Machine Learning Algorithms – Fish Classification Dataset

Introduction

This documentation goes over in basic detail on the project that I have worked on. The project revolves around the use of four clustering methods namely:

- KMeans
- DBScan
- Agglomerative Clustering
- Gaussian Mixture

These clustering methods are implemented with the help of the python libraries and placing in appropriate parameters according to the task such as the number of clusters and epsilon values. Towards the end of the project, we will do a comparison to see which clustering project closely resembles the final clustering group values more closely. This will allow us to pick the best clustering group filtering method.

Working with the Code

We start off by implementing the basic libraries that every data scientist would require which includes:

```
import numpy as np
from scipy import stats
import pandas as pd
import seaborn as sns
```

```
#Importing all packages needed for matplotlib
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap
from matplotlib.patches import Patch
```

```
#Importing all the libraries and tools that we need for sklearn
import sklearn
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.cluster import DBSCAN
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
from sklearn.mixture import GaussianMixture
```

All these packages play a contributive role towards the data analysis that we will perform as we go along. These are mainly used to clean our datasets and ensure that only relevant pieces of data remain before we start implementing them with our machine learning models.

1. **Numpy:** A library that helps you to do fast operations on arrays and provides a multidimensional array object.
2. **Pandas:** Cleans your data and helps to make it more relevant. When you work with data science, it is so important to ensure that your data is relevant.
3. **Matplotlib:** Good tool for the creation of interactive Python visualizations and helps to create static and animated graphs.
4. **Scipy:** Used for scientific and mathematical computations and it is built off the Numpy library. Some of these packages include cluster, fftpack, constants, etc.
5. **Seaborn:** Built on top of the matplotlib library and helps with the statistical analysis of graphs. It works really well with the pandas data frame.
6. **SkLearn:** It is a machine learning library in Python and makes use of algorithms such as regression, classification and clustering algorithms which includes KMeans, Gaussian Mixture, Agglomerative Clustering, and DBScan.

To ensure that data is communicated well and is well-designed with similar colors, I have included palettes in my code with color codes accompanied by their hashtag values.

```
#Creating all the palette that we will need
palette_1 = ["#f72585", "#b5179e", "#7A0CC3", "#560bad", "#330A92", "#3f37c9", "#4361ee", "#4895ef", "#4cc9f0"]
palette_2 = ["#2C7B7B", "#004D4A", "#D02748", "#F9E03B", "#F0C808", "#D5A2D5", "#3E2A63", "#6BC6D3", "#007BA8"]


sns.set_theme(context="notebook", palette=palette_2, style="darkgrid")
```

Since our file is a CSV file, we are making use of the read_csv function to read our files. If our file was an EXCEL file, we would be utilizing the read_excel function instead.

```
#Reading our CSV file
df=pd.read_csv(r"C:/Users/samsung/fish_clustering_visualization/fish_data.csv")
```

Followed by this, we perform methods such as head(), info() and describe() to better understand our data. To work with the null values, we have checked if there are any null values that are present in our code.

```
df.isnull().sum()
```

```
species      0
length       0
weight       0
w_l_ratio    0
dtype: int64
```

Since there are no null values present, no further cleaning needs to be done to remove any null values. We must check to see if there are any duplicated values that are present. Upon checking our code as shown below, there were about 109 rows that had duplicated values within them. Hence, we made use of the drop_duplicates to remove off any of those duplicated rows.

```
df.duplicated().sum()
```

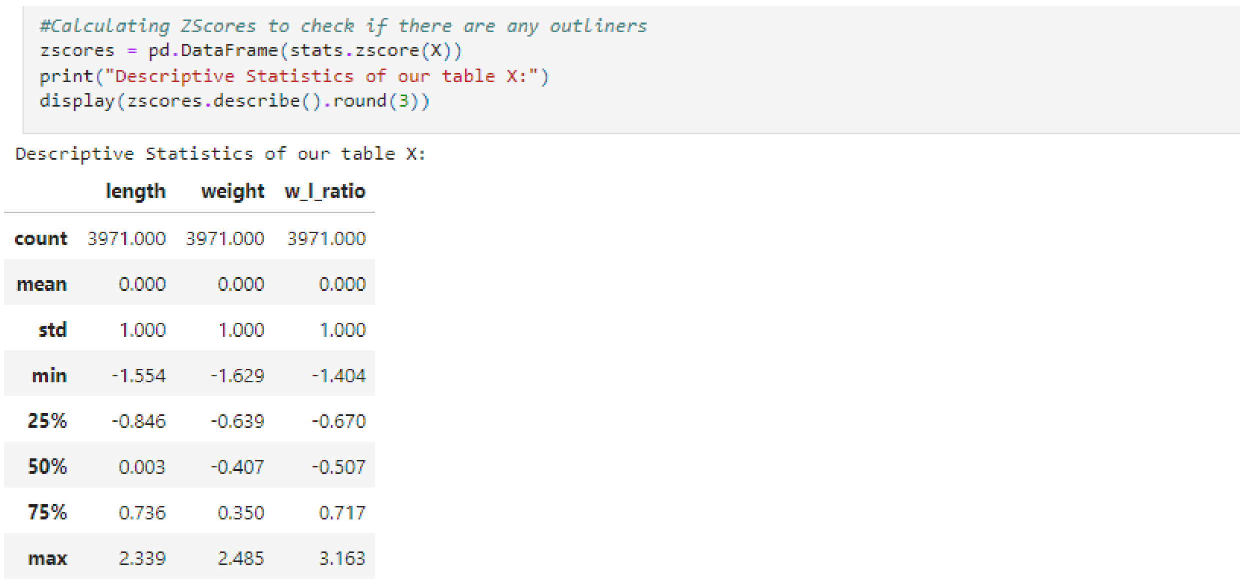
```
109
```

```
df.drop_duplicates(inplace=True)
```

We then create two data frames called X and y whereby X has the entire data frame without the “species” column and the y dataset only has the “species” column. All this is needed to be able to generate our true clustering graph at the very end with our “y” dataset and perform machine learning algorithms from the sklearn library with our “X” dataset.

```
X = df.drop("species", axis=1) #dataframe without any species column
y = df.species #dataframe with only species column
```

After this, we are checking our z_score values to see if there are any outliners that are present inside our data.



```
#Calculating ZScores to check if there are any outliners
zscores = pd.DataFrame(stats.zscore(X))
print("Descriptive Statistics of our table X:")
display(zscores.describe().round(3))
```

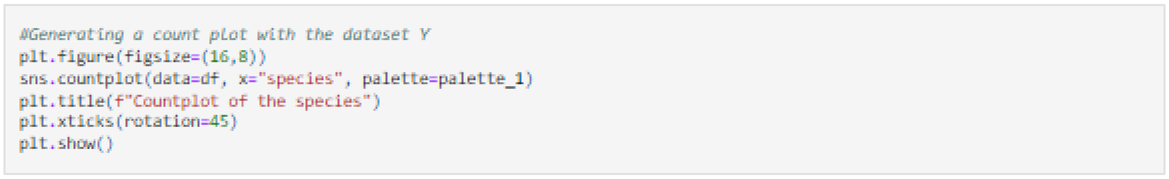
```
Descriptive Statistics of our table X:
```

	length	weight	w_l_ratio
count	3971.000	3971.000	3971.000
mean	0.000	0.000	0.000
std	1.000	1.000	1.000
min	-1.554	-1.629	-1.404
25%	-0.846	-0.639	-0.670
50%	0.003	-0.407	-0.507
75%	0.736	0.350	0.717
max	2.339	2.485	3.163

Next, we are going to create a histplot with the help of our seaborn package to visually represent each of the columns that we will be working with in our dataset. It is accompanied with the probability density curve as well to show the distribution of the continuous variables on the data range. By iterating through the columns in our "X" labeled dataset, we are allowing our code to investigate the individual columns of the dataset and produce a histplot accordingly. Through the code in our Jupyter Notebook as shown on my GitHub repository, you will have the opportunity to visually see those histplots.

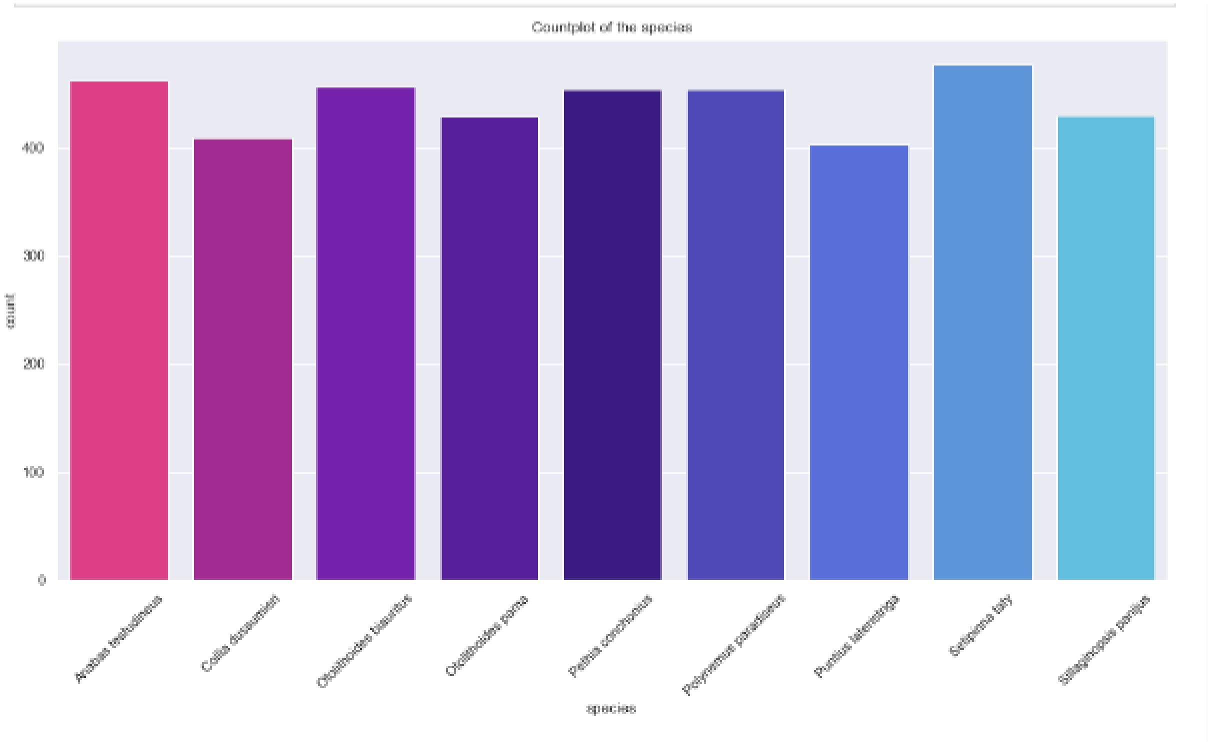
```
for columns in X:
    plt.figure(figsize=(8,5))
    sns.histplot(data=df,x=columns,kde=True, bins=20, color=palette_1[4])
    plt.title(f"Histogram of {columns}")

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

The countplot is used to showcase the number of species in specific groups and with the help of this graph, we can derive that we have nine groups/ clusters of species in total that we have to work with.



```
#Generating a count plot with the dataset Y
plt.figure(figsize=(16,8))
sns.countplot(data=df, x="species", palette=palette_1)
plt.title(f"Countplot of the species")
plt.xticks(rotation=45)
plt.show()
```

The produced graph appears in this way:



By instantiating a scaler, we can take all the data values inside our “X” dataset, and we are re-scaling them between 0 and 1. This allows us to assess the data values on a common scale instead of having a scale that is varied in nature. With this common scale, we have a better idea of where our data is sitting on that numbered value. After doing so, our data values that have been scaled are now sitting inside “X_scaled.”

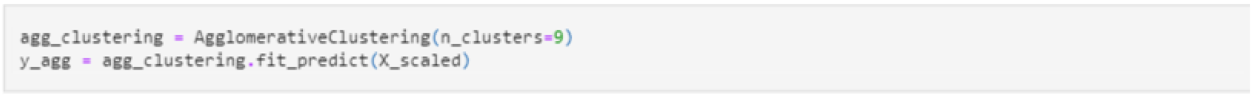
```
#Instantiating a Scaler
scaler = MinMaxScaler()

#Fitting all values of X into a Scaler
X_scaled = scaler.fit_transform(X)
```

```
kmeans = KMeans(n_clusters=9, max_iter=100)
y_kmeans = kmeans.fit_predict(X_scaled) #Predicting the cluster labels

X_transformed = scaler.inverse_transform(X_scaled) #Placing it back to the original scale of data
```

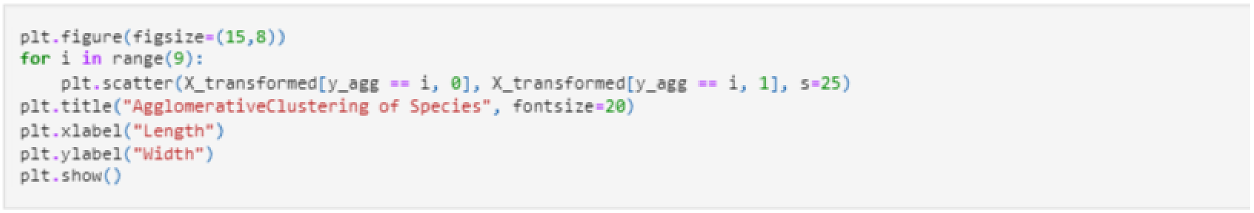
```
# Plotting the results
plt.figure(figsize=(15, 8))
for i in range(9):
    plt.scatter(X_transformed[y_kmeans == i, 0], X_transformed[y_kmeans == i, 1], s=25)

plt.title("K Means Clustering of the species", fontsize=20)
plt.xlabel("Length")
plt.ylabel("Weight")

plt.show()
```

In the code above, we have made use of KMeans function from the sklearn package and have instantiated the algorithm by creating nine clusters and will allow for 100 iterations at maximum. We then go ahead to plot all the values accordingly.

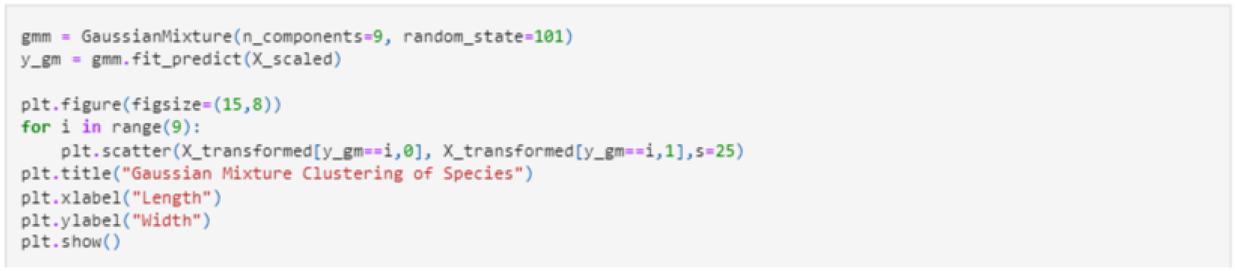
This is done in a similar nature with DBScan, GaussianMixture, and Agglomerative Clustering as such.



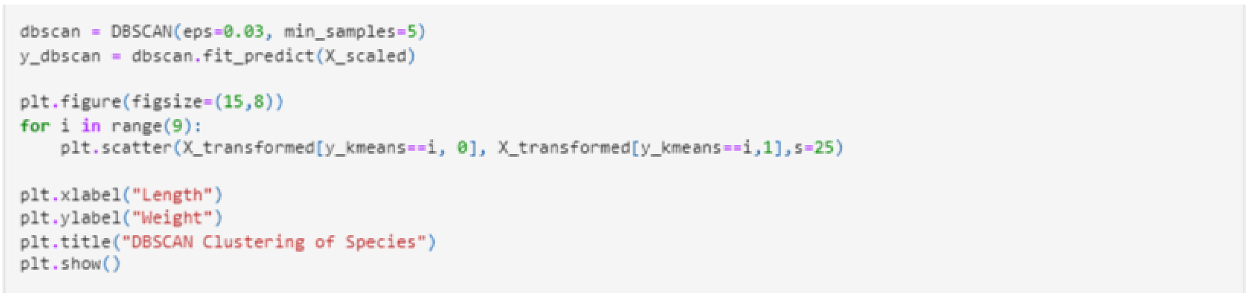
```
agg_clustering = AgglomerativeClustering(n_clusters=9)
y_agg = agg_clustering.fit_predict(X_scaled)
```



```
plt.figure(figsize=(15,8))
for i in range(9):
    plt.scatter(X_transformed[y_agg == i, 0], X_transformed[y_agg == i, 1], s=25)
plt.title("AgglomerativeClustering of Species", fontsize=20)
plt.xlabel("Length")
plt.ylabel("Width")
plt.show()
```



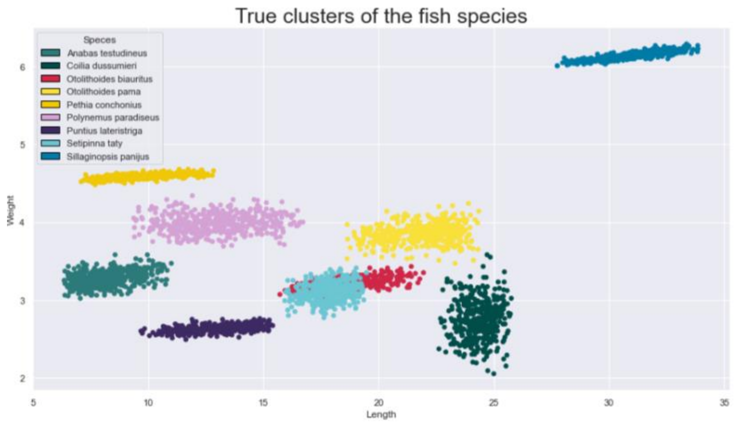
```
gmm = GaussianMixture(n_components=9, random_state=101)
y_gm = gmm.fit_predict(X_scaled)

plt.figure(figsize=(15,8))
for i in range(9):
    plt.scatter(X_transformed[y_gm==i,0], X_transformed[y_gm==i,1],s=25)
plt.title("Gaussian Mixture Clustering of Species")
plt.xlabel("Length")
plt.ylabel("Width")
plt.show()
```

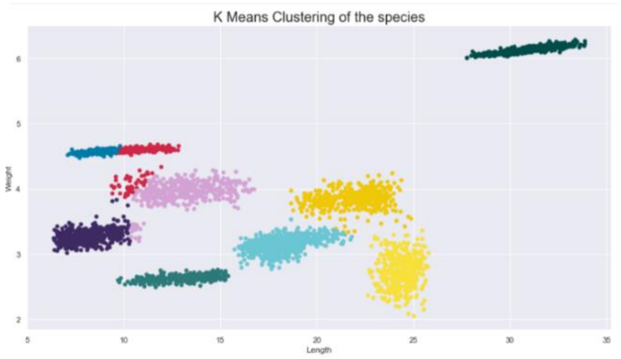


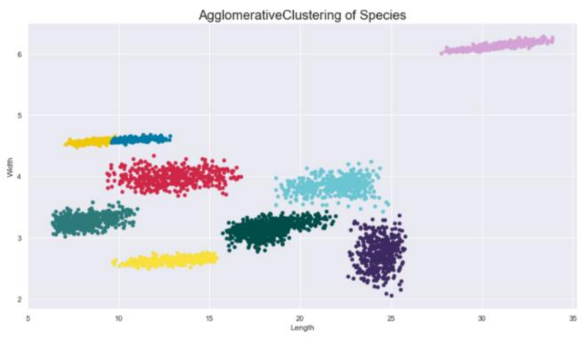
```
dbscan = DBSCAN(eps=0.03, min_samples=5)
y_dbscan = dbscan.fit_predict(X_scaled)

plt.figure(figsize=(15,8))
for i in range(9):
    plt.scatter(X_transformed[y_kmeans==i, 0], X_transformed[y_kmeans==i,1],s=25)

plt.xlabel("Length")
plt.ylabel("Weight")
plt.title("DBSCAN Clustering of Species")
plt.show()
```

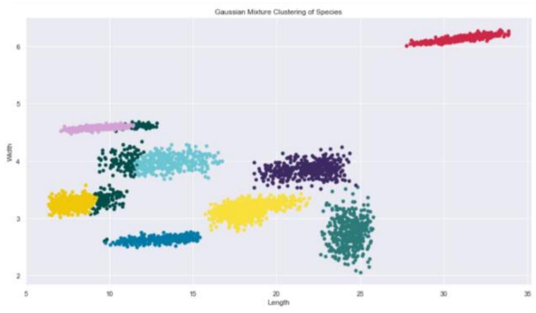
Once we have done through our four clustering methods that we wanted to implement, we will now work with our true clustering. This is done by making use of the LabelEncoder and then applying its labels onto our data.

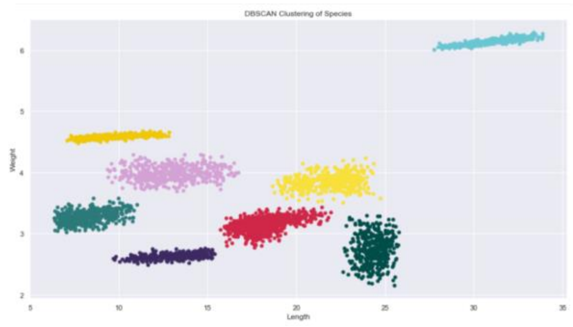
```
# Finding the true clustering between the species

#Instantiating a LabelEncoder() to be used on the dataset
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

colors = [palette_2[i] for i in y_encoded] #Should be 9 labels only

plt.figure(figsize=(15,8))
scatter = plt.scatter(X.iloc[:, 0], X.iloc[:,1], c=colors, s=25)

plt.title("True clusters of the fish species", fontsize=25)
plt.xlabel("Length")
plt.ylabel("Weight")

legend_labels = label_encoder.classes_

legend_elements = [Patch(facecolor=palette_2[i], edgecolor="k", label=legend_labels[i]) for i in range (len(legend_label

plt.legend(handles=legend_elements, title="Speces")

plt.show()
```











From this, we can be able to observe that the DBScan clustering filter method is the best method out of the four since it is able to clearly differentiate between all the cluster groups. When compared with the true clustering graph, it can also be seen that the DBScan filter method resembles more closely with the true clustering filtering method.