Data Analysis - It is a process of inspecting, cleansing, transforming and modeling data with the goal of discovering useful information, informing conclusions, and support decision-making.

In today's business world, data analysis plays a role in making decisions more scientific and helping business operate more effectively.

Data Analysis using Python: we use various Python modules for Data Analysis.

- numpy
- pandas
- matplotlib
- seaborn
- plotly

numpy: helps in creation of high dimensional array which helps in Deep learning and image processing.

pandas: Extension of numpy has lots of tools for Data Analysis and extensively used.

matplotlib: It is a core visualization library that creates static, animated and interactive

Visualizations in python. (Pictorial representation)

Seaborn & plotly - These are visualization tools based on matplotlib.

Data Analyst: Core duty is to analyze the data. whenever we get data the job of data analyst is to analyze the data and get information from the data.

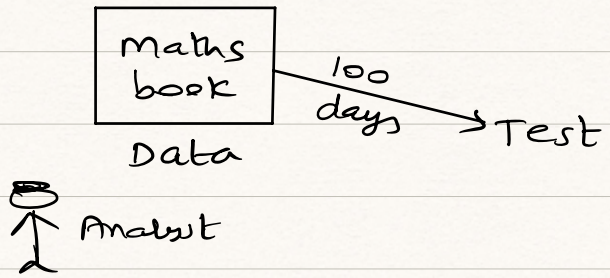
ex:-

1) Analyze the whole forest
2) Identify the place where we can find gold.

we can analyze information from data
- for research
- build machine learning model
- company growth

Data Scientist: 60% of Data scientist job is to do Data Analysis. we have to get correct info on data to create a model

ex:-



If the Analyst did not do correct analysis then in the test he fails.

Main steps for Analysis

1) First and foremost important is knowing the Problem statement. i.e we need to understand the experiment.

2) collecting Data based on the problem statement. we can get data either from a source like (API, csv, txt, ...) or collect it manually.

ex:- Find out how many candidates in innomatics are in Data science and how many are in web development.

- first check if data is available from a source (easier this way)
- collecting data manually (hard)

* we can also see if from source we can do webscraping to collect data.

3) Once data is collected perform analysis on data based on problem statement.

In real world there are mainly 2 diff scenarios.

Problem statement & data are given

Problem statement is not given and blindly asked info on data

* If data is given blindly without problem statement we can analyze in free path.
* Based on the result we can decide problem statement
* If problem statement is given we have to analyze in single path based on problem statement.

Data:

we mainly have 2 types of data.

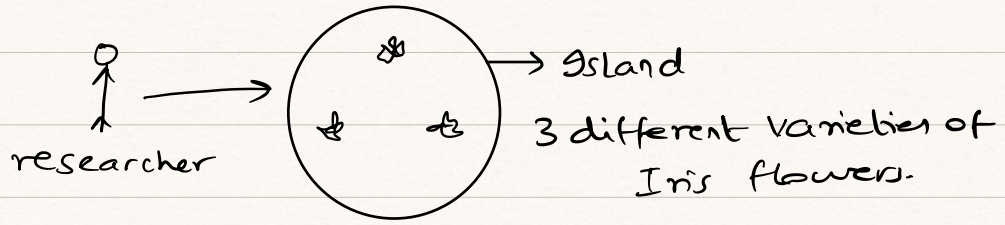
- Categorical data
- numerical data

Categorical data: This refers to a data type that can be sorted and identified based on the names or labels given to them.

Numerical data: This refers to data that is in the form of numbers, and not in any language or descriptive form. Also known as qualitative data as it qualifies data before classifying it.

we will discuss further with the help of iris data-set.

Iris data set (Given to us) : This is similar to "Hello world" we use in other programming languages.



A researcher explored an island and collected data on all the iris flower varieties. Then based on the information from a farmer he labelled them mainly as 3 different types.

Since flowers mainly have sepal & petal. He collected below features for every flower.

- sepal length
- sepal width
- petal length
- petal width

researcher has great difficulty in seperating the flower varieties. He collected the data gave the

data to us as Iris dataset.

3 different types of Iris flowers are

- setosa
- versicolor
- Virginica

Iris dataset :

Id	SL	SW	PL	PW	Species
1					
2					
150					

* Analysis needs to be done to seperate the flowers.

Terminologies :

SL, SW, PL, PW ⟶ features based on problem statement (features of a label)

Species ⟶ class label

* we have to understand the features based on a class label.

* Species is depedent on feature variables.
i.e species is dependent on SL, SW, PL, PW.
So class label is dependent on feature variables.

* Feature variables are independent of each other.
i.e SL, SW, PL, PW are independent of each other.

* Variable is nothing but container storing different variables.
So each column will be variable.

ex:- SL, SW, PL, PW, Species

SL, SW, PL, PW ⟶ feature variables (numerical data)

Species ⟶ Class variable (Categorical data)

SL = 1.1
SL = 1.5
⋮

Species ⟵ Setosa
Species ⟵ Versicolor
Species ⟵ Virginica

Based on problem statement we can seperate

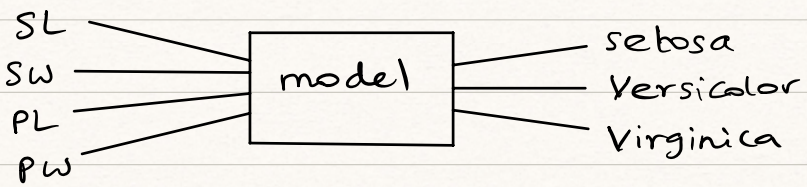
dependent variables (class label)

↳ always deduced from problem statement

Independent variables (feature label)

After gathering all the information:

Data scientist can create a model



ex:- Seperate AP people from Telangana People.

