



Tecnológico
de Monterrey

Ciencia Y Analítica De Datos (Gpo 10)
Trimestral Sep-Dic 2022

Profesor Jobish Vallikavungal Devassia

Reto Entrega 2

Ramón Ariel Iván Muñoz Corona

A01330566

Fecha: 17/11/2022

Ramon Muñoz - Ao1330566

Modelo Predictivo del Semáforo de Calidad de Aguas Subterráneas

Agenda

1. Planteamiento del Problema
2. Introducción a la base de datos
3. Insights de los Datos
4. Preparación de datos y generación del modelo
5. Presentación de Resultados
6. Conclusiones

Introducción y Planteamiento del Problema

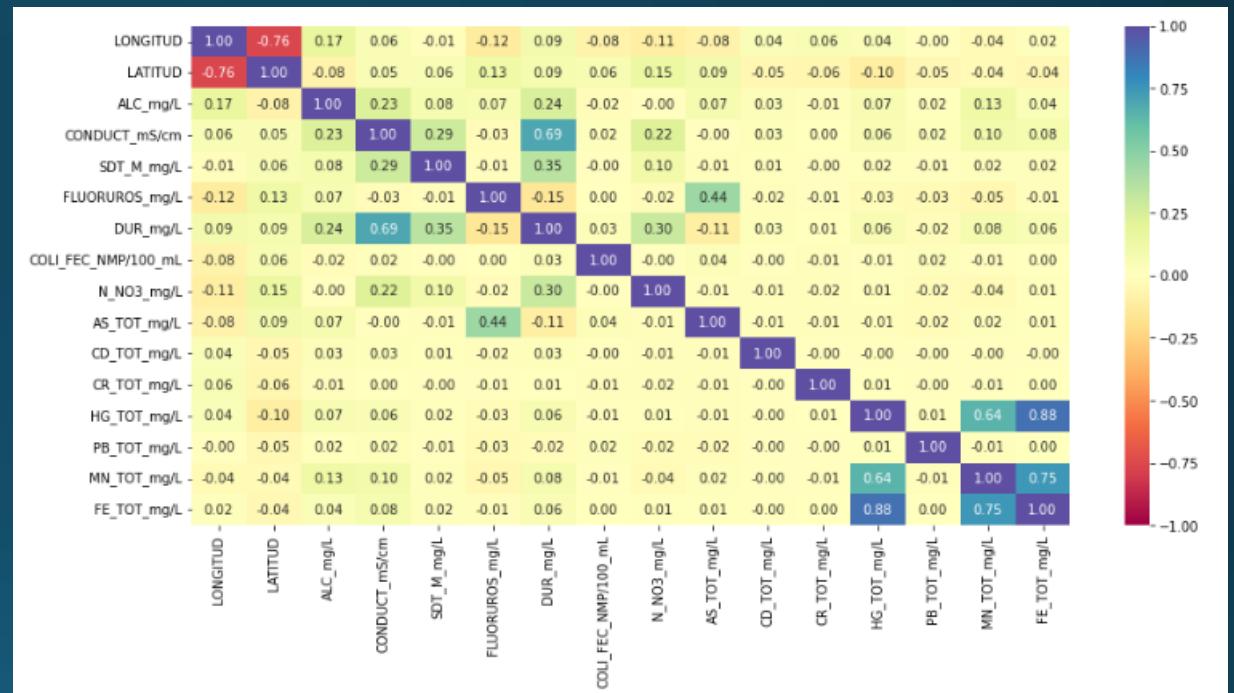
- Para este ejercicio se nos planteo hacer un modelo de clasificación para poder identificar en qué semáforo está el manto acuífero subterráneo
- A lo largo de este ejercicio usamos varias técnicas de limpieza de datos, preprocesamiento y modelado
- El éxito del proyecto será medido por el performance del modelo con los datos de prueba

Introducción a la base de datos

- La base de datos que nos presentan contiene 1068 filas que representan a los mantos y 57 columnas, que representan a sus características
- Dentro de las columnas podemos ver de 4 tipos, en general:
 - Columnas referentes a la localización y organismo que lo maneja (Sitio, Organismo de Cuenca, Estado, Municipio, Acuífero, Latitud, Longitud)
 - Columnas que describen el manto (Clave y Subtipo)
 - Columnas sobre los contaminantes (Contaminantes y columnas de Cantidad del químico)
 - Columnas sobre la calidad (Calidad del químico y si es que cumple con los estándares)
- Como mencionamos en la diapositiva pasada, usaremos el Semáforo salida esperada para generar el modelo predictivo

Insights

- Los features con mayor correlación son:
 - Dur_mg/L vs. CONDUCT_mS/cm (69%)
 - HG_TOT_mg/L vs. FE_TOT_mg/L (88%)
 - MN_TOT_mg/L vs. FE_TOT_mg/L (75%)
 - HG_TOT_mg/L vs. MN_TOT_mg/L (64%)
- Lo podemos observar en la gráfica de Heat Map donde se muestra la correlación



Insights

- La mayor parte de los mantos subterráneos son Pozos (Figura 1)
- El organismo de cuenca con mayor control de los mantos es de las cuencas centrales del norte con 232

Figura 1: Subtipo de Manto

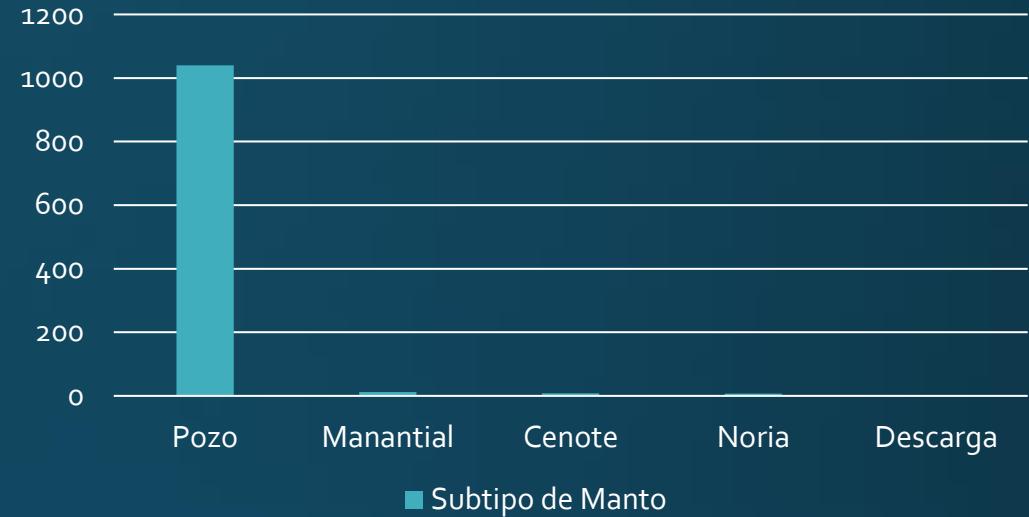


Figura 2: Organismo de Cuenca

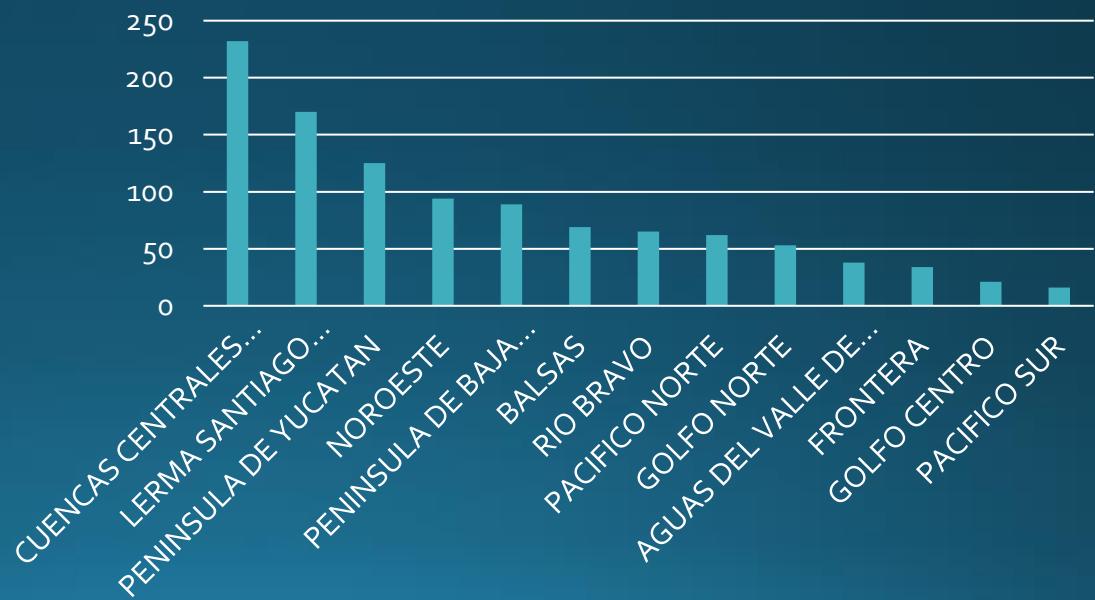
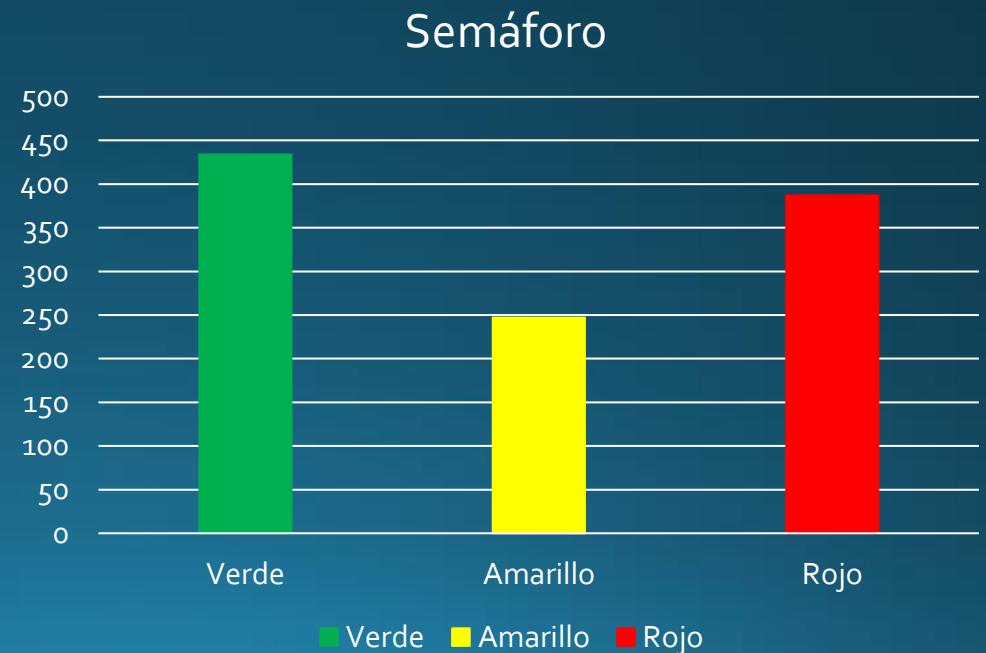
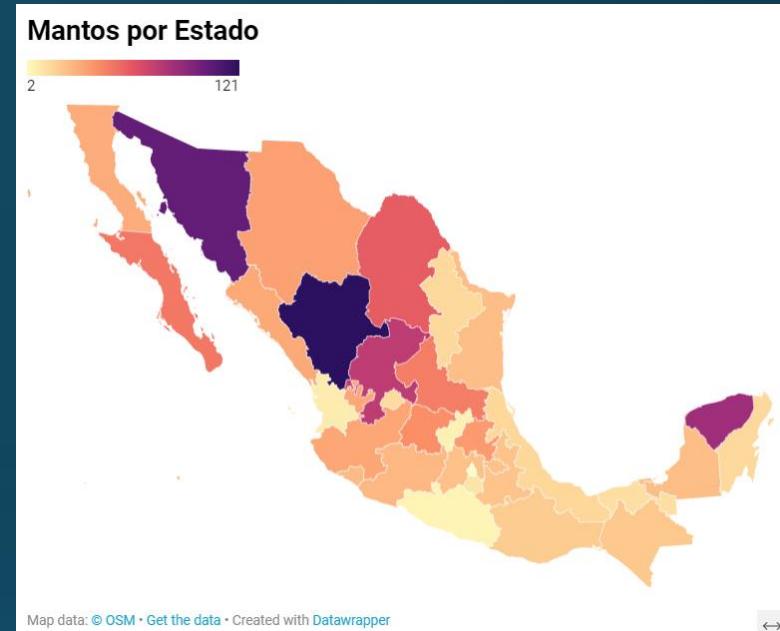


Figura 2: Organismo de Cuenca

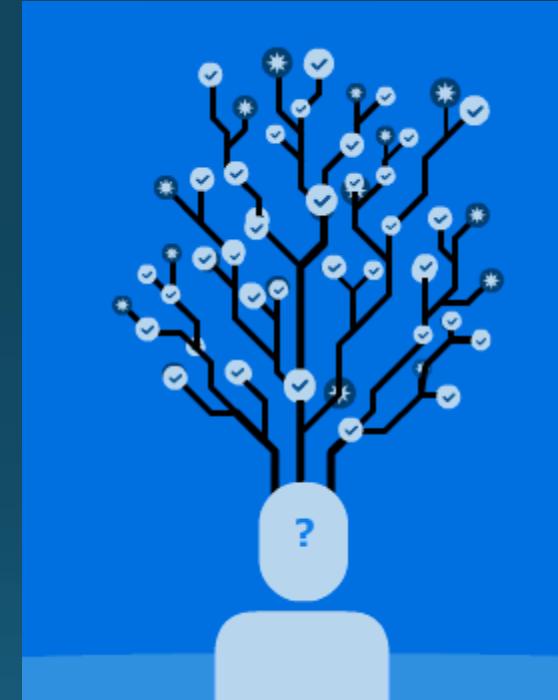
Insights

- El estado donde se encuentran la mayor parte de los mantos subterráneos es Durango (121)
- La mayor parte de los mantos se encuentran en semáforo verde, aunque está relativamente alto también los que se encuentran en rojo



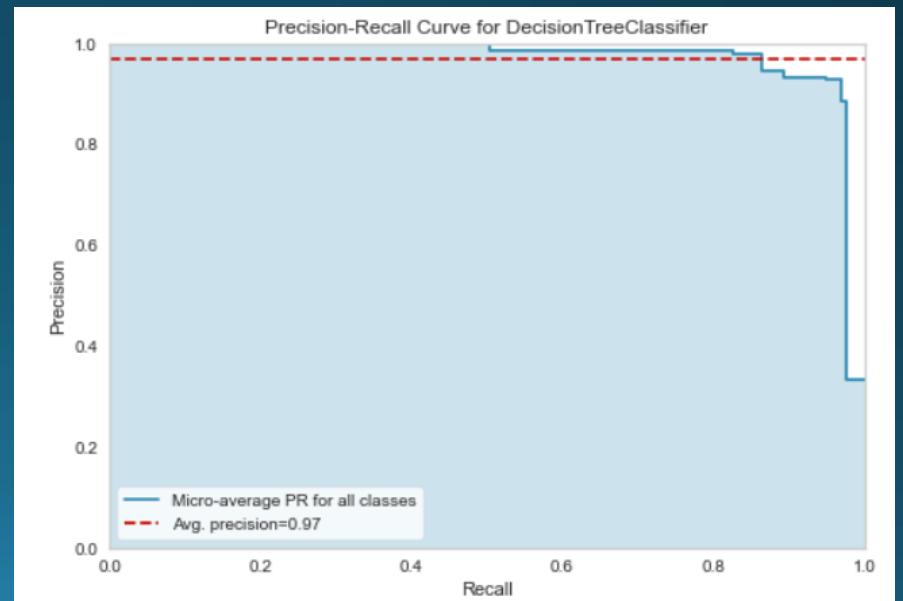
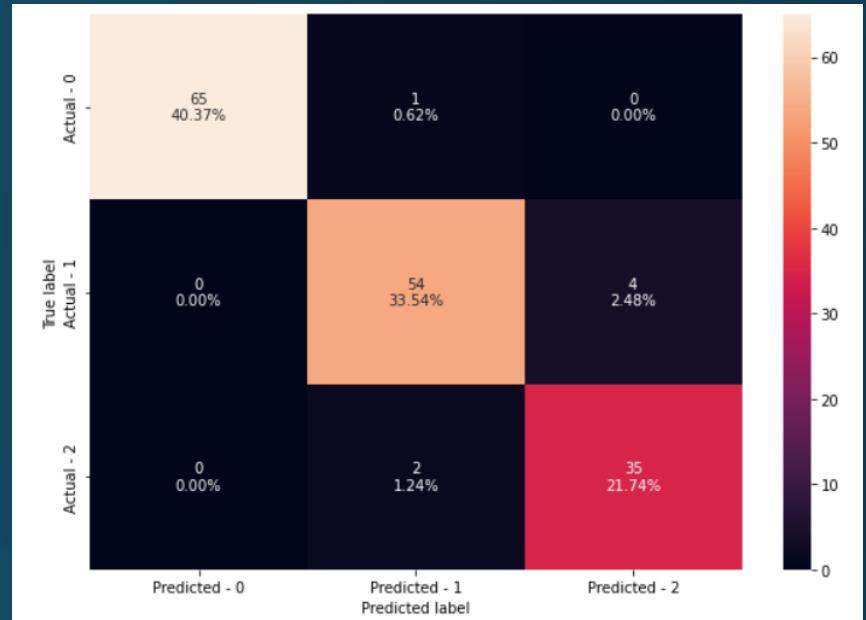
Preparación de datos y generación del modelo

- Para poder utilizar los datos para entrenar al modelo, se tuvo que hacer un preprocessamiento donde se llenaron los valores faltantes, se convirtieron las columnas categóricas y se escalaron las variables.
- Hecho esto, se procedió a generar el modelo
- El modelo que se utilizó fue un Decision Tree Classifier y para obtener los datos más óptimos, se hizo la búsqueda de los hiper parámetros utilizando una técnica llamada Randomized Search
- Y para evaluar al modelo durante el entrenamiento se utilizó el Brier Score Loss
- Al final se generó un modelo con un alto nivel de precisión tanto con los datos de entrenamiento



Resultados

- Como podemos observar en la primera gráfica (Confusion Matrix), existe un mayor número de casos en la diagonal, lo que indica un buen performance del modelo
- Posteriormente, vemos en la curva de Precision Recall, que la curva cubre casi con su totalidad a la gráfica, lo que indica que el nivel de precisión y de recall que presenta es por arriba del 95% con datos de prueba
- Con base en estos resultados podemos asegurar que el modelo generado puede predecir el semáforo con un 95% de precisión



Conclusiones

- Como se mencionó en la diapositiva pasada, el modelo predice con una precisión por arriba del 95% para datos no vistos, lo que lo hace suficientemente confiable para que el cliente pueda utilizarlo
- Probablemente se necesite hacer un análisis con datos de más años para que los resultados puedan extrapolarse, dado que estos datos solo aplican para el 2020
- El éxito del modelo tiene correlación con la técnica que se utilizó para encontrar los hiper parámetros

Reto Entrega 2 - Clasificación-ensambles

Datos de calidad del agua de sitios de monitoreo de aguas subterráneas del 2020

Ramon Muñoz A01330566

Objetivo

Dados los datos de calidad de agua, hacer una limpieza, análisis y generación de un modelo que sea capáz de predecir el semáforo de los mantos acuíferos.

Libraries Import

```
In [1]: import warnings  
  
warnings.filterwarnings("ignore")  
# Libraries to help with reading and manipulating data  
import pandas as pd  
import numpy as np  
  
# Libraries to help with data visualization  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
# Libraries to tune model, get different metric scores, and split data  
from sklearn import metrics  
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score  
from sklearn.preprocessing import LabelEncoder, StandardScaler, OrdinalEncoder  
from sklearn.compose import ColumnTransformer  
from sklearn.metrics import confusion_matrix  
from sklearn.metrics import make_scorer  
from sklearn.preprocessing import OneHotEncoder  
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, StratifiedKFold, RepeatedStratifiedKFold  
  
from sklearn.impute import KNNImputer  
from sklearn.pipeline import Pipeline, make_pipeline  
from imblearn.combine import SMOTETomek  
  
from sklearn.metrics import brier_score_loss  
from sklearn import metrics  
from sklearn.metrics import multilabel_confusion_matrix  
  
#libraries to help with model building  
from sklearn.tree import DecisionTreeClassifier  
  
# to suppress scientific notations  
pd.set_option('display.float_format', lambda x: '%.3f' % x)
```

Lectura de la base de datos

```
In [2]: df = pd.read_csv("Datos_de_calidad_del_agua_de_sitios_de_monitoreo_de_aguas_subterráneas_2020.csv", encoding='utf-8')
```

Visualización de las primeras 5 filas

```
In [3]: df.head()
```

Out[3]:	CLAVE	SITIO	ORGANISMO_DE CUENCA	ESTADO	MUNICIPIO	ACUIFERO	SUBTIPO	LONGITU
0	DLAGU6	POZO SAN GIL	LERMA SANTIAGO PACIFICO	AGUASCALIENTES	ASIENTOS	VALLE DE CHICALOTE	POZO	-102.02
1	DLAGU6516	POZO R013 CAÑADA HONDA	LERMA SANTIAGO PACIFICO	AGUASCALIENTES	AGUASCALIENTES	VALLE DE CHICALOTE	POZO	-102.20
2	DLAGU7	POZO COSIO	LERMA SANTIAGO PACIFICO	AGUASCALIENTES	COSIO	VALLE DE AGUASCALIENTES	POZO	-102.28
3	DLAGU9	POZO EL SALITRILLO	LERMA SANTIAGO PACIFICO	AGUASCALIENTES	RINCON DE ROMOS	VALLE DE AGUASCALIENTES	POZO	-102.29
4	DLBAJ107	RANCHO EL TECOLOTE	PENINSULA DE BAJA CALIFORNIA	BAJA CALIFORNIA SUR	LA PAZ	TODOS SANTOS	POZO	-110.24

5 rows × 57 columns

Entendiendo el Data Set

```
In [4]: print("SHAPE => ", df.shape)
print("INFO:")
print(df.info())
```

```

SHAPE => (1068, 57)
INFO:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1068 entries, 0 to 1067
Data columns (total 57 columns):
 #   Column            Non-Null Count Dtype  
 ---  -- 
 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(5), int64(1), object(51)
memory usage: 475.7+ KB
None

```

- Tenemos 1 Columna que no contiene datos (SDT_mg/L), por lo que la podemos eliminar
- Hay columnas que parecen ser numéricas pero posiblemente tienen datos no numéricos, lo que las hacen en tipo Object

In [5]: df = df.drop("SDT_mg/L", axis = 1)

Primero dividiremos las 57 columnas en categoricas, numericas, ordinales y binarias, para posteriormente poder hacer un analisis un poco mas profundo de los valores de las variables.

```
In [6]: BinCols = ["CUMPLE_CON_ALC", "CUMPLE_CON_COND", "CUMPLE_CON_SDT_ra", "CUMPLE_CON_SDT_salin", "CUMPLE_CON_FLUO", 'CatCols = ["CLAVE", "SITIO", "ORGANISMO_DE_CUENCA", "ESTADO", "MUNICIPIO", "ACUIFERO", "SUBTIPO", "CALIDAD_SDT_sali' NumCols = ["LONGITUD", "LATITUD", "PERIODO", "ALC_mg/L", "CONDUCT_mS/cm", "SDT_M_mg/L", "FLUORUROS_mg/L", "DUR_mg/L' OrdCols = ["CALIDAD_ALC", "CALIDAD_CONDUC", "CALIDAD_SDT_ra", "CALIDAD_FLUO", "CALIDAD_COLI_FEC"]
```

De lo que se alcanza a ver, dentro de las columnas numericas hay valores en formato de texto especificando que son menores a un valor. Esto hace que nuestras variables no puedan ser tomadas como numericas, por lo tanto, se sustituiran por el valor minimo que se establecen.

```
In [7]: rows = df.shape[0]
cols = df.shape[1]

for row in range(0,rows):
    for col in NumCols:
        val = str(df.loc[row,col])
        if '>' in val:
            val = val.replace(">","")
            df.loc[row,col] = float(val)
        elif '<' in val:
            val = val.replace("<","");
            df.loc[row,col] = float(val)
```

Una vez convertidas todos los valores a numeros, procederemos a convertir las columnas a float.

```
In [8]: df[NumCols] = df[NumCols].astype("float")
```

Hecho esto, podemos empezar con un analisis estadistico de las columnas numericas

```
In [9]: df[NumCols].describe().T
```

	count	mean	std	min	25%	50%	75%	max
LONGITUD	1068.000	-101.891	6.703	-116.664	-105.389	-102.174	-98.975	-86.864
LATITUD	1068.000	23.164	3.888	14.561	20.212	22.617	25.510	32.678
PERIODO	1068.000	2020.000	0.000	2020.000	2020.000	2020.000	2020.000	2020.000
ALC_mg/L	1064.000	235.634	116.874	26.640	164.000	215.528	292.710	1650.000
CONDUCT_mS/cm	1062.000	1138.953	1245.564	50.400	501.750	815.000	1322.750	18577.000
SDT_M_mg/L	1066.000	896.102	2751.531	25.000	337.500	550.400	916.100	82170.000
FLUORUROS_mg/L	1068.000	1.076	1.924	0.200	0.267	0.504	1.140	34.803
DUR_mg/L	1067.000	347.938	359.669	20.000	121.195	245.336	453.930	3810.692
COLI_FEC_NMP/100_mL	1068.000	355.490	2052.457	1.100	1.100	1.100	13.250	24196.000
N_NO3_mg/L	1067.000	4.320	8.345	0.020	0.650	2.081	5.202	121.008
AS_TOT_mg/L	1068.000	0.020	0.035	0.010	0.010	0.010	0.010	0.452
CD_TOT_mg/L	1068.000	0.003	0.001	0.003	0.003	0.003	0.003	0.032
CR_TOT_mg/L	1068.000	0.013	0.154	0.005	0.005	0.005	0.005	5.003
HG_TOT_mg/L	1068.000	0.001	0.000	0.001	0.001	0.001	0.001	0.014
PB_TOT_mg/L	1068.000	0.005	0.003	0.005	0.005	0.005	0.005	0.081
MN_TOT_mg/L	1068.000	0.072	0.377	0.002	0.002	0.002	0.010	8.982
FE_TOT_mg/L	1068.000	0.410	5.538	0.025	0.025	0.047	0.173	178.615

Podemos observar que el valor del Periodo es el mismo en todos los casos, asi que podemos eliminar esa columna.

```
In [10]: df = df.drop("PERIODO", axis=1)
NumCols.pop(NumCols.index("PERIODO"))
```

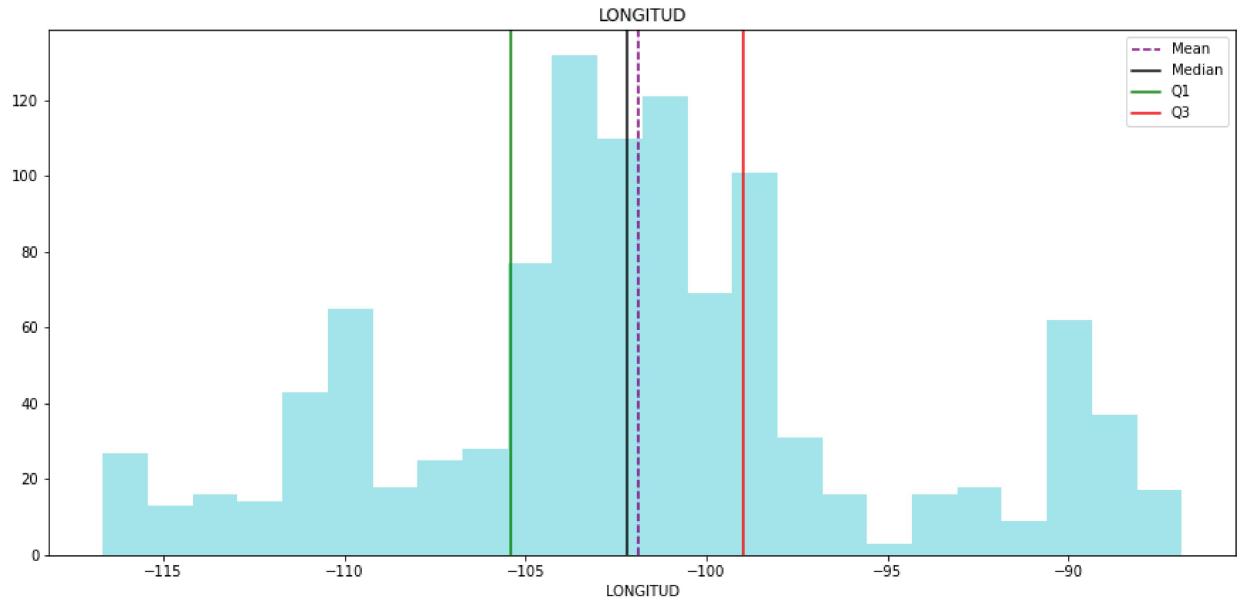
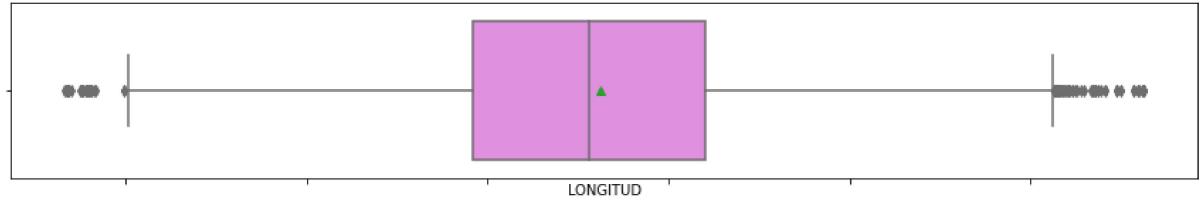
```
Out[10]: 'PERIODO'
```

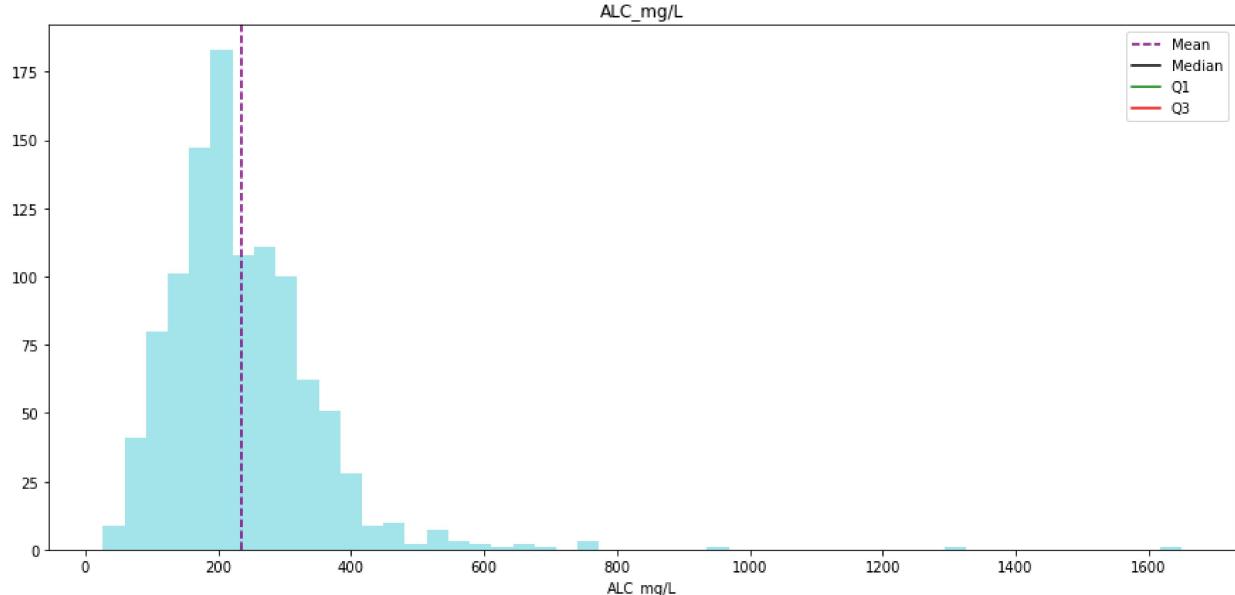
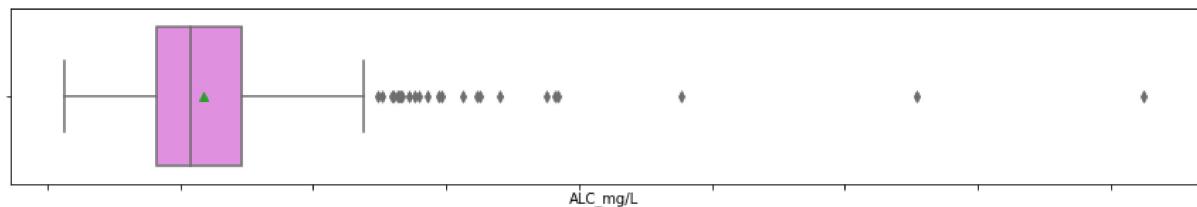
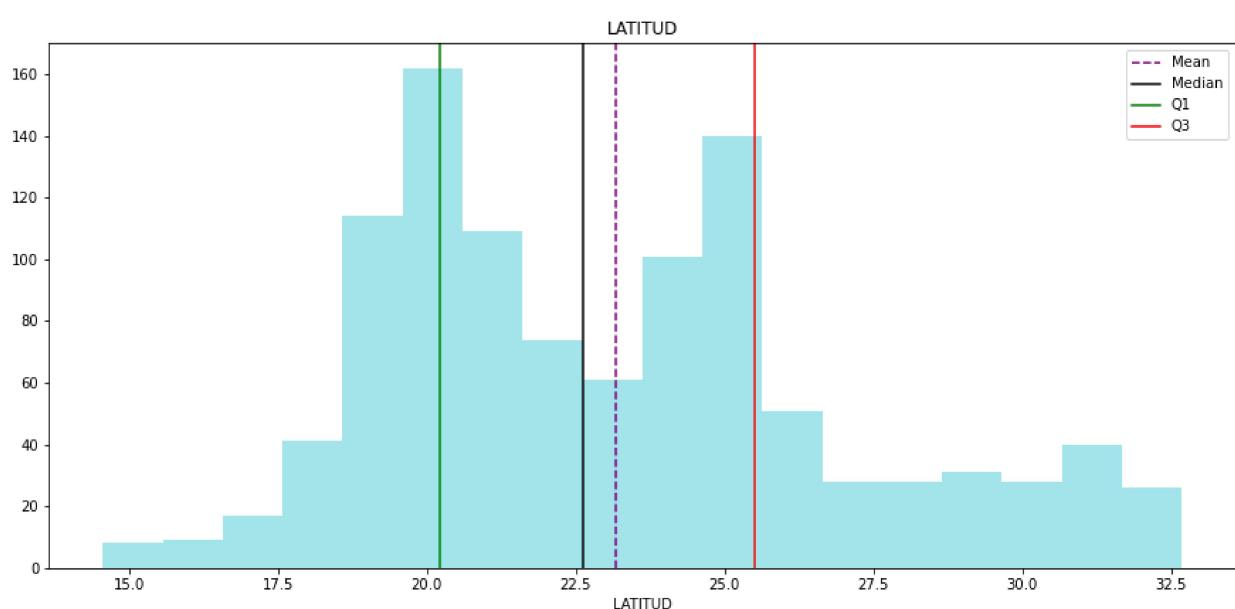
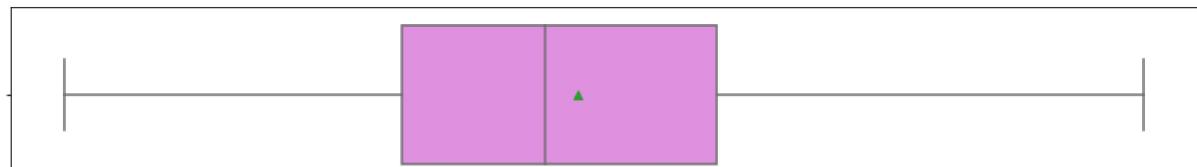
Ademas, podemos ver que todo se encuentra ene un rango normal, por lo que solo deberemos preocuparnos por los valores faltantes de cada columna.

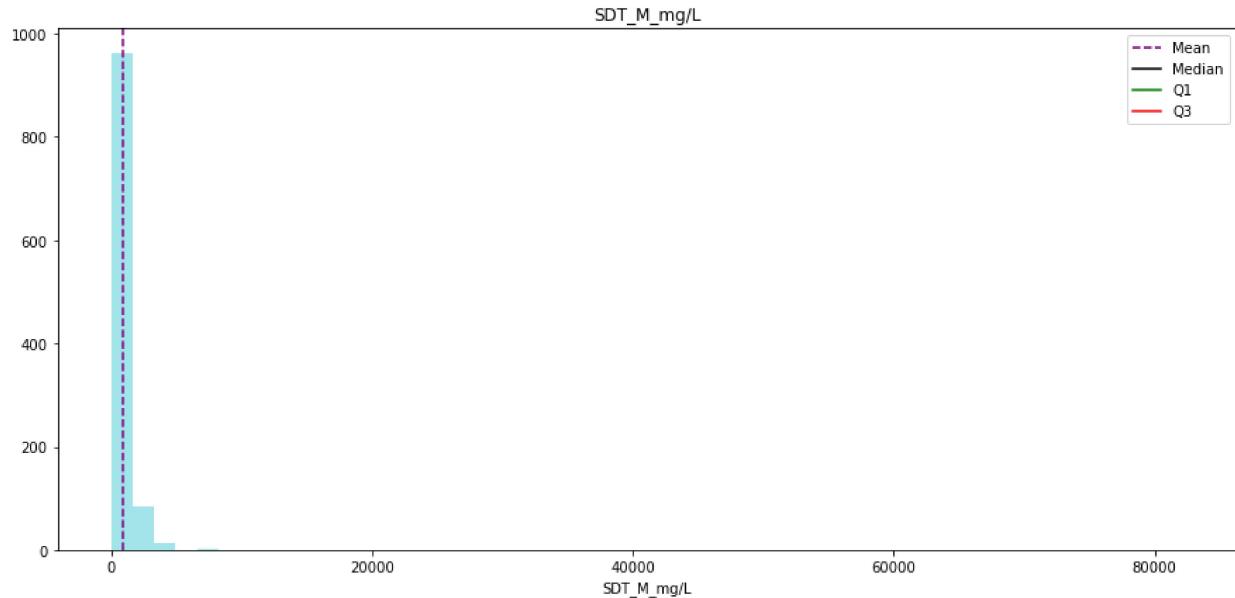
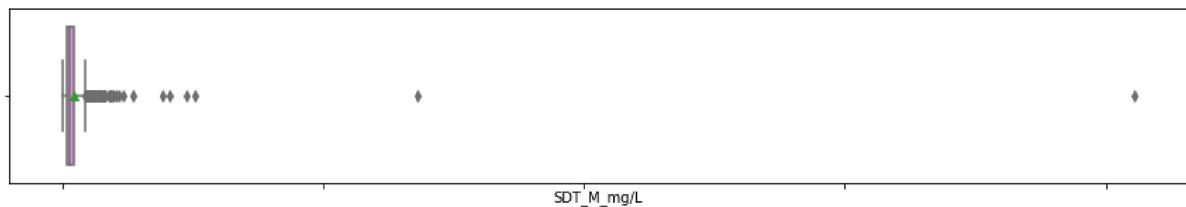
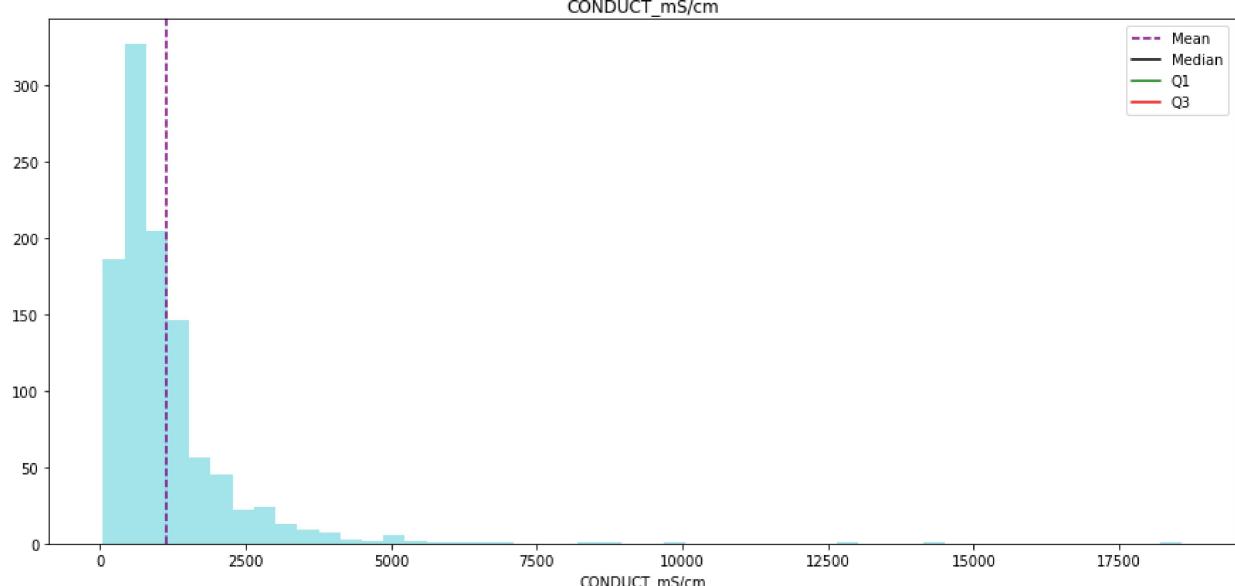
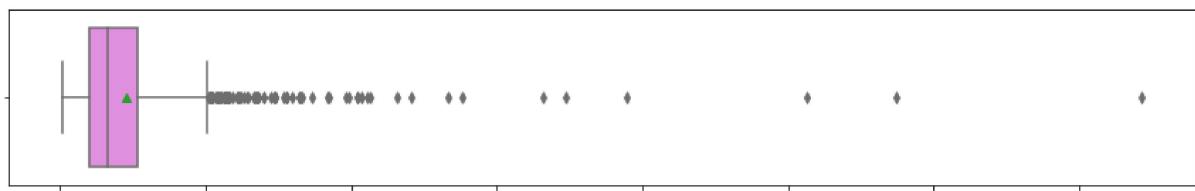
En las siguientes graficas mostraremos tanto los boxplot para mostrar los outliers, como la distribucion de cada columna

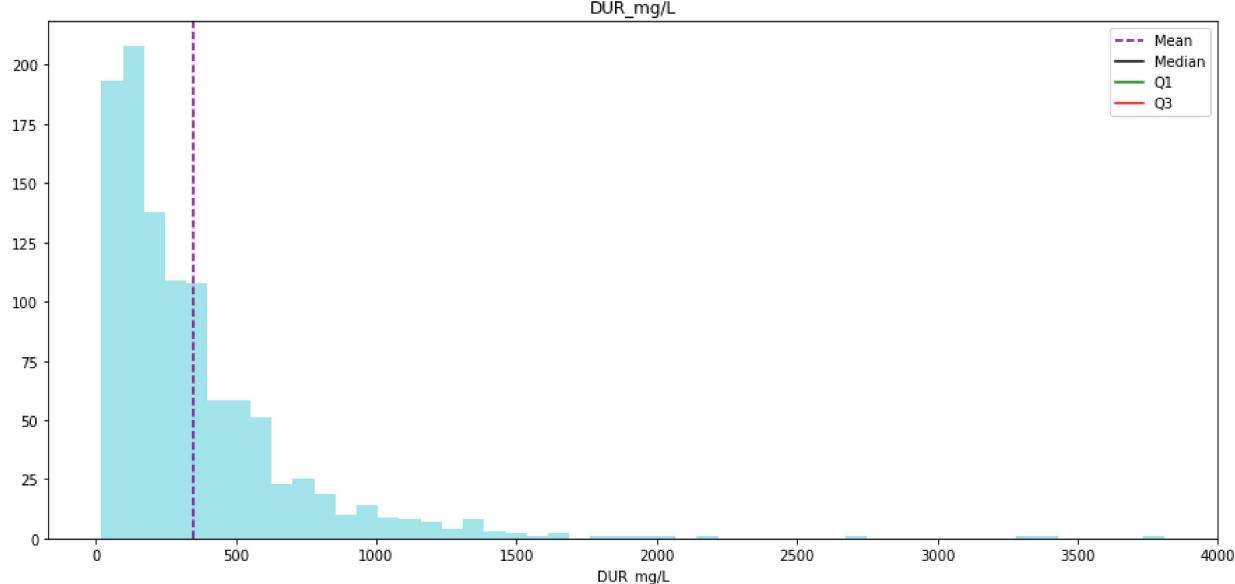
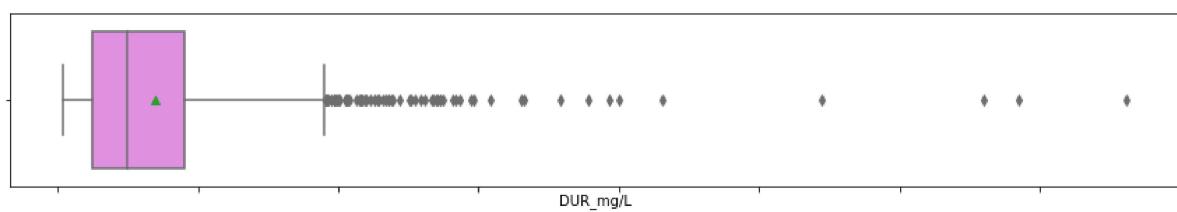
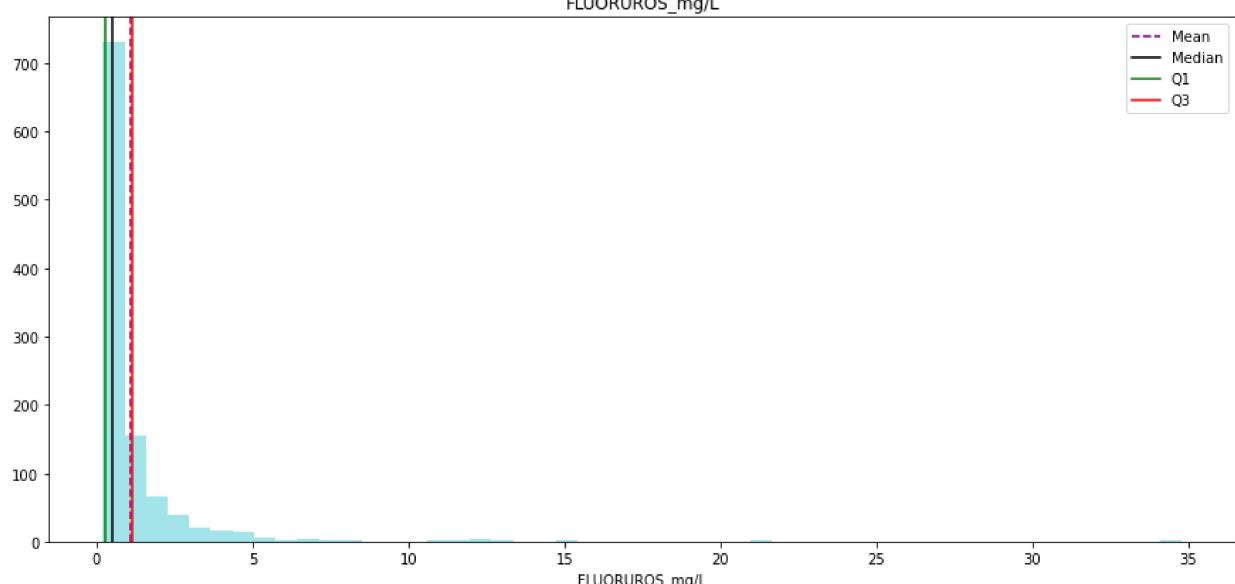
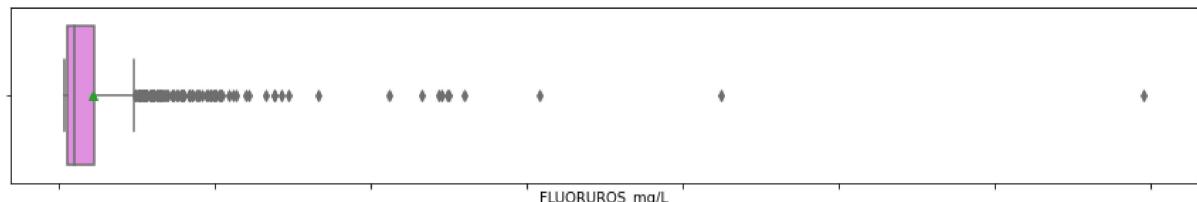
```
In [11]: def histogram_boxplot(dfr,feat, figsize=(15,10), bins = None):
    feature = dfr[feat]
    """ Boxplot and histogram combined
    feature: 1-d feature array
    figsize: size of fig (default (9,8))
    bins: number of bins (default None / auto)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(nrows = 2, # Number of rows of the subplot grid= 2
                                           sharex = True, # x-axis will be shared among all subplots
                                           gridspec_kw = {"height_ratios": (.25, .75)},
                                           figsize = figsize
                                         ) # creating the 2 subplots
    plt.title(feat)
    sns.boxplot(feature, ax=ax_box2, showmeans=True, color='violet') # boxplot will be created and a star will be placed at the mean
    sns.distplot(feature, kde=False, ax=ax_hist2, bins=bins, color = 'orange') if bins else sns.distplot(feature,
    ax_hist2.axvline(np.mean(feature), color='purple', linestyle='--', label = "Mean") # Add mean to the histogram
    ax_hist2.axvline(np.median(feature), color='black', linestyle='-', label = "Median") # Add median to the histogram
    ax_hist2.axvline(np.quantile(feature,0.25), color='green', linestyle='--', label = "Q1") # Add Q1 to the histogram
    ax_hist2.axvline(np.quantile(feature,0.75), color='red', linestyle='--', label = "Q3") # Add Q3 to the histogram
    plt.legend(loc='upper right')
    plt.show()
```

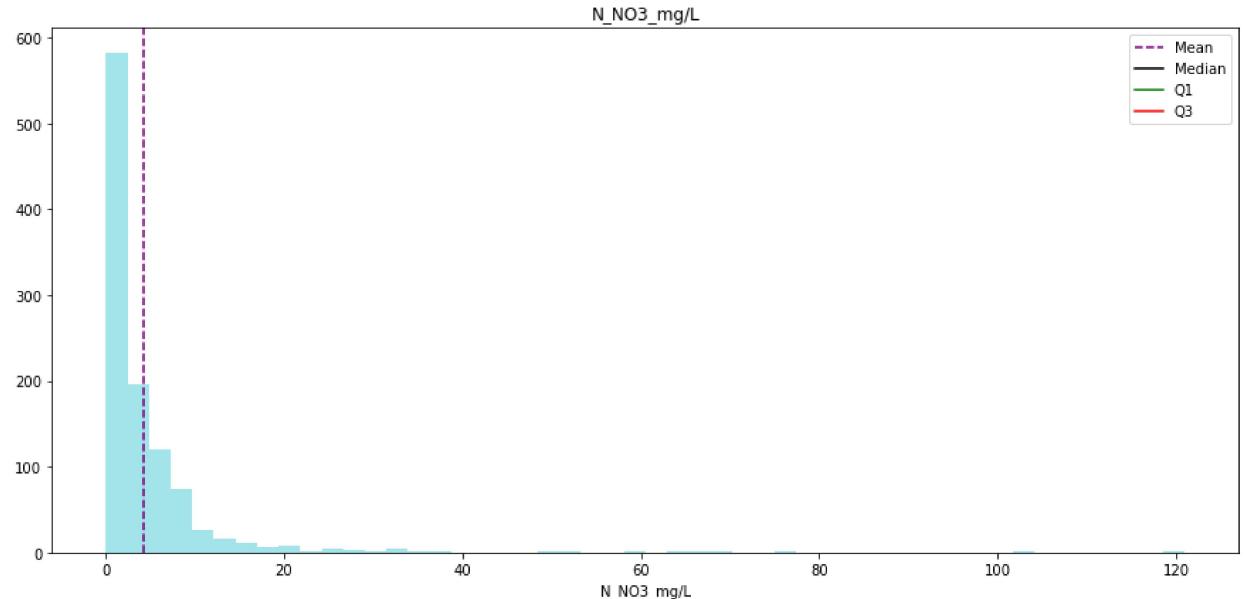
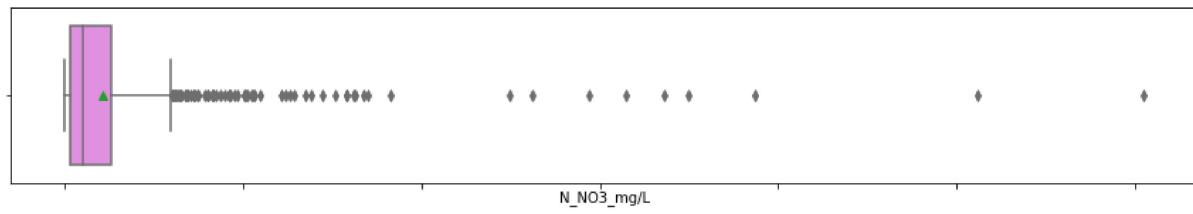
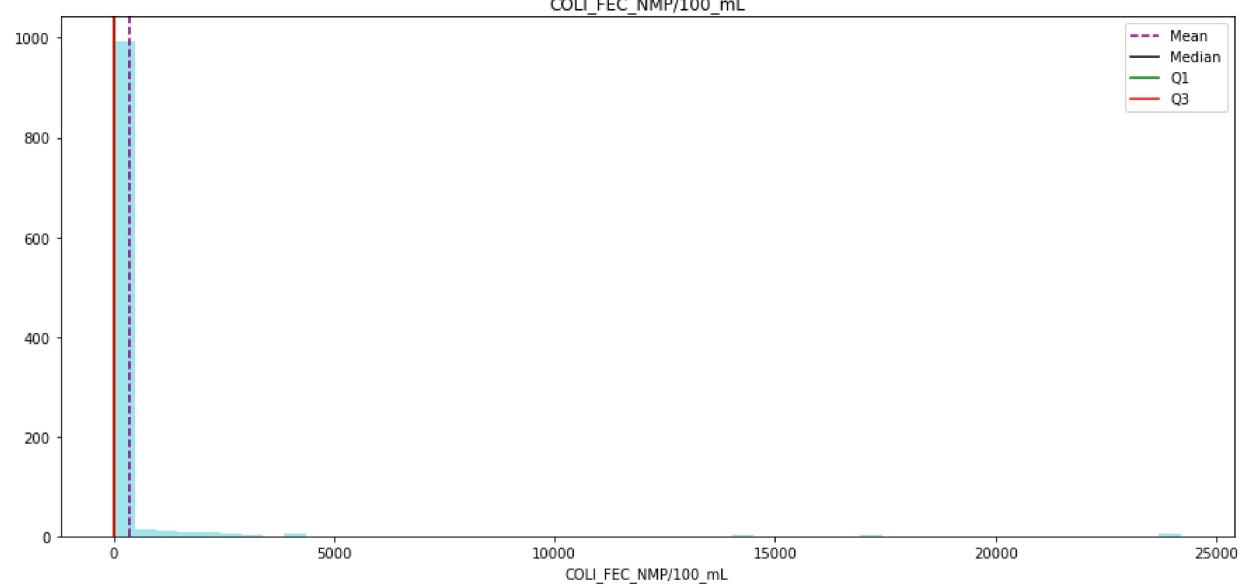
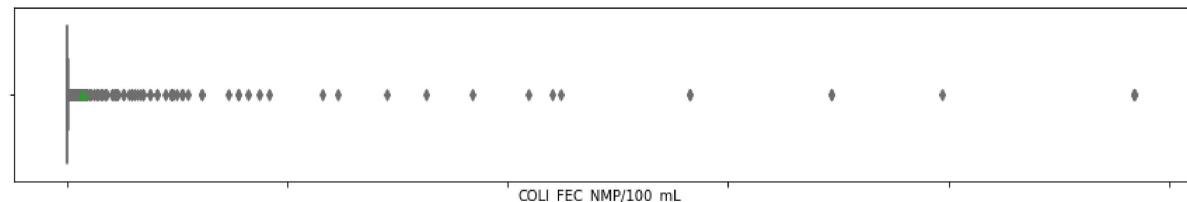
```
In [12]: for col in NumCols:
    histogram_boxplot(df,col)
```

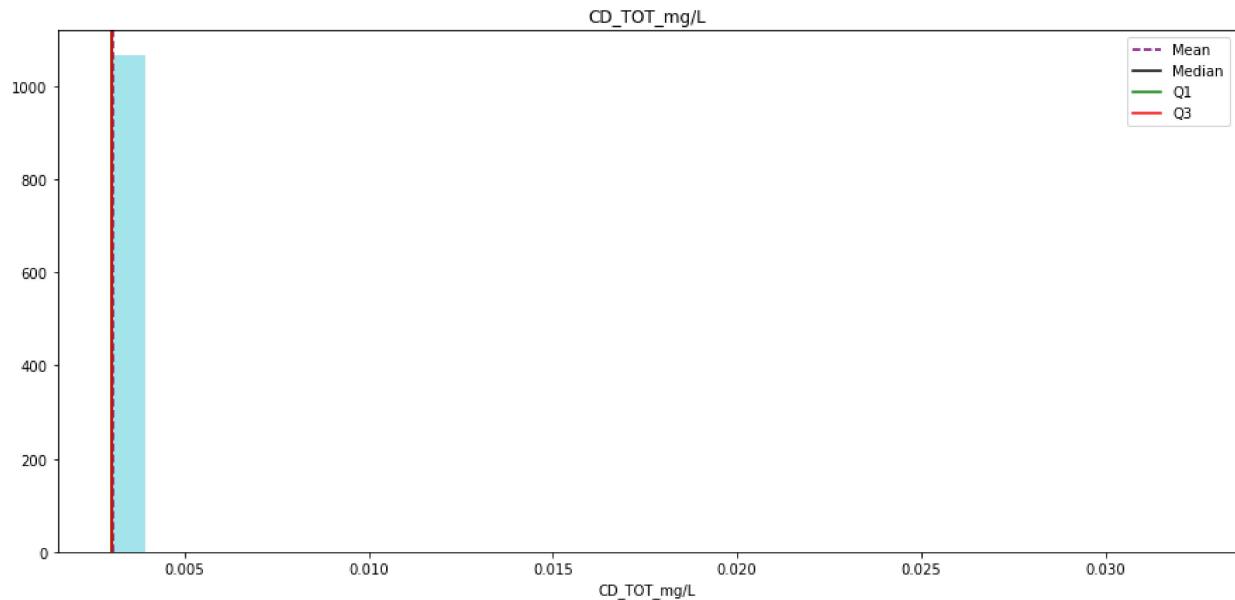
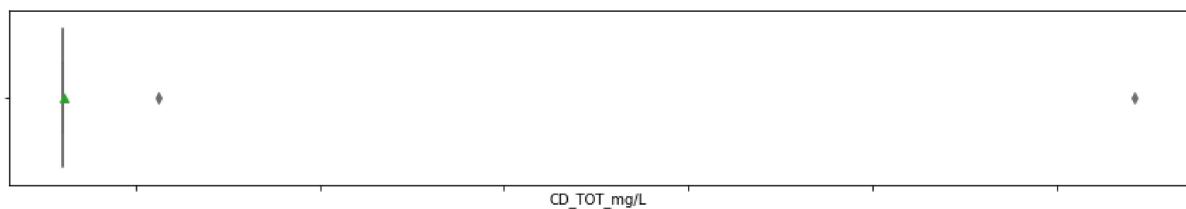
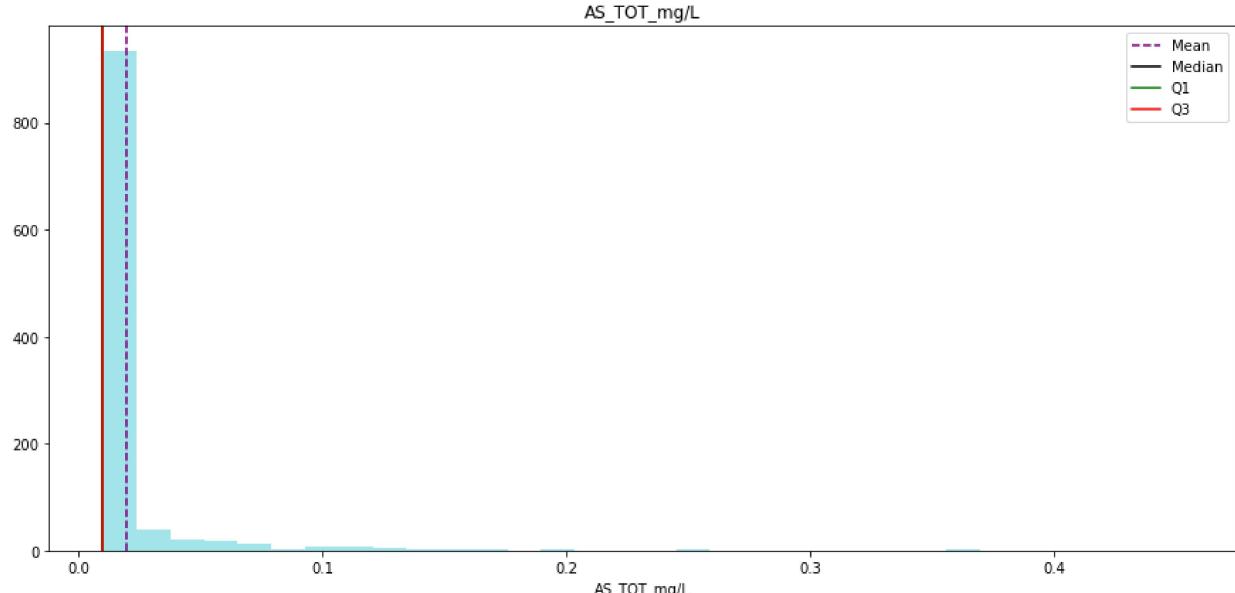
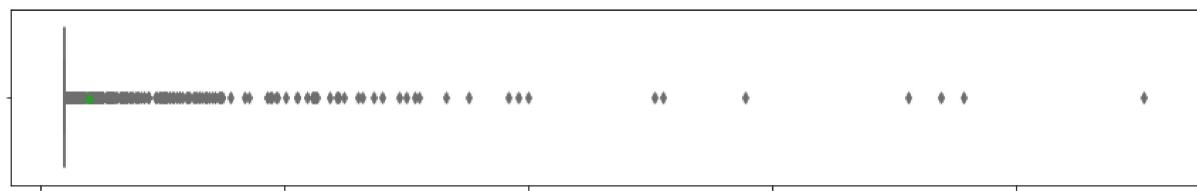


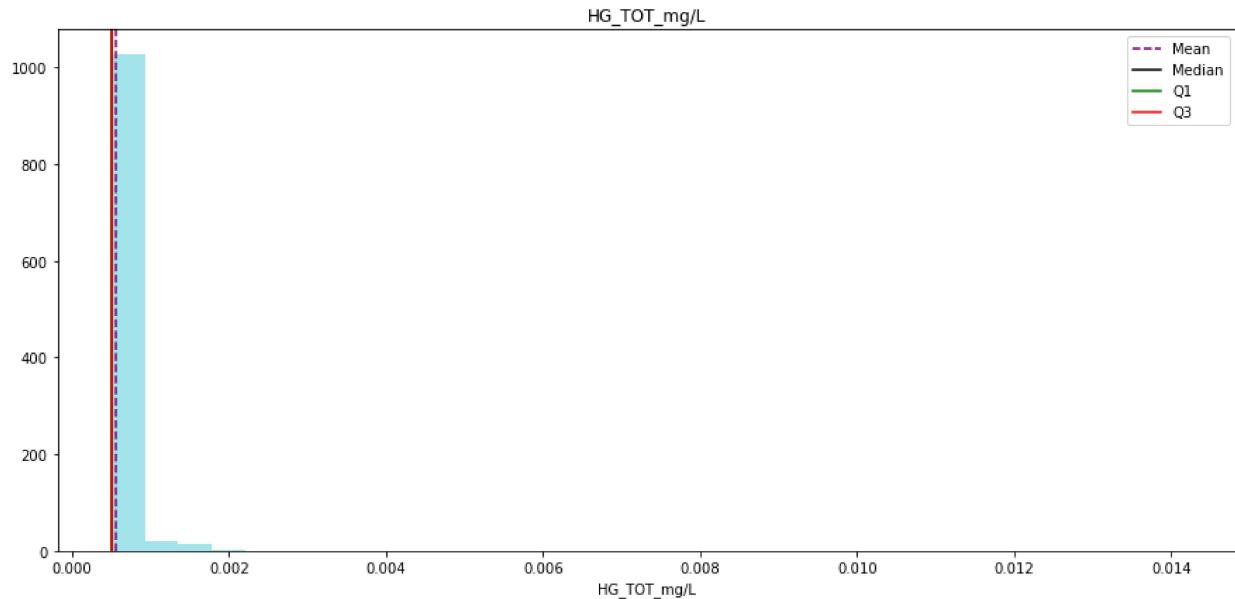
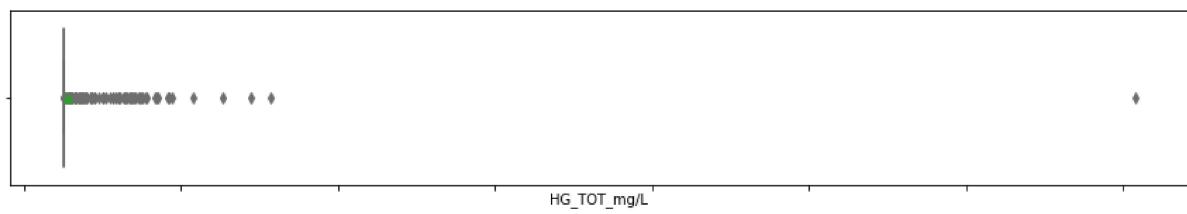
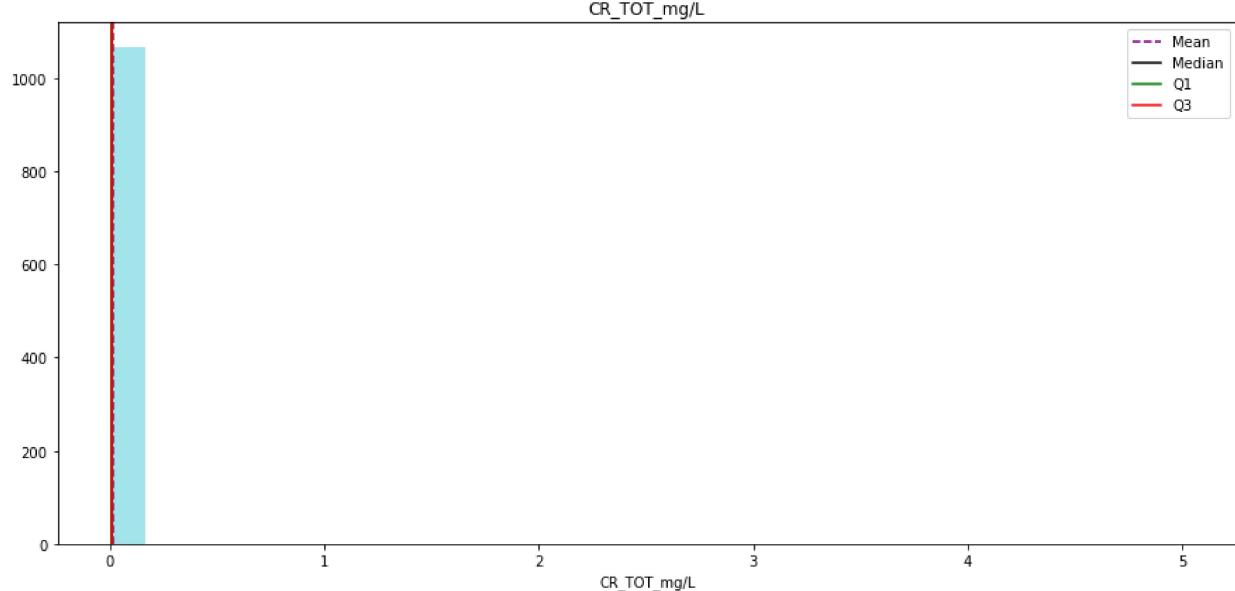
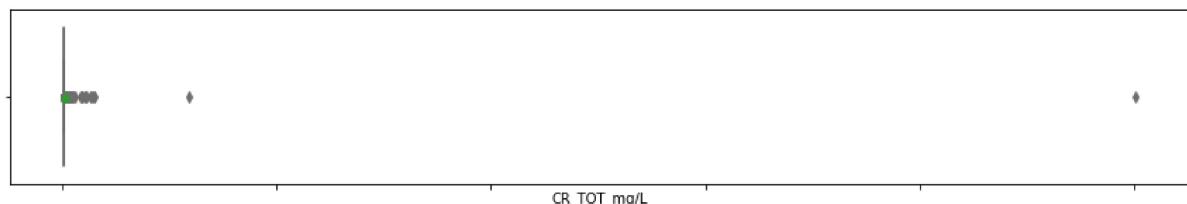


