make sure you have all the required modules installed before importing

In [2]:

```
import pandas as pd
from pandas import Series,DataFrame
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('darkgrid')
%matplotlib inline
from pandas_datareader import data
from datetime import datetime
from __future__ import division
import yfinance as yf
```

We will use Yahoo to grab some data for some tech stocks

In [401]:

```
# 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)

# using yf.download function to obtain data in a Data Fram
google = yf.download("GOOG",start,end)
print('GOT GOOGLES DATA')

apple = yf.download("AAPL",start,end)
print("GOT APPLE,S DATA")

Microsoft = yf.download("MSFT",start,end)
print("GOT MICROSOFT,S DATA")

Amazon = yf.download("AMZN",start,end)
print("GOT AMAZON,S DATA")
```

```
[*********************100%***********************]  1 of 1 completed
GOT GOOGLES DATA
[*********************100%***********************]  1 of 1 completed
GOT APPLE,S DATA
[*********************100%***********************]  1 of 1 completed
GOT MICROSOFT,S DATA
[*********************100%***********************]  1 of 1 completed
GOT AMAZON,S DATA
```

In [403]:

```
# Let,s work on apple data frame
```

In [404]:
```python
# Summary Stats
apple.describe()
```

Out[404]:

	Open	High	Low	Close	Adj Close	Volume
count	251.000000	251.000000	251.000000	251.000000	251.000000	2.510000e+02
mean	156.240240	158.085498	154.664940	156.498884	156.090902	7.166628e+07
std	15.447192	15.228435	15.768078	15.550968	15.669339	2.274350e+07
min	126.010002	127.769997	124.169998	125.019997	124.656975	3.145820e+07
25%	145.420006	147.264999	143.315002	145.675003	145.078445	5.517395e+07
50%	153.399994	155.240005	151.380005	153.720001	153.139252	6.840220e+07
75%	166.485001	168.135002	165.549995	166.939995	166.444038	8.161725e+07
max	193.779999	194.479996	191.759995	193.970001	193.970001	1.647624e+08

In [405]:
```python
# General Info about the apple data frame
apple.info()
```

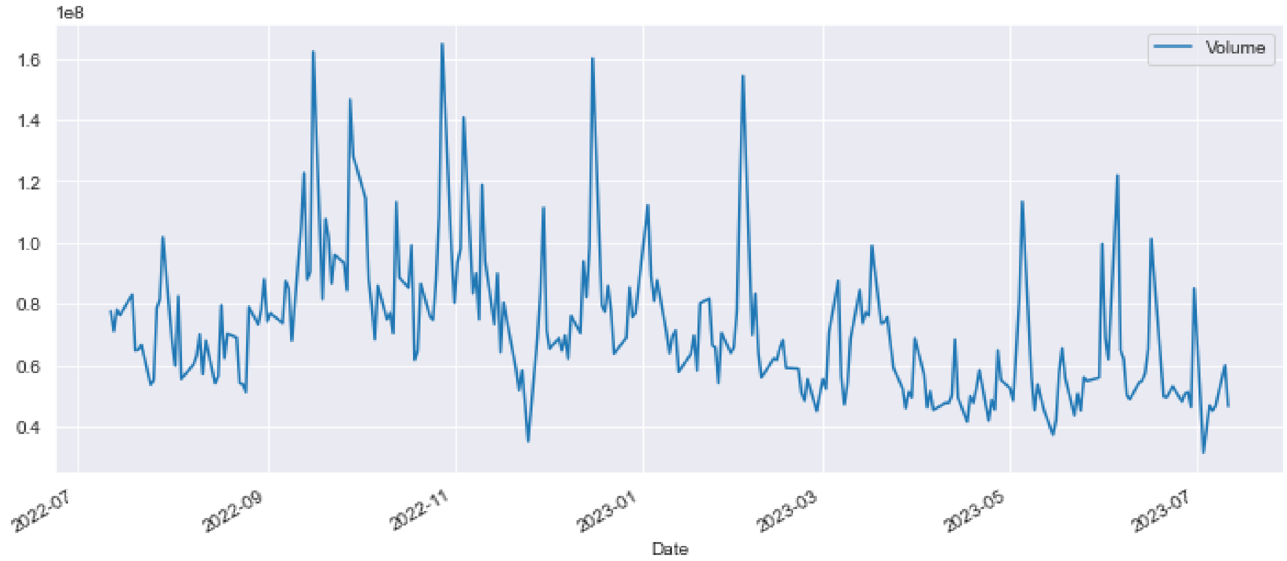
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 251 entries, 2022-07-12 to 2023-07-11
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Open       251 non-null    float64
 1   High       251 non-null    float64
 2   Low        251 non-null    float64
 3   Close      251 non-null    float64
 4   Adj Close  251 non-null    float64
 5   Volume     251 non-null    int64  
dtypes: float64(5), int64(1)
memory usage: 13.7 KB
```

In [406]:
```python
# Let's see a historical view of the closing price
apple['Adj Close'].plot(legend=True,figsize=(12,5))
```

Out[406]: <AxesSubplot:xlabel='Date'>

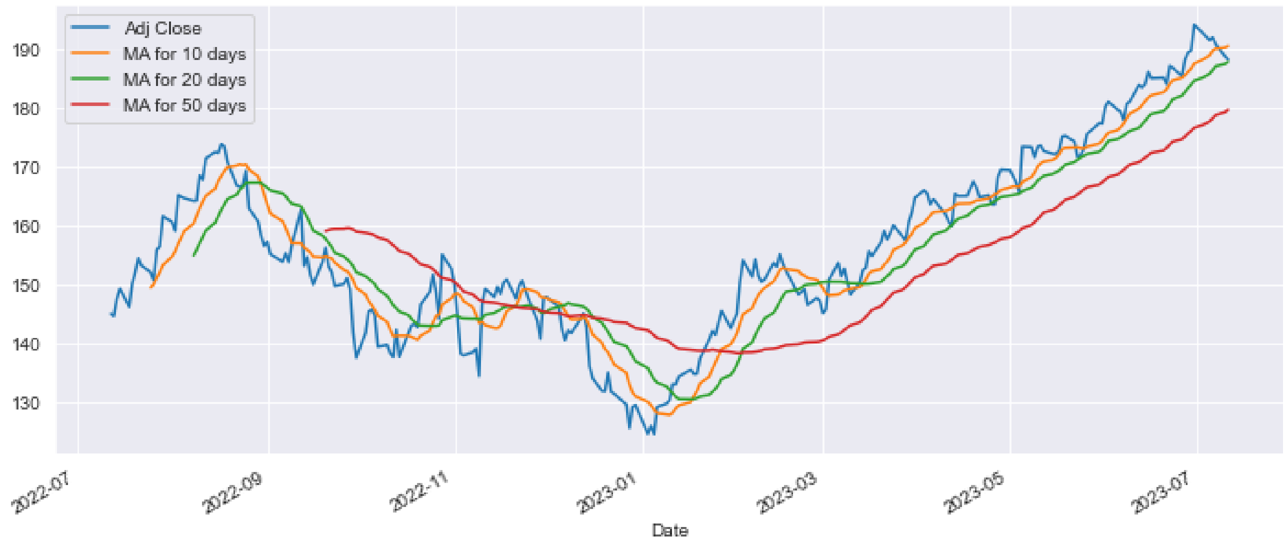
In [407]:
```python
# Now let's plot the total volume of stock being traded each day
apple['Volume'].plot(legend=True,figsize=(12,5))
```

Out[407]: <AxesSubplot:xlabel='Date'>



In [408]:
```python
#let's  caculate the moving average for the stock and plot it
# A moving average is a statistic that captures the average change in a data series o
ma_day=[10,20,50] #for moving averages for 10,20,50 days

