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]