Machine Learning Full Course with AI

Google Microsoft ChatGPT

From Beginner to Advanced

Python for data analyst | day 01 | 30 days free course | beginners to...

923K views • Streamed 2 years ago

MS Excel Full Course in Hindi | Basic to Advanced | Learn Excel in Just ...

2.3M views • 1 year ago

AI Complete Crash Course for Beginners in Hindi | Learn Artificial...

1.2M views • 11 months ago

Artificial Intelligence Full Course (Free) | Master AI Tools & Core...

789K views • 3 months ago

Learn AI in 20 Minutes for Beginners (2026 Guide)

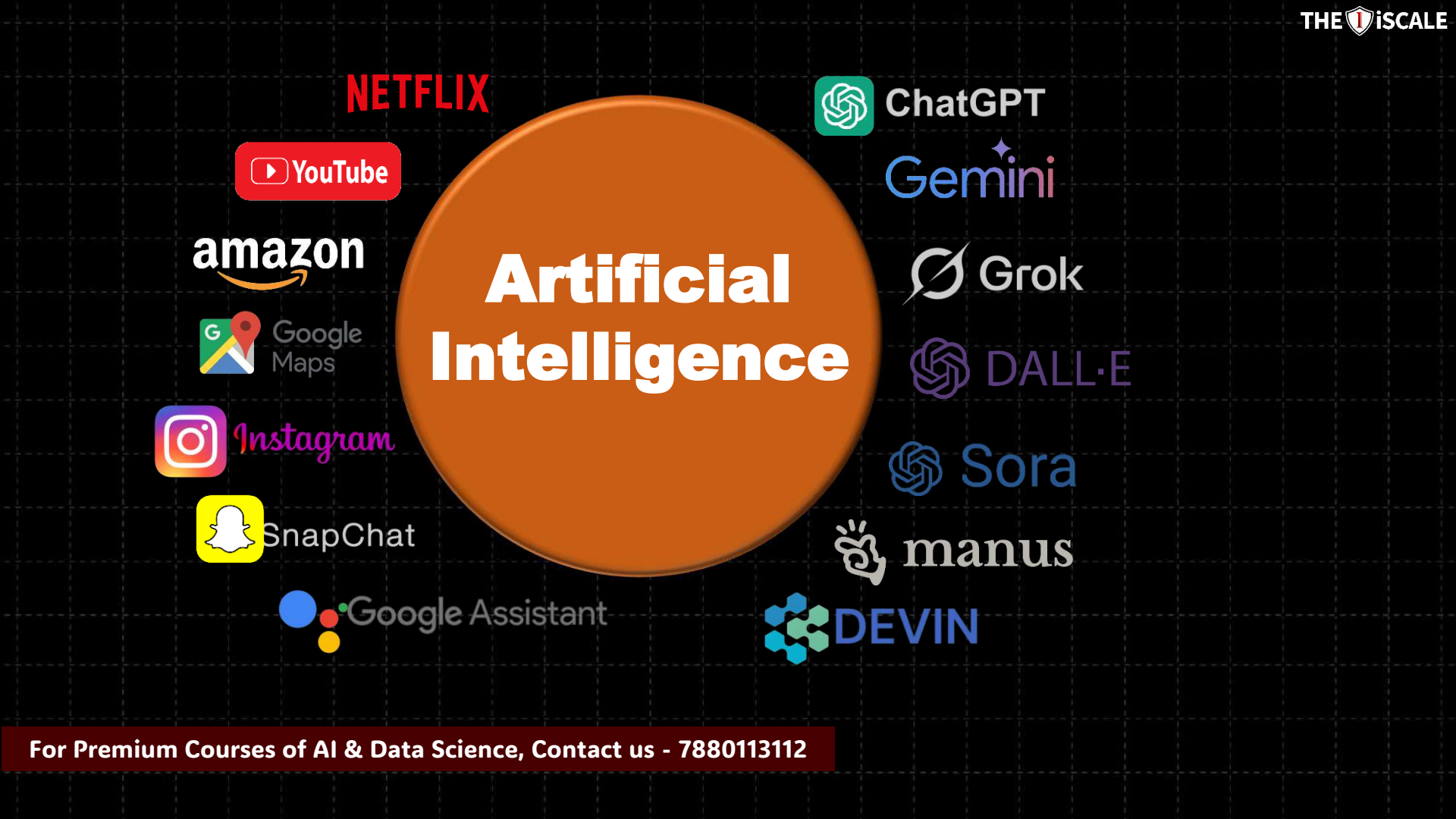
400K views • 2 months ago

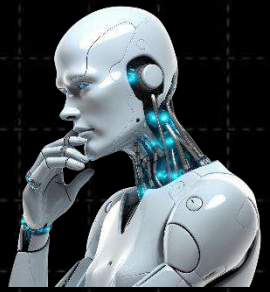
Topics To Be Covered

- Mathematical Foundations
- Linear Algebra (Matrices, Vectors, Eigenvalues)
- Calculus (Derivatives, Gradients)
- Probability (Bayes' Theorem, Distributions)
- Descriptive & Inferential Statistics
- Data Engineering & Preprocessing
- Data Imputation (Mean, Median, Mode, KNN Imputer)
- Outlier Detection (Z-Score, IQR Method)
- Feature Transformation (Log Transform, Box-Cox)
- Feature Scaling (Min-Max Scaling, Standard Scaler)
- Handling Imbalanced Data (SMOTE, Undersampling)
- Dimensionality Reduction (PCA, LDA, t-SNE)
- Supervised Learning: Regression
- Simple Linear Regression
- Multiple Linear Regression
- Polynomial Regression
- Support Vector Regression (SVR)
- Decision Tree Regression
- Random Forest Regression
- Regularization (L1 Lasso, L2 Ridge, ElasticNet)
- Supervised Learning: Classification
- Logistic Regression
- K-Nearest Neighbors (KNN)
- Support Vector Machines (SVM)
- Kernel SVM

Topics To Be Covered

- Naive Bayes (Gaussian, Multinomial, Bernoulli)
- Decision Tree Classification
- Random Forest Classification
- Unsupervised Learning
- K-Means Clustering
- K-Means++
- Elbow Method & Silhouette Score
- Hierarchical Clustering (Dendrograms)
- DBSCAN (Density-Based Clustering)
- Gaussian Mixture Models (GMM)
- Anomaly Detection
- Association Rule Learning (Apriori, Eclat)
- Ensemble Learning & Boosting
- Voting Classifier/Regressor
- Bagging (Bootstrap Aggregating)
- Pasting
- Random patches and Random subspaces
- Random patches and Random subspaces
- Gradient Boosting (GBM)
- XGBoost (Extreme Gradient Boosting)
- LightGBM
- CatBoost
- AdaBoost
- Stacking & Blending
- Model Selection & Evaluation
- K-Fold Cross Validation
- Stratified K-Fold
- Hyperparameter Tuning (GridSearchCV, RandomizedSearchCV)
- Confusion Matrix (TP, TN, FP, FN)
- Accuracy, Precision, Recall,





Machine Learning

Full Course

Beginner to Advanced with AI

Google Microsoft ChatGPT NETFLIX amazon

Artificial Intelligence

Machine Learning

Google Photos

OLA

ZERODHA

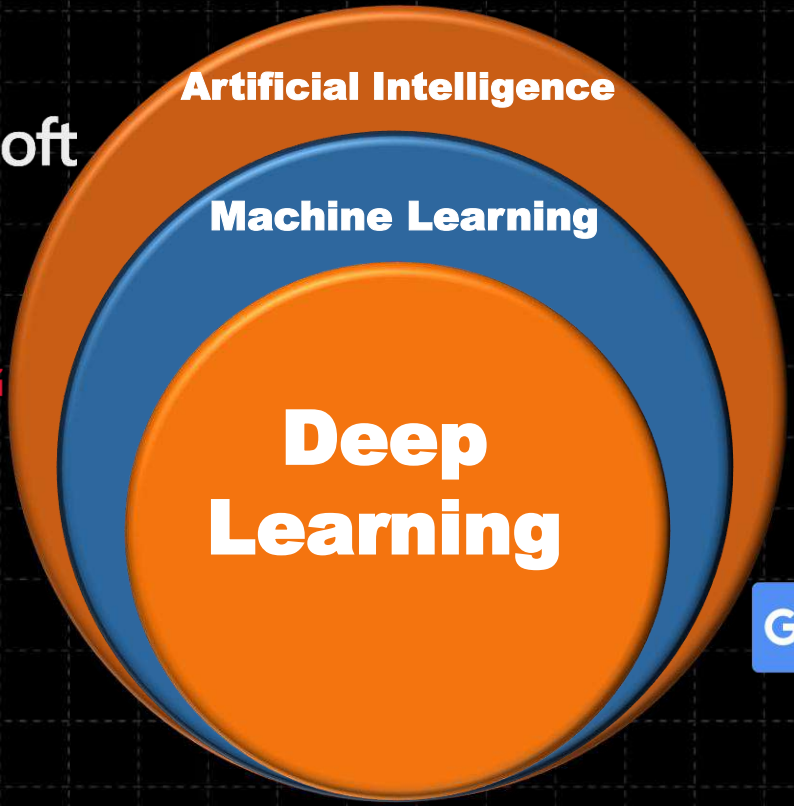
Spotify

OCR Microsoft

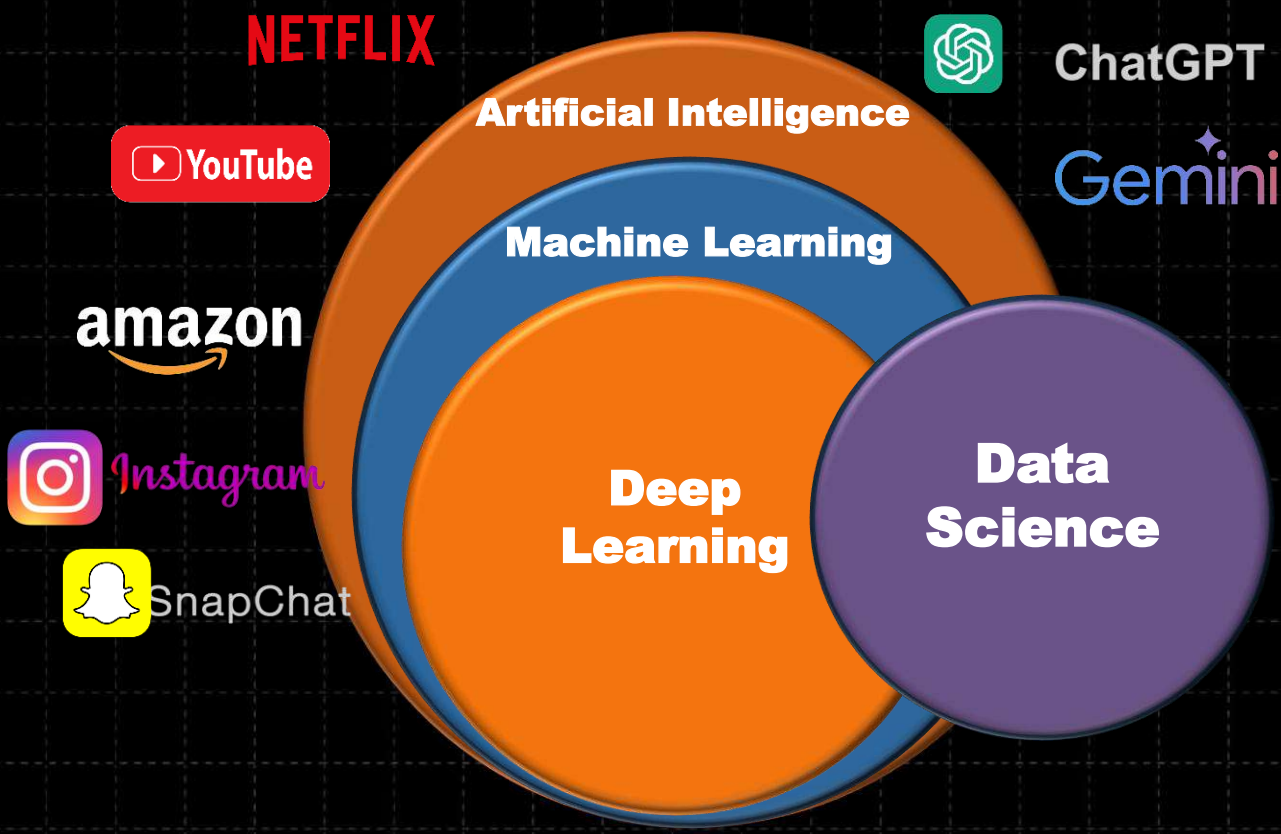
Mimic Human Brain

TESLA

alexa



Google Translate



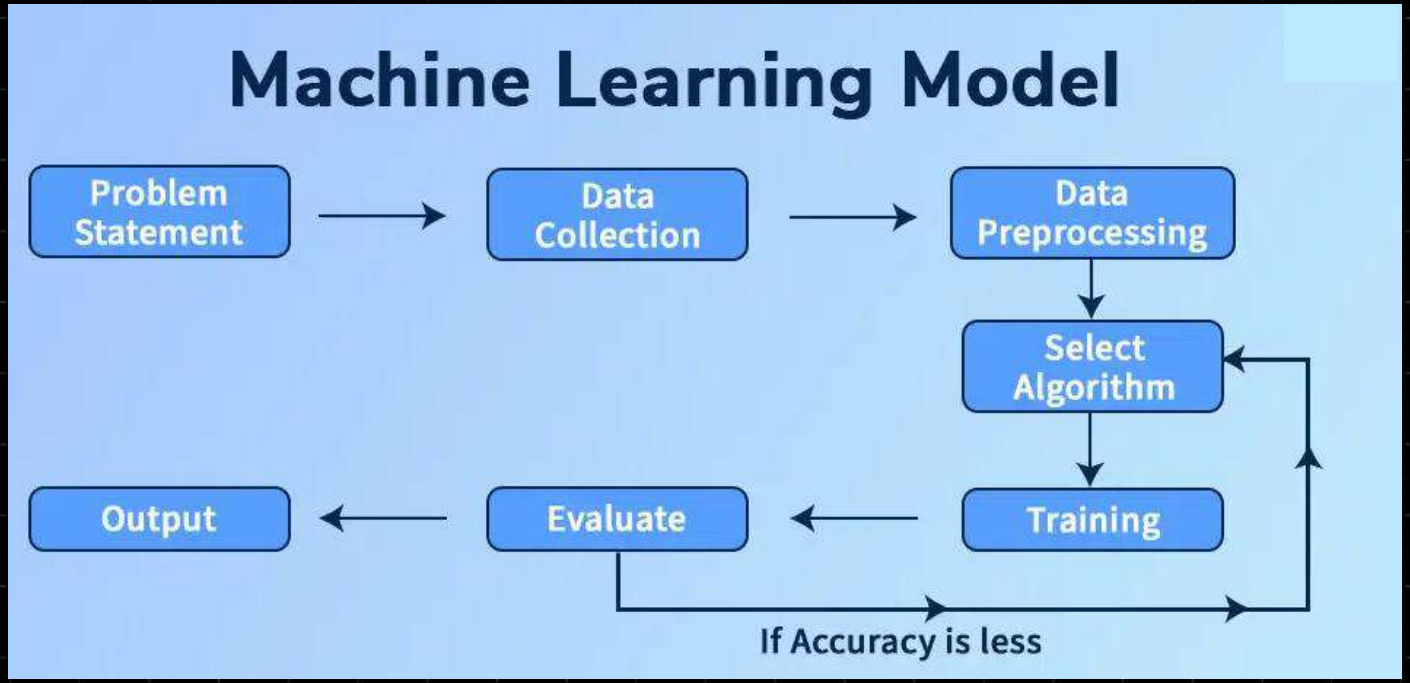
Data is the DNA of AI

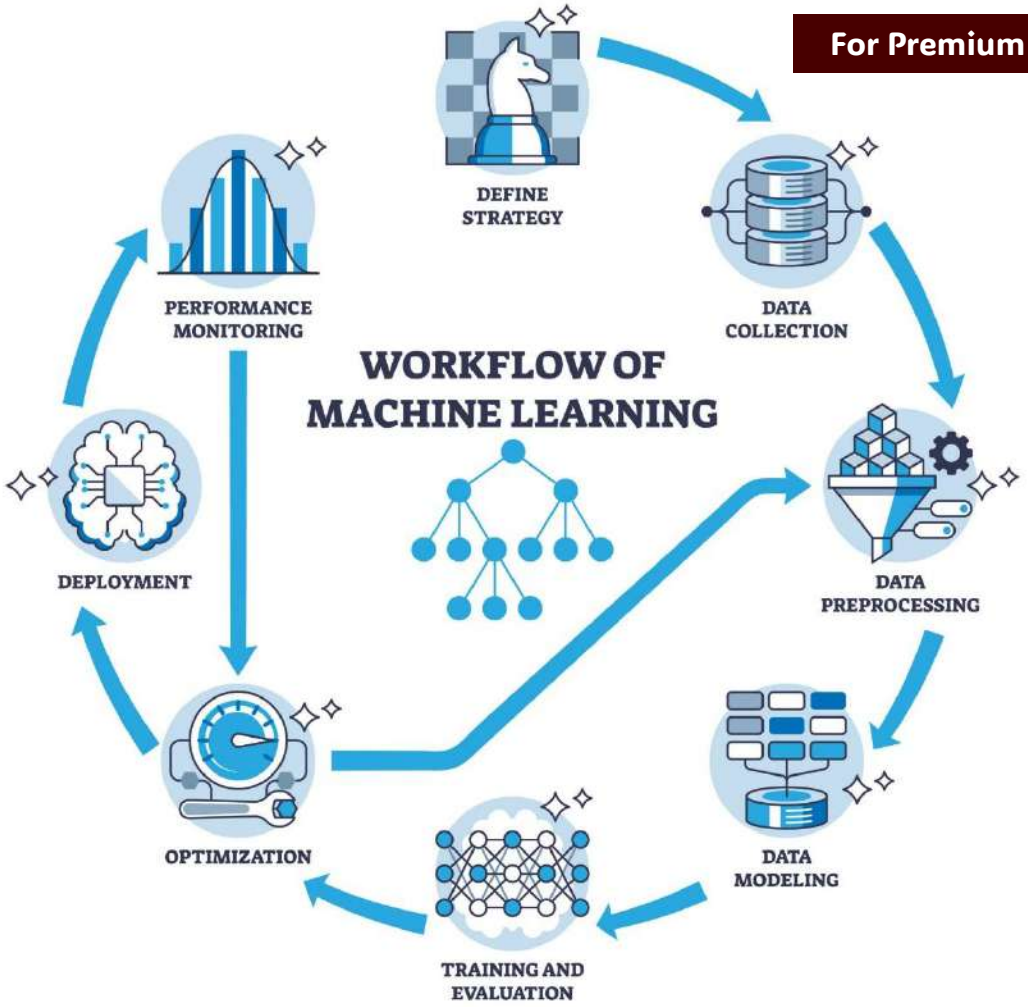
A2 Sales

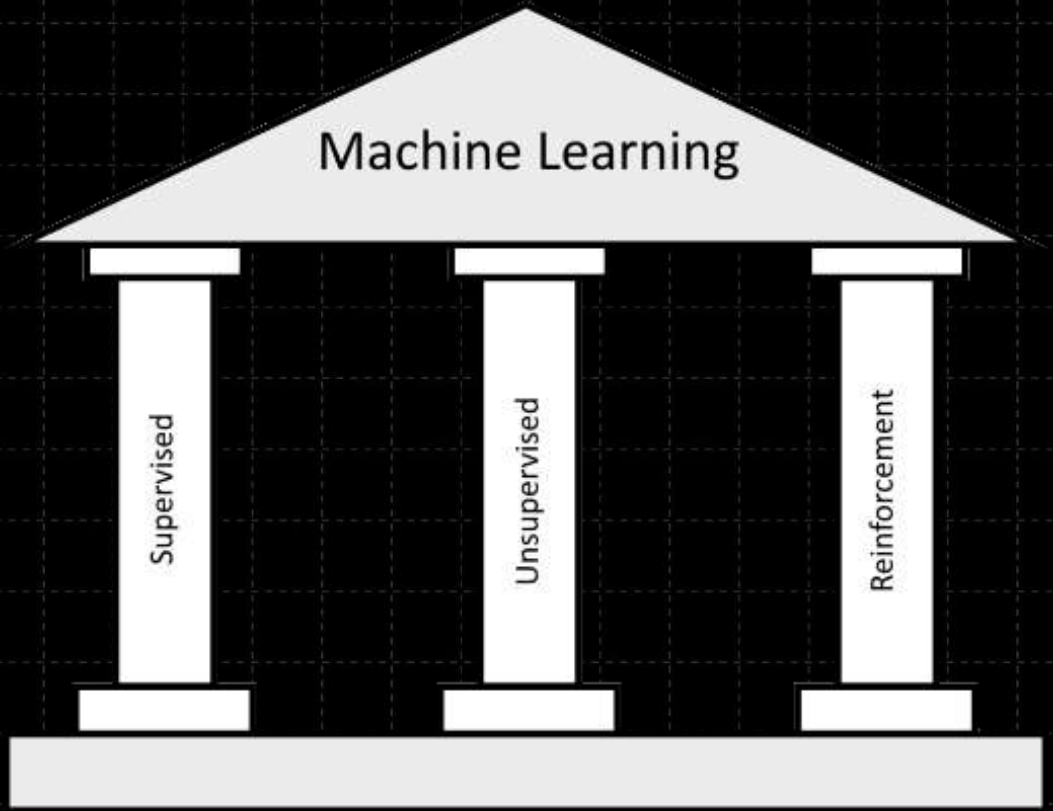
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Account	Businees Unit	Currency	Year	Scenario	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
272	Sales	Software	USD	2022	Actuals	$78,338,286	$58,034,151	$75,874,351	$87,725,417	$84,878,696	$73,990,027	$95,372,852	$83,104,256	$52,883,151	$61,528,451	$91,089,451	$87,746,916
273	Cost of Goods Sold	Software	USD	2022	Actuals	($32,377,742)	($26,379,976)	($33,823,729)	($41,186,038)	($40,605,218)	($31,555,193)	($46,768,008)	($33,951,735)	($24,520,783)	($24,743,822)	($36,760,421)	($43,815,665)
274	Commissions Expense	Software	USD	2022	Actuals	($3,477,740)	($2,612,821)	($3,043,000)	($4,206,639)	($3,635,596)	($3,390,988)	($4,751,711)	($3,385,162)	($2,396,080)	($2,929,053)	($4,196,677)	($3,591,506)
275	Payroll Expense	Software	USD	2022	Actuals	($8,777,061)	($5,956,476)	($8,788,853)	($9,995,832)	($8,550,645)	($8,956,466)	($11,183,485)	($8,871,134)	($6,351,611)	($7,670,731)	($9,827,555)	($9,599,585)
276	Travel & Entertainment Expense	Software	USD	2022	Actuals	($851,217)	($648,182)	($827,626)	($971,146)	($1,009,775)	($865,150)	($1,184,627)	($992,651)	($601,399)	($727,376)	($1,128,223)	($1,000,127)
277	R&D Expense	Software	USD	2022	Actuals	($3,158,339)	($2,728,823)	($3,421,161)	($4,332,736)	($4,162,923)	($3,139,829)	($4,315,056)	($3,425,707)	($2,305,237)	($2,579,902)	($4,086,474)	($4,146,068)
278	Consulting Expense	Software	USD	2022	Actuals	($4,069,215)	($3,519,759)	($4,009,533)	($4,836,632)	($5,257,809)	($4,412,376)	($4,985,773)	($4,979,583)	($2,840,712)	($3,475,369)	($4,584,362)	($4,474,219)
279	Software/Hardware Expense	Software	USD	2022	Actuals	($6,585,349)	($4,759,687)	($6,493,370)	($6,546,763)	($6,734,243)	($6,212,732)	($8,195,799)	($6,417,083)	($4,176,227)	($5,193,726)	($6,901,697)	($7,639,360)
280	Marketing Expense	Software	USD	2022	Actuals	($1,828,234)	($1,342,350)	($1,751,692)	($2,175,232)	($1,829,981)	($1,789,035)	($1,954,312)	($2,007,974)	($1,094,268)	($1,521,032)	($1,827,984)	($1,900,742)
281	Sales	Advertising	USD	2022	Actuals	$19,584,572	$14,508,538	$17,451,101	$22,808,608	$21,219,674	$19,977,307	$21,935,756	$24,931,277	$15,336,114	$13,536,259	$26,415,941	$20,181,791
282	Cost of Goods Sold	Advertising	USD	2022	Actuals	($8,362,230)	($6,091,621)	($8,316,059)	($10,159,328)	($8,624,806)	($9,693,159)	($9,579,056)	($9,973,478)	($6,228,310)	($6,023,754)	($10,593,290)	($8,974,298)
283	Commissions Expense	Advertising	USD	2022	Actuals	($915,086)	($612,941)	($823,113)	($916,944)	($880,691)	($867,181)	($960,417)	($1,047,426)	($623,688)	($675,831)	($1,132,922)	($970,183)
284	Payroll Expense	Advertising	USD	2022	Actuals	($2,065,945)	($1,474,640)	($2,117,096)	($2,790,604)	($2,540,908)	($2,325,558)	($2,387,390)	($3,056,599)	($1,798,103)	($1,677,834)	($3,256,778)	($2,424,353)
285	Travel & Entertainment Expense	Advertising	USD	2022	Actuals	($240,447)	($157,072)	($183,016)	($241,630)	($239,007)	($238,586)	($266,862)	($274,476)	($188,419)	($144,601)	($275,592)	($214,592)
286	R&D Expense	Advertising	USD	2022	Actuals	($796,397)	($610,923)	($729,783)	($913,075)	($1,012,763)	($882,471)	($1,093,311)	($1,137,735)	($686,706)	($582,618)	($1,127,220)	($876,587)
287	Consulting Expense	Advertising	USD	2022	Actuals	($1,199,233)	($811,310)	($900,254)	($1,164,976)	($1,151,854)	($1,034,607)	($1,285,049)	($1,518,350)	($866,477)	($705,238)	($1,549,744)	($1,249,787)
288	Software/Hardware Expense	Advertising	USD	2022	Actuals	($1,445,011)	($1,126,211)	($1,306,570)	($1,839,960)	($1,539,232)	($1,482,785)	($1,588,028)	($1,880,876)	($1,288,625)	($1,107,308)	($2,083,925)	($1,761,373)
289	Marketing Expense	Advertising	USD	2022	Actuals	($420,494)	($304,699)	($395,657)	($479,567)	($522,262)	($404,189)	($497,618)	($499,475)	($321,370)	($275,373)	($589,065)	($494,276)
290	Sales	Hardware	USD	2022	Actuals	$30,551,932	$19,731,611	$24,279,792	$35,090,167	$26,312,396	$25,896,509	$31,473,041	$27,424,404	$21,153,260	$24,611,380	$30,059,519	$30,711,421
291	Cost of Goods Sold	Hardware	USD	2022	Actuals	($13,819,091)	($9,286,529)	($10,933,750)	($15,827,962)	($12,543,885)	($11,801,803)	($15,410,493)	($12,281,930)	($8,667,400)	($10,416,735)	($12,874,137)	($15,312,570)
292	Commissions Expense	Hardware	USD	2022	Actuals	($1,369,961)	($942,250)	($1,067,439)	($1,630,125)	($1,266,063)	($1,202,899)	($1,418,412)	($1,366,956)	($922,278)	($1,025,165)	($1,394,203)	($1,252,426)
293	Payroll Expense	Hardware	USD	2022	Actuals	($3,344,503)	($2,300,612)	($2,595,976)	($3,778,056)	($3,167,287)	($2,947,517)	($3,546,969)	($2,816,091)	($2,552,955)	($2,506,235)	($3,473,277)	($3,338,330)
294	Travel & Entertainment Expense	Hardware	USD	2022	Actuals	($338,744)	($244,628)	($271,723)	($354,541)	($300,962)	($260,911)	($327,018)	($336,207)	($259,047)	($300,466)	($322,986)	($313,283)
295	R&D Expense	Hardware	USD	2022	Actuals	($1,468,744)	($973,887)	($1,032,242)	($1,717,271)	($1,287,760)	($1,206,283)	($1,470,777)	($1,246,641)	($1,025,992)	($1,104,742)	($1,379,202)	($1,486,638)
296	Consulting Expense	Hardware	USD	2022	Actuals	($1,633,292)	($1,089,178)	($1,472,323)	($2,057,777)	($1,473,412)	($1,538,962)	($1,683,014)	($1,385,482)	($1,129,867)	($1,300,291)	($1,689,964)	($1,833,600)
297	Software/Hardware Expense	Hardware	USD	2022	Actuals	($2,276,556)	($1,464,050)	($2,030,413)	($2,759,580)	($2,279,559)	($2,087,858)	($2,509,070)	($2,137,231)	($1,498,973)	($2,006,400)	($2,628,526)	($2,421,145)
298	Marketing Expense	Hardware	USD	2022	Actuals	($661,654)	($401,978)	($494,551)	($729,508)	($547,250)	($536,085)	($699,474)	($582,080)	($490,459)	($554,075)	($629,658)	($720,295)
299	Sales	Software	USD	2023	Budget	$89,862,727	$99,687,807	$99,378,570	$55,271,910	$83,758,431	$51,637,163	$76,348,472	$66,086,281	$80,081,331	$83,886,832	$95,750,606	$87,036,633
300	Cost of Goods Sold	Software	USD	2023	Budget	($39,040,130)	($39,921,367)	($45,671,498)	($25,801,859)	($39,706,126)	($24,992,558)	($31,073,960)	($27,470,867)	($33,718,513)	($35,019,709)	($45,131,670)	($36,256,184)
301	Commissions Expense	Software	USD	2023	Budget	($4,014,208)	($4,730,507)	($4,051,116)	($2,388,743)	($3,788,314)	($2,165,366)	($3,369,016)	($2,871,769)	($3,799,113)	($3,974,617)	($4,757,153)	($3,920,259)
302	Payroll Expense	Software	USD	2023	Budget	($11,164,524)	($11,563,467)	($10,590,962)	($6,628,891)	($10,201,147)	($6,336,226)	($8,308,362)	($8,049,766)	($9,552,167)	($9,069,604)	($11,039,826)	($9,218,680)

HE iSCALE

- Recommendation System
- Dynamic Pricing Models
- Customer Segmentation
- Risk Fraud Detection
- Chatbots & Virtual Assistants
- Customer Sentiment Analysis
- Product Demand Forecasting







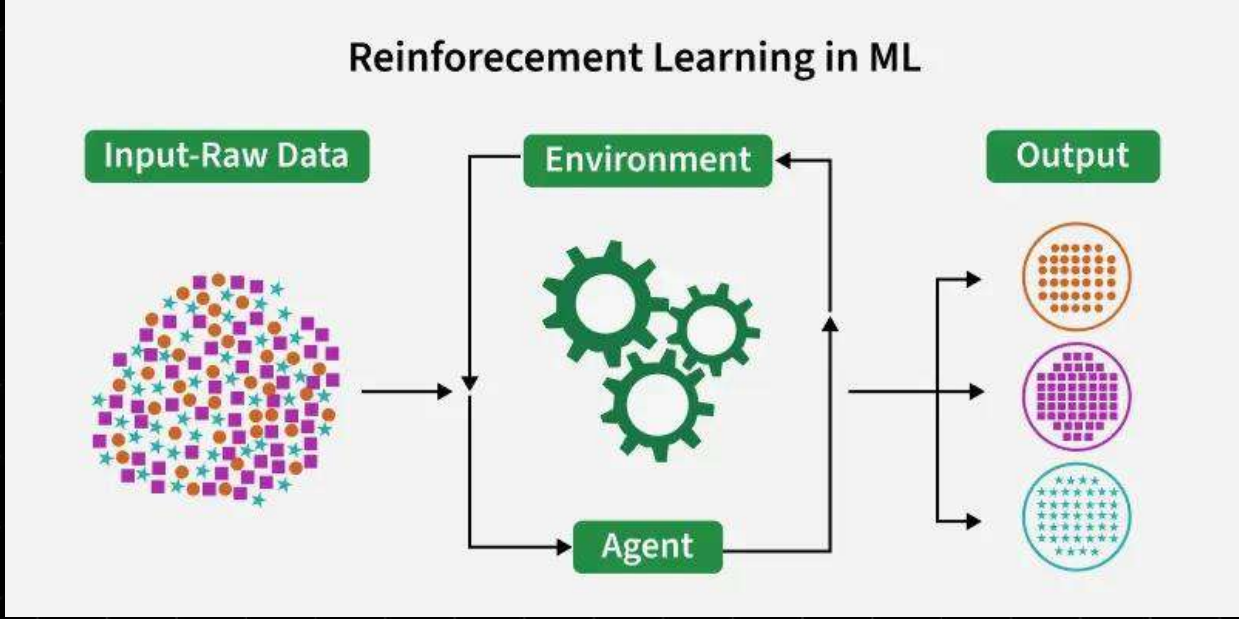
1 Agent (Learner)
👉 Jo seekh raha hai

2 Environment (World)
👉 Jisme agent kaam karta hai

3 Action
👉 Agent kya karta hai

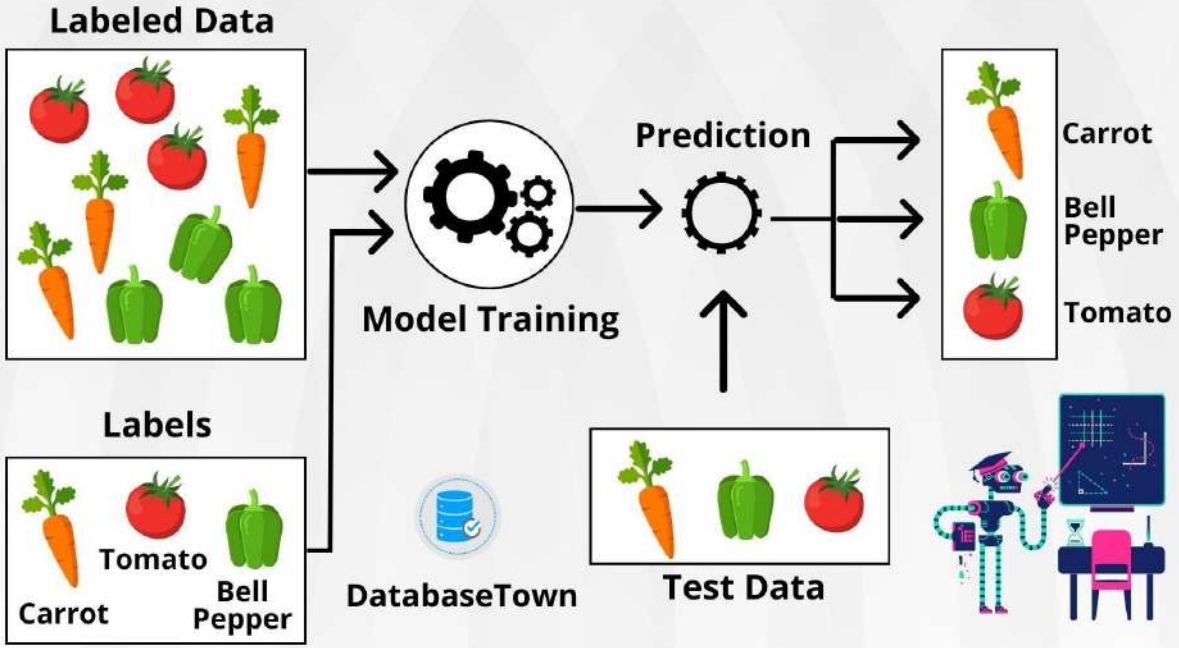
4 State
👉 Current situation

5 Reward
👉 Feedback



SUPERVISED LEARNING

Supervised machine learning is a branch of artificial intelligence that focuses on training models to make predictions or decisions based on labeled training data.



Supervised ka matlab hota hai ki model ko training ke dauran guide kiya ja raha hai — jese teacher student ko sikha raha ho

Supervised Machine Learning

Regression

Classification

🌍 Real-life Examples of Supervised Learning:

Use Case	Type	Explanation
Email Spam Detection	Classification	Spam ya Not Spam
House Price Prediction	Regression	Ghar ke features se price predict karna
Loan Approval System	Classification	Approve/Reject based on credit history
Stock Price Prediction	Regression	Future ka price estimate karna
Face Recognition in Photos	Classification	Identify person
Sentiment Analysis of Reviews	Classification	Positive/Negative review detect karna

Supervised Machine Learning

Dependent and Independent Features

📦 Simple Real-Life Example: House Price Prediction

Size (sqft)	Location	Bedrooms	Price (₹)
1000	Delhi	2	60,00,000
1500	Mumbai	3	95,00,000
1200	Bangalore	2	70,00,000

Regression ka kaam hota hai:
Independent features (X) ke basis par
Continuous value wale Dependent feature (Y) ko predict karna
Output 👉 **numerical & continuous value** hoti hai

Supervised Machine Learning

Regression Problem Statement

Experience (Years)	Education Level	Job Role	Location	Company Size	Salary (₹)
1	B.Tech	Data Analyst	Pune	Small	25,000
2	M.Tech	Data Analyst	Bangalore	Medium	35,000
3	B.Sc	Business Analyst	Delhi	Large	45,000
4	MBA	Data Scientist	Mumbai	Large	70,000
5	PhD	ML Engineer	Bangalore	Large	90,000
6	B.Tech	Data Scientist	Hyderabad	Medium	75,000

Supervised Machine Learning

Classification Problem Statement

Classification ka output discrete / categorical hota hai

Binary Classification
👉 Jab sirf 2 classes hoti hain
📌 Examples:
Spam ❌ / Not Spam ✅
Fraud ❌ / Not Fraud ✅

Multi-Class Classification
👉 Jab 2 se zyada classes hoti hain
📌 Examples:
Handwritten digits → (0–9)
Traffic Signal → Red / Yellow / Green
Student Grade → A / B / C / D

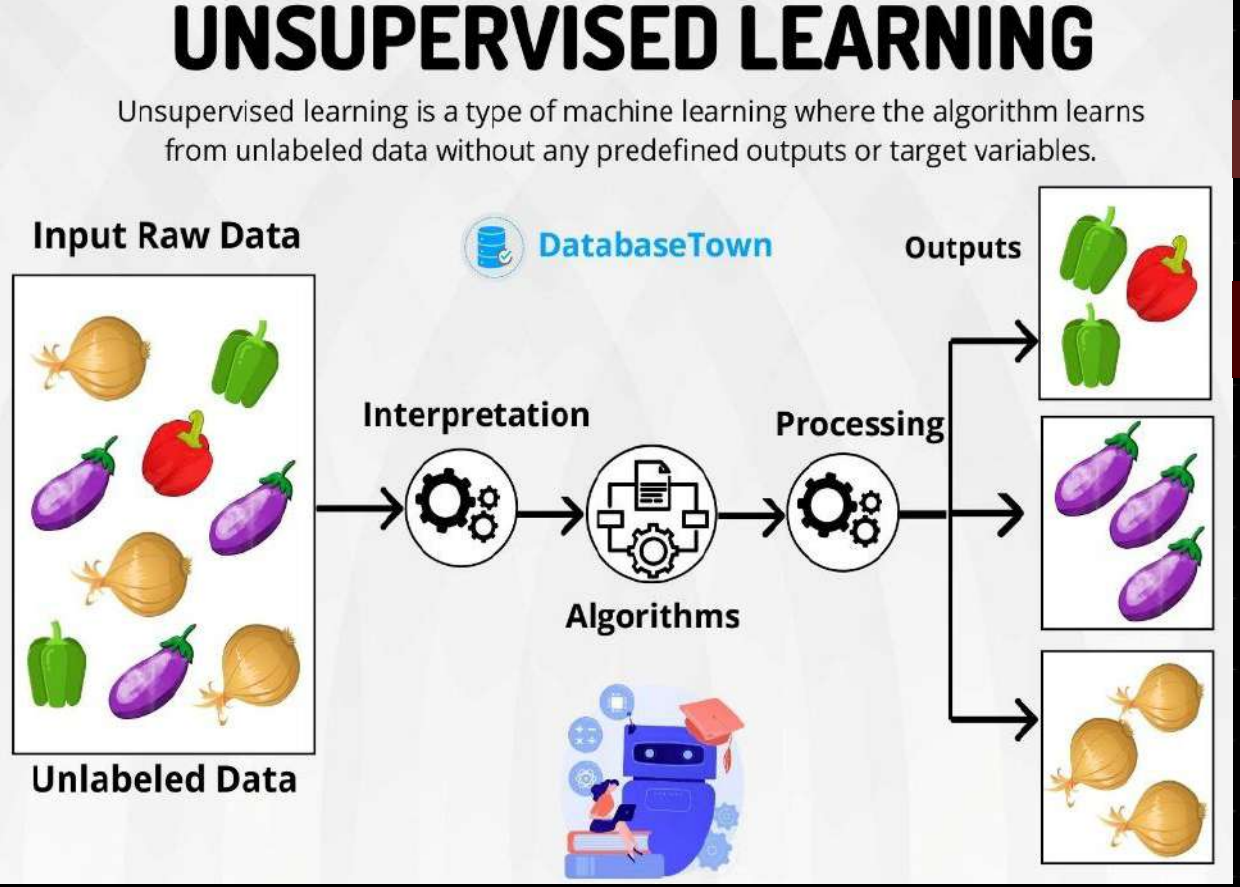
Attendance (%)	Assignment Score	Exam Score	Participation	Final Grade
90	85	80	High	A
75	70	68	Medium	B
60	55	50	Low	C
45	40	38	Very Low	D
85	88	90	High	A
70	60	65	Medium	B

Supervised Machine Learning

Classification Problem Statement

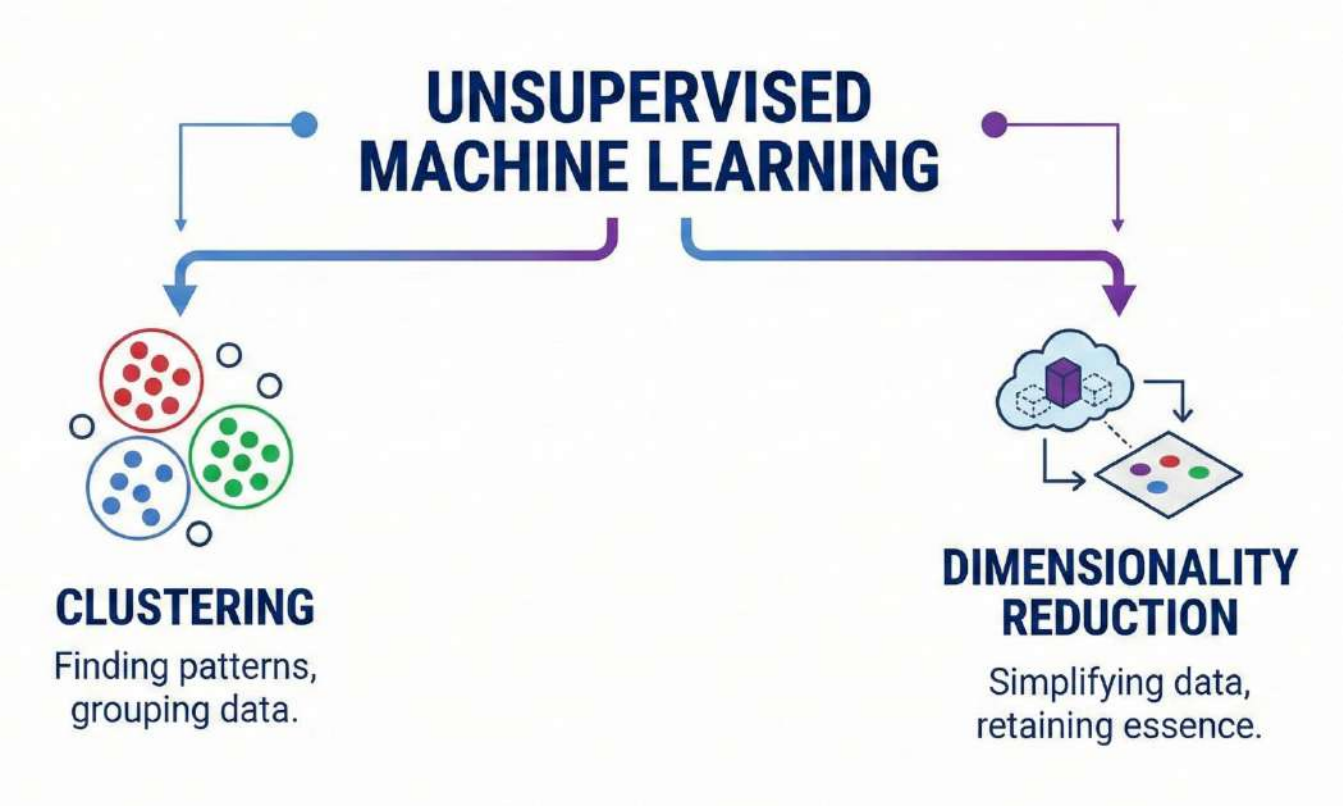
Age	Income (₹)	Credit Score	Employment Type	Loan Amount	Education	Marital Status	Output (Loan Approved?)
25	30,000	680	Salaried	1,50,000	Graduate	Single	1 (Yes)
40	22,000	540	Self-employed	2,00,000	12th Pass	Married	0 (No)
29	50,000	750	Salaried	3,00,000	Graduate	Married	1 (Yes)
35	18,000	620	Unemployed	1,00,000	Graduate	Single	0 (No)

Unsupervised Machine Learning



Jab aapke paas data hota hai, lekin uske saath koi label (output) nahi hota — tab hum unsupervised learning use karte hain. 📌 Yani model ko sirf input diya jaata hai, aur model khud pattern find karta hai.

Unsupervised Machine Learning



Clustering

Clustering ek unsupervised learning technique hai jisme model similar data points ko ek group (cluster) me daalta hai, bina kisi predefined label ke.

Yeh technique useful hai jab:
Tumhare paas labels nahi hote
Tumhe hidden patterns ya groups discover karne hote hain

Problem Statement:
Ek mall chahta hai ki wo apne customers ko unke income aur spending pattern ke basis par group kare — taaki targeted offers bhej sake.

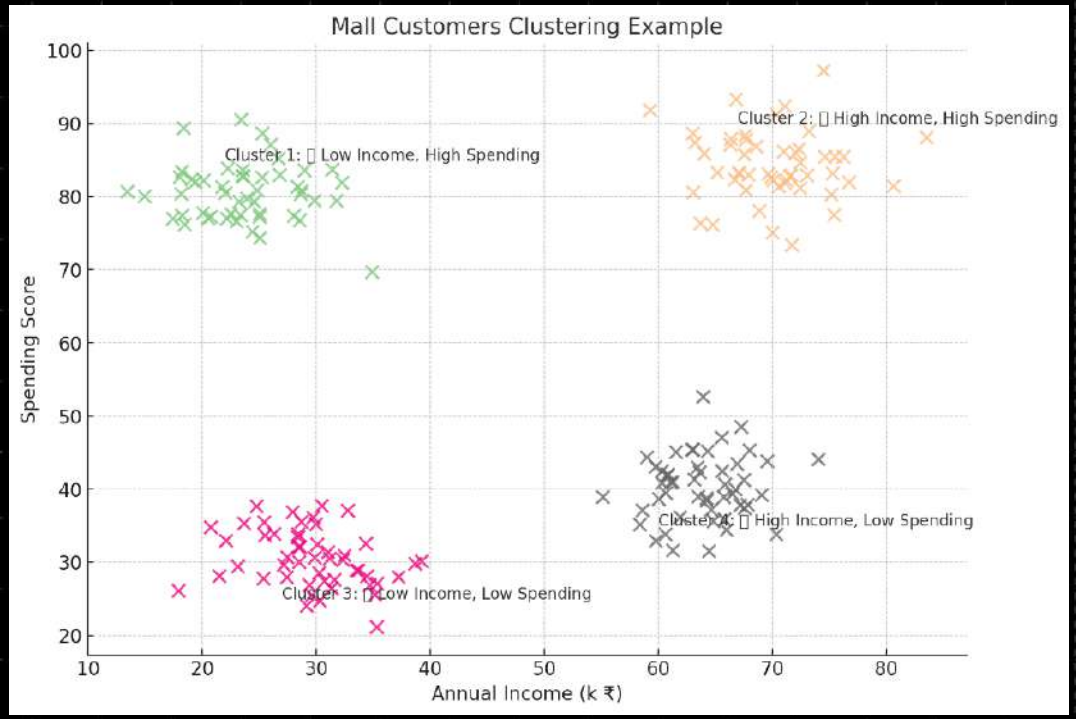
Machine ko bas input milta hai:
Annual Income (in ₹)
Spending Score (kitna spend karte ho mall me)

Unsupervised Machine Learning

Clustering

Problem Statement:

Ek mall chahta hai ki wo apne customers ko unke income aur spending pattern ke basis par group kare — taaki targeted offers bhej sake.



Unsupervised Machine Learning

Dimensionality Reduction

Jab data me bahut zyada features hote hain, to unhe **kam aur useful dimensions** me convert karna.

Jab hum high-dimensional data (bahut saare features) ko kam features me convert karte hain — bina important information ko lose kiye — use kehte hain Dimensionality Reduction.

- ✅ Yeh Unsupervised Learning ka part hai
- ✅ Real-world data me 1000+ features hote hain
- ✅ Visualize karna mushkil hota hai — isliye hum 2D ya 3D me reduce karte hain

Step	Description
Input Image	100 x 100 = 10,000 pixel values (10,000D)
Dimensionality Reduction	Convert 10,000 → 100 key facial features
Model Comparison	New face ke 100 features match karo
Unlock Decision	Match ho gaya? Unlock!

Supervised Machine Learning

- Linear Regression
- Ridge & Laso
- Logistic Regression
- Decision Tree
- Adaboost
- Random Forest
- Gradient Boosting
- Xg Boost
- Naïve Bayes
- SVM
- KNN

Unsupervised Machine Learning

- K Means Clustering
- DB Scan
- Hierarchical Clustering
- PCA
- LDA

Supervised Machine Learning

- Linear Regression

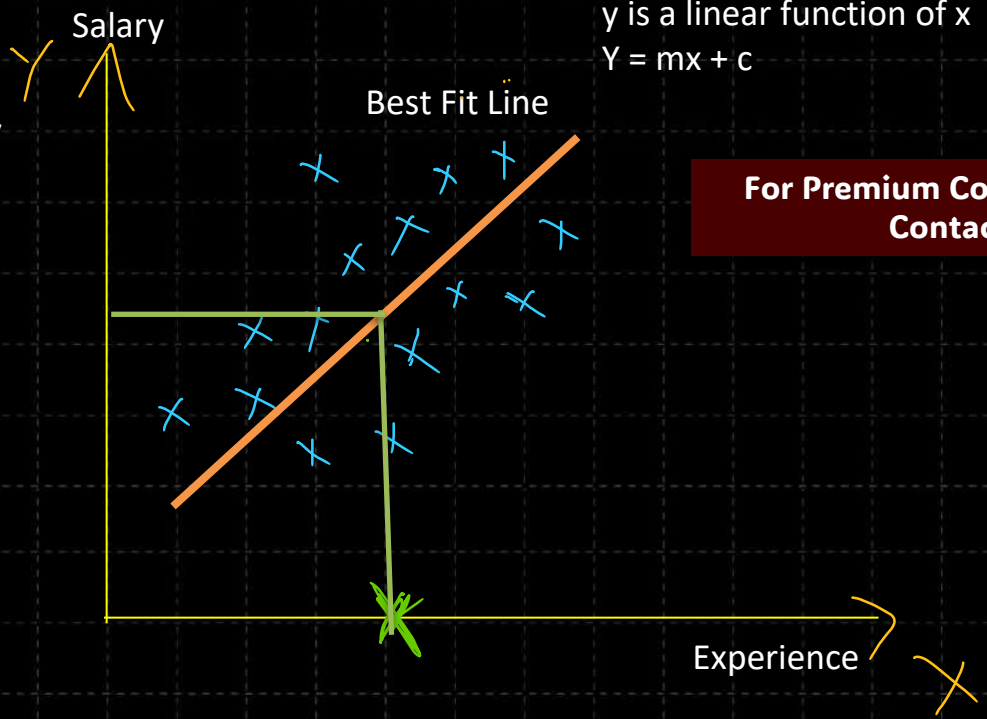
Linear Regression

Linear Regression ek supervised machine learning algorithm hai jo:
- Input features (X) ke basis par
- Continuous numerical output (y) predict karta hai
- Using a straight line (linear relationship)

Linear regression best-fit straight line nikalta hai jo:
- Data ke sabse paas se guzre
- Future ke liye prediction kare

📌 Example:
House size → House price
Years of experience → Salary
Hours studied → Marks

Experience (years)	Salary (LPA)
1	3
2	4
3	5
4	6



y is a linear function of x
Y = mx + c

Linear Regression

Experience (years)	Salary (LPA)
1	3
2	4
3	5
4	6

y is a linear function of x

Y = mx + c

$h_\theta(x) = \theta_0 + \theta_1 x$

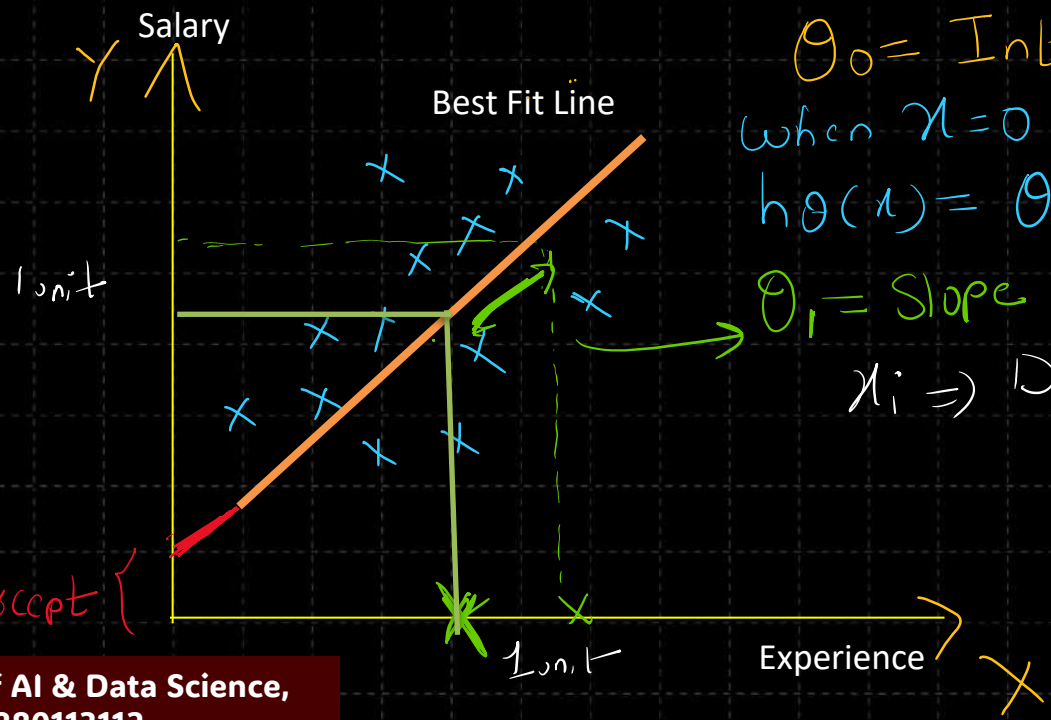
θ_0 = Intercept

when $x = 0$

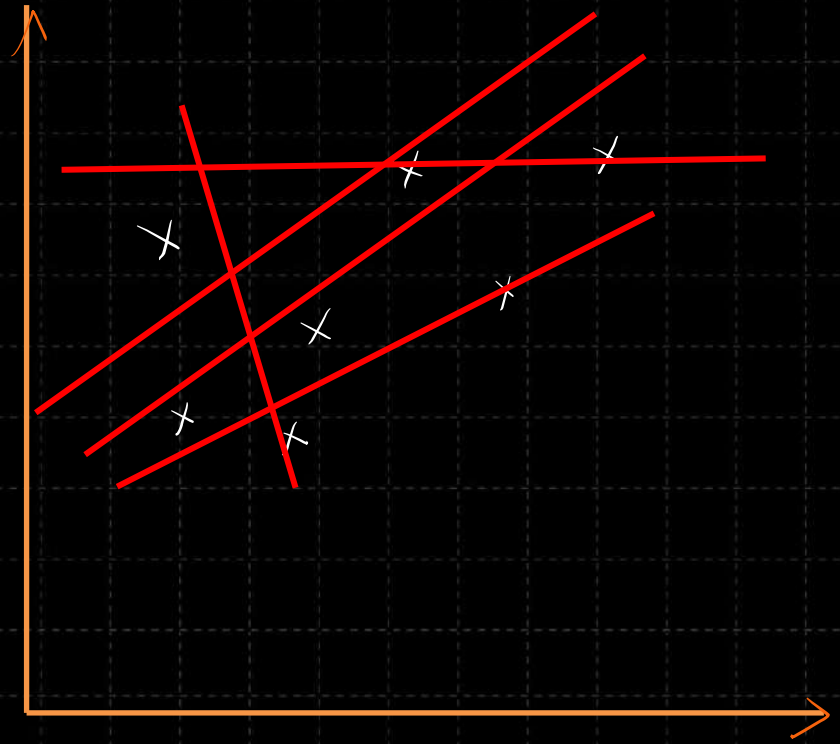
$h_\theta(x) = \theta_0$

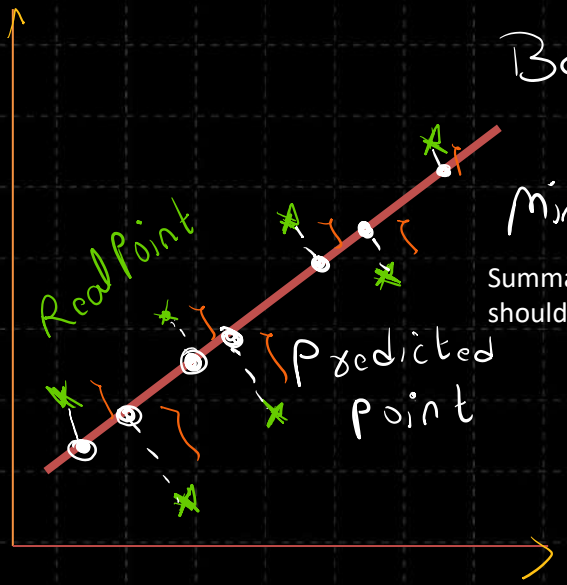
θ_1 = Slope

x_i ⇒ Data Points



How will you find which is the best fit line ?





