**Week 1 – EDA & Data Understanding**

**Cell 1: Import Libraries**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

* **What it does**: Imports core libraries for data handling, analysis, and visualization.
* **Why used**:
  + pandas: For handling tabular data.
  + numpy: For numerical operations.
  + matplotlib & seaborn: For plotting and visualizing trends.

✅ **Possible Questions & Answers**

* Q: Why Pandas instead of Excel?  
  A: Pandas is faster, handles larger datasets, automates cleaning, and integrates with ML workflows.
* Q: Difference between Matplotlib and Seaborn?  
  A: Matplotlib is low-level and customizable; Seaborn is built on top of it and provides cleaner, statistical plots.

**Cell 2: Load Dataset**

df = pd.read\_csv("file.csv")

df.head()

* **What it does**: Loads dataset into DataFrame and shows first 5 rows.
* **Why used**: To confirm successful loading and get a quick look at dataset structure.

✅ **Possible Q&A**

* Q: How many records and features are in your dataset?  
  A: We can check using df.shape. Example: 2000 rows × 20 columns.
* Q: Why head() and not tail()?  
  A: head() is usually used to preview starting rows, but tail() can also check last rows to ensure consistency.

**Cell 3: Dataset Info**

df.shape

df.info()

df.describe()

* **What it does**:
  + shape: Number of rows × columns.
  + info(): Data types & missing values.
  + describe(): Summary statistics (mean, std, min, max).

✅ **Possible Q&A**

* Q: What datatypes are present?  
  A: Mostly numerical (int, float) and categorical (object).
* Q: What insights from describe()?  
  A: Mean, median, range of values, detection of skewed distributions or extreme values.

**Cell 4: Missing Values Check**

df.isnull().sum()

* **What it does**: Shows count of missing values column-wise.
* **Why used**: Missing data must be handled before modeling.

✅ **Possible Q&A**

* Q: Did your dataset have missing values?  
  A: Yes, we identified missing values in some columns.
* Q: How did you handle missing values?  
  A: We used techniques like filling with mean/median/mode, or dropping if very high percentage.

**Cell 5: Target Distribution**

sns.countplot(x='target', data=df)

* **What it does**: Visualizes class distribution of target variable.
* **Why used**: To check balance of Yes/No (or 0/1).

✅ **Possible Q&A**

* Q: Is your dataset balanced?  
  A: Slightly imbalanced (example: 55% No, 45% Yes).
* Q: Why does imbalance matter?  
  A: Accuracy may be misleading; recall/precision are more important in such cases.

**Cell 6: Numerical Distributions**

df.hist(figsize=(10,8))

plt.show()

* **What it does**: Plots histograms for all numeric columns.
* **Why used**: To detect skewness and value distribution.

✅ **Possible Q&A**

* Q: What is skewness?  
  A: Skewness shows asymmetry in distribution. Positive skew means long right tail, negative skew means long left tail.
* Q: Did you find skewness?  
  A: Yes, some features were right-skewed; we planned transformations later.

**Cell 7: Outlier Detection**

sns.boxplot(x='feature', data=df)

* **What it does**: Plots boxplot to identify outliers.
* **Why used**: Outliers affect mean-based models and scaling.

✅ **Possible Q&A**

* Q: What is an outlier?  
  A: A value much higher or lower than the rest, outside 1.5×IQR.
* Q: How did you handle outliers?  
  A: We capped extreme values using IQR or transformed them.
* Q: Why not remove all outliers?  
  A: Some outliers may carry important information (e.g., high transaction in fraud dataset).

**Week 2 – Data Preprocessing**

**Cell 1: Handling Missing Values**

df.isnull().sum()

df['column\_name'].fillna(df['column\_name'].median(), inplace=True)

* **What it does**:
  + Checks missing values.
  + Replaces nulls with **median** (robust against outliers).
* **Why used**: Missing values break models; imputation prevents data loss.

✅ **Possible Q&A**

* Q: Did your dataset have missing values?  
  A: Yes, some columns had missing values.
* Q: Why median instead of mean?  
  A: Median is more robust when data is skewed or has outliers.
* Q: What if too many missing values?  
  A: Drop column/row if >40% missing, else impute.

**Cell 2: Outlier Treatment (IQR method)**

Q1 = df['feature'].quantile(0.25)

Q3 = df['feature'].quantile(0.75)

IQR = Q3 - Q1

lower = Q1 - 1.5 \* IQR

upper = Q3 + 1.5 \* IQR

df['feature'] = np.where(df['feature'] > upper, upper,

np.where(df['feature'] < lower, lower, df['feature']))

* **What it does**: Detects outliers using **Interquartile Range** and caps them to lower/upper limits.
* **Why used**: Prevents extreme values from skewing results.

✅ **Possible Q&A**

* Q: What is an outlier?  
  A: A value that lies outside the normal range of data (beyond 1.5×IQR).
* Q: How did you handle outliers?  
  A: We capped them within the IQR range instead of dropping.
* Q: Why not remove?  
  A: Removing may lose valid but extreme information.

