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
```