

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

# Python Cheat Sheet 4

## Pandas

### Union de datos

**.concat()** unir dataframes con columnas en comun

**df\_union = pd.concat([df1, df2, df3], axis=b, join = 'inner/outer', ignore\_index = True/False)**

parametros:

- axis = 0** une por columnas - los dataframes van uno encima del otro; las columnas tienen que ser de formatos compatible
- axis = 1** une por filas - los dataframes van uno al lado del otro; los datos deben ser relacionados para que tenga sentido
- join = 'inner'** solo se quedan elementos que aparecen en todos los dataframes
- join = 'outer'** se queda todo los datos de todos los dataframes
- ignore\_index = True/False** por defecto es False; si es True no usa las índices para la union (por ejemplo para union por el axis 0)

**.merge()** unir las columnas de un dataframe a otro

**df\_nuevo = df1.merge(df2, on = 'columna') inner merge**

**df\_nuevo = pd.merge(left = df1, right = df2, how='left', left\_on = 'columna\_df1', right\_on = 'columna\_df2')** left merge

parametros:

- how = 'left' | 'right' | 'outer' | 'inner' | 'cross'**
- on = columna | [columna1, columna2, etc]** si las columnas se llaman igual en los dos dataframes
- left\_on = columna\_df1 | right\_on = columna\_df2** para especificar por donde hacer el merge
- suffixes = ['left', 'right']** por defecto nada, el sufijo que apareciera en columnas duplicadas

**.join()** unir dataframes por los indices

**df\_nuevo = df1.join(df2, on = 'columna', how = 'left')** inner merge

parametros:

- how = 'left' | 'right' | 'outer' | 'inner'** por defecto left
- on = columna** la columna o indice por el que queremos hacer el union; tienen que tener el mismo nombre en los dos dataframes
- lsuffix = 'string' | rsuffix = 'string'** por defecto nada, el sufijo que apareciera en columnas duplicadas

### Group By

**df\_groupby = df.groupby("columna\_categoria")** crea un objeto DataFrameGroupBy; agrupa los valores segun las categorías de los valores de la columna indicada (o múltiples columnas en una lista)

**df\_groupby.ngroups** devuelve el numero de grupos

**df\_groupby.groups** devuelve un diccionario donde los keys son las categorías y los valores son listas de los índices de cada elemento en la categoría

**df\_grupo1 = df\_groupby.get\_group("grupo1")** devuelve un dataframe con los resultados de un grupo (la categoría indicada como grupo1)

**Cálculos con groupby:**

**df\_nuevo = df.groupby("columna\_categoria").mean()** devuelve un dataframe con la media de todas las columnas de valores numéricos, por categoría

**df\_nuevo = df.groupby("columna\_categoria")["columna1"].mean()** devuelve un dataframe con la media de la columna especificada

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

**df\_nuevo = df.groupby("columna\_categoria", dropna = False) ["columna\_valores"].agg([nombre\_columna = 'estadistico1', nombre\_columna2 = 'estadistico2'])** añade columnas con los cálculos de los estadísticos especificados

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

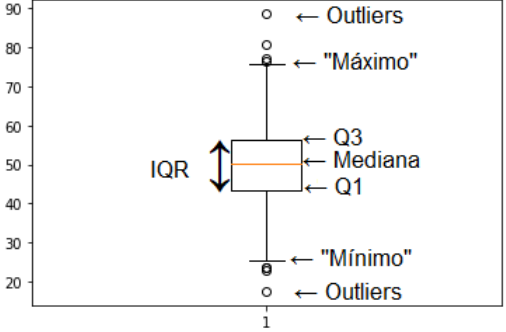
Apply
<p>- apply() toma una función como argumento y la aplica a lo largo de un eje del DataFrame</p>
<p><b>df['columna_nueva'] = (df['col_1'].apply(función))</b> crea una columna nueva con los valores de otra columna transformados según la función indicada</p>
<p><b>apply()</b> con datetime</p> <p>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

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

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

Funciones de conjuntos
<p><b>np.unique(array)</b> devuelve un array con los valores únicos del array ordenados</p> <p><b>np.unique(array, return_index=True)</b> devuelve un array con los valores únicos del array ordenados y un array con la posición de la primera instancia de cada valor</p> <p><b>np.unique(array, return_inverse=True)</b> devuelve un array con los valores únicos del array ordenados y un array con las posiciones de cada elemento de cada valor</p> <p><b>np.unique(array, return_counts=True)</b> devuelve un array con los valores únicos del array ordenados y un array con el número de veces que aparece cada valor</p> <p><b>np.unique(array, axis = b)</b> devuelve un array con los valores únicos ordenados de las filas o columnas</p>
<p><b>Funciones para arrays unidimensionales</b></p> <p><b>np.intersect1d(array1, array2)</b> devuelve un array con los valores únicos de los elementos en común de dos arrays</p> <p><b>np.intersect1d(array1, array2, return_indices=True)</b> devuelve un array con los valores únicos de los elementos en común de dos arrays y arrays con los índices de cada valor, por array</p> <p><b>np.union1d(array1, array2)</b> devuelve un array ordenado con los elementos resultantes de unir dos arrays (valores únicos)</p> <p><b>np.in1d(array1, array2)</b> devuelve un array con True o False por cada elemento de array1 según si aparece el mismo valor en array2</p> <p><b>np.setdiff1d(array1, array2)</b> devuelve un array ordenado con los valores únicos que están en array1 pero no en array2</p> <p><b>np.setxor1d(array1, array2)</b> devuelve un array ordenado con los valores únicos que NO están en común de los dos arrays</p>
NumPy Random
<p><b>np.random.seed(x)</b> establece la semilla aleatoria del generador de números aleatorios, para que las funciones random que van después siempre cogerán los mismos valores “aleatorios”</p> <p><b>Crear arrays con valores aleatorios</b></p> <p><b>array = np.random.randint(inicio, final, forma_matriz)</b> crea un array de números aleatorios entre dos valores; <b>forma_matriz:</b> (z,y,x)</p> <p>z: número de arrays</p> <p>y: número de filas</p> <p>x: número de columnas</p> <p><b>array = np.random.randint(inicio, final)</b> devuelve un número aleatorio en el rango</p> <p><b>array = np.random.rand(z,y,x)</b> crea un array de floats aleatorias con la forma que le especificemos; por defecto genera números aleatorios entre 0-1</p> <p><b>array = np.random.random_sample((z,y,x))</b> crea un array de floats aleatorias con la forma que le especificemos; por defecto genera números aleatorios entre 0-0.9999999...</p> <p><b>array = np.random.z,y,x=None)</b> devuelve un número aleatorio en 0 y 0.999999999999...</p> <p><b>np.round(np.random.rand(z,y,x), n)</b> crear array con floats de n decimales</p> <p><b>np.random.uniform(n,m, size = (z,y,x))</b> genera muestras aleatorias de una distribución uniforme en el intervalo entre n y m</p> <p><b>np.random.binomial(n,m, size = (z,y,x))</b> genera muestras con una distribución binomial; n es el numero total de pruebas; m es la probabilidad de éxito</p> <p><b>np.random.normal(loc = n, scale = m, size = (z,y,x))</b> genera números aleatorios de una distribución normal (curva de campana); loc es la media; scale es la desviación estándar</p> <p><b>np.random.permutation(array)</b> devuelve un array con los mismos valores mezclados aleatoriamente</p>



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

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

## Personalización

`color = "color"` establece el color de la grafica

`facecolor = "color"` establece el color del relleno

`edgecolor = "color"` establece el color de los bordes

### Colores en Scatter Plots:

`c= df['columna'].map(diccionario)`

`diccionario = {"valor1": "color1", "valor1": "color1"}`

`lista de colores`

`plt.xlabel("etiqueta_eje_x")` asignar nombre al eje x

`plt.ylabel("etiqueta_eje_y")` asignar nombre al eje y

`plt.legend(labels = ['label1', 'label2', etc])` muestra la leyenda cuando mostramos la figura

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

`plt.xlim([n,m])` establece el rango del eje x; donde n es el mínimo y m es el máximo

`plt.ylim([n,m])` establece el rango del eje y; donde n es el mínimo y m es el máximo

`plt.grid()` crea una cuadrícula al fondo de la figura; coge los parámetros:

`color = "color"`

`linestyle = "solid" | "dashed" | "dashdot" | "dotted"`

`linewidth = n` establece la anchura de la linea

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

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

## Multigráficas

`fig, ax = plt.subplots(numero_filas, numero_columnas)` crear una figura con multiples graficas; fig es la figura y ax es un array con subplots como elementos

Se usan los indices para establecer como es cada grafica:

`ax[indice].tipo_grafica(detalles de la grafica)`

`ax[indice].set_title('titulo')`

`ax[indice].set_xlabel('xlabel')`

`ax[indice].set_ylabel('ylabel')`

`ax[indice].set_xlim(min, max)`

`ax[indice].set_ylim(min, max)`

## Exportar figuras

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

## Seaborn

### Line plot

`fig = sns.lineplot(x = 'columna1', y = 'columna2', data = df, ci = None)` crea una gráfica lineal donde los ejes son: columna1 - x, columna2 - y

`ci = None` para que no muestra el intervalo de confianza de los datos

`hue = columna` opcional; muestra lineas en diferentes colores por categorias segun una variable

### Scatter plot

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

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

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

Estadística
<b>Tablas de contingencia</b> - una tabla de frecuencias que cuenta todas las combinaciones posibles de cada pareja de valores de las columnas que estamos intentando comparar - deberá contener al menos dos filas y dos columnas para representar datos categóricos - permite medir la interacción entre las variables <pre>df_crosstab = pd.crosstab(df[‘columna1’], df[‘columna2’])</pre>
<b>Coefficiente de correlación de Pearson</b> - nos permite conocer la intensidad y dirección de la relación entre las dos variables - coeficiente > 0: correlación positiva (según aumente el valor de una variable aumenta el valor de la otra) - coeficiente < 0: correlación negativa (según aumente el valor de una variable disminuye el valor de la otra) - coeficiente = 1 o -1: correlación total (positiva o negativa) - coeficiente = 0: no existe relación lineal entre las dos variables, pero puede haber algún otro tipo de relación que no se capture bien con este coeficiente <pre>df[‘columna1’].corr(df[‘columna2’])</pre> calcula la correlacion entre dos variables <pre>matriz_correlacion = df.corr()</pre> crea una matriz mostrando las correlaciones entre todos los variables
<b>Heatmap</b> <pre>sns.heatmap(df.corr(), cmap = ‘color_palette’, annot = True, vmin = -1, vmax = 1)</pre> crea un heatmap con una escala de colores que refleja los valores de correlacion <pre>annot = True</pre> para que aparezcan los valores <pre>vmin/vmax</pre> establecen la escala de color
<b>Sesgos (skewness)</b> - una medida de la asimetría de la distribución de los valores de una variable alrededor de su valor medio - valor de sesgo positivo: sesgado a la derecha - valor de sesgo negativo: sesgado a la izquierda - valor de sesgo igual a 0: valores simetricos <pre>sns.displot(df[‘columna’], kde = True)</pre> crea un histograma que muestra la distribution de los valores <pre>skew(df[‘columna’])</pre> muestra el valor del sesgo de una variable
<b>Intervalos de confianza</b> describe la variabilidad entre la medida obtenida en un estudio y la medida real de la población (el valor real)
<b>Confidence Interval = x +/- t*(s/√n)</b>
where: x: sample mean t: t-value that corresponds to the confidence level s: sample standard deviation n: sample size <pre>import scipy.stats as st</pre> <pre>st.t.interval(alpha = n, df = len(df[‘columna’])-1, loc = np.mean(df[‘columna’]), scale = st.sem(df[‘columna’]))</pre> devuelve el rango de valores para lo cual hay un n% de probabilidad que un valor real cae en ese rango alpha: porcentaje de confianza (p.ej. 90%, 95%, o 99%) df: los datos loc: la media scale: la desviación estándar