**Crypto Currency Prediction and Recommendation System**

**Introduction**

This project provides access to historical and real-time price data for the two leading cryptocurrency projects, Bitcoin and Ethereum, with a granularity of 1 minute.

The dataset for Bitcoin covers the period from approximately December 2014 up to the current month of August 2020. It can be effortlessly updated to include more recent data as needed.

For Ethereum, the data spans from the re-release of Ethereum (following a significant fork due to coin theft) around May 2016 until August 2020. Updating the Ethereum data is a straightforward process that can be done using a Jupyter notebook provided in this repository.

**Overview**

Crypto currency has become a popular form of investment in recent years, with many people looking for ways to capitalize on its potential growth. However, investing in crypto currency can be risky, as the market is highly volatile and unpredictable.

This is where crypto currency analysis and prediction comes into picture. By analyzing trends and patterns in the market, investors can gain insight into the future performance of various crypto currencies, allowing them to make more informed investment decisions.

**Extract Data**

**Downloaded Files**

The foundation of the data used in this project was pulled from kaggle.com, specifically from [here](https://www.kaggle.com/mczielinski/bitcoin-historical-data) for Bitcoin and [here](https://www.kaggle.com/prasoonkottarathil/ethereum-historical-dataset) for Ethereum. The initial data provided by these sources determined the earliest data points for this project, for reasons to be explained shortly. The 1-minute time interval was chosen because it was the smallest level of granularity, and the other data sets for hourly or daily prices could be closely replicated using this information.

Two .csv files were used as the backbone of this project. The first, for Bitcoin, is named bitstampUSD\_1-min\_data\_2012-01-01\_to\_2020-04-22.csv and is available [here](https://github.com/Gendo90/Crypto-Historical-Prices/blob/master) in this repository or from the link above.

**API to Update Data**

The Coinbase Pro "candles" API was used for gathering the more recent data (from January 2019 to August 2020 for Bitcoin and from April 2020 to August 2020 for Ethereum) and is used by to update the csvs and database to maintain consistency. The Bitcoin data was originally drawn from Coinbase, as well as other exchanges, but the Coinbase API was free-to-use and works fine - see it [here](https://docs.pro.coinbase.com/#get-historic-rates). Unfortunately, the Coinbase API did not return historical prices at the 1-minute granularity level before the start dates for the Bitcoin or Ethereum data sets, so that data is not available in this project at this time - there may be a way to get more historical data at some point, especially using another exchange and possibly by paying to get an API key, so that will hopefully be added when/if possible.

**Transform Data**

The data had to be transformed from its original state into workable, clear columns for human usage. Both datasets use the same columns in the .csv's from this project, and this standardized format is scalable if more cryptocurrencies are added at a later date. The column headers in the .csv files are:

["Unix Timestamp", "Date", "Symbol", "Open", "High", "Low", "Close", "Volume"]

are measured in the following units:

[Epoch Time, YYYY-MM-DD HH:MM:SS+GMT, 'Coin Code'-'Fiat Currency Code', Fiat Currency, Fiat Currency, Fiat Currency, Fiat Currency, # of Coins]

and have the Python data types:

[int64, datetime64[ns, UTC], object, float64, float64, float64, float64, float64]

respectively.

The data cleaning process is beyond the scope of this README - needless to say, it is usually 80% of the work on any given data science project, and this was no exception! Any new data sources, for more cryptocurrencies or older Bitcoin or Ethereum prices, would need to be carefully, manually cleaned, and standardized to match the csv's included in this repo already.

Some examples of the transform process that involved cleaning and updating the data would include dropping null values, ensuring that the Unix timestamps are sorted and contain no duplicate values, and that the cryptocurrency symbol and date formats match the requirements of the Coinbase API. In fact, every time the csv data is read, the "Date" column (which is an aware datetime object) needs to be converted to the correct type. Converting this column automatically is one of the planned features of this project.

There is a combined "Extract and Transform" Jupyter notebook in this repository, but it is rough and will be refined when possible, likely when a new data set is introduced. There is no need to run this notebook, but anyone can look through it to get some idea of the iterative data cleaning process involved with this sort of data.

**Load Data**

**Create SQL Tables**

The file [create\_tables.sql](https://github.com/Gendo90/Crypto-Historical-Prices/blob/master/create_tables.sql) should be loaded or copied into a PostgreSQL database editor, like pgAdmin. Once the code to create the 'bitcoin' and 'ethereum' tables is loaded, run the code to create those two tables. This must be completed before the next step, or it will not work.

The tables have the same structure but different values. The basic table columns are as follows:

[unix\_timestamp, entry\_date, symbol, open\_price, high\_price, low\_price, close\_price, coin\_volume]

These column names were chosen as SQL-compatible names, because some of the original column names had spaces in them ("Unix Timestamp") and a lot of the others were important in the SQL language ("Date", "Open", "Close", etc.).

The tables have the following kinds of data types, corresponding to each column:

[INT, TIMESTAMP WITH TIME ZONE, VARCHAR(10), DECIMAL, DECIMAL, DECIMAL, DECIMAL, DECIMAL]

The tables have composite primary keys, which consists of the "unix\_timestamp" and "symbol" columns because that combination should be unique for each table and across other tables - because different tables could have the same timestamp values but then they would have different symbols, and the timestamp values should be unique within a single table (no duplicate data) even if they all share the same symbol. This combination could therefore be helpful in identifying data in future tables that consist of combinations (or aggregations - basically any transformation) of the data from the raw coin price tables created here.

**Load Cryptocurrencies into SQL**

Create a config file named config.py in the main repo directory - here - in order to load your specific username and password for PostgreSQL. The file should contain only two lines, that look like:

user = 'USERNAME'

pw = 'PASSWORD'

where the USERNAME is the username for your PostgreSQL database, and PASSWORD is the password for the database.

You are now ready to run [this notebook](./ETH\ &\ BTC\ Load\ Into\ SQL.ipynb) and load the .csv data into your database!

Follow the instructions provided in the "ETH & BTC Load Into SQL.ipynb" notebook listed above. Remember to check your connection string to connect on the correct port to the correct database! There should be printed feedback that you have loaded first the Ethereum, and then the Bitcoin data correctly.

**Update Data**

Open [this notebook](https://github.com/Gendo90/Crypto-Historical-Prices/blob/master/Update_Data.ipynb) and run once the data has been loaded into SQL to confirm that the .csv file and database information match for a specific cryptocurrency, and then scrape data from the Coinbase API to update the .csv file and database table for that specific cryptocurrency to the current time!

Please note that Update\_Data.ipynb should be the only notebook you run after the data has initially been loaded into the SQL database, as described above. None of the other notebooks should be touched after that point!

Finally, the notebook only updates the data by one year maximum. So if you had a cryptocurrency with data from 2015 and it is now 2020, you may need to run the notebook about 5 times successively to get the most up-to-date data from 2020. This date limit was created to break the updates for long periods into chunks - scraping about a year of data takes about an hour on my laptop, so breaking the work into smaller segments makes the process more robust and reliable.

**Cryptocurrency Historical Prices**

**Content**

The dataset has one csv file for each currency. Price history is available on a daily basis from April 28, 2013. This dataset has the historical price information of some of the top crypto currencies by market capitalization.

Date : date of observation

Open : Opening price on the given day

High : Highest price on the given day

Low : Lowest price on the given day

Close : Closing price on the given day

Volume : Volume of transactions on the given day

Market Cap : Market capitalization in USD

**Streaming**

**Trade Streams**

The Trade Streams push raw trade information; each trade has a unique buyer and seller.

Stream Name: <symbol>@trade

Update Speed: Real-time

Payload:

{

"e": "trade", // Event type

"E": 1672515782136, // Event time

"s": "BNBBTC", // Symbol

"t": 12345, // Trade ID

"p": "0.001", // Price

"q": "100", // Quantity

"b": 88, // Buyer order ID

"a": 50, // Seller order ID

"T": 1672515782136, // Trade time

"m": true, // Is the buyer the market maker?

"M": true // Ignore

}

**Kline/Candlestick Streams**

The Kline/Candlestick Stream push updates to the current klines/candlestick every second.

Kline/Candlestick chart intervals:

s-> seconds; m -> minutes; h -> hours; d -> days; w -> weeks; M -> months

1s

1m

3m

5m

15m

30m

1h

2h

4h

6h

8h

12h

1d

3d

1w

1M

Stream Name: <symbol>@kline\_<interval>

Update Speed: 1000ms for 1s, 2000ms for the other intervals

Payload:

{

"e": "kline", // Event type

"E": 1672515782136, // Event time

"s": "BNBBTC", // Symbol

"k": {

"t": 1672515780000, // Kline start time

"T": 1672515839999, // Kline close time

"s": "BNBBTC", // Symbol

"i": "1m", // Interval

"f": 100, // First trade ID

"L": 200, // Last trade ID

"o": "0.0010", // Open price

"c": "0.0020", // Close price

"h": "0.0025", // High price

"l": "0.0015", // Low price

"v": "1000", // Base asset volume

"n": 100, // Number of trades

"x": false, // Is this kline closed?

"q": "1.0000", // Quote asset volume

"V": "500", // Taker buy base asset volume

"Q": "0.500", // Taker buy quote asset volume

"B": "123456" // Ignore

}

}

**Database**

**Create and Connect to Database Deployments**

To create and connect to database deployments in Atlas, follow these steps:

1. Choose a database deployment type:

Compare different use cases and feature support to determine the most suitable database deployment type for your needs.

2. Create the database deployment:

Depending on your chosen deployment type, create a cluster, serverless instance, or global cluster to set up your database deployment in Atlas.

3. Connect to the database deployment:

There are several ways to connect to your database deployment:

- Browse Data via the Atlas UI:

Use the Atlas UI to browse and manage your data directly through the web interface.

- Connect via Your Application:

Configure your application to connect to the database deployment using the connection details provided by Atlas. This typically involves specifying the connection string, authentication credentials, and relevant drivers or libraries for your programming language.

- Connect via Compass:

MongoDB Compass is a graphical tool for interacting with MongoDB databases. Use Compass to connect to your Atlas database deployment by providing the connection string and authentication details.

- Connect via mongosh:

mongosh is a shell interface for MongoDB. Connect to your database deployment by launching mongosh and providing the connection string and credentials.

- Connect via BI Connector for Atlas:

If you need to connect business intelligence (BI) tools to your Atlas database deployment, you can use the BI Connector for Atlas. Follow the documentation provided by Atlas to configure and connect the BI Connector.

By following these steps, you can create and connect to your database deployments in Atlas, allowing you to easily store and manage your data.