In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Part1: We will be doing Cleaning and wrangling the datas. The dataset consist of information of different locations traffic cameras. The dataset is imported from google drive and mounted for the data analysis process.

Read the libraries by importing it

In [ ]:

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

Reading the CSV file and printing top 5 rows of the dataset

Code imports necessary libraries, reads a CSV file containing traffic camera data into a Pandas DataFrame, and then displays the top 5 rows of the DataFrame to examine the data's structure. This is typically one of the initial steps in data analysis and data wrangling to understand the dataset before making any necessary transformations or cleaning.

In [ ]:

```
# Reading the CSV file
path="/content/drive/MyDrive/DATASET/traffic_cameras.csv"
df = pd.read_csv(path)

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

Out[ ]:

	Camera ID	Location Name	Camera Status	Turn on Date	Camera Manufacturer	ATD Location ID	Landmark	Signal Engineer Area
0	370	PLEASANT VALLEY RD / NUCKOLS CROSSING RD	TURNED_ON	5/24/2018	Advidia	LOC16-003180	NaN	SOUTHEAST
1	379	BARTON SPRINGS RD / KINNEY AVE	TURNED_ON	5/21/2018	Advidia	LOC16-000640	NaN	SOUTHWEST
2	404	SPRINGDALE RD / OAK SPRINGS DR	TURNED_ON	6/7/2018	Advidia	LOC16-000800	NaN	NORTHEAST
3	447	BRAKER LN / STONELAKE BLVD	TURNED_ON	9/9/2016	Advidia	LOC16-003740	NaN	NORTHWEST
4	552	EXPOSITION BLVD / WESTOVER RD	TURNED_ON	2/24/2020	Advidia	LOC16-003710	NaN	CENTRAL

5 rows × 28 columns

1. How many rows and columns does your data have?

Found the dimensions of the DataFrame which are are columns and rows using axes[ ] function we will 802 rows and 28 columns

In [ ]:

```
### Your code goes here ###
row = len(df.axes[0])
col = len(df.axes[1])
print("Number of Rows: ", row)
print("Number of Columns: ", col)
```

```
Number of Rows:  802
Number of Columns:  28
```

2. What can you tell us about the type of variables we have?

int64, float64, object are the three datatypes that are present in the traffic dataset which correspond to integer, floating-point, or string (object) data types

In [ ]:

```
### Your code goes here ###
var_type = df.dtypes
print(var_type)
```

```
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. Delete only the columns that have all null values, name it df1 (nothing else, but null)

The code creates a new DataFrame df1 where columns that contain only NaN values have been removed. The resulting DataFrame df1 will contain only the columns that have at least one non-NaN value. Deleted the columns using dropna function

In [ ]:

```
### Your code goes here ###
df1 = df.dropna(axis=1, how='all')
```

info() method is used for quickly understanding the structure of your DataFrame, including the number of rows, columns, data types, and memory usage

In [ ]:

```
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
```

4. Dropp columns that have (any) null values name it df2

The code creates a new DataFrame df2 where columns that contain any NaN values have been removed. Dropped columns using dropna function.

In [ ]:

```
### Your code goes here ###
df2 = df.dropna(axis=1, how='any')
```

shape provides the with essential information about the structure of the DataFrame df2, including the number of rows, columns, data types, and memory usage, used shape function to access it

In [ ]:

```
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: 69.0+ KB
```

Out[ ]:

```
(802, 11)
```

5. Rename column names in df2 so they are more usable (name the new dataframe df3) to the followings: cam_id, loc_name, cam_stat, atd_loc_id, loc_type, date, comm_stat, comm_stat_date, screen_addr, id, location

Columns are renamed using rename function 'Camera ID' is renamed to 'cam_id'.'Location Name' is renamed to 'loc_name'.'Camera Status' is renamed to 'cam_stat'.'ATD Location ID' is renamed to 'atd_loc_id'.'Location Type' is renamed to 'loc_type'.'Modified Date' is renamed to 'date'.'IP Comm Status' is renamed to 'comm_stat'.'IP Comm Status Date and Time' is renamed to 'comm_stat_date'.'Screenshot Address' is renamed to 'screen_addr'.'ID' is renamed to 'id'.'Location' is renamed to 'location'.

In [ ]:

```python
### Your code goes here ###
df3 = df2.rename(columns={
    'Camera ID': 'cam_id',
    'Location Name': 'loc_name',
    'Camera Status': 'cam_stat',
    'ATD Location ID': 'atd_loc_id',
    'Location Type': 'loc_type',
    'Modified Date': 'date',
    'IP Comm Status': 'comm_stat',
    'IP Comm Status Date and Time': 'comm_stat_date',
    'Screenshot Address': 'screen_addr',
    'ID': 'id',
    'Location': 'location',
})
```

Gives top 5 rows of the dataset after renaming using head method

```
In [ ]: df3.head
```

Out[ ]:

```
<bound method NDFrame.head of      cam_id
loc_name   cam_stat  \
0       370           PLEASANT VALLEY RD / NUCKOLS CROSSING RD  TURNED_ON
1       379                     BARTON SPRINGS RD / KINNEY AVE  TURNED_ON
2       404                     SPRINGDALE RD / OAK SPRINGS DR  TURNED_ON
3       447                         BRAKER LN / STONELAKE BLVD  TURNED_ON
4       552                      EXPOSITION BLVD / WESTOVER RD  TURNED_ON
..      ...                                                ...        ...
797    1190                             GUADALUPE ST / 46TH ST  TURNED_ON
798    1274                    CESAR CHAVEZ ST / SAN MARCOS ST  TURNED_ON
799    1275                     BURNET RD / RESEARCH BLVD SVRD  TURNED_ON
800    1276   BASTROP HWY / MONTOPOLIS TO BASTROP NB RAMP (...  TURNED_ON
801    1277                       BURNET RD / BRIGHT VERDE WAY  TURNED_ON

       atd_loc_id loc_type                          date comm_stat  \
0    LOC16-003180  ROADWAY  10/28/2021 08:40:00 AM +0000    ONLINE
1    LOC16-000640  ROADWAY  10/29/2021 08:45:00 AM +0000    ONLINE
2    LOC16-000800  ROADWAY  10/29/2021 07:38:00 PM +0000    ONLINE
3    LOC16-003740  ROADWAY  10/29/2021 07:49:00 PM +0000    ONLINE
4    LOC16-003710  ROADWAY  10/29/2021 07:47:00 PM +0000    ONLINE
..            ...      ...                           ...       ...
797  LOC16-006535  ROADWAY  09/19/2021 06:17:00 PM +0000    ONLINE
798  LOC16-005790  ROADWAY  09/19/2021 06:17:00 PM +0000   OFFLINE
799  LOC16-003045  ROADWAY  09/19/2021 06:17:00 PM +0000    ONLINE
800  LOC21-017685  ROADWAY  10/31/2021 08:40:00 AM +0000    ONLINE
801  LOC20-017295  ROADWAY  09/19/2021 06:17:00 PM +0000    ONLINE

                   comm_stat_date  \
0    10/28/2021 08:30:00 AM +0000
1    10/29/2021 08:35:00 AM +0000
2    10/28/2021 08:35:00 AM +0000
3    10/23/2021 08:35:00 AM +0000
4    10/20/2021 08:35:00 AM +0000
..                            ...
797  06/08/2021 08:30:00 AM +0000
798  05/12/2021 08:30:00 AM +0000
799  03/13/2021 09:35:00 AM +0000
800  10/31/2021 08:30:00 AM +0000
801  06/17/2021 08:30:00 AM +0000

                                       screen_addr                        id  \
0     https://cctv.austinmobility.io/image/370.jpg  591a10a020eacf2d16669b94
1     https://cctv.austinmobility.io/image/379.jpg  591a10a020eacf2d16669ba6
2     https://cctv.austinmobility.io/image/404.jpg  591a10a120eacf2d16669bd8
3     https://cctv.austinmobility.io/image/447.jpg  591a10a320eacf2d16669c2e
4     https://cctv.austinmobility.io/image/552.jpg  5aa6bb0121cbcf4b8b767294
..                                             ...                       ...
797  https://cctv.austinmobility.io/image/1190.jpg  5f8da8c34e4035067602f80c
798  https://cctv.austinmobility.io/image/1274.jpg  5fa580608c53d7001593adf2
799  https://cctv.austinmobility.io/image/1275.jpg  604b73eed89027001b43b7ca
800  https://cctv.austinmobility.io/image/1276.jpg  60709789cd04d0001b7605d1
801  https://cctv.austinmobility.io/image/1277.jpg  60ca5d216ce423001e32595e

                           location
0    POINT (-97.7449036 30.1844883)
1      POINT (-97.761467 30.261982)
2    POINT (-97.6904221 30.2735615)
3    POINT (-97.7392883 30.3989582)
4    POINT (-97.7643051 30.3030338)
..                              ...
797     POINT (-97.73252 30.313028)
798  POINT (-97.7346111 30.2605892)
799  POINT (-97.7264786 30.3728848)
800  POINT (-97.6902124 30.2427789)
```

```
801        POINT (-97.7228 30.3883)

[802 rows x 11 columns]>
```

6. Split "date" column into two new columns within df3 ('Dates' and 'Time') /modify df3 data/

The provided code is used to split a column named 'date' in a Pandas DataFrame df3 into two separate columns named 'Dates' and 'Time' using split function

In [ ]:

```
### Your code goes here ###
df3[['Dates', 'Time']] = df3['date'].str.split(' ', 1, expand=True)
df3.drop(columns=['date'], inplace=True)
```

```
<ipython-input-37-0168beb7e045>:2: FutureWarning: In a future version of pandas al
l arguments of StringMethods.split except for the argument 'pat' will be keyword-o
nly.
  df3[['Dates', 'Time']] = df3['date'].str.split(' ', 1, expand=True)
```

In [ ]:

```
df3.head
```

Out[ ]:

```
<bound method NDFrame.head of      cam_id
loc_name   cam_stat  \
0      370          PLEASANT VALLEY RD / NUCKOLS CROSSING RD  TURNED_ON
1      379                    BARTON SPRINGS RD / KINNEY AVE  TURNED_ON
2      404                    SPRINGDALE RD / OAK SPRINGS DR  TURNED_ON
3      447                        BRAKER LN / STONELAKE BLVD  TURNED_ON
4      552                     EXPOSITION BLVD / WESTOVER RD  TURNED_ON
..     ...                                               ...        ...
797   1190                            GUADALUPE ST / 46TH ST  TURNED_ON
798   1274                   CESAR CHAVEZ ST / SAN MARCOS ST  TURNED_ON
799   1275                    BURNET RD / RESEARCH BLVD SVRD  TURNED_ON
800   1276   BASTROP HWY / MONTOPOLIS TO BASTROP NB RAMP (...  TURNED_ON
801   1277                      BURNET RD / BRIGHT VERDE WAY  TURNED_ON

       atd_loc_id loc_type comm_stat                comm_stat_date  \
0    LOC16-003180  ROADWAY    ONLINE  10/28/2021 08:30:00 AM +0000
1    LOC16-000640  ROADWAY    ONLINE  10/29/2021 08:35:00 AM +0000
2    LOC16-000800  ROADWAY    ONLINE  10/28/2021 08:35:00 AM +0000
3    LOC16-003740  ROADWAY    ONLINE  10/23/2021 08:35:00 AM +0000
4    LOC16-003710  ROADWAY    ONLINE  10/20/2021 08:35:00 AM +0000
..            ...      ...       ...                           ...
797  LOC16-006535  ROADWAY    ONLINE  06/08/2021 08:30:00 AM +0000
798  LOC16-005790  ROADWAY   OFFLINE  05/12/2021 08:30:00 AM +0000
799  LOC16-003045  ROADWAY    ONLINE  03/13/2021 09:35:00 AM +0000
800  LOC21-017685  ROADWAY    ONLINE  10/31/2021 08:30:00 AM +0000
801  LOC20-017295  ROADWAY    ONLINE  06/17/2021 08:30:00 AM +0000

                                     screen_addr                        id  \
0     https://cctv.austinmobility.io/image/370.jpg  591a10a020eacf2d16669b94
1     https://cctv.austinmobility.io/image/379.jpg  591a10a020eacf2d16669ba6
2     https://cctv.austinmobility.io/image/404.jpg  591a10a120eacf2d16669bd8
3     https://cctv.austinmobility.io/image/447.jpg  591a10a320eacf2d16669c2e
4     https://cctv.austinmobility.io/image/552.jpg  5aa6bb0121cbcf4b8b767294
..                                             ...                       ...
797  https://cctv.austinmobility.io/image/1190.jpg  5f8da8c34e4035067602f80c
798  https://cctv.austinmobility.io/image/1274.jpg  5fa580608c53d7001593adf2
799  https://cctv.austinmobility.io/image/1275.jpg  604b73eed89027001b43b7ca
800  https://cctv.austinmobility.io/image/1276.jpg  60709789cd04d0001b7605d1
801  https://cctv.austinmobility.io/image/1277.jpg  60ca5d216ce423001e32595e

                           location       Dates                Time
0    POINT (-97.7449036 30.1844883)  10/28/2021  08:40:00 AM +0000
1      POINT (-97.761467 30.261982)  10/29/2021  08:45:00 AM +0000
2    POINT (-97.6904221 30.2735615)  10/29/2021  07:38:00 PM +0000
3    POINT (-97.7392883 30.3989582)  10/29/2021  07:49:00 PM +0000
4    POINT (-97.7643051 30.3030338)  10/29/2021  07:47:00 PM +0000
..                              ...         ...                ...
797     POINT (-97.73252 30.313028)  09/19/2021  06:17:00 PM +0000
798  POINT (-97.7346111 30.2605892)  09/19/2021  06:17:00 PM +0000
799  POINT (-97.7264786 30.3728848)  09/19/2021  06:17:00 PM +0000
800  POINT (-97.6902124 30.2427789)  10/31/2021  08:40:00 AM +0000
801        POINT (-97.7228 30.3883)  09/19/2021  06:17:00 PM +0000

[802 rows x 12 columns]>
```

7. Split atd_loc into two new columns 'Loc' and 'code' within df3

Splitted a column named 'atd_loc_id' using split function in a DataFrame df3 into two separate columns named 'Loc' and 'code,' and then assign

the split values to these new columns using split function

In [ ]:

```
### Your code goes here ###
df3[['Loc', 'code']] = df3['atd_loc_id'].str.split('-', 1, expand=True)
df3.drop(columns=['atd_loc_id'], inplace=True)
```

```
<ipython-input-39-3980546a788d>:2: FutureWarning: In a future version of pandas al
l arguments of StringMethods.split except for the argument 'pat' will be keyword-o
nly.
  df3[['Loc', 'code']] = df3['atd_loc_id'].str.split('-', 1, expand=True)
```

