**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.

**Implementation:**

Dataset used: [Meteorite Landings dataset](https://data.nasa.gov/Space-Science/Meteorite-Landings/gh4g-9sfh)

**Importing the required libraries:**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

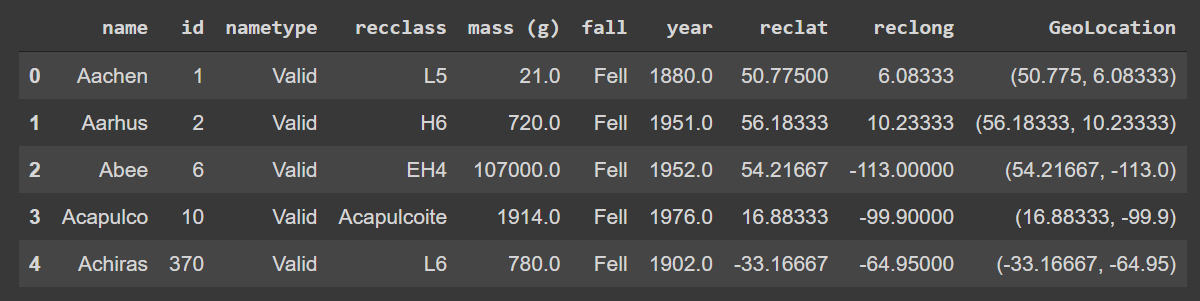
sns.set(color\_codes=True)

**Loading the data into the dataframe:**

df = pd.read\_csv("Meteorite\_Landings.csv")

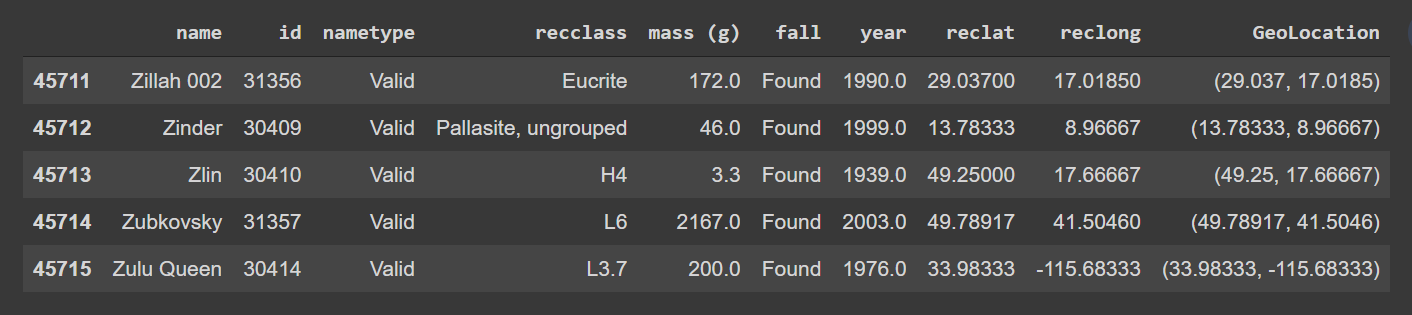
df.head()

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



df.tail()

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

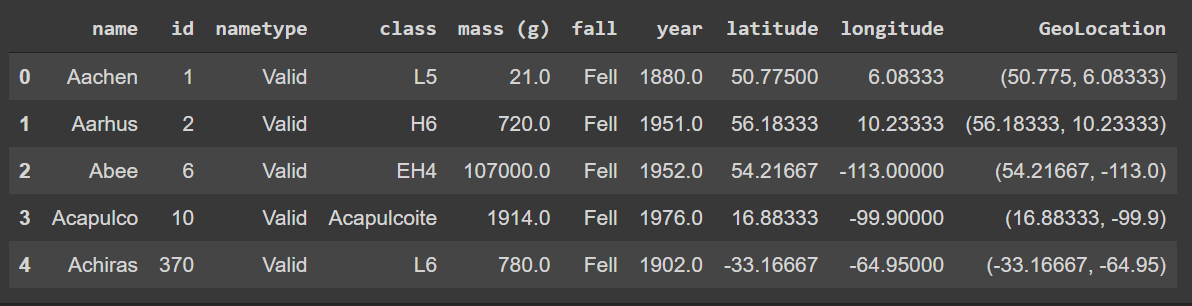


Under NameType, 'valid' is for most meteorites and 'relict' are for objects that were once meteorites but are now highly altered by weathering on Earth. The ‘recclass’ column contains the type of meteors. The ‘fall’ column contains either of two values – ‘Fell’ or ‘Found’. Found basically means the meteorite 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", "reclong": "longitude"})

df.head()



