CSCI 5040 Programming Assignment 3: BitCoin

Learning Outcomes

- Open a CSV file in Spark
- Understand the Unix Time Stamp
- Inch closer to Big Data
- Work with aggregation functions and (hopefully) pivot a dataset

Required Reading

Chapter 6 and 7 (up through the description on joins) in the textbook.

More required reading.

- https://en.wikipedia.org/wiki/Unix_time

Unix time is the number of seconds which have passed since midnight of January 1st, 1970 Coordinated Universal Time (UTC).

The concept is as follows. On January 1st, 1970 at midnight (UTC) the time was 0. 1 second after that, the time was 1. There are 86,400 seconds in a day, so midnight of January 2nd, 1970 (UTC) was time 86400. Time has been incrementing by 1 per second ever since. January 1st, 1970 (UTC) is the beginning of time.

At the time of writing this homework assignment, it is 1538037601. On February 13, 2009, it was 1234567890 and there were Unix parties marking the occasion. (Humans are odd.)

There are numerous benefits to representing time as a single integer.

- First, numbers are small to a computer. Any date and time since 1970 (up to 2038) can by represented in 4 bytes (or 32 bits). Contrast this to the ISO 8601 standard for representing dates (such as "2018-09-27T08:51:48+00:00") which can be as long as 25 bytes. Clearly, a single number wins.
- Second, numbers can be easily converted to dates that we are familiar using. With a little math, you can figure out the date that a Unix time value took place on. Take a Unix time value, divide by 86400, and you have the date the days since January 1st, 1970. Of course, you'll have to use a mathematical floor operation on your result.

The only real downside is that numbers aren't readable as dates to humans. These aren't for a humans. They are for computers. To a computer, that isn't a weak point.

Motivation!

I found a dataset on BitCoin from 4 different market indexes. I think the bitcoin craze is settling down (thank goodness). They record all times using Unix time. Let's see what we can learn, shall we? Here are the for indexes.

- CoinBase (US Dollars)
- BitStamp (US Dollars)
- BitFlyer (Japanese Yen)
- CoinCheck (Japanese Yen)

When you start to look at the closing prices for each of the indexes, you'll notice that the US exchanges amounts are much smaller than the Japanese exchanges. This is due to the exchange rate of currency.

Each index has the same schema. Most of these are obvious except for the Timestamp (which is in Unix time, described earlier).

```
root
 |-- Timestamp: integer (nullable = true)
 |-- Open: integer (nullable = true)
 |-- High: integer (nullable = true)
 |-- Low: integer (nullable = true)
 |-- Close: integer (nullable = true)
 |-- Volume_(BTC): double (nullable = true)
 |-- Volume_(Currency): double (nullable = true)
 |-- Weighted_Price: double (nullable = true)
```

We will only use 2 columns from each dataset: Timestamp and Close. I will ask you to rename the close column.

Instructions

1. Load each dataset into their own dataframes. I did mine the typical way with an "inferSchema" option. I stored all of these in "var" variables instead of "val" to save on memory.
2. In each data set, rename the "Close" column to that of the name of the index (Coinbase, CoinCheck, etc.). For example, I named my CoinCheck dataframe "cc". To rename the column, I ran: cc = cc.withColumnRenamed("Close", "CoinCheck").
3. Create a single, unified dataset of records with timestamps common across all four indices. I used "inner" joins as described in chapter 7, but I used code to prevent duplicated columns described on page 147 and 148 in the textbook. We didn't get that far in class in my examples. Let's see if you can figure out what the book is trying to do. Specifically, I used "Approach 1" on page 148. After completing this step, I had a dataset with 511,571 rows.
4. Condense the data set down to the five relevant columns: timestamp and the four named columns from step 2.
5. Compute the mean of each of the 4 closing columns in a single select statement.
6. Compute the sample standard deviation of each of the 4 closing columns in a single select statement.
7. Comparing US and Japanese currency is like comparing apples to oranges. The current exchange rate changes independently of BitCoin changes. Still, we can learn something. Divide the sample standard deviation by the mean **for each of the four exchanges**.

This term is called the "coefficient of variation". Again, do this in a single select statement.

8. Which exchange has the smallest "coefficient of variance"? This exchange is the most stable to price fluctuations.
9. Finally, since I need an excuse for telling you about Unix timestamps, take the timestamp, divide by 86400, then **floor** the result and count the number of distinct entries. (My answer was between 300 and 400.)
10. Do the same thing that you did in step 9, except use Spark's approximate counting tool with an approximation value of 0.1 for a faster result. (My answer this time was still between 300 and 400.)

Notes and Comments

Build a text file of the commands that you used to produce the results.