Class label

Ap

Telangana

feature label

hair

feature label

hair → this we can scrap

Intelligence → we know Ap > Telangana

* So we have to Analyze data to see which feature gives more information to seperate the class label and to understand which features are

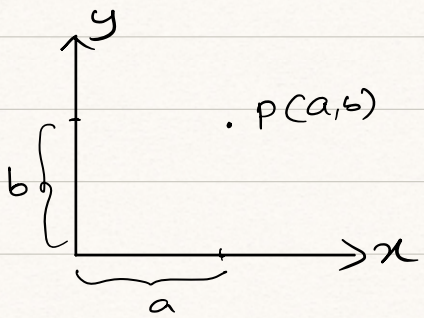
most important.

Datapoint: If we remove the class label (i.e species) the entire row will behave as one data point.

i.e [SL, SW, PL, PW] ⟶ 4 dimensional data point

Each feature specifies a dimension so here datapoint is 4 dimensional.

for 2D representation (we use vectors in math)



P = [a, b]
↑ x component
↳ y component

P = [x1, x2, x3] ⟶ 3D

P = [x1, x2, x3, x4] ⟶ 4D

P = [x1, x2, x3, ... xn] ⟶ n-Dimension

* To use iris.csv we first need to read it.

```
import pandas as pd
pd.read_csv(r"C:\...\iris.csv", sep=',')
(
```

↳

	Id	SL	SW	PL	PW	Species
0	1					
1	2					
:	:					
149	150					

150 rows × 6 columns

we assign this data to a variable

```
data = pd.read_csv(r"C:\....\Iris.csv", sep=',')
```

Copy this DataFrame to a different variable so as not to corrupt original DataFrame

```
datac = data.copy()
```

To understand data we will check its head()

```
datac.head()
```

↳

	Id	SL	SW	PL	PW	Species
0	1					
1	2					
2	3					
3	4					
4	5					

here the problem statement is to seperate Setosa, Versicolor, Virginica.

first we will gather information using info()

```
data.info()
```

↳ <class 'pandas.core.frame.DataFrame'>

RangeIndex: 150 entries, 0 to 149

Data columns (total 6 columns):

#	Column	Non-Null Count	Dtype
0	Id	150 non-null	int64
1	SepalLengthCm	150 non-null	float64
2	SepalWidthCm	150 non-null	float64
3	PetalLengthCm	150 non-null	float64
4	PetalWidthCm	150 non-null	float64
5	Species	150 non-null	Object

dtypes: float64(4), int64(1), Object(1)

memory usage: 7.2+ KB

here we got to know that in this dataset we have around 150 flowers.

Species is the dependent variable.

Id is a redudant column that we can drop.

There are no-null values so perfect data.

4 feature variables, 1 class variable, 1 unnecessary column.

Now

we need to understand the shape

```
datac.shape
↳ (150,6)
```

drop the unnecessary column

```
datac.drop(['Id'],axis=1,inplace=True)
```

Since Id is dropped we have 4 feature variables so 4 dimension data point.

find unique values of species column to understand the categorical data.

```
datac["species"].unique()
↳ array(["Iris-setosa","Iris-versicolor","Iris-
                    Virginica"],dtype=Object)
```

we can count the different category values.
can only be applied on categorical data.

```
datac["species"].value_counts()
↳ Iris-setosa      50
  Iris-Versicolor  50
  Iris-Virginica   50
  Name: species, dtype=int64
```

our dataset is balanced dataset based on the species column.

Balanced data set

Based on the class label we are going to determine data is balanced or not.

balanced dataset - for each class we are getting equal information.

ex:- male = 100 female = 10

find avg height of male and female.

this is imbalanced dataset

↳ heavily imbalanced.

male = 100 female = 90

↳ slightly imbalanced

male = 100 female = 100

↳ Perfectly balanced dataset

describe() will give descriptive statistics of our dataset which is a sample.

This is applied to numerical columns.

```
data.describe()
```

	SepalLength(cm)	Sepalwidth(cm)	Petallength(cm)	petalwidth(cm)
count	150.0...	150.0...	150.0...	150.0...
mean	5.843...	3.054...	3.758...	1.198...
std	0.82...	0.433..	1.764...	0.763...
min	4.30...	2.0...	1.0...	0.1...
25%	5.10...	2.8...	1.6...	0.30...
50%	5.80...	3.0...	4.350...	1.30...
75%	6.40...	3.30...	5.10...	1.80...
max	7.90...	4.40...	6.90...	2.50...

count will calculate all non null values.

mean - average of each column

we observe that avg petalwidth is small compared to other features.

Sepal length mean is more

Sepal width and petal length are similar

$$\text{sample mean} = \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

standard deviation gives us the spread of the points from mean.

$$std = \sqrt{variance}$$

$$Variance = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^2$$

$$std = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^2}$$

* if std is low points are close to mean
* if std is high points are away from mean

so

Sepalwidth is spread very low
Petal length is spread more away from mean value.
Therefore sepal width for 150 flowers will be around 3cm.
min will give the minimum value of that column
max will give maximum value of that column.

percentile (25%., 50%., 75%. we will learn)

to find out if dataset will have null values we can use isnull()
This returns a boolean array so True means null.
sum() will help us see if null value is there because 1 means there is a null value in

that column.

