Stock Market Analysis and Prediction

Importing packages and datasets

```
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

# For reading stock data from yahoo
from pandas_datareader.data import DataReader

# For time stamps
from datetime import datetime
```

Importing datasets from Yahoo reader

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

# Set up End and Start times for data grab
end = datetime.now()
start = datetime(end.year - 1, end.month, end.day)


#For loop for grabing yahoo finance data and setting as a dataframe
for stock in tech_list:
    # Set DataFrame as the Stock Ticker
    globals()[stock] = DataReader(stock, 'yahoo', start, end)
```

Display data characteristics

```
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)
```

Date	High	Low	Open	Close	Volume	Adj Close	c
2020-11-10	3114.000000	3019.479980	3095.020020	3035.020020	6591000.0	3035.020020	
2020-11-11	3139.149902	3050.000000	3061.780029	3137.389893	4366900.0	3137.389893	
2020-11-12	3175.879883	3086.050049	3159.949951	3110.280029	4362000.0	3110.280029	
2020-11-13	3141.719971	3085.389893	3122.000000	3128.810059	3756200.0	3128.810059	
2020-11-16	3142.699951	3072.689941	3093.199951	3131.060059	3808700.0	3131.060059	
2020-11-17	3189.250000	3135.260010	3183.540039	3135.659912	3444700.0	3135.659912	

Individual Stocks Description

```
AAPL.describe()
```

	High	Low	Open	Close	Volume	Adj Close
count	252.000000	252.000000	252.000000	252.000000	2.520000e+02	252.000000
mean	90.798700	88.234137	89.498700	89.578482	1.512577e+08	88.957172
std	20.814265	20.104491	20.665935	20.412608	7.569766e+07	20.765101
min	57.125000	53.152500	57.020000	56.092499	2.043060e+07	55.291519
25%	73.358124	71.272501	71.615000	72.314377	1.044864e+08	71.243650
50%	81.355000	80.130001	80.877499	81.026249	1.354628e+08	80.067600
75%	113.934376	110.073126	112.619999	112.167498	1.848412e+08	111.974318
max	137.979996	130.529999	137.589996	134.179993	4.268848e+08	133.948898

```
GOOG.describe()
```

	High	Low	Open	Close	Volume	Adj Close
count	252.000000	252.000000	252.000000	252.000000	2.520000e+02	252.000000
mean	1453.862414	1418.664735	1435.254270	1436.933391	1.876525e+06	1436.933391
std	149.597689	151.881462	150.527080	149.588543	7.748254e+05	149.588543
min	1071.319946	1013.536011	1056.510010	1056.619995	3.475000e+05	1056.619995

