Fatima Jinnah Women University

Opening Portals of Excellence Through Higher Education

Artificial Intelligence

Submitted to: Dr. Aamir Arsalan

Submitted by: Ansa Khalid(2020-BSE-42)

Bisma(2020-BSE-46)

Zaupash Tariq(2020-BSE-69)

Semester: VII

Contents

Introduction

In our project, we're exploring stock market data analysis, looking closely at major tech stocks like Apple, Amazon, Google, and Microsoft. We will utilize yfinance, a powerful Python library, to meticulously retrieve and analyze stock information, complemented by insightful visualizations through Seaborn and Matplotlib. Beyond surface-level observations, our analysis incorporates a rigorous examination of risk by scrutinizing the historical performance of these tech giants, employing statistical methods and visualizations to discern intricate patterns. Taking the analysis further, we leverage Long Short Term Memory (LSTM) networks for predictive modeling, enabling the forecast of future stock prices. By the project's conclusion, you will gain a nuanced understanding of past stock performances and the skills to navigate the intricate landscape of the stock market, offering valuable insights and predictions for strategic decision-making.

Analytical Focus Points

1. What was the change in price of the stock over time?
2. What was the daily return of the stock on average?
3. What was the moving average of the various stocks?
4. What was the correlation between different stocks'?
5. How much value do we put at risk by investing in a particular stock?
6. How can we attempt to predict future stock behavior? (Predicting the closing price stock price of APPLE inc using LSTM)

Getting the Data

The first step is to get the data and load it to memory. We will get our stock data from the Yahoo Finance website. Yahoo Finance is a rich resource of financial market data and tools to find compelling investments. To get the data from Yahoo Finance, we will be using yfinance library which offers a threaded and Pythonic way to download market data from Yahoo.

1. What was the change in price of the stock overtime?

In this section, we'll cover how to handle requesting stock information with pandas, and how to analyze basic attributes of a stock.

```
!pip install -q yfinance
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
import yfinance as yf
from pandas_datareader import data as pdr

yf.pdr_override()
# For time stamps
from datetime import datetime
# The tech stocks we'll use for this analysis
```

```python
tech_list = ['AAPL', 'GOOG', 'MSFT', 'AMZN']
# Set up End and Start times for data grab
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
```

	Open	High	Low	Close	Adj Close	Volume	company_name
Date							
2023-12-22	153.770004	154.350006	152.710007	153.419998	153.419998	29480100	AMAZON
2023-12-26	153.559998	153.979996	153.029999	153.410004	153.410004	25067200	AMAZON
2023-12-27	153.559998	154.779999	153.119995	153.339996	153.339996	31434700	AMAZON
2023-12-28	153.720001	154.080002	152.949997	153.380005	153.380005	27057000	AMAZON
2023-12-29	153.100006	153.889999	151.029999	151.940002	151.940002	39789000	AMAZON
2024-01-02	151.539993	152.380005	148.389999	149.929993	149.929993	47339400	AMAZON
2024-01-03	149.199997	151.050003	148.330002	148.470001	148.470001	49425500	AMAZON
2024-01-04	145.589996	147.380005	144.050003	144.570007	144.570007	56039800	AMAZON
2024-01-05	144.690002	146.589996	144.529999	145.240005	145.240005	45124800	AMAZON
2024-01-08	146.740005	149.399994	146.149994	149.100006	149.100006	46711600	AMAZON

our data, we can see that the data is numeric and the date is the index of the data. Notice also that weekends are missing from the records.

Descriptive Statistics about the Data:

`.describe()` generates descriptive statistics. Descriptive statistics include those that summarize the central tendency, dispersion, and shape of a dataset's distribution, excluding `NaN` values.

Analyzes both numeric and object series, as well as `DataFrame` column sets of mixed data types. The output will vary depending on what is provided. Refer to the notes below for more detail.

```python
# Summary Stats
AAPL.describe()
```

	Open	High	Low	Close	Adj Close	Volume
count	251.000000	251.000000	251.000000	251.000000	251.000000	2.510000e+02
mean	173.193068	174.790956	171.938008	173.504861	173.089883	5.883899e+07
std	16.608415	16.336289	16.539447	16.336108	16.511069	1.724662e+07
min	130.259995	131.259995	128.119995	130.149994	129.426559	2.404830e+07
25%	162.750000	165.165001	161.955002	164.215004	163.551559	4.790395e+07
50%	176.380005	177.679993	174.800003	176.080002	175.848328	5.520920e+07
75%	186.989998	188.250000	185.119995	187.220001	186.952606	6.552400e+07
max	198.020004	199.619995	197.000000	198.110001	198.110001	1.543573e+08

Information About the Data:

`.info()` method prints information about a DataFrame including the index `dtype` and columns, non-null values, and memory usage.

