# Exploratory Data Analysis on Iris Dataset

**Introduction**

**Exploratory Data Analysis (EDA)**is a technique to analyze data using some visual Techniques. With this technique, we can get detailed information about the statistical summary of the data. We will also be able to deal with the duplicates values, outliers, and also see some trends or patterns present in the dataset.

Now let’s see a brief about the Iris dataset

## **Iris Dataset**

Iris Dataset is considered as the Hello World for data science. It contains five columns namely – Petal Length, Petal Width, Sepal Length, Sepal Width, and Species Type. Iris is a flowering plant, the researchers have measured various features of the different iris flowers and recorded them digitally

Example code:-

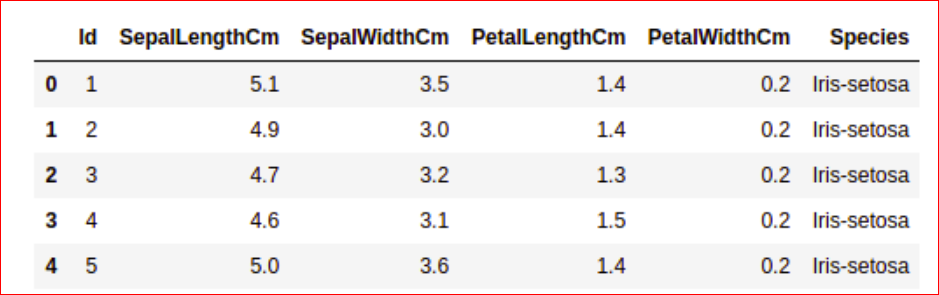
**import** pandas as pd

# Reading the CSV file

df **=** pd.read\_csv("Iris.csv")

# Printing top 5 rows

df.head()



## Getting Information about the Dataset

We will use the shape parameter to get the shape of the dataset.

df.shape

**Output:**

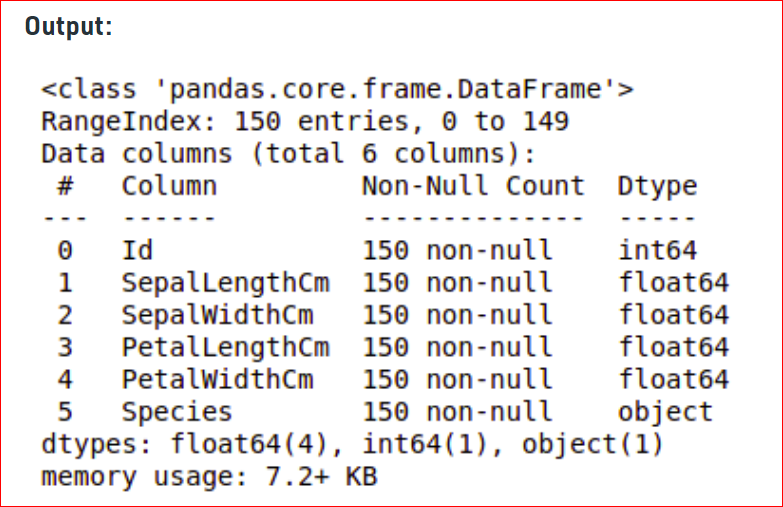
(150, 6)

We can see that the dataframe contains 6 columns and 150 rows.

Now, let’s also the columns and their data types. For this, we will use the [info()](https://www.geeksforgeeks.org/python-pandas-dataframe-info/) method.

**Example:**

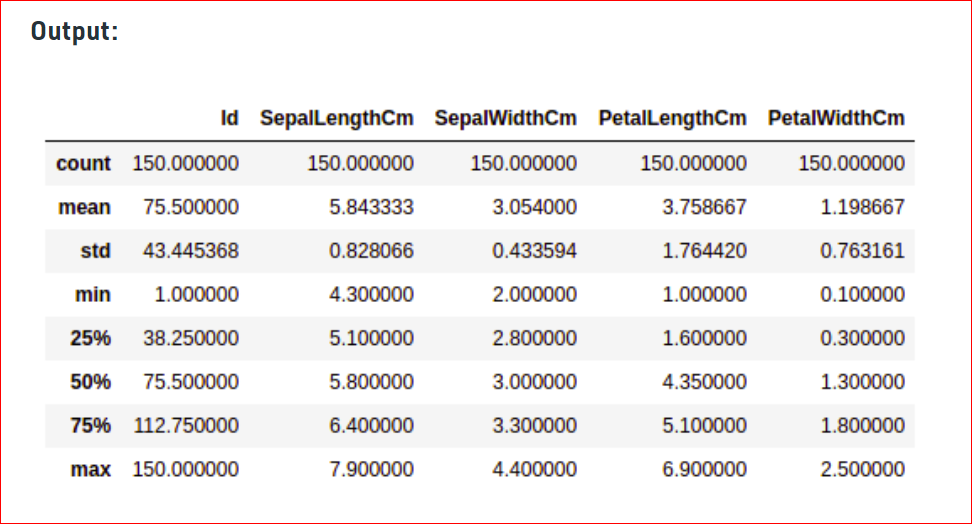
|  |
| --- |
| df.info() |



We can see that only one column has categorical data and all the other columns are of the numeric type with non-Null entries.

