bcrpy

Release 1.0

Andrew Garcia, Ph.D.

Mar 20, 2023

CONTENTS

bcrpy

Este modulo de Python es un cliente API para la extraccion, consulta y analisis de la base de datos BCRPData del Banco Central de Reserva del Peru (BCRP) el cual trabaja como un wrapper de la API para Desarrolladores del BCRP

BANCO CENTRAL DE RESERVA DEL PERÚ

`bcrpy` esta siendo diseñado para su integracion con algoritmo(s) de Inteligencia Artificial (AI) para generar modelos estocasticos de prediccion.

Revisa la seccion de *Uso* para mas informacion, incluyendo la *Instalacion* del modulo.

Tambien podrias hacer clic en la imagen de abajo para un tutorial interactivo a traves de un guardable Colab Jupyter notebook:

Note: This project is under active development.

CHAPTER

ONE

CONTENIDO

1.1 Uso

1.1.1 Instalacion

En su sistema local (laptop o computadora) *bcrpy* puede ser instalada con el comando *pip install bcrpy*. Aun asi, se recomienda instalar *bcrpy* dentro de un ambiente virtual virtualenv. El protocolo para aquel seria el siguiente:

```
$ virtualenv venv
$ source venv/bin/activate
(.venv) $ pip install bcrpy
```

bcrpy ha sido desarrollado con un protocolo de programación orientada a objetos (tambien conocido como *Object Oriented Programming (OOP)*) lo cual se reduce a que objetos pueden ser usados a almacenar metodos (funciones), datos, y su manejo de aquellos.

1.1.2 Extraccion de metadatos y busqueda de palabras en aquellos

En el caso de abajo, vemos como el objeto definido con la variable `banco` se usa para extraer los metadatos del BCRPData con el metodo `get_metadata`, el cual la almacena como un `Pandas DataFrame` dentro de su variable constructora `metadata`

```
import bcrpy

banco = bcrpy.Marco()                          # cargar objeto
banco.get_metadata()                           # obtener todos los metadatos del BCRPData
```

```
>>> print(type(banco.metadata))           # imprimir estructura de data de metadatos
<class 'pandas.core.frame.DataFrame'>
>>> print(banco.metadata.shape)      # imprimir numero de filas y columnas
(14858, 14)
```

Arriba vemos que los metadatos almacenados en `banco.metadata` contienen 14,858 filas con 14 columnas.

El siguiente ejemplo muestra el metodo `wordsearch`, el cual usa un algoritmo de *fuzzy string matching* para encontrar palabras parecidas a la palabra que esta siendo buscada. En el caso de abajo, usamos `wordsearch` para buscar la palabra “economia” en las columnas 0 y 1 (primera y segunda) de la base de datos del BCRP.

```
>>> banco.wordsearch('economia',columnas =[0,1])
corriendo wordsearch: `economia` (fidelity = 0.65)*
*medido con Levenshtein similarity ratio
por favor esperar...
```

```
50%|| 1/2 [00:01<00:01,  1.65s/it]
100%|| 2/2 [00:04<00:00,  2.10s/it]

      Código de serie              Categoría de serie  \
8437       CD11605DA  Primera centuria independiente
8438       CD11606DA  Primera centuria independiente
8439       CD11607DA  Primera centuria independiente
8440       CD11608DA  Primera centuria independiente
8441       CD11609DA  Primera centuria independiente
...              ...                             ...
6960       PM10083FA             Resultado económico
6961       PM10084FA             Resultado económico
6962       PM10085FA             Resultado económico
6963       PM10086FA             Resultado económico
6964       PM10087FA             Resultado económico


   Fecha de inicio Fecha de fin  Memo  Unnamed: 13
8437            1791         1862   NaN          NaN
8438            1791         1876   NaN          NaN
8439            1791         1876   NaN          NaN
8440            1791         1876   NaN          NaN
8441            1791         1876   NaN          NaN
...              ...          ...   ...          ...
6960            1970         2022   NaN          NaN
6961            1970         2022   NaN          NaN
6962            1970         2022   NaN          NaN
6963            1970         2022   NaN          NaN
6964            1970         2022   NaN          NaN

[1608 rows x 14 columns]
```

Podemos ver en la primera linea del output que la fidelidad a encontrar la palabra exacta esta predeterminada en 0.65 (65%).