for ma in ma_day:
    column_name="MA for %s days" %(str(ma))

    #apple[column_name]=pd.rolling_mean(apple['Adj Close'],ma)
    apple[column_name] = apple['Adj Close'].rolling(window=ma).mean()



apple[['Adj Close','MA for 10 days','MA for 20 days','MA for 50 days']].plot(figsize=
```

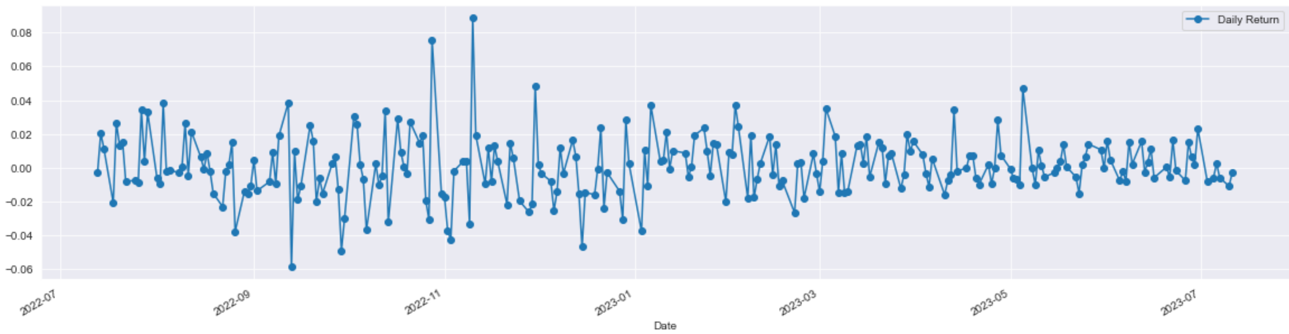
Out[408]: <AxesSubplot:xlabel='Date'>



Daily Return Analysis

In [409]:
```
# We'll use pct_change to find the percent change for each day and add it to the new
apple['Daily Return']=apple['Adj Close'].pct_change()
# Then we'll plot the daily return percentage
apple['Daily Return'].plot(legend=True,figsize=(20,5),marker='o')
```

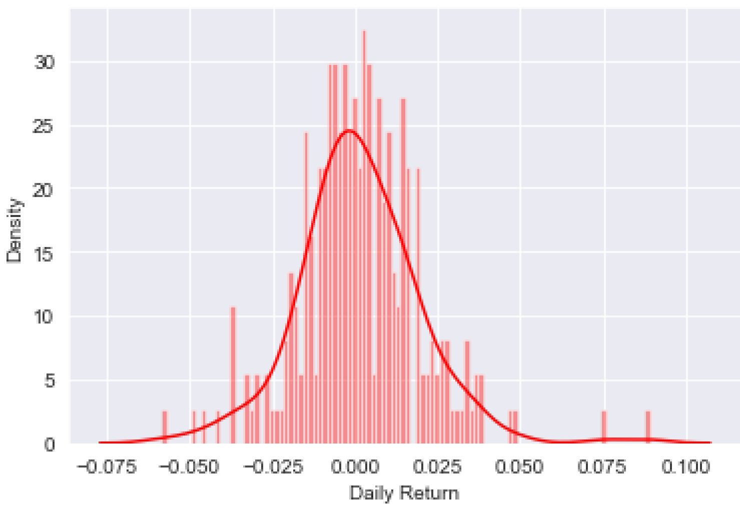
Out[409]: <AxesSubplot:xlabel='Date'>



In [410]:
```
#ploting Daily Return
sns.distplot(apple['Daily Return'],bins=100,color='red')
```

```
C:\Users\welcome\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWa
rning: `distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with similar
flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[410]: <AxesSubplot:xlabel='Daily Return', ylabel='Density'>



In [411]:
```
# Grab all the closing prices for the tech stock list into one DataFrame
closing_df = yf.download(tech_list,start,end)['Adj Close'] #Data frame for Adj Close
closing_df.head()
```

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

Out[411]:

	AAPL	AMZN	GOOG	MSFT
Date				
2022-07-12	144.994202	109.220001	114.849503	251.241440
2022-07-13	144.626404	110.400002	112.186996	250.300537
2022-07-14	147.588715	110.629997	111.440002	251.647522
2022-07-15	149.278610	113.550003	112.766998	254.262253
2022-07-18	146.197037	113.760002	109.910004	251.815903

In [412]:

```
#Data frame for Daily Returns of stocks
tech_rets = closing_df.pct_change()
tech_rets.head()
```

Out[412]:

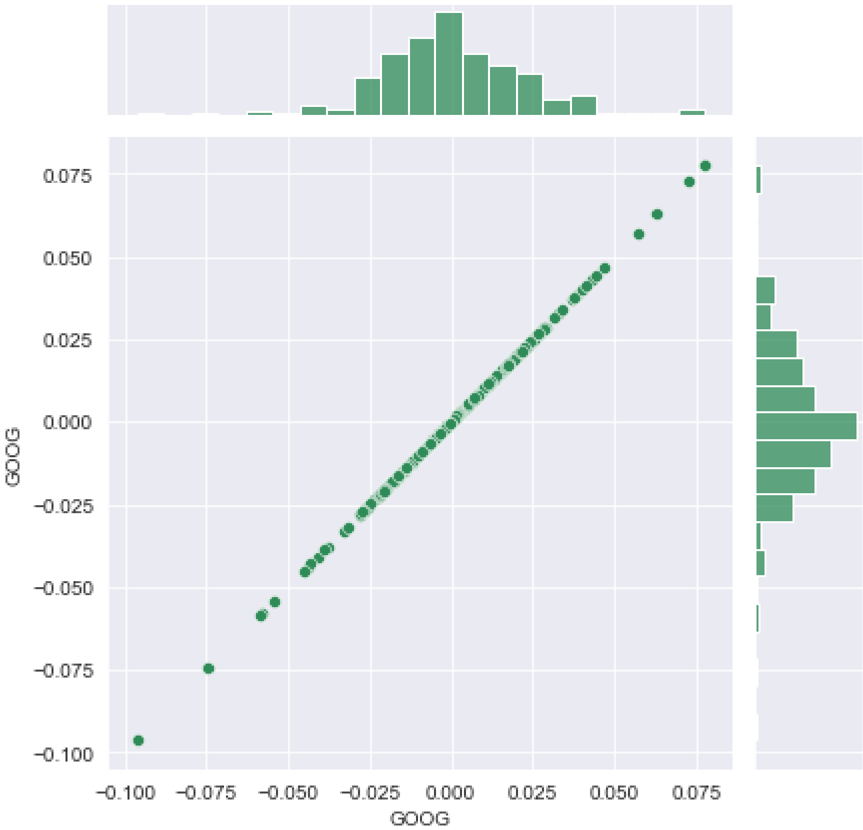
Date	AAPL	AMZN	GOOG	MSFT
2022-07-12	NaN	NaN	NaN	NaN
2022-07-13	-0.002537	0.010804	-0.023183	-0.003745
2022-07-14	0.020483	0.002083	-0.006658	0.005381
2022-07-15	0.011450	0.026394	0.011908	0.010390
2022-07-18	-0.020643	0.001849	-0.025335	-0.009621

Comparing the daily percentage return of two stocks to check how correlated they are

In [413]:

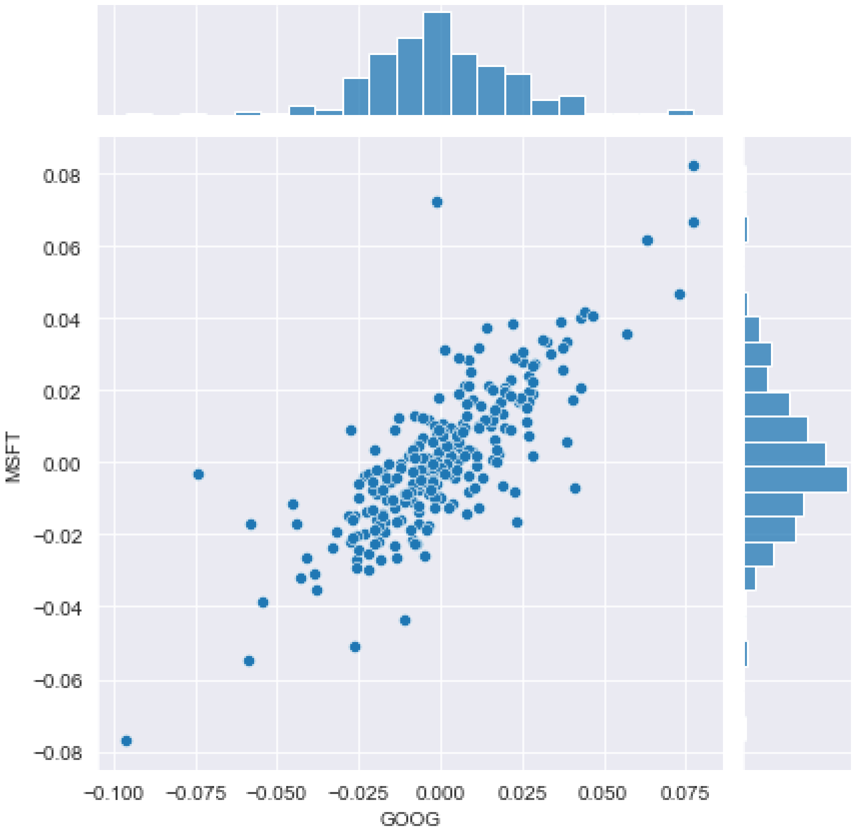
```
# Comparing Google to itself should show a perfectly linear relationship
sns.jointplot(x='GOOG', y='GOOG', data=tech_rets, kind='scatter',color='seagreen')
# This shows that google stock is perfectly correlated to itself
```

Out[413]: <seaborn.axisgrid.JointGrid at 0x14d6f1bcdc0>



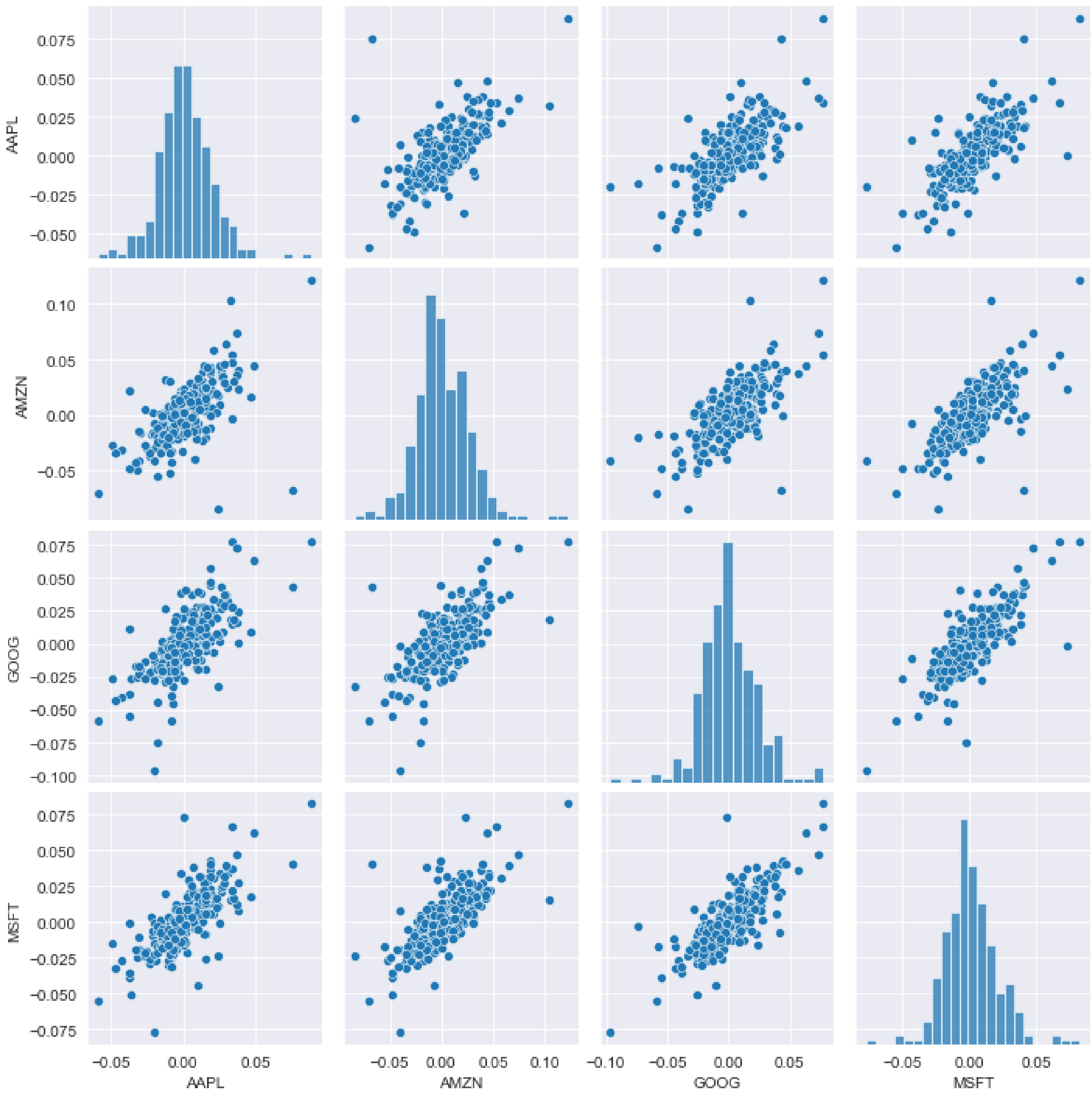
In [414]:
```python
# We'll use jointplot to compare the daily returns of Google and Microsoft
sns.jointplot(x='GOOG', y='MSFT', data=tech_rets, kind='scatter')
```

