**Data Analysis With Python**

**Data Analysis** is the technique of collecting, transforming, and organizing data to make future predictions and informed data-driven decisions. It also helps to find possible solutions for a business problem. There are six steps for Data Analysis. They are:

* Ask or Specify Data Requirements
* Prepare or Collect Data
* Clean and Process
* Analyze
* Share
* Act or Report

*Data Analysis with Python*

**Note:** To know more about these steps refer to our [Six Steps of Data Analysis Process](https://www.geeksforgeeks.org/six-steps-of-data-analysis-process/) tutorial.

**Analyzing Numerical Data with NumPy**

[NumPy](https://www.geeksforgeeks.org/python-numpy/) is an array processing package in Python and provides a high-performance multidimensional array object and tools for working with these arrays. It is the fundamental package for scientific computing with Python.

**Arrays in NumPy**

[NumPy Array](https://www.geeksforgeeks.org/basics-of-numpy-arrays/) is a table of elements (usually numbers), all of the same types, indexed by a tuple of positive integers. In Numpy, the number of dimensions of the array is called the rank of the array. A tuple of integers giving the size of the array along each dimension is known as the shape of the array.

**Creating NumPy Array**

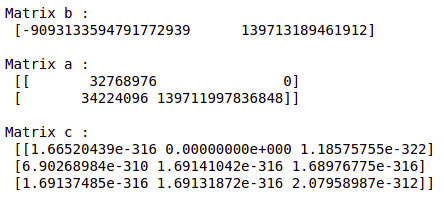
NumPy arrays can be created in multiple ways, with various ranks. It can also be created with the use of different data types like lists, tuples, etc. The type of the resultant array is deduced from the type of elements in the sequences. NumPy offers several functions to create arrays with initial placeholder content. These minimize the necessity of growing arrays, an expensive operation.

**Create Array using**[**numpy.empty(shape, dtype=float, order=’C’)**](https://www.geeksforgeeks.org/numpy-empty-python/)

* Python3

|  |
| --- |
| import numpy as np    b = np.empty(2, dtype = int)  print("Matrix b : \n", b)    a = np.empty([2, 2], dtype = int)  print("\nMatrix a : \n", a)    c = np.empty([3, 3])  print("\nMatrix c : \n", c) |

**Output:**



*Empty Matrix using pandas*

**Create Array using**[**numpy.zeros(shape, dtype = None, order = ‘C’)**](https://www.geeksforgeeks.org/numpy-zeros-python/)

* Python3

|  |
| --- |
| import numpy as np    b = np.zeros(2, dtype = int)  print("Matrix b : \n", b)    a = np.zeros([2, 2], dtype = int)  print("\nMatrix a : \n", a)    c = np.zeros([3, 3])  print("\nMatrix c : \n", c) |

**Output:**

Matrix b :

[0 0]

Matrix a :

[[0 0]

[0 0]]

Matrix c :

[[0. 0. 0.]

[0. 0. 0.]

[0. 0. 0.]]

**Operations on Numpy Arrays**

**Arithmetic Operations**

* **Addition:**
* Python3

|  |
| --- |
| import numpy as np    # Defining both the matrices  a = np.array([5, 72, 13, 100])  b = np.array([2, 5, 10, 30])    # Performing addition using arithmetic operator  add\_ans = a+b  print(add\_ans)    # Performing addition using numpy function  add\_ans = np.add(a, b)  print(add\_ans)    # The same functions and operations can be used for  # multiple matrices  c = np.array([1, 2, 3, 4])  add\_ans = a+b+c  print(add\_ans)    add\_ans = np.add(a, b, c)  print(add\_ans) |

**Output:**

[ 7 77 23 130]

[ 7 77 23 130]

[ 8 79 26 134]

[ 7 77 23 130]

* **Subtraction:**
* Python3

|  |
| --- |
| import numpy as np    # Defining both the matrices  a = np.array([5, 72, 13, 100])  b = np.array([2, 5, 10, 30])    # Performing subtraction using arithmetic operator  sub\_ans = a-b  print(sub\_ans)    # Performing subtraction using numpy function  sub\_ans = np.subtract(a, b)  print(sub\_ans) |

**Output:**

[ 3 67 3 70]

[ 3 67 3 70]

* **Multiplication:**
* Python3

|  |
| --- |
| import numpy as np    # Defining both the matrices  a = np.array([5, 72, 13, 100])  b = np.array([2, 5, 10, 30])    # Performing multiplication using arithmetic  # operator  mul\_ans = a\*b  print(mul\_ans)    # Performing multiplication using numpy function  mul\_ans = np.multiply(a, b)  print(mul\_ans) |

**Output:**

[ 10 360 130 3000]

[ 10 360 130 3000]

* **Division:**
* Python3

|  |
| --- |
| import numpy as np    # Defining both the matrices  a = np.array([5, 72, 13, 100])  b = np.array([2, 5, 10, 30])    # Performing division using arithmetic operators  div\_ans = a/b  print(div\_ans)    # Performing division using numpy functions  div\_ans = np.divide(a, b)  print(div\_ans) |

**Output:**

[ 2.5 14.4 1.3 3.33333333]

[ 2.5 14.4 1.3 3.33333333]

For more information, refer to our [NumPy – Arithmetic Operations Tutorial](https://www.geeksforgeeks.org/numpy-arithmetic-operations/)

**NumPy Array Indexing**

[Indexing](https://www.geeksforgeeks.org/numpy-indexing/) can be done in NumPy by using an array as an index. In the case of the slice, a view or shallow copy of the array is returned but in the index array, a copy of the original array is returned. Numpy arrays can be indexed with other arrays or any other sequence with the exception of tuples. The last element is indexed by -1 second last by -2 and so on.

**Python NumPy Array Indexing**

* Python3

|  |
| --- |
| # Python program to demonstrate  # the use of index arrays.  import numpy as np    # Create a sequence of integers from  # 10 to 1 with a step of -2  a = np.arange(10, 1, -2)  print("\n A sequential array with a negative step: \n",a)    # Indexes are specified inside the np.array method.  newarr = a[np.array([3, 1, 2 ])]  print("\n Elements at these indices are:\n",newarr) |

**Output:**

A sequential array with a negative step:

[10 8 6 4 2]

Elements at these indices are:

[4 8 6]

**NumPy Array Slicing**

Consider the syntax x[obj] where x is the array and obj is the index. The slice object is the index in the case of [basic slicing](https://www.geeksforgeeks.org/indexing-in-numpy/). Basic slicing occurs when obj is :

* a slice object that is of the form start: stop: step
* an integer
* or a tuple of slice objects and integers

All arrays generated by basic slicing are always the view in the original array.

* Python3

|  |
| --- |
| # Python program for basic slicing.  import numpy as np    # Arrange elements from 0 to 19  a = np.arrange(20)  print("\n Array is:\n ",a)    # a[start:stop:step]  print("\n a[-8:17:1] = ",a[-8:17:1])    # The : operator means all elements till the end.  print("\n a[10:] = ",a[10:]) |

**Output:**

Array is:

[ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]

a[-8:17:1] = [12 13 14 15 16]

a[10:] = [10 11 12 13 14 15 16 17 18 19]

Ellipsis can also be used along with basic slicing. Ellipsis (…) is the number of : objects needed to make a selection tuple of the same length as the dimensions of the array.

* Python3

|  |
| --- |
| # Python program for indexing using basic slicing with ellipsis  import numpy as np    # A 3 dimensional array.  b = np.array([[[1, 2, 3],[4, 5, 6]],              [[7, 8, 9],[10, 11, 12]]])    print(b[...,1]) #Equivalent to b[: ,: ,1 ] |

**Output:**

[[ 2 5]

[ 8 11]]

**NumPy Array Broadcasting**

The term [broadcasting](https://www.geeksforgeeks.org/python-broadcasting-with-numpy-arrays/) refers to how numpy treats arrays with different Dimensions during arithmetic operations which lead to certain constraints, the smaller array is broadcast across the larger array so that they have compatible shapes.

Let’s assume that we have a large data set, each datum is a list of parameters. In Numpy we have a 2-D array, where each row is a datum and the number of rows is the size of the data set. Suppose we want to apply some sort of scaling to all these data every parameter gets its own scaling factor or say Every parameter is multiplied by some factor.

Just to have a clear understanding, let’s count calories in foods using a macro-nutrient breakdown. Roughly put, the caloric parts of food are made of fats (9 calories per gram), protein (4 CPG), and carbs (4 CPG). So if we list some foods (our data), and for each food list its macro-nutrient breakdown (parameters), we can then multiply each nutrient by its caloric value (apply scaling) to compute the caloric breakdown of every food item.



With this transformation, we can now compute all kinds of useful information. For example, what is the total number of calories present in some food or, given a breakdown of my dinner know how many calories did I get from protein and so on.

**Let’s see a naive way of producing this computation with Numpy:**

* Python3

|  |
| --- |
| import numpy as np    macros = np.array([  [0.8, 2.9, 3.9],  [52.4, 23.6, 36.5],  [55.2, 31.7, 23.9],  [14.4, 11, 4.9]  ])    # Create a new array filled with zeros,  # of the same shape as macros.  result = np.zeros\_like(macros)    cal\_per\_macro = np.array([3, 3, 8])    # Now multiply each row of macros by  # cal\_per\_macro. In Numpy, `\*` is  # element-wise multiplication between two arrays.  for i in range(macros.shape[0]):      result[i, :] = macros[i, :] \* cal\_per\_macro    result |

**Output:**

array([[ 2.4, 8.7, 31.2],

[157.2, 70.8, 292. ],

[165.6, 95.1, 191.2],

[ 43.2, 33. , 39.2]])

**Broadcasting Rules:** Broadcasting two arrays together follow these rules:

* If the arrays don’t have the same rank then prepend the shape of the lower rank array with 1s until both shapes have the same length.
* The two arrays are compatible in a dimension if they have the same size in the dimension or if one of the arrays has size 1 in that dimension.
* The arrays can be broadcast together if they are compatible with all dimensions.
* After broadcasting, each array behaves as if it had a shape equal to the element-wise maximum of shapes of the two input arrays.
* In any dimension where one array had a size of 1 and the other array had a size greater than 1, the first array behaves as if it were copied along that dimension.
* Python3

|  |
| --- |
| import numpy as np    v = np.array([12, 24, 36])  w = np.array([45, 55])    # To compute an outer product we first  # reshape v to a column vector of shape 3x1  # then broadcast it against w to yield an output  # of shape 3x2 which is the outer product of v and w  print(np.reshape(v, (3, 1)) \* w)    X = np.array([[12, 22, 33], [45, 55, 66]])    # x has shape 2x3 and v has shape (3, )  # so they broadcast to 2x3,  print(X + v)    # Add a vector to each column of a matrix X has  # shape 2x3 and w has shape (2, ) If we transpose X  # then it has shape 3x2 and can be broadcast against w  # to yield a result of shape 3x2.    # Transposing this yields the final result  # of shape 2x3 which is the matrix.  print((X.T + w).T)    # Another solution is to reshape w to be a column  # vector of shape 2X1 we can then broadcast it  # directly against X to produce the same output.  print(X + np.reshape(w, (2, 1)))    # Multiply a matrix by a constant, X has shape 2x3.  # Numpy treats scalars as arrays of shape();  # these can be broadcast together to shape 2x3.  print(X \* 2) |

**Output:**

[[ 540 660]

[1080 1320]

[1620 1980]]

[[ 24 46 69]

[ 57 79 102]]

[[ 57 67 78]

[100 110 121]]

[[ 57 67 78]

[100 110 121]]

[[ 24 44 66]

[ 90 110 132]]

**Note:**For more information, refer to our [Python NumPy Tutorial](https://www.geeksforgeeks.org/numpy-tutorial/).

**Analyzing Data Using Pandas**

Python Pandas Is used for relational or labeled data and provides various data structures for manipulating such data and time series. This library is built on top of the NumPy library. This module is generally imported as:

import pandas as pd

Here, pd is referred to as an alias to the Pandas. However, it is not necessary to import the library using the alias, it just helps in writing less amount code every time a method or property is called. Pandas generally provide two data structures for manipulating data, They are:

* Series
* Dataframe

**Series:**

[Pandas Series](https://www.geeksforgeeks.org/python-pandas-series/) is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called indexes. Pandas Series is nothing but a column in an excel sheet. Labels need not be unique but must be a hashable type. The object supports both integer and label-based indexing and provides a host of methods for performing operations involving the index.

*Pandas Series*

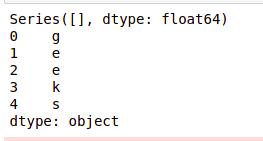
It can be created using the Series() function by loading the dataset from the existing storage like SQL, Database, CSV Files, Excel Files, etc., or from data structures like lists, dictionaries, etc.

**Python Pandas Creating Series**

* Python3

|  |
| --- |
| import pandas as pd  import numpy as np      # Creating empty series  ser = pd.Series()    print(ser)    # simple array  data = np.array(['g', 'e', 'e', 'k', 's'])    ser = pd.Series(data)  print(ser) |

**Output:**



*pnadas series*

**Dataframe:**

[Pandas DataFrame](https://www.geeksforgeeks.org/python-pandas-dataframe/) is a two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the data, rows, and columns.

*Pandas Dataframe*

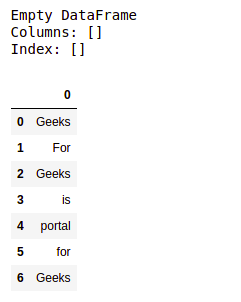
It can be created using the Dataframe() method and just like a series, it can also be from different file types and data structures.

**Python Pandas Creating Dataframe**

* Python3

|  |
| --- |
| import pandas as pd    # Calling DataFrame constructor  df = pd.DataFrame()  print(df)    # list of strings  lst = ['Geeks', 'For', 'Geeks', 'is',              'portal', 'for', 'Geeks']    # Calling DataFrame constructor on list  df = pd.DataFrame(lst)  df |

**Output:**



*Creating Dataframe from python list*

**Creating Dataframe from CSV**

We can [create a dataframe from the CSV](https://www.geeksforgeeks.org/creating-a-dataframe-using-csv-files/) files using the [read\_csv()](https://www.geeksforgeeks.org/python-read-csv-using-pandas-read_csv/) function.

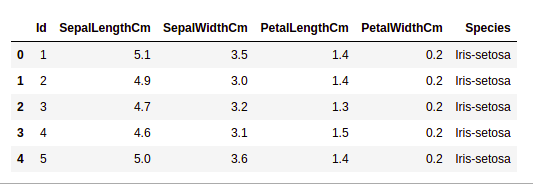
**Note:** This dataset can be downloaded from [here](https://datahub.io/machine-learning/iris).

**Python Pandas read CSV**

* Python3

|  |
| --- |
| import pandas as pd    # Reading the CSV file  df = pd.read\_csv("Iris.csv")    # Printing top 5 rows  df.head() |

**Output:**



*head of  a dataframe*

**Filtering DataFrame**

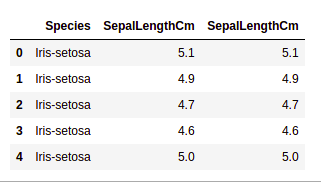
Pandas [dataframe.filter()](https://www.geeksforgeeks.org/python-pandas-dataframe-filter/) function is used to Subset rows or columns of dataframe according to labels in the specified index. Note that this routine does not filter a dataframe on its contents. The filter is applied to the labels of the index.

**Python Pandas Filter Dataframe**

* Python3

|  |
| --- |
| import pandas as pd    # Reading the CSV file  df = pd.read\_csv("Iris.csv")    # applying filter function  df.filter(["Species", "SepalLengthCm", "SepalLengthCm"]).head() |

**Output:**



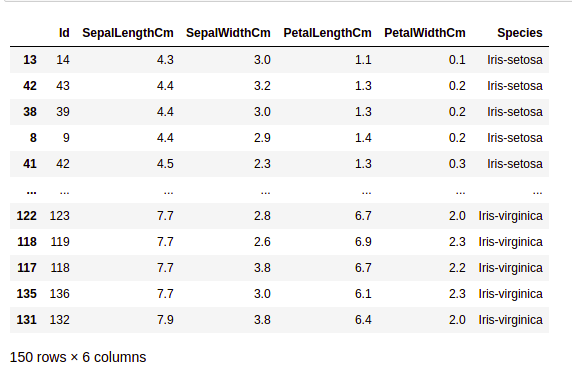
*Applying filter on dataset*

**Sorting DataFrame**

In order to sort the data frame in pandas, the function [sort\_values()](https://www.geeksforgeeks.org/python-pandas-dataframe-sort_values-set-1/) is used. Pandas sort\_values() can sort the data frame in Ascending or Descending order.

**Python Pandas Sorting Dataframe in Ascending Order**

**Output:**



*Sorted dataset based on a column value*

**Pandas GroupBy**

[Groupby](https://www.geeksforgeeks.org/pandas-groupby/) is a pretty simple concept. We can create a grouping of categories and apply a function to the categories. In real data science projects, you’ll be dealing with large amounts of data and trying things over and over, so for efficiency, we use the Groupby concept.  Groupby mainly refers to a process involving one or more of the following steps they are:

* **Splitting:** It is a process in which we split data into group by applying some conditions on datasets.
* **Applying:**It is a process in which we apply a function to each group independently.
* **Combining:**It is a process in which we combine different datasets after applying groupby and results into a data structure.

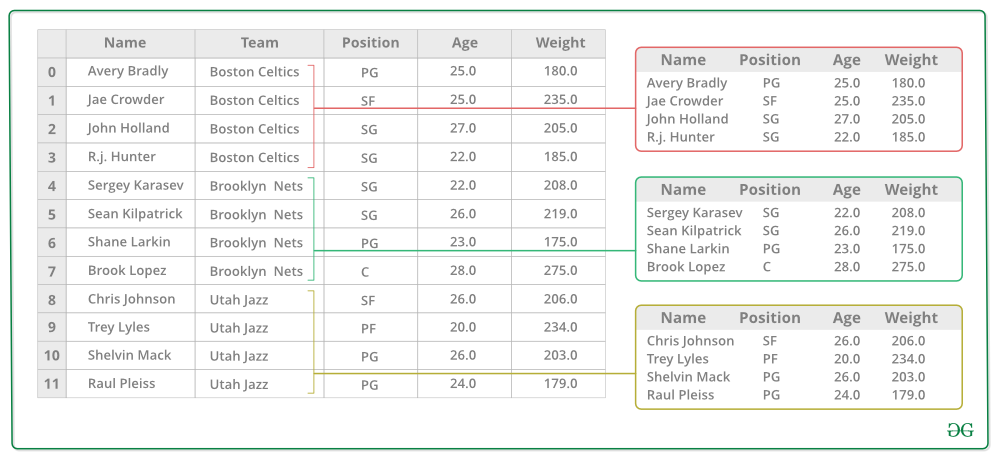
The following image will help in understanding the process involve in the Groupby concept.

1. Group the unique values from the Team column

*Pandas Groupby Method*

2. Now there’s a bucket for each group

3. Toss the other data into the buckets



4. Apply a function on the weight column of each bucket.

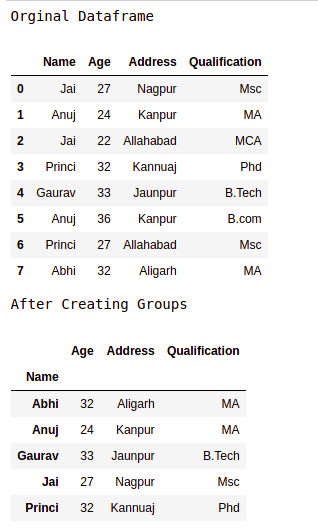
*Applying Function on the weight column of each column*

**Python Pandas GroupBy**

* Python3

|  |
| --- |
| # importing pandas module  import pandas as pd    # Define a dictionary containing employee data  data1 = {'Name': ['Jai', 'Anuj', 'Jai', 'Princi',                    'Gaurav', 'Anuj', 'Princi', 'Abhi'],           'Age': [27, 24, 22, 32,                   33, 36, 27, 32],           'Address': ['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj',                       'Jaunpur', 'Kanpur', 'Allahabad', 'Aligarh'],           'Qualification': ['Msc', 'MA', 'MCA', 'Phd',                             'B.Tech', 'B.com', 'Msc', 'MA']}    # Convert the dictionary into DataFrame  df = pd.DataFrame(data1)    print("Original Dataframe")  display(df)    # applying groupby() function to  # group the data on Name value.  gk = df.groupby('Name')    # Let's print the first entries  # in all the groups formed.  print("After Creating Groups")  gk.first() |

**Output:**



*pandas groupby*

**Applying function to group:**

After splitting a data into a group, we apply a function to each group in order to do that we perform some operations they are:

* **Aggregation:**It is a process in which we compute a summary statistic (or statistics) about each group. For Example, Compute group sums or means
* **Transformation:**It is a process in which we perform some group-specific computations and return a like-indexed. For Example, Filling NAs within groups with a value derived from each group
* **Filtration:** It is a process in which we discard some groups, according to a group-wise computation that evaluates True or False. For Example, Filtering out data based on the group sum or mean

**Pandas Aggregation**

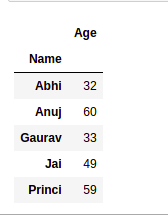
[Aggregation](https://www.geeksforgeeks.org/grouping-and-aggregating-with-pandas/) is a process in which we compute a summary statistic about each group. The aggregated function returns a single aggregated value for each group. After splitting data into groups using groupby function, several aggregation operations can be performed on the grouped data.

**Python Pandas Aggregation**

* Python3

|  |
| --- |
| # importing pandas module  import pandas as pd    # importing numpy as np  import numpy as np    # Define a dictionary containing employee data  data1 = {'Name': ['Jai', 'Anuj', 'Jai', 'Princi',                    'Gaurav', 'Anuj', 'Princi', 'Abhi'],           'Age': [27, 24, 22, 32,                   33, 36, 27, 32],           'Address': ['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj',                       'Jaunpur', 'Kanpur', 'Allahabad', 'Aligarh'],           'Qualification': ['Msc', 'MA', 'MCA', 'Phd',                                    'B.Tech', 'B.com', 'Msc', 'MA']}      # Convert the dictionary into DataFrame  df = pd.DataFrame(data1)    # performing aggregation using  # aggregate method    grp1 = df.groupby('Name')    grp1.aggregate(np.sum) |

**Output:**



*Use of sum aggregate function on dataset*

**Concatenating DataFrame**

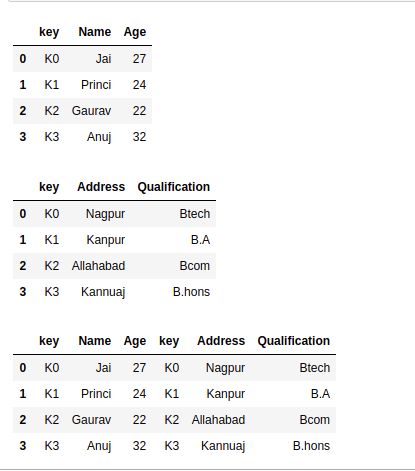
In order to concat the dataframe, we use [concat()](https://www.geeksforgeeks.org/pandas-concat-function-in-python/) function which helps in concatenating the dataframe. This function does all the heavy lifting of performing concatenation operations along with an axis of Pandas objects while performing optional set logic (union or intersection) of the indexes (if any) on the other axes.

**Python Pandas Concatenate Dataframe**

* Python3

|  |
| --- |
| # importing pandas module  import pandas as pd    # Define a dictionary containing employee data  data1 = {'key': ['K0', 'K1', 'K2', 'K3'],           'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],          'Age':[27, 24, 22, 32],}    # Define a dictionary containing employee data  data2 = {'key': ['K0', 'K1', 'K2', 'K3'],           'Address':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj'],          'Qualification':['Btech', 'B.A', 'Bcom', 'B.hons']}    # Convert the dictionary into DataFrame  df = pd.DataFrame(data1)    # Convert the dictionary into DataFrame  df1 = pd.DataFrame(data2)      display(df, df1)    # combining series and dataframe  res = pd.concat([df, df1], axis=1)    res |

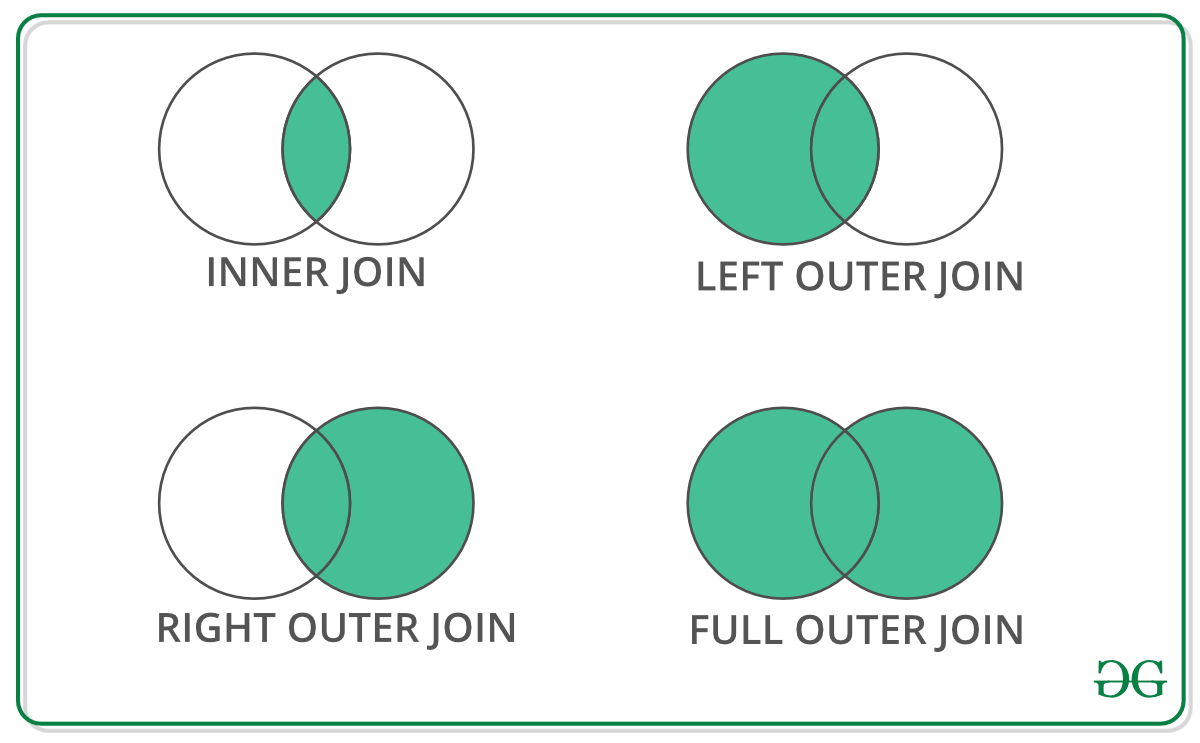
**Output:**



**Merging DataFrame**

When we need to combine very large DataFrames, joins serve as a powerful way to perform these operations swiftly. Joins can only be done on two DataFrames at a time, denoted as left and right tables. The key is the common column that the two DataFrames will be joined on. It’s a good practice to use keys that have unique values throughout the column to avoid unintended duplication of row values. Pandas provide a single function, [merge()](https://www.geeksforgeeks.org/joining-two-pandas-dataframes-using-merge/), as the entry point for all standard database join operations between DataFrame objects.

There are four basic ways to handle the join (inner, left, right, and outer), depending on which rows must retain their data.

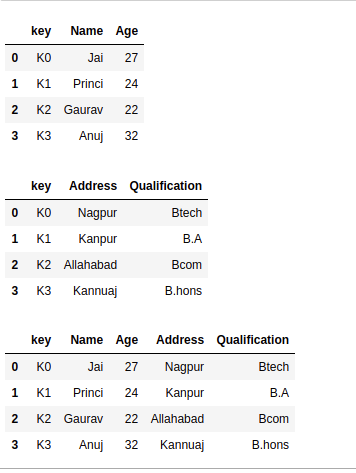


**Python Pandas Merge Dataframe**

* Python3

|  |
| --- |
| # importing pandas module  import pandas as pd    # Define a dictionary containing employee data  data1 = {'key': ['K0', 'K1', 'K2', 'K3'],           'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],          'Age':[27, 24, 22, 32],}    # Define a dictionary containing employee data  data2 = {'key': ['K0', 'K1', 'K2', 'K3'],           'Address':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj'],          'Qualification':['Btech', 'B.A', 'Bcom', 'B.hons']}    # Convert the dictionary into DataFrame  df = pd.DataFrame(data1)    # Convert the dictionary into DataFrame  df1 = pd.DataFrame(data2)      display(df, df1)    # using .merge() function  res = pd.merge(df, df1, on='key')    res |

**Output:**



*Concatinating Two datasets*

**Joining DataFrame**

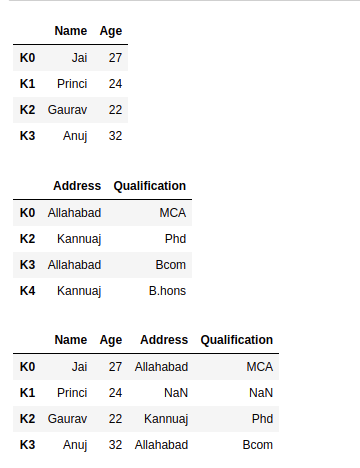
In order to join the dataframe, we use [.join()](https://www.geeksforgeeks.org/python-string-join-method/) function this function is used for combining the columns of two potentially differently indexed DataFrames into a single result DataFrame.

**Python Pandas Join Dataframe**

* Python3

|  |
| --- |
| # importing pandas module  import pandas as pd    # Define a dictionary containing employee data  data1 = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],          'Age':[27, 24, 22, 32]}    # Define a dictionary containing employee data  data2 = {'Address':['Allahabad', 'Kannuaj', 'Allahabad', 'Kannuaj'],          'Qualification':['MCA', 'Phd', 'Bcom', 'B.hons']}    # Convert the dictionary into DataFrame  df = pd.DataFrame(data1,index=['K0', 'K1', 'K2', 'K3'])    # Convert the dictionary into DataFrame  df1 = pd.DataFrame(data2, index=['K0', 'K2', 'K3', 'K4'])      display(df, df1)    # joining dataframe  res = df.join(df1)    res |

**Output:**



*Joining two datasets*

For more information, refer to our [Pandas Merging, Joining, and Concatenating](https://www.geeksforgeeks.org/python-pandas-merging-joining-and-concatenating/) tutorial

For a complete guide on Pandas refer to our [Pandas Tutorial](https://www.geeksforgeeks.org/pandas-tutorial/).

**Visualization with Matplotlib**

Matplotlib is easy to use and an amazing visualizing library in Python. It is built on NumPy arrays and designed to work with the broader SciPy stack and consists of several plots like line, bar, scatter, histogram, etc.

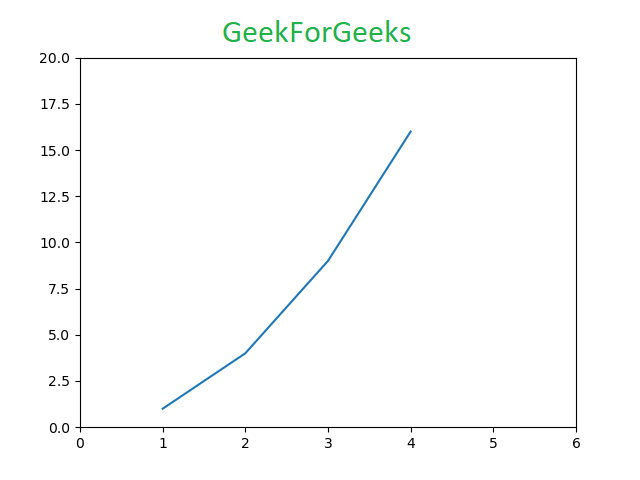
**Pyplot**

[**Pyplot**](https://www.geeksforgeeks.org/pyplot-in-matplotlib/) is a Matplotlib module that provides a MATLAB-like interface. Pyplot provides functions that interact with the figure i.e. creates a figure, decorates the plot with labels, and creates a plotting area in a figure.

* Python3

|  |
| --- |
| # Python program to show pyplot module  import matplotlib.pyplot as plt      plt.plot([1, 2, 3, 4], [1, 4, 9, 16])  plt.axis([0, 6, 0, 20])  plt.show() |

**Output:**



**Bar chart**

A [bar plot](https://www.geeksforgeeks.org/bar-plot-in-matplotlib/) or bar chart is a graph that represents the category of data with rectangular bars with lengths and heights that is proportional to the values which they represent. The bar plots can be plotted horizontally or vertically. A bar chart describes the comparisons between the discrete categories. It can be created using the bar() method.

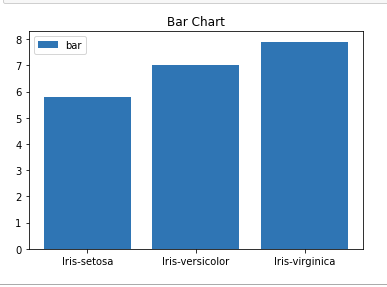
**Python Matplotlib Bar Chart**

Here we will use the iris dataset only

* Python3

|  |
| --- |
| import matplotlib.pyplot as plt  import pandas as pd    df = pd.read\_csv("Iris.csv")    # This will plot a simple bar chart  plt.bar(df['Species'], df['SepalLengthCm'])    # Title to the plot  plt.title("Iris Dataset")    # Adding the legends  plt.legend(["bar"])  plt.show() |

**Output:**



*Bar chart using matplotlib library*

**Histograms**

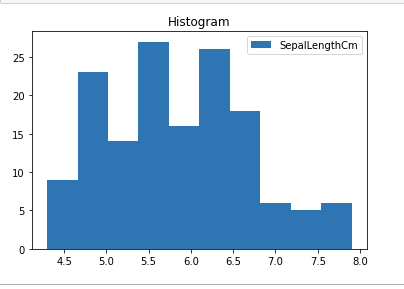
A [histogram](https://www.geeksforgeeks.org/plotting-histogram-in-python-using-matplotlib/) is basically used to represent data in the form of some groups. It is a type of bar plot where the X-axis represents the bin ranges while the Y-axis gives information about frequency. To create a histogram the first step is to create a bin of the ranges, then distribute the whole range of the values into a series of intervals, and count the values which fall into each of the intervals. Bins are clearly identified as consecutive, non-overlapping intervals of variables. The [hist()](https://www.geeksforgeeks.org/matplotlib-pyplot-hist-in-python/) function is used to compute and create a histogram of x.

**Python Matplotlib Histogram**

* Python3

|  |
| --- |
| import matplotlib.pyplot as plt  import pandas as pd    df = pd.read\_csv("Iris.csv")    plt.hist(df["SepalLengthCm"])    # Title to the plot  plt.title("Histogram")    # Adding the legends  plt.legend(["SepalLengthCm"])  plt.show() |

**Output:**



*Histplot using matplotlib library*

**Scatter Plot**

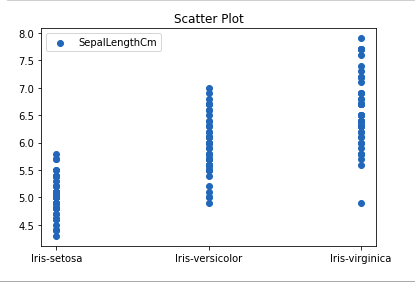
Scatter plots are used to observe relationship between variables and uses dots to represent the relationship between them. The [scatter()](https://www.geeksforgeeks.org/matplotlib-pyplot-scatter-in-python/) method in the matplotlib library is used to draw a scatter plot.

**Python Matplotlib Scatter Plot**

* Python3

|  |
| --- |
| import matplotlib.pyplot as plt  import pandas as pd    df = pd.read\_csv("Iris.csv")    plt.scatter(df["Species"], df["SepalLengthCm"])    # Title to the plot  plt.title("Scatter Plot")    # Adding the legends  plt.legend(["SepalLengthCm"])  plt.show() |

**Output:**



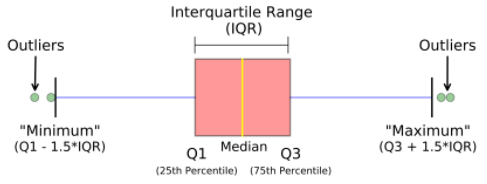
*Scatter plot using matplotlib library*

**Box Plot**

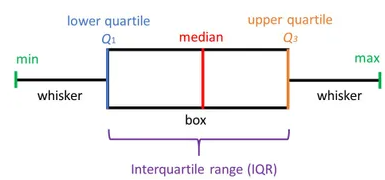
A [boxplot](https://www.geeksforgeeks.org/box-plot-in-python-using-matplotlib/),Correlation also known as a box and whisker plot. It is a very good visual representation when it comes to measuring the data distribution. Clearly plots the median values, outliers and the quartiles. Understanding data distribution is another important factor which leads to better model building. If data has outliers, box plot is a recommended way to identify them and take necessary actions. The box and whiskers chart shows how data is spread out. Five pieces of information are generally included in the chart

* The minimum is shown at the far left of the chart, at the end of the left ‘whisker’
* First quartile, Q1, is the far left of the box (left whisker)
* The median is shown as a line in the center of the box
* Third quartile, Q3, shown at the far right of the box (right whisker)
* The maximum is at the far right of the box

Representation of box plot



*Inter quartile range*



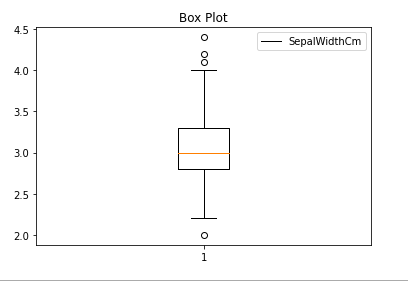
*Illustrating box plot*

**Python Matplotlib Box Plot**

* Python3

|  |
| --- |
| import matplotlib.pyplot as plt  import pandas as pd    df = pd.read\_csv("Iris.csv")    plt.boxplot(df["SepalWidthCm"])    # Title to the plot  plt.title("Box Plot")    # Adding the legends  plt.legend(["SepalWidthCm"])  plt.show() |

**Output:**



*Boxplot using matplotlib library*

**Correlation Heatmaps**

A 2-D Heatmap is a data visualization tool that helps to represent the magnitude of the phenomenon in form of colors. A correlation heatmap is a heatmap that shows a 2D correlation matrix between two discrete dimensions, using colored cells to represent data from usually a monochromatic scale. The values of the first dimension appear as the rows of the table while the second dimension is a column. The color of the cell is proportional to the number of measurements that match the dimensional value. This makes correlation heatmaps ideal for data analysis since it makes patterns easily readable and highlights the differences and variation in the same data. A correlation heatmap, like a regular heatmap, is assisted by a colorbar making data easily readable and comprehensible.

