Python Cheat Sheet 1 Variables ampliadas por text

type(variable) devuelve: class 'float/int/str'

/ dividir

division)

// divider y redondear (modulus)

% resto de una division (floor

round(x) redondear número x

(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

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

restar multiplicar ** elevar

Operaciones Binarias == comprobar si valores coinciden

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

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

Metodos String

lista

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

por defecto, o especifica otro divisor en ()

string.replace("frase", "frase") remplaza la primera frase del string por el otro " ".join(string) une los elementos de una lista en una string con el separador espificado en " " list(string) convierte un variable string en una

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

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

len(lista) devuelve el no. de elementos

Listas [] Metodos no permanentes

lista = [] crea una lista vacia

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

lista.count() devuelve el no. de elementos que hay en la lista de un valor determinado en los() <mark>sorted(lista)</mark> ordenar una lista de menor a mayor

lista.copy() hacer una copia de la lista

Metodos con indices

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

Listas – Acciones Permanentes

[lista1, lista2] junta listas pero se mantienen como listas separadas lista1 + lista2 hace una lista mas larga

Ampliar una lista

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

string, integer o tuple) a la lista .extend()

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

.insert()

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

Ordenar una lista .sort()

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

Quitar elementos de una lista

.pop()

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

lista.remove(x)# quita el primer elemento de la lista con valor x lista.clear()# vacia la lista

del lista# borra la lista

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

variable = dict(x=y, m=n) crear un diccionario dicc.copv() crear una copia

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

Diccionarios { key : value , }

dict()

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

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

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

Diccionarios – Metodos

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

Ampliar un diccionario .update()

dicc.update({x:y})# para insertar nuevos elementos dicc["key"] = valor# para inserter un nuevo key o valor, o cambiar el valor de un key dicc. setdefault(x, y)# devuelve el value del key

Quitar elementos de un diccionario

dicc.pop(x)# elimina la key x (y lo devuelve)

Tuplas (,) inmutables, indexados

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

len(tupla) devuelve el no. de elementos

convertirla en una lista y luego a tupla*

tuple(dicc.values()) crear tuplas de los values

in/not in comprobar si hay un elemento *para cambiar el contenido de una tupla hay que listzip.sort() ordena las tuplas del zip por el primer elemento

Sets {} no permiten duplicados, no tienen orden

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

parejas de los elementos de las dos listas (mientras se

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

zip()

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

[] o {} o un variable tipo lista o set Quitar elementos de un set set.pop()# elimina un elemento al azar

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

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

set.remove(x)# elimina el elemento x

Operaciones con dos Sets

dos sets son diferentes

set2 estan en set1

los 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

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

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

input()

• permite obtener texto escrito por teclado del usuario

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

False) while x < 5: print("x es mayor que 5")

if x > y:

else:

elif x == v:

Sentencias de control

llevar condiciones nuevas

print("x es mayor que y")

print("x es igual que y")

print("x e y son iguales")

parará cuando la condición sea False

if ... elif ... else

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

if estableca una condición para que se ejecute el código que

else agrupa las condiciones que no se han cumplido; no puede

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

se pueden incluir condiciones con if... elif... else

pueden ser infinitos (si la condición no llega a ser

esta debajo del if. *tiene que estar indentado*

elif para chequear mas condiciones después de un if

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

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

List comprehension

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

try ... except

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

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

print("no funciona")

range()

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

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

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

x, o si no existe la key x, la crea y asigna el valor y por defecto

dicc.popitem()# elimina el ultimo par de key:value dicc.clear()# vacia el diccionario

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

tupla1 + tupla2 juntar tuplas tuple(lista) crear tuplas de una lista

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

tupla.index(x) devuelve el indice de x tupla.count(x) devuelve el no. de elementos con valor x en la tupla

input("el texto que quieres mostrar al usuario") · se puede guardar en un variable

metodos permanentes (cambia el variable, no devuelve nada)

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

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

Python Cheat Sheet 4			
Pandas			
Union de datos			
<pre>.concat() unir dataframes con columnas en comun df_union = pd.concat([df1, df2, df3], axis=b, join = 'inner/outer', ignore_index = True/False) parametros:</pre>			
axis = 0 une por columnas - los dataframes van uno encima del otro; las columnas tienen que ser de formatos compatible axis = 1 une por filas - los dataframes van uno al lado del otro; los datos deben ser relacionados para que tenga sentido join = 'inner' solo se quedan elementos que aparecen en todos los dataframes			
<pre>join = 'outer' se queda todo los datos de todos los dataframes ignore_index = True/False por defecto es False; si es True no usa las índices para la union (por ejemplo para union por el axis 0)</pre>			
<pre>.merge() unir las columnas de un dataframe a otro df_nuevo = df1.merge(df2, on = 'columna') inner merge</pre>			
<pre>df_nuevo = pd.merge(left = df1, right = df2, how='left', left_on =</pre>			
<pre>'columna_df1', right_on = 'columna_df2') left merge parametros:</pre>			
how = 'left' 'right' 'outer' 'inner' 'cross' on = columna [columna1, columna2, etc] si las columnas se llaman			
igual en los dos dataframes left on = columna df1 right on = columna df2 para especificar			
<pre>por donde hacer el merge suffixes = ['left', 'right'] por defecto nada, el sufijo que</pre>			
aparecera en columnas duplicadas			
<pre>.join() unir dataframes por los indices df_nuevo = df1.join(df2, on = 'columna', how = 'left') inner merge parametros:</pre>			
how = 'left' 'right' 'outer' 'inner' por defecto left on = columna la columna o indice por el que queremos hacer el union; tienen que tener el mismo nombre en los dos dataframes lsuffix = 'string' rsuffix = 'string' por defecto nada, el sufijo que aparecera en columnas duplicadas			
Group By			
<pre>df_groupby = df.groupby("columna_categoría") crea un objeto DataFrameGroupBy; agrupa los valores segun las categorías de los valores de la columna indicada (o múltiples columnas en una lista) df_groupby.ngroups devuelve el numero de grupos df_groupby.groups sevuelve un diccionario donde los keys son las categorías y los valores son listas de los índices de cada elemento en la categoría df_grupo1 = df_groupby.get_group("grupo1") devuelve un dataframe con los resultados de un grupo (la categoria indicada como grupo1) Cálculos con groupby: df_nuevo = df.groupby("columna_categoría").mean() devuelve un dataframe con la media de todas las columnas de valores numéricos, por categoría</pre>			
<pre>df_nuevo = df.groupby("columna_categoría")["columna1"].mean() devuelve un dataframe con la media de la columna especificada</pre>			
<pre>count() número de observaciones no nulas describe() resumen de los principales estadísticos sum() suma de todos los valores mean() media de los valores</pre>			
<pre>df_nuevo = df.groupby("columna_categoría", dropna = False) ["columna_valores"].agg([nombre_columna = 'estadistico1', nombre_columna2 = 'estadistico2']) añade columnas con los cálculos de los estadisticos especificados dropna = False para tener en cuenta los Nan en los cálculos (por defecto es True)</pre>			

Python Chast Shoot 1

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

Subsets: loc e iloc

de una fila

df.loc["etiqueta fila", "etiqueta columna"] devuelve el contenido de un campo en una columna

df.loc["etiqueta fila",:] devuelve los valores

df.loc[:,"etiqueta columna"] devuelve los

de todas las columnas de una fila

filas cuvas valores de la columna nombrada están en el iterable (una lista, serie, dataframe o diccionario) df filtrado= df[df["nombre columna"].str.contains (patron, regex = True, na = False)] extrae las filas cuvas valores de la columna nombrada contenienen el natron de regex 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 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 df[pd.notnull(df["nombre columna"])] devuelve las filas que no tiene valores nulos en la columna especificada Cambiar columnas lista columnas = df.columns.to list() crea una lista de los nombres de las columnas del dataframe df.set index(["nombre columna"], inplace = True) establece el índice utilizando uno o mas columnas; puede sustituir o ampliar un índice existente inplace = True los cambios sobreescriben sobre el df * cuando una columna se cambia a índice ya no es df.reset index(inplace = True) quitar una columna como indice para que vuelva a ser columna; crea un dataframe de una serie Renombrar columnas df.rename(columns = {"nombre columna": "nombre nueva"}, inplace = True) cambia los nombres de una o mas columnas ejemplo de dict comprehension para crear diccionario sobre las columnas existentes de un dataframe: diccionario = {col : col.upper() for col in df.rename(columns = diccionario, inplace = True) cambia los nombres de las columnas según el diccionario Eliminar columnas df.drop(columns = ["columna1", "columna2"], axis = b, inplace=True) eliminar una o mas columnas o filas segun lo que especificamos Reordenar columnas df = df.reindex(columns = lista reordenada) cambia el orden de las columnas del dataframe segun el orden de la lista reordenada

Filtrados de datos

df filtrado =

Metodos de pandas de filtrar

df[df["nombre columna"].isin(iterable)] extrae las

Creacion de porcentaies def porcentaje(columna1, columna2): return (columna1 * 100) / columna2 df["columna %"] = df.apply(lambda df: porcentaje(df["columna1"], datos["columna2"]), axis = 1) df["nueva_columna"] = np.where(df["nombre_columna"] > n, "categoria if true", "categoria if false") crea una nueva columna basada en una condición df["nueva columna"] = np.select(lista de condiciones, lista de opciones) crea una nueva columna con los valores basados en multiples condiciones df["columna nueva"] = pd.cut(x = df["nombre columna"], bins = [n,m,1..], labels = ['a', 'b', 'c']) separa los elementos de un dataframe en diferentes intervalos (n-m, m-l, etc), creando una columna nueva que indica en cual intervalo cae el valor; con labels se puede asignar un string a cada intervalo df["nueva_columna"] = (df["etiqueta_columna"] + x) crea una nueva columna basada en otra df = df.assign(nueva columna= df["etiqueta columna] + x) crea una nueva basada en otra df = df.assign(nueva columna= [lista valores]) crea una nueva columna de una lista de valores *tiene que ser de la misma longitud como el número de filas del dataframe* df.insert(indice_nueva_columna, "nombre_columna", valores) crea una nueva columna en la indice indicada allow_duplicates = True parametro cuando queremos permitir columnas duplicadas (por defecto es False) Apply apply() toma una función como argumento y la aplica a lo largo de un eje del DataFrame df['columna nueva'] = df['col 1'].apply(función) crea una columna nueva con los valores de otra columna transformados según la función indicada df['columna nueva'] = df['col 1'].apply(lambda x: x.método() if x > 1)crea una columna nueva con los valores de otra columna transformados según la lambda indicada 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) 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 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. apply() con datetime df['columna fecha'] = df['columna fecha'] .apply(pd.to datetime) cambia una columna de datos tipo fecha en el formato datetime def sacar año(x): return x.strftime("%Y") df['columna_año'] = (df['columna_fecha'] .apply (sacar_año) crea una columna nueva del año solo usando un

método de la librería datetime; ("%B") para meses

Crear columnas

/ df["columna2"], axis = 1)

Creacion de ratios

Reemplazar valores basados en indices y condiciones: df["columna?ratio"] = df.apply(lambda df: df["columna1"]

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

Cambiar valores

fig. ax = plt.subplots(numero filas, numero columnas) Line plot **Titulos** Colores Matplotlib crear una figura con multiples graficas; fig es la fig = sns.lineplot(x = 'columna1', y = 'columna2', data = plt.title(label = "titulo")
asignar un titulo a color = "color" establece el color de la grafica figura y ax es un array con subplots como elementos df, ci = None) crea una gráfica lineal donde los ejes son: la gráfica facecolor = "color" establece el color del relleno columna1 - x, columna2 - y se establece como es cada grafica con los indices: Gráficas ci = None para que no muestra el intervalo de confianza de edgecolor = "color" establece el color de los bordes ax[indice].tipo grafica(detalles de la grafica) los datos Colores en Scatter Plots: ax[indice].set title('titulo') import matplotlib.pyplot as plt Eies hue = columna opcional; muestra lineas en diferentes c= df['columna'].map(diccionario) ax[indice].set xlabel('xlabel') colores por categorias segun una variable plt.xlabel("etiqueta_eje_x") asignar nombre al eje x diccionario = {"valor1": "color1", "valor1": "color1"} plt.rcParams["figure.figsize"] = (10,8) ax[indice].set vlabel('vlabel') plt.ylabel("etiqueta eje y") asignar nombre al eje y Scatter plot plt.figure(figsize = (n,m)) inicia una grafica <u>lista de colores</u> fig = sns.scatterplot(x = 'columna1', y = 'columna2', data ax[indice].set xlim(min, max plt.xlim([n,m] establece el rango del eje x; donde n es dibujando el marco de la figura; n es la anchura y Paletas Seaborn: = df, hue = 'columna') crea una gráfica de dispersión ax[indice].set ylim(min, max) el mínimo v m es el máximo m es la altura, en pulgadas plt.show() muestra la figura plt.ylim([n,m]) establece el rango del eje y; donde n Accent', 'Accent r', 'Blues', 'Blues r', 'BrBG', 'BrBG r', Swarm plot ax[indice].set xticklabels(labels = df['column'], fig = sns.swarmplot(x = 'columna1', y = 'columna2', data = rotation = n) para cambiar los nombres y/o la rotacion es el mínimo y m es el máximo 'BuGn', 'BuGn r', 'BuPu', 'BuPu r', 'CMRmap', 'CMRmap r', 'Dark2', 'Dark2_r', 'GnBu', 'GnBu r', 'Greens'. df, hue = 'columna') crea una gráfica de dispersión donde de las etiquetas de los valores en los ejes Gráficas básicas 'Greens r', 'Greys', 'Greys r', 'OrRd', 'OrRd r', los marcadores no se solapan fig.set(xlabel = 'etiqueta eje x', ylabel = Bar plot 'Oranges', 'Oranges_r', 'PRGn', 'PRGn_r', 'Paired', Count plot 'etiqueta eje y') asignar nombre a los ejes Crear subplots en un for loop 'Paired_r', 'Pastel1', 'Pastel1 r', 'Pastel2'. plt.bar(df["columna1"], df["columna2"]) crea un fig = sns.countplot(x = 'columna1', data = df, hue = fig, axes = plt.subplots(numero filas, numero columnas, fig.set title('titulo') asignar un titulo a la gráfica diagrama de barras donde los ejes son: columna1 -'Pastel2 r', 'PiYG', 'PiYG r', 'PuBu', 'PuBuGn', 'columna') crea una gráfica de barras con la cuenta de una figsize = (n, m)PuBuGn r', 'PuBu r', 'PuOr', 'PuOr r', 'PuRd', 'PuRd r', x. columna2 - v variable categórica; se puede especificar solo una axes = axes.flatten(Purples', 'Purples r', 'RdBu', 'RdBu r', 'RdGy', fig.set_xlabel(xlabel = "etiqueta_eje_x", fontsize = n) Horizontal bar plot variable en la eje x o v. mas una variable opcional con RdGy r', 'RdPu', 'RdPu r', 'RdYlBu', 'RdYlBu r', for col in df.columns: plt.barh(df["columna1"], df["columna2"]) crea una fig.set_ylabel(ylabel = "etiqueta_eje_y", fontsize = n) RdYlGn', 'RdYlGn r', 'Reds', 'Reds r', 'Set1', 'Set1 r', diagramma de barras horizontales donde los ejes fig = sns.plot(x=col, data=df, ax=axes[i] Histogram Set2', 'Set2 r', 'Set3', 'Set3 r', 'Spectral', son: columna1 - x, columna2 - y fig = sns.histplot(x = 'columna1', data = df, hue = fig.set(xticks = [1, 2, 3]) 'Spectral r', 'Wistia', 'Wistia r', 'YlGn', 'YlGnBu' Stacked bar plot 'columna3', kde = True, bins = n) crea una histograma que 'YlGnBu r', 'YlGn r', 'YlOrBr', 'YlOrBr r', 'YlOrRd', fig.set(yticks = [1, 2, 3, 4, 5]) Usos de los tipos de gráficas plt.bar(x, y, label = 'etiqueta') muestra la frecuencias de una distribución de datos; donde 'YlOrRd_r', 'afmhot', 'afmhot_r', 'autumn', 'autumn_r', fig.set(xticklabels = ['0%','20%', '40%', '60%', '80%', plt.bar(x2, y2, bottom = y, label = 'etiqueta2') x es la variable de interés y n es el número de barras binary', 'binary r', 'bone', 'bone r', 'brg', 'brg r', '100%'1) crea una diagrama de barras apiladas para kde = True muestra una curva de la distribucion Datos categóricos bwr', 'bwr_r', 'cividis', 'cividis_r', 'cool', 'cool_r', fig.set(yticklabels = ['cat1', 'cat2', 'cat3']) visualizar dos variables juntas; y indica la barra 'coolwarm', 'coolwarm_r', 'copper', 'copper_r', 'crest', **Box Plot** de referencia 'crest r', 'cubehelix', 'cubehelix r', 'flag', 'flag r', fig = sns.boxplot(x = 'columna1', data = df, hue = muestra la relación entre una variable numérica y fig.set xticklabels(labels = [0, 500, 1000, 1500], flare', 'flare_r', 'gist_earth', 'gist_earth_r', 'columna') crea un diagrama de cajas; x es la variable de 'gist_gray', 'gist_gray_r', 'gist_heat', 'gist_heat_r', plt.scatter(df["columna1"], df["columna2"]) crea interés; por defecto se muestra con orientación horizontal - barplot si tienes una variable numérica gist_ncar', 'gist_ncar_r', 'gist_rainbow', una gráfica de dispersión donde los ejes son: - usar eje y para orientación vertical fig.set yticklabels(labels = fig.get yticklabels(), countplot para contar registros/filas por categoría gist_rainbow_r', 'gist_stern', 'gist_stern_r', columna1 - x, columna2 - y Catplot gist_yarg', 'gist_yarg_r', 'gnuplot', 'gnuplot2', fig = sns.catplot(x = 'columna1', y = 'columna2', data = Pie chart/quesitos 'gnuplot2_r', 'gnuplot_r', 'gray', 'gray_r', 'hot', Gráficas estadísticas df, hue = 'columna', kind = 'tipo') crea una gráfica que determinación de frecuencias Para poner etiquetas encima de las barras hot r', 'hsv', 'hsv r', 'icefire', 'icefire r', muestra la relacion entre una variable categorica y una Histogram for indice, valor in enumerate(df ["col"]): 'inferno', 'inferno_r', 'jet', 'jet_r', 'magma', variable numerica plt.hist(x = df['columna1'], bins = n) crea una magma_r', 'mako', 'mako_r', 'nipy_spectral', plt.text(valor+1, indice, valor, kind = 'box' | 'bar' | 'violín' | 'boxen' | 'point' por Datos numéricos histograma que muestra la frecuencias de una 'nipy_spectral_r', 'ocean', 'ocean_r', 'pink', 'pink r', horizontalalignment='left', fontsize= 16) defecto es strip plot distribución de datos: donde x es la variable de plasma', 'plasma r', 'prism', 'prism r', 'rainbow', interés y n es el número de barras Pairplot rainbow r', 'rocket', 'rocket r', 'seismic', 'seismic r', - tendencias/evolución de una o más variables numéricas fig = sns.pairplot(data = df, hue = 'columna', kind = order = df.sort values('columnay', ascending=False) spring', 'spring r', 'summer', 'summer r', 'tab10', (normalmente sobre un periódo de tiempo) plt.boxplot(x = df['columna1']) crea un diagrama de 'tipo') crea los histogramas y diagramas de dispersión de ['columnax'] 'tab10 r', 'tab20', 'tab20 r', 'tab20b', 'tab20b r', cajas para estudiar las caracteristicas de una todas las variables numéricas de las que disponga el tab20c', 'tab20c_r', 'terrain', 'terrain_r', 'turbo', sns.set(font scale=2) variable numerica: x es la variable de interés dataset con el que estemos trabajando; hue es opcional - distribución de una variable numérica turbo_r', 'twilight', 'twilight_r', 'twilight_shifted', plt.rcParams.update({'font.size': 22}) font size el mínimo es lo mismo que O1 - 1.5 * IOR kind = 'scatter' | 'kde' | 'hist' | 'reg' | 'point' por 'twilight shifted r', 'viridis', 'viridis r', 'vlag', general el máximo es lo mismo que Q3 + 1.5 * IQR 'vlag_r', 'winter', 'winter_r defecto es scatter -representación de las medidas de posición más usadas: Heatmap palette='light:nombre paleta'|'dark:nombre paleta' Outliers mediana, IQR, outliers Levendas sns.heatmap(df.corr(), cmap = 'color palette', annot = Scatterplot _ઁ ← "Máximo" True, vmin = -1, vmax = 1) crea un heatmap con una escala plt.legend(labels = ['label1', 'label2', etc) muestra Marcadores de colores que refleja los valores de correlacion muestra la relación entre dos variables numéricas la leyenda cuando mostramos la figura marker = 'tipo' establece el tipo de marcador; se usa con annot = True para que aparezcan los valores Regplot plt.legend(bbox_to_anchor = (1, 1) coloca la leyenda en plt.scatter y plt.plot relación con los ejes vmin/vmax establecen la escala de color - scatterplot con una línea de regresión '." Punto "P" Más (relleno) Swarmplot Regplot "*" Estrella ," Pixel fig = sns.regplot(x = 'columna1', y = 'columna2', data = **Ouitar bordes** "o" Circulo "h" Hexágono 1 - tipo de gráfica de dispersión para representar — ← "Mínimo" df, scatter_kws = {'color':'blue'}, line_kws = {'color'; "v" Triángulo abajo "H" Hexágono 2 variables categóricas; evita que se solapan los fig.spines[["top", "right"]].set visible(False) 'blue'})crea un scatterplot mas la línea de regresión; nos "^" Triángulo arriba "+" Más marcadores ← Outliers permite encontrar la mejor función de la recta que permite "<" Triángulo izquierda "x" x Violinplot Linea de tres desviaciones estandares: predecir el valor de una variable sabiendo los valores de "X" x (relleno) ">" Triángulo derecha Pie Chart fig.axvline(x=valor, c='color', label='valor' - para visualizar la distribución de los datos y su otra variable "8" Octágono "D" Diamante plt.pie(x, labels = categorias, radius = n) crea un fig.axvline(x=valor, c='color', label='valor') densidad de probabilidad "d" Diamante fino "s" Cuadrado Jointplot gráfico de sectores donde x es la variable de "p" Pentágono sns.jointplot(x = 'columna1', y = 'columna2', data = df, interés (debe esta agrupado por categorias); n es color = 'blue', kind = 'tipo') crea un scatterplot o - para representar múltiples relaciones entre dos el tamaño plt.grid() crea una cuadrícula al fondo de la figura; regplot con histogramas pegados en los lados para cada variables coge los parámetros: variable Violin Plot color = "color" plt.violinplot(x, showmedians = True, showmeans = - evaluar la correlación entre las variables en una True) crea un diagrama de violin donde x es la Exportar figuras linestyle = "solid" | "dashed" | "dashdot" | "dotted" matriz de correlación plt.savefig('nombre de la figura.extension') variable de interés y muestra la mediana y la media linewidth = n establece la anchura de la linea

