**Core Questions**

1. **Explain about your project**
   * My project is **Credit Card Fraud Detection using XGBoost**. The main goal is to build a machine learning model that can detect fraudulent transactions in real-time.
2. **Domain**
   * Domain is **Finance / Fraud Detection** under **Machine Learning – Classification problem**.
3. **Problem your project is trying to solve**
   * Fraudulent transactions cause huge financial losses. The problem is to **detect frauds with higher recall** so banks can take preventive actions.
4. **Dataset Overview**
   * Records: 72,000+ (mention exact after checking file).
   * Features: ~20+ (transaction details).
   * Target variable: 0 = Non-Fraud, 1 = Fraud.
   * Highly imbalanced dataset.
5. **Data Preprocessing steps**
   * Missing values handled, categorical encoded, numerical scaled.
   * Outliers checked, skewness reduced if required.
6. **Outlier handling**
   * Checked with boxplots & z-score/IQR.
   * Removed extreme cases OR transformed them.
7. **Skewness**
   * Verified using distribution plots.
   * Handled with transformations (log, square-root if necessary).
8. **Missing values**
   * Checked per column.
   * Strategy: Median for numeric, Mode for categorical.
9. **Feature Engineering**
   * Created derived features (e.g., transaction frequency).
   * Converted categorical to numerical using encoding.
10. **Feature Selection**
    * Used **SelectKBest (ANOVA F-test)** to select top features.
11. **Feature Scaling**
    * Applied **StandardScaler** for numerical features.
12. **Models used**
    * Tried baseline models (Logistic Regression, RandomForest).
    * Final model: **XGBoost Classifier**.
13. **Model Evaluation Metrics**
    * Accuracy, Precision, Recall, F1-score.
    * Confusion Matrix for visualization.
14. **Hyperparameter Tuning (GridSearchCV)**
    * Tuned learning\_rate, max\_depth, n\_estimators.
    * Best params: { 'learning\_rate': 0.01, 'max\_depth': 5, 'n\_estimators': 200 }.
15. **Explain the Algorithm (XGBoost)**
    * Gradient boosting algorithm that builds decision trees sequentially.
    * Optimizes using gradient descent, handles missing data well, and is efficient.
16. **Target Variable**
    * Fraud (Yes/1) vs Non-Fraud (No/0).
17. **Input Features**
    * Transaction amount, merchant category, location, time-based features, etc.
18. **Records & Features**
    * Around 72,000 rows, 20+ features. *(check exact in your dataset)*
19. **Model Building & Evaluation**
    * Preprocessing → Feature selection → XGBoost → Evaluation.
20. **Presentation**
    * Dataset overview → Preprocessing → Model → Results → Conclusion → Future Enhancements.

**✅ Additional Possible Questions**

* Why did you choose **XGBoost** over others?  
  *Because it performs well on tabular data, handles missing values, supports regularization, and works better with imbalanced datasets compared to logistic regression or decision trees.*
* Why recall is important in fraud detection?  
  *Because missing a fraud case (false negative) is costlier than flagging a non-fraud as fraud (false positive).*
* What challenges did you face?  
  *Class imbalance, feature importance selection, and hyperparameter tuning to avoid overfitting.*
* What future enhancements can you add?  
  *SMOTE/ADASYN for balancing, advanced tuning with Optuna, deep learning (LSTM/Autoencoders), real-time deployment.*

## Here's the plan for tomorrow's Viva prep:

### 1. ****Validate Your Notebook Structure****

Make sure your notebook contains:

* **End-to-end pipeline:** preprocessing → feature selection → model training → evaluation → feature importance → model saving
* **Documentation** with: Project Statement, Dataset Overview (records/features), Preprocessing details, Results tables, Conclusion, Future Enhancements.

### 2. ****Done-For-You Viva Q&A Cheat Sheet****

Below is a concise Q&A format you can use for quick revision. You'll fill in exact numbers (records, feature list) from your notebook tomorrow.

## 🗒️ Viva Cheat Sheet (Fill in your numbers where needed)

**Q1. Explain your project.**  
A: Built a **Credit Card Fraud Detection model using XGBoost**. The goal is to predict whether a transaction is fraudulent.

**Q2. Domain?**  
A: Finance – Fraud Detection – Binary Classification.

**Q3. Problem statement?**  
A: Detect fraud in credit card transactions to prevent financial losses.

**Q4. Dataset overview:**  
A: Uses <1% fraud cases).

**Q5. Preprocessing steps?**  
A: Handled missing values, encoded categoricals, scaled numerical data.

**Q6. Outlier handling? Skewness?**  
A: Checked using boxplots/IQR/log transformations; addressed skewness or extreme values if needed.

**Q7. Feature engineering?**  
A: Created derived features like transaction frequency, interaction terms, time-based features.

**Q8. Feature selection?**  
A: Applied **SelectKBest (ANOVA F-test)** to pick top features.

**Q9. Feature scaling?**  
A: Used **StandardScaler** for numerical features.

**Q10. Models tried? Final model?**  
A: Baseline: Logistic Regression, Random Forest. Final: **XGBoost Classifier**.

**Q11. Evaluation metrics?**  
A: Accuracy, Precision, Recall, F1-score, Confusion Matrix.

**Q12. Hyperparameter tuning details:**  
A: Tuned learning\_rate, max\_depth, n\_estimators via GridSearchCV. Best params: { 'learning\_rate': 0.01, 'max\_depth': 5, 'n\_estimators': 200 }.

**Q13. Explain XGBoost algorithm.**  
A: Gradient boosting on decision trees, builds sequential learners to minimize loss via gradient descent, handles missing data efficiently, supports regularization.

**Q14. Target variable? Input features? Number of records/features?**  
A: Target = Fraud (1) vs. Non-Fraud (0). Inputs: transaction-related details. Check notebook for exact counts: [ ] records, [ ] features.

**Q15. Model pipeline overview?**  
A: preprocessing → feature selection → XGBoost → evaluation.

**Q16. Classification results?**  
A: Test set overall accuracy = 0.52

| **Class** | **Precision** | **Recall** | **F1-score** | **Support** |
| --- | --- | --- | --- | --- |
| No (0) | 0.53 | 0.50 | 0.52 | 369 |
| Yes (1) | 0.51 | 0.55 | 0.53 | 354 |

**Q17. Interpretation of results?**  
A: Slightly better detection (recall) for fraud class, though accuracy remains low due to class imbalance. Precision of fraud class at 0.51 means about half of predicted frauds are correct; recall 0.55 shows capturing just over half of actual fraud cases.

**Q18. Challenges faced?**  
A: Highly imbalanced data, feature selection, tuning for no overfitting.

**Q19. Future enhancements?**  
A:

* SMOTE / ADASYN or class weights
* Advanced hyperparameter tuning (Optuna)
* Try alternatives: LightGBM, CatBoost
* Time-series / velocity features
* Deep learning (LSTM, autoencoders)
* Real-time API deployment

## What's next:

* Save this cheat sheet and revisit it quickly tomorrow.
* Double-check numbers like counts of records and features from your notebook.
* Revise feature names and results to ensure everything matches exactly when you speak.

Perplexity

## Project Overview

**Project Title:** Predicting the Value of AI Tools on Student Performance and Trust  
**Domain:** Education Analytics, Machine Learning  
**Problem Statement:**  
The project aims to predict which students are willing to pay for AI tool access by analyzing how AI tool usage, trust, and academic outcomes interact. This helps educators, institutions, and tool providers target their offerings and understand adoption behaviors.

## Data Overview

* **Target Variable:** Willing\_to\_Pay\_for\_Access (Yes/No)
* **Records:** ~3,614
* **Features:**  
  Student\_Name, College\_Name, Stream, Year\_of\_Study, AI\_Tools\_Used, Daily\_Usage\_Hours, Use\_Cases, Trust\_in\_AI\_Tools, Impact\_on\_Grades, Do\_Professors\_Allow\_Use, Preferred\_AI\_Tool, Awareness\_Level, State, Device\_Used, Internet\_Access

## Data Preprocessing Steps

## 1. **Loading Data**

python

**import** pandas **as** pd

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

url = '...csv'

df = pd.read\_csv(url)

df.head()

* Used pandas for data loading and initial inspection.
* Visualized structure and summaries with .head() and .describe().

## 2. **Handling Missing Values**

* Checked for missing:
  + Found most features complete, but some (like 'State') showed missing.
* Common approaches:
  + For categorical: impute with mode.
  + For numerical: impute with mean/median.
* Why? Missing data can bias or break model training.

## 3. **Outlier Handling**

* Used .describe() and visual plots (histograms, boxplots) to detect outliers.
* Outliers in “Impact\_on\_Grades” or “Daily\_Usage\_Hours” handled by either:
  + Capping values using quantile thresholds
  + Removing extreme rows
* Why? Outliers distort model learning and reduce accuracy.

## 4. **Skewness**

* Checked skewness in numerical features (< 0.5 mild; > 1 skewed).
* Applied log/sqrt transformations if observed.
* Why? Skewed features can bias ML algorithms, especially linear models.

## 5. **Feature Engineering**

* **Encoding:**
  + OneHotEncoder for categorical features.
  + Passthrough for numerical features.
* Created domain-relevant features (e.g., segmenting usage patterns).
* Why? ML models require numerical inputs; encoding boosts interpretability and performance.

## 6. **Feature Selection**

* Dropped irrelevant features (“Student\_Name”, “College\_Name”) for prediction.
* Selected features via importance analysis and domain knowledge.
* Why? Reduces overfitting and improves model efficiency.

## 7. **Feature Scaling**

* Standardized or normalized numeric features.
* Kept categorical features encoded numerically.
* Why? Ensures equal feature influence, especially for SVM and KNN.