Si quisieramos buscar una palabra en la base de datos que sea 100% igual (capitalizacion incluida), podemos aumentar el input `fidelity` a un valor de 1, como lo hacemos abajo con la palabra "centuria". Notemos que si no se especifica el input `columns`, el metodo corre la busqueda de la palabra en todas las columnas.

```
>>> banco.wordsearch('centuria',fidelity=1)
corriendo wordsearch: `centuria` (fidelity = 1)*
*medido con Levenshtein similarity ratio
por favor esperar...
```

```
8%|| 1/12 [00:01<00:17,  1.59s/it]
17%|| 2/12 [00:07<00:39,  3.95s/it]
25%|| 3/12 [00:20<01:15,  8.34s/it]
```

(continues on next page)

(continued from previous page)

```
33%|| 4/12 [00:25<00:54,  6.87s/it]
42%|| 5/12 [00:27<00:36,  5.26s/it]
50%|| 6/12 [00:28<00:23,  3.85s/it]
58%|| 7/12 [00:30<00:15,  3.15s/it]
67%|| 8/12 [00:38<00:19,  4.81s/it]
75%|| 9/12 [00:44<00:15,  5.04s/it]
83%|| 10/12 [00:47<00:08,  4.28s/it]
92%|| 11/12 [00:48<00:03,  3.37s/it]
100%|| 12/12 [00:49<00:00,  4.13s/it]

      Código de serie              Categoría de serie  \
8437       CD11605DA  Primera centuria independiente
8438       CD11606DA  Primera centuria independiente
8439       CD11607DA  Primera centuria independiente
8440       CD11608DA  Primera centuria independiente
8441       CD11609DA  Primera centuria independiente
...              ...                              ...
9028       CD12207DA  Primera centuria independiente
9029       CD12208DA  Primera centuria independiente
9030       CD12209DA  Primera centuria independiente
9031       CD12210DA  Primera centuria independiente
9032       CD12211DA  Primera centuria independiente

   Fecha de inicio Fecha de fin  Memo  Unnamed: 13
8437            1791         1862   NaN          NaN
8438            1791         1876   NaN          NaN
8439            1791         1876   NaN          NaN
8440            1791         1876   NaN          NaN
8441            1791         1876   NaN          NaN
...              ...          ...   ...          ...
9028            1926         1933   NaN          NaN
9029            1918         1924   NaN          NaN
9030            1918         1924   NaN          NaN
9031            1922         1933   NaN          NaN
9032            1921         1933   NaN          NaN

[596 rows x 14 columns]
```

