VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CSE3020 – DATA VISUALIZATION

STOCK MARKET ANALYSIS

UNDER THE GUIDANCE OF:

Prof. MEENAKSHI S P

SUBMITTED BY:

VAASU GUPTA	**16BCB0062**
ADWAY UJJWAL	**16BCE0699**

INTRODUCTION

Advanced mathematics and statistics has been present in finance for some time. Prior to the 1980s, banking and finance were well known for being "boring"; investment banking was distinct from commercial banking and the primary role of the industry was handling "simple" (at least in comparison to today) financial instruments, such as loans. Deregulation under the Reagan administration, coupled with an influx of mathematical talent, transformed the industry from the "boring" business of banking to what it is today, and since then, finance has joined the other sciences as a motivation for mathematical research and advancement. For example one of the biggest recent achievements of mathematics was the derivation of the Black-Scholes formula, which facilitated the pricing of stock options (a contract giving the holder the right to purchase or sell a stock at a particular price to the issuer of the option). That said, bad statistical models, including the Black-Scholes formula, hold part of the blame for the 2008 financial crisis.

In recent years, computer science has joined advanced mathematics in revolutionizing finance and **trading**, the practice of buying and selling of financial assets for the purpose of making a profit. In recent years, trading has become dominated by computers; algorithms are responsible for making rapid split-second trading decisions faster than humans could make (so rapidly, the speed at which light travels is a limitation when designing systems). Additionally, machine learning and data mining techniques are growing in popularity in the financial sector, and likely will continue to do so. In fact, a large part of algorithmic trading is **high-frequency trading (HFT)**. While algorithms may outperform humans, the technology is still new and playing in a famously turbulent, high-stakes arena. HFT was responsible for phenomena such as the 2010 flash crash and a 2013 flash crash prompted by a hacked Associated Press tweet about an attack on the White House.

This lecture, however, will not be about how to crash the stock market with bad mathematical models or trading algorithms. Instead, I intend to provide you with basic tools for handling and analyzing stock market data with Python. I will also discuss moving averages, how to construct trading strategies using moving averages, how to formulate exit strategies upon entering a position, and how to evaluate a strategy with backtesting.

IMPLEMENTATION

```
import pandas as pd
from pandas import Series as sr, DataFrame as df
import numpy as np
```

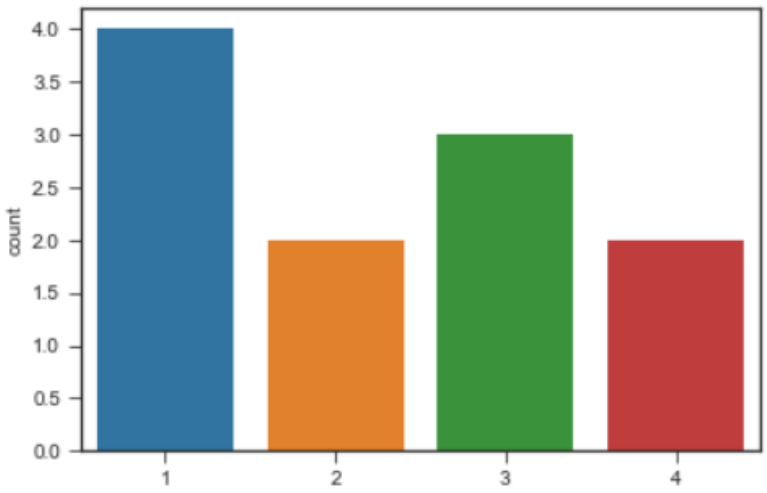
In [3]:

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('ticks')
#multiple grid styles in seaborn - whitegrid,darkgrid, white, dark, ticks
%matplotlib inline
```

In [4]:

```
sns.countplot([1,2,3,4,1,2,3,4,3,1,1],data=None)
#notice the ticks on axis
```

Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6e6fef2898>



In [ ]:

```
##---------PART 1 - STOCK DATA READING FROM YAHOO FINANCES ------------##
```

In [5]:

```
sns.set_style('whitegrid')
#importind datareader of pandas to read from internet the stock data from m
icrosoft,google,apple,amazon
#stock data from google-finances or yahoo-finances
import pandas_datareader.data as web
## DOCS : https://pandas-datareader.readthedocs.io/en/latest/remote_data.ht
ml#
from datetime import datetime
```

In [6]:

```
#start and end date initialized
end = datetime.now()
start = datetime(end.year-1,end.month,end.day)
end
```

Out[6]:

```
datetime.datetime(2017, 9, 10, 22, 1, 36, 3296
```

In [10]:

```
tech_list = ['AAPL','GOOG','MSFT','AMZN'] #apple,google,microsoft,amazon
#create datasets for each stock data
for stock in tech_list:
    globals()[stock] = web.DataReader(stock,'yahoo',start,end) #create glob
al datasets, reading from yahoo finances
```

In [11]:

```
AAPL.head()
```

Out[11]:

	Open	High	Low	Close	Adj Close	Volume
Date						
2016-09-09	104.639999	105.720001	103.129997	103.129997	101.342369	46557000
2016-09-12	102.650002	105.720001	102.529999	105.440002	103.612335	45292800
2016-09-13	107.510002	108.790001	107.239998	107.949997	106.078827	62176200
2016-09-14	108.730003	113.029999	108.599998	111.769997	109.832603	110888700
2016-09-15	113.860001	115.730003	113.489998	115.570000	113.566742	89983600

In [12]:

```
AAPL.describe()
```

Out[12]:

	Open	High	Low	Close	Adj Close	Volume
count	252.000000	252.000000	252.000000	252.000000	252.000000	2.520000e+02
mean	134.201230	135.127500	133.342143	134.324762	133.215407	2.944645e+07
std	17.659769	17.642084	17.476427	17.527964	18.015891	1.376634e+07
min	102.650002	105.720001	102.529999	103.129997	101.342369	1.147590e+07
25%	116.242498	116.832501	115.647501	116.297503	114.690593	2.126205e+07
50%	139.334999	140.064995	138.805000	139.650002	138.531937	2.612170e+07
75%	148.182502	149.530003	146.954998	148.965004	148.210480	3.275498e+07
max	164.800003	164.940002	163.630005	164.050003	164.050003	1.119850e+08

n [13]:

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

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6e6d2a7d30>
```

In [14]:

```
#total volume of stock being traded each day over the past 5 years
AAPL['Volume'].plot(legend = True ,figsize = (10,4))
```

Out[14]:

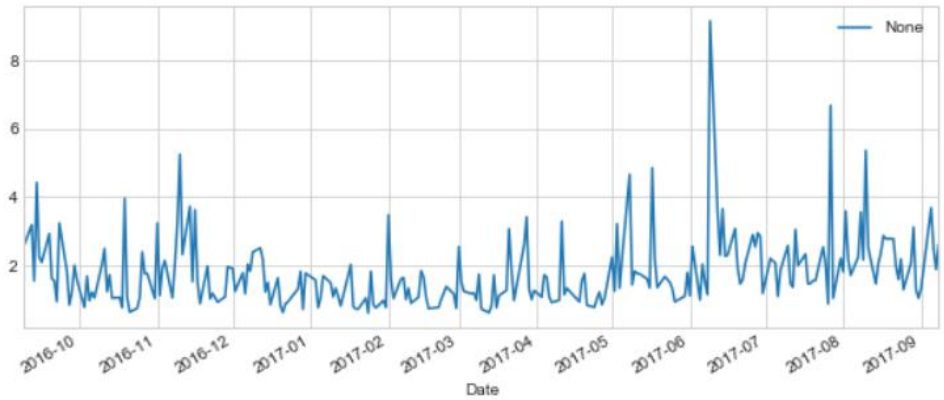
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6e6d1e1eb8>
```

In [15]:

```
#variatin (High-Low of each day)
(AAPL['High']-AAPL['Low']).plot(legend=True,figsize = (10,4))
```

Out[15]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6e6d175be0>
```



In [16]:

```
AAPL[AAPL.High-AAPL.Low > 4]
```

```
#note that peaks > 4 in the graph above and data below match
```

Out[16]:

	Open	High	Low	Close	Adj Close	Volume
Date						
2016-09-14	108.730003	113.029999	108.599998	111.769997	109.832603	110888700
2016-11-10	111.089996	111.089996	105.830002	107.790001	106.465424	57134500
2017-05-08	149.029999	153.699997	149.029999	153.009995	151.784973	48752400
2017-05-17	153.600006	154.570007	149.710007	150.250000	149.662277	50767700
2017-06-09	155.190002	155.190002	146.020004	148.979996	148.397247	64882700
2017-07-27	153.750000	153.990005	147.300003	150.559998	149.971069	32476300
2017-08-10	159.899994	160.000000	154.630005	155.320007	155.320007	40804300

In [17]:

```
(AAPL['High']-AAPL.Low).max() #155.190002-146.020004 on 2017-06-09 denotes
the global maxima of the graph above
```

Out[17]:

```
9.1699979999999925
```

In [ ]:

```
##------------- PART 2 - SIMPLE MOVING AVERAGE METHOD FOR PREDICTION ------
-------###
```

In [18]:

```
##Moving Average
```

In [19]:

```
ma_day = [10,20,50] #moving average for 10,20 and 50 days
for ma in ma_day :
    col_name = "Mov. Avg. for %s days" %str(ma)
    AAPL[col_name] = AAPL['Adj Close'].rolling(window = ma).mean() #Create
3 columns for each and computes moving average
```

In [20]:

```
AAPL.head(50)
```

Out[20]:

	Open	High	Low	Close	Adj Close	Volume	Mov. Avg. for 10 days	Mov. Avg. for 20 days	Mov. Avg. for 50 days
Date									
2016-09-09	104.639999	105.720001	103.129997	103.129997	101.342369	46557000	NaN	NaN	NaN
2016-09-12	102.650002	105.720001	102.529999	105.440002	103.612335	45292800	NaN	NaN	NaN
2016-09-13	107.510002	108.790001	107.239998	107.949997	106.078827	62176200	NaN	NaN	NaN
2016-09-14	108.730003	113.029999	108.599998	111.769997	109.832603	110888700	NaN	NaN	NaN
2016-09-15	113.860001	115.730003	113.489998	115.570000	113.566742	89983600	NaN	NaN	NaN
2016-09-16	115.120003	116.129997	114.040001	114.919998	112.928009	79886900	NaN	NaN	NaN
2016-09-19	115.190002	116.180000	113.250000	113.580002	111.611237	47023000	NaN	NaN	NaN
2016-09-20	113.050003	114.120003	112.510002	113.570000	111.601410	34514300	NaN	NaN	NaN
2016-09-21	113.849998	113.989998	112.440002	113.550003	111.581757	36003200	NaN	NaN	NaN
2016-09-22	114.349998	114.940002	114.000000	114.620003	112.633217	31074000	109.478851	NaN	NaN
2016-09-23	114.419998	114.790001	111.550003	112.709999	110.756317	52481200	110.420245	NaN	NaN
2016-09-26	111.639999	113.389999	111.550003	112.879997	110.923363	29869400	111.151348	NaN	NaN
2016-09-27	113.000000	113.180000	112.339996	113.089996	111.129723	24607400	111.656438	NaN	NaN
2016-	113.690002	114.639999	113.430000	113.949997	111.974823	29641100	111.870660	NaN	NaN

2016-10-10	115.019997	116.750000	114.720001	116.050003	114.038429	36236000	111.516901	111.334125	NaN
2016-10-11	117.699997	118.690002	116.199997	116.300003	114.284088	64041000	111.832338	111.744388	NaN
2016-10-12	117.349998	117.980003	116.750000	117.339996	115.306061	37586800	112.165462	112.018061	NaN
2016-10-13	116.790001	117.440002	115.720001	116.980003	114.952301	35192400	112.637141	112.087339	NaN
2016-10-14	117.879997	118.169998	117.129997	117.629997	115.591034	35652200	113.087202	112.220490	NaN
2016-10-17	117.330002	117.839996	116.779999	117.550003	115.512428	23624900	113.581484	112.415549	NaN
2016-10-18	118.180000	118.209999	117.449997	117.470001	115.433815	24553500	114.020737	112.607170	NaN
2016-10-19	117.250000	117.760002	113.800003	117.120003	115.089874	20034600	114.420681	112.782576	NaN
2016-10-20	116.860001	117.379997	116.330002	117.059998	115.030914	24125800	114.732186	112.902460	NaN
2016-10-21	116.809998	116.910004	116.279999	116.599998	114.578880	23192700	114.981782	113.093589	NaN
2016-10-24	117.099998	117.739998	117.000000	117.650002	115.610680	23538700	115.139007	113.327954	NaN
2016-10-25	117.949997	118.360001	117.309998	118.250000	116.200294	48129000	115.330628	113.581483	NaN
2016-10-26	114.309998	115.699997	113.309998	115.589996	113.586395	66134200	115.158661	113.662062	NaN
2016-10-27	115.389999	115.860001	114.099998	114.480003	112.495644	34562000	114.912996	113.775069	NaN
2016-10-28	113.870003	115.209999	113.449997	113.720001	111.748817	37861700	114.528774	113.807988	NaN

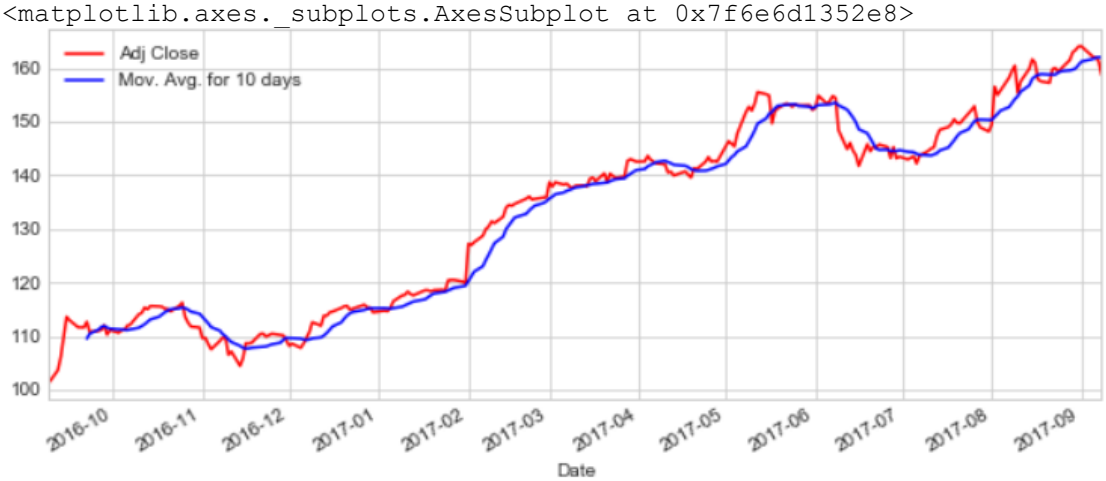
2016-11-01	113.459999	113.769997	110.529999	111.489998	109.557465	43825800	113.547089	113.783913	NaN
2016-11-02	111.400002	112.349998	111.230003	111.589996	109.655731	28331700	113.003675	113.712178	NaN
2016-11-03	110.980003	111.459999	109.550003	109.830002	108.480354	26932600	112.348619	113.540402	NaN
2016-11-04	108.529999	110.250000	108.110001	108.839996	107.502518	30837000	111.640983	113.311383	NaN
2016-11-07	110.080002	110.510002	109.459999	110.410004	109.053238	32560000	110.985239	113.062123	NaN
2016-11-08	110.309998	111.720001	109.699997	111.059998	109.695236	24054500	110.334733	112.832680	NaN
2016-11-09	109.879997	111.320000	108.050003	110.879997	109.517456	59176400	109.927839	112.543250	NaN
2016-11-10	111.089996	111.089996	105.830002	107.790001	106.465424	57134500	109.324817	112.118906	NaN
2016-11-11	107.120003	108.870003	106.550003	108.430000	107.097557	34094100	108.859691	111.694232	NaN
2016-11-14	107.709999	107.809998	104.080002	105.709999	104.410980	51175500	108.143596	111.139160	NaN
2016-11-15	106.570000	107.680000	106.160004	107.110001	105.793777	32264500	107.767227	110.657158	NaN
2016-11-16	106.699997	110.230003	106.599998	109.989998	108.638382	58840500	107.665492	110.334584	NaN
2016-11-17	109.809998	110.349998	108.830002	109.949997	108.598877	27632000	107.677345	110.012982	111.061947

In [21]:

```
AAPL[['Adj Close','Mov. Avg. for 10 days']].plot(legend = True, figsize = (10,4), color = ['r','b'])
```

Out[21]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6e6d1352e8>
```



In [22]:

```
AAPL[['Adj Close','Mov. Avg. for 20 days']].plot(legend = True, figsize = (10,4), color = ['r','b'])
```

Out[22]:

```
<matplotlib.axes. subplots.AxesSubplot at 0x7f6e6d0c70f0>
```

In [23]:

```
AAPL[['Adj Close','Mov. Avg. for 50 days']].plot(legend = True, figsize = (10,4), color = ['r','b'])
```

Out[23]:

```
<matplotlib.axes. subplots.AxesSubplot at 0x7f6e6d2806d8>
```

In [24]:

```
##------------ PART  3 - DAILY RETURN ANALYSIS -------------##
```

In [25]:

```
AAPL['Adj Close'].pct_change().head() #using percentage change function pct_change
```

Out[25]:

```
Date
2016-09-09         NaN
2016-09-12    0.022399
2016-09-13    0.023805
2016-09-14    0.035387
2016-09-15    0.033998
Name: Adj Close, dtype: float64
```

In [26]:

```
#Calculating percentage change myself
pct_change = sr([np.NaN]*len(AAPL), index = AAPL.index)
for i in range(1,len(AAPL)):
    pct_change[i] = (AAPL['Adj Close'][i] - AAPL['Adj Close'][i-1]) / AAPL[
'Adj Close'][i-1]
print(pct_change.head())
```

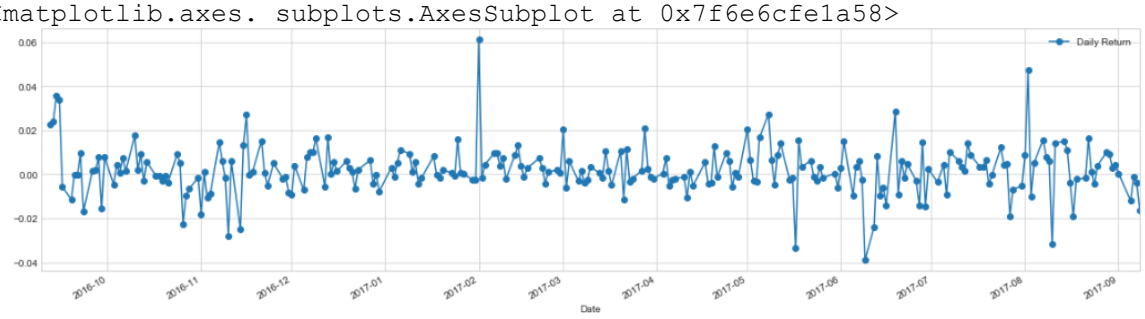
```
Date
2016-09-09         NaN
2016-09-12    0.022399
2016-09-13    0.023805
2016-09-14    0.035387
2016-09-15    0.033998
dtype: float64
```

In [27]:

```
#plotting Daily return
AAPL['Daily Return'] = pct_change
AAPL['Daily Return'].plot(legend = True , figsize = (20,5) , marker ='o')
```

Out[27]:

```
<matplotlib.axes. subplots.AxesSubplot at 0x7f6e6cfe1a58>
```

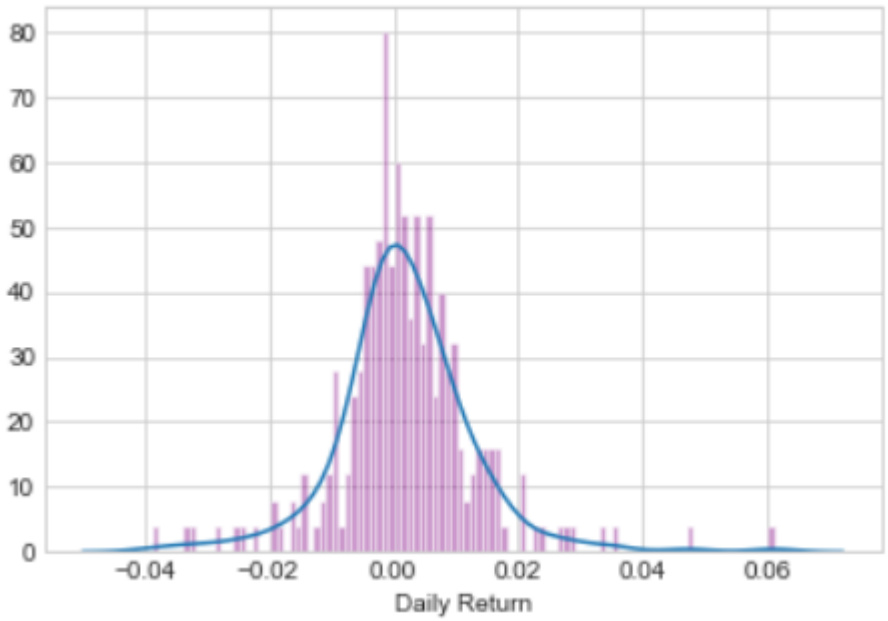


In [28]:

```
sns.distplot(AAPL['Daily Return'].dropna(), bins = 100 , kde = True,hist_kw
s = {'edgecolor' : 'w', 'color' : 'purple'})
```

Out[28]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6e6cf1cdd8>
```



In [29]:

```
#Computing daily return for all four stocks
adj_close_df = df({'AAPL':AAPL['Adj Close'],'GOOG':GOOG['Adj Close'],
                   'MSFT':MSFT['Adj Close'],'AMZN':AMZN['Adj Close']})
adj_close_df.head()
```

Out[29]:

	AAPL	AMZN	GOOG	MSFT
Date				
2016-09-09	101.342369	760.140015	759.659973	54.887646
2016-09-12	103.612335	771.489990	769.020020	55.707886
2016-09-13	106.078827	761.010010	759.690002	55.200119
2016-09-14	109.832603	761.090027	762.489990	54.936470
2016-09-15	113.566742	769.690002	771.760010	55.844593

In [30]:

```
pt_change = adj_close_df.pct_change()
pt_change.head()
```

Out[30]:

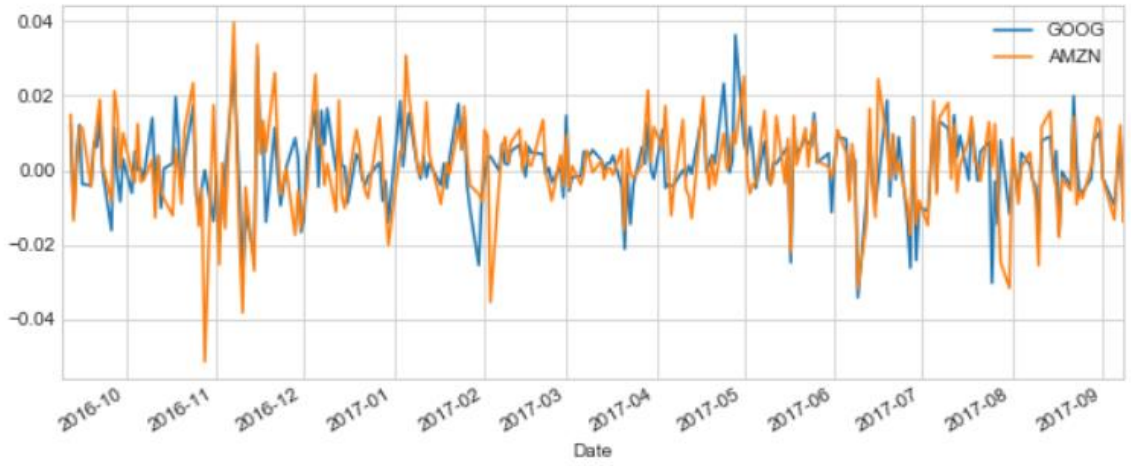
	AAPL	AMZN	GOOG	MSFT
Date				
2016-09-09	NaN	NaN	NaN	NaN
2016-09-12	0.022399	0.014931	0.012321	0.014944
2016-09-13	0.023805	-0.013584	-0.012132	-0.009115
2016-09-14	0.035387	0.000105	0.003686	-0.004776
2016-09-15	0.033998	0.011300	0.012158	0.016530

In [31]:

```
pt_change[['GOOG','AMZN']].plot(legend = True, figsize = (10,4))
```

Out[31]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6e6cf364a8>
```



In [32]:

```
sns.lmplot(x='GOOG',y = 'AMZN' , data=pt_change, markers = 'o',scatter_kws
= {'edgecolor' : 'w', 'color' : 'seagreen'})
```

Out[32]:

```
<seaborn.axisgrid.FacetGrid at 0x7f6e6c463240>
```

In [33]:

```
sns.jointplot(x='GOOG',y='AMZN', data= pt_change, marker='o',edgecolor = 'w
')
```

Out[33]:

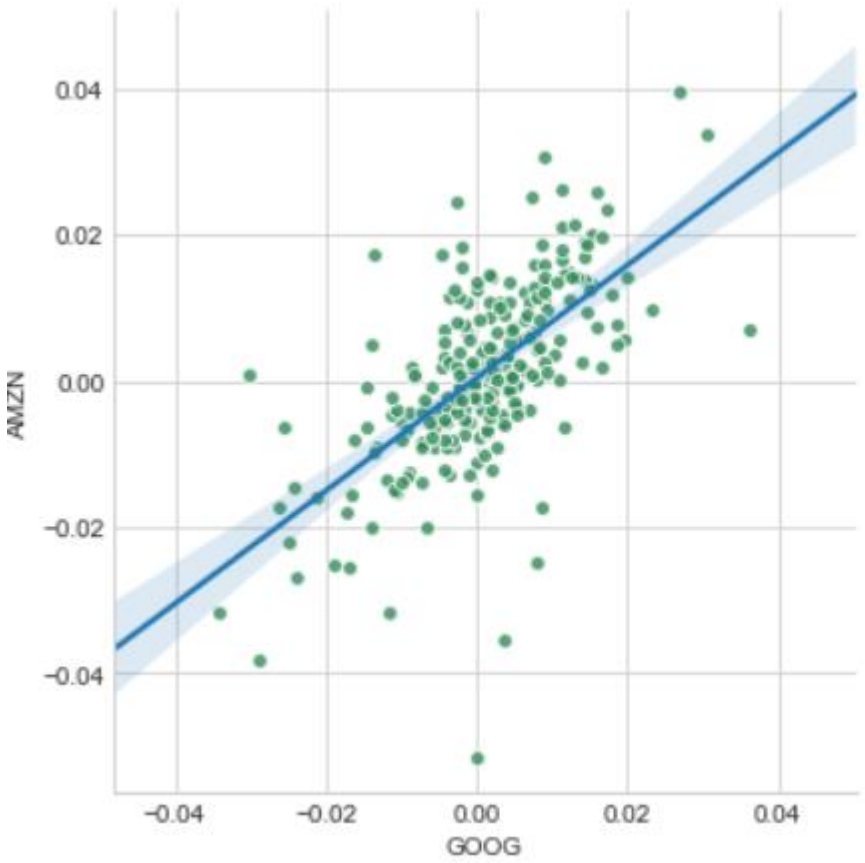
```
<seaborn.axisgrid.JointGrid at 0x7f6e6c418b00>
```

In [33]:

```
sns.jointplot(x='GOOG',y='AMZN', data= pt_change, marker='o',edgecolor = 'w
')
```

Out[33]:

```
<seaborn.axisgrid.JointGrid at 0x7f6e6c418b00>
```

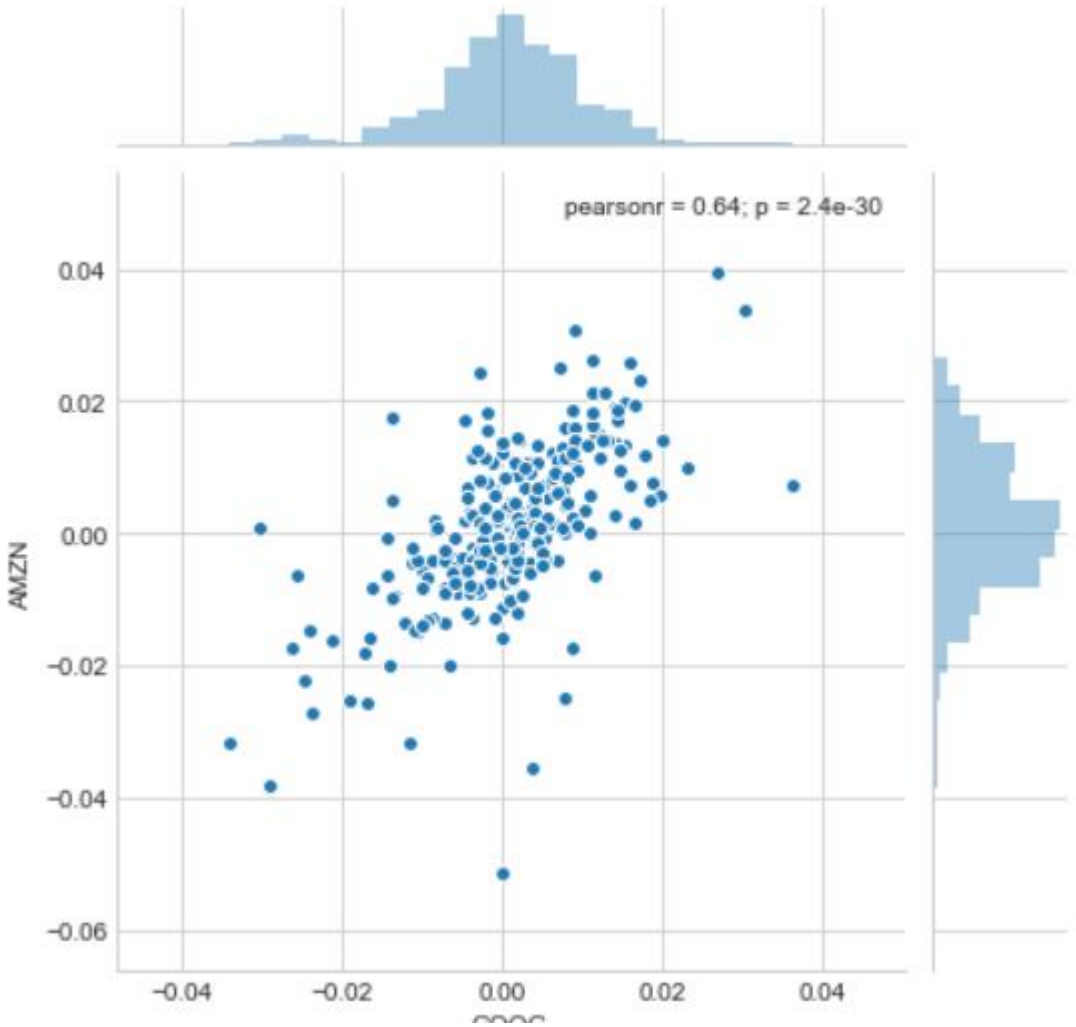


In [33]:

```
sns.jointplot(x='GOOG',y='AMZN', data= pt_change, marker='o',edgecolor = 'w
')
```

Out[33]:

```
<seaborn.axisgrid.JointGrid at 0x7f6e6c418b00>
```



In [34]:

```
##Calculating the Pearon r value (correlation coefficient)
```

In [35]:

```
sum_x = sum_y = sum_x_y = sum_x_sq = sum_y_sq = 0
n = len(pt_change)-1
for i in range(1,n+1):
    sum_x_y = sum_x_y + pt_change.GOOG[i]*pt_change.AMZN[i]
    sum_x = sum_x + pt_change.GOOG[i]
    sum_y = sum_y + pt_change.AMZN[i]
    sum_x_sq = sum_x_sq + pt_change.GOOG[i]**2
    sum_y_sq = sum_y_sq + pt_change.AMZN[i]**2

r = (n * sum_x_y - sum_x*sum_y)/(((n*sum_x_sq-(sum_x)**2)**0.5) * ((n*sum_y
_sq-(sum_y)**2)**0.5))
R = r**2
print("Pearson r : %f" %r)
print("Coefficient of determination : %f" %R)
```

```
Pearson r : 0.640138
Coefficient of determination : 0.409776
```

In [36]:

```
##The above value and the pearson r value in the graph above are same = 0.6
4
##This shows that GOOG and AMZN daily return are POSITIVELY CORRELATED
### that means, the daily returns of Amazon are dependent on Google's by 42
%
```

In [37]:

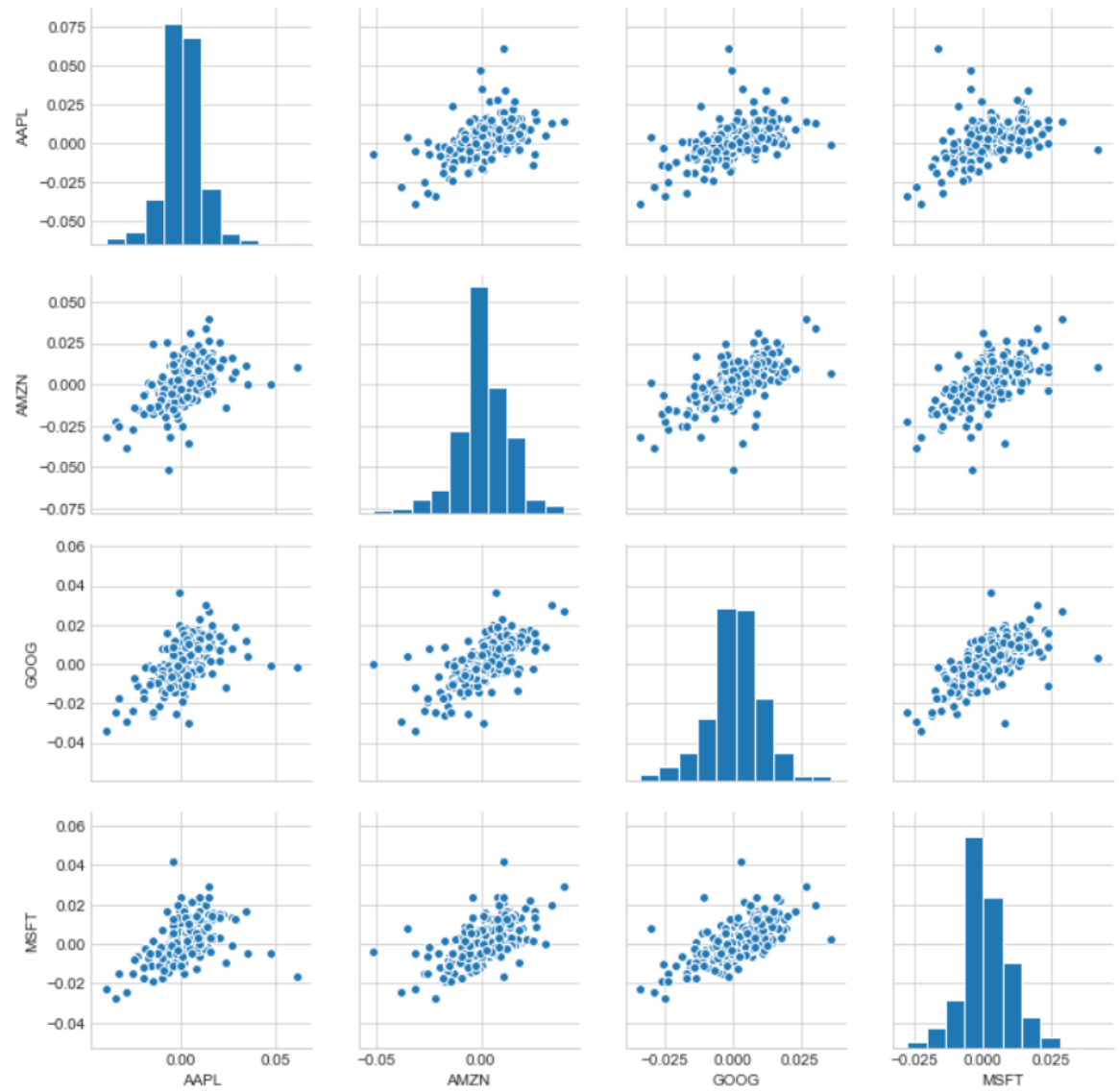
```
sns.pairplot(pt_change.dropna() , aspect = 1 ,diag_kws = {'edgecolor' : 'w'
})

###seaborn.pairplot(data, hue=None, hue_order=None, palette=None, vars=None
, x_vars=None, y_vars=None,
###     kind='scatter/'reg', diag_kind='hist'/'kde', markers=None, size=2.5
, aspect=1, dropna=True,
###     plot_kws=None, diag_kws=None, grid_kws=None)

##it is used for plotting joint plot for every pair of columns in dataset i
n a form of PairGrid
```

Out[37]:

```
<seaborn.axisgrid.PairGrid at 0x7f6e6c276e80>
```

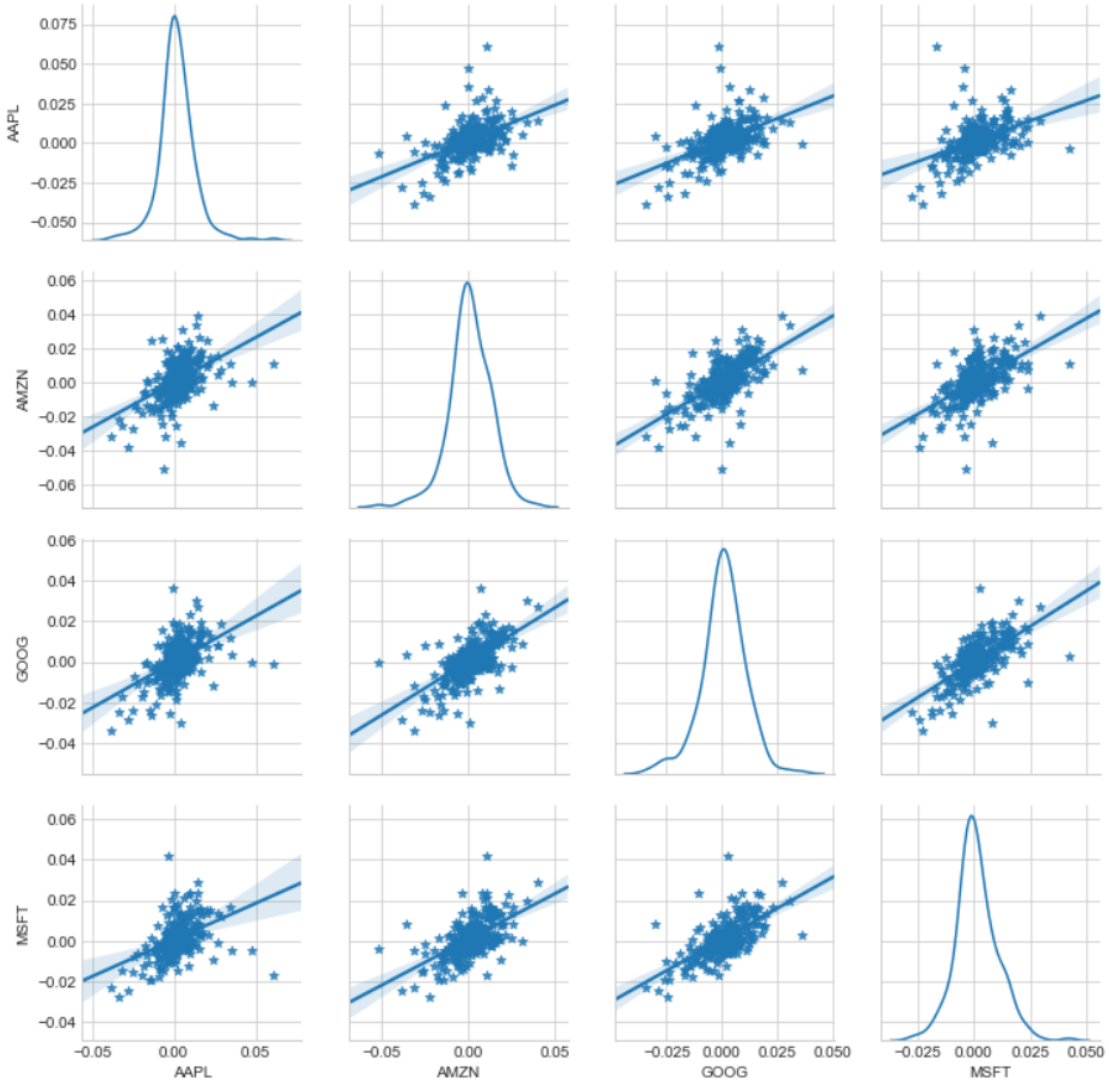


In [38]:

```
sns.pairplot(pt_change.dropna() , aspect = 1 , diag_kind = 'kde', kind = 'reg', markers = '*')
#s for square #d dor diamond
```

Out[38]:

```
<seaborn.axisgrid.PairGrid at 0x7f6e66a593c8>
```

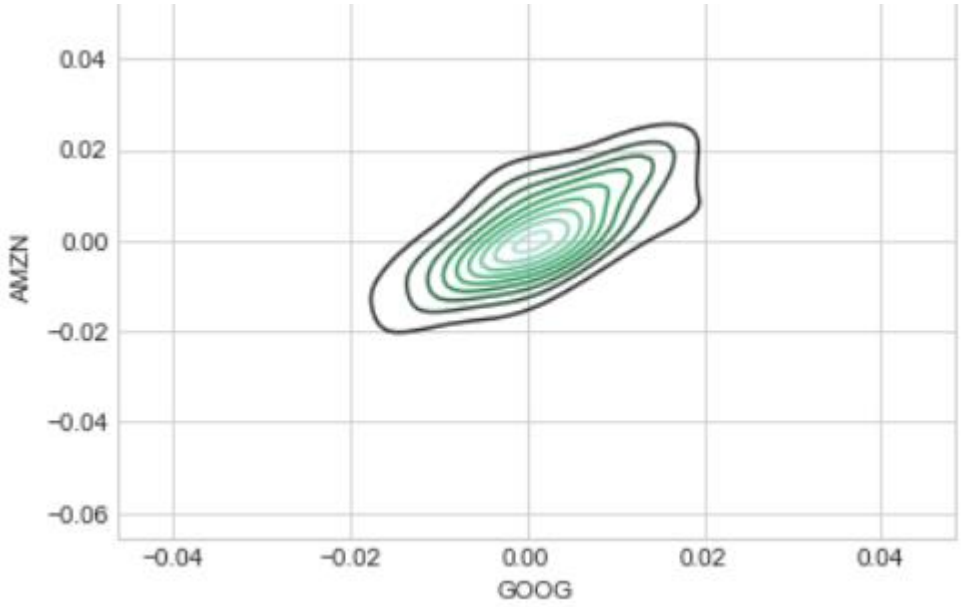


In [39]:

```
sns.kdeplot(pt_change.GOOG.dropna(), pt_change['AMZN'].dropna(), shade = False, legend = True)
#bivariate kernel density estimation
```

Out[39]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6e6657a898>
```



In [40]:

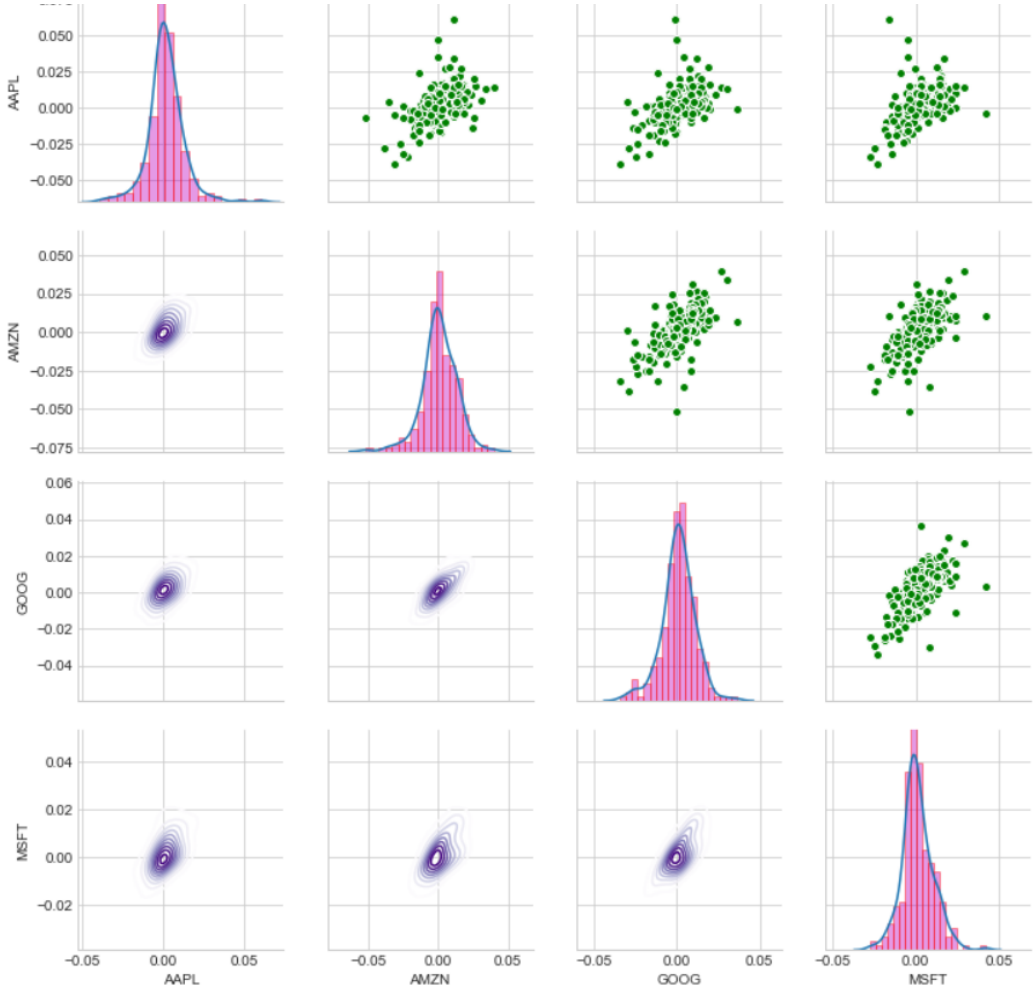
```
### The pairplot can take only two values for diag_kind = 'hist' or 'kde' a
nd two values for kind = 'scatter' or 'reg'
### We can change this by using the PairGrid itself instead of pairplot (wh
ich internally uses PairGrid)
```

In [41]:

```
fig = sns.PairGrid(pt_change.dropna())
#map upper diagnol elements to plt.scatter, diagnol to distplot and lower d
iagnol to kdeplot
fig.map_upper(plt.scatter , edgecolor = 'w', color = 'green')
fig.map_diag(sns.distplot, rug = False , bins = 20, hist_kws = {'edgecolor'
:'r', 'color':'m'})
fig.map_lower(sns.kdeplot, cmap = 'Purples')
```

