Data Project - Stock Market Analysis

Time Series data is a series of data points indexed in time order. Time series data is everywhere, so manipulating them is important for any data analyst or data scientist.

In this notebook, we will discover and explore data from the stock market, particularly some technology stocks (Apple, Amazon, Google, and Microsoft). We will learn how to use yfinance to get stock information, and visualize different aspects of it using Seaborn and Matplotlib. we will look at a few ways of analyzing the risk of a stock, based on its previous performance history. We will also be predicting future stock prices through a Long Short Term Memory (LSTM) method!

We'll be answering the following questions along the way:

```
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? (Predicti
ng 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. Check this article to learn more about yfinance: [Reliably download historical market data from with Python (https://aroussi.com/post/python-yahoo-finance)](https://aroussi.com/post/python-yahoo-finance)

In [ ]:

```
```

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

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

In [3]:
```
import pandas as pd
import numpy as np
import pandas_datareader.data as pdr

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
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
```

Out[3]:

Date	Open	High	Low	Close	Adj Close	Volume	company_nam
2024-01-18	152.770004	153.779999	151.820007	153.500000	153.500000	37850200	AMAZON
2024-01-19	153.830002	155.759995	152.740005	155.339996	155.339996	51033700	AMAZON
2024-01-22	156.889999	157.050003	153.899994	154.779999	154.779999	43687500	AMAZON
2024-01-23	154.850006	156.210007	153.929993	156.020004	156.020004	37986000	AMAZON
2024-01-24	157.800003	158.509995	156.479996	156.869995	156.869995	48547300	AMAZON
2024-01-25	156.949997	158.509995	154.550003	157.750000	157.750000	43638600	AMAZON
2024-01-26	158.419998	160.720001	157.910004	159.119995	159.119995	51001100	AMAZON
2024-01-29	159.339996	161.289993	158.899994	161.259995	161.259995	45270400	AMAZON
2024-01-30	160.699997	161.729996	158.490005	159.000000	159.000000	44888800	AMAZON
2024-01-31	157.000000	159.009995	155.339996	156.259995	156.259995	20445600	AMAZON

Type *Markdown* and LaTeX: α^2

Reviewing the content of 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.

Quick note: Using `globals()` is a sloppy way of setting the `DataFrame` names, but it's simple. Now we have our data, let's perform some basic data analysis and check our data.

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.

In [4]:
```
# Summary Stats
AAPL.describe()
```

Out[4]:

	Open	High	Low	Close	Adj Close	Volume
count	252.000000	252.000000	252.000000	252.000000	252.000000	2.520000e+02
mean	176.364643	177.905814	175.085476	176.624946	176.257168	5.784672e+07
std	14.159222	13.931209	14.095954	13.943918	14.113869	1.739699e+07
min	142.699997	144.339996	141.320007	144.289993	143.487961	1.928864e+07
25%	168.095005	169.627506	166.540001	168.072502	167.535404	4.728925e+07
50%	177.610001	179.555000	176.560005	177.970001	177.635818	5.381205e+07
75%	189.100002	189.934998	187.509998	188.785004	188.659035	6.428920e+07
max	198.020004	199.619995	197.000000	198.110001	198.110001	1.543573e+08

We have only 255 records in one year because weekends are not included in the data.

Information About the Data

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

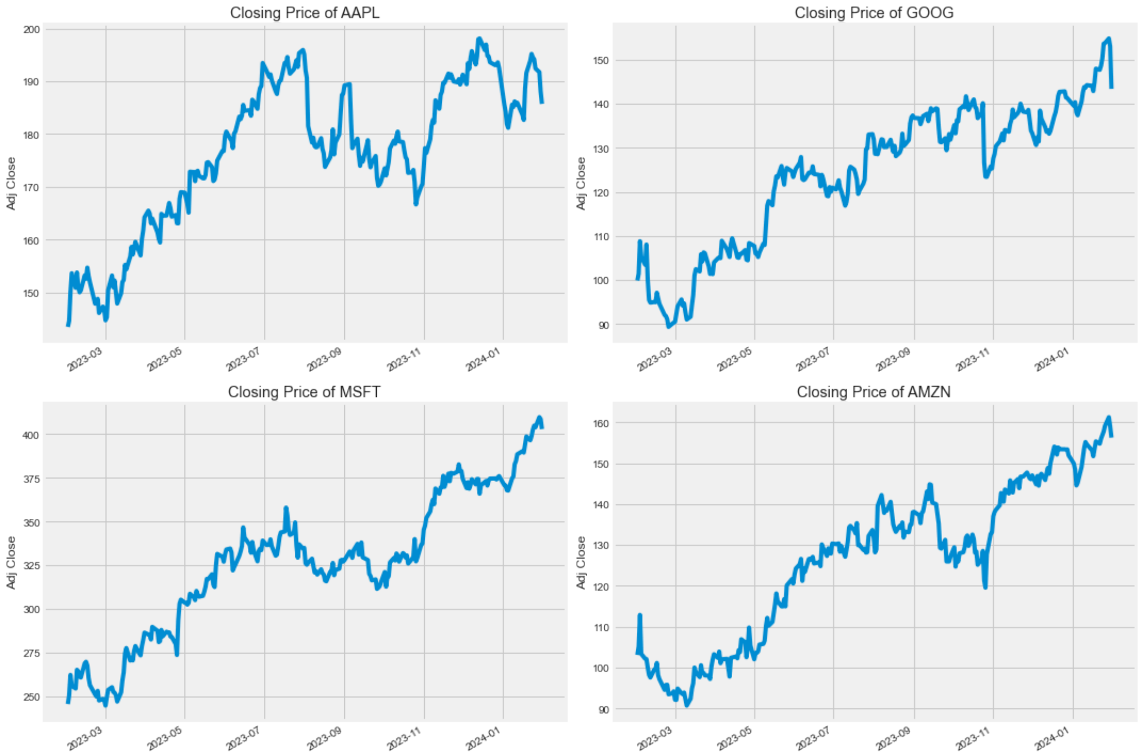
In [5]:
```
# General info
AAPL.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 252 entries, 2023-01-31 to 2024-01-31
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Open          252 non-null    float64
 1   High          252 non-null    float64
 2   Low           252 non-null    float64
 3   Close         252 non-null    float64
 4   Adj Close     252 non-null    float64
 5   Volume        252 non-null    int64  
 6   company_name  252 non-null    object 
dtypes: float64(5), int64(1), object(1)
memory usage: 15.8+ 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.

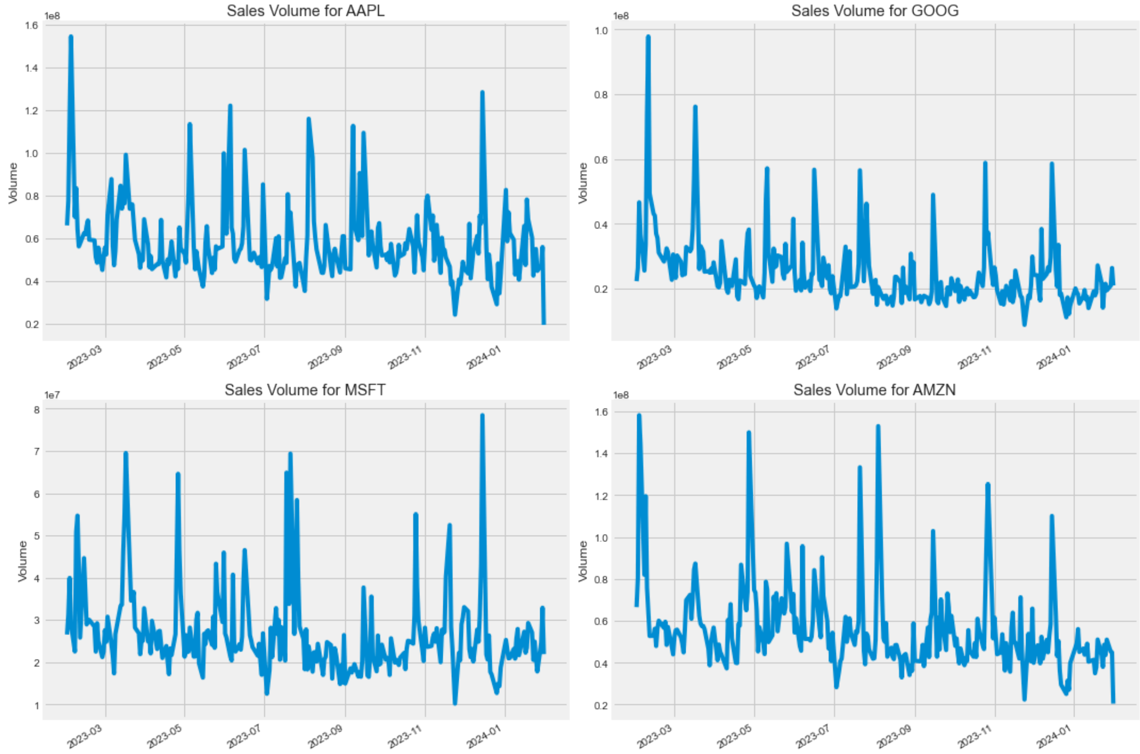
In [6]:
```
# Let's see a historical view of the closing price
plt.figure(figsize=(15, 10))
plt.subplots_adjust(top=1.25, bottom=1.2)

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.

In [7]:
```
# 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()
```



Now that we've seen 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.

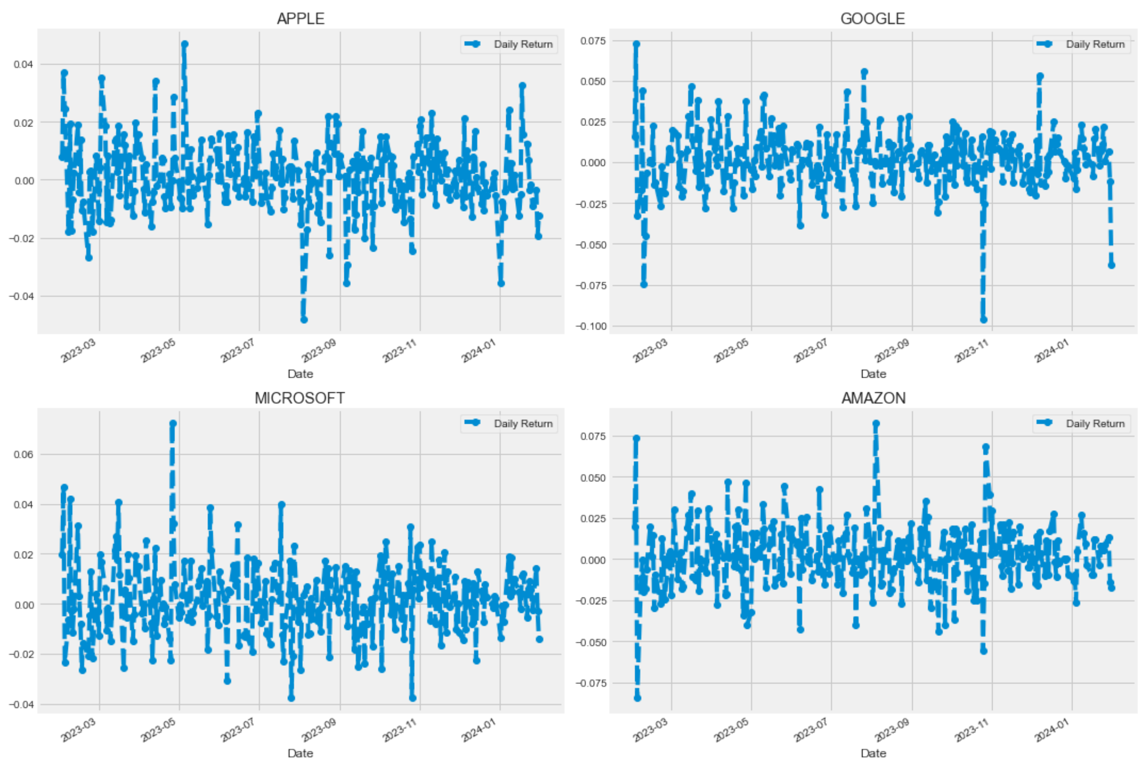
In [8]:
```
ma_day = [10, 20, 50]

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']].p
axes[0,0].set_title('APPLE')

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

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

AMZN[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].p
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?

