04_Tipos_de_Datos

November 29, 2023

ETL Y DATOS

0.1 ETL: TIPOS DE DATOS

Vamos a tratar brevemente los tipos de datos o mejor tipo de valores que podemos encontrarnos en un conjunto de datos desde una perspectiva de Data Scientist. Es decir, no hablamos de tipos de datos en genérico como en píldoras anteriores, ni tampoco de tipos de datos de Python, numpy o pandas.

Para ello carguemos una vez más nuestro dataset de viajes aéreos (que pronto nos dejará por otros datos diferentes):

[1]:
```
import pandas as pd
import random
df_datos = pd.read_csv("./data/dataset_ETL_full_aviones.csv")
```

0.2 Tipos de datos

En el contexto de la ciencia de datos o Data Science se suele hablar de los siguientes tipos de datos (no es una lista exahustiva pero sí son todos los que están): * Numéricos * Categóricos (incluye booleanos) * Tipo Texto * Tipo Fecha * Multimedia: Imágenes, vídeos, Audio

Hasta que tratemos el procesamiento de lenguaje natural (NLP) y las imágenes con redes, nosotros nos vamos a centrar en lo que denominamos datos estructurados o tabulares en las sesiones anteriores y dentro de este conjunto de datos en los tipos de valores no multimedia. Véamos que pinta tiene cada uno

0.2.1 Numéricos

Es decir los campos o características o columnas que contienen valores numéricos y por tanto susceptibles de ser operados matemáticamente. Si miramos nuestro dataset:

[ ]:

”Distancia”,”consumo_kg”,”duracion”, ”ingresos” y el indicador ”IC” son valores numéricos. Podemos calcular sus medias, varianzas, distribuciones, ver esas distribuciones gráficamente, etc.

[2]:
```
df_datos.info()
```

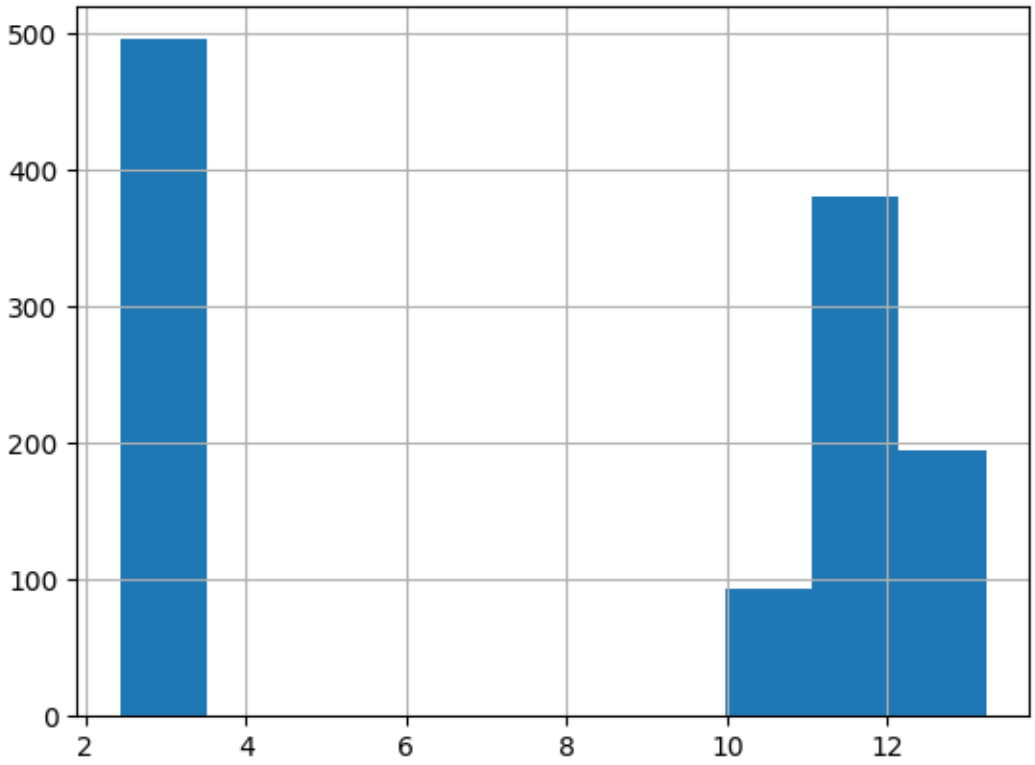
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1166 entries, 0 to 1165
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Aircompany    1166 non-null   object
 1   Origen        1166 non-null   object
 2   Destino       1166 non-null   object
 3   Distancia     1166 non-null   int64
 4   avion         1166 non-null   object
 5   con_escala    1166 non-null   bool
 6   consumo_kg    1166 non-null   float64
 7   duracion      1166 non-null   int64
 8   ingresos      1166 non-null   float64
 9   Id_vuelo      1166 non-null   object
 10  IC            1166 non-null   float64
 11  Categoria_IC  1166 non-null   object
 12  Incidencias   1166 non-null   object
 13  Hora_Vuelo    1166 non-null   object
dtypes: bool(1), float64(3), int64(2), object(8)
memory usage: 119.7+ KB
```

[3]:
```
df_datos.IC.value_counts()
```

[3]:
```
IC
10.660000    28
11.906800    28
12.022400    27
12.369200    26
11.086400    25
             ..
2.819232      1
2.610400      1
11.193000     1
2.628288      1
2.535100      1
Name: count, Length: 147, dtype: int64
```

[4]:
```
df_datos.IC.hist()
```

[4]:
```
<Axes: >
```

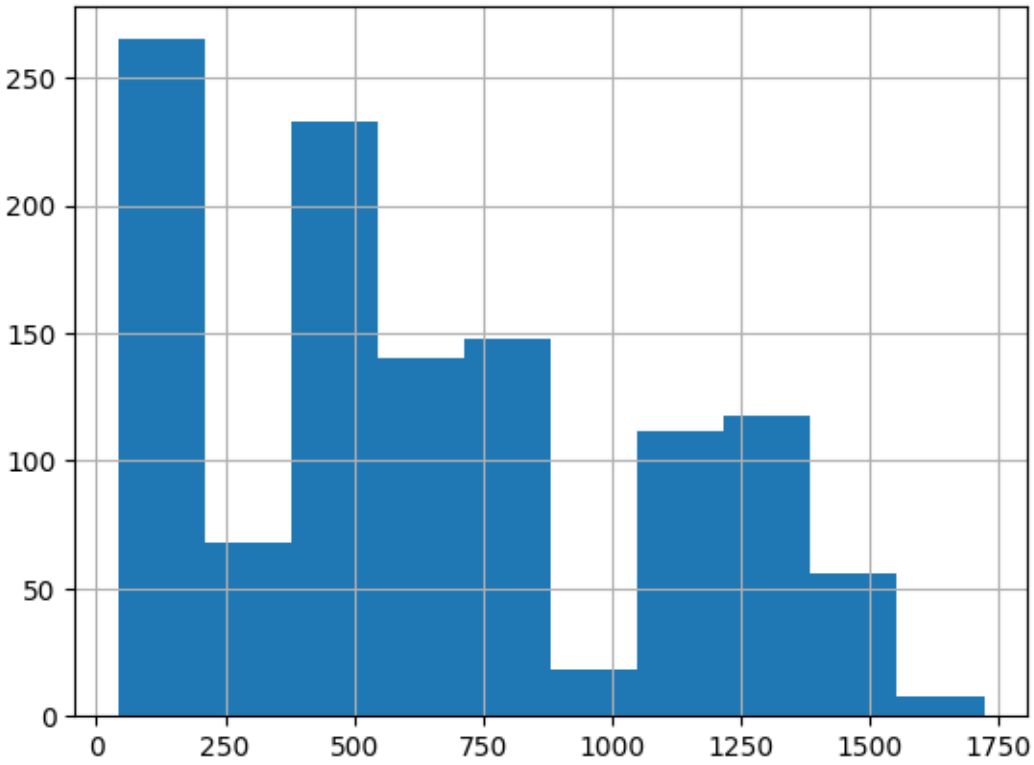


[8]:
```
df_datos.duracion.value_counts()
```

[8]:
```
duracion
818     38
845     36
1326    28
433     26
856     24
        ..
687      2
151      2
488      2
129      1
731      1
Name: count, Length: 116, dtype: int64
```

[9]:
```
df_datos["duracion"].hist()
```

[9]:
```
<Axes: >
```



Muchas veces es interesante en la misma exploración para ”transformar y limpiar” hacer pequeñas visualizaciones para entender cómo están los datos.

[ ]:

Para hacer un EDA no es necesario, pero cuando entremos en Machine y Deep Learning veremos que todos los valores, todas las columnas deben pasar a valores numéricos y ya veremos cómo hacerlo.

0.2.2 Categóricos

Con categóricos nos referimos a los valores (columnas en nuestro caso) de datos cuyo rango de valores está limitado y que suelen representar categorías o etiquetas. Por ejemplo, la columna ”Categoria_IC”:

[10]:
```
df_datos["Categoria_IC"].value_counts()
```

[10]:
```
Categoria_IC
C    576
A    496
B     94
Name: count, dtype: int64
```

Pero también podríamos considerar los Orígenes y Destinos, para este Dataset concreto, ojo, podemos verlos como "etiquetas" o datos de categorías:

[13]:
```
df_datos["Origen"].value_counts()
```

[13]:
```
Origen
Bali           161
Ginebra        149
Cincinnati     136
Londres        127
Nueva York     123
París          107
Melbourne       99
Los Angeles     86
Cádiz           70
Roma            66
Barcelona       42
Name: count, dtype: int64
```

[14]:
```
len(df_datos.Destino.unique())#calcular el numero de destinos unicos
```

[14]:
```
11
```

El proceso para que pensemos en clasificar un dato, columna o característica como categórica (además de que nos lo digan exprofeso) es el de calcular la cardinalidad:

$$Cardinalidad(< Columna/Caracterstica >) = \frac{NmerodeValoresUnicos(< Columna/Caracterstica >)}{NmeroTotalDeRegistros} * 100\%$$

Por ejemplo, la cardinalidad para Categoria_IC

[19]:
```
cardinalidad_IC = len(df_datos["Categoria_IC"].unique())/len(df_datos) *100
print(cardinalidad_IC)
```

```
0.2572898799313894
```

La de Destino (empleando el método de pandas, `nunique`):

[23]:
```
cardinalidad_destino = df_datos["Destino"].nunique()/len(df_datos)*100
print(cardinalidad_destino)
```

```
0.9433962264150944
```

Menos del 1%, compara ahora con la de los valores numéricos:

[25]:
```
cardinalidad_consumo = df_datos["consumo_kg"].nunique()/len(df_datos)*100
print(cardinalidad_consumo)
```

```
71.09777015437393
```

Prueba tu a calcular la cardinalidad de "ingresos", por ejemplo.

[26]:
```
cardinalidad_ingresos = df_datos["ingresos"].nunique()/len(df_datos)*100
print(cardinalidad_ingresos)
```

```
100.0
```

Con cardinalidades por debajo del 5-10% ya podríamos considerar el valor o columna como categórico. ¿Y eso por qué es importante? Por dos motivos principales: 1. Para el EDA: Suponen, en general, interesantes columnas o características para hacer agrupaciones y explorarlas 2. Para Machine y Deep learnnig: Existen métodos de transformación a números bastante potentes y utilizados, por lo que suelen ser buenas características para incluir

En general, los datos categóricos suelen ser etiquetas de texto, pero podemos hacer datos "categóricos" de datos numéricos y los datos booleanos (como "con_escala") se suelen considerar datos categóricos con dos categorías (Verdadero o Falso, 1 o 0)

Ah y no pierdas de vista el concepto de Cardinalidad de un conjunto de datos porque lo vamos a usar muchas más veces.

[Nota: Extrictamente hablando cardinalidad es el número de valores diferentes, pero nosotros vamos a emplearlo como el porcentaje mostrado antes]

0.2.3 Tipo Texto

Por valores de tipo texto entendemos aquellas columnas que son texto y no se pueden asimilar directamente a una categoría. Consideremos por ejemplo la columna "Incidencias":

[28]:
```
df_datos.Incidencias.value_counts()
```

[28]:
```
Incidencias
Sin incidencias
510
Mal funcionamiento del tren de aterrizaje
33
Turbulencia severa
31
Ave impacta con el avión
29
Aterrizaje de emergencia
25
          …
Condiciones meteorológicas malas y Emergencia médica a bordo
1
Emergencia en el sistema de entretenimiento y Mal funcionamiento del tren de
aterrizaje      1
Problemas de navegación,
1
Sambódromo a bordo y Conflictos entre pasajeros
1
```

```
Problemas con el suministro de combustible,
1
Name: count, Length: 154, dtype: int64
```

[30]:
```
cardinalidad_incidencias = df_datos.Incidencias.nunique()/len(df_datos)*100
print(cardinalidad_incidencias)
```

```
13.20754716981132
```

Considerando la cardinalidad, podríamos intentar hacer de esta columna una categórica pero observando el tipo de dato que tiene, con lenguaje natural que puede cambiar y en general es de por si desestructurado y donde las abreviaturas, sinónimos, etc, lo hacen difícil de considerar de primeras, lo normal es considerarlo como un campo de texto libre y tratarlo a partir de ahí.

En general, lo trataremos con técnicas de procesamiento de lenguaje natural y, otras veces además, obtendremos otras columnas con datos resumidos a partir de estas columnas de texto.

0.2.4 Tipo Fecha

Como su nombre indica hace referencia a los valores que son fechas. El hecho de que no sigan el sistema decimal, y tengan sus particularidades de calendario (vamos que 45 minutos más 55 minutos no sean 1 hora, sino 1 hora y 30 minutos, y que del 28 de Febrero a veces pasemos al 29 y otras al 1 de Marzo, por ejemplo) hace que haya que tratarlos de una forma un poco especial o de aprovechar las capacidades de Python y pandas para hacerlo.

En nuestro dataset, claramente "Hora_Vuelo" es de este tipo.

[31]:
```
df_datos["Hora_Vuelo"].value_counts()
```

[31]:
```
Hora_Vuelo
23:00 19/04/2023    1
09:15 05/04/2023    1
10:45 06/03/2023    1
07:30 30/01/2023    1
16:15 29/05/2023    1
                   ..
17:00 11/06/2023    1
22:00 10/02/2023    1
23:15 17/09/2023    1
06:00 08/06/2023    1
13:45 12/03/2023    1
Name: count, Length: 1166, dtype: int64
```

Pero si miramos el tipo que considera pandas:

[32]:
```
df_datos["Hora_Vuelo"].dtypes# tipo objeto
```

[32]:
```
dtype('O')
```

Hace referencia a que es un "object" o sea un string o cualquier mezcla de tipos. Tendremos que hacer conversiones especiales y tratarlas con cariño para sacar el potencial a las fechas. Lo veremos

en la pildora correspondiente.

[ ]:

[ ]:

[ ]:

[ ]:

06_Transformacion_Texto_III

November 29, 2023

ETL Y DATOS

0.1 ETL: Tratamiento de Textos (II)

Además de limpiar los procesos de transformación del ciclo ETL se encargan de obtener o crear nuevas columnas para potenciar el análisis de datos. Qué columnas debemos crear dependerán del problema o investigación que estemos tratando y de los datos.

En esta sesión vamos a ver dos ejemplos pero a lo largo del bootcamp verás muchoas más que te servirán de guía para cuando tengas que aplicarlo por ti mismo.

Esta vez vamos a emplear dos datasets, nuestro conocido de vuelos y un nuevo más deportivo.

[1]:
```
import pandas as pd
import random
df_viajes = pd.read_csv("./data/df_viajes.csv")
df_liga  = pd.read_csv("./data/df_liga_2019.csv")
```

0.1.1 Sacando provecho de las columnas/campos tipo Texto: Categorización

En sesiones anteriores comentamos que un campo de tipo texto es necesario, en general, tratarlo con técnicas de procesamiento de lenguaje natura (o NLP), pero no siempre tendremos que ser tan sofisticados para obtener algún valor de los mismos.

Veamos la columna "Incidencias" del dataset de viajes:

[2]:
```
df_viajes.Incidencias.sample(30)
```

[2]:
```
728                                  Sin incidencias
123            Retraso extenso en la pista de despegue
633                                  Sin incidencias
```

```
744                                 Sambódromo a bordo,
973                            Sambódromo entre pasajeros
129                                        Sin incidencias
952                  Emergencia grave eléctrico general,
324                                      Turbulencia severa
719            Mal funcionamiento del tren de aterrizaje
46                                         Sin incidencias
556                                        Sin incidencias
162                                        Sin incidencias
695                           Emergencia eléctrico general
8              Problema en el sistema de entretenimiento
756                                        Sin incidencias
729     Condiciones meteorológicas ligeramente complejas
62             Mal funcionamiento del tren de aterrizaje
924     Emergencia grave de los sistemas de comunicación
763           Mal funcionamiento del tren de aterrizaje,
373     Condiciones meteorológicas ligeramente complejas
142                                        Sin incidencias
352                           Amenaza de seguridad a bordo
417                                        Sin incidencias
484                                        Sin incidencias
935                                        Sin incidencias
113                                        Sin incidencias
776                                        Sin incidencias
97                                         Sin incidencias
41                               Aterrizaje de emergencia
83                                       Turbulencia severa
Name: Incidencias, dtype: object
```

[4]:
```
cardinalidad= df_viajes.Incidencias.nunique()/len(df_viajes)*100
print(cardinalidad)
```

```
14.6
```

Parece que así de primeras no se puede obtener una categórica, podríamos buscar quizás una categorización más sencilla... Pero si nos fijamos en su distribución

[6]:
```
df_viajes.Incidencias.value_counts(True)# ya podemos crear 2 categorias: con y␣
↪sin indencias por los porcentajes, y asi podemos sacar mas valor a nuestros␣
↪datos
```

[6]:
```
Incidencias
Sin incidencias                                                  0.445
Mal funcionamiento del tren de aterrizaje                        0.030
Turbulencia severa                                               0.029
Ave impacta con el avión                                         0.025
Aterrizaje de emergencia                                         0.023
                                                                 …
```

```
Problemas de navegación,                                                    0.001
Sambódromo a bordo y Conflictos entre pasajeros                             0.001
Fallo eléctrico general                                                     0.001
Fallo grave de los sistemas de comunicación,                                0.001
Problemas con el catering y Fallo de los sistemas de comunicación           0.001
Name: proportion, Length: 146, dtype: float64
```

... podemos ver que hay una clasificación sencilla,...piénsalo

[12]:
```
df_viajes["Hubo_Incidencia"] = df_viajes.Incidencias != "Sin incidencias"

df_viajes.Hubo_Incidencia.value_counts(True)
```

[12]:
```
Hubo_Incidencia
True     0.555
False    0.445
Name: proportion, dtype: float64
```

[15]:
```
# lo podemos ver como una categoria para analizar por grupos y sobre el campo␣
 ↪hubo_incidencias veremo slo svalores medios por cada compañia
df_viajes.groupby(["Aircompany"])["Hubo_Incidencia"].mean()
```

[15]:
```
Aircompany
Airnar        0.527778
FlyQ          0.531792
MoldaviAir    0.592920
PamPangea     0.500000
TabarAir      0.602620
Name: Hubo_Incidencia, dtype: float64
```

Pues eso, mejor que no tener nada de esa columna, es que podemos clasificar los viajes con o sin incidencias y a partir de ahí trabajar en ciertos análisis. Este es un criterio (simplificar categorizando) que puedes guardarte ya en tus apuntes.

[18]:
```
df_liga.sample(10)
```

[18]:

	id_partido	equipo_local	equipo_visitante	Division	\
330	214240	Valencia	Celta Vigo	1	
21	214036	Celta Vigo	Valencia	1	
542	214366	Eibar	Leganes	1	
494	214796	Sporting Gijon	Lugo	2	
197	214569	Extremadura	Deportivo La Coruna	2	
352	214694	Lugo	Elche	2	
461	214767	Deportivo La Coruna	Rayo Vallecano	2	
569	214390	Real Sociedad	Sevilla	1	
397	214725	Fuenlabrada	Alcorcon	2	
145	214542	Racing Santander	Deportivo La Coruna	2	

```
     Temporada            fecha_dt  goles_local  goles_visitante  \
330       2019 2020-02-01 21:00:00            1                0
21        2019 2019-08-24 21:00:00            1                0
542       2019 2020-07-09 19:30:00            0                0
494       2019 2020-06-28 17:00:00            2                0
197       2019 2019-11-16 16:00:00            2                0
352       2019 2020-02-09 18:15:00            2                2
461       2019 2020-06-20 17:00:00            3                3
569       2019 2020-07-16 21:00:00            0                0
397       2019 2020-03-01 18:15:00            3                4
145       2019 2019-10-26 18:00:00            1                1

                 arbitro                          estadio  odd_1  odd_x  \
330       Pablo González              Estadio de Mestalla   1.80   3.60
21   Alejandro Hernández                  Abanca-Balaídos   3.10   3.40
542       Eduardo Prieto      Estadio Municipal de Ipurúa   2.45   3.00
494          Daniel Ocón     Estadio Municipal El Molinón   1.75   3.25
197      Daniel Trujillo     Estadio Francisco de la Hera   2.70   3.00
352  Oliver De La Fuente               Estadio Anxo Carro   3.00   3.00
461        Iñaki Bikandi            Estadio Abanca-Riazor   3.10   2.90
569     Carlos Del Cerro      Estadio Municipal de Anoeta   2.37   3.30
397      Javier Iglesias          Estadio Fernando Torres   2.20   2.87
145         Rubén Ávalos  Campos de Sport de El Sardinero   2.60   3.10

     odd_2                                  Informe_Tarjetas
330   4.50  Hubo 00 tajetas rojas al equipo visitante;Hubo…
21    2.25  Hubo 0 rojas a jugadores visitantes;Hubo 0 roj…
542   3.10  Hubo 03  tarjetas amarillas de jugadores visit…
494   5.25  Hubo 2 amarillas mostradas al equipo local;Hub…
197   2.75  Hubo 2 tarjetas amarillas para el equipo visit…
352   2.50  Hubo 4 amarillas para jugadores del equipo loc…
461   2.50  Hubo 0 amarillas mostradas al equipo local;Hub…
569   2.90  Hubo 00 rojas a jugadores del equipo local;Hub…
397   3.80  Hubo 00 tajetas rojas al equipo visitante;Hubo…
145   2.80  Hubo 00 rojas a jugadores visitantes;Hubo 02 a…
```

Por ejemplo podríamos ver cual es la compañía con mayor incidencias:

[19]:
```
df_liga.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 592 entries, 0 to 591
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   id_partido        592 non-null    int64
 1   equipo_local      592 non-null    object
 2   equipo_visitante  592 non-null    object
```

```
 3   Division          592 non-null    int64
 4   Temporada         592 non-null    int64
 5   fecha_dt          592 non-null    object
 6   goles_local       592 non-null    int64
 7   goles_visitante   592 non-null    int64
 8   arbitro           592 non-null    object
 9   estadio           592 non-null    object
 10  odd_1             592 non-null    float64
 11  odd_x             592 non-null    float64
 12  odd_2             592 non-null    float64
 13  Informe_Tarjetas  592 non-null    object
dtypes: float64(3), int64(5), object(6)
memory usage: 64.9+ KB
```

Y ya tenemos un campo más bastante potente para analizar nuestro dataset.

[30]:
```
df_liga.equipo_visitante.value_counts()
```

[30]:
```
equipo_visitante
Getafe                19
Villarreal            19
Barcelona             19
Levante               19
Espanyol              19
Valencia              18
Celta Vigo            18
Osasuna               18
Atletico Madrid       17
Alaves                17
Athletic Club         17
Real Madrid           17
Sevilla               16
Almeria               16
Zaragoza              16
Mallorca              16
Valladolid            16
Real Betis            16
Eibar                 16
Real Sociedad         16
Leganes               15
Granada CF            15
Huesca                15
Cadiz                 14
Elche                 13
Alcorcon              13
Rayo Vallecano        13
Deportivo La Coruna   13
Girona                12
```

```
Numancia                12
Oviedo                  12
Fuenlabrada             12
Malaga                  11
Las Palmas              11
Sporting Gijon          11
Mirandes                10
Racing Santander        10
Lugo                    10
Ponferradina             8
Tenerife                 7
Albacete                 7
Extremadura              3
Name: count, dtype: int64
```

No hubo nulos y las columnas que son susceptibles de tener texto son ”equipo local”, ”equipo visitantes”, ”arbitro”, ”estadio”, ”fecha_dt” e ”informe tarjetas”. Te invito a que evalues (NO HAY NADA DESTABLE PARA CATEGORIZAR TODAS TIENEN VALORES POR CESCA DEL 0 Y MUY PROXIMOS ENTRE SI) el resto y juegues con el dataset, pero nossotros nos vamos a dedicar en esta clase a la de ”informe_tarjetas”

No hay nulos y las columnas que son susceptibles de contener Texto son ”equipo_local”,”equipo_visitante”,”arbitro”,”estadio”,”fecha_dt”,”Informe_Tarjetas”. Te invito a que evalúes el resto y juegues con el dataset, pero para no dedicar demasiado tiempo nos vamos a centrar en la que tiene más pinta de tener texto libre: ”Informe_Tarjetas”

cardinalidad = df_liga[]

0.1.2 Sacando provecho de las columnas/campos tipo Texto: Extracción

Pero no sólo la categorización es una forma de obtener información de un campo de tipo Texto libre, por ejemplo, veamos el dataset futbolero:

[26]:
```
df_liga.Informe_Tarjetas.value_counts(True)
```

[26]:
```
Informe_Tarjetas
Hubo 02  tarjetas amarillas de jugadores visitantes;Hubo 0 rojas a jugadores
visitantes;Hubo 0 tarjetas rojas sobre el equipo local;Hubo 1 amarillas
mostradas al equipo local                  0.003378
Hubo 01 tajetas rojas al equipo visitante;Hubo 0 rojas a jugadores del equipo
local;Hubo 2 tarjetas amarillas para el equipo visitantes;Hubo 6 amarillas para
jugadores del equipo local       0.001689
Hubo 01 rojas a jugadores del equipo local;Hubo 1 tajetas rojas al equipo
visitante;Hubo 04  tarjetas amarillas de jugadores visitantes;Hubo 3 amarillas
mostradas al equipo local             0.001689
Hubo 03 amarillas mostradas al equipo local;Hubo 0 tajetas rojas al equipo
visitante;Hubo 0 rojas a jugadores del equipo local;Hubo 05 tarjetas amarillas
para el equipo visitantes            0.001689
Hubo 02 tarjetas amarillas para el equipo visitantes;Hubo 2 amarillas para
```

```
jugadores del equipo local;Hubo 00 rojas a jugadores del equipo local;Hubo 1
rojas a jugadores visitantes           0.001689
                                      …
Hubo 2 tarjetas amarillas para el equipo visitantes;Hubo 0 rojas a jugadores del
equipo local;Hubo 00 tajetas rojas al equipo visitante;Hubo 3 amarillas
mostradas al equipo local             0.001689
Hubo 00 tarjetas rojas sobre el equipo local;Hubo 02  tarjetas amarillas de
jugadores visitantes;Hubo 02 amarillas mostradas al equipo local;Hubo 0 tajetas
rojas al equipo visitante          0.001689
Hubo 03  tarjetas amarillas de jugadores visitantes;Hubo 01 amarillas mostradas
al equipo local;Hubo 0 rojas a jugadores del equipo local;Hubo 0 rojas a
jugadores visitantes                   0.001689
Hubo 00 rojas a jugadores visitantes;Hubo 0 rojas a jugadores del equipo
local;Hubo 03  tarjetas amarillas de jugadores visitantes;Hubo 1 amarillas
mostradas al equipo local                  0.001689
Hubo 02 amarillas para jugadores del equipo local;Hubo 0 tajetas rojas al equipo
visitante;Hubo 0 tarjetas rojas sobre el equipo local;Hubo 01 tarjetas amarillas
para el equipo visitantes    0.001689
Name: proportion, Length: 591, dtype: float64
```

Como siempre, debemos investigar un poco más:

[33]:
```
cardi_T = df_liga.Informe_Tarjetas.nunique()/len(df_liga)*100
print(cardi_T)
```

```
99.83108108108108
```

Es un campo claramente complejo. Pero si observamos con cierto cuidado, podemos ver que hay información que puede ser interesante para el contexto del dataset. Están las tarjetas rojas y amarillas por equipo. ¿Cómo podríamos obtener ese valor de este campo?

[ ]:

[ ]:

... Muy bien, podríamos intentar algo con expresiones regulares, ya que existe cierto patrón (básicamente que los números están como tal y no como letra por ejemplo)

¿Cómo procederíamos? Primero hay que encontrar esos patrones. Veríamos unas cuantas filas escogidas aleatoriamente:

[36]:
```
df_liga["Informe_Tarjetas"].sample(5).to_list()
```

[36]:
```
['Hubo 00 rojas a jugadores visitantes;Hubo 02  tarjetas amarillas de jugadores
visitantes;Hubo 02 amarillas para jugadores del equipo local;Hubo 0 rojas a
jugadores del equipo local',
 'Hubo 1 amarillas mostradas al equipo local;Hubo 0 rojas a jugadores del equipo
local;Hubo 02  tarjetas amarillas de jugadores visitantes;Hubo 0 rojas a
jugadores visitantes',
 'Hubo 00 tajetas rojas al equipo visitante;Hubo 00 rojas a jugadores del equipo
```

```
local;Hubo 2 tarjetas amarillas para el equipo visitantes;Hubo 04 amarillas
mostradas al equipo local',
 'Hubo 04  tarjetas amarillas de jugadores visitantes;Hubo 0 rojas a jugadores
visitantes;Hubo 0 tarjetas rojas sobre el equipo local;Hubo 03 amarillas para
jugadores del equipo local',
 'Hubo 2  tarjetas amarillas de jugadores visitantes;Hubo 00 rojas a jugadores
visitantes;Hubo 0 tarjetas rojas sobre el equipo local;Hubo 2 amarillas para
jugadores del equipo local']
```

Esto lleva un tiempo y puede que la información a extraer no sea tan interesante como para dedicárselo, yo te lo doy hecho y te dejo como ejercicio que veas que los patrones están ahí:

1. Todos los informes están divididos en cuatro partes por ";"

2. En cada parte se informa de las tarjetas rojas o amarillas pare el equipo local o el visitante.

3. Parece que después del número vienen el tipo de tarjeta (amarilla, roja), a veces acompañado de otras palabras

4. Después del tipo de tarjeta aparece el equipo o mención al mismo (visitante, visitantes, local, locales)

Vamos a intentar sacar las rojas de los visitantes (y tendrás como ejercicio hacerlo para el resto en los ejercicios de este grupo :-):

[ ]:
```
```

Tomamos dos ejemplos:

[48]:
```
ejemplo_1 = "Hubo 00 tajetas rojas al equipo visitante" # siempre empieza por␣
 ↪hubo+sp+numeros+ a vece espacion a veces carecteres(.*)+rojas+sp+ otro␣
 ↪conjunto de caracteres
#([s]? a vces si aparece a veces no) y agrupamos lo que queremos quedarnos␣
 ↪entre ().
ejemplo_2 = "Hubo 01 rojas a jugadores visitantes"
```

[49]:
```
import re

patron = "Hubo ([0-9]+) .*rojas .* visitante[s]?"#cada grupo se cuenta por␣
 ↪orden que los ponga, 1, 2,3....)

print(re.match(patron, ejemplo_1))

print(re.match(patron, ejemplo_1).group(1))
```

```
<re.Match object; span=(0, 41), match='Hubo 00 tajetas rojas al equipo
visitante'>
00
```

[50]:
```
print(re.match(patron, ejemplo_2))

print(re.match(patron, ejemplo_2).group(1))
```

```
<re.Match object; span=(0, 36), match='Hubo 01 rojas a jugadores visitantes'>
01
```

[51]:
```
num_tarjetas = int(re.match(patron, ejemplo_2).group(1))# pasarlo a una␣
 ↪variable y convertirlo a int
print(num_tarjetas)
```

```
1
```

[ ]:
```

```

Lo tenemos... ¿y ahora cómo lo aplicamos al dataframe? Pues sí podríamos hacer una función, pero esta vez te voy a ahorrar un poco de tiempo... Pandas tiene métodos para aplicar expresiones regulares:

[52]:
```
df_liga.Informe_Tarjetas.str.match(patron)# nos develeve donde se da ese patron
```

[52]:
```
0      True
1      True
2      True
3      True
4      True
       …
587    True
588    True
589    True
590    True
591    True
Name: Informe_Tarjetas, Length: 592, dtype: bool
```

[53]:
```
df_liga.Informe_Tarjetas.str.extract(patron)# aqui ya me da lo que he puesto␣
 ↪entre parentesis
```

[53]:
```
      0
0    01
1    03
2     4
3    00
4    01
..   ..
587   0
588  00
589   2
590   2
591  02
```

```
[592 rows x 1 columns]
```

[63]:
```
df_liga["Tarjetas_Rojas_Visitante"] = df_liga.Informe_Tarjetas.str.
↪extract(patron).astype("int")# creamos una catehria nueva con la informacion␣
↪extrada y con asstype ña comvertimos a enterp
```

[64]:
```
df_liga.Tarjetas_Rojas_Visitante.value_counts(True)
```

[64]:
```
Tarjetas_Rojas_Visitante
0    0.432432
1    0.141892
2    0.131757
3    0.119932
4    0.109797
5    0.038851
6    0.021959
7    0.003378
Name: proportion, dtype: float64
```

Aquí te dejo otros métodos que sirven para aplicar expresiones regulares a valores string. Ojo recuerda que tienes que poner ".str." antes de invocarlos:

Method	Description
`match()`	Call `re.match()` on each element, returning a boolean.
`extract()`	Call `re.match()` on each element, returning matched groups as strings.
`findall()`	Call `re.findall()` on each element
`replace()`	Replace occurrences of pattern with some other string
`contains()`	Call `re.search()` on each element, returning a boolean
`count()`	Count occurrences of pattern

Method	Description
`split()`	Equivalent to `str.split()`, but accepts regexps
`rsplit()`	Equivalent to `str.rsplit()`, but accepts regexps

[ ]:

[ ]:

07_Transformacion_Fechas

November 29, 2023

ETL Y DATOS

0.1 ETL: Tratamiento de Fechas

Los valores de tipo Fecha son uno de esos tipos "difíciles" pero útiles cuando puedes manejarlos con cierta soltura y sin complejas funciones. Es decir cuando hay una librería o módulo que te lo solucione. En caso de Python hay, y nosotros vamos a usar en concreto Datetime. Así sin más carga datos y esta nueva librería y veamos como hacer un primer tratamiento (simple) de los valores tipo fecha.

[2]:
```
import datetime as dt # Este alias es mío, no es como pd o np
import pandas as pd


df_liga  = pd.read_csv("./data/df_liga_2019.csv")
```

0.1.1 El tipo datetime, in two kicks

Para empezar echemos otro vistazo a nuestro dataset futbolero:

[3]:
```
df_liga
```

[3]:
```
     id_partido         equipo_local equipo_visitante  Division  Temporada  \
0        214023           Celta Vigo      Real Madrid         1       2019
1        214403     Racing Santander           Malaga         2       2019
2        214024              Valencia    Real Sociedad         1       2019
3        214404               Almeria         Albacete         2       2019
4        214026            Villarreal       Granada CF         1       2019
..          …                     …              …       …        …
```

```
587      214853             Alcorcon           Girona         2       2019
588      214863             Zaragoza     Ponferradina         2       2019
589      214854              Almeria           Malaga         2       2019
590      214862       Sporting Gijon           Huesca         2       2019
591      214856  Deportivo La Coruna      Fuenlabrada         2       2019

                fecha_dt  goles_local  goles_visitante             arbitro  \
0    2019-08-17 17:00:00            1                3      Javier Estrada
1    2019-08-17 18:00:00            0                1    Aitor Gorostegui
2    2019-08-17 19:00:00            1                1           Jesús Gil
3    2019-08-17 19:00:00            3                0            Saúl Ais
4    2019-08-17 21:00:00            4                4      Adrián Cordero
..                   …          …              …                   …
587  2020-07-20 21:00:00            2                0         Juan Pulido
588  2020-07-20 21:00:00            2                1    Dámaso Arcediano
589  2020-07-20 21:00:00            0                0            Saúl Ais
590  2020-07-20 21:00:00            0                1        Gorka Sagues
591  2020-08-07 20:00:00            2                1  Isidro Díaz de Mera

                                 estadio  odd_1  odd_x  odd_2  \
0                        Abanca-Balaídos   4.75   4.20   1.65
1          Campos de Sport de El Sardinero   2.87   3.10   2.55
2                     Estadio de Mestalla   1.66   3.75   5.50
3    Estadio de los Juegos Mediterráneos   2.37   3.10   3.10
4                  Estadio de la Cerámica   1.60   3.80   6.50
..                                    …    …    …    …
587                Estadio Santo Domingo   2.37   2.87   3.40
588                Estadio de la Romareda   2.10   3.30   3.50
589  Estadio de los Juegos Mediterráneos   2.10   3.20   3.60
590         Estadio Municipal El Molinón   3.30   3.10   2.15
591                 Estadio Abanca-Riazor   2.10   3.20   3.60

                                       Informe_Tarjetas
0    Hubo 01 tajetas rojas al equipo visitante;Hubo…
1    Hubo 03 amarillas mostradas al equipo local;Hu…
2    Hubo 4 amarillas mostradas al equipo local;Hub…
3    Hubo 00 rojas a jugadores visitantes;Hubo 01 a…
4    Hubo 01  tarjetas amarillas de jugadores visit…
..                                                 …
587  Hubo 0 tajetas rojas al equipo visitante;Hubo …
588  Hubo 00  tarjetas amarillas de jugadores visit…
589  Hubo 2  tarjetas amarillas de jugadores visita…
590  Hubo 2 amarillas para jugadores del equipo loc…
591  Hubo 02 amarillas para jugadores del equipo lo…

[592 rows x 14 columns]
```

Hay un campo, "fecha_dt", que claramente es una fecha con hora. ¿Pero qué tipo tiene?

[4]:
```
df_liga.fecha_dt.dtypes
```

[4]:
```
dtype('O')
```

Es un string, y por lo tanto si lo queremos manipular tal y como filtrar los partidos de Agosto, o los que empiezan a las 20:00 horas, tendremos que hacer funciones y eso es lo que queremos evitarnos. Pues es posible si logramos convertirlo a un tipo de Python denominado Datetime que Pandas maneja bastante bien.

Y qué pinta tiene Datetime... Creemos uno:

[5]:
```
fecha = dt.datetime(year = 2023, month = 11, day = 3, hour = 20)
print(fecha)
```

```
2023-11-03 20:00:00
```

¿Y ahora qué, Jaime? ¿Qué hemos ganado?

Pues que operar con ellos es bastante sencillo:

[6]:
```
# Crear fechas sumando periodos de tiempo
fecha2 = fecha + dt.timedelta(hours = 20)
print(fecha2)

fecha3 = fecha - dt.timedelta(days = 20)
print(fecha3)
```

```
2023-11-04 16:00:00
2023-10-14 20:00:00
```

[8]:
```
# Comparar fechas
fecha3 < fecha < fecha2
```

[8]:
```
True
```

[7]:
```
# Encontrar diferencias de tiempo
diferencia = fecha2 - fecha3
print(diferencia)
print(type(diferencia))
print(diferencia.seconds)
```

```
20 days, 20:00:00
<class 'datetime.timedelta'>
72000
```

Es decir es bastante útil para manejo directo. La cuestión ahora es como pasar de string a datetime y viceversa...

0.1.2 Datetime <-> String

Tenemos dos métodos y ambos hacen uso de un patrón de conversión cuyas convenciones puedes encontrar aquí para strftime y aquí para ambos: * strftime para convertir de `datetime` a `string`

* strptime para convertir de `string` a `datetime`

[9]:
```
# Strftime, ejemplo
fecha.strftime("Hoy es %d de %m de %Y, es %a, y son las %H y %M minutos")#␣
 ↪poner mayusculas o misniculas despues e los % es formato 1 solo numero o␣
 ↪maYUS FORMATO 2 NUMEROS
# Puedes usar %d,%m, etc como quieras
```

[9]:
```
'Hoy es 03 de 11 de 2023, es Fri, y son las 20 y 00 minutos'
```

[10]:
```
# Strptime, ejemplo
cadena_con_fecha = 'Hoy es 03 de 11 de 2023, es Fri, y son las 20 y 00 minutos'
patron = "Hoy es %d de %m de %Y, es %a, y son las %H y %M minutos"
fecha_de_string = dt.datetime.strptime(cadena_con_fecha,patron)
print(fecha_de_string)
```

```
2023-11-03 20:00:00
```

0.1.3 Pandas y Datetime

Ahora si queremos convertir nuestro campo ”fecha_dt” a datetime, solo tendríamos que hacernos una función que usaser strptime.... Vale, vale, existe un método para hacerlo, siempre que sepamos el patrón o formato como en el ejemplo anterior:

[11]:
```
df_liga.fecha_dt.sample(1)
```

[11]:
```
359    2020-02-15 18:30:00
Name: fecha_dt, dtype: object
```

[12]:
```
patron = "%Y-%m-%d %H:%M:%S"
```

[13]:
```
df_liga["FECHA"] = pd.to_datetime(df_liga.fecha_dt, format = patron)
```

[14]:
```
df_liga.FECHA
```

[14]:
```
0     2019-08-17 17:00:00
1     2019-08-17 18:00:00
2     2019-08-17 19:00:00
3     2019-08-17 19:00:00
4     2019-08-17 21:00:00
              …
587   2020-07-20 21:00:00
588   2020-07-20 21:00:00
589   2020-07-20 21:00:00
590   2020-07-20 21:00:00
591   2020-08-07 20:00:00
Name: FECHA, Length: 592, dtype: datetime64[ns]
```

Y ya podemos sacarle provecho:

[15]:
```
# Partidos de liga de la temporada 19-20 que se jugaron en 2020
df_liga[df_liga.FECHA > "2020"]

# Partidos de la liga de la temporada que se jugaron en Domingo (Lunes -> 0,␣
 ↪Domingo -> 6):
df_liga[df_liga.FECHA.dt.day_of_week == 6]

# Partidos en diciembre de 2019
df_liga[df_liga.FECHA.dt.month == 12]
```

[15]:

	id_partido	equipo_local	equipo_visitante	Division	\
227	214170	Sevilla	Leganes	1	
228	214592	Fuenlabrada	Cadiz	2	
229	214163	Athletic Club	Granada CF	1	
230	214166	Espanyol	Osasuna	1	
231	214597	Oviedo	Rayo Vallecano	2	
232	214593	Lugo	Deportivo La Coruna	2	
233	214590	Elche	Racing Santander	2	
234	214167	Getafe	Levante	1	
235	214591	Extremadura	Las Palmas	2	
236	214164	Atletico Madrid	Barcelona	1	
237	214179	Real Madrid	Espanyol	1	
238	214606	Huesca	Rayo Vallecano	2	
239	214175	Granada CF	Alaves	1	
240	214610	Sporting Gijon	Ponferradina	2	
241	214177	Levante	Valencia	1	
242	214172	Barcelona	Mallorca	1	
243	214605	Girona	Lugo	2	
244	214174	Eibar	Getafe	1	
245	214173	Real Betis	Athletic Club	1	
246	214181	Valladolid	Real Sociedad	1	
247	214604	Deportivo La Coruna	Zaragoza	2	
248	214607	Las Palmas	Numancia	2	
249	214176	Leganes	Celta Vigo	1	
250	214603	Cadiz	Elche	2	
251	214178	Osasuna	Sevilla	1	
252	214188	Granada CF	Levante	1	
253	214615	Mirandes	Huesca	2	
254	214189	Real Sociedad	Barcelona	1	
255	214617	Ponferradina	Deportivo La Coruna	2	
256	214183	Athletic Club	Eibar	1	
257	214621	Zaragoza	Racing Santander	2	
258	214184	Atletico Madrid	Osasuna	1	
259	214612	Extremadura	Malaga	2	
260	214187	Getafe	Valladolid	1	
261	214185	Celta Vigo	Mallorca	1	
262	214186	Espanyol	Real Betis	1	

263	214611		Elche	Las Palmas	2
264	214190		Sevilla	Villarreal	1
265	214191		Valencia	Real Madrid	1
266	214197		Mallorca	Sevilla	1
267	214192		Barcelona	Alaves	1
268	214629		Las Palmas	Rayo Vallecano	2
269	214200		Villarreal	Getafe	1
270	214201		Valladolid	Valencia	1
271	214625		Cadiz	Numancia	2
272	214624		Almeria	Ponferradina	2
273	214195		Leganes	Espanyol	1
274	214198		Osasuna	Real Sociedad	1
275	214628		Huesca	Zaragoza	2
276	214193		Real Betis	Atletico Madrid	1
277	214623		Alcorcon	Fuenlabrada	2
278	214196		Levante	Celta Vigo	1
279	214627		Girona	Mirandes	2
280	214199		Real Madrid	Athletic Club	1

	Temporada	fecha_dt	goles_local	goles_visitante	\
227	2019	2019-12-01 12:00:00	1	0	
228	2019	2019-12-01 12:00:00	1	0	
229	2019	2019-12-01 14:00:00	2	0	
230	2019	2019-12-01 16:00:00	2	4	
231	2019	2019-12-01 16:00:00	2	1	
232	2019	2019-12-01 18:00:00	0	0	
233	2019	2019-12-01 18:00:00	2	0	
234	2019	2019-12-01 18:30:00	4	0	
235	2019	2019-12-01 20:00:00	0	1	
236	2019	2019-12-01 21:00:00	0	1	
237	2019	2019-12-07 13:00:00	2	0	
238	2019	2019-12-07 16:00:00	0	2	
239	2019	2019-12-07 16:00:00	3	0	
240	2019	2019-12-07 18:00:00	1	0	
241	2019	2019-12-07 18:30:00	2	4	
242	2019	2019-12-07 21:00:00	5	2	
243	2019	2019-12-08 12:00:00	3	1	
244	2019	2019-12-08 12:00:00	0	1	
245	2019	2019-12-08 14:00:00	3	2	
246	2019	2019-12-08 16:00:00	0	0	
247	2019	2019-12-08 16:00:00	1	3	
248	2019	2019-12-08 18:00:00	3	1	
249	2019	2019-12-08 18:30:00	3	2	
250	2019	2019-12-08 21:00:00	0	0	
251	2019	2019-12-08 21:00:00	1	1	
252	2019	2019-12-14 13:00:00	1	2	
253	2019	2019-12-14 16:00:00	2	0	

254	2019	2019-12-14 16:00:00	2	2
255	2019	2019-12-14 18:00:00	2	0
256	2019	2019-12-14 18:30:00	0	0
257	2019	2019-12-14 21:00:00	2	0
258	2019	2019-12-14 21:00:00	2	0
259	2019	2019-12-15 12:00:00	0	0
260	2019	2019-12-15 12:00:00	2	0
261	2019	2019-12-15 14:00:00	2	2
262	2019	2019-12-15 16:00:00	2	2
263	2019	2019-12-15 18:00:00	2	3
264	2019	2019-12-15 18:30:00	1	2
265	2019	2019-12-15 21:00:00	1	1
266	2019	2019-12-21 13:00:00	0	2
267	2019	2019-12-21 16:00:00	4	1
268	2019	2019-12-21 18:00:00	1	1
269	2019	2019-12-21 18:30:00	1	0
270	2019	2019-12-21 21:00:00	1	1
271	2019	2019-12-21 21:00:00	2	4
272	2019	2019-12-22 12:00:00	2	3
273	2019	2019-12-22 12:00:00	2	0
274	2019	2019-12-22 14:00:00	3	4
275	2019	2019-12-22 16:00:00	2	1
276	2019	2019-12-22 16:00:00	1	2
277	2019	2019-12-22 18:00:00	1	1
278	2019	2019-12-22 18:30:00	3	1
279	2019	2019-12-22 20:00:00	0	3
280	2019	2019-12-22 21:00:00	0	0

	arbitro	estadio	odd_1	odd_x	\
227	Ricardo De Burgos	Estadio Ramón Sánchez Pizjuán	1.57	4.00	
228	Saúl Ais	Estadio Fernando Torres	2.50	3.00	
229	Adrián Cordero	San Mamés Barria	1.70	3.40	
230	Mario Melero	RCDE Stadium	2.15	3.10	
231	Juan Pulido	Estadio Nuevo Carlos Tartiere	2.70	3.10	
232	José López	Estadio Anxo Carro	3.10	3.10	
233	Jorge Figueroa	Estadio Manuel Martínez Valero	2.05	3.10	
234	Javier Estrada	Coliseum Alfonso Pérez	1.66	3.80	
235	Gorka Sagues	Estadio Francisco de la Hera	2.45	3.00	
236	Antonio Mateu	Estadio Wanda Metropolitano	2.75	3.30	
237	Santiago Jaime	Estadio Santiago Bernabéu	1.20	7.00	
238	Daniel Ocón	Estadio El Alcoraz	1.85	3.40	
239	Jesús Gil	Estadio Nuevo Los Cármenes	2.20	3.10	
240	Isidro Díaz de Mera	Estadio Municipal El Molinón	1.83	3.25	
241	David Medié	Estadio Ciudad de Valencia	3.20	3.75	
242	José Munuera	Camp Nou	1.14	8.50	
243	Santiago Varón	Estadi Municipal de Montilivi	1.45	4.00	
244	Eduardo Prieto	Estadio Municipal de Ipurúa	2.80	2.90	

245	Valentín Pizarro	Estadio Benito Villamarín	2.30	3.20	
246	Pablo González	Estadio Municipal José Zorrilla	3.20	3.20	
247	Iosu Galech	Estadio Abanca-Riazor	2.37	3.25	
248	Álvaro Moreno	Estadio de Gran Canaria	2.10	3.20	
249	José Sánchez	Estadio Municipal de Butarque	2.37	3.10	
250	Daniel Trujillo	Estadio Ramón de Carranza	1.65	3.40	
251	Javier Estrada	Estadio El Sadar	3.40	3.30	
252	Carlos Del Cerro	Estadio Nuevo Los Cármenes	2.00	3.60	
253	Gorka Sagues	Estadio Municipal de Anduva	2.75	2.80	
254	Javier Alberola	Reale Arena	5.25	4.20	
255	Miguel Ortiz	Estadio El Toralín	2.10	2.80	
256	Mario Melero	San Mamés Barria	1.66	3.75	
257	Juan Pulido	Estadio de la Romareda	1.60	3.50	
258	José Munuera	Estadio Wanda Metropolitano	1.40	4.33	
259	Santiago Varón	Estadio Francisco de la Hera	2.70	2.70	
260	Antonio Mateu	Coliseum Alfonso Pérez	1.66	3.60	
261	Ricardo De Burgos	Abanca-Balaídos	1.70	3.80	
262	Guillermo Cuadra	RCDE Stadium	2.60	3.25	
263	Luis Milla	Estadio Manuel Martínez Valero	2.40	2.80	
264	Adrián Cordero	Estadio Ramón Sánchez Pizjuán	1.66	4.00	
265	José Sánchez	Estadio de Mestalla	3.80	3.80	
266	Jesús Gil	Iberostar Estadi	5.00	4.00	
267	Mario Melero	Camp Nou	1.12	9.00	
268	Saúl Ais	Estadio de Gran Canaria	2.37	3.00	
269	Santiago Jaime	Estadio de la Cerámica	2.20	3.25	
270	Valentín Pizarro	Estadio Municipal José Zorrilla	3.50	3.40	
271	Rubén Ávalos	Estadio Ramón de Carranza	1.72	3.10	
272	Santiago Varón	Estadio de los Juegos Mediterráneos	1.75	3.40	
273	Javier Alberola	Estadio Municipal de Butarque	2.10	3.10	
274	David Medié	Estadio El Sadar	2.80	3.40	
275	Isidro Díaz de Mera	Estadio El Alcoraz	1.80	3.40	
276	Javier Estrada	Estadio Benito Villamarín	3.80	3.30	
277	Víctor Areces	Estadio Santo Domingo	2.37	2.90	
278	Eduardo Prieto	Estadio Ciudad de Valencia	2.62	3.40	
279	Jon González	Estadi Municipal de Montilivi	1.65	3.60	
280	Adrián Cordero	Estadio Santiago Bernabéu	1.33	5.25	

	odd_2	Informe_Tarjetas	\
227	6.00	Hubo 00 rojas a jugadores del equipo local;Hub…	
228	3.00	Hubo 5 amarillas para jugadores del equipo loc…	
229	6.00	Hubo 00 tarjetas rojas sobre el equipo local;H…	
230	3.60	Hubo 0 tarjetas rojas sobre el equipo local;Hu…	
231	2.70	Hubo 0 tarjetas rojas sobre el equipo local;Hu…	
232	2.40	Hubo 02 amarillas mostradas al equipo local;Hu…	
233	4.00	Hubo 0 rojas a jugadores visitantes;Hubo 00 ro…	
234	5.50	Hubo 00 rojas a jugadores del equipo local;Hub…	
235	3.10	Hubo 01 rojas a jugadores visitantes;Hubo 07 t…	

236	2.60	Hubo 4 tarjetas amarillas de jugadores visita…
237	13.00	Hubo 03 tarjetas amarillas para el equipo visi…
238	4.50	Hubo 00 tarjetas rojas sobre el equipo local;H…
239	3.50	Hubo 0 tarjetas rojas sobre el equipo local;Hu…
240	4.75	Hubo 1 rojas a jugadores visitantes;Hubo 04 ta…
241	2.15	Hubo 0 rojas a jugadores visitantes;Hubo 1 roj…
242	17.00	Hubo 03 amarillas para jugadores del equipo lo…
243	7.50	Hubo 1 rojas a jugadores visitantes;Hubo 0 tar…
244	2.87	Hubo 00 rojas a jugadores visitantes;Hubo 01 r…
245	3.30	Hubo 0 tajetas rojas al equipo visitante;Hubo …
246	2.37	Hubo 0 amarillas mostradas al equipo local;Hub…
247	3.00	Hubo 1 amarillas mostradas al equipo local;Hub…
248	3.60	Hubo 0 rojas a jugadores visitantes;Hubo 04 ta…
249	3.30	Hubo 00 tarjetas rojas sobre el equipo local;H…
250	6.00	Hubo 00 tarjetas rojas sobre el equipo local;H…
251	2.20	Hubo 01 rojas a jugadores del equipo local;Hub…
252	3.80	Hubo 4 amarillas para jugadores del equipo loc…
253	2.62	Hubo 0 rojas a jugadores visitantes;Hubo 3 ama…
254	1.61	Hubo 0 rojas a jugadores del equipo local;Hubo…
255	3.80	Hubo 00 tarjetas rojas sobre el equipo local;H…
256	5.50	Hubo 00 tajetas rojas al equipo visitante;Hubo…
257	5.50	Hubo 0 tarjetas rojas sobre el equipo local;Hu…
258	10.00	Hubo 01 amarillas mostradas al equipo local;Hu…
259	2.87	Hubo 0 rojas a jugadores del equipo local;Hubo…
260	6.00	Hubo 03 amarillas para jugadores del equipo lo…
261	5.00	Hubo 00 rojas a jugadores del equipo local;Hub…
262	2.80	Hubo 0 rojas a jugadores visitantes;Hubo 01 t…
263	3.10	Hubo 03 amarillas mostradas al equipo local;Hu…
264	5.00	Hubo 00 rojas a jugadores visitantes;Hubo 00 t…
265	1.90	Hubo 2 tarjetas amarillas para el equipo visit…
266	1.66	Hubo 03 tarjetas amarillas para el equipo visi…
267	21.00	Hubo 4 tarjetas amarillas de jugadores visita…
268	3.25	Hubo 03 amarillas para jugadores del equipo lo…
269	3.50	Hubo 01 tajetas rojas al equipo visitante;Hubo…
270	2.15	Hubo 0 rojas a jugadores visitantes;Hubo 03 ta…
271	6.00	Hubo 1 tarjetas rojas sobre el equipo local;Hu…
272	5.00	Hubo 00 tarjetas rojas sobre el equipo local;H…
273	3.90	Hubo 6 tarjetas amarillas para el equipo visit…
274	2.50	Hubo 0 tajetas rojas al equipo visitante;Hubo …
275	4.50	Hubo 2 tajetas rojas al equipo visitante;Hubo …
276	2.10	Hubo 00 rojas a jugadores del equipo local;Hub…
277	3.30	Hubo 00 tarjetas rojas sobre el equipo local;H…
278	2.62	Hubo 0 rojas a jugadores del equipo local;Hubo…
279	5.75	Hubo 00 rojas a jugadores visitantes;Hubo 01 a…
280	8.00	Hubo 01 amarillas mostradas al equipo local;Hu…

FECHA

```
227 2019-12-01 12:00:00
228 2019-12-01 12:00:00
229 2019-12-01 14:00:00
230 2019-12-01 16:00:00
231 2019-12-01 16:00:00
232 2019-12-01 18:00:00
233 2019-12-01 18:00:00
234 2019-12-01 18:30:00
235 2019-12-01 20:00:00
236 2019-12-01 21:00:00
237 2019-12-07 13:00:00
238 2019-12-07 16:00:00
239 2019-12-07 16:00:00
240 2019-12-07 18:00:00
241 2019-12-07 18:30:00
242 2019-12-07 21:00:00
243 2019-12-08 12:00:00
244 2019-12-08 12:00:00
245 2019-12-08 14:00:00
246 2019-12-08 16:00:00
247 2019-12-08 16:00:00
248 2019-12-08 18:00:00
249 2019-12-08 18:30:00
250 2019-12-08 21:00:00
251 2019-12-08 21:00:00
252 2019-12-14 13:00:00
253 2019-12-14 16:00:00
254 2019-12-14 16:00:00
255 2019-12-14 18:00:00
256 2019-12-14 18:30:00
257 2019-12-14 21:00:00
258 2019-12-14 21:00:00
259 2019-12-15 12:00:00
260 2019-12-15 12:00:00
261 2019-12-15 14:00:00
262 2019-12-15 16:00:00
263 2019-12-15 18:00:00
264 2019-12-15 18:30:00
265 2019-12-15 21:00:00
266 2019-12-21 13:00:00
267 2019-12-21 16:00:00
268 2019-12-21 18:00:00
269 2019-12-21 18:30:00
270 2019-12-21 21:00:00
271 2019-12-21 21:00:00
272 2019-12-22 12:00:00
273 2019-12-22 12:00:00
```

```
274 2019-12-22 14:00:00
275 2019-12-22 16:00:00
276 2019-12-22 16:00:00
277 2019-12-22 18:00:00
278 2019-12-22 18:30:00
279 2019-12-22 20:00:00
280 2019-12-22 21:00:00
```

Siempre que tengas campos con pinta de fechas en tus datasets conviertelos a `datetime`.

08_Transformacion_Numeros

November 29, 2023

ETL Y DATOS

0.1 ETL: Tratamiento de Numeros

El tratamiento real de los campos numéricos llega con el análisis y re-análisis, pero antes además de hacer limpieza de los campos numéricos podemos, al igual que hicimos con los campos de tipo texto, obtener algunas columnas adicionales bastante directas previas a entrar en el ciclo de análisis-re-análisis. A eso vamos a dedicar la sesión.

Para empezar, cargamos nuestro dataset de partidos de 2019.

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

df_liga  = pd.read_csv("./data/df_liga_2019.csv")
```

0.1.1 Transformaciones directas

Para empezar echemos otro vistazo a nuestro dataset futbolero:

[2]:
```
df_liga
```

[2]:

	id_partido	equipo_local	equipo_visitante	Division	Temporada	\
0	214023	Celta Vigo	Real Madrid	1	2019	
1	214403	Racing Santander	Malaga	2	2019	
2	214024	Valencia	Real Sociedad	1	2019	
3	214404	Almeria	Albacete	2	2019	
4	214026	Villarreal	Granada CF	1	2019	
..	…	…	…	…	…	
587	214853	Alcorcon	Girona	2	2019	

```
588      214863             Zaragoza     Ponferradina         2       2019
589      214854              Almeria           Malaga         2       2019
590      214862       Sporting Gijon           Huesca         2       2019
591      214856  Deportivo La Coruna      Fuenlabrada         2       2019

                fecha_dt  goles_local  goles_visitante             arbitro  \
0    2019-08-17 17:00:00            1                3       Javier Estrada
1    2019-08-17 18:00:00            0                1     Aitor Gorostegui
2    2019-08-17 19:00:00            1                1           Jesús Gil
3    2019-08-17 19:00:00            3                0            Saúl Ais
4    2019-08-17 21:00:00            4                4       Adrián Cordero
..                   …          …              …                  …
587  2020-07-20 21:00:00            2                0          Juan Pulido
588  2020-07-20 21:00:00            2                1     Dámaso Arcediano
589  2020-07-20 21:00:00            0                0            Saúl Ais
590  2020-07-20 21:00:00            0                1         Gorka Sagues
591  2020-08-07 20:00:00            2                1  Isidro Díaz de Mera

                              estadio  odd_1  odd_x  odd_2  \
0                       Abanca-Balaídos   4.75   4.20   1.65
1        Campos de Sport de El Sardinero   2.87   3.10   2.55
2                     Estadio de Mestalla   1.66   3.75   5.50
3    Estadio de los Juegos Mediterráneos   2.37   3.10   3.10
4                  Estadio de la Cerámica   1.60   3.80   6.50
..                                    …    …    …    …
587                Estadio Santo Domingo   2.37   2.87   3.40
588                Estadio de la Romareda   2.10   3.30   3.50
589  Estadio de los Juegos Mediterráneos   2.10   3.20   3.60
590         Estadio Municipal El Molinón   3.30   3.10   2.15
591                 Estadio Abanca-Riazor   2.10   3.20   3.60

                                    Informe_Tarjetas
0    Hubo 01 tajetas rojas al equipo visitante;Hubo…
1    Hubo 03 amarillas mostradas al equipo local;Hu…
2    Hubo 4 amarillas mostradas al equipo local;Hub…
3    Hubo 00 rojas a jugadores visitantes;Hubo 01 a…
4    Hubo 01  tarjetas amarillas de jugadores visit…
..                                                 …
587  Hubo 0 tajetas rojas al equipo visitante;Hubo …
588  Hubo 00  tarjetas amarillas de jugadores visit…
589  Hubo 2  tarjetas amarillas de jugadores visita…
590  Hubo 2 amarillas para jugadores del equipo loc…
591  Hubo 02 amarillas para jugadores del equipo lo…

[592 rows x 14 columns]
```

Fíjate que no está computado el resultado. Cuando te encuentres dataframes donde faltan campos evidentes y que se pueden obtener de los ya existentes, no dudes, créalos en vez de estar haciéndolo

cada vez que lo necesites. La única excepción es para aquellos campos que dependan de variables externas que se actualicen en medio del programa (de la ejecución del mismo)

[5]:
```
def obten_resultado(row):
    goles_local = row["goles_local"]
    goles_visitante = row["goles_visitante"]
    if goles_local > goles_visitante:
        return "1"
    elif goles_local < goles_visitante:
        return"2"
    else:
        return"X"
```

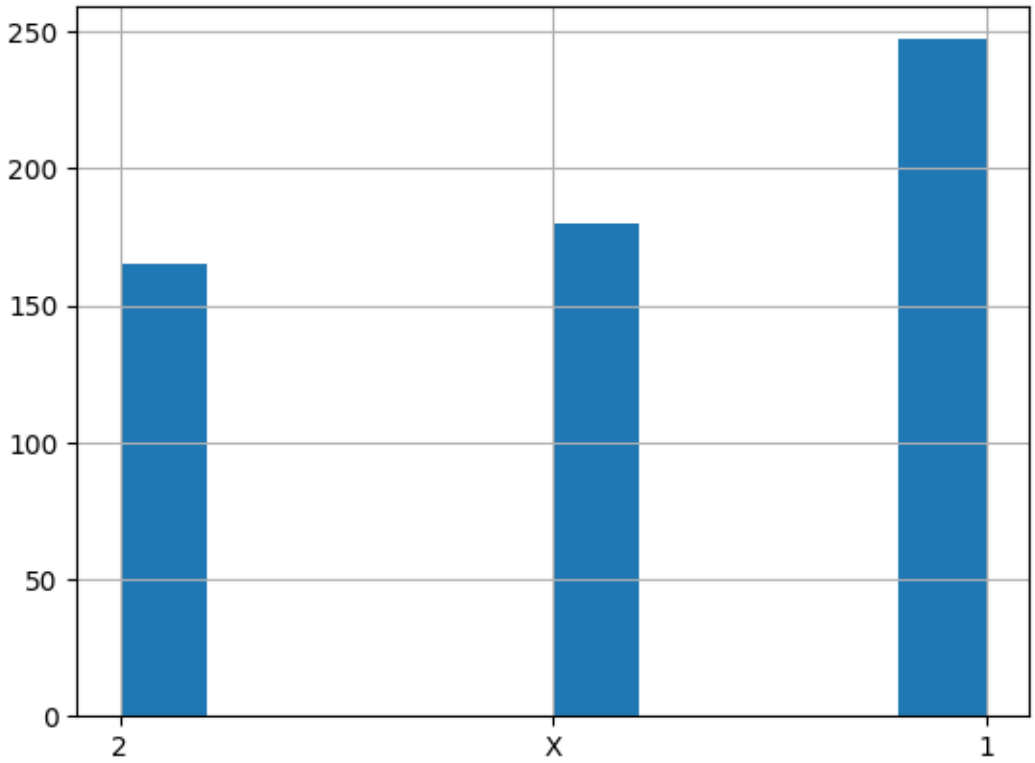
[6]:
```
df_liga["resultado"]=df_liga[["goles_local", "goles_visitante"]].
↪apply(obten_resultado, axis=1)
```

[7]:
```
df_liga.resultado.value_counts(normalize = True)
```

[7]:
```
resultado
1    0.417230
X    0.304054
2    0.278716
Name: proportion, dtype: float64
```

[8]:
```
df_liga.resultado.hist()
```

[8]:
```
<Axes: >
```



Un detalle, no destruyas de primeras las columnas o campos que te han servido para hacer las transformaciones directas, no sabes si al profundizar en el análisis te pueden servir para crear otros campos interesantes.

0.1.2 Binning

Para terminar otra de las posibles transformaciones que podemos hacer es convertir determinados campos con una gran dispersión de valores en grupos o rangos, también llamados ”bins” y luego tratar esa nueva columna como una categórica (algo particular porque existe una relación de orden matemático entre los bins, por ejemplo estar en un bin con un índice mayor que otro significa tener más o menos valor, dependiendo la forma en que hayamos hecho el bin)

Utilicemos, como ejemplo, el campo ”Asistencia_miles” de nuestro dataframe.

[ ]:
```
df_liga["Asistencia_miles"].value_counts()
```

[ ]:
```
df_liga.Asistencia_miles.hist()
```

Puede ser interesante hacer cortes, 30.000 y 60.000 para buscar 3 binnes bien distribuidos (los puntos de corte y el número de binnes dependerán de los datos y del contexto del estudio. Elegirlos bien es cuestión de experiencia). Un criterio posible es que los binnes estén equilibrados en cuanto a número de partidos (en este caso) que haya en cada uno, pero siempre depende...

¿Y cómo se hacen bins en pandas?:

[ ]:
```
# Indicándole los límites:
bins = [0,30000,60000, df_liga.Asistencia_miles,max()]

df_liga["Categoria_Asistencia"] = pd.cut(df_liga.Asistencia_miles, bins, labels␣
↪=["baja", "media", "alta"])
```

[ ]:
```
df_liga["Categoria_Asistencia"].value_counts(normalize= True)
```

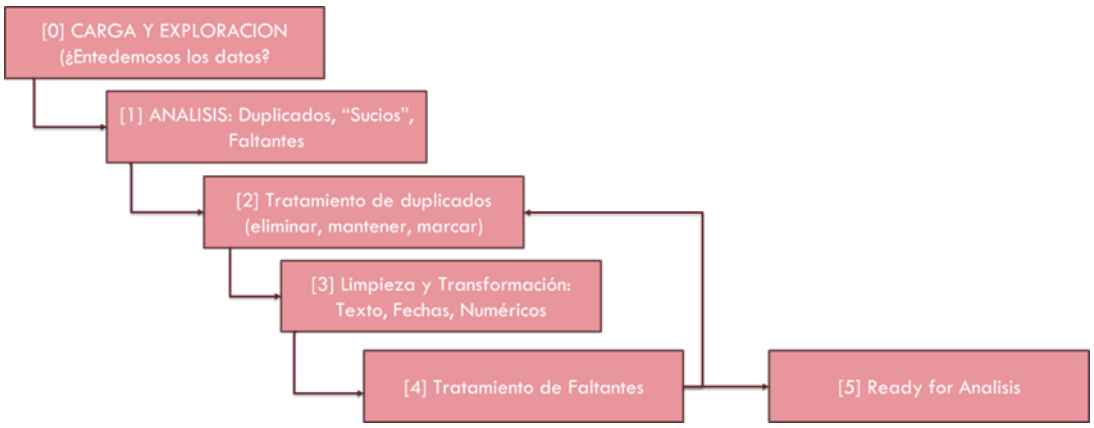
[ ]:
```

```

[ ]:
```
# Bines de ancho equivalente
bins = 3
pd.cut(df_liga.Asistencia_miles, bins, labels =["baja", "media", "alta"]).
↪values_counts()
```

NOTA IMPORTANTE: El tratamiento que hemos comentado es un tratamiento previo al análisis, sencillo y para nada constituye el verdadero corazón del EDA que es precisamente el análisis numérico y tampoco al ánalisis y tratamiento de features que haremos antes de trabajar los modelos de Machine Learning y, en menor medida, de Deep Learning.

09_Tratamiento_Missings

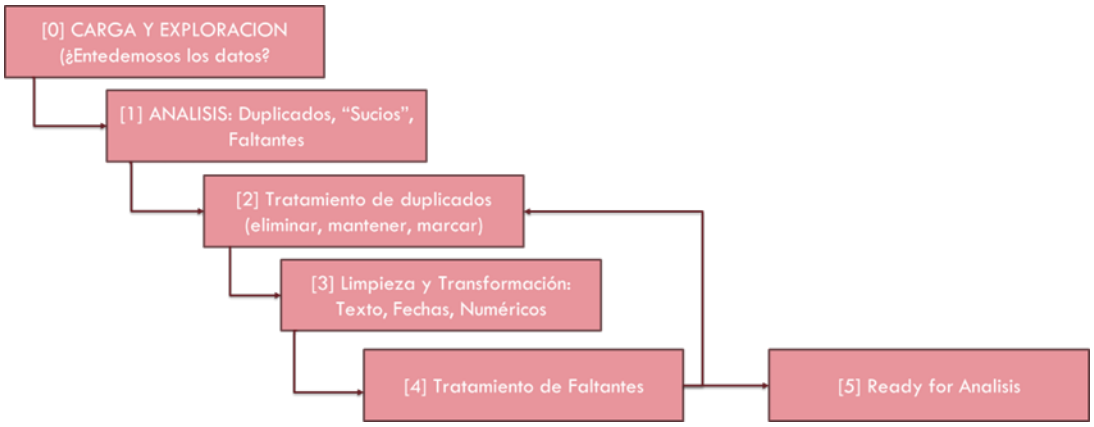
November 29, 2023



0.1 ETL: Missings y Wrap-Up

0.1.1 WrapUp

Como final vamos a repasar primero todo el proceso inicial una vez tengo los datos sobre los que quiero trabajar:



No necesariamente los pasos 1 a 4 tienen por qué seguir ese orden pero esta imágen te da una guía de trabajo que con el tiempo, o ya, adaptarás a tu estilo. En cualquier caso, recuerda siempre que antes de empezar debes hacer el paso 0, entender o tener claro que quieren decir cada uno de los datos que manejo y en ese entendimiento va el rango de valores que pueden tomar.

0.2 Tratamiento de faltantes

Existen varias alternativas cuando nos encontramos con datos incompletos o *missing data*. Y ahora vamos a ver de una forma teórica el tratamiento pero también de una forma práctica, repasando el dataset que ya utilizamos en la práctica obligatoria final del sprint anterior. Carguemos esos datos en nuestro dataframe.

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

df_viajes = pd.read_csv("./data/dataset_viajes.csv")
```

[2]:
```
df_viajes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Id_vuelo    1000 non-null   object
 1   Aircompany  1000 non-null   object
 2   Origen      1000 non-null   object
 3   Destino     1000 non-null   object
 4   Distancia   872 non-null    float64
 5   avion       1000 non-null   object
 6   consumo_kg  862 non-null    object
 7   duracion    853 non-null    float64
dtypes: float64(2), object(6)
memory usage: 62.6+ KB
```

Vamos a ver ejemplos con ”Distancia”,”consumo_kg”,”duracion”

0.2.1 Clasificación

Por su origen, y dificultad para ”recuperarlos”, los datos faltantes se pueden clasificar en tres categorías: 1. **Missing Completely at Random (MCAR)**. Estos son datos que se pierden de verdad de forma esporádica y aleatoria. La pérdida de datos no tiene que ver con la observación estudiada. Por ejemplo, un sensor que se quede sin batería, un cuestionario perdido en una oficina de correos, o una muestra sanguínea fallida en un laboratorio. En general, como es tan aleatoria la perdida, si no es masiva, son datos que se pueden recuperar o estimar a partir de otras ”filas” con datos similares (los que rellenamos con las medidas, etc de otros campos) 2. **Missing at Random (MAR)**. El hecho de la pérdida está relacionado con otra variable. Por ejemplo, telefonos que se estropean y ya no podemos medir su velocidad de acceso a internet. Podemos recuperar ese datos mirando telefonos similares que no se estropean, es peor que el anterior pero aún podemos tratarlo. 3. **Missing not at Random (MNAR)**. Datos incompletos que no se explican por motivos anteriores y que en general no podemos recuperar (estos son candidatos a que los borremos)

Todo esto está muy bien, pero ¿para qué nos sirve? Para fardar en un concurso, normalmente no vas a saber el origen de la pérdida de datos y salvo en el caso de que sepas que es un punto 3, que los tirarás, tus decisiones de qué hacer dependerán de la cantidad de datos que tengas y de lo

informativo o importante que consideres el dato faltante.

En general, si tienes muchos datos, tira los missings, no pierdas el tiempo. Sólo si tus datos son escasos y estás en modo EDA, intenta recuperar, pero ojo si tienes missings, pocos datos y crees que son valiosos los que has perdido, intenta generar nuevos o recuperarlos pero no estimarlos (para Machine Learning, estimar sobre lo que ya existe y luego usarlo como si fuera verdad, es una forma de introducir sesgos "peligrosos", vease ChatsGPTs que aprenden de ChatGPTs)

Y sin aún así quieres "recuperar"...

1 Aproximaciones para tratar los missings

1.0.1 A. Intenta obtenerlos

A veces es posible encontrar los valores incompletos (repitiendo una encuesta, buscando en otras fuentes, etc.). Esto no suele ser lo habitual.

En nuestro dataset era posible para las distancias, ya que teniendo Origen y Destino la distancia no cambia y podemos completar esos campos faltantes.

[3]:
```
df_viajes_sin_na = df_viajes.copy()
df_viajes_sin_na["Destino"] = df_viajes_sin_na.Destino.str.lower().str.
 ↪capitalize() # Primero hay que limpiar, recuerda el ciclo pintado␣
 ↪anteriormente

df_viajes_sin_na["Distancia_Corregida"] = df_viajes_sin_na.
 ↪groupby(["Origen","Destino"])["Distancia"].transform("mean")
#tansform no da una media por cada valor sino un valor unico que aplicara a␣
 ↪ambos elementos en este caso la media
```

[ ]:
```
```

[4]:
```
df_viajes_sin_na.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Id_vuelo             1000 non-null   object
 1   Aircompany           1000 non-null   object
 2   Origen               1000 non-null   object
 3   Destino              1000 non-null   object
 4   Distancia            872 non-null    float64
 5   avion                1000 non-null   object
 6   consumo_kg           862 non-null    object
 7   duracion             853 non-null    float64
 8   Distancia_Corregida  1000 non-null   float64
dtypes: float64(3), object(6)
memory usage: 70.4+ KB
```

1.0.2 Descartar datos, es decir las filas

Omitir los registros (filas) con algún dato faltante y analizar el dataset resultante. Si el tamaño del conjunto de datos es grande, y no hay demasiados missing values, puede ser una estrategia válida. Sin embargo, cuando no tenemos muchos datos o no se satisface MCAR, no es la mejor aproximación, y puede causar sesgo en los datos.

Aún así, cómo hacerlo con Pandas, recordamos que en la práctica nuestro criterio fue descartar las filas que tenían missing en los tres campos, ¿por qué? Porque tenían demasiado "error", rellenarlos con estimaciones (medias, modas, modelos de IA, etc) no tienen sentido porque convierte al dato en demasiado "artificial". Hagámoslo:

[5]:
```
df_viajes_sin_na_I = df_viajes.dropna(subset =␣
↪["Distancia","consumo_kg","duracion"], how = "all")
# tira todas todas las filas cuyos 3 columnas tenga nulos(es lo que quiere␣
↪decir)
```

[6]:
```
df_viajes_sin_na_I.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 997 entries, 0 to 999
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Id_vuelo    997 non-null    object
 1   Aircompany  997 non-null    object
 2   Origen      997 non-null    object
 3   Destino     997 non-null    object
 4   Distancia   872 non-null    float64
 5   avion       997 non-null    object
 6   consumo_kg  862 non-null    object
 7   duracion    853 non-null    float64
dtypes: float64(2), object(6)
memory usage: 70.1+ KB
```

[7]:
```
condicion = (df_viajes_sin_na_I["duracion"].isna()) &␣
↪(df_viajes_sin_na_I["consumo_kg"].isna()) & (df_viajes_sin_na_I["Distancia"].
↪isna())
df_viajes_sin_na_I[condicion]
```

[7]:
```
Empty DataFrame
Columns: [Id_vuelo, Aircompany, Origen, Destino, Distancia, avion, consumo_kg,
duracion]
Index: []
```

1.0.3 Eliminar campos

Si una variable tiene muchos missings, una opción puede ser eliminar la columna del dataset. Por ejemplo, una variable con el 99% de nulos, no aportará mucha información y podremos eliminarla. En cualquier caso, es una decisión que hay que tomar con cuidado, y depende de cada caso.

Para saber cual es la prorporción de nulos, acudíamos a:

[8]:
```
df_viajes["duracion"].value_counts(normalize = True, dropna = False)[np.NaN]
```

[8]: 0.147

[9]:
```
df_viajes["consumo_kg"].value_counts(normalize = True, dropna = False)[np.NaN]
```

[9]: 0.138

[10]:
```
df_viajes["Distancia"].value_counts(normalize = True, dropna = False)[np.NaN]
```

[10]: 0.128

Son porcentajes muy bajos, es decir perdemos mucha info (más del 85% de los datos) si nos deshacemos de la columna, el valor de la columna ya es otra cosa. Pero si aún así quisieramos eliminar esas columnas, usaríamos drop:

[11]:
```
df_viajes_sin_na_II = df_viajes.drop(columns=["consumo_kg"])
```

1.0.4 Media, Mediana y Moda

En lugar de eliminar, reemplazamos valores missing con estimaciones estadísticas como la media, la moda o la mediana. En una sustitución por la media, el valor medio de una variable se usa en lugar del valor de los datos que faltan para esa misma variable. Esto tiene la ventaja de no cambiar la media muestral de esa variable. Sin embargo, con valores faltantes que no son estrictamente aleatorios, especialmente en presencia de una gran desigualdad en el número de valores faltantes para las diferentes variables, el método de sustitución de medias puede conducir a un sesgo inconsistente.

Este es el método que sugeríamos en las unidades dedicadas a Pandas, pero ten en cuenta que como se dice anteriormente se introduce sesgo. En cualquier caso, y como vimos en la práctica se puede hilar fino empleando la media de agrupaciones.

[12]:
```
df_viajes_sin_na_IV = df_viajes.copy()
df_viajes_sin_na_IV["consumo_kg"] = df_viajes_sin_na_IV["consumo_kg"].str.
 ↪replace(",",".").astype("float") # Primero limpiar
df_viajes_sin_na_IV["consumo_kg_medio"] = df_viajes_sin_na_IV.
 ↪groupby(["Aircompany","avion"])["consumo_kg"].transform("mean")
df_viajes_sin_na_IV.loc[df_viajes_sin_na_IV.consumo_kg.isna(),"consumo_kg"] =\
      df_viajes_sin_na_IV.loc[df_viajes_sin_na_IV.consumo_kg.
 ↪isna(),"consumo_kg_medio"]
df_viajes_sin_na_IV.info()#oye en aquellas filas dodne el consumo xkg es nulo,␣
 ↪cambiame el valor nulo, c por el valor de la columna del consumo por kg medio
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
```

```
 0   Id_vuelo          1000 non-null   object
 1   Aircompany        1000 non-null   object
 2   Origen            1000 non-null   object
 3   Destino           1000 non-null   object
 4   Distancia         872 non-null    float64
 5   avion             1000 non-null   object
 6   consumo_kg        1000 non-null   float64
 7   duracion          853 non-null    float64
 8   consumo_kg_medio  1000 non-null   float64
dtypes: float64(4), object(5)
memory usage: 70.4+ KB
```

1.0.5 Añadir variable binaria indicando NaNs

Como complemento a estimar los valores perdidos o NaN, podemos capturar el hecho de que es un missings re-estimado creando una variable binaria adicional indicando si era un valor missing (1, True) o no (0, False). Esto nos permitira descontarlos en casos que no queramos contar con ellos o tenerlos en cuenta en casos que sí con solo filtrar o no por ese campo adicional.

IMPORTANTE: Esto se tiene que hacer antes de reimputar o estimar los missing, después ya no se sabrá cuales lo eran o no. O tratar con una copia como hemos hecho nosotros.

[13]:
```
df_viajes_sin_na_IV["Era_missing_consumo_kg"] = df_viajes.consumo_kg.isna() #␣
 ↪Como están alineados nos podemos permitir esto
```

[14]:
```
# Para cualquier calculo que no queramos usar los valores que eran missing:
df_viajes_sin_na_IV.loc[df_viajes_sin_na_IV.Era_missing_consumo_kg == False] #␣
 ↪y operamos con el resultado de este filtrado
```

[14]:
```
           Id_vuelo  Aircompany      Origen     Destino  Distancia  \
1    Fly_BaRo_10737        FlyQ        Bali        Roma    12738.0
3    Mol_PaCi_10737  MoldaviAir       París  Cincinnati     6370.0
4    Tab_CiRo_10747    TabarAir  Cincinnati        Roma     7480.0
5    Mol_CaMe_10737  MoldaviAir       Cádiz   Melbourne    20029.0
6    Mol_PaLo_11320  MoldaviAir       París     Londres      344.0
..              …         …          …          …        …
995  Pam_LoNu_10747   PamPangea     Londres  Nueva York     5566.0
996  Mol_MeLo_10747  MoldaviAir   Melbourne     Londres    16900.0
997  Mol_BaPa_10747  MoldaviAir        Bali       París    11980.0
998  Air_CaCi_10747      Airnar       Cádiz  Cincinnati     6624.0
999  Air_PaCi_10737      Airnar       París  Cincinnati     6370.0

            avion     consumo_kg  duracion  consumo_kg_medio  \
1     Boeing 737   33479.132544    1167.0      19625.167545
3     Boeing 737   17027.010000     503.0      30645.282695
4     Boeing 747   86115.744000     518.0      57244.692000
5     Boeing 737   53148.153240       NaN      30645.282695
6    Airbus A320     915.246400      44.0       8395.822284
```

```
..           …             …       …              …
995   Boeing 747   62300.238000     391.0     136578.780816
996   Boeing 747  194854.566400    1326.0     132384.256186
997   Boeing 747  128983.868000     818.0     132384.256186
998   Boeing 747   72024.076800     461.0      83111.282099
999   Boeing 737   16872.219000     503.0      25359.943711

     Era_missing_consumo_kg
1                     False
3                     False
4                     False
5                     False
6                     False
..                      …
995                   False
996                   False
997                   False
998                   False
999                   False

[862 rows x 10 columns]
```

1.0.6 Otras técnicas

Además de las indicadas, existen otras técnicas más complejas para la imputación de missings, basadas en modelos de Machine Learning, las cuales consisten en predecir un missing en función de otras variables, que se aconsejan sólo para EDA (esto es cosa mía, de Jaime, pero como no habemos visto Machine Learning no lo haréis hasta el bloque siguiente...)