**APUNTES PANDAS**

**1. - Importando la librería**

Para importar Pandas, lo haremos como con cualquier otra librería pero, por convención se le suele asociar el alias “pd”.

import pandas as pd

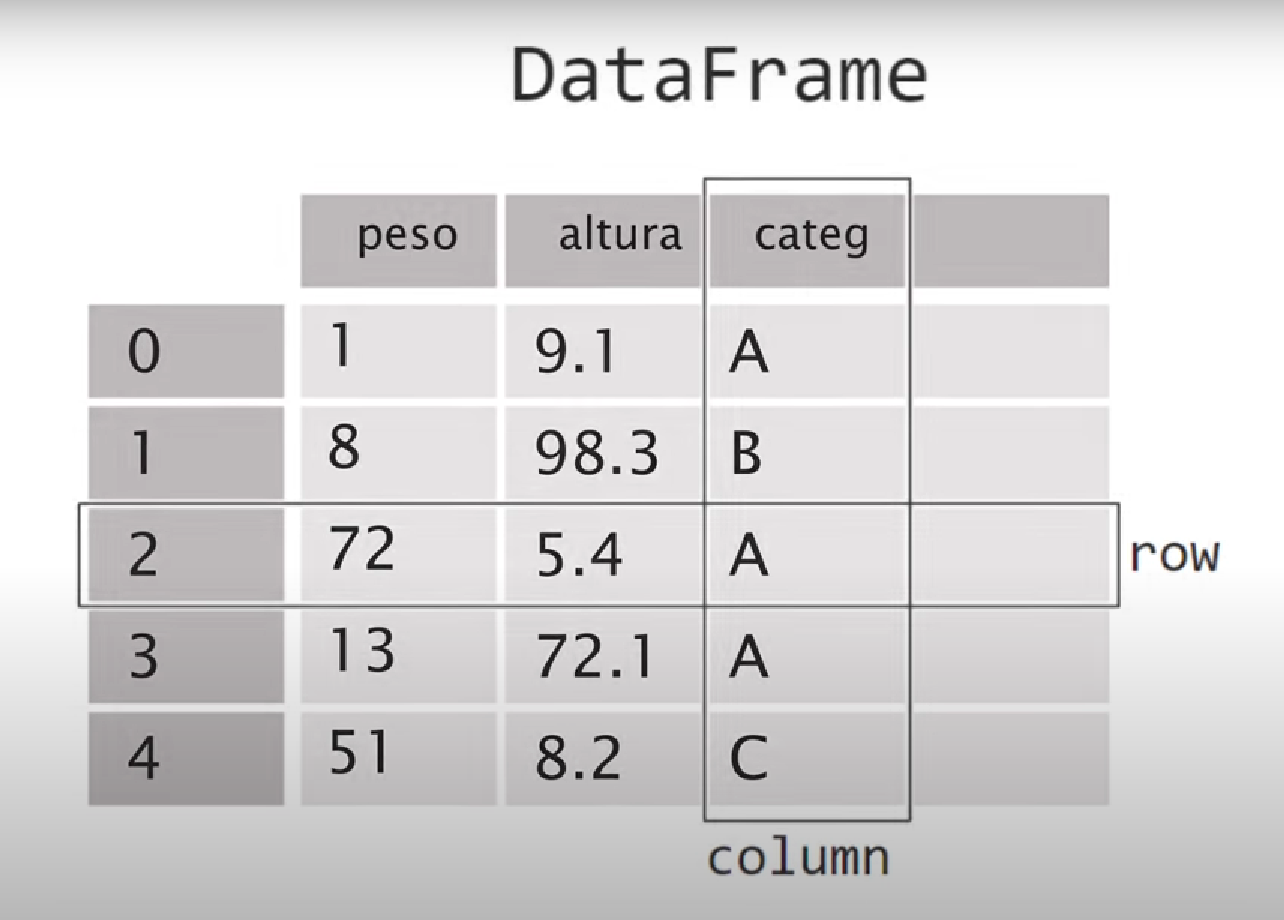
Si no está instalada la librería, abrimos el cmd y escribimos “pip install pandas”.

**2. - ¿Qué es un dataframe?**

Es una tabla en pandas, con sus filas y sus columnas.

Cada columna tiene su propio tipo de dato, sin admitir datos mezclados dentro de la misma columna. Además cada columna estará identificada por un nombre (encabezado) y un índice (como una lista de Python), aunque en la realidad nosotros, como en Excel, siempre identificaremos las columnas a través de un nombre.

Además, cada fila será identificada por un índice único, el cual, por defecto, será un índice numérico secuencial.



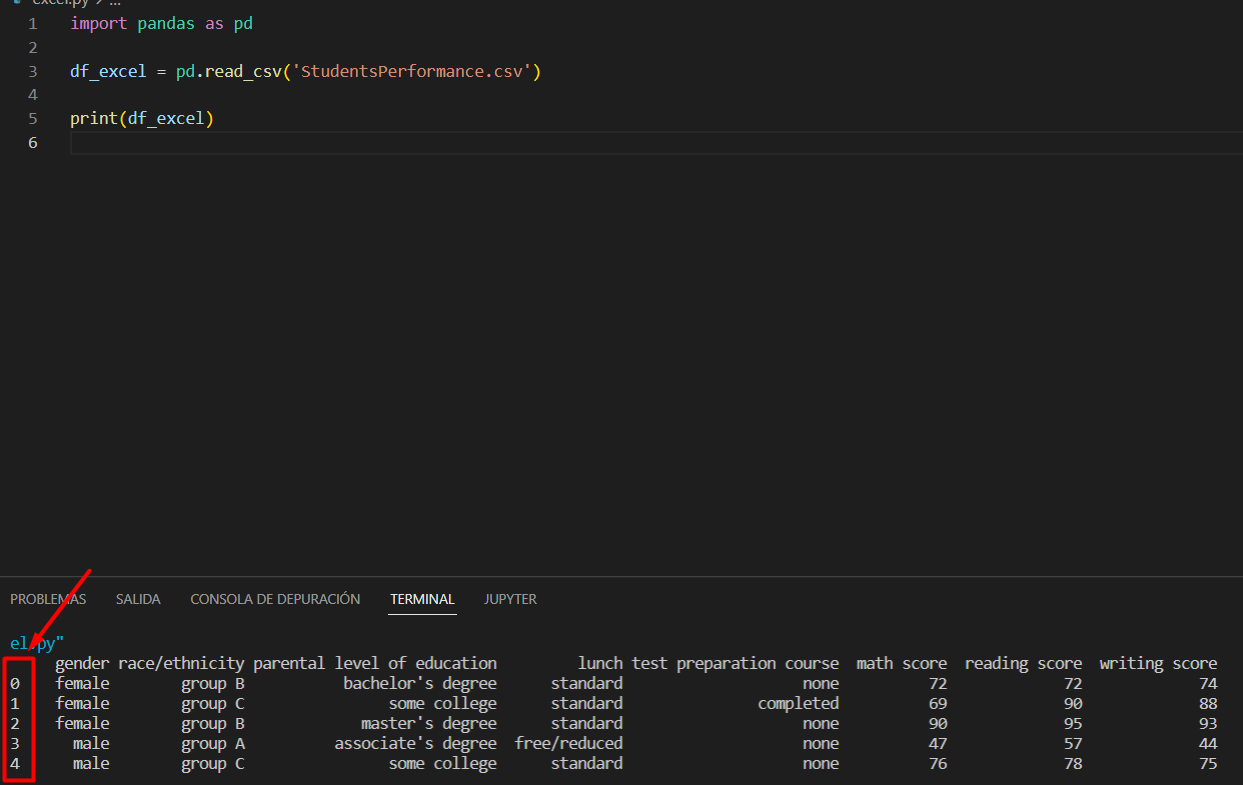
**3. – Carga de datos**

Para cargar los datos de un archivo csv en un dataframe lo haremos de la siguiente forma:

Nombre\_dataframe = pd.read\_csv(‘nombre\_fichero\_csv’)

Por convención, a los dataframe suele empezar llamándose “df”.

Si por ejemplo creamos un dataframe llamado df\_prueba y le cargamos un csv y despues lo imprimimos por pantalla con un “print(df\_prueba)”, veremos que nos aparecen todos los registros con un identificador numérico secuencial a la izquierda de cada uno. Ese es el que pandas crea por defecto.



Sin embargo, si nosotros ya tenemos una columna que nos sirva como ID y queremos utilizarlo como tal, haremos uso del parámetro “index\_col” de la siguiente manera:

df\_prueba = pd.read\_csv(‘fichero\_csv\_prueba”, index\_col = ‘nombre\_columna’)

**4. – Descubriendo los datos**

Si queremos que pandas nos muestre solo los primeros registros de nuestro dataframe, haremos uso del método head().

Si le pasamos un número por parámetro, nos devolverá ese número de registros contando desde el principio, en caso contrario, nos mostrará los 5 primeros.

Ej.:

print(df\_prueba.head(25)) # muestra los 25 primeros registros

print(df\_prueba.head()) # muestra los 5 primeros registros

Si por el contrario, queremos que pandas nos muestre solo los últimos registros de nuestro dataframe, haremos uso del método tail().

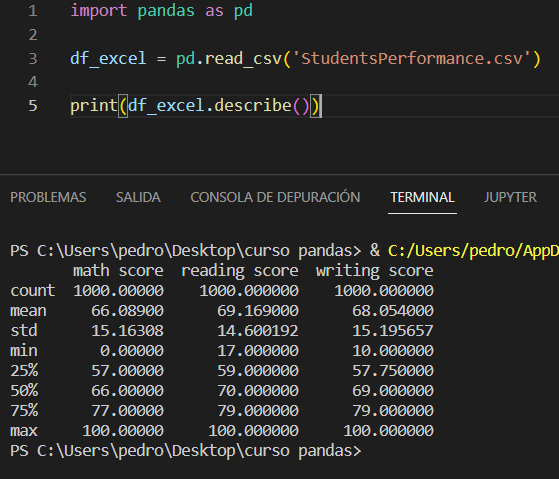
Si le pasamos un número por parámetro, nos devolverá ese numero de registros contando desde el final, en caso contrario, nos mostrará los 5 últimos.

Ej.:

print(df\_prueba.tail(25)) # muestra los 25 últimos registros

print(df\_prueba.tail()) # muestra los 5 últimos registros

Para descubrir mis datos calculando estadísticos descriptivos de todo el dataframe lo haremos usando la función describe().



Para cada una de las columnas numéricas calculará el número de registros que contienen datos, la media, la desviación estándar, el valor mínimo, el 25%, 50% y 75% (percentiles), y el máximo.

**5. – Limpieza de datos**

Lo siguiente que querremos hacer siempre, es hacer una limpieza de nuestros datos. Muchas veces en nuestros datos habrá espacios vacíos o datos incompletos. Con pandas nosotros podremos borrar estos datos o llenarlos con valores por defecto. A este proceso se le llama limpieza de datos.

Si queremos eliminar las filas que tienen datos incompletos, haremos uso de la función dropna().

Ej.:

df\_filtrado = df\_prueba.dropna()

Si imprimimos el df filtrado, veremos que nos habrá borrado las filas con datos vacíos o incompletos, pero si lo que queremos es que nos rellene esos datos con un valor específico, haríamos uso de la función fillna() de la siguiente manera:

df\_filtrado = df\_prueba.fillna(valor\_deseado\_por\_defecto)

Si queremos que nos rellene datos incompletos de columnas en concreto con un valor específico lo haríamos de la siguiente manera:

Df\_filtrado = df\_prueba.fillna({“nombre\_col”: valor, “nombre\_col2”: valor})

**6. – Filtrado de datos**

**6.1. – Filtrado por columnas**

Si queremos que se nos muestre los datos de un dataframe filtrado por columnas lo haremos de la siguiente manera:

print(nombre\_df[“nombre\_col”])

Si queremos que nos muestre más de una columna:

nombre\_df[[“nombre\_col”,”nombre\_col2”]]

**6.2. – Filtrado por filas**

Si queremos que se nos muestre el dataframe filtrado por filas podremos hacerlo identificando estas ya sea por el índice (valor numérico secuencial) o por el valor del identificador que hubiéramos definido.

* **A través de índice. Método iloc[]**

Si quiero el valor de un índice concreto:

Nombre\_df.iloc[índice\_fila]

Si quiero un rango de filas:

Nombre\_df.iloc[índice\_inicial:indicefinal]

Si quiero filas con índices concretos:

Nombre\_df.iloc[índice\_fila, índice\_fila2…]

* **A través de identificador. Método loc[]**

Funciona igual que el anterior en todos los casos, pero indicando el valor del id:

Nombre\_df.loc[15678, 15679]

**6.3. – Filtrado por filas y columnas**

Se indicaría de la siguiente forma:

df.loc[[id\_fila1, id\_fila2…],[índice\_col1, índice\_col2…]]

**6.4. – Filtrado por condiciones**

Si, por ejemplo, queremos que nos muestre los datos de la columna favoritos que tengan un valor mayor de 400, lo haríamos de la siguiente forma:

Df [ df [“favoritos”] > 400 ]

Si queremos concatenar condiciones con un and, lo haríamos con “&” y encerrando cada condición entre paréntesis. Si quisiéramos usar un or lo haríamos con “|”

Df [ ( df [“favoritos”] > 400 ) & ( df [“menciones”] > 20 ) ]

Si lo que queremos es filtrar texto, tendremos que hacerlo con el método “str.contains()”

Df [df[“col\_texto”].str.contains(“texto”)]

**7. – Transformación de datos**

Muchas veces, previo al proceso de análisis o ejecución de algoritmos existe una fase de transformación de datos, que no es más que, a partir de los datos ya existentes, crear o calcular nuevas columnas que tengan datos transformados. Estas transformaciones pueden ser tan simples como convertir una unidad que está en metros creando otra columna en mi dataframe donde tenga esa misma unidad en Km, o pueden ser tan complejos como aplicar una fórmula o usar un algoritmo bastante complejo sobre un campo.

En primer lugar, debemos definir la función que usaremos para la transformación. Por ejemplo, vamos a crear una que nos multiplique el número de retweets por un número aleatorio del 1 al 5:

Import random

Def calcularGanancias(retweets):

Ganancia = retweets \* random.randint(1,5)

Return ganancia

Ahora, para aplicarlo al dataframe creando una nueva columna con las ganancias, lo haríamos de la siguiente manera:

Df[“nombre\_nueva\_col”] = df[“columna\_a\_transformar”].apply(función\_a\_aplicar)

Siguiendo el ejemplo anterior, sería así:

Df[“ganancias”] = df [“retweets”].apply(calcularGanancias)

Básicamente le estamos diciendo que por cada fila, en el campo “retweets” ejecute la función calcular ganancias.

También puede surgir otro caso, y es el de que queramos crear una nueva columna a partir de un cálculo de 2 o más existentes. Para ello, lo haremos de la siguiente manera:

Supongamos que yo quiero crear una nueva columna llamada “popularidad” y que quiero que sea el resultado de dividir el número de personas que sigo por el número de seguidores.

Como antes, lo primero es definir la función:

def popularidad(fila):

resultado = fila[“followees”]/[“followers”]

return resultado

Como se puede ver, lo que se le pasa por parámetro ahora sería la fila. Ahora, para aplicar esta función:

Df[“popularidad”] = df.apply (popularidad, axis = 1)

Básicamente, estamos diciendo que por cada fila, ejecute la función popularidad (tomando esa fila como parámetro).

Dado que estamos haciendo un .apply() del DataFrame completo, y ya no de una columna individual, “axis = 1”, le especifica a .apply() que la función debe ser aplicada por cada fila de df.

**8. – Agrupación y agregación de datos**

Cuando yo ya tengo mis datos filtrados y transformados, muchas veces me será muy útil agruparlos. Podemos agruparlos por una columna, por ejemplo. Podemos agrupar los datos por una categoría, lo cual nos sirve para agregar los datos de mis otras columnas dada una función.

Por ejemplo, imaginemos que quiero calcular el promedio de los favoritos de los tweets por cada país. Deberíamos hacer una agrupación por país y usar una función de agrupación para poder agrupar los favoritos de cada país con una función que calcule el promedio de todos los datos que se van a agrupar. Esto lo hacemos con una función en pandas que se llama groupby() especificando el nombre de la columna que queremos agrupar y aplicándole nuestra función de aplicación (también llamada función de agregación).

Nota: Agrupar significa unir filas que tengan un mismo valor dentro de una columna. Pero ¿Qué sucede con los valores de otras columnas que se agrupan? Estos valores deben de sufrir una transformación. Que puede ser por ejemplo, quedarnos con el promedio de todos los valores que se agrupan o coger el máximo, o el mínimo, o sumar todos los valores, etc.

Df.groupby(“nombre de la columna por la que quiero agrupar mis filas”).función\_de\_agrupacion

Ej.:

Df.groupby(“country”).mean() # halla la media del conjunto de cada valor del campo “country”

También podríamos aplicar una función de agregación para cada columna con el método “agg()”:

Df.groupby(“country”).agg({

“followers”: ‘sum’,

“mentions”: ‘mean’,

“retweets”: ‘max’

})

**9. Graficación de datos**

Para sacar gráficos, utilizaremos la librería matplotlib y la importaremos de la siguiente manera:

import matplotlib.pyplot as plt

Para un gráfico de líneas:

Nombre\_df[“columna de la que sacar grafico”].plot()

Para mostrar el gráfico:

Plt.show()

Para un gráfico de barras:

Nombre\_df[“columna de la que sacar grafico”].plot(kind = “bar”)

**10. – Guardado de datos**

Para exportar nuestra tabla tratada a otro archivo seria de la siguiente manera:

A un csv:

df.to\_csv(“nombre.csv”)

Esto sería aplicable para otros formatos (to\_excel, to\_json…)