Best fit Line

Minimal Distance

Summation of all points distance should be minimum

hypothesis

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Cost function / Squared Error Funct

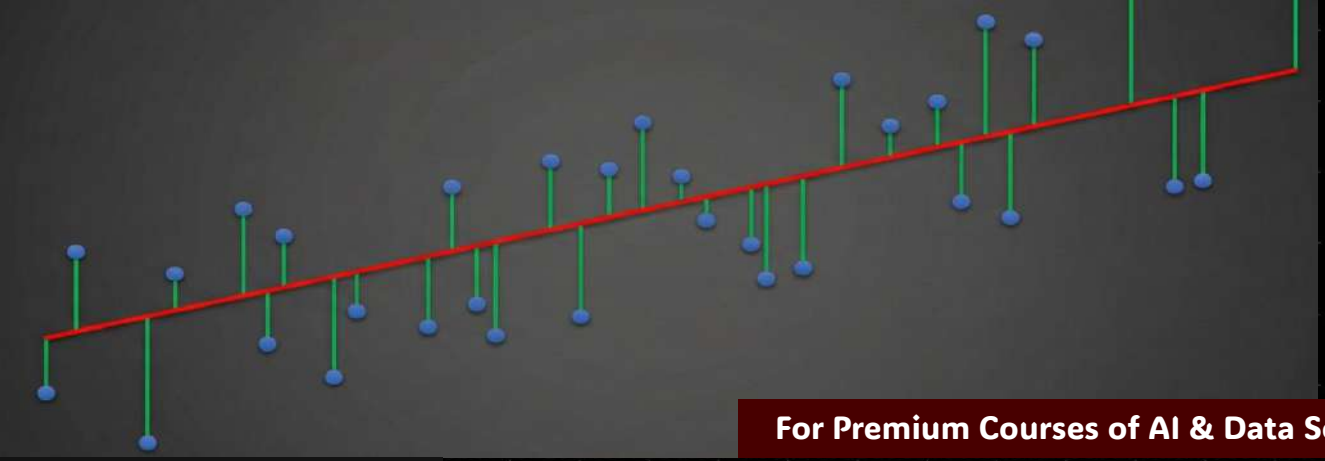
$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x) - y)^2$$

Derivative $\frac{d}{dx}(x^2) = 2x$

$x^n = n x^{n-1}$

m is all the data points

↳ Average



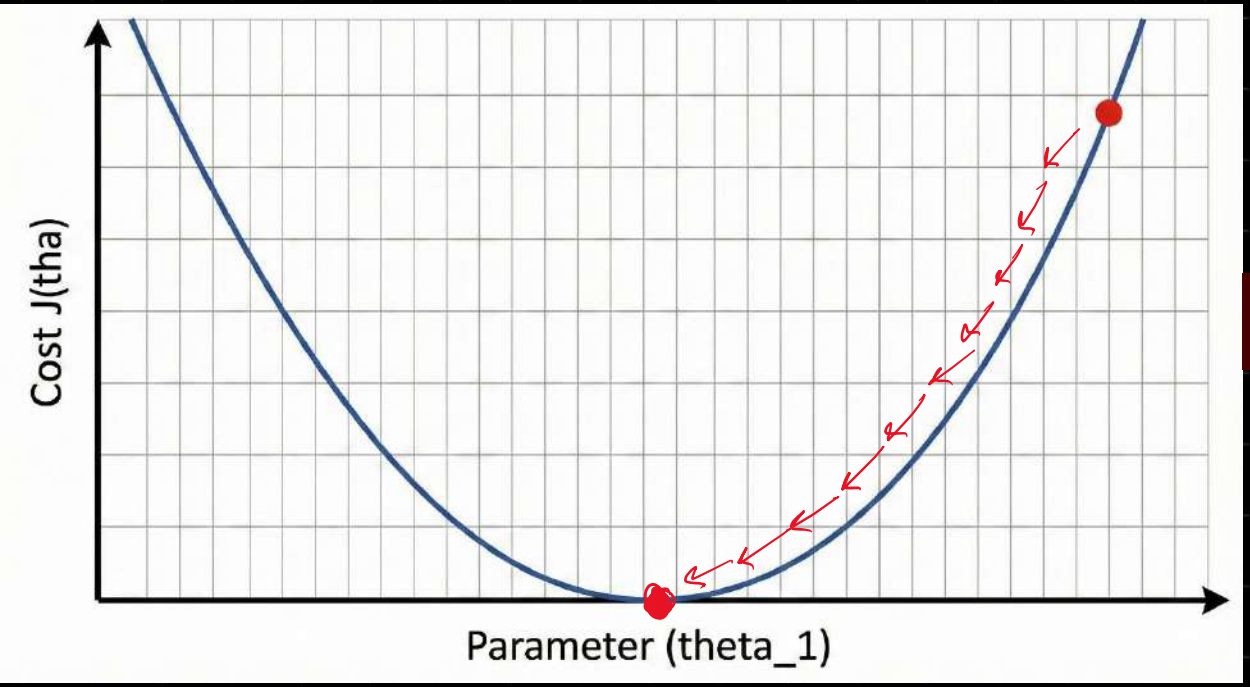
STEP 2: Cost Function

Formula: $J(\theta) = \frac{1}{2m} \sum (\text{Prediction} - \text{Asliyat})^2$

Ab humein ye napna hai ki hamari line kitni "bekar" hai. Jo upar green lines aapne dekhi, un sabki lambai ko square karke jodne par humein **Total Cost (J)** milti hai.

Agar hum is Cost ko graph karein, toh wo ek **Katore (Bowl)** jaisa dikhta hai.

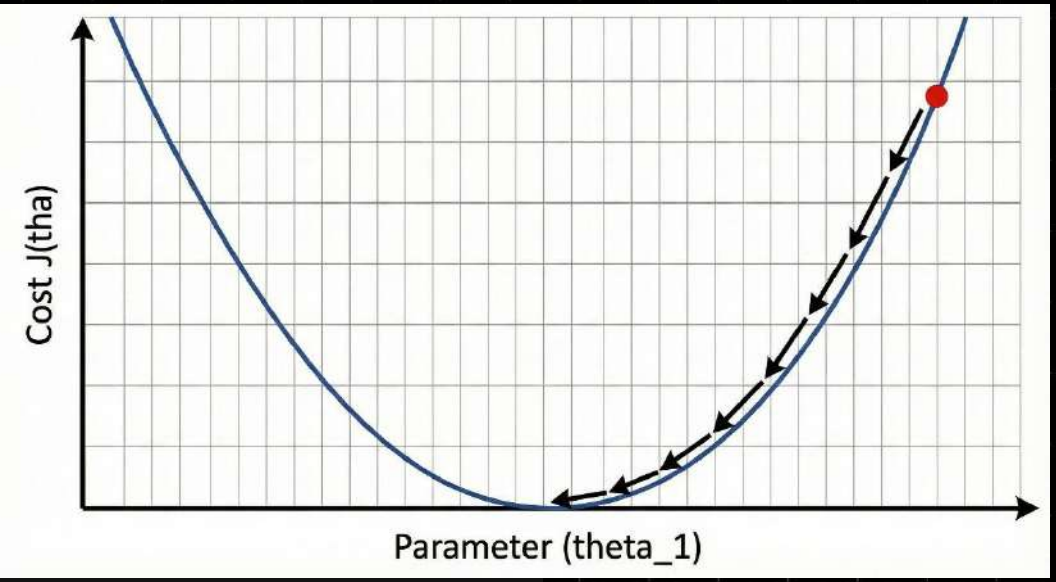
- **X-axis (Parameter):** Ye hamara slope (θ_1) hai.
- **Y-axis (Cost J):** Ye hamari galti hai.
- **Red Dot:** Ye hamari abhi ki line ki sthiti hai.



Step 2
hamara **Red Dot** graph mein kitna upar hai! Iska matlab hamari galti bahut zyada hai. Hamara maqsad hai graph ke sabse niche pahunchna.

Formula: $J(\theta) = \frac{1}{2m} \sum (\text{Prediction} - \text{Asliyat})^2$

Agar hum is Cost ko graph karein, toh wo ek Bowl jaisa dikhta hai.
X-axis (Parameter): Ye hamara slope 01 hai.
Y-axis (Cost J): Ye hamari galti hai.
Red Dot: Ye hamari abhi ki line ki sthiti hai.



STEP 3: Gradient Descent

Formula: $\theta_{new} = \theta_{old} - \alpha \times \frac{\partial J}{\partial \theta}$ (Derivative)

- **Derivative ($\frac{\partial J}{\partial \theta}$):** Ye batata hai ki dhalan (slope) kis taraf hai.
- **Learning Rate (α):** Ye tay karta hai ki hum kitna bada kadam lenge.

Har kadam par hamara slope (θ_1) update hota hai aur galti kam hoti jati hai.

STEP 4: Final Result & Evaluation (R^2)

Formula: $R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$

Term	Kya batata hai?	Formula ka logic
R^2	Model average se kitna behtar hai?	$1 - \frac{\text{My Error}}{\text{Average Error}}$
Adjusted R^2	Kya saare features kaam ke hain?	R^2 ko features (k) ke saath adjust karna.

THE iSCALE

Significance:

- Agar SS_{res} (aapka error) bahut chota hai, toh $\frac{SS_{res}}{SS_{tot}}$ zero ke paas hoga, aur R^2 **1** ke paas (Perfect Model).
- Agar aapki line average line se bhi buri hai, toh R^2 **0** ya negative bhi ho sakta hai.

Adjusted R^2 ki entry

Agar main model mein faltu ke columns (features) dalta rahoon (jaise: 'Ghar ke owner ka naam', 'Ghar ka rang'), toh R^2 kabhi niche nahi girega. Wo ya toh badhega ya wahi rahega. **Model ko lagta hai ki jitna zyada data, utna achcha, bhale hi wo data kachra ho.**

Is "kachre" ko pakadne ke liye aata hai **Adjusted** R^2.

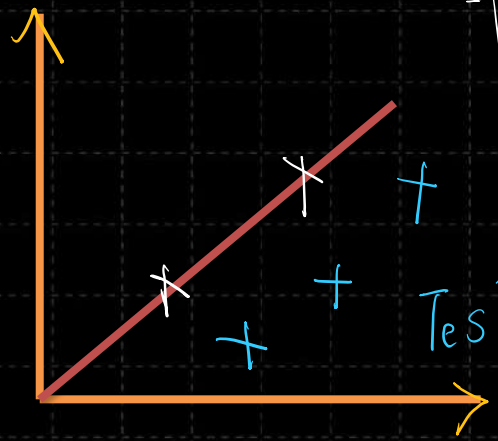
Formula:

$$Adjusted\ R^2 = 1 - \left[\frac{(1 - R^2)(n - 1)}{n - k - 1}\right]$$

n: Total data points (Rows).

k: Number of independent variables (Columns/Features).

Concept of Overfitting



Training Data Overfitting → Low Bias

Model Performs well → Training Data

Test data

Fails to perform well → Test Data

High Variance

Bias vs Variance (Machine Learning)

🟢 Pehle ek simple example samjho

Socho tum cricket me batting kar rahi ho 🎯

•Ball ko center me hit karna = correct prediction

•Ball kahi bhi lagna = error

🔴 Bias kya hota hai?

📌 Simple definition:

Bias ka matlab: Model zyada simple hai aur data ko achhe se samajh hi nahi pa raha

🧠 ML language me:

•Model strong assumptions bana leta hai

•Real pattern ko ignore kar deta hai

🔵 Variance kya hota hai?

📌 Simple definition:

Variance ka matlab: Model zyada complex ho gaya aur har chhoti cheez yaad kar raha

🧠 ML language me:

•Noise ko bhi pattern samajh leta hai

•Training data rat leta hai

Concept of Underfitting

- Model Accuracy is bad with training data
- Model accuracy is also bad with Test data

High Bias
High Variance

Model 1	Model 2	Model 3
Training Accuracy = 90 %	Training Accuracy = 92 %	Training Accuracy = 70 %
Testing Accuracy = 80 %	Testing Accuracy = 91 %	Testing Accuracy = 65 %
Overfitting Low Bias High Variance	Generalised Model Low Bias Low Variance	Underfitting High Bias High Variance

Best Output

Ridge Regression – L2 Regularization

Ridge Regression = Linear Regression ka improved version

👉 Jab **Linear Regression model overfit** hone lagta hai

👉 Jab **features bahut zyada ho jaate hain**

👉 Jab **features aapas me strongly related (multicollinearity)** hote hain

tab hum **Ridge Regression** use karte hain.

→ Overfitting

Multicollinearity matlab:

- Features aapas me highly related

  Example:
- `house_size`
- `number_of_rooms`

👉 Linear Regression confuse ho jaata hai

👉 Weights unstable ho jaate hain

✔ Ridge Regression:

- Dono features ko **thoda-thoda importance** deta hai
- Extreme weight nahi deta

Ridge Regression

THE iSCALE

Problem in Linear Regression

For Premium Courses of AI & Data Science, Contact us - 7880113112

Linear Regression formula:

$$y = w_1x_1 + w_2x_2 + ... + w_nx_n + b$$

Model training ke time:

- Model **weights (w)** ko adjust karta hai
- Taaki **error minimum** ho

❌ **Problem kya hoti hai?**

1. Agar features **zyada ho**
2. Ya features **ek jaise behave kar rahe ho**
3. Ya data **kam ho**

➡️ Model kuch weights ko **bahut bada** bana deta hai

➡️ Result: **Overfitting**

📌 Overfitting matlab:

- Training data pe best
- New data pe fail ❌

Ridge Regression

Ridge kehta hai:

> "Error kam karo, **lekin weights ko bhi chhota rakho**"

Ridge Regression ka formula (Simple samjho)

Linear Regression Loss:

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

Ridge Regression Loss:

$$Loss = \sum (y - \hat{y})^2 + \lambda \sum w^2$$

λ value	Effect
λ = 0	Normal Linear Regression
λ small	Thoda control
λ large	Strong control
λ bahut zyada	Model almost flat

Ridge Regression

Ridge Regression ka main idea (One Line)

Model simple rakho, weights ko zyada bada mat hone do

Feature	Linear	Ridge
Overfitting	High chance	Low
Multicollinearity	Problem	Handle karta
Weights	Bahut bade ho sakte	Controlled
Bias	Low	Thoda high
Variance	High	Low

📌 Ridge = **Bias ↑ , Variance ↓**

Ridge Regression

🔹 Real Life Example (Best for understanding)

🎓 **Student marks prediction**

Features:

- Study hours
- Online lectures
- Mock tests
- Notes reading

👉 Sab features thode-thode important

👉 Ridge sabko use karega

👉 Kisi ek ko extreme weight nahi dega

Lasso Regression Vs Ridge Regression

THE iSCALE

Socho tum **salary predict** kar rahi ho

Tumhare paas ye inputs (features) hain:

1. College CGPA
2. Internship experience
3. Projects
4. Coding skills
5. School marks
6. Extra activities

👉 Model ko decide karna hai:

- Kisko kitni importance deni hai

Ridge ka mindset:

> "Har cheez ka thoda-thoda role ho sakta hai"

Ridge kya karega?

- CGPA ✔ (thoda weight)
- Internship ✔ (thoda weight)
- Projects ✔ (thoda weight)
- Coding ✔ (thoda weight)
- School marks ✔ (bahut kam)
- Extra activities ✔ (bahut kam)

📌 Result:

- **Koi feature hata nahi**
- Sabke weights **small & controlled**

Lasso ka mindset:

> "Jo kaam ka nahi, usse hatao"

Lasso kya karega?

- CGPA ✔
- Internship ✔
- Projects ✔
- Coding ✔
- School marks ❌ (remove)
- Extra activities ❌ (remove)

📌 Result:

- Model simple
- Easy to explain
- Sirf strong signals

Lasso Regression, L1 Regularization

THE iSCALE

Ridge Regression
Purpose
1) Preventing Overfitting

$$Loss = \sum (y - \hat{y})^2 + \lambda \sum w^2$$

Lasso Regression
Purpose
1) Preventing Overfitting
2) Feature Selection

$$Loss = \sum (y - \hat{y})^2 + \lambda \sum |w|$$

👉 Lasso automatically:
•Important features rakhta hai
•Unimportant features hata deta hai

📌 Isliye Lasso ko bolte hain:
Embedded feature selection method

λ value	Effect
λ = 0	Normal Linear Regression
λ small	Thoda feature cleaning
λ large	Bahut saare weights = 0

Lasso Regression, L1 Regularization

Ridge Regression

$$Loss = \sum (y - \hat{y})^2 + \lambda \sum w^2$$

📌 Example:

- w = 2 → penalty = 4
- w = 5 → penalty = 25

👉 Matlab:

> "Weight bada hua to strongly punish karunga"

Effect:

- Weight chhota hota jata hai
- Lekin **zero tak nahi jata**

Lasso Regression

$$Loss = \sum (y - \hat{y})^2 + \lambda \sum |w|$$

📌 Example:

- w = 2 → penalty = 2
- w = 5 → penalty = 5

👉 Matlab:

> "Ya kaam ka hai, ya bahar"

Effect:

- Weight directly **zero ho sakta hai**
- Feature remove ho jata hai

Ridge: Weight bada ho raha hai → dheere-dheere kam karo
Lasso: Weight ka kaam hi nahi → seedha ZERO karo

Lasso Regression

🔹 Real-life example (Best for understanding)

🎓 Student salary prediction

Features:

- College CGPA
- Internships
- Projects
- School marks
- Extra activities

👉 Lasso karega:

- Projects ✔
- Internships ✔
- School marks ❌ (weight = 0)

📌 Result:

- Simple model
- Easy explanation

🔹 Kab Lasso use kare?

✔ Jab:

- Features zyada ho
- Tumhe **important features chahiye**
- Interpretability important ho

❌ Jab:

- Sab features useful ho → Ridge better

Logistic Regression

Logistic Regression ek Supervised Machine Learning algorithm hai jo classification problems solve karta hai.

👉 Simple words me:

Jab output sirf YES/NO, TRUE/FALSE, 0/1 hota hai — tab Logistic Regression use hota hai

🔥 Examples (Real life)

- Email → Spam / Not Spam
- Student → Pass / Fail
- Customer → Buy karega / Nahi karega
- Disease → Hai / Nahi hai

📌 Output = Category, number nahi

Pehle Linear Regression ko yaad karo

Linear Regression:

$$y = wx + b$$

Problem:

- Output **-∞ se +∞** tak ja sakta hai
- Classification ke liye useful nahi

Example:

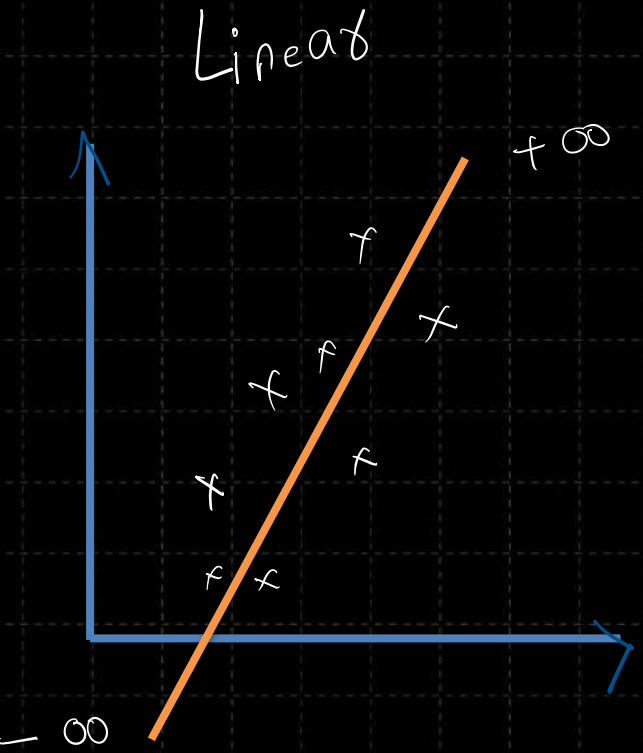
- Pass = 1
- Fail = 0

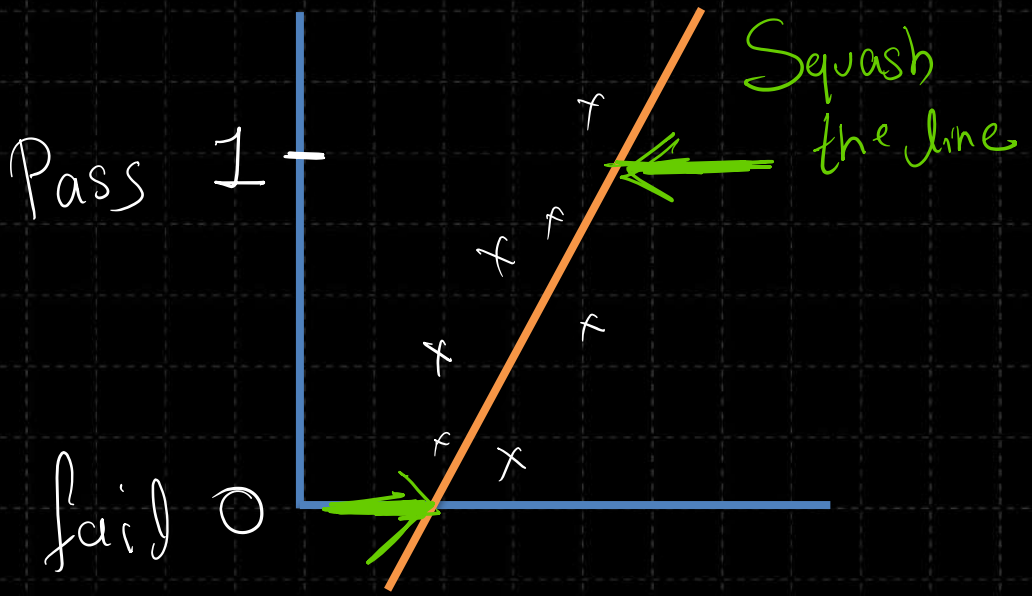
Linear Regression:

- 1.4 ❌
- -0.7 ❌

👉 Invalid outputs

Logistic Regression





Logistic Regression

$h_\theta(x) = \theta_0 + \theta_1 x_1$

$h_\theta(x) = g(\theta_0 + \theta_1 x_1)$

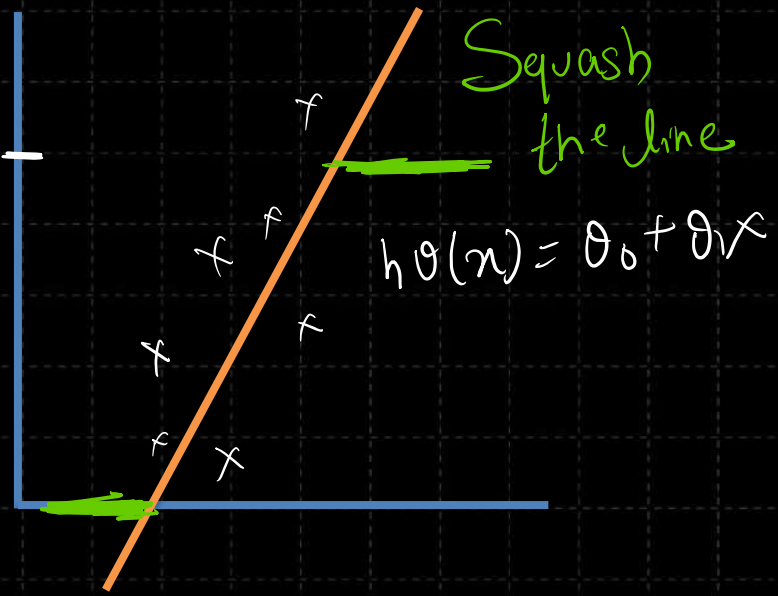
Let $z = \theta_0 + \theta_1 x_1$

$h_\theta(x) = g(z)$

$$h_\theta(x) = \frac{1}{1 + e^{-z}}$$

→ Sigmoid or Logistic function

$$h_\theta(x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x)}}$$



Logistic Regression

Sigmoid Function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

z value	Sigmoid output
Bahut negative	0 ke paas
0	0.5
Bahut positive	1 ke paas

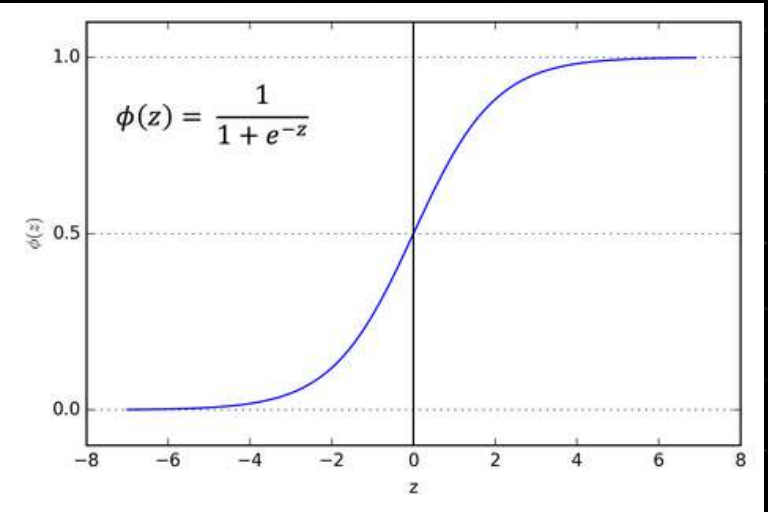
🔹 Decision boundary kya hoti hai?

👉 Rule:

•Probability ≥ 0.5 → Class 1

•Probability < 0.5 → Class 0

📌 0.5 ke aas-paas ek boundary banti hai jise kehte hain Decision Boundary



Logistic Regression

Cost function

Linear Regression $J(\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h\theta(x) - y)^2$

$z = \theta_0 + \theta_1 x$

$\theta_0 \Rightarrow 0$ (passing from origin)

Logistic Regression

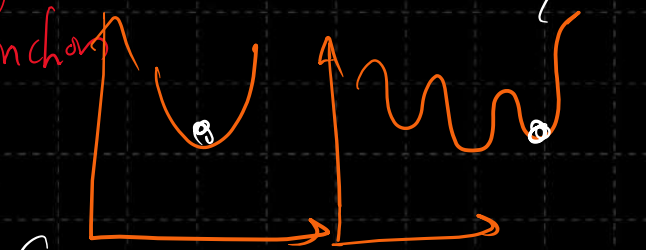
$h\theta(x) = \frac{1}{1+e^{-z}}$

$h\theta(x) = \frac{1}{1+e^{-\theta_1 x}}$

$J(\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(\frac{1}{1+e^{-\theta_1 x}} - y\right)^2$

We Cannot use this cost function for Logistic Regression

Never Reach global minima

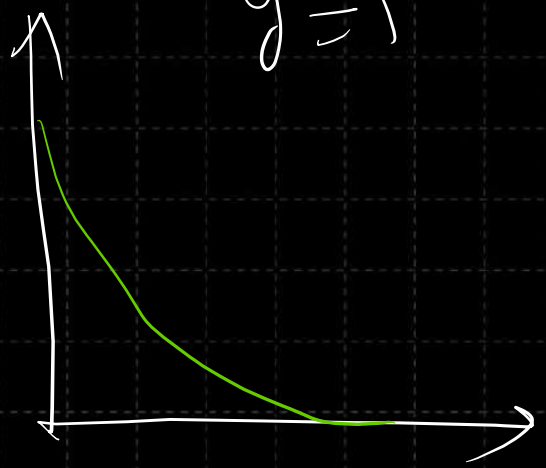


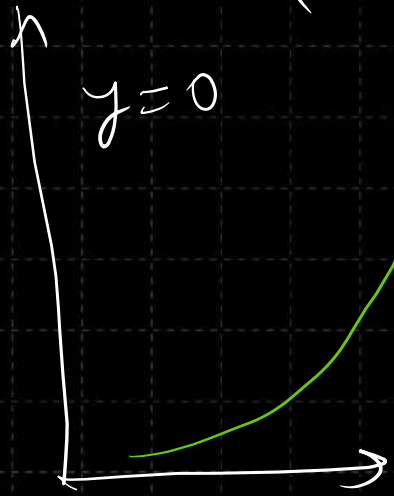
Logistic Regression

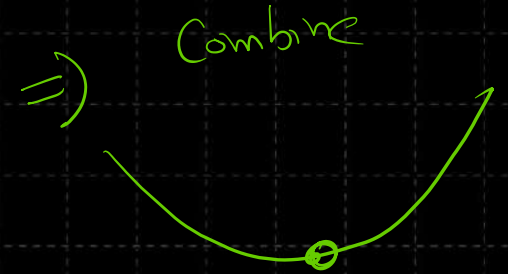
Logistic Regression Cost function

$$J(\theta_1) = \begin{cases} -\log(h\theta(x)) & y=1 \\ -\log(1-h\theta(x)) & y=0 \end{cases}$$

$$h\theta(x) = \frac{1}{1+e^{-\theta_1 x}}$$







Logistic Regression

Logistic Regression Cost function

$$J(\theta_1) = \begin{cases} -\log(h\theta(x)) & y=1 \\ -\log(1-h\theta(x)) & y=0 \end{cases}$$

$$h\theta(x) = \frac{1}{1+e^{-\theta_1 x}}$$

Combined Equation

$$\text{Cost}(h\theta(x), y) = -y \log(h\theta(x)) - (1-y) \log(1-h\theta(x))$$

$y=1$

$$\text{Cost}(h\theta(x), y) = -\log(h\theta(x)) - (1-1) \log(1-h\theta(x))$$

$y=0$

$$\text{Cost}(h\theta(x), y) = -0 - (1-0) \log(1-h\theta(x))$$

Logistic Regression

Logistic Regression

Cost function

$$J(\theta) = \frac{-1}{2m} \sum_{i=1}^{m} \left(y \log(h\theta(x)) + (1-y) \log(h\theta(x)) \right)$$

$$h\theta(x) = \frac{1}{1 + e^{-(\theta, x)}}$$

Logistic Regression

THE iSCALE

Performance Metrics

x_1	x_2	y actual	$\hat{y}$ predicted
		0	1
—	—	1	1
—	—	0	0
—	—	1	1
—	—	1	1
—	—	0	1
—	—	1	0

$$\text{Accuracy} = \frac{3+1}{3+2+1+1}$$

Predict \ Actual	1	0
1	3	2
0	1	1

$$\text{Accuracy} = \frac{4}{7} = 0.57$$

Accuracy = 57%

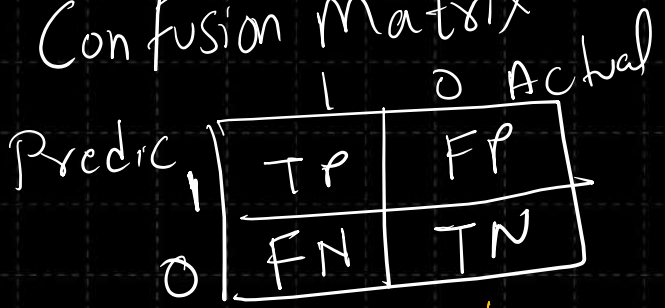
Confusion Matrix

Predict \ Actual	1	0
1	TP	FP
0	FN	TN

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+FN+TN}$$

Logistic Regression

Confusion Matrix



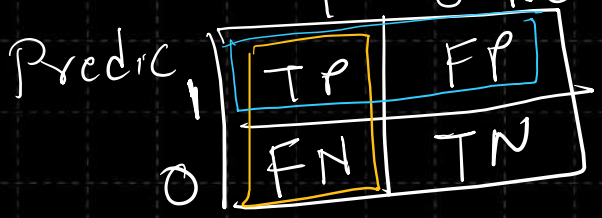
Always try to reduce FP↓ & FN↓

0 → 900
1 → 100
} Imbalanced Dataset

0 → 600
1 → 400
} Balanced Dataset

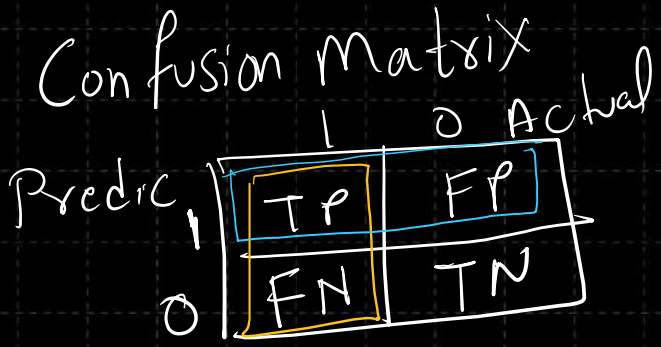
Logistic Regression

Confusion Matrix



For Imbalanced Data we need to work on

Logistic Regression



✅ TP – True Positive
Actual = YES
Predicted = YES
👉 Example:
Patient ko disease thi aur model ne disease predict ki

❌ FP – False Positive (Type-1 Error)
Actual = NO
Predicted = YES
👉 Example:
Healthy patient ko galti se disease bol diya

❌ FN – False Negative (Type-2 Error)

Actual = YES
Predicted = NO
👉 Example:
Disease thi lekin model miss kar gaya
⚠️ Most dangerous in medical cases

✅ TN – True Negative
Actual = NO
Predicted = NO

Logistic Regression

THE iSCALE

$$Precision = \frac{TP}{TP + FP}$$

Meaning:
Jitne positive predict kiye, unme se kitne sahi the?
Simple words:
👉 *"Model jab YES bolta hai, to kitna bharosa karein?*

Example:

- TP = 40
- FP = 10

$$Precision = \frac{40}{40 + 10} = 0.80$$

👉 80% time model ka YES correct hai

Logistic Regression

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

Meaning:
Actual positives me se kitne detect kar paye?
Simple words:
👉 *"Jo YES actually the, unme se kitne pakad liye?"*

Example:

- TP = 40
- FN = 20

$$Recall = \frac{40}{40 + 20} = 0.67$$

👉 67% disease cases detect hue

Logistic Regression

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Meaning:
Precision aur Recall ka balanced average
📌 Harmonic mean use hota hai
➡️ Extreme values ko punish karta hai

When to Use F1-Score?
•Imbalanced data
•Jab FP aur FN **dono important** ho
•Single metric chahiye model compare karne ke liye

Example:

- Precision = 0.8
- Recall = 0.6

$$F1 = \frac{2 \times 0.8 \times 0.6}{0.8 + 0.6} = 0.685$$

Naive Bayes Algorithm

Naive Bayes algorithm Machine Learning ki ek bahut hi popular aur effective supervised learning technique hai. Ye basically Classification tasks ke liye use hoti hai. "Bayes" isliye kyunki ye Bayes' Theorem par based hai.

