project-3

August 26, 2023

1 Problem Statement

2 For investors to properly manage their portfolios,they need to visualize dataset,find useful patterns,and gain valuable insights such as stock daily returns and risks

IMPORT DATASETS AND LIBRARIES

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from copy import copy
from scipy import stats
import plotly.express as px
import plotly.figure_factory as ff
import plotly.graph_objects as go
```

[2]:
```
stocks_df = pd.read_csv('stock.csv')
stocks_df
```

[2]:

	Date	AAPL	BA	T	MGM	AMZN	\
0	2012-01-12	60.198570	75.510002	30.120001	12.130000	175.929993	
1	2012-01-13	59.972858	74.599998	30.070000	12.350000	178.419998	
2	2012-01-17	60.671429	75.239998	30.250000	12.250000	181.660004	
3	2012-01-18	61.301430	75.059998	30.330000	12.730000	189.440002	
4	2012-01-19	61.107143	75.559998	30.420000	12.800000	194.449997	
…	…	…	…	…	…	…	
2154	2020-08-05	440.250000	174.279999	29.850000	16.719999	3205.030029	
2155	2020-08-06	455.609985	172.199997	29.840000	18.459999	3225.000000	
2156	2020-08-07	444.450012	170.020004	30.020000	19.030001	3167.459961	
2157	2020-08-10	450.910004	179.410004	30.200001	21.650000	3148.159912	
2158	2020-08-11	437.500000	180.130005	30.200001	21.500000	3080.669922	

	IBM	TSLA	GOOG	sp500
0	180.550003	28.250000	313.644379	1295.500000
1	179.160004	22.790001	311.328064	1289.089966

```
2     180.000000    26.600000   313.116364  1293.670044
3     181.070007    26.809999   315.273285  1308.040039
4     180.520004    26.760000   318.590851  1314.500000
…          …          …           …           …
2154  125.449997  1485.020020  1473.609985  3327.770020
2155  126.120003  1489.579956  1500.099976  3349.159912
2156  124.959999  1452.709961  1494.489990  3351.280029
2157  127.110001  1418.569946  1496.099976  3360.469971
2158  126.750000  1374.390015  1480.319946  3333.689941

[2159 rows x 10 columns]
```

[3]:
```
# Sort the stock data by date
stocks_df = stocks_df.sort_values(by = ['Date'])
stocks_df
```

[3]:
```
            Date        AAPL          BA           T        MGM          AMZN  \
0     2012-01-12   60.198570   75.510002  30.120001  12.130000   175.929993
1     2012-01-13   59.972858   74.599998  30.070000  12.350000   178.419998
2     2012-01-17   60.671429   75.239998  30.250000  12.250000   181.660004
3     2012-01-18   61.301430   75.059998  30.330000  12.730000   189.440002
4     2012-01-19   61.107143   75.559998  30.420000  12.800000   194.449997
…          …          …           …          …          …          …
2154  2020-08-05  440.250000  174.279999  29.850000  16.719999  3205.030029
2155  2020-08-06  455.609985  172.199997  29.840000  18.459999  3225.000000
2156  2020-08-07  444.450012  170.020004  30.020000  19.030001  3167.459961
2157  2020-08-10  450.910004  179.410004  30.200001  21.650000  3148.159912
2158  2020-08-11  437.500000  180.130005  30.200001  21.500000  3080.669922

             IBM        TSLA        GOOG        sp500
0     180.550003   28.250000   313.644379  1295.500000
1     179.160004   22.790001   311.328064  1289.089966
2     180.000000   26.600000   313.116364  1293.670044
3     181.070007   26.809999   315.273285  1308.040039
4     180.520004   26.760000   318.590851  1314.500000
…          …          …           …           …
2154  125.449997  1485.020020  1473.609985  3327.770020
2155  126.120003  1489.579956  1500.099976  3349.159912
2156  124.959999  1452.709961  1494.489990  3351.280029
2157  127.110001  1418.569946  1496.099976  3360.469971
2158  126.750000  1374.390015  1480.319946  3333.689941

[2159 rows x 10 columns]
```

[4]:
```
# Print out the number of stocks
print('Total Number of stocks : {}'.format(len(stocks_df.columns[1:])))
```

