Author: ivanjdevs

Data Cleaning

In [1]:
```
# Importar la librería pandas que nos ofrece un casi ilimitado número de herramientas para manejar datasets y numpy para temas de arrays y operaciones numéricas.
import pandas as pd
import numpy as np
```

In [2]:
```
#Creemos un dataframe que contien los scores de 20 personas (scores en lo que sea), su país y región de origen:

dict = {'First Score':[98, 90, 86, 95, '?', 88, '?', 90, 86, 95, 80, 88,100, 90, 86, 95, 80, 88, 95, 70],
        'Second Score': ['?', '?', '!', 68, 60, 48, 53, 67, 49, '!', 50, 45, 56, 68, 60, 48, 53, 67, 49, 50],
        'Third Score':[68, 76, 88, 71, 74, 79, 80, 45, 49, 76, 90, 88, 67, '?', 81, '?', 91, '?', 59, 72],
        'Fourth Score':[50, 45, 56, 68, '?', 48, 53, 67, '?', '?', '?', 45, '?', 68, 60, 48, 53, 67, 49, 50],
        'Fifth Score':[66, '@', 56, 68, 60, 48, '?', 67, 49, 50, 50, 45, 56, 68, 60, 55, 89, '!', 78, '!'],
        'Country':['Algeria', 'Australia', 'Hungary', 'Sweden', 'Australia', 'Australia', 'Canada', 'Australia', 'New Zealand', 'Iraq',
                   'Philippines', 'Philippines', 'United Kingdom', 'Malaysia', 'New Zealand', '?', 'Iran', 'New Zealand', 'New Zealand', 'Spain'],
        'Region':['Africa', 'Oceania', 'Europe', 'Europe', 'Oceania', 'Oceania', 'North America', 'Oceania', 'Oceania', 'Asia',
                   'Asia', 'Asia', 'Europe', 'Asia', '!', '?', 'Asia', 'Oceania', 'Oceania', '!']
       }

# creating a dataframe from dictionary
dataf1 = pd.DataFrame(dict)
dataf1
```

Out[2]:

	First Score	Second Score	Third Score	Fourth Score	Fifth Score	Country	Region
0	98	?	68	50	66	Algeria	Africa
1	90	?	76	45	@	Australia	Oceania
2	86	!	88	56	56	Hungary	Europe
3	95	68	71	68	68	Sweden	Europe
4	?	60	74	?	60	Australia	Oceania
5	88	48	79	48	48	Australia	Oceania
6	?	53	80	53	?	Canada	North America
7	90	67	45	67	67	Australia	Oceania
8	86	49	49	?	49	New Zealand	Oceania
9	95	!	76	?	50	Iraq	Asia
10	80	50	90	?	50	Philippines	Asia
11	88	45	88	45	45	Philippines	Asia
12	100	56	67	?	56	United Kingdom	Europe
13	90	68	?	68	68	Malaysia	Asia
14	86	60	81	60	60	New Zealand	!
15	95	48	?	48	55	?	?
16	80	53	91	53	89	Iran	Asia
17	88	67	?	67	!	New Zealand	Oceania
18	95	49	59	49	78	New Zealand	Oceania
19	70	50	72	50	!	Spain	!

In [3]:
```
#Busquemos si existen estos caracteres en el dataframe que hemos llamado dataf1
carac=['?', '#', '!', '@', '/', '', '%']
[True if item in dataf1.values else False for item in carac]

##Esto nos arrojará un array donde nos dirá si cada caracter dentro de la lista 'carac' existe (TRUE) o no (FALSE) dentro del dataframe
```

Out[3]:
```
[True, False, True, True, False, False, False]
```

La desventaja de lo anterior es que nos tocaría colocar dentro de 'carac' todos los caracteres especiales posibles que existan, con el fin de que todos ellos sean buscados dentro del dataframe.

In [4]:
```
#Hagamos un dataframe auxiliar que contendrá valores booleanos: TRUE en cada celda donde exista un caracter especificado dentro de 'isin' y FALSE donde no.
daux=dataf1.isin(['?', '@', '!'])
daux
```

Out[4]:

	First Score	Second Score	Third Score	Fourth Score	Fifth Score	Country	Region
0	False	True	False	False	False	False	False
1	False	True	False	False	True	False	False
2	False	True	False	False	False	False	False
3	False	False	False	False	False	False	False
4	True	False	False	True	False	False	False
5	False	False	False	False	False	False	False
6	True	False	False	False	True	False	False
7	False	False	False	False	False	False	False
8	False	False	False	True	False	False	False
9	False	True	False	True	False	False	False
10	False	False	False	True	False	False	False
11	False	False	False	False	False	False	False
12	False	False	False	True	False	False	False
13	False	False	True	False	False	False	False
14	False	False	False	False	False	False	True
15	False	False	True	False	False	True	True
16	False	False	False	False	False	False	False
17	False	False	True	False	True	False	False
18	False	False	False	False	False	False	False
19	False	False	False	False	True	False	True

¿Para qué sirve lo anterior?. Sirve por si se desea contar cuantos caracteres (cualquiera que especifiquemos dentro de la lista carac) tiene cada columna, a modo de tener una idea de qué tantos valores no válidos hay en una columna y decidir el método a usar para lidiar con ellos (eliminar esas filas, reemplazar por la media o valor mas repetido..) Esto es útil sobre todo en dataframes grandes. En nuestro dataframe de ejemplo, visualmente podemos hacer esa cuenta.

In [5]:
```
#Ya después de lo anterior, lo que queda es contar los valores de cada columna con el metodo values_counts().
#Usemos un ciclo for para hacerlo para cada columna del dataframe dataf1.

#Ya sabemos que TRUE será la cantidad de caracteres (cualquiera de los especificados dentro de la lista 'carac') que hay en cada columna.
for column in daux:
    print (column)
    print (daux[column].value_counts())
    print(" ")
```

```
First Score
False    18
True      2
Name: First Score, dtype: int64

Second Score
False    16
True      4
Name: Second Score, dtype: int64

Third Score
False    17
True      3
Name: Third Score, dtype: int64

Fourth Score
False    15
True      5
Name: Fourth Score, dtype: int64

Fifth Score
False    16
True      4
Name: Fifth Score, dtype: int64

Country
False    19
True      1
Name: Country, dtype: int64

Region
False    17
True      3
Name: Region, dtype: int64
```

Como se observa, en cada columna no hay mas de 5 valores con caracteres no válidos ('?', '#', '!', '@', '/', '', '%'), entendiendo claramente que son no válidos dentro del contexto de la información que contiene el dataframe, que son scores y países.

En caso de que no nos interese saber que tantos valores no válidos hay dentro del dataframe sino reemplazarlos de una vez por NaN values, podemos optar directamente por hacer lo que se presenta en la siguiente línea de código:

In [6]:
```
#Reemplazar todos esos valores no válidos con np.nan

#(Recordar que Nan es el marcador por defecto de Python para valores faltantes por razones de conveniencia y velocidad de cómputo.)

carac=['?', '#', '!', '@', '/', '', '%']
dataf1.replace(carac, np.nan, inplace = True)
dataf1

#Estas líneas también funcionan igual que las anteriores
# dataf1.replace(['?', '#', '!', '@', '/', '', '%'], np.nan, inplace = True)
#dataf1
```

Out[6]:

	First Score	Second Score	Third Score	Fourth Score	Fifth Score	Country	Region
0	98.0	NaN	68.0	50.0	66.0	Algeria	Africa
1	90.0	NaN	76.0	45.0	NaN	Australia	Oceania
2	86.0	NaN	88.0	56.0	56.0	Hungary	Europe
3	95.0	68.0	71.0	68.0	68.0	Sweden	Europe
4	NaN	60.0	74.0	NaN	60.0	Australia	Oceania
5	88.0	48.0	79.0	48.0	48.0	Australia	Oceania
6	NaN	53.0	80.0	53.0	NaN	Canada	North America
7	90.0	67.0	45.0	67.0	67.0	Australia	Oceania
8	86.0	49.0	49.0	NaN	49.0	New Zealand	Oceania
9	95.0	NaN	76.0	NaN	50.0	Iraq	Asia
10	80.0	50.0	90.0	NaN	50.0	Philippines	Asia
11	88.0	45.0	88.0	45.0	45.0	Philippines	Asia
12	100.0	56.0	67.0	NaN	56.0	United Kingdom	Europe
13	90.0	68.0	NaN	68.0	68.0	Malaysia	Asia
14	86.0	60.0	81.0	60.0	60.0	New Zealand	NaN
15	95.0	48.0	NaN	48.0	55.0	NaN	NaN
16	80.0	53.0	91.0	53.0	89.0	Iran	Asia
17	88.0	67.0	NaN	67.0	NaN	New Zealand	Oceania
18	95.0	49.0	59.0	49.0	78.0	New Zealand	Oceania
19	70.0	50.0	72.0	50.0	NaN	Spain	NaN

In [7]:
```
#Hecho lo anterior, revisemos el tipo de dato de cada columna.
dataf1.dtypes
```

Out[7]:
```
First Score     float64
Second Score    float64
Third Score     float64
Fourth Score    float64
Fifth Score     float64
Country          object
Region           object
dtype: object
```

Vemos que hay coherencia entre los tipos de datos anteriormente listados y los valores que se observan en cada columna del dataframe, es decir, efectivamente las primeras cinco columnas son datos numéricos tipo decimal y las dos últimas son columnas con datos tipo objeto o string.

Podemos proceder a reemplazar en cada columna los valores NaN mediante el método que más se nos acomode. En el caso de las columnas con datos tipo numérico podemos reemplazar los NaN values por la media de cada columna.

In [8]:
```
#Vamos a reemplazar los valores NaN de las primeras cinco columnas con la media de cada una de ellas. Hagámoslo con un ciclo for
i=0
for col in dataf1:
    i=i+1
    if i <= 5:
```

```
        dataf1[col].replace(np.nan, dataf1[col].mean(), inplace=True)

#Usamos un contador i que llegue solo hasta 5 ya solo queremos que haga la función replace hasta l qui ta columna, que es hasta donde hay datos numéricos.
```

In [9]:
```
dataf1
```

Out[9]:

	First Score	Second Score	Third Score	Fourth Score	Fifth Score	Country	Region
0	98.000000	55.6875	68.000000	50.000000	66.0000	Algeria	Africa
1	90.000000	55.6875	76.000000	45.000000	60.3125	Australia	Oceania
2	86.000000	55.6875	88.000000	56.000000	56.0000	Hungary	Europe
3	95.000000	68.0000	71.000000	68.000000	68.0000	Sweden	Europe
4	88.888889	60.0000	74.000000	55.133333	60.0000	Australia	Oceania
5	88.000000	48.0000	79.000000	48.000000	48.0000	Australia	Oceania
6	88.888889	53.0000	80.000000	53.000000	60.3125	Canada	North America
7	90.000000	67.0000	45.000000	67.000000	67.0000	Australia	Oceania
8	86.000000	49.0000	49.000000	55.133333	49.0000	New Zealand	Oceania
9	95.000000	55.6875	76.000000	55.133333	50.0000	Iraq	Asia
10	80.000000	50.0000	90.000000	55.133333	50.0000	Philippines	Asia
11	88.000000	45.0000	88.000000	45.000000	45.0000	Philippines	Asia
12	100.000000	56.0000	67.000000	55.133333	56.0000	United Kingdom	Europe
13	90.000000	68.0000	73.764706	68.000000	68.0000	Malaysia	Asia
14	86.000000	60.0000	81.000000	60.000000	60.0000	New Zealand	NaN
15	95.000000	48.0000	73.764706	48.000000	55.0000	NaN	NaN
16	80.000000	53.0000	91.000000	53.000000	89.0000	Iran	Asia
17	88.000000	67.0000	73.764706	67.000000	60.3125	New Zealand	Oceania
18	95.000000	49.0000	59.000000	49.000000	78.0000	New Zealand	Oceania
19	70.000000	50.0000	72.000000	50.000000	60.3125	Spain	NaN

In [10]:
```
#Revisemos la media de cada columna y comprobemos visualmente que en cada una de ellas los valores NaN han sido reemplazados por dicha media.
print ("La media de la primera columna es\n", dataf1['First Score'].mean(),
       "\nLa media de la segunda columna es\n", dataf1['Second Score'].mean(),
       "\nLa media de la tercera columna es\n", dataf1['Third Score'].mean(),
       "\nLa media de la cuarta columna es\n", dataf1['Fourth Score'].mean(),
       "\nLa media de la quinta columna es\n", dataf1['Fifth Score'].mean())
```

```
La media de la primera columna es
 88.88888888888889
La media de la segunda columna es
 55.6875
La media de la tercera columna es
 73.76470588235294
La media de la cuarta columna es
 55.133333333333326
La media de la quinta columna es
 60.3125
```

Para las dos últimas columnas podemos revisar cual es el dato mas común.

In [11]:
```
print("El país con mas apariciones en la columna Country es:\n", dataf1['Country'].value_counts().idxmax())
print(" ")
print("La región con mas apariciones en la columna Region es:\n", dataf1['Region'].value_counts().idxmax())
```

```
El país con mas apariciones en la columna Country es:
 Australia

La región con mas apariciones en la columna Region es:
 Oceania
```

Aquí dependerá de lo que requiera el análisis que se esté haciendo. Se puede optar por reemplazar los valores NaN de las últimas dos columnas por el valor mas común o no.

Traigamos un archivo mas grande. El achivo que se va a trabajar esta en la web. Viene en formato csv (Comma separated value)

In [12]:
```
## Guardando la url donde está contenido el archivo que en una variable
loc="https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DA0101EN/auto.csv"
```

In [13]:
```
headers = ["symboling","normalized-losses","make","fuel-type","aspiration", "num-of-doors","body-style",
         "drive-wheels","engine-location","wheel-base", "length","width","height","curb-weight","engine-type",
         "num-of-cylinders", "engine-size","fuel-system","bore","stroke","compression-ratio","horsepower",
         "peak-rpm","city-mpg","highway-mpg","price"]
```

In [14]:
```
##Cargando el dataset en una variable (puede darle a la variable el nombre que se le de la gana)
dcars = pd.read_csv(loc, names = headers)
```

In [15]:
```
dcars.head()
```

Out[15]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg	price
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	13495
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	16500
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154	5000	19	26	16500
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102	5500	24	30	13950
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115	5500	18	22	17450

5 rows × 26 columns

In [16]:
```
##Número de filas y columnas
dcars.shape
```

Out[16]:
```
(205, 26)
```

In [17]:
```
#Revisemos cuáles de estos caracteres están presentes en el dataframe que hemos importado y llamado dcars
carac=['?', '#', '!', '@', '/', '', '%']
[True if item in dcars.values else False for item in carac]
```

Out[17]:
```
[True, False, False, False, False, False, False]
```

Nos damos cuenta de que, de la lista 'carac', el único caracter presente en el dataframe es "?"

Otra manera de encontrar los diferentes datos y su número de apariciones en una columna es con el método value_counts()

In [18]:
```
#Encontremos los diferentes datos de la columna 'normalized-losses' y cuántas veces aparece cada uno.
dcars['normalized-losses'].value_counts()
```

Out[18]:
```
?      41
161    11
91      8
150     7
128     6
104     6
134     6
94      5
168     5
65      5
95      5
74      5
103     5
102     5
85      5
93      4
122     4
106     4
118     4
148     4
154     3
137     3
115     3
83      3
101     3
125     3
113     2
145     2
164     2
197     2
129     2
188     2
153     2
89      2
110     2
119     2
108     2
81      2
194     2
158     2
192     2
87      2
107     1
77      1
98      1
142     1
90      1
121     1
186     1
231     1
78      1
256     1
Name: normalized-losses, dtype: int64
```

Identificamos de lo anterior que el caracter '?' aparece 41 veces en la columna 'normalized-losses'

In [19]:
```
# Otra forma es usar un ciclo for, preguntando directamente por el caracter en cuestion.
c=0
for i in range (len(dcars)):
    if (dcars.iloc[i]['normalized-losses']=='?'):
        c=c+1
print (c)
```

```
41
```

In [20]:
```
#Podemos hacerlo para todas las columnas del dataframe dcars.
for col in dcars:
    c=0
    for i in range (len(dcars)):
        if (dcars.iloc[i][col]=='?'):
            c=c+1
    print(col)
    print(c)
```

```
symboling
0
normalized-losses
41
make
0
fuel-type
0
aspiration
0
num-of-doors
2
body-style
0
drive-wheels
0
engine-location
0
wheel-base
0
length
0
width
0
height
0
curb-weight
0
engine-type
0
num-of-cylinders
0
engine-size
0
fuel-system
0
bore
4
stroke
4
compression-ratio
0
horsepower
2
peak-rpm
2
city-mpg
0
highway-mpg
0
price
4
```

In [21]:
```
#Hagamos un dataframe auxiliar que contendrá valores booleanos: TRUE en cada celda donde encuentre un caracter '?' y FALSE donde no.
aux=dcars.isin(['?'])
aux
```

Out[21]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg	price
0	False	True	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
1	False	True	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
2	False	True	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
200	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
201	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
202	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
203	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
204	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False

205 rows × 26 columns

In [22]:
```
#Ya después de lo anterior, lo que queda es contar los valores de cada columna. Usemos un ciclo for.
#Ya sabemos que TRUE serán la cantidad de caracteres '?' en cada columna.

for column in aux:
    print (column)
    print (aux[column].value_counts())
    print(" ")
```

```
symboling
False    205
Name: symboling, dtype: int64

normalized-losses
False    164
True      41
Name: normalized-losses, dtype: int64

make
False    205
Name: make, dtype: int64

fuel-type
False    205
Name: fuel-type, dtype: int64

aspiration
False    205
Name: aspiration, dtype: int64

num-of-doors
False    203
True       2
Name: num-of-doors, dtype: int64

body-style
False    205
Name: body-style, dtype: int64

drive-wheels
False    205
Name: drive-wheels, dtype: int64

engine-location
False    205
Name: engine-location, dtype: int64

wheel-base
False    205
Name: wheel-base, dtype: int64

length
False    205
Name: length, dtype: int64

width
False    205
Name: width, dtype: int64

height
False    205
Name: height, dtype: int64

curb-weight
False    205
Name: curb-weight, dtype: int64

engine-type
False    205
Name: engine-type, dtype: int64
```

```
num-of-cylinders
False    205
Name: num-of-cylinders, dtype: int64

engine-size
False    205
Name: engine-size, dtype: int64

fuel-system
False    205
Name: fuel-system, dtype: int64

bore
False    201
True       4
Name: bore, dtype: int64

stroke
False    201
True       4
Name: stroke, dtype: int64

compression-ratio
False    205
Name: compression-ratio, dtype: int64

horsepower
False    203
True       2
Name: horsepower, dtype: int64

peak-rpm
False    203
True       2
Name: peak-rpm, dtype: int64

city-mpg
False    205
Name: city-mpg, dtype: int64

highway-mpg
False    205
Name: highway-mpg, dtype: int64

price
False    201
True       4
Name: price, dtype: int64
```

Y hasta acá va este tema.

Author

Iván P.