ASSIGNMENT 1 - REPORT

Data manipulation + Exploratory data analysis

Chirag Hebbal Rudresh
ID: 1002160960

9/24/23

DASC 5301-002 Data Science

INDEX

ASSIGNMENT 1 REPORT

This assignment focuses data cleaning by removing inaccurate data from the traffic cameras data set. Data wrangling focuses on transforming the data's format by converting "raw" data into usable format more suitable for getting insights. This assignment uses the traffic_cameras csv file. The data will be modified and insights from this data can be extracted.

PART 1: Cleaning and Wrangling data

```
#read in libraries
import numpy as np
# from sklearn.datasets import load_iris
from sklearn import preprocessing
import pandas as pd
```

First, we import the libraries required for the process of data analysis.

Numpy module is used since it has efficient multidimensional array operations, and it has a wide range of mathematical function which are used for computing and data analysis. It is imported with the alias np.

Preprocessing module from scikit-learn is imported. It contains various tools for data preprocessing, scaling, and transformation. We can use the functions and classes provided by this module for tasks like feature scaling, label encoding, and data transformation.

Pandas is python library for data manipulation and analysis. It provides powerful and flexible tools for working with structured data, making it a fundamental library in the field of data science, data analysis, etc. It is mainly used for data exploration, cleaning, transformation, and analysis purposes. It is imported with the alias pd.

```
# Reading the CSV file
df = pd.read_csv("traffic_cameras.csv")

# Printing top 5 rows
df.head()
```

The dataset to be analysed is first equated to pd.read_csv() i.e df= pd.read_csv((function that is used to get the data from the csv file into the notebook environment. If the csv file is in the same folder(location) as that of the python notebook, we can just give the name of the file as argument in the function. For example, in this case it will pd.read_csv('traffic_cameras.csv"). In case the data file is at a different location on the computer, we need to provide the path for that file as argument in the function.

We can check if the data is properly imported into the notebook by checking the values of the file that is read using pandas. The df.head() function displays the first 5 rows of the data and the column names and data can be verified.

1. **To calculate the number of rows and columns in the dataset**: To get the information regarding rows and columns df.shape function can be used which will give the output in (row, column) format. Here, the number of rows and columns was found to be (802 rows, 28 columns).
   Alternatively, when the whole dataset is printed, at the bottom of the output the following information is also printed: [802 rows x 28 columns].

2. **To find out variable types**: To find the type of variables in the dataset, inbuilt function of df.dtypes is used. It provides information about the data types of all the columns in the dataset. In this case, it was found that majority of the variable were of object data type and rest were float data type, excluding camera id column which was of type integer.

```
### Your code goes here ###
df.dtypes # gives the data types of values present in each column
```

```
Camera ID                        int64
Location Name                   object
Camera Status                   object
Turn on Date                    object
Camera Manufacturer             object
ATD Location ID                 object
Landmark                        object
Signal Engineer Area            object
Council District                object
Jurisdiction                    object
Location Type                   object
Primary St Segment ID          float64
Cross St Segment ID            float64
Primary Street Block           float64
Primary Street                  object
PRIMARY_ST_AKA                 float64
Cross Street Block             float64
Cross Street                    object
CROSS_ST_AKA                   float64
COA Intersection ID            float64
Modified Date                   object
IP Comm Status                  object
IP Comm Status Date and Time    object
Published Screenshots          float64
Screenshot Address              object
Funding                         object
ID                              object
Location                        object
dtype: object
```

3. **To delete columns that have all null values**: The pandas function *pd.isnull(df).all()* is used which checks in which column all values are null. If any column has all values as null, the output will be in the following format: *Column name True* .

   Alternatively, *df.info()* can be used to check if any column contains all non-null values. Here the count of non-null values is displayed as output. If the number of non-null values in any column is not equal to number of rows in dataset, then those columns contain null values. So, in this case we look for column which displays 0 non-null values and conclude that that column has all null values.

   After identifying the columns that need to be removed, *df.drop(columns=['column name']* function is used and this is equated to df1. The following columns are dropped:
   ['Primary St Segment ID','Cross St Segment ID', 'PRIMARY_ST_AKA', 'CROSS_ST_AKA', 'Published Screenshots'].

4. **Dropping columns with any number of null values**: In this case the function *df1.dropna(axis=1)*
is used and it is stored in variable df2 axis=1 implies to delete the column to which null value belongs to.
The *dropna* function is a method in the Pandas library for Python used to remove missing or NaN (Not-a-Number) values from a DataFrame. After dropping these columns, the dataset will be left with 10 columns.

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 802 entries, 0 to 801
Data columns (total 23 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Camera ID                     802 non-null    int64  
 1   Location Name                 802 non-null    object 
 2   Camera Status                 802 non-null    object 
 3   Turn on Date                  442 non-null    object 
 4   Camera Manufacturer           646 non-null    object 
 5   ATD Location ID               802 non-null    object 
 6   Landmark                      94 non-null     object 
 7   Signal Engineer Area          799 non-null    object 
 8   Council District              790 non-null    object 
 9   Jurisdiction                  799 non-null    object 
 10  Location Type                 802 non-null    object 
 11  Primary Street Block          800 non-null    float64
 12  Primary Street                801 non-null    object 
 13  Cross Street Block            757 non-null    float64
 14  Cross Street                  765 non-null    object 
 15  COA Intersection ID           740 non-null    float64
 16  Modified Date                 802 non-null    object 
 17  IP Comm Status                802 non-null    object 
 18  IP Comm Status Date and Time  802 non-null    object 
 19  Screenshot Address            802 non-null    object 
 20  Funding                       750 non-null    object 
 21  ID                            802 non-null    object 
 22  Location                      802 non-null    object 
dtypes: float64(3), int64(1), object(19)
memory usage: 144.2+ KB
```

```
df2.info()
df2.shape
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 802 entries, 0 to 801
Data columns (total 11 columns):
 #   Column                        Non-Null Count  Dtype 
---  ------                        --------------  ----- 
 0   Camera ID                     802 non-null    int64 
 1   Location Name                 802 non-null    object
 2   Camera Status                 802 non-null    object
 3   ATD Location ID               802 non-null    object
 4   Location Type                 802 non-null    object
 5   Modified Date                 802 non-null    object
 6   IP Comm Status                802 non-null    object
 7   IP Comm Status Date and Time  802 non-null    object
 8   Screenshot Address            802 non-null    object
 9   ID                            802 non-null    object
 10  Location                      802 non-null    object
dtypes: int64(1), object(10)
memory usage: 69.1+ KB
(802, 11)
```

5. **Renaming columns in df2:** The copy of df2 data frame will be used further of manipulation. A copy is used so that the original data need to be changed because it might be required in any other process at a later stage. The function *df2.copy()* is used for this process and the copy is stored in variable df3.

Now, to rename the columns, the following function and syntax is used:

- *df3.columns=['cam_id','loc_name','cam_stat','atd_loc_id','loc_type','date','comm_stat','comm_stat_date','screen_addr','id','location']*.

Here, all remaining 10 columns have been renamed.