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

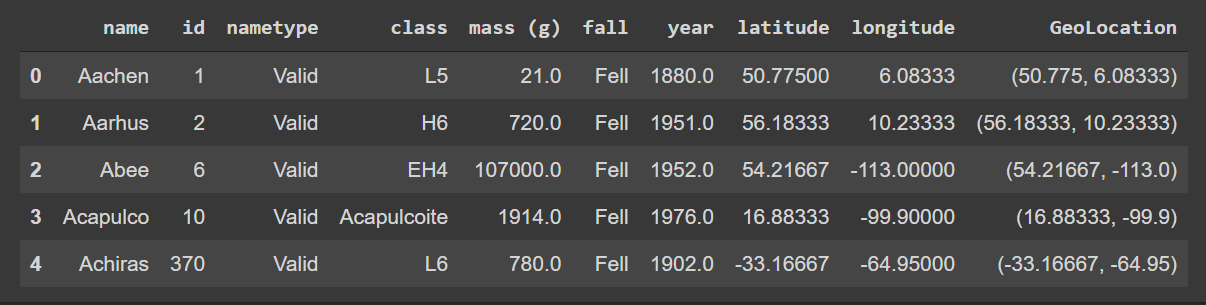
GeoLocation 38401

dtype: int64

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

df = df.drop\_duplicates()

df.head()



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)'])

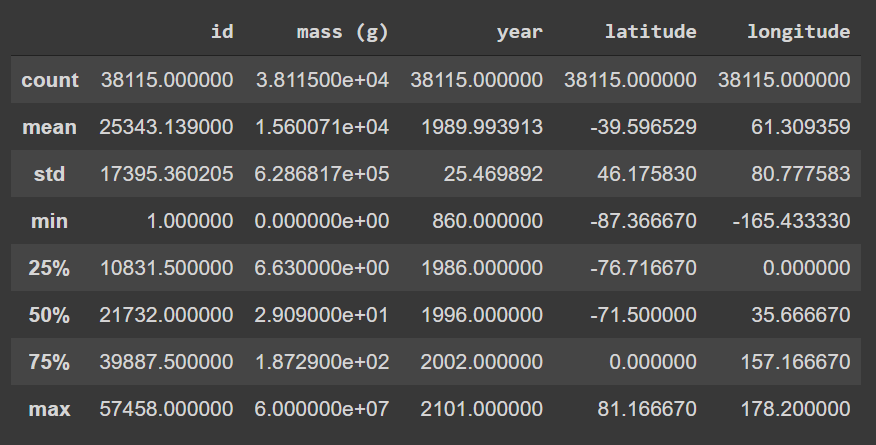
This will show a box plot for the ‘mass (g) column:



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 \* 107. 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()

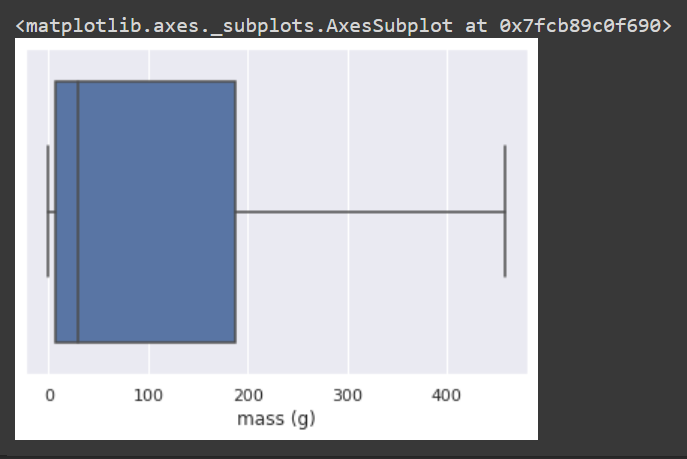


The box in the boxplot lies in the range of 1st quartile (25%) and 3rd quartile (75%). So, the actual range of the box for the ‘mass (g)’ column would be from 6.63e+00 to 1.87e+02, or 6.63 to 187.29, which is quite small compared to the maximum value of 6e+07 or 6 \* 107, which is an outlier. In fact, 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:



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:

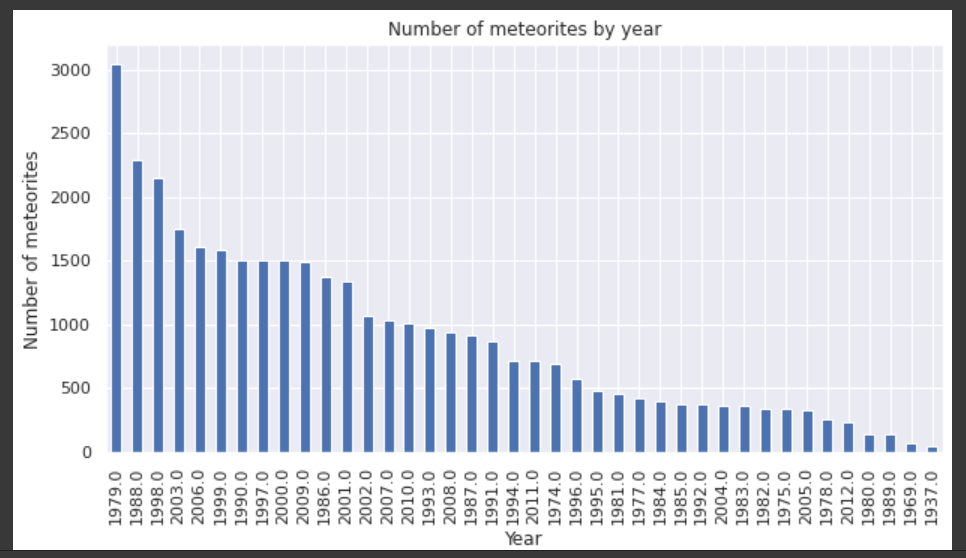
df.year.value\_counts().nlargest(40).plot(kind='bar', figsize=(10, 5))

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.

