⭐ 1. What is Logistic Regression?

Logistic Regression is a **classification algorithm**, NOT a regression model (despite the name).

✔ Predicts **categories** → Example: Spam / Not Spam
✔ Outputs **probabilities** between **0 and 1**
✔ Converts probability into class label (0 or 1) using a **threshold**

⭐ 2. Why do we call it Regression?

Because it uses a linear model:

$z=w1x1+w2x2+...+b$

But applies the **sigmoid function** on top to convert z → probability.

⭐ 3. Sigmoid Function

The heart of Logistic Regression.

$p=1+e−z1$

✔ Output always lies between **0 and 1**
✔ Used to interpret model output as “probability of class 1”

Example:

```
z = 2.0 → sigmoid = 0.88 → 88% chance of class 1
z = -1.0 → sigmoid = 0.27 → 27% chance of class 1
```

⭐ 4. Probability → Class Conversion (Threshold)

The model **does NOT directly predict 0 or 1**.

It predicts probability **p**.
We use a threshold (default = **0.5**) to convert:

```
if p ≥ 0.5 → class = 1
if p < 0.5 → class = 0
```

Example:

```
p=0.84 → 1
p=0.33 → 0
p=0.51 → 1
```

⭐ 5. Why is Threshold important?

You can modify threshold based on domain:

✔ Medical diagnosis → need high recall → threshold low (0.3)
✔ Fraud detection → avoid false alarms → threshold high (0.7)
✔ Normal binary classification → threshold = 0.5

ROC curve helps identify the **best threshold**.

⭐ 6. Loss Function Used: Log Loss (Cross-Entropy)

Logistic Regression tries to minimize:

$$\text{Loss} = -[y\log(p) + (1-y)\log(1-p)]$$

✔ Punishes confident wrong predictions
✔ Improves probability calibration

⭐ 7. Regularization in Logistic Regression

To prevent overfitting:

✔ L2 (Ridge Regularization)

- Shrinks weights
- Does NOT make coefficients zero
- DEFAULT penalty
- Very stable
- Works with most solvers

✔ L1 (Lasso Regularization)

- Makes some weights exactly **zero**
- Performs **feature selection**
- Works with solvers: `liblinear`, `saga`

✔ ElasticNet (L1 + L2 combination)

- Balances sparsity and stability
- Works only with solver: `saga`

⭐ 8. Solvers (Optimization Algorithms)

Solver	Supports	Use Case
lbfgs	L2	Best default, fast
newton-cg	L2	Multiclass problems
liblinear	L1, L2	Small datasets
sag	L2	Very large datasets
saga	L1, L2, Elastic	Best for large sparse datasets

⭐ 9. Confusion Matrix

Used to compute TPR, FPR, Accuracy, etc.

	Pred 1	**Pred 0**
Actual 1	TP	FN
Actual 0	FP	TN

Definitions:

- **TP:** Correct positive
- **TN:** Correct negative
- **FP:** Wrong positive (false alarm)
- **FN:** Missed positive

⭐ 10. Important Metrics

==========
LOGISTIC REGRESSION METRICS
==========

1. Accuracy

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Accuracy tells how many total predictions were correct.
Fails on imbalanced datasets.

2. Precision

Precision = TP / (TP + FP)

Meaning:
When the model predicts Positive, how often is it correct?
Avoids False Positives.

3. Recall (TPR) - True Positive Rate

Recall = TP / (TP + FN)

Meaning:
How many actual positives did the model catch?
Avoids False Negatives.

4. FPR - False Positive Rate

FPR = FP / (FP + TN)

Meaning:
Out of all actual negatives, how many were wrongly predicted as positive?

5. Confusion Matrix

	Predicted 1	Predicted 0
Actual 1	TP	FN
Actual 0	FP	TN

Where:
TP = True Positive
TN = True Negative
FP = False Positive
FN = False Negative

6. F1 Score

F1 = 2 * (Precision * Recall) / (Precision + Recall)

Balances Precision and Recall.
Useful for imbalanced datasets.

7. ROC Curve

ROC curve plots:
  X-axis → FPR
  Y-axis → TPR (Recall)

Shows performance at different probability thresholds.

8. AUC Score

AUC = Area Under ROC Curve

AUC = 1 → perfect model
AUC = 0.5 → random guessing
AUC < 0.5 → worse than random

==========
END OF LOGISTIC METRICS NOTES
==========

⭐ 11. ROC Curve (Receiver Operating Characteristic

⭐ 11. ROC Curve (Receiver Operating Characteristic)

ROC curve plots:

- **X-axis:** FPR
- **Y-axis:** TPR

Purpose:
✔ Shows model performance for **all thresholds**
✔ Helps visualize separability power

HOW ROC IS MADE?

For every threshold:

1. Convert probability → predicted class
2. Compute TP, FP, FN, TN
3. Compute TPR & FPR
4. Plot point (FPR, TPR)

ROC curve = all such points.

⭐ 12. AUC Score (Area Under Curve)

Measures total area under ROC curve.

AUC Score	Model Quality
1.0	Perfect
0.9–1.0	Excellent
0.8–0.9	Good

0.7–0.8	Okay
0.5	Random guessing
< 0.5	Worse than random

AUC = probability model ranks positive higher than negative.

⭐ 13. Understanding Thresholds on ROC Graph

From code:

```
for xyz in zip(model_fpr, model_tpr, thresholds):
    ax.annotate(str(round(xyz[2],2)), xy=(xyz[0], xyz[1]))
```

Meaning:

- `xyz[0]` = FPR
- `xyz[1]` = TPR
- `xyz[2]` = threshold

This writes the **threshold value** at each ROC point.

⭐ 14. Why ROC-AUC is Better Than Accuracy

Accuracy fails when data is **imbalanced**.

Example:

- 99% class 0
- 1% class 1

Model predicts all zeros → 99% accuracy but **useless**.

ROC-AUC measures **ranking quality**, not class labels.

⭐ 15. Key Advantages of Logistic Regression

✔ Easy to understand
✔ Fast
✔ Works on large datasets
✔ Outputs probability (very useful)
✔ Regularization available
✔ Performs well if classes are linearly separable

⭐ 16. Limitations

❌ Cannot capture complex non-linear relationships
❌ Performance drops if features not scaled
❌ Poor on heavily imbalanced data (without AUC/threshold tuning)

⭐ 17. Example Summary (Simple)

Suppose probabilities:

```
[0.95, 0.80, 0.40, 0.30]
```

Threshold = 0.8

Predictions:

```
[1,1,0,0]
```

Confusion matrix → compute TPR, FPR → plot ROC point.

⭐ 18. When to use Logistic Regression?

Use when:

✔ Binary classification
✔ Features are linearly separable
✔ You need interpretability
✔ You want probability output
✔ Dataset is not too huge or too complex

⭐ 19. Important Notes About Thresholds

- Threshold = 0.5 is **NOT always best**
- Use ROC-AUC to choose best threshold
- Use precision–recall curve for imbalance
- Domain-specific thresholds improve results

⭐ 20. Best Practice Pipeline

1. Train Logistic Regression with scaling
2. Tune hyperparameters (solver, C, penalty)
3. Plot ROC & compute AUC
4. Select best threshold
5. Evaluate with confusion matrix
6. Deploy model with chosen threshold

📘 ROC Curve – Notes

1. Logistic Regression gives probabilities

Example:
`[0.95, 0.80, 0.40, 0.30]`

These are NOT classes — they are probabilities of being class **1**.

---

2. Converting Probability → Class (0/1)

To make predictions:

```
if probability ≥ threshold → predict 1
else → predict 0
```

Default threshold = **0.5**
But ROC does **not** use only one threshold — it tries many.

---

3. How ROC Curve Works (Step-by-Step)

ROC tests **all possible thresholds**, for example:

```
[1.0, 0.95, 0.80, 0.40, 0.30, 0.0]
```

For each threshold:

1. Convert probabilities → 0/1 predictions
2. Compute:
    - **TPR (Recall)** = TP / (TP + FN)
    - **FPR** = FP / (FP + TN)
3. Plot the point → (FPR, TPR)

All points combined create the **ROC Curve**.

---

4. Why We Use ROC Curve?

- Works across **all thresholds**, not just 0.5
- Best for **imbalanced datasets**
- Shows model's ability to **separate** classes
- Helps compare classifiers clearly

---

5. AUC Score (Area Under Curve)

- AUC = **1** → Perfect model
- AUC = **0.5** → Random guessing
- AUC < **0.5** → Worse than random

Bigger AUC = Better model.

---

6. ROC Graph Axes

- **X-axis** = FPR (False Positive Rate)
- **Y-axis** = TPR (True Positive Rate / Recall)

---

7. Key Idea (One Line)

ROC changes the threshold again and again, calculates TPR & FPR, plots all points → this becomes the ROC curve.

📘 Hyperparameter Tuning Notes

GridSearchCV vs RandomizedSearchCV

⭐ 1. Why Hyperparameter Tuning?

Machine learning models have hyperparameters (like C, penalty, solver, max_depth, n_estimators)
that control learning.

Bad hyperparameters → poor accuracy
Good hyperparameters → high accuracy

Hyperparameter tuning finds the best combination automatically.

🔷 GRIDSEARCHCV

✔ Definition

GridSearchCV tests every possible combination of hyperparameters.

It performs:

- **Exhaustive search**
- **Cross-validation for each combination**
- **Picks the combination with the highest score**

✔ How GridSearchCV Works (Internally)

Given:

```
C = [0.1, 1, 10]
penalty = ['l1', 'l2']
```

Total combinations = 3 × 2 = 6

For each combination:

1. **Perform K-fold cross-validation**
2. **Compute mean accuracy**
3. **Store the score**
4. **Move to next combination**

Finally selects best_params_, best_score_, best_estimator_

✔ Advantages

- **Guaranteed to find the best hyperparameter set**
- **Simple & exhaustive**
- **Works well when parameter search space is small**

✔ Disadvantages

- **Slow**
- **Expensive for large parameter grids**
- **Not good for high-dimensional spaces**

🔷 RANDOMIZEDSEARCHCV

✔ Definition

RandomizedSearchCV tests only a random subset of the hyperparameter combinations.

Instead of trying all combinations, it tries n_iter random ones.

✔ How RandomizedSearchCV Works (Internally)

You tell it:

```
n_iter = 20
```

It will:

1. **Randomly sample 20 combinations**

2. **For each combination → run cross-validation**
3. **Pick the best one among the sampled options**

It does not try all combinations.

✔ Advantages

- **Much faster**
- **Works well with large hyperparameter spaces**
- **Can sample from distributions (uniform, log-uniform, etc.)**
- **Good when model training is expensive (e.g., XGBoost, Random Forest, Neural Nets)**

✔ Disadvantages

- **Does not guarantee the absolute best combination**
- **Performance depends on number of iterations**

🔷 GRIDSEARCHCV vs RANDOMIZEDSEARCHCV (Summary Table)

Feature	GridSearchCV	RandomizedSearchCV
Tries all combinations	✔ Yes	❌ No
Speed	❌ Slow	✔ Fast

Best params guaranteed	**✔ Yes**	**❌ Not guaranteed**
Good for small search space	**✔**	**❌**
Good for large search space	**❌**	**✔**
Supports distributions	**❌**	**✔**
Typical use case	**Linear/Logistic Regression, SVM small grids**	**Random Forest, XGBoost, Deep Learning**

🔷 When To Use Which?

✔ Use GridSearchCV when:

- **You have few hyperparameters**
- **Search space is small**
- **You need exact optimal values**

✔ Use RandomizedSearchCV when:

- **Hyperparameter space is huge**
- **Model training is slow**
- **You want results fast**
- **You're exploring parameter ranges**

🔷 Best Practice (Use Both Together)

1. **RandomizedSearchCV first**
   → Quickly finds good range of hyperparameters

2. **GridSearchCV next**
   → Fine-tunes within that good range

Best of both worlds:
✔ Speed
✔ Accuracy