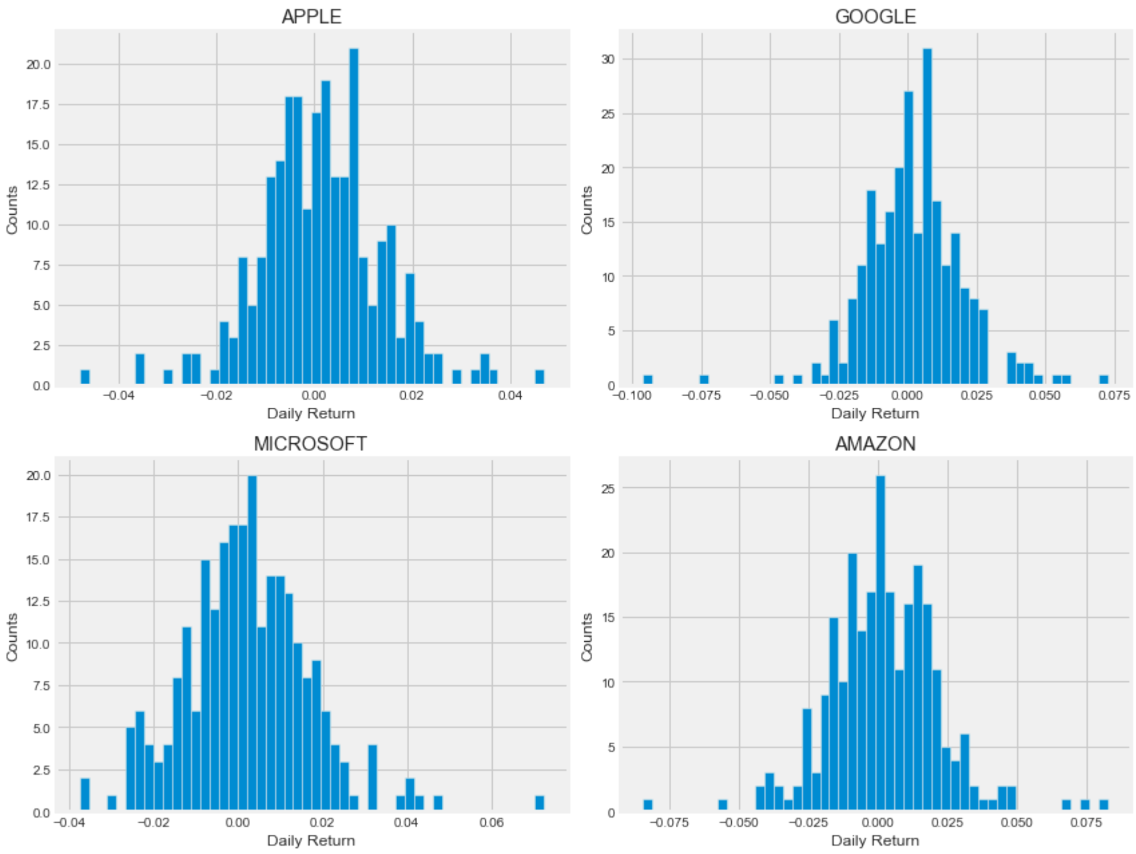
Now that we've done some baseline analysis, let's go ahead and dive a little deeper. We're now going to analyze the risk of the stock. In order to do so we'll need to take a closer look at the daily changes of the stock, and not just its absolute value. Let's go ahead and use pandas to retrieve teh daily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

In [9]:
```
# 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
axes[0,0].set_title('APPLE')

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

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

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

fig.tight_layout()
```



Great, now let's get an overall look at the average daily return using a histogram. We'll use seaborn to create both a histogram and kde plot on the same figure.

