SID : 16103104

NAME: Lovedeep Singh

BIG DATA ANALYTICS

Stock Market Analysis and Prediction

Here data is collected from past one year till today, as there is limit of number of queries in free services provided by IEX Cloud. start date is 19 Nov 2018 and end date is 18 Nov 2019.

Stock Market Data From IEX Cloud

Introduction

Stock Market Analysis and Prediction is the project on technical analysis, visualization and prediction using data provided by IEX Cloud. By looking at data from the stock market, particularly some giant technology stocks and others. Used pandas to get stock information, visualize different aspects of it, and finally looked at a few ways of analyzing the risk of a stock, based on its previous performance history. Predicted future stock prices through a Monte Carlo method!

NOTE : you need to have a API key to access data at IEX Cloud to run this notebook, you can obtain it through IEX Cloud account setup.

Questions

In this analysis, I would like to explore the following questions.

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' closing prices?
5. What was the correlation between different stocks' daily returns?
6. How much value do we put at risk by investing in a particular stock?
7. How can we attempt to predict future stock behavior?

In [24]:
```
# For Data Processing
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
import os as os

# Data Visualization
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
%matplotlib inline
```

In [18]:
```
conda install pandas-datareader
```

```
Collecting package metadata (current_repodata.json): done
Solving environment: -
The environment is inconsistent, please check the package plan car
efully
The following packages are causing the inconsistency:

  - defaults/osx-64::anaconda==2019.07=py37_0
  - defaults/osx-64::numba==0.44.1=py37h6440ff4_0
done

## Package Plan ##

  environment location: /Users/lovedeepsingh/anaconda3

  added / updated specs:
    - pandas-datareader


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    _anaconda_depends-2019.03  |           py37_0           6 KB
    anaconda-custom            |           py37_1           3 KB
    ca-certificates-2019.10.16 |                0         131 KB
    certifi-2019.9.11          |           py37_0         154 KB
    conda-4.7.12               |           py37_0         3.0 MB
    openssl-1.1.1d             |       h1de35cc_3         3.4 MB
    pandas-datareader-0.8.1    |             py_0          71 KB
    tbb-2019.8                 |       h04f5b5a_0         149 KB
    ------------------------------------------------------------
                                           Total:         6.9 MB

The following NEW packages will be INSTALLED:

  _anaconda_depends  pkgs/main/osx-64::_anaconda_depends-2019.03-p
y37_0
  pandas-datareader  pkgs/main/noarch::pandas-datareader-0.8.1-py_
```

```
0
  tbb                pkgs/main/osx-64::tbb-2019.8-h04f5b5a_0

The following packages will be UPDATED:

  ca-certificates                                2019.5.15-0 --> 20
19.10.16-0
  certifi                                    2019.6.16-py37_0 --> 20
19.9.11-py37_0
  openssl                                 1.1.1c-h1de35cc_1 --> 1.
1.1d-h1de35cc_3

The following packages will be SUPERSEDED by a higher-priority cha
nnel:

  conda                                          conda-forge --> pk
gs/main

The following packages will be DOWNGRADED:

  anaconda                                   2019.07-py37_0 --> cu
stom-py37_1



Downloading and Extracting Packages
tbb-2019.8           | 149 KB    | ##################################
###### | 100%
anaconda-custom      | 3 KB      | ##################################
###### | 100%
pandas-datareader-0. | 71 KB     | ##################################
###### | 100%
_anaconda_depends-20 | 6 KB      | ##################################
###### | 100%
conda-4.7.12         | 3.0 MB    | ##################################
###### | 100%
openssl-1.1.1d       | 3.4 MB    | ##################################
###### | 100%
certifi-2019.9.11    | 154 KB    | ##################################
###### | 100%
ca-certificates-2019 | 131 KB    | ##################################
###### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done

Note: you may need to restart the kernel to use updated packages.
```

In [25]:
```
# For reading stock data from yahoo
from pandas_datareader import DataReader

# For time stamps
from datetime import datetime

# For division
exec('from __future__ import division')
```

In [28]:
```
os.environ["IEX_API_KEY"] = "pk_c46ca269b55e442e8799174e19e93a74"
```

Section 1 - Basic Analysis of Stock Information

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

In [30]:
```
# List of Tech_stocks for analytics
tech_list = ['AAPL','GOOGL','MSFT','AMZN']

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

#For-loop for grabing google finance data and setting as a datafram
e
# Set DataFrame as the Stock Ticker

for stock in tech_list:
    globals()[stock] = DataReader(stock,'iex',start,end)
```

Quick note: Using globals() is a sloppy way of setting the DataFrame names, but its simple

Let's go ahead and play aorund with the AAPL(Apple) Stock DataFrame to get a feel for the data.

In [31]:
```
AAPL.head()
```

Out[31]:

date	open	high	low	close	volume
2018-11-19	190.00	190.70	184.99	185.86	41920872
2018-11-20	178.37	181.47	175.51	176.98	67825247
2018-11-21	179.73	180.27	176.55	176.78	31124210
2018-11-23	174.94	176.60	172.10	172.29	23623972
2018-11-26	174.24	174.95	170.26	174.62	44998520

In [32]:
```
# Summery stats for Apple Stock
AAPL.describe()
```

Out[32]:

	open	high	low	close	volume
count	250.000000	250.000000	250.000000	250.000000	2.500000e+02
mean	195.842120	197.784040	194.068720	196.032960	3.030626e+07
std	26.675518	26.720691	26.809741	26.874866	1.250439e+07
min	143.980000	145.720000	142.000000	142.190000	1.136204e+07
25%	174.737500	175.902500	173.362500	174.757500	2.159191e+07
50%	197.880000	199.705000	196.070000	198.185000	2.697120e+07
75%	209.295000	211.540000	207.305000	209.000000	3.576799e+07
max	263.750000	265.780000	263.010000	265.760000	9.574438e+07

In [33]:
```
# General Info
AAPL.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 250 entries, 2018-11-19 to 2019-11-15
Data columns (total 5 columns):
open      250 non-null float64
high      250 non-null float64
low       250 non-null float64
close     250 non-null float64
volume    250 non-null int64
dtypes: float64(4), int64(1)
memory usage: 11.7+ KB
```

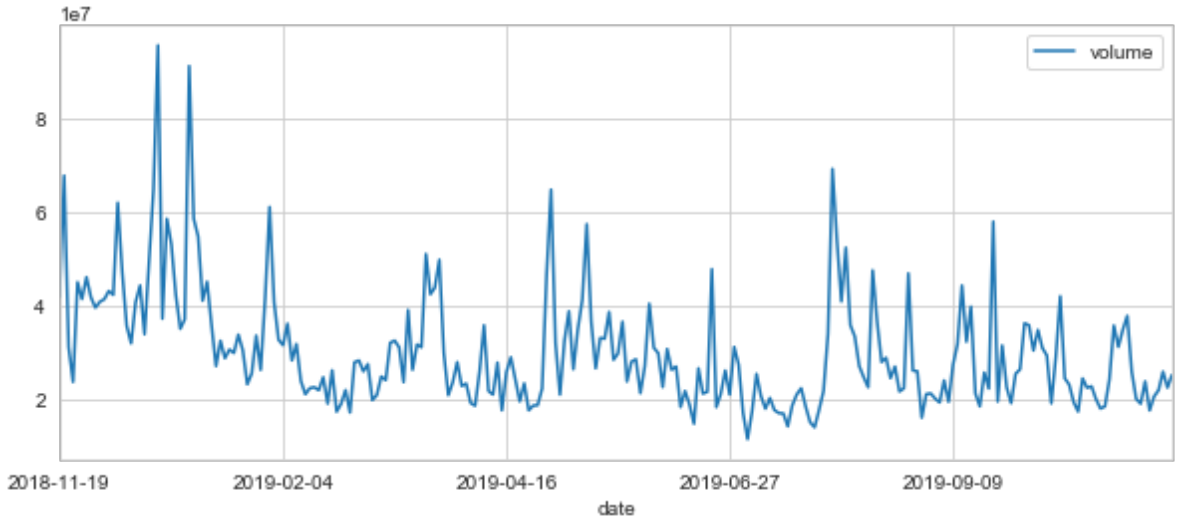
Now that we've seen the DataFrame, let's go ahead and plot out the volume and closing price of the AAPL(Apple) stocks.

In [36]:
```
# Let's see a historical view of the closing price
AAPL['close'].plot(legend=True, figsize=(10,4))
```

Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1ac68828>

In [38]:
```
# Now let's plot the total volume of stock being traded each day ov
er the past year

AAPL['volume'].plot(legend=True, figsize=(10,4))
```

Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1b4fbb00>



We can see that on Nov'2018 was the higher for AAPL stock being traded.

Now that we've seen the visualizations for the closing price and the volume traded each day for AAPL stock. Let's go ahead and caculate the moving average for the AAPL stock.

For more info on the Moving Average(SMA & EMA) check out the following links:

1.) http://www.investopedia.com/terms/m/movingaverage.asp (http://www.investopedia.com/terms/m/movingaverage.asp)

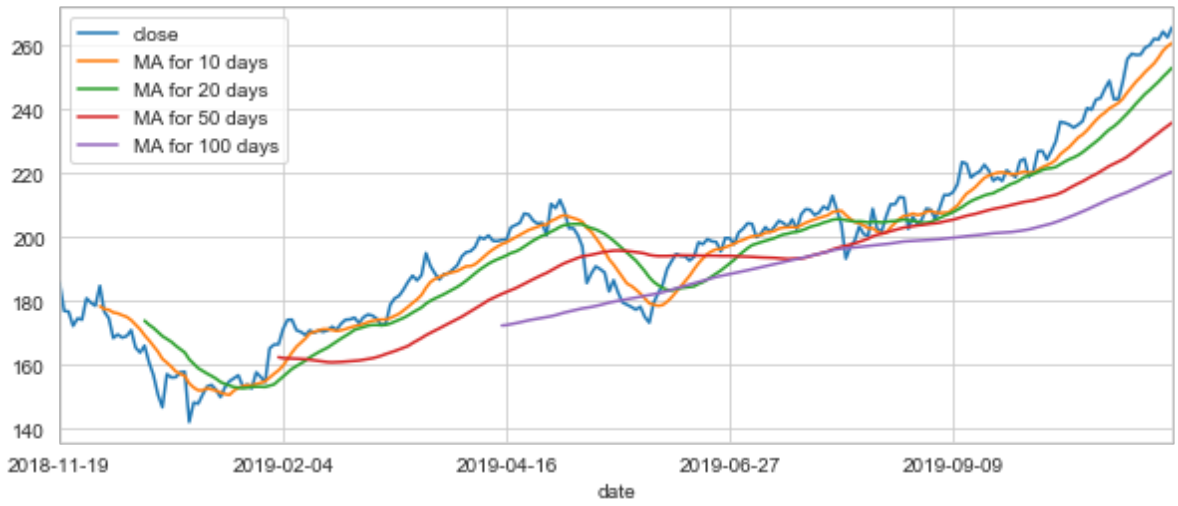
2.) http://www.investopedia.com/articles/active-trading/052014/how-use-moving-average-buy-stocks.asp (http://www.investopedia.com/articles/active-trading/052014/how-use-moving-average-buy-stocks.asp)

In [42]:
```
# Pandas has a built-in rolling mean calculator

# Let's go ahead and plot out several moving averages
MA_day = [10,20,50,100]

for ma in MA_day:
    column_name = 'MA for %s days' %(str(ma))
    AAPL[column_name] = AAPL['close'].rolling(ma).mean()
```

Now, lets plot all the additional Moving Averages for AAPL stock

In [43]:
```
AAPL[['close','MA for 10 days','MA for 20 days','MA for 50 days','MA for 100 days']].plot(subplots=False,figsize=(10,4))
```

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1b4fb390>



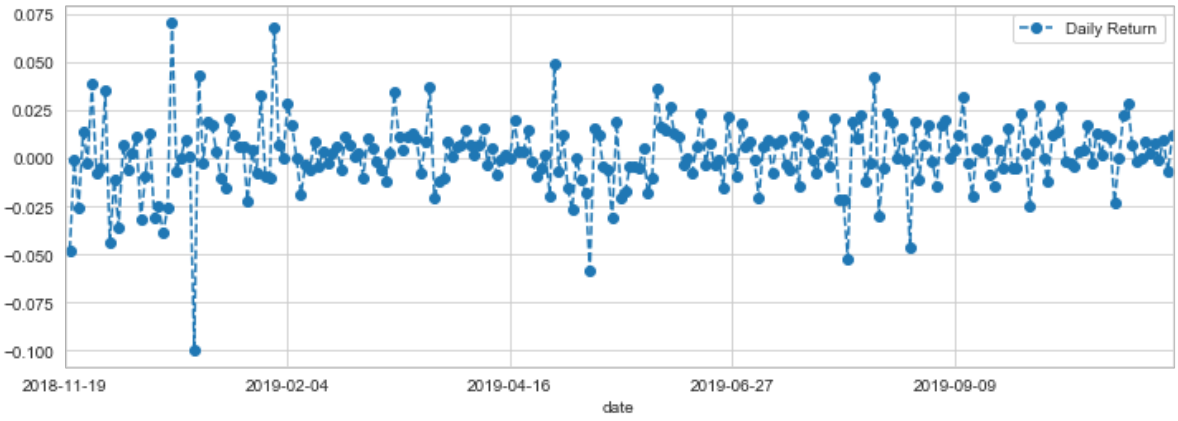
Section 2 - Daily Return Analysis

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 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 APPL stock.

In [45]:
```
# We'll use pct_change to find the percent change for each day
AAPL['Daily Return'] = AAPL['close'].pct_change()

# Lets plot the daily return percentage
AAPL['Daily Return'].plot(figsize=(12,4), legend=True, linestyle='--', marker='o')
```

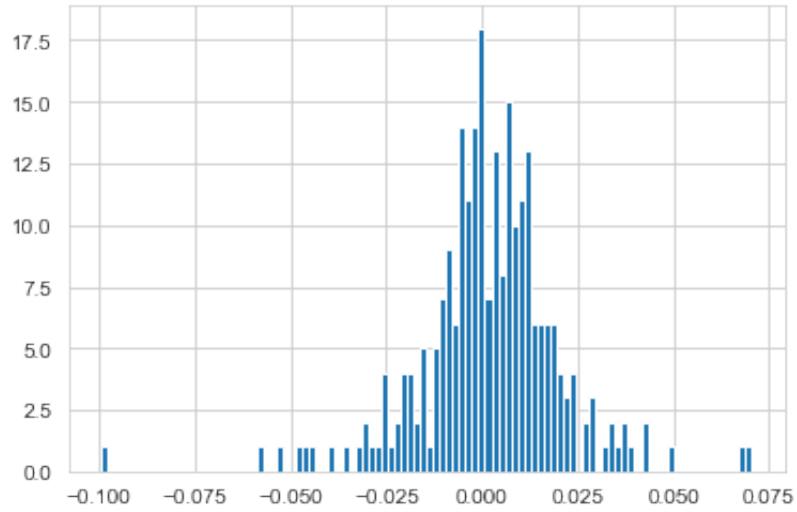
Out[45]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x105a24b00>
```



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

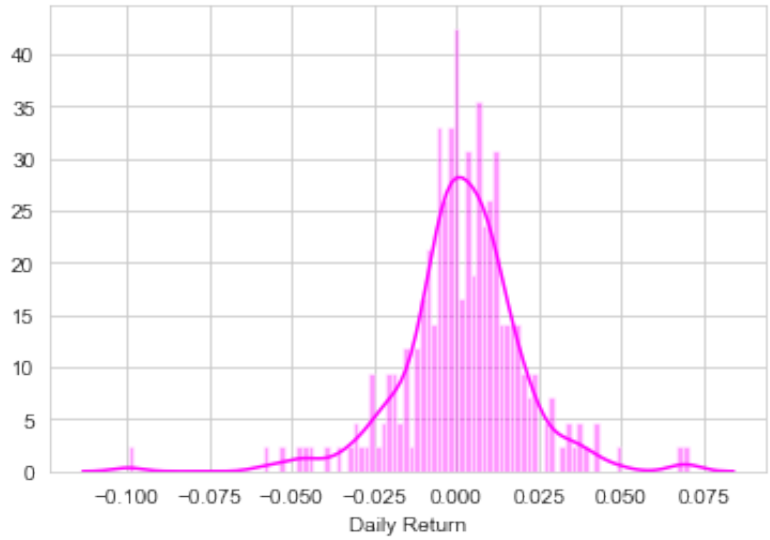
In [46]:
```
# only with histogram
AAPL['Daily Return'].hist(bins=100)
```

Out[46]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1ba120f0>



In [47]:
```
# Note the use of dropna() here, otherwise the NaN values can't be
read by seaborn
sns.distplot(AAPL['Daily Return'].dropna(), bins=100, color='magent
a')
```

Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1bb5bfd0>



Now what if we wanted to analyze the returns of all the stocks in our list? For that, we need to build a DataFrame with all the ['Close'] columns for each of the stocks dataframes.

In [49]:
```
# Grab all the closing prices for the tech stock list into one Data
Frame

closingprice_df = DataReader(tech_list, 'iex', start, end)['close']
```

In [50]:
```
closingprice_df.head(10)
```

Out[50]:

Symbols	AAPL	AMZN	GOOGL	MSFT
date				
2018-11-19	185.86	1512.29	1027.42	104.62
2018-11-20	176.98	1495.46	1030.45	101.71
2018-11-21	176.78	1516.73	1043.43	103.11
2018-11-23	172.29	1502.06	1030.10	103.07
2018-11-26	174.62	1581.33	1055.94	106.47
2018-11-27	174.24	1581.42	1052.28	107.14
2018-11-28	180.94	1677.75	1091.79	111.12
2018-11-29	179.55	1673.57	1094.58	110.19
2018-11-30	178.58	1690.17	1109.65	110.89
2018-12-03	184.82	1772.36	1116.36	112.09

Now that we have all the closing prices, let's go ahead and get the daily return for all the stocks, like we did for the APPL stock.

In [51]:
```
# make a new tech returns DataFrame
tech_returns = closingprice_df.pct_change()
```

In [52]:
```
tech_returns.head()
```

Out[52]:

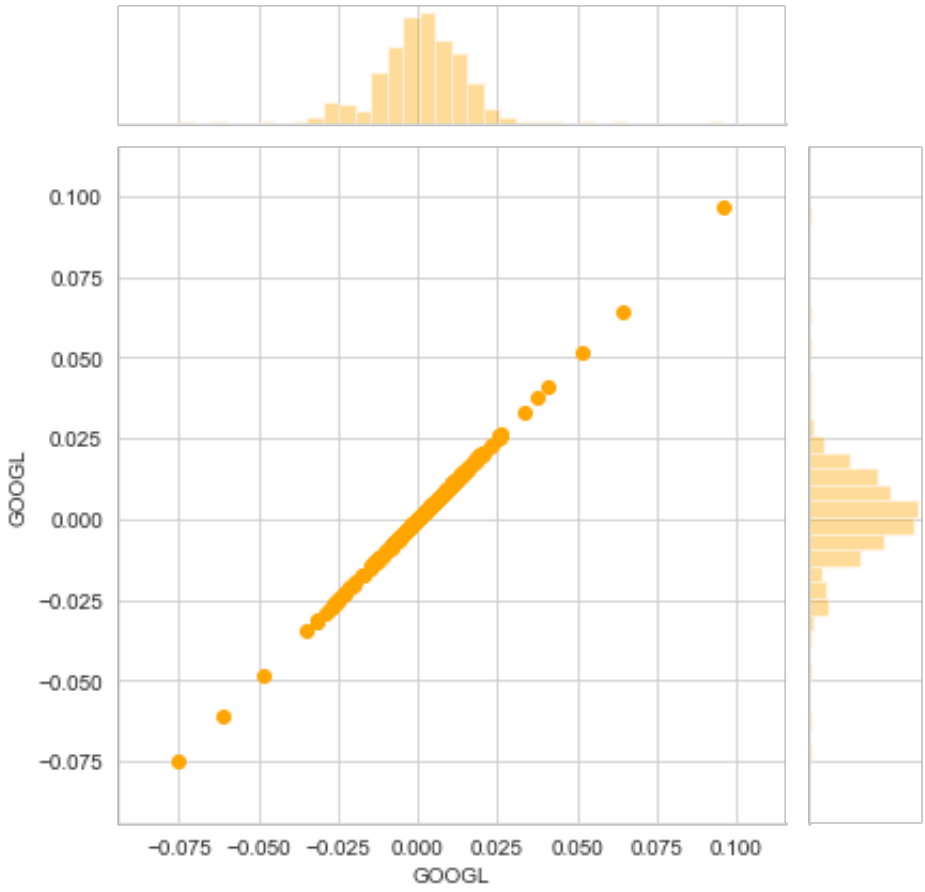
Symbols	AAPL	AMZN	GOOGL	MSFT
date				
2018-11-19	NaN	NaN	NaN	NaN
2018-11-20	-0.047778	-0.011129	0.002949	-0.027815
2018-11-21	-0.001130	0.014223	0.012596	0.013765
2018-11-23	-0.025399	-0.009672	-0.012775	-0.000388
2018-11-26	0.013524	0.052774	0.025085	0.032987

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

GOOGL is a Alphabet Inc Class A Stock.

In [53]:
```
# Comparing Google to itself should show a perfectly linear relationship
sns.jointplot('GOOGL','GOOGL',tech_returns,kind='scatter',color='orange')
```

Out[53]: <seaborn.axisgrid.JointGrid at 0x1a1be70198>



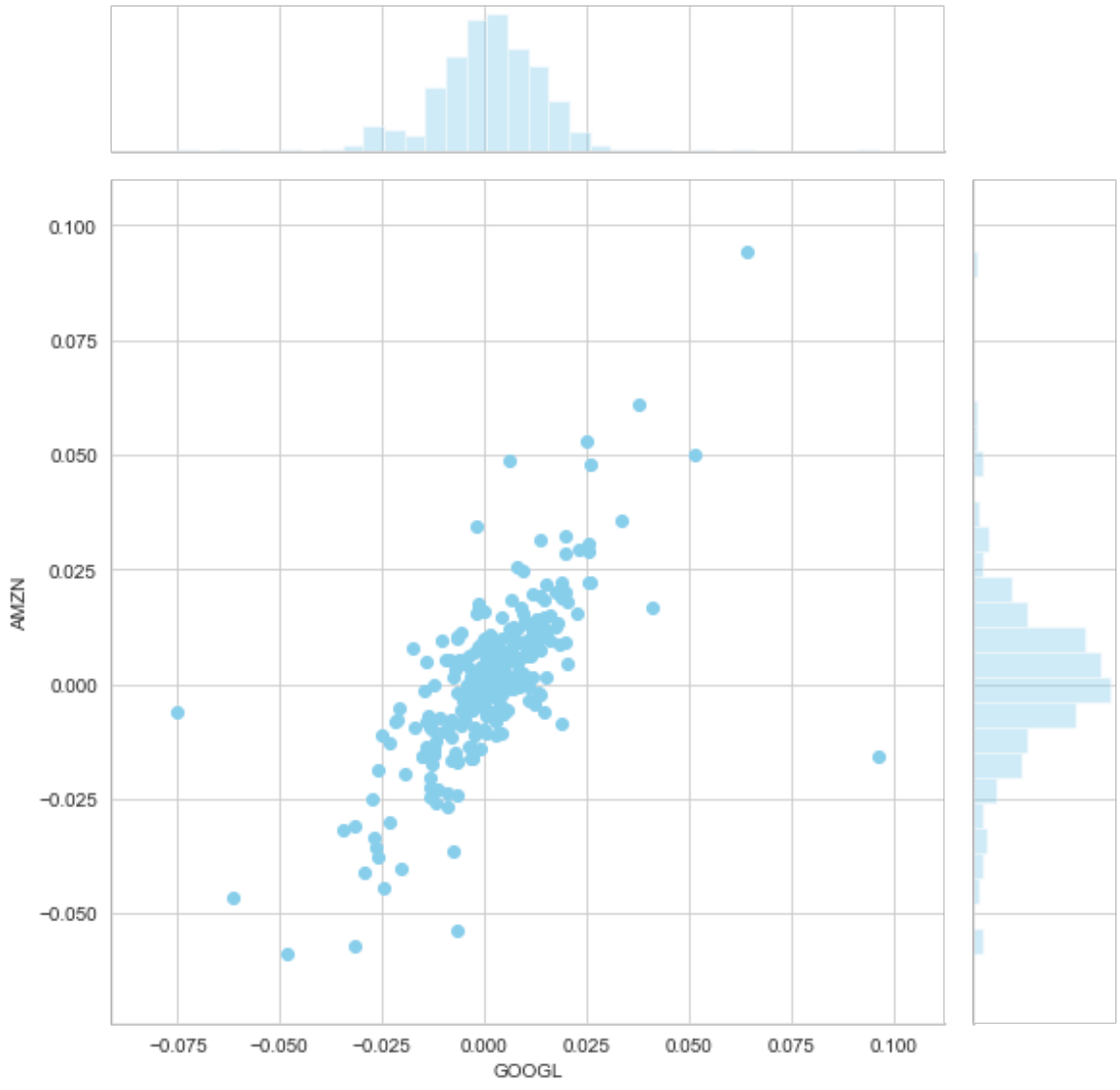
So now we can see that if two stocks are perfectly (and positivley) correlated with each other a linear relationship bewteen its daily return values should occur.

So let's go ahead and compare Google and Amazon the same way.

In [54]:
```
# We'll use joinplot to compare the daily returns of Google and Ama
zon.

