Submission for 5.4 - Project: Capstone Project 1: Data Wrangling

Jason Su

This exercise is about collecting data and performing data wrangling on the dataset.

I obtained a Bitcoins price dataset from Kaggle and started by loading it into my notebook environment and inspecting the structure and content of the data in it.

This data file is a CSV file for two bitcoin exchanges (Bitstamp and Coinbase) for the time period from Jan 2012 to August 2019, with minute to minute updates of OHLC (Open, High, Low, Close) prices, trading volume in Bitcoins (BTC) and trading volume in USD, and weighted bitcoin prices. The timestamps are in Unix time. Timestamps without any trades or activity have their data fields filled with NaNs.

Here are the steps I took to inspect and clean my data:

First, I imported the data file into a dataframe called Bitstamp.

In[2]:
```
bitstamp = pd.read_csv('../input/bitcoin-historical-data/bitstampUSD_1-min_data_2012-01-01_to_2019-08-12.csv')
```

A quick inspection of the first few rows of the dataframe showed that there were a lot of missing values, and the ‘Timestamp’ column was coded in Unix time, which is not very human readable.

In[3]:

```
bitstamp.head(30)
```

Out[3]:

	Timestamp	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price
0	1325317920	4.39	4.39	4.39	4.39	0.455581	2.0	4.39
1	1325317980	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	1325318040	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	1325318100	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	1325318160	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	1325318220	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	1325318280	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	1325318340	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8	1325318400	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	1325318460	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10	1325318520	NaN	NaN	NaN	NaN	NaN	NaN	NaN
11	1325318580	NaN	NaN	NaN	NaN	NaN	NaN	NaN
12	1325318640	NaN	NaN	NaN	NaN	NaN	NaN	NaN
13	1325318700	NaN	NaN	NaN	NaN	NaN	NaN	NaN
14	1325318760	NaN	NaN	NaN	NaN	NaN	NaN	NaN
15	1325318820	NaN	NaN	NaN	NaN	NaN	NaN	NaN
16	1325318880	NaN	NaN	NaN	NaN	NaN	NaN	NaN
17	1325318940	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Further inspection showed that there are about 4 million rows of data (including NaN's) and the columns are mostly in float64 format, which is good.

In[4]:

```
bitstamp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3997697 entries, 0 to 3997696
Data columns (total 8 columns):
Timestamp            int64
Open                 float64
High                 float64
Low                  float64
Close                float64
Volume_(BTC)         float64
Volume_(Currency)    float64
Weighted_Price       float64
dtypes: float64(7), int64(1)
memory usage: 244.0 MB
```

A quick look at the statistics of the values in the dataframe seems to indicate that the values are pretty correct. The mean prices are around $3,000 over the 7-year period, which makes sense. The minimum price was around 1.5 to 4 dollars per coin, while maximum was at around $19,600 which was around the all-time-high at the end of 2017. Therefore, there didn't seem to be any outliers in this dataset.

In[5]:
```
bitstamp.describe()
```

Out[5]:

	Timestamp	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price
count	3.997697e+06	2.765819e+06	2.765819e+06	2.765819e+06	2.765819e+06	2.765819e+06	2.765819e+06	2.765819e+06
mean	1.445483e+09	3.059659e+03	3.062027e+03	3.057022e+03	3.059638e+03	1.042232e+01	2.686812e+04	3.059499e+03
std	6.940318e+07	3.741168e+03	3.744835e+03	3.736985e+03	3.741134e+03	3.375010e+01	9.620425e+04	3.740910e+03
min	1.325318e+09	3.800000e+00	3.800000e+00	1.500000e+00	1.500000e+00	0.000000e+00	0.000000e+00	3.800000e+00
25%	1.385283e+09	3.742700e+02	3.745200e+02	3.740000e+02	3.742500e+02	4.530000e-01	2.865515e+02	3.742434e+02
50%	1.445637e+09	7.794500e+02	7.799100e+02	7.790100e+02	7.794900e+02	2.100451e+00	2.209966e+03	7.794137e+02
75%	1.505603e+09	5.635745e+03	5.639500e+03	5.631100e+03	5.635355e+03	8.179424e+00	1.554689e+04	5.635118e+03
max	1.565568e+09	1.966576e+04	1.966600e+04	1.964996e+04	1.966575e+04	5.853852e+03	7.569437e+06	1.966330e+04

To find the number of rows with NaN's I ran the following line of code, and found that there were around 1.2 million rows of such data.

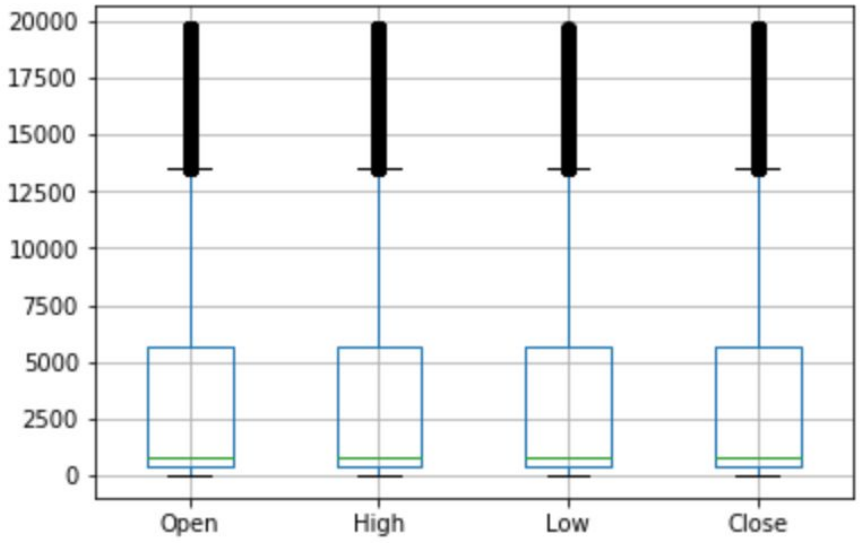
In[8]:
```
bitstamp['Open'].value_counts(dropna = False)
```

Out[8]:
```
NaN         1231878
580.00          622
450.00          622
635.00          568
448.00          562
               ...
2515.84           1
6445.64           1
9263.09           1
10241.31          1
12700.02          1
Name: Open, Length: 680541, dtype: int64
```

A quick plot of the four price columns shows that they are in the approximate range that seem to be correct based on my impression of the historical Bitcoin prices.

In[10]:
```
bitstamp.boxplot(column=['Open', 'High', 'Low', 'Close'])
```

Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbc4e7a6f28>
```



After inspection I realized that there are two steps of the cleaning that are required. Step 1 was to convert the ‘Timestamp’ column from Unix time to Pandas datetime format, and since the column values are unique I set the datetime column to be the index of the dataframe. Now it is clear to see that each row is the time-series price data for Bitcoin at a particular 1-minute interval.

In[15]:
```
bitstamp.set_index(pd.to_datetime(bitstamp['Timestamp'], unit='s'), inplace=True, drop=True)
```

In[16]:
```
bitstamp.head(30)
```

Out[16]:

Timestamp	Timestamp	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price
2011-12-31 07:52:00	1325317920	4.39	4.39	4.39	4.39	0.455581	2.0	4.39
2011-12-31 07:53:00	1325317980	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-31 07:54:00	1325318040	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-31 07:55:00	1325318100	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-31 07:56:00	1325318160	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-31 07:57:00	1325318220	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-31 07:58:00	1325318280	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-31 07:59:00	1325318340	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-31 08:00:00	1325318400	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-31 08:01:00	1325318460	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-31 08:02:00	1325318520	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Step 2 involved figuring out a way to remove or fill out the NaN's in the dataframe. Since the NaN rows tend to show up as a group of consecutive rows, I decided to use the forward fill method to fill the NaN values with the last available price values right before the NaN's began. This way the data would be filled out with relatively accurate values (since the prices won't fluctuate too much in the matter of a few minutes) and the overall trend of prices during those time segments would still be preserved. Also, since according to the description of the dataset the NaN rows represent time periods where there were no trading activities on the exchanges, it is safe to assume that the prices stayed constant during those periods.

In[17]:
```
bitstamp.fillna(method = 'ffill', inplace = True)
```

In[18]:
```
bitstamp.head(30)
```

Out[18]:

Timestamp	Timestamp	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price
2011-12-31 07:52:00	1325317920	4.39	4.39	4.39	4.39	0.455581	2.0	4.39
2011-12-31 07:53:00	1325317980	4.39	4.39	4.39	4.39	0.455581	2.0	4.39
2011-12-31 07:54:00	1325318040	4.39	4.39	4.39	4.39	0.455581	2.0	4.39
2011-12-31 07:55:00	1325318100	4.39	4.39	4.39	4.39	0.455581	2.0	4.39
2011-12-31 07:56:00	1325318160	4.39	4.39	4.39	4.39	0.455581	2.0	4.39
2011-12-31 07:57:00	1325318220	4.39	4.39	4.39	4.39	0.455581	2.0	4.39
2011-12-31 07:58:00	1325318280	4.39	4.39	4.39	4.39	0.455581	2.0	4.39
2011-12-31 07:59:00	1325318340	4.39	4.39	4.39	4.39	0.455581	2.0	4.39
2011-12-31 08:00:00	1325318400	4.39	4.39	4.39	4.39	0.455581	2.0	4.39
2011-12-31 08:01:00	1325318460	4.39	4.39	4.39	4.39	0.455581	2.0	4.39
2011-12-31 08:02:00	1325318520	4.39	4.39	4.39	4.39	0.455581	2.0	4.39
2011-12-31 08:03:00	1325318580	4.39	4.39	4.39	4.39	0.455581	2.0	4.39

A quick inspection of the tail of the dataset shows that they do match the prices of Bitcoin in August this year.

In[19]:
```
bitstamp.tail(30)
```

Out[19]:

Timestamp	Timestamp	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price
2019-08-11 23:31:00	1565566260	11525.00	11532.11	11496.79	11504.84	29.196789	336153.864640	11513.384837
2019-08-11 23:32:00	1565566320	11506.83	11506.83	11506.83	11506.83	0.022696	261.163156	11506.830000
2019-08-11 23:33:00	1565566380	11508.67	11510.43	11498.74	11498.74	5.393892	62085.156759	11510.270561
2019-08-11 23:34:00	1565566440	11517.73	11525.37	11517.73	11525.37	0.415343	4784.035332	11518.264964
2019-08-11 23:35:00	1565566500	11525.37	11525.37	11504.74	11514.93	5.482086	63135.383551	11516.670945
2019-08-11 23:36:00	1565566560	11520.34	11520.34	11520.34	11520.34	0.558979	6439.627902	11520.340000
2019-08-11 23:37:00	1565566620	11520.31	11520.31	11508.49	11508.49	0.140438	1616.506321	11510.457497
2019-08-11 23:38:00	1565566680	11514.01	11514.01	11514.01	11514.01	0.354032	4076.332709	11514.010000
2019-08-11 23:39:00	1565566740	11520.36	11520.36	11512.01	11512.01	0.049529	570.398243	11516.342869

Now the dataset seems to be suitable for further analysis. Finally I removed the Unix time column to make the dataset easier to read.

In[37]:
```
bitstamp_clean = bitstamp.loc[:, ['Open', 'High', 'Low', 'Close', 'Volume_(BTC)', 'Volume_(Currency)', 'Weighted_Price']]
```

In[39]:
```
bitstamp_clean.head()
```

Out[39]:

Timestamp	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price
2011-12-31 07:52:00	4.39	4.39	4.39	4.39	0.455581	2.0	4.39
2011-12-31 07:53:00	4.39	4.39	4.39	4.39	0.455581	2.0	4.39
2011-12-31 07:54:00	4.39	4.39	4.39	4.39	0.455581	2.0	4.39
2011-12-31 07:55:00	4.39	4.39	4.39	4.39	0.455581	2.0	4.39
2011-12-31 07:56:00	4.39	4.39	4.39	4.39	0.455581	2.0	4.39

In[44]:
```
bitstamp_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3997697 entries, 2011-12-31 07:52:00 to 2019-08-12 00:00:00
Data columns (total 7 columns):
Open                 float64
High                 float64
Low                  float64
Close                float64
Volume_(BTC)         float64
Volume_(Currency)    float64
Weighted_Price       float64
dtypes: float64(7)
memory usage: 404.0 MB
```