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
```

[ ]:
```

```