sum will do operation column wise

```
datac.isnull().sum()
```

↳

```
SepalLengthcm    0
Sepalwidthcm     0
petallengthcm    0
Petalwidthcm     0
species          0
dtype: int64
```

attribute to get all the column names: columns

```
datac.columns
```

↳

```
Index(["SepalLengthcm", "Sepalwidthcm", "petallengthcm",
       "Petalwidthcm", "species"], dtype="object")
```

attribute to get all the row names: Index

```
datac.Index
```

↳ RangeIndex(start=0, stop=150, step=1)

Using these if we only want to consider feature variables (numerical values)

```
for y in datac.columns[:4]:
    print(y)
```

↳ Sepal Length cm

Sepal width cm

petal length cm

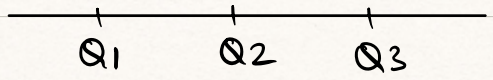
petal width cm

What are quartiles, deciles and percentiles in statistics :

Distribution: A distribution is simply a collection of data, or scores, on a variable. Usually, these scores are arranged in order from smallest to largest and they can be presented graphically.

* Usually, partition values are those values of the variable which divide the distribution in to a certain number of equal parts.
* Please note that the data should be sorted in ascending order.
* Commonly used partition values are quartiles, deciles and percentiles.

Quartiles: Quartiles divide the whole distribution in to four equal parts.

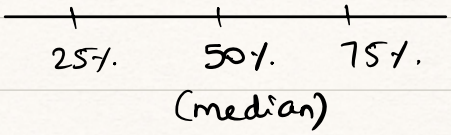


Q1, Q2, Q3
Quartiles

Q1 : 1st Quartile contains $\frac{1}{4}$th data : 25th percentile

Q : 2nd Quartile contains $\frac{1}{2}$ th data : 50 th percentile (median)

Q : 3rd Quartile contains $\frac{3}{4}$th data : 75th percentile



Deciles: Deciles divide the whole distribution in to ten equal parts.

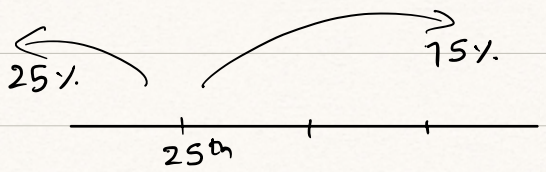
Deciles - 1st Decile, 2nd Decile, ... 9th Decile

Percentiles: percentiles divide the whole distribution in to 100 equal parts.

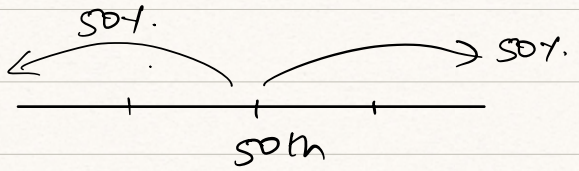
Percentiles - P1, P2, ... P99 are known as 1st Percentile, 2nd Percentile .... 99th Percentile.

* In machine learning we mainly use 25th Percentile, 50th percentile, 75th percentile.

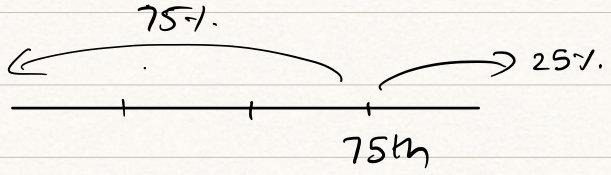
25th percentile (25%) - out of all the points 25% of values will be less than this and 75% are greater than this.



50th percentile (50%.) - out of all the points 50% of values will be less than this and 50% are greater than this.



75th percentile (75 %) - out of all the points 75% of values will be less than this and 25% are greater than this.



* In numpy we have a percentile function that we can use.

```
np.percentile(datac["Sepallengthcm"], [25, 50, 75])
↳ array([5.1, 5.8, 6.4])
```

* default is 100th percentile.

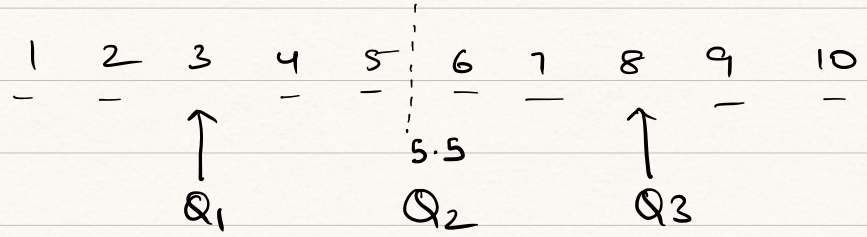
How to find Quartiles :

1) First get the column for which we want to find the Quartiles
2) Sort the column in ascending order.
3) Find the median this will be Q_2 or 50th percentile.
4) divide the dataset in to 2 parts based on median.
5) Now find the median for these sub data sets, there will be Q1, Q2 i.e 25th & 75th percentile.

ex:- 10, 9, 7, 5, 4, 3, 2, 1, 6, 8

sort : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

find median - $\frac{5+6}{2} = 5.5$

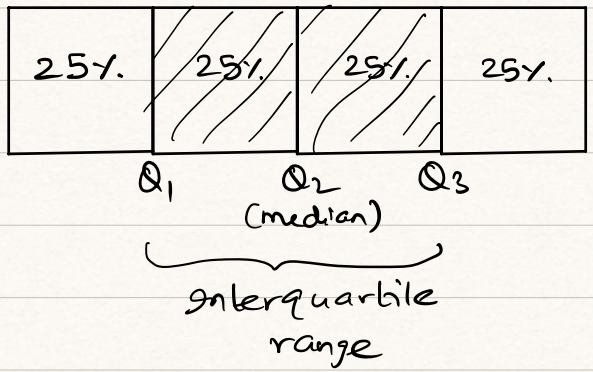


Q1 = 3 (25%)

Q2 = 5.5 (50%)

Q3 = 8 (75%)

Interquartile Range: In descriptive statistics, the interquartile range tells you the spread of the middle half of our distribution.



range: Spread of the whole data set.

