**Tipología y Ciclo de Vida de los Datos**

**Máster de Ciencia de Datos de la UOC**

**Práctica 1**

**Enrique Javier Andrés Orera**

**Sergio Fernández Bertolin**

1. **Contexto. Explicar en qué contexto se ha recolectado la información. Explique por qué el sitio web elegido proporciona dicha información.**

La finalidad del proyecto es educativa y de investigación, no comercial. Idealista, el sitio web "raspado", es uno de los portales inmobiliarios de referencia en el contexto estatal y probablemente uno de los mayores portales de compra venta en España. Esta web pone al alcance del usuario multitud de información inmobiliaria para hacer más fácil al usuario su experiencia en la compraventa de inmuebles.

1. **Definir un título para el dataset. Elegir un título que sea descriptivo.**

Llamaremos al dataset "Euribor, precios de la vivienda y cotizaciones Ibex35"

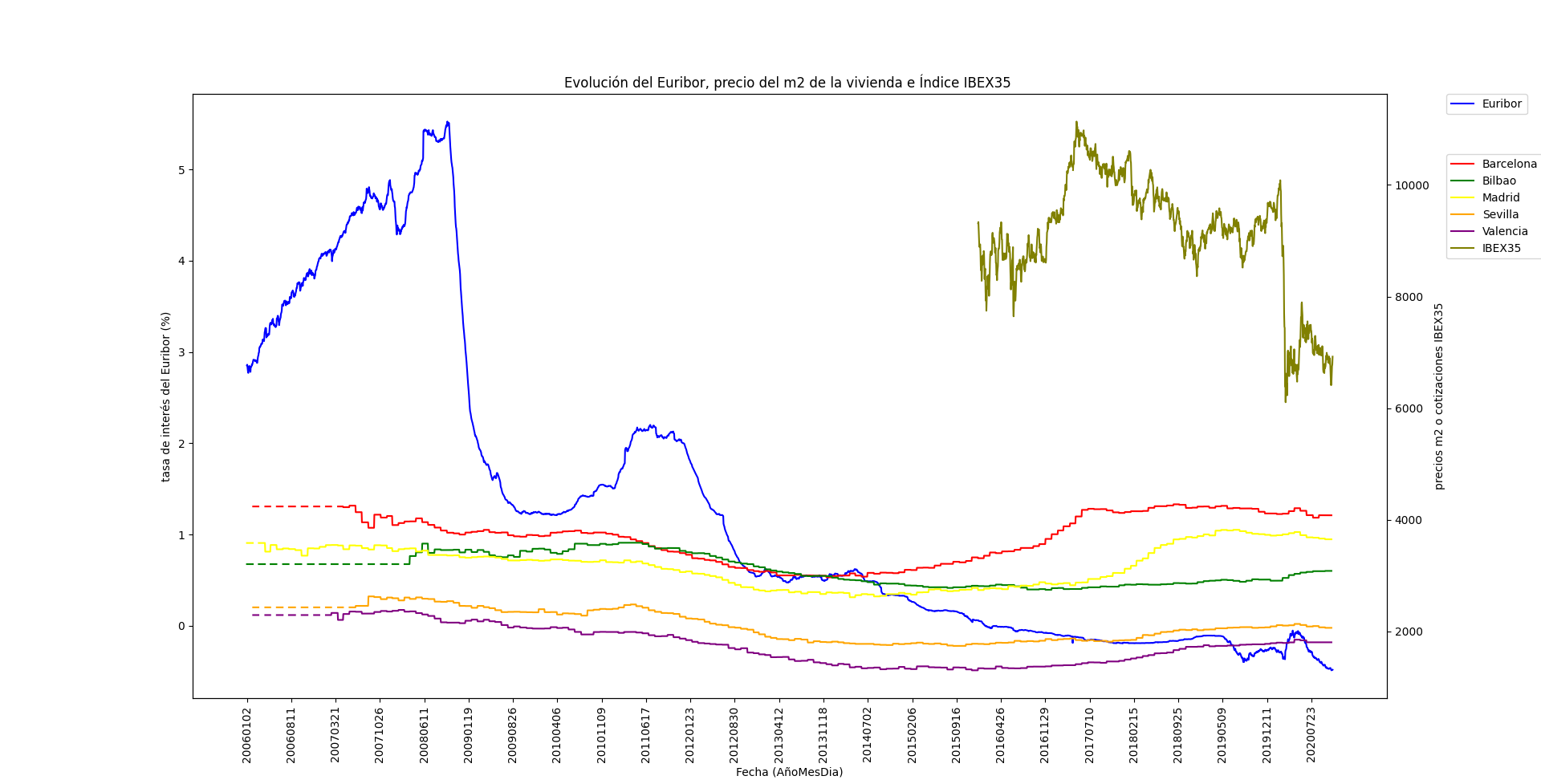
1. **Descripción del dataset. Desarrollar una descripción breve del conjunto de datos que se ha extraído (es necesario que esta descripción tenga sentido con el título elegido).**

