In [1]:

```
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
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[1]:

In [2]:

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

Out[2]:

Date	Open	High	Low	Close	Adj Close	Volume	company_name
2023-07-17	134.559998	135.619995	133.210007	133.559998	133.559998	48450200	AMAZON
2023-07-18	132.710007	133.860001	131.350006	132.830002	132.830002	54969100	AMAZON
2023-07-19	133.389999	135.990005	132.529999	135.360001	135.360001	54531000	AMAZON
2023-07-20	134.070007	134.789993	129.330002	129.960007	129.960007	59820600	AMAZON
2023-07-21	131.339996	131.369995	128.419998	130.000000	130.000000	133265000	AMAZON
2023-07-24	130.309998	131.660004	128.350006	128.600003	128.600003	45591100	AMAZON
2023-07-25	129.369995	129.860002	128.529999	129.130005	129.130005	39236700	AMAZON
2023-07-26	126.540002	129.080002	126.110001	128.149994	128.149994	53910100	AMAZON
2023-07-27	131.000000	132.630005	127.790001	128.250000	128.250000	52610700	AMAZON
2023-07-28	129.690002	133.009995	129.330002	132.210007	132.210007	46234700	AMAZON

	Open	High	Low	Close	Adj Close	Volume
count	251.000000	251.000000	251.000000	251.000000	251.000000	2.510000e+02
mean	158.435220	160.256933	156.880359	158.648605	158.287193	7.066039e+07
std	17.368964	17.178195	17.698029	17.450473	17.587156	2.320638e+07
min	126.010002	127.769997	124.169998	125.019997	124.656975	3.145820e+07
25%	145.815002	147.340004	144.014999	145.919998	145.470390	5.383755e+07
50%	154.789993	157.089996	153.369995	155.000000	154.602554	6.643510e+07
75%	170.775002	172.294998	169.909996	171.540001	171.027412	8.135480e+07
max	196.020004	198.229996	194.139999	195.830002	195.830002	1.647624e+08

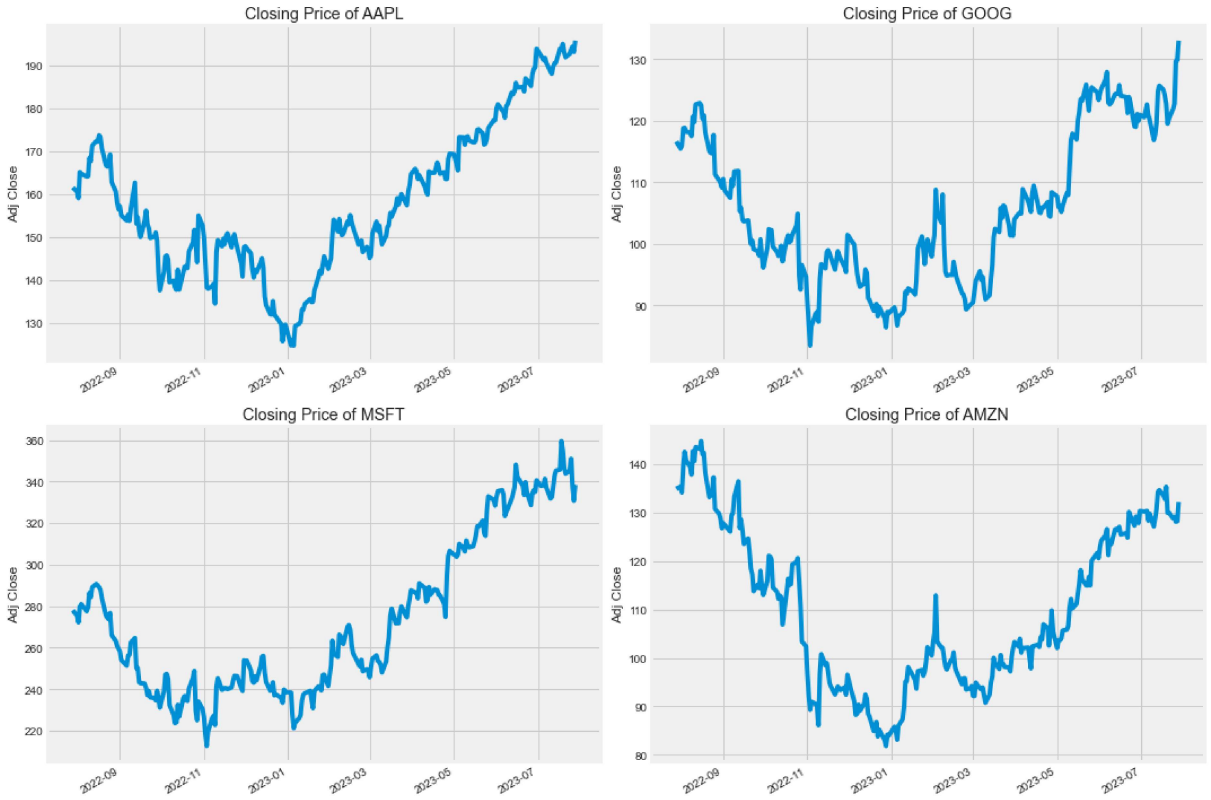
In [3]:

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

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 251 entries, 2022-07-29 to 2023-07-28
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
```

