Journal of Data Science xx (xx), 1–8
July 2020
DOI: 0000

Supplementary Materials to 'The Python package open-crypto: A cryptocurrency data collector'

Steffen Günther[1], Christian Fieberg [2,*], and Thorsten Poddig[1]

[1]*Department of Finance, University Bremen, Germany*
[2]*Empirical Capital Market Research and Derivatives, University Bremen, Germany*

This online supplemental covers several aspects. Section 1 relates to the installation process. The following Section 2 illustrates the usage of *open-crypto* with real-time data, in particular order book and recent transactions data. Section 3 covers the cross-software usability and database connectivity to *MySQL* and *PostgreSQL*.

Furthermore, the online supplemental provides scripts (as *Jupyter Notebooks* and Python files) for all figures and tables used in the main manuscript. However, often the replication of figures is limited, as cryptocurrency data sources change frequently. Specifically, all real-time related data can no longer be obtained. Order-book snapshots (provided below in Section 2) are typically not offered ex-post by an exchange and are thus no longer retrievable (but are provided as *SQLite* databases upon request). Figures and tables regarding historical data may change as well, as exchanges (and cryptocurrencies) occur and vanish frequently. For example, when recreating the distribution of currency pairs across exchanges, the histogram will most likely change over time as some exchange disappear or list/remove cryptocurrencies.

Our scripts provided in the online supplemental replicate all tables and figures, as illustrated in the manuscript. The data itself may be down-scaled (i.e. from minute data to daily data) to decrease the size of the files. All original datasets are available upon request. The results remain unchanged.

1 Installation

1.1 Installation via pip

The package is uploaded to the Python package index (PyPI). Several third-party dependencies are defined in the setup file and installed automatically. The requirements for this tool are Python greater or equal to version 3.8. After downloading Python (`https://www.python.org/downloads/`), it needs to be added to `PATH` during the installation when using Windows. Users should be able to open Python from the terminal by typing either `python` or `python3`.

1.2 Troubleshooting

During development and testing of *open-crypto*, several issues occurred and got resolved. Nevertheless, the most important issues are listed below:

- MacOS users may need to install ssl-certificates before running *open-crypto*. We provide a root-certificate if none is found by Python. However, since this issue can be resolved easily by executing `Install Certificates.command` within the Python folder of the application directory, we recommend doing so. However, *open-crypto* will notify the user if this is necessary.

*Corresponding author Email: cfieberg@uni-bremen.de.

- Using popular IPython kernels like *Spyder* or *Jupyter Notebook* may conflict with the *asyncio* event loop opened by *open-crypto*. That is, these kernels run within an *asyncio.BaseEventLoop* itself, and by construction *asyncio* does not support nested event loops. We tackle this issue by applying *nest-asyncio* (https://pypi.org/project/nest-asyncio/) which patches this shortcoming for the most popular distributions. If this work-around fails, *open-crypto* will raise a `RuntimeError`. In this case, we recommend either switching the IDE or running *open-crypto* from the terminal with plain Python.
- Running the program with *PostgreSQL* as database server may result in installation problems with the client, *psycopg2*. Try installing the client manually, by typing `pip install psycopg2-binary` in the terminal.

2 Real-time data

2.1 Data types and explanations

This Section demonstrates the capabilities of *open-crypto* when dealing with real-time data. In the main article we only provided illustrations on historical data, from both, platforms and exchanges. This section introduces two additional request methods for order book data and recent transactions. We start with an introduction into order books, followed by transactions and their linkage. Finally, we provide snippets from our database and add some (basic) financial insights.

The two request methods considered for real-time data are order books and trades. Order book data provides information about orders filled but not yet executed. Commonly, the 50 highest buy orders (bids) and lowest sell orders (asks) at a particular point in time, sorted by price, are retrieved. The spread between the highest bid price (buy) and lowest ask price (sell) is called the bid-ask spread. The volume on both sides of the order book is called market depth and regularly serves to proxy the strength behind price movements. The `runner` module provides an example for this request method resulting in a plot of the accumulated amount on both sides of the order book by executing:

```
# Request the current order book
runner.Examples.order_books()
```

Table 1 shows a snapshot of the database for the exchange Coinbase and the currency pair ETH-BTC. By default, and common for many other cryptocurrency exchanges, Coinbase returns the first 50 entries from both sides of the order book.