Out[41]:

```
<seaborn.axisgrid.PairGrid at 0x7f6e662954a8>
```

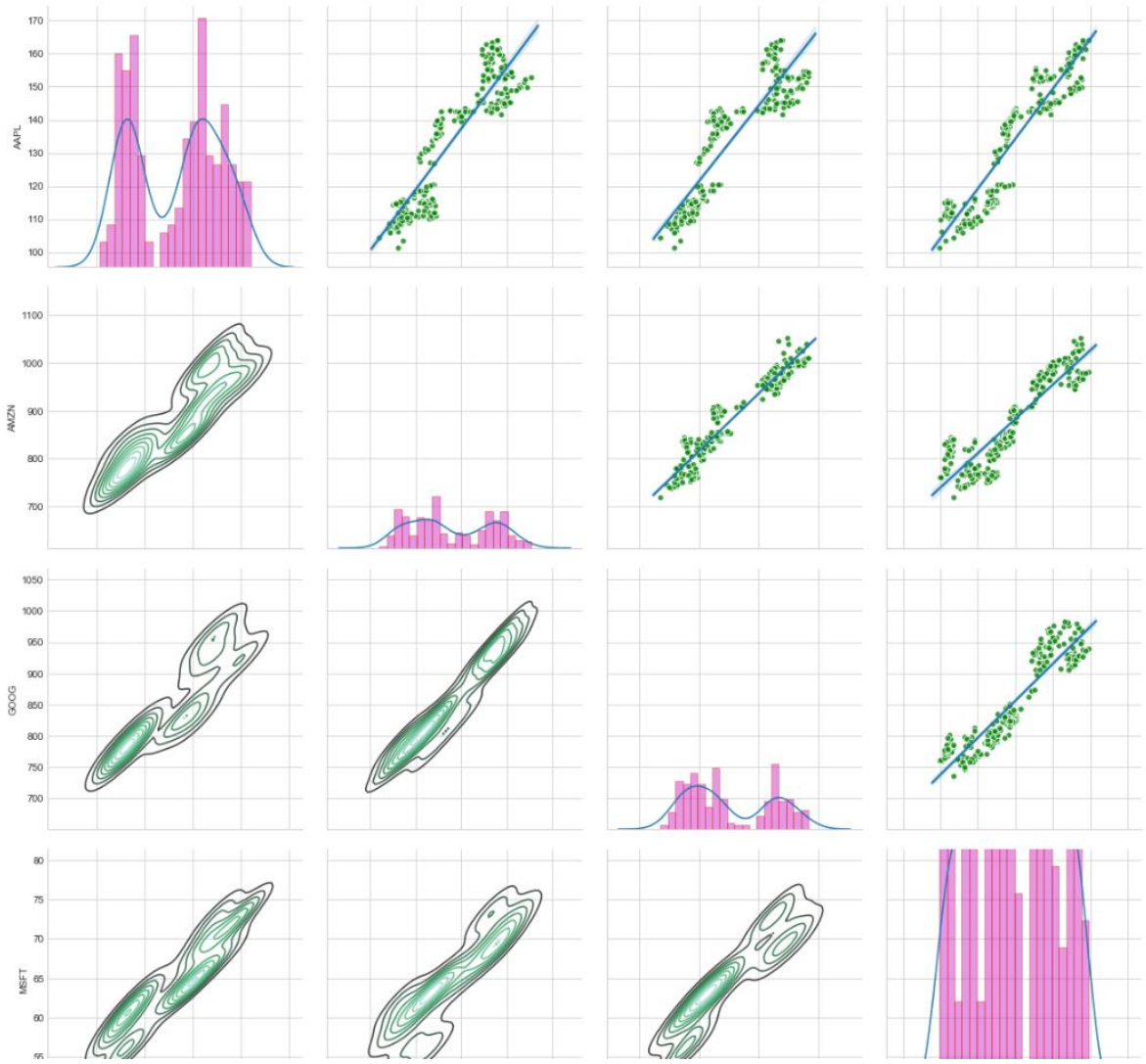


In [42]:

```
### PairGrid for Adjusted Closing prices
fig = sns.PairGrid(adj_close_df, size = 4, aspect =1)
#map upper diagnol elements to plt.scatter, diagnol to distplot and lower d
iagnol to kdeplot
fig.map_upper(sns.regplot , scatter_kws = {'edgecolor' : 'w', 'color':'g'})
fig.map_diag(sns.distplot, rug = False , bins = 20, hist_kws = {'edgecolor'
:'r', 'color':'m'})
fig.map_lower(sns.kdeplot)
```

Out[42]:

```
<seaborn.axisgrid.PairGrid at 0x7f6e6599d6a0>
```



In [123]:

```
sns.__version__
```

Out[123]:

```
'0.8.0'
```

In [43]:

```
## ----------PART 4 - RISK ANALYSIS ---------------####
```

In [46]:

```
pt_change.std()
```

Out[46]:

```
AAPL    0.011185
AMZN    0.012104
GOOG    0.010043
MSFT    0.009347
dtype: float64
```

In [52]:

```
##Code for std -- just for fun!
var = 0
for i in range(1,len(pt_change)):
```

```
    var = var + (pt_change.ix[i] - pt_change.mean())**2
sd = (var / (len(pt_change)-2)) ** 0.5 ##Computing standard deviantion by d
ividing by n-1 and not n as it is a sample
sd
##values below match exactly above
```

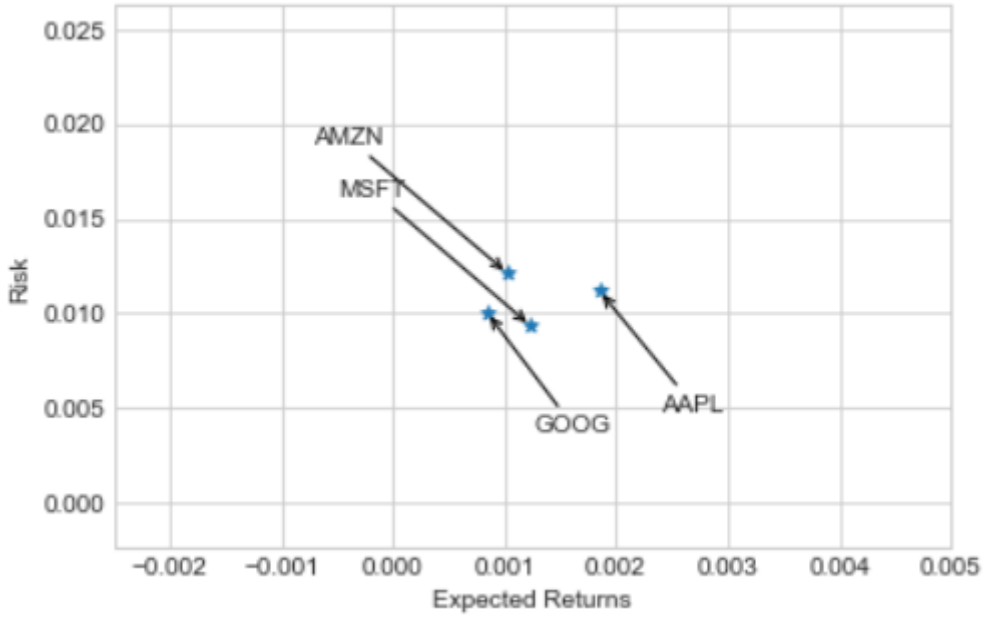
Out[52]:

```
AAPL    0.011185
AMZN    0.012104
GOOG    0.010043
MSFT    0.009347
dtype: float64
```

In [92]:

```
rets = pt_change.dropna()
area = np.pi *10
plt.scatter(rets.mean(),rets.std(), s =area , marker = '*') ## s is the are
a of the scatter point
plt.xlim([-0.0025,0.0050])
#plt.ylim([0.005,0.0020])
plt.xlabel('Expected Returns')
plt.ylabel('Risk')
k=0
##Set annotation with text, arrows, boxes and many more on the plot
### check out http://matplotlib.org/users/annotations_guide.html
for (label, x, y) in zip(rets.columns,rets.mean(),rets.std()): ##zip functi
on zips the data to make it iteratable
    print(label," ",x," ",y)
    plt.annotate(label, xy = (x, y), xytext = ((-1)**k*50,(-1)**(k+1)*50),
        textcoords = 'offset points', ha = 'right', va = 'bottom',
        arrowprops = dict(arrowstyle = '->', connectionstyle = 'arc3'))
    k+=1
```

```
AAPL   0.00184868844651   0.0111852199605
AMZN   0.00102803657834   0.0121039259429
GOOG   0.000841617844089   0.0100427892765
MSFT   0.00123333034988   0.00934700348358
```



In [ ]:

```
###----------------PART 5 - VALUE AT RISK ----------------###
```

In [101]:

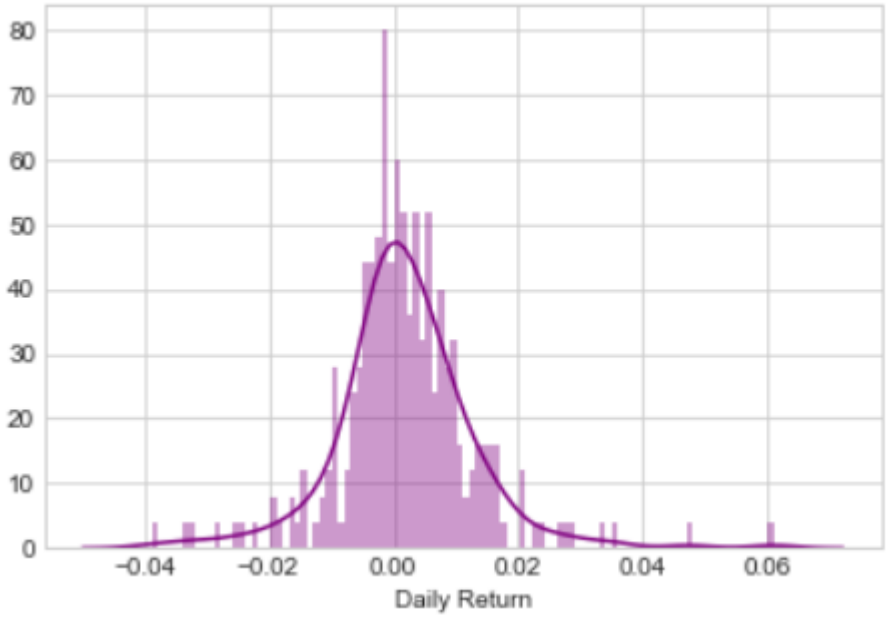
```
## METHOD 1 - BOOTSTRAP METHOD
```

In [109]:

```
sns.distplot(AAPL['Daily Return'].dropna() , color = 'purple', bins = 100)
```

Out[109]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6e64042828>
```



In [112]:

```
AAPL['Daily Return'].quantile(0.05) ## 5 % quantile range
## 5% probability that daily return will be less than -1.5%
```

Out[112]:

```
-0.015138197789679852
```

In [ ]:

```
## METHOD 2 - MONTE CARLO METHOD USING GEOMETRIC BROWNIAN MOTION MODEL
```

In [115]:

```
days = 365
dt = 1/days

mu = rets.mean()['GOOG']
sigma = rets.std()['GOOG']

def monte_carlo(start_price, days , mu, sigma):
    price = np.zeros(days)
    drift = np.zeros(days)
    shock = np.zeros(days)

    price[0] = start_price

    for i in range(1,days):
        drift[i] = mu * dt
        shock[i] = np.random.normal(loc = mu*dt , scale = sigma*np.sqrt(dt)
)
        price[i] = price[i-1] + price[i-1]*(drift[i]+shock[i])

    return price
```

In [116]:

```
GOOG.head()
```

	Open	High	Low	Close	Adj Close	Volume
Date						
2016-09-09	770.099976	773.244995	759.659973	759.659973	759.659973	1885500
2016-09-12	755.130005	770.289978	754.000000	769.020020	769.020020	1311000
2016-09-13	764.479980	766.219971	755.799988	759.690002	759.690002	1395000
2016-09-14	759.609985	767.679993	759.109985	762.489990	762.489990	1087400
2016-09-15	762.890015	773.799988	759.960022	771.760010	771.760010	1305100

In [122]:

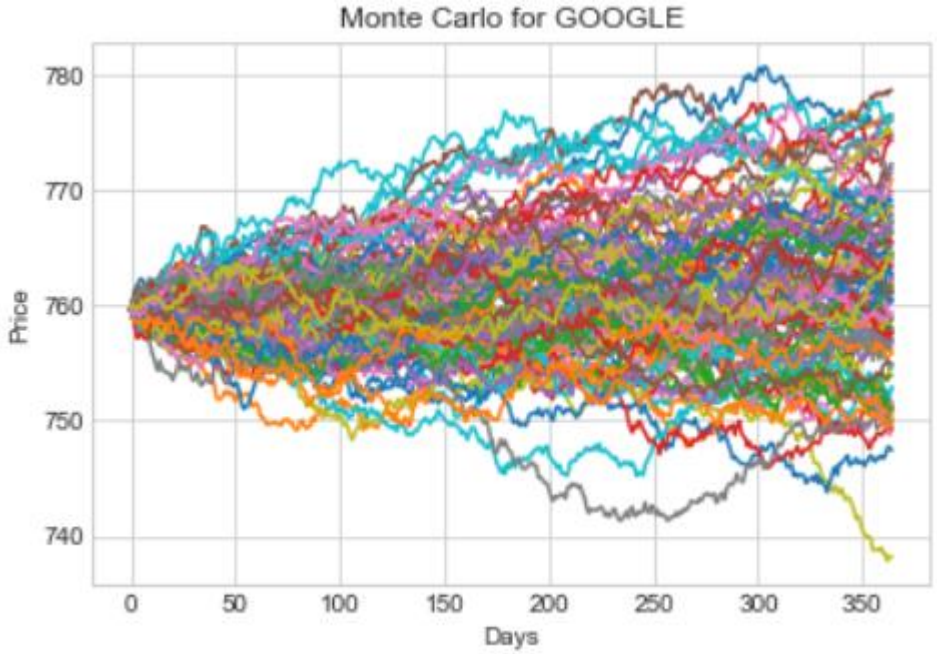
```
start_price  = 759.659973

for trials in range(1,100):
    plt.plot(monte_carlo(start_price,days,mu,sigma))

plt.xlabel('Days')
plt.ylabel('Price')
plt.title('Monte Carlo for GOOGLE')
```

Out[122]:

```
<matplotlib.text.Text at 0x7f6e5fa45358>
```



In [125]:

```
# Set a large numebr of runs
runs = 10000

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

# Set the print options of numpy to only display 0-5 points from an array t
o suppress output
np.set_printoptions(threshold=5)

for run in range(runs):
    # Set the simulation data point as the last stock price for that run
    simulations[run] = monte_carlo(start_price,days,mu,sigma)[days-1];
```

In [131]:

```
# Now we'lll define q as the 1% empirical qunatile, this basically means th
at 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, "Start price: $%.2f" %start_price)
# Mean ending price
plt.figtext(0.6, 0.7, "Mean final price: $%.2f" % simulations.mean())
```

```
# Variance of the price (within 99% confidence interval)
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')

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

CONCLUSION

In the project, we proposed the use of the data collected from different global financial markets with machine learning algorithms in order to predict the stock index movements. SVM algorithm works on the large dataset value which is collected from different global financial markets. Also, SVM does not give a problem of over fitting. Various machine learning based models are proposed for predicting the daily trend of Market stocks. Numerical results suggest the high efficiency. The practical trading models built upon our well-trained predictor. The model generates higher profit compared to the selected benchmarks.

REFERENCES

[1] Zhen Hu, Jibe Zhu, and Ken Tse "Stocks Market Prediction Using Support Vector Machine", 6th International Conference on Information Management, Innovation Management and Industrial Engineering, 2013.M.

[2] Wei Huang, Yoshiteru Nakamori, Shou-Yang Wang, "Forecasting stock market movement direction with support vector machine", Computers & Operations Research, Volume 32, Issue 10, October 2005, Pages 2513–2522.

[3] N. Ancona, Classification Properties of Support Vector Machines for Regression, Technical Report, RIIESI/CNR-Nr. 02/99.

[4] K. jae Kim, "Financial time series forecasting using support vector machines," Neurocomputing, vol. 55, 2003.

[5] Debashish Das and Mohammad shorif uddin data mining and neural network techniques in stock market prediction: a methodological review, international journal of artificial intelligence & applications, vol.4, no.1, January 2013