Nombre y matricula:

Diego Alonso Luna Ramirez - A01793035

Josep Romagosa Llorden - A01374637

Equipo 112

Nombre del trabajo:

Reto-> Entrega 2 (18/11) -> Clasificación-ensambles y presentación ejecutiva

Fecha de entrega: 18 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 y 2 -> 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
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whe
Collecting qeds
  Downloading qeds-0.7.0.tar.gz (24 kB)
Collecting fiona
  Downloading Fiona-1.8.22-cp37-cp37m-manylinux2014_x86_64.whl (16.7 MB)
     |████████████████████████████████| 16.7 MB 26.3 MB/s
Collecting geopandas
  Downloading geopandas-0.10.2-py2.py3-none-any.whl (1.0 MB)
     |████████████████████████████████| 1.0 MB 60.3 MB/s
Requirement already satisfied: xgboost in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: gensim in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: folium in /usr/local/lib/python3.7/dist-packages
Collecting pyLDAvis
  Downloading pyLDAvis-3.3.1.tar.gz (1.7 MB)
     |████████████████████████████████| 1.7 MB 12.1 MB/s
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Installing backend dependencies ... done
    Preparing wheel metadata ... done
Requirement already satisfied: descartes in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-package
Collecting quandl
  Downloading Quandl-3.7.0-py2.py3-none-any.whl (26 kB)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (
Collecting quantecon
  Downloading quantecon-0.5.3-py3-none-any.whl (179 kB)
     |████████████████████████████████| 179 kB 68.3 MB/s
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: pyarrow in /usr/local/lib/python3.7/dist-packages
```

```
Requirement already satisfied: openpyxl in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: plotly in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pandas_datareader in /usr/local/lib/python3.7/dist
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: seaborn in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: statsmodels in /usr/local/lib/python3.7/dist-packa
Collecting munch
  Downloading munch-2.5.0-py2.py3-none-any.whl (10 kB)
Collecting cligj>=0.5
  Downloading cligj-0.7.2-py3-none-any.whl (7.1 kB)
Requirement already satisfied: click>=4.0 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: attrs>=17 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packa
Collecting click-plugins>=1.0
  Downloading click_plugins-1.1.1-py2.py3-none-any.whl (7.5 kB)
Requirement already satisfied: six>=1.7 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages
Collecting pyproj>=2.2.0
  Downloading pyproj-3.2.1-cp37-cp37m-manylinux2010_x86_64.whl (6.3 MB)
     |████████████████████████████████| 6.3 MB 45.5 MB/s
Requirement already satisfied: shapely>=1.6 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.
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-pac
Requirement already satisfied: jinja2>=2.9 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-
```

```
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
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
```

```
data = pd.read_csv('https://raw.githubusercontent.com/PosgradoMNA/actividades-del-proj
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
3	DLAGU9	POZO EL SALITRILLO	LERMA SANTIAGO PACIFICO	AGUASCALIENTES	RINCON DE ROMOS
4	DLBAJ107	RANCHO EL TECOLOTE	PENINSULA DE BAJA CALIFORNIA	BAJA CALIFORNIA SUR	LA PAZ

5 rows × 57 columns

Analisis de Variables y Limpieza de Base de Datos

```
data.describe()
```

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

Analizamos el nombre de cada columna.

```
data.info()
```

```
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
```

```
 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(5), int64(1), object(51)
memory usage: 475.7+ KB
```

```
variables = ["LONGITUD", "LATITUD", "SEMAFORO", "PERIODO"]
variablesBool = ["CUMPLE_CON_ALC", "CUMPLE_CON_COND", "CUMPLE_CON_SDT_ra", "CUMPLE_CON
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: CLAVE
Cantidad de Valores Unicos: 1068
Tipo de Dato: object
Ejemplo: ['DLAGU6' 'DLAGU6516' 'DLAGU7' 'DLAGU9' 'DLBAJ107' 'DLBAJ108' 'DLBAJ110
 'DLBAJ111' 'DLBAJ117' 'DLBAJ118']
-----------------------------------------------------
Nombre de la Columna: SITIO
Cantidad de Valores Unicos: 1066
Tipo de Dato: object
Ejemplo: ['POZO SAN GIL' 'POZO R013 CAÑADA HONDA' 'POZO COSIO' 'POZO EL SALITRILL
 'RANCHO EL TECOLOTE' 'POZO A.P. CNA 7 (ANTES POZO A.P. CNA 6)'
 'POZO 26, SAN JUAN' 'VICTOR HUGO CESEÑA' 'LAS PARRITAS' 'SAN ANTONIO']
-----------------------------------------------------
Nombre de la Columna: ORGANISMO_DE_CUENCA
Cantidad de Valores Unicos: 13
Tipo de Dato: object
Ejemplo: ['LERMA SANTIAGO PACIFICO' 'LERMA SANTIAGO PACIFICO'
 'LERMA SANTIAGO PACIFICO' 'LERMA SANTIAGO PACIFICO'
 'PENINSULA DE BAJA CALIFORNIA' 'PENINSULA DE BAJA CALIFORNIA'
 'PENINSULA DE BAJA CALIFORNIA' 'PENINSULA DE BAJA CALIFORNIA'
 'PENINSULA DE BAJA CALIFORNIA' 'PENINSULA DE BAJA CALIFORNIA']
-----------------------------------------------------
Nombre de la Columna: ESTADO
```

```
Cantidad de Valores Unicos: 32
Tipo de Dato: object
Ejemplo: ['AGUASCALIENTES' 'AGUASCALIENTES' 'AGUASCALIENTES' 'AGUASCALIENTES'
 'BAJA CALIFORNIA SUR' 'BAJA CALIFORNIA SUR' 'BAJA CALIFORNIA SUR'
 'BAJA CALIFORNIA SUR' 'BAJA CALIFORNIA SUR' 'BAJA CALIFORNIA SUR']
----------------------------------------------------
Nombre de la Columna: MUNICIPIO
Cantidad de Valores Unicos: 452
Tipo de Dato: object
Ejemplo: ['ASIENTOS' 'AGUASCALIENTES' 'COSIO' 'RINCON DE ROMOS' 'LA PAZ' 'LA PAZ
 'LA PAZ' 'LOS CABOS' 'LA PAZ' 'LA PAZ']
----------------------------------------------------
Nombre de la Columna: ACUIFERO
Cantidad de Valores Unicos: 273
Tipo de Dato: object
Ejemplo: ['VALLE DE CHICALOTE' 'VALLE DE CHICALOTE' 'VALLE DE AGUASCALIENTES'
 'VALLE DE AGUASCALIENTES' 'TODOS SANTOS' 'TODOS SANTOS' 'TODOS SANTOS'
 'CABO SAN LUCAS' 'EL CARRIZAL' 'LOS PLANES']
----------------------------------------------------
Nombre de la Columna: SUBTIPO
Cantidad de Valores Unicos: 8
Tipo de Dato: object
Ejemplo: ['POZO' 'POZO' 'POZO' 'POZO' 'POZO' 'POZO' 'POZO' 'POZO' 'POZO' 'POZO']
----------------------------------------------------
Nombre de la Columna: LONGITUD
Cantidad de Valores Unicos: 1066
Tipo de Dato: float64
Ejemplo: [-102.0221   -102.20075  -102.28801  -102.29449  -110.2448   -110.22067
 -110.21396  -109.907306 -110.088778 -110.054722]
----------------------------------------------------
Nombre de la Columna: LATITUD
Cantidad de Valores Unicos: 1067
Tipo de Dato: float64
Ejemplo: [22.20887  21.99958  22.36685  22.18435  23.45138  23.46493  23.4746
 22.8905   23.799861 23.824722]
```

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(['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 complementaría de esa base con nuevas caracteristicas y la clave sea la llave para hacer la unión con estos nuevos datos, pero en este caso, como no esta

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
PERIODO                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
PERIODO                 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: SettingWithCopy
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/stabl
  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
```

```
PERIODO                 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()
```

```
[nan,
 'FLUO,AS,',
 'NO3,',
```

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

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 sería 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
PERIODO                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
PERIODO                 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()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1054 entries, 0 to 1067
Data columns (total 56 columns):
 #   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   PERIODO               1054 non-null   int64  
 9   ALC_mg/L              1054 non-null   float64
 10  CALIDAD_ALC           1054 non-null   object 
 11  CONDUCT_mS/cm         1054 non-null   float64
 12  CALIDAD_CONDUC        1054 non-null   object 
 13  SDT_M_mg/L            1054 non-null   object 
 14  CALIDAD_SDT_ra        1054 non-null   object 
 15  CALIDAD_SDT_salin     1054 non-null   object 
```

```
 16  FLUORUROS_mg/L        1054 non-null   object
 17  CALIDAD_FLUO          1054 non-null   object
 18  DUR_mg/L              1054 non-null   object
 19  CALIDAD_DUR           1054 non-null   object
 20  COLI_FEC_NMP/100_mL   1054 non-null   object
 21  CALIDAD_COLI_FEC      1054 non-null   object
 22  N_NO3_mg/L            1054 non-null   object
 23  CALIDAD_N_NO3         1054 non-null   object
 24  AS_TOT_mg/L           1054 non-null   object
 25  CALIDAD_AS            1054 non-null   object
 26  CD_TOT_mg/L           1054 non-null   object
 27  CALIDAD_CD            1054 non-null   object
 28  CR_TOT_mg/L           1054 non-null   object
 29  CALIDAD_CR            1054 non-null   object
 30  HG_TOT_mg/L           1054 non-null   object
 31  CALIDAD_HG            1054 non-null   object
 32  PB_TOT_mg/L           1054 non-null   object
 33  CALIDAD_PB            1054 non-null   object
 34  MN_TOT_mg/L           1054 non-null   object
 35  CALIDAD_MN            1054 non-null   object
 36  FE_TOT_mg/L           1054 non-null   object
 37  CALIDAD_FE            1054 non-null   object
 38  SEMAFORO              1054 non-null   object
 39  CONTAMINANTES         1054 non-null   object
 40  CUMPLE_CON_ALC        1054 non-null   object
 41  CUMPLE_CON_COND       1054 non-null   object
 42  CUMPLE_CON_SDT_ra     1054 non-null   object
 43  CUMPLE_CON_SDT_salin  1054 non-null   object
 44  CUMPLE_CON_FLUO       1054 non-null   object
 45  CUMPLE_CON_DUR        1054 non-null   object
 46  CUMPLE_CON_CF         1054 non-null   object
 47  CUMPLE_CON_NO3        1054 non-null   object
 48  CUMPLE_CON_AS         1054 non-null   object
 49  CUMPLE_CON_CD         1054 non-null   object
 50  CUMPLE_CON_CR         1054 non-null   object
 51  CUMPLE_CON_HG         1054 non-null   object
 52  CUMPLE CON PB         1054 non-null   object
```

```
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()].ur


def convertValue(columnName, stringValue, numericValue):
  print("\nRemplazando 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[columnName][pd.to_numeric(data[columnName], errors='coerce').isnull()].ur


mgl_list = ['AS_TOT_mg/L','CD_TOT_mg/L','COLI_FEC_NMP/100_mL','CR_TOT_mg/L','DUR_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
```

```
for x in mgl_list:
  convertString(x)
```

```
 AS_TOT_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...
```