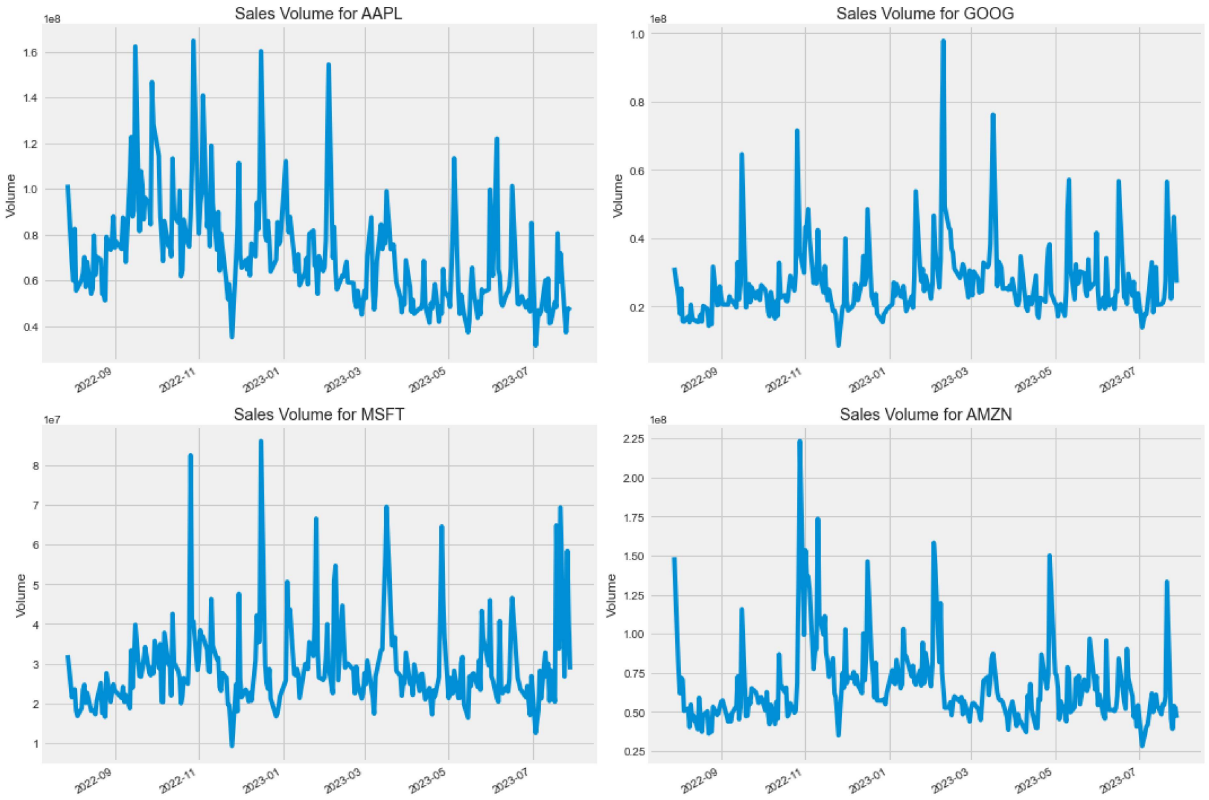
In [4]:

```
# 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()
```



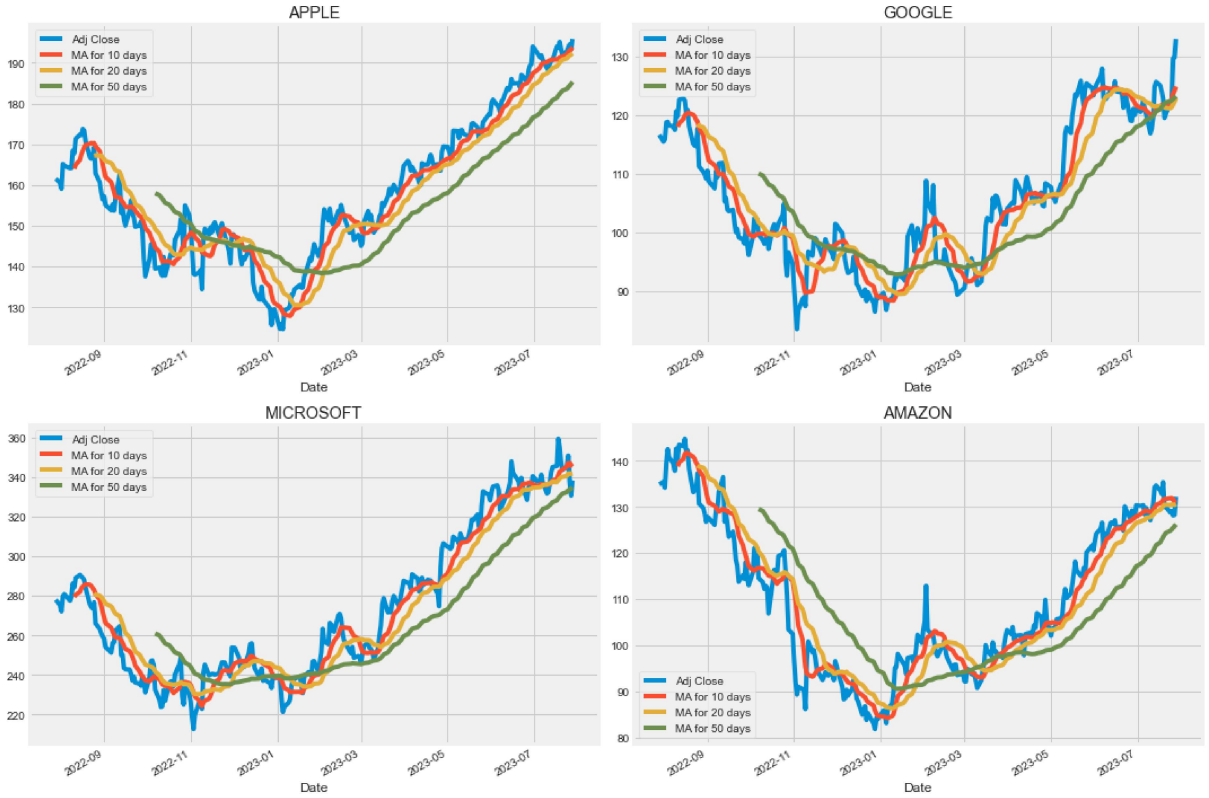
In [5]:

```
# 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()
```



