Librería Pandas, trabajando con DataFrames

https://pandas.pydata.org/docs/reference/index.html

Índice de contenidos

1.Importar librerías

Ir al índice

In [588]:

```
import os
import pandas as pd
import numpy as np
import datetime
```

2.Cargar y exploración de datos

Ir al índice

In [632]:

```
path = "C:\\Users\\oscargc\\Documents\\NATURGY\\PANDAS\\"
file = "SalariesExtended.xlsx"
```

In [ ]:

```
df = pd.read_excel(path + file)
df
```

In [ ]:

```
df.info()
```

In [ ]:

```
df.describe()
```

In [ ]:

```
#Valores unicos de una columna
df["prov"].unique()
```

In [ ]:

```
#Valores unicos de todas las columnas
for c in df.columns:
    print()
    print(c, df[c].unique())
```

3.Selección de registros

Ir al índice

3.1 Seleccionar primeros registros

Ir al índice

In [ ]:

```
df.head()
```

In [ ]:

```
df.head(3)
```

3.2 Seleccionar últimos registros

Ir al índice

In [ ]:

```
df.tail()
```

3.3 Cuando queremos obtener una única columna:

Ir al índice

In [ ]:

```
# opcion1
df["mesesexp"]
```

In [ ]:

```
#opcion 2, esta opcion no será válida si el nombre de la columna contiene espacios.
df.mesesexp
```

In [ ]:

```

```

3.4 Seleccionando columnas por tipos y cambios de tipo

Ir al índice

In [ ]:

```
numericas = df.select_dtypes(include = ["int16", "int32", "int64", "float16", "float32", "float64"])
numericas.head()
```

In [ ]:

```
numericas = df.select_dtypes(include = ["number"])
numericas.head()
```

In [ ]:

```
categóricas = df.select_dtypes(include = ["object"])
categóricas.head()
```

Algo interesante que podemos hacer una vez sabemos como aislar las variables categóricas es hacer un conteo de las categorías. Esto forma parte del proceso de exploración de los datos. Lo veremos un poco mas adelante, en el apartado 7.Agrupar datos.

También podemos cambiar los tipos de las variables. Por ejemplo, vamos a pasar prov a categórico, y de nuevo a integer:

In [649]:

```
df["prov"] = df["prov"].astype("str")
```

In [ ]:

```
df.info()
```

In [651]:

```
#volvemos a dejarlo como estaba
df["prov"] = df["prov"].astype("int64")
```

In [ ]:

```
df.info()
```

3.5 Métodos loc() y iloc()

Ir al índice

Método iloc

El método iloc se utiliza en los DataFrames para seleccionar los elementos en base a su ubicación. Su sintaxis es

data.iloc[filas, columnas]

donde y son la posición de las filas y columnas que se desean seleccionar en el orden que aparecen en el objeto.

In [ ]:

```
df
```

In [ ]:

```
df.iloc[:,5] #7 no incluido
```

In [ ]:

```
df.iloc[:,5:7] #7 no incluido
```

In [ ]:

```
df.iloc[50:52,5:7] #7 no incluido
```

In [ ]:

```
df.iloc[0:5] # Primeras cinco filas
```

In [ ]:

```
df.iloc[:, 0:5] # Primeras cinco columnas
```

In [ ]:

```
df.iloc[[0,2,1]]  # Primera, tercera y segunda filas
```

In [ ]:

```
df.iloc[:, [0,2,1]]  # Primera, tercera y segunda columnas
```

Método loc

El método loc se puede usar de dos formas diferentes: seleccionar filas o columnas en base a una etiqueta o seleccionar filas o columnas en base a una condición.

En base a etiqueta

In [ ]:

```
df.loc[:,:]
```

In [ ]:

```
df.loc[:,"prov"]
```

In [ ]:

```
df.loc[:,["prov","tattoo"]]
```

In [ ]:

```
df.loc[:5,["prov","tattoo"]]
```

In [ ]:

```
df.loc[90:99,["prov","tattoo"]]
```

En base a condición:

In [406]:

```
is_male = df.loc[:, 'sex'] == 'H'
```

In [ ]:

```
#Este es el objeto resultante
print(is_male)
print()
print( "El objeto es de tipo: ",type(is_male))
```

In [408]:

```
# con la siguiente expresion, conseguimos un DF al que aplicaremos el filtro creado anteriormente.
df_male = df.loc[is_male]
```

In [ ]:

```
df_male.head()
```

No obstante, existe una manera mas directa de realizar un filtro como el anterior, sin utilizar el método loc, como veremos a continuación.

3.6 Otras formas de filtrar

Ir al índice

En este caso obtenemos lo mismo que obtuvimos en el último ejemplo con el método loc, pero sin usar dicho método y de una manera mas directa:

In [ ]:

```
df_male_2 = df[df["sex"] == "H"]
df_male_2.head()
```

También podemos hacer filtros en base a valores numéricos o rangos:

In [ ]:

```
df_expertos = df[df["mesesexp"] > 30]
df_expertos.head()
```

In [ ]:

```
df_no_expertos = df[df["mesesexp"] < 10]
df_no_expertos.head()
```

In [ ]:

```
df_expe_media = df[(df["mesesexp"] >= 10) & (df["mesesexp"] <= 30)]
df_expe_media.head()
```

3.7 Método isin()

Ir al índice

In [ ]:

```
df[df['mesesexp'].isin([29,15])]
```

3.8 Ejercicios de selección de columnas y registros

Ir al índice

- **Vamos a crear un DF que contenga únicamente las columnas preg1, preg2, preg3, preg4, preg5, preg6, y aquellas filas cuyo valor en "sex" sea M y con salario superior a la media.**

Como sucede con cada problema de este tipo, debemos descomponerlo en pequeños problemas y dar solución a cada uno de ellos. Además, una práctica muy recomendable es escribir con nuestro porpio lenguaje lo que queremeos conseguir y cómo vamos a hacerlo. Un posible ejemplo sería:

- **Necesito filtrar mi DF de modo que solo contenga registros con "sex" = M y salary > que la media de salarios.**
- **Sobre el resultado anterior, necesito seleccionar las columnas preg1, preg2, preg3, preg4, preg5, preg6**
- **Necesito identificar un patrón para conseguirlo. He encontrado 2:**
  - **Estan ubicadas desde la columna 8 a la columna 13.**
  - **Todas ellas empiezan por "preg"**

In [ ]:

```
# lo haremos por pasos
#Filtramos por sexo
df_m = df[df["sex"] == "M"]
df_m["sex"].unique() #el método unique() nos permite revisar los posibles valores de la variable "sex"
```

In [ ]:

```
#Filtramos por salary mayor a la media
   #necesito conocer la media:
media = df.salary.mean()
media
```

In [ ]:

```
#ya tenemos almacenado en una variable la media de salarios; ahora solo tenemos que aplicar un filtro:

df_m_mayor_media = df_m[df_m["salary"] > media]
df_m_mayor_media.head(15)
```

Ahora solo tenemos que filtrar para quedarnos únicamente con las columnas preg_. recordamos las opciones:

- **Necesito identificar un patrón para conseguirlo. He encontrado 2:**
  - **Estan ubicadas desde la columna 8 a la columna 13.**
  - **Todas ellas empiezan por "preg"**

In [ ]:

```
# primera opcion, muy sencilla pero si no estan en orden no nos vale
```

```
df_m_mayor_media_pregs = df_m_mayor_media.iloc[:,8:14]
df_m_mayor_media_pregs.head()
```

La segunda opción es considerablemente mas compleja, pero también es independiente del orden de las columnas, y mucho mas reutilizable. Para llevarla a cabo debemos repasar algunos conceptos relativos a las cadenas de texto.

In [470]:

```
#una cadena de texto es iterable

texto = "Cadena de texto"
```

In [ ]:

```
for letra in texto:
    print (letra)
```

In [ ]:

```
print(texto[2:4])
```

Por tanto, podemos fácilmente localizar las primeras letras de una cadena de texto:

In [473]:

```
texto = "preg1"
```

In [ ]:

```
texto[0:4]
```

y aplicar cualquier condicion al respecto:

In [ ]:

```
if texto[0:4] == "preg":
    print("Correcto")
else:
    print("incorrecto!")
```

y si en vez de una palabra, tenemos una lista de palabras, podemos iterar por cada una y verificar si se cumple o no la condicion anterior:

In [476]:

```
#necesitamos una lista como esta, con el nombre de las columnas
lista = ['sex', 'prov', 'tattoo', 'years', 'salary', 'mesesexp', 'mesesempr',
       'position', 'preg1', 'preg2', 'preg3', 'preg4', 'preg5', 'preg6', 'end',
       'start', 'antiguedad']
```

In [ ]:

```
lista
```

In [478]:

```
#la buena noticia es que podemos obtener esta lista automáticamente
lista = list(df.columns)
```

In [ ]:

```
lista
```

In [ ]:

```
#iteramos
# para ver lo que estamos haciendo, imprimimos cada iteracion y comprobamos lo que contiene la variable p
alabra
for palabra in lista:
    print(palabra[:4])
```

In [ ]:

```
# por tanto, lo único que tenemos que hacer es someter cada iteración el contenido de la cariable palabra
a la condicion
for palabra in lista:
```

```
    if palabra[:4] == "preg":
        print(palabra)
```

In [482]:

```
# ya casi lo tenemos! Ahora solo tenemos que meter estas palabra en una nueva lista
preglist = []
for palabra in lista:
    if palabra[:4] == "preg":
        preglist.append(palabra)
```

In [ ]:

```
#por tanto, estas serán las columnas que han de componer nuestro df
preglist
```

In [484]:

```
#con el método loc, es fácil
df_m_mayor_media_pregs_op2 = df_m_mayor_media.loc[:,preglist]
```

NOTA: Tambien podíamos haber utilizado el método startswith(), que funciona independientemente del número de caracteres.

In [ ]:

```
texto = "pregunta"
texto.startswith("preg")
```

In [ ]:

```
texto.startswith("pr")
```

In [ ]:

```

```

4.Ordenando el dataframe

Ir al índice

In [ ]:

```
df_m_mayor_media_pregs.sort_values(by=['preg1'],ascending = True).head(10)
```

In [ ]:

```
df_m_mayor_media_pregs.sort_values(by=['preg1',"preg2"],ascending = True).head(10)
```

5.Creando nuevas columnas

Ir al índice

5.1 Operando

Ir al índice

In [489]:

```
#para abreviar, damos otro nombre a nuestro df
df2 = df_m_mayor_media_pregs_op2
```

In [490]:

```
#creamos una columna que sea la suma de preg1 y preg2
df2["p1+p2"] = df.preg1 + df.preg2
```

In [ ]:

```
df2.head(2)
```

In [492]:

```
#eliminamos la columna que acabamos de crear
```

```
df2.drop("p1+p2", axis=1, inplace = True) #fijaros en el detalle de axis1, cuando nos referimos a column
as
```

In [ ]:

```
df2.head()
```

5.2 Resumiendo

Ir al índice

In [ ]:

```
#podemos crear una nueva columna con la suma de todas las columnas preg
df2["suma"] = df2.sum(axis=1) #de nuevo, importante axis=1
df2.head()
```

In [ ]:

```
## o con la media, excluyendo la ultima columna que acabamos de crear
df2["media"] = df2.iloc[:,0:6].mean(axis=1)
df2.head()
```

5.3 Categorizando

Ir al índice

Podemos crear nuevas columnas categoricas en base a columnas numéricas. Por ejemplo, vamos a crear una nueva columna que admita los valores "baja" cuando la media sea menor a 3, "normal" cuando este entre 3 y 4 , y "alta" cuando sea superior a 4.

In [498]:

```
#para que funcione correctamente el método loc, debemos resetear el indice
df2 = df2.reset_index()
```

In [499]:

```
for i in range(len(df2)):
    if df2.loc[i,"media"] < 3:
        df2.loc[i,"cat_med"] = "Baja"
    elif df2.loc[i,"media"] <= 4:
        df2.loc[i,"cat_med"] = "Media"

    else:
        df2.loc[i,"cat_med"] =  "Alta"
```

In [ ]:

```
df2
```

5.4 Concatenando otras columnas

Ir al índice

Volvemos a nuestro dataframe original. Creamos una nueva columna como resultado de concatenar startyear-startmonth-startday. Para lograrlo, debemos iterar y recorrer todas las filas del dataframe.

In [ ]:

```
df.head()
```

In [503]:

```
for i in range (len(df)):
    df.loc[i,"start"] = str(df.loc[i,"startyear"]) + "-" + str(df.loc[i,"startmonth"]) + "-" + str(df.l
oc[i,"startday"])
```

In [ ]:

```
#la columna resultante, de momento es de tipo texto
df.info()
```

Cambiamos el tipo de la nueva columna a "datetime64"

Cambiamos el tipo de la nueva columna a "datetime64"

In [505]:

```
df["start"] = df["start"].astype("datetime64")
```

In [ ]:

```
df.info()
```

5.5 Manejando columnas de fechas

Ir al índice

Restamos las dos fechas "start" y "end" para obtener los dias transcurridos entre una fecha y otra:

In [507]:

```
df["antiguedad"] = ((df["end"] - df["start"]).dt.days)
```

In [ ]:

```
df.head()
```

In [ ]:

```
df.info()
```

Podemos ver que el tipo de la columna resultante es "int32" por tanto, perfectamente podemos dividir /30 para así majear la antiguedad en meses en vez de en días:

In [510]:

```
df["antiguedad"] = (((df["end"] - df["start"]).dt.days) / 30).astype("int") #si no usamos astype el resu
ltado saldrá en decimales
```

In [ ]:

```
df.head()
```

6.Reemplazo de valores

Ir al índice

Sustituimos los siguientes valores:

- **prov (1: Barcelona, 2: Tarragona, 3: Girona, 4: Lleida)**
- **position (1: Administración, 2: Compras, 3: Dirección)**

In [ ]:

```
df = df.replace({'prov' : { 1 : "BCN", 2: "TGN", 3:"GIR",4:"LLE" }})
df = df.replace({'position' : { 1 : "Admin", 2: "Compra", 3:"Direc"}})
df
```

In [543]:

```
#cambiamos M H por "Mujer" "Hombre"
df = df.replace({'sex' : { "M" : "Mujer", "H": "Hombre"}})
```

In [ ]:

```

```

In [ ]:

```

```

7.Agrupar datos

Ir al índice

7.1 Groupby

Ir al índice

Siguiendo con la tabla anterior, vamos a obtener la media de salario por genero

In [544]:

```
df_g = df.groupby(by = "sex").salary.mean()
```

In [ ]:

```
df_g
```

In [ ]:

```

```

In [ ]:

```
df.iloc[0:5] # Primeras cinco filas
df.iloc[:, 0:5] # Primeras cinco columnas
df.iloc[[0,2,1]]  # Primera, tercera y segunda filas
df.iloc[:, [0,2,1]]  # Primera, tercera y segunda columnas
```

Ahora, tal y como hemos visto en ejemplos anterioresm vamos a crear una nueva columna que categorice los meses de experiencia:

In [526]:

```
for i in range(len(df)):
    if df.loc[i,"antiguedad"] < 12:
        df.loc[i,"cat_ant"] = "Poca"
    elif df.loc[i,"antiguedad"] <= 24:
        df.loc[i,"cat_ant"] = "Media"

    else:
        df.loc[i,"cat_ant"] =  "Alta"
```

In [ ]:

```
df.head()
```

Ahora crearemos una agrupacion por sexo y categoría de antiguedad:

In [546]:

```
df_g = df.groupby(by = ["sex","cat_ant"]).salary.mean()
```

In [ ]:

```
df_g
```

A continuación, crearemos un df con las variables categóricas "sex" y "cat_ant", y con las variables numéricas "salary" y "years"

In [553]:

```
df_sex_ant = df.loc[:,["sex","cat_ant","years","salary"] ]
```

In [ ]:

```
df_sex_ant.groupby(by=["sex","cat_ant"]).mean()
```

In [ ]:

```
df_sex_ant.groupby(by=["sex","cat_ant"]).mean()
```

7.2 Conteos por categorías: value_counts()

Ir al índice

Agrupacion por conteo de filas

In [ ]:

```
df["sex"].value_counts()
```

In [ ]:

```
df["cat_ant"].value_counts()
```

Podemos crear un bucle para recorrer todas las variables categóricas y obtener el conteo de todas las posibles categorías en cada una de ellas:

In [ ]:

```
for i in df.select_dtypes(include = ["object"]):
    print()
    print (df[i].value_counts())
    print()
```

8.Cruce de variables categóricas

Ir al índice

In [ ]:

```
pd.crosstab(df.sex,df.cat_ant)
```

Cruzando todas las variables categóricas de 2 en 2 sin repeticion:

In [ ]:

```
from IPython.display import display, HTML

columnastexto = df.select_dtypes(include=['object']).columns

for i in range(len(columnastexto)):
    for j in range(i,len(columnastexto)):
        if i != j:
            tabla = pd.crosstab(df.loc[:,columnastexto[i]], df.loc[:,columnastexto[j]])

            display(HTML(tabla.to_html())) #esta linea únicamente es estética
```

9.Pivotando una tabla

Ir al índice

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.pivot_table.html

In [ ]:

```
df.head()
```

In [ ]:

```
df.pivot_table(index = "prov", columns = "sex", values = "salary", aggfunc="count")
```

In [ ]:

```
df.pivot_table(index = "sex", columns = "prov", values = "years", aggfunc="mean")
```

10.Eliminar columnas

Ir al índice

In [ ]:

```
#método drop y usando inplace
df.drop(['startday', 'startmonth'], axis = 'columns', inplace=True)
```

In [556]:

```
#método drop sin usar inplace
```

```
df = df.drop(columns=['startyear'])
```

In [ ]:

```
df
```

11.Creando DataFrames a partir de otros objetos

Ir al índice

In [ ]:

```
costes_envio =  [100, 150, 125]
costes_almacenaje = [500,600,800]
productos = ["Abrigos","Sombreros","Cinturones"]

#creamos un df con las listas anteriores
df_prod = pd.DataFrame(list(zip(costes_envio, costes_almacenaje, productos )),
               columns =["costes de envio", "costes de almacenaje", "productos"])
df_prod
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

12.Tratamiento de nulos

Ir al índice

En primer lugar tenemos que analizar y tomar decisiones en cuanto a los valores nulos. Vamos a cuantificar cuantos valores nulos tenemos:

In [ ]:

```
df.info()
```

Observamos que todas las variables tienen 100 valores no nulos, salvo tattoo, que solo presenta 94, lo que nos hace pensar que en esa variable tenemos 6 valores nulos. No obstante, vamos a cuantificarlo:

In [ ]:

```
# de este modo obtenemos la suma de valores nulos por columna
df.isna().sum()
```

In [ ]:

```
#de este modo obtenemos el % de valores nulos por columna
df.isna().mean() * 100
```

Una de las opciones es eliminar directamente las filas con valores nulos en alguna columna, pero no lo vamos a hacer.

Vamos a echar un vistazo a los registros con valores nulos en la columna "tattoo"

In [ ]:

```
df[df.tattoo.isnull()]
```

Creamos un df sin valores nulos:

In [76]:

```
df_nn = df[df.tattoo.notnull()]
```

In [ ]:

```
df_nn.isnull().sum()
```

Agrupamos por "sex" y obtenemos la media de tattoo por dicha categoría:

In [ ]:

```
#primera opcion
df_nn.groupby(by = "sex" ).mean()["tattoo"]
```

In [ ]:

```
#segunda opcion
df_nn.groupby(by = "sex" ).tattoo.mean()
```

Como es obvio, un apersona no puede tener un 60% o un 40% de un tatuaje. De modo que haremos lo siguiente: aquellos valores nulos en el campo "tattoo" que pertenezcan a género "H" serán reemplazados por 1 (0.4 es mas cerca de 0 que de 1) y aquellos que pertenezcan a género "M" serán reemplazados por 1 (0.67 mas cerca de 1 que de 0)

In [ ]:

```
#opcion 1
for i in range (len(df)):
    df.loc[i,"tattoo"] = 1 if df.loc[i,"sex"] == "M" else 0
```

In [85]:

```
#opcion 2
for i in range (len(df)):
    if df.loc[i,"sex"] == "M":
        df.loc[i,"tattoo"] = "1"
    else:
        df.loc[i,"tattoo"] = "0"
```

In [ ]:

```
#comprobamos que ya no tenemos nulos
df.isnull().info()
```

13.Merging and joining Dataframes

[Ir al índice]

Comenta los siguientes ejemplos, para ver qué es lo que hacen, y para que sirven.

In [592]:

```
df_prueba = pd.DataFrame()
```

In [ ]:

```
data1 = {'id': ['1', '2', '3', '4', '5'],
        'Feature1': ['A', 'C', 'E', 'G', 'I'],
        'Feature2': ['B', 'D', 'F', 'H', 'J']}

df1 = pd.DataFrame(data1, columns = ['id', 'Feature1', 'Feature2'])
df1
```

In [ ]:

```
data2 = {'id': ['1', '2', '6', '7', '8'],
         'Feature1': ['K', 'M', 'O', 'Q', 'S'],
         'Feature2': ['L', 'N', 'P', 'R', 'T']}

df2 = pd.DataFrame(data2, columns = ['id', 'Feature1', 'Feature2'])
df2
```

In [ ]:

```
data3 = {'id': ['1', '2', '3', '4', '5', '7', '8', '9', '10', '11'],
         'Feature3': [12, 13, 14, 15, 16, 17, 15, 12, 13, 23]}

df3 = pd.DataFrame(data3, columns = ['id', 'Feature3'])
df3
```

In [ ]:

```
pd.concat([df1, df2])
```

In [ ]:

```

```

y ahora?

In [ ]:

```
pd.concat([df1, df2], ignore_index=True)
```

In [599]:

```
df_cc = pd.concat([df1,df2], axis=1)
```

In [ ]:

```
df_cc
```

In [ ]:

```
pd.merge(df1, df3, on='id')
```

Puede ser interesante explorar los parámetros right_on y left_on.

Vamos a ver algunos ejemplos de Joins. Conviene tener delante los objetos originales para entender correctamente su funcionamiento.

In [ ]:

```
pd.merge(df1, df2, left_on='id',right_on='id', how='right')
```

In [ ]:

```
pd.merge(df1, df2, on='id', how='right')
```

In [ ]:

```
pd.merge(df1, df2, on='id', how='left')
```

In [ ]:

```
pd.merge(df1, df2, on='id', how='inner')
```

In [ ]:

```
pd.merge(df1, df2, left_on='id',right_on='id',how='outer',suffixes=('_i','_d')) # Como cambiar la nomenc
latura
```

In [ ]:

```
pd.merge(df1, df2, right_index=True, left_index=True)
```

14.Exportando

Ir al índice

A CSV

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_csv.html

In [609]:

```
df.to_csv(os.getcwd() + "\\mydf.csv")
```

A EXCEL

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_excel.html

In [610]:

```
df.to_excel(os.getcwd() + "\\mydf.xlsx")
```

In [ ]:

```

```