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

Practical Report

Practice 4
Exploring bitcoin transaction graphs

Name: Lin Yijun ID: 18120189

Date: 29 Apr. 2020

Contents

1 Introduction **3**

2 Materials, Methods and Results **3**

2.1 Bitcoin and blockchain graphs 3

2.2 Exploring, cleaning up, and analyzing data 3

2.3 Visualizing data 4

2.4 Mining Difficulty 5

3 Conclusion **7**

3.1 Functions Used 7

3.2 Practise Conclusion 7

4 References **7**

1 Introduction

In this practise, we will learn about how to get the blockchain data, and provide step-by-step information as to how to explore, clean up, analyse, and visualize this data.

2 Materials, Methods and Results

2.1 Bitcoin and blockchain graphs

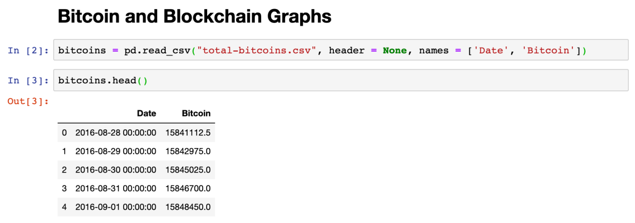
Blockchain.info is one of the best places to look at the latest bitcoin stats and graphs. There are different kinds of charts and graphs concerning bitcoin and blockchain that are available for analysis. We can also download the data in a variety of formats—CSV, JSON, and so on. We have downloaded some of this data in CSV format in the previous section, and now we will explore this data in a Jupyter Notebook.

We start by importing the modules we need. We need *pandas* for data reading, exploration, and cleanup, and we need *matplotlib* for creating the graphs.

```
import pandas as pd
import \textit{matplotlib}.pyplot as plt
pd.options.mode.chained_assignment = None
%\textit{matplotlib} inline
```

Look at the data showing the total number of bitcoins in circulation. Read the CSV file that has this data and create a *pandas DataFrame*.

The following screenshot shows the data for the total number of bitcoins in circulation:



Bitcoin and Blockchain Graphs

```
In [2]: bitcoins = pd.read_csv("total-bitcoins.csv", header = None, names = ['Date', 'Bitcoin'])

In [3]: bitcoins.head()

Out[3]:
```

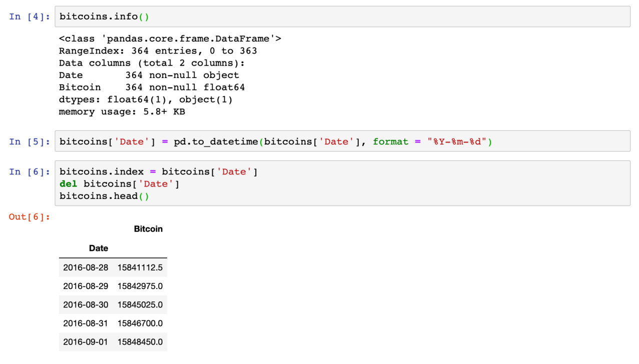
	Date	Bitcoin
0	2016-08-28 00:00:00	15841112.5
1	2016-08-29 00:00:00	15842975.0
2	2016-08-30 00:00:00	15845025.0
3	2016-08-31 00:00:00	15846700.0
4	2016-09-01 00:00:00	15848450.0

2.2 Exploring, cleaning up, and analyzing data

In order to explore this data, we use the *head()* method to look at the records from the top and we call the *info()* method on the DataFrame to get some more information, such as how many records there are, how many null or missing records there are, or what the data types of the various columns are.

We can see that the Date column is shown as object. We change this to date-time to visualize this data. To do this, we use the *to_datetime* method, and assign the converted values back to the same column— the DataFrame.

The following screenshot depicts the date format of bitcoins:

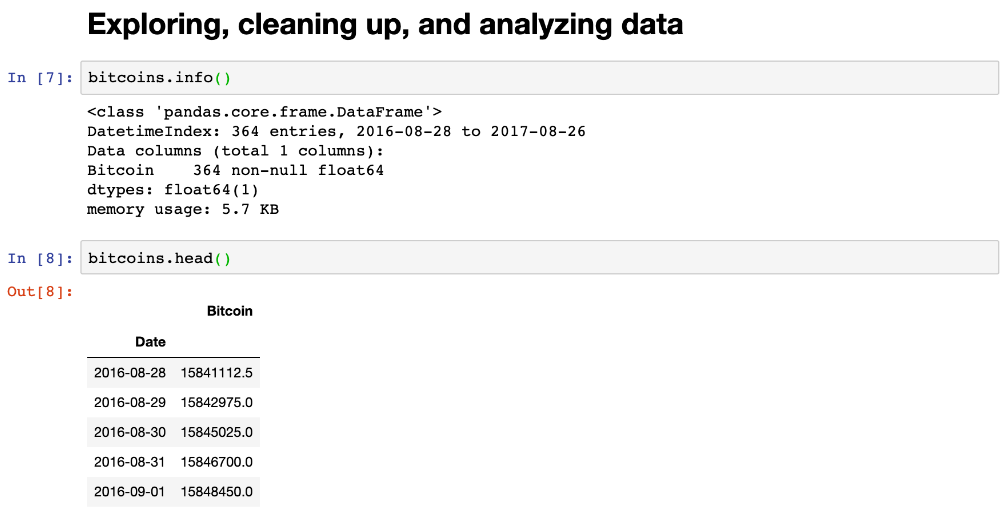


```
In [4]: bitcoins.info()

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 364 entries, 0 to 363
        Data columns (total 2 columns):
        Date       364 non-null object
        Bitcoin    364 non-null float64
        dtypes: float64(1), object(1)
        memory usage: 5.8+ KB

In [5]: bitcoins['Date'] = pd.to_datetime(bitcoins['Date'], format = "%Y-%m-%d")

In [6]: bitcoins.index = bitcoins['Date']
        del bitcoins['Date']
        bitcoins.head()

Out[6]:
```

	Bitcoin
Date	
2016-08-28	15841112.5
2016-08-29	15842975.0
2016-08-30	15845025.0
2016-08-31	15846700.0
2016-09-01	15848450.0

Set the index of the DataFrame to the Date column and delete the Date column as a separate column. Perform this step in order to take advantage of the time series features of pandas.

Now, check again whether the changes took place by calling info and head on the DataFrame. The following screenshot shows the bitcoins for a particular range of dates:



Exploring, cleaning up, and analyzing data

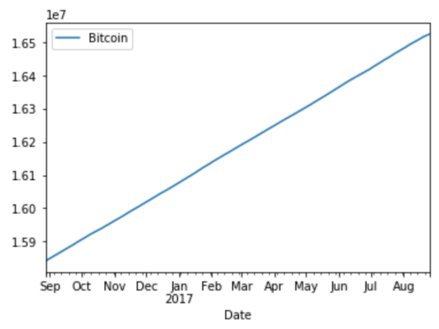
```
In [7]: bitcoins.info()

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 364 entries, 2016-08-28 to 2017-08-26
Data columns (total 1 columns):
Bitcoin    364 non-null float64
dtypes: float64(1)
memory usage: 5.7 KB

In [8]: bitcoins.head()
Out[8]:
```

	Bitcoin
Date	
2016-08-28	15841112.5
2016-08-29	15842975.0
2016-08-30	15845025.0
2016-08-31	15846700.0
2016-09-01	15848450.0

We are now ready to create a graph from this data. Call the plot () method on the DataFrame and then call the show () method to display the graph.

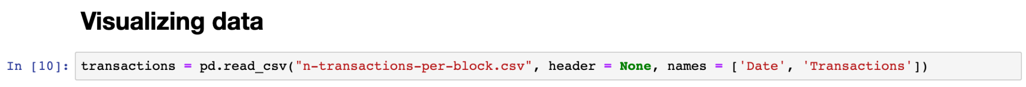
It shows the total number of bitcoins that have already been mined over the time period for which we have this record. The following screenshot describes the graph for the preceding data:

```
In [9]: bitcoins.plot()
        plt.show()
```



2.3 Visualizing data

Let's look at another example. Here, we are looking at transactions for block data that we read into the data:



Visualizing data

```
In [10]: transactions = pd.read_csv("n-transactions-per-block.csv", header = None, names = ['Date', 'Transactions'])
```

Initially, we visually explore this data using the head and info methods, as shown in the following screenshot:

```
In [11]: transactions.head()
```

Out[11]:

	Date	Transactions
0	2016-08-28 00:00:00	1147.953947
1	2016-08-29 00:00:00	1511.959732
2	2016-08-30 00:00:00	1542.339394
3	2016-08-31 00:00:00	1676.866667
4	2016-09-01 00:00:00	1689.411348

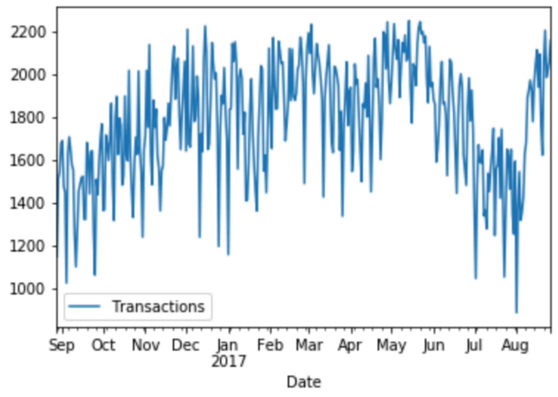
```
In [12]: bitcoins.info()

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 364 entries, 2016-08-28 to 2017-08-26
Data columns (total 1 columns):
Bitcoin     364 non-null float64
dtypes: float64(1)
memory usage: 5.7 KB
```

Next, we clean up, convert, and reshape the data, as shown in the following screenshot:

```
In [13]: transactions['Date'] = pd.to_datetime(transactions['Date'], format = "%Y-%m-%d")
         transactions.index = transactions['Date']
         del transactions['Date']
```

Finally, we visualize our transactions in the block data, as shown in the following screenshot:

```
In [14]: transactions.plot()
         plt.show()
```



2.4 Mining Difficulty

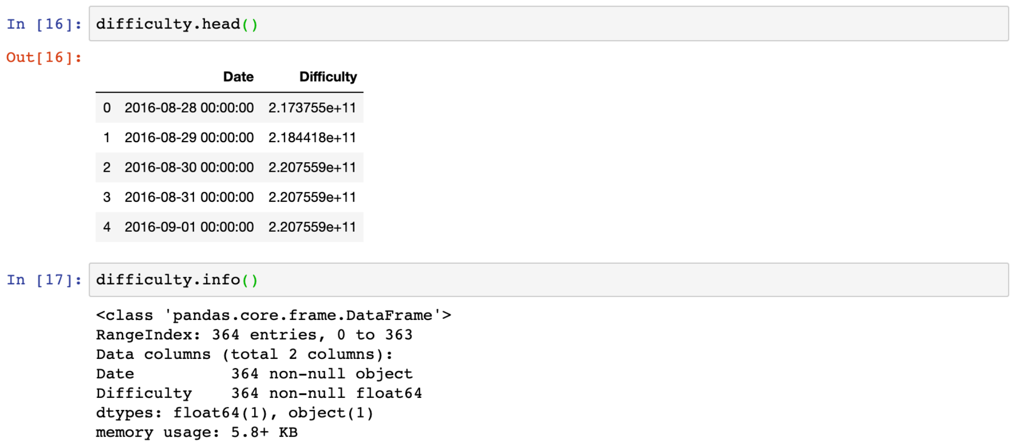
Similarly, there is another example that we should look at regarding the data showing mining difficulty. The steps for mining data are as follows:

1. Read in the data, as shown in the following screenshot:

Mining Difficulty

```
In [15]: difficulty = pd.read_csv("difficulty.csv", header = None, names = ['Date', 'Difficulty'])
```

2. Explore the data, as shown in the following screenshot:

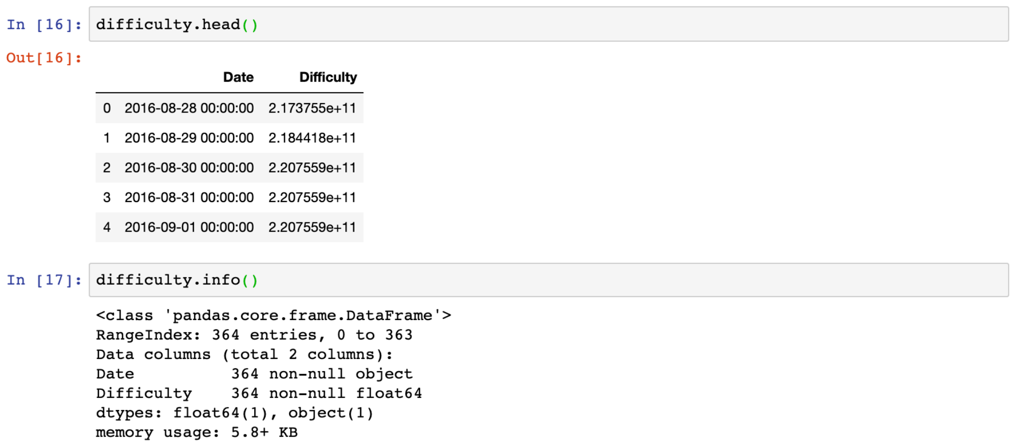


```
In [16]: difficulty.head()
Out[16]:
```

	Date	Difficulty
0	2016-08-28 00:00:00	2.173755e+11
1	2016-08-29 00:00:00	2.184418e+11
2	2016-08-30 00:00:00	2.207559e+11
3	2016-08-31 00:00:00	2.207559e+11
4	2016-09-01 00:00:00	2.207559e+11

```
In [17]: difficulty.info()

         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 364 entries, 0 to 363
         Data columns (total 2 columns):
         Date          364 non-null object
         Difficulty    364 non-null float64
         dtypes: float64(1), object(1)
         memory usage: 5.8+ KB
```

3. Clean up the data, as shown in the following screenshot:

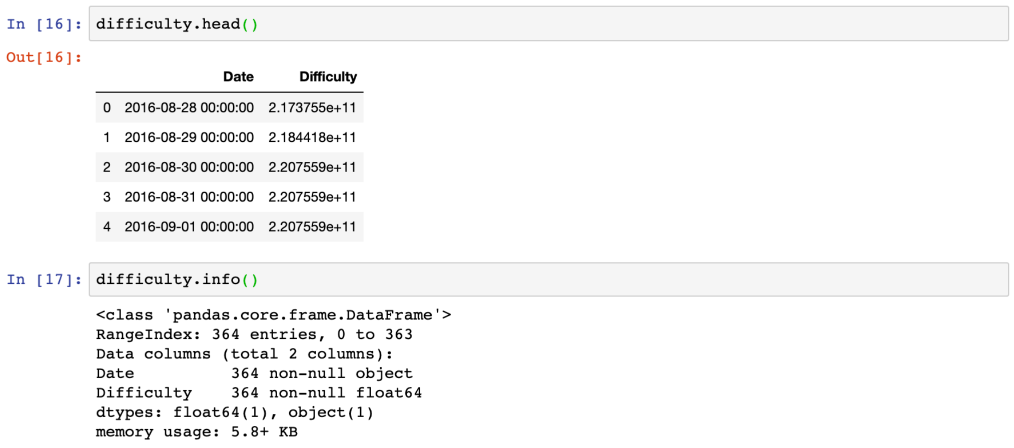


```
In [16]: difficulty.head()
Out[16]:
```

	Date	Difficulty
0	2016-08-28 00:00:00	2.173755e+11
1	2016-08-29 00:00:00	2.184418e+11
2	2016-08-30 00:00:00	2.207559e+11
3	2016-08-31 00:00:00	2.207559e+11
4	2016-09-01 00:00:00	2.207559e+11

```
In [17]: difficulty.info()

         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 364 entries, 0 to 363
         Data columns (total 2 columns):
         Date          364 non-null object
         Difficulty    364 non-null float64
         dtypes: float64(1), object(1)
         memory usage: 5.8+ KB
```

4. Finally, visualize the data, as shown in the following screenshot:



```
In [16]: difficulty.head()
Out[16]:
```

	Date	Difficulty
0	2016-08-28 00:00:00	2.173755e+11
1	2016-08-29 00:00:00	2.184418e+11
2	2016-08-30 00:00:00	2.207559e+11
3	2016-08-31 00:00:00	2.207559e+11
4	2016-09-01 00:00:00	2.207559e+11

```
In [17]: difficulty.info()

         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 364 entries, 0 to 363
         Data columns (total 2 columns):
         Date          364 non-null object
         Difficulty    364 non-null float64
         dtypes: float64(1), object(1)
         memory usage: 5.8+ KB
```

We can see that there has been a gradual increase in mining difficulty over the years, and it trends upwards. These are just a few examples of transaction graphs. There is a lot of other data available for us to explore from the bitcoin and blockchain ecosystem.

3 Conclusion

3.1 Functions Used

read_csv Read a comma-separated values (csv) file into DataFrame. Also supports optionally iterating or breaking of the file into chunks. Additional help can be found in the online docs for IO Tools.

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 the 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.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.

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 use *pandas* and *matplotlib* library and use Jupyter Notebook as IDE the same as last time.

Using the functions in the library, I successfully import the csv file again and print it out in Jupyter Notebook. And both explore and clean up the data.

Then I try *pandas* works with *matplotlib* again for visualizing data in a basic way. It is a more powerful data visualization tool than office software like Excel 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 4 Guide File.
Website: pandas.pydata.org/pandas-docs/stable/index.html
PDF document: How to Write a Practical/Laboratory Report—Learning Guide.