Source: https://www.kaggle.com/bmconrad/exploratory-analysis-stock-data

Colab link: https://colab.research.google.com/drive/1-fS-iu3PHhvHNO2kZhvrGeS-6sFyAx8T

Another EDA version: https://colab.research.google.com/drive/1gE7gf-b3ORjMMVoqFePMfE8bzlZOGutV

In [0]:

```
!nvidia-smi
```

```
Wed Nov 27 12:07:39 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.33.01    Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   70C    P8    12W /  70W |      0MiB / 15079MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

In [0]:

```
# Importing libraries.
import pandas as pd
import numpy as np
import seaborn as sns #visualisation
import matplotlib.pyplot as plt #visualisation
%matplotlib inline
sns.set(color_codes=True)
from scipy.stats import norm
from subprocess import check_output
```

Load data

In [0]:

```
#install gdown
import os.path as path
if not path.exists('M3final.zip'):
 !pip install gdown
 !gdown https://drive.google.com/uc?id=1w4Sf7GXSzVs5Gcci9Dx2GqiT96A0OeeU
 !unzip M3final.zip
else :
 print('data is in places')
```

```
Requirement already satisfied: gdown in /usr/local/lib/python3.6/dist-packages (3.6.4)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from gdown)
(1.12.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-packages (from gdown
) (4.28.1)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from g
down) (2.21.0)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/python3.6/dist-pac
kages (from requests->gdown) (1.24.3)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-pac
kages (from requests->gdown) (3.0.4)
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.6/dist-packages (
from requests->gdown) (2.8)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packag
es (from requests->gdown) (2019.9.11)
Downloading...
From: https://drive.google.com/uc?id=1w4Sf7GXSzVs5Gcci9Dx2GqiT96A0OeeU
To: /content/M3final.zip
31.8MB [00:00, 119MB/s]
Archive:  M3final.zip
  inflating: fundamentals.csv
  inflating: __MACOSX/._fundamentals.csv
  inflating: prices-split-adjusted.csv
  inflating: __MACOSX/._prices-split-adjusted.csv
  inflating: prices.csv
  inflating: __MACOSX/._prices.csv
  inflating: securities.csv
  inflating: __MACOSX/._securities.csv
```

In [0]:

```
fund = pd.read_csv('fundamentals.csv')
price_split = pd.read_csv('prices-split-adjusted.csv')
prices = pd.read_csv('prices.csv')
comp_info = pd.read_csv('securities.csv')
#details of the data are described in stakeholders report
#in EDA part, we will use the price_split dataframe
```

In [0]:

```
#add new column named "change" which is the result of columns "open" and "close"
price_split['change'] = price_split['close'] - price_split['open']
price_split.head()
```

Out[0]:

	date	symbol	open	close	low	high	volume	change
0	2016-01-05	WLTW	123.430000	125.839996	122.309998	126.250000	2163600.0	2.409996
1	2016-01-06	WLTW	125.239998	119.980003	119.940002	125.540001	2386400.0	-5.259995
2	2016-01-07	WLTW	116.379997	114.949997	114.930000	119.739998	2489500.0	-1.430000
3	2016-01-08	WLTW	115.480003	116.620003	113.500000	117.440002	2006300.0	1.140000
4	2016-01-11	WLTW	117.010002	114.970001	114.089996	117.330002	1408600.0	-2.040001

Exploratory data analysis

First, we inspect the data and understand the dataframe before we do further analysis such as:

- **What does some of our closing data look like?**
- **Who were the latest top 10 closers?**
- **What do their specs look like over time with respect to one another?**
- **What is their riskiness comparatively?**

In [0]:

```
#inspect the datatype of the dataframe
price_split.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 851264 entries, 0 to 851263
Data columns (total 8 columns):
date      851264 non-null object
symbol    851264 non-null object
open      851264 non-null float64
close     851264 non-null float64
low       851264 non-null float64
high      851264 non-null float64
volume    851264 non-null float64
change    851264 non-null float64
```

```
dtypes: float64(6), object(2)
memory usage: 52.0+ MB
```

We notice that ‘Date’ column is displayed as an object and this can be problematic when we plot histograms/line chart so we convert it to a datetime datatype.

In [0]:

```
#turn 'Date' column to datetime
price_split["date"] = pd.to_datetime(price_split["date"])
```

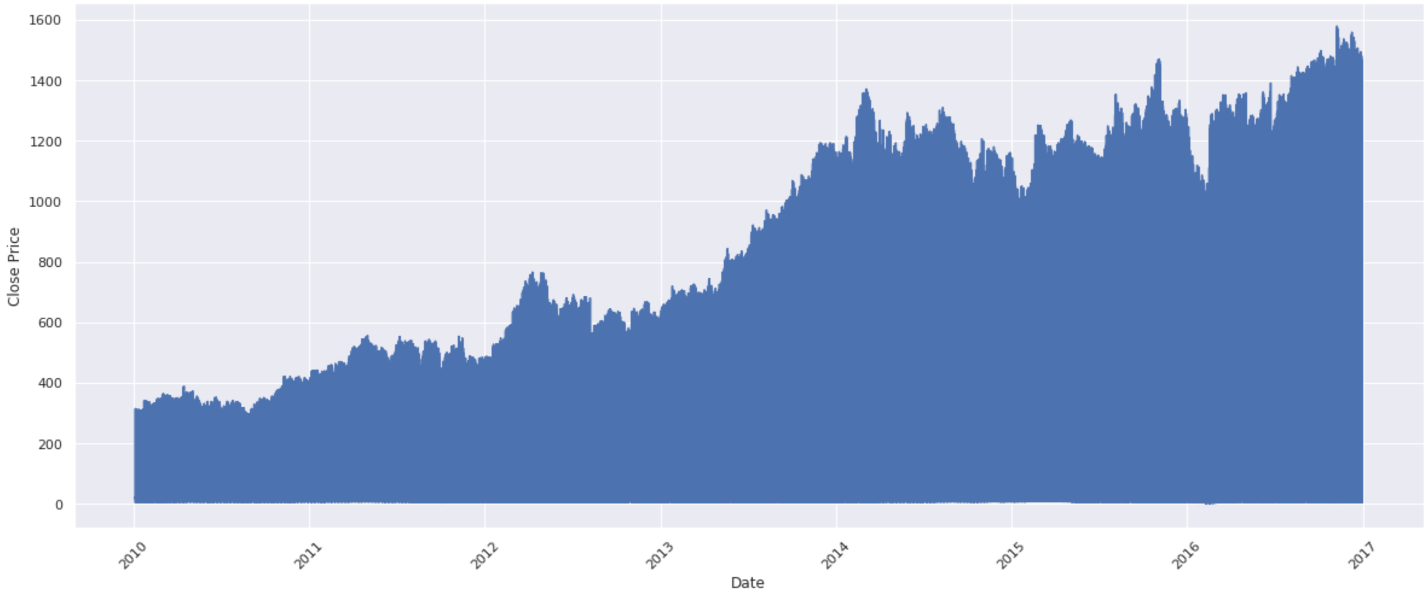
In [0]:

```
#Here we use the date and close columns to have an overview over time
plt.figure(figsize=(20,8))
plt.plot('date','close',data=price_split)
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.xticks(rotation=45)
```

```
/usr/local/lib/python3.6/dist-packages/pandas/plotting/_matplotlib/converter.py:103: FutureWarning: Using an implicitly registered datetime converter for a matplotlib plotting method. The converter was registered by pandas on import. Future versions of pandas will require you to explicitly register matplotlib converters.

To register the converters:
 >>> from pandas.plotting import register_matplotlib_converters
 >>> register_matplotlib_converters()
  warnings.warn(msg, FutureWarning)
```

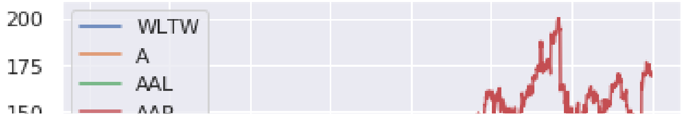
Out[0]:

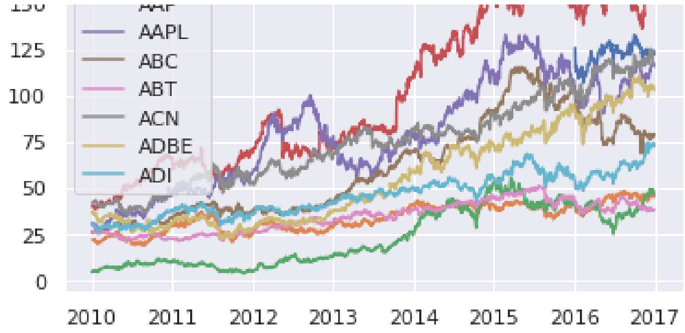
```
(array([733408., 733773., 734138., 734503., 734869., 735234., 735599.,
        735964., 736330.]), <a list of 9 Text xticklabel objects>)
```



In [0]:

```
#Next we look at stocks of the first 10 companies over time
symbols = price_split["symbol"].unique().tolist() #get the unique value from 'symbol' column and convert such series to list
for u in symbols[:10]: #we also use date and close columns
    dates = price_split[(price_split["symbol"] == u)]["date"]
    values = price_split[(price_split["symbol"] == u)]["close"]
    plt.plot(dates.tolist(), values.tolist())
plt.legend(symbols, loc='upper left')
plt.show()
```





Here we can see that...

In [0]:

```
#Mask by most recent year
latest_year = max(pd.unique(list(price_split["date"].apply(lambda x:x.year)))) #find the most recent year which is 2016 only
latest_year_mask = [i==latest_year for i in price_split["date"].apply(lambda x:x.year)]
#create a filtered list of the stocks in 2016
df_masked1 = price_split[latest_year_mask] #apply such filter to price_split
df_masked1.head()
```

Out[0]:

	date	symbol	open	close	low	high	volume	change
0	2016-01-05	WLTW	123.430000	125.839996	122.309998	126.250000	2163600.0	2.409996
1	2016-01-06	WLTW	125.239998	119.980003	119.940002	125.540001	2386400.0	-5.259995
2	2016-01-07	WLTW	116.379997	114.949997	114.930000	119.739998	2489500.0	-1.430000
3	2016-01-08	WLTW	115.480003	116.620003	113.500000	117.440002	2006300.0	1.140000
4	2016-01-11	WLTW	117.010002	114.970001	114.089996	117.330002	1408600.0	-2.040001

In [0]:

```
#Mask by most top recent closers
#remind: symbols = price_split["symbol"].unique().tolist() (the same as above so we will not run again)
maxPerSymbol={} #create a dictionary
for u in symbols:
    values = price_split[(price_split["symbol"] == u)]["close"].tolist() #get the close price of companies and convert them to list
    maxPerSymbol[u] = max(values) #Get the names of companies from the dictionary having the maximum close price
    #print(maxPerSymbol[u], u) #this step is optional to see the result

top10Closers = list(dict(sorted(maxPerSymbol.items(), #get the name of top 10 recent closers
                   key=lambda v:v[1], #sort by key -> v[0] | sort by value->v[1]
                   reverse=True)[:10]).keys()) #sort in a descending order

top_10_mask = [i in top10Closers for i in df_masked1["symbol"].tolist()] #create filtered list of top 10 closes in 2016
df_masked2 = df_masked1[top_10_mask] #apply such filter to df_masked1 which is in 2016
df_masked2.head()
```

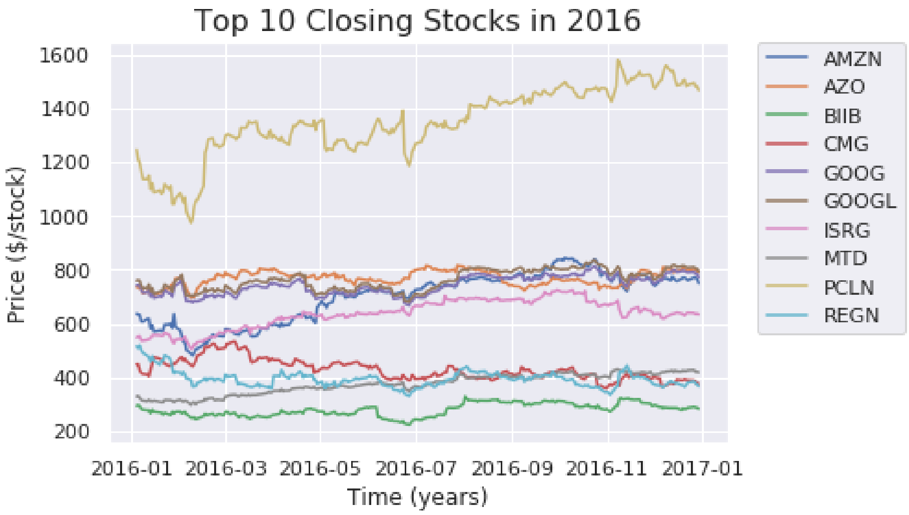
Out[0]:

	date	symbol	open	close	low	high	volume	change
725425	2016-01-04	AMZN	656.289978	636.989990	627.510010	657.719971	9314500.0	-19.299988
725441	2016-01-04	AZO	733.000000	735.479980	728.520020	742.229980	299100.0	2.479980
725452	2016-01-04	BIIB	300.339996	294.619995	291.399994	301.019989	2451100.0	-5.720001
725485	2016-01-04	CMG	468.700012	448.809998	447.500000	469.000000	2690300.0	-19.890014
725589	2016-01-04	GOOG	743.000000	741.840027	731.257996	744.059998	3272800.0	-1.159973

In [0]:

```
#Visualize top 10 most recent closers in 2016
symbols = df_masked2["symbol"].unique().tolist() #get the unique value from 'symbol' col
umn from df_masked2 and convert such series to list
for u in symbols[:10]: #choose the first 10 top closers, we still use date and close colu
mns
    dates = df_masked2[(price_split["symbol"] == u)]["date"]
    values = df_masked2[(price_split["symbol"] == u)]["close"]
    plt.plot(dates.tolist(), values.tolist())
#plt.legend(symbols, loc='upper left')
plt.legend(symbols, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.title(r'Top 10 Closing Stocks in 2016', fontsize=16)

plt.xlabel('Time (years)', fontsize=12)
plt.ylabel('Price ($/stock)', fontsize=12)
plt.show()
```

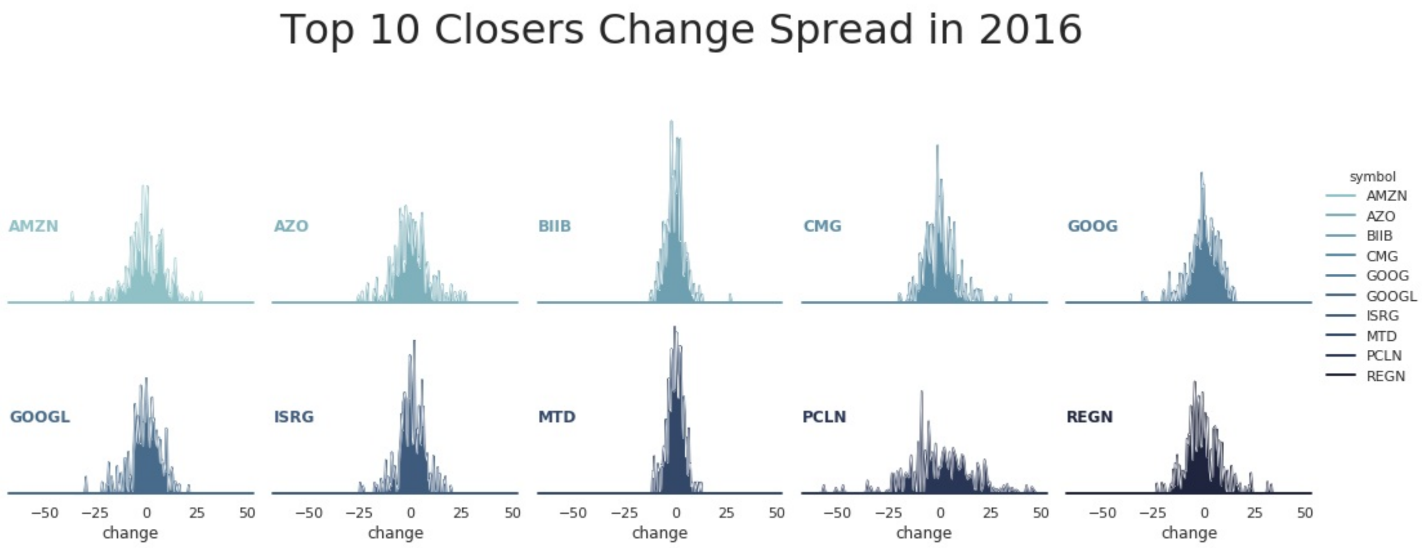
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: UserWarning: Boolean Seri
es key will be reindexed to match DataFrame index.
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: UserWarning: Boolean Seri
es key will be reindexed to match DataFrame index.
  after removing the cwd from sys.path.
```



The Priceline Group has the highest close price in 2016

In [0]:

```
#Visualize top 10 closers change. Now let's have a look at 'change' column
def build_density_facetwrap(somedf, colName, valName, wrapAmount, yourTitle):
    sns.set(style="white", rc={"axes.facecolor": (0, 0, 0, 0)})

    # Initialize the FacetGrid object
    pal = sns.cubehelix_palette(10, rot=-.25, light=.7)
    g = sns.FacetGrid(somedf,col=colName, hue=colName, col_wrap=wrapAmount, palette=pal)
    # Draw the densities in a few steps
    g.map(sns.kdeplot, valName, clip_on=False, shade=True, alpha=1, lw=1.5, bw=.2)
    g.map(sns.kdeplot, valName, clip_on=False, color="w", lw=0.5, bw=.2)
    g.map(plt.axhline, y=0, lw=2, clip_on=False)


    # Define and use a simple function to label the plot in axes coordinates
    def label(x, color, label):
        ax = plt.gca()
        ax.text(0, .4, label, fontweight="bold", color=color,
                ha="left", va="center", transform=ax.transAxes)
        plt.xlabel('Change ($/day)', fontsize=12)
        plt.ylabel('Frequency', fontsize=12)

    g.map(label, "change")
    # Set the subplots to overlap
    g.fig.subplots_adjust(hspace=0)

    # Remove axes details that don't play will with overlap
    g.set_titles("")
    g.fig.suptitle(yourTitle, fontsize=32)
    g.set(yticks=[])
    g.despine(bottom=True, left=True)
    g.fig.subplots_adjust(top=.8)
    g.add_legend()

build_density_facetwrap(df_masked2,
                       "symbol",
                        "change",
                        5,
                        "Top 10 Closers Change Spread in 2016"
                       )
```



In [0]:

```
df_masked2["date"] = pd.to_datetime(df_masked2["date"]) #change the date to datetime
months = list(df_masked2["date"].apply(lambda x:x.month).unique())#get the unique values
of months and turn them to list
df_masked2["month"] = df_masked2["date"].apply(lambda x: x.month) #create a month column
```

```
             date symbol         open  ...     volume     change  month
725425 2016-01-04   AMZN   656.289978  ...  9314500.0 -19.299988      1
725441 2016-01-04    AZO   733.000000  ...   299100.0   2.479980      1
725452 2016-01-04   BIIB   300.339996  ...  2451100.0  -5.720001      1
725485 2016-01-04    CMG   468.700012  ...  2690300.0 -19.890014      1
725589 2016-01-04   GOOG   743.000000  ...  3272800.0  -1.159973      1
...           ...    ...          ...  ...        ...        ...    ...
850964 2016-12-30  GOOGL   803.210022  ...  1728300.0 -10.760010     12
851007 2016-12-30   ISRG   638.320007  ...   267300.0  -4.150024     12
851079 2016-12-30    MTD   421.980011  ...   124200.0  -3.420013     12
851115 2016-12-30   PCLN  1483.489990  ...   405100.0 -17.429931     12
851147 2016-12-30   REGN   375.299988  ...   612900.0  -8.209992     12

[2520 rows x 9 columns]
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports until
```

Selecting any 3 companies for visualizations on respective opening and closing stock prices.