```
Total Number of stocks : 9
```

[5]:
```
# Print the name of stocks
print('Stocks under consideration are:')

for i in stocks_df.columns[1:]:
  print(i)
```

```
Stocks under consideration are:
AAPL
BA
T
MGM
AMZN
IBM
TSLA
GOOG
sp500
```

[7]:
```
stocks_df.describe()
```

[7]:

	AAPL	BA	T	MGM	AMZN	\
count	2159.000000	2159.000000	2159.000000	2159.000000	2159.000000	
mean	140.819823	189.942700	35.162899	23.105743	915.665665	
std	70.827601	103.678586	3.207490	6.963847	697.838905	
min	55.790001	67.239998	26.770000	7.140000	175.929993	
25%	89.165714	124.015000	33.040001	18.545000	316.490005	
50%	116.599998	142.419998	34.930000	23.780001	676.010010	
75%	175.019997	297.044998	37.419998	28.430000	1593.645019	
max	455.609985	440.619995	43.470001	38.029999	3225.000000	

	IBM	TSLA	GOOG	sp500
count	2159.000000	2159.000000	2159.000000	2159.000000
mean	161.853001	259.600815	783.712512	2218.749554
std	25.561938	210.988003	334.448057	537.321727
min	94.769997	22.790001	278.481171	1278.040039
25%	142.769997	184.595001	527.214416	1847.984985
50%	156.949997	231.960007	737.599976	2106.629883
75%	185.974998	307.350006	1079.744995	2705.810059
max	215.800003	1643.000000	1568.489990	3386.149902

PERFORM EXPLORATORY DATA ANALYSIS AND BASIC VISUALIZATION

[8]:
```
# Check if data contains any null values
stocks_df.isnull().sum()
```

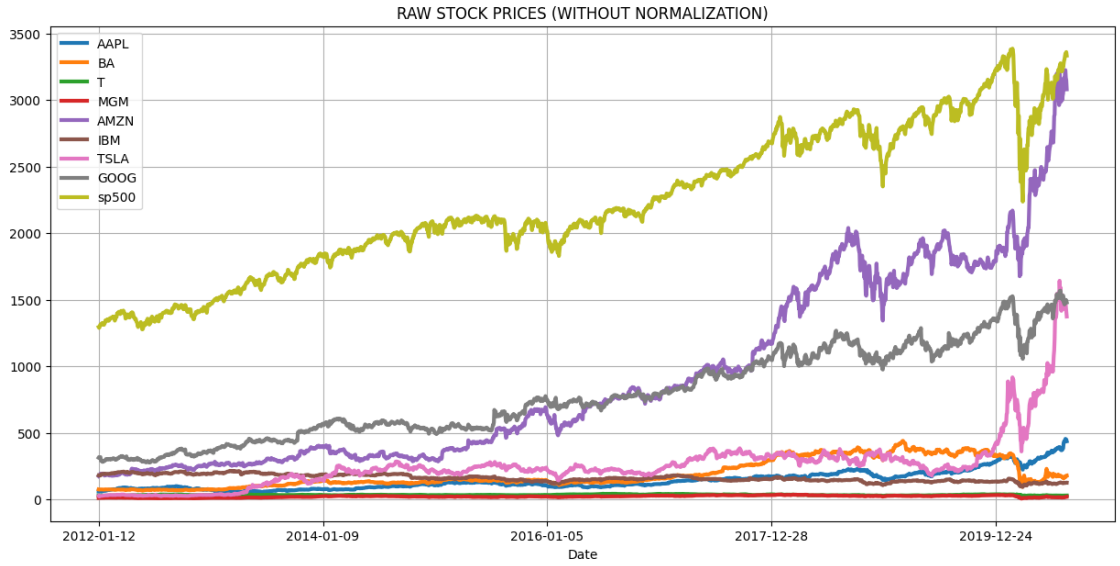
[8]:
```
Date     0
AAPL     0
```

```
BA        0
T         0
MGM       0
AMZN      0
IBM       0
TSLA      0
GOOG      0
sp500     0
dtype: int64
```

[9]:
```
# Getting dataframe info
stocks_df.info()
```

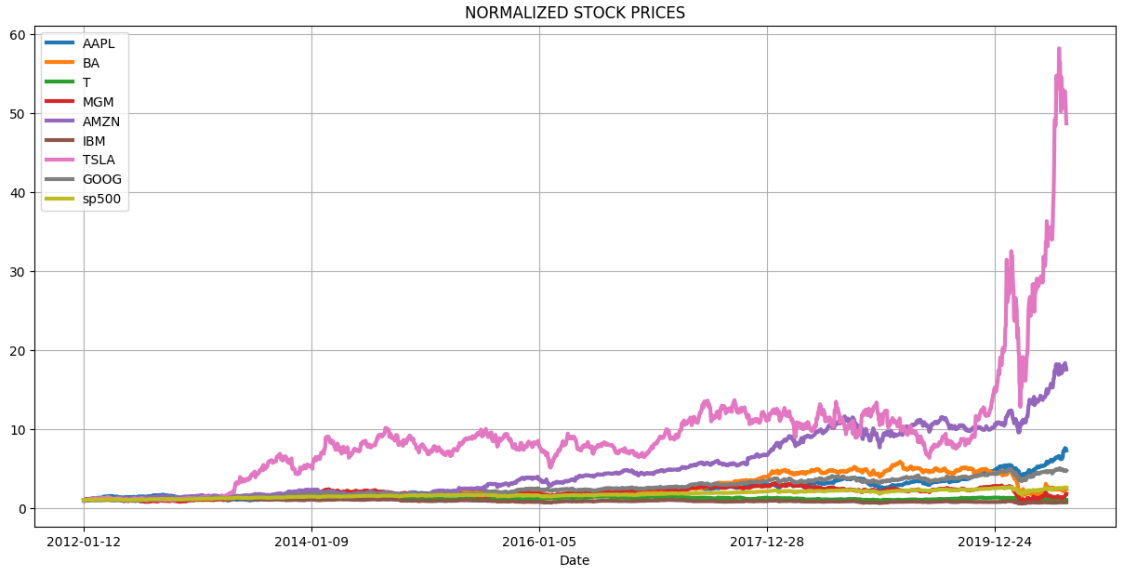
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2159 entries, 0 to 2158
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    2159 non-null   object
 1   AAPL    2159 non-null   float64
 2   BA      2159 non-null   float64
 3   T       2159 non-null   float64
 4   MGM     2159 non-null   float64
 5   AMZN    2159 non-null   float64
 6   IBM     2159 non-null   float64
 7   TSLA    2159 non-null   float64
 8   GOOG    2159 non-null   float64
 9   sp500   2159 non-null   float64
dtypes: float64(9), object(1)
memory usage: 168.8+ KB
```

[11]:
```
# Define a function to plot the entire dataframe
# The function takes in a dataframe df as an input argument and does not return␣
 ↪anything back!
# The function performs data visualization
# Pandas works great with matplotlib, you can simply plot data directly from a␣
 ↪Pandas DataFrame using plot() method

def show_plot(df, fig_title):
  df.plot(x = 'Date', figsize = (15,7), linewidth = 3, title = fig_title)
  plt.grid()
  plt.show()
show_plot(stocks_df, 'RAW STOCK PRICES (WITHOUT NORMALIZATION)')
```



normalized (scaled) stock prices

[12]:
```
# Function to normalize the prices based on the initial price
# The function simply divides every stock by it's price at the start date (i.e.:
  ↪ Date = 2012-01-12)
def normalize(df):
  x = df.copy()

  # Loop through each stock (while ignoring time columns with index 0)
  for i in x.columns[1:]:
    x[i] = x[i]/x[i][0]
  return x
show_plot(normalize(stocks_df), 'NORMALIZED STOCK PRICES')
```



Notice the massive gains in Tesla Stock compared to Amazon (second place)

PERFORM INTERACTIVE DATA VISUALIZATION

[13]:
```
# Function to perform an interactive data plotting using plotly express
# Plotly.express module which is imported as px includes functions that can␣
 ↪plot interactive plots easily and effectively.
# Every Plotly Express function uses graph objects internally and returns a␣
 ↪plotly.graph_objects.Figure instance.

def interactive_plot(df, title):
  fig = px.line(title = title)

  # Loop through each stock (while ignoring time columns with index 0)
  for i in df.columns[1:]:
    fig.add_scatter(x = df['Date'], y = df[i], name = i) # add a new Scatter␣
 ↪trace

  fig.show()
interactive_plot(stocks_df, 'Prices')
```