Independent Event – Rolling a Dice

$P(1) = 1/6$

$P(2) = 1/6$

$P(3) = 1/6$

$P(4) = 1/6$

$P(5) = 1/6$

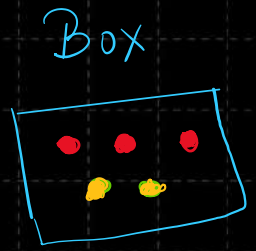
$P(6) = 1/6$

→ Independent Event

Naive Bayes Algorithm

THE iSCALE

Dependent Event



3 Apple , 2 Banana

$P(A) = 3/5$

$P(B) = \frac{2}{4} = \frac{1}{2}$

$P(A \text{ and } B) = P(A) \times P(B/A)$

$= \frac{3}{5} \times \frac{1}{2}$

$= \frac{3}{10} = 0.3$

$P(B) = \frac{2}{5}$

$P(A) = \frac{3}{4}$

$P(B \text{ and } A) = P(B) \times P(A/B)$

$= \frac{2}{5} \times \frac{3}{4}$

$= \frac{3}{10} = 0.3$

Naive Bayes Algorithm

$$P(A \text{ and } B) = P(B \text{ and } A)$$

$$P(A) \times P(B/A) = P(B) \times P(A/B)$$

$$P(B/A) = \frac{P(B) \times P(A/B)}{P(A)}$$

Bayes Theorem

Naive Bayes

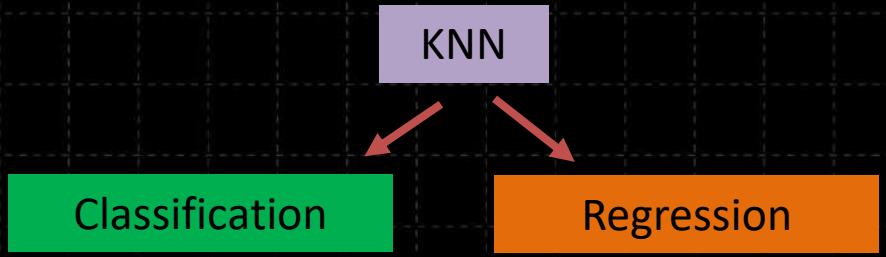
K Nearest Neighbour - KNN Algorithm

KNN yaani **K-Nearest Neighbors** ek bahut hi simple lekin powerful supervised machine learning algorithm hai.

KNN ek **Instance-based** aur **Lazy Learner** algorithm hai.

Lazy Learner kyun? Kyunki ye training ke waqt kuch nahi seekhta. Ye bas saara data yaad kar leta hai. Asli kaam tab shuru hota hai jab hum isse koi naya prediction karne ko kehte hain.

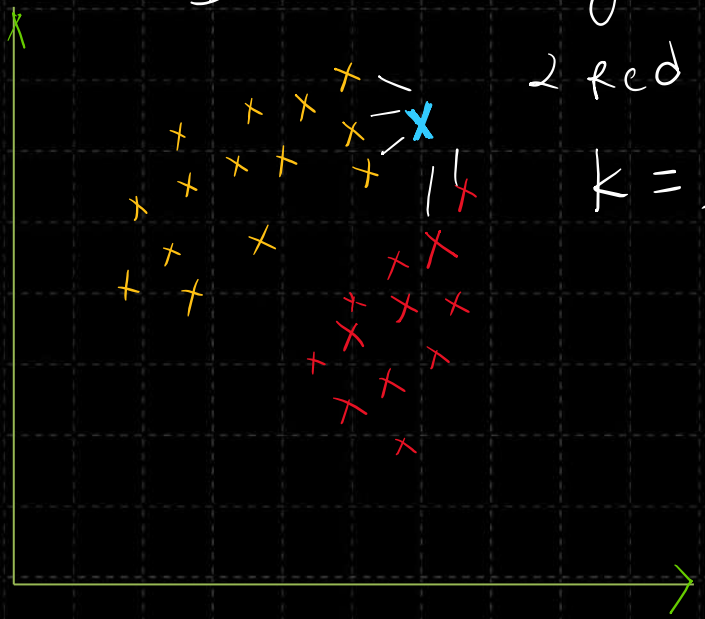
Kaise kaam karta hai? Jab koi naya data point aata hai, KNN apne purane data mein uske sabse kareeb (nearest) wale points dhoondta hai.



K Nearest Neighbour - KNN Algorithm

THE iSCALE

Classification



Euclidean Distance

(x_2, y_2)

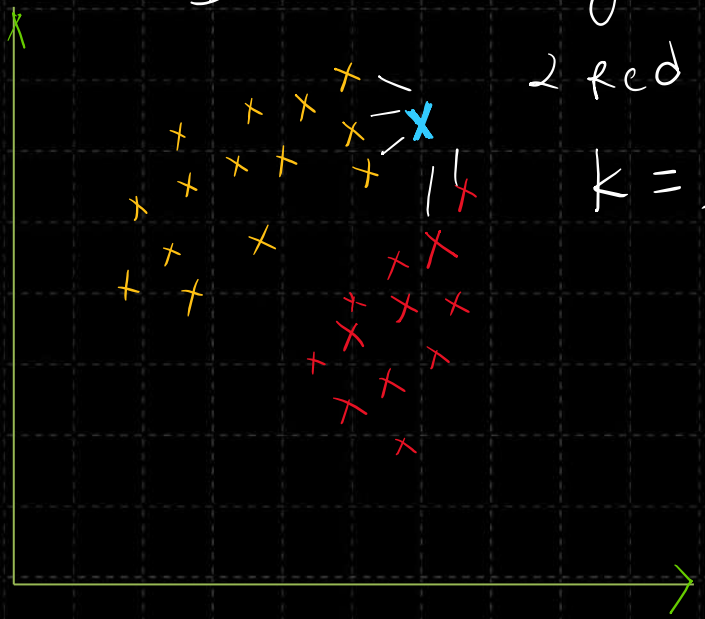
(x_1, y_1)

$$\sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}$$

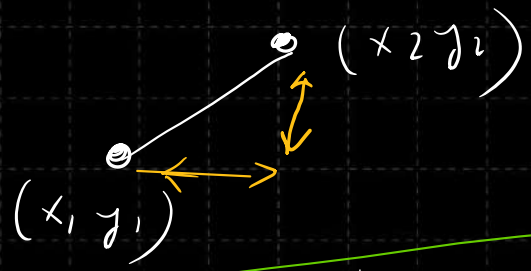
L_2 Norm → Euclidean Dist

K Nearest Neighbour - KNN Algorithm

Classification



Manhattan Distance

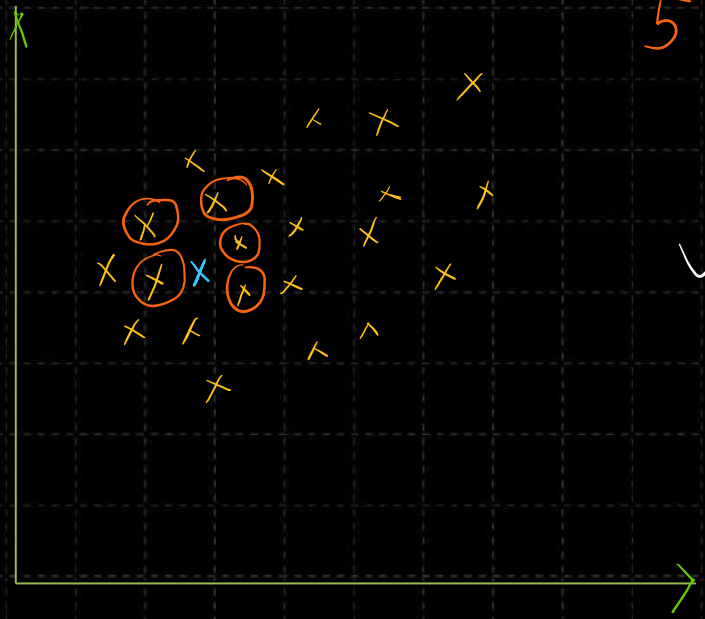


$$|x_2 - x_1| + |y_2 - y_1|$$

L1 Norm

K Nearest Neighbour - KNN Algorithm

Regression



5 Nearest Neighbours

$k = 5$

Value of k varies

K Nearest Neighbour - KNN Algorithm

KNN is not performing well under 2 situations

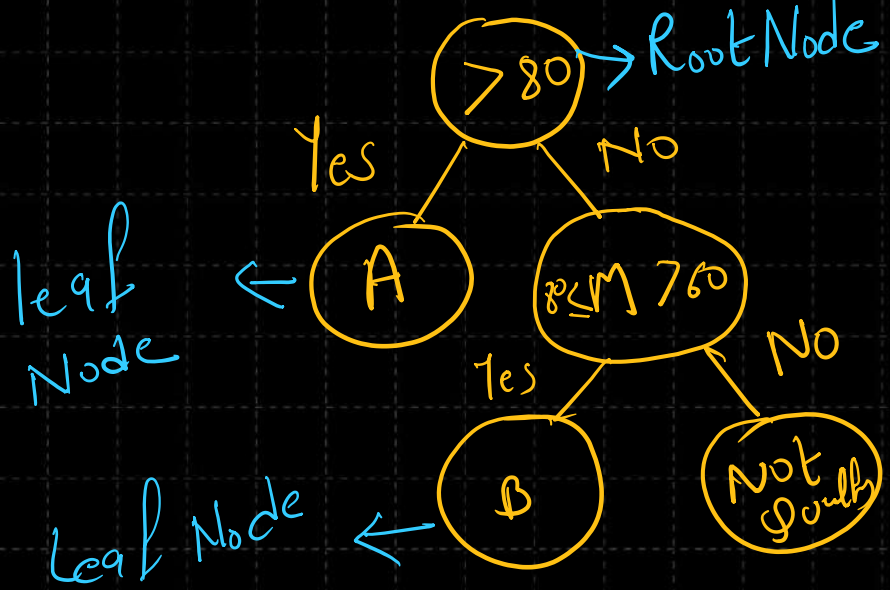
1) High Sensitivity to Outliers
2) Imbalanced data

If you have 90 points of "Class A" and 10 points of "Class B," any new point is mathematically more likely to have "Class A" neighbors, even if it actually belongs to "Class B.

Decision Tree Algorithm

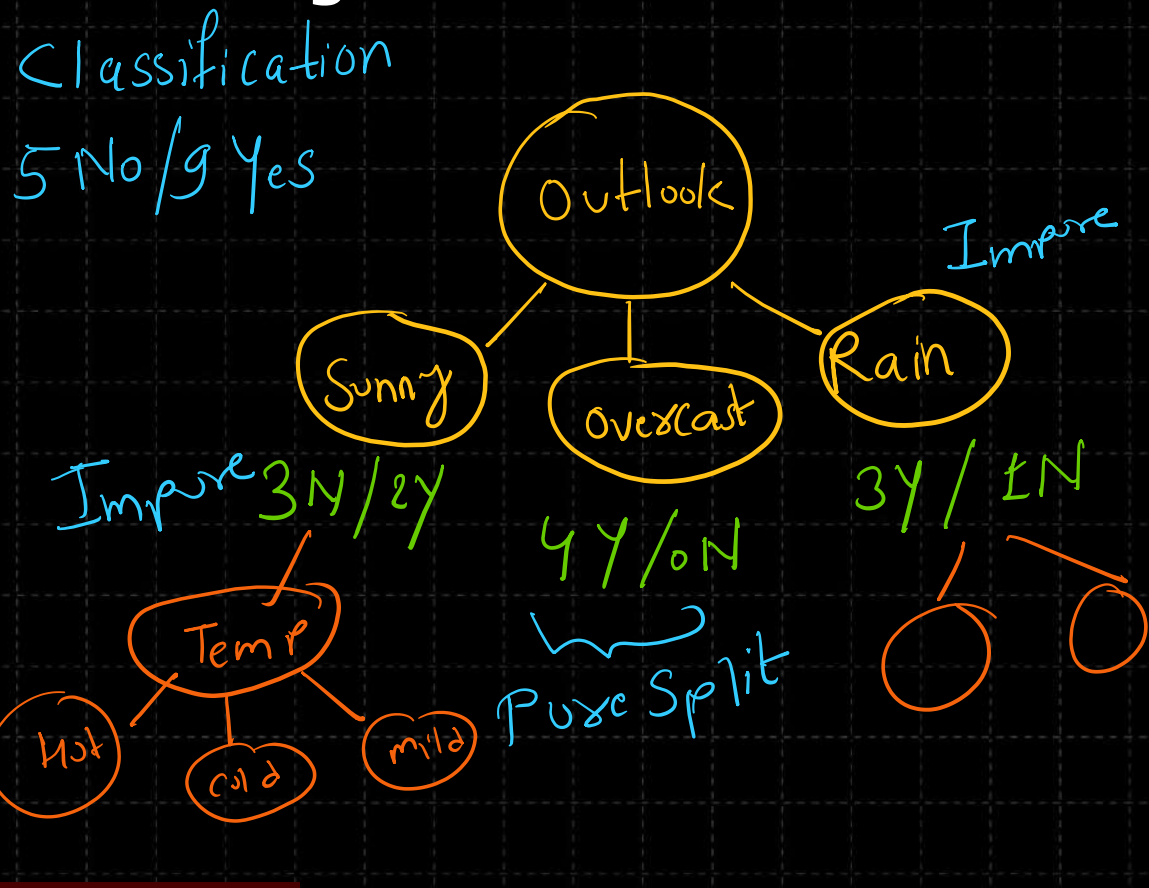
A **Decision Tree** is a non-parametric supervised learning algorithm used for both classification and regression. Think of it as a flowchart where each internal node represents a "test" on an attribute, each branch represents the outcome of the test, and each leaf node represents a final decision

If Marks > 80
    A Grade
Elif 80 <= Marks > 60
    B Grade
Else
    Not Qualify

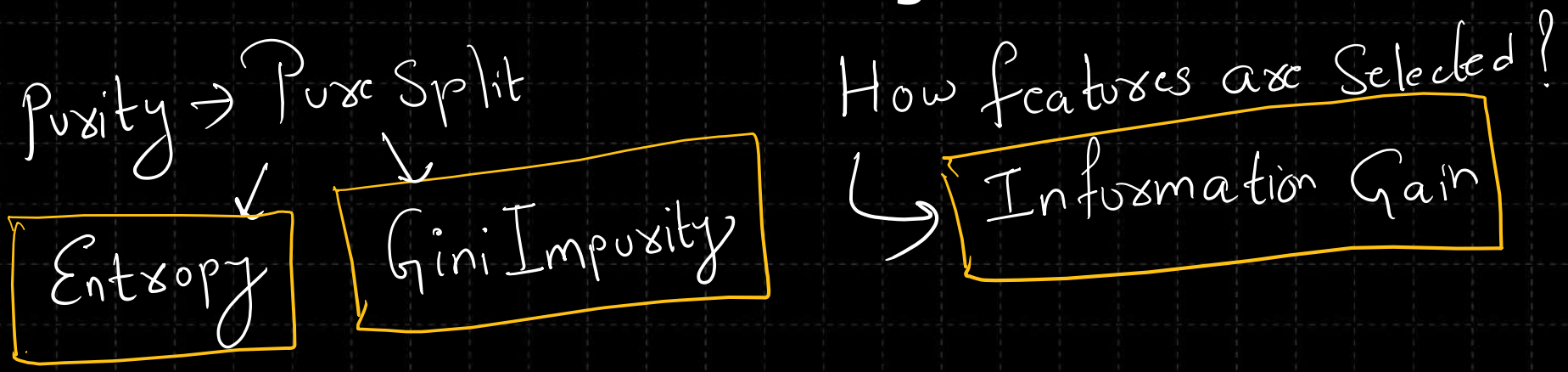


Decision Tree Algorithm

Outlook	Temperatu	Humidity	Windy	Play
Sunny	Hot	High	FALSE	No
Sunny	Hot	High	TRUE	No
Overcast	Hot	High	FALSE	Yes
Rain	Mild	High	FALSE	Yes
Rain	Cool	Normal	FALSE	Yes
Rain	Cool	Normal	TRUE	No
Overcast	Cool	Normal	TRUE	Yes
Sunny	Mild	High	FALSE	No
Sunny	Cool	Normal	FALSE	Yes
Rain	Mild	Normal	FALSE	Yes
Sunny	Mild	Normal	TRUE	Yes
Overcast	Mild	High	TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Rain	Mild	High	TRUE	No



Decision Tree Algorithm



Decision Tree Algorithm

THE iSCALE

Entropy

$$H(S) = -P_+ \log_2 P_+ - P_- \log_2 P_-$$

$P_+ \rightarrow$ Probability of Yes

$P_- \rightarrow$ Probability of No

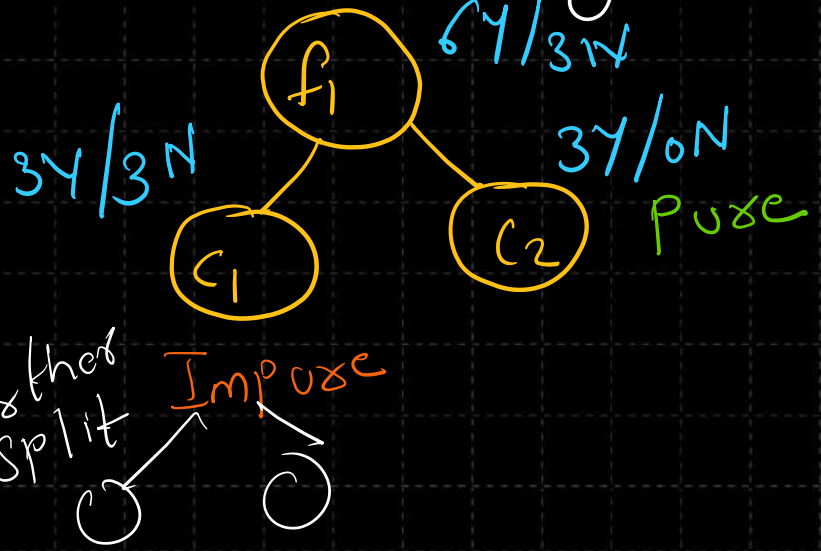
$$H(S) = -\frac{3}{6}\log_2\frac{3}{6} - \frac{3}{6}\log_2\frac{3}{6}$$

$$\boxed{H(S) = 1} \rightarrow \text{impure}$$

$$H(S) = -\frac{3}{3}\log_2\frac{3}{3} - \frac{0}{3}\log_2\frac{0}{3}$$

$$= -1\log_2 1$$

$$\boxed{H(S) = 0} \rightarrow \text{Pure}$$



Decision Tree Algorithm

THE iSCALE

1. Entropy = 0 (Minimum)
Iska matlab hai ki data **100% Pure** hai. Koi randomness ya disorder nahi hai.

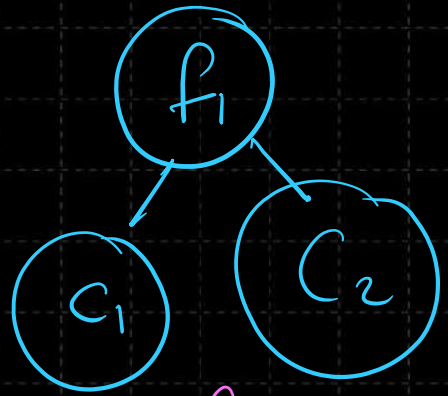
2. Entropy = 1 (Maximum)
Iska matlab hai ki data **Perfectly Impure** hai. Disorder sabse zyada hai.

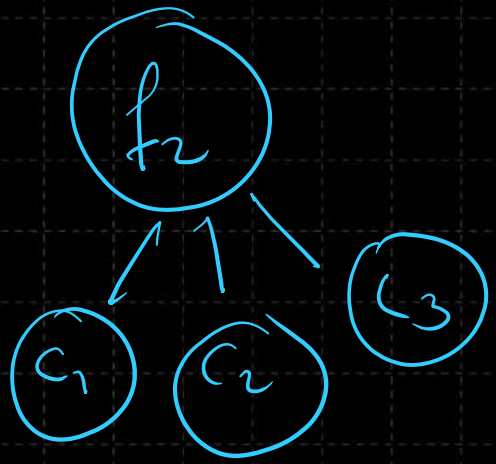
3. Entropy between 0 and 1
Agar data thoda mixed hai, toh value 0 aur 1 ke beech aayegi.

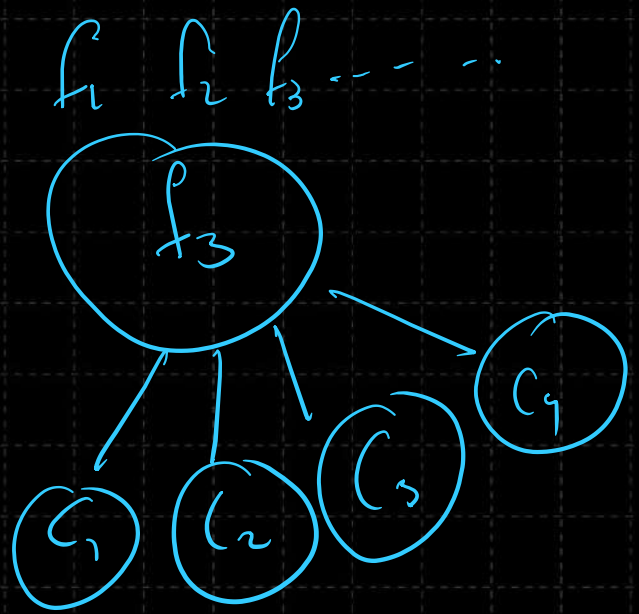
$$H(S) = -p_{yes}\log_2(p_{yes}) - p_{no}\log_2(p_{no})$$

Decision Tree Algorithm

Information Gain







Which feature to Select ? ? ?

f1, f2, f3 . . .

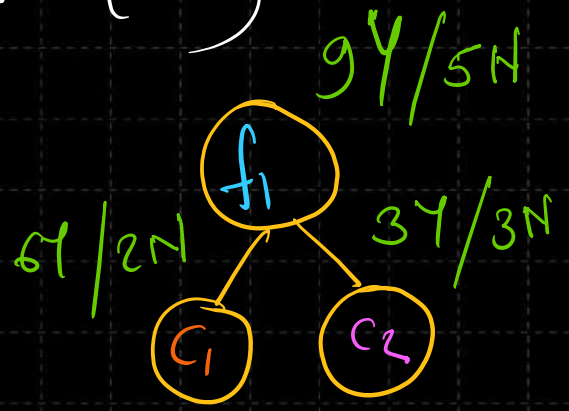
Decision Tree Algorithm

Information Gain

$$Gain(S, f_1) = H(S) - \sum_{v \in value} \frac{|S_v|}{|S|} H(S_v)$$

$$H(S) = -9/14 \log_2 9/14 - 5/14 \log_2 5/14$$

$$H(S) = 0.94$$



$$C_1 \rightarrow H(S) = -6/8 \log_2 6/8 - 2/8 \log_2 2/8$$

$$C_1 \rightarrow H(S) = 0.81$$

$$C_2 \rightarrow H(S) = -3/6 \log_2 3/6 - 3/6 \log_2 3/6$$

$$C_2 \rightarrow H(S) = 1$$

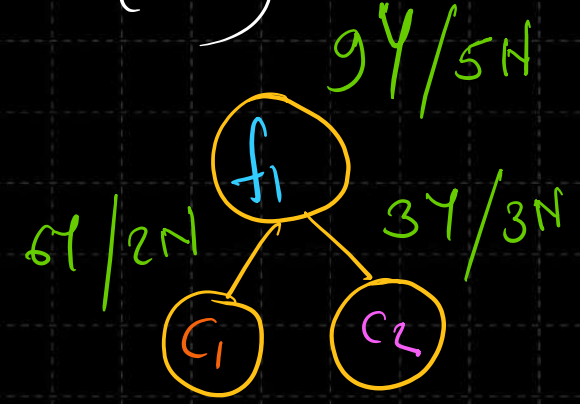
Decision Tree Algorithm

Information Gain

$$Gain(S, f_1) = H(S) - \sum_{v \in value} \frac{|S_v|}{|S|} H(S_v)$$

$H(S) = 0.94$ $\quad H(S) \to C_1\ 0.81$ $\quad H(S) C_2 \to 1$

$$gain = 0.94 - \left[\frac{8}{14}(0.81) + \frac{6}{14}(1) \right]$$



(f1) gain = 0.049

(f2) gain = 0.051

(f2) → feature Selected

Decision Tree Algorithm

Gini Impurity

$$GI = 1 - \sum_{i=1}^{n} (p)^2$$

$$GI = 1 - [(p_Y)^2 + (p_N)^2]$$

$$GI = 1 - [(1/2)^2 + (1/2)^2]$$

$$\boxed{GI = 0.5}$$

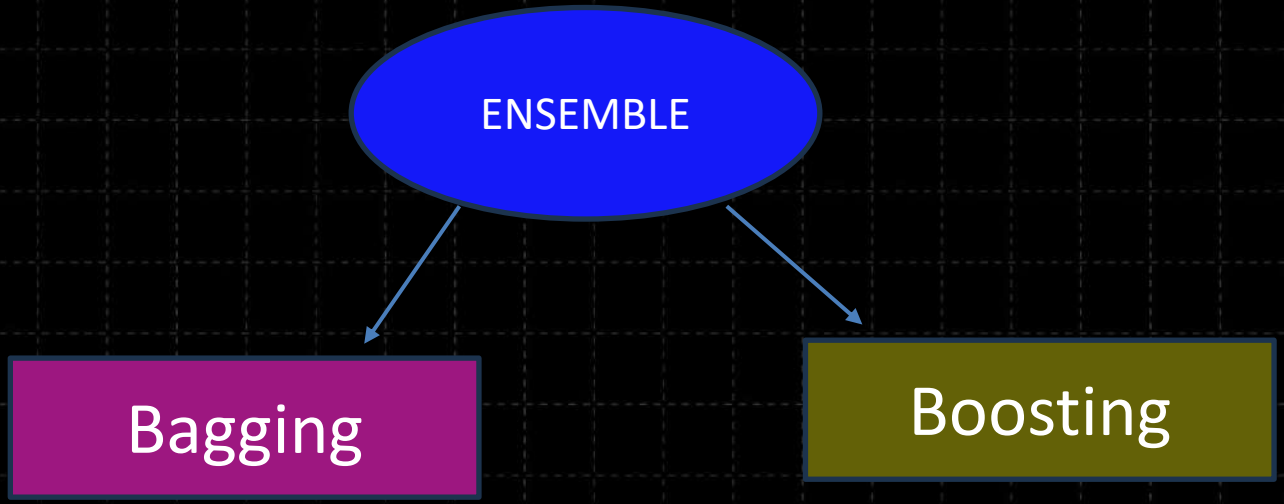
Decision Tree Algorithm

Feature	Gini Impurity	Entropy
Default Choice	Scikit-learn ka default hai.	Jab complex patterns samajhne ho tab use karein.
Speed	**Zyada Fast hai.**	Thoda Slow hai.
Mathematics	Isme sirf "Squaring" hoti hai.	Isme "Logarithm" (log) calculate hota hai.
Accuracy	Dono ka result lagbhag same aata hai.	Kabhi-kabhi thoda zyada balanced tree banata hai.

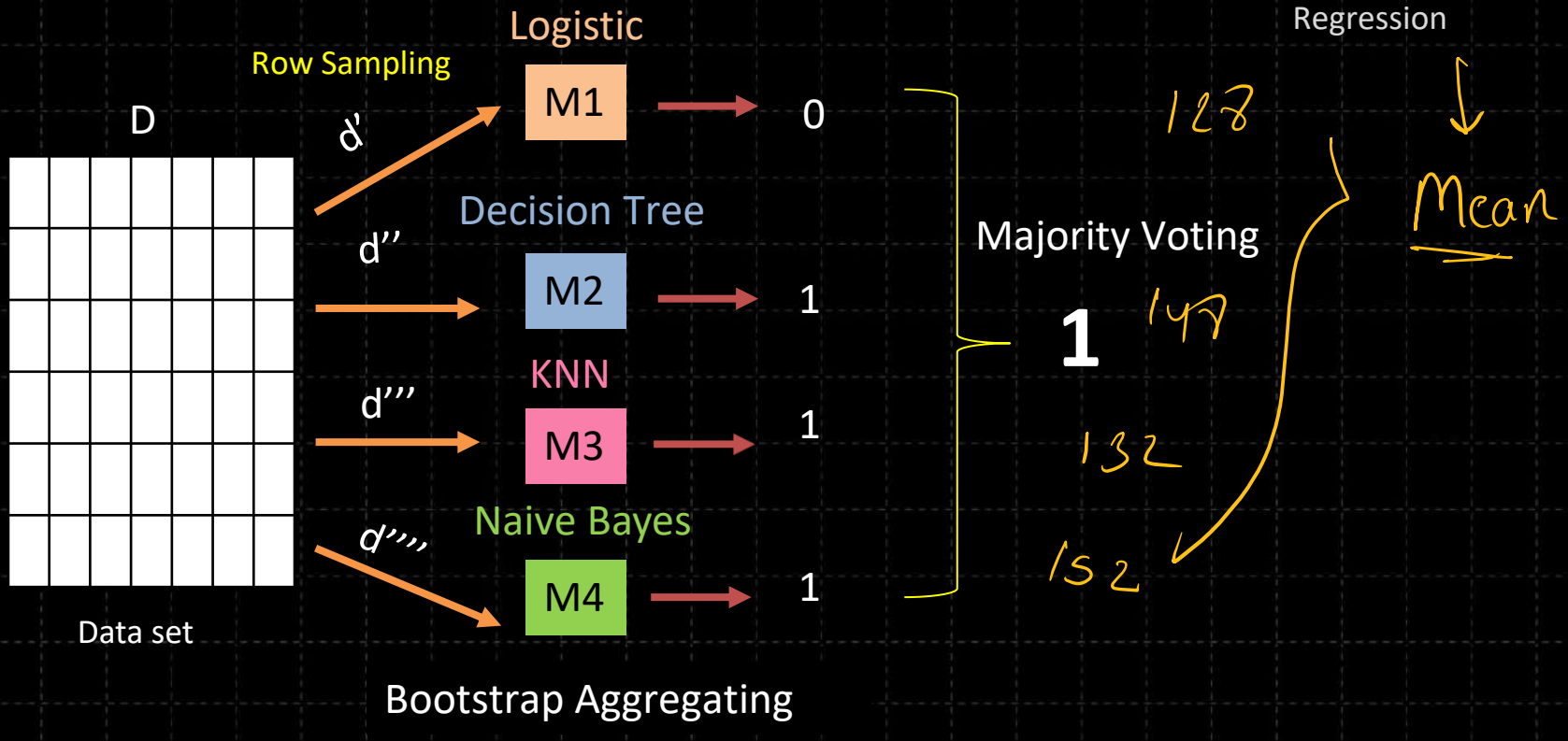
Ensemble Technique

Can we use Multiple Algorithm to solve a problem ?

