Experiment 2: Programs to perform exploratory data analysis, variance, standard deviation, summarization, distribution, statistical inference.

What is Exploratory Data Analysis (EDA)?
In simple words: EDA is a process or approach to finding out the most useful features from a dataset according to your problem which helps you to choose the correct and efficient algorithm for your solution.

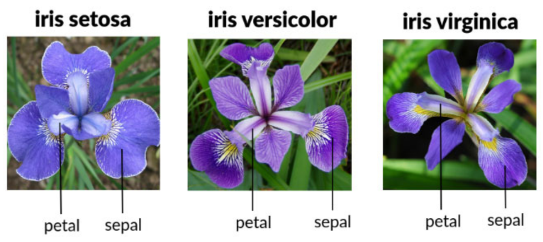
Why is Exploratory Data Analysis Important?
As we know there are some prerequisites for every project and we can say that EDA is a prerequisite process for any data science or machine learning project. If you are in a hurry and skip EDA then you may face some outliers and too many missing values and, therefore, some bad outcomes for the project:

1. Wrong model
2. Right model on wrong data
3. Selection of wrong features for model building.

How to do Exploratory Data Analysis?
We can do EDA with programming languages and data visualization tools.
The most popular programming languages are:
1. Python
2. R

The most popular data visualization tools are:
1. Tableau
2. Power BI
3. Infogram
4. Plotly

Let's dive into the Iris Flower dataset:



To download the Iris flower dataset: https://www.kaggle.com/arshid/iris-flower-dataset

Key points about the dataset:
1. The shape of data is (150 * 4) which means rows are 150 and columns are 4 and these columns are named sepal length, sepal width, petal length, and petal width.
2. There is a species column that tells us about the label of the flower according to the given data there are three categories of flower named Iris setosa, Iris verginica, and Iris versicolor.
3. Dataset is having 33% of each category's data.

Now we are going to do EDA with the programming language named Python. Python has a massive number of libraries to apply the various types of operations on data to find out the best results.

Import some important Libraries.

```
# Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
```

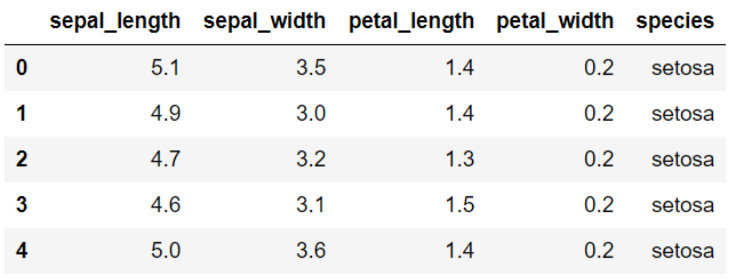
The next step is to load data. if data is in CSV format, then use these lines of code to load data into a variable named Data.

```
iris = pd.read_csv("iris.csv")
```

To observe the top 5 rows of the data using the head function which gives the first 5 rows of the data.

```
print(iris.head())
```

Output:



	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

To check the last five rows of the data we can use **data.tail()** and for modifying the number of rows we can use **data.head(3)** or **data.tail(3)**.

To check the dimensionality of the dataset

```
print(iris.shape)
```

Output: The shape of the data is (150, 4)

To check the column names or feature names

```
print(iris.columns)
```

Output: Index([‘sepal_length’, ‘sepal_width’, ‘petal_length’, ‘petal_width’, ‘species’], dtype = ‘object’)

To check how many points of each class

print(iris['species'].value_counts())

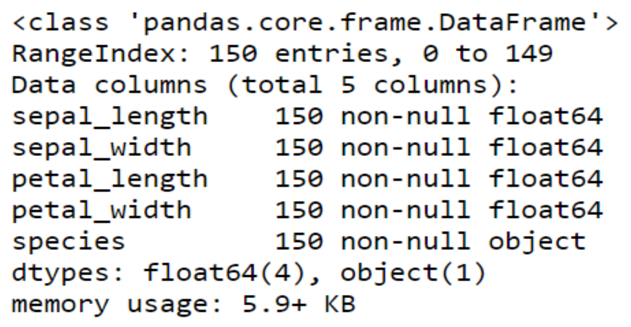
Output: versicolor: 50, setosa: 50, virginica: 50

As we can observe all three classes are equally distributed in terms of the number of counts of each class. Here we can understand a very interesting concept called balanced and imbalanced dataset.

Some Basic information about the dataset

print(iris.info())

Output:



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
sepal_length    150 non-null float64
sepal_width     150 non-null float64
petal_length    150 non-null float64
petal_width     150 non-null float64
species         150 non-null object
dtypes: float64(4), object(1)
memory usage: 5.9+ KB
```

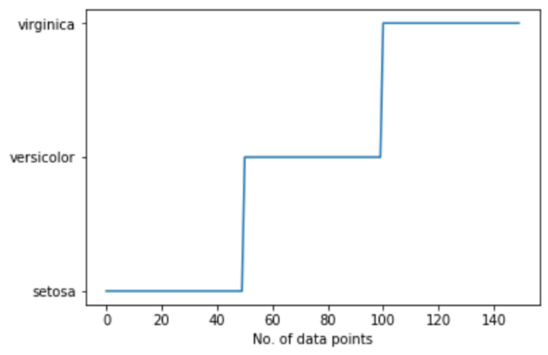
Observation:

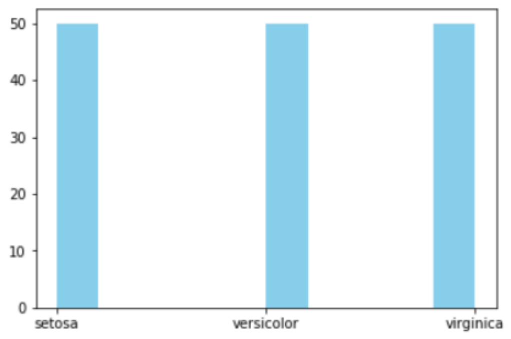
Now we can say that there is not any data point missing in any feature. And all first 4 features are of float64 type which is used in numpy and pandas. And indexes are from 0 to 149 for 150 entries. And the last column is of type object which is used for the class labels. The memory used by this data frame is around 6 KB.

Some Visual views of data

```
# First plot
plt.plot(iris["species"])
plt.xlabel("No. of data points")
plt.show()

# Second plot
plt.hist(iris["species"],color="green")
plt.show()
```

Output:





Here we see visually by plotting a graph by no. of data points of each class label.

To observe more about the data we can see a basic description of the data

print(iris.describe())

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

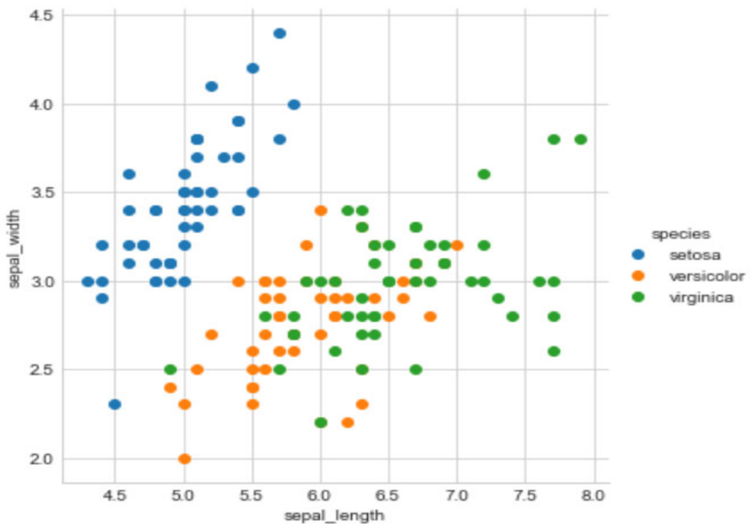
Observation:

By describing the dataset, we can find out the overall mean, standard deviation, minimum and maximum values in each feature, 25, 50, 75 percentile of data distribution. And many more things in another dataset. We can't find any useful information in this dataset that tells us about a very useful feature that helps classify all three iris, and flower classes. To do that we can apply some other techniques to find out important features such as plots of various types.

Scatter plot

A scatter plot is a set of points plotted on horizontal and vertical axes. Scatter plots are important in statistics because they can show the extent of correlation, if any, between the values of observed quantities or phenomena (called variables). If no correlation exists between the variables, the points appear randomly scattered on the coordinate plane. If a large correlation exists, the points concentrate near a straight line. This has come under bivariate analysis.