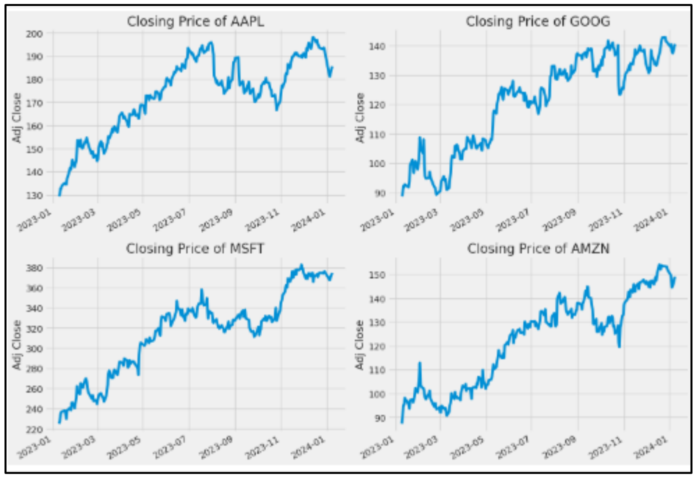
```
# General info
AAPL.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 251 entries, 2023-01-09 to 2024-01-08
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Open          251 non-null    float64
 1   High          251 non-null    float64
 2   Low           251 non-null    float64
 3   Close         251 non-null    float64
 4   Adj Close     251 non-null    float64
 5   Volume        251 non-null    int64
 6   company_name  251 non-null    object
dtypes: float64(5), int64(1), object(1)
memory usage: 15.7+ KB
```

Closing Price:

The closing price is the last price at which the stock is traded during the regular trading day. A stock's closing price is the standard benchmark used by investors to track its performance over time.

```
# Let's see a historical view of the closing price
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()
```



Volume of Sales:

Volume is the amount of an asset or security that changes hands over some period of time, often over the course of a day. For instance, the stock trading volume would refer to the number of shares of security traded between its daily open and close. Trading volume, and changes to volume over the course of time, are important inputs for technical traders.

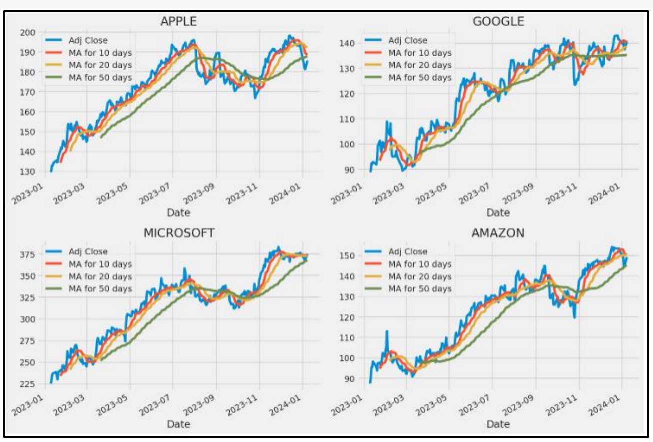
```
# Now let's plot the total volume of stock being traded each day
plt.figure(figsize=(15, 10))
plt.subplots_adjust(top=1.25, bottom=1.2)

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()
```

The visualizations for the closing price and the volume traded each day, let's go ahead and caculate the moving average for the stock.

2. What was the moving average of the various stocks?

The moving average (MA) is a simple technical analysis tool that smooths out price data by creating a constantly updated average price. The average is taken over a specific period of time, like 10 days, 20 minutes, 30 weeks, or any time period the trader chooses.

```
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(ax=axes[0,0])
axes[0,0].set_title('APPLE')
GOOG[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(ax=axes[0,1])
axes[0,1].set_title('GOOGLE')
MSFT[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(ax=axes[1,0])
axes[1,0].set_title('MICROSOFT')

AMZN[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(ax=axes[1,1])
axes[1,1].set_title('AMAZON')
fig.tight_layout()
```