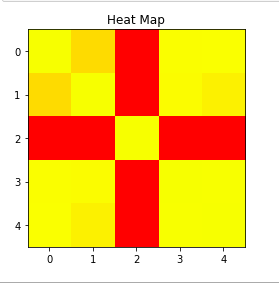
**Note:**The data here has to be passed with corr() method to generate a correlation heatmap. Also, corr() itself eliminates columns that will be of no use while generating a correlation heatmap and selects those which can be used.

**Python Matplotlib Correlation Heatmap**

* Python3

|  |
| --- |
| import matplotlib.pyplot as plt  import pandas as pd    df = pd.read\_csv("Iris.csv")    plt.imshow(df.corr() , cmap = 'autumn' , interpolation = 'nearest' )    plt.title("Heat Map")  plt.show() |

**Output:**



*Heatmap using matplotlib library*

***For more information on data visualization refer to our below tutorials –***

* [*Data Visualization using Matplotlib*](https://www.geeksforgeeks.org/data-visualization-using-matplotlib/)
* [*Data Visualization with Python Seaborn*](https://www.geeksforgeeks.org/data-visualization-with-python-seaborn/)
* [*Data Visualisation in Python using Matplotlib and Seaborn*](https://www.geeksforgeeks.org/data-visualisation-in-python-using-matplotlib-and-seaborn/)
* [*Using Plotly for Interactive Data Visualization in Python*](https://www.geeksforgeeks.org/using-plotly-for-interactive-data-visualization-in-python/)
* [*Interactive Data Visualization with Bokeh*](https://www.geeksforgeeks.org/python-bokeh-tutorial-interactive-data-visualization-with-bokeh/)

**Exploratory Data Analysis**

[Exploratory Data Analysis (EDA)](https://www.geeksforgeeks.org/what-is-exploratory-data-analysis/) 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.

**Note:** We will be using Iris Dataset.

**Getting Information about the Dataset**

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

**Shape of Dataframe**

* Python3

|  |
| --- |
| df.shape |

**Output:**

(150, 6)

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

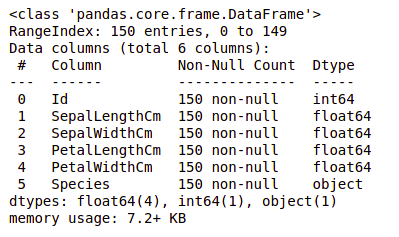
Now, let’s also the columns and their data types. For this, we will use the [info()](https://www.geeksforgeeks.org/python-pandas-dataframe-info/) method.

**Information about Dataset**

* Python3

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

**Output:**



*information about the dataset*

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

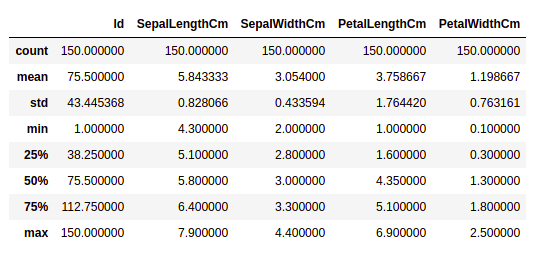
Let’s get a quick statistical summary of the dataset using the [**describe()**](https://www.geeksforgeeks.org/python-pandas-dataframe-describe-method/) method. The describe() function applies basic statistical computations on the dataset like extreme values, count of data points standard deviation, etc.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

**Description of dataset**

* Python3

|  |
| --- |
| df.describe() |

**Output:**



*Description about the dataset*

We can see the count of each column along with their mean value, standard deviation, minimum and maximum values.

**Checking Missing Values**

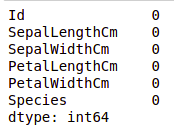
We will check if our data contains any missing values or not. Missing values can occur when no information is provided for one or more items or for a whole unit. We will use the [isnull()](https://www.geeksforgeeks.org/python-pandas-isnull-and-notnull/) method.

**python code for missing value**

* Python3

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

**Output:**



*Missing values in the dataset*

We can see that no column has any missing value.

**Checking Duplicates**

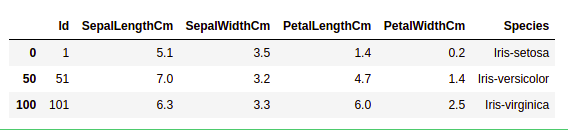
Let’s see if our dataset contains any duplicates or not. Pandas [drop\_duplicates()](https://www.geeksforgeeks.org/python-pandas-dataframe-drop_duplicates/) method helps in removing duplicates from the data frame.

**Pandas function for missing values**

* Python3

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

**Output:**



*Dropping duplicate value in the dataset*

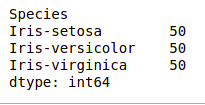
We can see that there are only three unique species. Let’s see if the dataset is balanced or not i.e. all the species contain equal amounts of rows or not. We will use the [Series.value\_counts()](https://www.geeksforgeeks.org/python-pandas-series-value_counts/) function. This function returns a Series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

**Python code for value counts in the column**

* Python3

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

**Output:**



*value count in the dataset*

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

**Relation between variables**

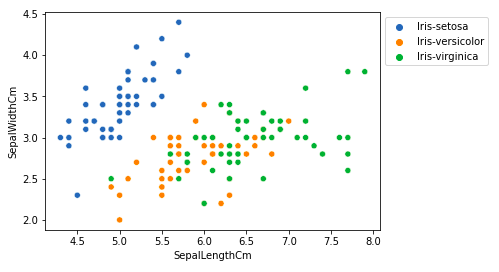
We will see the relationship between the sepal length and sepal width and also between petal length and petal width.

**Comparing Sepal Length and Sepal Width**

* Python3

|  |
| --- |
| # 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() |

**Output:**



*Scatter plot using matplotlib library*

From the above plot, we can infer that –

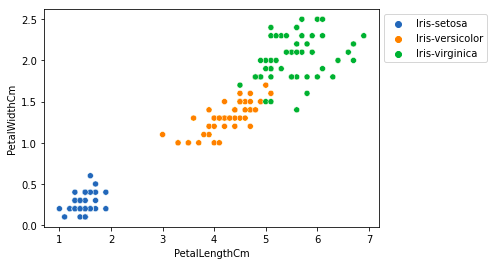
* Species Setosa has smaller sepal lengths but larger sepal widths.
* Versicolor Species lies in the middle of the other two species in terms of sepal length and width
* Species Virginica has larger sepal lengths but smaller sepal widths.

**Comparing Petal Length and Petal Width**

* Python3

|  |
| --- |
| # 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() |

**Output:**



*sactter plot petal length*

From the above plot, we can infer that –

* The 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 petal lengths and widths.

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

**Python code for pairplot**

* Python3

|  |
| --- |
| # importing packages  import seaborn as sns  import matplotlib.pyplot as plt      sns.pairplot(df.drop(['Id'], axis = 1),               hue='Species', height=2) |

**Output:**

*Pairplot for the dataset*

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.

**Handling Correlation**

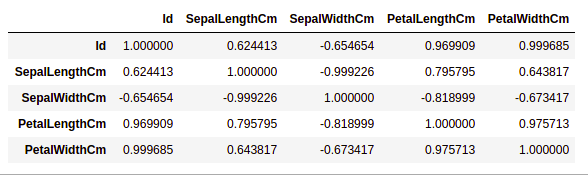
Pandas [dataframe.corr()](https://www.geeksforgeeks.org/python-pandas-dataframe-corr/) is used to find the pairwise correlation of all columns in the dataframe. Any NA values are automatically excluded. Any non-numeric data type columns in the dataframe are ignored.

**Example:**

* Python3

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

**Output:**



*correlation between columns in the dataset*

**Heatmaps**

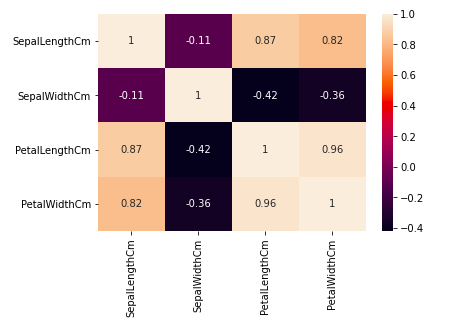
The heatmap is a data visualization technique that is used to analyze the dataset as colors in two dimensions. Basically, it shows a correlation between all numerical variables in the dataset. In simpler terms, we can plot the above-found correlation using the heatmaps.

**python code for heatmap**

* Python3

|  |
| --- |
| # 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() |

**Output:**



*Heatmap for correlation in the dataset*

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.

**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 outliers, and the removal process is the data frame same as removing a data item from the panda’s dataframe.

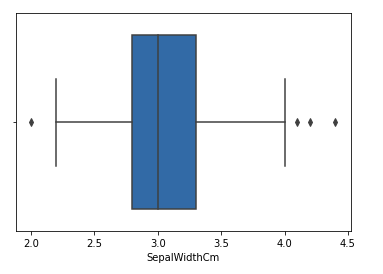
Let’s consider the iris dataset and let’s plot the boxplot for the SepalWidthCm column.

**python code for Boxplot**

* Python3

|  |
| --- |
| # 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) |

**Output:**



*Boxplot for sepalwidth column*

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

**Removing Outliers**

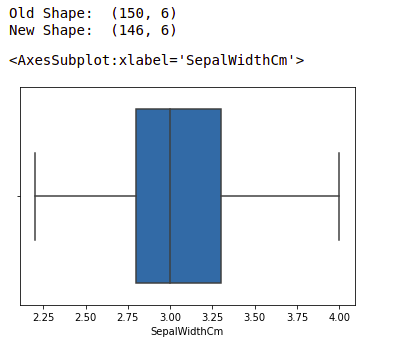
For removing the outlier, one must follow the same process of removing an entry from the dataset using its exact position in the dataset because in all the above methods of detecting the outliers end result is the list of all those data items that satisfy the outlier definition according to the method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

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.

* Python3

|  |
| --- |
| # Importing  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) |

**Output:**



*boxplot using seaborn library*

**Exploratory Data Analysis (EDA)**

Exploratory Data Analysis (EDA) refers to the method of studying and exploring record sets to apprehend their predominant traits, discover patterns, locate outliers, and identify relationships between variables. EDA is normally carried out as a preliminary step before undertaking extra formal statistical analyses or modeling.

**The Foremost Goals of EDA**

**1. Data Cleaning:** EDA involves examining the information for errors, lacking values, and inconsistencies. It includes techniques including records imputation, managing missing statistics, and figuring out and getting rid of outliers.

**2. Descriptive Statistics:**EDA utilizes precise records to recognize the important tendency, variability, and distribution of variables. Measures like suggest, median, mode, preferred deviation, range, and percentiles are usually used.

**3. Data Visualization:** EDA employs visual techniques to represent the statistics graphically. Visualizations consisting of histograms, box plots, scatter plots, line plots, heatmaps, and bar charts assist in identifying styles, trends, and relationships within the facts.

**4. Feature Engineering:** EDA allows for the exploration of various variables and their adjustments to create new functions or derive meaningful insights. Feature engineering can contain scaling, normalization, binning, encoding express variables, and creating interplay or derived variables.

**5. Correlation and Relationships:**EDA allows discover relationships and dependencies between variables. Techniques such as correlation analysis, scatter plots, and pass-tabulations offer insights into the power and direction of relationships between variables.

**6. Data Segmentation:** EDA can contain dividing the information into significant segments based totally on sure standards or traits. This segmentation allows advantage insights into unique subgroups inside the information and might cause extra focused analysis.

**7. Hypothesis Generation:** EDA aids in generating hypotheses or studies questions based totally on the preliminary exploration of the data. It facilitates form the inspiration for in addition evaluation and model building.

**8. Data Quality Assessment:** EDA permits for assessing the nice and reliability of the information. It involves checking for records integrity, consistency, and accuracy to make certain the information is suitable for analysis.

