



Tecnológico de Monterrey

▼ Reto: Base de datos calidad de agua

Parte 2: Clasificación y ensambles

Ciencia y Analítica de datos

Profesor Titular: María de la Paz Rico Fernández

Maestría en Inteligencia Artificial Aplicada (MNA-V)

18/11/2022

Equipo 24

Victor Hugo Avila Felipe - A01794425

Andrés Eduardo Figueroa García - A01378536

▼ Instrucciones - Parte 1

En esta base de datos encontrarás:

- Aguas subterráneas.
- Aguas superficiales.

Elije una base de datos, ya sea la de aguas superficiales o la de aguas subterráneas.

- Limpieza de base de datos.
 - Explorar cada datos (auxiliante de describe(), mean(), plot, boxplot de pandas):
 - Identificando tendencias centrales promedio, media y mediana de los datos.
 - Identificar medidas de dispersión, máximo, mínimo .
 - Identificar medidas de posición no centrales , los cuartiles , outliers.
 - Identificar correlaciones.
 - Preparar los datos
- Realizar análisis para encontrar si existe una relación entre la calidad del agua y su ubicación geográfica a través de K-means.

- Mostrar resultados de agrupamiento de latitudes y longitudes con K means en el mapa de México.

▼ Desarrollo - Parte 1

Se agregan las librerías que se van a estar utilizando a lo largo de esta entrega.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

!pip install qeds
import qeds
!pip install geopandas
import geopandas as gpd
from shapely.geometry import Point
from geopy.geocoders import Nominatim
import geopy

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

import warnings
warnings.filterwarnings("ignore")

pd.set_option('display.max_columns', None)
pd.options.mode.chained_assignment = None

Requirement already satisfied: seaborn in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: quantecon in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: openpyxl in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: quandl in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: statsmodels in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: pandas-datareader in /usr/local/lib/python3.7/dist
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: pyarrow in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: plotly in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/l
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/c
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages

Requirement already satisfied: et-xmlfile in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-pac
```

```
Requirement already satisfied: lxml in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: urllib3!=1.25.0,!>=1.25.1,<1.26,>=1.21.1 in /usr/lc
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dis
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: more-itertools in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: inflection>=0.3.1 in /usr/local/lib/python3.7/dist
Requirement already satisfied: numba in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: sympy in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in /usr/local/lib/pytho
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dis
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/c
Requirement already satisfied: patsy>=0.5 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.7/dist-pac
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whee
Requirement already satisfied: geopandas in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: shapely>=1.6 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: pyproj>=2.2.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: fiona>=1.8 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: pandas>=0.25.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: click>=4.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: munch in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: cligj>=0.5 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: click-plugins>=1.0 in /usr/local/lib/python3.7/dis
Requirement already satisfied: attrs>=17 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: six>=1.7 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.7/dist-pac
```

Se optó por utilizar la base de datos de aguas superficiales, por lo que el primer paso es importar la base de datos.

```
path = 'Datos_superficiales_2020.csv'
df = pd.read_csv(path, encoding='latin-1')
df
```

	CLAVE	SITIO	ORGANISMO_DE_CUENCA	ESTADO	MUNICIPIO	CUENC
0	DLAGU8	PRESA EL SAUCILLO 100M AGUAS ARRIBA DE LA CORTINA	LERMA SANTIAGO PACIFICO	AGUASCALIENTES	RINCON DE ROMOS	RIO SA PEDRI
1	DLBAJ100	LOS CABOS SEG 22, 2 ISA10B	PENINSULA DE BAJA CALIFORNIA	BAJA CALIFORNIA SUR	LOS CABOS	SA JOS DE CABI
2	DLBAJ101	LOS CABOS SEG 22, 1 ISA10B	PENINSULA DE BAJA CALIFORNIA	BAJA CALIFORNIA SUR	LOS CABOS	SA LUCA
3	DLBAJ102	LOS CABOS 3	PENINSULA DE BAJA CALIFORNIA	BAJA CALIFORNIA SUR	LOS CABOS	SA LUCA
4	DLBAJ103	LOS CABOS 1	PENINSULA DE BAJA CALIFORNIA	BAJA CALIFORNIA SUR	LOS CABOS	SA LUCA
...
...

Se puede observar de esta tabla que al menos las últimas 5 filas están llenas de datos vacíos. Se comprueba esto mediante la siguiente línea de código.

```
df.isnull().any()
```

CLAVE	True
SITIO	True
ORGANISMO_DE_CUENCA	True
ESTADO	True
MUNICIPIO	True
CUENCA	True
CUERPO_DE_AGUA	True
TIPO	True
SUBTIPO	True
LONGITUD	True
LATITUD	True
PERIODO	True
DBO_mg/L	True
CALIDAD_DBO	True
DQO_mg/L	True
CALIDAD_DQO	True
SST_mg/L	True
CALIDAD_SST	True
COLI_FEC_NMP_100mL	True
CALIDAD_COLI_FEC	True
E_COLI_NMP_100mL	True
CALIDAD_E_COLI	True

```

ENTEROC_NMP_100mL      True
CALIDAD_ENTEROC         True
OD_PORC                 True
CALIDAD_OD_PORC          True
OD_PORC_SUP              True
CALIDAD_OD_PORC_SUP      True
OD_PORC_MED              True
CALIDAD_OD_PORC_MED      True
OD_PORC_FON              True
CALIDAD_OD_PORC_FON      True
TOX_D_48_UT               True
CALIDAD_TOX_D_48          True
TOX_V_15_UT               True
CALIDAD_TOX_V_15          True
TOX_D_48_SUP_UT           True
CALIDAD_TOX_D_48_SUP      True
TOX_D_48_FON_UT           True
CALIDAD_TOX_D_48_FON      True
TOX_FIS_SUP_15_UT          True
CALIDAD_TOX_FIS_SUP_15     True
TOX_FIS_FON_15_UT          True
CALIDAD_TOX_FIS_FON_15     True
SEMAFORO                  True
CONTAMINANTES             True
CUMPLE_CON_DBO            True
CUMPLE_CON_DQO             True
CUMPLE_CON_SST             True
CUMPLE_CON_CF              True
CUMPLE_CON_E_COLI           True
CUMPLE_CON_ENTEROC          True
CUMPLE_CON_OD               True
CUMPLE_CON_TOX              True
GRUPO                      True
dtype: bool

```

Al revisar el archivo '.CSV' se puede verificar que contiene filas vacías a partir de la línea 3495. Esto no significa que no haya valores faltantes en otras filas, pero de inicio se propone eliminar todas las filas que no cuenten con el identificador en la columna CLAVE.

```
df = df[df['CLAVE'].notnull()]
```

Se vuelve a revisar si es que hay valores vacíos en todas la columnas.