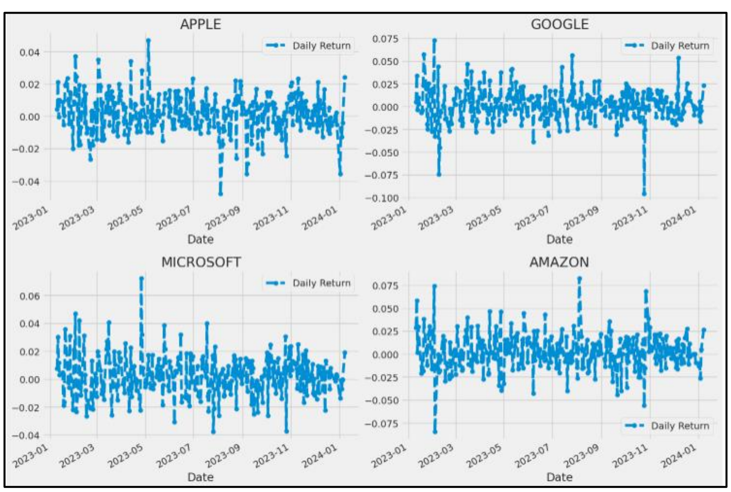


We see in the graph that the best values to measure the moving average are 10 and 20 days because we still capture trends in the data without noise.

3. What was the daily return of the stock on average?

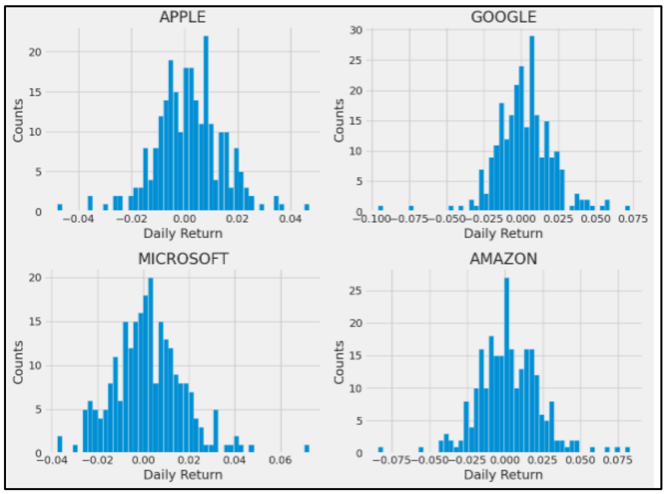
Now that we've done some baseline analysis, let's go ahead and dive a little deeper. We're now going to analyze the risk of the stock. In order to do so we'll need to take a closer look at the daily changes of the stock, and not just its absolute value. Let's go ahead and use pandas to retrieve the daily returns for the Apple stock.

```
# 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(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()
```



We'll use seaborn to create both a histogram and kde plot on the same figure.

```python
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()
```



4. What was the correlation between different stocks closing prices?

Correlation is a statistic that measures the degree to which two variables move in relation to each other which has a value that must fall between -1.0 and +1.0. Correlation measures association, but doesn't show if x causes y or vice versa — or if the association is caused by a third factor[1].

Now what if we wanted to analyze the returns of all the stocks in our list? Let's go ahead and build a DataFrame with all the ['Close'] columns for each of the stocks dataframes.

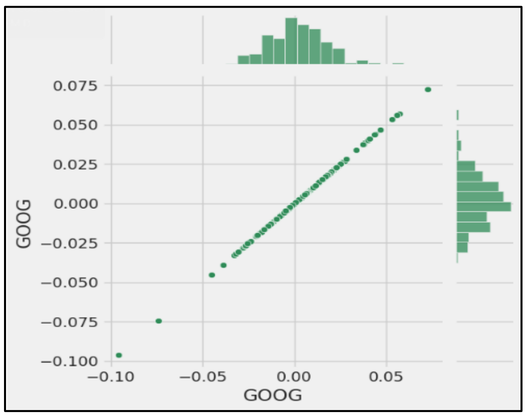
```python
# Grab all the closing prices for the tech stock list into one DataFrame

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

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

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

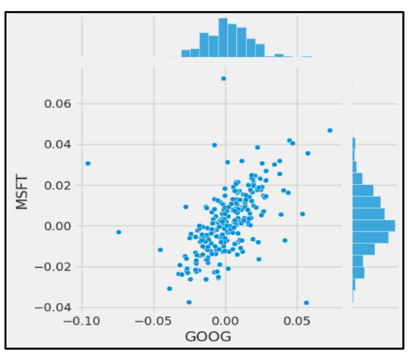
Date	AAPL	AMZN	GOOG	MSFT
2023-01-09	NaN	NaN	NaN	NaN
2023-01-10	0.004456	0.028732	0.004955	0.007617
2023-01-11	0.021112	0.058084	0.033841	0.030238
2023-01-12	-0.000599	0.001893	-0.003794	0.011621
2023-01-13	0.010119	0.029915	0.009683	0.003019

Now we can compare the daily percentage return of two stocks to check how correlated. First let's see a stock compared to itself.

```
# Comparing Google to itself should show a perfectly linear relationship
sns.jointplot(x='GOOG', y='GOOG', data=tech_rets, kind='scatter', color='seagreen')
```



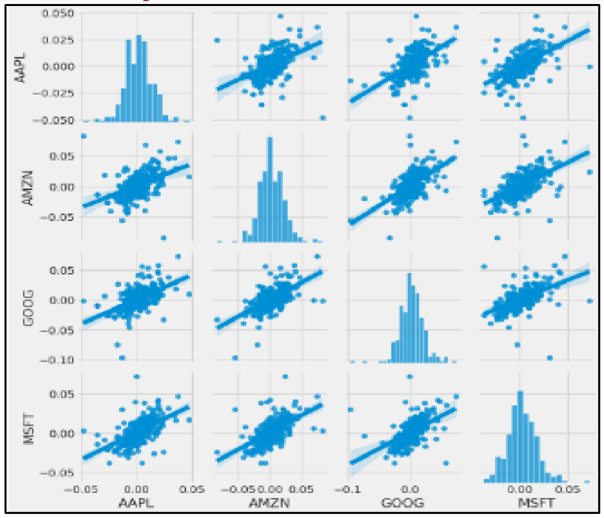

```
# We'll use joinplot to compare the daily returns of Google and Microsoft
sns.jointplot(x='GOOG', y='MSFT', data=tech_rets, kind='scatter')
```



So now we can see that if two stocks are perfectly (and positivley) correlated with each other a linear relationship bewteen its daily return values should occur.

Seaborn and pandas make it very easy to repeat this comparison analysis for every possible combination of stocks in our technology stock ticker list. We can use sns.pairplot() to automatically create this plot.

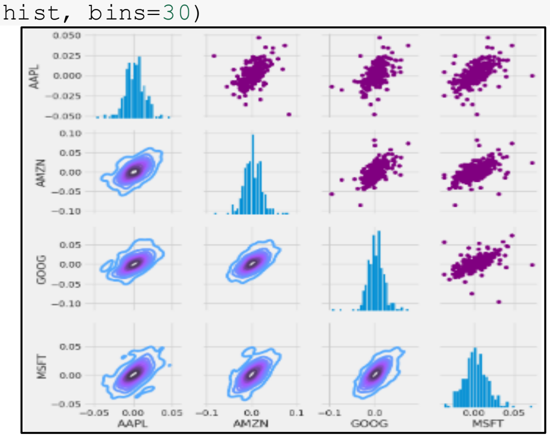
```
# We can simply call pairplot on our DataFrame for an automatic visual analysis
# of all the comparisons
sns.pairplot(tech_rets, kind='reg')
```



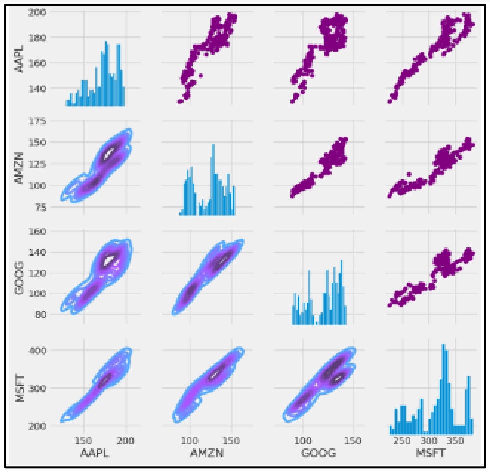
Above we can see all the relationships on daily returns between all the stocks. A quick glance shows an interesting correlation between Google and Amazon daily returns. It might be interesting to investigate that individual comaprison.

While the simplicity of just calling `sns.pairplot()` is fantastic we can also use `sns.PairGrid()` for full control of the figure, including what kind of plots go in the diagonal, the upper triangle, and the lower triangle. Below is an example of utilizing the full power of seaborn to achieve this result.

