💡 Data Scientist Interview Questions at Zepto

(0-4 Years Experience) 💡

General Data Science Knowledge

1. Difference Between Supervised and Unsupervised Learning

Supervised Learning:

- In supervised learning, the model is trained on a labeled dataset, meaning each input data is paired with the correct output.
- The aim is to learn the mapping function from input to output.
- **Examples:** Linear regression, logistic regression, decision trees, and support vector machines.
- **Use Cases:** Spam detection, fraud detection, predicting house prices.

Unsupervised Learning:

- In unsupervised learning, the data is unlabeled, and the model attempts to find hidden patterns or structures in the data.
- The model does not have predefined output labels.
- **Examples:** Clustering (e.g., K-means), dimensionality reduction (e.g., PCA).
- **Use Cases:** Customer segmentation, anomaly detection.

Key Difference:
Supervised learning requires labeled data and predicts outcomes, while unsupervised learning identifies patterns in unlabeled data.

---

2. What is Overfitting in Machine Learning, and How to Prevent It?

Overfitting:

- Overfitting occurs when a machine learning model performs well on training data but poorly on unseen or test data.
- The model captures noise and random patterns rather than the underlying relationship.

How to Prevent Overfitting:

1. **Cross-validation:** Use techniques like k-fold cross-validation to ensure the model generalizes well.
2. **Regularization:** Add penalties to the model complexity (e.g., L1/L2 regularization).
3. **Pruning:** Reduce the size of decision trees or simplify the model.
4. **Reduce Features:** Eliminate irrelevant features to avoid over-complexity.
5. **Early Stopping:** Stop training the model when validation performance starts to degrade.
6. **Increase Data:** Use more data or apply data augmentation techniques.

---

3. Handling Missing or Incomplete Data

- **Steps to Handle Missing Data:**
    1. **Understand the Missingness:** Check whether the data is missing completely at random, missing at random, or not missing at random.
    2. **Remove Missing Data:** If the percentage of missing data is small, drop the rows or columns with missing values.
    3. **Imputation:**
        - Replace missing values with statistical measures like mean, median, or mode.
        - Use advanced techniques like KNN imputation or regression models to predict missing values.
    4. **Flag Missing Data:** Create an indicator variable to denote missing values for certain columns.
    5. **Domain Knowledge:** Use knowledge of the dataset to impute values or decide whether to ignore them.

---

4. Explain Cross-Validation and Its Importance

Cross-Validation:

- A statistical technique to evaluate the performance of a model by splitting data into training and validation sets multiple times.
- The most common method is **k-fold cross-validation**, where the dataset is divided into k subsets (folds). The model is trained on k-1 folds and tested on the remaining fold, and the process is repeated k times.

Why It's Important:

1. **Prevents Overfitting:** Helps ensure the model is not overly tuned to the training set.
2. **Reliable Model Performance:** Provides a more robust estimate of model performance by testing it on different subsets of data.
3. **Optimal Hyperparameters:** Assists in fine-tuning hyperparameters without relying solely on a single test set.

---

5. What are Precision, Recall, and F1-score?

- **Precision:**
  The proportion of true positive predictions out of all positive predictions.

- **Precision:**
  The proportion of true positive predictions out of all positive predictions.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

  - High precision means fewer false positives.

- **Recall (Sensitivity):**
  The proportion of true positive predictions out of all actual positive instances.

The proportion of true positive predictions out of all actual positive instances.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

  - High recall means fewer false negatives.

- **F1-Score:**
  The harmonic mean of precision and recall. It balances the trade-off between precision and recall.

The harmonic mean of precision and recall. It balances the trade-off between precision and recall.

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

  - Useful when the class distribution is imbalanced.

Interpretation Example:

- If precision is high but recall is low, the model is good at identifying positives but misses many true positives.
- A high F1-score means the model balances precision and recall effectively.

Machine Learning Algorithms

1. Explain How a Decision Tree Algorithm Works. What Are Its Strengths and Weaknesses?

How It Works:

- A Decision Tree is a flowchart-like structure used for both classification and regression tasks.
- It splits data into subsets based on feature values, using decision nodes, branches, and leaf nodes.

- The splitting process continues until a stopping criterion is met (e.g., no further splits are possible or a maximum depth is reached).
- Common metrics used for splitting are **Gini Impurity**, **Entropy** (for classification), and **Mean Squared Error** (for regression).

Steps in Building a Decision Tree:

1. Start at the root node containing the entire dataset.
2. Calculate the best feature to split data using a metric like Information Gain or Gini Index.
3. Split data into subsets based on the feature's values.
4. Repeat recursively for each subset until a stopping condition is met.

Strengths:

- **Easy to Understand:** Intuitive and visually interpretable.
- **Non-Parametric:** No assumptions about data distribution.
- **Handles Nonlinear Data:** Effectively captures complex relationships.

Weaknesses:

- **Overfitting:** Tends to overfit on the training data, especially with deep trees.
- **Instability:** Small changes in data can lead to drastically different trees.
- **Bias Towards Features with Many Levels:** Can favor features with more unique values.

---

2. What is the Difference Between Random Forest and Gradient Boosting?

Random Forest:

- An ensemble method that builds multiple Decision Trees and combines their outputs (via majority vote for classification or averaging for regression).
- Uses **bagging (Bootstrap Aggregating)** to train each tree on a random subset of the data and features.
- Reduces overfitting and increases stability.

Gradient Boosting:

- Builds trees sequentially, where each tree corrects the errors of the previous tree.
- Optimizes a loss function using gradient descent.
- Focuses on misclassified instances, making it powerful but prone to overfitting.

Key Differences:

Feature	Random Forest	Gradient Boosting
Approach	Bagging	Boosting
Tree Independence	Trees are independent	Trees are sequentially dependent
Speed	Faster training (parallelizable)	Slower due to sequential training
Overfitting	Less prone to overfitting	More prone to overfitting if not tuned
Use Cases	General-purpose	Optimized models for imbalanced/complex data

---

3. Describe the Steps You Would Take to Build a Recommendation System for Zepto's E-Commerce Platform

1. **Understand the Business Goal:**
    - Increase sales, customer engagement, or retention.
    - Focus on personalizing product recommendations.
2. **Collect and Prepare Data:**
    - Gather data like user purchase history, product categories, browsing history, and demographics.
    - Clean the data to handle missing values, duplicates, and inconsistencies.
3. **Choose a Recommendation Approach:**
    - **Collaborative Filtering:**
        - Based on user-user or item-item similarities.

        - Example: "Customers who bought X also bought Y."
    - **Content-Based Filtering:**
        - Uses product features (e.g., price, category) to recommend similar items.
    - **Hybrid Approach:**
        - Combines collaborative and content-based methods for better accuracy.
4. **Build the Model:**
    - Use libraries like surprise or scikit-learn to implement collaborative filtering models like **Matrix Factorization (SVD)**.
    - Implement deep learning models (e.g., Neural Collaborative Filtering) for large-scale datasets.
5. **Evaluation and Optimization:**
    - Use metrics like Precision@K, Recall@K, or Mean Absolute Error (MAE).
    - Tune hyperparameters for optimal performance.
6. **Deploy and Monitor:**
    - Integrate the model into Zepto's platform.
    - Continuously update recommendations as new data comes in.

---

4. What is the Difference Between Bagging and Boosting in Machine Learning?

Feature	Bagging	Boosting
Definition	Trains multiple models in parallel on random subsets of data and combines results.	Trains models sequentially, with each model focusing on errors of the previous one.
Goal	Reduce variance.	Reduce bias and variance.

Feature	Bagging	Boosting
Model Independence	Independent models.	Models are interdependent.
Examples	Random Forest.	Gradient Boosting, AdaBoost, XGBoost.
Performance	Effective for reducing overfitting.	Effective for improving accuracy on complex datasets.

Key Takeaway:

- Bagging stabilizes models and reduces overfitting.
- Boosting improves performance by focusing on difficult-to-predict instances.

---

5. Explain How K-Means Clustering Works. How Would You Determine the Optimal Number of Clusters?

How K-Means Works:

1. **Initialize Centroids:** Choose k random points as initial cluster centroids.
2. **Assign Clusters:** Assign each data point to the nearest centroid.
3. **Update Centroids:** Recalculate centroids as the mean of all points in a cluster.
4. **Repeat:** Iterate the assign-and-update process until centroids stabilize or a maximum number of iterations is reached.

Challenges:

- Sensitive to the initial choice of centroids.
- Assumes spherical clusters of similar sizes.

Determining Optimal Clusters:

1. **Elbow Method:**
    - Plot the sum of squared distances (SSE) for different values of k.
    - Look for the "elbow point," where the SSE decreases sharply and levels off.
2. **Silhouette Score:**

- Measures how similar data points in a cluster are compared to other clusters.
- Ranges from -1 to 1 (higher is better).

