Università della Svizzera italiana

Prof. P. Gruber
P. Montemurro
2021-04-09

Programming for Finance II
Riccardo Cosenza, Jan Gobeli , Silvia Magni & Ivana Pallister

Binance-Rekt
Assignment 1

Hand in before:
12 April, 2021
23:00 pm

1 Introduction

The main aim of this project is to set-up an automated connection between Binance, a financial exchange, and to retrieve trading data in real time. The data concerning this project is mainly focused on monetary losses and liquidations during a given trading period. The data about is then stored into a server, until the total hourly statistics are tweeted out using a cron job.
The minimum requirements of this project were indeed to:

- Set up a Linux server with a virtual server provider;
- set up a database using SQL commands;
- write a program which downloads some data and writes it into the database;
- set up a cron job to regularly execute the above mentioned task.

The full documentation of this project can be found on **Github**.

2 User Guide

2.1 Settings

In order to start the project, the user needs to access its virtual server from its computer through Putty or Terminal. There are various virtual server providers online, both free and upon subscription; we personally choose Contabo as our virtual server provider.
This server was selected as it was already in use by one of the team members prior to starting the project. The specifics of the server in question are listed in the table below.

Storage VPS
2 vCPU Cores
4 GB RAM
300 GB HDD
100 Mbit/s Port

2.2 Deployment on server - README.md:

1. After logging in the server, the user must install miniforge using the 'wget' command. Miniforge will enable the user to work on a separate environment and to install all the needed packages. Additionally, everything is installed through Conda. Ultimately, the Conda must be activated. The next code will be directly imported from a file

```
>>> wget -N https://github.com/conda-forge/miniforge/releases/latest/download/
    Miniforge3-Linux-x86_64.sh

>>> bash Miniforge3-Linux-x86_64.sh -b -u -p ~/conda

>>> source ~/conda/bin/activate
```

2. A series of packages needs to be installed:

   - Websocket-client is installed through Conda, and it is needed in order to connect to Binance.
   - Pytz serves for date-time conversion and helps users serving international client's base.
   - pandas, an open source library providing the user data structures and data analysis tools.
   - Tweepy, a Python library for accessing Twitter API. influx_line_protocol is installed through pip, and its function is to send data to the server.

```
>>> conda install websocket-client pytz pandas tweepy
>>> pip install influx_line_protocol
>>> sudo apt-get install tmux docker.io
```

3. At this point the user must clone the Github repository of the project.

```
>>> git clone https://github.com/jaNGOB/Binance_rekt.git
```

4. The user needs to open a new Tmux window and run the docker, which creates the quest database 'questdb'.

```
>>> tmux
>>> docker run -p 9009:9009 questdb/questdb
```

5. Following, the user needs to update the credentials.ini file with his Twitter API information and desired database name.

6. Lastly, the user should create another tmux window and to run the main python code 'main.py'.

```
>>> tmux
>>> python main.py
```

   Note: in order to tweet out regularly our hourly findings, we use a CRON-job.
   In order to create a new CRON-job, crontab -e can be typed into the Terminal which will open a file. After the file is opened, the following line should be added to the bottom of it:

```
0 * * * * ~/usi/bin/python3.8 ~/Binance_rekt/twitter.py
```

3 Code analysis

3.1 main.py

This file contains the main function of the project. Running this code will create a connection to the Binance exchange using Websocket, which in turn will create a connection to the Database.
Once started, this program runs forever until it encounters an error or CTRL+c is pressed. In particular, at the top of the main code we import some packages, notably the BinanceWebsocket from the binance_connect file. Below we find a basic logging function: the logging is the process of writing information into log files, the basicConfig() configures the root logger.
The ConfigParser module, used in order to manage user-editable configuration files for applications, is needed in order to read and configure the credentials.ini file. A handler function captures commands such as Ctrl+C to kill the program, and it basically performs three actions: prints "This is the end", closes the websocket and exits. At the bottom of the main code we initiate a signal, which is the handler function defined.
At this point we check if the websocket is connected into the main file.

3.2 binance_connect.py

This object connects to the stream of forced orders, which are orders Binance executes on behalf of traders because they are being liquidated. Several subroutines are implemented. As usual different packages are loaded: json for the format, the websocket to connect to Binance, DataBase, logging, Thread and time.
We find, as in the main file, the logging process, and then we define a new object, the Binancewesocket, in order to initiate the program. Inside the Binancewebsocket class the two most important portions of code concern:

- On_messages, in words what happens if we receive a json message from Binance. Every time this happens, the message is saved to the database for later access. In particular messages are loaded in json format and then sent in new_message into the database.
- _connect, which constitutes the websocket to establish a connection with Binance. In particular, we establish a connection with all market liquidation order streams taking into account the US dollars futures market of Binance. This connection enables us to see in real time all liquidations happening in payload format.
- We also define __init__ to initialize the websocket, on_error, on_close, on_open. Ultimately, we initiate the websocket connection using multithreading, which allows us to speed up the program executing different subprocesses in parallel.

3.3 database.py

This file connects to the database QuestDB and receives new liquidations data. The liquidations are added to a current batch until the set batch_size is reached. Once this happens, the whole batch gets pushed into the database using the influx_line_protocol.
First of all, three packages are imported: Metric from influx_line_protocol, logging and socket. We also find the logging function as in the previous two files.
We initialize the metric to be pushed into the database. The name of the database is set in another file, called credentials.ini, if the database does not exist yet, one will be created. Alternatively, a new database could also be created by accessing our server on Port9000, by using the following line of code.

```
1 CREATE TABLE test (PAIR symbol, PRICE double, QUANTITY double, USDVALUE double, timestamp
      timestamp);
```

Two empty strings are defined, in these we will write down what we actually need to send to the database. We also define a counter and the batch size: in this way the program will wait until we have enough data (according to the batch size) and then send all this data together at once to the database. Also the batch_size is set through the credentials.ini file. Then self-HOST and self.PORT are defined.
Follows a definition of what happens in practice anytime we receive a new message: the counter counts it as one, we get a timestamp and add values for price, quantity, usd-value, and a tag for pair to metrics. Everything is converted into a string in numerical format and effectively added back to metrics.
Thanks to an if function, anytime the counter achieves the batch size all data collected is sent to the server.

3.4 credentials.ini

This file has been set up so that the user has one single site where he can define multiple variables: the database name, the batch size and, for what concerns Twitter, consumer key, consumer secret, access token and access token secret.
In order to create a new interface which uses Twitter data, we needed to register a Twitter application. Twitter's keys and secrets we defined in the credentials.ini code come from the configuration of this application.

3.5 tweets.txt

Inside this file we set the general form of the tweets, leaving blank spaces identified by open and closed curly brackets.

3.6 twitter.py

This file reads data from a local QuestDB and creates a tweet. When providing the keys and secrets to the Twitter account, it will automatically tweet out the collected statistics in a predefined format.
We start by loading a series of packages, notably 'tweepy', an open source Python package which allows us to conveniently access the Twitter API we registered through Python. The other modules that we load are io, random, requests, configparser, pandas.
Through the configparser module we read the credentials.ini file and set the access keys for authentication which is performed through OAuth. Then we set the api variable to the object returned by tweepy.API.
At this point we create the get_data function: this function goes into the database, reads everything that is inside it, takes the data collected in the last hour and in this time frame counts how many liquidations happened, the maximum value liquidations reached, and sums the value of all liquidations.
Ultimately we define the create _tweets function, which opens the tweets.txt file and fills in the blank spaces with the collected data. The content of the updated file is then tweeted out through the api.update_status command, which takes the just completed text file as input.

3.7 get_data.ipynb

This file is basically a testing notebook, set up in order to try out various functions and commands, such as the connection to the database and the configparser.

4 @burning_money

Once the user performs all the steps described in the user guide in order to deploy the server and implement the program, Binance_rekt starts working in a fully automated way.
The final visible outcome produced by the server can be seen on the Twitter account burning_money (**@burning_money**), which, as desired, tweets out every hour liquidations news in the form detailed in the tweets.txt file.
As is clear from this wonderful user guide, **@burning_money** is on the verge of going viral. As such, it is of great importance for any reader of this document to follow this account. Enjoy the sneak peak below!

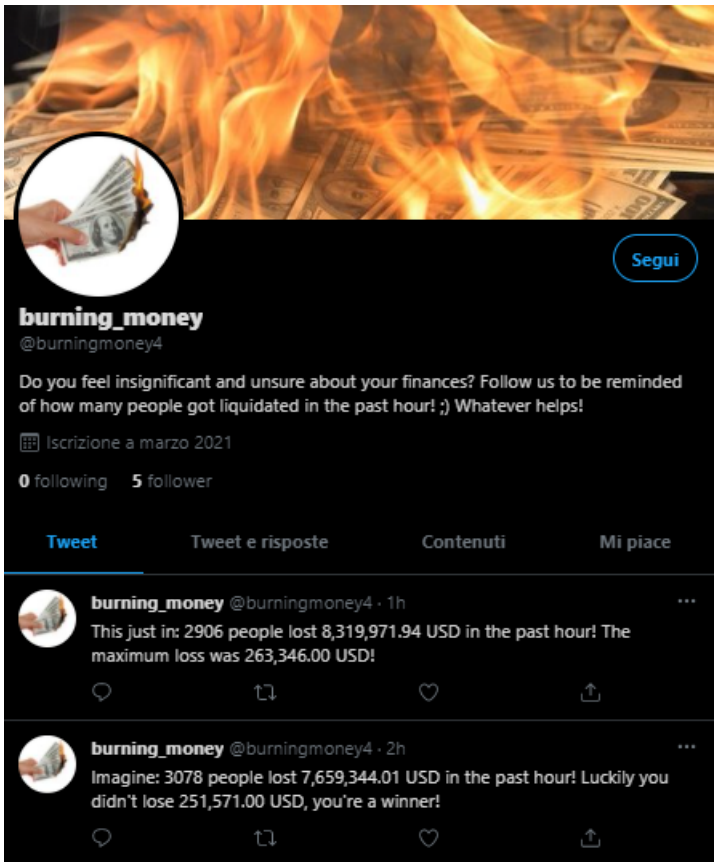

Figure 1: @burning_money