Download historical price quotes

We aim to download AMZN stock on Yahoo Finance.

Method 1

- Get to Yahoo Finance
- Search for Amazon
- Click on `Historical Data`
- Specify the time period as `01/01/2000` - `12/31/2021`
- Click on `Apply` , and download the file `AMZN.csv` to the working directory

In [1]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# change the working directory
os.chdir('/Users/wanzhang/Google Drive/Econ 432_22W/TA notes/Week 2')

#load data into a DataFrame object
df = pd.read_csv('AMZN.csv')
df
```

Out[1]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2000-01-03	81.500000	89.562500	79.046875	89.375000	89.375000	16117600
1	2000-01-04	85.375000	91.500000	81.750000	81.937500	81.937500	17487400
2	2000-01-05	70.500000	75.125000	68.000000	69.750000	69.750000	38457400
3	2000-01-06	71.312500	72.687500	64.000000	65.562500	65.562500	18752000
4	2000-01-07	67.000000	70.500000	66.187500	69.562500	69.562500	10505400
...	...	...	...	...	...	...	...
5530	2021-12-23	3408.560059	3439.500000	3403.000000	3421.370117	3421.370117	1839400
5531	2021-12-27	3420.739990	3458.860107	3384.310059	3393.389893	3393.389893	2934400
5532	2021-12-28	3403.649902	3443.520020	3382.709961	3413.219971	3413.219971	2731900
5533	2021-12-29	3416.800049	3424.239990	3372.010010	3384.020020	3384.020020	1787700
5534	2021-	3394.000000	3417.760010	3370.479980	3372.889893	3372.889893	1879200

Date	Open	High	Low	Close	Adj Close	Volume
12-30						

5535 rows × 7 columns

Method 2

Also, we can directly import the data into memory.

In [2]:
```
# install new package
# pip install pandas_datareader

import pandas_datareader as web
df = web.get_data_yahoo("AMZN", start = "2000-01-01", end = "2021-12-31", interv
# 'd': daily, 'wk': weekly, 'mo': monthly
df
```

Out[2]:

Date	High	Low	Open	Close	Volume	Adj Close
2000-01-03	89.562500	79.046875	81.500000	89.375000	16117600	89.375000
2000-01-04	91.500000	81.750000	85.375000	81.937500	17487400	81.937500
2000-01-05	75.125000	68.000000	70.500000	69.750000	38457400	69.750000
2000-01-06	72.687500	64.000000	71.312500	65.562500	18752000	65.562500
2000-01-07	70.500000	66.187500	67.000000	69.562500	10505400	69.562500
...	...	...	...	...	...	...
2021-12-27	3458.860107	3384.310059	3420.739990	3393.389893	2934400	3393.389893
2021-12-28	3443.520020	3382.709961	3403.649902	3413.219971	2731900	3413.219971
2021-12-29	3424.239990	3372.010010	3416.800049	3384.020020	1787700	3384.020020
2021-12-30	3417.760010	3370.479980	3394.000000	3372.889893	1879200	3372.889893
2021-12-31	3387.000000	3331.169922	3379.120117	3334.340088	2387300	3334.340088

5536 rows × 6 columns

Note that `2000-01-03` is an index instead of an observation of `Date`. It can be seen below that the dataset `df` does not have a column called `Date`.

In [3]:
```
# report the columns
df.columns
```

Out[3]:
```
Index(['High', 'Low', 'Open', 'Close', 'Volume', 'Adj Close'], dtype='object')
```

A cure for that is to reset the index.

In [4]:
```
df.reset_index(inplace = True)
# When we reset the index, the old index is added as a column, and a new sequent
```

```
df
```

Out[4]:

	Date	High	Low	Open	Close	Volume	Adj Close
0	2000-01-03	89.562500	79.046875	81.500000	89.375000	16117600	89.375000
1	2000-01-04	91.500000	81.750000	85.375000	81.937500	17487400	81.937500
2	2000-01-05	75.125000	68.000000	70.500000	69.750000	38457400	69.750000
3	2000-01-06	72.687500	64.000000	71.312500	65.562500	18752000	65.562500
4	2000-01-07	70.500000	66.187500	67.000000	69.562500	10505400	69.562500
...	...	...	...	...	...	...	...
5531	2021-12-27	3458.860107	3384.310059	3420.739990	3393.389893	2934400	3393.389893
5532	2021-12-28	3443.520020	3382.709961	3403.649902	3413.219971	2731900	3413.219971
5533	2021-12-29	3424.239990	3372.010010	3416.800049	3384.020020	1787700	3384.020020
5534	2021-12-30	3417.760010	3370.479980	3394.000000	3372.889893	1879200	3372.889893
5535	2021-12-31	3387.000000	3331.169922	3379.120117	3334.340088	2387300	3334.340088

5536 rows × 7 columns

In [5]:
```
df.columns
```

Out[5]:
```
Index(['Date', 'High', 'Low', 'Open', 'Close', 'Volume', 'Adj Close'], dtype='ob
ject')
```

Basic Data Analysis

In [6]:
```
# view the first 5 rows
df.head()
```

Out[6]:

	Date	High	Low	Open	Close	Volume	Adj Close
0	2000-01-03	89.5625	79.046875	81.5000	89.3750	16117600	89.3750
1	2000-01-04	91.5000	81.750000	85.3750	81.9375	17487400	81.9375
2	2000-01-05	75.1250	68.000000	70.5000	69.7500	38457400	69.7500
3	2000-01-06	72.6875	64.000000	71.3125	65.5625	18752000	65.5625
4	2000-01-07	70.5000	66.187500	67.0000	69.5625	10505400	69.5625

In [7]:
```
# view the first 10 rows
df.head(10)
```

Out[7]:

	Date	High	Low	Open	Close	Volume	Adj Close
0	2000-01-03	89.5625	79.046875	81.5000	89.3750	16117600	89.3750
1	2000-01-04	91.5000	81.750000	85.3750	81.9375	17487400	81.9375
2	2000-01-05	75.1250	68.000000	70.5000	69.7500	38457400	69.7500
3	2000-01-06	72.6875	64.000000	71.3125	65.5625	18752000	65.5625
4	2000-01-07	70.5000	66.187500	67.0000	69.5625	10505400	69.5625
5	2000-01-10	72.6250	65.562500	72.5625	69.1875	14757900	69.1875
6	2000-01-11	70.0000	65.000000	66.8750	66.7500	10532700	66.7500
7	2000-01-12	68.0000	63.000000	67.8750	63.5625	10804500	63.5625
8	2000-01-13	67.1875	63.125000	64.9375	65.9375	10448100	65.9375
9	2000-01-14	68.6250	64.000000	66.7500	64.2500	6853600	64.2500

In [8]:
```
# view the last 5 rows
df.tail(10)
```

Out[8]:

	Date	High	Low	Open	Close	Volume	Adj Close
5526	2021-12-17	3417.969971	3312.270020	3354.209961	3400.350098	4277100	3400.350098
5527	2021-12-20	3357.489990	3312.000000	3337.000000	3341.580078	2868600	3341.580078
5528	2021-12-21	3414.330078	3312.949951	3357.010010	3408.340088	2797800	3408.340088
5529	2021-12-22	3441.000000	3370.010010	3385.399902	3420.739990	2751800	3420.739990
5530	2021-12-23	3439.500000	3403.000000	3408.560059	3421.370117	1839400	3421.370117
5531	2021-12-27	3458.860107	3384.310059	3420.739990	3393.389893	2934400	3393.389893
5532	2021-12-28	3443.520020	3382.709961	3403.649902	3413.219971	2731900	3413.219971
5533	2021-12-29	3424.239990	3372.010010	3416.800049	3384.020020	1787700	3384.020020
5534	2021-12-30	3417.760010	3370.479980	3394.000000	3372.889893	1879200	3372.889893
5535	2021-12-31	3387.000000	3331.169922	3379.120117	3334.340088	2387300	3334.340088

In [9]:
```
# summary statistics
df.describe()
```

Out[9]:

	High	Low	Open	Close	Volume	Adj Close
count	5536.000000	5536.000000	5536.000000	5536.000000	5.536000e+03	5536.000000
mean	609.489256	595.819304	602.978040	602.828734	6.332314e+06	602.828734
std	932.705289	912.320767	923.080063	922.511388	5.041159e+06	922.511388
min	6.100000	5.510000	5.910000	5.970000	8.813000e+05	5.970000
25%	43.490002	42.110000	42.628751	42.732501	3.493100e+06	42.732501
50%	175.184998	171.739998	173.514999	173.785004	5.226950e+06	173.785004
75%	733.067505	721.774979	727.527496	728.134979	7.542425e+06	728.134979
max	3773.080078	3696.790039	3744.000000	3731.409912	1.043292e+08	3731.409912

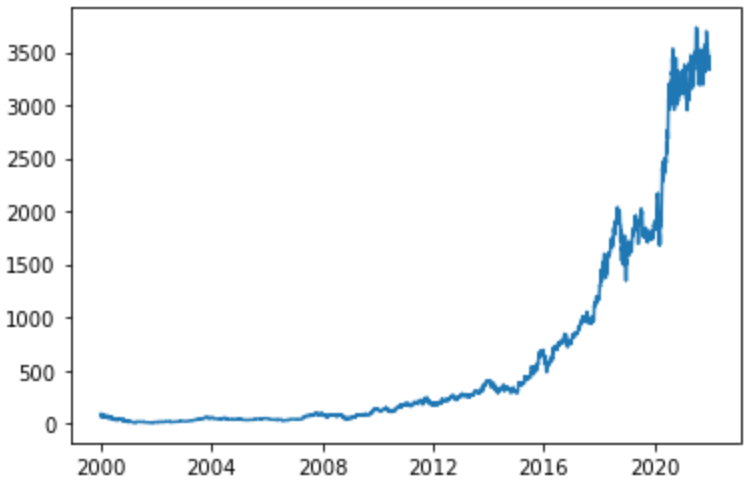
In [10]:
```
df['Adj Close'].describe()
```

Out[10]:
```
count    5536.000000
mean      602.828734
std       922.511388
min         5.970000
25%        42.732501
50%       173.785004
75%       728.134979
max      3731.409912
Name: Adj Close, dtype: float64
```

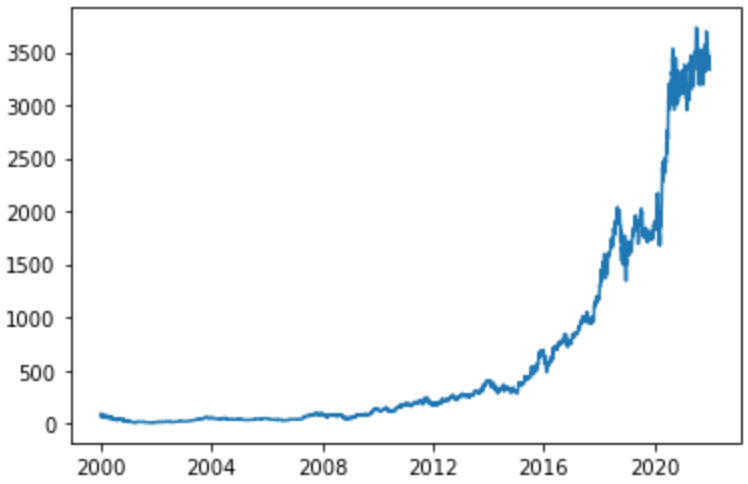
Plot the time series

We will next plot the adjusted prices against the dates.

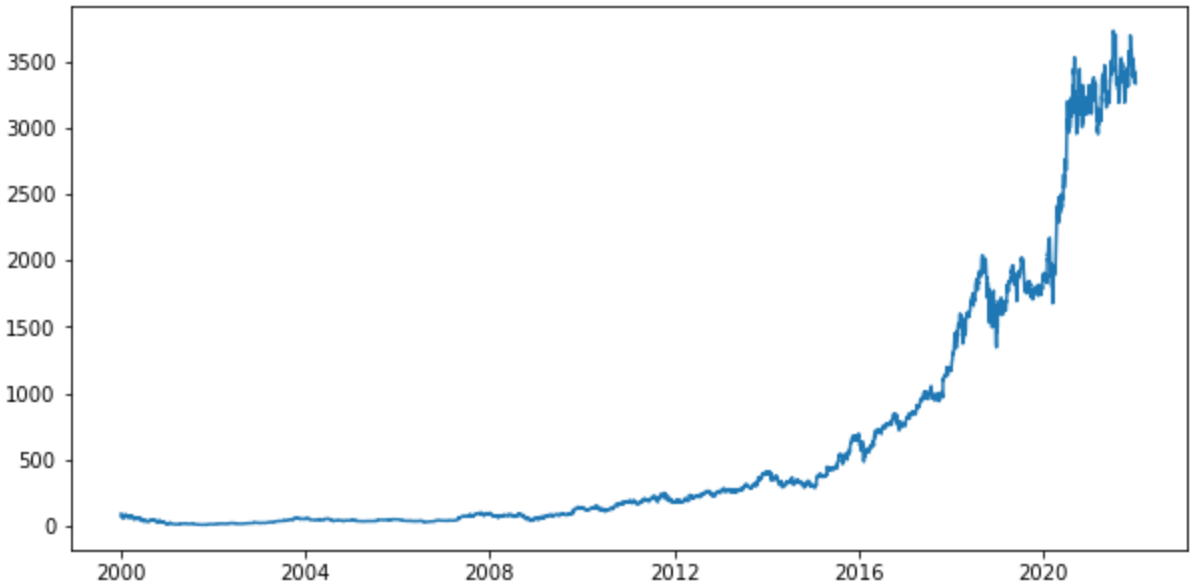
In [11]:
```
df['Date'] = pd.to_datetime(df['Date']) # convert the strings to dates

plt.plot('Date', 'Adj Close', data = df)
plt.show()
```



In [12]:
```
plt.plot(df['Date'], df['Adj Close'])
plt.show()
```



In [13]:
```
# 10in by 5in in size
plt.figure(figsize=(10,5))
plt.plot('Date', 'Adj Close', data = df)
plt.show()
```



In [14]:
```
# add title, xlabels, ylabels
plt.figure(figsize=(10,5))
plt.plot('Date', 'Adj Close', data = df)
plt.title('Daily Prices of Amazon Shares (2000 Jan -- 2021 Dec)')
plt.xlabel('Year')
plt.ylabel('Amazon Prices')
plt.savefig('Daily_prices.png') # export the figures
plt.show()
```

Calculating Returns

Simple Returns

Let P_t be the stock price at the end of time t. The simple return over time t is given by

$$R_t = \frac{P_t - P_{t-1}}{P_{t-1}}.$$

We can thus compute the simple returns on Amazon shares.

In [15]:
```
# by shift(1), we get the row just above the present row
df['Simple return'] = df['Adj Close'] / df['Adj Close'].shift(1) - 1
df
```

Out[15]:

	Date	High	Low	Open	Close	Volume	Adj Close	
0	2000-01-03	89.562500	79.046875	81.500000	89.375000	16117600	89.375000	
1	2000-01-04	91.500000	81.750000	85.375000	81.937500	17487400	81.937500	-0
2	2000-01-05	75.125000	68.000000	70.500000	69.750000	38457400	69.750000	-(
3	2000-01-06	72.687500	64.000000	71.312500	65.562500	18752000	65.562500	-0.
4	2000-01-07	70.500000	66.187500	67.000000	69.562500	10505400	69.562500	(
...	...	...	...	...	...	...	...	

	Date	High	Low	Open	Close	Volume	Adj Close	
5531	2021-12-27	3458.860107	3384.310059	3420.739990	3393.389893	2934400	3393.389893	-0
5532	2021-12-28	3443.520020	3382.709961	3403.649902	3413.219971	2731900	3413.219971	0.
5533	2021-12-29	3424.239990	3372.010010	3416.800049	3384.020020	1787700	3384.020020	-0.
5534	2021-12-30	3417.760010	3370.479980	3394.000000	3372.889893	1879200	3372.889893	-0.
5535	2021-12-31	3387.000000	3331.169922	3379.120117	3334.340088	2387300	3334.340088	-0

5536 rows × 8 columns

In [16]:
```
df['Adj Close'].pct_change()
# df['Simple return'] = df['Adj Close'].pct_change()
df
```

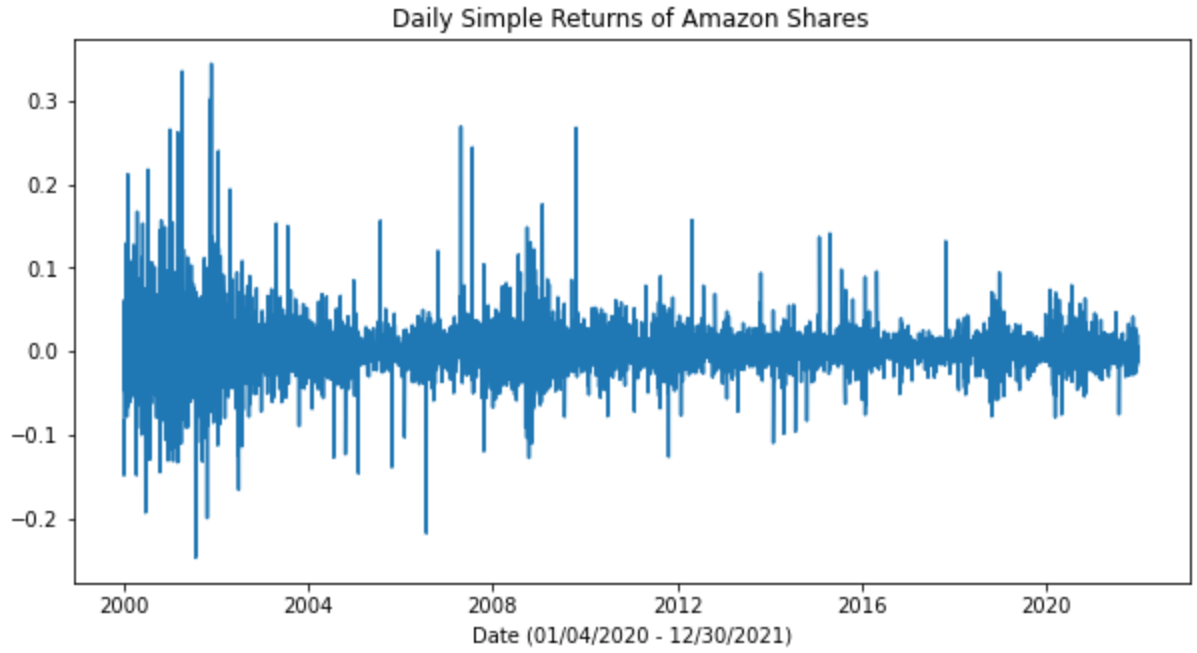
Out[16]:

	Date	High	Low	Open	Close	Volume	Adj Close	
0	2000-01-03	89.562500	79.046875	81.500000	89.375000	16117600	89.375000	
1	2000-01-04	91.500000	81.750000	85.375000	81.937500	17487400	81.937500	-0
2	2000-01-05	75.125000	68.000000	70.500000	69.750000	38457400	69.750000	-0
3	2000-01-06	72.687500	64.000000	71.312500	65.562500	18752000	65.562500	-0.
4	2000-01-07	70.500000	66.187500	67.000000	69.562500	10505400	69.562500	0
...	...	...	...	...	...	...	...	
5531	2021-12-27	3458.860107	3384.310059	3420.739990	3393.389893	2934400	3393.389893	-0
5532	2021-12-28	3443.520020	3382.709961	3403.649902	3413.219971	2731900	3413.219971	0.
5533	2021-12-29	3424.239990	3372.010010	3416.800049	3384.020020	1787700	3384.020020	-0.
5534	2021-12-30	3417.760010	3370.479980	3394.000000	3372.889893	1879200	3372.889893	-0.
5535	2021-12-31	3387.000000	3331.169922	3379.120117	3334.340088	2387300	3334.340088	-0

5536 rows × 8 columns

We can plot the simple returns.

In [17]:
```
plt.figure(figsize = (10, 5))
plt.plot('Date', 'Simple return', data = df[1:])
plt.title('Daily Simple Returns of Amazon Shares')
plt.xlabel('Date (01/04/2020 - 12/30/2021)')
plt.show()
```



Continuously Compounded Returns

Given the continuously compounded return r_t, when considering multiple conpounding, we have

$$\frac{P_t}{P_{t-1}} = \left(1 + \frac{r_t}{n}\right)^n.$$

When $n \to \infty$,

$$\lim_{n\to\infty} \left(1 + \frac{r_t}{n}\right)^n = \left[\lim_{n\to\infty} \left(1 + \frac{r_t}{n}\right)^{n/r_t}\right]^{r_t} = e^{r_t}.$$

Thus,

$$r_t = \ln\left(\frac{P_t}{P_{t-1}}\right) = \ln(1 + R_t).$$

We compute the cc returns on Amazon shares as follows.

$$e^x - 1 \approx x$$

$$e^x \approx 1 + x$$

$$x \approx \log(1 + x)$$

In [18]:
```
df['CC return'] = np.log(df['Adj Close']/df['Adj Close'].shift(1))
df
```

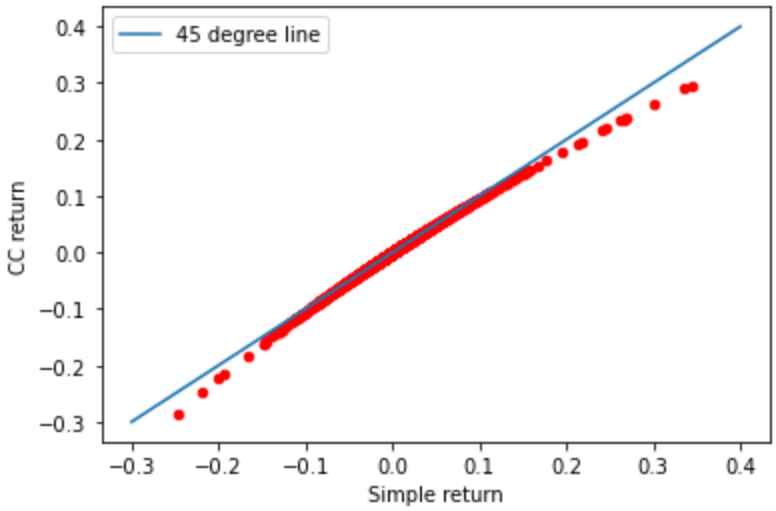
Out[18]:

	Date	High	Low	Open	Close	Volume	Adj Close	
0	2000-01-03	89.562500	79.046875	81.500000	89.375000	16117600	89.375000	
1	2000-01-04	91.500000	81.750000	85.375000	81.937500	17487400	81.937500	-0
2	2000-01-05	75.125000	68.000000	70.500000	69.750000	38457400	69.750000	-(
3	2000-01-06	72.687500	64.000000	71.312500	65.562500	18752000	65.562500	-0.
4	2000-01-07	70.500000	66.187500	67.000000	69.562500	10505400	69.562500	(
...	...	...	...	...	...	...	...	
5531	2021-12-27	3458.860107	3384.310059	3420.739990	3393.389893	2934400	3393.389893	-0
5532	2021-12-28	3443.520020	3382.709961	3403.649902	3413.219971	2731900	3413.219971	0.
5533	2021-12-29	3424.239990	3372.010010	3416.800049	3384.020020	1787700	3384.020020	-0.
5534	2021-12-30	3417.760010	3370.479980	3394.000000	3372.889893	1879200	3372.889893	-0.
5535	2021-12-31	3387.000000	3331.169922	3379.120117	3334.340088	2387300	3334.340088	-(

5536 rows × 9 columns

As we mentioned last time, when x is small, $\ln(1+x) \approx x$. It implies that simple returns and cc returns differ little. This can be verified by the following graph.

In [19]:
```
# a scatter plot comparing num_children and num_pets
df.plot(kind='scatter',x='Simple return',y='CC return',color='red')
plt.plot(np.linspace(-0.3, 0.4, 101), np.linspace(-0.3, 0.4, 101), label = '45 d
plt.legend()
plt.show()
```



We can further plot the two kinds of returns together.

In [20]:
```
plt.figure(figsize = (10, 5))
plt.plot('Date', 'Simple return', data = df[1:], label = 'Simple return')
plt.plot('Date', 'CC return', data = df[1:], label = 'CC return')
plt.title('Daily Returns of Amazon Shares')
plt.xlabel('Date (01/04/2020 - 12/30/2021)')
plt.legend()
plt.show()
```