Out[414]: <seaborn.axisgrid.JointGrid at 0x14d6f2bb370>



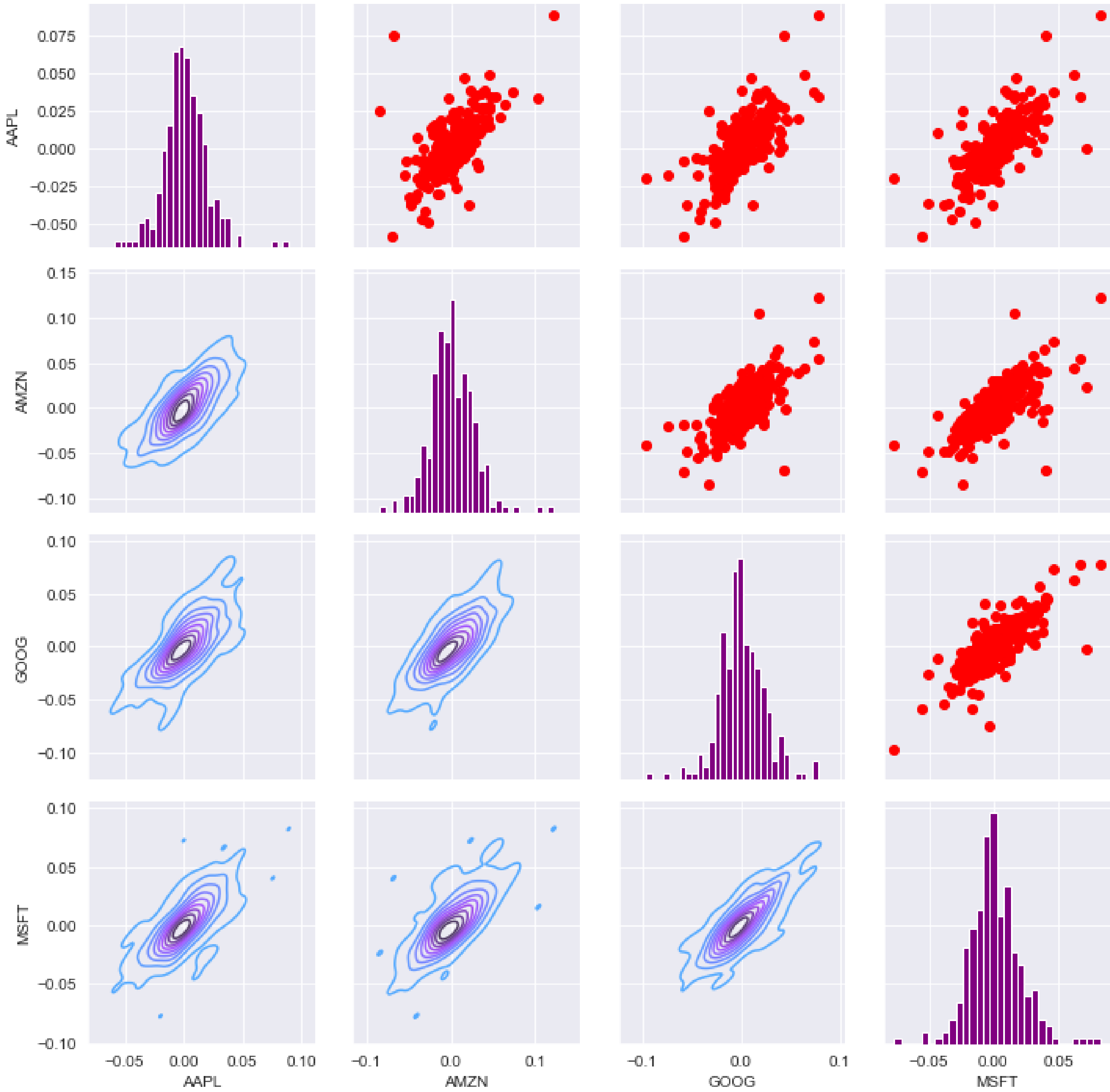
In [415]:
```
# We will call pairplot on our DataFrame for an automatic visual analysis of all the
sns.pairplot(tech_rets.dropna())
# Below we can see all the relationships on daily returns between all the stocks
```

Out[415]: <seaborn.axisgrid.PairGrid at 0x14d6f204070>



In [417]:
```
# Set up our figure by naming it returns_fig, call PairPLot on the DataFrame
returns_fig = sns.PairGrid(tech_rets.dropna())

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

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

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

Out[417]: <seaborn.axisgrid.PairGrid at 0x14d6f346850>



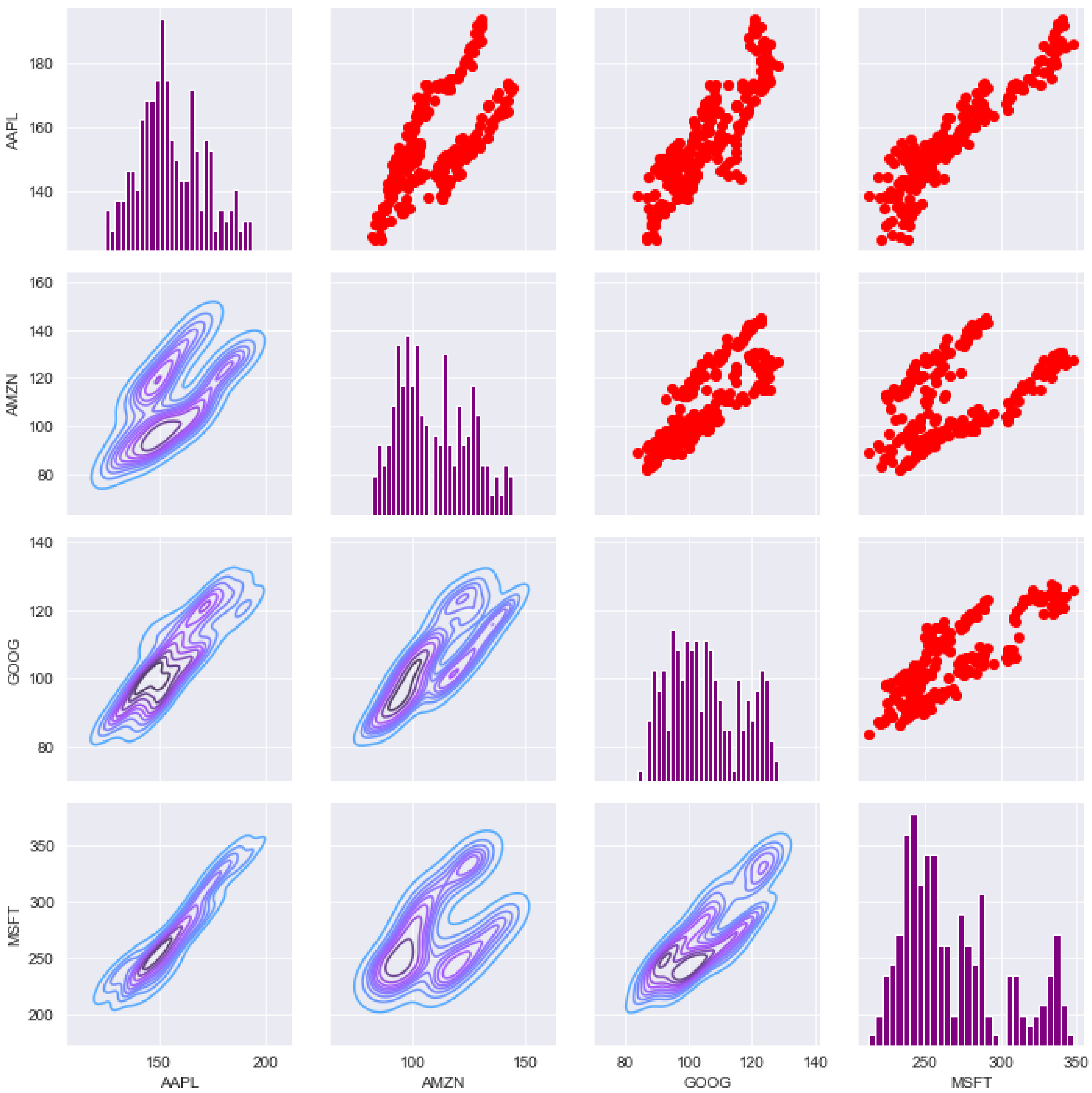
In [418]:
```python
#we could also do a heat map plot, to get actual numerical values for the correlation
#between the stocks' daily return values. By comparing the closing prices,
#we see an interesting relationship between Microsoft and Google.

corr_matrix_rets = tech_rets.dropna().corr()
sns.heatmap(corr_matrix_rets, annot=True)
```

Out[418]: <AxesSubplot:>

We will also analyzed the correlation of the closing prices using this exact same technique

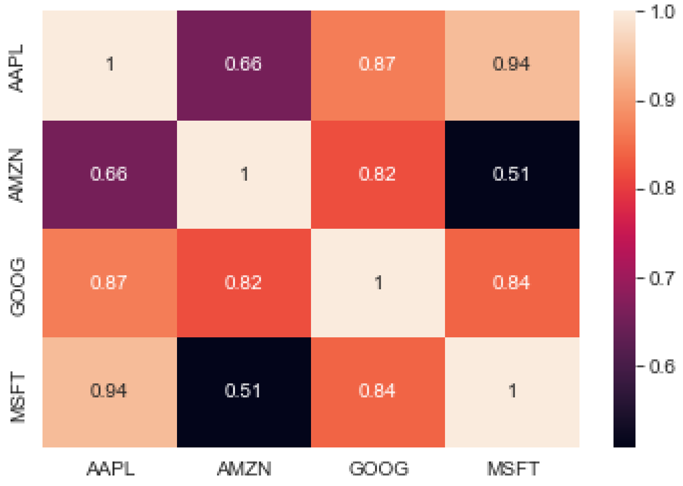
In [296]:
```python
# Set up our figure by naming it returns_fig, call PairPLot on the DataFrame
returns_fig = sns.PairGrid(closing_df.dropna())