```
df.isnull().any()
```

CLAVE	False
SITIO	False
ORGANISMO_DE_CUENCA	False
ESTADO	False
MUNICIPIO	False
CUENCA	True
CUERPO DE AGUA	True
TIPO	False
SUBTIPO	True
LONGITUD	False

LATITUD	False
PERIODO	False
DBO_mg/L	True
CALIDAD_DBO	True
DQO_mg/L	True
CALIDAD_DQO	True
SST_mg/L	True
CALIDAD_SST	True
COLI_FEC_NMP_100mL	True
CALIDAD_COLI_FEC	True
E_COLI_NMP_100mL	True
CALIDAD_E_COLI	True
ENTEROC_NMP_100mL	True
CALIDAD_ENTEROC	True
OD_PORC	True
CALIDAD_OD_PORC	True
OD_PORC_SUP	True
CALIDAD_OD_PORC_SUP	True
OD_PORC_MED	True
CALIDAD_OD_PORC_MED	True
OD_PORC_FON	True
CALIDAD_OD_PORC_FON	True
TOX_D_48_UT	True
CALIDAD_TOX_D_48	True
TOX_V_15_UT	True
CALIDAD_TOX_V_15	True
TOX_D_48_SUP_UT	True
CALIDAD_TOX_D_48_SUP	True
TOX_D_48_FON_UT	True
CALIDAD_TOX_D_48_FON	True
TOX_FIS_SUP_15_UT	True
CALIDAD_TOX_FIS_SUP_15	True
TOX_FIS_FON_15_UT	True
CALIDAD_TOX_FIS_FON_15	True
SEMAFORO	False
CONTAMINANTES	True
CUMPLE_CON_DBO	False
CUMPLE_CON_DQO	False
CUMPLE_CON_SST	False
CUMPLE_CON_CF	False
CUMPLE_CON_E_COLI	False
CUMPLE_CON_ENTEROC	False
CUMPLE_CON_OD	False
CUMPLE_CON_TOX	False
GRUPO	False
dtype:	bool

Analizando los datos en búsqueda de más variables a eliminar, se plantea hacer la siguiente lista con las columnas que se consideran más importantes, añadiendo después aquellas que resultan de evaluar cumplimiento de parámetros.

```
columns = ['CLAVE', 'ORGANISMO_DE_CUENCA', 'GRUPO', 'SUBTIPO', 'LONGITUD', 'LATITUD']

for i in df.columns:
    if i[:6] == 'CUMPLE':
```

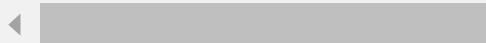
```
columns.append(i)

columns.append('SEMAFORO')
```

```
ndf = df[columns]
ndf
```

		CLAVE	ORGANISMO_DE_CUENCA	GRUPO	SUBTIPO	LONGITUD	LATITUD
0		DLAGU8	LERMA SANTIAGO PACIFICO	LENTICO	PRESA	-102.33911	22.24730
1		DLBAJ100	PENINSULA DE BAJA CALIFORNIA	COSTERO	OCEANO-MAR	-109.84290	22.90473
2		DLBAJ101	PENINSULA DE BAJA CALIFORNIA	COSTERO	OCEANO-MAR	-109.86442	22.89880
3		DLBAJ102	PENINSULA DE BAJA CALIFORNIA	COSTERO	BAHIA	-109.88604	22.89609
4		DLBAJ103	PENINSULA DE BAJA CALIFORNIA	COSTERO	BAHIA	-109.89657	22.87694
...	
3488		OCRBR5206M1	RIO BRAVO	LOTICO	RIO	-99.42142	26.78971
3489		OCRBR5207M1	RIO BRAVO	LENTICO	LAGO	-99.53064	27.43714
3490		OCRBR5208M1	RIO BRAVO	LOTICO	RIO	-99.50727	27.49901
3491		OCRBR5209M1	RIO BRAVO	LOTICO	RIO	-99.52221	27.49631
3492		OCRBR5210M1	RIO BRAVO	LOTICO	RIO	-99.52572	27.51697

3493 rows × 15 columns



Se revisa la cantidad de ND (No Disponible) en cada una de las columnas resultantes.

```
for column in columns:
    print(column + ': ', ndf[ndf[column] == 'ND'].shape)

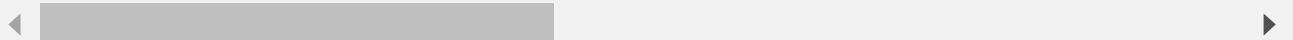
CLAVE: (0, 15)
ORGANISMO_DE_CUENCA: (0, 15)
GRUPO: (0, 15)
SUBTIPO: (0, 15)
LONGITUD: (0, 15)
LATITUD: (0, 15)
CUMPLE_CON_DBO: (912, 15)
CUMPLE_CON_DQO: (912, 15)
CUMPLE_CON_SST: (4, 15)
CUMPLE_CON_CF: (911, 15)
CUMPLE_CON_E_COLI: (911, 15)
```

```
CUMPLE_CON_ENTEROC: (2589, 15)
CUMPLE_CON_OD: (43, 15)
CUMPLE_CON_TOX: (0, 15)
SEMAFORO: (0, 15)
```

▼ Análisis variables de tipo objeto

```
columns_o = ndf.columns[np.array(ndf.dtypes.astype(str)=='object')]
ndf[columns_o].describe()
```

	CLAVE	ORGANISMO_DE_CUENCA	GRUPO	SUBTIPO	CUMPLE_CON_DBO	CUMPLE_CON_I	
count	3493	3493	3493	3479	3493	3493	3493
unique	3493	13	3	27		3	
top	DLAGU8	LERMA SANTIAGO PACIFICO	LOTICO	RIO		SI	
freq	1	709	1772	1478	2319	16	



Se puede observar que en la columna 'SUBTIPO' faltan elementos. Esto se puede arreglar con una imputación simple.

```
ndf[ndf['SUBTIPO'].isnull()]['GRUPO'].unique()
```

```
array(['LOTICO'], dtype=object)
```

```
ndf[ndf['GRUPO'] == 'LOTICO']['SUBTIPO'].mode()
```

```
0    RIO
dtype: object
```

```
ndf['SUBTIPO'][ndf['SUBTIPO'].isnull()] = 'RIO'
```

```
ndf[columns_o].describe()
```

	CLAVE	ORGANISMO_DE_CUENCA	GRUPO	SUBTIPO	CUMPLE_CON_DBO	CUMPLE_CON_I
count	3493	3493	3493	3493	3493	3493

```
a = columns_o.tolist()
a.remove('CLAVE')
a.remove('SUBTIPO')

fig, axes = plt.subplots(4, 3, figsize =(10, 14))

for i in range(len(a)):
    plt.subplot(4,3,i+1)
    plt.hist(ndf[a[i]], 20)
    plt.title(a[i])

plt.show()
```



▼ Análisis de variables de tipo float

| 500 |

```
columns_f = ndf.columns[np.array(ndf.dtypes.astype(str)=='float64')]
ndf[columns_f].describe()
```

	LONGITUD	LATITUD
count	3493.000000	3493.000000
mean	-100.359969	21.046992
std	6.122773	3.893696
min	-117.124030	14.534910
25%	-103.882310	18.396070
50%	-99.795530	20.148980
75%	-96.860230	22.828930
max	-86.732150	32.706500

www | 1000 | 1000 |