In [ ]:

```
df3.head
```

Out[ ]:

```
<bound method NDFrame.head of      cam_id  
loc_name   cam_stat  \
0       370           PLEASANT VALLEY RD / NUCKOLS CROSSING RD  TURNED_ON
1       379                     BARTON SPRINGS RD / KINNEY AVE  TURNED_ON
2       404                     SPRINGDALE RD / OAK SPRINGS DR  TURNED_ON
3       447                         BRAKER LN / STONELAKE BLVD  TURNED_ON
4       552                      EXPOSITION BLVD / WESTOVER RD  TURNED_ON
..      ...                                                ...        ...
797    1190                             GUADALUPE ST / 46TH ST  TURNED_ON
798    1274                    CESAR CHAVEZ ST / SAN MARCOS ST  TURNED_ON
799    1275                     BURNET RD / RESEARCH BLVD SVRD  TURNED_ON
800    1276   BASTROP HWY / MONTOPOLIS TO BASTROP NB RAMP (...  TURNED_ON
801    1277                       BURNET RD / BRIGHT VERDE WAY  TURNED_ON

    loc_type comm_stat                comm_stat_date  \
0    ROADWAY    ONLINE  10/28/2021 08:30:00 AM +0000   
1    ROADWAY    ONLINE  10/29/2021 08:35:00 AM +0000   
2    ROADWAY    ONLINE  10/28/2021 08:35:00 AM +0000   
3    ROADWAY    ONLINE  10/23/2021 08:35:00 AM +0000   
4    ROADWAY    ONLINE  10/20/2021 08:35:00 AM +0000   
..       ...       ...                           ...   
797  ROADWAY    ONLINE  06/08/2021 08:30:00 AM +0000   
798  ROADWAY   OFFLINE  05/12/2021 08:30:00 AM +0000   
799  ROADWAY    ONLINE  03/13/2021 09:35:00 AM +0000   
800  ROADWAY    ONLINE  10/31/2021 08:30:00 AM +0000   
801  ROADWAY    ONLINE  06/17/2021 08:30:00 AM +0000   

                                       screen_addr                        id  \
0     https://cctv.austinmobility.io/image/370.jpg  591a10a020eacf2d16669b94   
1     https://cctv.austinmobility.io/image/379.jpg  591a10a020eacf2d16669ba6   
2     https://cctv.austinmobility.io/image/404.jpg  591a10a120eacf2d16669bd8   
3     https://cctv.austinmobility.io/image/447.jpg  591a10a320eacf2d16669c2e   
4     https://cctv.austinmobility.io/image/552.jpg  5aa6bb0121cbcf4b8b767294   
..                                             ...                       ...   
797  https://cctv.austinmobility.io/image/1190.jpg  5f8da8c34e4035067602f80c   
798  https://cctv.austinmobility.io/image/1274.jpg  5fa580608c53d7001593adf2   
799  https://cctv.austinmobility.io/image/1275.jpg  604b73eed89027001b43b7ca   
800  https://cctv.austinmobility.io/image/1276.jpg  60709789cd04d0001b7605d1   
801  https://cctv.austinmobility.io/image/1277.jpg  60ca5d216ce423001e32595e   

                           location       Dates               Time    Loc  \
0    POINT (-97.7449036 30.1844883)  10/28/2021  08:40:00 AM +0000  LOC16   
1      POINT (-97.761467 30.261982)  10/29/2021  08:45:00 AM +0000  LOC16   
2    POINT (-97.6904221 30.2735615)  10/29/2021  07:38:00 PM +0000  LOC16   
3    POINT (-97.7392883 30.3989582)  10/29/2021  07:49:00 PM +0000  LOC16   
4    POINT (-97.7643051 30.3030338)  10/29/2021  07:47:00 PM +0000  LOC16   
..                              ...         ...                ...    ...   
797     POINT (-97.73252 30.313028)  09/19/2021  06:17:00 PM +0000  LOC16   
798  POINT (-97.7346111 30.2605892)  09/19/2021  06:17:00 PM +0000  LOC16   
799  POINT (-97.7264786 30.3728848)  09/19/2021  06:17:00 PM +0000  LOC16   
800  POINT (-97.6902124 30.2427789)  10/31/2021  08:40:00 AM +0000  LOC21   
801        POINT (-97.7228 30.3883)  09/19/2021  06:17:00 PM +0000  LOC20   

       code  
0    003180  
1    000640  
2    000800  
3    003740  
4    003710  
..      ...  
797  006535  
798  005790  
799  003045  
800  017685  
```

```
801  017295

[802 rows x 13 columns]>
```

8. What are the unique values in loc_type?

To display the distinct categories or values that exist in the 'loc_type' column of the DataFrame using unique() where unique values roadway and building will be printed.

In [ ]:

```
### Your code goes here ###
unique_loc_types = df3['loc_type'].unique()
print(unique_loc_types)
```

```
['ROADWAY' 'BUILDING']
```

9. Replace 'ROADWAY' to '0', 'BUILDING' to '1' in the loc_type column within df3

'loc_type' column in df3 will have the specified replacements. if the original 'loc_type' column had values like 'ROADWAY', they would be replaced with '0', and if it had values like 'BUILDING', they would be replaced with '1' using replace()

In [ ]:

```
### Your code goes here ###
df3['loc_type'].replace({'ROADWAY': '0', 'BUILDING': '1'}, inplace=True)
```

In [ ]:

```
df3.head
```

Out[ ]:

```
<bound method NDFrame.head of      cam_id
loc_name   cam_stat  \
0       370           PLEASANT VALLEY RD / NUCKOLS CROSSING RD  TURNED_ON
1       379                     BARTON SPRINGS RD / KINNEY AVE  TURNED_ON
2       404                     SPRINGDALE RD / OAK SPRINGS DR  TURNED_ON
3       447                         BRAKER LN / STONELAKE BLVD  TURNED_ON
4       552                      EXPOSITION BLVD / WESTOVER RD  TURNED_ON
..      ...                                                ...        ...
797    1190                             GUADALUPE ST / 46TH ST  TURNED_ON
798    1274                    CESAR CHAVEZ ST / SAN MARCOS ST  TURNED_ON
799    1275                     BURNET RD / RESEARCH BLVD SVRD  TURNED_ON
800    1276   BASTROP HWY / MONTOPOLIS TO BASTROP NB RAMP (...  TURNED_ON
801    1277                       BURNET RD / BRIGHT VERDE WAY  TURNED_ON

    loc_type comm_stat                comm_stat_date  \
0          0    ONLINE  10/28/2021 08:30:00 AM +0000
1          0    ONLINE  10/29/2021 08:35:00 AM +0000
2          0    ONLINE  10/28/2021 08:35:00 AM +0000
3          0    ONLINE  10/23/2021 08:35:00 AM +0000
4          0    ONLINE  10/20/2021 08:35:00 AM +0000
..       ...       ...                           ...
797        0    ONLINE  06/08/2021 08:30:00 AM +0000
798        0   OFFLINE  05/12/2021 08:30:00 AM +0000
799        0    ONLINE  03/13/2021 09:35:00 AM +0000
800        0    ONLINE  10/31/2021 08:30:00 AM +0000
801        0    ONLINE  06/17/2021 08:30:00 AM +0000

                                       screen_addr                        id  \
0     https://cctv.austinmobility.io/image/370.jpg  591a10a020eacf2d16669b94
1     https://cctv.austinmobility.io/image/379.jpg  591a10a020eacf2d16669ba6
2     https://cctv.austinmobility.io/image/404.jpg  591a10a120eacf2d16669bd8
3     https://cctv.austinmobility.io/image/447.jpg  591a10a320eacf2d16669c2e
4     https://cctv.austinmobility.io/image/552.jpg  5aa6bb0121cbcf4b8b767294
..                                             ...                       ...
797  https://cctv.austinmobility.io/image/1190.jpg  5f8da8c34e4035067602f80c
798  https://cctv.austinmobility.io/image/1274.jpg  5fa580608c53d7001593adf2
799  https://cctv.austinmobility.io/image/1275.jpg  604b73eed89027001b43b7ca
800  https://cctv.austinmobility.io/image/1276.jpg  60709789cd04d0001b7605d1
801  https://cctv.austinmobility.io/image/1277.jpg  60ca5d216ce423001e32595e

                           location       Dates               Time    Loc  \
0    POINT (-97.7449036 30.1844883)  10/28/2021  08:40:00 AM +0000  LOC16
1      POINT (-97.761467 30.261982)  10/29/2021  08:45:00 AM +0000  LOC16
2    POINT (-97.6904221 30.2735615)  10/29/2021  07:38:00 PM +0000  LOC16
3    POINT (-97.7392883 30.3989582)  10/29/2021  07:49:00 PM +0000  LOC16
4    POINT (-97.7643051 30.3030338)  10/29/2021  07:47:00 PM +0000  LOC16
..                              ...         ...                ...    ...
797     POINT (-97.73252 30.313028)  09/19/2021  06:17:00 PM +0000  LOC16
798  POINT (-97.7346111 30.2605892)  09/19/2021  06:17:00 PM +0000  LOC16
799  POINT (-97.7264786 30.3728848)  09/19/2021  06:17:00 PM +0000  LOC16
800  POINT (-97.6902124 30.2427789)  10/31/2021  08:40:00 AM +0000  LOC21
801        POINT (-97.7228 30.3883)  09/19/2021  06:17:00 PM +0000  LOC20

       code
0    003180
1    000640
2    000800
3    003740
4    003710
..      ...
797  006535
798  005790
799  003045
800  017685
```

```
801   017295

[802 rows x 13 columns]>
```

df3.loc_type.unique(), will get an array containing these unique values, which can be useful for various data analysis and manipulation tasks, such as summarizing data, filtering data based on specific categories, or creating visualizations to explore the distribution of these categories in the dataset.

In [ ]:

```
df3.loc_type.unique()
```

Out[ ]:

```
array(['0', '1'], dtype=object)
```

10. Split on on '/' the loc_name column into two new variables 'corner1', 'corner2'

Splitted a column using split function and named 'loc_name' in a Pandas DataFrame df3 into two separate columns named 'corner1' and 'corner2,' and then assign the split values to these new columns

In [ ]:

```
### Your code goes here ###
df3[['corner1', 'corner2']] = df3['loc_name'].str.split('/', 1, expand=True)
```

```
<ipython-input-45-757d10efe195>:2: FutureWarning: In a future version of pandas al
l arguments of StringMethods.split except for the argument 'pat' will be keyword-o
nly.
  df3[['corner1', 'corner2']] = df3['loc_name'].str.split('/', 1, expand=True)
```

In [ ]:

```
df3.head
```

Out[ ]:

```
<bound method NDFrame.head of      cam_id
loc_name   cam_stat  \
0       370           PLEASANT VALLEY RD / NUCKOLS CROSSING RD  TURNED_ON
1       379                     BARTON SPRINGS RD / KINNEY AVE  TURNED_ON
2       404                     SPRINGDALE RD / OAK SPRINGS DR  TURNED_ON
3       447                         BRAKER LN / STONELAKE BLVD  TURNED_ON
4       552                      EXPOSITION BLVD / WESTOVER RD  TURNED_ON
..      ...                                                ...        ...
797    1190                             GUADALUPE ST / 46TH ST  TURNED_ON
798    1274                    CESAR CHAVEZ ST / SAN MARCOS ST  TURNED_ON
799    1275                     BURNET RD / RESEARCH BLVD SVRD  TURNED_ON
800    1276   BASTROP HWY / MONTOPOLIS TO BASTROP NB RAMP (...  TURNED_ON
801    1277                       BURNET RD / BRIGHT VERDE WAY  TURNED_ON

    loc_type comm_stat               comm_stat_date  \
0          0    ONLINE  10/28/2021 08:30:00 AM +0000
1          0    ONLINE  10/29/2021 08:35:00 AM +0000
2          0    ONLINE  10/28/2021 08:35:00 AM +0000
3          0    ONLINE  10/23/2021 08:35:00 AM +0000
4          0    ONLINE  10/20/2021 08:35:00 AM +0000
..       ...       ...                           ...
797        0    ONLINE  06/08/2021 08:30:00 AM +0000
798        0   OFFLINE  05/12/2021 08:30:00 AM +0000
799        0    ONLINE  03/13/2021 09:35:00 AM +0000
800        0    ONLINE  10/31/2021 08:30:00 AM +0000
801        0    ONLINE  06/17/2021 08:30:00 AM +0000

                                         screen_addr                        id  \
0     https://cctv.austinmobility.io/image/370.jpg  591a10a020eacf2d16669b94
1     https://cctv.austinmobility.io/image/379.jpg  591a10a020eacf2d16669ba6
2     https://cctv.austinmobility.io/image/404.jpg  591a10a120eacf2d16669bd8
3     https://cctv.austinmobility.io/image/447.jpg  591a10a320eacf2d16669c2e
4     https://cctv.austinmobility.io/image/552.jpg  5aa6bb0121cbcf4b8b767294
..                                               ...                       ...
797  https://cctv.austinmobility.io/image/1190.jpg  5f8da8c34e4035067602f80c
798  https://cctv.austinmobility.io/image/1274.jpg  5fa580608c53d7001593adf2
799  https://cctv.austinmobility.io/image/1275.jpg  604b73eed89027001b43b7ca
800  https://cctv.austinmobility.io/image/1276.jpg  60709789cd04d0001b7605d1
801  https://cctv.austinmobility.io/image/1277.jpg  60ca5d216ce423001e32595e

                           location       Dates               Time    Loc  \
0    POINT (-97.7449036 30.1844883)  10/28/2021  08:40:00 AM +0000  LOC16
1      POINT (-97.761467 30.261982)  10/29/2021  08:45:00 AM +0000  LOC16
2    POINT (-97.6904221 30.2735615)  10/29/2021  07:38:00 PM +0000  LOC16
3    POINT (-97.7392883 30.3989582)  10/29/2021  07:49:00 PM +0000  LOC16
4    POINT (-97.7643051 30.3030338)  10/29/2021  07:47:00 PM +0000  LOC16
..                              ...         ...                ...    ...
797     POINT (-97.73252 30.313028)  09/19/2021  06:17:00 PM +0000  LOC16
798  POINT (-97.7346111 30.2605892)  09/19/2021  06:17:00 PM +0000  LOC16
799  POINT (-97.7264786 30.3728848)  09/19/2021  06:17:00 PM +0000  LOC16
800  POINT (-97.6902124 30.2427789)  10/31/2021  08:40:00 AM +0000  LOC21
801        POINT (-97.7228 30.3883)  09/19/2021  06:17:00 PM +0000  LOC20

       code             corner1  \
0    003180  PLEASANT VALLEY RD
1    000640   BARTON SPRINGS RD
2    000800       SPRINGDALE RD
3    003740           BRAKER LN
4    003710     EXPOSITION BLVD
..      ...                 ...
797  006535        GUADALUPE ST
798  005790     CESAR CHAVEZ ST
799  003045           BURNET RD
800  017685         BASTROP HWY
```

```
801  017295            BURNET RD

                                              corner2
0                                 NUCKOLS CROSSING RD
1                                          KINNEY AVE
2                                      OAK SPRINGS DR
3                                      STONELAKE BLVD
4                                         WESTOVER RD
..                                                ...
797                                           46TH ST
798                                     SAN MARCOS ST
799                                RESEARCH BLVD SVRD
800   MONTOPOLIS TO BASTROP NB RAMP (US 183/Montopo...
801                                  BRIGHT VERDE WAY

[802 rows x 15 columns]>
```

Part2: Exploratory Data Analysis (EDA)

We will be doing Exploratory Data Analysis that perform initial investigations on data so as to discover patterns,to spot anomalies,to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

Steps in EDA:

1. Provide descriptions of your sample and features
2. Check for missing data
3. Identify the shape of your data
4. Identify significant correlations
5. Spot/deal with outliers in the dataset

We will be doing EDA on the dataset that consist of information of fishes that includes type of species, height, width, weight and three different lengths. The dataset is imported from google drive and mounted for the data analysis process.

