Python Cheat Sheet 1

Variables ampliadas por text (CONCATENATION)

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)

Operaciones Algebraicas

```
+ sumar
- restar
* multiplicar
** elevar
```

```
/ dividir
// divider y redondear (modulus)
% resto de una division (floor 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)
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

Listas [ ] Metodos no permanentes

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

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

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, 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)

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

Diccionarios { key : value , }

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 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

Tuplas (,) inmutables, indexados

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

`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

zip()

`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

Sets {}

no permiten duplicados, no tienen orden

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

input()

- permite obtener texto escrito por teclado del usuario

`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

Sentencias de control

`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")
```

For loops

- 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 loop
- 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

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.

```
try:
    print("2.split())
except:
    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 stop)
- tambien se puede especificar saltos

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

Python Cheat Sheet 2

Funciones

Definir una funcion:
```
def nombre_funcion(parametro1, parametro2, ...):
    return valor_del_return
```

Llamar una funcion:
```
nombre_funcion(argumento1, argumento2, ...)
```

`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

```
def nombre_funcion(parametros, *args, **kwargs,
                   parametro_por_defecto = valor)
    arg/kwarg: sin */** dentro de la funcion
    arg[0]
```

Llamar una funcion con *args:
```
nombre_funcion(argumento, argumento, argumento, ...)
```
o
```
nombre_funcion(*[lista_o_tupla_de_args])
```

Llamar una funcion con **kwargs:
```
nombre_funcion(**diccionario)
```

Clases

Definir una clase:
```
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}"
```

Definir una clase hija:
```
class NombreClaseHija(NombreClaseMadre):
    def __init__(self, atributo1, atributo2):
    super().__init__(atributo_heredado1, ...)

    def nombre_funcion_hija (self, parametros):
```

Crear un objeto de la clase:

`variable_objeto = NombreClase(valor_atributo1, valor_atributo2)` instanciar (crear) un objeto
`variable_objeto.atributo` devuelve el valor del atributo guardado para ese objeto
`variable_objeto.atributo = nuevo_valor` para cambiar el valor del atributo
`variable_objeto.nombre_funcion()` llamar una funcion

`print(help(NombreClase)` imprime informacion sobre la clase

Regex

- una abreviatura de `expresión regular`, `regex` es una cadena de texto que permite crear patrones que ayudan a emparejar, localizar y gestionar strings

`import re` 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 número
`\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

`re.findall("patron", string)` busca en todo el string y devuelve una lista con todas las coincidencias en nuestro string

`re.search("patron", string_original)` busca en todo el string y devuelve un objeto con la primera coincidencia en nuestro string

`re.match("patron", "string_original)` busca en la primera linea del string y devuelve un objeto con la primera coincidencia en nuestro string

`resultado_match.span()` devuelve la referencia de las posiciones donde hizo el "match"

`resultado_match.group()` devuelve el element resultando de la coincidencia del "match"

`re.split("patron", "string_original")` busca en todo el string y devuelve una lista con los elementos separados por el patron

`re.sub("patron", "string_nuevo", "string_original")` 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

`import modulo` para importar un modulo
`from modulo import funcion` importar solo una funcion
`modulo.funcion()` usar una funcion de un modulo
`modulo.clase.funcion()` para usar una funcion de una clase
`import modulo as md` asignar un alias a un modulo

Libreria os

`os.getcwd()` devuelve la ruta de donde estamos trabajando; se puede guardar en un variable e.g. ruta = os.getcwd()
`os.listdir()` devuelve una lista de los archivos y carpetas donde estamos trabajando
`os.listdir('carpeta')` devuelve los contenidos de otra carpeta
`os.chdir('ruta')` cambia la carpeta en la que estes
`os.mkdir('nueva_carpeta')` crear una nueva carpeta
`os.rename('nombre_carpeta', 'nueva_nombre')` cambia el nombre de una carpeta
`os.rmdir('carpeta')` borra la carpeta

Libreria shutil

from shutil inmport rmtree
`rmtree('carpeta')` 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"
`ubicacion_archivo = ubicacion_carpeta + "/" + nombre_archivo`

`f = open(ubicacion_archivo)` abrir un archivo en variable f
`f.close()` cerrar un archivo `* IMPORTANTE *`
```
with open(ubicacion_archivo) as f:
    codigo e.g. variable = f.read()
```
abre el archivo solo para ejecutar el codigo indicado (y despues lo deja)

Encoding

`from locale import getpreferredencoding`
`getpreferredencoding()` para saber que sistema de encoding estamos usando
`f = open(ubicacion_archivo, encoding="utf-8")` abrir un archivo y leerlo con el encoding usado; guardar con .read()

mode: argumento opcional al abrir un archivo

`r` – read
`w` – write - sobreescribe
`x` – exclusive creation, sólo crearlo si no existe todavía
`a` – appending, añadir texto al archivo sin manipular el texto que ya había
hay que anadir otra letra:
`t` – texto – leer en texto
`b` – bytes – leer en bytes (no se puede usar con encoding)

`f = open(ubicacion_archivo, mode = "rt")`

Leer ficheros

`f.read()` leer el contenido de un archivo
`f.read(n)` leer los primeros n caracteres de un archivo
`variable = f.read()` guardar el contenido del archivo (o n caracteres de un archivo) en un variable
`f.readline(n)` por defecto devuelve la primera linea o n lineas
`f.readlines()` 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