Creación de un dataset con los valores diarios del índice del Euribor y los precios de venta medios de la vivienda/m2 desde 2006 hasta el día actual y de las cotizaciones del Ibex35, con fines educativos

Se obtienen las cotizaciones del euribor y los precios de venta medios de la vivienda con un programa en python mediante técnicas de webscraping contra el portal inmobiliario [www.idealista.com](http://www.idealista.com/)

Las cotizaciones del Ibex35 se obtienen de www.infobolsa.com

1. **Representación gráfica. Presentar una imagen o esquema que identifique el dataset visualmente**



1. **Contenido. Explicar los campos que incluye el dataset, el periodo de tiempo de los datos y cómo se ha recogido.**

El Dataset generado tiene el siguiente formato:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | Dia | Euribor | precio\_m2\_Barcelona | precio\_m2\_Bilbao | precio\_m2\_Madrid | precio\_m2\_Sevilla | precio\_m2\_Valencia | IBEX35 |
| 2473 | 20160104 | 0.058 | 3313 | 2814 | 2743 | 1778 | 1342 | 9313.200 |
| 2474 | 20160105 | 0.059 | 3313 | 2814 | 2743 | 1778 | 1342 | 9335.200 |

Donde **Index** es un valor entero auto incremental, empezando en cero

**Dia** es el dia en el que se ha registrado el valor, con el formato AAAAMMDD, donde las cuatro primeras cifras son el año, las dos siguientes el mes en formato numérico y las dos últimas el día del mes

**Valor** es el valor del índice del Euribor registrado ese día, en formato decimal

**precio\_m2\_Barcelona** es el precio de venta medio por metro cuadrado en la ciudad de Barcelona

**precio\_m2\_Bilbao** es el precio de venta medio por metro cuadrado en la ciudad de Bilbao

**precio\_m2\_Madrid** es el precio de venta medio por metro cuadrado en la ciudad de Madrid

**precio\_m2\_Sevilla** es el precio de venta medio por metro cuadrado en la ciudad de Sevilla

**precio\_m2\_Valencia** es el precio de venta medio por metro cuadrado en la ciudad de Valencia

**IBEX35** es la cotización del indice Ibex 35

El periodo de tiempo de recogida es desde el 2 de enero de 2006 hasta el 30 de octubre de 2020 y se ha recogido mediante un raspado recurrente de las páginas de idalista con la información diaria y mensual del índice Euribor, y de los precios de venta medios/m² . También del raspado del indice Ibex 35 de la pagina www.infobolsa.com

1. **Agradecimientos. Presentar al propietario del conjunto de datos. Es necesario incluir citas de investigación o análisis anteriores (si los hay).**

Idealista es uno de los portales inmobialiarios de referencia en el contexto estatal y probablemente uno de los mayores portales de compra venta en España.

Infobolsa es el portal de información financiera de BME Bolsas y Mercados Españoles.

Desconocemos la existencia de estudios previos.

1. **Inspiración. Explique por qué es interesante este conjunto de datos y qué preguntas se pretenden responder.**

En el momento actual, dentro de la mayor pandemia del siglo, hay una incertidumbre creciente en todos los aspectos financieros, de los que no queda exento el mercado inmobiliario.

Mediante esta recolección de datos se pretende obtener una herramienta que permita analizar la tendencia del principal índice inmobiliario en Europa, el Euribor, para poder realizar tanto análisis como predicciones.

1. **Licencia. Seleccione una de estas licencias para su dataset y explique el motivo de su selección: ○ Released Under CC0: Public Domain License ○ Released Under CC BY-NC-SA 4.0 License ○ Released Under CC BY-SA 4.0 License ○ Database released under Open Database License, individual contents under Database Contents License ○ Other (specified above) ○ Unknown License**

Hemos escogido la licencia **CC BY-NC-SA 4.0 License**, porque creemos que recoge adecuadamente el espiritu de la creación del dataset. Un dataset creado para un uso educativo, no comercial.

Está licencia tiene los siguientes términos:

Atribución : debe otorgar el crédito correspondiente , proporcionar un enlace a la licencia e indicar si se realizaron cambios . Puede hacerlo de cualquier manera razonable, pero no de ninguna manera que sugiera que el licenciante lo respalda a usted o su uso.

No comercial: no puede utilizar el material con fines comerciales .

ShareAlike : si remezcla, transforma o construye sobre el material, debe distribuir sus contribuciones bajo la misma licencia que el original.

Terminos extraídos de <https://creativecommons.org/licenses/by-nc-sa/4.0/>

1. **Código. Adjuntar el código con el que se ha generado el dataset, preferiblemente en Python o, alternativamente, en R.**

# -\*- coding: utf-8 -\*-

"""

Script para extraer el euribor diario de la página web de Idealista

Allí hay almacenados los datos del Euribor desde Enero de 2006 hasta

el momento actual

"""

# Importación de librerías necesarias

import locale

import datetime

import random

from bs4 import BeautifulSoup as bsoup

import requests

import calendar

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import time

from zipfile import ZipFile

from selenium import webdriver

# Necesario para calendar (configuración local a hora española)

locale.setlocale(locale.LC\_ALL, 'es\_ES.utf8')

# Creamos listas donde guardaremos los días y los valores

dias = []

valores = []

# Inicializaciones para interfaz con datos IBEX35

username\_input = '//\*[@id="login"]/ul[1]/li[1]/input'

password\_input = '//\*[@id="login"]/ul[1]/li[2]/input'

remember\_input = '//\*[@id="login"]/ul[1]/li[3]/input'

login\_submit = '//\*[@id="login"]/ul[1]/li[2]/a'

logout = '//\*[@id="ifb-menu"]/div/ul/li[2]/a[2]'

# Lista de user agents

userAgents = [

'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36',

'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.90 Safari/537.36',

'Mozilla/5.0 (Windows NT 5.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.90 Safari/537.36',

'Mozilla/5.0 (Windows NT 6.2; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.90 Safari/537.36',

'Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/44.0.2403.157 Safari/537.36',

'Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36',

'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/57.0.2987.133 Safari/537.36',

'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/57.0.2987.133 Safari/537.36',

'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36',

'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36',

'Mozilla/4.0 (compatible; MSIE 9.0; Windows NT 6.1)',

'Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko',

'Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)',

'Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko',

'Mozilla/5.0 (Windows NT 6.2; WOW64; Trident/7.0; rv:11.0) like Gecko',

'Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko',

'Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.0; Trident/5.0)',

'Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko',

'Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0)',

'Mozilla/5.0 (Windows NT 6.1; Win64; x64; Trident/7.0; rv:11.0) like Gecko',

'Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0)',

'Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; Trident/6.0)',

'Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)'

]

# Función para extraer el euribor diario

def get\_euribor(dias, valores):

# Fecha actual

now = datetime.datetime.now()

anyo = int(now.strftime("%Y"))

# Defino año final 2006, que es hasta donde tengo datos de precios

anyo\_final = 2006

mes = int(now.strftime("%m"))

controlmesactual = True

# Para cada año

for i\_anyo in range(anyo, anyo\_final-1, -1):

# Para cada mes

for i\_mes in range(mes, 0, -1):

if controlmesactual:

url = "https://www.idealista.com/news/euribor/mensual/mes-actual/"

controlmesactual = False

else:

url = "https://www.idealista.com/news/euribor/mensual/%s-%d/" % (

calendar.month\_name[int(i\_mes)], i\_anyo)

# Seleccionamos user agent aleatoriamente

userAgent = random.choice(userAgents)