```
# 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)
```

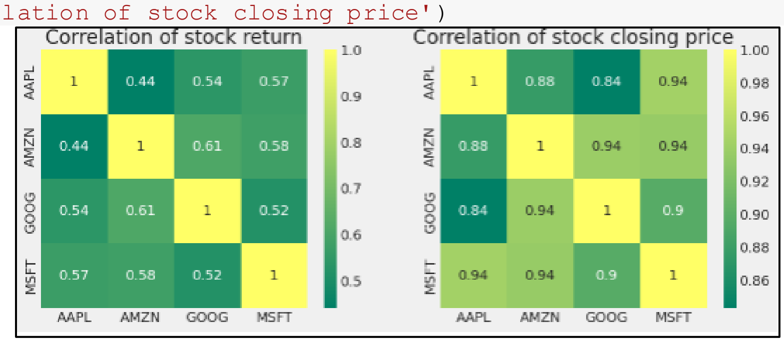


```
# Set up our figure by naming it returns_fig, call PairPLot on the DataFrame
returns_fig = sns.PairGrid(closing_df)
# Using map upper we can specify what the upper triangle will look like.
returns_fig.map_upper(plt.scatter,color='purple')
# We can also define the lower triangle in the figure, incluging the plot type (kde)
or the color map (BluePurple)
returns_fig.map_lower(sns.kdeplot,cmap='cool_d')
# Finally we'll define the diagonal as a series of histogram plots of the daily return
returns_fig.map_diag(plt.hist,bins=30)
```



we could also do a correlation plot, to get actual numerical values for the correlation between the stocks' daily return values. By comparing the closing prices, we see an interesting relationship between Microsoft and Apple.

```
plt.figure(figsize=(12, 10))
plt.subplot(2, 2, 1)
sns.heatmap(tech_rets.corr(), annot=True, cmap='summer')
plt.title('Correlation of stock return')
plt.subplot(2, 2, 2)
sns.heatmap(closing_df.corr(), annot=True, cmap='summer')
plt.title('Correlation of stock closing price')
```

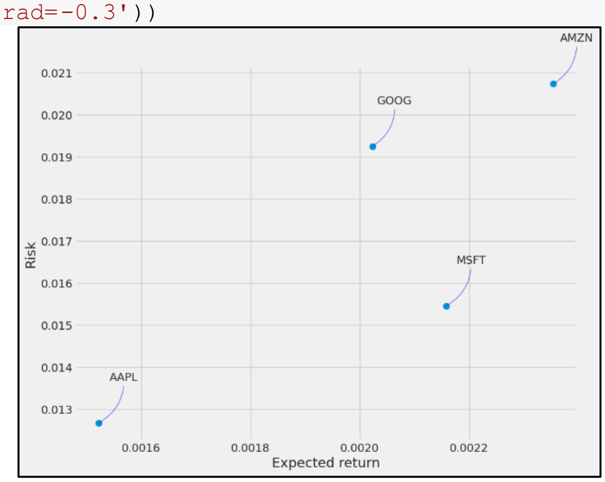


we suspected in our `PairPlot` we see here numerically and visually that Google and Amazon had the strongest correlation of daily stock return. It's also interesting to see that all the technology companies are positively correlated. Microsoft and Amazon had the strongest correlation of daily stock closing price.

5. How much value do we put at risk by investing in a particular stock?

There are many ways we can quantify risk, one of the most basic ways using the information we've gathered on daily percentage returns is by comparing the expected return with the standard deviation of the daily returns.

```
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',
ha='right', va='bottom',
                 arrowprops=dict(arrowstyle='-', color='blue',
connectionstyle='arc3,rad=-0.3'))
```



6. Predicting the closing price stock price of APPLE inc?

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

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

	Open	High	Low	Close	Adj Close	Volume
Date						
2012-01-03	14.621429	14.732143	14.607143	14.686786	12.449691	302220800
2012-01-04	14.642857	14.810000	14.617143	14.765714	12.516595	260022000
2012-01-05	14.819643	14.948214	14.738214	14.929643	12.655556	271269600
2012-01-06	14.991786	15.098214	14.972143	15.085714	12.787854	318292800
2012-01-09	15.196429	15.276786	15.048214	15.061786	12.767571	394024400
...	...	...	...	...	...	...
2024-01-02	187.149994	188.440002	183.889999	185.639999	185.639999	82488700
2024-01-03	184.220001	185.880005	183.429993	184.250000	184.250000	58414500
2024-01-04	182.149994	183.089996	180.880005	181.910004	181.910004	71983600
2024-01-05	181.990005	182.759995	180.169998	181.179993	181.179993	62303300
2024-01-08	182.089996	185.600006	181.500000	185.559998	185.559998	59087900

3023 rows × 6 columns

```
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()
```


```
# 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
training_data_len = int(np.ceil( len(dataset) * .95 ))

training_data_len
```

```
2872
```

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

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

scaled_data
```

```
array([[0.00401431],
       [0.00444289],
       [0.00533302],
       ...,
       [0.91203423],
       [0.90807028],
       [0.93185365]])
```

```
from keras.models import Sequential
from keras.layers import Dense, LSTM
from keras.callbacks import EarlyStopping
from keras.layers import Dropout
from keras.regularizers import l2
```

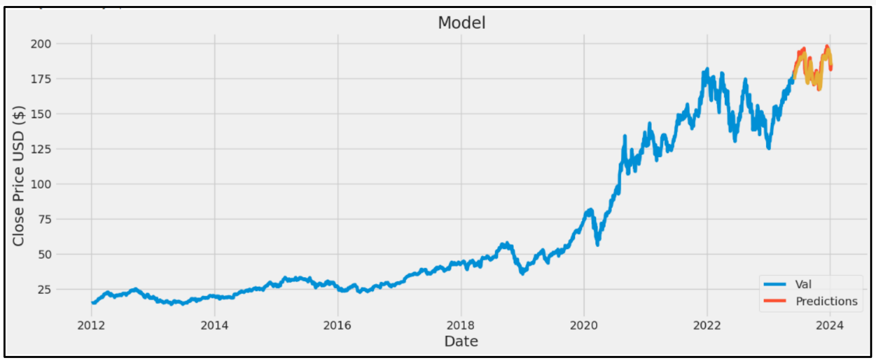
```python
from keras.optimizers import Adam
# Build the LSTM model
model = Sequential()
model.add(LSTM(64, input_shape=(x_train.shape[1], 1), kernel_regularizer=l2(0.01)))
model.add(Dropout(0.2))
model.add(Dense(25))
model.add(Dense(1))
# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')
# Early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)
# Train the model with early stopping
model.fit(x_train, y_train, batch_size=1, epochs=10, validation_data=(x_val, y_val),
callbacks=[early_stopping])
```

```
Epoch 1/10
2237/2237 [==============================] - 40s 17ms/step - loss: 0.0028 - val_loss: 0.0014
Epoch 2/10
2237/2237 [==============================] - 38s 17ms/step - loss: 0.0012 - val_loss: 0.0089
Epoch 3/10
2237/2237 [==============================] - 37s 17ms/step - loss: 0.0011 - val_loss: 0.0014
Epoch 4/10
2237/2237 [==============================] - 38s 17ms/step - loss: 9.0824e-04 - val_loss: 0.0025
Epoch 5/10
2237/2237 [==============================] - 36s 16ms/step - loss: 9.5174e-04 - val_loss: 8.9806e-04
Epoch 6/10
2237/2237 [==============================] - 37s 16ms/step - loss: 8.3182e-04 - val_loss: 0.0088
Epoch 7/10
2237/2237 [==============================] - 37s 16ms/step - loss: 8.3606e-04 - val_loss: 0.0012
Epoch 8/10
2237/2237 [==============================] - 36s 16ms/step - loss: 7.6207e-04 - val_loss: 0.0032
<keras.src.callbacks.History at 0x7fa69cf17760>
```

```python
# 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
```

```
5/5 [==============================] - 1s 14ms/step
4.84790003721448
```

```
# Plot the data
train = data[:training data len]
valid = data[training_data_len:]
# Plot the data
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)
# Comment out the training data plot
plt.plot(train['Close'])
plt.plot(valid[['Close', 'Predictions']])
plt.legend(['Val', 'Predictions'], loc='lower right')
plt.show()
```




```
# Show the valid and predicted prices
valid
```

Date	Close	Predictions
2023-06-02	180.949997	174.232208
2023-06-05	179.580002	175.382904
2023-06-06	179.210007	176.520813
2023-06-07	177.820007	177.545746
2023-06-08	180.570007	178.266708
...	...	...
2024-01-02	185.639999	191.879791
2024-01-03	184.250000	190.440109
2024-01-04	181.910004	188.663727
2024-01-05	181.179993	186.559891
2024-01-08	185.559998	184.336502

151 rows × 2 columns