In [9]:
```
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.

In [10]:
```
# 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
```

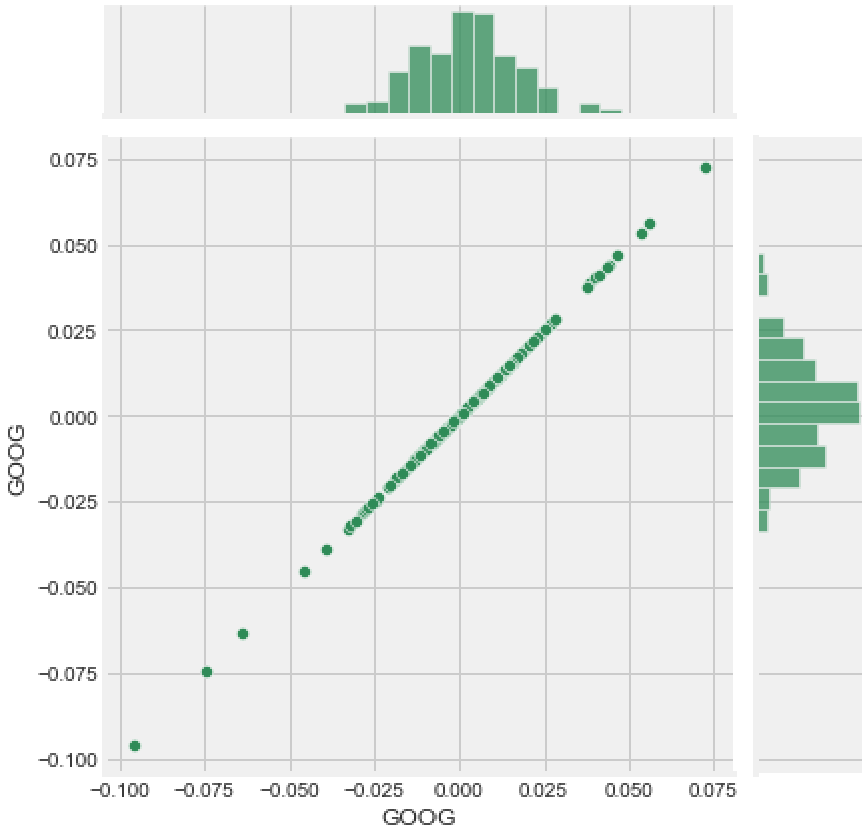
Out[10]:

Ticker	AAPL	AMZN	GOOG	MSFT
Date				
2023-01-31	NaN	NaN	NaN	NaN
2023-02-01	0.007901	0.019587	0.015620	0.019935
2023-02-02	0.037063	0.073799	0.072661	0.046884
2023-02-03	0.024400	-0.084315	-0.032904	-0.023621
2023-02-06	-0.017929	-0.011703	-0.016632	-0.006116

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

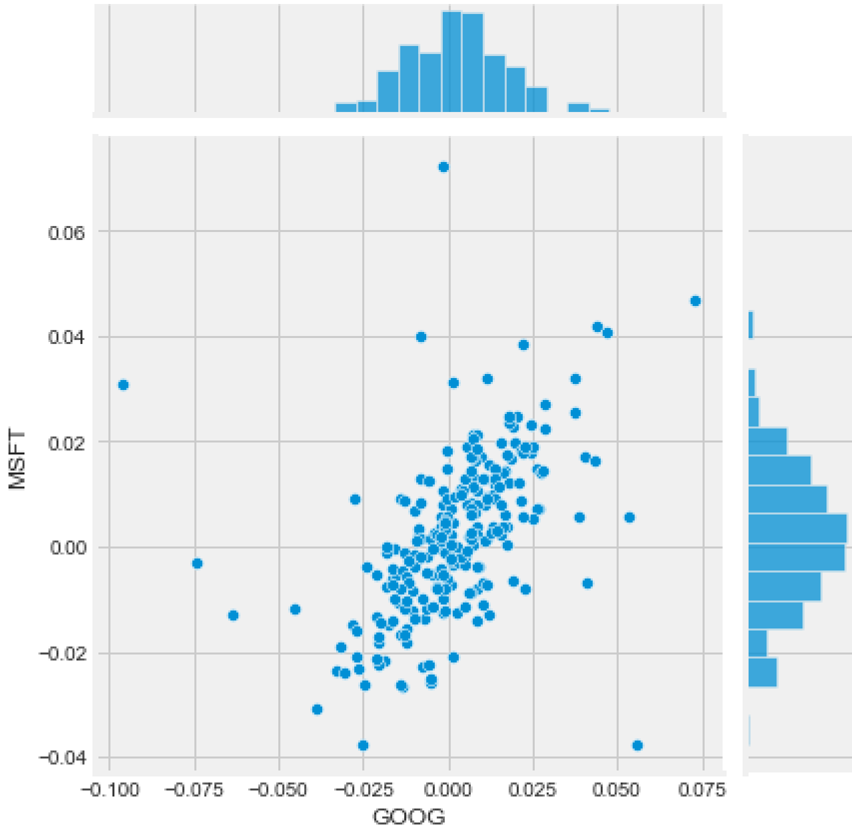
In [11]:
```
# Comparing Google to itself should show a perfectly linear relationship
sns.jointplot(x='GOOG', y='GOOG', data=tech_rets, kind='scatter', color='se
```

Out[11]: <seaborn.axisgrid.JointGrid at 0x1e16d75fdf0>



In [12]:
```
# We'll use joinplot to compare the daily returns of Google and Microsoft
sns.jointplot(x='GOOG', y='MSFT', data=tech_rets, kind='scatter')
```

Out[12]:
```
<seaborn.axisgrid.JointGrid at 0x1e16d41ccd0>
```



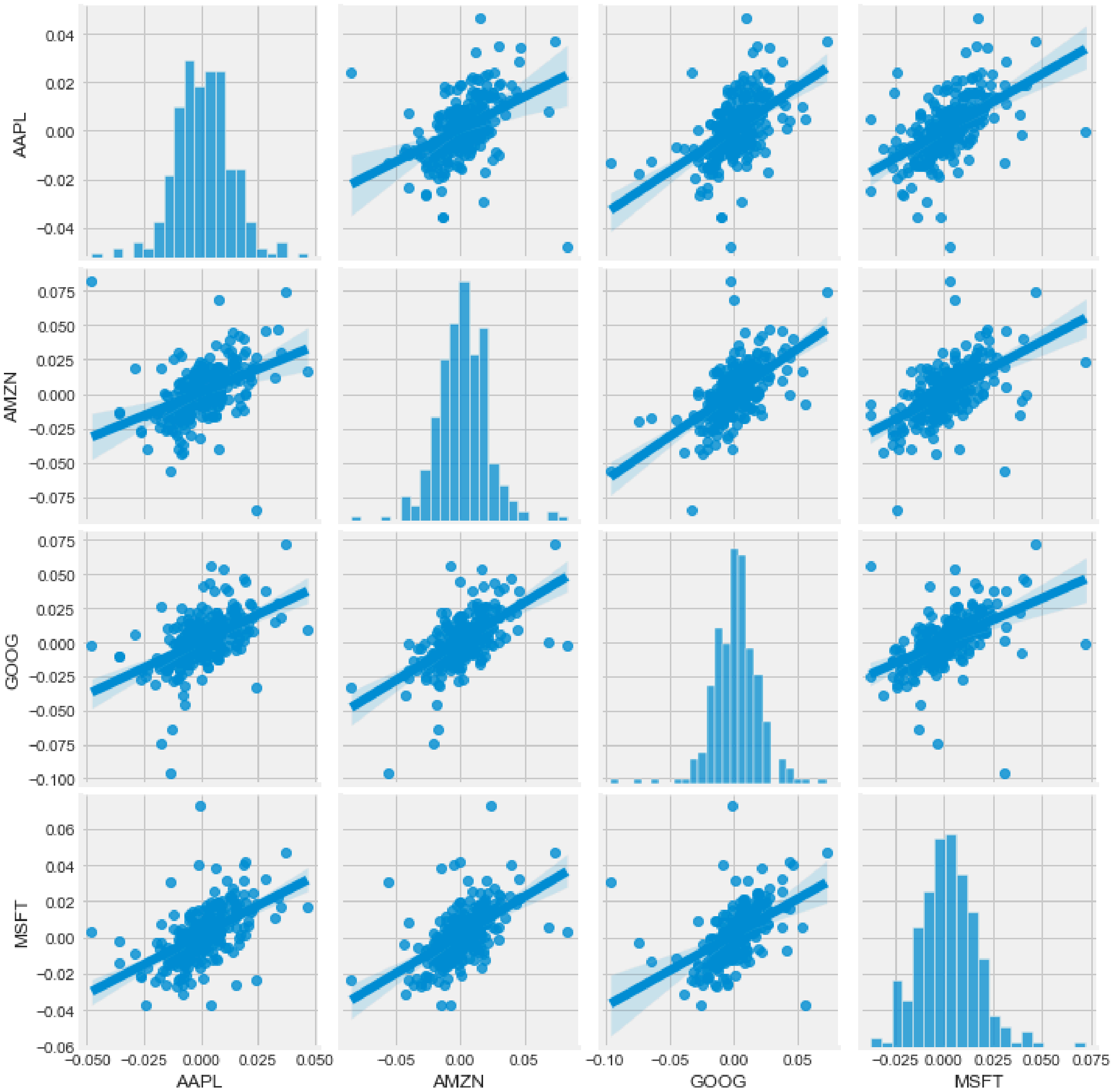
So now we can see that if two stocks are perfectly (and positivley) correlated with each other a linear relationship bewteen its daily return values should occur.

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

In [13]:
```
# We can simply call pairplot on our DataFrame for an automatic visual anal
# of all the comparisons

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

```
C:\Users\vasan\AppData\Roaming\Python\Python310\site-packages\seaborn\axis
grid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[13]:
```
<seaborn.axisgrid.PairGrid at 0x1e16d49f280>
```

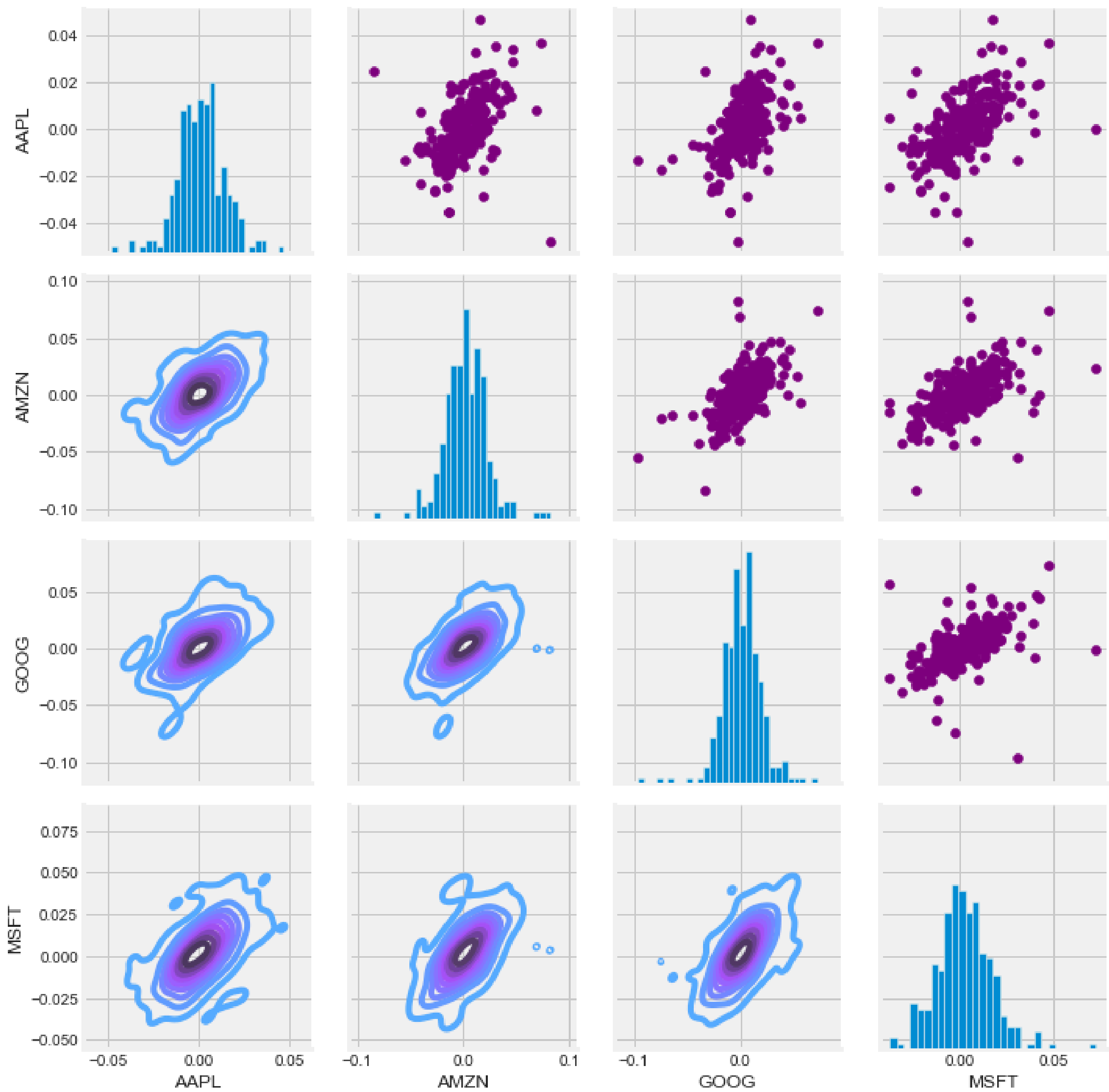


Above we can see all the relationships on daily returns between all the stocks. A quick glance shows an interesting correlation between Google and Amazon daily returns. It might be interesting to investigate that individual comaprison.

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.

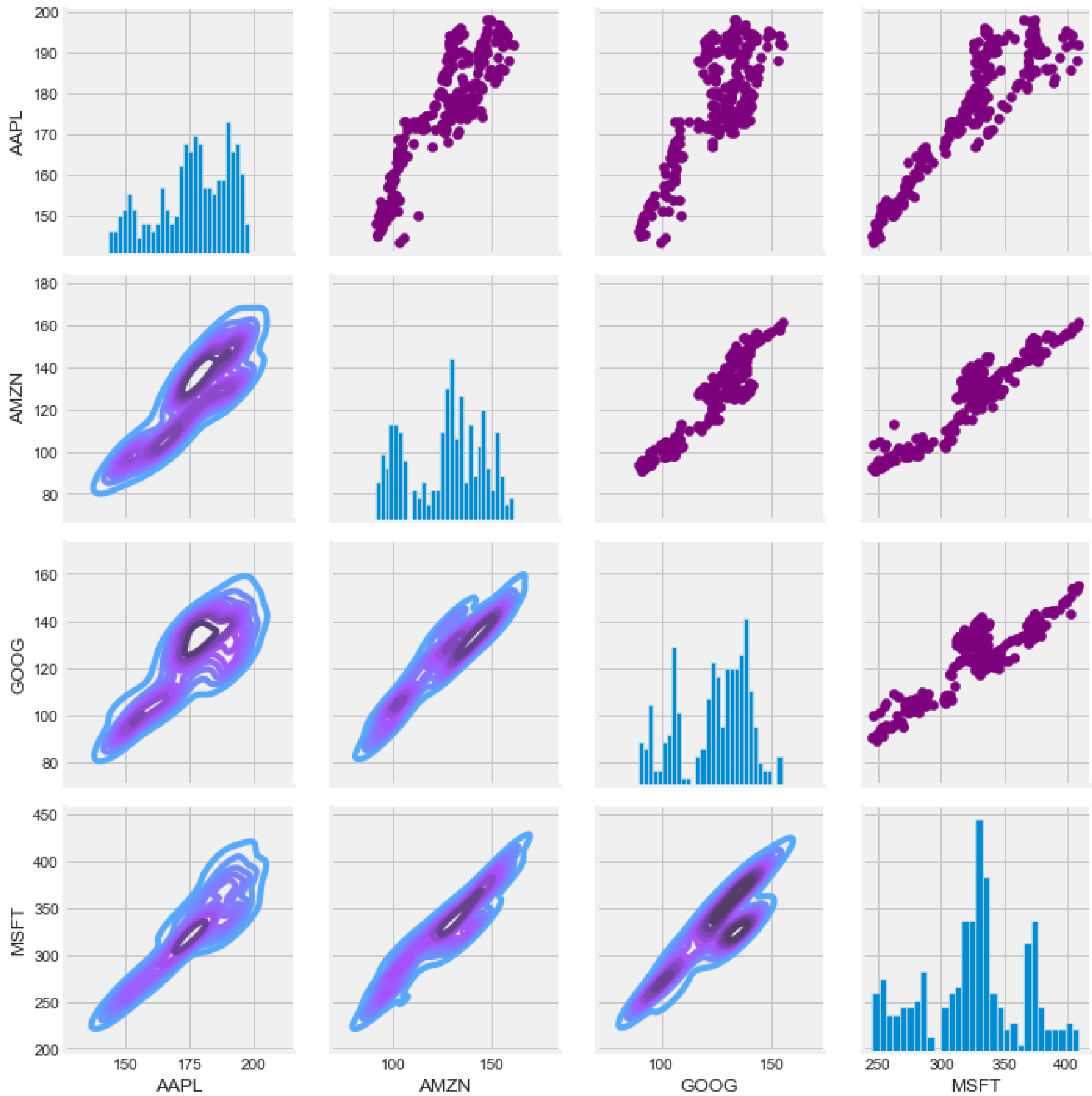
In [14]:
```
# Set up our figure by naming it returns_fig, call PairPLot on the DataFram
return_fig = sns.PairGrid(tech_rets.dropna())

# 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 t
# 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 d
return_fig.map_diag(plt.hist, bins=30)
```

Out[14]: <seaborn.axisgrid.PairGrid at 0x1e171832a70>



In [15]:
```
# Set up our figure by naming it returns_fig, call PairPLot on the DataFram
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, inclufing the plot t
returns_fig.map_lower(sns.kdeplot,cmap='cool_d')

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

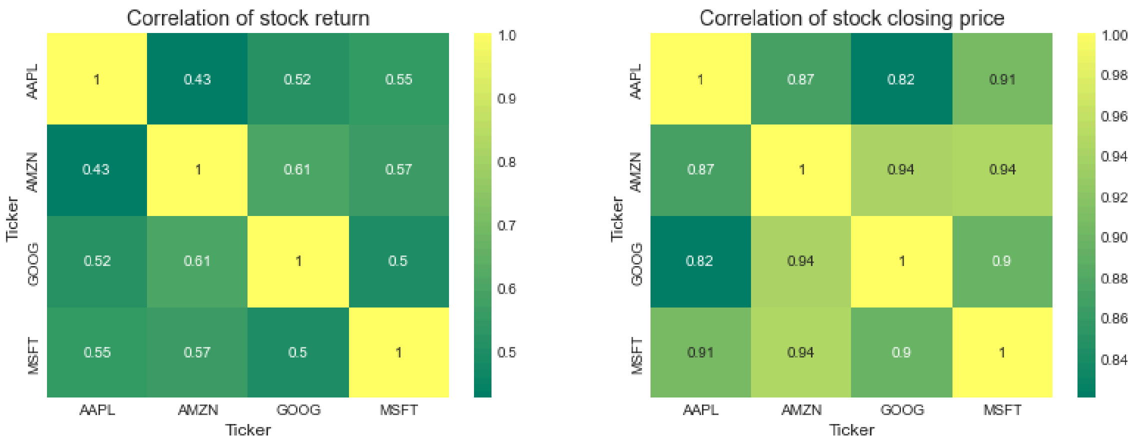
Out[15]: <seaborn.axisgrid.PairGrid at 0x1e17450d690>



Finally, 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.

In [16]:
```
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')
```

Out[16]:
```
Text(0.5, 1.0, 'Correlation of stock closing price')
```

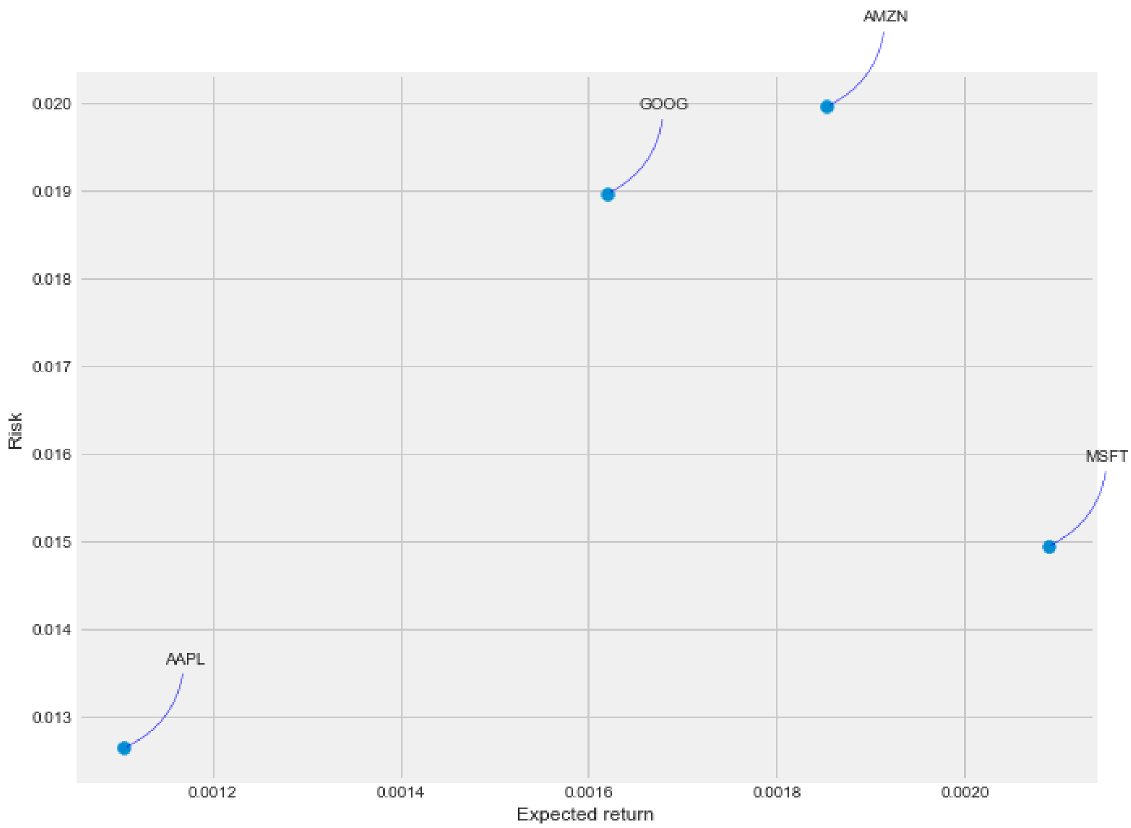


Just like we suspected in our `PairPlot` we see here numerically and visually that Microsoft and Amazon had the strongest correlation of daily stock return. It's also interesting to see that all the technology comapnies are positively correlated.

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.

In [17]:
```
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 poin
                 arrowprops=dict(arrowstyle='-', color='blue', connectionst
```



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

In [18]:
```
# 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
```

Out[18]:

Date	Open	High	Low	Close	Adj Close	Volume
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	12.516598	260022000
2012-01-05	14.819643	14.948214	14.738214	14.929643	12.655553	271269600
2012-01-06	14.991786	15.098214	14.972143	15.085714	12.787853	318292800
2012-01-09	15.196429	15.276786	15.048214	15.061786	12.767568	394024400
...	...	...	...	...	...	...
2024-01-25	195.220001	196.270004	193.110001	194.169998	194.169998	54822100
2024-01-26	194.270004	194.759995	191.940002	192.419998	192.419998	44553400
2024-01-29	192.009995	192.199997	189.580002	191.729996	191.729996	47145600
2024-01-30	190.940002	191.800003	187.470001	188.039993	188.039993	55753700
2024-01-31	187.039993	187.095001	184.789993	185.735992	185.735992	19526604

3039 rows × 6 columns

In [20]:
```
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 [21]:
```
# 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
```

Out[21]:
```
2888
```

In [23]:
```
# Scale the data
from sklearn.preprocessing import MinMaxScaler

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

scaled_data
```

Out[23]:
```
array([[0.00401431],
       [0.00444289],
       [0.00533302],
       ...,
       [0.96535666],
       [0.94531999],
       [0.9328093 ]])
```

In [25]:
```
# 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([0.00401431, 0.00444289, 0.00533302, 0.00618049, 0.00605056,
       0.00634339, 0.00620958, 0.00598462, 0.00567821, 0.00662652,
       0.00748175, 0.007218  , 0.00577323, 0.00715207, 0.00579457,
       0.01088518, 0.01049151, 0.01100542, 0.01211663, 0.01278955,
       0.01273332, 0.01252582, 0.01341013, 0.01424207, 0.01518457,
       0.01670691, 0.01990478, 0.01995326, 0.02173353, 0.02306387,
       0.02077746, 0.02165789, 0.02164044, 0.02410915, 0.02375813,
       0.02440779, 0.02557523, 0.0262249 , 0.02809631, 0.02945961,
       0.02985329, 0.02999098, 0.02765997, 0.02709757, 0.02718096,
       0.02937236, 0.02998905, 0.03131358, 0.03443581, 0.03860139,
       0.0378218 , 0.03782373, 0.04083544, 0.04177794, 0.04110694,
       0.04049413, 0.03985611, 0.04197573, 0.0434302 , 0.04403914])]
[0.042534249860459186]

[array([0.00401431, 0.00444289, 0.00533302, 0.00618049, 0.00605056,
       0.00634339, 0.00620958, 0.00598462, 0.00567821, 0.00662652,
       0.00748175, 0.007218  , 0.00577323, 0.00715207, 0.00579457,
       0.01088518, 0.01049151, 0.01100542, 0.01211663, 0.01278955,
       0.01273332, 0.01252582, 0.01341013, 0.01424207, 0.01518457,
       0.01670691, 0.01990478, 0.01995326, 0.02173353, 0.02306387,
       0.02077746, 0.02165789, 0.02164044, 0.02410915, 0.02375813,
       0.02440779, 0.02557523, 0.0262249 , 0.02809631, 0.02945961,
       0.02985329, 0.02999098, 0.02765997, 0.02709757, 0.02718096,
       0.02937236, 0.02998905, 0.03131358, 0.03443581, 0.03860139,
       0.0378218 , 0.03782373, 0.04083544, 0.04177794, 0.04110694,
       0.04049413, 0.03985611, 0.04197573, 0.0434302 , 0.04403914]), array
([0.00444289, 0.00533302, 0.00618049, 0.00605056, 0.00634339,
       0.00620958, 0.00598462, 0.00567821, 0.00662652, 0.00748175,
       0.007218  , 0.00577323, 0.00715207, 0.00579457, 0.01088518,
       0.01049151, 0.01100542, 0.01211663, 0.01278955, 0.01273332,
       0.01252582, 0.01341013, 0.01424207, 0.01518457, 0.01670691,
       0.01990478, 0.01995326, 0.02173353, 0.02306387, 0.02077746,
       0.02165789, 0.02164044, 0.02410915, 0.02375813, 0.02440779,
       0.02557523, 0.0262249 , 0.02809631, 0.02945961, 0.02985329,
       0.02999098, 0.02765997, 0.02709757, 0.02718096, 0.02937236,
       0.02998905, 0.03131358, 0.03443581, 0.03860139, 0.0378218 ,
       0.03782373, 0.04083544, 0.04177794, 0.04110694, 0.04049413,
       0.03985611, 0.04197573, 0.0434302 , 0.04403914, 0.04253425])]
[0.042534249860459186, 0.04053485447430975]
```

In [28]:
```
#from keras.models import Sequential
#from keras.layers import Dense, LSTM
#from tensorflow.keras.models import Sequential
#from tensorflow.keras.layers import Dense, LSTM

import torch.nn as nn

class AirModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.lstm = nn.LSTM(input_size=1, hidden_size=50, num_layers=1, bat
        self.linear = nn.Linear(50, 1)
    def forward(self, x):
        x, _ = self.lstm(x)
        x = self.linear(x)
        return x


import numpy as np
import torch.optim as optim
import torch.utils.data as data

model = AirModel()
optimizer = optim.Adam(model.parameters())
loss_fn = nn.MSELoss()
loader = data.DataLoader(data.TensorDataset(x_train, y_train), shuffle=True

n_epochs = 2000
for epoch in range(n_epochs):
    model.train()
    for X_batch, y_batch in loader:
        y_pred = model(X_batch)
        loss = loss_fn(y_pred, y_batch)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    # Validation
    if epoch % 100 != 0:
        continue
    model.eval()
    with torch.no_grad():
        y_pred = model(x_train)
        train_rmse = np.sqrt(loss_fn(y_pred, y_train))
        y_pred = model(x_test)
        test_rmse = np.sqrt(loss_fn(y_pred, y_test))
    print("Epoch %d: train RMSE %.4f, test RMSE %.4f" % (epoch, train_rmse,

# Build the LSTM model
model = Sequential()
model.add(LSTM(128, return_sequences=True, input_shape= (x_train.shape[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)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [28], in <cell line: 26>()
     24 optimizer = optim.Adam(model.parameters())
     25 loss_fn = nn.MSELoss()
---> 26 loader = data.DataLoader(data.TensorDataset(x_train, y_train), shuffle=True, batch_size=8)
     28 n_epochs = 2000
     29 for epoch in range(n_epochs):

File c:\Program Files\Python310\lib\site-packages\torch\utils\data\dataset.py:202, in TensorDataset.__init__(self, *tensors)
    201 def __init__(self, *tensors: Tensor) -> None:
--> 202     assert all(tensors[0].size(0) == tensor.size(0) for tensor in tensors), "Size mismatch between tensors"
    203     self.tensors = tensors

File c:\Program Files\Python310\lib\site-packages\torch\utils\data\dataset.py:202, in <genexpr>(.0)
    201 def __init__(self, *tensors: Tensor) -> None:
--> 202     assert all(tensors[0].size(0) == tensor.size(0) for tensor in tensors), "Size mismatch between tensors"
    203     self.tensors = tensors

TypeError: 'int' object is not callable
```

In [24]:

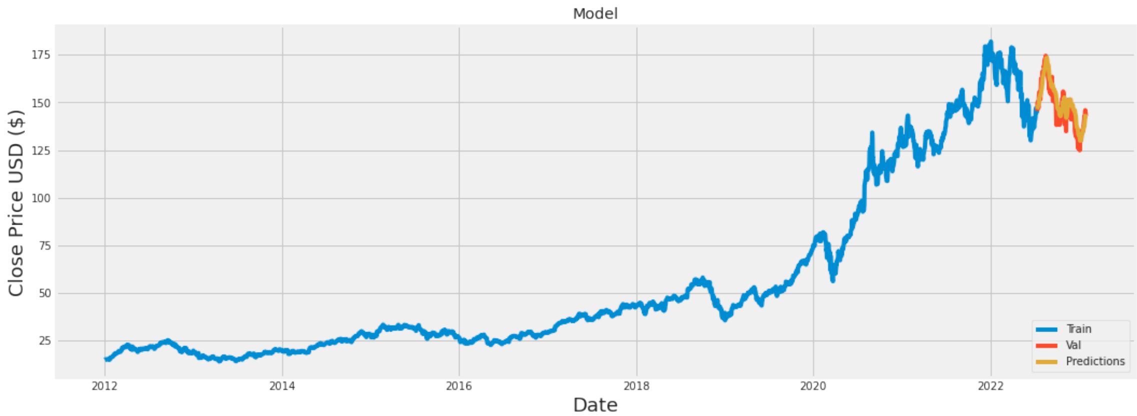
```
# Create the testing data set
# Create a new array containing scaled values from index 1543 to 2002
test_data = scaled_data[training_data_len - 60: , :]
# Create the data sets x_test and y_test
x_test = []
y_test = dataset[training_data_len:, :]
for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 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
```

Out[24]:

```
4.982936594544208
```

In [25]:
```
# Plot the data
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()
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:4: SettingWit
hCopyWarning:
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-doc
s/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)
  after removing the cwd from sys.path.
```



In [26]:
```
# Show the valid and predicted prices
valid
```

Out[26]:

	Close	Predictions
Date		
2022-07-13 00:00:00-04:00	145.490005	146.457565
2022-07-14 00:00:00-04:00	148.470001	146.872879
2022-07-15 00:00:00-04:00	150.169998	147.586197
2022-07-18 00:00:00-04:00	147.070007	148.572937
2022-07-19 00:00:00-04:00	151.000000	148.995255
...	...	...
2023-01-24 00:00:00-05:00	142.529999	138.565536
2023-01-25 00:00:00-05:00	141.860001	140.022110
2023-01-26 00:00:00-05:00	143.960007	141.225128
2023-01-27 00:00:00-05:00	145.929993	142.469315
2023-01-30 00:00:00-05:00	143.000000	143.833130

139 rows × 2 columns

Summary

In this notebook, you discovered and explored stock data.

Specifically, you learned:

- How to load stock market data from the YAHOO Finance website using yfinance.
- How to explore and visualize time-series data using Pandas, Matplotlib, and Seaborn.
- How to measure the correlation between stocks.
- How to measure the risk of investing in a particular stock.

Do you have any questions? Ask your questions in the comments below and I will do my best to answer.

References: [https://www.investopedia.com/terms/c/correlation.asp (https://www.investopedia.com/terms/c/correlation.asp)](https://www.investopedia.com/terms/c/correlation.asp) [Jose Portilla Udemy Course: Learning Python for Data Analysis and Visualization (https://www.udemy.com/course/learning-python-for-data-analysis-and-visualization/)](https://www.udemy.com/course/learning-python-for-data-analysis-and-visualization/)

In [ ]: