Pregunta

¿Se puede predecir a que tipo de especie pertenece cada pescado, dependiendo de su tamaño fisico (Altura, Ancho, Largo)?

Limpiando data (Data cleaning)

In [92]:
```
# Ignora los warnings
import warnings
warnings.filterwarnings("ignore")
```

In [93]:
```
#Importamos librerias
import pandas as pd # Manipulación de datos
import numpy as np # Funciones matemáticas
```

In [94]:
```
# delimiter: Es un delimitador que se usa para eliminar las tabulaciones que se encuen
# Aginando dataframe a variable "data"
data = pd.read_csv("/content/Body_measurements_diet.txt",delimiter = "\t")
```

In [95]:
```
# Muestra las 10 primeras filas
data.head(10)
```

Out[95]:

	Lake	LC	Date	Species	Fishing_Zone	Gear	No	TL	BH	BW	...	SIAcode	Peagic_clad
0	Motjennet	2	2018-07-04	Crucian	L1	BT	1.0	10.4	2.9	17.4	...	NaN	
1	Motjennet	2	2018-07-04	Crucian	L1	BT	2.0	9.4	2.5	12.2	...	NaN	
2	Motjennet	2	2018-07-04	Crucian	L1	BT	3.0	10.9	2.7	17.6	...	NaN	
3	Motjennet	2	2018-07-04	Crucian	L1	BT	4.0	15.0	4.4	54	...	A2L1BT4	
4	Motjennet	2	2018-07-04	Crucian	L1	BT	5.0	10.0	2.5	14.6	...	A2L1BT5	
5	Motjennet	2	2018-07-04	Crucian	L1	BT	6.0	11.3	3.0	21.1	...	NaN	
6	Motjennet	2	2018-07-04	Crucian	L1	BT	7.0	12.1	3.6	32.2	...	A2L1BT7	
7	Motjennet	2	2018-07-04	Crucian	L1	BT	8.0	9.3	2.5	12.2	...	A2L1BT8	
8	Motjennet	2	2018-07-04	Crucian	L1	BT	9.0	11.4	3.1	21.4	...	NaN	
9	Motjennet	2	2018-07-04	Crucian	L1	BT	10.0	11.6	3.0	22	...	A2L1BT10	

10 rows × 21 columns

In [96]:
```
# Numero de filas X columnas
data.shape
```

Out[96]:
```
(3134, 21)
```

In [97]:
```
# Describción eestadistica del dataset
data.describe()
```

Out[97]:

	LC	No	TL	BH	Peagic_cladocerans	Copepods	Benthic_cl
count	3134.000000	2738.000000	3132.000000	2811.000000	417.000000	417.000000	4
mean	6.941289	35.574142	15.573225	4.547862	32.731607	3.352590	
std	3.365831	37.066734	7.906200	3.141894	36.965353	12.355106	
min	1.000000	1.000000	2.100000	0.600000	0.000000	0.000000	
25%	4.000000	9.000000	9.800000	2.470000	0.000000	0.000000	
50%	7.000000	22.000000	14.000000	3.600000	15.000000	0.000000	
75%	10.000000	48.000000	18.800000	5.550000	62.500000	0.000000	
max	12.000000	174.000000	61.000000	17.000000	100.000000	100.000000	

In [98]:
```
#b = data.loc[(data['Lake'] == "Svartkulp") & (data['Species'] == "Crucian") & (data['
#b
```

In [99]:
```
#Información del dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3134 entries, 0 to 3133
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Lake                    3134 non-null   object 
 1   LC                      3134 non-null   int64  
 2   Date                    3134 non-null   object 
 3   Species                 3134 non-null   object 
 4   Fishing_Zone            3134 non-null   object 
 5   Gear                    3134 non-null   object 
 6   No                      2738 non-null   float64
 7   TL                      3132 non-null   float64
 8   BH                      2811 non-null   float64
 9   BW                      2958 non-null   object 
 10  SEX                     2045 non-null   object 
 11  SIAcode                 408 non-null    object 
 12  Peagic_cladocerans      417 non-null    float64
 13  Copepods                417 non-null    float64
 14  Benthic_cladocerans     417 non-null    float64
 15  Chironomid_larvae       417 non-null    float64
 16  Benthic_invertebrates   417 non-null    float64
 17  Sediment                417 non-null    float64
 18  Plant                   417 non-null    float64
 19  Pelagic_invertebrates   417 non-null    float64
 20  Gastropods              417 non-null    float64
dtypes: float64(12), int64(1), object(8)
memory usage: 514.3+ KB
```

Cambiando tipo de dato de columnas "object" a numericas

In [100…
```
#Funcion que permite reenombrar datos de cierta columna
def imputacion_categorica (df,columna,busqueda,reemplazar):
  df[columna] = np.where(df[columna] == busqueda,reemplazar,df[columna])
  return df[columna]
```

A la hora de cambiar el tipo de dato "object" a numerico en la variable "BW" se encontro un datos inconsistente.

In [101…
```
#Encontramos un valor inconsistente
data.loc[data['BW'] == "3..8"]
```

Out[101]:

	Lake	LC	Date	Species	Fishing_Zone	Gear	No	TL	BH	BW	...	SIAcode	Peagic_clad
1533	Svartkulp	5	2019-05-29	Minnow		L1	ST	21.0	7.4	1.4	3..8	...	NaN

1 rows × 21 columns

In [102…
```
#Se asigna correctamente el valor
data["BW"] = imputacion_categorica(data,"BW","3..8","3.8")
```

In [103…
```
#Cambiar tipo de dato de una columna
# Todos se pasan a tipo de dato numerico excepto Departamento y Municipio
def cambiotextnum(df, nomcol, tipo):
    df[nomcol] = df[nomcol].astype(tipo)
    return df[nomcol]

data['BW'] = cambiotextnum(data, 'BW','float64')
data['Date'] = cambiotextnum(data, 'Date','datetime64[ns]')
```

Como la columna Date son fechas y estan de tipo object, lo que se hace es pasar los datos a tipo de fecha y solo vamos a tomar el año

In [104…
```
#data['Date'] = data['Date'].dt.year
```

In [105…
```
#data["Date"].unique()
```

Eliminamos la columna "SIAcode" porque son identificadores unicos, entonces no nos serviria para la predicción más adelante

In [106…
```
#Eliminando columna "SIAcode"
data.drop(columns=["SIAcode"], inplace=True)
```

In [107…
```
#Nuevas filas X columnas
data.shape
```

Out[107]:
```
(3134, 20)
```

In [108…
```
# Muestras todas las columnas
data.columns
```

Out[108]:
```
Index(['Lake', 'LC', 'Date', 'Species', 'Fishing_Zone', 'Gear', 'No', 'TL',
       'BH', 'BW', 'SEX', 'Peagic_cladocerans', 'Copepods',
       'Benthic_cladocerans', 'Chironomid_larvae',
       'Benthic_invertebrates            ', 'Sediment', 'Plant',
       'Pelagic_invertebrates', 'Gastropods'],
      dtype='object')
```

Encontrando valores NaN que se encuentran en el dataset

In [109…
```
def datosnan(df):

# isinstance es una funcion de pandas
  if isinstance(df,pd.DataFrame): # Para que valide que lo que ingresa sea un DataFram
   total_nan = df.isna().sum().sum() # Totaliza todos los datos NaN

# %d : Permite imprimir la cantida de %d filas O %d columnas
   print("Dimensiones: %d filas, %d columnas" % (df.shape[0],df.shape[1])) # df.shape[
   print("Total de datos NaN: %d" % total_nan) # Concatena la valiable total_nan

  # %30s = Permite correr la tabla mas hacia la izquierda o derecha
   print("%40s %16s %15s %10s" % ("Nombre columna", "Tipo de dato", "# Unicos","Cantic

#Creamos variables
   nombre_columnas = df.columns # sacar nombres de columnas
   dtype = df.dtypes # Saca los tipos de datos
   unicos = df.nunique() #Saca los valores unicos
   valores_nan = df.isna().sum() # Saca los valores NaN

   # Hace un recorrido por todas las columnas
   for i in range(len(df.columns)): #Arranca des de 0 hasta el total de las datos que
     print("%40s %16s %15s %10s" % (nombre_columnas[i],dtype[i],unicos[i],valores_nan[

  else:
    print("Se  esperaba un dataframe: %15s" % (type(df)))
```

In [110…
```
#Muestra los datos NaN que hay en cada columna
datosnan(data)
```

```
Dimensiones: 3134 filas, 20 columnas
Total de datos NaN: 26439
                          Nombre columna     Tipo de dato        # Unicos Cantidad va
lores NaN
                                    Lake           object              12          0
                                      LC            int64              12          0
                                    Date   datetime64[ns]              33          0
                                 Species           object              11          0
                           Fishing_Zone           object               3          0
                                    Gear           object              11          0
                                      No          float64             174        396
                                      TL          float64             376          2
                                      BH          float64             636        323
                                      BW          float64            1122        176
                                     SEX           object               3       1089
                     Peagic_cladocerans          float64              27       2717
                              Copepods           float64              20       2717
                    Benthic_cladocerans          float64              29       2717
                      Chironomid_larvae          float64              22       2717
      Benthic_invertebrates                     float64              30       2717
                           Sediment           float64              19       2717
                              Plant           float64              14       2717
                Pelagic_invertebrates          float64              12       2717
                           Gastropods          float64               6       2717
```

Encontramos que la columna "Benthic_invertebrates" tiene espacios que pueden generar problemas

In [111…
```
# Reenombrando la variable
data = data.rename(columns={'Benthic_invertebrates           ': 'Benthic_invertebrate
```

Tambien se encontro que en algunas variables categoricas no coincide el numero de datos unicos con lo que se describe en el dataset

En la variable "Species" solo se registran 9 especies, pero en el dataset se encontraron 11.

In [112…
```
#Revisando valores unicos de la columna "Species"
data["Species"].unique()
```

Out[112]:
```
array(['Crucian', 'Bream', 'PerchL', 'Pike', 'Roach', 'Perch', 'Trout',
       'TroutL', 'Tench', 'Minnow', 'Rudd'], dtype=object)
```

Las especies que deben aparecer en el dataset son: **Crucian** , **Trout**, **Perch**, **Pike** , **Roach** , **Bream** , **Rudd** , **Minnow** , **Tench**.

In [113…
```
# Reenombrando correctamente las columnas
data["Species"] = imputacion_categorica(data,"Species","PerchL","Perch")
data["Species"] = imputacion_categorica(data,"Species","TroutL","Trout")
```

Lo mismo pasa con la variable "Gear" se registraron 6 equipos de pesca y hay 11.

Las equipos de pesca que deben aparecer en el dataset son: **N** , **BMS** , **SMS** , **ST** ,**MT** , **B**T .

In [114…
```
#Revisando valores unicos de la columna "Gear"
data["Gear"].unique()
```

Out[114]:
```
array(['BT', 'MT', 'N', 'ST', 'MT18', 'N14', 'F', 'BMS', 'N9', 'N25',
       'SMS'], dtype=object)
```

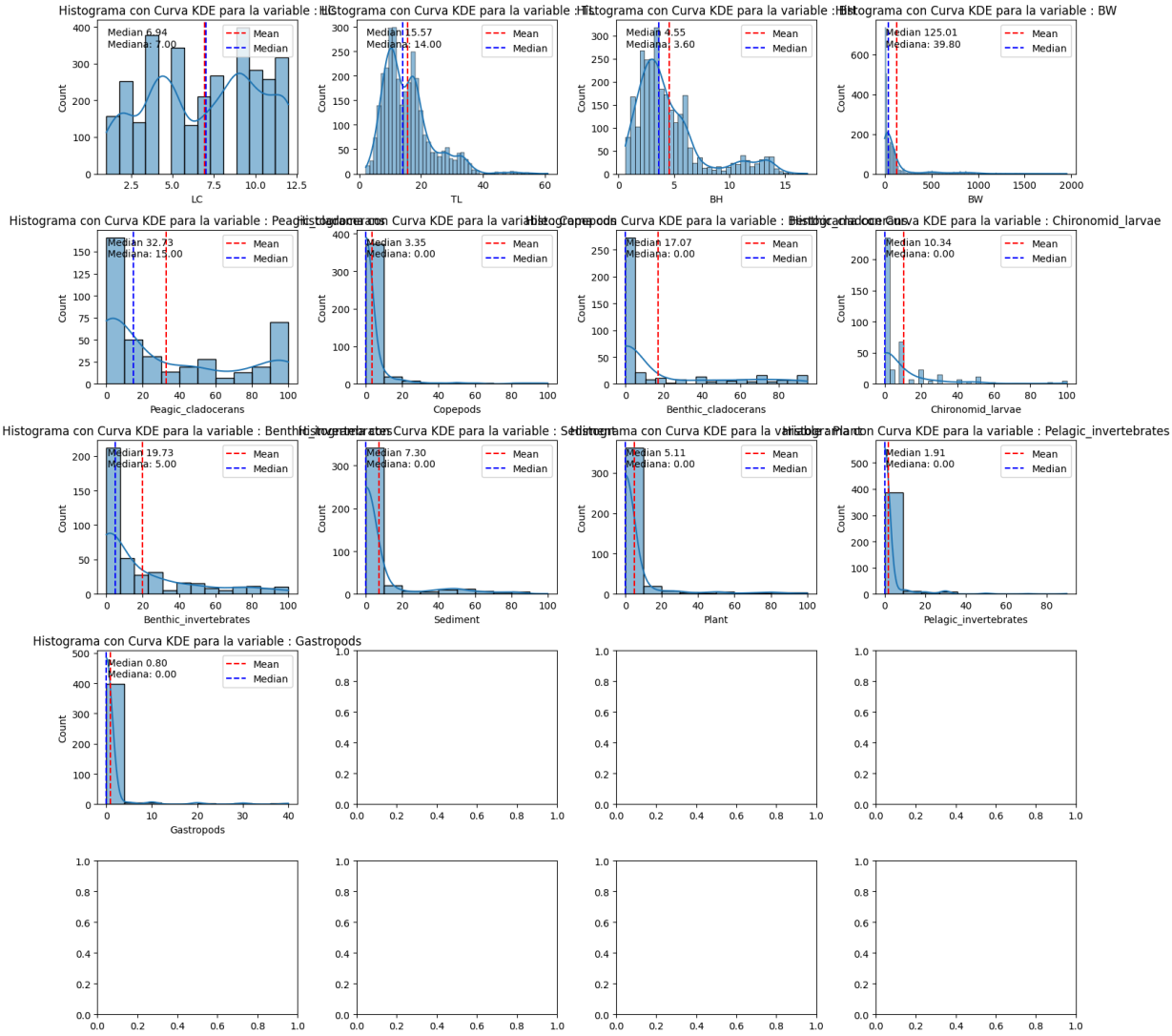
In [115…
```
# Reenombrando correctamente las columnas
data["Gear"] = imputacion_categorica(data,"Gear","MT18","MT")
data["Gear"] = imputacion_categorica(data,"Gear","N14","N")
data["Gear"] = imputacion_categorica(data,"Gear","N25","N")
data["Gear"] = imputacion_categorica(data,"Gear","N9","N")
```

In [116…
```
#Reemplazando los valores de F a NaN
data.loc[data['Gear'] == "F", 'Gear'] = np.nan
```

Modificando los valores NaN del dataset

Grafica de barras con su densidad y media y mediana respectiva

In [117…
```
import matplotlib.pyplot as plt
import seaborn as sns

#Asignamos las columnas del dataset a la variable "columnas"
columnas = ['LC', 'TL',
       'BH', 'BW', 'Peagic_cladocerans', 'Copepods',
       'Benthic_cladocerans', 'Chironomid_larvae', 'Benthic_invertebrates',
       'Sediment', 'Plant', 'Pelagic_invertebrates', 'Gastropods']

fig,axes = plt.subplots(nrows=5, ncols = 4, figsize = (15,15))

axes = axes.flatten()

for i, var in enumerate(columnas):
  ax = axes[i]
  sns.histplot(data[var],kde=True,ax=ax)
  ax.axvline(data[var].mean(), color = "red", linestyle = "--", label ="Mean")
  ax.axvline(data[var].median(), color = "blue", linestyle = "--", label ="Median")

  ax.annotate (f'Median {data[var].mean():.2f}\nMediana: {data[var].median():.2f}',
               xy = (0.05,0.95), xycoords = "axes fraction",ha = "left", va = "top")

  ax.set_title(f'Histograma con Curva KDE para la variable : {var}')
  ax.set_xlabel(var)
  ax.legend()

plt.tight_layout()
plt.show()
```



Vemos que todos los valores NaN de las variables numéricas se pueden reemplazar con la mediana.

In [118…

```
#Funcion que permita modificar los datos NaN con alguna mededida de tendencia central

def imputacion(df,columnas,parametro):
  if parametro =="mediana": #Mediana
    df[columnas] = df[columnas].fillna(df[columnas].median())
    return df[columnas]

  elif parametro=="media": #Media
    df[columnas]= df[columnas].fillna(df[columnas].mean())
    return df[columnas]

  elif parametro =="moda":#Moda (Solo para variables categoricas)
    df[columnas] = df[columnas].fillna(df[columnas].mode()[0]) # En la posicion [0]. s
    return df[columnas]

  else:
    print("Parametro no exite")
```

In [119…

```
#Mediana
data["No"] = imputacion(data,"No","mediana")
data["TL"] = imputacion(data,"TL","mediana")
```

```
data["BH"] = imputacion(data,"BH","mediana")
data["BW"] = imputacion(data,"BW","mediana")
data["Peagic_cladocerans"] = imputacion(data,"Peagic_cladocerans","mediana")
data["Copepods"] = imputacion(data,"Copepods","mediana")
data["Benthic_cladocerans"] = imputacion(data,"Benthic_cladocerans","mediana")
data["Chironomid_larvae"] = imputacion(data,"Chironomid_larvae","mediana")
data["Benthic_invertebrates"] = imputacion(data,"Benthic_invertebrates","mediana")
data["Sediment"] = imputacion(data,"Sediment","mediana")
data["Plant"] = imputacion(data,"Copepods","mediana")
data["Pelagic_invertebrates"] = imputacion(data,"Pelagic_invertebrates","mediana")
data["Gastropods"] = imputacion(data,"Gastropods","mediana")

#Moda
data["SEX"] = imputacion(data,"SEX","moda")
data["Gear"] = imputacion(data,"Gear","moda")
```

In [120…

```
#Mostrando valores NaN
datosnan(data)
```

```
Dimensiones: 3134 filas, 20 columnas
Total de datos NaN: 0
                          Nombre columna     Tipo de dato        # Unicos Cantidad va
lores NaN
                                    Lake           object              12           0
                                      LC            int64              12           0
                                    Date   datetime64[ns]              33           0
                                 Species           object               9           0
                           Fishing_Zone           object               3           0
                                    Gear           object               6           0
                                      No          float64             174           0
                                      TL          float64             376           0
                                      BH          float64             636           0
                                      BW          float64            1122           0
                                     SEX           object               3           0
                     Peagic_cladocerans          float64              27           0
                                Copepods          float64              20           0
                     Benthic_cladocerans          float64              29           0
                       Chironomid_larvae          float64              22           0
                  Benthic_invertebrates          float64              30           0
                               Sediment          float64              19           0
                                  Plant          float64              20           0
                  Pelagic_invertebrates          float64              12           0
                             Gastropods          float64               6           0
```

Graficas

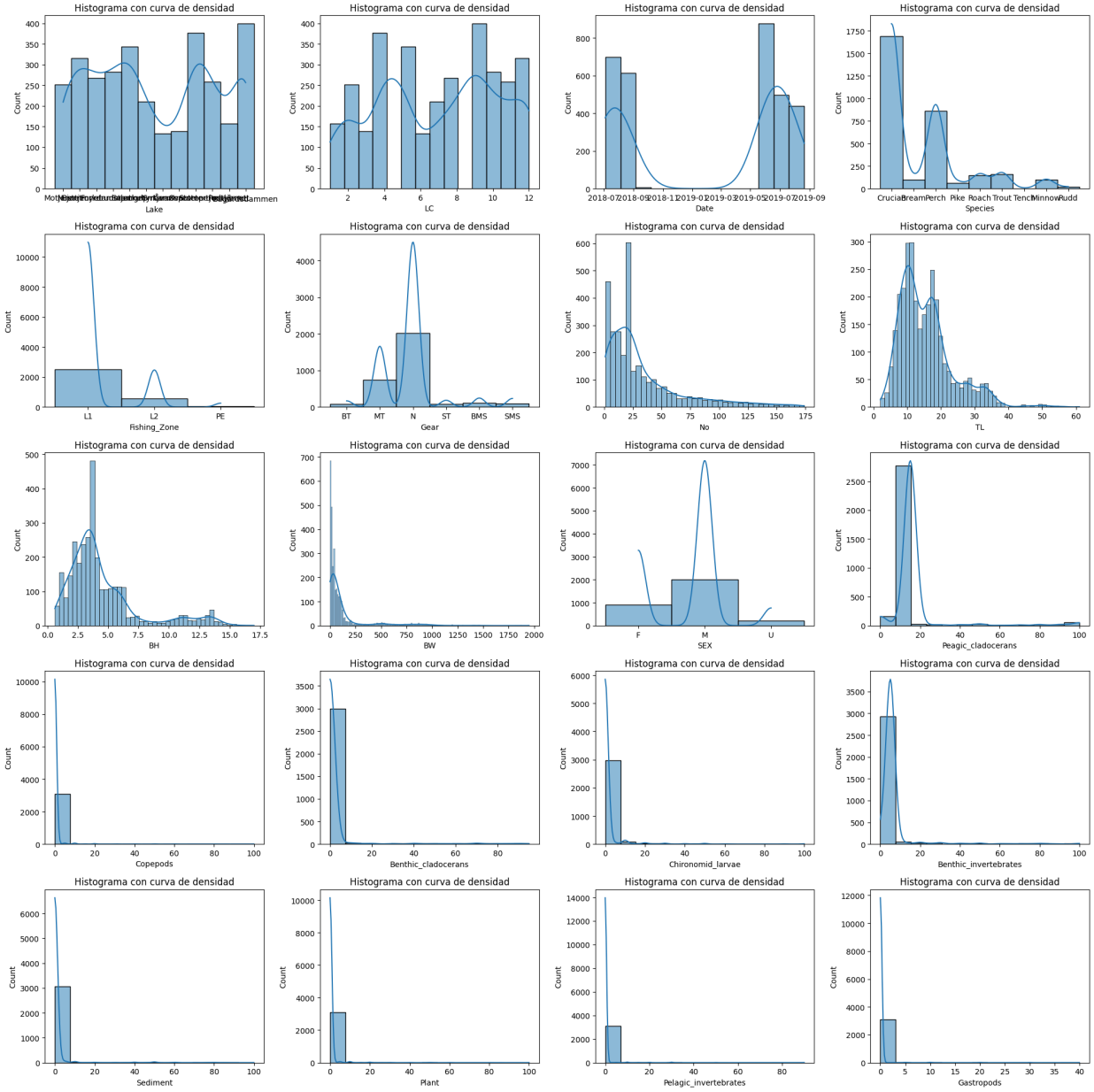
Distribución de los datos con su densidad

Vemos que toca normalizar la data porque tiene desviaciones estandar muy apartadas unas de otras y sus densidades no cumplen o no forma la campana Gaussiana.

In [121…

```
import matplotlib.pyplot as plt

import seaborn as sns
#Creamos grafico de barras con su densidad la cual permite ver la distribución de los
```

```
#Asignamos las columnas del dataset a la variable "columnas"
columnas = ['Lake', 'LC', 'Date', 'Species', 'Fishing_Zone', 'Gear', 'No', 'TL',
       'BH', 'BW', 'SEX', 'Peagic_cladocerans', 'Copepods',
       'Benthic_cladocerans', 'Chironomid_larvae', 'Benthic_invertebrates',
       'Sediment', 'Plant', 'Pelagic_invertebrates', 'Gastropods']

fig, axes = plt.subplots(nrows=5,ncols=4,figsize=(20,20))#nrows:# filas - ncols:#colum
axes = axes.flatten()

for i, var in enumerate(columnas):
  ax = axes[i]
  sns.histplot(data[var],kde=True,ax=ax)
  ax.set_title("Histograma con curva de densidad")
  ax.set_xlabel(var)

plt.tight_layout()
plt.show()
```



Distribucion de peces en cada laguna clasificados por su especie

In [122…

```
#Libreria para graficar
import plotly.express as px

#Agrupar peces en cada lago clasificados por su especie
ax1 = data.groupby(["Lake"])["Species"].value_counts().unstack()

# Diagrama de barras interactivo
fig = px.bar(ax1, barmode='group')

# Establecer titulos
fig.update_layout(title="Conteo de especies por lago clasificados por su especie",
                  xaxis_title="Lago",
                  yaxis_title="Conteo")

# Mostrar grafica
fig.show()
```

Distribucion de peces por cada especie clasificados por la profundidad de pesca

In [123…

```
#Libreria para graficar
import plotly.express as px
```

```
#Agrupar por numero de peces por cada especie clasificados por la profundidad de pesca
ax2 = data.groupby(["Fishing_Zone"])["Species"].value_counts().unstack()

# Diagrama de barras interactivo
fig = px.bar(ax2, barmode='stack')

# Establecer titulos
fig.update_layout(title="Conteo de especies por profundidad clasificado por especies",
                  xaxis_title="Pofundidad",
                  yaxis_title="Conteo")

# Mostrar grafica
fig.show()
```

Arte utilizado para atrapar a los diferentes tipos de peces

In [124…

```
#Libreria para graficar
import plotly.express as px

#Agrupar por el arte utilizado para atrapar a los diferentes tipos de peces
ax3 = data.groupby(["Gear"])["Species"].value_counts().unstack()

# Diagrama de barras interactivo
```

```
fig = px.bar(ax3, barmode='stack')

# Establecer titulos
fig.update_layout(title="Conteo de especies por arte de pesca",
                  xaxis_title="Arte de pesca",
                  yaxis_title="Conteo")

# Mostrar grafica
fig.show()
```

Distribucion de peces segun su género

In [125…

```
#Libreria para graficar
import plotly.express as px

#Agrupar peces segun su genero
ax4 = data.groupby(["SEX"])["Species"].value_counts().unstack()

# Diagrama de barras interactivo
fig = px.bar(ax4, barmode='stack')

# Establecer titulos
fig.update_layout(title="Conteo de especies por arte de pesca",
                  xaxis_title="Arte de pesca",
```

```
                  yaxis_title="Conteo")

# Mostrar grafica
fig.show()
```

Media de longitud (cm) de las especies

In [126…

```
#Agrupar peces segun su especie y saca la media de longitud de las diferentes especies
ax5 = data.groupby('Species')['TL'].mean().reset_index().round(2)

# Diagrama de barras interactivo
fig = px.bar(ax5, x='Species', y='TL',
             title='Media de longitud por especie (cm)',
             labels={'TL': 'Media de la longitud', 'Species': 'Especie'})

# Mostrar grafica
fig.show()
```

Media de la altura (cm) de las especies

In [127…

```
# Agrupar peces segun su especie y saca la media de altura de las diferentes especies
ax6 = data.groupby('Species')['BH'].mean().reset_index().round(2)

# Diagrama de barras interactivo
fig = px.bar(ax6, x='Species', y='BH',
             title='Media de altura por especie (cm)',
             labels={'TL': 'Media de altura', 'Species': 'Especie'})

# Mostrar grafica
fig.show()
```