IQR: range of the middle half of the data set (spread)

Calculate the IQR

The Interquartile range is found by subtracting the Q1 value from the Q3 value.

Formula	Explanation
IQR = Q3 - Q1	IQR: Interquartile range Q3: 3rd quartile or 75th percentile Q1: 1st quartile or 25th percentile

So in IQR we will have 50% of total points.

ex:-

```
x = np.percentile(data[["Sepal Length cm"]], [25, 50, 75])
q1 = x[0]
q2 = x[1]
q3 = x[2]
```

$IQR = q_3 - q_1$

IQR

↳ 1.30....

Outliners: In statistics, an outlier is a data point that differs significantly from other observations. An outlier may be due to variability in the measurement or it may indicate experimental error; the latter sometimes excluded from the dataset. An outlier can cause serious problems in statistical analysis.

* An outlier is data that has a value too high or too low with respect to other data we are analyzing. Of course in a dataset we won't find a unique outlier. There are several outliers, this is why we often exclude them from the dataset; otherwise, the outliers can cause statistical problems in our analysis.

ex:-

Calculate the salary of all the Innomatics Employees.

most of them (≃100) are between 1L-3L/month

house maid: 10k/month

CEO: 1cr/month

here house maid and CEO are outliers.

outliers can impact the system badly sometimes.

- not all are bad
- not all are good

* Based on the problem statement we need to find if dataset has outliers.

How to find outliers:

we have many methods to do it few are:

1) using IQR
2) using percentile
3) using boxplot
4) Z-score method
5) Isolation Forest algorithm

we will learn using IQR, Percentile and boxplot:

Different type of Analysis:

- **Uni-Variant Analysis**: when we are analyzing single variable at a time (ex:- SepalLengthcm)
- **bi-Variant Analysis**: when we are analyzing two variables at a time. (ex:- SL vs SW 2D-axis)
- **multi-Variant Analysis**: when we are analyzing more than 2 variables at a time. (>2D axis)

How to deal with outliers

- trim it
- replace with a different values depending on problem statement.

Every distribution can be organized using these five numbers:

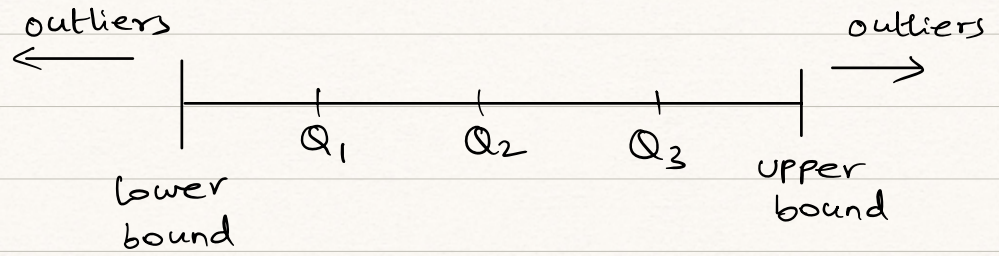
- lower bound (lowest value)
- Q1 : 25th percentile
- Q2 : Median : 50th percentile
- Q3 : 75th percentile
- Upper bound (highest value)

Finding outlier with IQR (uni-variant method)

Here using IQR we will define the lower bound and the upper bound.

lower bound = Q1 − 1.5 * IQR

upper bound = Q3 + 1.5 * IQR



then outlier any value < lower bound

or

> upper bound

ex:- iqr = 1.30...

lb = q1 − (1.5 * iqr)

ub = q3 + (1.5 * iqr)

lb # lower bound

↘ 3.149...

ub

↘ 8.350 # upper bound.

datac.loc[datac["sepal length cm"] > ub]

↘

Empty

SepalLength cm	Sepal width cm	Petal length cm	Petal width cm	Species

datac.loc[datac["sepal length cm"] < lb]

↘

Empty

SepalLength cm	Sepal width cm	Petal length cm	Petal width cm	Species

Similarly we can do for other variables and for this example we find that "sepal width cm" has outliers.

Finding outliers using percentile

* Based on the domain knowledge and the problem statement and the data we can determine the lower bound and the upper bound.

ex:- lower bound = 1%

upper bound = 99.8%.

ex:-

```
x = np.percentile(datac["sepalwidthcm"], [1, 99.8])
lb = x[0]
ub = x[1]
datac.loc[datac["sepallengthcm"] < lb]
```

	SepalLengthcm	Sepalwidthcm	Petallengthcm	petalwidthcm	species
60	5.0	2.0	3.5	1.0	Iris-Versicolor

```
datac.loc[datac["sepallengthcm"]  ]
```

	SepalLengthcm	Sepalwidthcm	Petallengthcm	petalwidthcm	species
15	5.7	4.4	1.5	0.4	Iris-setosa

* Instead of finding these values for all the feature variables and finding the outliers, we can use for loop.

```
ex:- for y in datac.columns[:4]:
        x = np.percentile(data[y], [25, 50, 75])
        iqr = x[2] - x[0]
```

```
lb = x[0] - (1.5*iqr)
ub = x[3] - (1.5*iqr)
print(y)
print(datac.loc[data[y]<lb])
print(datac.loc[data[y]>ub])
print("----------")
```

(Using IQR Prints outliers)

this will output all the outliers in the individual columns.

```
for y in data.columns[:4]:
    x = np.percentile(data[y], [1, 99.8])
    print(y)
    print(len(datac.loc[datac[y] < x[0]])+
      len(datac.loc[data[y] > x[1]]))
    print("--------")
```

```
SepalLengthcm
2
------
Sepalwidthcm
2
------
Petallengthcm
3
--------
Petalwidthcm
0
--------
```

Just prints the count of outliers using percentile method.

* After plotting the species against all the combination of feature variables we come to the conclusion that:

  petallengthcm and petalwidthcm are best to classify species.

* with bi-variant analysis we have concluded that petallengthcm and petalwidthcm are most important.
* If we want to understand single feature that is most important and least important we have to perform uni-variant analysis.
* For this we will learn distribution.

Exploratory Data Analysis: EDA is an approach to analyze data using statistics and visual techniques.

Distribution (will learn further in statistics)

The common distribution functions are

- PDF (Probability distribution function)
- PMF (Probability mass function)
- CDF (Cummulative density function)

* we will be given a Numerical data.

* Question is how the data is distributed?
  - we have different concepts like
    - normal distribution (Gaussian or Probability bell curve)
    - Uniform distribution

* Numerical data is mainly divided to 2 types.
  - discrete data (finite)
  - Continous data (non-finite)

Discrete data :

* This is a data that we can count.

ex:- $x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$

here x is discrete as x cannot be anything other than list given.

i.e $x \times 1.1$

or $x \times 1.2$ ....

* random variable takes value from finite set.
* For discrete data we use PMF (Probability mass function) as mass is an exact answer so PMF gives exact probability.

ex:-

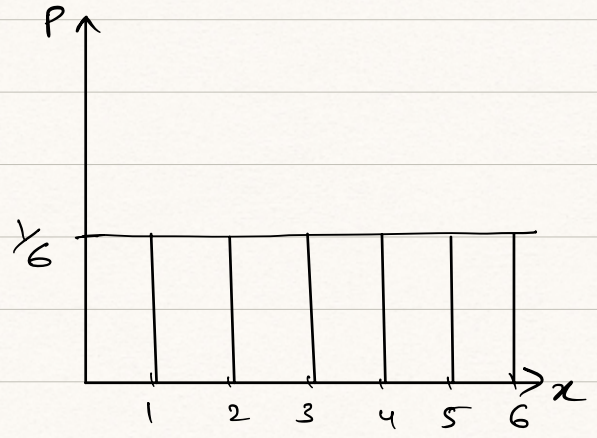
roll a dice then result in $\{1, 2, 3, 4, 5, 6\}$

<u>Experiment</u>: rolling a dice and finding distribution for this experiment (PMF)

$P(x=1) = \frac{1}{6}$ — probability that $x=1$

$\vdots$

$P(x=6) = \frac{1}{6}$ → probability mass values



So from this if $P(x=4) = \frac{1}{6}$

Continous data:

* This is a data that we cannot count.

ex:- $133.2 > x > 153.4$

(start) (end)

even though we have start end the x can take any number of values in this range

i.e $x = 153.20001\ldots$

$x = 133.20001\ldots$

* The data is non-finite as there is start and end but it is impossible to count.
* random variable that takes value from non-finite set then continous variable.
* For continous data we use PDF (Probability Density function) as density is mass per volume and does not give exact answer instead gives an assumption based on probability.

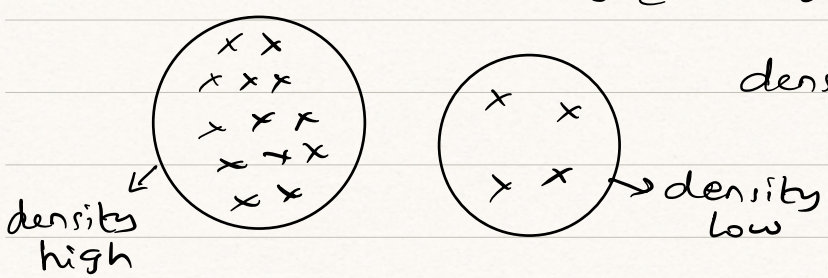
Continous Experiment : pick a height in given range

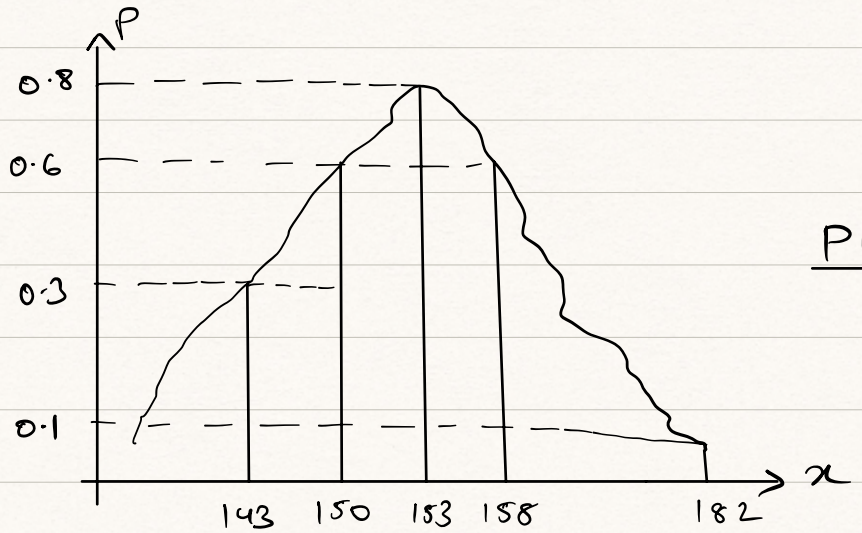
$182 > x > 153$

$P(x = 162) = ?$ (don't know exactly)

we will assume

density = mass/volume





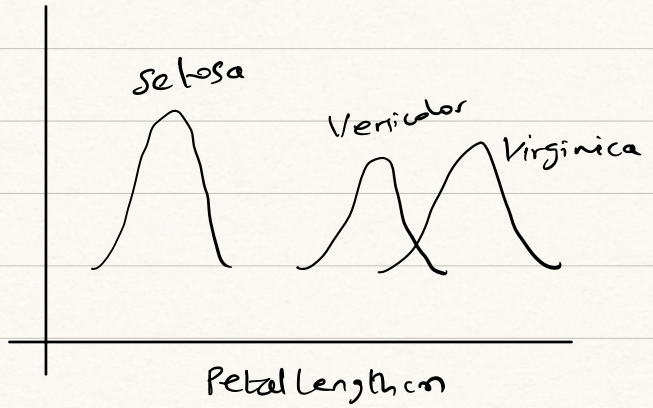
PDF

* 30% of people height is around 143.
* 60% of People height between 150 & 158
* 80% of People height around 153.
* 10% of People height around 182.
* When we smooth the histogram we get PDF.
* PDF is mainly used for uni-variant analysis.
* PDF takes Numerical data
  - Continous
  - Discrete

ex:-

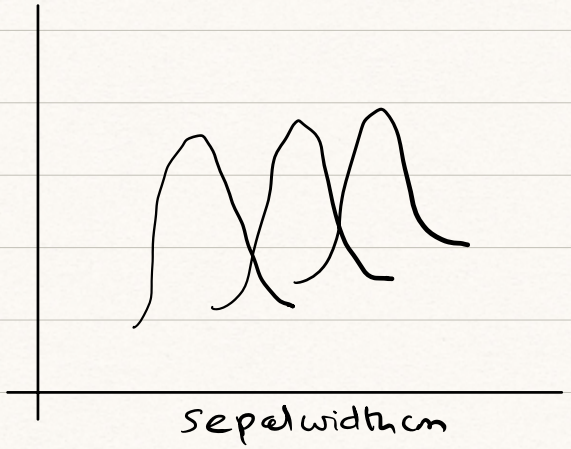
We can apply PDF on petallengthcm we will get 3PDF.

- Setosa
- Versicolor
- Virginica



* If PDF's are far from each other than that feature variable is best for distinction.

If PDF's are plotted with SepalWidthcm.



* overlapping means this feature cannot be used to seperate the species.

* So using PDF we can find which feature is most important and which is learnt important based on the problem statement.

How to deal with outliers:

* we first find outliers using IQR or percentile.
* After we find outliers we have 2 methods to deal with them
  1) trim it (deleting) – only can be done when problem statement mentions removing them.
     - generally if in a column outliers are less than 5%. we can try to remove them (just one way to think).
  2) cap it (filling) – we use mean or median to fill them. This is also not completely right.

* will learn more in machine learning to see if it is right to fill it with mean or median.

How to deal with missing values (NaN), unnecessary values

* In real world we will not get perfect dataset like iris.
* Instead we will get data with lot of missing values or data that is irrelevant like string in age & so on.

```
import pandas as pd
pd.read_csv(r"C:\....\new_csv1.csv")
```

↳

	Name	Age	Gender	Height	weight
0	P1	23.0	m	154.0	43.0
1	P2	21.0	f	123.0	34.0
2	P3	NaN	NaN	NaN	NaN
3	P4	24.0	m	145.0	12.0
4	P5	NaN	m	135.0	NaN
5	P6	12.0	NaN	124.0	45.0

```
data = pd.read_csv(r"C:\....\new_csv1.csv")
datac = data.copy()
datac["Gender"].apply(type)
```

↳

```
0    <class 'str'>
1    <class 'str'>
2    <class 'float'>
```

```
3    <class 'str'>
4    <class 'str'>
5    <class 'float'>
Name: Gender, dtype= object
```

To find null values columnwise

```
datac.isnull().sum()
```

↳

```
Name      0
Age       2
Gender    2
Height    1
weight    2
dtype: int64
```

* If the column is having less than 5%. null values we will try to fill the null values.
* If null values are more than 40-50%. of null values we will just drop the column as this data is no longer useful.
* We will learn ways to deal better later on in machine learning.

To find average of null values

```
datac.isnull().mean()
```

↳

```
Name      0.000...
Age       0.333....
```

Gender	0.333...
Height	0.166...
weight	0.333...
dtype: float64	

To find percentage of null values

```
datac.isnull().mean()*100
```

↳

Name	0.00...
Age	33.33...
Gender	33.33...
Height	16.667...
weight	33.33...
dtype: float64	

* Because this is small data so even though percentage of null values is >5%. we will try and fill them.

How to fill null values (missing values):

- If numerical data we will use mean or median
- If categorical data we will use mode.

* mean is effected by outliers

ex:- 1, 2, 3, 4, 5, 6, 100 ← outlier

$$\text{mean without outlier} = \frac{1+2+3+4+5+6}{6}$$

$$= 3.5$$

mean with outlier $= \frac{1+2+3+4+5+6+100}{7}$ → huge jump

$= 17.28...$

↓ Corrupted mean

- The outlier will pull the mean towards itself and corrupts it.

* median is not effected by outliers.
* median is effected only if more than 50% points are outliers.

ex:- 1, 2, 3, 4, 5, 6, 100 ← outlier

median without outlier $=$ 1 2 3 4 5 6

$= \frac{3+4}{2} = 3.5$

median with outlier $=$ 1 2 3 4 5 6 100 → not a huge jump

$= 4$

ex:- 1, 2, 100, 101, 102, 103

median without outlier $=$ 1 2

$= \frac{1+2}{2} = 1.5$

median with outlier $=$ 1 2 100 101 102 103

$$= \frac{100+101}{2} = 100.5$$

(Corrupted since outliers > 50%.)

** So if outliers are present we will use median to fill the null values.

* Once dataset is given we will check if missing values are there
  - check if outliers are there
  - if outliers are there use median to fill the missing values
  - if there are no outliers use mean to fill the missing values.

fillna( ): This function will fill the null values of the entire DataFrame with the value provided irrespective of numerical or categorial.
  - ↳ this is logically not right

instead we will use fillna({ }) ← this dictionary will contain columns to fill and what values to fill it with

** These are temporary manipulations, so if we want to make these changes permanently we need to use inplace = True.

```
datac.fillna({"Age":1})
```

	Name	Age	Gender	Height	weight
0	P1	23.0	m	154.0	43.0
1	P2	21.0	f	123.0	34.0
2	P3	1.0	NaN	NaN	NaN
3	P4	24.0	m	145.0	12.0
4	P5	1.0	m	135.0	NaN
5	P6	12.0	NaN	124.0	45.0

```
datac.fillna({"Age":1,"Gender":"no","Height":12,"weight":100})
```

↳

	Name	Age	Gender	Height	weight
0	P1	23.0	m	154.0	43.0
1	P2	21.0	f	123.0	34.0
2	P3	1.0	no	12.0	100.0
3	P4	24.0	m	145.0	12.0
4	P5	1.0	m	135.0	100.0
5	P6	12.0	no	124.0	45.0

```
# we will use mean, median & mode to fill.
datac.fillna({'Age': datac['Age'].mean(), 'Gender':
              datac['Gender'].mode()[0], 'Height':
              datac['Height'].median(),'weight':
              datac['weight'].median()})
```

↳

	Name	Age	Gender	Height	weight
0	P1	23.0	m	154.0	43.0
1	P2	21.0	f	123.0	34.0
2	P3	20.0	m	135.0	38.5
3	P4	24.0	m	145.0	12.0
4	P5	20.0	m	135.0	38.5
5	P6	12.0	m	124.0	45.0

mode() gives series so we cannot directly use mode()

```
datac["Gender"].mode()
↳ 0  m
  Name: Gender, dtype: object

datac["Gender"].mode()[0]
↳ 'm'
```

* All manipulations temporary.
* If we want to edit the data use inplace=True.

* So datac.fillna({'Age': datac['Age'].mean(), 'Gender': datac['Gender'].mode()[0], 'Height': datac['Height'].median(), 'weight': datac['weight'].median()}, inplace=True)

will change datac permanently.

Methods inside fillna()

- forward fill method (fill null value with above num)
  (ok to use)
- backward fill method (fill null value with below num)
  (ok to use)
- axis=1 (rowwise) fill method (will fill from before row)
  don't ever use: no logical sense

`datac.fillna(method="ffill")`

	Name	Age	Gender	Height	weight
0	P1	23.0	m	154.0	43.0
1	P2	21.0	f	123.0	34.0
2	P3	21.0	f	123.0	34.0
3	P4	24.0	m	145.0	12.0
4	P5	24.0	m	135.0	12.0
5	P6	12.0	m	124.0	45.0

`datac.fillna(method="bfill")`

	Name	Age	Gender	Height	weight
0	P1	23.0	m	154.0	43.0
1	P2	21.0	f	123.0	34.0
2	P3	24.0	m	145.0	12.0
3	P4	24.0	m	145.0	12.0
4	P5	12.0	m	135.0	45.0
5	P6	12.0	NaN	124.0	45.0

`datac.fillna(method = "ffill", axis=1)` → makes no logical sense

	Name	Age	Gender	Height	weight
0	P1	23.0	m	154.0	43.0
1	P2	21.0	f	123.0	34.0
2	P3	P3 (1)	P3 (2)	P3 (3)	P3 (4)
3	P4	24.0	m	145.0	12.0
4	P5	P5 (1)	m	135.0	135.0 (1)
5	P6	12.0 (1)	12.0	124.0	45.0

* we can also limit how many times we want to fill same value while using 'ffill' or 'bfill'. default there is no limit to number of times it can fill.

`datac.fillna(method = "ffill", axis=1, limit=1)`

	Name	Age	Gender	Height	weight
0	P1	23.0	m	154.0	43.0
1	P2	21.0	f	123.0	34.0
2	P3	P3 (1) ✗	NaN	NaN	NaN
3	P4	24.0	m	145.0	12.0
4	P5	P5 (1)	m	135.0	135.0 (1)
5	P6	12.0 (1)	12.0	124.0	45.0

How to drop null values

dropna() - this will go row wise and will drop the entire row even if there is a single null value.

```
datac.dropna()
```

↳

	Name	Age	Gender	Height	weight
0	P1	23.0	m	154.0	43.0
1	P2	21.0	f	123.0	34.0
3	P4	24.0	m	145.0	12.0

instead if we want to say drop only when all the row are null values. → means if all null values

```
datac.dropna(how='all')
```

↳

	Name	Age	Gender	Height	weight
0	P1	23.0	m	154.0	43.0
1	P2	21.0	f	123.0	34.0
3	P4	24.0	m	145.0	12.0
4	P5	NaN	m	135.0	NaN
5	P6	12.0	NaN	124.0	45.0

* Temporary results, if we want modify data then use inplace = True

default of how = 'any' i.e even if one value in the row is null drop the row.

* thresh (another parameter in dropna())

** this checks non-null values.

i.e datac.dropna(thresh=1) → even if there is 1 non-null value then dont drop the row

	Name	Age	Gender	Height	weight
0	P1	23.0	m	154.0	43.0
1	P2	21.0	f	123.0	34.0
3	P4	24.0	m	145.0	12.0
4	P5	NaN	m	135.0	NaN
5	P6	12.0	NaN	124.0	45.0

datac.dropna(thresh=3)

	Name	Age	Gender	Height	weight
0	P1	23.0	m	154.0	43.0
1	P2	21.0	f	123.0	34.0
3	P4	24.0	m	145.0	12.0
5	P6	12.0	NaN	124.0	45.0

datac.dropna(thresh=4)

	Name	Age	Gender	Height	weight
0	P1	23.0	m	154.0	43.0
1	P2	21.0	f	123.0	34.0
3	P4	24.0	m	145.0	12.0

* If we want to drop name columns from the manipulations

```
datac.columns[1:]
```

↳ `Index(['Age', 'Gender', 'Height', 'weight'], dtype='object')`

```
datac[datac.columns[1:]]
```

↳

	Age	Gender	Height	weight
0	23.0	m	154.0	43.0
1	21.0	f	123.0	34.0
2	NaN	NaN	NaN	NaN
3	24.0	m	145.0	12.0
4	NaN	m	135.0	NaN
5	12.0	NaN	124.0	45.0

How to replace to handle unnecessary values or irrelevant values

- we can fill them with mean, median or mode.
- replace() function will first take all unnecessary values.
- we need to give a value with which we want to replace.

* we need to keep a note of the datatype of the values we are accessing to replace.

```
data = pd.read_csv(r"C:\.....\new_csv2.csv")
```

	Name	Age	Gender	Height	weight
0	P1	23	m	154	43
1	P2	21	f	19999999	34
2	P3	hi	23	m	f
3	P4	24	m	145	12
4	P5	NAN	m	135	NAN
5	P6	12	NAN	124	45

```
data["Age"].apply(type)
```

```
0    <class 'str'>
1    <class 'str'>
2    <class 'str'>
3    <class 'str'>
4    <class 'float'>
5    <class 'float'>
Name: Age, dtype: object
```

data.replace(['hi', '23', '19999999', 'f', 'm'], 0)

↳

	Name	Age	Gender	Height	weight
0	P1	0	0.0	154	43
1	P2	21	0.0	0	34
2	P3	0	0.0	0	0
3	P4	24	0.0	145	12
4	P5	NaN	0.0	135	NaN
5	P6	12	NaN	124	45

* This is wrong as unnecessary values in one column can be useful data in another column. replace() changes the complete DataFrame.

So for every column unnecessary values can get their own value to replace with using dictionary.

data.replace({'Age': 'hi', 'Gender': 23, 'Height': ['19999999', 'm']}, 0)

↳

	Name	Age	Gender	Height	weight
0	P1	23	m	154	43
1	P2	21	f	0	34
2	P3	0	0	0	f
3	P4	24	m	145	12
4	P5	NaN	m	135	NaN

5	P6	12	NAN	124	45

or Best we can replace unnecessary values with np.nan using numpy and then fill NAN values.

```
import numpy as np
datac = data.replace({'Age': 'hi', 'Gender': '23', 'Height':
        ['19999999', 'm'], 'weight': 'f'3, np.nan)
datac
```

↳

	Name	Age	Gender	Height	weight
0	P1	23	m	154	43
1	P2	21	f	NAN	34
2	P3	NAN	NAN	NAN	NAN
3	P4	24	m	145	12
4	P5	NAN	m	135	NAN
5	P6	12	NAN	124	45

Now fill NAN using previous learned topics

```
datac.fillna(method = "ffill", in place = True)
datac
```

↳

	Name	Age	Gender	Height	weight
0	P1	23	m	154	43
1	P2	21	f	154	34
2	P3	21	f	154	34
3	P4	24	m	145	12
4	P5	24	m	135	12

5	P6	12	m	124	45

set_index()

* using set_index() function we can set any Colum as a index in a DataFrame.
* This is temporary if we want to use it we can run the function with inplace = True
* This is important as we can change index to what we want.
* Better use for numerical data.
* If we have to use for categorical data for manipulation purposes. Keep note that categorical data will have duplicates.

```
data.set_index("Name", inplace=True)
data
```

↳

Name	Age	Gender	Height	weight
P1	23	m	154	43
P2	21	f	1999999	34
P3	hi	23	m	f
P4	24	m	145	12
P5	NAN	m	135	NAN
P6	12	NAN	124	45

reset_index( )

This will reset the index to the original index.

```
data.reset_index(inplace=True)
data
```

	Name	Age	Gender	Height	weight
0	P1	23	m	154	43
1	P2	21	f	19999999	34
2	P3	hi	23	m	f
3	P4	24	m	145	12
4	P5	NaN	m	135	NaN
5	P6	12	NaN	124	45

If we want all male data

ex:- select where Gender = 'm'

```
data.set_index("Gender",inplace=True)
data.loc["m"]
```

Gender	Name	Age	Height	weight
m	P1	23	154	43
m	P4	24	145	12
m	P5	24	135	12
m	P6	12	124	45