YES



Ensemble Technique - Bagging



Ensemble Technique - Boosting

Random Forest Algorithm - Bagging

THE iSCALE

Both for Classification & Regression

Decision Tree

Row Sampling & Feature

D

d'

d''

d'''

d''''

DT 1

DT 2

DT 3

DT 4

0

1

1

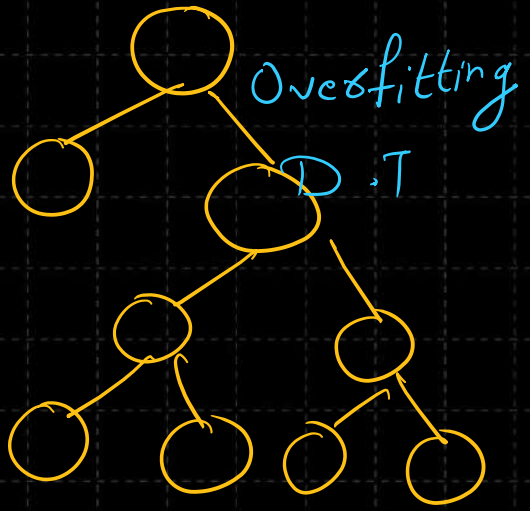
1

Majority Voting

1

Data set

Bootstrap Aggregating

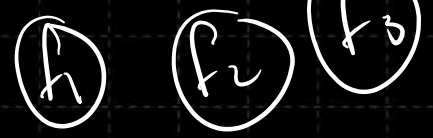


for Regression → Mean

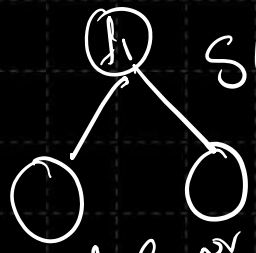
Adaboost Algorithm – Boosting

f_1	f_2	f_3	f_4	o/p	Weight
–	–	–	–	Y	1/7
–	–	–	–	N	1/7
–	–	–	–	N	1/7
–	–	–	–	Y	1/7
–	–	–	–	Y	1/7
–	–	–	–	Y	1/7
–	–	–	–	N	1/7
–	–	–	–	Y	1/7
–	–	–	–	Y	1/7
–	–	–	–	N	1/7

Error



Information gain



Total Error = 1/7

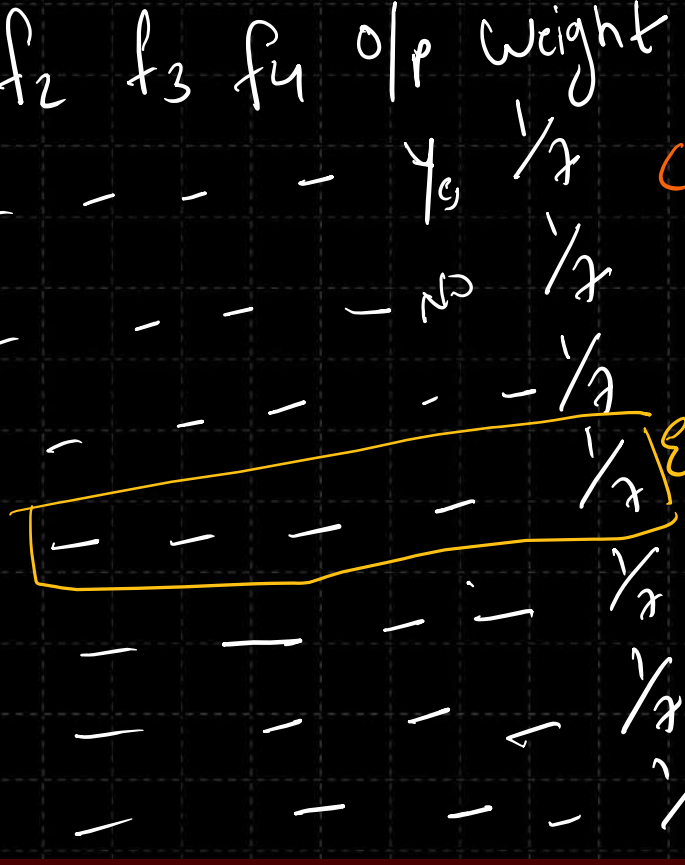
(2) Performance of Stump

$$= \frac{1}{2}\log_e\left(\frac{1-TE}{TE}\right)$$

$$= \frac{1}{2}\log_2\left(\frac{1-1/7}{1/7}\right)$$

$$= 0.895$$

Adaboost Algorithm – Boosting



Correct weight
$= \text{weight} \times e^{-PS}$
$= \frac{1}{7} \times e^{-0.895}$
$= 0.05$

Error

Incorrect weight
$= \text{weight} \times e^{PS}$
$= \frac{1}{7} \times e^{0.895}$
$= 0.349$

(f3) Information gain

(2) Performance of stump
$= \frac{1}{2} \log_e\left(\frac{1-TE}{TE}\right)$
$= 0.895$

Adaboost Algorithm – Boosting

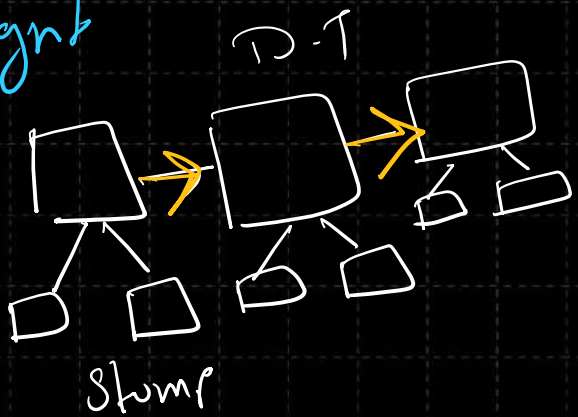
f1 f2 f3 f4 o/p Weight

				Y	1/7
				N	1/7
					1/7
					1/7
					1/7
					1/7
					1/7
					1/7

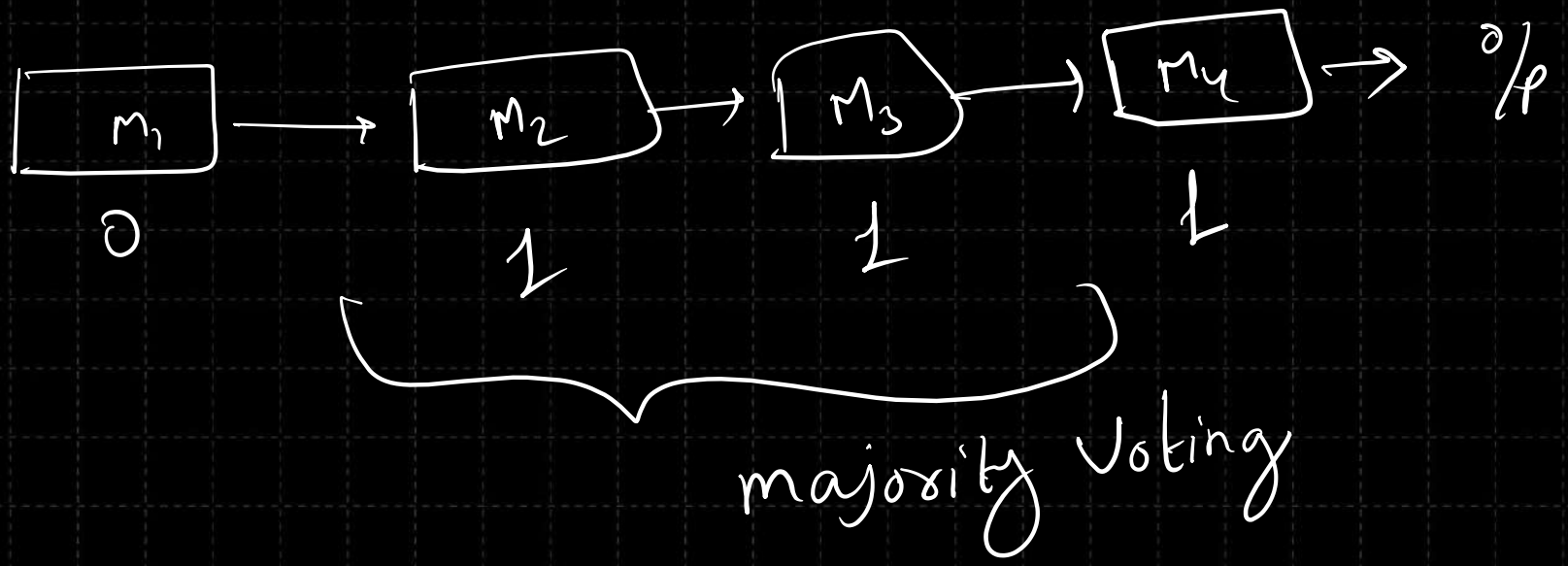
Error

New weight

0.05
0.05
0.05
0.349
0.05
0.05
0.05

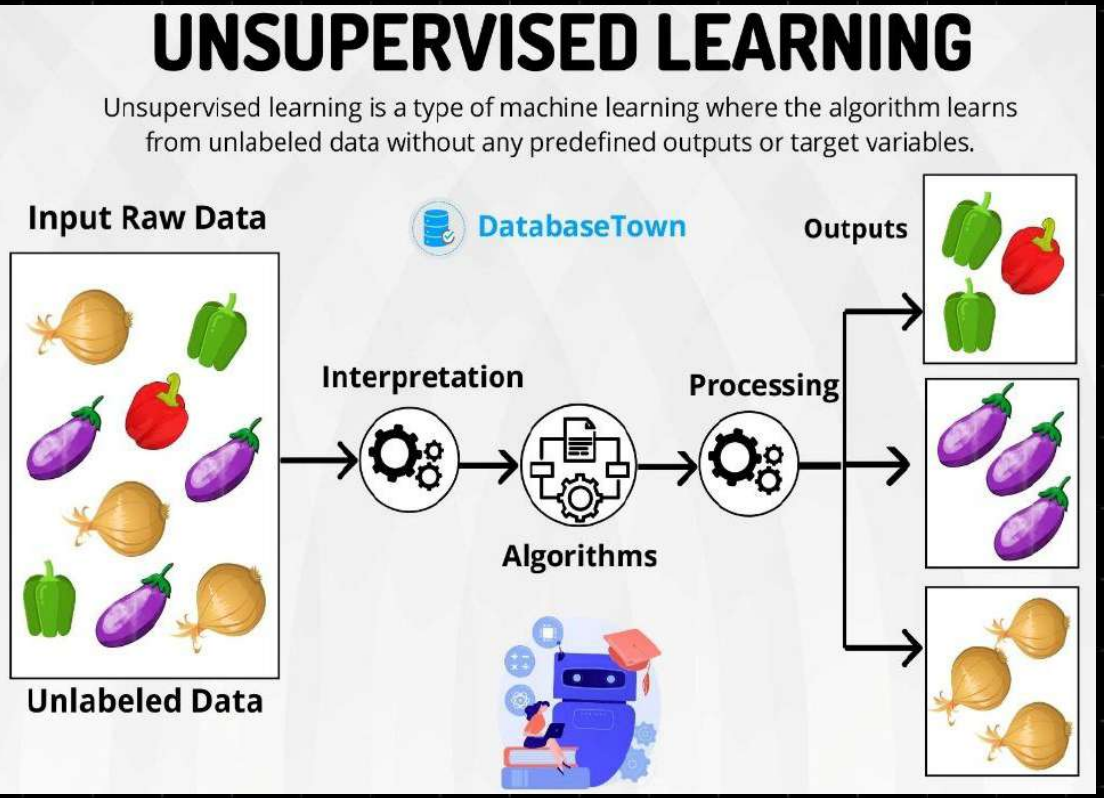


Adaboost Algorithm – Boosting



Regression → Mean

Unsupervised Machine Learning



Jab aapke paas data hota hai, lekin uske saath koi label (output) nahi hota — tab hum unsupervised learning use karte hain. 📌 Yani model ko sirf input diya jaata hai, aur model khud pattern find karta hai.

K Means Clustering Algorithm

K-Means Clustering unsupervised learning ka sabse popular algorithm hai. Iska kaam hai unlabeled data ko K groups (clusters) mein baantna, jahan har group ke andar ke points ek dusre se milte-julte hote hain.

Don't have Any Specific o/p

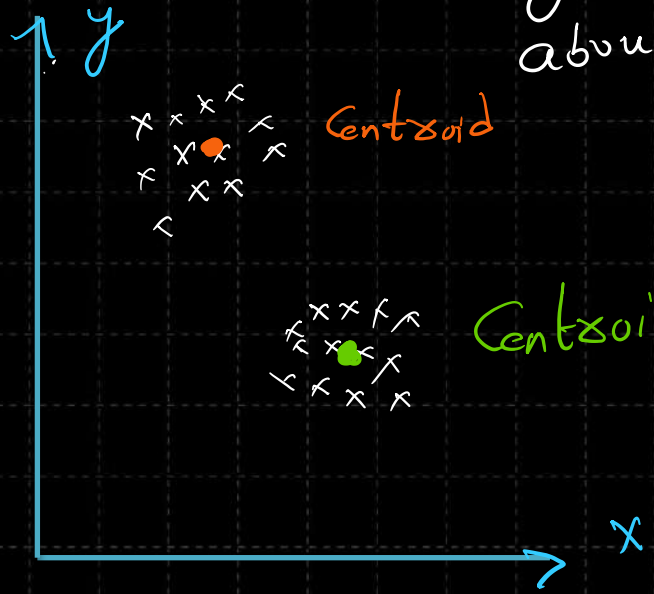
f_1 f_2

Based on Data we Create Clusters

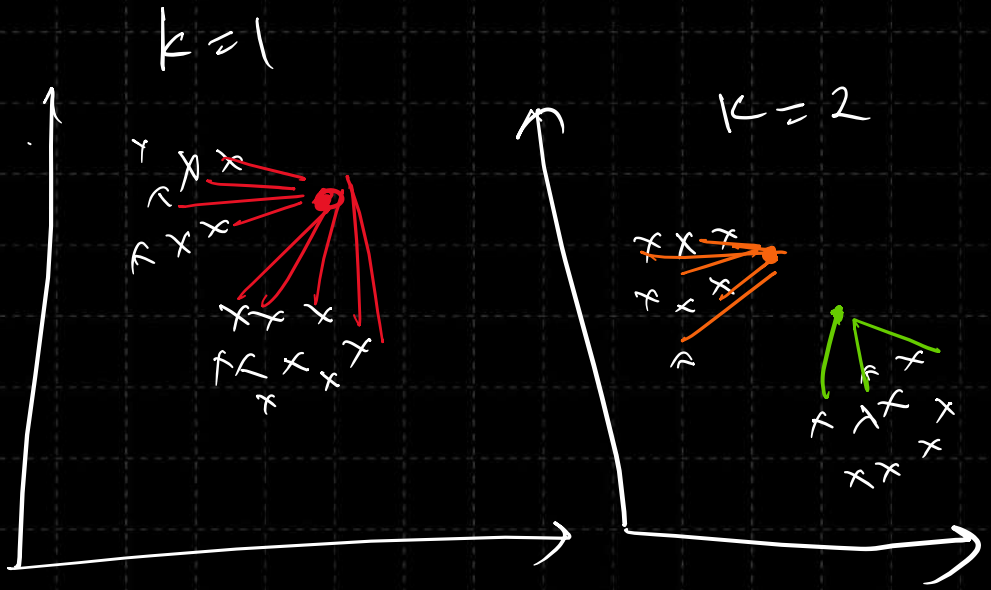
K Means Clustering Algorithm

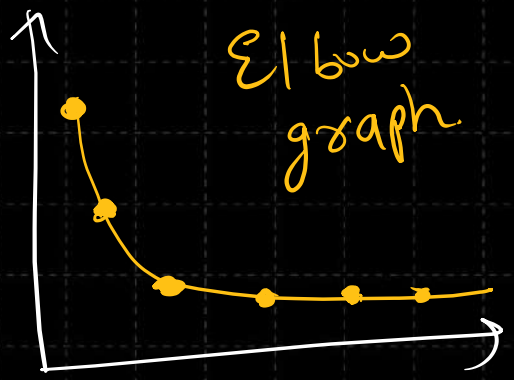
Centroids

How you will come to know about 2 groups ?



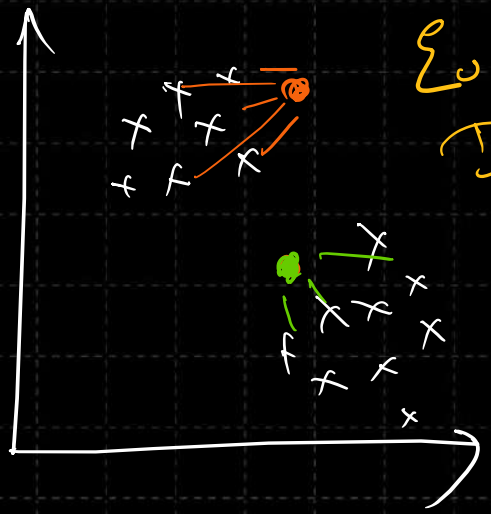
K Means Clustering Algorithm





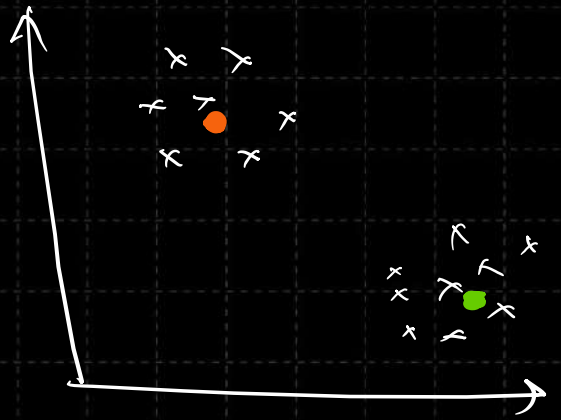
K Means Clustering Algorithm

1) We toy k values, k=2
2) Initialize k Number of Centroids
3) Calculate Avg to update Centroids

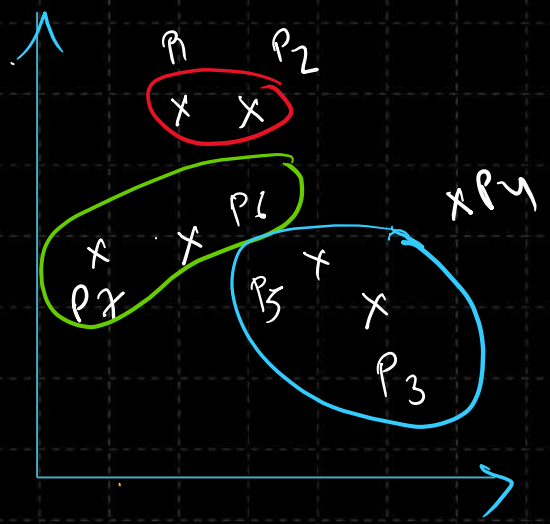


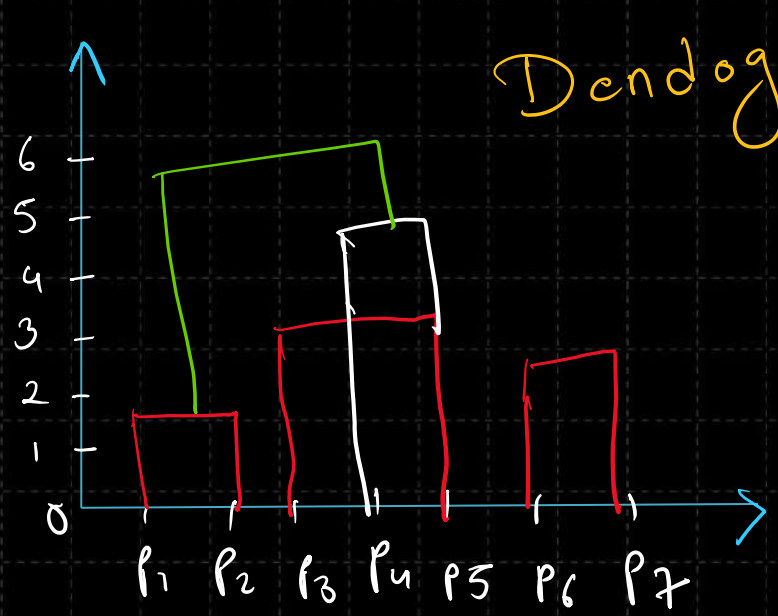
Euclidian Distance

$$\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$$



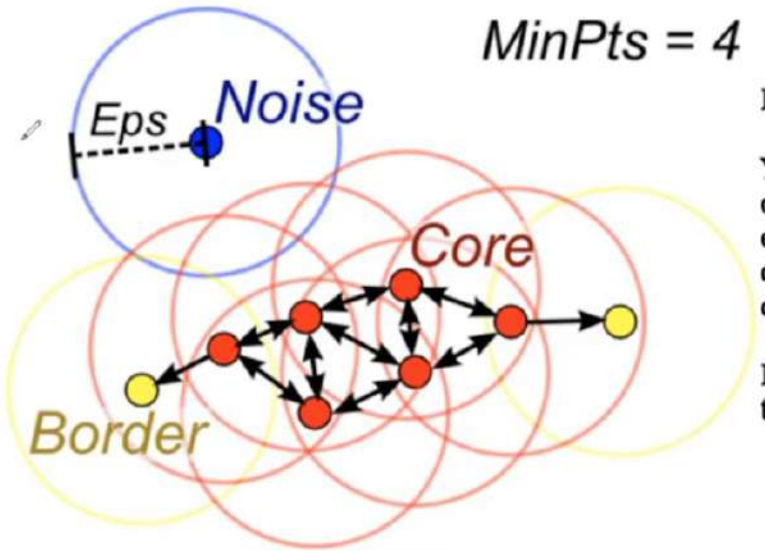
Hierarchical Clustering Algorithm





DBSCAN Clustering Algorithm

Density-Based Spatial Clustering of Applications with Noise(DBSCAN)



Red: Core Points

Yellow: Border points. Still part of the cluster because it's within epsilon of a core point, but not does not meet the min_points criteria

Blue: Noise point. Not assigned to a cluster

DBSCAN Clustering Algorithm

DBSCAN ka full form hai **Density-Based Spatial Clustering of Applications with Noise**.
K-Means aur Hierarchical clustering hamesha points ke beech ka "distance" dekhte hain, lekin DBSCAN data ki **"Density"** (ghanta) dekhta hai. Yeh un points ko ek group mein daalta hai jo bahut ghane base hue hain, aur jo points akele (isolated) hote hain, unhe **Outliers** ya **Noise** ghoshit kar deta hai.

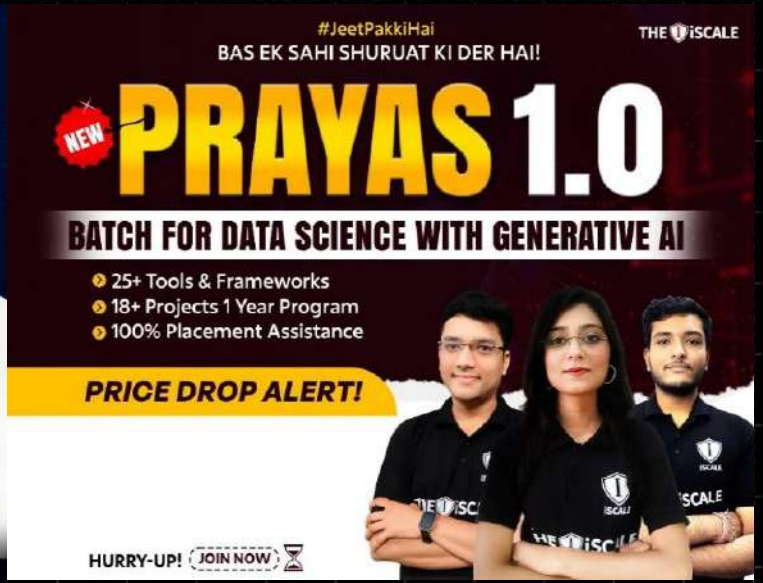
DBSCAN ko chalane ke liye humein 2 cheezein batani padti hain:

1. **Epsilon (ϵ):** Yeh ek radius (circle ka size) hai. Hum har point ke charo taraf itni doori tak check karte hain ki koi dusra point hai ya nahi.
2. **MinPoints:** Ek circle ke andar kam se kam kitne points hone chahiye taaki hum use ek "Ghanan" (Dense) area maanein.

DBSCAN har data point ko teen categories mein baant-ta hai:

- **Core Point:** Agar kisi point ke ϵ radius ke andar kam se kam `MinPoints` jitne padosi (neighbors) hain, toh wo ek **Core Point** hai. Yeh cluster ka "Dil" hota hai.
- **Border Point:** Agar kisi point ke radius mein `MinPoints` se kam points hain, lekin wo kisi Core Point ka padosi hai, toh wo **Border Point** hai.
- **Noise Point (Outlier):** Jo na Core point hai aur na hi kisi Core point ke paas hai, wo **Noise** hai.