In [0]:

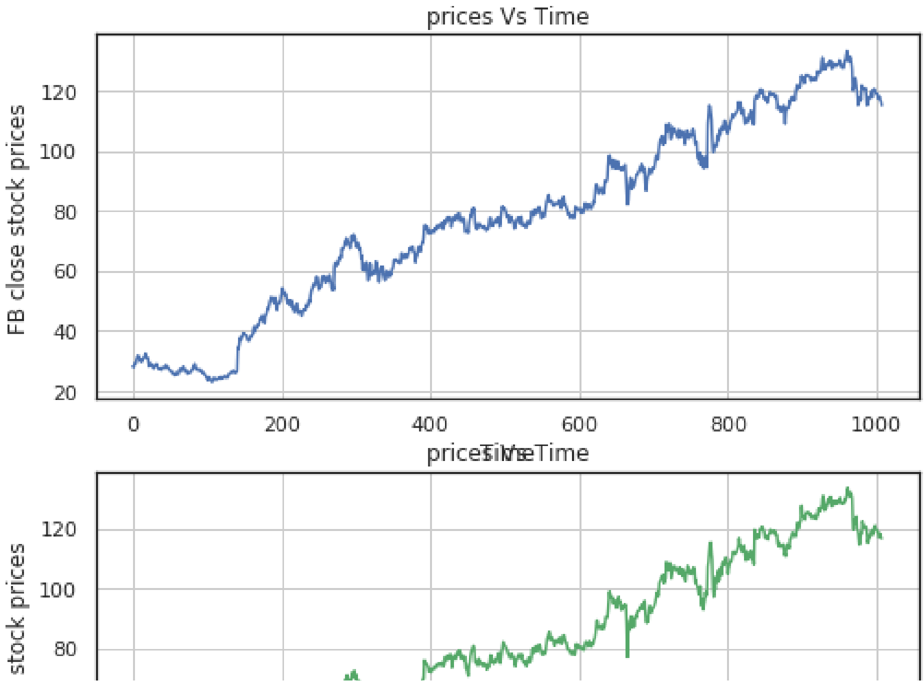
```
comp_plot = comp_info.loc[(comp_info["Security"] == 'Yahoo Inc.') | (comp_info["Security
"] == 'Microsoft Corp.') | (comp_info["Security"] == 'Facebook'), ["Ticker symbol"] ]["T
icker symbol"]
print(comp_plot)
```

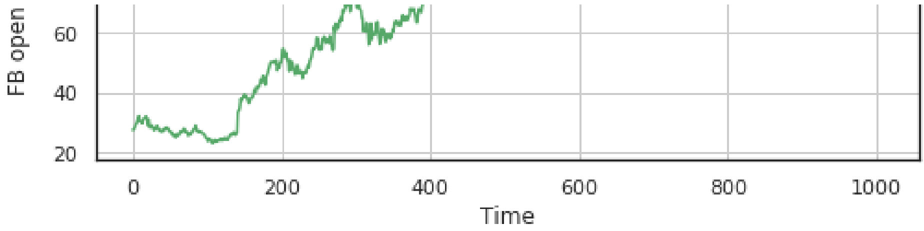
```
181      FB
306    MSFT
Name: Ticker symbol, dtype: object
```

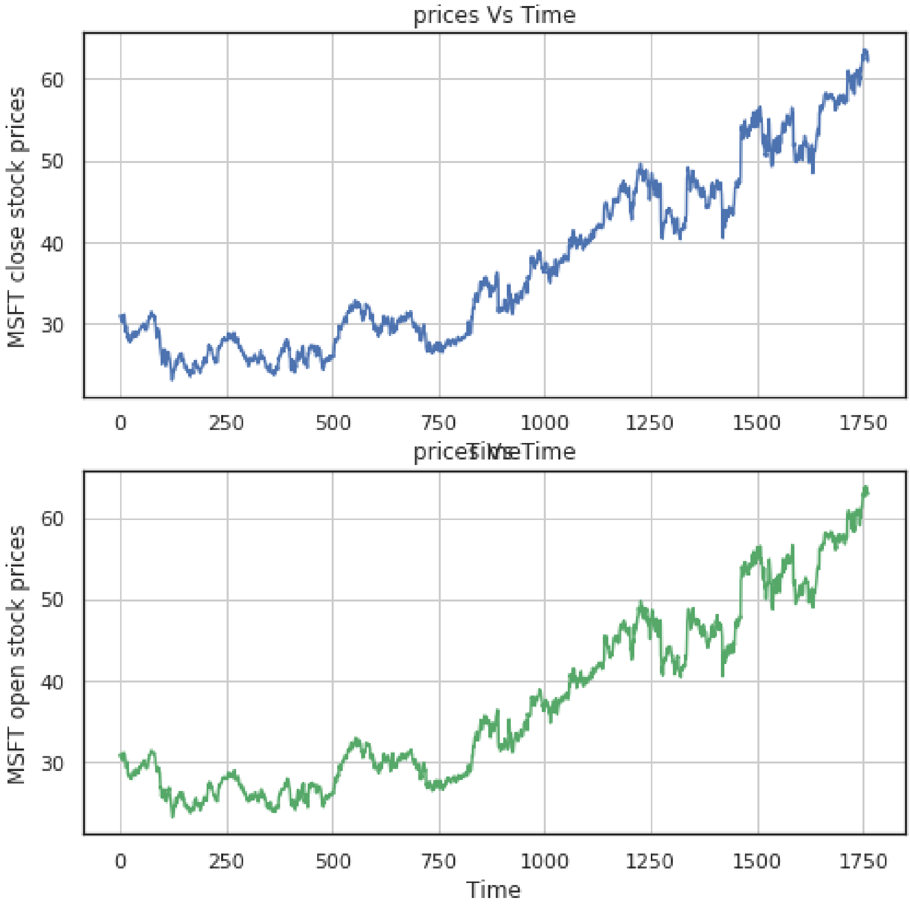
We notice that the differences is marginal , you might take open and close same but they are actually different after looking carefully.

