bcrpy

Release 1.1

Andrew Garcia, Ph.D.

Mar 24, 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 *Instalación* del modulo.

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

Note: Este proyecto se encuentra bajo desarrollo activo.

CHAPTER

ONE

CONTENIDO

1.1 Uso

1.1.1 Instalación

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`
cutoff = 0.65; columnas = [0, 1]
por favor esperar...
```

```
  0%|                                                                                | 0/2 [00:00<?, ?
↪it/s]
 50%|||||||||||||||||||||||||||||||                                                 | 1/2 [00:00
↪<00:00,  3.17it/s]
100%|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 2/2 [00:00
↪<00:00,  2.07it/s]


      Código de serie      Categoría de serie  ... Memo Unnamed: 13
1137        PN01205PM  Tipo de cambio nominal  ...  NaN         NaN
1138        PN01206PM  Tipo de cambio nominal  ...  NaN         NaN
1139        PN01207PM  Tipo de cambio nominal  ...  NaN         NaN
1140        PN01208PM  Tipo de cambio nominal  ...  NaN         NaN
1141        PN01209PM  Tipo de cambio nominal  ...  NaN         NaN
...               ...                    ...  ...  ...         ...
13886       PN38685FM     Resultado económico  ...  NaN         NaN
13887       PN38686FM     Resultado económico  ...  NaN         NaN
13888       PN38687FM     Resultado económico  ...  NaN         NaN
13889       PN38688FM     Resultado económico  ...  NaN         NaN
14586       PN39524FM     Resultado económico  ...  NaN         NaN

[476 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 `cutoff` 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',cutoff=1)
corriendo wordsearch: `centuria`
cutoff = 1; columnas = `all`(todas)
por favor esperar...
```

```
 0%|                                                                              | 0/14 [00:00<?, ?it/
↪s]
 7%||||||                                                                        | 1/14 [00:00<00:02,␣
↪ 4.36it/s]
14%|||||||||                                                                       | 2/14 [00:00<00:04,␣
↪ 2.45it/s]
21%|||||||||||||||                                                                 | 3/14 [00:02<00:09,␣
↪ 1.13it/s]
29%|||||||||||||||||||                                                             | 4/14 [00:02<00:08,␣
↪ 1.22it/s]
36%|||||||||||||||||||||||                                                         | 5/14 [00:03<00:06,␣
↪ 1.42it/s]
```

(continues on next page)

(continued from previous page)

```
43%|||||||||||||||||||||||||||                                          | 6/14 [00:03<00:04,␣
↪ 1.83it/s]
50%|||||||||||||||||||||||||||||||                                      | 7/14 [00:03<00:03,␣
↪ 2.24it/s]
57%|||||||||||||||||||||||||||||||||||                                  | 8/14 [00:04<00:03,␣
↪ 1.84it/s]
64%|||||||||||||||||||||||||||||||||||||||                              | 9/14 [00:05<00:03,␣
↪ 1.61it/s]
71%|||||||||||||||||||||||||||||||||||||||||||                          | 10/14 [00:05<00:02,
↪  1.96it/s]
79%|||||||||||||||||||||||||||||||||||||||||||||||||                    | 11/14 [00:06<00:01,
↪  2.22it/s]
86%|||||||||||||||||||||||||||||||||||||||||||||||||||||||              | 12/14 [00:06<00:00,
↪  2.44it/s]
93%|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||       | 13/14 [00:06<00:00,
↪  2.73it/s]
100%|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 14/14 [00:06<00:00,
↪  2.03it/s]


      Código de serie              Categoría de serie  ... Memo Unnamed: 13
8437       CD11605DA  Primera centuria independiente  ...  NaN          NaN
8438       CD11606DA  Primera centuria independiente  ...  NaN          NaN
8439       CD11607DA  Primera centuria independiente  ...  NaN          NaN
8440       CD11608DA  Primera centuria independiente  ...  NaN          NaN
8441       CD11609DA  Primera centuria independiente  ...  NaN          NaN
...              ...                                ...  ...  ...          ...
9028       CD12207DA  Primera centuria independiente  ...  NaN          NaN
9029       CD12208DA  Primera centuria independiente  ...  NaN          NaN
9030       CD12209DA  Primera centuria independiente  ...  NaN          NaN
9031       CD12210DA  Primera centuria independiente  ...  NaN          NaN
9032       CD12211DA  Primera centuria independiente  ...  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...
```

(continues on next page)

(continued from previous page)

