

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 | / dividir |
| - restar | // divider y redondear (modulus) |
| * multiplicar | % resto de una division (floor division) |
| ** elevar | 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() 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

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 revés 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 valores solos

Diccionarios – Metodos

Obtener informacion de un diccionario

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)

Ampliar un diccionario

.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

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 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

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 indéntado*
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 valores

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 código
[**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: <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:</code> |
| <code> def __init__(self, atributo1, atributo2): self.atributo1 = atributo1 self.atributo2 = atributo2 self.atributo_por_defecto = 'valor'</code> |
| <code> 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, ...)</code> |
| <code> 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 <code>\w</code> cualquier caracter de tipo alfabético <code>\d</code> cualquier caracter de tipo númeroico <code>\s</code> espacios <code>\n</code> saltos de línea <code>\W</code> cualquier caracter que no sea una letra <code>\D</code> cualquier caracter que no sea un dígitos <code>\S</code> cualquier elemento que no sea un espacio <code>()</code> aísla sólo una parte de nuestro patrón de búsqueda que queremos devolver <code>[]</code> incluye todos los caracteres que queremos que coincidan e incluso incluye rangos como este: a-z y 0-9 <code> </code> es como el operador 'or' <code>\</code> señala una secuencia especial (escapar caracteres especiales) <code>{}</code> Exactamente el número especificado de ocurrencias <code>{n}</code> Exactamente n veces <code>{n,}</code> Al menos n veces <code>{n,m}</code> 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, 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 | | | | |
|--------------------------------------|--|--|--|---|
| Pandas | <div><div>Crear DataFrames</div><div><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</div><div><code>df = pd.DataFrame(array)</code> crear un dataframe a partir de un array con indices y columnas por defecto <code>df = pd.DataFrame(diccionario)</code> crear un dataframe a partir de un diccionario - los keys son los nombres de las columnas</div></div> | <div><div><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.duplicated().sum()</code> devuelve el numero de filas duplicadas</div><div>Eliminar filas duplicadas</div><div><code>df.drop_duplicates(inplace = True, ignore_index=True)</code> elimina filas duplicadas; ignore_index para no tener el indice en cuenta</div></div> | <div>Tipos de datos en Pandas:<ul style="list-style-type: none">- object- int64- float64- datetime, timedelta[ns]- category- bool<code>df.dtypes</code> devuelve el tipo de datos que hay en cada columna <code>df_tipo = df.select_dtypes(include = "tipo")</code> crea un dataframe de las columnas del tipo de datos especificado <code>df['columna'] = df['columna'].astype('tipo', copy = True, errors = 'ignore')</code> convierte una columna en el tipo de dato especificado <code>copy = True</code> devuelve una copia <code>copy = False</code> *cuidado: los cambios en los valores pueden propagarse a otros objetos pandas* <code>errors = ignore</code> omite excepciones; en caso de error devuelve el objeto original <code>errors = raise</code> permite que se generen excepciones</div> <div><code>pd.options.display.max_columns = None</code> ejecutar antes del df.head() para poder ver todas las columnas</div> <div><code>pd.set_option("display.precision", 2)</code></div> | <div>Identificar nulos</div> <div><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 una serie con el número de valores nulos por columnas <code>df_% nulos = ((df.isnull().sum() / df.shape[0] * 100).reset_index())</code> <code>df_% nulos.columns</code> = ['columna', '% nulos'] crea un dataframe de los porcentajes de los valores nulos</div> <div>Eliminar nulos</div> <div><code>df.dropna(inplace = True, axis=b, subset=[lista_de_columnas], how=)</code> quitar nulos <code>how = 'any' 'all'</code> por defecto 'any': si hay algun valor NA, se elimina la fila o columna; all: si todos los valores son NA, se elimina la fila o columna <code>subset</code> una columna o lista de columnas</div> <div>Tipos de nulos</div> <div><code>np.nan</code> significa "not a number"; es un tipo numérico <code>None</code> valores nulos en columnas tipo string <code>NaT</code> valores nulos tipo datetime valores texto: "n/a", "NaN", "nan", "null" strings que normalmente se convierten automaticamente a np.nan 99999 o 00000 integers que se pueden convertir a nulos</div> <div>Reemplazar nulos</div> <div><code>df = pd.read_csv('archivo.csv', na_values = ['n/a'])</code> <code>.fillna(np.nan)</code> reemplaza los strings 'n/a' con np.nan al cargar el dataframe</div> <div><code>df.fillna(df[value=n, axis=b, inplace=True])</code> reemplazar todos los NaN del dataframe con el valor que especifiquemos <code>df['columna'].fillna(df['columna'].median, axis=b, inplace=True)</code> reemplazar los nulos de una columna por la mediana de esa columna <code>value=n</code> por defecto NaN; es el valor por el que queremos reemplazar los valores nulos que puede ser un escalár, diccionario, serie o dataframe <code>axis</code> por defecto 0 (filas) <code>df.replace(valor_nulo, valor_nuevo, inplace=True, regex=False)</code> reemplazar los nulos por el valor nuevo</div> <div>Imputacion de nulos</div> <div><code>from sklearn.impute import SimpleImputer</code> <code>imputer = SimpleImputer(strategy='mean', missing_values = np.nan)</code> inicia la instancia del metodo, especificando que queremos reemplazar los nulos por la media <code>imputer = imputer.fit(df['columna1'])</code> aplicamos el imputer <code>df['media_columna1'] = imputer.transform(df[['price']])</code> rellena los valores nulos según como hemos especificado <code>from sklearn.experimental import enable_iterative_imputer</code> <code>from sklearn.impute import IterativeImputer</code> <code>imputer = IterativeImputer(n_nearest_features=n, imputation_order='ascending')</code> crea la instancia <code>n_nearest_features</code> por defecto None; numero de columnas a utilizar para estimar los valores nulos <code>imputation_order</code> por defecto ascendente; el orden de imputacion <code>imputer.fit(df_numericas)</code> aplicamos el imputer <code>df_datos_trans = pd.DataFrame(imputer.transform(df_numericas), columns = df_numericas.columns)</code> crea un dataframe de los datos transformados; metemos estas columnas en el dataframe original <code>from sklearn.impute import KNNImputer</code> <code>imputerKNN = KNNImputer(n_neighbors=5)</code> crea la instancia <code>imputerKNN.fit(df_numericas)</code> <code>df_knn_imp = pd.DataFrame(imputerKNN.transform(df_numericas), columns = numericas.columns)</code> crea un dataframe de los datos transformados; metemos estas columnas en el dataframe original</div> |
| Series: estructuras en una dimension | <div><div>Crear series</div><div><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 escalár con la longitud igual al número de indices <code>serie = pd.Series(diccionario)</code> crear una serie a partir de un diccionario</div></div> | <div><div>Carga de datos</div><div><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</div><div><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: <code>pip3 install openpyxl</code> o <code>pip install openpyxl</code></div><div><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</div><div><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.</div><div>with open('ruta/nombre_archivo.pkl', 'wb') as f: pickle.dump(df,f) pone los datos de un dataframe en el archivo pkl</div><div><code>pd.read_pickle('ruta/nombre_archivo.csv').head(n)</code> leer n filas y 5 columnas del archivo pickle</div><div><code>pd.read_parquet('ruta/nombre_archivo.parquet')</code> leer un archivo parquet</div></div> | <div><div>Metodos de estadistica</div><div><code>df['columna'].mean()</code> <code>mode()</code> <code>median()</code> <code>var()</code> <code>std()</code> calcula la media/moda/mediana/variación/desviación estándar de los valores de una columna <code>df['columna1'].corr(df['columna2'])</code> calcula la correlacion entre dos variables <code>matriz_correlacion = df.corr()</code> crea una matriz mostrando las correlaciones entre todos los variables <code>df_crosstab = pd.crosstab(df['columna1'], df['columna2'], normalize = True, margins = True)</code> <code>normalize</code> muestra los valores en porcentajes (por uno) <code>margins</code> muestra los totales y subtotales <code>media_ponderada = np.average(df['columna'], weights = w)</code> calcula la media ponderada según los pesos <code>percentil_n = np.percentile(df['columna'], n)</code> saca el valor en el percentil n <code>q3, q1 = np.percentile(df["columna"], [75, 25])</code> saca los tercer y primer cuartiles</div><div>Sidetable: frecuencias de datos</div><div><code>df.stb.freq(['columna'])</code> devuelve un dataframe con informacion sobre la frecuencia de ocurrencia de cada categoría de un variable categorica parametros: <code>thresh = n</code> limita los valores mostrados a los más frecuentes hasta un umbral de n% cumulative y agrupando los restantes bajo la etiqueta "other" <code>other_label = 'etiqueta'</code> cambia la etiqueta 'other' <code>value = 'columna'</code> ordena los resultados por la columna especificada <code>df.stb.freq(['columna1', 'columna2'])</code> combina dos columnas y devuelve las frecuencias de las subcategories</div><div><code>df.stb.missing(['columna'])</code> devuelve informacion sobre la frecuencia de datos nulos</div></div> | <div>Calcular tres desviaciones estandares: <code>media = df.column.mean()</code> <code>desviacion = df.column.std()</code></div> <div><code>lcb = media - desviacion * 3</code> <code>ucb = media + desviacion * 3</code></div> <div>Eliminar Outliers</div> <div><code>outlier_step = 1.5 * IQR</code> calcular outlier step <code>outliers_data = df[(df['columna'] < Q1 - outlier_step) (df['columna'] > Q3 + outlier_step)]</code> identificar datos fuera del rango del maximo hasta el minimo <code>lista_outliers_index = list(outliers_data.index)</code> crear una lista de los indices de las filas con outliers</div> <div><code>if outliers_data.shape[0] > 0:</code> dicc_indices[key] = (list(outliers_data.index)) crear un diccionario de los indices de las filas con nulos; se puede hacer iterando por columnas</div> <div><code>valores = dicc_indices.values()</code> sacar todos los valores e.g. todos los indices <code>valores = {indice for sublista in valores for indice in sublista}</code> set comprehension para eliminar duplicados <code>df_sin_outliers = df.drop(df.index[list(valores)])</code> crear nuevo dataframe sin outliers</div> <div>Reemplazar Outliers</div> <div><code>for k, v in dicc_indices.items():</code> media = df[k].mean() for i in v: df.loc[i,k] = media reemplazar outliers por la media</div> |

| Python Cheat Sheet 4 |
|---|
| |
| Pandas |
| |
| Union de datos |
| <p>.concat() unir dataframes con columnas en comun</p> <p>df_union = pd.concat([df1, df2, df3], axis=b, join = 'inner/outer', ignore_index = True/False)</p> <p>parametros:</p> <p>axis = 0 une por columnas - los dataframes van uno encima del otro; las columnas tienen que ser de formatos compatible</p> <p>axis = 1 une por filas - los dataframes van uno al lado del otro; los datos deben ser relacionados para que tenga sentido</p> <p>join = 'inner' solo se quedan elementos que aparecen en todos los dataframes</p> <p>join = 'outer' se queda todo los datos de todos los dataframes</p> <p>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)</p> <p>.merge() unir las columnas de un dataframe a otro</p> <p>df_nuevo = df1.merge(df2, on = 'columna') inner merge</p> <p>df_nuevo = pd.merge(left = df1, right = df2, how='left', left_on = 'columna_df1', right_on = 'columna_df2') left merge</p> <p>parametros:</p> <p>how = 'left' 'right' 'outer' 'inner' 'cross'</p> <p>on = columna [columna1, columna2, etc] si las columnas se llaman igual en los dos dataframes</p> <p>left_on = columna_df1 right_on = columna_df2 para especificar por donde hacer el merge</p> <p>suffixes = ['left', 'right'] por defecto nada, el sufijo que apareciera en columnas duplicadas</p> <p>.join() unir dataframes por los indices</p> <p>df_nuevo = df1.join(df2, on = 'columna', how = 'left') inner merge</p> <p>parametros:</p> <p>how = 'left' 'right' 'outer' 'inner' por defecto left</p> <p>on = columna la columna o indice por el que queremos hacer el union; tienen que tener el mismo nombre en los dos dataframes</p> <p>lsuffix = 'string' rsuffix = 'string' por defecto nada, el sufijo que apareciera en columnas duplicadas</p> |
| Group By |
| <p>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)</p> <p>df_groupby.ngroups devuelve el numero de grupos</p> <p>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</p> <p>df_grupo1 = df_groupby.get_group("grupo1") devuelve un dataframe con los resultados de un grupo (la categoria indicada como grupo1)</p> <p>Cálculos con groupby:</p> <p>df_nuevo = df.groupby("columna_categoria").mean() devuelve un dataframe con la media de todas las columnas de valores numéricos, por categoría</p> <p>df_nuevo = df.groupby("columna_categoria")["columna1"].mean() devuelve un dataframe con la media de la columna especificada</p> <p>count() número de observaciones</p> <p>no nulas</p> <p>describe() resumen de los principales estadísticos</p> <p>sum() suma de todos los valores</p> <p>mean() media de los valores</p> <p>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</p> <p>dropna = False para tener en cuenta los Nan en los cálculos (por defecto es True)</p> |

| Subsets: loc e iloc |
|---|
| <p>df.loc["etiqueta_fila", "etiqueta_columna"] devuelve el contenido de un campo en una columna de una fila</p> <p>df.loc["etiqueta_fila",:] devuelve los valores de todas las columnas de una fila</p> <p>df.loc[:, "etiqueta_columna"] devuelve los valores de todas las filas de una columna</p> <p>df.iloc[indice_fila, indice_columna] devuelve el contenido de un campo en una columna de una fila</p> <p>df.iloc[indice_fila, :] devuelve los valores de todas las columnas de una fila</p> <p>df.iloc[:, indice_columna] devuelve el contenido de un campo en una columna de una fila</p> <p>df.loc[[lista_etiquetas_filas], [lista_etiquetas_columnas]] devuelve el contenido de varias filas / varias columnas</p> <p>df.loc[[lista_indices_filas], [lista_indices_columnas]] devuelve el contenido de varias filas / varias columnas</p> <p>- se puede usar los indices/rangos de las listas [start:stop:step] dentro de los loc/iloc</p> <p>df.loc[df.etiqueta > x] seleccionar datos basado en una condición usando operadores comparativos</p> <p>df.loc[(df.etiqueta > x) & (df.etiqueta == y)] seleccionar datos que tienen que cumplir las dos condiciones (and)</p> <p>df.loc[(df.etiqueta > x) (df.etiqueta == y)] seleccionar datos que tienen que deben cumplir una de las dos condiciones (or)</p> <p>df.iloc[list(df.etiqueta > x), :] iloc no acepta una Serie booleana; hay que convertirla en lista</p> <p>variable_df.head(n) devuelve las n primeras filas del df, o 5 por defecto</p> |
| Filtrados de datos |
| <p>Filtrado por una columna con operadores de comparación</p> <p>df_filtrado = df[df["nombre_columna"] == valor] extrae las filas donde el valor de la columna igual al valor dado</p> <p>Filtrado por multiples columnas con operadores logicos</p> <p>df_filtrado = df[(df["columna1"] == valor) & (df["columna2"] == valor) & (df["columna3"] > n valor)] extrae las filas donde los valores de las columnas cumplan las condiciones en parentesis</p> <p>df_filtrado = df[(df["columna1"] == valor) (df["columna1"] == valor) extrae las filas donde los valores de las columnas cumplan con una condición u otra</p> <p>df_filtrado = ~(df[df["columna1"] == valor]) extrae las filas donde los valores de las columnas NO cumplan con la condición</p> |

| Filtrados de datos |
|--|
| <p>Metodos de pandas de filtrar</p> <p>df_filtrado = 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)</p> <p>df_filtrado= df[df["nombre_columna"].str.contains (patron, regex = True, na = False)] extrae las filas cuyas valores de la columna nombrada contienen el patron de regex</p> <p>df_filtrado = df[df["nombre_columna"].str.contains ("substring", case = False, regex = False)] extrae las filas cuyas valores de la columna nombrada contienen el substring, no siendo case sensitive</p> <p>df_filtrado = df[df["nombre_columna"].str.contains ("substring", case = False, regex = False)] extrae las filas cuyas valores de la columna nombrada contienen el substring, no siendo case sensitive</p> <p>df[pd.notnull(df["nombre_columna"])] devuelve las filas que no tiene valores nulos en la columna especificada</p> |
| Cambiar columnas |
| <p>lista_columnas = df.columns.to_list() crea una lista de los nombres de las columnas del dataframe</p> <p>df.set_index(["nombre_columna"], inplace = True) establece el índice utilizando uno o mas columnas; puede sustituir o ampliar un índice existente</p> <p>inplace = True los cambios sobrescriben sobre el df</p> <p>* cuando una columna se cambia a índice ya no es columna *</p> <p>df.reset_index(inplace = True) quitar una columna como indice para que vuelva a ser columna; crea un dataframe de una serie</p> <p>Renombrar columnas</p> <p>df.rename(columns = {"nombre_columna": "nombre_nueva"}, inplace = True) cambia los nombres de una o mas columnas</p> <p>ejemplo de dict comprehension para crear diccionario sobre las columnas existentes de un dataframe:</p> <p>diccionario = {col : col.upper() for col in df.columns}</p> <p>df.rename(columns = diccionario, inplace = True) cambia los nombres de las columnas según el diccionario</p> <p>Eliminar columnas</p> <p>df.drop(columns = ["columna1", "columna2"], axis = b, inplace=True) eliminar una o mas columnas o filas segun lo que especificamos</p> <p>Reordenar columnas</p> <p>df = df.reindex(columns = lista_reordenada) cambia el orden de las columnas del dataframe segun el orden de la lista reordenada</p> |

| Crear columnas |
|--|
| <p>Creacion de ratios</p> <p>df["columna?ratio"] = df.apply(lambda df: df["columna1"] / df["columna2"], axis = 1)</p> <p>Creacion de porcentajes</p> <p>def porcentaje(columna1, columna2): return (columna1 * 100) / columna2</p> <p>df["columna_%"] = df.apply(lambda df: porcentaje(df["columna1"], datos["columna2"]), axis = 1)</p> <p>df["nueva_columna"] = np.where(df["nombre_columna"] > n, "categoria_if_true", "categoria_if_false") crea una nueva columna basada en una condición</p> <p>df["nueva_columna"] = np.select(lista_de_condiciones, lista_de_opciones) crea una nueva columna con los valores basados en multiples condiciones</p> <p>df["columna_nueva"] = pd.cut(x = df["nombre_columna"], bins = [n,m,l..], labels = ['a', 'b', 'c']) separa los elementos de un dataframe en diferentes intervalos (n-m, m-l, etc), creando una columna nueva que indica en cual intervalo cae el valor; con labels se puede asignar un string a cada intervalo</p> <p>Crear columnas</p> <p>df["nueva_columna"] = (df["etiqueta_columna"] + x) crea una nueva columna basada en otra</p> <p>df = df.assign(nueva_columna= df["etiqueta_columna"] + x) crea una nueva basada en otra</p> <p>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*</p> <p>df.insert(indice_nueva_columna, "nombre_columna", valores) crea una nueva columna en la indice indicada</p> <p>allow_duplicates = True parametro cuando queremos permitir columnas duplicadas (por defecto es False)</p> |
| 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)</p> <p>crea una columna nueva con los valores de otra columna transformados según la función indicada</p> <p>df['columna_nueva'] = df['col_1'].apply(lambda x: x.método() if x > 1)</p> <p>crea una columna nueva con los valores de otra columna transformados según la lambda indicada</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> <p>df.applymap(funcion, na_action=None, **kwargs) acepta y devuelve un escalar a cada elemento de un dataframe; se tiene que aplicar a todo el DataFrame</p> <p>df['columna'] = df['columna'].map(mapa, na_action = 'ignore') reemplaza valores de la columna según el mapa, que puede ser un diccionario o una serie; solo se puede aplicar a una columa en particular.</p> <p>apply() con 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) crea una columna nueva del año solo usando un método de la libreria datetime; ("%B") para meses</p> |

| Cambiar valores |
|--|
| <p>Reemplazar valores basados en indices y condiciones:</p> <p>indices_filtrados = df.index[df["columna"] == "valor"]</p> <p>for indice in indices_filtrados: df["nombre_columna"].iloc[indice] = "valor_nuevo"</p> <p>Reemplazar valores basados en metodos NumPy:</p> <p>df.replace(to_replace = valor, value = valor_nuevo, inplace = True) reemplaza cierto valor por otro que especificamos</p> <p>df["nombre_columna"].replace(to_replace = valor, value = valor_nuevo, inplace = True) reemplaza cierto valor en una columna por otro que especificamos</p> <p>df[["columna1", "columna2"]] = df[["columna1", "columna2"]].replace(r"string", "string", regex=True) cambiar un patron/string por otro en multiples columnas</p> <p>df["nombre_columna"] = df["nombre_columna"] + x reemplaza los valores de la columna por el valor + x (o otro valor que indicamos)</p> |
| datetime |
| <p>import datetime</p> <p>datetime.now() devuelve la fecha actual</p> <p>timedelta(n) representa una duración la diferencia entre dos instancias; n es un numero de días</p> <p>datetime.strptime(variable_fecha, '%Y-%m-%d') formatea la fecha al formato indicado</p> <p>ayer = datetime.now() - timedelta(1)</p> <p>ayer = datetime.strptime(ayer, '%Y-%m-%d')</p> <p>df["fecha"] = ayer crea una columna con la fecha de ayer</p> |

Python Cheat Sheet 5

Matplotlib

Gráficas

```
import matplotlib.pyplot as plt
```

`plt.rcParams["figure.figsize"] = (10,8)`
`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

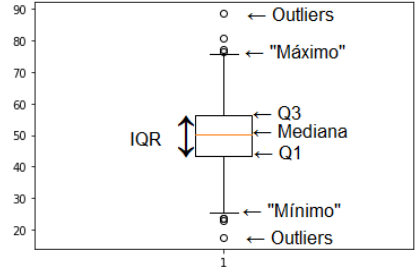
`plt.scatter(df["columna1"], df["columna2"])` crea una gráfica de dispersión donde los ejes son: columna1 – x, columna2 – y

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 características de una variable numerica; x es la variable de interés
- el mínimo es lo mismo que Q1 - 1.5 * IQR
- el máximo es lo mismo que Q3 + 1.5 * IQR

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

Seaborn gráficas

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 donde los marcadores no se solapan

Swarm plot

`fig = sns.swarmplot(x = 'columna1', y = 'columna2', data = df, hue = 'columna')` crea una gráfica de dispersión donde los marcadores no se solapan

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

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
kde = True muestra una curva de la distribucion

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' | 'violin' | '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

Heatmap

`sns.heatmap(df.corr(), cmap = 'color_palette', annot = True, vmin = -1, vmax = 1)` crea un heatmap con una escala de colores que refleja los valores de correlacion
annot = True para que aparezcan los valores
vmin/vmax establecen la escala de color

Regplot

`fig = sns.regplot(x = 'columna1', y = 'columna2', data = df, scatter_kws = {'color': 'blue'}, line_kws = {'color': 'blue'})`crea un scatterplot mas la línea de regresión; nos permite encontrar la mejor función de la recta que permite predecir el valor de una variable sabiendo los valores de otra variable

Jointplot

`sns.jointplot(x = 'columna1', y = 'columna2', data = df, color = 'blue', kind = 'tipo')` crea un scatterplot o regplot con histogramas pegados en los lados para cada variable

Exportar figuras

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

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 establece como es cada grafica con los indices:
`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 los nombres y/o la rotacion de las etiquetas de los valores en los ejes

Crear subplots en un for loop

`fig, axes = plt.subplots(numero_filas, numero_columnas, figsize = (n, m))`
`axes = axes.flatten()`
`for col in df.columns:`
`fig = sns.plot(x=col, data=df, ax=axes[i])`

Usos de los tipos de gráficas

Datos categóricos

Barras

- muestra la relación entre una variable numérica y categórica
- barplot si tienes una variable numérica
- countplot para contar registros/filas por categoría

Pie chart/quesitos

- determinación de frecuencias

Datos numéricos

Líneas

- tendencias/evolución de una o más variables numéricas (normalmente sobre un período de tiempo)

Histograma

- distribución de una variable numérica

Boxplot

- representación de las medidas de posición más usadas: mediana, IQR, outliers

Scatterplot

- muestra la relación entre dos variables numéricas

Regplot

- scatterplot con una línea de regresión

Swarmplot

- tipo de gráfica de dispersión para representar variables categóricas; evita que se solapan los marcadores

Violinplot

- para visualizar la distribución de los datos y su densidad de probabilidad

Pairplot

- para representar múltiples relaciones entre dos variables

Heatmap

- evaluar la correlación entre las variables en una matriz de correlación

Personalización

Titulos

`plt.title(label = "titulo")` asignar un titulo a la gráfica

Ejes

`plt.xlabel("etiqueta_eje_x")` asignar nombre al eje x
`plt.ylabel("etiqueta_eje_y")` asignar nombre al eje y
`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`fig.set(xlabel = 'etiqueta_eje_x', ylabel = 'etiqueta_eje_y')` asignar nombre a los ejes
`fig.set_title('titulo')` asignar un titulo a la gráfica`fig.set_xlabel(xlabel = "etiqueta_eje_x", fontsize = n)`
`fig.set_ylabel(ylabel = "etiqueta_eje_y", fontsize = n)``fig.set(xticks = [1, 2, 3])`
`fig.set(yticks = [1, 2, 3, 4, 5])`
`fig.set(xticklabels = ['0%', '20%', '40%', '60%', '80%', '100%'])`
`fig.set(yticklabels = ['cat1', 'cat2', 'cat3'])``fig.set_xticklabels(labels = [0, 500, 1000, 1500], size=n)`
`fig.set_yticklabels(labels = fig.get_yticklabels(), size=n)`

Para poner etiquetas encima de las barras

`for indice, valor in enumerate(df ["col"]):`
`plt.text(valor+1, indice, valor,`
`horizontalalignment='left', fontsize= 16)``order = df.sort_values('columnay', ascending=False) ['columnax']`
`sns.set(font_scale=2)`
`plt.rcParams.update({'font.size': 22})` font size general

Legendas

`plt.legend(labels = ['label1', 'label2', etc])` muestra la leyenda cuando mostramos la figura
`plt.legend(bbox_to_anchor = (1, 1))` coloca la leyenda en relación con los ejes

Quitar bordes

`fig.spines[["top", "right"]].set_visible(False)`

Linea de tres desviaciones estandares:

`fig.axvline(x=valor, c='color', label='valor')`
`fig.axvline(x=valor, c='color', label='valor')`

Cuadrícula

`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

Personalización

Colores

`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

Paletas Seaborn:

Accent', 'Accent_r', 'Blues', 'Blues_r', 'BrBG', 'BrBG_r', 'BuGn', 'BuGn_r', 'BuPu', 'BuPu_r', 'CMRmap', 'CMRmap_r', 'Dark2', 'Dark2_r', 'GnBu', 'GnBu_r', 'Greens', 'Greens_r', 'Greys', 'Greys_r', 'OrRd', 'OrRd_r', 'Oranges', 'Oranges_r', 'PRGn', 'PRGn_r', 'Paired', 'Paired_r', 'Pastell1', 'Pastell1_r', 'Pastel2', 'Pastel2_r', 'PiYG', 'PiYG_r', 'PuBu', 'PuBuGn', 'PuBuGn_r', 'PuBu_r', 'PuOr', 'PuOr_r', 'PuRd', 'PuRd_r', 'Purples', 'Purples_r', 'RdBu', 'RdBu_r', 'RdGy', 'RdGy_r', 'RdPu', 'RdPu_r', 'RdYlBu', 'RdYlBu_r', 'RdYlGn', 'RdYlGn_r', 'Reds', 'Reds_r', 'Set1', 'Set1_r', 'Set2', 'Set2_r', 'Set3', 'Set3_r', 'Spectral', 'Spectral_r', 'Wistia', 'Wistia_r', 'YlGn', 'YlGnBu', 'YlGnBu_r', 'YlGn_r', 'YlOrBr', 'YlOrBr_r', 'YlOrRd', 'YlOrRd_r', 'afmhot', 'afmhot_r', 'autumn', 'autumn_r', 'binary', 'binary_r', 'bone', 'bone_r', 'brg', 'brg_r', 'bwr', 'bwr_r', 'cividis', 'cividis_r', 'cool', 'cool_r', 'coolwarm', 'coolwarm_r', 'copper', 'copper_r', 'crest', 'crest_r', 'cubehelix', 'cubehelix_r', 'flag', 'flag_r', 'flare', 'flare_r', 'gist_earth', 'gist_earth_r', 'gist_gray', 'gist_gray_r', 'gist_heat', 'gist_heat_r', 'gist_ncar', 'gist_ncar_r', 'gist_rainbow', 'gist_rainbow_r', 'gist_stern', 'gist_stern_r', 'gist_yarg', 'gist_yarg_r', 'gnuplot', 'gnuplot2', 'gnuplot2_r', 'gnuplot_r', 'gray', 'gray_r', 'hot', 'hot_r', 'hsv', 'hsv_r', 'icefire', 'icefire_r', 'inferno', 'inferno_r', 'jet', 'jet_r', 'magma', 'magma_r', 'mako', 'mako_r', 'nipy_spectral', 'nipy_spectral_r', 'ocean', 'ocean_r', 'pink', 'pink_r', 'plasma', 'plasma_r', 'prism', 'prism_r', 'rainbow', 'rainbow_r', 'rocket', 'rocket_r', 'seismic', 'seismic_r', 'spring', 'spring_r', 'summer', 'summer_r', 'tab10', 'tab10_r', 'tab20', 'tab20_r', 'tab20b', 'tab20b_r', 'tab20c', 'tab20c_r', 'terrain', 'terrain_r', 'turbo', 'turbo_r', 'twilight', 'twilight_r', 'twilight_shifted', 'twilight_shifted_r', 'viridis', 'viridis_r', 'vlag', 'vlag_r', 'winter', 'winter_r'

palette='light:nombre_paleta'|'dark:nombre_paleta'

Marcadores

`marker = 'tipo'` establece el tipo de marcador; se usa con plt.scatter y plt.plot

| | |
|-------------------------|-------------------|
| "." Punto | "P" Más (relleno) |
| "," Pixel | "*" Estrella |
| "o" Circulo | "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 | |

| NumPy (Numerical Python) | Indices, Subsets, Metodos de Arrays | Operaciones estadísticas y matemáticas | Funciones de conjuntos | Estadística |
|--|--|---|---|---|
| <h3>Crear arrays</h3> <h4>Crear arrays de listas</h4> <code>array = np.array(lista, dtype= tipo)</code> crea un array unidimensional de una lista <code>array = np.array([lista1, lista2])</code> crea un array bidimensional de dos listas <code>array = np.array([listadelistas1, listadelistas2])</code> crea un array bidimensional de dos listas <h4>Crear otros tipos de arrays</h4> <code>array = np.arange(valor_inicio, valor_final, saltos)</code> crea un array usando el formato [start:stop:step] <code>array = np.ones(z,y,x)</code> crea un array de todo unos de la forma especificada <code>array2 = np.ones_like(array1)</code> crea un array de todo unos de la forma basada en otra array <code>array = np.zeros(z,y,x)</code> crea un array de todo zeros de la forma especificada <code>array2 = np.zeros_like(array1)</code> crea un array de todo zeros de la forma basada en otra array <code>array = np.empty((z,y,x), tipo)</code> crea un array vacio con datos por defecto tipo float <code>array2 = np.empty_like(array1)</code> crea un array vacia con la forma basada en otra array <code>array = np.eye(z,y,x, k = n)</code> crea un array con unos en diagonal empezando en la posicion k <code>array = np.identity(x)</code> crea una matriz de identidad con ceros en filas y unos en la diagonal, de forma cuadrada | <h3>Indices de arrays</h3> <code>array[i]</code> devuelve la indice i; las indices de los arrays unidimensionales funcionan igual que las listas <code>array[i, j]</code> o <code>array[i][j]</code> devuelve el elemento de la columna j de la fila i <code>array[:,n]</code> seleccionar todas las filas y las columnas hasta n-1 <code>array[h, i, j]</code> o <code>array[h][i][j]</code> devuelve el elemento de la columna j de la fila i del array h <code>array[h][i][j] = n</code> cambiar el valor del elemento en esta posicion al valor n <h3>Subsets</h3> <code>array > n</code> devuelve la forma del array con True o False según si el elemento cumple con la condición o no <code>array[array > n]</code> devuelve un subset: todos los valores que cumplen la condición en una lista dentro de un array <code>array[(array > n) & (array < m)]</code> devuelve un subset: todos los valores que cumplen las condiciones en una lista dentro de un array; se puede usar para "or" <h3>Metodos de arrays</h3> <code>nuevo_array = array.copy()</code> crea un a copia del array <code>np.transpose(array_bidimensional)</code> cambia los filas del array a columnas y las columnas a filas <code>np.transpose(array_multidimensional)</code> cambia el número de columnas al número de arrays y viceversa; el número de filas no cambia <code>np.transpose(array_multidimensional, (z,y,x))</code> hace la transposicion segun lo que especifecemos usando las posiciones de la tupla (0,1,2) de la forma original <code>array = np.arange(n).reshape((y,x))</code> crea un array usando reshape para definir la forma <code>array = np.reshape(array, (z,y,x))</code> crea un array con los valores de otro array usando reshape para definir la forma <code>array = np.swapaxes(array, posicion, posicion)</code> intercambia dos ejes de una matriz usando las posiciones (z=0,y=1,x=2) de la forma original <h3>Otras operaciones</h3> <code>np.sort(array)</code> devuelve un array con los valores de cada fila ordenados en orden ascendente por defecto <code>np.sort(array, axis = 0)</code> devuelve un array con los valores de cada columna ordenados en orden ascendente <code>np.sort(-array)</code> devuelve un array con los valores de cada fila ordenados en orden descendente <code>np.round(array, decimals = x)</code> devuelve un array con los valores del array redondeados a x decimales <code>np.round(array, decimals = x)</code> devuelve un array con los valores del array redondeados a x decimales <code>np.where(array > x)</code> devuelve los indices de los valores que cumplan la condición, por fila y columna <h3>Operaciones con arrays</h3> <code>np.add(array1, array2)</code> suma dos arrays <code>np.subtract(array1, array2)</code> resta el array2 del array1 <code>np.multiply(array1, array2)</code> multiplica dos arrays <code>np.divide(array1, array2)</code> divide el array1 por el array2 array + n, n * array, etc. - operadores algebraicos | <h3>Operaciones estadísticas y matemáticas</h3> <h4>El parametro axis en arrays bidimensionales:</h4> <code>axis = 0</code> columnas <code>axis = 1</code> filas - si especificamos el axis, la operación devuelve el resultado por cada fila o columna. Por ejemplo: <code>np.sum(array, axis = 0)</code> devuelve un array con la suma de cada fila <h4>El parametro axis en arrays multidimensionales:</h4> <code>axis = 0</code> dimensión <code>axis = 1</code> columnas <code>axis = 2</code> filas - si especificamos el axis, la operación devuelve el resultado por cada dimensión, fila o columna. Por ejemplo: <code>np.sum(array_3D, axis = 0)</code> devuelve un array de una matriz con la suma de todas las matrices <code>np.sum(array_3D, axis = 1)</code> devuelve un array donde las filas contienen las sumas de las columnas de cada matriz <h4>Operaciones con parámetro del axis:</h4> <code>np.sum(array_3D)</code> devuelve la suma de todos los elementos de los matrices <code>np.mean(array)</code> devuelve la media de todo el array <code>np.std(array)</code> devuelve la desviación estándar de todo <code>np.var(array)</code> devuelve la varianza de valores de todo <code>np.min(array)</code> devuelve el valor mínimo del array <code>np.max(array)</code> devuelve el valor máximo del array <code>np.sum(array)</code> devuelve la suma de los elementos del array <code>np.cumsum(array)</code> devuelve un array con la suma acumulada de los elementos a lo largo del array <code>np.cumprod(array)</code> devuelve un array con la multiplicación acumulada de los elementos a lo largo del array <h4>Operaciones sin parámetro del axis:</h4> <code>np.sqrt(array)</code> devuelve un array con la raíz cuadrada no negativa de cada elemento del array <code>np.exp(array)</code> devuelve un array con el exponencial de cada elemento del array <code>np.mod(array1, array2)</code> devuelve un array con el resto de la división entre dos arrays <code>np.mod(array1, n)</code> devuelve un array con el resto de la división entre el array y el valor de n <code>np.cos(array)</code> devuelve un array con el coseno de cada elemento del array <code>np.sin(array)</code> devuelve un array con el seno de cada elemento del array <code>np.sin(array)</code> devuelve un array con la tangente de cada elemento del array <h4>Operaciones de comparación en arrays bidimensionales</h4> <code>np.any(array > n)</code> devuelve True o False segun si cualquier valor del array cumpla con la condicion <code>np.any(array > n, axis = b)</code> 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 <code>np.all(array > n)</code> devuelve True o False segun si todos los valores del array cumpla con la condicion <code>np.all(array > n, axis = b)</code> 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 | <code>np.unique(array)</code> devuelve un array con los valores únicos del array ordenados <code>np.unique(array, return_index=True)</code> devuelve un array con los valores únicos del array ordenados y un array con la posición de la primera instancia de cada valor <code>np.unique(array, return_inverse=True)</code> devuelve un array con los valores únicos del array ordenados y un array con las posiciones de cada elemento de cada valor <code>np.unique(array, return_counts=True)</code> devuelve un array con los valores únicos del array ordenados y un array con el número de veces que aparece cada valor <code>np.unique(array, axis = b)</code> devuelve un array con los valores únicos ordenados de las filas o columnas <h3>Funciones para arrays unidimensionales</h3> <code>np.intersect1d(array1, array2)</code> devuelve un array con los valores únicos de los elementos en común de dos arrays <code>np.intersect1d(array1, array2, return_indices=True)</code> 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 <code>np.union1d(array1, array2)</code> devuelve un array ordenado con los elementos resultantes de unir dos arrays (valores únicos) <code>np.in1d(array1, array2)</code> devuelve un array con True o False por cada elemento de array1 según si aparece el mismo valor en array2 <code>np.setdiff1d(array1, array2)</code> devuelve un array ordenado con los valores únicos que están en array1 pero no en array2 <code>np.setxor1d(array1, array2)</code> devuelve un array ordenado con los valores únicos que NO están en común de los dos arrays | <h3>Tablas de frecuencias</h3> <h4>Frecuencias absolutas</h4> el número de veces que se repite un número en un conjunto de datos <code>df = df.groupby('columna').count().reset_index()</code> <h4>Frecuencias relativas</h4> las veces que se repite un número o categoría en un conjunto de datos respecto al total, en porcentajes <code>df_group_sin_str = df_group.drop('columna_str', axis=1)</code> <code>frecuencia_relativa = df_group_sin_str / df.shape[0] * 100</code> <code>columnas = df_group_sin_strings.columns</code> <code>df_group[columnas] = frecuencia_relativa</code> <h3>Tablas de contingencia</h3> tabla de frecuencias que cuenta todas las combinaciones posibles de cada pareja de valores de las columnas que estamos intentando comparar <code>df_crosstab = pd.crosstab(df['columna1'], df['columna2'], normalize = True, margins = True)</code> <code>normalize</code> muestra los valores en porcentajes (por uno) <code>margins</code> muestra los totales y subtotales <h3>Coefficiente de correlación de Pearson</h3> - nos permite conocer la intensidad y dirección de la relación entre las dos variables - coeficiente > 0: correlación positiva - coeficiente < 0: correlación negativa - coeficiente = 1 o -1: correlación total - coeficiente = 0: no existe relación lineal <code>df['columna1'].corr(df['columna2'])</code> calcula la correlacion entre dos variables <code>matriz_correlacion = df.corr()</code> crea una matriz mostrando las correlaciones entre todos los variables <code>sns.heatmap(df.corr()[['column1', 'column2']], cmap = 'color_palette', annot = True, vmin = -1, vmax = 1)</code> crea una grafica heatmap de la matriz de correlaciones <h3>Sesgos (skewness)</h3> 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 <code>sns.displot(df['columna'], kde = True)</code> crea un histograma que muestra la distribution de los valores <code>import scipy.stats import skew</code> <code>skew(df['columna'])</code> muestra el valor del sesgo de una variable <h3>Intervalos de confianza</h3> describe la variabilidad entre la medida obtenida en un estudio y la medida real de la población (el valor real) <code>import scipy.stats as st</code> <code>st.t.interval(alpha = n, df = len(df['columna'])-1, loc = np.mean(df['columna']), scale = st.sem(df['columna']))</code> 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 |
| <h3>NumPy Random</h3> <code>np.random.seed(x)</code> establece la semilla aleatoria del generador de números aleatorios, para que las funciones random que van después siempre cogerán los mismos valores "aleatorios" <h3>Crear arrays con valores aleatorios</h3> <code>array = np.random.randint(inicio, final, forma_matriz)</code> 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 <code>array = np.random.randint(inicio, final)</code> devuelve un número aleatorio en el rango <code>array = np.random.rand(z,y,x)</code> crea un array de floats aleatorias con la forma que le especificemos; por defecto genera números aleatorios entre 0-1 <code>array = np.random.random_sample((z,y,x))</code> crea un array de floats aleatorias con la forma que le especificemos; por defecto genera números aleatorios entre 0-0.9999999... <code>array = np.random.z,y,x=None)</code> devuelve un número aleatorio en 0 y 0.999999999999... <code>np.round(np.random.rand(z,y,x), n)</code> crear array con floats de n decimales <code>np.random.uniform(n,m, size = (z,y,x))</code> genera muestras aleatorias de una distribución uniforme en el intervalo entre n y m <code>np.random.binomial(n,m, size = (z,y,x))</code> genera muestras con una distribución binomial; n es el numero total de pruebas; m es la probabilidad de éxito <code>np.random.normal(loc = n, scale = m, size = (z,y,x))</code> genera números aleatorios de una distribución normal (curva de campana); loc es la media; scale es la desviación estándar <code>np.random.permutation(array)</code> devuelve un array con los mismos valores mezclados aleatoriamente | | | | |

EDA Exploratory Data Analysis

Análisis exploratorio de datos

El Análisis Exploratorio de Datos se refiere al proceso de realizar una serie de investigaciones inciales sobre los datos que tenemos para poder descubrir patrones, detectar anomalías, probar hipótesis y comprobar suposiciones con la ayuda de estadísticas y representaciones gráficas.

1. Entender las variables

- que variables temenos
- `head()`, `.tail()`, `.describe()`, `.info()`, `.shape`
- que tipos de datos
- `.dtypes()`, `.info()`
- si temenos nulos o duplicados
- `.isnull().sum()`
- `.duplicated().sum()`
- que valores unicos temenos
- `.unique()`, `.value_counts()`

librería `sidetable`:
`stb.freq()` devuelve el `value_counts` de variables categóricas, mas el porcentaje, cuenta cumulativa y porcentaje cumulativa
`stb.missing()` tabla de cuenta de nulos y el porcentaje del total

2. Limpiar el dataset

- quitar duplicados (filas o columnas)
- cambiar nombres de columnas
- cambiar tipo de datos de columnas
- ordenar columnas
- separar columna en dos con `str.split()`
- crear intervalos con `pd.cut()`
- crear porcentajes o ratios
- decidir como tratar outliers: mantenerlos, eliminarlos, o reemplazarlos con la media, mediana o moda; o aplicar una imputacion
- decidir como tratar nulos:
 - eliminar filas o columnas con nulos `drop.na()`
 - imputar valores perdidos:
 - reemplazarlos con la media, mediana o moda usando `.fillna()` o `.replace()`
 - imputer con metodos de machine learning usando la libreria `sklearn: Simple-Imputer`, `Iterative-Imputer`, o `KNN Imputer`

3. Analizar relaciones entre variables

Analizar relaciones entre las variables

- para encontrar patrones, relaciones o anomalías

Relaciones entre dos variables numéricas:

- `scatterplot`
- `regplot` - `scatterplot` con línea de regresion
- matriz de correlación y `heatmap`
- `joinplot` - permite emparejar dos gráficas - una histograma con `scatter` o `reg plot` por ejemplo

Relaciones entre dos variables categóricas:

- `countplot`

Relaciones entre variables numéricas y categóricas:

- `swarmplot`
- `violinplot`
- `pointplot`
- `boxplot`

ETL: Extract, Transform, Load

Extraccion

- obtener datos crudos y almacenarlos
 - Tablas de bases de datos SQL o NoSQL
 - Ficheros de texto plano
 - Emails
 - Información de páginas web
 - Hojas de cálculo
 - Ficheros obtenidos de API's

Transformación

- procesar los datos, unificarlos, limpiarlos, validarlos, filtrarlos, etc.
 - Formetear fechas
 - Reordenar filas o columnas
 - Unir o separar datos
 - Combinar las fuentes de datos
 - Limpiar y estandarizar los datos
 - Verificar y validar los datos
 - Eliminar duplicados o datos erroneos
 - Filtrado, realización de calculos o agrupaciones

Carga

- cargar los datos en su formato de destino, el tipo de lo cual dependerá de la naturaleza, el tamaño y la complejidad de los datos. Los sistemas más comunes suelen ser:
 - Ficheros csv
 - Ficheros json
 - Bases de datos
 - Almacenes de datos (Data Warehouse)
 - Lagos de datos (Data Lakes)

APIs

`import requests` libreria para realizar petitions HTTP a una URL, para hacer web scraping

`url = 'enlace'` el enlace de la que queremos extraer datos

`header = {}` opcional; contiene informacion sobre las peticiones realizadas (tipo de ficheros, credenciales)

`response = requests.get(url=url, header = header)` pedimos a la API que nos de los datos

`variables = {'parametro1': 'valor1', 'parametro2': 'valor2'}`

`response = request.get(url=url, params=variables)` pedimos a la API que nos de los datos con los parametros segun el diccionario de parametros que le pasamos

`response.status_code` devuelve el status de la peticion

`response.reason` devuelve el motive de codigo de estado

`response.text` devuelve los datos en formato string

`response.json()` devuelve los datos en formato json

`df = pd.json_normalize(response.json)` devuelve los datos en un dataframe

Codigos de respuesta de HTTP

| | |
|---------------------------------------|--------------------------------|
| 1XX informa de una respuesta correcta | 4XX error durante peticion |
| 2XX codigo de exito | 401 peticion incorrecta |
| 200 OK | 402 sin autorizacion |
| 201 creado | 403 prohibido |
| 202 aceptado | 404 no encontrado |
| 204 sin contenido | 5XX error del servidor |
| 3XX redireccion | 501 error interno del servidor |
| | 503 servicio no disponible |

Machine Learning: Preparación

Hipotesis Nula y Errores Tipo I y II

Hipótesis nula (H0)

- en general es la afirmación contraria a la que queremos probar

Hipótesis alternativa (H1)

- en general la afirmación que queremos comprobar

p-valor

- medida de la probabilidad de que una hipótesis nula sea cierta
- valor entre 0 y 1
- si `*p-valor* < 0.05` ✗ Rechazamos la hipótesis nula.
- si `*p-valor* > 0.05` ✓ Aceptamos la hipótesis nula.

Error Tipo I:

- rechazar la hipótesis nula cuando es verdadera

Error Tipo II:

- aceptar la hipótesis nula cuando es falsa

Tests estadísticos

Normalidad

- la variable respuesta tiene que tener una distribución normal para poder crear un modelo de regresión lineal

Visualmente:

- histograma o distribución
- grafico de cuantiles teóricos (Q-Q)

más alineados están los puntos entorno a la recta, más normales serán nuestros datos

```
import statsmodels.api as sm
```

```
sm.qqplot(datos, line ='45')
```

Metodos analiticos:

Asimetría

- distribuciones asimétricas positivas: media > mediana y moda
- distribuciones asimétricas negativas: media < mediana y moda

```
from scipy.stats import skew
```

`skew(datos_normales)` método de scipy que calcula el sesgo

`df['columna'].skew()` método de pandas que calcula el sesgo

Curtosis

- leptocurtosis: valor de curtosis mayor que 0 (pico alto)
- mesocurtosis: valor de curtosis igual a 0 (pico medio)
- platocurtosis: valor de curtosis menor que 0 (plana)

```
from scipy.stats import kurtosistest
```

`kurtosistest(datos)` devuelve un p-valor

- `p-valor del test > 0.05`: datos normales ✓
- `p-valor del test < 0.05`: datos NO normales

Test de Shapiro-Wilk

- para muestras < 5000
- hipótesis nula: distribución normal

```
from scipy import stats
```

```
stats.shapiro(df["datos"])
```

- `p-valor del test > 0.05`: datos normales ✓
- `p-valor del test < 0.05`: datos NO normales

Test de Kolmogorov-Smirnov

- para muestras > 5000
- hipótesis nula: distribución normal

```
from scipy import kstest
```

```
kstest(df["datos"], 'norm')
```

- `p-valor del test > 0.05`: datos normales ✓
- `p-valor del test < p-valor (alfa) 0.05`: datos NO normales

Tests estadísticos

Independencia entre variables predictoras

- las variables predictoras tienen que ser independientes para poder crear un modelo de regresión lineal

Variables numéricas: Correlaciones

- `pairplot`
- `sns.pairplot(df)`
- covarianza
- `df_numéricas.cov()`
- correlación de Pearson (relación lineal)
- `df_numéricas.corr()`
- correlación de Spearman (relación no lineal)
- `df_numéricas.corr(method = 'spearman')`
- correlación de Kendall (datos numéricos pero categóricos y ordinales)
- `df_numéricas.corr(method = 'kendall')`

Variables categóricas: Chi-cuadrado

- V-Cramer: varía entre 0 y 1
 - más cerca a 1 más dependientes
 - `resultado < 0,7 para hacer ML` ✓

```
import researchpy as rp
```

```
crosstab, test_results, expected = rp.crosstab
```

```
(df["col1"], df["col2"], test= "chi-square", expected_freqs= True, prop= "cell")
```

`test_results` devuelve los resultados del test en un dataframe

Homocedasticidad (homogeneidad de varianzas)

- las variables predictoras tienen que tener homogeneidad de varianzas en comparación con la variable respuesta

Visualmente:

- `violinplot`
- `boxplot`

- `regplot` (columnas numéricas vs variable respuesta)

Metodos analiticos:

- test de Levene (más robusto ante falta de normalidad) o Bartlett

```
from scipy import stats
```

```
from scipy.stats import levene
```

Variables categóricas:

- hay que crear un dataframe para cada valor único de las columnas categóricas

```
df_valor1 = df[df['col1'] == 'valor1']['col_VR']
```

```
df_valor2 = df[df['col1'] == 'valor2']['col_VR']
```

```
levene_test = stats.levene(df_valor1, df_valor2, center='median')
```

```
bartlett_test = stats.bartlett(df_valor1, df_valor2, center='median')
```

Variables numéricas:

- hay que crear un dataframe de las columnas numéricas sin la variable respuesta

```
for col in df_numericas.columns:
```

```
    statistic, p_val = levene(df[col], df['col_VR'], center='median')
```

```
    resultados[col] = p_val
```

devuelve los p-valores en un diccionario

- `p-valor del test > 0.05`: varianzas iguales, `homocedasticidad` ✓

- `p-valor del test < 0.05`: varianzas diferentes, `heterocedasticidad`

Normalización

Método manual

```
df["col_norm"] = (df["col_VR"] - df["col_VR"].media()) / (df["col_VR"].max() - df["col_VR"].min())
```

Método logarítmica

no se puede hacer si algún valor sea 0

```
df["col_norm"] = df["col_VR"].apply(lambda x: np.log(x) if x > 0 else 0)
```

Método raiz cuadrada

```
import math
```

```
df["col_norm"] = df["col_VR"].apply(lambda x: math.sqrt(x))
```

Método stats.boxcox()

```
from scipy import stats
```

```
df["col_norm"], lambda ajustada = stats.boxcox(df["col_VR"])
```

Método MinMaxScaler

```
from sklearn.preprocessing import MinMaxScaler
```

```
modelo = MinMaxScaler()
```

```
modelo.fit(df["col_VR"])
```

```
datos_normalizados = modelo.transform(df["col_VR"])
```

```
df_datos_norm = pd.DataFrame(datos_normalizados, columns = ['col_norm'])
```

```
df['col_norm'] = df_datos_norm
```

Estandarización

Método manual

```
df["col_esta"] = (df ["col_VR"] - df ["col_VR"].media()) / (df ["col_VR"].std())
```

Sklearn StandardScaler

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
scaler.fit(df_num_sin_VR)
```

```
datos_estandarizados = scaler.transform
```

```
(df_num_sin_VR)
```

```
df_datos_esta = pd.DataFrame(datos_estandarizados, columns = df_num_sin_VR.columns)
```

Sklearn RobustScaler

```
from sklearn.preprocessing import RobustScaler
```

```
scaler = RobustScaler()
```

```
scaler.fit(df_num_sin_VR)
```

```
datos_estandarizados = scaler.transform
```

```
(df_num_sin_VR)
```

```
df_datos_esta = pd.DataFrame(datos_estandarizados, columns = df_num_sin_VR.columns)
```

| Machine Learning: Preparación | Regresión Lineal | |
|--|--|--|
| <div>ANOVA</div> <div><pre>import statsmodels.api as sm from statsmodels.formula.api import ols lm = ols('col_VR ~ col_VP1 + col_VP2 + col_VP3', data=df).fit()</pre>devuelve un dataframe de los resultados: df (degrees of freedom): número de observaciones en los datos que pueden variar libremente al estimar los parámetros estadísticos; para variables categóricas será el número de valores únicos menos 1; para variables numéricas será siempre 1 sum_sq: una medida de variación o desviación de la media; suma de los cuadrados de las diferencias con respecto a la media mean_sq: es el resultado de dividir la suma de cuadrados entre el número de grados de libertad. F: un test que se utiliza para evaluar la capacidad explicativa que tiene la variable predictora sobre la variación de la variable respuestae - PR(>F): si el p-valor < 0.05 es una variable significativa; que puede afectar a la VR lm.summary() devuelve una resumen de los resultados: coef: el coeficiente que representa los cambios medios en la variable respuesta para una unidad de cambio en la variable predictora mientras se mantienen constantes el resto de las VP; los signos nos indican si esta relación es positiva o negativa std err: el error estándar del coeficiente que se usa para medir la precisión de la estimación del coeficiente; cuanto menor sea el error estándar, más precisa será la estimación t: es el resultado de dividir el coeficiente entre su error estándar</div> | <div>1. separar los datos de las variables predictoras (x) de la variable respuesta (y) x = df.drop('col_VR', axis=1) y = df['col_VR'] 2. dividimos los datos en datos de entrenamiento y datos de test con train_test_split() x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 42) 3. Ajustamos el modelo lr = LinearRegression(n_jobs=-1) lr.fit(x_train, y_train) 4. Hacemos las predicciones y_predict_train = lr.predict(x_train) y_predict_test = lr.predict(x_test) 5. Guardamos los resultados en dataframes y los concatenamos train_df = pd.DataFrame({'Real': y_train, 'Predicted': y_predict_train, 'Set': ['Train']*len(y_train)}) test_df = pd.DataFrame({'Real': y_test, 'Predicted': y_predict_test, 'Set': ['Test']*len(y_test)}) resultados = pd.concat([train_df,test_df], axis = 0) 6. creamos una columna de los residuos: la diferencia entre los valores observados y los de la predicción resultados['residuos'] = resultados['Real'] - resultados['Predicted']</div> <div>Cross-validation cv_scores = cross_val_score(estimator = LinearRegression(), X = X, y = y, scoring = 'neg_root_mean_squared_error', cv = 10) cv_scores.mean() calcula la media de los resultados de validación de una métrica cv_scores = cross_validate(estimator = LinearRegression(), X = X, y = y, scoring = 'r2', 'neg_root_mean_squared_error', cv = 10) cv_scores["test_r2"].mean() cv_scores["test_neg_root_mean_squared_error"].mean() calcula las medias de los resultados de validación de múltiples métricas</div> <div>Métricas R2: una medida estadística que representa la proporción de la varianza que puede ser explicada por las variables independientes (o predictoras) del modelo de regresión r2_score(y_train,y_predict_train) r2_score(y_test,y_predict_test) MAE (Mean absolute error): una medida de la diferencia entre los valores predichos frente a los reales. A menor MAE, mejor es capaz de ajustar los datos del modelo que hemos creado. mean_absolute_error(y_train,y_predict_train) mean_absolute_error(y_test,y_predict_test) MSE (Mean Squared Error): mide el promedio(media) de los errores al cuadrado. A menor MSE, mejor es capaz de ajustar los datos del modelo que hemos creado. mean_squared_error(y_train,y_predict_train) mean_squared_error(y_test,y_predict_test) RMSE (Root Mean Squared Error): nos muestra la distancia promedio entre los valores predichos y los valores reales del dataset. A menor RMSE, mejor es capaz de ajustarse el modelo obtenido. np.sqrt(mean_squared_error(y_train,y_predict_train)) np.sqrt(mean_squared_error(y_test,y_predict_test))</div> | |
| <div>Variables categóricas</div> <div>Ordinaria: no requiere números pero sí consta de un orden o un puesto; diferencias de medianas entre categorías Nominal: variable que no es representada por números, no tiene algún tipo de orden, y por lo tanto es matemáticamente menos precisa; no habrá grandes diferencias de medianas entre categorías Binaria: dos posibilidades; puede tener orden o no *podemos sacar un boxplot con la VR para comparar medianas*</div> <div>Variables sin orden:</div> <div>One-Hot Encoding crea una columna nueva por valor único, asignando unos y zeros según los valores que corresponden from sklearn.preprocessing import OneHotEncoder oh = OneHotEncoder() df_transformados = oh.fit_transform(df[['columna']]) oh_df = pd.DataFrame(df_transformados.toarray()) oh_df.columns = oh.get_feature_names_out() df_final = pd.concat([df, oh_df], axis=1) get_dummies</div> <div>Variables que tienen orden:</div> <div>Label Encoding asigna un número a cada valor único de una variable from sklearn.preprocessing import LabelEncoder le = LabelEncoder() df['col_VR_le'] = le.fit_transform(df[col_VR]) map() asigna el valor que queramos según el mapa que creamos df['col_VR_map'] = df[col_VR].map(diccionario) Ordinal-Encoding asignamos etiquetas basadas en un orden o jerarquía from sklearn.preprocessing import OrdinalEncoder</div> | | |