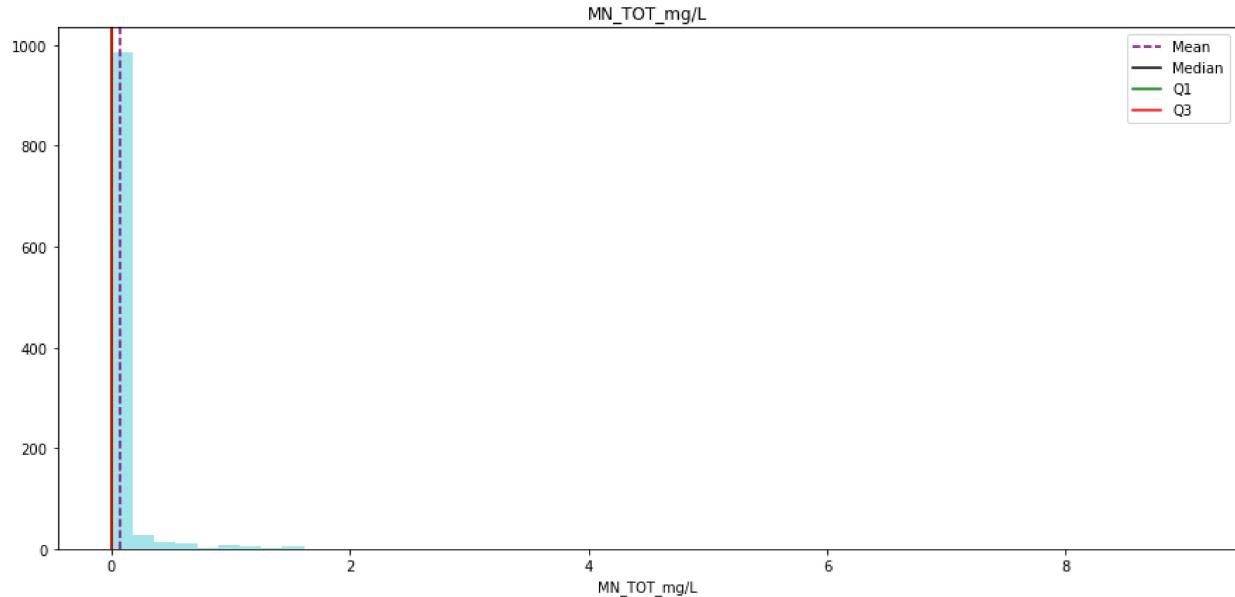
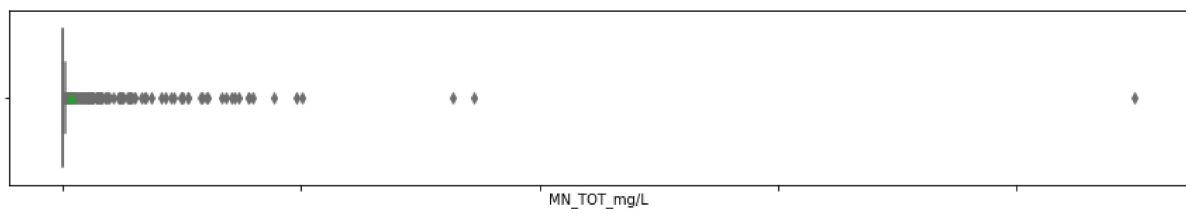
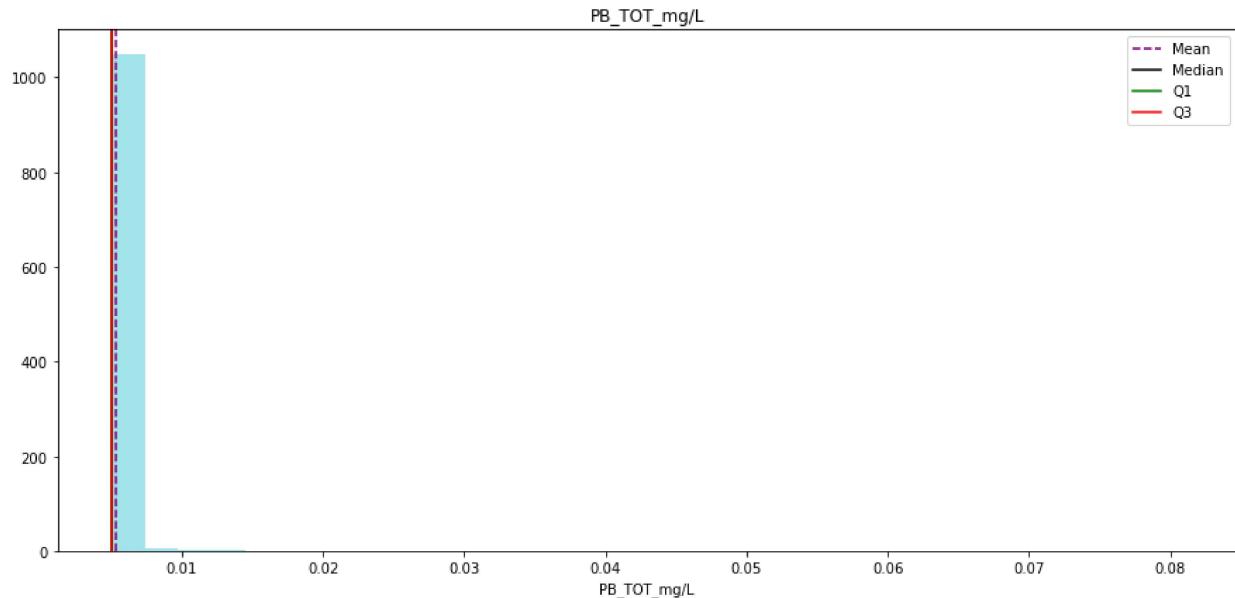
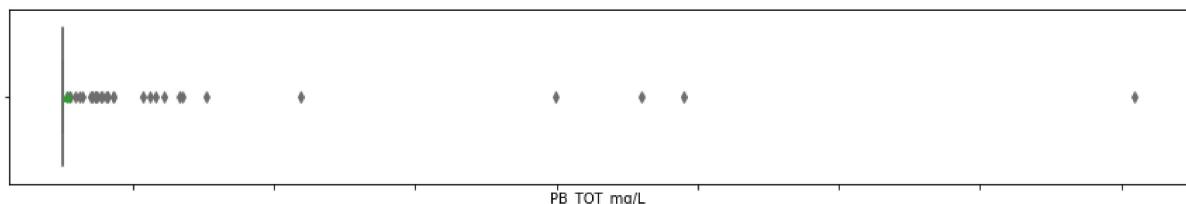


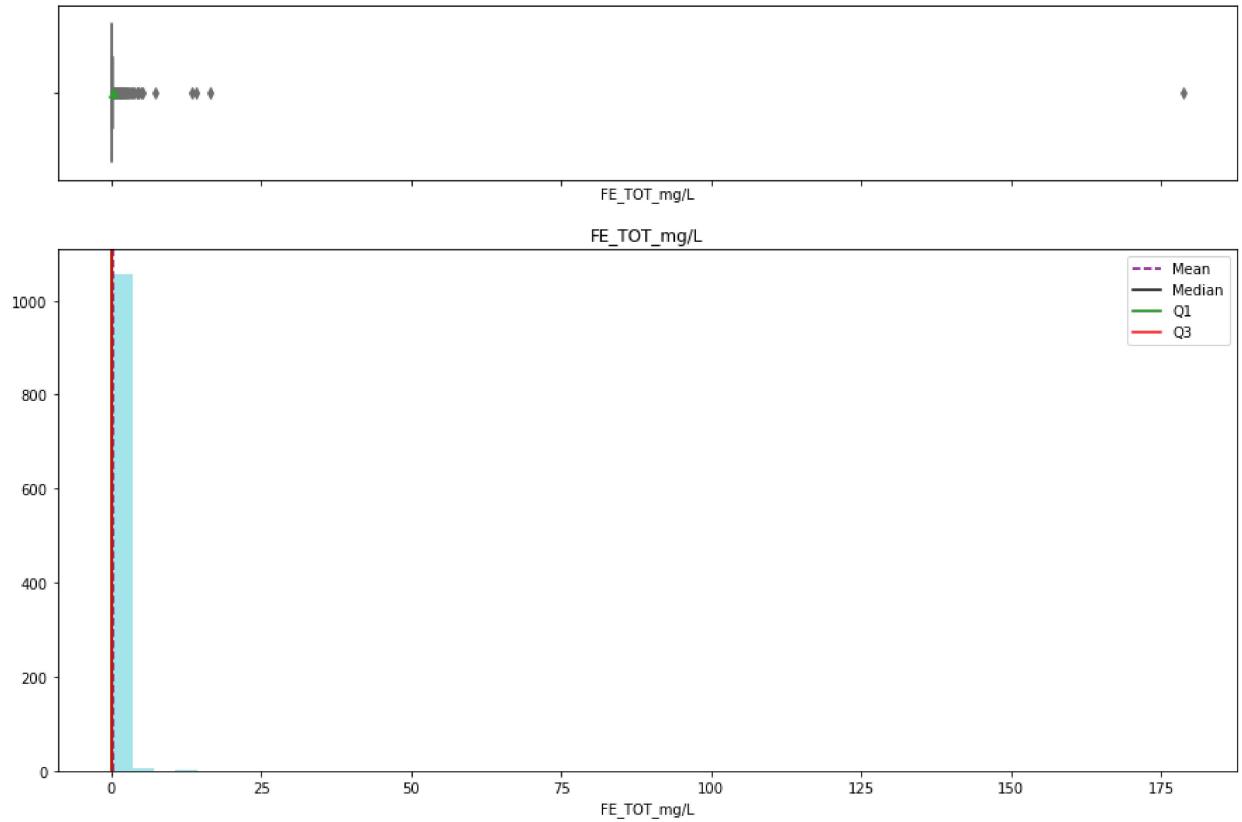








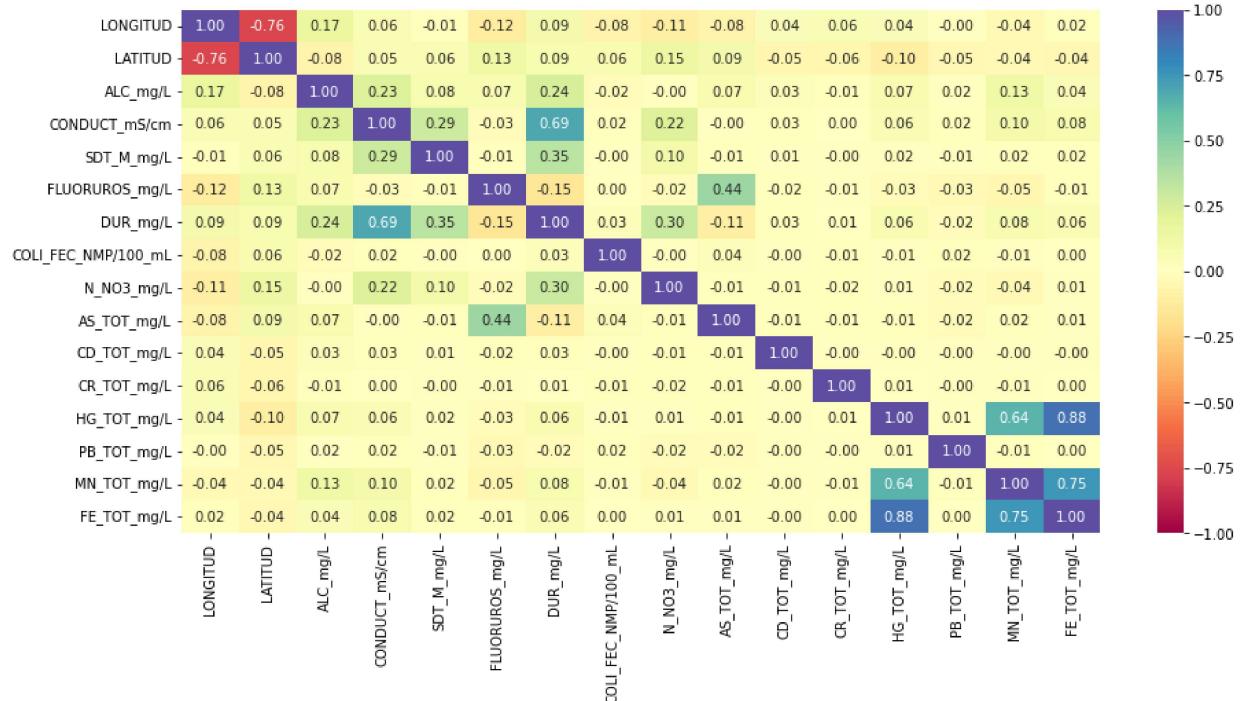




Podemos ver que hay un gran numero de outliers en todas las medidas por litro. Probablemente, no sea por ruido sino por casos muy aislados que se deben considerar al momento de hacer el analisis.

Dicho esto, procederemos a revisar la correlacion entre las variables numericas.

```
In [13]: plt.figure(figsize=(15,7))
sns.heatmap(df[NumCols].corr(), annot=True, vmin=-1, vmax=1, fmt='.2f', cmap='Spectral')
plt.show()
```



Podemos ver que existe una correlacion relativamente grande entre los contaminantes por Litro. Por ejemplo:

- Dur_mg/L vs. CONDUCT_mS/cm
- HG_TOT_mg/L vs. FE_TOT_mg/L
- MN_TOT_mg/L vs. FE_TOT_mg/L
- HG_TOT_mg/L vs. MN_TOT_mg/L

Ahora analizaremos las variables categoricas

```
In [14]: print("===== Binary Columns =====")
for col in BinCols:
    print(df[col].value_counts())
print("===== Categorical Columns =====")
for col in CatCols:
    print(df[col].value_counts())
print("===== Ordinal Columns =====")
for col in OrdCols:
    print(df[col].value_counts())
```

```
===== Binary Columns =====
SI    1005
NO     59
ND      4
Name: CUMPLE_CON_ALC, dtype: int64
SI    939
NO   123
ND      6
Name: CUMPLE_CON_COND, dtype: int64
SI    995
NO    71
ND      2
Name: CUMPLE_CON_SDT_ra, dtype: int64
SI    995
NO    71
ND      2
Name: CUMPLE_CON_SDT_salin, dtype: int64
SI    876
NO   192
Name: CUMPLE_CON_FLUO, dtype: int64
SI    841
NO   226
ND      1
Name: CUMPLE_CON_DUR, dtype: int64
SI   1007
NO    61
Name: CUMPLE_CON_CF, dtype: int64
SI    985
NO    82
ND      1
Name: CUMPLE_CON_NO3, dtype: int64
SI    941
NO   127
Name: CUMPLE_CON_AS, dtype: int64
SI   1066
NO      2
Name: CUMPLE_CON_CD, dtype: int64
SI   1053
NO    15
Name: CUMPLE_CON_CR, dtype: int64
SI   1067
NO      1
Name: CUMPLE_CON_HG, dtype: int64
SI   1056
NO    12
Name: CUMPLE_CON_PB, dtype: int64
SI    982
NO    86
Name: CUMPLE_CON_MN, dtype: int64
SI    932
NO   136
Name: CUMPLE_CON_FE, dtype: int64
===== Categorical Columns =====
DLAGU6      1
OCGCE3209    1
OCFSU2993    1
OCFSU2994    1
OCFSU3048    1
                  ..
DLHID6458    1
DLHID6461    1
DLHID6463    1
DLHID6467    1
OCRBR5109M1  1
Name: CLAVE, Length: 1068, dtype: int64
EL FUERTE          2
POZO VILLA UNION    2
POZO BERRIOZBAL    1
RANCHO GIUSEPPE CONSTANZO  1
QUINTA 2 POTRILLOS  1
                  ..
POZO SAN FRANCISCO BOJAY COLONIA  1
POZO SANTA ANA AHUEHUEPAN    1
POZO SANTA MARIA DAXTHO      1
POZO PEDRO MARIA ANAYA       1
COMUNIDAD LA REFORMA        1
Name: SITIO, Length: 1066, dtype: int64
CUENCAS CENTRALES DEL NORTE  232
LERMA SANTIAGO PACIFICO     170
PENINSULA DE YUCATAN       125
```

