

Python: Los básicos

Variables ampliadas por text (CONCATENATION)

Para encadenar texto

```
categorial = "verde"
color_detalle = categorial + ' ' + 'oscuro'
```

```
print(categorial + ' oscuro')
print(categorial, '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 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 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)

Funciones y Clases; Librerías

Funciones

Definir una funcion:

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

Lllamar 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]
```

Lllamar una funcion con *args:

```
nombre_funcion(argumento, argumento, argumento, ...)  
o  
nombre_funcion(*[lista_o_tupla_de_args])
```

Lllamar 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_hereditado1, ...)
```

```
    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 numérico
\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 import 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 - sobrescribe
x – exclusive creation, sólo crearlo si no existe todavía
a – appending, añadir texto al archivo sin manipular el texto que ya habia
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

MySQL Connector/Python

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: Pandas	DataFrames	Metodos de exploracion	Tipos de datos	Valores nulos
Series: estructuras en una dimension	Crear DataFrames <code>df = pd.DataFrame(data, index, columns)</code> data : NumPy Array, diccionario, lista de diccionarios index : indice que por defecto se asigna como 0-(n-1), n siendo el número de filas; index = [lista] para asignar "etiquetas" (nombres de filas) column : nombre de las columnas; por defecto 0-(n-1); columns =[lista] para poner mas nombres	df.head(n) devuelve las primeras n lineas del dataframe df.tail(n) devuelve las últimas n lineas del dataframe 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 de las columnas numéricas df.info() devuelve un resumen sobre el no. de columnas, nombres de columnas, numero de valores no nulos y los tipos de datos <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 df.duplicated().sum() devuelve el numero de filas duplicadas	Tipos de datos en Pandas: - object - int64 - float64 - datetime, timedelta[ns] - category - bool df.dtypes devuelve el tipo de datos que hay en cada columna df.tipo = df.select_dtypes(include = "tipo") 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 copy = True devuelve una copia copy = False *cuidado: los cambios en los valores pueden propagarse a otros objetos pandas* errors = ignore omite excepciones; en caso de error devuelve el objeto original errors = raise permite que se generen excepciones	Identificar nulos 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 una serie con el número de valores nulos por columnas df.% nulos = ((df.isnull().sum() / df.shape[0] * 100).reset_index()) df.% nulos.columns = ['columna', '% nulos'] crea un dataframe de los porcentajes de los valores nulos
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	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	df.drop_duplicates(inplace = True, ignore_index=True) elimina filas duplicadas; ignore_index para no tener el indice en cuenta	pd.options.display.max_columns = None ejecutar antes del df.head() para poder ver todas las columnas	Eliminar nulos df.dropna(inplace = True, axis=b, subset=[lista_de_columnas], how=) quitar nulos how = 'any' 'all' por defecto 'any': si hay algun valor NA, se elimina la fila o columna; all: si todos los valores son NA, se elimina la fila o columna subset una columna o lista de columnas
Acceder a informacion de una serie serie.index devuelve los indices serie.values devuelve los valores serie.shape devuelve la forma (no. filas) serie.size devuelve el tamaño serie.dtypes devuelve el tipo de dato	Carga de datos df = pd.read_csv("ruta/nombre_archivo.csv") crear un dataframe de un archivo de Comma Separated Values df = pd.read_csv("ruta/nombre_archivo", sep= ";") crear un dataframe de un csv si el separador es ; df = pd.read_csv("ruta/nombre_archivo", index_col= 0) crear un dataframe de un csv si el archivo ya tiene una columna indice	Eliminar filas duplicadas df.drop_duplicates(inplace = True, ignore_index=True) elimina filas duplicadas; ignore_index para no tener el indice en cuenta	pd.set_option("display.precision", 2)	Tipos de nulos np.nan significa "not a number"; es un tipo numérico None valores nulos en columnas tipo string NaT valores nulos tipo datetime valores texto: "n/a", "NaN", "nan", "null" strings que normalmente se convierten automaticamente a np.nan 99999 o 00000 integers que se pueden convertir a nulos
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	DataFrames: carga de datos	Metodos de estadistica	Outliers	Reemplazar nulos df = pd.read_csv('archivo.csv', na_values = ['n/a']) .fillna(np.nan) reemplaza los strings 'n/a' con np.nan al cargar el dataframe
serie["etiqueta"] devuelve el valor de los elementos en indices i y j	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['columna'].mean() mode() median() var() std() calcula la media/moda/mediana/variación/desviación estándar de los valores de una columna df['columna1'].corr(df['columna2']) calcula la correlacion entre dos variables matriz_correlacion = df.corr() crea una matriz mostrando las correlaciones entre todos los variables df.crosstab = pd.crosstab(df['columna1'], df['columna2'], normalize = True, margins = True) normalize muestra los valores en porcentajes (por uno) margins muestra los totales y subtotales media_ponderada = np.average(df['columna'], weights = w) calcula la media ponderada según los pesos percentil_n = np.percentile(df['columna'], n) saca el valor en el percentil n q3, q1 = np.percentile(df["columna"], [75, 25]) saca los tercer y primer cuartiles	Calcular tres desviaciones estandares: media = df.column.mean() desviacion = df.column.std() lcb = media - desviacion * 3 ucb = media + desviacion * 3	df.fillna(df[value=n, axis=b, inplace=True]) reemplazar todos los NaN del dataframe con el valor que especifiquemos df['columna'].fillna(df['columna'].median, axis=b, inplace=True) reemplazar los nulos de una columna por la mediana de esa columna value=n por defecto NaN; es el valor por el que queremos reemplazar los valores nulos que puede ser un escalar, diccionario, serie o dataframe axis por defecto 0 (filas) df.replace(valor_nulo, valor_nuevo, inplace=True, regex=False) reemplazar los nulos por el valor nuevo
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 <u>mayor</u> que serie2 y devuelve True o False serie1.le(serie2) compara si serie1 es <u>menor</u> que serie2 y devuelve True o False	df = pd.read_clipboard(sep='\\t') crear un dataframe de datos en forma de dataframe en el clipboard; el separador podria ser \n ; , etc.	Sidetable: frecuencias de datos	Eliminar Outliers outlier_step = 1.5 * IQR calcular outlier step outliers_data = df[(df['columna'] < Q1 - outlier_step) (df['columna'] > Q3 + outlier_step)] identificar datos fuera del rango del maximo hasta el minimo lista_outliers_index = list(outliers_data.index) crear una lista de los indices de las filas con outliers	Imputacion de nulos from sklearn.impute import SimpleImputer imputer = SimpleImputer(strategy='mean', missing_values = np.nan) inicia la instancia del metodo, especificando que queremos reemplazar los nulos por la media imputer = imputer.fit(df['columna1']) aplicamos el imputer df['media_columna1'] = imputer.transform(df[['price']]) rellena los valores nulos segun como hemos especificado from sklearn.experimental import enable_iterative_imputer from sklearn.impute import IterativeImputer imputer = IterativeImputer(n_nearest_features=n, imputation_order='ascending') crea la instancia
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)	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	df.stb.freq(['columna']) devuelve un dataframe con informacion sobre la frecuencia de ocurrencia de cada categoría de un variable categorica parametros: thresh = n limita los valores mostrados a los más frecuentes hasta un umbral de n% cumulative y agrupando los restantes bajo la etiqueta "other" other_label = 'etiqueta' cambia la etiqueta 'other' value = 'columna' ordena los resultados por la columna especificada df.stb.freq(['columna1', 'columna2']) combina dos columnas y devuelve las frecuencias de las subcategorias	valores = dicc_indices.values() sacar todos los valores e.g. todos los indices valores = {indice for sublista in valores for indice in sublista} set comprehension para eliminar duplicados df_sin_outliers = df.drop(df.index[list (valores)]) crear nuevo dataframe sin outliers	df.replace(valor_nulo, valor_nuevo, inplace=True, regex=False) reemplazar los nulos por el valor nuevo
	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	df.stb.missing(['columna']) devuelve informacion sobre la frecuencia de datos nulos	Reemplazar Outliers for k, v in dicc_indices.items(): media = df[k].mean() for i in v: df.loc[i,k] = media reemplazar outliers por la media	from sklearn.impute import KNNImputer imputerKNN = KNNImputer(n_neighbors=5) crea la instancia imputerKNN.fit(df_numericas)
	ExcelWriter with pd.ExcelWriter("ruta/archivo.ext") as writer: df.to_Excel(writer, nombre_hoja = 'nombre') guardar un dataframe en una hoja de Excel			df_knn_imp = pd.DataFrame(imputerKNN.transform(df_numericas), columns = numericas.columns) crea un dataframe de los datos transformados; metemos estas columnas en el dataframe original

