Dt:- 09/5/22

# Deep Learning :-

Used for

1. Artificial neural network [ANN] [Regression/classification]
2. Convolution neural networks [CNN] [computer vision & image classification]
3. Recurrent neural network [RNN] [Time series analysis]

and

long short term memory [LSTM]

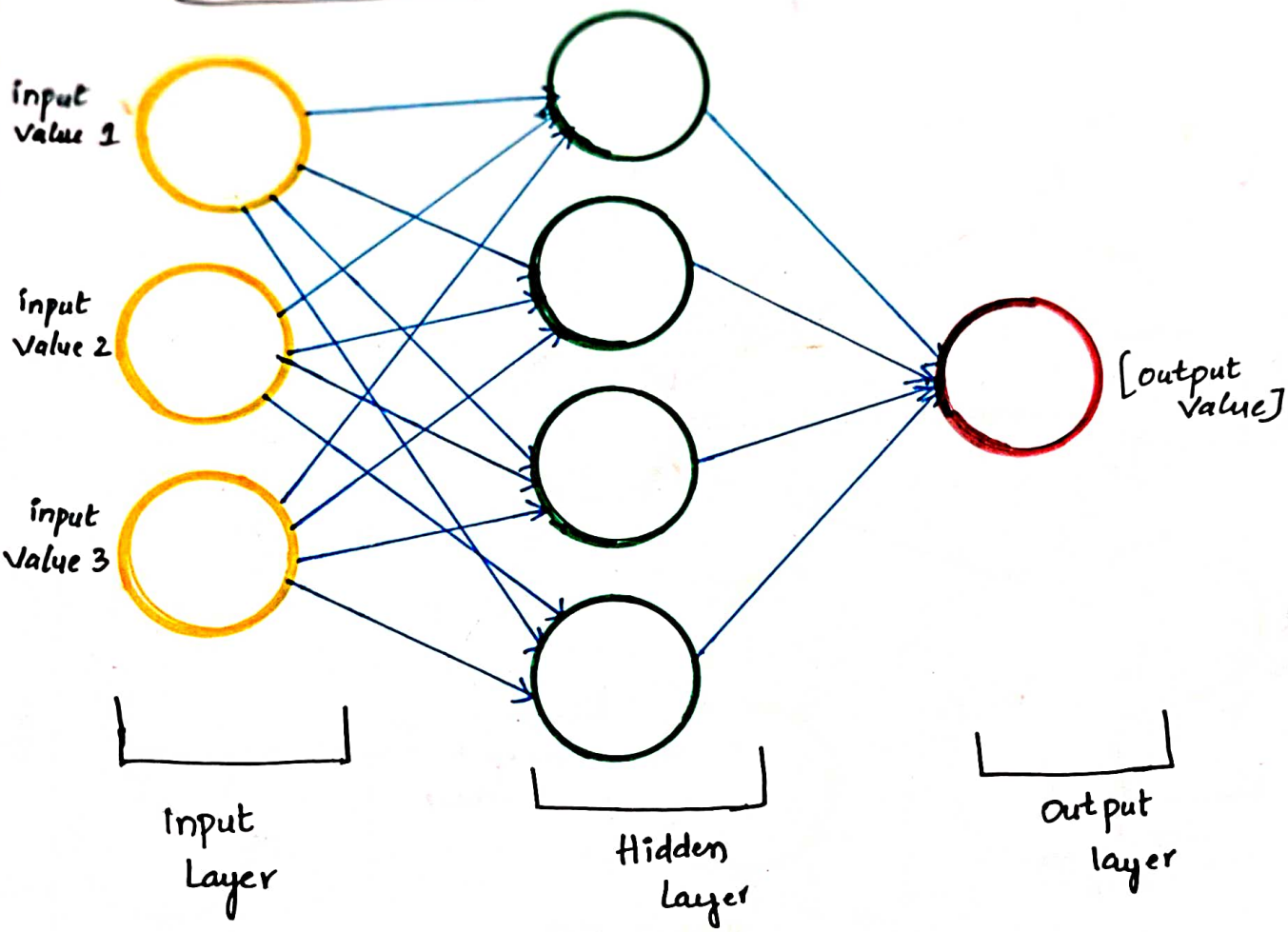


## 1. Artificial neural network

* Artificial intelligence :- application which can do its own task with out any human intervention

Ex:- netflix app
- Self driving Cars
- amazon application
- Sofia (robot)

* Neural network :- Connecting the neurons working as one connection is known as "Neural network".




Ex :-

| $X_1$ | $X_2$ | $X_3$ | $y$ |
|---|---|---|---|
| - | - | - | - |
| - | - | - | - |
| - | - | - | - |

Here, we have

* Three input layers :

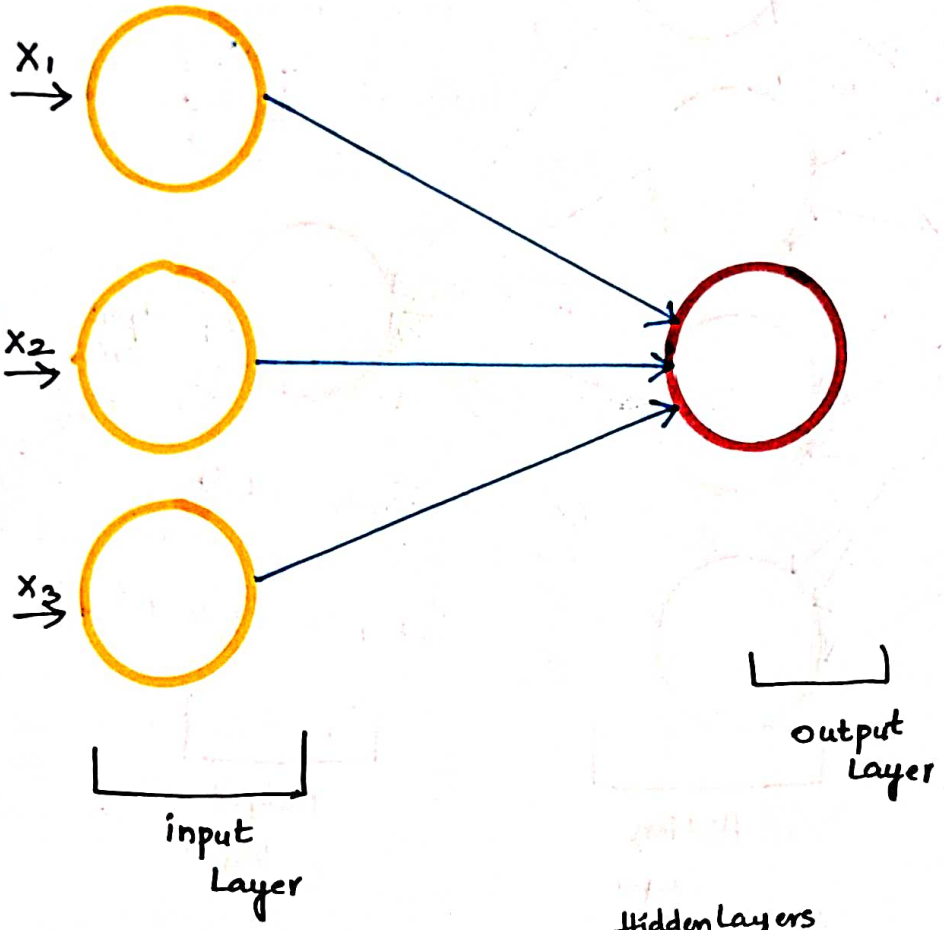
$x_1$ $x_2$ $x_3$

* One output layer :

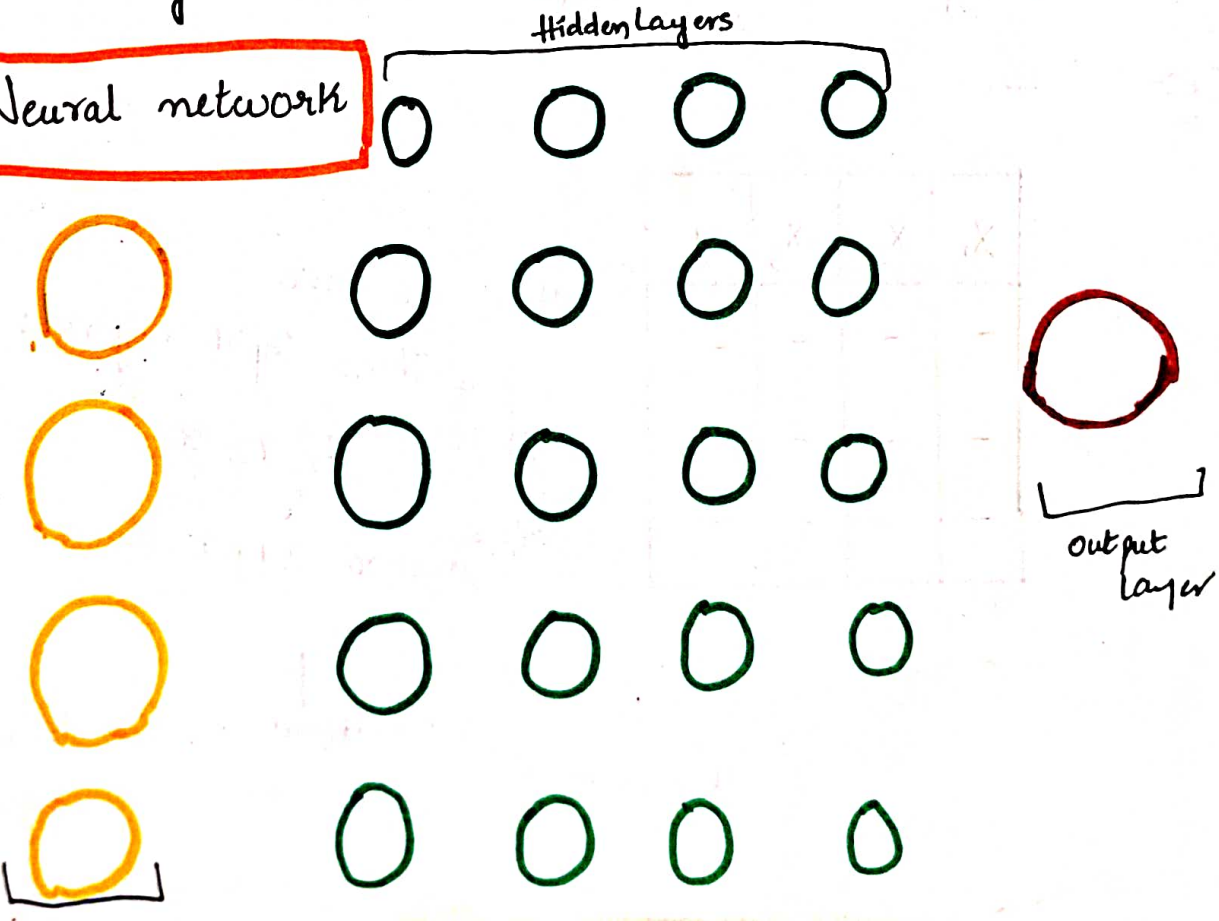
"y"

"y" can be
- continous
- discrete
  - binary
  - multiclass

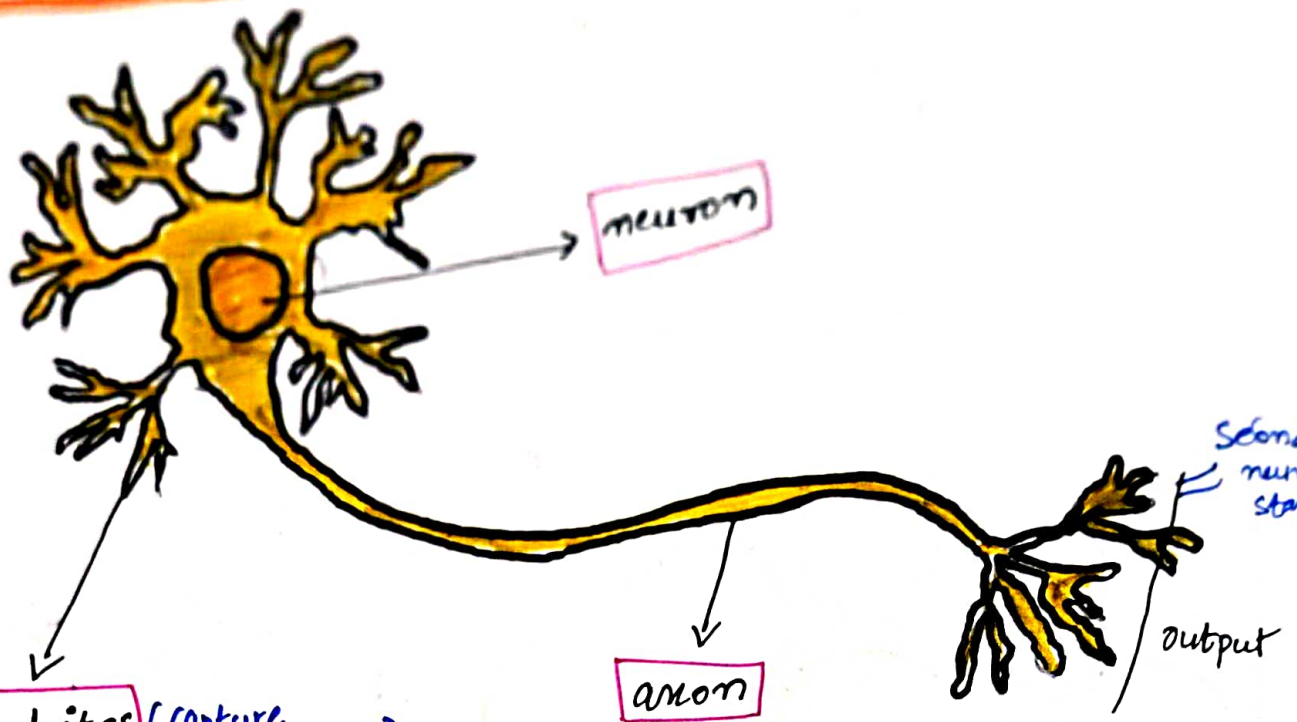
1. **Perceptron** :- connecting Input Layer to output layers directly without any hidden Layers.




2. **Neural network**



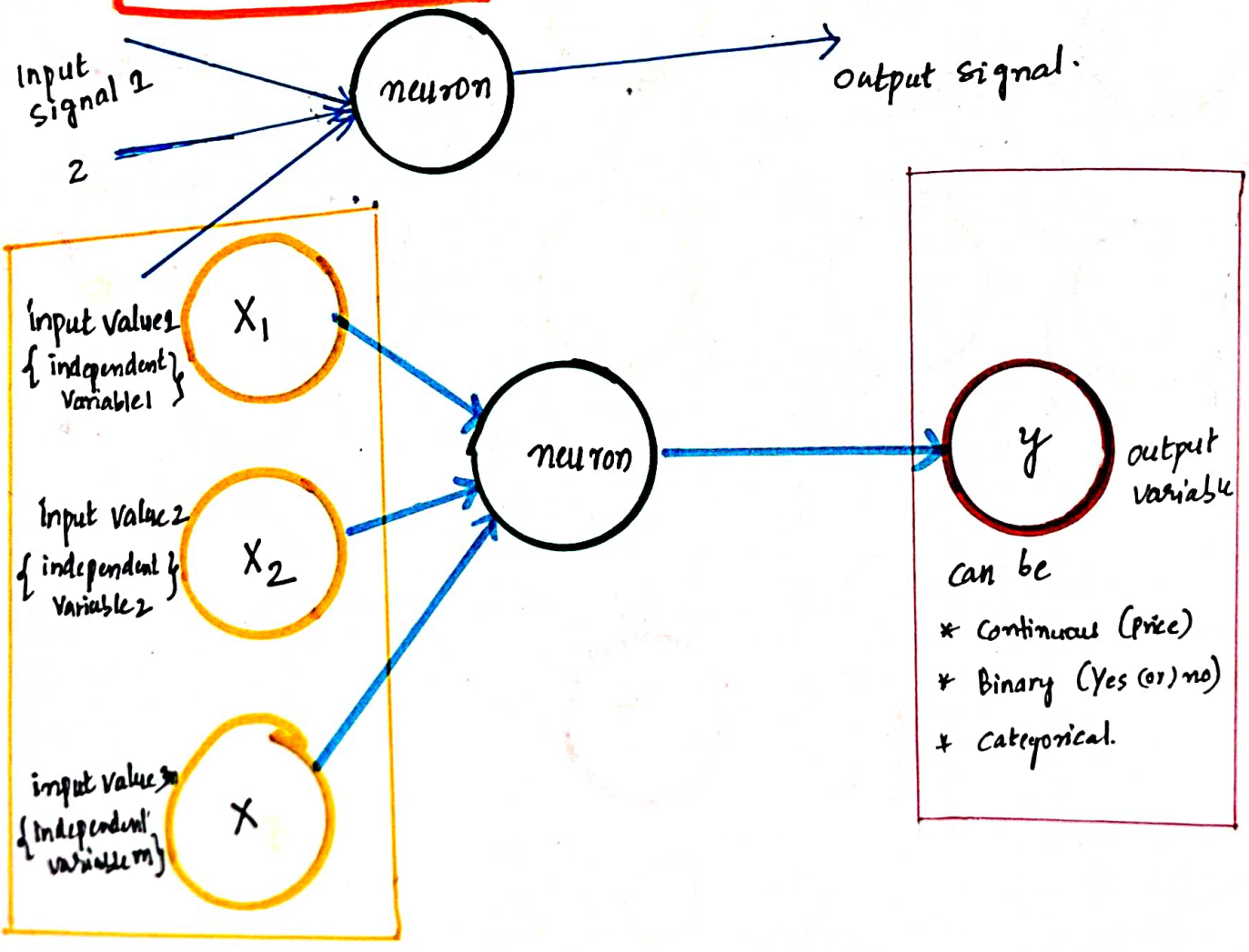
* **The neuron** connected to each other to pass the information




**Dendrites** (capture information) (input) from other neuron

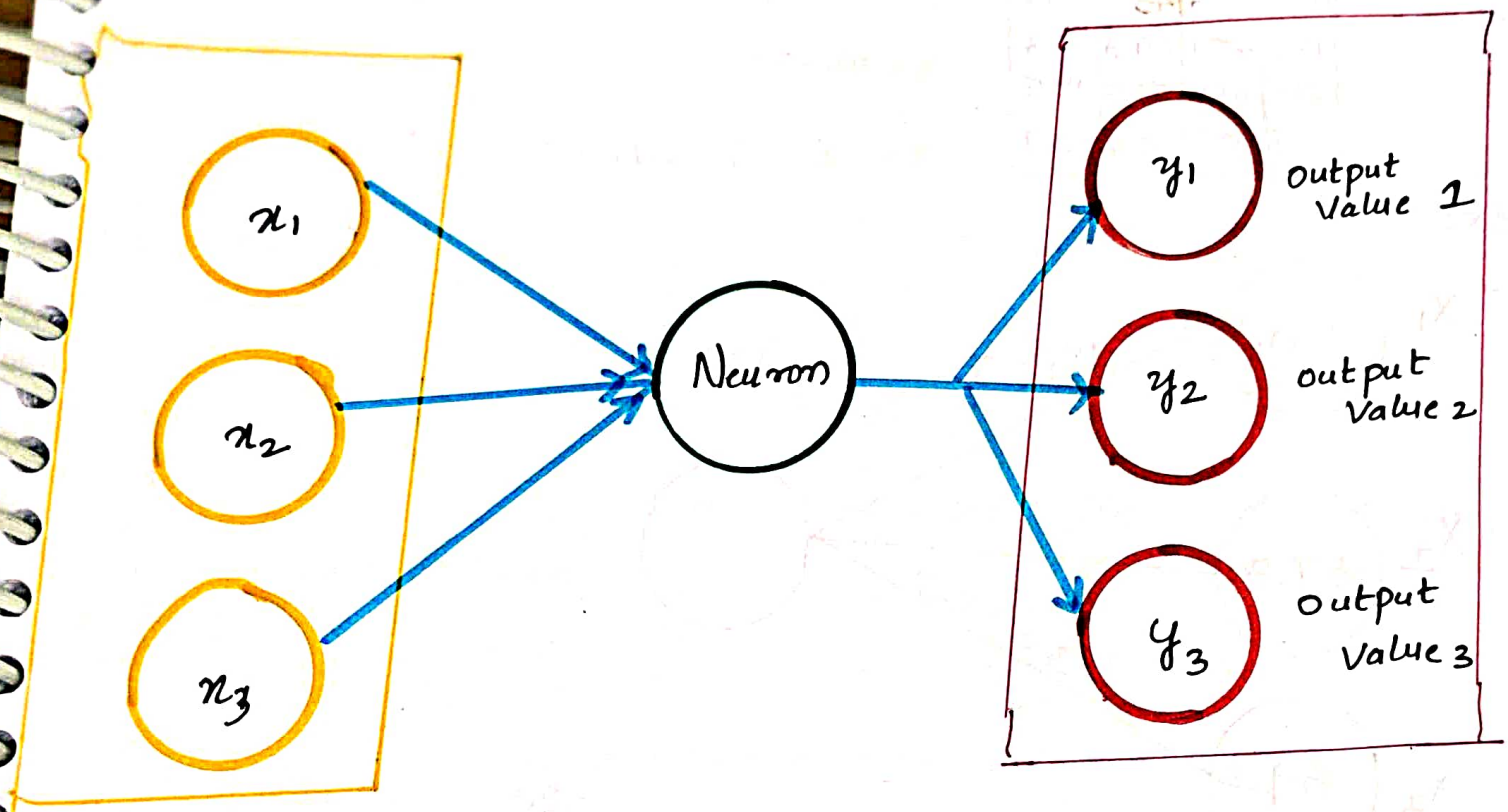
**axon** (pass the information)

* **Artificial Neuron**



* Binary classification → o/p neuron ⇒ 1 neuron
* Regression → output neuron ⇒ 1 neuron
* Multiclassification → output neuron ⇒ no. of categories (or) classes.

Ex:- Penguin Age, chi, Gentoo




* **Weights**

Machine learning :- $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 = (y - \hat{y})_{min}$

deep learning :- $b + w_1 x_1 + w_2 x_2 + w_3 x_3 = (y - \hat{y})_{min}$

→ it can be written as

$$\sum w_i x_i + b$$

Ex:- linear Regression.

| $X_1$ TV | $X_2$ Radio | $X_3$ Newspaper | $y$ Sales |
|---|---|---|---|
| 230.1 | 37.8 | 69.2 | 22.1 |
| 44.5 | 39.3 | 45.1 | 10.4 |
| 17.2 | 45.9 | 69.3 | 9.3 |
| 151.5 | 41.3 | 58.5 | 18.5 |
| 180.8 | 10.8 | 58.4 | 12.9 |

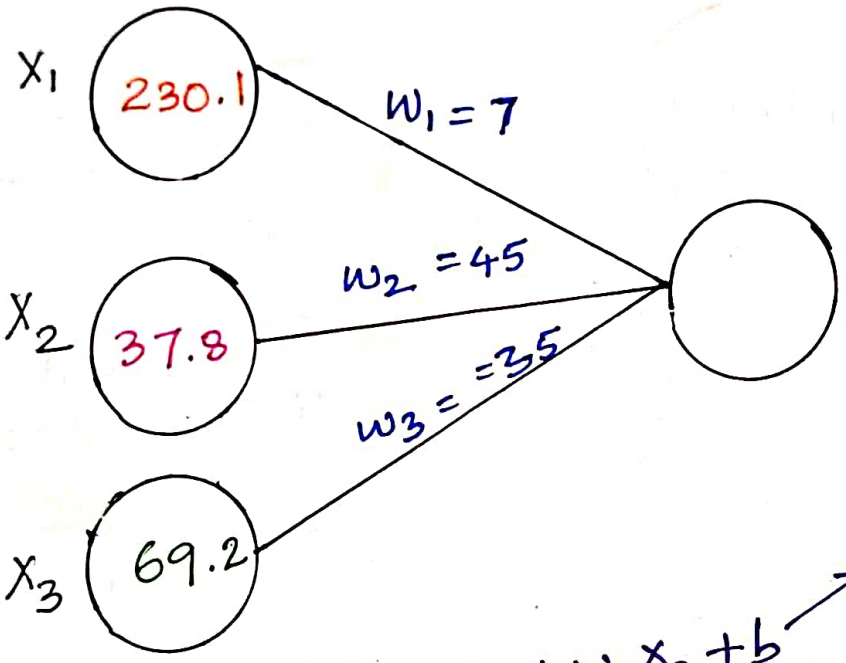
we consider weights

Ex:- Equation $(2x+5=9)$

Initialization of value

| | $\hat{y}$ | $y$ | error |
|---|---|---|---|
| $x=100 \Rightarrow 2(100)+5$ | 205 | 9 | -196 |
| $x=99 \Rightarrow 2(99)+5$ | 203 | 9 | -194 |
| $x=98 \Rightarrow$ . | . | . | . |
| $x=2 \Rightarrow 2(2)+5$ | 9 | 9 | 0 |

Ex:-




* $w_1x_1 + w_2x_2 + w_3x_3 + b$ → bias

* $[7 \times 230] + [45 \times 37] + [35 \times 69] + 100$

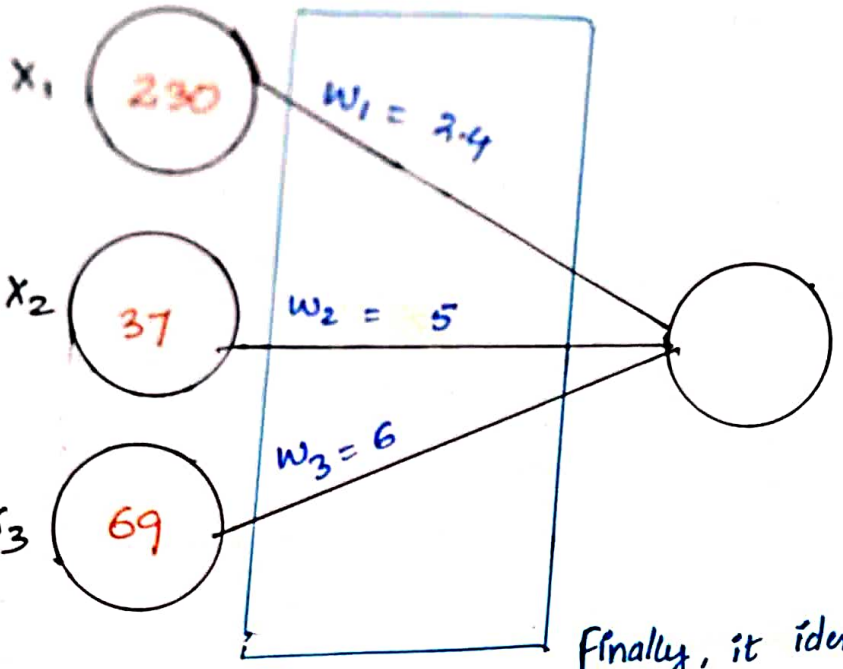
$\hat{y} = 5790.$

| $y$ | $\hat{y}$ | $y-\hat{y}$ |
|---|---|---|
| 22.1 | 5790 | 5767.9 |

So, $(y-\hat{y})$ Largest amount of difference is there,

- weights will be readjusted

* weights will readjusted

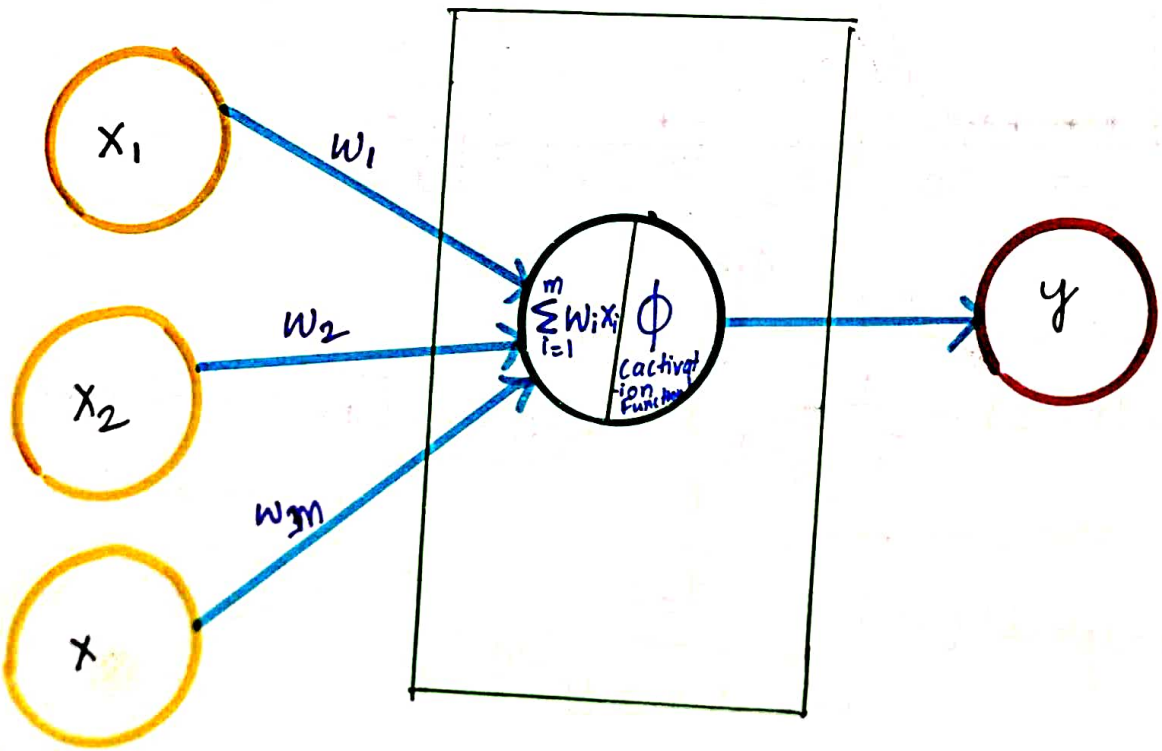



Finally, it identify the best weights which gives sum of square Errors to minimum.

Here, it's identify only weights.

$(SSE)_{min}$.

* **Steps in neuron**

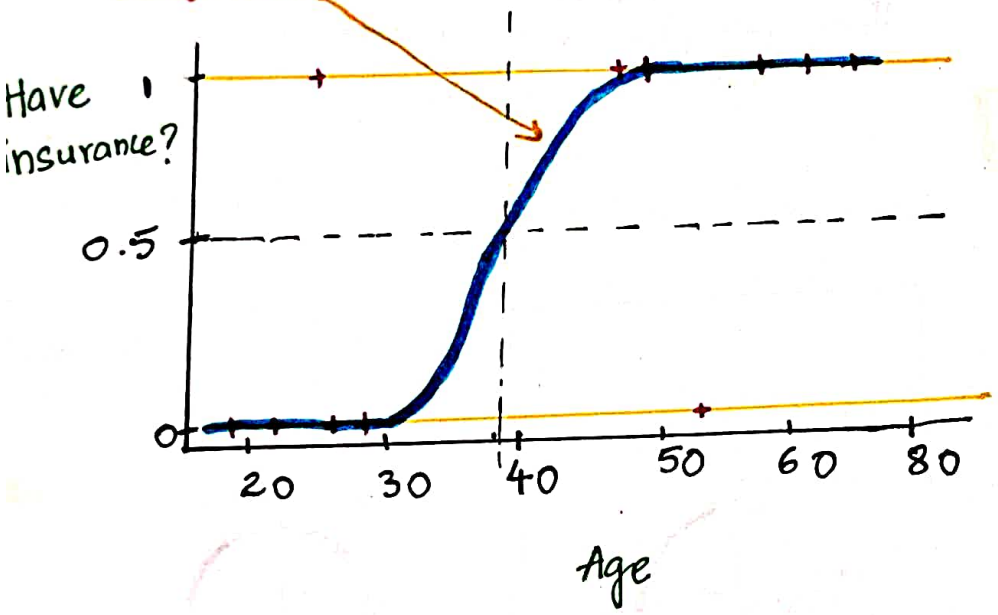


Ex :- **Binary classification**

| age | have_insurance |
|---|---|
| 22 | 0 |
| 25 | 0 |
| 47 | 1 |
| 52 | 0 |
| 46 | 1 |
| 56 | 1 |
| ⋮ | ⋮ |

Given, an age of a person, come up with a function that can predict if person will buy insurance (or) not.

* **Sigmoid** or **Logit function**




$$\text{Sigmoid}(z) = \frac{1}{1+e^{-y}}$$

$\therefore e$ = Euler's number $\sim 2.71828$

- $\text{Sigmoid}(200) = \frac{1}{1+2.71^{-200}} =$ almost close to 1

$\because -200$

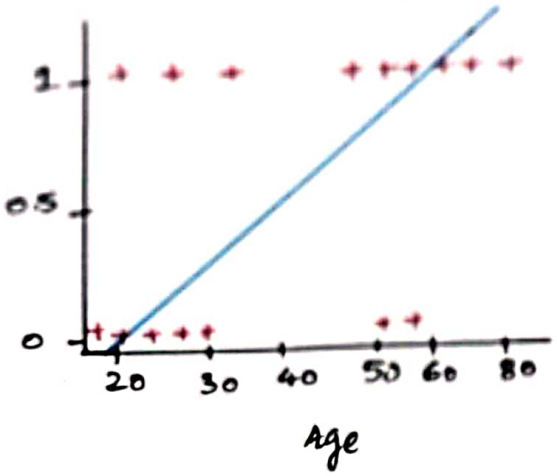
- $\text{Sigmoid}(-200) = \frac{1}{1+2.71^{+200}} =$ almost close to 0

* Sigmoid function converts input into range 0 to 1.

## STEP : 1

$y = m * x + b$

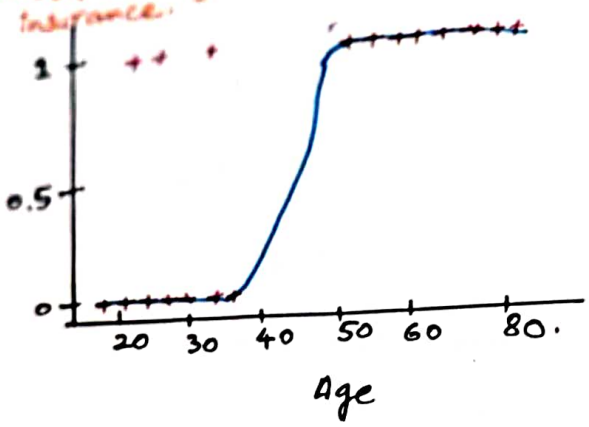
$x$ → age




## step : 2

$Z = \frac{1}{1 + e^{-y}}$

$Z$ → If person will by insurance.




$y = 0.042 * x - 1.53$

$x$ → Age.

- **STEPS :-**

| $y = 0.042 * x - 1.53$ | $Z = \frac{1}{1 + e^{-y}}$ |
|---|---|

- Value < 0.5 = Person will n Buy
- Value > 0.5 = Person will buy insur

Age = 35

| $y = 0.042 \times 35 - 1.53$ <br> $= -0.06$ | $Z = \frac{1}{1 + 2.71^{0.06}}$ <br> $= 0.48$ |
|---|---|

→ 0.48

Age = 43

| $y = 0.042 \times 43 - 1.53$ <br> $= 0.276$ | $Z = \frac{1}{1 + 2.71^{-0.276}}$ <br> $= 0.568$ |
|---|---|

→ 0.57

$$y = 0.042 * x - 1.53$$ (b)

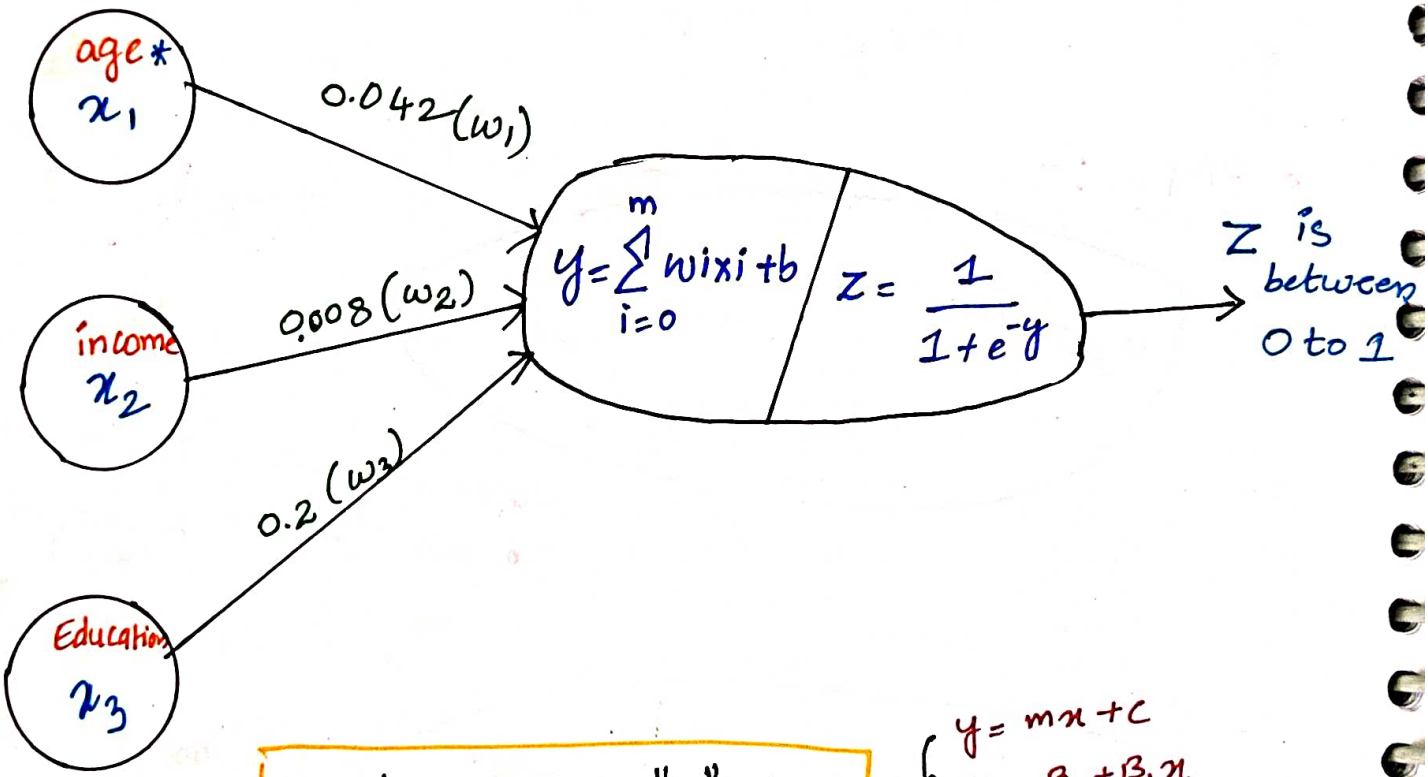
↳ age

$$y = [0.042 * x_1] + [0.008 * x_2] + [0.2 * x_3] - 1.53$$ (b)

↳ Age ↳ income ↳ Education.

$$y = [w_1 * x_1] + [w_2 * x_2] + [w_3 * x_3] + b$$

$$y = \sum_{i=0}^{m} w_i x_i + b$$




* "weights" and "b" values can be +ve, -ve

$y = mx + c$

$y = \beta_0 + \beta_1 x$

$y = w_1 x_1 + b$

all are same (but change is notation)

Ex:-

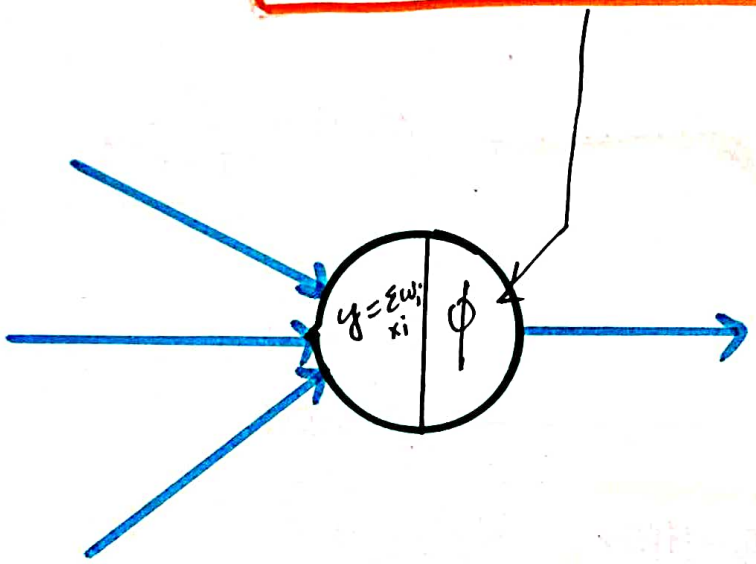
| x | y |
|---|---|
| 1 | 7 |
| 2 | 17 |
| 3 | 27 |
| 4 | 37 |
| 5 | 47 |

$\therefore y = 10x - 3$

$= w_1 x_1 + b$ → w, b can be (+ve) (or) (-ve)

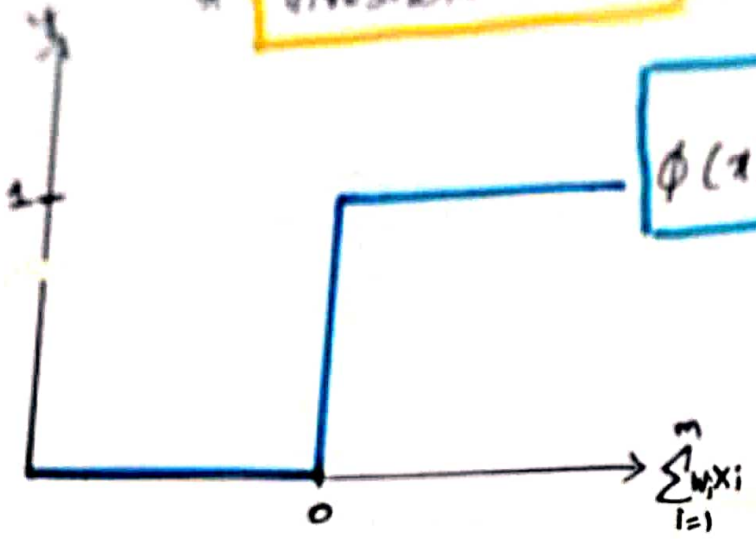
$= 10[x] + [-3]$

## Activation function



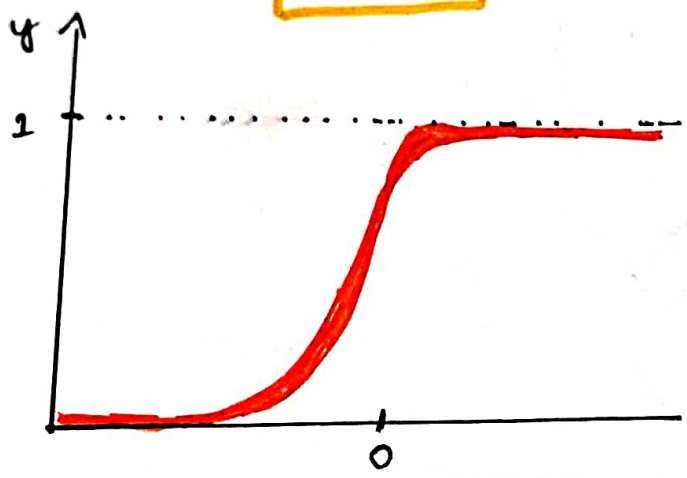

1. Threshold Function (or) step function
2. Sigmoid Function
3. Rectifier (or) relu.
4. Hyperbolic Tangent (tanh)

* Threshold function (or) Step function



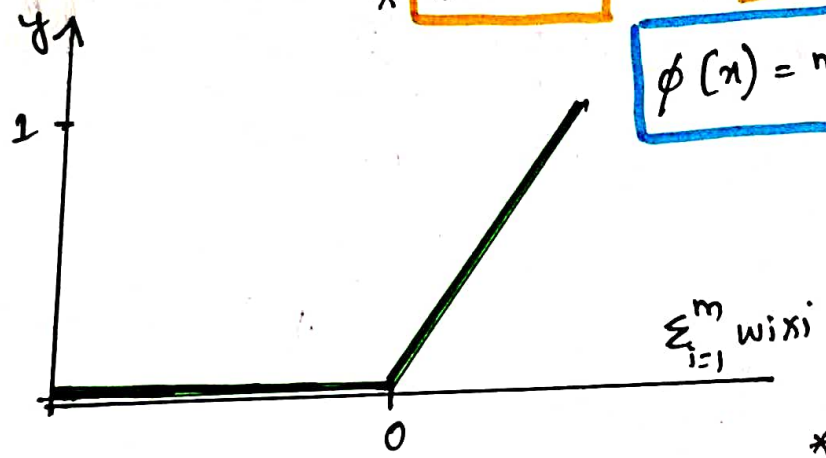

$$\phi(x) = \begin{cases} 1 & \text{if } y \geq 0 \\ 0 & \text{if } y < 0 \end{cases}$$

* Sigmoid



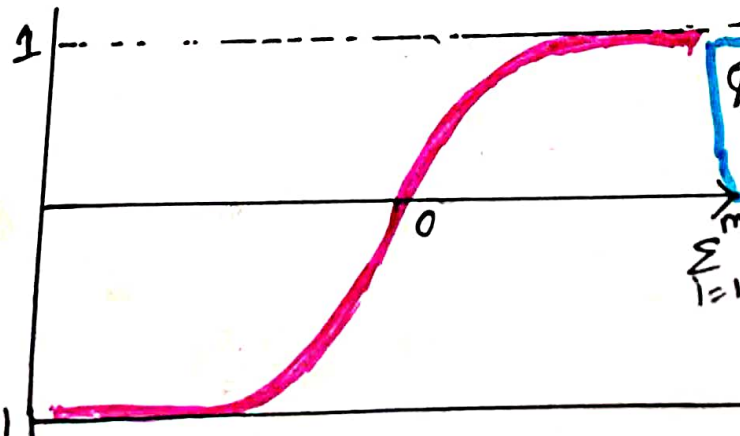

$$\phi(x) = \frac{1}{1+e^{-y}}$$

* Rectifier (or) Relu




$$\phi(x) = \max(x, 0)$$

* Hyperbolic Tangent (Tanh)




$$\phi(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

* Threshold function (or) Step function

$\phi($ • if $y \geq 0 = 1$
• if $y \leq 0 = 0)$

* Sigmoid function

$$\phi(x) = \frac{1}{1+e^{-y}}$$

$$\Rightarrow \frac{ey}{1+ey}$$

* Rectifier (or) relu

• Relu = Rectifier Linear Unit.

$$\phi(x) = \max(x, 0)$$

* Tanh (or) hyperbolic Tangent

$$\phi(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$$

Ex:- Regression → Relu

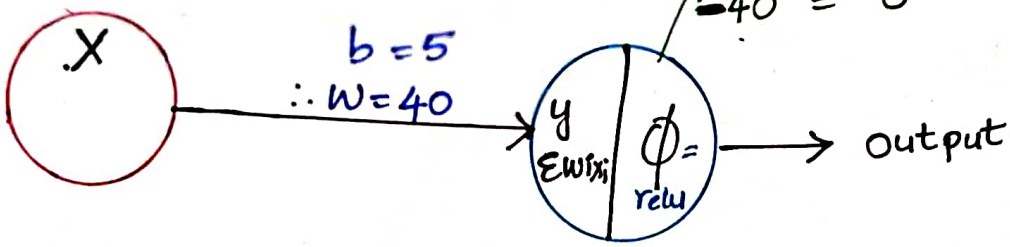
| x | y | $\hat{y}$ | $y-\hat{y}$ |
|---|---|---|---|
| 1 | 11 | 45 | - |
| 2 | 12 | 85 | - |
| 3 | 13 | 125 | - |
| 4 | 14 | 165 | - |
| 5 | 15 | 205 | - |

$\varepsilon(y-\hat{y})^2 = -$

$\hat{y}$

45 = 45

-40 = 0

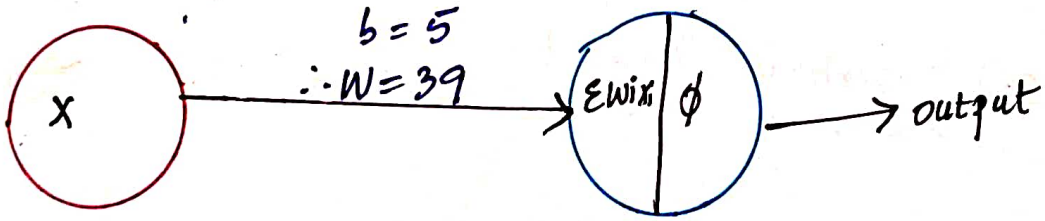


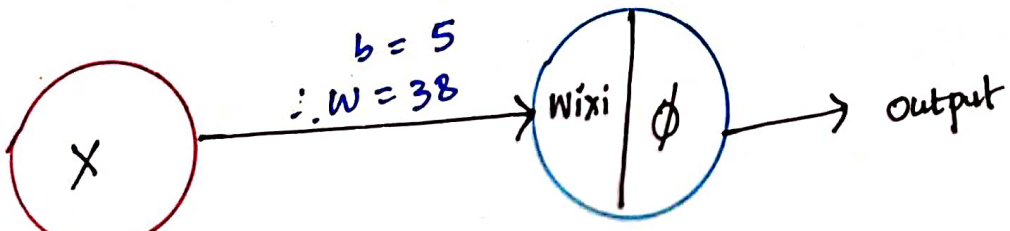

$\therefore \hat{y} = wx + b$

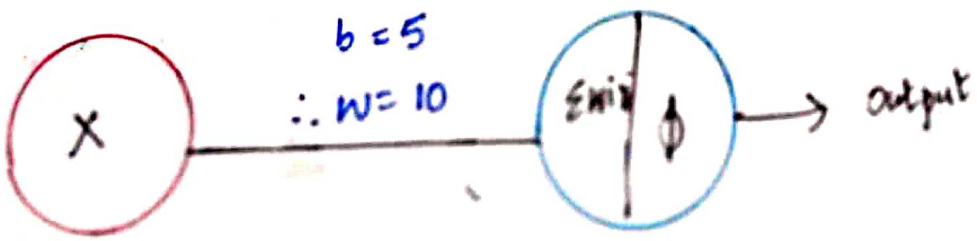
$x = 1, \quad \hat{y} = 40(1) + 5 = 45$

$x = 2, \quad \hat{y} = 40(2) + 5 = 85$

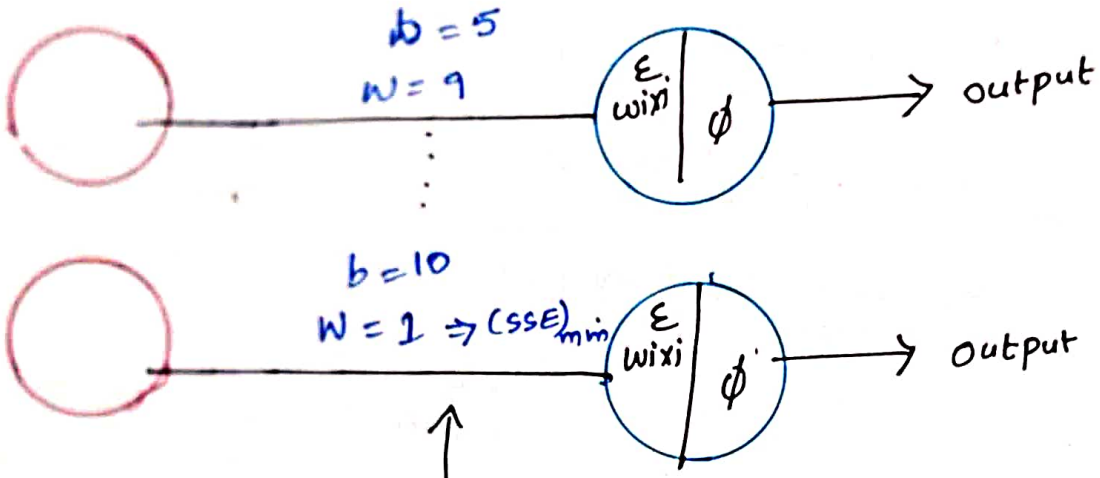
* adjusting weights



* after that it changes 'bias" ⇒ 'b'




$\therefore y = wx + b$

$\Rightarrow y = 1(x) + 10$

* iterations :- try & Errors [ Apply loop ]

For better answer.

---

## important points

- no. of input Neurons = no. of input features.
- no. of output Neurons =
  - Regression :- 1
  - binary class :- 1
  - Multiclass :- no. of categories

* Activation function of output Neuron :

1. regression ⇒ relu
2. binary classification ⇒ Sigmoid
3. Multiclass ⇒ Soft max

↓

Calculate the probability of Each class. and which of them having highest probablity we consider into that class.

Ex :- drug(x), drug(y), drug(z)

P(drug x)
Prob(drug y)
Prob (drug z)
} → highest

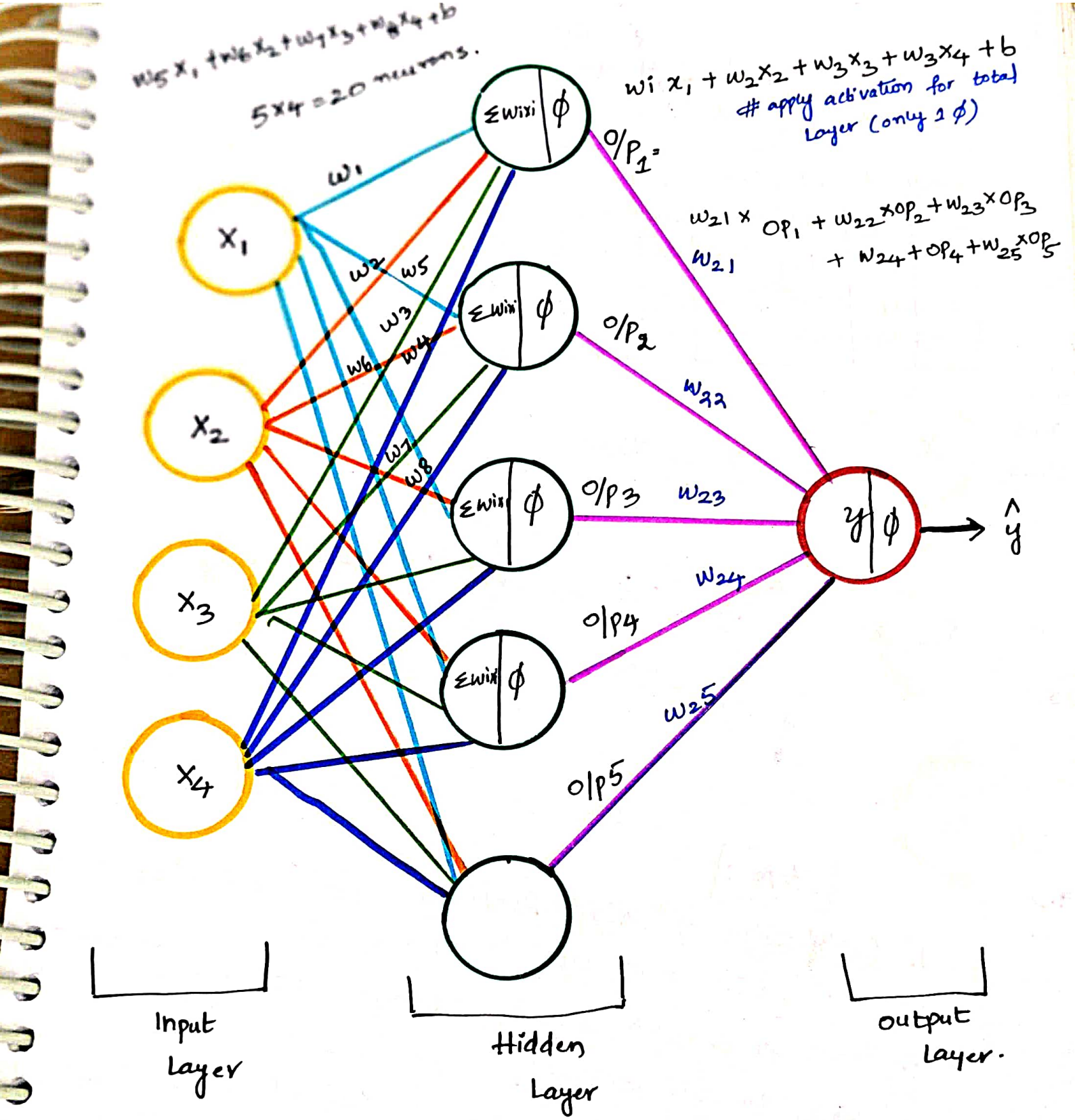
09/05/22
5:30pm.

Dt:-9/05/22
7:40pm.

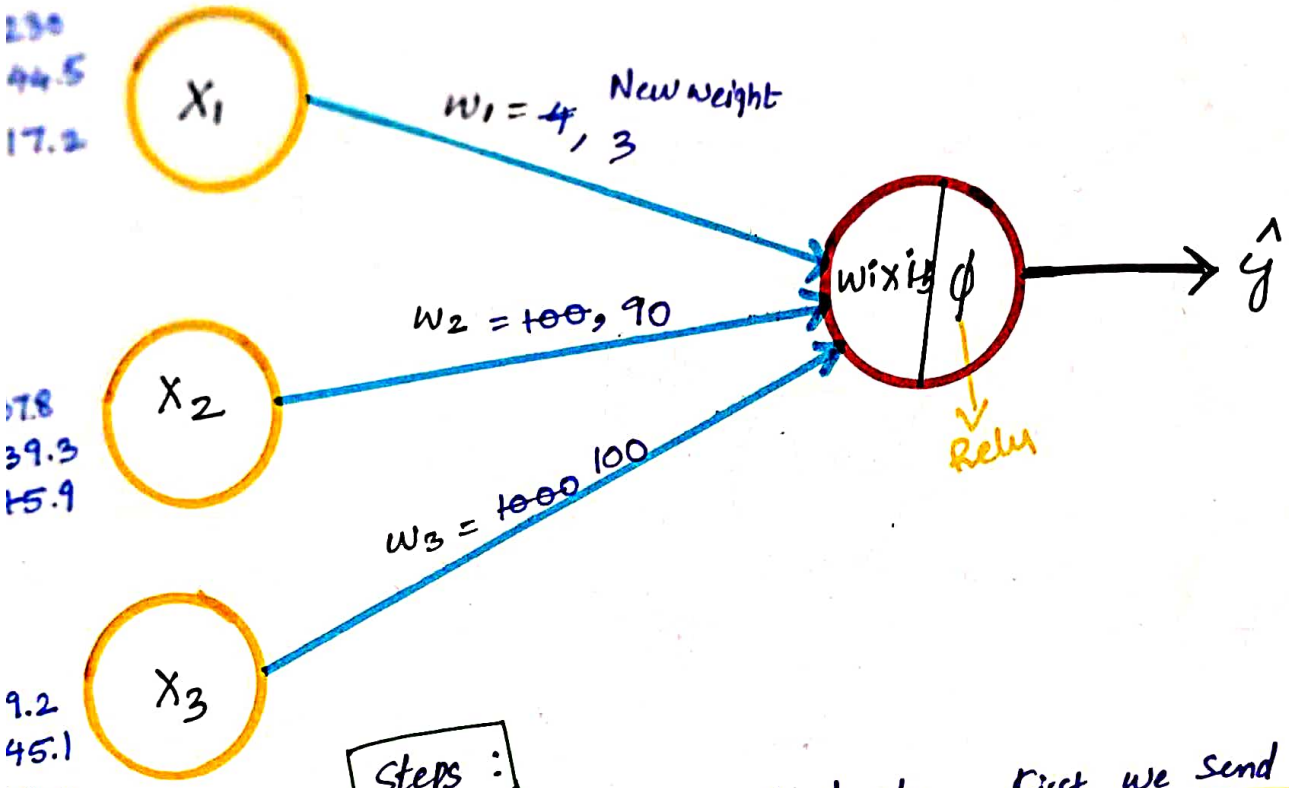
# How do neural network Work ?

# We can Take as per choice, Hidden Layers in Neural network

# and Same hidden Layers, we can Take as many as no. of neurons in the neural network.


w5x1 + w6x2 + w7x3 + w8x4 + b
5 x 4 = 20 neurons.
wi x1 + w2x2 + w3x3 + w3x4 + b
# apply activation for total Layer (only 1 ϕ)
Σwixi ϕ
o/p1
o/p2
o/p3
o/p4
o/p5
w21 x op1 + w22 x op2 + w23 x op3 + w24 + op4 + w25 x op5
w21
w22
w23
w24
w25
w1
w2
w3
w4
w5
w6
w7
w8
x1
x2
x3
x4
y ϕ
ŷ
Input Layer
Hidden Layer
output Layer.

| TV | Radio | Newspaper | Sales | $\hat{y}$ | $y-\hat{y}$ | $(y-\hat{y})^2$ |
|---|---|---|---|---|---|---|
| 230.1 | 37.8 | 69.2 | 22.1 | — | | |
| 44.5 | 39.3 | 45.1 | 10.4 | — | | |
| 17.2 | 45.9 | 69.3 | 9.3 | — | | |
| 151.5 | 41.3 | 58.5 | 18.5 | | | |
| 180.8 | 10.9 | 58.4 | 12.9 | | | |




**Steps :**

1. Intialization of weights Randomly, First we send 230.1, 37.8, 69.2 into input layer ($x_1, x_2, x_3$)
2. Than, we calculate $230 \times 4$, $37.8 \times 100$, $69.2 \times 1000$ with weights and going to calculate $(\hat{y})$.
3. In the neuron, we calculate $(wixi+b)$ and $(\phi)$(Relu) and output is predicted $(\hat{y})$
4. Similarly same as first, we continue with all data. --
5. Than, it Calculate Error $(y-\hat{y})$
6. After that, it Calculate $(y-\hat{y})^2$ (predicted - Actual)$^2$.

7. at last, we want (SSE)min.
8. In order to reduce Error, the weights will recapdate To 3,90,100
9. again, it will do same pattern, in loop. (In forward & Backward).
10. Finally it make weights such that, it gives (SSE)min. till that, it process.

Ex:- **Clear Example of Updating weights**

$2x+5=8$

| | $\hat{y}$ | $y$ | $y-\hat{y}$ |
|---|---|---|---|
| * $x=100$ | $2(100)+5=205$ | 8 | -197 |
| * $x=105$ | $2(105)+5=215$ | 8 | -207 |
| * $x=95$ | $2(95)+5=195$ | 8 | -187 |

As "x" value increasing, Error also increasing.
so, we decrease "x".

as "x" value decreasing, Error also decreasing.

$\frac{dx}{\Delta x}=5$ (change in $x=5$)

* $x=90$
* $x=85$
* $x=80$
* $x=5$
* $x=0$

| | | |
|---|---|---|
| $2(5)+5=15$ | 8 | -7 |
| $2(0)+5=5$ | 8 | 3 |

so, The Error is in between -5 & 8

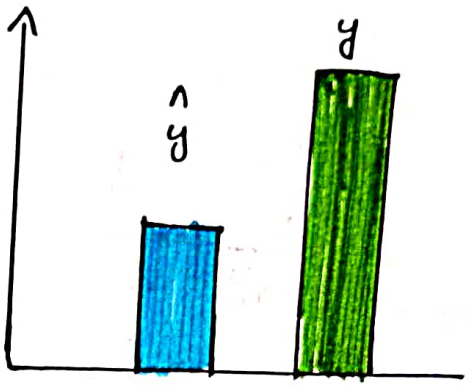
* $x=1$
* $x=2$
* $x=1.5$

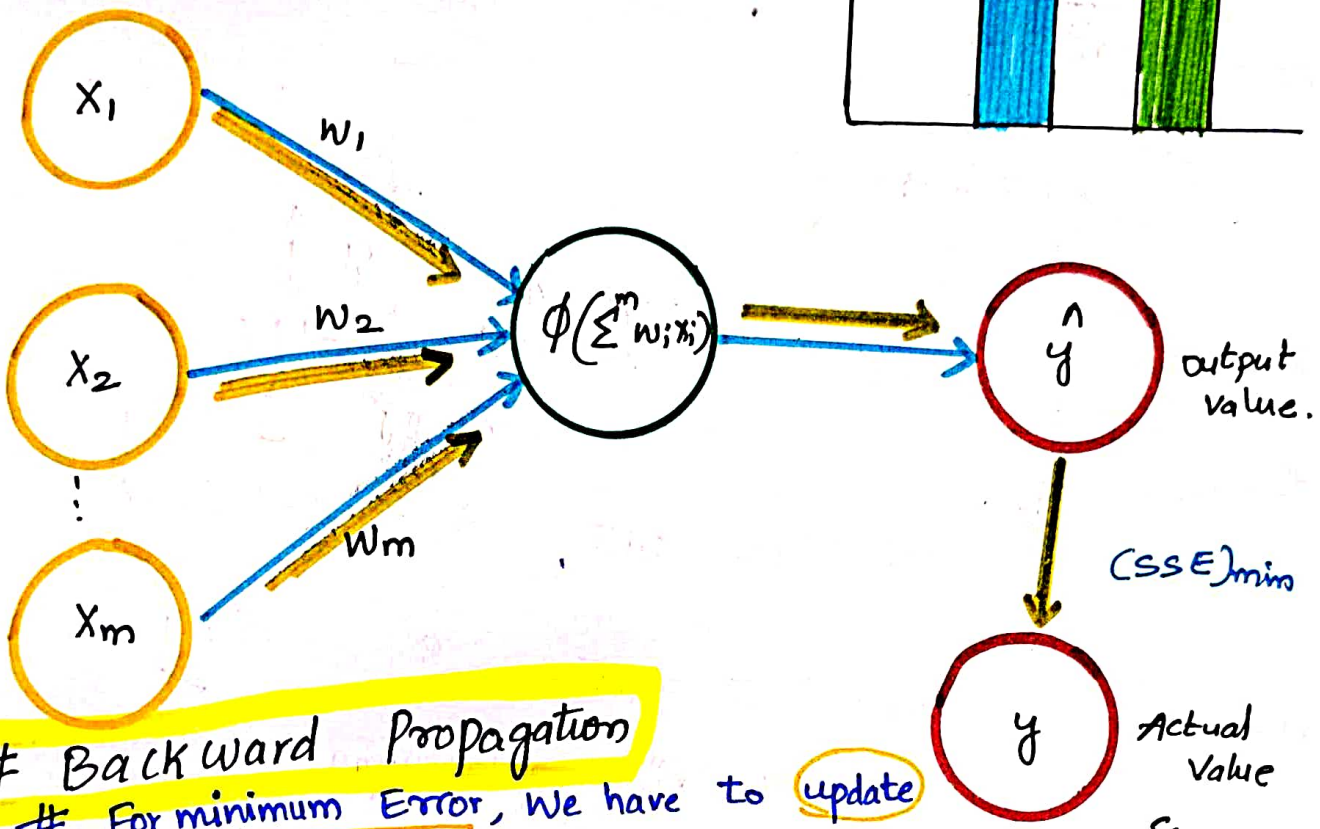
We updated:
till we get minimum value

# How do neural networks Learn ?

# Neural networks on based on Connections.

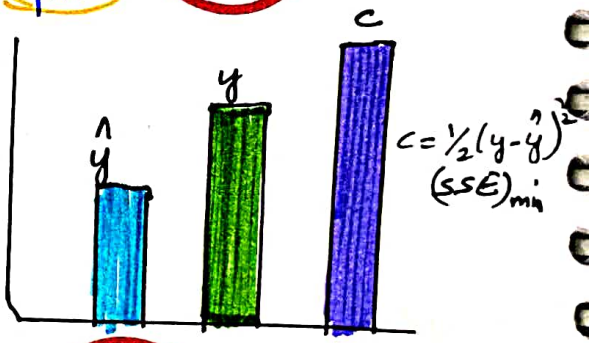
## # Forward propagation

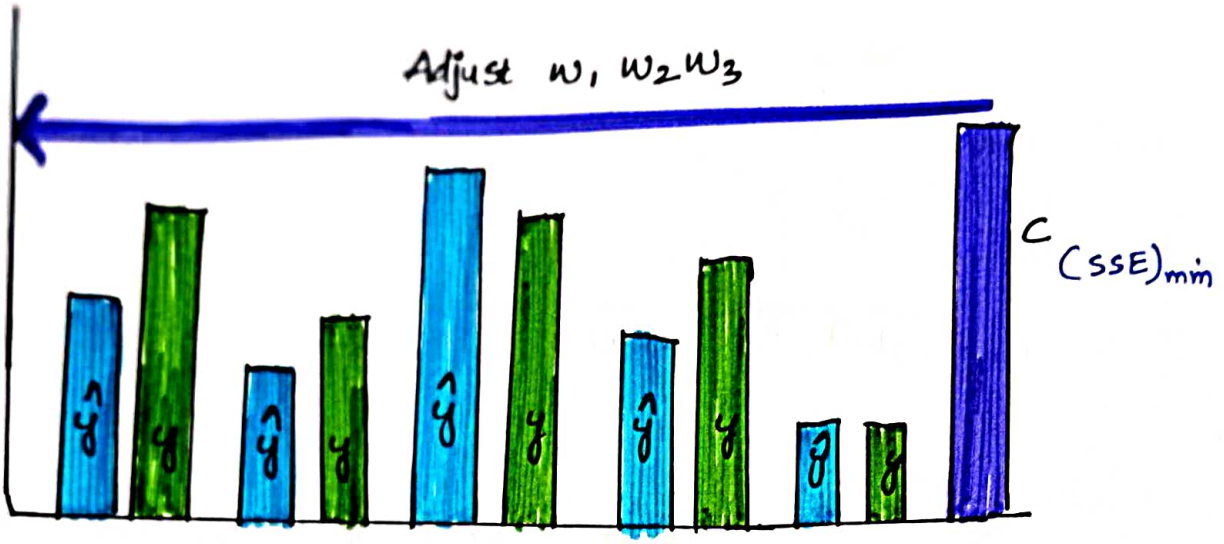



## # Backward Propagation

# For minimum Error, We have to update weights.






# forward propagation :-

First, Calculating $\Sigma w_i x_i + b$ and Calculating the output Variable in The forward direction is known as forward propagation.

# backward propagation :

Adjusting (or) updating the weights, coming backward direction is known as backward propagation.

intially, weights are assigned randomly, and from that we update weights.

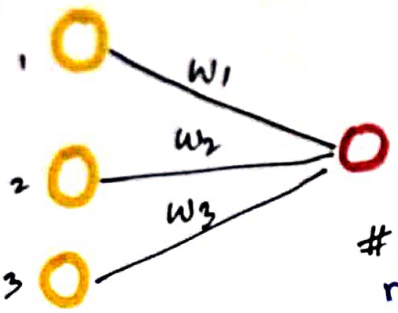
The weights will be updated in a way that $(SSE)_{min}$

# Weight initialization

1. **Uniform** = $\left[\frac{-1}{\sqrt{inputs}}, \frac{+1}{\sqrt{inputs}}\right]$

# we have select the weights, that forms, uniform distribution.

Ex:-



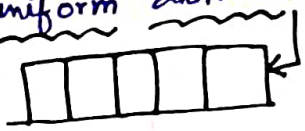

$\left[\frac{-1}{\sqrt{3}}, \frac{+1}{\sqrt{3}}\right]$

# we have select values in these only.

# These weights should form uniform distribution.

# Backend code.

np.random.uniform()



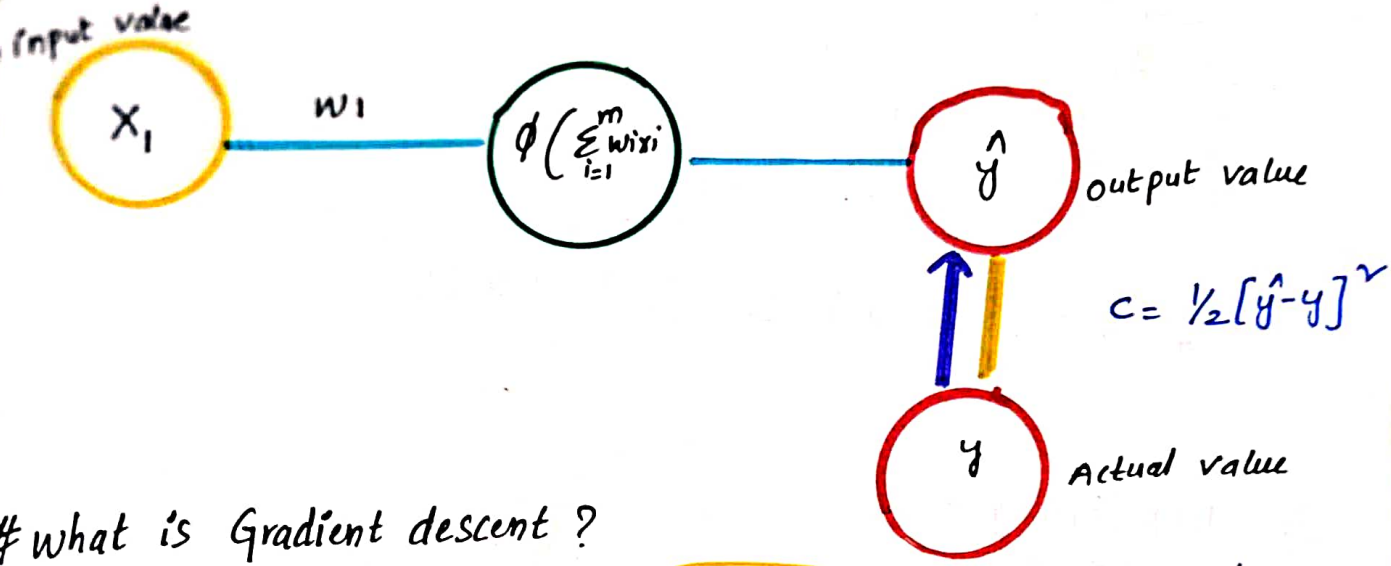
2. **Xavier Uniform** = $U\left[-\sqrt{\frac{6}{intout}}, +\sqrt{\frac{6}{intout}}\right]$

3. **Xavier normal** = $N\left[0, \sqrt{\frac{2}{intout}}\right]$

4. **He_int Uniform** = $U\left[-\sqrt{\frac{6}{in}}, \sqrt{\frac{6}{in}}\right]$

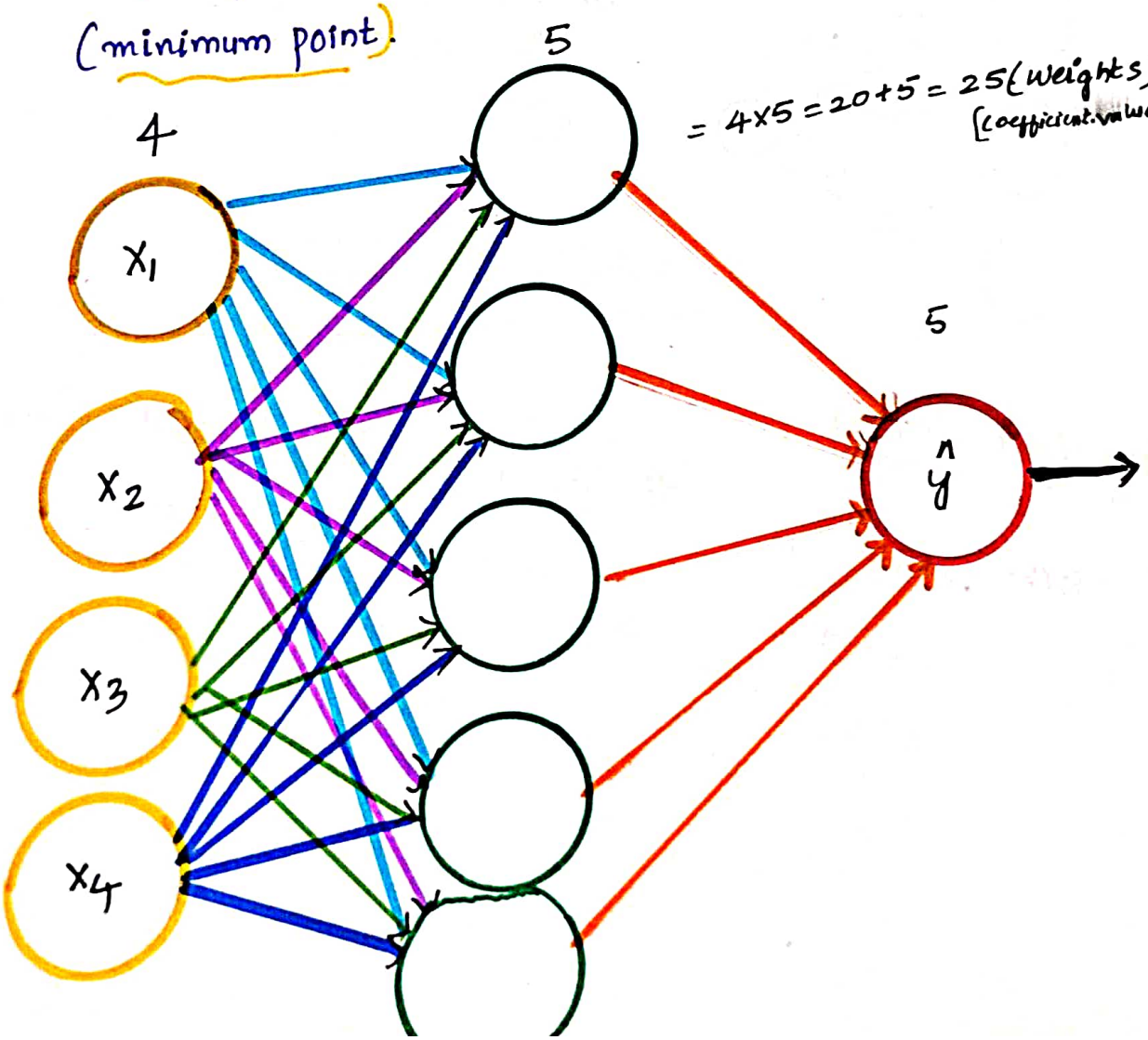
∴ **He_int normal** = $N\left[0, \sqrt{\frac{2}{int}}\right]$

# Gradient Descent




# What is Gradient descent ?

# Gradually decreasing and identifying the Local minima (minimum point).



In Machine learning :

Ex:- In multiple linear regression

model.intercept_ , model.coef_

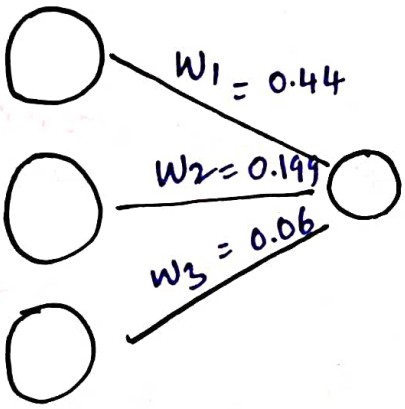
out : (2.708), array (0.44, 0.199, 0.006)

$\beta_0$ $\beta_1$ $\beta_2$ $\beta_3$

$\hat{y} = \underset{(b)}{\beta_0} + \underset{(w_1)}{\beta_1 x_1} + \underset{(w_2)}{\beta_2 x_2} + \underset{(w_3)}{\beta_3 x_3}$

$= 2.7 + 0.44[x_1] + 0.199[x_2] + 0.06[x_3]$

In deep learning :




# By Using this Technique, it gives "accurate results".

# Gradient descent

Ex:- $2x + 5 = 8$

$x = 100$

$x = 99$

⋮

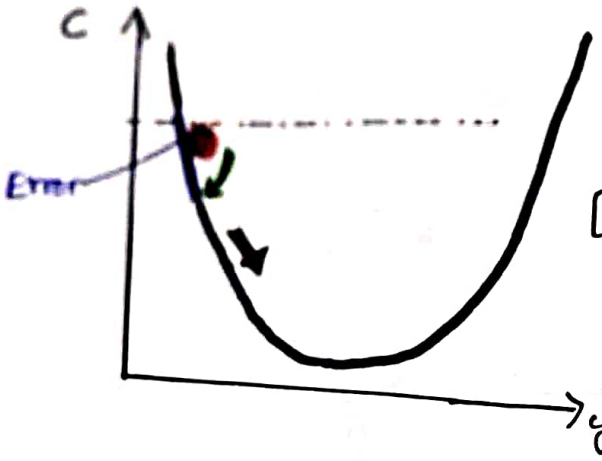
$x = 0$

$x = 1$

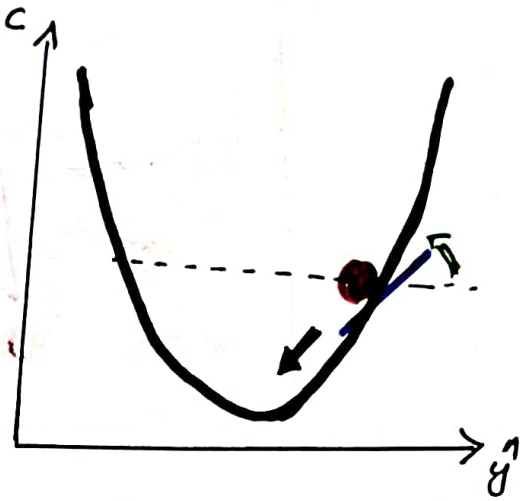
$x = 1.5$

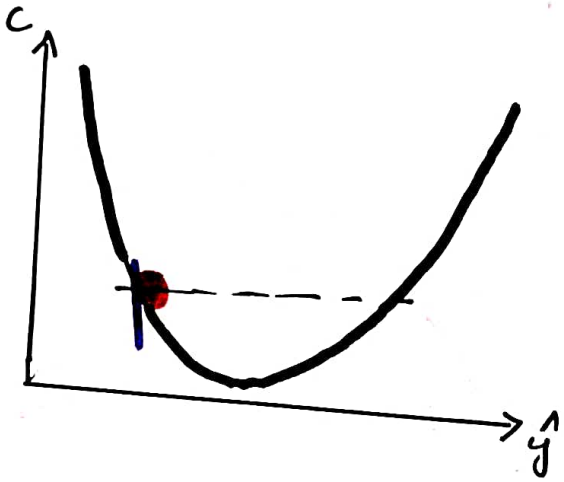
# Gradually reducing weights. and making Error as minimum value.

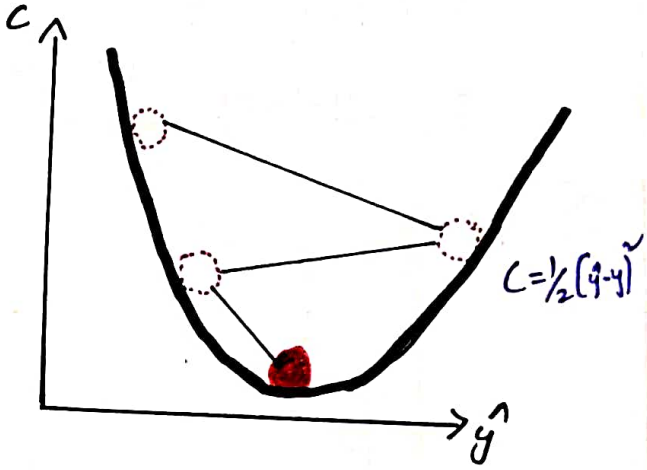
* By applying different "x" values and identfing the "perfect "x" value till, we get error as "0" ⇒ geradient decent




$[C = \frac{1}{2}[\hat{y} - y)^2$



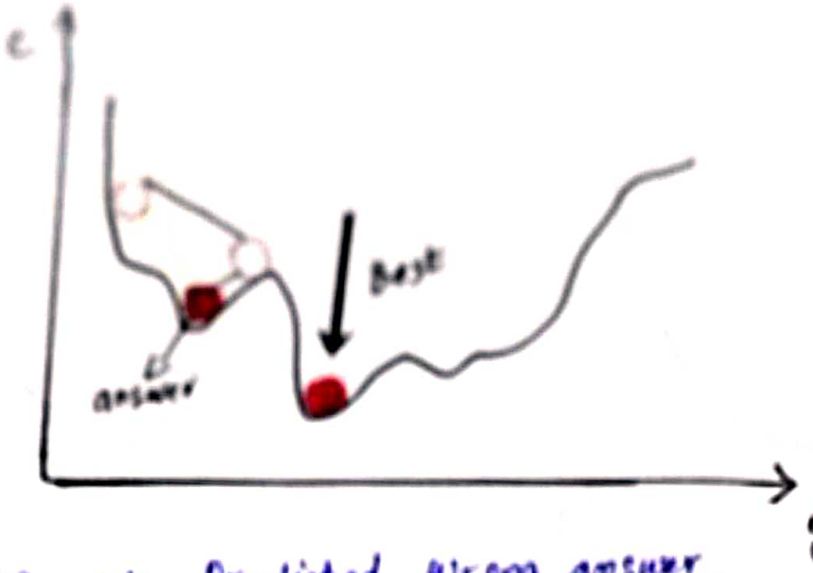




## Stochastic Gradient Descent

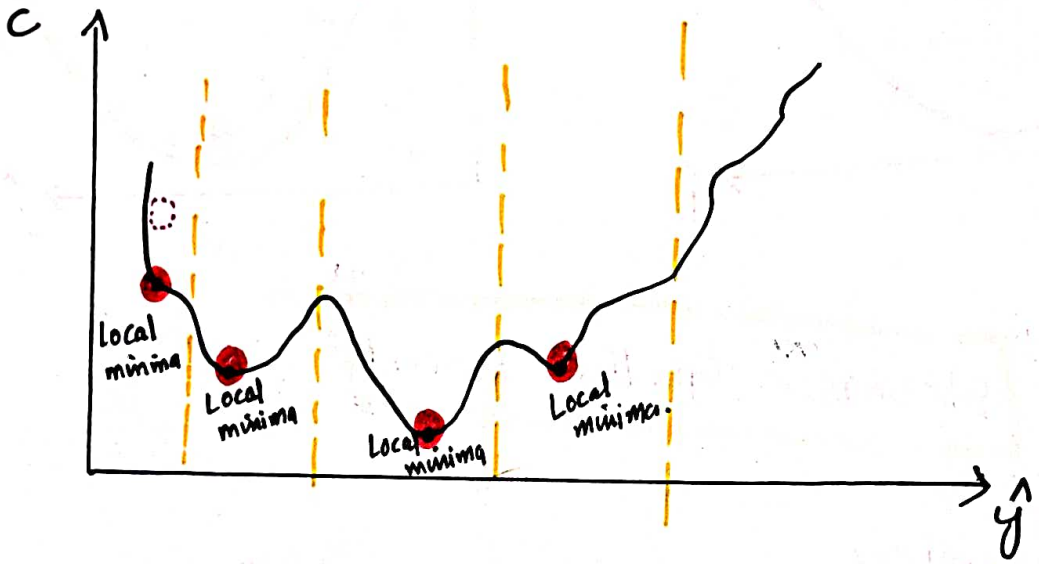
* Local minima
* Global minima

EX:- • Every state 1st ranker (local minima)

• All India ranker 1st ranker (global minima)




# Here, we predicted wrong answer,
But, there is another curve. So, How we find that?
by dividing into parts.




# identifing the Global minima is called
"Stochastic Gradient Descent"
(or)
# Entire data we divide into parts (or) batches. Each part
We are identifying the "Local minima" and from
that overall identify "Global minima".
↓
"Error".

2nd 09/05/22 10:20pm

DL: 10/5/20 10:00 am

# ANN - Classification

Code :- on Tensorflow, Keras.

Tensorflow, CNTK, theano

Google | Micro soft | FaceBook

```
# PIP install tensorflow
# PIP install keras
```

```
import numpy as np
import pandas as pd
import tensorflow as tf
import keras
# from tensorflow import keras.
# df = pd.read_csv("churn_modelling.csv")
# df.head()
```

out:

| Row no. | customer id | Surname | Credit Score | Geo graphy | Gender | Age | Tenure | Bal ance | num of Products | Has card | Is active numb | Estimated Salary |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 15634602 | Hargrave | 619 | France | Female | 42 | 2 | 0.00 | 1 | 1 | 1 | 101348 |
| 2 | 15647311 | Hill | 608 | Spain | Female | 41 | 1 | 83807 | 1 | 0 | 1 | 112542 |
| 3 | 15619304 | Onio | 502 | France | Female | 42 | 8 | 159660. | 3 | 1 | 0 | 113931 |
| 4 | 15701354 | Boni | 699 | Feance | Female | 39 | 1 | 0.00 | 2 | 0 | 0 | 93826. |
| 5 | 15737888 | Mitchell | 850 | Spain | Femal | 43 | 2 | 125510 | 1 | 1 | 1 | 79084. |

```
# df.info()
```

out:

```
Range Index : 10000 entries , 0 to 9999
Data columns.                Nonnull columns      Dtype.
0.  Row number               10000 Non null       int64
1.  Customer Id              1000 Non null        int64
```

| | | | |
|---|---|---|---|
| 3 | Credit score | 10000 non null | int 64 |
| 2 | Surname | 10000 non null | object |
| categorical data 4 | Geography | 10000 non-null | object ] convert to numeric |
| 5 | Gender | " | |
| 6 | Age | " | int64 |
| 7 | Tenure | " | int 64 |
| 8 | Balance | " | float 64 |
| 9 | No. of Products | " | int 64 |
| 10 | has Cr Card | " | int 64 |
| 11 | Is Active member | " | int 64 |
| 12 | Estimated Salary | " | float 64 |
| 13 | Exited | " | int 64 |

# we can use drop function also [x & y]

```
# x = df.iloc[:, 3:13].values
# y = df.iloc[:, 13].values
```

: → all; 3:13 → 3 to 12; [:,13] → only 13 column

original data: 13 input — 3 drop (row no., customer id, surname) → 10 inputs; 1 output → 1 column

```
# x.shape
```

Out: (10000, 10)

```
# y.shape
```

Out: (10000,)

ordinal data (sequence) — natural

↑ Label Encoder

```
# encoding [categorical data]
from sklearn.preprocessing import
# x[:, 2]
```

Out: array ["Female", "Female", ... "male", "Female"].

```
# labelencoder = LabelEncoder()
# x[:,2] = labelencoder.fit_transform(x[:,2])
# changed to numerical.
# x[:,2]
out: array [0,0,0 .... 0,1,0]
```

---

```
# one hot encoding (nominal data = Geography (column)) # We can use get_dummies.
from sklearn.preprocessing import OneHotEncoder
***
from sklearn.compose import ColumnTransformer
                              ↳ Transforms data.

# x[:,1]
out: array([["France", "spain", "France" ... "France", "Germany"].
           dtype= object.

# ct = ColumnTransformer([["encoder", OneHotEncoder(), [1]]
                          , remainder = "passthrough")
# x = np.array(ct.fit_transform(x))
# changed to numerical
# x[:,1]
out: array([[0.0, 0.0, 0.0, .... 0.0, 1.0, 0.0].
```

## train_test

```
from sklearn.model_selection import train_test_split

# x_train, x_test, y_train, y_test = train_test_split(x, y,
                                     test_size = 0.2, random_state = 0)
```

```
# Shape.
# x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out: (8000, 12), (2000, 12), (8000,), (2000,)

↳ after applying dummies it converts into 12 columns.

## Scaling

```
from sklearn.preprocessing import StandardScaler

# sc = StandardScaler()
# x_train = sc.fit_transform(x_train)
# x_test = sc.fit_transform(x_test)
```

## MODELLING

### Initializing The ANN

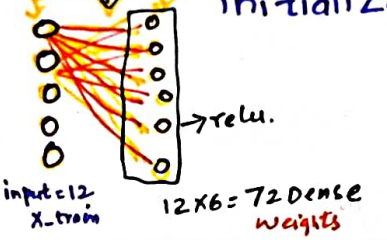
```
from keras.models import Sequential
# ann = Sequential()
```

## Adding The input layer and first hidden layer.

```
from keras.layers import Dense.

# ann.add(Dense(input_dim = 12, units = 6, kernel_
    initializer = "uniform", activation = "relu"
    ))
```

connections

Hidden Layer

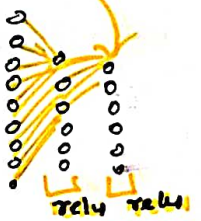



$\cup \left[\frac{-1}{\sqrt{in}}, \frac{+1}{\sqrt{in}}\right]$

$\cup \left[\frac{-1}{\sqrt{12}}, \frac{1}{\sqrt{12}}\right]$

## Adding the second hidden Layer.

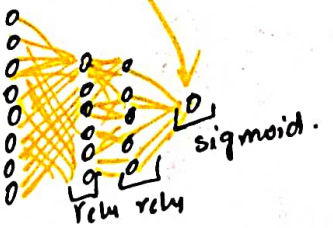
```
# ann.add(Dense(units = 6, kernel_initializer = "uniform",
    activation = "relu"))
```



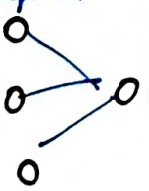

## Adding the output layer

output

```
# ann.add(Dense(units = 1, kernel_initializer = "uniform",
    activation = "sigmoid"))
```




Ex:- Perceptron model.



```
(ann.add(Dense(input_dim = 3, units = 1, kernel_initialization
    = "uniform", activation = "relu")).
```

# Part-3 Training the ANN

## Compiling the ANN

Ex: current
# past: Connection is made (wiring), (data)

$\sum(y-\hat{y})^2$
regression = "mean_squared_error"

```
# ann.compile(optimizer = "adam", loss = "binary_crossentropy",
                  Gradient descent          function
                  metrics = ["accuracy"])
```

model is created, fit the data

## training the ANN on training Set

```
# ann.fit(x_train, y_train, epochs = 100)
```

epochs → no. of iteration. (backward, forward directions)

Out

```
Epoch 1/100
250/250 [=========] - 2s 1ms/step - loss: 0.6457, - accuracy - 0.7830)
Epoch 2/100
. . . . . .
. . . . . .
. . . . . .
Epoch 100/100
```

accu: 0.838

# Part-4 Predictions & Evaluating model

## making the predictions

Predict
# Test accuracy.

```
# y_pred = ann.Predict(x_test)
# y_pred = (y_pred > 0.5)
```

## Evaluating the model

```
from sklearn.metrics import confusion_matrix, accuracy_score

# print ("Test Accuracy:", accuracy_score (y_test, y_pred))

# confusion_matrix (y_test, y_pred)
```

Out: Test accuracy : 08455

```
array([[1536, 59
        [150, 155]].
```

## cross validate the model

```
# user defined function.

def build_cross_classifier():
    classifier = Sequential()
    classifier.add(Dense(input_dim=12, units=6, kernel_
                         initializer="uniform", activation="relu
    classifier.add(Dense(units=6, kernel_initializer=
                         "uniform", activation="relu"))
    classifier.add(Dense(units=1, kernel_initializer
                         "uniform", activation="sigmo
    classifier.compile(optimizer="adam", loss="binary_
                       cross_entropy, metrics=["Acc

    return classifier
```

ann (or) anything we can use

input layer

hidden layer

output layer

connecting

## Classifier

```
from keras.wrappers.scikit_learn import KerasClassifier

# classifier = KerasClassifier(build_fn = build_cross_classifier
                               , batch_size = 10 , epochs = 100)
```

build_fn = User defined function

batch_size = if stochastic descent

epochs = iterations

## Cross_validation

```
from sklearn.model_selection import cross_val_score

# accuracies = cross_val_score(estimator = classifier ,
                               X = x_train , y = y_train, cv=5)
```

out :

```
Epoch 1/100
640/640 [======] - 1s  605us/step - loss: 0.4922, -accr: 0.7997
Epoch 2/100
640/640 [======] - 0s  617us/step - loss: 0.4296, -acc, 0.8003
. . . . . .
. . . . . .
. . . . . .
. . . . . .
Epoch 101/100
640/640 [======] 1s 805/step - loss: 0.4003. acc: 0.8342]
160/160 [======] 0s 666us/step - loss: 0.4110 . acc: 0.8375 ]
```

```
# print(accuracies)
# accuracies.mean()
[0.8237 . 0.8343 , 0.8362 , 0.8362, 0.8374)
```

out

0.83362

⇒ Hyperparameter Tunning

## Part-5 - Improving and Tuning the ANN

This can done by 3 options

1. Hyperparamter tuning
2. Regularization (L1 & L2) to reduce overfitting if needed (for regression problems)
3. Dropout

Hyperparameter tuning Can be done for identifying no. of hidden Layers & no. of neurons in each hidden Layer.

- Layers = [[20],[10],[30],[40]] → 1 hidden Layer with different options of no. of neurons in hidden Layer
- Layers = [[20],[40,20],[45,30,15]] → Multiple hidden Layers with different options of no. of neurons.

Hyperparameter tuning Can be done for identifying best activation function for hidden Layers.

- activations = ["relu", "sigmoid"]

Hyperpameter tuning Can be done for identify best optimizers

- Optimizer = ["adam", "rmsprop"]
  - "adam" → gradient desent
  - "rmsprop" → Stochatic gradient desent.

Hyper parameter tunning for batchsize for building/training model.

- batch size = [10, 16, 32, 64, 128, 256]

Code: tunning :-

```
def build_classifier(optimizer):
    classifier = Sequential()
    classifier.add(Dense(input_dim=12, units=6, kernel_
                         initializer="uniform",
    classifier.add(Dense(units=6, kernel_initializer="uniform
                         ,activation="relu"))
    classifier.add(Dense(units=1, kernel_initializer=
                         "uniform", activation="sigmoid"))
    classifier.compile(optimizer=optimizer, loss="binary_
                       crossentropy", metrics=["accuracy"])
    return classifier

# Classifier = KerasClassifier(build_fn=build_classifier)
# parameters = {"batch_size": [10,32], "epochs": [50,100],
               "optimizer": ["adam", "rmsprop"]}
```

```
from sklearn.model_selection import GridSearchCV

# grid = GridSearchCV(estimator = classifier, param_grid =
                      parameters, scoring = "accuracy", cv= 5)

# grid_result = grid.fit(x_train, y_train)
```

Out: Epoch 1/100
200/200 [==========] - 0s 711us/step - loss: 0.5877, accu. 0.7117]
Epoch 2/100
200/200 [==========] - 0s 630us/step - loss: 0.4392 - acc -0.8003)

.....

Epoch 100/100
200/200 [==========] - 0s 83 0us/step - loss: 0.6392. acc-84]

```
# grid_result.best_params_
```

Out: {"batch_size": 32, "epochs": 50, "optimizer": "adam"}

# best accuracy

```
# grid_result.best_score_
```

Out: 0.844125

```
# y_pred = grid_result.predict(x_test)
# y_pred = (y_pred > 0.5)
```

# accuracy, confusion_matrix

```
from sklearn.metrics import confusion_matrix, accuracy_
                            score.

# print ("test accuracy:", accuracy_score (y_test, y_pred))
# confusion_matrix (y_test, y_pred)
```

Out: Test accuracy : 0.8405

```
array([[1530,  65]
       [ 254,  151]],
```

10/05/22

5:30 pm.

Dt:- 21/12/22
5:50pm

# * Data Preprocessing / Data Preparation

⇒ Feature Engineering

# Data cleaning
- Missing Values
- Outliers

+

# Data Wrangling
↳ (Data Transformation)
- Encoding
- Scaling
- Transformation.

---

Us.
(Car Insurance)

# Data cleaning (In spyder)

```
# import numpy as np
# import pandas as pd
```

Why two Dataset
Because to check old(
and new dataset.

⇒ Load Data set

```
df_raw = pd.read_csv("claimants.csv")
# df = pd.read_csv("claimants.csv")
```

Out:- df_raw | dataFrame | 1340,7 | column name : CASENUM,
df | dataframe | 1340,7 | column names : "

⇒ Check is there null values.

```
# df.isnull().sum()
```

Out :-

| | |
|---|---|
| CASENUM | 0 |
| CLMSEX | 12 |
| CLMINSUR | 41 |
| SEATBELT | 48 |
| CLMAGE | 189 |
| LOSS | 0 |
| ATTORNEY | 0 |

MISSING Values.

# Data Understanding

Case Num :- Number of case (insurance)

Discrete clm sex :- 0, 1 [Male] [Female]

Discrete clm insur :- 1(Yes), 0 [NO] # having Insurance or not

Discrete Seat belt :- 0 [NO], 1 [Yes] # wearing seat belt

Continuoy clm age :- Age of the person

Loss :- Loss / Damage happened

attorney :- Whether they hire a lawyer (or) not, [0] No, [1] Yes.

Person who is Claming Isurance is called "claimants" During the accident

# There are various methods for Filling null values

* For Continous Data
  - mean (no outliers)
  - median (outliers)
* For Discrete Data → Mode

Because Mean is Heavily Impacted with Outliers [Less affected]

# Replacing null values For discrete Variables

So, we have to Replace Them with "MODE"

```
# df["CLMSEX"].value_counts()
```

(MODE)

Out :- 1.0 742 → Highest, so we Replace with "1"
0.0 586

```
# df["CLMSEX"].fillna(1, inplace=True)
```

↓ changing in orginal DaTa = "TRUE"

Out Filled with "1"

```
# df["CLMINSUR"].value_counts()
```

Out :- 1.0 1220 → MODE
0.0 120

```
# df["CLMINSUR"].fillna(1, inplace=True)
```

Out Filled with "1", so, replaced with "1" value

# df["SEATBELT"].valuecounts()

Out :- 0.0 1270 → Mode.
1.0 22

# df["SEATBELT"].fillna(0, inplace = True)

Out :- Filled with "0"

Time 11:30pm

# Replacing null Values For Continous Variables By #Pandas

# df["CLMAGE"].mean()

Out 28.414

# df["CLMAGE"].median()

Out 30.00

# df["CLMAGE"].fillna(28.414, inplace True)

Out Filled with 28.414

If we ask again what is median value.

# df["CLMAGE"].median()

Out 28.414 , so it changes 30.00 To 28.414
Because we Filled with null values 28.414.

2nd method

# Sklearn (Scikit Learn) (One stop shop For ML)

## * SimpleImputer

# Load data set again

Liberary → (module) (Function)

```
From Sklearn.impute import SimpleImputer
```

Ex: missing values = "???"

Calculating mean value

```
# mean_imputer = SimpleImputer (strategy = "mean") → E/n
```

(Function) → (Argument)

name as

Function. (or) Store in mean_imputer

Ex:- Core Python [Keyword argument]

If we don't write this line it gives

Same : a = 4

# Converting array To data Frame

To Remove array. From mean value.

```
# df ["CLMAGE"] = Pd.DataFrame (mean_imputer.fit_trans
```

storing

```
_Trans Form (df [["CLMAGE"]]))
```

```
array ([50.],
       [18.],
       [5.],
       ⋮
```

Fill in This column (or) Replacing

Calculate The mean value

Out

Filled with 28.414

For Median only change mean To median

```
# median_imputer = SimpleImputer (strategy = "Median")
```

```
# df["CLMAGE"] = pd.DataFrame(median_imputer.fit_
                Transform(df[["CLMAGE"]])
```

For Mode [Discrete Data]

```
mode_imputer = SimpleImputer(strategy="mode")

#df["CLMSEX"] = pd.DataFrame(mode_imputer.fit_transform(df[[
                                    "CLMSEX"]])

# df["CLMINSUR"] = pd.DataFrame(mode_imputer.fit_transfo
                        (df[["CLMINSUR"]])

#df("SEATBELT"] = pd.Data Frame( mode_imputer.fit_trans
                        m(df[["SEATBELT"]])
```

# Output — Fills Directly The mode Value in to the dataset.

# MODE = it is for Discrete DaTa.

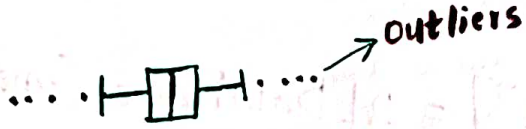
# MEAN
# MEDIAN
} → Continous DaTa

3/4/22 2:08 AM

# Outliers → # Applicable Only For Continous Data

1Que :- What is outlier ?

Ans :- An Outlier is a data point in a data set. that is distant From all other observation which is significantly different From The Remaing Data




(or)

- The data point that lies outside the over all Distribution of the dataset.

2 Que :- What are Impacts having Outliers in a DataSet ?

Ans :- it will cause Various Problems in Statistical Analysis. (it may cause a significant impact on mean and Standard deviation)

Ex :- $\bar{x}$ = mean (affect)

$\frac{\Sigma[x-\bar{x}]}{n-1}$ = varience → $\sqrt{\frac{\Sigma(x-\bar{x})^2}{n-1}}$ (Standard deviation)

all are Effected by Outliers

* In addition, Some Machine learning models are Sensitive to outliers, which may Decrease There Performance.

3 Que : Reasons For Outliers ?

Ans :
1. Data Entry Errors (Ex:- Salary = 1,00,000, But 10,000 Entering)
2. Measurement Errors (Ex :- Measuring in meters instead of KM)
3. Instrumental Error

4 Que :- Types of Outliers ?

Ans :- Univariate Outlier ----> Identifying Outlier For Single Variable

* Bivariate Outlier ----> Identifying as Outlier by Analyzing 2 Variables at a Time

# Importing Libraries

```
# Import numpy as np
# Import pandas as pd
# Import matplotlib as plt.
# Import Seaborn as Sns.
```

# Real Time Case study

# boston - House Price Dataset

```
# boston = pd.read_csv("D:\\datascience\\shivaramakrishna class\\6. Data cleaning\\boston.csv")
```

```
boston.head()
```

out

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRTIO | B | LSTAT | PRICE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.00632 | 18.0 | 2.31 | 0 | 0.538 | 6.575 | 65.2 | 4.09 | 1 | 296 | 15.3 | 396.90 | 4.98 | 24.0 |
| 1 | 0.02731 | 0.0 | 7.07 | 0 | 0.469 | 6.421 | 78.9 | 4.9671 | 2 | 242 | 17.8 | 396.90 | 9.14 | 21.6 |
| 2 | 0.02729 | 0.0 | 7.07 | 0 | 0.469 | 7.185 | 61.1 | 4.9671 | 2 | 242 | 17.8 | 392.83 | 4.03 | 34.7 |
| 3 | 0.03237 | 0.0 | 2.18 | 0 | 0.458 | 6.998 | 45.8 | 6.0622 | 3 | 222 | 18.7 | 394.63 | 2.94 | 33.4 |
| 4 | 0.06905 | 0.0 | 2.18 | 0 | 0.458 | 7.147 | 54.2 | 6.0622 | 3 | 222 | 18.7 | 396.90 | 5.33 | 36.2 |

```
# boston.info()
```

506 rows, 14 columns

out

| | | |
|---|---|---|
| 0 | CRIM | Float 64 |
| 1 | ZN | Float 64 |
| 2 | INDUS | Float 64 |
| 3 | CHAS | → int 64 |
| 4 | NOX | Float 64 |
| 5 | RM | Float 64 |
| 6 | AGE | Float 64 |
| 7 | DIS | |
| 8 | RAD | → Int 64 |
| 9 | TAX | → Int 64 |
| 10 | PTRTIO | Float 64 |
| 11 | B | Float 64 |
| 12 | LSTAT | Float 64 |
| 13 | PRICE | Float 64 |

(Float 64 (11), int 64 (3))

# Various ways of finding The Outlier

Changing Every Value to Zscore.

Ex:-

| X | Zscore |
|---|---|
| 6 | $\frac{6-5.6}{1}$ |
| 4 | $\frac{4-5.6}{1}$ |
| 9 | $\frac{9-5.6}{1}$ |
| 7 | $\frac{7-5.6}{1}$ |
| 2 | $\frac{2-5.6}{1}$ |

$Z = \frac{X-\mu}{\sigma}$

$\therefore \mu = \frac{6+4+9+7+2}{5}$

$\mu = 28/5 = 5.6$

Ex:- $\sigma = 1$

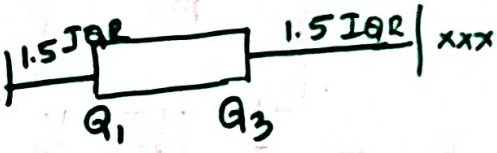
Option (1) **Z score**

$|Z\text{score}| > 2$

$[\because Z < -2, \; Z > +2]$ |modulus|

Option (2) **IQR** (Inter Quartile Range)

- any Value $< Q_1 - 1.5\,IQR$
- any value $> Q_3 + 1.5\,IQR$




# any value Less than $Q_1 - 1.5\,IQR$ is Outlier.

# any value More Than $Q_3 - 1.5\,IQR$ is Outlier

Option (3) **visualization**

* Box plot, Histogram For Univarte Outliers
* Scatter plot For bivariate Outliers.

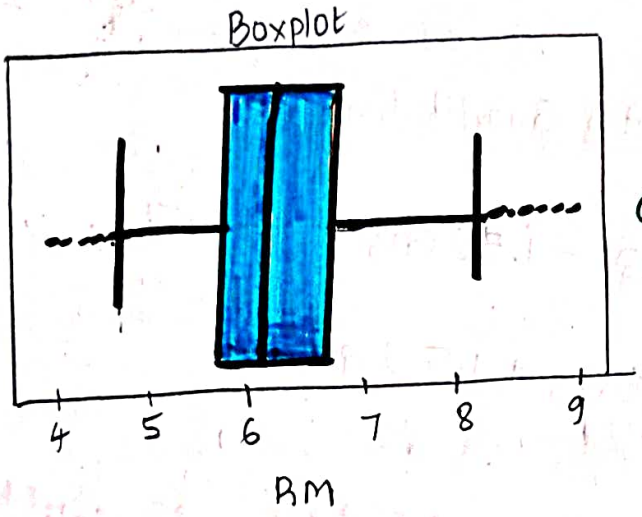
Option ③

Detection of Outliers of "RM" column (Based on Boxplot)

```
# sns.boxplot(boston.RM)
  plt.title("Boxplot")
  plt.show()
```

* Over all view of Outlier

Out :




Outliers in Both Tails of "RM".

Option ①

Detection of Outliers of "RM" column (Based on Zscore)

* For Exact Outliers in Data

$$Z\text{score} = \frac{X - \mu}{\sigma}$$

```
# boston["RM"].mean(), boston["RM"].std()
```

Out (6.284, 0.702)

```
# boston["RM_Zscore"] = (boston["RM"] - boston["RM"].mean()) / boston["RM"].std()
```

creating new column and storing Zscores

X → boston["RM"]; μ → boston["RM"].mean(); σ → boston["RM"].std()

Output :-

Clarity :-

$$\frac{\text{boston["RM"]} - \text{boston["RM"].mean()}}{\text{boston["RM"].std()}}$$

(x − μ) / σ

E Answers Ammi

boston["RM_Zscore"] = ↑ Endulo store ayethaye.

Zscore

original

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B | ... | RM SCORE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.0632 | 18.0 | 2.31 | . | . | 6.575 | . | . | . | . | . | . | | 0.413263 |
| 1 | 0.0273 | 0.0 | 7.07 | . | . | 6.421 | . | . | . | . | . | . | | 0.194082 |
| 2 | 0.0272 | 0.0 | 2.18 | . | . | 7.185 | . | . | . | . | . | . | | 1.281446 |
| ... | | | | | | | | | | | | | | |
| 505 | 0.0474 | 0.0 | 11.93 | . | . | 6.030 | . | . | . | . | . | . | | -0.362408 |

506 x 15 columns

Now, Identify Outliers in this column ↑

# Outlier = boston[(boston["RM_Zscore"] > +2) OR (boston["RM_Zscore"] < -2)]

2 conditions

Greater Than +2

Less Than -2

Out :- Outlier

| | CRIM | ... | RM SCORE |
|---|---|---|---|
| 97 | | | 2.539 |
| 98 | | | 2.185 |
| 162 | | | 3.47 |
| 163 | | | -2.127 |
| 66 | | | -3.055 |

3/4/22 4:30 Am

Time
1:00pm

# Outlier. Shape

Out :: (31, 15)

option (2)
# Detection of Outlier of RM ( based on IQR)

- Calculate First ($q_1$) and third quartile ($q_3$)
- Find interquartile range ($q_3 - q_1$)
- Find Lower bound $q_1 1.5$ & Find upper bond $q_3 1.5$

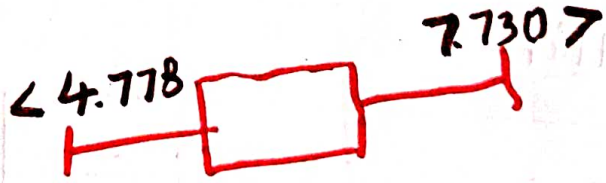
for Exact no. of range + - of outliers

```
# Q3 = boston["RM"].quantile(0.75)
  Q1 = boston["RM"].quantile(0.25)

  IQR = Q3 - Q1
  Lower_limit = Q1 - (IQR * 1.5)
  Upper_limit = Q3 + (IQR * 1.5)

  Lower_limit, upper_Limit
```

$Q_1 = 0.25$ (ol)
$Q_3 = 0.75$ (ol)

$IQR = Q_3 - Q_1$

$\Rightarrow Q_3 + IQR * 1.5$
$\Rightarrow Q_1 - IQR * 1.5$




Out :- 4.778 , 7.730
Min Max

* Which is having less than 4.778 is outlier
* which is having Greater than 7.730 is Outlier

## Methods of deal The Outliers

3R Techinque
* Remove → Deleting The outliers
* Replace → changing values / Data Manipulation
* Retain → Treat Them Seperately.
Work without outlier / with outlier "Work" seperately.

Outliers should be detected and "REMOVED" Only From Training data set, NOT from The Test set. So we Should first divide our data set into train and tests. and remove Outliers in the train set, but keep those in test set, and Measure how well our model is doing.

option (1) Removing * (Let's trimm The dataset) → 4.77

```
# boston_trimmed = boston[(boston["RM"] > Lower_limit) &
                          (boston["RM"] < upper_limit)]
                                          → 7.73
```

```
boston_trimmed
```

Out:

Which ever Data is Greater than 4.77 and Less Than 7.73 Consider That data Only. Remove Remaining data.

| | | RM | | RM_Zsco |
|---|---|---|---|---|
| 0 | | 6.575 | | 0.413263 |
| 1 | | 6.421 | | 0.194082 |
| 2 | | 7.185 | | 1.28 14416 |
| ... | | ... | ... | ... |
| 504 | | 6.794 | | 0.724955 |
| 505 | | 6.030 | | -0.362408 |

476 x 15 columns

∴ From 506 records, Removed 30 records

That Means 476 Having (or) there.

⇒ it's about 6% of the data is Removed

⇒ it is only in One Variabe (1 column) = 6%.

⇒ If Total Variable (1,2,3,4..... all columns (14) → [10] continous Data) → Remove → May Delete 100 rows ↓ 20% of Data

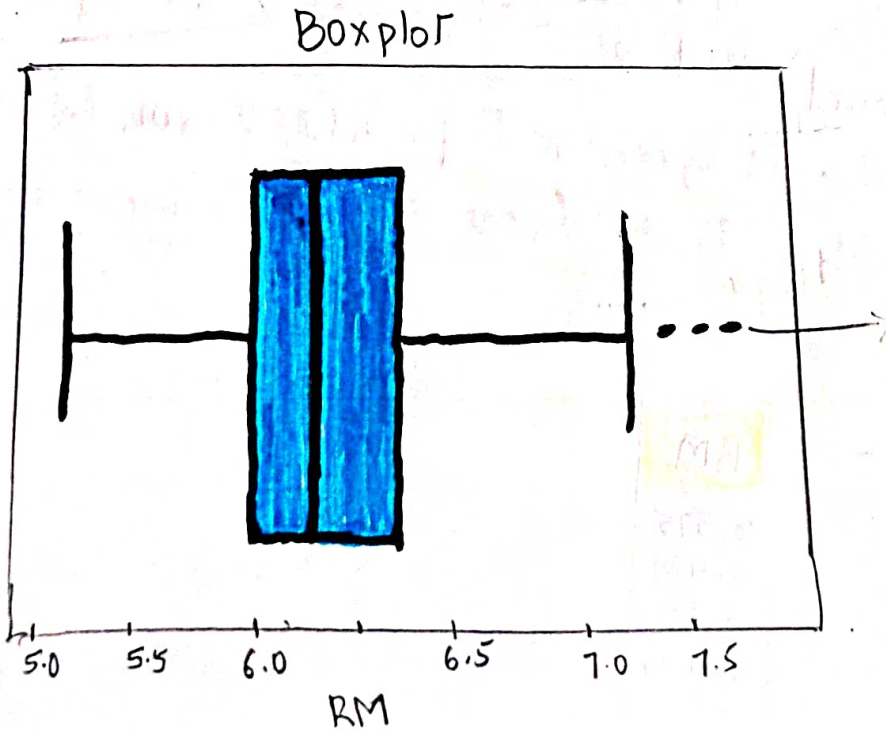
So, In real Life, we don't use outliers "Remove"

Even, Though, If we want Remove, still it contains Outliers.

⇒ Outliers in The Trimmed dataset

```
# sns.boxplot (boston_trimmed . RM)
plt. title ("Boxplot")
plt. show ().
```

Out




still, we have Outliers, after Removing The Outlier records.

Q:- Why ? it showing Outliers after Removing records of outliers

A:- When we remove datapoints From Our dataset, all the Parameters of The distribution are re-calculated.

For 476 records,

Again, it's calculating ($Q_1$, $Q_3$, IQR)

for 506

$Q_1 = 4.77$, $Q_1$ = New Value (again)

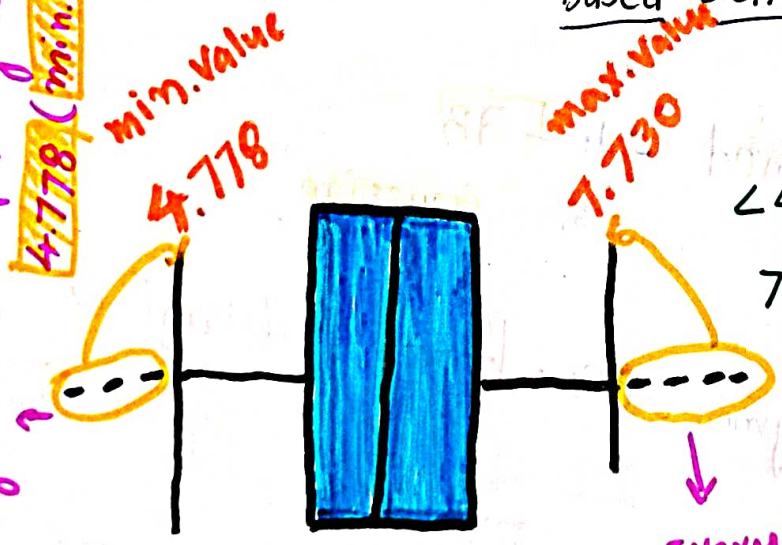
$Q_3 = 7.730$, $Q_3$ = New value (again)

* Therefore, In The new-trimmed variables values that before were not considered Outliers.

**That's why, it showing new Outliers after removing some records also.**

option (2). **Replace**

# Replace with Upper Limit & Lower Limit (Calculated Based IQR)




$< 4.778 = Q_1 - [IQR * 1.5]$

$> 7.730 = Q_3 + [IQR * 1.5]$

Every Outiler Replacing with 7.730 (max.value)

⇒ More DeTailness

Replacing with $Q_1$, $Q_3$ Values

| | CRIM | ..... | RM | ..... RAD |
|---|---|---|---|---|
| 97 | | More Than 7.730, so, They are outliers. Reduce with $Q_3$ = 7.730 | 8.069 7.730 | |
| 98 | | | 7.820 7.730 | |
| 162 | | | 7.802 7.730 | |
| 163 | | | 8.375 7.730 | |

$Q_3$ - IQR * 1.5 < [4.778] (Lower_value) → less Than 4.778 = outlier

$Q_1$ IQR * 1.5 > [7.730] (Upper_value) → greater Than 7.730 = outlier

* Winsorizer

# PIP install feature_engine

Winsorzier

# From feature_engine.Outliers (Module) Import Winsozier Function

(Library)

In Winsozier Function ( capping method argument )

* capping method = 1. IQR
2. "Gaussian"

# In feature_engine.Outliers Import ArbitaryOutlierCapper Function

(Library) (module)

We can give our own Lower, upper Values.

# 1. By IQR method

In Depth of IQR.

**Firstly,**

**# We Calculate $Q_1$ Value**

```
# Q1 = boston["RM"].quantile(0.25)
```

Out :- 5.855

**# We calculate $Q_3$ value.**

```
# Q3 = boston["RM"].quantile(0.75)
```

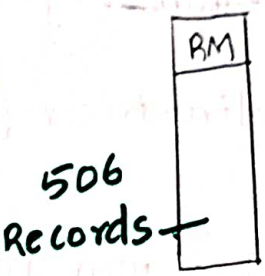
Out :- 6.623

**# IQR Value**

```
# IQR = Q3 - Q1
```

IQR

Out 0.737



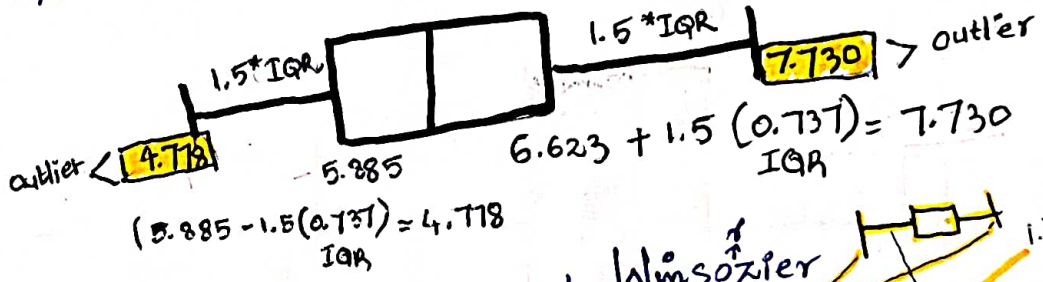

$\therefore Q_1 = 5.885$

$Q_3 = 6.623$

$IQR = Q_3 - Q_1$

$= 0.737$

Distance between $Q_3$ and $Q_1$




CODE :-

```
# From feature_engine.Outliers Import Winsozier
# Win = Winsozier(capping_method="iqr", tail='both', fold=1.5, Variables=["RM"])
# boston_t = Win.fit_transform(boston[["RM"]])
# Print(win.left_tail_caps_, Win.right_tail_caps_)
# Sns.boxplot(boston_t.RM)
# Plt.title("boxplot")
# Plt.show()
```

Win → Random name

store in → boston_t

boston[["RM"]] → original data

4.778 ← win.left_tail_caps_

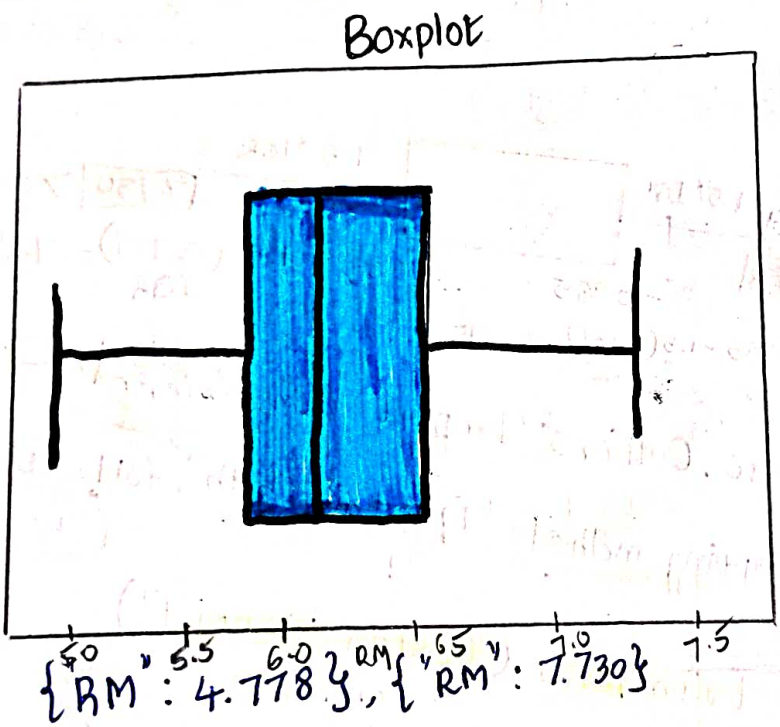
7.730 ← Win.right_tail_caps_

Edited (Replaced IQR).RM

again

```
From feature_engine.Outliers import Winsorizer

win = Winsorizer (capping_method = "iqr", fold = 1.5, Tail = "both",
                  Variables = ["RM"])
boston_t = win.fit_transform (boston [["RM"]]

Print (win.left_tail_caps_ , win.right_tail_caps_)

Sns.boxplot (boston_t.RM)
plt.title ("boxplot")
plt.show()
```

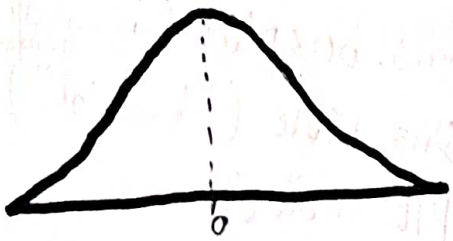
Out




2) Replacing using Winsorizer (min, max, automatically Taken by "Gaussian method")

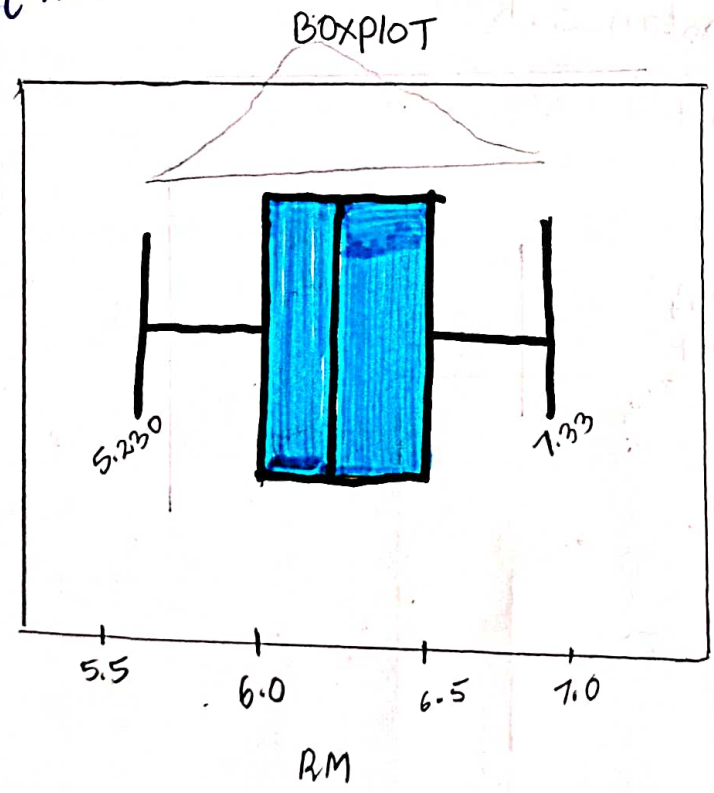
# Gaussian method = "Normal Distribution method"

same



```
# from feature_engine.Outliers import Winsorizer

# win = Winsozier (capping method = "gaussian", tail = "both",
                   fold = 1.5, variables = ['RM'])

boston_t = win.fit_transform (boston[["RM"]])

print (win.left_tail_caps_, win.right_tail_caps)

sns.boxplot (boston_t, RM)
plt.title ("Boxplot")
plt.show()
```

Out {'RM': 5.230} {"RM": 7.33}



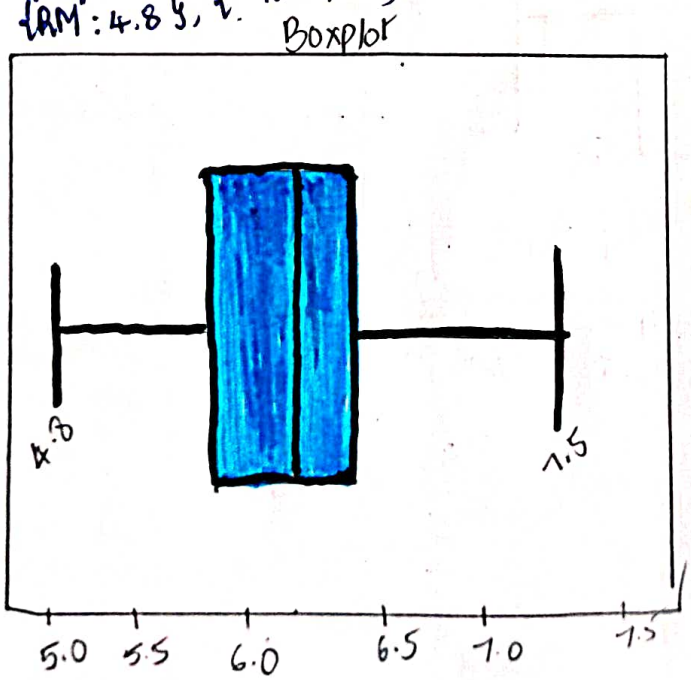
3. Replace arbitary Outlier Capper (The min, max, by user)

# we get the min and max values based on "Domain Expert"
(or) by our Own "Research"

```
# From feature_engine.Outliers Import Arbitrary Outlier Capper

Capper = ArbitraryOutlierCapper (max_capping_dict = {"RM": 7.5},
                                 min_capping_dict = {"RM": 4.8})

boston_c = Capper.fit_transform (boston[["RM"]])

Print (capper.right_tail_caps_, Capper.left_tail_caps_)

Sns.boxplot (boston_c.RM)
plt.title ("Boxplot")
plt.show()
```

Out

{'RM': 4.8}, {'RM': 7.5}




3/4/22 5:20 pm

5/4/22
10:30pm

3 * Discretization *

↓

Converting Continous Data To Discrete Data

↓

also called as "Binning"

Code :-

```
Import pandas as pd

# stroke = pd.read_csv("stroke prediction.csv")

# stroke.head()
```

Out

| id | gend | age | hyper Tension | heart_ disease | Ever_married | Work type | Residence Type | avg_ glucose Level | bmi | Smok status | Stroke |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 30669 | Male | 3.0 | 0 | 0 | No | children | Rural | 95.12 | 18.0 | NaN | 0 |
| 30468 | Male | 58.0 | 1 | 0 | Yes | Private | urban | 87.96 | 39.2 | Never | 0 |
| 16523 | Female | 8.0 | 0 | 0 | No | Private | urban | 110.89 | 17.6 | NAN | 0 |
| 56543 | Female | 70.0 | 0 | 0 | Yes | Private | Rural | 69.04 | 35.9 | Formerly smoked | 0 |
| 46136 | male | 14.0 | 0 | 0 | No | Never_ Worked | Rural | 161.28 | 19.1 | NAN | 0 |

discrete (cat) | continous | discrete (const) | discrete (count) | discrete | discrete | discrete | continous | Contin | discrete | discrete

```
# stroke.shape
```

Out : (43400, 12)

```
# stroke.info()
```

Out : information about stroke data.

## ⇒ Create Bins

# intervals = [0, 12, 19, 30, 60, 90] → #bins ⇒ Each interval

# categories = ["child", "teenager", "young adult", "middle aged", "Senior_citizen"]

#The Age column is Continous Data. So, We converting into Discrete Variable

intervals → categorised
* 0-12 → child
* 12-19 → teenager
* 19-30 → young adult
* 30-60 → Middle aged
* 60-90 → Senior citizen

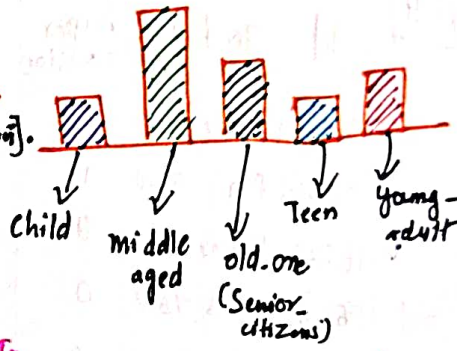
# Creating New column & Storing the data of "Age"

# Storke ["Age_category"] = Pd.cut [X = ]  (Divide.)

X = Stroke ['Age'], bins = intervals,

Labels = categories]

# Storke ["Age category"].hist()




# stroke.head()

Out

Changed Continous Data To Discrete Categorical data

Newly-added

| Id | gender | age | hyper Tension | Heart-disease | Ever married | work type | Residence Type | avg. glucose Level | bmi | Smoke | Stroke | Age_Category |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 30669 | Male | 3.0 | - | - | - | - | - | - | - | - | - | child |
| 30468 | Male | 58.0 | - | - | - | - | - | - | - | - | - | middle-aged |
| 16523 | Female | 8.0 | - | - | - | - | - | - | - | - | - | child |
| 56543 | Female | 70.0 | - | - | - | - | - | - | - | - | - | Senior_citizen |
| 46143 | Male | 14.0 | - | - | - | - | - | - | - | - | - | teen |

# 4. * Encoding *

Applicable For Discrete Categorical Variables

* Machine Can't Understand Text Data. So, We Convert the "discrete categorical" to "discrete count" Data

* There are two types of Categorical Data
  1. Nominal (No Natural Sequence)
  2. Ordinal (Natural sequence)

Ex: Grades

| | |
|---|---|
| O | 10 |
| A+ | 9 |
| A | 8 |
| B+ | 7 |
| B | 6 |
| F | 0 |

⇒ O > A+ > ... > 10 > 9

# Ordinal Data

| citynames | |
|---|---|
| Hyd | 0 |
| Delhi | 1 |
| Mumbai | 2 |
| Kerala | 3 |
| Gujarat | 4 |

In this case, we can't Decide by numbering which is greater given To state.

# we should Treat Every variable Equally.

⇒ To convert Categorical Data To Numeric:

# Nominal Data → (dummies)
1. get Dummies (Pandas)
2. One hot Encoding (sklearn)

# Odinal Data
1. map (pandas)
2. Label Encoder (sklearn)

| # One Hot Encoding | # Label Encoding |
|---|---|
| * Nominal | * Odinal |

```
import pandas as pd
import Numpy as np
import matplotlib.pyplot as plt
% matplotlib inline
```

```
# df = pd.read_csv("homeprices.csv")
# df.head()
```

Out :-

spt

| | Town | area | Price |
|---|---|---|---|
| 0 | chennai | 2600 | 5500000 |
| 1 | chennai | 3000 | 5650000 |
| 2 | chennai | 3200 | 6100000 |
| 3 | chennai | 3600 | 6800000 |
| 4 | Banglore | 2600 | 5850000 |

# Town

1. Chennai
2. Banglore
3. hyderabad

Discrete → Categorical Data

Nominal Data

⇒ **Pd.get_dummies** [Nominal Variable Encoding using pandas]

```
# dummies = pd.get_dummies(df["Town"])
# dummies
```

Out :-

| | Banglore | Chennai | Hyderabad |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 2 | 0 | 1 | 0 |
| 3 | 0 | 1 | 0 |
| 4 | 1 | 0 | 0 |
| 5 | 1 | 0 | 0 |
| 6 | 1 | 0 | 0 |
| 7 | 1 | 0 | 0 |
| 8 | 0 | 0 | 1 |
| 9 | 0 | 0 | 1 |
| 10 | 0 | 0 | 1 |

Out :-

| | Banglore | Chennai | Hyderabad |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 2 | 0 | 1 | 0 |
| 3 | 0 | 1 | 0 |
| 4 | 1 | 0 | 0 |
| 5 | 1 | 0 | 0 |
| 6 | 1 | 0 | 0 |
| 7 | 1 | 0 | 0 |
| 8 | 0 | 0 | 1 |
| 9 | 0 | 0 | 1 |
| 10 | 0 | 0 | 1 |
| 11 | 0 | 0 | 1 |

→ Dummies

# which Ever Record is there it shows = 1

* For other columns it shows = 0

Adding → original → converted one [by]

```
# df_dummies = pd.Concat([df, dummies], axis = "columns")
```

```
df_dumies
```

Out :-

| | Town | Area | Price | Banglore | Chennai | Hyderabad |
|---|---|---|---|---|---|---|
| 0 | Chennai | 2600 | 5500000 | 0 | 1 | 0 |
| 1 | Chennai | 3000 | 5650000 | 0 | 1 | 0 |
| 2 | Chennai | 3200 | 6100000 | 0 | 1 | 0 |
| 3 | Chennai | 3600 | 6800000 | 0 | 1 | 0 |
| 4 | Banglore | 3600 | 5850000 | 1 | 0 | 0 |
| 5 | Banglore | 2600 | 6150000 | 1 | 0 | 0 |
| 6 | Banglore | 2800 | 6500000 | 1 | 0 | 0 |
| 7 | Banglore | 3300 | 6100000 | 1 | 0 | 0 |
| 8 | Hyderabad | 3600 | 7100000 | 0 | 0 | 1 |
| 9 | Hyderabad | 2600 | 5750000 | 0 | 0 | 1 |
| 10 | Hyderabad | 2900 | 6000000 | 0 | 0 | 1 |
| 11 | Hyderabad | 3100 | 6200000 | 0 | 0 | 1 |

# Delete The Original column

```
# df_dummies.drop("town", axis = "columns", inplace = True)

df_dummies
```

Out:

| | Area | Price | Banglore | chennai | Hyderabad |
|---|---|---|---|---|---|
| 0 | 2600 | 5500000 | 0 | 1 | 0 |
| 1 | 3000 | 5650000 | 0 | 1 | 0 |
| 2 | 3200 | 6100000 | 0 | 1 | 0 |
| 3 | 3800 | 6800000 | 0 | 1 | 0 |
| 4 | 2600 | 5850000 | 1 | 0 | 0 |
| 5 | 2800 | 6150000 | 1 | 0 | 0 |
| 6 | 3300 | 7100000 | 1 | 0 | 0 |
| 7 | 3600 | 5750000 | 1 | 0 | 0 |
| 8 | 2600 | 6000000 | 0 | 0 | 1 |
| 9 | 2900 | 6200000 | 0 | 0 | 1 |
| 10 | 3100 | 6900000 | 0 | 0 | 1 |
| 11 | 3600 | 6500000 | 0 | 0 | 1 |

# after deleting original column in New Data (ie. df_dummies) Old_data (df) is still remains same.

## Dummy Variable Trap

⇒

| Bang | chenn | Hyder |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 0 | 1 |

→

| Bang | chenn |
|---|---|
| 0 | 1 |
| 0 | 1 |
| 1 | 0 |
| 1 | 0 |
| 0 | 0 |
| 0 | 0 |

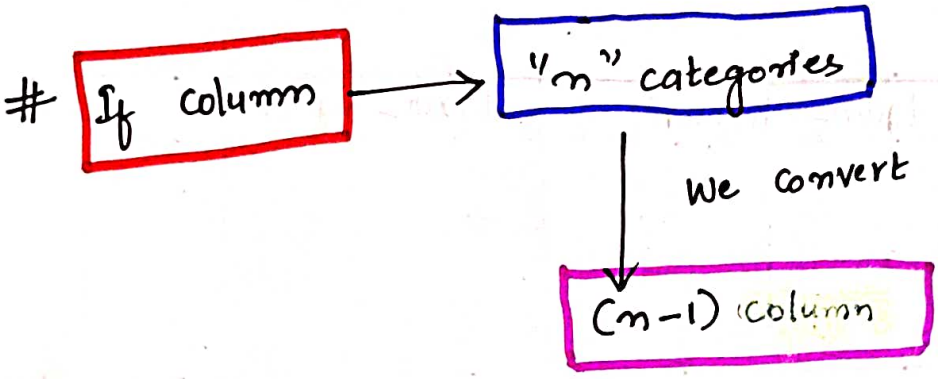
→ hyderabad

If we Remove One of The column also, still we can Identify, col name by (0,0)

# Town — 3 categories

↓ Dummies

3 columns [Multi-collinearity / collerations Problem]




```
# df_dummies.drop("chennai", axis="columns", inplace=True)

# df_dummies
```

out

| | Area | Price | Banglore | Hyderabad |
|---|---|---|---|---|
| 0 | 2600 | 5500000 | 0 | 0 |
| 1 | — | — | 0 | 0 |
| 2 | — | — | 0 | 0 |
| 3 | — | — | 0 | 0 |
| 4 | — | — | 0 | 0 |
| 5 | — | — | 1 | 0 |
| 6 | — | — | 1 | 0 |
| 7 | — | — | 1 | 0 |
| 8 | — | — | 1 | 0 |
| 9 | — | — | 0 | 1 |
| 10 | — | — | 0 | 1 |
| 11 | 36 000 | 6500000 | 0 | 1 |

(Same) as old data

# where ever Both column Having Equal Zero's, it i... another column's data.

Chennai = 0 0

# We Can Write One line of Code To do all these

```
# df_dum = pd.get_dummies (df, drop_first = True)

# df_dum
```

Out:

| | Area | Price | town_chennai | town_hyderabad |
|---|---|---|---|---|
| 0 | | | 1 | 0 |
| 1 | | | | |
| 2 | | | 1 | 0 |
| ... | | | | |
| 10 | | | 1 | 0 |
| 11 | | | | |

# This Diagram is Exact of (previous) Last page diagram. But deleted banglore Replaced chennai"

town_banglore [deleted] → Because, B ✓ (Alphabetical order)
chennai
hyd

Replaced with (0 0)

⇒ One Hot Encoding → Nominal using sklearn

```
From sklearn.Preprocessing import OneHotEncoder

# enc = OneHotEncoder (drop = "First")
```

↑ stores in enc

```
# enc_df = pd.DataFrame (enc.fit_transform
                          (df [["town"]]).
                          toarray ())
```

↑ converting

↓ output

# enc_df

Out

| | 0 | 1 |
|---|---|---|
| 0 | 1.0 | 0.0 |
| 1 | 1.0 | 0.0 |
| 2 | 1.0 | 0.0 |
| 3 | 1.0 | 0.0 |
| 4 | 0.0 | 0.0 |
| 5 | 0.0 | 0.0 |
| 6 | 0.0 | 0.0 |
| 7 | 0.0 | 0.0 |
| 8 | 0.0 | 1.0 |
| 9 | 0.0 | 1.0 |
| 10 | 0.0 | 1.0 |
| 11 | 0.0 | 1.0 |

# merge with main df

# df_ohe = df.join(enc_df)

# df_ohe.drop("town", axis="columns", inplace=True)

# df_ohe

Out

| | area | Price | 0 | 1 |
|---|---|---|---|---|
| 0 | 2600 | 5500000 | 1.0 | 0.0 |
| 1 | 3000 | 5650000 | 1.0 | 0.0 |
| 2 | - | - | 1.0 | 0.0 |
| 3 | - | - | 0.0 | 0.0 |
| 4 | - | - | 0.0 | 0.0 |
| 5 | - | - | 0.0 | 0.0 |
| 6 | - | - | 0.0 | 0.0 |
| 7 | - | - | 0.0 | 1.0 |
| 8 | - | - | 0.0 | 1.0 |
| 9 | - | - | 0.0 | 1.0 |
| 10 | - | - | 0.0 | 1.0 |
| 11 | - | - | | |

Alphabetical order

⇒ **Label Encoder** → **Ordinal Variable** [sklearn]

# Used For binary category Variable

# dfnew = df.copy() → creating "copy" From original Data.

# dfnew

Out

| | town | Area | price |
|---|---|---|---|
| 0 | Chennai | 2600 | 5500000 |
| 1 | chennai | 3000 | 5650000 |
| 2 | Chemai | 3200 | 61000000 |
| . | . | . | . |
| . | . | . | . |
| . | . | . | . |
| 11 | Hyderabad | 3600 | 6950000 |

From sklearn.preprocessing Import LabelEncoder

# Le = LabelEncoder()

# dfnew.town = Le.fit_transform(dfnew.town)

# dfnew

Out:-

| | town | Area | price |
|---|---|---|---|
| 0 | 1 | | |
| 1 | 1 | | |
| 2 | 1 | | |
| 3 | 1 | | |
| 4 | 0 | | |
| 5 | 0 | | |
| 6 | 0 | | |
| 7 | 0 | | |
| 8 | 2 | | |
| 9 | 2 | | |
| 10 | 2 | | |
| 11 | 2 | | |

Alphabetical Order

Banglore [B] — 0
Chennai [C] — 1
Hyderabad [H] — 2

↓ User Desired values we use this.

⇒ map() → Ordinal Variable [pandas]

# map function.

```
# df_m = df.copy()
# df_m ["town"] = df_m ["town"]. map ({ "Chennai :0,
                                        "Bangalore ":2 , "hyderabad": 1})
# df_m
```

# Taken by our choices

Here

Chennai = 0, 20

Banglore = 2, 15

hyderabad = 1, 25

# We can Replace with any Value

Out :-

| | town | Area | Price |
|---|---|---|---|
| 0 | 0 | | |
| 1 | 0 | | |
| 2 | 0 | | |
| 3 | 0 | | |
| 4 | 2 | | |
| 5 | 2 | | |
| 6 | 2 | | |
| 7 | 2 | | |
| 8 | 1 | | |
| 9 | 1 | | |
| 10 | 1 | | |
| 11 | 1 | | |

Which Ever Order we write it Takes in The Order ↑

⇒ Ordinal Encoder → Ordinal Variable (Ascending Order)

From Sklearn.Preprocessing import OrdinalEncoder

```
# df_new1 = df.copy()
# oe = OrdinalEncoder (categories = [["Banglore", "Hyderabad", "Chennai"]])
#                                     0           1            2
# df_new1.town = oe.fit_transform (df_new1 [["town"]])
# df_new1
```

Out :-

| | Town |
|---|---|
| 0 | 2.0 |
| 1 | 2.0 |
| 2 | 2.0 |
| 3 | 2.0 |
| 4 | 0.0 |
| 5 | 0.0 |
| 6 | 0.0 |
| 7 | 0.0 |
| 8 | 1.0 |
| 9 | 1.0 |
| 10 | 1.0 |
| 11 | 1.0 |

6/4/22 3:00AM

Dt:-3/4/22
11:30pm

"1" ⇒ Feature Transformation → Only For Continous Data

Feature processing : Transformation.

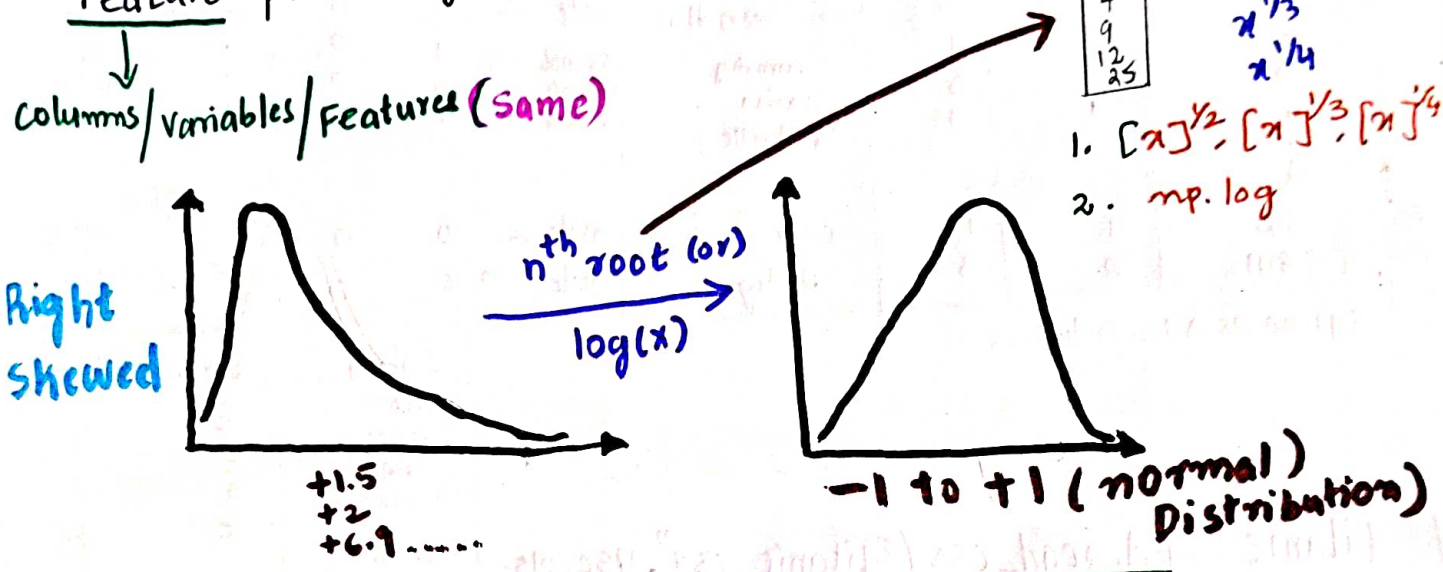
Feature ↓
Columns / Variables / Features (same)

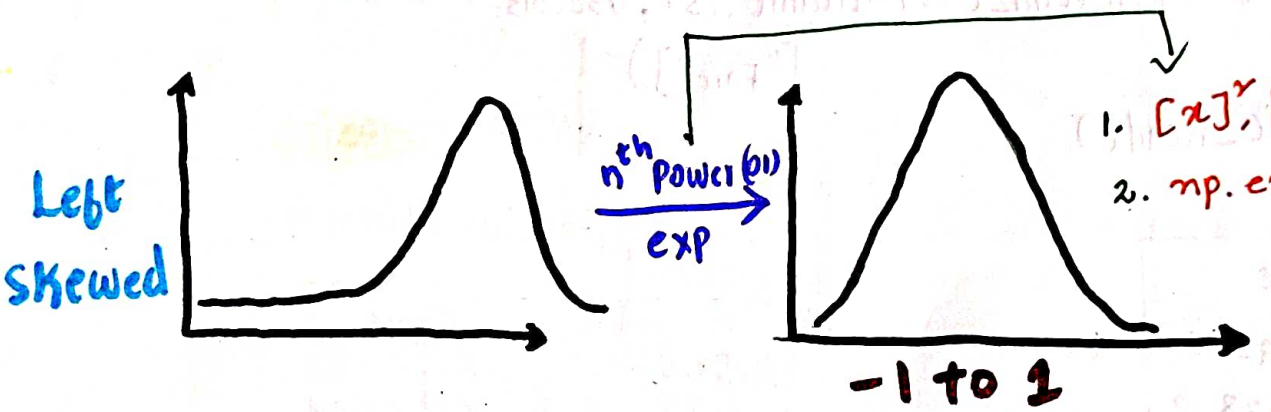
Ex:

| x |
|---|
| 2 |
| 4 |
| 9 |
| 12 |
| 25 |

$x^{-0.5}$ ✓
$x^{1/2}$
$x^{1/3}$
$x^{1/4}$

1. $[x]^{1/2}$, $[x]^{1/3}$, $[x]^{1/4}$
2. np.log




1. $[x]^{r}$, $[x]^{3}$, $[x]^{4}$.
2. np.exp

# import Liberaris / packages.

```
# import numpy as np
# Import pandas as pd
# Import matplotlib.pyplot as plt
# % matplotlib inline
# Import Seaborn as sns
```

1. Root Transformations
2. log Transformations
3. Reciprocal Transformat
4. Boxcox Transformatio

# titanic = pd.read_csv("titanic_csv")

location of csv

titanic

| | Passenger Id | Survied | Pclass | Name | Sex | Age | Sibsp | Parch | Ticket | Fare | Cabin | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 3 | Owen Harris | male | 22.0 | 1 | 0 | A/52771 | 7.2500 | NAN | S |
| 1 | 2 | 1 | 1 | cumng | Female | 38.0 | 1 | 0 | . | 71.2833 | C85 | C |
| 2 | 3 | 1 | 3 | Laina | Female | 26.0 | 0 | 0 | . | 7.9250 | NAN | S |
| 3 | 4 | 1 | 1 | Futrelle | Female | 350 | 1 | 0 | | 53.1000 | C123 | S |
| | ⋮ | | | | | | | | | | ⋮ | |
| 889 | 890 | 1 | 1 | Behr | male | 26.0 | 0 | 0 | . | 30,0000 | C148 | C |
| 890 | 891 | 0 | 3 | Dooley | male | 32.0 | 0 | 0 | | 7.7500 | | |

891 rows x 12 columns.

# titanic = pd.read_csv("titanic.csv", usecols=["Fare"])

Particular column

usecols = ["Fare"]

col. Name.

titanic.head()

Out

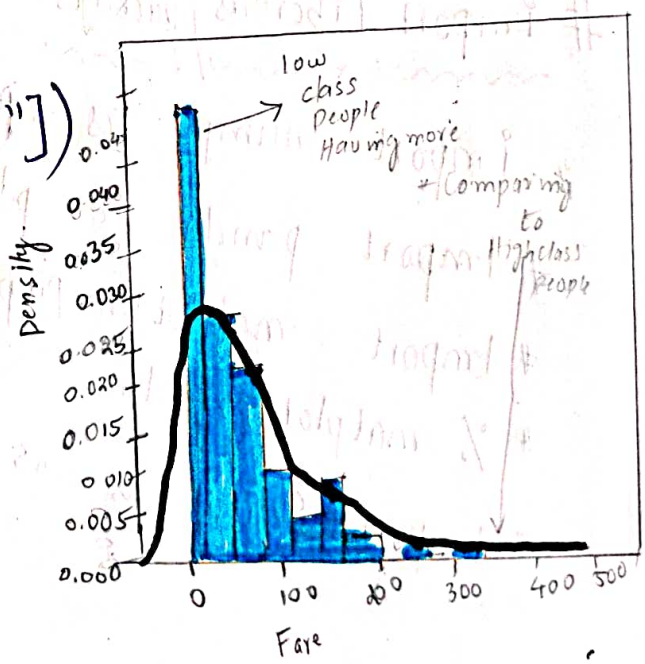
| | Fare |
|---|---|
| 0 | 7.2500 |
| 1 | 71.2833 |
| 2 | 7.9250 |
| 3 | 53.1000 |
| 4 | 8.0500 |

# sns.distplot(titanic["Fare"])

plt.show()

titanic["Fare"].skew()

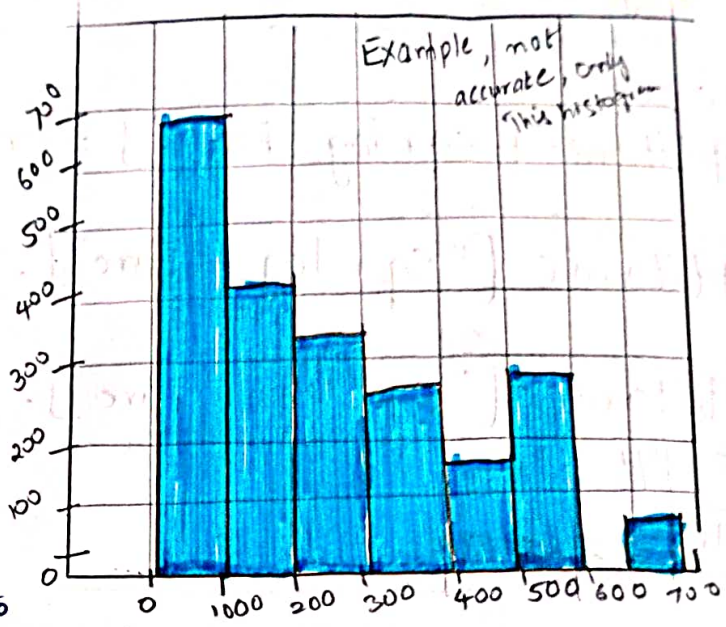
Out: + 4.7873



# another way

```
t - ["Fare"].hist()
plt.show()
titanic["Fare"].skew()
```

Out :- +4.7873




Values

Skew = +2.085 [2nd root]
Skew = +0.5 (4th root)
Skew = +0.21 (5th root) ← Consider, This which is close To "0"
Skew = -0.95 (6th root)

* Root Transformation # For Right skewed

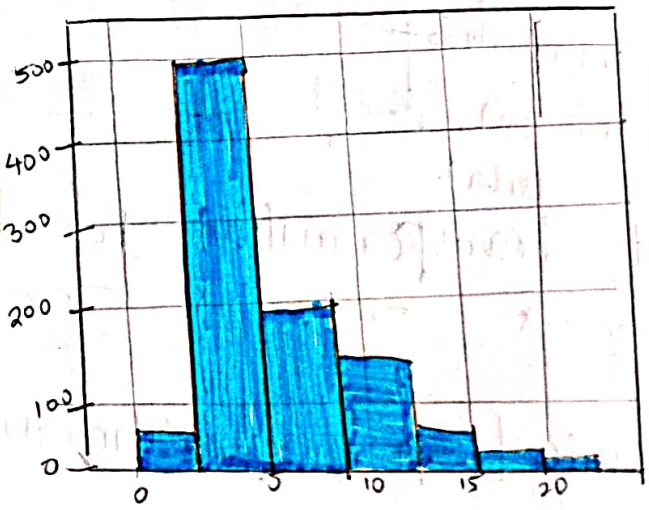
```
# titanic["sqr_Fare"] = titanic["Fare"] ** (1/2)
# titanic["sqr-Fare"].skew()
# titanic["sqr-Fare"].hist()
```

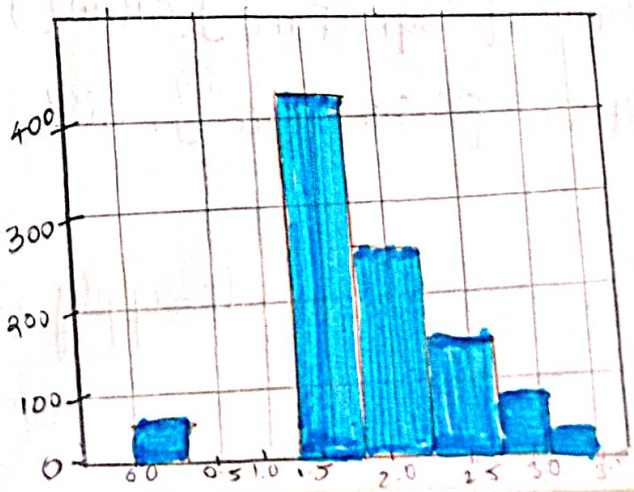
→ For Right skewed we use $n^{th}$ root $\sqrt{}, \sqrt[3]{}, \sqrt[4]{}, \sqrt[5]{}$

Out :- +2.085 More Than +1 or -1

* Try with [1/5], [1/6]...

```
# titanic["sqr_Fare"] = titanic["Fare"] ** (1/5)
# titanic["sqr_Fare"].hist()
# titanic["sqr_Fare"].skew()
```




Out :- -0.212676 Final

less Than -1 to +1

## → log Transformation

Q. Is, This mandatory, To write Every time?
- No, Only if we "0" in The Data.

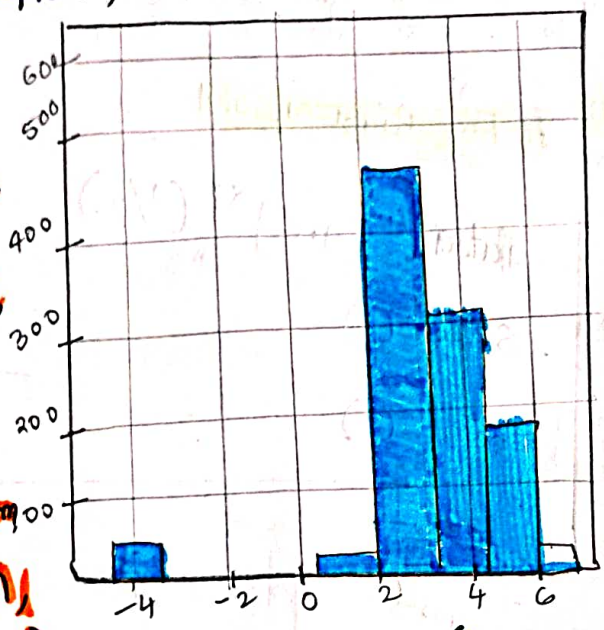
```
#titanic["sqr log_"Fare"] = np.log(titanic["Fare"]+0.01)
#titanic["sqr log_"Fare"].skew()
#titanic["sqr log_"Fare"].hist()
# plt.show()
```

Why?

Because In Data "Fare" column, we have some records "0" Values. So, If we calculate with log. it give a (infinite) Value, so, we add +0.01 to "0" column.

Out

-2.41004




Here, The Value, in This Have more, so we don't use log Transform For This col, Data)

Root Transformation For (Left Skewed) → $n^{th}$ power → $(x)^2, (x)^3, (x)^4$ → left skewed

→ $n^{th}$ root $(x)^{1/2}, (x)^{1/3}, (x)^{1/4}$ → Right skewed

we use "n" power

Fare data is not Left skewed, But To understand, Left skewed

```
titanic["sqr_Fare"] = titanic[Fare]**(2)
titanic["sqr_Fare"].skew()
titanic["sqr_Fare"].hist()
```

it can be any Value (2) (3) (4) ....

* when The Data is left skewed.

Out:

* We apply $n^{th}$ power To The Data (or) column.

* Exponential

```
titanic["sqr_exp Fare"] = np.exp(titanic["Fare"])
titanic["sqr_exp Fare"].skew()
titanic["sqr_exp Fare"].hist()
plt.show
```

→ another method For Left Skewed Distibution

* Evaluates $e^1 x$ For Each Element in The given Input.

* Reciprocal Transformation

```
titanic["Rec_Fare"] = 1 / (titanic["Fare"]+0.01)
titanic["Rec-Fare"].skew()
titanic["Rec-Fare"].hist()
plt.show()
```

"Reciprocal" is that We divide Every value with (1)/[column] +0.0 to Elimi "0" value In The Data.

of The data .

* Box Cox Transformation

| Fare |
|---|
| 7.250 |
| 71.283 |
| – |
| – |
| – |
| – |

$\rightarrow \frac{7.250^{\lambda}-1}{\lambda}$

$\rightarrow \frac{71.283^{\lambda}-1}{\lambda}$

$$T[x] = \frac{x^{\lambda}-1}{\lambda}$$

∴ $\lambda$ [lamba] Varies From -5 to +

∴ Where x is The response Variable and

⇒ In This Transformation, all Values o

"$\lambda$" are considered and The optimal Value for a given Variables are Selected.

```
from scipy import stats
# stats.boxcox (titanic["Fare"]+0.01)
```

If we don't write this value
Error: Data must be +ve
+0.01 (or) +1 (user defined)

Out :- (array([2.3844558, 6.43357655, 2.51273973, ...
2.607914, 5.7648427, 4.06763781, ...
...]),
which ever gives best result

From help
we see Two return values
1. array values (boxcox)
2. optimal Lamda parameter (max log)

```
# Fitting data, Fitting Lambda
# titanic["Fare_boxcox"], param = stats.boxcox
                                  (titanic["Fare"]+1)
```

Multiple Variable assignment
Out :- a, b = (1) (2)
stores

```
# print("λ=", param) (or) param
```

Out :- -0.09

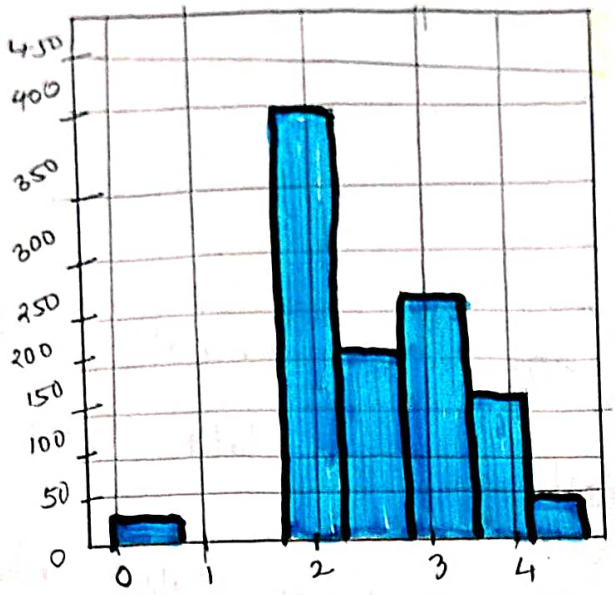
```
# titanic["Fare_boxcox"].skew()
```

Out :- -0.04 → lowest values while Compare with other Transformations

## Code

```
# titanic["Fare_boxcox], param = stats.boxcox (titanic.Fare+1
# titanic["Fare_boxcox].hist()
# titanic["Fare_boxcox].skew()
```

Out :- -0.04




4/4/22
5:30pm.

5/4/22
11:00 AM

## 2 ⇒ Feature Scaling ⟹ Applicable only For Continous data

Feature Scaling ?

Ans:- it refers (or) Techniques used to normalize the ranges of independent Variables (input Variables) in our data. (or)

* The methods to set the Features value range within a similar Scale. (same scale)

* Variables with bigger magnitude / larger value range dominate over those with smaller magnitude / value range.

Ex:- 10,00,00 , 10 (x)
If we take
10 ——— 10 → Both are Equal Import

* Scale of the features is an important consideration when building Machine Learning models

* Feature scaling is generally the last step in the data pre processing pipeline, PerFormed Just Before Training the machine Learning algorithms.

5. Gradient descent converges Faster when Features are on Similar scales

# * Feature Scaling *

Example :-

| C.C | Milage |
|---|---|
| 1600 | 14 |
| 1800 | 15 |
| 2200 | 16.5 |
| 2100 | 18 |
| 2600 | 22.5 |
| 1800 | 19.4 |
| 1900 | 25.4 |

of a car

Continous Data

Q. When we ask Machine which is important In The both columns ?

* Machine is Giving Importance To C.C Because it is Having Large Values.

* But, We know both are importa

So, we have train model in Initially such that, it should consider "Both The columns Similar

So, we Reduce The Value with 200

$= \frac{800}{1000} = \frac{80}{100} = \frac{8}{10} = \frac{4}{5}$ (Every one is Same)

with 100

We are scaling down Dividing Every Value with "10" In this case.

* To Reduce The Value.

* Feature Scaling Importance in Some ML Algorithms

1. Linear Regression & logistic Regression

* The regression coefficients of linear models are Directly influenced by Scale of The Variable.

2. Support Vector Machines :-

* Feature scaling helps Decrease The Time to Find Support Vectors For SVM's

3. K-means clustering

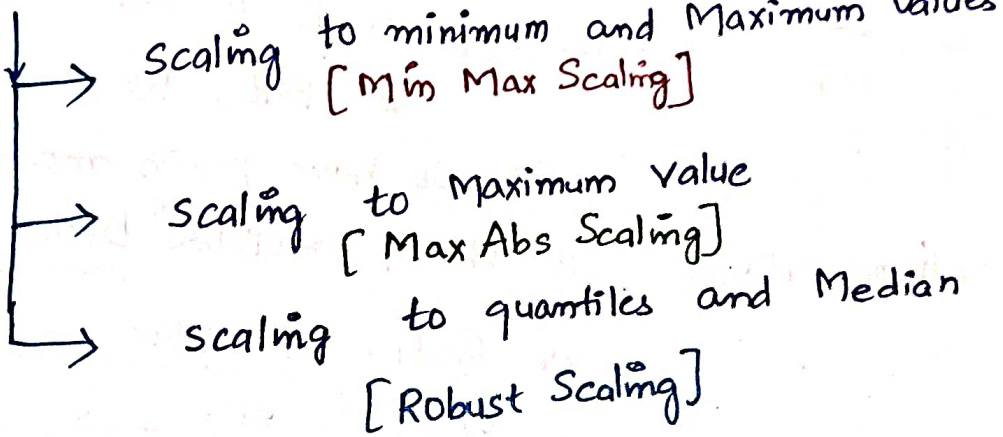
* Euclidean distances are Sensitive to Feature magnitude

4. Principal component analysis (PCA) :-

* PCA requires The Features to be Centered at "0".

* Various Feature Scaling Techniques :-

1. Standardisation
2. Normalisation Technique




```
# import pandas as pd
# import matplotlib.pyplot as plt
  %. matplotlib inline.
```

→ Using (titanic Data set)

code:

```
# titanic = pd.read_csv("titanic.csv", usecols=["Age"
  titanic.head()
```

Out

| Age |
|---|
| 22.0 |
| 38.0 |
| 26.0 |
| 35.0 |
| 35.0 |

# check Null values in "Age" column.

```
# titanic.isnull().sum()
```

Out  Age  177 → Null values

# Replacing with "Median Value" For Null values

```
# titanic["Age"].fillna(titanic["Age"].median(),
                        inplace = True)
# titanic["Age"].isnull().sum()
```

Out :  Age  0 → Zero Null values

---

I. **Standardisation**

↓

Converting Each Value To Zscore

Ex :-

| X | $\frac{x-\mu}{\sigma}$ = Zscore |
|---|---|
| 23 | $\frac{23-60}{2}$ = -18.5 |
| 45 | $\frac{45-60}{2}$ = -7.5 |
| 67 | $\frac{67-60}{2}$ = 3.5 |
| 89 | $\frac{89-60}{2}$ = 14.5 |
| 78 | $\frac{78-60}{2}$ = 9 |

These Five values converting To Zscores is called as "Standard" Scaling

Avg/mean = 60.

(Std) $\sigma = \underset{Exp}{2}$

Code :-

```
From sklearn.Preprocessing import StandardScaler (SC)

# sc = StandardScaler ()        # call The Function (Short Cut)
                  ↓ main (don't Forget)
# titanic ["Age_sc"] = sc. fit_tranform (titanic [["Age"]
```

# creating new column

# fit_transform → calculation → Converting The value → stores in

```
# titanic ["Age_sc"]
```

Out :-

| | Age_sc |
|---|---|
| 0 | -0.5657 |
| 1 | 0.6638 |
| 2 | -0.2583 |
| 3 | 0.43312 |
| ⋮ | ⋮ |
| 890 | 0.20272 |

# Here Every value is Converted To Zscore.

$$\left[\frac{X-\mu}{\sigma}\right] = Z\ score.$$

Ex :- Age [22] # First record X = 22

```
# titanic.mean()
μ = 29.361
# titanic.std ()
σ = 13.01
```

$$\left[\frac{X-\mu}{\sigma}\right] = \left[\frac{22-29.361}{13.01}\right]$$

"Age_sc" → [-0.5657]

```
# titanic
```

Out

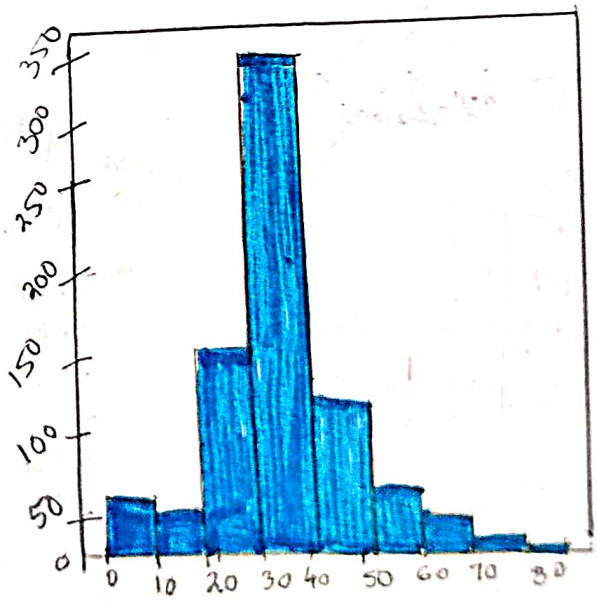
| | Age | Age_sc |
|---|---|---|
| 0 | 22.0 | -0.5657 |
| 1 | 38.0 | -0.6638 |
| 2 | 25.0 | -0.2583 |
| ⋮ | ⋮ | ⋮ |
| 889 | 26.0 | -0.2583 |
| 890 | 32.0 | 0.20272 |

Original Data

Converted To Z_Score

# histogram of Original ["Age"]

```
# titanic ["Age"]. hist()
plt. show
```

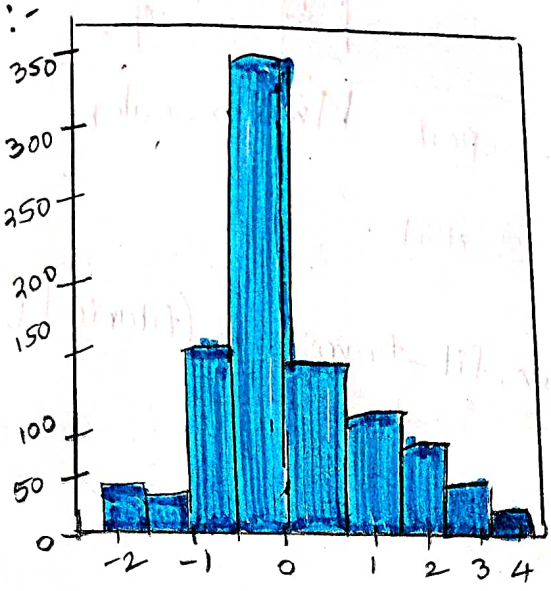


# Histogram of Converted "Age"

# titanic["Age_sc"].hist()

# plt.show()

Out :-




# Here, we see, Histogram is not going to change, But, Values does.

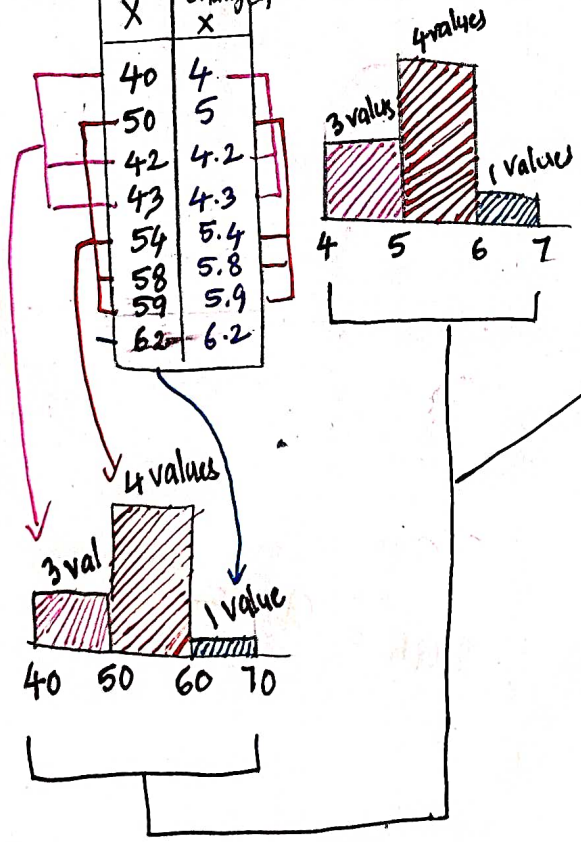
⇒ Example :-

* How ? Histogram Remains Same, But changed Values

Ans :-

Ex :-

| X | Changed x |
|---|---|
| 40 | 4 |
| 50 | 5 |
| 42 | 4.2 |
| 43 | 4.3 |
| 54 | 5.4 |
| 58 | 5.8 |
| 59 | 5.9 |
| 62 | 6.2 |




# Here only "x" values Only changes But Histogram Remains Same.

# II # Normalisation

## * MinMax Scaling

⇒ MinMax Scaling Scales, The Values between "0" to "1" (Every Thing is (+ve) value No -ve value)

Ex:-

| X | |
|---|---|
| 65 | $\frac{65-34}{53}$ |
| 45 | $\frac{45-34}{53}$ |
| 34 ← min. value | $\frac{34-34}{53}=0$ |
| 67 | $\frac{67-34}{53}$ |
| 67 | |
| 87 ← Max. value | $\frac{87-34}{53}=1$ |

$$\frac{X - X_{min}}{X_{max} - X_{min}}$$

MinMax Scaler

Code: From sklearn.Preprocessing import

```
# min_max = MinMaxScaler()  # Shortcut
# titanic ["Age_mm"] = min_max.fit_transform(titanic [["Age"]])
```

fit_transform

creating new column

```
# titanic   (calling/Show Data)
```

Out :-

| | Age | Age_SC | Age_mm |
|---|---|---|---|
| 0 | 22.0 | -0.5653 | 0.2711 |
| 1 | 38.0 | 0.4337 | 0.4722 |
| 2 | 26.0 | -0.1858 | 0.322 |
| ⋮ | ⋮ | ⋮ | ⋮ |
| 888 | 26.0 | -0.25 | 0.32 |
| 890 | 32.0 | 0.20 | 0.39 |

## * Robust Scaling

$$\frac{X - X_{median}}{IQR}$$

$\therefore IQR = Q_3 - Q_1$

Code:

```
From sklearn.preprocessing import RobustScaler

# rs = RobustScaler()  → shortcut()
# titanic["Age_rs"] = rs.fit_transform(titanic[["Age"]])
```

titanic["Age_rs"] (or) = RobustScaler().fit_transform(...)

If not using any shortcut like rs = RobustScaler()

```
# titanic
```

Out

| Age | Age_SC | Age_Mm | Age_RS |
|---|---|---|---|
| 22.0 | 0.565 | 0.461 | 0.4615 |
| 38.0 | 0.66 | 0.769 | 0.769 |
| 26.0 | 0.25 | -0.153 | -0.153 |
| ⋮ | | | |
| 26.0 | -0.25 | 0.153 | 0.153 |
| 32.0 | 0.2 | 0.396 | 0.3076 |

* **Max Abs Scaling**

$$\frac{X}{X_{max}}$$

Ex:-

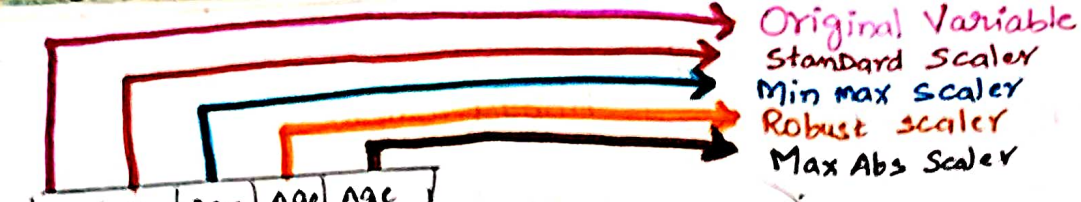
| X | |
|---|---|
| 68 | 68/80 |
| 44 | 44/80 |
| 52 | 52/80 |
| 69 | 69/80 |
| 80 | 80/80 = 1 |

Max value

code

```
From sklearn.preprocessing import MaxAbsScaler

# mas = MaxAbsScaler()

# titanic["Age_mas"] = mas.fit_transform(titanic[["Age"]]

# titanic
```

Out :-




| | Age | Age_sc | Age_mm | Age_rs | Age_mas |
|---|---|---|---|---|---|
| 0 | . | . | . | . | 0.2750 |
| 1 | . | . | . | . | 0.4150 |
| 2 | . | . | . | . | 0.3250 |
| | | | | | ⋮ |
| 889 | . | . | . | . | 0.3250 |
| 890 | . | . | . | . | 0.4000 |

891 x 5 col

5/04/22
4:58 pm.

Example :- Discretization

Continous Data ⇒ Milage of a Car (Discrete, categorical) Data

| |
|---|
| 12.5 |
| 22.8 |
| 25.2 |
| 18.3 |
| 12.4 |
| 15.8 |
| 23.3 |

- <15 → low. milage
- 15-20 → Avg. milage
- 20-25 → good. milage
- >25 → Very good

→ By Our Own choice To Divide the into parts

26/04/22 10:06pm

*******

# Ensemble Learning

Ex :- Avengers.

# Boosting Technique :- it is methodology, not a m.L Algorithm, it is applied to an Existing m.L, mostly applied on Decision Tree.

Ex :- Student

* Exam

⇒ not prepared well, Focus on wrong answers only

* Attempt Exam again

⇒ rectified but missed another section

* Again Exam

⇒ Total correctly done.

* Boosting :-

⇒ Decision Tree 1 :- misclassified records

⇒ Decision Tree 2 :- misclassified records

⇒ Decision Tree 3 :-

* 2nd Weak learner is dependent on 1st weak learner

↓

algorithm doesn't give more accuracy.

* Formula for Boosting :-

$$F_T(x) = \sum_{t=1}^{T} f_t(x) \rightarrow f_t(x) = \alpha_t \, h(x)$$

Summation of t=1 to T

Learning rate

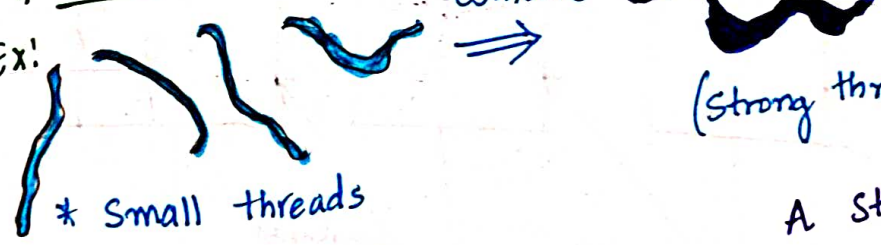
$h(x)$ → [Each model]

* boosting is the Combination of weak learners.

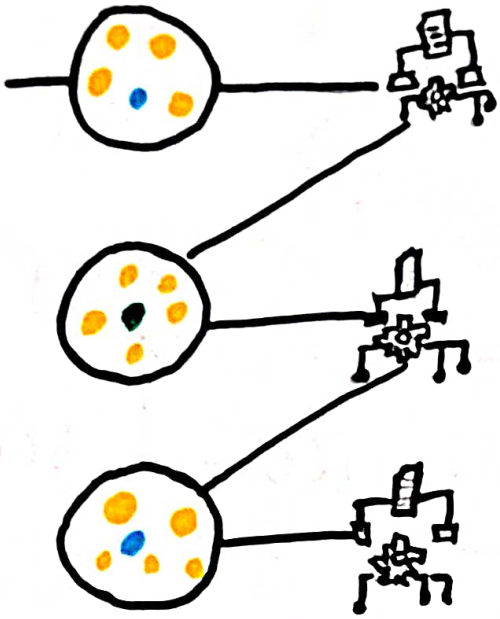
* Implies that Combination of Estimators (models) with an applied Coefficient Could act as an effective Ensemble Estimator.

# Boosting

Ex:




* Multiple Weak classifer (Stumps)



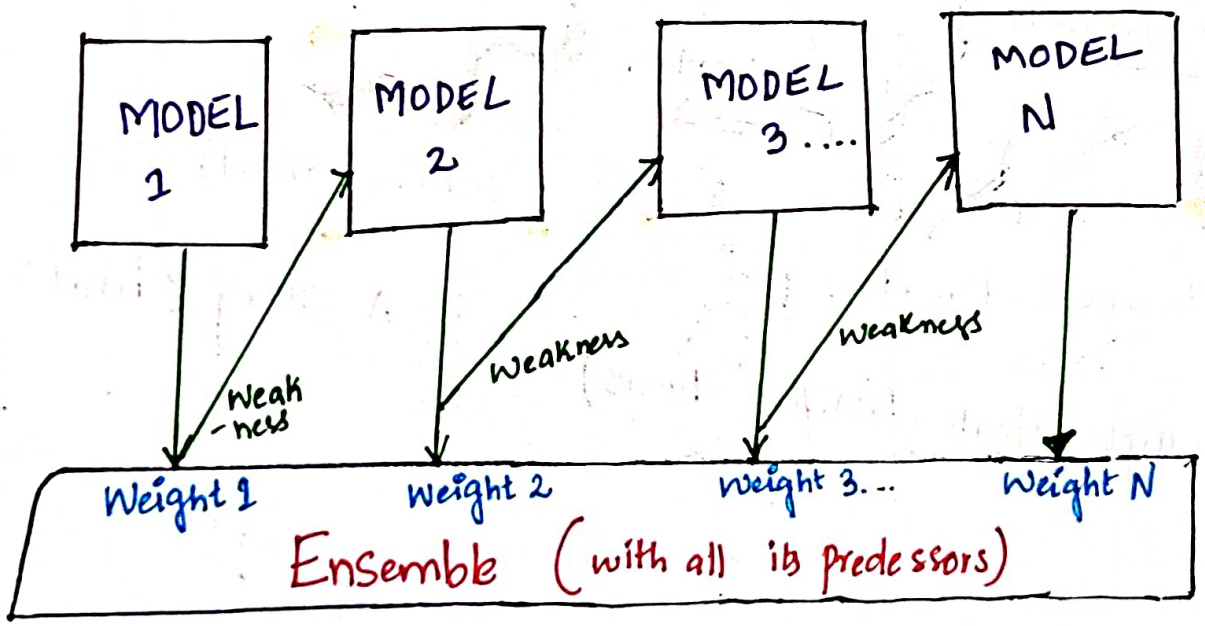
"Sequential learning":

* From the first algorithm whatever mistakes was done
* we share to the second algorithm.
* again it shared to third algorithm. - - -
* it is in "Sequential Order"

# Boosting - "Sequential" Learning.

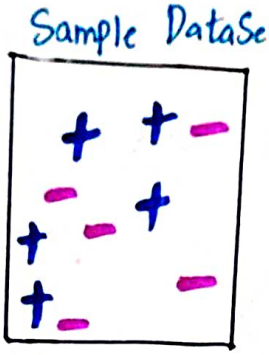
1. Ada Boost → Boosting.
   - Adaptive

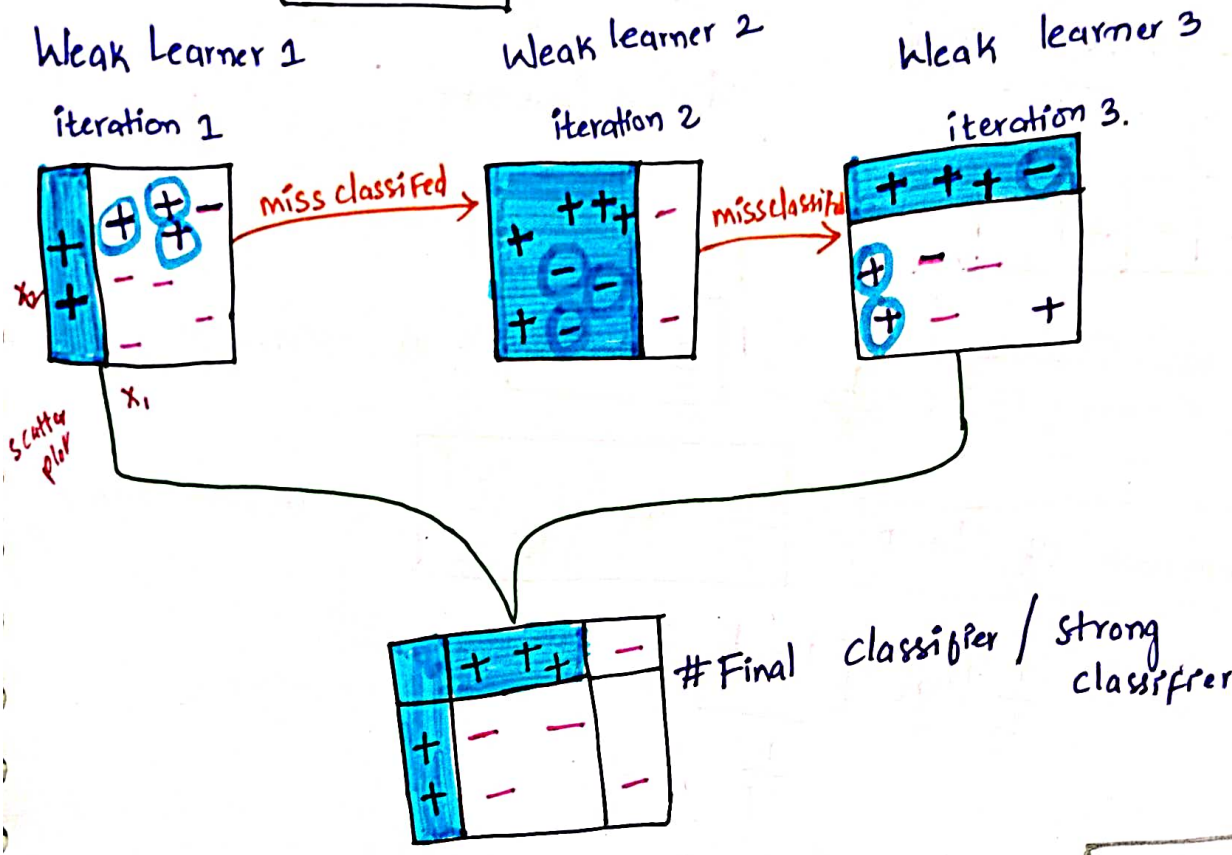
Ex:- Model, 1, 2, ... N are individual models (eg. Decision Tree)



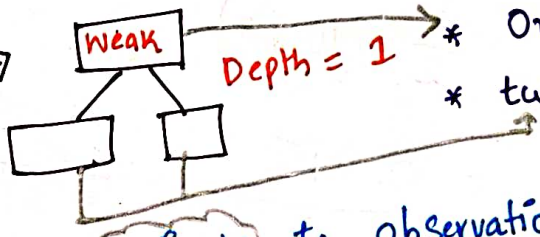
covid (positive / negative)

Ex:-




* In Adaboost :- Each weak learner is called "stump"

* Stump ⇒ 


* One root node
* two Leaf nodes.

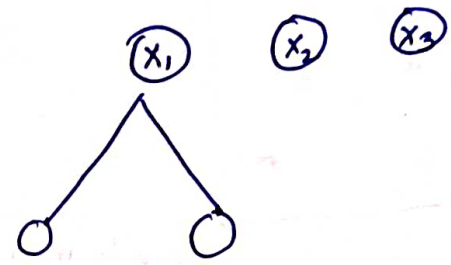
* Step 1 :- Adaboost assigns weights to observations which are incorrectly predicted

Step 2 :- Next model works to predict these values correctly which are wrongly predicted before.

Ex:-

# Taken by Entropy / Information gain

| | $X_1$ | $X_2$ | $X_3$ | $X_4$ | O/p | weights |
|---|---|---|---|---|---|---|
| 1 | – | – | – | – | Yes | 1/7 ✓ |
| 2 | – | – | – | – | No | 1/7 ✓ |
| 3 | – | – | – | – | | 1/7 ✓ |
| X 4 | – | – | – | – | | 1/7 X |
| 5 | – | – | – | – | | 1/7 ✓ |
| 6 | – | – | – | – | | 1/7 ✓ |
| 7 | – | – | – | – | | 1/7 ✓ |




{weak Learner}

*1. **Total Error [TE]** :- $\frac{1}{7}$ :- $\varepsilon_t = \varepsilon_w$ ∴ $\varepsilon_t$ = "Episinol" T

2. **Performance of Stump** :- $\frac{1}{2}\log_e\left[\frac{1-TE}{TE}\right]$ ∴ TE = Total Error

$$= \frac{1}{2}\log_e\left[\frac{1-\frac{1}{7}}{\frac{1}{7}}\right]$$

$$= \frac{1}{2}\log_e\left[\frac{6/7}{1/7}\right]$$

$$= \frac{1}{2}\log_e(6)$$

$$= 0.895$$

**New Sample weight** = $\text{Weight} * e^{-Ps}$ → correct records ∴ Ps = Performance Stump

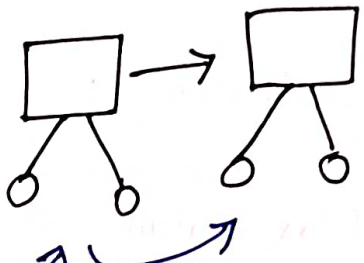
$$= \frac{1}{7} * e^{-0.895} = 0.05$$

+ **incorrect record** = $\text{weight} * e^{Ps}$ → incorrect records.

$$= \frac{1}{7} * e^{0.895} = 0.349$$

is it 1??

| Weights | New weights | Normalized weights | Buckets |
|---|---|---|---|
| $\frac{1}{7}$ | 0.05 ÷ 0.649 | 0.077 | (0 - 0.07) |
| $\frac{1}{7}$ | 0.05 ÷ 0.649 | 0.077 | (0.07 - 0.14) |
| $\frac{1}{7}$ | 0.05 | 0.077 | (0.14 - 0.21) |
| $\frac{1}{7}$ | 0.349 | 0.537 | (0.21 - 0.747) ✓ |
| $\frac{1}{7}$ | 0.05 | 0.077 | [1.747 - 0.751] ✓ |
| $\frac{1}{7}$ | 0.05 | 0.077 | — |
| $\frac{1}{7}$ | 0.05 | 0.077 | — |
| | 0.649 | ~ 1 | |



∴ 1.537 + 0.21 = 1.747

* Steps :-

⇒ Weight = $\frac{1}{n}$

⇒ Total Error_t = $\frac{[\text{correct Prediction} - n]}{n}$

⇒ Total weighted error_t = $\frac{\text{Sum (weight (i) * Error (i))}}{\text{Sum(weight)}}$

⇒ $\alpha_t = \frac{\ln [1 - TE]}{TE\ [\text{Total Error}]}$

⇒ Weight = weight * exp ( $\alpha_t$ * Total error )

⇒ total error = 0 if ($y == y^{hat}$), else 1.

# * Parameters

* n_estimors :- no. of weak learners
* Cretierion :- Gini / Entropy
* Max_Feature :- all Feature / selected ones
* Max_depth :
* min_samples_split
* n_jobs :- System is quadcore :- no. of cores = 4
  dual core :- no. of cores = 2
  octo core : no. of cores = 8
  -1 (all cores)
* Random_state :
* Learning rate :

CODE :- AdaBoost

Data :-

This Data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in agaricus and Lepiota Family (pp 500-525). Each species in indentified as definetly edible, definitely poisonous, or of unknown edibility and not recommended. this latter class was combined with the poisinous one. the Guide clearly states that there is no simple rule for determing the edibility of a mushroom; no rule like "leaflets three, Let it be" for poisinous Oak and Ivy.


→ Mushroom

Attribute Information

1. cap_shape : bell = b, Conical = c, Convex = x, flat = f, Knobbed = k, Sunken = s

2. Cap_Surface : fibrous = f, grooves = g, Scaly = y, Smooth = s

Cap_colour : brown = n, buff = b, cinnamon = c, gray = g, green = r, pink = p, purple = u, red = e, white = w, Yellow = y.

bruises ? : bruises = t, no = F

dor : almond = a, anise = l, creoste = c, fishy = y, foul = f, musty = m, none = n, punget = p, spicy = s

6. gill-attachment : black = k, brown = n, buff = b, chocolate = h,
gray = g, green = r, orange = o, pink = p,
purple = u, red = e, white = w, yellow = y

7. gill-spacing : close = c, crowded = w, distant = d

8. gill-size : broad = b, narrow = n

1. gill-color : black = k, brown = n, buff = b, chocolate = h, grave = g
green = r, orange = o, pink = p, purple = u, red = e,
white = w, yellow = y.

. stalk-shape : enlarging = e, tapering = t

stalk-root : bulbous = b, club = c, cup = u, equal = e,
rhizomorphs = z, rooted = r, missing = ?

stalk surface-above-ring : fibrous = f, scaly = y, silky = k,
smooth = s

stalk surface-below-ring : fibrous = f, scaly = y, silky = k,
Smooth = s

stalk-colour-above-ring : brown = n, buff = b, Cinnamon = c
gray = g, orange = o, pink = p, red = e,
white = w, yellow = y.

stalk-colour-below-ring : "

veil-type : Partial = p, universal = u

veil-color : brown = n, Orange = o, white = w, Yellow = y

18. ring-number :- none = n, One = o, two = t

19. ring-type :- cobwebby = c, evanescent = e, flaring = f, large = l, none = n, Pendent = p, sheathing = s, zone = z,

20. Spore-print Colour :- black = k, brown = n, buff = b, chocolate = h, green = r, orange = o, Purple = u, white = w, Yellow = y

21. Population :- abundant = a, clustered = c, numerous = n, Scattered = s, several = v, solitary = y

22. habitat :- grasses = g, Leaves = l, meadows = m, paths = p, urban = u, waste = w, woods = d.

23. class (Output)

## Bussiness Problem

Goal here is to see if we can harness the power of machine Learning and boosting to help create not just a predective model, but general Guideline for features people should look out for when picking mushrooms.

```
# df = Pd.redd_csv("mushrooms.csv"))
# df.head()
```

| class | cap Shape | cap surface | cap color | bruises | odor | gill attachment | gill spacing | gill size | gill color | ... | Stalk surface above Ring | Stalk color below ring | Stalk color below ring | veil type | veil color | ring number | ring type | Spore print color | Population | habitat |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| p | x | s | n | t | p | f | c | n | k | | s | w | w | p | w | o | p | k | s | u |
| e | x | s | y | t | a | f | c | b | k | | s | w | w | p | w | o | p | n | n | g |
| e | b | s | w | t | l | f | c | b | n | | s | w | w | p | w | o | p | n | n | m |
| p | x | y | w | t | p | f | c | n | n | | s | w | w | p | w | o | p | k | s | u |
| e | x | s | g | f | n | f | w | b | k | | s | w | w | p | w | o | e | n | a | g |

# df.shape

Out: [8124, 23]

# df.info()

Out: Range Index : 8124 Entries, 0 to 8123

| | Column | Non null Count | Dtype. |
|---|---|---|---|
| * | class | 8124 Non null | Object |
| * | cap-shape | " | " |
| * | cap surface | " | " |
| | : | : | : |
| * | habitat | " | " |

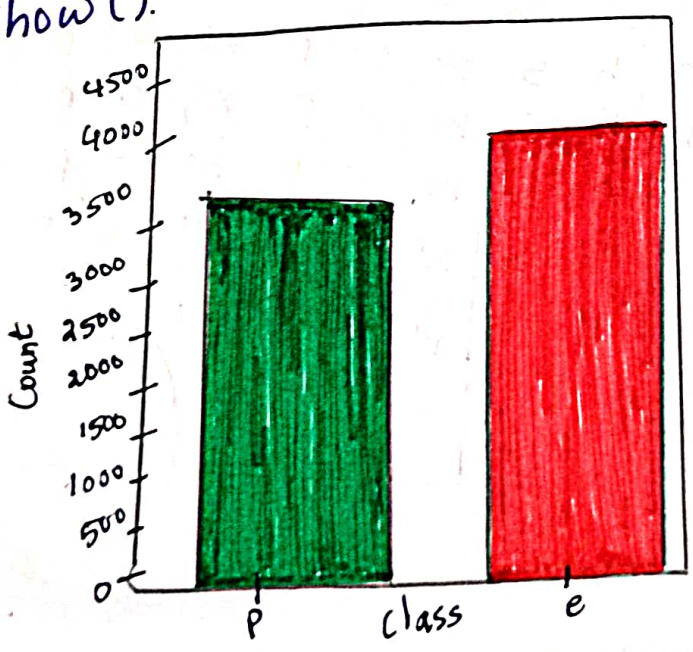
df.isnull().sum()

Out: 0.

EDA

sns.countplot(data=df, X="class", palette="Dark2")
output

plt.show().



# Transpose. gives total no. columns. without ....

```
# df.describe().transpose()
```

Out

| | Count | Unique | top | Freq |
|---|---|---|---|---|
| (o/p) Class | 8124 | 2 | e | 4208 |
| Cap Shape | 8124 | 6 | x | 3656 |
| Cap - Surface | 8124 | 4 | y | 3244 |
| Cap - color | 8124 | 10 | n | 2284 |
| bruises | 8124 | 2 | f | 4748 |
| odor | 8124 | 9 | n | 3528 |
| gill attachment | 8124 | 2 | f | 7914 |
| gill spacing | 8124 | 2 | c | 6812 |
| gill size | 8124 | 2 | b | 5612 |
| gill - color | 8124 | 12 | b | 1728 |
| Stalk - shape | 8124 | 2 | t | 4608 |
| Stalk - root | 8124 | 5 | b | 3776 |
| Stalk - Surface - above ring | 8124 | 4 | s | 5176 |
| Stalk - Surface - below ring | 8124 | 4 | s | 4936 |
| Stalk - color above ring | 8124 | 9 | w | 4464 |
| Stalk - color bebw ring | 8124 | 9 | w | 4384 |
| Veil - type | 8124 | 1 | p | 8124 |
| Veil - color | 8124 | 4 | w | 7924 |
| ring - number | 8124 | 3 | o | 7488 |
| ring - type | 8124 | 5 | p | 3968 |
| Spore - Print color | 8124 | 9 | w | 2388 |
| Population | 8124 | 6 | v | 4040 |
| habitant | 8124 | 7 | d | 3148 |

* posinous
* non posinous

## X & Y

Nomal ... (nominal data) ⇒ One hot Encoding

```
# X = pd.get_dummies(df.drop("class", axis=1), drop_first=True)

# y = df["class"]
```

# x

Out

| cap-shape_c | cap-shape_f | cap-shape_k | cap-shape_s | cap-shape_x | cap-surface_g | cap-surface_s | cap-surface_y | cap-color_c | cap-color_e | ... | population_c | population_s | population_v | population_y | habitat_g | habitat_l | habitat_m | habitat_p | habitat_u |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

8124 rows x 95 columns

# y

Out:

| | |
|---|---|
| 0 | p |
| 1 | e |
| 2 | e |
| 3 | p |
| 4 | e |

```
# x.shape, y.shape
```

Out: (8124, 95), (8124,)

## train / test split

```
from sklearn.model_selection import train_test_split

# x_train, x_test, y_train, y_test = train_test_split (x, y,
                                     test_size = 0.2, random_state=29)
```

## MODELLING

```
from sklearn.ensemble import AdaBoostClassifier

# model = AdaBoostClassifier ()   → n_estimator = 100 (stumps)

# model.fit (x_train, y_train)
```

out : AdaBoostClassifier()

## PREDICTION

```
# ypred_train = model.predict (x_train)

# ypred_test = model.predict (x_test)
```

## Evaluation

* Accuracy

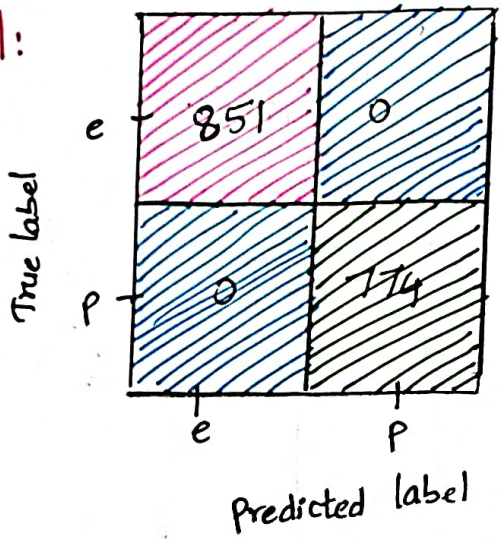
```
from sklearn.metrics import accuracy_score

# Print ("Train accuracy" : , accuracy_score (y_train, ypred_train)
# print ("Test accuray": , accuracy_score (y_test, ypred_test)
```

out:
Train Accuray : 1.0
Test Accuray : 1.0

## * Confusion Matrix

```
from sklearn.metrics import plot_confusion_matrix
# plot_confusion_matrix (model, x_test, y_test)
# plt.show()
```

Out:




## * classification Report

```
from sklearn.metrics import classification_report
# print (classification_report (y_test, y_pred_test))
```

Out:

| | Precision | recall | f1 score | Support |
|---|---|---|---|---|
| e | 1.00 | 1.00 | 1.00 | 851 |
| p | 1.00 | 1.00 | 1.00 | 774 |
| | | | 1.00 | 1625 |
| Accuracy | 1.00 | 1.00 | 1.00 | 1625 |
| macro avg | | 1.00 | 1.00 | 1625 |
| weighted Avg | 1.00 | | | |

* Cross-Validation-Score

```
from sklearn.model_selection import cross_val_score.

# scores = cross_val_score (model, x, y, cv=5)

# print (" cross validation score:", scores.mean())
```

Out : Cross validation score : 0.925

* model.feature_importances

```
# model.feature_importances_

Out  array([0. , 0. , 0. 0, 0. , 0, 0., 0.,
            0. . 0,  0. , 0.02, 0. 0.04,
            0.06, 0.02, 0.0, 0, 0.1, 0.12,
            0 ]

# f_imp = pd.DataFrame (index = X.columns, data = model.feat
                        -re = importances_ columns = ["Feature Import
                                                      ances.
```

```
# f_imp
```

Out: Feature Importances

| | |
|---|---|
| cap_shape_C | 0.00 |
| cap_shape_f | 0.00 |
| cap_shape_K | 0.00 |
| cap_shape_s | 0.00 |
| cap_shape_x | 0.00 |
| ⋮ | ⋮ |

95 rows x 1 columns

# Data Extraction (Pandas)

```
# f_imp[f_imp["feature importance"]>0]
```

Out :

| | Feature Importance |
|---|---|
| cap_color w | 0.02 |
| odor_c | 0.04 |
| odor_f | 0.04 |
| odor_n | 0.06 |
| odor_p | 0.02 |
| gill_spacingw | 0.10 |
| ⋮ | ⋮ |

## Hyper Parameter Tuning

```
from sklearn.model_selection import GridSearchCV
# estimator = AdaBoostClassifier()
# param_grid = {"n_estimators": list(range(1,101))}
# grid = GridSearchCV(estimator, param_grid, cv=5, scoring="Accuray")
# grid.fit(x_train, y_train)
# grid.best_params_
```

Out : {"n_estimator : 20}

## final model

```
# final_model = AdaBoostClassifier (n_estimators = 20)
# final_model.fit (x_train, y_train)

# Pred_train = final_model.predict (x_train)
# Pred_test = final_model.predict (x_test)

# print ("train accuray:", accuray score (y_train, Pred_train)
# print ("test accuray:", accuray score (y_test, Pred_test)
```

Out: train accuray : 1.0
Test accuray : 1.0

25/04/22
10:00pm

```
# final model.feature_importances_
```

Out: array ([0. , 0. , 0., 0. 0.
0.0434 0. 0, 0, 0.
0 0 0 0 0.434...]]

27/4/22
4:30pm.

```
# f_imp2 = Pd.DataFrame (index = X.columns, data = final model
        .feature_importances_, columns = ["importance-
        Feature"])
```

# f_imp2

Out:

| | importance_Feature |
|---|---|
| cap_shape_c | 0.0 |
| cap-shape-f | 0.0 |
| ⋮ | |
| habitat-p | 0.0 |
| habitat-u | 0.0 |

# important = f_imp2 [ f_imp2 [ "importance_feature" ] >0 ]

# important . Sort_Values ( " importance-feature")

Out:

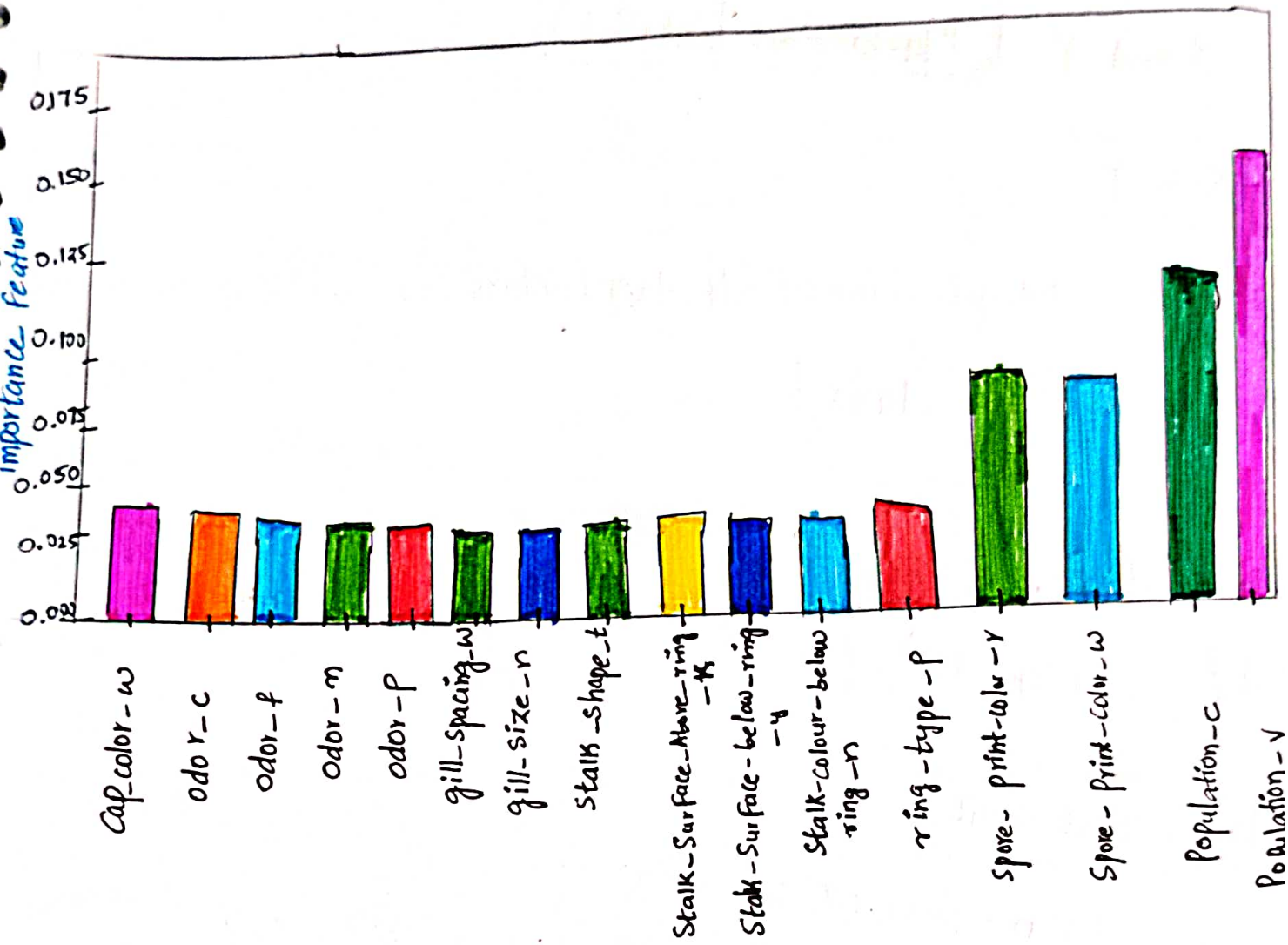
| | importance_Feature |
|---|---|
| cap_color_w | 0.043478 |
| odor_c | 0.043478 |
| odor_f | 0.043478 |
| odor_p | 0.043478 |
| ⋮ | |
| order_n | 0.130435 |
| gill_size_n | 0.173913 |

# plt. figure [fig_size = (14,6), dpi=200)

# sns. barplot (data = important. Sort_values ("importance_feature"), x = important. index, y = "importance_Feature")

# plt.xticks (rotation = 90)

# plt. show ()



***

# Gradient Boosting

* with in Gradient Boost, The **Decision Trees** are **Pruned** To **Maximum Leaf nodes** from **8 to 32.**

Ex:-

| $X_1$ Age | $X_2$ City | y Income | | Prediction $\hat{y}$ (using D.T-1) | Residuals Error |
|---|---|---|---|---|---|
| 32 | A | 51000 | | 53500 | -2500 |
| 30 | B | 78000 | model: 1 → | 61000 | 17000 |
| 21 | A | 20000 | | 28500 | -8500 |
| 27 | B | 44000 | | 61000 | -17000 |
| 36 | B | 89000 | | 90500 | -15000 |
| 25 | A | 37000 | | 28500 | 8500 |

D.T-2

| Prediction | Residuals error |
|---|---|
| -5500 | 3000 |
| 8000 | 9000 |
| -5500 | -3000 |
| -4300 | -12700 |
| 8000 | -9500 |
| 8000 | 5800 |

3000-5100 = 48000

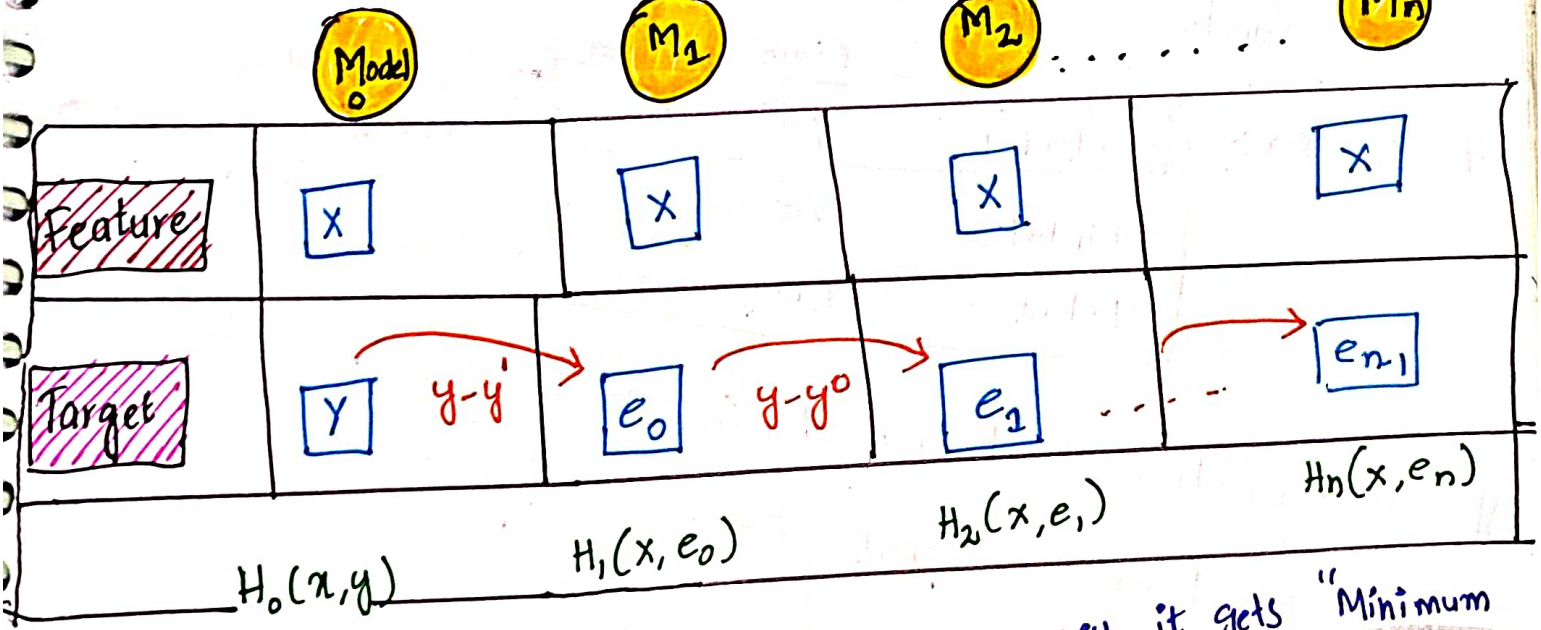
| y-Error |
|---|
| 48000 |
| 69000 |
| 23000 |
| 56700 |
| 98500 |
| 36500 |

model 2 →

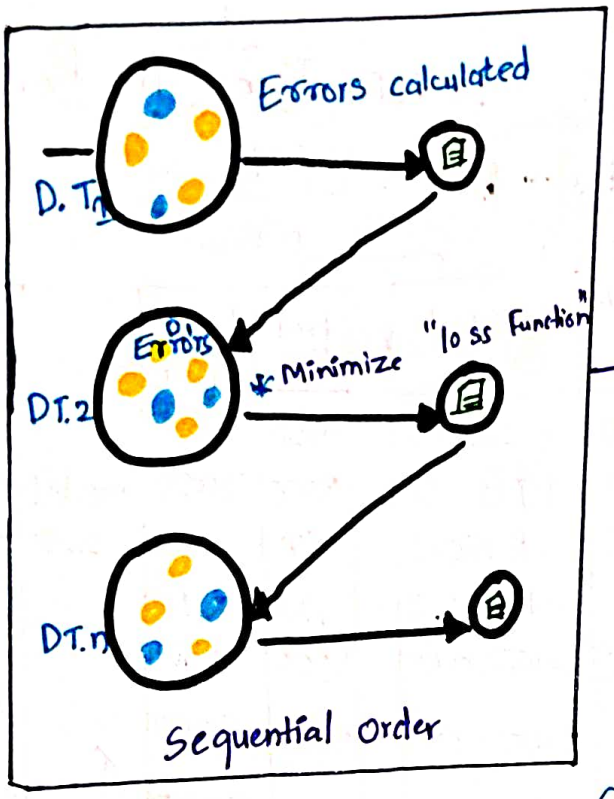
Decision Tree -2

Target (error)min

MODEL 2 Income = MODEL 1 income + Predicted Errors




* it Continues, till it gets "Minimum Error".




→ GBM

"Gradient Boosting method"

↓ How it works

1. A loss function to be optimized
2. A weak learner to make predictions
3. An additive mode to add weak learners to minimum

(or) minimize the loss function.

* **Loss function** :- $(\text{Sum of Error})_{min}$

(or)

(Overall Error should be Minimum)

Que :- What is difference between cost function & loss function ?

Ans :- Ex :-

Total = 100

| | Exam | Error |
|---|---|---|
| Student 1 | 70 | 30 |
| Student 2 | 85 | 15 |
| Student 3 | 60 | 40 |

→ cost function. (Focusing on only one student)

→ loss function

: Cost function :- Error of individual record

Loss function :- Overall Error

Que :- What is difference b/w "Adaboost" and "gradient boost"?

A :- * Adaboost :- We consider only stump/ depth = 1

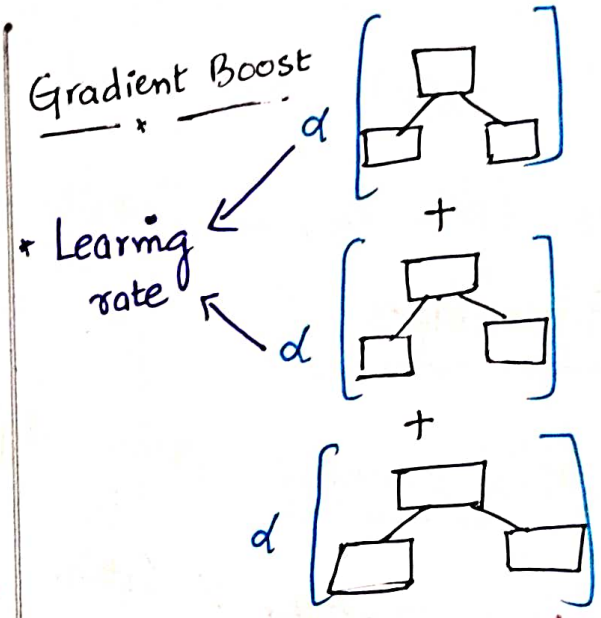
* Gradient boost :- We Consider some what grown The Tree, Max. Leaf nodes = 8 to 32

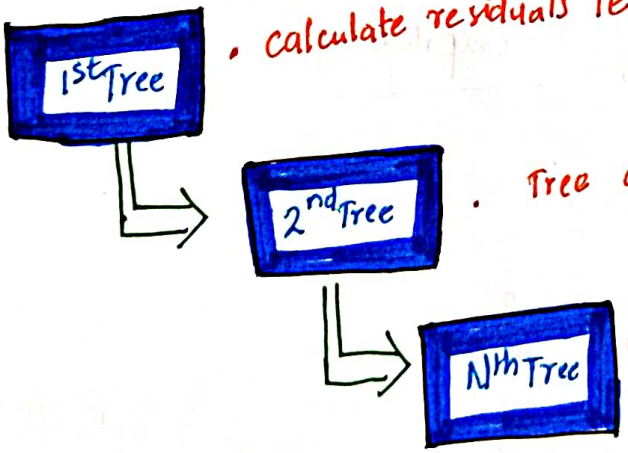
How it works

* $\hat{y}$ hat
* [y_actual - y_hat]^2
* y = m(x) + error 1
* error 1 = G[x] + error 2
* error 2 = H[x] + error 3
* y = m(x) + G[x] + H[x] + error 3
* y = alpha * M(x) + beta * G(x) + gamma * H(x) + error 4

Gradient Boost

* Learning rate




. calculate residuals /Errors

. Tree after adjusting for Residuals /Errors

. Final Tree after adjusting Errors multiple times.

Data :-

This Data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in agaricus and Lepiota Family (pp 500-525). Each species in indentified as definetly edible, definitely poisonous, or of unknown edibility and not recommended. this latter class was combined with the poisinous one. the Guide clearly states that there is no simple rule for determing the edibility of a mushroom; no rule like "leaflets three, Let it be" for poisinous Oak and Ivy.


→ Mushroom

| Attribute | Information |
|---|---|
| 1. cap_shape : | bell = b, Conical = c, Convex = x, flat = f, Knobbed = k, sunken = s |
| 2. Cap_surface : | fibrous = f, grooves = g, Scaly = y, Smooth = s |
| Cap_colour : | brown = n, buff = b, cinnamon = c, gray = g, green = r, pink = p, purple = u, red = e, white = w, Yellow = y. |
| bruises ? : | bruises = t, no = F |
| dor : | almond = a, anise = l, Creoste = c, fishy = y, foul = f, musty = m, none = n, punget = p, spicy = s |

6. gill-attachment : black = k, brown = n, buff = b, chocolate = h,
gray = g, green = r, orange = o, pink = p,
purple = u, red = e, white = w, yellow = y

7. gill-spacing : close = c, crowded = w, distant = d

8. gill-size : broad = b, narrow = n

1. gill-color : black = k, brown = n, buff = b, chocolate = h, grave = g
green = r, orange = o, pink = p, purple = u, red = e,
white = w, yellow = y.

. stalk-shape : enlarging = e, tapering = t

stalk-root : bulbous = b, club = c, cup = u, equal = e,
rhizomorphs = z, rooted = r, missing = ?

stalk surface-above-ring : fibrous = f, scaly = y, silky = k,
smooth = s

stalk surface-below-ring : fibrous = f, scaly = y, silky = k,
Smooth = s

stalk-colour-above-ring : brown = n, buff = b, Cinnamon = c
gray = g, orange = o, pink = p, red = e,
white = w, yellow = y.

stalk-colour-below-ring : "

veil-type : Partial = p, universal = u

veil-color : brown = n, Orange = o, white = w, Yellow = y

18. ring-number :- none = n, One = o, two = t

19. ring-type :- cobwebby = c, evanescent = e, flaring = f, large = l, none = n, Pendent = p, sheathing = s, zone = z,

20. Spore-print Colour :- black = k, brown = n, buff = b, chocolate = h, green = r, orange = o, Purple = u, white = w, Yellow = y

21. Population :- abundant = a, clustered = c, numerous = n, Scattered = s, several = v, solitary = y

22. habitat :- grasses = g, Leaves = l, meadows = m, paths = p, urban = u, waste = w, woods = d.

23. class (Output)

## Bussiness Problem

Goal here is to see if we can harness the power of machine Learning and boosting to help create not just a predective model, but general Guideline for features people should look out for when picking mushrooms.

```
# df = Pd.redd_csv("mushrooms.csv"))
# df.head()
```

| class | cap Shape | cap surface | cap color | bruises | odor | gill attachment | gill spacing | gill size | gill color | ... | Stalk surface above Ring | Stalk color below ring | Stalk color below ring | veil type | veil color | ring number | ring type | Spore print color | Population | habitat |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| p | x | s | n | t | p | f | c | n | k | | s | w | w | p | w | o | p | k | s | u |
| e | x | s | y | t | a | f | c | b | k | | s | w | w | p | w | o | p | n | n | g |
| e | b | s | w | t | l | f | c | b | n | | s | w | w | p | w | o | p | n | n | m |
| p | x | y | w | t | p | f | c | n | n | | s | w | w | p | w | o | p | k | s | u |
| e | x | s | g | f | n | f | w | b | k | | s | w | w | p | w | o | e | n | a | g |

# df.shape

Out: [8124, 23]

# df.info()

Out: Range Index : 8124 Entries, 0 to 8123

| | Column | Non null Count | Dtype. |
|---|---|---|---|
| * | class | 8124 Non null | Object |
| * | cap-shape | " | " |
| * | cap surface | " | " |
| | : | : | : |
| * | habitat | " | " |

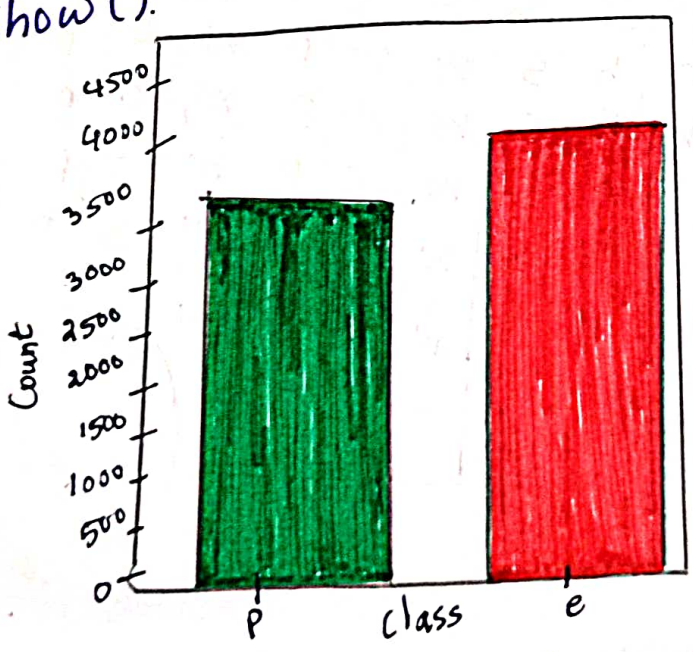
df.isnull().sum()

Out: 0.

EDA

sns.countplot(data=df, X="class", palette="Dark2")
output

plt.show().



# Transpose. gives total no. columns. without ....

```
# df.describe().transpose()
```

Out

| | Count | Unique | top | Freq |
|---|---|---|---|---|
| (o/p) Class | 8124 | 2 | e | 4208 |
| Cap Shape | 8124 | 6 | x | 3656 |
| Cap - Surface | 8124 | 4 | y | 3244 |
| Cap - color | 8124 | 10 | n | 2284 |
| bruises | 8124 | 2 | f | 4748 |
| odor | 8124 | 9 | n | 3528 |
| gill attachment | 8124 | 2 | f | 7914 |
| gill spacing | 8124 | 2 | c | 6812 |
| gill size | 8124 | 2 | b | 5612 |
| gill - color | 8124 | 12 | b | 1728 |
| Stalk - shape | 8124 | 2 | t | 4608 |
| Stalk - root | 8124 | 5 | b | 3776 |
| Stalk - Surface - above ring | 8124 | 4 | s | 5176 |
| Stalk - Surface - below ring | 8124 | 4 | s | 4936 |
| Stalk - color above ring | 8124 | 9 | w | 4464 |
| Stalk - color bebw ring | 8124 | 9 | w | 4384 |
| Veil - type | 8124 | 1 | p | 8124 |
| Veil - color | 8124 | 4 | w | 7924 |
| ring - number | 8124 | 3 | o | 7488 |
| ring - type | 8124 | 5 | p | 3968 |
| Spore - Print color | 8124 | 9 | w | 2388 |
| Population | 8124 | 6 | v | 4040 |
| habitant | 8124 | 7 | d | 3148 |

* posinous
* non posinous

CODE :

## Gradient Boost :-

Same Data. and Same Procedure from import to Xand Y (Mushroom DataSet)

### X & Y

```
# x = pd.get_dummies (df.drop("class",axis = 1 drop_first = True)
# y = df ["class"]

# X.shape, y.shape
```

Out: (8124, 95), (8124,))

### train Test Split

```
from sklearn.model_selection import train_test_split
# x_train, x_test, y_train, y_test = train_test_split (x, y, test_size = 0.
                                     random_state = 29)
```

### modelling

```
from sklearn.ensemble import Gradient Boosting Classifier
# gradmodel = Gradient Boosting classifier ()
# gradmodel.fit (x_train, y_train)
```

Out : Gradient Boosting classifier ()

## Prediction

```
# ypred_train = gradmodel.predict(x_train)
# ypred_test = gradmodel.predict(x_test)
```

## Evaluation

* **Accuracy**

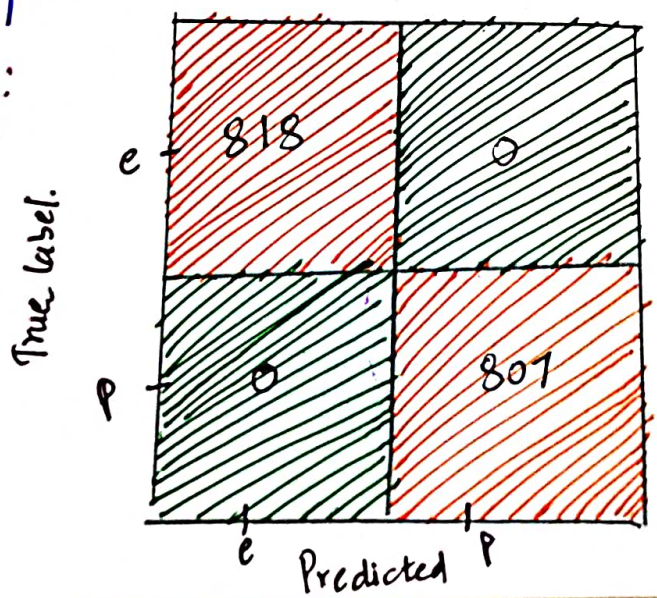
```
from sklearn.metrics import accuracy_score
# print("train accuracy:", accuracy_score(y_train, ypred_train))
# print("test accuracy:", accuracy_score(y_test, ypred_test))
```

Out: train accuracy : 1.0
Test accuracy : 1.0

* **Confusion Matrix**

```
from sklearn.metrics import plot_confusion_matrix
# plot_confusion_matrix(gradmodel, X_test, y_test)
# plt.show()
```

Out:



## * Classification report

```
from sklearn.metrics import classification_report
# Print (classification_report (y_test, ypred_test))
```

Out :

| | Precision | Recall | f1-score | Support |
|---|---|---|---|---|
| e | 1.00 | 1.00 | 1.00 | 818 |
| p | 1.00 | 1.00 | 1.00 | 807 |
| | | | | 1625 |
| accuracy | | | 1.00 | 1625 |
| macro avg | 1.00 | 1.00 | 1.00 | 1625 |
| Weighted avg | 1.00 | 1.00 | 1.00 | 1625 |

## * Cross-validation Score

```
from sklearn.model_selection import cross_val_score
# scores = cross_val_score (gradmodel, x, y, cv=5)
# Print ( "cross_val_score :", scores.mean())
```

Out : cross_val_score : 0.9192

## * feature importance

```
# gradmodel.feature_importances_
```

Out : array ([ 0.00e+00, 1.0465, 1.93018, 0.000e+0,
6.19492, 1.36263, 4.49731 -3.5778, ...])

# Hyper Parameter Tunning

```
from sklearn.model_selection import GridSearchCV

# estimator = GradientBoostingClassifier()
# param_grid = {"n_estimators": [1,5,10,20,40,100], "learning_rate":
                                                    [0.1, 0.2, 0.5, 0.9]}
```

Important Parameter for GradientBoost

```
# grid = GridSearchCV(estimator, param_grid, scoring="accuracy"
                                                    , cv=5)

# grid.fit(x_train, y_train)

# grid.best_params_
```

Out: {"learning_rate": 0.2, "n_estimators": 100}

## Final Model

```
# final model = GradientBoostingClassifier(n_estimators=100,
                                            learning_rate=0.2)
# final model.fit(x_train, y_train)

# ypred_train = final model.predict(x_train)
# ypred_test = final model.predict(x_test)
# print("train accuracy:", accuracy_score(y_train, ypred_train)
# print("test accuracy:", accuracy_score(y_test, ypred_test)
```

Out: train accuracy: 1.0
test accuracy : 1.0

# final model . feature_ importances_

Out : array ([ 0.000e+00, 2.45745e-16, 1.27008, 4.735
1.137228 1.485440 1.923168 , 3.20965)
0.000 bev ]

# important = pd. Data Frame (index = x.columns, data = final model.
feature_ importances , Columns = ["importance_feature

# important

Out :

| | importance_ Feature |
|---|---|
| Cap_shape_c | : 0.0000e+00 |
| Cap_ shape_f | : 2.45745e-16 |
| Cap_ shape_K | : 1.270085e-23 |
| Cap_ shape_s | : 4.735679e-08 |
| ⋮ | ⋮ |
| habitat_u | : 9.502012e-05 |
| habitat_w | : 0.0000 e+00 |

# Jack = important [important ["importance_ feature"] > 0.01]

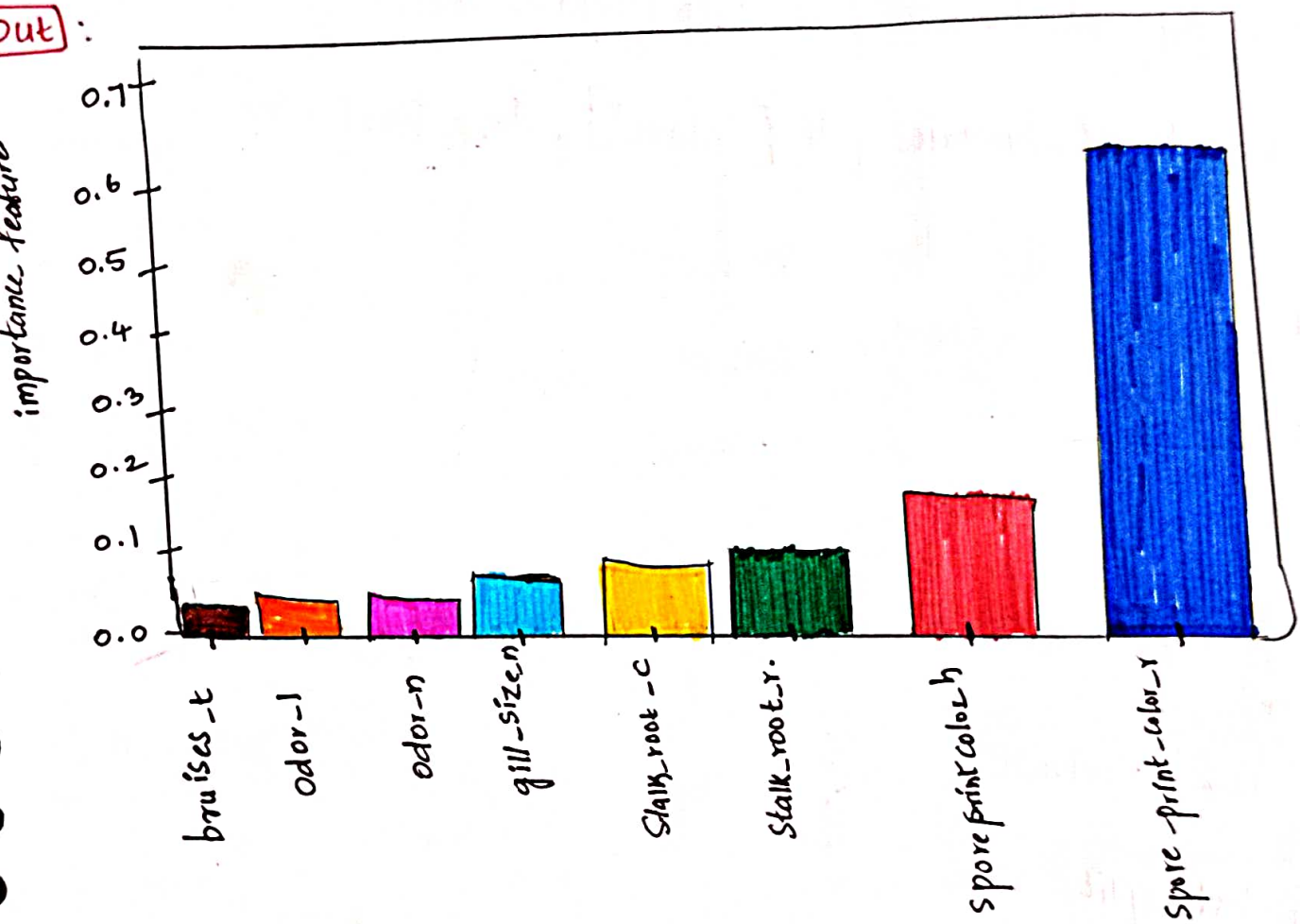
# Jack. Sort_ Values ( " importance_ Feature")

Out :

| | importance_ feature |
|---|---|
| gill_size_n | 0.010084 |
| Spore print color_h | 0.014901 |
| odor_l | 0.019874 |
| Spore printcolor_r | 0.032914 |
| Stalk root_r | 0.052216 |
| bruises_t | 0.060678 . . . . |

```
# plt.figure (figsize=(14,6), dpi = 200)
# sns.barplot (data = Jack.sort_values ("importance_feature"),
                x = Jack.index , y = "importance_feature")

# plt.xticks( rotation = 90)
# plt.show()
```

Out:



**** XG BOOST → Gradient

↓

Xtreme

* Adding "Penality" term Reduce over all Error
* "10" times Faster Than "Gradient Boosting".
* "XG" Boost also called as "Regularized Boosting".
  * Like $L_1$, $L_2$ Regularization

* Advantages

* Tree Pruning Using depth first approach
* High-flexibity :- Catche awarness and out of core computing. (-1) it uses all cores in system
* Reduce Overfitting :- "Regularization" for avoiding overfitting

Efficient Handling missing data

Inbuilt "cross-validation" Capability

XG-boost

error + Penality Term

$L_2$ : $\alpha$ ( slope)$^2$

$L_1$ : $\alpha$ |slope|

with in "XG Boost :-
$\alpha$ = gamma

* **XG boost classifier** Black box Model.

**Base model** = Probability = 0.5

Ex:- DataSet (Pr - Approval)

| Salary | Credit | Approval | Residuals |
|---|---|---|---|
| <= 50k | Bad | 0 | 0 - 0.5 = -0.5 |
| <= 50k | Good | 1 | 1 - 0.5 = 0.5 |
| <= 50k | Good | 1 | 0.5 |
| > 50k | Bad | 0 | -0.5 |
| > 50k | Good | 1 | 0.5 |
| > 50k | Normal | 1 | 0.5 |
| <= 50k | Normal | 0 | -0.5 |

* **Steps** :-

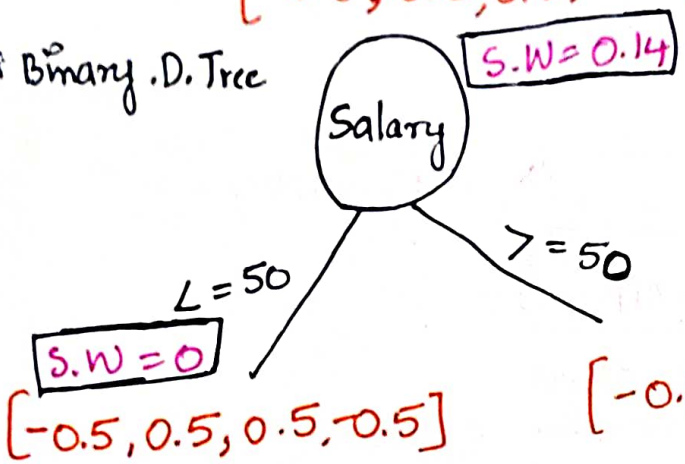
1. Create Binary Decision Tree using Features
2. Calculate Similarity weight

$$= \frac{\Sigma [\text{Residual}]^2}{\Sigma [\text{Probability}\,[1 - \text{Probability}] + \lambda]} \rightarrow + \lambda \text{ (hyper parameter)}$$

3. Information Gain.

Errors :- $[-0.5, 0.5, 0.5, -0.5, 0.5, 0.5, -0.5]$

* Binary .D. Tree




$$\text{Similarity weight} \Rightarrow \frac{\varepsilon(res)^2}{\varepsilon p(1-p)\lambda}$$

$$= \frac{[-0.5+0.5+0.5-0.5]^2}{0.5[1-0.5]+0.5[1-0.5]+0.5[1-0.5]+0.5[1-0.5]+0}$$

$\because \lambda = 0$

$$= 0$$

$$\text{Similarity weight} = \frac{\varepsilon(res)^2}{\varepsilon p(1-p)\lambda}$$

$$= \frac{[-0.5+0.5+0.5]^2}{0.5[1-0.5]+0.5[1-0.5]+0.5[1-0.5]+\frac{0}{\lambda}}$$

$$= \frac{0.25}{0.75} = \frac{1}{3} \Rightarrow 0.33$$

# For Root node Similarity weight $= \frac{\varepsilon(res)^2}{\varepsilon p(1-p)\lambda}$

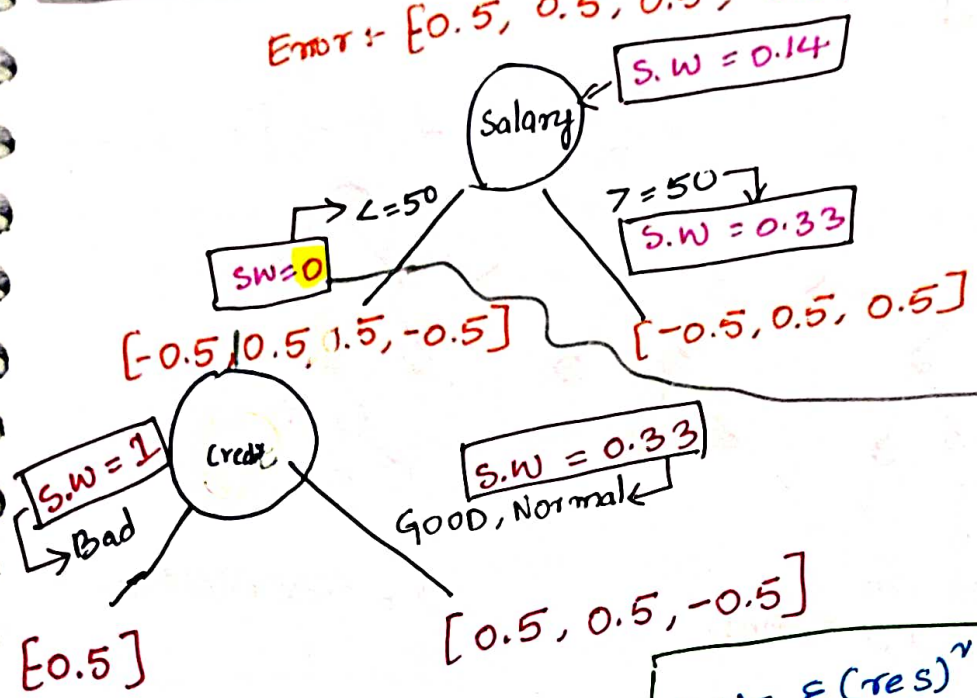
$$= \frac{[-0.5, 0.5, 0.5, -0.5, 0.5, 0.5, -0.5]^2}{0.5[1-0.5]+0.5[1-0.5]+0.5[1-0.5]+0.5[1-0.5]+0.5[1-0.5]+0.5[1-0.5]+0.5[1-0.5]}$$

$$= \frac{0.25}{1.75} = \frac{1}{7} = 0.14$$

* Information Gain = 0 + 0.33 − 0.14

↳ = 0.19

Error = [0.5, 0.5, 0.5, -0.5, 0.5, 0.5, -0.5]




* $S.W = \frac{\varepsilon(res)^2}{\varepsilon p(1-p)}$

$= \frac{[0.5]^2}{0.5[1-0.5]} = \frac{0.25}{0.25}$

= 1

* $S.W = \frac{\varepsilon(res)^2}{\varepsilon p(1-p)}$

$= \frac{[0.5, 0.5, -0.5]^2}{0.5[1-0.5] + 0.5[1-0.5] + 0.5[1-0.5]}$

$= \frac{0.25}{0.75} = \frac{1}{3} = 0.33$

* Information Gain : 1 + 0.33 − 0 → ?

= 1.33

For classification

| Salary | Credit | Approval |
|---|---|---|
| <=50 | → B | 0 |

Goes to binary

activation function → sigmoid

$\sigma\left[0 + \alpha(1)\right]^{\alpha=0.01}$ ⇒ Based on "α" value → o/p 0-1

Learning rate, S.W

∴ Base model = 0.5

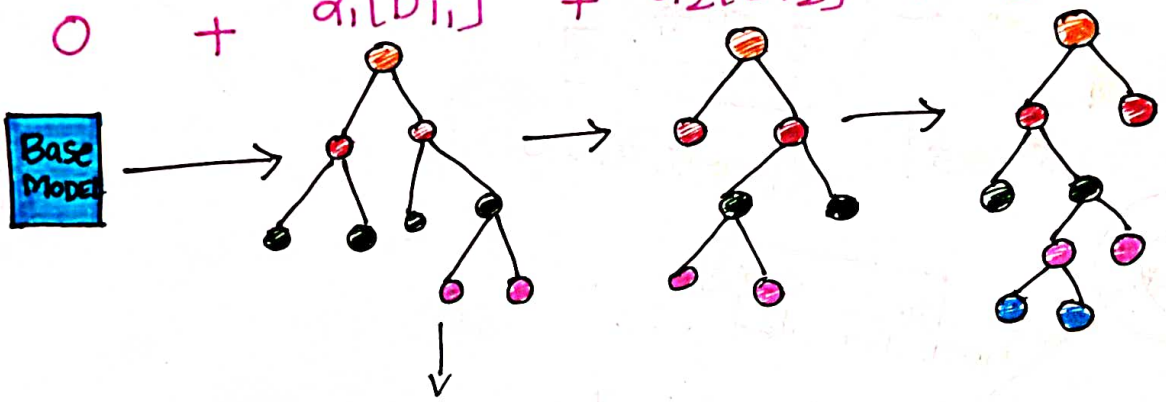
Real probability = $\log\left(\frac{p}{1-p}\right) = \log\left(\frac{0.5}{1-0.5}\right) = 0$

Output :-

$$\left[\sigma\left[0+\alpha_1[DT_1]+\alpha_2[DT_2]+\alpha_3[DT_3]+\cdots\alpha_n[DT_n]\right]\right.$$

For any new record

$$0 + \alpha_1[DT_1] + \alpha_2[DT_2] + \alpha_3[DT_3] + \cdots\cdots$$




Based on independent Features.

$\therefore \lambda$ = Cross validation.

* **XG boost regressor**

* Base model = 51K

Ex:-

| $X_1$ Exp | $X_2$ Gap | $y$ Salary | Residual |
|---|---|---|---|
| 2 | Yes | 40 K | -11K |
| 2.5 | Yes | 42 K | 9 |
| 3 | NO | 52 K | 1 |
| 4 | No | 60 K | 9 |
| 4.5 | Yes | 62 K | 11 |

Avg = 51.

S.W = $\frac{1}{6}$

[-11, -9, 1, 9, 11]

$$\frac{[-11 - 9 + 1 + 9 + 11]^2}{5+1} = \frac{1}{6}$$

Experience

S.W = 45.5 (< 2) → [-11]

S.W = 28.8 (> 2) → [-9, 1, 9, 11]

$$\frac{[-9 + 1 + 9 + 11]^2}{4+1} \quad \lambda = 1$$

$$= \frac{[12]^2}{5} = \frac{144}{5} = 28.8$$

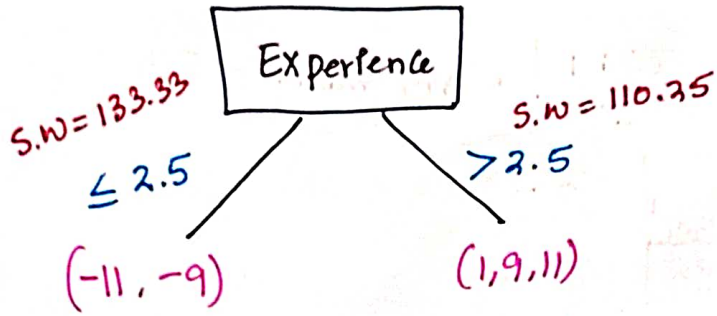
* Similarity weight = $\frac{\Sigma [\text{Residual}]^2}{\text{no. of residuals} + \lambda}$

$$= \frac{[-11]^2}{1+1} = \frac{121}{2} = 60.5$$

$\therefore \lambda = 1$

* Information Gain = $60.5 + 28.8 - \frac{1}{6} = 89.13$

S.W 1/6




$$S.W = \frac{[-11-9]^2}{2+1} = \frac{[-20]^2}{3} = \frac{400}{3} \Rightarrow 133.33$$

$$= \frac{[1+9+11]^2}{3+1} = \frac{[21]^2}{4} = 110.25$$

any records goes from base model = 51

≤ 2.5 Avg. of S.W

$51 + \alpha_1 \left[\frac{[-11-9]}{2}\right] = (-10)$

* $51 + \alpha_1 [-10]$

Avg. of S.W

$51 + \alpha_1 \left[\frac{1+11+9}{3}\right] \Rightarrow \frac{21}{3} = 7$

* $51 + \alpha_1 [7]$

* $51 + \alpha_1 (-10) + \alpha_2 [DT_2] + \alpha_3 [DT_3] - \cdots \alpha_n [DT_n]$

↳ Output.

27/04/22
6:00 Am.

Data :-

This Data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in agaricus and Lepiota Family (pp 500-525). Each species in indentified as definetly edible, definitely poisonous, or of unknown edibility and not recommended. this latter class was combined with the poisinous one. the Guide clearly states that there is no simple rule for determing the edibility of a mushroom; no rule like "leaflets three, Let it be" for poisinous Oak and Ivy.


→ Mushroom

| Attribute | Information |
|---|---|
| 1. cap_shape : | bell = b, Conical = c, Convex = x, flat = f, Knobbed = k, sunken = s |
| 2. Cap_surface : | fibrous = f, grooves = g, Scaly = y, Smooth = s |
| Cap_colour : | brown = n, buff = b, cinnamon = c, gray = g, green = r, pink = p, purple = u, red = e, white = w, Yellow = y. |
| bruises ? : | bruises = t, no = F |
| dor : | almond = a, anise = l, Creoste = c, fishy = y, foul = f, musty = m, none = n, punget = p, spicy = s |

6. gill-attachment : black = k, brown = n, buff = b, chocolate = h,
gray = g, green = r, orange = o, pink = p,
purple = u, red = e, white = w, yellow = y

7. gill-spacing : close = c, crowded = w, distant = d

8. gill-size : broad = b, narrow = n

1. gill-color : black = k, brown = n, buff = b, chocolate = h, grave = g
green = r, orange = o, pink = p, purple = u, red = e,
white = w, yellow = y.

. stalk-shape : enlarging = e, tapering = t

stalk-root : bulbous = b, club = c, cup = u, equal = e,
rhizomorphs = z, rooted = r, missing = ?

stalk surface-above-ring : fibrous = f, scaly = y, silky = k,
smooth = s

stalk surface-below-ring : fibrous = f, scaly = y, silky = k,
Smooth = s

stalk-colour-above-ring : brown = n, buff = b, Cinnamon = c
gray = g, orange = o, pink = p, red = e,
white = w, yellow = y.

stalk-colour-below-ring : "

veil-type : Partial = p, universal = u

veil-color : brown = n, Orange = o, white = w, Yellow = y

18. ring-number :- none = n, One = o, two = t

19. ring-type :- cobwebby = c, evanescent = e, flaring = f, large = l, none = n, Pendent = p, sheathing = s, zone = z,

20. Spore-print Colour :- black = k, brown = n, buff = b, chocolate = h, green = r, orange = o, Purple = u, white = w, Yellow = y

21. Population :- abundant = a, clustered = c, numerous = n, Scattered = s, several = v, solitary = y

22. habitat :- grasses = g, Leaves = l, meadows = m, paths = p, urban = u, waste = w, woods = d.

23. class (Output)

## Bussiness Problem

Goal here is to see if we can harness the power of machine Learning and boosting to help create not just a predective model, but general Guideline for features people should look out for when picking mushrooms.

```
# df = Pd.redd_csv("mushrooms.csv"))
# df.head()
```

| class | cap Shape | cap surface | cap color | bruises | odor | gill attachment | gill spacing | gill size | gill color | ... | Stalk surface above Ring | Stalk color below ring | Stalk color below ring | veil type | veil color | ring number | ring type | Spore print color | Population | habitat |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| p | x | s | n | t | p | f | c | n | k | | s | w | w | p | w | o | p | k | s | u |
| e | x | s | y | t | a | f | c | b | k | | s | w | w | p | w | o | p | n | n | g |
| e | b | s | w | t | l | f | c | b | n | | s | w | w | p | w | o | p | n | n | m |
| p | x | y | w | t | p | f | c | n | n | | s | w | w | p | w | o | p | k | s | u |
| e | x | s | g | f | n | f | w | b | k | | s | w | w | p | w | o | e | n | a | g |

# df.shape

Out: [8124, 23]

# df.info()

Out: Range Index : 8124 Entries, 0 to 8123

| | Column | Non null Count | Dtype. |
|---|---|---|---|
| * | class | 8124 Non null | Object |
| * | cap-shape | " | " |
| * | cap surface | " | " |
| | : | : | : |
| * | habitat | " | " |

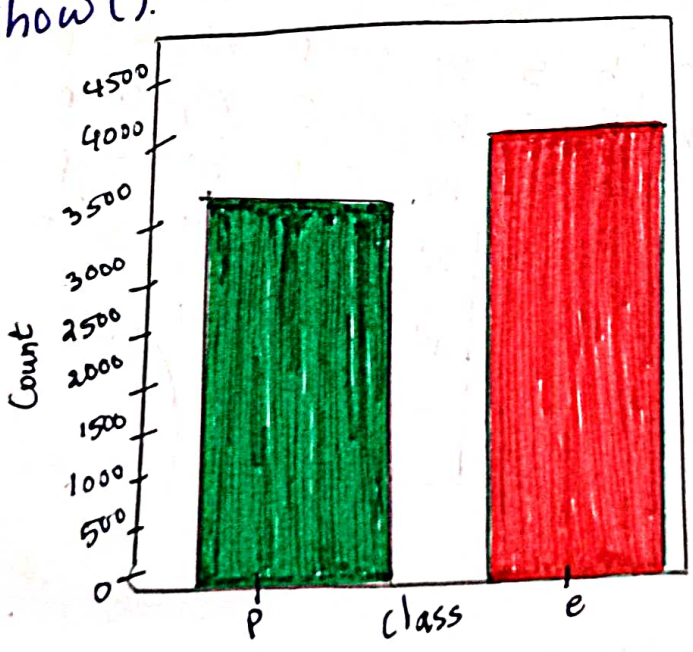
df.isnull().sum()

Out: 0.

EDA

sns.countplot(data=df, X="class", palette="Dark2")
output

plt.show().



# Transpose. gives total no. columns. without ....

```
# df.describe().transpose()
```

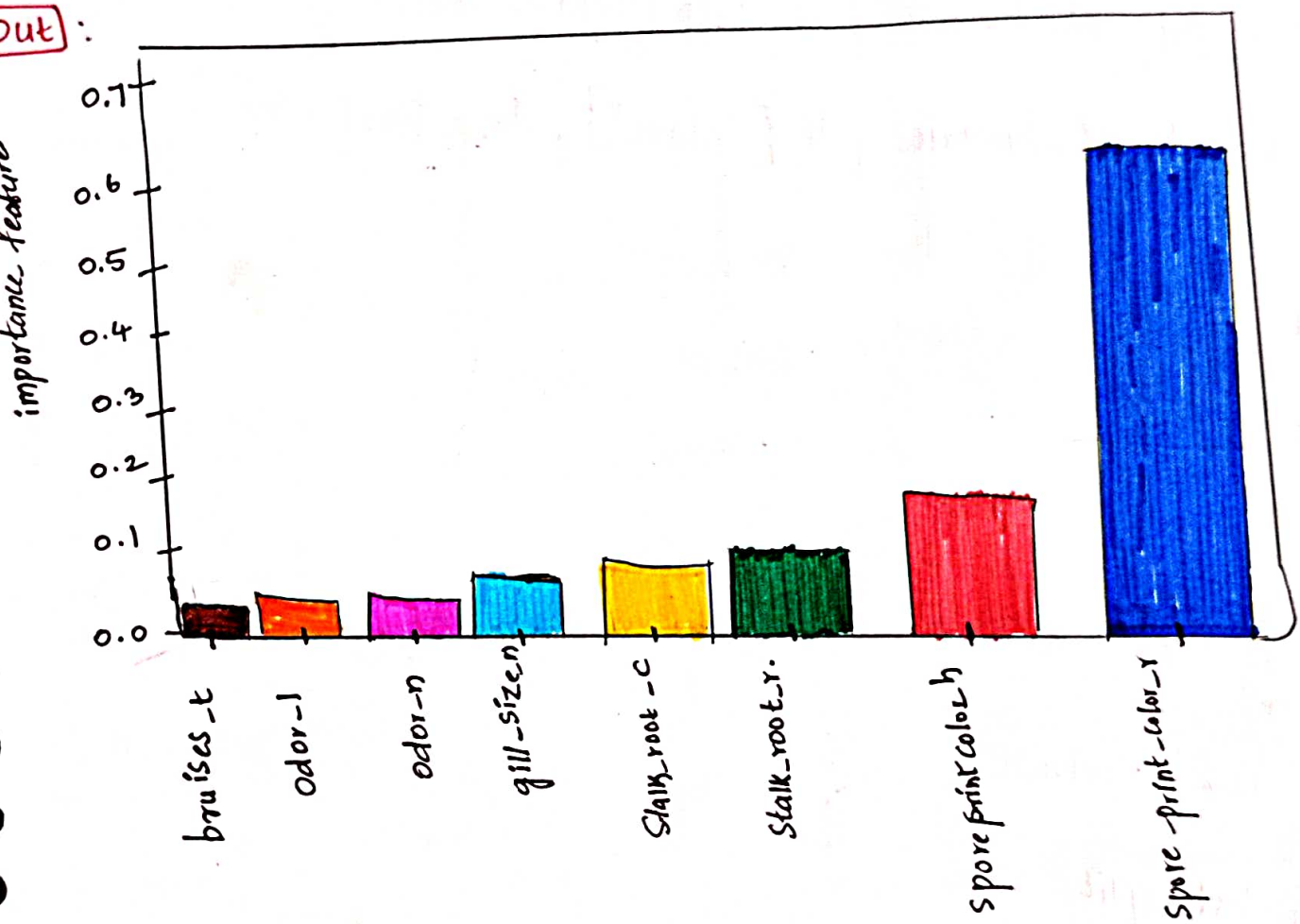
Out

| | Count | Unique | top | Freq |
|---|---|---|---|---|
| (o/p) Class | 8124 | 2 | e | 4208 |
| Cap Shape | 8124 | 6 | x | 3656 |
| Cap - Surface | 8124 | 4 | y | 3244 |
| Cap - color | 8124 | 10 | n | 2284 |
| bruises | 8124 | 2 | f | 4748 |
| odor | 8124 | 9 | n | 3528 |
| gill attachment | 8124 | 2 | f | 7914 |
| gill spacing | 8124 | 2 | c | 6812 |
| gill size | 8124 | 2 | b | 5612 |
| gill - color | 8124 | 12 | b | 1728 |
| Stalk - shape | 8124 | 2 | t | 4608 |
| Stalk - root | 8124 | 5 | b | 3776 |
| Stalk - Surface - above ring | 8124 | 4 | s | 5176 |
| Stalk - Surface - below ring | 8124 | 4 | s | 4936 |
| Stalk - color above ring | 8124 | 9 | w | 4464 |
| Stalk - color bebw ring | 8124 | 9 | w | 4384 |
| Veil - type | 8124 | 1 | p | 8124 |
| Veil - color | 8124 | 4 | w | 7924 |
| ring - number | 8124 | 3 | o | 7488 |
| ring - type | 8124 | 5 | p | 3968 |
| Spore - Print color | 8124 | 9 | w | 2388 |
| Population | 8124 | 6 | v | 4040 |
| habitant | 8124 | 7 | d | 3148 |

* posinous
* non posinous

```
# plt.figure (figsize=(14,6), dpi = 200)
# sns.barplot (data = Jack.sort_values ("importance_feature"),
                x = Jack.index , y = "importance_feature")

# plt.xticks( rotation = 90)
# plt.show()
```

Out:



Code :

# XG boost Classifier

Same Example, Same Question of Mushroom dataset.

import to X & Y.

## X & Y

```
# X = pd.get_dummies(df.drop("class", axis=1), drop_first=True)
# y = pd.get_dummies(df["class"], drop_first=True)
```

```
# y
```

out :

| | p |
|---|---|
| 0 | 1 |
| 1 | 0 |
| 2 | 0 |
| 3 | 1 |
| 4 | 0 |
| ⋮ | ⋮ |

8124 x 1 column

## train test split

```
from sklearn.model_selection import train_test_split

# x_train, x_test, y_train, y_test = train_test_split(x, y,
                                     test_size = 0.2, random_state=29
```

## modelling

```
! Pip install xgboost

from xgboost import XGBClassifier

# xgb_model = XGBClassifier()

# xgb_model.fit(x_train, y_train)
```

Out: XGBClassifier(base_score=0.5, booster='gbtree', ...
.. .. . .... --)

## Prediction

```
# ypred_train = xgb_model.predict(x_train)

# ypred_test = xgb_model.predict(x_test)
```

## Evaluation

### * Accuracy

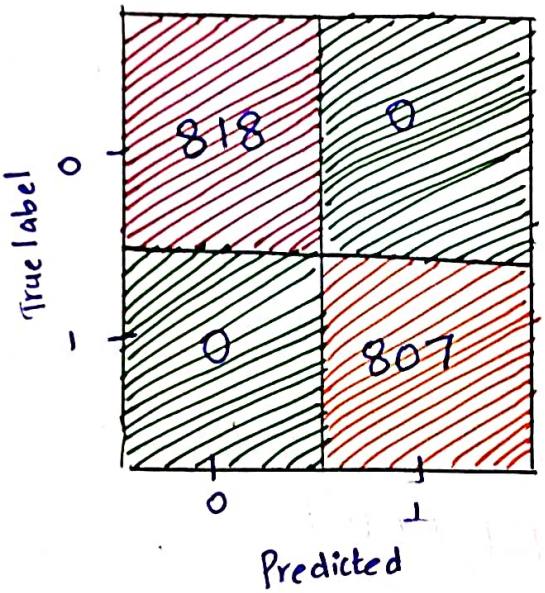
```
from sklearn.metrics import accuracy_score

# print("train accuracy:", accuracy_score(y_train, ypred_train

# print("test accuracy:", accuracy_score(y_test, ypred_tes
```

Out:
Train accuracy : 1.0
Test accuracy : 1.0

## * Confusion Matrix

```
from sklearn.metrics import plot_confusion_matrix
# plot_confusion_matrix (xgb_model , x_test , y_test)
# plt.show()
```

Out :




## * classification report

```
from sklearn.metrics import Classification_report
# Print (classification_report (y_test , ypred_test)
```

Out :

| | Precision | recall | f1 score | Support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 818 |
| 1 | 1.00 | 1.00 | 1.00 | 807 |
| accuracy | | | 1.00 | 1625 |
| macro avg | 1.00 | 1.00 | 1.00 | 1625 |
| Weighted Avg | 1.00 | 1.00 | 1.00 | 1625 |

## * Cross Validation Score

```
from Sklearn.model_selection import Cross_val_score

# scores = Cross_val_score (xgb_model, x, y, cv=5)
# print ("Cross_val_score:", scores.mean())
```

Out: Cross-val-score : 0.935

## * Feature importance

```
# xgb_model.feature_importances_
```

Out: array([ 0.0000e+0 , 0.0000e+00 , 0.0000e+0, 0.0000e+0
3.961512e5 0.00000e+0 , 9.23.225 ---])

## Hyper Parameter tuning

Important Paramen in XGboos

```
from sklearn.model_selection import GridSearchCV

# estimator = XGBClassifier()
# param_grid = {"n_estimators": [1,5,10,20,40,100], "gamma": [0,1,
                ,0.5,0.9], "max_depth": [2,6
# grid = GridSearchCV(estimator, param_grid, scoring = "accuracy
                      cv= 5)
# grid.fit(x_train, y_train)
# grid.best_params
```

Out: {"gamma":0.1, "max_depth":2, "n_estimators":20}

# final Model

```
# final_model = XGBClassifier (n_estimators = 20, gamma = 0.1, max_
                                depth = 2)
# final_model . fit (x_train, y_train)

# ypredictions_test  = final_model. Predict (x_test)
# ypreditions_train  = final_model. Predict (x_train)

# print ("train accuracy:" ; accuracy_score (yprediction_train, y_train))
# print ("test accuracy:", accuracy_score (yprediction_test, y_test))
```

Out: Test accuracy : 0.99815
Train accuracy : 0.9979

```
# final_model. feature_importances_
```

Out: array ([0., 0., 0., 0.,
0., 0., 0. ..])

```
# important = pd. DataFrame ( index = X. columns , data = final_
model . feature_importances_ , columns = ['importance_
Features'])
```

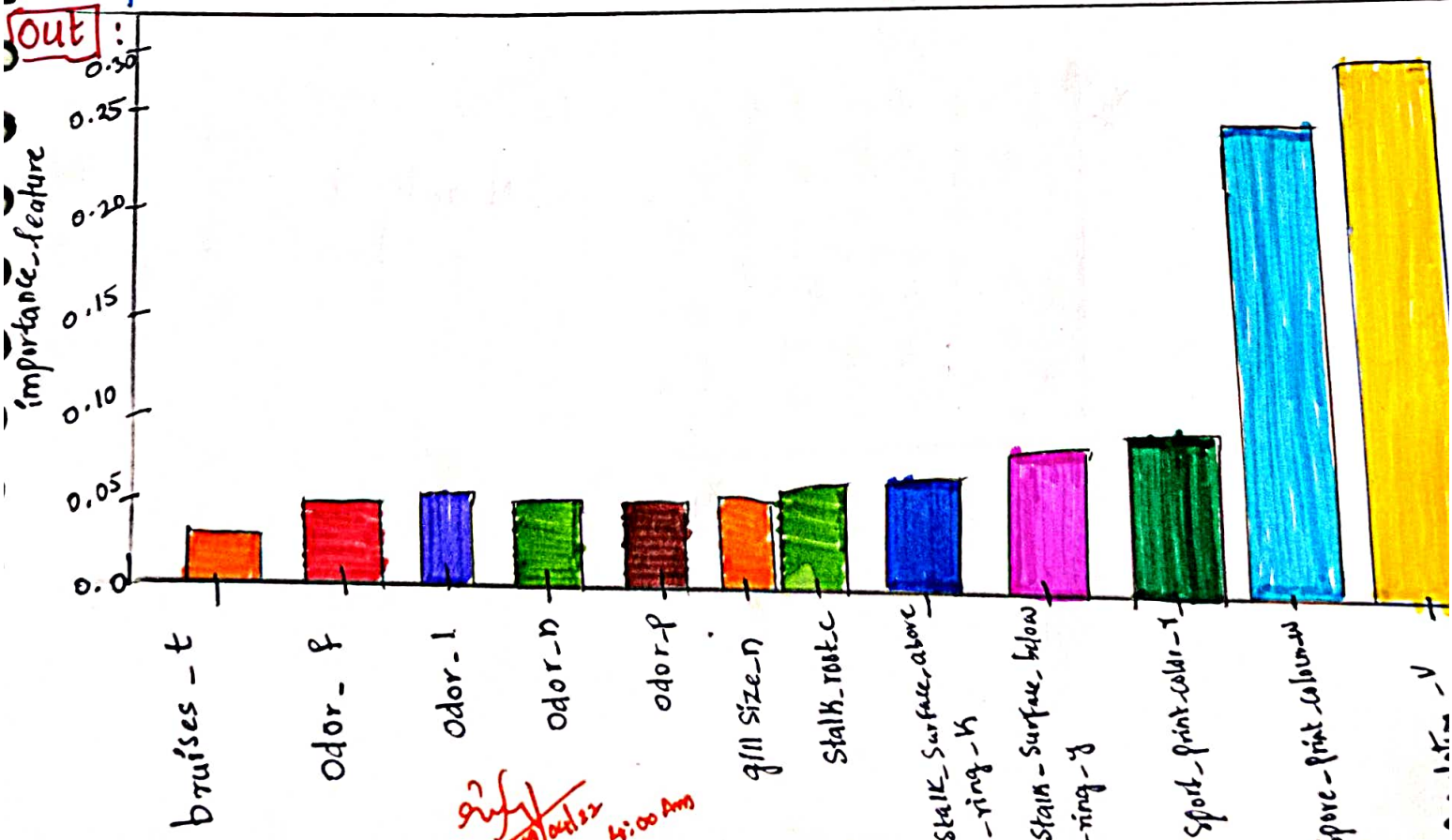
```
# Jacks = important[important["importance_feature"] > 0.01]

# Jacks.sort_values("importance_feature")
```

Out:

| | important_features |
|---|---|
| odor-p | 0.015831 |
| spore-print-color-w | 0.020152 |
| ⋮ | ⋮ |
| odor-n | 0.290767 |

```
# plt.figure(fig_size=(14,6), dpi=200)
# sns.barplot(data = Jacks.sort_values("importance_feature"),
                x = Jacks.index, y= "importance_feature")

# plt.xticks(rotation=90)
# plt.show()
```

Out:



Dt:- 18/4/22
2:30 pm

* Supervised Algorithm :-
  it has Label Data

* Predicting :- Discrete (categories) | class (Yes/No)
  * Binary classification [0, 1]
  * Multi classe | classification [drug x, drug, drug
  Ex :-

#Classification [Algorithms]

→ Logistic Regression
→ KNN [K-Nearest Neighbours]
→ Decision Tree
→ Random Forest
→ Ada Boost
→ Gradient Boost
→ XG Boost
→ SVM [Support Vector Machine]
→ Navies Bayes

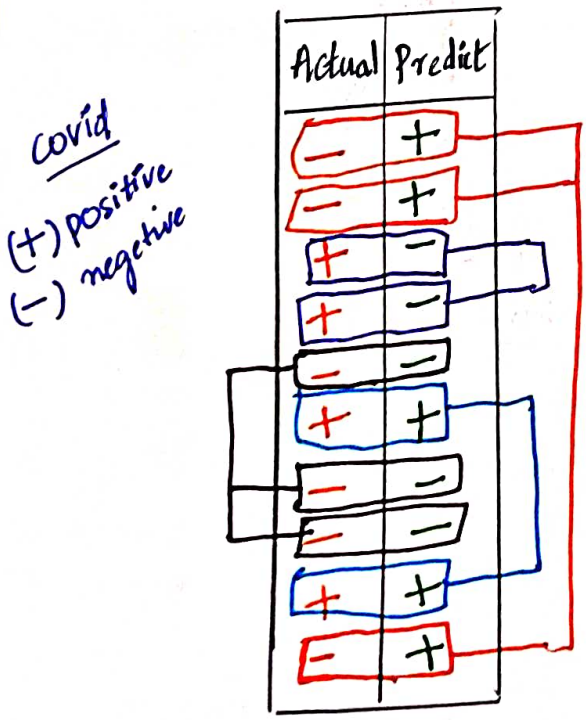
## ⇒ Classification Use Cases :

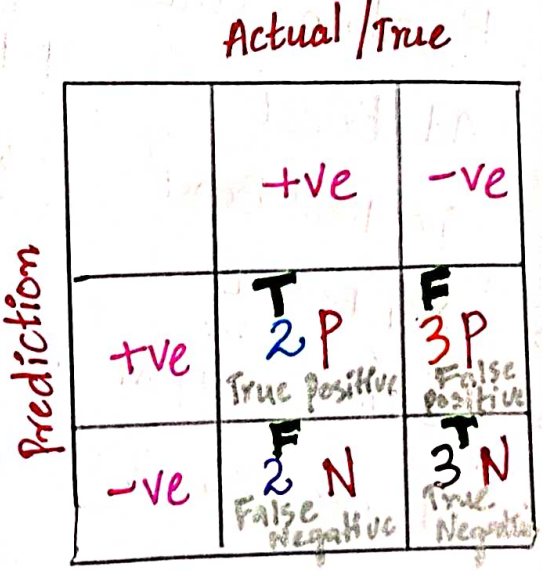
1. Which **Category** a customer belongs to ?
2. Whether Customer **Switches** to another provider / brand ?
3. Whether customer **responds** to a particular advertising Campaign ?

## ⇒ Evaluation Metrics :

### ⇒ Confusion Matrix

Ex:- Whether The person is covid: (+) positive
(-) negative.




⇒ T/F - Correct Prediction | incorrect Prediction

⇒ P/N - Predicted Positive | Predicted Negative

* Another Type

| Actual \ Predict | +ve | -ve |
|---|---|---|
| +ve | TP | FN |
| -ve | FP | TN |

| Actual \ Predict | +ve | -ve |
|---|---|---|
| +ve | 2 [TP] | 2 [FN] |
| -ve | 3 [FP] | 3 [TN] |

* Classification [Accuracy] = $\frac{\text{Correct Predictions}}{\text{total predictions}} = \frac{5}{10}$

* Mis-classification [Error] = $\frac{\text{Incorrect prediction}}{\text{total predictions}} = \frac{5}{10}$

Que :- Which is more important "False positive", "False Negat

Ans :- it is dependent on "Bussiness problem" → FN ↓
FP ↓
Error ↓
Accuray ↑

⇒ Person is Having covid

Actual +ve

Predicted -ve

Always Predicted Comes last (Second Letter)

[FN] More important
Because. if person is Having covid,
You predicted negative, Problem: (i) die
(ii) Spread

F N

⇒ Person is not having covid

Actual -ve

Predicted +ve

[FP]

F P

If person is not having covid,
You predicted covid, (i) quarantine
(ii) Antibotics (medicins)

[FP] More important
But, Person is Havin
cancer, But predict
positive.
(i) Kimotherapy

Que :- In bussiness problem, They said to Focus of "FN" shou

be low?

If client, wants Accuracy and FN

↓ Best

| | Actual | |
|---|---|---|
| Pred | 1 TP | 1 FP |
| | 8 FN | TN 90 |

Accuracy = 91 %.

FN = 8

| | Actual | |
|---|---|---|
| Pred | 4 TP | 6 FP |
| | 3 FN | 87 TN |

Accuracy = 91 %.

False Negative = 3

If client, wants only FN should be low, Not Accuracy

| | Actual | |
|---|---|---|
| Pred | 38 TP | 24 FP |
| | 0 FN | 38 TN |

Accuracy = 76 %

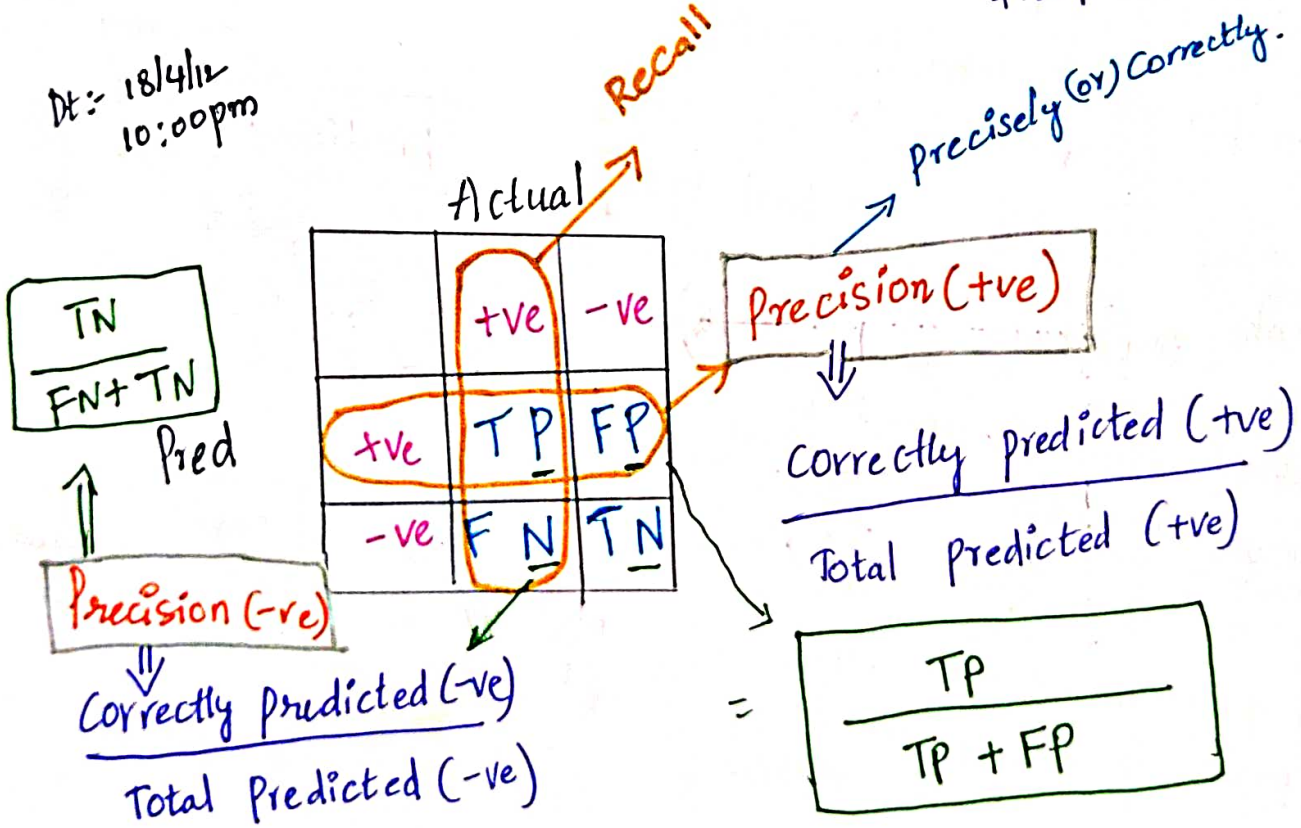
FN = 0

← best

18/4/22

4:30pm.

Dt :- 18/4/22
10:00pm



2) Recall (+ve) = Correctly Predicted (+ve) / total Actual Positive (+ve)

$$= \frac{TP}{TP + FN}$$

* Recall (-ve) = correctly predicted (-ve) / total Actual negative (-ve)

$$\frac{TN}{FP + TN}$$

** Que :- What is difference between Recall and Precision ?

Ans :- Precision :- Comparing with The Predicted Values

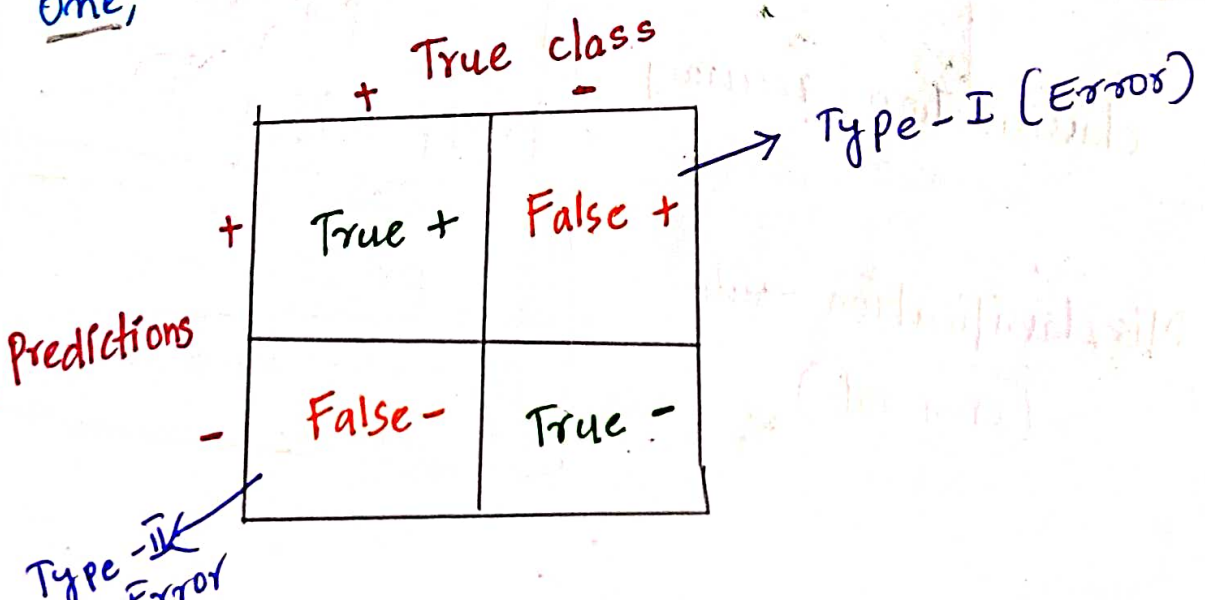
Recall :- Comparing with Actual values.

| Preci | Recall |
|---|---|
| Comparing with Predicted | Actual |

* denominator changes, where numerator is same

* denominator instead of prediction, Take actual

One, instead of predicted Positive, Take actual positive



* True positive [TP] :- when classifer predicted True and correct class was True
Ex: They have disease, predicted correctly.

* True Negative [TN] : when model predicted False and correct class was False
Ex:- No disease, But predictive not have disease.

* False positive [FP] [Type-I Error]
classifier predicted True, but correct class was False.
Ex:- Patient did not have disease.

* False Negatives [FN] [Type-II Error]
classifier predicted False, but they actually do have disease.
Ex:- Patient do not have disease.

** Key Performance Indicators [KPI]

# classification accuracy = $\frac{TP + TN}{(TP + TN + FP + FN)}$

# Misclassification rate (error rate) = $\frac{FP + FN}{(TP + TN + FP + FN)}$

Example

| Pred \ Actual | + | − |
|---|---|---|
| + | TP = 1 | FP = 1 |
| − | FN = 8 | TN = 90 |

# classification accuracy = $\frac{TP + TN}{TP + FP + FN + TN}$ = 91 %

# precision (+ve) = $\frac{TP}{TP + FP}$ = $\frac{1}{2}$ = 50 %.

# Recall (+ve) = $\frac{TP}{TP + FN}$ = $\frac{1}{1+8}$ = $\frac{1}{9}$ = 11 %.

# precision (-ve) = $\frac{TN}{FN + TN}$ = $\frac{90}{8+98}$ = $\frac{90}{98}$ = 0.91 %.

# Recall (-ve) = $\frac{TN}{FP + TN}$ = $\frac{90}{1+90}$ = $\frac{90}{91}$ = 0.98 %.

* TPR (recall) = True Positive Rate = $\frac{TP}{TP + FN}$ ⇒ Actual (+ve)

* FPR = False Positive Rate = $\frac{FP}{TP + FN}$ ⇒ Actual (+ve)

* TNR = True Negative Rate = $\frac{TN}{TN + FP}$ ⇒ Actual (-ve)

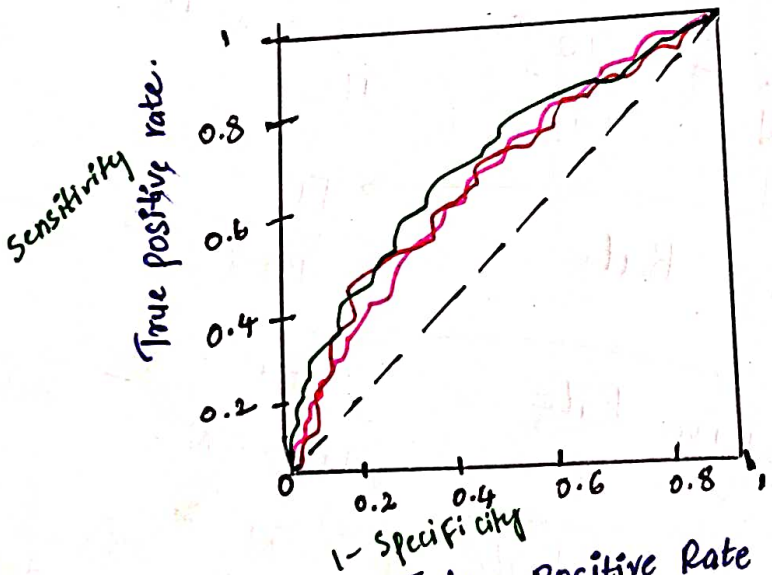
* FNR = False Negative Rate = $\frac{FN}{FP + TN}$ ⇒ Actual (-ve)

* Specificity = TNR ⇒ 1 - FPR

= True Negative Rate = $\frac{\text{True Negatives}}{\text{True negatives + False positives}}$

* Sensitivity ⇒ TPR ⇒ Recall ⇒ $\frac{TP}{TP+FN}$

# ROC [Reciver operating characteristic Curve]




False Positive Rate = 1 - specifity = 1 - TNR

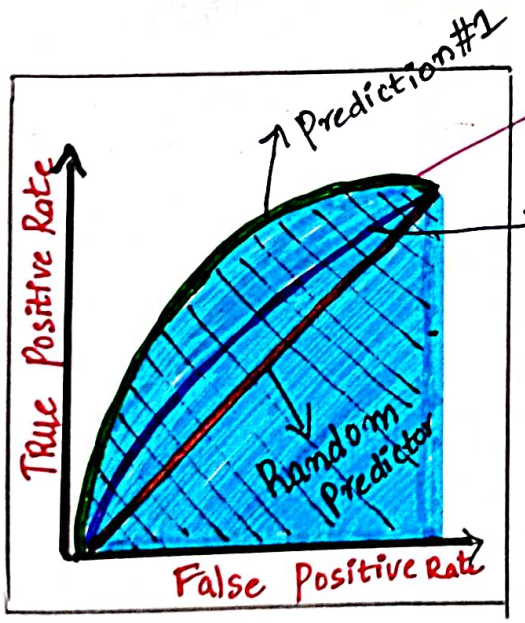
* ROC curve is a metrics that asses model ability to distinguish between binary (0 (or) 1) classes

* Roc curve is created by plotting the true positive rate (TPR) against the False positive Rate [FPR] at various threshold settings

* True positive rate is also known as Sensitivity, recall (or) Probability of detection in Machine Learning.

* False positive rate is also known as the probability of False alarm and can be calculated as [1 - specificity]

# AUC [Area Under Curve]




# the light Blue area represents area under Curve of receiver operating characteristic [AUROC]

# Higher the AUC, the better the model is at predicting

18/04/22
11:30 pm

5/4/22
11:00 AM

## 2 ⇒ Feature Scaling ⟹ Applicable only For Continous data

Feature Scaling ?

Ans:- it refers (or) Techniques used to normalize the ranges of independent Variables (input Variables) in our data. (or)

* The methods to set the Features value range within a similar Scale. (same scale)

* Variables with bigger magnitude / larger value range dominate over those with smaller magnitude / value range.

Ex:- 10,00,00 , 10 (x)
If we take
10 ——— 10 → Both are Equal Import

* Scale of the features is an important consideration when building Machine Learning models

* Feature scaling is generally the last step in the data pre processing pipeline, PerFormed Just Before Training the machine Learning algorithms.

5. Gradient descent converges Faster when Features are on Similar scales

# * Feature Scaling *

Example :-

| C.C | Milage |
|---|---|
| 1600 | 14 |
| 1800 | 15 |
| 2200 | 16.5 |
| 2100 | 18 |
| 2600 | 22.5 |
| 1800 | 19.4 |
| 1900 | 25.4 |

of a car

Continous Data

Q. When we ask Machine which is important In The both columns ?

* Machine is Giving Importance To C.C Because it is Having Large Values.

* But, We know both are importa

So, we have train model in Initially such that, it should consider "Both The columns Similar

So, we Reduce The Value with 200

$= \frac{800}{1000} = \frac{80}{100} = \frac{8}{10} = \frac{4}{5}$ (Every one is Same)

with 100

We are scaling down Dividing Every Value with "10" In this case.

* To Reduce The Value.

* Feature Scaling Importance in Some ML Algorithms

1. Linear Regression & logistic Regression

* The regression coefficients of linear models are Directly influenced by Scale of The Variable.

2. Support Vector Machines :-

* Feature scaling helps Decrease The Time to Find Support Vectors For SVM's

3. K-means clustering

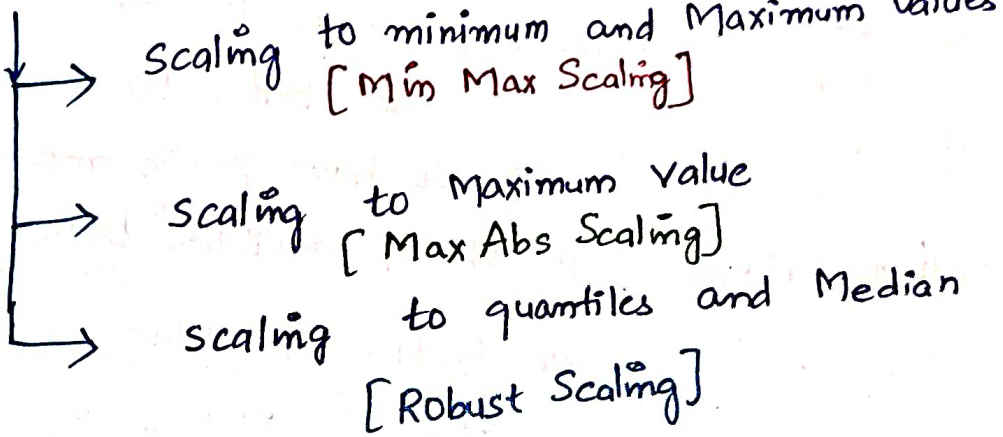
* Euclidean distances are Sensitive to Feature magnitude

4. Principal component analysis (PCA) :-

* PCA requires The Features to be Centered at "0".

* Various Feature Scaling Techniques :-

1. Standardisation
2. Normalisation Technique




```
# import pandas as pd
# import matplotlib.pyplot as plt
  %. matplotlib inline.
```

→ Using (titanic Data set)

code:

```
# titanic = pd.read_csv("titanic.csv", usecols=["Age"
  titanic.head()
```

Out

| Age |
|---|
| 22.0 |
| 38.0 |
| 26.0 |
| 35.0 |
| 35.0 |

# check Null values in "Age" column.

```
# titanic.isnull().sum()
```

Out  Age  177 → Null values

# Replacing with "Median Value" For Null values

```
# titanic["Age"].fillna(titanic["Age"].median(),
                        inplace = True)
# titanic["Age"].isnull().sum()
```

Out :  Age  0 → Zero Null values

---

I. **Standardisation**

↓

Converting Each Value To Zscore

Ex :-

| X | $\frac{x-\mu}{\sigma}$ = Zscore |
|---|---|
| 23 | $\frac{23-60}{2}$ = -18.5 |
| 45 | $\frac{45-60}{2}$ = -7.5 |
| 67 | $\frac{67-60}{2}$ = 3.5 |
| 89 | $\frac{89-60}{2}$ = 14.5 |
| 78 | $\frac{78-60}{2}$ = 9 |

These Five values converting To Zscores is called as "Standard" Scaling

Avg/mean = 60.

(Std) $\sigma = \underset{Exp}{2}$

Code :-

```
From sklearn.Preprocessing import StandardScaler (SC)

# sc = StandardScaler ()        # call The Function (Short Cut)
                  ↓ main (don't Forget)
# titanic ["Age_sc"] = sc. fit_tranform (titanic [["Age"]
```

# creating new column

# fit_transform → calculation → Converting The value → stores in

```
# titanic ["Age_sc"]
```

Out :-

| | Age_sc |
|---|---|
| 0 | -0.5657 |
| 1 | 0.6638 |
| 2 | -0.2583 |
| 3 | 0.43312 |
| ⋮ | ⋮ |
| 890 | 0.20272 |

# Here Every value is Converted To Zscore.

$$\left[\frac{X-\mu}{\sigma}\right] = Z\ score.$$

Ex :- Age [22] # First record X = 22

```
# titanic.mean()
μ = 29.361
# titanic.std ()
σ = 13.01
```

$$\left[\frac{X-\mu}{\sigma}\right] = \left[\frac{22-29.361}{13.01}\right]$$

"Age_sc" → [-0.5657]

```
# titanic
```

Out

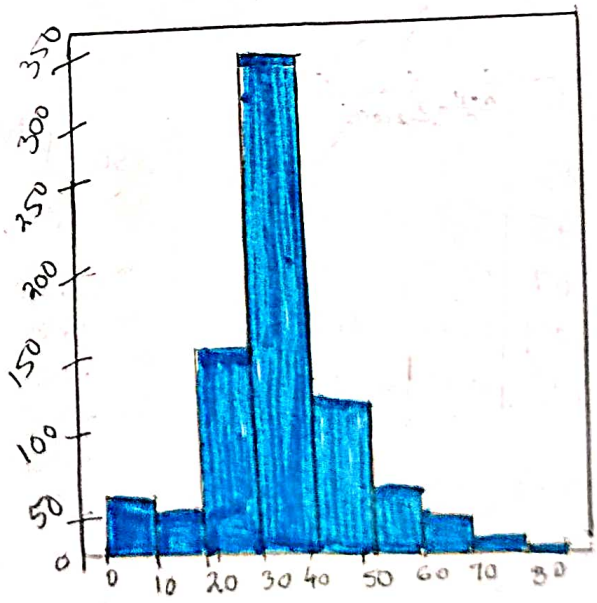
| | Age | Age_sc |
|---|---|---|
| 0 | 22.0 | -0.5657 |
| 1 | 38.0 | -0.6638 |
| 2 | 25.0 | -0.2583 |
| ⋮ | ⋮ | ⋮ |
| 889 | 26.0 | -0.2583 |
| 890 | 32.0 | 0.20272 |

Original Data

Converted To Z_Score

# histogram of Original ["Age"]

```
# titanic ["Age"]. hist()
plt. show
```

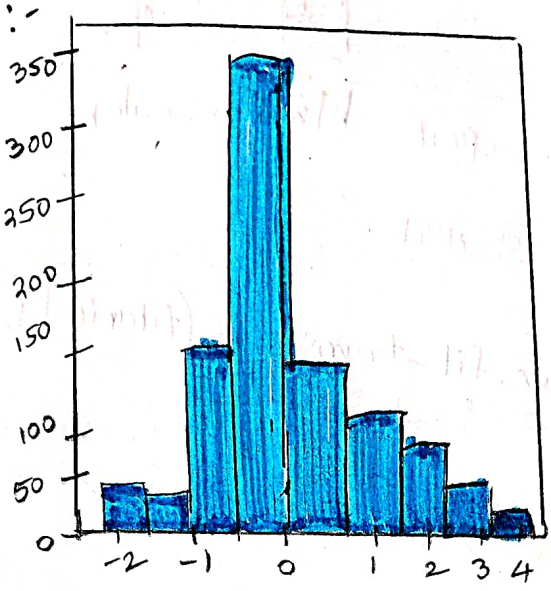


# Histogram of Converted "Age"

# titanic["Age_sc"].hist()

# plt.show()

Out :-




# Here, we see, Histogram is not going to change, But, Values does.

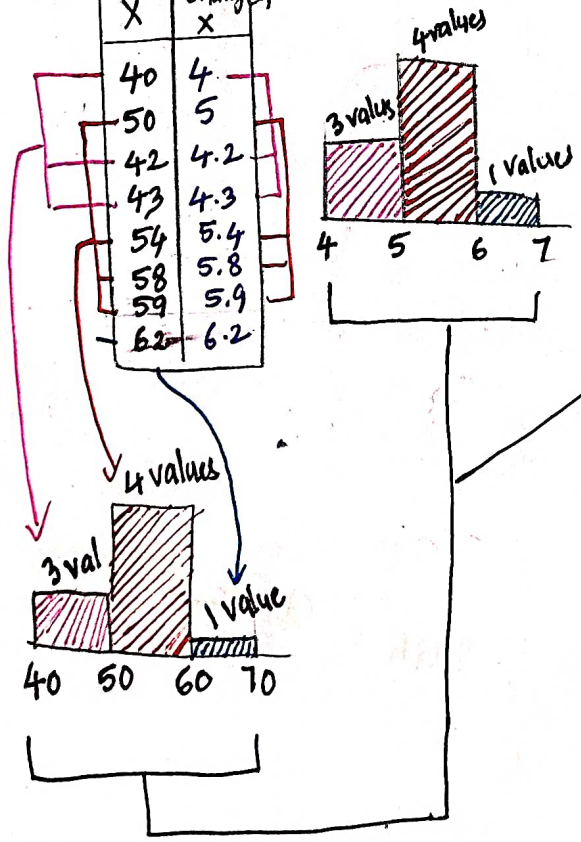
⇒ Example :-

* How ? Histogram Remains Same, But changed Values

Ans :-

Ex :-

| X | Changed x |
|---|---|
| 40 | 4 |
| 50 | 5 |
| 42 | 4.2 |
| 43 | 4.3 |
| 54 | 5.4 |
| 58 | 5.8 |
| 59 | 5.9 |
| 62 | 6.2 |




# Here only "x" values Only changes But Histogram Remains Same.

# II # Normalisation

## * MinMax Scaling

⇒ MinMax Scaling Scales, The Values between "0" to "1" (Every Thing is (+ve) value No -ve value)

Ex:-

| X | |
|---|---|
| 65 | $\frac{65-34}{53}$ |
| 45 | $\frac{45-34}{53}$ |
| 34 ← min. value | $\frac{34-34}{53}=0$ |
| 67 | $\frac{67-34}{53}$ |
| 67 | |
| 87 ← Max. value | $\frac{87-34}{53}=1$ |

$$\frac{X - X_{min}}{X_{max} - X_{min}}$$

MinMax Scaler

Code: From sklearn.Preprocessing import

```
# min_max = MinMaxScaler()  # Shortcut
# titanic ["Age_mm"] = min_max.fit_transform(titanic [["Age"]])
```

fit_transform

creating new column

```
# titanic   (calling/Show Data)
```

Out :-

| | Age | Age_SC | Age_mm |
|---|---|---|---|
| 0 | 22.0 | -0.5653 | 0.2711 |
| 1 | 38.0 | 0.4337 | 0.4722 |
| 2 | 26.0 | -0.1858 | 0.322 |
| ⋮ | ⋮ | ⋮ | ⋮ |
| 888 | 26.0 | -0.25 | 0.32 |
| 890 | 32.0 | 0.20 | 0.39 |

## * Robust Scaling

$$\frac{X - X_{median}}{IQR}$$

$\therefore IQR = Q_3 - Q_1$

Code:

```
From sklearn.preprocessing import RobustScaler

# rs = RobustScaler()  → shortcut()
# titanic["Age_rs"] = rs.fit_transform(titanic[["Age"]])
```

titanic["Age_rs"] (or) = RobustScaler().fit_transform(...)

If not using any shortcut like rs = RobustScaler()

```
# titanic
```

Out

| Age | Age_SC | Age_Mm | Age_RS |
|---|---|---|---|
| 22.0 | 0.565 | 0.461 | 0.4615 |
| 38.0 | 0.66 | 0.769 | 0.769 |
| 26.0 | 0.25 | -0.153 | -0.153 |
| ⋮ | | | |
| 26.0 | -0.25 | 0.153 | 0.153 |
| 32.0 | 0.2 | 0.396 | 0.3076 |

* **Max Abs Scaling**

$$\frac{X}{X_{max}}$$

Ex:-

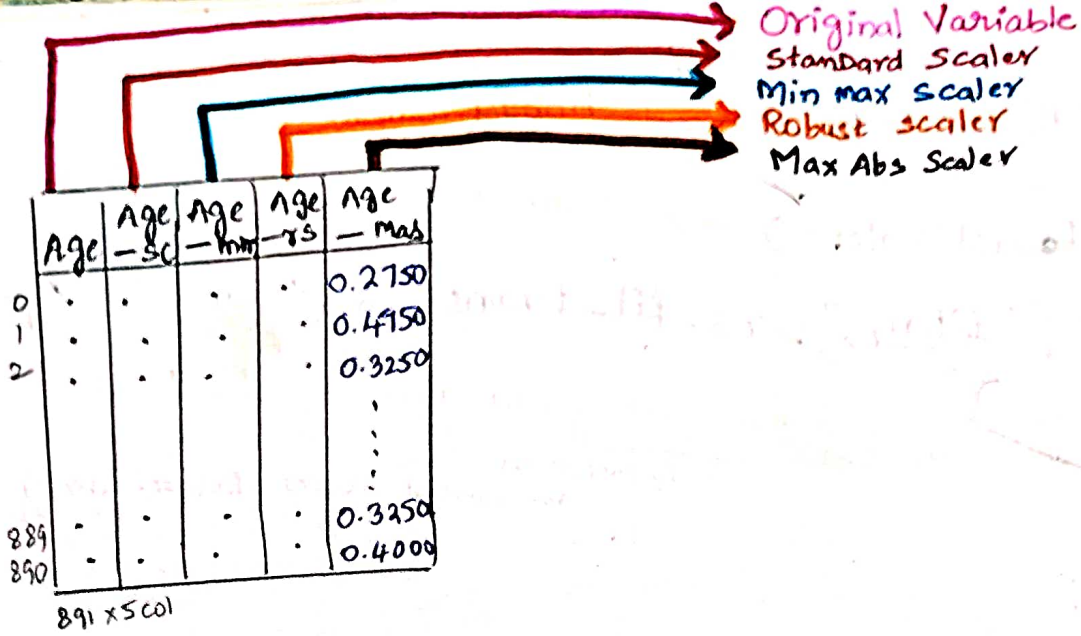
| X | |
|---|---|
| 68 | 68/80 |
| 44 | 44/80 |
| 52 | 52/80 |
| 69 | 69/80 |
| 80 | 80/80 = 1 |

Max value

code

```
From sklearn.preprocessing import MaxAbsScaler

# mas = MaxAbsScaler()

# titanic["Age_mas"] = mas.fit_transform(titanic[["Age"]]

# titanic
```

Out :-




5/04/22
4:58 pm.

Example :- Discretization

Continous Data ⇒ Milage of a Car (Discrete, categorical) Data

12.5
22.8
15.2
18.7
12.4
15
23.3

< 15 → Poor milage
15-20 → Avg. milage
20-25 → good milage
> 25 → Very good

→ By our own choice To Divide the Data into bins/parts