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

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

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

'''In the figure below we can see a strong correlation between Microsoft and Apple cl
```

Out[296]: <seaborn.axisgrid.PairGrid at 0x14d607748e0>



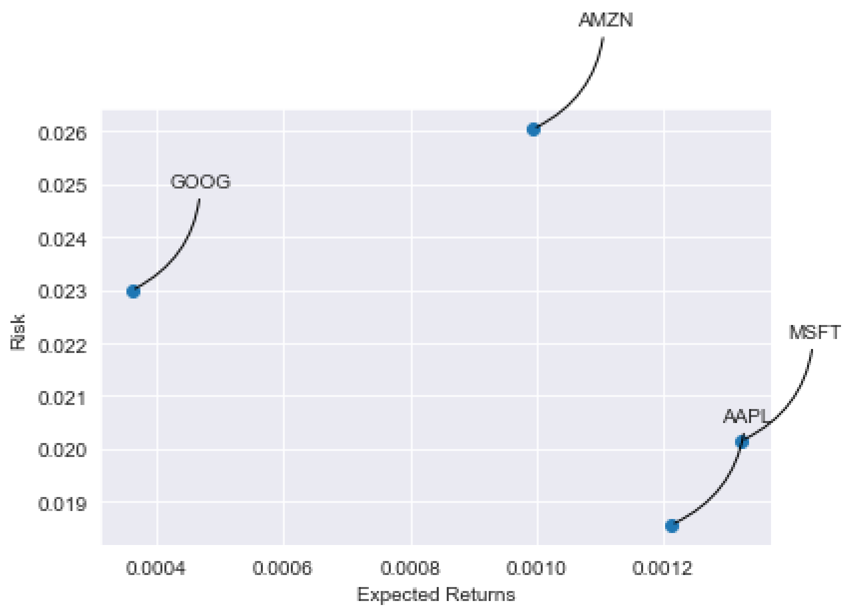
In [419]:
```
# We can see the correlations better by actual numerical values for the correlation by
corr_matrix_closing = closing_df.dropna().corr()
sns.heatmap(corr_matrix_closing, annot=True)
'''In the figure below we can see a that numericalvalue of correlation between Micros
```

Out[419]:
```
'In the figure below we can see a that numericalvalue of correlation between Microso
ft and Apple closing prices'
```



Risk Analysis

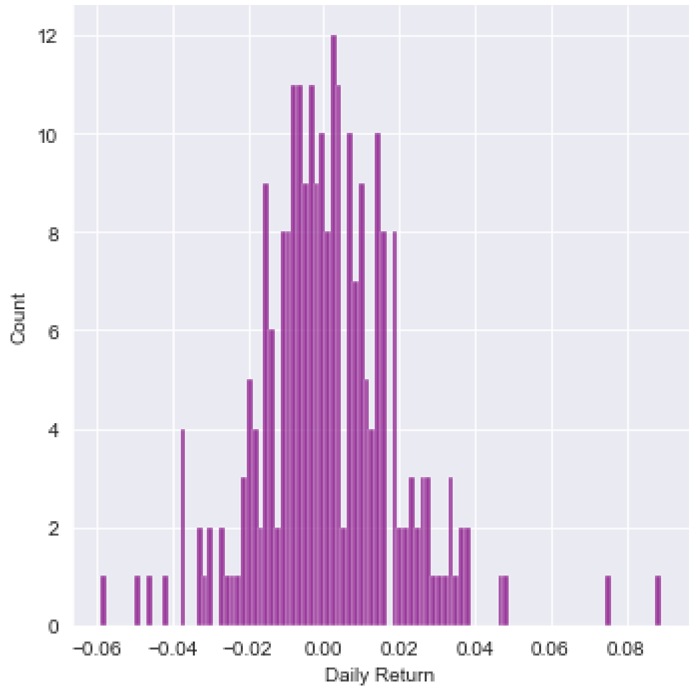
In [421]:
```
#fix scatter plot and anotate

# Let's start by defining a new DataFrame as a clenaed version of the oriignal tech_re
rets = tech_rets.dropna()

plt.scatter(rets.mean(),rets.std())

#Set the plot axis titles
plt.xlabel('Expected Returns')
plt.ylabel('Risk')

'''we calculate risk by taking standard diviation of returns

It's worth noting that standard deviation alone may not fully capture all aspects of
Other factors such as market conditions, company-specific news,
and industry trends should also be considered when assessing the risk of a stock.'''

# Label the scatter plots
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 = '-',connectionstyle = 'arc3,rad=-0.3',color='black



    # we should pick stock with highest expected return and lowest risk
    # So as seen in figure below Apple has lowest risk with High Returns
```



Value at risk

In [422]:
```python
#Value at risk using the "bootstrap" method
sns.displot(apple['Daily Return'].dropna(),bins=100,color='purple')
```

Out[422]:
```
<seaborn.axisgrid.FacetGrid at 0x14d70f07460>
```



In [423]:
```python
rets.head()
```

Out[423]:

	AAPL	AMZN	GOOG	MSFT
Date				
2022-07-13	-0.002537	0.010804	-0.023183	-0.003745
2022-07-14	0.020483	0.002083	-0.006658	0.005381
2022-07-15	0.011450	0.026394	0.011908	0.010390
2022-07-18	-0.020643	0.001849	-0.025335	-0.009621
2022-07-19	0.026722	0.039117	0.042853	0.020767

In [424]:
```python
# The 0.05 empirical quantile of daily returns
rets['AAPL'].quantile(0.05) #with 95% times your worst daily lose would not exceed th
```

Out[424]:
```
-0.028527532893352595
```

The 0.05 empirical quantile of daily returns is at -0.028. That means that with 95% confidence, our worst daily loss will not exceed 2.8%. If we have a 1 million dollar investment, our one-day 5% VaR is 0.028 * 1,000,000 = $28,000.

In [425]:

```
# Value at Risk by monte Carlos Method
'''Using the Monte Carlo to run many trials with random market conditions,
then we'll calculate portfolio losses for each trial.
After this, we'll use the aggregation of all these simulations to establish how risky

days = 365                         # Set up our time horizon
dt = 1/days                        # Now our delta
mu = rets.mean()['GOOG']           # Now let's grab our mu (drift) from the expected retu
sigma = rets.std()['GOOG']         # Now let's grab our mu (drift) from the expected retu


#function that takes in the starting price and number of days,
#and uses teh sigma and mu we already calculated form out daily returns
def stock_monte_carlo(start_price,days,mu,sigma):

    # Define a price array
    price = np.zeros(days)
    price[0] = start_price

    # Schok and Drift
    shock = np.zeros(days)
    drift = np.zeros(days)

        # Run price array for number of days
    for x in range(1,days):

        shock[x] = np.random.normal(loc=mu*dt,scale= sigma*np.sqrt(dt))
        drift[x] = mu*dt
        price[x] = price[x-1] + (price[x-1] * (drift[x] + shock[x]))
    #print(price)
    return price
```

For more info on the Monte Carlo method for stocks, check out the following link: [http://www.investopedia.com/articles/07/montecarlo.asp (http://www.investopedia.com/articles/07/montecarlo.asp)](http://www.investopedia.com/articles/07/montecarlo.asp)

In [426]:

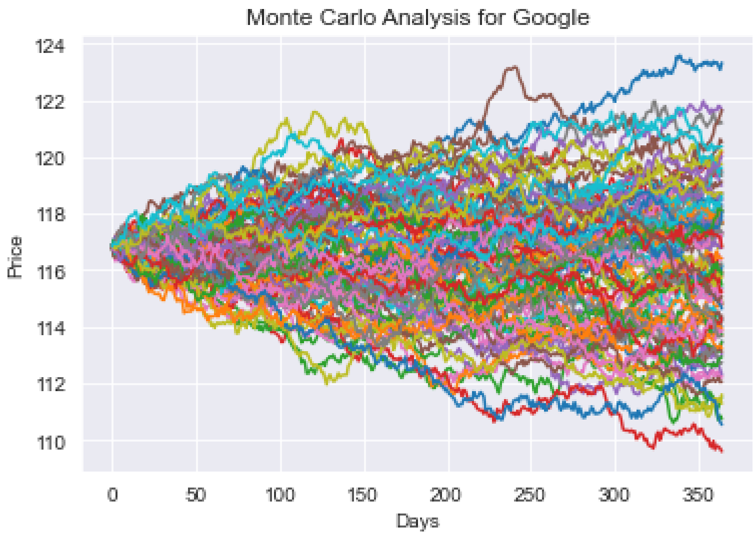
```
google.head()
```

Out[426]:

Date	Open	High	Low	Close	Adj Close	Volume
2022-07-12	116.838501	117.849503	114.614998	114.849503	114.849503	24970000
2022-07-13	112.639000	115.156998	111.822998	112.186996	112.186996	38958000
2022-07-14	110.825996	111.987503	109.325500	111.440002	111.440002	32366000
2022-07-15	112.962997	114.000504	111.822502	112.766998	112.766998	34330000
2022-07-18	113.440002	114.800003	109.300003	109.910004	109.910004	33354000

In [427]:
```python
# Get start price from GOOG.head()
start_price = 116.83