```
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␣
```

(continues on next page)

(continued from previous page)

```
↪banco.idioma (='ing') son predeterminados, pero tambien se pueden cambiar)
banco.codigos = ['PN01273PM','PN00015MM','PN01289PM','PD39793AM']
banco.fechaini = '2011-1'
banco.fechafin = '2021-1'

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

# obtener informacion de los inputs seleccionados (arriba) en el mismo orden
df = banco.GET()

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

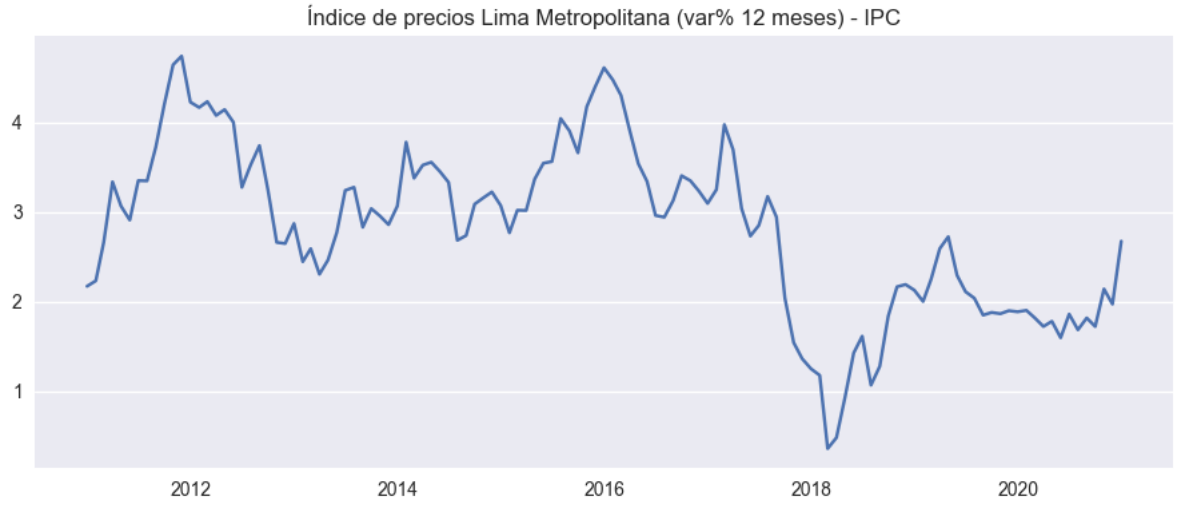
```
corriendo estado actual de todas las variables constructoras...
```

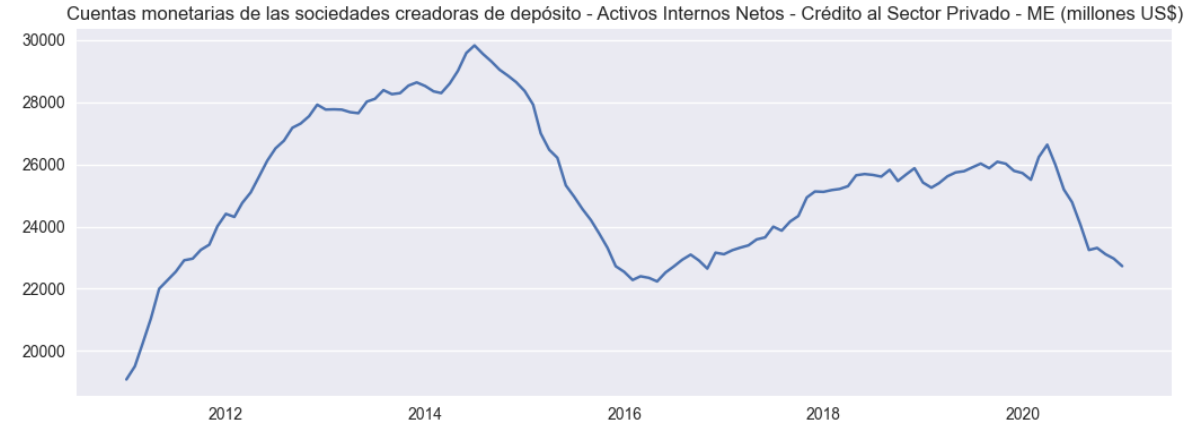
objeto.metadata	=	<class 'pandas.core.frame.DataFrame'> size: (14858, 14)
objeto.codigos	=	['PN01273PM', 'PN00015MM', 'PN01289PM', 'PD39793AM']
objeto.formato	=	json
objeto.fechaini	=	2011-1
objeto.fechafin	=	2021-1
objeto.idioma	=	ing

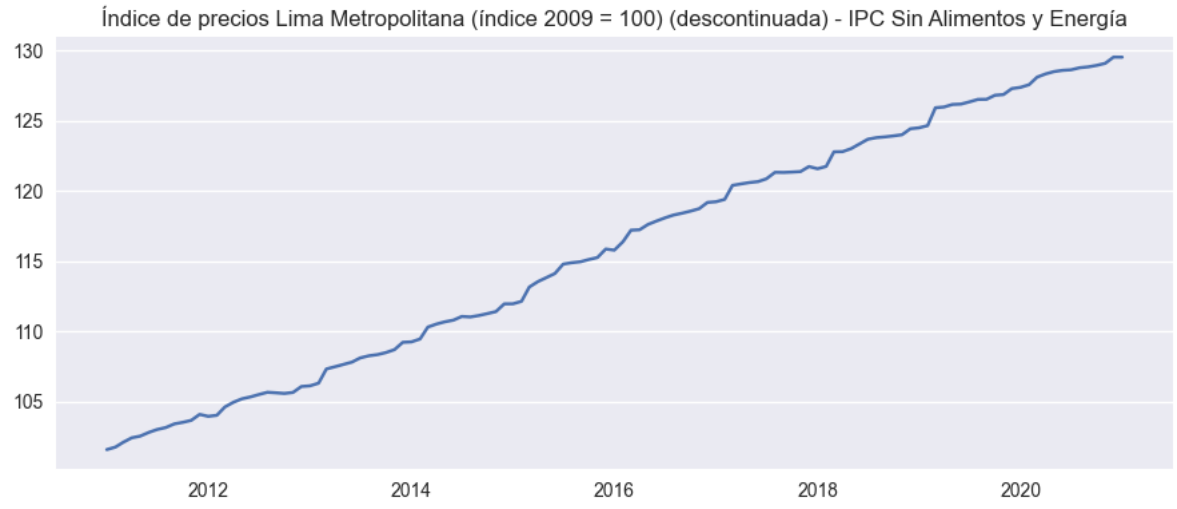
```
Orden de datos determinados por usuario:
```

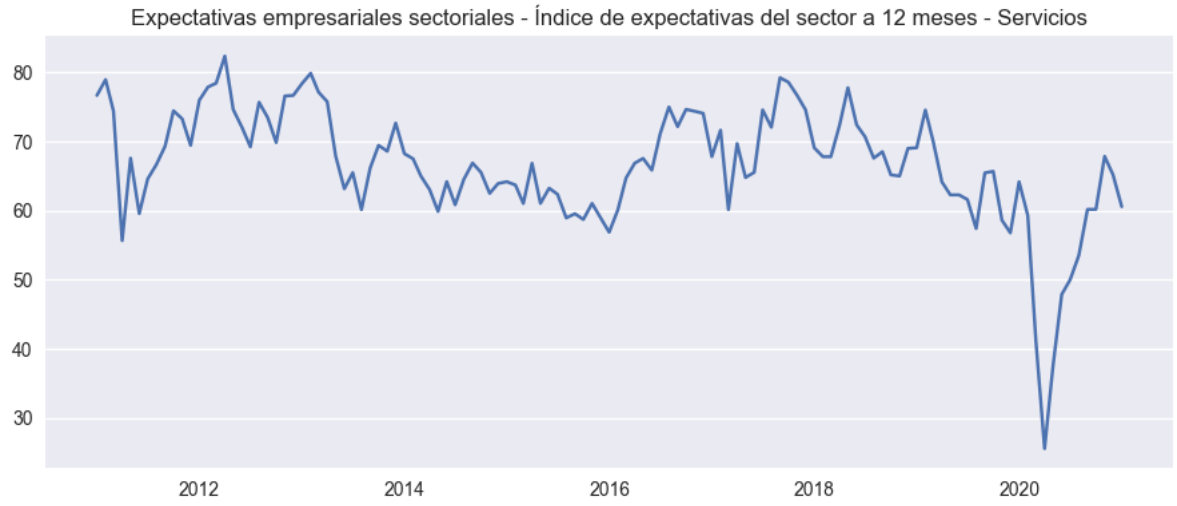
1	PN01273PM	Índice de precios Lima Metropolitana (var% 12 meses) - IPC
2	PN00015MM	Cuentas monetarias de las sociedades creadoras de depósito - Activos Internos Netos - Crédito al Sector Privado - ME (millones US$)
3	PN01289PM	Índice de precios Lima Metropolitana (índice 2009 = 100) (descontinuada) - IPC Sin Alimentos y Energía
4	PD39793AM	Expectativas empresariales sectoriales - Índice de expectativas del sector a 12 meses - Servicios