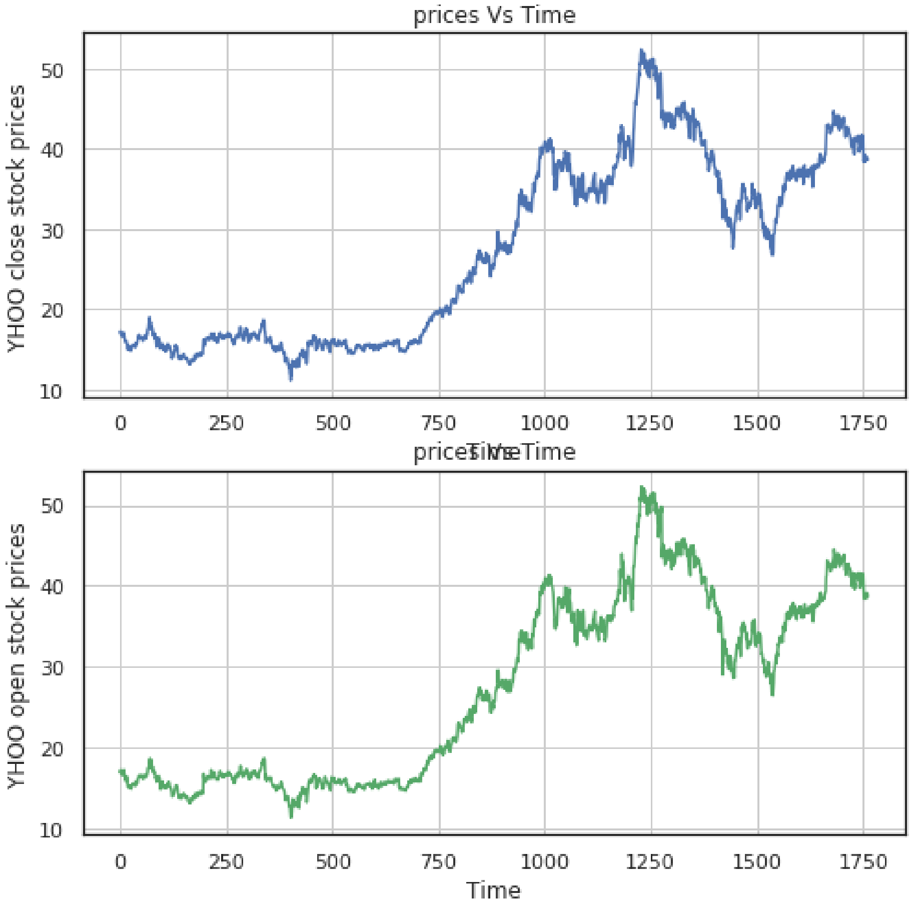
In [0]:

```
def plotter(code):
    global closing_stock
    global opening_stock
    f, axs = plt.subplots(2,2,figsize=(8,8))
    plt.subplot(212)
    company = prices[prices['symbol']==code]
    company = company.open.values.astype('float32')
    company = company.reshape(-1, 1)
    opening_stock = company
    plt.grid(True)
    plt.xlabel('Time')
    plt.ylabel(code + " open stock prices")
    plt.title('prices Vs Time')
    plt.plot(company , 'g')

    plt.subplot(211)
    company_close = prices[prices['symbol']==code]
    company_close = company_close.close.values.astype('float32')
    company_close = company_close.reshape(-1, 1)
    closing_stock = company_close
    plt.xlabel('Time')
    plt.ylabel(code + " close stock prices")
    plt.title('prices Vs Time')
    plt.grid(True)
    plt.plot(company_close , 'b')
    plt.show()
for i in comp_plot:
    plotter(i)
```









Data Manufacturing

Data Manufacturing

Feature Engineering

- **The shape should be: |symbol| x |new features|**
- **Create a standard deviation for each stock**
- **Create a count for each positive day changes that occured per stock**

In [0]:

```
from __future__ import division

# Get each symbols standard deviation in time
ss = price_split.groupby(by=["symbol"])["change"].std()

# Count each symbol groups positive day change in time
## NOTE: it is a proportion to account for some days not appearing for some symbols in ti
me
pcs = price_split.groupby('symbol').apply(lambda grp: grp[grp['change'] > 0]['change'].c
ount() / grp['change'].size)

avgv = price_split.groupby(by=['symbol'])['volume'].mean()/10000000

newdf = pd.concat([ss, pcs, avgv], axis=1).reset_index() #create new dataframe
newdf.columns = ['symbol', 'std', 'prop_pos_day_change', "avg_volume"] #set the names fo
r columns
newdf.head()
```

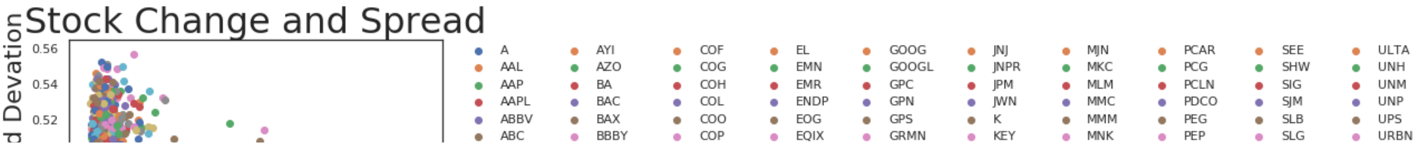
Out[0]:

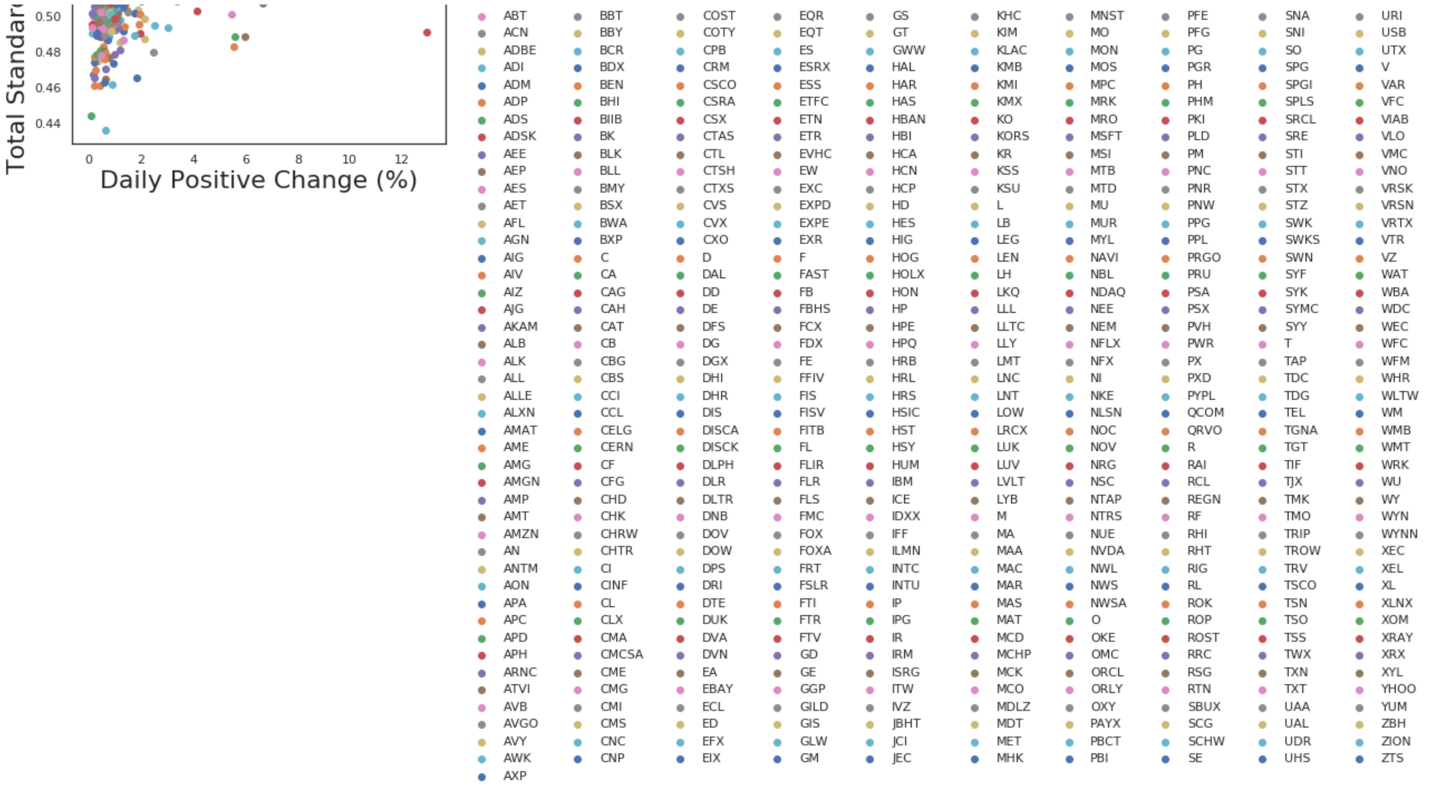
	symbol	std	prop_pos_day_change	avg_volume
0	A	0.483919	0.520431	0.392759
1	AAL	0.583850	0.482974	0.935404
2	AAP	1.526761	0.513053	0.102203
3	AAPL	1.094628	0.507378	9.422578
4	ABBV	0.850244	0.542659	0.847079

Visualize our spread and proportion of GOOD change . i.e., Lower standard deviation and high positive change is the ideal stock choice.

In [0]:

```
for i in newdf['symbol'].tolist():
    x = newdf[newdf['symbol'] == i]['std']
    y = newdf[newdf['symbol'] == i]['prop_pos_day_change']
    plt.scatter(x,y)

legend = plt.legend(newdf['symbol'].tolist(),
           bbox_to_anchor=(1.05, 1),
           loc=2,
           borderaxespad=0.,
          ncol=10,
           frameon = 1)
frame = legend.get_frame()
frame.set_color('white')

plt.title(r'Stock Change and Spread', fontsize=32)
plt.xlabel('Daily Positive Change (%)', fontsize=22)
plt.ylabel('Total Standard Devation', fontsize=22)
plt.figure(figsize=(10,100))
plt.show()
```





```
<Figure size 720x7200 with 0 Axes>
```

Honestly, we think this chart is not really helpful as it contains so many companies that we are not able to figure out each individual name.