3. **Gap Statistic:**
   - Compares cluster performance on actual data versus random data.

Example for Zepto:
Use K-Means to group customers based on purchasing behavior (e.g., order frequency, basket size) to improve targeted marketing campaigns.

Data Processing & Feature

1. How Would You Handle Categorical Variables in a Machine Learning Model?

Categorical variables are non-numeric variables that represent categories or labels (e.g., "Male/Female," "Red/Green/Blue"). Machine learning algorithms typically require numerical inputs, so these variables must be encoded.

Techniques to Handle Categorical Variables:

1. **Label Encoding:**
   - Assigns a unique integer to each category.
   - Example: {"Male": 0, "Female": 1}.
   - **Use Case:** When the categories have a natural order (ordinal data).
   - **Limitations:** Can introduce unintended ordinal relationships for nominal data.
2. **One-Hot Encoding:**
   - Creates a new binary column for each category, assigning 1 or 0.
   - Example: {"Red": [1, 0, 0], "Green": [0, 1, 0], "Blue": [0, 0, 1]}.

- **Use Case:** Nominal data without inherent order.
- **Limitations:** Can lead to high dimensionality for datasets with many categories.

3. **Binary Encoding:**
   - Converts categories to binary numbers and encodes them as columns.
   - Example: "Red" → 001, "Green" → 010, "Blue" → 011.
   - **Use Case:** Reduces dimensionality compared to one-hot encoding.
4. **Frequency Encoding:**
   - Replaces categories with their frequency in the dataset.
   - Example: {"Male": 60%, "Female": 40%}.
   - **Use Case:** When the frequency of a category holds predictive information.
5. **Target Encoding (Mean Encoding):**
   - Replaces categories with the mean of the target variable for that category.
   - **Use Case:** Effective for high-cardinality categorical variables but prone to overfitting.

---

2. What Is Feature Scaling, and When Is It Necessary?

Feature Scaling ensures that numerical features are on a similar scale, preventing features with larger ranges from dominating the model.

Common Scaling Techniques:

1. **Standardization (Z-Score Scaling):**
   - Formula: $z=(x-\mu)/z$
   - Centers data to have a mean of 0 and a standard deviation of 1.
   - **Use Case:** Algorithms like Logistic Regression, SVM, and PCA.
2. **Min-Max Scaling (Normalization):**
   - Formula: $x'=(x-\min(x)) / (\max(x)-\min(x))$
   - Scales data between 0 and 1.

- **Use Case:** Useful for models sensitive to the scale of input features (e.g., Neural Networks).

3. **Robust Scaling:**
    - Uses the median and interquartile range (IQR) to scale.
    - **Use Case:** Effective when data contains outliers.

When Necessary:

- When features have different units (e.g., age in years, income in dollars).
- Algorithms sensitive to scale: SVM, KNN, Gradient Descent, PCA.
- Algorithms like Tree-based models (e.g., Random Forest) are less affected by scaling.

---

3. What Techniques Do You Use to Deal with Imbalanced Datasets?

An imbalanced dataset occurs when one class dominates others (e.g., 90% Class A, 10% Class B).

Techniques to Address Imbalance:

1. **Resampling Methods:**
    - **Oversampling:** Duplicate samples from the minority class (e.g., SMOTE - Synthetic Minority Oversampling Technique).
    - **Undersampling:** Remove samples from the majority class to balance the dataset.
    - **Hybrid Methods:** Combine oversampling and undersampling.
2. **Class Weight Adjustment:**
    - Modify the algorithm to penalize misclassification of the minority class more heavily.
    - Example: class_weight='balanced' in scikit-learn models.
3. **Anomaly Detection:**
    - Treat the minority class as anomalies and use anomaly detection techniques.

4. **Data Augmentation:**
    - Generate synthetic samples for the minority class using transformations or noise.
5. **Algorithmic Approaches:**
    - Use algorithms designed to handle imbalance (e.g., XGBoost, LightGBM).
6. **Evaluation Metrics:**
    - Use metrics like Precision, Recall, F1-Score, ROC-AUC instead of accuracy.

---

4. Explain the Concept of Feature Importance. How Do You Evaluate It in a Model?

Feature Importance refers to the contribution of each feature to a model's predictive power.

Techniques to Evaluate Feature Importance:

1. **Tree-based Models:**
    - Algorithms like Random Forest, XGBoost, and LightGBM provide built-in feature importance based on Gini Impurity or split gain.
2. **Permutation Importance:**
    - Shuffle a feature's values and observe the drop in model performance. A large drop indicates high importance.
3. **SHAP (SHapley Additive exPlanations):**
    - A unified framework to explain predictions by assigning importance scores to each feature.
4. **LIME (Local Interpretable Model-agnostic Explanations):**
    - Generates explanations for individual predictions by approximating the model locally.
5. **Regression Coefficients (Linear Models):**
    - In linear models, coefficients indicate feature importance. Higher absolute values mean higher importance.

6. **Feature Selection Techniques:**
    - Techniques like Recursive Feature Elimination (RFE) can rank features by importance iteratively.

---

5. How Do You Deal with Multicollinearity in a Dataset?

Multicollinearity occurs when independent variables are highly correlated, making it difficult for the model to determine their individual effects.

Steps to Address Multicollinearity:

1. **Identify Multicollinearity:**
    - Calculate the **correlation matrix** and look for high correlation values (e.g., > 0.8).
    - Compute the **Variance Inflation Factor (VIF):** VIF=1/(1−R^2). A VIF > 10 indicates severe multicollinearity.
2. **Remove One of the Correlated Features:**
    - Drop features with high VIF or high correlation to simplify the model.
3. **Combine Features:**
    - Use **Principal Component Analysis (PCA)** to combine correlated features into uncorrelated components.
4. **Regularization:**
    - Apply **Ridge Regression** (L2 regularization) or **Lasso Regression** (L1 regularization). These methods penalize high coefficients and reduce the impact of multicollinearity.
5. **Domain Knowledge:**
    - Use domain expertise to retain features that are more meaningful or important.

Programming and Tools

1. What Programming Languages Are You Comfortable With for Data Science Tasks?

For data science tasks, the most commonly used programming languages are:

1. **Python:**
    - Widely used for its rich ecosystem of libraries like Pandas, NumPy, Matplotlib, Scikit-learn, TensorFlow, and PyTorch.
    - Provides excellent support for data cleaning, manipulation, visualization, and machine learning.
    - **Popular Libraries/Frameworks:** Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn, TensorFlow, PyTorch, Statsmodels.
2. **R:**
    - Another powerful language for data analysis, especially for statistics and visualizations.
    - Known for its extensive statistical modeling and visualization capabilities.
    - **Popular Libraries:** ggplot2, dplyr, caret, tidyr, shiny.
3. **SQL (Structured Query Language):**
    - Essential for querying databases and manipulating structured data.
    - Frequently used for data extraction, transformation, and processing.
    - **Popular Tools/Databases:** MySQL, PostgreSQL, SQL Server, SQLite.
4. **Other Languages/Tools:**
    - **Julia:** Used for high-performance numerical and scientific computing.
    - **Scala:** Often used with Apache Spark for big data processing.
    - **Java/Scala (for Big Data):** Often used with Hadoop and Spark.

---

2. Explain the Difference Between NumPy Arrays and Pandas DataFrames

Both **NumPy arrays** and **Pandas DataFrames** are powerful data structures used for numerical and tabular data, but they have different purposes and capabilities.

NumPy Arrays:

- **Type:** Homogeneous (all elements must be of the same type, e.g., all integers or all floats).
- **Purpose:** Mainly used for numerical computation, especially when working with multi-dimensional arrays.
- **Operations:** Provides fast mathematical operations like element-wise operations, linear algebra, and matrix manipulations.
- **Data Handling:** Handles only numerical data efficiently and lacks advanced data manipulation features for non-numeric data.
- **Shape:** Can have 1D, 2D, or higher dimensions (e.g., vectors, matrices, tensors).

Pandas DataFrames:

- **Type:** Heterogeneous (can hold different types of data in columns, e.g., integers, floats, strings).
- **Purpose:** Mainly used for handling structured/tabular data, similar to a database table or Excel sheet.
- **Operations:** Supports indexing, grouping, merging, and handling missing data. Provides better tools for data manipulation than NumPy.
- **Data Handling:** Handles both numerical and categorical data and supports complex operations on columns like filtering, aggregation, and pivoting.
- **Shape:** Always 2D, with labeled rows and columns.

Key Difference:
While **NumPy arrays** are great for efficient numerical computations, **Pandas DataFrames** offer more advanced capabilities for handling structured and labeled data, making them ideal for data analysis and manipulation.