Personalización

Personalización

Multigráficas

Python Cheat Sheet 5

Seaborn gráficas

NumPy (Numerical Python)	Indices, Subsets, Metodos de Arrays	Operaciones estadísticas y matemáticas	Funciones de conjuntos	Estadística
Crear arrays	Indices de arrays	Operaciones estadísticas y matemáticas	np.unique(array) devuelve un array con los valores únicos del array ordenados	Tablas de frecuencias
Crear arrays de listas array = np.array(lista, dtype= tipo) crea un array unidimensional de una lista	<pre>array[i] devuelve la indice i; las indices de los arrays unidimensionales funcionan igual que las listas array[i, j] o array[i][j] devuelve el elemento de la columna j de la fila i</pre>	El parametro axis en arrays bidimensionales: axis = 0 columnas axis = 1 filas - si especificamos el axis, la operación devuelve el	np.unique(array, return_index=True) devuelve un array con los valores únicos del array ordenados y un array con la posición de la primera instancia de cada valor np.unique(array, return_inverse=True) devuelve un array con	Frecuencias absolutas el número de veces que se repite un número en un conjunto de datos df = df.groupby('columna').count().reset_index()
<pre>array = np.array([lista1, lista2]) crea un array bidimensional de dos listas array = np.array([listadelistas1, listadelistas2]) crea un array bidimensional de dos listas</pre>	<pre>array[:,:n] seleccionar todas las filas y las columnas hasta n-1 array[h, i, j] o array[h][i][j] devuelve el elemento de la columna j de la fila i del array h</pre>	resultado por cada fila o columna. Por ejemplo: np.sum(array, axis = 0) devuelve un array con la suma de cada fila	los valores únicos del array ordenados y un array con las posiciones de cada elemento de cada valor np.unique(array, return_counts=True) devuelve un array con los valores únicos del array ordenados y un array con el	Frecuencias relativas las veces que se repite un número o categoría en un conjunto de datos respecto al total, en porcentajes
Crear otros tipos de arrays array = np.arange(valor_inicio, valor_final, saltos) crea	<pre>array[h][i][j] = n cambiar el valor del elemento en esta posicion al valor n</pre>	El parametro axis en arrays multidimensionales: axis = 0 dimensión axis = 1 columnas	número de veces que aparece cada valor np.unique(array, axis = b) devuelve un array con los valores únicos ordenados de las filas o columnas	<pre>df_group_sin_str = df_group.drop('columna_str', axis=1) frecuencia_relativa = df_group_sin_str / df.shape[0] * 100</pre>
un array usando el formato [start:stop:step] array = np.ones(z,y,x) crea un array de todo unos de la forma especificada array2 = np.ones_like(array1) crea un array de todo unos	Subsets array > n devuelve la forma del array con True o False según si el elemento cumple con la condición o no	<pre>axis = 2 filas - si especificamos el axis, la operación devuelve el resultado por cada dimensión, fila o columna.</pre>	Funciones para arrays unidimensionales np.intersect1d(array1, array2) devuelve un array con los valores únicos de los elementos en común de dos arrays	columnas = df_group_sin_strings.columns df_group[columnas] = frecuencia_relativa
<pre>de la forma basada en otra array array = np.zeros(z,y,x) crea un array de todo zeros de la forma especificada array2 = np.zeros_like(array1)</pre> crea un array de todo	array[array > n] devuelve un subset: todos los valores que cumplen la condición en una lista dentro de un array	Por ejemplo: np.sum(array_3D, axis = 0) devuelve un array de una matriz con la suma de todas las matrices np.sum(array_3D, axis = 1) devuelve un array donde	np.intersect1d(array1, array2, return_indices=True) devuelve un array con los valores únicos de los elementos en común de dos arrays y arrays con los índices de cada valor, por array	Tablas de contingencia tabla de frecuencias que cuenta todas las combinaciones posibles de cada pareja de valores de las columnas que estamos intentando comparar
zeros de la forma basada en otra array array = np.empty((z,y,x), tipo) crea un array vacio con datos por defecto tipo float array2 = np.empty_like(array1) crea un array vacia con la	array[(array > n) & (array < m)] devuelve un subset: todos los valores que cumplen las condiciones en una lista dentro de un array; se puede usar para "or"	las filas contienen las sumas de las columnas de cada matriz Operaciones con parámetro del axis:	<pre>np.union1d(array1, array2) devuelve un array ordenado con los elementos resultantes de unir dos arrays (valores únicos)</pre>	<pre>df_crosstab = pd.crosstab(df['columna1'], df['columna2'], normalize = True, margins = True) normalize muestra los valores en porcentajes (por uno)</pre>
forma basada en otra array array = np.eye(z,y,x, k = n) diagonal empezando en la posicion k array = np.identity(x) crea una matriz de identidad con	<pre>Metodos de arrays nuevo_array = array.copy() crea un a copia del array np.transpose(array_bidimensional) cambia los filas del</pre>	<pre>np.sum(array_3D) devuelve la suma de todos los elementos de los matrices np.mean(array) devuelve la media de todo el array np.std(array) devuelve la desviación estándar de</pre>	<pre>np.in1d(array1, array2) devuelve un array con True o False por cada elemento de array1 según si aparece el mismo valor en array2 np.setdiff1d(array1, array2) devuelve un array ordenado con</pre>	margins muestra los totales y subtotales Coeficiente de correlación de Pearson - nos permite conocer la intensidad y dirección de la
ceros en filas y unos en la diagonal, de forma cuadrada	array a columnas y las columnas a filas np.transpose(array multidimensional) de columnas al número de arrays y viceversa; el número	todo np.var(array) devuelve la varianza de valores de todo	los valores únicos que están en array1 pero no en array2 np.setxor1d(array1, array2) devuelve un array ordenado con los valores únicos que NO están en común de los dos arrays	relación entre las dos variables - coeficiente > 0: correlación positiva - coeficiente < 0: correlación negativa
NumPy Random	de filas no cambia np.transpose(array_multidimensional, (z,y,x)) hace la	<pre>np.min(array) devuelve el valor mínimo del array np.max(array) devuelve el valor máximo del array np.sum(array) devuelve la suma de los elementos del</pre>	Estadística	- coeficiente = 1 o -1: correlación total - coeficiente = 0: no existe relación lineal
