Udemy Nvidia

Udemy Nvidia

mean(residual) = 0 → unbias

Primary Goal of ML - To extract insights from data

Raw Data
↓
Train–Test Split
↓
Fit preprocessing on TRAIN only
  ├─ Imputation
  ├─ Encoding
  ├─ Scaling / Normalization
  ├─ Log transform (if skewed)
↓
Transform TRAIN and TEST
↓
Feature Engineering (on train, then apply to test)
↓
Feature Selection / PCA (fit on train, apply to test)
↓
Model Training

Regression - predict numeric value

Classification - predict category

Label - output

SGD - disadv - introduce noise

BLEU - depends on Precision(word to word) (LLM)

Rough - depends on Recall(summary context)

Perceptron - basic unit of neural n/w

Techniques that combat underfitting include:

Technique	Why it helps

➡ Add more features	**Gives the model more relevant information to learn patterns**
➡ Increase model complexity (more layers, more neurons)	**Allows the model to learn more complex functions**
➡ Reduce regularization (lower L1/L2, reduce dropout)	**Prevents the model from being too constrained**
➡ Train longer (more epochs)	**Allows the model to learn deeper representations**
➡ Use better feature engineering	**Provides the model clearer, higher-quality inputs**
➡ Use a stronger model type	**Switching from linear → tree model → deep model improves learning**

Regularisation - **prevent overfitting -**“Stop the model from memorizing. Make it generalized.” [L1(apply absolute value as penalty), L2(add squa re value of penalty), Elastic Net(L1+L2), dropout, robust dataset]

Activation f’n - to add non linearity in model

Residual (error) = Actual - prediction , when mean(residual) = 0 i.e. model is unbiased

Logistic Regression(Binary classification) - gives output in binary form 0 or 1 , use sigmoid function,Classification Algorithm then WHY Regression bcoz it uses linear equation internally [Linear regression + sigmoid = logistic regression]

Gradient Decent - We must find the best values for **parameters/weights that minimize error.Gradient Descent is the tool that finds those best values**.

S**MOTE** = Synthetic Minority Oversampling Technique,It artificially generates extra rows for the **minority class** (y=1). Result - balanced dataset

Confusion Matrix - Fundamental evaluation structure - Evaluation metrics for Classification

Required for calculating precision, recall(how many TP catch), F1

Key terms: TP, TN, FP, FN

TP → Actual positive, predicted positive (actual is 1 , predict 1) true +ve

TN → Actual negative, predicted negative   (actual is 0 , predict 0)

FP → Type-1 mistake (predict positive but wrong)   (actual is 0 , predict 1)

FN → Type-2 mistake (predict negative but wrong) (actual is 1 , predict 0)

Accuracy - Out of all predictions, how many were correct?
Precision - **How reliable are the model's positive predictions?**Out of all predicted positives, how many are truly positive? TP ÷ (TP + FP)
Recall - How many real positive cases did we catch? TP ÷ (TP + FN)

F1 Score - Balances precision and recall.
Useful when:Dataset is imbalanced, Both FP and FN matter

Task	Loss Function	Evaluation Metrics
Classification	Cross-entropy, Hinge(only for SVM),Label Smoothing loss (reduce overconfident), FOCAL(focus on hard to classify sample)	Confusion matrix, Precision, Recall, F1, AUC
Regression	MSE, MAE, RMSE, Huber	MSE, RMSE, MAE, R^2, MAPE

ROC Curve: Plots: **TPR (Recall)** vs. **FPR**

□ Loss Functions — Description & Use Cases (Exam Level)

Loss Function	Used For	What It Does (Conceptually)	Key Strength / Why Used	Typical Use Case

Cross-Entropy Loss	Classification	Measures difference between predicted probability distribution and true labels	Strong probabilistic interpretation	Multi-class & binary classification
Binary Cross-Entropy (Log Loss)	Binary classification	Penalizes confident wrong predictions heavily	Stable gradients	Spam detection, fraud detection
Categorical Cross-Entropy	Multi-class classification	Extension of cross-entropy for multiple classes	Works with softmax outputs	Image / text classification
Sparse Categorical Cross-Entropy	Multi-class classification	Same as categorical CE but with integer labels	Memory efficient	Large-class problems
Mean Squared Error (MSE)	Regression	Squares prediction error	Penalizes large errors strongly	Price prediction, forecasting
Mean Absolute Error (MAE)	Regression	Absolute difference between prediction and target	Robust to outliers	Noisy regression data
Huber Loss	Regression	Combines MSE and MAE	**Balance between robustness and smoothness**	Regression with some outliers
Hinge Loss	Classification (SVM)	**Maximizes margin** between classes	Strong margins	Support Vector Machines
Focal Loss	Classification (imbalanced data)	**Focuses on hard-to-classify samples**	Handles class imbalance	Object detection, rare events
Label Smoothing Loss	Classification	Softens target labels	**Reduces over-confidence**	Large LLMs, image classifiers
Contrastive Loss	Metric learning	Pulls similar samples together, pushes dissimilar apart	Embedding learning	Siamese networks

Triplet Loss	Metric learning	Enforces relative distance between anchor/positive/negative	Strong embedding separation	Face recognition
KL Divergence Loss	Distribution matching	**Measures divergence between distributions**	Knowledge distillation	Teacher–student models
Negative Log Likelihood (NLL)	Probabilistic models	Penalizes unlikely predictions	Stable probabilistic modeling	Language modeling
CTC Loss	Sequence modeling	Aligns input-output of different lengths	No explicit alignment needed	Speech recognition
Diffusion Loss	Generative models	Learns noise removal step-by-step	Stable image generation	Diffusion models
Reconstruction Loss	Autoencoders	Measures how well input is reconstructed	Feature learning	Dimensionality reduction

Training Optimization Techniques (Exam Level)

#	Technique	What it Optimizes	How it Works (Simple)	When to Use	Exam Keywords
1	**Mixed Precision Training**	Memory + Speed	Uses FP16/FP8 instead of FP32 where safe	GPU training, large models	FP16, Tensor Cores
2	**Gradient Accumulation**	Memory	Accumulates gradients over steps to simulate large batch	Small GPU memory	Effective batch size
3	**Data Parallelism**	Speed	Split data across multiple GPUs	Multi-GPU systems	Data split
4	**Model Parallelism**	Memory	Split model layers across GPUs	Very large models	Layer partitioning
5	**Pipeline Parallelism**	Speed + Memory	Different GPUs handle different	Deep models	Micro-batches

			stages		
6	**Activation Checkpointing**	Memory	Recompute activations instead of storing	Memory-bound training	Recompute tradeoff
7	**Gradient Clipping**	Stability	Limits gradient magnitude	Exploding gradients	Max norm
8	**Learning Rate Scheduling**	Convergence	Adjust LR over time	Faster, stable training	Cosine, step
9	**Weight Initialization**	Convergence	Proper initial weights	Deep networks	Xavier, He
10	**Batch Normalization**	Stability	Normalizes activations	Faster convergence	Internal covariate shift
11	**Layer Normalization**	Stability	Normalizes per layer	Transformers	Sequence models
12	**Early Stopping**	Overfitting	Stop training early	Limited data	Validation loss
13	**Label Smoothing**	Generalization	Softens hard labels	Classification	Reduce over-confidence
14	**Regularization (L1/L2)**	Overfitting	Penalizes large weights	High variance models	Weight decay
15	**Dropout**	Overfitting	Randomly drops neurons	Dense networks	Noise injection
16	**Optimizer Choice (Adam, SGD)**	Convergence speed	Adaptive vs non-adaptive	Most DL training	Momentum
17	**Quantization-Aware Training (QAT)**	Inference efficiency	Train with low-precision awareness	INT8 deployment	Calibration

18	**Pruning**	Model size	Remove low-importance weights	Edge deployment	Sparsity
19	**Distributed Training (NCCL)**	Speed	GPU-GPU communication	Large clusters	AllReduce
20	**Caching / Prefetching**	I/O bottleneck	Overlap data loading	Data-heavy training	Input pipeline

Decision Tree = A **Decision Tree** is a model that asks **a series of questions** to **split the data into meaningful groups**. (2 types Regression Decision, Classification Decision) (split criteria - Gini Impurity , Entropy (Information Gain))

Pruning is used to reduce overfitting in decision tree algorithms.

Entropy - measure of randomness in dataset, so higher the entropy, harder to fetch information from the dataset. When entropy =1 -> probability = 0.5, probability = 1 -> entropy (randomness in dataset) = 0

Information gain = difference between root node entropy and child nodes entropy

Gini impurity Index = use for evaluating splits in the data (finding the error during classification)**Gini Impurity aims to minimize impurity in decision trees.**

Overfitting - Training score → **Very high,** Test score → **Low**
Underfitting - Training score → Low, Test score → Low
K-Fold CV - helps us test the model **multiple times** on different subsets. (k-1 - training, kth - testing) **Kfold crossvalidation helps prevent overfitting by providing more robust estimates of model performance.**

Hyperparameters - are the "settings" of an algorithm — not learned from the data.
Example:In Decision Tree → depth, criterion,In Linear Regression → penalties (L1/L2)
In Neural Networks → learning rate, batch size
Models learn parameters (weights), but YOU choose hyperparameters.
Types of Hyperparameter -
Training Hyp…

Hyperparameter	**What it controls**
Learning rate	How big each learning step is
Batch size	How many samples are processed at once
Epochs	How many times the model sees the data
Optimizer	How gradients update weights (Adam, SGD)

Regularization Hyp..

Hyperparameter	**Purpose**
Dropout	Prevent overfitting
L1 / L2 weight decay	Penalize large weights
Early stopping patience	Stop training early

LLM inference Hyp…

Hyperparameter	**Effect**
Temperature	Controls randomness
Top-k	Limits token choices
Top-p (nucleus)	Probability-based filtering
Max tokens	Output length

Model Architecture Hyperparameter

Hyperparameter	**Example**
Number of layers	**12 vs 24 transformer layers**
Hidden size	**768, 1024**
Number of attention heads	**8, 12, 16**
Dropout rate	**0.1, 0.3**

Boosting models (e.g., XGBoost) allow:

A) Regularization
B) Parallel tree construction
C) Feature importance analysis
D) Deep neural network layers

Correct Answer: A, B, C

First: Clear the BIG misconception

> "Boosting always increases overfitting"

! Truth:

- **Classic boosting** (AdaBoost) can overfit
- **Modern boosting (XGBoost, LightGBM)** is **explicitly designed to CONTROL overfitting**

That's why NVIDIA exams often mention **XGBoost**, not just "boosting".

Now let's justify each option properly 👇

A) Regularization — YES (very important)

You're right that boosting increases model complexity — **but XGBoost adds regularization to control it**.

XGBoost explicitly supports:

- **L1 regularization** (alpha)
- **L2 regularization** (lambda)
- Tree penalties
- Max depth
- Min child weight
- Subsampling

👉 This is **not present in basic boosting**, but **is a core feature of XGBoost**.

Exam line:

> **XGBoost uses regularization to reduce overfitting despite sequential learning.**

A is correct

B) Parallel tree construction — YES (XGBoost-specific)

This is where your confusion is totally valid 👌

Traditional boosting:

- Trees built **sequentially**
- Each tree depends on previous errors

XGBoost optimization:

- **Parallelizes split finding within a tree**
- Uses histogram-based & level-wise optimization
- Still logically sequential, but **computationally parallel**

☞ NVIDIA exams expect this nuance.

□ Exam wording matters:

- □ “Fully parallel boosting” → wrong
- □ “Parallel tree construction” → correct (implementation-level)

□ B is correct

□ C) Feature importance analysis — YES

You said:

> “Feature importance is done by bagging & bootstrapping”

That’s **partially true**, but **not exclusive**.

Boosting models ALSO provide feature importance:

- Gain
- Frequency
- Cover

XGBoost can tell:

- Which features split the most
- Which features reduce loss the most

☞ Feature importance is available in:

- Random Forest (bagging)
- XGBoost (boosting)

□ C is correct

□ D) Deep neural network layers — NO

This one is clearly wrong.

- Boosting models are **tree-based**
- They do NOT use:
  - Neurons
  - Layers
  - Backpropagation

Even though XGBoost is powerful, it is **not a neural network**.

□ D is false

□ FINAL CLARIFICATION (THIS IS THE KEY)

Your Concern	Correct Explanation
Boosting increases overfitting	✔ True in theory
XGBoost overfits by default	□ False
XGBoost controls overfitting	□ Regularization
Boosting is fully sequential	□ Implementation is optimized
Feature importance only via bagging	□ Boosting also supports it

□ NVIDIA Exam One-Liner (MEMORIZE)

> **Modern boosting models like XGBoost combine sequential learning with regularization, parallel computation, and feature importance analysis.**

□ Ultra-short memory hook
XGBoost = Boosting + Regularization + Parallelism + Feature importance

KNN Algo - (k nearest neighbor algo - under supervised m/c learning) - use for classification, regression.use case - imputing the missing value and re-semple of dataset. (SMOTE), fast to train but slow to predict.

A) Instance-based algorithm — □ True

KNN is an **instance-based (lazy learning)** algorithm.

- It **stores the entire training dataset**
- Makes predictions by comparing a new point to stored instances
- No explicit model is built

✔ This is a core property of KNN.

B) No training phase — ▯ True

KNN has **no traditional training phase**.

- “Training” only means **storing data**
- All computation happens at **prediction time**
- This is why KNN becomes slow with large datasets

✔ This is a classic exam point.

C) Sensitive to scaling — ▯ True

KNN relies on **distance metrics** (Euclidean, Manhattan, etc.).

- Features with larger scales dominate distance
- Example: `Age (0–100)` vs `Salary (0–1,000,000)`
- Requires **normalization or standardization**

✔ Very important exam concept.

SVM - is an algorithm that draws the **best line** between two groups of points., slow to train but fast to predict., Hinge loss function use(focus on max’m margin)

SVM aims to find the hyperplane with the maximum margin.

- That line = **hyperplane**
- Closest points to the line = **support vectors**
- **Increase C → tighter margin → fewer misclassification**
- **Hyperparameter → C, gamma, kernel type**
- Distance between line and nearest points = **margin**
- SVM tries to **maximize the margin**

Ensemble - group of predictor, Types - Voting, Baggers, Random Forest, Boosting
Voting - aggregation of predictions(Hard Voting), aggregation of probabilities(Soft Voting)
Bagging - building **many versions of the model(Sampling with Replacement)** and then **aggregating their predictions**.
Random forest = Bootstrap samples + Random Feature Selection + Many Decision Trees (Bagging + Extra Randomness)

OOB Score - When bootstrap sampling(in Bagging, Random Forest) is done:
Around **36% of rows are NOT selected**
These left-out rows are called **OOB samples (Out-of-Bag Score)**

Boosting - is a machine learning technique where **many weak models are trained one after another**, and **each new model fixes the mistakes** made by the previous one.
All models are then combined to make a **strong, accurate final model**.

Types - AdaBoost (Adaptive Boosting) [weight updated], Gradient Boosting (GBM) [reduce residual], XGBoost (Extreme Gradient Boosting) [Improved & faster version of Gradient Boosting.use Regularization]

Clustering - grouping the objects or data points based on the information found on the data which describe the information or relation about them. Goal - the object of one group similar to each other and objects between different groups as much as dissimilar objects.
Internal cohesion - similarity inside groups of objects, external separation - dissimilarity between groups
TYPES - Exclusive Clustering (Exp - KMeans), Overlapping Clustering(Fuzzy/c - means), Hierarchical Clustering
From sklearn.cluster import KMeans
KMean - only predict spherical data

distance metric - we will find the similarity between one data point to the another data point. And then we will decide as to which group it would belong to.

Exam Questions -
> Cross validation useful when - data is limited
> KNN & SVM use for →

Algorithm	Supervised /	Classificatio	Regression

	Unsupervised	n	
KNN	✔ **Supervised**	✔ **Yes**	✔ **Yes**
SVM	✔ **Supervised**	✔ **Yes**	✔ **Yes (SVR)**
Decision Tree	✔ **Supervised**	✔ **Yes**	✔ **Yes**
Random Forest	✔ **Supervised**	✔ **Yes**	✔ **Yes**
Logistic Regression	✔ **Supervised**	✔ **Yes**	▯ **No**
Linear Regression	✔ **Supervised**	▯ **No**	✔ **Yes**
Naive Bayes	✔ **Supervised**	✔ **Yes**	▯ **No**
Gradient Boosting	✔ **Supervised**	✔ **Yes**	✔ **Yes**
XGBoost	✔ **Supervised**	✔ **Yes**	✔ **Yes**
AdaBoost	✔ **Supervised**	✔ **Yes**	✔ **Yes**
K-Means	✔ **Unsupervised**	▯ **No**	▯ **No**
PCA	✔ **Unsupervised**	▯ **No**	▯ **No**
Autoencoder	✔ **Unsupervised**	▯ **No**	▯ **No**
DBSCAN	✔ **Unsupervised**	▯ **No**	▯ **No**
Neural Networks (MLP)	✔ **Supervised**	✔ **Yes**	✔ **Yes**

Important points -
✔ Focus more on **conceptual usage**, not mathematical derivations
✔ PCA, K-Means → **Dimensionality reduction & clustering**
✔ XGBoost → **High-performance supervised learning**
✔ Neural Networks → **Foundation for LLMs & deep learning**

MCQ -
1)What is the term used to describe the phenomenon where predictors in a linear regression model are highly correlated with each other? - Multicollinearity
2)In logistic regression, what function is used to map the predicted values between 0 and 1? - Sigmoid Function
3)What is the main goal of the gradient descent algorithm? - Minimizing the Cost Function

4)What is a common technique for exploratory data analysis (EDA) when preparing data for logistic regression? - Feature Scaling
5)In a decision tree algorithm, what is the primary purpose of splitting nodes based on certain features? - Splitting nodes aims to maximize information gain.
6)What is the loss function commonly used in decision tree algorithms for regression tasks? - MSE(Mean Square Error) is commonly used as the loss function for regression in decision trees.
7)Which factor determines the **size of steps taken during each iteration** of the gradient descent algorithm? - Learning rate determines the step size in gradient descent.
8)**What technique is used to handle categorical variables in logistic regression during data preprocessing? -** OneHot Encoding is commonly used for categorical variables in logistic regression.
9)Which evaluation metric is influenced by both false positives and false negatives in a binary classification problem? - **F1 Score considers both false positives and false negatives. (use for imbalance dataset)**

10)What technique is used to handle missing values in decision tree algorithms during data preprocessing? - Imputation is often used to handle missing values in decision trees.
11)In classification tasks, what is the objective of the Gini Impurity criterion used in decision trees? - To minimize impurity
12)What is the purpose of pruning in decision tree algorithms? - **Pruning is used to reduce overfitting in decision tree algorithms.**
13)Which hyperparameter optimization technique randomly selects a combination of hyperparameters and evaluates them independently?-Random Search evaluates hyperparameter combinations independently.

14)What is the primary purpose of kfold crossvalidation in machine learning? - **Kfold crossvalidation helps prevent overfitting by providing more robust estimates of model performance.**
15) Regularization techniques are used to combat underfitting in machine learning models. - False, it’s use for overfitting
16)In the K-nearest neighbors (KNN) algorithm, what determines the class of a new data point? - **Majority Vote : The class of a new data point in KNN is determined by majority voting.**
17)What is the primary goal of a support vector machine (SVM) algorithm in binary classification tasks? - SVM aims to find the hyperplane with the maximum margin.

Introduction of ML:

Topic	Definition	Why It Matters	Where Used	Key Concept	Notes
Machine Learning	System learns patterns from data	Automates decision-making	Predictive analytics	Input → Model → Output	Core to AI/LLMs
Traditional Programming	Human writes rules manually	Limited, non-scalable	Small systems	Hard-coded logic	Poor with evolving data
Training	Model learns from data	Build predictive model	Regression/ classification	Fit(X, Y)	Needs labeled data
Prediction	Using trained model on new data	Generates outputs	Real-world apps	model.predict(X_new)	Requires generalization
Dataset	Collection of input-output pairs	Needed for supervised ML	All ML tasks	Features + Labels	Quality matters
Features	Descriptive characteristics (X)	Input to learn from	Numeric/ categorical data	Preprocessing needed	Feature engineering
Label	Target outcome (Y)	Guides prediction	Classification/ regression	Must match task	Supervised learning
Rule-based System	Handwritten conditions	Hard to maintain	Legacy apps	IF-ELSE logic	Replaced by ML
Generalization	Model performs	Prevents	Deployment	Train/Val split	Model quality

	well on new data	overfitting		metric

Type of ML:

Supervised(X + Y) - 1)Regression(Predict numbers) 2)Classification (Predict Category)
Unsupervised(X only) - 1)clustering 2)Data structure
RL (Reinforcement) - Automatic CAR - Agent Environment learning - Trial & Error

Use **supervised ML** when labels are available and target output is known.