for x in range(100):
    plt.plot(stock_monte_carlo(start_price,days,mu,sigma))
plt.xlabel("Days")
plt.ylabel("Price")
plt.title('Monte Carlo Analysis for Google')
```

Out[427]: Text(0.5, 1.0, 'Monte Carlo Analysis for Google')

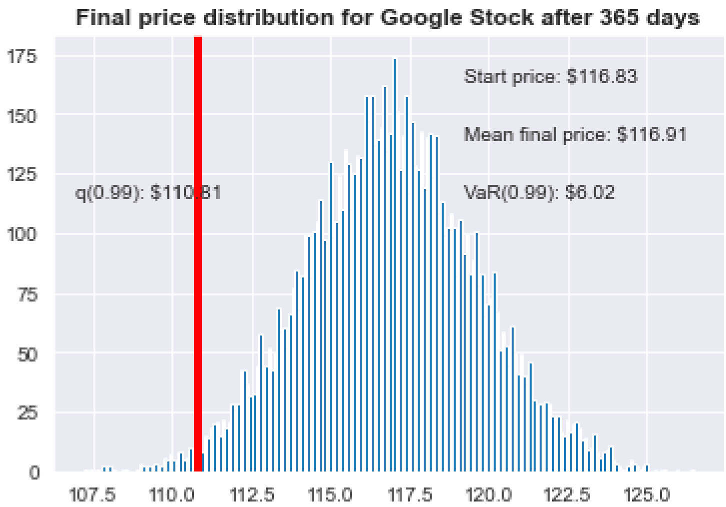


In [431]:
```python
#Let's get a histogram of the end results for a much larger run

# Set a large numebr of runs
runs = 10000

# Create an empty matrix to hold the end price data
simulations = np.zeros(runs)

for run in range(runs):
    # Create an empty matrix to hold the end price data
    simulations[run] = stock_monte_carlo(start_price, days, mu, sigma)[days-1]
```

In [430]:

```
# Now we'lll define q as the 1% empirical qunatile, this basically means that 99% of
q = np.percentile(simulations,1)

# Now let's plot the distribution of the end prices
plt.hist(simulations,bins=200)


# Title
plt.title(u"Final price distribution for Google Stock after %s days" % days, weight='

# Starting Price
plt.figtext(0.6, 0.8, "Start price: $%.2f" % start_price)

# Mean ending price
plt.figtext(0.6, 0.7, "Mean final price: $%.2f" % np.mean(simulations))

# Variance of the price (within 99% confidence interval)
#q = np.percentile(simulations, 1)
plt.figtext(0.6, 0.6, "VaR(0.99): $%.2f" % (start_price - q,))

# Display 1% quantile
plt.figtext(0.15, 0.6, "q(0.99): $%.2f" % q)

# Plot a line at the 1% quantile result
plt.axvline(x=q, linewidth=4, color='r')
```

Out[430]: <matplotlib.lines.Line2D at 0x14d5e8b7220>



Making Prediction of google Stock using Rnn

In [32]:

```
# using yf.download function to obtain data in a Data Fram
end=datetime.now()
en = datetime(end.year,end.month-1,end.day)
start = datetime(end.year-5,end.month,end.day)

#in data frame getting google,s 5 years stock information
dataset_train = yf.download("GOOG",start,en)
print('GOT GOOGLES DATA')
# getting the open stock prices into a training set
training_set = dataset_train.iloc[:, 1:2].values
```

```
[*********************100%***********************]  1 of 1 completed
GOT GOOGLES DATA
```

In [27]:
```
dataset_train
```

Out[27]:

	Open	High	Low	Close	Adj Close	Volume
Date						
2018-07-26	62.549999	63.488548	62.451000	63.416500	63.416500	48112000
2018-07-27	63.549999	63.694500	61.549999	61.924999	61.924999	42612000
2018-07-30	61.400501	61.745800	60.573502	60.987000	60.987000	36998000
2018-07-31	61.000500	61.379398	60.279999	60.862999	60.862999	32894000
2018-08-01	61.400002	61.673500	60.510502	61.000500	61.000500	31344000
...	...	...	...	...	...	...
2023-06-16	126.699997	126.699997	123.790001	124.059998	124.059998	56686800
2023-06-20	123.535004	125.175003	122.830002	123.849998	123.849998	22698000
2023-06-21	123.235001	123.410004	120.860001	121.260002	121.260002	22612000
2023-06-22	120.660004	123.934998	119.599998	123.870003	123.870003	20781900
2023-06-23	122.040001	123.440002	121.860001	123.019997	123.019997	29542900

1236 rows × 6 columns

In [8]:
```
training_set
```

Out[8]:
```
array([[ 56.66049957],
       [ 56.59180069],
       [ 56.1155014 ],
       ...,
       [123.41000366],
       [123.93499756],
       [123.44000244]])
```

In [7]:
```
print(training_set.shape)
```

```
(1257, 1)
```

In [9]:
```
### Feature Scaling
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(training_set)
```

In [11]:
```
#Creating a data structure with 60 timesteps and 1 output

X_train = []
y_train = []
for i in range(60, 1257):
    X_train.append(training_set_scaled[i-60:i, 0])
    y_train.append(training_set_scaled[i, 0])
X_train, y_train = np.array(X_train), np.array(y_train)
```

In [12]:
```
#Reshaping
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
```

In [13]:
```python
#Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
```

In [17]:
```python
regressor = Sequential()  #Initialising the RNN

#Adding the LSTM layers
regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))

#Adding the output layer
regressor.add(Dense(units = 1))