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_NO3_mg/L
convertValue('N_NO3_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
```

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

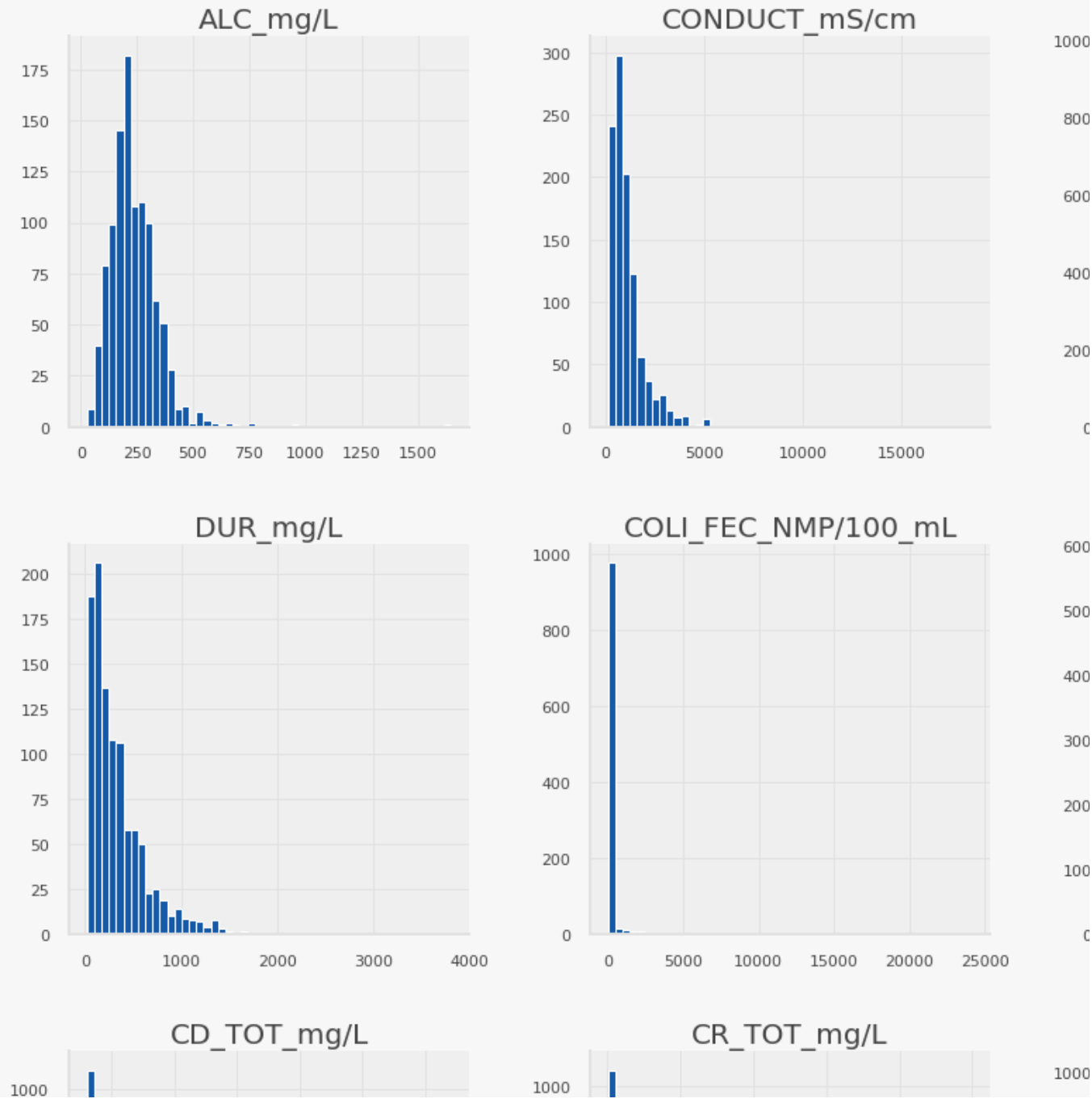
```
#Categoricos
categoricas = ['CALIDAD_CONDUC','CALIDAD_CONDUC','CALIDAD_SDT_ra','CALIDAD_SDT_salin',
#Numericos
numericas = ['ALC_mg/L','CONDUCT_mS/cm','SDT_M_mg/L','FLUORUROS_mg/L','DUR_mg/L','COLI


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

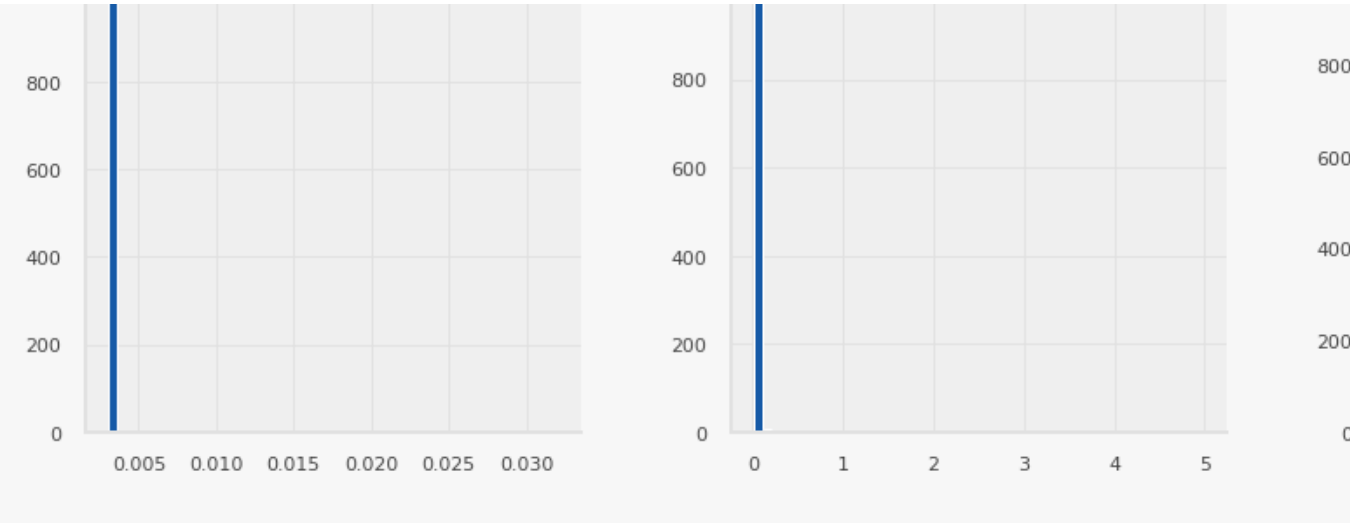
```
 CALIDAD_CONDUC
['Permisible para riego', 'Buena para riego', 'Dudosa para riego', 'Indeseable pa

 CALIDAD_CONDUC
['Permisible para riego', 'Buena para riego', 'Dudosa para riego', 'Indeseable pa

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

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

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

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

 CALIDAD_N_NO3
['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 - Pue

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

Ahora realicemos el histograma basado en los valores numericos.

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

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f8ad4b2dfd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8ad4c63190>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8ad4c59a90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8ad4aa1f90>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f8ad5afd4d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8ad5aee9d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8ad4865f50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8ad61f43d0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f8ad61f4410>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8ad52eea10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8ad4bf1350>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8ad679a850>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f8ad6668d50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8ad494c290>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8ad4925790>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8ad5c8bc90>]],
      dtype=object)
```



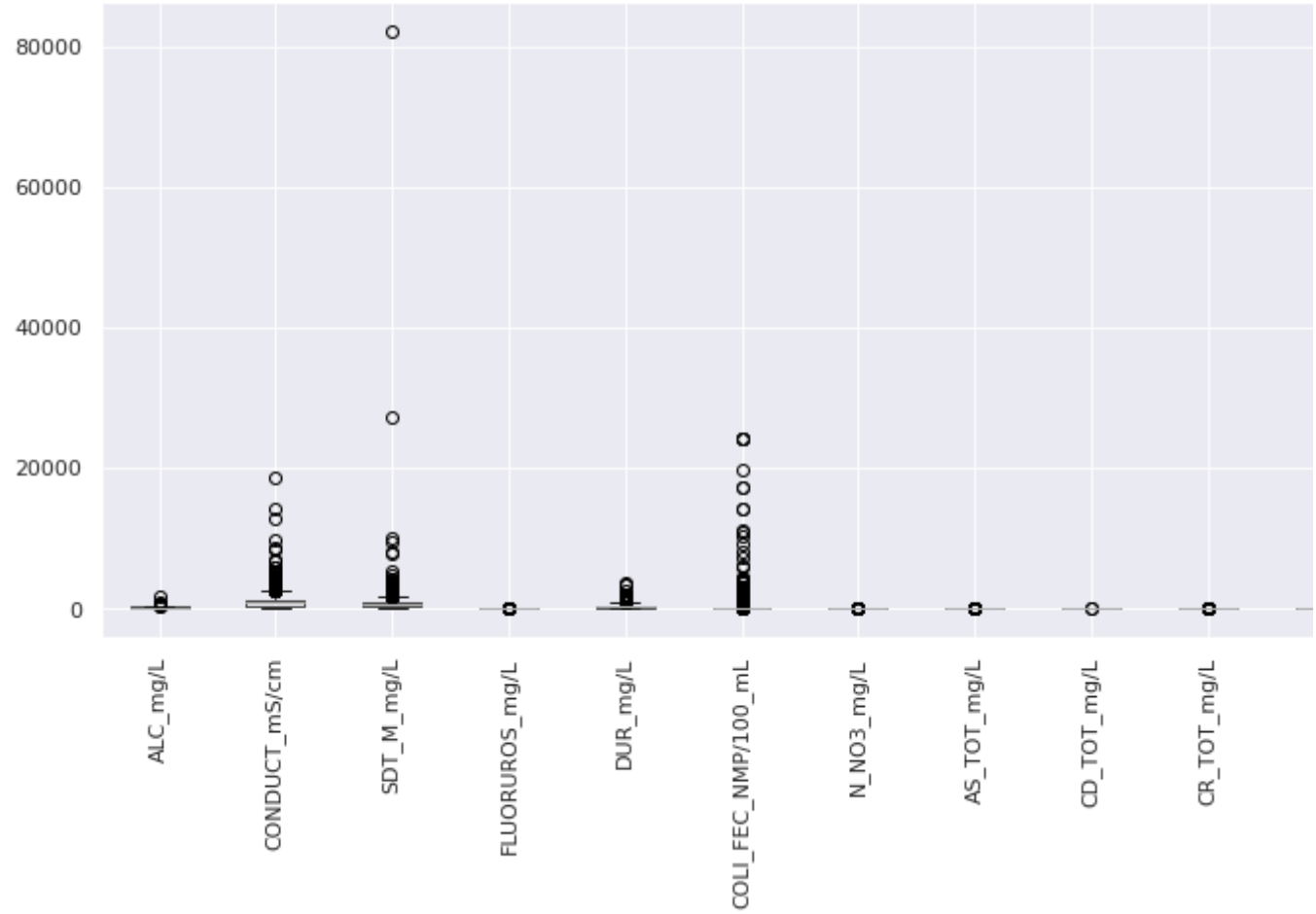


Con la grafica, Conseguimos valores estadísticos como la media, el conteo, la desviación estándar, el valor mímimo, los cuartiles y valor maximo. Esto por cada una de las varibles del dataset.

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 0x7f8ad53e3b10>
```



Identificamos los outliers

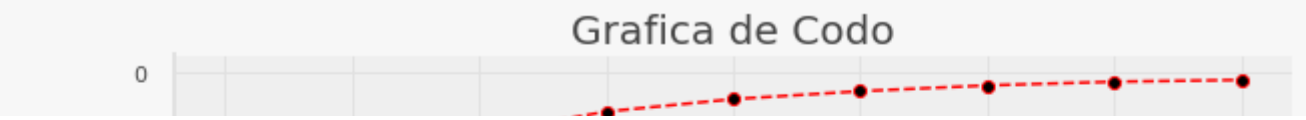
[ ] ↳ *28 cells hidden*

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

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

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()
```

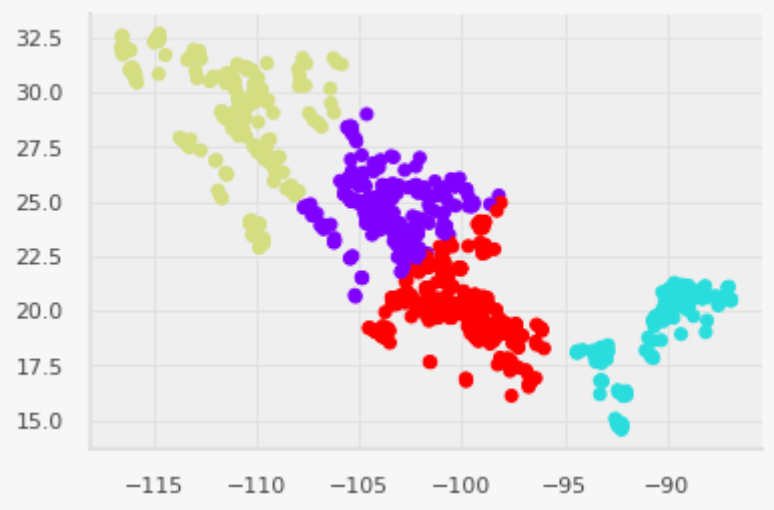



```
kmeans = KMeans(4)
kmeans.fit(ldf)
identified_clusters = kmeans.fit_predict(ldf)
identified_clusters
```

```
array([3, 3, 0, ..., 0, 0, 0], dtype=int32)
```

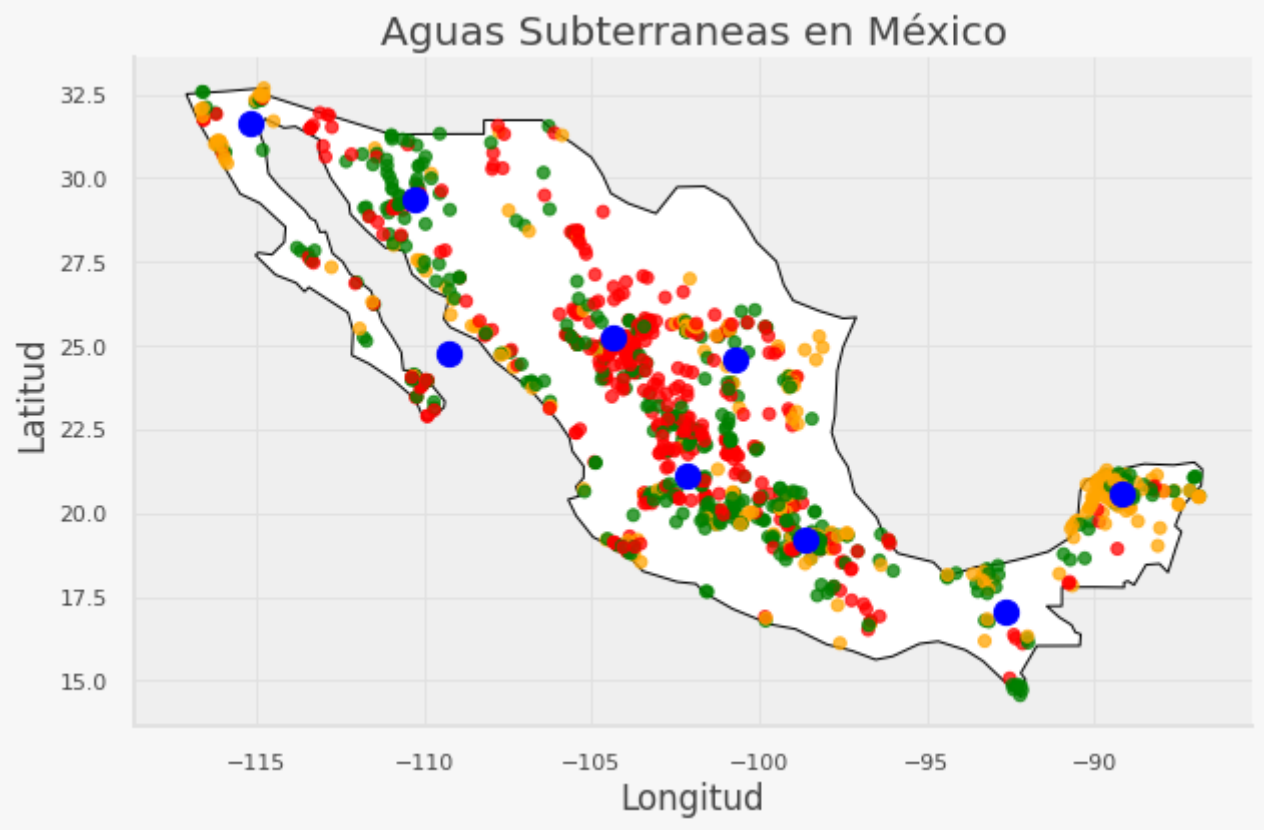
```
data_with_clusters = data.copy()
data_with_clusters['CLUSTERS'] = identified_clusters
plt.scatter(data_with_clusters['LONGITUD'],data_with_clusters['LATITUD'],c=data_with_c
```

```
<matplotlib.collections.PathCollection at 0x7f8ad85a4b10>
```




```
%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'}
```

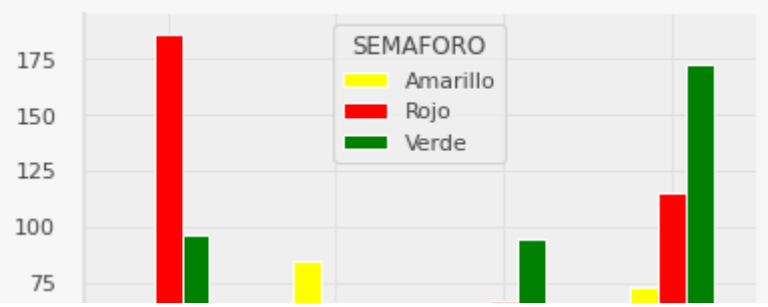
```
gdf = gpd.GeoDataFrame(data, geometry="Coordinates")

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 semaforo en cada uno de los clusters

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8ad84dead0>
```

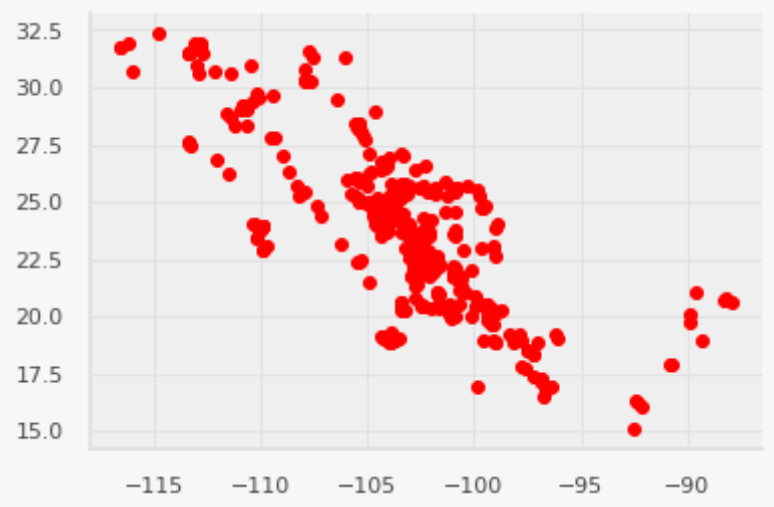


Parte 2 - Centros de gravedad de acuerdo a la calidad del agua

CLUSTERS

```
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')
```

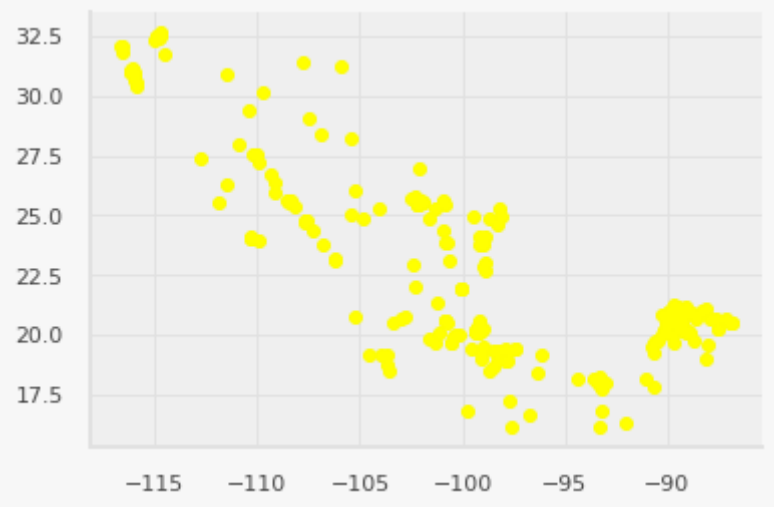
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: SettingWithCopyW
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
  """
<matplotlib.collections.PathCollection at 0x7f8ad67284d0>
```




```
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: SettingWithCopyW
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/stab
  """
<matplotlib.collections.PathCollection at 0x7f8addd07dd0>
```




```
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')
```

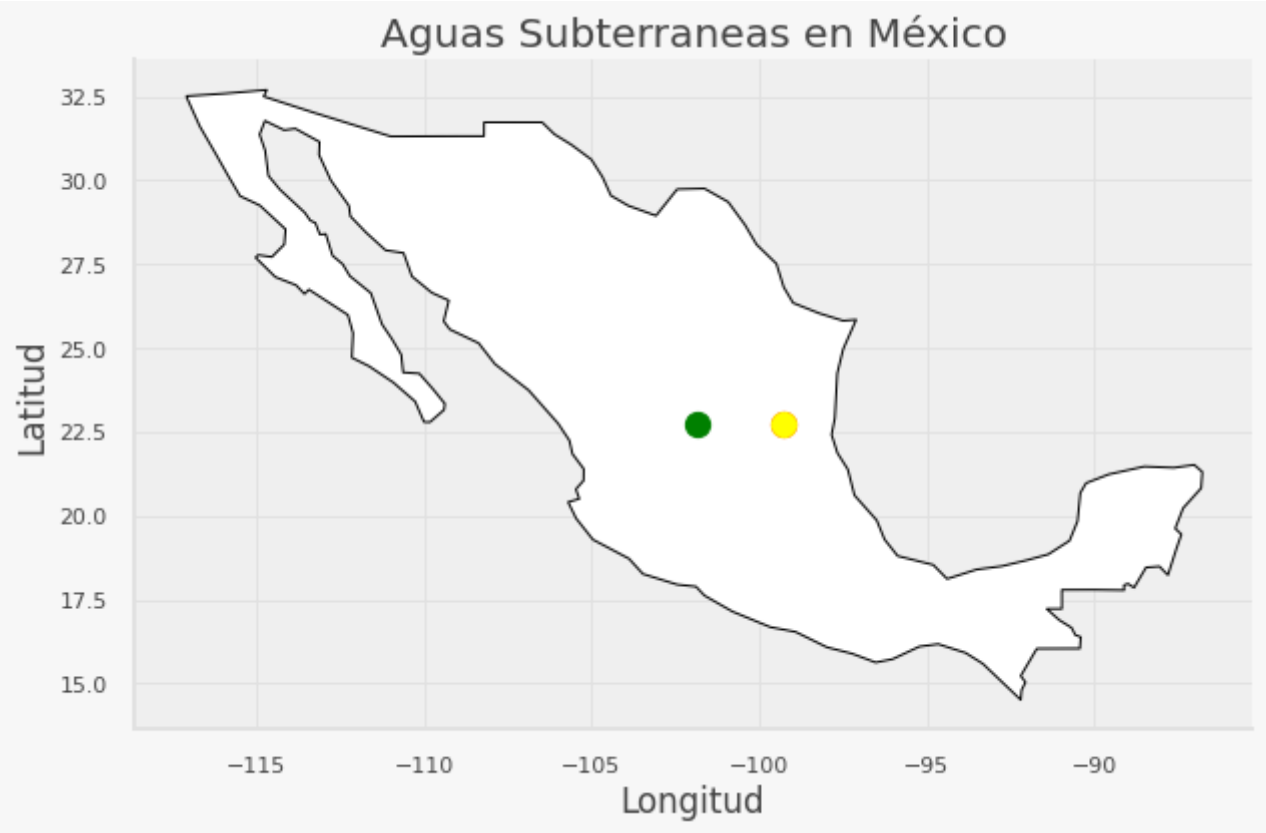
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: SettingWithCopyW
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
  """
```

```
gravityRed = pd.DataFrame(kmeansRed.cluster_centers_, columns = ['Latitude','Longitude
gravityYellow = pd.DataFrame(kmeansYellow.cluster_centers_, columns = ['Latitude','Lor
gravityGreen = pd.DataFrame(kmeansGreen.cluster_centers_, columns = ['Latitude','Longi
```

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['Lat
gravityYellow["Coordinates"] = gravityYellow["Coordinates"].apply(Point)
gdfY = gpd.GeoDataFrame(gravityYellow, geometry="Coordinates")
gravityYellow = gpd.GeoDataFrame(gravityYellow, geometry="Coordinates")

gravityGreen["Coordinates"] = list(zip(gravityGreen['Longitude'], gravityRed['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 Subterraneas 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 interacciónes 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 acuiferos, determinar los clusters de la calidad del agua. Nos damos cuenta ahora que con K-means, acerca las ubicaciones geograficas por sus cercanías a los centroides, a diferencia de las ubicaciones identificadas con semaforos 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.

Random forest

```
data.columns
```

```
Index(['SITIO', 'ORGANISMO_DE_CUENCA', 'ESTADO', 'MUNICIPIO', 'ACUIFERO',
       'SUBTIPO', 'LONGITUD', 'LATITUD', 'PERIODO', 'ALC_mg/L', 'CALIDAD_ALC',
       'CONDUCT_mS/cm', 'CALIDAD_CONDUC', 'SDT_M_mg/L', 'CALIDAD_SDT_ra',
       'CALIDAD_SDT_salin', 'FLUORUROS_mg/L', 'CALIDAD_FLUO', 'DUR_mg/L',
       'CALIDAD_DUR', 'COLI_FEC_NMP/100_mL', 'CALIDAD_COLI_FEC', 'N_NO3_mg/L',
       'CALIDAD_N_NO3', 'AS_TOT_mg/L', 'CALIDAD_AS', 'CD_TOT_mg/L',
       'CALIDAD_CD', 'CR_TOT_mg/L', 'CALIDAD_CR', 'HG_TOT_mg/L', 'CALIDAD_HG',
```

```
       'PB_TOT_mg/L', 'CALIDAD_PB', 'MN_TOT_mg/L', 'CALIDAD_MN', 'FE_TOT_mg/L',
       'CALIDAD_FE', 'SEMAFORO', 'CONTAMINANTES', 'CUMPLE_CON_ALC',
       'CUMPLE_CON_COND', 'CUMPLE_CON_SDT_ra', 'CUMPLE_CON_SDT_salin',
       'CUMPLE_CON_FLUO', 'CUMPLE_CON_DUR', 'CUMPLE_CON_CF', 'CUMPLE_CON_NO3',
       'CUMPLE_CON_AS', 'CUMPLE_CON_CD', 'CUMPLE_CON_CR', 'CUMPLE_CON_HG',
       'CUMPLE_CON_PB', 'CUMPLE_CON_MN', 'CUMPLE_CON_FE', 'SEMAFORO NUMERO',
       'CUMPLE_CON_ALC BOOL', 'CUMPLE_CON_COND BOOL', 'CUMPLE_CON_SDT_ra BOOL',
       'CUMPLE_CON_SDT_salin BOOL', 'CUMPLE_CON_FLUO BOOL',
       'CUMPLE_CON_DUR BOOL', 'CUMPLE_CON_CF BOOL', 'CUMPLE_CON_NO3 BOOL',
       'CUMPLE_CON_CD BOOL', 'CALIDAD'],
      dtype='object')
```

```
dataModel = data[['SEMAFORO NUMERO','LONGITUD','LATITUD']]
```

```
features = pd.get_dummies(dataModel)
features.head(5)
```

	SEMAFORO NUMERO	LONGITUD	LATITUD
0	1	-102.02210	22.20887
1	1	-102.20075	21.99958
2	2	-102.28801	22.36685
3	1	-102.29449	22.18435
4	2	-110.24480	23.45138

```
import numpy as np
labels = np.array(features['SEMAFORO NUMERO'])
features= features.drop('SEMAFORO NUMERO', axis = 1)
feature_list = list(features.columns)
features = np.array(features)
```

Separamos la data

```
train_features, test_features, train_labels, test_labels = train_test_split(features,
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size = 0.25
```

```
print('Training Features Shape:', train_features.shape)
print('Training Labels Shape:', train_labels.shape)
print('Testing Features Shape:', test_features.shape)
print('Testing Labels Shape:', test_labels.shape)
```

```
    Training Features Shape: (790, 2)
    Training Labels Shape: (790,)
```

```
    Testing Features Shape: (264, 2)
    Testing Labels Shape: (264,)
```

```
baseline_preds = test_features[:, feature_list.index('LONGITUD')]
baseline_errors = abs(baseline_preds - test_labels)
print('Promedio de error: ', round(np.mean(baseline_errors), 2))
```

```
    Promedio de error:  103.41
```

```
baseline_preds = test_features[:, feature_list.index('LATITUD')]
baseline_errors = abs(baseline_preds - test_labels)
print('Promedio de error: ', round(np.mean(baseline_errors), 2))
```

```
    Promedio de error:  21.15
```

```
rf = RandomForestRegressor(n_estimators = 1000, random_state = 42)
rf.fit(train_features, train_labels);
```

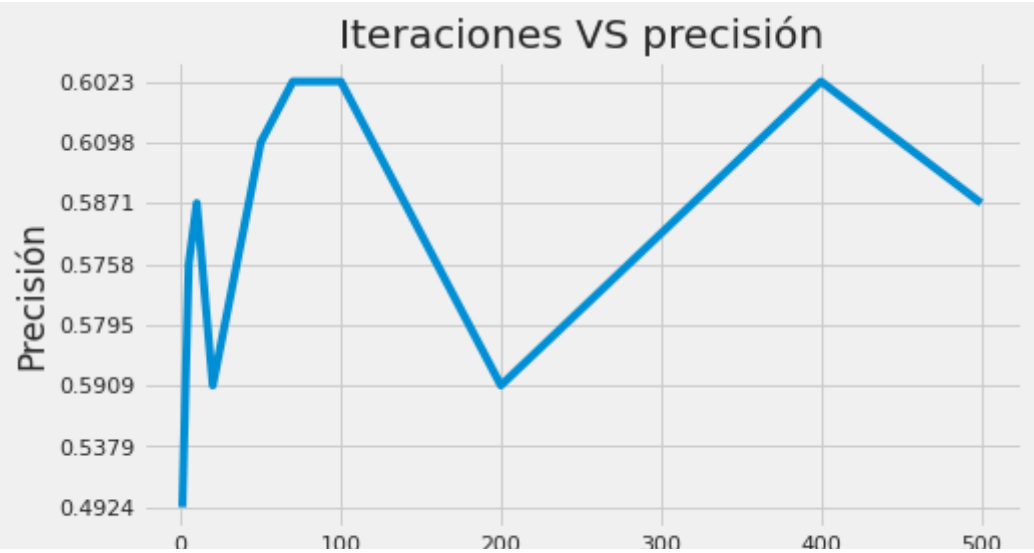
```
predictions = rf.predict(test_features)
errors = abs(predictions - test_labels)
print('Mean Absolute Error:', round(np.mean(errors), 2))
```

```
    Mean Absolute Error: 0.61
```

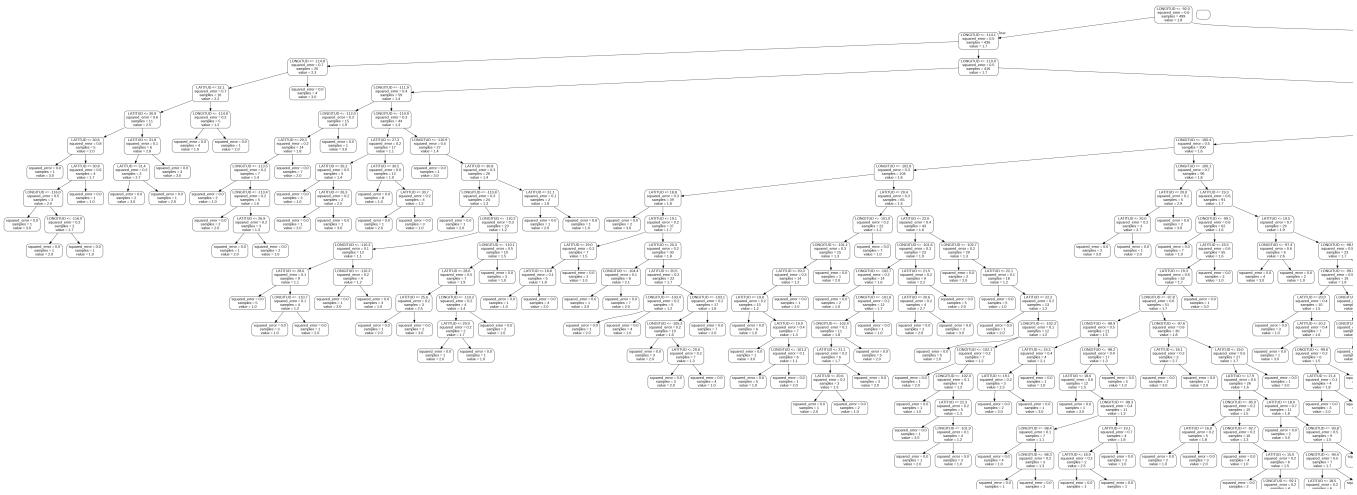
```
mape = 100 * (errors / test_labels)
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
    Accuracy: 57.29 %.
```

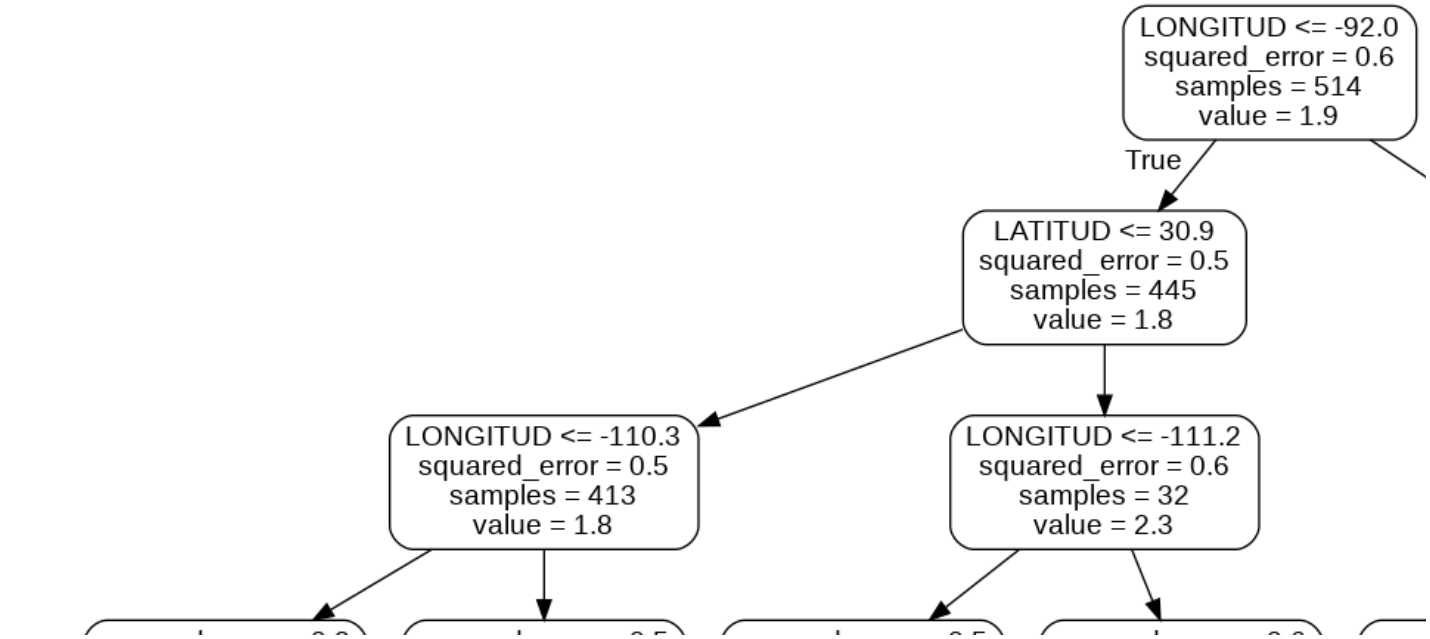
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

number_int = [1,2,3,4,5,10,20,50,70,100,200,400,500]
accuracy = []
for i in number_int:
    rfc = RandomForestClassifier(n_estimators=i, random_state=41)
    rfc.fit(X_train, y_train)
    y_pred = rfc.predict(X_test)

    accuracy.append('{0:0.4f}'.format(accuracy_score(y_test, y_pred)))
fig, ax = plt.subplots(figsize=(7,4))
ax.plot(number_int, accuracy)
ax.title.set_text('Iteraciones VS precisión')
ax.set(xlabel='NIteraciones', ylabel='Precisión')
plt.show()
```




```
from sklearn.tree import export_graphviz
import pydot
tree = rf.estimators_[5]
from sklearn.tree import export_graphviz
import pydot
from IPython.display import Image

tree = rf.estimators_[5]
export_graphviz(tree, out_file = 'tree.dot', feature_names = feature_list, rounded = T
(graph, ) = pydot.graph_from_dot_file('tree.dot')
graph.write_png('tree.png')

Image(filename='tree.png')
```



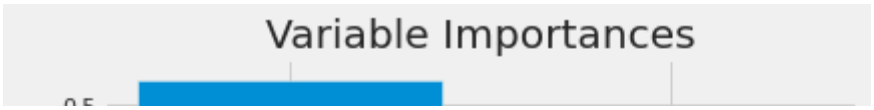
```
rf_small = RandomForestRegressor(n_estimators=10, max_depth = 3)
rf_small.fit(train_features, train_labels)
tree_small = rf_small.estimators_[5]
export_graphviz(tree_small, out_file = 'small_tree.dot', feature_names = feature_list,
(graph, ) = pydot.graph_from_dot_file('small_tree.dot')
graph.write_png('small_tree.png');
Image(filename='small_tree.png')
```




```
importances = list(rf.feature_importances_)
feature_importances = [(feature, round(importance, 2)) for feature, importance in zip(
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_importances
```

```
Variable: LONGITUD             Importance: 0.53
Variable: LATITUD              Importance: 0.47
```

```
import matplotlib.pyplot as plt
%matplotlib inline

plt.style.use('fivethirtyeight')
x_values = list(range(len(importances)))
plt.bar(x_values, importances, orientation = 'vertical')
plt.xticks(x_values, feature_list, rotation='vertical')
plt.ylabel('Importancia'); plt.xlabel('Variable'); plt.title('Importancia de Variables
```



Matriz de confusión

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

cm = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred))

cmap = sns.color_palette("Greys", as_cmap=True)
fig, ax = plt.subplots(figsize=(8, 8))
cm_percent = ((cm/cm.sum(axis=0))*100)
cm_percent = np.nan_to_num(cm_percent, copy=True, nan=0.0, posinf=None, neginf=None)
sns.heatmap(cm_percent, cmap = cmap, annot=True, fmt=".2f", linewidth=1, cbar_kws={"la
ax.set_title('Confusion matrix')
plt.show()
```

	precision	recall	f1-score	support
1	0.62	0.57	0.59	115
2	0.57	0.66	0.61	88
3	0.56	0.51	0.53	61
accuracy			0.59	264
macro avg	0.58	0.58	0.58	264
weighted avg	0.59	0.59	0.59	264