sns.jointplot('GOOGL','AMZN',tech_returns, kind='scatter',size=8, c
olor='skyblue')
```

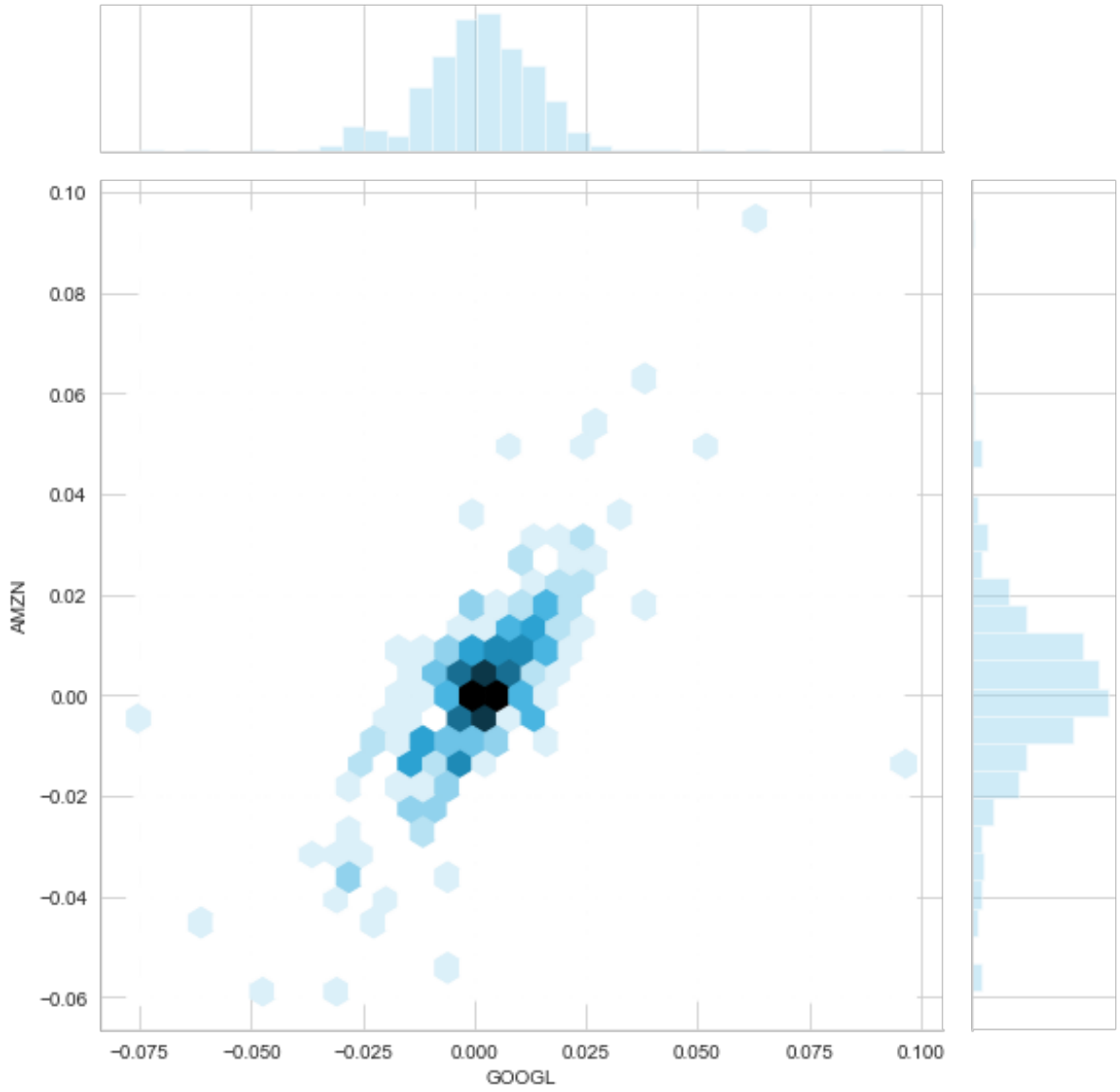
```
/Users/lovedeepsingh/anaconda3/lib/python3.7/site-packages/seaborn
/axisgrid.py:2262: UserWarning: The `size` paramter has been renam
ed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```

Out[54]: <seaborn.axisgrid.JointGrid at 0x1a1c0db6d8>



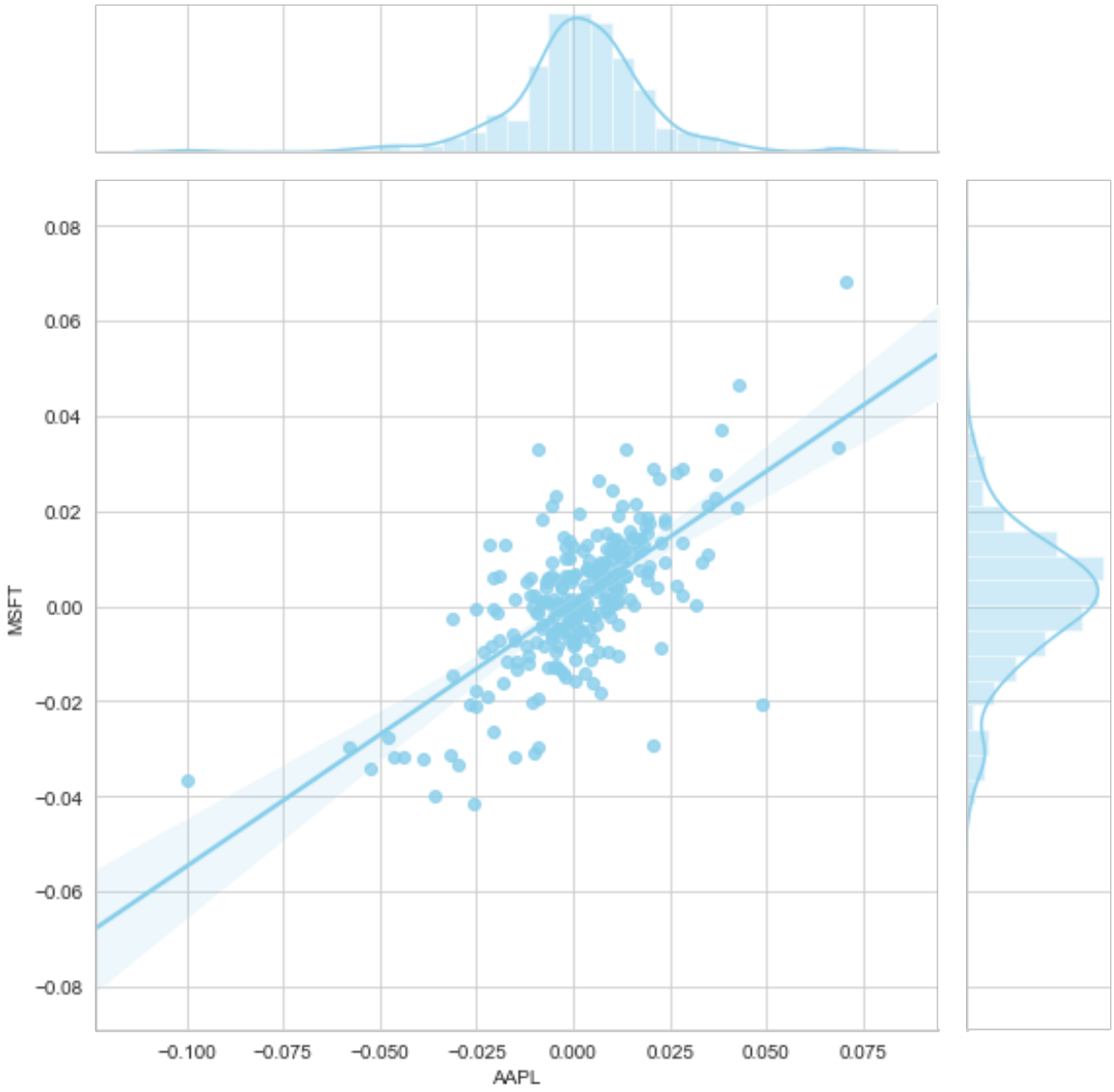
In [55]:
```
# with Hex plot
sns.jointplot('GOOGL','AMZN',tech_returns, kind='hex',size=8, color
='skyblue')
```