Media de peso(g) de las especies

In [128…

```
# Agrupar peces segun su especie y saca la media de peso de las diferentes especies
ax7 = data.groupby('Species')['BW'].mean().reset_index().round(1)

# Diagrama de barras interactivo
fig = px.bar(ax7, x='Species', y='BW',
             title='Media de altura por especie (g)',
             labels={'TL': 'Media de peso', 'Species': 'Especie'})

# Mostrar grafica
fig.show()
```

One Hot Encoder

Pasamos las variables categoricas a numericas con One Hot Encoder para poder trabajar regresion Logistica Multiclase

In [129…

```
#Columanas categoricas : "SEX" - "Fishing_Zone" - "Gear" - "Lake"
#Aplicando Label Encoder

#Importamos libreria
from sklearn import preprocessing
#Aplicando Label Enconder
label = preprocessing.LabelEncoder()

data["Date"] = label.fit_transform(data["Date"])
data["SEX"] = label.fit_transform(data["SEX"])
data["Fishing_Zone"] = label.fit_transform(data["Fishing_Zone"])
data["Gear"] = label.fit_transform(data["Gear"])
data["Lake"] = label.fit_transform(data["Lake"])
```

In [130…

```
data.head()
```

Out[130]:

	Lake	LC	Date	Species	Fishing_Zone	Gear	No	TL	BH	BW	SEX	Peagic_cladocerans	Copep
0	5	2	0	Crucian	0	1	1.0	10.4	2.9	17.4	0	15.0	
1	5	2	0	Crucian	0	1	2.0	9.4	2.5	12.2	1	15.0	
2	5	2	0	Crucian	0	1	3.0	10.9	2.7	17.6	1	15.0	
3	5	2	0	Crucian	0	1	4.0	15.0	4.4	54.0	0	15.0	
4	5	2	0	Crucian	0	1	5.0	10.0	2.5	14.6	1	15.0	

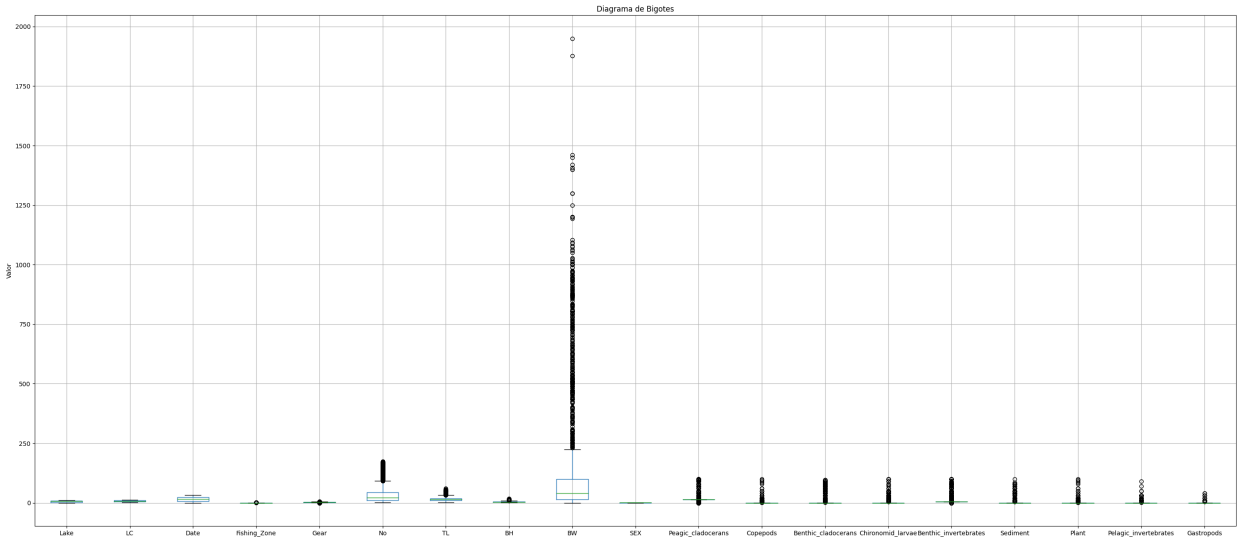
Ruido de la data en un diagrama de bigotes

In [131…

```
#Asignamos las columnas del dataset a la variable "columnas"
columnas = ['Lake', 'LC', 'Date', 'Species', 'Fishing_Zone', 'Gear', 'No', 'TL',
       'BH', 'BW', 'SEX', 'Peagic_cladocerans', 'Copepods',
       'Benthic_cladocerans', 'Chironomid_larvae', 'Benthic_invertebrates',
       'Sediment', 'Plant', 'Pelagic_invertebrates', 'Gastropods']
#Creamos diagrama de bigotes para ver los datos atipicos del dataset de cada columna
data[columnas].boxplot(figsize=(35, 15))
plt.title('Diagrama de Bigotes')
plt.ylabel('Valor')
```

Out[131]:

```
Text(0, 0.5, 'Valor')
```



Se puede evidenciar que existe mucho ruido en las variables, lo que es recomendable normalizar el dataset

---

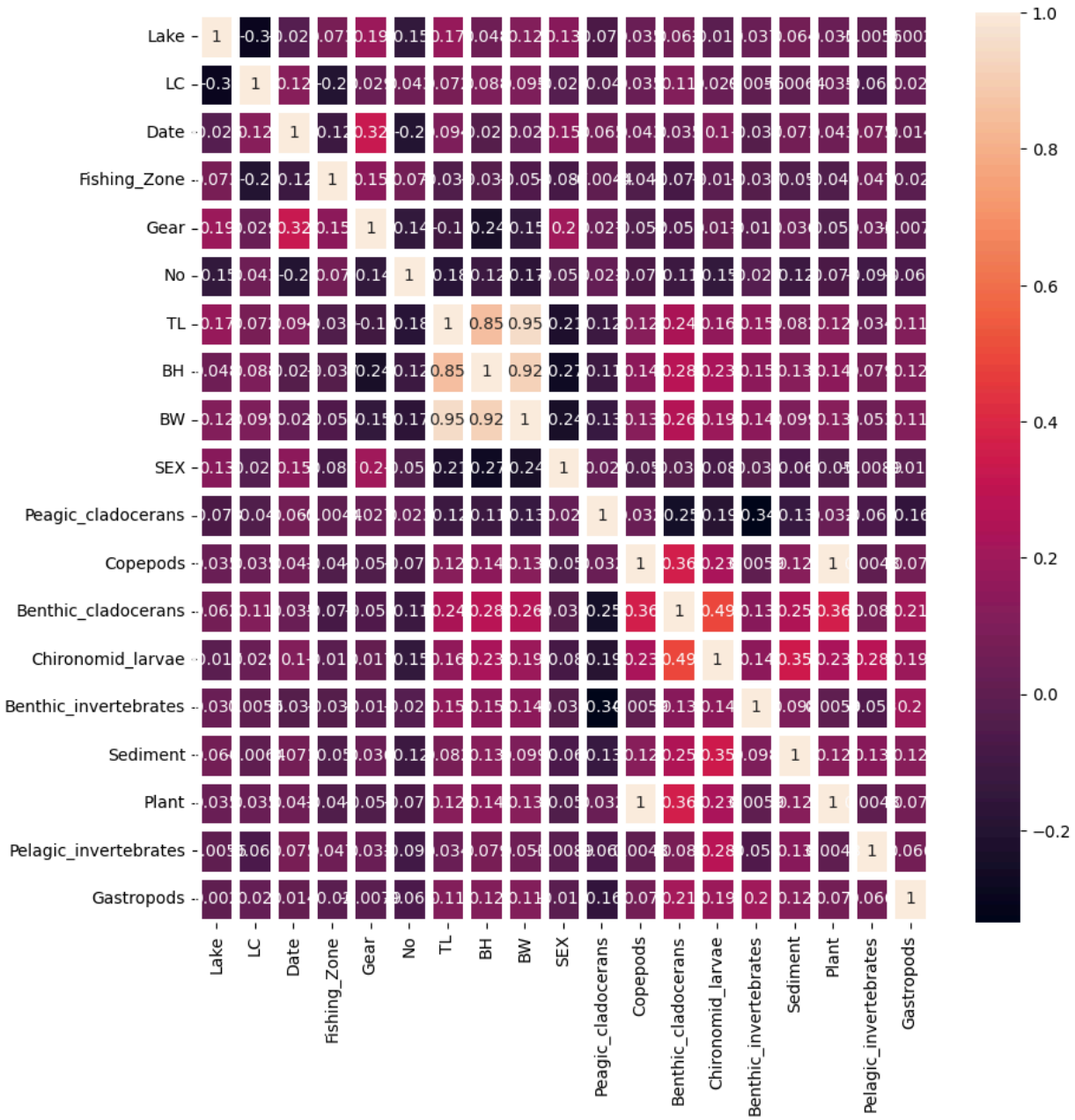
In [132…

```
#Definimos X,y
X = data.drop(["Species"],axis = 1) #Variables independientes
y = data["Species"] # Variables dependientes
```

In [133…

```
#Aplicando correlacion con metodo spearman
# La correlación se deja con el metodo spearman porque genera correlaciones altas
import seaborn as sns
```

```
def diagrama (df,tamuno,tamdos):
  f, ax = plt.subplots(figsize=(tamuno,tamdos))
  sns.heatmap(df.corr(method="spearman"),annot=True, linewidths=5, ax=ax)

diagrama(X,10,10)
```



Solo pocas variables tienen una buena correlación, al ser tan pocas se puede optar por hacer un árbol de desición para resolver el problema.

Caso 1 : Regresión logística multiclase y Árbol de decisión

- Sin normalizar.
- Sin hiperparámetros.

- Sin mejores características.

En este caso se trabajoron con 2 modelos de clasificación los cuales son: regresión logística multiclase y árbol de decisión,el objetivo de trabajar con estos dos modelos era poder evidenciar cual de los modelos era el mas optimo para responder a la problemática propuesta anteriormente.

En este caso en especifico se trabajo con la data sin normalizar, sin balancear y sin mejores características para ver cual era la reacción que tenían los modelos.

Regresion Logistica Multiclase

In [134…
```
# Importar librerias
from sklearn.model_selection import train_test_split #Librerias para entrenar modelo
from sklearn.linear_model import LogisticRegression # Importamos modelo de Regresión L
from sklearn.metrics import mean_squared_error,r2_score,accuracy_score,confusion_matri
```

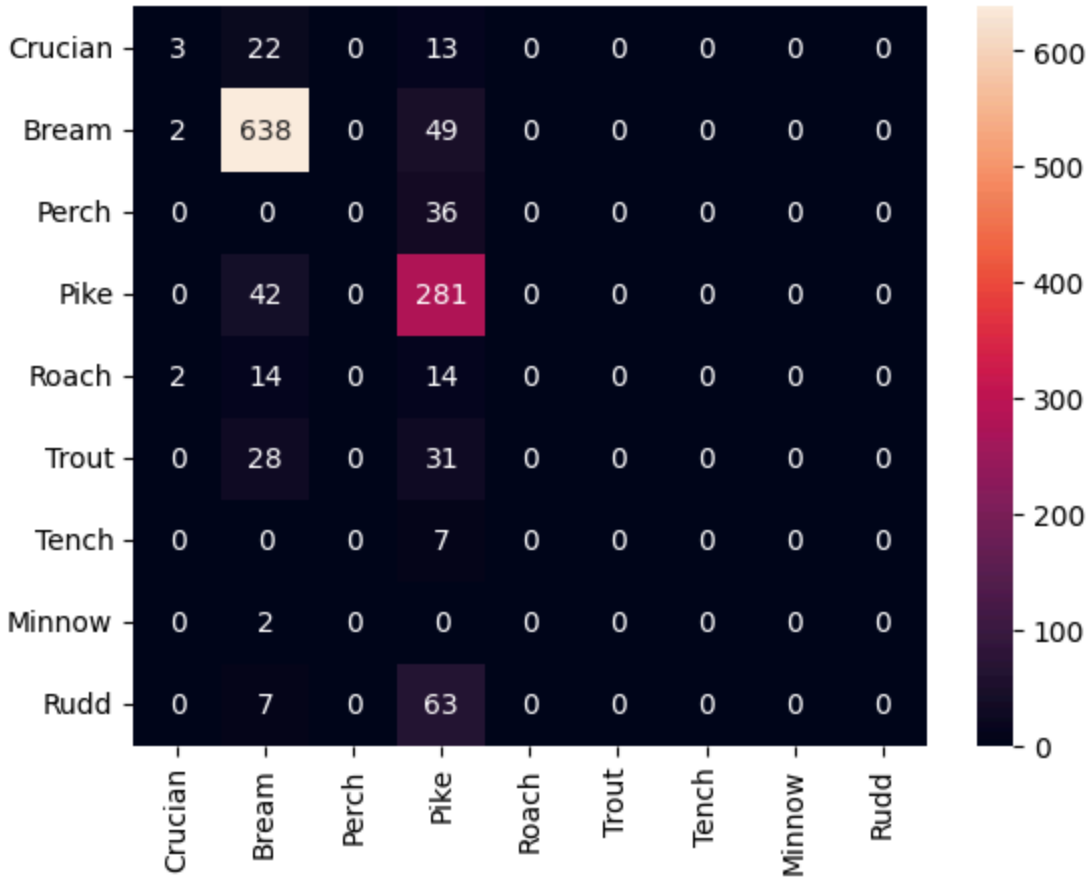
In [145…
```
# Entrenamos modelo y realizamos partición de datos para testear y entrenar
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.4, random_state = 8

# Creando modelo de Regresión Logística
model = LogisticRegression(multi_class ="multinomial", solver ="saga", max_iter = 100)
model.fit(X_train,y_train)

#Predicción de datos de testeo
y_pred = model.predict(X_test)

#Calculamos métricas
accuracy = accuracy_score(y_test,y_pred) # Acuraccy
conf_matrix = confusion_matrix(y_test,y_pred) # Confusion_matrix

print("Acurracy:", accuracy)
print("")
print(conf_matrix)
#0.56 - 80
```

```
Acurracy: 0.7352472089314195

[[  3  22   0  13   0   0   0   0   0]
 [  2 638   0  49   0   0   0   0   0]
 [  0   0   0  36   0   0   0   0   0]
 [  0  42   0 281   0   0   0   0   0]
 [  2  14   0  14   0   0   0   0   0]
 [  0  28   0  31   0   0   0   0   0]
 [  0   0   0   7   0   0   0   0   0]
 [  0   2   0   0   0   0   0   0   0]
 [  0   7   0  63   0   0   0   0   0]]
```

