Stocks Analysis

In [22]:

```python
# Provides ways to work with large multidimensional arrays
import numpy as np
# Allows for further data manipulation and analysis
import pandas as pd
from pandas_datareader import data as web # Reads stock data
import matplotlib.pyplot as plt # Plotting
import matplotlib.dates as mdates # Styling dates
import seaborn as sns
%matplotlib inline

import datetime as dt # For defining dates
import mplfinance as mpf # Matplotlib finance
```

In [3]:

```python
# scrapping data for S&P500 stocks name.

payload=pd.read_html('https://en.wikipedia.org/wiki/List_of_S%26P_500_companies')
snp_500 = payload[0]

snp_symbol = snp_500["Symbol"].values.tolist()

snp_symbol[1:15]
```

Out[3]:

```
['ABT',
 'ABBV',
 'ABMD',
 'ACN',
 'ATVI',
 'ADBE',
 'AMD',
 'AAP',
 'AES',
 'AFL',
 'A',
 'APD',
 'AKAM',
 'ALK']
```

Function that Saves Stock Data to CSV

In [4]:

```python
# Function that gets a dataframe by providing a ticker and starting date
def save_to_csv_from_yahoo(ticker, syear, smonth, sday, eyear, emonth, eday):

    # Time period
    start = dt.datetime(syear, smonth, sday)
    end = dt.datetime(eyear, emonth, eday)

    try:
        df = web.DataReader(ticker, 'yahoo', start, end)
        alls = web.DataReader(ticker, 'yahoo', start, end)['Adj Close']
    except:pass

    # adding daily returns as well
    # formula : (closing price/previous closing price) - 1
    df['daily_return'] = (df['Adj Close'] / df['Adj Close'].shift(1)) - 1

    # Save data to a CSV file
    # For Windows
    df.to_csv("D:\Python Class\Python4finance/" + ticker + '.csv')
```

Function that Returns a Dataframe from a CSV

In [5]:

```
# Reads a dataframe from the CSV file, changes index to date and returns it
def get_df_from_csv(ticker):
    # Try to get the file
    try:
        df = pd.read_csv("D:\Python Class\Python4finance/" + ticker + '.csv')
    # through error if file not in the directory
    except FileNotFoundError:
        print("File Not Found In The Folder")
    # otherwise return the file
    else:
        return df
```

Returns Total Return over Time

In [6]:

```
# use daily returns to calculate mean returns over two dates, multiply mean returns to the

def get_return_defined_time(df, syear, smonth, sday, eyear, emonth, eday):
    # Create string representations for the dates
    start = f"{syear}-{smonth}-{sday}"
    end = f"{eyear}-{emonth}-{eday}"
    df['Date'] = pd.to_datetime(df['Date'])

    # Use a mask to grab data between defined dates
    mask = (df['Date'] >= start) & (df['Date'] <= end)

    # Get the mean of the column named daily return
    daily_ret = df.loc[mask]['daily_return'].mean()

    ret_mean = df['daily_return'].mean()

    # Get the number of days between 2 dates
    df2 = df.loc[mask]
    days = df2.shape[0]

    # Return the total return between 2 dates
    return (days * daily_ret)
```

In [ ]:

```

```

Matplotlib Finance

In [7]:

```
# Receives a ticker and the date range for which to plot
def mplfinance_plot(ticker, chart_type, syear, smonth, sday, eyear, emonth, eday):
    # Create string representations for the dates
    start = f"{syear}-{smonth}-{sday}"
    end = f"{eyear}-{emonth}-{eday}"

    try:
        # For Windows
        df = pd.read_csv('D:\Python Class\Python4finance/' + ticker + '.csv',index_col=Nor
    except FileNotFoundError:
        print("File Doesn't Exist")
    else:

        # Set data.index as DatetimeIndex
        df.index = pd.DatetimeIndex(df['Date'])

        # Define to only use data between provided dates
        df_sub = df.loc[start:end]

        # A candlestick chart demonstrates the daily open, high, low and closing price of
        mpf.plot(df_sub,type='candle')

        # Plot price changes
        mpf.plot(df_sub,type='line')

        # Moving averages provide trend information (Average of previous 4 observations)
        mpf.plot(df_sub,type='ohlc',mav=4)

        # Define a built in style
        s = mpf.make_mpf_style(base_mpf_style='charles', rc={'font.size': 8})
        # Pass in the defined style to the whole canvas
        fig = mpf.figure(figsize=(12, 8), style=s)
        # Candle stick chart subplot
        ax = fig.add_subplot(2,1,1)
        # Volume chart subplot
        av = fig.add_subplot(2,1,2, sharex=ax)

        # You can plot multiple MAVs, volume, non-trading days
        mpf.plot(df_sub,type=chart_type, mav=(3,5,7), ax=ax, volume=av, show_nontrading=Tr
```

