



**Nombre y matricula:**

Diego Alonso Luna Ramirez - A01793035

Josep Romagosa Llorden - A01374637

**Equipo 112**

**Nombre del trabajo:**

Reto-> Entrega 1 (16/11) -> Limpieza, análisis, visualización y kmeans

**Fecha de entrega:** 15 de noviembre del 2022

**Campus:** Querétaro

**Programa:** Maestría en Inteligencia Artificial Aplicada (MNA-V)

**Trimestre:** Segundo

**Materia:** Ciencia y analítica de datos

**Nombre del maestro:** Maria de la Paz Rico Fernandez

## • Reto: Entrega 1 -> Aguas Subterraneas

Tecnológico de Monterrey

Materia: Ciencia de Datos

Maestría: Inteligencia Artificial Aplicada

Profesora Maria de la Paz Rico Fernandez

Integrantes del equipo:

Diego Alonso Luna Ramirez - A01793035

Josep Romagosa Llorden - A01374637

#Usar geopandas

```
! pip install qeds fiona geopandas xgboost gensim folium pyLDAvis descartes  
Requirement already satisfied: xgboost in /usr/local/lib/python3.7/dist-packages (1.5.2)  
Requirement already satisfied: gensim in /usr/local/lib/python3.7/dist-packages (3.8.1)  
Requirement already satisfied: folium in /usr/local/lib/python3.7/dist-packages (0.11.0)  
Requirement already satisfied: pyLDAvis in /usr/local/lib/python3.7/dist-packages (2.1.2)  
Requirement already satisfied: descartes in /usr/local/lib/python3.7/dist-packages (1.1.2)  
Requirement already satisfied: quandl in /usr/local/lib/python3.7/dist-packages (f  
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from  
Requirement already satisfied: quantecon in /usr/local/lib/python3.7/dist-packages (0.8.0)  
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (3.1.3)  
Requirement already satisfied: pandas-datareader in /usr/local/lib/python3.7/dist-  
Requirement already satisfied: pyarrow in /usr/local/lib/python3.7/dist-packages (0.15.0)  
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (1.1.3)  
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from  
Requirement already satisfied: openpyxl in /usr/local/lib/python3.7/dist-packages (3.0.9)  
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages (0.23.2)  
Requirement already satisfied: seaborn in /usr/local/lib/python3.7/dist-packages (0.10.0)  
Requirement already satisfied: statsmodels in /usr/local/lib/python3.7/dist-packages (0.11.1)  
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (2.25.1)  
Requirement already satisfied: plotly in /usr/local/lib/python3.7/dist-packages (from  
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages (from  
Requirement already satisfied: attrs>=17 in /usr/local/lib/python3.7/dist-packages (2.3.1)  
Requirement already satisfied: click>=4.0 in /usr/local/lib/python3.7/dist-packages (from  
Requirement already satisfied: six>=1.7 in /usr/local/lib/python3.7/dist-packages (2.4.1)  
Requirement already satisfied: munch in /usr/local/lib/python3.7/dist-packages (from  
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from  
Requirement already satisfied: cligj>=0.5 in /usr/local/lib/python3.7/dist-packages (0.7.0)  
Requirement already satisfied: click-plugins>=1.0 in /usr/local/lib/python3.7/dist-  
Requirement already satisfied: shapely>=1.6 in /usr/local/lib/python3.7/dist-packages (1.7.1)  
Requirement already satisfied: pyproj>=2.2.0 in /usr/local/lib/python3.7/dist-packages (2.4.2)  
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (2020.1)  
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-  
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.7/dist-  
Requirement already satisfied: branca>=0.3.0 in /usr/local/lib/python3.7/dist-packages (0.3.0)  
Requirement already satisfied: jinja2>=2.9 in /usr/local/lib/python3.7/dist-packages (2.11.3)  
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-p  
Requirement already satisfied: funcy in /usr/local/lib/python3.7/dist-packages (from
```

```
Requirement already satisfied: numexpr in /usr/local/lib/python3.7/dist-packages (1.7.2)
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (1.0.1)
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-packages (0.17.1)
Requirement already satisfied: sklearn in /usr/local/lib/python3.7/dist-packages (0.23.2)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (1.4.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (2.4.7)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (0.1.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (3.7.4)
Requirement already satisfied: et-xmlfile in /usr/local/lib/python3.7/dist-packages (1.0.1)
Requirement already satisfied: lxml in /usr/local/lib/python3.7/dist-packages (4.5.2)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (1.25.1)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (2.8.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.7/dist-packages (6.2.2)
Requirement already satisfied: inflection>=0.3.1 in /usr/local/lib/python3.7/dist-packages (0.3.3)
Requirement already satisfied: more-itertools in /usr/local/lib/python3.7/dist-packages (8.1.0)
Requirement already satisfied: numba in /usr/local/lib/python3.7/dist-packages (0.47.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.7/dist-packages (1.6.1)
Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in /usr/local/lib/python3.7/dist-packages (0.39.1)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-packages (3.7.1)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (3.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (2.2.0)
```

```
import numpy as np
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn import preprocessing
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import statistics as sts
import qeds
qeds.themes.mpl_style();
import math
from tqdm import tqdm
import geopandas as gpd
from shapely.geometry import Point
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder

data = pd.read_csv('https://raw.githubusercontent.com/PosgradoMNA/actividades-del-proyecto/main/datos/medellin.csv')
data.head()
```

	CLAVE	SITIO	ORGANISMO_DE_CUENCA	ESTADO	MUNICIPIO
0	DLAGU6	POZO SAN GIL	LERMA SANTIAGO PACIFICO	AGUASCALIENTES	ASIENTOS
1	DLAGU6516	POZO R013 CAÑADA HONDA	LERMA SANTIAGO PACIFICO	AGUASCALIENTES	AGUASCALIENTES
2	DLAGU7	POZO COSIO	LERMA SANTIAGO PACIFICO	AGUASCALIENTES	COSIO

## ▼ Análisis de Variables y Limpieza de Base de Datos

```
4      DLBAJ107      EL      LERMA SANTIAGO PACIFICO      AGUASCALIENTES      LA PAZ
data.describe()
```

	LONGITUD	LATITUD	PERIODO	ALC_mg/L	CONDUCT_mS/cm	SDT_mg/L	EDA
count	1068.000000	1068.000000	1068.0	1064.000000	1062.000000	0.0	1
mean	-101.891007	23.163618	2020.0	235.633759	1138.953013	NaN	2
std	6.703263	3.887670	0.0	116.874291	1245.563674	NaN	3
min	-116.664250	14.561150	2020.0	26.640000	50.400000	NaN	4
25%	-105.388865	20.212055	2020.0	164.000000	501.750000	NaN	5
50%	-102.174180	22.617190	2020.0	215.527500	815.000000	NaN	6
75%	-98.974716	25.510285	2020.0	292.710000	1322.750000	NaN	7
max	-86.864120	32.677713	2020.0	1650.000000	18577.000000	NaN	8

Analizamos el nombre de cada columna.

```
data.info()
```

---	---	---	---
0	CLAVE	1068 non-null	object
1	SITIO	1068 non-null	object
2	ORGANISMO_DE_CUENCA	1068 non-null	object
3	ESTADO	1068 non-null	object
4	MUNICIPIO	1068 non-null	object
5	ACUIFERO	1068 non-null	object
6	SUBTIPO	1068 non-null	object
7	LONGITUD	1068 non-null	float64
8	LATITUD	1068 non-null	float64
9	PERIODO	1068 non-null	int64
10	ALC_mg/L	1064 non-null	float64
11	CALIDAD_ALC	1064 non-null	object
12	CONDUCT_mS/cm	1062 non-null	float64
13	CALIDAD_CONDUC	1062 non-null	object
14	SDT_mg/L	0 non-null	float64
15	SDT_M_mg/L	1066 non-null	object
16	CALIDAD_SDT ra	1066 non-null	object

```

17 CALIDAD_SDT_salin      1066 non-null object
18 FLUORUROS_mg/L          1068 non-null object
19 CALIDAD_FLUO             1068 non-null object
20 DUR_mg/L                 1067 non-null object
21 CALIDAD_DUR              1067 non-null object
22 COLI_FEC_NMP/100_mL     1068 non-null object
23 CALIDAD_COLI_FEC         1068 non-null object
24 N_NO3_mg/L                1067 non-null object
25 CALIDAD_N_NO3             1067 non-null object
26 AS_TOT_mg/L               1068 non-null object
27 CALIDAD_AS                1068 non-null object
28 CD_TOT_mg/L               1068 non-null object
29 CALIDAD_CD                1068 non-null object
30 CR_TOT_mg/L               1068 non-null object
31 CALIDAD_CR                1068 non-null object
32 HG_TOT_mg/L               1068 non-null object
33 CALIDAD_HG                1068 non-null object
34 PB_TOT_mg/L               1068 non-null object
35 CALIDAD_PB                1068 non-null object
36 MN_TOT_mg/L               1068 non-null object
37 CALIDAD_MN                1068 non-null object
38 FE_TOT_mg/L               1068 non-null object
39 CALIDAD_FE                1068 non-null object
40 SEMAFORO                  1068 non-null object
41 CONTAMINANTES             634 non-null object
42 CUMPLE_CON_ALC            1068 non-null object
43 CUMPLE_CON_COND            1068 non-null object
44 CUMPLE_CON_SDT_ra          1068 non-null object
45 CUMPLE_CON_SDT_salin       1068 non-null object
46 CUMPLE_CON_FLUO             1068 non-null object
47 CUMPLE_CON_DUR             1068 non-null object
48 CUMPLE_CON_CF               1068 non-null object
49 CUMPLE_CON_NO3              1068 non-null object
50 CUMPLE_CON_AS               1068 non-null object
51 CUMPLE_CON_CD               1068 non-null object
52 CUMPLE_CON_CR               1068 non-null object
53 CUMPLE_CON_HG               1068 non-null object
54 CUMPLE_CON_PB               1068 non-null object
55 CUMPLE_CON_MN               1068 non-null object
56 CUMPLE_CON_FE               1068 non-null object
dtypes: float64(51) int64(1) object(51)

```

```

variables = ["LONGITUD", "LATITUD", "SEMAFORO"]
variablesBool = ["CUMPLE_CON_ALC", "CUMPLE_CON_COND", "CUMPLE_CON_SDT_ra", "CUMPLE_CON_SDT",
data["SEMAFORO"].unique()

array(['Verde', 'Rojo', 'Amarillo'], dtype=object)

data['SEMAFORO NUMERO'] = data['SEMAFORO'].map({
    'Verde': 1,
    'Rojo': 2,
    'Amarillo': 3
})
data["SEMAFORO NUMERO"].unique()

array([1, 2, 3])

```

Checamos las dimensiones del dataset

```
data.shape
```

```
(1068, 58)
```

Revisemos, algunas columnas que puedan contener valores NA o duplicados en todas sus filas.

```
for j in data.columns:  
    print('Nombre de la Columna: ' + str(j))  
    print('Cantidad de Valores Unicos: ' + str(data[j].nunique()))  
    print('Tipo de Dato: ' + str(data.dtypes[j]))  
    print('Ejemplo: ' + str(np.array(data.loc[:,j][0:10])))  
    print('-----')  
  
    Nombre de la Columna: PB_TOT_mg/L  
    Cantidad de Valores Unicos: 31  
    Tipo de Dato: object  
    Ejemplo: ['<0.005' '<0.005' '<0.005' '<0.005' '<0.005' '<0.005' '<0.005' '<0.005'  
    '<0.005' '<0.005']  
    -----  
    Nombre de la Columna: CALIDAD_PB  
    Cantidad de Valores Unicos: 2  
    Tipo de Dato: object  
    Ejemplo: ['Potable - Excelente' 'Potable - Excelente' 'Potable - Excelente'  
    'Potable - Excelente' 'Potable - Excelente' 'Potable - Excelente'  
    'Potable - Excelente' 'Potable - Excelente' 'Potable - Excelente'  
    'Potable - Excelente']  
    -----  
    Nombre de la Columna: MN_TOT_mg/L  
    Cantidad de Valores Unicos: 362  
    Tipo de Dato: object  
    Ejemplo: ['<0.0015' '<0.0015' '<0.0015' '<0.0015' '<0.0015' '<0.0015' '<0.0015'  
    '0.0041' '0.0113' '1.167']  
    -----  
    Nombre de la Columna: CALIDAD_MN  
    Cantidad de Valores Unicos: 3  
    Tipo de Dato: object  
    Ejemplo: ['Potable - Excelente' 'Potable - Excelente' 'Potable - Excelente'  
    'Potable - Excelente' 'Potable - Excelente' 'Potable - Excelente'  
    'Potable - Excelente' 'Potable - Excelente' 'Potable - Excelente'  
    'Puede afectar la salud']  
    -----  
    Nombre de la Columna: FE_TOT_mg/L  
    Cantidad de Valores Unicos: 615  
    Tipo de Dato: object  
    Ejemplo: ['0.0891' '<0.025' '<0.025' '<0.025' '<0.025' '<0.025' '<0.025' '0.0692'  
    '0.1615' '14.06']  
    -----  
    Nombre de la Columna: CALIDAD_FE  
    Cantidad de Valores Unicos: 2  
    Tipo de Dato: object  
    Ejemplo: ['Potable - Excelente' 'Potable - Excelente' 'Potable - Excelente'  
    'Potable - Excelente' 'Potable - Excelente' 'Potable - Excelente'  
    'Potable - Excelente' 'Potable - Excelente' 'Potable - Excelente'  
    'Sin efectos en la salud - Puede dar color al agua']  
    -----
```

```

Nombre de la Columna: SEMAFORO
Cantidad de Valores Unicos: 3
Tipo de Dato: object
Ejemplo: ['Verde' 'Verde' 'Rojo' 'Verde' 'Rojo' 'Rojo' 'Verde' 'Rojo' 'Verde'
'Rojo']

-----
Nombre de la Columna: CONTAMINANTES
Cantidad de Valores Unicos: 126
Tipo de Dato: object
Ejemplo: [nan nan 'FLUO,AS,' nan 'NO3,' 'CF,' nan 'CONDUC,NO3,' nan
'DT,CF,AS,MN,FE,']

-----
Nombre de la Columna: CUMPLE_CON_ALC
Cantidad de Valores Unicos: 3
Tipo de Dato: object
Ejemplo: ['SI' 'SI' 'SI' 'SI' 'SI' 'SI' 'SI' 'SI' 'SI' 'SI']

```

Observamos que en el apartado hay 2 columnas con 1 solo "**Valor unico**" en el apartado de "**Cantidad de Valores Unicos**", la columna "**SDT\_mg/L**" tiene todas sus valores NA, y en el caso de la columna "**PERIODO**" tiene todos los valores repetidos. Por lo que se eliminarán ambas columnas al considerarse no aptas.

```
data.drop(['PERIODO', 'SDT_mg/L'], axis = 1, inplace = True)
```

Revisamos si tenemos alguna otra columna con valores NA.

Eliminamos la **columna CLAVE** ya que puede interferir en el análisis y tiene un valor distinto para cada renglón. Generalmente esta columna no se eliminaría ya que si proyectamos el modelo a un futuro puede que llegue información complementaria de esa base con nuevas características y la clave sea la llave para hacer la unión con estos nuevos datos, pero en este caso, como no es necesario esta columna; Considerando que tenemos la longitud y latitud (Que es información que tampoco se puede repetir entre los registros).

```
data.drop(columns=["CLAVE"], inplace=True)
```

```
data.isna().any()
```

SITIO	False
ORGANISMO_DE CUENCA	False
ESTADO	False
MUNICIPIO	False
ACUIFERO	False
SUBTIPO	False
LONGITUD	False
LATITUD	False
ALC_mg/L	True
CALIDAD_ALC	True
CONDUCT_mS/cm	True
CALIDAD_CONDUC	True
SDT_M_mg/L	True

```

CALIDAD_SDT_ra           True
CALIDAD_SDT_salin        True
FLUORUROS_mg/L           False
CALIDAD_FLUO              False
DUR_mg/L                  True
CALIDAD_DUR               True
COLI_FEC_NMP/100_mL      False
CALIDAD_COLI_FEC          False
N_NO3_mg/L                True
CALIDAD_N_NO3              True
AS_TOT_mg/L                False
CALIDAD_AS                 False
CD_TOT_mg/L                False
CALIDAD_CD                 False
CR_TOT_mg/L                False
CALIDAD_CR                 False
HG_TOT_mg/L                False
CALIDAD_HG                 False
PB_TOT_mg/L                False
CALIDAD_PB                 False
MN_TOT_mg/L                False
CALIDAD_MN                 False
FE_TOT_mg/L                False
CALIDAD_FE                 False
SEMAFORO                   False
CONTAMINANTES              True
CUMPLE_CON_ALC             False
CUMPLE_CON_COND             False
CUMPLE_CON_SDT_ra           False
CUMPLE_CON_SDT_salin        False
CUMPLE_CON_FLUO             False
CUMPLE_CON_DUR              False
CUMPLE_CON_CF                False
CUMPLE_CON_NO3              False
CUMPLE_CON_AS                False
CUMPLE_CON_CD                False
CUMPLE_CON_CR                False
CUMPLE_CON_HG                False
CUMPLE_CON_PB                False
CUMPLE_CON_MN                False
CUMPLE_CON_FE                False
SEMAFORO NUMERO             False
dtype: bool

```

Observando que tenemos algunas columnas con valores NA, contabilicemos cuandos valores NA hay por columna.

```
data.isna().sum()
```

SITIO	0
ORGANISMO_DE_CUENCA	0
ESTADO	0
MUNICIPIO	0
ACUIFERO	0
SUBTIPO	0
LONGITUD	0
LATITUD	0

ALC_mg/L	4
CALIDAD_ALC	4
CONDUCT_mS/cm	6
CALIDAD_CONDUC	6
SDT_M_mg/L	2
CALIDAD_SDT_ra	2
CALIDAD_SDT_salin	2
FLUORUROS_mg/L	0
CALIDAD_FLUO	0
DUR_mg/L	1
CALIDAD_DUR	1
COLI_FEC_NMP/100_mL	0
CALIDAD_COLI_FEC	0
N_NO3_mg/L	1
CALIDAD_N_NO3	1
AS_TOT_mg/L	0
CALIDAD_AS	0
CD_TOT_mg/L	0
CALIDAD_CD	0
CR_TOT_mg/L	0
CALIDAD_CR	0
HG_TOT_mg/L	0
CALIDAD_HG	0
PB_TOT_mg/L	0
CALIDAD_PB	0
MN_TOT_mg/L	0
CALIDAD_MN	0
FE_TOT_mg/L	0
CALIDAD_FE	0
SEMAFORO	0
CONTAMINANTES	434
CUMPLE_CON_ALC	0
CUMPLE_CON_COND	0
CUMPLE_CON_SDT_ra	0
CUMPLE_CON_SDT_salin	0
CUMPLE_CON_FLUO	0
CUMPLE_CON_DUR	0
CUMPLE_CON_CF	0
CUMPLE_CON_NO3	0
CUMPLE_CON_AS	0
CUMPLE_CON_CD	0
CUMPLE_CON_CR	0
CUMPLE_CON_HG	0
CUMPLE_CON_PB	0
CUMPLE_CON_MN	0
CUMPLE_CON_FE	0
SEMAFORO_NUMERO	0

dtype: int64

Observamos que algunas pocas columnas contienen algunos valores NA, entre 1 a 6... y una columna ("CONTAMINANTES") con 434 valores. Realmente no afectaría eliminar estos valores NA por ser un pequeño porcentaje (Menos del 3%)

```
dt_NA = data[data.columns[data.isna().any()]]
dt_NA.drop(['CONTAMINANTES'],axis = 1, inplace = True)
dt_NA.columns
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py:4913: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#inplace-mutation
errors=errors,
Index(['ALC_mg/L', 'CALIDAD_ALC', 'CONDUCT_mS/cm', 'CALIDAD_CONDUC',
       'SDT_M_mg/L', 'CALIDAD_SDT_ra', 'CALIDAD_SDT_salin', 'DUR_mg/L',
       'CALIDAD_DUR', 'N_NO3_mg/L', 'CALIDAD_N_NO3'],
      dtype='object')
```

```
◀ ━━━━━━ ▶
```

```
data.dropna(subset = dt_NA.columns, axis = 0, inplace = True)
data.isna().sum()
```

SITIO	0
ORGANISMO_DE_CUENCA	0
ESTADO	0
MUNICIPIO	0
ACUIFERO	0
SUBTIPO	0
LONGITUD	0
LATITUD	0
ALC_mg/L	0
CALIDAD_ALC	0
CONDUCT_mS/cm	0
CALIDAD_CONDUC	0
SDT_M_mg/L	0
CALIDAD_SDT_ra	0
CALIDAD_SDT_salin	0
FLUORUROS_mg/L	0
CALIDAD_FLUO	0
DUR_mg/L	0
CALIDAD_DUR	0
COLI_FEC_NMP/100_mL	0
CALIDAD_COLI_FEC	0
N_NO3_mg/L	0
CALIDAD_N_NO3	0
AS_TOT_mg/L	0
CALIDAD_AS	0
CD_TOT_mg/L	0
CALIDAD_CD	0
CR_TOT_mg/L	0
CALIDAD_CR	0
HG_TOT_mg/L	0
CALIDAD_HG	0
PB_TOT_mg/L	0
CALIDAD_PB	0
MN_TOT_mg/L	0
CALIDAD_MN	0
FE_TOT_mg/L	0
CALIDAD_FE	0
SEMAFORO	0
CONTAMINANTES	427
CUMPLE_CON_ALC	0
CUMPLE_CON_COND	0
CUMPLE_CON_SDT_ra	0
CUMPLE_CON_SDT_salin	0
CUMPLE_CON_FLUO	0
CUMPLE_CON_DUR	0

```
CUMPLE_CON_CF          0
CUMPLE_CON_NO3         0
CUMPLE_CON_AS          0
CUMPLE_CON_CD          0
CUMPLE_CON_CR          0
CUMPLE_CON_HG          0
CUMPLE_CON_PB          0
CUMPLE_CON_MN          0
CUMPLE_CON_FE          0
SEMAFORO NUMERO        0
dtype: int64
```

```
data['CONTAMINANTES'].unique().tolist()
[...,
 'CF,PB,' ,
 'CF,MN,' ,
 'CF,MN,FE,' ,
 'ALC,FLUO,AS,' ,
 'AS,FE,' ,
 'DT,AS,NO3,' ,
 'CONDUC,DT,AS,' ,
 'FLUO,AS,FE,' ,
 'ALC,AS,FE,' ,
 'PB,MN,FE,' ,
 'ALC,CONDUC,SDT_ra,SDT_salin,FLUO,DT,MN,FE,' ,
 'ALC,CONDUC,SDT_ra,SDT_salin,FLUO,DT,AS,MN,FE,' ,
 'ALC,CONDUC,SDT_ra,SDT_salin,DT,NO3,' ,
 'ALC,FLUO,AS,FE,' ,
 'MN,FE,' ,
 'ALC,AS,MN,FE,' ,
 'ALC,MN,' ,
 'NI,FE,' ,
 'CONDUC,MN,FE,' ,
 'ALC,DT,FE,' ,
 'CONDUC,PB,FE,' ,
 'PB,FE,' ,
 'CONDUC,FLUO,DT,' ,
 'ALC,DT,' ,
 'ALC,CONDUC,' ,
 'ALC,CONDUC,AS,MN,' ,
 'AS,MN,FE,' ,
 'DT,PB,' ,
 'PB,' ,
 'CONDUC,SDT_ra,SDT_salin,DT,HG,MN,FE,' ,
 'ALC,CONDUC,DT,NO3,' ,
 'DT,MN,' ,
 'MN,NO3,' ,
 'ALC,CF,' ,
 'FLUO,MN,FE,' ,
 'CONDUC,SDT_ra,SDT_salin,FLUO,DT,PB,' ,
 'CF,CD,MN,FE,' ,
 'ALC,FE,' ,
 'CONDUC,SDT_ra,SDT_salin,DT,AS,FE,' ,
 'ALC,FLUO,AS,NO3,' ,
 'SDT_ra,SDT_salin,DT,NO3,' ,
 'SDT_ra,SDT_salin,DT,CR,' ,
 'CONDUC,FLUO,NO3,' ,
 'CONDUC,DT,CF,AS,NO3,' ,
 'AS,NO3,' ,
 'CONDUC,DT,AS,NO3,' ]
```

```
'CONDUC,FE,NO3,' ,  
'CONDUC,SDT_ra,SDT_salin,DT,AS,MN,' ,  
'CONDUC,DT,CD,' ,  
'DT,CF,' ,  
'CONDUC,FLUO,DT,CF,' ,  
'DT,MN,FE,' ,  
'CONDUC,SDT_ra,SDT_salin,CR,' ,  
'CONDUC,FE,' ,  
'CR,MN,FE,' ,  
'CONDUC,SDT_ra,SDT_salin,FLUO,DT,FE,' ,  
'ALC,DT,AS,MN,' ,  
'FLUO,AS,NO3,' ,  
'ALC,MN,FE,' ,
```

Observamos que pese a relizar la limpieza de valores NA, sobre todas las columnas, aun sigue quedando en la columna "CONTAMINANTES" valores NA. Es posible que algunas filas tienen valores NA, a causa de que en aquellos campos el agua no este contaminada, por lo tanto, esta definida como "NA" (No aplica), por lo que seria ideal modificar estos valores por el valor "**"NO\_CONTAMINADO"**

```
data['CONTAMINANTES'].fillna('NO_CONTAMINADO', inplace=True)  
data.isna().values.any()
```

```
False
```

```
data.dropna(subset = dt_NA.columns, axis = 0, inplace = True)  
data.isna().sum()
```

SITIO	0
ORGANISMO_DE_CUENCA	0
ESTADO	0
MUNICIPIO	0
ACUIFERO	0
SUBTIPO	0
LONGITUD	0
LATITUD	0
ALC_mg/L	0
CALIDAD_ALC	0
CONDUCT_mS/cm	0
CALIDAD_CONDUC	0
SDT_M_mg/L	0
CALIDAD_SDT_ra	0
CALIDAD_SDT_salin	0
FLUORUROS_mg/L	0
CALIDAD_FLUO	0
DUR_mg/L	0
CALIDAD_DUR	0
COLI_FEC_NMP/100_mL	0
CALIDAD_COLI_FEC	0
N_NO3_mg/L	0
CALIDAD_N_NO3	0
AS_TOT_mg/L	0
CALIDAD_AS	0
CD_TOT_mg/L	0
CALIDAD_CD	0

```
CR_TOT_mg/L          0
CALIDAD_CR           0
HG_TOT_mg/L          0
CALIDAD_HG           0
PB_TOT_mg/L          0
CALIDAD_PB           0
MN_TOT_mg/L          0
CALIDAD_MN           0
FE_TOT_mg/L          0
CALIDAD_FE           0
SEMAFORO             0
CONTAMINANTES        0
CUMPLE_CON_ALC       0
CUMPLE_CON_COND       0
CUMPLE_CON_SDT_ra     0
CUMPLE_CON_SDT_salin 0
CUMPLE_CON_FLUO       0
CUMPLE_CON_DUR       0
CUMPLE_CON_CF         0
CUMPLE_CON_NO3        0
CUMPLE_CON_AS         0
CUMPLE_CON_CD         0
CUMPLE_CON_CR         0
CUMPLE_CON_HG         0
CUMPLE_CON_PB         0
CUMPLE_CON_MN         0
CUMPLE_CON_FE         0
SEMAFORO NUMERO      0
dtype: int64
```

Revisamos si algunas de las columnas aun contiene valores nulos

```
print('Cantidad de valores nulos:',data.isnull().sum())
```

```
Cantidad de valores nulos: SITIO          0
ORGANISMO_DE_CUENCA    0
ESTADO                  0
MUNICIPIO               0
ACUIFERO                0
SUBTIPO                 0
LONGITUD                0
LATITUD                 0
ALC_mg/L                0
CALIDAD_ALC              0
CONDUCT_mS/cm            0
CALIDAD_CONDUC            0
SDT_M_mg/L               0
CALIDAD_SDT_ra             0
CALIDAD_SDT_salin          0
FLUORUROS_mg/L            0
CALIDAD_FLUO              0
DUR_mg/L                 0
CALIDAD_DUR               0
COLI_FEC_NMP/100_mL        0
CALIDAD_COLI_FEC            0
N_NO3_mg/L                0
CALIDAD_N_NO3              0
AS_TOT_mg/L                0
```

```

CALIDAD_AS          0
CD_TOT_mg/L        0
CALIDAD_CD          0
CR_TOT_mg/L         0
CALIDAD_CR          0
HG_TOT_mg/L         0
CALIDAD_HG          0
PB_TOT_mg/L         0
CALIDAD_PB          0
MN_TOT_mg/L         0
CALIDAD_MN          0
FE_TOT_mg/L         0
CALIDAD_FE          0
SEMAFORO            0
CONTAMINANTES       0
CUMPLE_CON_ALC      0
CUMPLE_CON_COND     0
CUMPLE_CON_SDT_ra   0
CUMPLE_CON_SDT_salin 0
CUMPLE_CON_FLUO    0
CUMPLE_CON_DUR      0
CUMPLE_CON_CF       0
CUMPLE_CON_NO3      0
CUMPLE_CON_AS       0
CUMPLE_CON_CD       0
CUMPLE_CON_CR       0
CUMPLE_CON_HG       0
CUMPLE_CON_PB       0
CUMPLE_CON_MN       0
CUMPLE_CON_FE       0
SEMAFORO NUMERO     0
dtype: int64

```

Verificamos el tipo de dato, si admite o no admite valores no nulos en cada columna.

```
data.info()
```

#	Column	Non-Null Count	Dtype
0	SITIO	1054 non-null	object
1	ORGANISMO_DE_CUENCA	1054 non-null	object
2	ESTADO	1054 non-null	object
3	MUNICIPIO	1054 non-null	object
4	ACUIFERO	1054 non-null	object
5	SUBTIPO	1054 non-null	object
6	LONGITUD	1054 non-null	float64
7	LATITUD	1054 non-null	float64
8	ALC_mg/L	1054 non-null	float64
9	CALIDAD_ALC	1054 non-null	object
10	CONDUCT_mS/cm	1054 non-null	float64
11	CALIDAD_CONDUC	1054 non-null	object
12	SDT_M_mg/L	1054 non-null	object
13	CALIDAD_SDT_ra	1054 non-null	object
14	CALIDAD_SDT_salin	1054 non-null	object
15	FLUORUROS_mg/L	1054 non-null	object
16	CALIDAD_FLUO	1054 non-null	object
17	DUR_mg/L	1054 non-null	object
18	CALIDAD_DUR	1054 non-null	object

```

19  COLI_FEC_NMP/100_mL    1054 non-null  object
20  CALIDAD_COLI_FEC      1054 non-null  object
21  N_NO3_mg/L             1054 non-null  object
22  CALIDAD_N_NO3          1054 non-null  object
23  AS_TOT_mg/L            1054 non-null  object
24  CALIDAD_AS              1054 non-null  object
25  CD_TOT_mg/L            1054 non-null  object
26  CALIDAD_CD              1054 non-null  object
27  CR_TOT_mg/L            1054 non-null  object
28  CALIDAD_CR              1054 non-null  object
29  HG_TOT_mg/L            1054 non-null  object
30  CALIDAD_HG              1054 non-null  object
31  PB_TOT_mg/L            1054 non-null  object
32  CALIDAD_PB              1054 non-null  object
33  MN_TOT_mg/L            1054 non-null  object
34  CALIDAD_MN              1054 non-null  object
35  FE_TOT_mg/L            1054 non-null  object
36  CALIDAD_FE              1054 non-null  object
37  SEMAFORO                1054 non-null  object
38  CONTAMINANTES           1054 non-null  object
39  CUMPLE_CON_ALC          1054 non-null  object
40  CUMPLE_CON_COND          1054 non-null  object
41  CUMPLE_CON_SDT_ra        1054 non-null  object
42  CUMPLE_CON_SDT_salin     1054 non-null  object
43  CUMPLE_CON_FLUO          1054 non-null  object
44  CUMPLE_CON_DUR           1054 non-null  object
45  CUMPLE_CON_CF             1054 non-null  object
46  CUMPLE_CON_NO3            1054 non-null  object
47  CUMPLE_CON_AS             1054 non-null  object
48  CUMPLE_CON_CD             1054 non-null  object
49  CUMPLE_CON_CR             1054 non-null  object
50  CUMPLE_CON_HG             1054 non-null  object
51  CUMPLE_CON_PB             1054 non-null  object
52  CUMPLE_CON_MN             1054 non-null  object
53  CUMPLE_CON_FE             1054 non-null  object
54  SEMAFORO NUMERO          1054 non-null  int64
dtypes: float64(4), int64(1), object(50)
memory usage: 161.11 KB

```

```

def Analisys(columnName):
    print("\n", columnName)
    print("Tipo ", data[columnName].dtypes)
    print("String value")
    print(data[columnName][pd.to_numeric(data[columnName], errors='coerce').isnull()].unique)

def convertValue(columnName, stringValue, numericValue):
    print("\nReemplazando valores...")
    data.loc[ data[columnName] == stringValue, columnName] = numericValue
    print("Convirtiendo valores a float...")
    data[columnName] = data[columnName].astype(float, errors='ignore')
    print("Tipo: " , data[columnName].dtypes)

def convertString(columnName):
    print("\n", columnName)
    print("Convirtiendo a float...")
    data[columnName] = data[columnName].astype(float, errors='ignore')

```

```
print("Valores String faltantes: ")
print(data[columnNames1].ndarray().columns)
mgl_list = ['AS_TOT_mg/L','CD_TOT_mg/L','COLI_FEC_NMP/100_mL','CR_TOT_mg/L','DUR_mg/L','FE_TOT_mg/L','HG_TOT_mg/L','MN_TOT_mg/L','N_NO3_mg/L','PB_TOT_mg/L']

for x in mgl_list:
    Analisys(x)

    AS_TOT_mg/L
    Tipo object
    String value
    ['<0.01']

    CD_TOT_mg/L
    Tipo object
    String value
    ['<0.003']

    COLI_FEC_NMP/100_mL
    Tipo object
    String value
    ['<1.1']

    CR_TOT_mg/L
    Tipo object
    String value
    ['<0.005']

    DUR_mg/L
    Tipo object
    String value
    ['<20']

    FE_TOT_mg/L
    Tipo object
    String value
    ['<0.025']

    FLUORUROS_mg/L
    Tipo object
    String value
    ['<0.2']

    HG_TOT_mg/L
    Tipo object
    String value
    ['<0.0005']

    MN_TOT_mg/L
    Tipo object
    String value
    ['<0.0015']

    N_NO3_mg/L
    Tipo object
    String value
    ['<0.02']

    PB_TOT_mg/L
    Tipo object
```

```
String value
['<0.005']

SDT_M_mg/L
Tipo object
String value

for x in mgl_list:
    convertString(x)

    AS_IUI_mg/L
    Convirtiendo a float...
    Valores String faltantes:
    ['<0.01']

    CD_TOT_mg/L
    Convirtiendo a float...
    Valores String faltantes:
    ['<0.003']

    COLI_FEC_NMP/100_mL
    Convirtiendo a float...
    Valores String faltantes:
    ['<1.1']

    CR_TOT_mg/L
    Convirtiendo a float...
    Valores String faltantes:
    ['<0.005']

    DUR_mg/L
    Convirtiendo a float...
    Valores String faltantes:
    ['<20']

    FE_TOT_mg/L
    Convirtiendo a float...
    Valores String faltantes:
    ['<0.025']

    FLUORUROS_mg/L
    Convirtiendo a float...
    Valores String faltantes:
    ['<0.2']

    HG_TOT_mg/L
    Convirtiendo a float...
    Valores String faltantes:
    ['<0.0005']

    MN_TOT_mg/L
    Convirtiendo a float...
    Valores String faltantes:
    ['<0.0015']

    N_NO3_mg/L
    Convirtiendo a float...
    Valores String faltantes:
    ['<0.02']
```

```
PB_TOT_mg/L  
Convirtiendo a float...  
Valores String faltantes:  
['<0.005']
```

```
SDT_M_mg/L  
Convirtiendo a float...  
Valores String faltantes:  
[]
```

Convertir los valores numericos de "String" a "Float"

```
#PB_TOT_mg  
convertValue('PB_TOT_mg/L','<0.005',0.005)  
#CD_TOT_mg/L  
convertValue('CD_TOT_mg/L','<0.003',0.003)  
#N_N03_mg/L  
convertValue('N_N03_mg/L','<0.02',0.02)  
#MN_TOT_mg/L  
convertValue('MN_TOT_mg/L','<0.0015',0.0015)  
#HG_TOT_mg/L  
convertValue('HG_TOT_mg/L','<0.0005',0.0005)  
#FLUORUROS_mg/L  
convertValue('FLUORUROS_mg/L','<0.2',0.2)  
#FE_TOT_mg/L  
convertValue('FE_TOT_mg/L','<0.025',0.025)  
#DUR_mg/L  
convertValue('DUR_mg/L','<20',20)  
#CR_TOT_mg/L  
convertValue('CR_TOT_mg/L','<0.005',0.005)  
#COLI_FEC_NMP/100_mL  
convertValue('COLI_FEC_NMP/100_mL','<1.1',1.1)  
#AS_TOT_mg/L  
convertValue('AS_TOT_mg/L','<0.01',0.01)
```

```
Remplazando valores...  
Convirtiendo valores a float...  
Tipo: float64
```

```
Remplazando valores...  
Convirtiendo valores a float...  
Tipo: float64
```

```
Remplazando valores...  
Convirtiendo valores a float...  
Tipo: float64
```

```
Remplazando valores...  
Convirtiendo valores a float...  
Tipo: float64
```

```
Remplazando valores...  
Convirtiendo valores a float...  
Tipo: float64
```

```
Remplazando valores...
Convirtiendo valores a float...
Tipo: float64
```

```
Remplazando valores...
Convirtiendo valores a float...
Tipo: float64
```

```
Remplazando valores...
Convirtiendo valores a float...
Tipo: float64
```

```
Remplazando valores...
Convirtiendo valores a float...
Tipo: float64
```

```
Remplazando valores...
Convirtiendo valores a float...
Tipo: float64
```

```
Remplazando valores...
Convirtiendo valores a float...
Tipo: float64
```

## Ahora revisemos si los valores categoricos son correctos acordide a lo que corresponde.

```
#Categoricos
categoricas = ['CALIDAD_CONDUC','CALIDAD_CONDUC','CALIDAD_SDT_ra','CALIDAD_SDT_salin','CALIDAD_DUR','CALIDAD_COLI_FEC','CALIDAD_N_N03']
#Numericos
numericas = ['ALC_mg/L','CONDUCT_mS/cm','SDT_M_mg/L','FLUORUROS_mg/L','DUR_mg/L','COLI_FEC','NITROGENO_N','FOSFORO_P','CLORO_C','CALIDAD_N_N03']

for x in categoricas:
    print("\n",x)
    print(data[x].unique().tolist())

    CALIDAD_CONDUC
    ['Permisible para riego', 'Buena para riego', 'Dudosa para riego', 'Indeseable para riego', 'No apto para riego']

    CALIDAD_CONDUC
    ['Permisible para riego', 'Buena para riego', 'Dudosa para riego', 'Indeseable para riego', 'No apto para riego']

    CALIDAD_SDT_ra
    ['Cultivos sensibles', 'Excelente para riego', 'Cultivos con manejo especial', 'Cultivos con manejo riguroso', 'Cultivos con manejo estricto']

    CALIDAD_SDT_salin
    ['Potable - Dulce', 'Ligeramente salobres', 'Salobres', 'Salinas']

    CALIDAD_DUR
    ['Potable - Dura', 'Muy dura e indeseable usos industrial y domestico', 'Potable - Muy dura', 'Muy dura e indeseable usos industrial y domestico']

    CALIDAD_COLI_FEC
    ['Potable - Excelente', 'Aceptable', 'Contaminada', 'Buena calidad', 'Fuertemente contaminada']

    CALIDAD_N_N03
```

```
['Potable - Excelente', 'Potable - Buena calidad', 'No apta como FAAP']

CALIDAD_AS
['Apta como FAAP', 'No apta como FAAP', 'Potable - Excelente']

CALIDAD_CD
['Potable - Excelente', 'No apta como FAAP']

CALIDAD_CR
['Potable - Excelente', 'No apta como FAAP']

CALIDAD_HG
['Potable - Excelente', 'No apta como FAAP']

CALIDAD_PB
['Potable - Excelente', 'No apta como FAAP']

CALIDAD_MN
['Potable - Excelente', 'Puede afectar la salud', 'Sin efectos en la salud - Puede da...']

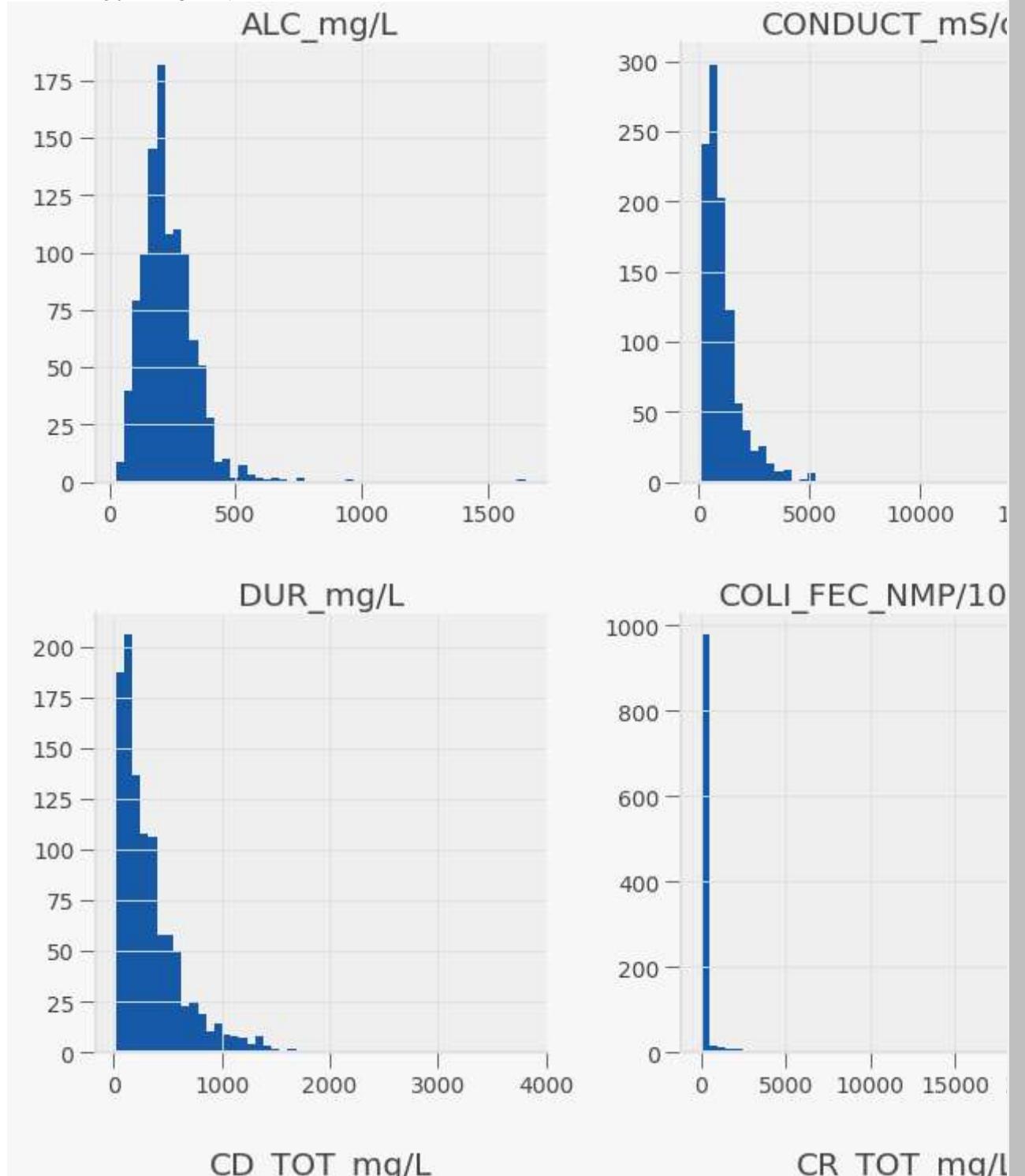
CALIDAD_FE
['Potable - Excelente', 'Sin efectos en la salud - Puede dar color al agua']
```

Ahora realicemos el histograma basado en los valores numéricos.

```
data[numericas].hist(bins = 50, figsize=(25,25))
```

☞

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f2ce6d2c4d0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f2ce6cfbb10>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f2ce6cc1110>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f2ce6c77710>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x7f2ce6c2bd10>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f2ce6bef350>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f2ce6ba59d0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f2ce6b5bf10>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x7f2ce6b5bf50>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f2ce6b1d690>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f2ce6a961d0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f2ce6a4d7d0>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x7f2ce6a04dd0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f2ce69c7410>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f2ce6befd10>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f2ce6a7a110>]],
     dtype=object)
```



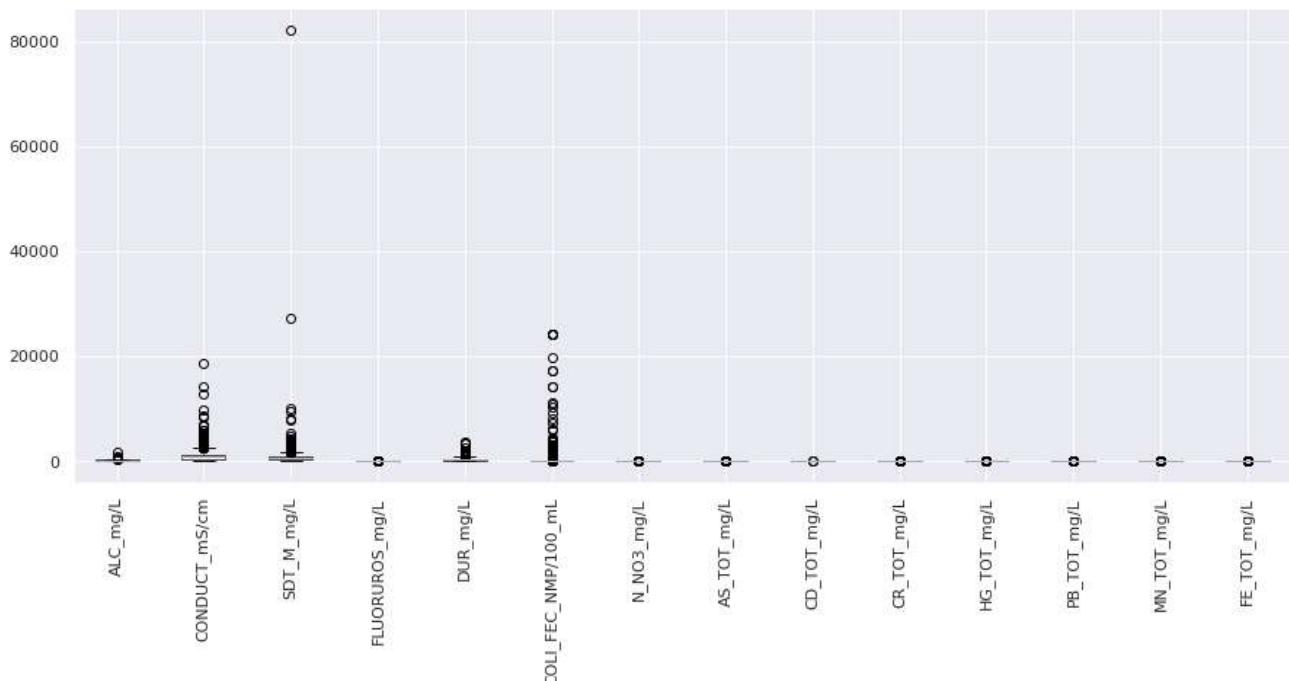


Con la grafica, Conseguimos valores estadísticos como la media, el conteo, la desviación estándar, el valor mínimo, los cuartiles y valor maximo. Esto por cada una de las variables del dataset.

### Graficamos los datos numéricos de entrada para visualizar los outliers

```
sns.set(rc={'figure.figsize':(15,6)})
data[numericas].boxplot(rot=90)
```

<matplotlib.axes.\_subplots.AxesSubplot at 0x7f2ce5f7f590>



### ▼ Identificamos los outliers

```
Q1 = data[numericas].quantile(0.25)
Q3 = data[numericas].quantile(0.75)
IQR = Q3-Q1
```

```
BM = (data[numericas] > (Q3+1.5 *IQR)) | (data[numericas] < (Q1-1.5 *IQR))
```

## Vemos los valores de los outliers

```
data[numericas][BM]
```

	ALC_mg/L	CONDUCT_mS/cm	SDT_M_mg/L	FLUORUROS_mg/L	DUR_mg/L	COLI_FEC_NMP/10
0	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...
1063	NaN	NaN	NaN	NaN	NaN	NaN
1064	NaN	NaN	NaN	NaN	NaN	NaN
1065	NaN	2600.0	1873.0	NaN	NaN	NaN
1066	NaN	NaN	NaN	NaN	NaN	NaN
1067	NaN	NaN	NaN	NaN	NaN	NaN

1054 rows × 14 columns

Enviamos los outliers al upper cap [Q3+IQR\*1.5]

```
for att in numericas:
```

```
    Q1 = data[att].quantile(0.25)
    Q3 = data[att].quantile(0.75)
```

```
IQR = Q3 - Q1
```

```
lower_cap = Q1-IQR*1.5
upper_cap = Q3+IQR*1.5
```

```
BM = data[att] < lower_cap
```

```
candidate_index = data[BM].index
data.loc[candidate_index,att] = lower_cap
```

```
BM = data[att] > upper_cap
```

```
candidate_index = data[BM].index
```

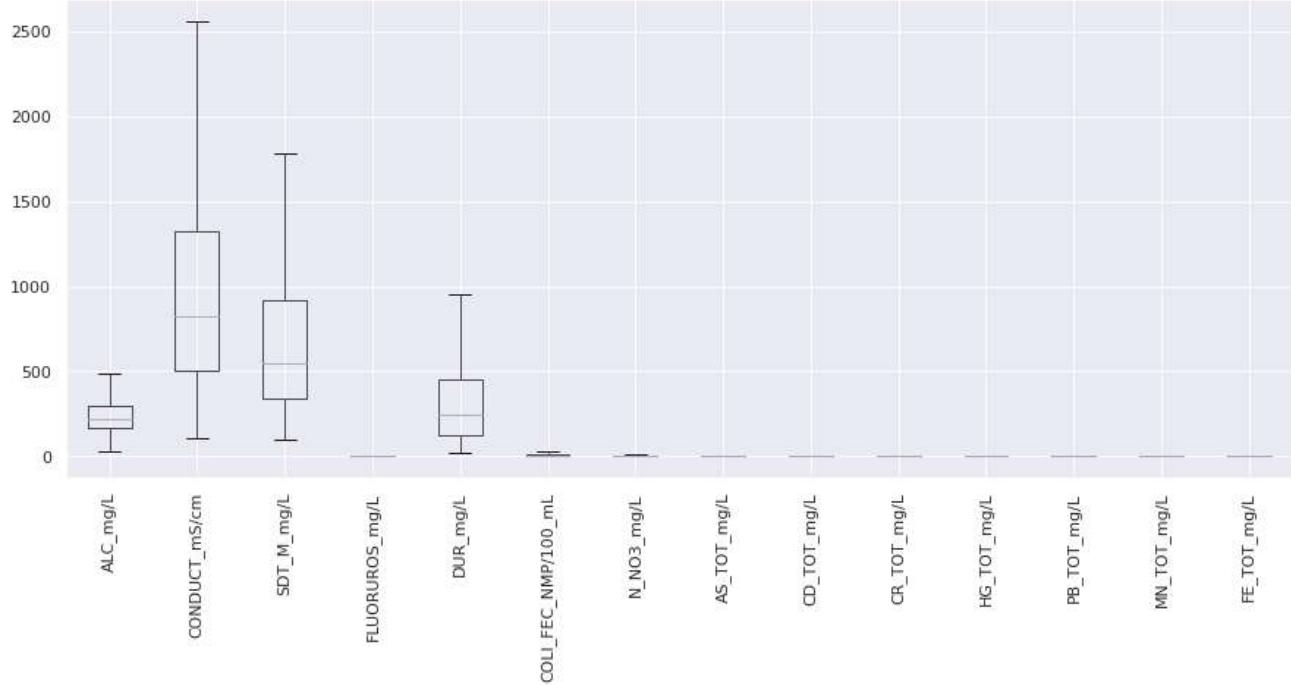
```
data.loc[candidate_index,att] = upper_cap
```

## Graficamos los resultados

```
sns.set(rc={'figure.figsize':(15,6)})
```

```
data[numericas].boxplot(rot=90)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2ce40498d0>
```



## Escalamos los datos numericos de 0 a 1

```
scaler = MinMaxScaler(feature_range=(0, 1))
```

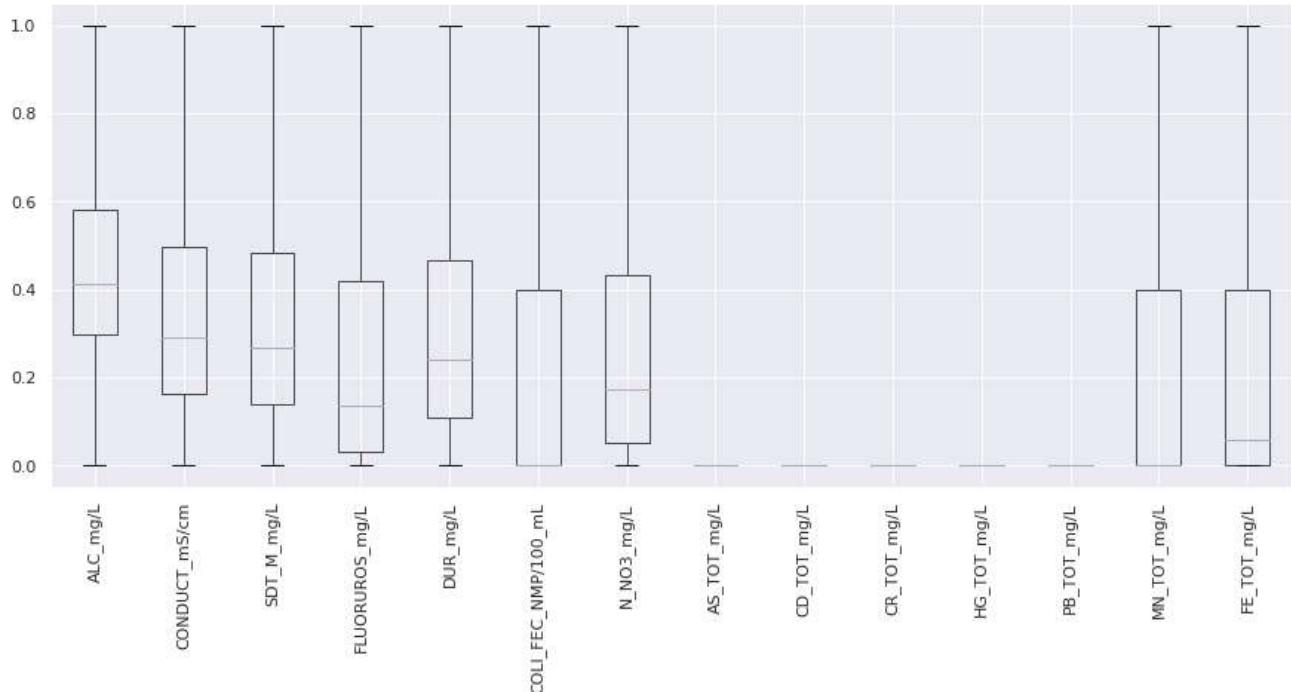
```
data[numericas] = scaler.fit_transform(data[numericas])
```

## Graficamos resultados despues de normalizar

```
sns.set(rc={'figure.figsize':(15,6)})
```

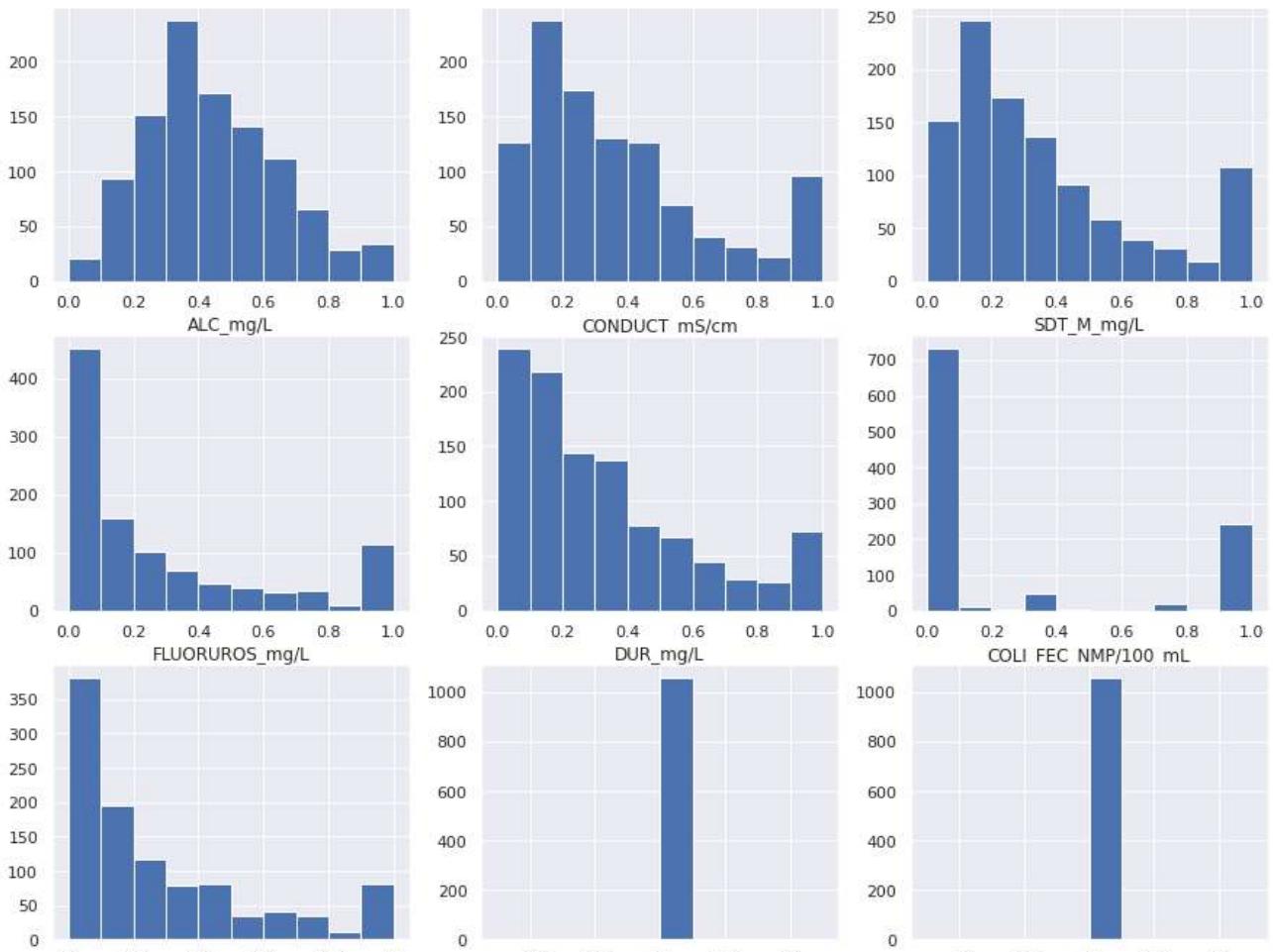
```
data[numericas].boxplot(rot=90)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2ce3e97510>
```



Corremos las gráficas de dispersión pero sin outliers

