Basic Stock Analysis Using Python

I.W.Twain

11/7/2020

Introduction

In this article, I analyze closing stock prices data of Apple, Amazon.com, JPMorgan Chase, MGM Resorts International, Tesla, Vanguard 500 Index Fund ETF, and Verizon.

```
# Import the needed libraries
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
from tabulate import tabulate
```

Data

Jason Strimpel's bulk stock data series download web system allows anyone to easily download daily stock prices into a csv file. My data ranges from 2012-01-03 to 2020-11-05. You can choose any companies you like by manually entering tickers separated by a comma.

Use pandas' read_csv() function whose input is the absolute path to your csv file. Here is what my data looks like.

```
# Sort the Date column values
print(tabulate(stocks_df.head(), headers = 'keys', tablefmt = 'psql'))
```

```
## +----+------------+---------+--------+---------+---------+--------+---------+---------+
## |    | Date       |    AAPL |   AMZN |     JPM |     MGM |   TSLA |     VOO |      VZ |
## |----+------------+---------+--------+---------+---------+--------+---------+---------|
## |  0 | 2012-01-03 | 12.6696 | 179.03 | 27.0605 | 10.5011 |  5.616 | 97.7553 | 26.4979 |
## |  1 | 2012-01-04 | 12.7377 | 177.51 | 27.2319 | 10.435  |  5.542 | 97.923  | 26.151  |
## |  2 | 2012-01-05 | 12.8791 | 177.61 | 27.8007 | 10.52   |  5.424 | 98.2416 | 25.971  |
## |  3 | 2012-01-06 | 13.0137 | 182.61 | 27.5514 | 10.52   |  5.382 | 98.0404 | 25.8967 |
## |  4 | 2012-01-09 | 12.9931 | 178.56 | 27.5047 | 10.6996 |  5.45  | 98.1913 | 25.9237 |
## +----+------------+---------+--------+---------+---------+--------+---------+---------+
```

Data Analysis and Visulization

Note: If you want to view an interactive plot, check this out.

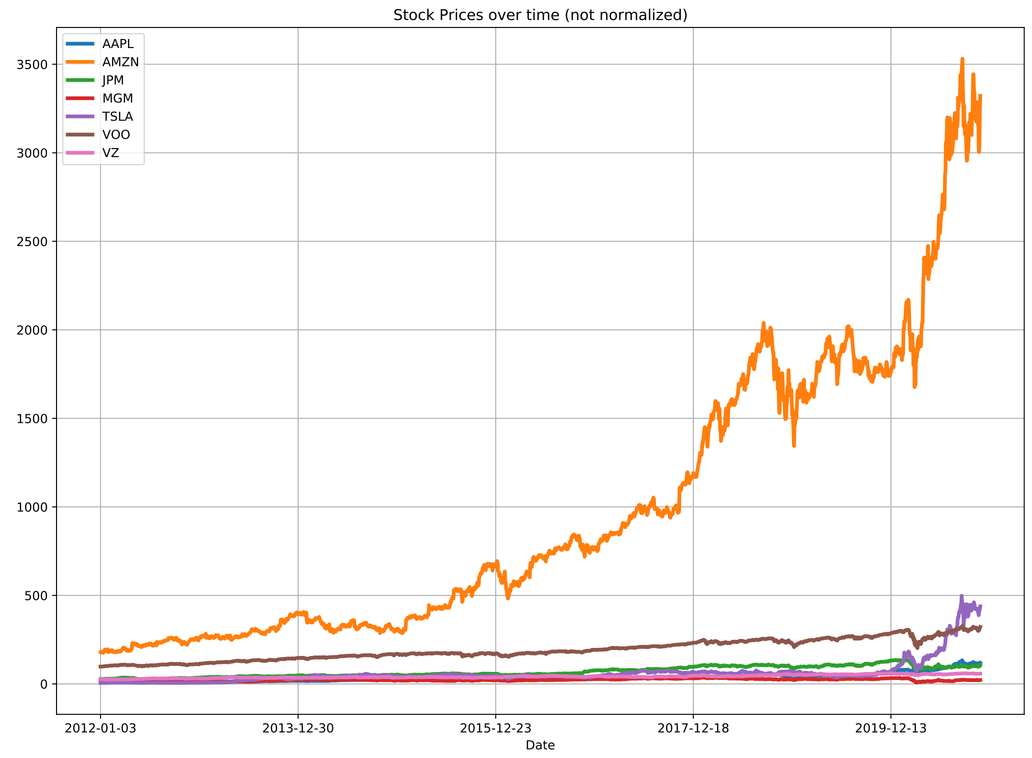
In general, missing values are tricky to deal with. Fortunately, the data does not include any missing values.

```
# How many null values does each column have?
stocks_df.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 2227 entries, 0 to 2226
## Data columns (total 8 columns):
##  #   Column  Non-Null Count  Dtype
## ---  ------  --------------  -----
##  0   Date    2227 non-null   object
##  1   AAPL    2227 non-null   float64
##  2   AMZN    2227 non-null   float64
##  3   JPM     2227 non-null   float64
##  4   MGM     2227 non-null   float64
##  5   TSLA    2227 non-null   float64
##  6   VOO     2227 non-null   float64
##  7   VZ      2227 non-null   float64
## dtypes: float64(7), object(1)
## memory usage: 139.3+ KB
```

Not surprisingly, Amazon has had the highest stock price compared to all other companies in any given year.

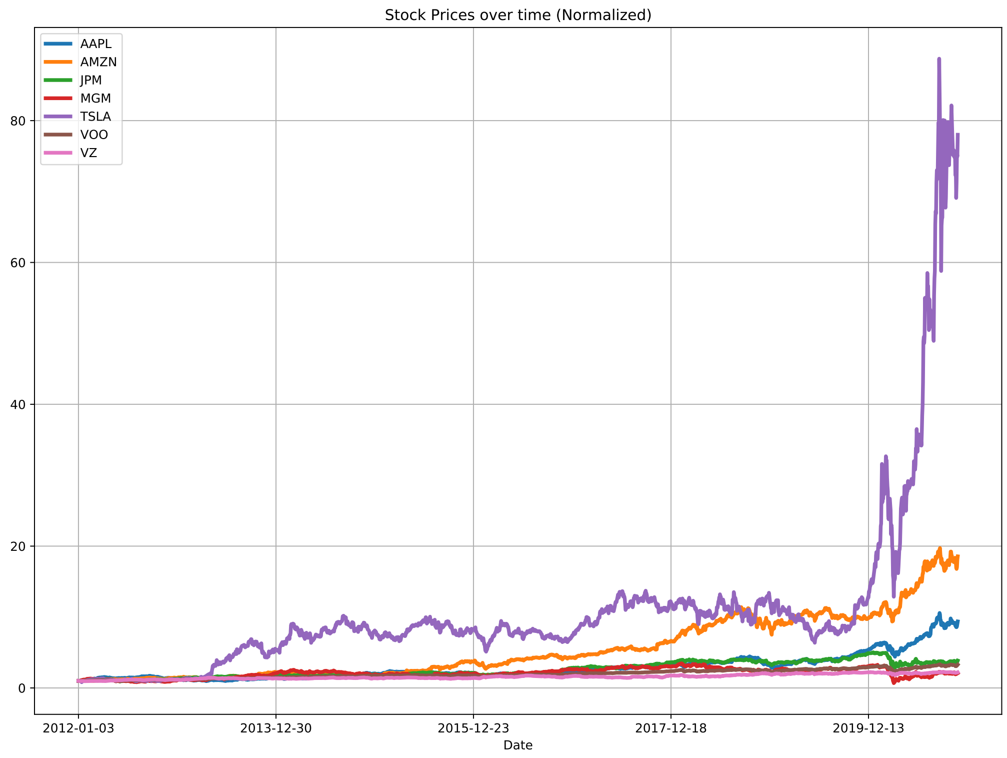
```
# Plot all stock prices versus date using a function
def show_plot(data, title):
    data.plot(x='Date', figsize=(14,10), linewidth=3, title=title)
    plt.grid()
    plt.show()

show_plot(stocks_df, 'Stock Prices over time (not normalized)')
```



But, the stock with the highest price is not always the most profitable. Graphing the normalized prices graphs the price movement of a stock with $1 as the base value in 2012-01-03. Thus, a value of 1.15 means the price has increased 15% since the base date.