Courses of
AI & Data Science, Data Analytics
Contact us - 7880113112



Connect with us

Instagram Handle

theiscale

The iScale Organization Handle

theiscale.founders

Siblings - Nishant Dhote & Swati Dhote

Linear Regression practical implementaion

In [56]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [58]:
```
from sklearn.datasets import fetch_openml
boston = fetch_openml(name = 'boston', version = 1, as_frame = True, parser = 'pand
```

In [60]:
```
boston
```

Out[60]:
```
{'data':         CRIM    ZN  INDUS CHAS    NOX     RM   AGE     DIS RAD    TAX  \
0    0.00632  18.0   2.31    0  0.538  6.575  65.2  4.0900   1  296.0
1    0.02731   0.0   7.07    0  0.469  6.421  78.9  4.9671   2  242.0
2    0.02729   0.0   7.07    0  0.469  7.185  61.1  4.9671   2  242.0
3    0.03237   0.0   2.18    0  0.458  6.998  45.8  6.0622   3  222.0
4    0.06905   0.0   2.18    0  0.458  7.147  54.2  6.0622   3  222.0
..       ...   ...    ...  ...    ...    ...   ...     ...  ..    ...
501  0.06263   0.0  11.93    0  0.573  6.593  69.1  2.4786   1  273.0
502  0.04527   0.0  11.93    0  0.573  6.120  76.7  2.2875   1  273.0
503  0.06076   0.0  11.93    0  0.573  6.976  91.0  2.1675   1  273.0
504  0.10959   0.0  11.93    0  0.573  6.794  89.3  2.3889   1  273.0
505  0.04741   0.0  11.93    0  0.573  6.030  80.8  2.5050   1  273.0

     PTRATIO       B  LSTAT
0       15.3  396.90   4.98
1       17.8  396.90   9.14
2       17.8  392.83   4.03
3       18.7  394.63   2.94
4       18.7  396.90   5.33
..       ...     ...    ...
501     21.0  391.99   9.67
502     21.0  396.90   9.08
503     21.0  396.90   5.64
504     21.0  393.45   6.48
505     21.0  396.90   7.88

[506 rows x 13 columns],
'target': 0      24.0
1      21.6
2      34.7
3      33.4
4      36.2
       ...
501    22.4
502    20.6
503    23.9
504    22.0
505    11.9
Name: MEDV, Length: 506, dtype: float64,
'frame':         CRIM    ZN  INDUS CHAS    NOX     RM   AGE     DIS RAD    TAX  \
0    0.00632  18.0   2.31    0  0.538  6.575  65.2  4.0900   1  296.0
1    0.02731   0.0   7.07    0  0.469  6.421  78.9  4.9671   2  242.0
2    0.02729   0.0   7.07    0  0.469  7.185  61.1  4.9671   2  242.0
3    0.03237   0.0   2.18    0  0.458  6.998  45.8  6.0622   3  222.0
4    0.06905   0.0   2.18    0  0.458  7.147  54.2  6.0622   3  222.0
..       ...   ...    ...  ...    ...    ...   ...     ...  ..    ...
501  0.06263   0.0  11.93    0  0.573  6.593  69.1  2.4786   1  273.0
502  0.04527   0.0  11.93    0  0.573  6.120  76.7  2.2875   1  273.0
503  0.06076   0.0  11.93    0  0.573  6.976  91.0  2.1675   1  273.0
504  0.10959   0.0  11.93    0  0.573  6.794  89.3  2.3889   1  273.0
505  0.04741   0.0  11.93    0  0.573  6.030  80.8  2.5050   1  273.0

     PTRATIO       B  LSTAT  MEDV
0       15.3  396.90   4.98  24.0
1       17.8  396.90   9.14  21.6
2       17.8  392.83   4.03  34.7
```

```
3       18.7  394.63   2.94  33.4
4       18.7  396.90   5.33  36.2
..       ...     ...    ...   ...
501     21.0  391.99   9.67  22.4
502     21.0  396.90   9.08  20.6
503     21.0  396.90   5.64  23.9
504     21.0  393.45   6.48  22.0
505     21.0  396.90   7.88  11.9

[506 rows x 14 columns],
'categories': None,
'feature_names': ['CRIM',
 'ZN',
 'INDUS',
 'CHAS',
 'NOX',
 'RM',
 'AGE',
 'DIS',
 'RAD',
 'TAX',
 'PTRATIO',
 'B',
 'LSTAT'],
'target_names': ['MEDV'],
'DESCR': "**Author**:   \n**Source**: Unknown - Date unknown  \n**Please cite**:
\n\nThe Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic\npric
es and the demand for clean air', J. Environ. Economics & Management,\nvol.5, 81-1
02, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics\n...', Wiley, 1
980.   N.B. Various transformations are used in the table on\npages 244-261 of the
latter.\nVariables in order:\nCRIM     per capita crime rate by town\nZN       pro
portion of residential land zoned for lots over 25,000 sq.ft.\nINDUS    proportion
of non-retail business acres per town\nCHAS     Charles River dummy variable (= 1
if tract bounds river; 0 otherwise)\nNOX      nitric oxides concentration (parts p
er 10 million)\nRM       average number of rooms per dwelling\nAGE      proportion
of owner-occupied units built prior to 1940\nDIS      weighted distances to five B
oston employment centres\nRAD      index of accessibility to radial highways\nTAX
full-value property-tax rate per $10,000\nPTRATIO  pupil-teacher ratio by town\nB
1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town\nLSTAT    % lower s
tatus of the population\nMEDV     Median value of owner-occupied homes in $1000's
\n\n\nInformation about the dataset\nCLASSTYPE: numeric\nCLASSINDEX: last\n\nDownl
oaded from openml.org.",
'details': {'id': '531',
 'name': 'boston',
 'version': '1',
 'description_version': '1',
 'format': 'ARFF',
 'creator': ['D. and Rubinfeld', "D.L. 'Hedonic"],
 'collection_date': '1978',
 'upload_date': '2014-09-29T00:08:07',
 'language': 'English',
 'licence': 'Public',
 'url': 'https://openml.org/data/v1/download/52643/boston.arff',
 'parquet_url': 'https://data.openml.org/datasets/0000/0531/dataset_531.pq',
 'file_id': '52643',
 'default_target_attribute': 'MEDV',
```

```
 'tag': ['Data Science',
  'Housing Economics',
  'Kaggle',
  'OpenML-Reg19',
  'study_130',
  'Urban Studies'],
 'visibility': 'public',
 'status': 'active',
 'processing_date': '2020-11-20 20:16:37',
 'md5_checksum': 'cdd361fb886627eaa80c92f90d0610cc'},
'url': 'https://www.openml.org/d/531'}
```

In [62]:
```
df = boston.frame
```

In [64]:
```
df.head()
```

Out[64]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90

In [66]:
```
df['price'] = boston.target
df.head()
```

Out[66]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90

Divding the dataset into dependent and independent features

In [69]:
```
X =  df.iloc[ :,:-1]
```

In [71]:
```
X.head()
```

Out[71]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90

In [73]:
```
y = df.iloc[:,-1]
```

In [75]:
```
y.head()
```

Out[75]:
```
0    24.0
1    21.6
2    34.7
3    33.4
4    36.2
Name: price, dtype: float64
```

Linear Regression

In [78]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
X = X.astype(float)
y = y.astype(float)

lin_reg = LinearRegression()
mse = cross_val_score(lin_reg, X,y, scoring  = 'neg_mean_squared_error', cv = 5)
print(mse)
```

```
[-5.43385953e-28 -4.77534215e-28 -4.48635350e-29 -1.04874762e-27
 -7.26047270e-27]
```

In [80]:
```
mean_mse = np.mean(mse)

print("MSE scores per fold:", mse)
print('Average Negative MSE', mean_mse)
```

```
MSE scores per fold: [-5.43385953e-28 -4.77534215e-28 -4.48635350e-29 -1.04874762e-2
7
 -7.26047270e-27]
Average Negative MSE -1.8750008035126038e-27
```

In [82]:
```
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_st
lin_reg.fit(X_train, y_train)
```

Out[82]:
```
LinearRegression
LinearRegression()
```

In [84]:
```
prediction = lin_reg.predict(X_test)
```

In [86]:
```
print(prediction)
```

```
[23.6 32.4 13.6 22.8 16.1 20.  17.8 14.  19.6 16.8 21.5 18.9  7.  21.2
 18.5 29.8 18.8 10.2 50.  14.1 25.2 29.1 12.7 22.4 14.2 13.8 20.3 14.9
 21.7 18.3 23.1 23.8 15.  20.8 19.1 19.4 34.7 19.5 24.4 23.4 19.7 28.2
 50.  17.4 22.6 15.1 13.1 24.2 19.9 24.  18.9 35.4 15.2 26.5 43.5 21.2
 18.4 28.5 23.9 18.5 25.  35.4 31.5 20.2 24.1 20.  13.1 24.8 30.8 12.7
 20.  23.7 10.8 20.6 20.8  5.  20.1 48.5 10.9  7.  20.9 17.2 20.9  9.7
 19.4 29.  16.4 25.  25.  17.1 23.2 10.4 19.6 17.2 27.5 23.  50.  17.9
  9.6 17.2 22.5 21.4]
```

In [88]:
```
# 4. Compare a few actual values vs predicted values
import pandas as pd
comparison_df = pd.DataFrame({'Actual': y_test, 'Predicted': prediction})
print(comparison_df.head())
```

```
     Actual  Predicted
173    23.6       23.6
274    32.4       32.4
491    13.6       13.6
72     22.8       22.8
452    16.1       16.1
```

In [90]:
```
from sklearn import metrics
import numpy as np

# 1. R-squared Score (Best possible score is 1.0)
r2 = metrics.r2_score(y_test, prediction)

# 2. Mean Absolute Error (Average error in 'price' units)
mae = metrics.mean_absolute_error(y_test, prediction)

# 3. Root Mean Squared Error (Standard deviation of prediction errors)
rmse = np.sqrt(metrics.mean_squared_error(y_test, prediction))

print(f"R-squared Score: {r2:.4f}")
print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
```

```
R-squared Score: 1.0000
Mean Absolute Error (MAE): 0.0000
Root Mean Squared Error (RMSE): 0.0000
```

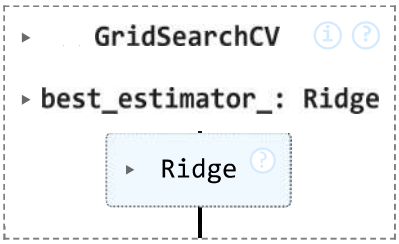
Ridge Regression

In [107…
```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
```

```
ridge = Ridge()
```

In [109…
```
params = {'alpha':[1e-15, 1e-10, 1e-8, 1e-3, 1e-2,1,5,10,20]}

ridge_regressor = GridSearchCV(ridge,params, scoring = 'neg_mean_squared_error', cv
ridge_regressor.fit(X,y)
```

Out[109…



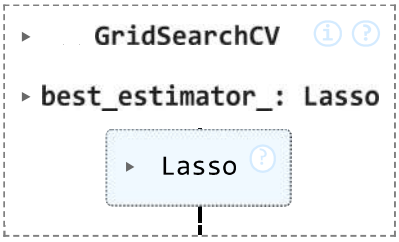
In [111…
```
print(ridge_regressor.best_params_)
print(ridge_regressor.best_score_)
```

```
{'alpha': 1e-15}
-1.8978428215872376e-28
```

Lasso Regression

In [114…
```
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
lasso = Lasso()
params = {'alpha':[1e-15, 1e-10, 1e-8, 1e-3, 1e-2,1,5,10,20]}

lasso_regressor = GridSearchCV(lasso,params, scoring = 'neg_mean_squared_error', cv
lasso_regressor.fit(X,y)
```

Out[114…



In [116…
```
print(lasso_regressor.best_params_)
print(lasso_regressor.best_score_)
```

```
{'alpha': 1e-08}
-1.062608588948593e-07
```

In [118…
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score

# Ensure data is numeric
X = X.astype(float)
y = y.astype(float)
```

```
# 1. Scale the features (Essential for Ridge and Lasso)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 2. Split into Training and Testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, ran
```

linear regression

In [121…

```
from sklearn.linear_model import LinearRegression

# Initialize and Train
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

# Predict and Calculate Accuracy
lin_pred = lin_reg.predict(X_test)
lin_acc = r2_score(y_test, lin_pred) * 100

print(f"Linear Regression Accuracy: {lin_acc:.2f}%")
```

```
Linear Regression Accuracy: 100.00%
```

Ridge Regression

In [124…

```
from sklearn.linear_model import Ridge

# Alpha is the penalty strength (higher alpha = more restriction)
ridge_reg = Ridge(alpha=10.0)
ridge_reg.fit(X_train, y_train)

# Predict and Calculate Accuracy
ridge_pred = ridge_reg.predict(X_test)
ridge_acc = r2_score(y_test, ridge_pred) * 100

print(f"Ridge Regression Accuracy: {ridge_acc:.2f}%")
```

```
Ridge Regression Accuracy: 99.76%
```

Lasso Regression

In [127…

```
from sklearn.linear_model import Lasso

# Alpha=0.1 is a common starting point for Lasso
lasso_reg = Lasso(alpha=0.1)
lasso_reg.fit(X_train, y_train)

# Predict and Calculate Accuracy
lasso_pred = lasso_reg.predict(X_test)
lasso_acc = r2_score(y_test, lasso_pred) * 100
```

```
print(f"Lasso Regression Accuracy: {lasso_acc:.2f}%")
```

```
Lasso Regression Accuracy: 99.99%
```

IOGISTIC Regression

1) Importing The libraries

In [45]:
```
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
#LogisticRegression: Yeh hamara main algorithm hai jo classification karega (Malign
from sklearn.preprocessing import StandardScaler
# StandardScaler: Numbers ko ek sahi scale par laane ke liye (jaise sabhi numbers k
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
# metrics: Model kitna achha kaam kar raha hai, yeh check karne ke tools.
```

2 Load dataset

In [22]:
```
data = load_breast_cancer()
```

In [24]:
```
data
```

Out[24]:

```
{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
         1.189e-01],
        [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
         8.902e-02],
        [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
         8.758e-02],
        ...,
        [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
         7.820e-02],
        [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
         1.240e-01],
        [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
         7.039e-02]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
        1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
        1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
        1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
        0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
        1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
        0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
        1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
        0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
        1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
        1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
        1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
        1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
        1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]),
 'frame': None,
 'target_names': array(['malignant', 'benign'], dtype='<U9'),
 'DESCR': '.. _breast_cancer_dataset:\n\nBreast cancer wisconsin (diagnostic) data
set\n--------------------------------------------\n\n**Data Set Characteristics:**
\n\n:Number of Instances: 569\n\n:Number of Attributes: 30 numeric, predictive att
ributes and the class\n\n:Attribute Information:\n    - radius (mean of distances
from center to points on the perimeter)\n    - texture (standard deviation of gray
-scale values)\n    - perimeter\n    - area\n    - smoothness (local variation in
radius lengths)\n    - compactness (perimeter^2 / area - 1.0)\n    - concavity (se
verity of concave portions of the contour)\n    - concave points (number of concav
e portions of the contour)\n    - symmetry\n    - fractal dimension ("coastline ap
proximation" - 1)\n\n    The mean, standard error, and "worst" or largest (mean of
the three\n    worst/largest values) of these features were computed for each imag
e,\n    resulting in 30 features.  For instance, field 0 is Mean Radius, field\n
10 is Radius SE, field 20 is Worst Radius.\n\n    - class:\n            - WDBC-Mal
ignant\n            - WDBC-Benign\n\n:Summary Statistics:\n\n====================
```

```
================ ====== ======\n                                        Min    Max
\n=================================== ====== ======\nradius (mean):
6.981  28.11\ntexture (mean):                      9.71   39.28\nperimeter (mea
n):                    43.79  188.5\narea (mean):                        143.5
2501.0\nsmoothness (mean):                   0.053  0.163\ncompactness (mean):
0.019  0.345\nconcavity (mean):                    0.0    0.427\nconcave points
(mean):               0.0    0.201\nsymmetry (mean):                     0.106
0.304\nfractal dimension (mean):           0.05   0.097\nradius (standard erro
r):             0.112  2.873\ntexture (standard error):            0.36   4.885
\nperimeter (standard error):          0.757  21.98\narea (standard error):
6.802  542.2\nsmoothness (standard error):          0.002  0.031\ncompactness (sta
ndard error):         0.002  0.135\nconcavity (standard error):          0.0
0.396\nconcave points (standard error):      0.0    0.053\nsymmetry (standard erro
r):           0.008  0.079\nfractal dimension (standard error):   0.001  0.03\nra
dius (worst):                       7.93   36.04\ntexture (worst):
12.02  49.54\nperimeter (worst):                   50.41  251.2\narea (worst):
185.2  4254.0\nsmoothness (worst):                  0.071  0.223\ncompactness (wo
rst):                 0.027  1.058\nconcavity (worst):                   0.0
1.252\nconcave points (worst):               0.0    0.291\nsymmetry (worst):
0.156  0.664\nfractal dimension (worst):           0.055  0.208\n================
==================== ====== ======\n\n:Missing Attribute Values: None\n\n:Class D
istribution: 212 - Malignant, 357 - Benign\n\n:Creator:  Dr. William H. Wolberg,
W. Nick Street, Olvi L. Mangasarian\n\n:Donor: Nick Street\n\n:Date: November, 199
5\n\nThis is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.\nhttp
s://goo.gl/U2Uwz2\n\nFeatures are computed from a digitized image of a fine needle
\naspirate (FNA) of a breast mass.  They describe\ncharacteristics of the cell nuc
lei present in the image.\n\nSeparating plane described above was obtained using\n
Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree\nConstruction Via
Linear Programming." Proceedings of the 4th\nMidwest Artificial Intelligence and C
ognitive Science Society,\npp. 97-101, 1992], a classification method which uses l
inear\nprogramming to construct a decision tree.  Relevant features\nwere selected
using an exhaustive search in the space of 1-4\nfeatures and 1-3 separating plane
s.\n\nThe actual linear program used to obtain the separating plane\nin the 3-dime
nsional space is that described in:\n[K. P. Bennett and O. L. Mangasarian: "Robust
Linear\nProgramming Discrimination of Two Linearly Inseparable Sets",\nOptimizatio
n Methods and Software 1, 1992, 23-34].\n\nThis database is also available through
the UW CS ftp server:\n\nftp ftp.cs.wisc.edu\ncd math-prog/cpo-dataset/machine-lea
rn/WDBC/\n\n.. dropdown:: References\n\n  - W.N. Street, W.H. Wolberg and O.L. Man
gasarian. Nuclear feature extraction\n    for breast tumor diagnosis. IS&T/SPIE 19
93 International Symposium on\n    Electronic Imaging: Science and Technology, vol
ume 1905, pages 861-870,\n    San Jose, CA, 1993.\n  - O.L. Mangasarian, W.N. Stre
et and W.H. Wolberg. Breast cancer diagnosis and\n    prognosis via linear program
ming. Operations Research, 43(4), pages 570-577,\n    July-August 1995.\n  - W.H.
Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques\n    to di
agnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994)\n    163
-171.\n',
 'feature_names': array(['mean radius', 'mean texture', 'mean perimeter', 'mean ar
ea',
        'mean smoothness', 'mean compactness', 'mean concavity',
        'mean concave points', 'mean symmetry', 'mean fractal dimension',
        'radius error', 'texture error', 'perimeter error', 'area error',
        'smoothness error', 'compactness error', 'concavity error',
        'concave points error', 'symmetry error',
        'fractal dimension error', 'worst radius', 'worst texture',
        'worst perimeter', 'worst area', 'worst smoothness',
        'worst compactness', 'worst concavity', 'worst concave points',
```

```
       'worst symmetry', 'worst fractal dimension'], dtype='<U23'),
 'filename': 'breast_cancer.csv',
 'data_module': 'sklearn.datasets.data'}
```

In [26]:
```
X = pd.DataFrame(data.data, columns =data.feature_names)
```

In [28]:
```
X
```

Out[28]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	sy
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	
...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	

569 rows × 30 columns

In [30]:
```
y = data.target
```

In [33]:
```
y
```

Out[33]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1])
```

In [35]:
```
# y (Target): Yeh hamara "Answer Key" hai. 0 matlab cancer hai (Malignant) aur 1 ma
```

3 Training & Testing data split

In [39]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# test_size=0.2: Matlab 20% data humne side mein rakh diya testing ke liye, aur 80%

# random_state=42: Yeh ek seed number hai taaki har baar jab aap code run karein, t
```

4 Feature Scaling

In [42]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Problem: Kuch features bade hote hain (jaise Area = 1000) aur kuch chote (jaise S

#Solution: StandardScaler sabko ek standard range mein le aata hai taaki model conf
#fit_transform training data se seekhta hai aur badalta hai,
#jabki transform test data ko usi tarah badal deta hai.
```

5 Model training

In [47]:
```
model = LogisticRegression()
model.fit(X_train, y_train)

#LogisticRegression(): Humne ek khali model banaya.

#model.fit: Yeh asli "Padhai" (Learning) hai.
#Model X_train (sawal) aur y_train (jawab) ko dekh kar patterns samajhta hai.
```

Out[47]:

LogisticRegression

LogisticRegression()

6 Predictions

In [50]:
```
y_pred = model.predict(X_test)
```

In [52]:
```
y_pred
```

Out[52]:
```
array([1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 0, 0])
```

In [54]:
```
y_test
```

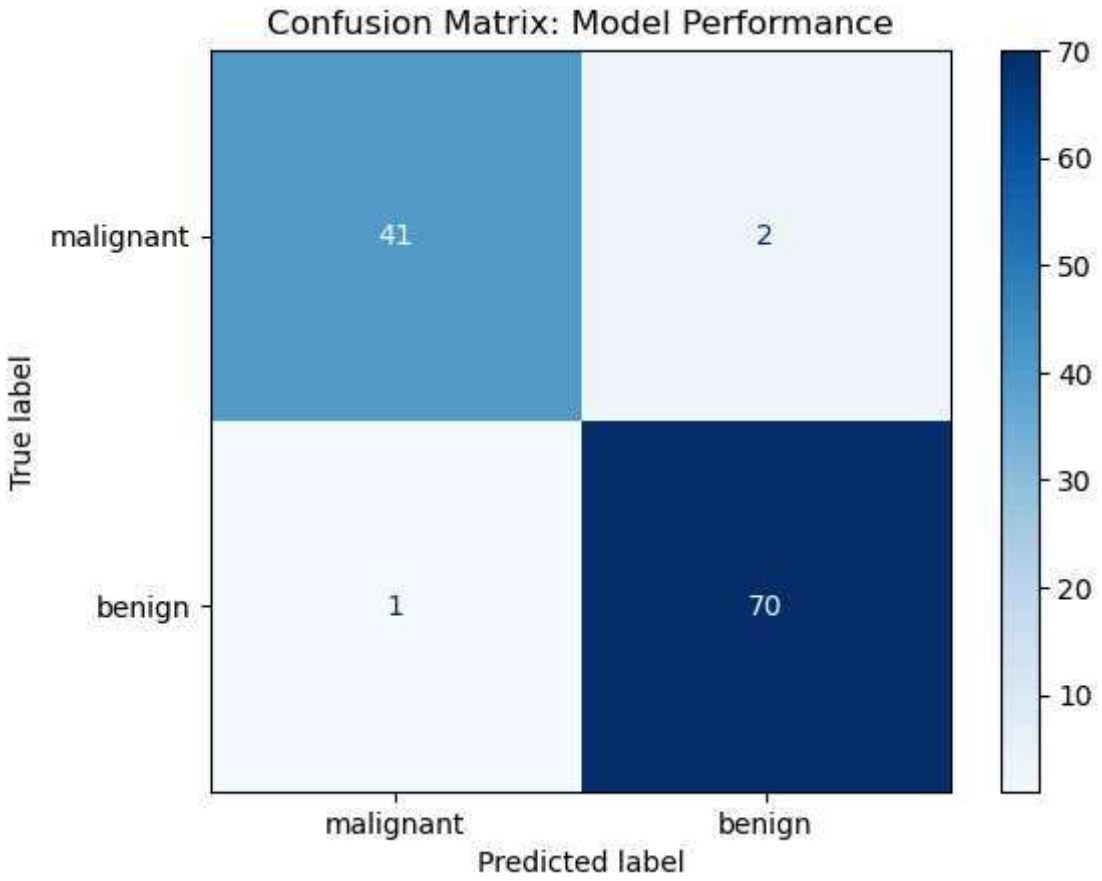
Out[54]:
```
array([1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 0])
```

7 Accuracy

In [60]:
```
print(f"Accuracy Score: {accuracy_score(y_test, y_pred):.4f}")
print(classification_report(y_test, y_pred))

#accuracy_score: Yeh batata hai ki 100 mein se kitne baar model ne sahi jawab diya.

#classification_report: Yeh detail mein batata hai ki
#model ne kitne logo ko galti se bimar bata diya (Precision)
#aur kitne asli bimaron ko pakad paya (Recall).
```

```
Accuracy Score: 0.9737
              precision    recall  f1-score   support

           0       0.98      0.95      0.96        43
           1       0.97      0.99      0.98        71

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```

In [64]:
```python
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# 1. Confusion matrix calculate karna
# y_test = Asli jawab, y_pred = Model ke guesses
cm = confusion_matrix(y_test, y_pred)

# 2. Visual representation create karna
# display_labels mein hum 'Malignant' aur 'Benign' dikhayenge
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=data.target_names

# 3. Plot draw karna
disp.plot(cmap=plt.cm.Blues) # Blues color theme use ki hai
plt.title("Confusion Matrix: Model Performance")
plt.show()
```



Naive Bayes Algorithm

In [ ]:
```
#Is code ka main maqsad hai ek Machine Learning model banana
#jo phoolon (Iris flowers) ki dimensions dekh kar
#unka sahi naam (Species) predict kar sake.

#Hum Gaussian Naive Bayes algorithm ka use kar
#rahe hain, jo probability (sambhavna) par kaam karta hai.
```

1. ImportING libraries

In [87]:
```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

#sklearn: Yeh ML ki main library hai. Isse humne dataset,
#data split karne ka tool, aur Naive Bayes algorithm liya hai.

#pandas: Data ko table (DataFrame) format mein dekhne ke liye.

#seaborn/matplotlib: Result ko graph (Confusion Matrix) ke roop mein dikhane ke liy
```

2 Data SET lOAD

In [71]:
```
iris = datasets.load_iris()
```

In [73]:
```
iris
```

Out[73]:

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
                [4.9, 3. , 1.4, 0.2],
                [4.7, 3.2, 1.3, 0.2],
                [4.6, 3.1, 1.5, 0.2],
                [5. , 3.6, 1.4, 0.2],
                [5.4, 3.9, 1.7, 0.4],
                [4.6, 3.4, 1.4, 0.3],
                [5. , 3.4, 1.5, 0.2],
                [4.4, 2.9, 1.4, 0.2],
                [4.9, 3.1, 1.5, 0.1],
                [5.4, 3.7, 1.5, 0.2],
                [4.8, 3.4, 1.6, 0.2],
                [4.8, 3. , 1.4, 0.1],
                [4.3, 3. , 1.1, 0.1],
                [5.8, 4. , 1.2, 0.2],
                [5.7, 4.4, 1.5, 0.4],
                [5.4, 3.9, 1.3, 0.4],
                [5.1, 3.5, 1.4, 0.3],
                [5.7, 3.8, 1.7, 0.3],
                [5.1, 3.8, 1.5, 0.3],
                [5.4, 3.4, 1.7, 0.2],
                [5.1, 3.7, 1.5, 0.4],
                [4.6, 3.6, 1. , 0.2],
                [5.1, 3.3, 1.7, 0.5],
                [4.8, 3.4, 1.9, 0.2],
                [5. , 3. , 1.6, 0.2],
                [5. , 3.4, 1.6, 0.4],
                [5.2, 3.5, 1.5, 0.2],
                [5.2, 3.4, 1.4, 0.2],
                [4.7, 3.2, 1.6, 0.2],
                [4.8, 3.1, 1.6, 0.2],
                [5.4, 3.4, 1.5, 0.4],
                [5.2, 4.1, 1.5, 0.1],
                [5.5, 4.2, 1.4, 0.2],
                [4.9, 3.1, 1.5, 0.2],
                [5. , 3.2, 1.2, 0.2],
                [5.5, 3.5, 1.3, 0.2],
                [4.9, 3.6, 1.4, 0.1],
                [4.4, 3. , 1.3, 0.2],
                [5.1, 3.4, 1.5, 0.2],
                [5. , 3.5, 1.3, 0.3],
                [4.5, 2.3, 1.3, 0.3],
                [4.4, 3.2, 1.3, 0.2],
                [5. , 3.5, 1.6, 0.6],
                [5.1, 3.8, 1.9, 0.4],
                [4.8, 3. , 1.4, 0.3],
                [5.1, 3.8, 1.6, 0.2],
                [4.6, 3.2, 1.4, 0.2],
                [5.3, 3.7, 1.5, 0.2],
                [5. , 3.3, 1.4, 0.2],
                [7. , 3.2, 4.7, 1.4],
                [6.4, 3.2, 4.5, 1.5],
                [6.9, 3.1, 4.9, 1.5],
                [5.5, 2.3, 4. , 1.3],
                [6.5, 2.8, 4.6, 1.5],
                [5.7, 2.8, 4.5, 1.3],
```

```
[6.3, 3.3, 4.7, 1.6],
[4.9, 2.4, 3.3, 1. ],
[6.6, 2.9, 4.6, 1.3],
[5.2, 2.7, 3.9, 1.4],
[5. , 2. , 3.5, 1. ],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1. ],
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2. ],
[6.4, 2.7, 5.3, 1.9],
```

```
        [6.8, 3. , 5.5, 2.1],
        [5.7, 2.5, 5. , 2. ],
        [5.8, 2.8, 5.1, 2.4],
        [6.4, 3.2, 5.3, 2.3],
        [6.5, 3. , 5.5, 1.8],
        [7.7, 3.8, 6.7, 2.2],
        [7.7, 2.6, 6.9, 2.3],
        [6. , 2.2, 5. , 1.5],
        [6.9, 3.2, 5.7, 2.3],
        [5.6, 2.8, 4.9, 2. ],
        [7.7, 2.8, 6.7, 2. ],
        [6.3, 2.7, 4.9, 1.8],
        [6.7, 3.3, 5.7, 2.1],
        [7.2, 3.2, 6. , 1.8],
        [6.2, 2.8, 4.8, 1.8],
        [6.1, 3. , 4.9, 1.8],
        [6.4, 2.8, 5.6, 2.1],
        [7.2, 3. , 5.8, 1.6],
        [7.4, 2.8, 6.1, 1.9],
        [7.9, 3.8, 6.4, 2. ],
        [6.4, 2.8, 5.6, 2.2],
        [6.3, 2.8, 5.1, 1.5],
        [6.1, 2.6, 5.6, 1.4],
        [7.7, 3. , 6.1, 2.3],
        [6.3, 3.4, 5.6, 2.4],
        [6.4, 3.1, 5.5, 1.8],
        [6. , 3. , 4.8, 1.8],
        [6.9, 3.1, 5.4, 2.1],
        [6.7, 3.1, 5.6, 2.4],
        [6.9, 3.1, 5.1, 2.3],
        [5.8, 2.7, 5.1, 1.9],
        [6.8, 3.2, 5.9, 2.3],
        [6.7, 3.3, 5.7, 2.5],
        [6.7, 3. , 5.2, 2.3],
        [6.3, 2.5, 5. , 1.9],
        [6.5, 3. , 5.2, 2. ],
        [6.2, 3.4, 5.4, 2.3],
        [5.9, 3. , 5.1, 1.8]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),
 'frame': None,
 'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
 'DESCR': '.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Dat
a Set Characteristics:**\n\n:Number of Instances: 150 (50 in each of three classe
s)\n:Number of Attributes: 4 numeric, predictive attributes and the class\n:Attrib
ute Information:\n    - sepal length in cm\n    - sepal width in cm\n    - petal l
ength in cm\n    - petal width in cm\n    - class:\n            - Iris-Setosa\n
- Iris-Versicolour\n            - Iris-Virginica\n\n:Summary Statistics:\n\n======
======== ==== ==== ======= ===== ====================\n                Min  Max
Mean    SD   Class Correlation\n============== ==== ==== ======= ===== ===========
```

```
=========\nsepal length:   4.3  7.9   5.84   0.83    0.7826\nsepal width:    2.0
4.4   3.05   0.43   -0.4194\npetal length:   1.0  6.9   3.76   1.76    0.9490  (hi
gh!)\npetal width:    0.1  2.5   1.20   0.76    0.9565  (high!)\n============== ==
== ==== ======= ===== ====================\n\n:Missing Attribute Values: None\n:Cl
ass Distribution: 33.3% for each of 3 classes.\n:Creator: R.A. Fisher\n:Donor: Mic
hael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n:Date: July, 1988\n\nThe famous Iris
database, first used by Sir R.A. Fisher. The dataset is taken\nfrom Fisher\'s pape
r. Note that it\'s the same as in R, but not as in the UCI\nMachine Learning Repos
itory, which has two wrong data points.\n\nThis is perhaps the best known database
to be found in the\npattern recognition literature.  Fisher\'s paper is a classic
in the field and\nis referenced frequently to this day.  (See Duda & Hart, for exa
mple.)  The\ndata set contains 3 classes of 50 instances each, where each class re
fers to a\ntype of iris plant.  One class is linearly separable from the other 2;
the\nlatter are NOT linearly separable from each other.\n\n.. dropdown:: Reference
s\n\n  - Fisher, R.A. "The use of multiple measurements in taxonomic problems"\n
Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to\n    Mathem
atical Statistics" (John Wiley, NY, 1950).\n  - Duda, R.O., & Hart, P.E. (1973) Pa
ttern Classification and Scene Analysis.\n    (Q327.D83) John Wiley & Sons.  ISBN
0-471-22361-1.  See page 218.\n  - Dasarathy, B.V. (1980) "Nosing Around the Neigh
borhood: A New System\n    Structure and Classification Rule for Recognition in Pa
rtially Exposed\n    Environments".  IEEE Transactions on Pattern Analysis and Mac
hine\n    Intelligence, Vol. PAMI-2, No. 1, 67-71.\n  - Gates, G.W. (1972) "The Re
duced Nearest Neighbor Rule".  IEEE Transactions\n    on Information Theory, May 1
972, 431-433.\n  - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOC
LASS II\n    conceptual clustering system finds 3 classes in the data.\n  - Many,
many more ...\n',
 'feature_names': ['sepal length (cm)',
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 'filename': 'iris.csv',
 'data_module': 'sklearn.datasets.data'}
```

In [75]:
```
X = iris.data # Features: Length aur width
y = iris.target # Labels: Phoolon ke naam (0, 1, 2)

#Humne iris dataset load kiya. X mein input features
#hain (jaise petal length), aur y mein answer hai (kaunsa phool hai)
```

In [77]:
```
X
```

Out[77]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
```

```
[6.3, 3.3, 4.7, 1.6],
[4.9, 2.4, 3.3, 1. ],
[6.6, 2.9, 4.6, 1.3],
[5.2, 2.7, 3.9, 1.4],
[5. , 2. , 3.5, 1. ],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1. ],
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2. ],
[6.4, 2.7, 5.3, 1.9],
```

```
       [6.8, 3. , 5.5, 2.1],
       [5.7, 2.5, 5. , 2. ],
       [5.8, 2.8, 5.1, 2.4],
       [6.4, 3.2, 5.3, 2.3],
       [6.5, 3. , 5.5, 1.8],
       [7.7, 3.8, 6.7, 2.2],
       [7.7, 2.6, 6.9, 2.3],
       [6. , 2.2, 5. , 1.5],
       [6.9, 3.2, 5.7, 2.3],
       [5.6, 2.8, 4.9, 2. ],
       [7.7, 2.8, 6.7, 2. ],
       [6.3, 2.7, 4.9, 1.8],
       [6.7, 3.3, 5.7, 2.1],
       [7.2, 3.2, 6. , 1.8],
       [6.2, 2.8, 4.8, 1.8],
       [6.1, 3. , 4.9, 1.8],
       [6.4, 2.8, 5.6, 2.1],
       [7.2, 3. , 5.8, 1.6],
       [7.4, 2.8, 6.1, 1.9],
       [7.9, 3.8, 6.4, 2. ],
       [6.4, 2.8, 5.6, 2.2],
       [6.3, 2.8, 5.1, 1.5],
       [6.1, 2.6, 5.6, 1.4],
       [7.7, 3. , 6.1, 2.3],
       [6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]])
```

In [79]:
```
y
```

Out[79]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [81]:
```
# Convert to DataFrame just for a cleaner look at the data
df = pd.DataFrame(X, columns=iris.feature_names)
df['species'] = y
print("First 5 rows of the dataset:")
print(df.head())
```

```
First 5 rows of the dataset:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   species
0        0
1        0
2        0
3        0
4        0
```

3 Train test split

In [84]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

#Hum saara data model ko nahi sikhate. 80% data (Train) hum model ko
#padhane ke liye rakhte hain, aur 20% data (Test)
#hum model ka exam lene ke liye chhupa kar rakhte hain.
```

4. Model training

In [89]:
```
model = GaussianNB()
model.fit(X_train, y_train)

#GaussianNB(): Humne Naive Bayes ka "dimag" (object) banaya.

# .fit(): Yahan asli Training ho rahi hai.
# Model X_train aur y_train ke beech ka pattern samajh raha hai.
```

Out[89]:
```
GaussianNB
GaussianNB()
```

5 Prediction

In [92]:
```
y_pred = model.predict(X_test)
```

In [94]:
```
y_pred
```

Out[94]:
```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0])
```

In [96]:
```
y_test
```

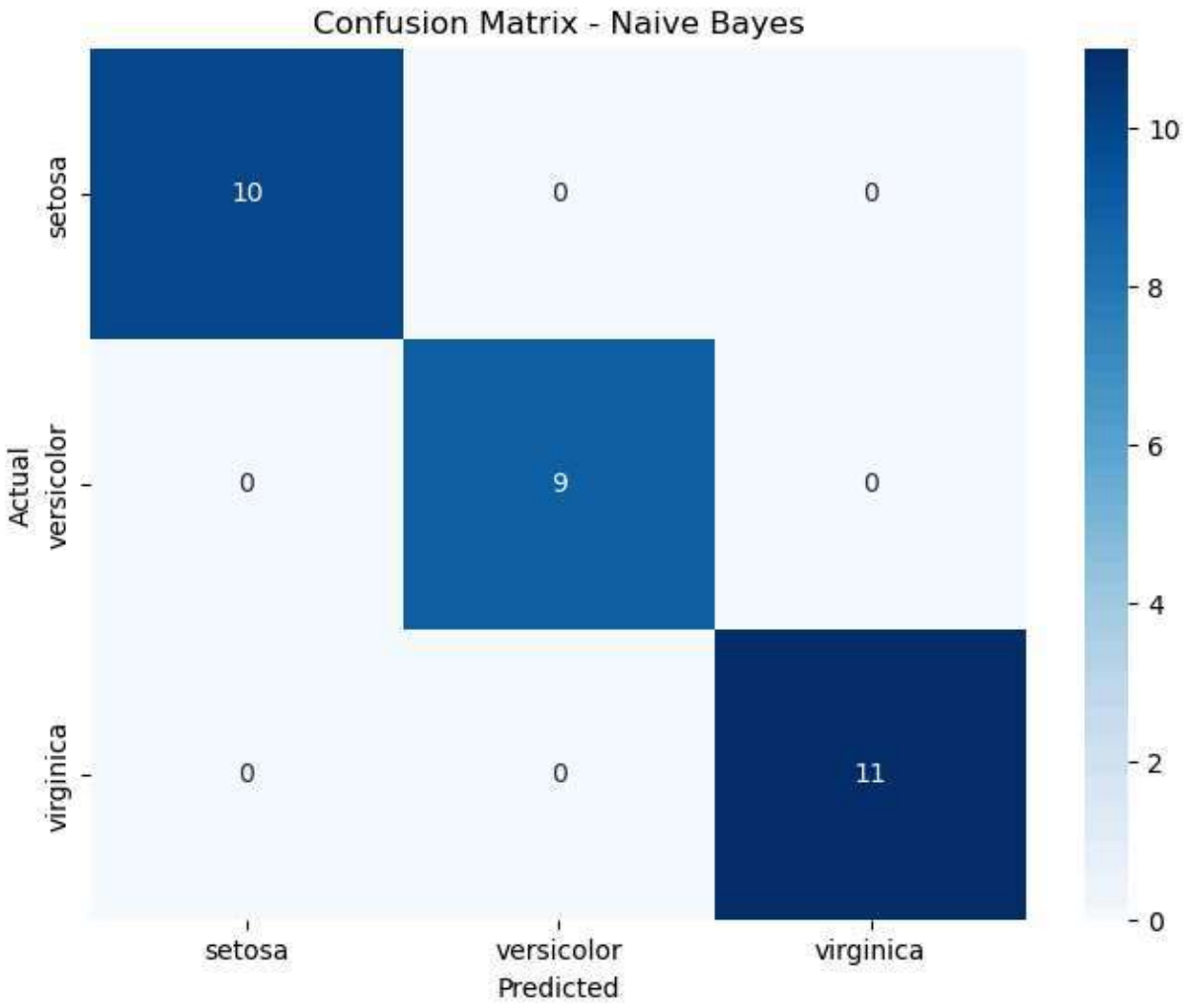
Out[96]:
```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0])
```

6 Accuracy

In [99]:
```
accuracy = accuracy_score(y_test, y_pred)

print(f"\nModel Accuracy: {accuracy * 100:.2f}%")
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

```
Model Accuracy: 100.00%

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

In [101…
```
# 6. Visualize the Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Naive Bayes')
plt.show()
```



KNN Algorithm

1 Importing libraries

In [117…
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
```

2 Load data set

In [120…
```
iris = load_iris()
X = iris.data  # Features
y = iris.target # Target labels
```

In [122…
```
X
```

Out[122…
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
```

```
[6.3, 3.3, 4.7, 1.6],
[4.9, 2.4, 3.3, 1. ],
[6.6, 2.9, 4.6, 1.3],
[5.2, 2.7, 3.9, 1.4],
[5. , 2. , 3.5, 1. ],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1. ],
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2. ],
[6.4, 2.7, 5.3, 1.9],
```

```
       [6.8, 3. , 5.5, 2.1],
       [5.7, 2.5, 5. , 2. ],
       [5.8, 2.8, 5.1, 2.4],
       [6.4, 3.2, 5.3, 2.3],
       [6.5, 3. , 5.5, 1.8],
       [7.7, 3.8, 6.7, 2.2],
       [7.7, 2.6, 6.9, 2.3],
       [6. , 2.2, 5. , 1.5],
       [6.9, 3.2, 5.7, 2.3],
       [5.6, 2.8, 4.9, 2. ],
       [7.7, 2.8, 6.7, 2. ],
       [6.3, 2.7, 4.9, 1.8],
       [6.7, 3.3, 5.7, 2.1],
       [7.2, 3.2, 6. , 1.8],
       [6.2, 2.8, 4.8, 1.8],
       [6.1, 3. , 4.9, 1.8],
       [6.4, 2.8, 5.6, 2.1],
       [7.2, 3. , 5.8, 1.6],
       [7.4, 2.8, 6.1, 1.9],
       [7.9, 3.8, 6.4, 2. ],
       [6.4, 2.8, 5.6, 2.2],
       [6.3, 2.8, 5.1, 1.5],
       [6.1, 2.6, 5.6, 1.4],
       [7.7, 3. , 6.1, 2.3],
       [6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]])
```

In [124…

```
y
```

Out[124…

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

3 Train test split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

4 Standard scaler

In [130…
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

5 model training

In [133…
```
# We'll use k=3 for this example
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
```

Out[133…
```
KNeighborsClassifier(n_neighbors=3)
```

6 Prediction

In [138…
```
y_pred = knn.predict(X_test)
```

In [140…
```
y_pred
```

Out[140…
```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0])
```

In [142…
```
y_test
```

Out[142…
```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0])
```

7 accuracy

In [145…
```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 100.00%
```

Decision Tree Algorithm

1 Importing Libraries

In [150…
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

2 Loading the Dataset

In [159…
```
iris = load_iris()
X = iris.data # Sepal/Petal dimensions
y = iris.target # Flower species (0, 1, 2)
```

In [161…
```
iris
```

Out[161… {'data': array([[5.1, 3.5, 1.4, 0.2],
[4.9, 3. , 1.4, 0.2],
[4.7, 3.2, 1.3, 0.2],
[4.6, 3.1, 1.5, 0.2],
[5. , 3.6, 1.4, 0.2],
[5.4, 3.9, 1.7, 0.4],
[4.6, 3.4, 1.4, 0.3],
[5. , 3.4, 1.5, 0.2],
[4.4, 2.9, 1.4, 0.2],
[4.9, 3.1, 1.5, 0.1],
[5.4, 3.7, 1.5, 0.2],
[4.8, 3.4, 1.6, 0.2],
[4.8, 3. , 1.4, 0.1],
[4.3, 3. , 1.1, 0.1],
[5.8, 4. , 1.2, 0.2],
[5.7, 4.4, 1.5, 0.4],
[5.4, 3.9, 1.3, 0.4],
[5.1, 3.5, 1.4, 0.3],
[5.7, 3.8, 1.7, 0.3],
[5.1, 3.8, 1.5, 0.3],
[5.4, 3.4, 1.7, 0.2],
[5.1, 3.7, 1.5, 0.4],
[4.6, 3.6, 1. , 0.2],
[5.1, 3.3, 1.7, 0.5],
[4.8, 3.4, 1.9, 0.2],
[5. , 3. , 1.6, 0.2],
[5. , 3.4, 1.6, 0.4],
[5.2, 3.5, 1.5, 0.2],
[5.2, 3.4, 1.4, 0.2],
[4.7, 3.2, 1.6, 0.2],
[4.8, 3.1, 1.6, 0.2],
[5.4, 3.4, 1.5, 0.4],
[5.2, 4.1, 1.5, 0.1],
[5.5, 4.2, 1.4, 0.2],
[4.9, 3.1, 1.5, 0.2],
[5. , 3.2, 1.2, 0.2],
[5.5, 3.5, 1.3, 0.2],
[4.9, 3.6, 1.4, 0.1],
[4.4, 3. , 1.3, 0.2],
[5.1, 3.4, 1.5, 0.2],
[5. , 3.5, 1.3, 0.3],
[4.5, 2.3, 1.3, 0.3],
[4.4, 3.2, 1.3, 0.2],
[5. , 3.5, 1.6, 0.6],
[5.1, 3.8, 1.9, 0.4],
[4.8, 3. , 1.4, 0.3],
[5.1, 3.8, 1.6, 0.2],
[4.6, 3.2, 1.4, 0.2],
[5.3, 3.7, 1.5, 0.2],
[5. , 3.3, 1.4, 0.2],
[7. , 3.2, 4.7, 1.4],
[6.4, 3.2, 4.5, 1.5],
[6.9, 3.1, 4.9, 1.5],
[5.5, 2.3, 4. , 1.3],
[6.5, 2.8, 4.6, 1.5],
[5.7, 2.8, 4.5, 1.3],

```
[6.3, 3.3, 4.7, 1.6],
[4.9, 2.4, 3.3, 1. ],
[6.6, 2.9, 4.6, 1.3],
[5.2, 2.7, 3.9, 1.4],
[5. , 2. , 3.5, 1. ],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1. ],
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2. ],
[6.4, 2.7, 5.3, 1.9],
```

```
        [6.8, 3. , 5.5, 2.1],
        [5.7, 2.5, 5. , 2. ],
        [5.8, 2.8, 5.1, 2.4],
        [6.4, 3.2, 5.3, 2.3],
        [6.5, 3. , 5.5, 1.8],
        [7.7, 3.8, 6.7, 2.2],
        [7.7, 2.6, 6.9, 2.3],
        [6. , 2.2, 5. , 1.5],
        [6.9, 3.2, 5.7, 2.3],
        [5.6, 2.8, 4.9, 2. ],
        [7.7, 2.8, 6.7, 2. ],
        [6.3, 2.7, 4.9, 1.8],
        [6.7, 3.3, 5.7, 2.1],
        [7.2, 3.2, 6. , 1.8],
        [6.2, 2.8, 4.8, 1.8],
        [6.1, 3. , 4.9, 1.8],
        [6.4, 2.8, 5.6, 2.1],
        [7.2, 3. , 5.8, 1.6],
        [7.4, 2.8, 6.1, 1.9],
        [7.9, 3.8, 6.4, 2. ],
        [6.4, 2.8, 5.6, 2.2],
        [6.3, 2.8, 5.1, 1.5],
        [6.1, 2.6, 5.6, 1.4],
        [7.7, 3. , 6.1, 2.3],
        [6.3, 3.4, 5.6, 2.4],
        [6.4, 3.1, 5.5, 1.8],
        [6. , 3. , 4.8, 1.8],
        [6.9, 3.1, 5.4, 2.1],
        [6.7, 3.1, 5.6, 2.4],
        [6.9, 3.1, 5.1, 2.3],
        [5.8, 2.7, 5.1, 1.9],
        [6.8, 3.2, 5.9, 2.3],
        [6.7, 3.3, 5.7, 2.5],
        [6.7, 3. , 5.2, 2.3],
        [6.3, 2.5, 5. , 1.9],
        [6.5, 3. , 5.2, 2. ],
        [6.2, 3.4, 5.4, 2.3],
        [5.9, 3. , 5.1, 1.8]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),
 'frame': None,
 'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
 'DESCR': '.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Dat
a Set Characteristics:**\n\n:Number of Instances: 150 (50 in each of three classe
s)\n:Number of Attributes: 4 numeric, predictive attributes and the class\n:Attrib
ute Information:\n    - sepal length in cm\n    - sepal width in cm\n    - petal l
ength in cm\n    - petal width in cm\n    - class:\n            - Iris-Setosa\n
- Iris-Versicolour\n            - Iris-Virginica\n\n:Summary Statistics:\n\n======
======== ==== ==== ======= ===== ====================\n                Min  Max
Mean    SD   Class Correlation\n============== ==== ==== ======= ===== ===========
```

```
=========\nsepal length:   4.3  7.9   5.84   0.83    0.7826\nsepal width:    2.0
4.4   3.05   0.43   -0.4194\npetal length:   1.0  6.9   3.76   1.76    0.9490  (hi
gh!)\npetal width:    0.1  2.5   1.20   0.76    0.9565  (high!)\n============== ==
== ==== ======= ===== ====================\n\n:Missing Attribute Values: None\n:Cl
ass Distribution: 33.3% for each of 3 classes.\n:Creator: R.A. Fisher\n:Donor: Mic
hael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n:Date: July, 1988\n\nThe famous Iris
database, first used by Sir R.A. Fisher. The dataset is taken\nfrom Fisher\'s pape
r. Note that it\'s the same as in R, but not as in the UCI\nMachine Learning Repos
itory, which has two wrong data points.\n\nThis is perhaps the best known database
to be found in the\npattern recognition literature.  Fisher\'s paper is a classic
in the field and\nis referenced frequently to this day.  (See Duda & Hart, for exa
mple.)  The\ndata set contains 3 classes of 50 instances each, where each class re
fers to a\ntype of iris plant.  One class is linearly separable from the other 2;
the\nlatter are NOT linearly separable from each other.\n\n.. dropdown:: Reference
s\n\n  - Fisher, R.A. "The use of multiple measurements in taxonomic problems"\n
Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to\n    Mathem
atical Statistics" (John Wiley, NY, 1950).\n  - Duda, R.O., & Hart, P.E. (1973) Pa
ttern Classification and Scene Analysis.\n    (Q327.D83) John Wiley & Sons.  ISBN
0-471-22361-1.  See page 218.\n  - Dasarathy, B.V. (1980) "Nosing Around the Neigh
borhood: A New System\n    Structure and Classification Rule for Recognition in Pa
rtially Exposed\n    Environments".  IEEE Transactions on Pattern Analysis and Mac
hine\n    Intelligence, Vol. PAMI-2, No. 1, 67-71.\n  - Gates, G.W. (1972) "The Re
duced Nearest Neighbor Rule".  IEEE Transactions\n    on Information Theory, May 1
972, 431-433.\n  - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOC
LASS II\n    conceptual clustering system finds 3 classes in the data.\n  - Many,
many more ...\n',
 'feature_names': ['sepal length (cm)',
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 'filename': 'iris.csv',
 'data_module': 'sklearn.datasets.data'}
```

In [163…

```
X
```

Out[163…

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
```

```
[6.3, 3.3, 4.7, 1.6],
[4.9, 2.4, 3.3, 1. ],
[6.6, 2.9, 4.6, 1.3],
[5.2, 2.7, 3.9, 1.4],
[5. , 2. , 3.5, 1. ],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1. ],
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2. ],
[6.4, 2.7, 5.3, 1.9],
```

```
       [6.8, 3. , 5.5, 2.1],
       [5.7, 2.5, 5. , 2. ],
       [5.8, 2.8, 5.1, 2.4],
       [6.4, 3.2, 5.3, 2.3],
       [6.5, 3. , 5.5, 1.8],
       [7.7, 3.8, 6.7, 2.2],
       [7.7, 2.6, 6.9, 2.3],
       [6. , 2.2, 5. , 1.5],
       [6.9, 3.2, 5.7, 2.3],
       [5.6, 2.8, 4.9, 2. ],
       [7.7, 2.8, 6.7, 2. ],
       [6.3, 2.7, 4.9, 1.8],
       [6.7, 3.3, 5.7, 2.1],
       [7.2, 3.2, 6. , 1.8],
       [6.2, 2.8, 4.8, 1.8],
       [6.1, 3. , 4.9, 1.8],
       [6.4, 2.8, 5.6, 2.1],
       [7.2, 3. , 5.8, 1.6],
       [7.4, 2.8, 6.1, 1.9],
       [7.9, 3.8, 6.4, 2. ],
       [6.4, 2.8, 5.6, 2.2],
       [6.3, 2.8, 5.1, 1.5],
       [6.1, 2.6, 5.6, 1.4],
       [7.7, 3. , 6.1, 2.3],
       [6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]])
```

In [165…

```
y
```

Out[165…

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

3 Training & test data split

In [168…

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

4 Standard Scaler

In [171…
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [173…
```
X_train
```

Out[173…
```
array([[-1.47393679,  1.20365799, -1.56253475, -1.31260282],
       [-0.13307079,  2.99237573, -1.27600637, -1.04563275],
       [ 1.08589829,  0.08570939,  0.38585821,  0.28921757],
       [-1.23014297,  0.75647855, -1.2187007 , -1.31260282],
       [-1.7177306 ,  0.30929911, -1.39061772, -1.31260282],
       [ 0.59831066, -1.25582892,  0.72969227,  0.95664273],
       [ 0.72020757,  0.30929911,  0.44316389,  0.4227026 ],
       [-0.74255534,  0.98006827, -1.27600637, -1.31260282],
       [-0.98634915,  1.20365799, -1.33331205, -1.31260282],
       [-0.74255534,  2.32160658, -1.27600637, -1.44608785],
       [-0.01117388, -0.80864948,  0.78699794,  0.95664273],
       [ 0.23261993,  0.75647855,  0.44316389,  0.55618763],
       [ 1.08589829,  0.08570939,  0.55777524,  0.4227026 ],
       [-0.49876152,  1.87442714, -1.39061772, -1.04563275],
       [-0.49876152,  1.4272477 , -1.27600637, -1.31260282],
       [-0.37686461, -1.47941864, -0.01528151, -0.24472256],
       [ 0.59831066, -0.58505976,  0.78699794,  0.4227026 ],
       [ 0.72020757,  0.08570939,  1.01622064,  0.8231577 ],
       [ 0.96400139, -0.13788033,  0.38585821,  0.28921757],
       [ 1.69538284,  1.20365799,  1.3600547 ,  1.75755292],
       [-0.13307079, -0.36147005,  0.27124686,  0.15573254],
       [ 2.18297047, -0.13788033,  1.64658307,  1.22361279],
       [-0.2549677 , -0.13788033,  0.44316389,  0.4227026 ],
       [-0.86445224,  0.98006827, -1.33331205, -1.31260282],
       [ 2.30486738, -0.58505976,  1.70388875,  1.09012776],
       [-0.01117388, -0.80864948,  0.21394119, -0.24472256],
       [-0.74255534,  0.75647855, -1.33331205, -1.31260282],
       [-0.98634915,  0.98006827, -1.39061772, -1.17911778],
       [-0.86445224,  1.65083742, -1.04678367, -1.04563275],
       [-0.98634915, -2.37377751, -0.12989286, -0.24472256],
       [ 0.59831066, -0.80864948,  0.67238659,  0.8231577 ],
       [-1.23014297,  0.75647855, -1.04678367, -1.31260282],
       [-0.98634915, -0.13788033, -1.2187007 , -1.31260282],
       [-0.86445224,  0.53288883, -1.16139502, -0.91214772],
       [-0.2549677 , -0.80864948,  0.27124686,  0.15573254],
       [-0.86445224,  0.75647855, -1.27600637, -1.31260282],
       [-0.13307079, -0.13788033,  0.27124686,  0.02224751],
       [ 2.30486738,  1.65083742,  1.70388875,  1.35709783],
       [-1.47393679,  0.30929911, -1.33331205, -1.31260282],
       [ 0.47641375, -0.36147005,  0.32855254,  0.15573254],
       [-0.13307079, -1.25582892,  0.72969227,  1.09012776],
       [-0.37686461,  2.5451963 , -1.33331205, -1.31260282],
       [ 0.23261993, -0.13788033,  0.61508092,  0.8231577 ],
       [-0.01117388, -0.80864948,  0.78699794,  0.95664273],
       [ 0.23261993, -1.92659808,  0.15663551, -0.24472256],
       [-0.49876152, -0.13788033,  0.44316389,  0.4227026 ],
       [ 0.47641375,  0.75647855,  0.95891497,  1.49058286],
       [-0.37686461, -1.70300836,  0.15663551,  0.15573254],
       [-0.49876152,  1.87442714, -1.16139502, -1.04563275],
       [-0.98634915, -1.70300836, -0.24450422, -0.24472256],
       [ 0.72020757, -0.80864948,  0.90160929,  0.95664273],
       [-0.98634915,  0.53288883, -1.33331205, -1.31260282],
       [-0.98634915,  0.30929911, -1.4479234 , -1.31260282],
       [-0.37686461, -1.47941864,  0.04202416, -0.11123753],
       [ 1.08589829, -0.13788033,  0.72969227,  0.68967267],
       [-1.10824606,  0.08570939, -1.27600637, -1.31260282],
```

```
[-0.01117388, -0.58505976,  0.78699794,  1.62406789],
[-0.98634915,  0.75647855, -1.27600637, -1.31260282],
[-0.98634915,  0.98006827, -1.2187007 , -0.77866269],
[ 0.11072303,  0.30929911,  0.61508092,  0.8231577 ],
[-0.86445224, -1.25582892, -0.41642124, -0.11123753],
[ 1.32969211,  0.30929911,  1.130832  ,  1.49058286],
[ 0.23261993, -0.80864948,  0.78699794,  0.55618763],
[ 0.35451684, -1.0322392 ,  1.07352632,  0.28921757],
[ 2.30486738, -0.13788033,  1.3600547 ,  1.49058286],
[-0.37686461, -1.25582892,  0.15663551,  0.15573254],
[-1.7177306 , -0.36147005, -1.33331205, -1.31260282],
[-1.83962751, -0.13788033, -1.50522907, -1.44608785],
[ 0.23261993, -1.92659808,  0.72969227,  0.4227026 ],
[ 1.69538284,  0.30929911,  1.30274902,  0.8231577 ],
[-1.47393679,  0.08570939, -1.27600637, -1.31260282],
[-0.86445224,  0.98006827, -1.33331205, -1.17911778],
[-1.7177306 , -0.13788033, -1.39061772, -1.31260282],
[ 0.59831066, -1.25582892,  0.67238659,  0.4227026 ],
[ 0.59831066,  0.75647855,  1.07352632,  1.62406789],
[-1.47393679,  0.75647855, -1.33331205, -1.17911778],
[ 1.2077952 , -0.13788033,  1.01622064,  1.22361279],
[ 0.59831066,  0.53288883,  1.30274902,  1.75755292],
[-1.35203988,  0.30929911, -1.39061772, -1.31260282],
[ 0.35451684, -0.36147005,  0.55777524,  0.28921757],
[ 0.84210448, -0.58505976,  0.50046957,  0.4227026 ],
[ 0.47641375, -0.58505976,  0.61508092,  0.8231577 ],
[ 1.45158902,  0.30929911,  0.55777524,  0.28921757],
[ 0.72020757,  0.30929911,  0.90160929,  1.49058286],
[-0.86445224,  1.65083742, -1.2187007 , -1.31260282],
[ 1.32969211,  0.08570939,  0.95891497,  1.22361279],
[ 0.11072303, -0.13788033,  0.27124686,  0.4227026 ],
[ 0.84210448, -0.13788033,  0.84430362,  1.09012776],
[-0.13307079, -1.0322392 , -0.12989286, -0.24472256],
[-0.74255534, -0.80864948,  0.09932984,  0.28921757],
[ 0.35451684, -0.13788033,  0.50046957,  0.28921757],
[-1.5958337 , -1.70300836, -1.39061772, -1.17911778],
[ 0.96400139, -0.36147005,  0.50046957,  0.15573254],
[-0.37686461, -1.0322392 ,  0.38585821,  0.02224751],
[-0.62065843,  1.4272477 , -1.27600637, -1.31260282],
[-0.2549677 , -0.13788033,  0.21394119,  0.15573254],
[ 1.81727975, -0.36147005,  1.47466605,  0.8231577 ],
[ 1.08589829,  0.53288883,  1.130832  ,  1.22361279],
[-0.86445224,  1.4272477 , -1.27600637, -1.04563275],
[-1.10824606, -1.47941864, -0.24450422, -0.24472256],
[ 1.08589829,  0.53288883,  1.130832  ,  1.75755292],
[ 1.69538284, -0.13788033,  1.18813767,  0.55618763],
[-1.10824606,  1.20365799, -1.33331205, -1.44608785],
[ 1.08589829,  0.08570939,  1.07352632,  1.62406789],
[-1.10824606, -0.13788033, -1.33331205, -1.31260282],
[ 1.32969211,  0.08570939,  0.67238659,  0.4227026 ],
[ 1.93917666, -0.58505976,  1.3600547 ,  0.95664273],
[ 0.59831066, -0.36147005,  1.07352632,  0.8231577 ],
[-0.13307079, -0.58505976,  0.21394119,  0.15573254],
[ 0.84210448, -0.13788033,  1.01622064,  0.8231577 ],
[ 0.59831066, -1.70300836,  0.38585821,  0.15573254],
[ 0.72020757, -0.36147005,  0.32855254,  0.15573254],
```

```
       [-0.2549677 , -0.58505976,  0.67238659,  1.09012776],
       [ 0.11072303, -0.13788033,  0.78699794,  0.8231577 ],
       [-0.49876152,  0.75647855, -1.16139502, -1.31260282],
       [ 0.35451684, -0.58505976,  0.15663551,  0.15573254],
       [-1.10824606, -1.25582892,  0.44316389,  0.68967267],
       [-0.01117388,  2.09801686, -1.4479234 , -1.31260282],
       [-0.01117388, -1.0322392 ,  0.15663551,  0.02224751],
       [ 1.57348593, -0.13788033,  1.24544335,  1.22361279]])
```

In [175…
```
X_test
```

Out[175…
```
array([[ 0.35451684, -0.58505976,  0.55777524,  0.02224751],
       [-0.13307079,  1.65083742, -1.16139502, -1.17911778],
       [ 2.30486738, -1.0322392 ,  1.8185001 ,  1.49058286],
       [ 0.23261993, -0.36147005,  0.44316389,  0.4227026 ],
       [ 1.2077952 , -0.58505976,  0.61508092,  0.28921757],
       [-0.49876152,  0.75647855, -1.27600637, -1.04563275],
       [-0.2549677 , -0.36147005, -0.07258719,  0.15573254],
       [ 1.32969211,  0.08570939,  0.78699794,  1.49058286],
       [ 0.47641375, -1.92659808,  0.44316389,  0.4227026 ],
       [-0.01117388, -0.80864948,  0.09932984,  0.02224751],
       [ 0.84210448,  0.30929911,  0.78699794,  1.09012776],
       [-1.23014297, -0.13788033, -1.33331205, -1.44608785],
       [-0.37686461,  0.98006827, -1.39061772, -1.31260282],
       [-1.10824606,  0.08570939, -1.27600637, -1.44608785],
       [-0.86445224,  1.65083742, -1.27600637, -1.17911778],
       [ 0.59831066,  0.53288883,  0.55777524,  0.55618763],
       [ 0.84210448, -0.13788033,  1.18813767,  1.35709783],
       [-0.2549677 , -1.25582892,  0.09932984, -0.11123753],
       [-0.13307079, -0.58505976,  0.44316389,  0.15573254],
       [ 0.72020757, -0.58505976,  1.07352632,  1.35709783],
       [-1.35203988,  0.30929911, -1.2187007 , -1.31260282],
       [ 0.35451684, -0.13788033,  0.67238659,  0.8231577 ],
       [-0.98634915,  0.75647855, -1.2187007 , -1.04563275],
       [ 0.72020757, -0.58505976,  1.07352632,  1.22361279],
       [ 2.5486612 ,  1.65083742,  1.53197172,  1.09012776],
       [ 1.08589829, -0.13788033,  0.84430362,  1.49058286],
       [ 1.08589829, -1.25582892,  1.18813767,  0.8231577 ],
       [ 1.2077952 ,  0.30929911,  1.24544335,  1.49058286],
       [-1.23014297, -0.13788033, -1.33331205, -1.17911778],
       [-1.23014297,  0.08570939, -1.2187007 , -1.31260282]])
```

