Blockchain Technology and Data Economics

Practice 3 - Manipulating and visualizing bitcoin price data

Version: 2020 Spring

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.*

I. Getting set up for data analysis

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

```
pip install pandas
```

Similarly, to install matplotlib, execute the following command:

```
pip install matplotlib
```

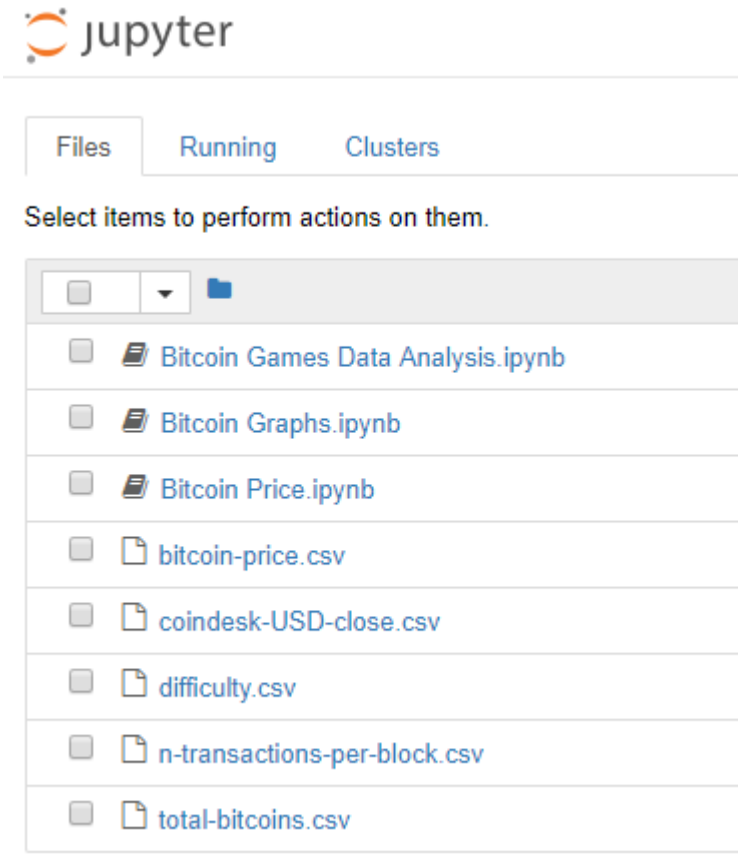
To install Jupyter, execute the following command:

```
pip 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:

```
G:\Draft Folder\Bitcoin\Chapter04>jupyter notebook
[I 06:28:12.977 NotebookApp] JupyterLab alpha preview extension loaded from E:\P
ython\Anaconda\Anaconda3\lib\site-packages\jupyterlab
JupyterLab v0.27.0
Known labextensions:
[I 06:28:12.985 NotebookApp] Running the core application with no additional ext
ensions or settings
[I 06:28:16.911 NotebookApp] Serving notebooks from local directory: G:\Draft Fo
lder\Bitcoin\Chapter04
[I 06:28:16.911 NotebookApp] 0 active kernels
[I 06:28:16.912 NotebookApp] The Jupyter Notebook is running at: http://localhos
t:8888/?token=83fb194d60c2f96fc39106e4ebefd092db993675d3272c60
[I 06:28:16.912 NotebookApp] Use Control-C to stop this server and shut down all
 kernels (twice to skip confirmation).
[C 06:28:16.915 NotebookApp]
```

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



II. 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:

```
In [1]: import pandas
```

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*:

```
In [1]: import pandas

In [2]: import matplotlib.pyplot as plt
```

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.

The following screenshot shows the options.mode.chained_assignment = None option:

```
In [5]: import pandas as pd

In [6]: import matplotlib.pyplot as plt

In [7]: pd.options.mode.chained_assignment = None
```

Also, set *matplotlib* to visualize and display all the charts shown in the following screenshot:

```
In [5]: import pandas as pd

In [6]: import matplotlib.pyplot as plt

In [7]: pd.options.mode.chained_assignment = None

In [8]: %matplotlib inline
```

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, as shown in the following screenshot:

```
In [5]: import pandas as pd

In [6]: import matplotlib.pyplot as plt

In [7]: pd.options.mode.chained_assignment = None

In [8]: %matplotlib inline

In [9]: price = pd.read_csv("D:/bitcoin-price.csv")
```

III. 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 two columns—Date and Close Price—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 two columns, Date and Close Price:

```
In [5]: import pandas as pd
In [6]: import matplotlib.pyplot as plt
In [7]: pd.options.mode.chained_assignment = None
In [8]: %matplotlib inline
In [9]: price = pd.read_csv("D:/bitcoin-price.csv")
In [10]: price.head()
Out[10]:
```

	Date	Close Price
0	2010-07-18 00:00:00	0.09
1	2010-07-19 00:00:00	0.08
2	2010-07-20 00:00:00	0.07
3	2010-07-21 00:00:00	0.08
4	2010-07-22 00:00:00	0.05

To get top-level information about this data, call the *info()* method on it. After calling this method, we get 2,592 records. There are two columns: Date and Close Price. Date has 2,592 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 as a numeric float type. It has 2,590 non-null records, which are two records fewer than the Date field.
The following screenshot shows the details of the info() method:

```
In [11]: price.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2592 entries, 0 to 2591
Data columns (total 2 columns):
Date           2592 non-null object
Close Price    2590 non-null float64
dtypes: float64(1), object(1)
memory usage: 40.6+ KB
```

In order to print for the records from bottom, call the tail() method. This method shows that the last two records should not exist, as they are not a date or price. We need to remove these before proceeding with further analysis.

We can see that the close price has NaN values, which means that it has missing values. We can use

this factor to remove these two records 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.

The code in the following screenshot shows the tail() method implementation:

```
In [12]: price.tail()
```

Out[12]:

	Date	Close Price
2587	2017-08-17 00:00:00	4316.34
2588	2017-08-18 00:00:00	4159.46
2589	2017-08-19 15:54:00	4062.17
2590	This data was produced from the CoinDesk price...	NaN
2591	http://www.coindesk.com/price/	NaN

Also, 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 [13]: price = price.dropna()

In [14]: price.tail()
```

Out[14]:

	Date	Close Price
2585	2017-08-15 00:00:00	4204.43
2586	2017-08-16 00:00:00	4425.30
2587	2017-08-17 00:00:00	4316.34
2588	2017-08-18 00:00:00	4159.46
2589	2017-08-19 15:54:00	4062.17

IV. 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, as shown in the following screenshot:

```
In [15]: 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 [15]: price['Date'] = pd.to_datetime(price['Date'], format = "%Y-%m-%d")

In [16]: price.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 2590 entries, 0 to 2589
Data columns (total 2 columns):
Date           2590 non-null datetime64[ns]
Close Price    2590 non-null float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 60.7 KB
```

V. 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.
The code in the following screenshot shows how to set the index to the Date column:

```
In [14]: price.tail()

Out[14]:
```

	Date	Close Price
2585	2017-08-15 00:00:00	4204.43
2586	2017-08-16 00:00:00	4425.30
2587	2017-08-17 00:00:00	4316.34
2588	2017-08-18 00:00:00	4159.46
2589	2017-08-19 15:54:00	4062.17

```
In [15]: price['Date'] = pd.to_datetime(price['Date'], format = "%Y-%m-%d")

In [16]: price.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 2590 entries, 0 to 2589
Data columns (total 2 columns):
Date           2590 non-null datetime64[ns]
Close Price    2590 non-null float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 60.7 KB

In [17]: price.index = price["Date"]
```

Next, delete the Date column as a separate column, since it is already set up as an index, as shown in the following screenshot:

```
In [17]: price.index = price["Date"]

In [18]: del price["Date"]
```

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

```
In [17]: price.index = price["Date"]

In [18]: del price["Date"]

In [19]: price.head()
```

Out[19]:

Date	Close Price
2010-07-18	0.09
2010-07-19	0.08
2010-07-20	0.07
2010-07-21	0.08
2010-07-22	0.05

VI. 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 2010:

```
In [20]: price['2010']
```

Out[20]:

Date	Close Price
2010-07-18	0.09
2010-07-19	0.08
2010-07-20	0.07
2010-07-21	0.08
2010-07-22	0.05
2010-07-23	0.06
2010-07-24	0.05
2010-07-25	0.05
2010-07-26	0.06
2010-07-27	0.06
2010-07-28	0.06
2010-07-29	0.07

We can also specify whether we want the data from a particular date.
The following screenshot shows the bitcoin price in USD from August 1, 2017:

```
In [21]: price['2017-08-01']
Out[21]:
```

Date	Close Price
2017-08-01	2735.59

We can also specify whether we want the data from a particular period spanning certain dates. The following screenshot shows the data from August 1, 2017, onward:

```
In [22]: price['2017-08-01':]
Out[22]:
```

Date	Close Price
2017-08-01 00:00:00	2735.59
2017-08-02 00:00:00	2723.58
2017-08-03 00:00:00	2814.36
2017-08-04 00:00:00	2883.68
2017-08-05 00:00:00	3301.76
2017-08-06 00:00:00	3255.00
2017-08-07 00:00:00	3431.97
2017-08-08 00:00:00	3453.16
2017-08-09 00:00:00	3377.54
2017-08-10 00:00:00	3445.28
2017-08-11 00:00:00	3679.61
2017-08-12 00:00:00	3917.65
2017-08-13 00:00:00	4111.20
2017-08-14 00:00:00	4382.74
2017-08-15 00:00:00	4204.43

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, as shown in the following screenshot:

```
In [23]: price.min()
Out[23]: Close Price    0.05
         dtype: float64
```

To get the maximum price, use the max () method, as shown in the following screenshot:

```
In [24]: price.max()
Out[24]: Close Price    4425.3
         dtype: float64
```

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

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

Out[25]:

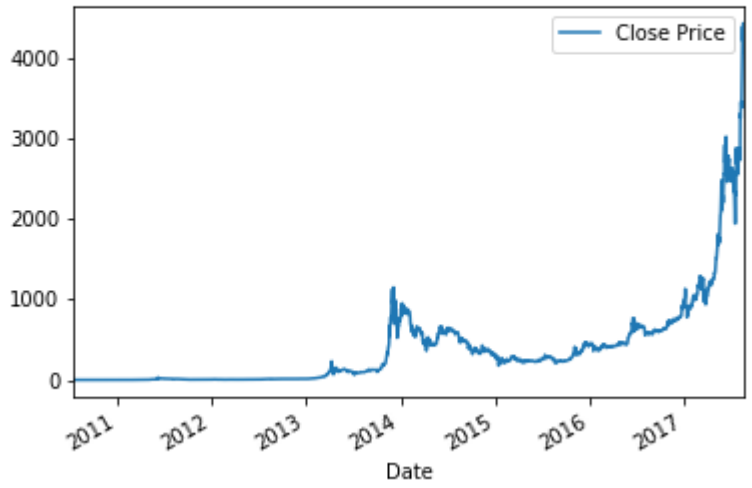
	Close Price
count	2590.000000
mean	381.646371
std	570.770109
min	0.050000
25%	8.752500
50%	233.705000
75%	545.132500
max	4425.300000

VII. 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 [17]: price.plot()
         plt.show()
```



We can also zoom in on a certain time period. For example, in order to plot the data from 2017 only, first select the data that is from 2017 and then call the plot () method on the subset of data.

In the following screenshot, we have a plot for the price data from 2017 onward:

```
In [18]: price['2017'].plot()
         plt.show()
```

Well done!! Show your results to TA!

The End of Practice 3