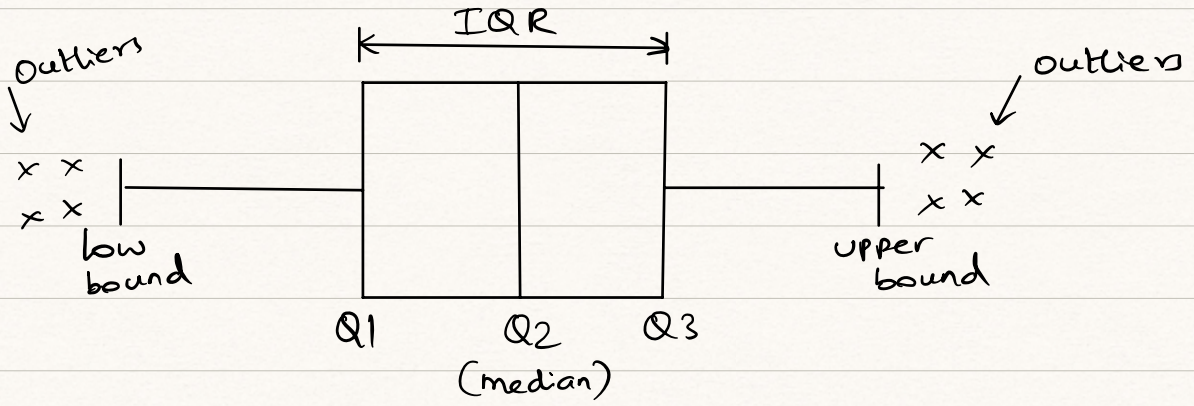
Finding outliers using boxplot

boxplot: A boxplot, or a box-and-whisker plot summarizes a data set virtually using a five point summary.

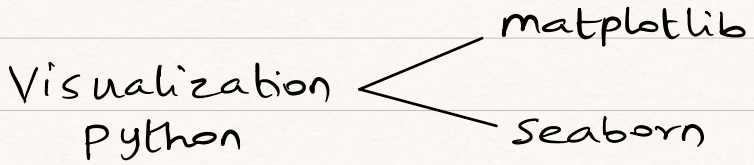
- lower bound (lowest value)
- Q1 : 25th percentile
- Q2 : Median : 50th percentile
- Q3 : 75th percentile
- upper bound (highest value)

* boxplot is a uni-variant plot
* boxplot can only be applied to single numerical column.



* The vertical lines in the box show Q1, Q2 (median), Q3
* The whiskers at the end show low bound, upper bound.
* In the boxplot the width of the box shows IQR.

To find outliers visually



matplotlib : matplotlib is a low level graph plotting library in python that serves as visualization utility.

Install matplotlib

```
pip install matplotlib

import matplotlib
```

most of the matplotlib utilities lies under the pyplot submodule and are usually imported under 'plt' alias.

```
import matplotlib.pyplot as plt
```

* matplotlib is mostly from matlab.
* In matplotlib we have complete control over plot
* It is a difficult way.

Seaborn : Seaborn is a library that uses matplotlib underneath to plot graphs.

Install seaborn

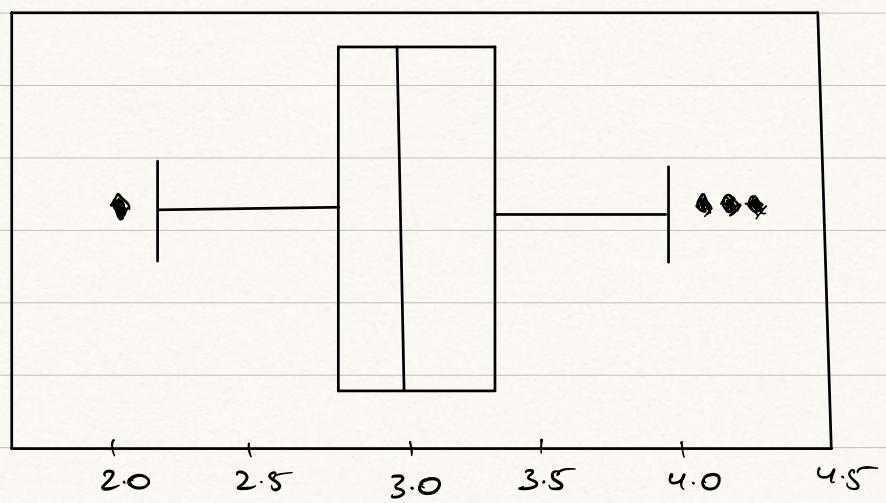
```
pip install seaborn
import seaborn as sns
```

* Seaborn is more customizable than matplotlib
* It is a easier way.

Finding outliers visually we use seaborn boxplot :

```
sns.boxplot(data1["sepalwidthcm"])
```

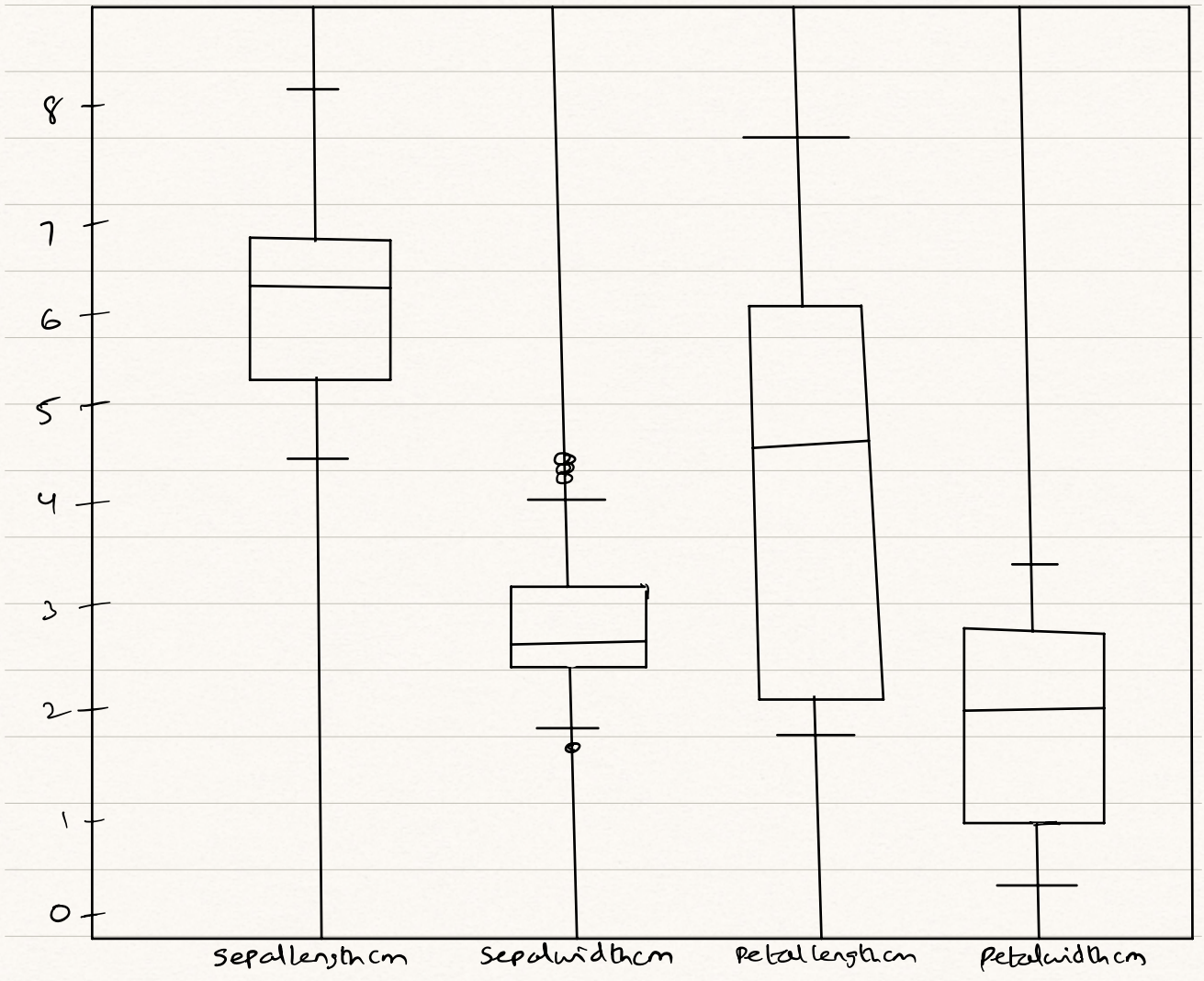
↳ <Axes:subplot:xlabel='sepalwidthcm'>



* Though visually we see one point less than lower inbound and 3 points more than upper bound, visually one point can be duplicate of many points.

* So using boxplot we can check if outliers exist and then using formulae we can find outliers.
* Formula's give the exact outliers.
* Using seaborn boxplot we can plot for each feature variable seperately.
* we can use pandas function called boxplot()

```
dataC.boxplot()
```
↳ <Axes:subplot:>



* This function applies to all numerical values.
* This shows outliers column wise
* Using pandas to boxplot() and find if outliers exist
* Using formulae to find the exact outliers.
* boxplot is a uni-variant numerical plot to find outliers and visualize the spread.
* boxplot we can see spread and std also.

Data Analysis using 2 numerical variables with matplotlib (bi-variant Analysis):

* matplotlib is a library
* we can use pyplot submodule to plot python plots.

```
import matplotlib.pyplot as plt
```

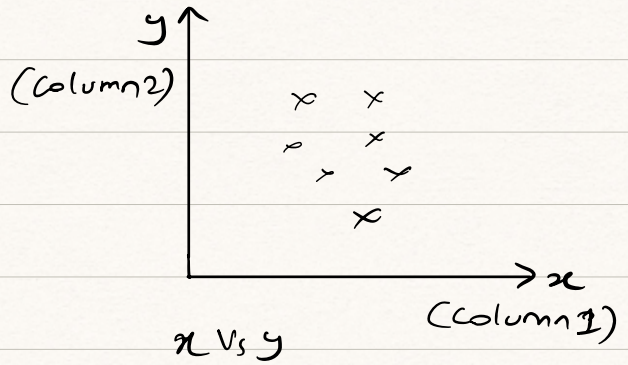
* we can use plot() function to draw a scatter plot or line plot (joins all scatter points)
* There is a scatter() function also to draw a scatter plot.

Scatter plot: A scatter plot is a diagram where each value in the data set is represented by a dot.

The matplotlib module has a method for drawing scatter plots, it needs two arrays of the same length, one for the value of x-axis and one for the value of y-axis.

- we can use plt.scatter(x,y)
- we can use plt.plot(x,y) with additional parameters

* This is a bi-variant plot
* Both the columns have to be numerical columns.

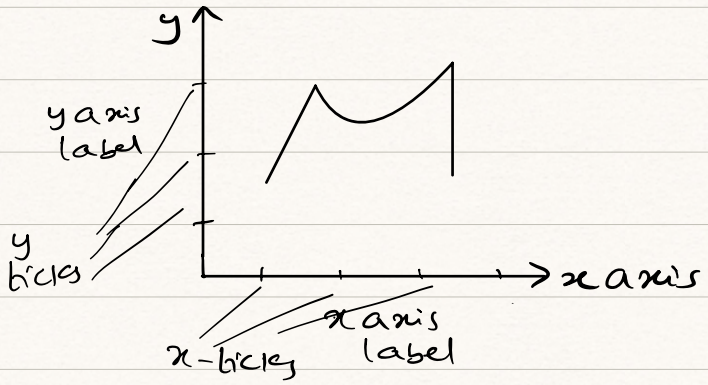


x Vs y

numerical vs numerical

line-plot : This is a plot that is a result of line joining each value in the dataset.

* This is a bi-variant plot
* Both the columns have to be numerical columns.



- we use plt.plot(x,y) from matplotlib to draw the line plot.
- By default plt.plot() draws a line plot.

In matplotlib we have to mention everything.

* To give x-label, y-label and title we use

plt.xlabel(" ")

↳ string

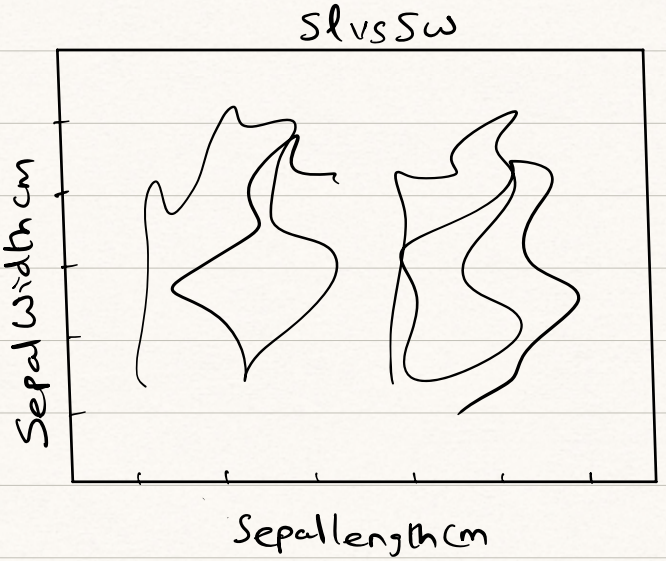
plt.ylabel (" ")
↳ string

plt.title (" ")
↳ string

ex:-

```
plt.plot(datac["Sepallengthcm"], datac["sepalwidthcm"])
plt.xlabel("Sepallengthcm")
plt.ylabel("Sepalwidthcm")
plt.title("sl vs sw")
```

↳ Text(0.5, 1.0, slvssw)



Additional parameters

linestyle: defines the style of the line

- default (solid line = "-")
- if we change to (dotted line = "--")
- if we change to (space = " ")

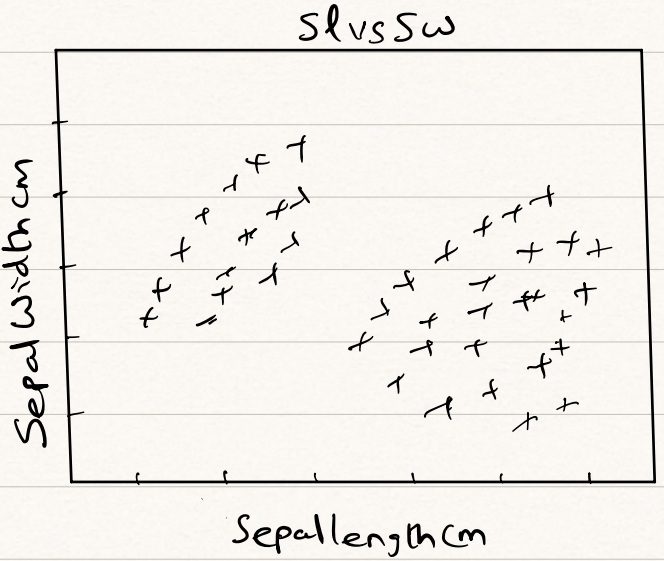
i.e linestyle = " "

we use this to get scatter plot.

* But here the values are not marked so we need to mark them to see scatter plot

modifications to do to get scatter plot :

```
plt.plot(datac["Sepallengthcm"], datac["sepalwidthcm"],
         linestyle=" ", marker="+")
plt.xlabel("Sepallengthcm")
plt.ylabel("Sepalwidthcm")
plt.title("sl vs sw")
```



* Various options for markers

markers - "d" - diamond

"s" - square

"o" - circle

* Various options for color

color – "k" – black

"r" – red

⋮

* markersize = 5.....

* linesize = 7.....

* So far we have learned to get scatter plot using different combinations.

* we are not getting any info from this plot as we don't know what these points mean.

* we can plot scatter plot for various combinations to understand and analyze the data better.

ex:-

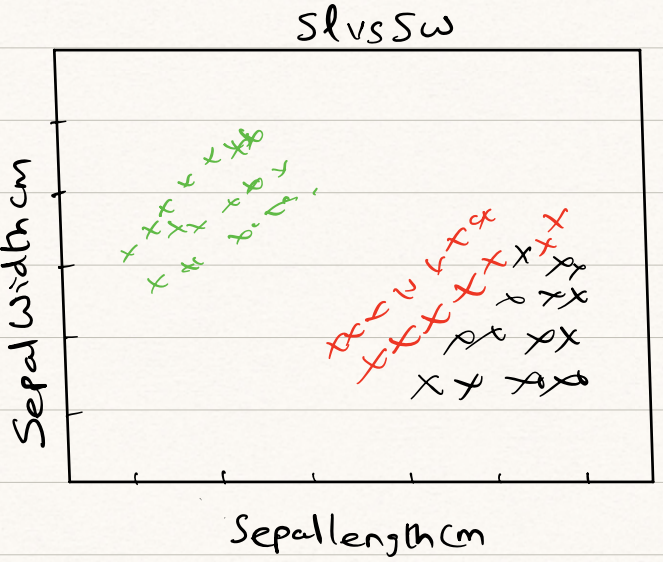
we can split dataset in to 3 data sets depending on species.

```
setosa = datac.loc[datac["species"] == "Iris-setosa"]
versicolor = datac.loc[datac["species"] == "Iris-Versicolor"]
virginica = datac.loc[datac["species"] == "Iris-Virginica"]
```

to add label we need to call one more function called legend() ---- to visualize label to plot.

```
plt.plot(setosa["Sepallengthcm"], setosa["sepalwidthcm"],
         linestyle=" ", marker="o", color="g", markersize=7,
         label="setosa")
```

```
plt.plot(versicolor["Sepal length cm"], versicolor["sepal width cm"],
         linestyle=" ", marker="o", color="r", markersize=7,
         label="Versicolor")
plt.plot(virginica["Sepal length cm"], virginica["sepal width cm"],
         linestyle=" ", marker="o", color="k", markersize=7,
         label="Virginica")
plt.xlabel("Sepal length cm")
plt.ylabel("Sepal width cm")
plt.title("sl vs sw")
plt.legend()
```



Assignment : Do scatter plot using all the combinations and find 2 important variables that can help in seperating the 3 different species.

SL, SW, PL, PW

Combinations : $^{n}C_{r}$ i.e $^{4}C_{2}$

SL SW

SL PL

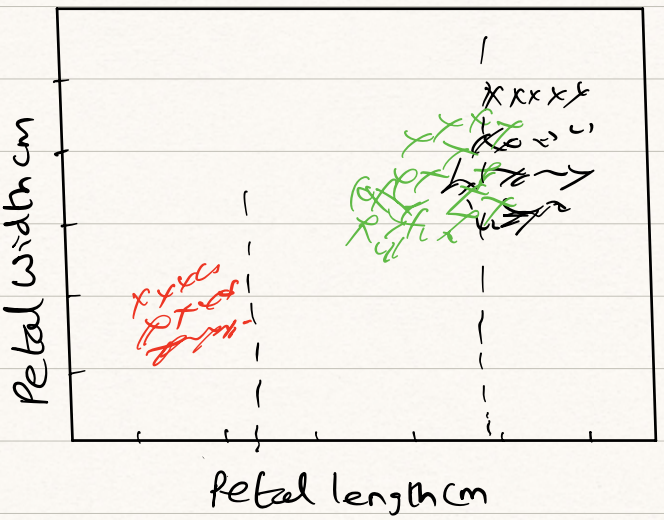
SL PW

SW PL

SW PW

PL PW

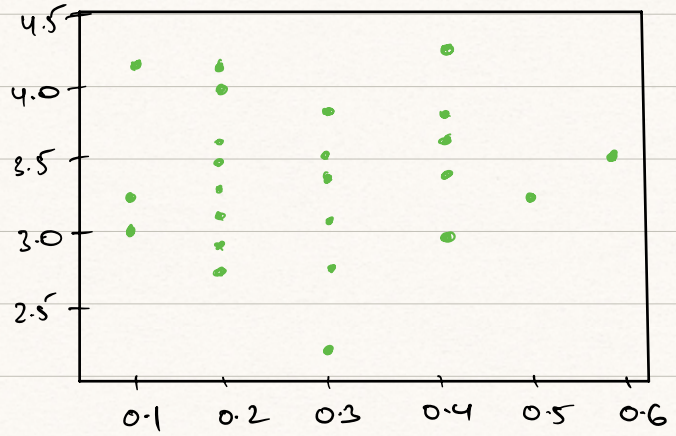
Conclusion : PetalLengthcm, PetalWidthcm are the features that are most important.



we have another matplotlib function called scatter() that plots scatter plot directly.

```
plt.scatter(setosa["PetalWidthcm"], setosa["SepalWidthcm"],
            marker='o', color='g', label='setosa')
```

<matplotlib.collections.PathCollection at 0x9746f60c20>

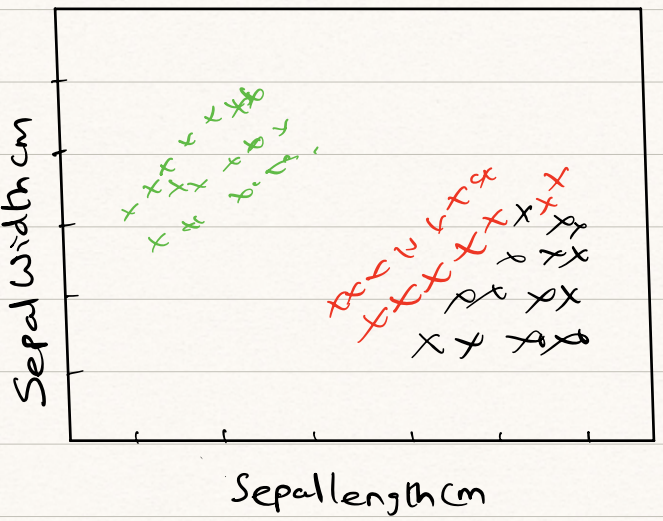


Using seaborn for bi-variant data Analysis:

* with seaborn we can do scatter plot.
* As parameters we mention the data we are going to use.

ex:-

```
sns.scatterplot(data=datac, x="Sepallengthcm",
                y="Sepalwidthcm")
```



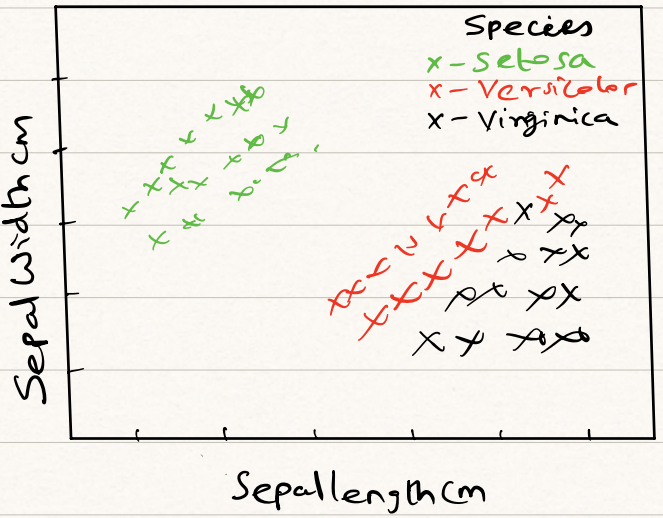
from this we are not able to analyze anything

* we can add class variable as another dimension.
* we use parameter hue for this.
* Here the 3D plot is represented in 2D.
* hue only takes categorical data (class variable)
* we can directly plot two numerical columns (feature variables) for all different class variables.

i.e here we can plot "Sepallengthcm" vs "Sepalwidthcm" for Setosa, Versicolor and Virginica.

sns.scatterplot(data=datac, x="Sepallengthcm", y="Sepalwidthcm", hue="Species")

<Axes: xlabel="Sepallengthcm", ylabel="Sepalwidthcm">



* When we are using Sepallengthcm and Sepalwidthcm we can only seperate Setosa.
* Here we can write if conditions to seperate the species.

i.e 1) if Sepallengthcm is <6 and petalwidthcm<0.6 then it is setosa

2) if petalwidthcm<1.5 and >0.8 then versicolor but we have few mistakes.

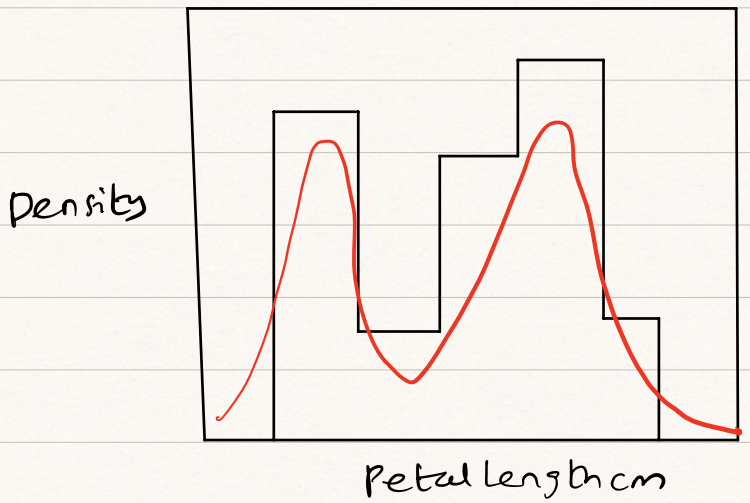
3) if petalwidthcm>1.5 then Virginica but we have few mistakes.

4) so here Sepallengthcm & petalWidthcm are more important than Sepalwidthcm.

* This is called Exploratory Data Analysis (EDA)

Uni-Variant analysis of data

* Since uni-variant we use one variable.
* we will use PDF (Probability Density function)
* In seaborn we have a function called distplot()
  i.e sns.distplot(datac["petallengthcm"])



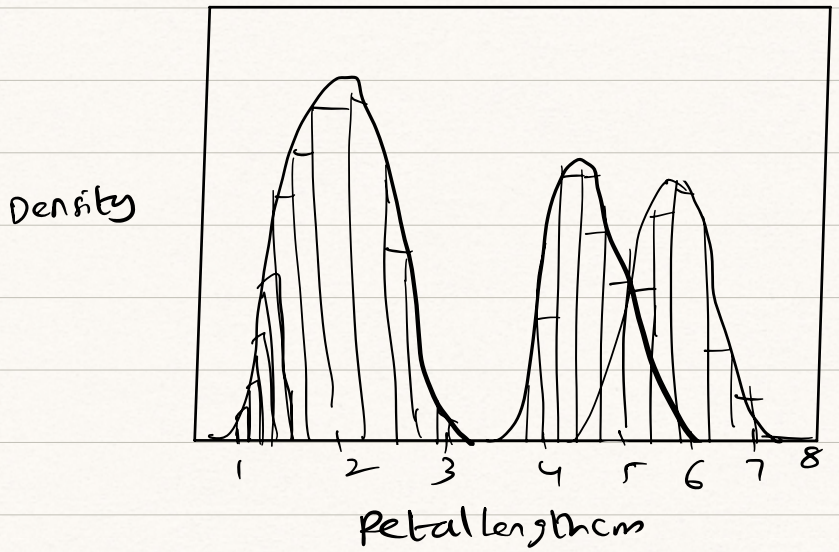
* on Histogram if we smooth we will get PDF.

Similarly we can check PDF for all feature variables but since we do not know which data corresponds against which species we will not be able to classify.

So first we will divide the data and do analysis with PDF.

```
Setosa = datac.loc[datac["species"]=='Iris-setosa']
Versicolor= datac.loc[datac["species"]=='Iris-Versicolor']
Virginica = datac.loc[datac["species"]=='Iris-Virginica']
```

```
sns.distplot(setosa["petallengthcm"])
sns.distplot(Versicolor["petallengthcm"])
sns.distplot(Virginica["petallengthcm"])
```



with just petallengthcm we can seperate from other flowers.

if Petallengthcm > 1 and < 2 then setosa

As we analyze other features we see that there is lot of overlapping.

conclusion:

most important: Petal length cm

next: petalwidthcm

next: Sepallengthcm

worst: Sepalwidthcm

inform the client Sepallengthcm and Sepalwidthcm not useful.

we can use if condition to create a model to

distinguish

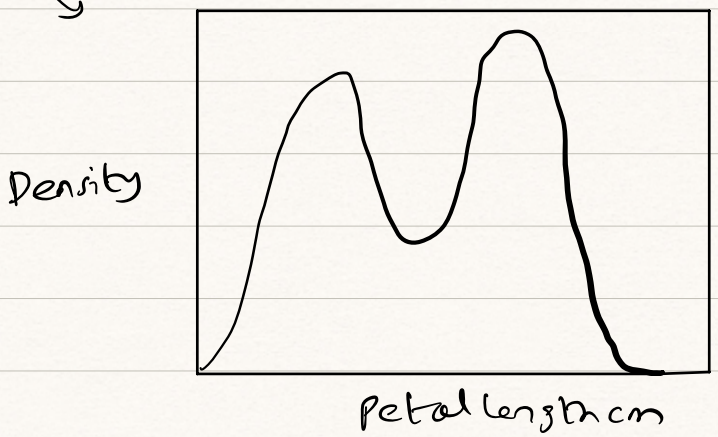
we can also make machine learning model using this.

