1. Preprocess
   1. Checking Null:

In the preprocessing phase, we systematically check for null values across all columns in the dataset to ensure data integrity. Using PySpark's count and isNull functions, we identify missing values in features such as Age, Flight Distance, and service ratings (e.g., Inflight Wifi Service). The provided notebook fills null values with zeros for numerical columns, which we adopt here for simplicity, using the fillna method. This approach assumes missing values can be reasonably approximated as zero, particularly for service ratings (0–5 scale) or delays, though in practice, we could explore median imputation or predictive methods for more robustness.

* 1. Checking outlier:

Outlier detection is crucial to prevent skewed model performance, especially for numerical features like Flight Distance and Departure/Arrival Delay in Minutes. We implement a basic outlier check using the interquartile range (IQR) method, calculating Q1 (25th percentile) and Q3 (75th percentile) for each numerical column and flagging values outside the range [Q1 - 1.5IQR, Q3 + 1.5IQR]. While the provided notebook does not explicitly handle outliers, we incorporate RobustScaler in the pipeline (as specified) to mitigate their impact by scaling features based on median and IQR, which is less sensitive to extreme values than standard scaling.

1. Build Pipeline:
   1. Function ScaledFeatureExpander with the Robust Scaler:

We define a custom ScaledFeatureExpander transformer that extends PySpark's Transformer class. This transformer combines feature assembly (using VectorAssembler) and scaling (using RobustScaler) into a single step. RobustScaler normalizes numerical features by subtracting the median and dividing by the IQR, making it robust to outliers. The transformer takes input columns, assembles them into a vector, and applies scaling, outputting a scaled feature vector.

* 1. Drop unecessary columns:

To streamline the dataset, we drop unnecessary columns like \_c0 (index) and id (identifier), as they do not contribute to predictive modeling. This is done using PySpark's drop method before feeding the data into the pipeline, ensuring only relevant features are processed.

* 1. Split columns into Features: target\_variable, numeric\_variables, categorical\_variables\_str and categorical\_variables\_int:

We categorize the dataset's columns into:

Target Variable: satisfaction (binary: "satisfied" or "neutral or dissatisfied").

Numeric Variables: Continuous features like Age, Flight Distance, Departure Delay in Minutes, Arrival Delay in Minutes, and service ratings (e.g., Inflight wifi service, Seat comfort) on a 0–5 scale.

Categorical Variables (String): Gender, Customer Type, Type of Travel, Class.

Categorical Variables (Integer): None in this dataset, as all categorical variables are string-based.

* 1. 5 stages in the Pipeline:

The pipeline consists of five stages:

1. StringIndexer (Categorical Variables): Converts string-based categorical columns (Gender, Customer Type, Type of Travel, Class) into numerical indices.
2. StringIndexer (Target Variable): Encodes the target variable satisfaction into a binary label column (0 for "neutral or dissatisfied", 1 for "satisfied").
3. ScaledFeatureExpander: Assembles all features (numerical and indexed categorical) into a vector and applies RobustScaler.
4. Classifier: Placeholder for one of the four machine learning models (Logistic Regression, Random Forest, Gradient Boosted Trees, or Multi-Layer Perceptron).
5. Pipeline Integration: Combines all stages into a cohesive workflow, ensuring consistent preprocessing and modeling.
6. Train Models:
   1. Logistics Regression:

Logistic Regression serves as a baseline model, modeling the linear relationship between features and the binary satisfaction outcome. It is interpretable, making it ideal for understanding feature importance.

* 1. Random Forest:

Random Forest, an ensemble method, builds multiple decision trees to capture complex feature interactions and reduce overfitting, suitable for the diverse feature set including service ratings and demographic variables.

* 1. Gradient Boost Tree:

Gradient Boosted Trees sequentially build trees to correct errors from previous iterations, optimizing for predictive accuracy, particularly effective for imbalanced or noisy data.

* 1. Multi-Layer Perceptron:

The Multi-Layer Perceptron, a neural network model, captures non-linear patterns through multiple hidden layers (e.g., [input\_size, 64, 32, 2]), offering flexibility for complex relationships in the dataset.

Each model is integrated into the pipeline and evaluated using 5-fold cross-validation with a BinaryClassificationEvaluator measuring the area under the ROC curve (AUC). This ensures robust performance assessment and model selection.

In the preprocessing phase of the airline passenger satisfaction project, handling null values and outliers is a foundational step to ensure high-quality data for modeling. We begin by inspecting the dataset for missing values across all features, such as Age, Flight Distance, and service ratings like Inflight Wifi Service. The provided code fills missing numerical values with zeros using the fillna method in PySpark, a simple imputation strategy to maintain dataset integrity. This approach assumes that missing values can be reasonably approximated as zero, though in a more robust implementation, we could explore alternatives like median imputation or predictive filling based on feature distributions. For outliers, the original code does not explicitly address them, but we could enhance preprocessing by applying techniques like RobustScaler to minimize the impact of extreme values or by filtering records beyond a certain threshold (e.g., using interquartile range for Flight Distance or delays). These steps ensure the dataset is clean and suitable for downstream machine learning tasks, preventing skewed model performance due to incomplete or anomalous data.

The construction of the PySpark pipeline is a critical step to streamline and automate feature engineering and preprocessing. The pipeline begins with StringIndexer stages to convert categorical columns—such as Gender, Customer Type, Type of Travel, and Class—into numerical indices, ensuring compatibility with machine learning algorithms. These indices are stored with mappings for later use in the Streamlit application. Next, a VectorAssembler combines all relevant features, including the indexed categorical variables and numerical columns like Age, Flight Distance, and service ratings, into a single feature vector stored in a column named "features." To handle potential scale differences, a StandardScaler could be added to normalize numerical features, though this was not explicitly shown in the provided code. The pipeline encapsulates these transformations, ensuring consistency between training and testing datasets and enabling seamless integration with model training. This modular approach enhances scalability and reproducibility, allowing easy modifications or extensions, such as adding imputation or outlier handling steps.

For model development, we implement four distinct machine learning models to predict passenger satisfaction: Logistic Regression, Random Forest, Gradient Boosted Trees, and Multi-Layer Perceptron. Logistic Regression serves as a baseline, offering interpretability by modeling the linear relationship between features and the binary satisfaction outcome (satisfied vs. neutral/dissatisfied). Random Forest, an ensemble method, leverages multiple decision trees to capture complex feature interactions and improve robustness against overfitting, making it suitable for diverse feature sets like service ratings. Gradient Boosted Trees further enhance predictive power by sequentially building trees that correct errors from previous ones, optimizing for accuracy on imbalanced or noisy data. Finally, the Multi-Layer Perceptron, a neural network model, is employed to capture non-linear patterns through multiple hidden layers, offering flexibility for complex relationships in the dataset. Each model is integrated into the pipeline and evaluated using cross-validation with a BinaryClassificationEvaluator (measuring area under ROC), allowing us to compare their performance and select the best model for deployment in the Streamlit application.