Reading the libraries

imported the Pandas library for for handling structured data.

imported the NumPy library for multi-dimensional arrays and mathematical functions.

imported the Seaborn library for creating informative and attractive statistical graphics.

imported the Pyplot module from Matplotlib for creating static, animated, and interactive visualizations in Python.

imports the Axes3D class from the mpl_toolkits.mplot3d to create 3D plots or visualizations.

In [ ]:

```
# importing packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline
```

The dataset is imported from google drive and mounted for the data analysis process.

Reading the CSV file and printing top 5 rows of the dataset that gives information about the fishes with the type of species, their weight, height, width and 3 length measuremnts

```
In [ ]: # Reading the CSV file
        df_fish = pd.read_csv("/content/drive/MyDrive/DATASET/Fish.csv")

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

Out[ ]:

	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

Step 1 :Descriptions and features

printing top 5 rows of the dataset that gives information about the fishes with the type of species, their weight, height, width and 3 types of length measuremnts, we will using head function for that.

```
In [ ]: df_fish.head()
```

Out[ ]:

	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

printing last 5 rows of the dataset that gives information about the fishes with the type of species, their weight, height, width and 3 types of length measurements,we will using tail function

```
In [ ]: df_fish.tail()
```

Out[ ]:

	Species	Weight	Length1	Length2	Length3	Height	Width
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

Describe method in pandas that generates summary statistics for the numerical columns (columns containing numeric data) in the DataFrame. It does not include non-numeric (string) columns.It gives the gives the information about the count, mean ,standard deviation, minimum and maximum value along with 25th percentile value,median and 75th percentile value of each column in the dataset

In [ ]:

```
df_fish.describe()
```

Out[ ]:

	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

Gives the total number of rows and columns of the dataset using shape method.

In [ ]:

```
df_fish.shape
```

Out[ ]:

```
(159, 7)
```

Gives the information about the columns in the dataset

In [ ]:

```
df_fish.columns
```

Out[ ]:

```
Index(['Species', 'Weight', 'Length1', 'Length2', 'Length3', 'Height',
       'Width'],
      dtype='object')
```

info() is helpful for understanding the structure and characteristics of the datasets, which can be important when performing data analysis, data cleaning, or data manipulation tasks. It allows to quickly identify missing values, check data types, and assess memory usage.

In [ ]:

```
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
```

Step 2 :Checking Missing value

To quickly identify missing values in the dataset we will be using isnull() this information helps us to decide how to handle those missing values, either by imputing them with some default value or removing rows/columns with too many missing values.Here the code will give us total number of missing data by adding using a sum function.

In [ ]:

```
print("There are {} missing values in the data.".format(df_fish.isna().sum().sum())
```

```
There are 0 missing values in the data.
```

Checking whether it contains duplicate values,that is, by using unique() you get an list that contains all the unique values found in the "Species" column of the Dataset df_fish. Each value in this list represents a distinct species of fish that appears in the "Species" column.

In [ ]:

```
df_fish.Species.unique()
```

Out[ ]:

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

Checking whether the datasets are balanced are not by using a count function.

In [ ]:

```
df_fish.value_counts("Species")
```

Out[ ]:

```
Species
Perch        56
Bream        34
Pike         17
Roach        15
Smelt        12
Parkki       11
Whitefish     6
dtype: int64
```

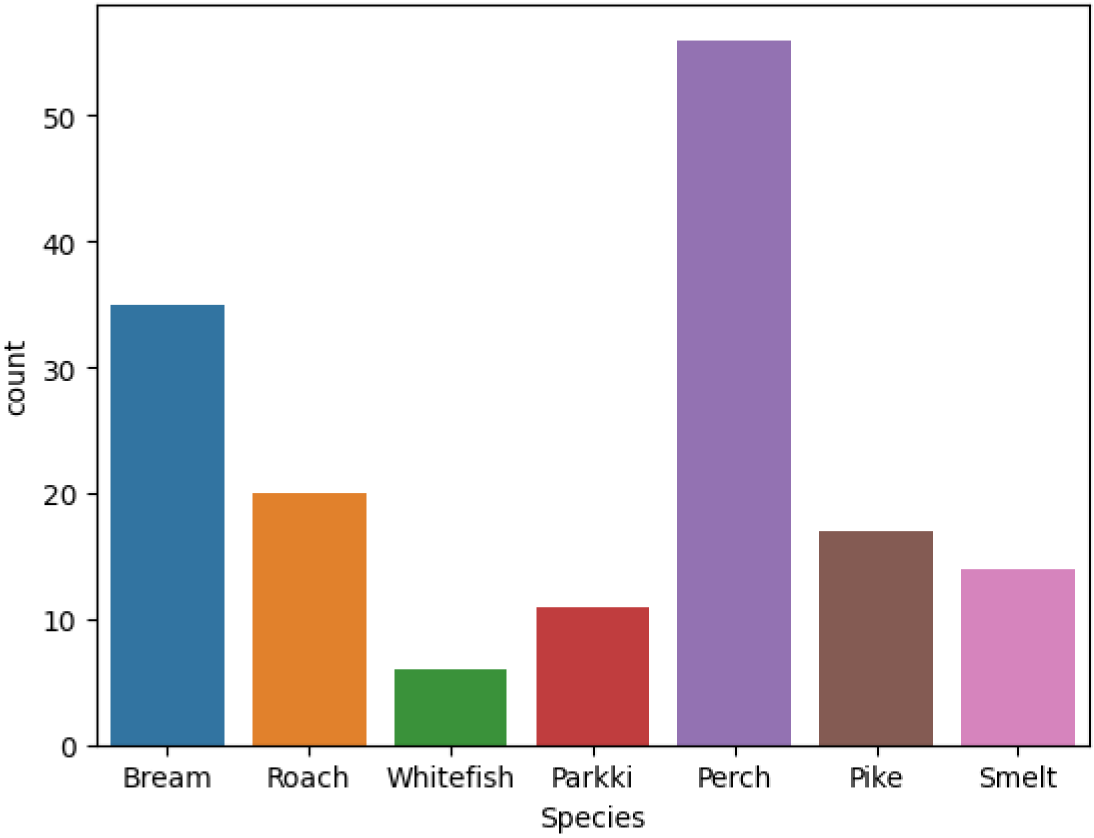
Step 3 :Checking the shape of the data

A countplot is created using species column and gives us the information which is the largest species ,that is Perch with more than 50 and the smallest number of species will be whitefish which will be less than 10

In [ ]:

```
sns.countplot(x='Species',data=df_fish)
```

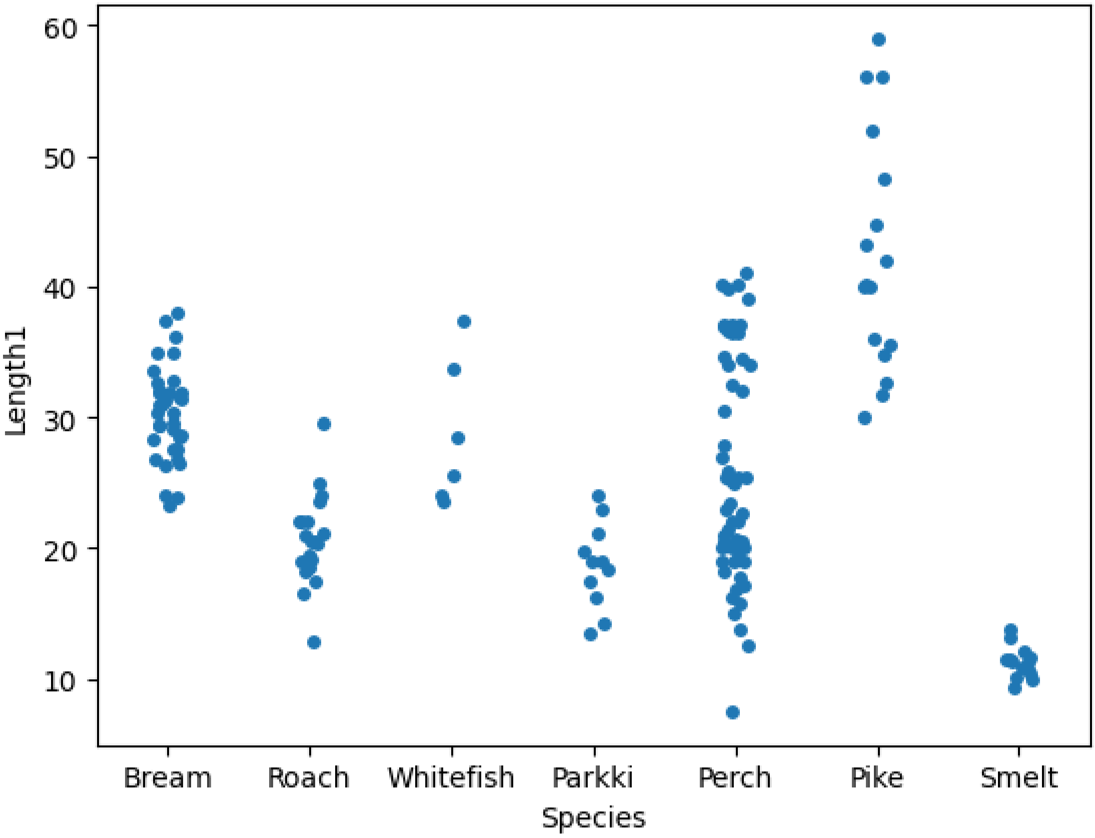
```
Out[ ]: <Axes: xlabel='Species', ylabel='count'>
```



A stripplot is created with x-axis as species and y-axis as Length1 which shows Pike will have length1 of nearly 60 as highest and Perch will be lowest that will be less than 10

```
In [ ]: sns.stripplot(y='Length1',x='Species',data=df_fish)
```

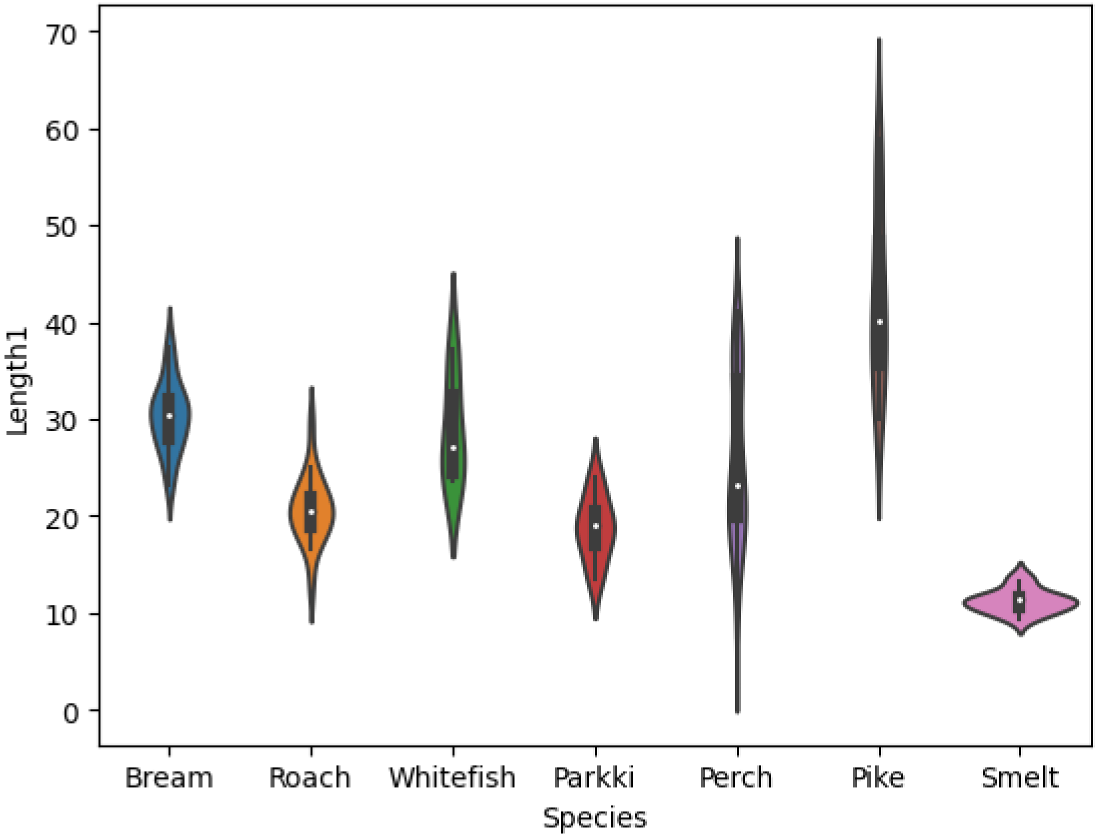
```
Out[ ]: <Axes: xlabel='Species', ylabel='Length1'>
```



A violin plot is created with x-axis as species and y-axis as Length1, we can see Length1 values vary highly with species as Perch and pike and it is less varied in Smelt species

```
In [ ]: sns.violinplot(x="Species",y="Length1",data=df_fish)
```

```
Out[ ]: <Axes: xlabel='Species', ylabel='Length1'>
```



A histogram is also plotted which gives the following info:

The highest frequency of weight is above 50 which is between 0 and 500.

The highest frequency of Length1 is nearly 28 which is in 20.

The highest frequency of Length2 is nearly 28 which is in between 20 and 40.

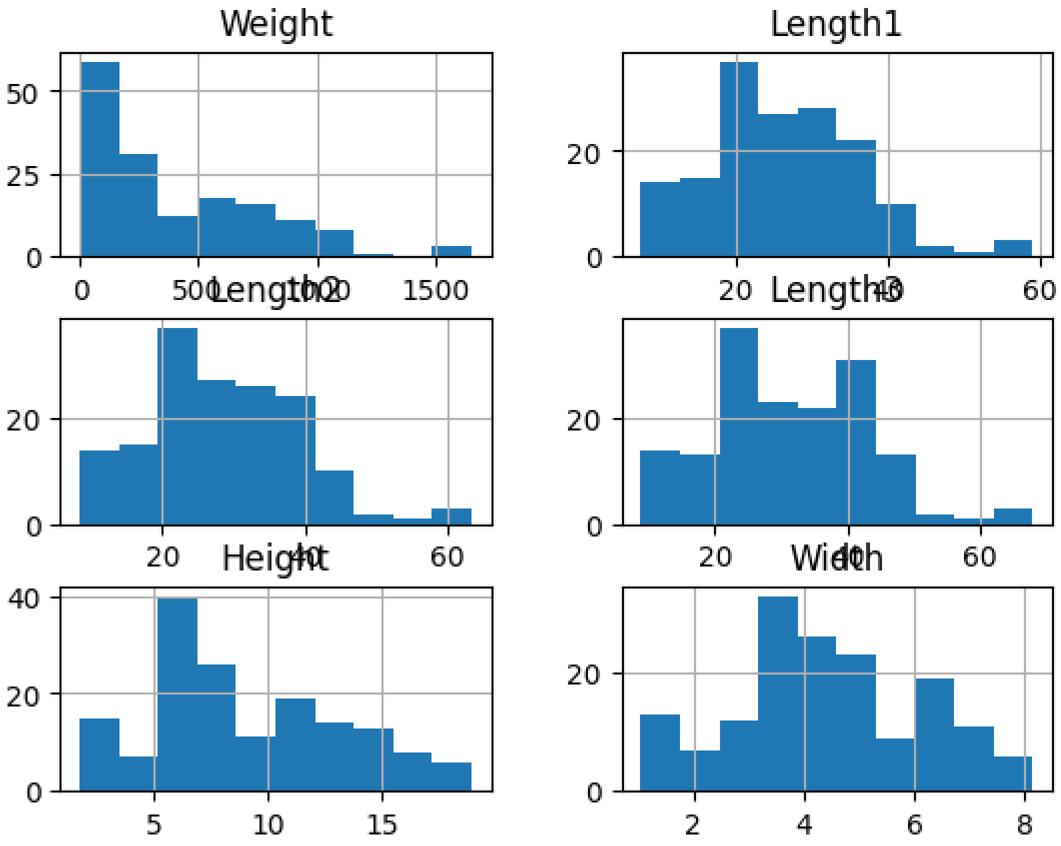
The highest frequency of Length3 is nearly 27 which is between 20 and 40.

The highest frequency of Height will be 40 which is between 5 and 10.

The highest frequency of Width is above 25 which is between 2 and 4.

```
In [ ]: plt.figure(figsize=(8,8))
        df_fish.hist()
        plt.show()
```

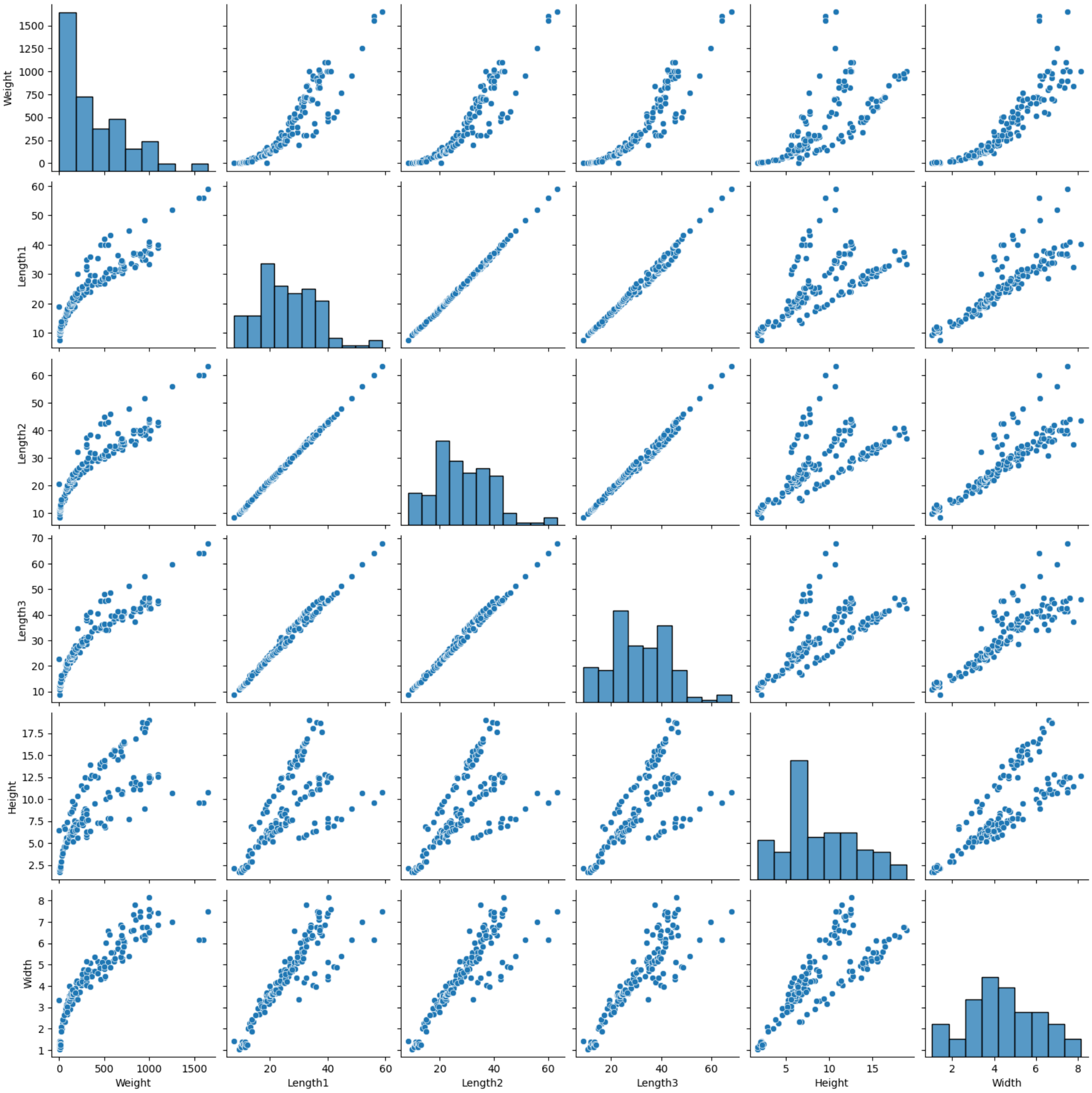
```
Out[ ]: array([[<Axes: title={'center': 'Weight'}>,
                <Axes: title={'center': 'Length1'}>],
               [<Axes: title={'center': 'Length2'}>,
                <Axes: title={'center': 'Length3'}>],
               [<Axes: title={'center': 'Height'}>,
                <Axes: title={'center': 'Width'}>]], dtype=object)
```



Pairplot will generate a grid of scatterplots where each cell in the grid corresponds to a pair of numerical columns from df_fish.The Diagonal Plots Are A Univariate Distribution Plot That Helps To Draw The Marginal Distribution Of The Data In Each Column.A Pair Plot Pairwise Relationships With Other Columns In The Data Frame And Also Plot Pair Plot With Itself.Here the pairplot is created on every column in the fish datasets

In [ ]:
```
sns.pairplot(df_fish)
```

Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x7cff07f11750>
```



A distribution plot graph is also created when can identify that there is a kind of overlapping in the case of weight

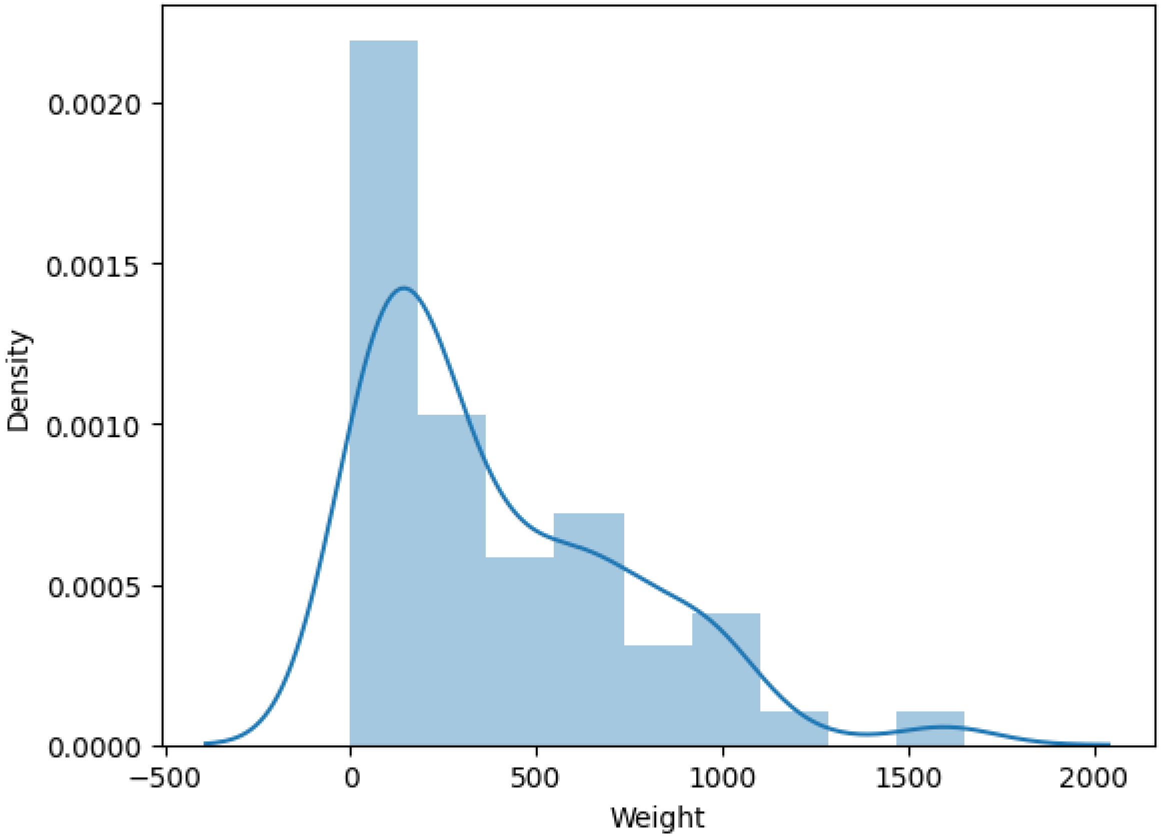
In [ ]:

```
sns.distplot(df_fish['Weight'])
```

```
<ipython-input-146-e39bafd42fe1>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df_fish['Weight'])
```

Out[ ]:

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

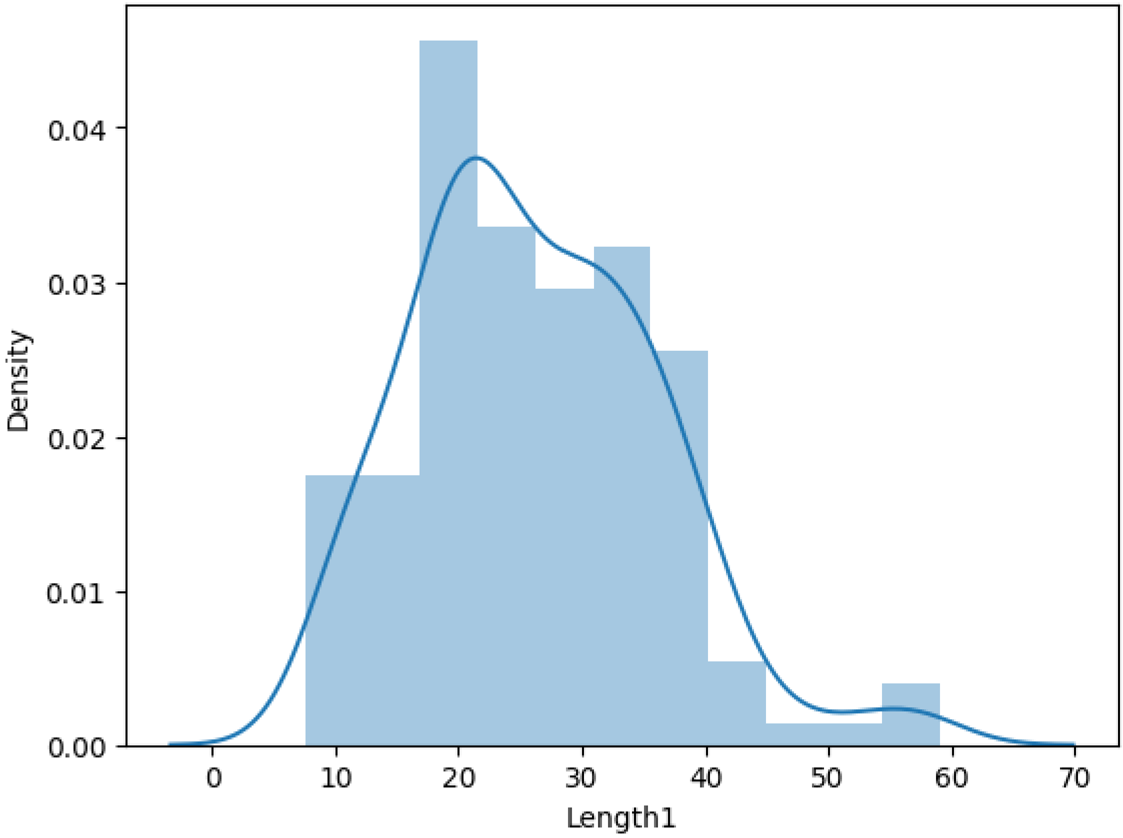


In case of Length1 there will be more overlapping than weight

In [ ]:
```
sns.distplot(df_fish['Length1'])
```

```
<ipython-input-147-5d14fec1f781>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df_fish['Length1'])
```

Out[ ]:
```
<Axes: xlabel='Length1', ylabel='Density'>
```

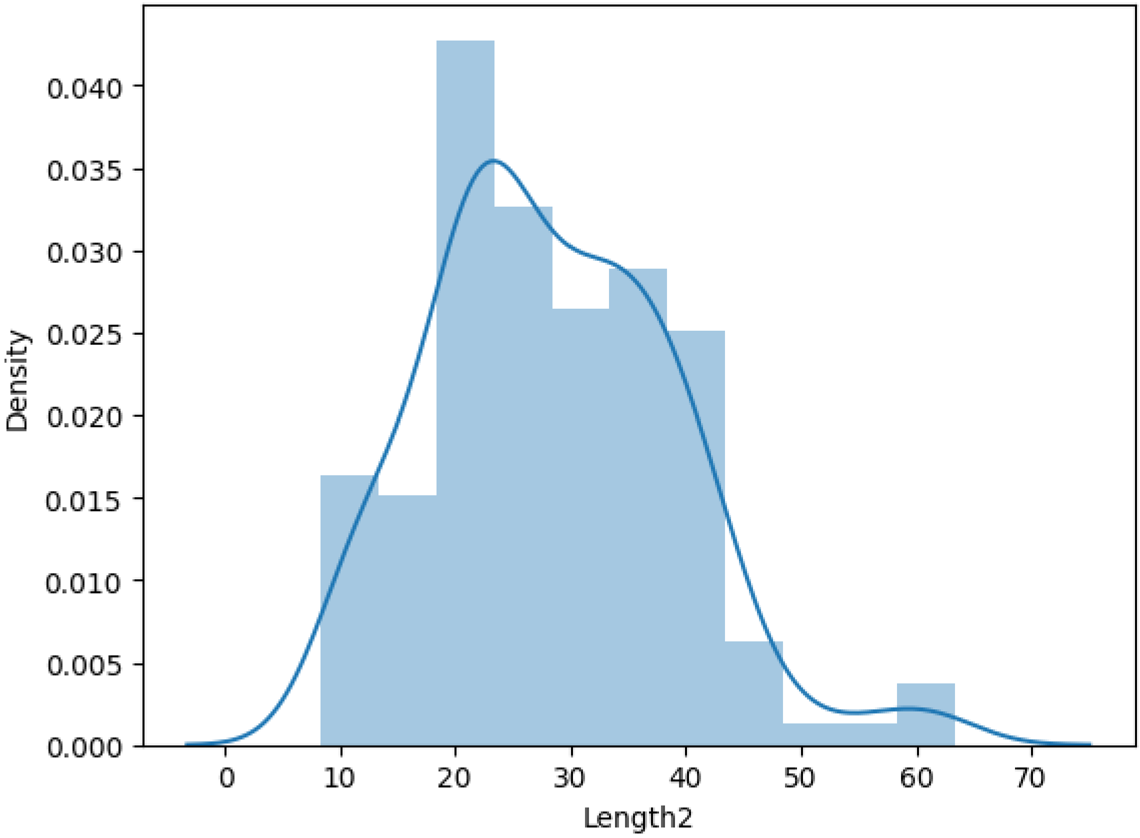


In case of Length2 there will be more overlapping.

```
In [ ]: sns.distplot(df_fish['Length2'])
```

```
<ipython-input-148-3329b3e40755>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df_fish['Length2'])
```

```
Out[ ]: <Axes: xlabel='Length2', ylabel='Density'>
```

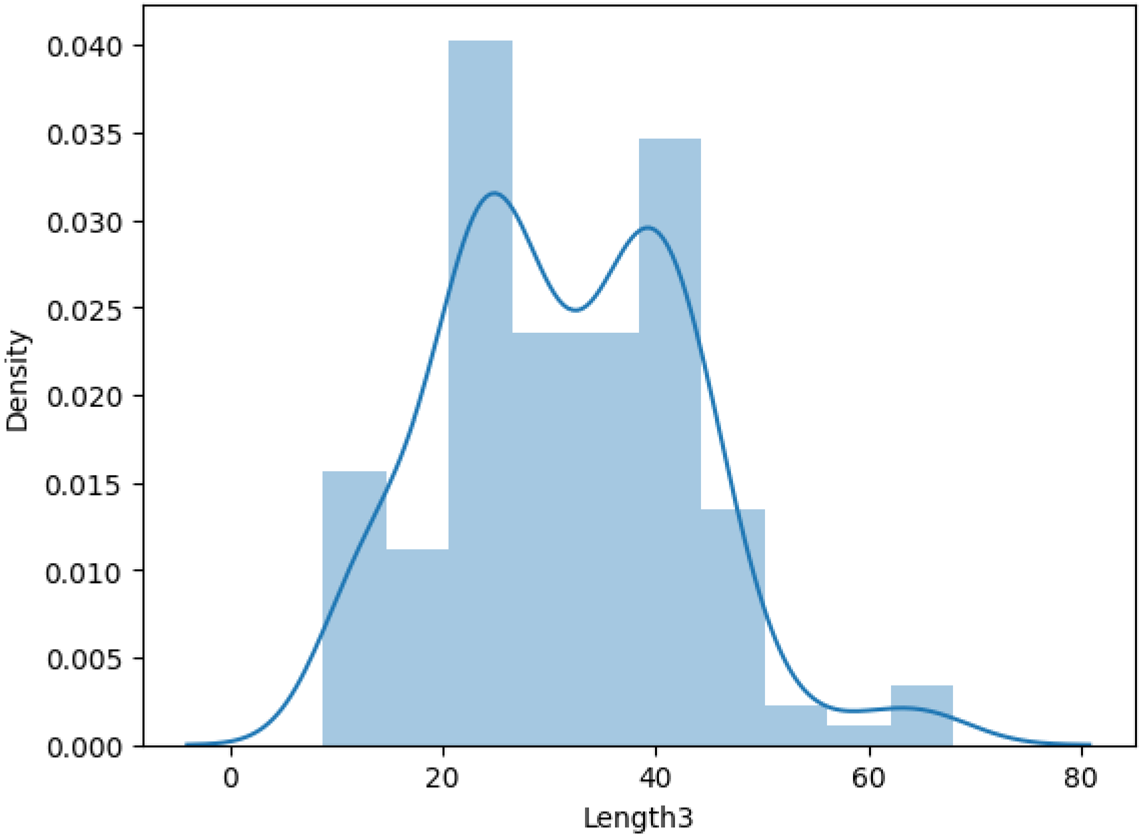


In case of Length3 there will be less overlapping as compared to length1 and length2

```
In [ ]: sns.distplot(df_fish['Length3'])
```

```
<ipython-input-56-4d89d843ad11>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df_fish['Length3'])
```

```
Out[ ]: <Axes: xlabel='Length3', ylabel='Density'>
```

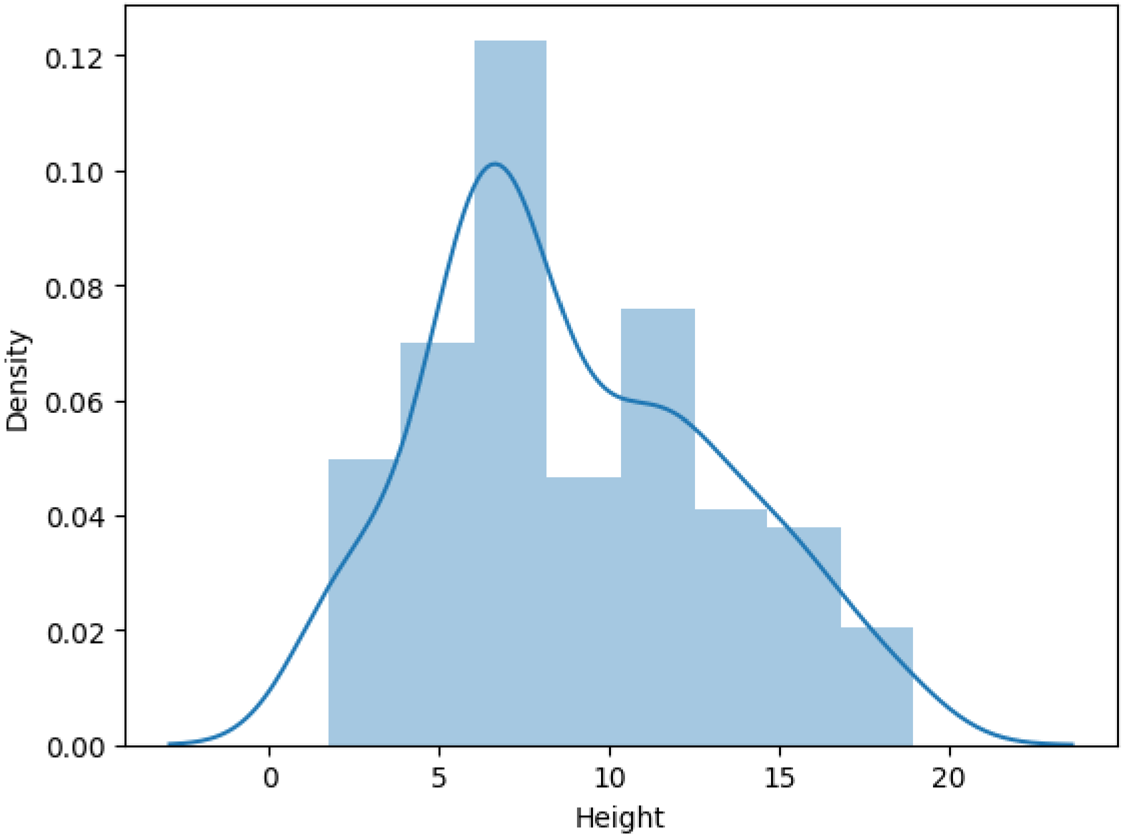


In case of Height there will be huge overlapping

```
In [ ]: sns.distplot(df_fish['Height'])
```

```
<ipython-input-150-72cd11077539>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df_fish['Height'])
```

```
Out[ ]: <Axes: xlabel='Height', ylabel='Density'>
```

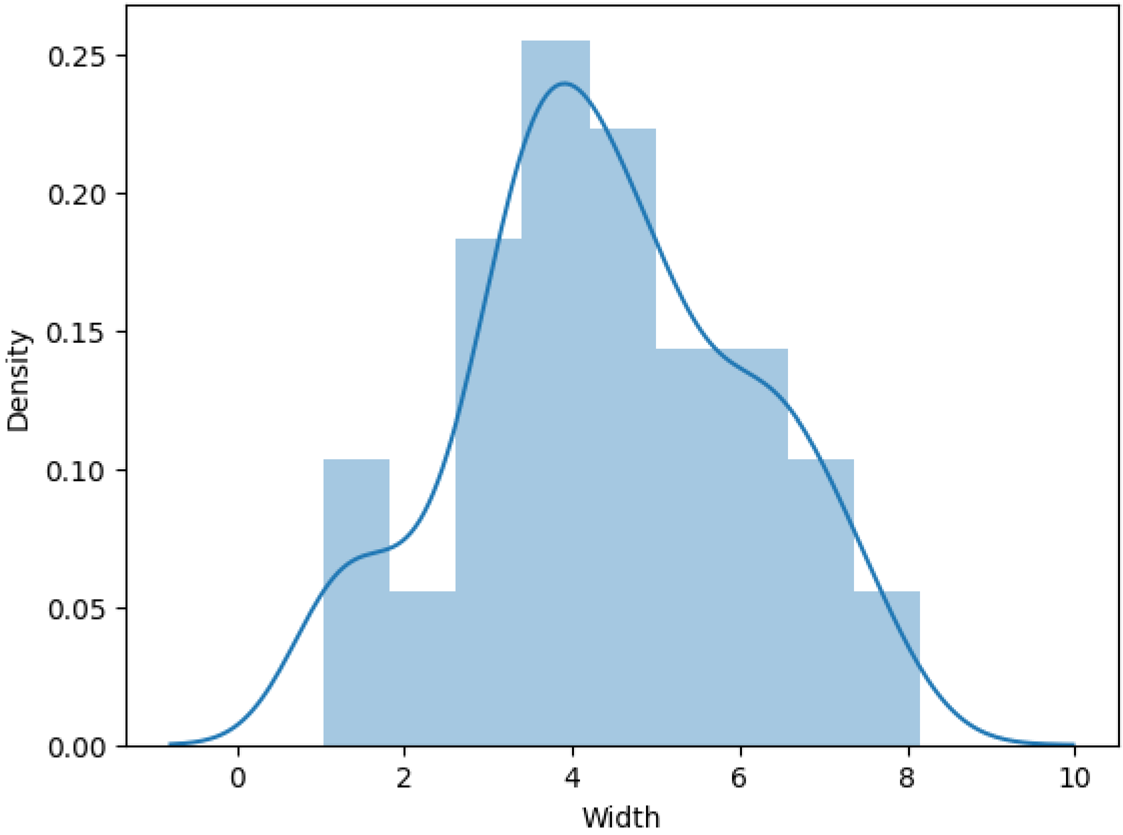


In case of width there will be huge overlapping

```
In [ ]: sns.distplot(df_fish['Width'])
```

```
<ipython-input-151-ad4de4188a0b>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df_fish['Width'])
```

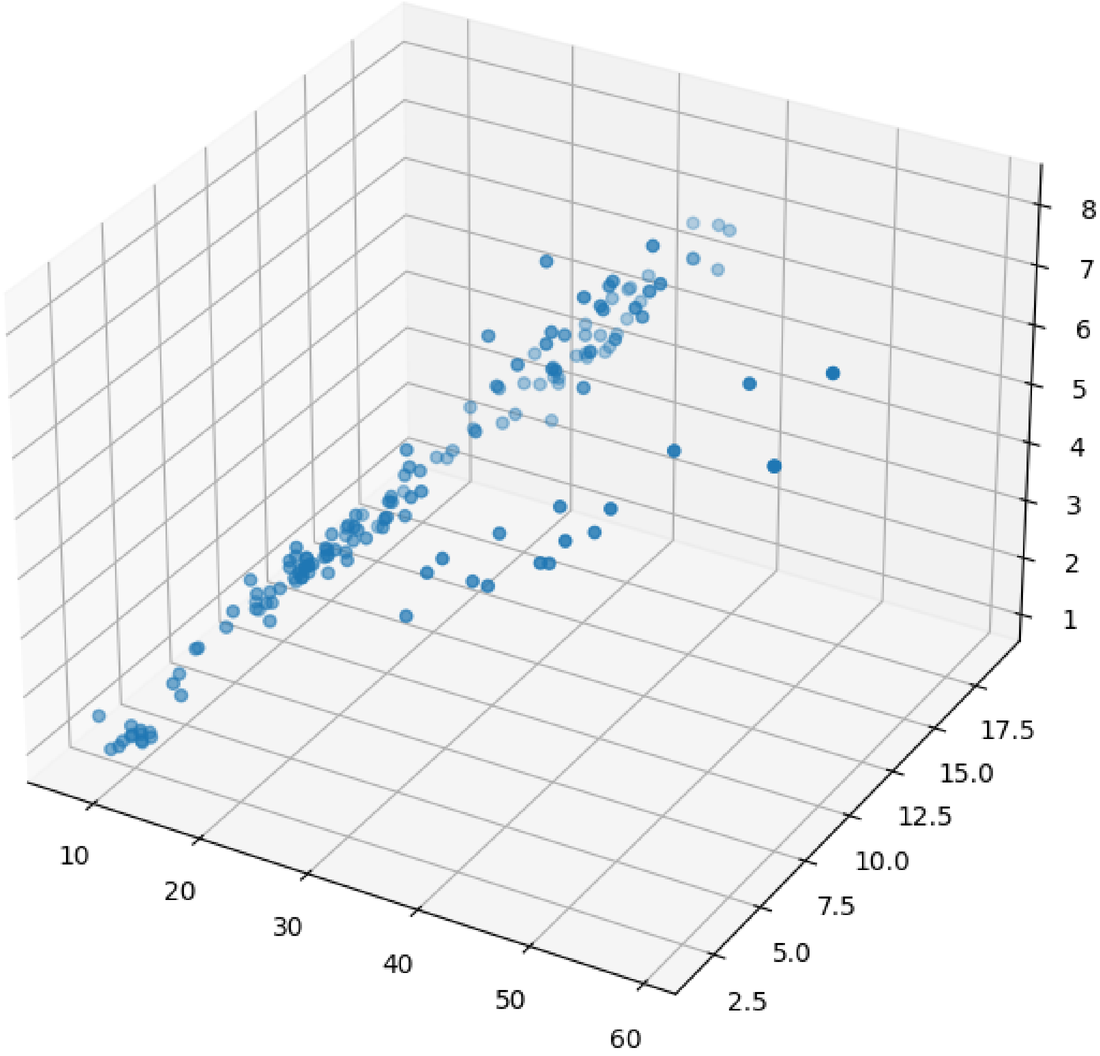
```
Out[ ]: <Axes: xlabel='Width', ylabel='Density'>
```



3D visualization for dataset using x-axis as Length1, y-axis as Height and z-axis as Width is created below

In [ ]:

```
from matplotlib.figure import projections
plt.figure(figsize=(10,8))
ax= plt.axes(projection='3d')
fg=ax.scatter3D(df_fish['Length1'],df_fish['Height'],df_fish['Width'])
```



Step 4 :Identifying significant correlations

Finding correlation using pearson method

corr(method='pearson') is used to calculate the Pearson correlation coefficients between numerical columns in a pandas DataFrame called df_fish.It measures the linear relationship between two continuous variables.The values indicate the strength and direction of the linear relationships between these variables. Positive values suggest positive correlations, negative values suggest negative correlations, and values close to 0 suggest little to no linear correlation.

In [ ]:
```
df_fish.corr(method='pearson')
```

```
<ipython-input-153-562e1f368bfb>:1: FutureWarning: The default value of numeric_on
ly in DataFrame.corr is deprecated. In a future version, it will default to False.
Select only valid columns or specify the value of numeric_only to silence this war
ning.
  df_fish.corr(method='pearson')
```

Out[ ]:

	Weight	Length1	Length2	Length3	Height	Width
Weight	1.000000	0.915712	0.918618	0.923044	0.724345	0.886507
Length1	0.915712	1.000000	0.999517	0.992031	0.625378	0.867050
Length2	0.918618	0.999517	1.000000	0.994103	0.640441	0.873547
Length3	0.923044	0.992031	0.994103	1.000000	0.703409	0.878520
Height	0.724345	0.625378	0.640441	0.703409	1.000000	0.792881
Width	0.886507	0.867050	0.873547	0.878520	0.792881	1.000000

In [ ]:

```
correlation= df_fish.corr('pearson')
```

```
<ipython-input-154-004b4a27028e>:1: FutureWarning: The default value of numeric_on
ly in DataFrame.corr is deprecated. In a future version, it will default to False.
Select only valid columns or specify the value of numeric_only to silence this war
ning.
  correlation= df_fish.corr()
```

A heatmap is also created.

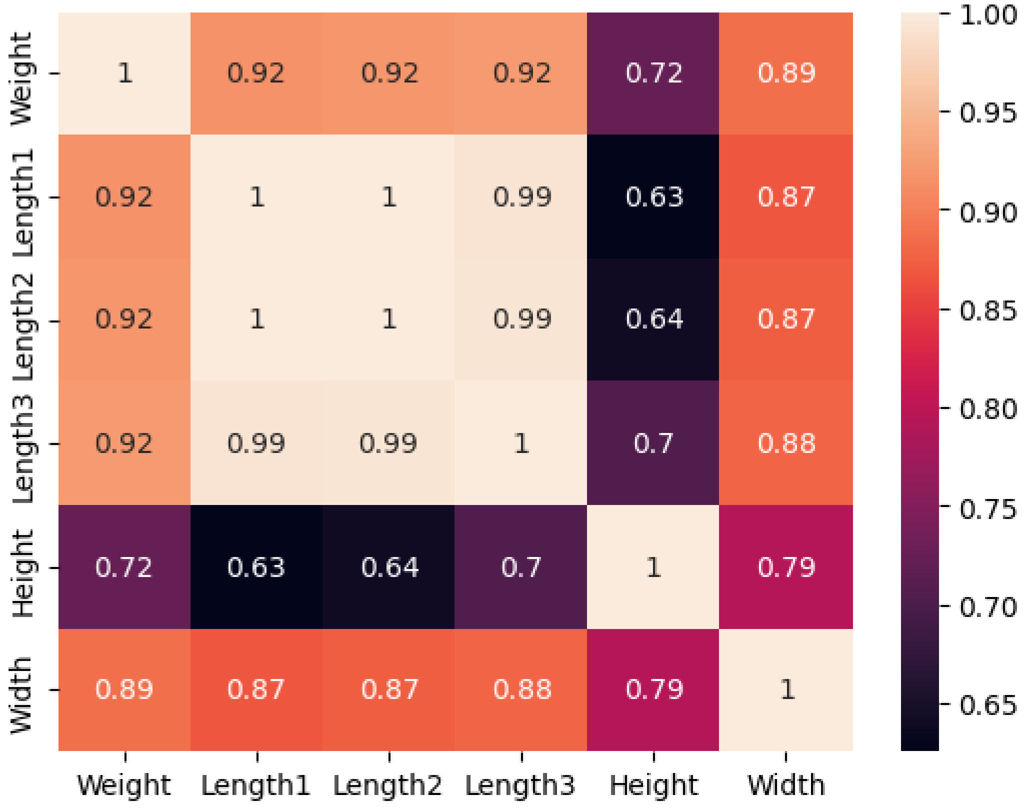
Each cell in the heatmap represents the correlation between two variables, with color intensity indicating the strength and direction of the correlation. Positive correlations and negative correlations are displayed with a different colors. The x-axis and y-axis of the heatmap represent the variables being correlated, with labels indicating their names. If annot=True, the actual correlation values will be displayed within each cell of the heatmap, making it easier to interpret.

In [ ]:

```
sns.heatmap(correlation, xticklabels=correlation.columns, yticklabels=correlation.
```

Out[ ]:

```
<Axes: >
```

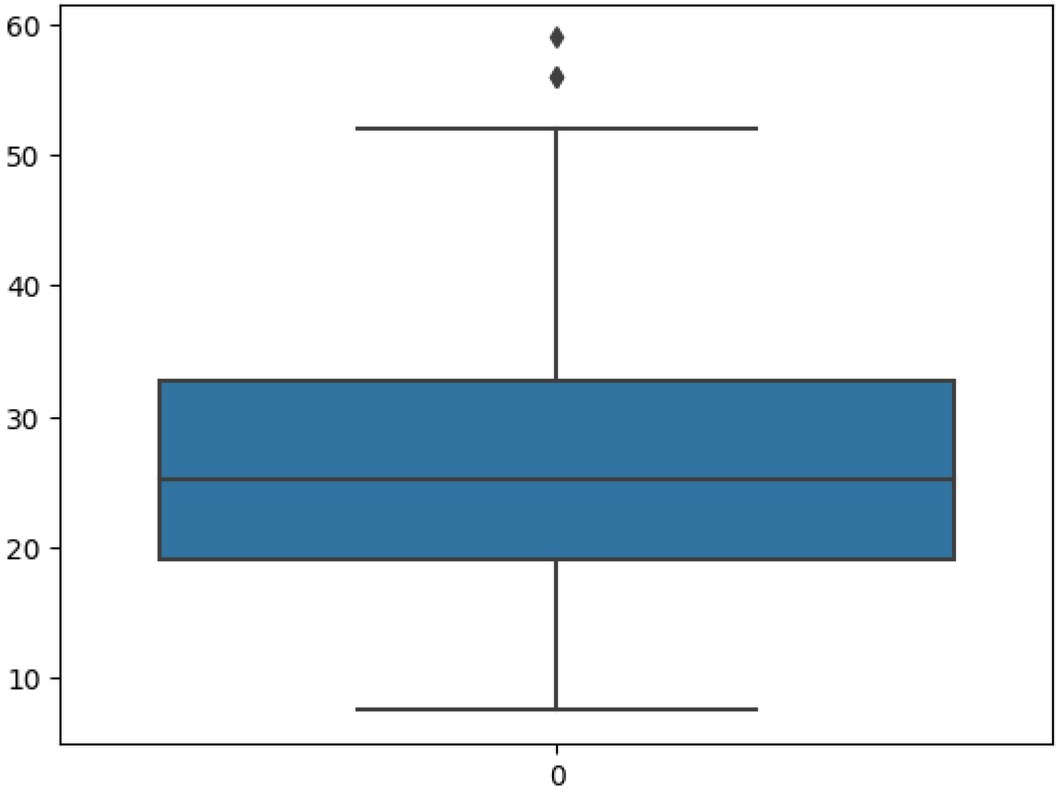


Step 5: Detecting and Handling outliers

Creating a box plot to find outliers

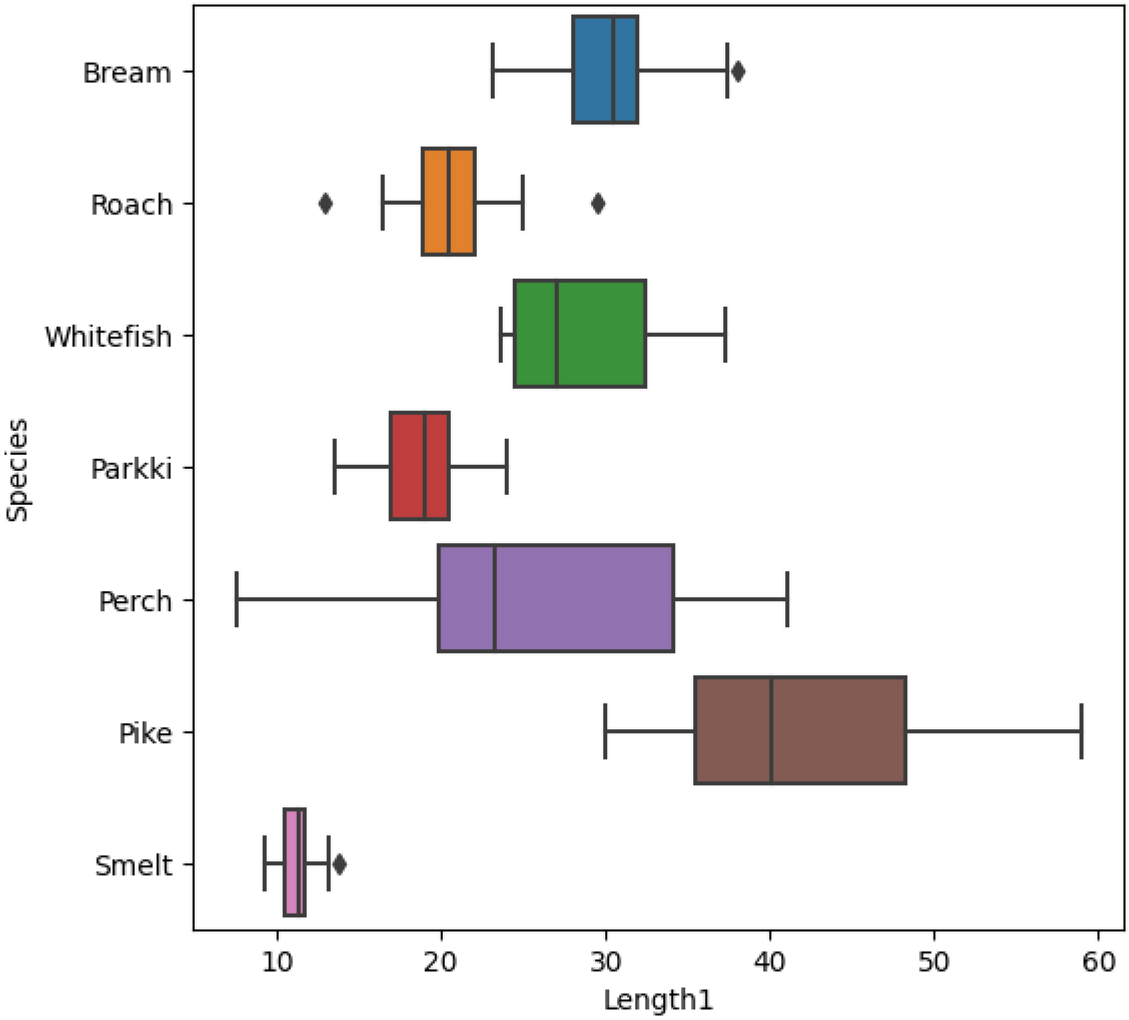
```
In [ ]: sns.boxplot(df_fish['Length1'])
```

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



Checking for the outliers present by using x-axis as "Length1" and y-axis as "Species"

```
In [ ]: plt.figure(figsize=(6,6))
        sns.boxplot(data=df_fish, y="Species", x="Length1")
        plt.show()
```



Finding the index positions of the outliers having species as bream and Length1>35

```
In [ ]: df_fish[(df_fish["Species"] == "Bream") & (df_fish["Length1"] >35)]
```

Out[ ]:

	Species	Weight	Length1	Length2	Length3	Height	Width
32	Bream	925.0	36.2	39.5	45.3	18.7542	6.7497
33	Bream	975.0	37.4	41.0	45.9	18.6354	6.7473
34	Bream	950.0	38.0	41.0	46.5	17.6235	6.3705

dropping the rows

```
In [ ]: df_fish.drop(34,inplace=True)
```

Finding the index positions of the outliers having species as roach and Length1<20

```
In [ ]: df_fish[(df_fish["Species"] == "Roach") & (df_fish["Length1"] <20)]
```

Out[ ]:

	Species	Weight	Length1	Length2	Length3	Height	Width
35	Roach	40.0	12.9	14.1	16.2	4.1472	2.2680
36	Roach	69.0	16.5	18.2	20.3	5.2983	2.8217
37	Roach	78.0	17.5	18.8	21.2	5.5756	2.9044
38	Roach	87.0	18.2	19.8	22.2	5.6166	3.1746
39	Roach	120.0	18.6	20.0	22.2	6.2160	3.5742
40	Roach	0.0	19.0	20.5	22.8	6.4752	3.3516
41	Roach	110.0	19.1	20.8	23.1	6.1677	3.3957
42	Roach	120.0	19.4	21.0	23.7	6.1146	3.2943

In [ ]:

```
df_fish.drop(35,inplace=True)
```

Finding the index positions of the outliers having species as roach and Length1>25

In [ ]:

```
df_fish[(df_fish["Species"] == "Roach") & (df_fish["Length1"] >25)]
```

Out[ ]:

	Species	Weight	Length1	Length2	Length3	Height	Width
54	Roach	390.0	29.5	31.7	35.0	9.485	5.355

dropping the rows

In [ ]:

```
df_fish.drop(54,inplace=True)
```

Finding the index positions of the outliers having species as smelt and Length1>10

In [ ]:

```
df_fish[(df_fish["Species"] == "Smelt") & (df_fish["Length1"] >10)]
```

Out[ ]:

	Species	Weight	Length1	Length2	Length3	Height	Width
147	Smelt	7.0	10.1	10.6	11.6	1.7284	1.1484
148	Smelt	9.7	10.4	11.0	12.0	2.1960	1.3800
149	Smelt	9.8	10.7	11.2	12.4	2.0832	1.2772
150	Smelt	8.7	10.8	11.3	12.6	1.9782	1.2852
151	Smelt	10.0	11.3	11.8	13.1	2.2139	1.2838
152	Smelt	9.9	11.3	11.8	13.1	2.2139	1.1659
153	Smelt	9.8	11.4	12.0	13.2	2.2044	1.1484
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

dropping the rows

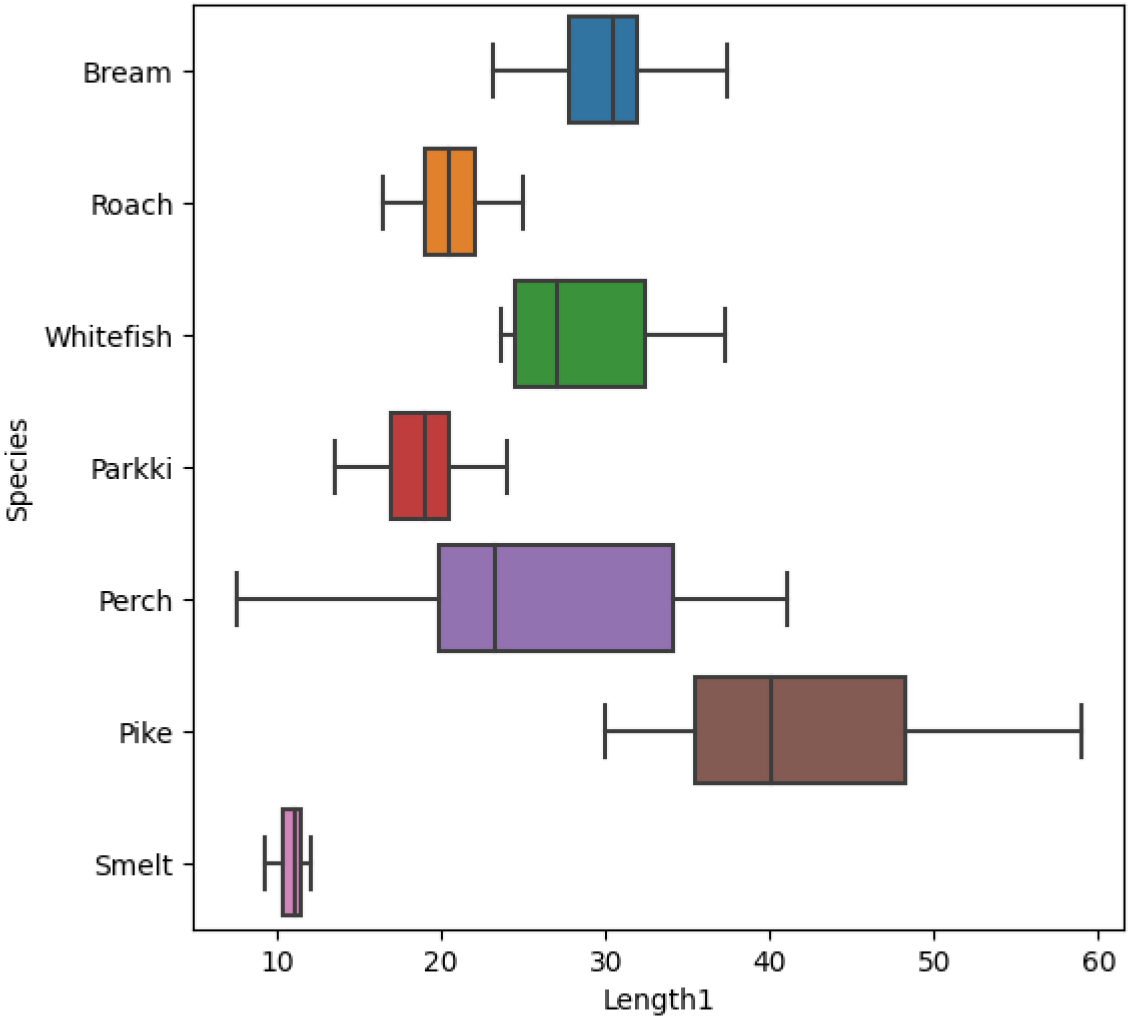
```
In [ ]: df_fish.drop(157,inplace=True)
```

