

# Apple Stock Prediction Using LSTM

## Importing Libraries

```
In [32]: import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
plt.style.use("fivethirtyeight")
%matplotlib inline
```

## using YAHOO for getting the data -> yfinance

```
In [33]: from pandas_datareader.data import DataReader
import yfinance as yf #yahoo finance for data collection
from pandas_datareader import data as pdr

yf.pdr_override()

# For time stamps in stocks analysis

from datetime import datetime

# The tech stocks we'll use for this analysis are
tech_list = ['AAPL', 'GOOG', 'MSFT', 'AMZN']
tech_list = ['AAPL', 'GOOG', 'MSFT', 'AMZN']

end = datetime.now()
start = datetime(end.year - 1, end.month, end.day)

for stock in tech_list:
    globals()[stock] = yf.download(stock, start, end)

company_list = [AAPL, GOOG, MSFT, AMZN]
company_name = ["APPLE", "GOOGLE", "MICROSOFT", "AMAZON"]

for company, com_name in zip(company_list, company_name):
    company["company_name"] = com_name

df = pd.concat(company_list, axis=0)
df.tail(10)
```

```
[*****100%*****] 1 of 1 completed
[*****100%*****] 1 of 1 completed
[*****100%*****] 1 of 1 completed
[*****100%*****] 1 of 1 completed
```

Out[33]:

	Open	High	Low	Close	Adj Close	Volume	company_name
Date							
2023-07-31	133.199997	133.869995	132.380005	133.679993	133.679993	41901500	AMAZON
2023-08-01	133.550003	133.690002	131.619995	131.690002	131.690002	42098500	AMAZON
2023-08-02	130.149994	130.229996	126.820000	128.210007	128.210007	51027600	AMAZON
2023-08-03	127.480003	129.839996	126.410004	128.910004	128.910004	88585200	AMAZON
2023-08-04	141.059998	143.630005	139.320007	139.570007	139.570007	152938700	AMAZON
2023-08-07	140.990005	142.539993	138.949997	142.220001	142.220001	71213100	AMAZON
2023-08-08	140.619995	140.839996	138.419998	139.940002	139.940002	51710500	AMAZON
2023-08-09	139.970001	140.320007	137.100006	137.850006	137.850006	50017300	AMAZON
2023-08-10	139.070007	140.410004	137.490005	138.559998	138.559998	58928400	AMAZON
2023-08-11	137.399994	139.330002	137.000000	138.410004	138.410004	42832100	AMAZON

In [34]: *#describing APPLE stocks - getting descriptive data*  
AAPL.describe()

Out[34]:

	Open	High	Low	Close	Adj Close	Volume
count	250.000000	250.000000	250.000000	250.000000	250.000000	2.500000e+02
mean	159.283201	161.043800	157.671680	159.401040	158.862302	7.050679e+07
std	18.260184	18.011765	18.534540	18.243009	18.377383	2.361556e+07
min	126.010002	127.769997	124.169998	125.019997	124.488869	3.145820e+07
25%	145.812500	147.320004	143.957497	145.915001	145.256302	5.246052e+07
50%	154.719994	156.915001	153.360001	154.824997	154.193527	6.583690e+07
75%	173.015003	173.887497	171.682503	173.150002	172.681816	8.141450e+07
max	196.240005	198.229996	195.279999	196.449997	196.185074	1.647624e+08

In [35]: AAPL.info()

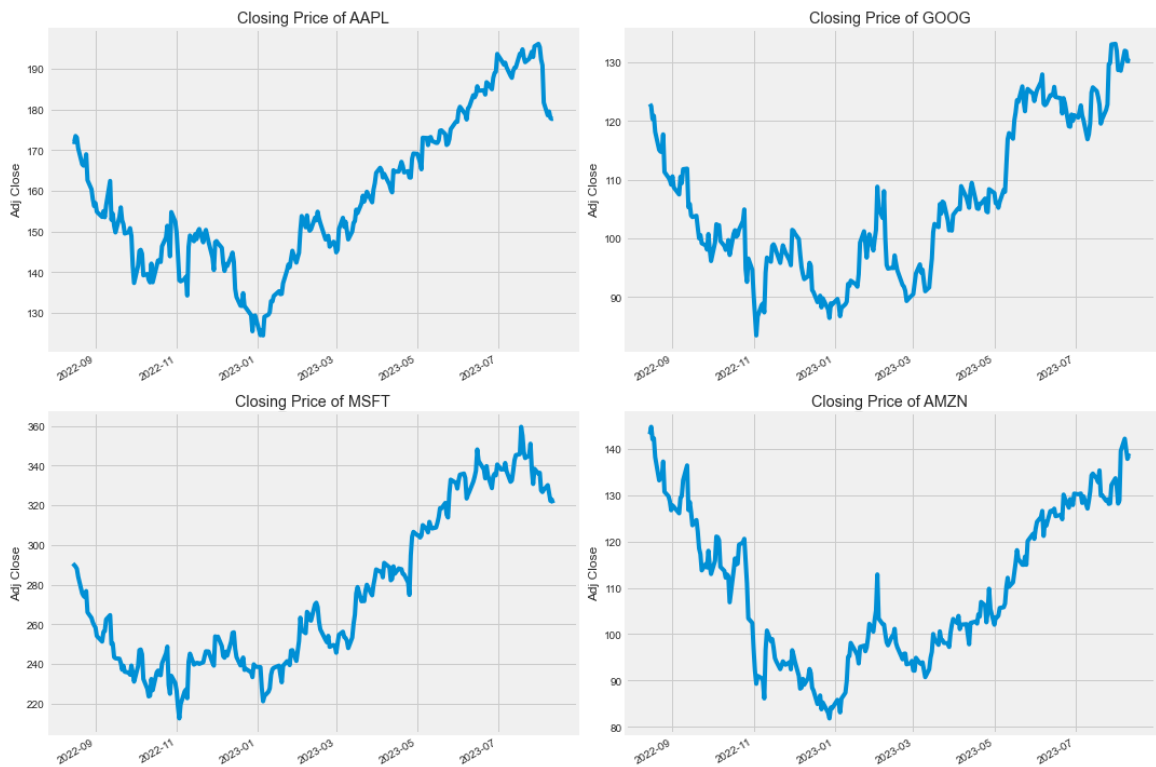
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 250 entries, 2022-08-15 to 2023-08-11
Data columns (total 7 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Open            250 non-null   float64
1   High            250 non-null   float64
2   Low             250 non-null   float64
3   Close           250 non-null   float64
4   Adj Close       250 non-null   float64
5   Volume          250 non-null   int64
6   company_name    250 non-null   object
dtypes: float64(5), int64(1), object(1)
memory usage: 15.6+ KB
```

## plotting the closing price of the STOCKS

