Name: Soham Belurgikar

Roll No.: 2019130006

Course: DA (Data Analytics)

Experiment No.: 1

Name of the Experiment: Exploratory Data Analysis

Objective: Perform EDA such as number of data samples, number of features, number of classes, number of data samples per class, removing missing values, conversion to numbers, using seaborn library to plot different graphs.

Theory:

What is Exploratory Data Analysis?

Exploratory Data Analysis or (EDA) is understanding the data sets by summarizing their main characteristics often plotting them visually. This step is very important especially when we arrive at modelling the data in order to apply Machine learning. Plotting in EDA consists of Histograms, Box plot, Scatter plot and many more. It often takes much time to explore the data. Through the process of EDA, we can ask to define the problem statement or definition on our data set which is very important. There is no one method or common methods in order to perform EDA, since EDA is not a set of methods but it is instead a philosophy.

The objectives of EDA can be summarized as follows:

1. Maximize insight into the database/understand the database structure;
2. Visualize potential relationships (direction and magnitude) between exposure and outcome variables;
3. Detect outliers and anomalies (values that are significantly different from the other observations);
4. Develop parsimonious models (a predictive or explanatory model that performs with as few exposure variables as possible) or preliminary selection of appropriate models;
5. Extract and create clinically relevant variables.

Problem Statement:

This analysis is aimed at those who are interested in statistics related to meteorites landed on / observed from the Earth. The dataset contains various attributes associated with meteorites such as mass, type of meteorite, year of landing / observation, the location of landing / observation, etc. The main aim is to determine the number of meteorites belonging to particular attributes and also to determine if there exists a relation between the attributes.

Implementation:

Dataset used: Meteorite Landings dataset

Importing the required libraries:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(color_codes=True)
```

Loading the data into the dataframe:

```
df = pd.read_csv("Meteorite_Landings.csv")
df.head()
```

.head() displays the first 5 rows of the table:

	name	id	nametype	recclass	mass (g)	fall	year	reclat	reclong	GeoLocation
0	Aachen	1	Valid	L5	21.0	Fell	1880.0	50.77500	6.08333	(50.775, 6.08333)
1	Aarhus	2	Valid	H6	720.0	Fell	1951.0	56.18333	10.23333	(56.18333, 10.23333)
2	Abee	6	Valid	EH4	107000.0	Fell	1952.0	54.21667	-113.00000	(54.21667, -113.0)
3	Acapulco	10	Valid	Acapulcoite	1914.0	Fell	1976.0	16.88333	-99.90000	(16.88333, -99.9)
4	Achiras	370	Valid	L6	780.0	Fell	1902.0	-33.16667	-64.95000	(-33.16667, -64.95)

```
df.tail()
```

.tail() displays the last 5 rows of the table:

	name	id	nametype	recclass	mass (g)	fall	year	reclat	reclong	GeoLocation
45711	Zillah 002	31356	Valid	Eucrite	172.0	Found	1990.0	29.03700	17.01850	(29.037, 17.0185)
45712	Zinder	30409	Valid	Pallasite, ungrouped	46.0	Found	1999.0	13.78333	8.96667	(13.78333, 8.96667)
45713	Zlin	30410	Valid	H4	3.3	Found	1939.0	49.25000	17.66667	(49.25, 17.66667)
45714	Zubkovsky	31357	Valid	L6	2167.0	Found	2003.0	49.78917	41.50460	(49.78917, 41.5046)
45715	Zulu Queen	30414	Valid	L3.7	200.0	Found	1976.0	33.98333	-115.68333	(33.98333, -115.68333)

Under NameType, 'valid' is for most meteorites and 'relict' are for objects that were once meteorites but are now highly altered by weathering on Earth. The ‘recclass’ column contains the type of meteors. The ‘fall’ column contains either of two values – ‘Fell’ or ‘Found’. Found basically means the meteorite was just observed, but it did not necessarily fall on the earth. The rest of the columns are self-explanatory.

Renaming some of the columns:

```
df = df.rename(columns={"recclass": "class", "reclat": "latitude", "rec
long": "longitude"})
df.head()
```

	name	id	nametype	class	mass (g)	fall	year	latitude	longitude	GeoLocation
0	Aachen	1	Valid	L5	21.0	Fell	1880.0	50.77500	6.08333	(50.775, 6.08333)
1	Aarhus	2	Valid	H6	720.0	Fell	1951.0	56.18333	10.23333	(56.18333, 10.23333)
2	Abee	6	Valid	EH4	107000.0	Fell	1952.0	54.21667	-113.00000	(54.21667, -113.0)
3	Acapulco	10	Valid	Acapulcoite	1914.0	Fell	1976.0	16.88333	-99.90000	(16.88333, -99.9)
4	Achiras	370	Valid	L6	780.0	Fell	1902.0	-33.16667	-64.95000	(-33.16667, -64.95)

```
df.shape
```

Using .shape() we can get information about the number of rows and columns of the dataset:

```
(45716, 10)
```

So, the dataset contains 45716 rows (samples) and 10 columns (features).

Removing duplicate rows:

```
duplicate_rows_df = df[df.duplicated()]
print("number of duplicate rows: ", duplicate_rows_df.shape)
```

This gives us the number of rows which have the same values for every column:

```
number of duplicate rows:  (0, 10)
```

So, the dataset doesn't contain any duplicates.

```
df.count()
```

You can also check the number of rows that each column contains using the .count() method:

```
name           45716
id             45716
nametype       45716
class          45716
mass (g)       45585
fall           45716
year           45425
latitude       38401
longitude      38401
GeoLocation    38401
dtype: int64
```

You can delete the duplicate rows using just a simple method, i.e., .drop_duplicates():

```
df = df.drop_duplicates()
df.head()
```

	name	id	nametype	class	mass (g)	fall	year	latitude	longitude	GeoLocation
0	Aachen	1	Valid	L5	21.0	Fell	1880.0	50.77500	6.08333	(50.775, 6.08333)
1	Aarhus	2	Valid	H6	720.0	Fell	1951.0	56.18333	10.23333	(56.18333, 10.23333)
2	Abee	6	Valid	EH4	107000.0	Fell	1952.0	54.21667	-113.00000	(54.21667, -113.0)
3	Acapulco	10	Valid	Acapulcoite	1914.0	Fell	1976.0	16.88333	-99.90000	(16.88333, -99.9)
4	Achiras	370	Valid	L6	780.0	Fell	1902.0	-33.16667	-64.95000	(-33.16667, -64.95)

```
df.count()
```

```
name           45716
id             45716
nametype       45716
class          45716
mass (g)       45585
fall           45716
year           45425
latitude       38401
longitude      38401
GeoLocation    38401
dtype: int64
```

Determining the number of rows for each class:

```
df['class'].value_counts()
```

The .value_counts() method returns the number of unique rows associated with a given value (in this case, all the values in the ‘class’ column):

```
L6          8285
H5          7142
L5          4796
H6          4528
H4          4211
            ...
EL7            1
CH/CBb         1
H/L~4          1
LL3.7-6        1
L/LL           1
Name: class, Length: 466, dtype: int64
```

So, the ‘class’ column contains 466 unique values, with L6 having the highest count.

Removing null / missing values:

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

The .isnull().sum() command will return the number of values which are missing for every column:

```
name               0
id                 0
nametype           0
class              0
mass (g)         131
fall               0
year             291
latitude        7315
longitude       7315
GeoLocation     7315
dtype: int64
```

As you can see, the last three columns contain over 7000 null values. We can drop these rows using .dropna():

```
df = df.dropna()
df.count()
```

```
name           38115
id             38115
nametype       38115
class          38115
```

```
mass (g)        38115
fall            38115
year            38115
latitude        38115
longitude       38115
GeoLocation     38115
dtype: int64
```

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

