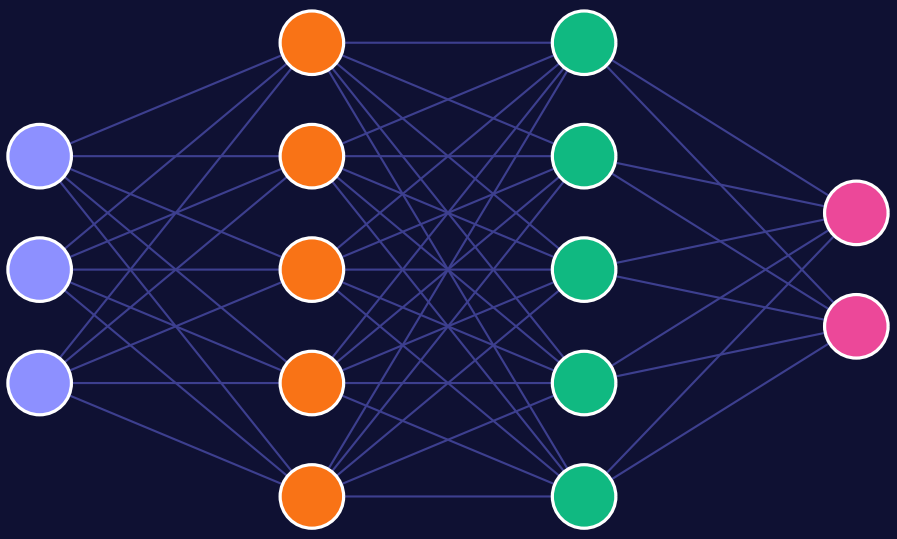
100 Machine Learning Interview Questions

With answers and tested Python code, freshers to experienced



Code and Debug

codeanddebug.in/blog | YouTube: @codeanddebug

Contents

How to use these notes 5

Machine learning interview questions for freshers 6

1. What is machine learning? . . . 6
2. What is the difference between AI, machine learning, deep learning and generative AI? . . . 6
3. What are the main types of machine learning? . . . 6
4. What is the difference between supervised and unsupervised learning? . . . 7
5. What is the difference between classification and regression? . . . 7
6. What are features, labels and training data? . . . 7
7. Why do we split data into training, validation and test sets? . . . 7
8. What are overfitting and underfitting? . . . 8
9. What is the bias-variance tradeoff? . . . 8
10. How can you prevent overfitting? . . . 9
11. What is feature engineering? . . . 9
12. How do you handle missing values? . . . 9
13. What is the difference between normalisation and standardisation? . . . 10
14. How do you encode categorical variables? . . . 10
15. What is data leakage? . . . 11
16. What are outliers and how do you handle them? . . . 12
17. What is the curse of dimensionality? . . . 12
18. How does linear regression work, and what are its assumptions? . . . 12
19. What is the difference between MAE, MSE and RMSE? . . . 13
20. What is R^2 (the coefficient of determination)? . . . 13
21. How does logistic regression work, and why not use linear regression for classification? . . . 13
22. What is the sigmoid function? . . . 14
23. How does the KNN algorithm work, and how do you choose K? . . . 14
24. What is Naive Bayes, and why is it called naive? . . . 15
25. How does a decision tree work? . . . 15
26. What are Gini impurity and entropy? . . . 15
27. What is a support vector machine (SVM)? . . . 16
28. What is a confusion matrix? . . . 16
29. Why can accuracy be misleading? . . . 16
30. What is the difference between precision and recall? . . . 17
31. What is the F1 score? . . . 17
32. What are the ROC curve and AUC? . . . 17
33. What is cross-validation? . . . 18
34. What is clustering, and how does k-means work? . . . 18
35. How do you choose the number of clusters K in k-means? . . . 18
36. What is PCA (principal component analysis)? . . . 19
37. What is gradient descent? . . . 19
38. What is the difference between a parameter and a hyperparameter? . . . 20

39. What is a neural network? . . . 20
40. What is an activation function, and what are the common ones? . . . 20

Intermediate machine learning interview questions 22

41. What is the difference between batch, stochastic and mini-batch gradient descent? . . . 22
42. What is the difference between L1 and L2 regularisation? . . . 22
43. What is Elastic Net, and when would you use it? . . . 23
44. How do you tune hyperparameters? . . . 23
45. What is stratified k-fold cross-validation, and when do you need it? . . . 23
46. How do you handle imbalanced data? . . . 24
47. When should you use a precision-recall curve instead of a ROC curve? . . . 24
48. How do you choose the classification threshold? . . . 25
49. What is log loss (cross-entropy)? . . . 25
50. How do models handle multiclass classification, and what are macro, micro and weighted averages? . . . 25
51. What is the difference between bagging and boosting? . . . 26
52. How does a random forest work? . . . 26
53. How does gradient boosting work, and what are its key hyperparameters? . . . 27
54. How does AdaBoost work? . . . 27
55. How do you measure feature importance? . . . 27
56. What are the main feature selection methods? . . . 28
57. What is multicollinearity, and why does it matter? . . . 28
58. What is pruning in decision trees? . . . 28
59. What is the kernel trick, and what do C and gamma do in an SVM? . . . 28
60. When would you use DBSCAN instead of k-means? . . . 29
61. How does hierarchical clustering work? . . . 29
62. How does PCA work, and how do you choose the number of components? . . . 29
63. How are t-SNE and UMAP different from PCA? . . . 29
64. What is a scikit-learn Pipeline, and why should you use one? . . . 29
65. How do you validate a model on time series data? . . . 30
66. How do you check whether a regression model is good beyond its score? . . . 30
67. What is the difference between generative and discriminative models? . . . 31
68. What is maximum likelihood estimation, and how does it relate to loss functions? . . . 31
69. How does backpropagation work? . . . 31
70. What are vanishing and exploding gradients? . . . 32
71. What are dropout and batch normalisation? . . . 32
72. What is a convolutional neural network (CNN)? . . . 33
73. What are RNNs and LSTMs? . . . 33
74. What are word embeddings? . . . 33
75. What is transfer learning? . . . 33

Interview questions for experienced candidates 34

76. How do you approach an end-to-end machine learning project? . . . 34
77. Your model scored well offline but performs badly in production. What could be wrong? . . . 34

78. What is the difference between data drift and concept drift, and how do you monitor them? 34
79. How do you deploy a machine learning model? 35
80. How do you A/B test a new model, and why can offline and online metrics disagree? 35
81. How do you train a model when the data does not fit in memory? 35
82. How do recommendation systems work? 35
83. What is probability calibration, and why does it matter? 36
84. Explain the bias-variance decomposition. 36
85. How do the learning rate and early stopping work together in gradient boosting? 37
86. What makes XGBoost different from classic gradient boosting? 37
87. How do LightGBM, XGBoost and CatBoost compare? 37
88. How do SHAP and LIME explain a model's predictions? 38
89. How do you find and reduce bias (unfairness) in a model? 38
90. How do you choose the right metric for a business problem? 38
91. What is a transformer, and why did it replace RNNs? 39
92. How does self-attention work? 39
93. How are large language models trained? 40
94. When would you use prompt engineering, RAG or fine-tuning? 40
95. How does a RAG pipeline work, and what are embeddings and vector databases for? 40
96. What is hallucination, and how do you evaluate LLM outputs? 41
97. What are tokens, temperature and top-p in LLMs? 41
98. What are LoRA and parameter-efficient fine-tuning? 41
99. How do you handle categorical features with thousands of categories? 42
100. How would you design a fraud detection system for card transactions? 42

How to use these notes

- The 100 questions are in three levels: questions 1 to 40 for freshers, 41 to 75 intermediate, 76 to 100 for experienced candidates.
- Every code example was run with Python and scikit-learn; the output shown is the real output.
- Answer each question out loud before reading the answer. Interviewers care about clear explanations with examples.
- The interactive tools, quiz and latest version of these notes are on the blog: codeanddebug.in/blog/machine-learning-interview-questions

Machine learning interview questions for freshers

1. What is machine learning?

Freshers · Basics

Machine learning is a way of building software that learns patterns from data instead of following rules a programmer wrote by hand. You show the model many examples, it adjusts its internal numbers (parameters) to fit them, and then it makes predictions on new data it has never seen.

A spam filter is the classic example. Writing rules for every kind of spam is impossible, but if you give a model thousands of emails labelled "spam" or "not spam", it learns which words and patterns matter. Tom Mitchell's textbook definition is often quoted in interviews: a program learns from experience E on task T, measured by performance P, if its performance on T improves with E.

2. What is the difference between AI, machine learning, deep learning and generative AI?

Freshers · Basics

Term	What it means	Example
Artificial intelligence (AI)	The broad goal of making machines do tasks that need human intelligence	Chess engines, chatbots, self-driving cars
Machine learning (ML)	A part of AI where systems learn patterns from data	Spam filters, loan approval, price prediction
Deep learning (DL)	A part of ML that uses neural networks with many layers	Face recognition, speech to text
Generative AI	Deep learning models that create new text, images, audio or code	ChatGPT, Claude, image generators

Each one sits inside the one before it: all deep learning is machine learning, and all machine learning is AI, but not the other way round. Our AI and ML roadmap shows how they connect as a learning path.

3. What are the main types of machine learning?

Freshers · Basics

- **Supervised learning:** the training data has the correct answers (labels). The model learns to map inputs to outputs. Examples: predicting house prices, detecting spam.
- **Unsupervised learning:** no labels. The model finds structure on its own, such as groups of similar customers or a compressed version of the data.
- **Reinforcement learning:** an agent takes actions in an environment and learns from rewards and penalties. Examples: game-playing agents, robot control.

Two in-between types also come up: **semi-supervised learning** uses a few labelled examples plus many unlabelled ones, and **self-supervised learning** creates labels from the data itself, such as predicting the next word. Large language models are trained mostly with self-supervised learning.

4. What is the difference between supervised and unsupervised learning?

Freshers · Basics

Point	Supervised	Unsupervised
Labels	Needed for every training example	Not needed
Goal	Predict a known target	Find hidden structure or groups
Typical tasks	Classification, regression	Clustering, dimensionality reduction, anomaly detection
Algorithms	Linear and logistic regression, decision trees, SVM, random forest	K-means, DBSCAN, hierarchical clustering, PCA
How you evaluate	Compare predictions with true labels	Harder; use measures like silhouette score or business judgement

A simple way to remember it: if you know what the right answer looks like for past data, it is supervised.

5. What is the difference between classification and regression?

Freshers · Basics

Both are supervised learning. **Classification** predicts a category: spam or not spam, which digit is in an image, whether a loan will default. **Regression** predicts a continuous number: a house price, tomorrow's temperature, a delivery time.

The difference also changes how you measure success. Classification uses accuracy, precision, recall and F1; regression uses MAE, RMSE and R^2. Note that logistic regression is a classification algorithm despite its name, because it outputs a probability that is turned into a class.

6. What are features, labels and training data?

Freshers · Basics

Features (also called inputs, predictors or independent variables) are the pieces of information the model uses, such as a flat's size, location and floor. The **label** (target or dependent variable) is what you want to predict, such as its rent. **Training data** is the set of past examples that have both features and labels.

In code, features are usually a 2D array or DataFrame called `X`, with one row per example and one column per feature, and labels are a 1D array called `y`.