Plot normalized stock data in an interactive way

[14]:
```
# Plot normalized interactive chart
interactive_plot(normalize(stocks_df), 'Normalized Prices')
```

CALCULATE INDIVIDUAL STOCKS DAILY RETURNS

[15]:
```
# Let's calculate daily return for a single security
# Let's take the S&P500 as an example first

df = stocks_df['sp500']

# Define a dataframe names df_daily_return
df_daily_return = df.copy()

#Loop through every element in the dataframe
for j in range(1, len(df)):

  # Calculate the percentage of change from the previous day
  df_daily_return[j] = ((df[j]- df[j-1])/df[j-1]) * 100

# put zero in the first line item
df_daily_return[0] = 0
df_daily_return
```

[15]:
```
0       0.000000
1      -0.494792
2       0.355295
3       1.110793
4       0.493866
          …
2154    0.642974
2155    0.642770
2156    0.063303
2157    0.274222
2158   -0.796913
Name: sp500, Length: 2159, dtype: float64
```

for tesla stock

[19]:
```
df = stocks_df['TSLA']
df
```

[19]:
```
0          28.250000
1          22.790001
2          26.600000
3          26.809999
4          26.760000
            …
2154    1485.020020
2155    1489.579956
2156    1452.709961
2157    1418.569946
2158    1374.390015
Name: TSLA, Length: 2159, dtype: float64
```

CALCULATE MULTIPLE STOCKS DAILY RETURNS

[22]:
```
# Let's define a function to calculate stocks daily returns (for all stocks)
def daily_return(df):
  df_daily_return = df.copy()

  # Loop through each stock (while ignoring time columns with index 0)
  for i in df.columns[1:]:

    # Loop through each row belonging to the stock
    for j in range(1, len(df)):

      # Calculate the percentage of change from the previous day
      df_daily_return[i][j] = ((df[i][j]- df[i][j-1])/df[i][j-1]) * 100

    # set the value of first row to zero since the previous value is not␣
 ↪available
    df_daily_return[i][0] = 0

  return df_daily_return

stocks_daily_return = daily_return(stocks_df)

stocks_daily_return
```

[22]:

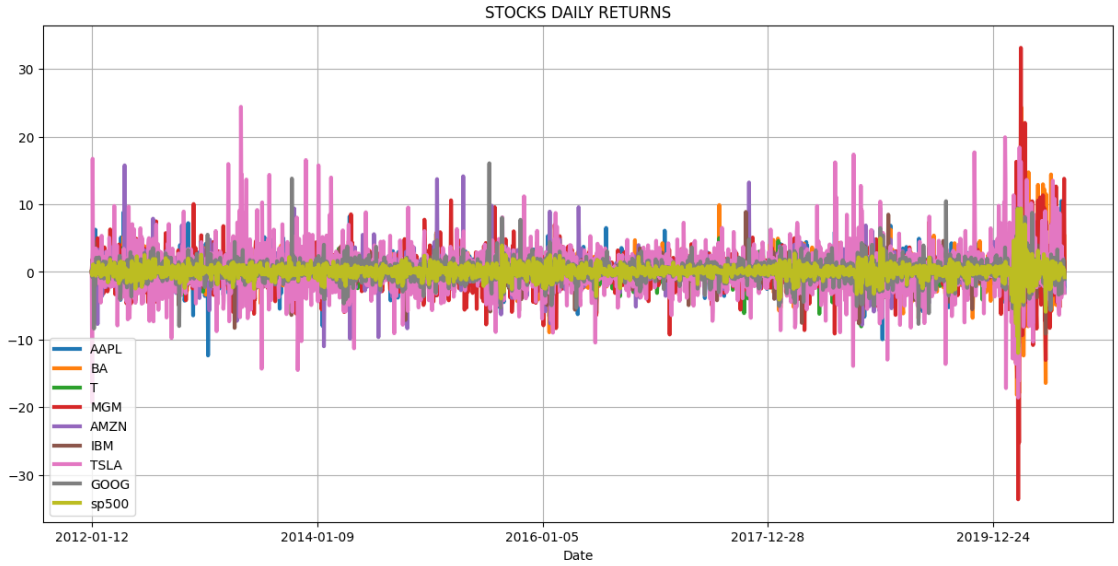
	Date	AAPL	BA	T	MGM	AMZN	IBM	\
0	2012-01-12	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
1	2012-01-13	-0.374946	-1.205144	-0.166006	1.813685	1.415339	-0.769869	
2	2012-01-17	1.164812	0.857909	0.598603	-0.809717	1.815943	0.468852	
3	2012-01-18	1.038382	-0.239234	0.264463	3.918367	4.282725	0.594448	
4	2012-01-19	-0.316937	0.666134	0.296736	0.549882	2.644634	-0.303752	
…	…	…	…	…	…	…	…	
2154	2020-08-05	0.362467	5.579446	-0.533156	0.000000	2.109065	-0.309917	
2155	2020-08-06	3.488923	-1.193483	-0.033501	10.406699	0.623082	0.534082	
2156	2020-08-07	-2.449458	-1.265966	0.603217	3.087768	-1.784187	-0.919762	
2157	2020-08-10	1.453480	5.522880	0.599604	13.767729	-0.609323	1.720552	
2158	2020-08-11	-2.973987	0.401316	0.000000	-0.692841	-2.143792	-0.283220	