Use **unsupervised ML** when exploring data, finding segments, or detecting anomalies.

Use **RL** when agent-environment interaction with reward maximization is required.

Reinforcement learning requires significant computation and is used primarily in specialized use cases (robotics, game AI).

Most enterprise business ML workloads use supervised + unsupervised learning, not RL.

Type	Definition	Data Required	Algorithms	Use Cases	Notes
Supervised ML	Learn mapping from X → Y	Labeled data	Linear Regression, Logistic Regression, Random Forest, SVM, CNN, Transformers	Spam detection, fraud, price prediction, image classification	Most common ML type
Unsupervised ML	Find hidden patterns w/o labels	Unlabeled data	K-means, PCA, DBSCAN, Hierarchical	Clustering, segmentation, anomaly detection	Used for discovery

Reinforcement Learning	Learn behavior from rewards	States, actions, rewards	Q-learning, Policy Gradients, Deep RL	Robotics, games, self-driving	Research-heavy
Clustering	Group similar points	X only	K-means	Customer segmentation	Part of unsupervised
Regression	Predict continuous output	X + Y	Linear Regression	Price prediction	Supervised
Classification	Predict categories	X + Y	Logistic Regression, SVM	Spam detection	Supervised
MDP (RL)	Model of RL environment	State/Action/ Reward	Q-Learning	RL tasks	Foundation of RL

Linear Regression -

Topic	Definition	Why Used	Formula	Where Used	Notes
Linear Regression	Maps $X \rightarrow Y$ using linear equation	Predict continuous values	$\hat{y} = a + bX$ (simple)	Pricing, forecasting	Simple, interpretable
Multiple Linear Regression	Linear model with multiple features	Complex predictions	$\hat{y} = \theta_0 + \Sigma\theta_i X_i$	Marketing mix modeling	Produces coefficients
MAE	Avg absolute error	Easy to interpret	mean(	y - yp	)
MSE	Avg squared error	Penalizes large errors	$\text{mean}((y - yp)^2)$	Training loss	Common in ML training
RMSE	Square root of MSE	Same unit as target	$\sqrt{MSE}$	Industry reporting	Most preferred metric
R^2 Score	Variance explained by model	Measures goodness of fit	1 - (RSS/TSS)	Regression eval	Higher is better

Linear Regression Flow

```
Data → Train/Test Split → Train Model → Coefficients → Predict → Evaluate
```

---

2. Equation Diagram

```
ŷ = θ0 + θ1X1 + θ2X2 + … + θnXn
```

---

3. Evaluation Metrics Mind Map

```
Regression Metrics
   ├── MAE → average absolute error
   ├── MSE → squared error penalty
   ├── RMSE → sqrt(MSE)
   └── R² → variance explained
```

---

4. R² Score Interpretation

```
R² = 1 → Perfect
R² = 0 → No relationship
0 < R² < 1 → Some relationship
Negative R² → Model is worse than a horizontal line
```

Regularization & Linear Regression -

Regularization

- Prevents overfitting
- L1 (**Lasso)** → coefficient shrinkage + feature elimination
- L2 **(Ridge)** → coefficient shrinkage, no elimination
- Elastic Net → hybrid of L1 + L2

- Regularization adds a penalty term to the loss function
- Hyperparameter λ (or α) controls penalty strength

Linear Regression Assumptions

- Relationship between features & target is linear
- Residual mean ≈ 0
- Residuals follow normal distribution
- No multicollinearity (checked using VIF)
- Higher VIF → remove feature
- R^2 measures variance explained
- Use correlation/Pearson coefficient to check linearity

Topic	What It Means	Why Used	Formula / Rule	Exam Notes
L1 Regularization (Lasso)	Adds absolute value of coefficients	Feature selection, sparsity	Loss = MSE + λ	θ
L2 Regularization (Ridge)	Adds square of coefficients	Stabilizes model, prevents overfitting	Loss = MSE + $\lambda\theta^2$	Coeffs shrink but $\neq 0$
Elastic Net	L1 + L2 combined	Handles correlated features	Loss = MSE + α	θ
Linearity Assumption	X and Y must have linear relationship	Ensures correct model fit	Check pairplot/correlation	Important
Residual Mean Zero	Avg error = 0	Unbiased model	mean($y-\hat{y}$) ≈ 0	Must satisfy
Normal Errors	Residuals follow bell curve	Reliable predictions	Plot histogram	Required
No	Features	Avoid confusion	VIF < 4	High

Multicollinearity	shouldn't correlate	in model		priority
Homoscedasticity	Constant error variance	Stable predictions	Residual plot	Nice-to-have

Logistic Regression - Classification

Topic	Definition	Why Used	Formula / Core Idea	Extra Notes
Logistic Regression	Binary classifier	Predict 0/1 outcomes	sigmoid(θX)	Outputs probabilities
Sigmoid Function	Maps any value to 0–1	Converts linear output → probability	1/(1+e^-z)	S-shaped curve
Threshold	Cut-off probability	Convert probability → class	p≥0.5 →1	Business can tune
Linear vs Logistic	Regression vs Classification	Logistic handles binary tasks	Logistic uses sigmoid	Linear can't limit range
Loss Function	Binary cross entropy	Better for classification	−[y log p + (1−y) log (1−p)]	NVIDIA expects this
Gradient Descent	Optimization method	Learn θ parameters	update θ ← θ − α * gradient	Same as linear regression

Gradient Descent -

Term	NVIDIA Definition
Gradient Descent	Optimization algorithm used to minimize a model's loss function
Gradient	Derivative of loss w.r.t. a parameter
Learning Rate	Step size for weight updates; controls convergence

Loss Function	Tells how wrong the model is; gradient descent reduces it
Update Rule	w ← w – α * ∂L/∂w
Intuition	Moves in direction of steepest descent
Used In	Logistic regression, linear regression, neural networks

Topic	Definition	Why Used	Formula / Core Idea	Additional Metadata
Gradient Descent	Optimization algorithm	Minimize loss; improve accuracy	w ← w – α * ∂L/∂w	Used in ML & DL
Gradient	Derivative of loss w.r.t parameter	Shows direction to move	∂L/∂w	Positive = move left; negative = move right
Learning Rate (α)	Step size hyperparameter	Controls speed of convergence	Small α = slow; big α = unstable	Tuned during training
Loss Function	Measures model error	GD reduces this	Depends on model (MSE, log-loss, cross-entropy)	Key to training quality
Iterative Updates	Repeated weight changes	Helps find minimum loss	Loop until convergence	Stopping criteria required
Convergence	**Reaching minimum loss**	Ensures stable model	Loss stops decreasing	Depends on α & data
Global Minimum	Best possible loss	Perfect training	Ideal loss position	Complex in deep nets
Local Minimum	Sub-optimal low point	Common in DL	GD may get stuck	Solved via momentum/Adam

Logistic Regression Implementation and EDA -

Logistic Regression
↓

EDA → Understand Categorical + Numeric Features
↓
One-Hot Encoding (convert words → numbers)
↓
Imbalanced Data → SMOTE
↓
Feature Selection → RFE
↓
Model Training → Logistic Regression
↓
Prediction + Evaluation → Accuracy, Confusion Matrix

Topic	Meaning	Why Important
EDA	Explore the data patterns	Helps determine cleaning strategy
One-Hot Encoding	Convert text → numbers	Logistic regression requires numeric input
Class Imbalance	Unequal number of 0 & 1	Causes biased models
SMOTE	Balances minority class	Improves recall & fairness
RFE	Rank + select best features	Reduces noise & boosts accuracy
Logistic Regression	Classification algorithm	Predicts probability of class 1
Accuracy = 88%	Final performance	Shows successful training

Evaluate Metrics for Classification

✔ Accuracy

- Best for balanced datasets
- Fails in imbalanced settings

✔ Confusion Matrix

- Fundamental evaluation structure

- Required for calculating precision, recall, F1
- Key terms: TP, TN, FP, FN

✔ Precision

- Model's positive prediction correctness
- Controls "false alarm rate"

✔ Recall (Sensitivity / TPR)

- Model's ability to retrieve actual positives
- Critical in medical / risk detection

✔ F1 Score

- Harmonic mean of precision & recall
- Handles class imbalance

✔ ROC AUC - Receiver Operating Characteristic – Area Under the Curve.

ROC AUC measures a classifier's ability to distinguish between classes by evaluating the trade-off between **true positive rate and false positive rate** across all thresholds.

- Measures separability of classes
- Threshold-independent
- AUC close to 1 shows high discrimination power
- Range:
  - `1.0 → Perfect classifier`
  - `0.5 → Random guessing`
  - `< 0.5 → Worse than random`

Decision Tree

Topic	Definition / Explaination	Why It Matters	NVIDIA Exam Angle
Decision Tree	Splits data via questions to reach a final decision	Simple yet powerful model	Know how splitting works
Root Node	First split	Drives all model behavior	High scoring questions
Leaf Node	Final prediction	Determines final class	End of recursion
Gini Index	Impurity measure	Used for best split	Must know formula
Entropy	Information gain metric	Alternative to Gini	Compare Gini vs Entropy
Classification Tree	Predicts category	Fraud, churn, etc.	They ask examples
Regression Tree	Predicts number	Salary prediction	Splitting logic same
Overfitting	Tree becomes too deep	Poor generalization	Ask how to prevent it
Stopping Criteria	Depth, purity, samples	Prevent overfit	Operational tuning

Overfitting vs Underfitting vs Good Fit

Topic	Meaning	Why Occurs	Symptoms	Fix
Overfitting	Model memorizes training data	High complexity, noisy data	High train score, low test score	Cross-validation, regularization, **pruning(decision tree)**
Underfitting	Model too simple	Low complexity, poor features	Low train score, low test score	Add features, increase complexity
Good Fit	Balanced learning	Optimal tuning	Train & test both good	Early stopping,

hyperparameter tuning

K-Fold Cross Validation Breakdown

Element	Definition	Benefit	Used When	NVIDIA Exam Expectation
K value	Number of splits	Controls rotation cycles	Hyperparameter tuning	Understand ranges (5,10 common)
Fold	A segment of data	Helps balanced evaluation	Small datasets	Conceptual knowledge
K–1 folds	Used for training	Ensures full dataset usage	Model training	Must explain intuition
1 fold	Used for testing	Validates generalization	CV loop	Must interpret

Types of Cross Validation

CV Type	Definition	When to Use
K-Fold	Split into K equal parts	General ML tasks
Stratified K-Fold	Preserves class distribution	Classification imbalance
Leave-One-Out (LOOCV)	Each sample used once as test	Very small datasets
Holdout	Simple train/test split	Quick baseline

Hyperparameter Optimization Technique -

Manual, Random(random chose), Grid(all combination try)

Topic	Definition	Why Use	When Use	Key Risk
Hyperparameter	Model configuration not learned from data	Control model behaviour	Before training	Wrong values → bad model
Manual Tuning	Manually choose values	Quick baseline	Small datasets	Very inefficient

Grid Search CV	Exhaustive sweep of all combinations	**Best accuracy, deterministic**	Small search space	Very slow
Random Search CV	Random sampling from search space	**Fast, scalable**	Large search space	May miss optimal value
CV (K-Fold)	Rotate training/testing partitions	Reduces overfitting	Model validation	High compute cost

Decision Tree Hyperparameters Used

Hyperparameter	Meaning	Example Values	Impact
`criterion`	How split impurity is measured	gini, entropy, log_loss	Controls splitting logic
`max_depth`	Maximum tree depth	2,4,5,6	Prevents overfitting
`min_samples_split`	Minimum samples to split node	2,4,6	Controls tree growth

KNN

KNN = Lazy Learning (no training parameters)
✔ Requires **scaling**
✔ Decision happens using **majority vote** (classification)
✔ Uses **average** (regression)
✔ K is chosen by:

- Error curve
- Cross validation
- Grid search
  ✔ Distance metrics matter
  ✔ Sensitive to outliers and large datasets

SVM

Term	Meaning	Why Important
Hyperplane	Decision boundary	Separates classes

Support Vectors	Points closest to hyperplane	Define margin; model depends on these, not all data
Margin	Distance between support vectors & hyperplane	Larger margin → better generalization
Hard Margin	No misclassification allowed	Only works if data is perfectly separable
Soft Margin	Allows some misclassification	Real-world SVM runs soft margin
Kernel	Function that transforms data	Handles nonlinear patterns
C (Regularization)	Controls margin softness	Low C → large margin, High C → tighter fit
Gamma (Kernel coefficient)	Defines influence of single point	High γ overfits; low γ underfits

SVM Kernel Comparison

Kernel	Use Case	Shape	Notes
Linear	Linearly separable data	Straight line	Fastest
RBF (Gaussian)	Most real-world scenarios	Curved	Default & most powerful
Polynomial	Multi-power interactions	Curved complex	More expensive
Sigmoid	Neural-net-like behavior	Curved	Rarely best

Esemble Learning -

Types - Voting, Beggar,

Definition

Voting Classifier = ensemble method that combines predictions of multiple base models using majority voting or probability averaging.

Why we use it

- Improves accuracy
- Reduces overfitting
- Stabilizes predictions

Hard vs Soft Voting

Hard Voting	Soft Voting
Majority vote	Average probabilities
No need for probability outputs	Requires `predict_proba`
More stable with diverse models	Often more accurate

Sklearn Class

```
from sklearn.ensemble import VotingClassifier
```

Requirement for Soft Voting

All estimators must support:
`predict_proba()`
(For SVM → `probability=True`)

---

8. Python Logic (Concept)

Hard Voting

```
VotingClassifier(
    estimators=[('lr', lr), ('dt', dt), ('svc', svc)],
    voting='hard'
)
```

Soft Voting

```
VotingClassifier(
    estimators=[('lr', lr), ('dt', dt), ('svc', svc_probability)],
    voting='soft'
)
```

9. Strengths & Weaknesses

Strengths

- Simple to implement
- Improves stability/accuracy
- Works well with small/medium datasets
- No complex tuning required

Weaknesses

- Expensive during inference → multiple models predicting together
- Requires calibrated probabilities for soft voting
- Not ideal for very large datasets in real-time systems

Bagging, Random Forest Esemble

- **Bagging = Bootstrap (sampling with replacement) + Aggregation (vote/average)**
- Reduces variance, stabilizes predictions
- Works best with high-variance models (like Decision Trees)
- **Random Forest = Bagging + Random Feature Selection + Decision Trees**
- Random Forest prevents overfitting better than a single tree
- **OOB Score = internal validation score , When bootstrap sampling is done:**

  Around 36% of rows are NOT selected

  These left-out rows are called OOB samples
- Outputs **Feature Importance**

Boosting Ensemble -

Topic	Definition	Why Use It?	How It Works	Where Used	Formula/Key Idea	Exam Tips
Boosting	Sequential ensemble method	Converts weak learners → strong	Models train one after another	Classification, regression	Each model fixes previous errors	Sequential learning is key word
AdaBoost	Adaptive Boosting	Focuses on difficult rows	Updates weights after each model	Binary classification	New Weight = Old Weight × exp(±performance)	Uses decision stumps
Gradient Boosting	Residual-learning boosting	Very accurate	Each model fits residual errors	Regression, classification	Prediction = sum(all models)	Sensitive to overfitting
XGBoost	Extreme GBM	Fastest & most accurate	Parallel boosting + regularization	All ML competitions, enterprise AI	Regularization + advanced tree growth	Always in top ML algorithms
Weak Learner	A simple imperfect model	Boosting builds many weak → strong	Usually tree depth=1	Used in AdaBoost & GBM	Small decision tree	Key exam term
Residual	Error leftover from last model	Helps next model correct mistakes	Residual = y - y_pred	Gradient Boosting	Fits on residuals	Must know definition
Learning Rate	Controls contribution of each	Prevent overfitting	Smaller LR = more models	GBM, XGBoost	Typical LR: 0.01–0.1	Very important hyperparameter

	model					
Sequential Training	One model after another	Allows error correction	Kills bias	AdaBoost, GBM, XGBoost	—	Key difference from bagging

XGBoost Ensemble

XGBoost (Extreme Gradient Boosting) is an optimized implementation of Gradient Boosting that uses **parallel tree boosting**, **regularization**, and **efficient memory usage**.

□ Key Concepts You Must Know for Exam

Concept	Simple Meaning
Boosting	Train trees sequentially; each tree fixes previous mistakes
DMatrix	Special XGBoost data structure for speed
eta (learning rate)	How fast the model learns; lower = more stable
max_depth	Tree depth; controls model complexity
min_child_weight	Controls overfitting by restricting leaf nodes
objective = binary:logistic	Used for binary classification
eval_metric = auc	Used for performance monitoring

□ Core Strengths of XGBoost (Exam-Focused)

- High accuracy
- Regularization (L1 + L2) prevents overfitting

- Handles missing values automatically
- Works on CPU/GPU
- Built-in cross-validation
- Watchlist for monitoring training vs. validation performance
- Faster than traditional Gradient Boosting

Cluster

K-Means is an unsupervised clustering algorithm that partitions data into **K non-overlapping groups** by minimizing intra-cluster variance.

□ Core Principles

- **Exclusive clustering** (one point → one cluster)
- Objective function: **Minimize Sum of Squared Errors (SSE)**
- Uses **Euclidean distance** as similarity metric
- Highly scalable and fast (O(nkt))

□ Algorithm Steps (Exam Style)

1. Select **K** clusters
2. Initialize centroids (randomly or k-means++)
3. Assign each point to the closest centroid (distance-based)
4. Recompute centroids as mean of assigned points
5. Iterate until convergence (centroids stabilize)

□ Key Metrics

1. **Inertia (SSE)**
    - Internal cohesion

   - Lower inertia = tighter clusters
2. **Silhouette Score**
   - External separation
   - −1-1−1 to 111, higher = better

□ Choosing Optimal K

- Use **Elbow Method** (k vs inertia curve)
- Use **Silhouette Analysis**
- Consider domain knowledge (business context)

□ Limitations (Exam-Important)

- Sensitive to **initialization**
- Sensitive to **feature scaling**
- Works only with **numerical data**
- Assumes **spherical cluster shapes**
- Outliers can disturb cluster centroids

□ Preprocessing Needed for K-Means

- Standardization (StandardScaler)
- Remove outliers
- Convert categorical → numerical if required

□ Applications (Askable in NCA-GENL)

- Embedding clustering
- Similar image grouping
- Customer segmentation
- Prompt dataset grouping
- Topic clustering for LLM pre-processing
- Chunk clustering before RAG

Hierarchical Clustering

Hierarchical Clustering is an **unsupervised learning method** that recursively merges or splits clusters to form a tree-structured grouping (dendrogram).
Does **not** require specifying K in advance.

□ Types

1. **Agglomerative (Bottom-Up)**
   - Start with single-point clusters
   - Merge closest clusters
   - Most widely used
2. **Divisive (Top-Down)**
   - Start with one cluster
   - Recursively split based on dissimilarity

□ Key Concepts

Dendrogram

- Visualization of hierarchical cluster formation
- Cut at specific height to choose number of clusters

Linkage Criteria

- Defines cluster similarity
- Common linkage functions:
    - **Single Linkage:** minimum point-to-point distance
    - **Complete Linkage:** maximum distance
    - **Average Linkage:** average pairwise distances
    - **Centroid Linkage:** distance between centroids

☐ Advantages

- No need to specify K
- Produces hierarchy → interpretable
- Works well for nested cluster structures

☐ Limitations

- Computationally expensive ($O(n^2)$)
- Sensitive to scaling
- Sensitive to noise/outliers
- Hard to correct once clusters merge or split

☐ Implementation (Exam View)

- SciPy: `dendrogram()` + `linkage()`
- sklearn: `AgglomerativeClustering()`
- Evaluate by visually analyzing dendrogram separation height

Title	Definition	Why It Matters	Comment	Formula / Logic	Where Used
Hierarchical Clustering	Builds clusters by merging/splitting recursively	Reveals natural hierarchy in data	No need to predefine K	Tree-structured merging	Customer segmentation, gene analysis
Dendrogram	Tree diagram of cluster merging	Helps decide number of clusters visually	Cut height → cluster count	Distance (vertical height) represents dissimilarity	Data analysis, bioinformatics
Agglomerative Clustering	Build clusters bottom-up	Most common in practice	Computationally heavy	Start: each point = cluster; repeatedly merge	NLP embeddings, image grouping
Divisive Clustering	Top-down splitting of clusters	Good for large clusters	Less common	Start: one cluster; recursively divide	Large datasets, document classification
Single Linkage	Min distance between two clusters	Captures chained patterns	Can form long "chains"	(\min d(x_i, x_j))	Spatial data
Complete Linkage	Max distance between two clusters	Produces compact clusters	Good separation	(\max d(x_i, x_j))	Fraud detection, anomaly detection
Average Linkage	Mean pairwise distance	Balanced cluster structure	Widely used	(\text{avg}(d(x_i, x_j)))	Topic modeling
Centroid Linkage	Distance between centroids	Fast & intuitive	Can violate monotonicity	(d(\mu_1, \mu_2))	Embedding clustering
Cluster Extraction	Cutting dendrogram at height	Visual cluster count	Manual selection	Horizontal cut	Hierarchy interpretation

Time Series

Definition

Time Series Analysis is a statistical technique to model and interpret temporal dependencies where **time acts as the independent variable**.

Data:
$\{y_1 \text{ at } t_1, y_2 \text{ at } t_2, \ldots, y_n \text{ at } t_n\}$ at equal intervals.

Core Components (Critical for Exam)

1. Trend

Long-term systematic increase or decrease in series.

2. Seasonality

Short-term, repeating patterns at fixed intervals.
Periodicity is known (e.g., weekly, monthly).

3. Cyclical

Long-duration, irregular periodicity (business cycles).
Not predictable and varies in length.

4. Irregular / Residual

Random noise caused by unexpected events.

Stationarity (Key Exam Concept)

A time series is **stationary** if:

- Mean is constant
- Variance is constant
- Covariance is time-invariant

Most statistical forecasting models (AR, MA, ARIMA) **assume stationarity**.

Non-stationarity arises from:

- Trend
- Seasonality
- Cyclic patterns
- Structural changes

Stationarity Tests

1. **ADF Test**
   - Null hypothesis: non-stationary
   - p-value < 0.05 → reject H0 → stationary
2. **KPSS Test**
   - Null: stationary
   - p-value < 0.05 → non-stationary

Making Data Stationary

Methods:

- **Differencing** (most robust and widely used)
- **Detrending**
- **Transformation (log, Box-Cox)**

Title	Definition	Why It Matters	Comment	Formula / Logic	Where Used

Time Series	Data indexed by time at equal intervals	Enables temporal forecasting & pattern detection	Time = independent variable	(y_t = f(t))	Forecasting, business KPIs, weather
Trend	Long-term movement (up or down)	Indicates macro behavior	Helps identify growth/decline	—	Sales growth, demand trend
Seasonality	Fixed, repeating periodic pattern	Captures predictable cycles	High frequency pattern	(y_t = f(t \ mod m))	Retail, holidays, temperature
Cyclical Pattern	Non-fixed, long-term waves	Shows economic or market cycles	Hard to predict	—	Business cycles, finance
Irregularity	Random noise or shocks	Helps identify outliers	Caused by unforeseen events	—	Disaster impact, anomalies
Stationary Data	Constant mean, variance, covariance	Required for ARIMA-style models	Improves prediction stability	(E[y_t] = \ text{constant})	Statistical forecasting
Non-Stationary Data	Changing mean/variance	Leads to unreliable models	Must be transformed	—	Trend-heavy datasets
ADF Test	Stationarity statistical test	Determines TS modeling approach	Null = non-stationary	Test statistic & p-value	Pre-processing
KPSS Test	Complementary stationarity test	Validates seasonality/trend	Null = stationary	Test statistic	Validation
Differencing	Subtract previous value from current	Removes trend/seasonality	Most common method	(y'*t = y_t - y*{t-1})	Pre-processing for ARIMA
Detrending	Remove long-term direction	Converts non-stationary → stationary	Often used before differencing	Regression residuals	Long-term forecasting
Transformations	Scale adjustment via log/sqrt	Stabilizes variance	Use cautiously	(y'_t = \ log(y_t))	High variance datasets

—--

OPtimization Techniques

□ What Are Optimization Techniques in Machine Learning?

Optimization techniques are **algorithms that adjust model weights** to minimize the **loss function** (error).

☞ Think of them as “smart ways to update model parameters so the model learns better.”

□ Why Do We Need Optimization?

Because the model guesses something → measures error → **updates weights** to reduce error.

Without optimizers → the model **cannot learn**.

☆ Key Concepts (explained super simply)

1️⃣Learning Rate (α)

How big a step we take when updating weights.

- Too large → overshoots, never learns.
- Too small → learns very slowly.

2️⃣Gradient

It is the “direction of steepest increase.”
We move **opposite of the gradient** to reduce error.

🏋 Types of Optimization Techniques (Super Easy Explanation)

1️⃣ Gradient Descent (GD) — *The simplest one*

- Uses **all data** to calculate gradient.
- Very slow & expensive.

Use when: datasets are small → rarely used in deep learning.

2️⃣ Stochastic Gradient Descent (SGD) — *Fast version*

- Uses **one sample at a time**.
- Faster but noisy updates.

Use when: datasets are large.

3️⃣ Mini-Batch SGD — *Most common*

Uses small batches like 32, 64, 128 samples.

- More stable than SGD
- Faster than GD
  ☞ **Default for most DL models**

⚡ Why do we need smarter optimizers?

SGD has 2 problems:

1️⃣ Gets stuck easily
2️⃣ Learning rate must be hand-tuned
3️⃣ Updates are unstable

So smarter optimizers were created.

4️⃣ Momentum — *Adds speed*

Think of a ball rolling down a hill.

- Moves faster in the right direction.
- Reduces oscillations.

But: learning rate still constant → not adaptive.

5️⃣ RMSProp — *Adapts learning rate per parameter*

- Gives each parameter its own learning rate.
- Helps when gradients change drastically.

6️⃣ ☆ Adam — The BEST default optimizer

(Adaptive Moment Estimation)

Adam = Momentum + RMSProp
So it:

✔ Learns fast
✔ Adapts learning rate automatically
✔ Works well for almost all deep learning tasks

That's why the correct MCQ answer is **Adam**.

□ Simple Summary Table

Optimizer	Learning Rate	Advantage	Disadvantage
GD	Fixed	Stable	Very slow
SGD	Fixed	Fast	Noisy
Mini-Batch SGD	Fixed	Best tradeoff	Still tuned manually
Momentum	Fixed	Faster learning	LR still fixed
RMSProp	Adaptive	Handles irregular gradients	No momentum
Adam	**Adaptive**	**Best overall**	Sometimes too quick

□ When to use which?

Use Adam → 90% of the time

Best for deep learning (NLP, CV, RL).

Use SGD + Momentum → for very large CNNs

E.g., ImageNet training.

Use RMSProp → for RNN / LSTM

Because gradients are unstable.

Fundamentals of Deep learning

Traditional Programming vs Machine Learning

```
Traditional Programming:
Input + Rules  → Program → Output

Machine Learning:
Input + Output → Learning → Model → Predict Output
```

---

2. ML Training Loop

```
Data → Train Model → Model Learns Patterns → Save Trained Model
```

---

3. Prediction Process

```
New Input → Model → Output Prediction
```

---

4. Spam Detection Comparison

Rule-based:

```
IF "lottery" in title → spam
IF "winner" in title → spam
```

Machine learning:

```
Feed many examples → model learns → classify new emails automatically
```

Linear regression

❤ 1. What is Linear Regression?

Linear Regression is a **supervised machine learning algorithm** used when:

- We have **X (input features)**
- And **Y (numeric output)**

It learns a straight-line (or hyperplane) relationship between X and Y.

Simple Example

Predict:

```
TV Spend → Sales
```

OR

```
Bike Features → Bike Price
```

This is a **regression task** because the output (price or sales) is a **continuous number**.

❤ 2. Why is it called “Linear” Regression?

Because the model tries to fit a line:

Simple Linear Regression (only 1 feature)

```
ŷ = a + bX
```

Where:

- **a = intercept**
- **b = slope (coefficient)**
- **ŷ = predicted value**

Multiple Linear Regression (many features)

```
ŷ = θ0 + θ1X1 + θ2X2 + … + θnXn
```

Where:

- **θ0 = intercept**
- **θ1..θn = learned coefficients**

The model learns these parameter values during training.

❤ 3. How Linear Regression Works

1. You split data into **training** and **test** sets
2. Fit the model on training data
3. Model learns **coefficients + intercept**
4. Predict on test data
5. Evaluate the model using the right metrics
6. Deploy in production

❤ 4. Evaluation Metrics (Beginner-friendly)

These metrics measure how good your regression model is.

✔ A. MAE — Mean Absolute Error

Measures the **average absolute difference** between actual and predicted values.

```
MAE = mean(|y_true − y_pred|)
```

Easy to understand because it uses absolute differences.

Smaller MAE → Better model.

✔ B. MSE — Mean Squared Error

Squares the error before averaging:

```
MSE = mean((y_true − y_pred)^2)
```

- Punishes large errors more strongly
- Used in training many ML/DL models

Smaller MSE → Better.

✔ C. RMSE — Root Mean Squared Error

```
RMSE = √MSE
```

Interpretable in the **same units as the target Y**.

Most commonly used regression metric.

✔ D. R² Score (Coefficient of Determination)

Measures how much of the **variance (information)** Y is explained by X.

```
0   → model explains 0% variance
1   → model explains 100% variance
```

Interpretation:

- 0.91 means **91% of sales are explained by TV + Radio + Newspaper spend**.

Higher R² → Better generalization.

❤ 5. Why We Evaluate Models?

Because training accuracy means nothing.
We care about **prediction performance on unseen test data**.

Evaluation ensures:

- Model generalizes
- Not overfitting
- Not underfitting
- Ready for deployment

Regularization - Linear Regression -

❤ 1. What is Regularization? (Beginner-friendly)

Regularization is a technique to **prevent overfitting** in machine learning models.

When a model learns **too much noise** from training data, it performs poorly on unseen data.
Regularization forces the model to **simplify** itself by **shrinking coefficients**.

Think:

> "Stop the model from memorizing. Make it generalize."

❤ 2. Why Regularization is Needed?

- Sometimes the linear regression model learns **very large coefficients**.
- Larger coefficients → Higher sensitivity to small changes → **Overfitting**
- Regularization controls the coefficient size and keeps the model stable.

❤ 3. Three Types of Regularization

▧ A. L1 Regularization (Lasso)

Adds the absolute value of coefficients as a penalty:

```
Loss = Squared Error + λ * |θ|
```

What it does:

✔ Pushes some coefficients to **exact zero**
✔ Automatically performs **feature selection**
✔ Makes model simpler

Think:

> "Lasso = eliminates features."

▤ B. L2 Regularization (Ridge Regression)

Adds **squared value of coefficients**:

```
Loss = Squared Error + λ * θ²
```

What it does:

✔ Shrinks coefficients but **never makes them zero**
✔ Helps reduce model complexity but keeps all features
✔ More stable than Lasso

Think:

> "Ridge = shrinks but never kills."

▦ C. Elastic Net

Combination of L1 + L2:

```
Loss = Squared Error + α[L1] + β[L2]
```

What it does:

✔ Handles correlated features
✔ Gives more flexibility
✔ Used in real-world enterprise ML pipelines more often

Think:

“Elastic Net = best of both worlds.”

▥ 4. Assumptions of Linear Regression (Beginner-friendly)

To use Linear Regression properly, the data **must** follow certain rules.

✔ A. Linear Relationship

There must be a straight-line relationship between X → Y.

Checked using:

- Pair plots
- Correlation matrices

✔ B. Errors (Residuals) Have Mean = Zero

Residual = (Actual – Predicted)

When you average all residuals:

```
mean(residual) ≈ 0
```

This means the model is unbiased.

✔ C. Errors Follow Normal Distribution

Plot residuals in histogram → should form a **bell curve**.

If not → predictions will be inconsistent.

✔ D. No Multicollinearity

Features should **not strongly correlate with each other**.

Because if:

- X1 = height
- X2 = weight
- BMI depends on both

Then height and weight also depend on each other → model gets confused.

Detected using:

- **VIF (Variance Inflation Factor)**
- If VIF > **4**, feature is problematic

✔ E. Homoscedasticity

Variance of errors should be **constant**.

(Not deeply tested in NVIDIA exam but good awareness.)

Classification Algorithm - Logistic Regression

❤ 1. What is Logistic Regression?

Logistic Regression is a **supervised machine learning algorithm** used when the **output is binary**:

- 0 or 1
- Yes or No
- Spam or Not Spam
- Disease or No Disease

This makes it a **classification algorithm**, NOT a regression model (despite the name).

❤ 2. Why Not Use Linear Regression for Classification?

Linear Regression:

- Predicts continuous values (e.g., 48.6, 102.2)
- Output range is **−∞ to +∞**
- Not suitable for binary classification

Imagine predicting:

```
cancer = 0 or 1
```

Linear Regression may output:

```
-2.4,  1.8,  3.2 → impossible for classification
```

Also:

- Outliers push the line
- Straight line cannot fit 0/1 structured data

❤ 3. Logistic Regression Fixes This Problem

Logistic Regression takes the **linear regression output (Z)**:

```
Z = θ0 + θ1X1 + θ2X2 + … + θnXn
```

Then applies the **Logistic (Sigmoid) Function**:

✔ Logistic (Sigmoid) Function

```
sigmoid(z) = 1 / (1 + e^-z)
```

This transforms any number (−∞ to +∞) into a value between **0 and 1**.

✔ Output Interpretation

- If sigmoid output ≥ 0.5 → predict **1**
- If sigmoid output < 0.5 → predict **0**

You can choose other thresholds (0.4, 0.7) depending on business need.

❤ 4. Why Is It Called "Regression" Then?

Because:

- It uses the **linear regression equation** internally
- But the output is passed through a sigmoid to convert into probability

So mathematically:

```
Linear regression + sigmoid = logistic regression
```

❤ 5. The S-Shaped Curve (S-Curve)

When sigmoid squeezes values between 0 and 1, the shape looks like:

```
    1 |          ________
      |        /
      |      /
      |    /
     0 |___/
```

This is why logistic regression is perfect for **classification**.

❤ 6. How Does Logistic Regression Learn?

It still uses **gradient descent**, same as linear regression.

Key difference:

- Logistic regression uses **log-loss / cross-entropy loss**, not MSE.
- Gradient descent updates θ0, θ1, θ2… to reduce classification error.

Gradient Descent -

❤ 1. What is Gradient Descent?

Gradient Descent is a **method used by machine learning models to learn**.

When a model learns, it tries to:

✔ Reduce the error
✔ Improve accuracy
✔ Adjust parameters (weights) so predictions become correct

Think of Gradient Descent as a **method that helps the model slowly improve its guesses**.

❤ 2. Very Simple Analogy (Beginner Level)

Imagine:

- You are **blindfolded**
- Standing on a **mountain**
- Your goal: **reach the lowest valley**

You cannot see.
So you touch the ground with your foot and check which direction slopes down.

You take **small steps downhill**, and eventually you reach the bottom.

This is **exactly how gradient descent works**:

- Mountain = loss function (error)

- Your steps = model updates
- Lowest point = minimum error
- Blindfold = you don’t know the best values at first

The model keeps taking steps until it finds the lowest error.

❤ 3. Why Do We Need Gradient Descent?

When training models like:

- Logistic regression
- Linear regression
- Neural networks
- Deep learning models

We must find the best values for **parameters/weights** that minimize error.

Gradient Descent is the **tool** that finds those best values.

❤ 4. What is a Gradient?

Gradient = **derivative** = slope = how much a small change in weight changes the loss.

If slope is:

- Positive → model must move **left**
- Negative → model must move **right**

Gradient tells the direction to move to reduce error.

❤ 5. What is the Update Formula?

```
w_new = w_old − learning_rate * gradient
```

Meaning:

- Subtract → move downhill
- Learning rate → step size
- Gradient → direction

❤ 6. What is Learning Rate?

Learning rate (α):

- Controls **how big each step is**
- If too big → model jumps over the minimum (overshoots)
- If too small → model moves very slowly

Think of it as **speed control**.

❤ 7. Why Gradient Descent Works?

Because every update:

✔ Reduces error slightly
✔ Moves toward the global minimum
✔ Finds the best possible model

What is a Classification Evaluation Metric?

When you train a model that predicts categories (like spam vs not spam), you must check how correct it is. These checking methods are called **evaluation metrics**.

❤ 1. Accuracy

Meaning:
Out of all predictions, how many were correct?

Formula:
Correct predictions ÷ Total predictions

Good when:

- Dataset is **balanced** (equal class distribution)

Bad when:

- Dataset is **imbalanced** (e.g., 99% non-fraud, 1% fraud)
  → Even a dumb model can show 99% accuracy.

❤ 2. Confusion Matrix

A 2×2 box showing how the model behaves.

	Predicted 1	**Predicted 0**
Actual 1	True Positive	False Negative
Actual 0	False Positive	True Negative

Meaning:

- **TP** → Actual positive, predicted positive
- **TN** → Actual negative, predicted negative
- **FP** → Type-1 mistake (predict positive but wrong)
- **FN** → Type-2 mistake (predict negative but wrong)

In medical models:

- **FN is VERY dangerous** (missed disease).
- **FP is okay** (false alarm → further tests).

❤ 3. Precision

How reliable are the model's positive predictions?
Out of all predicted positives, how many are truly positive?

Formula:
TP ÷ (TP + FP)

High precision means:
Few false alarms.

Useful in:

- Spam detection
- Fraud alerts
- Intrusion detection

❤ 4. Recall

How many real positive cases did we catch?

Formula:
TP ÷ (TP + FN)

Useful when:

- Missing a positive case is dangerous
  (e.g., cancer detection → MUST catch true cases)

❤ 5. F1 Score

Balances precision and recall.
Useful when:

- Dataset is imbalanced
- Both FP and FN matter

Formula:
2 × (Precision × Recall) ÷ (Precision + Recall)

❤️ 6. ROC Curve & AUC

ROC Curve:

Plots:

- **TPR (Recall)** vs. **FPR**

AUC (Area Under Curve):

- 0.5 → random guessing
- 1.0 → perfect classifier
- Higher is better

Used for:

- Model threshold analysis
- Probability-based models (Logistic Regression, Neural Nets)

🟰 1. Decision Tree — Beginner-Friendly Notes (ELI5 Style)

A **Decision Tree** is a model that asks **a series of questions** to split the data into meaningful groups.
Every question splits the data → until a final decision ('leaf') is reached.

Example flow:

```
Are you hungry?
|
├─ Yes → Do you have ₹1000?
|       ├─ Yes → Go to Novotel
|       └─ No  → Go to South Kitchen
└─ No → Go home
```

□ What a decision tree actually does

- Starts with **all data at the root**
- At each step, picks **the best question** to split data
- Splits continue until:
    - Classes become pure
    - No more useful questions
    - Stopping criteria reached

□ Where Decision Trees Fit in ML

- **Supervised Learning**
- Works for **Classification** (Yes/No, categories)
- Works for **Regression** (numeric predictions)

□ Important Terms (ELI5 style)

Term	Meaning
Root Node	First question (top of tree)
Decision Node	A node that splits further
Leaf Node	Final output (end of path)
Branch / Edge	Answer to a question (Yes/No)
Parent & Child	Upper & lower nodes

▩ 2. NVIDIA-Exam Style Concept Notes

Key Idea

A decision tree learns **simple business rules** to classify or predict an outcome by repeatedly splitting data into “more pure” groups.

□ Core Properties

- Non-linear model
- Handles numeric + categorical data
- Requires no feature scaling
- Sensitive to overfitting
- Uses impurity-based splitting

□ Two Types of Decision Trees

Type	Output	Example
Classification Tree	Class labels (0/1)	Fraud detection
Regression Tree	Continuous value	Predict income

□ Splitting Criteria

1. **Gini Impurity**
2. **Entropy (Information Gain)**

Used to determine the best variable split.

□ Gini Impurity Formula

G=1−∑pi2G = 1 - \sum p_i^2G=1−∑pi2

□ Entropy Formula

Entropy=−∑pilog2piEntropy = -\sum p_i \log_2 p_iEntropy=−∑pilog2pi

---

□ How the Algorithm Works (Corporate Workflow)

1. **Initialize** full dataset at root
2. **Evaluate** all possible features and thresholds
3. **Select** split that maximizes impurity reduction
4. **Recursively partition** into child nodes
5. **Stop** when:
    - Maximum depth reached
    - Leaf node becomes pure
    - No more useful splits

□ Advantages

- Easy to interpret
- Fast to train
- Works without scaling
- Captures non-linear patterns

□ Disadvantages

- High risk of overfitting
- Small changes in data change the tree
- May prefer biased splits on imbalanced data

Overfiting, Underfiting, Kfold Cross Validation -

▥ What is Overfitting? (Simple Explanation)

Overfitting happens when a model learns **too much** from the training data —
including the **noise**, mistakes, and random patterns.

Think of it like:

- Memorizing the answer key instead of actually learning the subject.

Signs:

- Training score → **Very high**
- Test score → **Low**

Why it happens?

- Model is too complex
- Too many branches/leaves (in trees)
- Not enough training data
- Dirty/noisy data

▧ What is Underfitting? (Simple Explanation)

Underfitting happens when the model is **too simple**.
It fails to capture important patterns.

Think of it like:

- Only studying one chapter but exam has 10 chapters.

Signs:

- Training score → Low
- Test score → Low

Why it happens?

- Model too simple
- Not enough features
- Not trained enough

☆ What is a Good Fit?

A good model should:

- Perform well on training data
- Also generalize well on new data
- Not memorize or oversimplify

You want the **sweet spot** between overfitting and underfitting.

▩ K-Fold Cross Validation (Beginner Notes)

K-Fold CV helps us test the model **multiple times** on different subsets.

Simple Steps:

1. Split the data into **K equal parts** (folds).
2. Use **K–1 folds** for training, and **1 fold** for testing.
3. Rotate the test fold each time until every fold has been test once.

4. Average the results → gives a stable score.

Why is it useful?

- Helps prevent overfitting
- Ensures model performs well on different subsets
- Gives reliable accuracy

Example with K=5:

- Split data into 5 parts
- Train on 4 → test on 1
- Repeat 5 times
- Average all 5 scores

SVM

SVM is an algorithm that draws the **best line** between two groups of points.

- That line = **hyperplane**
- Closest points to the line = **support vectors**
- Distance between line and nearest points = **margin**
- SVM tries to **maximize the margin**

Example

If you have blue points on left and green points on right:

```
Blue |-------------| Green
```

SVM finds the line in the middle that *best separates* them.

Linear vs Non-Linear

- **Linear** → straight line separation

- **Non-linear** → curved boundaries

SVM uses **kernels** to bend/shape the boundary:

- RBF (most used)
- Polynomial
- Sigmoid

Esemble Learning - Voting

1. What Problem Does Ensemble Learning Solve?

In real-world ML pipelines, a single model often:

- Overfits,
- Misses patterns,
- Performs inconsistently on complex data.

To mitigate this, enterprise systems combine **multiple diverse models**.
This collective decision-making mechanism reflects *Wisdom of the Crowd*, where aggregated opinions outperform a single expert.

Outcome: Higher accuracy, stronger generalization, reduced variance.

---

2. What Is an Ensemble?

An **ensemble** = a group of predictors (classifiers or regressors) that operate together.

Example ensemble:

- Logistic Regression
- SVM
- Random Forest
- KNN

Each model votes or contributes a score → the ensemble produces the final output.

This aggregated model **almost always** performs better than individual models.

---

3. Types of Ensemble Learning

Your exam will expect you to remember these four:

1. **Voting Classifier** → Aggregates predictions of heterogeneous models
2. **Bagging** → Same model trained on different bootstrap samples
3. **Random Forest** → Bagging + decision tree diversity
4. **Boosting** → Sequential learning (AdaBoost, XGBoost, LightGBM)

Voting Classifier is the simplest and foundational method.

---

4. Voting Classifier – Core Idea

A Voting Classifier trains multiple diverse models and combines their predictions via voting.

There are two types:

A) Hard Voting (Majority Voting)

- Each model predicts a class label.
- The class with **most votes** becomes the final prediction.

Example:

Model	Prediction
Logistic Regression	1
Decision Tree	1

SVM	2
KNN	1

Votes:

- Class 1 → **3 votes**
- Class 2 → 1 vote

Final output → **Class 1**

When to use Hard Voting:

- When individual models do not provide probabilities.
- When probability calibration is unreliable.

B) Soft Voting (Probability Averaging)

- Models output probabilities for each class.
- Ensemble averages probabilities.
- Class with highest probability becomes final output.

Example:

```
P(class=1) = Average of all model probabilities for class1

P(class=2) = Average of all model probabilities for class2

Choose class with higher probability
```

Condition: All models must support `predict_proba()`.
(E.g., SVM requires `probability=True`)

Soft voting usually > hard voting in performance.

---

5. Why Voting Classifier Works

- Different models learn **different patterns**.
- Some are linear learners (Logistic Regression).
- Some are distance-based (KNN).
- Some capture non-linear boundaries (SVM).
- Some capture hierarchical splits (Decision Trees).

Combining them reduces:

- Variance (overfitting),
- Algorithm bias,
- Sensitivity to feature distributions.

Key Principle: Diversity + Aggregation → Higher accuracy.

---

6. Practical Example (Exam-Level Understanding)

Models:

- Logistic Regression → 86%
- Decision Tree → 85%
- SVM → 89%

When combined via **Hard Voting** → ~90.4%
When combined via **Soft Voting** → ~91.2%

The ensemble **outperforms every individual model**.

Why Bagging Exists – The Core Problem

Most models (especially Decision Trees) suffer from:

- **High variance** → small change in data → big change in prediction
- **Overfitting** → memorizing noise, not learning patterns
- **Instability** → small errors compound

Bagging solves this by building **many versions of the model** and then **aggregating their predictions**.

This dramatically improves:

- Stability
- Accuracy
- Generalization

---

2. What Is Bagging? (Bootstrap + Aggregation)

Bagging = **BOOTSTRAP + AGGREGATION**

A) Bootstrap = Sampling With Replacement

Imagine a bowl of 8 marbles.

Normally:

- Friend picks 1 marble → bowl now has 7.

Bootstrap:

- Friend picks 1 marble
- Shows it
- **Puts it back in the bowl**
  → Still 8 marbles
  → Next friend may pick the same marble again

This is **sampling WITH replacement**, so:

- Some data points repeat
- Some data points are missing
- Each sample is slightly different

This creates **diverse training datasets**.

B) Aggregation = Combining Results

After multiple models predict:

- If classification → majority vote
- If regression → average of predictions

---

3. Bagging Classifier – Simple Explanation

Steps:

1. Take your full dataset.
2. Create many bootstrap datasets (sample with replacement).
3. Train the **same model type** on each bootstrap dataset
    - All Decision Trees
    - Or all Logistic Regression
    - etc.
4. Send new data through all models
5. Aggregate predictions
    - Majority vote (classification)
    - Average (regression)

Result:
Reduced overfitting + higher accuracy compared to a single model.

---

4. Why Bagging Works

Bagging reduces **variance** by averaging many unstable models.

Decision Trees are:

- High variance
- Sensitive to small data changes

But when you train MANY trees on many bootstrap samples →
collective decision becomes stable and highly accurate.

---

5. Random Forest = Bagging + Extra Randomness

Random Forest is a **special case** of Bagging.

□ **Difference between Bagging and Random Forest**

Bagging	Random Forest
Uses bootstrapped datasets	Uses bootstrapped datasets
Same model for all trees	Same model (Decision Tree)
No restriction on feature selection	Random subset of FEATURES also used
Trees can look similar	Trees become more diverse

More variance   Less variance, more accuracy

Random Forest =
Bootstrap samples + Random Feature Selection + Many Decision Trees

This extra randomness makes Random Forest:

- More robust
- More generalizable
- Less correlated
- More accurate

---

6. How Random Forest Works (Beginner Version)

For every tree:

1. Create bootstrap sample of rows
2. For each split inside the tree:
    - Randomly pick some columns
    - Only split on those columns
3. Build the tree fully
4. Repeat for many trees
5. Aggregate results by majority vote

---

7. Why Random Forest Is One of the Best ML Models

- Works well on small + large datasets
- Handles missing values

- Resistant to overfitting
- Handles nonlinear relationships
- Works for classification and regression
- Needs minimal tuning
- Produces **Feature Importance** (very exam-important)

---

8. Out-of-Bag (OOB) Score (Exam MUST-KNOW)

When bootstrap sampling is done:

- Around **36% of rows are NOT selected**
- These left-out rows are called **OOB samples**

OOB samples are used as a **test set** for that tree.

OOB Score = internal validation score for bagging/random forest.

You get a FREE built-in cross validation metric.

Enable via:

```
oob_score=True
```

---

9. Exam-Level Comparison Table

Concept	Explanation	Why It Matters
Bootstrap	Sampling with replacement	Creates diverse datasets
Aggregation	Vote or average	Smooths model noise
Bagging	Multiple models(but same type) on	Reduces variance

	bootstrap samples	
Random Forest	Bagging + Random feature selection	Most powerful, stable
OOB Score	Internal testing	Avoids separate validation set

10. Feature Importance (Random Forest)

Random Forest naturally measures:

- How much each feature reduces impurity across all trees
- Gives a % importance score

Example:

```
Petal Length → 44%
Petal Width  → 42%
Sepal Length → 7%
Sepal Width  → 7%
```

Used heavily in:

- Enterprise ML pipelines
- Explainability reports
- Feature selection

---

11. Simple Visual Intuition

Bagging

```
[Tree 1] → 1

[Tree 2] → 0

[Tree 3] → 1

[Tree 4] → 1

[Tree 5] → 0


Votes → Class 1 wins → Final = 1
```

Random Forest adds:

- Random rows
- Random features
  → More diverse trees → Better accuracy

---

12. Beginner Analogy

Imagine you want to decide where to travel.

One person's opinion = unreliable.
Ask 100 people = more stable decision.

Bagging = asking the same kind of people
Random Forest = also ensuring each friend focuses on different topics:

- Food
- Weather
- Budget
- Travel time

More diversity → Better decision.

XGBoost Ensemble -

□ 1. Beginner Explanation – XGBoost (Very Simple Words)

Think of XGBoost as a **super-smart version** of Gradient Boosting.

What problem does it solve?

- When models are trained one after another (boosting), training becomes **slow**, and models can easily **overfit**.
- XGBoost solves these problems with **speed**, **regularization**, and **parallelization**.

What is XGBoost in simple words?

- XGBoost builds **many small decision trees**, one after another.
- Each tree tries to fix the **errors made by the previous tree**.
- All the trees' predictions are added together to get the final output.
- It is extremely fast, accurate, and handles large, complex datasets.

Why do companies love XGBoost?

- Super fast
- Very accurate
- Works well with messy, real-world data
- Supports missing values
- Great for tabular data and competitions (Kaggle champion algorithm)

□ What is Clustering?

Clustering means **grouping similar data points** together.
Example: Grouping customers based on behavior.

□ Types of Clustering

1. **Exclusive Clustering** → each point belongs to *one* cluster
    - Example: **K-Means**
2. **Overlapping Clustering** → points can belong to *multiple* clusters
    - Example: Fuzzy C-Means
3. **Hierarchical Clustering** → builds tree-like structure of clusters

---

□ K-Means in Simple Words

K-Means groups data by:

- Finding **K** centers (centroids)
- Assigning points to the **nearest** center
- Updating centers based on the **mean** of points
- Repeating until stable

Steps (Super Simple)

1. Choose number of clusters (**K**)
2. Place K centroids randomly
3. Assign points to the nearest centroid
4. Move centroids to the **average** of assigned points
5. Repeat until no more changes

□ Why Distance Matters?

K-Means uses **Euclidean distance** to calculate similarity.

□ How to Choose K?

- **Elbow Method** → find where inertia stops dropping fast
- **Silhouette Score** → higher score = better cluster separation

---

□ Where K-Means Is Used?

- Customer segmentation
- Image segmentation
- Document grouping
- Market segmentation
- Behavior clustering

□ What is Hierarchical Clustering?

Hierarchical Clustering is a method of **grouping data into clusters by building a hierarchy (tree structure)**.

It does **not require specifying the number of clusters beforehand**, unlike K-Means.

□ What is a Dendrogram?

A **dendrogram** is a tree-like diagram that shows how data points merge step-by-step into clusters.

Process:

1. Start with each data point as its **own cluster**
2. Find **most similar** clusters
3. Merge them
4. Continue until **all points** become one giant cluster

□ Types of Hierarchical Clustering

1. **Agglomerative Clustering** (Bottom-Up)
    - Start with individual points
    - Merge closest clusters
    - Continue until all points join into one cluster
    - *Most common method*
2. **Divisive Clustering** (Top-Down)
    - Start with all points in one cluster
    - Split repeatedly into smaller clusters

□ Linkage Methods (How similarity is measured)

1. **Single Linkage** → minimum distance between two clusters
2. **Complete Linkage** → maximum distance between two clusters
3. **Average Linkage** → mean pairwise distance
4. **Centroid Linkage** → distance between centroids (cluster centers)

🛠 Implementation Steps

- Create dendrogram using SciPy
- Analyze best separation
- Choose cluster count visually
- Apply Agglomerative Clustering in sklearn
- Inspect cluster labels

□ Where It Is Used?

- Biological taxonomy (gene similarity)
- Customer segmentation
- Document grouping
- Fraud detection
- Image grouping

□ What is Time Series?

A **time series** is data collected over time at regular intervals.
 Examples: rainfall per month, stock prices daily, temperature hourly.

- **Time = independent variable (X)**
- **Value = dependent variable (Y)**
- Written as:
  `y = f(t)` → values depend on time.

□ Components of a Time Series

1. Trend

Long-term upward or downward movement.

- Increasing sales over years → upward trend
- Falling demand over time → downward trend

2. Seasonality

Repeated patterns at fixed time intervals.
 Examples:

- Ice-cream sales ↑ every summer
- Mobile sales ↑ every June

Has **fixed period** (daily, monthly, yearly).

---

3. Cyclical Patterns

Up & down movements **without a fixed period**.

- Business cycles (expansion → recession)
- Real estate price cycles

May span >1 year and timing is irregular.

---

4. Irregularity (Noise)

Random events without pattern.
Examples:

- Natural disasters
- Sudden political events
- Pandemics

Hard to predict.

□ Stationary vs Non-Stationary Data

Stationary Data

- No trend
- No seasonality
- No cyclic pattern
- Mean & variance are constant over time
- Many algorithms REQUIRE this.

Non-Stationary Data

- Has trend/seasonality/cycles/noise
- Mean & variance change over time

☐ How to Test for Stationarity?

1. **ADF Test (Augmented Dickey-Fuller Test)**
   - Default assumption: data is **non-stationary**
   - If p-value < threshold → data **is stationary**
2. **KPSS Test**
   - Opposite assumption of ADF.

☐ How to Convert Non-Stationary → Stationary?

Key methods:

- **Differencing (most important)**
- **Detrending**
- **Transformations** (log, sqrt)

☐ Where Time Series Is Used?

- Forecasting sales
- Weather prediction
- Energy consumption prediction
- Business performance tracking
- Stock price movement

This completes the **beginner-friendly** version.

ARIMA

Definition

ARIMA is a classical time-series forecasting model that incorporates:

- **p** autoregressive terms
- **d** differencing operations
- **q** moving-average terms

Model notation: **ARIMA(p, d, q)**

Time Series Preparation Pipeline

1. Verify Datetime Index

- Convert date column
- Set it as index

2. Check Stationarity

Tools:

- Visual inspection
- **ADF Test** (Augmented Dickey-Fuller)
    - H0 = non-stationary
    - p-value > 0.05 → cannot reject H0 → data is non-stationary

Dataset p-value = **0.99 → strongly non-stationary**

---

3. Convert to Stationary

Primary methods:

- **Differencing (d = 1, 2)**
- **Detrending**
- **Transformations**

Dataset results:

- First difference: borderline stationary
- Second difference: stationary ($p < 0.05$)

---

4. Model Order Identification

Use **PMDARIMA (auto_arima)** to determine optimal p, d, q.

For your dataset:

- p = 4
- d = 1
- q = 3

Model = **ARIMA(4,1,3)**

Core Concepts

Autoregression (AR)

yt=c+ϕ1yt−1+...+ϕpyt−p+ϵty_t = c + \phi_1 y_{t-1} + ... + \phi_p y_{t-p} + \epsilon_tyt=c+ϕ1yt−1+...+ϕpyt−p+ϵt

Integration (I)

Differencing to achieve stationarity.

Moving Average (MA)

Uses past error terms to correct forecasts.

Forecasting

- Short-term → `result.forecast()`
- Long-term → `result.predict(start, end)`

Certification-Relevant Focus Areas

- Identify trend/seasonality
- Diagnose stationarity
- Understand differencing and ARIMA components
- Interpret auto_arima output
- Forecast future values

(Not required for exam: deep math, full implementation coding)

libraries used

TensorFlow & Keras ->

Keras provides the user-friendly interface to build models.

✔ Keras = front-end

✔ TensorFlow = back-end compute engine

Core components used in exam topics:

- `keras.layers` → Dense, LSTM, Conv2D, Normalization
- `keras.models` → Sequential API, Functional API
- `keras.optimizers` → Adam, SGD
- `keras.losses` → CrossEntropy, MSE
- `keras.metrics` → Accuracy, F1, Precision

Keras is a **high-level deep learning API** used to build, train, and evaluate neural networks with **simple, human-friendly Python code**.

from tensorflow import keras

Aspect	TensorFlow	scikit-learn
Type	Deep learning framework	ML utility library
Best for	Neural networks, LLMs, CNNs, Transformers	Classical ML
Models	DNN, CNN, RNN, Transformers	SVM, KNN, RF, LR
GPU support	□ Yes (NVIDIA GPUs)	□ Mostly CPU (cuML use)
Scale	Large-scale, production	Small/medium datasets
Used for LLMs?	□ Yes	□ No

spaCy

is a fast, production-grade NLP library used for preprocessing tasks like tokenization, NER, POS tagging, and text normalization, supporting ML workflows including embeddings, RAG, and vector DB pipelines.

spaCy vs NLTK / HuggingFace

Exam tests differences:

- **spaCy** → fast, production NLP
- **NLTK** → academic / slow
- **HuggingFace** → transformer models

Library	Module/Class	Purpose
pandas	read_csv	Load dataset
pandas	drop, groupby, describe	Data exploration, cleaning
pandas	get_dummies	One-hot encoding
numpy	array ops	Numerical transformations
sklearn.model_selection	train_test_split	Split data
sklearn.linear_model	LinearRegression	Regression model
sklearn.linear_model	LogisticRegression	Classification model
sklearn.linear_model	Lasso, Ridge, ElasticNet	Regularization
sklearn.metrics	mean_squared_error	Regression error metric
sklearn.metrics	mean_absolute_error	Regression error metric
sklearn.metrics	r2_score	Variance explained
sklearn.preprocessing	LabelEncoder	Encode categories

sklearn.feature_selection	RFE	Feature selection
imblearn.over_sampling	SMOTE	Class balancing
statsmodels	variance_inflation_factor	Multicollinearity check
seaborn	pairplot, distplot, barplot	EDA visualizations
matplotlib	pyplot	Charting

Classification evaluation metrics

1. scikit-learn Metrics

✔ Import

```
from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    ConfusionMatrixDisplay,
    precision_score,
    recall_score,
    f1_score,
    roc_auc_score,
    roc_curve
)
```

✔ Methods & Use Cases

Method	Purpose	Use Case
accuracy_score	Basic correctness measure	Balanced datasets
confusion_matrix	Build confusion matrix	Error-type analysis
ConfusionMatrixDisplay	Plot matrix	Visual dashboards
precision_score	Precision	Reduce false alarms
recall_score	Recall	Reduce false negatives
f1_score	Harmonic mean	Imbalanced datasets

roc_curve	Compute TPR, FPR across thresholds	Plot ROC
roc_auc_score	Compute AUC	Class separability

2. Matplotlib for Visualization

✔ Import

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

✔ Methods

- `plt.plot()`
- `plt.show()`

Use case:
Plotting ROC curves and evaluation charts.

---

3. NumPy (optional usage)

✔ Import

```
import numpy as np
```

Used for array handling when plotting ROC/TPR/FPR.

Decision Tree -

1 scikit-learn: Decision Tree

✔ Imports

```
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.tree import plot_tree
```

✔ Core Methods & Use Cases

Method	Purpose	Use Case
`fit(X, y)`	Train decision tree	Classification/regression
`predict(X)`	Generate predictions	Model inference
`predict_proba(X)`	Get probability scores	ROC, risk scoring
`plot_tree(model)`	Visualize tree	Explainability dashboards
Parameters: `criterion`	Choose Gini or Entropy	Tune impurity reduction
Parameters: `max_depth`	Control tree height	Prevent overfitting
Parameters: `min_samples_split`	Minimum samples to split	Regularization
Parameters: `min_samples_leaf`	Minimum samples in leaf	Smooth decision boundaries

2️⃣ Graphviz (Optional but used by mentor)

✔ Import

```
from graphviz import Source
```

✔ Use Case

- High-resolution decision tree visualization
- Exporting tree diagrams to PDF/SVG
- Used in enterprise model documentation