7. Why do we split data into training, validation and test sets?

Freshers · Basics

A model must be judged on data it has not seen, otherwise a model that simply memorised the training data would look perfect. So the data is split:

- **Training set** (often 60 to 80%): the model learns from it.
- **Validation set**: used to compare models and tune hyperparameters. Cross-validation can replace a fixed validation set.
- **Test set** (often 10 to 20%): touched only once, at the very end, to estimate real-world performance.

PYTHON

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42)

print("Train:", X_train.shape, "| Test:", X_test.shape)
print("Class counts in test:", [int((y_test == c).sum()) for c in range(3)])
```

OUTPUT

```
Train: (120, 4) | Test: (30, 4)
Class counts in test: [10, 10, 10]
```

Using `stratify=y` keeps the class proportions the same in both parts, which matters for small or imbalanced datasets. If you keep checking the test set while tuning, it stops being an honest estimate.

8. What are overfitting and underfitting?

Freshers · Basics

Overfitting means the model has learned the training data too closely, including its noise. It scores very well on training data and badly on new data. **Underfitting** means the model is too simple to capture the real pattern, so it does badly on both.

Here a polynomial is fitted to 15 noisy points from a sine curve. Degree 1 (a straight line) underfits, degree 3 is about right, and degree 12 overfits: its training error is the lowest, but its test error explodes.

PYTHON

```python
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures

rng = np.random.default_rng(1)
x = rng.uniform(0, 1, 15).reshape(-1, 1)
y = np.sin(2 * np.pi * x).ravel() + rng.normal(0, 0.2, 15)
x_test = rng.uniform(0, 1, 200).reshape(-1, 1)
y_test = np.sin(2 * np.pi * x_test).ravel() + rng.normal(0, 0.2, 200)

for degree in [1, 3, 12]:
    model = make_pipeline(PolynomialFeatures(degree), LinearRegression()).fit(x, y)
    print(f"degree {degree:2d}: train MSE={mean_squared_error(y, model.predict(x)):.3f}  "
          f"test MSE={mean_squared_error(y_test, model.predict(x_test)):.3f}")
```

OUTPUT

```
degree  1: train MSE=0.298  test MSE=0.251
degree  3: train MSE=0.016  test MSE=0.060
degree 12: train MSE=0.007  test MSE=3.418
```

The bias-variance slider further up this page lets you see the same thing visually.

9. What is the bias-variance tradeoff?

Freshers · Basics

Bias is error from wrong or overly simple assumptions: a straight line can never fit a curve, however much data you give it. **Variance** is error from being too sensitive to the particular training data: a very flexible model changes a lot if you train it on a slightly different sample.

Simple models have high bias and low variance (they underfit). Complex models have low bias and high variance (they overfit). Expected test error is roughly bias² + variance + irreducible noise, so the best model sits in the middle, where the total is lowest. More data lowers variance, which is why complex models need a lot of it.

10. How can you prevent overfitting?

Freshers · Basics

- Get more training data, or use data augmentation for images and text.
- Use a simpler model or fewer features.
- Add regularisation (L1, L2, dropout for neural networks).
- Use cross-validation to choose hyperparameters honestly.
- Limit model complexity: `max_depth` for trees, fewer layers or units for neural networks.
- Stop training early when the validation score stops improving.
- Use ensembles such as random forests, which average out the variance of single models.

In an interview, mention that you first *detect* overfitting by comparing training and validation scores. A large gap is the signal.

11. What is feature engineering?

Freshers · Data & features

Feature engineering is creating, transforming or selecting input features so the model can learn more easily. Good features often matter more than the choice of algorithm.

- Extracting parts of a date: day of week, month, is_weekend, is_festival_season.
- Ratios and differences: price per square foot, debt-to-income ratio.
- Aggregates: a customer's average order value over the last 30 days.
- Text features: word counts, TF-IDF, embeddings.
- Encoding categories, scaling numbers and handling missing values (the next few questions).

12. How do you handle missing values?

Freshers · Data & features

First find out *why* values are missing, because that decides the fix. Then choose:

- **Drop** rows or columns, if very few rows are affected or a column is mostly empty.
- **Impute** with the mean or median (numbers) or the most frequent value (categories).
- **Model-based imputation**, such as KNN imputation or iterative imputation.
- **Add a "was missing" indicator** column, because missingness itself can carry information.
- Use models that handle missing values natively, such as gradient boosting libraries.

PYTHON

```
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer

df = pd.DataFrame({"age": [25, np.nan, 40, 35, np.nan],
                   "city": ["Surat", "Pune", np.nan, "Pune", "Pune"]})

num = SimpleImputer(strategy="median").fit_transform(df[["age"]])
cat = SimpleImputer(strategy="most_frequent").fit_transform(df[["city"]])
print("age: ", num.ravel())
print("city:", cat.ravel())
```

OUTPUT

```
age:  [25. 35. 40. 35. 35.]
city: ['Surat' 'Pune' 'Pune' 'Pune' 'Pune']
```

Fit the imputer on the training data only and reuse it on the test data, otherwise information leaks from the test set. A scikit-learn Pipeline (question 64) does this for you.

13. What is the difference between normalisation and standardisation?

Freshers · Data & features

Normalisation (min-max scaling) squeezes values into a fixed range, usually 0 to 1: `(x − min) / (max − min)`. **Standardisation** (z-score) rescales values to mean 0 and standard deviation 1: `(x − mean) / std`.

PYTHON

```
import numpy as np
from sklearn.preprocessing import MinMaxScaler, StandardScaler

salary = np.array([[20000], [35000], [50000], [120000]])

print("Min-max:     ", MinMaxScaler().fit_transform(salary).ravel().round(3))
print("Standardised:", StandardScaler().fit_transform(salary).ravel().round(3))
```

OUTPUT

```
Min-max:      [0.   0.15 0.3  1.  ]
Standardised: [-0.946 -0.555 -0.163  1.664]
```

Standardisation is the usual default and copes better with outliers; one extreme salary squashes all other min-max values towards 0. Min-max is handy when you need a bounded range, such as pixel values. Scaling matters for distance-based and gradient-based models (KNN, k-means, SVM, linear models, neural networks) but not for tree-based models.

14. How do you encode categorical variables?

Freshers · Data & features

Models need numbers, so text categories must be encoded:

- **One-hot encoding:** one 0/1 column per category. Safe for categories with no order, like city.
- **Ordinal (label) encoding:** one integer per category. Use it only when the categories have a real order, like S, M, L, or with tree models.
- **Target encoding:** replace each category with the average target for it. Useful for many categories, but it must be done inside cross-validation to avoid leakage.

PYTHON

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder

df = pd.DataFrame({"city": ["Surat", "Pune", "Delhi", "Pune"],
                   "size": ["S", "L", "M", "S"]})

onehot = OneHotEncoder(sparse_output=False).fit(df[["city"]])
print(onehot.get_feature_names_out())
print(onehot.transform(df[["city"]]))

ordinal = OrdinalEncoder(categories=[["S", "M", "L"]])
print("size:", ordinal.fit_transform(df[["size"]]).ravel())
```

OUTPUT

```
['city_Delhi' 'city_Pune' 'city_Surat']
[[0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]
 [0. 1. 0.]]
size: [0. 2. 1. 0.]
```

Giving cities the numbers 0, 1 and 2 with a linear model would wrongly tell it that Delhi < Pune < Surat. That is why one-hot encoding is the default for unordered categories.

15. What is data leakage?

Freshers · Data & features

Data leakage happens when information that would not be available at prediction time sneaks into training. The model looks great in testing and fails in real use. Common causes:

- A feature that is created after the outcome, such as "refund issued" when predicting fraud.
- Scaling, imputing or selecting features on the whole dataset before splitting.
- Duplicate or near-duplicate rows in both train and test.
- Random splits on time series, so the model trains on the future.

Here the labels are pure coin flips, so no model can beat 50%. Selecting features on all the data first gives a fake 85%:

PYTHON

```python
import numpy as np
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import make_pipeline

rng = np.random.default_rng(0)
X = rng.normal(size=(100, 5000))        # pure noise features
y = rng.integers(0, 2, 100)             # random labels: nothing to learn

# WRONG: pick the "best" features using all the data, then cross-validate
X_sel = SelectKBest(f_classif, k=20).fit_transform(X, y)
leaky = cross_val_score(LogisticRegression(max_iter=1000), X_sel, y, cv=5).mean()

# RIGHT: feature selection happens inside each training fold
pipe = make_pipeline(SelectKBest(f_classif, k=20), LogisticRegression(max_iter=1000))
honest = cross_val_score(pipe, X, y, cv=5).mean()

print(f"Leaky accuracy:  {leaky:.2f}")
print(f"Honest accuracy: {honest:.2f}")
```

OUTPUT

```
Leaky accuracy:  0.85
Honest accuracy: 0.48
```

16. What are outliers and how do you handle them?

Freshers · Data & features

Outliers are values far away from the rest of the data, such as a salary of ₹5 crore in a dataset of students' first jobs. Detect them with box plots, the IQR rule (below Q1 − 1.5 × IQR or above Q3 + 1.5 × IQR), z-scores above about 3, or models like Isolation Forest.

Then decide whether each one is an error or a real but rare value. Fix or remove errors. Keep real ones but reduce their effect: cap them (winsorising), apply a log transform, use robust models (trees, median-based methods) or robust losses (MAE, Huber). Never drop outliers blindly; in fraud detection the outliers are often the whole point.

17. What is the curse of dimensionality?

Freshers · Data & features

As the number of features grows, the space the data lives in grows so fast that the data becomes very sparse. You would need exponentially more data to cover it. Distances also become less meaningful, because in high dimensions the nearest and furthest points are almost the same distance away.

This hurts distance-based models like KNN and k-means the most, and raises the risk of overfitting for everything. Fixes are feature selection, dimensionality reduction (PCA), regularisation, or simply collecting more data.

18. How does linear regression work, and what are its assumptions?

Freshers · Regression

Linear regression predicts a number as a weighted sum of the features plus a constant: $\hat{y} = w_1x_1 + w_2x_2 + \ldots + b$. Training finds the weights that minimise the mean squared error between predictions and actual values, either with a closed-form solution (ordinary least squares) or with gradient descent.

The classic assumptions are: a linear relationship between features and target, independent errors, constant error variance (homoscedasticity), normally distributed errors (mainly for confidence intervals), and little multicollinearity between features. Our linear regression guide

shows each one visually.

19. What is the difference between MAE, MSE and RMSE?

Freshers · Regression

- **MAE** (mean absolute error): the average size of the errors. Easy to explain and treats all errors equally.
- **MSE** (mean squared error): the average of squared errors. Big errors count much more. Its unit is squared, which is hard to read.
- **RMSE**: the square root of MSE, back in the original unit, still punishing big errors.

PYTHON

```
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

y_true = np.array([50, 60, 70, 80, 90])     # actual flat prices (lakh)
y_pred = np.array([52, 58, 75, 78, 70])     # one big miss on the last flat

print("MAE: ", mean_absolute_error(y_true, y_pred))
print("MSE: ", mean_squared_error(y_true, y_pred))
print("RMSE:", round(np.sqrt(mean_squared_error(y_true, y_pred)), 2))
print("R2:  ", round(r2_score(y_true, y_pred), 3))
```

OUTPUT

```
MAE:  6.2
MSE:  87.4
RMSE: 9.35
R2:   0.563
```

One flat is off by ₹20 lakh. MAE says the typical miss is 6.2 lakh, while RMSE (9.35) is pulled up by that single large miss. Choose RMSE when big errors are especially costly, and MAE when you want a measure that outliers do not dominate.

20. What is R² (the coefficient of determination)?

Freshers · Regression

R² tells you what share of the variation in the target your model explains, compared with always predicting the average. R² = 1 − (sum of squared errors of the model / sum of squared errors of the mean). 1 is perfect, 0 means no better than predicting the mean, and it can be negative for a model worse than that. In the example above, R² is 0.563.

Plain R² never goes down when you add features, even useless ones. **Adjusted R²** corrects for the number of features, so it is better for comparing models with different feature counts.

21. How does logistic regression work, and why not use linear regression for classification?

Freshers · Classification

Logistic regression computes a weighted sum of the features, just like linear regression, and then passes it through the sigmoid function to get a probability between 0 and 1. If the probability is above a threshold (usually 0.5), it predicts the positive class. It is trained by minimising log loss, not MSE.

PYTHON

```
import numpy as np
from sklearn.linear_model import LogisticRegression

hours = np.array([[1], [2], [3], [4], [5], [6], [7], [8]])
passed = np.array([0, 0, 0, 1, 0, 1, 1, 1])

model = LogisticRegression().fit(hours, passed)
for h in [2, 4.5, 7]:
    print(f"{h} hours -> P(pass) = {model.predict_proba([[h]])[0, 1]:.3f}")
```

OUTPUT

```
2 hours -> P(pass) = 0.103
4.5 hours -> P(pass) = 0.500
7 hours -> P(pass) = 0.897
```

Linear regression is a poor fit for classification because its output is not limited to 0 to 1, so it cannot be read as a probability, and a few extreme points can shift the line and move the decision boundary badly. Our logistic regression guide walks through this with interactive charts.

22. What is the sigmoid function?

Freshers · Classification

The sigmoid (logistic) function is `σ(z) = 1 / (1 + e^-z)`. It turns any real number into a value between 0 and 1: large positive inputs go close to 1, large negative inputs go close to 0, and 0 maps to exactly 0.5.

That makes it ideal for turning a score into a probability in logistic regression and in the output layer of binary neural networks. Its derivative is simple, σ(z) × (1 − σ(z)), which is useful in backpropagation. Its weakness is that it flattens at both ends, so gradients become tiny there (question 70).

23. How does the KNN algorithm work, and how do you choose K?

Freshers · Classification

K-nearest neighbours stores the training data and, for a new point, finds the K closest training points (usually by Euclidean distance). For classification it takes a majority vote of their labels; for regression it averages their values. It has no real training step, so it is called a lazy learner.

Small K (like 1) follows the noise and overfits; large K smooths too much and underfits. Choose K with cross-validation, use an odd K for two classes to avoid ties, and always scale features first.

PYTHON

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

X, y = load_breast_cancer(return_X_y=True)
for k in [1, 3, 5, 11, 21, 51]:
    model = make_pipeline(StandardScaler(), KNeighborsClassifier(n_neighbors=k))
    print(f"K={k:2d}  CV accuracy={cross_val_score(model, X, y, cv=5).mean():.3f}")
```

OUTPUT

```
K= 1  CV accuracy=0.954
K= 3  CV accuracy=0.960
K= 5  CV accuracy=0.965
K=11  CV accuracy=0.965
K=21  CV accuracy=0.956
K=51  CV accuracy=0.951
```

Here K = 5 and K = 11 do best. KNN is easy to understand but slow to predict on large datasets and weak in high dimensions. Do not confuse it with k-means clustering, where K means the number of clusters.

24. What is Naive Bayes, and why is it called naive?

Freshers · Classification

Naive Bayes is a classifier based on Bayes' theorem: P(class | features) ∝ P(features | class) × P(class). It is called naive because it assumes every feature is independent of the others once you know the class. In spam filtering, that means it treats "free" and "prize" as unrelated words, which is not true, but the model still works surprisingly well.

PYTHON

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB

texts = ["win a free prize now", "free cash offer win", "meeting at 5 pm",
         "project report due tomorrow", "claim your free prize", "lunch tomorrow?"]
labels = [1, 1, 0, 0, 1, 0]                  # 1 = spam

vec = CountVectorizer()
model = MultinomialNB().fit(vec.fit_transform(texts), labels)

new = ["free prize for you", "report meeting tomorrow"]
print(model.predict(vec.transform(new)))
print(model.predict_proba(vec.transform(new)).round(3))
```

OUTPUT

```
[1 0]
[[0.095 0.905]
 [0.944 0.056]]
```

Variants: Gaussian NB for continuous features, Multinomial NB for word counts, Bernoulli NB for yes/no features. It is very fast, needs little data and is a strong baseline for text. Its probability estimates are often poorly calibrated (question 83).

25. How does a decision tree work?

Freshers · Trees & ensembles

A decision tree predicts by asking a series of yes or no questions about the features, such as "income ≤ ₹36,500?". At each node it tries every feature and threshold and picks the split that makes the child nodes purest, measured with Gini impurity or entropy for classification and squared error for regression. It repeats this until a stopping rule is met, and each leaf holds the prediction.

Trees are easy to explain and need no scaling, but they overfit easily, so you limit `max_depth` or prune them. You can grow one step by step in our interactive decision tree guide.

26. What are Gini impurity and entropy?

Freshers · Trees & ensembles

Both measure how mixed the classes in a node are. **Gini impurity** = $1 - \Sigma\, p_i^2$, the chance of mislabelling a random point if you guess using the node's class mix. **Entropy** = $-\Sigma\, p_i \log_2 p_i$, the uncertainty in the node. Both are 0 for a pure node; for two classes Gini peaks at 0.5 and entropy at 1.

The tree picks the split with the largest drop in impurity; with entropy this drop is called **information gain**. Gini and entropy choose the same split in most cases. Gini is the scikit-learn default and is slightly faster because it needs no logarithm.

27. What is a support vector machine (SVM)?

Freshers · Classification

An SVM finds the boundary that separates two classes with the widest possible gap, called the margin. Only the points closest to the boundary, the **support vectors**, decide where it goes. The `C` parameter controls how many points may sit inside the margin or on the wrong side: small C gives a wide, tolerant margin, large C a narrow, strict one.

With the **kernel trick** (question 59), an SVM can draw curved boundaries without ever computing new features. SVMs work well on small to medium datasets with many features, but need scaled features and get slow on very large data. Try it in our SVM playground.

28. What is a confusion matrix?

Freshers · Model evaluation

A confusion matrix is a table that compares predicted classes with actual classes. For two classes it has four cells: true positives (TP), false positives (FP), false negatives (FN) and true negatives (TN). Every classification metric, from accuracy to F1, is built from these four numbers.

PYTHON

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

# 10 patients: 1 = has the disease
y_true = [1, 1, 1, 1, 0, 0, 0, 0, 0, 0]
y_pred = [1, 1, 1, 0, 1, 0, 0, 0, 0, 0]

print(confusion_matrix(y_true, y_pred))      # rows: actual 0, 1
print("Accuracy: ", accuracy_score(y_true, y_pred))
print("Precision:", precision_score(y_true, y_pred))
print("Recall:   ", recall_score(y_true, y_pred))
print("F1:       ", f1_score(y_true, y_pred))
```

OUTPUT

```
[[5 1]
 [1 3]]
Accuracy:  0.8
Precision: 0.75
Recall:    0.75
F1:        0.75
```

scikit-learn puts actual classes in rows and predicted classes in columns, with class 0 first, so the layout is [[TN, FP], [FN, TP]]. Here: 5 true negatives, 1 false positive, 1 false negative, 3 true positives. Our interactive confusion matrix guide covers every metric built from it.

29. Why can accuracy be misleading?

Freshers · Model evaluation

Accuracy is the share of correct predictions. On imbalanced data it can look excellent while the model is useless. If 99% of transactions are genuine, a model that always says "not fraud" scores 99% accuracy and catches no fraud at all:

PYTHON

```python
import numpy as np
from sklearn.dummy import DummyClassifier
from sklearn.metrics import accuracy_score, recall_score

y = np.array([0] * 990 + [1] * 10)          # 1% fraud
X = np.zeros((1000, 1))

lazy = DummyClassifier(strategy="most_frequent").fit(X, y)
pred = lazy.predict(X)
print("Accuracy:", accuracy_score(y, pred))
print("Fraud caught (recall):", recall_score(y, pred))
```

OUTPUT

```
Accuracy: 0.99
Fraud caught (recall): 0.0
```

For imbalanced problems, report precision, recall, F1, PR AUC or a confusion matrix instead, and always compare against a simple baseline like this one.

30. What is the difference between precision and recall?

Freshers · Model evaluation

Precision = TP / (TP + FP): of everything the model flagged as positive, how much really was positive. **Recall** (sensitivity) = TP / (TP + FN): of all the real positives, how many the model found.

- Favour **recall** when missing a positive is costly: cancer screening, fraud detection.
- Favour **precision** when false alarms are costly: spam filtering (you do not want real email in spam), recommending a risky medical procedure.

They usually trade off: lowering the decision threshold raises recall and lowers precision (question 48). The "Pick the right metric" game above lets you practise this choice.

31. What is the F1 score?

Freshers · Model evaluation

The F1 score is the harmonic mean of precision and recall: $F1 = 2 \times P \times R / (P + R)$. It ranges from 0 to 1 and is high only when both precision and recall are high. With precision 0.9 and recall 0.1 the simple average is 0.5, but F1 is only 0.18, which reflects how poor the model really is.

Use F1 when you care about both kinds of error and the classes are imbalanced. If one matters more, use the F-beta score: $\beta > 1$ weights recall more, $\beta < 1$ weights precision more.

32. What are the ROC curve and AUC?

Freshers · Model evaluation

The ROC curve plots the true positive rate (recall) against the false positive rate for every possible threshold. AUC is the area under that curve. It equals the probability that the model gives a random positive example a higher score than a random negative one: 0.5 is random guessing and 1.0 is perfect.

AUC is useful because it does not depend on one chosen threshold. On heavily imbalanced data, though, it can look too optimistic, and the precision-recall curve is more honest (question 47).

33. What is cross-validation?

Freshers · Model evaluation

In k-fold cross-validation the data is split into k equal parts (folds). The model is trained k times, each time on k − 1 folds and tested on the remaining one, and the k scores are averaged. Every data point is used for testing exactly once.

PYTHON

```
from sklearn.datasets import load_wine
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.tree import DecisionTreeClassifier

X, y = load_wine(return_X_y=True)
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
scores = cross_val_score(DecisionTreeClassifier(random_state=0), X, y, cv=cv)

print("Fold scores:", scores.round(3))
print(f"Mean: {scores.mean():.3f}  Std: {scores.std():.3f}")
```

OUTPUT

```
Fold scores: [0.917 0.833 0.972 0.971 0.943]
Mean: 0.927  Std: 0.051
```

It gives a more reliable estimate than a single split and shows how stable the model is: here scores range from 0.83 to 0.97 depending on the fold. Five or ten folds is typical. Use stratified folds for classification and time-based folds for time series (question 65). Reference: scikit-learn cross-validation guide.

34. What is clustering, and how does k-means work?

Freshers · Unsupervised

Clustering is grouping similar data points without labels. K-means is the most common method:

1. Choose K, the number of clusters.
2. Place K starting centroids (k-means++ spreads them out).
3. Assign every point to its nearest centroid.
4. Move each centroid to the mean of its points.
5. Repeat steps 3 and 4 until nothing changes.

It minimises the within-cluster sum of squares (inertia). It is fast but assumes round clusters of similar size, needs scaled features and only finds a local minimum, so it is run several times. Watch it step by step in our k-means clustering playground.

35. How do you choose the number of clusters K in k-means?

Freshers · Unsupervised

Two standard methods: the **elbow method** plots inertia against K and looks for the bend where adding clusters stops helping much, and the **silhouette score** measures how close each point is to its own cluster compared with the next nearest one, so you pick the K with the highest average.

PYTHON

```
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_score

X, _ = make_blobs(n_samples=500, centers=4, random_state=7)
for k in range(2, 8):
    km = KMeans(n_clusters=k, n_init=10, random_state=0).fit(X)
    print(f"K={k}  inertia={km.inertia_:8.0f}  silhouette={silhouette_score(X, km.labels_):.3f}")
```

OUTPUT

```
K=2  inertia=   15172  silhouette=0.588
K=3  inertia=    4313  silhouette=0.763
K=4  inertia=     920  silhouette=0.811
K=5  inertia=     818  silhouette=0.693
K=6  inertia=     731  silhouette=0.578
K=7  inertia=     651  silhouette=0.472
```

Inertia drops sharply until K = 4 and then flattens, and the silhouette score peaks at K = 4, so both agree on four clusters. In real projects, also check whether the clusters are useful for the business.

36. What is PCA (principal component analysis)?

Freshers · Unsupervised

PCA is a dimensionality reduction technique. It finds new axes, called principal components, that are combinations of the original features. The first component points in the direction where the data varies most, the second in the next-most direction at right angles to the first, and so on. Keeping only the first few components keeps most of the information with far fewer features.

PYTHON

```
import numpy as np
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA

X, _ = load_digits(return_X_y=True)           # 64 features (8x8 pixels)
pca = PCA().fit(X)
cum = np.cumsum(pca.explained_variance_ratio_)

print("First 5 components explain:", cum[4].round(3))
print("Components for 90% variance:", int(np.argmax(cum >= 0.90)) + 1)
print("Components for 95% variance:", int(np.argmax(cum >= 0.95)) + 1)
```

OUTPUT

```
First 5 components explain: 0.545
Components for 90% variance: 21
Components for 95% variance: 29
```

The 64 pixel features of the digits dataset can be cut to 21 components while keeping 90% of the variance. PCA is used to speed up models, reduce noise and plot high-dimensional data in 2D. Scale features before PCA, and remember that the new components are harder to interpret than the original features.

37. What is gradient descent?

Freshers · Optimisation

Gradient descent is the method most models use to learn. It starts with some parameter values, computes the gradient of the loss (the direction in which the loss rises fastest), and takes a small step the opposite way. It repeats this until the loss stops falling. The size of each step is the **learning rate**.

PYTHON

```python
import numpy as np

# Fit y = w*x + b with plain gradient descent
rng = np.random.default_rng(0)
x = rng.uniform(0, 10, 100)
y = 3 * x + 5 + rng.normal(0, 1, 100)

w, b, lr = 0.0, 0.0, 0.01
for step in range(2001):
    pred = w * x + b
    dw = 2 * np.mean((pred - y) * x)       # gradient of MSE with respect to w
    db = 2 * np.mean(pred - y)             # gradient of MSE with respect to b
    w, b = w - lr * dw, b - lr * db
    if step % 500 == 0:
        print(f"step {step:4d}  MSE={np.mean((pred - y) ** 2):8.3f}  w={w:.3f}  b={b:.3f}")
```

OUTPUT

```
step    0  MSE= 539.563  w=2.892  b=0.427
step  500  MSE=   0.987  w=3.050  b=4.545
step 1000  MSE=   0.941  w=2.993  b=4.948
step 1500  MSE=   0.940  w=2.988  b=4.988
step 2000  MSE=   0.940  w=2.987  b=4.992
```

The true line is y = 3x + 5, and gradient descent finds w ≈ 2.99 and b ≈ 4.99. If the learning rate is too small, training is slow; if it is too large, the loss jumps around or blows up. The gradient descent tool on this page shows all three cases.

38. What is the difference between a parameter and a hyperparameter?

Freshers · Optimisation

Parameters are learned from the data during training: the weights of a linear model, the split thresholds of a tree, the weights of a neural network. **Hyperparameters** are set by you before training and control how learning happens: the learning rate, K in KNN, `max_depth` of a tree, `C` in an SVM, the number of layers in a network.

Hyperparameters are chosen by trying different values and comparing validation scores, usually with grid search, random search or Bayesian optimisation (question 44).

39. What is a neural network?

Freshers · Deep learning

A neural network is a model made of layers of simple units called neurons. Each neuron takes inputs, multiplies them by weights, adds a bias and passes the result through an activation function. An input layer receives the features, one or more hidden layers learn intermediate patterns, and an output layer gives the prediction.

With enough neurons and non-linear activations, a network can approximate almost any function. It learns its weights with backpropagation and gradient descent. Networks with many hidden layers are called deep neural networks, which is where the term deep learning comes from.

40. What is an activation function, and what are the common ones?

Freshers · Deep learning

An activation function adds non-linearity to a neuron. Without it, stacking layers would still give a single linear model, however deep the network.

Function	Output	Where it is used
ReLU: max(0, x)	0 to ∞	Default for hidden layers; fast and avoids vanishing gradients for positive inputs
Sigmoid	0 to 1	Output layer for binary classification
Tanh	−1 to 1	Hidden layers in older networks and some RNNs
Softmax	Probabilities that sum to 1	Output layer for multiclass classification
GELU, Leaky ReLU	Smooth or slightly negative	Transformers (GELU); avoiding "dead" ReLU units (Leaky ReLU)

Intermediate machine learning interview questions

41. What is the difference between batch, stochastic and mini-batch gradient descent?

Intermediate · Optimisation

Type	Data used for each update	Pros	Cons
Batch	The whole training set	Smooth, stable steps	Slow and memory-heavy on big data
Stochastic (SGD)	One example	Very fast updates; noise can help escape poor local minima	Noisy, zig-zagging path
Mini-batch	A small batch, such as 32 to 512 examples	Balance of speed and stability; fits GPUs well	Batch size is one more hyperparameter

Mini-batch is what almost everyone uses for neural networks, and people often just call it SGD. Optimisers like Momentum, RMSProp and Adam build on it by adapting the step size for each parameter.

42. What is the difference between L1 and L2 regularisation?

Intermediate · Optimisation

Regularisation adds a penalty on the size of the weights to the loss, so the model prefers simpler solutions. **L2 (Ridge)** adds $\lambda \Sigma w^2$, which shrinks all weights smoothly towards zero but rarely makes them exactly zero. **L1 (Lasso)** adds $\lambda \Sigma |w|$, which pushes some weights to exactly zero, so it also selects features.

PYTHON

```
import numpy as np
from sklearn.datasets import make_regression
from sklearn.linear_model import Lasso, LinearRegression, Ridge

X, y = make_regression(n_samples=200, n_features=20, n_informative=5, noise=10, random_state=0)

for name, model in [("Linear", LinearRegression()), ("Ridge (L2)", Ridge(alpha=10)),
                    ("Lasso (L1)", Lasso(alpha=5))]:
    coef = model.fit(X, y).coef_
    print(f"{name:11s} zero coefs: {np.sum(np.abs(coef) < 1e-6):2d}   "
          f"largest |coef|: {np.abs(coef).max():.1f}")
```

OUTPUT

```
Linear      zero coefs:  0   largest |coef|: 83.8
Ridge (L2)  zero coefs:  0   largest |coef|: 79.0
Lasso (L1)  zero coefs: 15   largest |coef|: 77.8
```

The data has 20 features but only 5 useful ones. Lasso set 15 coefficients to zero, while Ridge kept all of them. Use L1 when you expect few useful features or want a simpler model, and L2 when many features each contribute a little or features are correlated. In scikit-learn the strength is `alpha` for Ridge and Lasso, and `C = 1/λ` for logistic regression and SVM.

43. What is Elastic Net, and when would you use it?

Intermediate · Optimisation

Elastic Net combines the L1 and L2 penalties: α × (r × Σ|w| + (1 − r)/2 × Σw²), where `l1_ratio` (r) sets the mix. It keeps Lasso's ability to zero out features while behaving better when features are correlated. Lasso tends to pick one feature from a correlated group at random, whereas Elastic Net tends to keep or drop the group together.

Use it when you have many correlated features, or more features than rows, and tune both `alpha` and `l1_ratio` with cross-validation (`ElasticNetCV` does this directly).

44. How do you tune hyperparameters?

Intermediate · Model evaluation

- **Grid search:** tries every combination from lists you give. Thorough, but the cost multiplies with each new hyperparameter.
- **Random search:** tries a fixed number of random combinations. Usually finds a good result faster, because only a few hyperparameters really matter.
- **Bayesian optimisation** (Optuna, Hyperopt): uses past results to pick promising combinations next. Best when each training run is expensive.

PYTHON

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

X, y = load_breast_cancer(return_X_y=True)
pipe = make_pipeline(StandardScaler(), SVC())
grid = GridSearchCV(pipe, {"svc__C": [0.1, 1, 10, 100],
                           "svc__gamma": [0.001, 0.01, 0.1]}, cv=5)
grid.fit(X, y)

print("Best params:", grid.best_params_)
print("Best CV accuracy:", round(grid.best_score_, 3))
print("Combinations tried:", len(grid.cv_results_["params"]))
```

OUTPUT

```
Best params: {'svc__C': 10, 'svc__gamma': 0.01}
Best CV accuracy: 0.979
Combinations tried: 12
```

Always tune with cross-validation on the training data only, and put preprocessing inside a pipeline so it is refitted in every fold. Note the `svc__C` naming: the step name, two underscores, then the parameter.

45. What is stratified k-fold cross-validation, and when do you need it?

Intermediate · Model evaluation

Stratified k-fold makes sure every fold has roughly the same class proportions as the full dataset. With plain k-fold on imbalanced data, a fold might end up with very few or even no positive examples, which makes its score meaningless and the overall estimate noisy.

Use it for any classification problem, especially imbalanced ones; scikit-learn uses it by default when you pass an integer `cv` to a classifier. Do not use it for time series (use `TimeSeriesSplit`) or when rows are grouped, such as several rows per patient (use `GroupKFold` so one patient never appears in both train and test).

46. How do you handle imbalanced data?

Intermediate · Model evaluation

- Use the right metrics: recall, precision, F1, PR AUC, not accuracy.
- Give the rare class more weight: `class_weight="balanced"` in most scikit-learn models, `scale_pos_weight` in XGBoost.
- Resample: oversample the minority class (randomly or with SMOTE) or undersample the majority, on the training data only.
- Tune the decision threshold instead of using 0.5.
- Collect more examples of the rare class if you can, or treat it as anomaly detection when positives are extremely rare.

PYTHON

```
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score
from sklearn.model_selection import train_test_split

X, y = make_classification(n_samples=10000, weights=[0.98], flip_y=0.01, random_state=3)
X_tr, X_te, y_tr, y_te = train_test_split(X, y, stratify=y, random_state=3)

for cw in [None, "balanced"]:
    pred = LogisticRegression(class_weight=cw, max_iter=1000).fit(X_tr, y_tr).predict(X_te)
    print(f"class_weight={str(cw):8s}  recall={recall_score(y_te, pred):.2f}  "
          f"precision={precision_score(y_te, pred):.2f}")
```

OUTPUT

```
class_weight=None      recall=0.08  precision=0.33
class_weight=balanced  recall=0.73  precision=0.12
```

With 2% positives, the default model caught only 8% of them. Class weights raised recall to 73%, at the cost of more false alarms. Which balance is right depends on the cost of each kind of mistake.

47. When should you use a precision-recall curve instead of a ROC curve?

Intermediate · Model evaluation

Use the precision-recall (PR) curve when the positive class is rare and you mostly care about it, as in fraud, disease or defect detection. ROC uses the false positive rate, whose denominator is the huge number of negatives, so even thousands of false alarms barely move it. Precision shows those false alarms directly.

PYTHON

```
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, average_precision_score
from sklearn.model_selection import train_test_split

X, y = make_classification(n_samples=5000, weights=[0.95], random_state=1)
X_tr, X_te, y_tr, y_te = train_test_split(X, y, stratify=y, random_state=1)

proba = LogisticRegression(max_iter=1000).fit(X_tr, y_tr).predict_proba(X_te)[:, 1]
print("ROC AUC:          ", round(roc_auc_score(y_te, proba), 3))
print("PR AUC (avg prec):", round(average_precision_score(y_te, proba), 3))
print("Positive rate:    ", round(y_te.mean(), 3))
```

OUTPUT

```
ROC AUC:           0.926
PR AUC (avg prec): 0.621
Positive rate:     0.054
```

With about 5% positives, ROC AUC is a comfortable 0.93, but PR AUC is only 0.62. The model is good at ranking, but many of its alerts would still be false. A random model has PR AUC equal to the positive rate (0.054 here), so always compare against that baseline.

48. How do you choose the classification threshold?

Intermediate · Model evaluation

Most classifiers output a probability, and 0.5 is only a default. Choose the threshold from the cost of each error: for cancer screening, missing a malignant tumour is far worse than an extra biopsy, so you lower the threshold to raise recall.

PYTHON

```
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

X, y = load_breast_cancer(return_X_y=True)
y = 1 - y                                  # make 1 = malignant (the class we must catch)
X_tr, X_te, y_tr, y_te = train_test_split(X, y, stratify=y, random_state=0)
proba = make_pipeline(StandardScaler(), LogisticRegression()).fit(X_tr, y_tr).predict_proba(X_te)[:, 1]

for t in [0.5, 0.3, 0.1]:
    pred = (proba >= t).astype(int)
    print(f"threshold={t}  recall={recall_score(y_te, pred):.3f}  precision={precision_score(y_te, pred):.3f}")
```

OUTPUT

```
threshold=0.5  recall=0.962  precision=1.000
threshold=0.3  recall=0.962  precision=0.895
threshold=0.1  recall=1.000  precision=0.828
```

Lowering the threshold to 0.1 catches every malignant case (recall 1.0), at the cost of precision dropping to 0.83. Pick the threshold on validation data, using a cost formula, a minimum recall target, or the point that maximises F1. Never pick it on the test set.

49. What is log loss (cross-entropy)?

Intermediate · Classification

Log loss measures how good predicted probabilities are, not just the final class. For one example with true label y and predicted probability p, it is $-[y \log p + (1 - y) \log(1 - p)]$. A confident correct prediction gives a loss near 0, and a confident wrong one gives a very large loss: predicting 0.01 for a true positive costs 4.6, while 0.4 costs 0.92.

It is the loss that logistic regression and classification neural networks minimise, and it comes straight from maximum likelihood (question 68). For more than two classes it becomes categorical cross-entropy: $-\Sigma\, y_k \log p_k$.

50. How do models handle multiclass classification, and what are macro, micro and weighted averages?

Intermediate · Classification

Some models handle many classes natively: trees, Naive Bayes, KNN, and softmax regression or neural networks. Binary models are extended with **one-vs-rest** (one model per class against all others) or **one-vs-one** (one model per pair of classes, used by SVC).

- **Macro average:** compute the metric for each class and take the plain average. Every class counts equally, so small classes matter.
- **Weighted average:** like macro, but weighted by each class's size.

- **Micro average:** pool all TP, FP and FN first. For single-label problems, micro F1 equals accuracy.

Report macro F1 when rare classes matter as much as common ones.

51. What is the difference between bagging and boosting?

Intermediate · Trees & ensembles

	Bagging	Boosting
How models are trained	In parallel, each on a bootstrap sample (random sample with replacement)	One after another, each fixing the errors of the previous ones
Main effect	Reduces variance	Reduces bias (and variance, when tuned)
Base models	Usually deep, low-bias trees	Usually shallow trees
Overfitting risk	Low	Higher; needs a learning rate and early stopping
Examples	Random forest, bagged trees	AdaBoost, gradient boosting, XGBoost, LightGBM, CatBoost

On tabular data, a well-tuned gradient boosting model usually beats a random forest, while the random forest needs much less tuning.

52. How does a random forest work?

Intermediate · Trees & ensembles

A random forest trains many decision trees, each on a bootstrap sample of the rows, and at every split each tree considers only a random subset of the features (`max_features`). Classification takes a majority vote; regression takes the average. The two kinds of randomness make the trees different from each other, so their errors partly cancel out.

PYTHON

```
from sklearn.datasets import load_breast_cancer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

X, y = load_breast_cancer(return_X_y=True)
X_tr, X_te, y_tr, y_te = train_test_split(X, y, stratify=y, random_state=0)

tree = DecisionTreeClassifier(random_state=0).fit(X_tr, y_tr)
rf = RandomForestClassifier(n_estimators=300, oob_score=True, random_state=0).fit(X_tr, y_tr)

print("Single tree test accuracy:  ", round(tree.score(X_te, y_te), 3))
print("Random forest test accuracy:", round(rf.score(X_te, y_te), 3))
print("Random forest OOB score:    ", round(rf.oob_score_, 3))
```

OUTPUT

```
Single tree test accuracy:   0.902
Random forest test accuracy: 0.944
Random forest OOB score:     0.962
```

Each tree leaves out about 37% of the rows from its bootstrap sample. Predicting each row with only the trees that did not see it gives the **out-of-bag (OOB) score**, a free validation estimate. Main hyperparameters: `n_estimators`, `max_features`, `max_depth`, `min_samples_leaf`.

53. How does gradient boosting work, and what are its key hyperparameters?

Intermediate · Trees & ensembles

Gradient boosting builds trees one after another. It starts with a simple prediction (such as the average), then each new shallow tree is trained to predict the gradient of the loss, which for squared error is just the residuals of the current model. The new tree's output, multiplied by a small learning rate, is added to the model. Step by step the ensemble corrects its own mistakes.

- `learning_rate`: how much each tree contributes. Smaller is usually better but needs more trees.
- `n_estimators`: number of trees; choose it with early stopping (question 85).
- `max_depth` (often 3 to 8): how complex each tree is.
- `subsample` and column sampling: add randomness and reduce overfitting.
- Regularisation terms in XGBoost: `lambda`, `alpha`, `gamma`, `min_child_weight`.

54. How does AdaBoost work?

Intermediate · Trees & ensembles

AdaBoost (adaptive boosting) trains a sequence of weak learners, usually one-split trees called stumps. After each round it increases the weights of the examples that were misclassified, so the next learner focuses on the hard cases. Each learner also gets a vote weight based on its accuracy, and the final prediction is the weighted vote of all of them.

It was the first practical boosting algorithm. Gradient boosting generalises the idea to any differentiable loss. Because AdaBoost keeps raising the weight of misclassified points, it is sensitive to noisy labels and outliers.

55. How do you measure feature importance?

Intermediate · Data & features

- **Coefficients** of linear models, on scaled features.
- **Impurity-based importance** (`feature_importances_`) in trees: fast, but biased towards features with many unique values and computed on training data.
- **Permutation importance:** shuffle one feature in the validation set and measure how much the score drops. Model-agnostic and more reliable.
- **SHAP values:** how much each feature pushed each individual prediction up or down (question 88).

PYTHON

```
import numpy as np
from sklearn.datasets import load_diabetes
from sklearn.ensemble import RandomForestRegressor
from sklearn.inspection import permutation_importance
from sklearn.model_selection import train_test_split

data = load_diabetes()
X_tr, X_te, y_tr, y_te = train_test_split(data.data, data.target, random_state=0)
rf = RandomForestRegressor(random_state=0).fit(X_tr, y_tr)

result = permutation_importance(rf, X_te, y_te, n_repeats=10, random_state=0)
for i in np.argsort(result.importances_mean)[::-1][:4]:
    print(f"{data.feature_names[i]:4s} drop in R2 when shuffled: {result.importances_mean[i]:.3f}")
```

OUTPUT

```
bmi  drop in R2 when shuffled: 0.114
s5   drop in R2 when shuffled: 0.094
bp   drop in R2 when shuffled: 0.043
age  drop in R2 when shuffled: 0.018
```

Shuffling BMI drops R^2 by 0.11, so the model relies on it most. If two features are strongly correlated, permutation importance can underrate both, because the model can still use the other one. Reference: scikit-learn permutation importance.

56. What are the main feature selection methods?

Intermediate · Data & features

- **Filter methods:** score each feature on its own, without a model: correlation, chi-square, mutual information, ANOVA F-test. Fast, but they ignore interactions.
- **Wrapper methods:** try subsets of features with a model and keep the best: forward selection, backward elimination, recursive feature elimination (RFE). Accurate but slow.
- **Embedded methods:** selection happens during training: Lasso, tree-based importance, `SelectFromModel`.

Fewer features means faster training, less overfitting and simpler explanations. Whatever the method, do the selection inside cross-validation, or you get the leakage shown in question 15.

57. What is multicollinearity, and why does it matter?

Intermediate · Regression

Multicollinearity means two or more features are strongly correlated, such as flat size in square feet and number of rooms. Predictions may still be fine, but the individual coefficients of a linear model become unstable: they can swing wildly or even change sign with small changes in the data, so you cannot trust them for explanation.

Detect it with a correlation matrix or the **variance inflation factor (VIF)**; a VIF above about 5 to 10 is a warning. Fix it by dropping or combining correlated features, using PCA, or using Ridge regression, which handles correlated features gracefully.

58. What is pruning in decision trees?

Intermediate · Trees & ensembles

Pruning makes a decision tree smaller so it generalises better. **Pre-pruning** stops growth early with limits such as `max_depth`, `min_samples_split`, `min_samples_leaf` or `max_leaf_nodes`. **Post-pruning** grows the full tree and then removes branches that do not earn their place; scikit-learn uses cost complexity pruning with `ccp_alpha`.

Tune either kind with cross-validation. The decision tree guide shows a fully grown tree with 18 leaves losing to a pruned one with 11.

59. What is the kernel trick, and what do C and gamma do in an SVM?

Intermediate · Classification

The kernel trick lets an SVM act as if the data had been mapped into a higher-dimensional space where it becomes linearly separable, without ever computing that mapping. It only needs a kernel function that returns the dot product in that space. Common kernels: linear, polynomial and RBF (Gaussian).

`C` controls how strongly misclassified points are punished: large C fits the training data tightly and can overfit, while small C gives a wider, smoother margin. `gamma` (for RBF) controls how far one point's influence reaches: large gamma gives very wiggly boundaries, small gamma very smooth ones. Tune them together on a log scale.

60. When would you use DBSCAN instead of k-means?

Intermediate · Unsupervised

DBSCAN groups points that are packed closely together (at least `min_samples` points within distance `eps`) and marks points in sparse areas as noise. Choose it when:

- clusters have irregular shapes, such as moons or rings, which k-means cuts with straight lines;
- you do not know the number of clusters;
- you want outliers flagged instead of forced into a cluster.

Its weaknesses: it struggles when clusters have very different densities, and `eps` is hard to choose in high dimensions. HDBSCAN fixes much of this.

61. How does hierarchical clustering work?

Intermediate · Unsupervised

Agglomerative hierarchical clustering starts with every point as its own cluster and repeatedly merges the two closest clusters until only one is left. The full history of merges is drawn as a tree called a **dendrogram**, and you choose the number of clusters by cutting the tree at a certain height.

The linkage decides what "closest" means: single (nearest points), complete (furthest points), average, or Ward (smallest increase in variance, the usual default). It needs no K up front and shows the structure at every level, but it uses $O(n^2)$ memory, so it suits small to medium datasets.

62. How does PCA work, and how do you choose the number of components?

Intermediate · Unsupervised

PCA centres (and usually scales) the data, computes the covariance matrix, and finds its eigenvectors and eigenvalues. The eigenvectors are the principal components, the directions of the new axes, and each eigenvalue tells how much variance that direction holds. In practice it is computed with singular value decomposition (SVD), which is faster and more stable.

To choose how many components to keep, plot the cumulative explained variance and keep enough for 90 to 95% (question 36 needed 21 of 64 for 90%), look for an elbow in the scree plot, or tune the number of components as a hyperparameter of the model that uses them.

63. How are t-SNE and UMAP different from PCA?

Intermediate · Unsupervised

PCA is linear and keeps the large-scale structure, such as overall spread. t-SNE and UMAP are non-linear methods built mainly for visualisation: they keep neighbourhoods, so points that are close in the original space stay close in 2D, and clusters show up clearly.

- Distances between clusters and cluster sizes in a t-SNE plot do not mean much.
- t-SNE is slow on large data and has no simple way to transform new points; UMAP is faster and can.
- Use PCA for preprocessing and compression; use t-SNE or UMAP to look at embeddings or clusters.

64. What is a scikit-learn Pipeline, and why should you use one?

Intermediate · Data & features

A Pipeline chains preprocessing steps and a model into a single object with one `fit` and one `predict`. A `ColumnTransformer` applies different steps to different columns.

PYTHON

```
import pandas as pd
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler

df = pd.DataFrame({"age": [22, 35, None, 51, 46, 29], "city": ["Surat", "Pune", "Pune", None, "Delhi", "Surat"],
                   "income": [3.2, 8.5, 6.1, 12.0, 9.4, 4.0], "bought": [0, 1, 0, 1, 1, 0]})
X, y = df.drop(columns="bought"), df["bought"]

prep = ColumnTransformer([
    ("num", Pipeline([("impute", SimpleImputer(strategy="median")), ("scale", StandardScaler())]), ["age", "income"]),
    ("cat", Pipeline([("impute", SimpleImputer(strategy="most_frequent")),
                      ("onehot", OneHotEncoder(handle_unknown="ignore"))]), ["city"]),
])
model = Pipeline([("prep", prep), ("clf", LogisticRegression())]).fit(X, y)

new = pd.DataFrame({"age": [40], "city": ["Mumbai"], "income": [10.0]})   # unseen city
print("Prediction:", model.predict(new), "| P(buy):", model.predict_proba(new)[0, 1].round(3))
```

OUTPUT

```
Prediction: [1] | P(buy): 0.741
```

- It prevents leakage: during cross-validation, imputers and scalers are fitted only on the training folds.
- The exact same steps run in training and in production, so there is no training-serving skew.
- It handles unseen values safely, like the new city "Mumbai" above, thanks to `handle_unknown="ignore"`.
- You can tune preprocessing and model hyperparameters together with GridSearchCV.

65. How do you validate a model on time series data?

Intermediate · Model evaluation

Never shuffle time series data before splitting. A random split lets the model train on the future and test on the past, which is leakage, and the score will look far better than it will be in production. Train on the past and test on the period right after it.

PYTHON

```
import numpy as np
from sklearn.model_selection import TimeSeriesSplit

days = np.arange(12)                          # 12 days of data, in time order
for i, (train, test) in enumerate(TimeSeriesSplit(n_splits=3).split(days), 1):
    print(f"Fold {i}: train days {train.min()}-{train.max()}, test days {test.min()}-{test.max()}")
```

OUTPUT

```
Fold 1: train days 0-2, test days 3-5
Fold 2: train days 0-5, test days 6-8
Fold 3: train days 0-8, test days 9-11
```

`TimeSeriesSplit` does this with an expanding window. Also make sure features only use information available at prediction time, for example lag features and rolling averages that stop at the day before.

66. How do you check whether a regression model is good beyond its score?

Intermediate · Regression

Look at the **residuals** (actual minus predicted). Plot them against the predictions: a random cloud around zero is good. Patterns mean problems:

- A curve means the relationship is not linear; add features or use a non-linear model.

- A funnel shape (spread growing with the prediction) is **heteroscedasticity**; try a log transform of the target or a model that handles it.
- Residuals that follow each other over time mean the errors are autocorrelated, often missing time features.
- A few huge residuals point to outliers or data errors.

Also compare against a simple baseline, such as predicting the mean or last week's value. A model has to beat it to be worth using.

67. What is the difference between generative and discriminative models?

Intermediate · Classification

A **discriminative** model learns the boundary between classes directly, P(y | x). Examples: logistic regression, SVM, decision trees, most neural network classifiers. A **generative** model learns how the data of each class is produced, P(x | y) and P(y), or the joint P(x, y), and uses Bayes' rule to classify. Examples: Naive Bayes, Gaussian mixture models, hidden Markov models.

Discriminative models usually classify more accurately when there is plenty of data. Generative models can work with less data, handle missing features and can generate new samples. Today's generative AI models, such as large language models, are generative models of text.

68. What is maximum likelihood estimation, and how does it relate to loss functions?

Intermediate · Optimisation

Maximum likelihood estimation (MLE) chooses the parameters that make the observed data most probable under the model. Taking the negative log of the likelihood turns a product into a sum and a maximisation into a minimisation, which gives the familiar loss functions:

- If you assume Gaussian noise around a linear prediction, MLE gives **mean squared error**, so ordinary least squares is MLE.
- If you assume the label follows a Bernoulli distribution with probability σ(wx + b), MLE gives **log loss**, which is logistic regression.
- Adding a prior on the weights gives MAP estimation; a Gaussian prior is L2 regularisation, a Laplace prior is L1.

69. How does backpropagation work?

Intermediate · Deep learning

Backpropagation computes the gradient of the loss with respect to every weight in a network. In the forward pass, inputs flow through the layers to produce a prediction and a loss. In the backward pass, the chain rule is applied from the output back to the input: each layer receives the gradient from the layer after it, multiplies by its own local derivative, and passes the result back. Gradient descent then updates every weight.

PYTHON

```python
import numpy as np

# One neuron: y_hat = sigmoid(w*x + b), loss = (y_hat - y)^2
x, y = 2.0, 1.0
w, b = 0.5, 0.0

z = w * x + b
y_hat = 1 / (1 + np.exp(-z))
loss = (y_hat - y) ** 2

# Chain rule, from the loss back to the weights
dloss_dyhat = 2 * (y_hat - y)
dyhat_dz = y_hat * (1 - y_hat)
dz_dw, dz_db = x, 1.0
dw = dloss_dyhat * dyhat_dz * dz_dw
db = dloss_dyhat * dyhat_dz * dz_db

print(f"prediction={y_hat:.4f}  loss={loss:.4f}")
print(f"dLoss/dw={dw:.4f}  dLoss/db={db:.4f}")
w -= 0.5 * dw; b -= 0.5 * db
print(f"after one step: prediction={1 / (1 + np.exp(-(w * x + b))):.4f}")
```

OUTPUT

```
prediction=0.7311  loss=0.0723
dLoss/dw=-0.2115  dLoss/db=-0.1058
after one step: prediction=0.7798
```

For one neuron the chain is dLoss/dw = dLoss/dŷ × dŷ/dz × dz/dw. After a single update the prediction moves from 0.731 towards the target of 1. Frameworks like PyTorch and TensorFlow do this automatically (autograd), but interviewers like to see that you can do it by hand.

70. What are vanishing and exploding gradients?

Intermediate · Deep learning

In deep networks the gradient is a product of many layer derivatives. If those are mostly smaller than 1, the gradient shrinks towards zero as it travels back, so the early layers barely learn: **vanishing gradients**. Sigmoid and tanh cause this because their slopes are tiny for large inputs. If the factors are larger than 1, the gradient grows huge and training blows up: **exploding gradients**, common in RNNs.

- ReLU-family activations instead of sigmoid in hidden layers.
- Careful weight initialisation (He for ReLU, Xavier/Glorot for tanh).
- Batch normalisation or layer normalisation.
- Residual (skip) connections, as in ResNets and transformers.
- Gradient clipping for exploding gradients; LSTM or GRU cells instead of plain RNNs.

71. What are dropout and batch normalisation?

Intermediate · Deep learning

Dropout randomly switches off a fraction of neurons (for example 20 to 50%) in each training step, so the network cannot rely on any single neuron and learns more robust features. It acts like training many smaller networks and averaging them. Dropout is turned off at inference time.

Batch normalisation normalises each layer's inputs to mean 0 and variance 1 within the mini-batch, then applies a learned scale and shift. It makes training faster and more stable, allows higher learning rates and adds a little regularisation. At inference it uses running averages collected during training. Transformers use **layer normalisation** instead, which normalises across features for each example and does not depend on batch size.

72. What is a convolutional neural network (CNN)?

Intermediate · Deep learning

A CNN is a neural network designed for grid-like data such as images. Instead of connecting every pixel to every neuron, a **convolutional layer** slides small filters (for example 3×3) across the image. Each filter learns to detect a pattern, like an edge or a texture, anywhere in the image. **Pooling layers** shrink the feature maps, and fully connected layers at the end make the prediction.

Because the same filter is reused everywhere (weight sharing), CNNs need far fewer parameters than fully connected networks and handle shifted objects well. Early layers learn edges, later layers learn shapes and objects. Uses: image classification, object detection, medical imaging, and some audio and text tasks.

73. What are RNNs and LSTMs?

Intermediate · Deep learning

A recurrent neural network (RNN) processes a sequence one step at a time and keeps a hidden state that carries information from earlier steps. That suits text, speech and time series. Plain RNNs struggle with long sequences because of vanishing gradients, so they forget early information.

An **LSTM** (long short-term memory) adds a cell state and three gates (forget, input and output) that learn what to keep, add and output, so it can remember information over much longer spans. A GRU is a simpler variant with two gates. For most language tasks, transformers (question 91) have replaced RNNs because they process all positions in parallel and handle long-range context better.

74. What are word embeddings?

Intermediate · Deep learning

Word embeddings represent each word as a dense vector of numbers, often 100 to 1,000 values, learned so that words used in similar contexts get similar vectors. Unlike one-hot vectors, they capture meaning: "king" and "queen" are close, and directions in the space can capture relationships, as in the famous king − man + woman ≈ queen.

Word2Vec and GloVe give each word one fixed vector. Modern models such as BERT and GPT produce **contextual embeddings**, where "bank" in "river bank" and "bank loan" gets different vectors. Sentence and document embeddings power semantic search and RAG systems (question 95).

75. What is transfer learning?

Intermediate · Deep learning

Transfer learning reuses a model trained on a large dataset as the starting point for a new, related task. An image model trained on ImageNet already knows edges, textures and shapes, so you can replace its last layer and train only that on a few thousand images of, say, plant diseases. You can also **fine-tune** some or all of the earlier layers with a small learning rate.

It needs much less data and compute, and it usually gives better accuracy than training from scratch. It is the standard approach in computer vision and NLP: fine-tuning BERT for classification and adapting large language models (question 94) are both transfer learning.

Interview questions for experienced candidates

76. How do you approach an end-to-end machine learning project?

Experienced · ML in production

1. **Frame the problem:** what decision will the model support, what does success look like in business terms, and is ML even needed? Agree on a metric and a baseline.
2. **Get and understand the data:** sources, labels, quality, leakage risks, and how the data will be available at prediction time.
3. **Build a simple baseline** (a rule, or logistic regression) to know what "good" means.
4. **Iterate:** features, models and tuning, judged with honest validation that matches how the model will be used (time-based splits, grouped splits).
5. **Analyse errors:** look at where the model fails, by segment, not just at the overall score.
6. **Deploy** with a reproducible pipeline, then **monitor** data, predictions and business metrics, and plan retraining.

Interviewers listen for the first and last steps most, because that is where real projects usually fail.

77. Your model scored well offline but performs badly in production. What could be wrong?

Experienced · ML in production

- **Data leakage** in training: a feature that is not really available at prediction time, or preprocessing fitted on all the data.
- **Training-serving skew:** features computed differently in the batch training code and the live service, different units, or different handling of missing values.
- **Wrong validation:** a random split on time-ordered or grouped data, so the offline score was too optimistic.
- **Data or concept drift** since training (next question).
- **Feedback loops:** the model's own decisions change the data it later sees, for example a fraud model blocking transactions so their labels are never observed.
- Bugs: wrong model version, stale features, a different threshold.

Debug by logging the exact features the live model receives and comparing them with the training data for the same entities.

78. What is the difference between data drift and concept drift, and how do you monitor them?

Experienced · ML in production

Data drift (covariate shift) means the distribution of the inputs changes: a new marketing campaign brings in younger users. **Concept drift** means the relationship between inputs and the target changes: after a policy change, the same customer profile now defaults at a different rate. Concept drift is worse, because the model's learned rules are now wrong.

- Monitor input feature distributions against training: population stability index (PSI), KL divergence, Kolmogorov-Smirnov tests.
- Monitor the distribution of predictions and the share of each predicted class.
- When labels arrive (often with a delay), track the real metrics over time and by segment.
- Set alerts, and retrain on a schedule or when drift crosses a limit, then validate before replacing the old model.

79. How do you deploy a machine learning model?

Experienced · ML in production

The first choice is between **batch** and **online** prediction. Batch scoring runs on a schedule and stores the results, which is simple and cheap and works for things like daily churn scores or nightly recommendations. Online serving exposes the model behind an API (FastAPI, TorchServe, a managed endpoint) for real-time requests such as fraud checks, where latency matters.

- Package the whole pipeline (preprocessing and model) with fixed library versions, usually in a Docker container.
- Track model versions, training data and metrics in a model registry (MLflow and similar tools).
- Release safely: shadow mode (score live traffic without acting on it), canary rollout or an A/B test.
- Log inputs and predictions for monitoring and future retraining, and keep a way to roll back.

80. How do you A/B test a new model, and why can offline and online metrics disagree?

Experienced · ML in production

Randomly split live users or requests between the current model (control) and the new one (treatment), decide the primary business metric and the sample size in advance, run long enough to cover weekly patterns, and check for a statistically significant difference. Also watch guardrail metrics such as latency, complaints and revenue.

Offline metrics like AUC or RMSE measure prediction quality on past data. Online metrics measure what the business cares about, such as clicks, conversions or losses, and they include effects that offline data cannot show: user behaviour changing in response to the model, position bias in recommendations, and the new model's effect on future data. That is why a better offline model sometimes loses online.

81. How do you train a model when the data does not fit in memory?

Experienced · ML in production

- Start by sampling: a well-chosen sample of a few million rows is often enough to choose features and models.
- Use efficient types: smaller numeric types, categoricals, and columnar formats like Parquet.
- Process in chunks, and use models that learn incrementally with `partial_fit` (SGDClassifier, MiniBatchKMeans, naive Bayes).
- Use libraries built for big data: LightGBM and XGBoost with external memory, Dask, or Spark MLlib on a cluster.
- For deep learning, stream mini-batches from disk with data loaders and use several GPUs if needed.
- Push aggregations to the database or data warehouse with SQL, so only the features come back.

82. How do recommendation systems work?

Experienced · ML in production

Content-based filtering recommends items similar to what a user liked, using item features such as genre or product category. **Collaborative filtering** uses behaviour: users who liked the same things in the past will like similar things in future. Matrix factorisation learns a vector for every user and every item so that their dot product predicts the rating or click. Most real systems are **hybrids**, often in two stages: fast candidate generation (for example with embeddings and nearest-neighbour search), then a ranking model on the shortlist.

Topics interviewers probe: the cold-start problem for new users and items (fall back to content features or popularity), implicit feedback (clicks, watch time) versus explicit ratings, and ranking metrics like precision@k, recall@k, MAP and NDCG.

83. What is probability calibration, and why does it matter?

Experienced · Model evaluation

A model is well calibrated if, among all the cases where it predicts 70%, about 70% really are positive. Many models rank cases well but give poorly calibrated probabilities: Naive Bayes is usually overconfident, SVMs do not output probabilities at all, and boosted trees can be skewed. Calibration matters whenever the probability itself is used, such as expected loss = probability × amount, or comparing risk across models.

PYTHON

```
from sklearn.calibration import CalibratedClassifierCV
from sklearn.datasets import make_classification
from sklearn.metrics import brier_score_loss
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

X, y = make_classification(n_samples=20000, n_features=20, n_informative=4,
                           n_redundant=10, random_state=0)
X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=0.5, random_state=0)

raw = GaussianNB().fit(X_tr, y_tr)
cal = CalibratedClassifierCV(GaussianNB(), method="isotonic", cv=5).fit(X_tr, y_tr)

print("Brier score, raw Naive Bayes:", round(brier_score_loss(y_te, raw.predict_proba(X_te)[:, 1]), 4))
print("Brier score, calibrated:     ", round(brier_score_loss(y_te, cal.predict_proba(X_te)[:, 1]), 4))
```

OUTPUT

```
Brier score, raw Naive Bayes: 0.1852
Brier score, calibrated:      0.1578
```

Fix it with Platt scaling (a logistic fit on the scores) or isotonic regression on held-out data, using `CalibratedClassifierCV`. Check it with a reliability curve (`CalibrationDisplay`) and the Brier score, which fell from 0.185 to 0.158 here. Lower is better.

84. Explain the bias-variance decomposition.

Experienced · Basics

For squared error, the expected test error at a point x, averaged over many possible training sets, splits into three parts:

$$E[(y - \hat{f}(x))^2] = \text{Bias}[\hat{f}(x)]^2 + \text{Var}[\hat{f}(x)] + \sigma^2$$

- **Bias²**: how far the average prediction across training sets is from the truth. It comes from the model's assumptions.
- **Variance**: how much the prediction changes from one training set to another. It comes from sensitivity to the sample.
- **σ²**: irreducible noise in the labels, which no model can remove.

Bagging lowers variance by averaging, boosting mainly lowers bias, and regularisation trades a little bias for less variance. Very large neural networks show "double descent", where test error falls again past the point of fitting the training data perfectly, so the classic U-shaped curve is a guide rather than a law.

85. How do the learning rate and early stopping work together in gradient boosting?

Experienced · Trees & ensembles

Each new tree adds learning_rate × its prediction. A large learning rate fits the training data fast but overshoots and overfits after few trees; a small one takes many more trees but generalises better. Early stopping keeps adding trees only while the score on a validation set keeps improving, so you do not have to guess `n_estimators`.

PYTHON

```
from sklearn.datasets import make_classification
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split

X, y = make_classification(n_samples=3000, n_features=20, flip_y=0.1, random_state=0)
X_tr, X_te, y_tr, y_te = train_test_split(X, y, random_state=0)

for lr in [1.0, 0.1]:
    gb = GradientBoostingClassifier(learning_rate=lr, n_estimators=1000, n_iter_no_change=20,
                                    validation_fraction=0.2, random_state=0).fit(X_tr, y_tr)
    print(f"learning_rate={lr}: stopped at {gb.n_estimators_} trees, "
          f"test accuracy {gb.score(X_te, y_te):.3f}")
```

OUTPUT

```
learning_rate=1.0: stopped at 21 trees, test accuracy 0.904
learning_rate=0.1: stopped at 54 trees, test accuracy 0.936
```

With learning rate 1.0, training stopped after 21 trees at 90.4% test accuracy; with 0.1 it used 54 trees and reached 93.6%. A common recipe is a small learning rate (0.01 to 0.1), a high tree limit, and early stopping on a validation set.

86. What makes XGBoost different from classic gradient boosting?

Experienced · Trees & ensembles

- **Regularised objective:** it adds penalties on the number of leaves (`gamma`) and the leaf weights (`lambda`, `alpha`), which reduces overfitting.
- **Second-order optimisation:** it uses both the gradient and the Hessian (curvature) of the loss to choose splits and leaf values, which is more accurate per tree.
- **Sparsity-aware splits:** each split learns a default direction for missing values.
- **Speed:** histogram-based split finding, parallel tree building, cache-aware data layout, GPU support and out-of-core training.
- Row and column subsampling, and built-in cross-validation and early stopping.

87. How do LightGBM, XGBoost and CatBoost compare?

Experienced · Trees & ensembles

	XGBoost	LightGBM	CatBoost
Tree growth	Level by level (depth-wise) by default	Leaf by leaf: grows the best leaf first	Symmetric (oblivious) trees
Speed on large data	Fast	Usually fastest	Slower to train, fast to predict
Categorical features	Needs encoding (native support is newer)	Native support	Best native support, with ordered target statistics
Overfitting risk	Medium	Higher on small data; limit num_leaves	Lower; good defaults

	XGBoost	LightGBM	CatBoost
Good choice when	You want a mature, widely used default	Data is very large	Many categorical features, little tuning time

All three are close in accuracy after tuning. In practice, try whichever fits your data, and spend the saved time on features.

