## Variables ampliadas por text (CONCATENATION)

**Python Cheat Sheet 1** 

Para encadenar texto

categoria1 = "verde" color\_detalle = categoria1 + ' ' + 'oscuro'

print(categoria1 + ' oscuro') print(categoria1, 'oscuro') type() and isinstance()

float/int/str(variable) cambia el tipo de data/type type(variable) devuelve: class 'float/int/str'

isinstance(variable, float/int/str) comprobar el tipo de dato (devuelve True/False)

## / dividir

**Operaciones Algebraicas** 

restar // divider y redondear (modulus) % resto de una division (floor multiplicar \*\* elevar division) round(x) redondear número x

## **Operaciones Binarias**

== comprobar si valores coinciden is comprobar si valores son exacamente igual != comprobar si valores son diferentes is not comprobar si valores no son exactamente iguales > (>=) mayor que (mayor o igual que) < (<=) menor que (menor o igual que)</pre>

and ambas verdaderas or ambas o solo una verdadera in/not in comprobar si hay un valor en una lista etc.

**Metodos String** 

string.upper()z MAYUSCULAS string.lower() minusculas string.capitalize()
Primera letra de la frase en may. string.title() Primera Letra De Cada Palabra En May. string.swapcase() mINUSCULAS A mAYUSCULAS O vICEVERSA string.strip() quita espacios del principio y final

string.split() divide string en lista - por espacios por defecto, o especifica otro divisor en () string.replace("frase", "frase") remplaza la primera frase del string por el otro

".join(string) une los elementos de una lista en una string con el separador espificado en " " list(string) convierte un variable string en una lista

string.find("substring") encuentra el indice en que empiece el substring/'-1' si no existe el substring

string[i] devuelve el elemento en la indice i string[i:j] devuelve un rango de caracteres

len(lista) devuelve el no. de elementos

Listas [ ] Metodos no permanentes

lista = [] crea una lista vacia

min(lista)/max(lista) saca el valor minimo y maximo

lista.count() devuelve el no. de elementos que hay en la lista de un valor determinado en los()

<mark>sorted(lista)</mark> ordenar una lista de menor a mayor lista.copy() hacer una copia de la lista

Metodos con indices

list.index(x) devuelve la indice de x en la lista lista[i] devuelve el elemento en la indice i [start:stop:step] lista[i:j:x] devuelve los elementos por el rango de i a j (incluye i pero no j) saltando por x lista[-i:-j] devuelve los elementos por los indices negativos (incluye -j pero no -i)

Listas – Acciones Permanentes

string, integer o tuple) a la lista

Ampliar una lista [lista1, lista2] junta listas pero se mantienen como

listas separadas lista1 + lista2 hace una lista mas larga

.append() lista.append(x)# añade un solo elemento (lista,

.extend() lista.extend(lista2)# añade los elementos de una lista al final de la lista

.insert()

.insert(i, x)# mete un elemento (x) en un índice(i)

Ordenar una lista .sort()

lista.sort()# ordena de menor a mayor, usar con (reverse=True) para ordenar de mayor a menor lista.reverse()# ordena los elementos al reves del orden guardado

Quitar elementos de una lista

.pop()

lista.pop(i)# quita el elemento en indice i y devuelve su valor .remove()

lista.remove(x)# quita el primer elemento de la lista

con valor x lista.clear()# vacia la lista

del lista# borra la lista

del lista[i]# borra el elemento en indice i

variable = dict(x=y, m=n) crear un diccionario

dict()

dicc.copv() crear una copia

diccionario =  $\{x:y\}$  compuestos por un key(x) unica

len(dicc) devuelve el no. de elementos (x:y) hay en el diccionario

Diccionarios { key : value , }

y un valor(y) (cualquier tipo de datos)

sorted(dicc) ordena los kevs: usar con .items() para ordenar tuplas de los elementos o .values() para ordenar los values solos

Diccionarios – Metodos

Obtener informacion de un diccionario dicc.keys() devuelve todas las keys dicc.values() devuelve todos los values dicc.items() devuelve tuplas de los key:value in/not in comprobar si existe una clave dicc.get(x, y) devuelve el valor asociado al key x, o si no existe devuelve el output y dicc["key"] devuelve el valor del key (ver abajo que tiene mas usos)

Ampliar un diccionario .update()

dicc.update({x:y})# para insertar nuevos elementos dicc["key"] = valor# para inserter un nuevo key o

valor, o cambiar el valor de un key dicc. setdefault(x, y)# devuelve el value del key x, o si no existe la key x, la crea y asigna el valor y por defecto

Quitar elementos de un diccionario

dicc.pop(x)# elimina la key x (y lo devuelve) dicc.popitem()# elimina el ultimo par de key:value dicc.clear()# vacia el diccionario

tupla = (x,y) tuplas se definen con () y , o solo ,

Tuplas (,) inmutables, indexados

tupla1 + tupla2 juntar tuplas

tuple(lista) crear tuplas de una lista tuple(dicc) crear tuplas de los keys de un diccionario

len(tupla) devuelve el no. de elementos

convertirla en una lista y luego a tupla\*

tuple(dicc.values()) crear tuplas de los values tuple(dicc.items()) crear tuplas de los key:values

in/not in comprobar si hay un elemento tupla.index(x) devuelve el indice de x tupla.count(x) devuelve el no. de elementos con valor x en la tupla \*para cambiar el contenido de una tupla hay que listzip.sort() ordena las tuplas del zip por el primer elemento

zip(iterable1, iterable2) crea una lista de tuplas de

parejas de los elementos de las dos listas (mientras se

Sets {} no permiten duplicados, no tienen orden

 $set = \{x,v\}$ set(iterable) solo permite un argumento iterable; elimina duplicados

zip()

in/not in comprobar si hay un elemento

len(set) devuelve el no. de elementos

Ampliar un set set.add(x)# añadir un elemento

[] o {} o un variable tipo lista o set Quitar elementos de un set

set.update(set o lista)# añadir uno o mas elementos con

set.pop()# elimina un elemento al azar

set.remove(x)# elimina el elemento x set.discard(x)# elimina el elemento x (y no devuelve error si no existe) set.clear()# vacia el set

**Operaciones con dos Sets** 

set1.union(set2) devuelve la union de los dos sets: todos los elementos menos dupl. set1.intersection(set2) devuelve los elementos comunes de los dos sets

set1.difference(set2) devuelve los sets que estan en set1 pero no en set2 (restar) set1.symmetric difference(set2) devuelve todos los

elementos que no estan en ambos set1.isdisjoint(set2) comprobar si todos los elementos de dos sets son diferentes

set1.issubset(set2) comprobar si todos los elementos de set1 estan en set2 set1.superset(set2) comprobar si todos los elementos de

• permite obtener texto escrito por teclado del usuario

input("el texto que quieres mostrar al usuario")

input()

set2 estan en set1

· se puede guardar en un variable

 por defecto se guarda como un string x = int(input("escribe un número") para usar el variable

como integer o float se puede convertir en el variable

esta debajo del if. \*tiene que estar indentado\* elif para chequear mas condiciones después de un if else agrupa las condiciones que no se han cumplido; no puede llevar condiciones nuevas if x > y: print("x es mayor que y")

if estableca una condición para que se ejecute el código que

elif x == v: print("x es igual que y") else: print("x e y son iguales")

Sentencias de control

if ... elif ... else

repite el código mientras la condición sea True, o sea se

parará cuando la condición sea False se pueden incluir condiciones con if... elif... else \*pueden ser infinitos\* (si la condición no llega a ser

False) while x < 5:

print("x es mayor que 5")

For loops

que tiene que ser un iterable (lista, diccionario, tupla, set, or string) se pueden combinar con if ... elif ... else, while, u otro

• sirven para iterar por todos los elementos de un variable

en diccionarios por defecto intera por las keys; podemos usar dicc.values() para acceder a los values

for i in lista: print("hola mundo")

List comprehension

su principal uso es para crear una lista nueva de un un for loop en una sola línea de codigo [ lo que queremos obtener iterable condición (opcional) ]

try ... except

en el código. Se puede imprimir un mensaje que avisa del error. try: print("2.split()) except:

Se usan para evitar que nuestro código se pare debido a un error

print("no funciona")

range()

nos devuelve una lista de números que por defecto se aumentan de uno en uno empezando por 0 range(start:stop:step)

se puede especificar por donde empieza y el limite (que debe

ser +1 por que se para uno antes del limite que ponemos como tambien se puede especificar saltos

# metodos permanentes (cambia el variable, no devuelve nada)

Python Cheat Sheet 2	Regex	Modulos/Librerias (paquetes de funciones)	Ficheros xml	MySQL Connector/Python
Funciones	<ul> <li>una abreviatura de `expresión regular`,     `regex` es una cadena de texto que permite     crear patrones que ayudan a emparejar,</li> </ul>	Importar y usar modulos y sus funciones import modulo from modulo import funcion from modulo import funcion	<pre>import xml.etree.ElementTree as ET importa la librería xml variable_tree = ET.parse('ruta/archivo.xml') abre el archivo</pre>	Obtener resultados de una query  variable_cursor.fetchone()  devuelve el primer resultado
Definir una funcion:	localizar y gestionar strings	modulo.funcion() usar una funcion de un modulo	<pre>variable_root = variable_tree.getroot() que envuelve todo (el elemento raíz) en una lista</pre>	<pre>variable_cursor.fetchall() como iterable - cada fila es una tupla</pre>
<pre>def nombre_funcion(parametro1, parametro2,):     return valor del return</pre>	import re para poder trabajar con regex	<pre>modulo.clase.funcion() import modulo as md asignar un alias a un modulo</pre>	<root></root>	·
return varor_ucr_retain	Operadores communes de regex		<pre><child_tag atributo1="valor" atributo2="valor"></child_tag></pre>	Pandas dataframe with SQL
Llamar una funcion:	+ coincide con el carácter precedente una o más veces	Libreria os os.getcwd() devuelve la ruta de donde estamos trabajando; se	<pre></pre>	import pandas as pd
<pre>nombre_funcion(argumento1, argumento2,)</pre>	* coincide con el carácter precedente cero o	puede guardar en un variable e.g. ruta = os.getcwd()		<pre>variable_df = pd.DataFrame(variable_resultado_fetchall, columns = ['columna1', 'columna2',]) crear un</pre>
return: es opcional, pero sin return devuelve None	más veces u opcional	os.listdir() devuelve una lista de los archivos y carpetas	<pre>variable_root.tag devuelve el nombre del tag del raiz variable root.attrib devuelve los atributos del fichero</pre>	dataframe con los resultados de una query en una variable
parametros por defecto: - siempre deben ser lo	indica cero o una ocurrencia del elemento	donde estamos trabajando <pre>os.listdir('carpeta')</pre> devuelve los contenidos de otra carpeta		<pre>variable_df.head(n) devuelve las n primeras filas del df,</pre>
ultimo	<pre>precedente . coincide con cualquier carácter individual</pre>	os.chdir('ruta') cambia la carpeta en la que estes	<pre>variable_root.find("tag").find("childtag").text la primera ocasión en que el tag de un elemento coincida</pre>	o 5 por defecto
*args: una tupla de argumentos sin limite	^ coincide con la posición inicial de cualquier	<pre>os.mkdir('nueva_carpeta') os.rename('nombre_carpeta', 'nueva_nombre') cambia el nombre</pre>	con el string	<pre>variable_df = pd.read_sql_query(variable_query, variable_cnx) convertir los resultados de la query en df</pre>
**kwargs: diccionarios cuyas keys se convierten en parámetros y sus valores en los argumentos de los	string	de una carpeta	<pre>variable_root.findall("tag").findall("childtag").text devuelve todos los elementos cuyos tag coincide</pre>	<pre>pd.read_sql(variable_query, variable_cnx)</pre>
parámetros	\$ coincide con la posición final de cualquier	os.rmdir('carpeta') borra la carpeta	devactive codos tos elementos cayos cag connetae	variable_df.to_csv("nombre_archivo.csv") guardar en csv
<pre>def nombre funcion(parametros, *args, **kwargs,</pre>	string	Libreria shutil	MySQL Connector/Python	<pre>variable_df.to_string() formatear el dato en string</pre>
parametro_por_defecto = valor)	Sintaxis básica de regex	from shutil inmport rmtree	Competent a super hand de datas	<pre>variable_df.to_latex()</pre> formatear el dato en un string que
arg/kwarg: sin */** dentro de la funcion	w cualquier caracter de tipo alfabético	rmtree('carpeta') borra la carpeta y subcarpetas	Conectar a una base de datos	facilite la inserción en un documento latex
arg[0]	<pre>\d cualquier caracter de tipo númerico \s espacios</pre>	Abrir y cerrar ficheros	<pre>import mysql.connector para importar MySQL Connector</pre>	Crear y alterar una base de datos
Llamar una funcion con *args:	\n saltos de línea	Primero hay que guardar la ruta del archivo:	<pre>pip install mysql-connector pip install mysql-connector-Python</pre>	<pre>variable_cursor.execute("CREATE DATABASE nombre_BBDD")</pre>
<pre>nombre_funcion(argumento, argumento,)</pre>	\₩ cualquier caracter que no sea una letra	ubicacion_carpeta = os.getcwd() nombre archivo = "text.txt"	<pre>connect() para conectar a una base de datos:</pre>	variable_cursor.execute("CREATE TABLE nombre_tabla
<pre>nombre_funcion(*[lista_o_tupla_de_args])</pre>	<b>\D</b> cualquier caracter que no sea un dígitos	<pre>ubicacion_archivo = ubicacion_carpeta + "/" + nombre_archivo</pre>	<pre>variable_cnx = mysql.connector.connect(user='root',</pre>	(nombre_columna TIPO, nombre_columna2 TIPO2)")
	\S cualquier elemento que no sea un espacio	<pre>f = open(ubicacion archivo) abrir un archivo en variable f</pre>	<pre>password='AlumnaAdalab', host='127.0.0.1',</pre>	<pre>variable_cursor.execute("ALTER TABLE nombre_tabla ALTERACIONES")</pre>
Llamar una funcion con **kwargs:	() aísla sólo una parte de nuestro patrón de búsqueda que queremos devolver	f.close() cerrar un archivo * IMPORTANTE *	database='nombre_BBDD')	ALTERACTORES
nombre_funcion(**diccionario)	[] incluye todos los caracteres que queremos	with open(ubicacion_archivo) as f:	<pre>from mysql.connector import errorcode importar errores</pre>	Insertar datos
	que coincidan e incluso incluye rangos como	<pre>codigo e.g. variable = f.read() abre el archivo solo para ejecutar el codigo indicado (y despues lo deja)</pre>	<pre>mysql.connector.Error se puede usar en un try/except</pre>	variable_query = "INSERT INTO nombre_tabla (columna1,
Clases	este: a-z y 0-9		<pre>cnx.close() desconectar de la base de datos</pre>	columna2) VALUES (%s, %s)"  variable valores = (valor1, valor2)
	es como el operador 'or' \text{\ señala una secuencia especial ( escapar)}	Encoding	Realizar queries	variable_cursor.execute(variable_query, variable_valores)
Definir una clase:  class NombreClase:	caracteres especiales)	<pre>from locale import getpreferredencoding getpreferredencoding() para saber que sistema de encoding</pre>	<pre>variable_cursor = cnx.cursor() crear el objeto cursor que</pre>	otro método:
CLASS NOIDD ECLASE.	{} Exactamente el número especificado de	estamos usando	nos permite comunicar con la base de datos	variable query = "UPDATE nombre tabla SET nombre columna =
<pre>definit(self, atributo1, atributo2):</pre>	ocurrencias <pre>{n} Exactamente n veces</pre>	<pre>f = open(ubicacion_archivo, encoding="utf-8") y leerlo con el encoding usado; guardar con .read()</pre>	variable_cursor.close() desconectar el cursor	"nuevo_valor" WHERE nombre_columna = "valor"
self.atributo1 = atributo1	{n,} Al menos n veces	y reer to come the emoderning about of guaratum com in case()	<pre>variable_query = ("SQL Query") variable</pre>	Insertar múltiples filas a una tabla
<pre>self.atributo2 = atributo2 self.atributo_por_defecto = 'valor'</pre>	{n,m} Entre n y m veces	mode: argumento opcional al abrir un archivo	<pre>variable cursor.execute(variable query) variable cursor.execute(variable query)</pre>	<pre>variable_valores_en_tuplas = ((valor1columna1,</pre>
<u></u>		r - read w - write - sobreescribe	devuelve una lista de tuplas	<pre>valor1columna2), (valor2columna1, valor2columna2))</pre>
<pre>def nombre_funcion1(self, parametros)</pre>	Métodos Regex	x - exclusive creation, sólo crearlo si no existe todavía	import datetime sacar fechas en el formato AAAA-MM-DD	<pre>variable_cursor.executemany(variable_query, variable_valores_en_tuplas)</pre>
<pre>self.atributo += 1 noturn f"cl nuovo valon os (self atribute)"</pre>	re.findall("patron", string) busca en todo el	a - appending, añadir texto al archivo sin manipular el texto que ya había	datetime.date(AAAA, M, D) devuelve el formato de fecha	vai Taute_vatoi es_eii_tuptas)
return f"el nuevo valor es {self.atributo}"	string y devuelve una lista con todas las	hay que anadir otra letra:	<pre>variable_query = "SQL Query %s AND %s") query dinamica</pre>	<pre>variable conexion.commit() después de ejecutar la</pre>
Definir una clase hija:	coincidencias en nuestro string	t - texto - leer en texto	<pre>variable_cursor.execute(query, (variable1, variable2)) valores que van en lugar de los %s</pre>	inserción, para que los cambios efectúen en la BBDD
<pre>class NombreClaseHija(NombreClaseMadre):</pre>	<pre>re.search("patron", string_original) todo el string y devuelve un objeto con la</pre>	<ul><li>b - bytes - leer en bytes (no se puede usar con encoding)</li></ul>	variable cursor.execute("SHOW DATABASES") mostrar las BBDD	<pre>variable_conexion.rollback()</pre> se puede usar después de
<pre>definit(self, atributo1, atributo2):</pre>	primera coincidencia en nuestro string	<pre>f = open(ubicacion_archivo, mode = "rt")</pre>	variable cursor.execute("SHOW TABLES") mostrar las tablas	execute y antes de commit para deshacer los cambios
<pre>super()init(atributo_heredado1,)</pre>	re.match("patron", "string original) busca en	Leer ficheros	de la BBDD indicado en la conexión	<pre>print(variable_cursor.rowcount, "mensaje") imprimir el número de filas en las cuales se han tomado la accion</pre>
<pre>def nombre_funcion_hija (self, parametros):</pre>	la primera linea del string y devuelve un	f.read() leer el contenido de un archivo	<pre>variable_cursor.execute("SHOW TABLES")</pre>	
	objeto con la primera coincidencia en nuestro string	f.read(n) leer los primeros n caracteres de un archivo	variable_cursor.execute("SHOW COLUMNS FROM bbdd.table")	Eliminar registros
Crear un objeto de la clase:		<pre>variable = f.read() guardar el contenido del archivo (o n caracteres de un archivo) en un variable</pre>	mostrar las columnas de la tabla especificada; hay que conectarse a la bbdd information schema	<pre>variable_query = "DROP TABLE nombre_tabla"</pre>
<pre>variable_objeto = NombreClase(valor_atributo1, valor_atributo2) instanciar (crear) un objeto</pre>	<pre>resultado_match.span() de las posiciones donde hizo el "match"</pre>	<pre>f.readline(n) por defecto devuelve la primera linea o n lineas</pre>	Argumentos cursor:	Añadir errores
variable_objeto.atributo devuelve el valor del	<u> </u>	<pre>f.readlines() devuelve una lista de todas las lineas del archivo (cada linea es un elemento); se usa vacio sin n y</pre>	<pre>variable_cursor = cnx.cursor([arg=value[, arg=value]])</pre>	importar errorcode y usar try/except:
atributo guardado para ese objeto	<pre>resultado_match.group() devuelve el element resultando de la coincidencia del "match"</pre>	list_name[x:] para seleccionar lineas especificas	buffered=True devuelve todas las filas de la bbdd	try:
<pre>variable_objeto.atributo = nuevo_valor el valor del atributo</pre>	re.split("patron", "string original") busca en	Facultiu au fichausa	raw=True el cursor no realizará las conversiones	accion except mysgl.connector.Error as err:
<pre>variable_objeto.nombre_funcion() </pre>	todo el string y devuelve una lista con los	Escribir en ficheros with open(ubicacion archivo, "w") as f:	automáticas entre tipos de datos	print(err)
	elementos separados por el patron	f.write("Texto que va en el fichero.") para escribir	dictionary=True devuelve las filas como diccionarios	<pre>print("Error Code:", err.errno) print("SQLSTATE", err.sqlstate)</pre>
<pre>print(help(NombreClase) imprime informacion sobre la</pre>		with open(ubicacion archivo, "a") as f:	<pre>named tuple=True devuelve las filas como named tuples</pre>	print( Sylstate, err.sqistate)
clase	<pre>re.sub("patron", "string_nuevo",</pre>		named_tuple=11 de devdeive las l'ilas como named tuples	<pre>print("Message", err.msg)</pre>
clase	<pre>re.sub("patron", "string_nuevo",     "string_original") busca en todo el string y devuelve un string con el element que coincide</pre>	f.write("Texto que va en el fichero.") para anadir texto f.writelines('lista') para anadir lineas de texto de una lista	cursor_class un argumento que se puede usar para indicar que subclase queremos usar para instanciar el nuevo cursor	<pre>print("Message", err.msg)</pre>

Python Cheat Sheet 3	DataFrames	Metodos de exploracion	Tipos de datos	Valores nulos
Pandas	Crear DataFrames  df = pd.DataFrame(data, index, columns)  data: NumPy Array, diccionario, lista de diccionarios	<pre>df.head(n) devuelve las primeras n lineas del dataframe df.tail(n) devuelve las últimas n lineas del dataframe</pre>	Tipos de datos en Pandas: - object - int64	Identificar nulos   df.isnull() o df.isnull() o df.isnull() o df.isna() devuelve True o False según si cada
Series: estructuras en una dimension	<pre>index: indice que por defecto se asigna como 0-(n-1), n siendo el número de filas;</pre>	<pre>df.sample(n) devuelve n filas aleatorias de nuestro dataframe, o uno por defecto</pre>	- float64 - datetime, timedelta[ns]	valor es nulo o no  df.isnull().sum() o df.isna().sum() devuelve una serie con el número de valores nulos por columnas
<pre>Crear series serie = pd.Series() crear serie vacía serie = pd.Series(array) crear serie a partir de un array con el indice por defecto</pre>	<pre>index = [lista] para asignar "etiquetas" (nombres de filas) column: nombre de las columnas; por defecto 0-(n-1); columns =[lista] para poner mas nombres</pre>	df.shape devuelve el número de filas y columnas  df.dtypes devuelve el tipo de datos que hay en cada columna  df.columns devuelve los nombres de las columnas	<pre>- category - bool  df.dtypes devuelve el tipo de datos que hay en cada columna df_tipo = df.select_dtypes(include = "tipo")</pre>	<pre>df_%_nulos = ((df.isnull().sum() / df.shape[0] * 100).reset_index() df_%_nulos.columns = ['columna', '% nulos'] crea un dataframe de los porcentajes de los valores nulos</pre>
<pre>serie = pd.Series(array, index = ['a', 'b', 'c']) crear una serie con indice definida; debe ser lista de la misma longitude del array serie = pd.Series(lista) crear una seria a partir de una lista serie = pd.Series(número, indice) crear una serie a</pre>	<pre>df = pd.DataFrame(array) crear un dataframe a partir de un array con indices y columnas por defecto df = pd.DataFrame(diccionario) crear un dataframe a partir de un diccionario - los keys son los nombres de las columnas</pre>	<pre>df.describe devuelve un dataframe con un resumen de los principales estadísticos de las columnas numéricas  df.info() devuelve un resumen sobre el no. de columnas, nombres de columnas, numero de valores no nulos y los tipos de datos df["nombre_columna"].unique() o</pre>	<pre>crea un dataframe de las columnas del tipo de datos especificado df['columna'] = df['columna'].astype('tipo', copy = True, errors = 'ignore') convierte una columna en el tipo de dato especificado copy = True devuelve una copia</pre>	Eliminar nulos  df.dropna(inplace = True, axis=b, subset=[lista_de_columnas], how=) quitar nulos how = 'any'   'all' por defecto 'any': si hay algun valor NA, se elimina la fila o columna; all: si todos los valores son NA,
<pre>partir de un escalar con la longitude igual al número de indices serie = pd.Series(diccionario) crear una serie a</pre>	DataFrames: carga de datos	df.nombre_columna.unique() devuelve un array con los valores únicos de la columna	<pre>copy = False *cuidado: los cambios en los valores pueden propagarse a otros objetos pandas*</pre>	se elimina la fila o columna subset una columna o lista de columnas
Acceder a informacion de una serie serie.index devuelve los indices serie.values devuelve los valores serie.shape devuelve la forma (no. filas) serie.size devuelve el tamaño serie.dypes devuelve el tipo de dato	Carga de datos  df = pd.read_csv("ruta/nombre_archivo.csv") crear un dataframe de un archivo de Comma Separated Values  df = pd.read_csv("ruta/nombre_archivo", sep= ";") crear un dataframe de un csv si el separador es ; df = pd.read_csv("ruta/nombre_archivo", index_col= 0) crear un dataframe de un csv si el archivo ya tiene una columna indice	<pre>df["nombre_columna"].value_counts() o df.nombre_columna.value_counts() devuelve una serie con el recuento de valores únicos en orden descendente  df.duplicated().sum() devuelve el numero de filas duplicadas  Eliminar filas duplicadas  df.drop_duplicates(inplace = True) elimina filas</pre>	<pre>errors = ignore omita excepciones; en caso de error devuelve el objeto original errors = raise permite que se generen excepciones  pd.options.display.max_columns = None</pre>	Tipos de nulos  np.nan significa "not a number"; es un tipo numérico  None valores nulos en columnas tipo string  NaT valores nulos tipo datetime  valores texto: "n/a", "NaN", "nan", "null" strings que normalmente se convierten automaticamente a np.nan 99999 o 00000 integers que se pueden convertir a nulos
<pre>serie[i] devuelve el valor del elemento en indice i serie[[i,j]] devuelve el valor de los dos elementos serie[i:m] devuelve el valor de un rango</pre>	<pre>df = pd.read_excel("ruta/nombre_archivo.xlsx") crear un dataframe de un archivo de Excel</pre>	duplicadas  Metodos de estadistica	<pre>antes del df.head() para poder ver todas las columnas  pd.set_option("display.precision", 2)</pre>	<pre>Reemplazar nulos  df = pd.read_csv('archivo.csv', na_values = ['n/a'])    .fillna(np.nan) reemplaza los strings 'n/a' con np.nan al    cargar el dataframe</pre>
serie["etiqueta"] devuelve el valor de los elementos	pip3 install openpyxl o pip install openpyxl	df['columna'].mean()   mode()   median()   var()   std()	Outliers	<pre>df.fillna(df[value=n, axis=b, inplace=True) reemplazar todos los NaN del dataframe con el valor que especifiquemos</pre>
<pre>en indices i y j  Operaciones con series serie1 +-*/ serie2 suma/resta/multiplica/divide las filas con indices comunes entre las dos series serie1.add(serie2, fill_value = número) suma las filas con indices comunes, y suma el fill value a los valores sin indice comun serie1.sub(serie2, fill_value = número) restan las</pre>	<pre>df = pd.read_json("ruta/nombre_archivo.json") crear un dataframe de un archivo de JavaScript Object Notation (formato crudo) df = df['data'].apply(pd.Series) convertir el dataframe de json en un formato legible  df = pd.read_clipboard(sep='\t') crear un dataframe de datos en forma de dataframe en el clipboard; el separador podria ser \n ; , etc.</pre>	calcula la media/moda/mediana/variación/desviación estándar de los valores de una columna  df['columna1'].corr(df['columna2'] calcula la correlacion entre dos variables  matriz_correlacion = df.corr() crea una matriz mostrando las correlaciones entre todos los variables  df_crosstab = pd.crosstab(df['columna1'],  df['columna2'], normalize = True, margins = True)	Calcular tres desviaciones estandares: media = df.column.mean() desviacion = df.column.std()  lcb = media - desviacion * 3 ucb = media + desviacion * 3  Eliminar Outliers	<pre>df['columna'].fillna(df['columna'].median, axis=b,   inplace=True) reemplazar los nulos de una columna por la   mediana de esa columna   value=n por defecto NaN; es el valor por el que queremos   reemplazar los valores nulos que puede ser un escalar,   diccionario, serie o dataframe   axis por defecto 0 (filas)   df.replace(valor_nulo, valor_nuevo, inplace=True, regex=False)   reemplazar los nulos por el valor nuevo</pre>
filas de la seria2 de la serie1 cuando tienen indices comunes, y resta el fill value de las otras indices de serie1  serie1.mul(serie2, fill_value = número) multiplica las filas con indices comunes y multiplica el fill value con las otras *usar 1 para conservar el valor*  serie1.mul(serie2, fill_value = número) divida las filas de la serie1 entre las de la serie2 cuando tienen indices comunes, y divide las otras por el fill value	<pre>with open('ruta/nombre_archivo.pkl', 'wb') as f:     pickle.dump(df,f)</pre>	<pre>normalize muestra los valores en porcentajes (por uno) margins muestra los totales y subtotales media_ponderada = np.average(df['columna'], weights = w) calcula la media ponderada según los pesos percentil_n = np.percentile(df['columna'], n) saca el valor en el percentil n q3, q1 = np.percentile(df["columna"], [75, 25]) saca los tercer y primer cuartiles</pre>	<pre>outlier_step = 1.5 * IQR calcular outlier step outliers_data = df[(df['columna'] &lt; Q1 - outlier_step)   (df['columna'] &gt; Q3 + outlier_step)] identificar datos fuera del rango del maximo hasta el minimo lista_outliers_index = list(outliers_data. index) crear una lista de los indices de las filas con outliers  if outliers_data.shape[0] &gt; 0:</pre>	<pre>Imputacion de nulos from sklearn.impute import SimpleImputer imputer = SimpleImputer(strategy='mean', missing_values = np.nan) inicia la instancia del metodo, especificando que queremos reemplazar los nulos por la media imputer = imputer.fit(df['columna1']) aplicamos el imputer df['media_columna1'] = imputer.transform(df[['price']]) rellena los valores nulos segun como hemos especificado</pre>
<pre>serie1.mod(serie2, fill_value = número) devuelve el modulo (division sin resta) serie1.pow(serie2, fill_value = número) calcula el exponencial serie1.ge(serie2) compara si serie1 es mayor que serie2 y devuelve True o False serie1.le(serie2) compara si serie1 es menor que</pre>	Guardado de datos  df.to_csv('ruta/nombre_archivo.csv') guardar dataframe como archivo csv  df.to_excel('ruta/nombre_archivo.xlsx') guardar dataframe como archivo de Excel  df.to_json('ruta/nombre_archivo.json') guardar dataframe	Sidetable: frecuencias de datos  df.stb.freq(['columna']) devuelve un dataframe con informacion sobre la frecuencia de ocurrencia de cada categoría de un variable categorica	<pre>dicc_indices[key] =   (list(outliers_data.index)) crear un   diccionario de los indices de las filas con   nulos; se puede hacer iterando por columnas   valores = dicc_indices.values() sacar todos   los valores e.g. todos los indices</pre>	from sklearn.experimental import enable_iterative_imputer from sklearn.impute import IterativeImputer imputer = IterativeImputer(n_nearest_features=n, imputation_order='ascending') crea la instancia n_nearest_features por defecto None; numero de columnas a utilizar para estimar los valores nulos
Filtrado booleanos serie < > >= <= = valor si cada condición cumple la condición serie1[serie1 < > >= <= = valor] devuelve solo los valores que cumplen la condición	<pre>como archivo de JSON df.to_parquet('ruta/nombre_archivo.parquet') guardar dataframe como archivo de parquet df.to_pickle('ruta/nombre_archivo.pkl') guardar dataframe como archivo de pickle ExcelWriter</pre>	parametros:  thresh = n limita los valores mostrados a los más frecuentes hasta un umbral de n% cumulative y agrupando los restantes bajo la etiqueta "other" other_label = 'etiqueta' cambia la etiqueta 'other' value = 'columna' ordena los resultados por la columna especificada df.stb.freq(['columna1', 'columna2']) combina dos	<pre>valores = {indice for sublista in valores for indice in sublista} set comprehension para eliminar duplicados  df_sin_outliers = df.drop(df.index[list (valores)]) crear nuevo dataframe sin outliers</pre> Reemplazar Outliers	<pre>imputation_order por defecto ascendente; el orden de imputacion imputer.fit(df_numericas) aplicamos el imputer  df_datos_trans = pd.DataFrame(imputer.transform(df_numericas), columns = df_numericas.columns) crea un dataframe de los datos transformados; metemos estas columnas en el dataframe original from sklearn.impute import KNNImputer</pre>
<pre>np.nan crear valor nulo (NaN) serie.isnull() devuelve True o False segun si los valores existen o son nulos ("" no cuenta como nulo) serie.notnull() devuelve True o False segun si los valores existen o son nulos ("" no cuenta como nulo)</pre>	<pre>with pd.ExcelWriter("ruta/archivo.ext") as writer:     df.to_Excel(writer, nombre_hoja = 'nombre') guardar un dataframe en una hoja de Excel</pre>	columnas y devuele las frecuencias de las subcategories  df.stb.missing(['columna'] devuelve informacion sobre la frequencia de datos nulos	<pre>for k, v in dicc_indices.items():     media = df[k].mean()     for i in v:         df.loc[i,k] = media reemplazar outliers por la media</pre>	<pre>imputerKNN = KNNImputer(n_neighbors=5) crea la instancia imputerKNN.fit(df_numericas)  df_knn_imp = pd.DataFrame(imputerKNN.transform(df_numericas), columns = numericas.columns) crea un dataframe de los datos transformados; metemos estas columnas en el dataframe original</pre>

Python Cheat Sheet 4	Subsets: loc e iloc	Filtrados de datos	Crear columnas	Cambiar valores
Pandas	<pre>df.loc["etiqueta_fila", "etiqueta_columna"] devuelve el contenido de un campo en una columna</pre>	Metodos de pandas de filtrar df filtrado =	<pre>Creacion de ratios df["columna?ratio"] = df.apply(lambda df: df["columna1"]</pre>	Reemplazar valores basados en indices y condiciones:
Union de datos	de una fila	<pre>df[df["nombre_columna"].isin(iterable)] extrae las filas cuyas valores de la columna nombrada están en</pre>	/ df["columna2"], axis = 1)	<pre>indices_filtrados = df.index[df["columna"] == "valor"] for indice in indices_filtrados:</pre>
.concat() unir dataframes con columnas en comun	<pre>df.loc["etiqueta_fila",:] devuelve los valores de todas las columnas de una fila</pre>	el iterable (una lista, serie, dataframe o diccionario)	<pre>Creacion de porcentajes def porcentaje(columna1, columna2):</pre>	df["nombre_columna"].iloc[indice] = "valor_nuevo"  Reemplazar valores basados en metodos NumPy:
<pre>df_union = pd.concat([df1, df2, df3], axis=b, join =     'inner/outer', ignore_index = True/False) parametros: axis = 0 une por columnas - los dataframes van uno encima del</pre>	<pre>df.loc[:,"etiqueta_columna"] devuelve los valores de todas las filas de una columna df.iloc[indice_fila, indice_columna] devuelve el</pre>	<pre>df_filtrado= df[df["nombre_columna"].str.contains (patron, regex = True, na = False)] extrae las filas cuyas valores de la columna nombrada contenienen el</pre>	<pre>return (columna1 * 100) / columna2  df["columna_%"] = df.apply(lambda df: porcentaje(df["columna1"], datos["columna2"]), axis = 1)</pre>	<pre>df.replace(to_replace = valor, value = valor_nuevo,   inplace = True) reemplaza cierto valor por otro que   especificamos</pre>
otro; las columnas tienen que ser de formatos compatible  axis = 1 une por filas - los dataframes van uno al lado del otro;  los datos deben ser relacionados para que tenga sentido  join = 'inner' solo se quedan elementos que aparecen en todos los	<pre>contenido de un campo en una columna de una fila df.iloc[indice_fila, :] devuelve los valores de todas las columnas de una fila</pre>	patron de regex  df_filtrado = df[df["nombre_columna"].str.contains	<pre>df["nueva_columna"] = np.where(df["nombre_columna"] &gt; n,     "categoria_if_true", "categoria_if_false") crea una nueva columna basada en una condición</pre>	<pre>df["nombre_columna"].replace(to_replace = valor, value = valor_nuevo, inplace = True) reemplaza cierto valor en una columna por otro que especificamos</pre>
dataframes  join = 'outer' se queda todo los datos de todos los dataframes  ignore_index = True/False por defecto es False; si es True no usa	<pre>df.iloc[:,indice_columna] devuelve el contenido de un campo en una columna de una fila</pre>	<pre>("substring", case = False, regex = False)] extrae las filas cuyas valores de la columna nombrada contienen el substring, no siendo case sensitive</pre>	<pre>df["nueva_columna"] = np.select(lista_de_condiciones, lista_de_opciones) crea una nueva columna con los valores basados en multiples condiciones</pre>	<pre>df[["columna1", "columna2"]] = df[["columna1",     "columna2"]].replace(r"string", "string", regex=True)     cambiar un patron/string por otro en multiples     columnas</pre>
las indices para la union (por ejemplo para union por el axis 0)  .merge() unir las columnas de un dataframe a otro  df_nuevo = df1.merge(df2, on = 'columna') inner merge  df nuevo = pd.merge(left = df1, right = df2, how='left', left on =	<pre>df.loc[[lista_etiquetas_filas], [lista_etiquetas_columnas]] devuelve el contenido de varias filas / varias columnas</pre>	<pre>df_filtrado = df[df["nombre_columna"].str.contains   ("substring", case = False, regex = False)] extrae las filas cuyas valores de la columna nombrada   contienen el substring, no siendo case sensitive</pre>	<pre>df["columna_nueva"] = pd.cut(x = df["nombre_columna"], bins = [n,m,1], labels = ['a', 'b', 'c']) separa los elementos de un dataframe en diferentes intervalos (n-m, m-l, etc), creando una columna nueva que indica en cual</pre>	<pre>df["nombre_columna"] = df["nombre_columna"] + x reemplaza los valores de la columna por el valor + x (o otro valor que indicamos)</pre>
'columna_df1', right_on = 'columna_df2') left merge parametros:	<pre>df.loc[[lista_indices_filas],   [lista_indices_columnas]] devuelve el contenido   de varias filas / varias columnas</pre>	<pre>df[pd.notnull(df["nombre_columna"])] devuelve las</pre>	intervalo cae el valor; con labels se puede asignar un string a cada intervalo	datetime
<pre>how = 'left'   'right'   'outer'   'inner'   'cross' on = columna   [columna1, columna2, etc] si las columnas se llaman igual en los dos dataframes</pre>	- se puede usar los indices/rangos de las listas [start:stop:step] dentro de los loc/iloc	filas que no tiene valores nulos en la columna especificada	<pre>Crear columnas df["nueva_columna"] = (df["etiqueta_columna"] + x) crea una nueva columna basada en otra</pre>	<pre>import datetime datetime.now() devuelve la fecha actual</pre>
<pre>left_on = columna_df1   right_on = columna_df2 para especificar por donde hacer el merge suffixes = ['left', 'right'] por defecto nada, el sufijo que</pre>	<pre>df.loc[df.etiqueta &gt; x] seleccionar datos basado en una condición usando operadores comparativos</pre>	Cambiar columnas	<pre>df = df.assign(nueva_columna= df["etiqueta_columna] + x) crea una nueva basada en otra</pre>	<pre>timedelta(n) representa una duración la diferencia entre dos instancias; n es un numero de días datetime.strftime(variable_fecha, '%Y-%m-%d') formatea</pre>
aparecera en columnas duplicadas .join() unir dataframes por los indices	<pre>df.loc[(df.etiqueta &gt; x) &amp; (df.etiqueta == y)] seleccionar datos que tienen que cumplir las dos</pre>	<pre>lista_columnas = df.columns.to_list() crea una lista de los nombres de las columnas del dataframe</pre>	<pre>df = df.assign(nueva_columna= [lista_valores]) crea una nueva columna de una lista de valores *tiene que ser de la misma longitud como el número de filas del dataframe*</pre>	la fecha al formato indicado
<pre>df_nuevo = df1.join(df2, on = 'columna', how = 'left') inner merge parametros: how = 'left'   'right'   'outer'   'inner' por defecto left on = columna la columna o indice por el que queremos hacer el</pre>	<pre>condiciónes (and)  df.loc[(df.etiqueta &gt; x)   (df.etiqueta == y)] seleccionar datos que tienen que deben cumplir una de las dos condiciones (or)</pre>	<pre>df.set_index(["nombre_columna"], inplace = True) establece el índice utilizando uno o mas columnas; puede sustituir o ampliar un índice existente</pre>	<pre>df.insert(indice_nueva_columna, "nombre_columna", valores) crea una nueva columna en la indice indicada allow_duplicates = True parametro cuando queremos permitir columnas duplicadas (por defecto es False)</pre>	<pre>ayer = datetime.now() - timedelta(1) ayer = datetime.strftime(ayer, '%Y-%m-%d') df["fecha"] = ayer crea una columna con la fecha de ayer</pre>
<pre>union; tienen que tener el mismo nombre en los dos dataframes lsuffix = 'string'   rsuffix = 'string' por defecto nada, el sufijo que aparecera en columnas duplicadas</pre>	<pre>df.iloc[list(df.etiqueta &gt; x), :] iloc no acepta una Serie booleana; hay que convertirla en lista</pre>	<pre>inplace = True los cambios sobreescriben sobre el df * cuando una columna se cambia a índice ya no es columna *</pre>	Apply	APIs
Group By	<pre>variable_df.head(n) devuelve las n primeras filas del df, o 5 por defecto</pre>	<pre>df.reset_index(inplace = True) quitar una columna como indice para que vuelva a ser columna; crea un dataframe de una serie</pre>	<pre>apply() toma una función como argumento y la aplica a lo largo de un eje del DataFrame df['columna_nueva'] = df['col_1'].apply(función)</pre>	<pre>import requests libreria para realizar peticions HTTP a una URL, para hacer web scraping url = 'enlace' el enlace de la que queremos extraer</pre>
<pre>df_groupby = df.groupby("columna_categoría") crea un objeto DataFrameGroupBy; agrupa los valores segun las categorías de los</pre>		Renombrar columnas	crea una columna nueva con los valores de otra columna transformados según la función indicada	datos <pre>header = {} opcional; contiene informacion sobre las</pre>
<pre>valores de la columna indicada (o múltiples columnas en una lista) df_groupby.ngroups devuelve el numero de grupos df_groupby.groups devuelve un diccionario donde los keys son las</pre>	Filtrados de datos	<pre>df.rename(columns = {"nombre_columna":     "nombre_nueva"}, inplace = True) cambia los nombres</pre>	<pre>df['columna_nueva'] = df['col_1'].apply(lambda x: x.método() if x &gt; 1)</pre>	peticiones realizadas (tipo de ficheros, credenciales) response = requests.get(url=url, header = header)
categorías y los valores son listas de los índices de cada elemento en la categoría	Filtrado por una columna con operadores de comparación	de una o mas columnas ejemplo de dict comprehension para crear diccionario	crea una columna nueva con los valores de otra columna transformados según la lambda indicada	<pre>pedimos a la API que nos de los datos variables = {'parametro1':'valor1',     'parametro2':'valor2'}</pre>
<pre>df_grupo1 = df_groupby.get_group("grupo1") devuelve un dataframe con los resultados de un grupo (la categoria indicada como grupo1) Cálculos con groupby:</pre>	<pre>df_filtrado = df[df["nombre_columna"] == valor] extrae las filas donde el valor de la columna igual al valor dado</pre>	<pre>sobre las columnas existentes de un dataframe: diccionario = {col : col.upper() for col in df.columns}</pre>	<pre>df['columna_nueva'] = df.apply(lambda nombre: función(nombre['columna1'], nombre['columna2']), axis = b) crea una columna nueva usando una función que coge dos</pre>	response = request.get(url=url, params=variables) pedimos a la API que nos de los datos con los parametros segun el diccionario de parametros que le
<pre>df_nuevo = df.groupby("columna_categoría").mean() devuelve un dataframe con la media de todas las columnas de valores numéricos, por categoría df.groupby("columna categoría")["columna"] mean()</pre>	Filtrado por multiples columnas con operadores logicos	<pre>df.rename(columns = diccionario, inplace = True) cambia los nombres de las columnas según el diccionario</pre>	parámetros (columna 1 y columna2) pd.applymap: se puede aplicar solo a todo el DataFrame. df.applymap(func, na_action=None, **kwargs)	response.reason devuelve el status de la peticion response.reason devuelve el motive de codigo de estado
<pre>df_nuevo = df.groupby("columna_categoría")["columna1"].mean() devuelve un dataframe con la media de la columna especificada count() número de observaciones</pre>	<pre>df_filtrado = df[(df["columna1"] == valor) &amp;   (df["columna2"] == valor) &amp; (df["columna3"] &gt; n   valor)] extrae las filas donde los valores de</pre>	Eliminar columnas  df.drop(columns = ["columna1", "columna2"], axis =	map solo lo podremos aplicar a una columa en particular. MiColumna.map(arg, na_action=None)	<pre>response.text devuelve los datos en formato string response.json() devuelve los datos en formato json df = pd.json_normalize(response.json) devuelve los</pre>
no nulas describe() resumen de los	las columnas cumplan las condiciónes en parentesis	<pre>b, inplace=True) eliminar una o mas columnas o filas segun lo que especificamos</pre>	apply() con datetime	datos en un dataframe
principales estadísticos  sum() suma de todos los valores  mean() media de los valores	<pre>df_filtrado = df[(df["columna1"] == valor)     (df["columna1"] == valor) extrae las filas donde los valores de las columnas cumplan con una</pre>	Reordenar columnas  df = df.reindex(columns = lista_reordenada) cambia el orden de las columnas del dataframe segun el orden de	<pre>df['columna_fecha'] = df['columna_fecha'] .apply(pd.to_datetime) cambia una columna de datos tipo fecha en el formato datetime</pre>	Codigos de respuesta de HTTP  1XX informa de una
<pre>df_nuevo = df.groupby("columna_categoría", dropna = False) ["columna_valores"].agg([nombre_columna = 'estadistico1', nombre_columna2 = 'estadistico2']) añade columnas con los cálculos</pre>	condición u otra  df_filtrado = ~(df[df["columna1"] == valor])	la lista reordenada	<pre>def sacar_año(x):     return x.strftime("%Y") df['columna año'] = (df['columna fecha'] .apply</pre>	200 OK 403 prohibido 201 creado 404 no encontrado 202 aceptado 5XX error del servidor
de los estadísticos especificados dropna = False para tener en cuenta los Nan en los cálculos (por defecto es True)	extrae las filas donde los valores de las columnas NO cumplan con la condición		(sacar_año) crea una columna nueva del año solo usando un método de la librería datetime; ("%B") para meses	204 sin contenido 501 error del servidor 3XX redireccion 503 servicio no disponible