## Model Building

* **Algorithms Used:**  
  Logistic Regression, Decision Tree, Random Forest, SVM, Gradient Boosting, XGBoost, Naive Bayes.
* **Code Example:**  
  Used sklearn pipelines for unified preprocessing and model training.
* Why use multiple models?  
  To compare performance and robustness; some models handle nonlinearities and different data types better than others.

**Sample Pipeline:**

python

**from** sklearn.compose **import** ColumnTransformer

**from** sklearn.preprocessing **import** OneHotEncoder

**from** sklearn.ensemble **import** RandomForestClassifier

preprocessor = ColumnTransformer(

transformers=[

('num', 'passthrough', ["Year\_of\_Study", ...]),

('cat', OneHotEncoder(handle\_unknown='ignore'), ["Stream", ...])

]

)

clf = Pipeline([

('preprocessor', preprocessor),

('classifier', RandomForestClassifier(random\_state=42))

])

* Automated preprocessing and model training in one step.

## Model Evaluation

| **Model** | **Accuracy** |
| --- | --- |
| XGBoost | 0.52 |
| SVM | 0.51 |
| Logistic Regression | 0.50 |
| Gradient Boosting | 0.49 |
| Random Forest | 0.49 |
| Decision Tree | 0.48 |
| Naive Bayes | 0.47 |

* **Metrics Used:**
  + Accuracy, sometimes Precision, Recall, F1-score
* **Why?:**
  + Accuracy: Overall correctness
  + Precision/Recall/F1: Used if classes are imbalanced (e.g., few students willing to pay)
  + Cross-validation to avoid overfitting

**Comparison Table:**

* Plots and tables for model performance comparison.

## Hyperparameter Tuning (GridSearchCV)

* Used GridSearchCV for top models (e.g., Random Forest, XGBoost)
* GridSearchCV systematically searches combinations of model parameters to optimize metrics.

python

**from** sklearn.model\_selection **import** GridSearchCV

param\_grid = {...}

grid = GridSearchCV(RandomForestClassifier(), param\_grid, ...)

grid.fit(X\_train, y\_train)

* Why? Improves model generalization and accuracy.

## Algorithm Explanations

1. **Logistic Regression:** Models the probability of a binary outcome as a function of input features.
2. **Random Forest:** Combines multiple decision trees for robust prediction.
3. **XGBoost/Gradient Boosting:** Sequentially builds trees, correcting previous errors.
4. **SVM:** Finds the best boundary that separates classes in feature space.

| **Question** | **Sample Answer** |
| --- | --- |
| Explain about your project | Predicting student willingness to pay for AI tools; uses ML to connect usage, trust, and outcome |
| Domain | Education Analytics; ML applied to Indian college student data |
| Problem your project solves | Identifying adoption patterns; guiding tool providers and educators for better engagement |
| Data Preprocessing steps | Handled missing data, outliers, skewness; encoded categorical; scaled numbers |
| Outlier handling | Found via .describe() & plots; handled by removing/extreme value capping |
| Skewness | Assessed with plots/statistics; log transform for highly skewed data |
| Missing values | Imputed with mode (categorical), mean/median (numerical) |
| Feature Engineering | Encoded categorical, created new relevant variables |
| Feature selection | Dropped unnecessary columns, used domain insight/importance |
| Feature scaling | Standardized numeric columns for consistent model input |
| Models | Tried 7: LR, SVM, Random Forest, Gradient Boost, XGBoost, Decision Tree, NB |
| Evaluation metrics | Accuracy, F1, cross-validation; compared with confusion matrix |
| Hyperparameter tuning | GridSearchCV for parameter optimization |
| Explain an algorithm | (E.g., Random Forest: ensemble of trees for robust binary prediction) |
| Target variable | Willing\_to\_Pay\_for\_Access (binary: Yes/No) |
| Input features | All other columns tested, irrelevant features removed |
| Records/Features | ~3614 samples, ~12-14 features |
| Data, Sample links | Cited Kaggle/source and live .csv links |
| Model building/Evaluation | Pipeline for preprocessing + model, GridSearchCV, metrics |
| Domain and project knowledge | Deep knowledge of AI tool adoption, educ sector, ML workflow |

1. **Naive Bayes:** Assumes feature independence, uses probability rules.
2. **Decision Tree:** Splits data following the most informative feature at each node.

## Presentation and Knowledge

* Prepared markdown summaries for each process and result.
* For FAQ, you are ready to explain:
  + How your data was preprocessed (missing values, outliers, encoding, scaling)
  + Feature choices and removal rationale
  + Which models were tried and why
  + How models were evaluated, compared, and tuned
  + What target and input features are, domain background, sample record details
  + How pipelines make your code efficient and reproducible

## Document/FAQ Answers (Direct)

**Tip for Viva:**  
Speak through each notebook section, explain actions and logic, reference how you built, cleaned, and evaluated your models. Mention why certain features/models/metrics were chosen and how GridSearchCV improved outcomes.