**Types of EDA**

Depending on the number of columns we are analyzing we can divide EDA into two types.

EDA, or Exploratory Data Analysis, refers back to the method of analyzing and analyzing information units to uncover styles, pick out relationships, and gain insights. There are various sorts of EDA strategies that can be hired relying on the nature of the records and the desires of the evaluation. Here are some not unusual kinds of EDA:

**1. Univariate Analysis:**This sort of evaluation makes a speciality of analyzing character variables inside the records set. It involves summarizing and visualizing a unmarried variable at a time to understand its distribution, relevant tendency, unfold, and different applicable records. Techniques like histograms, field plots, bar charts, and precis information are generally used in univariate analysis.

**2. Bivariate Analysis:**Bivariate evaluation involves exploring the connection between  variables. It enables find associations, correlations, and dependencies between pairs of variables. Scatter plots, line plots, correlation matrices, and move-tabulation are generally used strategies in bivariate analysis.

**3. Multivariate Analysis:** Multivariate analysis extends bivariate evaluation to encompass greater than  variables. It ambitions to apprehend the complex interactions and dependencies among more than one variables in a records set. Techniques inclusive of heatmaps, parallel coordinates, aspect analysis, and primary component analysis (PCA) are used for multivariate analysis.

**4. Time Series Analysis:** This type of analysis is mainly applied to statistics sets that have a temporal component. Time collection evaluation entails inspecting and modeling styles, traits, and seasonality inside the statistics through the years. Techniques like line plots, autocorrelation analysis, transferring averages, and ARIMA (AutoRegressive Integrated Moving Average) fashions are generally utilized in time series analysis.

**5. Missing Data Analysis:** Missing information is a not unusual issue in datasets, and it may impact the reliability and validity of the evaluation. Missing statistics analysis includes figuring out missing values, know-how the patterns of missingness, and using suitable techniques to deal with missing data. Techniques along with lacking facts styles, imputation strategies, and sensitivity evaluation are employed in lacking facts evaluation.

**6. Outlier Analysis:** Outliers are statistics factors that drastically deviate from the general sample of the facts. Outlier analysis includes identifying and knowledge the presence of outliers, their capability reasons, and their impact at the analysis. Techniques along with box plots, scatter plots, z-rankings, and clustering algorithms are used for outlier evaluation.

**7. Data Visualization:** Data visualization is a critical factor of EDA that entails creating visible representations of the statistics to facilitate understanding and exploration. Various visualization techniques, inclusive of bar charts, histograms, scatter plots, line plots, heatmaps, and interactive dashboards, are used to represent exclusive kinds of statistics.

These are just a few examples of the types of EDA techniques that can be employed at some stage in information evaluation. The choice of strategies relies upon on the information traits, research questions, and the insights sought from the analysis.

**Exploratory Data Analysis (EDA) Using Python Libraries**

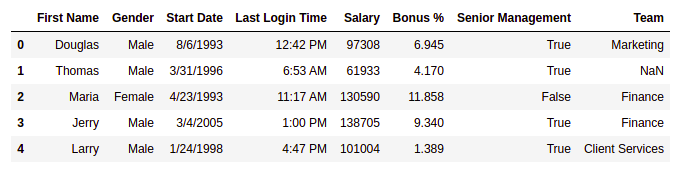
For the simplicity of the article, we will use a single dataset. We will use the employee data for this. It contains 8 columns namely – First Name, Gender, Start Date, Last Login, Salary, Bonus%, Senior Management, and Team. Wecan get the dataset here[Employees.csv](https://media.geeksforgeeks.org/wp-content/uploads/employees.csv)

Let’s read the dataset using the Pandas [read\_csv()](https://www.geeksforgeeks.org/python-read-csv-using-pandas-read_csv/) function and print the 1st five rows. To print the first five rows we will use the [head()](https://www.geeksforgeeks.org/python-pandas-dataframe-series-head-method/) function.

* Python3

|  |
| --- |
| import pandas as pd  import numpy as np  # read datasdet using pandas  df = pd.read\_csv('employees.csv')  df.head() |

**Output:**



*First five rows of the dataframe*

**Getting Insights About The Dataset**

Let’s see the shape of the data using the shape.

* Python3

|  |
| --- |
| df.shape |

**Output:**

(1000, 8)

This means that this dataset has 1000 rows and 8 columns.

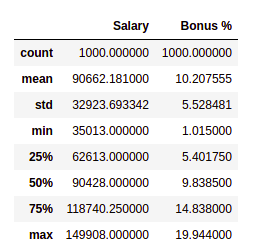
Let’s get a quick summary of the dataset using the pandas [**describe()**](https://www.geeksforgeeks.org/python-pandas-dataframe-describe-method/) method. The describe() function applies basic statistical computations on the dataset like extreme values, count of data points standard deviation, etc. Any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

**Example:**

* Python3

|  |
| --- |
| df.describe() |

**Output:**



*description of the dataframe*

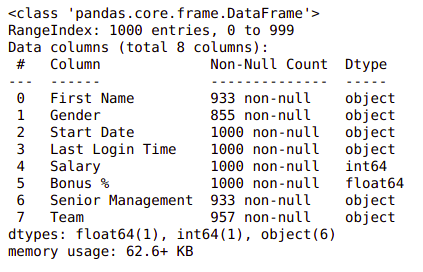
Note we can also get the description of categorical columns of the dataset if we specify ***include =’all’***in the describe function.

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

* Python3

|  |
| --- |
| # information about the dataset  df.info() |

**Output:**



*Information about the dataset*

**Changing Dtype from Object to Datetime**

Start Date is an important column for employees. However, it is not of much use if we can not handle it properly to handle this type of data pandas provide a special function [datetime()](https://www.geeksforgeeks.org/python-datetime-module/) from which we can change object type to DateTime format.

* Python3

|  |
| --- |
| # convert "Start Date" column to datetime data type  df['Start Date'] = pd.to\_datetime(df['Start Date']) |

 We can see the number of unique elements in our dataset. This will help us in deciding which type of encoding to choose for converting categorical columns into numerical columns.

* Python3

|  |
| --- |
| df.nunique() |

**Output:**

First Name 200  
 Gender 2  
 Start Date 972  
 Last Login Time 720  
 Salary 995  
 Bonus % 971  
 Senior Management 2  
 Team 10  
 dtype: int64

Till now we have got an idea about the dataset used. Now Let’s see if our dataset contains any missing values or not.

**Handling Missing Values**

You all must be wondering why a dataset will contain any missing values. It can occur when no information is provided for one or more items or for a whole unit. For Example, Suppose different users being surveyed may choose not to share their income, and some users may choose not to share their address in this way many datasets went missing. Missing Data is a very big problem in real-life scenarios. Missing Data can also refer to as NA(Not Available) values in pandas. There are several useful functions for detecting, removing, and replacing null values in Pandas DataFrame :

* [isnull()](https://www.geeksforgeeks.org/python-pandas-isnull-and-notnull/)
* [notnull()](https://www.geeksforgeeks.org/python-pandas-isnull-and-notnull/)
* [dropna()](https://www.geeksforgeeks.org/python-pandas-dataframe-dropna/)
* [fillna()](https://www.geeksforgeeks.org/python-pandas-dataframe-fillna-to-replace-null-values-in-dataframe/)
* [replace()](https://www.geeksforgeeks.org/python-pandas-dataframe-replace/)
* [interpolate()](https://www.geeksforgeeks.org/python-pandas-dataframe-interpolate/)

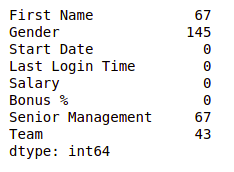
Now let’s check if there are any missing values in our dataset or not.

**Example:**

* Python3

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

**Output:**



*Null values in dataframe*

We can see that every column has a different amount of missing values. Like Gender has 145 missing values and salary has 0. Now for handling these missing values there can be several cases like dropping the rows containing NaN or replacing NaN with either mean, median, mode, or some other value.

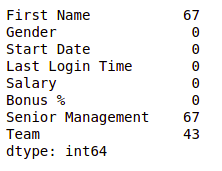
Now, let’s try to fill in the missing values of gender with the string “No Gender”.

**Example:**

* Python3

|  |
| --- |
| df["Gender"].fillna("No Gender", inplace = True)    df.isnull().sum() |

**Output:**



*Null values in dataframe after filling Gender column*

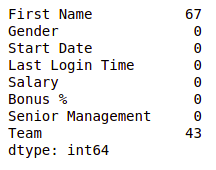
We can see that now there is no null value for the gender column. Now, Let’s fill the senior management with the mode value.

**Example:**

* Python3

|  |
| --- |
| mode = df['Senior Management'].mode().values[0]  df['Senior Management']= df['Senior Management'].replace(np.nan, mode)    df.isnull().sum() |

**Output:**



*Null values in dataframe after filling S senior management column*

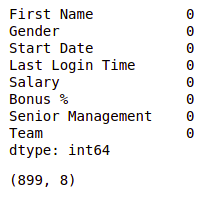
Now for the first name and team, we cannot fill the missing values with arbitrary data, so, let’s drop all the rows containing these missing values.

**Example:**

* Python3

|  |
| --- |
| df = df.dropna(axis = 0, how ='any')    print(df.isnull().sum())  df.shape |

**Output:**



*Null values in dataframe after dropping all null values*

We can see that our dataset is now free of all the missing values and after dropping the data the number of rows also reduced from 1000 to 899.

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

After removing the missing data let’s visualize our data.

**Data Encoding**

There are some models like [Linear Regression](https://www.geeksforgeeks.org/ml-linear-regression/) which does not work with categorical dataset in that case we should try to encode categorical dataset into the numerical column. we can use different methods for encoding like Label encoding or [One-hot encoding](https://www.geeksforgeeks.org/ml-one-hot-encoding-of-datasets-in-python/). pandas and sklearn provide different functions for encoding in our case we will use the [LabelEncoding](https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/) function from sklearn to encode the *Gender* column.

* Python3

|  |
| --- |
| from sklearn.preprocessing import LabelEncoder  # create an instance of LabelEncoder  le = LabelEncoder()    # fit and transform the "Senior Management"  # column with LabelEncoder  df['Gender'] = le.fit\_transform\                      (df['Gender']) |

Noe

**Data visualization**

Data Visualization is the process of analyzing data in the form of graphs or maps, making it a lot easier to understand the trends or patterns in the data.

Let’s see some commonly used graphs –

***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/)

**Histogram**

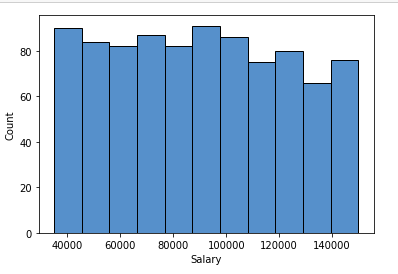
It can be used for both uni and bivariate analysis.

**Example:**

* Python3

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

**Output:**



*Histogram plot of salary column*

**Boxplot**

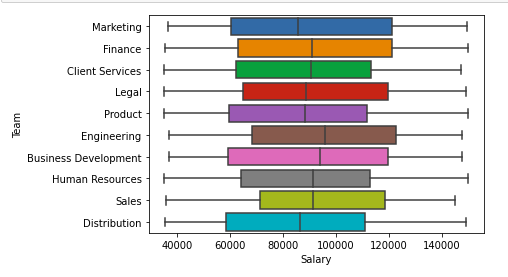
It can also be used for univariate and bivariate analyses.

**Example:**

* Python3

|  |
| --- |
| # importing packages  import seaborn as sns  import matplotlib.pyplot as plt      sns.boxplot( x="Salary", y='Team', data=df, )  plt.show() |

**Output:**



*Boxplot of Salary and team column*

**Scatter Boxplot For Data Visualization**

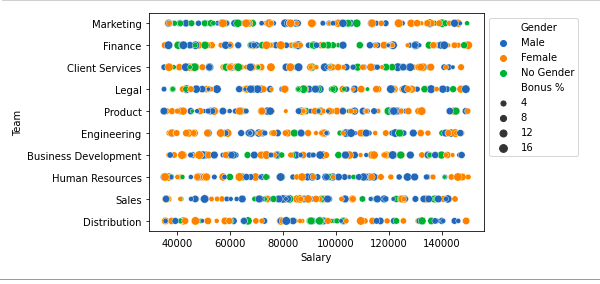
It can be used for bivariate analyses.