Out[55]: <seaborn.axisgrid.JointGrid at 0x1a1c5c0c50>



In [56]:
```
# Lets check out for Apple and Microsoft with reg jointplot
sns.jointplot('AAPL','MSFT',tech_returns, kind='reg', size=8, color
='skyblue')
```

Out[56]: <seaborn.axisgrid.JointGrid at 0x1a1c7fd320>



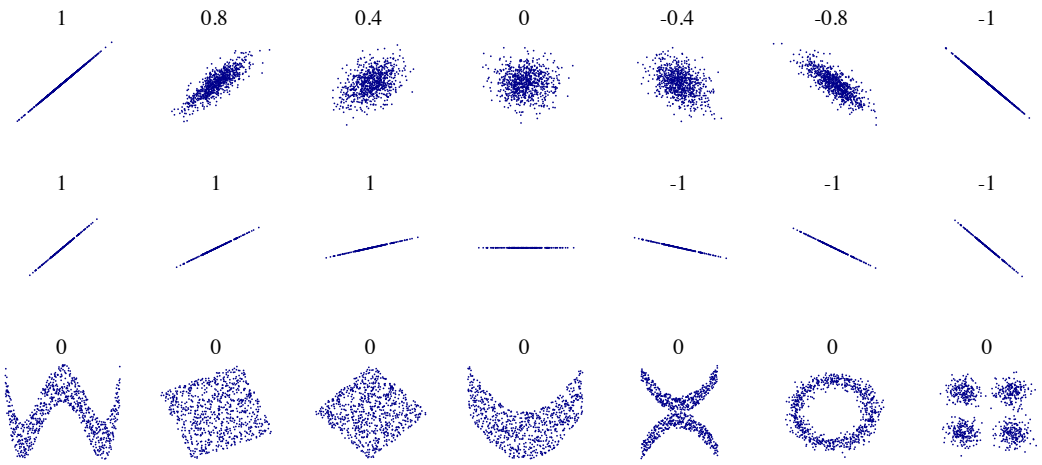
Intersting, the pearsonr value (officially known as the Pearson product-moment correlation coefficient) can give you a sense of how correlated the daily percentage returns are. You can find more information about it at this link:

Url - http://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient (http://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient)

But for a quick intuitive sense, check out the picture below.

In [57]:
```
from IPython.display import SVG
SVG(url='http://upload.wikimedia.org/wikipedia/commons/d/d4/Correlation_examples2.svg')
```

Out[57]:

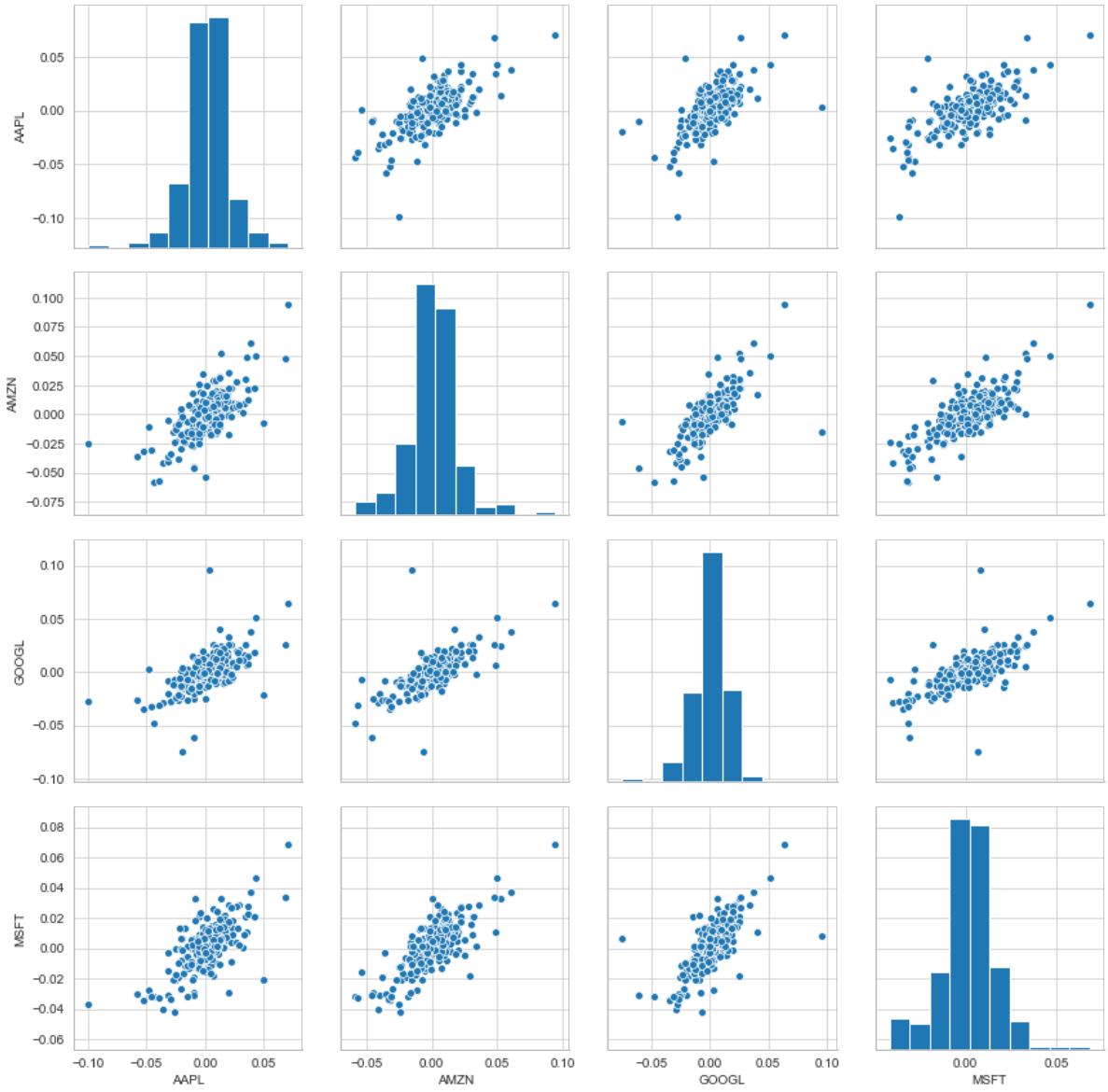


Seaborn and Pandas make it very easy to repeat this comparison analysis for every possible combination of stocks in our technology stock ticker list. We can use sns.pairplot() to automatically create this plot

In [58]:
```
# We can simply call pairplot on our DataFrame for an automatic visual analysis of all the comparisons
sns.pairplot(tech_returns.dropna(),size=3)
```

```
/Users/lovedeepsingh/anaconda3/lib/python3.7/site-packages/seaborn
/axisgrid.py:2065: UserWarning: The `size` parameter has been rena
med to `height`; pleaes update your code.
  warnings.warn(msg, UserWarning)
```

Out[58]:
```
<seaborn.axisgrid.PairGrid at 0x1a1fa43f60>
```

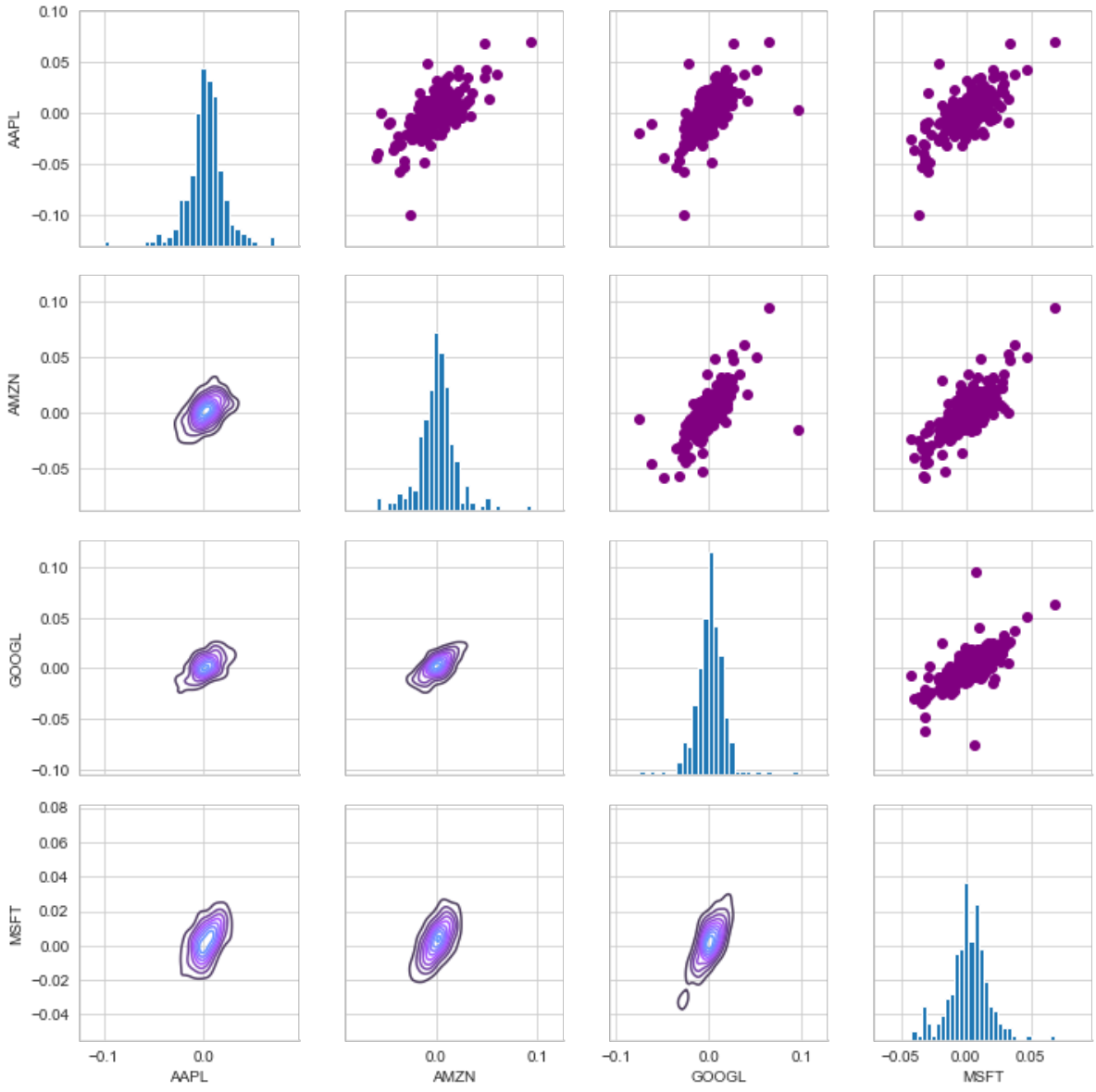


Above we can see all the relationships on daily returns between all the stocks. A quick glance shows an interesting correlation between Google and Amazon daily returns. It might be interesting to investigate that individual comaprison. While the simplicity of just calling sns.pairplot() is fantastic 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 [59]:
```
# Set up the figure by naming it returns_fig, call PairGrid on the DataFrame
returns_fig = sns.PairGrid(tech_returns.dropna())

# 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, including the plot type (kde) & 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)
```

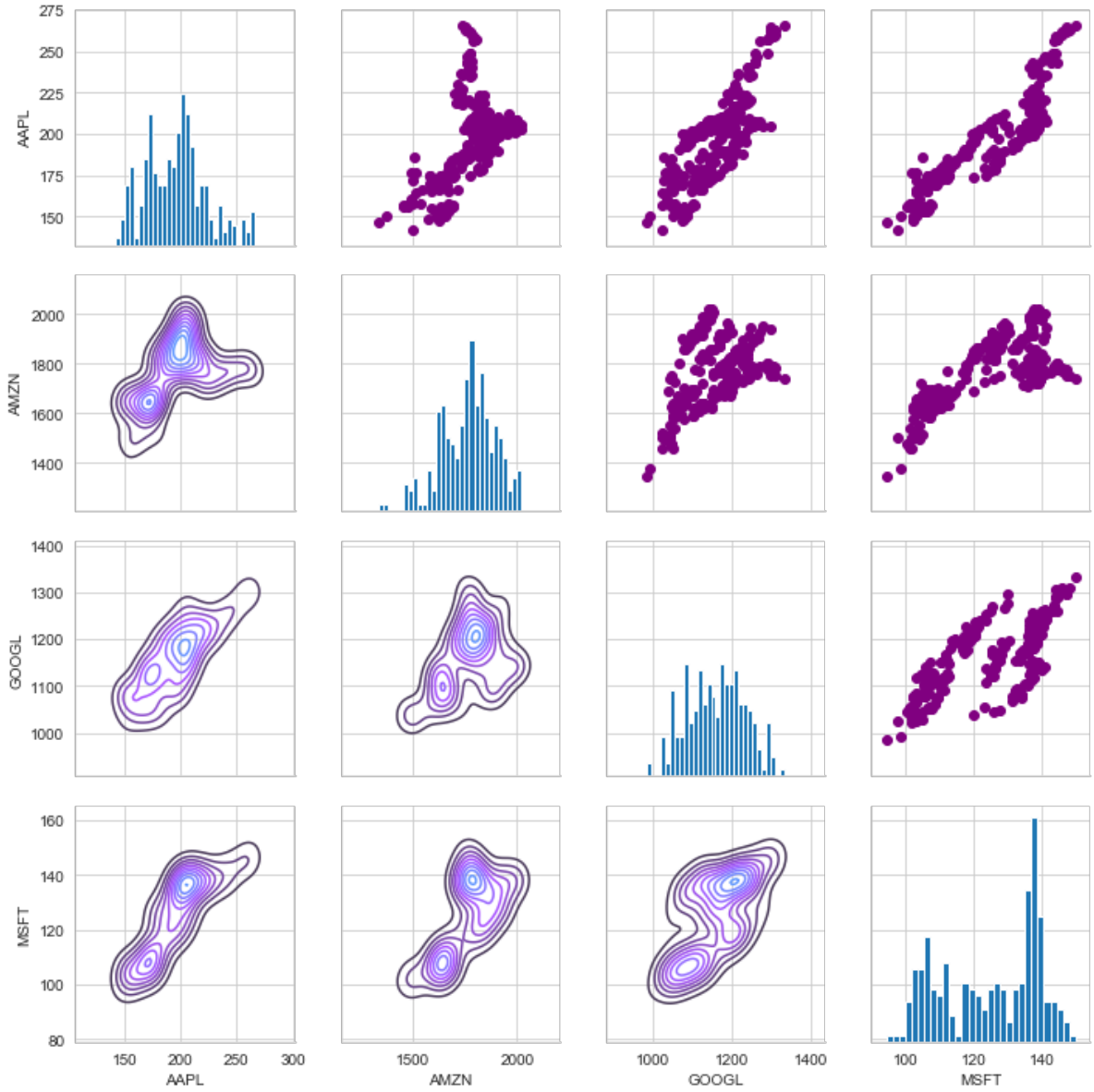
Out[59]: <seaborn.axisgrid.PairGrid at 0x1a1f970cc0>



We can also analyze the correlation of the closing prices using this exact same technique. Here it is shown, the code repeated from above with the exception of the DataFrame called.

In [60]:
```
# Set up the figure by naming it returns_fig, call PairGrid on the DataFrame
returns_fig = sns.PairGrid(closingprice_df.dropna())

# 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, including the plot type (kde) & 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)
```

Out[60]:
```
<seaborn.axisgrid.PairGrid at 0x1a1d990400>
```



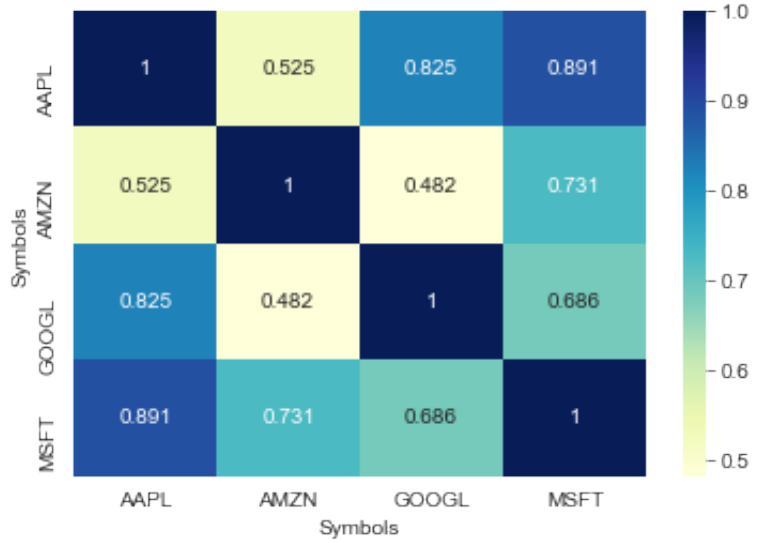
Finally, we can also do a correlation plot, to get actual numerical values for the correlation between the stocks' daily return values. By comparing the closing prices, we see an interesting relationship between Google and Amazon stocks.

In [61]:
```
# Let's go ahead and use seaborn for a quick heatmap to get correlation for the daily return of the stocks.
sns.heatmap(tech_returns.corr(),annot=True,fmt=".3g",cmap='YlGnBu')
```

Out[61]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1ec34668>

In [62]:
```
# Lets check out the correlation between closing prices of stocks
sns.heatmap(closingprice_df.corr(),annot=True,fmt=".3g",cmap='YlGnBu')
```

Out[62]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1f008710>



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

Great! Now that we've done some daily return analysis, let's go ahead and start looking deeper into actual risk analysis.

Risk Analysis

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(Risk).

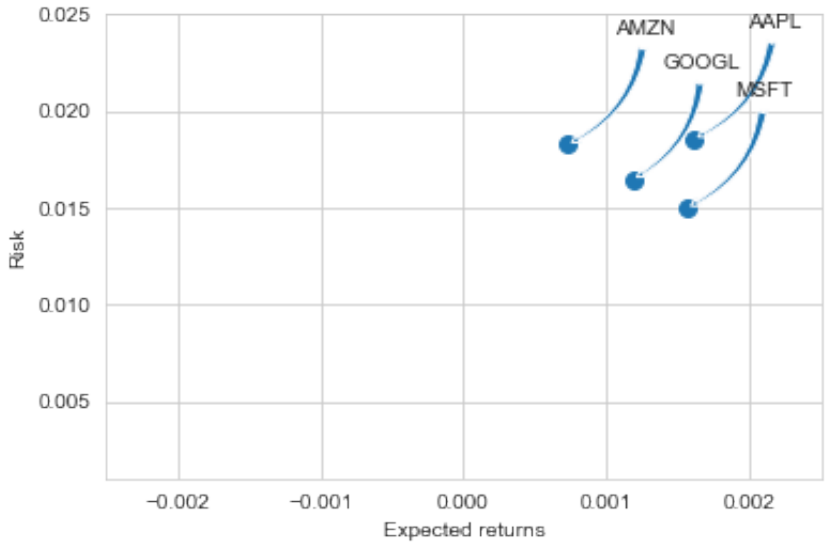
In [63]:
```
# Let's start by defining a new DataFrame as a clenaed version of t
he oriignal tech_returns DataFrame
rets = tech_returns.dropna()
```

In [64]:
```
rets.head()
```

Out[64]:

Symbols	AAPL	AMZN	GOOGL	MSFT
date				
2018-11-20	-0.047778	-0.011129	0.002949	-0.027815
2018-11-21	-0.001130	0.014223	0.012596	0.013765
2018-11-23	-0.025399	-0.009672	-0.012775	-0.000388
2018-11-26	0.013524	0.052774	0.025085	0.032987
2018-11-27	-0.002176	0.000057	-0.003466	0.006293

In [65]:
```
# Defining the area for the circles of scatter plot to avoid tiny l
ittle points
area = np.pi*20

plt.scatter(rets.mean(),rets.std(),s=area)

# Set the x and y limits of the plot (optional, remove this if you
don't see anything in your plot)
plt.xlim([-0.0025,0.0025])
plt.ylim([0.001,0.025])

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

# Label the scatter plots, for more info on how this is done, chekc
out the link below
# http://matplotlib.org/users/annotations_guide.html
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 = 'fancy', connectionstyle = '
arc3,rad=-0.3'))
```



By looking at the scatter plot we can say these stocks have lower risk and positive expected returns.

Value at Risk

Let's go ahead and define a value at risk parameter for our stocks. We can treat value at risk as the amount of money we could expect to lose (aka putting at risk) for a given confidence interval. There's several methods we can use for estimating a value at risk. Let's go ahead and see some of them in action.

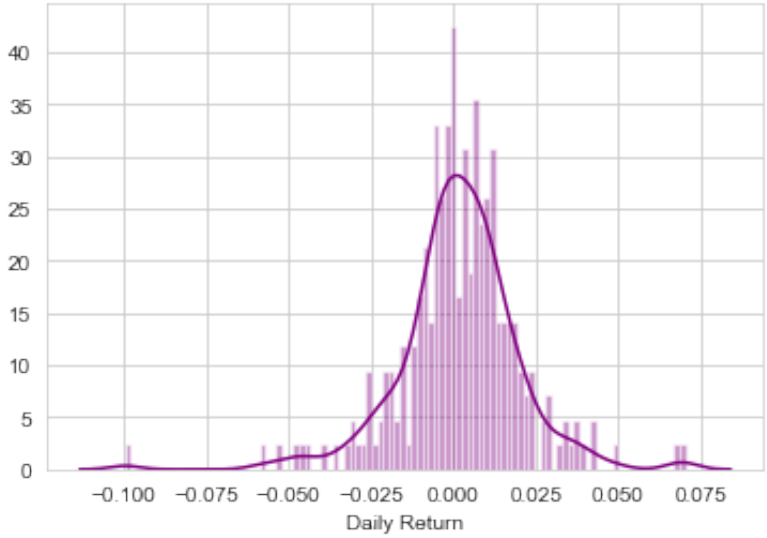
Value at risk using the "bootstrap" method

For this method we will calculate the empirical quantiles from a histogram of daily returns. For more information on quantiles, check out this link: http://en.wikipedia.org/wiki/Quantile (http://en.wikipedia.org/wiki/Quantile)

Let's go ahead and repeat the daily returns histogram for Apple stock.