Matplotlib y Seaborn

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

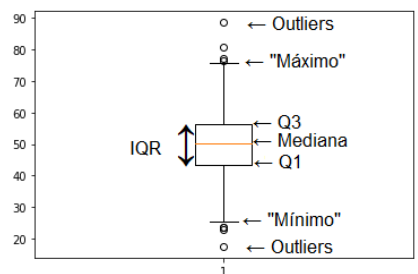
```
plt.hist(x = df['columna1'], bins = n)
```

 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 categorías); 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])
```

Crear subplots en un for loop

```
fig, axes = plt.subplots(numero_filas, numero_columnas, figsize = (n, m))
```

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

Títulos

```
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

Leyendas

```
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', 'Pastell', 'Pastell_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"	Círculo	"h"	Hexágono 1
"v"	Triángulo abajo	"H"	Hexágono 2
"^"	Triángulo arriba	"+"	Más
"<"	Triángulo izquierda	"x"	x
">"	Triángulo derecha	"X"	x (relleno)
"8"	Octágono	"D"	Diamante
"s"	Cuadrado	"d"	Diamante fino
"p"	Pentágono		

NumPy (Numerical Python)
Crear arrays
<div><div>Crear arrays de listas</div><div><div><div><div><div></div></div><div><div></div></div></div><div><div><div>array = np.array(lista, dtype= tipo)</div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div><div><div><div>array = np.array([lista1, lista2])</div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div><div><div><div>array = np.array([listadelistas1, listadelistas2])</div></div><div><div></div></div></div></div><div>crea un array unidimensional de una lista</div><div>crea un array bidimensional de dos listas</div><div>crea un array bidimensional de dos listas</div></div></div>

Crear otros tipos de arrays

array = np.arange(valor_inicio, valor_final, saltos)

array = np.ones(z,y,x)

array2 = np.ones_like(array1)

array = np.zeros(z,y,x)

array2 = np.zeros_like(array1)

array = np.empty((z,y,x), tipo)

array2 = np.empty_like(array1)

crea un array usando el formato [start:stop:step]

crea un array de todo unos de la forma especificada

crea un array de todo unos de la forma basada en otra array

crea un array de todo zeros de la forma especificada

crea un array de todo zeros de la forma basada en otra array

crea un array vacio con datos por defecto tipo float

crea un array vacia con la forma basada en otra array

crea un array con unos en diagonal empezando en la posicion k

crea una matriz de identidad con ceros en filas y unos en la diagonal, de forma cuadrada

Indices, Subsets, Metodos de Arrays
Indices de arrays
<div><div><div><div><div></div></div><div><div></div></div></div><div><div><div>array[i]</div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div><div><div><div>array[i, j] o array[i][j]</div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div><div><div><div>array[:,n]</div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div><div><div><div>array[h, i, j] o array[h][i][j]</div></div><div><div></div></div></div><div><div><div>array[h][i][j] = n</div></div><div><div></div></div></div></div><div>devuelve la indice i; las indices de los arrays unidimensionales funcionan igual que las listas</div><div>devuelve el elemento de la columna j de la fila i</div><div>seleccionar todas las filas y las columnas hasta n-1</div><div>devuelve el elemento de la columna j de la fila i del array h</div><div>cambiar el valor del elemento en esta posicion al valor n</div></div>

Subsets

array > n

array[array > n]

array[(array > n) & (array < m)]

devuelve la forma del array con True o False según si el elemento cumple con la condición o no

devuelve un subset: todos los valores que cumplen la condición en una lista dentro de un array

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()

np.transpose(array_bidimensional)

np.transpose(array_multidimensional)

np.transpose(array_multidimensional, (z,y,x))

array = np.arange(n).reshape((y,x))

array = np.reshape(array, (z,y,x))

crea un a copia del array

cambia los filas del array a columnas y las columnas a filas

cambia el número de columnas al número de arrays y viceversa; el número de filas no cambia

hace la transposicion segun lo que especificemos usando las posiciones de la tupla (0,1,2) de la forma original

crea un array usando reshape para definir la forma

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)

np.sort(array, axis = 0)

np.sort(-array)

np.round(array, decimals = x)

np.round(array, decimals = x)

np.where(array > x)

devuelve un array con los valores de cada fila ordenados en orden ascendente por defecto

devuelve un array con los valores de cada columna ordenados en orden ascendente

devuelve un array con los valores de cada fila ordenados en orden descendente

devuelve un array con los valores del array redondeados a x decimales

devuelve un array con los valores del array redondeados a x decimales

devuelve los indices de los valores que cumplan la condición, por fila y columna

Operaciones con arrays

np.add(array1, array2)

np.subtract(array1, array2)

np.multiply(array1, array2)

np.divide(array1, array2)

suma dos arrays

resta el array2 del array1

multiplica dos arrays

divide el array1 por el array2

array + n, n * array, etc. - operadores algebraicos

Operaciones estadísticas y matemáticas
Operaciones estadísticas y matemáticas
<div><div><div><div><div></div></div><div><div></div></div></div><div><div><div>El parametro axis en arrays bidimensionales:</div></div><div><div><div>axis = 0</div></div><div><div></div></div></div><div><div><div>axis = 1</div></div><div><div></div></div></div></div><div>columnas</div><div>filas</div><div>- si especificamos el axis, la operación devuelve el resultado por cada fila o columna.</div><div>Por ejemplo:</div><div>devuelve un array con la suma de cada fila</div></div></div> <div><div><div><div><div></div></div><div><div></div></div></div><div><div><div>El parametro axis en arrays multidimensionales:</div></div><div><div><div>axis = 0</div></div><div><div></div></div></div><div><div><div>axis = 1</div></div><div><div></div></div></div><div><div><div>axis = 2</div></div><div><div></div></div></div></div><div>dimensión</div><div>columnas</div><div>filas</div><div>- si especificamos el axis, la operación devuelve el resultado por cada dimensión, fila o columna.</div><div>Por ejemplo:</div><div>devuelve un array de una matriz con la suma de todas las matrices</div><div>devuelve un array donde las filas contienen las sumas de las columnas de cada matriz</div></div></div>