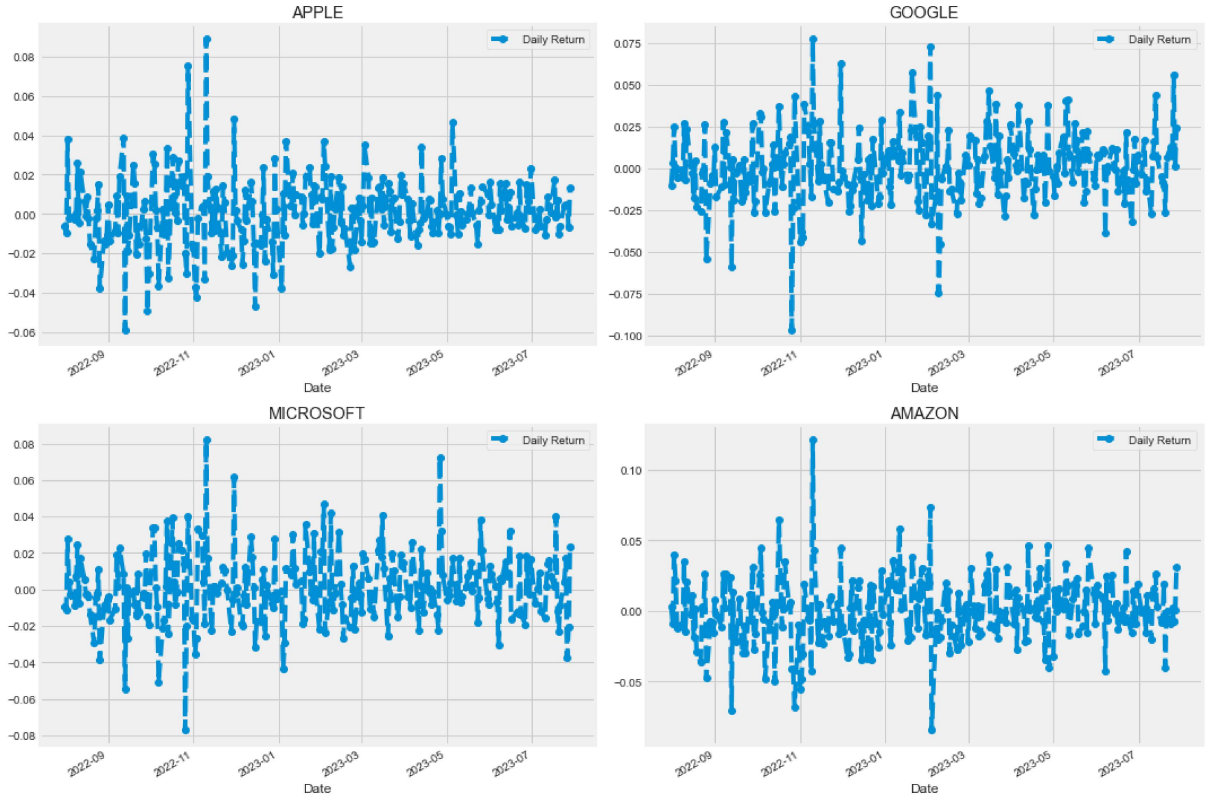
In [6]:

```
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()
```



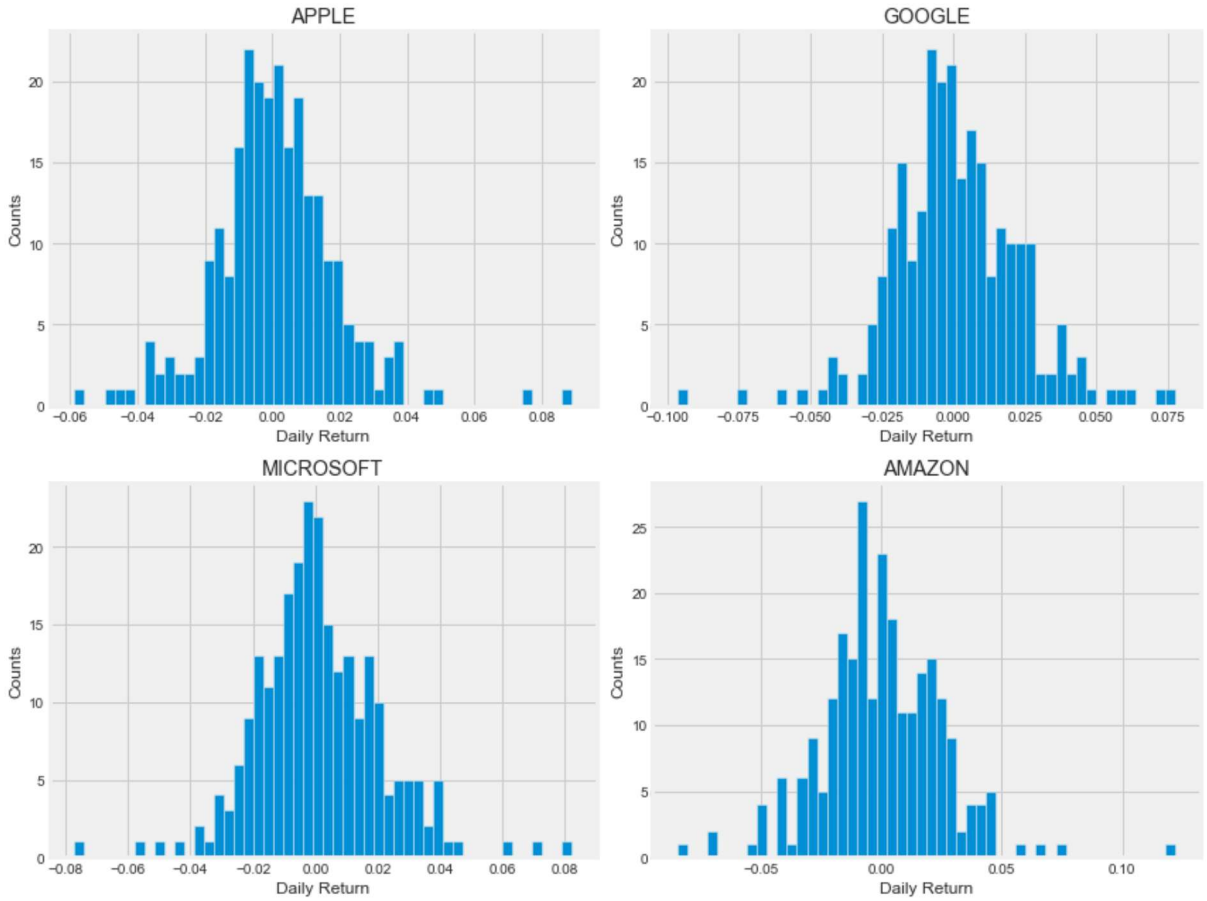
In [7]:

```
# 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()
```



