API BDE desde Python

Banco Central de Chile, 10 de junio de 2020

Cómo acceder a datos de la Base de datos Estadísticos del BCCh

Introducción

En el siguiente Notebook se explicará cómo acceder a datos de la Base de Datos Estadísticos (BDE) del Banco usando Python3.

Acceso a documentos y formularios:
https://si3.bcentral.cl/estadisticas/Principal1/Web_Services/index.htm

Para acceder a la BDE a través de la API deben solicitarse credenciales de acceso (usuario y contraseña) al correo electrónico contacto_ws@bcentral.cl , y completar lo siguiente: Formulario

Ejemplo

Cómo obtener una base de datos (DataFrame) que incluirá:

- Códigos de serie
- Frecuencia y nombre en español (obtenidos desde el Webservice "SearchSeries")
- Las observaciones existentes dentro de un rango de fechas (obtenidas desde el Webservice "GetSeries")

Requisitos

- Acceso a Internet
- Conocimiento básico de Python3
- Credenciales de acceso (usuario y contraseña)
- Fecha de inicio (como string en formato aaaa-mm-dd, ej: "2017-01-01")
- Fecha de término (como string en formato aaaa-mm-dd, ej: "2019-01-01")
- Lista de códigos de series a consultar

El Banco provee de dos Webservices que contienen la información necesaria para la elaboración del Dataframe mencionado:

SearchSeries

Utilizando **usuario, contraseña** y una frecuencia de serie **("Daily","Monthly","Quarterly" o "Annual")**, retorna un dataframe con todas las series correspondientes a la frecuencia ingresada (por ejemplo, para todas las series anuales) con los siguientes datos:

- SeriesId
- Frequency
- FrequencyCode
- Observed
- ObservedCode

- SpanishTitle
- EnglishTitle
- firstObservation
- lastObservation
- updatedAt
- createdAt

GetSeries

Utilizando **usuario, contraseña, fecha de inicio, fecha de término y código de serie**, devuelve un dataframe con todas las observaciones en el período comprendido entre **fecha de inicio y fecha de término** para el código de serie consultado. Formalmente, la consulta arrojará los siguientes campos:

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

Para este ejercicio en particular, se utilizarán sólo los campos “indexDateString”, “seriesKey” y “value”.

A continuación, se importarán los módulos necesarios para implementar el código. En caso de un error en la siguiente celda, verificar la instalación:

[1]:
```
# 1.
#Importar los módulos necesarios
#Crea la conexión con el Webservice
import zeep
#Permite trabajar con Dataframes
import pandas as pd
#Facilita el arreglo de los datos recibidos desde el Webservice
from zeep.helpers import serialize_object
#Verifica el orden correlativo de las fechas
import datetime as dt
#Permite trabajar con arreglos de datos
import numpy as np
#Permite que el programa espere un determinado período de
#tiempo antes de hacer una nueva consulta
from time import sleep
#Detiene la ejecución en caso de error
import sys
```

Luego, se definen los inputs a usar:

[2]:
```
# 2.
#Creación de inputs para conectar al Webservice

user="usuario"
pw="password"

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

#A modo de ejemplo, se consultan 2 series:
# - 'Tasa de política monetaria (TPM) (porcentaje)'
# - 'Tipo de Cambio del Dólar Observado'
# Cuyos códigos son, respectivamente:
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']
```

Un listado con todas las series disponibles en Webservice puede ser descargado desde: https://si3.bcentral.cl/estadisticas/Principal1/Web_Services/Webservices/series.xls

A continuación, se revisa la validez de los códigos de serie ingresados. Luego de esto, se asignan las variables necesarias para hacer la primera consulta al servicio "SearchSeries". Con el objetivo de hacer la consulta más eficiente, se busca una vez, cada una de las distintas frecuencias presentes en la lista de series a consultar:

[3]:
```
# 3.
#Se revisa si hay un código inválido. Todos debiesen terminar en d, m, t, o a␣
 ↪(correspondiente a las frecuencias). En caso de ser inválido, se avisa al␣
 ↪usuario y se retira de la lista a consultar
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)


#Se identifican las distintas frecuencias de series que pudieran existir y se␣
 ↪clasifican por cada tipo
series_freq=[x[-1] for x in series]
series_freq=list(np.unique(series_freq))
#series_freq será la lista que contendrá los valores únicos de frecuencia
print(series_freq)
```

```
#Ahora, se convierte la inicial de la frecuencia a su nombre, que es el que se␣
 ↪necesita para hacer la consulta
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']
```

En el siguiente cuadro se realiza la consulta al Webservice “SearchSeries”. Nótese que el resultado obtenido muestra las variables de interés únicamente (“seriesId”,“frequency”,“spanishTitle”):

[4]:
```
# 4.
#Se identifica la dirección del WSDL (Web Service Definition Language) que␣
 ↪permitirá a la librería zeep identificar qué consultas pueden hacerse al␣
 ↪Webservice, además de generar el objeto client, que permitirá el intercambio␣
 ↪de datos
wsdl="https://si3.bcentral.cl/SieteWS/SieteWS.asmx?wsdl"
client = zeep.Client(wsdl)

#meta_series contendrá los datos recolectados y ordenados obtenidos desde␣
 ↪"SearchSeries" para todas las frecuencias
meta_series=pd.DataFrame()

#Se itera dentro de la lista series_freq para consultar las distintas␣
 ↪frecuencias de interés:
for frequ in series_freq:
    for attempt in range(4):
        try:
            #Se consulta usando el usuario, password y frecuencia de interés
            res_search=client.service.SearchSeries(user,pw,frequ)
            #Se limpia la información obtenida
            res_search=res_search["SeriesInfos"]["internetSeriesInfo"]
            res_search = serialize_object(res_search)
            #Se crea un diccionario con las series obtenidas y los datos de␣
 ↪interés (título, código y frencuencia)
```

```
            res_search = { serie_dict['seriesId']:
↪[serie_dict['spanishTitle'],serie_dict['frequency']] for serie_dict in␣
↪res_search }
            #A partir del diccionario creado, se arma un dataframe␣
↪(meta_series_aux) que luego se agrega al dataframe
            #que contendrá todas las frecuencias (meta_series)
            meta_series_aux=pd.DataFrame.from_dict(res_search,orient='index')
            meta_series=meta_series.append(meta_series_aux)
            print("Frecuencia " + str(frequ) + " encontrada. Agregando")
            break
        except:
            print("Intento " + str(attempt) + ": La frecuencia " + str(frequ) +␣
↪" no fue encontrada")
            #En caso de error, se esperan 20 segundos antes de volver a␣
↪consultar la serie
            sleep(20)
    else:
        print("Frecuencia " + str(frequ) + " no fue encontrada. Deteniendo␣
↪ejecución")
        sys.exit("Deteniendo ejecución")

#Finalmente, se limpia el Dataframe obtenido para conservar sólo las series de␣
↪interés:
meta_series=meta_series.loc[series]
meta_series.columns=["spanishTitle","frequency"]
print(meta_series)
```

```
Frecuencia DAILY encontrada. Agregando
Frecuencia MONTHLY encontrada. Agregando
                                                          spanishTitle  \
F022.TPM.TIN.D001.NO.Z.D  Tasa de política monetaria (TPM) (porcentaje)
F073.TCO.PRE.Z.M                     Tipo de Cambio del Dólar Observado

                         frequency
F022.TPM.TIN.D001.NO.Z.D     DAILY
F073.TCO.PRE.Z.M           MONTHLY
```

El resultado final de la ejecución anterior es la variable meta_series, un dataframe en que cada una de sus filas es un código de serie, y en sus columnas contiene las variables "spanishTitle" y "frequency". En el siguiente cuadro se consultará al Webservice "GetSeries".

[5]:
```
# 5.
#Creación del DataFrame values_df, que incluirá todas las series que se␣
↪consultarán
values_df=pd.DataFrame()
#Iteración por cada una de las series consultando los datos. Los valores␣
↪obtenidos se agregan a values_df
```

```
for serieee in series:
    #Se genera un loop para hacer 10 intentos de consulta por serie. Si tiene␣
↪éxito, continúa con la siguiente serie, si no tiene éxito, intenta nuevamente
    for attempt in range(4):
        try:
            #Creación del objeto que contendrá el código de serie
            ArrayOfString = client.get_type('ns0:ArrayOfString')
            value = ArrayOfString(serieee)

            #Se ejecuta la consulta utilizando los parámetros ingresados␣
↪(usuario, password, fecha de inicio, fecha final y código de serie) y se␣
↪asigna a la variable result
            result = client.service.GetSeries(user,pw,fInic,fFin, value)
            #Se omite la serie si no hay observaciones en el período solicitado
            if result["Series"]["fameSeries"][0]["obs"]==[]:
                print("La serie "+ str(serieee) + " no tiene observaciones para␣
↪el período seleccionado")
                break
            #Se limpia la información obtenida, dejando como nombre de fila el␣
↪código de serie y, como columnas, las fechas en formato dd-mm-aaaa
            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]

            #Se agrega la serie ordenada al DataFrame values_df

            values_df=values_df.append(result,sort=True)
            print("Serie " + str(serieee) + " encontrada. Agregando")
            break
        except:
            print("Intento " + str(attempt) + ": La serie " + str(serieee) + "␣
↪no fue encontrada")
            #En caso de error, se esperan 20 segundos antes de volver a␣
↪consultar la serie
            sleep(20)
    else:
        print("La serie " + str(serieee) + " no fue encontrada. Omitiendo")
```

```
Serie F022.TPM.TIN.D001.NO.Z.D encontrada. Agregando
Serie F073.TCO.PRE.Z.M encontrada. Agregando
```

La última parte de este tutorial consiste en revisar que las extracciones estén ordenadas por fecha (desde la más antigua a la más nueva) y crear un diccionario de dataframes por frecuencia, con toda la información extraída, en la variable final_dic

[6]:
```
# 6.
#Se guarda en new_col los nombres de las columnas de values_df
new_col=list(values_df.columns)
#Se ordenan las fechas de new_col para asegurar su disposición desde la más␣
 ↪antigua a la más nueva
new_col.sort(key = lambda date: dt.datetime.strptime(date, '%d-%m-%Y'))
#Se ordena el dataframe values_df con el orden descrito en la línea anterior
values_df=values_df[new_col]
#Se unen los resultados de meta_series con values_df en final_dic
final_dic=pd.merge(meta_series,values_df,left_index=True,right_index=True)
#Se separan las salidas para obtener un dataframe por frecuencia
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())
#Se muestra el resultado de final_dic
print(final_dic)
```

```
{'DAILY':
spanishTitle  \
F022.TPM.TIN.D001.NO.Z.D  Tasa de política monetaria (TPM) (porcentaje)

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

                         05-01-2018 08-01-2018 09-01-2018 10-01-2018  \
F022.TPM.TIN.D001.NO.Z.D        2.5        2.5        2.5        2.5

                         11-01-2018  … 17-12-2019 18-12-2019 19-12-2019  \
F022.TPM.TIN.D001.NO.Z.D        2.5  …       1.75       1.75       1.75

                         20-12-2019 23-12-2019 24-12-2019 26-12-2019  \
F022.TPM.TIN.D001.NO.Z.D       1.75       1.75       1.75       1.75

                         27-12-2019 30-12-2019 31-12-2019
F022.TPM.TIN.D001.NO.Z.D       1.75       1.75       1.75

[1 rows x 497 columns], 'MONTHLY':
spanishTitle frequency 01-12-2017  \
F073.TCO.PRE.Z.M  Tipo de Cambio del Dólar Observado   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]}
```

Este es el resultado final del tutorial: Un diccionario de dataframes por frecuencia, que por cada fila contiene un código de serie, nombre, frecuencia, y las observaciones existentes entre las fechas especificadas. Si se quisiera agregar más series se debe agregar el código de serie, como string, a la lista "series", definida en la segunda celda. A su vez, si se quisiera cambiar las fechas basta con re-definir fInic y fFin

También se incluye una función para comenzar a hacer consultas de inmediato a la BDE.

Se recuerda que, según los terminos y condiciones de uso del Webservice, el usuario podrá requerir un máximo de **5 series por segundo**, correspondientes a 5 consultas al Webservice, por cuenta habilitada, independiente desde la o las direcciones IP que realicen las consultas.