#compiling
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

In [18]:
```python
#Fitting the RNN to the Training set
regressor.fit(X_train, y_train, epochs = 100, batch_size = 32)
```

```
Epoch 1/100
38/38 [==============================] - 15s 109ms/step - loss: 0.0397
Epoch 2/100
38/38 [==============================] - 4s 111ms/step - loss: 0.0062
Epoch 3/100
38/38 [==============================] - 5s 120ms/step - loss: 0.0057
Epoch 4/100
38/38 [==============================] - 5s 118ms/step - loss: 0.0056
Epoch 5/100
38/38 [==============================] - 4s 115ms/step - loss: 0.0049
Epoch 6/100
38/38 [==============================] - 5s 128ms/step - loss: 0.0046
Epoch 7/100
38/38 [==============================] - 5s 124ms/step - loss: 0.0047
Epoch 8/100
38/38 [==============================] - 4s 111ms/step - loss: 0.0051
Epoch 9/100
38/38 [==============================] - 4s 109ms/step - loss: 0.0053
Epoch 10/100
```

In [35]:
```python
# Getting the real stock price for test set
end=datetime.now()
en = datetime(end.year,end.month,end.day)
start = datetime(end.year,end.month-1,end.day)

dataset_test = yf.download("GOOG",start,en)
print('GOT GOOGLES DATA')
real_stock_price = dataset_test.iloc[:, 1:2].values
```

```
[*********************100%***********************]  1 of 1 completed
GOT GOOGLES DATA
```

In [37]:
```
dataset_test
```

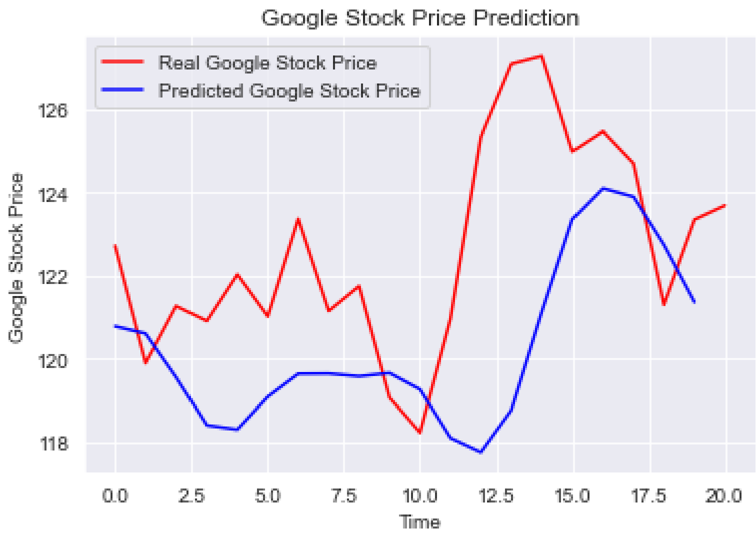
Out[37]:

Date	Open	High	Low	Close	Adj Close	Volume
2023-06-26	121.466003	122.720001	118.989998	119.089996	119.089996	23185000
2023-06-27	117.839996	119.894997	116.910004	119.010002	119.010002	27221700
2023-06-28	117.959999	121.269997	117.599998	121.080002	121.080002	19753100
2023-06-29	120.089996	120.910004	119.209999	120.010002	120.010002	18517500
2023-06-30	121.099998	122.029999	120.879997	120.970001	120.970001	23865800
2023-07-03	120.320000	121.019997	119.705002	120.559998	120.559998	13888300
2023-07-05	120.059998	123.370003	120.059998	122.629997	122.629997	17830300
2023-07-06	120.639999	121.150002	119.250000	120.930000	120.930000	17732500
2023-07-07	120.889999	121.750000	120.089996	120.139999	120.139999	20982400
2023-07-10	119.070000	119.070000	116.639999	116.870003	116.870003	32960100
2023-07-11	116.760002	118.224998	115.830002	117.709999	117.709999	18286600
2023-07-12	119.300003	120.959999	119.000000	119.620003	119.620003	22059600
2023-07-13	121.540001	125.334999	121.059998	124.830002	124.830002	31535900
2023-07-14	125.129997	127.089996	124.900002	125.699997	125.699997	20482800
2023-07-17	126.059998	127.279999	124.500000	125.059998	125.059998	20675300
2023-07-18	124.904999	124.989998	123.300003	124.080002	124.080002	21071200
2023-07-19	124.790001	125.470001	122.470001	122.779999	122.779999	22313800
2023-07-20	122.120003	124.699997	118.684998	119.529999	119.529999	27541700
2023-07-21	120.870003	121.300003	119.070000	120.309998	120.309998	56498100
2023-07-24	121.926003	123.349998	121.379997	121.879997	121.879997	22276100
2023-07-25	121.879997	123.690002	121.529999	122.790001	122.790001	31252200

In [38]:
```
#Getting the predicted stock price

dataset_total = pd.concat((dataset_train['Open'], dataset_test['Open']), axis = 0)
inputs = dataset_total[len(dataset_total) - len(dataset_test) - 60:].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)
X_test = []
for i in range(60, 80):
    X_test.append(inputs[i-60:i, 0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
predicted_stock_price = regressor.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

```
1/1 [==============================] - 3s 3s/step
```

In [39]:
```python
#Visualising the results

plt.plot(real_stock_price, color = 'red', label = 'Real Google Stock Price')
plt.plot(predicted_stock_price, color = 'blue', label = 'Predicted Google Stock Price
plt.title('Google Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Google Stock Price')
plt.legend()
plt.show()
```



In [42]:
```python
print(len(real_stock_price))
print(len(predicted_stock_price))
```

```
21
20
```

In [44]:
```python
real_stock_price
```

Out[44]:
```
array([[122.72000122],
       [119.89499664],
       [121.26999664],
       [120.91000366],
       [122.02999878],
       [121.01999664],
       [123.37000275],
       [121.15000153],
       [121.75      ],
       [119.06999969],
       [118.22499847],
       [120.95999908],
       [125.33499908],
       [127.08999634],
       [127.27999878],
       [124.98999786],
       [125.47000122],
       [124.69999695],
       [121.30000305],
       [123.34999847],
       [123.69000244]])
```

In [45]:
```
predicted_stock_price
```

Out[45]:
```
array([[120.783   ],
       [120.608925],
       [119.5601  ],
       [118.40214 ],
       [118.29909 ],
       [119.09161 ],
       [119.64357 ],
       [119.64821 ],
       [119.581375],
       [119.661476],
       [119.26773 ],
       [118.09787 ],
       [117.75654 ],
       [118.76369 ],
       [121.12555 ],
       [123.36107 ],
       [124.10214 ],
       [123.9015  ],
       [122.74122 ],
       [121.365295]], dtype=float32)
```