**Example:**

* Python3

|  |
| --- |
| # importing packages  import seaborn as sns  import matplotlib.pyplot as plt      sns.scatterplot( x="Salary", y='Team', data=df,                  hue='Gender', size='Bonus %')    # Placing Legend outside the Figure  plt.legend(bbox\_to\_anchor=(1, 1), loc=2)    plt.show() |

**Output:**



*Scatter plot of salary and Team column*

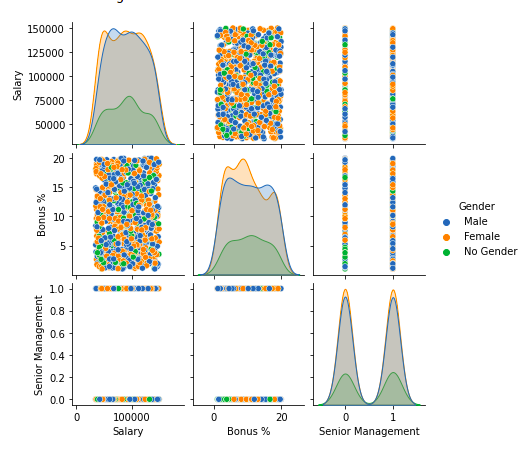
For multivariate analysis, we can use pairplot()method of the seaborn module. We can also use it for the multiple pairwise bivariate distributions in a dataset.

**Example:**

* Python3

|  |
| --- |
| # importing packages  import seaborn as sns  import matplotlib.pyplot as plt      sns.pairplot(df, hue='Gender', height=2) |

**Output:**



*Pairplot of columns of dataframe*

**Handling Outliers**

An [Outlier](https://www.geeksforgeeks.org/machine-learning-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 outliers, and the removal process of these outliers from the dataframe is the same as removing a data item from the panda’s dataframe.

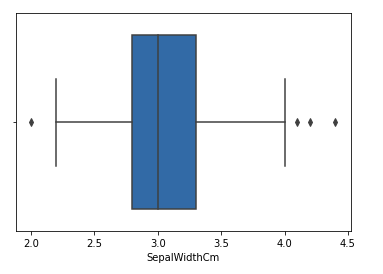
Let’s consider the iris dataset and let’s plot the boxplot for the SepalWidthCm column.

**Example:**

* Python3

|  |
| --- |
| # 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) |

**Output:**



*Boxplot of sample width column before outliers removal*

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

**Removing Outliers**

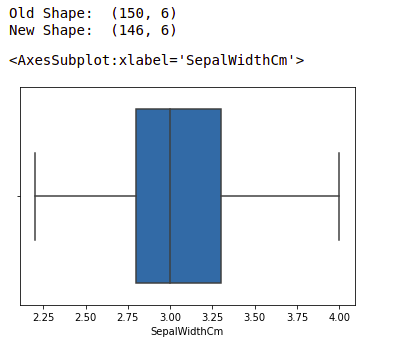
For removing the outlier, one must follow the same process of removing an entry from the dataset using its exact position in the dataset because in all the above methods of detecting the outliers end result is the list of all those data items that satisfy the outlier definition according to the method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

* Python3

|  |
| --- |
| # Importing  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) |

**Output:**



*Boxplot of sample width after outlier removal*

**Note:**for more information, refer[Detect and Remove the Outliers using Python](https://www.geeksforgeeks.org/detect-and-remove-the-outliers-using-python/)

These are some of the EDA we do during our data science project however it depends upon your requirement and how much data analysis we do.

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 projects need to detect the outliers arouse during the data analysis step, the same approach can be used on lists and series-type objects.

**Dataset Used For Outlier Detection**

The dataset used in this article is the Diabetes dataset and it is preloaded in the sklearn library.

* Python3

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

**Output:**

*first five rows of the dataset*

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

**Outliers Visualization**

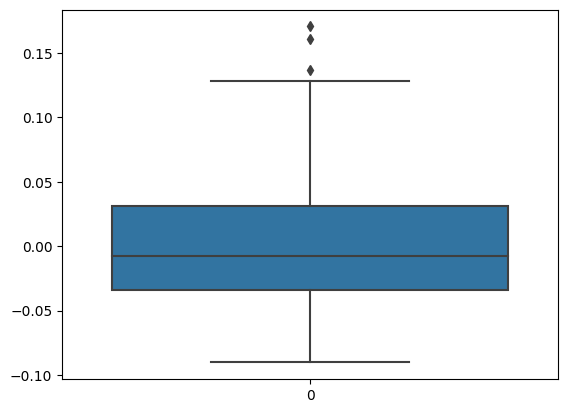
**Visualizing Outliers Using Box Plot**

It captures the summary of the data effectively and efficiently with only a simple box and whiskers. [Boxplot](https://www.geeksforgeeks.org/horizontal-boxplots-with-seaborn-in-python/)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.

* Python3

|  |
| --- |
| # Box Plot  import seaborn as sns  sns.boxplot(df\_diabetics['bmi']) |

**Output**:



*Outliers present in the bmi columns*

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

* Python3

|  |
| --- |
| # Position of the Outlier  import numpy as np  print(np.where(df\_diabetics['bmi']>0.12)) |

**Output:**

(array([ 32, 145, 256, 262, 366, 367, 405]),)

**Visualizing Outliers Using ScatterPlot.**

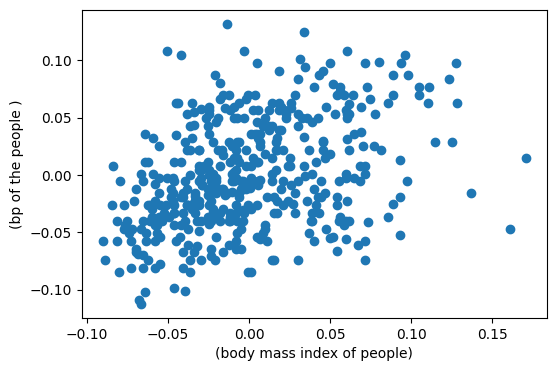
It is used when you have paired numerical data and when your dependent variable has multiple values for each reading independent variable, or when trying to determine the relationship between the two variables. In the process of utilizing the [scatter plot](https://www.geeksforgeeks.org/scatter-plot-in-plotly-using-graph_objects-class/), 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.

* Python3

|  |
| --- |
| # Scatter plot  fig, ax = plt.subplots(figsize = (6,4))  ax.scatter(df\_diabetics['bmi'],df\_diabetics['bp'])    # x-axis label  ax.set\_xlabel('(body mass index of people)')    # y-axis label  ax.set\_ylabel('(bp of the people )')  plt.show() |

**Output**:



*Scatter plot of bp and bmi*

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.

**Outliers in BMI and BP Column Combined**

* Python3

|  |
| --- |
| # Position of the Outlier  print(np.where((df\_diabetics['bmi']>0.12) & (df\_diabetics['bp']<0.8))) |

**Output**:

(array([ 32, 145, 256, 262, 366, 367, 405]),)

**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*

* Python3

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

**Output**:

[0.80050009 0.03956713 1.79330681 1.87244107 0.11317236 1.94881082

0.9560041 1.33508832 0.87686984 1.49059233 2.02518057 0.57139085

0.34228161 0.11317236 0.95323959 1.1087436 0.11593688 1.48782782

0.80326461 0.57415536 1.03237385 1.79607132 1.79607132 0.95323959

1.33785284 1.41422259 2.25428981 0.49778562 1.10597908 1.41145807

1.26148309 0.49778562 0.72413034 0.6477606 0.34228161 1.02960933

0.26591186 0.19230663 0.03956713 0.03956713 0.11317236 2.10155031

1.26148309 0.41865135 0.95323959 0.57139085 1.18511334 1.64333183

1.41145807 0.87963435 0.72413034 1.25871858 1.1087436 0.19230663

1.03237385 0.87963435 0.87963435 0.57415536 0.87686984 1.33508832

1.49059233 0.87963435 0.57415536 0.72689486 1.41145807 0.9560041

0.19230663 0.87686984 0.80050009 0.34228161 0.03956713 0.03956713

1.33508832 0.26591186 0.26591186 0.19230663 0.65052511 2.02518057

0.11317236 2.17792006 1.48782782 0.26591186 0.34504612 0.80326461

0.03680262 0.95323959 1.49059233 0.95323959 1.1087436 0.9560041

0.26591186 0.95323959 0.42141587 1.03237385 1.64333183 1.49059233

1.18234883 0.57415536 0.03680262 0.03956713 0.34228161 0.34228161]

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).

**Rows where Z value is greater than 2**

* Python3

|  |
| --- |
| threshold = 2    # Position of the outlier  print(np.where(z > 2)) |

**Output:**

(array([ 10, 26, 41, 77, 79, 106, 131, 204, 223, 226, 242, 311, 321,344, 374, 402]),)

**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*

* Python3

|  |
| --- |
| # IQR  Q1 = np.percentile(df\_diabetics['bmi'], 25, method='midpoint')  Q3 = np.percentile(df\_diabetics['bmi'], 75, method='midpoint')  IQR = Q3 - Q1  print(IQR) |

**Output:**

0.06520763046978838

Syntax: numpy.percentile(arr, n, axis=None, out=None)   
Parameters :   
arr :input array.  
n : percentile value.

To define the outlier base value is defined above and below dataset’s 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.

* Python3

|  |
| --- |
| # Above Upper bound  upper=Q3+1.5\*IQR  upper\_array=np.array(df\_diabetics['bmi']>=upper)  print("Upper Bound:",upper)  print(upper\_array.sum())    #Below Lower bound  lower=Q1-1.5\*IQR  lower\_array=np.array(df\_diabetics['bmi']<=lower)  print("Lower Bound:",lower)  print(lower\_array.sum()) |

**Output:**

Upper Bound: 0.12879000811776306

3

Lower Bound: -0.13204051376139045

0

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

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

* Python3

|  |
| --- |
| # Importing  import sklearn  from sklearn.datasets import load\_diabetes  import pandas as pd    # Load the dataset  diabetes = load\_diabetes()    # Create the dataframe  column\_name = diabetes.feature\_names  df\_diabetes = pd.DataFrame(diabetes.data)  df\_diabetes .columns = column\_name  df\_diabetes .head()  print("Old Shape: ", df\_diabetes.shape)    ''' Detection '''  # IQR  # Calculate the upper and lower limits  Q1 = df\_diabetes['bmi'].quantile(0.25)  Q3 = df\_diabetes['bmi'].quantile(0.75)  IQR = Q3 - Q1  lower = Q1 - 1.5\*IQR  upper = Q3 + 1.5\*IQR    # Create arrays of Boolean values indicating the outlier rows  upper\_array = np.where(df\_diabetes['bmi']>=upper)[0]  lower\_array = np.where(df\_diabetes['bmi']<=lower)[0]    # Removing the outliers  df\_diabetes.drop(index=upper\_array, inplace=True)  df\_diabetes.drop(index=lower\_array, inplace=True)    # Print the new shape of the DataFrame  print("New Shape: ", df\_diabetes.shape) |

**Output**:

Old Shape: (442, 10)

New Shape: (439, 10)

**Exploratory Data Analysis** is a technique to analyze data with visual techniques and all statistical results. We will learn about how to apply these techniques before applying any Machine Learning Models.

To get the link to csv file used, click [here](https://drive.google.com/open?id=1zhw660ojq4eT6d8TQzCDSL1PloKMQMG6).