<pre>np.random.seed(x) generador de números aleatorios, para que las funciones random que van después siempre cogerán los mismos valores "aleatorios" Crear arrays con valores aleatorios array = np.random.randint(inicio, final, forma_matriz)</pre>	transposicion segun lo que especificemos usando las posiciones de la tupla (0,1,2) de la forma original array = np.arange(n).reshape((y,x)) crea un array usando reshape para definir la forma array = np.reshape(array, (z,y,x)) crea un array con los valores de otro array usando reshape para definir la forma	array np.cumsum(array) devuelve un array con la suma acumulada de los elementos a lo largo del array np.cumprod(array) devuelve un array con la multiplicación acumulada de los elementos a lo largo del array	Medidas de dispersión Desviación respecto a la media la diferencia en valor absoluto entre cada valor de los datos y su media aritmética diferencias = df['columna'] - df['columna'].mean() desviación_media = np.abs(diferencias)	<pre>df['columna1'].corr(df['columna2'] calcula la correlacion entre dos variables matriz_correlacion = df.corr() crea una matriz mostrando las correlaciones entre todos los variables sns.heatmap(df.corr()[['column1', 'column2']], cmap = 'color_palette', annot = True, vmin = -1, vmax = 1) crea una grafica heatmap de la matriz de correlaciones</pre>
crea un array de números aleatorios entre dos valores; forma_matriz: (z,y,x) z: número de arrays y: número de filas x: número de columnas array = np.random.randint(inicio, final) número aleatorio en el rango array = np.random.rand(z,y,x) crea un array de floats aleatorias con la forma que le especificemos; por defecto genera números aleatorios entre 0-1 array = np.random.random_sample((z,y,x)) crea un array de	array = np.swapaxes(array, posicion, posicion) intercambia dos ejes de una matriz usando las posiciones (z=0,y=1,x=2) de la forma original Otras operaciones np.sort(array) devuelve un array con los valores de cada fila ordenados en orden ascendente por defecto np.sort(array, axis = 0) devuelve un array con los valores de cada columna ordenados en orden ascendente np.sort(-array) devuelve un array con los valores de	Operaciones sin parámetro del axis: np.sqrt(array) devuelve un array con la raíz cuadrada no negativa de cada elemento del array np.exp(array) devuelve un array con el exponencial de cada elemento del array np.mod(array1, array2) devuelve un array con el resto de la división entre dos arrays np.mod(array1, n) devuelve un array con el resto de la división entre el array y el valor de n np.cos(array) devuelve un array con el coseno de cada elemento del array np.sin(array) devuelve un array con el seno de cada	Varianza medida de dispersión; la variabilidad respecto a la media df['columna'].var() Desviación estándar o desviación típica la raíz cuadrada de la varianza; cuanto mayor sea, mayor será la dispersión o variabilidad en nuestros datos df['columna'].std() Robustez - cuanto más cantidad de datos, más robustos	Sesgos (skewness) medida de la asimetría de la distribución de los valores de una variable alrededor de su valor medio - valor de sesgo positivo: sesgado a la derecha - valor de sesgo negativo: sesgado a la izquierda - valor de sesgo igual a 0: valores simetricos sns.displot(df['columna'], kde = True) crea un histograma que muestra la distribution de los valores import scipy.stats import skew skew(df['columna'] muestra el valor del sesgo de una
floats aleatorias con la forma que le especificemos; por defecto genera números aleatorios entre 0-0.9999999 array = np.random.z,y,x=None) devuelve un número aleatorio en 0 y 0.99999999999 np.round(np.random.rand(z,y,x), n) crear array con floats	cada fila ordenados en orden descendente np.round(array, decimals = x) devuelve un array con los valores del array redondeados a x decimales np.round(array, decimals = x) devuelve un array con los valores del array redondeados a x decimales	elemento del array np.sin(array) devuelve un array con la tangente de cada elemento del array Operaciones de comparación en arrays	<pre>1/n donde n es el numero de registros Coeficiente de variación el cociente entre la desviación típica y la media; cuanto mayor sea, mayor será la dispersión en nuestros datos df['columna'].std() / df['columna'].mean()</pre>	Intervalos de confianza describe la variabilidad entre la medida obtenida en un estudio y la medida real de la población (el valor real)
<pre>de n decimales np.random.uniform(n,m, size = (z,y,x)) genera muestras aleatorias de una distribución uniforme en el intervalo entre n y m np.random.binomial(n,m, size = (z,y,x)) genera muestras con una distribución binomial; n es el numero total de</pre>	np.where(array > x) devuelve los indices de los valores que cumplan la condición, por fila y columna Operaciones con arrays	bidimensionales np.any(array > n) devuelve True o False segun si cualquier valor del array cumpla con la condicion np.any(array > n, axis = b) devuelve un array con True o False por cada columna o fila según si algún	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	<pre>import scipy.stats as st st.t.interval(alpha = n, df = len(df['columna']-1, loc = np.mean(df['columna']), scale = st.sem(df['columna'])) devuelve el rango de valores para lo cual hay un n% de</pre>
pruebas; m es la probabilidad de éxito np.random.normal(loc = n, scale = m, size = (z,y,x)) genera números aleatorios de una distribución normal (curva de campana); loc es la media; scale es la desviación estándar np.random.permutation(array) devuelve un array con los mismos valores mezclados aleatoriamente	<pre>np.add(array1, array2) suma dos arrays np.subtract(array1, array2) resta el array2 del array1 np.multiply(array1, array2) multiplica dos arrays np.divide(array1, array2) divide el array1 por el array2 array + n, n * array, etc operadores algebraicos</pre>	valor de la fila o columna cumpla con la condición np.all(array > n) devuelve True o False segun si todos los valores del array cumpla con la condicion np.all(array > n, axis = b) devuelve un array con True o False por cada columna o fila según si todos los valores de la fila o columna cumplan con la condición	<pre>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</pre>	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

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

hipótesis y comprobar suposiciones con la ayuda de

descubrir patrones, detectar anomalías, probar

.head(), .tail(), .describe(), .info(), .shape

stb.freq() devuelve el value counts de variables

stb.missing() tabla de cuenta de nulos y el

quitar duplicados (filas o columnas)

cambiar tipo de datos de columnas

categóricas, mas el porcentaje, cuenta cumulativa y

estadísticas y representaciones gráficas.

1. Entender las variables

si temenos nulos o duplicados

que valores unicos temenos

.unique(), .value_counts()

que variables temenos

que tipos de datos

.dtypes(), .info()

.duplicated().sum()

librería sidetable:

porcentaje cumulativa

porcentaie del total

2. Limpiar el dataset

isnull().sum()

Extraccion obtener datos crudos v almacenarlos

- Tablas de bases de datos SOL o NoSOL - Ficheros de texto plano

ETL: Extract, Transform, Load

- Emails - Información de páginas web

- Hoias de cálculo - Ficheros obtenidos de API's

Transformación procesar los datos, unificarlos, limpiarlos,

Carga

APIs

datos

pasamos

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

suelen ser:

de lo cual dependerá de la naturaleza, el tamaño y la complejidad de los datos. Los sistemas más comunes - Ficheros csv - Ficheros ison

import requests libreria para realizar peticions HTTP

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

header = {} opcional; contiene informacion sobre las

response = requests.get(url=url, header = header)

response = request.get(url=url, params=variables)

parametros segun el diccionario de parametros que le

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

pedimos a la API que nos de los datos con los

peticiones realizadas (tipo de ficheros, credenciales)

- cargar los datos en su formato de destino, el tipo

- Bases de datos - Almacenes de datos (Data Warehouse)

- Lagos de datos (Data Lakes)

a una URL, para hacer web scraping

pedimos a la API que nos de los datos

variables = {'parametro1':'valor1'.