Normalizing the prices show that TSLA generated the highest returns.

```
# Plot the normalized stock prices over time

# Normalize the data
def normalize(data):
    x=data.copy() # Create a copy

    # Select every column, except Date, and divide the column by its price on 2012-01-03
    for i in x.columns[1:]:
        x[i] = data[i] / data[i][0]

    return x

stocks_df_normalized = normalize(stocks_df) # Pass stocks_df to normalize()
show_plot(stocks_df_normalized, 'Stock Prices over time (Normalized)') # Plot the data
```



Calculating Daily Returns

Let's see how to calculate daily returns for one stock in Python. Then, I replicate the same process for all the columns in stocks_df. Here are the daily returns for Amazon.com.

```
# We can do the same thing for Amazon Stock
df=stocks_df['AMZN']
daily_returns = df.copy() # pull the values from df, calculate the return, overwrite df_returns

for i in range(1, len(daily_returns)):
    daily_returns[i] = ((df[i] - df[i-1]) / df[i-1]) * 100

daily_returns[0] = 0
print(daily_returns)
```

```
## 0        0.000000
## 1       -0.849022
## 2        0.056338
## 3        2.815157
## 4       -2.217843
##            ...
## 2222    -5.445642
## 2223    -1.043095
## 2224     1.462148
## 2225    6.322969
## 2226    2.494172
## Name: AMZN, Length: 2227, dtype: float64
```

Calculating a stock's daily return is simple. For percent return on 2012-01-04, subtract that day's price from the closing price of 2012-01-03. Finally, divide the difference by closing price of 2012-01-03 and multiply the result by 100.

Extend the same principle to all stock_df columns

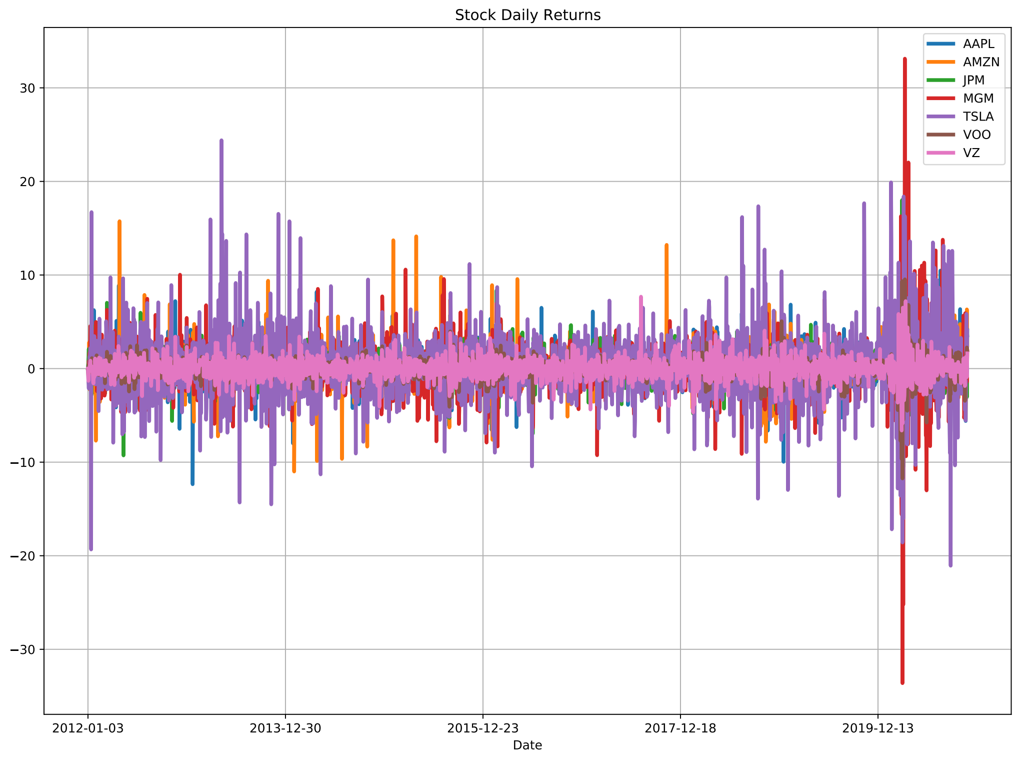
```python
# Basically the same idea must be implemented to every column

def daily_returns(data):
    daily_returns = data.copy()

    for i in data.columns[1:]: # Loop over the column names
        for j in range(1, len(daily_returns)):
            daily_returns[i][j] = ((data[i][j] - data[i][j-1]) / data[i][j-1]) * 100 # Calculate the re
        daily_returns[i][0] = 0 # Change the first value of ith column to 0

    return daily_returns

df_daily_return = daily_returns(stocks_df)
print(tabulate(df_daily_return.head(10), headers = 'keys', tablefmt = 'psql'))
```

```
## +----+------------+-----------+------------+-----------+-----------+-------------+-----------+-------
## |    | Date       |      AAPL |       AMZN |       JPM |       MGM |        TSLA |       VOO |
## |----+------------+-----------+------------+-----------+-----------+-------------+-----------+-------
## |  0 | 2012-01-03 |  0        |  0         |  0        |  0        |   0         |  0        |  0
## |  1 | 2012-01-04 |  0.537453 | -0.849022  |  0.633465 | -0.630057 |  -1.31766   |  0.171536 | -1.308
## |  2 | 2012-01-05 |  1.11022  |  0.0563382 |  2.0887   |  0.815231 |  -2.1292    |  0.325369 | -0.688
## |  3 | 2012-01-06 |  1.04537  |  2.81516   | -0.896883 |  0        |  -0.774336  | -0.204835 | -0.280
## |  4 | 2012-01-09 | -0.15861  | -2.21784   | -0.169636 |  1.70708  |   1.26347   |  0.153935 |  0.104
## |  5 | 2012-01-10 |  0.35806  |  0.436827  |  2.12463  |  1.06006  |   1.3578    |  0.853829 |  0.52
## |  6 | 2012-01-11 | -0.163073 | -0.245345  |  1.69209  |  4.54547  |   2.20854   |  0.101572 |  0.855
## |  7 | 2012-01-12 | -0.274482 | -1.66015   |  0.518251 |  1.4214   |   0.0708466 |  0.219884 |  0.05
## |  8 | 2012-01-13 | -0.374971 |  1.41534   | -2.5237   |  1.81369  | -19.3274    | -0.42194  |  0
## |  9 | 2012-01-17 |  1.16484  |  1.81594   | -2.81184  | -0.809704 |  16.7179    |  0.271192 |  0.250
## +----+------------+-----------+------------+-----------+-----------+-------------+-----------+-------
```

Here is the plot of all stock's daily returns. Evidently, Tesla has the highest volatility throughout all the years as it has the highest peaks and the lowest troughs. Additionally, during the pandemic MGM resorts suffered a massive daily loss and also had the highest return than any other stock a few days later.

Finally, it looks like Verizon and Apple's daily returns do have very high peaks or low troughs.

```python
# Make the static plot
show_plot(df_daily_return, 'Stock Daily Returns')
```


Stock Daily Returns
AAPL
AMZN
JPM
MGM
TSLA
VOO
VZ
30
20
10
0
−10
−20
−30
2012-01-03
2013-12-30
2015-12-23
2017-12-18
2019-12-13
Date