```
In [39]: plt.figure(figsize=(15, 10))
plt.subplots_adjust(top=1.25, bottom=1.2)

for i, company in enumerate(company_list, 1):
    plt.subplot(2, 2, i)
    company['Adj Close'].plot()
    plt.ylabel('Adj Close')
    plt.xlabel(None)
    plt.title(f"Closing Price of {tech_list[i - 1]}")

plt.tight_layout()
```

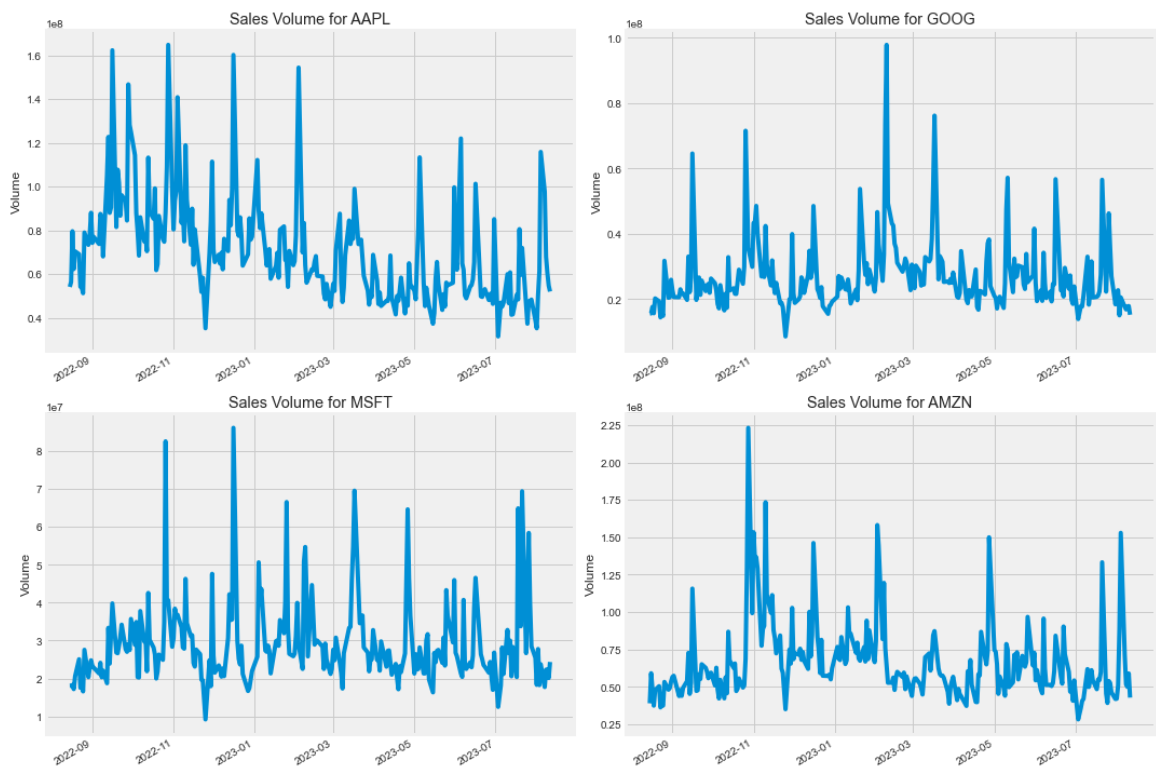


## Plotting the volume of sales

```
In [40]: plt.figure(figsize=(15, 10))
plt.subplots_adjust(top=1.25, bottom=1.2)

for i, company in enumerate(company_list, 1):
    plt.subplot(2, 2, i)
    company['Volume'].plot()
    plt.ylabel('Volume')
    plt.xlabel(None)
    plt.title(f"Sales Volume for {tech_list[i - 1]}")

plt.tight_layout()
```



## Moving average of various Stocks

```

In [41]: ma_day = [10, 20, 50]

for ma in ma_day:
    for company in company_list:
        column_name = f"MA for {ma} days"
        company[column_name] = company['Adj Close'].rolling(ma).mean()

fig, axes = plt.subplots(nrows=2, ncols=2)
fig.set_figheight(10)
fig.set_figwidth(15)

AAPL[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(a
axes[0,0].set_title('APPLE')

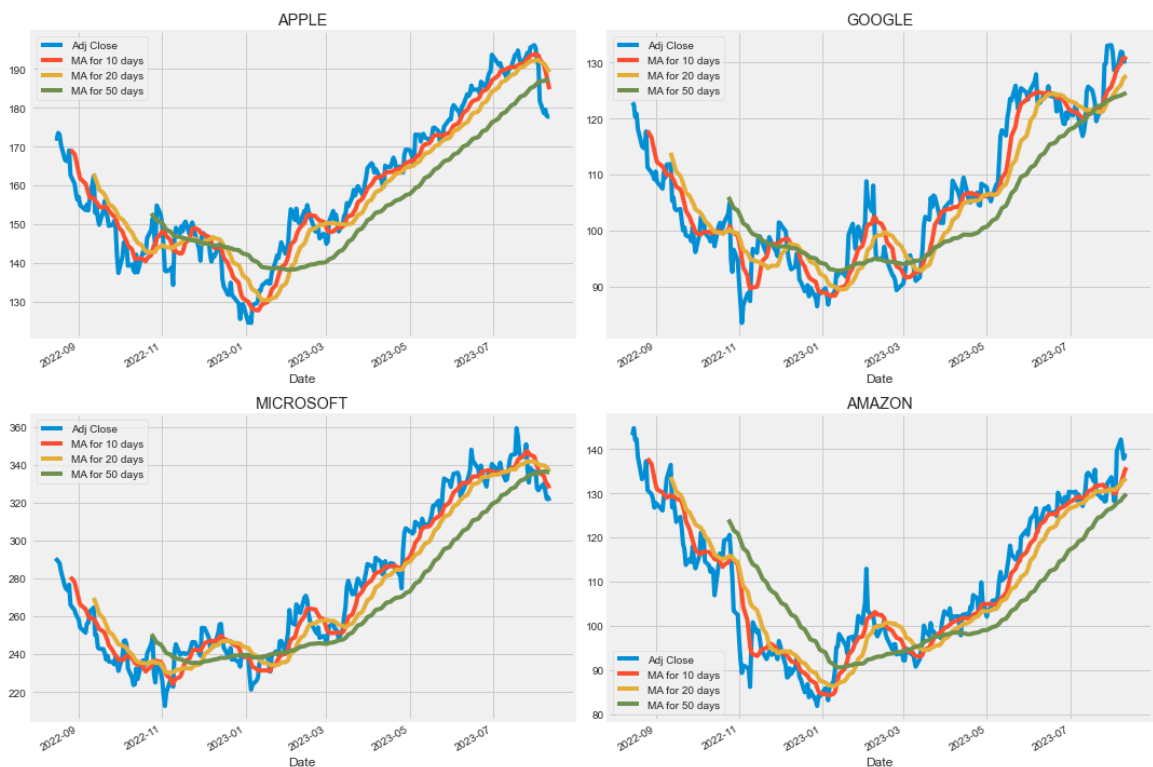
GOOG[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(a
axes[0,1].set_title('GOOGLE')