# Cargamos cabeceras por defecto

headers = requests.utils.default\_headers()

# Actualizamos cabeceras con el User-Agent aleatorio

headers.update({'User-Agent': userAgent})

# Espaciado entre peticiones (2 ó 3 segundos)

sleep\_secs = random.randrange(2, 4)

time.sleep(sleep\_secs)

# Descargamos el sitio web de interés

html = requests.get(url, headers=headers)

soup = bsoup(html.content)

contador = 0

for dato in soup.body.tbody.find\_all('td'):

contador = contador + 1

if contador % 2 == 1:

# fecha

fecha = "%d%s%s" % (i\_anyo, '{:02d}'.format(i\_mes), '{:02d}'.format(int(dato.string)))

# Añadimos la fecha a la lista

dias.append(fecha)

print(fecha)

else:

# euribor

euribor = dato.string[:-1].replace(",", ".")

# Añadimos el euribor a la lista

valores.append(float(euribor))

print(float(euribor))

mes = 12

return euribor

# Función para extraer el IBEX35 diario

def precios\_ibex(dict\_ibex, anio):

url\_anio = "https://www.infobolsa.es/cotizacion/historico-ibex\_35?startDate=%d0101&endDate=%d1231" % (anio, anio)

# Opciones para el driver de Selenium

options = webdriver.ChromeOptions()

# Headless impide que el navegador Crhrome controlado por python se muestre en pantalla

options.add\_argument('headless')

# Ignoramos posibles errores

options.add\_argument('--ignore-certificate-errors')

options.add\_argument('--ignore-ssl-errors')

# Elegimos el user-agent del pool de user-agents userAgents

options.add\_argument('user-agent=%s' % (random.choice(userAgents)))

# Creamos objeto webdriver (selenium), que es el que realiza la petición con las opciones anteriormente establecidas

try:

web\_driver = webdriver.Chrome(options=options)

except:

# Si no podemos crear el objeto webdriver es porque no tenemos el driver Chrome.

# Lo descargamos y descomprimimos.

print("Descargamos y descomprimimos driver Chrome")

driver = requests.get('http://chromedriver.storage.googleapis.com/86.0.4240.22/chromedriver\_win32.zip')

with open('chrome\_driver.zip', 'wb') as d:

d.write(driver.content)

with ZipFile('chrome\_driver.zip', 'r') as zfile:

try:

zfile.extractall()

except:

print("Something else went wrong")

# Una vez descargado y descomprimido el driver Chrome, creamos objeto webdriver (selenium), que es el que

# realiza la petición con las opciones anteriormente establecidas.

web\_driver = webdriver.Chrome(options=options)

# Espaciado entre peticiones (de 10 a 15 segundos)

sleep\_secs = random.randrange(10, 15)

time.sleep(sleep\_secs)

# Tiempo de espera del driver para asegurarse de que carga la página en su totalidad, 5 segundos

seconds = 5

web\_driver.implicitly\_wait(seconds)

# Se abre una nueva petición a la página deseada

web\_driver.get('https://www.infobolsa.es/account/login')

web\_driver.find\_element\_by\_xpath(username\_input).send\_keys('sergiofbertolin@gmail.com')

web\_driver.find\_element\_by\_xpath(password\_input).send\_keys('7y.zRt47ywsi')

web\_driver.find\_element\_by\_xpath(remember\_input).click()

web\_driver.find\_element\_by\_xpath(login\_submit).click()

web\_driver.get(url\_anio)

# Cambiamos al marco

web\_driver.switch\_to.frame

# Obtenemos el marco

web\_driver.page\_source

soup\_tabla\_ibex = bsoup(web\_driver.page\_source, "lxml")

anyomes = []

ibex35 = []

contador = 0

for dato\_tabla\_ibex in soup\_tabla\_ibex.body.tbody.find\_all('td'):

contador += 1

if contador % 6 == 1:

print(dato\_tabla\_ibex.string)

a = dato\_tabla\_ibex.string

anyomes = "%s%s%s" % (a[6:10], a[3:5], a[0:2])

elif contador % 6 == 2:

print(dato\_tabla\_ibex.string)

b = dato\_tabla\_ibex.string

indice = b.replace(".", "")

indice = indice.replace(",", ".")

ibex35.append(indice)

elif contador % 6 == 0:

contador = 0

if anyomes in dias:

dict\_ibex[anyomes] = indice

# Cerramos driver

web\_driver.get('https://www.infobolsa.es/auth/bye')

web\_driver.close()

return (dict\_ibex)

# Función para extraer los precios medios por m2 de una Ciudad.

# Se trata de un contenido dinámico.

# Utilizando la libreria Selenium, podremos acceder al marco donde se encuentra la tabla que se genera de forma

# dinámica

def precios\_ciudad(dias, ciudad):

ciudad\_url\_dict = {

'Barcelona': 'https://www.idealista.com/sala-de-prensa/informes-precio-vivienda/venta/cataluna/barcelona-provincia/barcelona/historico/',

'Bilbao': 'https://www.idealista.com/sala-de-prensa/informes-precio-vivienda/venta/euskadi/vizcaya/bilbao/historico/',

'Madrid': 'https://www.idealista.com/sala-de-prensa/informes-precio-vivienda/venta/madrid-comunidad/madrid-provincia/madrid/historico/',

'Sevilla': 'https://www.idealista.com/sala-de-prensa/informes-precio-vivienda/venta/andalucia/sevilla-provincia/sevilla/historico/',

'Valencia': 'https://www.idealista.com/sala-de-prensa/informes-precio-vivienda/venta/comunitat-valenciana/valencia-valencia/valencia/historico/'}

url\_ciudad = ciudad\_url\_dict[ciudad]

# Opciones para el driver de Selenium

options = webdriver.ChromeOptions()

# Headless impide que el navegador Crhrome controlado por python se muestre en pantalla

options.add\_argument('headless')

# Ignoramos posibles errores

options.add\_argument('--ignore-certificate-errors')

options.add\_argument('--ignore-ssl-errors')

# Elegimos el user-agent del pool de user-agents userAgents

options.add\_argument('user-agent=%s' % (random.choice(userAgents)))

# Creamos objeto webdriver (selenium), que es el que realiza la petición con las opciones anteriormente establecidas

try:

web\_driver = webdriver.Chrome(options=options)

except:

# Si no podemos crear el objeto webdriver es porque no tenemos el driver Chrome.

# Lo descargamos y descomprimimos.

print("Descargamos y descomprimimos driver Chrome")

driver = requests.get('http://chromedriver.storage.googleapis.com/86.0.4240.22/chromedriver\_win32.zip')

with open('chrome\_driver.zip', 'wb') as d:

d.write(driver.content)

with ZipFile('chrome\_driver.zip', 'r') as zfile:

try:

zfile.extractall()

except:

print("Something else went wrong")

# Una vez descargado y descomprimido el driver Chrome, creamos objeto webdriver (selenium), que es el que

# realiza la petición con las opciones anteriormente establecidas.

web\_driver = webdriver.Chrome(options=options)

# Espaciado entre peticiones (de 10 a 15 segundos)

sleep\_secs = random.randrange(10, 16)

time.sleep(sleep\_secs)

# Tiempo de espera del driver para asegurarse de que carga la página en su totalidad, 5 segundos

seconds = 5

web\_driver.implicitly\_wait(seconds)

# Se abre una nueva petición a la página deseada

web\_driver.get(url\_ciudad)

# Buscamos el elemento 'iframe' que es el marco donde se encuentra la tabla dinámica que queremos obtener.

frame = web\_driver.find\_element\_by\_tag\_name('iframe')

# Cambiamos al marco

web\_driver.switch\_to.frame

# Obtenemos el marco

web\_driver.page\_source

soup\_tabla\_precios = bsoup(web\_driver.page\_source, "lxml")

anyomes = []

precio\_m2 = []

contador = 0

for dato\_tabla\_precios in soup\_tabla\_precios.body.tbody.find\_all('td'):

contador += 1

if contador % 5 == 1:

a = dato\_tabla\_precios.string.split()

for i in range(1, 13):

if calendar.month\_name[i] == a[0].lower():

anyomes.append("%s%s" % (a[1], '{:02d}'.format(i)))

elif contador % 5 == 2:

b = dato\_tabla\_precios.string.split()

if b[0] == 'n.d.':

precio\_m2.append(np.NaN)

else:

precio = b[0].replace(".", "")

precio\_m2.append(precio)

diccionario = dict(zip(anyomes, precio\_m2))

precio\_m2\_2 = []

for i in range(len(dias)):

try:

precio\_m2\_2.append(np.int(diccionario[dias[i][0:6]]))

except:

precio\_m2\_2.append(np.NaN)

# Cerramos driver

web\_driver.close()

return precio\_m2\_2

get\_euribor(dias, valores)

start\_period = 2020

end\_period = 2015

# Inicializo un diccionario con NAs para todos los días

dict\_ibex = {}

for i in range(len(dias)):

dict\_ibex[dias[i]] = np.NaN

for i in range(start\_period, end\_period, -1):

dict\_ibex = precios\_ibex(dict\_ibex, i)

ciudades = ['Barcelona', 'Bilbao', 'Madrid', 'Sevilla', 'Valencia']

# Creamos diccionario de listas con las listas días, valores, precios\_m2\_Barcelona, precios\_m2\_Bilbao,

# precios\_m2\_Madrid, precios\_m2\_Sevilla, precios\_m2\_Valencia

euribor\_dict = {'Dia': dias[::-1], 'Euribor': valores[::-1],

'precio\_m2\_' + ciudades[0]: precios\_ciudad(dias, ciudades[0])[::-1],

'precio\_m2\_' + ciudades[1]: precios\_ciudad(dias, ciudades[1])[::-1],

'precio\_m2\_' + ciudades[2]: precios\_ciudad(dias, ciudades[2])[::-1],

'precio\_m2\_' + ciudades[3]: precios\_ciudad(dias, ciudades[3])[::-1],

'precio\_m2\_' + ciudades[4]: precios\_ciudad(dias, ciudades[4])[::-1],

'IBEX35': list(dict\_ibex.values())[::-1]}

# Creamos un DataFrame para almacenar el diccionario de listas

euribor\_df = pd.DataFrame(euribor\_dict)

# Almacenamos los resultados de nuestro dataset en un csv

euribor\_df.to\_csv('euribordiario.csv')

# Representamos gráficamente la evolución temporal del Euribor

print(euribor\_df)

f, ax1 = plt.subplots()

ax1.plot(euribor\_df.index, euribor\_df.Euribor, color='blue', label='Euribor')

ax1.set(xlabel='Fecha (AñoMesDia)', ylabel='tasa de interés del Euribor (%)',

title='Evolución del Euribor, precio del m2 de la vivienda e Índice IBEX35 ')

plt.xticks(np.arange(euribor\_df.shape[0])[::150], euribor\_df.Dia[::150], rotation=90)

plt.legend(bbox\_to\_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)

# un segundo eje que comparte el eje x

ax2 = ax1.twinx()

ax2.set\_ylabel('precios m2 o cotizaciones IBEX35')

ax2.plot(euribor\_df.index, euribor\_df.precio\_m2\_Barcelona, color='red', label='Barcelona')

ax2.plot(euribor\_df.index, euribor\_df.precio\_m2\_Bilbao, color='green', label='Bilbao')

ax2.plot(euribor\_df.index, euribor\_df.precio\_m2\_Madrid, color='yellow', label='Madrid')

ax2.plot(euribor\_df.index, euribor\_df.precio\_m2\_Sevilla, color='orange', label='Sevilla')

ax2.plot(euribor\_df.index, euribor\_df.precio\_m2\_Valencia, color='purple', label='Valencia')

ax2.plot(euribor\_df.index, euribor\_df.IBEX35, color='olive', label='IBEX35')

plt.legend(bbox\_to\_anchor=(1.05, 0.90), loc='upper left', borderaxespad=0.)

plt.show()

# Almacenamos la gráfica

f.savefig("euribor.png")

1. **Dataset. Publicación del dataset en formato CSV en Zenodo (obtención del DOI) con una breve descripción.**

|  |  |
| --- | --- |
| CONTRIBUCIONES | FIRMAS |
| Investigación previa | EJAO, SFB |
| Redacción de las respuestas | EJAO, SFB |
| Desarrollo del código | EJAO, SFB |