Let’s get a quick statistical summary of the dataset using the [**describe()**](https://www.geeksforgeeks.org/python-pandas-dataframe-describe-method/) method. The describe() function applies basic statistical computations on the dataset like extreme values, count of data points standard deviation, etc. Any missing value or NaN value is automatically skipped. describe() function gives a good picture of the distribution of data

|  |
| --- |
| df.describe() # it return 8 data |



## Checking Missing Values

We will check if our data contains any missing values or not. Missing values can occur when no information is provided for one or more items or for a whole unit. We will use the [isnull()](https://www.geeksforgeeks.org/python-pandas-isnull-and-notnull/) method.

**Example:**

|  |
| --- |
| df.isnull().sum() |

We can see that no column as any missing value.

**Note:**For more information, refer [Working with Missing Data in Pandas](https://www.geeksforgeeks.org/working-with-missing-data-in-pandas/).

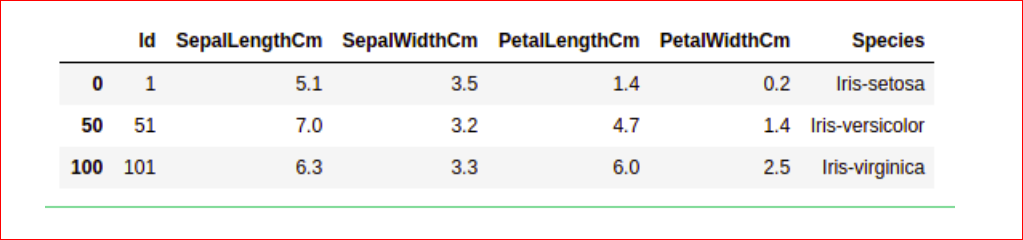
‘

## Checking Duplicates

Let’s see if our dataset contains any duplicates or not. Pandas [drop\_duplicates()](https://www.geeksforgeeks.org/python-pandas-dataframe-drop_duplicates/) method helps in removing duplicates from the data frame.

Example:

|  |
| --- |
| data **=** df.drop\_duplicates(subset **=**"Species",)  data |



We can see that there are only three unique species. Let’s see if the dataset is balanced or not i.e. all the species contain equal amounts of rows or not. We will use the [Series.value\_counts()](https://www.geeksforgeeks.org/python-pandas-series-value_counts/) function. This function returns a Series containing counts of unique values

**Example:**

|  |
| --- |
| df.value\_counts("Species") |

We can see that all the species contain an equal amount of rows, so we should not delete any entries.

## Data Visualization

### Visualizing the target column

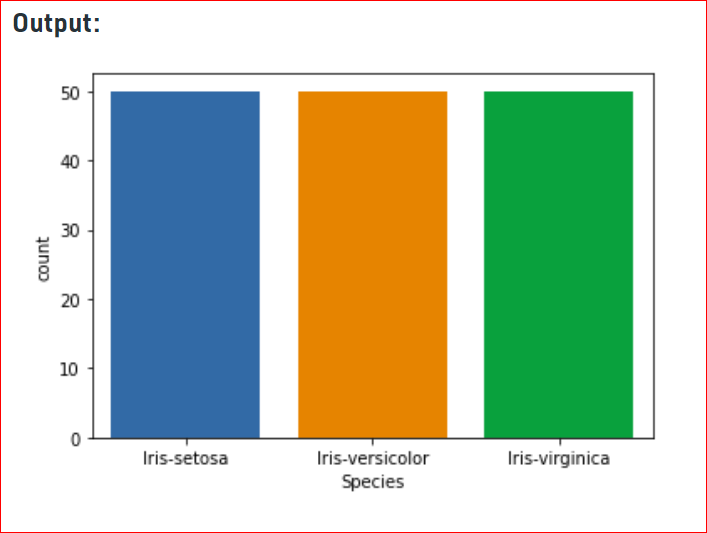
Our target column will be the Species column because at the end we will need the result according to the species only. Let’s see a countplot for species.

***Note:****We will use Matplotlib and Seaborn library for the data visualization. If you want to know about these modules refer to the articles –*

* [*Matplotlib Tutorial*](https://www.geeksforgeeks.org/matplotlib-tutorial/)
* [*Python Seaborn Tutorial*](https://www.geeksforgeeks.org/python-seaborn-tutorial/)

**Example:**

|  |
| --- |
| # importing packages  **import** seaborn as sns  **import** matplotlib.pyplot as plt      sns.countplot(x**=**'Species', data**=**df, )  plt.show() |



### Relation between variables

We will see the relationship between the sepal length and sepal width and also between petal length and petal width.

**Example 1:**Comparing Sepal Length and Sepal Width

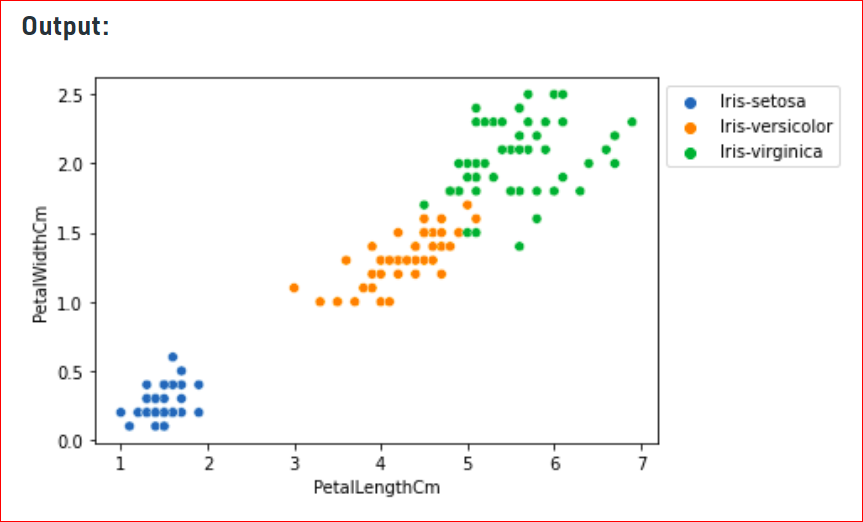
|  |
| --- |
| # importing packages  **import** seaborn as sns  **import** matplotlib.pyplot as plt      sns.scatterplot(x**=**'SepalLengthCm', y**=**'SepalWidthCm',                  hue**=**'Species', data**=**df, )    # Placing Legend outside the Figure  plt.legend(bbox\_to\_anchor**=**(1, 1), loc**=**2)    plt.show() |

From the above plot, we can infer that –

* Species Setosa has smaller sepal lengths but larger sepal widths.
* Versicolor Species lies in the middle of the other two species in terms of sepal length and width
* Species Virginica has larger sepal lengths but smaller sepal widths.

**Example 2:**Comparing Petal Length and Petal Width

|  |
| --- |
| # importing packages  **import** seaborn as sns  **import** matplotlib.pyplot as plt      sns.scatterplot(x**=**'PetalLengthCm', y**=**'PetalWidthCm',                  hue**=**'Species', data**=**df, )    # Placing Legend outside the Figure  plt.legend(bbox\_to\_anchor**=**(1, 1), loc**=**2)    plt.show() |



From the above plot, we can infer that –

* Species Setosa has smaller petal lengths and widths.
* Versicolor Species lies in the middle of the other two species in terms of petal length and width
* Species Virginica has the largest of petal lengths and widths.

Let’s plot all the column’s relationships using a pairplot. It can be used for multivariate analysis.

**Example:**

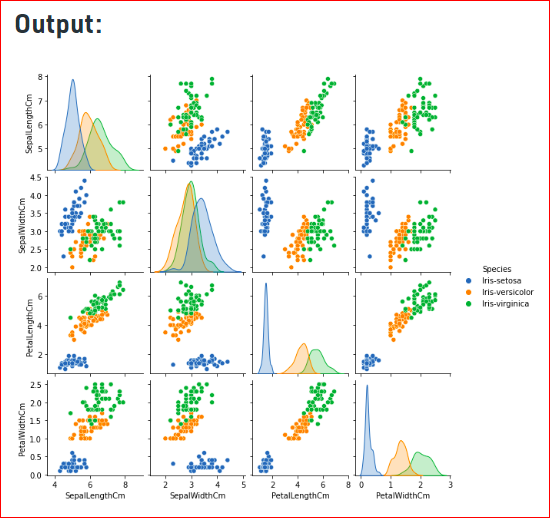
# importing packages

**import** seaborn as sns

**import** matplotlib.pyplot as plt

sns.pairplot(df.drop(['Id'], axis **=** 1),

             hue**=**'Species', height**=**2)



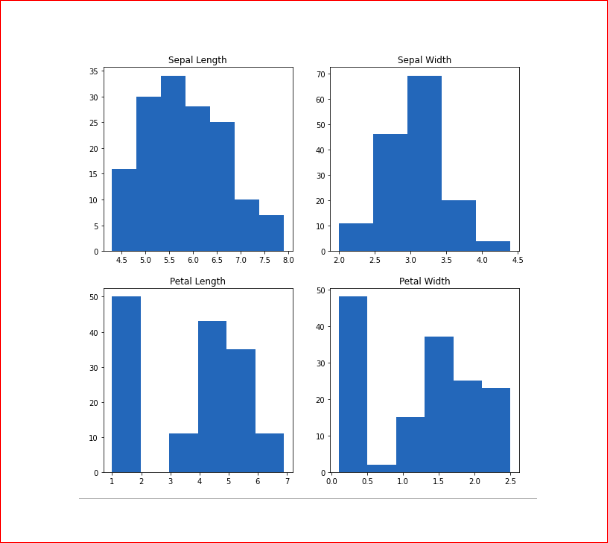
We can see many types of relationships from this plot such as the species Seotsa has the smallest of petals widths and lengths. It also has the smallest sepal length but larger sepal widths. Such information can be gathered about any other species.

### Histograms

Histograms allow seeing the distribution of data for various columns. It can be used for uni as well as bi-variate analysis.

**Example**

|  |
| --- |
| # importing packages  **import** seaborn as sns  **import** matplotlib.pyplot as plt      fig, axes **=** plt.subplots(2, 2, figsize**=**(10,10))    axes[0,0].set\_title("Sepal Length")  axes[0,0].hist(df['SepalLengthCm'], bins**=**7)    axes[0,1].set\_title("Sepal Width")  axes[0,1].hist(df['SepalWidthCm'], bins**=**5);    axes[1,0].set\_title("Petal Length")  axes[1,0].hist(df['PetalLengthCm'], bins**=**6);    axes[1,1].set\_title("Petal Width")  axes[1,1].hist(df['PetalWidthCm'], bins**=**6); |



From the above plot, we can see that –

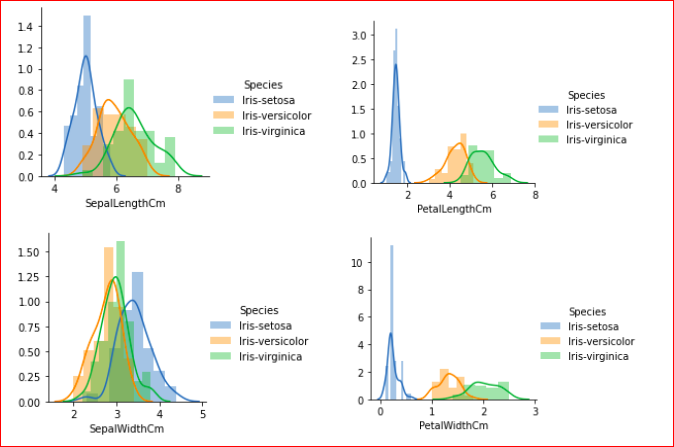
* The highest frequency of the sepal length is between 30 and 35 which is between 5.5 and 6
* The highest frequency of the sepal Width is around 70 which is between 3.0 and 3.5
* The highest frequency of the petal length is around 50 which is between 1 and 2
* The highest frequency of the petal width is between 40 and 50 which is between 0.0 and 0.5

### Histograms with Distplot Plot

Distplot is used basically for the univariant set of observations and visualizes it through a histogram i.e. only one observation and hence we choose one particular column of the dataset.

|  |
| --- |
| **import** seaborn as sns  **import** matplotlib.pyplot as plt    plot **=** sns.FacetGrid(df, hue**=**"Species")  plot.map(sns.distplot, "SepalLengthCm").add\_legend()    plot **=** sns.FacetGrid(df, hue**=**"Species")  plot.map(sns.distplot, "SepalWidthCm").add\_legend()    plot **=** sns.FacetGrid(df, hue**=**"Species")  plot.map(sns.distplot, "PetalLengthCm").add\_legend()    plot **=** sns.FacetGrid(df, hue**=**"Species")  plot.map(sns.distplot, "PetalWidthCm").add\_legend()    plt.show() |

**Output:**



From the above plots, we can see that –

* In the case of Sepal Length, there is a huge amount of overlapping.
* In the case of Sepal Width also, there is a huge amount of overlapping.
* In the case of Petal Length, there is a very little amount of overlapping.
* In the case of Petal Width also, there is a very little amount of overlapping.

So we can use Petal Length and Petal Width as the classification feature.

### Handling Correlation

Pandas [dataframe.corr()](https://www.geeksforgeeks.org/python-pandas-dataframe-corr/) is used to find the pairwise correlation of all columns in the dataframe. Any NA values are automatically excluded. For any non-numeric data type columns in the dataframe it is ignored.

**Example:**

|  |
| --- |
| data.corr(method**=**'pearson') |

### Heatmaps

The heatmap is a data visualization technique that is used to analyze the dataset as colors in two dimensions. Basically, it shows a correlation between all numerical variables in the dataset. In simpler terms, we can plot the above-found correlation using the heatmaps.

**Example:**

# importing packages

**import** seaborn as sns

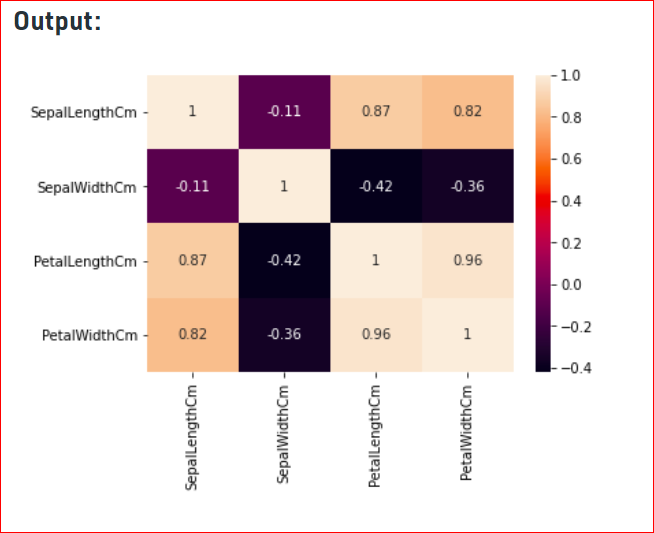
**import** matplotlib.pyplot as plt

sns.heatmap(df.corr(method**=**'pearson').drop(

  ['Id'], axis**=**1).drop(['Id'], axis**=**0),

            annot **=** True);

plt.show()



From the above graph, we can see that –

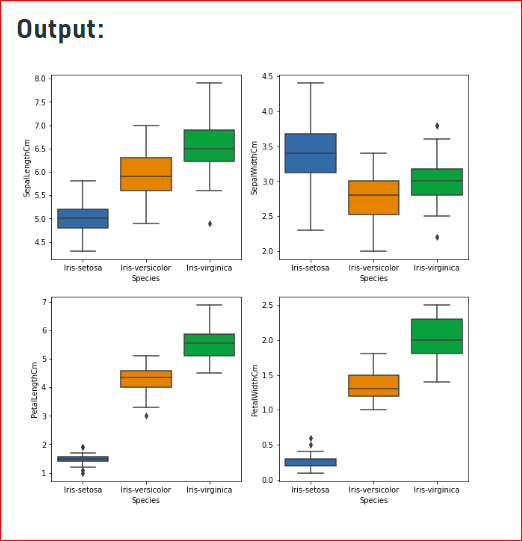
* Petal width and petal length have high correlations.
* Petal length and sepal width have good correlations.
* Petal Width and Sepal length have good correlations.

### Box Plots

We can use boxplots to see how the categorical value os distributed with other numerical values.

**Example:**

|  |
| --- |
| # importing packages  **import** seaborn as sns  **import** matplotlib.pyplot as plt    **def** graph(y):      sns.boxplot(x**=**"Species", y**=**y, data**=**df)    plt.figure(figsize**=**(10,10))    # Adding the subplot at the specified  # grid position  plt.subplot(221)  graph('SepalLengthCm')    plt.subplot(222)  graph('SepalWidthCm')    plt.subplot(223)  graph('PetalLengthCm')    plt.subplot(224)  graph('PetalWidthCm')    plt.show() |

’

From the above graph, we can see that –

* Species Setosa has the smallest features and less distributed with some outliers.
* Species Versicolor has the average features.
* Species Virginica has the highest features

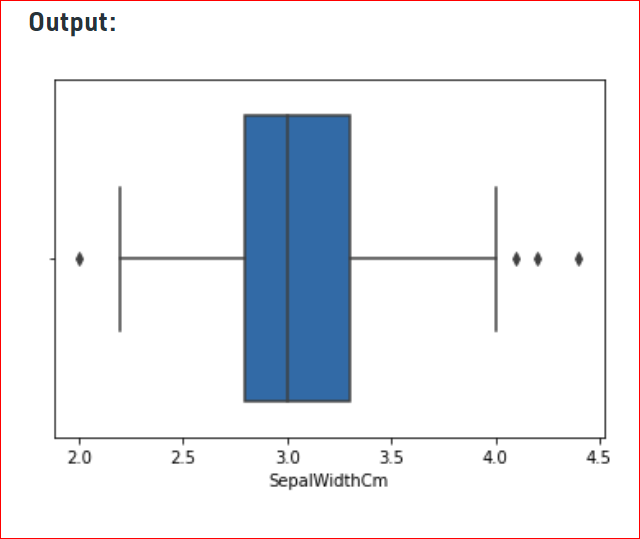
## Handling Outliers

An Outlier is a data-item/object that deviates significantly from the rest of the (so-called normal)objects. They can be caused by measurement or execution errors. The analysis for outlier detection is referred to as outlier mining. There are many ways to detect the outliers, and the removal process is the data frame same as removing a data item from the panda’s dataframe.