```
name            0
id              0
nametype        0
class           0
mass (g)        0
fall            0
year            0
latitude        0
longitude       0
GeoLocation     0
dtype: int64
```

Now our dataset is free of null values.

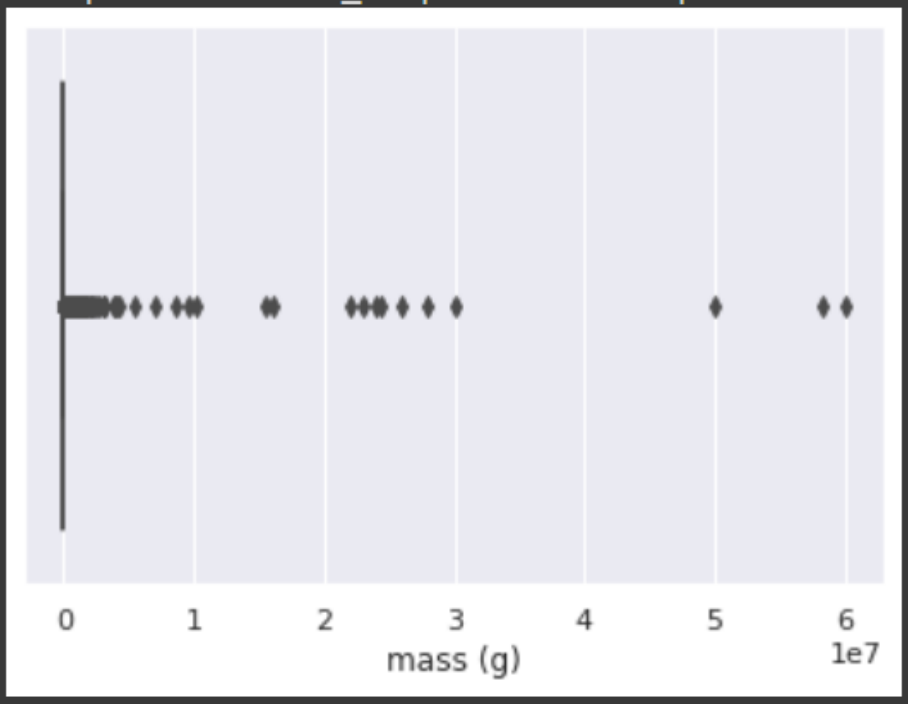
Detecting outliers:

Outliers can be seen with visualizations using a box plot. We can plot a box plot using the seaborn library:

```
sns.boxplot(x=df['mass (g)'])
```

This will show a box plot for the ‘mass (g) column:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcb89c41050>
```



The output is interesting because at first glance one would think the box is not present but on observing the scale, we can see that the X- axis, i.e., the mass (g) ranges from 0 to 6 * 1e7, i.e., 6 * 10^7. So, in fact, the box is present, but is too small to be shown correctly.

For finding the actual dimensions of the box, we can take the help of the .describe() method:

```
df.describe()
```

	id	mass (g)	year	latitude	longitude
count	38115.000000	3.811500e+04	38115.000000	38115.000000	38115.000000
mean	25343.139000	1.560071e+04	1989.993913	-39.596529	61.309359
std	17395.360205	6.286817e+05	25.469892	46.175830	80.777583
min	1.000000	0.000000e+00	860.000000	-87.366670	-165.433330
25%	10831.500000	6.630000e+00	1986.000000	-76.716670	0.000000
50%	21732.000000	2.909000e+01	1996.000000	-71.500000	35.666670
75%	39887.500000	1.872900e+02	2002.000000	0.000000	157.166670
max	57458.000000	6.000000e+07	2101.000000	81.166670	178.200000

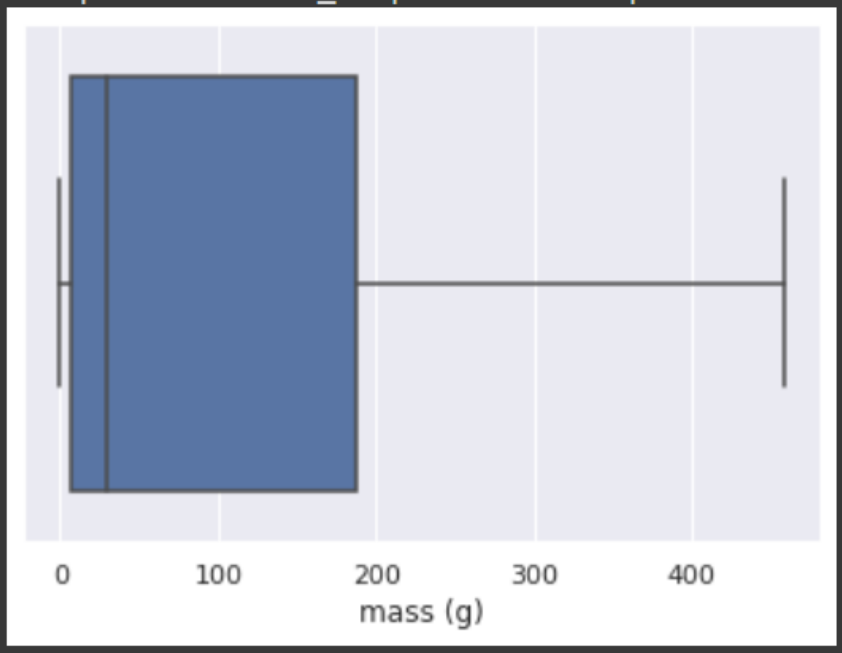
The box in the boxplot lies in the range of 1st quartile (25%) and 3rd quartile (75%). So, the actual range of the box for the 'mass (g)' column would be from 6.63e+00 to 1.87e+02, or 6.63 to 187.29, which is quite small compared to the maximum value of 6e+07 or 6 * 10^7, which is an outlier. In fact, almost all of the points which can be seen on the above boxplot are outliers.

To hide the outliers, we can pass showfliers=False to the .boxplot() method:

```
sns.boxplot(x=df['mass (g)'], showfliers=False)
```

The boxplot is now shown without any outliers, and thus is scaled down:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcb89c0f690>
```



Alternatively, we can completely get rid of the outliers using the IQR (InterQuartile Range) score technique:

```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

Here we are calculating the difference between the 3rd quantile (Q3) and the 1st quantile (Q1) for each column:

```
id           29056.00000
mass (g)       180.66000
year            16.00000
latitude        76.71667
longitude      157.16667
```

```
dtype: float64
```

Based on this, we delete any rows which contain values which are either smaller than Q1 – 1.5 * IQR or greater than Q3 + 1.5 * IQR:

```
df = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1
)]
df.shape
```

```
(31738, 10)
```

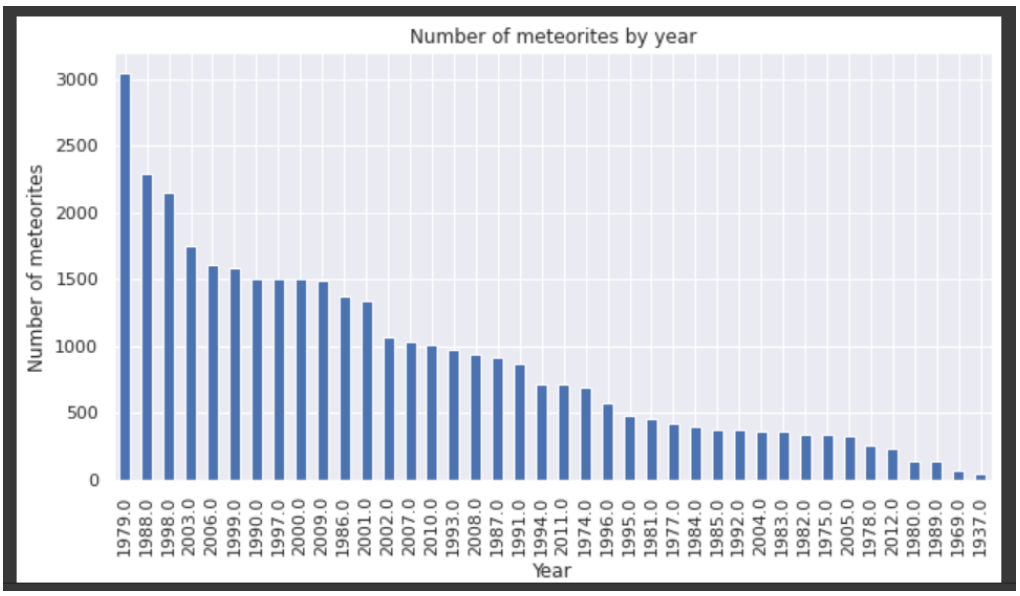
So, there were about 7000 rows which were outliers.

Histograms:

We can find out which year had the most meteorite landings compared to other years with the help of a histogram:

```
df.year.value_counts().nlargest(40).plot(kind='bar', figsize=(10, 5))
plt.title("Number of meteorites by year")
plt.ylabel('Number of meteorites')
plt.xlabel('Year');
```

Here we are taking the top 40 years with the most meteorite landings and plotting them on a histogram using matplotlib:

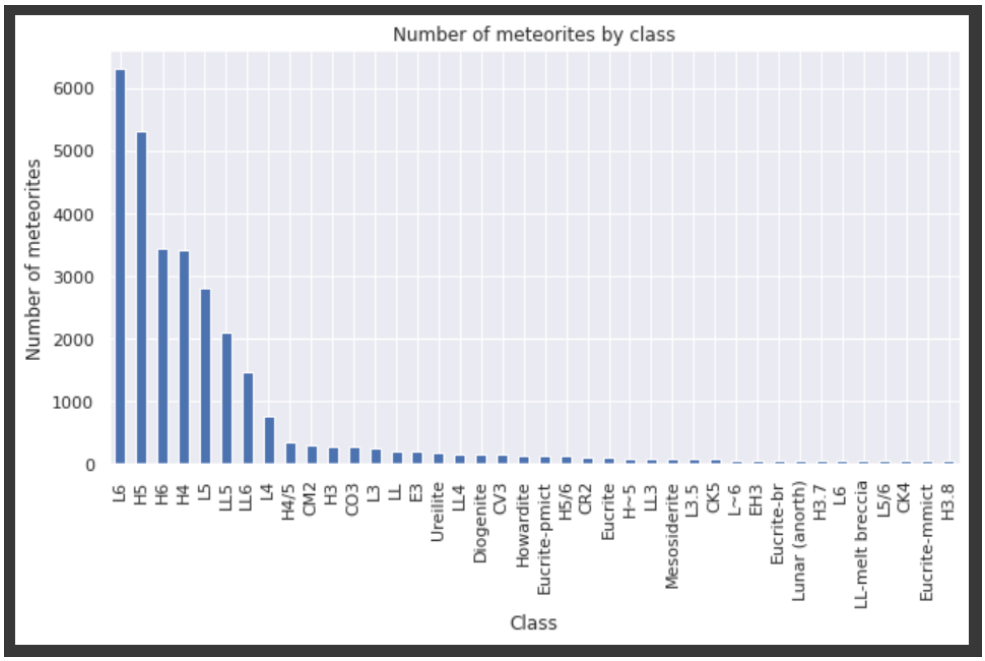


As you can see, 1979, with over 3000 meteorites had the highest number of meteorite landings by far. The most recent year that can be seen on the graph is 2012, which had around 200 meteorite landings.

```
df['class'].value_counts().nlargest(40).plot(kind='bar', figsize=(10, 5
))
```

```
plt.title("Number of meteorites by class")
plt.ylabel('Number of meteorites')
plt.xlabel('Class');
```

We can also sort the number of meteorites by the ‘class’ column and plot it using histogram:



As you can see, meteorites of type L6 have the highest frequency (over 6000).

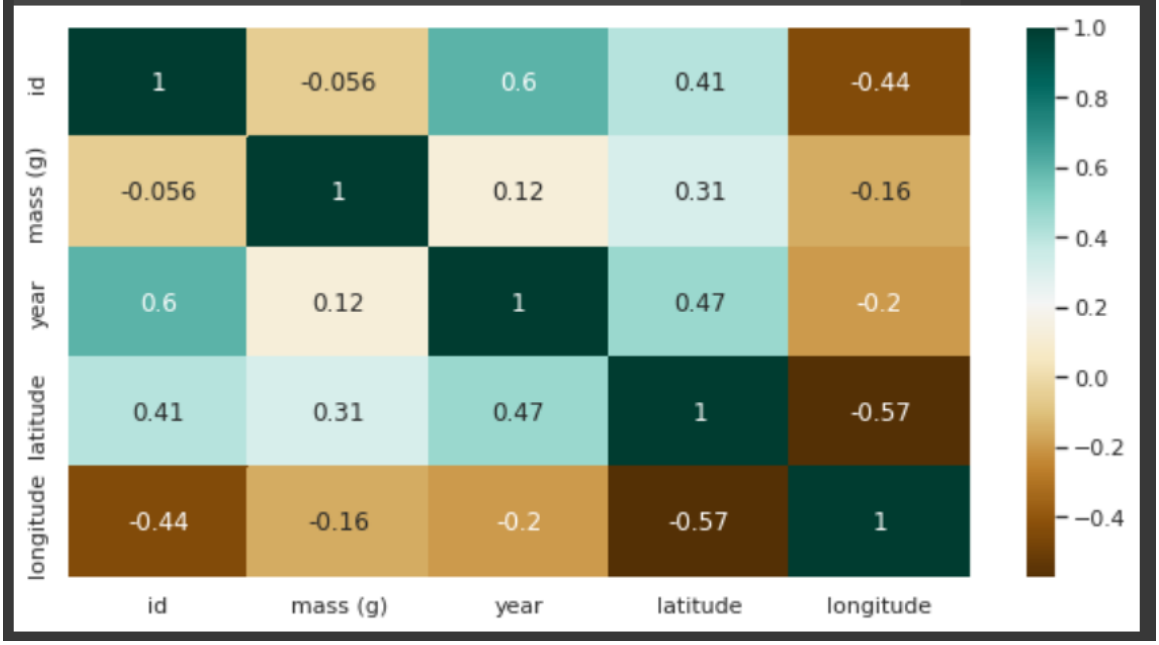
Heat Map:

We can find the dependent variables using heat map.

```
plt.figure(figsize=(10,5))
c= df.corr()
sns.heatmap(c,cmap="BrBG",annot=True)
c
```

It first finds the correlation between any two variables and then plots it along with a colour theme using the seaborn library:

	id	mass (g)	year	latitude	longitude
id	1.000000	-0.056198	0.602697	0.407276	-0.441195
mass (g)	-0.056198	1.000000	0.124319	0.312772	-0.155516
year	0.602697	0.124319	1.000000	0.465574	-0.195780
latitude	0.407276	0.312772	0.465574	1.000000	-0.573616
longitude	-0.441195	-0.155516	-0.195780	-0.573616	1.000000



No variable is strongly dependent on another variable.

Conclusion:

- The dataset contained 45716 rows, out of which 7315 contained null values and 6377 were outliers, making the final dataset of 31738 rows and 10 columns.
- I learned that outliers can be hidden with showfliers=False or can be removed using the IQR techinique.
- The year 1979 saw over 3000 meteorite landings, thus becoming the year with the highest number of meteorite landings.
- Meteorites of type L6 are the most common, with over 6000 meteorites to its name.
- There exists no particular relation between the mass of a meteorite and its other attributes.

References: Meteorite Landings dataset