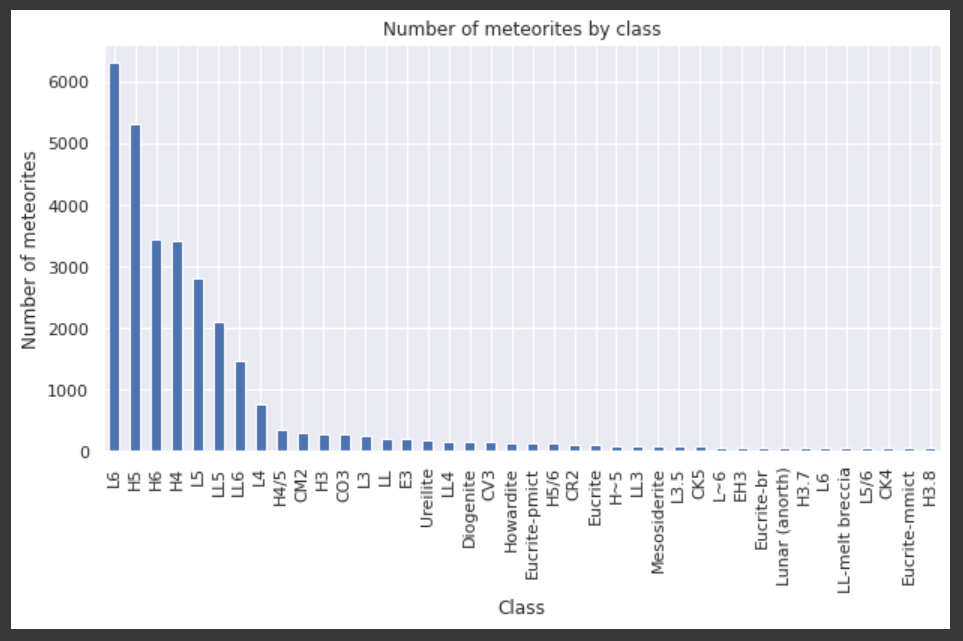
df['class'].value\_counts().nlargest(40).plot(kind='bar', figsize=(10, 5))

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.

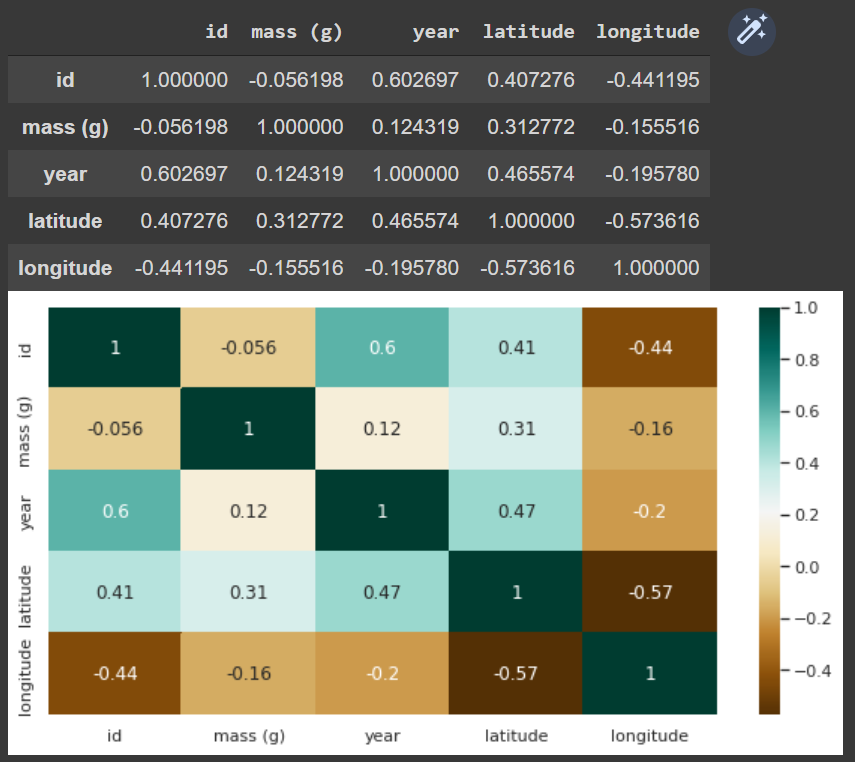
plt.figure(figsize=(10,5))

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:



No variable is strongly dependent on another variable.

**Conclusion:** I learnt about the concept of Exploratory Data Analysis. I also learnt how to perform EDA on a dataset using various Python libraries like pandas, seaborn, matplotlib which allowed me to plot boxplot, histogram, etc.

**References:** [Meteorite Landings dataset](https://data.nasa.gov/Space-Science/Meteorite-Landings/gh4g-9sfh)