	TSLA	GOOG	sp500
0	0.000000	0.000000	0.000000
1	-19.327430	-0.738516	-0.494792
2	16.717854	0.574410	0.355295
3	0.789470	0.688856	1.110793
4	-0.186494	1.052283	0.493866
…	…	…	…
2154	-0.133153	0.589774	0.642974
2155	0.307062	1.797626	0.642770
2156	-2.475194	-0.373974	0.063303

```
2157  -2.350092  0.107728  0.274222
2158  -3.114399 -1.054744 -0.796913

[2159 rows x 10 columns]
```

Plot the returns vs. time using both static and interactive plots

[23]:
```
show_plot(stocks_daily_return, 'STOCKS DAILY RETURNS')
```



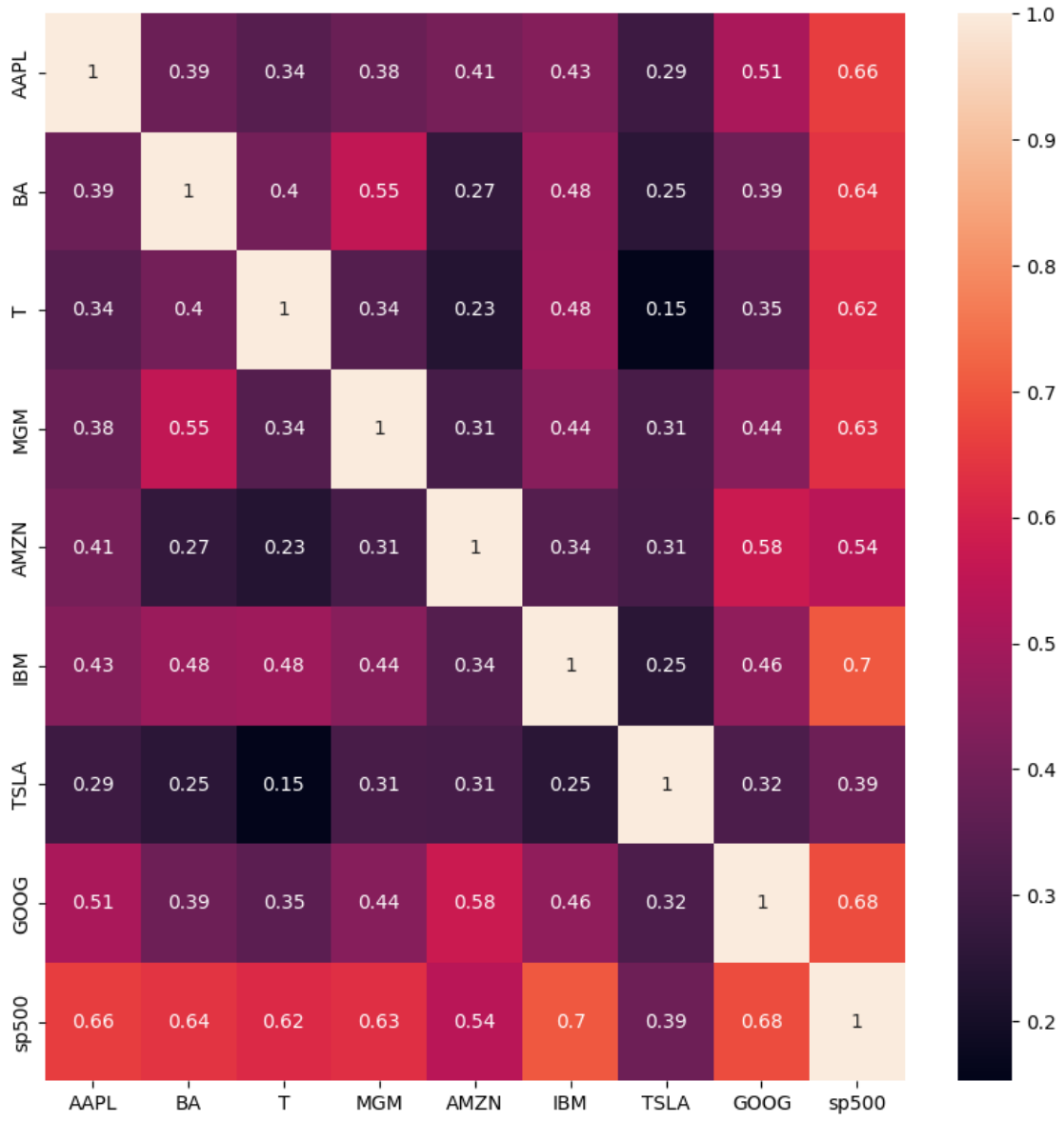
Notice huge drops in MGM around March 2020 (Pandemic effect)

[26]:
```
# Plot the interactive chart

interactive_plot(stocks_daily_return, 'STOCKS DAILY RETURNS')
```

Zoom in to view the sharp decline during March 2020

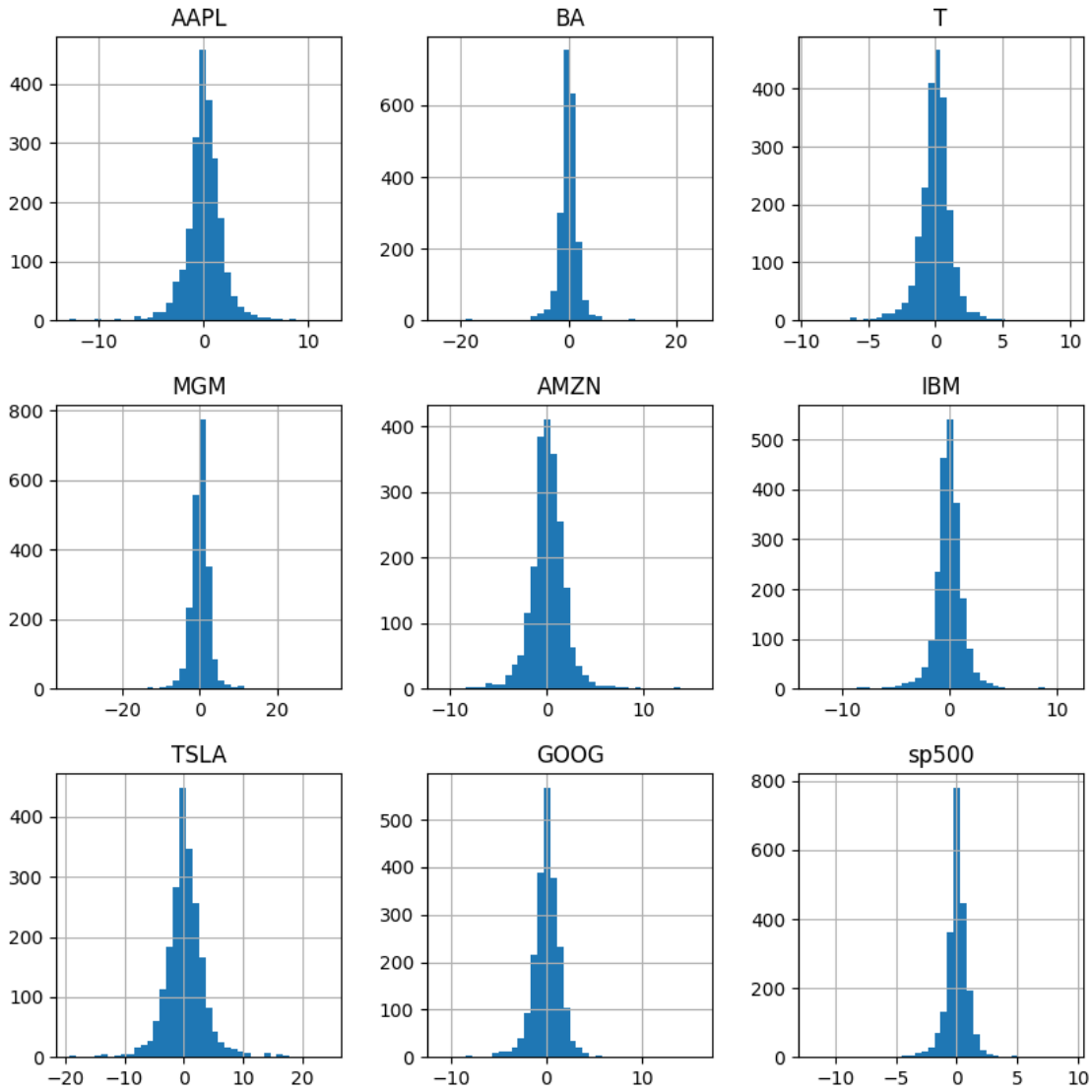
CALCULATE THE CORRELATIONS BETWEEN DAILY RETURNS