```
# plotting a scatterplot using seaborn
sb.set_style('whitegrid')
sb.scatterplot(data=iris, x='sepal_length', y='sepal_width', hue='species')
plt.plot()
```



Observation:

Wow! Here we can see that blue color data points (setosa) are totally separate from the other two class data points by a straight line when we plot a scatter plot with the help of sepal length and sepal width. But the separation of Versicolor from Virginica is much harder because they are considered overlaps. To solve this problem, we have to observe many other plots.

Here we have a scatter plot with only two features but we have to observe all pairs of the features (C_2^4= 6) combinations to see all features in 2D space which is not a good practice and effective by doing one by one. So, we have good news we can do it by a single line of code with a pair plot.

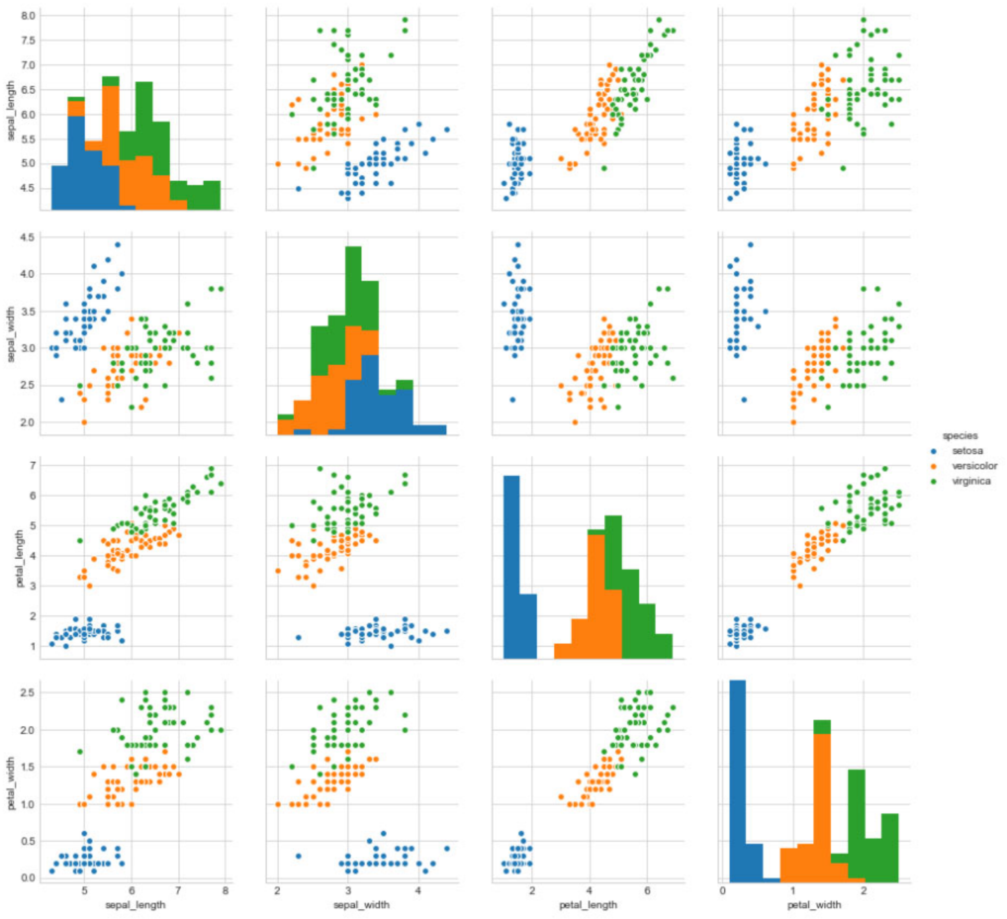
Pair plot

By default, this function will create a grid of Axes such that each variable in data will be shared in the y-axis across a single row and in the x-axis across a single column. The diagonal Axes are treated differently, drawing a plot to show the univariate distribution of the data for the variable in that column.

Disadvantage: We cannot visualize higher dimension data like 10D data. It is perfect for up to 6D data. And work only in 2D data. If you want to observe, you have to use 3D scatter plot.
This has come under bivariate data analysis.

```
sb.set_style('whitegrid')
sb.pairplot(iris,hue='species',size=3)
plt.show()
```

Output:



Observation:

We can observe from the pair plot that petal length and petal width are the most useful features to classify iris flower to their respective class. Verginica and Versicolor are a little bit overlapped but they are almost linearly separable.

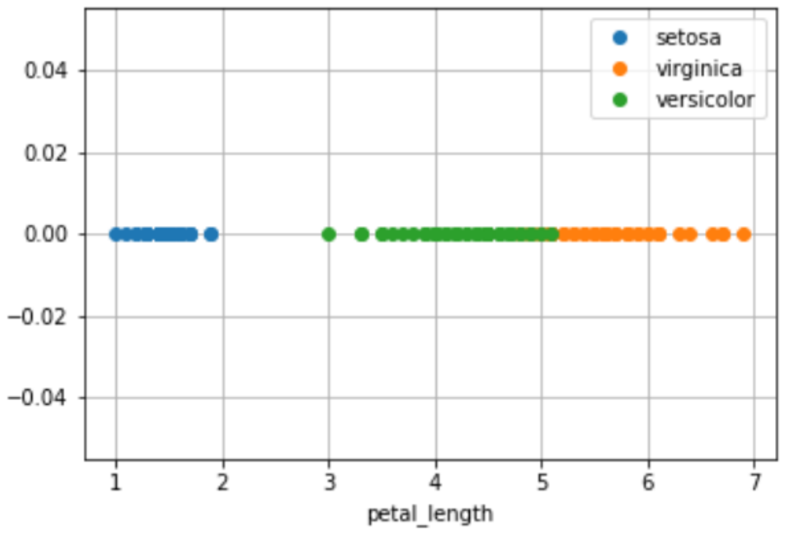
Now we are going to study PDF, CDF, and HISTOGRAM.

Why do we need a histogram?

```
iris_setosa = iris.loc[iris['species']=='setosa']
iris_versicolor = iris.loc[iris['species']=='versicolor']
iris_virginica = iris.loc[iris['species']=='virginica']
```

```
plt.plot(iris_setosa['petal_length'],np.zeros_like(iris_setosa['petal_length'],),'o',label='setosa')
plt.plot(iris_virginica['petal_length'],np.zeros_like(iris_virginica['petal_length'],),'o',label='vir
ginica')
plt.plot(iris_versicolor['petal_length'],np.zeros_like(iris_versicolor['petal_length'],),'o',label='
versicolor')

plt.xlabel("petal_length")
plt.grid()
plt.legend()
plt.show()
```

Output:



Observation:

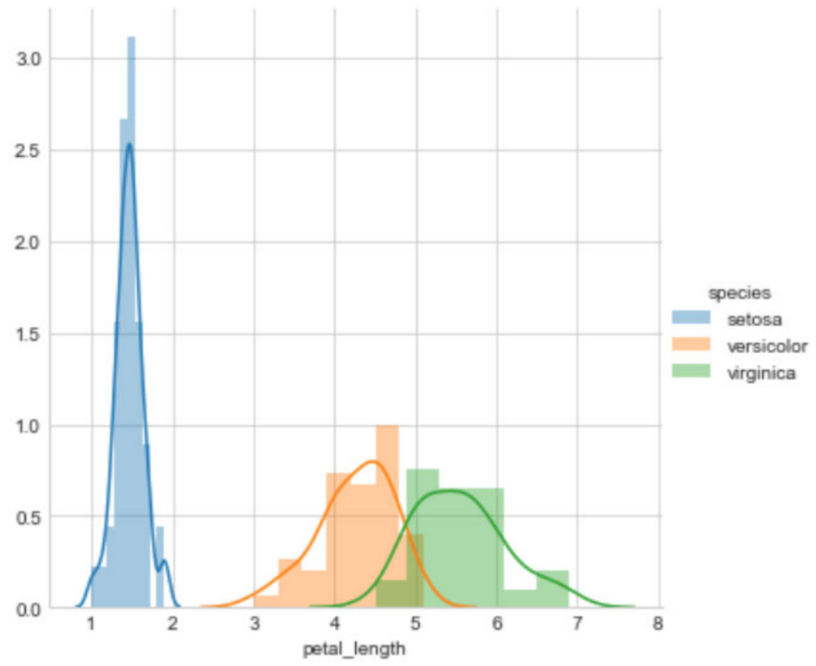
The disadvantage of this 1D scatter plot is a lot of overlap between Versicolor and Verginica and we cannot say anything about it. To overcome this problem, we need a histogram.

Histogram and Density Curve on the same plot

If you wish to have both the histogram and densities in the same plot, the seaborn package (imported as sb) allows you to do that via the distplot(). Since Seaborn is built on top of matplotlib, you can use the sb and plt one after the other. This has come under univariate data analysis.

```
sb.FacetGrid(iris,hue="species").map(sb.distplot,'petal_length').add_legend()
plt.show()
```

Output:



Observation:

Its X-axis tells us all setosa flowers are having a petal length between 1 and 1.8. And Versicolor has petal lengths between 3 and 5.2 and Virginica have petal length between 4.5 and 6.9.
Its Y-axis tells us the count of the flower at this x value. or how often they come at this value.
And Setosa is fully separated from the other two classes but Versicolor and Virginica are not fully separated they have some overlap of some data points.
At x = 5 there is a high probability to get Virginica rather than Versicolor because the height of the Virginica histogram if larger than Versicolor.
And this smooth curve is called PDF (Probability Density Function). It is a smooth histogram.

PDF (Probability Density Function)

The PDF is the density of probability rather than the probability mass. The concept is very similar to mass density in physics.

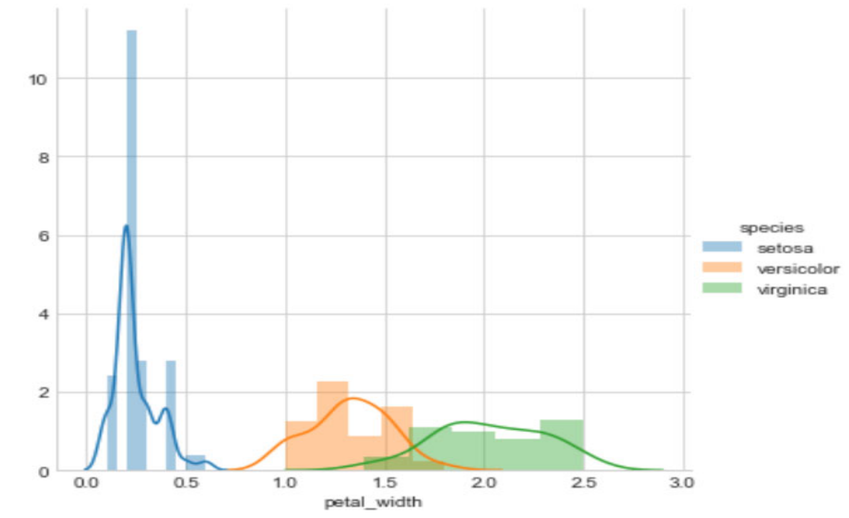
OR

The probability density function (PDF) is a statistical expression that defines probability distribution as a continuous random variable as opposed to a discrete random variable. When the PDF is graphically portrayed, the area under the curve will indicate the interval in which the variable will fall. The total area in this interval of the graph equals the probability of a continuous random variable occurring.

For other features

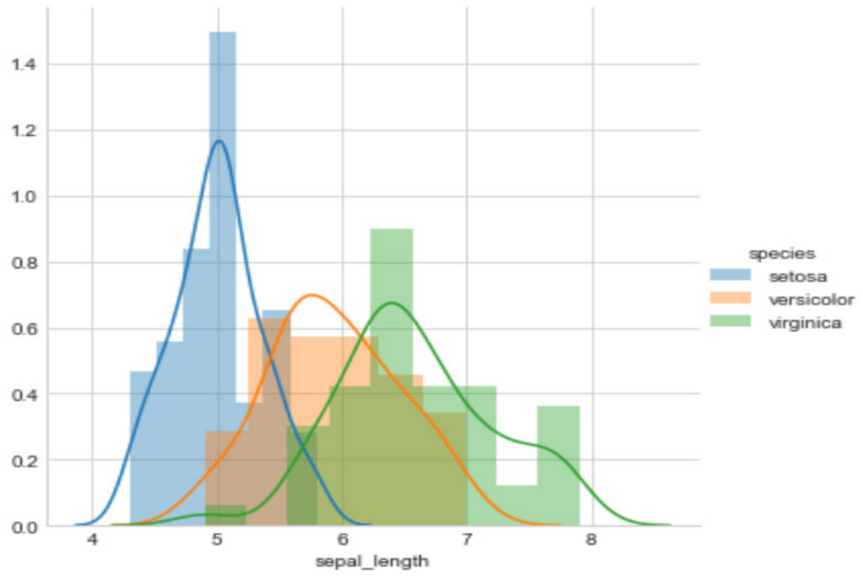
```
sb.FacetGrid(iris,hue="species",size=5).map(sb.distplot, 'petal_width').add_legend()
plt.show()
```

Output:



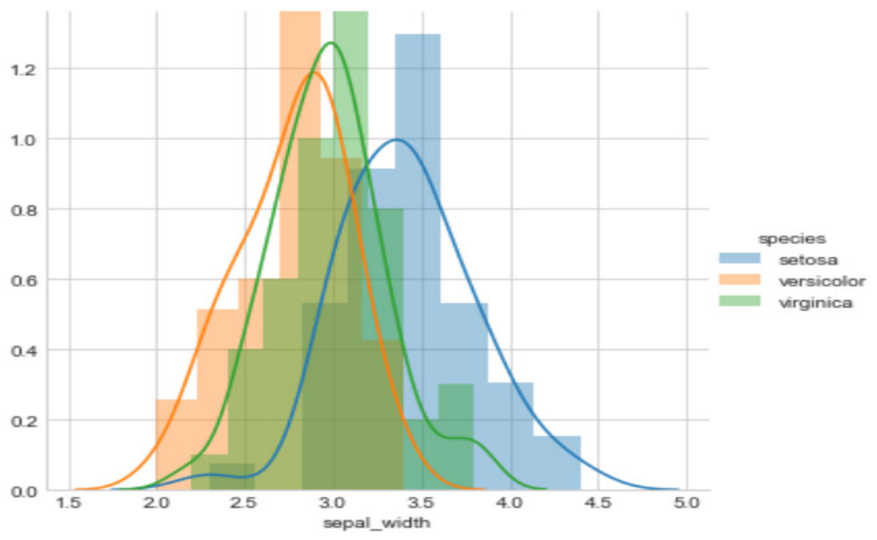
```
sb.FacetGrid(iris,hue="species").map(sb.distplot, 'sepal_length').add_legend()
plt.show()
```

Output:



```
sb.FacetGrid(iris,hue="species").map(sb.distplot, 'sepal_width').add_legend()
plt.show()
```

Output:



Observation:

Petal length will do a great job to classify all classes of flowers but we can say that petal length is slightly better than the petal width because petal width is also doing a better job. Sepal length and sepal width are not good features to classify iris flowers

petal length > petal width >>> sepal length >>>sepal width.

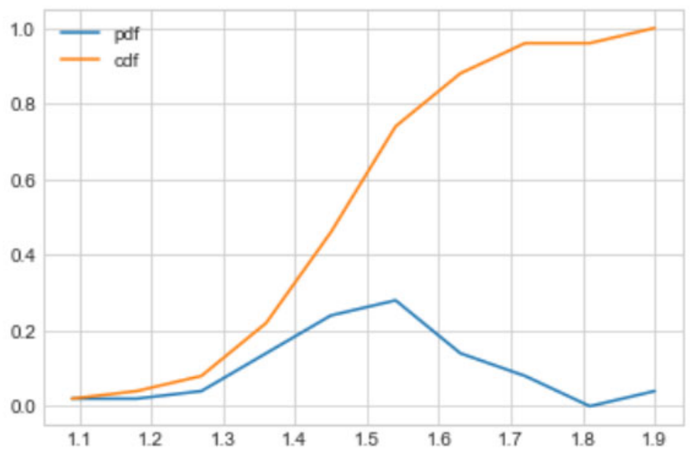
CDF (Cumulative distribution function)

In probability theory and statistics, the cumulative distribution function (CDF) of a real-valued random variable X, or just distribution function of X, evaluated at x, is the probability that X will take a value less than or equal to x.

To plot a CDF and PDF

```
counts,bin_edges=np.histogram(iris_setosa['petal_length'],bins=10,density=True)
pdf= counts/(sum(counts))
print(pdf)
print(bin_edges)

cdf=np.cumsum(pdf)

plt.plot(bin_edges[1:],pdf,label='pdf')
plt.plot(bin_edges[1:],cdf,label='cdf')

plt.legend()
plt.show()
```

Output:



This is only for the setosa class petal length.

Observation:

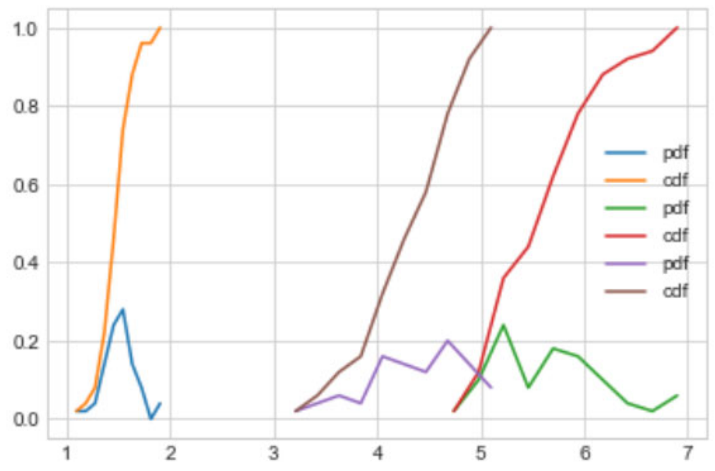
Let's take petal length is equals to 1.5 and then observe it by CDF and PDF. Now we can say that 61% of setosa flowers having petal length is less than 1.5 or in another way we can say between 1.4 and 1.5 petal lengths we have 28% setosa flowers.

To plot PDF and CDF for all class

```
counts,bin_edges=np.histogram(iris_setosa['petal_length'],bins=10,density=True)
pdf= counts/(sum(counts))
print(pdf)
print(bin_edges)

# to compute cdf
cdf=np.cumsum(pdf)

plt.plot(bin_edges[1:],pdf,label='pdf')
plt.plot(bin_edges[1:],cdf,label='cdf')


counts,bin_edges=np.histogram(iris_virginica['petal_length'],bins=10,density=True)
pdf= counts/(sum(counts))
print(pdf)
print(bin_edges)

# to compute cdf
cdf=np.cumsum(pdf)

plt.plot(bin_edges[1:],pdf,label='pdf')
plt.plot(bin_edges[1:],cdf,label='cdf')

counts,bin_edges=np.histogram(iris_versicolor['petal_length'],bins=10,density=True)
```

```
pdf= counts/(sum(counts))
print(pdf)
print(bin_edges)

# to compute cdf
cdf=np.cumsum(pdf)
plt.plot(bin_edges[1:],pdf,label='pdf')
plt.plot(bin_edges[1:],cdf,label='cdf')
plt.legend()
plt.show()
```

Output:



Mean, variance, and standard deviation

```
# Means of the petal length
print("Means")
print("setosa",np.mean(iris_setosa["petal_length"]))
print("versicolor",np.mean(iris_versicolor["petal_length"]))
print("virginica",np.mean(iris_virginica["petal_length"]))

# standard-daviation
print("standard-daviation")
print("setosa      :",np.std(iris_setosa["petal_length"]))
print("versicolor  :",np.std(iris_versicolor["petal_length"]))
print("virginica   :",np.std(iris_virginica["petal_length"]))

# median
print("median")
print("setosa      :",np.median(iris_setosa["petal_length"]))
print("versicolor  :",np.median (iris_versicolor["petal_length"]))
print("virginica   :",np.median (iris_virginica["petal_length"]))
```

Observation:

Here if we talk about variance then this is a spread of how far our elements are spread (width of histogram's graph). We can observe various things by only one or two plots and the names of those plots are BOX plot and VIOLIN plot. but to understand these plots we have to understand the basic concept of percentile.

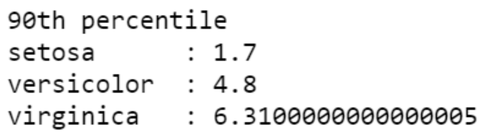
Percentile

x percentile tells us what % of data points are smaller than this value and what % of elements are greater than this value.

1. The 50th percentile is the median.
2. 25th,50th,75th, and 100th percentiles are called quantiles.
3. 25th is a first, 50th is a second, 75th is a third, and 100th is the fourth quantile.

```
print("90th percentile")
print("setosa:",np.percentile(iris_setosa["petal_length"],90))
print("versicolor:",np.percentile(iris_versicolor["petal_length"],90))
print("virginica:",np.percentile(iris_virginica["petal_length"],90))
```

Output:

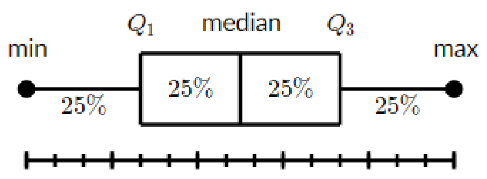


```
90th percentile
setosa      : 1.7
versicolor  : 4.8
virginica   : 6.3100000000000005
```

What is a box and whisker plot?

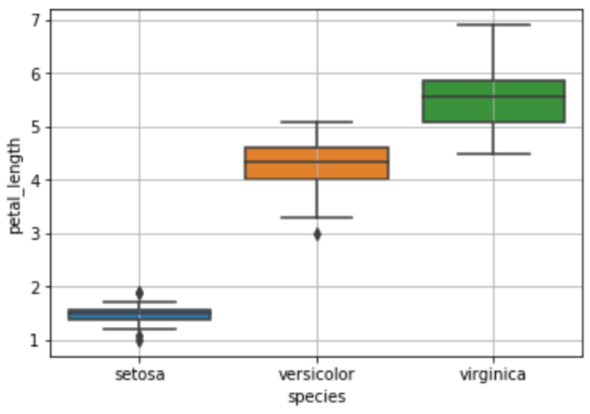
A box and whisker plot — also called a box plot — displays the five-number summary of a set of data. The five-number summary is the minimum, first quartile, median, third quartile, and maximum.

In a box plot, we draw a box from the first quartile to the third quartile. A vertical line goes through the box at the median. The whiskers go from each quartile to the minimum or maximum.



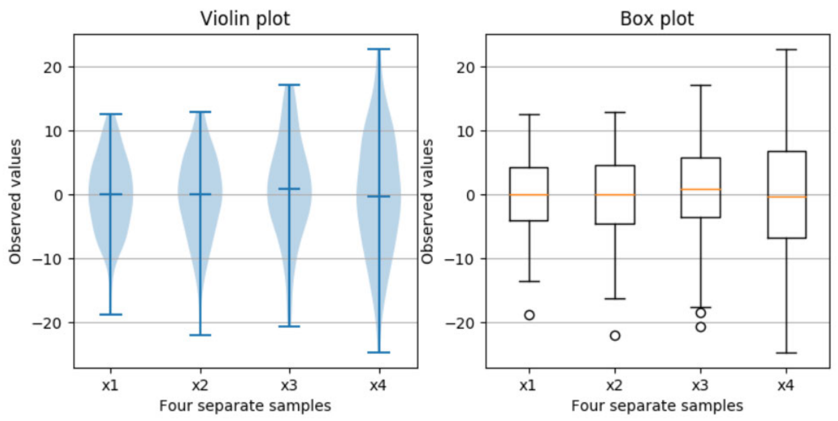
```
sb.boxplot(x='species',y='petal_length',data=iris)
plt.grid()
plt.show()
```

Output:



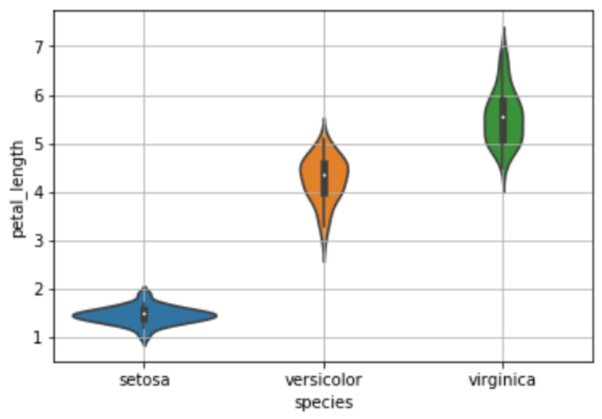
What is a violin plot?

A violin plot is a method of plotting numeric data. It is similar to a box plot, with the addition of a rotated kernel density plot on each side. Violin plots are similar to box plots, except that they also show the probability density of the data at different values, usually smoothed by a kernel density estimator.



```
sb.violinplot(x='species',y='petal_length',data=iris,size=8)
plt.grid()
plt.show()
```

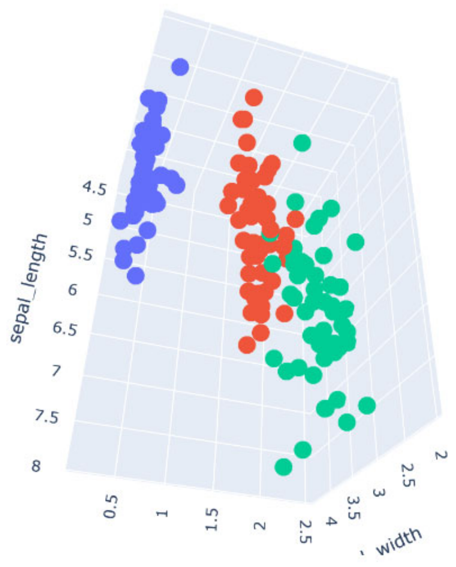
Output:



A 3D plot of the iris flower dataset

```
import plotly.express as px
fig = px.scatter_3d(iris, x='sepal_length', y='sepal_width', z='petal_width', color='species')
fig.show()
```

Output:


sepal_length
width