In [66]:
```
# Note the use of dropna() here, otherwise the NaN values can't be
read by seaborn
sns.distplot(AAPL['Daily Return'].dropna(),bins=100,color='purple')
```

Out[66]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1f0d2e48>
```



Now we can use quantile to get the risk value for the stock.

In [67]:
```
# The 0.05 empirical quantile of daily returns

# For APPL stocks
rets["AAPL"].quantile(0.05)
```

Out[67]:
```
-0.026523845145744684
```

The 0.05 empirical quantile of daily returns is at -0.026. That means that with 95% confidence, our worst daily loss will not exceed 2.6%. If we have a 1 million dollar investment, our one-day 5% VaR is 0.026 * 1,000,000 = $26,000.

In [68]:
```
# For AMZN stocks
rets["AMZN"].quantile(0.05)
```

Out[68]: -0.030582822164480383

In [69]:
```
# For GOOGL stocks
rets["GOOGL"].quantile(0.05)
```

Out[69]: -0.025006930978543274

In [70]:
```
# For MSFT stocks
rets["MSFT"].quantile(0.05)
```

Out[70]: -0.029396628744491206

Value at Risk using the Monte Carlo 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 the stock is.

Let's start with a brief explanation of what we're going to do:

We will use the geometric Brownian motion (GBM), which is technically known as a Markov process. This means that the stock price follows a random walk and is consistent with (at the very least) the weak form of the efficient market hypothesis (EMH): past price information is already incorporated and the next price movement is "conditionally independent" of past price movements.

This means that the past information on the price of a stock is independent of where the stock price will be in the future, basically meaning, you can't perfectly predict the future solely based on the previous price of a stock.

Now we see that the change in the stock price is the current stock price multiplied by two terms. The first term is known as "drift", which is the average daily return multiplied by the change of time. The second term is known as "shock", for each time period the stock will "drift" and then experience a "shock" which will randomly push the stock price up or down. By simulating this series of steps of drift and shock thousands of times, we can begin to do a simulation of where we might expect the stock price to be.

For more info on the Monte Carlo method for stocks and simulating stock prices with GBM model ie. geometric Brownian motion (GBM).

check out the following link: http://www.investopedia.com/articles/07/montecarlo.asp (http://www.investopedia.com/articles/07/montecarlo.asp)

To demonstrate a basic Monte Carlo method, we will start with just a few simulations. First we'll define the variables we'll be using in the Google stock DataFrame GOOGL

In [71]:
```
rets.head()
```

Out[71]:

Symbols	AAPL	AMZN	GOOGL	MSFT
date				
2018-11-20	-0.047778	-0.011129	0.002949	-0.027815
2018-11-21	-0.001130	0.014223	0.012596	0.013765
2018-11-23	-0.025399	-0.009672	-0.012775	-0.000388
2018-11-26	0.013524	0.052774	0.025085	0.032987
2018-11-27	-0.002176	0.000057	-0.003466	0.006293

In [72]:
```
# Set up our time horizon
days = 365

# Now our delta
dt = 1/days

# Now let's grab our mu (drift) from the expected return data we go
t for GOOGL
mu = rets.mean()['GOOGL']

# Now let's grab the volatility of the stock from the std() of the
average return for GOOGL
sigma = rets.std()['GOOGL']
```

Next, we will create a function that takes in the starting price and number of days, and uses the sigma and mu we already calculated form our daily returns.

In [73]:
```
def stock_monte_carlo(start_price,days,mu,sigma):
    ''' This function takes in starting stock price, days of simula
tion,mu,sigma, and returns simulated price array'''

    # 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):

        # Calculate Schock
        shock[x] = np.random.normal(loc=mu * dt, scale=sigma * np.s
qrt(dt))
        # Calculate Drift
        drift[x] = mu * dt
        # Calculate Price
        price[x] = price[x-1] + (price[x-1] * (drift[x] + shock[x])
)

    return price
```

Awesome! Now lets put above function to work.

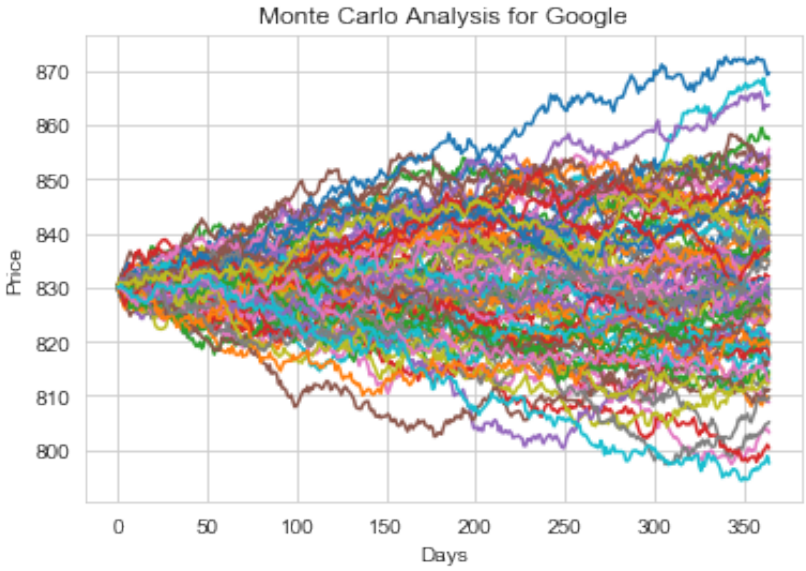
In [74]:
```
# For Google Stock - GOOGL
GOOGL.head()
```

Out[74]:

date	open	high	low	close	volume
2018-11-19	1063.39	1068.00	1022.87	1027.42	2284191
2018-11-20	1007.29	1037.35	1002.21	1030.45	2722934
2018-11-21	1045.31	1054.71	1039.46	1043.43	1485172
2018-11-23	1033.50	1043.00	1028.52	1030.10	708859
2018-11-26	1044.00	1057.00	1039.77	1055.94	1577941

In [75]:
```
start_price = 830.09

for run 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[75]:
```
Text(0.5, 1.0, 'Monte Carlo Analysis for Google')
```



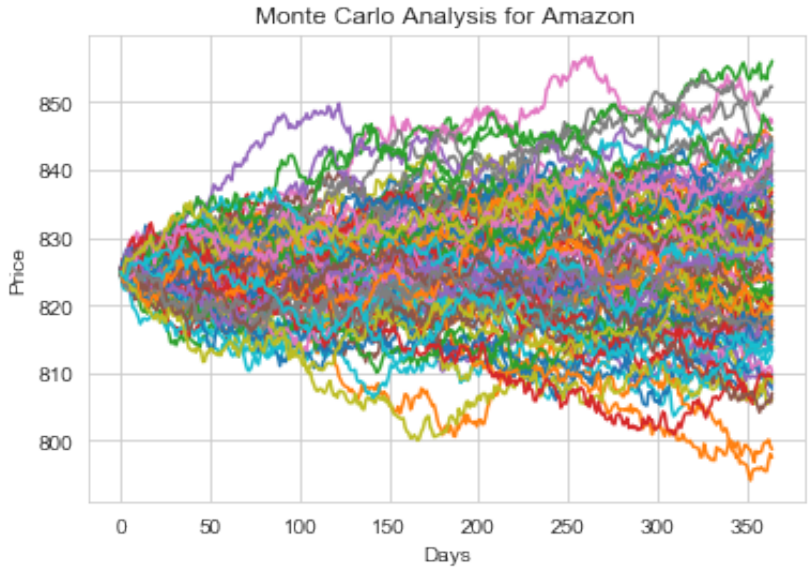
In [76]:
```
# For Amazon Stock - AMZN
AMZN.head()
```

Out[76]:

date	open	high	low	close	volume
2018-11-19	1577.01	1581.19	1503.36	1512.29	7789981
2018-11-20	1437.50	1534.75	1420.00	1495.46	10878823
2018-11-21	1542.99	1550.00	1515.00	1516.73	5716800
2018-11-23	1517.00	1536.20	1501.81	1502.06	2707642
2018-11-26	1539.00	1584.81	1524.22	1581.33	6257716

In [77]:
```
start_price = 824.95

for run 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 Amazon')
```

Out[77]:
```
Text(0.5, 1.0, 'Monte Carlo Analysis for Amazon')
```



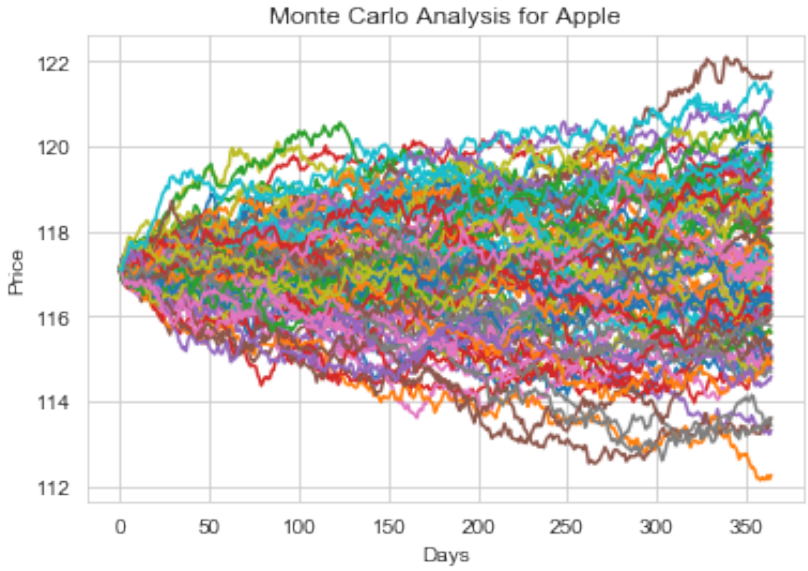
In [78]:
```
# For Apple Stock - AAPL
AAPL.head()
```

Out[78]:

date	open	high	low	close	volume	MA for 10 days	MA for 20 days	MA for 50 days	MA for 100 days	Daily Return
2018-11-19	190.00	190.70	184.99	185.86	41920872	NaN	NaN	NaN	NaN	NaN
2018-11-20	178.37	181.47	175.51	176.98	67825247	NaN	NaN	NaN	NaN	-0.047778
2018-11-21	179.73	180.27	176.55	176.78	31124210	NaN	NaN	NaN	NaN	-0.001130
2018-11-23	174.94	176.60	172.10	172.29	23623972	NaN	NaN	NaN	NaN	-0.025399
2018-11-26	174.24	174.95	170.26	174.62	44998520	NaN	NaN	NaN	NaN	0.013524

In [79]:
```
start_price = 117.10

for run 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 Apple')
```

Out[79]:
```
Text(0.5, 1.0, 'Monte Carlo Analysis for Apple')
```



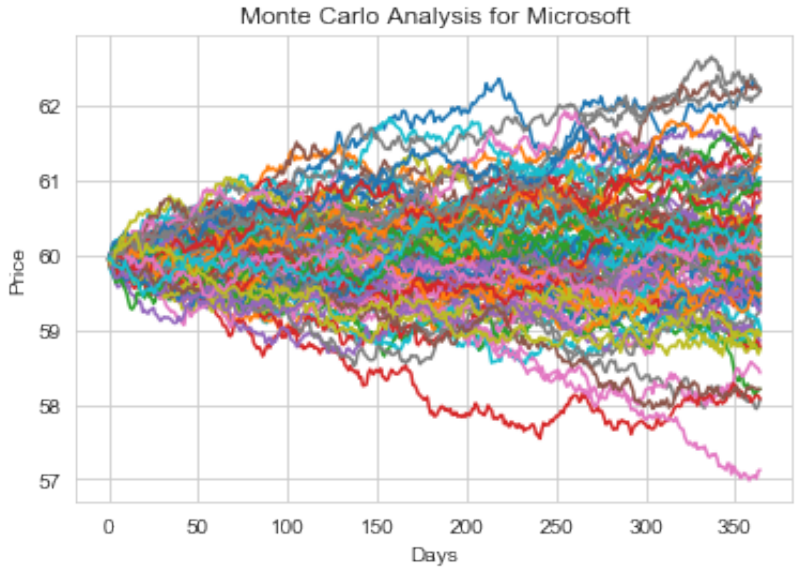
In [80]:
```
# For Microsoft Stock - MSFT
MSFT.head()
```

Out[80]:

date	open	high	low	close	volume
2018-11-19	108.27	108.56	103.55	104.62	44773899
2018-11-20	101.80	102.97	99.35	101.71	64052457
2018-11-21	103.60	104.43	102.24	103.11	28130621
2018-11-23	102.17	103.81	102.00	103.07	13823099
2018-11-26	104.79	106.63	104.58	106.47	32336165

In [81]:
```
start_price = 59.94

for run 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 Microsoft')
```

Out[81]:
```
Text(0.5, 1.0, 'Monte Carlo Analysis for Microsoft')
```



Let's go ahead and get a histogram of the end results for a much larger run. (note: This could take a little while to run , depending on the number of runs chosen)

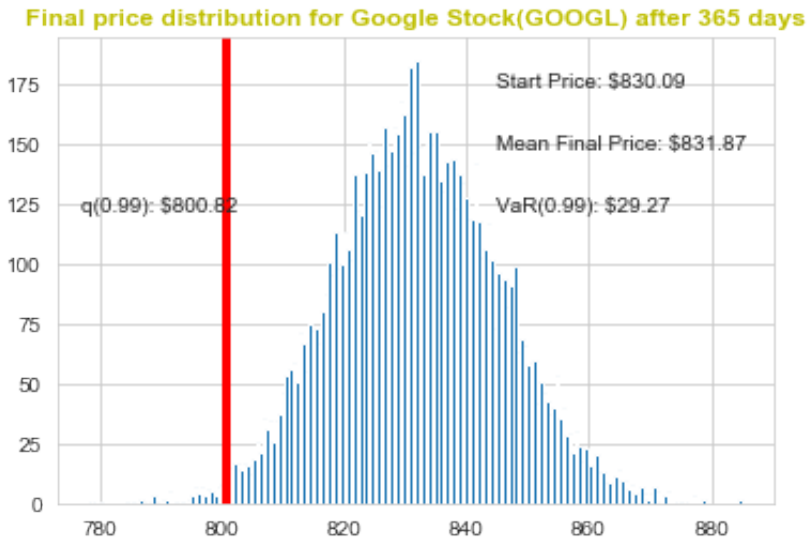
In [119]:
```
# Lets start with Google stock price
start_price = 830.09

# 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):
    # Set the simulation data point as the last stock price for tha
t run
    simulations[run] = stock_monte_carlo(start_price,days,mu,sigma)
[days-1]
```

Now that we have our array of simulations, we can go ahead and plot a histogram ,as well as use qunatile to define our risk for this stock.

For more info on quantiles, check out this link: http://en.wikipedia.org/wiki/Quantile (http://en.wikipedia.org/wiki/Quantile)

In [120]:
```
# Now we'll define q as the 1% empirical quantile, this basically m
eans that 99% of the values should fall between here
q = np.percentile(simulations,1)

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

# Using plt.figtext to fill in some additional information onto the
plot

# starting price
plt.figtext(0.6,0.8, s='Start Price: $%.2f' % start_price)

# mean ending price
plt.figtext(0.6,0.7, s='Mean Final Price: $%.2f' % simulations.mean
())

# Variance of the price (within 99% confidence interval)
plt.figtext(0.6,0.6, s='VaR(0.99): $%.2f' % (start_price - q))

# To display 1% quantile
plt.figtext(0.15, 0.6, s="q(0.99): $%.2f" % q)

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

# For plot title
plt.title(label="Final price distribution for Google Stock(GOOGL) a
fter %s days" % days, weight='bold', color='Y')
```

Out[120]: Text(0.5, 1.0, 'Final price distribution for Google Stock(GOOGL) a
fter 365 days')



Awesome! Now we have looked at the 1% empirical quantile of the final price distribution to estimate the Value at Risk for the Google Stock(GOOGL), which looks to be $29.27 for every investment of

830.09 (The price of one initial Google Stock).

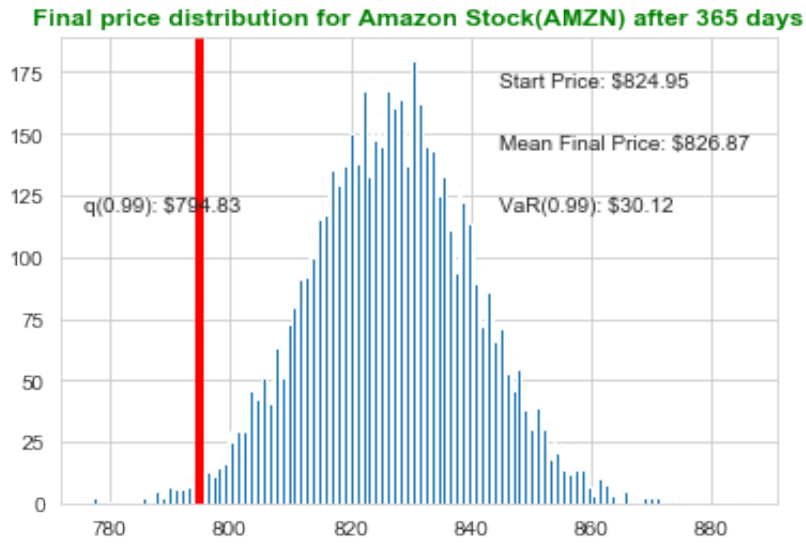
This basically means for every initial GOOGL stock you purchase you're putting about $29.27 at risk 99% of the time from our Monte Carlo Simulation.

Now lets plot remaining Stocks to estimate the VaR with our Monte Carlo Simulation.

In [121]:
```
# For Amazon Stock Price
start_price = 824.95

# 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):
    # Set the simulation data point as the last stock price for tha
t run
    simulations[run] = stock_monte_carlo(start_price,days,mu,sigma)
[days-1]
```

In [122]:
```
# Now we'll define q as the 1% empirical quantile, this basically m
eans that 99% of the values should fall between here
q = np.percentile(simulations,1)

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

# Using plt.figtext to fill in some additional information onto the
plot

# starting price
plt.figtext(0.6,0.8, s='Start Price: $%.2f' % start_price)

# mean ending price
plt.figtext(0.6,0.7, s='Mean Final Price: $%.2f' % simulations.mean
())

# Variance of the price (within 99% confidence interval)
plt.figtext(0.6,0.6, s='VaR(0.99): $%.2f' % (start_price - q))

# To display 1% quantile
plt.figtext(0.15, 0.6, s="q(0.99): $%.2f" % q)

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

# For plot title
plt.title(label="Final price distribution for Amazon Stock(AMZN) af
ter %s days" % days, weight='bold', color='G')
```

Out[122]:
```
Text(0.5, 1.0, 'Final price distribution for Amazon Stock(AMZN) af
ter 365 days')
```

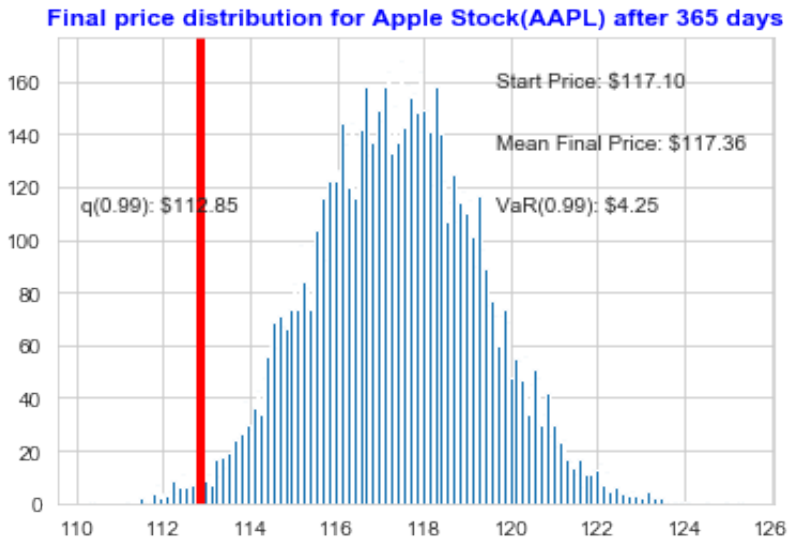


This basically means for every initial AMZN stock you purchase you're putting about $30.12 at risk 99% of the time from our Monte Carlo Simulation.

In [123]:
```
# For Apple Stock Price
start_price = 117.10

# 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):
    # Set the simulation data point as the last stock price for tha
t run
    simulations[run] = stock_monte_carlo(start_price,days,mu,sigma)
[days-1]
```

In [124]:
```
# Now we'll define q as the 1% empirical quantile, this basically m
eans that 99% of the values should fall between here
q = np.percentile(simulations,1)

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

# Using plt.figtext to fill in some additional information onto the
plot

# starting price
plt.figtext(0.6,0.8, s='Start Price: $%.2f' % start_price)

# mean ending price
plt.figtext(0.6,0.7, s='Mean Final Price: $%.2f' % simulations.mean
())

# Variance of the price (within 99% confidence interval)
plt.figtext(0.6,0.6, s='VaR(0.99): $%.2f' % (start_price - q))

# To display 1% quantile
plt.figtext(0.15, 0.6, s="q(0.99): $%.2f" % q)

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

# For plot title
plt.title(label="Final price distribution for Apple Stock(AAPL) aft
er %s days" % days, weight='bold', color='B')
```

Out[124]:
```
Text(0.5, 1.0, 'Final price distribution for Apple Stock(AAPL) aft
er 365 days')
```

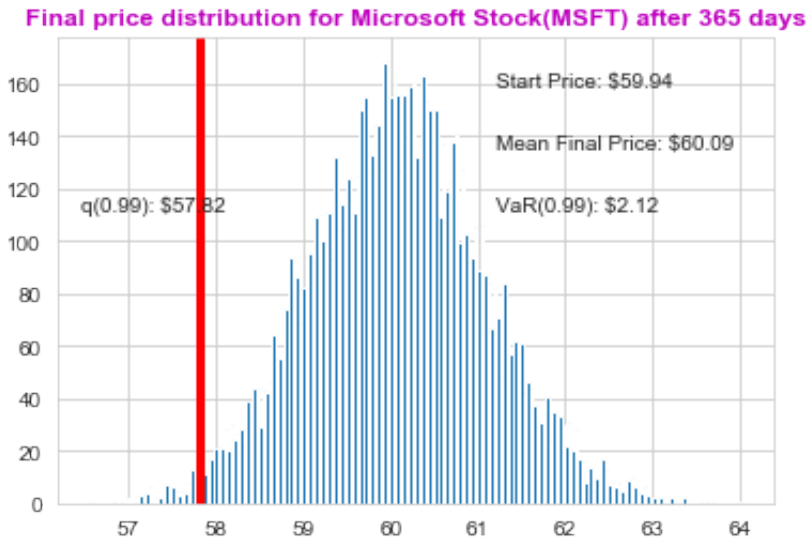


Great! This basically means for every initial AAPL stock you purchase you're putting about $4.25 at risk 99% of the time from our Monte Carlo Simulation.

In [125]:
```
# For Microsoft Stock Price
start_price = 59.94

# 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):
    # Set the simulation data point as the last stock price for tha
t run
    simulations[run] = stock_monte_carlo(start_price,days,mu,sigma)
[days-1]
```

In [94]:
```
# Now we'll define q as the 1% empirical quantile, this basically m
eans that 99% of the values should fall between here
q = np.percentile(simulations,1)

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

# Using plt.figtext to fill in some additional information onto the
plot

# starting price
plt.figtext(0.6,0.8, s='Start Price: $%.2f' % start_price)

# mean ending price
plt.figtext(0.6,0.7, s='Mean Final Price: $%.2f' % simulations.mean
())

# Variance of the price (within 99% confidence interval)
plt.figtext(0.6,0.6, s='VaR(0.99): $%.2f' % (start_price - q))

# To display 1% quantile
plt.figtext(0.15, 0.6, s="q(0.99): $%.2f" % q)

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

# For plot title
plt.title(label="Final price distribution for Microsoft Stock(MSFT)
after %s days" % days, weight='bold', color='M')
```

Out[94]: Text(0.5, 1.0, 'Final price distribution for Microsoft Stock(MSFT) after 365 days')



Nice, This basically means for every initial MSFT stock you purchase you're putting about $2.12 at risk 99% of the time from our Monte Carlo Simulation.

Now lets estiamte the Value at Risk(VaR) for a stock related to other domains.

We'll estimate the VaR for:

- Johnson & Johnson > JNJ (U.S.: NYSE) JNJ (http://quotes.wsj.com/JNJ)
- Wal-Mart Stores Inc. > WMT (U.S.: NYSE) WMT (http://quotes.wsj.com/WMT)
- Nike Inc. > NKE (U.S.: NYSE) NKE (http://quotes.wsj.com/NKE)

By using the above methods to get Value at Risk.

In [126]:
```
# List of NYSE_stocks for analytics
NYSE_list = ['JNJ','NKE','WMT']

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

#For-loop for grabing google finance data and setting as a datafram
e
# Set DataFrame as the Stock Ticker

for stock in NYSE_list:
    globals()[stock] = DataReader(stock,'iex',start,end)
```

Let's go ahead and play aorund with the JNJ(Johnson & Johnson) Stock DataFrame to get a feel for the data.

In [127]:
```
JNJ.head()
```

Out[127]:

date	open	high	low	close	volume
2018-11-19	146.43	148.44	146.37	147.73	8714603
2018-11-20	147.50	148.75	146.35	146.45	8937990
2018-11-21	146.23	146.23	140.78	141.99	10275810
2018-11-23	141.85	142.73	141.60	142.23	3404882
2018-11-26	142.00	142.05	140.72	141.37	7590941

In [128]:
```
JNJ.describe()
```

Out[128]:

	open	high	low	close	volume
count	250.000000	250.000000	250.000000	250.000000	2.500000e+02
mean	134.741640	135.681480	133.619960	134.652440	7.842597e+06
std	5.259334	5.293544	5.364461	5.314838	4.984280e+06
min	123.420000	126.760000	121.000000	122.840000	3.404882e+06
25%	130.382500	131.320000	129.212500	130.267500	5.511670e+06
50%	133.305000	134.540000	132.120000	133.290000	6.768298e+06
75%	138.925000	139.635000	137.677500	138.752500	8.428916e+06
max	147.500000	148.990000	147.000000	147.840000	5.814019e+07

In [129]:
```
JNJ.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 250 entries, 2018-11-19 to 2019-11-15
Data columns (total 5 columns):
open      250 non-null float64
high      250 non-null float64
low       250 non-null float64
close     250 non-null float64
volume    250 non-null int64
dtypes: float64(4), int64(1)
memory usage: 11.7+ KB
```

Now that we've seen the DataFrame, let's go ahead and plot out the closing prices of NYSE stocks.

In [130]:
```
# Let's see a historical view of the closing price for JNJ(Johnson & Johnson)
JNJ['close'].plot(title='Closing Price - JNJ',legend=True, figsize=(10,4))
```

Out[130]: <matplotlib.axes._subplots.AxesSubplot at 0x1a232a2198>

In [131]:
```
# Let's see a historical view of the closing price for NKE(Nike Inc.)
NKE['close'].plot(title='Closing Price - NKE',legend=True, figsize=(10,4))
```

Out[131]: <matplotlib.axes._subplots.AxesSubplot at 0x1a233c1f28>

In [132]:
```
# Let's see a historical view of the closing price for WMT(Wal-Mart Stores Inc.)
WMT['close'].plot(title='Closing Price - WMT',legend=True, figsize=(10,4))
```

Out[132]: <matplotlib.axes._subplots.AxesSubplot at 0x1a234ebe48>

Value at risk using the "Bootstrap" method

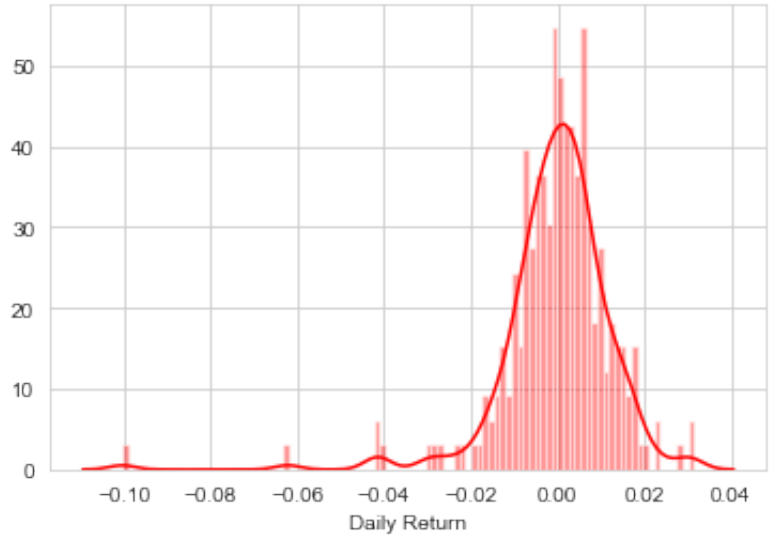
we will calculate the empirical quantiles from a histogram of daily returns.

Let's go ahead and use pandas to retrieve the daily returns for the JNJ, WMT & NKE stock.

In [133]:
```
# We'll use pct_change to find the percent change for each day

#For JNJ stocks
JNJ['Daily Return'] = JNJ['close'].pct_change()
```

In [134]:
```
# Note the use of dropna() here, otherwise the NaN values can't be
read by seaborn
sns.distplot(JNJ['Daily Return'].dropna(),bins=100,color='R')
```

Out[134]: <matplotlib.axes._subplots.AxesSubplot at 0x1a235f8048>



In [135]:
```
(JNJ['Daily Return'].dropna()).quantile(0.05)
```

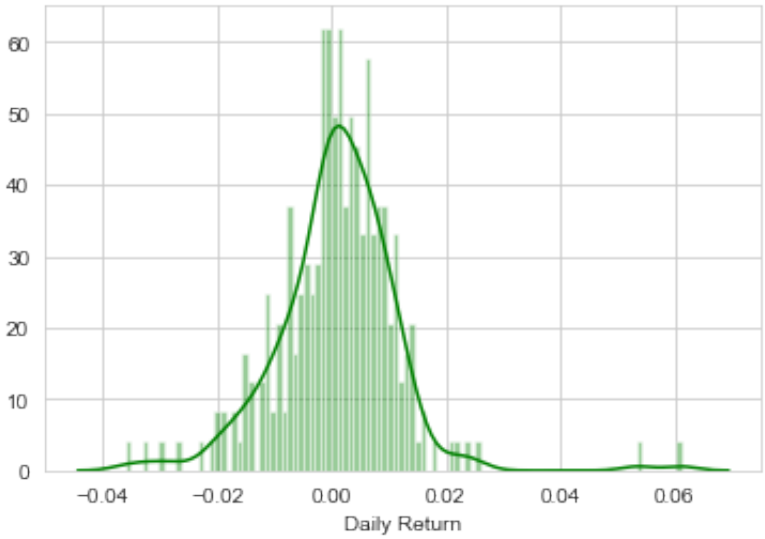
Out[135]: -0.016504008549332205

The 0.05 empirical quantile of JNJ stock daily returns is at -0.016. That means that with 95% confidence, our worst daily loss will not exceed 1.6%. If we have a 1 million dollar investment, our one-day 5% VaR is 0.016 * 1,000,000 = $16,000.

In [136]:
```
# For WMT stocks
WMT['Daily Return'] = WMT['close'].pct_change()
```

In [137]:
```
sns.distplot(WMT['Daily Return'].dropna(),bins=100,color='G')
```

Out[137]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a237e4710>
```



In [138]:
```
(WMT['Daily Return'].dropna()).quantile(0.05)
```

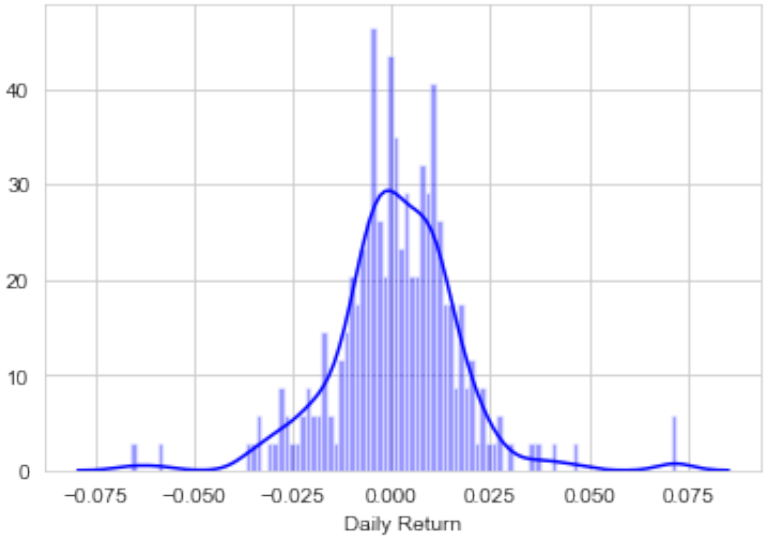
Out[138]:
```
-0.015689282197053324
```

The 0.05 empirical quantile of WMT stock daily returns is at -0.015. That means that with 95% confidence, our worst daily loss will not exceed 1.5%. If we have a 1 million dollar investment, our one-day 5% VaR is 0.015 * 1,000,000 = $15,000.

In [139]:
```
# For NKE stocks
NKE['Daily Return'] = NKE['close'].pct_change()
```

In [140]:
```
sns.distplot(NKE['Daily Return'].dropna(),bins=100,color='B')
```

Out[140]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2393e860>
```



In [141]:
```
(NKE['Daily Return'].dropna()).quantile(0.05)
```

Out[141]:
```
-0.025987996602671037
```

The 0.05 empirical quantile of NKE stock daily returns is at -0.025. That means that with 95% confidence, our worst daily loss will not exceed 2.5%. If we have a 1 million dollar investment, our one-day 5% VaR is 0.025 * 1,000,000 = $25,000.

In [ ]:
```

```