API BDE using Python

Central Bank of Chile, June 10, 2020

How to access the CBC Statistics Database

Introduction

This Notebook will explain how to access data from the Bank's Statistics Database (BDE) using Python.

Access to documents and forms:
https://si3.bcentral.cl/estadisticas/Principal1/Web_Services/index.htm

To access the data using the API, you must request access credentials (username and password) by send an email to contacto_ws@bcentral.cl , and fill this: Form

Example

How to get a dataframe that will include:

- Time series ID
- Frequency and English name (obtained from the Webservice "SearchSeries")
- Data existing within a date range (obtained from the Webservice "GetSeries")

Requirements

- Internet access
- Basic knowledge of Python3
- Access credentials (user and password)
- Start date (string format yyyy-mm-dd, e.g., "2017-01-01")
- End date (string format yyyy-mm-dd, e.g., "2019-01-01")
- List of Series IDs to request

The Bank provides two Webservices that contain the needed information to prepare the aforementioned dataframe:

SearchSeries

Using a **user, password** and a **frequency ("Daily","Monthly","Quarterly" or "Annual")**, it returns a dataframe with the time series corresponding to the desired frequency (for example, for all the annual time series) with the following fields:

- SeriesId
- Frequency
- FrequencyCode
- Observed
- ObservedCode
- SpanishTitle

- EnglishTitle
- firstObservation
- lastObservation
- updatedAt
- createdAt

GetSeries

Using a **user, password, start date, end date** and a **time series ID**, it returns a dataframe with the available data between **start date** and **end date** for the requested time series ID. Formally, the request will return the following fields:

- IndexDateString
- KeyFamilyId
- LastModified
- LastModifiedUser
- SeriesId
- DataStage
- Exists
- Description
- DescripIng
- DescripEsp
- StatusCode
- Value

In particular, this exercise will use the fields "indexDateString", "seriesKey" and "value".

Next, needed modules to implement the code will be imported. In case of an error in the next cell, please verify the installation:

[1]:
```python
# 1.
#Importing the necessary modules
#Generating the connection with the Webservice
import zeep
#Used to work with Dataframes
import pandas as pd
#Used to work with the data sent by the Webservice
from zeep.helpers import serialize_object
#Used to sort the dates of the observations
import datetime as dt
#Used to work with arrays
import numpy as np
#Allows the program to wait a certain period of
#time before performing a new request
from time import sleep
#On the event of an error, stops the execution
import sys
```

Then, the inputs to use are defined:

[2]:
```
# 2.
#Inputs creation for the connection with the Webservice

user="user"
pw="password"

fInic="2017-12-31"
fFin="2019-12-31"

#This exercise will request 2 time series:
# - 'Monetary policy rate (MPR) (percentage)'
# - 'Observed US Dollar Exchange Rate'
# Whose IDs are, respectively:
series=["F022.TPM.TIN.D001.NO.Z.D","F073.TCO.PRE.Z.M"]
series=[x.upper() for x in series]
print(series)
```

```
['F022.TPM.TIN.D001.NO.Z.D', 'F073.TCO.PRE.Z.M']
```

A catalogue with the available time series can be downloaded from: https://si3.bcentral.cl/estadisticas/Principal1/Web_Services/Webservices/series.xls

The code below reviews the validity of the entered time series IDs. After this, the variables needed are assigned to make the first query to the “SearchSeries” service. In order to make the query more efficient, the different frequencies in series are requested only once:

[3]:
```
# 3.
#It checks for an invalid ID. All of them should end in d, m, t, or a␣
 ↪(corresponding to the frequencies). If an ID is invalid, the user is␣
 ↪notified and the ID is removed from the list
for ser_cod in reversed(series):
    if ser_cod[-1] in ["D","M","T","A"]:
        pass
    else:
        print("Serie " + ser_cod + " inexistente. Chequea el código")
        series.remove(ser_cod)


#The different frequencies of the time series are identified and classified by␣
 ↪type
series_freq=[x[-1] for x in series]
series_freq=list(np.unique(series_freq))
#series_freq is the list that will contain the unique frequency values
print(series_freq)

#Now, the initial is replaced by the name of the frequency, which is needed to␣
 ↪create the request
for x in range(len(series_freq)):
    if series_freq[x]=="D":
        series_freq[x]=series_freq[x].replace("D","DAILY")
    elif series_freq[x]=="M":
        series_freq[x]=series_freq[x].replace("M","MONTHLY")
    elif series_freq[x]=="T":
        series_freq[x]=series_freq[x].replace("T","QUARTERLY")
    elif series_freq[x]=="A":
        series_freq[x]=series_freq[x].replace("A","ANNUAL")
    else:
        pass
print(series_freq)
```

```
['D', 'M']
['DAILY', 'MONTHLY']
```

In the next code the request to the Webservice “SearchSeries” is performed. Please notice that the obtained result shows only the variables of interest (“seriesId”,“frequency”,“englishTitle”):

[4]:
```
# 4.
#The WSDL (Web Service Definition Language) address is defined, which will␣
 ↪allow the zeep library to identify which queries can be made to the␣
 ↪Webservice, in addition to generate the client object, that will allow data␣
 ↪exchange
wsdl="https://si3.bcentral.cl/SieteWS/SieteWS.asmx?wsdl"
client = zeep.Client(wsdl)

#meta_series will contain the downloaded data obtained from "SearchSeries" for␣
 ↪all frequencies
meta_series=pd.DataFrame()

#Iterates through the list series_freq to request the different frequencies of␣
 ↪interest:
for frequ in series_freq:
    for attempt in range(4):
        try:
            #The request is performed operating the user, password and frequency
            res_search=client.service.SearchSeries(user,pw,frequ)
            #Cleaning the information
            res_search=res_search["SeriesInfos"]["internetSeriesInfo"]
            res_search = serialize_object(res_search)
            #A dictionary is created, using the downloaded time series metadata␣
 ↪(title, time series ID and frequency)
            res_search = { serie_dict['seriesId']:
 ↪[serie_dict['englishTitle'],serie_dict['frequency']] for serie_dict in␣
 ↪res_search }
```

```
            #Using the previous dictionary, a dataframe (meta_series_aux) is␣
↪created and then added to the dataframe that will contain all the␣
↪frequencies (meta_series)
            meta_series_aux=pd.DataFrame.from_dict(res_search,orient='index')
            meta_series=meta_series.append(meta_series_aux)
            print("Frequency " + str(frequ) + " found. Adding")
            break
        except:
            print("Attempt " + str(attempt) + ": The frequency " + str(frequ) +␣
↪" was not found")
            #On the event of an error, the function waits 20 seconds before␣
↪performing a new request on the frequency
            sleep(20)
    else:
        print("Frequency " + str(frequ) + " was not found. Stopping execution")
        sys.exit("Stopping execution")

#Finally, the obtained Dataframe is cleaned to keep only the series of interest:
meta_series=meta_series.loc[series]
meta_series.columns=["englishTitle","frequency"]
print(meta_series)
```

```
Frequency DAILY found. Adding
Frequency MONTHLY found. Adding
                                                      englishTitle frequency
F022.TPM.TIN.D001.NO.Z.D  Monetary policy rate (MPR) (percentage)     DAILY
F073.TCO.PRE.Z.M                   Observed US Dollar Exchange Rate   MONTHLY
```

The result of the execution of the previous code is the variable meta_series, a dataframe where each of its rows corresponds to a time series ID and its columns contain the variables "englishTitle" and "frequency". In the next code the request to the Webservice "GetSeries" is performed.

[5]:
```
# 5.
#Creation of the DataFrame values_df, that will contain the numeric data of all␣
↪the requested time series
values_df=pd.DataFrame()
#Iterates inside the list series to request the numeric data:
for serieee in series:
    #A loop is generated to perform 4 request attempts per time series. If it␣
↪is successful, it continues with the next time series
    for attempt in range(4):
        try:
            #Creation of the object that will contain the times series ID      ␣
↪
            ArrayOfString = client.get_type('ns0:ArrayOfString')
            value = ArrayOfString(serieee)
            #The request is performed using the parameters (user, password,␣
↪start date, end date and time series ID) The response is saved in the␣
↪variable result
            result = client.service.GetSeries(user,pw,fInic,fFin, value)
            #If there are no observations in the defined date range, the time␣
↪series is omitted
            if result["Series"]["fameSeries"][0]["obs"]==[]:
                print("Time series "+ str(serieee) + " does not have data for␣
↪the requested time range. Omitting")
                break
            #The information obtained is cleaned, leaving the time series ID as␣
↪the row name and, as columns, the dates (dd-mm-yyyy)
            result = serialize_object(result["Series"]["fameSeries"][0]["obs"])
            result=pd.DataFrame(result).T
            result.columns=result.iloc[0,:]
            result=result.drop(result.index[0:3],axis=0)
            result.index=[serieee]

            #The data is added to the DataFrame values_df

            values_df=values_df.append(result,sort=True)
            print("Time series " + str(serieee) + " found. Adding")
            break
        except:
            print("Attempt " + str(attempt) + ": The time series " +␣
↪str(serieee) + " was not found")
            #On the event of an error, the function waits 20 seconds before␣
↪performing a new request on the ID
            sleep(20)
    else:
        print("The time series " + str(serieee) + " was not founnd. Omitting")
```

```
Time series F022.TPM.TIN.D001.NO.Z.D found. Adding
Time series F073.TCO.PRE.Z.M found. Adding
```

The last section of this tutorial checks if the numeric data are sorted by date (oldest to newest) and join them, in a dictionary called final_dic, with the metadata extracted previously.

[6]:
```
# 6.
#new_col contains the columns names of values_df
new_col=list(values_df.columns)
#The dates in new_col are sorted from oldest to newest
new_col.sort(key = lambda date: dt.datetime.strptime(date, '%d-%m-%Y'))
#The dataframe values_df is sorted using the variable new_col
values_df=values_df[new_col]
#meta_series is joined with values_df. The result is saved in final_dic
final_dic=pd.merge(meta_series,values_df,left_index=True,right_index=True)
```

```
#final_dic is splitted to obtain a dataframe by frequency
final_dic = dict(iter(final_dic.groupby('frequency')))
final_dic.update((x, y.dropna(axis=1,how="all")) for x, y in final_dic.items())
#The result of final_dic is printed
print(final_dic)
```

```
{'DAILY':                                                        englishTitle
frequency  \
F022.TPM.TIN.D001.NO.Z.D  Monetary policy rate (MPR) (percentage)     DAILY

                          02-01-2018 03-01-2018 04-01-2018 05-01-2018  \
F022.TPM.TIN.D001.NO.Z.D         2.5        2.5        2.5        2.5

                          08-01-2018 09-01-2018 10-01-2018 11-01-2018  …  \
F022.TPM.TIN.D001.NO.Z.D         2.5        2.5        2.5        2.5  …

                          17-12-2019 18-12-2019 19-12-2019 20-12-2019  \
F022.TPM.TIN.D001.NO.Z.D        1.75       1.75       1.75       1.75

                          23-12-2019 24-12-2019 26-12-2019 27-12-2019  \
F022.TPM.TIN.D001.NO.Z.D        1.75       1.75       1.75       1.75

                          30-12-2019 31-12-2019
F022.TPM.TIN.D001.NO.Z.D        1.75       1.75

[1 rows x 497 columns], 'MONTHLY':
englishTitle frequency 01-12-2017  \
F073.TCO.PRE.Z.M  Observed US Dollar Exchange Rate   MONTHLY    636.924

                 01-01-2018 01-02-2018 01-03-2018 01-04-2018 01-05-2018  \
F073.TCO.PRE.Z.M    605.529    596.839    603.445    600.548    626.119

                 01-06-2018 01-07-2018  … 01-03-2019 01-04-2019 01-05-2019  \
F073.TCO.PRE.Z.M    636.146    652.407  …    667.679    667.399    692.004

                 01-06-2019 01-07-2019 01-08-2019 01-09-2019 01-10-2019  \
F073.TCO.PRE.Z.M    692.409     686.06    713.703    718.442    721.032

                 01-11-2019 01-12-2019
F073.TCO.PRE.Z.M     776.53     770.39

[1 rows x 27 columns]}
```

This is the final result of this tutorial: A dictionary of dataframes by frequency. These dataframes contain, by row, a time series ID, name, frequency and the observations between the date range specified. To include more time series, just add the time series ID, as a string, to the list "series", defined in the second section. Also, if a different date range is needed, just re-define fInic and fFin.

This code is also included as a function, to immediately start downloading data from the BDE.

According to the terms and conditions of use of the Webservice, the user may require a maximum of **5 time series per second**, corresponding to 5 requests to the Webservice, for each authorized user, regardless of the IP address or addresses sending the requests.