Operaciones con parámetro del axis:

np.sum(array_3D)

np.mean(array)

np.std(array)

np.var(array)

np.min(array)

np.max(array)

np.sum(array)

np.cumsum(array)

np.cumprod(array)

devuelve la suma de todos los elementos de los matrices

devuelve la media de todo el array

devuelve la desviación estándar de todo

devuelve la varianza de valores de todo

devuelve el valor mínimo del array

devuelve el valor máximo del array

devuelve la suma de los elementos del array

devuelve un array con la suma acumulada de los elementos a lo largo del array

devuelve un array con la multiplicación acumulada de los elementos a lo largo del array

Operaciones sin parámetro del axis:

np.sqrt(array)

np.exp(array)

np.mod(array1, array2)

np.mod(array1, n)

np.cos(array)

np.sin(array)

np.sin(array)

devuelve un array con la raíz cuadrada no negativa de cada elemento del array

devuelve un array con el exponencial de cada elemento del array

devuelve un array con el resto de la división entre dos arrays

devuelve un array con el resto de la división entre el array y el valor de n

devuelve un array con el coseno de cada elemento del array

devuelve un array con el seno de cada elemento del array

devuelve un array con la tangente de cada elemento del array

Operaciones de comparación en arrays bidimensionales

np.any(array > n)

np.any(array > n, axis = b)

np.all(array > n)

np.all(array > n, axis = b)

devuelve True o False segun si cualquier valor del array cumpla con la condicion

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

devuelve True o False segun si todos los valores del array cumpla con la condicion

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
<div><div><div><div><div></div></div><div><div></div></div></div><div><div><div>np.unique(array)</div></div><div><div></div></div></div><div><div><div>np.unique(array, return_index=True)</div></div><div><div></div></div></div><div><div><div>np.unique(array, return_inverse=True)</div></div><div><div></div></div></div><div><div><div>np.unique(array, return_counts=True)</div></div><div><div></div></div></div><div><div><div>np.unique(array, axis = b)</div></div><div><div></div></div></div></div><div>devuelve un array con los valores únicos del array ordenados</div><div>devuelve un array con los valores únicos del array ordenados y un array con la posición de la primera instancia de cada valor</div><div>devuelve un array con los valores únicos del array ordenados y un array con las posiciones de cada elemento de cada valor</div><div>devuelve un array con los valores únicos del array ordenados y un array con el número de veces que aparece cada valor</div><div>devuelve un array con los valores únicos ordenados de las filas o columnas</div></div>

Funciones para arrays unidimensionales

np.intersect1d(array1, array2)

np.intersect1d(array1, array2, return_indices=True)

np.union1d(array1, array2)

np.in1d(array1, array2)

np.setdiff1d(array1, array2)

np.setxor1d(array1, array2)

devuelve un array con los valores únicos de los elementos en común de dos arrays

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

devuelve un array ordenado con los elementos resultantes de unir dos arrays (valores únicos)

devuelve un array con True o False por cada elemento de array1 según si aparece el mismo valor en array2

devuelve un array ordenado con los valores únicos que están en array1 pero no en array2

devuelve un array ordenado con los valores únicos que NO están en común de los dos arrays

Estadística

Medidas de dispersión

Desviación respecto a la media

la diferencia en valor absoluto entre cada valor de los datos y su media aritmética

diferencias = df['columna'] - df['columna'].mean()

desviación_media = np.abs(diferencias)

Varianza

medida de dispersión; la variabilidad respecto a la media

df['columna'].var()

Desviación estándar o desviación típica

la raíz cuadrada de la varianza; cuanto mayor sea, mayor será la dispersión o variabilidad en nuestros datos

df['columna'].std()

Robustez

- cuanto más cantidad de datos, más robustos

1/n

donde n es el numero de registros

Coefficiente de variación

el cociente entre la desviación típica y la media; cuanto mayor sea, mayor será la dispersión en nuestros datos

df['columna'].std() / df['columna'].mean()

Percentiles

divide datos ordenados de menor a mayor en cien partes; muestra la proporción de datos por debajo de su valor

percentil_n = np.percentile(df['columna'], n)

saca el valor en el percentil n

Rangos intercuartílicos

medida de dispersión: diferencia entre cuartiles 75 y 25

q3, q1 = np.percentile(df["columna"], [75, 25])

saca los tercer y primer cuartiles

rango_intercuartílico = q3 - q1

Estadística
Tablas de frecuencias
<div><div><div><div><div></div></div><div><div></div></div></div><div><div><div>Frecuencias absolutas</div></div><div><div>el número de veces que se repite un número en un conjunto de datos</div></div><div><div><div>df = df.groupby('columna').count().reset_index()</div></div><div><div></div></div></div><div><div><div>Frecuencias relativas</div></div><div><div>las veces que se repite un número o categoría en un conjunto de datos respecto al total, en porcentajes</div></div><div><div><div>df_group_sin_str = df_group.drop('columna_str', axis=1)</div></div><div><div></div></div></div><div><div><div>frecuencia_relativa = df_group_sin_str / df.shape[0] * 100</div></div><div><div></div></div></div><div><div><div>columnas = df_group_sin_strings.columns</div></div><div><div></div></div></div><div><div><div>df_group[columnas] = frecuencia_relativa</div></div><div><div></div></div></div></div></div></div><div><div><div><div><div></div></div><div><div></div></div></div><div><div><div>Tablas de contingencia</div></div><div><div>tabla de frecuencias que cuenta todas las combinaciones posibles de cada pareja de valores de las columnas que estamos intentando comparar</div></div><div><div><div>df_crosstab = pd.crosstab(df['columna1'], df['columna2'], normalize = True, margins = True)</div></div><div><div></div></div></div><div><div><div>normalize</div></div><div><div>muestra los valores en porcentajes (por uno)</div></div><div><div><div>margins</div></div><div><div>muestra los totales y subtotales</div></div></div></div></div></div><div><div><div><div><div></div></div><div><div></div></div></div><div><div><div>Coefficiente de correlación de Pearson</div></div><div><div>- nos permite conocer la intensidad y dirección de la relación entre las dos variables</div></div><div><div>- coeficiente > 0: correlación positiva</div></div><div><div>- coeficiente < 0: correlación negativa</div></div><div><div>- coeficiente = 1 o -1: correlación total</div></div><div><div>- coeficiente = 0: no existe relación lineal</div></div><div><div><div>df['columna1'].corr(df['columna2'])</div></div><div><div>calcula la correlacion entre dos variables</div></div></div><div><div><div>matriz_correlacion = df.corr()</div></div><div><div>crea una matriz mostrando las correlaciones entre todos los variables</div></div></div><div><div><div>sns.heatmap(df.corr()[['column1', 'column2']], cmap = 'color_palette', annot = True, vmin = -1, vmax = 1)</div></div><div><div>crea una grafica heatmap de la matriz de correlaciones</div></div></div></div></div></div><div><div><div><div><div></div></div><div><div></div></div></div><div><div><div>Sesgos (skewness)</div></div><div><div>medida de la asimetría de la distribución de los valores de una variable alrededor de su valor medio</div></div><div><div>- valor de sesgo positivo: sesgado a la derecha</div></div><div><div>- valor de sesgo negativo: sesgado a la izquierda</div></div><div><div>- valor de sesgo igual a 0: valores simetricos</div></div><div><div><div>sns.displot(df['columna'], kde = True)</div></div><div><div>crea un histograma que muestra la distribution de los valores</div></div></div><div><div><div>import scipy.stats import skew</div></div><div><div>skew(df['columna'])</div></div><div><div>muestra el valor del sesgo de una variable</div></div></div></div></div></div><div><div><div><div><div></div></div><div><div></div></div></div><div><div><div>Intervalos de confianza</div></div><div><div>describe la variabilidad entre la medida obtenida en un estudio y la medida real de la población (el valor real)</div></div><div><div><div>import scipy.stats as st</div></div><div><div></div></div></div><div><div><div>st.t.interval(alpha = n, df = len(df['columna'])-1, loc = np.mean(df['columna']), scale = st.sem(df['columna']))</div></div><div><div>devuelve el rango de valores para lo cual hay un n% de probabilidad que un valor real cae en ese rango</div></div></div><div><div><div>alpha: porcentaje de confianza (p.ej. 90%, 95%, o 99%)</div></div><div><div></div></div></div><div><div><div>df: los datos</div></div><div><div></div></div></div><div><div><div>loc: la media</div></div><div><div></div></div></div><div><div><div>scale: la desviación estándar</div></div><div><div></div></div></div></div></div></div></div></div>

EDA y ETL	ETL: Extract, Transform, Load		Machine Learning: Preparación	Tests estadísticos	Normalización
<div>EDA: Análisis exploratorio de datos</div> <div>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.</div>	<div>Extraccion</div> <div>- obtener datos crudos y almacenarlos<ul style="list-style-type: none">- 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</div> <div>Transformación</div> <div>- procesar los datos, unificarlos, limpiarlos, validarlos, filtrarlos, etc.<ul style="list-style-type: none">- 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</div> <div>Carga</div> <div>- 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:<ul style="list-style-type: none">- Ficheros csv- Ficheros json- Bases de datos- Almacenes de datos (Data Warehouse)- Lagos de datos (Data Lakes)</div>		<div>Hipotesis Nula y Errores Tipo I y II</div> <div>Hipótesis nula (H0)<ul style="list-style-type: none">- en general es la afirmación contraria a la que queremos probar</div> <div>Hipótesis alternativa (H1)<ul style="list-style-type: none">- en general la afirmación que queremos comprobar</div> <div>p-valor<ul style="list-style-type: none">- 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.</div> <div>Error Tipo I:<ul style="list-style-type: none">- rechazar la hipótesis nula cuando es verdadera</div> <div>Error Tipo II:<ul style="list-style-type: none">- aceptar la hipótesis nula cuando es falsa</div>	<div>Independencia entre variables predictoras</div> <div>- las variables predictoras tienen que ser independientes para poder crear un modelo de regresión lineal</div> <div>Variables numéricas: Correlaciones</div> <div>- pairplot</div> <div>sns.pairplot(df)</div> <div>- covarianza</div> <div>df_numéricas.cov()</div> <div>- correlación de Pearson (relación lineal)</div> <div>df_numéricas.corr()</div> <div>- correlación de Spearman (relación no lineal)</div> <div>df_numéricas.corr(method = 'spearman')</div> <div>- correlación de Kendall (datos numéricos pero categóricos y ordinales)</div> <div>df_numéricas.corr(method = 'kendall')</div> <div>Variables categóricas: Chi-cuadrado</div> <div>- V-Cramer: varía entre 0 y 1<ul style="list-style-type: none">- más cerca a 1 más dependientes- resultado < 0,7 para hacer ML ✔</div> <div>import researchpy as rp</div> <div>crosstab, test_results, expected = rp.crosstab(df["col1"], df["col2"], test= "chi-square", expected_freqs= True, prop= "cell")</div> <div>test_results devuelve los resultados del test en un dataframe</div> <div>Homocedasticidad (homogeneidad de varianzas)</div> <div>- las variables predictoras tienen que tener homogeneidad de varianzas en comparación con la variable respuesta</div> <div>Visualmente:</div> <div>- violinplot</div> <div>- boxplot</div> <div>- regplot (columnas numéricas vs variable respuesta)</div> <div>Metodos analiticos:</div> <div>- test de Levene (más robusto ante falta de normalidad) o Bartlett</div> <div>from scipy import stats</div> <div>from scipy.stats import levene</div> <div>Variables categóricas:</div> <div>- hay que crear un dataframe para cada valor único de las columnas categóricas</div> <div>df_valor1 = df[df['col1'] == 'valor1']['col_VR']</div> <div>df_valor2 = df[df['col1'] == 'valor2']['col_VR']</div> <div>levene_test = stats.levene(df_valor1, df_valor2, center='median')</div> <div>bartlett_test = stats.bartlett(df_valor1, df_valor2, center='median')</div> <div>Variables numéricas:</div> <div>- hay que crear un dataframe de las columnas numéricas sin la variable respuesta</div> <div>for col in df_numericas.columns:</div> <div>statistic, p_val = levene(df[col], df['col_VR'], center='median')</div> <div>resultados[col] = p_val</div> <div>devuelve los p-valores en un diccionario</div> <div>- p-valor del test > 0.05: varianzas iguales, homocedasticidad ✔</div> <div>- p-valor del test < 0.05: varianzas diferentes, heterocedasticidad</div>	<div>Método manual</div> <div>- cogemos el valor que queremos normalizar y restamos la media de la columna, y dividimos el resultado por el maximo restado por el minimo de la columna</div> <div>df["col_norm"] = (df["col_VR"] - df["col_VR"].media()) / (df["col_VR"].max() - df["col_VR"].min())</div> <div>Método logarítmica</div> <div>*no se puede hacer si algún valor sea 0*</div> <div>df["col_norm"] = df["col_VR"].apply(lambda x: np.log(x) if x > 0 else 0)</div> <div>Método raiz cuadrada</div> <div>import math</div> <div>df["col_norm"] = df["col_VR"].apply(lambda x: math.sqrt(x))</div> <div>Método stats.boxcox()</div> <div>aplica una transformación logarítmica para los valores positivos y exponencial para valores negativos de nuestra columna</div> <div>from scipy import stats</div> <div>df["col_norm"], lambda ajustada = stats.boxcox(df["col_VR"])</div> <div>Método MinMaxScaler</div> <div>from sklearn.preprocessing import MinMaxScaler</div> <div>modelo = MinMaxScaler(feature_range=(0,1), copy=True)</div> <div>modelo.fit(df["col_VR"])</div> <div>datos_normalizados = modelo.transform(df["col_VR"])</div> <div>df_datos_norm = pd.DataFrame(datos_normalizados, columns = ['col_norm'])</div> <div>df['col_norm'] = df_datos_norm</div>
<div>1. Entender las variables</div> <div>- que variables temenos</div> <div>head(), .tail(), .describe(), .info(), .shape</div> <div>que tipos de datos</div> <div>.dtypes(), .info()</div> <div>- si temenos nulos o duplicados</div> <div>.isnull().sum()</div> <div>.duplicated().sum()</div> <div>- que valores unicos temenos</div> <div>.unique(), .value_counts()</div> <div>librería sidetable:</div> <div>stb.freq() devuelve el value_counts de variables categóricas, mas el porcentaje, cuenta cumulativa y porcentaje cumulativa</div> <div>stb.missing() tabla de cuenta de nulos y el porcentaje del total</div>					
<div>2. Limpiar el dataset</div> <div>- quitar duplicados (filas o columnas)</div> <div>- cambiar nombres de columnas</div> <div>- cambiar tipo de datos de columnas</div> <div>ordenar columnas</div> <div>- separar columna en dos con str.split()</div> <div>- crear intervalos con pd.cut()</div> <div>- crear porcentajes o ratios</div> <div>- decidir como tratar outliers: mantenerlos, eliminarlos, o reemplazarlos con la media, mediana o moda; o aplicar una imputation</div> <div>- decidir como tratar nulos:<ul style="list-style-type: none">- eliminar filas o columnas con nulos drop.na()- imputar valores perdidos:<ul style="list-style-type: none">- 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</div>					
<div>3. Analizar relaciones entre variables</div> <div>Analizar relaciones entre las variables</div> <div>- para encontrar patrones, relaciones o anomalias</div> <div>Relaciones entre dos variables numéricas:</div> <div>- scatterplot</div> <div>- regplot – scatterplot con línea de regresion</div> <div>- matriz de correlación y heatmap</div> <div>- joinplot – permite emparejar dos gráficas – una histograma con scatter o reg plot por ejemplo</div> <div>Relaciones entre dos variables categóricas:</div> <div>- countplot</div> <div>Relaciones entre variables numéricas y categóricas:</div> <div>- swarmplot</div> <div>- violinplot</div> <div>- pointplot</div> <div>- boxplot</div>	<div>APIs</div> <div>import requests</div> <div>libreria para realizar petitions HTTP a una URL, para hacer web scraping</div> <div>url = 'enlace'</div> <div>el enlace de la que queremos extraer datos</div> <div>header = {}</div> <div>opcional; contiene informacion sobre las peticiones realizadas (tipo de ficheros, credenciales)</div> <div>response = requests.get(url=url, header = header)</div> <div>pedimos a la API que nos de los datos</div> <div>variables = {'parametro1':'valor1', 'parametro2':'valor2'}</div> <div>response = request.get(url=url, params=variables)</div> <div>pedimos a la API que nos de los datos con los parametros segun el diccionario de parametros que le pasamos</div> <div>response.status_code</div> <div>devuelve el status de la peticion</div> <div>response.reason</div> <div>devuelve el motive de codigo de estado</div> <div>response.text</div> <div>devuelve los datos en formato string</div> <div>response.json()</div> <div>devuelve los datos en formato json</div> <div>df = pd.json_normalize(response.json)</div> <div>devuelve los datos en un dataframe</div> <div>Codigos de respuesta de HTTP</div> <div>1XX informa de una respuesta correcta</div> <div>2XX codigo de exito</div> <div>200 OK</div> <div>201 creado</div> <div>202 aceptado</div> <div>204 sin contenido</div> <div>3XX redireccion</div> <div>4XX error durante peticion</div> <div>401 peticion incorrecta</div> <div>402 sin autorizacion</div> <div>403 prohibido</div> <div>404 no encontrado</div> <div>5XX error del servidor</div> <div>501 error interno del servidor</div> <div>503 servicio no disponible</div>				

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

 Método manual - cogemos el valor que queremos normalizar y restamos la media de la columna, y dividimos el resultado por el maximo restado por el mínimo de la columna 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() aplica una transformación logarítmica para los valores positivos y exponencial para valores negativos de nuestra columna from scipy import stats df["col_norm"] = lambda ajustada = stats.boxcox(df["col_VR"]) |Método MinMaxScaler

from sklearn.preprocessing import MinMaxScaler

modelo = MinMaxScaler(feature_range=(0,1),

copy=True)

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

ANOVA

import statsmodels.api as sm

from statsmodels.formula.api import ols

lm = ols('col_VR ~ col_VP1 + col_VP2 + col_VP3',

data=df).fit()

devuelve un dataframe de los resultados:

df (degrees of freedom): para variables categóricas será el número de valores únicos menos 1; para variables numéricas será siempre 1

sum_sq: medida de variación/desviación de 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: representa los cambios medios en la VR para una unidad de cambio en la VP mientras se mantienen constantes el resto de las VP; los signos nos indican si esta relación es positiva o negativa

std err: cuanto menor sea el error estándar, más precisa será la estimación

t: es el resultado de dividir el coeficiente entre su error estándar

Machine Learning
Estandarización
- cambiar los valores de nuestras columnas de manera que la desviación estándar de la distribución sea igual a 1 y la media igual a 0; para que las VP sean comparables
Método manual
<pre>df["col_esta"] = (df ["col_VR"] - df ["col_VR"].media()) / (df ["col_VR"].std())</pre>
Sklearn StandardScaler
<pre>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)</pre>
Sklearn RobustScaler
<pre>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)</pre>
Encoding
Variables categóricas
Ordinaria: no requiere números pero si 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
Variables sin orden: creamos una columna nueva por valor único, asignando unos y zeros
One-Hot Encoding
<pre>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)</pre>
get_dummies
<pre>df_dum = pd.get_dummies(df['col'], prefix='prefijo', dtype=int) df[df_dum.columns] = df.dum df.drop('col', axis=1, inplace=True)</pre>
Variables que tienen orden:
Label Encoding asigna un número a cada valor único de una variable
<pre>from sklearn.preprocessing import LabelEncoder le = LabelEncoder() df['col_VR_le'] = le.fit_transform(df[col_VR'])</pre>
map() asigna el valor que queramos según el mapa que creamos
<pre>df['col_VR_map'] = df[col_VR'].map(diccionario)</pre>
Ordinal-Encoding asignamos etiquetas basadas en un orden o jerarquía
<pre>from sklearn.preprocessing import OrdinalEncoder</pre>

Regresión Lineal: Métricas
<pre>from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error</pre>
R2: representa la proporción de la varianza que puede ser explicada por las VP del modelo; mayor R2=mejor modelo
<pre>r2_score(y_train,y_predict_train) r2_score(y_test,y_predict_test)</pre>
MAE (Mean absolute error): medida de la diferencia entre los valores predichos vs los reales; menor MAE=mejor modelo
<pre>mean_absolute_error(y_train,y_predict_train) mean_absolute_error(y_test,y_predict_test)</pre>
MSE (Mean Squared Error): mide el promedio(media) de los errores al cuadrado; menor MSE=mejor modelo
<pre>mean_squared_error(y_train,y_predict_train) mean_squared_error(y_test,y_predict_test)</pre>
RMSE (Root Mean Squared Error): distancia promedio entre los valores predichos y los reales; menor RMSE=mejor modelo
<pre>np.sqrt(mean_squared_error(y_train,y_predict_train)) np.sqrt(mean_squared_error(y_test,y_predict_test))</pre>

Linear Regression: Modelo

1. separar los datos de las variables predictoras (x) de la variable respuesta (y)
<pre>X = df.drop('col_VR', axis=1) y = df['col_VR']</pre>
2. dividimos los datos en datos de entrenamiento y datos de test con train_test_split()
<pre>from sklearn.model_selection import train_test_split x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)</pre>
3. Ajustamos el modelo
<pre>from sklearn.linear_model import LinearRegression lr = LinearRegression(n_jobs=-1) lr.fit(x_train, y_train)</pre>
4. Hacemos las predicciones
<pre>y_predict_train = lr.predict(x_train) y_predict_test = lr.predict(x_test)</pre>
5. Guardamos los resultados en dataframes y los concatenamos
<pre>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)</pre>
6. creamos una columna de los residuos: la diferencia entre los valores observados y los de la predicción
<pre>resultados['residuos'] = resultados['Real'] - resultados['Predicted']</pre>
Cross-validation
<pre>from sklearn.model_selection import cross_val_score from sklearn.model_selection import cross_validate</pre>
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 CV 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

Regresión Logística: Métricas

Matriz de confusión

Matriz de confusión		Predicción	
		Positivo	Negativo
Realidad	Positivo	Verdadero positivo	Falso negativo
	Negativo	Falso positivo	Verdadero negativo

para crear un heatmap de una matriz de confusión:

```
from sklearn.metrics import confusion_matrix
mat_lr = confusion_matrix(y_test, y_pred_test)
plt.figure(figsize = (n,m))
sns.heatmap(mat_lr, square=True, annot=True=
plt.xlabel('valor predicho')
plt.ylabel('valor real')
plt.show()
```

Métricas

```
from sklearn.metrics import confusion_matrix,
accuracy_score, precision_score, recall_score,
f1_score , cohen_kappa_score,
roc_curve,roc_auc_score
```

Accuracy (exactitud): porcentaje de los valores predichos están bien predichos

```
accuracy_score(y_train,y_predict_train)
accuracy_score(y_test,y_predict_test)
```

Recall: porcentaje de casos positivos capturados

si preferimos FP, queremos recall alta

```
recall_score(y_train,y_predict_train)
recall_score(y_test,y_predict_test)
```

Precisión (sensibilidad): porcentaje de predicciones positivas correctas

si preferimos FN, queremos precisión alta

```
precision_score(y_train,y_predict_train)
presicion_score(y_test,y_predict_test)
```

Especificidad: porcentaje de los casos negativos capturados

F1: la media de la precisión y el recall

```
f1_score(y_train,y_predict_train)
f1_score(y_test,y_predict_test)
```

kappa: una medida de concordancia que se basa en comparar la concordancia observada en un conjunto de datos, respecto a la que podría ocurrir por mero azar

- <0 No acuerdo
- 0.0-0.2 Insignificante
- 0.2-0.4 Bajo
- 0.4-0.6 Moderado
- 0.6-0.8 Bueno
- 0.8-1.0 Muy bueno

