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]
```

[ ]: