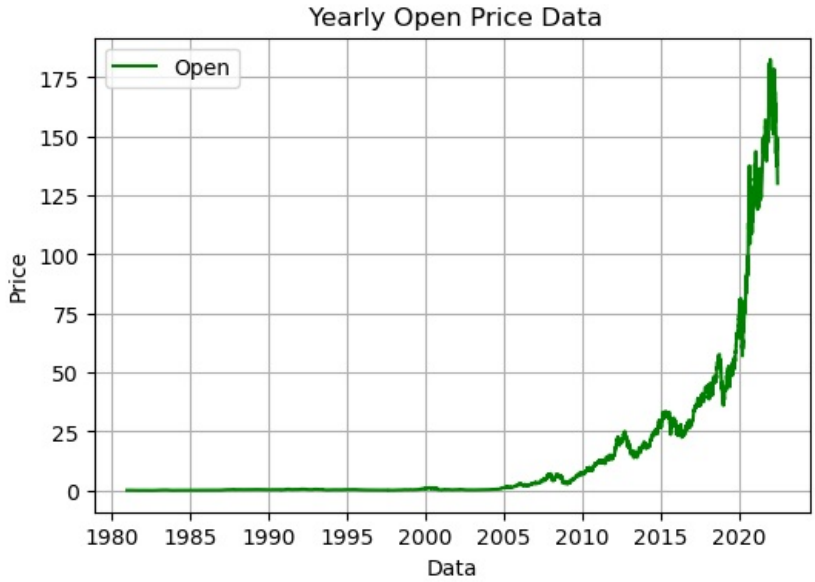
In [7]:
```
df_stock = df_stock[['Date','Open','Close']]

#convert date in datetime datatype
df_stock['Date'] = pd.to_datetime(df_stock['Date'].apply(lambda x: x.split()[0]))
df_stock.set_index('Date', drop=True, inplace=True)
df_stock.head()
```

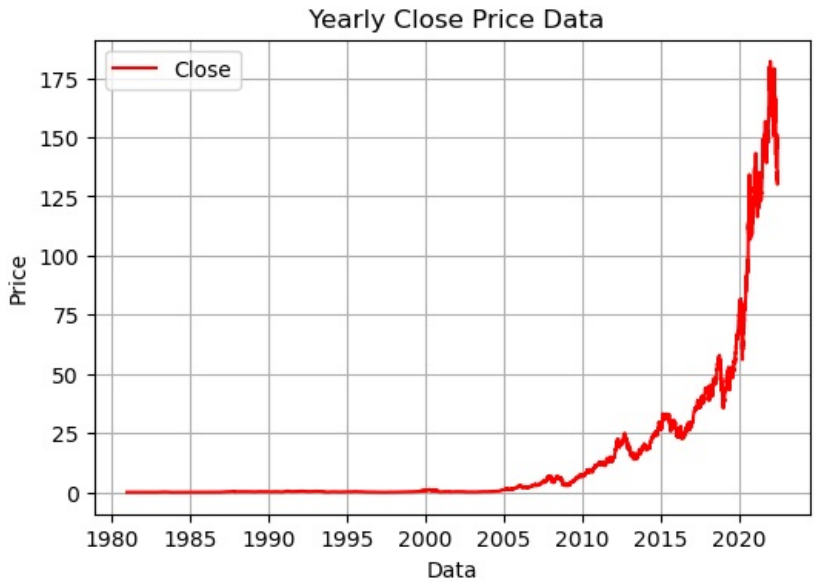
Out[7]:

Date	Open	Close
1980-12-12	0.128348	0.128348
1980-12-15	0.122210	0.121652
1980-12-16	0.113281	0.112723
1980-12-17	0.115513	0.115513
1980-12-18	0.118862	0.118862

In [8]:
```
plt.figure(figsize=(6,4))
plt.plot(df_stock['Open'], label = 'Open', color='g')
plt.xlabel('Data')
plt.ylabel('Price')
plt.title(' Yearly Open Price Data')
plt.grid()
plt.legend()
plt.show()
```



In [9]:
```
plt.figure(figsize=(6,4))
plt.plot(df_stock['Close'], label = 'Close', color='r')
plt.xlabel('Data')
plt.ylabel('Price')
plt.title(' Yearly Close Price Data')
plt.grid()
plt.legend()
plt.show()
```



data preprocessing

In [10]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler
```

Out[10]:
```
MinMaxScaler()
```

In [11]:
```
scaler.fit_transform(df_stock)
df_stock.head()
```

Out[11]:

Date	Open	Close
1980-12-12	0.128348	0.128348
1980-12-15	0.122210	0.121652
1980-12-16	0.113281	0.112723
1980-12-17	0.115513	0.115513
1980-12-18	0.118862	0.118862

In [12]:
```
training_size_data = round(len(df_stock)*0.75)
training_size_data
```

Out[12]:
```
7851
```

In [13]:
```
train_data = df_stock[:training_size_data] #75% of the data choose to train module
test_data = df_stock[training_size_data:] #25% of the data use for testing