Cluster Data Points

K-means

In [0]:

```
from sklearn.cluster import KMeans

# Visualize K = {3..9}
kValues = [i for i in range(3,10)]
for k in kValues:
    kmeans = KMeans(n_clusters=k, random_state=0).fit(newdf[['std','prop_pos_day_change'
]].as_matrix())
    newdf[str(k)] = kmeans.labels_
    print("Finished k=", k)
#we take "symbol", 'std', 'prop_pos_day_change' as identifier variables
newdf = pd.melt(newdf,
                id_vars=["symbol", 'std', 'prop_pos_day_change'],
                var_name="k",
                value_name="values",
                value_vars=list(newdf.columns[-7:]))

newdf.head()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: FutureWarning: Method .as
_matrix will be removed in a future version. Use .values instead.
```

```
Finished k= 3
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: FutureWarning: Method .as
_matrix will be removed in a future version. Use .values instead.
```

```
Finished k= 4
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: FutureWarning: Method .as
_matrix will be removed in a future version. Use .values instead.
```

```
Finished k= 5
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: FutureWarning: Method .as
_matrix will be removed in a future version. Use .values instead.
```

```
Finished k= 6
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: FutureWarning: Method .as
_matrix will be removed in a future version. Use .values instead.
```

```
Finished k= 7
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: FutureWarning: Method .as
_matrix will be removed in a future version. Use .values instead.
```

```
Finished k= 8
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: FutureWarning: Method .as
_matrix will be removed in a future version. Use .values instead.
```

```
Finished k= 9
```

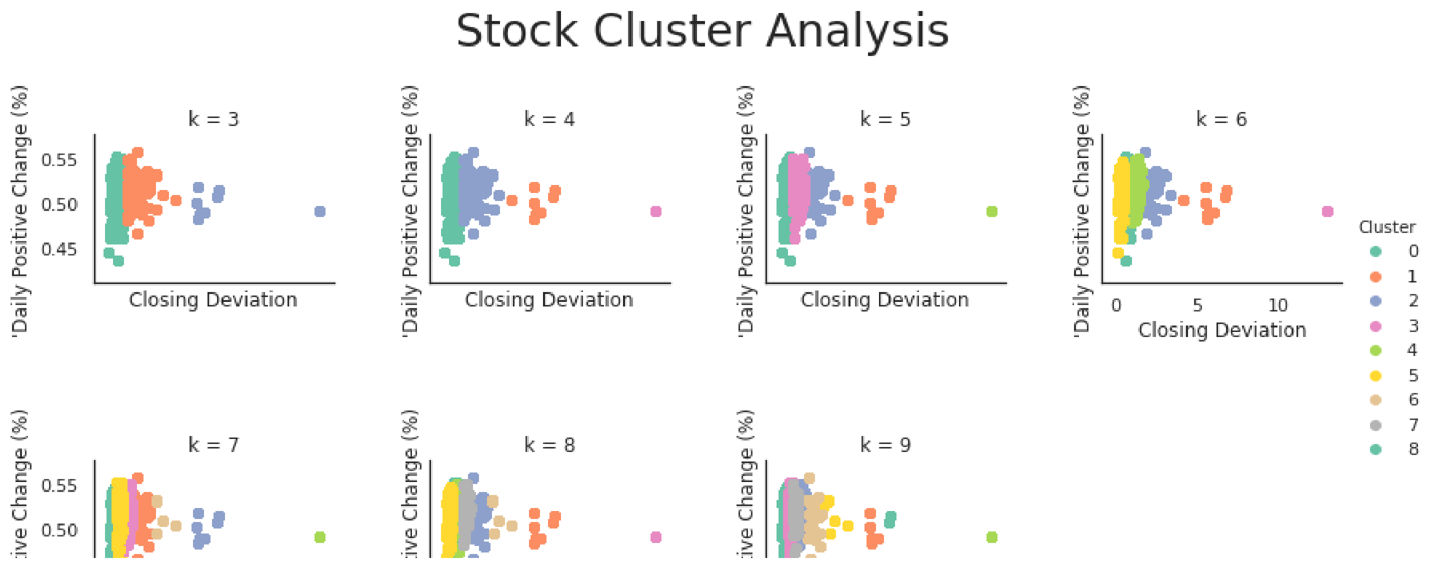
Out[0]:

	symbol	std	prop_pos_day_change	k	values
0	A	0.483919	0.520431	3	0
1	AAL	0.583850	0.482974	3	0
2	AAP	1.526761	0.513053	3	1
3	AAPL	1.094628	0.507378	3	0
4	ABBV	0.850244	0.542659	3	0

Visualize Cluster Analysis

In [0]:

```
g = sns.FacetGrid(newdf, col="k", hue="values", col_wrap=4, palette='Set2')
g = g.map(plt.scatter, "std", "prop_pos_day_change")
g.set(xlabel="Closing Deviation")
g.set(ylabel="'Daily Positive Change (%)")
g.fig.suptitle("Stock Cluster Analysis", size=28)
g.fig.subplots_adjust(top=.8)
plt.subplots_adjust(hspace=1.2, wspace=0.4)
g.add_legend()
g._legend.set_title("Cluster")
```



'Daily Posit
0.45
0
5
10
Closing Deviation
'Daily Posit
0
5
10
Closing Deviation
'Daily Posit
0
5
10
Closing Deviation