5 Model training

In [178…
```
# criterion='gini' (Default) ya 'entropy' use kar sakte hain
# max_depth=3 rakha hai taaki overfitting na ho (Pruning)
clf = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=42)
clf.fit(X_train, y_train)
```

Out[178…
```
DecisionTreeClassifier(max_depth=3, random_state=42)
```

6 Prediction

In [181…
```
y_pred = clf.predict(X_test)
```

In [183…
```
y_pred
```

Out[183…
```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0])
```

In [185…
```
y_test
```

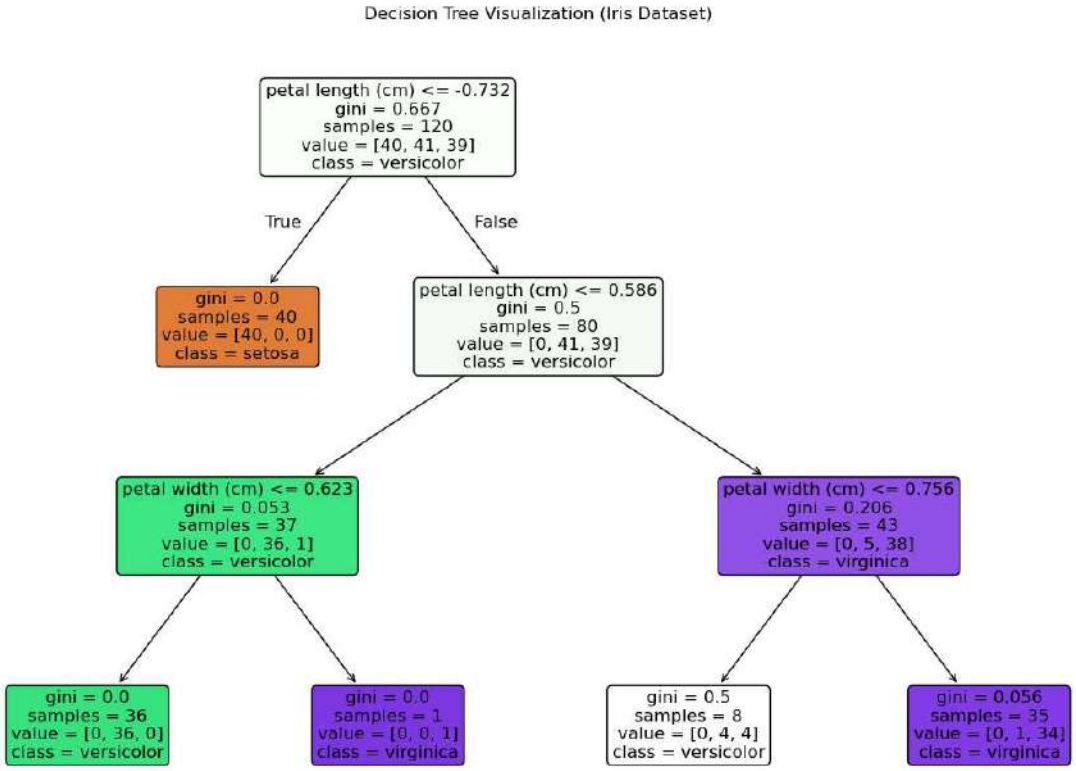
Out[185…
```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0])
```

7 Accuracy

print(f"Accuracy Score: {accuracy_score(y_test, y_pred) * 100:.2f}%") print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred)) print("\nDetailed Report:\n", classification_report(y_test, y_pred))

Decision tree visualisation

In [191…
```
import matplotlib.pyplot as plt

plt.figure(figsize=(15, 10))  # Diagram ka size bada karne ke liye

plot_tree(clf,
          feature_names=iris.feature_names,  # Features ke naam (e.g. Petal Length)
          class_names=iris.target_names,    # Classes ke naam (e.g. Setosa)
          filled=True,                      # Nodes mein color bharne ke liye
          rounded=True,                     # Boxes ke corners round karne ke liye
          fontsize=12)                      # Text ka size

plt.title("Decision Tree Visualization (Iris Dataset)")
plt.show()
```



Random Forest Algorithm

1 Importing Libraries

In [54]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

2. Dataset load

In [56]:
```
# 1. Dataset Load karna (Breast Cancer dataset - Diagnosis ke liye)
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = data.target
```

In [57]:
```
X
```

Out[57]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	sy
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	
...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	

569 rows × 30 columns

In [60]:
```
y
```

Out[60]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1])
```

Training Test data split

In [64]:
```
# 2. Data ko Train aur Test mein split karna (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [66]:
```
# 3. Random Forest Classifier banana
# n_estimators: Kitne trees banane hain (Forest mein kitne ped honge)
# max_depth: Ek tree kitna gehra ja sakta hai
# random_state: Har baar same result milne ke liye
rf_model = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=42)
#n_estimators=100
#Matlab: Forest mein kitne ped (Decision Trees) honge.

#Intuition: Yahan humne 100 trees banaye hain. Jab prediction ki baari aayegi, toh

#Tip: Zyadatar cases mein n_estimators jitna zyada hoga, model utna stable hoga, le
```

Model Training

In [69]:
```
# 4. Model Training
rf_model.fit(X_train, y_train)
```

Out[69]:
```
RandomForestClassifier
RandomForestClassifier(max_depth=5, random_state=42)
```

Prediction

In [71]:
```
y_pred = rf_model.predict(X_test)
```

In [74]:
```
y_pred
```

Out[74]:
```
array([1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 0])
```

In [76]:
```
y_test
```

Out[76]:
```
array([1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 0])
```

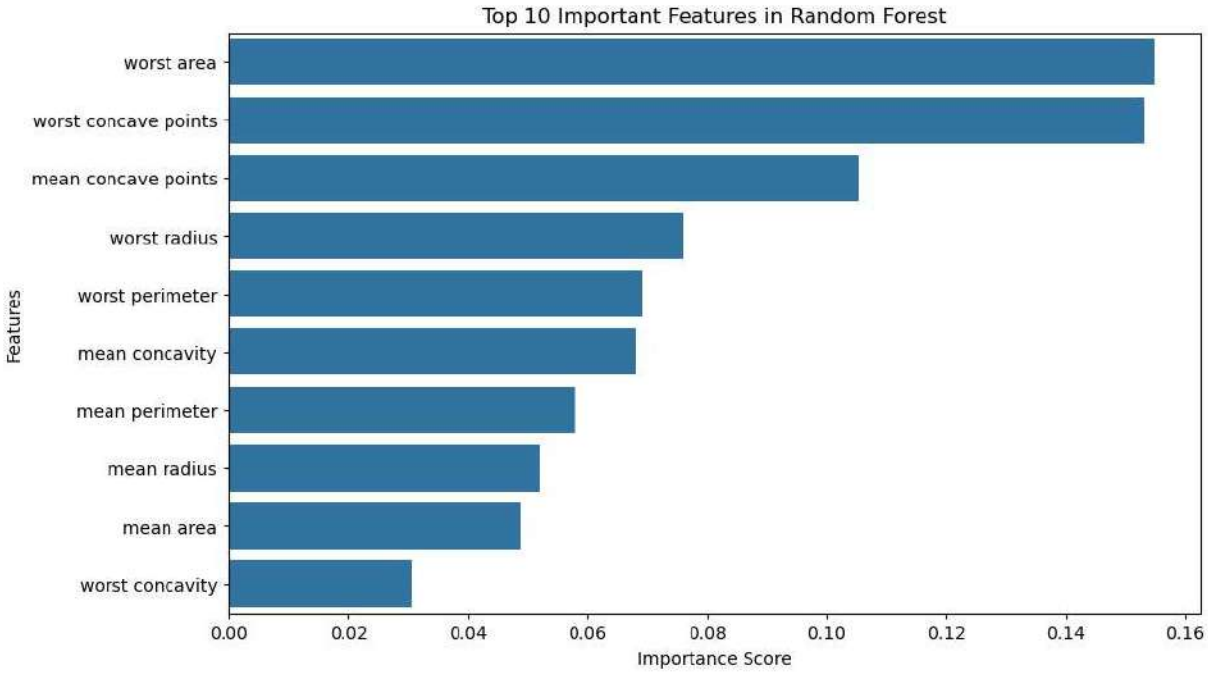
Accuracy

In [79]:
```
print(f"Model Accuracy: {accuracy_score(y_test, y_pred) * 100:.2f}%")
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Model Accuracy: 96.49%

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.93      0.95        43
           1       0.96      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114
```

In [81]:
```
# --- Visualization: Feature Importance ---
# Random Forest batata hai ki kaunsa feature (column) sabse important tha
importances = rf_model.feature_importances_
feature_imp = pd.Series(importances, index=data.feature_names).sort_values(ascendin

plt.figure(figsize=(10, 6))
sns.barplot(x=feature_imp[:10], y=feature_imp.index[:10]) # Top 10 features
```

```
plt.title("Top 10 Important Features in Random Forest")
plt.xlabel("Importance Score")
plt.ylabel("Features")
plt.show()
```



Adaboost Algorithm

1 Importing the Libraries

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

2 Load dataset

In [127…
```
# 1. Dataset Load karna
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = data.target
```

In [129…
```
X
```

Out[129…

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	sy
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	
...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	

569 rows × 30 columns

In [131…

```
y
```

Out[131…
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1])
```

3 Train test split

In [134…
```
# 2. Data ko Train aur Test mein split karna (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

4 Training

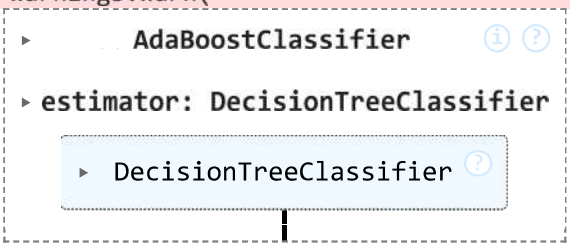
In [138…
```
# 3. Base Learner define karna (Decision Stump: depth=1)
# AdaBoost by default DecisionTreeClassifier(max_depth=1) hi use karta hai
base_estimator = DecisionTreeClassifier(max_depth=1)
```

In [140…
```
# 4. AdaBoost Classifier initialize aur train karna
# n_estimators: kitne stumps banane hain
# learning_rate: har stump ka contribution kitna hoga
ada_model = AdaBoostClassifier(estimator=base_estimator,
                               n_estimators=100,
                               learning_rate=1.0,
                               random_state=42)
```

In [142…
```
ada_model.fit(X_train, y_train)
```

```
C:\Users\swati\anaconda3\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be removed
in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
```

Out[142…



prediciton

In [145…
```
# 5. Predictions
y_pred = ada_model.predict(X_test)
```

In [147…
```
y_pred
```

Out[147…
```
array([1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 0])
```

In [149…
```
y_test
```

Out[149…
```
array([1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 0])
```

Accuracy

In [152…
```
# 6. Model Evaluation
print(f"AdaBoost Accuracy Score: {accuracy_score(y_test, y_pred) * 100:.2f}%")
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
AdaBoost Accuracy Score: 97.37%

Confusion Matrix:
 [[41  2]
 [ 1 70]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.95      0.96        43
           1       0.97      0.99      0.98        71

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```

K means Clustering

In [168…
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
```

In [170…
```
# 1. Data Load karna (Flowers ke measurements)
iris = load_iris()
X = iris.data[:, :2] # Sirf pehle 2 features le rahe hain (Sepal length aur width)
```

In [172…
```
X
```

Out[172…

```
array([[5.1, 3.5],
       [4.9, 3. ],
       [4.7, 3.2],
       [4.6, 3.1],
       [5. , 3.6],
       [5.4, 3.9],
       [4.6, 3.4],
       [5. , 3.4],
       [4.4, 2.9],
       [4.9, 3.1],
       [5.4, 3.7],
       [4.8, 3.4],
       [4.8, 3. ],
       [4.3, 3. ],
       [5.8, 4. ],
       [5.7, 4.4],
       [5.4, 3.9],
       [5.1, 3.5],
       [5.7, 3.8],
       [5.1, 3.8],
       [5.4, 3.4],
       [5.1, 3.7],
       [4.6, 3.6],
       [5.1, 3.3],
       [4.8, 3.4],
       [5. , 3. ],
       [5. , 3.4],
       [5.2, 3.5],
       [5.2, 3.4],
       [4.7, 3.2],
       [4.8, 3.1],
       [5.4, 3.4],
       [5.2, 4.1],
       [5.5, 4.2],
       [4.9, 3.1],
       [5. , 3.2],
       [5.5, 3.5],
       [4.9, 3.6],
       [4.4, 3. ],
       [5.1, 3.4],
       [5. , 3.5],
       [4.5, 2.3],
       [4.4, 3.2],
       [5. , 3.5],
       [5.1, 3.8],
       [4.8, 3. ],
       [5.1, 3.8],
       [4.6, 3.2],
       [5.3, 3.7],
       [5. , 3.3],
       [7. , 3.2],
       [6.4, 3.2],
       [6.9, 3.1],
       [5.5, 2.3],
       [6.5, 2.8],
       [5.7, 2.8],
```

```
       [6.3, 3.3],
       [4.9, 2.4],
       [6.6, 2.9],
       [5.2, 2.7],
       [5. , 2. ],
       [5.9, 3. ],
       [6. , 2.2],
       [6.1, 2.9],
       [5.6, 2.9],
       [6.7, 3.1],
       [5.6, 3. ],
       [5.8, 2.7],
       [6.2, 2.2],
       [5.6, 2.5],
       [5.9, 3.2],
       [6.1, 2.8],
       [6.3, 2.5],
       [6.1, 2.8],
       [6.4, 2.9],
       [6.6, 3. ],
       [6.8, 2.8],
       [6.7, 3. ],
       [6. , 2.9],
       [5.7, 2.6],
       [5.5, 2.4],
       [5.5, 2.4],
       [5.8, 2.7],
       [6. , 2.7],
       [5.4, 3. ],
       [6. , 3.4],
       [6.7, 3.1],
       [6.3, 2.3],
       [5.6, 3. ],
       [5.5, 2.5],
       [5.5, 2.6],
       [6.1, 3. ],
       [5.8, 2.6],
       [5. , 2.3],
       [5.6, 2.7],
       [5.7, 3. ],
       [5.7, 2.9],
       [6.2, 2.9],
       [5.1, 2.5],
       [5.7, 2.8],
       [6.3, 3.3],
       [5.8, 2.7],
       [7.1, 3. ],
       [6.3, 2.9],
       [6.5, 3. ],
       [7.6, 3. ],
       [4.9, 2.5],
       [7.3, 2.9],
       [6.7, 2.5],
       [7.2, 3.6],
       [6.5, 3.2],
       [6.4, 2.7],
```

```
       [6.8, 3. ],
       [5.7, 2.5],
       [5.8, 2.8],
       [6.4, 3.2],
       [6.5, 3. ],
       [7.7, 3.8],
       [7.7, 2.6],
       [6. , 2.2],
       [6.9, 3.2],
       [5.6, 2.8],
       [7.7, 2.8],
       [6.3, 2.7],
       [6.7, 3.3],
       [7.2, 3.2],
       [6.2, 2.8],
       [6.1, 3. ],
       [6.4, 2.8],
       [7.2, 3. ],
       [7.4, 2.8],
       [7.9, 3.8],
       [6.4, 2.8],
       [6.3, 2.8],
       [6.1, 2.6],
       [7.7, 3. ],
       [6.3, 3.4],
       [6.4, 3.1],
       [6. , 3. ],
       [6.9, 3.1],
       [6.7, 3.1],
       [6.9, 3.1],
       [5.8, 2.7],
       [6.8, 3.2],
       [6.7, 3.3],
       [6.7, 3. ],
       [6.3, 2.5],
       [6.5, 3. ],
       [6.2, 3.4],
       [5.9, 3. ]])
```

In [174…

```
# 2. Model banana aur fit karna
# Humein pata hai Iris mein 3 tarah ke phool hote hain, isliye K=3 rakhte hain
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)
```

```
C:\Users\swati\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
```

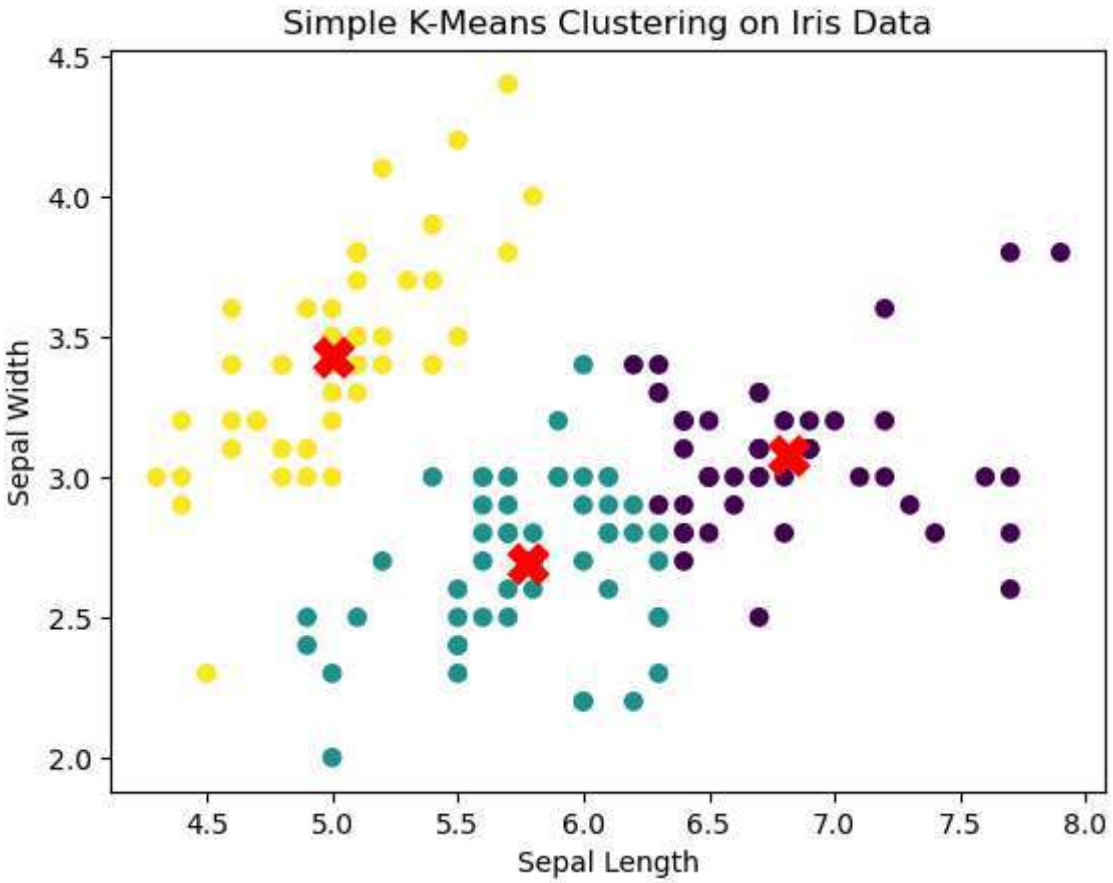
Out[174…

KMeans

```
KMeans(n_clusters=3, random_state=42)
```

In [176…
```
# 3. Predictions lena (Ki kaunsa phool kis group/cluster mein gaya)
y_kmeans = kmeans.predict(X)
centers = kmeans.cluster_centers_ # Ye humein centroids (center points) dega
```

In [178…
```
# 4. Result dikhana (Visualization)
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, cmap='viridis') # Data points
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, marker='X') # Centroids
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.title('Simple K-Means Clustering on Iris Data')
plt.show()
```

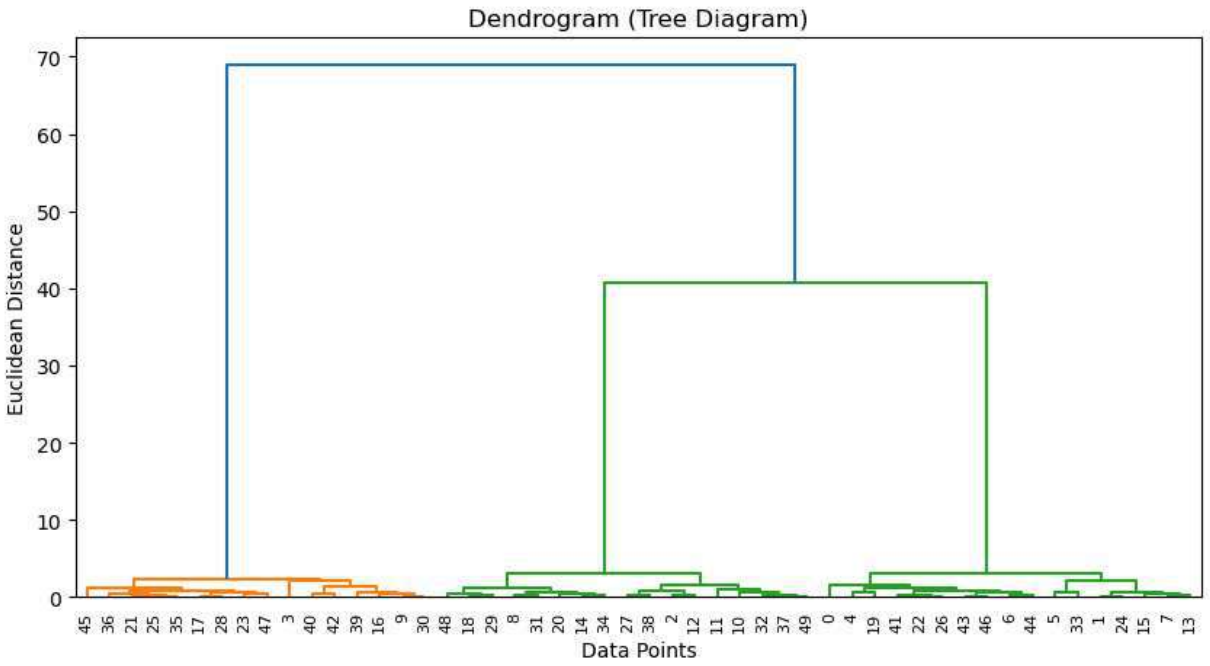


Hierarchical Clustering

In [181…
```
import matplotlib.pyplot as plt
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import make_blobs
```

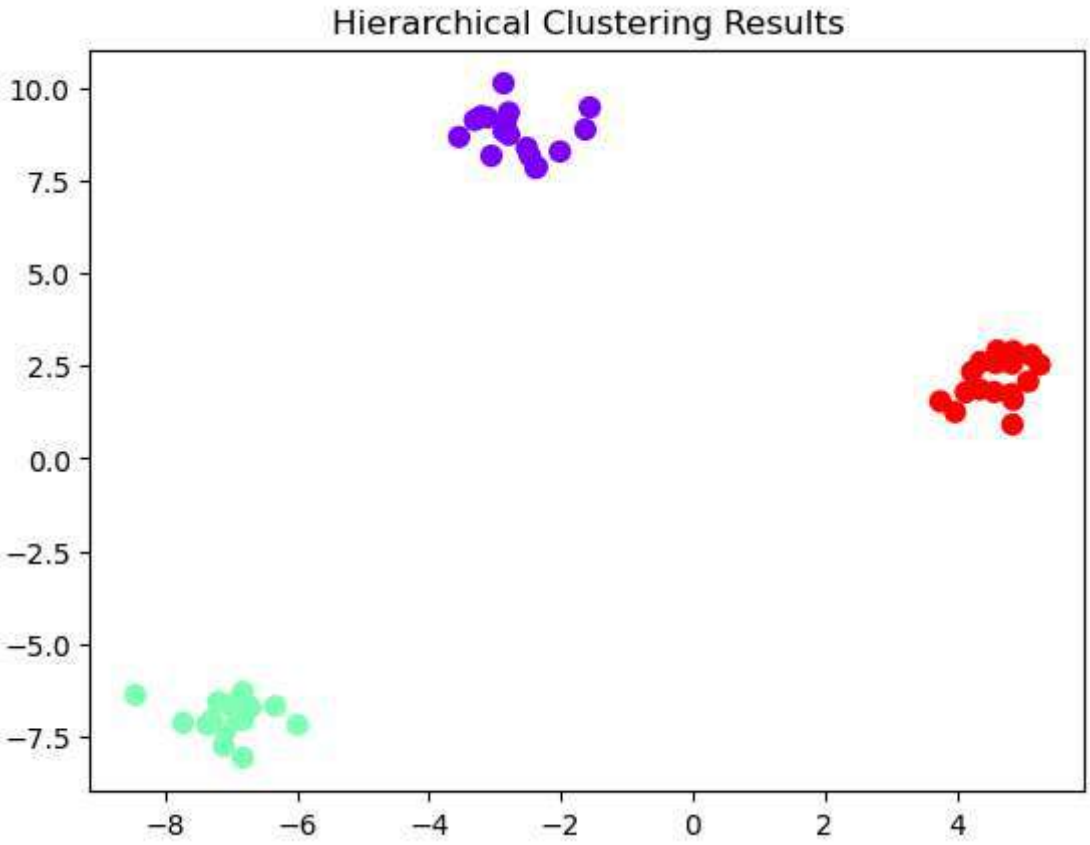
In [185…
```
# 1. Dummy Data banana (50 points, 3 groups)
X, _ = make_blobs(n_samples=50, centers=3, cluster_std=0.6, random_state=42)

# 2. Dendrogram banana (Humein batata hai ki kitne clusters hone chahiye)
plt.figure(figsize=(10, 5))
dendrogram = sch.dendrogram(sch.linkage(X, method='ward'))
```

```
plt.title('Dendrogram (Tree Diagram)')
plt.xlabel('Data Points')
plt.ylabel('Euclidean Distance')
plt.show()
```



In [188…

```
# 3. Model train karna (Clusters ki sankhya humne Dendrogram dekh kar 3 rakhi hai)
model = AgglomerativeClustering(n_clusters=3, linkage='ward')
y_predict = model.fit_predict(X)

# 4. Results visualize karna
plt.scatter(X[:, 0], X[:, 1], c=y_predict, cmap='rainbow', s=50)
plt.title('Hierarchical Clustering Results')
plt.show()
```



DBSCAN Algorithm

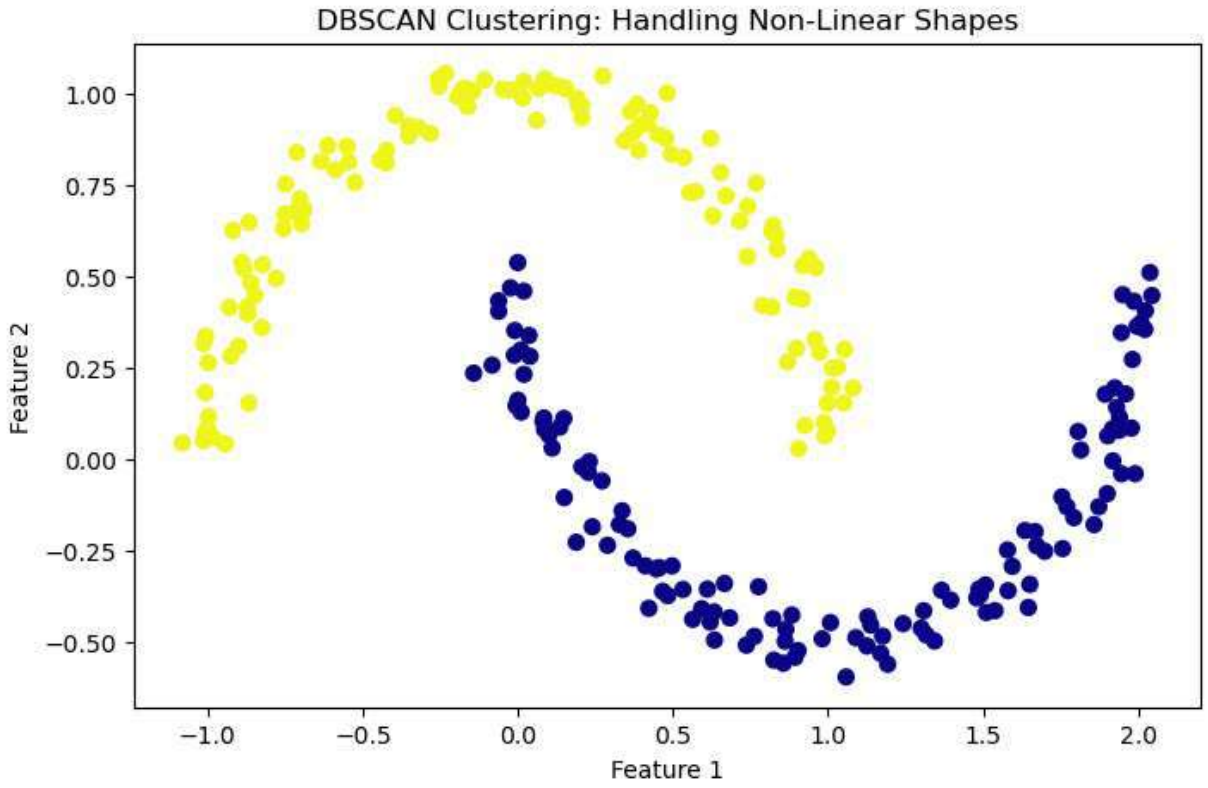
In [201…
```
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_moons
```

In [203…
```
# 1. Ajeeb shape wala data generate karna (Moon shape)
X, _ = make_moons(n_samples=250, noise=0.05, random_state=42)
```

In [205…
```
# 2. DBSCAN Model banana
# eps: Padosi dhundne ki radius (duri)
# min_samples: Cluster banane ke liye kam se kam kitne points chahiye
dbscan = DBSCAN(eps=0.2, min_samples=5)
```

In [207…
```
# 3. Model ko fit karna aur clusters predict karna
y_pred = dbscan.fit_predict(X)
```

In [209…
```
# 4. Result visualize karna
# Note: Cluster ID -1 matlab wo points 'Noise' (Outliers) hain
plt.figure(figsize=(8, 5))
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='plasma', s=40)
plt.title("DBSCAN Clustering: Handling Non-Linear Shapes")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```


DBSCAN Clustering: Handling Non-Linear Shapes
Feature 2
Feature 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: