01_Exploracion

November 28, 2023

0.1 Operaciones Básicas: Exploración

En esta sesión vamos a tratar con operaciones básicas sobre `DataFrame` de forma explicita, pero recuerda que un `DataFrame` es una colección de objetos `Series`así que muchas veces trataremos con este tipo de objetos aunque no hagamos mención directa a ello.

0.1.1 Vistazo general

Manos a la obra, ejecuta la siguiente celda

[2]:
```
import numpy as np
import pandas as pd

df_aviones = pd.read_csv("./data/dataset_aviones.csv", index_col = "Id_vuelo")
```

Una vez tenemos un DataFrame, tenemos varias formas de explorarlo y ver su contenido. Veamos su aspecto general

[4]:
```
df_aviones.head()# con este metodo mostraremos su aspecto general viendo las 5␣
↪primeras filas
```

[4]:

Id_vuelo	Aircompany	Origen	Destino	Distancia	avion
Air_PaGi_10737	Airnar	París	Ginebra	411	Boeing 737
Fly_BaRo_10737	FlyQ	Bali	Roma	12738	Boeing 737
Tab_GiLo_11380	TabarAir	Ginebra	Los Angeles	9103	Airbus A380
Mol_PaCi_10737	MoldaviAir	París	Cincinnati	6370	Boeing 737
Tab_CiRo_10747	TabarAir	Cincinnati	Roma	7480	Boeing 747

Id_vuelo	con_escala	consumo_kg	duracion	ingresos
Air_PaGi_10737	False	1028.691900	51	14232.65
Fly_BaRo_10737	True	33479.132544	1167	468527.19
Tab_GiLo_11380	False	109439.907200	626	584789.19
Mol_PaCi_10737	False	17027.010000	503	233342.51
Tab_CiRo_10747	False	86115.744000	518	438535.07

[5]:
```
df_aviones.head(15)# podemos ponerle las filas que queremos ver
```

[5]:

Id_vuelo	Aircompany	Origen	Destino	Distancia	avion	\
Air_PaGi_10737	Airnar	París	Ginebra	411	Boeing 737	
Fly_BaRo_10737	FlyQ	Bali	Roma	12738	Boeing 737	
Tab_GiLo_11380	TabarAir	Ginebra	Los Angeles	9103	Airbus A380	
Mol_PaCi_10737	MoldaviAir	París	Cincinnati	6370	Boeing 737	
Tab_CiRo_10747	TabarAir	Cincinnati	Roma	7480	Boeing 747	
Mol_CaMe_10737	MoldaviAir	Cádiz	Melbourne	20029	Boeing 737	
Mol_PaLo_11320	MoldaviAir	París	Londres	344	Airbus A320	
Pam_PaMe_11380	PamPangea	París	Melbourne	16925	Airbus A380	
Pam_NuBa_10737	PamPangea	Nueva York	Bali	16589	Boeing 737	
Air_GiCa_11380	Airnar	Ginebra	Cádiz	1725	Airbus A380	
Tab_LoCi_10737	TabarAir	Los Angeles	Cincinnati	3073	Boeing 737	
Mol_LoBa_11380	MoldaviAir	Londres	Bali	12553	Airbus A380	
Tab_CiLo_10747	TabarAir	Cincinnati	Los Angeles	3073	Boeing 747	
Tab_GiLo_11380	TabarAir	Ginebra	Londres	739	Airbus A380	
Tab_CiRo_11320	TabarAir	Cincinnati	Roma	7480	Airbus A320	

Id_vuelo	con_escala	consumo_kg	duracion	ingresos
Air_PaGi_10737	False	1028.691900	51	14232.65
Fly_BaRo_10737	True	33479.132544	1167	468527.19
Tab_GiLo_11380	False	109439.907200	626	584789.19
Mol_PaCi_10737	False	17027.010000	503	233342.51
Tab_CiRo_10747	False	86115.744000	518	438535.07
Mol_CaMe_10737	True	53148.153240	1721	728045.68
Mol_PaLo_11320	False	915.246400	44	13805.52
Pam_PaMe_11380	True	217722.658400	1328	1056735.47
Pam_NuBa_10737	True	45277.618464	1459	600836.96
Air_GiCa_11380	False	20339.820000	135	110108.07
Tab_LoCi_10737	False	7915.433400	253	111056.67
Mol_LoBa_11380	False	156721.694400	856	764998.83
Tab_CiLo_10747	False	32758.180000	224	184079.01
Tab_GiLo_11380	False	8542.840000	69	46200.30
Tab_CiRo_11320	True	21087.855360	662	299451.12

[6]:
```
df_aviones.head(-5)#PREGUNTAR EN CLASE
```

[6]:

Id_vuelo	Aircompany	Origen	Destino	Distancia	avion	con_escala	consumo_kg	duracion	ingresos
Air_PaGi_10737	Airnar	París	Ginebra	411	Boeing 737	False	1028.691900	51	14232.65
Fly_BaRo_10737	FlyQ	Bali	Roma	12738	Boeing 737	True	33479.132544	1167	468527.19
Tab_GiLo_11380	TabarAir	Ginebra	Los Angeles	9103	Airbus A380	False	109439.907200	626	584789.19
Mol_PaCi_10737	MoldaviAir	París	Cincinnati	6370	Boeing 737	False	17027.010000	503	233342.51
Tab_CiRo_10747	TabarAir	Cincinnati	Roma	7480	Boeing 747	False	86115.744000	518	438535.07
…	…	…	…	…	…	…	…	…	…
Tab_GiLo_10737	TabarAir	Ginebra	Los Angeles	9103	Boeing 737	False	22783.898700	711	314159.35
Pam_BaNu_10747	PamPangea	Bali	Nueva York	16589	Boeing 747	True	185751.412496	1305	962466.12
Pam_NuMe_11380	PamPangea	Nueva York	Melbourne	16082	Airbus A380	True	195277.679168	1272	1011040.68
Air_GiCa_11320	Airnar	Ginebra	Cádiz	1725	Airbus A320	False	4762.725000	145	64711.93
Fly_GiBa_11380	FlyQ	Ginebra	Bali	12383	Airbus A380	False	156030.753200	845	795002.13

[1195 rows x 9 columns]

[7]:
```
df_aviones.tail(10)# muestra las ultimas 10 filas
```

[7]:

Id_vuelo	Aircompany	Origen	Destino	Distancia	avion
Tab_GiLo_10737	TabarAir	Ginebra	Los Angeles	9103	Boeing 737
Pam_BaNu_10747	PamPangea	Bali	Nueva York	16589	Boeing 747
Pam_NuMe_11380	PamPangea	Nueva York	Melbourne	16082	Airbus A380
Air_GiCa_11320	Airnar	Ginebra	Cádiz	1725	Airbus A320
Fly_GiBa_11380	FlyQ	Ginebra	Bali	12383	Airbus A380
Tab_LoLo_11320	TabarAir	Los Angeles	Londres	8785	Airbus A320
Mol_CiLo_10737	MoldaviAir	Cincinnati	Londres	6284	Boeing 737
Fly_RoCi_11320	FlyQ	Roma	Cincinnati	7480	Airbus A320
Tab_RoLo_10747	TabarAir	Roma	Londres	1433	Boeing 747
Air_PaLo_10737	Airnar	París	Los Angeles	9099	Boeing 737

Id_vuelo	con_escala	consumo_kg	duracion	ingresos
Tab_GiLo_10737	False	22783.898700	711	314159.35
Pam_BaNu_10747	True	185751.412496	1305	962466.12
Pam_NuMe_11380	True	195277.679168	1272	1011040.68
Air_GiCa_11320	False	4762.725000	145	64711.93
Fly_GiBa_11380	False	156030.753200	845	795002.13
Tab_LoLo_11320	True	24766.953120	756	340889.30
Mol_CiLo_10737	False	16491.729600	497	222424.54
Fly_RoCi_11320	True	19721.049920	662	285377.03
Tab_RoLo_10747	False	15734.053400	115	86373.94
Air_PaLo_10737	False	22331.675700	711	317996.77

[8]:
```
len(df_aviones)# nos dice el total de filas con informacion de viajes
```

[8]:
```
1200
```

0.1.2 Descripción inicial

Lo primero en general es qué columnas tiene:

[9]:
```
df_aviones.columns
```

[9]:
```
Index(['Aircompany', 'Origen', 'Destino', 'Distancia', 'avion', 'con_escala',
       'consumo_kg', 'duracion', 'ingresos'],
      dtype='object')
```

Una descripción general matemática de los valores numéricos:

[10]:
```
df_aviones.describe()# nos da columnas con valores numericos, y nos calcula el␣
↪nuemro total de viejes, la media, la desviacion standar, porcentajes␣
↪diversos el min y el maximo
```

[10]:

	Distancia	consumo_kg	duracion	ingresos
count	1200.000000	1200.000000	1200.000000	1.200000e+03
mean	8231.710000	68736.758895	651.153333	4.248612e+05
std	5567.286768	67605.206302	454.035184	3.184731e+05
min	344.000000	835.920000	42.000000	1.169364e+04
25%	3073.000000	15733.520400	224.000000	1.586815e+05
50%	6969.000000	38315.157192	572.000000	3.853903e+05
75%	12738.000000	120858.499500	1053.000000	6.899085e+05
max	20029.000000	264876.314560	1721.000000	1.295516e+06

Si quiero ver los tipos de cada columna

[11]:
```
df_aviones.dtypes
```

[11]:
```
Aircompany     object
Origen         object
```

```
Destino        object
Distancia       int64
avion          object
con_escala       bool
consumo_kg    float64
duracion        int64
ingresos      float64
dtype: object
```

Los tipos en Pandas se heredan parcialmente de numpy, por eso tienes int64, float64, pero además ves que los tipos string (y aquellas columnas que tengan tipos mezclados) se denominan object y que luego al tratar cada valor ya interpretará su tipo.

Ahora una descripción más completa, dentro de su generalidad, con el método info:

[13]:
```
df_aviones.info()#metodo:  el orden, el nombre, valores no nulos, y objetos de␣
↪cada tipo, que nos sirve para limpiar con los valores nulos (en otros spring)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1200 entries, Air_PaGi_10737 to Air_PaLo_10737
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Aircompany  1200 non-null   object
 1   Origen      1200 non-null   object
 2   Destino     1200 non-null   object
 3   Distancia   1200 non-null   int64
 4   avion       1200 non-null   object
 5   con_escala  1200 non-null   bool
 6   consumo_kg  1200 non-null   float64
 7   duracion    1200 non-null   int64
 8   ingresos    1200 non-null   float64
dtypes: bool(1), float64(2), int64(2), object(4)
memory usage: 85.5+ KB
```

[15]:
```
df_aviones.info# atributo # preguntar EN CLASE
```

[15]:
```
<bound method DataFrame.info of                Aircompany       Origen
Destino  Distancia        avion  \
Id_vuelo
Air_PaGi_10737      Airnar        París      Ginebra        411   Boeing 737
Fly_BaRo_10737        FlyQ         Bali         Roma      12738   Boeing 737
Tab_GiLo_11380    TabarAir      Ginebra  Los Angeles       9103  Airbus A380
Mol_PaCi_10737  MoldaviAir        París   Cincinnati       6370   Boeing 737
Tab_CiRo_10747    TabarAir   Cincinnati         Roma       7480   Boeing 747
…                    …          …          …        …          …
Tab_LoLo_11320    TabarAir  Los Angeles      Londres       8785  Airbus A320
Mol_CiLo_10737  MoldaviAir   Cincinnati      Londres       6284   Boeing 737
Fly_RoCi_11320        FlyQ         Roma   Cincinnati       7480  Airbus A320
```

```
Tab_RoLo_10747    TabarAir         Roma      Londres       1433   Boeing 747
Air_PaLo_10737      Airnar        París  Los Angeles       9099   Boeing 737

                con_escala     consumo_kg  duracion   ingresos
Id_vuelo
Air_PaGi_10737       False    1028.691900        51   14232.65
Fly_BaRo_10737        True   33479.132544      1167  468527.19
Tab_GiLo_11380       False  109439.907200       626  584789.19
Mol_PaCi_10737       False   17027.010000       503  233342.51
Tab_CiRo_10747       False   86115.744000       518  438535.07
…                        …            …       …         …
Tab_LoLo_11320        True   24766.953120       756  340889.30
Mol_CiLo_10737       False   16491.729600       497  222424.54
Fly_RoCi_11320        True   19721.049920       662  285377.03
Tab_RoLo_10747       False   15734.053400       115   86373.94
Air_PaLo_10737       False   22331.675700       711  317996.77

[1200 rows x 9 columns]>
```

0.1.3 Rascando los valores de las columnas

Pero si ahora quiero entrar en más detalle, ¿cómo puedo hacer una primera observación de los valores de una columna?

[16]:
```
df_aviones["Aircompany"].unique()# para saber el nombre de las compañias aereas␣
↪hay en el data set
```

[16]:
```
array(['Airnar', 'FlyQ', 'TabarAir', 'MoldaviAir', 'PamPangea'],
      dtype=object)
```

[18]:
```
df_aviones["avion"].unique()# tipos der aviones tiene el dataset
```

[18]:
```
array(['Boeing 737', 'Airbus A380', 'Boeing 747', 'Airbus A320'],
      dtype=object)
```

Pero igual quiero saber cómo están distribuidos

[19]:
```
df_aviones["Aircompany"].value_counts()# nuemros de vuelos que hay de cada␣
↪compañia
```

[19]:
```
Aircompany
TabarAir      271
MoldaviAir    264
PamPangea     231
Airnar        218
FlyQ          216
Name: count, dtype: int64
```

Fijate en que estos métodos, unique y value_counts son realmente métodos de series (porque al

escoger la columna primero estamos escogiendo una serie de pandas) y por tanto se pueden aplicar a cualquier serie

[20]:
```
serie= pd.Series(np.random.randint(1,5,50))
serie
```

[20]:
```
0     3
1     3
2     3
3     2
4     3
5     3
6     4
7     2
8     2
9     3
10    2
11    3
12    1
13    2
14    4
15    3
16    1
17    1
18    1
19    2
20    1
21    4
22    1
23    2
24    4
25    2
26    1
27    4
28    1
29    1
30    1
31    1
32    1
33    4
34    4
35    4
36    3
37    4
38    2
39    1
40    1
```

```
41    2
42    3
43    4
44    4
45    4
46    4
47    3
48    1
49    4
dtype: int32
```

[21]:
```
serie.unique()# nos dara el total de valores unicos divididos
```

[21]:
```
array([3, 2, 4, 1])
```

[22]:
```
serie.value_counts()# distribucion de la serie 15 nuemro 1, etc
```

[22]:
```
1    15
4    14
3    11
2    10
Name: count, dtype: int64
```

[25]:
```
serie.tail()# total de elementos que tiene cada columna
```

[25]:
```
45    4
46    4
47    3
48    1
49    4
dtype: int32
```

[26]:
```
serie.info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 50 entries, 0 to 49
Series name: None
Non-Null Count  Dtype
--------------  -----
50 non-null     int32
dtypes: int32(1)
memory usage: 328.0 bytes
```

[28]:
```
serie.head()
```

[28]:
```
0    3
1    3
2    3
```

```
3    2
4    3
dtype: int32
```

[30]:
```
serie.describe()
```

[30]:
```
count    50.00000
mean      2.48000
std       1.19932
min       1.00000
25%       1.00000
50%       2.50000
75%       4.00000
max       4.00000
dtype: float64
```

[ ]:
```

```

02_Operaciones_Basicas

November 28, 2023

0.1 Operaciones Básicas (II)

En esta sesión volvemos a las operaciones básicas, pero ya no de exploración como en la sesión anterior. Ahora nos centraremos en qué posibilidades básicas hay para trabajar con las columnas, principalmente, de un `DataFrame`, comencemos creando columnas.

0.1.1 Columnas

Como en casi todos los notebooks de esta unidad, jugaremos con el dataset de aviones. Ejecuta la siguiente celda:

[2]:
```
import numpy as np
import pandas as pd

df_aviones = pd.read_csv("./data/dataset_inicial_aviones.csv", index_col =␣
 ↪"Id_vuelo")
```

Veamos sus columnas y digamos que nos gustaría añadir una columna nueva que por ahora tenga el mismo valor para todas. Por ejemplo, una columna booleana que nos diga si el viaje llego en hora.

[3]:
```
df_aviones.head()
```

[3]:

Id_vuelo	Aircompany	Origen	Destino	Distancia	avion
Air_PaGi_10737	Airnar	París	Ginebra	411	Boeing 737
Fly_BaRo_10737	FlyQ	Bali	Roma	12738	Boeing 737
Tab_GiLo_11380	TabarAir	Ginebra	Los Angeles	9103	Airbus A380

```
Mol_PaCi_10737  MoldaviAir       París   Cincinnati       6370   Boeing 737
Tab_CiRo_10747    TabarAir  Cincinnati         Roma       7480   Boeing 747

                   consumo_kg  duracion
Id_vuelo
Air_PaGi_10737    1028.691900        51
Fly_BaRo_10737   33479.132544      1167
Tab_GiLo_11380  109439.907200       626
Mol_PaCi_10737   17027.010000       503
Tab_CiRo_10747   86115.744000       518
```

[4]:
```
df_aviones.columns
```

[4]:
```
Index(['Aircompany', 'Origen', 'Destino', 'Distancia', 'avion', 'consumo_kg',
       'duracion'],
      dtype='object')
```

una columna booleana que nos diga si el viaje llego en hora

[5]:
```
df_aviones["En_hora"]= True# con el metodo de un doccionario
df_aviones# si no ponemos ningun metodo ni atributo al DF, nos dara las 5␣
 ↪primeras filas y las 5 ultimas(esto solo sire notebook no es asi script .py)
```

[5]:
```
                Aircompany       Origen      Destino  Distancia        avion  \
Id_vuelo
Air_PaGi_10737      Airnar        París      Ginebra        411   Boeing 737
Fly_BaRo_10737        FlyQ         Bali         Roma      12738   Boeing 737
Tab_GiLo_11380    TabarAir      Ginebra  Los Angeles       9103  Airbus A380
Mol_PaCi_10737  MoldaviAir        París   Cincinnati       6370   Boeing 737
Tab_CiRo_10747    TabarAir   Cincinnati         Roma       7480   Boeing 747
…                    …          …           …         …          …
Tab_LoLo_11320    TabarAir  Los Angeles      Londres       8785  Airbus A320
Mol_CiLo_10737  MoldaviAir   Cincinnati      Londres       6284   Boeing 737
Fly_RoCi_11320        FlyQ         Roma   Cincinnati       7480  Airbus A320
Tab_RoLo_10747    TabarAir         Roma      Londres       1433   Boeing 747
Air_PaLo_10737      Airnar        París  Los Angeles       9099   Boeing 737

                   consumo_kg  duracion  En_hora
Id_vuelo
Air_PaGi_10737    1028.691900        51     True
Fly_BaRo_10737   33479.132544      1167     True
Tab_GiLo_11380  109439.907200       626     True
Mol_PaCi_10737   17027.010000       503     True
Tab_CiRo_10747   86115.744000       518     True
…                         …         …        …
Tab_LoLo_11320   24766.953120       756     True
Mol_CiLo_10737   16491.729600       497     True
Fly_RoCi_11320   19721.049920       662     True
```

```
Tab_RoLo_10747   15734.053400       115     True
Air_PaLo_10737   22331.675700       711     True

[1200 rows x 8 columns]
```

Y ahora algo más útil de forma directa, el consumo por kilometro:

[6]:
```
df_aviones["consumo_por_km"] = df_aviones["consumo_kg"] /␣
 ↪df_aviones["Distancia"]# divide elemento a elmento
df_aviones["consumo_por_km"] # si lo hago asi, me devuelve el resultado como si␣
 ↪fuera una serie panda
```

[6]:
```
Id_vuelo
Air_PaGi_10737     2.502900
Fly_BaRo_10737     2.628288
Tab_GiLo_11380    12.022400
Mol_PaCi_10737     2.673000
Tab_CiRo_10747    11.512800
                     …
Tab_LoLo_11320     2.819232
Mol_CiLo_10737     2.624400
Fly_RoCi_11320     2.636504
Tab_RoLo_10747    10.979800
Air_PaLo_10737     2.454300
Name: consumo_por_km, Length: 1200, dtype: float64
```

[7]:
```
#df_aviones# asi nos dara como dataframe
display(df_aviones)# tb nos dara la vision de dataframa, pero con print no dara␣
 ↪tb la vision como  serie panda
```

```
                Aircompany       Origen      Destino  Distancia        avion  \
Id_vuelo
Air_PaGi_10737      Airnar        París      Ginebra        411   Boeing 737
Fly_BaRo_10737        FlyQ         Bali         Roma      12738   Boeing 737
Tab_GiLo_11380    TabarAir      Ginebra  Los Angeles       9103  Airbus A380
Mol_PaCi_10737  MoldaviAir        París   Cincinnati       6370   Boeing 737
Tab_CiRo_10747    TabarAir   Cincinnati         Roma       7480   Boeing 747
…                      …            …            …          …            …
Tab_LoLo_11320    TabarAir  Los Angeles      Londres       8785  Airbus A320
Mol_CiLo_10737  MoldaviAir   Cincinnati      Londres       6284   Boeing 737
Fly_RoCi_11320        FlyQ         Roma   Cincinnati       7480  Airbus A320
Tab_RoLo_10747    TabarAir         Roma      Londres       1433   Boeing 747
Air_PaLo_10737      Airnar        París  Los Angeles       9099   Boeing 737

                  consumo_kg  duracion  En_hora  consumo_por_km
Id_vuelo
Air_PaGi_10737   1028.691900        51     True        2.502900
Fly_BaRo_10737  33479.132544      1167     True        2.628288
```

```
Tab_GiLo_11380  109439.907200       626     True       12.022400
Mol_PaCi_10737   17027.010000       503     True        2.673000
Tab_CiRo_10747   86115.744000       518     True       11.512800
…                          …       …       …              …
Tab_LoLo_11320   24766.953120       756     True        2.819232
Mol_CiLo_10737   16491.729600       497     True        2.624400
Fly_RoCi_11320   19721.049920       662     True        2.636504
Tab_RoLo_10747   15734.053400       115     True       10.979800
Air_PaLo_10737   22331.675700       711     True        2.454300

[1200 rows x 9 columns]
```

Podríamos haber usado los atributos:

[8]:
```
df_aviones["consumo_por_Km"] = df_aviones.consumo_kg / df_aviones.Distancia
df_aviones
```

[8]:
```
                Aircompany       Origen      Destino  Distancia        avion  \
Id_vuelo
Air_PaGi_10737      Airnar        París      Ginebra        411   Boeing 737
Fly_BaRo_10737        FlyQ         Bali         Roma      12738   Boeing 737
Tab_GiLo_11380    TabarAir      Ginebra  Los Angeles       9103  Airbus A380
Mol_PaCi_10737  MoldaviAir        París   Cincinnati       6370   Boeing 737
Tab_CiRo_10747    TabarAir   Cincinnati         Roma       7480   Boeing 747
…                      …           …            …          …            …
Tab_LoLo_11320    TabarAir  Los Angeles      Londres       8785  Airbus A320
Mol_CiLo_10737  MoldaviAir   Cincinnati      Londres       6284   Boeing 737
Fly_RoCi_11320        FlyQ         Roma   Cincinnati       7480  Airbus A320
Tab_RoLo_10747    TabarAir         Roma      Londres       1433   Boeing 747
Air_PaLo_10737      Airnar        París  Los Angeles       9099   Boeing 737

                   consumo_kg  duracion  En_hora  consumo_por_km  \
Id_vuelo
Air_PaGi_10737    1028.691900        51     True        2.502900
Fly_BaRo_10737   33479.132544      1167     True        2.628288
Tab_GiLo_11380  109439.907200       626     True       12.022400
Mol_PaCi_10737   17027.010000       503     True        2.673000
Tab_CiRo_10747   86115.744000       518     True       11.512800
…                         …       …       …              …
Tab_LoLo_11320   24766.953120       756     True        2.819232
Mol_CiLo_10737   16491.729600       497     True        2.624400
Fly_RoCi_11320   19721.049920       662     True        2.636504
Tab_RoLo_10747   15734.053400       115     True       10.979800
Air_PaLo_10737   22331.675700       711     True        2.454300

                consumo_por_Km
Id_vuelo
Air_PaGi_10737        2.502900
```

```
Fly_BaRo_10737        2.628288
Tab_GiLo_11380       12.022400
Mol_PaCi_10737        2.673000
Tab_CiRo_10747       11.512800
…                          …
Tab_LoLo_11320        2.819232
Mol_CiLo_10737        2.624400
Fly_RoCi_11320        2.636504
Tab_RoLo_10747       10.979800
Air_PaLo_10737        2.454300

[1200 rows x 10 columns]
```

[ ]:

Pero quizá nos ha quedado un nombre de columna largo o creemos que algún nombre podría estar mejor expresado de otra forma... ¿Cómo cambiamos los nombres de las columnas?

[9]:
```
df_aviones.rename(columns ={"consumo_por_km": "Consumo/km", "avion":"Avion",␣
↪"consumo_kg":"Consumo/kg"}) # esto no cambia la variable aviones solo lo␣
↪muestra , si qeuisieramos cambair la variable tendria que reasignar o usar␣
↪el atributo inplace = True
```

[9]:
```
                Aircompany       Origen      Destino  Distancia        Avion  \
Id_vuelo
Air_PaGi_10737      Airnar        París      Ginebra        411   Boeing 737
Fly_BaRo_10737        FlyQ         Bali         Roma      12738   Boeing 737
Tab_GiLo_11380   TabarAir      Ginebra  Los Angeles       9103  Airbus A380
Mol_PaCi_10737  MoldaviAir        París   Cincinnati       6370   Boeing 737
Tab_CiRo_10747   TabarAir   Cincinnati         Roma       7480   Boeing 747
…                     …           …           …         …           …
Tab_LoLo_11320   TabarAir  Los Angeles      Londres       8785  Airbus A320
Mol_CiLo_10737  MoldaviAir   Cincinnati      Londres       6284   Boeing 737
Fly_RoCi_11320        FlyQ         Roma   Cincinnati       7480  Airbus A320
Tab_RoLo_10747   TabarAir         Roma      Londres       1433   Boeing 747
Air_PaLo_10737      Airnar        París  Los Angeles       9099   Boeing 737

                  Consumo/kg  duracion  En_hora  Consumo/km  consumo_por_Km
Id_vuelo
Air_PaGi_10737    1028.691900        51     True    2.502900        2.502900
Fly_BaRo_10737   33479.132544      1167     True    2.628288        2.628288
Tab_GiLo_11380  109439.907200       626     True   12.022400       12.022400
Mol_PaCi_10737   17027.010000       503     True    2.673000        2.673000
Tab_CiRo_10747   86115.744000       518     True   11.512800       11.512800
…                         …         …        …           …               …
Tab_LoLo_11320   24766.953120       756     True    2.819232        2.819232
Mol_CiLo_10737   16491.729600       497     True    2.624400        2.624400
Fly_RoCi_11320   19721.049920       662     True    2.636504        2.636504
```

```
Tab_RoLo_10747   15734.053400       115     True   10.979800       10.979800
Air_PaLo_10737   22331.675700       711     True    2.454300        2.454300

[1200 rows x 10 columns]
```

[10]:
```
df_aviones
```

[10]:
```
                Aircompany       Origen      Destino  Distancia        avion  \
Id_vuelo
Air_PaGi_10737      Airnar        París      Ginebra        411   Boeing 737
Fly_BaRo_10737        FlyQ         Bali         Roma      12738   Boeing 737
Tab_GiLo_11380    TabarAir      Ginebra  Los Angeles       9103  Airbus A380
Mol_PaCi_10737  MoldaviAir        París   Cincinnati       6370   Boeing 737
Tab_CiRo_10747    TabarAir   Cincinnati         Roma       7480   Boeing 747
…                     …          …           …        …          …
Tab_LoLo_11320    TabarAir  Los Angeles      Londres       8785  Airbus A320
Mol_CiLo_10737  MoldaviAir   Cincinnati      Londres       6284   Boeing 737
Fly_RoCi_11320        FlyQ         Roma   Cincinnati       7480  Airbus A320
Tab_RoLo_10747    TabarAir         Roma      Londres       1433   Boeing 747
Air_PaLo_10737      Airnar        París  Los Angeles       9099   Boeing 737

                   consumo_kg  duracion  En_hora  consumo_por_km  \
Id_vuelo
Air_PaGi_10737    1028.691900        51     True        2.502900
Fly_BaRo_10737   33479.132544      1167     True        2.628288
Tab_GiLo_11380  109439.907200       626     True       12.022400
Mol_PaCi_10737   17027.010000       503     True        2.673000
Tab_CiRo_10747   86115.744000       518     True       11.512800
…                         …       …      …                …
Tab_LoLo_11320   24766.953120       756     True        2.819232
Mol_CiLo_10737   16491.729600       497     True        2.624400
Fly_RoCi_11320   19721.049920       662     True        2.636504
Tab_RoLo_10747   15734.053400       115     True       10.979800
Air_PaLo_10737   22331.675700       711     True        2.454300

                consumo_por_Km
Id_vuelo
Air_PaGi_10737        2.502900
Fly_BaRo_10737        2.628288
Tab_GiLo_11380       12.022400
Mol_PaCi_10737        2.673000
Tab_CiRo_10747       11.512800
…                          …
Tab_LoLo_11320        2.819232
Mol_CiLo_10737        2.624400
Fly_RoCi_11320        2.636504
Tab_RoLo_10747       10.979800
```

```
Air_PaLo_10737        2.454300

[1200 rows x 10 columns]
```

O lo reasigno o bien utilizo el argumento `inplace` que es un argumento que existe en muchos métodos de los `DataFrame`.

[11]:
```
# Reasignado
df_aviones_2 = df_aviones.rename(columns ={"consumo_por_km": "Consumo/km",␣
 ↪"avion":"Avion", "consumo_kg":"Consumo/kg"})
df_aviones_2
```

[11]:

Id_vuelo	Aircompany	Origen	Destino	Distancia	Avion	Consumo/kg	duracion	En_hora	Consumo/km	consumo_por_Km
Air_PaGi_10737	Airnar	París	Ginebra	411	Boeing 737	1028.691900	51	True	2.502900	2.502900
Fly_BaRo_10737	FlyQ	Bali	Roma	12738	Boeing 737	33479.132544	1167	True	2.628288	2.628288
Tab_GiLo_11380	TabarAir	Ginebra	Los Angeles	9103	Airbus A380	109439.907200	626	True	12.022400	12.022400
Mol_PaCi_10737	MoldaviAir	París	Cincinnati	6370	Boeing 737	17027.010000	503	True	2.673000	2.673000
Tab_CiRo_10747	TabarAir	Cincinnati	Roma	7480	Boeing 747	86115.744000	518	True	11.512800	11.512800
…	…	…	…	…	…	…	…	…	…	…
Tab_LoLo_11320	TabarAir	Los Angeles	Londres	8785	Airbus A320	24766.953120	756	True	2.819232	2.819232
Mol_CiLo_10737	MoldaviAir	Cincinnati	Londres	6284	Boeing 737	16491.729600	497	True	2.624400	2.624400
Fly_RoCi_11320	FlyQ	Roma	Cincinnati	7480	Airbus A320	19721.049920	662	True	2.636504	2.636504
Tab_RoLo_10747	TabarAir	Roma	Londres	1433	Boeing 747	15734.053400	115	True	10.979800	10.979800
Air_PaLo_10737	Airnar	París	Los Angeles	9099	Boeing 737	22331.675700	711	True	2.454300	2.454300

[1200 rows x 10 columns]

[15]:
```
# Inplace , cambia el archivo original
df_aviones = df_aviones.rename(columns ={"consumo_por_km": "Consumo/km",␣
 ↪"avion":"Avion", "consumo_kg":"Consumo/kg"},inplace= True)# columns porque␣
 ↪cambia nombre de columnas si fura fila fill
```

```
---------------------------------------------------------------------------
```

```
AttributeError                            Traceback (most recent call last)
e:
 ↪\Cursos\BC_Data_Science\Repositorio\ONLINE_DS_THEBRIDGE_V\ONLINE_DS_THEBRIDGE_V-1\SPRING␣
 ↪4\UNIT 2\02_Operaciones_Basicas.ipynb Celda 23 line 2

      <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/ONLINE_DS_THEBRIDGE_V-1/SPRING%204/UNIT%202/
 ↪02_Operaciones_Basicas.ipynb#X31sZmlsZQ%3D%3D?line=0'>1</a> # Inplace , cambi␣
 ↪el archivo original
----> <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/ONLINE_DS_THEBRIDGE_V-1/SPRING%204/UNIT%202/
 ↪02_Operaciones_Basicas.ipynb#X31sZmlsZQ%3D%3D?line=1'>2</a> df_aviones =␣
 ↪df_aviones.rename(columns ={"consumo_por_km": "Consumo/km", "avion":"Avion",␣
 ↪"consumo_kg":"Consumo/kg"},inplace= True)# columns porque cambia nombre de␣
 ↪columnas si fura fila fill


AttributeError: 'NoneType' object has no attribute 'rename'
```

0.1.2 Operaciones Sencillas

Veamos para terminar algunas operaciones sencillas de agregación que te sonarán porque se comparten casi en su totalidad con numpy. Para ello iremos contestando a una serie de preguntas [Que es otra forma de explorar los datos]

[14]:
```
# Cual es la mayor distancia recorrida
df_aviones["Distancia"].max()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
e:
 ↪\Cursos\BC_Data_Science\Repositorio\ONLINE_DS_THEBRIDGE_V\ONLINE_DS_THEBRIDGE_V-1\SPRING␣
 ↪4\UNIT 2\02_Operaciones_Basicas.ipynb Celda 26 line 2

      <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/ONLINE_DS_THEBRIDGE_V-1/SPRING%204/UNIT%202/
 ↪02_Operaciones_Basicas.ipynb#X34sZmlsZQ%3D%3D?line=0'>1</a> # Cual es la mayo␣
 ↪distancia recorrida
----> <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/ONLINE_DS_THEBRIDGE_V-1/SPRING%204/UNIT%202/
 ↪02_Operaciones_Basicas.ipynb#X34sZmlsZQ%3D%3D?line=1'>2</a>␣
 ↪df_aviones["Distancia"].max()


TypeError: 'NoneType' object is not subscriptable
```

[ ]:
```
# Cual es el menor consumo
df_aviones["Consumo/Kg"].min()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
```

```
e:\Cursos\BC_Data_Science\Repositorio\ONLINE_DS_THEBRIDGE_V\SPRING 4\UNIT␣
 ↪2\02_Operaciones_Basicas.ipynb Celda 27 line 2

     <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/02_Operaciones_Basicas.
 ↪ipynb#X35sZmlsZQ%3D%3D?line=0'>1</a> # Cual es el menor consumo
----> <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/02_Operaciones_Basicas.
 ↪ipynb#X35sZmlsZQ%3D%3D?line=1'>2</a> df_aviones["Consumo/Kg"].min()

TypeError: 'NoneType' object is not subscriptable
```

[ ]:
```
# Cuanta distancia se han recorrido en los 1200 vuelos
df_aviones["Distancia"].sum()
```

[ ]:
```
9878052
```

[ ]:
```
# Cual es la media recorrida por estos viajes
df_aviones["Distancia"].mean()
```

[ ]:
```
8231.71
```

[ ]:
```
# Y el consumo medio
df_aviones["Consumo/Kg"].mean()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\indexes\
 ↪py:3790, in Index.get_loc(self, key)
   3789 try:
-> 3790     return self._engine.get_loc(casted_key)
   3791 except KeyError as err:

File index.pyx:152, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:181, in pandas._libs.index.IndexEngine.get_loc()

File pandas\_libs\hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

File pandas\_libs\hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

KeyError: 'Consumo/Kg'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
```

```
e:\Cursos\BC_Data_Science\Repositorio\ONLINE_DS_THEBRIDGE_V\SPRING 4\UNIT␣
 ↪2\02_Operaciones_Basicas.ipynb Celda 30 line 2

     <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/02_Operaciones_Basicas.
 ↪ipynb#X41sZmlsZQ%3D%3D?line=0'>1</a> # Y el consumo medio
----> <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/02_Operaciones_Basicas.
 ↪ipynb#X41sZmlsZQ%3D%3D?line=1'>2</a> df_aviones["Consumo/Kg"].mean()

File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\frame.
 ↪py:3893, in DataFrame.__getitem__(self, key)
  3891 if self.columns.nlevels > 1:
  3892     return self._getitem_multilevel(key)
-> 3893 indexer = self.columns.get_loc(key)
  3894 if is_integer(indexer):
  3895     indexer = [indexer]

File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\indexes\
 ↪py:3797, in Index.get_loc(self, key)
  3792     if isinstance(casted_key, slice) or (
  3793         isinstance(casted_key, abc.Iterable)
  3794         and any(isinstance(x, slice) for x in casted_key)
  3795     ):
  3796         raise InvalidIndexError(key)
-> 3797     raise KeyError(key) from err
  3798 except TypeError:
  3799     # If we have a listlike key, _check_indexing_error will raise
  3800     #  InvalidIndexError. Otherwise we fall through and re-raise
  3801     #  the TypeError.
  3802     self._check_indexing_error(key)

KeyError: 'Consumo/Kg'
```

Bueno, como medidas agregadas están bien, pero si quiero algo más de detalle y sin entrar en como quedarnos con solo las filas que cumplan una condición también podemos hacer lo siguiente

[ ]:
```
# Cual es el viaje con menor consumo
viaje = df_aviones["Consumo_kg"].idxmin()# el id de vuelo con menor consumo
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
e:\Cursos\BC_Data_Science\Repositorio\ONLINE_DS_THEBRIDGE_V\SPRING 4\UNIT␣
 ↪2\02_Operaciones_Basicas.ipynb Celda 32 line 2

     <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/02_Operaciones_Basicas.
 ↪ipynb#X43sZmlsZQ%3D%3D?line=0'>1</a> # Cual es el viaje con menor consumo
```

```
----> <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/02_Operaciones_Basicas.
 ↪ipynb#X43sZmlsZQ%3D%3D?line=1'>2</a> viaje = df_aviones["Consumo_kg"].
 ↪idxmin()# el id de vuelo con menor consumo

NameError: name 'df_aviones' is not defined
```

[ ]:
```
df_aviones.loc(viaje)# aqui nos dara informacion de toda la fila del avion con␣
 ↪menor consumo (Mol PaLo_10737)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
e:\Cursos\BC_Data_Science\Repositorio\ONLINE_DS_THEBRIDGE_V\SPRING 4\UNIT␣
 ↪2\02_Operaciones_Basicas.ipynb Celda 33 line 1
----> <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/02_Operaciones_Basicas.
 ↪ipynb#X44sZmlsZQ%3D%3D?line=0'>1</a> df_aviones.loc(viaje)

NameError: name 'viaje' is not defined
```

[ ]:
```
# Cual es el avion con el mayor consumo medio
viaje = df_aviones["Consumo/km"].idxmax()# nos dara el veulo con mayor consumo␣
 ↪( MoL_MeCa_11380)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\indexes\
 ↪py:3790, in Index.get_loc(self, key)
   3789 try:
-> 3790     return self._engine.get_loc(casted_key)
   3791 except KeyError as err:

File index.pyx:152, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:181, in pandas._libs.index.IndexEngine.get_loc()

File pandas\_libs\hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

File pandas\_libs\hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

KeyError: 'Consumo/km'

The above exception was the direct cause of the following exception:
```

```
KeyError                                  Traceback (most recent call last)
e:\Cursos\BC_Data_Science\Repositorio\ONLINE_DS_THEBRIDGE_V\SPRING 4\UNIT␣
 ↪2\02_Operaciones_Basicas.ipynb Celda 34 line 2

      <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/02_Operaciones_Basicas.
 ↪ipynb#X45sZmlsZQ%3D%3D?line=0'>1</a> # Cual es el avion con el mayor consumo␣
 ↪medio
----> <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/02_Operaciones_Basicas.
 ↪ipynb#X45sZmlsZQ%3D%3D?line=1'>2</a> viaje = df_aviones["Consumo/km"].idxmax(

File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\frame.
 ↪py:3893, in DataFrame.__getitem__(self, key)
   3891 if self.columns.nlevels > 1:
   3892     return self._getitem_multilevel(key)
-> 3893 indexer = self.columns.get_loc(key)
   3894 if is_integer(indexer):
   3895     indexer = [indexer]

File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\indexes\
 ↪py:3797, in Index.get_loc(self, key)
   3792     if isinstance(casted_key, slice) or (
   3793         isinstance(casted_key, abc.Iterable)
   3794         and any(isinstance(x, slice) for x in casted_key)
   3795     ):
   3796         raise InvalidIndexError(key)
-> 3797     raise KeyError(key) from err
   3798 except TypeError:
   3799     # If we have a listlike key, _check_indexing_error will raise
   3800     #  InvalidIndexError. Otherwise we fall through and re-raise
   3801     #  the TypeError.
   3802     self._check_indexing_error(key)

KeyError: 'Consumo/km'
```

[ ]:
```
# si qyeremos mas informacion de la fila del vuelo:
df_aviones.loc[viaje]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
e:\Cursos\BC_Data_Science\Repositorio\ONLINE_DS_THEBRIDGE_V\SPRING 4\UNIT␣
 ↪2\02_Operaciones_Basicas.ipynb Celda 35 line 2

      <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/02_Operaciones_Basicas.
 ↪ipynb#X46sZmlsZQ%3D%3D?line=0'>1</a> # si qyeremos mas informacion de la fila␣
 ↪del vuelo:
```

```
----> <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/02_Operaciones_Basicas.
 ↪ipynb#X46sZmlsZQ%3D%3D?line=1'>2</a> df_aviones.loc[viaje]

NameError: name 'viaje' is not defined
```

[ ]:
```
# Pero si solo queremos ver el uno...
df_aviones.loc[viaje,"Avion"].unique()# nos solo el avion que gasto mayor␣
 ↪consumo. esto nos dara un array,
#si quisieramos que nos lo imprima normal un print:
valor_avion= df_aviones.loc[viaje,"Avion"].unique()[0]
print(valor_avion)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
e:\Cursos\BC_Data_Science\Repositorio\ONLINE_DS_THEBRIDGE_V\SPRING 4\UNIT␣
 ↪2\02_Operaciones_Basicas.ipynb Celda 36 line 2

      <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/02_Operaciones_Basicas.
 ↪ipynb#X50sZmlsZQ%3D%3D?line=0'>1</a> # Pero si solo queremos ver el uno…
----> <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/02_Operaciones_Basicas.
 ↪ipynb#X50sZmlsZQ%3D%3D?line=1'>2</a> df_aviones.loc[viaje,"Avion"].unique()#␣
 ↪nos solo el avion que gasto mayor consumo. esto nos dara un array,
      <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/02_Operaciones_Basicas.
 ↪ipynb#X50sZmlsZQ%3D%3D?line=2'>3</a> #si quisieramos que nos lo imprima norma␣
 ↪un print:
      <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/02_Operaciones_Basicas.
 ↪ipynb#X50sZmlsZQ%3D%3D?line=3'>4</a> valor_avion= df_aviones.
 ↪loc[viaje,"Avion"].unique()[0]

NameError: name 'viaje' is not defined
```

Por si quieres practicas, las siguientes agregaciones vienen con el paquete de Pandas:

Agregación	Descripción
`count()`	Número total de elementos
`first()`, `last()`	Primer y último elemento
`mean()`, `median()`	Media y mediana
`min()`, `max()`	Mínimo y máximo
`std()`, `var()`	Desviación estándar y varianza
`mad()`	Desviación media absoluta
`prod()`	Producto de todos los elementos
`sum()`	Suma de todos los elementos

Todos están presentes como objetos de `Dataframe` y `Series`.

[ ]:

[ ]:

[ ]:

03_Filtrado

November 28, 2023

0.1 Filtrado y Selección

En esta sesión vamos a tratar el tema del filtrado de forma que podamos obtener el conjunto de filas y de filas y columnas de un dataframe que cumplan una serie de condiciones y en la siguiente usaremos esto para poder hacer modificaciones selectivas en nuestro dataframe que nos vendrá muy bien en el futuro para tratar datos.

0.1.1 Filtrado de Filas

Como en casi todos los notebooks de esta unidad, jugaremos con el dataset de aviones. Ejecuta la siguiente celda:

[2]:
```
import numpy as np
import pandas as pd

df_aviones = pd.read_csv("./data/dataset_inicial_aviones.csv", index_col =␣
 ↪"Id_vuelo")
```

[ ]:
```

```

Y comencemos a filtrar y seleccionar, en concreto seleccionemos todos los viajes de la compañía con menor número de vuelos.

[3]:
```
# Primero miramos el número de vuelos para recordar algunos métodos de sesiones␣
 ↪anteriores
df_aviones["Aircompany"].value_counts()
```

[3]:
```
Aircompany
TabarAir      271
MoldaviAir    264
PamPangea     231
Airnar        218
FlyQ          216
Name: count, dtype: int64
```

[4]:
```
# Y ahora filtramos/seleccionamos solo las filas de esos vuelos
resultado = df_aviones.loc[df_aviones["Aircompany"]=="FlyQ"]# lo que hace toda
 ↪una serie de tru o false que despues lo va a usar el .loc que se quedara con
 ↪los valores True
display(resultado)
```

Id_vuelo	Aircompany	Origen	Destino	Distancia	avion	consumo_kg	duracion
Fly_BaRo_10737	FlyQ	Bali	Roma	12738	Boeing 737	33479.132544	1167
Fly_RoNu_11320	FlyQ	Roma	Nueva York	6877	Airbus A320	18131.238008	618
Fly_GiCi_10737	FlyQ	Ginebra	Cincinnati	6969	Boeing 737	18289.443600	549
Fly_GiRo_10737	FlyQ	Ginebra	Roma	698	Boeing 737	1763.985600	73
Fly_BaGi_10737	FlyQ	Bali	Ginebra	12383	Boeing 737	31607.260776	1140
…	…	…	…	…	…	…	…
Fly_NuBa_11380	FlyQ	Nueva York	Barcelona	6170	Airbus A380	76317.964000	431
Fly_NuRo_11320	FlyQ	Nueva York	Roma	6877	Airbus A320	19567.375672	618
Fly_BaRo_10747	FlyQ	Bali	Roma	12738	Boeing 747	141218.563200	1049
Fly_GiBa_11380	FlyQ	Ginebra	Bali	12383	Airbus A380	156030.753200	845
Fly_RoCi_11320	FlyQ	Roma	Cincinnati	7480	Airbus A320	19721.049920	662

[216 rows x 7 columns]

Nos ha devuelto un `DataFrame` con solo los viajes de esa compañía.

Si hubieramos querido mantener, por lo que fuera, la estructura completa con el resto de vuelos pero con un valor especial (o nulo), usaríamos where

[5]:
```
# Usando Where, sin cambiar valor
df_aviones.where(df_aviones["Aircompany"] =="FlyQ")# devuelve todas las 1200␣
 ↪lineas con la condicion que hemos puesto y el resto de valores NaN
```

[5]:
```
               Aircompany Origen     Destino  Distancia        avion  \
Id_vuelo
Air_PaGi_10737        NaN    NaN         NaN        NaN          NaN
Fly_BaRo_10737       FlyQ   Bali        Roma    12738.0   Boeing 737
Tab_GiLo_11380        NaN    NaN         NaN        NaN          NaN
Mol_PaCi_10737        NaN    NaN         NaN        NaN          NaN
Tab_CiRo_10747        NaN    NaN         NaN        NaN          NaN
…                     …      …           …          …            …
Tab_LoLo_11320        NaN    NaN         NaN        NaN          NaN
Mol_CiLo_10737        NaN    NaN         NaN        NaN          NaN
Fly_RoCi_11320       FlyQ   Roma  Cincinnati     7480.0  Airbus A320
Tab_RoLo_10747        NaN    NaN         NaN        NaN          NaN
Air_PaLo_10737        NaN    NaN         NaN        NaN          NaN

                   consumo_kg  duracion
Id_vuelo
Air_PaGi_10737            NaN       NaN
Fly_BaRo_10737  33479.132544    1167.0
Tab_GiLo_11380            NaN       NaN
Mol_PaCi_10737            NaN       NaN
Tab_CiRo_10747            NaN       NaN
…                         …         …
Tab_LoLo_11320            NaN       NaN
Mol_CiLo_10737            NaN       NaN
Fly_RoCi_11320  19721.049920     662.0
Tab_RoLo_10747            NaN       NaN
Air_PaLo_10737            NaN       NaN

[1200 rows x 7 columns]
```

[ ]:
```
```

[6]:
```
# Usando Where, cambiando valor
# si quiero cambiar ese NaN por "no usar" y quedarme con la lista completa solo␣
 ↪con los valores de la condicion con el resto no usar:
df_aviones.where(df_aviones["Aircompany"] =="FlyQ", "no usar")
```

[6]:
```
               Aircompany   Origen     Destino Distancia        avion  \
Id_vuelo
Air_PaGi_10737    no usar  no usar     no usar   no usar      no usar
Fly_BaRo_10737       FlyQ     Bali        Roma     12738   Boeing 737
Tab_GiLo_11380    no usar  no usar     no usar   no usar      no usar
Mol_PaCi_10737    no usar  no usar     no usar   no usar      no usar
```

```
Tab_CiRo_10747    no usar  no usar     no usar   no usar      no usar
…                   …      …          …        …          …
Tab_LoLo_11320    no usar  no usar     no usar   no usar      no usar
Mol_CiLo_10737    no usar  no usar     no usar   no usar      no usar
Fly_RoCi_11320       FlyQ     Roma  Cincinnati      7480  Airbus A320
Tab_RoLo_10747    no usar  no usar     no usar   no usar      no usar
Air_PaLo_10737    no usar  no usar     no usar   no usar      no usar

                  consumo_kg duracion
Id_vuelo
Air_PaGi_10737       no usar  no usar
Fly_BaRo_10737  33479.132544     1167
Tab_GiLo_11380       no usar  no usar
Mol_PaCi_10737       no usar  no usar
Tab_CiRo_10747       no usar  no usar
…                      …      …
Tab_LoLo_11320       no usar  no usar
Mol_CiLo_10737       no usar  no usar
Fly_RoCi_11320   19721.04992      662
Tab_RoLo_10747       no usar  no usar
Air_PaLo_10737       no usar  no usar

[1200 rows x 7 columns]
```

Evidentemente podemos hacer consultas mucho más complejas y podemos ir asignandolas a variables intermedias.

[7]:
```
df_aviones.avion.value_counts()
```

[7]:
```
avion
Boeing 747     344
Airbus A380    343
Boeing 737     315
Airbus A320    198
Name: count, dtype: int64
```

[8]:
```
# Quiero quedarme con los vuelos de los A380 de más de 10000 km de distancia␣
 ↪con destino a Melbourne
vuelos_A380 = df_aviones["avion"] =="Airbus A380"
mayor_10km = df_aviones["Distancia"] > 10000
a_Melbourne = df_aviones["Destino"] == "Melbourne" # & se le llama ampersan y␣
 ↪para el or se usa |
condicion = vuelos_A380 & mayor_10km & a_Melbourne
df_aviones.loc[condicion]
```

[8]:
```
                Aircompany      Origen    Destino  Distancia        avion  \
Id_vuelo
Pam_PaMe_11380   PamPangea       París  Melbourne      16925  Airbus A380
```

```
Mol_LoMe_11380  MoldaviAir     Londres  Melbourne      16900  Airbus A380
Mol_CaMe_11380  MoldaviAir       Cádiz  Melbourne      20029  Airbus A380
Mol_PaMe_11380  MoldaviAir       París  Melbourne      16925  Airbus A380
Mol_CiMe_11380  MoldaviAir  Cincinnati  Melbourne      15262  Airbus A380
Mol_CaMe_11380  MoldaviAir       Cádiz  Melbourne      20029  Airbus A380
Pam_GiMe_11380   PamPangea     Ginebra  Melbourne      16674  Airbus A380
Mol_CiMe_11380  MoldaviAir  Cincinnati  Melbourne      15262  Airbus A380
Pam_LoMe_11380   PamPangea     Londres  Melbourne      16900  Airbus A380
Pam_LoMe_11380   PamPangea     Londres  Melbourne      16900  Airbus A380
Pam_LoMe_11380   PamPangea     Londres  Melbourne      16900  Airbus A380
Mol_PaMe_11380  MoldaviAir       París  Melbourne      16925  Airbus A380
Pam_NuMe_11380   PamPangea  Nueva York  Melbourne      16082  Airbus A380
Mol_CaMe_11380  MoldaviAir       Cádiz  Melbourne      20029  Airbus A380
Pam_PaMe_11380   PamPangea       París  Melbourne      16925  Airbus A380
Mol_CiMe_11380  MoldaviAir  Cincinnati  Melbourne      15262  Airbus A380
Pam_PaMe_11380   PamPangea       París  Melbourne      16925  Airbus A380
Mol_CiMe_11380  MoldaviAir  Cincinnati  Melbourne      15262  Airbus A380
Pam_GiMe_11380   PamPangea     Ginebra  Melbourne      16674  Airbus A380
Pam_NuMe_11380   PamPangea  Nueva York  Melbourne      16082  Airbus A380
Mol_PaMe_11380  MoldaviAir       París  Melbourne      16925  Airbus A380
Pam_NuMe_11380   PamPangea  Nueva York  Melbourne      16082  Airbus A380

                  consumo_kg  duracion
Id_vuelo
Pam_PaMe_11380  217722.658400      1328
Mol_LoMe_11380  213337.488000      1326
Mol_CaMe_11380  264876.314560      1535
Mol_PaMe_11380  207548.702400      1328
Mol_CiMe_11380  199999.596992      1217
Mol_CaMe_11380  248020.549088      1535
Pam_GiMe_11380  216498.417408      1311
Mol_CiMe_11380  190825.303552      1217
Pam_LoMe_11380  213337.488000      1326
Pam_LoMe_11380  203178.560000      1326
Pam_LoMe_11380  205210.345600      1326
Mol_PaMe_11380  203479.120000      1328
Pam_NuMe_11380  193344.236800      1272
Mol_CaMe_11380  245612.582592      1535
Pam_PaMe_11380  211618.284800      1328
Mol_CiMe_11380  201834.455680      1217
Pam_PaMe_11380  203479.120000      1328
Mol_CiMe_11380  187155.586176      1217
Pam_GiMe_11380  204470.727552      1311
Pam_NuMe_11380  199144.563904      1272
Mol_PaMe_11380  209583.493600      1328
Pam_NuMe_11380  195277.679168      1272
```

Fijate que en este caso y así será para condiciones en lo que se compara son `Series` se usa `&` en vez

de **and** y **|** en vez de **or**

[9]:
```
# Seleccionemos vuelos de PamPangea con destino Ginebra o salida en Nueva York
company = df_aviones["Aircompany"] == "PamPangea"
a_Ginebra = df_aviones["Destino"] == "Ginebra"
de_NY = df_aviones["Origen"] == "Nueva York"
condicion = company & ( a_Ginebra | de_NY)# para evitar preferencias se pone␣
 ↪entreparentesis las dos condiciones or
df_aviones.loc[condicion]# por salud mental en el futuro poner siempre loc␣
 ↪auqnue en este caso fucnione sin el loc tb
```

[9]:

Id_vuelo	Aircompany	Origen	Destino	Distancia	avion	consumo_kg	duracion
Pam_NuBa_10737	PamPangea	Nueva York	Bali	16589	Boeing 737	45277.618464	1459
Pam_MeGi_11380	PamPangea	Melbourne	Ginebra	16674	Airbus A380	220507.647360	1311
Pam_NuMe_10747	PamPangea	Nueva York	Melbourne	16082	Boeing 747	183640.229344	1272
Pam_NuPa_11380	PamPangea	Nueva York	París	5835	Airbus A380	72174.282000	409
Pam_PaGi_11380	PamPangea	París	Ginebra	411	Airbus A380	5083.741200	47
…	…	…	…	…	…	…	…
Pam_NuBa_11380	PamPangea	Nueva York	Bali	16589	Airbus A380	215394.761088	1305
Pam_NuGi_11380	PamPangea	Nueva York	Ginebra	6206	Airbus A380	72458.773600	433
Pam_NuMe_11380	PamPangea	Nueva York	Melbourne	16082	Airbus A380	199144.563904	1272
Pam_BaGi_10747	PamPangea	Bali	Ginebra	12383	Boeing 747	138602.919000	845
Pam_NuMe_11380	PamPangea	Nueva York	Melbourne	16082	Airbus A380	195277.679168	1272

[82 rows x 7 columns]

Y de igual manera, puedo usar **where** para conservar toda la estructura

[10]:
```
df_aviones.where(condicion, "no usar").tail(20)# que nos muestre todas las␣
 ↪filas pero con los valores dela condicion y el resto Nan o lo que le␣
 ↪asignemos (no usar),  pero si queremos
#reducir tamaño le podemos filtar tb por columnas, ya que el filtrado de estos␣
 ↪vuelos habra poco y es aconsejable usar tail
```

[10]:

Id_vuelo	Aircompany	Origen	Destino	Distancia	avion	consumo_kg	duracion
Fly_NuRo_11320	no usar	no usar	no usar	no usar	no usar	no usar	no usar
Mol_PaMe_11380	no usar	no usar	no usar	no usar	no usar	no usar	no usar
Air_GiCa_10737	no usar	no usar	no usar	no usar	no usar	no usar	no usar
Fly_BaRo_10747	no usar	no usar	no usar	no usar	no usar	no usar	no usar
Mol_LoCi_10737	no usar	no usar	no usar	no usar	no usar	no usar	no usar
Air_GiCa_10747	no usar	no usar	no usar	no usar	no usar	no usar	no usar
Mol_BaLo_10737	no usar	no usar	no usar	no usar	no usar	no usar	no usar
Pam_BaMe_11320	no usar	no usar	no usar	no usar	no usar	no usar	no usar
Pam_GiMe_10737	no usar	no usar	no usar	no usar	no usar	no usar	no usar
Tab_GiCi_11320	no usar	no usar	no usar	no usar	no usar	no usar	no usar
Tab_GiLo_10737	no usar	no usar	no usar	no usar	no usar	no usar	no usar
Pam_BaNu_10747	no usar	no usar	no usar	no usar	no usar	no usar	no usar
Pam_NuMe_11380	PamPangea	Nueva York	Melbourne	16082	Airbus A380	195277.679168	1272
Air_GiCa_11320	no usar	no usar	no usar	no usar	no usar	no usar	no usar
Fly_GiBa_11380	no usar	no usar	no usar	no usar	no usar	no usar	no usar
Tab_LoLo_11320	no usar	no usar	no usar	no usar	no usar	no usar	no usar
Mol_CiLo_10737	no usar	no usar	no usar	no usar	no usar	no usar	no usar
Fly_RoCi_11320	no usar	no usar	no usar	no usar	no usar	no usar	no usar
Tab_RoLo_10747	no usar	no usar	no usar	no usar	no usar	no usar	no usar
Air_PaLo_10737	no usar	no usar	no usar	no usar	no usar	no usar	no usar

0.1.2 Filtrado de todo el dataframe

A veces puede ser conveniente aplicar una mascara o filtro a todo el `DataFrame`, en ese caso se aplica directamente sin el loc

[13]:
```
df_test= pd.DataFrame(np.random.randint(0,10,(4,3)), columns=["protones",␣
 ↪"neutrones", "quarks"])
df_test
```

[13]:

	protones	neutrones	quarks
0	2	3	6
1	5	3	7
2	4	5	5
3	0	5	5

[ ]:
```
#en mis experimento  no quiero usar nada que tenga mas de 6 elemetos
df_test[df_test < 6]
```

[ ]:

	protones	neutrones	quarks
0	NaN	5.0	4.0
1	NaN	2.0	1.0
2	NaN	NaN	NaN
3	3.0	NaN	NaN

Es similar a aplicar where, solo que este permite enmascar (al permitir cambiar valores):

[ ]:
```

```

[ ]:
```
df_test.where(df_test < 6, "no usar")
```

[ ]:

	protones	neutrones	quarks
0	no usar	5	4
1	no usar	2	1
2	no usar	no usar	no usar
3	3	no usar	no usar

Y podríamos ya no usar esos valores.

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

Y ahora algo más útil de forma directa, el consumo por kilometro:

[ ]:
```

```

Podríamos haber usado los atributos:

[ ]:

Pero quizá nos ha quedado un nombre de columna largo o creemos que algún nombre podría estar mejor expresado de otra forma... ¿Cómo cambiamos los nombres de las columnas?

[ ]:

O lo reasigno o bien utilizo el argumento `inplace` que es un argumento que existe en muchos métodos de los `DataFrame`.

[ ]:

[ ]:

0.1.3 Operaciones Sencillas

Veamos para terminar algunas operaciones sencillas de agregación que te sonarán porque se comparten casi en su totalidad con numpy. Para ello iremos contestando a una serie de preguntas [Que es otra forma de explorar los datos]

[ ]:
```
# Cual es la mayor distancia recorrida
```

[ ]:
```
# Cual es el menor consumo
```

[ ]:
```
# Cuanta distancia se han recorrido en los 1200 vuelos
```

[ ]:
```
# Cual es la media recorrida por estos viajes
```

[ ]:
```
# Y el consumo medio
```

Bueno, como medidas agregadas están bien, pero si quiero algo más de detalle y sin entrar en como quedarnos con solo las filas que cumplan una condición también podemos hacer lo siguiente

[ ]:
```
# Cual es el viaje con menor consumo
```

[ ]:
```
# Cual es el avion con el mayor consumo medio
```

[ ]:
```
# Pero si solo queremos ver el uno...
```

[Ya vamos viendo cierto potencial, pero vemos que nos falta algo que nos permita ser más precisos, o hacer preguntas más complicadas, nos faltan los filtros. Los veremos en la siguiente sesión, mientras...]

Por si quieres practicas, las siguientes agregaciones vienen con el paquete de Pandas:

Agregación	Descripción
`count()`	Número total de elementos
`first()`, `last()`	Primer y último elemento
`mean()`, `median()`	Media y mediana

Agregación	Descripción
`min()`, `max()`	Mínimo y máximo
`std()`, `var()`	Desviación estándar y varianza
`mad()`	Desviación media absoluta
`prod()`	Producto de todos los elementos
`sum()`	Suma de todos los elementos

Todos están presentes como objetos de `Dataframe` y `Series`. 1. **std()**: La desviación media o estandar, medida de dispersión que indica cuánto se alejan, en promedio, los valores individuales de un conjunto de datos respecto a la media aritmética de esos datos. Formula: **Desviacion Media= 1/n ^n i=1 (Xi - X(estax respresenta la media aritmetica))** 2. **var** Varianza, es otra medida de dispersión que describe qué tan dispersos o alejados están los valores individuales de un conjunto de datos respecto a su media aritmética, que mide la dispersion en terminos cuadraticos no en terminos absolutos como la anterior. Formula:**Varianza (^2) = 1/n** n i=1 (Xi - X(estax respresenta la media aritmetica))^2 3. La Desviación Media Absoluta o desviación absoluta (DMA) es una medida de dispersión que cuantifica la magnitud promedio de las desviaciones entre cada punto de datos individual y la media de un conjunto de datos, no elevando las diferencias al cuadrado, lo que la hace menos sensible a valores atípicos o extremos en los datos que la var o la std. Formula: **Desviacion Media= 1/n ^n i=1 | Xi - X(estax respresenta la media aritmetica)|

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

Una vez tenemos un DataFrame, tenemos varias formas de explorarlo y ver su contenido. Veamos su aspecto general

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:
```
df_aviones.head()
```

[ ]:
```
df_aviones.head()
df_aviones.head(20)
df_aviones.tail()
df_aviones.tail(20)
```

0.1.4 Descripción inicial

Lo primero en general es qué columnas tiene: [e intentar ver ya si puedo entender a qué se refiere cada una, pero eso lo veremos con más detalle en el siguiente sprint]

[ ]:
```
df_aviones.columns
```

Una descripción general matemática de los valores numéricos:

[ ]:
```
df_aviones.describe()
```

Si quiero ver los tipos de cada columna

[ ]:
```
df_aviones.dtypes
```

Los tipos en Pandas se heredan parcialmente de numpy, por eso tienes int64, float64, pero además ves que los tipos string (y aquellas columnas que tengan tipos mezclados) se denominan object y que luego al tratar cada valor ya interpretará su tipo.

Ahora una descripción más completa, dentro de su generalidad, con el método info:

[ ]:
```
df_aviones.info()
```

0.1.5 Rascando los valores de las columnas

Pero si ahora quiero entrar en más detalle, ¿cómo puedo hacer una primera observación de los valores de una columna?

[ ]:
```
df_aviones["avion"].unique()
```

Pero igual quiero saber cómo están distribuidos

[ ]:
```
df_aviones["avion"].value_counts()
```

Fijate en que estos métodos, unique y value_counts son realmente métodos de series (porque al escoger la columna primero estamos escogiendo una serie de pandas) y por tanto se pueden aplicar a cualquier serie

[ ]:
```
serie = pd.Series(np.random.randint(0,4,40))
serie
```

[ ]:
```
serie.unique()
```

[ ]:
```
serie.value_counts()
```

[Y hasta aquí la primera pildora, juega con el resto de columnas y explora a tu gusto el dataframe antes de pasar a la siguiente sesión en la que empezaremos de verdad a manipularlos]

04_Operaciones_Sobre_Filtrado

November 28, 2023

0.1 Operaciones sobre subconjuntos

En esta sesión vamos a aprovechar lo aprendido sobre filtrado en la anterior para trabajar solo con parte del dataframe, de forma que podamos hacer cálculos más precios, obtener respuestas más concretas y manipular datos de una manera selectiva.

Y para empezar, como en las últimas sesiones, recuperemos nuestra `DataFrame` de referencia:

[1]:
```
import numpy as  np
import pandas as pd

df_aviones = pd.read_csv("./data/dataset_inicial_aviones.csv", index_col =␣
 ↪"Id_vuelo")
```

[27]:
```
df_aviones
```

[27]:

Id_vuelo	Aircompany	Origen	Destino	Distancia
Air_PaGi_10737	Airnar	París	Ginebra	411
Fly_BaRo_10737	FlyQ	Bali	Roma	12738
Tab_GiLo_11380	TabarAir	Ginebra	San Francisco	9103
Mol_PaCi_10737	MoldaviAir	París	Cincinnati	6370
Tab_CiRo_10747	TabarAir	Cincinnati	Roma	7480
…	…	…	…	…
Tab_LoLo_11320	TabarAir	Los Angeles	Londres	8785
Mol_CiLo_10737	MoldaviAir	Cincinnati	Londres	6284
Fly_RoCi_11320	FlyQ	Roma	Cincinnati	7480

```
Tab_RoLo_10747    TabarAir         Roma        Londres       1433
Air_PaLo_10737      Airnar        París    Los Angeles       9099

                       avion     consumo_kg  duracion  supertrayectos
Id_vuelo
Air_PaGi_10737   Boeing 737    1028.691900        51           False
Fly_BaRo_10737   Boeing 737   33479.132544      1167            True
Tab_GiLo_11380  Airbus A380  109439.907200       626           False
Mol_PaCi_10737   Boeing 737   17027.010000       503           False
Tab_CiRo_10747   Boeing 747   86115.744000       518           False
…                      …           …       …             …
Tab_LoLo_11320  Airbus A320   24766.953120       756           False
Mol_CiLo_10737   Boeing 737   16491.729600       497           False
Fly_RoCi_11320  Airbus A320   19721.049920       662           False
Tab_RoLo_10747   Boeing 747   15734.053400       115           False
Air_PaLo_10737   Boeing 737   22331.675700       711           False

[1200 rows x 8 columns]
```

0.1.1 Operaciones y Consultas Selectivas

Para empezar vamos a realizar algunas operaciones de las que hicimos hace dos sesiones pero ahora ya podemos ser más selectivos

NOTA PERSONAL :El metodo **groupby** divide el DataFrame en grupos de filas y columnas y cada grupo contendrá filas con valores idénticos en las columnas de agrupación, obteniendo un objeto de grupo con referencias a cada uno de los grupos creados, Despues puedes hacer operaciones diversas en esos grupos como la media, la suma, el máximo, el mínimo o cualquier otra operación en función de tus necesidades.

[40]:
```
#METODO FACIL PARA MAS ADEALNTE#
""" mayor distancia por compañias
Distancia_company = df_aviones.groupby("Aircompany")["Distancia"].max()#␣
 ↪agrupame en el dataframe las compañias con las mayores distancias recorridas␣
 ↪en sus viajes
display(Distancia_company)"""
```

[40]:
```
' mayor distancia por compañias\nDistancia_company =
df_aviones.groupby("Aircompany")["Distancia"].max()# agrupame en el dataframe
las compañias con las mayores distancias recorridas en sus
viajes\ndisplay(Distancia_company)'
```

[2]:
```
### mayor distancia por compañias:


"""#creo un diccionario para guarda de la mayor distancia recorrida por compañía
dict_mayor_dist_c = {}
```

```
# Obtener las compañías únicas
company_unicas = df_aviones['Aircompany'].unique()
#print(company_unicas) # lista de las 5 compañias

# Iterar a través de las compañías únicas, filtrando el DF por compañia
for company in company_unicas:
    listado_company = df_aviones[df_aviones['Aircompany'] == company]
    #print(listado_company) # extrayendo un listado de 231 elementos con␣
 ↪infromacion del DF

    # calculo con un array la distancia máxima para cada compañía
    distancia_maxima = np.max(listado_company['Distancia'])
    #print(distancia_maxima)# 5 distancias maximas

    # Guardo la distancia máxima en el diccionario (diccionario - clave - valor)
    dict_mayor_dist_c [company] = distancia_maxima
    #print(company) # misma informacion que company_unicas pero al iterar esta␣
 ↪en una columna

# Imprimir la mayor distancia por compañía, iterando por el dict en clave y␣
 ↪valor
for company, distancia in dict_mayor_dist_c.items():
    print(f"La Compañía {company}, ha recorrido un total de {distancia}␣
 ↪kilometros, siendo su mayor distancia recorrida")"""
```

[2]:
```
'#creo un diccionario para guarda de la mayor distancia recorrida por
compañía\ndict_mayor_dist_c = {}\n\n# Obtener las compañías
únicas\ncompany_unicas =
df_aviones[\'Aircompany\'].unique()\n#print(company_unicas) # lista de las 5
compañias\n\n# Iterar a través de las compañías únicas, filtrando el DF por
compañia\nfor company in company_unicas:\n    listado_company =
df_aviones[df_aviones[\'Aircompany\'] == company]\n    #print(listado_company) #
extrayendo un listado de 231 elementos con infromacion del DF\n    \n    #
calculo con un array la distancia máxima para cada compañía \n
distancia_maxima = np.max(listado_company[\'Distancia\'])\n
#print(distancia_maxima)# 5 distancias maximas\n\n    # Guardo la distancia
máxima en el diccionario (diccionario - clave - valor)\n    dict_mayor_dist_c
[company] = distancia_maxima\n    #print(company) # misma informacion que
company_unicas pero al iterar esta en una columna\n\n# Imprimir la mayor
distancia por compañía, iterando por el dict en clave y valor\nfor company,
distancia in dict_mayor_dist_c.items():\n    print(f"La Compañía {company}, ha
recorrido un total de {distancia} kilometros, siendo su mayor distancia
recorrida")'
```

[3]:
```
### mayor distancia por compañias:## ESTA ES LA DEL PROFE
```

```python
# itero sobre el las compañias del DF en valores unicos no todos los valores de␣
 ↪las 1200 columnas
for company in df_aviones["Aircompany"].unique():
    distancia_max = df_aviones.loc[df_aviones["Aircompany"] == company,␣
 ↪"Distancia"]# hago un filtrado con una serie panda con la mascara␣
 ↪(DF[compañia ])de las compañias del DF como filtro == me quedo con las␣
 ↪compañias y distancia
    # cumplen la condicion
    print(f"La mayor distancia cubierta por la compañia {company} es de␣
 ↪{distancia_max} Km.")
```

```
La mayor distancia cubierta por la compañia Airnar es de Id_vuelo
Air_PaGi_10737      411
Air_GiCa_11380     1725
Air_GiLo_10747     9103
Air_BaGi_11380    12383
Air_BaCa_10737    12798
                  …
Air_PaCa_11320     1447
Air_GiCa_10737     1725
Air_GiCa_10747     1725
Air_GiCa_11320     1725
Air_PaLo_10737     9099
Name: Distancia, Length: 218, dtype: int64 Km.
La mayor distancia cubierta por la compañia FlyQ es de Id_vuelo
Fly_BaRo_10737    12738
Fly_RoNu_11320     6877
Fly_GiCi_10737     6969
Fly_GiRo_10737      698
Fly_BaGi_10737    12383
                  …
Fly_NuBa_11380     6170
Fly_NuRo_11320     6877
Fly_BaRo_10747    12738
Fly_GiBa_11380    12383
Fly_RoCi_11320     7480
Name: Distancia, Length: 216, dtype: int64 Km.
La mayor distancia cubierta por la compañia TabarAir es de Id_vuelo
Tab_GiLo_11380    9103
Tab_CiRo_10747    7480
Tab_LoCi_10737    3073
Tab_CiLo_10747    3073
Tab_GiLo_11380     739
                  …
Tab_NuGi_10747    6206
Tab_GiCi_11320    6969
```

```
Tab_GiLo_10737    9103
Tab_LoLo_11320    8785
Tab_RoLo_10747    1433
Name: Distancia, Length: 271, dtype: int64 Km.
La mayor distancia cubierta por la compañia MoldaviAir es de Id_vuelo
Mol_PaCi_10737     6370
Mol_CaMe_10737    20029
Mol_PaLo_11320      344
Mol_LoBa_11380    12553
Mol_MeCa_10737    20029
                  …
Mol_PaLo_11320      344
Mol_PaMe_11380    16925
Mol_LoCi_10737     6284
Mol_BaLo_10737    12553
Mol_CiLo_10737     6284
Name: Distancia, Length: 264, dtype: int64 Km.
La mayor distancia cubierta por la compañia PamPangea es de Id_vuelo
Pam_PaMe_11380    16925
Pam_NuBa_10737    16589
Pam_MePa_10737    16925
Pam_GiNu_11320     6206
Pam_MeBa_10737     2779
                  …
Pam_GiMe_10747    16674
Pam_BaMe_11320     2779
Pam_GiMe_10737    16674
Pam_BaNu_10747    16589
Pam_NuMe_11380    16082
Name: Distancia, Length: 231, dtype: int64 Km.
```

[17]:
```
#METODO FACIL PARA MAS ADEALNTE#
"""viaje_menor = df_aviones.groupby("avion")["consumo_kg"].min()# agrupame en␣
 ↪el dataframe las aviones con los menores consumo por viaje
print(viaje_menor)"""
```

```
avion
Airbus A320     863.4400
Airbus A380    3976.6400
Boeing 737      835.9200
Boeing 747     3740.3808
Name: consumo_kg, dtype: float64
```

[3]:
```
"""#### Cual es el viaje con menor consumo por tipo de avion

# creo un diccionario para los viajes con menor consumo por tipo de avión
dict_menor_consumo_avion = {}
```

```
# Encuentra los tipos únicos de avión
tipos = df_aviones["avion"].unique()
#print(tipos)# una lista con 4 tipos de aviones

# Itera a través de los tipos de avión, filtrando el DF para cada tipo de avion
for tipo_avion in tipos:
    avionacos = df_aviones[df_aviones["avion"] == tipo_avion]
    #print(avionacos) #extrayendo un listado de 315 elementos con informaion de␣
 ↪todas las columnas de DF

    # claculo el índice de la fila donde se encuntre el valor minimo del viaje␣
 ↪con el menor consumo en ese tipo de avión
    indice_ecologista = avionacos["consumo_kg"].idxmin()
    #print(indice_ecologista) # informacion con id de  4 vuelos que seran los␣
 ↪de menor consumo

    # mediante una serie panda obtengo la fila correspondiente al viaje con el␣
 ↪menor consumo
    viaje_ecologista = avionacos.loc[indice_ecologista]

    # guardo el resultado en el diccionario (dict - clave - valor)
    dict_menor_consumo_avion[tipo_avion] = viaje_ecologista

# Imprimo los viajes con menor consumo por tipo de avión, iterando por el dict␣
 ↪en clave y calor
for tipo_avion, viaje in dict_menor_consumo_avion.items():
    print(f"El Tipo de Avión {tipo_avion} ha realizado los siguientes viajes␣
 ↪con menor consumo:\n {viaje}\n")"""
```

[3]:
```
'#### Cual es el viaje con menor consumo por tipo de avion \n\n# creo un
diccionario para los viajes con menor consumo por tipo de
avión\ndict_menor_consumo_avion = {}\n\n# Encuentra los tipos únicos de
avión\ntipos = df_aviones["avion"].unique()\n#print(tipos)# una lista con 4
tipos de aviones\n\n# Itera a través de los tipos de avión, filtrando el DF para
cada tipo de avion\nfor tipo_avion in tipos:\n    avionacos =
df_aviones[df_aviones["avion"] == tipo_avion]\n    #print(avionacos) #extrayendo
un listado de 315 elementos con informaion de todas las columnas de DF\n
\n    # claculo el índice de la fila donde se encuntre el valor minimo del viaje
con el menor consumo en ese tipo de avión\n    indice_ecologista =
avionacos["consumo_kg"].idxmin()\n    #print(indice_ecologista) # informacion
con id de  4 vuelos que seran los de menor consumo\n    \n    # mediante una
serie panda obtengo la fila correspondiente al viaje con el menor consumo\n
viaje_ecologista = avionacos.loc[indice_ecologista]\n    \n    # guardo el
resultado en el diccionario (dict - clave - valor)\n
dict_menor_consumo_avion[tipo_avion] = viaje_ecologista\n\n# Imprimo los viajes
con menor consumo por tipo de avión, iterando por el dict en clave y calor\nfor
```

```
tipo_avion, viaje in dict_menor_consumo_avion.items():\n    print(f"El Tipo de
Avión {tipo_avion} ha realizado los siguientes viajes con menor consumo:\n
{viaje}\n")'
```

[30]:
```
#METODO FACIL PARA MAS ADEALNTE#
"""aviones_company = df_aviones.groupby("avion")["Aircompany"].unique()#␣
 ↪agrupame el el DF los tipos de avion que esa las compañias en una lista de␣
 ↪valores unicos
print("Las compañias que usan cada tipo de avion son\n",aviones_company)
print("\n")
numero_aviones_company = df_aviones.groupby("avion")["Aircompany"].nunique()#␣
 ↪agrupa el numero de compañias que usas los diferentes tipos de aviones a␣
 ↪partir de valores unicos en la columna "avion" de cada compañia
display("El numero de compañias que usan los distintos tipos de aviones␣
 ↪son\n",numero_aviones_company)"""
```

```
Las compañias que usan cada tipo de avion son
 avion
Airbus A320     [MoldaviAir, TabarAir, PamPangea, FlyQ, Airnar]
Airbus A380     [TabarAir, PamPangea, Airnar, MoldaviAir, FlyQ]
Boeing 737      [Airnar, FlyQ, MoldaviAir, PamPangea, TabarAir]
Boeing 747      [TabarAir, Airnar, MoldaviAir, PamPangea, FlyQ]
Name: Aircompany, dtype: object
```

```
'El numero de compañias que usan los distintos tipos de aviones son\n'
```

```
avion
Airbus A320    5
Airbus A380    5
Boeing 737     5
Boeing 747     5
Name: Aircompany, dtype: int64
```

[78]:
```
## Cual es el viaje con menor consumo por tipo de avion

viaje = df_aviones["consumo_kg"].idxmin()


for tipo_avion in df_aviones["avion"].unique():
    res = df_aviones.loc[df_aviones["avion"] ==  viaje]
    #print(avioneta)



#print(condicion)
print(f" EL viaje con el consumo minimo {viaje} por tipo de avion: {res}")
```

```
 EL viaje con el consumo minimo Mol_PaLo_10737 por tipo de avion: Empty
DataFrame
Columns: [Aircompany, Origen, Destino, Distancia, avion, consumo_kg, duracion,
supertrayectos]
Index: []
```

[ ]:

[5]:
```
"""### Cuántos aviones usan de cada tipo en cada compañía

# creo un diccionario para guardar los resultados
dict_aviones_company = {}

# Encuentra los tipos únicos de avión
tipos = df_aviones["avion"].unique()
#print(tipos)# una lista con 4 tipos de aviones

# itero por los tipops de aviones , filtrando en el dataframe toda la␣
 ↪infomacion
for tipo in tipos:
    avioncillos = df_aviones[df_aviones["avion"] == tipo]
    #print(avioncillos) # salen 198 elementos con informacion de todas las␣
 ↪columnas del DF

# Itera a través de las compañías únicas, Filtrando el DataFrame por compañía
for company in avioncillos["Aircompany"].unique():
    filas_tipo_av = avioncillos[avioncillos["Aircompany"] == company]

# Ccantidad de aviones de ecada tipo en cada compañía
num_aviones = len(filas_tipo_av)

#almacena el resultado en el diccionario
dict_aviones_company[(company, tipo)] = num_aviones

# Imprimo los diferentes tipos de avión que usan cada compañia, iterando por el␣
 ↪dict en clave y calor
for (company, tipo), numero in dict_aviones_company.items():
    print(f" La compañía {company} usa {numero} aviones , siendo sus tipos:␣
 ↪{tipo}")"""
```

[5]:
```
'### Cuántos aviones usan de cada tipo en cada compañía\n\n# creo un diccionario
para guardar los resultados\ndict_aviones_company = {}\n\n# Encuentra los tipos
únicos de avión\ntipos = df_aviones["avion"].unique()\n#print(tipos)# una lista
con 4 tipos de aviones\n\n# itero por los tipops de aviones , filtrando en el
dataframe toda la infomacion \nfor tipo in tipos:\n    avioncillos =
df_aviones[df_aviones["avion"] == tipo]\n    #print(avioncillos) # salen 198
```

```
elementos con informacion de todas las columnas del DF\n\n# Itera a través de
las compañías únicas, Filtrando el DataFrame por compañía\nfor company in
avioncillos["Aircompany"].unique():\n    filas_tipo_av =
avioncillos[avioncillos["Aircompany"] == company]\n    \n# Ccantidad de aviones
de ecada tipo en cada compañía \nnum_aviones = len(filas_tipo_av)\n\n#almacena
el resultado en el diccionario\ndict_aviones_company[(company, tipo)] =
num_aviones\n\n# Imprimo los diferentes tipos de avión que usan cada compañia,
iterando por el dict en clave y calor\nfor (company, tipo), numero in
dict_aviones_company.items():\n    print(f" La compañía {company} usa {numero}
aviones , siendo sus tipos: {tipo}")'
```

[34]:
```
#METODO FACIL PARA MAS ADEALNTE#
"""a_Ginebra=(df_aviones['Destino'] == "Ginebra").sum()# buscame en el DF los␣
 ↪vuelos con destinos, sumados previante ,a Ginebra
print(f"Los vuelos con destino a Ginebra son un total de {a_Ginebra}")"""
```

```
Los vuelos con destino a Ginebra son un total de 163
```

[8]:
```
######## Cuántos aviones usan de cada tipo en cada compañía ##### ESTA ES LA␣
 ↪DEL PROFE

# itero sobre el las compañias del DF en valores unicos no todos los valores de␣
 ↪todas las columnas
for company in df_aviones["Aircompany"].unique():
      # da el mismo valor pero la diferencia entre poner df_aviones.Aircompany␣
 ↪( atributo ) y df_aviones ["Aircompany"], es que si la columna Aircompany ,␣
 ↪fuera Air company(separado) con el atributo daria error
    print(f" la distribucion de aviones de uso por tipo de avion para la␣
 ↪compañia {company} es:")
    print(df_aviones.loc[df_aviones.Aircompany == company, "avion"].
 ↪value_counts())# filtramos por sere panda en el DF por␣
 ↪compañias(DF[compañia])  para que nos de las columnas compañia y avion, pero␣
 ↪como quiero
    # saber la distrubucion de uso( realemnte el nuemro de ves que usa cada␣
 ↪avion cada compañia) , uso al final el metodo .value_counts()
```

```
 la distribucion de aviones de uso por tipo de avion para la compañia Airnar es:
avion
Boeing 747     68
Airbus A380    64
Boeing 737     49
Airbus A320    37
Name: count, dtype: int64
 la distribucion de aviones de uso por tipo de avion para la compañia FlyQ es:
avion
Boeing 747     67
Airbus A380    61
```

```
Boeing 737     54
Airbus A320    34
Name: count, dtype: int64
 la distribucion de aviones de uso por tipo de avion para la compañia TabarAir
es:
avion
Boeing 747     77
Airbus A380    71
Boeing 737     62
Airbus A320    61
Name: count, dtype: int64
 la distribucion de aviones de uso por tipo de avion para la compañia MoldaviAir
es:
avion
Boeing 737     84
Airbus A380    74
Boeing 747     71
Airbus A320    35
Name: count, dtype: int64
 la distribucion de aviones de uso por tipo de avion para la compañia PamPangea
es:
avion
Airbus A380    73
Boeing 737     66
Boeing 747     61
Airbus A320    31
Name: count, dtype: int64
```

[10]:
```python
# Cuántos vuelos hay a Ginebra## ESTA ES LA DEL PROFE

condicion = df_aviones ["Destino"] == "Ginebra"
print(len(df_aviones.loc[condicion])) # contamos (usamos la funcion len()) los␣
 ↪aviones con destino a Ginebra(condicion con una serie panda, y no tengo que␣
 ↪poner nada mas, porque solo quiero el numero de filas que
#tiene destino a ginebra




# Cómo se distribuyen los origenes de los vuelos anteriores
# usaremio value_counts (distrobucion)
df_aviones.loc[condicion, "Origen"].value_counts()# con uan serie panda␣
 ↪comprobamos la misma condicion anterior pero ahora, le pedimos tb la columna␣
 ↪Origen, y con el metodo vaule counts tenemos la distribucion
# del numero de vuelos
```

```
163
```

[10]:
```
Origen
Bali           32
Nueva York     29
Londres        21
Cincinnati     17
Los Angeles    17
París          15
Melbourne      10
Barcelona       8
Roma            7
Cádiz           7
Name: count, dtype: int64
```

Muchas de estas consultas tienen un forma alternativa y, a veces, más eficiente de realizarse. Es el caso de usar agrupaciones (groupby) y que veremos al final de la unidad

0.1.2 Manipulación de Valores

Para terminar la sesión, veamos como usar los filtros para cambiar los valores del dataframe de una forma selectiva, lo que nos vendrá muy bien en el futuro para hacer "limpieza".

[19]:
```
es_TabarAir = df_aviones.Aircompany == "TabarAir" # condicion 1
a_los_Angeles = df_aviones .Destino =="Los Angeles" # condicion 2
df_aviones.loc[es_TabarAir & a_los_Angeles, "Destino"] = "San Francisco"# con␣
 ↪uan sere panda comprubo las filas del DF las que cumplen las 2 condciones y␣
 ↪despues de la coma,
#ponermos la columna destino ques es lo que quiero cambiar
# como comprobamos si lo ha cambiado pues preguntado la condicion, si dice que␣
 ↪no hay vuelos se ha cambiado:
df_aviones.loc[es_TabarAir & a_los_Angeles]
# ahora miramos en las 20 filas si se ha cambiado ya que no muetra ninguno al␣
 ↪imprimr la condicion
df_aviones.head(20)
# lo que no hemos cambiado su identificador, podriamos dicho cambiamos en la␣
 ↪misma condicionm cambiar el indice y despues el destino (lo veremos mas␣
 ↪adelante)
```

[19]:

Id_vuelo	Aircompany	Origen	Destino	Distancia
Air_PaGi_10737	Airnar	París	Ginebra	411
Fly_BaRo_10737	FlyQ	Bali	Roma	12738
Tab_GiLo_11380	TabarAir	Ginebra	San Francisco	9103
Mol_PaCi_10737	MoldaviAir	París	Cincinnati	6370
Tab_CiRo_10747	TabarAir	Cincinnati	Roma	7480
Mol_CaMe_10737	MoldaviAir	Cádiz	Melbourne	20029
Mol_PaLo_11320	MoldaviAir	París	Londres	344
Pam_PaMe_11380	PamPangea	París	Melbourne	16925
Pam_NuBa_10737	PamPangea	Nueva York	Bali	16589

```
Air_GiCa_11380      Airnar      Ginebra          Cádiz       1725
Tab_LoCi_10737    TabarAir  Los Angeles     Cincinnati       3073
Mol_LoBa_11380  MoldaviAir      Londres           Bali      12553
Tab_CiLo_10747    TabarAir   Cincinnati  San Francisco       3073
Tab_GiLo_11380    TabarAir      Ginebra        Londres        739
Tab_CiRo_11320    TabarAir   Cincinnati           Roma       7480
Tab_RoLo_10747    TabarAir         Roma  San Francisco      10077
Mol_MeCa_10737  MoldaviAir    Melbourne          Cádiz      20029
Air_GiLo_10747      Airnar      Ginebra    Los Angeles       9103
Pam_MePa_10737   PamPangea    Melbourne          París      16925
Pam_GiNu_11320   PamPangea      Ginebra     Nueva York       6206

                     avion     consumo_kg  duracion
Id_vuelo
Air_PaGi_10737   Boeing 737    1028.691900        51
Fly_BaRo_10737   Boeing 737   33479.132544      1167
Tab_GiLo_11380  Airbus A380  109439.907200       626
Mol_PaCi_10737   Boeing 737   17027.010000       503
Tab_CiRo_10747   Boeing 747   86115.744000       518
Mol_CaMe_10737   Boeing 737   53148.153240      1721
Mol_PaLo_11320  Airbus A320     915.246400        44
Pam_PaMe_11380  Airbus A380  217722.658400      1328
Pam_NuBa_10737   Boeing 737   45277.618464      1459
Air_GiCa_11380  Airbus A380   20339.820000       135
Tab_LoCi_10737   Boeing 737    7915.433400       253
Mol_LoBa_11380  Airbus A380  156721.694400       856
Tab_CiLo_10747   Boeing 747   32758.180000       224
Tab_GiLo_11380  Airbus A380    8542.840000        69
Tab_CiRo_11320  Airbus A320   21087.855360       662
Tab_RoLo_10747   Boeing 747  109569.236400       691
Mol_MeCa_10737   Boeing 737   51629.634576      1721
Air_GiLo_10747   Boeing 747  104801.018400       626
Pam_MePa_10737   Boeing 737   46622.417400      1485
Pam_GiNu_11320  Airbus A320   16200.142400       569
```

Supongamos que nos avisan de que los datos de TabarAir son erróneos porque para los vuelos a Los Angeles en realidad son a San Francisco. ¿Cómo podríamos cambiar ese valor?

[ ]:

Y ahora nos dicen que creemos una columna nueva "Supertrayectos" que debe ser True cuando el trayecto es mayor de 12000Km y False en caso contrario

[20]:
```
# Creandolo con un filtro/Selección
# cuanno nos pidan cambiar una columna y despues cambair los valores, nos␣
 ↪crearemos la columna con un valor por defecto , y asi no te de problemas de␣
 ↪referencia  e igularlos a False
```

```
df_aviones["supertrayectos"] = False # condicion que decimos que todos los␣
 ↪valores valen False y despues:
df_aviones.loc[df_aviones.Distancia > 12000, "supertrayectos"] = True # despues␣
 ↪con una serie panda extraemos del DF las filas con distancia >120000, y␣
 ↪despues la columna que quiero cambiar  y el valor lo cambiamos a true
# asi todos los valores seran fdlse menos los que sea mayores a 12000
df_aviones
```

[20]:

Id_vuelo	Aircompany	Origen	Destino	Distancia	avion	consumo_kg	duracion	supertrayectos
Air_PaGi_10737	Airnar	París	Ginebra	411	Boeing 737	1028.691900	51	False
Fly_BaRo_10737	FlyQ	Bali	Roma	12738	Boeing 737	33479.132544	1167	True
Tab_GiLo_11380	TabarAir	Ginebra	San Francisco	9103	Airbus A380	109439.907200	626	False
Mol_PaCi_10737	MoldaviAir	París	Cincinnati	6370	Boeing 737	17027.010000	503	False
Tab_CiRo_10747	TabarAir	Cincinnati	Roma	7480	Boeing 747	86115.744000	518	False
…	…	…	…	…	…	…	…	…
Tab_LoLo_11320	TabarAir	Los Angeles	Londres	8785	Airbus A320	24766.953120	756	False
Mol_CiLo_10737	MoldaviAir	Cincinnati	Londres	6284	Boeing 737	16491.729600	497	False
Fly_RoCi_11320	FlyQ	Roma	Cincinnati	7480	Airbus A320	19721.049920	662	False
Tab_RoLo_10747	TabarAir	Roma	Londres	1433	Boeing 747	15734.053400	115	False
Air_PaLo_10737	Airnar	París	Los Angeles	9099	Boeing 737	22331.675700	711	False

[1200 rows x 8 columns]

[25]:
```
# pero despue spodemos hacer
df_aviones.loc[df_aviones.supertrayectos, "Destino"].value_counts()#␣
 ↪comprobamos en el DF con serie panda, la fila nueva creada, y recupero␣
 ↪"Destino" , pq quier saber la distrbucion de destinos
# me da el numero de destinos que han recorrido mas de 12000 km
```

[25]:
```
Destino
Bali           114
Melbourne       68
```

```
Londres         47
Ginebra         42
Cincinnati      36
Cádiz           22
París           19
Nueva York      18
Roma            16
Barcelona        7
Los Angeles      3
Name: count, dtype: int64
```

[26]:
```
# Versión alternativa(la de arriba mejor)

df_aviones["supertrayectos"] = df_aviones["Distancia"] >12000 # directamente␣
 ↪creo la columna de supertrayectos a partir de la condicion (> 120000km)
df_aviones
```

[26]:
```
                Aircompany       Origen        Destino  Distancia  \
Id_vuelo
Air_PaGi_10737      Airnar        París        Ginebra        411
Fly_BaRo_10737        FlyQ         Bali           Roma      12738
Tab_GiLo_11380    TabarAir      Ginebra  San Francisco       9103
Mol_PaCi_10737  MoldaviAir        París     Cincinnati       6370
Tab_CiRo_10747    TabarAir   Cincinnati           Roma       7480
…                     …          …              …        …
Tab_LoLo_11320    TabarAir  Los Angeles        Londres       8785
Mol_CiLo_10737  MoldaviAir   Cincinnati        Londres       6284
Fly_RoCi_11320        FlyQ         Roma     Cincinnati       7480
Tab_RoLo_10747    TabarAir         Roma        Londres       1433
Air_PaLo_10737      Airnar        París    Los Angeles       9099

                      avion     consumo_kg  duracion  supertrayectos
Id_vuelo
Air_PaGi_10737   Boeing 737    1028.691900        51           False
Fly_BaRo_10737   Boeing 737   33479.132544      1167            True
Tab_GiLo_11380  Airbus A380  109439.907200       626           False
Mol_PaCi_10737   Boeing 737   17027.010000       503           False
Tab_CiRo_10747   Boeing 747   86115.744000       518           False
…                      …           …       …              …
Tab_LoLo_11320  Airbus A320   24766.953120       756           False
Mol_CiLo_10737   Boeing 737   16491.729600       497           False
Fly_RoCi_11320  Airbus A320   19721.049920       662           False
Tab_RoLo_10747   Boeing 747   15734.053400       115           False
Air_PaLo_10737   Boeing 737   22331.675700       711           False

[1200 rows x 8 columns]
```

En sesiones posteriores veremos la forma de manipular valores utilizando funciones definidas por el

usuario, con lo que terminarás de ver ya casi todo el potencial de manipulación de datos a partir de un `DataFrame`.

05_Duplicados

November 28, 2023

0.1 Duplicados

ANtes de empezar con manipulacion de datos con funciones definidas por el usario, vamos atrabajar con los duplicados. Los duplicados en los datos (generalmente de toda una fila), no son necesariamente malos, puede que los datos necesiten precisamente tener esos duplicados. Pero en general siempre va a ser bueno saber si existen, y en el caso de no quererlos saber como eliminarlos.

Como ultimante, nos cargamos un `DataFrame` de referencia, que esta vez es algo especial porque sí contiene duplicados, aunque lo usaremos poquito:

[1]:
```
import numpy as  np
import pandas as pd

df_aviones = pd.read_csv("./data/dataset_dup_aviones.csv", index_col =␣
 ↪"Id_vuelo")
```

0.1.1 Detección y eliminación de filas duplicadas

La forma de detectar filas duplicadas es emplear el método duplicated. El método se puede aplicar a todo el `DataFrame` o a una selección en la que nos dirá si hay duplicados en esa selección y tb se puede aplicar a series.

La sintaxis para comprobar por filas es:8 comprbar si hay una fila completemante duplicada

```
df_aviones.duplicated() # con un parámetro importante keep, que por defecto está a "first" ( K
```

Este método devuelve una serie de booleanos donde True dice que la fila se considera duplicada y False que no se considera duplicada.

El criterio de Pandas para etiquetar una fila como duplicado depende del mencionado argumento `keep`: 1. Etiqueta todas las filas que están duplicadas como duplicadas. (keep = False) 2. Etiqueta como duplicadas todas las apariciones menos la primera. (keep = "first") 3. Etiqueta como duplicadas todas las apariciones menos la última (keep = "last") .

¿Y para qué tanto lío? Porque cuando queramos eliminar los duplicados, se eliminarán solo los etiquetados como tal y así de esta forma me puedo quedar con las primeras apariciones, o las últimas. Eligiremos first o last depende del orden que queramos en nuestro set de Datos

Por ejemplo, supongamos un dataframe como:

[4]:
```
df_dup = pd.DataFrame( {"col1": ["a","c","a","a","h","c"],# al usar un␣
 ↪diccionnario usa las claves para nombrar las columnas
                         "col2":["b","d","b","b","j","d"],
                         "col3":["c","a","c","c","k","a"]},
                         index =␣
 ↪["fila1","fila2","fila3","fila4","fila5","fila6"])# pa nombra las filas
df_dup
```

[4]:
```
       col1 col2 col3
fila1    a    b    c
fila2    c    d    a
fila3    a    b    c
fila4    a    b    c
fila5    h    j    k
fila6    c    d    a
```

Donde las filas 1,3 y 4 están duplicadas por un lado, y las filas 2 y 6 por otro. Veamos el efecto de duplicated:

[5]:
```
df_dup.duplicated() # es igual a df_dup.duplicated(keep = "first")
# aqui nos dice que los duplicados son la 3,4,y la 6 , dejando la 1,2 y 6 como␣
 ↪buenas
```

[5]:
```
fila1    False
fila2    False
fila3     True
fila4     True
fila5    False
fila6     True
dtype: bool
```

Y si aplicamos el método como condición:(como serie loc)

[7]:
```
df_dup.loc[df_dup.duplicated()]# aqui nos marca la 3,4 6 que son repetidas y no␣
 ↪muestra mas
```

[7]:
```
       col1 col2 col3
fila3    a    b    c
fila4    a    b    c
```

```
fila6    c    d    a
```

Con el **Keep** lo que queremos decir es quedate es decir le decimos con que valores nos vamos a quedar no los que eliminara

Ahora con keep = "last"

[9]:
```
df_dup.duplicated(keep = "last")# ahora ha puesto las primeras como repetidas y␣
↪la 4,5, 6 como buenas
```

[9]:
```
fila1     True
fila2     True
fila3     True
fila4    False
fila5    False
fila6    False
dtype: bool
```

Y de nuevo veamos que nos selecciona como duplicados:(como metodo con loc como condicion)

[12]:
```
df_dup.loc[df_dup.duplicated(keep = "last")]# considera como duplicados la␣
↪1,2,3 y nos quedamos con los ultimos(last) como buenos
```

[12]:
```
      col1 col2 col3
fila1    a    b    c
fila2    c    d    a
fila3    a    b    c
```

Y finalmente con keep = False:

[15]:
```
df_dup.duplicated(keep = False)# considera las duplicadas a true y a false la␣
↪buena
```

[15]:
```
fila1     True
fila2     True
fila3     True
fila4     True
fila5    False
fila6     True
dtype: bool
```

Y el `DataFrame` seleccionado según duplicado:(u si lo vemos como condicion en loc)

[16]:
```
df_dup.loc[df_dup.duplicated(keep = False)]# Todo menos la fila 5 nos da como␣
↪duplicado
```

[16]:
```
      col1 col2 col3
fila1    a    b    c
fila2    c    d    a
fila3    a    b    c
```

```
fila4    a    b    c
fila6    c    d    a
```

Y todo esto es importante, porque el método para eliminar duplicados sólo lo hara de aquellas que marquemos como duplicados y también tiene su parámetro `keep` que implica la misma filosofía

Veámoslo con df_dup y luego apliquemos a nuestro `DataFrame` de aviones

(Primero nos hacemos una copia de backup)

[17]:
```
df_dup_reserva = df_dup.copy()
```

[18]:
```
df_dup.drop_duplicates() #keep = "first"# nos quedamos con las primera␣
↪aparaciones
```

[18]:
```
      col1 col2 col3
fila1    a    b    c
fila2    c    d    a
fila5    h    j    k
```

[19]:
```
df_dup.drop_duplicates(keep = "last") # se queda con las ultimas y borra las␣
↪primeras
```

[19]:
```
      col1 col2 col3
fila4    a    b    c
fila5    h    j    k
fila6    c    d    a
```

[20]:
```
df_dup.drop_duplicates(keep= False)# borrra todos los que estan duplicados y se␣
↪queda con el unico que no esta duplicado
```

[20]:
```
      col1 col2 col3
fila5    h    j    k
```

[21]:
```
df_dup# drop no modifica el df original para ello tendremos que usar el inplace␣
↪= True
```

[21]:
```
      col1 col2 col3
fila1    a    b    c
fila2    c    d    a
fila3    a    b    c
fila4    a    b    c
fila5    h    j    k
fila6    c    d    a
```

[ ]:
```

```

Por cierto, si queremos que el método modifique el `DataFrame` que lo llama debemos usar el argumento `inplace` con valor a True

[25]:
```
df_dup.drop_duplicates(keep = False, inplace = True)# borra todos los␣
↪dupplicados y borralos del original  tb
```

[26]:
```
df_dup# nos ha dejado solo la unica fiola no dupliacda
```

[26]:
```
      col1 col2 col3
fila5    h    j    k
```

Si lo aplicamos a nuestro dataframe

[28]:
```
# Veamos algunos duplicados
df_aviones.loc[df_aviones.duplicated(keep = False)] # le marcamos False para␣
↪que nos diga todos los duplicados
# como son mucghas filas vamos a filtrar un indice
df_aviones.loc["Mol_PaBa_10747"]
```

[28]:
```
                Aircompany Origen Destino  Distancia       avion  consumo_kg  \
Id_vuelo
Mol_PaBa_10747  MoldaviAir  París    Bali      11980  Boeing 747  130260.936
Mol_PaBa_10747  MoldaviAir  París    Bali      11980  Boeing 747  134092.140
Mol_PaBa_10747  MoldaviAir  París    Bali      11980  Boeing 747  140477.480
Mol_PaBa_10747  MoldaviAir  París    Bali      11980  Boeing 747  135369.208
Mol_PaBa_10747  MoldaviAir  París    Bali      11980  Boeing 747  130260.936
Mol_PaBa_10747  MoldaviAir  París    Bali      11980  Boeing 747  134092.140

                duracion
Id_vuelo
Mol_PaBa_10747       818
Mol_PaBa_10747       818
Mol_PaBa_10747       818
Mol_PaBa_10747       818
Mol_PaBa_10747       818
Mol_PaBa_10747       818
```

[30]:
```
# Podemos ver los duplicados, filtrando por otros campos, veamos los de FlyQ␣
↪usando para ellos las series loc para establecer una condicion
df_aviones.loc[(df_aviones.Aircompany =="FlyQ") & (df_aviones.duplicated(keep=␣
↪False))] # aqui keep quere decir "no te queds con ningun duplicado"
df_aviones.loc["Fly_BaRo_10747"] #filtramos por indice
```

[30]:
```
                Aircompany     Origen Destino  Distancia       avion  \
Id_vuelo
Fly_BaRo_10747       FlyQ       Bali    Roma      12738  Boeing 747
Fly_BaRo_10747       FlyQ       Bali    Roma      12738  Boeing 747
Fly_BaRo_10747       FlyQ       Bali    Roma      12738  Boeing 747
Fly_BaRo_10747       FlyQ  Barcelona    Roma        859  Boeing 747
Fly_BaRo_10747       FlyQ  Barcelona    Roma        859  Boeing 747
Fly_BaRo_10747       FlyQ       Bali    Roma      12738  Boeing 747
```

```
Fly_BaRo_10747       FlyQ  Barcelona    Roma        859  Boeing 747
Fly_BaRo_10747       FlyQ       Bali    Roma      12738  Boeing 747
Fly_BaRo_10747       FlyQ  Barcelona    Roma        859  Boeing 747
Fly_BaRo_10747       FlyQ  Barcelona    Roma        859  Boeing 747
```

Id_vuelo	consumo_kg	duracion
Fly_BaRo_10747	141218.563200	1049
Fly_BaRo_10747	141218.563200	1049
Fly_BaRo_10747	155340.419520	1049
Fly_BaRo_10747	9431.648200	77
Fly_BaRo_10747	9248.509400	77
Fly_BaRo_10747	144042.934464	1049
Fly_BaRo_10747	9706.356400	77
Fly_BaRo_10747	149691.676992	1049
Fly_BaRo_10747	9248.509400	77
Fly_BaRo_10747	9797.925800	77

[31]:
```
# Finalmente veamos el agregado de duplicados en función del parámetro keep
for val_keep in [False, "first", "last"]:# itera con la variable val_keep para␣
 ↪ver [ lo contenido en la lista]
    num_vuelos= len(df_aviones.loc[df_aviones.duplicated(keep = val_keep)])#␣
 ↪dime el numero de vuelos que hay duplicados en el DF con valores que␣
 ↪iteramos en el For
    print(f"Para Keep ={val_keep}")# aqui imprimira los valores asignados en el␣
 ↪for a keep
    print(f"Nuemro de registros:{num_vuelos}")# imprimira el numero de vuelos ␣
 ↪dduplicados con las 3 condiciones citadas en for
```

```
Para Keep =False
Nuemro de registros:411
Para Keep =first
Nuemro de registros:215
Para Keep =last
Nuemro de registros:215
```

Nos dice que tenemos esos duplicados considerandolos todos. Ahora nosotros tenemos que decidir qué hacemos... [pero eso para la sesión en vivo]

0.1.2 Detección de columnas

Para terminar veamos como se pueden detectar duplicados en una o varias columnas, siempre teniendo en cuenta que este caso es aún más frecuente que el anterior y, en general, salvo para columnas que deban tener valores únicos (como por ejemplo el DNI de una persona), no será muy útil comprobarlo.

[32]:
```
df_dup = df_dup_reserva.copy() # recuperamos el anterior
df_dup
```

[32]:
```
      col1 col2 col3
fila1    a    b    c
fila2    c    d    a
fila3    a    b    c
fila4    a    b    c
fila5    h    j    k
fila6    c    d    a
```

[33]:
```
df_dup["col2"].duplicated()# exoime los primeros
```

[33]:
```
fila1    False
fila2    False
fila3     True
fila4     True
fila5    False
fila6     True
Name: col2, dtype: bool
```

[34]:
```
df_dup ["col2"].duplicated(keep = False)# no se queda bno ninguno duplicado
```

[34]:
```
fila1     True
fila2     True
fila3     True
fila4     True
fila5    False
fila6     True
Name: col2, dtype: bool
```

[35]:
```
# tb podria ver el DF filtrado por esa condicion con una serie panda
df_dup.loc[df_dup["col2"].duplicated(keep = False)] # no te quedes sin nunguna␣
 ↪dupl( las que muestra son las dupliocadas si comparamos por loc)
# toda funcion , todo metodo, toda comprobacion que devuelva una serie alineada␣
 ↪con un DF, es decir con el nombre de su filas y un booleano , me va a servir␣
 ↪para filtra ese DF
```

[35]:
```
      col1 col2 col3
fila1    a    b    c
fila2    c    d    a
fila3    a    b    c
fila4    a    b    c
fila6    c    d    a
```

Y si queremos ver varias columnas

[36]:
```
# recuerda para coger varias columnas , el fantasy indexado, abre dobles␣
 ↪corchetes y pon la liista con las col que quierrtas y en el orden que quieras
```

```
df_dup[["col2","col1"]].duplicated(keep ="last") #esto nos hace un␣
 ↪subdataframe, y nos dira las filas que tiene las col.1 y 2 repetidas con el␣
 ↪keep = "last"
#las true son las repetiodas por el last
```

[36]:
```
fila1     True
fila2     True
fila3     True
fila4    False
fila5    False
fila6    False
dtype: bool
```

Si ahora quisieramos eliminar esas filas, usariamos drop_duplicates también pero ojo nos cargaríamos toda la fila y a lo mejor no es lo que nos interesa.

[38]:
```
##vamos a borrar esas filas por esa condicion con una serie panda (condciion =␣
 ↪[df_dup[["col1", "col2"]].duplicated(keep= "last")])
df_dup.loc[df_dup[["col1", "col2"]].duplicated(keep= "last")].
 ↪drop_duplicates(keep= "last")
```

[38]:

	col1	col2	col3
fila2	c	d	a
fila3	a	b	c

[ ]:
```

```

06_Nulos

November 28, 2023

0.1 Nulos

A veces, en general muchas veces por no decir casi siempre, encontraremos en nuestros datos que no todos las columnas tienen valores (no ya que no sean correctos) sino que no tienen valores. Es decir, faltan datos.

Esta ausencia puede venir dada por un "", un valor vacío, o por un código especial. Además es posible que cuando lo pasemos a Pandas nos genere en vez de ese "", vacío o código especial el ya antes mencionado `NaN`. Los `NaN` son una forma particular de expresar un valor vacío o un valor faltante o un no-valor. Es decir que algo hay que hacer con ello. A los valores `NaN` (en otros lenguajes `null` y en otros `nil`) o nulos hay que tratarlos, al igual que a los datos faltantes.

En este sprint nos vamos a centrar en qué se puede hacer con los `NaN` cuando se detectan en Pandas, y parte de lo que veamos se podrá aplicar a datos faltantes en general. Pero, el tratamiento completo de datos faltantes (también *missing data* o *missings*, que es como lo vas a encontrar en la literatura) lo contemplaremos en los sprints siguientes.

[5]:
```
import numpy as  np
import pandas as pd

df_aviones = pd.read_csv("./data/dataset_missing_aviones.csv", index_col =␣
 ↪"Id_vuelo")
```

0.1.1 Detectando nulos y datos faltantes

Uno de los problemas de "echar un vistazo" (métodos head y tail) a un `DataFrame` es que así no es fácil detectar si hay datos faltantes o nulos o NaN. Por ejemplo

[6]:
```
df_aviones
```

[6]:
```
                Aircompany       Origen      Destino  Distancia        avion  \
Id_vuelo
Air_PaGi_10737      Airnar        París      Ginebra        411   Boeing 737
Fly_BaRo_10737        FlyQ         Bali         Roma      12738   Boeing 737
Tab_GiLo_11380    TabarAir      Ginebra  Los Angeles       9103  Airbus A380
Mol_PaCi_10737  MoldaviAir        París   Cincinnati       6370   Boeing 737
Tab_CiRo_10747    TabarAir   Cincinnati         Roma       7480   Boeing 747
…                    …          …           …        …           …
Tab_LoLo_11320    TabarAir  Los Angeles      Londres       8785  Airbus A320
Mol_CiLo_10737  MoldaviAir   Cincinnati      Londres       6284   Boeing 737
Fly_RoCi_11320        FlyQ         Roma   Cincinnati       7480  Airbus A320
Tab_RoLo_10747    TabarAir         Roma      Londres       1433   Boeing 747
Air_PaLo_10737      Airnar        París  Los Angeles       9099   Boeing 737

                   consumo_kg  duracion
Id_vuelo
Air_PaGi_10737    1028.691900      51.0
Fly_BaRo_10737   33479.132544    1167.0
Tab_GiLo_11380  109439.907200     626.0
Mol_PaCi_10737   17027.010000     503.0
Tab_CiRo_10747   86115.744000     518.0
…                         …       …
Tab_LoLo_11320   24766.953120     756.0
Mol_CiLo_10737   16491.729600     497.0
Fly_RoCi_11320   19721.049920     662.0
Tab_RoLo_10747   15734.053400     115.0
Air_PaLo_10737   22331.675700     711.0

[1200 rows x 7 columns]
```

Hay que acudir a info(), para empezar:

[7]:
```
df_aviones.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1200 entries, Air_PaGi_10737 to Air_PaLo_10737
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Aircompany  1200 non-null   object
 1   Origen      1200 non-null   object
 2   Destino     1023 non-null   object
 3   Distancia   1200 non-null   int64
 4   avion       1200 non-null   object
 5   consumo_kg  1200 non-null   float64
 6   duracion    1016 non-null   float64
```

```
dtypes: float64(2), int64(1), object(4)
memory usage: 75.0+ KB
```

Fijate en `Destino` y en `duracion`, nos dice que hay 1023 y 1016 valores non-null pero mira el resto de columnas (hay 1200 valores). Eso quiere decir...

Para verlo con más precisión, primero aplicaremos el método value_counts con un argumento nuevo: `dropna`

[9]:
```
df_aviones["Destino"].value_counts(dropna = False)
```

[9]:
```
Destino
NaN            177
Ginebra        137
Bali           137
Cincinnati     125
Londres        106
París          104
Nueva York      95
Roma            92
Melbourne       75
Cádiz           65
Los Angeles     57
Barcelona       30
Name: count, dtype: int64
```

[13]:
```
df_aviones["duracion"].value_counts(dropna = False)
```

[13]:
```
duracion
NaN       184
818.0      32
845.0      32
1326.0     27
433.0      23
          ...
687.0       2
488.0       2
1175.0      1
731.0       1
129.0       1
Name: count, Length: 117, dtype: int64
```

Confirmada su existencia doblemente. La manera de identificar las filas con valores faltantes es a través del metodo isna() aplicado a las columnas en las que sabemos que hay valores nulos (Siempre que queremos mostrar una condicion .loc serie panda)

[16]:
```
df_aviones.loc[df_aviones["Destino"].isna()]
# dame todas las filas de destinos con valores nulos(condicion)
```

[16]:
```
                Aircompany      Origen Destino  Distancia        avion  \
Id_vuelo
Pam_MePa_10737   PamPangea   Melbourne     NaN      16925   Boeing 737
Pam_GiNu_11320   PamPangea     Ginebra     NaN       6206  Airbus A320
Mol_CaBa_10747  MoldaviAir       Cádiz     NaN      12798   Boeing 747
Fly_RoNu_11320        FlyQ        Roma     NaN       6877  Airbus A320
Fly_GiCi_10737        FlyQ     Ginebra     NaN       6969   Boeing 737
…                      …          …      …          …           …
Mol_BaLo_10737  MoldaviAir        Bali     NaN      12553   Boeing 737
Tab_CiRo_11380    TabarAir  Cincinnati     NaN       7480  Airbus A380
Air_PaCa_11320      Airnar       París     NaN       1447  Airbus A320
Tab_GiLo_11380    TabarAir     Ginebra     NaN        739  Airbus A380
Pam_BaNu_10747   PamPangea        Bali     NaN      16589   Boeing 747

                    consumo_kg  duracion
Id_vuelo
Pam_MePa_10737   46622.417400    1485.0
Pam_GiNu_11320   16200.142400     569.0
Mol_CaBa_10747  156072.121920    1053.0
Fly_RoNu_11320   18131.238008     618.0
Fly_GiCi_10737   18289.443600     549.0
…                         …       …
Mol_BaLo_10737   34579.096344    1153.0
Tab_CiRo_11380   86468.800000     518.0
Air_PaCa_11320    3995.167000     124.0
Tab_GiLo_11380    9311.695600      69.0
Pam_BaNu_10747  185751.412496    1305.0

[177 rows x 7 columns]
```

Si queremos ver las dos juntas:

[19]:
```
es_destino_NaN = df_aviones["Destino"].isna()
es_duracion_NaN = df_aviones["duracion"].isna()
#vemos las condicones, y no le ponemos parentesis pq le he asignado variables
df_aviones.loc[es_destino_NaN & es_duracion_NaN]
# veremos intrsecciones de filas y columnas o solo en columnas o solo en filas
```

[19]:
```
                Aircompany      Origen Destino  Distancia        avion  \
Id_vuelo
Pam_NuPa_11380   PamPangea   Nueva York     NaN       5835  Airbus A380
Air_GiLo_11320      Airnar      Ginebra     NaN       9103  Airbus A320
Air_LoBa_10747      Airnar  Los Angeles     NaN      12845   Boeing 747
Air_CiPa_11380      Airnar   Cincinnati     NaN       6370  Airbus A380
Tab_LoGi_11380    TabarAir  Los Angeles     NaN       9103  Airbus A380
Tab_LoCi_10747    TabarAir  Los Angeles     NaN       3073   Boeing 747
Air_LoCi_11320      Airnar  Los Angeles     NaN       3073  Airbus A320
Tab_NuLo_11320    TabarAir   Nueva York     NaN       5566  Airbus A320
```

Pam_NuPa_11320	PamPangea	Nueva York	NaN	5835	Airbus A320
Tab_LoGi_11380	TabarAir	Los Angeles	NaN	9103	Airbus A380
Air_GiPa_11320	Airnar	Ginebra	NaN	411	Airbus A320
Pam_MeLo_11380	PamPangea	Melbourne	NaN	16900	Airbus A380
Pam_PaNu_10737	PamPangea	París	NaN	5835	Boeing 737
Air_CiGi_10747	Airnar	Cincinnati	NaN	6969	Boeing 747
Mol_LoCi_11320	MoldaviAir	Londres	NaN	6284	Airbus A320
Mol_LoBa_10737	MoldaviAir	Londres	NaN	12553	Boeing 737
Pam_LoNu_10747	PamPangea	Londres	NaN	5566	Boeing 747
Air_LoPa_10737	Airnar	Los Angeles	NaN	9099	Boeing 737
Air_CiPa_11320	Airnar	Cincinnati	NaN	6370	Airbus A320
Mol_PaBa_11380	MoldaviAir	París	NaN	11980	Airbus A380
Tab_GiLo_11380	TabarAir	Ginebra	NaN	739	Airbus A380
Air_GiBa_11380	Airnar	Ginebra	NaN	12383	Airbus A380
Mol_MeCa_10737	MoldaviAir	Melbourne	NaN	20029	Boeing 737
Air_CaPa_11320	Airnar	Cádiz	NaN	1447	Airbus A320
Mol_CaMe_10737	MoldaviAir	Cádiz	NaN	20029	Boeing 737
Pam_MeNu_11380	PamPangea	Melbourne	NaN	16082	Airbus A380
Mol_CiMe_11380	MoldaviAir	Cincinnati	NaN	15262	Airbus A380
Mol_CiBa_10737	MoldaviAir	Cincinnati	NaN	15011	Boeing 737
Air_BaCa_10747	Airnar	Bali	NaN	12798	Boeing 747
Mol_BaCi_10747	MoldaviAir	Bali	NaN	15011	Boeing 747

Id_vuelo	consumo_kg	duracion
Pam_NuPa_11380	72174.282000	NaN
Air_GiLo_11320	24475.345336	NaN
Air_LoBa_10747	148101.000320	NaN
Air_CiPa_11380	81000.920000	NaN
Tab_LoGi_11380	112596.827600	NaN
Tab_LoCi_10747	36033.998000	NaN
Air_LoCi_11320	8330.288400	NaN
Tab_NuLo_11320	15982.435040	NaN
Pam_NuPa_11320	16145.585040	NaN
Tab_LoGi_11380	111544.520800	NaN
Air_GiPa_11320	1124.454900	NaN
Pam_MeLo_11380	215369.273600	NaN
Pam_PaNu_10737	15171.583500	NaN
Air_CiGi_10747	77261.121600	NaN
Mol_LoCi_11320	18044.128960	NaN
Mol_LoBa_10737	31723.941600	NaN
Pam_LoNu_10747	63486.909200	NaN
Air_LoPa_10737	22773.887100	NaN
Air_CiPa_11320	17293.377920	NaN
Mol_PaBa_11380	149567.904000	NaN
Tab_GiLo_11380	9055.410400	NaN
Air_GiBa_11380	143147.480000	NaN

```
Mol_MeCa_10737   51123.461688       NaN
Air_CaPa_11320    3740.929100       NaN
Mol_CaMe_10737   55679.017680       NaN
Pam_MeNu_11380  193344.236800       NaN
Mol_CiMe_11380  187155.586176       NaN
Mol_CiBa_10737   41350.021128       NaN
Air_BaCa_10747  151815.609504       NaN
Mol_BaCi_10747  176403.027424       NaN
```

[20]:
```
df_aviones.loc[es_destino_NaN | es_duracion_NaN]# aqui nos cogera o uno o tro
↪(or
```

[20]:
```
                Aircompany      Origen     Destino  Distancia        avion  \
Id_vuelo
Pam_MePa_10737   PamPangea   Melbourne         NaN      16925   Boeing 737
Pam_GiNu_11320   PamPangea     Ginebra         NaN       6206  Airbus A320
Mol_CaBa_10747  MoldaviAir       Cádiz         NaN      12798   Boeing 747
Air_BaCi_10737      Airnar        Bali  Cincinnati      15011   Boeing 737
Fly_RoNu_11320        FlyQ        Roma         NaN       6877  Airbus A320
…                      …           …           …         …           …
Tab_GiLo_11380    TabarAir     Ginebra         NaN        739  Airbus A380
Fly_NuBa_11380        FlyQ  Nueva York   Barcelona       6170  Airbus A380
Pam_GiMe_10747   PamPangea     Ginebra   Melbourne      16674   Boeing 747
Mol_LoCi_10737  MoldaviAir     Londres  Cincinnati       6284   Boeing 737
Pam_BaNu_10747   PamPangea        Bali         NaN      16589   Boeing 747

                   consumo_kg  duracion
Id_vuelo
Pam_MePa_10737   46622.417400    1485.0
Pam_GiNu_11320   16200.142400     569.0
Mol_CaBa_10747  156072.121920    1053.0
Air_BaCi_10737   39073.873176       NaN
Fly_RoNu_11320   18131.238008     618.0
…                         …         …
Tab_GiLo_11380    9311.695600      69.0
Fly_NuBa_11380   76317.964000       NaN
Pam_GiMe_10747  188551.726272       NaN
Mol_LoCi_10737   15728.223600       NaN
Pam_BaNu_10747  185751.412496    1305.0

[331 rows x 7 columns]
```

¿Y para todo el `DataFrame` ?

[ ]:
```
df_aviones.isna()# nos devolvera un DF con False lo que no son NaN y true par
↪lo que son
```

Así que para identificar nulos, sigue los pasos: método info, value_counts(dropna = False) e iden-

tificadas las columnas busca que quieres hacer..

0.1.2 Tratamiento de Nulos/NaN (breve intro)

Aquí vamos a ver de una forma somera tres formas de tratar nulos: 1. Eliminar las filas o columnas con nulos. 2. Cambiar los valores de los nulos: Media. 3. Cambiar los valores de los nulos: Moda

Eliminar nulos Si no tenemos datos para algunas columnas y no sabemos como sustituirlos, entonces lo mejor es no considerarlos, bien no considerar las columnas con nulos o bien eliminar toda la fila

El criterio para escoger eliminar la columna o la fila lo veremos con detalle en el futuro, en general, siempre es un compromiso de la información que dejas de usar frente al valor de dicha información.

Eliminando columnas Empezando por las columnas, una vez escogida la columna para eliminar porque tiene nulos, la forma de eliminar esa columna es:

[24]:
```
df_aviones.drop(columns =["Destino"])# nos crea un dataf uevo pero no machaca␣
↪el original para hacerlo inplace = True
```

[24]:

Id_vuelo	Aircompany	Origen	Distancia	avion	consumo_kg	duracion
Air_PaGi_10737	Airnar	París	411	Boeing 737	1028.691900	51.0
Fly_BaRo_10737	FlyQ	Bali	12738	Boeing 737	33479.132544	1167.0
Tab_GiLo_11380	TabarAir	Ginebra	9103	Airbus A380	109439.907200	626.0
Mol_PaCi_10737	MoldaviAir	París	6370	Boeing 737	17027.010000	503.0
Tab_CiRo_10747	TabarAir	Cincinnati	7480	Boeing 747	86115.744000	518.0
…	…	…	…	…	…	…
Tab_LoLo_11320	TabarAir	Los Angeles	8785	Airbus A320	24766.953120	756.0
Mol_CiLo_10737	MoldaviAir	Cincinnati	6284	Boeing 737	16491.729600	497.0
Fly_RoCi_11320	FlyQ	Roma	7480	Airbus A320	19721.049920	662.0
Tab_RoLo_10747	TabarAir	Roma	1433	Boeing 747	15734.053400	115.0
Air_PaLo_10737	Airnar	París	9099	Boeing 737	22331.675700	711.0

```
[1200 rows x 6 columns]
```

Como ocurre con otros métodos ”destructivos”, el método drop no modifica el `DataFrame`que lo invoca, es necesario asignar el resultado o emplear el argumento `inplace` a True

[28]:
```
df_aviones_2 = df_aviones.copy()
df_aviones_2. drop(columns = ["Destino"], inplace = True)
df_aviones_2 # ha desparecido
```

[28]:

Id_vuelo	Aircompany	Origen	Distancia	avion	consumo_kg	duracion
Air_PaGi_10737	Airnar	París	411	Boeing 737	1028.691900	51.0
Fly_BaRo_10737	FlyQ	Bali	12738	Boeing 737	33479.132544	1167.0
Tab_GiLo_11380	TabarAir	Ginebra	9103	Airbus A380	109439.907200	626.0
Mol_PaCi_10737	MoldaviAir	París	6370	Boeing 737	17027.010000	503.0
Tab_CiRo_10747	TabarAir	Cincinnati	7480	Boeing 747	86115.744000	518.0
…	…	…	…	…	…	…
Tab_LoLo_11320	TabarAir	Los Angeles	8785	Airbus A320	24766.953120	756.0
Mol_CiLo_10737	MoldaviAir	Cincinnati	6284	Boeing 737	16491.729600	497.0
Fly_RoCi_11320	FlyQ	Roma	7480	Airbus A320	19721.049920	662.0
Tab_RoLo_10747	TabarAir	Roma	1433	Boeing 747	15734.053400	115.0
Air_PaLo_10737	Airnar	París	9099	Boeing 737	22331.675700	711.0

```
[1200 rows x 6 columns]
```

Observa que a `columns` le pasamos una lista, es decir que podríamos eliminar más de una columna de una vez

[37]:
```
df_aviones_2 = df_aviones.copy()
df_aviones.drop(columns = ["Destino", "duracion"], inplace = True)
df_aviones_2 # ekiminamos las dos columnas a la vez
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
```

```
Cell In[37], line 2
      1 df_aviones_2 = df_aviones.copy()
----> 2 df_aviones.drop(columns = ["Destino", "duracion"], inplace = True)
      3 df_aviones_2 # ekiminamos las dos columnas a la vez

File /usr/local/lib/python3.8/dist-packages/pandas/core/frame.py:5258, in␣
 ↪DataFrame.drop(self, labels, axis, index, columns, level, inplace, errors)
   5110 def drop(
   5111     self,
   5112     labels: IndexLabel = None,
   (…)
   5119     errors: IgnoreRaise = "raise",
   5120 ) -> DataFrame | None:
   5121     """
   5122     Drop specified labels from rows or columns.
   5123
   (…)
   5256             weight  1.0     0.8
   5257     """
-> 5258     return super().drop(
   5259         labels=labels,
   5260         axis=axis,
   5261         index=index,
   5262         columns=columns,
   5263         level=level,
   5264         inplace=inplace,
   5265         errors=errors,
   5266     )

File /usr/local/lib/python3.8/dist-packages/pandas/core/generic.py:4549, in␣
 ↪NDFrame.drop(self, labels, axis, index, columns, level, inplace, errors)
   4547 for axis, labels in axes.items():
   4548     if labels is not None:
-> 4549         obj = obj._drop_axis(labels, axis, level=level, errors=errors)
   4551 if inplace:
   4552     self._update_inplace(obj)

File /usr/local/lib/python3.8/dist-packages/pandas/core/generic.py:4591, in␣
 ↪NDFrame._drop_axis(self, labels, axis, level, errors, only_slice)
   4589         new_axis = axis.drop(labels, level=level, errors=errors)
   4590     else:
-> 4591         new_axis = axis.drop(labels, errors=errors)
   4592     indexer = axis.get_indexer(new_axis)
   4594 # Case for non-unique axis
   4595 else:

File /usr/local/lib/python3.8/dist-packages/pandas/core/indexes/base.py:6699, i␣
 ↪Index.drop(self, labels, errors)
```

```
   6697 if mask.any():
   6698     if errors != "ignore":
-> 6699         raise KeyError(f"{list(labels[mask])} not found in axis")
   6700     indexer = indexer[~mask]
   6701 return self.delete(indexer)

KeyError: "['Destino', 'duracion'] not found in axis"
```

La eliminación de columnas no sólo te sirve para los nulos sino siempre que necesites quitarte alguna columna por la razón que sea.

Eliminando Filas Si lo que queremos no es cargarnos toda la columna, porque consideramos que es un aspecto importante a considerar, deberemos eliminar las filas que no podamos cambiar sus valores NaN. En este caso el método es `dropna` aplicado a filas:

[43]:
```
df_aviones_2 = df_aviones.copy()
df_aviones_2.dropna(axis = "index", inplace = True) # equivalente a poner axis␣
 ↪=0
df_aviones_2.info()# no sale bien deberan salir 869 en vez de 1200 pq ha␣
 ↪eliminado filas con nulo
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1200 entries, Air_PaGi_10737 to Air_PaLo_10737
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Aircompany  1200 non-null   object
 1   Origen      1200 non-null   object
 2   Distancia   1200 non-null   int64
 3   avion       1200 non-null   object
 4   consumo_kg  1200 non-null   float64
dtypes: float64(1), int64(1), object(3)
memory usage: 56.2+ KB
```

Te preguntarás, ¿y esto no sirve también para columnas? Sí, pero quería enseñarte el método drop así que he aprovechado... Pero podemos eliminar las columnas con nulos (todas ojo, el drop es más selectivo) así

[46]:
```
df_aviones_2 = df_aviones.copy()
df_aviones_2.dropna(axis = "columns", inplace = True )# equivalente a poner␣
 ↪axis =1
# aqui te cargas las columnas donde habia nulos, con el drop puedes elegir que␣
 ↪cargarte,
#pq a lo mejor solo queria cargarte la columna destino y las filas que no␣
 ↪tenian columna duracion
df_aviones_2
```

[46]:

Id_vuelo	Aircompany	Origen	Distancia	avion	consumo_kg
Air_PaGi_10737	Airnar	París	411	Boeing 737	1028.691900
Fly_BaRo_10737	FlyQ	Bali	12738	Boeing 737	33479.132544
Tab_GiLo_11380	TabarAir	Ginebra	9103	Airbus A380	109439.907200
Mol_PaCi_10737	MoldaviAir	París	6370	Boeing 737	17027.010000
Tab_CiRo_10747	TabarAir	Cincinnati	7480	Boeing 747	86115.744000
…	…	…	…	…	…
Tab_LoLo_11320	TabarAir	Los Angeles	8785	Airbus A320	24766.953120
Mol_CiLo_10737	MoldaviAir	Cincinnati	6284	Boeing 737	16491.729600
Fly_RoCi_11320	FlyQ	Roma	7480	Airbus A320	19721.049920
Tab_RoLo_10747	TabarAir	Roma	1433	Boeing 747	15734.053400
Air_PaLo_10737	Airnar	París	9099	Boeing 737	22331.675700

[1200 rows x 5 columns]

0.1.3 Sustitucion por media y moda

Una opción frecuente es sustituir los valores numéricos por la media del resto de valores de la misma columna y los valores string por la moda del resto de valores de la columna. La moda es otra forma de decir "el valor más frecuente".

[50]:
```
df_aviones_2 = df_aviones.copy()
```

[51]:
```
df_aviones_2["Destino"].mode()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File /usr/local/lib/python3.8/dist-packages/pandas/core/indexes/base.py:3653, i
 ↪Index.get_loc(self, key)
   3652 try:
-> 3653     return self._engine.get_loc(casted_key)
   3654 except KeyError as err:

File /usr/local/lib/python3.8/dist-packages/pandas/_libs/index.pyx:147, in
 ↪pandas._libs.index.IndexEngine.get_loc()

File /usr/local/lib/python3.8/dist-packages/pandas/_libs/index.pyx:176, in
 ↪pandas._libs.index.IndexEngine.get_loc()

File pandas/_libs/hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

File pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

KeyError: 'Destino'
```

```
The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[51], line 1
----> 1 df_aviones_2["Destino"].mode()

File /usr/local/lib/python3.8/dist-packages/pandas/core/frame.py:3761, in␣
 ↪DataFrame.__getitem__(self, key)
   3759 if self.columns.nlevels > 1:
   3760     return self._getitem_multilevel(key)
-> 3761 indexer = self.columns.get_loc(key)
   3762 if is_integer(indexer):
   3763     indexer = [indexer]

File /usr/local/lib/python3.8/dist-packages/pandas/core/indexes/base.py:3655, i␣
 ↪Index.get_loc(self, key)
   3653     return self._engine.get_loc(casted_key)
   3654 except KeyError as err:
-> 3655     raise KeyError(key) from err
   3656 except TypeError:
   3657     # If we have a listlike key, _check_indexing_error will raise
   3658     #  InvalidIndexError. Otherwise we fall through and re-raise
   3659     #  the TypeError.
   3660     self._check_indexing_error(key)

KeyError: 'Destino'
```

[ ]:
```
# tenbcria que dar dos modas = Bali y Ginebra
```

[ ]:
```
# ahora voy hacer por serie panda por condcion una asignacion directa
```

[52]:
```
df_aviones.loc[df_aviones["Destino"].isna(), "destino"]= df_aviones["destino"].
 ↪mode().valius[1]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File /usr/local/lib/python3.8/dist-packages/pandas/core/indexes/base.py:3653, i␣
 ↪Index.get_loc(self, key)
   3652 try:
-> 3653     return self._engine.get_loc(casted_key)
   3654 except KeyError as err:

File /usr/local/lib/python3.8/dist-packages/pandas/_libs/index.pyx:147, in␣
 ↪pandas._libs.index.IndexEngine.get_loc()

File /usr/local/lib/python3.8/dist-packages/pandas/_libs/index.pyx:176, in␣
 ↪pandas._libs.index.IndexEngine.get_loc()
```

```
File pandas/_libs/hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

File pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

KeyError: 'destino'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[52], line 1
----> 1 df_aviones.loc[df_aviones["Destino"].isna(), "destino"]=␣
 ↪df_aviones["destino"].mode().valius[1]

File /usr/local/lib/python3.8/dist-packages/pandas/core/frame.py:3761, in␣
 ↪DataFrame.__getitem__(self, key)
   3759 if self.columns.nlevels > 1:
   3760     return self._getitem_multilevel(key)
-> 3761 indexer = self.columns.get_loc(key)
   3762 if is_integer(indexer):
   3763     indexer = [indexer]

File /usr/local/lib/python3.8/dist-packages/pandas/core/indexes/base.py:3655, i␣
 ↪Index.get_loc(self, key)
   3653     return self._engine.get_loc(casted_key)
   3654 except KeyError as err:
-> 3655     raise KeyError(key) from err
   3656 except TypeError:
   3657     # If we have a listlike key, _check_indexing_error will raise
   3658     #  InvalidIndexError. Otherwise we fall through and re-raise
   3659     #  the TypeError.
   3660     self._check_indexing_error(key)

KeyError: 'destino'
```

[ ]:
```
# DF aviones, paraq aquellas filas en las que el destino es nulo, quiero que su␣
 ↪columna destino la iguales a la moda
```

[53]:
```
df_aviones.loc[df_aviones["duracion"].isna(), "duracion"] =␣
 ↪df_aviones["duracion"].mean()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File /usr/local/lib/python3.8/dist-packages/pandas/core/indexes/base.py:3653, i␣
 ↪Index.get_loc(self, key)
```

```
   3652 try:
-> 3653     return self._engine.get_loc(casted_key)
   3654 except KeyError as err:

File /usr/local/lib/python3.8/dist-packages/pandas/_libs/index.pyx:147, in␣
 ↪pandas._libs.index.IndexEngine.get_loc()

File /usr/local/lib/python3.8/dist-packages/pandas/_libs/index.pyx:176, in␣
 ↪pandas._libs.index.IndexEngine.get_loc()

File pandas/_libs/hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

File pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

KeyError: 'duracion'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[53], line 1
----> 1 df_aciones.loc[df_aviones["duracion"].isna(), "duracion"] =␣
 ↪df_aviones["duracion"].mean()

File /usr/local/lib/python3.8/dist-packages/pandas/core/frame.py:3761, in␣
 ↪DataFrame.__getitem__(self, key)
   3759 if self.columns.nlevels > 1:
   3760     return self._getitem_multilevel(key)
-> 3761 indexer = self.columns.get_loc(key)
   3762 if is_integer(indexer):
   3763     indexer = [indexer]

File /usr/local/lib/python3.8/dist-packages/pandas/core/indexes/base.py:3655, i␣
 ↪Index.get_loc(self, key)
   3653     return self._engine.get_loc(casted_key)
   3654 except KeyError as err:
-> 3655     raise KeyError(key) from err
   3656 except TypeError:
   3657     # If we have a listlike key, _check_indexing_error will raise
   3658     #  InvalidIndexError. Otherwise we fall through and re-raise
   3659     #  the TypeError.
   3660     self._check_indexing_error(key)

KeyError: 'duracion'
```

[54]:
```
df_aviones.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1200 entries, Air_PaGi_10737 to Air_PaLo_10737
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Aircompany  1200 non-null   object
 1   Origen      1200 non-null   object
 2   Distancia   1200 non-null   int64
 3   avion       1200 non-null   object
 4   consumo_kg  1200 non-null   float64
dtypes: float64(1), int64(1), object(3)
memory usage: 56.2+ KB
```

[56]:
```
df_aviones_2.loc[df_aviones_2["destino"].isna()] # oye df _2( que no lo hemos␣
 ↪cambiado) dime las filas destino que tienen valores nulos
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File /usr/local/lib/python3.8/dist-packages/pandas/core/indexes/base.py:3653, i␣
 ↪Index.get_loc(self, key)
   3652 try:
-> 3653     return self._engine.get_loc(casted_key)
   3654 except KeyError as err:

File /usr/local/lib/python3.8/dist-packages/pandas/_libs/index.pyx:147, in␣
 ↪pandas._libs.index.IndexEngine.get_loc()

File /usr/local/lib/python3.8/dist-packages/pandas/_libs/index.pyx:176, in␣
 ↪pandas._libs.index.IndexEngine.get_loc()

File pandas/_libs/hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

File pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

KeyError: 'destino'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[56], line 1
----> 1 df_aviones_2.loc[df_aviones_2["destino"].isna()] # oye df _2( que no lo
 ↪hemos cambiado) dime las filas destino que tienen valores nulos
```

```
File /usr/local/lib/python3.8/dist-packages/pandas/core/frame.py:3761, in␣
 ↪DataFrame.__getitem__(self, key)
   3759 if self.columns.nlevels > 1:
   3760     return self._getitem_multilevel(key)
-> 3761 indexer = self.columns.get_loc(key)
   3762 if is_integer(indexer):
   3763     indexer = [indexer]

File /usr/local/lib/python3.8/dist-packages/pandas/core/indexes/base.py:3655, i␣
 ↪Index.get_loc(self, key)
   3653     return self._engine.get_loc(casted_key)
   3654 except KeyError as err:
-> 3655     raise KeyError(key) from err
   3656 except TypeError:
   3657     # If we have a listlike key, _check_indexing_error will raise
   3658     #  InvalidIndexError. Otherwise we fall through and re-raise
   3659     #  the TypeError.
   3660     self._check_indexing_error(key)

KeyError: 'destino'
```

[57]:
```
df_aviones.loc["Pam_BaNu_10747"]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[57], line 1
----> 1 df:aviones.loc["Pam_BaNu_10747"]

NameError: name 'aviones' is not defined
```

[ ]:
```
# aqui daria de resultado, ninguno va tener destino a NaN, sino la la moda␣
 ↪(Ginebra
```

07_Apply

November 28, 2023

0.1 Apply: Transformaciones en base a una columna

En esta sesión vamos a ver la capacidad de aplicar funciones definidas por el usuario (y no definidas por el) a los valores de una columna en concreto. Así podremos manipular esos datos para lo que necesitemos.

Lo primero cargarnos unos datos, un poco diferentes, pero muy similares a los de los viajes que hemos visto hasta ahora.

[1]:
```
import numpy as  np
import pandas as pd

df_aviones = pd.read_csv("./data/dataset_dirty_aviones.csv", index_col =␣
 ↪"Id_vuelo")
```

0.1.1 Transformacion y procesado con funciones

Para empezar, echemos un vistazo.

[2]:
```
df_aviones
```

[2]:

Id_vuelo	Aircompany	Origen	Destino	Distancia	avion
Air_PaGi_10737	Airnar	París	Ginebra	411	Boeing 737
Fly_BaRo_10737	FlyQ	Bali	Roma	12738	Boeing 737
Tab_GiLo_11380	TabarAir	Ginebra	Los Angeles	9103	Airbus A380
Mol_PaCi_10737	MoldaviAir	París	Cincinnati	6370	Boeing 737
Tab_CiRo_10747	TabarAir	Cincinnati	Roma	7480	Boeing 747

```
…                    …          …            …         …          …
Tab_LoLo_11320    TabarAir  Los Angeles      Londres      8785  Airbus A320
Mol_CiLo_10737  MoldaviAir   Cincinnati      Londres      6284   Boeing 737
Fly_RoCi_11320        FlyQ         Roma   Cincinnati      7480  Airbus A320
Tab_RoLo_10747    TabarAir         Roma      Londres      1433   Boeing 747
Air_PaLo_10737      Airnar        París  Los Angeles      9099   Boeing 737

                      consumo_kg  duracion
Id_vuelo
Air_PaGi_10737          1028.6919        51
Fly_BaRo_10737  33479.13254400001      1167
Tab_GiLo_11380        109439.9072       626
Mol_PaCi_10737          17027.01        503
Tab_CiRo_10747          86115.744       518
…                             …       …
Tab_LoLo_11320        24766.95312       756
Mol_CiLo_10737         16491.7296       497
Fly_RoCi_11320        19721.04992       662
Tab_RoLo_10747         15734.0534       115
Air_PaLo_10737         22331,6757       711

[1200 rows x 7 columns]
```

Como en los nulos, así de primeras, ¿ves algo diferente?

Así de primeras, no se ve nada muy diferente. Por eso seguimos con lo que nos han pedido, y ¿qué nos han pedido? Pues todavía no te lo he dicho... Nos piden que creemos una columna nueva para clasificar los vuelos en tres categorias: * LD -> Larga Distancia distancia > 10000Km * MD -> Media Distacia distancia entres 2000Km y 10000Km * CD -> Distancia corta -> distancias < 2000 Km

Pues nada, solo tenemos que comparar la distancia de cada vuelo con esos umbrales y... ¿y cómo se procesan los valores de una serie?

[3]:
```
# Solucion "antipattern"
clasificacion = []
 #poner esto en el for es una forma de seleccionar la columnas distancia para␣
 ↪itere por su valores y nos devulva una serie con los mismo indices que los␣
 ↪nombres de las filas y los valores de la columna distancia
for distancia in df_aviones["Distancia"]:
    if distancia > 10000:
        clasificacion.append("LD")
    elif distancia >= 2000:
        clasificacion.append("MD")
    else:
        clasificacion.append("CD")
df_aviones["categoria_vuelo"] = clasificacion# le pasamos los valores de la␣
 ↪lista al dicioonario y panda se encaga de ordenarla como la Distancia
```

[4]:
```
df_aviones
```

[4]:
```
                Aircompany       Origen      Destino  Distancia        avion  \
Id_vuelo
Air_PaGi_10737      Airnar        París      Ginebra        411   Boeing 737
Fly_BaRo_10737        FlyQ         Bali         Roma      12738   Boeing 737
Tab_GiLo_11380   TabarAir      Ginebra  Los Angeles       9103  Airbus A380
Mol_PaCi_10737  MoldaviAir        París   Cincinnati       6370   Boeing 737
Tab_CiRo_10747   TabarAir   Cincinnati         Roma       7480   Boeing 747
…                     …          …           …         …           …
Tab_LoLo_11320   TabarAir  Los Angeles      Londres       8785  Airbus A320
Mol_CiLo_10737  MoldaviAir   Cincinnati      Londres       6284   Boeing 737
Fly_RoCi_11320        FlyQ         Roma   Cincinnati       7480  Airbus A320
Tab_RoLo_10747   TabarAir         Roma      Londres       1433   Boeing 747
Air_PaLo_10737      Airnar        París  Los Angeles       9099   Boeing 737

                      consumo_kg  duracion categoria_vuelo
Id_vuelo
Air_PaGi_10737          1028.6919        51              CD
Fly_BaRo_10737  33479.13254400001      1167              LD
Tab_GiLo_11380       109439.9072       626              MD
Mol_PaCi_10737         17027.01       503              MD
Tab_CiRo_10747         86115.744       518              MD
…                            …         …             …
Tab_LoLo_11320       24766.95312       756              MD
Mol_CiLo_10737         16491.7296       497              MD
Fly_RoCi_11320       19721.04992       662              MD
Tab_RoLo_10747         15734.0534       115              CD
Air_PaLo_10737         22331,6757       711              MD

[1200 rows x 8 columns]
```

[5]:
```
#froma mas rapida
df_aviones["categoria_vuelo"].value_counts() # nos da el numero de vuelos por␣
 ↪categoria
```

[5]:
```
categoria_vuelo
MD    478
LD    460
CD    262
Name: count, dtype: int64
```

Una forma más ”estética” y un poco también más usable es empleando una función

[8]:
```
def clasificador_distancia(serie):
    clasificacion = []
    for distancia in serie:
        if distancia > 10000:
```

```
            clasificacion.append("LD")
        elif distancia >= 2000:
            clasificacion.append("MD")
        elif distancia > 500:
            clasificacion.append("OCD")
        else:
            clasificacion.append("CD")
    return clasificacion
```

Si ahora tuvieramos que hacer cambios sólo los haríamos en la función y santas pascuas.

[10]:
```
df_aviones["categoria_vuelo"]= clasificador_distancia(df_aviones["Distancia"])
```

[12]:
```
df_aviones.categoria_vuelo.value_counts()
```

[12]:
```
categoria_vuelo
MD     478
LD     460
OCD    199
CD      63
Name: count, dtype: int64
```

Aunque efectivo, lo que tienes que ir acostumbrandote a hacer es uso del método apply, y dirás ¿por qué? porque cuando quieras operar con dos o más columnas empezaran los problemas (lo veremos en la siguiente sesion y entonces es mejor ir ya acostumbrandose siempre a usar apply)

0.1.2 Apply

Es un método de las `Series` y de los `DataFrame` que sirve para aplicar funciones valor a valor.

[ ]:
```

```

[ ]:
```

```

Para verlo directamente: nos hacemos un clasificar apply(aplai) que le diras una distancia y en funcion de ese valor nos devolvera la clasificacion

[13]:
```
def clasificador_apply(distancia):
    if distancia > 10000:
            clasificacion = "LD"
    elif distancia >= 2000:
        clasificacion = "MD"
    else:
        clasificacion = "CD"
    return clasificacion
```

[14]:
```
df_aviones["Distancia"].apply(clasificador_apply)# esto va a apolicar a cada␣
↪uno de los elementos de la serie, la funcion, y como se lo aplica elemento a␣
↪elemento, ese elemento es lo valga distancia cada vez que lo llame,
```

```
#va a devolvernos una clasificacion para cada elelemnto pero no solo eso, esto␣
 ↪devuelve una serie que tiene unos valores y con los mismos indices( el que␣
 ↪le corresponde a su serie Distancia)
```

[14]:
```
Id_vuelo
Air_PaGi_10737    CD
Fly_BaRo_10737    LD
Tab_GiLo_11380    MD
Mol_PaCi_10737    MD
Tab_CiRo_10747    MD
                  ..
Tab_LoLo_11320    MD
Mol_CiLo_10737    MD
Fly_RoCi_11320    MD
Tab_RoLo_10747    CD
Air_PaLo_10737    MD
Name: Distancia, Length: 1200, dtype: object
```

Apply devuelve el resultado que devuelva la función para cada valor agrupándolo en una Serie con los mismos indices que la serie de entrada (la que invoca el apply)

[15]:
```
def test_func(distancia):
    if distancia < 200:
        print(distancia)


# como no tiene return devolvera None

resultado = df_aviones["Distancia"].apply(test_func)
resultado # y nos dara un resultado None en toda la columna distancia
```

[15]:
```
Id_vuelo
Air_PaGi_10737    None
Fly_BaRo_10737    None
Tab_GiLo_11380    None
Mol_PaCi_10737    None
Tab_CiRo_10747    None
                  …
Tab_LoLo_11320    None
Mol_CiLo_10737    None
Fly_RoCi_11320    None
Tab_RoLo_10747    None
Air_PaLo_10737    None
Name: Distancia, Length: 1200, dtype: object
```

0.1.3 Datos Susios

Terminemos, o intentémoslo, la sesión con otro ejemplo en el que ya usaremos apply directamente. Ahora queremos clasificar los vuelos por su potencial contaminante.

Considerando el consumo (consumo_kg): * Para mayorers de 8000, categoria C * Para consumos entre 5000 y 8000, categoria B * Para consumos menores de 5000, categoria A

Fácil, ¿no? Sólo tenemos que casi copiar el código de `clasificador_apply`. Venga:

[16]:
```
def clasificador_consumo(consumo):
    if consumo > 8000:
            clasificacion = "C"
    elif consumo >= 5000:
        clasificacion = "B"
    else:
        clasificacion = "A"
    return clasificacion
```

Y ahora lo "aplicamos" (apply)

[17]:
```
resultado = df_aviones["consumo_kg"].apply(clasificador_consumo)
# errro buscado a conciencia: nos dice que estan detectando int y str en los␣
 ↪datos.  la serie si los permite. pero como panda interpreta valor a valor, y␣
 ↪da error
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
e:\Cursos\BC_Data_Science\Repositorio\ONLINE_DS_THEBRIDGE_V\SPRING 4\UNIT␣
 ↪2\07_Apply.ipynb Celda 36 line 1
----> <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/07_Apply.ipynb#X45sZmlsZQ%3D%3D?
 ↪line=0'>1</a> resultado = df_aviones["consumo_kg"].apply(clasificador_consumo

File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\series.
 ↪py:4760, in Series.apply(self, func, convert_dtype, args, by_row, **kwargs)
  4625 def apply(
  4626     self,
  4627     func: AggFuncType,
  (…)
  4632     **kwargs,
  4633 ) -> DataFrame | Series:
  4634     """
  4635     Invoke function on values of Series.
  4636
  (…)
  4751     dtype: float64
  4752     """
  4753     return SeriesApply(
```

```
   4754         self,
   4755         func,
   4756         convert_dtype=convert_dtype,
   4757         by_row=by_row,
   4758         args=args,
   4759         kwargs=kwargs,
-> 4760     ).apply()

File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\apply.
 ↪py:1207, in SeriesApply.apply(self)
   1204     return self.apply_compat()
   1206 # self.func is Callable
-> 1207 return self.apply_standard()

File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\apply.
 ↪py:1287, in SeriesApply.apply_standard(self)
   1281 # row-wise access
   1282 # apply doesn't have a `na_action` keyword and for backward compat␣
 ↪reasons
   1283 # we need to give `na_action="ignore"` for categorical data.
   1284 # TODO: remove the `na_action="ignore"` when that default has been␣
 ↪changed in
   1285 #  Categorical (GH51645).
   1286 action = "ignore" if isinstance(obj.dtype, CategoricalDtype) else None
-> 1287 mapped = obj._map_values(
   1288     mapper=curried, na_action=action, convert=self.convert_dtype
   1289 )
   1291 if len(mapped) and isinstance(mapped[0], ABCSeries):
   1292     # GH#43986 Need to do list(mapped) in order to get treated as neste
   1293     #  See also GH#25959 regarding EA support
   1294     return obj._constructor_expanddim(list(mapped), index=obj.index)

File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\base.
 ↪py:921, in IndexOpsMixin._map_values(self, mapper, na_action, convert)
    918 if isinstance(arr, ExtensionArray):
    919     return arr.map(mapper, na_action=na_action)
--> 921 return algorithms.map_array(arr, mapper, na_action=na_action,␣
 ↪convert=convert)

File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\algorit
 ↪py:1814, in map_array(arr, mapper, na_action, convert)
   1812 values = arr.astype(object, copy=False)
   1813 if na_action is None:
-> 1814     return lib.map_infer(values, mapper, convert=convert)
   1815 else:
```

```
   1816     return lib.map_infer_mask(
   1817         values, mapper, mask=isna(values).view(np.uint8), convert=conve: t
   1818     )

File lib.pyx:2920, in pandas._libs.lib.map_infer()

e:\Cursos\BC_Data_Science\Repositorio\ONLINE_DS_THEBRIDGE_V\SPRING 4\UNIT␣
 ↪2\07_Apply.ipynb Celda 36 line 2
      <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/07_Apply.ipynb#X45sZmlsZQ%3D%3D?
 ↪line=0'>1</a> def clasificador_consumo(consumo):
----> <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/07_Apply.ipynb#X45sZmlsZQ%3D%3D?
 ↪line=1'>2</a>     if consumo > 8000:
      <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/07_Apply.ipynb#X45sZmlsZQ%3D%3D?
 ↪line=2'>3</a>             clasificacion = "C"
      <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/07_Apply.ipynb#X45sZmlsZQ%3D%3D?
 ↪line=3'>4</a>     elif consumo >= 5000:

TypeError: '>' not supported between instances of 'str' and 'int'
```

Hmmm, qué raro, qué está pasando aquí. `consumo_kg` se supone que es un float.

[18]:
```
df_aviones.dtypes # consumo_kg e sun object es decir una mezcla de valores por␣
 ↪eso el error
```

[18]:
```
Aircompany         object
Origen             object
Destino            object
Distancia           int64
avion              object
consumo_kg         object
duracion            int64
categoria_vuelo    object
dtype: object
```

[19]:
```
df_aviones["consumo_kg"].value_counts()# nos han metido comas y puntods,␣
 ↪entonces lo que hay es limpiar los datos
```

[19]:
```
consumo_kg
31607.26077600001    5
18400.052            4
45277.61846400001    4
144578.9548          4
151736.3288          4
                    ..
```

```
1134,771              1
18215.1099            1
17713.7766            1
150952.792            1
22331,6757            1
Name: count, Length: 915, dtype: int64
```

Aghhhh, los datos están "sucios", se han mezclado números con comas y con puntos. Antes de poder hacer el clasificador, tenemos que limpiarlos. Es decir aplicar una función que haga el replace

[20]:
```
# creamos una funcion que cambie las comas por puntos
def reemplaza(consumo):
    if type(consumo)== str:
        return float(consumo.replace(",","."))
    else:
        return consumo.replace
df_aviones["consumo_kg"] = df_aviones["consumo_kg"].apply(reemplaza)
```

[23]:
```
df_aviones.dtypes
```

[23]:
```
Aircompany          object
Origen              object
Destino             object
Distancia            int64
avion               object
consumo_kg         float64
duracion             int64
categoria_vuelo     object
dtype: object
```

[28]:
```
df_aviones["categoria_consumo"] = df_aviones["consumo_kg"].
↪apply(clasificador_consumo)
```

Y ahora sí, podemos clasificar por consumo

[29]:
```
df_aviones. categoria_consumo.value_counts()
```

[29]:
```
categoria_consumo
C    1004
A     154
B      42
Name: count, dtype: int64
```

[ ]:
```

```

08_Apply_Columnas

November 28, 2023

0.1 Apply: Transformaciones sobre varias columnas y sobre selecciones/filtrados

Para mostrarte cómo utilizar apply con varias columnas y cómo hacerlo sobre una selección o filtro, vamos a trabajar con nuestro conjunto de datos de vuelos y sobre dos peticiones nuevas: 1. Clasificar los vuelos según su capacidad contaminante pero teniendo en cuenta varias columnas. 2. Corregir los datos de dos compañías de las que nos han informado que tienen errores en los reportes recogidos en los datos que utilizamos.

Como en otras sesiones, comencemos creando nuestro `DataFrame` a partir de los datos de un fichero, pero esta vez usaremos dos `DataFrame`

[1]:
```
import numpy as  np
import pandas as pd

df_aviones = pd.read_csv("./data/dataset_inicial_aviones.csv", index_col =␣
 ↪"Id_vuelo")
df_aviones_2 = pd.read_csv("./data/dataset_apply_aviones.csv", index_col =␣
 ↪"Id_vuelo")
```

0.1.1 Apply en columnas: Categoria contaminante

Nos piden clasificar los vuelos según las siguientes reglas:

(Consumo por kilometro es consumo_kg/distancia)

- Para vuelos de > 10000Km:
  - Si su el consumo por kilometro es mayor que 11 y la duracion menos de 1000 minutos, cat: MC (muy contaminante)

  - Si su el consumo por kilometro es mayor que 11 y la duracion más de 1000 minutos o su consumo es menor que 11, cat: AC (altamente contaminante)
- Para vuelos de < 10000km:
  - Si su consumo por kilometro es mayor que 10, cat: MC
  - Si su consumo por kilometro es menor que 10, y su duracion menor que 600: AC
  - En cualquier otro caso: C (Contaminante)

Como puedes ver aquí tenemos varias columnas y recurrir a crear columnas intermedias y filtros es un poco más engorroso que crearse una función que opere sobre varias columnas y categorice

Veamos como sería con apply. Supongamos que la función es algo como:

[2]:
```
def cat_contaminacion(distancia, consumo_kg, duracion):
    consumo_km = consumo_kg/distancia
    if distancia > 10000:
        if consumo_km > 11 and duracion > 1000:
            categoria = "MC"
        else:
            categoria = "AC"
    elif distancia < 10000:
        if consumo_km > 10:
            categoria = "MC"
        elif duracion < 600:
            categoria = "AC"
        else:
            categoria = "C"
    return categoria
```

Podemos pensar que la aplicación es:

[ ]:
```
df_aviones[["Distancia", "consumo_kg", "duracion"]].apply(cat_contaminacion)
```

Es decir los valores de las columnas aplicados en el orden de los argumentos, pero...

[3]:
```
df_aviones[["Distancia", "consumo_kg", "duracion"]].apply(cat_contaminacion)#␣
 ↪el typeRrror es pq la funcion requiere 3 argumentoss porque  el apply lo␣
 ↪pasa todo como un argumento(en este caso distancia)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
e:\Cursos\BC_Data_Science\Repositorio\ONLINE_DS_THEBRIDGE_V\SPRING 4\UNIT␣
 ↪2\08_Apply_Columnas.ipynb Celda 14 line 1

----> <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/08_Apply_Columnas.
 ↪ipynb#X16sZmlsZQ%3D%3D?line=0'>1</a> df_aviones[["Distancia", "consumo_kg",␣
 ↪"duracion"]].apply(cat_contaminacion)


File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\frame.
 ↪py:10034, in DataFrame.apply(self, func, axis, raw, result_type, args, by_row␣
 ↪**kwargs)
```

```
  10022 from pandas.core.apply import frame_apply
  10024 op = frame_apply(
  10025     self,
  10026     func=func,
   (…)
  10032     kwargs=kwargs,
  10033 )
> 10034 return op.apply().__finalize__(self, method="apply")

File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\apply.
 ↪py:837, in FrameApply.apply(self)
    834 elif self.raw:
    835     return self.apply_raw()
--> 837 return self.apply_standard()

File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\apply.
 ↪py:963, in FrameApply.apply_standard(self)
    962 def apply_standard(self):
--> 963     results, res_index = self.apply_series_generator()
    965     # wrap results
    966     return self.wrap_results(results, res_index)

File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\apply.
 ↪py:979, in FrameApply.apply_series_generator(self)
    976 with option_context("mode.chained_assignment", None):
    977     for i, v in enumerate(series_gen):
    978         # ignore SettingWithCopy here in case the user mutates
--> 979         results[i] = self.func(v, *self.args, **self.kwargs)
    980         if isinstance(results[i], ABCSeries):
    981             # If we have a view on v, we need to make a copy because
    982             #  series_generator will swap out the underlying data
    983             results[i] = results[i].copy(deep=False)

TypeError: cat_contaminacion() missing 2 required positional arguments:␣
 ↪'consumo_kg' and 'duracion'
```

Puedes ver que sólo le pasa un argumento. Esto es así porque ocurren dos cosas, primero le está pasando los valores columna a columna y segundo sólo se lo va a pasar en el primer argumento.

Si queremos usar todas las columnas a la vez tenemos que que usar el argumento axis = 1, y saber que nos van a pasar los valores en una serie como un único argumento. Es decir, esto tampoco vale:

[4]:
```
df_aviones[["Distancia", "consumo_kg", "duracion"]].apply(cat_contaminacion,␣
 ↪axis =1)# igual porque l devuelve todo en 1 argumento
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
e:\Cursos\BC_Data_Science\Repositorio\ONLINE_DS_THEBRIDGE_V\SPRING 4\UNIT␣
 ↪2\08_Apply_Columnas.ipynb Celda 17 line 1
----> <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/08_Apply_Columnas.
 ↪ipynb#X22sZmlsZQ%3D%3D?line=0'>1</a> df_aviones[["Distancia", "consumo_kg",␣
 ↪"duracion"]].apply(cat_contaminacion, axis =1)

File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\frame.
 ↪py:10034, in DataFrame.apply(self, func, axis, raw, result_type, args, by_row␣
 ↪**kwargs)
  10022 from pandas.core.apply import frame_apply
  10024 op = frame_apply(
  10025     self,
  10026     func=func,
   (…)
  10032     kwargs=kwargs,
  10033 )
> 10034 return op.apply().__finalize__(self, method="apply")

File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\apply.
 ↪py:837, in FrameApply.apply(self)
    834 elif self.raw:
    835     return self.apply_raw()
--> 837 return self.apply_standard()

File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\apply.
 ↪py:963, in FrameApply.apply_standard(self)
    962 def apply_standard(self):
--> 963     results, res_index = self.apply_series_generator()
    965     # wrap results
    966     return self.wrap_results(results, res_index)

File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\apply.
 ↪py:979, in FrameApply.apply_series_generator(self)
    976 with option_context("mode.chained_assignment", None):
    977     for i, v in enumerate(series_gen):
    978         # ignore SettingWithCopy here in case the user mutates
--> 979         results[i] = self.func(v, *self.args, **self.kwargs)
    980         if isinstance(results[i], ABCSeries):
    981             # If we have a view on v, we need to make a copy because
    982             #  series_generator will swap out the underlying data
    983             results[i] = results[i].copy(deep=False)
```

```
TypeError: cat_contaminacion() missing 2 required positional arguments:␣
↪'consumo_kg' and 'duracion'
```

Resumiendo que tenemos que cambiar la definicion de la función y además añadir axis = 1.

[7]:
```
def cat_contaminacion(row):# si tenemos varias columnas, tenemos que poner el␣
↪apply un argumento y decirle a la funcion que lo pase como un unico argumento
    distancia = row["Distancia"]
    consumo_kg = row["consumo_kg"]
    duracion = row["duracion"]

    consumo_km = consumo_kg/distancia
    if distancia > 10000:
        if consumo_km > 11 and duracion > 1000:
            categoria = "MC"
        else:
            categoria = "AC"
    elif distancia < 10000:
        if consumo_km > 10:
            categoria = "MC"
        elif duracion < 600:
            categoria = "AC"
        else:
            categoria = "C"
    return categoria
```

Y ahora ya con axis = 1:

[8]:
```
df_aviones["Cat_Contaminacion"]= df_aviones[["Distancia", "consumo_kg",␣
↪"duracion"]].apply(cat_contaminacion, axis=1) # axis 1 te poasa todos los␣
↪valores a la vez en todas las columnas
```

[9]:
```
df_aviones
```

[9]:

Id_vuelo	Aircompany	Origen	Destino	Distancia	avion	\
Air_PaGi_10737	Airnar	París	Ginebra	411	Boeing 737	
Fly_BaRo_10737	FlyQ	Bali	Roma	12738	Boeing 737	
Tab_GiLo_11380	TabarAir	Ginebra	Los Angeles	9103	Airbus A380	
Mol_PaCi_10737	MoldaviAir	París	Cincinnati	6370	Boeing 737	
Tab_CiRo_10747	TabarAir	Cincinnati	Roma	7480	Boeing 747	
…	…	…	…	…	…	
Tab_LoLo_11320	TabarAir	Los Angeles	Londres	8785	Airbus A320	
Mol_CiLo_10737	MoldaviAir	Cincinnati	Londres	6284	Boeing 737	
Fly_RoCi_11320	FlyQ	Roma	Cincinnati	7480	Airbus A320	
Tab_RoLo_10747	TabarAir	Roma	Londres	1433	Boeing 747	
Air_PaLo_10737	Airnar	París	Los Angeles	9099	Boeing 737	

```
                  consumo_kg  duracion Cat_Contaminacion
Id_vuelo
Air_PaGi_10737    1028.691900        51                AC
Fly_BaRo_10737   33479.132544      1167                AC
Tab_GiLo_11380  109439.907200       626                MC
Mol_PaCi_10737   17027.010000       503                AC
Tab_CiRo_10747   86115.744000       518                MC
…                         …         …                 …
Tab_LoLo_11320   24766.953120       756                 C
Mol_CiLo_10737   16491.729600       497                AC
Fly_RoCi_11320   19721.049920       662                 C
Tab_RoLo_10747   15734.053400       115                MC
Air_PaLo_10737   22331.675700       711                 C

[1200 rows x 8 columns]
```

0.1.2 Apply en seleccion: Corrigiendo datos

Para terminar, corrijamos el `DataFrame` df_aviones_2:

[10]:
```
df_aviones_2
```

[10]:
```
                Aircompany       Origen      Destino  Distancia        avion  \
Id_vuelo
Air_PaGi_10737      Airnar        París      Ginebra        411   Boeing 737
Fly_BaRo_10737        FlyQ         Bali         Roma      12738   Boeing 737
Tab_GiLo_11380    TabarAir      Ginebra  Los Angeles       9103  Airbus A380
Mol_PaCi_10737  MoldaviAir        París   Cincinnati       6370   Boeing 737
Tab_CiRo_10747    TabarAir   Cincinnati         Roma       7480   Boeing 747
…                      …           …           …         …          …
Tab_LoLo_11320    TabarAir  Los Angeles      Londres       8785  Airbus A320
Mol_CiLo_10737  MoldaviAir   Cincinnati      Londres       6284   Boeing 737
Fly_RoCi_11320        FlyQ         Roma   Cincinnati       7480  Airbus A320
Tab_RoLo_10747    TabarAir         Roma      Londres       1433   Boeing 747
Air_PaLo_10737      Airnar        París  Los Angeles       9099   Boeing 737

                  consumo_kg duracion
Id_vuelo
Air_PaGi_10737    1028.691900   d:3060
Fly_BaRo_10737   33479.132544     1167
Tab_GiLo_11380  109439.907200      626
Mol_PaCi_10737   17027.010000  d:30180
Tab_CiRo_10747   86115.744000      518
…                         …        …
Tab_LoLo_11320   24766.953120      756
Mol_CiLo_10737   16491.729600  d:29820
Fly_RoCi_11320   19721.049920      662
```

```
Tab_RoLo_10747   15734.053400      115
Air_PaLo_10737   22331.675700  d:42660

[1200 rows x 7 columns]
```

Dos compañías, Airnar y MoldaviAir, han entregado mal sus datos. Después de preguntarles nos confirman que ambas por error han utilizado una codificación antigüa. Hay que quitar la "d:" y dividir entre 60 para tener la duración correcta. Pues nada construyamos la función y luego se la aplicamos (apply) a la columna `duracion`...

[17]:
```
def corrige_duracion(row):
    new_row = int(row.replace("d:",""))/60
    return int(new_row)
```

La función tiene buena pinta:

[19]:
```
corrige_duracion("d:3060")
```

[19]:
```
51
```

Pero sólo tenemos que aplicarla a los viajes de las compañías indicadas, tenemos que hacer un apply sobre una selección. Creemos la condición de la selección: **funcion isin()** sirve para comprbobar si el valor de una columna esta dentro de una lista y asi me ahhoro poner la condiones, es util si el valort esta dentro de un conjunto de valores

[21]:
```
es_bad_company = df_aviones_2["Aircompany"].isin(["Airnar","MoldaviAir"])
```

Y ahora, fíjate en la sintaxis:

[22]:
```
df_aviones_2.loc[es_bad_company,"duracion" ].apply(corrige_duracion)
```

[22]:
```
Id_vuelo
Air_PaGi_10737      51
Mol_PaCi_10737     503
Mol_CaMe_10737    1721
Mol_PaLo_11320      44
Air_GiCa_11380     135
                  …
Air_GiCa_10747     135
Mol_BaLo_10737    1153
Air_GiCa_11320     145
Mol_CiLo_10737     497
Air_PaLo_10737     711
Name: duracion, Length: 482, dtype: int64
```

Perfecto, pero ahora hay que hacer la asignacion y solo a esos vuelos, pues así:

[23]:
```
df_aviones_2.loc[es_bad_company, "duracion"] = df_aviones_2.loc[es_bad_company,␣
↪"duracion"].apply(corrige_duracion)
```

[ ]:

[24]:
```
df_aviones_2
```

[24]:
```
                Aircompany       Origen      Destino  Distancia        avion  \
Id_vuelo
Air_PaGi_10737      Airnar        París      Ginebra        411   Boeing 737
Fly_BaRo_10737        FlyQ         Bali         Roma      12738   Boeing 737
Tab_GiLo_11380    TabarAir      Ginebra  Los Angeles       9103  Airbus A380
Mol_PaCi_10737  MoldaviAir        París   Cincinnati       6370   Boeing 737
Tab_CiRo_10747    TabarAir   Cincinnati         Roma       7480   Boeing 747
…                    …          …           …         …          …
Tab_LoLo_11320    TabarAir  Los Angeles      Londres       8785  Airbus A320
Mol_CiLo_10737  MoldaviAir   Cincinnati      Londres       6284   Boeing 737
Fly_RoCi_11320        FlyQ         Roma   Cincinnati       7480  Airbus A320
Tab_RoLo_10747    TabarAir         Roma      Londres       1433   Boeing 747
Air_PaLo_10737      Airnar        París  Los Angeles       9099   Boeing 737

                   consumo_kg duracion
Id_vuelo
Air_PaGi_10737    1028.691900       51
Fly_BaRo_10737   33479.132544     1167
Tab_GiLo_11380  109439.907200      626
Mol_PaCi_10737   17027.010000      503
Tab_CiRo_10747   86115.744000      518
…                         …      …
Tab_LoLo_11320   24766.953120      756
Mol_CiLo_10737   16491.729600      497
Fly_RoCi_11320   19721.049920      662
Tab_RoLo_10747   15734.053400      115
Air_PaLo_10737   22331.675700      711

[1200 rows x 7 columns]
```

[ ]:

09_Groupby

November 28, 2023

0.1 Groupby

Como introducción al groupby (sí, eso ”agrupar por”) vamos a plantear unas cuestiones a nuestros datos de viajes aéreos que ya planteamos en sesiones anteriores. Recordaremos como las resolvimos y eso nos dará pie a ver una forma más eficiente de hacerlo a través de las agrupaciones con groupby y de ahí completaremos con más detalles y posibilidades adicionales.

Por tanto, como en otras sesiones, comencemos creando nuestro `DataFrame` a partir de los datos de vuelos:

[ ]:
```
import numpy as  np
import pandas as pd

df_aviones = pd.read_csv("./data/dataset_inicial_aviones.csv", index_col =␣
 ↪"Id_vuelo")
```

0.1.1 ¿Para qué groupby?

En sesiones anteriores le ”preguntamos” a nuestro `DataFrame`, `df_aviones`, por la media de consumo de los aviones por compañía, o algo parecido... y lo hacíamo de esta forma, válida, pero farragosa:

[ ]:
```
for compania in df_aviones["Aircompany"].unique():
    es_compania = df_aviones["Aircompany"] == compania
    print(f"Para la compañía {compania}:")
    for avion in df_aviones["avion"].unique():
        es_avion = df_aviones["avion"] == avion
        consumo = df_aviones.loc[es_compania & es_avion,"consumo_kg"].mean()
```

```
        print(f"Tipo <{avion}> consumo medio por vuelo <{consumo:.2f}>")
```

Esta forma aunque eficaz en términos humanos, es decir una persona tiene la info que quería, no es muy buena en términos de programacion por dos motivos: 1. El código es farragoso y aún haciéndolo una función que tuviera las columnas como parámetro sería complejo mantenerlo. 2. La salida no es muy manejable posteriormente, aunque es verdad que podríamos crearnos una estructura de salida, complicando aún más el código.

Es para este tipo de situaciones para lo que aparece el método groupby, pero ojo no solo para estas como iremos viendo en esta y las siguientes sesiones.

[ ]:

Compara los dos "trozos" de código... Sin duda es mejor el segundo ya sólo en tiempo de escritura... Pero además la salida del groupby la podemos almacenar directamente en una variable

[ ]:

Nos ha devuelto un objeto `Series` de Pandas que podemos manipular como cualquier otro `Series`:

[ ]:

[ ]:

El índice es un indice multidimensional (cada valor es una tupla) en los que no vamos a profundizar, pero que se puede operar como cualquier otro indice de una serie:

[ ]:

0.1.2 Groupby con cierto detalle

El método groupby de un `DataFrame` (también lo hay para `Series` pero eso puedes consultarlo aquí) tiene varios argumentos interesantes, el primero ya lo hemos visto la lista de columnas por las que queremos agrupar (puede ser sólo una) y otro es al argumento `as_index` que veremos luego. Antes es interesante destacar que groupby es un método "vago" [si como Jose Mota, si hay que ir se va pero ir pa ná]. Veámoslo, ejecutando solo el groupby:

[ ]:

Nos ha devuelto un objeto groupby, eso y nada de primeras es lo mismo. Para obtener algo hay que decir qué tiene que hacer con las columnas restantes:

[ ]:

En este caso, no como en el primer que previamente a la función de "agregacion" (mean en ese caso) dijimos que sólo queríamos operar sobre la columna "consumo_kg", nos devuelve un `DataFrame`. Comprobémoslo:

[ ]:

Como ya has visto podemos hacer el agrupado y luego sólo coger varias columnas y actuar sobre ellas, con diferentes funciones... Calcula ahora por Compañía y Destino el número de vuelos, venga...:

[ ]:

[ ]:

[ ]:

Pero puede que queramos ver por ejemplo la duración media del vuelo y el consumo medio realizado por Compañía, Origen, Destino:

[ ]:

Nos da error, porque si la función de agregación no se puede aplicar a alguna de las columnas que no hacen agrupación y no se han filtrado, en este caso "avion" que es un str no admite media, da un error (contar filas no da error :-)

Tendríamos que filtar previamente como hemos hecho con "consumo_kg", y quedarnos con las columnas que queremos o por lo menos con aquellas para las que la función final que aplicamos (mean, en este caso) sea válida

[ ]:

[ ]:

Pero son muchos resultados, ¿como filtro? Pues aunque no hablermos mucho más de multindice, miremos varios ejemplos:

[ ]:
```
# Resultados para la compañía Airnar


# Resultados para todos los vuelos París-Cádiz


# Resultados para los vuelos a Ginebra y Nueva York de FlyQ
```

Nosotros, en general, si queremos conservar la salida de un groupby para seguir procesándola, emplearemos el argumento `as_index` con valor "False"

[ ]:
```
resultado_no_index = df_aviones.groupby(["Aircompany","Origen","Destino"],␣
↪as_index = False)[["consumo_kg","duracion"]].mean()
```

[ ]:

Porque de esta forma seguiremos teniendo toda la información en columnas, eso sí como se han agrupado observa que los índices originales (los identificadores del vuelo) pierden sentido y se pierden...

10_Funciones_agrupacion

November 28, 2023

0.1 Groupby: Funciones Agregación Avanzadas

Hasta ahora hemos visto funciones de agregación sencillas (la función que se pone al final del groupby para que "haga algo" con las columnas indicadas), en esta sesión vamos a aumentar esa capacidad. ¿Cómo? Pues aplicando las funciones por columna, es decir no siempre la misma funión a cada columna, y además pudiendo hacer que sea una función de usuario (al estilo apply, del que tanto te acuerdas)

Y, ¿cómo no?, comencemos creando nuestro `DataFrame` a partir de los datos de vuelos:

[ ]:
```
import numpy as  np
import pandas as pd

df_aviones = pd.read_csv("./data/dataset_inicial_aviones.csv", index_col =␣
 ↪"Id_vuelo")
```

0.1.1 Agg: Funciones de usuario

Supongamos que ahora queremos obtener por Compañía y tipo de Avión, el número de vuelos y además el consumo medio y el destino más visitado. Hala ahí es nada. Descompongámoslo en tres pequeños problemas y luego veremos como juntarlos con la ayuda del método `agg`

1. Por compañia y tipo de avión, el número de vuelos. Ese es fácil, lo hicimos en la píldora anterior:

[ ]:
```

```

2. Por compañia y tipo de avión, el consumo medio. También lo tengo:

[ ]:

Para terminar:
3. Por compañía y tipo de avion el destino más visitado. Hmmm, está... ah, ya

Necesito calcular un estadístico la "moda", del que hablamos en algún momento y que viene a decirnos de una serie de valores (numeros o cadenas) aquel que más se repite. Las series tienen el metodo mode:

[ ]:

Pero desgraciadamente las series que devuelve un groupby....:

[ ]:

Vale, he hecho trampa para enseñarte primero como aplicar una función definida por el usuario. Aquí lo que tienes que tener en cuenta es que cómo parámetro o argumento la función va a recibir una serie por agrupación...

Si, como hemos venido haciendo, agrupamos por Compañía y vuelo y aplicamos una función (como ahora mismo veremos) esa función va a recibir una serie con todos los valores para "Airnar","Airbur A380" de una vez, y luego otra serie con todos los valores para "Airna","Airbus 320" y luego se llamará con todos los valores de "Airnar","Boeing 737" y así tantas llamadas como combinaciones de valores de "Aircompany" y "avion" (porque son las columnas que hemos elegidos) y en cada llamada todos los valores. Y qué valores pues los de las columnas que hayamos escogido después del groupby. Vale, que necesitas un ejemplo... Primero la función:

[ ]:

Ahora la forma de aplicarlo, sí eso... con apply:

[ ]:

Vale ya tenemos las tres cosas que queríamos.... por separado, ¿Cómo lo hago a la vez? Con agg o aggregate

0.1.2 Agg, aggregate: aplicando funciones y operaciones diferentes por columnas

Muy sencillo, vamos a usar el método agg o el aggregate al que se le pasa un diccionario:
{nombre_de_la_columna: operacion_sobre_esa_columna,
nombre_de_la_columna2: operacion_sobre_esa_columna,...
nombre_de_la_columnan: operacion_sobre_esa_columna}

Venga, en formato diccionario:

[ ]:

Para más info aquí (donde puedes ver también no más formatos para pasar las funciones), y ahora:

[ ]:

Por supuesto, podemos usar `as_index`:

[ ]:

11_Transform

November 28, 2023

0.1 Groupby: Transform

Vamos a dedicar esta sesión al metodo transform como cierre de las sesiones dedicadas a las agrupaciones hechas con groupby. Pero como tónica habitual de esta unidad, carga datos en nuestro `DataFrame` guía:

[ ]:
```
import numpy as  np
import pandas as pd

df_aviones = pd.read_csv("./data/dataset_inicial_aviones.csv", index_col =␣
 ↪"Id_vuelo")
```

0.1.1 Transform fuera del *groupby*

Aunque lo vamos a ver en el contexto de las agrupaciones hechas con "groupby", `transform` es un método que también se aplica a `Series` y `DataFrame` sin que haya un `groupby` de por medio.

Es similar a `apply`, y sus principales diferencias son: * Puede aplicar una o varias funciones (sí varias funciones a la vez, introducidas como lista o diccionario) * Sólo se puede aplicar a una serie (o columna) a la vez (no vale el método de varias columnas)

Supongamos que queremos pasar los Origenes y los Destinos a todo mayúsculas y además generar una abreviatura con las tres primeras letras, en vez de hacerlo en dos pasadas podemos:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

0.1.2 Transform para agrupaciones

La diferencia más importante respecto a apply o agg es que `transform` devuelve un serie con tantos elementos como la serie que se le pasa como input (recuerda que `transform` solo puede manejar un "columna" o serie a la vez). En concreto para los elementos de una misma agrupación devuelve el valor calculado por la función, veámoslo con un ejemplo.

[ ]:
```
### Sin transform
```

[ ]:
```
### Con transform
```

Fijate en el número de elementos y los índices de la salida de la línea de código con `transform`

¿Y esto para que sirve? Típicamente para pegar a cada elemento individual su valor de agrupación y poder utilizarlo después

Por ejemplo:

[ ]:

[ ]:

Y ahora podemos hacer cálculos directos que de otra forma serían más complejos de conseguir. Vamos a crearnos otra columna que recoja para cada vuelo el porcentaje sobre la media de su duración lo que nos permitirá luego por ejemplo hacer control de vuelos que se nos desvían mucho o poco de la media.

Tal como lo tenemos es fácil hacer esa columna

[ ]:

[ ]:

Ahora piensa tú en cómo harías para calcular la columna "desviacion_duracion" sin usar el transform previo... te lo dejo como ejercicio a ver en la sesión en vivo.