Journal of Data Science 21 (1), 127–144
January 2023
DOI: 10.6339/22-JDS1059
Computing in Data Science

The Python Package open-crypto: A Cryptocurrency Data Collector

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

[1]*Chair of Finance, University of Bremen, Germany*

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

Abstract

This paper introduces the package *open-crypto* for free-of-charge and systematic cryptocurrency data collecting. The package supports several methods to request (1) static data, (2) real-time data and (3) historical data. It allows to retrieve data from over 100 of the most popular and liquid exchanges world-wide. New exchanges can easily be added with the help of provided templates or updated with build-in functions from the project repository. The package is available on GitHub and the Python package index (PyPi). The data is stored in a relational SQL database and therefore accessible from many different programming languages. We provide a hands-on and illustrations for each data type, explanations on the received data and also demonstrate the usability from R and Matlab. Academic research heavily relies on costly or confidential data, however, open data projects are becoming increasingly important. This project is mainly motivated to contribute to openly accessible software and free data in the cryptocurrency markets to improve transparency and reproducibility in research and any other disciplines.

Keywords *cryptocurrency; open data; Python*

1 Introduction

Cryptocurrencies are a fairly new and rapidly developing market. The interest of researchers from finance, economics, law, politics, computer science and mathematics, among others, is growing and gaining importance. Moreover, cryptocurrencies as traded financial products are beyond a mere theoretical construct and are attracting the interest of investors, public policy makers, and regulators due to their success and popularity.

Cryptocurrencies are traded globally on mostly non-certified or non-registered exchanges with limited or no regulation from official authorities. From an economic perspective, the cryptocurrency markets still lack efficiency (Makarov and Schoar, 2020), are subject to fraud and manipulation (Hougan et al., 2019; Aloosh and Li, 2019; Cong et al., 2019; Pennec et al., 2021, among others), and their impact on the economy to date is mostly unexplored. Beyond Bitcoin and its most famous followers (Ethereum, Ripple, Litecoin, among others) exist over 10,000 cryptocurrencies (www.coinmarketcap.com) with as many projects and ideas. Only a fraction is actually developed as digital currency to serve as commonly accepted basis for exchanging goods and services. Most originate from projects and start-ups that use ICOs (Initial Coin Offerings) or similar methods to collect venture capital. In return, these coins grant specific privileges (utility tokens), represent assets (asset-backed tokens), or serve as investments (security tokens).

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

Received April 14, 2022; Accepted July 3, 2022

This project aims to address the data issue persistent in cryptocurrencies and is inspired by similar ideas across disciplines. Gewin (2016) addresses the shift in science to publicly available data repositories, in order to make findings reproducible. Reichman et al. (2011) call for federated data repositories and to overcome the inadequacy of rewards for sharing data in ecology. Iacus (2015) publishes a practical guide for data collection using R (R Core Team, 2020). Similar implementations to *open-crypto* already exist in other disciplines, for example, Szöcs et al. (2020) introduce the R package *webchem* to collect publicly available chemical data. In cryptocurrencies, only some aggregated information are easily accessible. For instance, the Bitcoin price published to the broad public is the (volume-weighted) average of several exchanges. The global turnover volume is the respective summation of all trade quantities. These aggregated data on cryptocurrencies are collected by well-known platforms like Coinmarketcap, Kaiko, Cryptocompare, Coingecko and Coinpaprika. Aggregated data can be retrieved from the above mentioned platforms while non-aggregated data must be retrieved from the exchanges' application programming interface (API) itself. The former keeps especially technical burdens outsourced, but has major disadvantages. Platforms can change their terms of use, (and) require expensive fees as introduced by Coinmarketcap (July 2018), data preprocessing can be intransparent, discontinuous and vulnerable to biases (e.g. survivorship bias) or data can be simply unavailable due to aggregation. On the other hand, exchanges emerge or go offline frequently and APIs are non-standardized in the request and response format. Consequently, there is a lack of customized, not preprocessed and free-of-charge data gathering.

We offer the Python (van Rossum and Drake, 2009) package *open-crypto* to fill this gap by directly integrating the exchanges' and platforms' public APIs into our program. We implement endpoints for (1) static currency pairs information like full-names or minimum trade quantities, (2) real-time market-data, in particular price quotations, trading volume and order book information, and (3) historical data for trading day summaries from over 100 of the most liquid and popular exchanges and platforms. Compared to platforms, exchanges allow to retrieve dis-aggregated and high-frequency data. The data is queried, extracted, and formatted automatically, and written into a relational database of choice. Supported are the most common database management systems included in the popular object-relational-mapper (orm) *SQLAlchemy* (Bayer, 2012), such as *SQLite*, *MySQL*, *MariaDB* and *PostgreSQL*.

The program is intuitive to use, does not require any specific knowledge and is highly customizable and expandable. New exchanges, platforms and API endpoints can be implemented with provided templates and manuals. The code is written in Python, however data can be retrieved from the database with any software capable to connect SQL (including e.g. R, Matlab, Stata or SPSS) or written into a csv-file. The open-source project comes with a standard *GNU General Public License (GPL)* and is offered as both a GitHub repository for development and a Python module, called *open-crypto*, for quick installation using Python's package installer *pip*. The data collector addresses the call of Gewin (2016) and Reichman et al. (2011), among others, for FAIR data principles in research to support the findability, accessibility, interoperability, and reusability of data (www.go-fair.org). It is an important tool for any type of application in need of original, free-of-charge, and high-quality cryptocurrency data and lines up with existing tools from other disciplines like the project *webchem*. We hope to support independent and transparent research.

Our paper is structured as follows: Section 2 describes the installation and provides a hands-on. Furthermore, we give detailed illustrations on the available data and request methods in Section 3. The Section 4 discusses different data export methods, while Section 5 summarizes and concludes. Additional notes on the installation process, a troubleshooting list, cross-platform

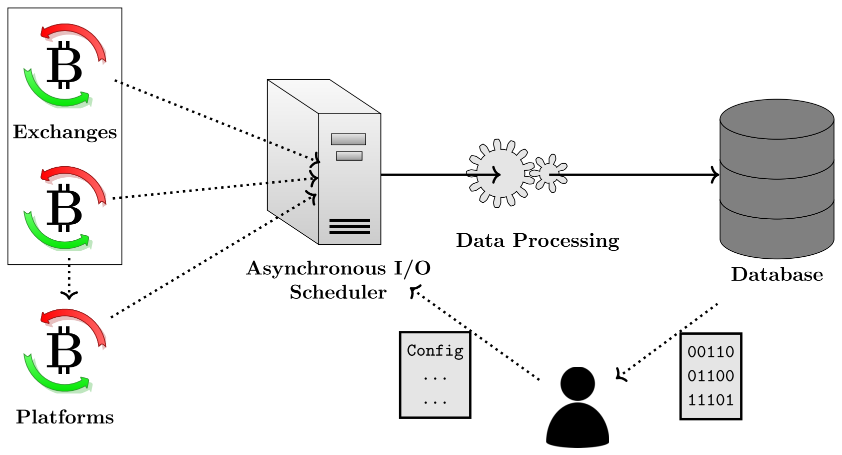

Figure 1: Workflow sketch of the program. From left to right: Cryptocurrency exchanges and platforms offer public available data over API endpoints. Platforms aggregate data available from exchanges. Every endpoint contains specific information, which can be controlled by the program. Requests can be scheduled asynchronously to increase efficiency. After requesting and retrieving, data is extracted, formatted and persisted into a database. The user controls the program by applying a configuration file.

interoperability from R and Matlab, and some advanced configurations are presented in the online supplement.

2 Installation and Hands-on

For a general overview, Figure 1 displays the work-flow of *open-crypto* and user interactions. Users can retrieve data from exchanges and/or platforms.

Cryptocurrency exchanges are market places, centralized or decentralized, for trading various pairs of cryptocurrencies. Several hundreds of exchanges around the globe exist and are very heterogeneous in architecture, liquidity, and reliability. Exchanges offer API endpoints for both market data retrieval and trading purposes. While trading obviously requires authentication, market data are often publicly available. We use these API endpoints to request recent and historical pricing information, exclusively from public Representational State Transfer (REST) APIs. However, data requirements may not be fulfilled by exchanges alone. Some additional information, like the global market capitalization of a cryptocurrency, can not be obtained from a single exchange. We therefore developed *open-crypto* to be applicable to both data sources, exchanges and platforms. This enables users to also retrieve aggregated cryptocurrency information. Referring to Figure 1, users take control by applying a configuration file. This defines the exchange(s) or platform(s), currency pair(s) of interest, and the data type (static, historical or real-time). Detailed instructions on creating the configuration file are provided in the subsequent Section 3. The file is read and executed in an asynchronous input/output (I/O) scheduler, which performs requests across exchanges concurrently by making use of the popular libraries *asyncio* (van Rossum, 2012) and *aiohttp* (Kim and Svetlov, 2020). After retrieval and several processing steps, the data is written into a relational (SQL) database system to which the user has access via *open-crypto* or any other software. The following of this section will cover the installation, discuss implemented API endpoints and the retrieved data from exchanges and platforms. This paper then continues with several illustrative examples in Section 3.

2.1 Installation

The program is uploaded to the Python Package Index (PyPI) and GitHub. For regular installation, use `pip install open-crypto`. This downloads and installs the program including all dependencies. For development, clone the GitHub repository (https://github.com/SteffenGue/open-crypto) and run the setup file within the directory. Further details on the installation process can be found in the online supplement. Python stores all modules in its site-package folder. However, as some files need to be modifiable to run the program, the current working directory (cwd) is used for all configuration files, exchange mappings, and the database(s). Furthermore, all relevant functionality is gathered in the wrapper module `runner`.

```
# Import the runner module from the package
from open_crypto import runner
```

More details on the documentation of all methods can be printed via Python's help function `help(runner)`. An abbreviated description yields the following:

- `update_maps`: Download the latest exchange mappings from the GitHub repository.
- `exchanges_and_methods`: List all exchanges and their implemented request methods.
- `get_session`: Return an open connection to the database.
- `get_config_template`: Create a new (blank) configuration file.
- `export`: Export data into a csv- or hdf-file.
- `run`: Start the program.
- `Examples`: Execute several example scripts.

To initialize the program in the first place, ensure that all resource files are located within the current working directory, and updated to the latest release. This can be achieved by using the following method:

```
# Prepare the program
runner.update_maps()
```

The command downloads the most recent exchange mappings from the GitHub repository and creates a directory named 'resources' within the cwd. The folder contains all important files to control the program. When set up and started, *open-crypto* imports and executes a configuration file which specifies the request. The introduction of the latter is covered in detail in Section 3. Meanwhile, the following Section 2.2 covers the data retrieved from exchanges and platforms. Every type of data is typically offered via separate API endpoints, which are referred to as request methods. Users of *open-crypto* specify the request method in the configuration along with the data source(s) and currency pair(s) of interest.

2.2 Data and API-endpoints

This part presents possible configurations before executing exemplary requests in Section 3. Table 1 outlines that *open-crypto* allows to retrieve three categories of information (static, historic, real-time) from two sources (exchanges and platforms). Whether certain information is available from a specific source is marked with a cross. The first line (Data) of Table 1 specifies the data

Table 1: Overview of all implemented request endpoints for both types of data providers.

	Static	Real-Time			Historical
Data	Currency Pairs	Tickers	Trades	Order Books	Historic Rates
Request Method	currency_pairs	tickers	trades	order_books	historic_rates
Exchanges	X	X	X	X	X
Platforms	X				X

type, while the second line (Request Method) names the request methods that are to be set in the configuration file. Each exchange API has several endpoints, that define the underlying data. Typically, exchanges offer (meta-) data for the underlying currency pairs (Static); real-time market data on transactions (Real-Time); and summarizing information on past trading periods (Historical). Up to date, *open-crypto* supports the public API endpoints from over 100 of the largest and most liquid exchanges worldwide. Platforms on the other hand offer aggregated market data like the market capitalization and global turnover volume of each cryptocurrency. This information is not provided by exchanges, which makes platforms an important supplement for a global perspective on the cryptocurrency market. We focus on Coingecko and Coinpaprika, as they are, as of the time of this study, free-of-charge and well known in the community.

For an overview of all exchanges and their supported request methods, the following command can be called:

```
# List available exchanges and their supported request methods
df = runner.exchanges_and_methods()
df.index.name = "exchanges"
# Print the first five exchanges
df.head()
```

```
             currency_pairs  historic_rates  trades  order_books  tickers
exchanges
50x                    True            True    True         True    False
aax                    True           False    True         True    False
alterdice              True            True    True         True     True
ascendex               True            True    True         True     True
b2bx                   True            True    True         True     True
```

The `static` endpoint contains information about the listed cryptocurrency pairs. Furthermore, many exchanges provide basic parameters about the currencies themselves, including transaction fees and associated full names. We provide a unique identifier for each exchange currency pair combination because cryptocurrencies do not necessarily share the same ticker across exchanges. The available currency pairs for each exchange are written into the database and imperative for all further requests. `Real-time` data subsumes most of the request methods and includes tickers, recent trades and order book data. `Tickers` returns the latest price, frequently including the bid-ask spread and rolling 24-hours turnover volume. `Trades` contains price, quantity, timestamp, direction (determined by the agent who adds volume to (maker) and who takes volume from (taker) the market) and some trade identifier from the most recent transactions. This request method is most suitable to obtain tick-level data for an exchange currency pair. `Order books` provides a snapshot of all open buy and sell orders at one point in time. The gap between the highest bid (buy) and lowest ask (sell) order is called bid-ask spread. The total volume on both sides of the order book is called market depth and is often interpreted as an indicator for price movements as it shows the ability to absorb the price impact of larger trade quantities. `Historic rates` summarizes a fixed period of trading by returning open-high-low-close prices and the daily turnover volume (OHLCV). Many exchanges offer to modify the frequency of historical data. Therefore, *open-crypto* supports requesting historical data from one-minute up to monthly candles. We illustrate the possibility for users to gather high-frequency minute candles in Section 3.2. One fundamental difference to the stock market is the non-stop trading as cryptocurrency exchanges never close. This leaves questions about the closing time on different exchanges across time-zones. The consensus is to use 23:59:59 UTC as the daily closing time.

3 Illustrations

In this Section, we provide examples on how to request static, real-time or historical cryptocurrency data using *open-crypto*. This is achieved by using configuration files that can easily be adapted by users of *open-crypto*. All configuration files are stored within the current working directory. The name can be arbitrarily chosen, and goes as an input argument when starting the program. Before we turn to the examples, we describe which settings users can make in the configuration file. It contains three sections, `database`, `operation_settings` and `jobs`. A full configuration file is available in the GitHub repository (./resources/templates/request_template.yaml). Each is separately explained in the following. To save space, we unfold only the parts of interest and mark the others with `<...>`.

• `database` – Firstly, the database is specified. Users can choose from a variety of embedded and server-based database management systems supported by *SQLAlchemy*, namely *SQLite*, *MariaDB*, *MySQL* or *PostgreSQL*. The client defines the adapter used by Python to establish a connection. These are not part of the package dependencies but are automatically installed upon usage. The database name (`db_name`) is free to choose, and the only necessary parameter for the (embedded) server-less system *SQLite*. Variables for server-client based systems, especially host and port, can be obtained from the respective documentation. We choose *SQLite* for all illustrations provided in what follows in this section. We demonstrate the connection to the other database systems in the online supplement.

```yaml
general:
  database:
    sqltype:    # sqlite,    mariadb,    mysql,     postgresql
    client:     # sqlite3,   pymysql,    pymysql,   psycopg2
    user_name:
    password:
    host:
    port:
    db_name: ExampleName  # Arbitrarily chosen name
  operation_settings: <..>
jobs: <...>
```

With the database specified, *open-crypto* is able to store the data. The next step defines how the data should be collected.

• `operation_settings` – Secondly, in the operational settings, several variables can be set to change the behaviour of *open-crypto*. The `frequency` is the time between two runs. It has to be specified in minutes (e.g. 0.1 is equal to six seconds) when real-time data is requested. If a single run takes longer than the target frequency, it is restarted immediately after completion but not terminated intermittently. A value of zero leaves no waiting time in-between runs. If instead `once` (e.g. to retrieve static or historical data) is passed, the program terminates automatically after a single run. The key `timeout` defines the maximum time (in seconds) *open-crypto* waits for an exchange to respond. If the value is exceeded, the exchange is neglected for this run but requested again thereafter. Further, `enable_logging` allows to write several status statements and/or intercepted error messages in a separate log-file, while `asynchronously` sets *open-crypto* to either request exchange currency pairs in parallel or iteratively. In addition, `interval` defines the granularity of historical candles and is only relevant for this request method. Accepted values are `minutes`, `hours`, `days`, `weeks`, `months`. The default value is `days`.

```
general:
  database: <...>
  operation_settings:
    frequency: once        # or any number in minutes
    timeout: 10
    interval: days         # minutes, hours, days, weeks, months
    enable_logging: true
    asynchronously: true
jobs: <...>
```

After this part of the configuration file, *open-crypto* knows how to request the data and where to store them. The last task for the user is to define what to request.

• `jobs` – Thirdly, specifications about the request itself follow. Users have to provide the name of the `request_method`, exchange(s) and currency pair(s) to consider. For every job, an arbitrary name can be selected and replaces the field `Job_Name`. The request methods allowed are specified in Table 1. To update the currency pairs of an exchange, `update_cp` can be enabled. This is recommended as new currencies are listed and unlisted frequently. The specification of exchanges, single cryptocurrencies and currency pairs allow for multiple values, which need to be comma-separated. Currency pairs can either be specified directly or filtered by base and/or quote currencies in order to reduce typing efforts. Furthermore, exchanges and currency pairs can be replaced by the key `all` which will take all objects available. Several illustrations on this are provided in the following Sections 3.2 for historical data, and the online supplement for real-time data.

```
general: <...>
jobs:
  Job_Name:
    request_method:
    exchanges:             # exchange1, exchange2
    update_cp: false
    exclude: null          # exclude a specific exchange or platform
    currency_pairs: btc-usd, eth-usd
    first_currencies: null
    second_currencies: null
```

For every data type introduced in Table 1 (static, real-time and historical), the remainder of this section and the online supplement provide intuitive illustrations. Section 3.1 (static) gives an idea of the amount and distribution of cryptocurrencies across exchanges, Section 3.2 (historical) requests time series from the data provider Coingecko and high-frequency one-minute candles from all supported exchanges to get an idea of the ex-post data coverage and the differences between both sources. Finally, the online supplement contains (real-time) exemplary analysis regarding the trading behaviour of exchanges as such data cannot be retrieved from platforms. Example scripts are embedded for each of the following request methods and are designed to reproduce the results, whenever possible.

3.1 Static Information

The request method `currency_pairs` allows to retrieve all listed pairs from one or several exchanges. The following code block shows the configuration to gather static data from all exchanges. The important fields in the configuration file are `request_method`, and `exchanges`, where the parameter to take all available exchanges is applied. The `Job_Name`, called `Static` in this example, is arbitrarily chosen.

```
general:
  database: <...>
  operation_settings: <...>
jobs:
  Static:
    request_method: currency_pairs
    exchanges: all
    update_cp: false
    exclude: null
    currency_pairs: null
    first_currencies: null
    second_currencies: null
```

The script which runs the above configuration file and creates the following Figure 2 can be executed by the following line of code. Note that requesting all currency pairs from over 100 exchanges may take several minutes to complete. All configuration files are available in the GitHub repository (./resources/configs/user_configs/examples), where the files follow the naming convention of the methods in this paper, along with the scripts which create the plots (./examples.py):

```
# Request all exchange currency pairs and plot a histogram
runner.Examples.static()
```

Figure 2 plots the distribution of currency pairs across exchanges. Aggregated over more than 100 exchanges, this endpoint returns about 6,300 tickers and 36,700 exchange currency pairs, excluding platforms. For comparison, the platform Coingecko lists 7,169 (crypto-)currencies from

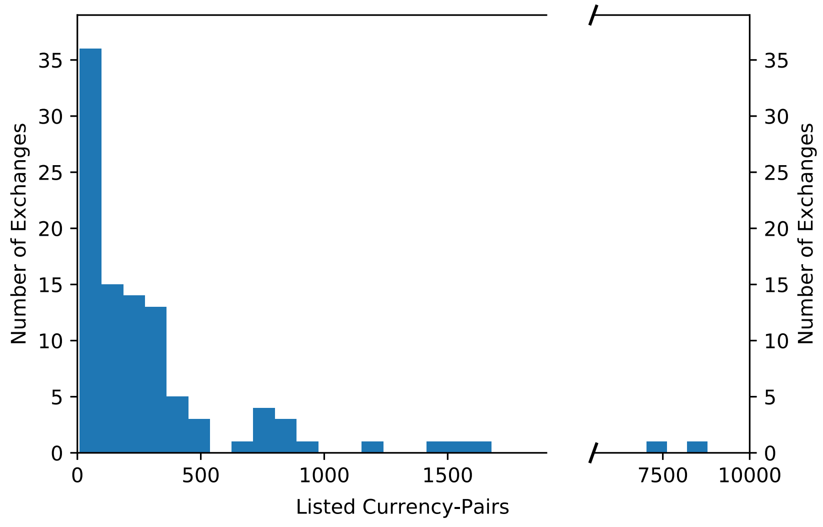

Figure 2: Histogram of listed currency pairs across exchanges. The x-axis is discontinuous for illustrative purposes. The highest values are retrieved from the exchanges/platforms YoBit (8,786), Coingecko (7,169), Crex24 (1,670), GateIO (1,543) and Binance (1,256). Figure and data as of 2021-05-12.

around 460 exchanges at the same point in time (all information as of 2021-05-12). The low difference of cryptocurrencies indicates that the implemented exchanges of *open-crypto* cover the vast majority of the (liquid) cryptocurrency market. To provide a better understanding about the distribution, Figure 2 plots the amount of currency pairs across exchanges. Most of the exchanges list under 500 currency pairs, with only a few exceptions reaching up to several thousand tradable asset pairs. Among those are large exchanges like Binance and HitBtc, but as well the platform Coingecko with 7,169 currency pairs, which are all quoted against the US-Dollar. Differently, YoBit lists 8,786 trading pairs but only 1,466 single currencies, due to multiple quotations. Most commonly, exchanges quote the cryptocurrencies against Tether with a share of 29.16%, Bitcoin (24.30%), Ethereum (14.96%), US-Dollar (8.08%) and Waves (4.12%) of all quotations in our data.

The database entries themselves can be inspected in several ways, as further demonstrated in section 4. Within Python, or any other SQL supporting language, the database is queried the following way:

```python
# Retrieve an open database session
session = runner.get_session("examples/static")

# Query the first exchange entry in the database
exchange_obj = session.query(runner.Exchange).first()
print(exchange_obj)
```

```
#1: 50X, Active: True
```

This returns the first exchange in the database. The following request shows how to retrieve the currency pairs related to the first exchange in our database:

```
# Query currency pairs related to the exchange object
query=session.query(runner.ExchangeCurrencyPair).filter(
  runner.ExchangeCurrencyPair.exchange_id == exchange_obj.id
).limit(5)

for pair in query:
  print(pair)
```

```
#1: 50X(1), XLM(1)-USDT(2)
#2: 50X(1), CRV(3)-USDT(2)
#3: 50X(1), AIV(4)-USDT(2)
#4: 50X(1), A2A(5)-USDT(2)
#5: 50X(1), LINK(6)-USDT(2)
```

For a more convenient way to look at the received data itself, we also recommend the use of third-party programs like *SQLBrowser* (https://sqlitebrowser.org). This is a quick and simple way to evaluate the received data. In particular, users can inspect the listed currency pairs of exchanges in order to perform subsequent individual requests.

With all the information about the listed currency pairs on every exchange and platform available in the database, users can specify the exchange and currency pairs in the configuration file accordingly. This step is demonstrated in the following subsections. Exemplary requests are performed for historical and real-time data for a variety of currency pairs, in Section 3.2 and the online supplemental, respectively.

3.2 Historical Data

This subsection is divided into three parts. Firstly, we query several cryptocurrencies from the platform Coingecko and plot the price, market capitalization and total supply. Secondly, we query historical data for major cryptocurrencies from all exchanges. Thirdly, we request data for the major currency pair ETH-BTC to compare the available data frequency of the various exchanges, in the following Section 3.2.3.

3.2.1 Platform Data

For the first part, we configure the program to retrieve historical data for Bitcoin, Ethereum and Dogecoin from Coingecko. There is one major difference in the configuration between exchanges and the platforms: Coingecko (and Coinpaprika) use full names as identifiers (i.e. Bitcoin or Btc-Bitcoin), whereas exchanges typically only provide the ticker (i.e. BTC). We do not map full names with exchange tickers, which allows for keeping data from both sources separated in the database. Users can query the currency pairs table, as shown in the previous section 3.1, to retrieve the correct spelling. Consequently, the configuration is as follows:

```
general:
  database: <...>
  operation_settings: <...>
jobs:
  Historical:
    request_method: historic_rates
    exchanges: coingecko
    update_cp: false
    exclude: null
    currency_pairs: null
    first_currencies: bitcoin, ethereum, dogecoin
    second_currencies: usd
```

The fields `request_method` and `exchanges` are set to retrieve historical data from the respective data source. The pairs of interest are provided under the keys `first_currencies` and `second_currencies`. It would yield the same result if instead the pairs are defined in the field `currency_pairs`. Users can perform this request and create the following Figure 3 (with more recent data) by executing the following script:

```
# Request time series data from a platform
runner.Examples.platforms()
```

To inspect the received data, users can choose different approaches in order to establish a link to the database, in particular between intern *open-crypto* functionalities and third party tools like the *SQLBrowser*. These approaches are described in detail in Section 4. For simplicity and illustrative purposes, we stick with console output. Using the open database session from Section 3.1, the following request the retrieved historical candles:

```
import pandas as pd  # Load the data directly into a dataframe

query = session.query(runner.HistoricRate.time,
                      runner.HistoricRate.close,
                      runner.HistoricRate.volume,
                      runner.HistoricRate.market_cap)
df = pd.read_sql(query.statement, con=session.bind, index_col="time")
print(df.head())
```

```
                                  close        volume    market_cap
time
2021-02-21 23:59:59+00:00  56377.633478  7.735179e+10  1.050556e+12
2021-02-20 23:59:59+00:00  56038.727759  7.139724e+10  1.044190e+12
2021-02-19 23:59:59+00:00  51733.075539  5.848651e+10  9.639114e+11
2021-02-18 23:59:59+00:00  52143.678845  6.847874e+10  9.715166e+11
2021-02-17 23:59:59+00:00  49238.136907  6.414113e+10  9.173397e+11
```

Each row of the database consists of the closing price, daily turnover volume, and the market-capitalization. The retrieved data in our database starts at 2013-04-28, volume is added on 2013-12-28.

The following Figure 3, created by `runner.Examples.platforms()`, shows the data for Bitcoin. The top of Figure 3 contains the price series, which is the average of daily closing prices from exchanges around the world. Beneath, the daily turnover volume, scaled to billion US-Dollars is shown. We also calculate the total amount of Bitcoin in existence at each point in time

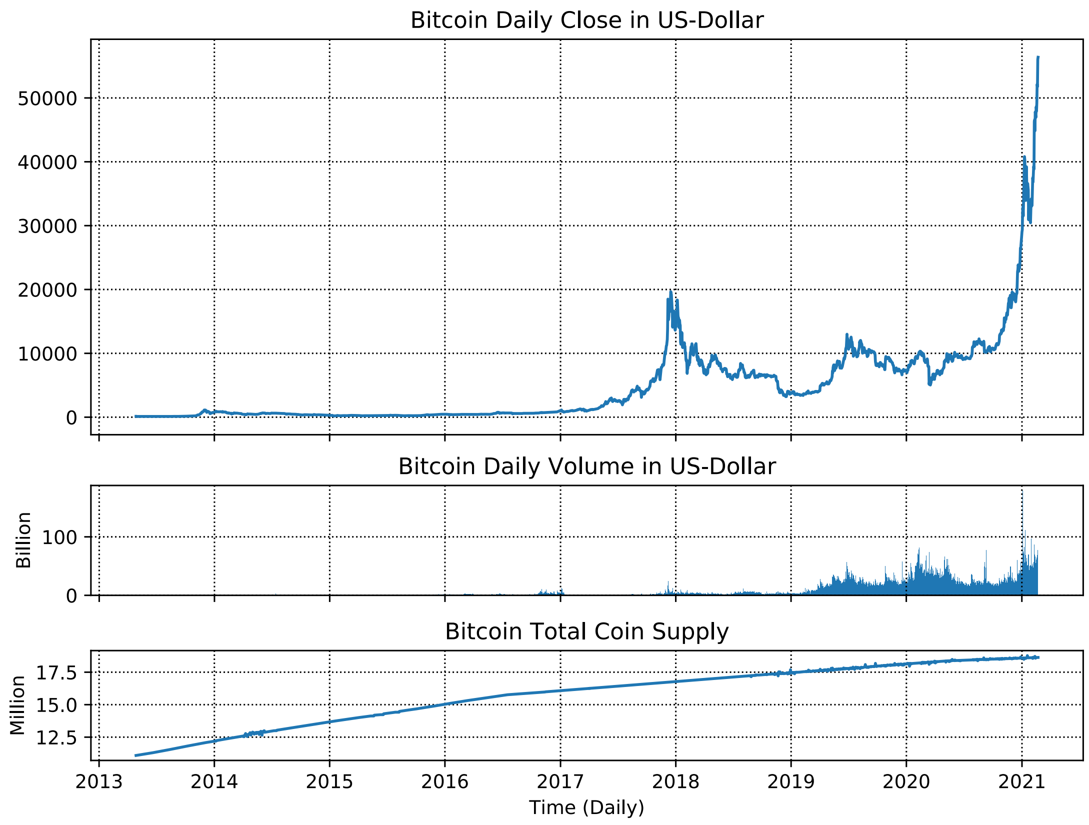

Figure 3: Bitcoin price, volume and total coin supply. Closing price and volume are in units of US-Dollar. The total coin supply is calculated by dividing the daily market capitalization by the daily closing price.

by dividing the market capitalization by the price. The line for Bitcoin shows a steady increase with breakpoints in mid 2016 and 2020. This is due to so-called halvings, which decrease the amount of newly mined coins per year by half. Halvings take place every 210,000 blocks. Bitcoin is designed to increase the total float by a (relatively) constant amount until the maximum of 21 million coins is reached.

Figure 3 illustrates the advantages of platform data, namely the availability of the aggregated (global) turnover volume and market capitalization. The data series are necessarily aggregated from several exchanges and cannot be decomposed. Dis-aggregated or high-frequency data (historical data) or trade and order book data (real-time data) requires to request exchanges directly. We will therefore deal with exchange data in what follows.

3.2.2 Exchange Data

Leaving the aggregated view on the cryptocurrency market and focusing on the exchanges themselves, *open-crypto* offers a large variety of data. The following part of this section investigates the ex-post coverage of cryptocurrencies by exchanges. For illustrative purposes, we focus on ten of the largest cryptocurrencies as representatives of the liquid market, namely BTC, ETH, LTC, XRP, ADA, DOGE, LINK, BCH, XLM, and ATOM, all quoted against USDT (Tether) or USD (US-Dollar). As Tether is the digital pendant to the US-Dollar, and therefore equal in

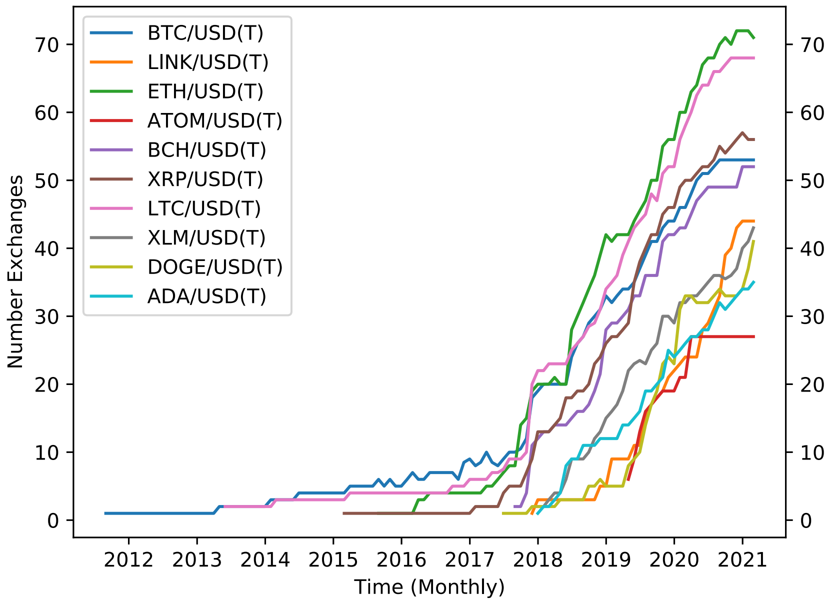

Figure 4: Monthly (median) amount of (cross-sectional) price notations per currency pair. Price quotations from all exchanges and for every currency pair are counted daily and are down-sampled to the monthly median. Tether and the US-Dollar are treated equally. The exchanges are summed up.

price by construction, we do not distinguish between both in the following. The setting below configures *open-crypto* to request daily historical data for the mentioned currency pairs from all available exchanges.

```
general:
  database: <...>
  operation_settings: <...>
jobs:
  DailyCandles:
    request_method: historic_rates
    exchanges: all
    update_cp: false
    exclude: null
    currency_pairs: null
    first_currencies: btc, eth, ltc, xpr, ada, doge, link, bch, xml, atom
    second_currencies: usd, usdt
```

Users can perform this request and replicate the following plot (with more recent data) by executing the following script (this request can take a long time to complete. It is therefore provided as cut-down version here to run in several minutes):

```
runner.Examples.exchange_listings()
```

Figure 4 shows the number of exchanges on which a certain cryptocurrency is available. The x-axis represents time and the y-axis the number of exchanges providing pricing information. For example, the plot shows that the number of exchanges on which the currency pair ETH/USD(T) is available increases from 20 in 2018 to 70 in 2021.

Bitcoin has the longest price history on exchanges (reaching back into 2011), far longer than the time series from the platform Coingecko as shown in Figure 3, as the whitepaper was already published in 2008 (Nakamoto, 2008). Starting with the end of 2017, many new cryptocurrencies and exchanges emerged, which accelerated the listings significantly. Ethereum (ETH) and Litecoin (LTC) are by far listed and traded on the most exchanges today, followed by Ripple (XRP), Bitcoin (BTC) and Bitcoin Cash (BCH).

3.2.3 Data Frequency

The previous paragraph demonstrated that *open-crypto* is capable to retrieve a comprehensive historical-data feed. However, we only focused on a daily frequency. We do now turn our focus to high-frequency (intra-day) data. As exchanges may restrict data availability in different ways, we address the question of historical-data quality, in terms of completeness. For illustrative purposes, we limit to a single but very commonly traded currency pair, namely ETH-BTC. The following configuration file sets `interval: minutes`. The task is to query historical ETH-BTC data (on a minute basis) from all exchanges available:

```yaml
general:
  database: <...>
  operation_settings:
    frequency: once
    timeout: 10
    interval: minutes
    update_cp: false
    enable_logging: true
    asynchronously: true

jobs:
  MinuteCandles:
    request_method: historic_rates
    exchanges: all
    update_cp: false
    exclude: null
    currency_pairs: eth-btc
    first_currencies: null
```

This request can take several hours to complete, as exchanges do not return the whole data at once, but only batches, therefore *open-crypto* has to request data iteratively. Again, the example script is provided as a cut-down version with only three exchanges and a timer variable to terminate the program (defined in seconds):

```python
# Request historical one minute candles
runner.Examples.minute_candles()
```

Table 2 summarizes the requested data from the 15 exchanges providing the most data points. Exchanges are sorted by the total amount of received observations. The earliest time-stamp dates back to 2015-08-14. We note that Ethereum was published on 2015-07-30, only

Table 2: Summary statistics of the top 15 exchanges for historical data coverage. Minutes is defined as the time delta between start/end in minutes. Entries counts the total number of 'OHLCV'-rows retrieved. The currency pair requested is ETH/BTC, with minute candles. A total of 53 exchanges returned data for this pair, including lower frequencies. The latest timestamps are from 2021-02-04. For reasons of comparability: one day is equal to 1,440 minutes. All timestamps are in UTC.

Exchanges	Start (UTC±0)	Time Delta (Min)	No. of Entries	%
Bittrex	2015-08-14 09:06:00	2,880,893	2,880,893	100.00
HitBTC	2015-12-08 10:53:00	2,714,660	2,090,747	77.02
Binance	2017-07-14 04:01:00	1,874,113	1,708,734	91.18
LBank	2017-09-29 01:48:00	1,763,364	1,708,585	96.89
KuCoin	2017-09-27 02:46:00	1,766,187	1,519,188	86.02
Bitfinex	2016-03-09 16:10:00	2,581,863	1,506,103	58.33
Currency.com	2017-05-01 00:00:00	1,980,913	1,474,454	74.43
Maicoin	2018-05-31 10:21:00	1,411,491	1,347,952	95.50
Oceanex	2018-11-30 03:01:00	1,148,411	1,138,098	99.10
Bitz	2018-06-30 17:52:00	1,367,841	1,038,316	75.91
Upbit	2017-08-20 08:33:00	1,820,554	873,569	47.98
Digifinex	2019-07-29 16:00:00	800,593	733,838	91.66
Bitstamp	2019-11-07 23:16:00	654,717	603,000	92.10
Nominex	2019-10-09 13:37:00	697,055	574,412	82.41
Ftx	2020-05-14 09:11:00	383,402	376,063	98.09

two weeks before. Calculating the difference (in minutes) between this date and the most recent date available results in about 2.88 million possible data points. We find that the most data points are available on Bittrex. In total, only two exchanges provide data points for less than 70% of the maximum time period, whereas most exchanges reach coverages above 90%. Reasons for missing data are multifaceted: exchanges may simply delete redundant candles with no changes and therefore no additional information to the previous period, can undergo maintenance or experience temporary blackouts. From over 100 exchanges, the request returned data from a total of 53 exchanges. Exchanges where the frequency is not available will request daily candles instead. This behaviour can be changed which is further described in the project's repository.

After having described how *open-crypto* can be used to retrieve static and historical data from platforms and exchanges, we do now turn our focus on how to access the database in which the data is stored. We provide additional insights on how to use *open-crypto* on real-time data, specifically order books and recent transactions, in the online supplementary materials.

4 Data Export

The export method depends on the purpose pursued. The following is based on data collected in Section 3.2, in particular historical minute candles. If not already executed during Section 3.2, in order to reproduce the following, run:

```
from open_crypto import runner
runner.run(configuration_file="examples/minute_candles", kill_after=60)
```

This section documents three possibilities to retrieve data from the database, (1) third-party tools like *SQLBrowser*, (2) inherent functions to write data into csv- or hdf-files, and (3) opening a database session and using either plain `SQL` or the ORM-mapper functionality.

Firstly, *SQLBrowser* comes with a clearly arranged interface and offers, among other functionalities, data inspection, manipulation, filtration and exportation. Server-based database platforms, for example *PostgreSQL*, offer different tools like *pgAdmin*.

Secondly, regarding the inherent export method, the block `database` in the export configuration file below is unchanged to the main program and sets the connection string to the database. The block `query_options` defines table and time horizon, followed by filters for exchanges and currency pairs. The key `query_everything` ignores any filter and returns the table as a whole.

```
export:
  database: <...>
  query_options:
    delimiter: ","
    decimal: "."
    table_name: HistoricRate
    query_everything: True
    from_timestamp: null
    to_timestamp: null
  exchanges: all
  currency_pairs: eth-btc
  first_currencies: null
  second_currencies: null
```

The export is executed by calling the function `export` within the module `runner`. The respective export configuration file needs to be passed as an argument and *open-crypto* will save the data within the current working directory as a csv- or hdf-file. The file name consists of the database table name and the current timestamp. The export itself is done with *Pandas* (The pandas development team (2020)). Additional arguments can be passed as keyword-arguments:

```
# Write the data into a csv file
runner.export(file='examples/example_export', file_format='csv')
```

Thirdly, it is possible to retrieve an open database session from the `runner` module. This is achieved by typing:

```
# Receive an open database session
session = runner.get_session('examples/minute_candles')
```

The command above takes the configuration file that was executed in order to run the example. The returned database connection is capable of handling either plain `SQL` language to construct queries, or to stick with `Python`'s object-oriented syntax. Taking the session to explore the database structure further, yields:

```
from sqlalchemy import inspect

# Inspect the associated database engine from the session
inspector = inspect(session.bind)
table_names = inspector.get_table_names()

for table in table_names:
  print(table)
```

```
currencies
exchanges
exchanges_currency_pairs
historic_rates
order_books
tickers
trades
```

This prints all table names from the database describing schema. Note that the table names, unlike the class name, are typically written in lowercase and plural. Consequently, the class `Exchange`, used for queries in the object oriented syntax, refers to the table `exchanges`. A valid query could be:

```
query = session.query(runner.Exchange.id, runner.Exchange.name)
query.order_by(runner.Exchange.id).first()
```

```
(1, '50X')
```

This returns the first (auto-incremented) exchange identifier and the corresponding exchange name. Depending on the order in which users have executed the examples, the returned exchange name may differ. Going one step further, currency pairs listed on exchanges can be retrieved by:

```
# Request the first ExchangeCurrencyPair entry in the database
session.query(runner.ExchangeCurrencyPair).first()
```

```
#1: 50X(1), XLM(1)-USDT(2)
```

The proposed package *open-crypto* thus offers a variety of possibilities to access the collected data from the database. The most intuitive and fastest approach is the *SQLBrowser* which fits most applications. Alternatively, users can write the data in a csv or hdf file. Finally, an open database session can be retrieved to load data directly into the RAM or to be further used within other programs.

5 Summary

This paper introduces a new open source Python package to request cryptocurrency market data. The program is designed to be OS independent, easily customizable and expandable to attract a wide range of users. Due to the increasing interoperability of programming languages, *open-crypto* is widely applicable. We provide a tool to request both platforms and over 100 exchanges directly, which offers users the possibility for unprocessed and free-of-charge data from the sources. The importance of being able to access both sources is demonstrated with short illustrations about the global aggregated price series in Section 3.2.1 and the underlying information only available from the exchanges themselves in Section 3.2.2. We contribute to open and FAIR data principles in research to make data findable, accessible, interoperable, and reusable. Independent and transparent research is build upon free data and software. The package is uploaded to Python's package index and provided as a Github repository. We hope for the cryptocurrency community to adapt, use and further develop this tool.

Supplementary Material

We provide several additional information in the supplementary materials, regarding (1) further information on the installation process, (2) troubleshooting list, (3) requesting real-time data, (4) exchanges and endpoints, (5) cross-software usability from R and Matlab and (6) connectivity to server-based database management systems.

References

Aloosh A, Li J (2019). Direct evidence of bitcoin wash trading. Working paper.
Bayer M (2012). Sqlalchemy. In: *The Architecture of Open Source Applications Volume II: Structure, Scale, and a Few More Fearless Hacks* (A Brown, G Wilson, eds.). aosabook.org.
Cong L, Li X, Tang K, Yang Y (2019). Crypto wash trading. Working paper.
Gewin V (2016). Data sharing: An open mind on open data. *Nature*, 529(7584): 117–119.
Hougan M, Kim H, Lerner M (2019). Economic and non-economic trading in bitcoin: Exploring the real spot market for the world's first digital commodity. Working Paper.
Iacus SM (2015). Automated data collection with R – A practical guide to web scraping and text mining. *Journal of Statistical Software, Book Reviews*, 68(3): 1–3.
Kim N, Svetlov A (2020). *Aiohttp. Release 4.0.0a1.*
Makarov I, Schoar A (2020). Trading and arbitrage in cryptocurrency markets. *Journal of Financial Economics*, 135(2): 293–319.
Nakamoto S (2008). Bitcoin: A peer-to-peer electronic cash system.
Pennec GL, Fiedler I, Ante L (2021). Wash trading at cryptocurrency exchanges. *Finance Research Letters*, 43: 101982.
R Core Team (2020). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria.
Reichman OJ, Jones MB, Schildhauer MP (2011). Challenges and opportunities of open data in ecology. *Science*, 331(6018): 703–705.
Szöcs E, Stirling T, Scott ER, Scharmüller A, Schäfer RB (2020). webchem: An R package to retrieve chemical information from the web. *Journal of Statistical Software*, 93(13): 1–17.
The pandas development team (2020). pandas-dev/pandas: Pandas.
van Rossum G (2012). Asyncio: Asynchronous IO support rebooted: The "asyncio" module.
van Rossum G, Drake FL (2009). *Python 3 Reference Manual.* CreateSpace, Scotts Valley, CA.