**Hands\_On Exploratory Data Analysis with Python**

Libro en EDA & AI.md

Lecturas obligatorias en:

Theory/pyton/ EDA & AI.md

class.me

Leer numpy, arrays y pandas

Ir a la carpeta numpy.summary

Pandas 1.

Si encontramos alguna función interesante incluirla en nuestra librería

Carpeta llamada librería o útiles. Dentro de ellas crear ficheros.py

Crear nuestra propia librería.

Crear un fichero .py en cada librería que creemos que tenga una función diferente

Un entregable este domingo y luego todos los domingos ants de las 23:59 habré que entregar uno

**Parámetros args y kwargs en Python**

**\*args**

Def nombre\_args(\*args):

print type(args)

El \* está haciendo referencia al mismo tipo de objeto

X= 2

Y = [“g”, “h”]

Podemos pasar todos los argumentos como

nombre\_args (x,y,9,”j”)

devuelve una tupla con todos los parámetros incluidos

le decimos que incluya todos los parámetros como tupla

def realiza\_suma\_numeros (x, \*args):

print(“x”: x)

print (args)

realiza\_suma\_numeros (7, 222, 999, -4549)

recoge el 7 como x y el resto como tupla

no se puede asignar ningún nombre de variable

arg es una tupla de todos los valores que no estén asociados a ningún argumento obligatorio (x=2)

**\*\*kwargs**

def función\_kwargs (\* \*kwargs):

print type(kwargs)

print (kwargs)

función\_kwards (x=2, p=3)

class dict

{“x”:2, “p”: 3}

no permite llamar a la función sin pasar nombres de variables

es un diccionario

incluye como kwargs todos los valores que no estén definidas en la función

es independiente el orden en el que se coloquen las variables al llamar a la función

es obligatorio al llamarlo dar un nombre de variable

def función\_kwargs (x, b=4, \*\*kwargs):

print type(kwargs)

print (kwargs)

función\_kwards (2, -1, p=3, s=[-8,-9])

si se utilizan juntos args y kwargs siempre ha de ir primero \*args

def función\_args\_kwargs (x, b=4, \* args, \*\*kwargs):

función\_args\_kwargs(1)

x:1

función\_args\_kwargs(x=1, k=2)

x:1

\*\*kwargs: {“k”: 2}

d={5:”po”}

función\_args\_kwargs (1, 2,3, 999, d, d=d, j=6)

x:1

b:2

\*args : (3,999,{5:”po”})

\*\*kwargs: {“d”: {5:”po”}, j:6}

**global\_local**

x=99

def función ():

global x # llama a la variable de fuera

x = 2 # a la variable de fuera que se ha llamado anteriormente se le asigna un nuevo valor que es 2

return 2

print (x)

2

X=99

L1 = [0,1,2]

def función (lista):

global x # llama a la variable de fuera

x = 2

l1 = lista.append(“A”) # como no tiene global l1 previamente no me puede hacer nada sobre la función de fuera

return 2

función(lista=l1)

también se pueden modificar los valores de variables de fuera de la función sin necesidad de utilizar global

lectura obligatoria: EDA % AI.md

dentro de class cap 3 y 4

en el explorador de Windows poner cmd, seleccionar ejecutar como administrador, abre la pantalla del administrador y poner pip3

pip3 install numpy

si sale un warning no hacemos nada. Lo que hace es actualizarnos la versión

pip3 install matplotlib

pip3 install scikit-image # permite leer imágenes

**NUMPY**

Tiene una clase list

Class List:

def \_\_init\_\_(self):

self.lista = []

def appeng (self,x):

self.lista.append(x)

l = List()

l.append(2)

print(type(l)

print(l.lista)

<class ‘\_\_main\_\_.List’>

2

Numpy es una librería en la que vamos a utilizar un tipo de array

Import numpy as np

Lista = []

arrray = np.array[] # array es una función que viene en la librería numpy.L o que hace es transformar una lista vacía en array

array

arrray = np.array ([2,5,1])

Crear una matriz a partir de diferentes listas incluidas en una lista

l = [[1,2], [3,4],[5,6]]

for x in l:

print (x)

output:

[1,2]

[3,4]

[5,6]

Dimensiones de una matriz: nxm; filas x columnas

matrix = [[1,2], [3,4],[5,6]]

matriz\_array = np.array(matrix)

len(matrix) # nº de filas

len(matrix[0]) # nº de columnas

matrix\_array\_shape # nos da el número de filas y columnas de una matriz (3,3)

se pueden representar 3 dimensiones) Una de tres dimensiones sería (3,3,3)

import matplotlib.pyplot as plt

imagen = inread(“

(433, 1000, 3)

len(imagen[0])

1000

imagen[0].shape

(1000,3)

Las imágenes están hechas con tres dimensiones:

El negro corresponde al 0

El 255 es el blanco absoluto [0][0][0]

Representar el amarillo

Image[300

Imagen aparece un array. Nos da tantos corchetes como dimensiones tiene

Zona amarilla

Print (image[200, 290, :])

nuevo\_color = np.array([0,0,0])

# modificar un color imagen2 = inread("./img/python.jpg") nuevo\_color = np.array([0, 255, 0]) imagen2[300][:] = nuevo\_color plt.imshow(imagen2)

Para quitar los ejes

plt.axis(“off”)

imagen2 = inread(“./img/Python.jpg”)

numpy.uint8 # son versiones de float que indica que se está utilizando un entero que se va a almancenar en 8 bits

si tenemos que modificarlo, poner el de 64 bits

Escalar: x = 2

Vector: v = np.array ([2,3,5])

v.shape

(3, )

Matrix = np.array ([2,3,5], [2,3,5], [2,3,5])

Tensor = np.array ([[[2,3,5], [2,3,5], [2,3,5])

**pandas**

es la librería para trabajar como si fuera un excel

pandas trabaja con numpy

panda recoge los datos para trabajar con ellos con filas y columnas

permite cargar ficheros sin necesidad de incluir toda la ruta

csv # commas separate values

import pandas as pd

df = pd.read\_csv(“seattle2014.csv”)

df # data frame

pasa un fichero csv a un formato de filas y columnas

cada columna es una serie. Cada serie están indexada por fila

Mirar los links que envían por discord. Coger los links para ir mirándolos. Son páginas con buena información sobre Python

Identificar la ruta en la que está el archivo. Mirar archivo paths/class.md

Si le doy toda la ruta es una lectura absoluta

My folder = my\_folder +

Si estoy en la misma carpeta del archivo

Si no estoy en la carpeta donde está el archivo y está en una carpeta superior a aquella en la que estoy:

os..path-dirname(os.getcwd()) + os.sep + “laliga.csv” # subo una carpeta a mi carpeta actual (en la que estoy) o

df = pd.read.csv(“../seattle2014.csv”) # se le indica que suba una carpeta en la misma ruta en la que me encuentro al indicarle “../”

si queremos subir varias carpetas se vuelve a repetir “../” tantas veces como carpetas queremos subir

df = pd.read.csv(“../../seattle2014.csv”)

Esta en una carpeta superior y dentro de otra carpeta incluida en la primera carpeta

df = pd.read.csv(“paths/seattle2014.csv”) # Si estoy dentro de la carpeta en la que está el archivo no hace falta que suba una carpeta. Directamente le doy el nombre de la carpeta en la que está

df = pd.read.csv(“../paths/seattle2014.csv”) # se sube una carpeta y se indica el nombre de la carpeta en la que está incluido (paths)

**Paths. Python paths**

Siempre vas a poder importar desde cualquier archivo que esté en el archivo en el que estés con sys.path (es una lista en la que están todas las rutas que va a tener en cuenta Python cuando hagamos import). La ruta absoluta funciona seguro porque es la ruta de nuestro ordenador.

os.path.dirname(python\_path).- nos da la ruta hasta la raíz del proyecto que nos interese

utilizar la función get\_root\_path(n=5)

n=5 indica el número de carpetas que hay desde donde estoy hasta la carpeta en la que me quiero situar

con ipynb siempre hay que indicarle una n más para subir carpetas porque con \_\_file\_\_ va al fichero

Para importar algo por encima de donde estamos hay que situar a la ruta en una carpeta anterior de aquella que queremos importar

En Windows hay que escribir doble barra aunque aparezca una sola.

Para abrir un archivo .csv: read\_csv y ya lo lee como un data frame. Cada una de las columnas de un data frame es una serie

De la lista de ejemplos de ficheros .csv que hay en internet podemos pinchar botón derecho y recoger la ruta. Esta ruta la podemos colocar como nombre del csv que queremos que nos pase a Data Frame

Para almacenar este fichero podemos hacer:

df.to\_json(“nombre.json”)

o en csv

df.to\_csv(“nombre.csv”)

o en Excel

df.to\_excel(“nombre.xlsxl”)

from datetime import datetime as dt

for i in range(3):

day = dt.now().day

month = dt.now().month

year = dt.now().yearde

s = str(day) +”/” + sr(montd) + “/” + sr(year)

print(s)

**JSON. Mirar semana 3 día 5 y semana 4 dia 3**

No es un archivo local sino que está en internet. Tenemos que poder traer ese archivo a nuestro ordenador para poder trabajar con él

Import requests # trae el fichero de internet

Import json # librería que trabaja con json

Import pandas as pd # librería para data mining/data wrangling

Para tenerlo en local el fichero

with open (‘data\_indented.json’, ‘r+’) as outfile:

json\_readed = json.load(outfile)

pd.read\_json(‘data indented.json’)

para recoger la base completa. Members incluye todos los diccionarios incluidos en el json

df\_members\_json = pd.DataFrame(json\_readed [‘members’])

se pueden unir dos data frame con la función concat

para darle formato adecuado al json

después de pasarlo al data frame se puede guardar como un fichero ‘.csv’

df\_f.to\_csv(‘nombre del archivo.csv’)

o a Excel

df\_f.to\_excel(‘nombre del archivo.xls’)

Para formato del data frame. En visual estudio le pinchamos en los cuadraditos hay una que es data preview.

csv to table

material theme

material icon theme

**iterabales y generadores**

crear una lista de números que tienda al infinito

def generate\_range(n):

lista = []

for i in range(n):

lista.append(i)

return lista

generador(tipo de variable): es una función que da x números cuando se le pidan. No da directamente toda la lista sino número a número según se lo pedimos .Ocupa menos memoria en caso de trabajar con grandes números. Es más eficiente ir pidiendo un número determinado de ellos y aplicar la función que nos interese e ir consecutivamente con el resto de números

def generate\_range(n):

i= 0

while i<n:

yield i # every call to yield produce a valor of the generator

i +=1

for i in generate\_range(100):

print(f’

el generador funciona como un iterable normal

El **yield** es el que le dice a la función que es un generador

for i in range(n):

print (next(generator)) # next es una función que hace que nos de un número cada vez que clicamos (que se lo pedimos)

si se coloca un return al activar la función cuando llega al return se inicia la función desde el contador (i=0) y no desde la condición iterable (while i<n)por lo que siempre retornará el inicio del contador

**Type annotations**

df total(xs:lista) -> float:

return(xs)

la función tiene un parámetro que es de tipo lista. Otras

En versiones anteriores a 3.9 si queremos que una lista sea de float hay que importar una función de la librería general

Esta va a ser la forma habitual de escribir código de Python. Decir de que tipo es la variable

**APIS**

Técnica para obtener datos de la web

http es un protocolo de internet de intercambio de hipertexto. Transmisión de datos, documentos, imágenes o vídeos.

Pieza de software intermedia que permite que dos aplicaciones se hablen entre ellas.

La API es un Json

Se puede acceder de forma libre a algunas APIS. Otras son de pago

Utilizaremos el json normalize para poder leerla como un dataframe

API sin token

Limit 🡪 el número que aparecen en la página

Con las claves del diccionario se puede acceder a lo que aparece en valor de esa clave

Offset🡪 si es el 0 empieza desde al ‘0’ si se pide desde el 40 empieza desde el 40

pokemon🡪 es el endpoint. Situado justo antes del ?

?--> a partir de aquí empiezan los metadatos

Con next le decimos que nos traiga lo siguiente con la pauta dada en la parte de metadatos.

Si hay limitaciones para traerlos todos, una forma de hacerlo es poner un límite muy alto. Otra forma es acceder con answer[‘results’]

df2 = pd.DataFrame(answer[‘results’]

df2

con el loop while de Python podemos limitar también la información que nos presenta en pantalla cada vez. Decirle que mientras la clave next no sea None continue haciendo llamadas de 20 en 20 (lo establecido por defecto). Esto es más eficiente que estar realizando llamadas de 20 en 20. Con una sola llamada podemos traer la información relevante que nos interesa.

Wall time 🡪 mide el tiempo que tarda en ejecutarse

Una forma de saber que ya no hay más información es ver que ‘next’ es None

Results🡪 es una lista con los resultados (diferentes diccionarios)

Podemos con Python transformar la información que nos de la API con el formato que nos venga bien

Con df.T🡪 lo que hace es transponer el data frame. Cambiar las filas por columnas

Obtenemos lo que hace un pokemos específico con

for nombre\_pokemon, url in diccionario\_pokemon\_url.items():

petición\_api\_url = requests.get(url).json()

podemos guardar toda la información de la API en un archivo local con formato .csv para poder trabajar después con él

como hacer que un archivo que no queremos que se traslade a git hub podemos crear un archivo gitnore. Ahí podemos meter todos los archivos o carpetas que no queremos que aparezcan en el repositorio de git hub. Si no queremos que un archivo que tenga la palabra password ponemos directamente \*password\*

si quitamos el asterisco de la derecha quiere decir que

discord/class file descargar el archivo que hay settings\_password.

Cuando ejecutemos este archivo, moverlo a la carpeta de Apis del día de hoy(10-05-2021)

Dentro de API, Api token

Api foursquare

Getting started

Con .value puedo cambiar un data frame o una serie a un array

Pasar de json a diccionario con json.load(‘nombre’)

Para identificar elementos de unos datos con ‘: ‘ es un data frame. Si no le ponemos los dos puntos es una Serie

Las claves de un dataframe son las columnas. Las claves de una serie es el índice (start :, stop: , step: )

df.loc [2:10,’response’ : ] # para transformar una columna en data frame

otra forma es:

pd.DataFrame df.loc [2:10,’response’ : ]

settings