**Python Cheat Sheet 5** Personalización Seaborn Personalización Seaborn Usos de los tipos de gráficas color = "color" establece el color de la grafica Line plot fig.set(xlabel = 'etiqueta eje x', ylabel = Matplotlib 'etiqueta\_eje\_y') asignar nombre a los ejes facecolor = "color" establece el color del relleno fig = sns.lineplot(x = 'columna1', y = 'columna2', data = df, ci = None) crea una gráfica lineal donde los ejes son: fig.set title('titulo') asignar un titulo a la gráfica edgecolor = "color" establece el color de los bordes columna1 - x, columna2 - y **Gráficas** Colores en Scatter Plots: ci = None para que no muestra el intervalo de confianza de sns.set(font scale=2) c= df['columna'].map(diccionario) los datos import matplotlib.pyplot as plt order = df.sort values('columnay', diccionario = {"valor1": "color1", "valor1": "color1"} hue = columna opcional; muestra lineas en diferentes ascending=False)['columnax'] colores por categorias segun una variable lista de colores plt.rcParams["figure.figsize"] = (10,8) plt.xlabel("etiqueta\_eje\_x") asignar nombre al eje x plt.figure(figsize = (n,m)) inicia una grafica fig = sns.scatterplot(x = 'columna1', y = 'columna2', data order = df emp.sort values('categoría', ascending = dibujando el marco de la figura; n es la anchura y plt.ylabel("etiqueta eje y") asignar nombre al eje y = df. hue = 'columna') crea una gráfica de dispersión False)['id empleado'] m es la altura, en pulgadas plt.legend(labels = ['label1', 'label2', etc) muestra plt.show() muestra la figura la leyenda cuando mostramos la figura fig = sns.swarmplot(x = 'columna1', y = 'columna2', data = plt.rcParams.update({'font.size': 22}) font size plt.legend(bbox to anchor = (1, 1) coloca la leyenda en df, hue = 'columna') crea una gráfica de dispersión donde Gráficas básicas relación con los eies los marcadores no se solapan plt.title(label = "titulo") asignar un titulo a la Bar plot Count plot Calcular tres desviaciones estandares: plt.bar(df["columna1"], df["columna2"]) crea un fig = sns.countplot(x = 'columna1', data = df, hue = media = df.column.mean() plt.xlim([n,m] establece el rango del eje x; donde n es diagrama de barras donde los ejes son: columna1 -'columna') crea una gráfica de barras con la cuenta de una desviacion = df.column.std() el mínimo y m es el máximo x. columna2 - v variable categórica; se puede especificar solo una plt.ylim([n,m]) establece el rango del eje y; donde n Horizontal bar plot variable en la eje x o y, mas una variable opcional con lcb = media - desviacion \* 3 es el mínimo y m es el máximo plt.barh(df["columna1"], df["columna2"]) crea una ucb = media + desviacion \* 3 diagramma de barras horizontales donde los ejes plt.grid() crea una cuadrícula al fondo de la figura; son: columna1 - x, columna2 - y coge los parámetros: fig = sns.histplot(x = 'columna1', data = df, hue = fig.axvline(x=lcb, c='color', label='99 inferior') color = "color' Stacked bar plot 'columna3', kde = True, bins = n) crea una histograma que fig.axvline(x=ucb, c='color', label='99 superior') plt.bar(x, y, label = 'etiqueta') linestyle = "solid" | "dashed" | "dashdot" | "dotted" muestra la frecuencias de una distribución de datos; donde plt.bar(x2, y2, bottom = y, label = 'etiqueta2') x es la variable de interés y n es el número de barras linewidth = n establece la anchura de la linea crea una diagrama de barras apiladas para kde = True muestra una curva de la distribucion marker = 'tipo' establece el tipo de marcador: se usa visualizar dos variables juntas; y indica la barra con plt.scatter y plt.plot de referencia fig = sns.boxplot(x = 'columna1', data = df, hue = "." Punto "P" Más (relleno) 'columna') crea un diagrama de cajas; x es la variable de "\*" Estrella "," Pixel plt.scatter(df["columna1"], df["columna2"]) crea interés; por defecto se muestra con orientación horizontal "o" Circulo "h" Hexágono 1 una gráfica de dispersión donde los ejes son: usar eie v para orientación vertical "v" Triángulo abajo "H" Hexágono 2 columna1 - x, columna2 - y "^" Triángulo arriba "+" Más fig = sns.catplot(x = 'columna1', y = 'columna2', data = "<" Triángulo izquierda "x" x Gráficas estadísticas df, hue = 'columna', kind = 'tipo') crea una gráfica que ">" Triángulo derecha "X" x (relleno) muestra la relacion entre una variable categorica y una Histogram "8" Octágono "D" Diamante variable numerica plt.hist(x = df['columna1'], bins = n) crea una "s" Cuadrado "d" Diamante fino kind = 'box' | 'bar' | 'violín' | 'boxen' | 'point' por histograma que muestra la frecuencias de una "p" Pentágono defecto es strip plot distribución de datos: donde x es la variable de interés y n es el número de barras Pairplot Exportar figuras fig = sns.pairplot(data = df, hue = 'columna', kind = plt.savefig('nombre de la figura.extension') (tipo') crea los histogramas y diagramas de dispersión de plt.boxplot(x = df['columna1']) crea un diagrama de cajas para estudiar las caracteristicas de una todas las variables numéricas de las que disponga el variable numerica: x es la variable de interés dataset con el que estemos trabajando; hue es opcional Multigráficas - el mínimo es lo mismo que Q1 - 1.5 \* IQR kind = 'scatter' | 'kde' | 'hist' | 'reg' | 'point' por el máximo es lo mismo que Q3 + 1.5 \* IQR defecto es scatter fig, ax = plt.subplots(numero filas, numero columnas) crear una figura con multiples graficas; fig es la ← Outliers sns.heatmap(df.corr(), cmap = 'color palette', annot = figura y ax es un array con subplots como elementos True, vmin = -1, vmax = 1) crea un heatmap con una escala \_<del>ઁ</del> ← "Máximo" se establece como es cada grafica con los indices: de colores que refleja los valores de correlacion ax[indice].tipo grafica(detalles de la grafica) annot = True para que aparezcan los valores ax[indice].set title('titulo') vmin/vmax establecen la escala de color ← Mediana ax[indice].set xlabel('xlabel') Regplot ax[indice].set ylabel('ylabel') fig = sns.regplot(x = 'columna1', y = 'columna2', data = ax[indice].set xlim(min, max ⊥ ← "Mínimo" df, scatter kws = {'color':'blue'}, line kws = {'color'; ax[indice].set ylim(min, max) 'blue'})crea un scatterplot mas la línea de regresión; nos ← Outliers ax[indice].set xticklabels(labels = df['column'], permite encontrar la mejor función de la recta que permite rotation = n) para cambiar los nombres y/o la rotacion predecir el valor de una variable sabiendo los valores de Pie Chart otra variable de las etiquetas de los valores en los ejes plt.pie(x, labels = categorias, radius = n) crea un Jointplot gráfico de sectores donde x es la variable de sns.jointplot(x = 'columna1', y = 'columna2', data = df, interés (debe esta agrupado por categorias); n es color = 'blue', kind = 'tipo') crea un scatterplot o el tamaño regplot con histogramas pegados en los lados para cada variable Violin Plot plt.violinplot(x, showmedians = True, showmeans = True) crea un diagrama de violin donde x es la variable de interés y muestra la mediana y la media

NumPy (Numerical Python)	Indices, Subsets, Metodos de Arrays	Operaciones estadísticas y matemáticas	Funciones de conjuntos	Estadística
Crear arrays	Indices de arrays	Operaciones estadísticas y matemáticas	np.unique(array) devuelve un array con los valores únicos del array ordenados	Tablas de frecuencias
Crear arrays de listas  array = np.array(lista, dtype= tipo) unidimensional de una lista	<pre>array[i] devuelve la indice i; las indices de los arrays unidimensionales funcionan igual que las listas array[i, j] o array[i][j] devuelve el elemento de la columna j de la fila i</pre>	El parametro axis en arrays bidimensionales:  axis = 0 columnas  axis = 1 filas  - si especificamos el axis, la operación devuelve el resultado por cada fila o columna.	np.unique(array, return_index=True) devuelve un array con los valores únicos del array ordenados y un array con la posición de la primera instancia de cada valor np.unique(array, return_inverse=True) devuelve un array con los valores únicos del array ordenados y un array con las	Frecuencias absolutas el número de veces que se repite un número en un conjunto de datos  df = df.groupby('columna').count().reset_index()
<pre>array = np.array([lista1, lista2]) bidimensional de dos listas array = np.array([listadelistas1, listadelistas2]) un array bidimensional de dos listas</pre>	<pre>array[:,:n] seleccionar todas las filas y las columnas hasta n-1 array[h, i, j] o array[h][i][j] devuelve el elemento de la columna j de la fila i del array h array[h][i][i] = n cambiar el valor del elemento en</pre>	Por ejemplo:  np.sum(array, axis = 0) devuelve un array con la suma de cada fila	posiciones de cada elemento de cada valor  np.unique(array, return_counts=True) devuelve un array con los valores únicos del array ordenados y un array con el número de veces que aparece cada valor	Frecuencias relativas  las veces que se repite un número o categoría en un conjunto de datos respecto al total, en porcentajes  df_group_sin_str = df_group.drop('columna_str',
Crear otros tipos de arrays  array = np.arange(valor_inicio, valor_final, saltos) un array usando el formato [start:stop:step]	esta posicion al valor n  Subsets	El parametro axis en arrays multidimensionales:  axis = 0  axis = 1  columnas  axis = 2  filas	np.unique(array, axis = b) devuelve un array con los valores únicos ordenados de las filas o columnas  Funciones para arrays unidimensionales	<pre>axis=1) frecuencia_relativa = df_group_sin_str / df.shape[0] * 100 columnas = df group sin strings.columns</pre>
<pre>array = np.ones(z,y,x) crea un array de todo unos de la forma especificada array2 = np.ones_like(array1) crea un array de todo unos de la forma basada en otra array</pre>	<pre>array &gt; n devuelve la forma del array con True o False según si el elemento cumple con la condición o no array[array &gt; n] devuelve un subset: todos los valores</pre>	- si especificamos el axis, la operación devuelve el resultado por cada dimensión, fila o columna. Por ejemplo:	<pre>np.intersect1d(array1, array2) devuelve un array con los valores únicos de los elementos en común de dos arrays np.intersect1d(array1, array2, return indices=True)</pre>	<pre>df_group[columnas] = frecuencia_relativa  Tablas de contingencia</pre>
<pre>array = np.zeros(z,y,x) crea un array de todo zeros de la forma especificada array2 = np.zeros_like(array1) crea un array de todo zeros de la forma basada en otra array</pre>	que cumplen la condición en una lista dentro de un array array[(array > n) & (array < m)] devuelve un subset:	<pre>np.sum(array_3D, axis = 0) devuelve un array de una matriz con la suma de todas las matrices np.sum(array_3D, axis = 1) devuelve un array donde las filas contienen las sumas de las columnas de</pre>	devuelve un array con los valores únicos de los elementos en común de dos arrays y arrays con los índices de cada valor, por array	tabla de frecuencias que cuenta todas las combinaciones posibles de cada pareja de valores de las columnas que estamos intentando comparar  df_crosstab = pd.crosstab(df['columna1'],
<pre>array = np.empty((z,y,x), tipo) datos por defecto tipo float array2 = np.empty_like(array1)</pre> crea un array vacia con la	todos los valores que cumplen las condiciones en una lista dentro de un array; se puede usar   para "or"	Cada matriz  Operaciones con parámetro del axis:  np.sum(array 3D) devuelve la suma de todos los	np.union1d(array1, array2) devuelve un array ordenado con los elementos resultantes de unir dos arrays (valores únicos) np.in1d(array1, array2) devuelve un array con True o False	df['columna2'], normalize = True, margins = True)  normalize muestra los valores en porcentajes (por uno)  margins muestra los totales y subtotales
forma basada en otra array  array = np.eye(z,y,x, k = n) crea un array con unos en  diagonal empezando en la posicion k  array = np.identity(x) crea una matriz de identidad con  ceros en filas y unos en la diagonal, de forma cuadrada	<pre>Metodos de arrays nuevo_array = array.copy() crea un a copia del array np.transpose(array_bidimensional) cambia los filas del array a columnas y las columnas a filas np.transpose(array_multidimensional) cambia el número</pre>	np.sum(array) devuelve la suma de todos los elementos de los matrices np.mean(array) devuelve la media de todo el array np.std(array) devuelve la desviación estándar de todo np.var(array) devuelve la varianza de valores de	por cada elemento de array1 según si aparece el mismo valor en array2 np.setdiff1d(array1, array2) devuelve un array ordenado con los valores únicos que están en array1 pero no en array2 np.setxor1d(array1, array2) devuelve un array ordenado con los valores únicos que NO están en común de los dos arrays	Coeficiente de correlación de Pearson - nos permite conocer la intensidad y dirección de la relación entre las dos variables - coeficiente > 0: correlación positiva
NumPy Random	de columnas al número de arrays y viceversa; el número de filas no cambia  np.transpose(array_multidimensional, (z,y,x)) hace la	todo  np.min(array) devuelve el valor mínimo del array np.max(array) devuelve el valor máximo del array np.sum(array) devuelve la suma de los elementos del	Estadística	- coeficiente < 0: correlación negativa - coeficiente = 1 o -1: correlación total - coeficiente = 0: no existe relación lineal
<pre>np.random.seed(x) generador de números aleatorios, para que las funciones random que van después siempre cogerán los mismos valores "aleatorios"  Crear arrays con valores aleatorios array = np.random.randint(inicio, final, forma matriz)</pre>	transposicion segun lo que especificemos usando las posiciones de la tupla (0,1,2) de la forma original array = np.arange(n).reshape((y,x)) crea un array usando reshape para definir la forma array = np.reshape(array, (z,y,x)) crea un array con los valores de otro array usando reshape para definir la forma	array  np.cumsum(array) devuelve un array con la suma acumulada de los elementos a lo largo del array np.cumprod(array) devuelve un array con la multiplicación acumulada de los elementos a lo largo del array	Medidas de dispersión  Desviación respecto a la media la diferencia en valor absoluto entre cada valor de los datos y su media aritmética  diferencias = df['columna'] - df['columna'].mean() desviación_media = np.abs(diferencias)	<pre>df['columna1'].corr(df['columna2'] calcula la correlacion entre dos variables matriz_correlacion = df.corr() crea una matriz mostrando las correlaciones entre todos los variables sns.heatmap(df.corr()[['column1', 'column2']], cmap = 'color_palette', annot = True, vmin = -1, vmax = 1) crea una grafica heatmap de la matriz de correlaciones</pre>
crea un array de números aleatorios entre dos valores;  forma_matriz: (z,y,x)  z: número de arrays y: número de filas x: número de columnas  array = np.random.randint(inicio, final) devuelve un número aleatorio en el rango  array = np.random.rand(z,y,x) crea un array de floats aleatorias con la forma que le especificemos; por defecto genera números aleatorios entre 0-1  array = np.random.random sample((z,y,x)) crea un array de	array = np.swapaxes(array, posicion, posicion) intercambia dos ejes de una matriz usando las posiciones (z=0,y=1,x=2) de la forma original  Otras operaciones  np.sort(array) devuelve un array con los valores de cada fila ordenados en orden ascendente por defecto  np.sort(array, axis = 0) devuelve un array con los valores de cada columna ordenados en orden ascendente	Operaciones sin parámetro del axis: np.sqrt(array) devuelve un array con la raíz cuadrada no negativa de cada elemento del array np.exp(array) devuelve un array con el exponencial de cada elemento del array np.mod(array1, array2) devuelve un array con el resto de la división entre dos arrays np.mod(array1, n) devuelve un array con el resto de la división entre el array y el valor de n np.cos(array) devuelve un array con el coseno de cada elemento del array	Varianza medida de dispersión; la variabilidad respecto a la media df['columna'].var()  Desviación estándar o desviación típica la raíz cuadrada de la varianza; cuanto mayor sea, mayor será la dispersión o variabilidad en nuestros datos df['columna'].std()  Robustez - cuanto más cantidad de datos, más robustos	Sesgos (skewness) medida de la asimetría de la distribución de los valores de una variable alrededor de su valor medio - valor de sesgo positivo: sesgado a la derecha - valor de sesgo negativo: sesgado a la izquierda - valor de sesgo igual a 0: valores simetricos sns.displot(df['columna'], kde = True) crea un histograma que muestra la distribution de los valores import scipy.stats import skew skew(df['columna'] muestra el valor del sesgo de una
floats aleatorias con la forma que le especificemos; por defecto genera números aleatorios entre 0-0.9999999  array = np.random.z,y,x=None) devuelve un número aleatorio en 0 y 0.999999999999  np.round(np.random.rand(z,y,x), n) crear array con floats	<pre>np.sort(-array) cada fila ordenados en orden descendente np.round(array, decimals = x) valores del array redondeados a x decimales np.round(array, decimals = x) valores del array redondeados a x decimales valores del array redondeados a x decimales</pre>	<pre>np.sin(array) devuelve un array con el seno de cada elemento del array np.sin(array) devuelve un array con la tangente de cada elemento del array  Operaciones de comparación en arrays</pre>	<pre>1/n donde n es el numero de registros Coeficiente de variación el cociente entre la desviación típica y la media; cuanto mayor sea, mayor será la dispersión en nuestros datos df['columna'].std() / df['columna'].mean()</pre>	Intervalos de confianza describe la variabilidad entre la medida obtenida en un estudio y la medida real de la población (el valor real)
de n decimales  np.random.uniform(n,m, size = (z,y,x))  aleatorias de una distribución uniforme en el intervalo  entre n y m  np.random.binomial(n,m, size = (z,y,x))  con una distribución binomial; n es el numero total de  pruebas; m es la probabilidad de éxito	np.where(array > x) devuelve los indices de los valores que cumplan la condición, por fila y columna  Operaciones con arrays np.add(array1, array2) suma dos arrays	bidimensionales  np.any(array > n) devuelve True o False segun si cualquier valor del array cumpla con la condicion  np.any(array > n, axis = b) devuelve un array con  True o False por cada columna o fila según si algún valor de la fila o columna cumpla con la condición	<pre>Percentiles divide datos ordenados de menor a mayor en cien partes; muestra la proporción de datos por debajo de su valor percentil_n = np.percentile(df['columna'], n) saca el valor en el percentil n</pre>	<pre>import scipy.stats as st st.t.interval(alpha = n, df = len(df['columna']-1, loc = np.mean(df['columna']), scale = st.sem(df['columna'])) devuelve el rango de valores para lo cual hay un n% de</pre>
np.random.normal(loc = n, scale = m, size = (z,y,x)) genera números aleatorios de una distribución normal (curva de campana); loc es la media; scale es la desviación estándar np.random.permutation(array) devuelve un array con los mismos valores mezclados aleatoriamente	np.subtract(array1, array2) resta el array2 del array1 np.multiply(array1, array2) multiplica dos arrays np.divide(array1, array2) divide el array1 por el array2 array + n, n * array, etc operadores algebraicos	np.all(array > n) devuelve True o False segun si todos los valores del array cumpla con la condicion np.all(array > n, axis = b) devuelve un array con True o False por cada columna o fila según si todos los valores de la fila o columna cumplan con la condición	Rangos intercuartílicos medida de dispersión: diferencia entre cuartiles 75 y 25 q3, q1 = np.percentile(df["columna"], [75, 25]) saca los tercer y primer cuartiles rango_intercuartílico = q3 - q1	probabilidad que un valor real cae en ese rango alpha: porcentaje de confianza (p.ej. 90%, 95%, o 99%) df: los datos loc: la media scale: la desviación estándar