```
cohen_kappa_score(y_train,y_predict_train)
cohen_kappa_score(y_test,y_predict_test)
```

curva ROC: forma gráfica de ver la kappa; la sensibilidad vs. la especificidad

AUC (área under curve): la área bajo la curva ROC; cuanto más cerca a 1, mejor será nuestro modelo clasificando los VP

Balanceo para Regresión Logística
Downsampling
ajustar la cantidad de datos de la categoría mayoritaria a la minoritaria
Método manual
<pre>df_minoritaria = df[df['col'] == valor_min] df_muestra = df[df['col'] == valor_max].sample (num_minoritarios, random_state = 42) df_balanceado = pd.concat([df_minoritaria, df_muestra],axis = 0)</pre>
Método RandomUnderSample
<pre>import imblearn X = df.drop('col_VR', axis=1) y = df['col_VR'] down_sampler = RandomUnderSampler() X_down, y_down = down_sampler.fit_resample(X,y) df_balanceado = pd.concat([X_down, y_down], axis = 1)</pre>
Método Tomek
<pre>x = df.drop('col_VR', axis=1) test_size = 0.2, random_state = 42) tomek_sampler = SMOTETomek() X_train_res, y_train_res = tomek_sampler.fit_resample(X_train, y_train)</pre>
Upsampling
ajustar la cantidad de datos de la categoría minoritaria a la mayoritaria
Método manual
<pre>df_mayoritaria = df[df['col'] == valor_may] df_muestra = df[df['col'] == valor_min].sample (num_mayoritarias, random_state = 42) df_balanceado = pd.concat([df_mayoritaria, df_muestra],axis = 0)</pre>
Método RandomOverSample
<pre>import imblearn X = df.drop('col_VR', axis=1) y = df['col_VR'] down_sampler = RandomUnderSampler() X_down, y_down = down_sampler.fit_resample(X,y) df_balanceado = pd.concat([X_down, y_down], axis = 1)</pre>
Logistic Regression: Modelo
seguir los mismos pasos como para la Regresión Lineal pero con LogisticRegression()
<pre>from sklearn.linear_model import LogisticRegression</pre>
Decision Tree: Modelo
<pre>from sklearn.model_selection import train_test_split from sklearn.ensemble import DecisionTreeRegressor from sklearn import tree</pre>
seguir los mismos pasos como para la Regresión Lineal pero con DecisionTreeRegressor() o DecisionTreeClassifier()
<pre>arbol = DecisionTreeRegressor(random_state=42)</pre>
Para dibujar el árbol:
<pre>fig = plt.figure(figsize = (10,6)) tree.plot_tree(arbol, feature_names = x_train.columns, filled = True) plt.show()</pre>

GridSearch y best_estimator_
Despues de hacer las predicciones de un modelo Decision Tree, examinamos lás métricas de los resultados:
- si temenos overfitting hay que reducir la profundidad del modelo
- si temenos underfitting hay que aumentar la profundidad del modelo
max_features = np.sqrt(len(x_train.columns))
podemos calcular el valor de max_features siendo la raíz cuadrada del número de variables predictoras
arbol.tree_.max_depth nos muestra el max depth usado por defecto, para poder ajustarlo; deberíamos usar la mitad como mucho
- GridSearch ejecuta todas las posibles combinaciones de hiperparámetros que le damos con el parámetro 'param' y best_estimator_ devuelve la major combinacion encontrado
1. Definimos un diccionario de los hiperparametros
<pre>param = {"max_depth": [n,m,l], "max_features": [a,b,c,d], "min_samples_split": [x,y,z], "min_samples_leaf": [r,s,t]} from sklearn.model_selection import GridSearchCV</pre>
2. Iniciamos el modelo con GridSearch
<pre>gs = GridSearchCV(estimator = DecisionTreeRegressor(), param_grid = param, cv=10, verbose=-1, return_train_score = True, scoring = "neg_mean_squared_error")</pre>
3. Ajustamos el modelo en el GridSearch
<pre>gs.fit(x_train, y_train)</pre>
4. Aplicamos el método de best_estimator_
<pre>mejor_modelo = gs.best_estimator_</pre>
devuelve la mejor combinación de hiperparámetros
5. Volvemos a sacar las predicciones
<pre>y_pred_test_dt2 = mejor_modelo.predict(x_test) y_pred_train_dt2 = mejor_modelo.predict(x_train)</pre>
Importancia de los predictores
<pre>importancia_predictores = pd.DataFrame({'predictor': x_train.columns, 'importancia': mejor_modelo.feature_importances_}) importancia_predictores.sort_values(by=["importancia"], ascending=False, inplace = True)</pre> crea un dataframe con la relativa importancia de cada VP
- para los variables categóricas nominales a los cuales se ha aplicado encoding, hay que sumar los resultados de las columnas divididas:
<pre>df_sum = importancia_predictores_esta.iloc[[n, m]]</pre>
<pre>importancia_predictores_esta.drop(df_sum.index, inplace = True)</pre>
<pre>importancia_predictores_esta.loc[n] = ["nombre_col", df_sum["importancia"].sum()]</pre>
Random Forest: Modelo
seguir los mismos pasos como para el Decision Tree pero con RandomForestRegressor() o RandomForestClassifier()
<pre>from sklearn.ensemble import RandomForestRegressor</pre>
- se puede usar los mismos hiperparámetros del best_estimator_ o volver a ejecutar el GridSearch