```
with open(ubicacion_archivo, "w") as f:
    f.write("Texto que va en el fichero.")
```
para escribir
```
with open(ubicacion_archivo, "a") as f:
    f.write("Texto que va en el fichero.")
```
para anadir texto
`f.writelines('lista')` para anadir lineas de texto de una lista

Ficheros xml

`import xml.etree.ElementTree as ET` importa la librería xml
`variable_tree = ET.parse('ruta/archivo.xml')` abre el archivo
`variable_root = variable_tree.getroot()` 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>
```

`variable_root.tag` devuelve el nombre del tag del raiz
`variable_root.attrib` devuelve los atributos del fichero

`variable_root.find("tag").find("childtag").text` devuelve la primera ocasión en que el tag de un elemento coincida con el string
`variable_root.findall("tag").findall("childtag").text` devuelve todos los elementos cuyos tag coincide

MySQL Connector/Python

Conectar a una base de datos

`import mysql.connector` para importar MySQL Connector

pip install mysql-connector
pip install mysql-connector-Python

`connect()` para conectar a una base de datos:

```
variable_cnx = mysql.connector.connect(user='root',
                                       password='AlumnaAdalab',
                                       host='127.0.0.1',
                                       database='nombre_BBDD')
```

`from mysql.connector import errorcode` importar errores
`mysql.connector.Error` se puede usar en un try/except
`cnx.close()` desconectar de la base de datos

Realizar queries

`variable_cursor = cnx.cursor()` crear el objeto cursor que nos permite comunicar con la base de datos
`variable_cursor.close()` desconectar el cursor
`variable_query = ("SQL Query")` guardar un query en un variable
`variable_cursor.execute(variable_query)` ejecutar el query; devuelve una lista de tuplas
`import datetime` sacar fechas en el formato AAAA-MM-DD
`datetime.date(AAAA, M, D)` devuelve el formato de fecha
`variable_query = "SQL Query... %s AND %s")` query dinamica
`variable_cursor.execute(query, (variable1, variable2))` valores que van en lugar de los %s
`variable_cursor.execute("SHOW DATABASES")` mostrar las BBDD
`variable_cursor.execute("SHOW TABLES")` mostrar las tablas de la BBDD indicado en la conexión
`variable_cursor.execute("SHOW TABLES")`
`variable_cursor.execute("SHOW COLUMNS FROM bbdd.table")` mostrar las columnas de la tabla especificada; hay que conectarse a la bbdd information_schema

Argumentos cursor:

`variable_cursor = cnx.cursor([arg=value[, arg=value]...])`
`buffered=True` devuelve todas las filas de la bbdd
`raw=True` el cursor no realizará las conversiones automáticas entre tipos de datos
`dictionary=True` devuelve las filas como diccionarios
`named_tuple=True` devuelve las filas como named tuples
`cursor_class` un argumento que se puede usar para indicar que subclase queremos usar para instanciar el nuevo cursor

Obtener resultados de una query

`variable_cursor.fetchone()` devuelve el primer resultado
`variable_cursor.fetchall()` devuelve todos los resultados como iterable – cada fila es una tupla