Simple Price Plot

In [8]:

```
# # Creates a simple price / date plot between dates
# def price_plot(ticker, syear, smonth, sday, eyear, emonth, eday):
#     # Create string representations for the dates
#     start = f"{syear}-{smonth}-{sday}"
#     end = f"{eyear}-{emonth}-{eday}"

#     try:
#         df = pd.read_csv("D:\Python Class\Python4finance/" + ticker + '.csv')
#     except FileNotFoundError:
#         print("File Doesn't Exist")
#     else:

#         # Set data.index as DatetimeIndex
#         df.index = pd.DatetimeIndex(df['Date'])

#         # Define to only use data between provided dates
#         df_sub = df.loc[start:end]

#         # Convert to Numpy array
#         df_np = df_sub.to_numpy()

#         # Get adjusted close data from the 5th column
#         np_adj_close = df_np[:,5]

#         # Get date from the 1st
#         date_arr = df_np[:,1]

#         # Defines area taken up by the plot
#         fig = plt.figure(figsize=(12,8),dpi=100)
#         axes = fig.add_axes([0,0,1,1])

#         # Define the plot line color as navy
#           axes.plot(date_arr, np_adj_close, color='navy')

#           # Set max ticks on the x axis
#           axes.xaxis.set_major_locator(plt.MaxNLocator(8))

#           # Add a grid, color, dashes(5pts 1 pt dashes separated by 2pt space)
#           axes.grid(True, color='0.6', dashes=(5, 2, 1, 2))

#           # Set grid background color
#           axes.set_facecolor('#FAEBD7')
```

Download Multiple Stocks

In [9]:

```
def download_multiple_stocks(syear, smonth, sday, eyear, emonth, eday, *tickers):
    tickers = ["FB", "AMZN", "AAPL", "NFLX", "GOOG"]
    for x in tickers:
        save_to_csv_from_yahoo(x, syear, smonth, sday, eyear, emonth, eday)
```

Merge Multiple Stocks in One Dataframe by Column Name

In [10]:

```
def merge_df_by_column_name(col_name, syear, smonth, sday, eyear, emonth, eday, *market):
    mult_df = pd.DataFrame()

    start = f"{syear}-{smonth}-{sday}"
    end = f"{eyear}-{emonth}-{eday}"


#     for x in snp_symbol (for downloading all s&p500 data)
    for x in tickers:
        try:
            mult_df[x] = web.DataReader(x, 'yahoo', start, end)[col_name]
        except:pass

    return mult_df
```

In [11]:

```
tickers = ["TSLA", "AMD", "ETSY", "DXCM", "GNRC", "AAPL", "NVDA", "ZBRA"]
mult_df = merge_df_by_column_name('Adj Close',  2018, 1, 1, 2021, 12, 1, *tickers)

# for saving all 500 stocks
'''
mult_df = merge_df_by_column_name('Adj Close',  2018, 1, 1, 2021, 12, 1, *snp_symbol)
plot_return = (mult_df / mult_df.iloc[1] * 100)
plot_return.to_csv('D:\Python Class\Python4finance/All_Stocks.csv') '''
```

Out[11]:

```
"\nmult_df = merge_df_by_column_name('Adj Close',  2018, 1, 1, 2021, 12, 1, *snp_symbol)\n
plot_return = (mult_df / mult_df.iloc[1] * 100)\nplot_return.to_csv('D:\\Python Class\\Pyt
hon4finance/All_Stocks.csv') "
```

Get Changing Value of Investment using Multiple Stocks

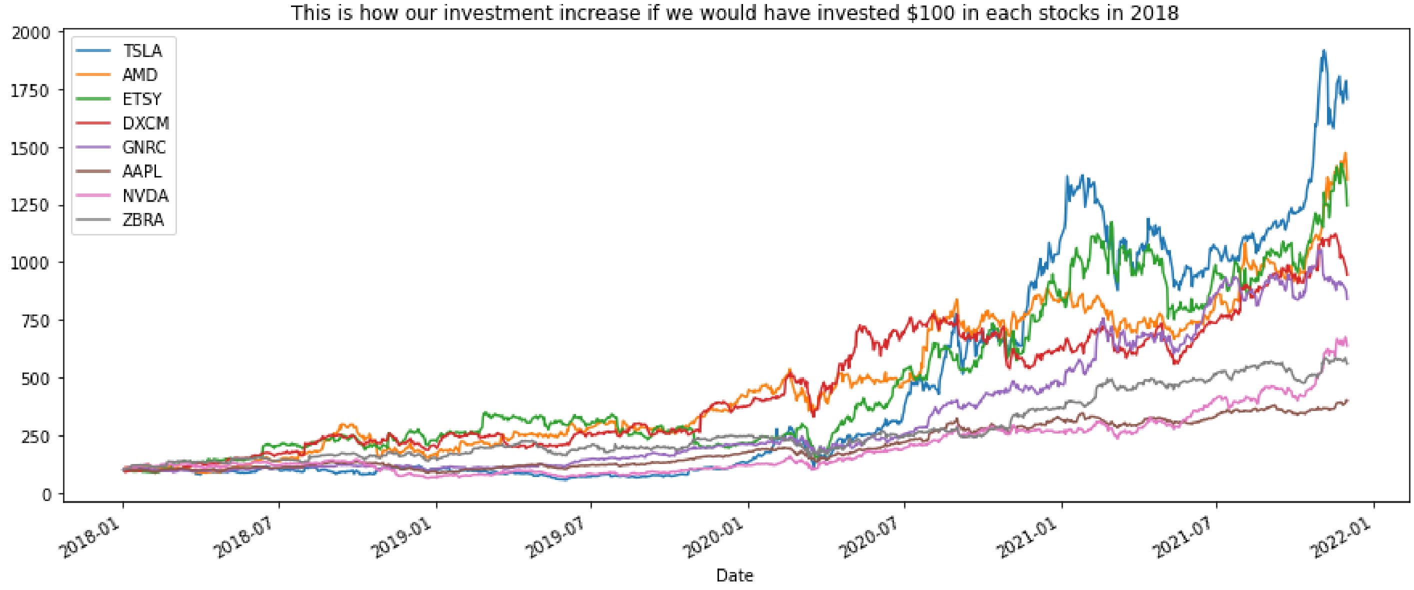
In [12]:

```
def plot_return_mult_stocks(investment, stock_df):
    (stock_df / stock_df.iloc[0] * investment).plot(figsize = (15,6), title = "This is hov
```

Get Standard Deviation for Multiple Stocks

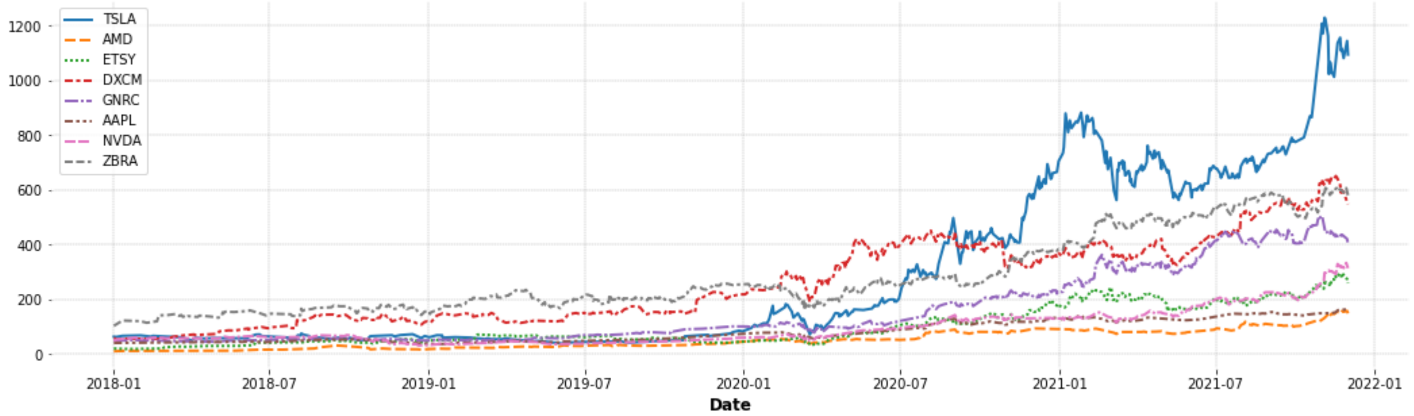
In [13]:

```
plot_return_mult_stocks(100, mult_df)
```


This is how our investment increase if we would have invested $100 in each stocks in 2018

In [38]:

```
plt.figure(figsize = (18, 5))
sns.lineplot(data=mult_df)
plt.show()
```



Get Mean, Standard Deviation and Coefficient of Variation for Multiple Stocks

In [14]:

```
# Receives the dataframe with the Adj Close data along with the stock ticker
# Returns the mean and standard deviation associated with the ticker
def get_stock_mean_sd(stock_df, ticker):
    return stock_df[ticker].mean(), stock_df[ticker].std()
```

In [15]:

```
get_stock_mean_sd(mult_df, 'AAPL')
```

Out[15]:

```
(81.02345724598615, 39.025398936573445)
```

In [16]:

```
# Receives the dataframe with the stock ticker as the column name and
# the Adj Close values as the column data and returns the mean and
# standard deviation
def get_mult_stock_mean_sd(stock_df):
    for stock in stock_df:
        mean, sd = get_stock_mean_sd(stock_df, stock)
        cov = sd / mean
        print("Stock: {:4} Mean: {:7.2f} Standard deviation: {:2.2f}".format(stock, mean,
        print("Coefficient of Variation: {}\n".format(cov))
```

In [17]:

```
get_mult_stock_mean_sd(mult_df)
```

```
Stock: TSLA Mean:  281.99 Standard deviation: 301.97
Coefficient of Variation: 1.0708657442037919

Stock: AMD  Mean:   51.62 Standard deviation: 33.95
Coefficient of Variation: 0.6576292134807044

Stock: ETSY Mean:   98.02 Standard deviation: 69.88
Coefficient of Variation: 0.712872173940725

Stock: DXCM Mean:  259.01 Standard deviation: 154.14
Coefficient of Variation: 0.5951213840323502

Stock: GNRC Mean:  155.38 Standard deviation: 131.31
Coefficient of Variation: 0.8451027329169493

Stock: AAPL Mean:   81.02 Standard deviation: 39.03
Coefficient of Variation: 0.48165556325364944

Stock: NVDA Mean:   94.54 Standard deviation: 62.00
Coefficient of Variation: 0.6557973641591032

Stock: ZBRA Mean:  279.13 Standard deviation: 141.03
Coefficient of Variation: 0.5052377576107893
```

Test Functions

In [18]:

```
# Call to read the data from Yahoo into a CSV and then retrieve a Dataframe
AMZN = save_to_csv_from_yahoo('AMZN', 2020, 1, 1, 2021, 1, 1)

# Retrieve data from the CSV file
AMZN = get_df_from_csv('AMZN')

AMZN
```

Out[18]:

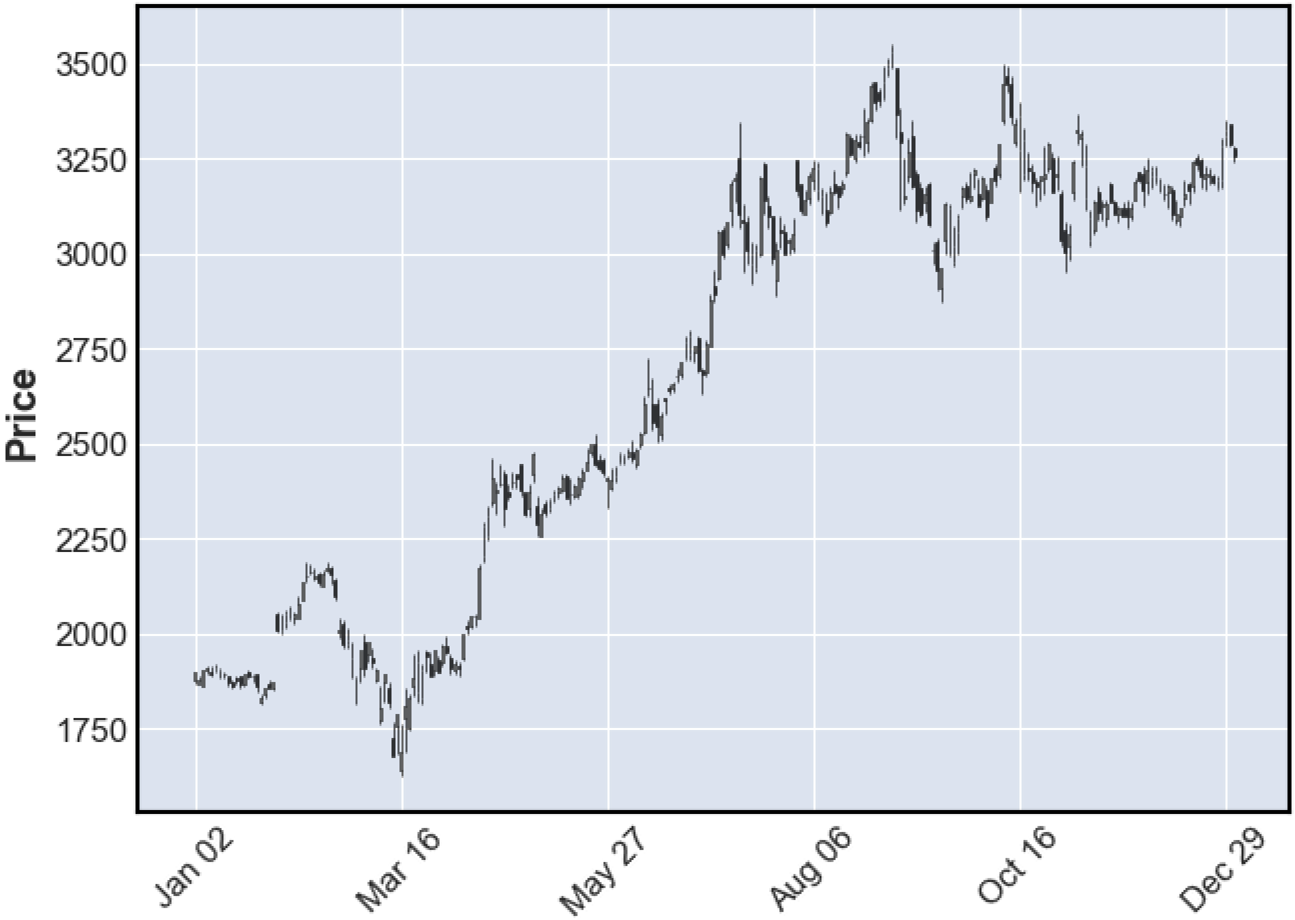
	Date	High	Low	Open	Close	Volume	Adj Close	daily_return
0	2019-12-31	1853.260010	1832.229980	1842.000000	1847.839966	2506500	1847.839966	NaN
1	2020-01-02	1898.010010	1864.150024	1875.000000	1898.010010	4029000	1898.010010	0.027151
2	2020-01-03	1886.199951	1864.500000	1864.500000	1874.969971	3764400	1874.969971	-0.012139
3	2020-01-06	1903.689941	1860.000000	1860.000000	1902.880005	4061800	1902.880005	0.014886
4	2020-01-07	1913.890015	1892.040039	1904.500000	1906.859985	4044900	1906.859985	0.002092
...	...	...	...	...	...	...	...	...
249	2020-12-24	3202.000000	3169.000000	3193.899902	3172.689941	1451900	3172.689941	-0.003949
250	2020-12-28	3304.000000	3172.689941	3194.000000	3283.959961	5686800	3283.959961	0.035071
251	2020-12-29	3350.649902	3281.219971	3309.939941	3322.000000	4872900	3322.000000	0.011584
252	2020-12-30	3342.100098	3282.469971	3341.000000	3285.850098	3209300	3285.850098	-0.010882
253	2020-12-31	3282.919922	3241.199951	3275.000000	3256.929932	2957200	3256.929932	-0.008801

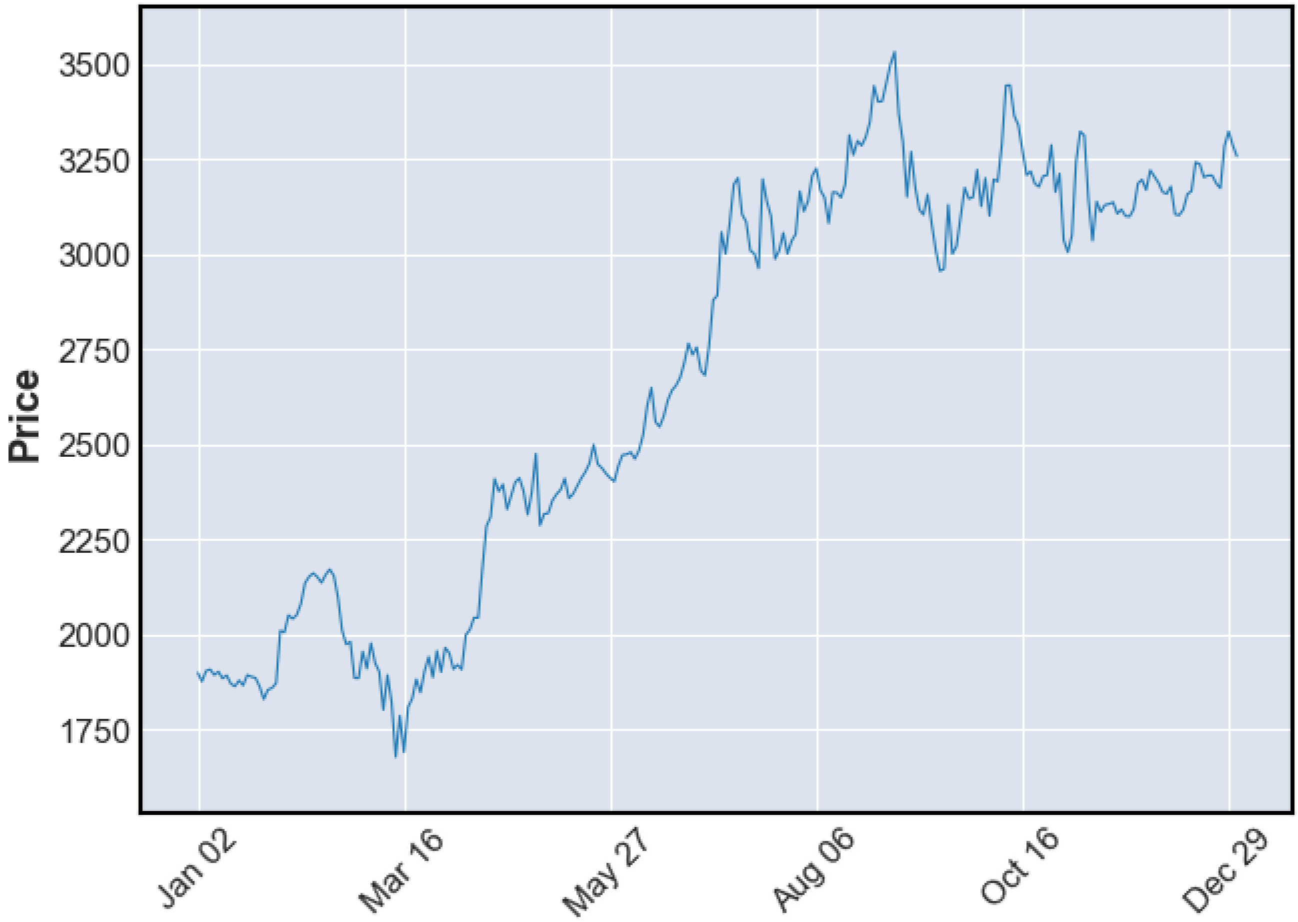
254 rows × 8 columns

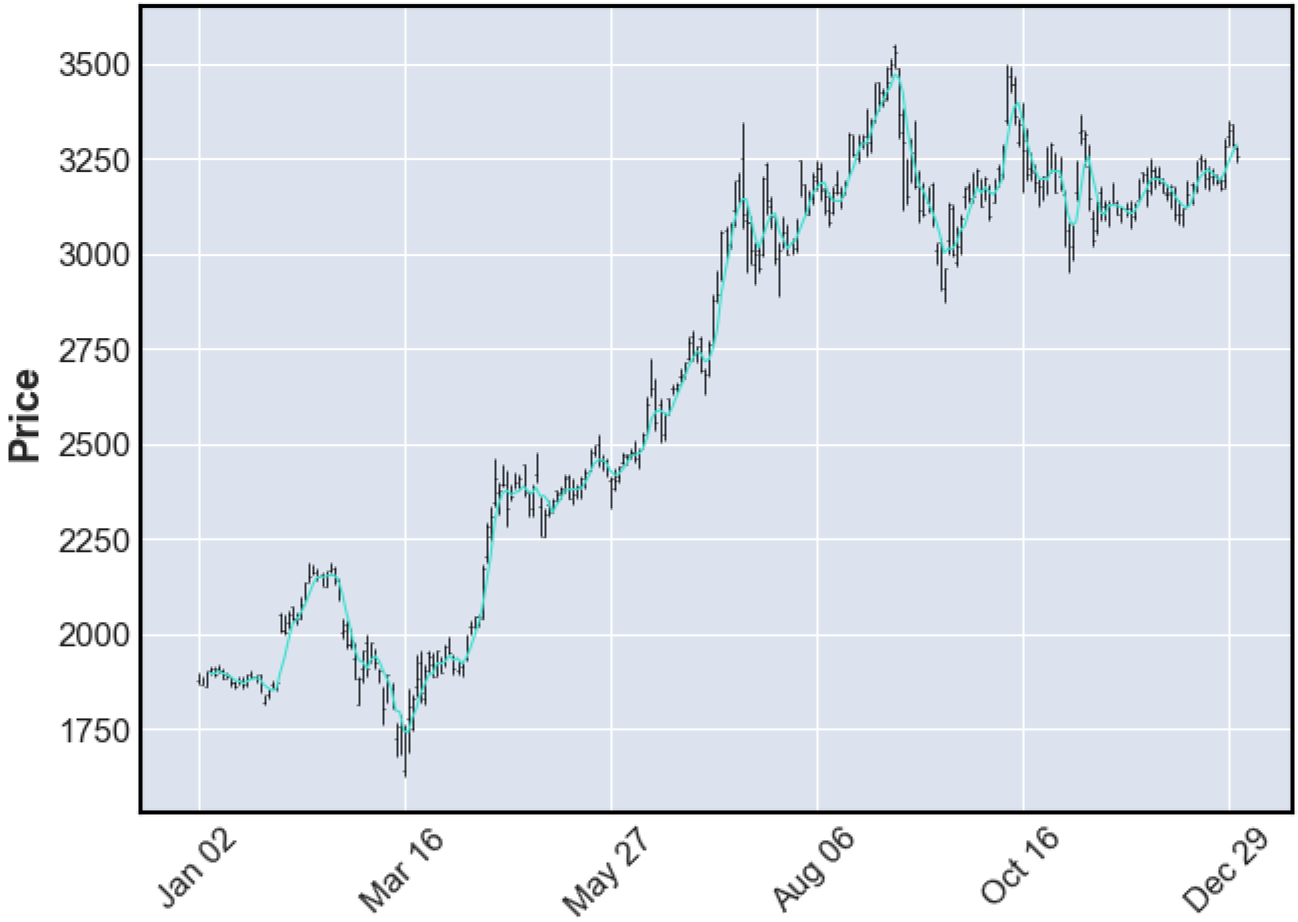
```
In [19]: get_return_defined_time(AMZN, 2020, 1,1,2021, 1,1)
```

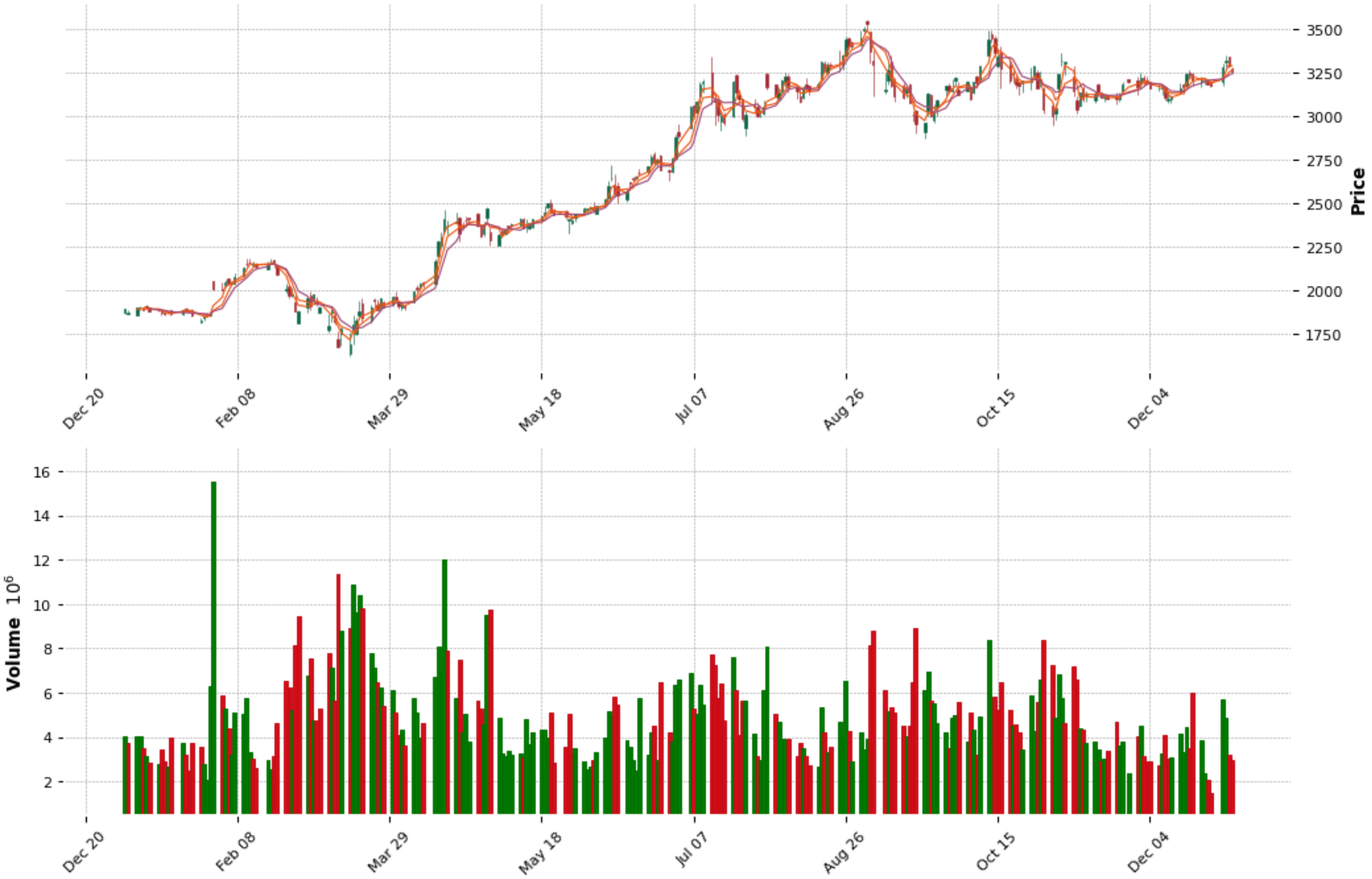
```
Out[19]: 0.6413534880927197
```

```
In [20]: mplfinance_plot('AMZN', 'candle', 2020, 1,1, 2021, 1,1)
```




Price
3500
3250
3000
2750
2500
2250
2000
1750
Jan 02
Mar 16
May 27
Aug 06
Oct 16
Dec 29


Price
3500
3250
3000
2750
2500
2250
2000
1750
Jan 02
Mar 16
May 27
Aug 06
Oct 16
Dec 29


3500
3250
3000
2750
2500
2250
2000
1750
Price
Dec 20
Feb 08
Mar 29
May 18
Jul 07
Aug 26
Oct 15
Dec 04
16
14
12
10
8
6
4
2
Volume 10^6
Dec 20
Feb 08
Mar 29
May 18
Jul 07
Aug 26
Oct 15
Dec 04