Importación de librerías

In [ ]:
```
import pandas as pd # Dataframes management
from zipfile import ZipFile  # Files compressed management
import os # Files management along OS
import re # Expresiones regulares
```

In [ ]:
```
from os import mkdir # Comprobaciones de existencia de archivos etc.
from os.path import exists
```

In [ ]:
```
from selenium import webdriver # Webscrapping bot
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.common.exceptions import NoSuchElementException
from selenium.webdriver.firefox.options import Options
```

In [ ]:
```
import nltk # Procesamiento del lenguaje natural
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords
```

In [ ]:
```
import logging # Para generar logs
import datetime
import sys
```

Variables a modificar para adaptar el código

In [ ]:
```
# Tabla a nombrar para la BBDD
tableMain = "ZipsInfo"

# True para deshabilitar la opción de webscrapping
localScrapping = False

urlToScrap = "https://worldpostalcode.com/lookup
```

Importación de módulos

In [ ]:
```
# Módulo personal para manejar la Base de datos del proyecto
from db import *
```

Importación de datos

In [ ]:
```
# Specifying the name of the zip file
fileZIP = "/items_ordered_2years_V2.zip"
fileCSV = "/items_ordered_2years_V2.csv"

path = "../Inputs/Modificados - Atmira_Pharma_Visualization"

# Open the zip file in read mode
with ZipFile(f"{path}{fileZIP}", 'r') as zip:
    # List all the contents of the zip file
    zip.printdir()

    # Extract all files
    print('extraction...')
    zip.extractall(path)
    print('Done!')

#Import CSV to pandas
itemsOrdered = pd.read_csv(f"{path}{fileCSV}")
print("CSV imported to Pandas successfully")

# Remove uncompressed CSV file
os.remove(f"{path}{fileCSV}")
print("Original CSV removed to preserve repo health")
```

```
File Name                                             Modified
Size
items_ordered_2years_V2.csv                    2022-01-10 08:27:40    16702
0440
extraction...
Done!
CSV imported to Pandas successfully
Original CSV removed to preserve repo health
```

In [ ]:
```
itemsOrdered.head(3)
```

Out[ ]:

	num_order	item_id	created_at	product_id
0	ce30c2f02458457e3c7b563a636ae2a1	0916c05c5c3f65f59d813a78ac35c8d2	2018-11-06 16:52:13	86434
1	ce30c2f02458457e3c7b563a636ae2a1	ff323b39ae36843396d2e53ce549fb10	2018-11-06 16:52:13	87652
2	ce30c2f02458457e3c7b563a636ae2a1	199916dffc95259f4d2daab6664ca9c0	2018-11-06 16:52:13	2785

In [ ]:
```
itemsOrdered.shape
```

Out[ ]:
```
(930960, 11)
```

In [ ]:
```
#Necesario para evitar malentendidos entre librerías posteriores
del zip
```

Arreglos para facilitar el webscrapping

In [ ]:
```
itemsOrdered.loc[itemsOrdered['zipcode'].eq('30139') & itemsOrdered['city'].
itemsOrdered['zipcode'] = itemsOrdered['zipcode'].replace("29039", "28039")
itemsOrdered.loc[itemsOrdered['zipcode'].eq('33195') & itemsOrdered['city'].
itemsOrdered["city"].replace(to_replace={'Cangas Del Narcea':"Cerezaliz"}, i
itemsOrdered['zipcode'] = itemsOrdered['zipcode'].replace("088oo", "08800")
itemsOrdered.loc[itemsOrdered['zipcode'].eq('43890') & itemsOrdered['city'].
itemsOrdered.loc[itemsOrdered['zipcode'].eq('07700') & itemsOrdered['city'].
itemsOrdered['zipcode'] = itemsOrdered['zipcode'].replace("47021", "46021")
itemsOrdered.loc[itemsOrdered['zipcode'].eq('33405'), "city"] = "Raices Nuev
itemsOrdered.loc[itemsOrdered['zipcode'].eq('29720') & itemsOrdered['city'].
itemsOrdered.loc[itemsOrdered['zipcode'].eq('08222'), "city"] = "Terrassa"
itemsOrdered.loc[itemsOrdered['zipcode'].eq('03195'), "city"] = "Los Arenale
itemsOrdered.loc[itemsOrdered['zipcode'].eq('08780'), "city"] = "Palleja"
itemsOrdered.loc[itemsOrdered['zipcode'].eq('33405'), "city"] = "Raices Nuev
itemsOrdered['zipcode'] = itemsOrdered['zipcode'].replace("347007", "37007")
itemsOrdered.loc[itemsOrdered['zipcode'].eq('36194'), "city"] = "Perdecanai"
itemsOrdered.loc[itemsOrdered['zipcode'].eq('29631'), "city"] = "Arroyo De L
itemsOrdered.loc[itemsOrdered['zipcode'].eq('27810'), "city"] = "Sancobade"
itemsOrdered['zipcode'] = itemsOrdered['zipcode'].replace("-39840", "39840")
itemsOrdered.loc[itemsOrdered['zipcode'].eq('50620'), "city"] = "Casetas"
```

USO DE WEBSCRAPPING PARA CÓDIGO POSTAL

Creación de Base de Datos para almacenamiento de resultados formateados del WebScrapping

In [ ]:
```
def IntroDB():
    try:
        pathDB = "../databases/"
        nameDB= "scrapedZips.db"
        if exists(pathDB):
            print(f"Carpeta {pathDB} encontrada")
        else:
            mkdir(pathDB)
            print (f"Creada carpeta {pathDB}")
    except OSError:
        print (f"Creación de carpeta {pathDB} falló")

    if exists(f"{pathDB}{nameDB}"):
        print("Database existe")
        con, cur = SqlConnection(f"{pathDB}{nameDB}")
        return con, cur
    else:
        print("Database no existe")
        CreateCon(f"{pathDB}{nameDB}")
        con, cur = SqlConnection(f"{pathDB}{nameDB}")
        PrepareCon(con, cur, option="insert",
            values=("Test","Test","Test","Test","Test"))
        return con, cur
```

In [ ]:
```
con, cur = IntroDB()
```

```
Carpeta ../databases/ encontrada
Database existe
Conexión establecida
```

In [ ]:
```
%%capture

# Comprobación y homogeneización de datos en la BBDD
try:
    if GetThings(cur, selection="Country,Region,City,Zipcode", where=["ID",
        PrepareCon(con,cur,where=["ID",1],option="delete")
        cur.execute("UPDATE `sqlite_sequence` SET `seq` = 0 WHERE `name` = '
except IndexError:
    pass

'''
Check the next query if was wrong
SELECT * FROM `sqlite_sequence`;
'''
```

In [ ]:
```
GetThings(cur, selection="Country,Region,City,Zipcode,id_ori", where=["ID",
```

Out[ ]:
```
[('Spain', 'Madrid', 'Madrid', '28008', '328')]
```

A continuación se genera una lista compuesta de tuplas compuestas de la siguiente forma: ("Ciudad", "Zipcode")

In [ ]:
```
rawDataZipcode = list(zip(itemsOrdered["city"].tolist(), itemsOrdered["zipco
# rawDataZipcode = rawDataZipcode[:int(len(rawDataZipcode)/30)]
```

In [ ]:
```
len(rawDataZipcode)
```

Out[ ]:
```
930960
```

Funciones destacadas

Función que "limpia" los nombres de ciudades para mejorar su emparejamiento automático

In [ ]:
```
def CityCleaner(text):
    stopWordSpanish = set(stopwords.words('spanish'))
    wordTokens = word_tokenize(AcentosLimpiador(text.lower()).rstrip())
    filteredSentence = [element for element in wordTokens if not element in
    return filteredSentence
```

Función que limpia zipcodes

```
In [ ]: #Limpieza de zipcodes con RegEx
        def num_guion(string):
            """ Get a string with the numbers and hyphens of another string

            Args:
                df: string used to extract the string with numbers abd hyphens

            Returns:
                df: the string with numbers and hyphens
            """
            aux = re.match("([\d-]+)", str(string))
            try:
                return str(aux.group())
            except:
                return string


        itemsOrdered["zipcode"] = itemsOrdered["zipcode"].apply(lambda x: num_guion(
```

Función que limpia los acentos con el fin de homogeneizar

```
In [ ]: def AcentosLimpiador(text):
                acentos = {'ñ':'n','á': 'a', 'é': 'e', 'í': 'i', 'ó': 'o', 'ú': 'u',
                for ele in acentos:
                        if ele in text:
                                text = text.replace(ele, acentos[ele])
                return text
```

Configuración de logging del scrapeo

```
In [ ]: logger = logging.getLogger('ScrapLog')
        logger.setLevel(logging.INFO)
        # logger.setLevel(logging.ERROR)

        timestamp = datetime.datetime.utcnow().strftime('%Y%m%d_%H-%M-%S')
        filename=f'Scrapping{timestamp}.log'
        formatter = logging.Formatter('[%(asctime)s] %(name)s {%(filename)s:%(linenc
```

```
In [ ]: file_handler = logging.FileHandler(filename=filename)
        file_handler.setLevel(logging.INFO)
        # file_handler.setLevel(logging.ERROR)
        file_handler.setFormatter(formatter)
        logger.addHandler(file_handler)

        # stream_handler = logging.StreamHandler(sys.stdout)
        # stream_handler.setLevel(logging.INFO)
```

In [ ]:
```
logging.basicConfig(
    filename=filename,
    level=logging.INFO,
    format='[{%(filename)s:%(lineno)d} %(levelname)s - %(message)s',
    #  handlers=[
    #      file_handler,
    #      stream_handler
    # ]
)
```

Función que formatea los resultados del webscrapping de forma adecuada a los requerimientos necesarios

In [ ]:
```
def zipCodeManipulation(city, zipcode, queryResult="", saved = False):

    # Este condicional chequea si la info ya se encuentra almacenada
    if saved == False:

        listTestingZipcode = queryResult.split("\n")
        indexMatchRegex = list(map(lambda x: [(m.start(0), m.end(0)) for m i

        resultScrapClean = []
        for pas,resultScrap in enumerate(listTestingZipcode):
            for pos,ele in enumerate(indexMatchRegex[pas]):
                if len(indexMatchRegex[pas])==2:
                    if pos ==0:
                        txt = resultScrap[:ele[0]+1]+","+resultScrap[ele[1]-
                    elif pos ==1:
                        txt = txt[:ele[0]+2]+","+txt[ele[1]:]
                        resultScrapClean.append(txt)
                elif len(indexMatchRegex[pas])==3:
                    if pos ==0:
                        txt = resultScrap[:ele[0]+1]+","+resultScrap[ele[1]-
                    elif pos ==1:
                        txt = txt[:ele[0]+2]+","+txt[ele[1]:]
                    elif pos ==2:
                        txt = txt[:ele[0]+3]+","+txt[ele[1]+1:]
                        resultScrapClean.append(txt)

        resultScrapListed = [element.split(",") for element in resultScrapCl
        resultScrapRearr = [(element[0], element[1], element[-2], element[-1

        resultZip = []
        for element in resultScrapRearr:
            for element2  in CityCleaner(element[2]):
                if element2 in CityCleaner(city):
                    resultZip.append(element)
                    break
        try:
            resultZip = resultZip[0]
            resultZip = [resultZip[0],resultZip[1],resultZip[2],zipcode]
        except IndexError:
            resultZip = []
            for element in resultScrapRearr:
                for element2 in CityCleaner(element[2]):
                    for element3 in CityCleaner(city):
                        if element2 .__contains__(element3):
                            resultZip.append(element)
                            break
            try:
                resultZip = resultZip[0]
                resultZip = [resultZip[0],resultZip[1],resultZip[2],zipcode]
            except IndexError:
                resultZip = []
                for element in resultScrapRearr:
                    for element2 in CityCleaner(element[2]):
                        for city in CityCleaner(element3):
                            if element3.__contains__(element2):
                                resultZip.append(element)
                                break
                if resultZip != []:
                    resultZip = resultZip[0]
                    resultZip = [resultZip[0],resultZip[1],resultZip[2],zipc
                elif resultZip ==[] : #ELIMINAR ESTA LÍNEA PARA PODER VERIFI
                    resultZip = ["ERROR1","ERROR","ERROR",zipcode]



    elif saved == True:
        resultZip = GetThings(cur, selection="Country,Region,City,Zipcode",

    return resultZip
```

Webscrapping!

In [ ]:
```
if localScrapping ==True:
    logger.info("Starting Webscrapping!")

    opts = Options()
    opts.add_argument("--headless")
    driver = webdriver.Firefox(options=opts)

    driver.get(urlToScrap)
    driver.set_page_load_timeout(5)
else:
    logger.info("Starting LocalScrapping!")

for pos, element in enumerate(rawDataZipcode):
    try:

        if element[1]=="323903":

            if pos in [ele[0]-1 for ele in GetThings(cur, selection="ID", wh
                logger.info(f"SP - Sin procesar |City: {element[0]} | Zipcod

            else:
                if GetThings(cur, selection="Zipcode", where=["Zipcode", ele
                    PrepareCon(con, cur, values=["China", "Zhejiang", "Lishu
                    logger.info(f"SP - Insertado | City: {element[0]} | Zipc

                else:
                    zipCodeDef = zipCodeManipulation(city=element[0], zipcod
                    logger.info(f"SP - Ya guardado en BBDD | City: {element[
                    logger.info(f"Devuelto de query: {zipCodeDef}")


        if GetThings(cur, selection="Zipcode", where=["Zipcode", element[1].

                if localScrapping ==True:
                    logger.info(f"Para scrapear |Ciudad: {element[0]} | Zipc
                    insertZipcode =driver.find_element(By.ID,"search")
                    insertZipcode.clear()
                    insertZipcode.send_keys(element[1])
                    clickButtonZipcode =driver.find_element(By.CLASS_NAME,"s
                    clickButtonZipcode.click()
                    driver.set_page_load_timeout(5)
                    getGeoInfo = driver.find_element(By.CLASS_NAME,"search_u
                    saveGeoInfo = getGeoInfo.text
                    zipCodeDef = zipCodeManipulation(city=element[0], zipcod

                    logger.info(f"Scrapeado: {zipCodeDef}")

                else:
                    logger.info(f"Para scrapear, pero no es el caso en este

                    zipCodeDef = ["ERROR2","ERROR","ERROR",element[1]]
                    logger.info(f"No Scrapeado: {zipCodeDef}")

        else:
            if pos in [ele[0]-1 for ele in GetThings(cur, selection="ID", wh
                logger.info(f"Sin procesar |City: {element[0]} |Zipcode: {el
                continue
            else:
                logger.info(f"Ya guardado en BBDD  | City: {element[0]} | Zi
                zipCodeDef = zipCodeManipulation(city=element[0], zipcode=el
                logger.info(f"Devuelto de query: {zipCodeDef}")

        PrepareCon(con, cur, values=[zipCodeDef[0],zipCodeDef[1],zipCodeDef[

    except:
        logger.info(f"No encontrado nada en ciudad: {element[0]}, zipcode: {
        logger.info("--------------------------------------------")
        zipCodeDef= ["ERROR3","ERROR","ERROR",element[1]]
        PrepareCon(con, cur, values=[zipCodeDef[0],zipCodeDef[1],zipCodeDef[
        continue


driver.close()
```

Comprobaciones del scrapping y formateo posterior realizado

In [ ]:
```
tableMain = "ZipsInfo"
```

In [ ]:
```
cur.execute(f"SELECT (select count() from {tableMain}) as count FROM {tableM
```

Out[ ]:
```
1251877
```

Transformación del scrapeo formateado a dataframe de Pandas

In [ ]:
```
sqlToList = cur.execute(f"SELECT Country,Region,City,Zipcode,id_ori FROM {ta
```

In [ ]:
```
df = pd.DataFrame(sqlToList, columns=["Country","Region","City","Zipcode","i

# En este caso se observa este iloc para comprobar la información original c
df.iloc[321671:-1]
```

Out[ ]:

	Country	Region	City	Zipcode	id_ori
321671	Spain	Castilla - La Mancha	Tarazona De La Mancha	02100	0
321672	Spain	Castilla - La Mancha	Tarazona De La Mancha	02100	1
321673	Spain	Castilla - La Mancha	Tarazona De La Mancha	02100	2
321674	Spain	Castilla - La Mancha	Tarazona De La Mancha	02100	3
321675	Spain	Comunidad Valenciana	Alboraya	46120	4
...	...	...	...	...	...
1251871	Spain	Andalucia	Ubeda	23400	930954
1251872	Spain	Andalucia	Ubeda	23400	930955
1251873	Spain	Andalucia	Ubeda	23400	930956
1251874	Spain	Cataluna	Vic	08500	930957
1251875	Spain	Cataluna	Vic	08500	930958

930205 rows × 5 columns

```
In [ ]: df["Country"].value_counts()
```

```
Out[ ]: Spain                 1140522
        ERROR                   65412
        ERROR2                  26488
        Portugal                12523
        ERROR3                   2917
        Mexico                   2100
        France                    557
        Peru                      348
        United States             155
        Colombia                  152
        Czech Republic            141
        Italy                     106
        Germany                    99
        Russian Federation         76
        India                      64
        Puerto Rico                60
        Poland                     39
        United Kingdom             35
        Argentina                  13
        Belgium                    12
        Canada                     10
        China                       8
        Switzerland                 8
        Denmark                     8
        Thailand                    8
        Croatia                     6
        Chile                       4
        Austria                     4
        Algeria                     2
        Name: Country, dtype: int64
```

```
In [ ]: df.shape
```

```
Out[ ]: (930206, 5)
```

```
In [ ]: # Cierre de la base de datos
        con.close()
```

Exportación del dataframe a csv para su conservación

```
In [ ]: df.to_csv('DataScrapped.csv')
```

Comprobación Final

```
In [ ]: df2 = pd.read_csv("DataScrapped.csv")
        df2.head()
```

Out[ ]:

	Unnamed: 0	Country	Region	City	Zipcode	id_ori
0	321671	Spain	Castilla - La Mancha	Tarazona De La Mancha	02100	0
1	321672	Spain	Castilla - La Mancha	Tarazona De La Mancha	02100	1
2	321673	Spain	Castilla - La Mancha	Tarazona De La Mancha	02100	2
3	321674	Spain	Castilla - La Mancha	Tarazona De La Mancha	02100	3
4	321675	Spain	Comunidad Valenciana	Alboraya	46120	4

In [ ]:

```
#Ejecutar sólo una vez para lograr el efecto deseado (Borrar Unnamed:0)
df2.drop(columns=df2.columns[0], axis=1, inplace=True)
df2.head()
```

Out[ ]:

	Country	Region	City	Zipcode	id_ori
0	Spain	Castilla - La Mancha	Tarazona De La Mancha	02100	0
1	Spain	Castilla - La Mancha	Tarazona De La Mancha	02100	1
2	Spain	Castilla - La Mancha	Tarazona De La Mancha	02100	2
3	Spain	Castilla - La Mancha	Tarazona De La Mancha	02100	3
4	Spain	Comunidad Valenciana	Alboraya	46120	4