'parametro2':'valor2'}

datos en un dataframe

Codigos de respuesta de HTTP

ordenar columnas separar columna en dos con str.split()

crear intervalos con pd.cut() crear porcentajes o ratios decidir como tratar outliers: mantenerlos,

cambiar nombres de columnas

decidir como tratar nulos: - eliminar filas o columnas con nulos drop.na() - imputar valores perdidos:

eliminarlos, o reemplazarlos con la media,

mediana o moda; o aplicar una imputacion

- 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 anomalias 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

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

- en general es la afirmación contraria a la que queremos probar Hipótesis alternativa (H1)

Hipótesis nula (H0)

- en general la afirmación que queremos comprobar

- medida de la probabilidad de que una hipótesis nula sea cierta - valor entre 0 v 1

- si *p-valor* > 0.05 ✔ Aceptamos la hipótesis nula.

- si *p-valor* < 0.05 **X** Rechazamos la hipótesis nula.

Machine Learning: Preparación

Hipotesis Nula y Errores Tipo I y II

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:

más alineados están los puntos entorno a la recta, más normales

- histograma o distribución - grafico de cuantiles teóricos (0-0)

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)

platicurtosis: 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

- p-valor del test > 0.05: datos normales ✓ - p-valor del test <) 0.05: datos NO normales Test de Kolmogorov-Smirnov

- para muestras > 5000

stats.shapiro(df["datos"])

- 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

Variables numéricas: Correlaciones

Independencia entre variables predictoras

- las variables predictoras tienen que ser independientes para

pairplot sns.pairplot(df)

poder crear un modelo de regresión lineal

covarianza df numéricas.cov()

Tests estadísticos

- 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

Homocedasticidad (homogeneidad de varianzas)

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

Visualmente: violinplot

- 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,

Variables numéricas: - hav que crear un dataframe de las columnas numéricas

heterocedasticidad

center='median')

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,

columns = ['col norm']) df['col norm'] = df datos norm

Normalización

df["col VR"].min())

Método raiz cuadrada

Método stats.boxcox()

Método MinMaxScaler

modelo = MinMaxScaler()

modelo.fit(df["col VR"])

from scipy import stats

stats.boxcox(df["col VR"])

df["col norm"], lambda ajustada =

import math

math.sqrt(x))

np.log(x) if x > 0 else 0)

Método logarítmica

df["col norm"] = (df["col VR"] -

df["col VR"].media()) / (df["col VR"].max() -

df["col norm"] = df["col VR"].apply(lambda x:

df["col norm"] = df["col VR"].apply(lambda x:

from sklearn.preprocessing import MinMaxScaler

datos normalizados = modelo.transform(df["col VR"])

df datos norm = pd.DataFrame(datos normalizados,

no se puede hacer si algún valor sea 0

Método manual

Estandarización Método manual

Sklearn StandardScaler

df["col esta"] = (df ["col VR"] - df ["col_VR"].media()) / (df ["col_VR"].std()

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()

(df num sin VR)

scaler.fit(df num sin VR)

datos estandarizados = scaler.transform

columns = df num sin VR.columns)

df datos esta = pd.DataFrame(datos estandarizados,

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

from sklearn.model selection import train test split from sklearn.ensemble import DecisionTreeRegressor from sklearn import tree from sklearn.metrics import r2 score, mean_squared_error, mean_absolute_error from sklearn.model_selection import GridSearchCV 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 arbol = DecisionTreeRegressor(random state=0) arbol.fit(x_train, y_train) Para dibujar el árbol: fig = plt.figure(figsize = (10,6)) tree.plot tree(arbol, feature names = x train.columns, plt.show() 4. Hacemos las predicciones y_predict_train = arbol.predict(x_train) y predict test = arbol.predict(x test) 5. 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 param = {"max depth": [n,m,1], "max features": [a,b,c,d], "min_samples_split": [10, 50, 100], "min samples leaf": [10, 50, 100]} definimos un diccionario de los hiperparametros 6. Iniciamos el modelo con GridSearch gs = GridSearchCV(estimator = DecisionTreeRegressor(), param_grid = param, cv=10, verbose=-1, return_train_score = True, scoring = "neg_mean_squared_error") - GridSearch ejecuta todas las posibles combinaciones de hiperparámetros 7. Ajustamos el modelo en el GridSearch gs.fit(x_train, y_train) 8. Aplicamos el método de best_estimator_ mejor_modelo = gs.best_estimator_ devuelve la mejor combinación de hiperparámetros 9. Volvemos a sacar las predicciones y_pred_test_dt2 = mejor_modelo.predict(x_test) y_pred_train_dt2 = mejor_modelo.predict(x_train) Regresión Lineal: Random Forest seguir los mismos pasos como para el Decision Tree pero

con RandomForestRegressor()

from sklearn.ensemble import RandomForestRegressor

Regresión Lineal: Decision Tree

Imbalanced Data Downsampling ajustar la cantidad de datos de la categoría mayoritaria a la minoritaria Método manual 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) Método RandomUnderSample 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 = Método Tomek x = df.drop('col VR', axis=1) v = df['col VR'] x train, x test, y train, y test = train test split(x, y, test size = 0.2, random state = 42)tomek sampler = SMOTETomek() X_train_res, y_train_res = tomek_sampler.fit_resample(X_train, y_train) Upsampling ajustar la cantidad de datos de la categoría minoritaria a la mayoritaria Método manual 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) Método RandomOverSample 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 =