Pandas dataframe with SQL

`import pandas as pd`
`variable_df = pd.DataFrame(variable_resultado_fetchall, columns = ['columna1', 'columna2', ...])` crear un dataframe con los resultados de una query en una variable
`variable_df.head(n)` devuelve las n primeras filas del df, o 5 por defecto
`variable_df = pd.read_sql_query(variable_query, variable_cnx)` convertir los resultados de la query en df
`pd.read_sql(variable_query, variable_cnx)`
`variable_df.to_csv("nombre_archivo.csv")` guardar en csv
`variable_df.to_string()` formatear el dato en string
`variable_df.to_latex()` formatear el dato en un string que facilite la inserción en un documento latex

Crear y alterar una base de datos

`variable_cursor.execute("CREATE DATABASE nombre_BBDD")`
`variable_cursor.execute("CREATE TABLE nombre_tabla (nombre_columna TIPO, nombre_columna2 TIPO2)")`
`variable_cursor.execute("ALTER TABLE nombre_tabla ALTERACIONES")`

Insertar datos

`variable_query = "INSERT INTO nombre_tabla (columna1, columna2) VALUES (%s, %s)"`
`variable_valores = (valor1, valor2)`
`variable_cursor.execute(variable_query, variable_valores)`

otro método:

`variable_query = "UPDATE nombre_tabla SET nombre_columna = "nuevo_valor" WHERE nombre_columna = "valor"`

Insertar múltiples filas a una tabla

```
variable_valores_en_tuplas = ((valor1columna1, valor1columna2), (valor2columna1, valor2columna2))
variable_cursor.executemany(variable_query, variable_valores_en_tuplas)
```

`variable_conexion.commit()` después de ejecutar la inserción, para que los cambios efectúen en la BBDD
`variable_conexion.rollback()` se puede usar después de execute y antes de commit para deshacer los cambios
`print(variable_cursor.rowcount, "mensaje")` imprimir el número de filas en las cuales se han tomado la accion

Eliminar registros

`variable_query = "DROP TABLE nombre_tabla"`

Añadir errores

importar errorcode y usar try/except:

```
try:
    accion
except mysql.connector.Error as err:
    print(err)
    print("Error Code:", err.errno)
    print("SQLSTATE", err.sqlstate)
    print("Message", err.msg)
```

Python Cheat Sheet 3

Pandas

Series: estructuras en una dimension

Crear series

`serie = pd.Series()` crear serie vacía
`serie = pd.Series(array)` crear serie a partir de un array con el indice por defecto
`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 partir de un escalar con la longitude igual al número de indices
`serie = pd.Series(diccionario)` crear una serie a partir de un diccionario

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.dtypes` devuelve el tipo de dato

`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

`serie[“etiqueta”]` devuelve el valor de los elementos 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 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
`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 serie2 y devuelve True o False

Filtrado booleanos

`serie < > >= <= == valor` devuelve True o False segun si cada condición cumple la condición
`serie1[serie1 < > >= <= == valor]` devuelve solo los valores que cumplen la condición
`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)

DataFrames

Crear DataFrames

`df = pd.DataFrame(data, index, columns)`
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

`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

Acceder a informacion de un DataFrame

`df.loc[“etiqueta_fila”, “etiqueta_columna”]` devuelve el contenido de un campo en una columna de una fila
`df.loc[“etiqueta_fila”,:]` devuelve los valores de todas las columnas de una fila
`df.loc[:,“etiqueta_columna”]` devuelve los valores de todas las filas de una columna
`df.iloc[indice_fila, indice_columna]` devuelve el contenido de un campo en una columna de una fila
`df.iloc[indice_fila, :]` devuelve los valores de todas las columnas de una fila
`df.iloc[:,indice_columna]` devuelve el contenido de un campo en una columna de una fila
`df.loc[[lista_etiquetas_filas], [lista_etiquetas_columnas]]` devuelve el contenido de varias filas / varias columnas
`df.loc[[lista_indices_filas], [lista_indices_columnas]]` 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
`df.loc[df.etiqueta > x]` seleccionar datos basado en una condición usando operadores comparativos
`df.loc[(df.etiqueta > x) & (df.etiqueta == y)]` seleccionar datos que tienen que cumplir las dos condiciónes (and)
`df.loc[(df.etiqueta > x) | (df.etiqueta == y)]` seleccionar datos que tienen que deben cumplir una de las dos condiciones (or)
`df.iloc[list(df.etiqueta > x), :]` iloc no acepta una Serie booleana; hay que convertirla en lista
`variable_df.head(n)` devuelve las n primeras filas del df, o 5 por defecto

Crear columnas

`df[“nueva_columna”] = (df[“etiqueta_columna”] + x)` crea una nueva columna basada en otra
`df = df.assign(nueva_columna= df[“etiqueta_columna] + x)` crea una nueva basada en otra
`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*
`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)

Eliminar columnas

`df = df.drop(columns = [“column1”, “column2”])` eliminar columnas

DataFrames: carga de datos

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

`df = pd.read_excel(“ruta/nombre_archivo.xlsx”)` crear un dataframe de un archivo de Excel
- si sale “ImportError:... openpyxl...”, en el terminal:
pip3 install openpyxl o **pip install openpyxl**

`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.

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

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 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

```
with pd.ExcelWriter(“ruta/archivo.ext”) as writer:
    df.to_Excel(writer, nombre_hoja = ‘nombre’)
```

guardar un dataframe en una hoja de Excel

Librería PyDataset

`pip install pydataset` o `pip3 install pydataset`
`from pydataset import data`
`data()` para ver los datasets listados en un dataframe por su id y título
`df = data(‘nombre_dataset’)` guardar un dataset en un dataframe

Metodos de DataFrames

Metodos para explorar un dataframe

`df.head(n)` devuelve las primeras n lineas del dataframe, o por defecto 5
`df.tail(n)` devuelve las últimas n lineas del dataframe, o por defecto 5
`df.sample(n)` devuelve n filas aleatorias de nuestro dataframe, o uno por defecto
`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
`df.describe` devuelve un dataframe con un resumen de los principales estadísticos (media, mediana, desviación estándar etc.) de las columnas numéricas
`df.describe(include = object)` devuelve un dataframe con un resumen de los principales estadísticosde las columnas con variables tipo string
`df.info()` devuelve un resumen sobre el no. de columnas, nombres de columnas, numero de valores no nulos y los tipos de datos de las columnas
`df[“nombre_columna”].unique()` o `df.nombre_columna.unique()` devuelve un array con los valores únicos de la columna
`df[“nombre_columna”].value_counts()` o `df.nombre_columna.value_counts()` devuelve una serie con el recuento de valores únicos en orden descendente
`df.isnull()` o `df.isna()` devuelve True o False según si cada valor es nulo o no
`df.isnull().sum()` o `df.isna().sum()` devuelve el número de valores nulos por columnas
`df.corr()` devuelve la correlación por pares de columnas, excluyendo valores NA/nulos
`df.set_index([“nombre_columna”], inplace = True)` establece el índice utilizando uno o mas columnas; puede sustituir o ampliar un índice existente
`inplace = True` los cambios sobreescriben sobre el df
*** cuando una columna se cambia a índice ya no es columna ***

Realizar cambios en el dataframe:

`df.reset_index(inplace = True)` quitar una columna como indice para que vuelva a ser columna; crea un dataframe de una serie
`df.rename(columns = {“nombre_columna”: “nombre_nueva”}, inplace = True)` cambia los nombres de una o mas columnas

ejemplo de dict comprehension para crear diccionario sobre las columnas existentes de un dataframe:

`diccionario = {col : col.upper() for col in df.columns}`

`df.rename(columns = diccionario, inplace = True)` cambia los nombres de las columnas según el diccionario
`df.drop([“columna1”, “columna2”], axis = b)` eliminar una o mas columnas o filas segun lo que especificamos
`axis = 1` columnas
`axis = 0` filas
`df.rename(columns = diccionario, inplace = True)` cambia los nombres de las columnas según el diccionario
`df[“columna_nueva”] = pd.cut(x=df[“nombre_columna”], bins=[n,m,l..])` 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
`df.replace(to_replace = valor, value = valor_nuevo, inplace = True)` reemplaza cierto valor por otro que especificamos
`df[“nombre_columna”].replace(to_replace = valor, value = valor_nuevo, inplace = True)` reemplaza cierto valor en una columna por otro que especificamos
`df[“nombre_columna”] = df[“nombre_columna”] + x` reemplaza los valores de la columna por el valor + x (o otro valor que indicamos)

Filtrados de datos

`pd.options.display.max_columns = None` ejecutar antes del df.head() para poder ver todas las columnas

Filtrado por una columna con operadores de comparación

`variable_filtro = df[df[“nombre_columna”] == valor]` 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

& and
| or
~ not

`variable_filtro = df[(df[“columna1”] == valor) & (df[“columna2”] == valor) & (df[“columna3”] > n valor)]` extrae las filas donde los valores de las columnas cumplan las condiciónes en parentesis

`variable_filtro = df[(df[“columna1”] == valor) | (df[“columna1”] == valor)` extrae las filas donde los valores de las columnas cumplan con una condición u otra

`variable_filtro = ~(df[df[“columna1”] == valor])` extrae las filas donde los valores de las columnas NO cumplan con la condición

Metodos de pandas de filtrar

`variable_filtro = df[df[“nombre_columna”].isin(iterable)]` extrae las filas cuyas valores de la columna nombrada están en el iterable (una lista, serie, dataframe o diccionario)

`variable_filtro = df[df[“nombre_columna”].str.contains (patron, regex = True)]` extrae las filas cuyas valores de la columna nombrada contenienen el patron de regex

`variable_filtro = 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

`variable_filtro = 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

`df[pd.notnull(df[“nombre_columna”])]` devuelve las filas que no tiene valores nulos en la columna especificada

Reemplazar valores basados en indices y condiciones:

```
indices_filtrados = df.index[df[“columna”] == “valor”]
for indice in indices_filtrados:
    df[“nombre_columna”].iloc[indice] = “valor_nuevo”
```

Reemplazar valores basados en metodos NumPy:

`df[“nueva_columna”] = np.where(df[“nombre_columna”] > n, “categoria_if_true”, “categoria_if_false”)` crea una nueva columna con los valores basados en una condición

`df[“nueva_columna”] = np.select(lista_de_condiciones, lista_de_opciones)` crea una nueva columna con los valores basados en multiples condiciones

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 aparecera 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 aparecera en columnas duplicadas

Group By

`df_groupby = df.groupby("columna_categoría")` 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 categoria indicada como grupo1)

Cálculos con groupby:
`df_nuevo = df.groupby("columna_categoría").mean()` devuelve un dataframe con la media de todas las columnas de valores numéricos, por categoría
`df_nuevo = df.groupby("columna_categoría")["columna1"].mean()` devuelve un dataframe con la media de la columna especificada

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

`df_nuevo = df.groupby("columna_categoría", 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

- 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)` crea una columna nueva con los valores de otra columna transformados según la función indicada

apply() con datetime
`import datetime`
`df['columna_fecha'] = df['columna_fecha'].apply(pd.to_datetime)` cambia una columna de datos tipo fecha en el formato datetime
```
def sacar_año(x):
    return x.strftime("%Y")
```
`df['columna_año'] = (df['columna_fecha'].apply(sacar_año)`
crea una columna nueva del año solo usando un método de la librería datetime; ("%B") para meses

apply() con funciones de mas de un parametro
`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)

NumPy (Numerical Python)

Crear arrays

Crear arrays de listas
`array = np.array(lista, dtype= tipo)` crea un array unidimensional de una lista
`array = np.array([lista1, lista2])` crea un array bidimensional de dos listas
`array = np.array([listadelistas1, listadelistas2])` crea un array bidimensional de dos listas

Crear otros tipos de arrays
`array = np.arange(valor_inicio, valor_final, saltos)` crea un array usando el formato [start:stop:step]
`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
`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
`array = np.empty((z,y,x), tipo)` crea un array vacio con datos por defecto tipo float
`array2 = np.empty_like(array1)` crea un array vacia con la 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

Guardar y salvar arrays en .txt
`np.savetxt('ruta/nombre_fichero.txt', array)` guardar un array de uno o dos dimensiones como .txt
`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

Indices, Subsets, Metodos de Arrays

Indices de arrays
`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
`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][j] = n` cambiar el valor del elemento en esta posicion al valor n

Subsets
`array > n` devuelve la forma del array con True o False según si el elemento cumple con la condición o no
`array[array > n]` devuelve un subset: todos los valores que cumplen la condición en una lista dentro de un array
`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"

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 de columnas al número de arrays y viceversa; el número de filas no cambia
`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
`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.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
`np.sort(-array)` devuelve un array con los valores de cada fila ordenados en orden descendente
`np.round(array, decimals = x)` devuelve un array con los valores del array redondeados a x decimales
`np.round(array, decimals = x)` devuelve un array con los valores del array redondeados a x decimales
`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
`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

Operaciones estadísticas y matemáticas

Operaciones estadísticas y matemáticas

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.
Por ejemplo:
`np.sum(array, axis = 0)` devuelve un array con la suma de cada fila

El parametro axis en arrays multidimensionales:
`axis = 0` dimensión
`axis = 1` columnas
`axis = 2` filas
- si especificamos el axis, la operación devuelve el resultado por cada dimensión, fila o columna.
Por ejemplo:
`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 cada matriz

Operaciones con parámetro del axis:
`np.sum(array_3D)` 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 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 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

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
`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 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
`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

Funciones de conjuntos

`np.unique(array)` devuelve un array con los valores únicos del array ordenados
`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 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
`np.unique(array, axis = b)` devuelve un array con los valores únicos ordenados de las filas o columnas

Funciones para arrays unidimensionales
`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)` 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
`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 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

NumPy Random

`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"

Crear arrays con valores aleatorios
`array = np.random.randint(inicio, fina1, forma_matriz)` 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 especifiquemos; por defecto genera números aleatorios entre 0-1
`array = np.random.random_sample((z,y,x))` crea un array de floats aleatorias con la forma que le especifiquemos; 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 de n decimales
`np.random.uniform(n,m, size = (z,y,x))` genera muestras aleatorias de una distribución uniforme en el intervalo entre n y m
`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
`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

Python Cheat Sheet 5

Matplotlib

Gráficas

`plt.figure(figsize = (n,m))` inicia una grafica dibujando el marco de la figura; n es la anchura y m es la altura, en pulgadas
`plt.show()` muestra la figura

Gráficas básicas

Bar plot
`plt.bar(df["columna1"], df["columna2"])` crea un diagrama de barras donde los ejes son: columna1 – x, columna2 – y

Horizontal bar plot
`plt.barh(df["columna1"], df["columna2"])` crea una diagramma de barras horizontales donde los ejes son: columna1 – x, columna2 – y

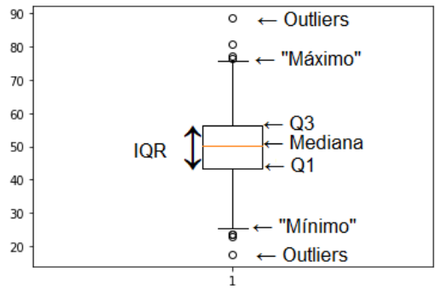
Stacked bar plot
`plt.bar(x, y, label = 'etiqueta')`
`plt.bar(x2, y2, bottom = y, label = 'etiqueta2')`
crea una diagrama de barras apiladas para visualizar dos variables juntas; y indica la barra de referencia

Scatter plot
`plt.scatter(df["columna1"], df["columna2"])` crea una gráfica de dispersión donde los ejes son: columna1 – x, columna2 – y

Gráficas estadísticas

Histogram
`plt.hist(x = df['columna1'], bins = n)` 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
`plt.boxplot(x = df['columna1'])` crea un diagrama de cajas para estudiar las caracteristicas de una variable numerica; x es la variable de interés



Pie Chart
`plt.pie(x, labels = categorias, radius = n)` crea un gráfico de sectores donde x es la variable de interés (debe esta agrupado por categorias); n es el tamaño

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

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" Cirulo	"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)`

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
parametros:
`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

Count plot
`fig = sns.countplot(x = 'columna1', data = df, hue = 'columna')` 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

Catplot
`fig = sns.catplot(x = 'columna1', y = 'columna2', data = df, hue = 'columna')` crea una gráfica que muestra la relacion entre una variable categorica y una variable numerica

Histogram
`fig = sns.histplot(x = 'columna1', data = df, hue = 'columna3', kde = True, bins = n)` 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
`fig = sns.boxplot(x = 'columna1', data = df, hue = 'columna')` 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
`fig = sns.catplot(x = 'columna1', y = 'columna2', data = df, hue = 'columna', kind = 'tipo')` crea una gráfica que muestra la relacion entre una variable categorica y una variable numerica
kind = 'box' | 'bar' | 'violín' | 'boxen' | 'point'
por defecto es strip plot

Pairplot
`fig = sns.pairplot(data = df, hue = 'columna', kind = 'tipo')` 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

`fig.set(xlabel = 'etiqueta_eje_x', ylabel = 'etiqueta_eje_y')` asignar nombre a los ejes
`fig.set_title('titulo')` figsieasignar un titulo a la gráfica
`plt.legend(bbox_to_anchor = (1, 1)` coloca la leyenda en relación con los ejes