School of Computer Engineering and Science SHU
Blockchain Technology and Data Economics
(08696017 1000)

Practical Report

Practice 3
Manipulating and visualizing bitcoin price data

Name: Lin Yijun ID: 18120189

Date: 22 Apr. 2020

Contents

1 Introduction

In this practise, we will introduce the following topics:

- Getting set up for data analysist
- Getting, reading in, and cleaning bitcoin price data
- Exploring, manipulating, and visualizing the cleaned-up data

We first need to install several Python libraries, which includes installing the pandas module for reading in data, and also doing some exploratory analysis. We'll also be installing ***matplotlib*** *for creating plots and charts, as well as Jupyter Notebooks, as they are the best for this kind of work involving data analysis.*

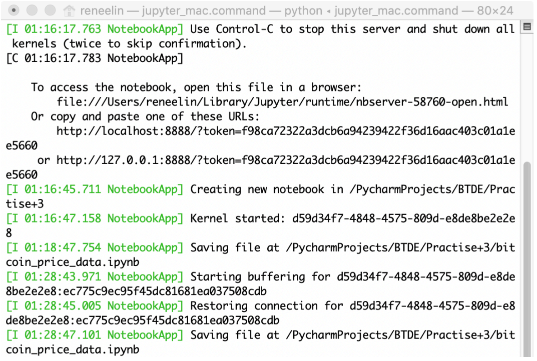
2 Materials, Methods and Results

2.1 Getting set up for data analysis

To install the Python modules, open the command-line program. In the command line, to install pandas, matplotlib, jupyter, execute the following command:

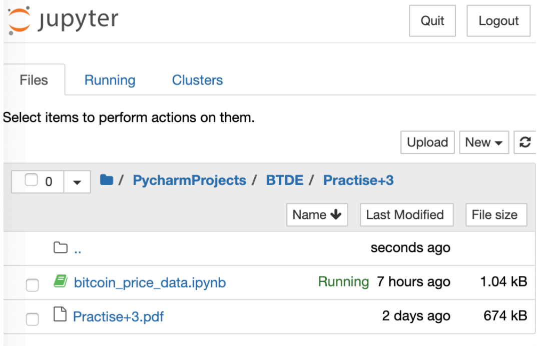
```
pip3 install pandas
pip3 install matplotlib
pip3 install jupyter
```

Having finished installing the required modules, launch the Jupyter Notebook by executing the jupyter notebook command. This will open up a new browser window, or a tab, where it will display the list of files that are already there from the folder where we executed the jupyter notebook command. The following screenshot shows the jupyter notebook command:


reneelin — jupyter_mac.command — python ◂ jupyter_mac.command — 80×24

```
[I 01:16:17.763 NotebookApp] Use Control-C to stop this server and shut down all
 kernels (twice to skip confirmation).
[C 01:16:17.783 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///Users/reneelin/Library/Jupyter/runtime/nbserver-58760-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=f98ca72322a3dcb6a94239422f36d16aac403c01a1e
e5660
     or http://127.0.0.1:8888/?token=f98ca72322a3dcb6a94239422f36d16aac403c01a1e
e5660
[I 01:16:45.711 NotebookApp] Creating new notebook in /PycharmProjects/BTDE/Prac
tise+3
[I 01:16:47.158 NotebookApp] Kernel started: d59d34f7-4848-4575-809d-e8de8be2e2e
8
[I 01:18:47.754 NotebookApp] Saving file at /PycharmProjects/BTDE/Practise+3/bit
coin_price_data.ipynb
[I 01:28:43.971 NotebookApp] Starting buffering for d59d34f7-4848-4575-809d-e8de
8be2e2e8:ec775c9ec95f45dc81681ea037508cdb
[I 01:28:45.005 NotebookApp] Restoring connection for d59d34f7-4848-4575-809d-e8
de8be2e2e8:ec775c9ec95f45dc81681ea037508cdb
[I 01:28:47.101 NotebookApp] Saving file at /PycharmProjects/BTDE/Practise+3/bit
coin_price_data.ipynb
```

Next, choose to create a new Python 3 notebook, as shown in the following screenshot:



Select items to perform actions on them.

/ PycharmProjects / BTDE / Practise+3

Name		Last Modified	File size
..		seconds ago	
bitcoin_price_data.ipynb	Running	7 hours ago	1.04 kB
Practise+3.pdf		2 days ago	674 kB

2.2 Getting, reading in, and cleaning bitcoin price data

We will start by importing the necessary modules. Import pandas to enable you to read in the data and start exploring it. The following screenshot shows the *import pandas* command:

Also, import *matplotlib* for drawing plots from the data.

We need to set some options for *pandas* and *matplotlib*. The following screenshot shows the command for importing *matplotlib*:

The first option we will set is called

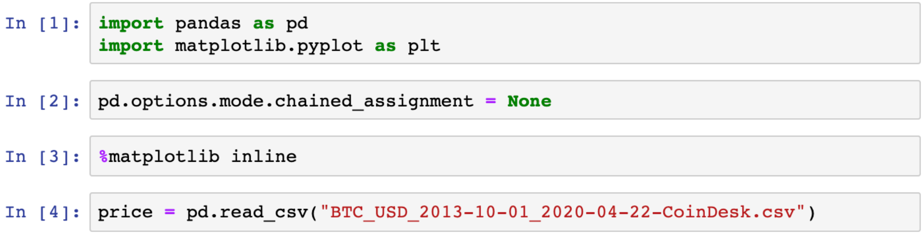
```
options.mode.chained_assignment = None.
```

The preceding option is to make sure that the operations are for the cleanup, which will be performed on the pandas DataFrame objects; we want the cleanup to happen on the original DataFrame objects and not on copies.

Also, set matplotlib to visualize and display all the charts.

Download the data in CSV format and read this data using pandas. This is a CSV file, so we will use the read_CSV method from pandas.

The following screenshot shows *Getting, reading in, and cleaning bitcoin price data*'s Jupyter Notebook Platform.

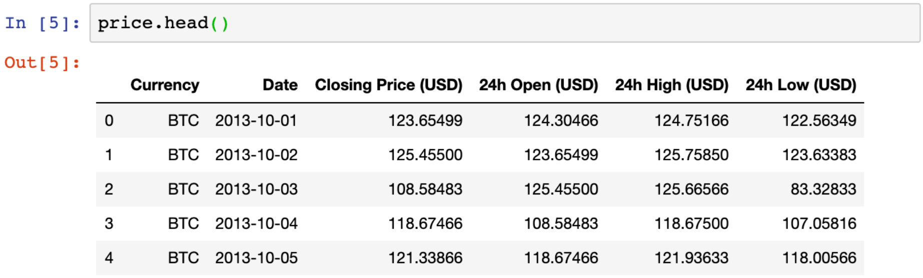


```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt

In [2]: pd.options.mode.chained_assignment = None

In [3]: %matplotlib inline

In [4]: price = pd.read_csv("BTC_USD_2013-10-01_2020-04-22-CoinDesk.csv")
```

2.3 Data Frame

The data in a pandas data object is called a DataFrame. A DataFrame is in a tabular data format. Now, print out some records to see how this looks. To print this out, we can call a method called *head()* on the price DataFrame.

When we do this, we get six columns—Currency, Date, Close Price, 24h Open, 24h High and 24h Low—for the bitcoin in USD for that day. We also have a default index for the rows starting from 0, which was inserted by pandas by default while reading in the data. The following screenshot shows the six columns, Currency, Date, Close Price, 24h Open, 24h High and 24h Low:

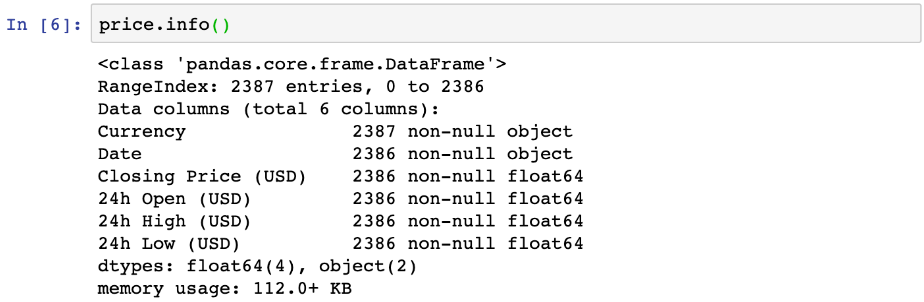


```
In [5]: price.head()

Out[5]:
```

	Currency	Date	Closing Price (USD)	24h Open (USD)	24h High (USD)	24h Low (USD)
0	BTC	2013-10-01	123.65499	124.30466	124.75166	122.56349
1	BTC	2013-10-02	125.45500	123.65499	125.75850	123.63383
2	BTC	2013-10-03	108.58483	125.45500	125.66566	83.32833
3	BTC	2013-10-04	118.67466	108.58483	118.67500	107.05816
4	BTC	2013-10-05	121.33866	118.67466	121.93633	118.00566

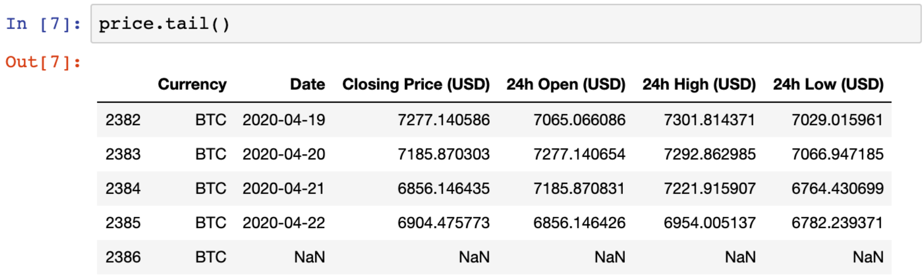
To get top-level information about this data, call the *info()* method on it. After calling this method, we get 2,387 records. There are two columns: Date and Close Price. Date has 2,386 non-null records of the type object, which means that the Date field has been read as text. We would have to change it to a proper date-time format later. We have the close price, 24h open, 24h high and 24h low all as numeric float type. They all have 2,386 non-null records, which are one records fewer than the whole field.

The following screenshot shows the details of the *info()* method:



```
In [6]: price.info()

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 2387 entries, 0 to 2386
        Data columns (total 6 columns):
        Currency                2387 non-null object
        Date                    2386 non-null object
        Closing Price (USD)     2386 non-null float64
        24h Open (USD)          2386 non-null float64
        24h High (USD)          2386 non-null float64
        24h Low (USD)           2386 non-null float64
        dtypes: float64(4), object(2)
        memory usage: 112.0+ KB
```

In order to print for the records from bottom, call the *tail()* method. This method shows that the last one records should not exist, as they are not available information. We need to remove this before proceeding with further analysis.



```
In [7]: price.tail()
```

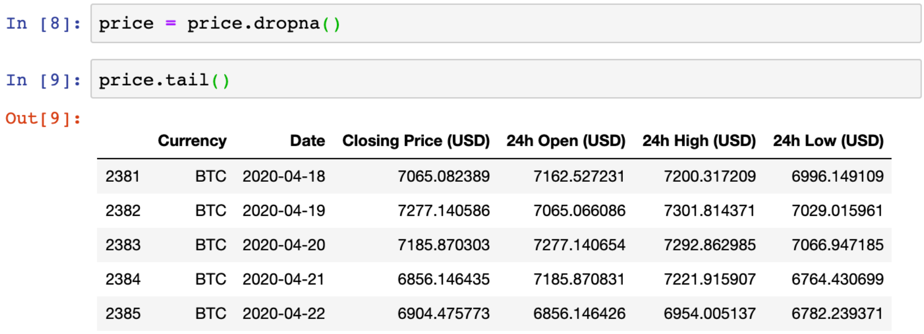
Out[7]:

	Currency	Date	Closing Price (USD)	24h Open (USD)	24h High (USD)	24h Low (USD)
2382	BTC	2020-04-19	7277.140586	7065.066086	7301.814371	7029.015961
2383	BTC	2020-04-20	7185.870303	7277.140654	7292.862985	7066.947185
2384	BTC	2020-04-21	6856.146435	7185.870831	7221.915907	6764.430699
2385	BTC	2020-04-22	6904.475773	6856.146426	6954.005137	6782.239371
2386	BTC	NaN	NaN	NaN	NaN	NaN

We can see that the last line has NaN values, which means that it has missing values. We can use this factor to remove this record from the DataFrame. We call drop any method on the price, which will remove the records that have one or more of the columns as null or missing values.

Bear in mind that we are just removing it from the DataFrame price and not from the CSV file from which we read the data.

Also, call the *tail()* method and look at the bottom rows again to see if the records we wanted to remove have been removed. We can see in the following screenshot that they have, in fact, been removed:



```
In [8]: price = price.dropna()

In [9]: price.tail()
```

Out[9]:

	Currency	Date	Closing Price (USD)	24h Open (USD)	24h High (USD)	24h Low (USD)
2381	BTC	2020-04-18	7065.082389	7162.527231	7200.317209	6996.149109
2382	BTC	2020-04-19	7277.140586	7065.066086	7301.814371	7029.015961
2383	BTC	2020-04-20	7185.870303	7277.140654	7292.862985	7066.947185
2384	BTC	2020-04-21	6856.146435	7185.870831	7221.915907	6764.430699
2385	BTC	2020-04-22	6904.475773	6856.146426	6954.005137	6782.239371

2.4 Data cleanup

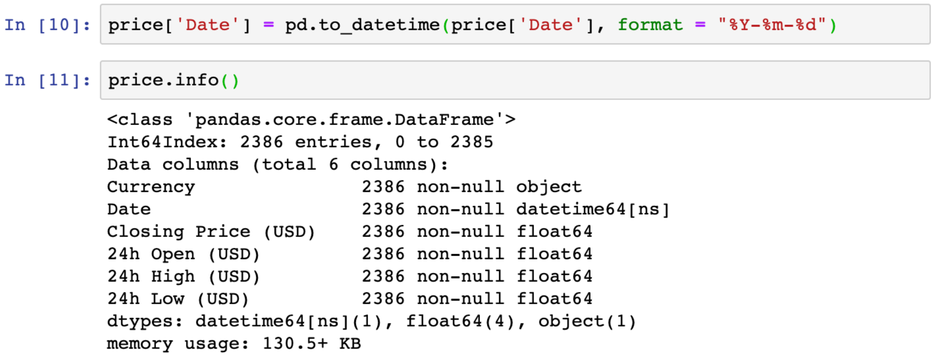
Another data cleaning task we need to do is convert the Date column from an object or text format to a date-time format. We use the pandas to_datetime method to do this.

Here, we ask the to_datetime method to convert the Date field or priceDataFrame, and we also supply the format. We then assign the Date field back to the DataFrame:

```
price['Date'] = pd.to_datetime(price['Date'], format = "%Y−%m−%d")
```

This is the reason that we set the chained assignment as equal to *null* earlier, because we wanted to make the changes back on the original DataFrame.

Call the *info()* method again to see whether the data cleanup has an impact. We can see that the Date field is now in a date-time format, as we wanted, and there are no non-null records in the data, as shown in the following screenshot:



```
In [10]: price['Date'] = pd.to_datetime(price['Date'], format = "%Y-%m-%d")

In [11]: price.info()

         <class 'pandas.core.frame.DataFrame'>
         Int64Index: 2386 entries, 0 to 2385
         Data columns (total 6 columns):
         Currency               2386 non-null object
         Date                   2386 non-null datetime64[ns]
         Closing Price (USD)    2386 non-null float64
         24h Open (USD)         2386 non-null float64
         24h High (USD)         2386 non-null float64
         24h Low (USD)          2386 non-null float64
         dtypes: datetime64[ns](1), float64(4), object(1)
         memory usage: 130.5+ KB
```

2.5 Setting the index to the Date column

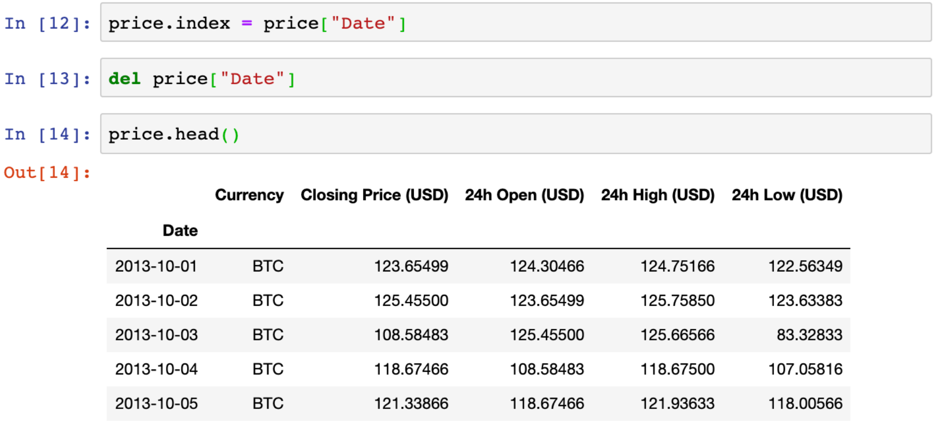
We also need to set the index to the Date column and remove the Datecolumn as a separate column. This will help us to run some interesting queries on the date data.

Use the following code to set the index to the Date column:

```
price.index = price["Date"]
```

Next, delete the Date column as a separate column, since it is already set up as an index.

Now, the Date column can be seen as an index and not a separate column anymore, as shown in the following screenshot:



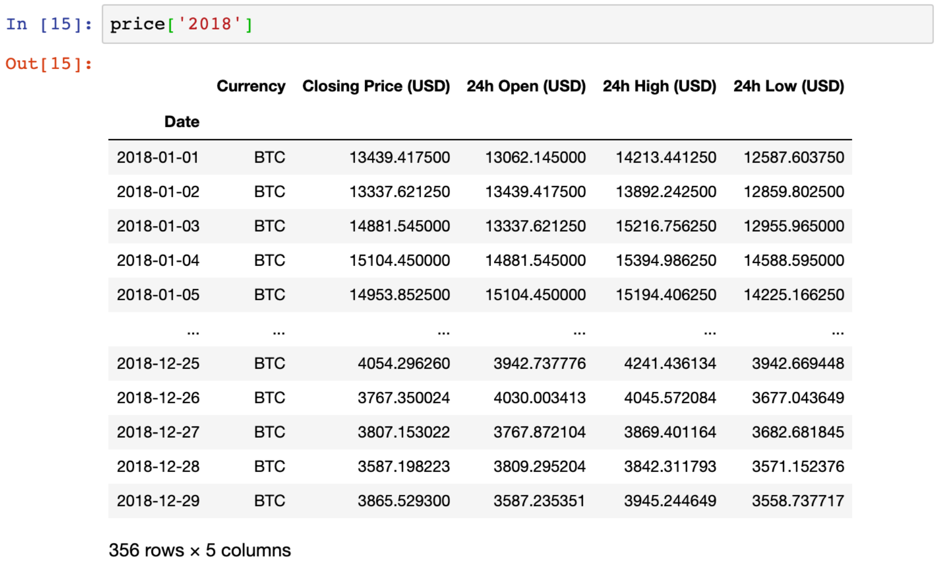
```
In [12]: price.index = price["Date"]

In [13]: del price["Date"]

In [14]: price.head()

Out[14]:
```

	Currency	Closing Price (USD)	24h Open (USD)	24h High (USD)	24h Low (USD)
Date					
2013-10-01	BTC	123.65499	124.30466	124.75166	122.56349
2013-10-02	BTC	125.45500	123.65499	125.75850	123.63383
2013-10-03	BTC	108.58483	125.45500	125.66566	83.32833
2013-10-04	BTC	118.67466	108.58483	118.67500	107.05816
2013-10-05	BTC	121.33866	118.67466	121.93633	118.00566

2.6 Exploring, manipulating, and visualizing the cleaned-up data

As the data cleanup is done, start with the data exploration tasks. We can use the pandas date-time capabilities to run some interesting queries.

For example, if we want to get all the records from a particular year, pass that year to the DataFrame inside square brackets. The following screenshot shows the price data from the year 2018:

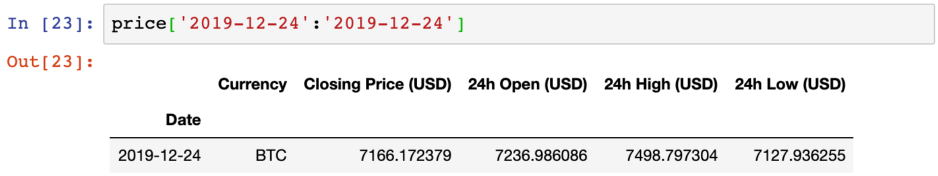


```
In [15]: price['2018']
Out[15]:
```

Date	Currency	Closing Price (USD)	24h Open (USD)	24h High (USD)	24h Low (USD)
2018-01-01	BTC	13439.417500	13062.145000	14213.441250	12587.603750
2018-01-02	BTC	13337.621250	13439.417500	13892.242500	12859.802500
2018-01-03	BTC	14881.545000	13337.621250	15216.756250	12955.965000
2018-01-04	BTC	15104.450000	14881.545000	15394.986250	14588.595000
2018-01-05	BTC	14953.852500	15104.450000	15194.406250	14225.166250
...	...	...	...	...	...
2018-12-25	BTC	4054.296260	3942.737776	4241.436134	3942.669448
2018-12-26	BTC	3767.350024	4030.003413	4045.572084	3677.043649
2018-12-27	BTC	3807.153022	3767.872104	3869.401164	3682.681845
2018-12-28	BTC	3587.198223	3809.295204	3842.311793	3571.152376
2018-12-29	BTC	3865.529300	3587.235351	3945.244649	3558.737717

356 rows × 5 columns

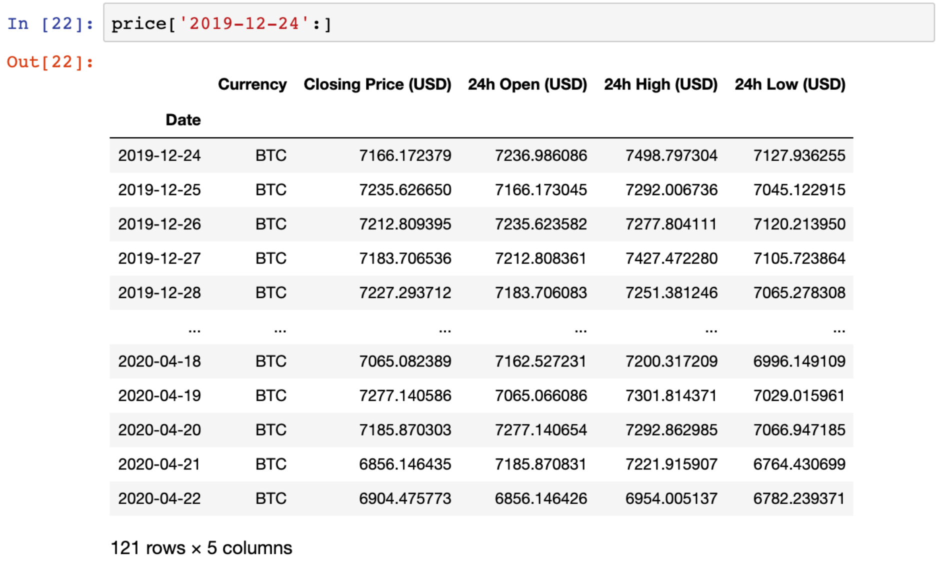
We can also specify whether we want the data from a particular date.
The following screenshot shows the bitcoin price in USD from December 24, 2019:



```
In [23]: price['2019-12-24':'2019-12-24']
Out[23]:
```

Date	Currency	Closing Price (USD)	24h Open (USD)	24h High (USD)	24h Low (USD)
2019-12-24	BTC	7166.172379	7236.986086	7498.797304	7127.936255

We can also specify whether we want the data from a particular period spanning certain dates. The following screenshot shows the data from December 24, 2019, onward:

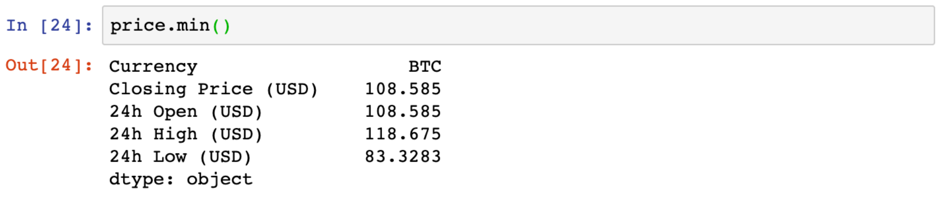


```
In [22]: price['2019-12-24':]
Out[22]:
```

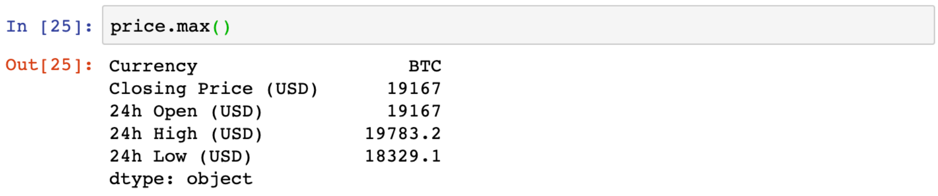
Date	Currency	Closing Price (USD)	24h Open (USD)	24h High (USD)	24h Low (USD)
2019-12-24	BTC	7166.172379	7236.986086	7498.797304	7127.936255
2019-12-25	BTC	7235.626650	7166.173045	7292.006736	7045.122915
2019-12-26	BTC	7212.809395	7235.623582	7277.804111	7120.213950
2019-12-27	BTC	7183.706536	7212.808361	7427.472280	7105.723864
2019-12-28	BTC	7227.293712	7183.706083	7251.381246	7065.278308
...	...	...	...	...	...
2020-04-18	BTC	7065.082389	7162.527231	7200.317209	6996.149109
2020-04-19	BTC	7277.140586	7065.066086	7301.814371	7029.015961
2020-04-20	BTC	7185.870303	7277.140654	7292.862985	7066.947185
2020-04-21	BTC	6856.146435	7185.870831	7221.915907	6764.430699
2020-04-22	BTC	6904.475773	6856.146426	6954.005137	6782.239371

121 rows × 5 columns

Statistical information can also be retrieved using pandas methods. For example, to get the minimum price from this dataset, we can use the *min()* method, and the same, to get the maximum price, use the *max()* method. The result is shown in the following screenshot:

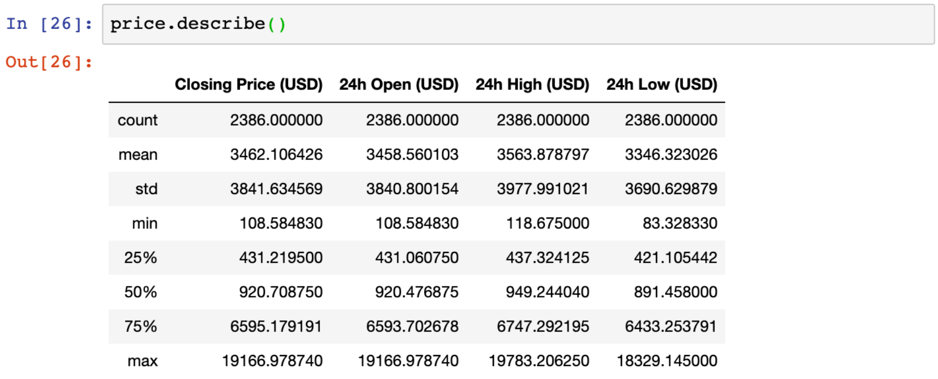


```
In [24]: price.min()

Out[24]: Currency                     BTC
         Closing Price (USD)    108.585
         24h Open (USD)         108.585
         24h High (USD)         118.675
         24h Low (USD)          83.3283
         dtype: object
```



```
In [25]: price.max()

Out[25]: Currency                     BTC
         Closing Price (USD)      19167
         24h Open (USD)           19167
         24h High (USD)         19783.2
         24h Low (USD)          18329.1
         dtype: object
```

A whole bunch of statistical information can be received in one go using the *describe()* method, as shown in the following screenshot:



```
In [26]: price.describe()
```

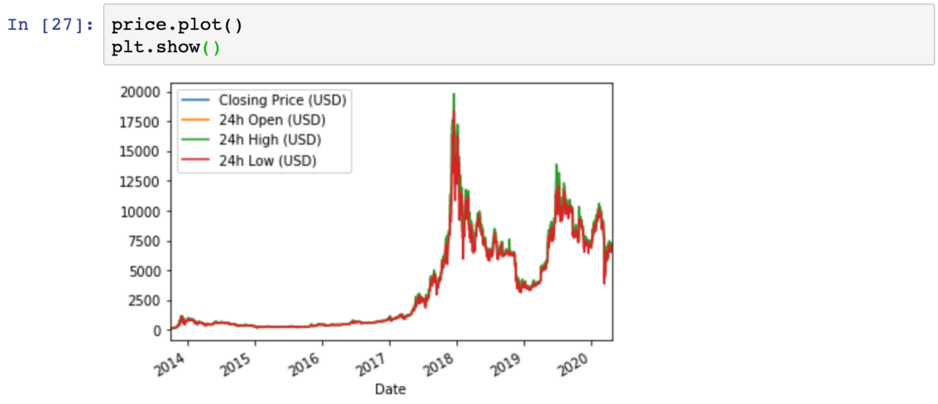
Out[26]:

	Closing Price (USD)	24h Open (USD)	24h High (USD)	24h Low (USD)
count	2386.000000	2386.000000	2386.000000	2386.000000
mean	3462.106426	3458.560103	3563.878797	3346.323026
std	3841.634569	3840.800154	3977.991021	3690.629879
min	108.584830	108.584830	118.675000	83.328330
25%	431.219500	431.060750	437.324125	421.105442
50%	920.708750	920.476875	949.244040	891.458000
75%	6595.179191	6593.702678	6747.292195	6433.253791
max	19166.978740	19166.978740	19783.206250	18329.145000

2.7 Data visualization

It is very easy to start creating plots from data using pandas and matplotlib. To plot the entirety of the data, we will call the plot method on the price DataFrame, and we will get a plot where the x axis is the date and the y axis is the price data.

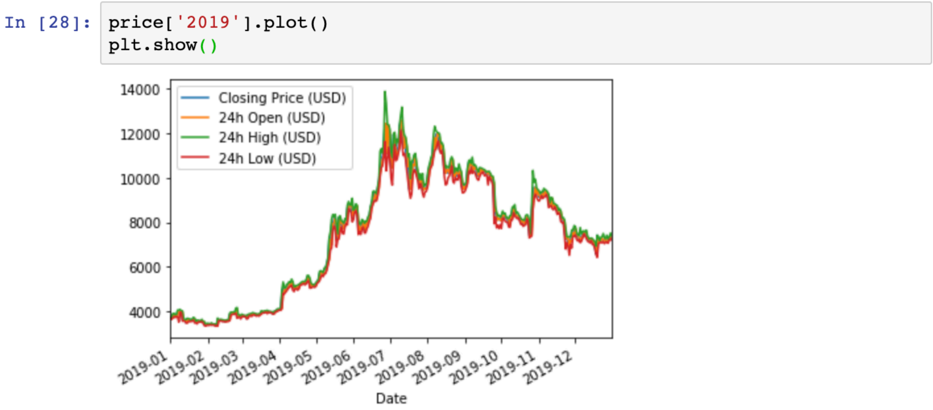
The following screenshot describes the plot, wherein the x axis is the date and the y axis is the price data:



```
In [27]: price.plot()
         plt.show()
```

We can also zoom in on a certain time period. For example, in order to plot the data from 2019 only, first select the data that is from 2019 and then call the *plot()* method on the subset of data.

In the following screenshot, we have a plot for the price data from 2019 onward:



```
In [28]: price['2019'].plot()
         plt.show()
```

3 Conclusion

3.1 Functions Used

DataFrame.head(self: ~FrameOrSeries, n: int = 5) → ~FrameOrSeries This function returns the first n rows for the object based on position. It is useful for quickly testing if your object has the right type of data in it. For negative values of n, this function returns all rows except the last n rows, equivalent to df[:-n].

DataFrame.tail(self: ~FrameOrSeries, n: int = 5) → ~FrameOrSeries This function returns last n rows from the object based on position. It is useful for quickly verifying data, for example, after sorting or appending rows.For negative values of n, this function returns all rows except the first n rows, equivalent to df[n:].

DataFrame.dropna(self, axis=0, how='any', thresh=None, subset=None, inplace=False) Remove missing values.

DataFrame.info(self, verbose=None, buf=None, max_cols=None, memory_usage=None, null_counts=None) → None Print a concise summary of a DataFrame. This method prints information about a DataFrame including the index dtype and column dtypes, non-null values and memory usage.

DataFrame.describe(self: ~FrameOrSeries, percentiles=None, include=None, exclude=None) → ~FrameOrSeries Generate descriptive statistics. Descriptive statistics include those that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values. Analyzes both numeric and object series, as well as DataFrame column sets of mixed data types. The output will vary depending on what is provided. Refer to the notes below for more detail.

pandas.to_datetime(arg, errors='raise', dayfirst=False, yearfirst=False, utc=None, format= None, exact=True, unit=None, infer_datetime_format=False, origin='unix', cache=True) Convert argument to datetime.

3.2 Practise Conclusion

In this practise, I have learnt to install pandas and matplotlib library and use Jupyter Notebook as IDE this time.

Using the functions in the library, I learnt about DataFrame and successfully import the csv file and print it out in Jupyter Notebook. By the way, I learn the methods of cleaning unused information not by modifying the origin csv file.

Then I knew the data structure 'date-time' in pandas. And convert text to date-time for further usage. Also, learning setting index for DataFrame. The capabilities of date-time are strong enough for data exploring and manipulating. However, I found error when using code 'price['2019-12-24']', it only works when it is changed to 'price['2019-12-24':'2019-12-24']'. I tried to figure out the reason but failed. Pandas works with matplotlib becomes a powerful data visualization tool and looks great when operating accurately.

In this practise, I get familiar with pandas library. Successfully complete the practical section this week.

4 References

PDF document: Practical 3 Guide File.
Website: www.coindesk.com/price/
Website: pandas.pydata.org/pandas-docs/stable/index.html
PDF document: How to Write a Practical/Laboratory Report—Learning Guide.