**Loading Libraries:**

|  |
| --- |
| import numpy as np  import pandas as pd  import seaborn as sns  import matplotlib.pyplot as plt      from scipy.stats import trim\_mean |

**Loading Data:**

|  |
| --- |
| data = pd.read\_csv("state.csv")    # Check the type of data  print ("Type : ", type(data), "\n\n")    # Printing Top 10 Records  print ("Head -- \n", data.head(10))    # Printing last 10 Records  print ("\n\n Tail -- \n", data.tail(10)) |

**Output :**

Type : class 'pandas.core.frame.DataFrame'

**Head --**

State Population Murder.Rate Abbreviation

0 Alabama 4779736 5.7 AL

1 Alaska 710231 5.6 AK

2 Arizona 6392017 4.7 AZ

3 Arkansas 2915918 5.6 AR

4 California 37253956 4.4 CA

5 Colorado 5029196 2.8 CO

6 Connecticut 3574097 2.4 CT

7 Delaware 897934 5.8 DE

8 Florida 18801310 5.8 FL

9 Georgia 9687653 5.7 GA

**Tail --**

State Population Murder.Rate Abbreviation

40 South Dakota 814180 2.3 SD

41 Tennessee 6346105 5.7 TN

42 Texas 25145561 4.4 TX

43 Utah 2763885 2.3 UT

44 Vermont 625741 1.6 VT

45 Virginia 8001024 4.1 VA

46 Washington 6724540 2.5 WA

47 West Virginia 1852994 4.0 WV

48 Wisconsin 5686986 2.9 WI

49 Wyoming 563626 2.7 WY

**Code #1 :** Adding Column to the dataframe

|  |
| --- |
| # Adding a new column with derived data    data['PopulationInMillions'] = data['Population']/1000000    # Changed data  print (data.head(5)) |

**Output :**

State Population Murder.Rate Abbreviation PopulationInMillions

0 Alabama 4779736 5.7 AL 4.779736

1 Alaska 710231 5.6 AK 0.710231

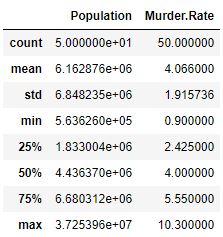
2 Arizona 6392017 4.7 AZ 6.392017

3 Arkansas 2915918 5.6 AR 2.915918

4 California 37253956 4.4 CA 37.253956

**Code #2 :** Data Description

|  |
| --- |
| data.describe() |

**Output :**  


**Code #3 :** Data Info

|  |
| --- |
| data.info() |

**Output :**

RangeIndex: 50 entries, 0 to 49

Data columns (total 4 columns):

State 50 non-null object

Population 50 non-null int64

Murder.Rate 50 non-null float64

Abbreviation 50 non-null object

dtypes: float64(1), int64(1), object(2)

memory usage: 1.6+ KB

**Code #4 :**Renaming a column heading

|  |
| --- |
| # Rename column heading as it  # has '.' in it which will create  # problems when dealing functions    data.rename(columns ={'Murder.Rate': 'MurderRate'}, inplace = True)    # Lets check the column headings  list(data) |

**Output :**

['State', 'Population', 'MurderRate', 'Abbreviation']

**Code #5 :** Calculating Mean

|  |
| --- |
| Population\_mean = data.Population.mean()  print ("Population Mean : ", Population\_mean)    MurderRate\_mean = data.MurderRate.mean()  print ("\nMurderRate Mean : ", MurderRate\_mean) |

**Output:**

Population Mean : 6162876.3

MurderRate Mean : 4.066

**Code #6 :** Trimmed mean

|  |
| --- |
| # Mean after discarding top and  # bottom 10 % values eliminating outliers    population\_TM = trim\_mean(data.Population, 0.1)  print ("Population trimmed mean: ", population\_TM)    murder\_TM = trim\_mean(data.MurderRate, 0.1)  print ("\nMurderRate trimmed mean: ", murder\_TM) |

**Output :**

Population trimmed mean: 4783697.125

MurderRate trimmed mean: 3.9450000000000003

**Code #7 :** Weighted Mean

|  |
| --- |
| # here murder rate is weighed as per  # the state population    murderRate\_WM = np.average(data.MurderRate, weights = data.Population)  print ("Weighted MurderRate Mean: ", murderRate\_WM) |

**Output :**

Weighted MurderRate Mean: 4.445833981123393

**Code #8 :** Median

|  |
| --- |
| Population\_median = data.Population.median()  print ("Population median : ", Population\_median)    MurderRate\_median = data.MurderRate.median()  print ("\nMurderRate median : ", MurderRate\_median) |

**Output :**

Population median : 4436369.5

MurderRate median : 4.0

In the [previous article](https://www.geeksforgeeks.org/exploratory-data-analysis-in-python-set-1/), we have discussed some basic techniques to analyze the data, now let’s see the visual techniques.

Let’s see the basic techniques –

|  |
| --- |
| # Loading Libraries    import numpy as np  import pandas as pd  import seaborn as sns  import matplotlib.pyplot as plt  from scipy.stats import trim\_mean    # Loading Data  data = pd.read\_csv("state.csv")    # Check the type of data  print ("Type : ", type(data), "\n\n")    # Printing Top 10 Records  print ("Head -- \n", data.head(10))    # Printing last 10 Records  print ("\n\n Tail -- \n", data.tail(10))    # Adding a new column with derived data  data['PopulationInMillions'] = data['Population']/1000000    # Changed data  print (data.head(5))    # Rename column heading as it  # has '.' in it which will create  # problems when dealing functions    data.rename(columns ={'Murder.Rate': 'MurderRate'},                                      inplace = True)    # Lets check the column headings  list(data) |

**Output :**

Type : class 'pandas.core.frame.DataFrame'

**Head --**

State Population Murder.Rate Abbreviation

0 Alabama 4779736 5.7 AL

1 Alaska 710231 5.6 AK

2 Arizona 6392017 4.7 AZ

3 Arkansas 2915918 5.6 AR

4 California 37253956 4.4 CA

5 Colorado 5029196 2.8 CO

6 Connecticut 3574097 2.4 CT

7 Delaware 897934 5.8 DE

8 Florida 18801310 5.8 FL

9 Georgia 9687653 5.7 GA

**Tail --**

State Population Murder.Rate Abbreviation

40 South Dakota 814180 2.3 SD

41 Tennessee 6346105 5.7 TN

42 Texas 25145561 4.4 TX

43 Utah 2763885 2.3 UT

44 Vermont 625741 1.6 VT

45 Virginia 8001024 4.1 VA

46 Washington 6724540 2.5 WA

47 West Virginia 1852994 4.0 WV

48 Wisconsin 5686986 2.9 WI

49 Wyoming 563626 2.7 WY

State Population Murder.Rate Abbreviation PopulationInMillions

0 Alabama 4779736 5.7 AL 4.779736

1 Alaska 710231 5.6 AK 0.710231

2 Arizona 6392017 4.7 AZ 6.392017

3 Arkansas 2915918 5.6 AR 2.915918

4 California 37253956 4.4 CA 37.253956

['State', 'Population', 'MurderRate', 'Abbreviation']

**Visualizing Population per Million**

|  |
| --- |
| # Plot Population In Millions  fig, ax1 = plt.subplots()  fig.set\_size\_inches(15,  9)      ax1 = sns.barplot(x ="State", y ="Population",                    data = data.sort\_values('MurderRate'),                                          palette ="Set2")    ax1.set(xlabel ='States', ylabel ='Population In Millions')  ax1.set\_title('Population in Millions by State', size = 20)    plt.xticks(rotation =-90) |

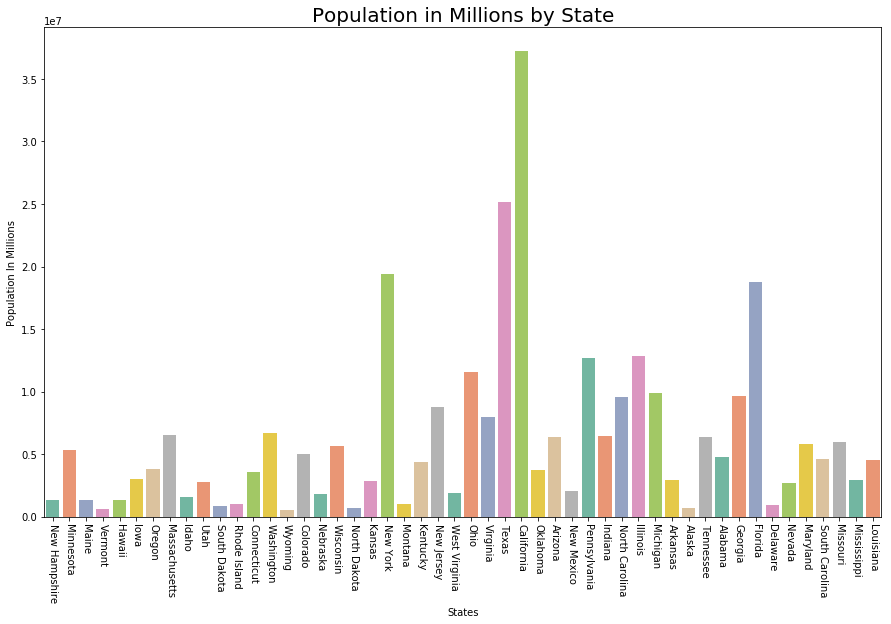
**Output:**

(array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,

17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,

34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49]),

a list of 50 Text xticklabel objects)



**Visualizing Murder Rate per Lakh**

|  |
| --- |
| # Plot Murder Rate per 1, 00, 000    fig, ax2 = plt.subplots()  fig.set\_size\_inches(15,  9)      ax2 = sns.barplot(      x ="State", y ="MurderRate",      data = data.sort\_values('MurderRate', ascending = 1),                                           palette ="husl")    ax2.set(xlabel ='States', ylabel ='Murder Rate per 100000')  ax2.set\_title('Murder Rate by State', size = 20)    plt.xticks(rotation =-90) |

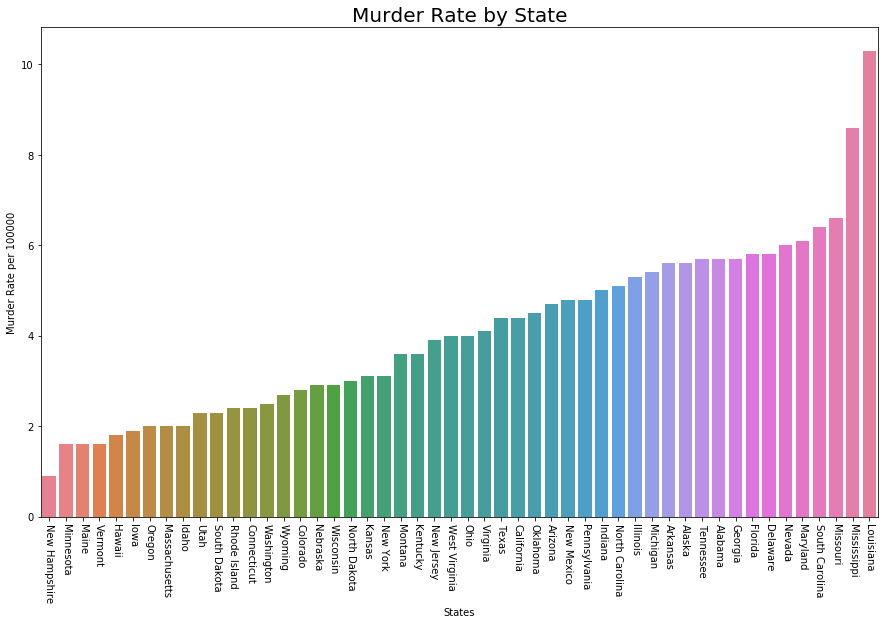
**Output :**

(array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,

17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,

34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49]),

a list of 50 Text xticklabel objects)

  
Although Louisiana is ranked 17 by population (about 4.53M), it has the highest Murder rate of 10.3 per 1M people.

**Code #1 :** Standard Deviation

|  |
| --- |
| Population\_std = data.Population.std()  print ("Population std : ", Population\_std)    MurderRate\_std = data.MurderRate.std()  print ("\nMurderRate std : ", MurderRate\_std) |

**Output :**

Population std : 6848235.347401142

MurderRate std : 1.915736124302923

**Code #2 :** Variance

|  |
| --- |
| Population\_var = data.Population.var()  print ("Population var : ", Population\_var)    MurderRate\_var = data.MurderRate.var()  print ("\nMurderRate var : ", MurderRate\_var) |

**Output :**

Population var : 46898327373394.445

MurderRate var : 3.670044897959184

**Code #3 :** Inter Quartile Range

|  |
| --- |
| # Inter Quartile Range of Population  population\_IQR = data.Population.describe()['75 %'] -                   data.Population.describe()['25 %']    print ("Population IQR : ", population\_IRQ)    # Inter Quartile Range of Murder Rate  MurderRate\_IQR = data.MurderRate.describe()['75 %'] -                   data.MurderRate.describe()['25 %']    print ("\nMurderRate IQR : ", MurderRate\_IQR) |

**Output :**

Population IQR : 4847308.0

MurderRate IQR : 3.124999999999999

**Code #4 :** Median Absolute Deviation (MAD)

|  |
| --- |
| Population\_mad = data.Population.mad()  print ("Population mad : ", Population\_mad)    MurderRate\_mad = data.MurderRate.mad()  print ("\nMurderRate mad : ", MurderRate\_mad) |

**Output :**

Population mad : 4450933.356000001

MurderRate mad : 1.5526400000000005

**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**

If you are from a data science background you all must be familiar with the Iris Dataset. If you are not then don’t worry we will discuss this here.

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.

**Note:**This dataset can be downloaded from [here](https://datahub.io/machine-learning/iris).

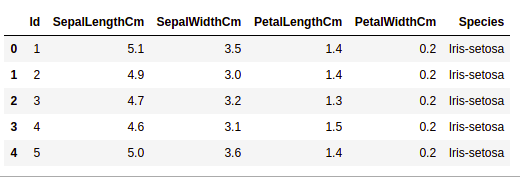
You can download the Iris.csv file from the above link. Now we will use the Pandas library to load this CSV file, and we will convert it into the [dataframe](https://www.geeksforgeeks.org/python-pandas-dataframe/). [read\_csv()](https://www.geeksforgeeks.org/python-read-csv-using-pandas-read_csv/) method is used to read CSV files.

