Irving Fisher Committee on
Central Bank Statistics

BIS

12th biennial IFC Conference: "Statistics and beyond: new data for decision making in central banks"

22-23 August 2024

BCChAPI: A python interface to the Central Bank of Chile statistical database API[1]

Ricardo Villarreal Zan,
Central Bank of Chile

[1] This contribution was prepared for the conference. The views expressed in this publication are those of the authors and do not necessarily represent the official views of the Committee, its members, or the BIS.

BCChAPI: A Python interface to the Central Bank of Chile Statistical Database API

Ricardo Villarreal Zan[1]

Abstract

The Central Bank of Chile provides the public with an extensive statistical database of economic time series where users can find all statistics published by the bank and other relevant data produced by different institutions. bcchapi is a Python library that offers an interface to the official API, making it easier for developers and analysts to automate the data retrieval process in a standardized and simple way. The main objects of the package aim to deliver the data in a ready-to-use format, taking into account the various modifications that a user may need. Furthermore, a set of tools are provided to allow manipulation of the query mechanism so that advanced programmers may also incorporate the library into their projects.

Keywords: Data access, time series, open source.

JEL classification: C82, C88.

[1] Analyst at the Statistics and Data Division, Central Bank of Chile (rvillarreal@bcentral.cl). The views and conclusions presented in this paper are exclusively those of the author and do not necessarily reflect the position of the Central Bank of Chile or its Board Members.

Introduction

The Statistical Database (SDB) of the Central Bank of Chile consists of more than 22,000 economic time series organized in tables and thematic chapters. It is one of the official channels of the institution to publish the statistics it produces, such as gross domestic product, the balance of payments and monetary policy rates, but data from various sources can also be found, like sectorial indicators, inflation and unemployment rates. Most of the series have a monthly frequency, but annual, quarterly, and daily series are also available. The SDB has tools to manipulate, visualize and export data, as well as the functionality to register an account to create new tables and save favourite series. In this way the SDB is a valuable asset for researchers and policy makers, allowing easy and direct access to a centralized economic data portal.

The SDB API[2] allows the user to extract the same data available on the website in a machine-friendly format. Two query methods are available: to obtain updated series data in real time, and to review the series available at the moment. To access this application the user is required to be registered in the SDB to activate their credentials as tokens for the API requests. This is a one-time procedure which, after accepting the terms of use, guarantees access to the service indefinitely and free of charge.

All the applications mentioned above are part of the Central Bank of Chile's statistics dissemination strategy. In addition to this, there are official reports, social media accounts, blogs, bulletins, mobile applications, among others. The joint objective of all these platforms is to facilitate the arrival of statistical products to different audiences. In the same vein, **bcchapi** is a free and open source Python library that provides an interface to the SDB API through functions and methods that deliver the results in a ready-to-use format. The main objective of the library is to facilitate the connection to the service in a simple and standardized way, but it also allows the transformation and manipulation of data using the pandas (McKinney, 2010) infrastructure. Python is a high-level programming language with a community that has grown significantly over the last years. It has been a preferred choice in areas such as machine learning and data science (Raschka, Patterson, & Nolet, 2020) where extensive data availability is required. Other disciplines such as econometrics have also begun to incorporate this language in their research (Bilina & Lawford, 2012).

The possibility of including the process of obtaining the input data in a code file facilitates the reproducibility of the models, since data origin and pre-processing can be explicitly described (Gundersen, 2021) and its execution yields the same results regardless of the machine that runs it. Additionally, the structure of the library is arranged so that developers can integrate the API into their Python applications regardless of the tools they use. These functionalities can also be used by institutions interested in model transparency, such as central banks (Araujo, 2024), simplifying the documentation of the data collection process in the models they share with the public. It is important to emphasise that bcchapi's support for the readability and replicability of a model is not a substitute for the use of good practice when programming, but a complement.

[2] The API documentation, along with examples, the activation system, terms of use and a list of available series can be found on its official website: https://si3.bcentral.cl/Siete/en/Siete/API

The library is available in the Python Package Index (PyPI) and needs two dependencies: pandas and requests. The last available version can be easily installed using the pip command:

```
pip install --upgrade bcchapi
```

The rest of the document shows the Python examples as monospaced text paragraphs. The three greater-than signs (>>>) at the start of each line means it is a prompt for executable code. Otherwise, it is a printed result. The following section describes the main bcchapi objects with examples on how to search and retrieve data, as well as showing the different data manipulation options. The Next section shows how to use the API objects to extend the possible functionalities. The final section concludes.

Using bcchapi

To illustrate the options provided by bcchapi to obtain data, series of the Chilean Residential Property Prices Index, (RPPI) of the Central Bank of Chile (Balsa & Vásquez, 2023) are searched and requested. The RPPI follows the behaviour of urban housing prices in the Chilean market. The index can be found in the SDB together with its breakdowns by type of property and geographical zone. This is a total of 19 quarterly series with observations since 2002.

After creating an account in the SDB and activating the API credentials, the user can start using the library. The main object of the package is `Stat` and to be initialized the credentials must be included as arguments:

```
>>> import bcchapi
>>> stat = bcchapi.Stat("user@example.com", "password")
```

If for security reasons the user does not wish to include his credentials in the code itself, it is possible to call an external text file containing the account in the first line and the password in the second line. The following code is equally valid as the previous one:

```
>>> import bcchapi
>>> stat = bcchapi.Stat(file="file.txt")
```

In the probable case that the user does not know the code of the series to be consulted, the `Stat` object contains the `browse` method to review the different series according to their name. The result is a pandas Data Frame, with the name, code and metadata of the series found, along with the frequency, date of creation and last update.

```
>>> stat.browse("RRPI, base year 2008")[["seriesId", "englishTitle"]]
                          seriesId                          englishTitle
0   F034.IPVCNEW.FLU.BCCH.2008.0.T       New Houses RPPI, base year 2008
1  F034.IPVCUSED.FLU.BCCH.2008.0.T      Used Houses RPPI, base year 2008
2   F034.IPVDNEW.FLU.BCCH.2008.0.T   New Apartments RPPI, base year 2008
3  F034.IPVDUSED.FLU.BCCH.2008.0.T  Used Apartments RPPI, base year 2008
4    F034.IPVNEW.FLU.BCCH.2008.0.T              New RPPI, base year 2008
5   F034.IPVUSED.FLU.BCCH.2008.0.T             Used RPPI, base year 2008
...
```

The first parameter of the method can be any type of string and accepts regular expressions for complex searches. Additional arguments are `spanish` (boolean, `False` by default) to search on Spanish names instead of English, and `cache` (boolean, `True` by default) to use previously queried data if it exists. Also, the resulting Data Frame already contains the dates formatted for filtering and sorting as needed by the user.

Having found the series to be queried, the `table` method allows the creation of a table with the specifications required by the user, equivalent to the functionalities of the SDB tables. The most direct and simple data extraction query is presented below for the general RPPI:

```
>>> stat.table("F034.IPV.FLU.BCCH.2008.0.T")
            F034.IPV.FLU.BCCH.2008.0.T
2002-01-01                   79.819860
2002-04-01                   82.868177
2002-07-01                   83.698377
2002-10-01                   83.196206
2003-01-01                   82.737826
...                                ...
2022-10-01                  207.904201
2023-01-01                  206.128934
2023-04-01                  209.207870
2023-07-01                  214.702977
2023-10-01                  216.197511

[88 rows x 1 columns]
```

Any iterable object can be included in the first parameter as a sequence of series. For ease of reading, the second argument (`names`) is used to rename columns in the final Data Frame. In the following example, RPPI series of *new* and *used* households are requested:

```
>>> stat.table(
>>>   ["F034.IPVNEW.FLU.BCCH.2008.0.T", "F034.IPVUSED.FLU.BCCH.2008.0.T"],
>>>   names=["NEW", "USED"]
>>>   )
                  NEW        USED
2002-01-01   74.369865   86.456022
2002-04-01   77.466766   89.466483
2002-07-01   78.574419   89.986566
2002-10-01   77.896892   89.680522
2003-01-01   78.639495   87.856824
...                ...         ...
2022-10-01  202.908282  221.927034
2023-01-01  200.830102  219.881758
2023-04-01  201.759438  225.095925
2023-07-01  207.270524  230.811105
2023-10-01  209.709356  231.489393

[88 rows x 2 columns]
```

The first parameter of the function can also be a dictionary, where its *key* is the desired name, and the associated *value* is the series to query. In addition, the `start` and end arguments define time limits of the table. These dates can be text strings in "YYYY-MM-DD" format or date objects that has the `strftime` method available.

```
>>> series = {
>>>   "NEW": "F034.IPVNEW.FLU.BCCH.2008.0.T"
>>>   "USED": "F034.IPVUSED.FLU.BCCH.2008.0.T"
```

```
>>>   }
>>>
>>> stat.table(
>>>   series,
>>>   first="2010-01-01"
>>>   last="2019-12-31"
>>>  )
                  NEW        USED
2010-01-01  101.575202  110.822143
2010-04-01  102.123649  118.690864
2010-07-01  101.918715  119.599060
2010-10-01   99.950539  123.707302
2011-01-01  103.341884  124.585955
...                ...         ...
2018-10-01  176.493581  214.787282
2019-01-01  176.474774  217.546302
2019-04-01  174.332188  218.053320
2019-07-01  180.650763  225.175379
2019-10-01  180.998875  230.384470

[40 rows x 2 columns]
```

The `variation` argument allows the calculation of fractional differences with respect to previous periods. If an integer is used, it is assumed to be the number of months to be considered.[3] For example, the number 12 is equivalent to calculating the year-on-year variation. Note in the following code that the `table` method performs the query and then calculates the variations, so the first observations will be filled with `NaN`.

```
>>> stat.table(
>>>   series,
>>>   first="2010-01-01"
>>>   last="2019-12-31",
>>>   variation=12
>>>  )
                NEW      USED
2010-01-01      NaN       NaN
2010-04-01      NaN       NaN
2010-07-01      NaN       NaN
2010-10-01      NaN       NaN
2011-01-01  0.017393  0.124197
...              ...       ...
2018-10-01  0.055570  0.090198
2019-01-01  0.071809  0.077099
2019-04-01  0.032073  0.067821
2019-07-01  0.042941  0.062367
2019-10-01  0.025527  0.072617

[40 rows x 2 columns]
```

To aggregate the series in a different frequency, the parameters `frequency` and `observed` define the temporality and aggregation function respectively. Within `frequency`, the values "A" for annual, "Q" for quarterly, "M" for monthly and "W" for weekly are accepted. By default, dates are grouped with respect to the last day of the period. If the first day is needed instead, an "S" (for start) after the frequency acronym can be added. That is: "AS", "QS" and "MS". To change the frequency, it is mandatory

[3] In case a different interval is required, such as bi-weekly or semi-annual evolution, the parameter also accepts `DateOffset` pandas objects.

to define the `observed` parameter, otherwise it will raise a `ValueError`. Here the user must enter a text string referring to a Data Frame method or directly input generic or anonymous functions. The `observed` argument can also receive a dictionary to use different functions for each series.

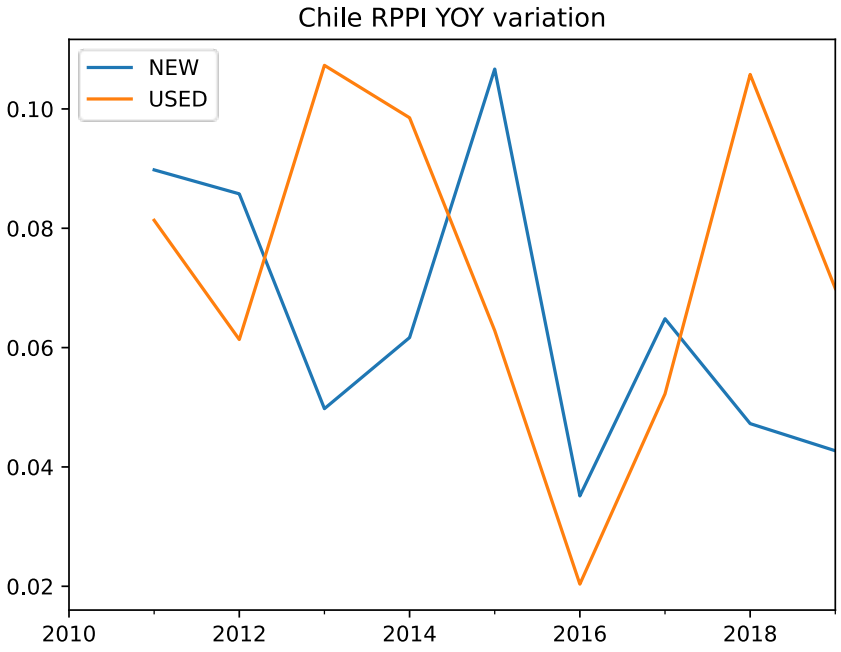
```
>>> stat.table(
>>>   series,
>>>   first="2010-01-01"
>>>   last="2019-12-31",
>>>   variation=12,
>>>   frequency="A",
>>>   observed="last"
>>>  )
                NEW      USED
2010-12-31      NaN       NaN
2011-12-31  0.141361  0.066650
2012-12-31  0.093145  0.069691
2013-12-31  0.031338  0.094581
2014-12-31  0.099993  0.083904
2015-12-31  0.101787  0.075774
2016-12-31  0.007428  0.008232
2017-12-31  0.064764  0.084694
2018-12-31  0.055570  0.090198
2019-12-31  0.025527  0.072617
```

The resulting Data Frame is ready to use. Any pandas method can be employed in it to further manipulate the series, both data and dates. Additional libraries allow direct use of this object, such as statsmodels (Seabold & Perktold, 2010) for econometric analysis, or Matplotlib (Hunter, 2007) for plotting. The following code concludes the RPPI examples with a line chart displayed in Figure 1.

```
>>> import numpy as np
>>>
>>> df = stat.table(
>>>   series,
>>>   first="2010-01-01"
>>>   last="2019-12-31",
>>>   variation=12,
>>>   frequency="AS",
>>>   observed=np.mean,
>>>  )
>>>
>>> df.plot(title="Chile RPPI YOY variation") # see Figure 1
>>> df
                NEW      USED
2010-01-01      NaN       NaN
2011-01-01  0.089787  0.081337
2012-01-01  0.085769  0.061354
2013-01-01  0.049766  0.107306
2014-01-01  0.061682  0.098522
2015-01-01  0.106686  0.062848
2016-01-01  0.035144  0.020361
2017-01-01  0.064839  0.052244
2018-01-01  0.047262  0.105783
2019-01-01  0.042713  0.069920
```

Example of plotting the result of a bcchapi object.

Figure 1



Sources: Prepared by the author based on data from the Central Bank of Chile.

The table method has three more arguments. Since the frequency change is performed with the methods `DataFrame.resample` and `DataFrame.aggregate`, the dictionary parameters `resample_kw` and `aggregate_kw` are passed as keyword arguments to each function. Finally, the `stop_invalids` argument (boolean, `True` by default) defines whether or not to raise an error when the queried series do not exist or are not available.

Extensions

The features that make Python a modern language lead to the fact that any of its parts can be deprecated and replaced by new and better tools. This section specifies the low-level objects available in the library to code new functionalities, for example, to implement new objects from different packages without the need of using pandas. In addition, the different types of errors that can be raised by them are listed.

The `Session` class allows the user to establish a direct connection to the API and receive the response data as is. This object is initialized in the same way as `Stat`, with the user's credentials in the first two arguments or by entering a text file. A `Session` object contains two methods, the same ones provided by the SDB API: `get` and `search`.

The `get` method accepts three arguments. The first one, `time_series` (required) is a string with the id of the series to be queried. The get object accepts three arguments. The first one, `time_series` (required) is a string with the id of the series to be queried. The other two parameters, `first_date` and `last_date`, both optional, accept a string in "YYYYY-MM-DD" format to delimit the start and end of the series. The method returns a `GSResponse` object, which contains the information in its attributes.

Specifically, the series names and data are contained in the `Series` attribute, which is a dictionary, where the "Obs" key is a list of observations. Additionally, this object has the methods `to_series` and `to_dataframe` to convert the content into pandas Series and Data Frames respectively. The following code creates a new class, inheriting from `Session`, with a method that allows the user to extract the last observation of a time series.

```
>>> import bcchapi
>>>
>>> class MySession(bcchapi.webservice.Session):
>>>   """MySession docstring"""
>>>   def last_observation(self, serie:str) -> float:
>>>     """Returns the last observation of a time series."""
>>>     ts = self.get(serie).Series["Obs"]
>>>     last = ts[-1]["value"]
>>>     return float(last)
```

The search method accepts only one argument, which is the frequency of the data to be searched. "DAILY", "MONTHLY", "QUARTERLY" or "ANNUAL" is allowed. It responds on an `SSResponse` object, which contains the information in its `SeriesInfos` attribute, a list of dictionaries. The `to_dataframe` method converts the data to a pandas object.

For error handling, all exception classes in the library are raised by `Session` methods. The main exception object is `ResponseException` and the rest of the available errors are inheritors of this class. An `InvalidCredentials` error is raised in case of making any API call having declared incorrect credentials when initializing the object, either username, password or both. The `search` method raises an `InvalidFrequency` error if the desired frequency is other than “ANNUAL”, “QUARTERLY”, “MONTHLY” or “DAILY”. Finally, the `get` method raises an `InvalidSeries` exception if the requested series does not exist and an `InvalidDate` error if the dates in the request cannot be recognized. Note that all these errors are raised only if the API returns an unsuccessful request. In case of other problems, such as Internet failures, they must be handled by the exception classes of the requests package.

Conclusion

In line with the Central Bank of Chile's strategy of disseminating statistics and seeking to facilitate access to data to new audiences, the bcchapi Python library provides a convenient and user-friendly interface to access the economic series available in the Central Bank of Chile's Statistical Database. By leveraging the SDB API, researchers and developers can easily extract and manipulate data for their analysis and applications, while facilitating readability and reproducibility in their work. The library's integration with pandas makes it easy to work with data, and its support for various methods ensures that users can access the information they need efficiently.

bcchapi also contributes to the broader goal of promoting transparency and replicability. By providing a user-friendly interface to the SDB API, the library serves as a valuable resource for researchers, developers, and institutions seeking to enhance the data collection process of their economic models. As the Python programming language continues to grow in popularity across various disciplines, the library offers a

practical solution for incorporating economic data into projects, ultimately contributing to more robust and reproducible analyses.

References

Araujo, D. (2024). Open-sourced central bank macroeconomic models. *SSRN*. doi:10.2139/ssrn.4755247

Balsa, J. J., & Vásquez, J. (2023). Índice de Precios de Vivienda. *Estudios Económicos Estadísticos*(139). Retrieved from https://www.bcentral.cl/contenido/-/detalle/estudio-economico-estadistico-n-139

Bilina, R., & Lawford, S. (2012). Python for Unified Research in Econometrics and Statistics. *Econometric Reviews, 31*(51), 558-591. doi:10.1080/07474938.2011.553573

Gundersen, O. E. (2021). The fundamental principles of reproducibility. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 379*(2197). doi:10.1098/rsta.2020.0210

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering, 9*(3), 90-95. doi:10.1109/MCSE.2007.55

McKinney, W. (2010). Data Structures for Statistical Computing in Python. In S. van der Walt, & J. Millman (Ed.), *Proceedings of the 9th Python in Science Conference*, (pp. 56-61). doi:10.25080/Majora-92bf1922-00a

Raschka, S., Patterson, J., & Nolet, C. (2020). Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence. *Information, 11*(4). doi:10.3390/info11040193

Seabold, S., & Perktold, J. (2010). statsmodels: Econometric and statistical modeling with python. *9th Python in Science Conference*.

BCChAPI:
A python interface to Central Bank of Chile Statistics Database API

Ricardo Villarreal Zan
Statistics and Data Division

Central Bank of Chile, august, 2024

Dissemination of Statistics at the CBCh

Statistics Database
(centralized DB)

Mobile Apps

Website

@bcch_stat

Reports

API

#Blog

Blogs

And more!

banco central Chile

banco central Chile

Statistics Database (BDE)

New search for statistics database

Type here

BDE | My account

National Accounts

IMACEC, GDP, Expenditure, financial and non-financial accounts.

Derivatives and Spot

Derivatives on underlying assets and spot market.

International Economy

Macroeconomic and financial indicators of industrialized economies.

Experimental Statistics

Daily retail sales, business statistics and housing market indicators.

Money and Banking

Loans, deposits and monetary aggregates.

Economic Expectations

Economic expectations surveys, perception and confidence index.

Public Finances

Government income and expenditure.

Gender

Population, employment, finances and entrepreneurship.

Housing Market Indicators

Activity, prices, rates and financial conditions.

Sectoral Indicators

Trade, construction, industry and mining indicators.

Employment, Wages and Demographics

Employment, wages and demographics.

Prices

UF, IVP and UTM. CPI and underlying measures.

Macro Economic Statistics

Most requested indicators.

Regional

Activity, employment and financial and economic indicators.

External Sector

Balance of payments, international investment position and external debt.

Interest Rates

Benchmark monetary policy and market rates.

17 chapters | 1,350 tables |+21,000 time series

banco central Chile

Statistics Database (BDE)

Home Daily Indicators Chart Pack

Log in Register

Contents

- Daily Retail Sales Index (DRSI)
- Purchases and sales with electronic invoices
- Business-to-business credit indicators (ICCE)
- Business statistics using administrative registers
- Mobility statistics
- Internet Job Ad Index (IALI)
- Housing market indicators
  - Property price index (RPPI)
    - Property price index
      - **General index and by type of property**
      - By geographical zone
    - Original Y/Y contribution
  - Market Value of Housing
  - Other indicators
- Daily economic uncertainty index

Experimental Statistics

Date: 2010 | Date end: 2024 | Frequency: Quarterly | Calculation: Original series

General RPPI, House and Apartment

Sel.	Serie	III.2019	IV.2019	I.2020	II.2020	III.2020	IV.2020	I.2021	II.2021	III.2021	IV.2021	I.2022	II.2022	III.2022	IV.2022
☐	General RPPI	200.8	203.5	204.9	206.4	203.1	206.7	214.9	219.5	219.2	224.5	213.9	210.7	208.6	207.6
☐	House RPPI	205.4	209.6	209.2	209.1	202.2	209.3	220.6	226.4	225.3	231.3	214.2	210.2	211.9	210.8
☐	Used Houses RPPI	215.7	222.2	221.8	219.9	208.3	216.3	233.3	243.6	240.6	246.1	222.4	220.5	217.0	214.0
☐	New Houses RPPI	190.5	190.8	190.1	193.8	196.6	202.1	202.7	200.7	202.8	209.9	209.9	199.7	215.5	219.3
☐	Aparment RPPI	196.9	198.2	201.4	204.3	203.9	204.5	210.1	213.6	214.1	218.9	214.1	211.7	205.9	205.1
☐	New Apartments RPPI	173.5	173.8	179.1	179.7	185.9	182.0	183.6	186.5	186.3	193.8	198.5	194.9	189.5	192.1
☐	Used Apartments RPPI	246.8	250.0	248.7	256.6	241.6	252.4	267.0	272.0	273.9	272.2	253.8	253.1	246.3	240.2
☐	New RPPI	180.7	181.0	184.4	185.9	191.2	189.9	191.4	192.8	193.3	200.8	204.2	198.6	199.4	202.3
☐	Used RPPI	225.2	230.4	229.7	231.4	218.6	227.5	243.6	251.9	250.7	253.6	232.0	230.5	225.9	221.8

Data handling functionalities

banco central Chile

Statistics Database (BDE)

Home Daily Indicators Chart Pack

Log in Register

Contents

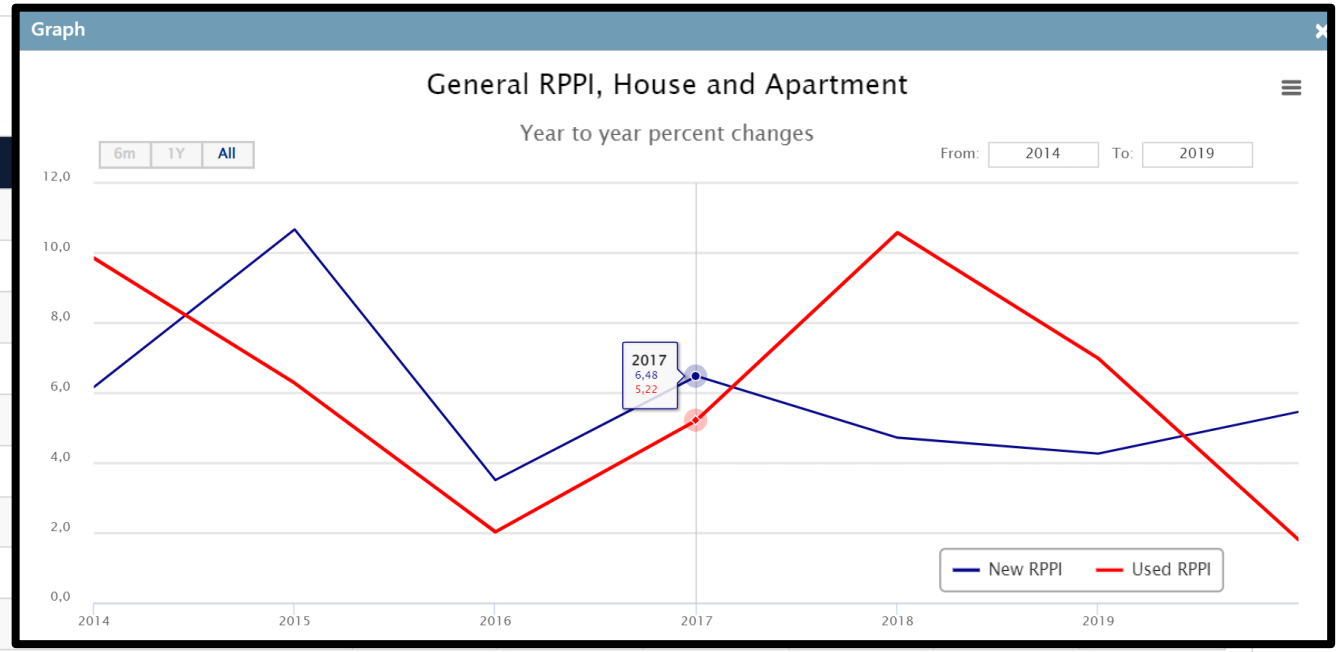
- Daily Retail Sales Index (DRSI)
- Purchases and sales with electronic invoices
- Business-to-business credit indicators (ICCE)
- Business statistics using administrative registers
- Mobility statistics
- Internet Job Ad Index (IALI)
- Housing market indicators
  - Property price index (RPPI)
    - Property price index
      - **General index and by type of property**
      - By geographical zone
    - Original Y/Y contribution
  - Market Value of Housing
  - Other indicators
- Daily economic uncertainty index

Experimental Statistics

Date	Date end	Frequency	Calculation
2014	2019	Annual	Year to year percent changes

General RPPI, House and Apartment

Sel.	Serie
☐	General RPPI
☐	House RPPI
☐	Used Houses RPPI
☐	New Houses RPPI
☐	Aparment RPPI
☐	New Apartments RPPI
☐	Used Apartments RPPI
☑	New RPPI
☑	Used RPPI



Tool to query economic data – API

Statistics Database API

Inicio / Statistics / BDE API

The Central Bank of Chile provides a web service that allows searching and extracting economic information contained in its Statistical Data Base (BDE). The records are displayed in a format useful for developers and analysts who wish to automate their query processes. Within this site you can find codes, manuals and documentation for accessing the BDE series, using Python, R and C#, as well as documentation of the APIs, along with the terms of use and the activation required to access this service.

Service Status

I have read and accept the Terms and conditions

Activate API usage

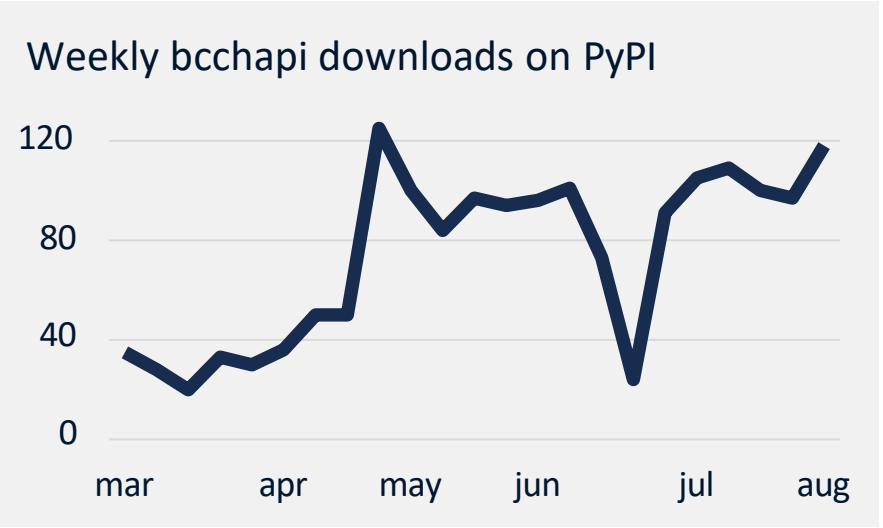
API BDE

- Home
- Examples
- List of series available
- Documentation
- Terms of use

A Python package

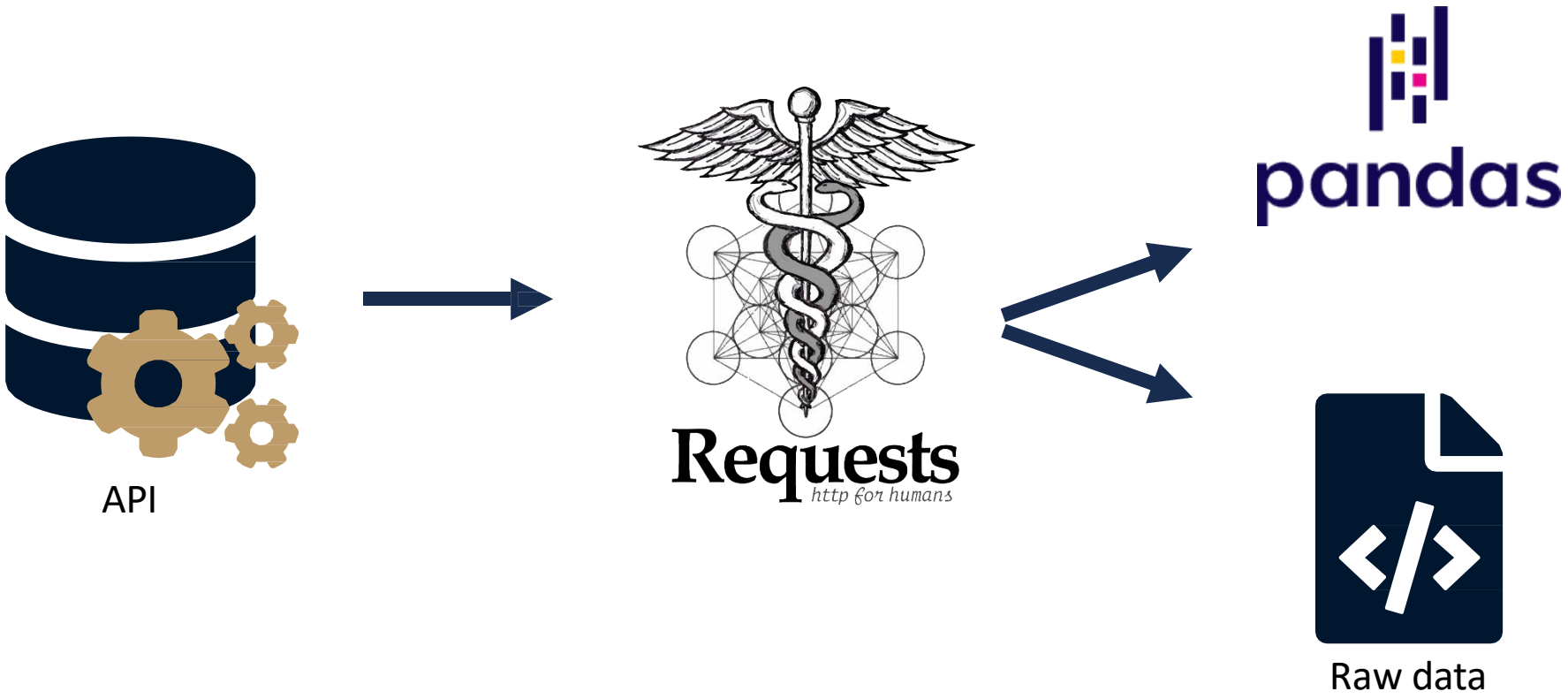
bcchapi

- ✓ Open source API wraper for python
- ✓ Methods includes series extraction, searching and constructing tables
- ✓ No need for repetitive downloading of data files
- ✓ Allows to explicitly describe data collection in a model
- ✓ Promotes transparency and replicability



banco central Chile

Inside bcchapi

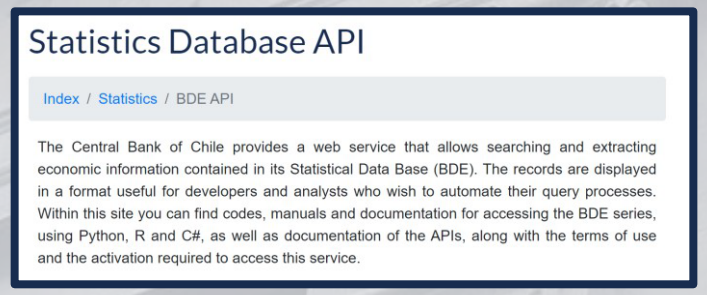


banco central Chile

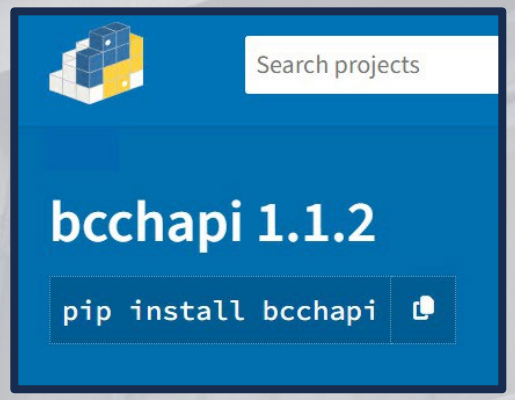
Thank you!

[Statistics Database]

[BDE API site]



[bcchapi on PyPI]



banco central Chile