---

3. Describe How You Would Use Python Libraries Like Pandas, NumPy, and Matplotlib for Data Analysis and Visualization

Here's how these libraries would be used together for data analysis and visualization:

1. **Pandas (Data Manipulation & Cleaning):**

- **Loading Data:**
  df = pd.read_csv("data.csv") (Load a dataset into a DataFrame).
- **Data Cleaning:**
  - Handle missing values: df.fillna(0) or df.dropna().
  - Remove duplicates: df.drop_duplicates().
- **Data Transformation:**
  - Filter rows: df[df['Age'] > 25].
  - Group data: df.groupby('Gender').mean().
- **Merging Data:**
  - Merge datasets: df1.merge(df2, on='id').

2. **NumPy (Numerical Computations):**
   - **Array Creation & Operations:**
     - np.array([1, 2, 3]) to create an array.
     - Perform operations: np.mean(array), np.dot(a, b) (dot product), np.std(array).
   - **Matrix Operations:**
     - Matrix multiplication: np.matmul(matrix1, matrix2).
3. **Matplotlib (Visualization):**
   - **Basic Plots:**
     - Line plot: plt.plot(x, y).
     - Bar plot: plt.bar(x, y).
     - Histogram: plt.hist(data).
   - **Customization:**
     - Add labels: plt.xlabel('X Axis'), plt.ylabel('Y Axis').
     - Add titles: plt.title('Plot Title').
   - **Displaying Plots:**
     - Show plot: plt.show().

Example Workflow:

- **Data Import & Clean:** Use **Pandas** to load and clean data.
- **Compute Statistics:** Use **NumPy** for numerical operations (e.g., mean, median).
- **Plot Results:** Use **Matplotlib** to create visualizations (e.g., bar charts, histograms) to interpret the findings.

---

4. What Is the Purpose of the train_test_split() Function in scikit-learn?

The train_test_split() function in **scikit-learn** is used to split a dataset into two sets:

1. **Training Set:** A subset of the data used to train the machine learning model.
2. **Test Set:** A subset of the data used to evaluate the performance of the model on unseen data.

Purpose:

- **Prevent Overfitting:** By splitting the data, we can train the model on one portion and evaluate its performance on a separate, unseen portion to ensure it generalizes well.
- **Validation of Model:** It helps to assess the model's ability to predict accurately on new data and prevents the model from learning the noise or patterns specific to the training data.

Syntax:

```
from sklearn.model_selection import train_test_split

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

Where:

- X is the feature matrix.
- y is the target vector.
- test_size=0.2 means 20% of the data is used for testing, and the rest (80%) is used for training.

---

5. Explain the Difference Between SQL Joins and How They Are Used in Data Processing

SQL Joins are used to combine records from two or more tables based on a related column, typically a foreign key.

1. **INNER JOIN:**
    - Combines rows from both tables where there is a match in both tables.
    - **Use Case:** To fetch rows where there is a matching record in both tables.
    - **Example:**

```
SELECT * FROM orders
INNER JOIN customers ON orders.customer_id = customers.customer_id;
```

2. **LEFT JOIN (or LEFT OUTER JOIN):**
    - Returns all rows from the left table and matching rows from the right table. If there's no match, NULL values are returned for columns from the right table.
    - **Use Case:** To fetch all records from the left table and the matching ones from the right.
    - **Example:**

```
SELECT * FROM customers
LEFT JOIN orders ON customers.customer_id = orders.customer_id;
```

3. **RIGHT JOIN (or RIGHT OUTER JOIN):**
    - Similar to the LEFT JOIN but returns all rows from the right table and the matching ones from the left table.
    - **Use Case:** To fetch all records from the right table and the matching ones from the left.
    - **Example:**

```
SELECT * FROM orders
RIGHT JOIN customers ON orders.customer_id = customers.customer_id;
```

4. **FULL OUTER JOIN:**

- Returns all rows when there's a match in either table. If no match, NULL values are returned for the missing side.
- **Use Case:** To get all records from both tables, even if there's no match.
- **Example:**

```
SELECT * FROM orders
FULL OUTER JOIN customers ON orders.customer_id = customers.customer_id;
```

5. **CROSS JOIN:**
   - Returns the Cartesian product of the two tables, combining every row of the first table with every row of the second.
   - **Use Case:** Rarely used; often for generating combinations of data.
   - **Example:**

```
SELECT * FROM products
CROSS JOIN categories;
```