```
df3.head()
```

	cam_id	loc_name	cam_stat	atd_loc_id	loc_type	date	comm_stat	comm_stat_date	screen_addr	id
0	370	PLEASANT VALLEY RD / NUCKOLS CROSSING RD	TURNED_ON	LOC16-003180	ROADWAY	10/28/2021 08:40:00 AM +0000	ONLINE	10/28/2021 08:30:00 AM +0000	https://cctv.austinmobility.io/image/370.jpg	591a10a020eacf2d16669b94
1	379	BARTON SPRINGS RD / KINNEY AVE	TURNED_ON	LOC16-000640	ROADWAY	10/29/2021 08:45:00 AM +0000	ONLINE	10/29/2021 08:35:00 AM +0000	https://cctv.austinmobility.io/image/379.jpg	591a10a020eacf2d16669ba6
2	404	SPRINGDALE RD / OAK SPRINGS DR	TURNED_ON	LOC16-000800	ROADWAY	10/29/2021 07:38:00 PM +0000	ONLINE	10/28/2021 08:35:00 AM +0000	https://cctv.austinmobility.io/image/404.jpg	591a10a120eacf2d16669bd8
3	447	BRAKER LN / STONELAKE BLVD	TURNED_ON	LOC16-003740	ROADWAY	10/29/2021 07:49:00 PM +0000	ONLINE	10/23/2021 08:35:00 AM +0000	https://cctv.austinmobility.io/image/447.jpg	591a10a320eacf2d16669c2e
4	552	EXPOSITION BLVD / WESTOVER RD	TURNED_ON	LOC16-003710	ROADWAY	10/29/2021 07:47:00 PM +0000	ONLINE	10/20/2021 08:35:00 AM +0000	https://cctv.austinmobility.io/image/552.jpg	5aa6bb0121cbcf4b8b767294

6. **Splitting date column into two new columns:** The *str.split* function was used for this process.

- *df3['Date']= df3['date'].str.split(' ').str[0]*.

In above code, we are splitting the data based on delimiter space (' ') which has to be given as an attribute in the brackets and at the end *.str[ ]* can be used to select which part of the split value we want to save into the new column.

- *df3['Time']= df3['date'].str.split(' ').str[1]*

- *df3['time_of_day']= df3['date'].str.split(' ').str[2]*

This is for extracting the AM or PM from the string.

- *df3['ex_time_data']=df3['date'].str.split(' ').str[3]*.

This is for extracting the AM or PM from the string.

- *df3['Time']= df3['Time']+' '+df3['time_of_day']+df3['ex_time_data']*

In the above code, we are updating the Time column with the old data plus new columns data.

The process above is not very efficient, so alternatively we can use:

- *df3[['Date','Time']]=df3['date'].str.split(' ', expand=True, n=1)*

This code snippet creates two new columns and the split data from 'date' column will be put in the required new columns. The attribute n=1 specifically informs the program to split the string into 2 halves at the first occurrence of the delimiter ' ' (space). Expand=True argument tells the program to split the string and put in the new columns.

Date	Time	time_of_day	ex_time_data
10/28/2021	08:40:00 AM+0000	AM	+0000
10/29/2021	08:45:00 AM+0000	AM	+0000
10/29/2021	07:38:00 PM+0000	PM	+0000
10/29/2021	07:49:00 PM+0000	PM	+0000
10/29/2021	07:47:00 PM+0000	PM	+0000

7. **To split atd_loc_id into two new columns:** Similar to the above task, str.split() function is used. But here the delimiter will be hyphen (-).
    - df3['LOC']=df3['atd_loc_id'].str.split('-').str[0
    - df3['code']=df3['atd_loc_id'].str.split('-').str[1]

atd_loc_id	LOC	code
LOC16-003180	LOC16	003180
LOC16-000640	LOC16	000640
LOC16-000800	LOC16	000800

8. **Unique values in loc_type column:** For this task, .unique() function will be used. The syntax is as follows: *df3.loc_type.unique()*. The output says there are only two unique values that are 'ROADWAY', 'BUILDING'.

```
### Your code goes here ###
df3.loc_type.unique() #function to check uni
```

```
array(['ROADWAY', 'BUILDING'], dtype=object)
```

9.
10. **Replace ROADWAY and BUILDING in loc_type column:** This can be done in 2 ways:
    a. we use the replace function to replace an existing value.
    b. We can use the lambda function by giving it conditions to execute on

For first method, the replace function is used:

- *df3['loc_type'].replace('ROADWAY',0,inplace=True)*

Inplace is used to alter the original DataFrame itself.

- *df3['loc_type'].replace('BUILDING',1,inplace=True)*

Alternatively, we can use the lambda function:

df3['loc_type']=df3['loc_type'].map(lambda x : 0 if (x =='ROADWAY') else x)

df3['loc_type']=df3['loc_type'].map(lambda x : 1 if (x =='BUILDING') else x)

loc_type
ROADWAY
ROADWAY
ROADWAY

loc_type
0
0
0

11. **Splitting loc_name column:**
    The str.split() method will be used by giving delimiter as '/ '. The following code is used:
    df3[['corner1','corner2']]=df3['loc_name'].str.split('/', expand=True, n=1)

loc_name
PLEASANT VALLEY RD / NUCKOLS CROSSING RD
BARTON SPRINGS RD / KINNEY AVE
SPRINGDALE RD / OAK SPRINGS DR
BRAKER LN / STONELAKE BLVD

corner1	corner2
PLEASANT VALLEY RD	NUCKOLS CROSSING RD
BARTON SPRINGS RD	KINNEY AVE
SPRINGDALE RD	OAK SPRINGS DR
BRAKER LN	STONELAKE BLVD

This will give us the final cleaned data set in variable df3 which can be used in next steps of data analytics.

PART 2: Exploratory data analysis (EDA)

This part of the assignment deals exploratory data analysis for the Fish data set. EDA involves the exploration, visualization, and summarization of data to understand its main characteristics or features, detect patterns, identify anomalies, and gain insights into the underlying structure of the dataset.

First, we import all the required libraries which will be used for upcoming analysis: **import seaborn as sns; import pandas as pd; import matplotlib.pyplot as plt; %matplotlib inline**

By using %matplotlib inline, any Matplotlib plots created in code cells of the Jupyter Notebook will be displayed directly below the code cell that generates them. This makes it easier to study the plots and analyze the data represented in the charts.

STEP 1: First we will be loading the csv file into Jupyter notebook using pandas. After printing the first 5 rows of dataset using .head() function, we get to know the different properties of fish which are present in the dataset. Here, the columns are the features of fish.

	Species	Weight	Length1	Length2	Length3	Height	Width
0	Bream	242.0	23.2	25.4	30.0	11.5200	4.0200
1	Bream	290.0	24.0	26.3	31.2	12.4800	4.3056
2	Bream	340.0	23.9	26.5	31.1	12.3778	4.6961
3	Bream	363.0	26.3	29.0	33.5	12.7300	4.4555
4	Bream	430.0	26.5	29.0	34.0	12.4440	5.1340

So, fish has features: [species, Weight, Length1, Length2, Length3, Height and Width]

To get informed about the structure of the data, .shape function is used which returns us the number of rows and columns in the dataset. After seeing the output, it's clear that fish.csv dataset has 159 rows and 7 columns of data. The columns of the dataset can be called as features or properties of fish that have been studied while collecting the data.

```
df_fish.shape #(no. of rows and columns)
```

```
(159, 7)
```

The .describe() function is used to get some common metrics of the dataset which can be used for analysis. The image below shows the output of the describe function.

```
df_fish.describe()
```

	Weight	Length1	Length2	Length3	Height	Width
count	159.000000	159.000000	159.000000	159.000000	159.000000	159.000000
mean	398.326415	26.247170	28.415723	31.227044	8.970994	4.417486
std	357.978317	9.996441	10.716328	11.610246	4.286208	1.685804
min	0.000000	7.500000	8.400000	8.800000	1.728400	1.047600
25%	120.000000	19.050000	21.000000	23.150000	5.944800	3.385650
50%	273.000000	25.200000	27.300000	29.400000	7.786000	4.248500
75%	650.000000	32.700000	35.500000	39.650000	12.365900	5.584500
max	1650.000000	59.000000	63.400000	68.000000	18.957000	8.142000

It gives information about the mean, standard deviation, minimum, maximum, Quartiles(25%,50%,75%) of the different column values(features).

STEP 2: **Checking for missing/null values and duplicates**. For initial stages the .info() method Is used to spot if there are any null values in any of the columns in the dataset. The output confirms that there are no null values as it shows 159 non-null values are present in each column of the dataset.

To be sure, we can use isnull().sum() function to check the count of null values. This returns the count of null values, but in this case, it shows 0 for all columns. So, it's safe to say there are no null values in the fish dataset and cleaning of null values is not required.

```
df_fish.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 159 entries, 0 to 158
Data columns (total 7 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   Species  159 non-null    object 
 1   Weight   159 non-null    float64
 2   Length1  159 non-null    float64
 3   Length2  159 non-null    float64
 4   Length3  159 non-null    float64
 5   Height   159 non-null    float64
 6   Width    159 non-null    float64
dtypes: float64(6), object(1)
memory usage: 8.8+ KB
```

```
df_fish.isnull().sum()
```

```
Species    0
Weight     0
Length1    0
Length2    0
Length3    0
Height     0
Width      0
dtype: int64
```

Since we are exploring the dataset, we can check what are the unique species that are present in the data. Here, .unique() function is run on the species column to get the different fish species names. Hence, we have discovered that the data contains fish of seven species.

```
df_fish.Species.unique()# check out different species of fish in the dataset
```

```
array(['Bream', 'Roach', 'Whitefish', 'Parkki', 'Perch', 'Pike', 'Smelt'],
      dtype=object)
```

Next step is to check for duplicates of the data in the dataset. Here we will be using two functions to verify if duplicates exist or not:

i. Drop_duplicates method: In this method, the code will execute and drop rows which are duplicates of other data and return the number of rows remaining in the dataset. Hence, we can know if there were any duplicates by checking if there is difference in number of rows after executing this command.

In our case, there are no duplicates since the command execution returned back 159 rows and 7 columns.

```
df_fish.drop_duplicates #dropping duplicate rows if they exist in the datset
```

```
<bound method DataFrame.drop_duplicates of     Species  Weight  Length1  Length2  Length3
Height   Width
0     Bream   242.0     23.2     25.4     30.0  11.5200  4.0200
1     Bream   290.0     24.0     26.3     31.2  12.4800  4.3056
2     Bream   340.0     23.9     26.5     31.1  12.3778  4.6961
3     Bream   363.0     26.3     29.0     33.5  12.7300  4.4555
4     Bream   430.0     26.5     29.0     34.0  12.4440  5.1340
..      ...     ...      ...      ...      ...      ...     ...
154   Smelt    12.2     11.5     12.2     13.4   2.0904  1.3936
155   Smelt    13.4     11.7     12.4     13.5   2.4300  1.2690
156   Smelt    12.2     12.1     13.0     13.8   2.2770  1.2558
157   Smelt    19.7     13.2     14.3     15.2   2.8728  2.0672
158   Smelt    19.9     13.8     15.0     16.2   2.9322  1.8792

[159 rows x 7 columns]>
```

ii. .duplicated() function: In this method, the code returns a Boolean value(True/ False) according to the presence or absence of duplicates with their indexes.

This data set doesn't have any duplicates since for all rows, the code returned False with length as 159 rows.

```
df_fish.duplicated() # to check if any row has duplicates
```

```
0      False
1      False
2      False
3      False
4      False
       ...
154    False
155    False
156    False
157    False
158    False
Length: 159, dtype: bool
```

STEP 3: **To see if the dataset is balanced**, we will check how many data points belong to certain fish species. The .value_counts() function will be used to count the number of fish entries for the column species. The table below shows that the data points for each species isn't equal, hence the dataset is not balanced.

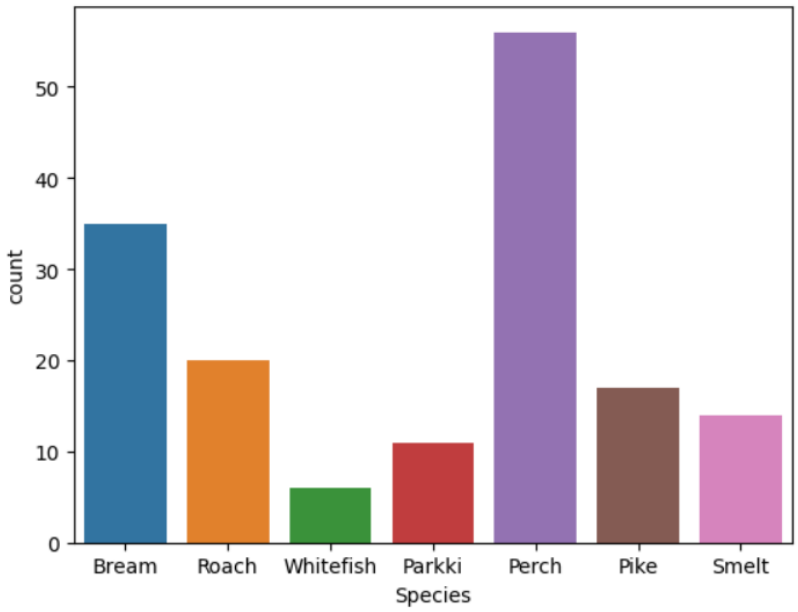
```
df_fish['Species'].value_counts()
```

```
Species
Perch        56
Bream        35
Roach        20
Pike         17
Smelt        14
Parkki       11
Whitefish     6
Name: count, dtype: int64
```

The same data can be visualized using seaborn countplot(). By using plots, it becomes easier to spot trends, outliers, abnormalities, and aids in the analysis process.

```
sns.countplot(x='Species',data=df_fish) # count plot to visualize the balance of the datas
```

```
<Axes: xlabel='Species', ylabel='count'>
```



STEP 4: Finding Relations between the features of the data set. This will help us understand the trends in the data and in some cases predict features which are unknown. Here, we will try to see what the relation is between Species of fish and other features/properties of the fish. For this process we will group the means of other features like weight, height etc. based on species of fish.

```
df_fish.groupby('Species').mean() # Find average value of features for all the species of
# we get to know on average, the Pike species fish weighs the most and Smelt species fish
```

Species	Weight	Length1	Length2	Length3	Height	Width
Bream	617.828571	30.305714	33.108571	38.354286	15.183211	5.427614
Parkki	154.818182	18.727273	20.345455	22.790909	8.962427	3.220736
Perch	382.239286	25.735714	27.892857	29.571429	7.861870	4.745723
Pike	718.705882	42.476471	45.482353	48.717647	7.713771	5.086382
Roach	152.050000	20.645000	22.275000	24.970000	6.694795	3.657850
Smelt	11.178571	11.257143	11.921429	13.035714	2.209371	1.340093
Whitefish	531.000000	28.800000	31.316667	34.316667	10.027167	5.473050

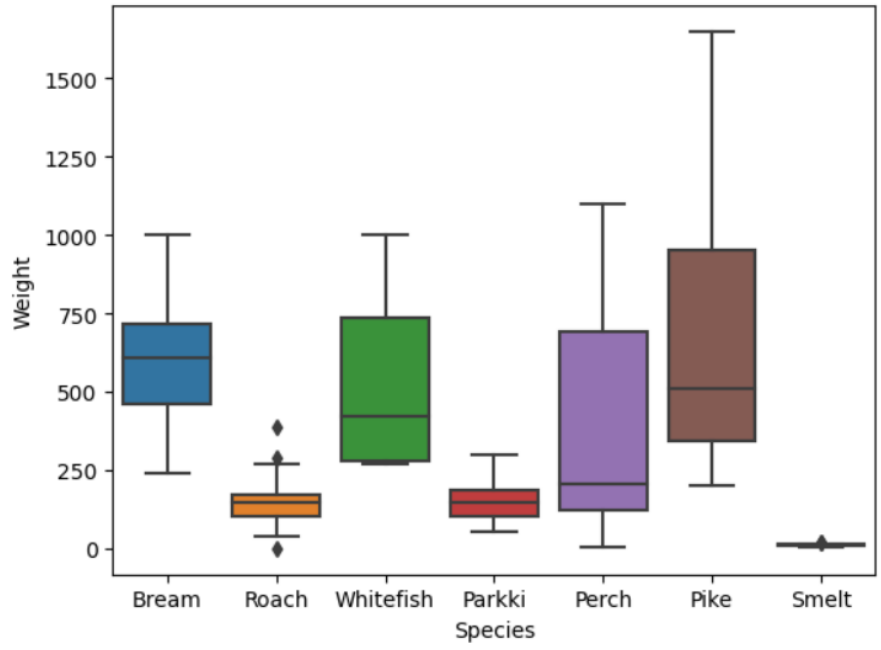
So, from the above table at first glance we can see that for some species of fish, certain characteristics are more than other species while some are less than others.

For example, the average weight of Pike fish is 718.70 grams (mostly probably weight is in grams). Whereas the Smelt species has a very low average weight of 11.17 grams. So, if a restaurant wanted to cut down on their menu of fish items, choosing Pike fish to stay on the menu would be suitable since its weight favors the restaurant business.

A box plot can be used to visualize this data.

```
sns.boxplot(data = df_fish,x = 'Species',y = 'Weight')# visualize the relation between spec
```

```
<Axes: xlabel='Species', ylabel='Weight'>
```



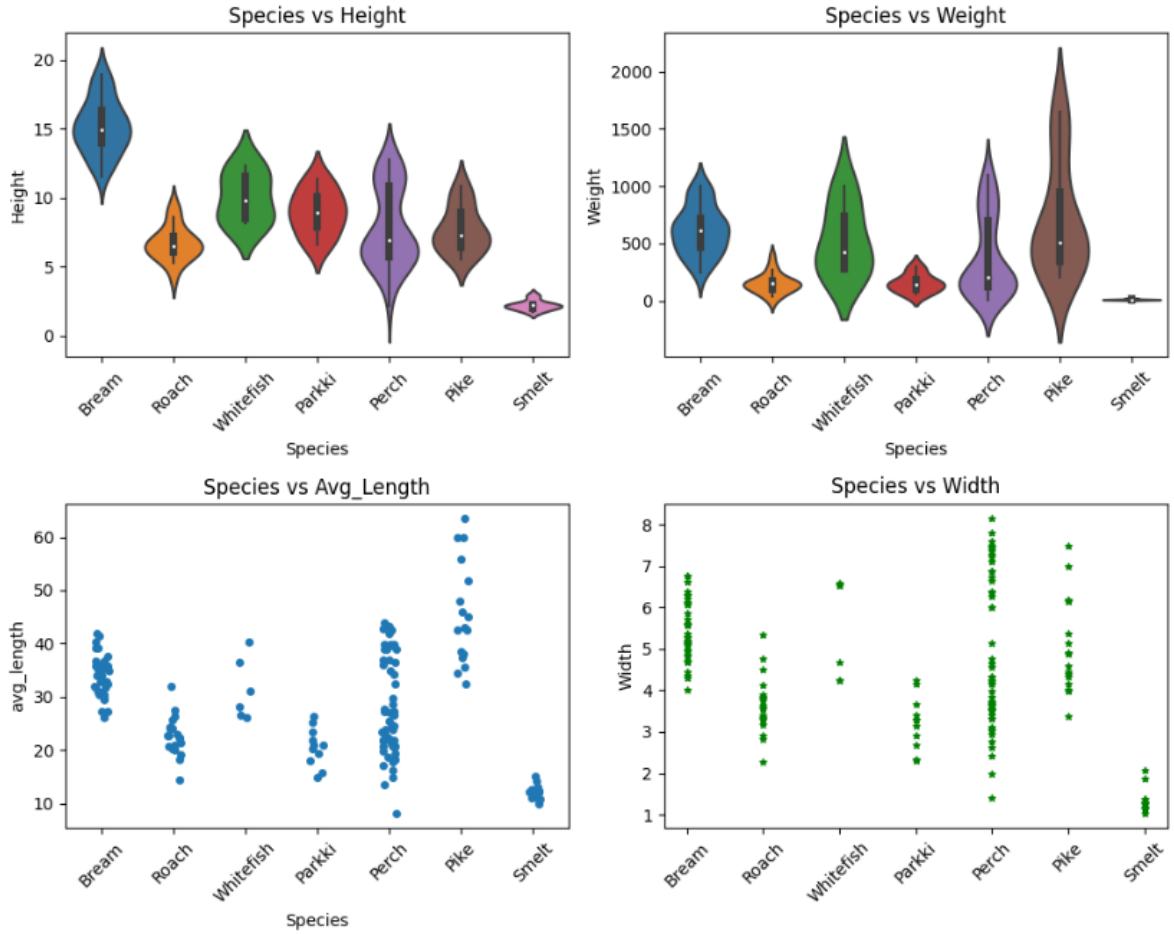
For better understanding of the data, the data will be plotted in different charts which will help to identify trends. So, first we make a copy of the original dataset and experiment on the copy. For easy visualization, the average of three lengths have been taken and that data is saved in a new column named avg_length.

```
df_fish2=df_fish.copy() # copying data to another variable to make changes and not cha
df_fish2['avg_length']=(df_fish2['Length1']+df_fish2['Length2']+df_fish2['Length3'])/3
df_fish2.head()
```

	Species	Weight	Length1	Length2	Length3	Height	Width	avg_length
0	Bream	242.0	23.2	25.4	30.0	11.5200	4.0200	26.200000
1	Bream	290.0	24.0	26.3	31.2	12.4800	4.3056	27.166667
2	Bream	340.0	23.9	26.5	31.1	12.3778	4.6961	27.166667
3	Bream	363.0	26.3	29.0	33.5	12.7300	4.4555	29.600000
4	Bream	430.0	26.5	29.0	34.0	12.4440	5.1340	29.833333

The following plots have been used:

- sns.violinplot(x='Species', y='Height', data=df_fish, scale="width")
- sns.violinplot(x='Species', y='Height', data=df_fish, scale="width")
- sns.stripplot(x='Species', y='avg_length', data=df_fish2, jitter=True)
- plt.scatter(df_fish2['Species'],df_fish2['Width'],c='green',s=15,marker='*')



From the graphs, it is evident that Smelt species of fish has the least metrics for any feature. This could be the reason for some people not to choose this species for any use.

Similarly, the length and weight of Pike are relatively more compared to other species. Also, The Parkki species has less width and more height, that can make the fish's body more streamlined that makes them fast swimmer perhaps.

STEP 5: **Correlations-** It is a statistical measure that quantifies the degree to which two variables are related or associated with each other. Correlation coefficients range from -1 to 1. A correlation of 1 indicates a perfect positive linear relationship, where both variables move together. A correlation of 0 suggests no linear relationship.

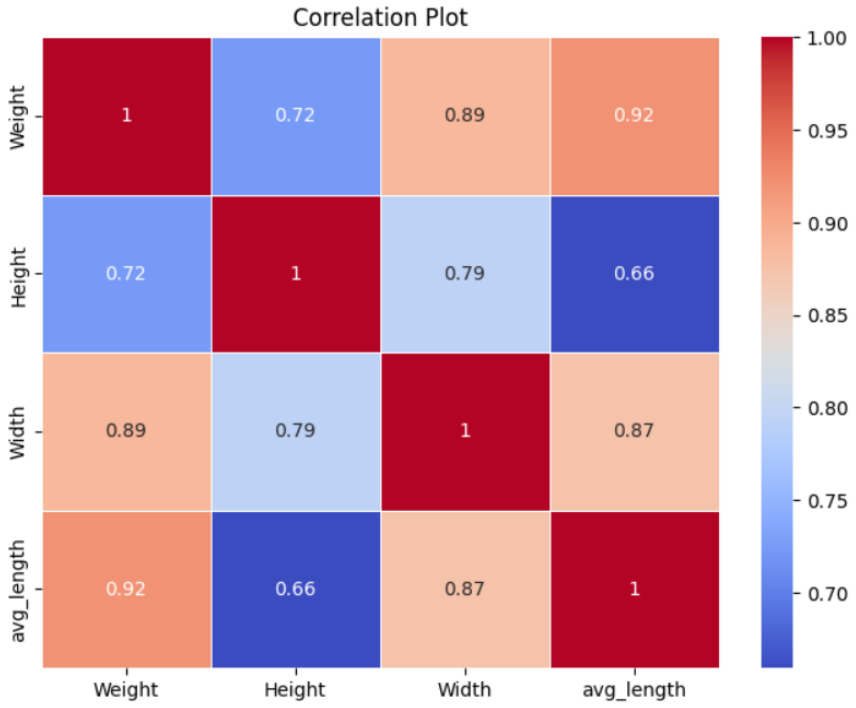
For this we will use the inbuilt Pearson correlation method.

```
df_fish3.corr(method='pearson')
```

	Weight	Height	Width	avg_length
Weight	1.000000	0.724345	0.886507	0.920817
Height	0.724345	1.000000	0.792881	0.659481
Width	0.886507	0.792881	1.000000	0.874757
avg_length	0.920817	0.659481	0.874757	1.000000

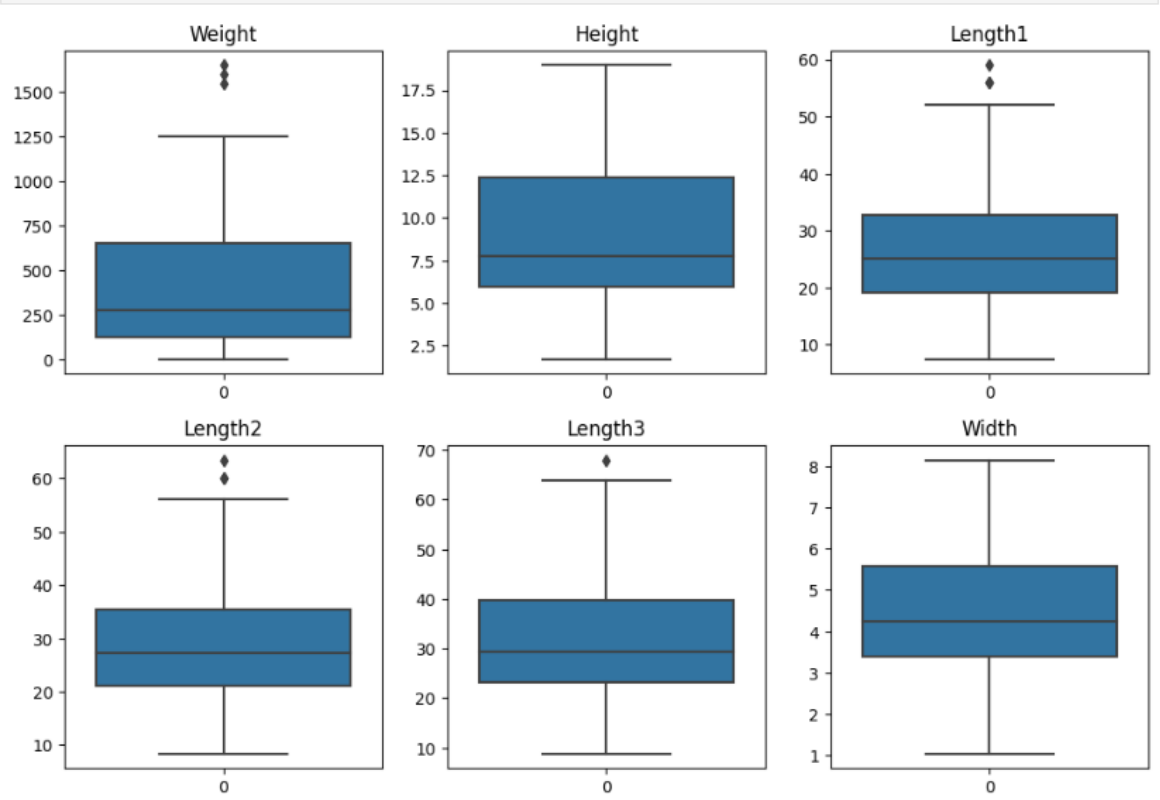
This table is not the best way to study correlation. So heat-map plot will be used to get more insights regarding correlation between height, weight, and average length of the fish.

- sns.heatmap(df_fish3.corr(method='pearson'), annot=True, cmap='coolwarm', linewidths=0.5)



From the above heatmap, we can deduce that length and weight of the fish are very much positively correlated. Meaning, if there is an increase in weight of the fish, length of the fish will also increase and vice versa. Similarly, we can see that height and length of the fish are less correlated, meaning that the probability that a fish is long and also has more height is less.

STEP 6: Finding and treating outliers. We will use boxplot to determine which columns(features) have outliers and treat these outliers.



From the plot, we find out weight, length 1, length 2 and length 3 have outliers.

Finding the outliers in the data- IQR METHOD:

In this method, we will find the first (Q1) and third (Q3) quartiles of each feature and find the IQR using the formula IQR= Q3-Q1. Then we set the upper and lower limits of the values that will help in filtering data.

```
# Weight column
q1_W= df_fish['Weight'].quantile(0.25) #First quartile
q3_W=df_fish['Weight'].quantile(0.75)# 3rd quartile
iqr_W=q3_W - q1_W   # finding IQR
upper_W= q3_W + (1.5*iqr_W) #upper limit formula
lower_W= q1_W- (1.5*iqr_W) #lower limit formula
```

Next, we will use capping and filtering function using loc (label-based indexing).

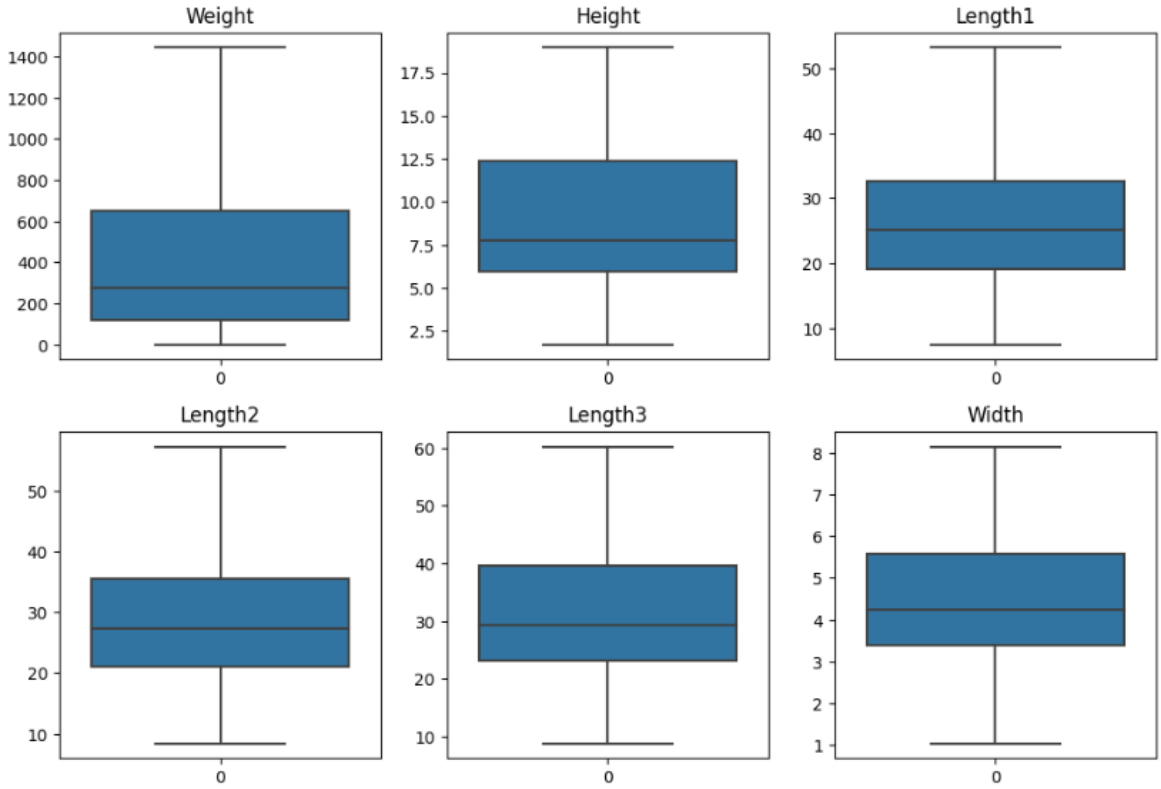
```
#capping and Filterting data
df_fish4.loc[(df_fish4['Weight']> upper_W), 'Weight']=upper_W
df_fish4.loc[(df_fish4['Weight']< lower_W), 'Weight']=lower_W

df_fish4.loc[(df_fish4['Length1']> upper_l1), 'Length1']=upper_l1
df_fish4.loc[(df_fish4['Length1']< lower_l1), 'Length1']=lower_l1

df_fish4.loc[(df_fish4['Length2']> upper_l2), 'Length2']=upper_l2
df_fish4.loc[(df_fish4['Length2']< lower_l2), 'Length2']=lower_l2

df_fish4.loc[(df_fish4['Length3']> upper_l3), 'Length3']=upper_l3
df_fish4.loc[(df_fish4['Length3']< lower_l3), 'Length3']=lower_l3
```

Then we plot the same data again to view updated box plots where outliers are removed.



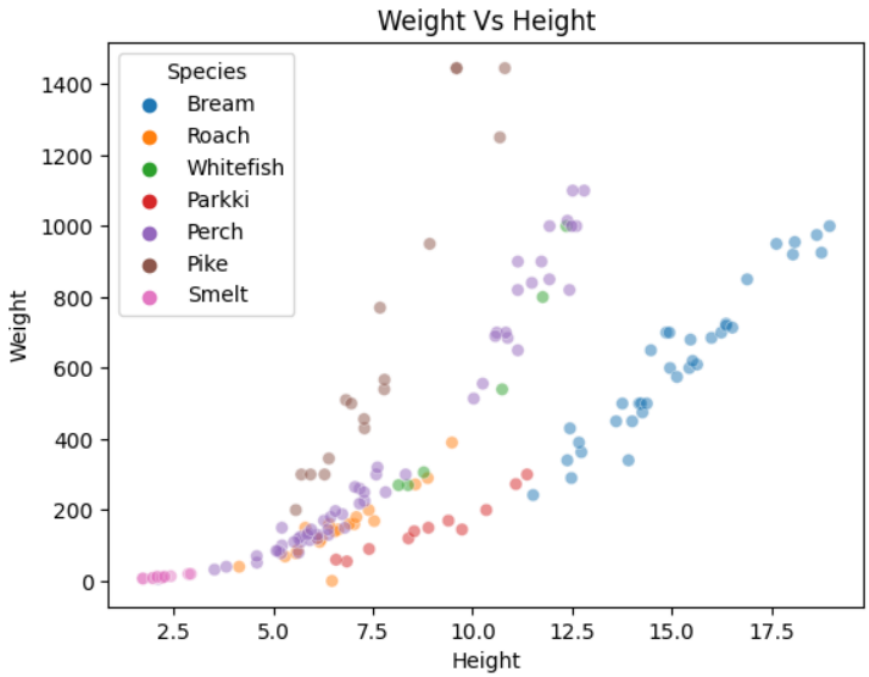
In the above plot we observe that all the outliers have been removed. Outliers can have a significant impact on the performance of statistical models. If a model is trained on this data, by removing outliers the model can be trained on a cleaner, more representative subset of the data, leading to better predictive accuracy and generalization. In situations where outliers are the result of data entry errors or measurement errors, removing them can help maintain the integrity and quality of the dataset. Also, removing outliers can reduce the risk of overfitting and improve a model's ability to generalize to new data.

STEP 7: EDA

The following scatter plot is plotted with a weight vs height scale, and it is grouped by species. This is done using the seaborn scatter plot with a hue attribute.

```
#Grouping the weight vs height data based on species
sns.scatterplot(x='Height',y='Weight',data=df_fish4, hue='Species', alpha=0.5).set(title='Weight Vs Height')
```

```
[Text(0.5, 1.0, 'Weight Vs Height')]
```



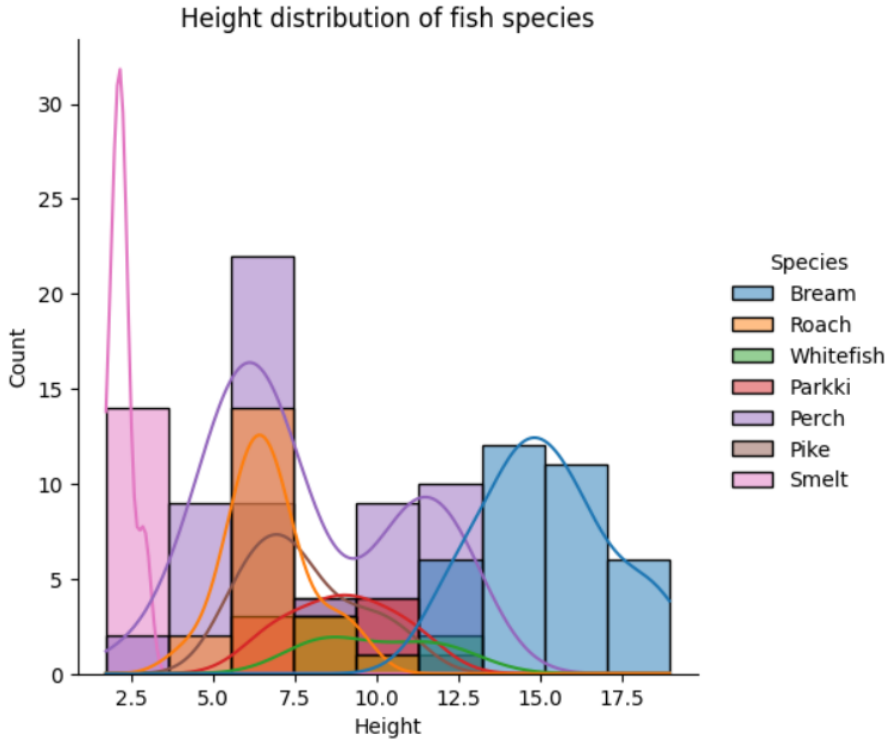
From the above plot, we can conclude that Smelt, Roach and most Perch fish have low body weight and less height since they are concentrated near the origin of the plot. The least number of data points are present for the white fish species that are represented as green markers.

Next, a distribution plot is used to study the height characteristics of different species. This is used to lot univariate or bivariate distributions using kernel density estimation. The KDE is a smoothed, continuous estimate of the probability density function (PDF) of the data. s

```
#distribution plot using kernel densitity estimatin(kde)
sns.displot(data=df_fish4, x="Height", kde=True, hue='Species').set(title='Height distribution of fish species')
```

```
C:\Users\Chirag\AppData\Local\Programs\Python\Python311\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning:
  self._figure.tight_layout(*args, **kwargs)
```

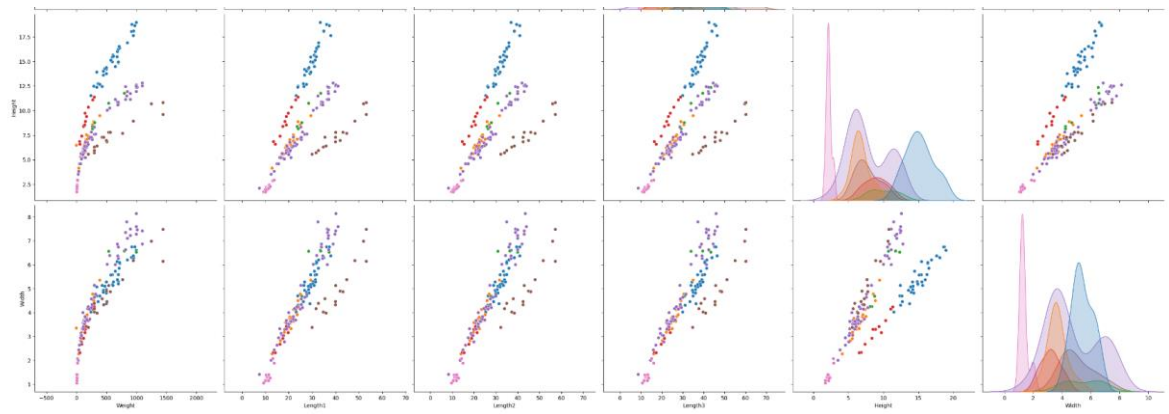
```
<seaborn.axisgrid.FacetGrid at 0x295c7738f50>
```



The plot shows the distribution of various species in the dataset. We can see 4 species have the height in the range of 5.0 – 7.5 units (mostly in inches) and the least fish count in for height between 7.5 – 10 units. The large spike of pink line i.e., the smelt species is because it has the highest number of fish for the 2.5 units height range among other species.

When we need to check plots for various combinations of the features and not sure which will yield best data readability, the pair plot is used.