```
g_df = ndf[['LONGITUD', 'LATITUD']]
g_df['Coordinates'] = list(zip(g_df['LONGITUD'], g_df['LATITUD']))
g_df['Coordinates'] = g_df['Coordinates'].apply(Point)
```

```
g_df = gpd.GeoDataFrame(g_df, geometry='Coordinates')
```

www | [0.84](#)

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

U.4.1

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

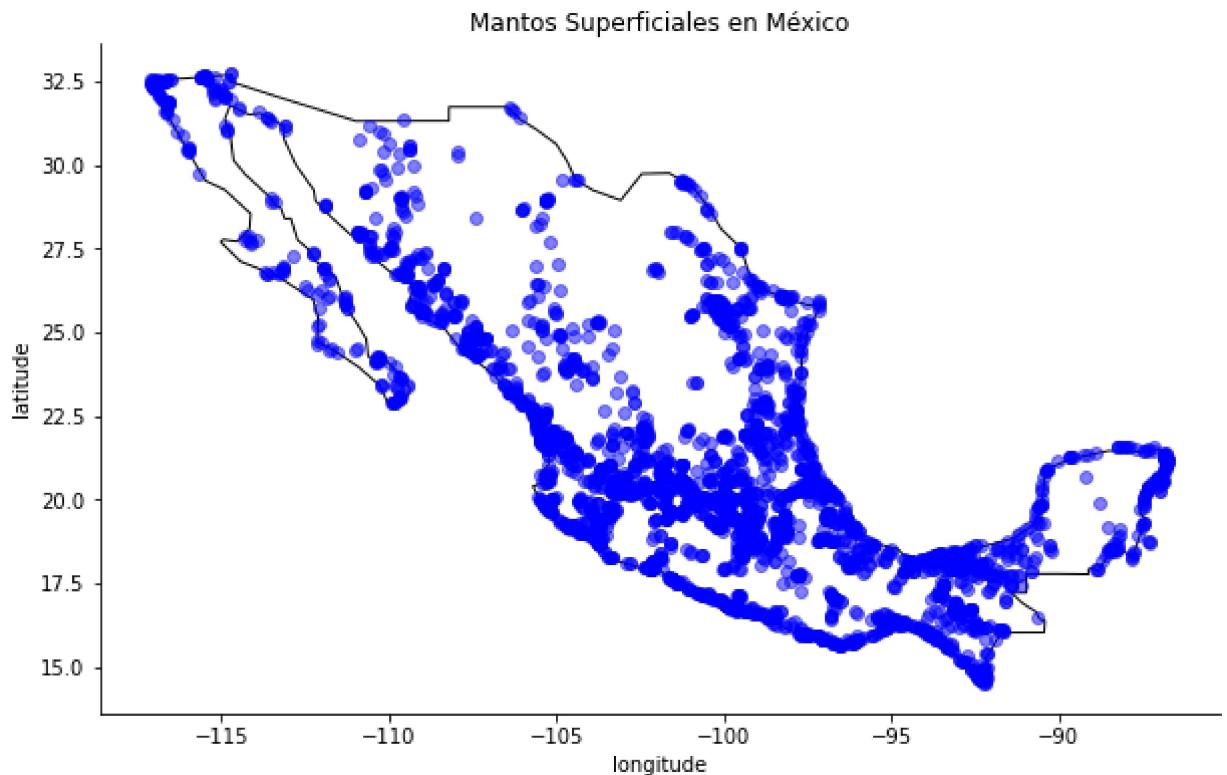
```
world.query("name == 'Mexico'").plot(ax = gax, edgecolor='black', color='white')
```

```
g = df.plot(ax=gax, color='blue', alpha=0.5)
```

```
gax.set_xlabel('longitude')
gax.set_ylabel('latitude')
gax.set_title('Mantos Superficiales en México')
```

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

```
plt.show()
```



▼ K-means

```

g_df = ndf[['LONGITUD', 'LATITUD']]
g_df['Coordinates'] = list(zip(g_df['LONGITUD'], g_df['LATITUD']))
g_df['Coordinates'] = g_df['Coordinates'].apply(Point)

g_df = gpd.GeoDataFrame(g_df, geometry='Coordinates')
g_df_v = g_df[ndf['SEMAFORO']=='Verde']
g_df_a = g_df[ndf['SEMAFORO']=='Amarillo']
g_df_r = g_df[ndf['SEMAFORO']=='Rojo']

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

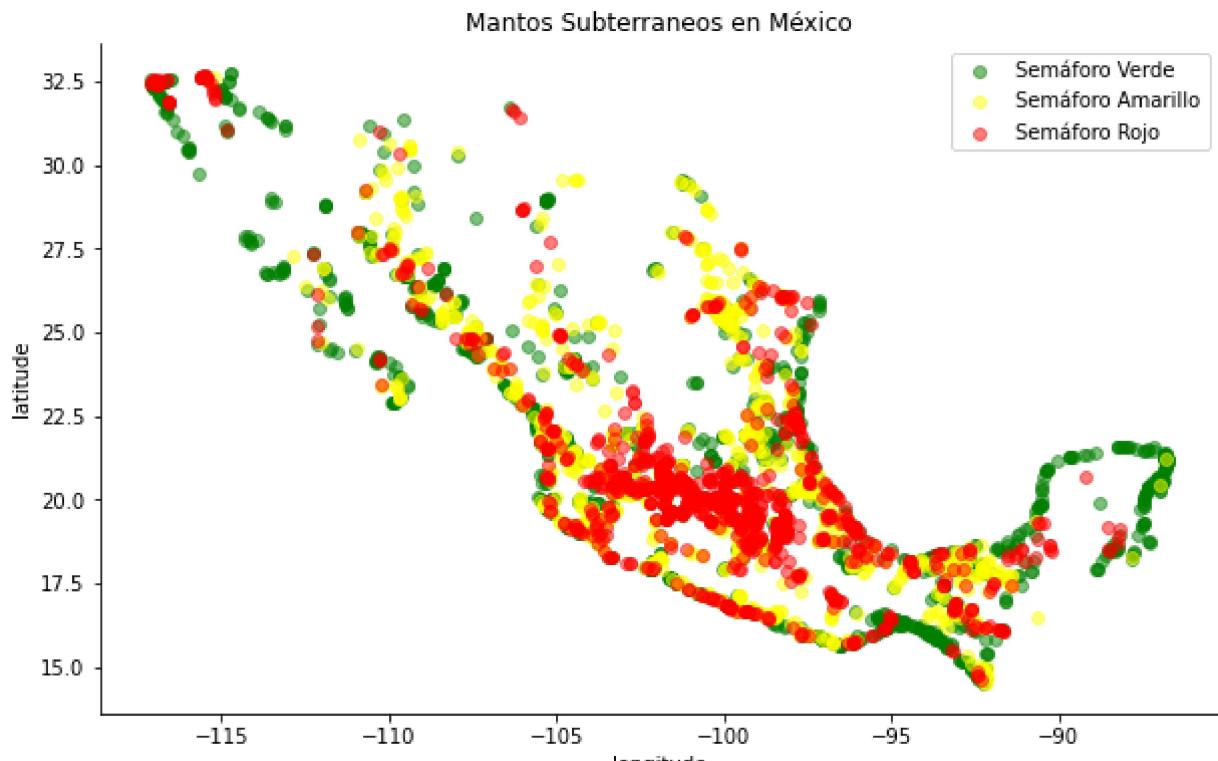
world.query("name == 'Mexico'").plot(ax = gax, edgecolor='black', color='white')