1.1.3 Consultas con codigos de serie

Tambien podemos hacer consultas individuales de un codigo de serie con el metodo `query`, para que nos den la informacion mas organizada en una estructura de mapa (json). Abajo, hacemos dos consultas con dos codigos de serie de la database:

```
#hacer una consulta del codigo de serie  'CD12209DA' con el API del BCRPData
banco.query('CD12209DA')

#hacer otra consulta, pero para el codigo de serie 'CD11608DA'
banco.query('CD11608DA')
```

```
[Out]

corriendo query para CD12209DA...

CD12209DA es indice 9030 en metadatos
{
      "Código de serie": "CD12209DA",
      "Categoría de serie": "Primera centuria independiente",
      "Grupo de serie": "Marina mercante nacional, 1918-1931",
      "Nombre de serie": "Tonelaje de Registro ",
      "Fuente": "Compendio de Historia Económica del Perú - Tomo IV",
      "Frecuencia": "Anual",
      "Fecha de creación": "2018-05-24",
      "Grupo de publicación": NaN,
      "Área que publica": "Departamento de Bases de Datos Macroeconómicas",
      "Fecha de actualización": "2018-05-24",
      "Fecha de inicio": "1918",
      "Fecha de fin": "1924",
      "Memo": NaN
}
corriendo query para CD11608DA...

CD11608DA es indice 8440 en metadatos
{
      "Código de serie": "CD11608DA",
      "Categoría de serie": "Primera centuria independiente",
      "Grupo de serie": "Población por departamentos y provincias para 1791, 1836, 1850,␣
↪1862 y 1876 (número)",
      "Nombre de serie": "Lima - Amazonas - Totales Departamentales",
      "Fuente": "Compendio de Historia Económica del Perú - Tomo IV",
      "Frecuencia": "Anual",
      "Fecha de creación": "2018-05-24",
      "Grupo de publicación": NaN,
      "Área que publica": "Departamento de Bases de Datos Macroeconómicas",
      "Fecha de actualización": "2018-05-24",
      "Fecha de inicio": "1791",
      "Fecha de fin": "1876",
      "Memo": NaN
}
```

1.1.4 Facil extraccion de series economicas y generacion de graficas

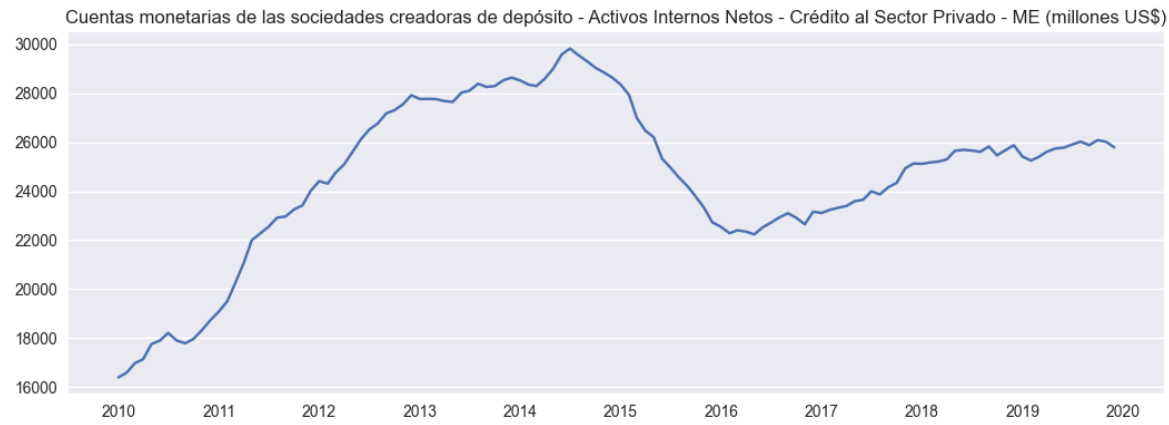
El ingenio del *Object Oriented Programming (OOP)* se encuentra en que los inputs del objeto (en este caso, el objeto definido como `banco`) pueden ser modificados y sus metodos (funciones) pueden funcionar con aquellos cambios.

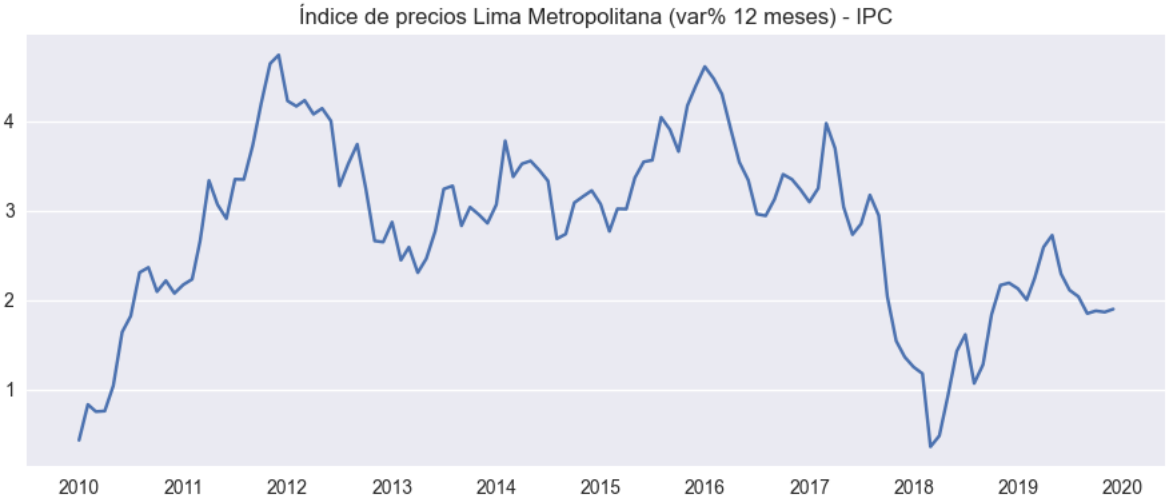
Abajo se definen los codigos de serie y el rango de fechas para despues imprimirlos con el metodo `state_inputs()` y extraear los datos con aquellas especificaciones del BCRPData con el metodo `GET()`, el cual regresa aquellos datos como un `Pandas DataFrame`.

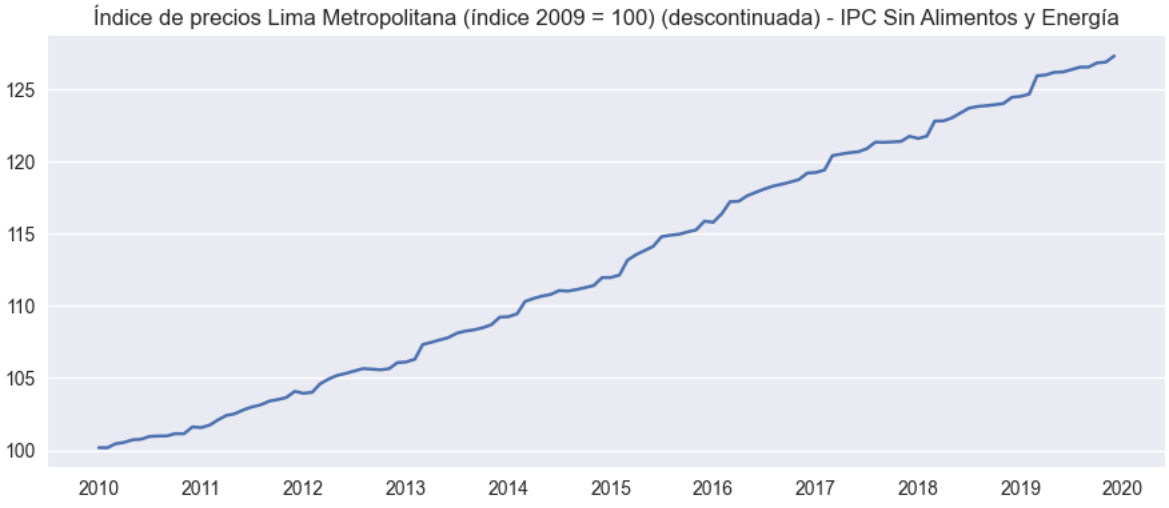
Como podemos ver abajo, estos datos son almacenados en la variable `df`, la cual se usa para hacer graficos con el metodo `plot()` del objeto definido como `banco`.

```
import matplotlib.pyplot as plt

#escoger los inputs de los datos que se desean extraer del BCRPData (otros datos como␣
↪banco.idioma (='ing') son predeterminados, pero tambien se pueden cambiar)
banco.codigos = ['PN00015MM','PN01289PM','PD39793AM','PN01273PM']
banco.fechaini = '2011-1'
banco.fechafin = '2021-1'

banco.state_inputs()                # mostrar el estado actual de los inputs escogidos

df = banco.GET()     # obtener informacion de los inputs escogidos (arriba) con el API␣
↪del BCRP

#graficos (plots)
for name in df.columns:
   plt.figure(figsize=(9, 4))
   banco.plot(df[name],name,12)

plt.show()
```

```
[Out]

corriendo estado actual de todas las variables constructoras...

self.metadata = <class 'pandas.core.frame.DataFrame'> size: (14858, 14)
self.codigos = ['PN00015MM', 'PN01289PM', 'PD39793AM', 'PN01273PM']
self.formato = json
self.fechaini = 2011-1
self.fechafin = 2021-1
self.idioma = ing

https://estadisticas.bcrp.gob.pe/estadisticas/series/api/PN00015MM-PN01289PM-PD39793AM-
↪PN01273PM/json/2011-1/2021-1/ing
```







Las graficas no se imprimen en el orden que se alistan en banco.codigos, pero en el orden que aparecen en las columnas en BCRPData.

Si se necesita consultar la identidad de los nombres de serie con sus codigos (y viceversa), este se puede hacer, nuevamente, con el metodo `query`, demostrado abajo:

```
>>> [banco.query(codigo) for codigo in banco.codigos]   #referencia, codigos
```

```
[Out]

corriendo query para PN00015MM...

PN00015MM es indice 14 en metadatos
{
      "Código de serie": "PN00015MM",
      "Categoría de serie": "Sociedades creadoras de depósito",
      "Grupo de serie": "Cuentas monetarias de las sociedades creadoras de depósito",
      "Nombre de serie": "Activos Internos Netos - Crédito al Sector Privado - ME␣
↪(millones US$)",
      "Fuente": "BCRP",
      "Frecuencia": "Mensual",
      "Fecha de creación": "2022-03-24",
      "Grupo de publicación": "Sistema financiero y empresas bancarias y expectativas␣
↪sobre condiciones crediticias",
      "Área que publica": "Departamento de Estadísticas Monetarias",
      "Fecha de actualización": "2023-02-24",
      "Fecha de inicio": "Abr-1992",
      "Fecha de fin": "Sep-2022",
      "Memo": NaN
}
corriendo query para PN01289PM...

PN01289PM es indice 1212 en metadatos
{
      "Código de serie": "PN01289PM",
      "Categoría de serie": "Inflación",
      "Grupo de serie": "Índice de precios Lima Metropolitana (índice 2009 = 100)␣
↪(descontinuada)",
      "Nombre de serie": "IPC Sin Alimentos y Energía",
      "Fuente": "INEI",
      "Frecuencia": "Mensual",
      "Fecha de creación": "2022-04-07",
      "Grupo de publicación": "Índice de precios al consumidor y tipo de cambio real",
      "Área que publica": "Departamento de Estadísticas de Precios",
      "Fecha de actualización": "2022-04-07",
      "Fecha de inicio": "Abr-1991",
      "Fecha de fin": "Sep-2021",
      "Memo": NaN
}
corriendo query para PD39793AM...

PD39793AM es indice 14855 en metadatos
{
      "Código de serie": "PD39793AM",
      "Categoría de serie": "Expectativas Empresariales",
      "Grupo de serie": "Expectativas empresariales sectoriales",
      "Nombre de serie": "Índice de expectativas del sector a 12 meses - Servicios",
      "Fuente": NaN,
```

(continues on next page)

(continued from previous page)

```
      "Frecuencia": "Mensual",
      "Fecha de creación": "2023-02-28",
      "Grupo de publicación": "Expectativas macroeconómicas y de ambiente empresarial",
      "Área que publica": "Departamento de Indicadores de la Actividad Economía",
      "Fecha de actualización": "2023-03-09",
      "Fecha de inicio": "Abr-2010",
      "Fecha de fin": "Sep-2022",
      "Memo": NaN
}
corriendo query para PN01273PM...

PN01273PM es indice 1198 en metadatos
{
      "Código de serie": "PN01273PM",
      "Categoría de serie": "Inflación",
      "Grupo de serie": "Índice de precios Lima Metropolitana (var% 12 meses)",
      "Nombre de serie": "IPC",
      "Fuente": "INEI",
      "Frecuencia": "Mensual",
      "Fecha de creación": "2022-04-08",
      "Grupo de publicación": "Índice de precios al consumidor y tipo de cambio real",
      "Área que publica": "Departamento de Estadísticas de Precios",
      "Fecha de actualización": "2023-03-09",
      "Fecha de inicio": "Abr-1950",
      "Fecha de fin": "Sep-2022",
      "Memo": NaN
}
```

1.2 API

See the below classes.

class bcrpy.**Marco**

GET(*filename=False*)

Extrae los datos del BCRPData selecionados por las previamente-declaradas variables *self.codigos*, *self.fechaini*, *self.fechafin*, *self.formato*, y *self.idioma*.

1.2.1 Parametros

filename
[str (opcional)] Nombre para guardar los datos extraidos como un archivo .csv

get_metadata(*filename='metadata.csv'*)

Extrae todos los metadatos de BCRPData.

1.2.2 Parametros

filename
[str] Nombre del archivo para guardar todos los metadatos extraidos como un archivo .csv (predeterminado: 'metadata.csv'). Si se desea no guardar un archivo, cambiar a filename="

`load_metadata`(*filename='metadata.csv'*)

Carga los metadatos guardados como archivo .csv a Python.

1.2.3 Parametros

filename
[str] Nombre del archivo .csv del cual cargar los metadatos a Python.

`plot`(*data*, *title=''*, *titlesize=9*, *func='plot'*)

Grafica x-y data.

1.2.4 Parametros

data
[<Pandas DataFrame>] Data x-y extraida de BCRPData, x es fecha y es cantidad.

title
[str] Titulo para grafica

func
[str] Tipo de grafica. 'plot' es grafica comun, 'semilogy' es grafica con escala logaritmica en y-axis.

titlesize
[str] Tamaño de titulo para grafica

`query`(*codigo='PD39793AM'*)

Consulta (query) de codigo de serie, impresa en formato json.

1.2.5 Parametros

codigo
[str] Nombre de codigo de series a consultar

`ref_metadata`(*filename=False*)

Reduce los metadatos en self.metadata a aquellos perteneciendo a los codigos de serie declarados en self.codigos.

1.2.6 Parametros

filename
[str (opcional)] Nombre para guardar la informacion de la modificada self.metadata como un archivo .csv

`save_metadata`(*filename='metadata_new.csv'*)

Guarda los metadatos de self.metadata como archivo .csv

1.2.7 Parametros

filename
[str] Nombre para el archivo .csv (predeterminado = 'metadata_new.csv')

`state_inputs`()

Declara el estado actual de todas las variables constructoras de la clase Marco.

`wordsearch`(*keyword='economia'*, *fidelity=0.65*, *columnas='all'*, *verbose=False*)

Busqueda difusa de palabra clave (keyword) en metadatos de BCRPData. Regresa una tabla de datos en formato <Pandas DataFrame> de los metadatos asociados con aquella palabra.

1.2.8 Parametros

keyword
[str] Palabra clave para reducir los metadatos

fidelity
[float] Este es el Levenshtein similarity ratio (predeterminado=0.65). Un fidelity de 1.00 solo regresara metadatos que contienen palabras que coinciden con la palabra clave al 100%.

columnas
[str] Indices de columnas de los metadatos seleccionados para correr el metodo. Predeterminado='all' corre el metodo en todas las columnas.

verbose
[bool] Muestra las columnas que estan siendo elegidas mientras el metodo corre (predeterminado=False)

bcrpy

1.2.9 bcrpy

PYTHON MODULE INDEX

INDEX