88. How do SHAP and LIME explain a model's predictions?

Experienced · Data & features

SHAP (SHapley Additive exPlanations) uses Shapley values from game theory: each feature gets a share of the difference between this prediction and the average prediction, and the shares add up exactly to that difference. TreeSHAP computes them exactly and quickly for tree models. You get local explanations ("why was this loan rejected?") and global ones (average absolute SHAP per feature).

LIME builds a simple, interpretable model, usually a linear one, around a single prediction by perturbing the input and watching how the output changes. It is model-agnostic and fast, but its explanations can change from run to run and depend on how the neighbourhood is defined. SHAP is more consistent and is the usual choice today; both explain the model, not the real world, so correlated features can still mislead.

89. How do you find and reduce bias (unfairness) in a model?

Experienced · ML in production

Models learn whatever patterns are in the data, including historical discrimination, and proxy features (such as pin code standing in for community or income) can bring it back even after you drop sensitive columns.

- **Measure:** compare metrics across groups: selection rate (demographic parity), true positive rate (equal opportunity), false positive rate, and calibration by group.
- **Before training:** fix sampling and labelling problems, reweight under-represented groups, remove clear proxies.
- **During training:** fairness constraints or regularisation (for example with Fairlearn).
- **After training:** group-specific thresholds where the law and policy allow.
- Document the choices, because some fairness definitions cannot all be satisfied at once.

90. How do you choose the right metric for a business problem?

Experienced · Model evaluation

Start from the decision the model supports and the cost of each kind of mistake, not from the most familiar metric. For a fraud model, a missed fraud might cost ₹20,000 and a false alarm ₹200 of review time. Then you can write the expected cost for each threshold and pick the cheapest one, instead of optimising accuracy or AUC.

- Ranking problems (which leads to call first): precision@k, recall@k, NDCG.
- Probability-based decisions: log loss, Brier score, calibration.
- Regression with big costly errors: RMSE; with outliers: MAE; for relative errors: MAPE (careful near zero).
- Keep one primary metric and a few guardrail metrics, such as latency or false positive rate.

91. What is a transformer, and why did it replace RNNs?

Experienced · GenAI & LLMs

A transformer is a neural network architecture built around **self-attention** (next question) instead of recurrence. Each layer lets every token look at every other token directly, then passes the result through a small feed-forward network, with residual connections and layer normalisation around both. Positional encodings add word-order information, since attention by itself ignores order.

- It processes all tokens in parallel, so it trains much faster on GPUs than an RNN, which must go step by step.
- Any two tokens are one step apart, so long-range dependencies are easy to learn.
- It scales very well with data and compute, which made large language models possible.

Encoder-only transformers (BERT) are used for understanding tasks like classification, decoder-only ones (GPT, Claude, Llama) for text generation, and encoder-decoder ones (T5) for translation and summarisation. The original paper is "Attention Is All You Need" (2017).

92. How does self-attention work?

Experienced · GenAI & LLMs

Each token's embedding is multiplied by three learned matrices to get a **query** (what am I looking for?), a **key** (what do I contain?) and a **value** (what do I pass on?). The attention weights are softmax(QK^T / √d), and each token's output is the weighted sum of all the values. Dividing by √d keeps the dot products from growing so large that softmax saturates.

PYTHON

```
import numpy as np

def softmax(z):
    e = np.exp(z - z.max(axis=-1, keepdims=True))
    return e / e.sum(axis=-1, keepdims=True)

rng = np.random.default_rng(0)
x = rng.normal(size=(3, 4))                  # 3 tokens, each a 4-number embedding
Wq, Wk, Wv = (rng.normal(size=(4, 4)) for _ in range(3))

Q, K, V = x @ Wq, x @ Wk, x @ Wv
weights = softmax(Q @ K.T / np.sqrt(K.shape[1]))   # how much each token looks at the others
output = weights @ V

print("Attention weights (each row sums to 1):")
print(weights.round(3))
print("Output shape:", output.shape)
```

OUTPUT

```
Attention weights (each row sums to 1):
[[0.384 0.3   0.315]
 [0.321 0.206 0.473]
 [0.053 0.008 0.939]]
Output shape: (3, 4)
```

Each row shows how much one token attends to each of the three tokens, and each row sums to 1. **Multi-head attention** runs several of these in parallel with different matrices, so different heads can track different relationships. In decoder models a causal mask stops a token from looking at future tokens. The cost grows with the square of the sequence length, which is why long context windows are expensive.

93. How are large language models trained?

Experienced · GenAI & LLMs

1. **Pre-training:** a decoder-only transformer learns to predict the next token over a huge text corpus. This self-supervised step is where most knowledge and ability come from, and it costs the most compute.
2. **Supervised fine-tuning (SFT):** the model is trained on curated examples of prompts and good responses, so it learns to follow instructions and hold a conversation.
3. **Preference tuning:** humans (or a model) compare responses, and the model is trained to prefer the better ones, with RLHF (a reward model plus reinforcement learning such as PPO) or simpler methods like DPO. This improves helpfulness and safety.

Inference is autoregressive: the model generates one token at a time, each time feeding its own output back in, using a sampling strategy (question 97). Techniques like quantisation and KV caching make this cheaper and faster.

94. When would you use prompt engineering, RAG or fine-tuning?

Experienced · GenAI & LLMs

Approach	What it changes	Use it when
Prompt engineering	Only the input: instructions, examples (few-shot), output format	Always start here; cheap and fast to iterate
RAG (retrieval-augmented generation)	Adds relevant documents to the prompt at query time	The model needs private, large or frequently changing knowledge, and answers must cite sources
Fine-tuning	The model's weights, using your examples	You need a consistent style or format, a narrow task done cheaply by a smaller model, or behaviour prompts cannot get

A common mistake is fine-tuning to add knowledge. Facts learned that way are hard to update and easy to hallucinate, so RAG is usually better for knowledge, and fine-tuning for behaviour. Many production systems combine all three.

95. How does a RAG pipeline work, and what are embeddings and vector databases for?

Experienced · GenAI & LLMs

1. **Ingest:** split documents into chunks (for example a few hundred tokens with some overlap).
2. **Embed:** turn each chunk into a vector with an embedding model, so chunks with similar meaning get similar vectors.
3. **Store** the vectors in a vector database or index (FAISS, pgvector, Pinecone, Chroma) that supports fast approximate nearest-neighbour search.
4. **Retrieve:** embed the user's question and fetch the most similar chunks, often combined with keyword search (hybrid) and a re-ranking model.
5. **Generate:** put the retrieved chunks into the prompt and ask the LLM to answer only from them, with citations.

Most RAG failures are retrieval failures, so evaluate retrieval on its own (recall@k for known question and document pairs) as well as the final answers.

96. What is hallucination, and how do you evaluate LLM outputs?

Experienced · GenAI & LLMs

A hallucination is fluent, confident output that is false or not supported by the given sources, such as an invented citation or a wrong figure. It happens because the model generates likely-sounding text, not checked facts.

- **Reduce it:** ground answers with RAG, tell the model to say "I don't know" when the context lacks the answer, ask for citations, lower the temperature for factual tasks, and use structured outputs.
- **Evaluate:** build a test set of real questions with reference answers; measure faithfulness to sources, answer correctness and relevance; use human review plus "LLM as a judge" scoring with a clear rubric, checked against human labels.
- Track these metrics over time, like any other model, because prompts, models and documents change.

97. What are tokens, temperature and top-p in LLMs?

Experienced · GenAI & LLMs

Tokens are the pieces of text a model reads and writes: whole words, parts of words or punctuation, produced by a tokenizer such as BPE. Context limits, speed and API pricing are all counted in tokens. For the next token, the model outputs a score (logit) for every token in its vocabulary.

PYTHON

```
import numpy as np

logits = np.array([2.0, 1.0, 0.5, -1.0])      # model scores for 4 possible next words
words = ["Surat", "Mumbai", "Pune", "Paris"]

for T in [0.5, 1.0, 2.0]:
    p = np.exp(logits / T) / np.exp(logits / T).sum()
    print(f"T={T}: " + ", ".join(f"{w} {q:.2f}" for w, q in zip(words, p)))
```

OUTPUT

```
T=0.5: Surat 0.84, Mumbai 0.11, Pune 0.04, Paris 0.00
T=1.0: Surat 0.61, Mumbai 0.22, Pune 0.14, Paris 0.03
T=2.0: Surat 0.43, Mumbai 0.26, Pune 0.21, Paris 0.10
```

Temperature divides the logits before softmax: below 1 makes the top choice more likely (more predictable output), above 1 flattens the distribution (more varied, more random). **Top-p** (nucleus sampling) samples only from the smallest set of tokens whose probabilities add up to p, such as 0.9, and **top-k** keeps only the k most likely tokens. Use low temperature for extraction and code, and higher for brainstorming.

98. What are LoRA and parameter-efficient fine-tuning?

Experienced · GenAI & LLMs

Fully fine-tuning a model with billions of parameters needs a lot of GPU memory and stores a full copy per task. **Parameter-efficient fine-tuning (PEFT)** freezes the original weights and trains only a small number of new ones.

LoRA (low-rank adaptation) adds two small matrices A and B next to selected weight matrices, so the update is $\Delta W = BA$ with a low rank r (such as 8 or 16). Often well under 1% of the parameters are trained, and the result is close to full fine-tuning. **QLoRA** also loads the frozen base model in 4-bit precision, so large models can be fine-tuned on a single GPU. The small adapters can be swapped per task or merged into the base weights for serving.

99. How do you handle categorical features with thousands of categories?

Experienced · Data & features

One-hot encoding a feature like user ID, product ID or pin code creates thousands of sparse columns. Better options:

- **Target encoding:** replace each category with a smoothed average of the target for it. It must be computed out of fold (inside cross-validation), or each row's own label leaks into its feature. scikit-learn's `TargetEncoder` does the cross-fitting for you.
- **Frequency or count encoding**, and grouping rare categories into "other".
- **Hashing** categories into a fixed number of buckets.
- **Learned embeddings** in a neural network.
- Models with native categorical handling: CatBoost and LightGBM.

Also plan for categories that appear only after deployment, and make sure the encoding has a sensible default for them.

100. How would you design a fraud detection system for card transactions?

Experienced · ML in production

- **Requirements:** a decision within about 100 milliseconds per transaction, extremely imbalanced labels (well under 1% fraud), labels that arrive late (chargebacks take weeks), and fraudsters who adapt.
- **Features:** transaction amount and merchant category, plus velocity features computed in real time from a feature store (number of transactions in the last hour, distance from the last location, amount compared with the card's usual spending), device and IP signals.
- **Model:** gradient-boosted trees as the main model, with business rules for known patterns, plus an anomaly detector for new kinds of fraud. Train with time-based validation.
- **Decision:** calibrated probabilities feed a cost-based policy: approve, ask for OTP verification, send to manual review, or block.
- **Evaluate and monitor:** PR AUC and recall at a fixed false positive rate, money saved, customer friction, drift, and retraining as new labels arrive. Watch for the feedback loop where blocked transactions never get labels.

In design questions, interviewers care more about this structure and the trade-offs you name than about the exact algorithm.

Practise with the interactive tools (bias-variance, metric game, gradient descent, algorithm picker) and take the quiz on the blog:
codeanddebug.in/blog/machine-learning-interview-questions