train_data.shape, test_data.shape
```

Out[13]:
```
((7851, 2), (2617, 2))
```

In [14]:
```
# create a sequence of the data for training and testing

def create_sequence(data):
    sequence = []
    label = []

    start_idx = 0

    for stop_idx in range(50, len(data)):#selecting 50 rows at a time
        sequence.append(data.iloc[start_idx:stop_idx])
        label.append(data.iloc[stop_idx])
        start_idx += 1
    return (np.array(sequence), np.array(label))
```

In [15]:
```
x_train , y_train = create_sequence(train_data)
x_test, y_test = create_sequence(test_data)
```

In [16]:
```
x_train.shape,y_train.shape, x_test.shape, y_test.shape
```

Out[16]:
```
((7801, 50, 2), (7801, 2), (2567, 50, 2), (2567, 2))
```

create LSTM model

In [17]:
```
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM, Bidirectional
```

In [18]:
```
#import sequential from keras.models
model = Sequential()

# import Dense, Dropout, LSTM, Bidirectional from keras.layers
model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1], x_train.shape[2])))

model.add(Dropout(0.1))
model.add(LSTM(units=50))

model.add(Dense(2))

model.compile(loss = 'mean_squared_error', optimizer = 'adam', metrics = ['mean_absolute_error'])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 50, 50)            10600

 dropout (Dropout)           (None, 50, 50)            0

 lstm_1 (LSTM)               (None, 50)                20200

 dense (Dense)               (None, 2)                 102

=================================================================
Total params: 30902 (120.71 KB)
Trainable params: 30902 (120.71 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [19]:
```
model.fit(x_train, y_train, epochs = 100, validation_data=(x_test,y_test),verbose=100)
```

```
Epoch 1/100
Epoch 2/100
Epoch 3/100
Epoch 4/100
Epoch 5/100
Epoch 6/100
Epoch 7/100
Epoch 8/100
Epoch 9/100
Epoch 10/100
Epoch 11/100
Epoch 12/100
Epoch 13/100
Epoch 14/100
Epoch 15/100
Epoch 16/100
Epoch 17/100
Epoch 18/100
Epoch 19/100
Epoch 20/100
Epoch 21/100
Epoch 22/100
Epoch 23/100
Epoch 24/100
Epoch 25/100
Epoch 26/100
Epoch 27/100
Epoch 28/100
Epoch 29/100
Epoch 30/100
Epoch 31/100
Epoch 32/100
Epoch 33/100
Epoch 34/100
Epoch 35/100
Epoch 36/100
Epoch 37/100
Epoch 38/100
Epoch 39/100
Epoch 40/100
Epoch 41/100
Epoch 42/100
Epoch 43/100
Epoch 44/100
Epoch 45/100
Epoch 46/100
Epoch 47/100
Epoch 48/100
Epoch 49/100
Epoch 50/100
Epoch 51/100
Epoch 52/100
Epoch 53/100
Epoch 54/100
Epoch 55/100
Epoch 56/100
Epoch 57/100
Epoch 58/100
Epoch 59/100
Epoch 60/100
Epoch 61/100
Epoch 62/100
Epoch 63/100
Epoch 64/100
Epoch 65/100
Epoch 66/100
Epoch 67/100
Epoch 68/100
Epoch 69/100
```

```
Epoch 70/100
Epoch 71/100
Epoch 72/100
Epoch 73/100
Epoch 74/100
Epoch 75/100
Epoch 76/100
Epoch 77/100
Epoch 78/100
Epoch 79/100
Epoch 80/100
Epoch 81/100
Epoch 82/100
Epoch 83/100
Epoch 84/100
Epoch 85/100
Epoch 86/100
Epoch 87/100
Epoch 88/100
Epoch 89/100
Epoch 90/100
Epoch 91/100
Epoch 92/100
Epoch 93/100
Epoch 94/100
Epoch 95/100
Epoch 96/100
Epoch 97/100
Epoch 98/100
Epoch 99/100
Epoch 100/100
```

Out[19]:
```
<keras.src.callbacks.History at 0x25e3bdb2810>
```

In [20]:
```
test_predicted = model.predict(x_test)
test_predicted[:5]
```

```
81/81 [==============================] - 1s 9ms/step
```

Out[20]:
```
array([[15.501729, 15.406836],
       [15.501266, 15.407749],
       [15.499518, 15.407742],
       [15.498555, 15.40855 ],
       [15.497701, 15.409233]], dtype=float32)
```

In [21]:
```
test_inverse_predicted = scaler.inverse_transform(test_predicted)
test_inverse_predicted[:5]
```

Out[21]:
```
array([[2830.3606, 2803.4905],
       [2830.276 , 2803.6567],
       [2829.957 , 2803.6555],
       [2829.781 , 2803.8025],
       [2829.625 , 2803.9268]], dtype=float32)
```

In [ ]:
```

```