```
sns.set(rc={'figure.figsize':(15,16)})
fig, axes = plt.subplots(4, 3)      # Definimos una ventana de 5x4 nichos para incluir en cada fila
for k in range(0,12):
    plt.subplot(4,3,k+1)          # Los nichos para cada histograma se numeran iniciando en 1.
    plt.hist(data[data[numericas].columns[k]], bins=10)        # X_train.columns nos devuelve un array con los nombres de las columnas
    plt.xlabel(data[numericas].columns[k])
plt.show()
```



```

for k in variablesBool:
    data[k + ' BOOL'] = data[k].map({
        'SI': True,
        'NO': False,
    })
    data[[s + " BOOL" for s in variablesBool]]

```

```
CUMPLE_CON_ALC  CUMPLE_CON_COND  CUMPLE_CON_SDT_ra  CUMPLE_CON_SDT_salin  CUMP  
      BOOL          BOOL           BOOL           BOOL          BOOL
```

Podemos observar que en general la calidad de agua en todo el pais se encuentra en semáforo naranja

```
media = data["SEMAFORO NUMERO"].mean()  
mediana = data["SEMAFORO NUMERO"].median()  
moda = data["SEMAFORO NUMERO"].mode()  
print("Media:", media)  
print("Mediana:", mediana)  
print("Moda:", moda)  
  
Media: 1.8273244781783682  
Mediana: 2.0  
Moda: 0    1  
dtype: int64  
    1000          1000          1000          1000          1000  
std = data["SEMAFORO NUMERO"].std(ddof=0)  
var = data["SEMAFORO NUMERO"].var(ddof=0)  
rango = data["SEMAFORO NUMERO"].max() - data["SEMAFORO NUMERO"].min()  
iqr = data["SEMAFORO NUMERO"].quantile(0.75) - data["SEMAFORO NUMERO"].quantile(0.25)  
cv = data["SEMAFORO NUMERO"].std(ddof=0) / data["SEMAFORO NUMERO"].mean()  
  
print("Desviación estandar", std)  
print("Varianza", var)  
print("Coeficiente de variación", cv)  
  
Desviación estandar 0.7795859937549438  
Varianza 0.6077543216588832  
Coeficiente de variación 0.4266270183892579
```

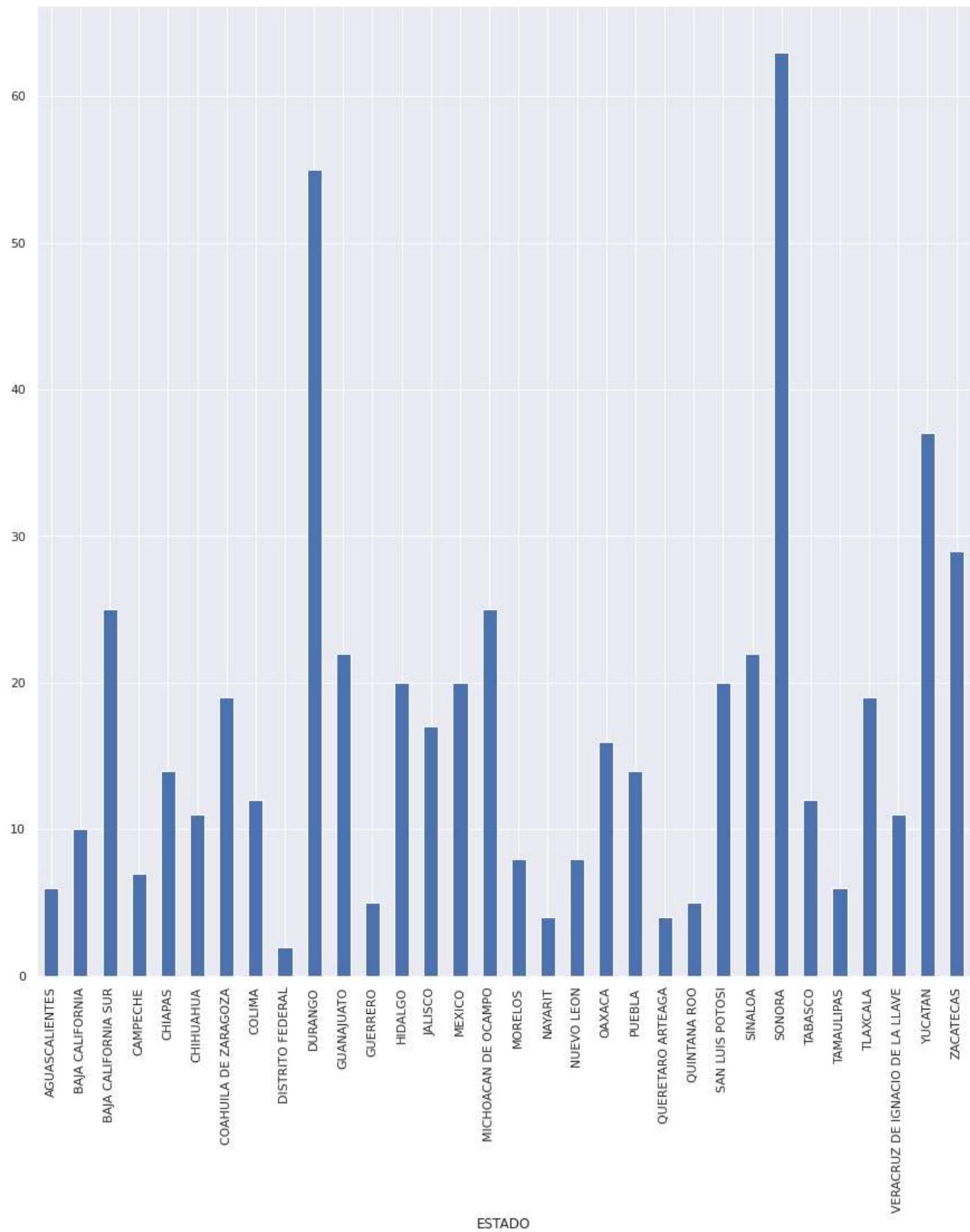
Determinamos el agua es de calidad asegurandonos que cumple con todo lo solicitado

```
data["CALIDAD"] = data[[s + " BOOL" for s in variablesBool]].all(1)  
data["CALIDAD"].unique()  
  
array([ True, False])
```

Determinamos el Estado con mejor calidad de agua

```
data.groupby(['ESTADO'])["CALIDAD"].sum().plot(kind='bar')
```

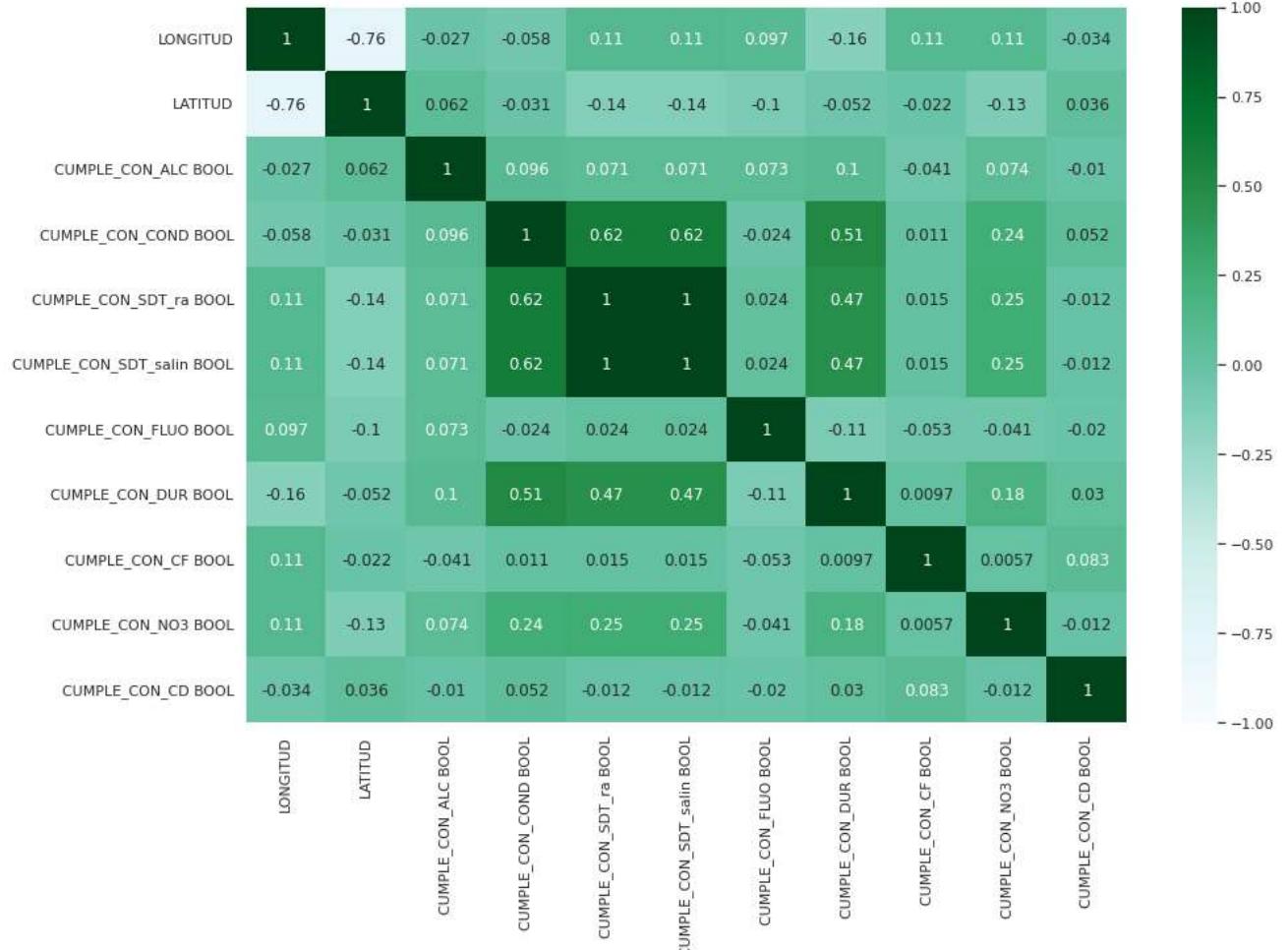
<matplotlib.axes.\_subplots.AxesSubplot at 0x7f2ce40494d0>



Mostramos la correlación entre longitud, latitud y los valores de cumplimiento de calidad del agua. Podemos determinar que no existe mucha correlación entre la ubicación y la calidad del agua.

```
corrs = data[variables + [s + " BOOL" for s in variablesBool]].corr()
sns.set(rc = {'figure.figsize':(15,10)})
sns.heatmap(corrs, vmin = -1, vmax = 1, cmap = "BuGn", annot= True)
```

<matplotlib.axes.\_subplots.AxesSubplot at 0x7f2ce338a450>



Mostramos un mapa de la replica con cada pozo y su semáforo de calidad

```
data["Coordinates"] = list(zip(data["LONGITUD"], data["LATITUD"]))
data["Coordinates"] = data["Coordinates"].apply(Point)
```

```

gdf = gpd.GeoDataFrame(data, geometry="Coordinates")

%matplotlib inline

import qeds
qeds.themes.mpl_style();

world = gpd.read_file(gpd.datasets.get_path("naturalearth_lowres"))
world = world.set_index("iso_a3")

fig, gax = plt.subplots(figsize=(10,10))

world.query("name == 'Mexico'").plot(ax = gax, edgecolor='black', color='white')

colors = {1:'green', 2:'red', 3:'orange'}

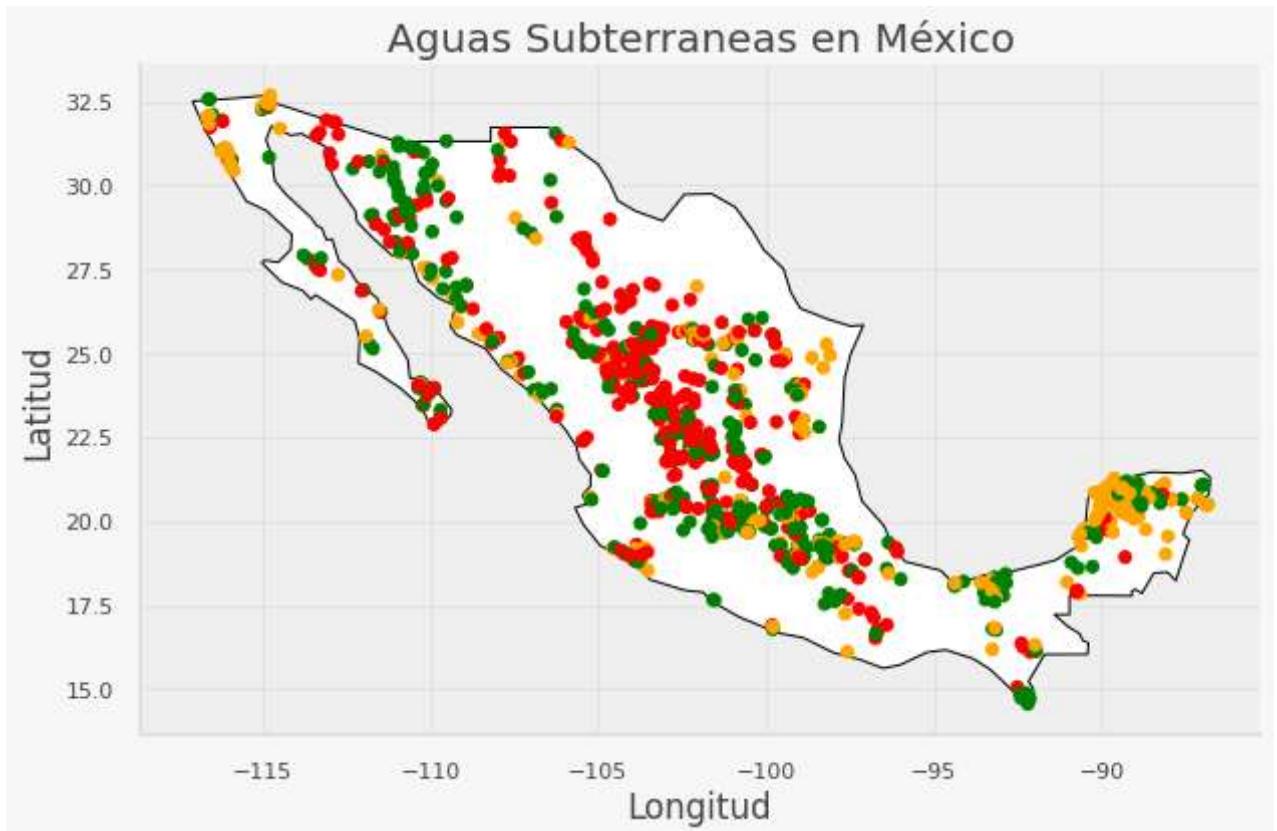
gdf.plot(ax=gax, c=data['SEMAFORO NUMERO'].map(colors), alpha = 1)

gax.set_xlabel('Longitud')
gax.set_ylabel('Latitud')
gax.set_title('Aguas Subterraneas en México')

gax.spines['top'].set_visible(False)
gax.spines['right'].set_visible(False)

plt.show()

```



```

l df = data[['LATITUD', 'LONGITUD', 'SEMAFORO NUMERO']]
wcss=[]
for i in range(1,10):
    kmeans = KMeans(i)

```

```

kmeans.fit(ldf)
wcss_iter = kmeans.inertia_
wcss.append(wcss_iter)

number_clusters = range(1,10)
plt.plot(number_clusters,wcss)
plt.xlabel('Numero de centros de gravedad')
plt.ylabel('WCSS')

Text(0, 0.5, 'WCSS')

```



## Parte 1 - Clusters y centros de gravedad de acuerdo a la ubicación

```
gravityDF = pd.DataFrame(kmeans.cluster_centers_, columns = ['Latitude','Longitude', 'Sema
```

### Visualización de Kmeans

```

Cluster_nums = range(1,10)
kmeans = [KMeans(n_clusters=i) for i in Cluster_nums]
Y_axis = data[['LATITUD']]
X_axis = data[['LONGITUD']]
kmean_calc = [kmeans[i].fit(Y_axis).score(Y_axis) for i in range(len(kmeans))]

plt.figure(figsize=(10,5))
plt.plot(Cluster_nums, kmean_calc,color='red', linestyle='dashed', marker='o',
         markerfacecolor='black')
plt.xlabel('Numeros de Clusters')
plt.ylabel('Score')
plt.title('Grafica de Codo')
plt.show()

```



```

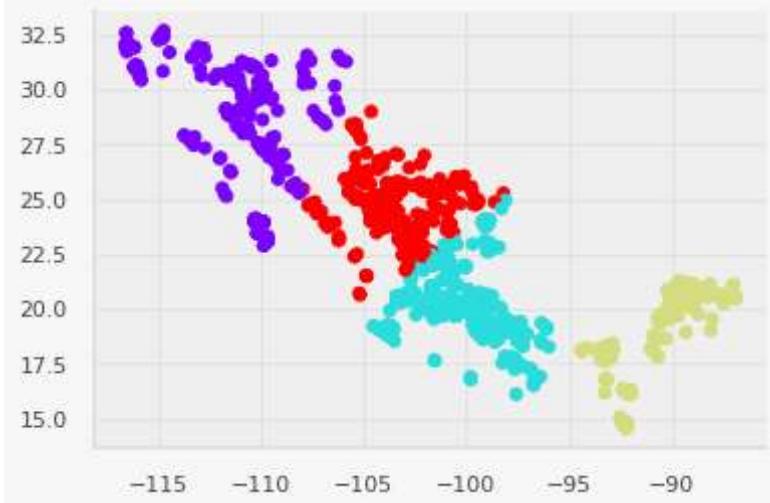
kmeans = KMeans(4)
kmeans.fit(ldf)
identified_clusters = kmeans.fit_predict(ldf)
identified_clusters

array([1, 1, 3, ..., 3, 3, 3], dtype=int32)

data_with_clusters = data.copy()
data_with_clusters['CLUSTERS'] = identified_clusters
plt.scatter(data_with_clusters['LONGITUD'],data_with_clusters['LATITUD'],c=data_with_clust

```

<matplotlib.collections.PathCollection at 0x7f2cddf798d0>



```

%matplotlib inline

import qeds
qeds.themes.mpl_style();

gravityDF["Coordinates"] = list(zip(gravityDF['Longitude'], gravityDF['Latitude']))
gravityDF["Coordinates"] = gravityDF["Coordinates"].apply(Point)
gdf = gpd.GeoDataFrame(gravityDF, geometry="Coordinates")

gravityDF = gpd.GeoDataFrame(gravityDF, geometry="Coordinates")

```

```

world = gpd.read_file(gpd.datasets.get_path("naturalearth_lowres"))
world = world.set_index("iso_a3")

fig, gax = plt.subplots(figsize=(10,10))

world.query("name == 'Mexico'").plot(ax = gax, edgecolor='black', color='white')

colors = {1:'green', 2:'red', 3:'orange'}

gdf = gpd.GeoDataFrame(data, geometry="Coordinates")

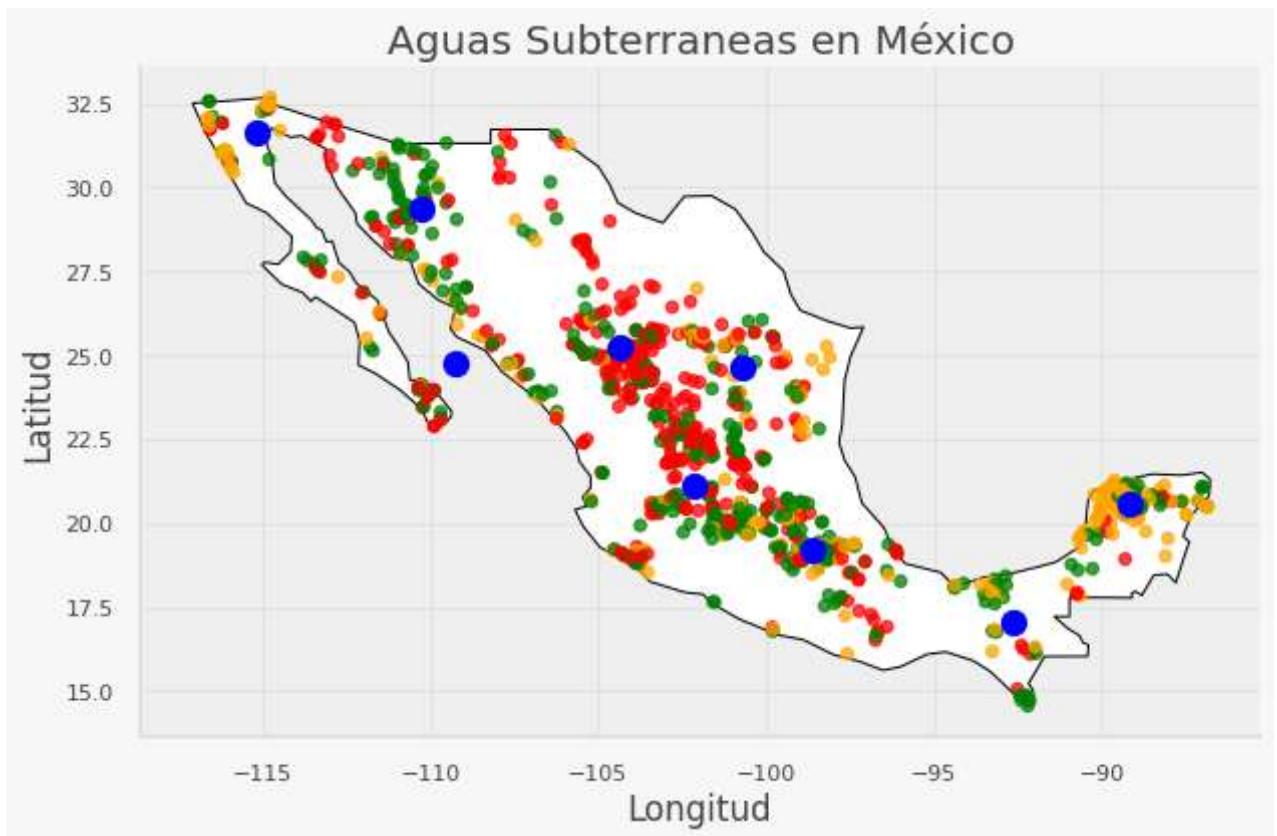
gdf.plot(ax=gax, c=data['SEMAFORO NUMERO'].map(colors), alpha = .75)
gravityDF.plot(ax=gax, color='blue', alpha = 1, markersize=150)

gax.set_xlabel('Longitud')
gax.set_ylabel('Latitud')
gax.set_title('Aguas Subterraneas en México')

gax.spines['top'].set_visible(False)
gax.spines['right'].set_visible(False)

plt.show()

```



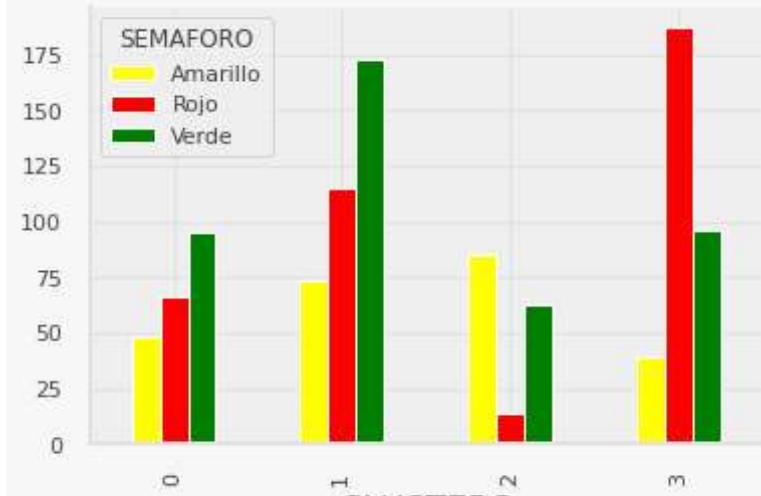
Visualizamos el conteo por semáforo en cada uno de los clusters

```

data_with_clusters_semaforo = data_with_clusters.groupby(['CLUSTERS', 'SEMAFORO']).size()
data_with_clusters_semaforo.plot(kind='bar', color=['yellow', 'red', 'green'])

```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2cddfd810>
```



## Parte 2 - Centros de gravedad de acuerdo a la calidad del agua

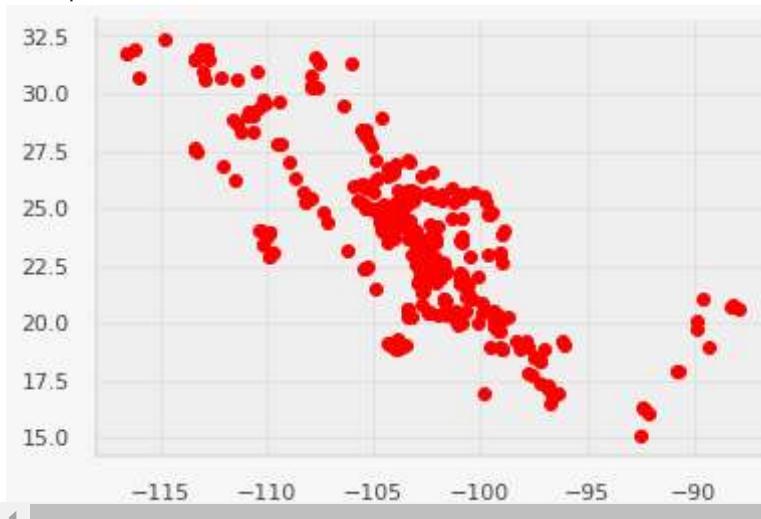
```
redData = data[data['SEMAFORO'] == 'Rojo']
yellowData = data[data['SEMAFORO'] == 'Amarillo']
greenData = data[data['SEMAFORO'] == 'Verde']
```

```
ldf = redData[['LATITUD', 'LONGITUD', 'SEMAFORO NUMERO']]
kmeansRed = KMeans(1)
kmeansRed.fit(ldf)
identified_clustersR = kmeansRed.fit_predict(ldf)
redData['CLUSTERS'] = identified_clustersR
plt.scatter(redData['LONGITUD'], redData['LATITUD'], color='red')
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: SettingWithCopyWarning
  A value is trying to be set on a copy of a slice from a DataFrame.
  Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#inplace-mutation
```

```
<matplotlib.collections.PathCollection at 0x7f2cdddeaf0>
```



```

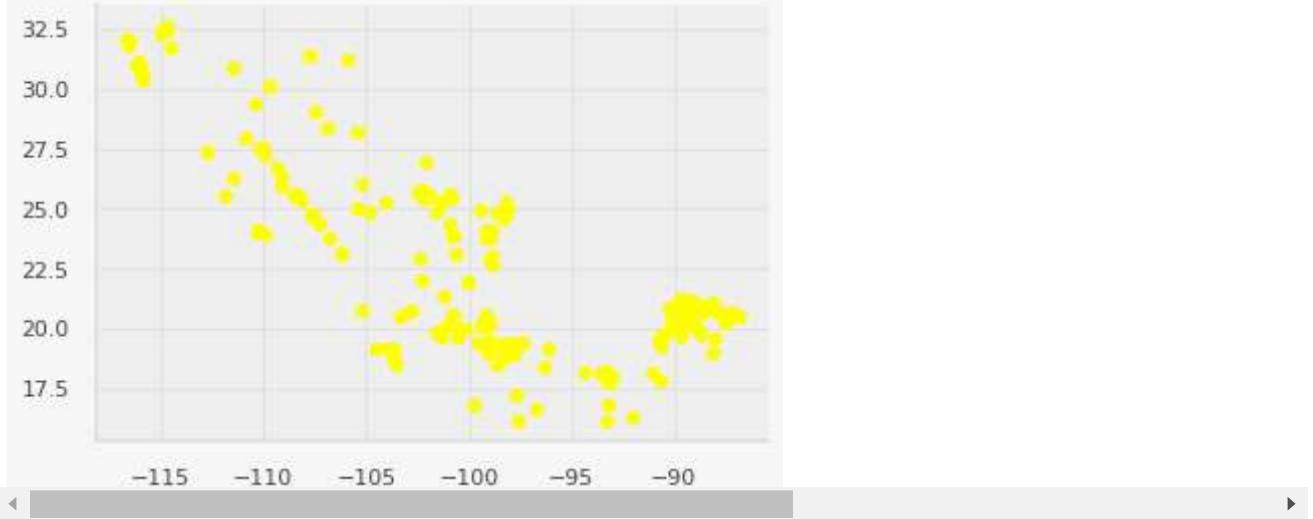
ldf = yellowData[['LATITUD', 'LONGITUD', 'SEMAFORO NUMERO']]
kmeansYellow = KMeans(1)
kmeansRed.fit(ldf)
identified_clustersY = kmeansYellow.fit_predict(ldf)
yellowData['CLUSTERS'] = identified_clustersY
plt.scatter(yellowData['LONGITUD'],yellowData['LATITUD'],color='yellow')

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: SettingWithCopyWarning
  A value is trying to be set on a copy of a slice from a DataFrame.
  Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#inplace-mutation-on-copy-with-settingwithcopywarning

```

<matplotlib.collections.PathCollection at 0x7f2cddd94690>



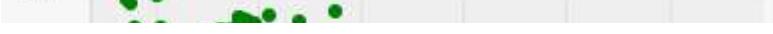
```

ldf = greenData[['LATITUD', 'LONGITUD', 'SEMAFORO NUMERO']]
kmeansGreen = KMeans(1)
kmeansGreen.fit(ldf)
identified_clustersG = kmeansGreen.fit_predict(ldf)
greenData['CLUSTERS'] = identified_clustersG
plt.scatter(greenData['LONGITUD'],greenData['LATITUD'],color='green')

```

```
/usr/local/lib/python3.7/dist-packages/invkernel_launcher.nv:5: SettingWithCopyWarning
gravityRed = pd.DataFrame(kmeansRed.cluster_centers_, columns = ['Latitude','Longitude', 'Altitude'])
gravityYellow = pd.DataFrame(kmeansYellow.cluster_centers_, columns = ['Latitude','Longitude', 'Altitude'])
gravityGreen = pd.DataFrame(kmeansGreen.cluster_centers_, columns = ['Latitude','Longitude', 'Altitude'])
....
```

Mostramos en el mapa los tres centros de gravedad de cada segmento de calidad del agua



```
%matplotlib inline

import qeds
qeds.themes.mpl_style();

gravityRed["Coordinates"] = list(zip(gravityRed['Longitude'], gravityRed['Latitude']))
gravityRed["Coordinates"] = gravityRed["Coordinates"].apply(Point)
gdfR = gpd.GeoDataFrame(gravityRed, geometry="Coordinates")
gravityRed = gpd.GeoDataFrame(gravityRed, geometry="Coordinates")

gravityYellow["Coordinates"] = list(zip(gravityYellow['Longitude'], gravityYellow['Latitude']))
gravityYellow["Coordinates"] = gravityYellow["Coordinates"].apply(Point)
gdfY = gpd.GeoDataFrame(gravityYellow, geometry="Coordinates")
gravityYellow = gpd.GeoDataFrame(gravityYellow, geometry="Coordinates")

gravityGreen["Coordinates"] = list(zip(gravityGreen['Longitude'], gravityGreen['Latitude']))
gravityGreen["Coordinates"] = gravityGreen["Coordinates"].apply(Point)
gdfG = gpd.GeoDataFrame(gravityGreen, geometry="Coordinates")
gravityGreen = gpd.GeoDataFrame(gravityGreen, geometry="Coordinates")

world = gpd.read_file(gpd.datasets.get_path("naturalearth_lowres"))
world = world.set_index("iso_a3")

fig, gax = plt.subplots(figsize=(10,10))

world.query("name == 'Mexico'").plot(ax = gax, edgecolor='black', color='white')

gravityRed.plot(ax=gax, color='red', alpha = 1, markersize=150)
gravityYellow.plot(ax=gax, color='yellow', alpha = 1, markersize=150)
gravityGreen.plot(ax=gax, color='green', alpha = 1, markersize=150)

gax.set_xlabel('Longitud')
gax.set_ylabel('Latitud')
gax.set_title('Aguas Subterráneas en México')

gax.spines['top'].set_visible(False)
gax.spines['right'].set_visible(False)

plt.show()
```



Podemos concluir que es muy util comprender este tipo de interacciones que hay entre diferentes variables y de que forma se pueden usar para graficar o incluso, establecer su ubicación geografica.

Como en este caso analizando la ubicación geografica de los mantos acuíferos, determinar los clusters de la calidad del agua. Nos damos cuenta ahora que con K-means, acerca las ubicaciones geográficas por sus cercanías a los centroides, a diferencia de las ubicaciones identificadas con semáforos de contaminantes.

Con la grafica de codos, nos percatamos que al establecer 3 clusters aun estaría por abajo del rendimiento deseado, aun en el 4to cluster hay un crecimiento considerablemente importante.