**Cell 3: Skewness Handling**

df['feature'] = np.log1p(df['feature'])

* **What it does**: Applies **log transformation** to reduce skewness.
* **Why used**: Models assume normality; log helps to normalize.

✅ **Possible Q&A**

* Q: What is skewness?  
  A: A measure of asymmetry in data distribution.
* Q: Why reduce skewness?  
  A: To improve model performance and make scaling more effective.
* Q: Why log1p instead of log?  
  A: log1p safely handles zero values (log(0) is undefined).

**Cell 4: Encoding Categorical Features**

df = pd.get\_dummies(df, drop\_first=True)

* **What it does**: Converts categorical columns into binary columns (dummy variables).
* **Why used**: ML models only understand numeric data.

✅ **Possible Q&A**

* Q: What is One-Hot Encoding?  
  A: Converts categorical values into separate columns with 0/1.
* Q: Why drop\_first=True?  
  A: To avoid **dummy variable trap** (multicollinearity).
* Q: Any other methods?  
  A: Label Encoding, Target Encoding, Ordinal Encoding.

**Cell 5: Feature Engineering**

df['new\_feature'] = df['feature1'] / df['feature2']

* **What it does**: Creates new features based on existing ones.
* **Why used**: Improves model’s predictive power by providing meaningful combinations.

✅ **Possible Q&A**

* Q: What is feature engineering?  
  A: Creating or modifying features to improve learning.
* Q: Can you give an example from your dataset?  
  A: Yes, e.g., ratio of study\_hours to assignments submitted.
* Q: Why is it important?  
  A: Helps model capture hidden patterns.

**Cell 6: Train-Test Split**

from sklearn.model\_selection import train\_test\_split

X = df.drop('target', axis=1)

y = df['target']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, stratify=y, random\_state=42)

* **What it does**: Splits dataset into training (80%) and testing (20%).
* **Why used**: Ensures model generalizes on unseen data.

✅ **Possible Q&A**

* Q: Why train-test split?  
  A: To evaluate model performance on unseen data.
* Q: Why stratify?  
  A: To maintain same class ratio in train/test as original dataset.
* Q: Why random\_state?  
  A: For reproducibility of results.

**Week 3 – Feature Selection & Scaling**

**Cell 1: Feature Scaling**

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

* **What it does**:
  + Standardizes features → mean = 0, std = 1.
  + Applies scaling on training set and reuses parameters for test set.
* **Why used**: ML models like Logistic Regression, SVM, KNN are sensitive to scale.

✅ **Possible Q&A**

* Q: Why scale features?  
  A: To give equal weight to all features, avoid bias toward larger numerical values.
* Q: Why fit on train and transform on test?  
  A: Prevents **data leakage** (test info leaking into training).
* Q: Why StandardScaler and not MinMaxScaler?  
  A: StandardScaler works better with normally distributed data; MinMax is used for bounded values (0–1).

**Cell 2: Feature Selection – Filter Method**

from sklearn.feature\_selection import SelectKBest, f\_classif

selector = SelectKBest(score\_func=f\_classif, k=10)

X\_train\_selected = selector.fit\_transform(X\_train\_scaled, y\_train)

X\_test\_selected = selector.transform(X\_test\_scaled)

* **What it does**:
  + Uses ANOVA F-test (f\_classif) to select **top k features** most correlated with target.
  + Keeps 10 features here.
* **Why used**: To remove irrelevant features, reduce overfitting, improve performance.

✅ **Possible Q&A**

* Q: Why feature selection?  
  A: Removes noise, speeds training, reduces overfitting.
* Q: What method did you use?  
  A: Filter method – ANOVA F-test via SelectKBest.
* Q: Why ANOVA F-test?  
  A: It measures dependency between input variables and categorical target.
* Q: How many features did you keep?  
  A: 10 features (chosen after experimenting).

**Cell 3: Pipeline Setup**

from sklearn.pipeline import Pipeline

from sklearn.linear\_model import LogisticRegression

pipeline = Pipeline([

('scaler', StandardScaler()),

('selector', SelectKBest(score\_func=f\_classif, k=10)),

('model', LogisticRegression())

])

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

* **What it does**:
  + Combines preprocessing + feature selection + model into one pipeline.
  + Runs end-to-end without manual steps.
* **Why used**: Prevents leakage, ensures consistent transformations, cleaner workflow.

✅ **Possible Q&A**

* Q: What is a pipeline in ML?  
  A: A sequence of data preprocessing and modeling steps combined into one.
* Q: Why is pipeline better?  
  A: Automates workflow, prevents leakage, makes code cleaner.
* Q: Which steps did your pipeline include?  
  A: Scaling → Feature Selection → Logistic Regression.

**Cell 4: Model Comparison (Basic Models)**

from sklearn.metrics import accuracy\_score

from sklearn.ensemble import RandomForestClassifier

from xgboost import XGBClassifier

models = {

'Logistic Regression': LogisticRegression(),

'Random Forest': RandomForestClassifier(),

'XGBoost': XGBClassifier()

}

for name, model in models.items():

model.fit(X\_train\_selected, y\_train)

y\_pred = model.predict(X\_test\_selected)

print(name, accuracy\_score(y\_test, y\_pred))

* **What it does**:
  + Tests multiple models on same preprocessed data.
  + Compares accuracy scores.
* **Why used**: To identify best-performing algorithm for final tuning.

✅ **Possible Q&A**

* Q: Why try multiple models?  
  A: To compare and select the one with best performance for the dataset.
* Q: Which models did you try?  
  A: Logistic Regression, Random Forest, and XGBoost.
* Q: Why Random Forest?  
  A: Good for handling non-linear data, robust, avoids overfitting.
* Q: Why XGBoost?  
  A: Gradient boosting algorithm, performs well on imbalanced datasets.

**Week 4 – Modeling, Hyperparameter Tuning & Evaluation**

**Cell 1: Define Models and Pipeline**

from sklearn.pipeline import Pipeline

from sklearn.ensemble import RandomForestClassifier

from xgboost import XGBClassifier

from sklearn.linear\_model import LogisticRegression

* **What it does**: Imports candidate models (Logistic Regression, RF, XGBoost).
* **Why used**: Compare linear vs ensemble vs boosting algorithms.

✅ **Possible Q&A**

* Q: Which models did you finally use?  
  A: Logistic Regression (baseline), Random Forest (ensemble), XGBoost (boosting).
* Q: Why these models?  
  A: They represent different approaches — linear, bagging, boosting — allowing fair comparison.

**Cell 2: Hyperparameter Tuning**

from sklearn.model\_selection import GridSearchCV

param\_grid = {

'learning\_rate': [0.01, 0.1],

'max\_depth': [3, 5],

'n\_estimators': [100, 200]

}

grid = GridSearchCV(XGBClassifier(), param\_grid, cv=5, scoring='accuracy')

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

print("Best Parameters:", grid.best\_params\_)

print("Best CV Accuracy:", grid.best\_score\_)

* **What it does**:
  + Uses **GridSearchCV** to try combinations of hyperparameters.
  + Selects the best one via cross-validation.
* **Why used**: Default parameters are suboptimal; tuning finds the best configuration.

✅ **Possible Q&A**

* Q: What is hyperparameter tuning?  
  A: Process of finding best model settings that improve performance.
* Q: Why GridSearchCV?  
  A: Tests all parameter combinations with cross-validation, reliable but computationally heavy.
* Q: What were the best parameters?  
  A: Example: {'learning\_rate': 0.01, 'max\_depth': 5, 'n\_estimators': 200}.
* Q: Why 5-fold CV?  
  A: Balances bias and variance, avoids overfitting.

**Cell 3: Train Final Model with Best Parameters**

best\_model = grid.best\_estimator\_

best\_model.fit(X\_train\_selected, y\_train)

* **What it does**: Trains XGBoost with tuned hyperparameters.
* **Why used**: Ensures final model uses optimized settings.

✅ **Possible Q&A**

* Q: Why choose XGBoost as final?  
  A: It gave the most balanced precision/recall compared to Logistic Regression and Random Forest.
* Q: What is boosting in simple terms?  
  A: Boosting builds models sequentially, each fixing the errors of the previous one.

**Cell 4: Predictions & Evaluation**

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

y\_pred = best\_model.predict(X\_test\_selected)

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

sns.heatmap(confusion\_matrix(y\_test, y\_pred), annot=True, fmt="d")

* **What it does**:
  + Evaluates using accuracy, precision, recall, f1-score.
  + Plots confusion matrix.
* **Why used**: Accuracy alone is misleading in imbalanced datasets.

✅ **Possible Q&A**

* Q: What was your final accuracy?  
  A: Around **52%**.
* Q: Why use F1-score in addition to accuracy?  
  A: It balances precision and recall, especially important for imbalanced data.
* Q: Which metric was most important here?  
  A: Recall, because detecting minority class (Yes) was crucial.

**Cell 5: Feature Importance**

importances = best\_model.feature\_importances\_

* **What it does**: Shows most important features for XGBoost.
* **Why used**: Helps interpretability — knowing what drives predictions.

✅ **Possible Q&A**

* Q: Why check feature importance?  
  A: To understand which features impact predictions most.
* Q: Which feature was most important?  
  A: Example: study\_hours, assignment\_score.

**Cell 6: Save the Model**

import joblib

joblib.dump(best\_model, "final\_model.pkl")

* **What it does**: Saves trained model for reuse.
* **Why used**: For deployment or future predictions without retraining.

✅ **Possible Q&A**

* Q: How do you deploy a model?  
  A: Save using joblib/pickle, then load in an API or app.
* Q: Why save model?  
  A: Saves time; no need to retrain every time.

**Cell 7: Future Enhancements (Markdown)**

* Use SMOTE for imbalance.
* Try LightGBM/CatBoost.
* Use deep learning for non-linear patterns.
* Deploy with Flask/Django API.

✅ **Possible Q&A**

* Q: What are the limitations of your model?  
  A: Low accuracy (~52%), dataset imbalance, limited features.
* Q: How can you improve?  
  A: Use SMOTE, advanced models, feature engineering, larger dataset.

File\_link: https://colab.research.google.com/drive/18RH6UExlRM6M2oiWjRTReEczf0Z3BVIe?usp=sharing