```
MSFT.describe()
```

	High	Low	Open	Close	Volume	Adj Close
count	252.000000	252.000000	252.000000	252.000000	2.520000e+02	252.000000
mean	188.973135	184.077698	186.526032	186.619524	3.712551e+07	185.511522
std	24.138847	24.072774	24.247144	24.096366	1.728312e+07	24.420203
min	140.570007	132.520004	137.009995	135.419998	8.989200e+06	134.366470
25%	166.767502	162.945007	165.070000	165.137501	2.504942e+07	163.815369
50%	187.154999	183.425003	185.489998	185.355003	3.215935e+07	184.206291
75%	211.190002	206.579994	208.922501	208.757500	4.391550e+07	207.740131
max	232.860001	227.350006	229.270004	231.649994	9.707360e+07	231.045105

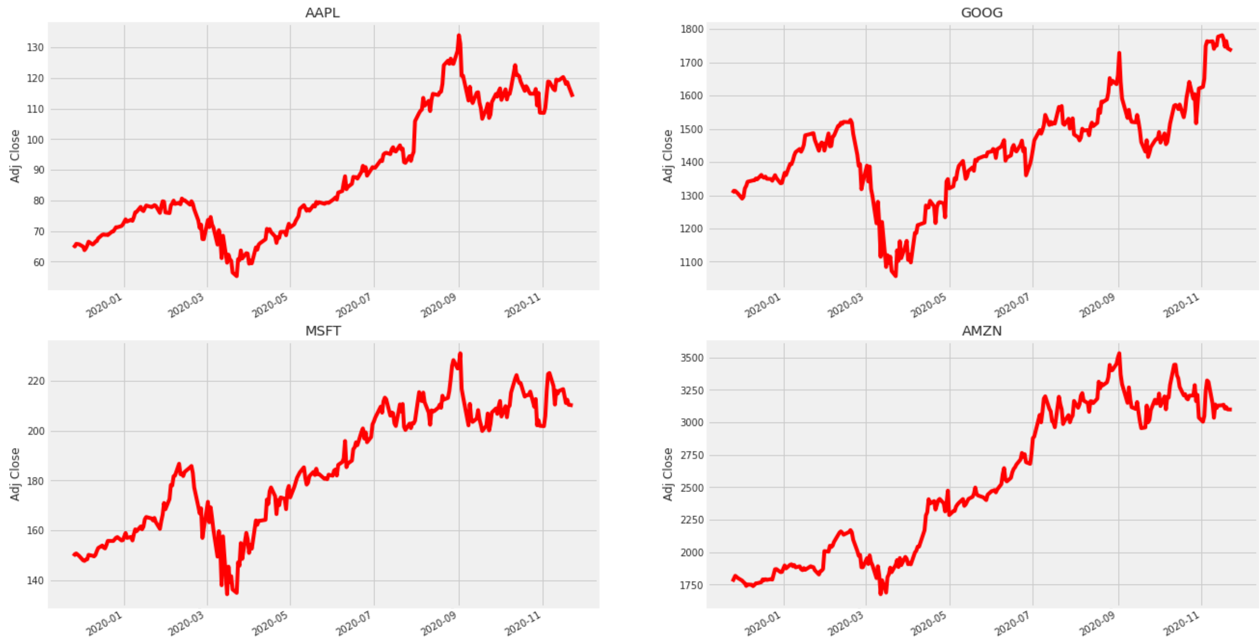
```
AMZN.describe()
```

	High	Low	Open	Close	Volume	Adj Close
count	252.000000	252.000000	252.000000	252.000000	2.520000e+02	252.000000
mean	2573.748925	2502.666507	2539.267663	2539.407336	4.904719e+06	2539.407336
std	587.659662	565.261876	580.502657	575.396847	2.009943e+06	575.396847
min	1750.000000	1626.030029	1641.510010	1676.609985	8.813000e+05	1676.609985
25%	1954.877502	1891.530029	1925.440033	1908.667480	3.429775e+06	1908.667480
50%	2475.964966	2433.629883	2451.505005	2454.965088	4.519400e+06	2454.965088
75%	3175.025024	3087.037537	3135.289978	3125.952515	5.807300e+06	3125.952515
max	3552.250000	3486.689941	3547.000000	3531.449951	1.556730e+07	3531.449951

Visualization of stocks

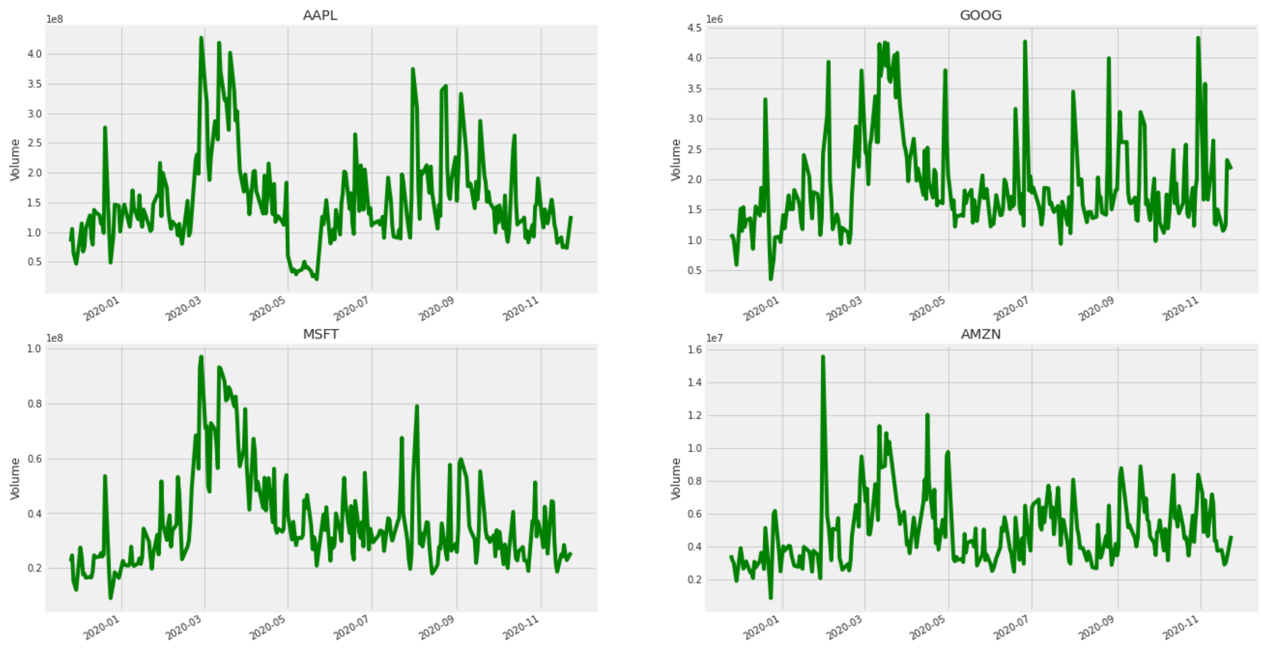
```
plt.figure(figsize=(20, 8))
plt.subplots_adjust(top=1.25, bottom=1.2)

for i, company in enumerate(company_list, 1):
    plt.subplot(2, 2, i)
```

```
    company['Adj Close'].plot(color='r')
    plt.ylabel('Adj Close')
    plt.xlabel(None)
    plt.title(f"{tech_list[i - 1]}")
```



Daily stocks exchange

```
# Now let's plot the total volume of stock being traded each day
plt.figure(figsize=(20, 8))
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(color='g')
    plt.ylabel('Volume')
    plt.xlabel(None)
    plt.title(f"{tech_list[i - 1]}")
```



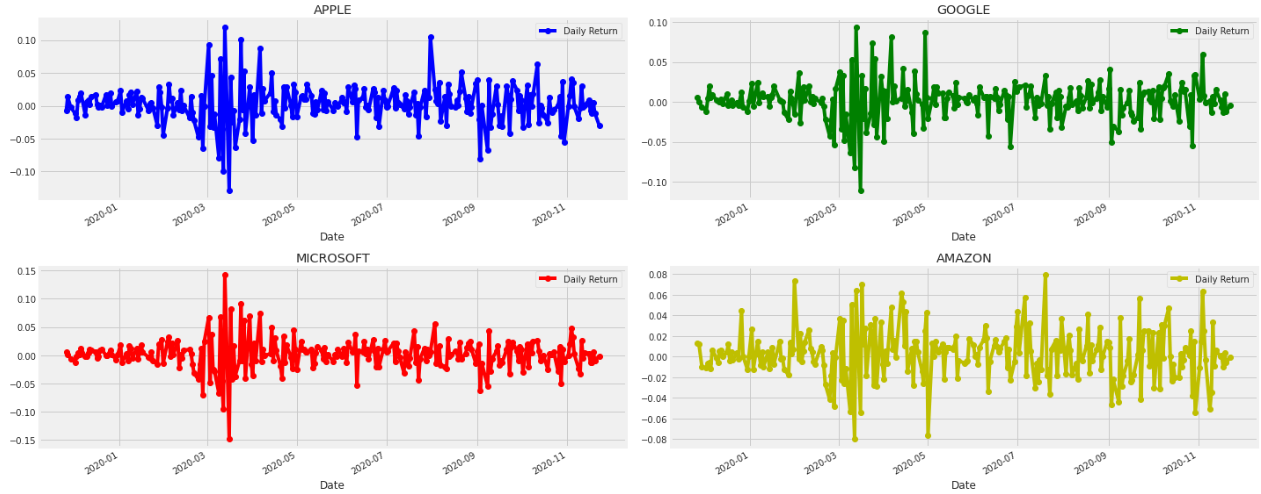
Moving Average of various stocks

[ ] ↳ *3 cells hidden*

Daily average return of stocks

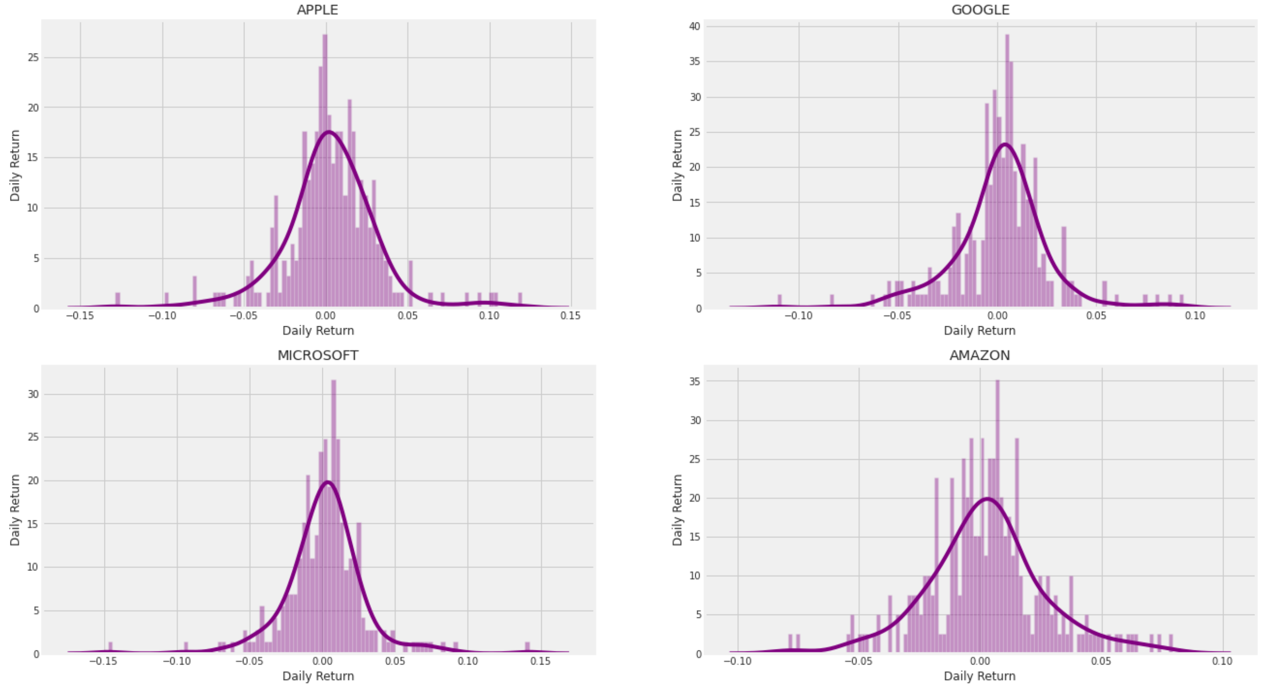
```
# We'll use pct_change to find the percent change for each day
for company in company_list:
    company['Daily Return'] = company['Adj Close'].pct_change()

# Then we'll plot the daily return percentage
fig, axes = plt.subplots(nrows=2, ncols=2)
fig.set_figheight(8)
fig.set_figwidth(20)
```

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

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

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

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

fig.tight_layout()
```




```
plt.figure(figsize=(20, 12))

for i, company in enumerate(company_list, 1):
    plt.subplot(2, 2, i)
    sns.distplot(company['Daily Return'].dropna(), bins=100, color='purple')
    plt.ylabel('Daily Return')
    plt.title(f'{company_name[i - 1]}')
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2551: FutureWarnin
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2551: FutureWarnin
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2551: FutureWarnin
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2551: FutureWarnin
  warnings.warn(msg, FutureWarning)
```



Correlation between stocks

```
closing_df = DataReader(tech_list, 'yahoo', start, end)['Adj Close']
closing_df.head()
```

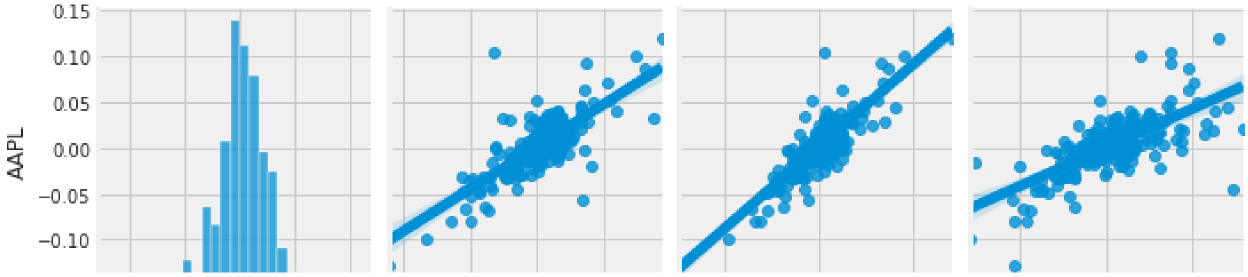
Symbols	AAPL	GOOG	MSFT	AMZN
Date				
2019-11-25	65.486168	1306.689941	149.644714	1773.839966

```
tech_rets = closing_df.pct_change()
tech_rets.head()
```