Note:- distplot() will be removed soon so instead we will use another function displot()

displot() – default is histogram

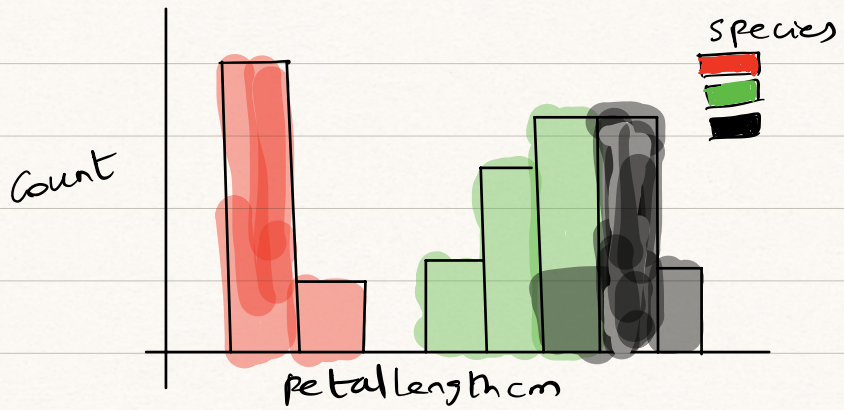
* If we add a parameter called kind = 'kde' we can get PDF from the histogram.

```
sns.displot(datac["petallengthcm"], kind='kde')
```

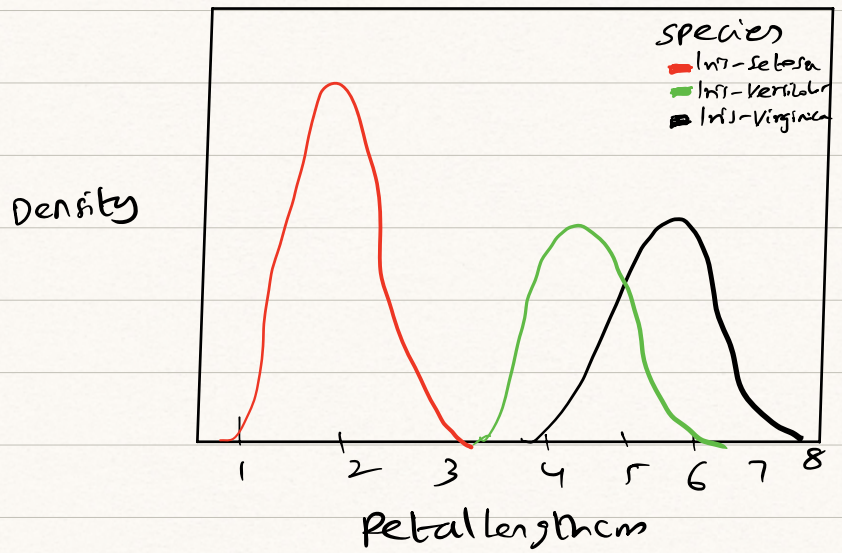


default is histogram

```
sns.displot(data = datac, x="petallengthcm", hue="species")
```



```
sns.displot(data=dataC, x="petallengthcm", hue="species",
            kind="kde")
```



As we analyze each feature variable we see that there is lot of overlapping.

conclusion

most important : petallengthcm

next : petalwidthcm

next : sepallengthcm

worst : sepalwidthcm

To see data is balanced or not visually

Iris data set is balanced data set as

```
data = pd.read_csv(r"C:\....\iris.csv")
data["species"].value_counts()
```

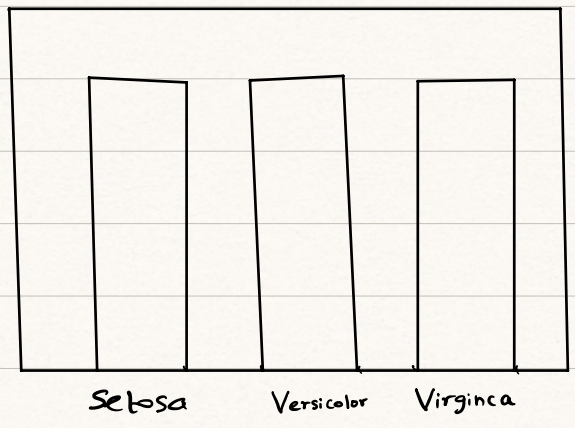
↳

```
Iris-setosa      50
Iris-versicolor  50
Iris-Virginica   50
Name: species, dtype=int64
```

If we want to see visually for **categorical data** if it is balanced or not we can use **bargraph**.

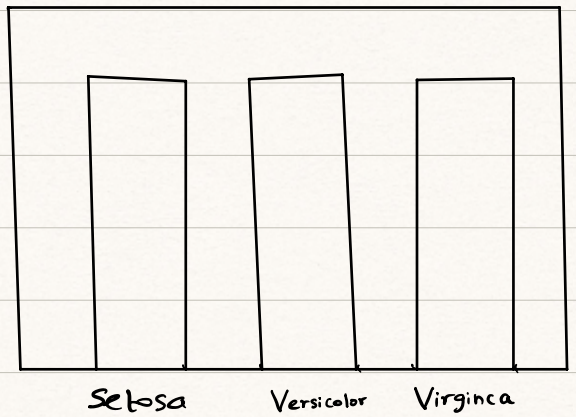
```
#matplotlib command
plt.bar(data["species"].uniques(), [50,50,50])
```

↳



(or)

```
plt.bar(data["species"].uniques(),
        data["species"].value_counts())
```



Bar graph – 1-Dimensional plot

x-axis unique value of categorical data

y-axis Count of Category

balanced

* If bars are of same height (even) then balanced.
* If bars are uneven then unbalanced.

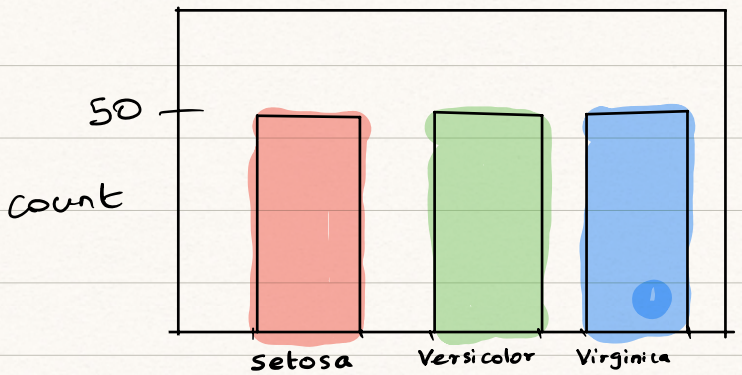
unbalanced

Seaborn command to plot bargraph

This is more beautiful with colors diffrentiating the species.

```
sns.barplot(data["species"].unique(),
            data["species"].value_counts())
```

↳

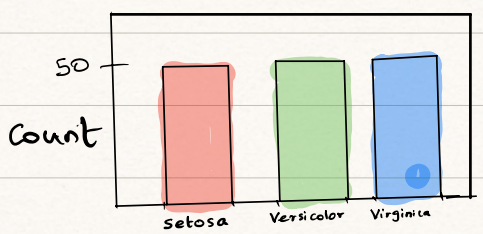


* we can reduce or increase the figure size using shape in tuple format

```
plt.figure(figsize=(,))
```

ex:-

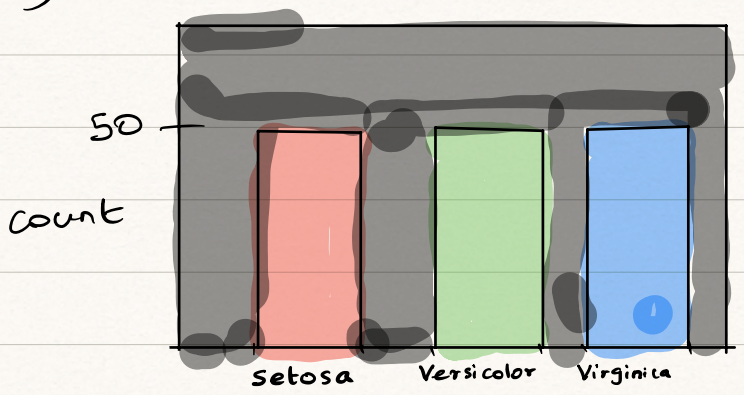
```
plt.figure(figsize=(5,5))
sns.barplot(data["species"].unique(),
            data["species"].value_counts())
```



* facecolor (or paper color) can be changed.

plt.figure(figsize=(5,5), facecolor="k")

↳



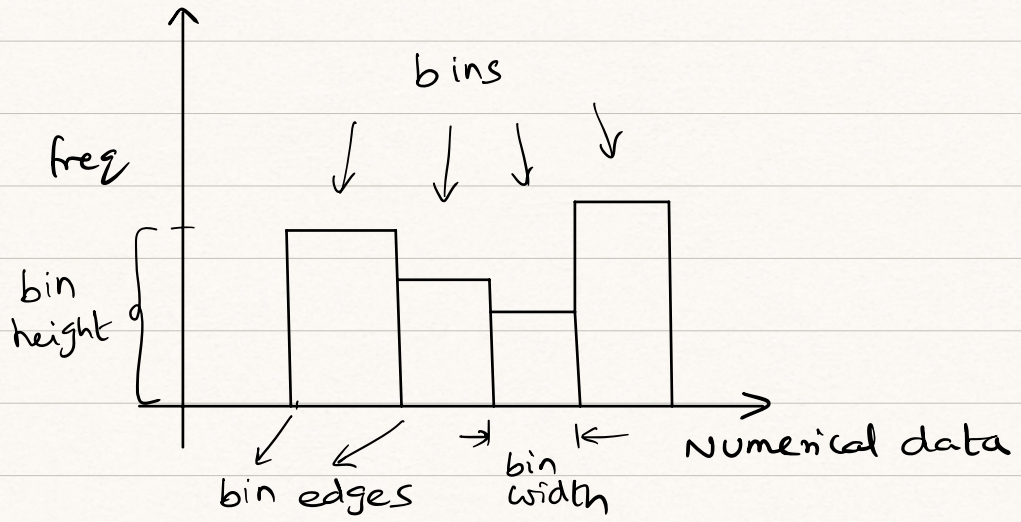
Histogram: A histogram is a graph showing frequency distributions. It is a graph showing the number of observations with in each given interval.

* Histogram is used on numerical data columns.
* Histogram is a uni-variant analysis.

Bargraph	Histogram
↓	↓
Uni-variant analysis on categorical data	Uni-variant analysis on numerical data.

x-axis - numerical data

y-axis - no.of observations with in each interval (frequency of data)



If we are given a dataset

1) sort the values in ascending order
2) How many bins we want (we can decide)

3) find the max value & min value

$$\text{bin width} = \frac{\text{max} - \text{min}}{\text{no. of bins}}$$

ex:- 6, 5, 4, 3, 1, 2, 7, 8, 9

1) 1, 2, 3, 4, 5, 6, 7, 8, 9 (sorted)

2) no. of bins = 4

3) max = 9 min = 1

$$\text{binwidth} = \frac{9-1}{4} = 2$$

bin edges = 1, 1+2, 1+2+2, 1+2+2+2, 1+2+2+2+2

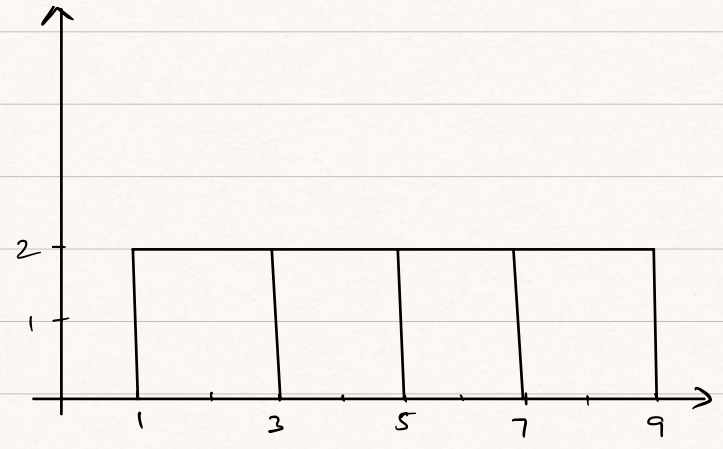
= 1, 3, 5, 7, 9

$1 \leq \text{Values} < 3 \quad = 2$

$3 \leq \text{Values} < 5 \quad = 2$

$5 \leq \text{Values} < 7 \quad = 2$

$7 \leq \text{Values} < 9 \quad = 2$

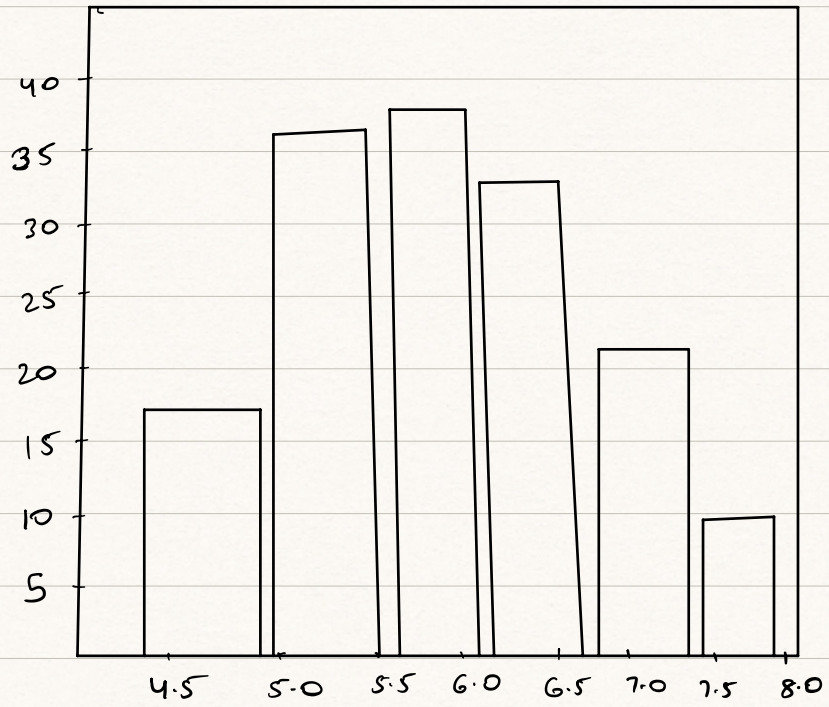


* If we choose less no.of bins then the bin width will be more.
* more no.of bins then bin width will be less.
* when we want to plot PDF over histogram, we make the no.of bins very very high so bin width becomes very less and we use kde(kernel density estimation) to plot PDF.

```
import pandas as pd

data= pd.read_csv(r"C:\....\Iris.csv")

import matplotlib.pyplot as plt

plt.hist(data["sepalLengthcm"], bins = 6, rwidth=0.97)
(array([16., 36., 37., 33., 21., 7.]),
 array([4.3, 4.9, 5.5, 6.1, 6.7, 7.3, 7.9])
 <BarContainer object of 6 artists>)
```

bin height

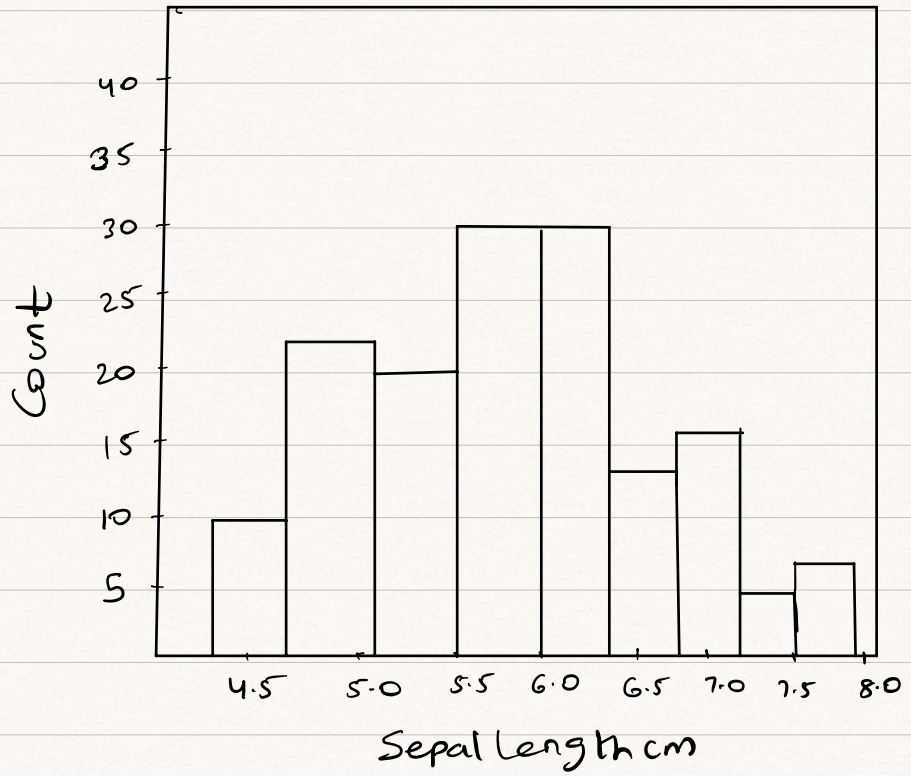
bin edges



* $0 \leq rwidth \leq 1$ gives a space between each bin.
  default is 1 (no width)
* output will be two arrays and histogram plot.
  - bin height
  - bin edges
* we can use seaborn to plot histogram
* default bins = 9

```
import seaborn as sns
sns.histplot(data=data, x="sepal length cm")
```

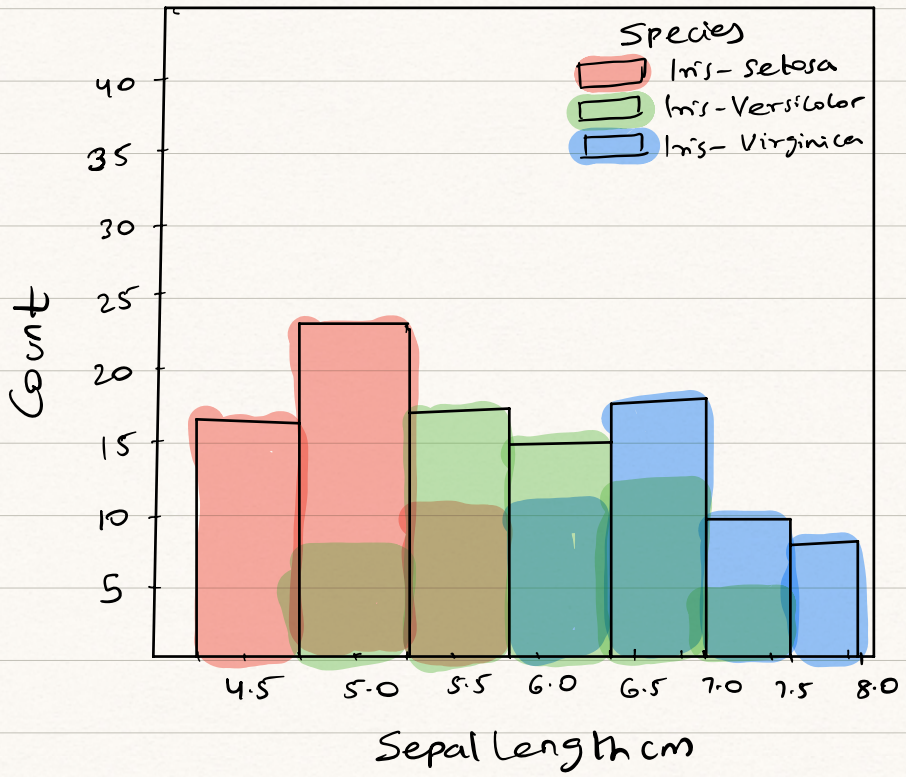
```
<Axes: xlabel='sepal length cm', ylabel='Count'>
```



* In seaborn histogram we can see x-axis and y-axis labels and clear lines showing each bin.
* Here we see overlapping but we do not understand what data corresponds to which Iris-species.
* when we use hue = 'species' then in 1-D histogram we will be adding another dimension and will be doing bi-variant analysis.
* Seaborn gives the plot vs species in single line of code instead of creating sub datasets in matplotlib.

```
sns.histplot(data=data, x="sepal length cm", bins=7,
             hue="species")
```

→ <AxesSubplot: xlabel='sepal length cm', ylabel='count'>

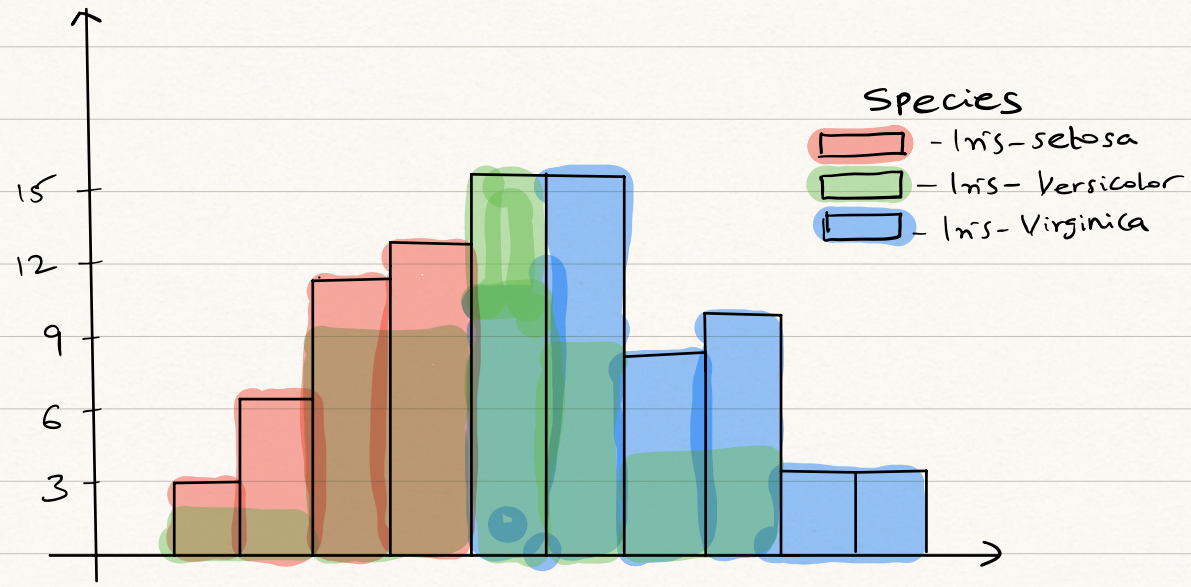


* In matplotlib if we want to plot the similar plot.

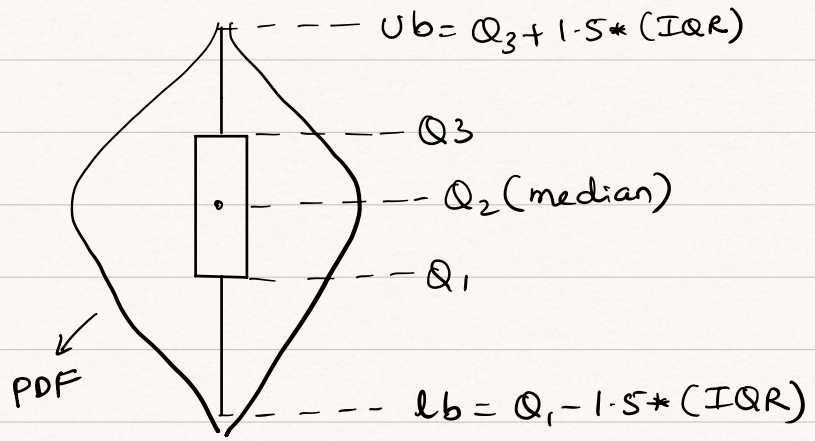
```
setosa = data.loc[data['species'] == 'Iris-setosa']
Versicolor = data.loc[data['species'] == 'Iris-Versicolor']
Virginica = data.loc[data['species'] == 'Iris-Virginia']

plt.hist(setosa["Petalwidthcm"], bin=6, rwidth=0.98,
                                    label='setosa')
plt.hist(Versicolor["Petalwidthcm"], bin=6, rwidth=0.98,
                                    label='versicolor')
plt.hist(Virginca["Petalwidthcm"], bin=6, rwidth=0.98,
                                    label='Virginca')
plt.legend()
```

* we can plot histogram using displot as default plot of displot( ) is histogram

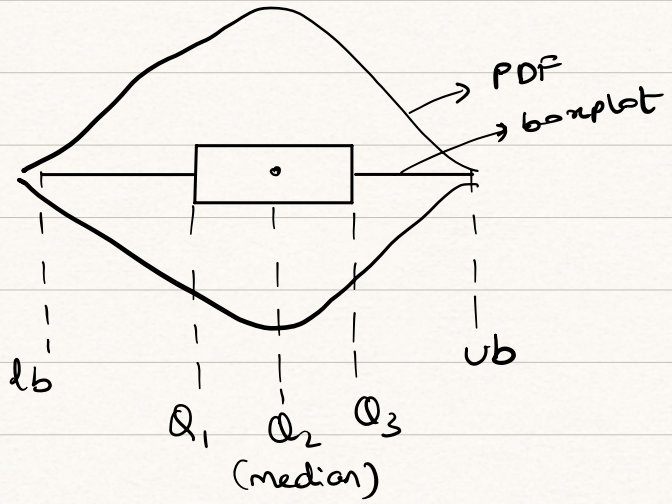
```
plt.figure(figsize=(15,10))
sns.displot(data=data, x="Sepal width cm", bins=10,
                                    hue="species")
```



Violin plot :- Violin plot is a statistical representation of numerical data. It is similar to boxplot, with addition of PDF on each side.

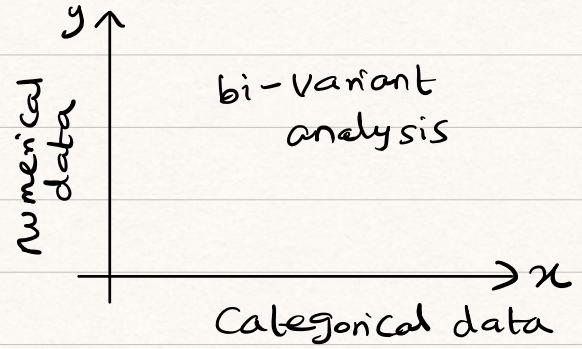


This can be done horizontally also



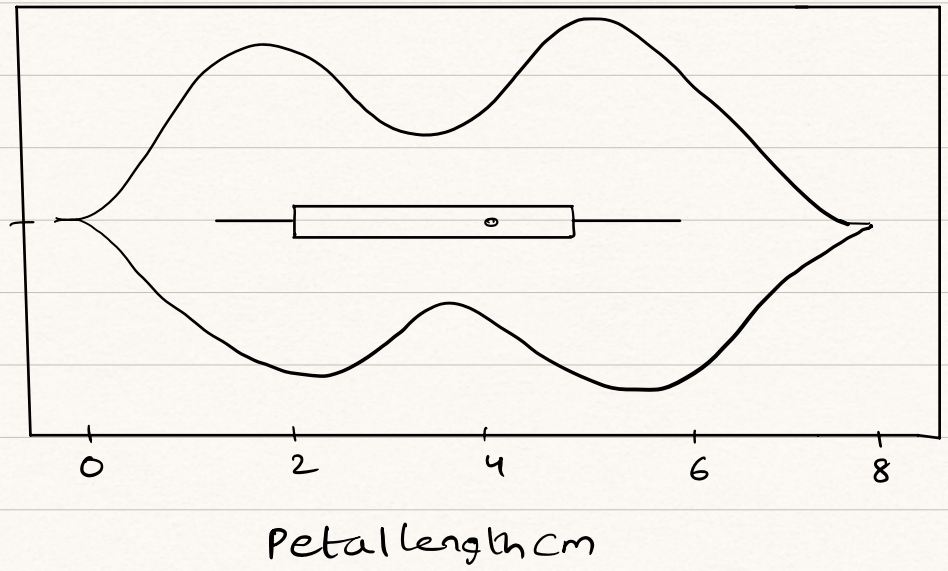
* The PDF is symmetrical on both sides of the box plot.
* This is a uni-variant analysis.
* This uses numerical data.

* with addition of hue = Categorical data we can make it act like bi-variant analysis.
with numerical data vs categorical data



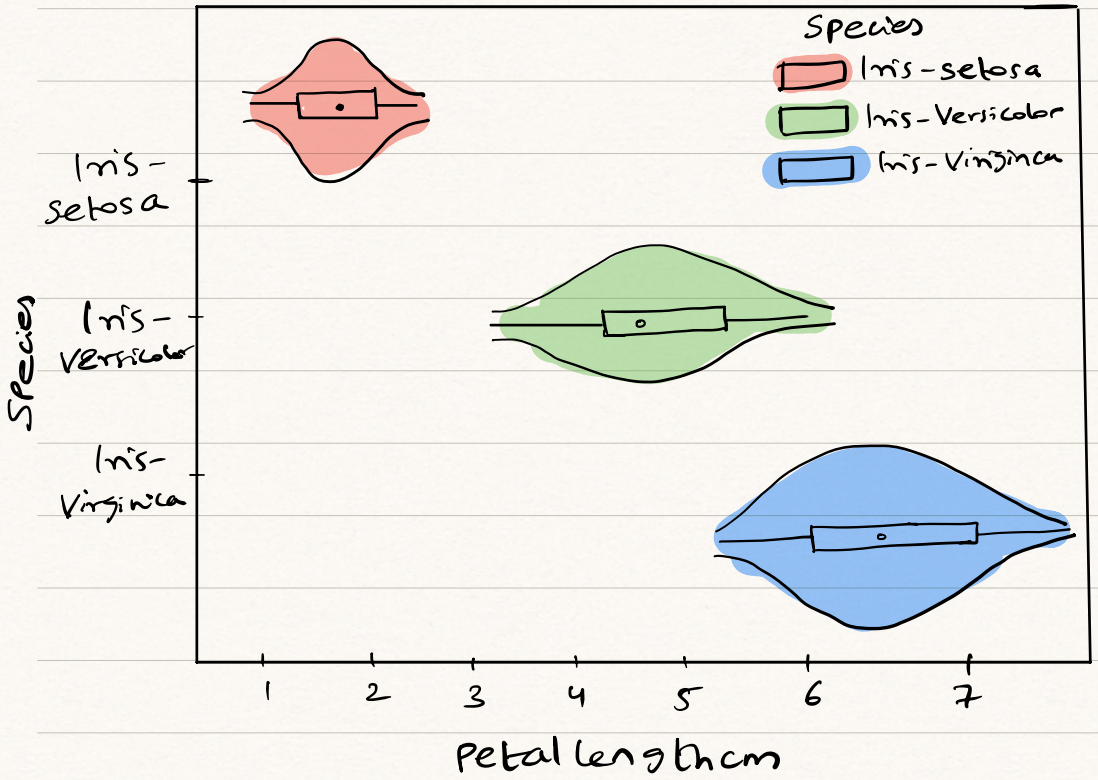

```
sns.violinplot(data=data, x="petallengthcm", size=10)
```

↳ <Axes subplot : xlabel = "petallengthcm">




```
sns.violinplot(data=data, x='petalwidthcm', y='species',
               hue='species', size=10)
```

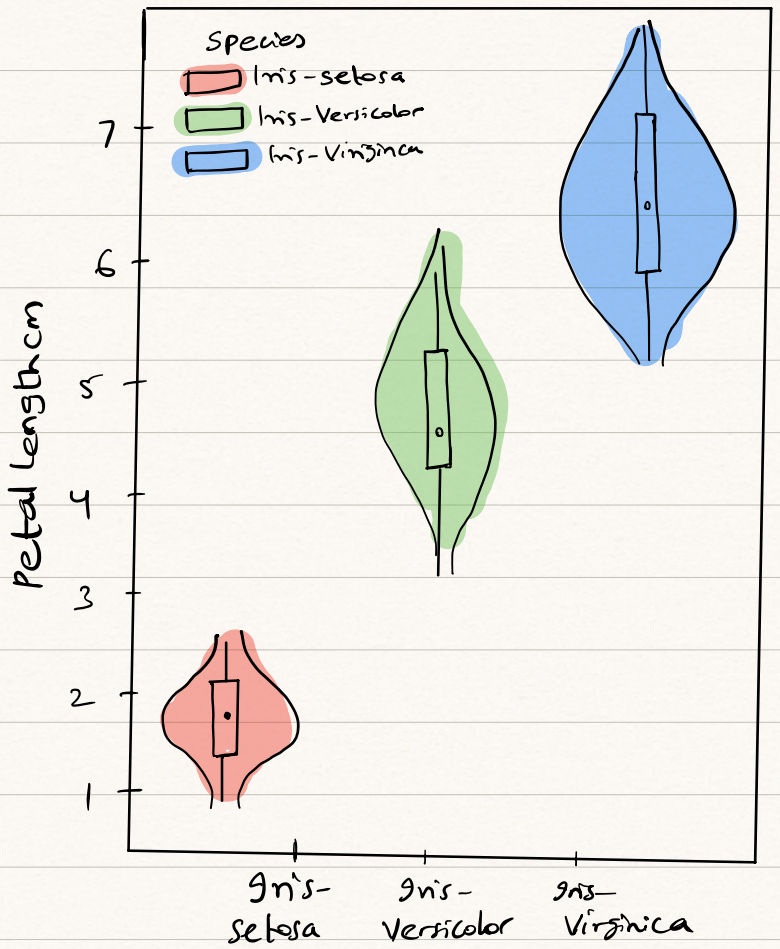
↳ <Axes subplot : xlabel='petallengthcm', ylabel='species'>



we can change the axis

```
sns.violinplot(data=data, x='species', y='petal length cm',
               hue='species', size=10)
```

<Axes subplot:xlabel='species', ylabel='petal length cm'>


Species
Iris-setosa
Iris-Versicolor
Iris-Virginica
Petal Length cm
7
6
5
4
3
2
1
Iris-setosa
Iris-versicolor
Iris-Virginica

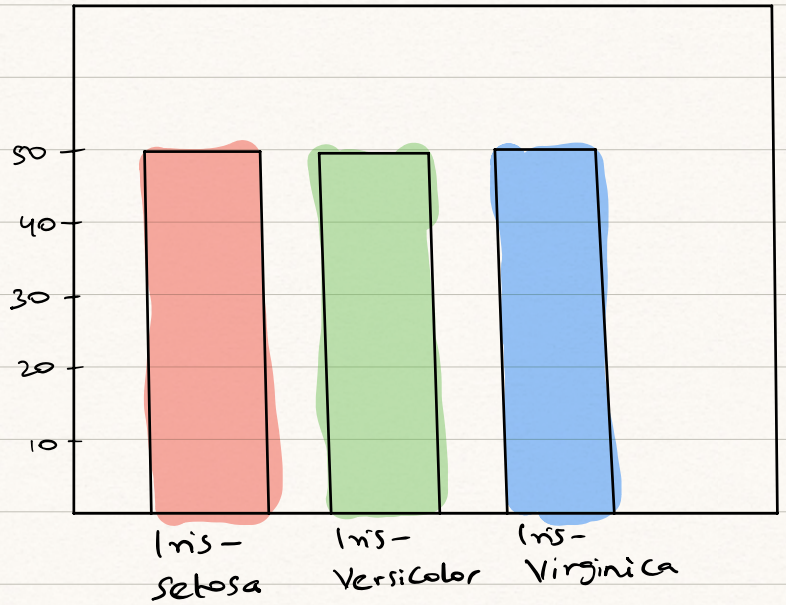
Countplot : This is similar to bar graph.

* Takes Categorical data
* Uni-Variant analysis
* Unlike bar graph we do not need to mention x-axis and y-axis we just need to mention data.

```
sns.countplot(data["species"])
```

↳

```
<Axes Subplot : x label = 'species', y label = 'count'>
```



* If we just give data it will identify the Categorical data & will plot based on unique values.

Pie-chart: A pie-chart is a circular statistical graph, which is divided in to slices to illustrate numerical proportion.

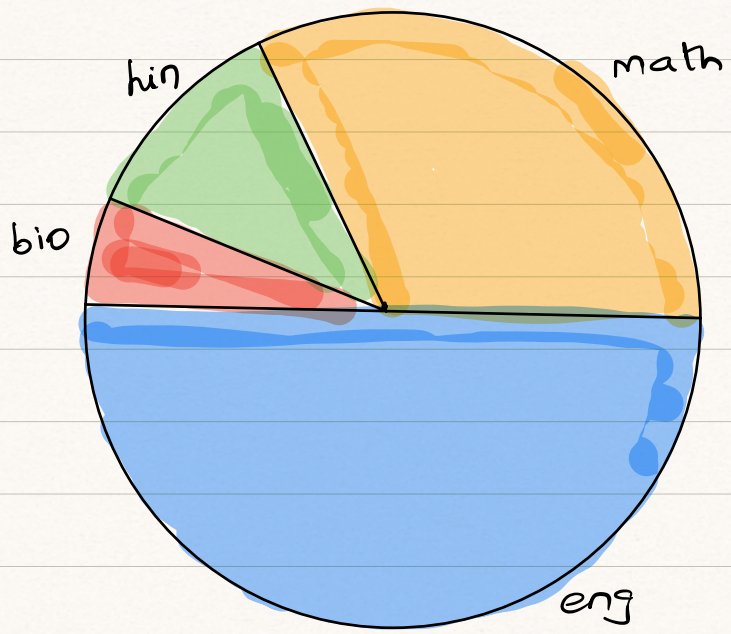
* This is used on discrete numerical data. (i.e Limited values)
* This is a uni-variant analysis.

ex:- if we have marks data

	math	hin	bio	eng
Name	60	20	10	90

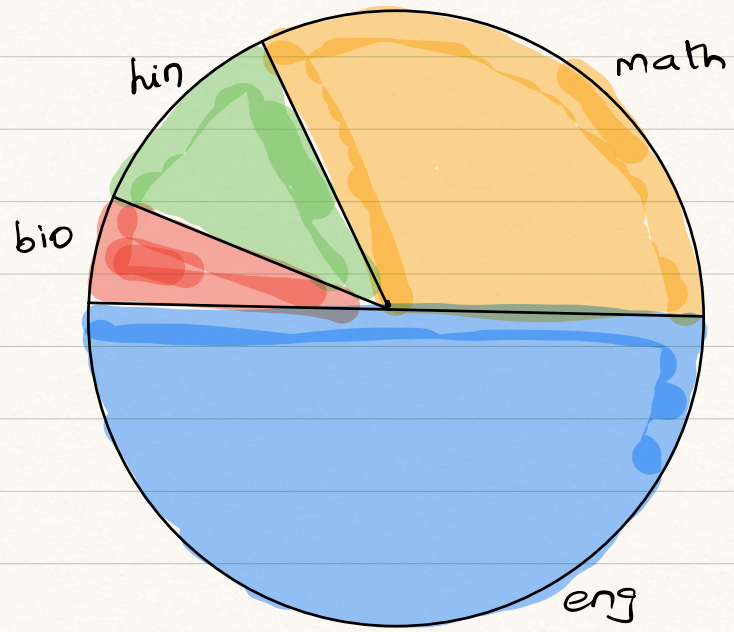
```
plt.pie([60, 20, 10, 90], labels = ['math', 'hin', 'bio', 'eng'])
```

↳ lot of info about axis is printed here



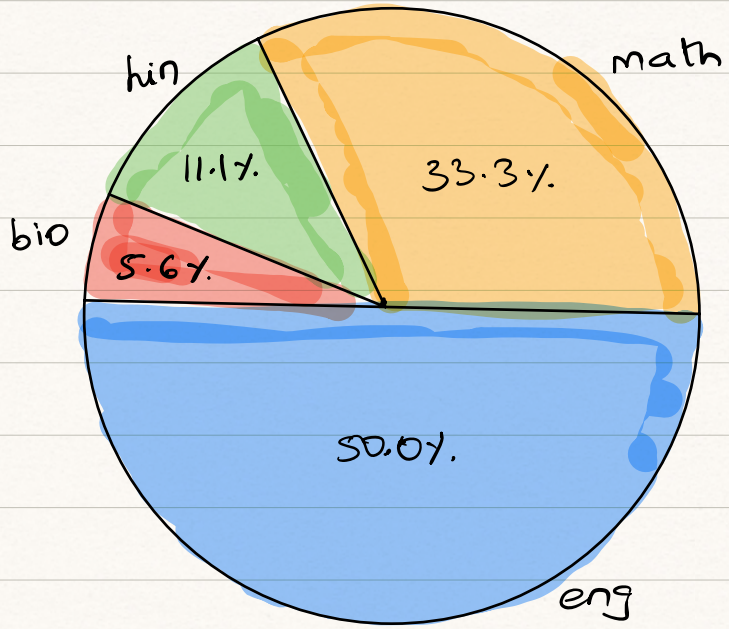
* Here to remove all the info we get after running this command. we use plt.show()

```
plt.pie([60,20,10,90], labels=['math','hin','bio','eng'])
plt.show()
```



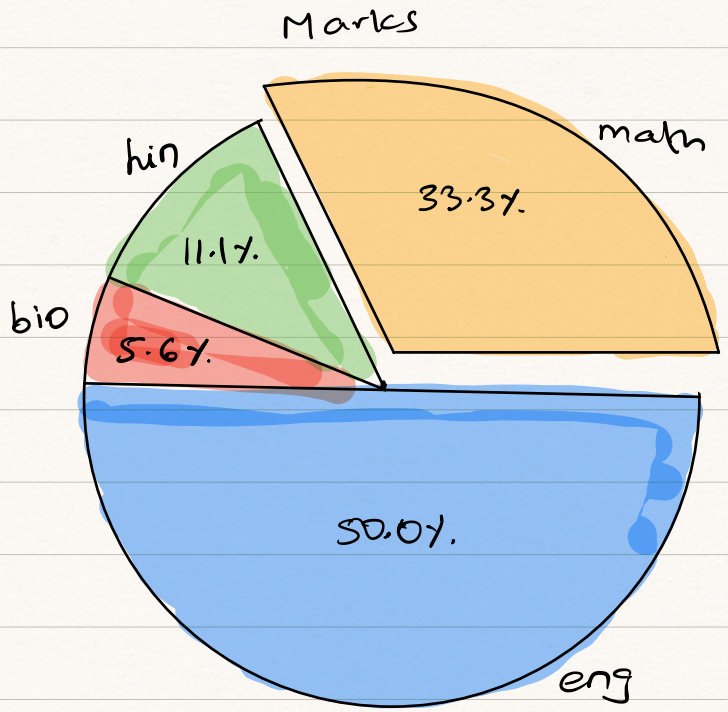
* we can print the percentage on the pie-chart using the parameter autopct (auto Percentage)
* here autopct = "%0.1f%%"
* here 0.1 represents 1 value after decimal so 0.0 is no point after decimal.

```
plt.pie([60,20,10,90], labels=['math','hin','bio','eng'],
        autopct="%0.1f%%")
plt.show()
```



* we can also make the wedges move away from the center or explode using explode parameter.

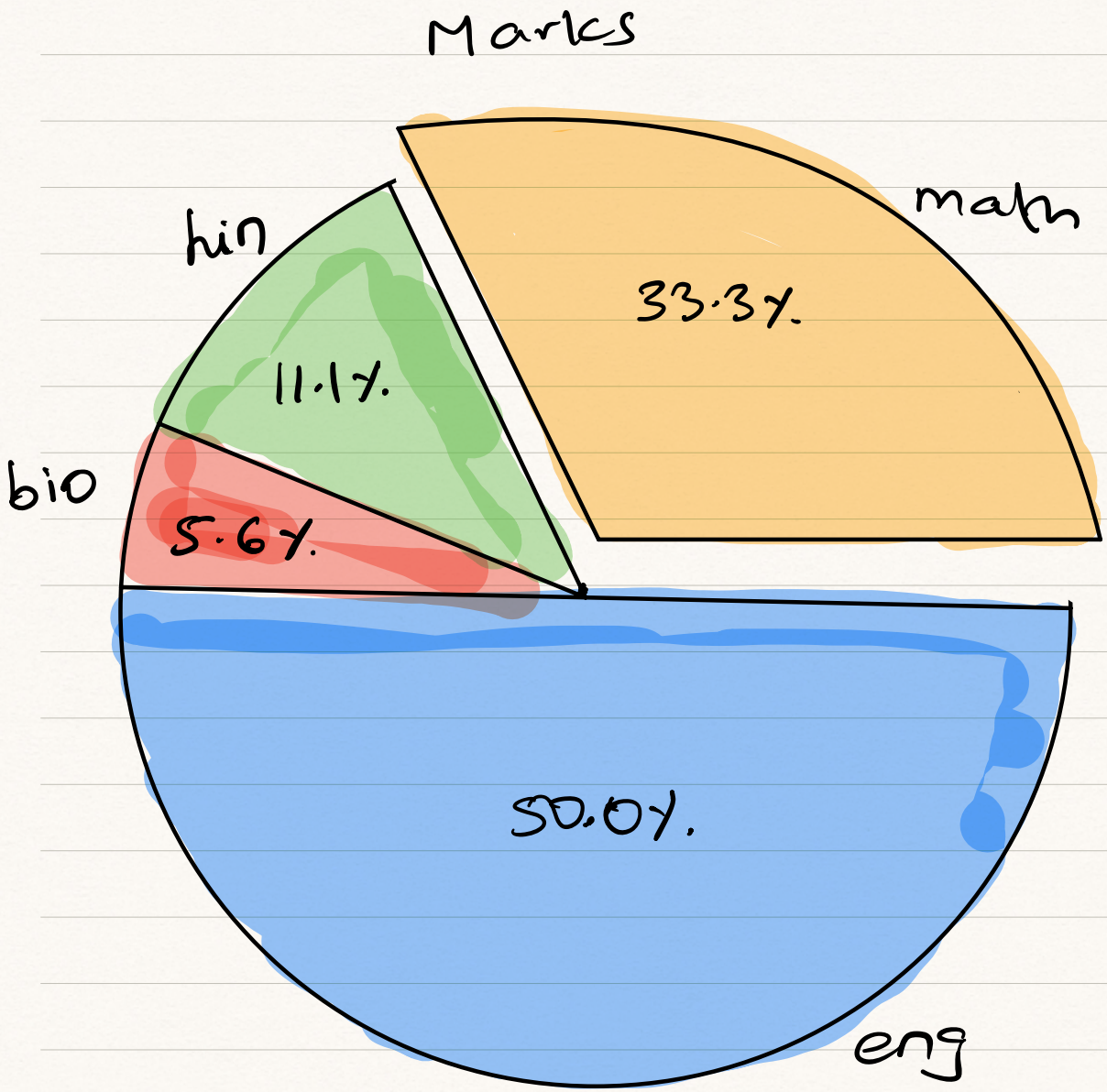
```
plt.pie([60,20,10,90], labels=['math','hin','bio','eng'],
        autopct="%.0.1f%%", explode=[0.1,0,0,0])
plt.title(marks)
plt.show()
```



* we can increase the size using radius parameter

i.e

```
plt.pie([60,20,10,90], labels=['math','hin','bio','eng'],
        autopct="%.0.1f%%", explode=[0.1,0,0,0],
        radius=3)
plt.title(marks)
plt.show()
```



* pie chart cannot be used for continous data.
* It can be used for discrete numerical data or categorical data.

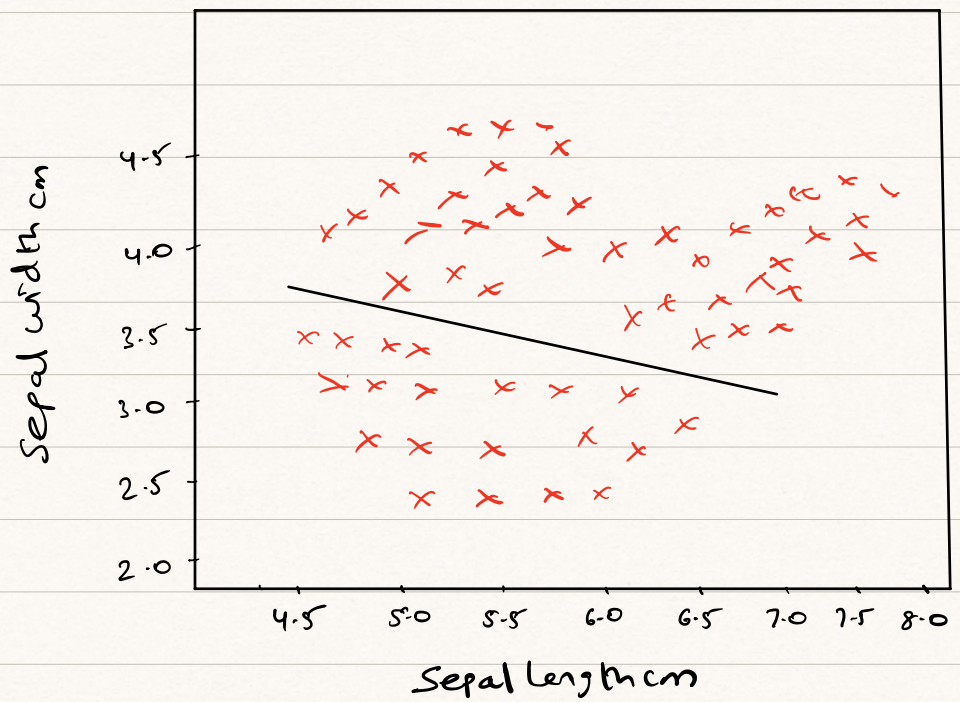
Regression plot : Regression plot as the name suggests creates a regression line between 2 parameters and helps to visualize their linear relationships.

* This is bi-variant analysis.
* This is Numerical data Vs Numerical data.
* This is a scatter plot with regression line on it.

* regplot we will learn more how to calculate the regression line in machine learning.

```
sns.regplot(data=data, x="Sepal Length cm", "Sepal width cm")
```

```
<Axes: xlabel='Sepal Length cm', ylabel='Sepal width cm'>
```



Summarization of plots learned so far

Numerical –
- box plot
- ✱ histogram
- violin plot
- dist plot
- pie-chart (Discrete data)

Categorical –
- ✱ bar graph
- Count plot
- pie-chart

Numerical – ✱ scatter plot
vs line plot
Numerical Regression plot

↓ seaborn gives this functionality

Numerical – box plot (by adding hue)
vs ✱ Violin plot (by adding hue)
Categorical histogram (by adding hue)