**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 matplolib.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 = imread("./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 = imread(“./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**

pandas trabaja con numpy

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

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