dropping the rows

```
In [ ]: df_fish.drop(158,inplace=True)
```

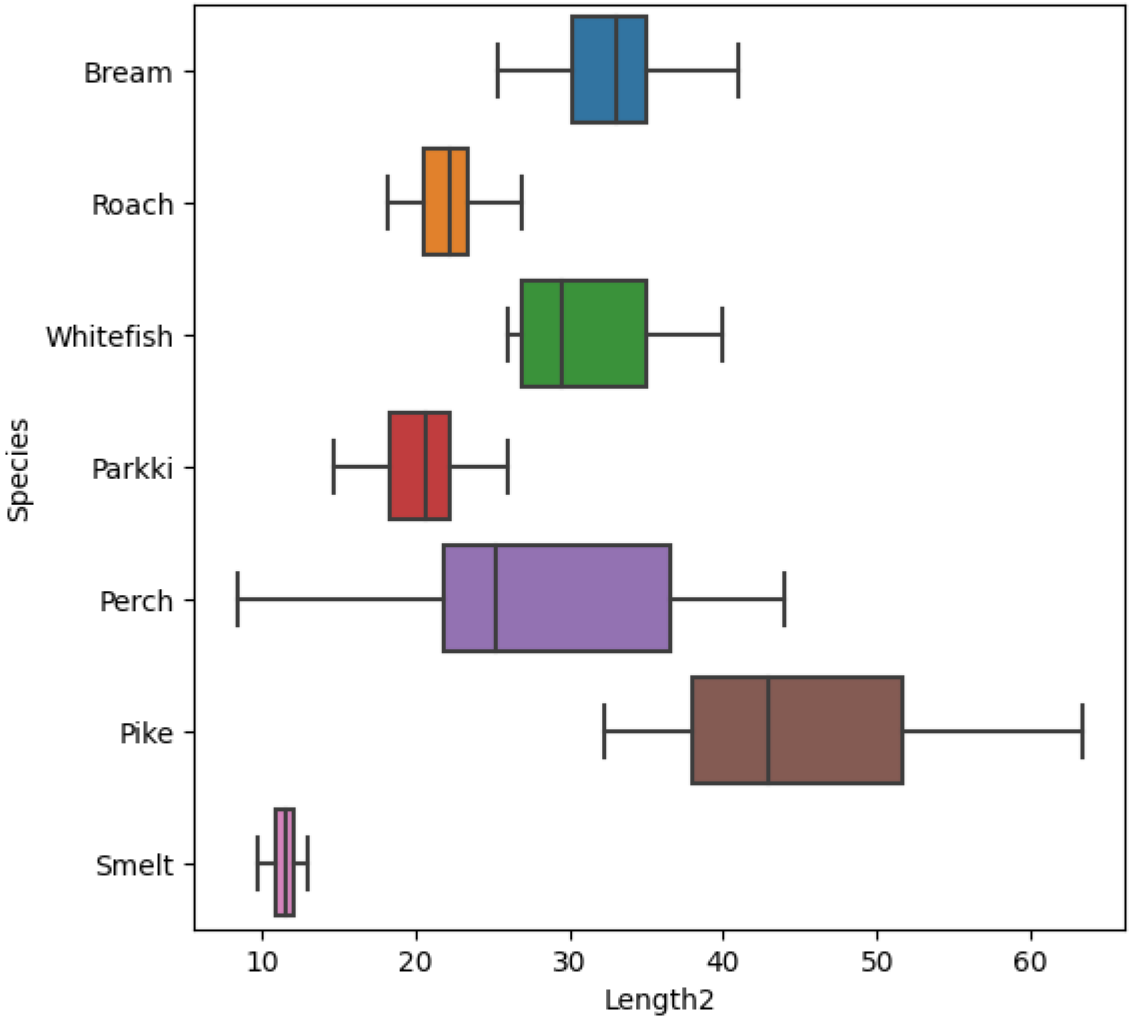
Outliers are completely removed from the data

```
In [ ]: plt.figure(figsize=(6,6))
        sns.boxplot(data=df_fish, y="Species", x="Length1")
        plt.show()
```



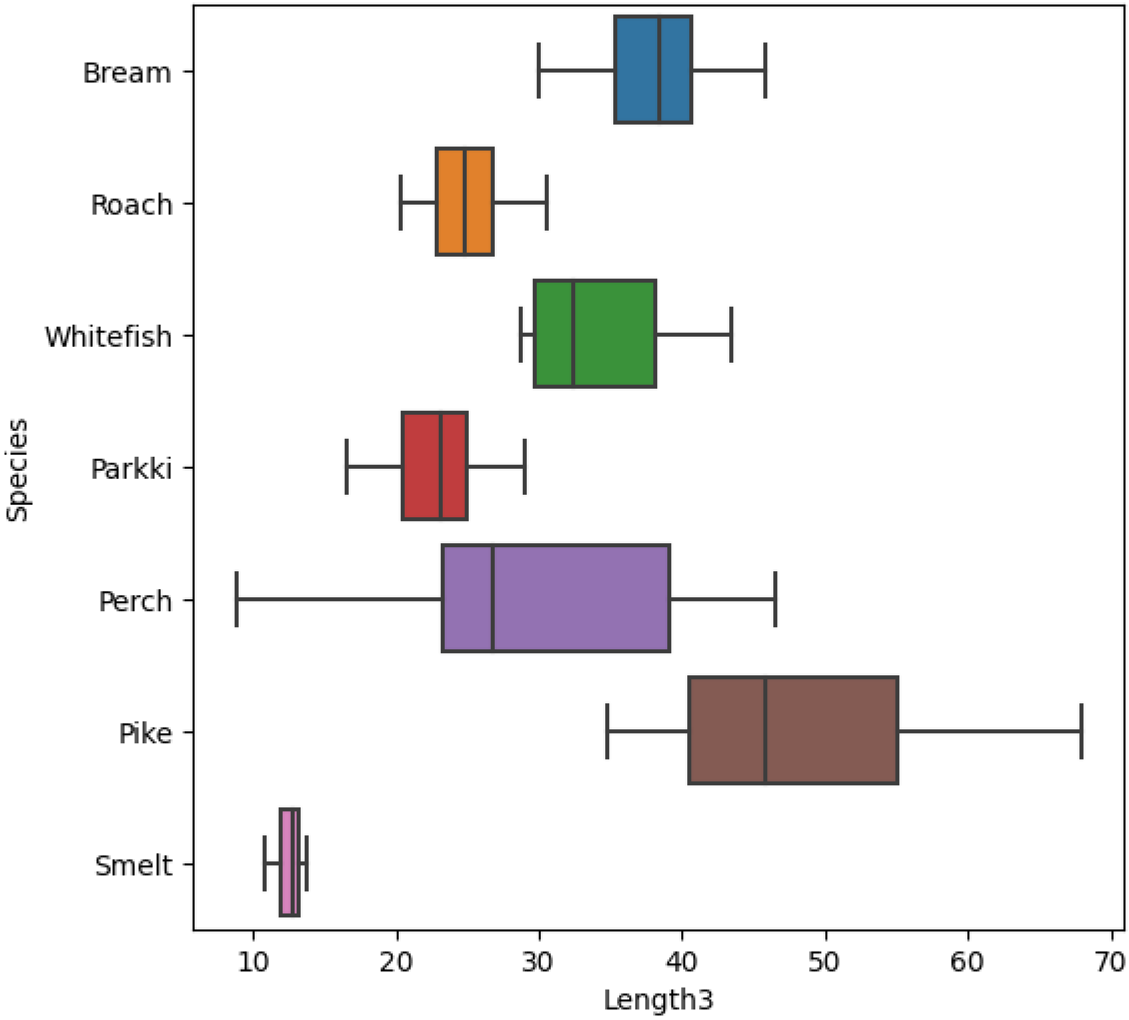
Checking for the outliers present by using x-axis as "Length2" and y-axis as "Species"

```
In [ ]: plt.figure(figsize=(6,6))
        sns.boxplot(data=df_fish, y="Species", x="Length2")
        plt.show()
```



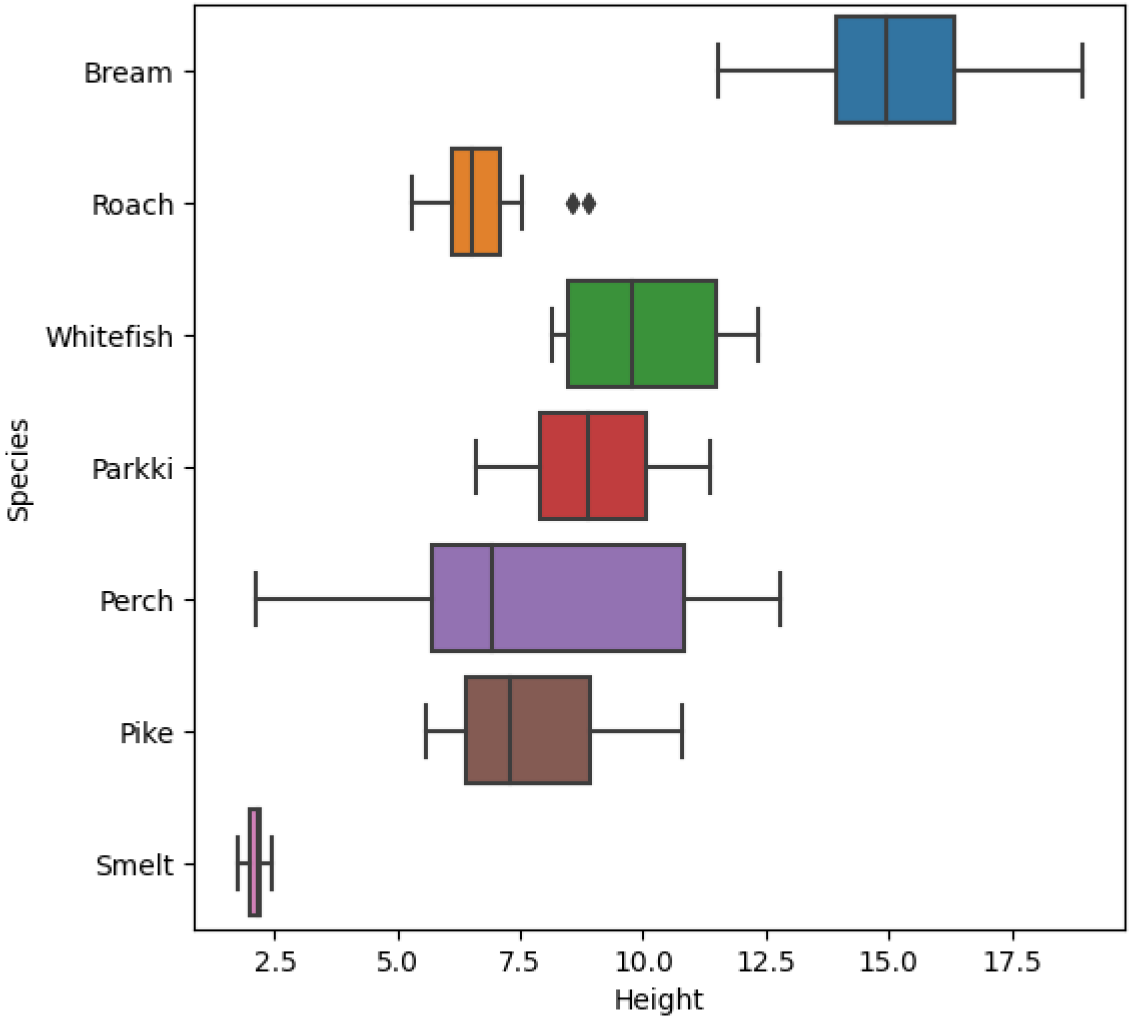
Checking for the outliers present by using x-axis as "Length3" and y-axis as "Species"

In [ ]:
```
plt.figure(figsize=(6,6))
sns.boxplot(data=df_fish, y="Species", x="Length3")
plt.show()
```



Checking for the outliers present by using x-axis as "Height" and y-axis as "Species"

In [ ]:
```
plt.figure(figsize=(6,6))
sns.boxplot(data=df_fish, y="Species", x="Height")
plt.show()
```



Finding the index positions of the outliers having species as roach and height>8

In [ ]:
```
df_fish[(df_fish["Species"] == "Roach") & (df_fish["Height"] >8)]
```

Out[ ]:

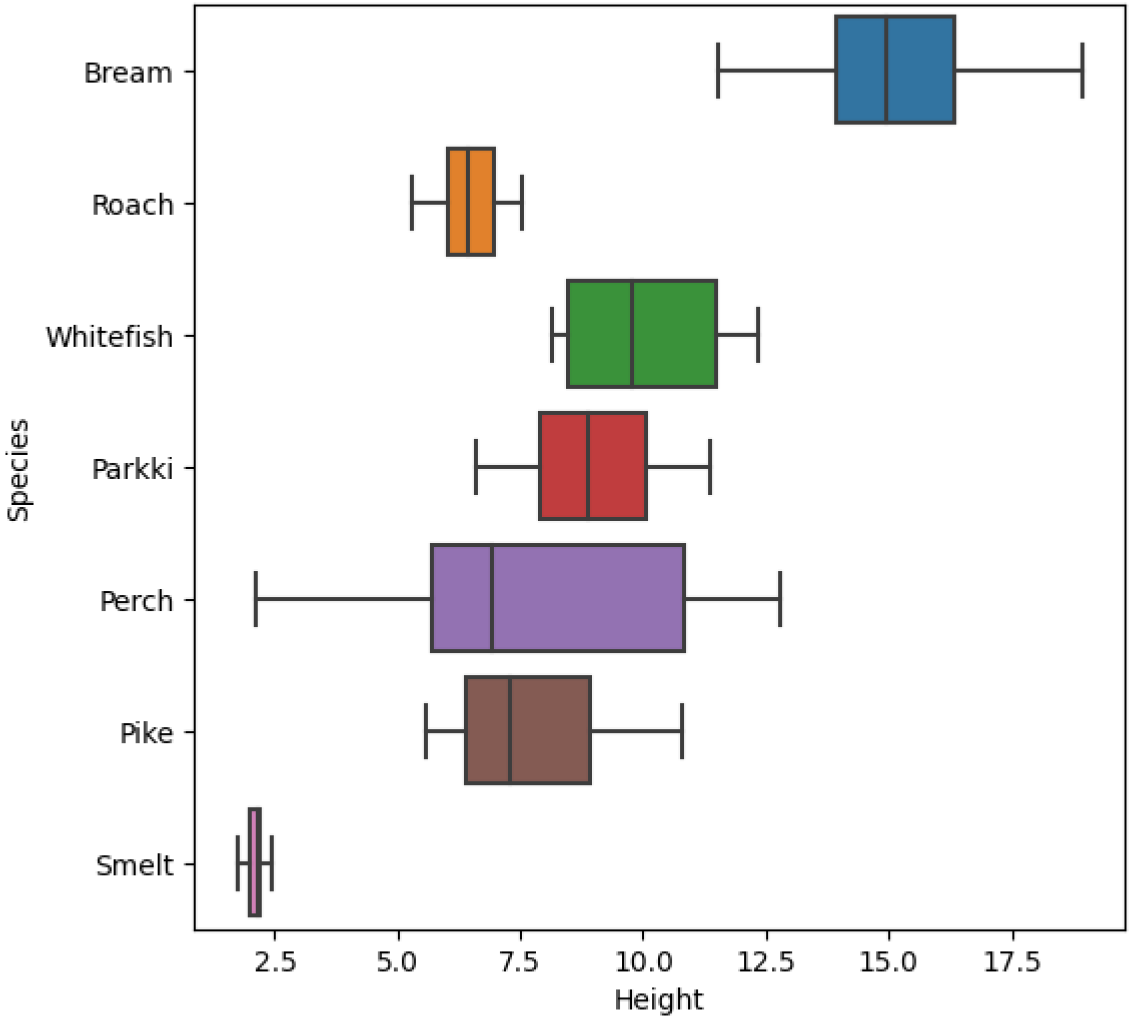
	Species	Weight	Length1	Length2	Length3	Height	Width
52	Roach	290.0	24.0	26.0	29.2	8.8768	4.4968
53	Roach	272.0	25.0	27.0	30.6	8.5680	4.7736

dropping the rows

In [ ]:
```
df_fish.drop(52,inplace=True)
df_fish.drop(53,inplace=True)
```

Outliers are removed

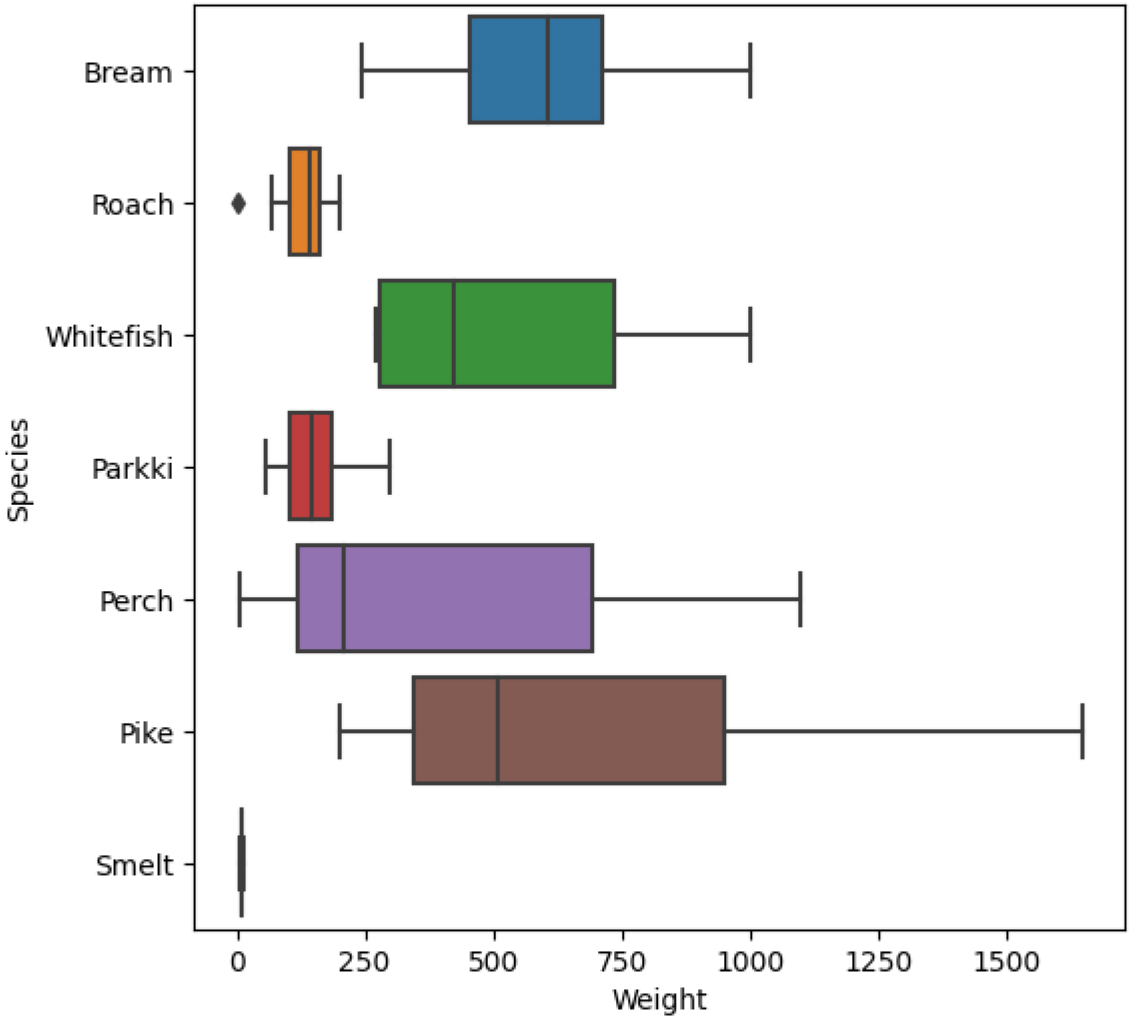
In [ ]:
```
plt.figure(figsize=(6,6))
sns.boxplot(data=df_fish, y="Species", x="Height")
plt.show()
```



Checking for the outliers present by using x-axis as "Weight" and y-axis as "Species"

In [ ]:

```
plt.figure(figsize=(6,6))
sns.boxplot(data=df_fish, y="Species", x="Weight")
plt.show()
```



Finding the index positions of the outliers having species as roach and Weight<100

In [ ]:
```
df_fish[(df_fish["Species"] == "Roach") & (df_fish["Weight"] <100)]
```

Out[ ]:

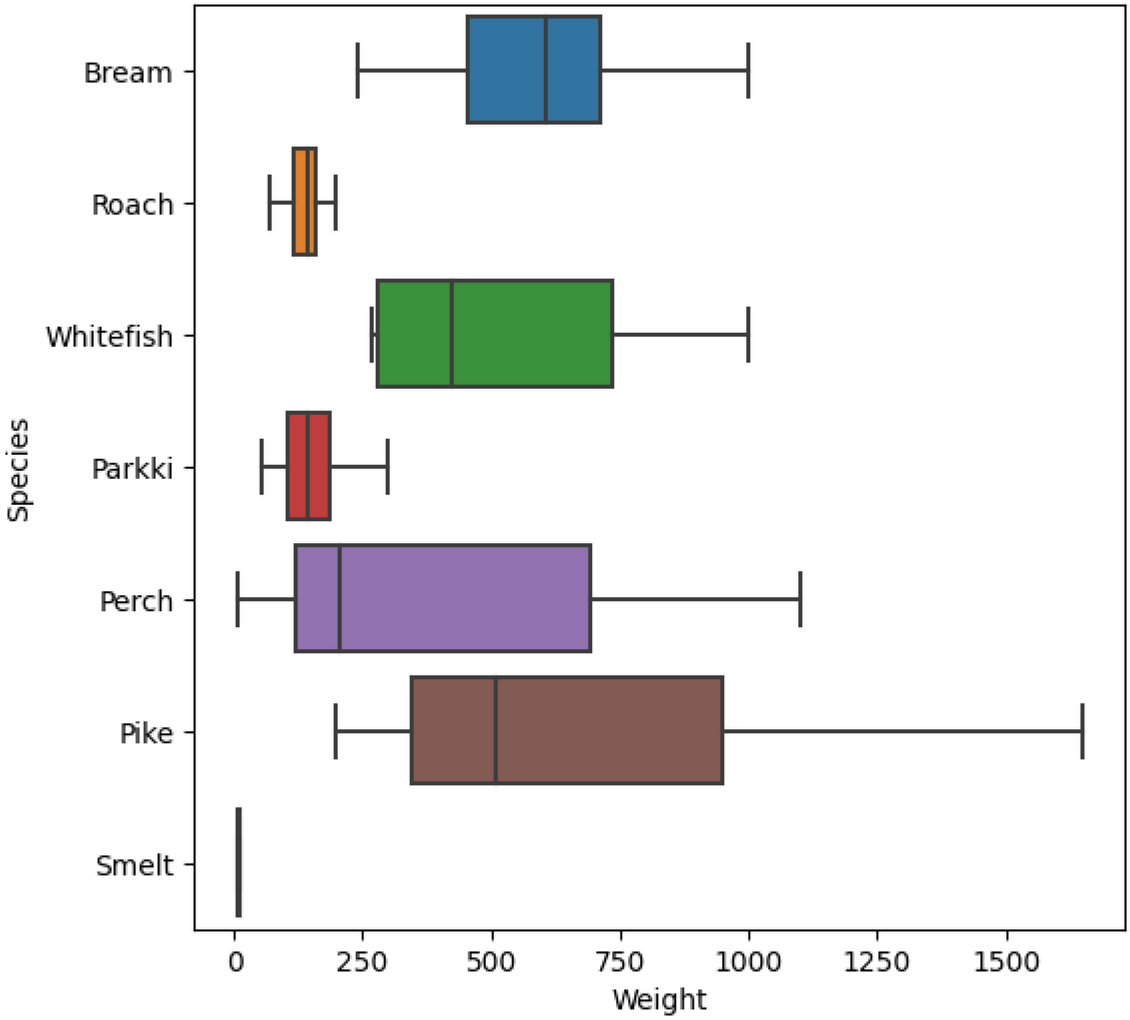
	Species	Weight	Length1	Length2	Length3	Height	Width
36	Roach	69.0	16.5	18.2	20.3	5.2983	2.8217
37	Roach	78.0	17.5	18.8	21.2	5.5756	2.9044
38	Roach	87.0	18.2	19.8	22.2	5.6166	3.1746
40	Roach	0.0	19.0	20.5	22.8	6.4752	3.3516

dropping the rows

In [ ]:
```
df_fish.drop(40,inplace=True)
```

Outliers are removed

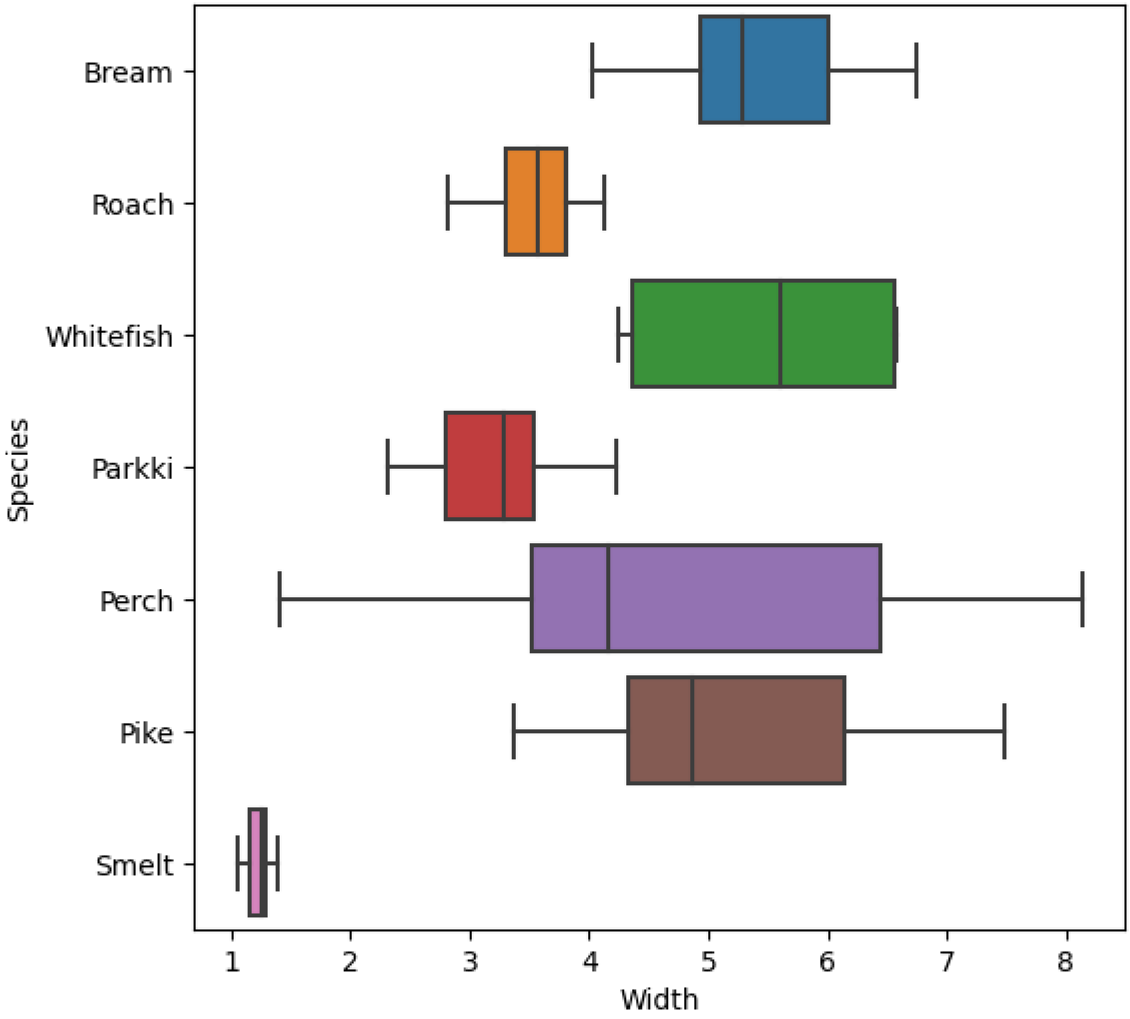
In [ ]:
```
plt.figure(figsize=(6,6))
sns.boxplot(data=df_fish, y="Species", x="Weight")
plt.show()
```



Checking for the outliers present by using x-axis as "Width" and y-axis as "Species"

In [ ]:

```
plt.figure(figsize=(6,6))
sns.boxplot(data=df_fish, y="Species", x="Width")
plt.show()
```



After doing EDA analysis, we could come into the following conclusion that:

- The perch species are very higher compared to any other species and whitefish is having very less number.
- It can be understood that smelt species of fish are very lower proportionate in size,that is, having very less height,weight,width and length values.
- Perch species consist of different size that is having less height, weight ,width and length values to higher values.
- The heaviest fish will be from Pike species