Symbols	AAPL	GOOG	MSFT	AMZN
Date				
2019-11-25	NaN	NaN	NaN	NaN
2019-11-26	-0.007809	0.005250	0.005290	0.013023
2019-11-27	0.013432	-0.000426	0.001907	0.012004
2019-11-29	-0.002203	-0.006116	-0.006171	-0.009739
2019-12-02	-0.011562	-0.011525	-0.012089	-0.010662

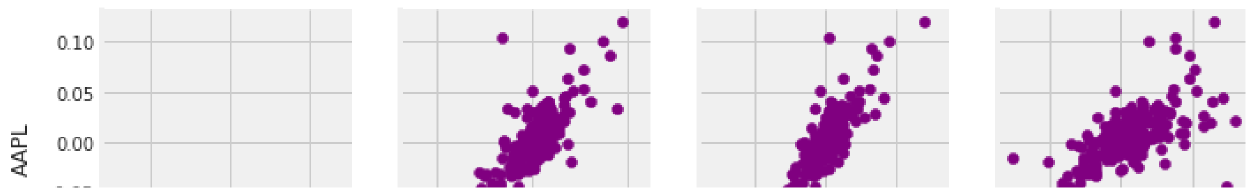
```
sns.pairplot(tech_rets, kind='reg')
```

```
<seaborn.axisgrid.PairGrid at 0x7f09bbb2a668>
```



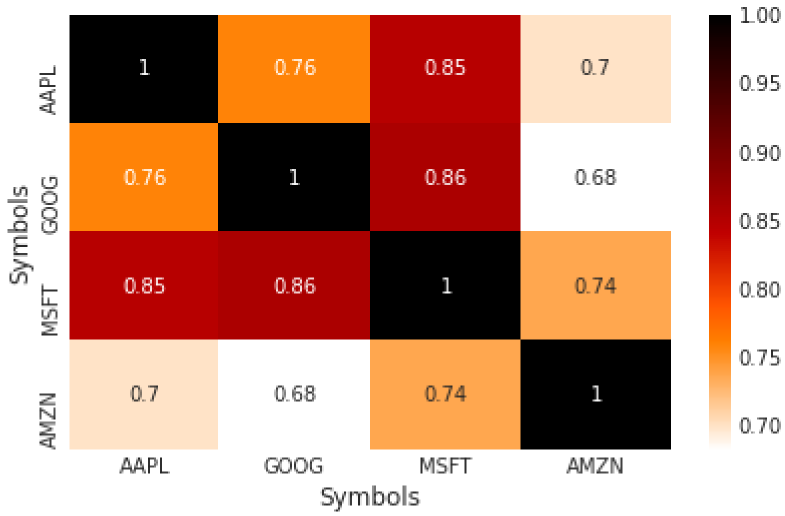

```
# Set up our figure by naming it returns_fig, call PairPLot on the DataFrame
return_fig = sns.PairGrid(tech_rets.dropna())

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

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

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

```
<seaborn.axisgrid.PairGrid at 0x7f09bb3df208>
```



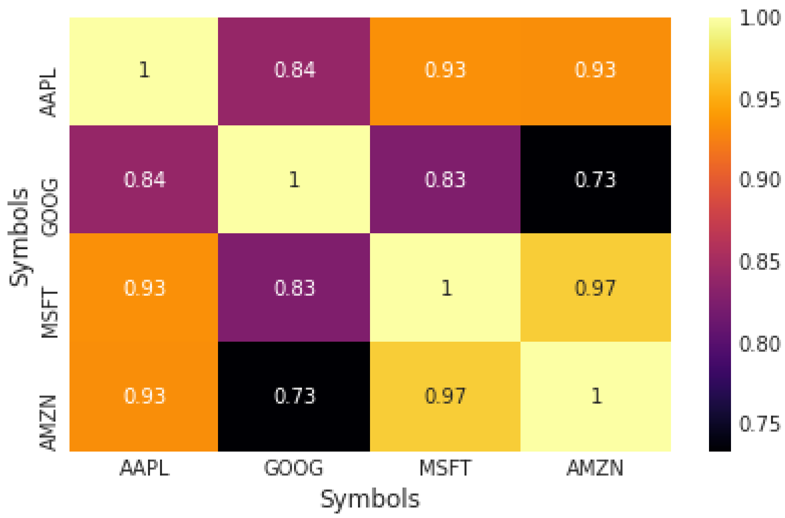

```
sns.heatmap(tech_rets.corr(), annot=True, cmap='gist_heat_r')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f09baf53c88>
```




```
sns.heatmap(closing_df.corr(), annot=True, cmap='inferno')
```

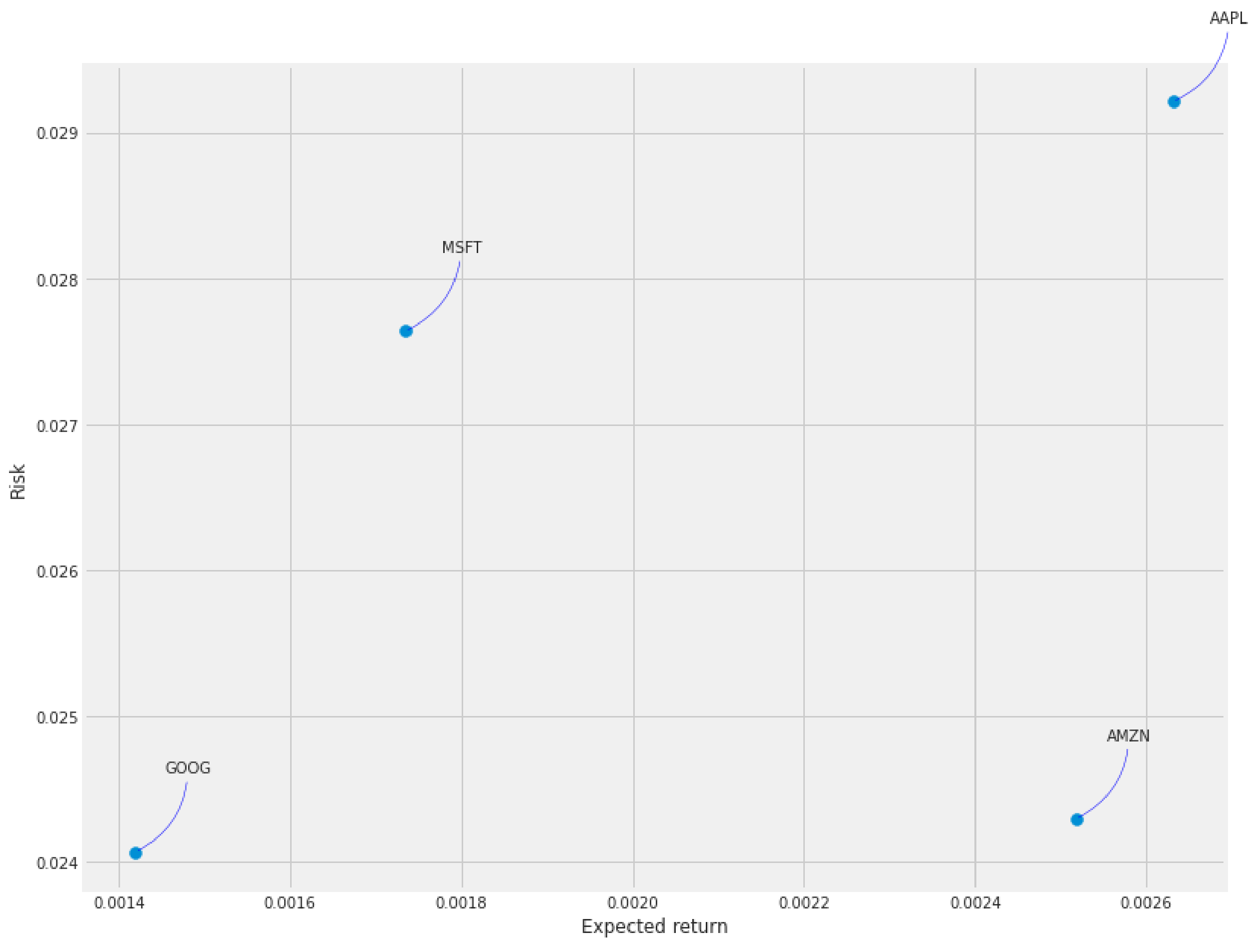
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f09bae82278>
```



Risk on a particular stock

```
rets = tech_rets.dropna()

area = np.pi*20

plt.figure(figsize=(12, 10))
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', ha='right'
                 arrowprops=dict(arrowstyle='-', color='blue', connectionstyle='arc3,rad=-
```




```
#Get the stock quote
df = DataReader('AAPL', data_source='yahoo', start='2012-01-01', end=datetime.now())
#Show teh data
df
```

	High	Low	Open	Close	Volume	Adj Close
Date						
2012-01-03	14.732142	14.607142	14.621428	14.686786	302220800.0	12.566676
2012-01-04	14.810000	14.617143	14.642858	14.765715	260022000.0	12.634213
2012-01-05	14.948215	14.738214	14.819643	14.929643	271269600.0	12.774481
2012-01-06	15.098214	14.972143	14.991786	15.085714	318292800.0	12.908023
2012-01-09	15.276786	15.048214	15.196428	15.061786	394024400.0	12.887549
...	...	...	...	...	...	...
2020-11-17	120.669998	118.959999	119.550003	119.389999	74271000.0	119.389999

```
plt.figure(figsize=(16,8))
plt.title('Close Price History')
plt.plot(df['Close'])
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.show()
```


```
#Create a new dataframe with only the 'Close column
data = df.filter(['Close'])
#Convert the dataframe to a numpy array
dataset = data.values
#Get the number of rows to train the model on
```

```
#Get the number of rows to train the model on
training_data_len = int(np.ceil( len(dataset) * .8 ))

training_data_len
```

```
1792
```

```
#Scale the data
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)

scaled_data
```

```
array([[0.0061488 ],
       [0.00680527],
       [0.00816869],
       ...,
       [0.87075047],
       [0.85993806],
       [0.83091098]])
```

```
#Create the training data set
#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)

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

```
[array([0.0061488 , 0.00680527, 0.00816869, 0.00946678, 0.00926776,
       0.00971629, 0.00951133, 0.00916676, 0.00869744, 0.01014998,
       0.01145994, 0.01105596, 0.00884299, 0.01095496, 0.00887566,
       0.01667305, 0.01607005, 0.01685722, 0.01855928, 0.01959001,
       0.01950387, 0.01918604, 0.02054056, 0.02181487, 0.02325851,
       0.0255903 , 0.03048855, 0.03056281, 0.03328967, 0.03532738,
       0.03182524, 0.03317382, 0.03314709, 0.03692846, 0.0363908 ,
       0.03738589, 0.0391741 , 0.0401692 , 0.04303567, 0.04512389,
       0.04572687, 0.04593778, 0.04236733, 0.04150589, 0.04163362,
       0.04499021, 0.04593482, 0.04796361, 0.05274602, 0.05912652,
       0.0579324 , 0.05793537, 0.06254846, 0.06399208, 0.06296431,
```

```
       0.06202567, 0.06104839, 0.06429507, 0.06652291, 0.06745562])]
[0.06515055661523342]

[array([0.0061488 , 0.00680527, 0.00816869, 0.00946678, 0.00926776,
       0.00971629, 0.00951133, 0.00916676, 0.00869744, 0.01014998,
       0.01145994, 0.01105596, 0.00884299, 0.01095496, 0.00887566,
       0.01667305, 0.01607005, 0.01685722, 0.01855928, 0.01959001,
       0.01950387, 0.01918604, 0.02054056, 0.02181487, 0.02325851,
       0.0255903 , 0.03048855, 0.03056281, 0.03328967, 0.03532738,
       0.03182524, 0.03317382, 0.03314709, 0.03692846, 0.0363908 ,
       0.03738589, 0.0391741 , 0.0401692 , 0.04303567, 0.04512389,
       0.04572687, 0.04593778, 0.04236733, 0.04150589, 0.04163362,
       0.04499021, 0.04593482, 0.04796361, 0.05274602, 0.05912652,
       0.0579324 , 0.05793537, 0.06254846, 0.06399208, 0.06296431,
       0.06202567, 0.06104839, 0.06429507, 0.06652291, 0.06745562]), array([0.0068052
       0.00951133, 0.00916676, 0.00869744, 0.01014998, 0.01145994,
       0.01105596, 0.00884299, 0.01095496, 0.00887566, 0.01667305,
       0.01607005, 0.01685722, 0.01855928, 0.01959001, 0.01950387,
       0.01918604, 0.02054056, 0.02181487, 0.02325851, 0.0255903 ,
       0.03048855, 0.03056281, 0.03328967, 0.03532738, 0.03182524,
       0.03317382, 0.03314709, 0.03692846, 0.0363908 , 0.03738589,
       0.0391741 , 0.0401692 , 0.04303567, 0.04512389, 0.04572687,
       0.04593778, 0.04236733, 0.04150589, 0.04163362, 0.04499021,
       0.04593482, 0.04796361, 0.05274602, 0.05912652, 0.0579324 ,
       0.05793537, 0.06254846, 0.06399208, 0.06296431, 0.06202567,
       0.06104839, 0.06429507, 0.06652291, 0.06745562, 0.06515056])]
[0.06515055661523342, 0.062088042929699744]
```

Stock portfolio prediction using LSTM

```
from keras.models import Sequential
from keras.layers import Dense, LSTM

#Build the LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape= (x_train.shape[1], 1)))
model.add(LSTM(50, 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)
```

```
1732/1732 [==============================] - 38s 22ms/step - loss: 3.5105e-04
<tensorflow.python.keras.callbacks.History at 0x7f098465c780>
```

```
#Create the testing data set
#Create a new array containing scaled values from index 1543 to 2002
test_data = scaled_data[training_data_len - 60: , :]
```

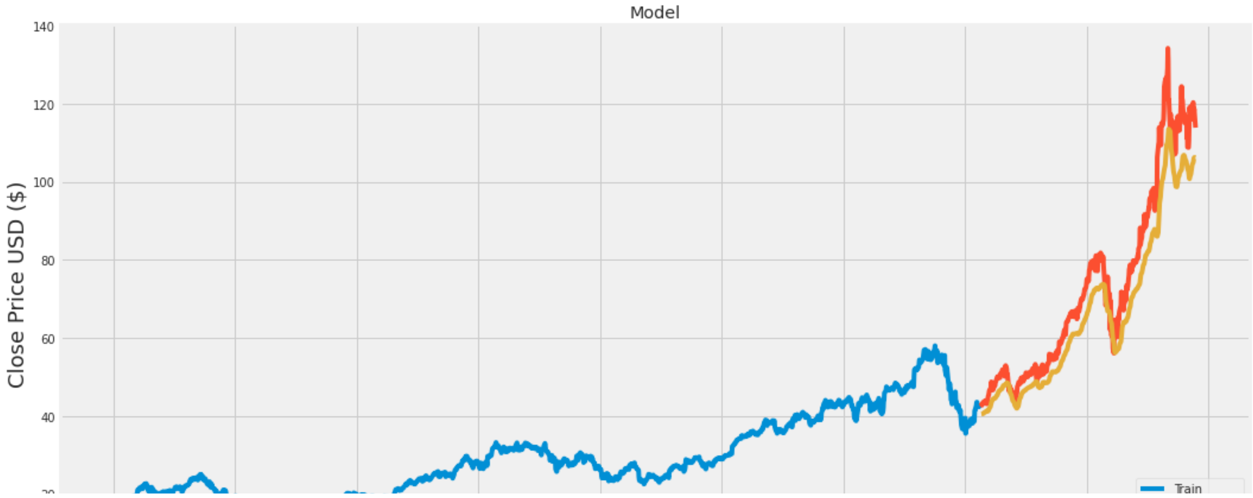
```
#Create the data sets x_test and y_test
x_test = []
y_test = dataset[training_data_len:, :]
for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 0])

# 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
```

```
8.146286565775492
```

Visualizing Stock Predictions

```
# Plot the data
train = data[:training_data_len]
valid = data[training_data_len:]
valid['Predictions'] = predictions
# Visualize the data
plt.figure(figsize=(16,8))
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()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: SettingWithCopyWar
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
  after removing the cwd from sys.path.
```




```
valid
```

Date	Close	Predictions
2019-02-19	42.732498	40.538265
2019-02-20	43.007500	40.543148
2019-02-21	42.764999	40.567841
2019-02-22	43.242500	40.580837
2019-02-25	43.557499	40.626148
...	...	...
2020-11-17	119.389999	105.517380
2020-11-18	118.029999	105.951279
2020-11-19	118.639999	106.144829
2020-11-20	117.339996	106.254135
2020-11-23	113.849998	106.179993

447 rows × 2 columns

Conclusion :

Predicted results resemble actual values to a good extent. Hence the model is successfully executed with near accurate expectancy.