g_df_v.plot(ax=gax, color='green', alpha = 0.5, label='Semáforo Verde')
g_df_a.plot(ax=gax, color='yellow', alpha = 0.5, label='Semáforo Amarillo')
g_df_r.plot(ax=gax, color='red', alpha = 0.5, label='Semáforo Rojo')

gax.set_xlabel('longitude')
gax.set_ylabel('latitude')
gax.set_title('Mantos Subterráneos en México')

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

```



```

km_scores= []
km_silhouette = []

for i in range(2,11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(ndf[['LATITUD', 'LONGITUD']])
    classification = kmeans.predict(ndf[['LATITUD', 'LONGITUD']])

    km_scores.append(-kmeans.score(ndf[['LATITUD', 'LONGITUD']]))

    silhouette = silhouette_score(ndf[['LATITUD', 'LONGITUD']], classification)
    km_silhouette.append(silhouette)

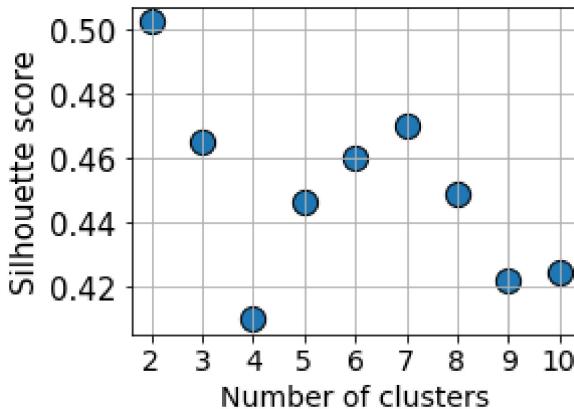
plt.figure(figsize=(4,3))
plt.scatter(x=[i for i in range(2,11)],y=km_scores,s=150,edgecolor='k')
plt.grid(True)
plt.xlabel("Number of clusters",fontsize=14)
plt.ylabel("K-means score",fontsize=15)
plt.xticks([i for i in range(2,11)],fontsize=14)
plt.yticks(fontsize=15)
plt.show()

```

```

plt.figure(figsize=(4,3))
plt.scatter(x=[i for i in range(2,11)],y=km_silhouette,s=150,edgecolor='k')
plt.grid(True)
plt.xlabel("Number of clusters",fontsize=14)
plt.ylabel("Silhouette score",fontsize=15)
plt.xticks([i for i in range(2,11)],fontsize=14)
plt.yticks(fontsize=15)
plt.show()

```



```

kmeans = KMeans(n_clusters=7)
kmeans.fit(ndf[['LATITUD', 'LONGITUD']])

centers = kmeans.cluster_centers_
centers = pd.DataFrame(centers, columns=['LATITUD', 'LONGITUD'])
centers['Coordinates'] = list(zip(centers['LONGITUD'], centers['LATITUD']))
centers['Coordinates'] = centers['Coordinates'].apply(Point)

g_centers = gpd.GeoDataFrame(centers, geometry='Coordinates')

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

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

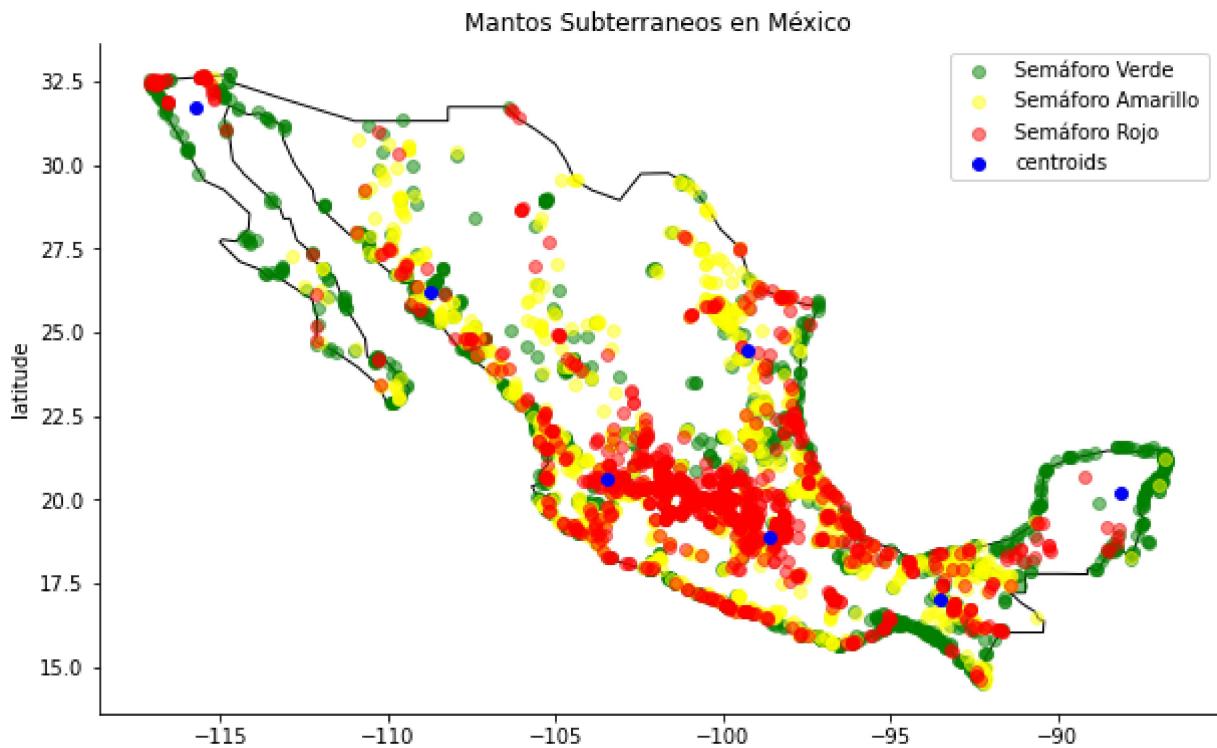
g_df_v.plot(ax=gax, color='green', alpha = 0.5, label='Semáforo Verde')
g_df_a.plot(ax=gax, color='yellow', alpha = 0.5, label='Semáforo Amarillo')
g_df_r.plot(ax=gax, color='red', alpha = 0.5, label='Semáforo Rojo')

g_centers.plot(ax=gax, color='blue', alpha = 1, label=str('centroids'))

gax.set_xlabel('longitude')
gax.set_ylabel('latitude')
gax.set_title('Mantos Subterraneos en México')

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

```



Para poder visualizar analizar de mejor manera la cantidad de puntos asignados a cada cluster, se raliza la siguiente gráfica cambiando la forma del punto.

```

clusters = kmeans.predict(ndf[['LATITUD', 'LONGITUD']])
ndf['clusters'] = clusters

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

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

markers = ["o", "^", "s", "P", "D", "X", "*"]
semaforos = ['Verde', 'Amarillo', 'Rojo']
colors = ['green', 'yellow', 'red']

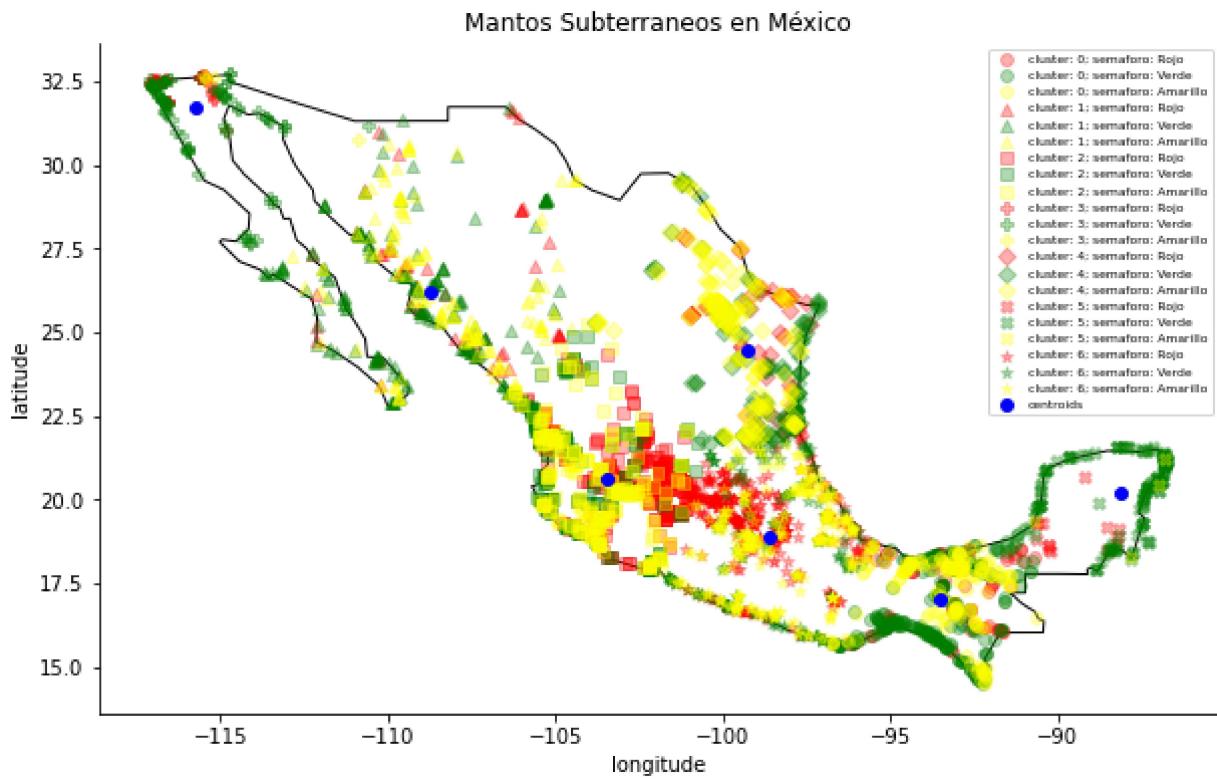
for cluster in range(len(ndf['clusters'].unique())):
    for semaforo in ndf['SEMAFORO'].unique():
        color = colors[semaforos.index(semaforo)]
        marker = markers[cluster]
        label = 'cluster: ' + str(cluster) + ' ; semaforo: ' + semaforo
        g_df[(ndf['clusters']==cluster) & (ndf['SEMAFORO']==semaforo)].plot(ax=gax, color=color)

g_centers.plot(ax=gax, color='blue', alpha = 1, label=str('centroids'))

gax.set_xlabel('longitude')
gax.set_ylabel('latitude')
gax.set_title('Mantos Subterraneos en México')

gax.spines['top'].set_visible(False)
gax.spines['right'].set_visible(False)
plt.legend(loc=1, prop={'size': 6})
plt.show()

```



A continuación se incluye un análisis del total de elementos por cluster y por color.

```
Dic = {}

Dic['semáforo'] = ['Verde', 'Amarillo', 'Rojo', 'Total']

for cluster in range(len(ndf['clusters'].unique())):
    Dic['cluster ' + str(cluster)] = [ndf[(ndf['SEMAFORO']=='Verde') & (ndf['clusters']==cluster)].shape[0],
                                       ndf[(ndf['SEMAFORO']=='Amarillo') & (ndf['clusters']==cluster)].shape[0],
                                       ndf[(ndf['SEMAFORO']=='Rojo') & (ndf['clusters']==cluster)].shape[0],
                                       ndf[ndf['clusters']==cluster].shape[0]
                                      ]

results = pd.DataFrame(Dic)
results = results.set_index('semáforo')
results.index.name = None
results
```

	cluster 0	cluster 1	cluster 2	cluster 3	cluster 4	cluster 5	cluster 6
Verde	276	212	205	87	134	148	205
Amarillo	188	148	275	5	189	4	326
Rojo	99	57	333	41	74	15	472
Total	563	417	813	133	397	167	1003

▼ Conclusiones - Parte 1

A partir del proceso de limpieza nos pudimos dar cuenta que la falta de datos a lo largo de toda la base de datos era recurrente, así como la duplicidad en los datos a tener la información de manera numérica y categórica al mismo tiempo. Se tomó la decisión de conservar solo las variables categóricas que resultaron de la evaluación de las condiciones.

El poder realizar gráficas y la tabla con las comparativas de los resultados de las clasificaciones se comprendió de mejor manera la colocación que le dio el K-means a cada uno de los puntos. En específico, se puede apreciar que el cluster 3 tiene mayoritariamente puntos verdes, lo cual se corrobora de manera visual en el mapa, donde la península de Yucatán muestra principalmente puntos de color verde.

Por otro lado, la zona del centro del país que corresponde a los clusters 0 y 2, sí agrupa de una manera especial a los puntos con semáforo rojo, aunque tiene de igual forma presencia de puntos verdes y amarillos.

En conclusión, para poder clasificar de forma correcta los puntos en relación al semáforo que tiene cada uno, no es suficiente la ubicación geográfica, ya que aunque existen ciertas agrupaciones que tienen una tendencia hacia cierto tipo de color de semáforo, existen otros en cuyas clases estos se encuentran muy parejos.

De manera general, con la clasificación realizada a partir de kmeans, se puede concluir que de encontrar un cuerpo acuático superficial en algún lugar del país, es probable que sea más seguro tomar de este a medida que estamos lejos del centro del país.

▼ Instrucciones - Parte 2

En esta base de datos encontrarás:

- Aguas subterráneas.
- Aguas superficiales.

Utilizando la base de datos que hayas elegido y hayas realizado su limpieza en la anterior entrega del reto, realiza lo siguiente:

- Selecciona tus variables independientes X y dependiente Y (semáforo)
- Cambia a label encoding el semáforo, ej, de ["clase 1", "clase 2", "clase 3"] a [1,2,3]
- Realiza un análisis general de las features importances a través de decision trees o random forest.
- Selecciona las variables de mayor importancia.
- Realiza tu clasificador, recuerda dividir los datos de manera balanceada (auxiliares de train test split)
- Explora qué clasificador es el más óptimo, ejemplo:
 - Decision trees

- Random Forest.
- Determina el grado de exactitud a través del reporte de clasificación https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
- Visualiza los resultados del modelo o las predicciones a través de una matriz de confusión.
- Realiza un reporte de los principales hallazgos y conclusiones del experimento e incluyelos en una presentación ejecutiva de 10 diapositivas. La presentación deberá incluir todos los pasos del pipeline seguidos, limpieza, análisis, kmeans, clasificación, resultados y conclusiones.

▼ Desarrollo - Parte 2

Se importan las librerías adicionales a utilizar:

```
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import learning_curve, validation_curve
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

Se comenzará por realizar algunas transformaciones para las variables que son categóricas. Al no ser binarias, se utiliza un LabelEncoder. También se retira la columna CLAVE como variable y se deja como ID

```
ndf = ndf.set_index('CLAVE')
ndf
```

	ORGANISMO_DE_CUENCA	GRUPO	SUBTIPO	LONGITUD	LATITUD	CUMPL
CLAVE						
DLAGU8	LERMA SANTIAGO PACIFICO	LENTICO	PRESA	-102.33911	22.24730	
DLBAJ100	PENINSULA DE BAJA CALIFORNIA	COSTERO	OCEANO-MAR	-109.84290	22.90473	
DLBAJ101	PENINSULA DE BAJA CALIFORNIA	COSTERO	OCEANO-MAR	-109.86442	22.89880	
DLBAJ102	PENINSULA DE BAJA CALIFORNIA	COSTERO	BAHIA	-109.88604	22.89609	
DLBAJ103	PENINSULA DE BAJA CALIFORNIA	COSTERO	BAHIA	-109.89657	22.87694	

Para las variables de SUPTIPO se convierten todas a mayúsculas a fin de no generar ruido en el LabelEncoder.

```
ndf['SUBTIPO'] = ndf['SUBTIPO'].str.upper()
```

OCRBR5208M1	RIO BRAVO	LOTICO	RIO	-99.50727	27.49901
-------------	-----------	--------	-----	-----------	----------

Se identifican las columnas de tipo objeto y de tipo flotante para poder realizar la transformación de label encoding.

```
ndf_o = ndf.columns[np.array(ndf.dtypes.astype(str)=='object')]
ndf_f = ndf.columns[np.array(ndf.dtypes.astype(str)=='float64')]
```

Se hace por medio de un ciclo for, ya que a través del column transformer no funcionó de manera correcta.

```
for column in ndf_o:
    le = preprocessing.LabelEncoder()
    le.fit(ndf[column])
    ndf[column] = le.transform(ndf[column])
```

```
ndf
```

CLAVE	ORGANISMO_DE_CUENCA	GRUPO	SUBTIPO	LONGITUD	LATITUD	CUMPLE_CON_
DLAGU8	6	1	19	-102.33911	22.24730	
DLBAJ100	10	0	18	-109.84290	22.90473	
DLBAJ101	10	0	18	-109.86442	22.89880	
DLBAJ102	10	0	1	-109.88604	22.89609	
DLBAJ103	10	0	1	-109.89657	22.87694	
...
OCRBR5206M1	12	2	20	-99.42142	26.78971	

Se hace mención que **0** es **AMARILLO**, **1** es **ROJO** y **2** es **VERDE**

Se separan las variables en X y Y

```
OCRBR5206M1    12    2    20  -99.42142  26.78971
```

```
X = ndf.drop(columns = ['SEMAFORO', 'clusters'])
Y = ndf[['SEMAFORO']]
```



Se crea la siguiente función que nos va a permitir graficar la curva de aprendizaje

```
def mi_CurvePlot(train_sizes_param_range, train_scores, val_scores, title, xlabel, ylabel)
    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)
    val_mean = np.mean(val_scores, axis=1)
    val_std = np.std(val_scores, axis=1)

    plt.figure(figsize=(7,6))
    plt.plot(train_sizes_param_range, train_mean, color='blue', marker='o', markersize=5,
    plt.fill_between(train_sizes_param_range, train_mean + train_std, train_mean - train_s

    plt.plot(train_sizes_param_range, val_mean, color='red', marker='+', markersize=5, lir
    plt.fill_between(train_sizes_param_range, val_mean + val_std, val_mean - val_std, alp[

    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.grid()
    plt.legend(loc='lower left')
    plt.show()
```

Se separan las X y Y en conjunto de entrenamiento y validación, con un 85% de los datos para entrenamiento.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.85, random_state=1)
```

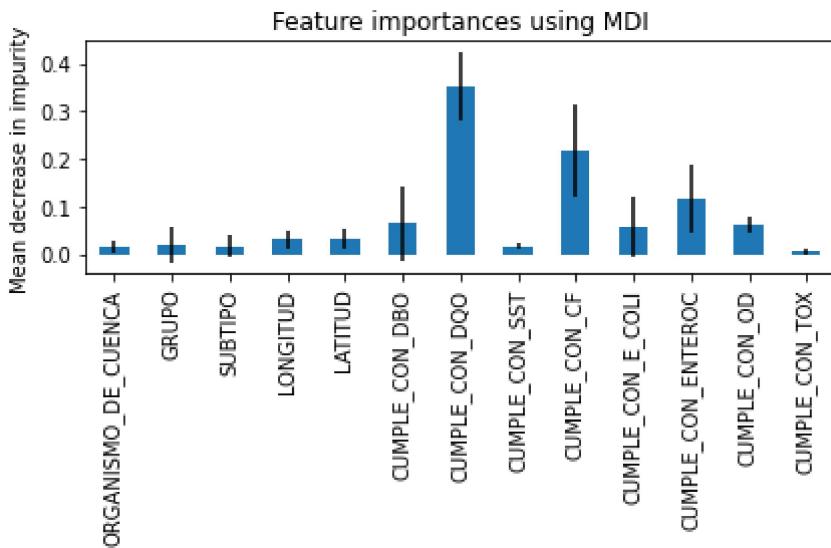
▼ Feature Importance

```
feature_names = X.columns
forest = RandomForestClassifier(random_state=0)
forest.fit(X_train, Y_train);

importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)

forest_importances = pd.Series(importances, index=feature_names)

fig, ax = plt.subplots()
forest_importances.plot.bar(yerr=std, ax=ax)
ax.set_title("Feature importances using MDI")
ax.set_ylabel("Mean decrease in impurity")
fig.tight_layout()
```



▼ Decision trees

```
#Usa los parámetros predeterminados del modelo.
modelo = DecisionTreeClassifier()

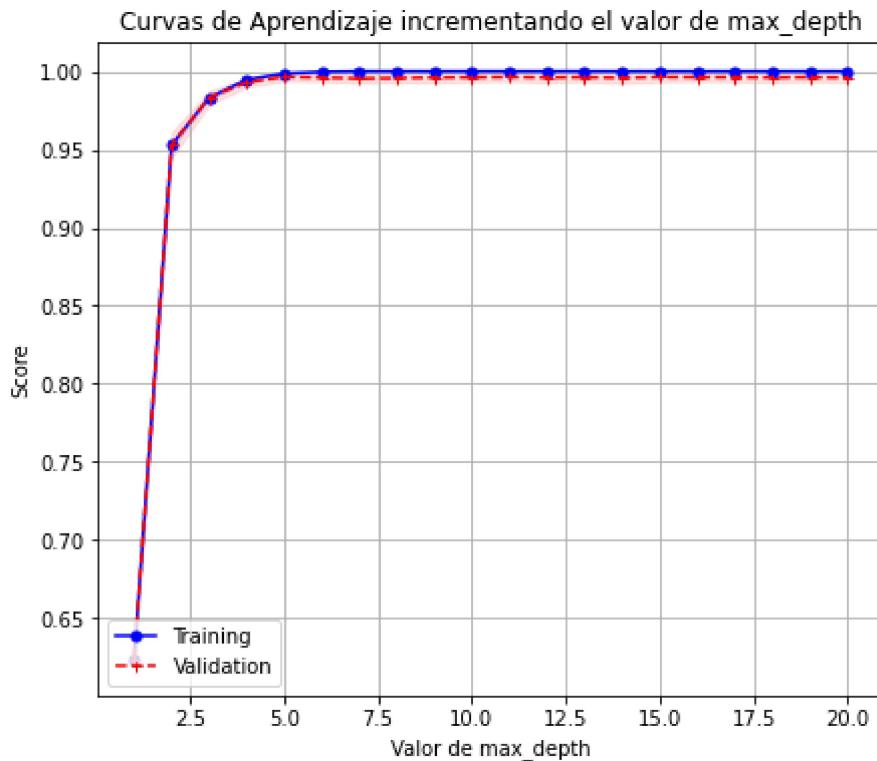
#Genera un arreglo con los 20 valores de los tamaños de muestra
mi_param_range = np.linspace(1, 20, num=20)

#Aplica validación cruzada estratificada y con repeticiones
kfold = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=7)

tr_scores, val_scores = validation_curve(estimator = modelo,
                                         X = X_train, y = Y_train,
                                         param_name = "max_depth",
                                         param_range = mi_param_range,
                                         cv = kfold,
```

```
n_jobs=-1)
```

```
# Obtenemos el gráfico con las curvas de aprendizaje:
mi_CurvePlot(mi_param_range, tr_scores, val_scores,
              title = 'Curvas de Aprendizaje incrementando el valor de max_depth',
              xlabel = 'Valor de max_depth',
              ylabel= 'Score'
            )
```



Se puede observar que a partir de una profundidad de 6, el sistema tiene un score de 1 de manera constante. Se realiza una evaluación sobre los datos de entrenamiento y se muestran resultados en una matriz de confusión.

```
modelo = DecisionTreeClassifier(max_depth=6)
modelo.fit(X_train, Y_train)
Y_hat = modelo.predict(X_train)

df_cm = confusion_matrix(Y_train, Y_hat)

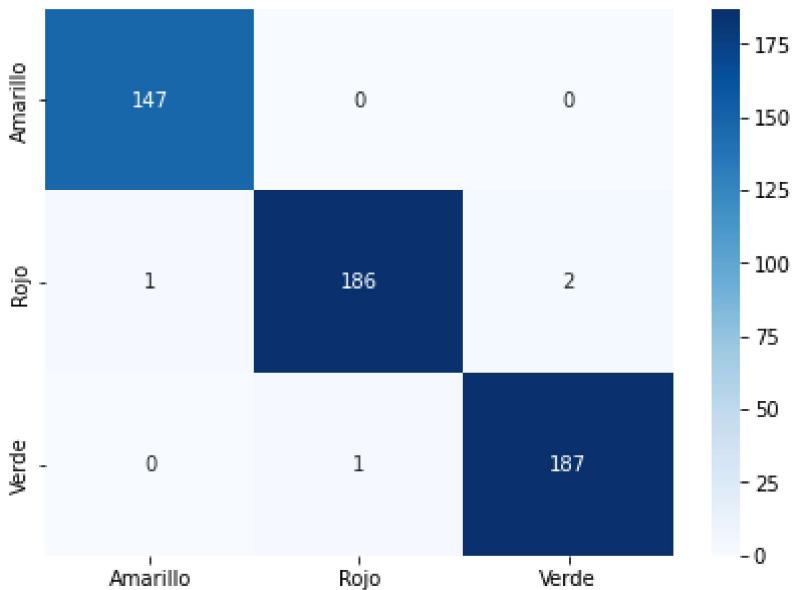
plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True, cmap='Blues', fmt='.4g', xticklabels=['Amarillo', 'Rojo'],
            yticklabels=['Amarillo', 'Rojo'])
plt.show()
```



Se realiza lo mismo para los datos de validación.

```
df_cm = confusion_matrix(Y_test, Y_hat)

plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True, cmap='Blues', fmt='.4g', xticklabels=['Amarillo', 'Rojo', 'Verde'], yticklabels=['Amarillo', 'Rojo', 'Verde'])
plt.show()
```



Por último se obtiene reporte de clasificación solicitado

```
target_names = ['Amarillo', 'Rojo', 'Verde']
print(classification_report(Y_test, Y_hat, target_names=target_names))
```

	precision	recall	f1-score	support
Amarillo	0.99	1.00	1.00	147
Rojo	0.99	0.98	0.99	189
Verde	0.99	0.99	0.99	188
accuracy			0.99	524
macro avg	0.99	0.99	0.99	524
weighted avg	0.99	0.99	0.99	524

▼ Random Forest

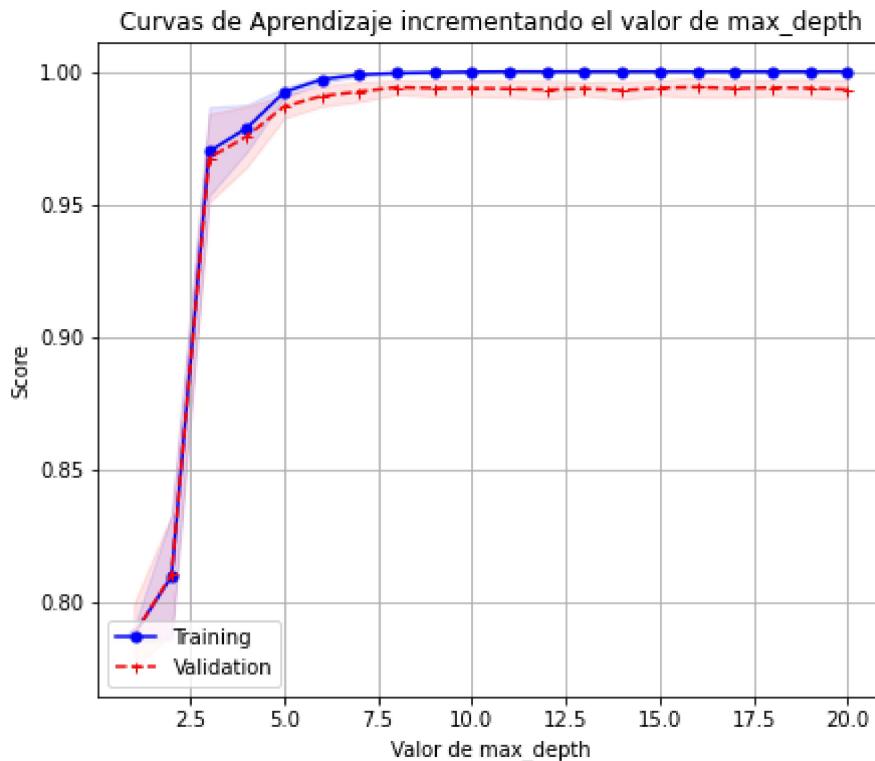
```
#Usa los parámetros predeterminados del modelo.
modelo = RandomForestClassifier()

#Genera un arreglo con los 20 valores de los tamaños de muestra
mi_param_range = np.linspace(1, 20, num=20)

#Aplica validación cruzada estratificada y con repeticiones
kfold = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=7)

tr_scores, val_scores = validation_curve(estimator = modelo,
                                         X = X_train, y = Y_train,
                                         param_name = "max_depth",
                                         param_range = mi_param_range,
                                         cv = kfold,
                                         n_jobs=-1)

# Obtenemos el gráfico con las curvas de aprendizaje:
mi_CurvePlot(mi_param_range, tr_scores, val_scores,
              title = 'Curvas de Aprendizaje incrementando el valor de max_depth',
              xlabel = 'Valor de max_depth',
              ylabel= 'Score'
              )
```

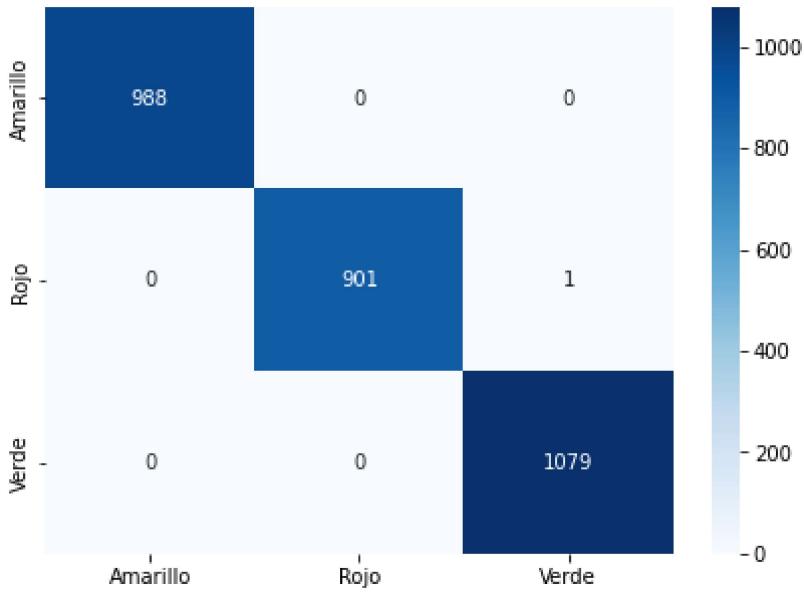


Se puede observar que a partir de una profundidad de 6, el sistema tiene un score de 1 de manera constante. Se realiza una evaluación sobre los datos de entrenamiento y se muestran resultados en una matriz de confusión.

```
modelo = RandomForestClassifier(max_depth=10)
modelo.fit(X_train, Y_train)
Y_hat = modelo.predict(X_train)

df_cm = confusion_matrix(Y_train, Y_hat)

plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True, cmap='Blues', fmt='.4g', xticklabels=['Amarillo', 'Rojo', 'Verde'],
            yticklabels=['Amarillo', 'Rojo', 'Verde'])
plt.show()
```



Se realiza lo mismo para los datos de validación.

```
Y_hat = modelo.predict(X_test)

df_cm = confusion_matrix(Y_test, Y_hat)

plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True, cmap='Blues', fmt='.4g', xticklabels=['Amarillo', 'Rojo', 'Verde'],
            yticklabels=['Amarillo', 'Rojo', 'Verde'])
plt.show()
```

Por último se obtiene reporte de clasificación solicitado

```
target_names = ['Amarillo', 'Rojo', 'Verde']
print(classification_report(Y_test, Y_hat, target_names=target_names))
```

	precision	recall	f1-score	support
Amarillo	0.99	0.99	0.99	147
Rojo	1.00	0.99	0.99	189
Verde	0.99	1.00	0.99	188
accuracy			0.99	524
macro avg	0.99	0.99	0.99	524
weighted avg	0.99	0.99	0.99	524

▼ Conclusiones - Parte 2

Realizar un análisis del FeatureImportance con el bosque aleatorio nos hizo ver que si bien la columna nombrada como 'CUMPLE_CON_DQO' es la que tiene más importancia en el modelo, todas tienen hasta cierto punto un impacto para este. Era de esperarse que las variables ya categorizadas a partir de los valores numéricos fueran más importantes, pero se puede ver que hay algunas variables que inician con la etiqueta 'CUMPLE' que resultaron tener menor importancia que algunas otras como el GRUPO, SUBTIPO, CUENTA e incluso las coordenadas geográficas.

Por otro lado, se puede observar que para el Árbol de decisiones y el Bosque Aleatorio se obtiene muy buenos resultados en el conjunto de entrenamiento y que estos se mantienen en muy buenos valores para los conjuntos de prueba. Del reporte de clasificación se observa que para Precisión, Recall y F1-Score, el valor más bajo en todas las clases es de 0.98. Esto es muy bueno, porque quiere decir que no sólo clasifica evitando falsos positivos, sino que también minimiza los falsos negativos. De igual forma la métrica de Accuracy es de 0.99 en el caso más bajo, lo cual hace que la evaluación de los modelos haya resultado de manera satisfactoria.

Colab paid products - [Cancel contracts here](#)

✓ 1s completed at 11:18 PM