```
# Pair plot
sns.pairplot(df_fish4, hue='Species', height=5)
```

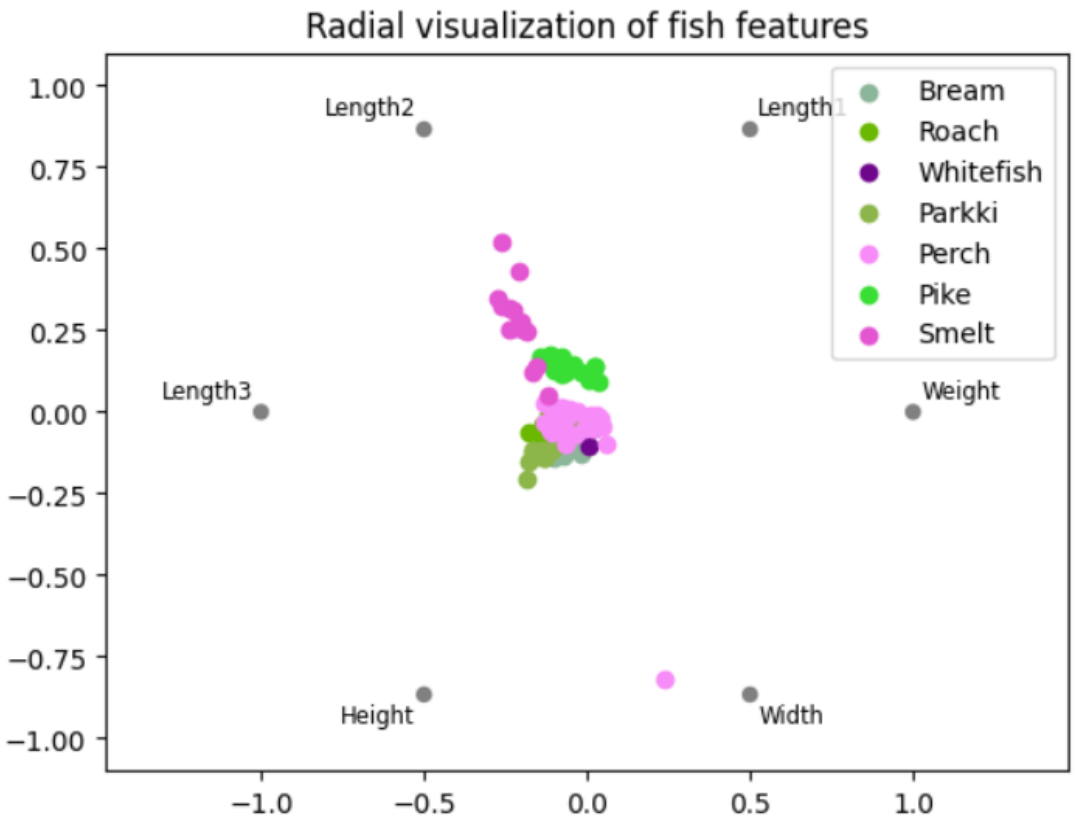


The above is a part of the pair plot shown for reference. It compares all features with one another while it's grouped by species. Pair plots are used to understand the best set of features to explain a relationship between two variables and identify trends for follow-up analysis.

Radial visualization plot: Radviz helps users recognize patterns and trends in multivariate data by showing how data points cluster around different attribute axes. It can reveal relationships, correlations, and groupings among variables. It's important to note that it may not be suitable for all types of datasets, especially those with a large number of variables.

```
# Radial Visualization
import pandas.plotting as pdplt
pdplt.radviz(df_fish4, 'Species').set(title='Radial visualization of fish features')
```

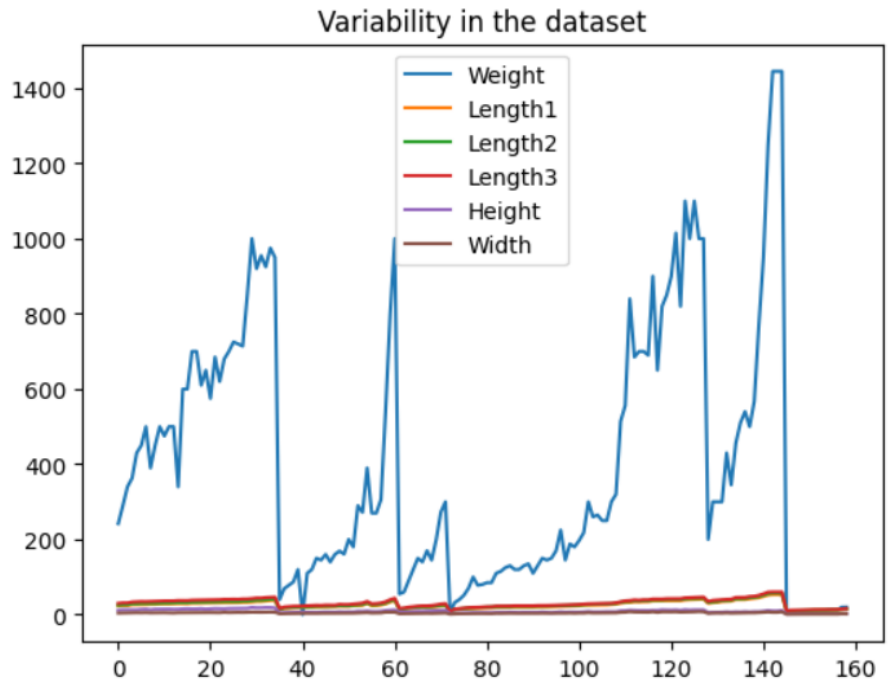
```
[Text(0.5, 1.0, 'Radial visualization of fish features')]
```



To check variability in data, we use the .plot() function. From the plot we see that all other features except weight are somewhat similar. This might be because there are no units of measurement given in the dataset.

```
df_fish4.plot().set(title='Variability in the dataset')
```

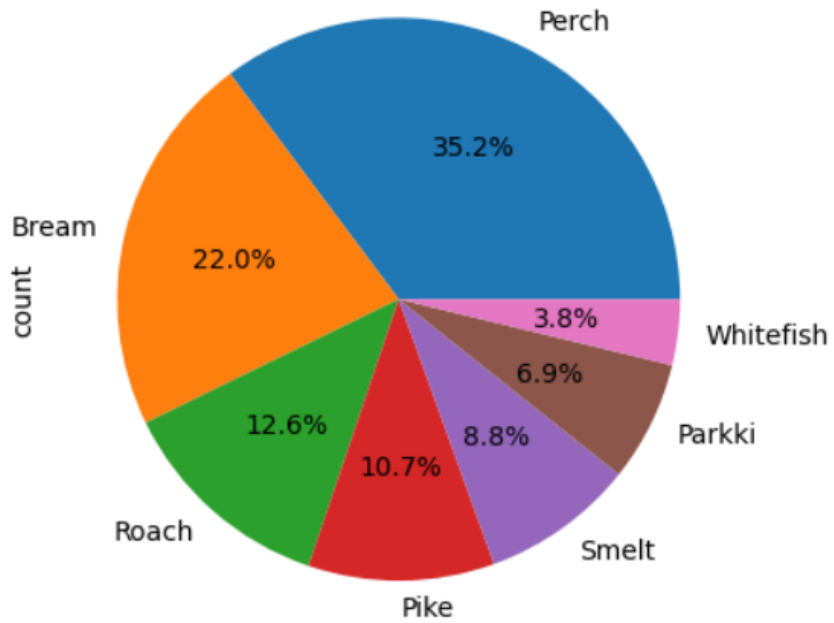
```
[Text(0.5, 1.0, 'Variability in the dataset')]
```



The pie chart below can be used to determine the percentage of fish species in the dataset. We can see most data is of Perch fish having 35.2% of the total sample and least is whitefish with 3.8%.

```
# percentage of data set
import plotly.express as px
df_fish4['Species'].value_counts().plot.pie(autopct='%1.1f%%' )
```

```
<Axes: ylabel='count'>
```



RESOURCES

- Pandas: https://pandas.pydata.org/docs/reference/index.html
- Numpy: https://numpy.org/doc/stable/reference/routines.html
-
- Matplotlib: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html
- Seaborn: https://seaborn.pydata.org/generated/seaborn.displot.html
- https://www.kdnuggets.com/2022/06/plotting-data-visualization-data-science.html#:~:text=Geometric%20Component%3AHere%20is%20where,heatmaps%2C%20pie%20charts%2C%20etc.
- https://chartio.com/learn/charts/essential-chart-types-for-data-visualization/