[24]:
```
# Daily Return Correlation
cm = stocks_daily_return.drop(columns = ['Date']).corr()

plt.figure(figsize=(10, 10))
ax = plt.subplot()
sns.heatmap(cm, annot = True, ax = ax);
```



PLOT THE HISTOGRAM FOR DAILY RETURNS

[25]:
```
# Histogram of daily returns
# Stock returns are normally distributed with zero mean

stocks_daily_return.hist(figsize=(10, 10), bins = 40);
```



Based on the histogram, which of the following stocks are more risky? T or TSLA

[27]:
```
# Group all data returns together in a list
# Make a copy of the daily returns dataframe
df_hist = stocks_daily_return.copy()

# Drop the date
df_hist = df_hist.drop(columns = ['Date'])

data = []

# Loop through every column
for i in df_hist.columns:
```

```
  data.append(stocks_daily_return[i].values)
data
```

[27]:
```
[array([ 0.        , -0.37494578,  1.16481192, …, -2.44945751,
         1.45347999, -2.9739868 ]),
 array([ 0.        , -1.20514366,  0.85790887, …, -1.26596576,
         5.52287953,  0.40131597]),
 array([ 0.        , -0.16600597,  0.59860326, …,  0.60321716,
         0.5996036 ,  0.        ]),
 array([ 0.        ,  1.81368508, -0.8097166 , …,  3.08776831,
        13.76772918, -0.69284065]),
 array([ 0.        ,  1.41533854,  1.8159433 , …, -1.78418726,
        -0.60932259, -2.14379167]),
 array([ 0.        , -0.76986928,  0.46885241, …, -0.91976211,
         1.72055219, -0.28322004]),
 array([  0.        , -19.32743009,  16.71785359, …,  -2.47519409,
         -2.35009162,  -3.1143992 ]),
 array([ 0.        , -0.73851634,  0.57441015, …, -0.37397414,
         0.10772812, -1.05474435]),
 array([ 0.        , -0.49479228,  0.35529545, …,  0.06330295,
         0.27422185, -0.79691324])]
```

[28]:
```
# Plotly's Python API contains a super powerful module known as figure factory␣
 ↪module
# Figure factory module includes wrapper functions that create unique chart␣
 ↪types such as interactive subplots
fig = ff.create_distplot(data, df_hist.columns)
fig.show()
```

Notice how Boeing, Tesla and MGM gains are losses are sometimes extreme! This will indicate a more risky investment and will be quantified later using Sharpe Ratio

[ ]:
```

```