In [136…
```
#Graficamos matriz de confusión
cm = confusion_matrix(y_test,y_pred)
cm_matrix = pd.DataFrame(data=cm,
                         columns =['Crucian','Bream', 'Perch', 'Pike', 'Roach', 'Trout


                          index =['Crucian','Bream', 'Perch', 'Pike', 'Roach', 'Trout'
```

```
sns.heatmap(cm_matrix,annot = True, fmt = "d")
```

Out[136]: <Axes: >



Reporte de clasificación

In [137…

```
#Importamos métrica de reporte de clasificación
from sklearn.metrics import classification_report

print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

       Bream       0.43      0.08      0.13        38
     Crucian       0.85      0.93      0.88       689
      Minnow       0.00      0.00      0.00        36
       Perch       0.57      0.87      0.69       323
        Pike       0.00      0.00      0.00        30
       Roach       0.00      0.00      0.00        59
        Rudd       0.00      0.00      0.00         7
       Tench       0.00      0.00      0.00         2
       Trout       0.00      0.00      0.00        70

    accuracy                           0.74      1254
   macro avg       0.20      0.21      0.19      1254
weighted avg       0.63      0.74      0.67      1254
```

Árbol de desición

In [138…

```
#Importamos librerias
from sklearn.tree import DecisionTreeClassifier # Arbol de desición para clasificar

#Creando y entrenando el modelo de árblo de desición
model_dt1 = DecisionTreeClassifier(random_state = 42, criterion = "gini")
model_dt1.fit(X_train,y_train) # Entrenando

#Muestra la profundidad del árbol
print("Profundidad del arbol: ", model_dt1.get_depth())
#Muestra la eficiencia del modelo
print("Eficiencia del modelo: ",model_dt1.score(X_train,y_train))

# Prediccion de los datos
yhat1 = model_dt1.predict(X_test) #Train

# Métrica matriz de confusión
cm3 = confusion_matrix(y_test.values,yhat1)
print("-"*50)
print("Matriz de confusion")
print("-"*50)
print(cm3)
```

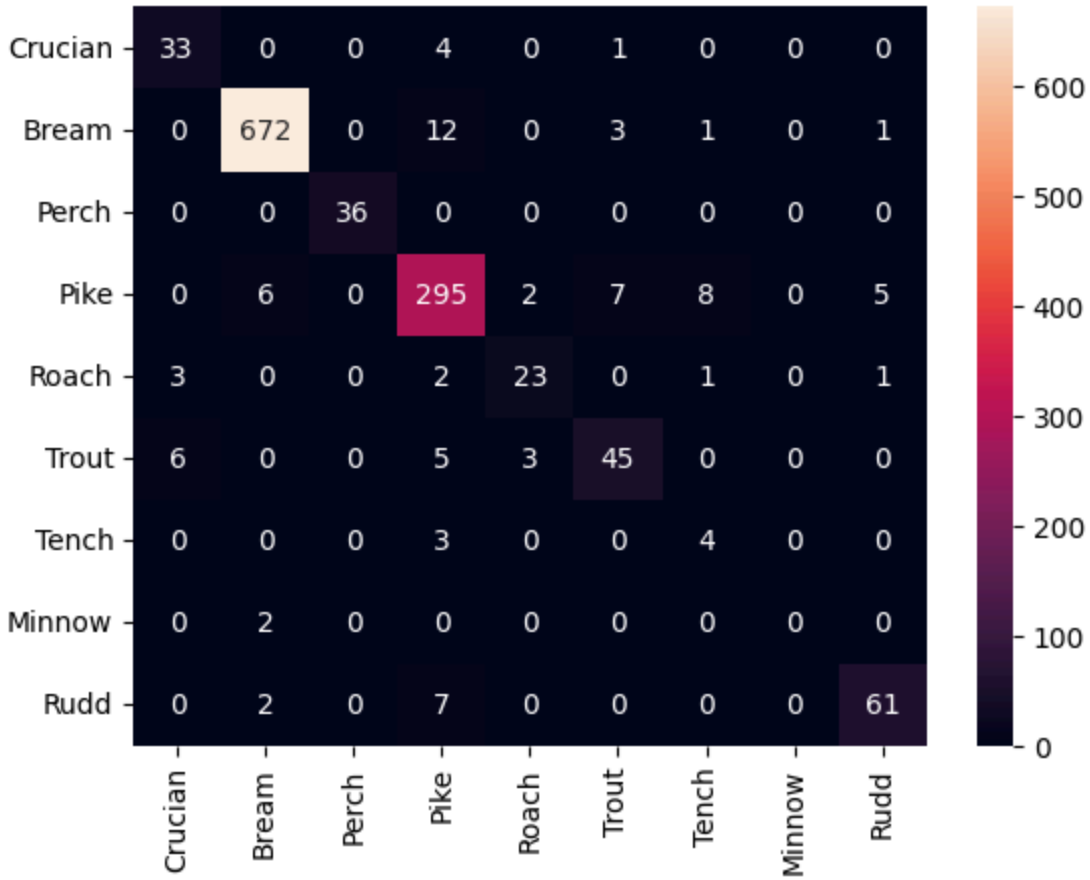
```
Profundidad del arbol:  15
Eficiencia del modelo:  1.0
--------------------------------------------------
Matriz de confusion
--------------------------------------------------
[[ 33   0   0   4   0   1   0   0   0]
 [  0 672   0  12   0   3   1   0   1]
 [  0   0  36   0   0   0   0   0   0]
 [  0   6   0 295   2   7   8   0   5]
 [  3   0   0   2  23   0   1   0   1]
 [  6   0   0   5   3  45   0   0   0]
 [  0   0   0   3   0   0   4   0   0]
 [  0   2   0   0   0   0   0   0   0]
 [  0   2   0   7   0   0   0   0  61]]
```

In [139…

```
#Graficamos matriz de confusión
cm_matrix2 = pd.DataFrame(data=cm3,
                          columns =['Crucian','Bream', 'Perch', 'Pike', 'Roach', 'Trout


                           index =['Crucian','Bream', 'Perch', 'Pike', 'Roach', 'Trout'

sns.heatmap(cm_matrix2,annot = True, fmt = "d")
```

Out[139]:

```
<Axes: >
```



Reporte de clasificación

In [147…

```
#Reporte de clasificación
print("-"*50)
print("Reporte de clasificación")
print("-"*50)
print(classification_report(y_test.values,yhat1))
```

```
--------------------------------------------------
Reporte de clasificación
--------------------------------------------------
              precision    recall  f1-score   support

       Bream       0.79      0.87      0.82        38
     Crucian       0.99      0.98      0.98       689
      Minnow       1.00      1.00      1.00        36
       Perch       0.90      0.91      0.91       323
        Pike       0.82      0.77      0.79        30
       Roach       0.80      0.76      0.78        59
        Rudd       0.29      0.57      0.38         7
       Tench       0.00      0.00      0.00         2
       Trout       0.90      0.87      0.88        70

    accuracy                           0.93      1254
   macro avg       0.72      0.75      0.73      1254
weighted avg       0.93      0.93      0.93      1254
```

Reporte caso 1:

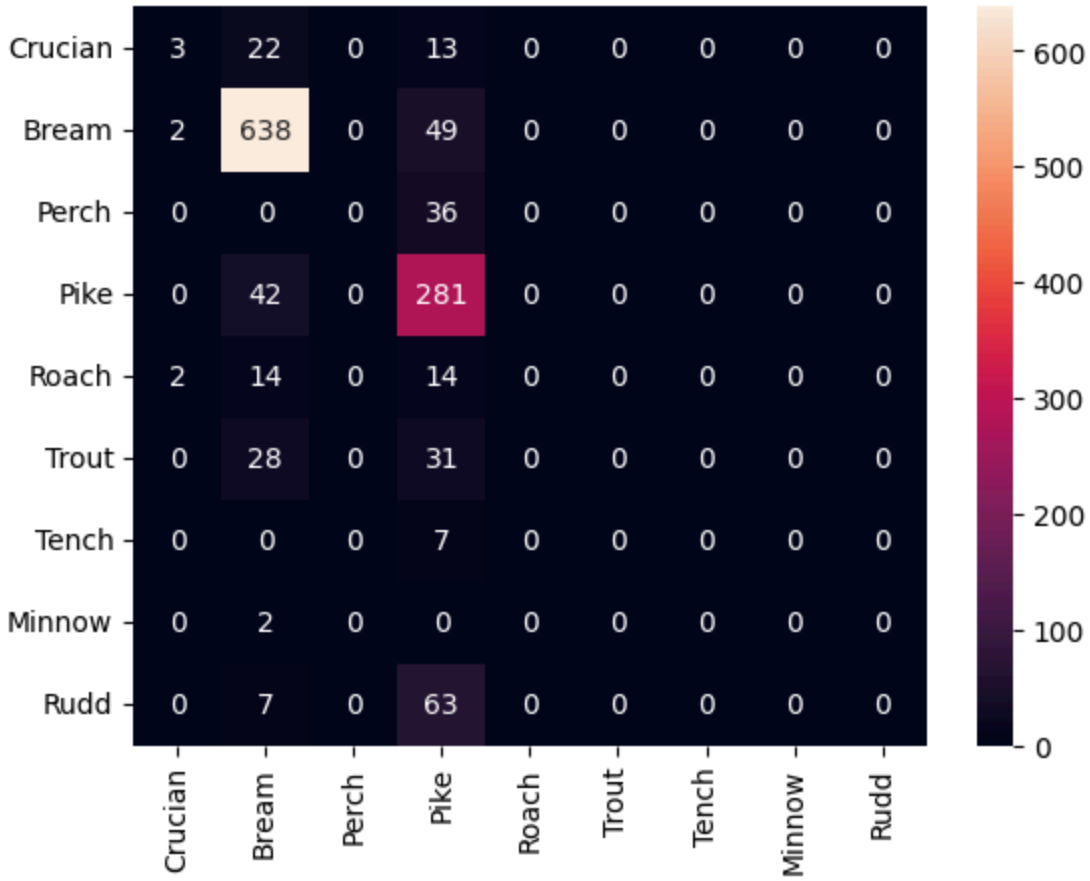
Acuraccy (Exactitud)

- La exactitud de la Regresión Logistica Multiclase es de 0.73 el cual se mantiene estable.
- La exactitud del Árbol de desición es de 0.93 el cual se mantiene estable.

Matriz de confusión

In [141…
```
#Matriz de confusión - Regresión Logistica Multiclase
sns.heatmap(cm_matrix,annot = True, fmt = "d")
```

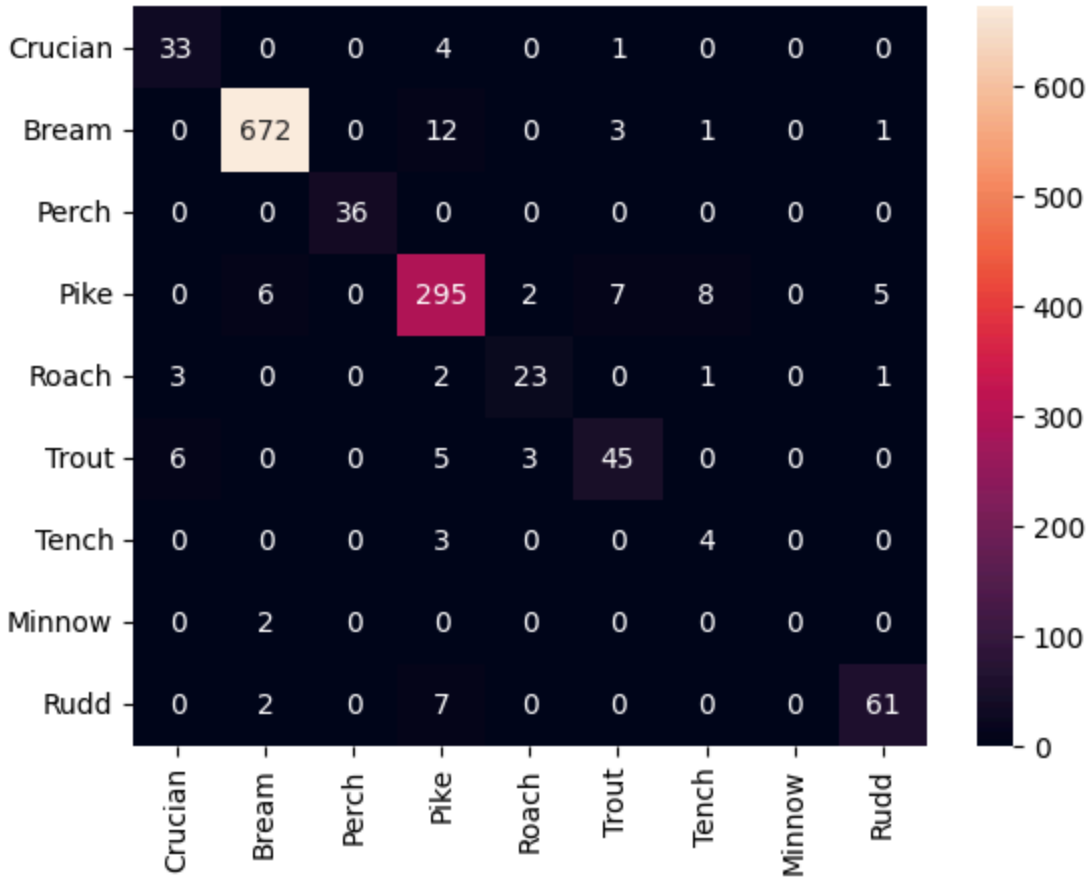
Out[141]:
```
<Axes: >
```



En la matriz de confusión se puede notar que en general nuestro modelo muestra un rendimiento muy poco eficaz ya que en algunas predicciones se ha equivocado como en las especies “Crucian”, “Pike”,” Bream” y también se puede notar que en las demas especies no logro clasificar ninguna correctamente obteniendo valores de 0.

In [142…
```
#Matriz de confusión - Arbol de desición
sns.heatmap(cm_matrix2,annot = True, fmt = "d")
```

Out[142]:
```
<Axes: >
```



En la matriz de confusión se pudo observar que el score del modelo era de 100%. Se nota que en general el modelo clasifica casi correctamente todas las especies a excepción de la especie "Minnow", eso indicaría que para este modelo al clasificar la especie" Minnow" no es la más optima.

Reporte de clasificación

In [143…

```
#Reporte de clasificación - Regresión Logistica Multiclase
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

      Bream       0.43      0.08      0.13        38
    Crucian       0.85      0.93      0.88       689
     Minnow       0.00      0.00      0.00        36
      Perch       0.57      0.87      0.69       323
       Pike       0.00      0.00      0.00        30
      Roach       0.00      0.00      0.00        59
       Rudd       0.00      0.00      0.00         7
      Tench       0.00      0.00      0.00         2
      Trout       0.00      0.00      0.00        70

   accuracy                           0.74      1254
  macro avg       0.20      0.21      0.19      1254
weighted avg       0.63      0.74      0.67      1254
```

- La especie "Bream" empieza con una presición de 0.43, pero al trancurrir el tiempo su presición va a bajar alcanzando un valor de 0.
- Las unicas especies que clasifica bien son la de "Crucian" que con el tiempo su presición aumenta, tambien "Perch" su presición mejorara a futuro.

In [144…
```
#Reporte de clasificación - Arbol de desición
print(classification_report(y_test.values,yhat1))
```

```
              precision    recall  f1-score   support

       Bream       0.79      0.87      0.82        38
     Crucian       0.99      0.98      0.98       689
      Minnow       1.00      1.00      1.00        36
       Perch       0.90      0.91      0.91       323
        Pike       0.82      0.77      0.79        30
       Roach       0.80      0.76      0.78        59
        Rudd       0.29      0.57      0.38         7
       Tench       0.00      0.00      0.00         2
       Trout       0.90      0.87      0.88        70

    accuracy                           0.93      1254
   macro avg       0.72      0.75      0.73      1254
weighted avg       0.93      0.93      0.93      1254
```

- Logra clasificar casi todas las categorias de especies, excepto en "Tench" que registra valores de 0.

Resumen Caso 1

En general en este caso 1, se evidencio que al trabajar con los 2 modelo de clasificacion el mejor era el de árbol de desición porque presentaba una presición y exactitud mejor a diferencia de la regresión logistica, pero ahi cierta duda con ese modelo de arbol de decisión porque presenta una eficicencia del modelo de un 100%, eso quiere decir que hay que sospechar porque son escasas las veces que un modelo sea tan perfecto.

Se noto que sin normalizar, sin balancear y sin utilizar los hiperparametros genera resultados poco convicentes para resolver a la problematica propuesta. Eso nos indica que debemos mejorar el modelo utilizando otras técnicas para evidenciar si mejora el modelo.

Caso 2: Regresión logística multiclase y Árbol de decisión

- Normalización de la data.

- Hiperparámetros
- Mejores características.

En este caso se trabajaron con los mismos modelos de clasificación utilizados anteriormente, cumpliendo a resolver la misma problemática

En este caso en específico se trabajó con la data normalizada, con hiperparámetros y con mejores características.

Normalizando la data

Para normalizar la data se utilizó el método de MinMaxScaler normalizando solo la variable X porque es donde están los datos numéricos del dataset, no se uso y porque es donde esta almacenada nuestra variable categórica.

In [148…
```
#Funcion normalizacion (MinMaxScaler)

#Importamos la libreria
from sklearn.preprocessing import MinMaxScaler

#Sacamos los valores sin escabezado
def datanormalizados(df):
  valores = df.values
  scaler = MinMaxScaler(feature_range = (0,1)) #Porque la normalizacion va de 0-1
  #Aplica la normaliacion
  scaler = scaler.fit(valores)


#np.vstack = Empieza a unir los valores entre un min y un max para que no se salga del
  pd.DataFrame(np.vstack((scaler.data_min_,scaler.data_max_)),
               index =["Min","Max"],
               columns = df.columns)
#Datos ya normalizados
  normalizados = scaler.transform(valores)
  df_norm = pd.DataFrame(normalizados,index=df.index,columns = df.columns)

  return df_norm
```

In [149…
```
# Aginamos los datos normalizados a la variable independiente X
datanormalizada = datanormalizados(X)
```

In [150…
```
#Mostrando valores estadisticos de la data normalizada
datanormalizada.describe().T
```

Out[150]:

	count	mean	std	min	25%	50%	75%	max
Lake	3134.0	0.505830	0.339936	0.0	0.181818	0.545455	0.818182	1.0
LC	3134.0	0.540117	0.305985	0.0	0.272727	0.545455	0.818182	1.0
Date	3134.0	0.472619	0.302739	0.0	0.218750	0.500000	0.718750	1.0
Fishing_Zone	3134.0	0.108966	0.228120	0.0	0.000000	0.000000	0.000000	1.0
Gear	3134.0	0.539119	0.166969	0.0	0.400000	0.600000	0.600000	1.0
No	3134.0	0.189936	0.201951	0.0	0.052023	0.121387	0.242775	1.0
TL	3134.0	0.228730	0.134190	0.0	0.130730	0.202037	0.283531	1.0
BH	3134.0	0.234767	0.182284	0.0	0.121951	0.182927	0.285213	1.0
BW	3134.0	0.061559	0.114284	0.0	0.006872	0.020310	0.050749	1.0
SEX	3134.0	0.388162	0.278528	0.0	0.000000	0.500000	0.500000	1.0
Peagic_cladocerans	3134.0	0.173593	0.147552	0.0	0.150000	0.150000	0.150000	1.0
Copepods	3134.0	0.004461	0.046439	0.0	0.000000	0.000000	0.000000	1.0
Benthic_cladocerans	3134.0	0.023902	0.125419	0.0	0.000000	0.000000	0.000000	1.0
Chironomid_larvae	3134.0	0.013759	0.077383	0.0	0.000000	0.000000	0.000000	1.0
Benthic_invertebrates	3134.0	0.069593	0.111974	0.0	0.050000	0.050000	0.050000	1.0
Sediment	3134.0	0.009713	0.070308	0.0	0.000000	0.000000	0.000000	1.0
Plant	3134.0	0.004461	0.046439	0.0	0.000000	0.000000	0.000000	1.0
Pelagic_invertebrates	3134.0	0.002819	0.034044	0.0	0.000000	0.000000	0.000000	1.0
Gastropods	3134.0	0.002672	0.040419	0.0	0.000000	0.000000	0.000000	1.0

Regresion logistica multiclase

Hiperparámetros

In [151…

```
#"solver":["saga","newton-cg","sag","lbfgs"]
#"penalty": ["l1","l2","elasticnet",None]

# Implementando hiperparámetros
param_grid = {
    "C": [0.01,0.1,1,10,100],
    "solver":["saga"], # Se hizo con "saga" porque se ajusta mejor al modelo
    "penalty": ["l1","l2"],
    "max_iter": [100,200,300]
}

#Importamos libreria que saca las mejores parámetros
from sklearn.model_selection import GridSearchCV

# Creando modelo y entrenando
grid_search = GridSearchCV(model, param_grid, cv = 10, scoring = "accuracy",verbose =
```

```
grid_search.fit(X_train,y_train)


print("Mejores parametros: ", grid_search.best_params_) #Mejores parámetros
print("Mejor score de croos-validation :",grid_search.best_score_) #Mejor eficiencia
#0.16
```

```
Fitting 10 folds for each of 30 candidates, totalling 300 fits
Mejores parametros:  {'C': 0.01, 'max_iter': 300, 'penalty': 'l2', 'solver': 'saga'}
Mejor score de croos-validation : 0.7468085106382979
```

In [154…

```
#Implementando hiperparámetros a nuevo modelo de Regresión Logistica
model2 = LogisticRegression(multi_class ="multinomial", solver ="saga", max_iter = 300
#Entrenando modelo
model2.fit(X_train,y_train)
#Prediciendo modelo
y_pred = model2.predict(X_test)

# Calculando métricas
accuracy = accuracy_score(y_test,y_pred)
conf_matrix = confusion_matrix(y_test,y_pred)

print("Acurracy:", accuracy)
print("")
print(conf_matrix)
#0.23
```

```
Acurracy: 0.74481658692185

[[  3  20   0  15   0   0   0   0   0]
 [  2 644   0  43   0   0   0   0   0]
 [  0   2   0  34   0   0   0   0   0]
 [  0  38   0 284   0   0   0   0   1]
 [  2  11   0  15   2   0   0   0   0]
 [  0  28   0  31   0   0   0   0   0]
 [  0   0   0   7   0   0   0   0   0]
 [  0   2   0   0   0   0   0   0   0]
 [  0   6   0  63   0   0   0   0   1]]
```

Árbol de desición

Hiperparámetros

In [155…

```
# Implementando hiperparámetros
parameter_grid2 = {
    "max_depth": [24,25,26,27,28,29,30],
    "max_features":[0.3,0.2,0.7],
    "min_samples_leaf": [5,10,20,50,100],
    "criterion": ["gini","entropy"],
    "random_state": [None,0,1,10,42,100]
}

#Importamos libreria que saca las mejores parámetros
from sklearn.model_selection import GridSearchCV

# Creando modelo y entrenando
gridsearch = GridSearchCV(estimator = model_dt1, param_grid = parameter_grid2, scoring
```

```
gridsearch.fit(X_train,y_train)

#Muestra mejores parámetros
print(gridsearch.best_params_)
```

```
{'criterion': 'gini', 'max_depth': 24, 'max_features': 0.3, 'min_samples_leaf': 5, 'r
andom_state': None}
```

In [166…

```
#Implementando hiperparámetros a nuevo modelo de Árbol de desición
model_dt1 = DecisionTreeClassifier(criterion= 'gini', max_depth= 24, max_features= 0.3

#Entrenando modelo
model_dt1.fit(X_train,y_train)

#Muestra la profundidad del árbol y eficiencia del modelo
print("Profundidad del arbol: ", model_dt1.get_depth())
print("Eficiencia del modelo: ",model_dt1.score(X_train,y_train))

# Predice el modelo
yhat1 = model_dt1.predict(X_test)

# Métrica matriz de confusión
cm3 = confusion_matrix(y_test.values,yhat1)
print("-"*50)
print("Matriz de confusion")
print("-"*50)
print(cm3)

# Métrica reporte de clasificación
print("-"*50)
print("Reporte de clasificación")
print("-"*50)
print(classification_report(y_test.values,yhat1))
#0.92 - 0.95
```

```
Profundidad del arbol:  14
Eficiencia del modelo:  0.9457446808510638
--------------------------------------------------
Matriz de confusion
--------------------------------------------------
[[ 26   0   0   0   3   9   0   0   0]
 [ 11 658   1  16   0   0   0   0   3]
 [  1   2  33   0   0   0   0   0   0]
 [ 12   8   0 292   2   3   0   0   6]
 [  7   0   0   6  16   1   0   0   0]
 [ 16   1   0   6   0  36   0   0   0]
 [  0   2   0   4   0   1   0   0   0]
 [  0   2   0   0   0   0   0   0   0]
 [  0   0   0   8   0   0   0   0  62]]
--------------------------------------------------
Reporte de clasificación
--------------------------------------------------
              precision    recall  f1-score   support

       Bream       0.36      0.68      0.47        38
     Crucian       0.98      0.96      0.97       689
      Minnow       0.97      0.92      0.94        36
       Perch       0.88      0.90      0.89       323
        Pike       0.76      0.53      0.63        30
       Roach       0.72      0.61      0.66        59
        Rudd       0.00      0.00      0.00         7
       Tench       0.00      0.00      0.00         2
       Trout       0.87      0.89      0.88        70

    accuracy                           0.90      1254
   macro avg       0.62      0.61      0.60      1254
weighted avg       0.90      0.90      0.90      1254
```

Trabajando con normalizacion

Regresión Logística con hiperparámetros

In [169…

```
# Entrenamos modelo y realizamos partición de datos normalizados para testear y entren

X_trainN,X_testN,y_trainN,y_testN = train_test_split(datanormalizada,y,test_size = 0.4

#Crando modelo de Regresión Logística
model2Nor = LogisticRegression(multi_class ="multinomial", solver ="saga", max_iter =
model2Nor.fit(X_trainN,y_trainN) # Entrenando modelo

#Predicción de los datos
y_predN = model2Nor.predict(X_testN)

# Calculando métricas
accuracy = accuracy_score(y_testN,y_predN)
conf_matrix3 = confusion_matrix(y_testN,y_predN)

print("Acurracy:", accuracy)
print("")
print(conf_matrix3)
```

```
Acurracy: 0.5909090909090909

[[  0  27   0  11   0   0   0   0   0]
 [  0 675   0  14   0   0   0   0   0]
 [  0  15   0  21   0   0   0   0   0]
 [  0 257   0  66   0   0   0   0   0]
 [  0  28   0   2   0   0   0   0   0]
 [  0  42   0  17   0   0   0   0   0]
 [  0   3   0   4   0   0   0   0   0]
 [  0   2   0   0   0   0   0   0   0]
 [  0  59   0  11   0   0   0   0   0]]
```

Reporte

In [170…

```
#Importamos métrica de reporte de clasificación
from sklearn.metrics import classification_report

print(classification_report(y_testN,y_predN))
```

```
              precision    recall  f1-score   support

       Bream       0.00      0.00      0.00        38
     Crucian       0.61      0.98      0.75       689
      Minnow       0.00      0.00      0.00        36
       Perch       0.45      0.20      0.28       323
        Pike       0.00      0.00      0.00        30
       Roach       0.00      0.00      0.00        59
        Rudd       0.00      0.00      0.00         7
       Tench       0.00      0.00      0.00         2
       Trout       0.00      0.00      0.00        70

    accuracy                           0.59      1254
   macro avg       0.12      0.13      0.11      1254
weighted avg       0.45      0.59      0.49      1254
```

Árbol de desición con hiperparámetros

In [176…

```
#Creando modelo Árbol de desición con hiperparámetros
model_dt2Nor = DecisionTreeClassifier(criterion= 'gini', max_depth= 24, max_features=
#Entrenando modelo
model_dt2Nor.fit(X_trainN,y_trainN)

# Profundidad del arbol
print("Profundidad del arbol: ", model_dt2Nor.get_depth())
# Eficiencia del modelo
print("Eficiencia del modelo: ",model_dt2Nor.score(X_trainN,y_trainN))

# Predicción de los datos de entrenamiento
yhat2Nor = model_dt2Nor.predict(X_testN)

# Matriz de confusión
cm4 = confusion_matrix(y_testN.values,yhat2Nor)
print("-"*50)
print("Matriz de confusion")
print("-"*50)
print(cm4)
```

```
#Reporte de clasificación
print("-"*50)
print("Reporte de clasificación")
print("-"*50)
print(classification_report(y_testN.values,yhat2Nor))
#0.92 - 0.95
```

```
Profundidad del arbol:  12
Eficiencia del modelo:  0.924468085106383
--------------------------------------------------
Matriz de confusion
--------------------------------------------------
[[ 12   4   0   1   0  21   0   0   0]
 [  0 651   2  23   0   3   1   0   9]
 [  0   2  33   0   0   0   0   0   1]
 [  0   6   2 292   0  13   6   0   4]
 [  1   2   0   6   9  11   0   0   1]
 [  1   1   0   2   0  55   0   0   0]
 [  0   0   0   2   0   1   4   0   0]
 [  0   1   0   1   0   0   0   0   0]
 [  0   2   1  16   2   0   0   0  49]]
--------------------------------------------------
Reporte de clasificación
--------------------------------------------------
              precision    recall  f1-score   support

       Bream       0.86      0.32      0.46        38
     Crucian       0.97      0.94      0.96       689
      Minnow       0.87      0.92      0.89        36
       Perch       0.85      0.90      0.88       323
        Pike       0.82      0.30      0.44        30
       Roach       0.53      0.93      0.67        59
        Rudd       0.36      0.57      0.44         7
       Tench       0.00      0.00      0.00         2
       Trout       0.77      0.70      0.73        70

    accuracy                           0.88      1254
   macro avg       0.67      0.62      0.61      1254
weighted avg       0.89      0.88      0.88      1254
```

Mejores caracteristicas

In [177…

```
# Encuentra las mejores caracteristicas del modelo normalizado
feature_importanceA = pd.Series(model_dt2Nor.feature_importances_, index = X.columns).
print(feature_importanceA)
```

```
BH                      0.189483
Lake                    0.175513
Date                    0.115621
LC                      0.112675
TL                      0.107151
Gear                    0.075867
SEX                     0.071048
Benthic_invertebrates   0.057718
BW                      0.056156
No                      0.026960
Peagic_cladocerans      0.011628
Chironomid_larvae       0.000139
Fishing_Zone            0.000040
Copepods                0.000000
Benthic_cladocerans     0.000000
Sediment                0.000000
Plant                   0.000000
Pelagic_invertebrates   0.000000
Gastropods              0.000000
dtype: float64
```

In [178…
```
#Seleccionamos las carateristicas mas importantes mayores a 0.1
top_features = feature_importanceA[feature_importanceA >0.1].index
```

In [179…
```
#Muestra las mejores caracteristicas
top_features
```

Out[179]:
```
Index(['BH', 'Lake', 'Date', 'LC', 'TL'], dtype='object')
```

In [180…
```
#Reducimos conjunto de datos a las caracteristicas mas importantes
X_train_reduced = X_trainN[top_features]
X_test_reduced = X_testN[top_features]

#Entrenando modelo
tree_classifier_reduced = DecisionTreeClassifier(criterion= 'gini', max_depth= 24, max

tree_classifier_reduced.fit(X_train_reduced,y_trainN)
```

Out[180]:
```
DecisionTreeClassifier(max_depth=24, max_features=0.3, min_samples_leaf=5)
```

In [181…
```
#Predecimos el modelo
y_pred_reduced = tree_classifier_reduced.predict(X_test_reduced)
```

In [186…
```
# Profundidad del arbol
print("Profundidad del arbol: ", tree_classifier_reduced.get_depth())
# Eficiencia del modelo
print("Eficiencia del modelo: ",tree_classifier_reduced.score(X_train_reduced,y_trainN

# Matriz de confusión
cm5 = confusion_matrix(y_testN,y_pred_reduced)
print("-"*50)
print("Matriz de confusion")
print("-"*50)
print(cm5)

# Reporte de clasificación
```

```
print("-"*50)
print("Reporte de clasificación")
print("-"*50)
print(classification_report(y_testN,y_pred_reduced))
#0.92
```

```
Profundidad del arbol:  13
Eficiencia del modelo:  0.9297872340425531
--------------------------------------------------
Matriz de confusion
--------------------------------------------------
[[ 24   0   0   0   4  10   0   0   0]
 [  0 668   1  14   0   5   0   0   1]
 [  0   2  30   4   0   0   0   0   0]
 [  3   6   3 288   5   9   4   0   5]
 [  2   2   0  11  11   4   0   0   0]
 [  5   5   0   3   0  45   1   0   0]
 [  0   4   0   0   0   1   2   0   0]
 [  0   1   0   1   0   0   0   0   0]
 [  0   4   0  13   0   0   0   0  53]]
--------------------------------------------------
Reporte de clasificación
--------------------------------------------------
              precision    recall  f1-score   support

       Bream       0.71      0.63      0.67        38
     Crucian       0.97      0.97      0.97       689
      Minnow       0.88      0.83      0.86        36
       Perch       0.86      0.89      0.88       323
        Pike       0.55      0.37      0.44        30
       Roach       0.61      0.76      0.68        59
        Rudd       0.29      0.29      0.29         7
       Tench       0.00      0.00      0.00         2
       Trout       0.90      0.76      0.82        70

    accuracy                           0.89      1254
   macro avg       0.64      0.61      0.62      1254
weighted avg       0.89      0.89      0.89      1254
```

Reporte Caso 2

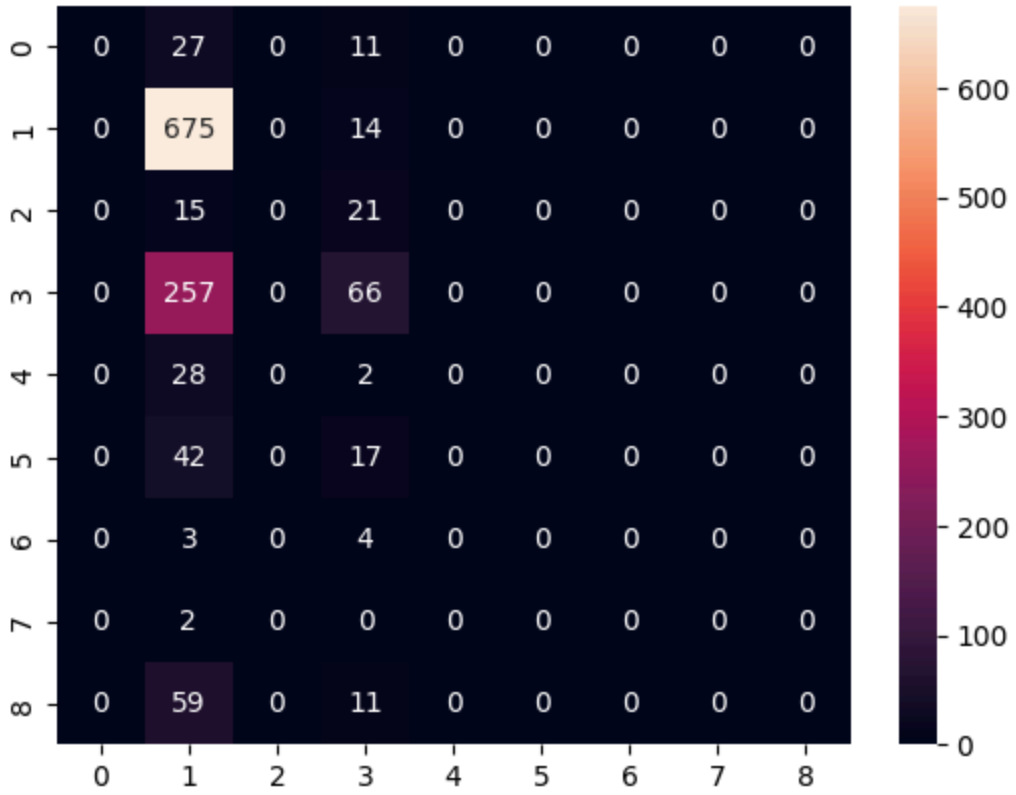
Acuraccy (Exactitud)

- La exactitud de la Regresión Logistica Multiclase es de 0.59 el cual se mantiene estable.
- La exactitud del Árbol de desición es de 0.89 el cual se mantenia estable

Matriz de confusión

In [187…
```
#Matriz de confusión - Regresión Logistica Multiclase
sns.heatmap(conf_matrix3,annot = True, fmt = "d")
```

Out[187]:
```
<Axes: >
```



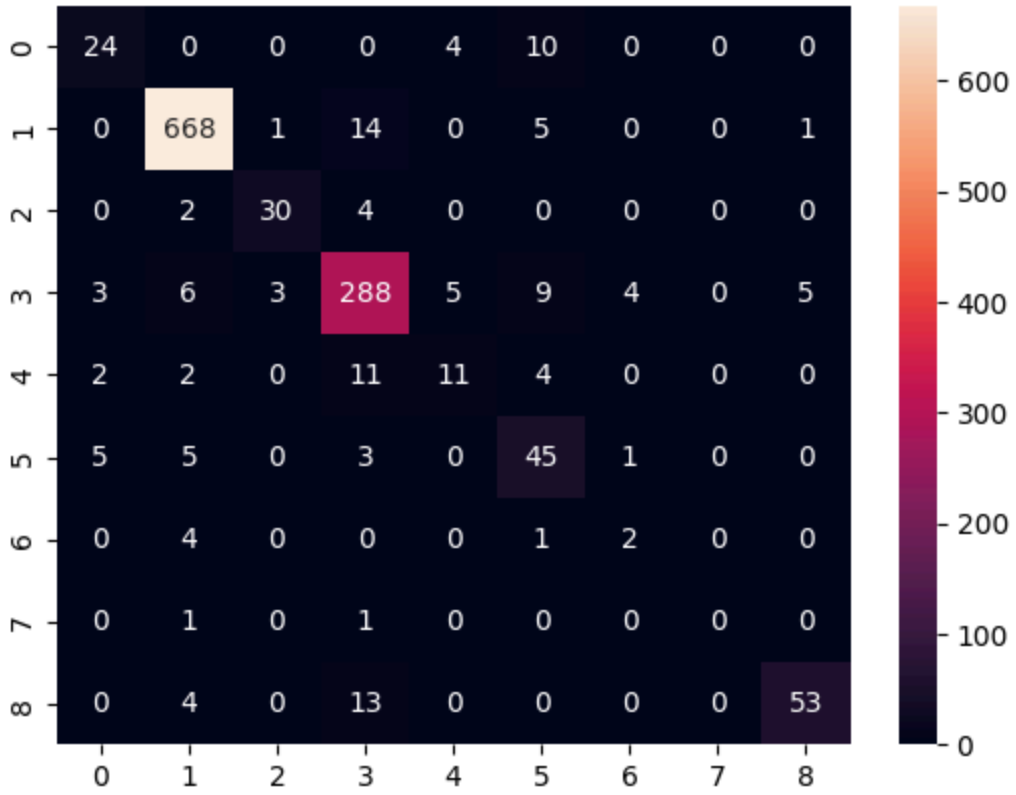
En la matriz de confusión se puede notar que en general nuestro modelo muestra tambien un rendimiento muy poco eficazya que la mayoria de las clasificaciones tiene valores de 0, lo que quiere decir que no reconoce muy bien todavia a que especies clasificarla. Solo reconoce a la especie "Crucian"

In [188…

```
#Matriz de confusión - Arbol de desición
sns.heatmap(cm5,annot = True, fmt = "d")
```

Out[188]:

```
<Axes: >
```



Al igual que la matriz de confusión de la regresión logistica anterior tampoco logra clasificar correctamente la mayoria de las clases, obteniendo resultados practicamente erroneos.

Reporte de clasificación

In [189…

```
#Reporte de clasificación - Regresión Logistica Multiclase
print(classification_report(y_testN,y_predN))
```

```
              precision    recall  f1-score   support

       Bream       0.00      0.00      0.00        38
     Crucian       0.61      0.98      0.75       689
      Minnow       0.00      0.00      0.00        36
       Perch       0.45      0.20      0.28       323
        Pike       0.00      0.00      0.00        30
       Roach       0.00      0.00      0.00        59
        Rudd       0.00      0.00      0.00         7
       Tench       0.00      0.00      0.00         2
       Trout       0.00      0.00      0.00        70

    accuracy                           0.59      1254
   macro avg       0.12      0.13      0.11      1254
weighted avg       0.45      0.59      0.49      1254
```

- La especie "Bream", "Minnow", "Pike" , "Roach" , "Rudd" , "Tench" , "Trout" no logra reconocerlas y por lo tanto no logra clasificarlas, como se muestra en su matriz de confusión

- Las unicas especies que clasifica bien son la de "Crucian" que con el tiempo su presición aumenta, tambien "Perch", pero su presición empreorara a futuro.

In [190…
```
#Reporte de clasificación - Arbol de desición
print(classification_report(y_testN,y_pred_reduced))
```

```
              precision    recall  f1-score   support

       Bream       0.71      0.63      0.67        38
     Crucian       0.97      0.97      0.97       689
      Minnow       0.88      0.83      0.86        36
       Perch       0.86      0.89      0.88       323
        Pike       0.55      0.37      0.44        30
       Roach       0.61      0.76      0.68        59
        Rudd       0.29      0.29      0.29         7
       Tench       0.00      0.00      0.00         2
       Trout       0.90      0.76      0.82        70

    accuracy                           0.89      1254
   macro avg       0.64      0.61      0.62      1254
weighted avg       0.89      0.89      0.89      1254
```

- Logra clasificar casi todas las categorias de especies, excepto en "Tench" que registra valores de 0.

Resumen Caso 2

- En general en este caso 2, se evidencio al igual que en el caso 1 es mejor trabajar con el modelo de árbol de desición porque presentaba una presición y exactitud mejor a diferencia de la regresión logistica, sin embargo sigue teniendo los mismos problemas que en el caso anterior, no lograba clasificar correctamente las especies teniendo en cuenta que se utilizarón las mejores caracteristicas y los hiperparámetros que se ajustaban al modelo.
- Se notó que al normalizar la data, utilizando los hiperparametros y las mejores caracteristicas genera resultados poco convicentes para resolver a la problematica propuesta.
- Eso nos indica que debemos mejorar el modelo utilizando otras técnicas para evidenciar si el modelo logra tener una mejoria con respecto a los resultados obtenidos.

Caso 3: Regresión logística multiclase y Árbol de decisión

- Normalización de la data.

- Hiperparámetros.
- Balanceo de datos.
- Mejores características.

En este caso se trabajar con la normalización de la data, hiperparametros,mejores características y balanceo de la data.

Balanceando la data

Para el balanceo de la data se utilizó ".resample" de la librería sklearn, donde se probaron dos métodos de balanceo (OverSampling – UnderSampling). Para este modelo se ajustó mejor el método OverSampling que balancea los datos con el numero máximo de veces que aparece la variable en el dataset.

In [191…

```
# Importando libreria  que se utiliza para realizar el remuestreo del conjunto de datc
from sklearn.utils import resample

# Hace un conteo de las especies y busca el número maximo de veces que aparece las var
max_sample = data["Species"].value_counts().max()

# Lista con el nombre de categorias que almacena la variable "Species"
categories = ['Crucian','Bream', 'Perch', 'Pike', 'Roach', 'Trout', 'Tench','Minnow',
#Lista vacia
frames = []

# Ciclo que itera sobre cada categoria para balancear los datos
for category in categories:

  category_subset = data[data ["Species"]== category]

  resampled_subset = resample(category_subset,
                              replace = True,
                              n_samples = max_sample,
                              random_state = 123) # Semilla
  frames.append(resampled_subset) # Guarda los datos balanceados en una lista

data_resampled = pd.concat(frames) # Concatena los datos balanceados
```

In [192…

```
# Muestra conteo de la variable cateegorica balanceada
data_resampled["Species"].value_counts()
```

Out[192]:

```
Species
Crucian    1691
Bream      1691
Perch      1691
Pike       1691
Roach      1691
Trout      1691
Tench      1691
Minnow     1691
Rudd       1691
Name: count, dtype: int64
```

In [193…
```
# #Seleccionamos nuestros datos en X,y con data balanceada.
XB = data_resampled.drop(["Species"],axis = 1)
yb = data_resampled["Species"]
```

Normalizando data balanceada

In [194…
```
# Normalizamos la data balanceada
datanormalizadaBalanceada = datanormalizados(XB)
datanormalizadaBalanceada.describe().T
```

Out[194]:

	count	mean	std	min	25%	50%	75%	max
Lake	15219.0	0.552330	0.327806	0.0	0.272727	0.545455	0.818182	1.0
LC	15219.0	0.649457	0.324363	0.0	0.363636	0.727273	0.909091	1.0
Date	15219.0	0.540431	0.280947	0.0	0.375000	0.531250	0.750000	1.0
Fishing_Zone	15219.0	0.041133	0.155241	0.0	0.000000	0.000000	0.000000	1.0
Gear	15219.0	0.563822	0.182598	0.0	0.400000	0.600000	0.600000	1.0
No	15219.0	0.101074	0.127417	0.0	0.017341	0.063584	0.121387	1.0
TL	15219.0	0.256387	0.168746	0.0	0.135823	0.203735	0.354839	1.0
BH	15219.0	0.197132	0.141530	0.0	0.124390	0.182927	0.213415	1.0
BW	15219.0	0.065321	0.122992	0.0	0.007796	0.020310	0.069648	1.0
SEX	15219.0	0.428675	0.256577	0.0	0.500000	0.500000	0.500000	1.0
Peagic_cladocerans	15219.0	0.155801	0.073764	0.0	0.150000	0.150000	0.150000	1.0
Copepods	15219.0	0.000737	0.018539	0.0	0.000000	0.000000	0.000000	1.0
Benthic_cladocerans	15219.0	0.004501	0.055314	0.0	0.000000	0.000000	0.000000	1.0
Chironomid_larvae	15219.0	0.002891	0.037812	0.0	0.000000	0.000000	0.000000	1.0
Benthic_invertebrates	15219.0	0.053727	0.050443	0.0	0.050000	0.050000	0.050000	1.0
Sediment	15219.0	0.002017	0.033632	0.0	0.000000	0.000000	0.000000	1.0
Plant	15219.0	0.000737	0.018539	0.0	0.000000	0.000000	0.000000	1.0
Pelagic_invertebrates	15219.0	0.000760	0.017252	0.0	0.000000	0.000000	0.000000	1.0
Gastropods	15219.0	0.000537	0.020168	0.0	0.000000	0.000000	0.000000	1.0

Regresión Logística con datos balanceados, normalizados y con hiperparámetros

---

In [195…
```
#Entrenamos modelo y realizamos partición de datos para testear y entrenar
X_trainNB,X_testNB,y_trainNB,y_testNB = train_test_split(datanormalizadaBalanceada,yb,

#Creando modelo de Regresión Logísstica
model5 = LogisticRegression(multi_class ="multinomial", solver ="saga", max_iter = 300
```

```
model5.fit(X_trainNB,y_trainNB) # Entrenando modelo

#Prediciendo modelo
y_predNB = model5.predict(X_testNB)
# Calculando métricas
accuracy = accuracy_score(y_testNB,y_predNB)
conf_matrix5 = confusion_matrix(y_testNB,y_predNB)

print("Acurracy:", accuracy)
print("")
print(conf_matrix5)
```

```
Acurracy: 0.5565045992115637

[[177   0   0   0  54 247 183   0   0]
 [ 64 344  26  26  19   2  14  90 102]
 [  0   0 518  27   0   0   0   0 142]
 [  7   0  82 100   3   6  37 259 164]
 [173   0   0   0 213  62  30 182   0]
 [194   0   0   0   0 241 271   0   0]
 [  0   0   0   0   0 121 548   0   0]
 [  0   0   0   0   0   0   0 674   0]
 [  0   9  56  40   0   0   0   8 573]]
```

In [196…

```
#Importamos métrica de reporte de clasificación
from sklearn.metrics import classification_report

print(classification_report(y_testNB,y_predNB))
```

```
              precision    recall  f1-score   support

       Bream       0.29      0.27      0.28       661
     Crucian       0.97      0.50      0.66       687
      Minnow       0.76      0.75      0.76       687
       Perch       0.52      0.15      0.24       658
        Pike       0.74      0.32      0.45       660
       Roach       0.35      0.34      0.35       706
        Rudd       0.51      0.82      0.63       669
       Tench       0.56      1.00      0.71       674
       Trout       0.58      0.84      0.69       686

    accuracy                           0.56      6088
   macro avg       0.59      0.55      0.53      6088
weighted avg       0.59      0.56      0.53      6088
```

Árbol de decisión con datos balanceados, normalizados y con hiperparámetros

In [203…

```
#Crando modelo de Árvol de decisión
model_dt5 = DecisionTreeClassifier(criterion= 'gini', max_depth= 24, max_features= 0.3
#Entrenando modelo
model_dt5.fit(X_trainNB,y_trainNB)

# Profundidad del árbol
print("Profundidad del arbol: ", model_dt5.get_depth())
# Eficiencia del modelo
print("Eficiencia del modelo: ",model_dt5.score(X_trainNB,y_trainNB))
```

```
# Prediciendo modelo
yhat5 = model_dt5.predict(X_testNB)

#Matriz de confusión
cm5 = confusion_matrix(y_testNB.values,yhat5)
print("-"*50)
print("Matriz de confusion")
print("-"*50)
print(cm5)

#Reporte de clasificación
print("-"*50)
print("Reporte de clasificación")
print("-"*50)
print(classification_report(y_testNB.values,yhat5))
```

```
Profundidad del arbol:  20
Eficiencia del modelo:  0.9862008542328332
--------------------------------------------------
Matriz de confusion
--------------------------------------------------
[[651   0   0   0   0  10   0   0   0]
 [  5 649   0  10   4   4   1   1  13]
 [  0   0 687   0   0   0   0   0   0]
 [  2  20   0 592   9  17  11   0   7]
 [  0   0   0   0 660   0   0   0   0]
 [  0   0   0   4   0 696   6   0   0]
 [  0   0   0   0   0   0 669   0   0]
 [  0   0   0   0   0   0   0 674   0]
 [  0   0   0   6   0   0   0   0 680]]
--------------------------------------------------
Reporte de clasificación
--------------------------------------------------
              precision    recall  f1-score   support

       Bream       0.99      0.98      0.99       661
     Crucian       0.97      0.94      0.96       687
      Minnow       1.00      1.00      1.00       687
       Perch       0.97      0.90      0.93       658
        Pike       0.98      1.00      0.99       660
       Roach       0.96      0.99      0.97       706
        Rudd       0.97      1.00      0.99       669
       Tench       1.00      1.00      1.00       674
       Trout       0.97      0.99      0.98       686

    accuracy                           0.98      6088
   macro avg       0.98      0.98      0.98      6088
weighted avg       0.98      0.98      0.98      6088
```

Mejores caracteristicas

In [204…

```
feature_importanceB = pd.Series(model_dt5.feature_importances_, index = XB.columns).sc
print(feature_importanceB)
```

```
Lake                    0.200888
LC                      0.157491
BH                      0.130431
BW                      0.115176
TL                      0.113520
No                      0.097655
Gear                    0.076241
Date                    0.075299
SEX                     0.020754
Peagic_cladocerans      0.005740
Benthic_invertebrates   0.005622
Fishing_Zone            0.001183
Copepods                0.000000
Benthic_cladocerans     0.000000
Chironomid_larvae       0.000000
Sediment                0.000000
Plant                   0.000000
Pelagic_invertebrates   0.000000
Gastropods              0.000000
dtype: float64
```

In [205…
```
#Seleccionamos las carateristicas mas importantes
top_featuresB = feature_importanceB[feature_importanceB >0.1].index
```

In [206…
```
top_featuresB
```

Out[206]:
```
Index(['Lake', 'LC', 'BH', 'BW', 'TL'], dtype='object')
```

In [218…
```
#Reducimos conjunto de datos a las caracteristicas mas importantes
X_train_reducedB = X_trainNB[top_featuresB]
X_test_reducedB = X_testNB[top_featuresB]

# Creando modelo de Árbol de decisión
tree_classifier_reducedB = DecisionTreeClassifier(criterion= 'gini', max_depth= 24, ma
#Entrenando modelo
tree_classifier_reducedB.fit(X_train_reducedB,y_trainNB)

#Predecimos el modelo
y_pred_reducedB = tree_classifier_reducedB.predict(X_test_reducedB)

# Profundidad del árbol
print("Profundidad del arbol: ", tree_classifier_reducedB.get_depth())
# Eficiencia del modelo
print("Eficiencia del modelo: ",tree_classifier_reducedB.score(X_train_reducedB,y_trai

# Matriz de confusión
cm5 = confusion_matrix(y_testNB,y_pred_reducedB)
print("-"*50)
print("Matriz de confusion")
print("-"*50)
print(cm5)

#Reporte de clasificación
print("-"*50)
print("Reporte de clasificación")
print("-"*50)
print(classification_report(y_testNB,y_pred_reducedB))
#0.94 - 0.97
```

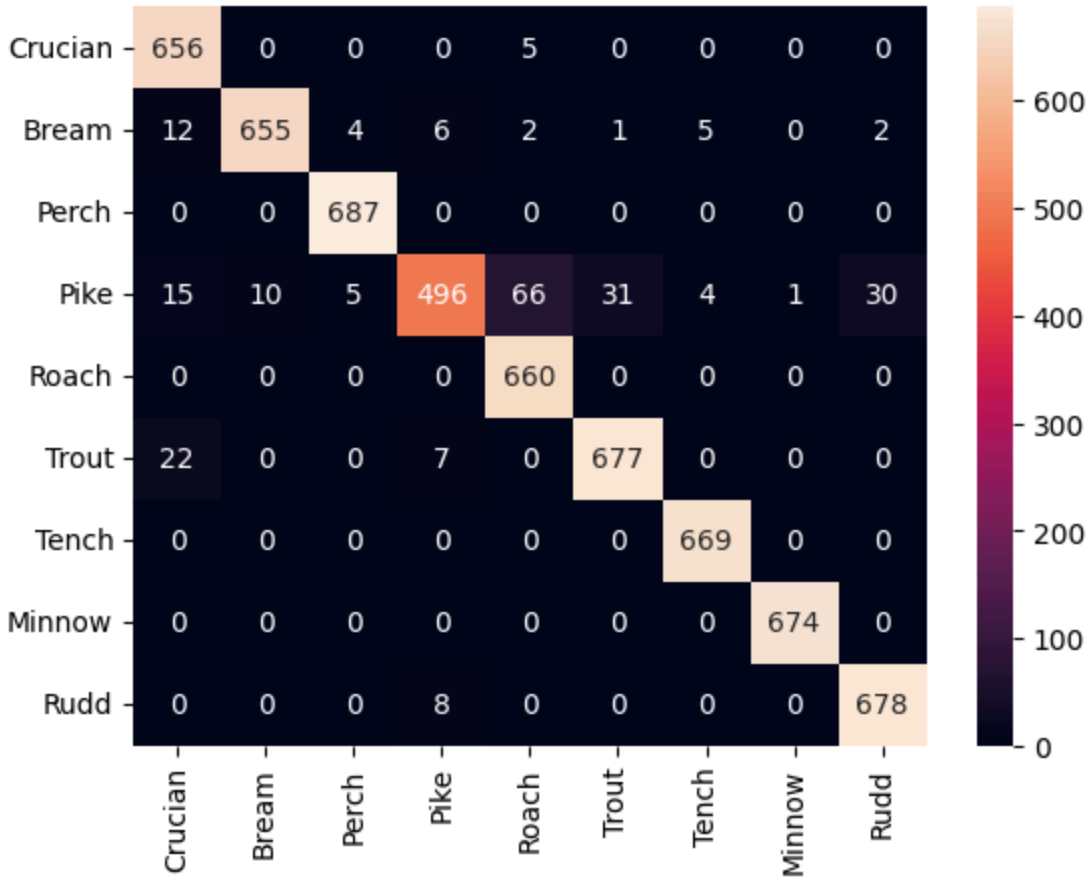
```
Profundidad del arbol:  15
Eficiencia del modelo:  0.9649545504325923
--------------------------------------------------
Matriz de confusion
--------------------------------------------------
[[656   0   0   0   5   0   0   0   0]
 [ 12 655   4   6   2   1   5   0   2]
 [  0   0 687   0   0   0   0   0   0]
 [ 15  10   5 496  66  31   4   1  30]
 [  0   0   0   0 660   0   0   0   0]
 [ 22   0   0   7   0 677   0   0   0]
 [  0   0   0   0   0   0 669   0   0]
 [  0   0   0   0   0   0   0 674   0]
 [  0   0   0   8   0   0   0   0 678]]
--------------------------------------------------
Reporte de clasificación
--------------------------------------------------
              precision    recall  f1-score   support

       Bream       0.93      0.99      0.96       661
     Crucian       0.98      0.95      0.97       687
      Minnow       0.99      1.00      0.99       687
       Perch       0.96      0.75      0.84       658
        Pike       0.90      1.00      0.95       660
       Roach       0.95      0.96      0.96       706
        Rudd       0.99      1.00      0.99       669
       Tench       1.00      1.00      1.00       674
       Trout       0.95      0.99      0.97       686

    accuracy                           0.96      6088
   macro avg       0.96      0.96      0.96      6088
weighted avg       0.96      0.96      0.96      6088
```

In [219…

```python
# Graficamos la matriz de confusión
cm6 = confusion_matrix(y_testNB, y_pred_reducedB)

cm_matrix6 = pd.DataFrame(data=cm6,
                          columns =['Crucian','Bream', 'Perch', 'Pike', 'Roach', 'Trout
                           index =['Crucian','Bream', 'Perch', 'Pike', 'Roach', 'Trout'

sns.heatmap(cm_matrix6,annot = True, fmt = "d")
```

Out[219]:

```
<Axes: >
```



Reporte Caso 3

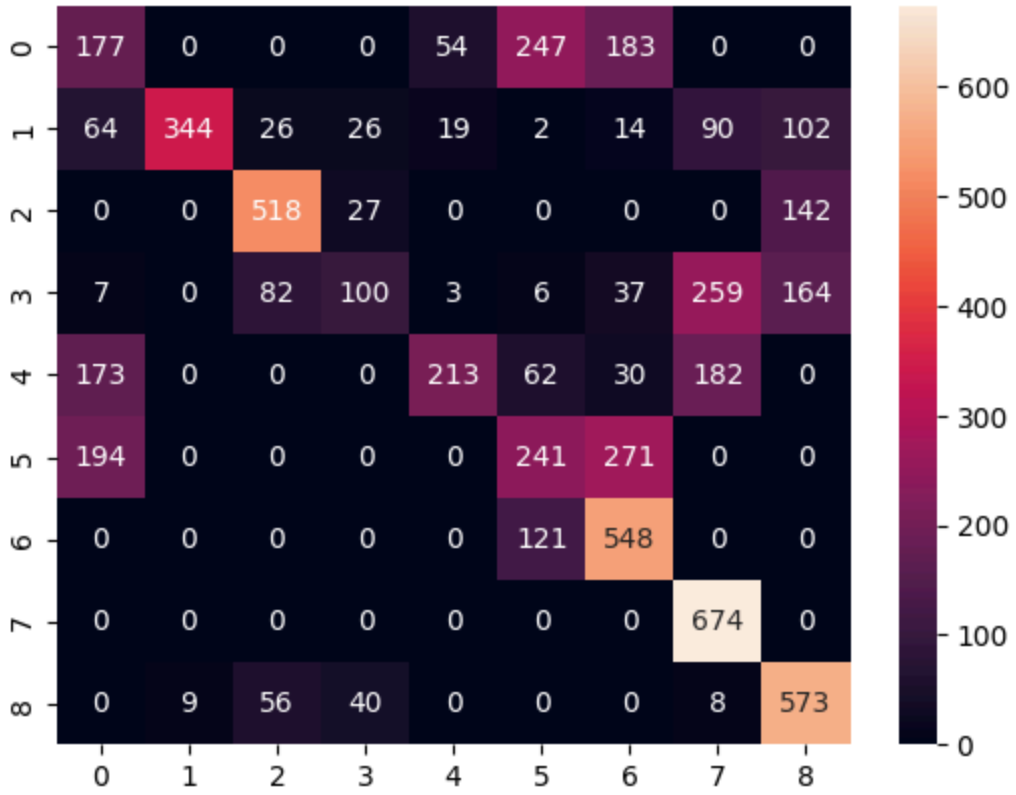
Acuraccy (Exactitud)

- La exactitud de la Regresión Logistica Multiclase es de 0.55 el cual se mantiene estable.
- La exactitud del Árbol de desición oscila entre un rango de (0.94 - 0.97)

Matriz de confusión

In [220…
```
#Matriz de confusión - Regresión Logistica Multiclase
sns.heatmap(conf_matrix5,annot = True, fmt = "d")
```

Out[220]:
```
<Axes: >
```

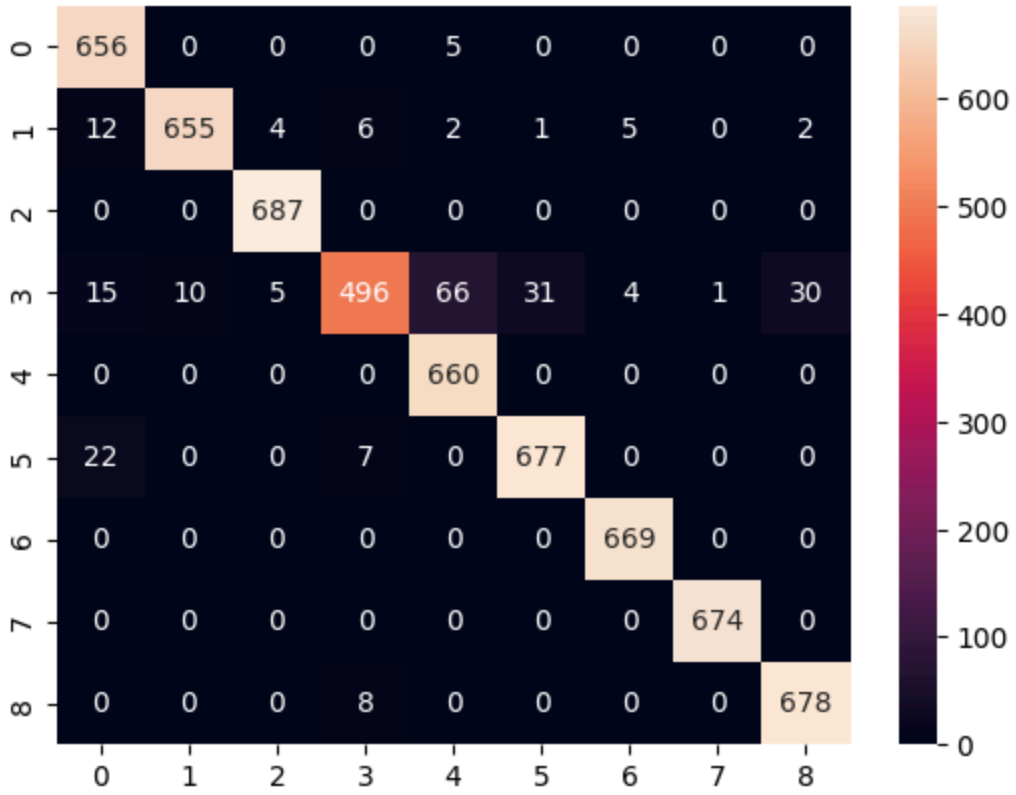


En la matriz de confusión se evidencia que logra clasificar todas las especies, donde se evidencia que la mejor clasificación surgio en las clases de "Minnow" , "Rudd", "Tench", pero tienen errores de clasificación que causa que su presición sea muy baja.

En general el modelo logra clasificar todas las especies pero no de la manera esperada.

In [221…
```
#Matriz de confusión - Arbol de desición
sns.heatmap(cm6,annot = True, fmt = "d")
```

Out[221]:
```
<Axes: >
```



En la matriz de confusión del árbol de desición se observa que es de las mejores matrices que hemos encontrado a comparación de las anteriores, en esta matriz logra clasificar las especies de manera eficiente, logrando una alto porcentaje de presición en este modelo.

Reporte de clasificación

In [222…

```
#Reporte de clasificación - Regresión Logistica Multiclase
print(classification_report(y_testNB,y_predNB))
```

```
              precision    recall  f1-score   support

       Bream       0.29      0.27      0.28       661
     Crucian       0.97      0.50      0.66       687
      Minnow       0.76      0.75      0.76       687
       Perch       0.52      0.15      0.24       658
        Pike       0.74      0.32      0.45       660
       Roach       0.35      0.34      0.35       706
        Rudd       0.51      0.82      0.63       669
       Tench       0.56      1.00      0.71       674
       Trout       0.58      0.84      0.69       686

    accuracy                           0.56      6088
   macro avg       0.59      0.55      0.53      6088
weighted avg       0.59      0.56      0.53      6088
```

- La especie "Crucian" es la que al principio empieza clasificando correctamente con una presición de 0.97, pero al trancurrir el tiempo su presición disminuye significativamente. A diferencia que las especies "Tench" y "Trout" empiezan con una presición baja pero aumentan con el tiempo.

- Logra clasificar todas las especies, pero con una presición poco confiable

In [223…
```
#Reporte de clasificación - Arbol de desición
print(classification_report(y_testNB,y_pred_reducedB))
```

```
              precision    recall  f1-score   support

       Bream       0.93      0.99      0.96       661
     Crucian       0.98      0.95      0.97       687
      Minnow       0.99      1.00      0.99       687
       Perch       0.96      0.75      0.84       658
        Pike       0.90      1.00      0.95       660
       Roach       0.95      0.96      0.96       706
        Rudd       0.99      1.00      0.99       669
       Tench       1.00      1.00      1.00       674
       Trout       0.95      0.99      0.97       686

    accuracy                           0.96      6088
   macro avg       0.96      0.96      0.96      6088
weighted avg       0.96      0.96      0.96      6088
```

- Se ve que en todas las especies se logro clasificar de manara correcta obteniendo una presición por encima de 0.90 en todas las variables.
- La unica especie que va a dismuniur su presición es "Perch"

Resumen Caso 3

- En general en este caso 3, se logro observar que el mejor metodo de clasificación para resolver la problematica es el árbol de desición obteniendo una eficiencia del modelo de que oscila entre (0.94 - 0.97) y una exactitud que oscila entre (0.94 - 0.96)respectivamente.
- Gracias a que se balanceo la data,se normalizo la data, se utilizarón los hiperparámetros y las mejores caracteristicas que se ajustaban a este modelo,se obtuvo un entrenamiento y una predicción eficientes que eran las esperadas evidenciando los resultados obtenidos en la matriz de confusión y en el reporte de clasificación del árbol de desición.
- Eso nos indica que el modelo de árbol de desición del caso 3 es el opción más optima para resolver a la problematica propuesta.

Modelo escogido para resolver la pregunta establecida

Se escogio el modelo de árbol de desición del caso 3, porque este modelo presenta un alto rendimiento con una eficacia del 97% y una exactitud del 96%. Estos son resultados muy

prometerdores y demuestran que el modelo es muy efectivo a la hora de clasificar las especies de peces dependiendo de sus caracteristicas fisicas.

Al pesar de que en algunas especies su presició va a variar a lo largo del tiempo, el modelo logra clasificar todas las especies con una presición por encima del 90%.

Tambien en las mejores caracteristicas se tomaron en cuenta las 3 variables propuestas en la pregunta como es el (Alto - Largo - Peso) de los peces.

Al utilizar varias técnicas de procesamiento de datos como el balanceo, la normalización, los hiperparámetros y la selección de las mejores caracteristicas; se noto una mejoria en el rendimiento del modelo mejorando su eficacia y presición.

En general obteniendo todos estos resultado se considera que el árbol de desición del caso 3 es la opción más optima para resolver la pregunta propuesta de la clasificación de peces con base en sus caracteristicas fisicas.