**Example:**

* Python3

|  |
| --- |
| import pandas as pd    # Reading the CSV file  df = pd.read\_csv("Iris.csv")    # Printing top 5 rows  df.head() |

**Output:**



**Getting Information about the Dataset**

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

**Example:**

* Python3

|  |
| --- |
| df.shape |

**Output:**

(150, 6)

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

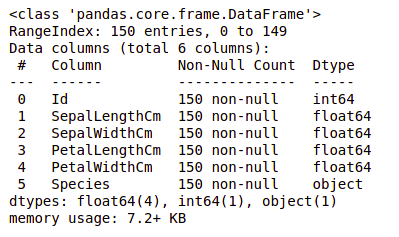
Now, let’s also the columns and their data types. For this, we will use the [info()](https://www.geeksforgeeks.org/python-pandas-dataframe-info/) method.

**Example:**

* Python3

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

**Output:**



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

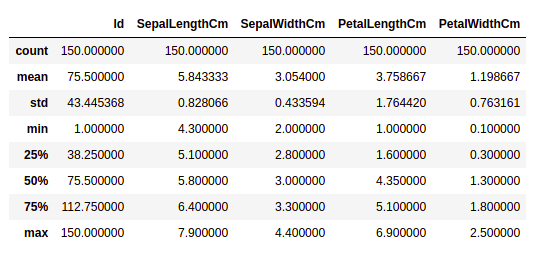
Let’s get a quick statistical summary of the dataset using the [**describe()**](https://www.geeksforgeeks.org/python-pandas-dataframe-describe-method/) method. The describe() function applies basic statistical computations on the dataset like extreme values, count of data points standard deviation, etc.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

**Example:**

* Python3

|  |
| --- |
| df.describe() |

**Output:**



We can see the count of each column along with their mean value, standard deviation, minimum and maximum values.

**Checking Missing Values**

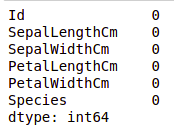
We will check if our data contains any missing values or not. Missing values can occur when no information is provided for one or more items or for a whole unit. We will use the [isnull()](https://www.geeksforgeeks.org/python-pandas-isnull-and-notnull/) method.

**Example:**

* Python3

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

**Output:**



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**

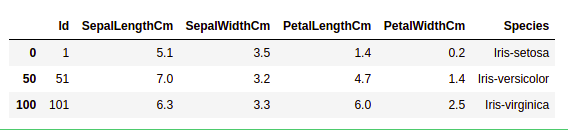
Let’s see if our dataset contains any duplicates or not. Pandas [drop\_duplicates()](https://www.geeksforgeeks.org/python-pandas-dataframe-drop_duplicates/) method helps in removing duplicates from the data frame.

**Example:**

* Python3

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

**Output:**



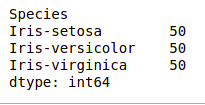
We can see that there are only three unique species. Let’s see if the dataset is balanced or not i.e. all the species contain equal amounts of rows or not. We will use the [Series.value\_counts()](https://www.geeksforgeeks.org/python-pandas-series-value_counts/) function. This function returns a Series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

**Example:**

* Python3

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

**Output:**



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 –*

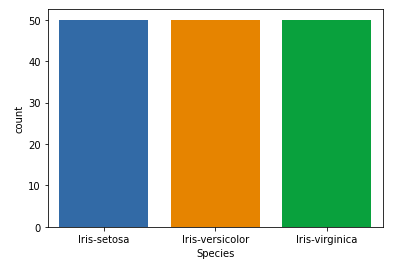
* [*Matplotlib Tutorial*](https://www.geeksforgeeks.org/matplotlib-tutorial/)
* [*Python Seaborn Tutorial*](https://www.geeksforgeeks.org/python-seaborn-tutorial/)

**Example:**

* Python3

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

**Output:**



**Relation between variables**

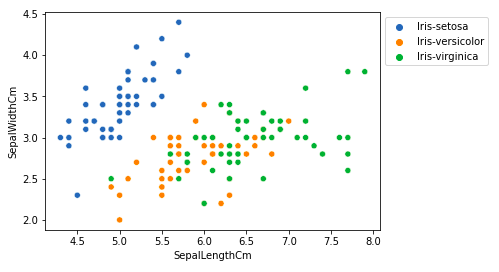
We will see the relationship between the sepal length and sepal width and also between petal length and petal width.

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

* Python3

|  |
| --- |
| # 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() |

**Output:**



From the above plot, we can infer that –

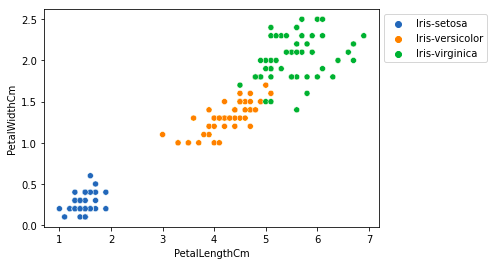
* Species Setosa has smaller sepal lengths but larger sepal widths.
* Versicolor Species lies in the middle of the other two species in terms of sepal length and width
* Species Virginica has larger sepal lengths but smaller sepal widths.

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

* Python3

|  |
| --- |
| # 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() |

**Output:**



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.

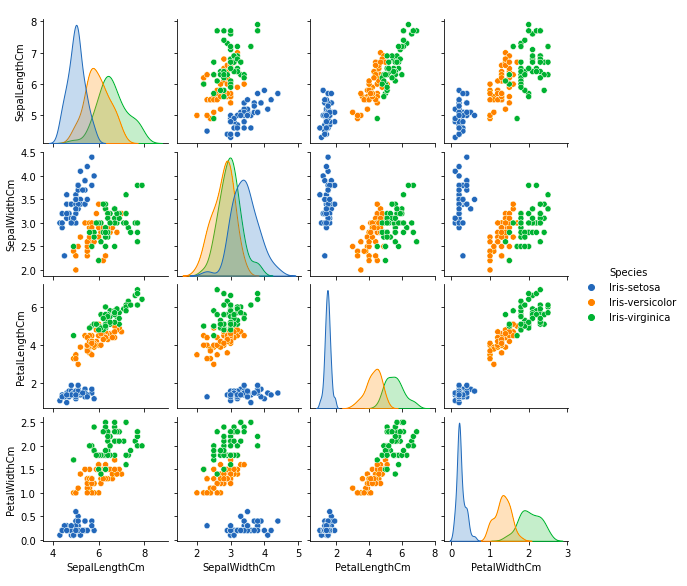
Let’s plot all the column’s relationships using a pairplot. It can be used for multivariate analysis.

**Example:**

* Python3

|  |
| --- |
| # importing packages  import seaborn as sns  import matplotlib.pyplot as plt      sns.pairplot(df.drop(['Id'], axis = 1),               hue='Species', height=2) |

**Output:**



We can see many types of relationships from this plot such as the species Setosa 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**

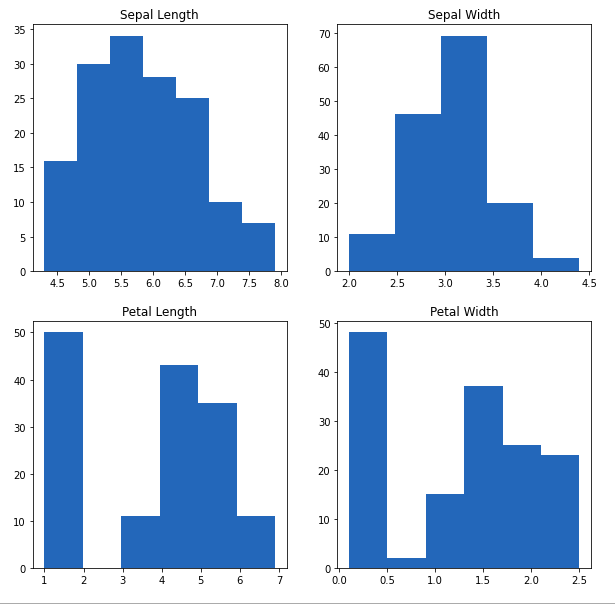
Histograms allow seeing the distribution of data for various columns. It can be used for uni as well as bi-variate analysis.

**Example:**

* Python3

|  |
| --- |
| # 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); |

**Output:**



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**

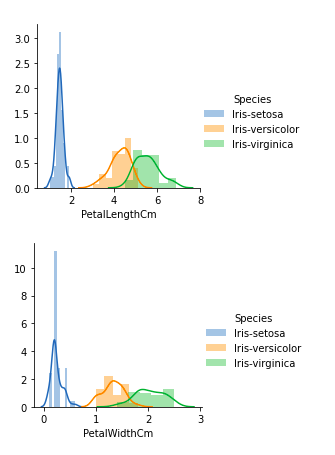
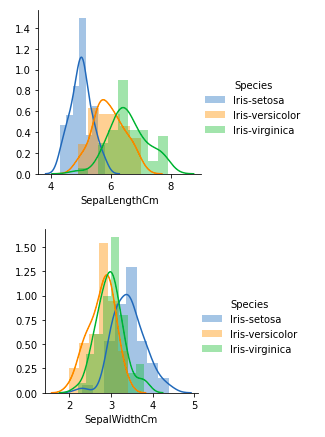
Distplot is used basically for the univariant set of observations and visualizes it through a histogram i.e. only one observation and hence we choose one particular column of the dataset.

**Example:**

* Python3

|  |
| --- |
| # importing packages  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**

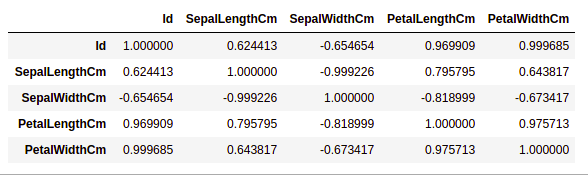
Pandas [dataframe.corr()](https://www.geeksforgeeks.org/python-pandas-dataframe-corr/) is used to find the pairwise correlation of all columns in the dataframe. Any NA values are automatically excluded. For any non-numeric data type columns in the dataframe it is ignored.

**Example:**

* Python3

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

**Output:**



**Heatmaps**

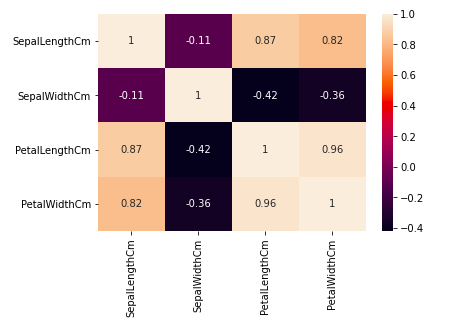
The heatmap is a data visualization technique that is used to analyze the dataset as colors in two dimensions. Basically, it shows a correlation between all numerical variables in the dataset. In simpler terms, we can plot the above-found correlation using the heatmaps.

**Example:**

* Python3

|  |
| --- |
| # 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() |

**Output:**



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**

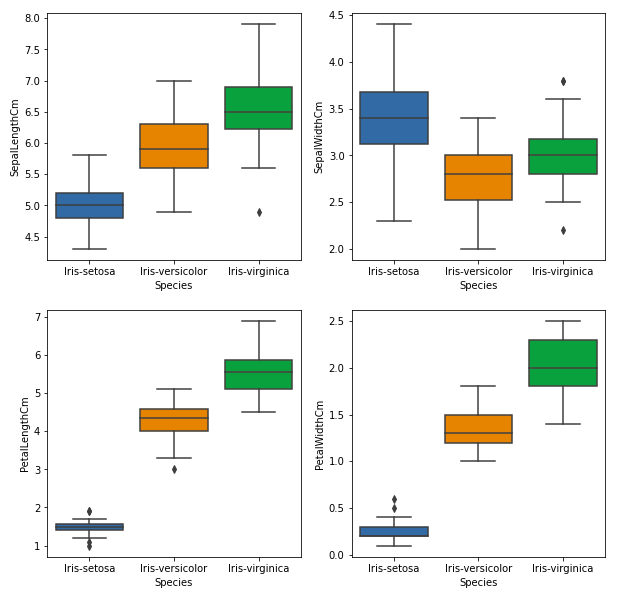
We can use boxplots to see how the categorical value os distributed with other numerical values.

**Example:**

* Python3

|  |
| --- |
| # 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() |

**Output:**



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.

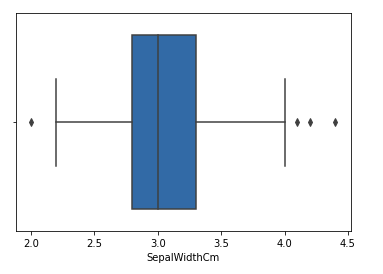
Let’s consider the iris dataset and let’s plot the boxplot for the SepalWidthCm column.

**Example:**

* Python3

|  |
| --- |
| # 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) |

**Output:**



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.

* Python3

|  |
| --- |
| # Importing  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) |

**Output:**