Let’s consider the iris dataset and let’s plot the boxplot for the SepalWidthCm column.

**Example:**

|  |
| --- |
| # importing packages  **import** seaborn as sns  **import** matplotlib.pyplot as plt    # Load the dataset  df **=** pd.read\_csv('Iris.csv')    sns.boxplot(x**=**'SepalWidthCm', data**=**df) |



In the above graph, the values above 4 and below 2 are acting as outliers.

### Removing Outliers

For removing the outlier, one must follow the same process of removing an entry from the dataset using its exact position in the dataset because in all the above methods of detecting the outliers end result is the list of all those data items that satisfy the outlier definition according to the method used.

**Example:**We will detect the outliers using [IQR](https://www.geeksforgeeks.org/interquartile-range-iqr/) and then we will remove them. We will also draw the boxplot to see if the outliers are removed or not.

**import** sklearn

**from** sklearn.datasets **import** load\_boston

**import** pandas as pd

**import** seaborn as sns

# Load the dataset

df **=** pd.read\_csv('Iris.csv')

# IQR

Q1 **=** np.percentile(df['SepalWidthCm'], 25,

                interpolation **=** 'midpoint')

Q3 **=** np.percentile(df['SepalWidthCm'], 75,

                interpolation **=** 'midpoint')

IQR **=** Q3 **-** Q1

print("Old Shape: ", df.shape)

# Upper bound

upper **=** np.where(df['SepalWidthCm'] >**=** (Q3**+**1.5**\***IQR))

# Lower bound

lower **=** np.where(df['SepalWidthCm'] <**=** (Q1**-**1.5**\***IQR))

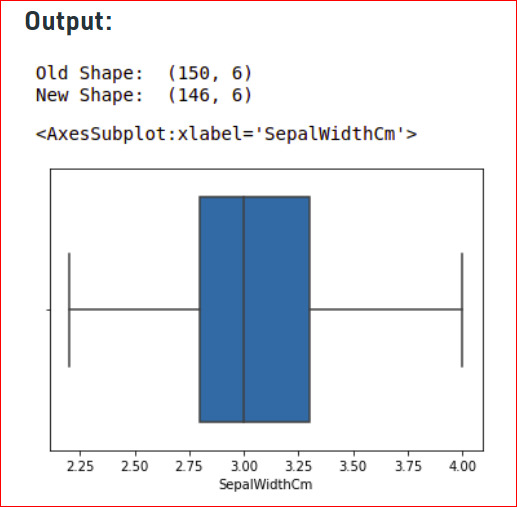
# Removing the Outliers

df.drop(upper[0], inplace **=** True)

df.drop(lower[0], inplace **=** True)

**print**("New Shape: ", df.shape)

sns.boxplot(x**=**'SepalWidthCm', data**=**df)



# Detect and Remove the Outliers using Python:-

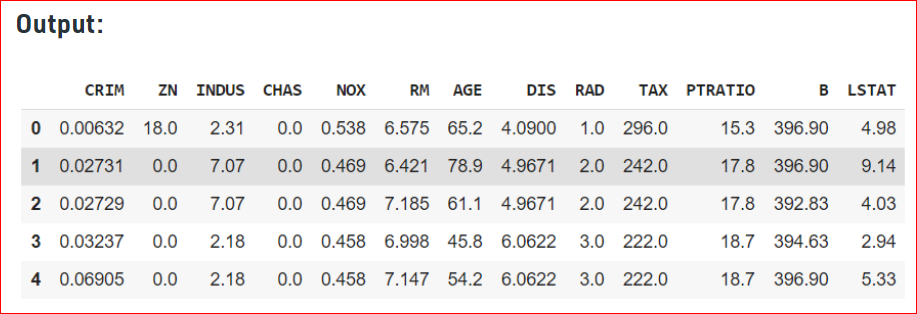
An Outlier is a data-item/object that deviates significantly from the rest of the (so-called normal)objects. They can be caused by measurement or execution errors. The analysis for outlier detection is referred to as outlier mining. There are many ways to detect the outliers, and the removal process is the data frame same as removing a data item from the panda’s data frame.

Here pandas data frame is used for a more realistic approach as in real-world project need to detect the outliers arouse during the data analysis step, the same approach can be used on lists and series-type objects.

### Dataset:

Dataset used is Boston Housing dataset as it is preloaded in the sklearn library.

|  |
| --- |
| # Importing  **import** sklearn  **from** sklearn.datasets **import** load\_boston  **import** pandas as pd  **import** matplotlib.pyplot as plt    # Load the dataset  bos\_hou **=** load\_boston()    # Create the dataframe  column\_name **=** bos\_hou.feature\_names  df\_boston **=** pd.DataFrame(bos\_hou.data)  df\_boston.columns **=** column\_name  df\_boston.head() |



## **Detecting the outliers**

Outliers can be detected using visualization, implementing mathematical formulas on the dataset, or using the statistical approach. All of these are discussed below.

### 1. Visualization

**Example 1: Using**[**Box Plot**](https://www.geeksforgeeks.org/box-plot-visualization-with-pandas-and-seaborn/)

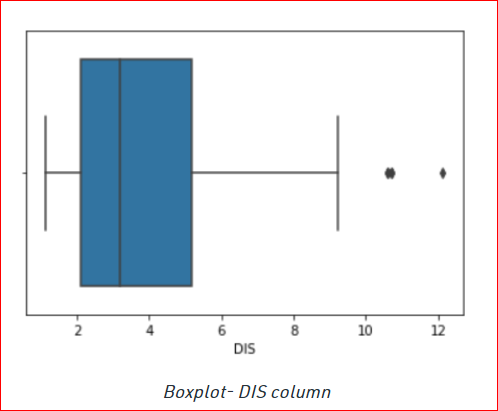
It captures the summary of the data effectively and efficiently with only a simple box and whiskers. Boxplot summarizes sample data using 25th, 50th, and 75th percentiles. One can just get insights(quartiles, median, and outliers) into the dataset by just looking at its boxplot.

# Box Plot

**import** seaborn as sns

sns.boxplot(df\_boston['DIS'])

Output:-



In the above graph, can clearly see that values above 10 are acting as the outliers.

|  |
| --- |
| # Position of the Outlier  print(np.where(df\_boston['DIS']>10)) |



*Outlier’s Index*

**Example 2: Using [ScatterPlot](https://www.geeksforgeeks.org/matplotlib-pyplot-scatter-in-python/).**

It is used when you have paired numerical data, or when your dependent variable has multiple values for each reading independent variable, or when trying to determine the relationship between the two variables. In the process of utilizing the scatter plot, one can also use it for outlier detection.

To plot the scatter plot one requires two variables that are somehow related to each other. So here, ‘Proportion of non-retail business acres per town’ and ‘Full-value property-tax rate per $10,000’ are used whose column names are “INDUS” and “TAX” respectively.

Scatter plot

fig, ax **=** plt.subplots(figsize **=** (18,10))

ax.scatter(df\_boston['INDUS'], df\_boston['TAX'])

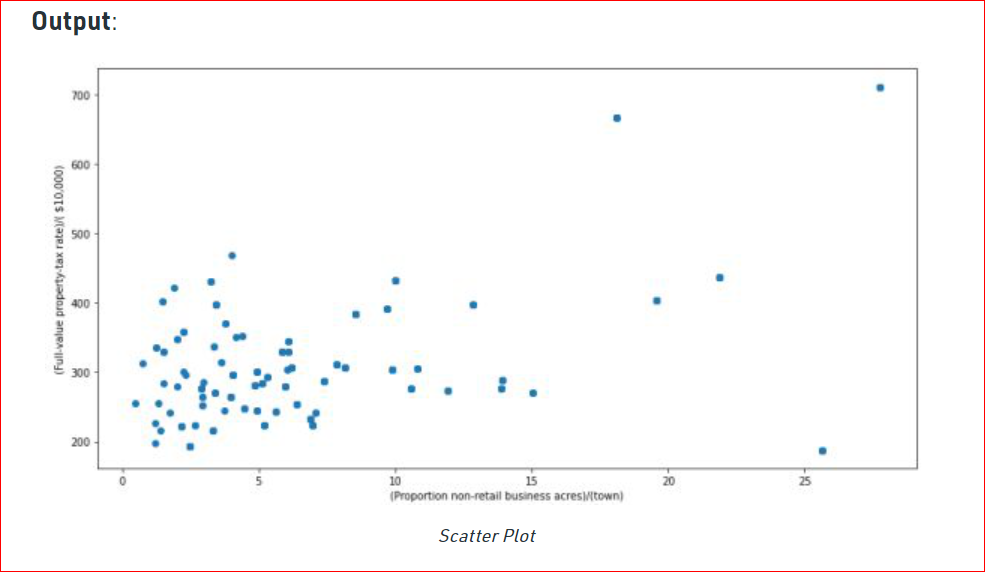
# x-axis label

ax.set\_xlabel('(Proportion non-retail business acres)/(town)')

# y-axis label

ax.set\_ylabel('(Full-value property-tax rate)/( $10,000)')

plt.show()



Looking at the graph can summarize that most of the data points are in the bottom left corner of the graph but there are few points that are exactly;y opposite that is the top right corner of the graph. Those points in the top right corner can be regarded as Outliers.

Using approximation can say all those data points that are x>20 and y>600 are outliers. The following code can fetch the exact position of all those points that satisfy these conditions.

# Position of the Outlier

print(np.where((df\_boston['INDUS']>20) & (df\_boston['TAX']>600)))

**Output**:



### **2. Z-score**

[Z- Score](https://www.geeksforgeeks.org/z-score-in-statistics/) is also called a standard score. This value/score helps to understand that how far is the data point from the mean. And after setting up a threshold value one can utilize z score values of data points to define the outliers.

*Zscore = (data\_point -mean) / std. deviation*

|  |
| --- |
| # Z score  **from** scipy **import** stats  **import** numpy as np    z **=** np.abs(stats.zscore(df\_boston['DIS']))  print(z) |

The above output is just a snapshot of part of the data; the actual length of the list(z) is 506 that is the number of rows. It prints the z-score values of each data item of the column

Now to define an outlier threshold value is chosen which is generally 3.0. As 99.7% of the data points lie between +/- 3 standard deviation (using Gaussian Distribution approach).

|  |
| --- |
| threshold **=** 3    # Position of the outlier  print(np.where(z > 3)) |

**Output:**



*Outlier’s Index*

### 3. IQR (Inter Quartile Range)

[IQR (Inter Quartile Range)](https://www.geeksforgeeks.org/interquartile-range-iqr/) Inter Quartile Range approach to finding the outliers is the most commonly used and most trusted approach used in the research field.

*IQR = Quartile3 – Quartile1*

|  |
| --- |
| # IQR  Q1 **=** np.percentile(df\_boston['DIS'], 25,                     interpolation **=** 'midpoint')    Q3 **=** np.percentile(df\_boston['DIS'], 75,                     interpolation **=** 'midpoint')  IQR **=** Q3 **-** Q1 |

**Output:**



To define the outlier base value is defined above and below datasets normal range namely Upper and Lower bounds, define the upper and the lower bound (1.5\*IQR value is considered) :

*upper = Q3 +1.5\*IQR*

*lower = Q1 – 1.5\*IQR*

In the above formula as according to statistics, the 0.5 scale-up of IQR (new\_IQR = IQR + 0.5\*IQR) is taken, to consider all the data between 2.7 standard deviations in the Gaussian Distribution.

# Above Upper bound

upper **=** df\_boston['DIS'] >**=** (Q3**+**1.5**\***IQR)

**print**("Upper bound:",upper)

**print**(np.where(upper))

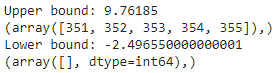
# Below Lower bound

lower **=** df\_boston['DIS'] <**=** (Q1**-**1.5**\***IQR)

print("Lower bound:", lower)

print(np.where(lower))

**Output:**



*Defined bounds and outlier’s index respective to the bounds*

## Removing the outliers

For removing the outlier, one must follow the same process of removing an entry from the dataset using its exact position in the dataset because in all the above methods of detecting the outliers end result is the list of all those data items that satisfy the outlier definition according to the method used.

References: [How to delete exactly one row in python?](https://www.geeksforgeeks.org/how-to-delete-only-one-row-in-csv-with-python/)

*dataframe.drop( row\_index, inplace = True*

The above code can be used to drop a row from the dataset given the row\_indexes to be dropped. Inplace =True is used to tell python to make the required change in the original dataset. row\_index can be only one value or list of values or NumPy array but it must be one dimensional.

**Example:**

*df\_boston.drop(lists[0],inplace = True)*

**Full Code**: Detecting the outliers using IQR and removing them.

|  |
| --- |
| # Importing  **import** sklearn  **from** sklearn.datasets **import** load\_boston  **import** pandas as pd    # Load the dataset  bos\_hou **=** load\_boston()    # Create the dataframe  column\_name **=** bos\_hou.feature\_names  df\_boston **=** pd.DataFrame(bos\_hou.data)  df\_boston.columns **=** column\_name  df\_boston.head()    ''' Detection '''  # IQR  Q1 **=** np.percentile(df\_boston['DIS'], 25,                     interpolation **=** 'midpoint')    Q3 **=** np.percentile(df\_boston['DIS'], 75,                     interpolation **=** 'midpoint')  IQR **=** Q3 **-** Q1    **print**("Old Shape: ", df\_boston.shape)    # Upper bound  upper **=** np.where(df\_boston['DIS'] >**=** (Q3**+**1.5**\***IQR))  # Lower bound  lower **=** np.where(df\_boston['DIS'] <**=** (Q1**-**1.5**\***IQR))    ''' Removing the Outliers '''  df\_boston.drop(upper[0], inplace **=** True)  df\_boston.drop(lower[0], inplace **=** True)    print("New Shape: ", df\_boston.shape) |

**Output**:

