Advanced Viva Questions and Ideal Answers

1. Conceptual Understanding & Problem Framing

Q: How did you define the "severity" of an accident, and why did you group certain severity levels together?

A: We grouped granular severity values (like 'fatal', 'major', 'minor') into three broad categories: Low, Moderate, and High. This was done for interpretability and model stability. Grouping helped balance the class distribution and improved generalization.

Q: What are the assumptions behind predicting accident severity from static features? How does this affect real-time applicability?

A: We assume that historical static features (like road type or weather) can predict accident severity. While this works for retrospective analysis, real-time use would require live sensor data integration and edge deployments.

Q: Why did you choose classification over regression for predicting severity?

A: Severity is a categorical concept, not a continuous measure. Classification helps better in threshold-based decision systems (e.g., alerting systems). Regression would require arbitrary thresholding for interpretation.

Q: How do you validate that the features used are causally related to severity?

A: Causality is hard to prove with observational data. However, we relied on domain knowledge and correlation analysis during EDA. Ideally, feature importance and SHAP values could be used.

Q: Is there a bias in the dataset regarding over-representation of certain weather/road types?

A: Yes, the dataset has more samples under 'Clear' weather and 'City Road'. This could introduce class imbalance, handled using stratified sampling and re-weighting techniques.

2. Data Handling & Feature Engineering

Q: Did you perform any exploratory data analysis (EDA)?

A: Yes. We visualized feature distributions, checked class balance, and found strong correlations between Road Condition, Time of Day, and Accident Severity.

Q: How did you handle missing values in the dataset?

A: We used simple strategies like mean/median imputation for numeric and mode for categorical values. Columns with over 40% missing were dropped.

Advanced Viva Questions and Ideal Answers

Q: Explain how Label Encoding might affect ordinal vs nominal variables.

A: Label Encoding imposes an order. For nominal variables, One-Hot Encoding is safer. We used Label Encoding due to model compatibility, with awareness of this limitation.

Q: You used LabelEncoder on the target. What issues might arise?

A: LabelEncoder converts classes to numbers without handling imbalance. This could bias predictions towards dominant classes. We mitigated it via balanced class weights in Random Forest.

Q: What preprocessing is required to handle unseen categories in deployment?

A: We use try-except while transforming categories. Unseen inputs are mapped to a default or most frequent encoded value (e.g., 0).

3. Machine Learning Model & Evaluation

Q: Why did you choose Random Forest?

A: It handles categorical and numerical features well, is robust to outliers and noise, and provides feature importance. It also works well on small datasets.

Q: How did you tune your Random Forest model?

A: We used GridSearchCV to tune parameters like n_estimators, max_depth, and min_samples_split. We also compared with default settings.

Q: What is the interpretability of your model?

A: Feature importance from Random Forest helped us rank predictors. Top features were Road Condition, Speed Limit, and Traffic Density.

Q: How did you validate your model?

A: Used 5-fold cross-validation and stratified split to evaluate generalization. Accuracy and F1-score were main metrics.

Q: Why not use AUC-ROC or Precision/Recall?

A: These metrics are useful for imbalanced datasets. We computed all, but for multi-class, macro F1-score and confusion matrix provided more clarity.

Advanced Viva Questions and Ideal Answers

4. Web Application and Deployment

Q: How is user input handled and passed to Flask?

A: JavaScript collects form values and sends a JSON payload via a fetch API call to the Flask endpoint '/predict'. Flask parses and processes this input.

Q: How do you ensure security and input validation?

A: Client-side validation restricts formats. Backend validates type and default values if unexpected inputs are received.

Q: What if the model receives corrupted input?

A: Fallbacks ensure non-numeric data are converted to 0.0. The server-side logic handles transformation errors using try-except.

Q: Can this be scaled for smart cities?

A: Yes, with containerization (Docker) and asynchronous request handling (Gunicorn + Nginx), plus streaming input integration from traffic sensors.

Q: How do you handle multiple prediction requests?

A: Using Flask's WSGI servers (like Gunicorn), multiple requests can be handled. For production, load balancing and stateless REST APIs are ideal.

5. Data Visualization & Power BI

Q: What transformations did you apply before Power BI import?

A: Categorical labels were mapped for readability. Dates were formatted properly. GroupBy summaries were created in Python before import.

Q: Why Power BI over Tableau or matplotlib?

A: Power BI offers intuitive dashboards with drag-and-drop and allows integration with real-time pipelines via Power BI Services. Tableau was not freely available.

Q: How do your visualizations aid decision-making?

A: They show trends over time and associations like 'High severity in wet roads at night'. Authorities can plan countermeasures accordingly.

Q: How to update dashboard dynamically with model results?

A: Power BI REST API or Python scripts (with Power BI Embedded) can push predictions to the dashboard, enabling dynamic visuals.

Q: What limitations were faced in Power BI integration?

A: Power BI is not meant for direct ML prediction. Integration required exporting CSV outputs and reloading dashboards manually or via APIs.