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


Índice de precios Lima Metropolitana (var% 12 meses) - IPC
4
3
2
1
2012
2014
2016
2018
2020


Cuentas monetarias de las sociedades creadoras de depósito - Activos Internos Netos - Crédito al Sector Privado - ME (millones US$)
30000
28000
26000
24000
22000
20000
2012
2014
2016
2018
2020


Índice de precios Lima Metropolitana (índice 2009 = 100) (descontinuada) - IPC Sin Alimentos y Energía
130
125
120
115
110
105
2012
2014
2016
2018
2020



El orden de las columnas en la tabla de datos `pandas.DataFrame` "`df`" ahora se colocan en el mismo orden en el cual han sido colocados por el usuario en la variable `banco.codigos` como opcion predeterminada. Si se desea usar el orden definido por BCRPData, reemplazar `banco.GET()` por `banco.GET(orden=False)`.

La identidad de los nombres de serie con sus codigos, y en si cualquier lista con *x* codigos de series, se puede consultar con una iteracion del metodo `query`, demostrado abajo:

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

```
[Out]

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
}
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",
```

(continues on next page)

(continued from previous page)

```
      "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,
      "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
}
```

1.2 Referencia API

Para los detalles de cualquier metodo de `bcrpy`, consule esta seccion o el MANUAL BCRPY en la seccion **Referencia API**. En este, se puede hacer una busqueda rapida de algun metodo con Ctrl+F y escribiendo el nombre del metodo (ejemplo: Ctrl+F "plot").

1.2.1 Metodos globales

class bcrpy.**dfsave**(*df*, *filename*, *formato='python'*)

Guarda la informacion de datos almacenados y procesados por Python en un archivo

Parametros

filename: str
Nombre del archivo para guardar

formato: str
El formato del archivo. Predeterminado="python" : archivo se guarda en formato pickle. Si el valor de formato no es "python" se guarda en formato .csv

class bcrpy.**dfload**(*filename*, *formato='python'*)

Carga la informacion de datos almacenados en un archivo a Python

Parametros

filename: str
Nombre del archivo

formato: str
El formato del archivo. Predeterminado="python" : archivo en formato pickle. Si el valor de formato no es "python" el formato es .csv

1.2.2 Metodos locales (dentro de la class *Marco*)

class bcrpy.**Marco**

GET(*filename=False*, *orden=True*)

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

Parametros

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

orden
[bool] Las columnas mantienen el orden declarados por el usuario en *objeto.codigos* con opcion *orden=True* (predeterminado). Cuando *orden=False*, las columnas de los datos es la predeterminada por BCRPData.

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

Extrae todos los metadatos de BCRPData.

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.

Parametros

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

`ordenar_columnas`(*hacer=True*)

sub-metodo para re-ordenar columnas de acuerdo a como fueron definidos en objeto.codigos (vea metodo *GET()* parametro "orden")

Parametros

hacer
[bool] ordenarlas (True)

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

Grafica x-y data.

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.

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.

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

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'*, *cutoff=0.65*, *columnas='all'*)

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.

Parametros

keyword
[str] Palabra clave para reducir los metadatos

cutoff
[float] Este es el Levenshtein similarity ratio (predeterminado=0.65). Un cutoff 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.

bcrpy

bcrpy

PYTHON MODULE INDEX

INDEX