NOROESTE	94
PENINSULA DE BAJA CALIFORNIA	89
BALSAS	69
RIO BRAVO	65
PACIFICO NORTE	62
GOLFO NORTE	53
AGUAS DEL VALLE DE MEXICO	38
FRONTERA SUR	34
GOLFO CENTRO	21
PACIFICO SUR	16
Name: ORGANISMO_DE_CUENCA, dtype: int64	
DURANGO	121
SONORA	103
YUCATAN	85
ZACATECAS	75
COAHUILA DE ZARAGOZA	59
BAJA CALIFORNIA SUR	49
SAN LUIS POTOSI	47
GUANAJUATO	41
HIDALGO	37
CHIHUAHUA	35
JALISCO	33
SINALOA	32
BAJA CALIFORNIA	31
MICHOACAN DE OCAMPO	27
COLIMA	26
CAMPECHE	25
TAMAULIPAS	25
MEXICO	24
TLAXCALA	24
PUEBLA	23
CHIAPAS	21
OAXACA	20
VERACRUZ DE IGNACIO DE LA LLAVE	16
NUEVO LEON	15
QUINTANA ROO	15
AGUASCALIENTES	14
TABASCO	13
MORELOS	11
NAYARIT	8
QUERETARO ARTEAGA	6
GUERRERO	5
DISTRITO FEDERAL	2
Name: ESTADO, dtype: int64	
LA PAZ	27
ENSENADA	26
PARRAS	24
HERMOSILLO	17
MERIDA	16
..	
CUAUITEPEC DE HINOJOSA	1
LAS ROSAS	1
SOCOLTENANGO	1
COMITAN DE DOMINGUEZ	1
MELCHOR OCAMPO	1
Name: MUNICIPIO, Length: 452, dtype: int64	
PENINSULA DE YUCATAN	119
PRINCIPAL-REGION LAGUNERA	28
ALTO ATOYAC	19
TEPEHUANES-SANTIAGO	16
LA PAILA	12
...	
CABRERA-OCAMPO	1
TENANCINGO	1
IRAPUATO-VALLE	1
ROSARIO-TESOPACO-EL QUIRIEGO	1
TEPEJI DEL RIO	1
Name: ACUIFERO, Length: 273, dtype: int64	
POZO	1039
MANANTIAL	12
CENOTE	7
POZO NORIA	4
NORIA	3
DESCARGA	1
Pozo	1
BOMBEO CENOTE	1
Name: SUBTIPO, dtype: int64	
Potable - Dulce	834
Ligeramente salobres	161
Salobres	68

```

Salinas          3
Name: CALIDAD_SDT_salin, dtype: int64
Potable - Dura      577
Muy dura e indeseable usos industrial y domestico 226
Potable - Moderadamente suave      168
Potable - Suave      96
Name: CALIDAD_DUR, dtype: int64
Potable - Excelente    788
Potable - Buena calidad   197
No apta como FAAP      82
Name: CALIDAD_N_NO3, dtype: int64
Potable - Excelente    816
No apta como FAAP      127
Apta como FAAP      125
Name: CALIDAD_AS, dtype: int64
Potable - Excelente    1066
No apta como FAAP      2
Name: CALIDAD_CD, dtype: int64
Potable - Excelente    1053
No apta como FAAP      15
Name: CALIDAD_CR, dtype: int64
Potable - Excelente    1067
No apta como FAAP      1
Name: CALIDAD_HG, dtype: int64
Potable - Excelente    1056
No apta como FAAP      12
Name: CALIDAD_PB, dtype: int64
Potable - Excelente      982
Puede afectar la salud      50
Sin efectos en la salud - Puede dar color al agua 36
Name: CALIDAD_MN, dtype: int64
Potable - Excelente      932
Sin efectos en la salud - Puede dar color al agua 136
Name: CALIDAD_FE, dtype: int64
Verde      434
Rojo       387
Amarillo    247
Name: SEMAFORO, dtype: int64
FLUO,          78
DT,           65
FLUO,AS,        51
CF,            31
AS,            31
..             ..
ALC,CONDUC,SDT_ra,SDT_salin,DT,NO3,      1
ALC,CONDUC,SDT_ra,SDT_salin,FLUO,DT,AS,MN,FE,      1
PB,MN,FE,        1
ALC,AS,FE,        1
ALC,DT,NO3,        1
Name: CONTAMINANTES, Length: 126, dtype: int64
===== Ordinal Columns =====
Alta      794
Media     187
Indeseable como FAAP      59
Baja      24
Name: CALIDAD_ALC, dtype: int64
Permisible para riego    460
Buena para riego      434
Dudosa para riego      72
Indeseable para riego    51
Excelente para riego     45
Name: CALIDAD_CONDUC, dtype: int64
Excelente para riego     491
Cultivos sensibles      343
Cultivos con manejo especial 161
Cultivos tolerantes      64
Indeseable para riego     7
Name: CALIDAD_SDT_ra, dtype: int64
Baja      434
Potable - Optima     226
Media      216
Alta       192
Name: CALIDAD_FLUO, dtype: int64
Potable - Excelente    739
Buena calidad      208
Aceptable        60
Contaminada        49
Fuertemente contaminada 12
Name: CALIDAD_COLI_FEC, dtype: int64

```

De las variables categoricas podemos ver que hay mucho trabajo por hacer.

1. En las variables binarias, vemos que hay un typo donde escribieron ND en lugar de no. Por lo tanto, convertiremos estos valores a NO

```
In [15]: for col in BinCols:
    for row in range(0,df.shape[0]):
        val = str(df.loc[row,col])
        if 'ND' in val:
            df.loc[row,col] = "NO"

        if df.loc[row,col] == "NO":
            df.loc[row,col] = 0
        elif df.loc[row,col] == "SI":
            df.loc[row,col] = 1
    df[col] = df[col].astype("int")
```

1. Podemos ver que hay valores redundantes en cuanto a locación y hay otros que no nos sirven de nada para el análisis estadístico:

- a. No aportan información extra al modelo: CLAVE, SITIO
- b. Información Redundante: ESTADO, MUNICIPIO, ACUIFERO

```
In [16]: df = df.drop(["CLAVE", "SITIO", "ESTADO", "MUNICIPIO", "ACUIFERO"], axis=1)

drpCols = ["CLAVE", "SITIO", "ESTADO", "MUNICIPIO", "ACUIFERO"]
for col in drpCols:
    CatCols.pop(CatCols.index(col))
```

1. La columna de contaminantes contiene multiples categorias, por lo que vamos a dividirlo en multiples columnas y convertirlos en columnas binarias. Como nota extra, podemos notar que los valores NA de esta columna, estan atribuidos a todos aquellos que no tienen contaminantes.

```
In [17]: # Con esto vamos a obtener todos los valores de los contaminantes
raw_cont = df["CONTAMINANTES"].dropna().unique()
# Ahora lo convertiremos en un array de una sola dimension

clean_cont = []
for cont in raw_cont:
    contArr = cont.split(",")
    for contSimp in contArr:
        if (not contSimp in clean_cont) and (contSimp != ""):
            clean_cont.append(contSimp)

# Ya con los contaminantes limpios, creamos las columnas nuevas.
# Si contiene el contaminante le agregaremos un 1 y si no un 0

for cont in clean_cont:
    cont_cont = []
    for row in range(0,df.shape[0]):
        val = str(df.loc[row,"CONTAMINANTES"])
        if cont in val:
            cont_cont.append(1)
        else:
            cont_cont.append(0)

    df["CONT_"+cont] = cont_cont

## Hecho esto, podemos eliminar la columna de contaminantes
df = df.drop("CONTAMINANTES", axis=1)
CatCols.pop(CatCols.index("CONTAMINANTES"))
```

Out[17]: 'CONTAMINANTES'

1. En el Subtipo de acuífero, hay valores que solo tienen un typo. Por lo que solo se requiere corregirlos. Y hay unos que son bi-clase, por lo que podemos juntarlos

```
In [18]: for row in range(0,df.shape[0]):
    val = str(df.loc[row,"SUBTIPO"]).upper()
    if val == "NORIA" or val == "POZO NORIA":
        df.loc[row,"SUBTIPO"] = "POZO/NORIA"
    elif val == "BOMBEO CENOTE":
```

```

        df.loc[row,"SUBTIPO"] = "CENOTE"
    else:
        df.loc[row,"SUBTIPO"] = val

df["SUBTIPO"].value_counts()

Out[18]:
POZO          1040
MANANTIAL      12
CENOTE         8
POZO/NORIA     7
DESCARGA        1
Name: SUBTIPO, dtype: int64

```

1. Haremos el encoding de las variables categóricas, convirtiéndolas de alfanuméricas a numéricas

```

In [19]:
DicCat = {}
cat_na = CatCols + BinCols + OrdCols
for col in cat_na:
    print("==> MAPPING FOR " + col + " ==>")
    if str(df[col].dtypes) == "object":
        uni = list(df[col].unique())
        i = 0
        CurrDic = {}
        for val in uni:
            if str(val) != "nan":
                CurrDic[val] = i
                print(" ==> " + str(val) + ": " + str(i))
                i+=1
        DicCat[col] = CurrDic
    else:
        uni = list(df[col].unique())
        i = 0
        CurrDic = {}
        for val in uni:
            if str(val) != "nan":
                CurrDic[val] = val
                print(" ==> " + str(val) + ": " + str(val))
                i+=1
        DicCat[col] = CurrDic
    df[col] = df[col].map(CurrDic).astype('Int32')

```

```

==== MAPPING FOR ORGANISMO_DE_CUENCA ====
=> LERMA SANTIAGO PACIFICO: 0
=> PENINSULA DE BAJA CALIFORNIA: 1
=> PENINSULA DE YUCATAN: 2
=> RIO BRAVO: 3
=> CUENCAS CENTRALES DEL NORTE: 4
=> PACIFICO NORTE: 5
=> BALSAS: 6
=> PACIFICO SUR: 7
=> GOLFO NORTE: 8
=> AGUAS DEL VALLE DE MEXICO: 9
=> GOLFO CENTRO: 10
=> FRONTERA SUR: 11
=> NOROESTE: 12
==== MAPPING FOR SUBTIPO ====
=> POZO: 0
=> MANANTIAL: 1
=> POZO/NORIA: 2
=> DESCARGA: 3
=> CENOTE: 4
==== MAPPING FOR CALIDAD_SDT_salin ====
=> Potable - Dulce: 0
=> Ligeramente salobres: 1
=> Salobres: 2
=> Salinas: 3
==== MAPPING FOR CALIDAD_DUR ====
=> Potable - Dura: 0
=> Muy dura e indeseable usos industrial y domestico: 1
=> Potable - Suave: 2
=> Potable - Moderadamente suave: 3
==== MAPPING FOR CALIDAD_N_NO3 ====
=> Potable - Excelente: 0
=> Potable - Buena calidad: 1
=> No apta como FAAP: 2
==== MAPPING FOR CALIDAD_AS ====
=> Apta como FAAP: 0
=> No apta como FAAP: 1
=> Potable - Excelente: 2
==== MAPPING FOR CALIDAD_CD ====
=> Potable - Excelente: 0
=> No apta como FAAP: 1
==== MAPPING FOR CALIDAD_CR ====
=> Potable - Excelente: 0
=> No apta como FAAP: 1
==== MAPPING FOR CALIDAD_HG ====
=> Potable - Excelente: 0
=> No apta como FAAP: 1
==== MAPPING FOR CALIDAD_PB ====
=> Potable - Excelente: 0
=> No apta como FAAP: 1
==== MAPPING FOR CALIDAD_MN ====
=> Potable - Excelente: 0
=> Puede afectar la salud: 1
=> Sin efectos en la salud - Puede dar color al agua: 2
==== MAPPING FOR CALIDAD_FE ====
=> Potable - Excelente: 0
=> Sin efectos en la salud - Puede dar color al agua: 1
==== MAPPING FOR SEMAFORO ====
=> Verde: 0
=> Rojo: 1
=> Amarillo: 2
==== MAPPING FOR CUMPLE_CON_ALC ====
=> 1: 1
=> 0: 0
==== MAPPING FOR CUMPLE_CON_COND ====
=> 1: 1
=> 0: 0
==== MAPPING FOR CUMPLE_CON_SDT_ra ====
=> 1: 1
=> 0: 0
==== MAPPING FOR CUMPLE_CON_SDT_salin ====
=> 1: 1
=> 0: 0
==== MAPPING FOR CUMPLE_CON_FLUO ====
=> 1: 1
=> 0: 0
==== MAPPING FOR CUMPLE_CON_DUR ====
=> 1: 1
=> 0: 0
==== MAPPING FOR CUMPLE_CON_CF ====

```

```
=> 1: 1
=> 0: 0
==> MAPPING FOR CUMPLE_CON_N03 ===
=> 1: 1
=> 0: 0
==> MAPPING FOR CUMPLE_CON_AS ===
=> 1: 1
=> 0: 0
==> MAPPING FOR CUMPLE_CON_CD ===
=> 1: 1
=> 0: 0
==> MAPPING FOR CUMPLE_CON_CR ===
=> 1: 1
=> 0: 0
==> MAPPING FOR CUMPLE_CON_HG ===
=> 1: 1
=> 0: 0
==> MAPPING FOR CUMPLE_CON_PB ===
=> 1: 1
=> 0: 0
==> MAPPING FOR CUMPLE_CON_MN ===
=> 1: 1
=> 0: 0
==> MAPPING FOR CUMPLE_CON_FE ===
=> 1: 1
=> 0: 0
==> MAPPING FOR CALIDAD_ALC ===
=> Alta: 0
=> Media: 1
=> Baja: 2
=> Indeseable como FAAP: 3
==> MAPPING FOR CALIDAD_CONDUC ===
=> Permitible para riego: 0
=> Buena para riego: 1
=> Dudosa para riego: 2
=> Indeseable para riego: 3
=> Excelente para riego: 4
==> MAPPING FOR CALIDAD_SDT_ra ===
=> Cultivos sensibles: 0
=> Excelente para riego: 1
=> Cultivos con manejo especial: 2
=> Cultivos tolerantes: 3
=> Indeseable para riego: 4
==> MAPPING FOR CALIDAD_FLUO ===
=> Potable - Optima: 0
=> Alta: 1
=> Baja: 2
=> Media: 3
==> MAPPING FOR CALIDAD_COLI_FEC ===
=> Potable - Excelente: 0
=> Aceptable: 1
=> Contaminada: 2
=> Buena calidad: 3
=> Fuertemente contaminada: 4
```

```
In [20]: def inverse_mapping(Dict,Cat,dataFr):
    inv_dict = {v: k for k, v in Dict.items()}
    dataFr[Cat] = np.round(dataFr[Cat]).map(inv_dict).astype('category')
```

Preprocesamiento

1. Separación de los datos en train y test

```
In [21]: x = df.drop(columns = ["SEMAFORO"])
y = df["SEMAFORO"].astype('int')
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.15, random_state=7,stratify=y)
```

1. Haremos la imputación de los datos faltantes. Usaremos el método de KNN Imputer

```
In [22]: imputer = KNNImputer(n_neighbors=6)
#Fit and transform the train data
x_train=pd.DataFrame(imputer.fit_transform(x_train),columns=x_train.columns)

#Transform the test data
x_test=pd.DataFrame(imputer.transform(x_test),columns=x_test.columns)
```

```
In [23]: #Checking that no column has missing values in train or test sets
print(x_train.isna().sum())
print('*30)
print("Total NaN: ",x_train.isna().sum().sum())
print('*30)
print(x_test.isna().sum())
print('*30)
print("Total NaN: ",x_test.isna().sum().sum())

ORGANISMO_DE_CUENCA      0
SUBTIPO                   0
LONGITUD                  0
LATITUD                   0
ALC_mg/L                  0
..
CONT_CR                   0
CONT_ALC                  0
CONT_NI                   0
CONT_HG                   0
CONT_CD                   0
Length: 64, dtype: int64
-----
Total Nan:  0
=====
ORGANISMO_DE_CUENCA      0
SUBTIPO                   0
LONGITUD                  0
LATITUD                   0
ALC_mg/L                  0
..
CONT_CR                   0
CONT_ALC                  0
CONT_NI                   0
CONT_HG                   0
CONT_CD                   0
Length: 64, dtype: int64
-----
Total Nan:  0
```

1. Antes de regresar a su valor original las variables categóricas, haremos un balanceo de las clases usando SMOTETomek

```
In [24]: from imblearn.over_sampling import SMOTE
oversample = SMOTE()
x_train, y_train = oversample.fit_resample(x_train, y_train)
```

1. Hecho esto haremos el mapeo inverso de las variables categóricas

```
In [25]: for col in cat_na:
    if col != "SEMAFORO":
        inverse_mapping(DicCat[col], col, x_train)
        inverse_mapping(DicCat[col], col, x_test)
```

1. Finalmente haremos el Encoding de las categorías para evitar que alguna categoría tenga mayor peso que otra. Para esto usaremos get_dummies

```
In [26]: x_train=pd.get_dummies(x_train,drop_first=True)
x_test=pd.get_dummies(x_test,drop_first=True)

x_train, x_test = x_train.align(x_test,join='left', axis=1)
x_test = x_test.fillna(0)
print(x_train.shape, x_test.shape)

(1104, 98) (161, 98)
```

```
In [27]: #Checking that no column has missing values in train or test sets
print(x_train.isna().sum())
print('*30)
print("Total NaN: ",x_train.isna().sum().sum())
print('*30)
print(x_test.isna().sum())
print('*30)
print("Total NaN: ",x_test.isna().sum().sum())
```

```

LONGITUD      0
LATITUD       0
ALC_mg/L      0
CONDUCT_mS/cm 0
SDT_M_mg/L    0
...
CUMPLE_CON_CR_1 0
CUMPLE_CON_HG_1 0
CUMPLE_CON_PB_1 0
CUMPLE_CON_MN_1 0
CUMPLE_CON_FE_1 0
Length: 98, dtype: int64
-----
Total NaN: 0
=====
LONGITUD      0
LATITUD       0
ALC_mg/L      0
CONDUCT_mS/cm 0
SDT_M_mg/L    0
...
CUMPLE_CON_CR_1 0
CUMPLE_CON_HG_1 0
CUMPLE_CON_PB_1 0
CUMPLE_CON_MN_1 0
CUMPLE_CON_FE_1 0
Length: 98, dtype: int64
-----
Total NaN: 0

```

Construcción del Modelo

- Para empezar debemos definir la métrica a utilizar para obtener el mejor resultado. Usaremos métricas clásicas como Accuracy, Recall y Precision, junto con la gráfica de Precision Recall y el reporte de clasificación

```
In [45]: def PrintScores(model, train, test, train_y, test_y, labels):
    score_list = []

    pred_train = model.predict(train)
    pred_test = model.predict(test)

    train_acc = metrics.balanced_accuracy_score(train_y, pred_train)
    test_acc = metrics.balanced_accuracy_score(test_y, pred_test)

    train_recall = metrics.recall_score(train_y, pred_train, labels = labels, average = 'weighted')
    test_recall = metrics.recall_score(test_y, pred_test, labels = labels, average = 'weighted')

    train_precision = metrics.precision_score(train_y, pred_train, labels = labels, average = 'weighted')
    test_precision = metrics.precision_score(test_y, pred_test, labels = labels, average = 'weighted')
    score_list.extend((train_acc, test_acc, train_recall, test_recall, train_precision, test_precision))

    print("Accuracy on training set : ", train_acc)
    print("Accuracy on test set : ", test_acc)
    print("Recall on training set : ", train_recall)
    print("Recall on test set : ", test_recall)
    print("Precision on training set : ", train_precision)
    print("Precision on test set : ", test_precision)
    return score_list
```

```
In [62]: def DisplayConfMat(model, x_, y_real, labels = [0,1,2]):
    y_pred = model.predict(x_)
    cm = confusion_matrix(y_test, y_pred)
    display(cm)
    df_cm = pd.DataFrame(cm, index = ["Actual - " + str(i) for i in labels], columns = ["Predicted - " + str(i) for i in labels])

    group_counts = ["{0:0.0f}".format(value) for value in
                   cm.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in
                          cm.flatten() / np.sum(cm)]
    labels = [f"{v1}\n{v2}" for v1, v2 in
              zip(group_counts, group_percentages)]
    labels = np.asarray(labels).reshape(3,3)

    plt.figure(figsize = (10,7))
    sns.heatmap(df_cm, annot=labels, fmt=' ')
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Hyper Parameter Tuning - Decision Tree

```
In [30]: # Creating pipeline
pipe = make_pipeline(StandardScaler(), DecisionTreeClassifier(random_state=1))

# Parameter grid to pass in RandomSearchCV
param_grid = {'decisiontreeclassifier__max_depth': np.arange(2,30),
              'decisiontreeclassifier__min_samples_leaf': [1, 2, 5, 7, 10],
              'decisiontreeclassifier__max_leaf_nodes' : [2, 3, 5, 10,15],
              'decisiontreeclassifier__min_impurity_decrease': [0.0001,0.001,0.01,0.1]
             }

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(brier_score_loss)

#Calling RandomizedSearchCV
kfold = RepeatedStratifiedKFold(n_splits=5, n_repeats=3)
randomized_cv = RandomizedSearchCV(estimator=pipe, param_distributions=param_grid, n_jobs = -1, n_iter=50, s

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(x_train,y_train)

print("Best parameters are {} with CV score={}: ".format(randomized_cv.best_params_,randomized_cv.best_score_))

Best parameters are {'decisiontreeclassifier__min_samples_leaf': 5, 'decisiontreeclassifier__min_impurity_d
ecrease': 0.001, 'decisiontreeclassifier__max_leaf_nodes': 15, 'decisiontreeclassifier__max_depth': 18} wit
h CV score=nan:
```

Generación de Modelo

Ya con los valores obtenidos, podemos generar un modelo "óptimo"

```
In [31]: dtree = make_pipeline(
    StandardScaler(),
    DecisionTreeClassifier(
        max_depth=18,
        max_leaf_nodes=15,
        random_state=1,
        min_impurity_decrease=0.001,
        min_samples_leaf=5
    ),
)

dtree.fit(x_train, y_train)
```

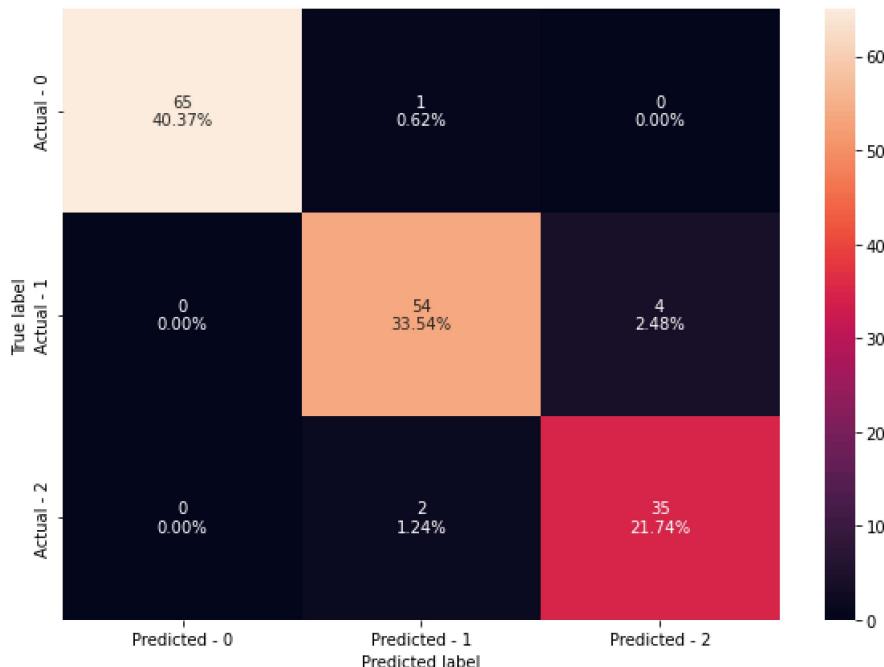
```
Out[31]: ▶ Pipeline
          ▶ StandardScaler
          ▶ DecisionTreeClassifier
```

Generación de Métricas

```
In [63]: # Calculating different metrics
PrintScores(dtree,x_train,x_test,y_train,y_test,labels = [0,1,2])

# Creating confusion matrix
DisplayConfMat(dtree, x_test, y_test, labels = [0,1,2])
```

```
Accuracy on training set : 0.9737318840579711
Accuracy on test set : 0.9539429711843505
Recall on training set : 0.9737318840579711
Recall on test set : 0.9565217391304348
Precision on training set : 0.9739704246370913
Precision on test set : 0.9574689231439804
array([[65,  1,  0],
       [ 0, 54,  4],
       [ 0,  2, 35]], dtype=int64)
```



Inicialmente podemos decir que el modelo está funcionando de una manera muy buena, dandonos un overall de las métricas por arriba del 90% tanto en datos de prueba, como de entrenamiento.

Haremos el reporte de clasificación y el análisis de la gráfica de Precision Recall.

```
In [70]: from sklearn.metrics import classification_report
y_pred = dtree.predict(x_test)

print(classification_report(y_test, y_pred, target_names=["Grupo " + str(i) for i in [0,1,2]]))

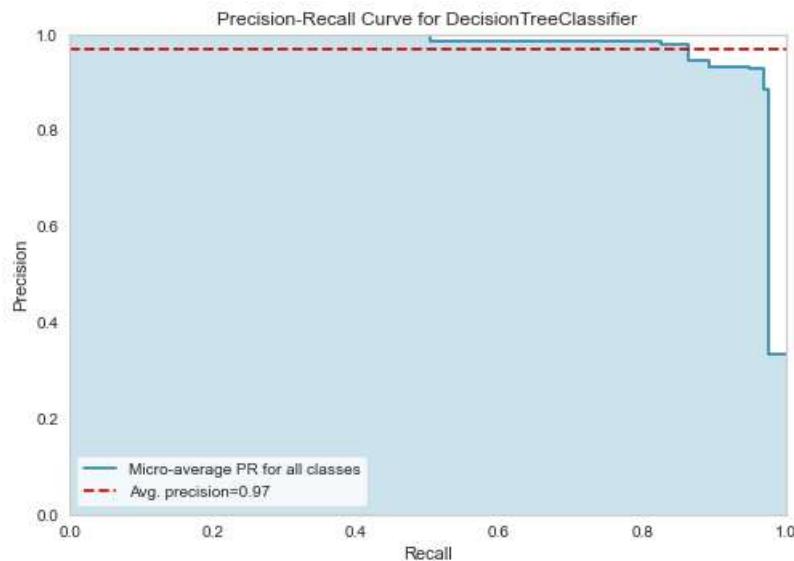
precision    recall    f1-score   support
Grupo 0      1.00      0.98      0.99      66
Grupo 1      0.95      0.93      0.94      58
Grupo 2      0.90      0.95      0.92      37

accuracy          0.96      0.96      0.96      161
macro avg       0.95      0.95      0.95      161
weighted avg    0.96      0.96      0.96      161
```

La tabla de precision recall, reafirma lo que habiamos visto con el confusion matrix.

Ahora procederemos con la gráfica de precision recall.

```
In [77]: from yellowbrick.classifier import PrecisionRecallCurve
viz = PrecisionRecallCurve(dtree)
viz.fit(x_train, y_train)
viz.score(x_test, y_test)
viz.show()
```



```
Out[77]: <AxesSubplot:title={'center':'Precision-Recall Curve for DecisionTreeClassifier'}, xlabel='Recall', ylabel='Precision'>
```

Una vez más podemos ver que el modelo funciona de manera esperada, con un alto nivel de confiabilidad.