CODE DOCUMENTATION

1) Code Structure

The Jupyter notebook is organized into the following logical components:

1. **Environment Setup**
   Configuration of Hadoop and Spark environments for distributed processing.

2. **Data Loading**
   Reading the dataset from **HDFS** using PySpark into a Spark DataFrame.

3. **Exploratory Analysis**
   Initial class distribution check to identify data imbalance.

4. **Preprocessing**

   - Encoding categorical variables using `StringIndexer`.
   - Feature assembly using `VectorAssembler`.
   - Conversion to a consistent numerical feature space.

5. **Data Balancing**

   - Converted to Pandas DataFrame.
   - Applied **SMOTE** to handle class imbalance.
   - Reconverted to Spark DataFrame with the new balanced samples.

6. **Model Training and Evaluation**
   Implementation of various ML models like Logistic Regression, KNN, Decision Tree, Random Forest, Gradient Boosting, MLP, and accuracy evaluation.

2) Modules Used

Library / Module	Purpose
`os`, `findspark`	To set up and initialize the Spark environment
`pyspark.sql`, `pyspark.ml`	For DataFrame operations, preprocessing, feature engineering
`pandas`, `numpy`	Data conversion for SMOTE and basic transformations
`imblearn.over_sampling.SMOTE`	To balance the dataset by oversampling the minority class
`sklearn.model_selection.train_test_split`	To split the balanced data into training and testing sets
`sklearn.linear_model`	For Logistic Regression and Linear Regression
`sklearn.neighbors`	For K-Nearest Neighbors (KNN) classifier
`sklearn.tree`	For Decision Tree Classifier
`sklearn.ensemble`	For Random Forest and Gradient Boosting Classifiers
`sklearn.neural_network`	For training a Multi-Layer Perceptron (MLP) model
`sklearn.metrics`	To evaluate the performance of all models using accuracy score

3) Execution Steps

1. **Spark Initialization**
   Environment variables set → Spark session initialized using `SparkSession.builder()`.

2. **Data Loading**
   CSV data loaded from **HDFS** using Spark and previewed with `.show()`.

3. **Label and Feature Engineering**

   - Used `StringIndexer` for categorical to numerical conversion.
   - Used `VectorAssembler` to bundle features into a single vector column.

4. **Class Imbalance Handling**

   - Converted Spark DataFrame to Pandas.
   - Applied `SMOTE` to generate synthetic samples for minority class.
   - Converted back to Spark for further processing.

5. **Train-Test Split**

   - Split the balanced data into 80% training and 20% testing using `train_test_split`.

6. **Model Training and Evaluation**

   - Trained each classifier (Logistic Regression, KNN, DT, RF, GBC, MLP) using scikit-learn.
   - Evaluated model accuracy on the test data using `accuracy_score`.

4) Dependencies and Environment Setup

Component	Version / Details
Python	3.x (configured in environment variables as `/usr/bin/python3`)
Apache Spark	3.5.5 (`SPARK_HOME` specified)
Apache Hadoop	Installed locally; HDFS used for reading datasets
findspark	Required for initializing Spark in Python/Jupyter
imbalanced-learn	Used for `SMOTE` algorithm
pandas, numpy	Used for data transformation, particularly for SMOTE handling
scikit-learn (sklearn)	Used for training and evaluating ML models
Jupyter Notebook	Used for interactive analysis and development