Taking the midpoint between lowest ask and highest bid as spot price, 1.0 ETH can be exchanged to 0.031565 BTC, or reverse, 1.0 BTC equals 31.68066 ETH. The bid-ask spread, calculated as spread relative to the spot price, is 3.168 basis points (0.03168%). The volume on different positions of the order book is distributed very unevenly. A possible reason is that orders are aggregated at the same price level. For example, the high bid volume at positions three (82.12 ETH) and ten (87.47 ETH) in the order book do most likely not represent a single large order, but instead an aggregated sum of several orders.

Table 1: Order book snapshot from the database. The table is structured as follows: PairID represents an internal exchange currency pair (ETH-BTC), ID an unique identifier to distinguish between order book snapshots, Time is the request time in UTC+0 (translated from UNIX timestamp), Position the ordered position in the book, AsksAmount and BidsAmount the respective base volume. Order book data may be grouped by price.

PairID	ID	Time	Position	BidsAmount	BidsPrice	AsksPrice	AsksAmount
108	2964948970	2021-03-03 08:14:27.069	0	18.97379004	0.03156	0.03157	15.43867249
108	2964948970	2021-03-03 08:14:27.069	1	59.00069015	0.03155	0.03158	46.94196312
108	2964948970	2021-03-03 08:14:27.069	2	42.27195103	0.03154	0.03159	30.72162118
108	2964948970	2021-03-03 08:14:27.069	3	82.11266096	0.03153	0.0316	78.55653857
108	2964948970	2021-03-03 08:14:27.069	4	24.69691738	0.03152	0.03161	24.50587123
108	2964948970	2021-03-03 08:14:27.069	5	39.12828203	0.03151	0.03162	17.27955085
108	2964948970	2021-03-03 08:14:27.069	6	9.84934641	0.0315	0.03163	20.205
108	2964948970	2021-03-03 08:14:27.069	7	16.3	0.03149	0.03164	3.56145
108	2964948970	2021-03-03 08:14:27.069	8	12.00049334	0.03148	0.03165	42.40052371
108	2964948970	2021-03-03 08:14:27.069	9	1.662	0.03147	0.03166	44.891038
108	2964948970	2021-03-03 08:14:27.069	10	87.47106332	0.03146	0.03167	12.3134
108	2964948970	2021-03-03 08:14:27.069	11	2.5966	0.03145	0.03168	70.39308132
108	2964948970	2021-03-03 08:14:27.069	12	15.78915905	0.03144	0.03169	37.00327507
...	...	...	...	...	...	...	...
108	2964948970	2021-03-03 08:14:27.069059	49	1.12377398	0.03103	0.03206	19.55223117

To make the connection to transactions, we request every executed trade for the time around the order book snapshot as shown in the Table 2. Whenever a transaction is executed, due to a matching buy and sell order listed in the order book, the information is obtained, stored and can be requested by *open-crypto*. This commonly includes a timestamp, an unique identifier, the volume and price information from the order book, as well as the trade direction. The direction is defined as the party of the trade who takes liquidity out of the market, i.e. matches an existing offer in the order book and completes the transaction. The `runner` module provides an example for this request method resulting in a plot of the most recent 1,000 transactions from the exchange Coinbase:

```
# Request the most recent transactions
runner.Examples.trades()
```

Referring to the order book snapshot above, the transaction in row four of Table 2 was executed right after the order book snapshot. The transaction is labeled as 'sell', which conveys that the seller of the asset took an existing buy offer to complete the transaction. Recent trades data as in Table 2 contain important information. For one, if all trades are captured, the data is stated to be at tick-level, which represents the most valuable and highest resolution of pricing data. Furthermore, the sum of all trade quantities reflects the daily trading volume and is typically used to calculate the global turnover and volume-weighted price indexes.

Table 2: Database entries for the request method trades. The table is structured as follows: PairID represents an internal exchange currency pair (ETH-BTC), ID an unique identifier to distinguish between trades, Time represents the transaction time (translated from UNIX timestamp), Amount the trade size in base currency. <null> values are not returned by an exchange. For example, the first row represents the transaction of 0.001 ETH for a price of 0.03159 BTC per one unit ETH.

PairID	ID	Time	Amount	BestBid	BestAsk	Price	Direction
108	14232964	2021-03-03 08:14:19.548	0.001	<null>	<null>	0.03159	buy
108	14232965	2021-03-03 08:14:20.179	0.00058714	<null>	<null>	0.03159	buy
108	14232966	2021-03-03 08:14:26.241	0.06132751	<null>	<null>	0.03157	sell
108	14232967	2021-03-03 08:14:29.514	0.04035327	<null>	<null>	0.03159	sell
108	14232968	2021-03-03 08:14:35.775	0.00620771	<null>	<null>	0.03157	sell
108	14232969	2021-03-03 08:14:42.421	0.00310545	<null>	<null>	0.03157	sell
108	14232970	2021-03-03 08:14:43.396	0.653052	<null>	<null>	0.03157	sell
108	14232971	2021-03-03 08:14:43.396	0.30196308	<null>	<null>	0.03157	sell
108	14232972	2021-03-03 08:14:44.917	0.00953594	<null>	<null>	0.03157	sell
108	14232974	2021-03-03 08:14:51.550	0.94785805	<null>	<null>	0.03156	buy
108	14232973	2021-03-03 08:14:51.550	0.05214195	<null>	<null>	0.03156	buy
...	...	...	...	...	...	...	...

2.2 Requesting real-time data

After clarifying the context and terminology of the data, this section continues with an illustrative example. Specifically, we collect transaction data for the currency pair ETH-BTC from 64 different exchanges, from 2021-01-13 until 2021-02-13, to analyze differences in trading frequency on the various exchanges. The time between transactions is dependent on the liquidity of an exchange and can range from seconds up to days. The settings in the configuration file to retrieve the respective data are shown below:

```
general: <...>
jobs:
  TradeData:
    request_method: trades
    exchanges: all
    update_cp: false
    exclude: null
    currency_pairs: eth-btc
    first_currencies: null
    second_currencies: null
```

Table 3 displays the average time passed between trades. For each exchange, the first and last timestamp is selected, as well as the total amount of trades. Afterwards, we divide the total amount of seconds passed between both timestamps by the total number of trades. The metric therefore returns the average time between transactions. The next step groups similar exchanges into buckets. The constituents of each bucket are listed in the right column. We add the total

amount of listed currency pairs available on a specific exchange in brackets. The purpose is to yield a better impression about the importance of the various exchanges. For instance, the first row groups all exchanges which record transactions, on average, more often than once a second.

Table 3: Summary of all recorded trades for the currency pair ETH-BTC. Exchanges are aggregated into buckets depending on the average time passed between trades. The data period starts 2021-01-13 and ends 2021-02-13. Intervals are given in seconds, are left-exclusive and right-inclusive. The column 'Exchanges' lists all exchanges inside a bucket and is sorted by the total amount of currency pairs supported by an exchange (brackets). For better readability: 3,600 seconds equal one hour; 43,200 seconds equal 12 hours; and 86,400 seconds equal one day.

Interval in Seconds	Total	Exchanges
0 - 1	6	Binance(1,434), HitBtc(1,164), CoinDcx(786), LBank(405), Bitrue(253), WhiteBit(253)
1 - 5	14	Mxc(840), KuCoin(654), Okex(509), Hbtc(377), P2pb2b(358), Bitmart(296), Bitz(205), Coinbase(198), Coinbene(180), Mexo(116), Hydax(93), Bidesk(85), Xtheta-Global(81), Chiliz(54)
5 - 10	9	Bittrex(874), Bkex(718), LAToken(443), Tidex(382), Exrates(341), Folglory(326), Xt(279), Btc-Alpha(105), Bw(96)
10 - 30	9	Yobit(8,786), Coinex(450), Ftx(442), Bithumb(337), Exmo(189), Alterdice(123), Aax(115), Bitstamp(75), Bitfront(13)
30 - 60	3	Gateio(1,543), Bitmax(353), Bitso(25)
60 - 600	10	Crex24(1,670), Mercatox(764), Hoo(511), Bibox(353), Wazirx(259), Cexio(202), Bitvavo(151), Bitbay(195), Gemini(64), Btc-Turk(42)
600 - 3,600	5	Lykke(206), Zaif(32), Currency.com(31), Bleutrade(26), Buda(19)
3,600 - 43,200	3	Vindax(726), Crypto.com(153), Gopax(35)
43,200 - 86,400	1	Braziliex(81)
86,400 - max	1	Bisq(52)

From these exchanges with the highest turnover frequency, Binance lists the largest amount of currency pairs, in total 1,434. Exactly half of the exchanges record transactions within ten seconds. Within ten minutes, the vast majority (86%) records at least one trade. Note, however, that the results may vary depending on the popularity of a cryptocurrency in general or in a specific region. By far the most trades are received from Binance with around 13.5 million, leading to five trades per second on average. The distance to the exchange with the second most

recorded trades (HitBtc) is large, 3.88 million trades and, on average, 1.44 trades per second. The exchange with the lowest reported turnover is Bisq, with a total of 136 trades over the whole period. This results in one trade every 40 hours. The considerations above provide an idea on how reliable the data on a certain cryptocurrency from a certain exchange is.

3 Further Information

3.1 Exchanges and endpoints

The proposed package has a total of 106 exchanges connected. The following Table 5 counts the amount of implemented request methods. Obviously all exchanges support to query the underlying cryptocurrency pairs, roughly 100 support the real-time request methods, while 87 (65) offer to return limited (unlimited) historical data. It is, nevertheless, noteworthy, that cryptocurrency exchanges emerge and disappear frequently.

Table 5: Summary of all implemented exchanges and their offered request methods. Historic Rates (iterate) is a genuine subset of 'Historic Rates' and represents exchanges offering to iterate backwards in time, thus being able to receive all available candlesticks.

Currency-Pairs	106
Tickers	98
Recent Trades	99
Order-Books	101
Historical Rates	87
Historical Rates (iterate)	65
Exchanges Total:	106

3.2 Cross-software usability

Relational database systems are widely supported, moreover, the interoperability of different software is increasingly demanded. It is noteworthy that programming languages often offer interfaces for Python, for example using *reticulate* (Ushey et al., 2022) within R. However, this section shows how to access the data gathered with *open-crypto* from R and Matlab. To access the database from R the packages *RSQLite* (Müller et al. (2022)) and *DBI* (R Special Interest Group on Databases (R-SIG-DB) et al. (2021)) can be used. Following the official documentation, connecting to the illustrative database and executing a simple query is achieved by:

```
library(RSQLite, DBI)

con <- dbConnect(RSQLite::SQLite(), "Trades.db")
statm <- "SELECT time, price, amount FROM trades LIMIT(3)"
query <- dbSendQuery(con, statm)
dbFetch(query)
```

```
                       time   price     amount
1 2021-03-03 08:14:35.775000 0.03157 0.00620771
2 2021-03-03 08:14:29.514000 0.03159 0.04035327
3 2021-03-03 08:14:26.241000 0.03157 0.06132751
```

The first command defines a variable to handle the database connection, followed by a simple query in standard SQL notation. The example returns time, price and amount of the first three database entries from the historical data. R users can furthermore make use of the package *dbplyr* (Wickham et al., 2021), the database backend for the popular package *dplyr* (Wickham et al., 2022), which implements, to some extent, object-relational mapping (ORM). *dbplyr* then automatically translates function calls into SQL notation (for more information, visit: https://dbplyr.tidyverse.org/).
The Matlab syntax is as well straightforward. Note that users of Matlab need the *database* toolbox. However, there are open-source alternatives provided by the community, e.g. *mklite* (https://sourceforge.net/projects/mksqlite/files/).

```
>>con = sqlite("ExampleDB.db");
>>statm = "SELECT time, price, amount FROM trades LIMIT(3)";
>>result = fetch(con, statm);
```

3.3 Database Connection

During the illustrations we chose *SQLite* as database. This system is server-less and well equipped for most applications. However, users may wish to make use of one of the other mentioned database management systems. Accordingly, in order to connect to *MariaDB* or *MySQL*, small changes need to be made in the configuration file. Note that we only use default values for the purpose of demonstration, which can be retrieved from the official documentations. Furthermore, users need to ensure to install the database system and the client beforehand.

```
general:
  database:
    sqltype: mariadb  # or mysql
    client: pymysql
    user_name: root
    password: *******
    host: localhost
    port: 3306
    db_name: ExampleDB
  operation_settings: <..>
  jobs: <...>
```

Slightly different, when using *PostgreSQL*:

```
general:
  database:
    sqltype: postgresql
    client: psycopg2
    user_name: postgres
    password: *******
    host: localhost
    port: 5432
    db_name: ExampleDB
```

```
  operation_settings: <..>
  jobs: <...>
```

Similar to the *SQLBrowser*, server-based database management systems do have visualization/administration tools available. Most popular for *PostgreSQL* is the open-source platform *pgAdmin* (https://www.pgadmin.org/). *MySQL* and *MariaDB* can be connected by the open-source tool *HeidiSQL* (https://www.heidisql.com/).

References

Müller K, Wickham H, James DA, Falcon S (2022). *RSQLite: SQLite Interface for R*. R package version 2.2.14.

R Special Interest Group on Databases (R-SIG-DB), Wickham H, Müller K (2021). *DBI: R Database Interface*. R package version 1.1.2.

Ushey K, Allaire J, Tang Y (2022). *reticulate: Interface to 'Python'*. R package version 1.25.

Wickham H, François R, Henry L, Müller K (2022). *dplyr: A Grammar of Data Manipulation*. R package version 1.0.9.

Wickham H, Girlich M, Ruiz E (2021). *dbplyr: A 'dplyr' Back End for Databases*. R package version 2.1.1.