

Python Cheat Sheet 1
Variables ampliadas por text (CONCATENATION)
Para encadenar texto
<pre>categoria1 = "verde" color_detalle = categoria1 + ' ' + 'oscuro' print(categoria1 + ' oscuro') print(categoria1, 'oscuro')</pre>
type() and isinstance()
<pre>float/int/str(variable) cambia el tipo de data/type type(variable) devuelve: class 'float/int/str'</pre>
<pre>isinstance(variable, float/int/str) comprobar el tipo de dato (devuelve True/False)</pre>
Operaciones Algebraicas
<pre>+ sumar / dividir - restar // divider y redondear (modulus) * multiplicar % resto de una division (floor division) ** elevar round(x) redondear número x</pre>
Operaciones Binarias
<pre>== 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) and ambas verdaderas or ambas o solo una verdadera in/not in comprobar si hay un valor en una lista etc.</pre>
Metodos String
<pre>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") reemplaza 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</pre>
metodos permanentes (cambia el variable, no devuelve nada)

Listas [] Metodos no permanentes
<pre>lista = [] crea una lista vacia len(lista) devuelve el no. de elementos 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() sorted(lista) ordenar una lista de menor a mayor lista.copy() hacer una copia de la lista</pre>
Metodos con indices
<pre>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)</pre>
Listas – Acciones Permanentes
Ampliar una lista
<pre>[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, string, integer o tuple) a la 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)</pre>
Ordenar una lista
<pre>.sort() lista.sort()# ordena de menor a mayor, usar con (reverse=True) para ordenar de mayor a menor lista.reverse()# ordena los elementos al revés del orden guardado</pre>
Quitar elementos de una lista
<pre>.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</pre>

Diccionarios { key : value , }
<pre>diccionario = {x:y} compuestos por un key(x) unica y un valor(y) (cualquier tipo de datos) dict() variable = dict(x=y, m=n) crear un diccionario dicc.copy() crear una copia len(dicc) devuelve el no. de elementos (x:y) hay en el diccionario sorted(dicc) ordena los keys; usar con .items() para ordenar tuplas de los elementos o .values() para ordenar los valores solos</pre>
Diccionarios – Metodos
Obtener informacion de un diccionario
<pre>dicc.keys() devuelve todas las keys dicc.values() devuelve todos los valores 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)</pre>
Ampliar un diccionario
<pre>.update() dicc.update({x:y})# para insertar nuevos elementos dicc["key"] = valor# para insertar 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</pre>
Quitar elementos de un diccionario
<pre>dicc.pop(x)# elimina la key x (y lo devuelve) dicc.popitem()# elimina el ultimo par de key:value dicc.clear()# vacia el diccionario</pre>
Tuplas (,) inmutables, indexados
<pre>tupla = (x,y) tuplas se definen con () y , o solo , tupla1 + tupla2 juntar tuplas tuple(lista) crear tuplas de una lista tuple(dicc) crear tuplas de los keys de un diccionario tuple(dicc.values()) crear tuplas de los valores tuple(dicc.items()) crear tuplas de los key:valores len(tupla) devuelve el no. de elementos 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 convertirla en una lista y luego a tupla*</pre>

zip()
<pre>zip(iterable1, iterable2) crea una lista de tuplas de parejas de los elementos de las dos listas (mientras se puede) listzip.sort() ordena las tuplas del zip por el primer elemento</pre>
Sets {} no permiten duplicados, no tienen orden
<pre>set = {x,y} set(iterable) solo permite un argumento iterable; elimina duplicados in/not in comprobar si hay un elemento len(set) devuelve el no. de elementos Ampliar un set set.add(x)# añadir un elemento set.update(set o lista)# añadir uno o mas elementos con [] o {} o un variable tipo lista o set Quitar elementos de un set 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 set2 estan en set1</pre>
input()
<ul style="list-style-type: none">permite obtener texto escrito por teclado del usuario <pre>input("el texto que quieres mostrar al usuario") 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</pre>

Sentencias de control
<pre>if ... elif ... else if estableca una condición para que se ejecute el código que 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") elif x == y: print("x es igual que y") else: print("x e y son iguales") while • 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")</pre>
For loops
<ul style="list-style-type: none">sirven para iterar por todos los elementos de un variable que tiene que ser un iterable (lista, diccionario, tupla, set, or string)se pueden combinar con if ... elif ... else, while, u otro for loopen diccionarios por defecto itera por las keys; podemos usar dicc.values() para acceder a los valores <pre>for i in lista: print("hola mundo")</pre>
List comprehension
<ul style="list-style-type: none">su principal uso es para crear una lista nueva de un un for loop en una sola línea de código <pre>[lo que queremos obtener iterable condición (opcional)]</pre>
try ... except
<p>Se usan para evitar que nuestro código se pare debido a un error en el código. Se puede imprimir un mensaje que avisa del error.</p> <pre>try: print("2.split()) except: print("no funciona")</pre>
range()
<ul style="list-style-type: none">nos devuelve una lista de números que por defecto se aumentan de uno en uno empezando por 0 <pre>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 stop) tambien se puede especificar saltos</pre>

Python Cheat Sheet 2
Funciones
Definir una funcion: <code>def nombre_funcion(parametro1, parametro2, ...): return valor_del_return</code>
Lllamar una funcion: <code>nombre_funcion(argumento1, argumento2, ...)</code>
return: es opcional, pero sin return devuelve None parametros por defecto: – siempre deben ser lo ultimo
*args: una tupla de argumentos sin limite **kwargs: diccionarios cuyas keys se convierten en parámetros y sus valores en los argumentos de los parámetros
<code>def nombre_funcion(parametros, *args, **kwargs, parametro_por_defecto = valor) arg/kwarg: sin */** dentro de la funcion arg[0]</code>
Lllamar una funcion con *args: <code>nombre_funcion(argumento, argumento, argumento, ...) o nombre_funcion(*[lista_o_tupla_de_args])</code>
Lllamar una funcion con **kwargs: <code>nombre_funcion(**diccionario)</code>
Clases
Definir una clase: <code>class NombreClase: def __init__(self, atributo1, atributo2): self.atributo1 = atributo1 self.atributo2 = atributo2 self.atributo_por_defecto = ‘valor’ def nombre_funcion1(self, parametros) self.atributo += 1 return f“el nuevo valor es {self.atributo}”</code>
Definir una clase hija: <code>class NombreClaseHija(NombreClaseMadre): def __init__(self, atributo1, atributo2): super().__init__(atributo_hereditado1, ...) def nombre_funcion_hija (self, parametros):</code>
Crear un objeto de la clase: <code>variable_objeto = NombreClase(valor_atributo1, valor_atributo2)</code> instanciar (crear) un objeto <code>variable_objeto.atributo</code> devuelve el valor del atributo guardado para ese objeto <code>variable_objeto.atributo = nuevo_valor</code> para cambiar el valor del atributo <code>variable_objeto.nombre_funcion()</code> llamar una funcion
<code>print(help(NombreClase))</code> imprime informacion sobre la clase

Regex
- una abreviatura de `expresión regular`, `regex` es una cadena de texto que permite crear patrones que ayudana a emparejar, localizar y gestionar strings <code>import re</code> para poder trabajar con regex
Operadores communes de regex + coincide con el carácter precedente una o más veces * coincide con el carácter precedente cero o más veces u opcional ? indica cero o una ocurrencia del elemento precedente . coincide con cualquier carácter individual ^ coincide con la posición inicial de cualquier string \$ coincide con la posición final de cualquier string
Sintaxis básica de regex \\w cualquier caracter de tipo alfabético \\d cualquier caracter de tipo númeroico \\s espacios \\n saltos de línea \\W cualquier caracter que no sea una letra \\D cualquier caracter que no sea un dígitos \\S cualquier elemento que no sea un espacio () aísla sólo una parte de nuestro patrón de búsqueda que queremos devolver [] incluye todos los caracteres que queremos que coincidan e incluso incluye rangos como este: a-z y 0-9 es como el operador ‘or’ \ señala una secuencia especial (escapar caracteres especiales) { } Exactamente el número especificado de ocurrencias {n} Exactamente n veces {n,} Al menos n veces {n,m} Entre n y m veces
Métodos Regex <code>re.findall(“patron”, string)</code> busca en todo el string y devuelve una lista con todas las coincidencias en nuestro string <code>re.search(“patron”, string_original)</code> busca en todo el string y devuelve un objeto con la primera coincidencia en nuestro string <code>re.match(“patron”, “string_original)</code> busca en la primera linea del string y devuelve un objeto con la primera coincidencia en nuestro string <code>resultado_match.span()</code> devuelve la referencia de las posiciones donde hizo el “match” <code>resultado_match.group()</code> devuelve el element resultando de la coincidencia del “match” <code>re.split(“patron”, “string_original”) </code> busca en todo el string y devuelve una lista con los elementos separados por el patron <code>re.sub(“patron”, “string_nuevo”, “string_original”) </code> busca en todo el string y devuelve un string con el element que coincide

Modulos/Librerias (paquetes de funciones)
Importar y usar modulos y sus funciones <code>import modulo</code> para importar un modulo <code>from modulo import funcion</code> importar solo una funcion <code>modulo.funcion()</code> usar una funcion de un modulo <code>modulo.clase.funcion()</code> para usar una funcion de una clase <code>import modulo as md</code> asignar un alias a un modulo
Libreria os <code>os.getcwd()</code> devuelve la ruta de donde estamos trabajando; se puede guardar en un variable e.g. ruta = os.getcwd() <code>os.listdir()</code> devuelve una lista de los archivos y carpetas donde estamos trabajando <code>os.listdir(‘carpeta’)</code> devuelve los contenidos de otra carpeta <code>os.chdir(‘ruta’)</code> cambia la carpeta en la que estes <code>os.mkdir(‘nueva_carpeta’)</code> crear una nueva carpeta <code>os.rename(‘nombre_carpeta’, ‘nueva_nombre’)</code> cambia el nombre de una carpeta <code>os.rmdir(‘carpeta’)</code> borra la carpeta
Libreria shutil from shutil import rmtree <code>rmtree(‘carpeta’)</code> borra la carpeta y subcarpetas
Abrir y cerrar ficheros Primero hay que guardar la ruta del archivo: ubicacion_carpeta = os.getcwd() nombre_archivo = “text.txt” <code>ubicacion_archivo = ubicacion_carpeta + “/” + nombre_archivo</code> <code>f = open(ubicacion_archivo)</code> abrir un archivo en variable f <code>f.close()</code> cerrar un archivo * IMPORTANTE * <code>with open(ubicacion_archivo) as f:</code> codigo e.g. <code>variable = f.read()</code> abre el archivo solo para ejecutar el codigo indicado (y despues lo deja)
Encoding <code>from locale import getpreferredencoding</code> <code>getpreferredencoding()</code> para saber que sistema de encoding estamos usando <code>f = open(ubicacion_archivo, encoding="utf-8")</code> abrir un archivo y leerlo con el encoding usado; guardar con .read()
mode: argumento opcional al abrir un archivo <code>r</code> – read <code>w</code> – write - sobreescribe <code>x</code> – exclusive creation, sólo crearlo si no existe todavía <code>a</code> – appending, añadir texto al archivo sin manipular el texto que ya habia hay que anadir otra letra: <code>t</code> – texto – leer en texto <code>b</code> – bytes – leer en bytes (no se puede usar con encoding)
<code>f = open(ubicacion_archivo, mode = “rt”)</code>
Leer ficheros <code>f.read()</code> leer el contenido de un archivo <code>f.read(n)</code> leer los primeros n caracteres de un archivo <code>variable = f.read()</code> guardar el contenido del archivo (o n caracteres de un archivo) en un variable <code>f.readline(n)</code> por defecto devuelve la primera linea o n lineas <code>f.readlines()</code> devuelve una lista de todas las lineas del archivo (cada linea es un elemento); se usa vacio sin n y list_name[x:] para seleccionar lineas especificas
Escribir en ficheros <code>with open(ubicacion_archivo, “w”) as f:</code> f.write(“Texto que va en el fichero.”) para escribir <code>with open(ubicacion_archivo, “a”) as f:</code> f.write(“Texto que va en el fichero.”) para anadir texto f.writelines(‘lista’) para anadir lineas de texto de una lista

Ficheros xml
<code>import xml.etree.ElementTree as ET</code> importa la librería xml <code>variable_tree = ET.parse(‘ruta/archivo.xml’)</code> abre el archivo <code>variable_root = variable_tree.getroot()</code> saca el elemento que envuelve todo (el elemento raíz) en una lista <root> <child_tag atributo1=“valor” atributo2=valor> <subchild_tag> elemento </subchild_tag> </child_tag> </root> <code>variable_root.tag</code> devuelve el nombre del tag del raiz <code>variable_root.attrib</code> devuelve los atributos del fichero <code>variable_root.find(“tag”).find(“childtag”).text</code> devuelve la primera ocasión en que el tag de un elemento coincida con el string <code>variable_root.findall(“tag”).findall(“childtag”).text</code> devuelve todos los elementos cuyos tag coincide
MySQL Connector/Python
Conectar a una base de datos <code>import mysql.connector</code> para importar MySQL Connector pip install mysql-connector pip install mysql-connector-Python <code>connect()</code> para conectar a una base de datos: <code>variable_cnx = mysql.connector.connect(user='root', password='AlumnaAdalab', host='127.0.0.1', database='nombre_BBDD')</code> <code>from mysql.connector import errorcode</code> importar errores <code>mysql.connector.Error</code> se puede usar en un try/except <code>cnx.close()</code> desconectar de la base de datos
Realizar queries <code>variable_cursor = cnx.cursor()</code> crear el objeto cursor que nos permite comunicar con la base de datos <code>variable_cursor.close()</code> desconectar el cursor <code>variable_query = (“SQL Query”)</code> guardar un query en un variable <code>variable_cursor.execute(variable_query)</code> ejecutar el query; devuelve una lista de tuplas <code>import datetime</code> sacar fechas en el formato AAAA-MM-DD <code>datetime.date(AAAA, M, D)</code> devuelve el formato de fecha <code>variable_query = “SQL Query... %s AND %s”</code> query dinamica <code>variable_cursor.execute(query, (variable1, variable2))</code> valores que van en lugar de los %s <code>variable_cursor.execute(“SHOW DATABASES”)</code> mostrar las BBDD <code>variable_cursor.execute(“SHOW TABLES”)</code> mostrar las tablas de la BBDD indicado en la conexión <code>variable_cursor.execute(“SHOW TABLES”)</code> <code>variable_cursor.execute(“SHOW COLUMNS FROM bbdd.table”)</code> mostrar las columnas de la tabla especificada; hay que conectarse a la bbdd information_schema
Argumentos cursor: <code>variable_cursor = cnx.cursor([arg=value[, arg=value]...])</code> <code>buffered=True</code> devuelve todas las filas de la bbdd <code>raw=True</code> el cursor no realizará las conversiones automáticas entre tipos de datos <code>dictionary=True</code> devuelve las filas como diccionarios <code>named_tuple=True</code> devuelve las filas como named tuples <code>cursor_class</code> un argumento que se puede usar para indicar que subclase queremos usar para instanciar el nuevo cursor

MySQL Connector/Python
Obtener resultados de una query <code>variable_cursor.fetchone()</code> devuelve el primer resultado <code>variable_cursor.fetchall()</code> devuelve todos los resultados como iterable – cada fila es una tupla
Pandas dataframe with SQL <code>import pandas as pd</code> <code>variable_df = pd.DataFrame(variable_resultado_fetchall, columns = [‘columna1’, ‘columna2’, ...])</code> crear un dataframe con los resultados de una query en una variable <code>variable_df.head(n)</code> devuelve las n primeras filas del df, o 5 por defecto <code>variable_df = pd.read_sql_query(variable_query, variable_cnx)</code> convertir los resultados de la query en df <code>pd.read_sql(variable_query, variable_cnx)</code> <code>variable_df.to_csv(“nombre_archivo.csv”)</code> guardar en csv <code>variable_df.to_string()</code> formatear el dato en string <code>variable_df.to_latex()</code> formatear el dato en un string que facilite la inserción en un documento latex
Crear y alterar una base de datos <code>variable_cursor.execute(“CREATE DATABASE nombre_BBDD”)</code> <code>variable_cursor.execute(“CREATE TABLE nombre_tabla (nombre_columna TIPO, nombre_columna2 TIPO2)”)</code> <code>variable_cursor.execute(“ALTER TABLE nombre_tabla ALTERACIONES”)</code>
Insertar datos <code>variable_query = “INSERT INTO nombre_tabla (columna1, columna2) VALUES (%s, %s)”</code> <code>variable_valores = (valor1, valor2)</code> <code>variable_cursor.execute(variable_query, variable_valores)</code> otro método: <code>variable_query = “UPDATE nombre_tabla SET nombre_columna = “nuevo_valor” WHERE nombre_columna = “valor”</code>
Insertar múltiples filas a una tabla <code>variable_valores_en_tuplas = ((valor1columna1, valor1columna2), (valor2columna1, valor2columna2))</code> <code>variable_cursor.executemany(variable_query, variable_valores_en_tuplas)</code>
<code>variable_conexion.commit()</code> después de ejecutar la inserción, para que los cambios efectúen en la BBDD <code>variable_conexion.rollback()</code> se puede usar después de execute y antes de commit para deshacer los cambios <code>print(variable_cursor.rowcount, “mensaje”)</code> imprimir el número de filas en las cuales se han tomado la accion
Eliminar registros <code>variable_query = “DROP TABLE nombre_tabla”</code>
Añadir errores importar errorcode y usar try/except: <code>try:</code> accion <code>except mysql.connector.Error as err:</code> print(err) print("Error Code:", err.errno) print("SQLSTATE", err.sqlstate) print("Message", err.msg)

Python Cheat Sheet 3	DataFrames	DataFrames: carga de datos	Metodos de DataFrames	Filtrados de datos
Pandas	Crear DataFrames <code>df = pd.DataFrame(data, index, columns)</code> data : NumPy Array, diccionario, lista de diccionarios index : indice que por defecto se asigna como 0-(n-1), n siendo el número de filas; index = [lista] para asignar “etiquetas” (nombres de filas) column : nombre de las columnas; por defecto 0-(n-1); columns =[lista] para poner mas nombres	Carga de datos <code>df = pd.read_csv(“ruta/nombre_archivo.csv”)</code> crear un dataframe de un archivo de Comma Separated Values <code>df = pd.read_csv(“ruta/nombre_archivo”, sep= “;”)</code> crear un dataframe de un csv si el separador es ; <code>df = pd.read_csv(“ruta/nombre_archivo”, index_col= 0)</code> crear un dataframe de un csv si el archivo ya tiene una columna indice <code>df = pd.read_excel(“ruta/nombre_archivo.xlsx”)</code> crear un dataframe de un archivo de Excel - si sale “ ImportError:... openpyxl... ”, en el terminal: pip3 install openpyxl o pip install openpyxl <code>df = pd.read_json(“ruta/nombre_archivo.json”)</code> crear un dataframe de un archivo de JavaScript Object Notation (formato crudo) <code>df = df[‘data’].apply(pd.Series)</code> convertir el dataframe de json en un formato legible <code>df = pd.read_clipboard(sep=‘\t’)</code> crear un dataframe de datos en forma de dataframe en el clipboard; el separador podria ser \n ; , etc. Pickle : modulo que serializa objetos (convertir objetos complejos en una serie de bytes, en este caso en formato binario) para guardarlos en un archivo with open(‘ruta/nombre_archivo.pkl’, ‘wb’) as f: pickle.dump(df,f) pone los datos de un dataframe en el archivo.pkl pd.read_pickle(‘ruta/nombre_archivo.csv’).head(n) leer n filas y 5 columnas del archivo pickle pd.read_parquet(‘ruta/nombre_archivo.parquet’) leer un archivo parquet pd.read_sas(‘ruta/nombre_archivo.sas7bdat’, format = ‘sas7bdat’) leer un archivo SAS de formato SAS7BDAT pd.read_spss(‘ruta/nombre_archivo.sav’) leer un archivo SAS de formato SAS7BDAT	Metodos para explorar un dataframe <code>df.head(n)</code> devuelve las primeras n lineas del dataframe <code>df.tail(n)</code> devuelve las últimas n lineas del dataframe <code>df.sample(n)</code> devuelve n filas aleatorias de nuestro dataframe, o uno por defecto <code>df.shape</code> devuelve el número de filas y columnas <code>df.dtypes</code> devuelve el tipo de datos que hay en cada columna <code>df.columns</code> devuelve los nombres de las columnas <code>df.describe</code> devuelve un dataframe con un resumen de los principales estadísticos de las columnas numéricas <code>df.info()</code> devuelve un resumen sobre el no. de columnas, nombres de columnas, numero de valores no nulos y los tipos de datos <code>df[“nombre_columna”].unique()</code> o <code>df.nombre_columna.unique()</code> devuelve un array con los valores únicos de la columna <code>df[“nombre_columna”.value_counts()</code> o <code>df.nombre_columna.value_counts()</code> devuelve una serie con el recuento de valores únicos en orden descendente <code>df.isnull()</code> o <code>df.isna()</code> devuelve True o False según si cada valor es nulo o no <code>df.isnull().sum()</code> o <code>df.isna().sum()</code> devuelve el número de valores nulos por columnas <code>df.duplicated().sum()</code> devuelve el numero de filas duplicadas <code>df.corr()</code> devuelve la correlación por pares de columnas, excluyendo valores NA/nulos Realizar cambios en el dataframe: <code>df.set_index([“nombre_columna”, inplace = True)</code> establece el índice utilizando uno o mas columnas; puede sustituir o ampliar un índice existente <code>inplace = True</code> los cambios sobrescriben sobre el df * cuando una columna se cambia a índice ya no es columna * <code>df.reset_index(inplace = True)</code> quitar una columna como indice para que vuelva a ser columna; crea un dataframe de una serie <code>df.rename(columns = {“nombre_columna”: “nombre_nueva”}, inplace = True)</code> cambia los nombres de una o mas columnas ejemplo de dict comprehension para crear diccionario sobre las columnas existentes de un dataframe: <code>diccionario = {col : col.upper() for col in df.columns}</code> <code>df.rename(columns = diccionario, inplace = True)</code> cambia los nombres de las columnas según el diccionario <code>df.drop(columns = [“columna1”, “columna2”], axis = b)</code> eliminar una o mas columnas o filas segun lo que especificamos <code>df.drop_duplicates(inplace = True)</code> elimina filas duplicadas <code>df.dropna(inplace = True)</code> quitar nulos <code>axis = 1</code> columnas <code>axis = 0</code> filas <code>df[“columna_nueva”] = pd.cut(x = df[“nombre_columna”, bins = [n,m,l..], labels = [‘a’, ‘b’, ‘c’])</code> separa los elementos de un dataframe en diferentes intervalos (n-m, m-l, etc), creando una columna nueva que indica en cual intervalo cae el valor; con labels se puede asignar un string a cada intervalo <code>df.replace(to_replace = valor, value = valor_nuevo, inplace = True)</code> reemplaza cierto valor por otro que especificamos <code>df[“nombre_columna”.replace(to_replace = valor, value = valor_nuevo, inplace = True)</code> reemplaza cierto valor en una columna por otro que especificamos <code>df[“nombre_columna”] = df[“nombre_columna”] + x</code> reemplaza los valores de la columna por el valor + x (o otro valor que indicamos)	<code>pd.options.display.max_columns = None</code> ejecutar antes del df.head() para poder ver todas las columnas Filtrado por una columna con operadores de comparación <code>df_filtrado = df[df[“nombre_columna”] == valor]</code> extrae las filas donde el valor de la columna igual al valor dado *se puede usar con cualquier operador de comparación* Filtrado por multiples columnas con operadores logicos <code>&</code> and <code> </code> or <code>~</code> not <code>df_filtrado = df[(df[“columna1”] == valor) & (df[“columna2”] == valor) & (df[“columna3”] > n valor)]</code> extrae las filas donde los valores de las columnas cumplan las condiciones en parentesis <code>df_filtrado = df[(df[“columna1”] == valor) (df[“columna1”] == valor)</code> extrae las filas donde los valores de las columnas cumplan con una condición u otra <code>df_filtrado = ~(df[df[“columna1”] == valor])</code> extrae las filas donde los valores de las columnas NO cumplan con la condición Metodos de pandas de filtrar <code>df_filtrado = df[df[“nombre_columna”.isin(iterable)]</code> extrae las filas cuyas valores de la columna nombrada están en el iterable (una lista, serie, dataframe o diccionario) <code>df_filtrado= df[df[“nombre_columna”.str.contains (patron, regex = True, na = False)]</code> extrae las filas cuyas valores de la columna nombrada contienen el patron de regex <code>df_filtrado = df[df[“nombre_columna”.str.contains (“substring”, case = False, regex = False)]</code> extrae las filas cuyas valores de la columna nombrada contienen el substring, no siendo case sensitive <code>df_filtrado = df[df[“nombre_columna”.str.contains (“substring”, case = False, regex = False)]</code> extrae las filas cuyas valores de la columna nombrada contienen el substring, no siendo case sensitive <code>df[pd.notnull(df[“nombre_columna”])]</code> devuelve las filas que no tiene valores nulos en la columna especificada Reemplazar valores basados en indices y condiciones: <code>indices_filtrados = df.index[df[“columna”] == “valor”]</code> <code>for indice in indices_filtrados:</code> <code>df[“nombre_columna”.iloc[indice] = “valor_nuevo”</code> Reemplazar valores basados en metodos NumPy: <code>df[“nueva_columna”] = np.where(df[“nombre_columna”] > n, “categoria_if_true”, “categoria_if_false”)</code> crea una nueva columna con los valores basados en una condición <code>df[“nueva_columna”] = np.select(lista_de_condiciones, lista_de_opciones)</code> crea una nueva columna con los valores basados en multiples condiciones
Series: estructuras en una dimension				
Crear series <code>serie = pd.Series()</code> crear serie vacía <code>serie = pd.Series(array)</code> crear serie a partir de un array con el indice por defecto <code>serie = pd.Series(array, index = [‘a’, ‘b’, ‘c’...])</code> crear una serie con indice definida; debe ser lista de la misma longitud del array <code>serie = pd.Series(lista)</code> crear una seria a partir de una lista <code>serie = pd.Series(número, indice)</code> crear una serie a partir de un escalar con la longitud igual al número de indices <code>serie = pd.Series(diccionario)</code> crear una serie a partir de un diccionario		Acceder a informacion de un DataFrame <code>df.loc[“etiqueta_fila”, “etiqueta_columna”]</code> devuelve el contenido de un campo en una columna de una fila <code>df.loc[“etiqueta_fila”,:]</code> devuelve los valores de todas las columnas de una fila <code>df.loc[:,“etiqueta_columna”]</code> devuelve los valores de todas las filas de una columna <code>df.iloc[indice_fila, indice_columna]</code> devuelve el contenido de un campo en una columna de una fila <code>df.iloc[indice_fila, :]</code> devuelve los valores de todas las columnas de una fila <code>df.iloc[:,indice_columna]</code> devuelve el contenido de un campo en una columna de una fila <code>df.loc[[lista_etiquetas_filas], [lista_etiquetas_columnas]]</code> devuelve el contenido de varias filas / varias columnas <code>df.loc[[lista_indices_filas], [lista_indices_columnas]]</code> devuelve el contenido de varias filas / varias columnas - se puede usar los indices/rangos de las listas [start:stop:step] dentro de los loc/iloc <code>df.loc[df.etiqueta > x]</code> seleccionar datos basado en una condición usando operadores comparativos <code>df.loc[(df.etiqueta > x) & (df.etiqueta == y)]</code> seleccionar datos que tienen que cumplir las dos condiciones (and) <code>df.loc[(df.etiqueta > x) (df.etiqueta == y)]</code> seleccionar datos que tienen que deben cumplir una de las dos condiciones (or) <code>df.iloc[list(df.etiqueta > x), :]</code> iloc no acepta una Serie booleana; hay que convertirla en lista <code>variable_df.head(n)</code> devuelve las n primeras filas del df, o 5 por defecto		
		Crear columnas <code>df[“nueva_columna”] = (df[“etiqueta_columna”] + x)</code> crea una nueva columna basada en otra <code>df = df.assign(nueva_columna= df[“etiqueta_columna”] + x)</code> crea una nueva basada en otra <code>df = df.assign(nueva_columna= [lista_valores])</code> crea una nueva columna de una lista de valores *tiene que ser de la misma longitud como el número de filas del dataframe* <code>df.insert(indice_nueva_columna, “nombre_columna”, valores)</code> crea una nueva columna en la indice indicada <code>allow_duplicates = True</code> parametro cuando queremos permitir columnas duplicadas (por defecto es False)		
	Eliminar columnas <code>df = df.drop(columns = [“column1”, “column2”])</code> eliminar columnas			

Python Cheat Sheet 4

Pandas

Union de datos

.concat() unir dataframes con columnas en comun

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 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 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 las índices 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 = 'columna_df1', right_on = 'columna_df2') left merge

parametros:

- how = 'left' | 'right' | 'outer' | 'inner' | 'cross'**
- on = columna | [columna1, columna2, etc]** si las columnas se llaman igual en los dos dataframes
- left_on = columna_df1 | right_on = columna_df2** para especificar por donde hacer el merge
- suffixes = ['left', 'right']** por defecto nada, el sufijo que apareciera en columnas duplicadas

.join() unir dataframes por los indices

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 union; tienen que tener el mismo nombre en los dos dataframes
- lsuffix = 'string' | rsuffix = 'string'** por defecto nada, el sufijo que apareciera en columnas duplicadas

Group By

df_groupby = df.groupby("columna_categoria") crea un objeto DataFrameGroupBy; agrupa los valores segun las categorías de los 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 categorías y los valores son listas de los índices de cada elemento en la categoría

df_grupo1 = df_groupby.get_group("grupo1") devuelve un dataframe con los resultados de un grupo (la categoría indicada como grupo1)

Cálculos con groupby:

df_nuevo = df.groupby("columna_categoria").mean() devuelve un dataframe con la media de todas las columnas de valores numéricos, por categoría

df_nuevo = df.groupby("columna_categoria")["columna1"].mean() devuelve un dataframe con la media de la columna especificada

count() número de observaciones no nulas	median() mediana de los valores
describe() resumen de los principales estadísticos	min() valor mínimo
sum() suma de todos los valores	max() valor máximo
mean() media de los valores	std() desviación estándar
	var() varianza

df_nuevo = df.groupby("columna_categoria", dropna = False) ["columna_valores"].agg([nombre_columna = 'estadistico1', nombre_columna2 = 'estadistico2']) añade columnas con los cálculos de los estadísticos especificados

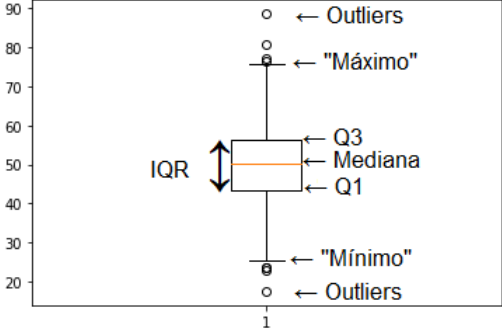
dropna = False para tener en cuenta los Nan en los cálculos (por defecto es True)

Apply
<p>- apply() toma una función como argumento y la aplica a lo largo de un eje del DataFrame</p>
<p>df['columna_nueva'] = (df['col_1'].apply(función)) crea una columna nueva con los valores de otra columna transformados según la función indicada</p>
<p>apply() con datetime</p> <p>import datetime</p> <p>df['columna_fecha'] = df['columna_fecha'] .apply(pd.to_datetime) cambia una columna de datos tipo fecha en el formato datetime</p> <p>def sacar_año(x): return x.strftime("%Y")</p> <p>df['columna_año'] = (df['columna_fecha'] .apply(sacar_año))</p> <p>crea una columna nueva del año solo usando un método de la librería datetime; ("%B") para meses</p>
<p>apply() con funciones de mas de un parametro</p> <p>df['columna_nueva'] = df.apply(lambda nombre: función(nombre['columna1'], nombre['columna2']), axis = b) crea una columna nueva usando una función que coge dos parámetros (columna 1 y columna2)</p>
NumPy (Numerical Python)
Crear arrays
<p>Crear arrays de listas</p> <p>array = np.array(lista, dtype= tipo) crea un array unidimensional de una lista</p> <p>array = np.array([lista1, lista2]) crea un array bidimensional de dos listas</p> <p>array = np.array([listadelistas1, listadelistas2]) crea un array bidimensional de dos listas</p>
<p>Crear otros tipos de arrays</p> <p>array = np.arange(valor_inicio, valor_final, saltos) crea un array usando el formato [start:stop:step]</p> <p>array = np.ones(z,y,x) crea un array de todo unos de la forma especificada</p> <p>array2 = np.ones_like(array1) crea un array de todo unos de la forma basada en otra array</p> <p>array = np.zeros(z,y,x) crea un array de todo zeros de la forma especificada</p> <p>array2 = np.zeros_like(array1) crea un array de todo zeros de la forma basada en otra array</p> <p>array = np.empty((z,y,x), tipo) crea un array vacio con datos por defecto tipo float</p> <p>array2 = np.empty_like(array1) crea un array vacia con la forma basada en otra array</p> <p>array = np.eye(z,y,x, k = n) crea un array con unos en diagonal empezando en la posicion k</p> <p>array = np.identity(x) crea una matriz de identidad con ceros en filas y unos en la diagonal, de forma cuadrada</p>
<p>Guardar y salvar arrays en .txt</p> <p>np.savetxt('ruta/nombre_fichero.txt', array) guardar un array de uno o dos dimensiones como .txt</p> <p>variable = np.loadtxt('ruta/nombre_fichero.txt', dtype = tipo) cargar datos de un archivo txt que tiene el mismo número de valores en cada fila</p>

Indices, Subsets, Metodos de Arrays
<p>Indices de arrays</p> <p>array[i] devuelve la indice i; las indices de los arrays unidimensionales funcionan igual que las listas</p> <p>array[i, j] o array[i][j] devuelve el elemento de la columna j de la fila i</p> <p>array[:,n] seleccionar todas las filas y las columnas hasta n-1</p> <p>array[h, i, j] o array[h][i][j] devuelve el elemento de la columna j de la fila i del array h</p> <p>array[h][i][j] = n cambiar el valor del elemento en esta posicion al valor n</p>
<p>Subsets</p> <p>array > n devuelve la forma del array con True o False según si el elemento cumple con la condición o no</p> <p>array[array > n] devuelve un subset: todos los valores que cumplen la condición en una lista dentro de un array</p> <p>array[(array > n) & (array < m)] devuelve un subset: todos los valores que cumplen las condiciones en una lista dentro de un array; se puede usar para “or”</p>
<p>Metodos de arrays</p> <p>nuevo_array = array.copy() crea un a copia del array</p> <p>np.transpose(array_bidimensional) cambia los filas del array a columnas y las columnas a filas</p> <p>np.transpose(array_multidimensional) cambia el número de columnas al número de arrays y viceversa; el número de filas no cambia</p> <p>np.transpose(array_multidimensional, (z,y,x)) hace la transposicion segun lo que especificemos usando las posiciones de la tupla (0,1,2) de la forma original</p> <p>array = np.arange(n).reshape((y,x)) crea un array usando reshape para definir la forma</p> <p>array = np.reshape(array, (z,y,x)) crea un array con los valores de otro array usando reshape para definir la forma</p> <p>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</p>
<p>Otras operaciones</p> <p>np.sort(array) devuelve un array con los valores de cada fila ordenados en orden ascendente por defecto</p> <p>np.sort(array, axis = 0) devuelve un array con los valores de cada columna ordenados en orden ascendente</p> <p>np.sort(-array) devuelve un array con los valores de cada fila ordenados en orden descendente</p> <p>np.round(array, decimals = x) devuelve un array con los valores del array redondeados a x decimales</p> <p>np.round(array, decimals = x) devuelve un array con los valores del array redondeados a x decimales</p> <p>np.where(array > x) devuelve los indices de los valores que cumplan la condición, por fila y columna</p>
<p>Operaciones con arrays</p> <p>np.add(array1, array2) suma dos arrays</p> <p>np.subtract(array1, array2) resta el array2 del array1</p> <p>np.multiply(array1, array2) multiplica dos arrays</p> <p>np.divide(array1, array2) divide el array1 por el array2</p> <p>array + n, n * array, etc. - operadores algebraicos</p>

Operaciones estadísticas y matemáticas
<p>Operaciones estadísticas y matemáticas</p> <p>El parametro axis en arrays bidimensionales:</p> <p>axis = 0 columnas</p> <p>axis = 1 filas</p> <p>- si especificamos el axis, la operación devuelve el resultado por cada fila o columna.</p> <p>Por ejemplo:</p> <p>np.sum(array, axis = 0) devuelve un array con la suma de cada fila</p>
<p>El parametro axis en arrays multidimensionales:</p> <p>axis = 0 dimensión</p> <p>axis = 1 columnas</p> <p>axis = 2 filas</p> <p>- si especificamos el axis, la operación devuelve el resultado por cada dimensión, fila o columna.</p> <p>Por ejemplo:</p> <p>np.sum(array_3D, axis = 0) devuelve un array de una matriz con la suma de todas las matrices</p> <p>np.sum(array_3D, axis = 1) devuelve un array donde las filas contienen las sumas de las columnas de cada matriz</p>
<p>Operaciones con parámetro del axis:</p> <p>np.sum(array_3D) devuelve la suma de todos los elementos de los matrices</p> <p>np.mean(array) devuelve la media de todo el array</p> <p>np.std(array) devuelve la desviación estándar de todo</p> <p>np.var(array) devuelve la varianza de valores de todo</p> <p>np.min(array) devuelve el valor mínimo del array</p> <p>np.max(array) devuelve el valor máximo del array</p> <p>np.sum(array) devuelve la suma de los elementos del array</p> <p>np.cumsum(array) devuelve un array con la suma acumulada de los elementos a lo largo del array</p> <p>np.cumprod(array) devuelve un array con la multiplicación acumulada de los elementos a lo largo del array</p>
<p>Operaciones sin parámetro del axis:</p> <p>np.sqrt(array) devuelve un array con la raíz cuadrada no negativa de cada elemento del array</p> <p>np.exp(array) devuelve un array con el exponencial de cada elemento del array</p> <p>np.mod(array1, array2) devuelve un array con el resto de la división entre dos arrays</p> <p>np.mod(array1, n) devuelve un array con el resto de la división entre el array y el valor de n</p> <p>np.cos(array) devuelve un array con el coseno de cada elemento del array</p> <p>np.sin(array) devuelve un array con el seno de cada elemento del array</p> <p>np.sin(array) devuelve un array con la tangente de cada elemento del array</p>
<p>Operaciones de comparación en arrays bidimensionales</p> <p>np.any(array > n) devuelve True o False segun si cualquier valor del array cumpla con la condicion</p> <p>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</p> <p>np.all(array > n) devuelve True o False segun si todos los valores del array cumpla con la condicion</p> <p>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</p>

Funciones de conjuntos
<p>np.unique(array) devuelve un array con los valores únicos del array ordenados</p> <p>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</p> <p>np.unique(array, return_inverse=True) devuelve un array con los valores únicos del array ordenados y un array con las posiciones de cada elemento de cada valor</p> <p>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</p> <p>np.unique(array, axis = b) devuelve un array con los valores únicos ordenados de las filas o columnas</p>
<p>Funciones para arrays unidimensionales</p> <p>np.intersect1d(array1, array2) devuelve un array con los valores únicos de los elementos en común de dos arrays</p> <p>np.intersect1d(array1, array2, return_indices=True) 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</p> <p>np.union1d(array1, array2) devuelve un array ordenado con los elementos resultantes de unir dos arrays (valores únicos)</p> <p>np.in1d(array1, array2) devuelve un array con True o False por cada elemento de array1 según si aparece el mismo valor en array2</p> <p>np.setdiff1d(array1, array2) devuelve un array ordenado con los valores únicos que están en array1 pero no en array2</p> <p>np.setxor1d(array1, array2) devuelve un array ordenado con los valores únicos que NO están en común de los dos arrays</p>
NumPy Random
<p>np.random.seed(x) establece la semilla aleatoria del generador de números aleatorios, para que las funciones random que van después siempre cogerán los mismos valores “aleatorios”</p> <p>Crear arrays con valores aleatorios</p> <p>array = np.random.randint(inicio, final, forma_matriz) crea un array de números aleatorios entre dos valores; forma_matriz: (z,y,x)</p> <p>z: número de arrays</p> <p>y: número de filas</p> <p>x: número de columnas</p> <p>array = np.random.randint(inicio, final) devuelve un número aleatorio en el rango</p> <p>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</p> <p>array = np.random.random_sample((z,y,x)) crea un array de floats aleatorias con la forma que le especificemos; por defecto genera números aleatorios entre 0-0.9999999...</p> <p>array = np.random.z,y,x=None) devuelve un número aleatorio en 0 y 0.999999999999...</p> <p>np.round(np.random.rand(z,y,x), n) crear array con floats de n decimales</p> <p>np.random.uniform(n,m, size = (z,y,x)) genera muestras aleatorias de una distribución uniforme en el intervalo entre n y m</p> <p>np.random.binomial(n,m, size = (z,y,x)) genera muestras con una distribución binomial; n es el numero total de pruebas; m es la probabilidad de éxito</p> <p>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</p> <p>np.random.permutation(array) devuelve un array con los mismos valores mezclados aleatoriamente</p>

Python Cheat Sheet 5
Matplotlib
Gráficas
<pre>import matplotlib.pyplot as plt</pre>
<pre>plt.figure(figsize = (n,m))</pre> inicia una grafica dibujando el marco de la figura; n es la anchura y m es la altura, en pulgadas <pre>plt.show()</pre> muestra la figura
Gráficas básicas
Bar plot <pre>plt.bar(df[“columna1”], df[“columna2”])</pre> crea un diagrama de barras donde los ejes son: columna1 – x, columna2 – y
Horizontal bar plot <pre>plt.barh(df[“columna1”], df[“columna2”])</pre> crea una diagramma de barras horizontales donde los ejes son: columna1 – x, columna2 – y
Stacked bar plot <pre>plt.bar(x, y, label = ‘etiqueta’)</pre> <pre>plt.bar(x2, y2, bottom = y, label = ‘etiqueta2’)</pre> crea una diagrama de barras apiladas para visualizar dos variables juntas; y indica la barra de referencia
Scatter plot <pre>plt.scatter(df[“columna1”], df[“columna2”])</pre> crea una gráfica de dispersión donde los ejes son: columna1 – x, columna2 – y
Gráficas estadísticas
Histogram <pre>plt.hist(x = df[‘columna1’], bins = n)</pre> crea una histograma que muestra la frecuencias de una distribución de datos; donde x es la variable de interés y n es el número de barras
Box Plot <pre>plt.boxplot(x = df[‘columna1’])</pre> crea un diagrama de cajas para estudiar las características de una variable numerica; x es la variable de interés

Pie Chart <pre>plt.pie(x, labels = categorias, radius = n)</pre> crea un gráfico de sectores donde x es la variable de interés (debe esta agrupado por categorias); n es el tamaño
Violin Plot <pre>plt.violinplot(x, showmedians = True, showmeans = True)</pre> crea un diagrama de violin donde x es la variable de interés y muestra la mediana y la media

Personalización

`color = "color"` establece el color de la grafica
`facecolor = "color"` establece el color del relleno
`edgecolor = "color"` establece el color de los bordes

Colores en Scatter Plots:

`c= df['columna'].map(diccionario)`
`diccionario = {"valor1": "color1", "valor1": "color1"}`
lista de colores
`plt.xlabel("etiqueta_eje_x")` asignar nombre al eje x
`plt.ylabel("etiqueta_eje_y")` asignar nombre al eje y
`plt.legend(labels = ['label1', 'label2', etc])` muestra la leyenda cuando mostramos la figura
`plt.title(label = "titulo")` asignar un titulo a la gráfica
`plt.xlim([n,m])` establece el rango del eje x; donde n es el mínimo y m es el máximo
`plt.ylim([n,m])` establece el rango del eje y; donde n es el mínimo y m es el máximo
`plt.grid()` crea una cuadrícula al fondo de la figura; coge los parámetros:
`color = "color"`
`linestyle = "solid" | "dashed" | "dashdot" | "dotted"`
`linewidth = n` establece la anchura de la linea
`marker = 'tipo'` establece el tipo de marcador; se usa con `plt.scatter` y `plt.plot`

"."	Punto	"P"	Más (relleno)
"."	Pixel	"*"	Estrella
"o"	Círculo	"h"	Hexágono 1
"v"	Triángulo abajo	"H"	Hexágono 2
"^"	Triángulo arriba	"+"	Más
"<"	Triángulo izquierda	"x"	x
">"	Triángulo derecha	"X"	x (relleno)
"8"	Octágono	"D"	Diamante
"s"	Cuadrado	"d"	Diamante fino
"p"	Pentágono		

Multigráficas

`fig, ax = plt.subplots(numero_filas, numero_columnas)` crear una figura con multiples graficas; fig es la figura y ax es un array con subplots como elementos
Se usan los indices para establecer como es cada grafica:
`ax[indice].tipo_grafica(detalles de la grafica)`
`ax[indice].set_title('titulo')`
`ax[indice].set_xlabel('xlabel')`
`ax[indice].set_ylabel('ylabel')`
`ax[indice].set_xlim(min, max)`
`ax[indice].set_ylim(min, max)`
`ax[indice].set_xticklabels(labels = df['column'], rotation = n)` para cambiar la rotation de los labels

Exportar figuras

`plt.savefig('nombre_de_la_figura.extension')`

Seaborn

Line plot

`fig = sns.lineplot(x = 'columna1', y = 'columna2', data = df, ci = None)` crea una gráfica lineal donde los ejes son: columna1 – x, columna2 – y
`ci = None` para que no muestra el intervalo de confianza de los datos
`hue = columna` opcional; muestra lineas en diferentes colores por categorias segun una variable

Scatter plot

`fig = sns.scatterplot(x = 'columna1', y = 'columna2', data = df, hue = 'columna')` crea una gráfica de dispersión

Seaborn
Count plot <pre>fig = sns.countplot(x = ‘columna1’, data = df, hue = ‘columna’)</pre> crea una gráfica de barras con la cuenta de una variable categórica; se puede especificar solo una variable en la eje x o y, mas una variable opcional con hue
Histogram <pre>fig = sns.histplot(x = ‘columna1’, data = df, hue = ‘columna3’, kde = True, bins = n)</pre> crea una histograma que muestra la frecuencias de una distribución de datos; donde x es la variable de interés y n es el número de barras <pre>kde = True</pre> muestra una curva de la distribucion
Box Plot <pre>fig = sns.boxplot(x = ‘columna1’, data = df, hue = ‘columna’)</pre> crea un diagrama de cajas; x es la variable de interés; por defecto se muestra con orientación horizontal – usar eje y para orientación vertical
Catplot <pre>fig = sns.catplot(x = ‘columna1’, y = ‘columna2’, data = df, hue = ‘columna’, kind = ‘tipo’)</pre> crea una gráfica que muestra la relacion entre una variable categorica y una variable numerica kind = ‘box’ ‘bar’ ‘violin’ ‘boxen’ ‘point’ por defecto es strip plot
Pairplot <pre>fig = sns.pairplot(data = df, hue = ‘columna’, kind = ‘tipo’)</pre> crea los histogramas y diagramas de dispersión de todas las variables numéricas de las que disponga el dataset con el que estemos trabajando; hue es opcional kind = ‘scatter’ ‘kde’ ‘hist’ ‘reg’ ‘point’ por defecto es scatter
Personalización <pre>fig.set(xlabel = ‘etiqueta_eje_x’, ylabel = ‘etiqueta_eje_y’)</pre> asignar nombre a los ejes <pre>fig.set_title(‘titulo’)</pre> figsieasignar un titulo a la gráfica <pre>plt.legend(bbox_to_anchor = (1, 1))</pre> coloca la leyenda en relación con los ejes
Estadística
Estadísticos generales mínimo el valor mas bajo dentro de un conjunto de datos máximo el valor mas alto dentro de un conjunto de datos
Medidas de tendencia central
Media el valor obtenido al sumar todos los datos y dividir el resultado entre el número total de elementos <pre>df[‘columna’].mean()</pre>
Media ponderada el valor obtenido al multiplicar cada uno de los datos por un peso antes de sumarlos (suma ponderada); después se divide la suma ponderada entre la suma de los pesos <pre>media_ponderada = np.average(df[‘columna’], weights = w)</pre>
Mediana el valor de la variable de posición central en un conjunto de datos <pre>df[‘columna’].median()</pre>
Moda el valor que tiene mayor frecuencia absoluta de entre todos los datos <pre>df[‘columna’].mode()</pre>

Estadística
Medidas de dispersión
Desviación respecto a la media la diferencia en valor absoluto entre cada valor de los datos y su media aritmética <pre>diferencias = df[‘columna’] - df[‘columna’].mean()</pre> <pre>desviación_media = np.abs(diferencias)</pre>
Varianza una medida de dispersión que representa la variabilidad de una serie de datos respecto a su media <pre>df[‘columna’].var()</pre>
Desviación estándar o desviación típica la raíz cuadrada de la varianza; nos dice como de dispersos están los valores de los datos en un conjunto de datos; cuanto mayor sea, mayor será la dispersión o variabilidad en nuestros datos <pre>df[‘columna’].std()</pre>
Robustez - normalmente, cuanto más cantidad de datos hayamos usado para calcular los estadísticos, más robustos serán estos, ya que la influencia de unos pocos datos inusuales sobre el total de los mismos será menor - punto de ruptura: la fracción de los datos a los que podríamos dar valores arbitrarios sin hacer que el estadístico se vea tan afectado como para no ser útil 1/n donde n es el numero de registros
Coefficiente de variación muestra la relación entre el tamaño de la media y la variabilidad de la variable; se calcula como el cociente entre la desviación típica y la media; cuanto mayor sea, mayor será la dispersión o variabilidad en nuestros datos <pre>df[‘columna’].std() / df[‘columna’].mean()</pre>
Percentiles divide una serie de datos ordenados de menor a mayor en cien partes iguales; se trata de un indicador que busca mostrar la proporción de la serie de datos que queda por debajo de su valor <pre>percentil_n = np.percentile(df[‘columna’], n)</pre> saca el valor en el percentil n percentil 25 → Q1 percentil 50 → Q2 (la mediana) percentil 75 → Q3
Rangos intercuartílicos una medida de dispersión estadística igual a la diferencia entre los cuartiles 75 y 25 <pre>q3, q1 = np.percentile(df[“columna”], [75, 25])</pre> saca los tercer y primer cuartiles <pre>rango_intercuartilico = q3 - q1</pre>
Tablas de frecuencias
Frecuencias absolutas son el número de veces que se repite un número en un conjunto de datos <pre>df_group = df.groupby(‘columna’).count().reset_index()</pre>
Frecuencias relativas las veces que se repite un número o categoría en un conjunto de datos respecto al total, en porcentajes <pre>df_group_sin_str = df_group.drop(‘columna_str’, axis=1)</pre> <pre>frecuencia_relativa = df_group_sin_str / df.shape[0] * 100</pre> <pre>columnas = df_group_sin_strings.columns</pre> <pre>df_group[columnas] = frecuencia_relativa</pre>

Estadística
Tablas de contingencia - una tabla de frecuencias que cuenta todas las combinaciones posibles de cada pareja de valores de las columnas que estamos intentando comparar - deberá contener al menos dos filas y dos columnas para representar datos categóricos - permite medir la interacción entre las variables <pre>df_crosstab = pd.crosstab(df[‘columna1’], df[‘columna2’], normalize = True, margins = True)</pre> <pre>normalize</pre> muestra los valores en porcentajes (por uno) <pre>margins</pre> muestra los totales y subtotales
Coefficiente de correlación de Pearson - nos permite conocer la intensidad y dirección de la relación entre las dos variables - coeficiente > 0: correlación positiva (según aumente el valor de una variable aumenta el valor de la otra) - coeficiente < 0: correlación negativa (según aumente el valor de una variable disminuye el valor de la otra) - coeficiente = 1 o -1: correlación total (positiva o negativa) - coeficiente = 0: no existe relación lineal entre las dos variables, pero puede haber algún otro tipo de relación que no se capture bien con este coeficiente <pre>df[‘columna1’].corr(df[‘columna2’])</pre> calcula la correlacion entre dos variables <pre>matriz_correlacion = df.corr()</pre> crea una matriz mostrando las correlaciones entre todos los variables
Heatmap <pre>sns.heatmap(df.corr(), cmap = ‘color_palette’, annot = True, vmin = -1, vmax = 1)</pre> crea un heatmap con una escala de colores que refleja los valores de correlacion <pre>annot = True</pre> para que aparezcan los valores <pre>vmin/vmax</pre> establecen la escala de color
<pre>sns.heatmap(df.corr())[[‘column1’, ‘column2’]], cmap = ‘color_palette’, annot = True, vmin = -1, vmax = 1)</pre>
Sesgos (skewness) - una 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 <pre>sns.displot(df[‘columna’], kde = True)</pre> crea un histograma que muestra la distribution de los valores <pre>import scipy.stats</pre> <pre>skew(df[‘columna’])</pre> muestra el valor del sesgo de una variable
Intervalos de confianza describe la variabilidad entre la medida obtenida en un estudio y la medida real de la población (el valor real) Confidence Interval = x +/- t*(s/√n) where: x: sample mean t: t-value that corresponds to the confidence level s: sample standard deviation n: sample size <pre>import scipy.stats as st</pre> <pre>st.t.interval(alpha = n, df = len(df[‘columna’])-1, loc = np.mean(df[‘columna’]), scale = st.sem(df[‘columna’]))</pre> devuelve el rango de valores para lo cual hay un n% de 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