In [8]:

```
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()
```



In [9]:

```
# 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[9]:

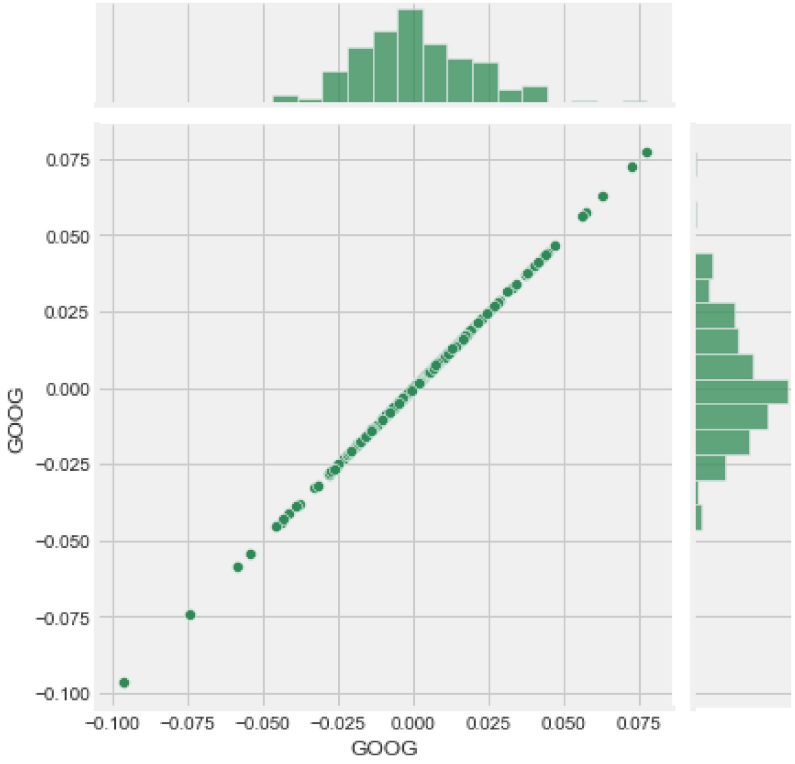
Date	AAPL	AMZN	GOOG	MSFT
2022-07-29	NaN	NaN	NaN	NaN
2022-08-01	-0.006153	0.003260	-0.009945	-0.009724
2022-08-02	-0.009287	-0.009085	0.003637	-0.011474
2022-08-03	0.038248	0.039952	0.024849	0.027836
2022-08-04	-0.001926	0.021861	0.000758	0.004177

In [10]:

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

Out[10]:

```
<seaborn.axisgrid.JointGrid at 0x1ff0e0dafb0>
```

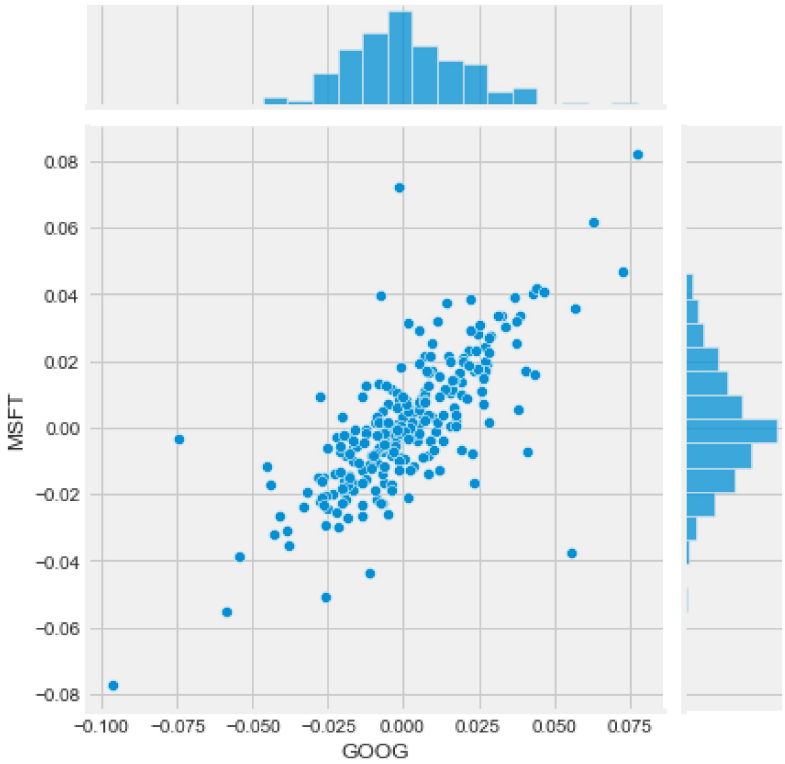


In [11]:

```
# 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[11]:

```
<seaborn.axisgrid.JointGrid at 0x1ff103ccdc0>
```

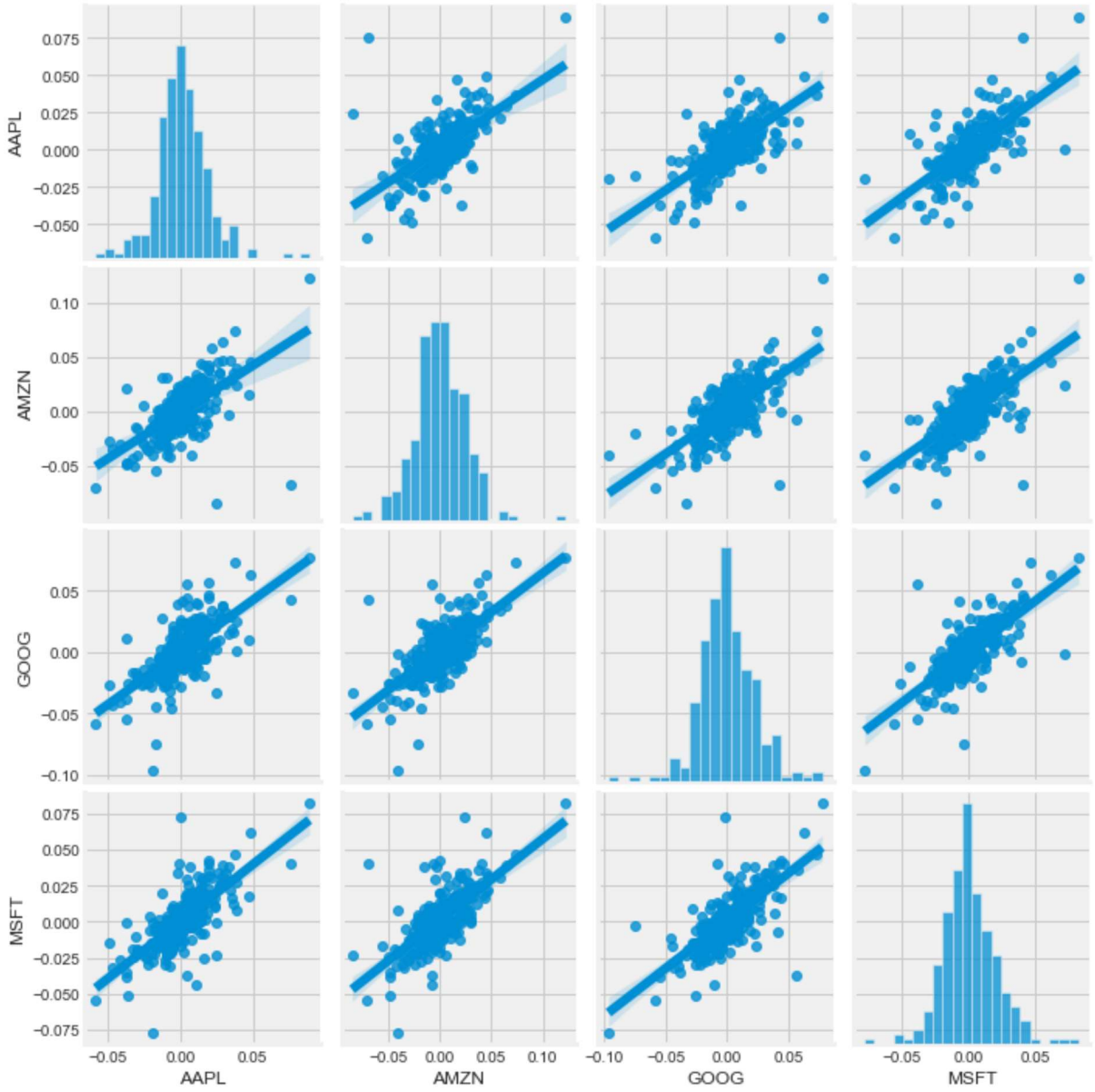


In [12]:

```
# We can simply call pairplot on our DataFrame for an automatic visual analysis
# of all the comparisons

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

Out[12]:

```
<seaborn.axisgrid.PairGrid at 0x1ff10de12a0>
```

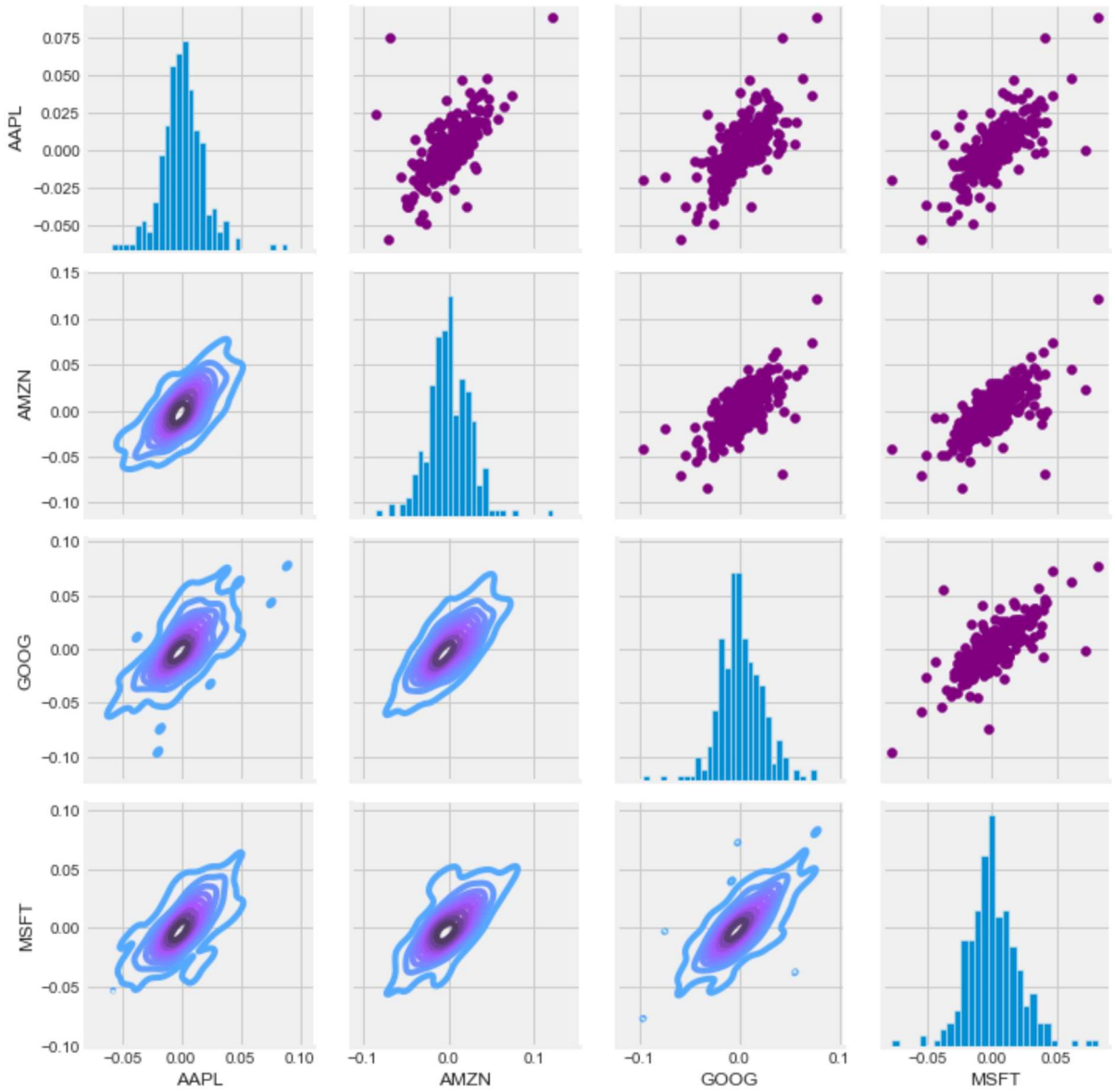


In [13]:

```
# 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)
```

Out[13]:

```
<seaborn.axisgrid.PairGrid at 0x1ff10f532e0>
```

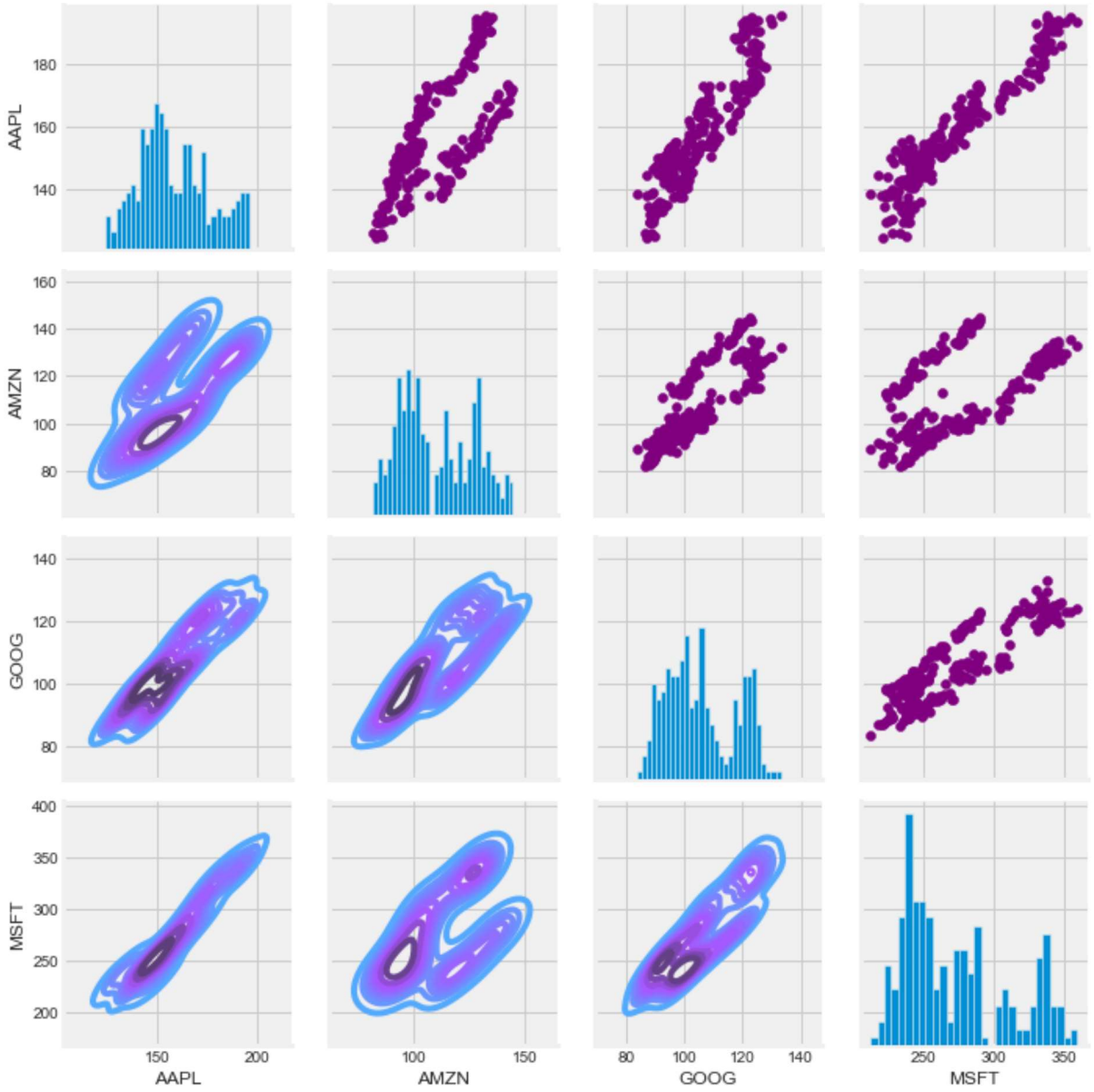


In [14]:

```
# 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, inclufing the plot type (kde) or the colo
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)
```

Out[14]:

```
<seaborn.axisgrid.PairGrid at 0x1ff14f87fa0>
```

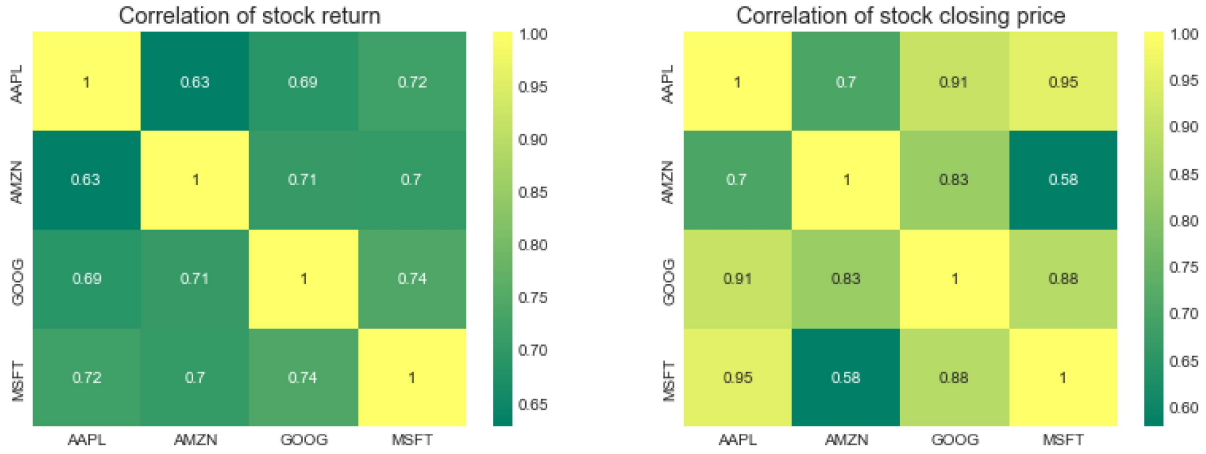


In [15]:

```
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')
```

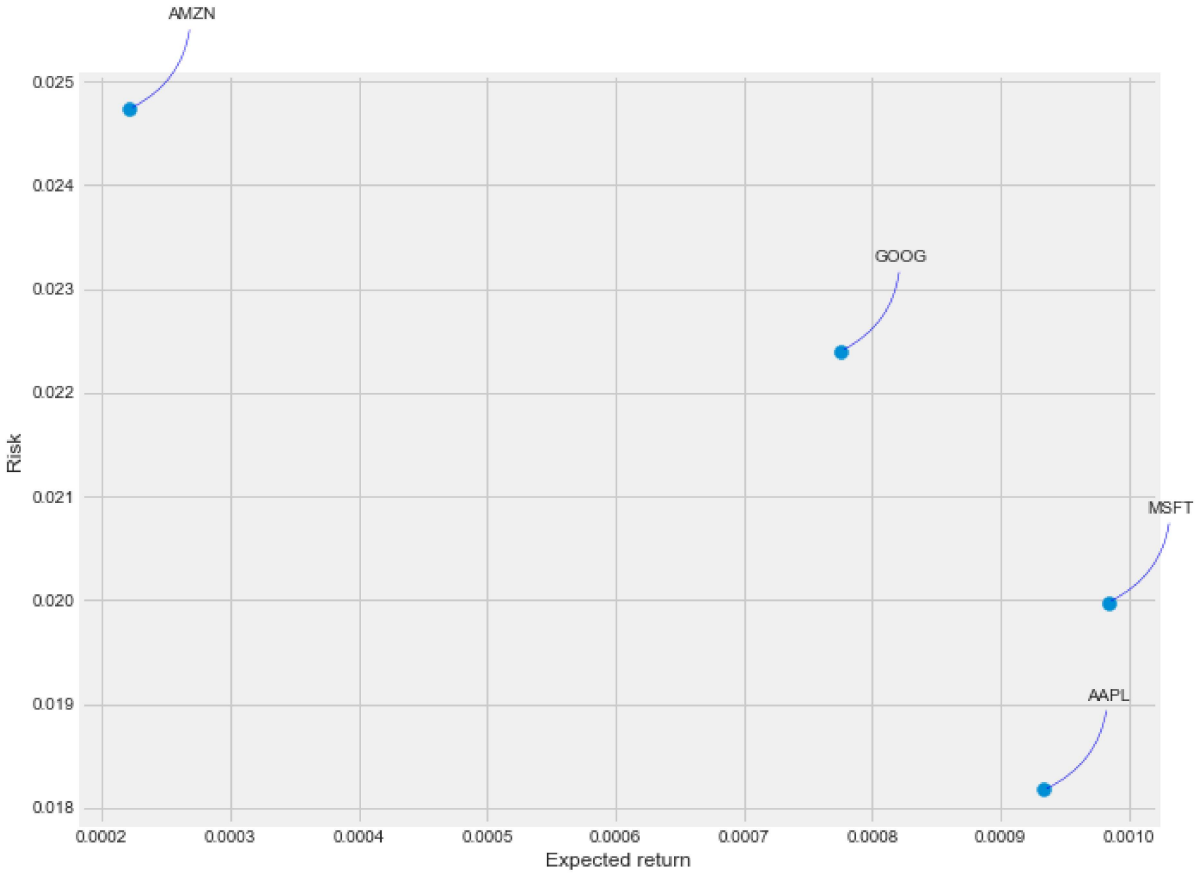
Out[15]:

```
Text(0.5, 1.0, 'Correlation of stock closing price')
```



In [16]:

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



In [17]:

```
# 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[17]:

Date	Open	High	Low	Close	Adj Close	Volume
2012-01-03	14.621429	14.732143	14.607143	14.686786	12.482925	302220800
2012-01-04	14.642857	14.810000	14.617143	14.765714	12.550009	260022000
2012-01-05	14.819643	14.948214	14.738214	14.929643	12.689343	271269600
2012-01-06	14.991786	15.098214	14.972143	15.085714	12.821992	318292800
2012-01-09	15.196429	15.276786	15.048214	15.061786	12.801657	394024400
...	...	...	...	...	...	...
2023-07-24	193.410004	194.910004	192.250000	192.750000	192.750000	45377800
2023-07-25	193.330002	194.440002	192.919998	193.619995	193.619995	37283200
2023-07-26	193.669998	195.639999	193.320007	194.500000	194.500000	47471900
2023-07-27	196.020004	197.199997	192.550003	193.220001	193.220001	47460200
2023-07-28	194.669998	196.630005	194.139999	195.830002	195.830002	48254600

2911 rows × 6 columns

In [18]:

```
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 [19]:

```
# 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[19]:

```
2766
```

In [20]:

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

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

scaled_data
```

Out[20]:

```
array([[0.00406463],
       [0.00449858],
       [0.00539987],
       ...,
       [0.99268758],
       [0.98565007],
       [1.        ]])
```

In [21]:

```
# 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([0.00406463, 0.00449858, 0.00539987, 0.00625796, 0.0061264 ,
       0.0064229 , 0.00628742, 0.00605964, 0.00574939, 0.00670959,
       0.00757553, 0.00730849, 0.00584561, 0.00724172, 0.00586721,
       0.01102163, 0.01062303, 0.01114338, 0.01226851, 0.01294988,
       0.01289294, 0.01268283, 0.01357823, 0.01442061, 0.01537492,
       0.01691634, 0.0201543 , 0.02020339, 0.02200597, 0.02335299,
       0.02103792, 0.02192938, 0.02191171, 0.02441137, 0.02405595,
       0.02471375, 0.02589583, 0.02655364, 0.02844851, 0.02982891,
       0.03022752, 0.03036694, 0.0280067 , 0.02743725, 0.02752169,
       0.02974055, 0.03036498, 0.03170611, 0.03486749, 0.03908528,
       0.03829592, 0.03829788, 0.04134734, 0.04230165, 0.04162224,
       0.04100175, 0.04035572, 0.04250192, 0.04397462, 0.04459119])]
[0.04306744044727151]

[array([0.00406463, 0.00449858, 0.00539987, 0.00625796, 0.0061264 ,
       0.0064229 , 0.00628742, 0.00605964, 0.00574939, 0.00670959,
       0.00757553, 0.00730849, 0.00584561, 0.00724172, 0.00586721,
       0.01102163, 0.01062303, 0.01114338, 0.01226851, 0.01294988,
       0.01289294, 0.01268283, 0.01357823, 0.01442061, 0.01537492,
       0.01691634, 0.0201543 , 0.02020339, 0.02200597, 0.02335299,
       0.02103792, 0.02192938, 0.02191171, 0.02441137, 0.02405595,
       0.02471375, 0.02589583, 0.02655364, 0.02844851, 0.02982891,
       0.03022752, 0.03036694, 0.0280067 , 0.02743725, 0.02752169,
       0.02974055, 0.03036498, 0.03170611, 0.03486749, 0.03908528,
       0.03829592, 0.03829788, 0.04134734, 0.04230165, 0.04162224,
       0.04100175, 0.04035572, 0.04250192, 0.04397462, 0.04459119]), array([0.0044985
8, 0.00539987, 0.00625796, 0.0061264 , 0.0064229 ,
       0.00628742, 0.00605964, 0.00574939, 0.00670959, 0.00757553,
       0.00730849, 0.00584561, 0.00724172, 0.00586721, 0.01102163,
       0.01062303, 0.01114338, 0.01226851, 0.01294988, 0.01289294,
       0.01268283, 0.01357823, 0.01442061, 0.01537492, 0.01691634,
       0.0201543 , 0.02020339, 0.02200597, 0.02335299, 0.02103792,
       0.02192938, 0.02191171, 0.02441137, 0.02405595, 0.02471375,
       0.02589583, 0.02655364, 0.02844851, 0.02982891, 0.03022752,
       0.03036694, 0.0280067 , 0.02743725, 0.02752169, 0.02974055,
       0.03036498, 0.03170611, 0.03486749, 0.03908528, 0.03829592,
       0.03829788, 0.04134734, 0.04230165, 0.04162224, 0.04100175,
       0.04035572, 0.04250192, 0.04397462, 0.04459119, 0.04306744])]
[0.04306744044727151, 0.04104298152285096]
```

In [22]:

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

# Build 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)
```

```
2706/2706 [==============================] - 150s 53ms/step - loss: 0.0015
```

Out[22]:

```
<keras.src.callbacks.History at 0x1ff20186bf0>
```

In [23]:

```
# 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 [==============================] - 2s 63ms/step
```

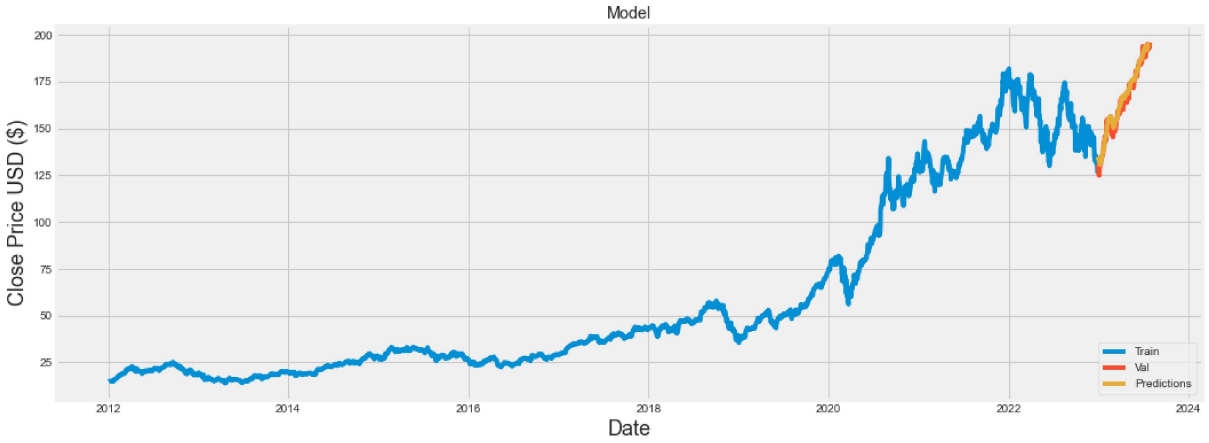
Out[23]:

```
3.0432974529144223
```

In [24]:

```
# 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()
```

```
C:\Users\Siddhi Jadhav\AppData\Local\Temp\ipykernel_16988\2388977846.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-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  valid['Predictions'] = predictions
```



In [25]:

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

Out[25]:

Date	Close	Predictions
2022-12-29	129.610001	134.775253
2022-12-30	129.929993	133.854248
2023-01-03	125.070000	133.334854
2023-01-04	126.360001	132.424393
2023-01-05	125.019997	131.628433
...	...	...
2023-07-24	192.750000	194.990494
2023-07-25	193.619995	195.045135
2023-07-26	194.500000	195.137436
2023-07-27	193.220001	195.352676
2023-07-28	195.830002	195.413116

145 rows × 2 columns

In [ ]:

```

```