MSFT[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(a
axes[1,0].set_title('MICROSOFT')

AMZN[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(a
axes[1,1].set_title('AMAZON')

fig.tight_layout()

```



```
In [43]: for company in company_list:
          company['Daily Return'] = company['Adj Close'].pct_change()

# Then we'll plot the daily return percentage
fig, axes = plt.subplots(nrows=2, ncols=2)
fig.set_figheight(10)
fig.set_figwidth(15)

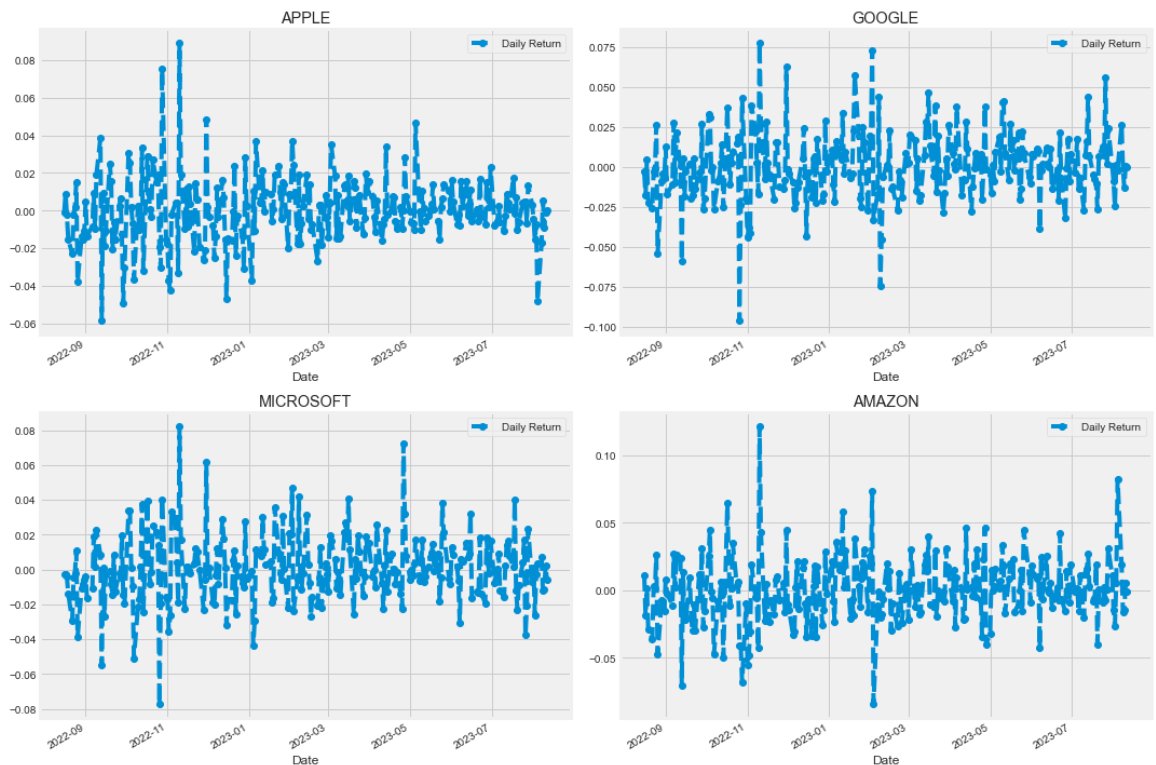
AAPL['Daily Return'].plot(ax=axes[0,0], legend=True, linestyle='--', marker='o')
axes[0,0].set_title('APPLE')

GOOG['Daily Return'].plot(ax=axes[0,1], legend=True, linestyle='--', marker='o')
axes[0,1].set_title('GOOGLE')

MSFT['Daily Return'].plot(ax=axes[1,0], legend=True, linestyle='--', marker='o')
axes[1,0].set_title('MICROSOFT')

AMZN['Daily Return'].plot(ax=axes[1,1], legend=True, linestyle='--', marker='o')
axes[1,1].set_title('AMAZON')

fig.tight_layout()
```

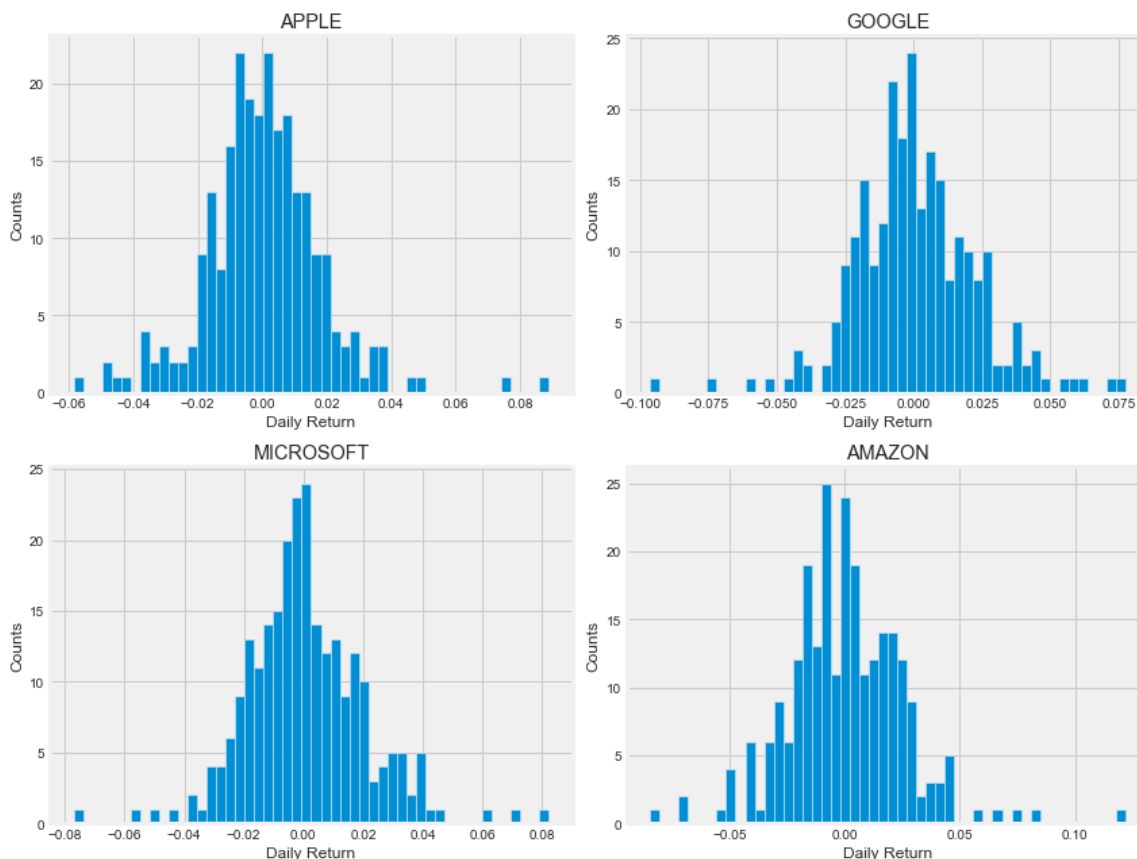


**Daily return of the stock on average with time**

```
In [44]: plt.figure(figsize=(12, 9))

for i, company in enumerate(company_list, 1):
    plt.subplot(2, 2, i)
    company['Daily Return'].hist(bins=50)
    plt.xlabel('Daily Return')
    plt.ylabel('Counts')
    plt.title(f'{company_name[i - 1]}')

plt.tight_layout()
```



## Correlation between stocks for closing prices

```
In [45]: closing_df = pdr.get_data_yahoo(tech_list, start=start, end=end)['Adj Close']

# Make a new tech returns DataFrame
tech_returns = closing_df.pct_change()
tech_returns.head()
```

[\*\*\*\*\*100%\*\*\*\*\*] 4 of 4 completed

Out[45]:

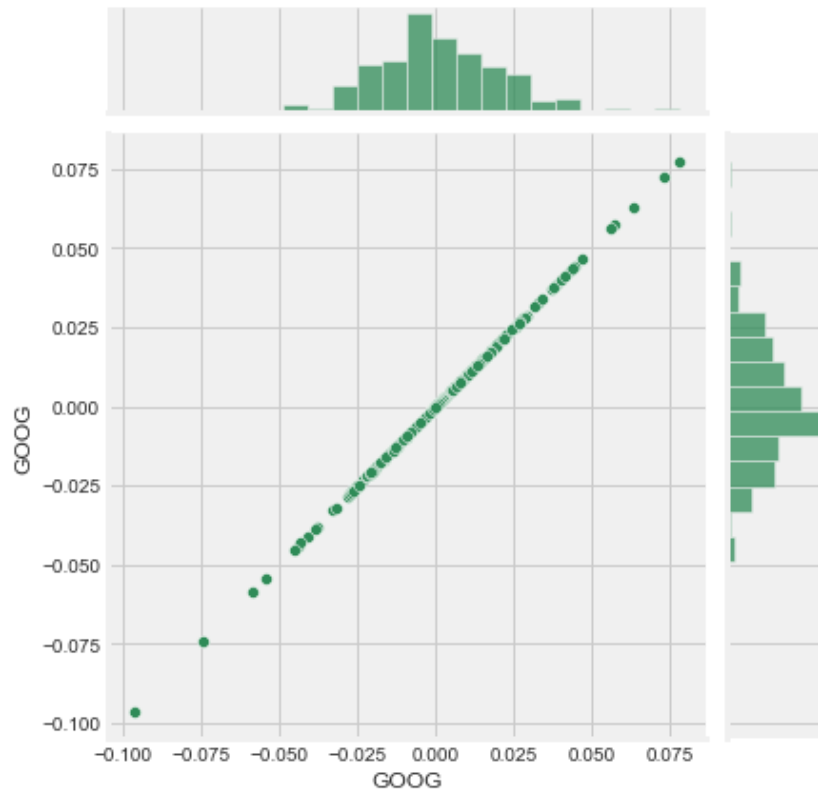
	AAPL	AMZN	GOOG	MSFT
Date				
2022-08-15	NaN	NaN	NaN	NaN
2022-08-16	-0.000924	0.011175	-0.003011	-0.002590
2022-08-17	0.008785	-0.018511	-0.017876	-0.002636
2022-08-18	-0.002292	0.001407	0.004488	-0.003947
2022-08-19	-0.015102	-0.028602	-0.022671	-0.013854



## comparing google stocks to itself for any relation

```
In [47]: sns.jointplot(x='GOOG', y='GOOG', data=tech_rets, kind='scatter', color='seagreen')
```

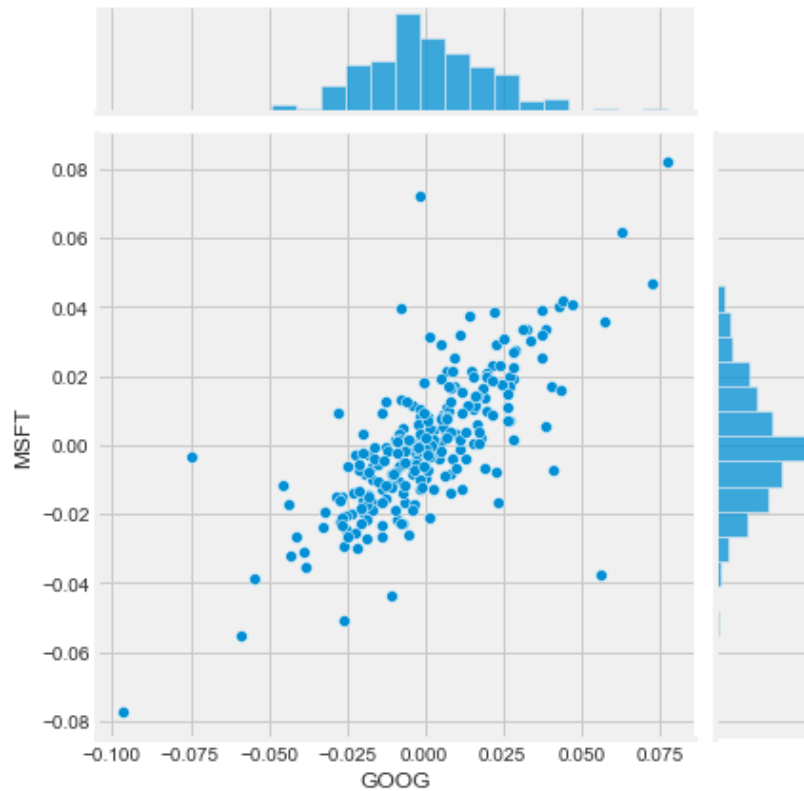
```
Out[47]: <seaborn.axisgrid.JointGrid at 0x26e970c1b20>
```



## comparing daily returns of google and microsoft stocks

```
In [48]: sns.jointplot(x='GOOG', y='MSFT', data=tech_rets, kind='scatter')
```

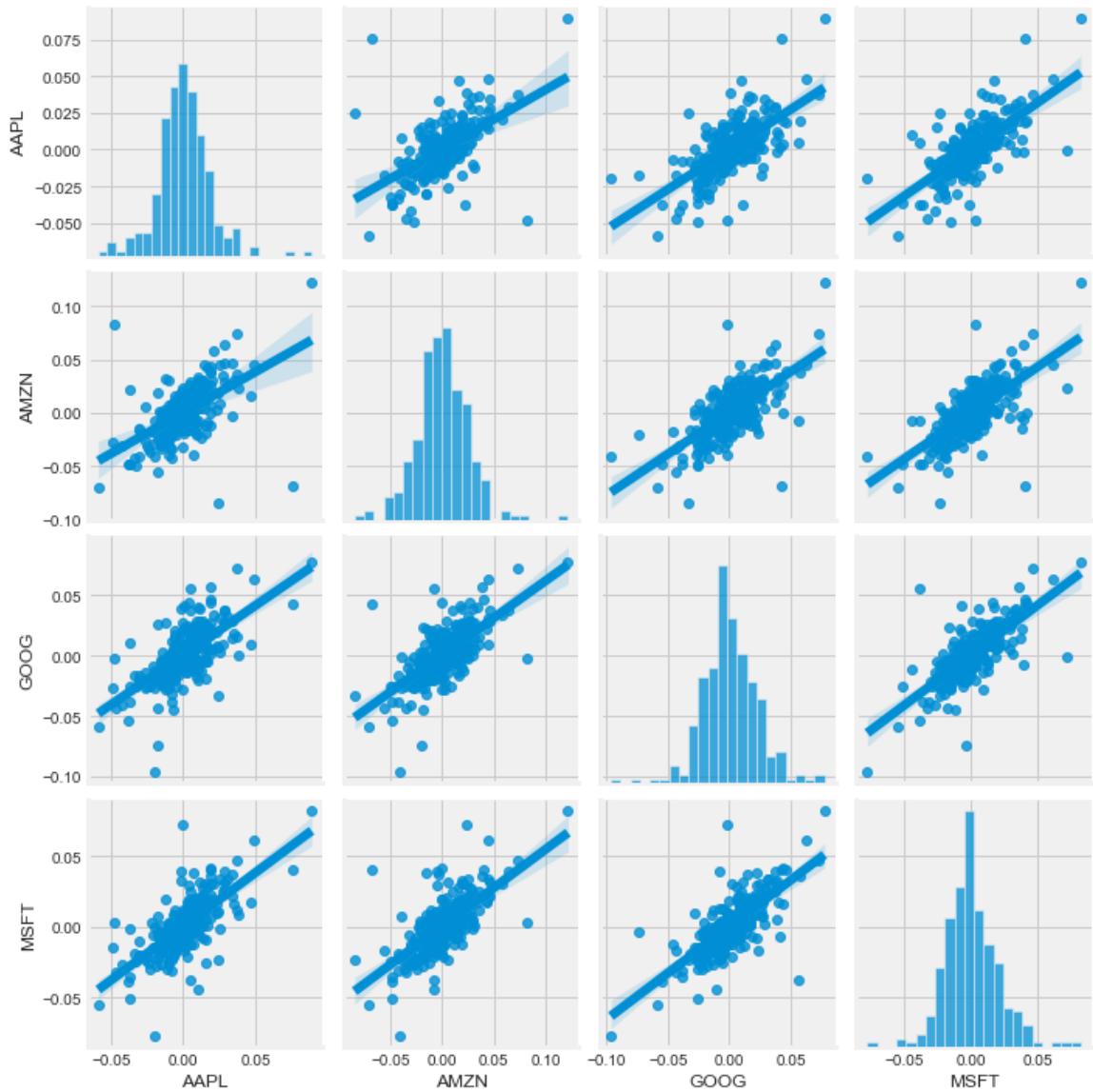
```
Out[48]: <seaborn.axisgrid.JointGrid at 0x26e9dcb9ee0>
```



## Pairplot - to understand the best set of features to explain a relationship between two variables or to form the most separated clusters

```
In [49]: sns.pairplot(tech_rets, kind='reg')
```

```
Out[49]: <seaborn.axisgrid.PairGrid at 0x26e9e1263a0>
```



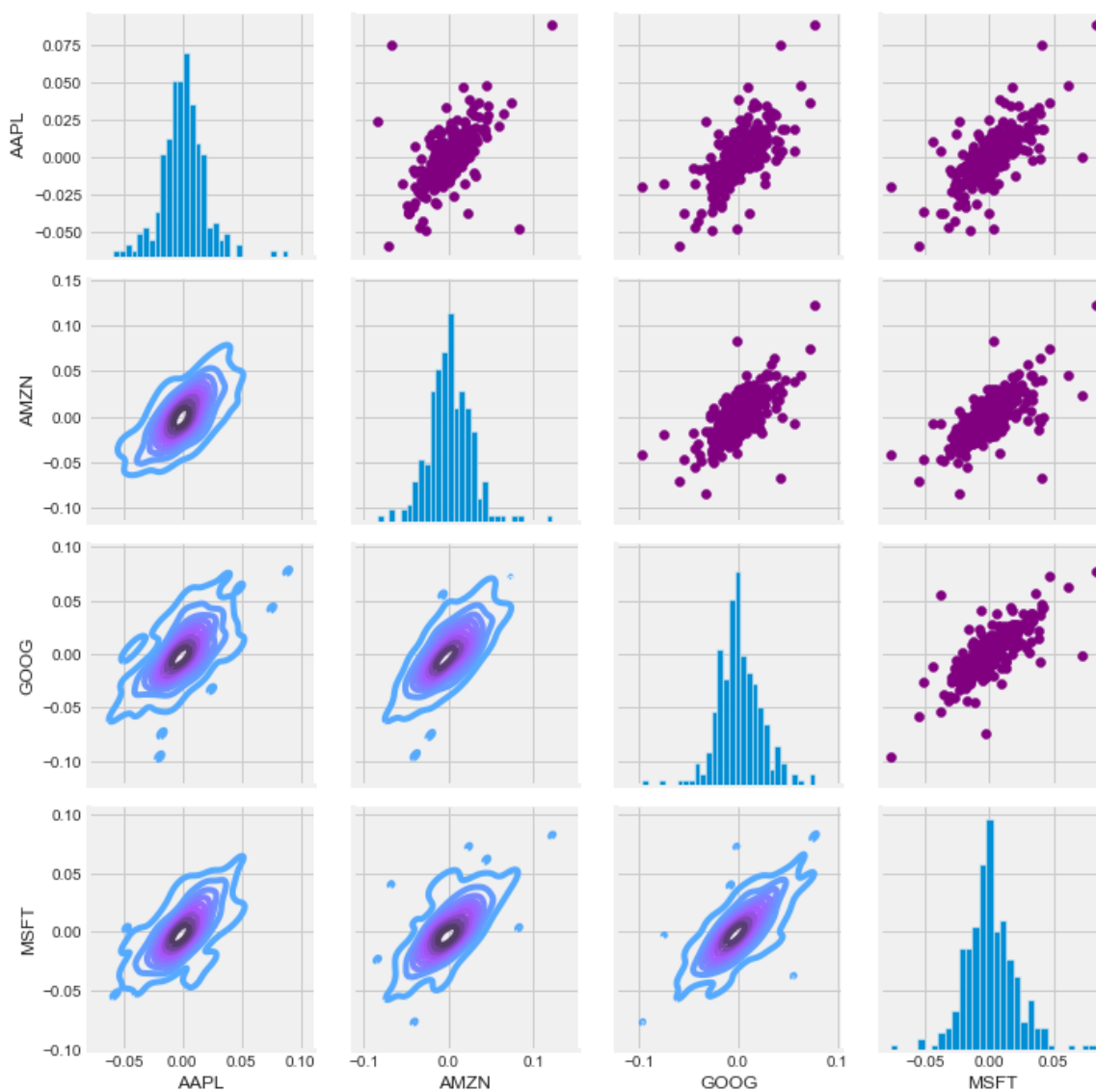
```
In [50]: return_fig = sns.PairGrid(tech_rets.dropna())

# map_upper is used to specify what the upper triangle will look like.
return_fig.map_upper(plt.scatter, color='purple')

#the lower triangle in the figure, including the plot type (kde)
# or the color map (BluePurple)
return_fig.map_lower(sns.kdeplot, cmap='cool_d')

#the diagonal as a series of histogram plots of the daily return
return_fig.map_diag(plt.hist, bins=30)
```

Out[50]: <seaborn.axisgrid.PairGrid at 0x26ea026e670>



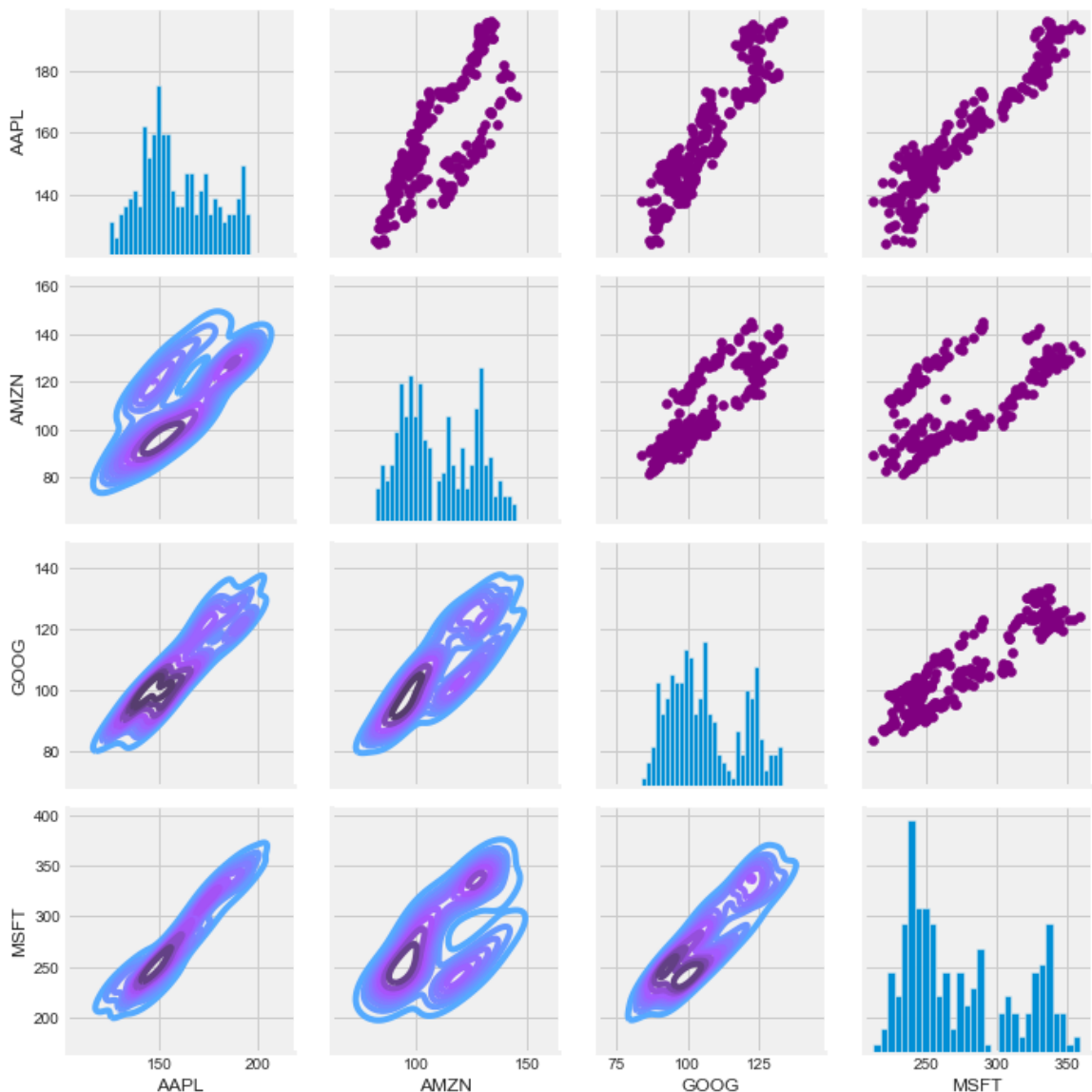
```
In [18]: returns_fig = sns.PairGrid(closing_df)

# Using map_upper we can specify what the upper triangle will look like.
returns_fig.map_upper(plt.scatter,color='purple')

# We can also define the lower triangle in the figure, including the plot type (
returns_fig.map_lower(sns.kdeplot,cmap='cool_d')

# Finally we'll define the diagonal as a series of histogram plots of the daily
returns_fig.map_diag(plt.hist,bins=30)
```

Out[18]: <seaborn.axisgrid.PairGrid at 0x26e94b22940>



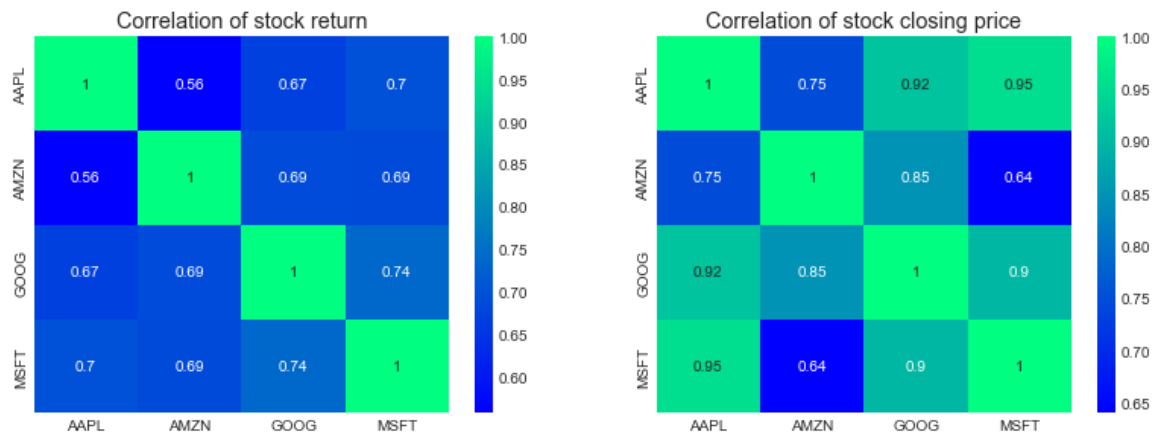
## Correlation Plot

```
In [52]: plt.figure(figsize=(12, 10))

plt.subplot(2, 2, 1)
sns.heatmap(tech_rets.corr(), annot=True, cmap='winter')
plt.title('Correlation of stock return')

plt.subplot(2, 2, 2)
sns.heatmap(closing_df.corr(), annot=True, cmap='winter')
plt.title('Correlation of stock closing price')
```

Out[52]: Text(0.5, 1.0, 'Correlation of stock closing price')



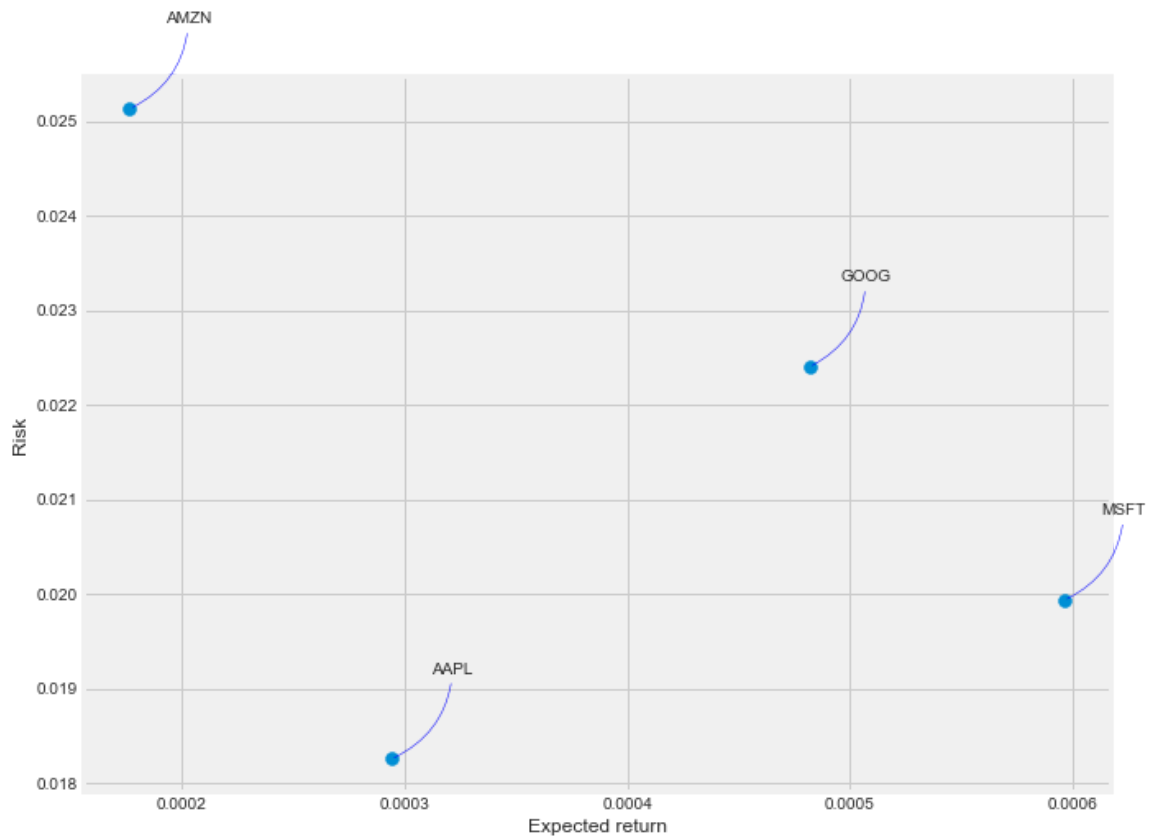
## Risk calculation for every stock

```
In [53]: rets = tech_rets.dropna()

area = np.pi * 20

plt.figure(figsize=(10, 8))
plt.scatter(rets.mean(), rets.std(), s=area)
plt.xlabel('Expected return')
plt.ylabel('Risk')

for label, x, y in zip(rets.columns, rets.mean(), rets.std()):
    plt.annotate(label, xy=(x, y), xytext=(50, 50), textcoords='offset points',
                 arrowprops=dict(arrowstyle='-', color='blue', connectionstyle=''))
```



**predicting the closing price of APPLE using LSTM**

```
In [54]: df = pdr.get_data_yahoo('AAPL', start='2012-01-01', end=datetime.now())
```

df

```
[*****100%*****] 1 of 1 completed
```

Out[54]:

	Open	High	Low	Close	Adj Close	Volume
Date						
2012-01-03	14.621429	14.732143	14.607143	14.686786	12.466089	302220800
2012-01-04	14.642857	14.810000	14.617143	14.765714	12.533087	260022000
2012-01-05	14.819643	14.948214	14.738214	14.929643	12.672231	271269600
2012-01-06	14.991786	15.098214	14.972143	15.085714	12.804701	318292800
2012-01-09	15.196429	15.276786	15.048214	15.061786	12.784392	394024400
...	...	...	...	...	...	...
2023-08-07	182.130005	183.130005	177.350006	178.850006	178.608810	97576100
2023-08-08	179.690002	180.270004	177.580002	179.800003	179.557526	67823000
2023-08-09	180.869995	180.929993	177.009995	178.190002	177.949707	60378500
2023-08-10	179.479996	180.750000	177.600006	177.970001	177.729996	54686900
2023-08-11	177.320007	178.619995	176.550003	177.789993	177.789993	51988100

2921 rows × 6 columns

```
In [55]: #plotting the data
```

```
plt.figure(figsize=(16,6))
plt.title('Close Price History')
plt.plot(df['Close'])
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.show()
```



```
In [56]: data = df.filter(['Close'])
# Convert the dataframe to a numpy array
dataset = data.values
# Get the number of rows to train the model on
training_data_len = int(np.ceil( len(dataset) * .95 ))

training_data_len
```

Out[56]: 2775



```
In [57]: from sklearn.preprocessing import MinMaxScaler
# scaling the data on a same scale
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)

scaled_data
```

```
Out[57]: array([[0.00405082],
                [0.0044833 ],
                [0.00538153],
                ...,
                [0.8999466 ],
                [0.89874113],
                [0.8977548 ]])
```

In [25]:

```
# Create the scaled training data set
train_data = scaled_data[0:int(training_data_len), :]
# Split the data into x_train and y_train data sets
x_train = []
y_train = []

for i in range(60, len(train_data)):
    x_train.append(train_data[i-60:i, 0])
    y_train.append(train_data[i, 0])
    if i <= 61:
        print(x_train)
        print(y_train)
        print()

# Convert the x_train and y_train to numpy arrays
x_train, y_train = np.array(x_train), np.array(y_train)

# Reshaping the data
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
```

```
[array([0.00405082, 0.0044833 , 0.00538153, 0.0062367 , 0.00610559,
        0.00640108, 0.00626606, 0.00603905, 0.00572986, 0.0066868 ,
        0.0075498 , 0.00728366, 0.00582575, 0.00721712, 0.00584728,
        0.01098419, 0.01058694, 0.01110552, 0.01222684, 0.01290588,
        0.01284914, 0.01263975, 0.0135321 , 0.01437162, 0.01532269,
        0.01685887, 0.02008583, 0.02013475, 0.02193121, 0.02327365,
        0.02096645, 0.02185489, 0.02183728, 0.02432844, 0.02397423,
        0.02462979, 0.02580786, 0.02646344, 0.02835186, 0.02972757,
        0.03012483, 0.03026377, 0.02791156, 0.02734404, 0.0274282 ,
        0.02963952, 0.03026182, 0.0315984 , 0.03474903, 0.0389525 ,
        0.03816582, 0.03816777, 0.04120687, 0.04215794, 0.04148084,
        0.04086246, 0.04021863, 0.04235754, 0.04382523, 0.04443971])]
[0.04292113229660477]
```

```
[array([0.00405082, 0.0044833 , 0.00538153, 0.0062367 , 0.00610559,
        0.00640108, 0.00626606, 0.00603905, 0.00572986, 0.0066868 ,
        0.0075498 , 0.00728366, 0.00582575, 0.00721712, 0.00584728,
        0.01098419, 0.01058694, 0.01110552, 0.01222684, 0.01290588,
        0.01284914, 0.01263975, 0.0135321 , 0.01437162, 0.01532269,
        0.01685887, 0.02008583, 0.02013475, 0.02193121, 0.02327365,
        0.02096645, 0.02185489, 0.02183728, 0.02432844, 0.02397423,
        0.02462979, 0.02580786, 0.02646344, 0.02835186, 0.02972757,
        0.03012483, 0.03026377, 0.02791156, 0.02734404, 0.0274282 ,
        0.02963952, 0.03026182, 0.0315984 , 0.03474903, 0.0389525 ,
        0.03816582, 0.03816777, 0.04120687, 0.04215794, 0.04148084,
        0.04086246, 0.04021863, 0.04235754, 0.04382523, 0.04443971]), array([0.0
044833 , 0.00538153, 0.0062367 , 0.00610559, 0.00640108,
        0.00626606, 0.00603905, 0.00572986, 0.0066868 , 0.0075498 ,
        0.00728366, 0.00582575, 0.00721712, 0.00584728, 0.01098419,
        0.01058694, 0.01110552, 0.01222684, 0.01290588, 0.01284914,
        0.01263975, 0.0135321 , 0.01437162, 0.01532269, 0.01685887,
        0.02008583, 0.02013475, 0.02193121, 0.02327365, 0.02096645,
        0.02185489, 0.02183728, 0.02432844, 0.02397423, 0.02462979,
        0.02580786, 0.02646344, 0.02835186, 0.02972757, 0.03012483,
        0.03026377, 0.02791156, 0.02734404, 0.0274282 , 0.02963952,
        0.03026182, 0.0315984 , 0.03474903, 0.0389525 , 0.03816582,
        0.03816777, 0.04120687, 0.04215794, 0.04148084, 0.04086246,
        0.04021863, 0.04235754, 0.04382523, 0.04443971, 0.04292113])]
[0.04292113229660477, 0.04090355083781154]
```

## Using LSTM for prediction

```
In [62]: from keras.models import Sequential
from keras.layers import Dense, LSTM

# Building the LSTM model
model = Sequential()
model.add(LSTM(128, return_sequences=True, input_shape= (x_train.shape[1], 1)))
model.add(LSTM(64, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(x_train, y_train, batch_size=1, epochs=1)

2715/2715 [=====] - 57s 21ms/step - loss: 0.0012
```

Out[62]: <keras.callbacks.History at 0x26ea624aee0>

## Calculating the RMSE

```
In [63]: test_data = scaled_data[training_data_len - 60: , :]
# Create the data sets x_test and y_test
x_test = []
y_test = dataset[training_data_len:, :]
for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 0])

# Convert the data to a numpy array
x_test = np.array(x_test)

# Reshape the data
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1 ))

# Get the models predicted price values
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)

# Get the root mean squared error (RMSE)
rmse = np.sqrt(np.mean((predictions - y_test) ** 2))
rmse
```

Out[63]: 4.158798600996457

## Plotting the predicted graph

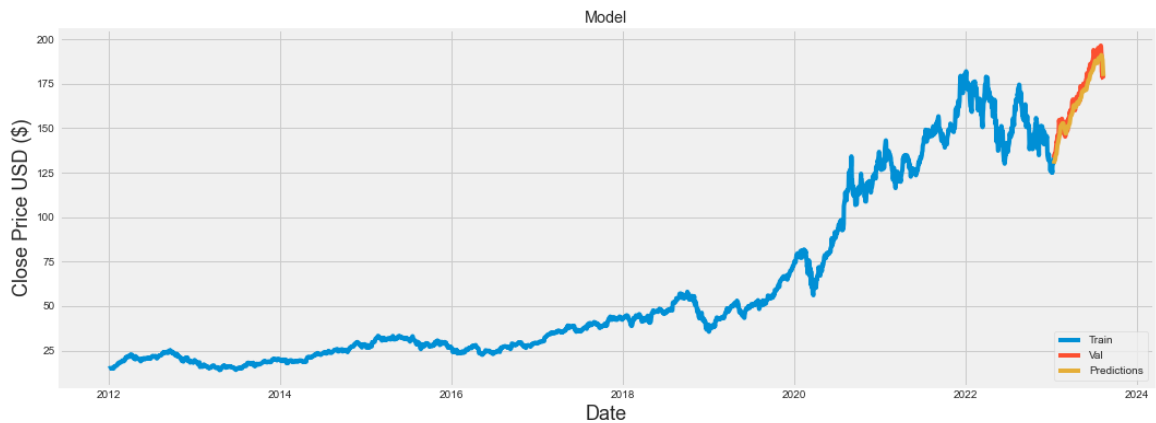
```
In [64]: train = data[:training_data_len]
valid = data[training_data_len:]
valid['Predictions'] = predictions
# Visualize the data
plt.figure(figsize=(16,6))
plt.title('Model')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.plot(train['Close'])
plt.plot(valid[['Close', 'Predictions']])
plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')
plt.show()
```

C:\Users\PUSPKA~1\AppData\Local\Temp\ipykernel\_14596\3344400724.py:3: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.  
Try using `.loc[row_indexer,col_indexer] = value` instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy) ([https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy))

```
valid['Predictions'] = predictions
```



## Comparing the closings with the predictions

In [65]: valid

Out[65]:

	Close	Predictions
Date		
2023-01-12	133.410004	130.041656
2023-01-13	134.759995	131.106140
2023-01-17	135.940002	132.220032
2023-01-18	135.210007	133.348862
2023-01-19	135.270004	134.203186
...	...	...
2023-08-07	178.850006	188.458984
2023-08-08	179.800003	185.776169
2023-08-09	178.190002	183.318893
2023-08-10	177.970001	181.086105
2023-08-11	177.789993	179.296158

146 rows × 2 columns

In [ ]: