BIG DATA TECHNOLOGY

AIRLINE PASSENGER SATISFACTION

INTRODUCTION: PROJECT OBJECTIVE

Building classification models to predict airline passengers' satisfaction by analyzing factors such as passerger personal information, service quality, flight punctuality, and comfort.

Using machine learning algorithms to help airlines improve customer experience by identifying key drivers of satisfaction and dissatisfaction.

TABLE OF CONTENT

1. USER DEFINE FUNCTIONS

1. DESCRIBE ROWS AND COLUMNS (SHAPE)

```
# Function to get number rows for each column
def count_rowncol(df):
  print(type(df), "\nNumber of Rows: ", df.count(), "\nNumber of Columns: ", len
  (df.columns))
```

2. GET NULL VALUES OF EACH VARIABLE

```
# Function to get null values for each column
def count_null(df):
  null_counts = df.select([
      spark_sum(when(col(c).isNull() | (col(c) == ""), 1).otherwise(0)).alias(c)
      for c in df.columns
      ])
  null_counts.show(truncate=False)
```

3. GET DISTINCT VALUES FOR EACH VARIABLE

```
# Function to get unique values for each column
def unique_values(df):
    unique_dict = {}
    for col_name in df.columns:
        unique = df.select(collect_set(col_name).alias("unique_values")).collect
        ()
        unique_dict[col_name] = unique[0]["unique_values"]
    return unique_dict
```

4. SORT VARIABLE DATA TYPES

```
# Sort the type of variables
def variable_type(df):
    str_variables = []
    num_variables = []
    for col in df.columns:
        if isinstance(df.schema[col].dataType, StringType) and col !=
        "satisfaction":
            str_variables.append(col)
        elif isinstance(df.schema[col].dataType, NumericType):
            num_variables.append(col)
    return str_variables, num_variables

variable_type(train)
```

5. CHECK FOR OUTLIERS IN NUMERIC VARIABLES

```
# Function to detect and replace outliers with the mean using IQR
def check_outlier(df, col_name):
    print(f"\nProcessing column: {col_name}")
    # Calculate Q1, Q3, and IQR
    quartiles = df.approxQuantile(col_name, [0.25, 0.75], 0.05)
    Q1 = quartiles[0]
    Q3 = quartiles[1]
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    print(f"  Q1: {Q1}, Q3: {Q3}, IQR: {IQR}")
    print(f"  Lower Bound (IQR method): {lower_bound}")
    print(f"  Upper Bound (IQR method): {upper_bound}")

    # Identify outliers
    is_outlier_col = (col(col_name) < lower_bound) | (col(col_name) >
    upper_bound)
    outliers_count = df.filter(is_outlier_col).count()
    total_count = df.count()
    print(f"  Number of outliers detected: {outliers_count} ({outliers_count/
    total_count:.2%})")

    return df
```

6. DEFINE MODEL EVALUATION METRICS

```python
from pyspark.sql.functions import col, expr

def evaluate_classification_metrics(predictions, model_name="Model"):
    """
    Calculate binary classification metrics from a Spark ML prediction
    DataFrame.

    Parameters:
    - predictions: Spark DataFrame containing 'prediction' and 'label' columns
    - model_name: optional name of the model (for display purposes)

    Returns:
    - Dictionary with Precision, Recall, Accuracy, and F1 Score
    """
    # Confusion matrix elements
    tp = predictions.filter("prediction == 1 AND label == 1").count()
    tn = predictions.filter("prediction == 0 AND label == 0").count()
    fp = predictions.filter("prediction == 1 AND label == 0").count()
    fn = predictions.filter("prediction == 0 AND label == 1").count()
```

6. DEFINE MODEL EVALUATION METRICS (CONT.)

```
# Derived metrics
precision = tp / (tp + fp) if (tp + fp) > 0 else 0
recall = tp / (tp + fn) if (tp + fn) > 0 else 0
accuracy = (tp + tn) / (tp + tn + fp + fn) if (tp + tn + fp + fn) > 0 else 0
f1 = 2 * precision * recall / (precision + recall) if (precision + recall)
> 0 else 0

# Display results
print(f"Evaluation Results for {model_name}:")
print(f"Precision: {precision:.3f}")
print(f"Recall:{recall:.3f}")
print(f"Accuracy:{accuracy:.3f}")
print(f"F1 Score:{f1:.3f}")

# Return all metrics as a dictionary
return {
    "model": model_name,
    "precision": precision,
    "recall": recall,
    "accuracy": accuracy,
    "f1": f1
}
```

2. EXPLORATORY DATA ANALYSIS

1. EXPLORE THE SCHEMA

```
# Checking numbers of rows and columns in each dataset:
print("Train dataset:")
count_rowncol(train)
print()
print("Test dataset:")
count_rowncol(test)
print()

# The typical schema of our dataset:
train.printSchema()
```

```
Train dataset:
<class 'pyspark.sql.dataframe.DataFrame'>
Number of Rows:  103904
Number of Columns:  25

Test dataset:
<class 'pyspark.sql.dataframe.DataFrame'>
Number of Rows:  25976
Number of Columns:  25
```

```
root
 |-- _c0: integer (nullable = true)
 |-- id: integer (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Customer Type: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Type of Travel: string (nullable = true)
 |-- Class: string (nullable = true)
 |-- Flight Distance: integer (nullable = true)
 |-- Inflight wifi service: integer (nullable = true)
 |-- Departure/Arrival time convenient: integer (nullable = true)
 |-- Ease of Online booking: integer (nullable = true)
 |-- Gate location: integer (nullable = true)
 |-- Food and drink: integer (nullable = true)
 |-- Online boarding: integer (nullable = true)
 |-- Seat comfort: integer (nullable = true)
 |-- Inflight entertainment: integer (nullable = true)
 |-- On-board service: integer (nullable = true)
 |-- Leg room service: integer (nullable = true)
 |-- Baggage handling: integer (nullable = true)
 |-- Checkin service: integer (nullable = true)
 |-- Inflight service: integer (nullable = true)
 |-- Cleanliness: integer (nullable = true)
 |-- Departure Delay in Minutes: integer (nullable = true)
 |-- Arrival Delay in Minutes: double (nullable = true)
 |-- satisfaction: string (nullable = true)
```

2. EXPLORE TOTAL COUNTS OF NULL VALUE

The null values in Train dataset:

_c0	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease of Online booking
0	0	0	0	0	0	0	0	0	0	

Gate location	Food and drink	Online boarding	Seat comfort	Inflight entertainment	On-board service	Leg room service
0	0	0	0	0	0	0

Baggage handling	Checkin service	Inflight service	Cleanliness	Departure Delay in Minutes	Arrival Delay in Minutes	satisfaction
0	0	0	0	0	310	0

3. EXPLORE TOTAL COUNTS OF DISTINCT VALUES

Distinct values check:

count(DISTINCT _c0)	count(DISTINCT Gender)	count(DISTINCT Customer Type)	count(DISTINCT Age)	count(DISTINCT Type of Travel)	count(DISTINCT Class)	count(DISTINCT Flight Distance)
103904	2	2	75	2	3	3802

count(DISTINCT Inflight wifi service)	count(DISTINCT Departure/Arrival time convenient)	count(DISTINCT Ease of Online booking)	count(DISTINCT Gate location)	count(DISTINCT Food and drink)
6	6	6	6	6

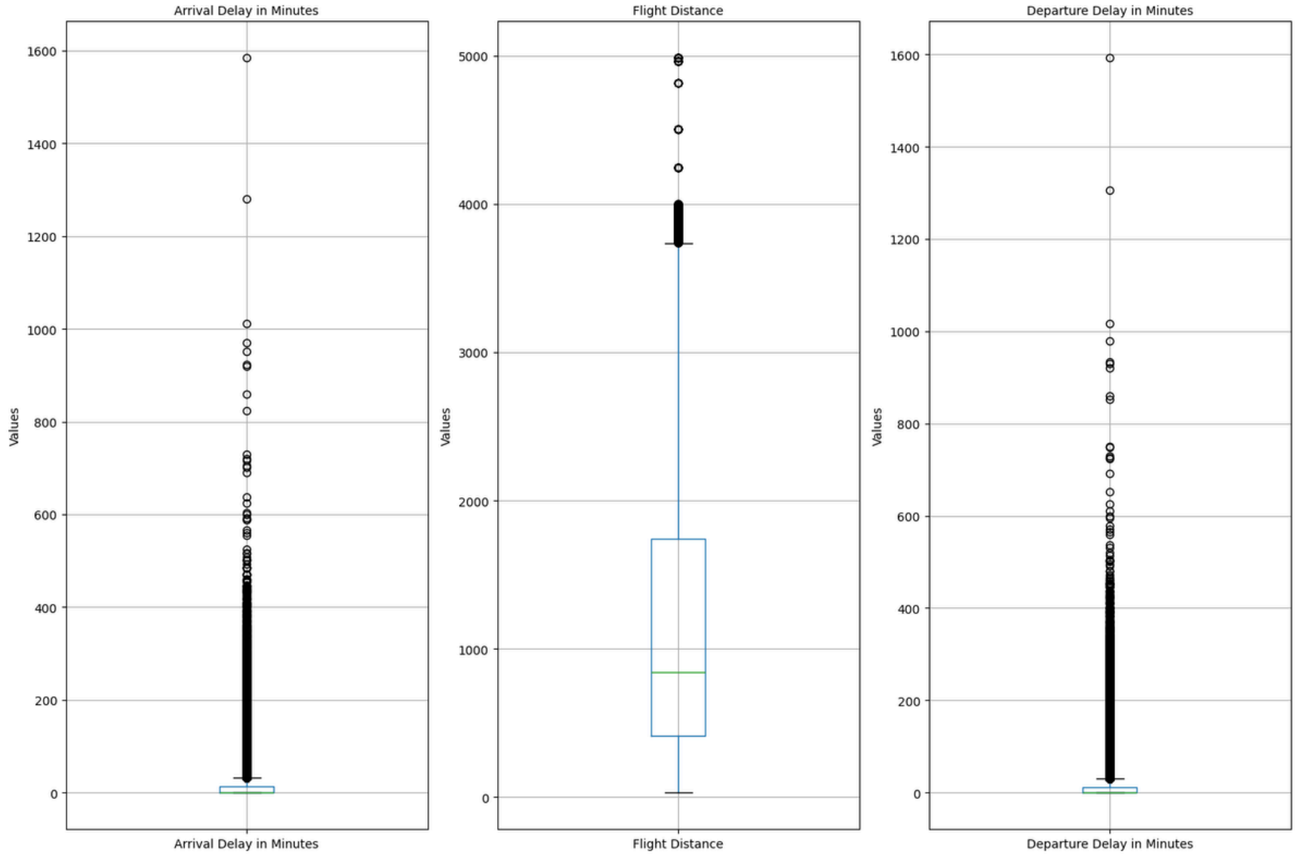
count(DISTINCT Online boarding)	count(DISTINCT Seat comfort)	count(DISTINCT Inflight entertainment)	count(DISTINCT On-board service)	count(DISTINCT Leg room service)	count(DISTINCT Baggage handling)
6	6	6	6	6	5

count(DISTINCT Checkin service)	count(DISTINCT Inflight service)	count(DISTINCT Cleanliness)	count(DISTINCT Departure Delay in Minutes)	count(DISTINCT Arrival Delay in Minutes)	count(DISTINCT satisfaction)
6	6	6	446	455	2

Column: _c0, Unique Values: [0, 35875, 93560, 93073, 89858, 89371, 72262, 36387, 35900, 68073, 68560, 47262, 14602, 11387, 10900, 22262, 86377, 85890, 50502, 50015, 82675, 82188, 85915, 85428, 81726, 64617, 64130, 28742, 28255, 60428,
Column: Gender, Unique Values: ['Female', 'Male']
Column: Customer Type, Unique Values: ['Loyal Customer', 'disloyal Customer']
Column: Age, Unique Values: [60, 52, 31, 75, 54, 25, 77, 48, 27, 19, 71, 50, 42, 21, 73, 65, 44, 23, 15, 67, 46, 38, 17, 69, 61, 40, 11, 63, 34, 13, 57, 36, 28, 7, 80, 59, 51, 30, 9, 74, 53, 32, 24, 76, 55, 47, 26, 78, 70, 49, 20, 72,
Column: Type of Travel, Unique Values: ['Business travel', 'Personal Travel']
Column: Class, Unique Values: ['Business', 'Eco', 'Eco Plus']
Column: Flight Distance, Unique Values: [1851, 3958, 743, 2107, 2594, 3271, 2784, 933, 3040, 2553, 2584, 2809, 2322, 471, 3973, 3486, 271, 1635, 2122, 3742, 3255, 3286, 1435, 948, 1204, 2824, 2337, 486, 973, 3957, 3988, 2137, 1650, 3
Column: Inflight wifi service, Unique Values: [0, 1, 5, 2, 3, 4]
Column: Departure/Arrival time convenient, Unique Values: [0, 1, 5, 2, 3, 4]
Column: Ease of Online booking, Unique Values: [0, 1, 5, 2, 3, 4]
Column: Gate location, Unique Values: [0, 1, 5, 2, 3, 4]
Column: Food and drink, Unique Values: [0, 1, 5, 2, 3, 4]
Column: Online boarding, Unique Values: [0, 1, 5, 2, 3, 4]
Column: Seat comfort, Unique Values: [0, 1, 5, 2, 3, 4]
Column: Inflight entertainment, Unique Values: [0, 1, 5, 2, 3, 4]
Column: On-board service, Unique Values: [0, 1, 5, 2, 3, 4]
Column: Leg room service, Unique Values: [0, 1, 5, 2, 3, 4]
Column: Baggage handling, Unique Values: [1, 5, 2, 3, 4]
Column: Checkin service, Unique Values: [0, 1, 5, 2, 3, 4]
Column: Inflight service, Unique Values: [0, 1, 5, 2, 3, 4]
Column: Cleanliness, Unique Values: [0, 1, 5, 2, 3, 4]
Column: Departure Delay in Minutes, Unique Values: [0, 437, 306, 277, 256, 227, 206, 177, 156, 148, 127, 106, 98, 77, 454, 933, 48, 27, 304, 296, 275, 123, 94, 73, 44, 23, 371, 342, 321, 292, 271, 750, 729, 242, 221, 213, 192, 163, 1
Column: Arrival Delay in Minutes, Unique Values: [0.0, 293.0, 123.0, 71.0, 392.0, 220.0, 122.0, 347.0, 388.0, 218.0, 106.0, 59.0, 33.0, 567.0, 186.0, 105.0, 320.0, 438.0, 104.0, 184.0, 243.0, 275.0, 271.0, 434.0, 88.0, 50.0, 28.0, 20
Column: satisfaction, Unique Values: ['satisfied', 'neutral or dissatisfied']

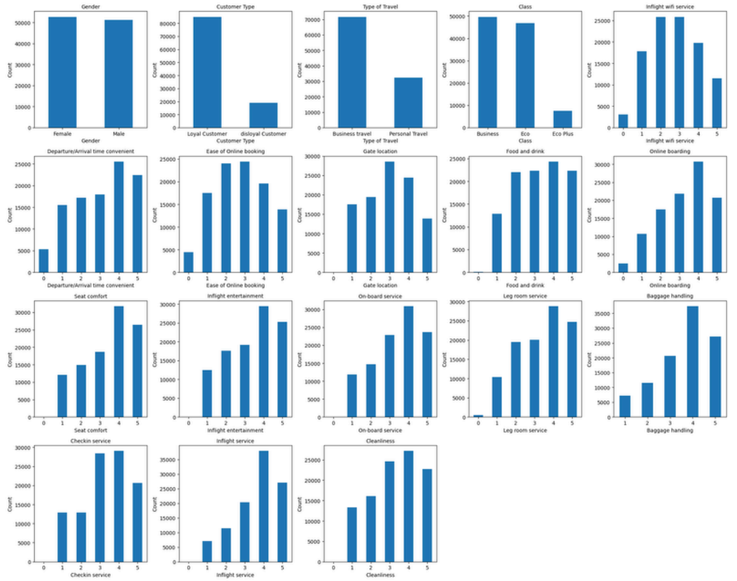
4. DISPLAYING OUTLIERS

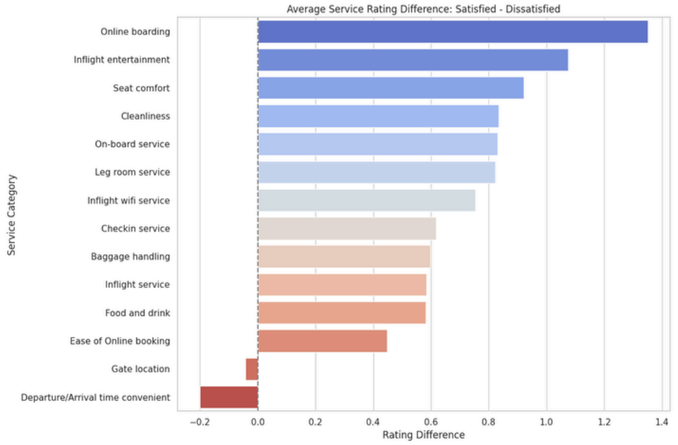
```
Processing column: Age
  Q1: 27.0, Q3: 50.0, IQR: 23.0
  Lower Bound (IQR method): -7.5
  Upper Bound (IQR method): 84.5
  Number of outliers detected: 17 (0.02%)

Processing column: Flight Distance
  Q1: 421.0, Q3: 1572.0, IQR: 1151.0
  Lower Bound (IQR method): -1305.5
  Upper Bound (IQR method): 3298.5
  Number of outliers detected: 5899 (5.68%)

Processing column: Departure Delay in Minutes
  Q1: 0.0, Q3: 9.0, IQR: 9.0
  Lower Bound (IQR method): -13.5
  Upper Bound (IQR method): 22.5
  Number of outliers detected: 18381 (17.69%)
```

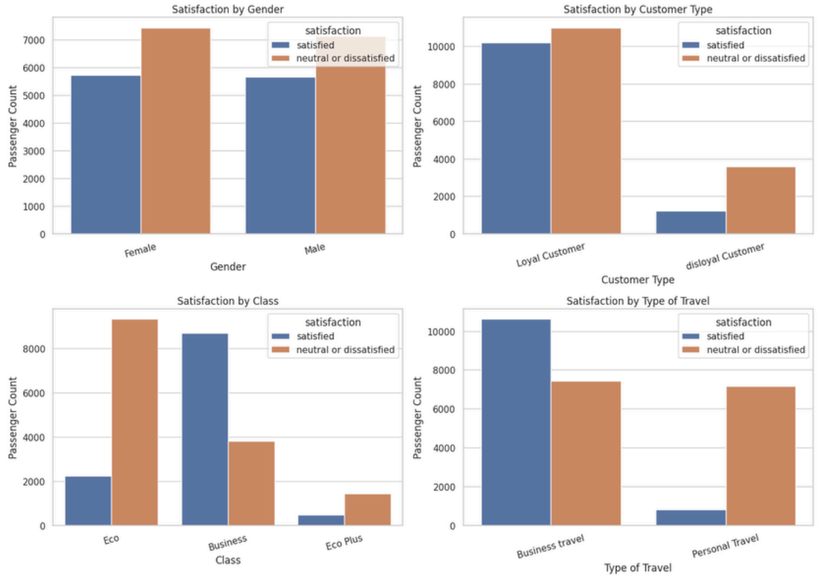


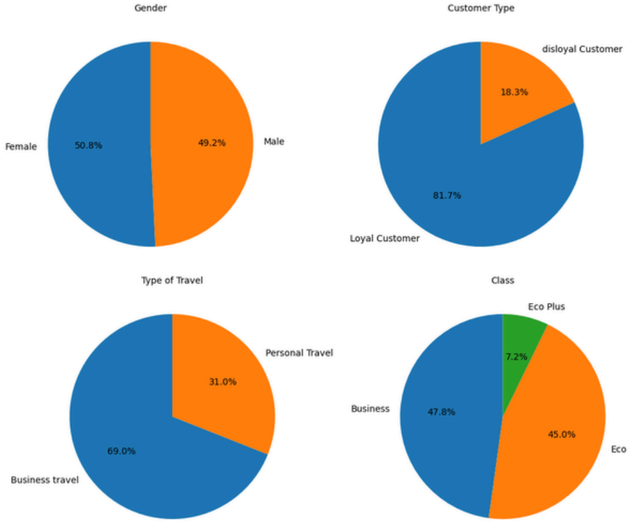
5. SORTING VARIABLE TYPES

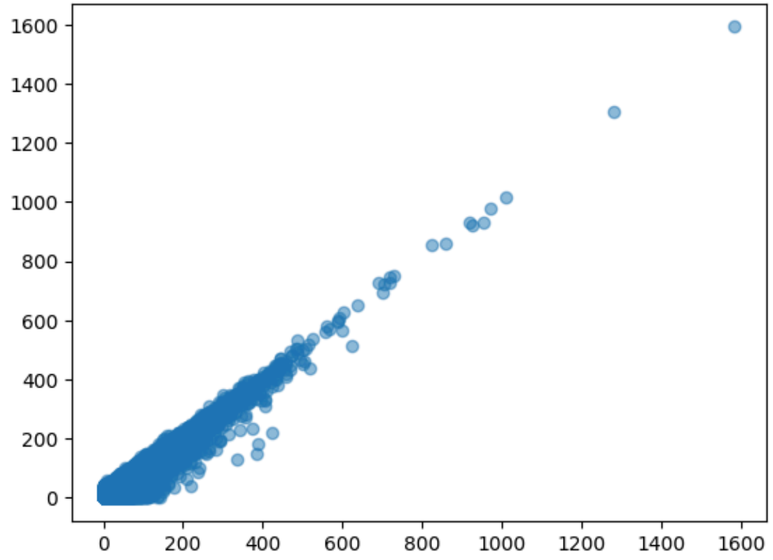
- **Categorical variables in int:** ['Inflight wifi service', 'Departure/Arrival time convenient', 'Ease of Online booking', 'Gate location', 'Food and drink', 'Online boarding', 'Seat comfort', 'Inflight entertainment', 'On-board service', 'Leg room service', 'Baggage handling', 'Checkin service', 'Inflight service', 'Cleanliness']
- **Categorical variables in str:** ['Gender', 'Customer Type', 'Type of Travel', 'Class']
- **Numeric variables:** ['Age', 'Flight Distance', 'Departure Delay in Minutes', 'Arrival Delay in Minutes']
- **Target variable:** ['satisfaction']

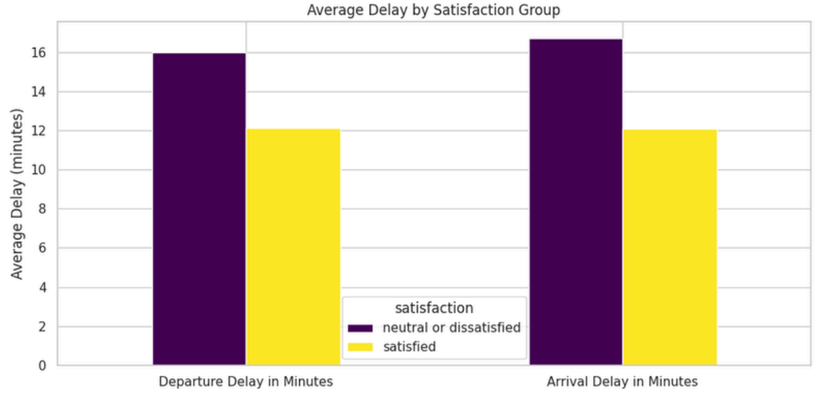

Gender
Female
Male
Customer Type
Loyal Customer
disloyal Customer
Type of Travel
Business travel
Personal Travel
Class
Business
Eco
Eco Plus
Inflight wifi service
Departure/Arrival time convenient
Ease of Online booking
Gate location
Food and drink
Online boarding
Seat comfort
Inflight entertainment
On-board service
Leg room service
Baggage handling
Checkin service
Inflight service
Cleanliness
Count


Average Service Rating Difference: Satisfied - Dissatisfied
Online boarding
Inflight entertainment
Seat comfort
Cleanliness
On-board service
Leg room service
Inflight wifi service
Checkin service
Baggage handling
Inflight service
Food and drink
Ease of Online booking
Gate location
Departure/Arrival time convenient
Service Category
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
Rating Difference


Satisfaction by Gender
satisfaction
satisfied
neutral or dissatisfied
Passenger Count
Female
Male
Gender
Satisfaction by Customer Type
Loyal Customer
disloyal Customer
Customer Type
Satisfaction by Class
Eco
Business
Eco Plus
Class
Satisfaction by Type of Travel
Business travel
Personal Travel
Type of Travel


Gender
Female
50.8%
49.2%
Male
Customer Type
disloyal Customer
18.3%
81.7%
Loyal Customer
Type of Travel
Personal Travel
31.0%
69.0%
Business travel
Class
Eco Plus
7.2%
Business
47.8%
45.0%
Eco


0
200
400
600
800
1000
1200
1400
1600


Average Delay by Satisfaction Group
Average Delay (minutes)
satisfaction
neutral or dissatisfied
satisfied
Departure Delay in Minutes
Arrival Delay in Minutes

3. DATA PREPROCESSING AND PIPELINE CREATING

PIPELINE

MAIN OBJECTIVE

To prepare raw airline passenger data into a clean, consistent, and machine-readable format, ready for model training and prediction.

This pipeline automates the full preprocessing flow including:

- Handling missing values
- Encoding categorical variables
- Scaling numeric features
- Assembling all features into a single vector
- *pipeline = Pipeline(stages=[...])*

Each stage plays a critical role in transforming the data for effective learning.

BUILDING PIPELINE

Pipeline stages:

- Target Indexer
- Imputer
- String Indexers
- String Encoder
- Vector Assembler for scaler
- Robust Scaler
- Vector Assembler

IMPUTER

MAIN OBJECTIVE

To handle missing values in numerical features, ensuring no *NaNs* break the model pipeline.

- Using the **median** strategy to replace missing values in *Arrival Delay in Minutes*.
- Ensuring the model remains reliable even when some delay data is missing.
- *Imputer(inputCols=..., strategy='median')*

SELECT THE BEST PARAMETER FOR IMPUTER

```python
# Define a function to find the best imputation strategy for a column
def find_best_imputation_strategy(df, col_name, strategy_options=["mean",
"median"]):
    """
    Finds the best imputation strategy (mean, median, or mode) for a given
    column
    by evaluating which one minimizes the standard deviation of the column
    after imputation.
    This is a heuristic and might not be the best strategy for all scenarios.
    """
    original_std = df.select(col_name).agg({'`{}`'.format(col_name): 'stddev'}).
    collect()[0][0]
    print(f"Original standard deviation of '{col_name}': {original_std}")
    best_strategy = None
    min_std_diff = float('inf')

    for strategy in strategy_options:
        imputer = Imputer(inputCols=[col_name], outputCols=[col_name],
        strategy=strategy)
        try:
            model = imputer.fit(df)
            df_imputed = model.transform(df)
            imputed_std = df_imputed.select(col_name).agg({'`{}`'.format
            (col_name): 'stddev'}).collect()[0][0]
            #print(f"Standard deviation after imputation with '{strategy}':
            {imputed_std}")

            if imputed_std is not None and original_std is not None:
                std_diff = abs(imputed_std - original_std)
                if std_diff < min_std_diff:
                    min_std_diff = std_diff
                    #print(f"New minimum standard deviation difference:
                    {min_std_diff}")
                    best_strategy = strategy
        except Exception as e:
            print(f"Could not apply strategy '{strategy}' to column '{col_name}
            ': {e}")
            continue

    return best_strategy
```

```python
# Columns with null values identified from previous count_null output
columns_to_impute = ['Arrival Delay in Minutes']

# Impute null values in the train dataset
print("\nImputing null values in the train dataset...")
for col_name in columns_to_impute:
    best_strategy_train = find_best_imputation_strategy(train, col_name,
    strategy_options=["mean", "median"])
    if best_strategy_train:
        print(f"Best imputation strategy for '{col_name}' in train:
        {best_strategy_train}")
        imputer = Imputer(inputCols=[col_name], outputCols=[col_name],
        strategy=best_strategy_train)
        model = imputer.fit(train)
        train = model.transform(train)
    else:
        print(f"Could not find a suitable strategy for '{col_name}' in train.")

print("\nChecking null values of Train dataset after imputation:")
count_null(train)
```

```
Imputing null values in the train dataset...
Original standard deviation of 'Arrival Delay in Minutes': 38.698682020966515
Best imputation strategy for 'Arrival Delay in Minutes' in train: median
```

BEFORE IMPUTATION

Departure Delay in Minutes	Arrival Delay in Minutes
0	310

AFTER IMPUTATION

Departure Delay in Minutes	Arrival Delay in Minutes
0	0

IMPUTER IN PIPELINE BUILDING

```
# 0.Handle Missing Values
imputer = Imputer(
    inputCols=["Arrival Delay in Minutes"],
    outputCols=["Arrival Delay in Minutes"],
    strategy="median"
    )
```

STRING INDEXER & ONEHOT ENCODER

MAIN OBJECTIVE

To convert categorical string variables into machine-readable numeric features.

- **StringIndexer**: Assigns a unique index to each category (*e.g., "Business" → 0, "Economy" → 1*)
- **OneHotEncoder**: Converts these indices into binary vectors to avoid implying ordinal relationships
- *StringIndexer(inputCol=..., outputCol=...)*
- *OneHotEncoder(inputCols=..., outputCols=...)*

STRING INDEXER & ONEHOT ENCODER IN PIPELINE BUILDING

```
# 1.Stage String Indexer and One Hot Encoder for the binary features:
str_indexers = [
    StringIndexer(inputCol=col, outputCol=f"{col}_index", handleInvalid="keep")
    for col in categorical_variables_str
]
str_encoder = OneHotEncoder(
    inputCols=[f"{col}_index" for col in categorical_variables_str],
    outputCols=[f"{col}_encoded" for col in categorical_variables_str],
    dropLast=True
)
```

```
# 4. StringIndexer for target variable
target_indexer = StringIndexer(
    inputCol="satisfaction",
    outputCol="label",
    handleInvalid="keep"
)
```

ROBUST SCALER

(WITH VECTORASSEMBLER)

MAIN OBJECTIVE

To normalize numerical features while reducing the effect of outliers.

- First, we use **VectorAssembler** to wrap each numeric column (e.g., Age, Distance) into a 1-element vector.
- Then we apply **RobustScaler**, which scales using interquartile range (IQR) — more stable for skewed data.
- *VectorAssembler(inputCols=['Age'], outputCol='Age_vec')*
- *RobustScaler(inputCol='Age_vec', outputCol='Age_scaled')*

ROBUST SCALER IN PIPELINE BUILDING

```
# 2. Stage Scale the feature (numeric columns) with RobustScaler
# Assemble individual columns into 1-item vectors
vec_age = VectorAssembler(inputCols=["Age"], outputCol="Age_vec")
vec_distance = VectorAssembler(inputCols=["Flight Distance"], outputCol="Flight
Distance_vec")
vec_depart = VectorAssembler(inputCols=["Departure Delay in Minutes"],
outputCol="Departure Delay_vec")
vec_arrive = VectorAssembler(inputCols=["Arrival Delay in Minutes"],
outputCol="Arrival Delay_vec")

scaler_age = RobustScaler(inputCol="Age_vec", outputCol="Age_scaled")
scaler_distance = RobustScaler(inputCol="Flight Distance_vec",
outputCol="Flight Distance_scaled")
scaler_depart = RobustScaler(inputCol="Departure Delay_vec",
outputCol="Departure Delay in Minutes_scaled")
scaler_arrive = RobustScaler(inputCol="Arrival Delay_vec", outputCol="Arrival
Delay in Minutes_scaled")
```

VECTOR ASSEMBLER

MAIN OBJECTIVE

To combine all transformed features — both numeric and encoded categorical — into a single feature vector.

- This is the final step before feeding data into the machine learning model.
- It ensures all inputs are in the correct format (*features* column) as required by Spark ML models.
- *VectorAssembler(inputCols=..., outputCol="features")*

VECTOR ASSEMBLER IN PIPELINE BUILDING

```
# 3. Stage vectorize the numeric variables (numeric columns and categorical int
columns) into Feature
assembler = VectorAssembler(inputCols= categorical_variables_int + [f"{col}
_scaled" for col in numeric_variables] + [f"{col}_encoded" for col in
categorical_variables_str],
                            outputCol="features"
                            )
```

RESULT OF DATA PREPROCESSING

FEATURES	LABEL
[3.0,4.0,3.0,1.0,5.0,3.0,5.0,5.0,4.0,3.0,4.0,4.0,5.0,5.0,0.542,0.347,2.083,1.384,0.0,1.0,1.0,0.0,0.0,1.0,0.0,0.0,1.0]	0.0
[3.0,2.0,3.0,3.0,1.0,3.0,1.0,1.0,1.0,5.0,3.0,1.0,4.0,1.0,1.042,0.1772,0.083,0.462,0.0,1.0,0.0,1.0,1.0,0.0,1.0,0.0,0.0]	0.0
[2.0,2.0,2.0,2.0,5.0,5.0,5.0,5.0,4.0,3.0,4.0,4.0,4.0,5.0,1.083,0.861,0.0,0.0,1.0,0.0,1.0,0.0,1.0,0.0,1.0,0.0,0.0]	1.0
...	...

4. MODEL TRAINING AND EVALUATION

MODEL SELECTION

LOGISTICS REGRESSION

RANDOM FOREST

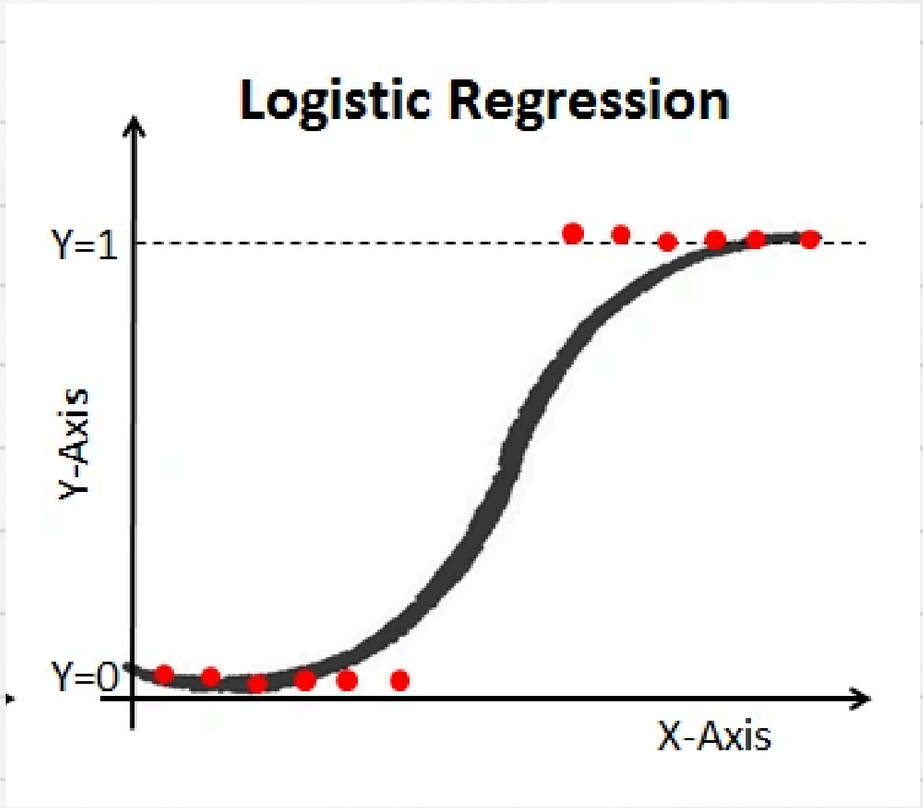
GRADIENT BOOST TREE

MULTI-LAYER PERCEPTRON

LOGISTICS REGRESSION

MAIN OBJECTIVE

- Modeling the linear relationship between features and the binary satisfaction outcome
- Ideal for understanding feature importance.



```python
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import BinaryClassificationEvaluator

# Define input feature size
input_size = len(train_final.select("features").first()[0])

# Define Logistic Regression
lr = LogisticRegression(
    featuresCol="features",
    labelCol="label",
    maxIter=500,
    family="binomial",
    elasticNetParam=1.0  # L1 by default; will vary in grid
)

# Define hyperparameter grid
paramGrid_lr = ParamGridBuilder()\
    .addGrid(lr.regParam, [0.001, 0.01, 0.1])\
    .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0])\
    .build()
```

```python
# Define evaluator
evaluator = BinaryClassificationEvaluator(
    labelCol="label",
    rawPredictionCol="rawPrediction",
    metricName="areaUnderROC"
)

# Define cross-validator
cv_lr = CrossValidator(
    estimator=lr,
    estimatorParamMaps=paramGrid_lr,
    evaluator=evaluator,
    numFolds=10,
    parallelism=10,
    seed=42
)

# Train, predict, evaluate, and save
cv_lr_model = cv_lr.fit(train_data)
lr_predictions = cv_lr_model.transform(test_data)
lr_auc = evaluator.evaluate(lr_predictions)
print(f"Tuned Logistic Regression AUC: {lr_auc:.4f}")
```

```python
# Show LR sample predictions
lr_predictions.orderBy(rand(seed=42)).select(
    "label", "prediction", "probability"
).show(10, truncate=False)
```

```python
# Show LR metrics
lr_result = evaluate_classification_metrics(lr_predictions, "Logistic Regression")
```

Evaluation Results for Logistic Regression:

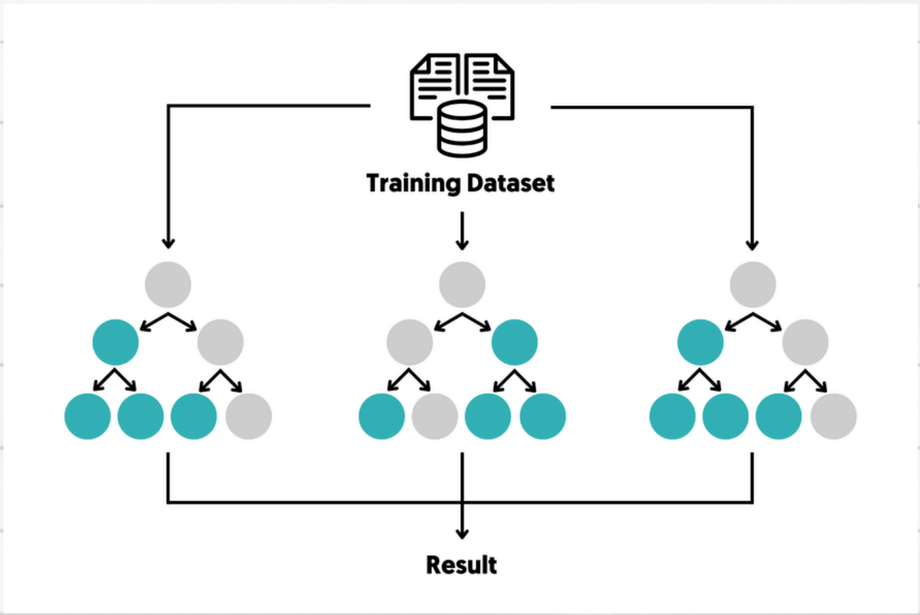
Precision: 0.868

Recall:0.834

Accuracy:0.871

F1 Score:0.851

RANDOM FOREST



MAIN OBJECTIVE

- Building multiple decision trees to capture complex feature interactions and reduce overfitting
- Suitable for the diverse feature set including service ratings and demographic variables

```python
from pyspark.ml.classification import RandomForestClassifier

rf = RandomForestClassifier(
    featuresCol="features",
    labelCol="label",
    seed=42
)

# Parameter grid for tuning
paramGrid_rf = ParamGridBuilder()\
    .addGrid(rf.numTrees, [50, 100])\
    .addGrid(rf.maxDepth, [5, 8, 12])\
    .build()

# CrossValidator
cv_rf = CrossValidator(
    estimator=rf,
    estimatorParamMaps=paramGrid_rf,
    evaluator=evaluator,
    numFolds=10,
    parallelism=10,
    seed=42
)
# Train tuned RF model
cv_rf_model = cv_rf.fit(train_data)

# Predict and evaluate
rf_predictions = cv_rf_model.transform(test_data)
rf_auc = evaluator.evaluate(rf_predictions)
print(f"Tuned Random Forest AUC: {rf_auc:.4f}")
```

```
Evaluation Results for Random Forest:
Precision: 0.954
Recall:0.944
Accuracy:0.955
F1 Score:0.949
```

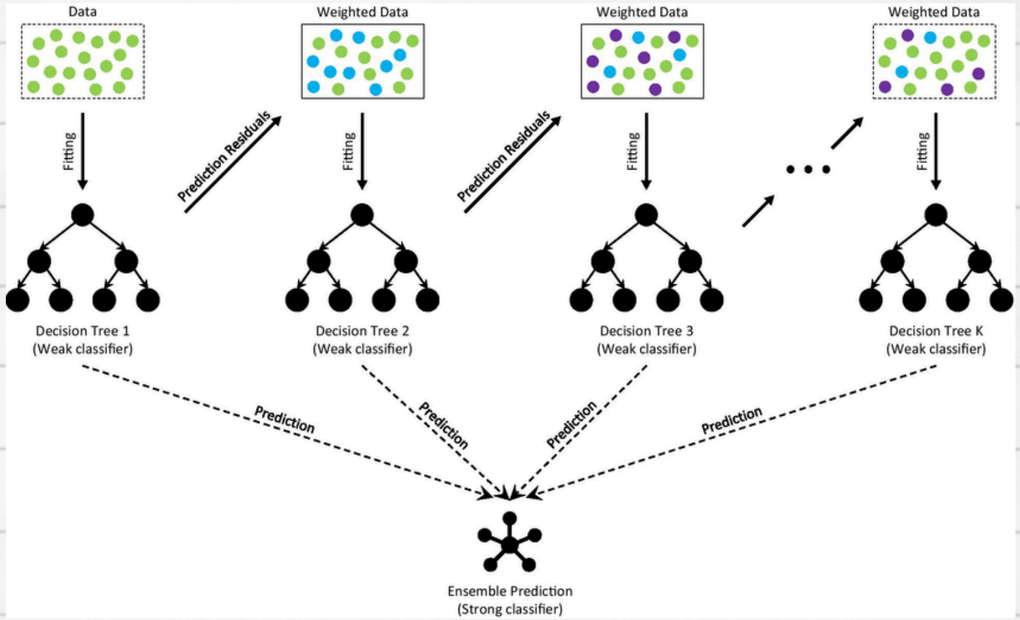
```python
# Show RF sample predictions
rf_predictions.orderBy(rand(seed=42)).select(
    "label", "prediction", "probability"
).show(10, truncate=False)
```

```python
# Show RF metrics
rf_result = evaluate_classification_metrics(rf_predictions, "Random Forest")
```

GRADIENT BOOST TREE

MAIN OBJECTIVE

- Sequentially building trees to correct errors from previous iterations, optimizing for predictive accuracy
- Particularly effective for imbalanced or noisy data



```python
from pyspark.ml.classification import MultilayerPerceptronClassifier
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import BinaryClassificationEvaluator

# Define input feature size
input_size = len(train_final.select("features").first()[0])

# Define MLP
mlp = MultilayerPerceptronClassifier(
    featuresCol="features",
    labelCol="label",
    maxIter=200,
    seed=123
)

# Hyperparameter grid (vary hidden layers)
paramGrid_mlp = ParamGridBuilder()\
    .addGrid(mlp.layers, [
        [input_size, 16, 2],
        [input_size, 32, 16, 2],
        [input_size, 64, 32, 2]
    ])\
    .build()
```

```python
# Binary evaluator
evaluator = BinaryClassificationEvaluator(
    labelCol="label",
    rawPredictionCol="rawPrediction",
    metricName="areaUnderROC"
)

# Cross-validator
cv_mlp = CrossValidator(
    estimator=mlp,
    estimatorParamMaps=paramGrid_mlp,
    evaluator=evaluator,
    numFolds=5,
    parallelism=10,
    seed=42
)

# Train, predict, evaluate, and save
cv_mlp_model = cv_mlp.fit(train_data)
mlp_predictions = cv_mlp_model.transform(test_data)
mlp_auc = evaluator.evaluate(mlp_predictions)
print(f"Tuned MLP AUC: {mlp_auc:.4f}")
```

```python
# Show MLP sample predictions
mlp_predictions.orderBy(rand(seed=42)).select(
    "label", "prediction", "probability"
).show(10, truncate=False)
```

```python
# Show MLP metrics
mlp_result = evaluate_classification_metrics(mlp_predictions, "MultiLayer Perceptron")
```

Evaluation Results for Gradient Boost Tree:

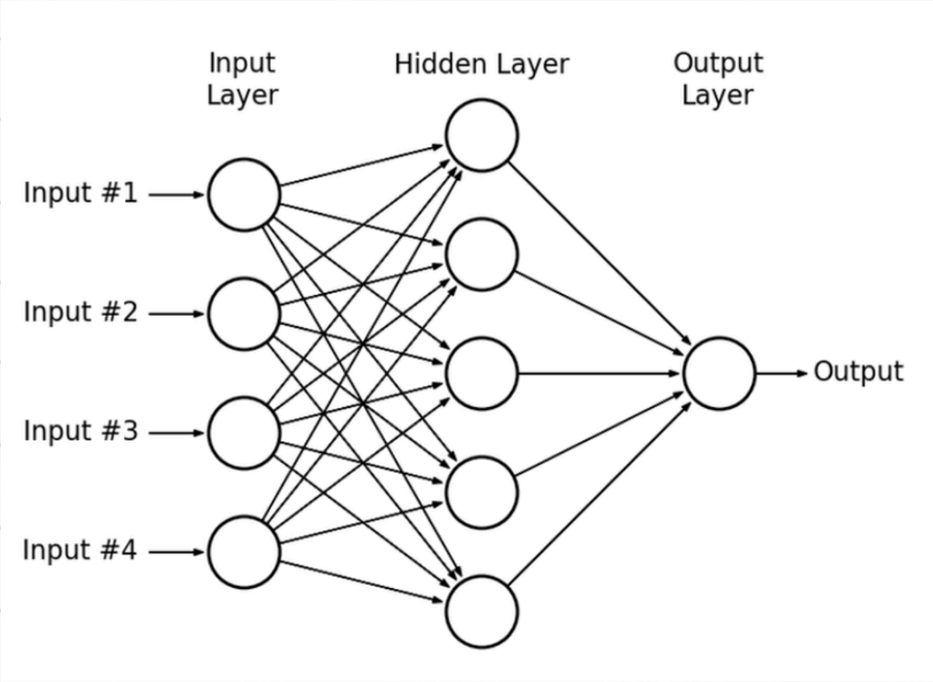
Precision: 0.945

Recall:0.929

Accuracy:0.945

F1 Score:0.937

MULTI-LAYER PERCEPTRON



MAIN OBJECTIVE

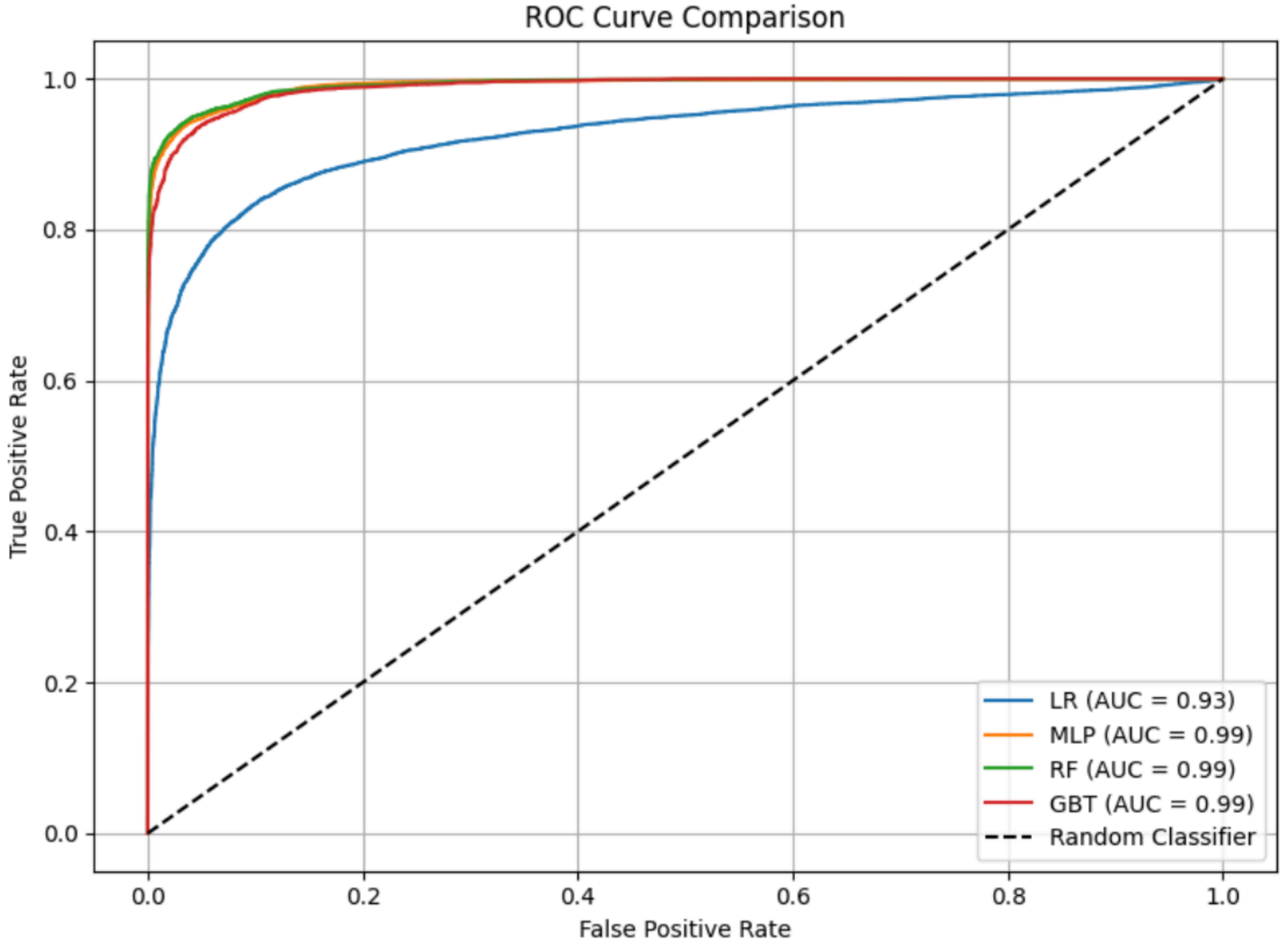
- A neural network model, capturing non-linear patterns through multiple hidden layers
- Offering flexibility for complex relationships in the dataset

```python
from pyspark.ml.classification import MultilayerPerceptronClassifier
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import BinaryClassificationEvaluator

# Define input feature size
input_size = len(train_final.select("features").first()[0])

# Define MLP
mlp = MultilayerPerceptronClassifier(
    featuresCol="features",
    labelCol="label",
    maxIter=200,
    seed=123
)

# Hyperparameter grid (vary hidden layers)
paramGrid_mlp = ParamGridBuilder()\
    .addGrid(mlp.layers, [
        [input_size, 16, 2],
        [input_size, 32, 16, 2],
        [input_size, 64, 32, 2]
    ])\
    .build()
# Binary evaluator
evaluator = BinaryClassificationEvaluator(
    labelCol="label",
    rawPredictionCol="rawPrediction",
    metricName="areaUnderROC"
)

# Cross-validator
cv_mlp = CrossValidator(
    estimator=mlp,
    estimatorParamMaps=paramGrid_mlp,
    evaluator=evaluator,
    numFolds=5,
    parallelism=10,
    seed=42
)

# Train, predict, evaluate, and save
cv_mlp_model = cv_mlp.fit(train_data)
mlp_predictions = cv_mlp_model.transform(test_data)
mlp_auc = evaluator.evaluate(mlp_predictions)
print(f"Tuned MLP AUC: {mlp_auc:.4f}")
```

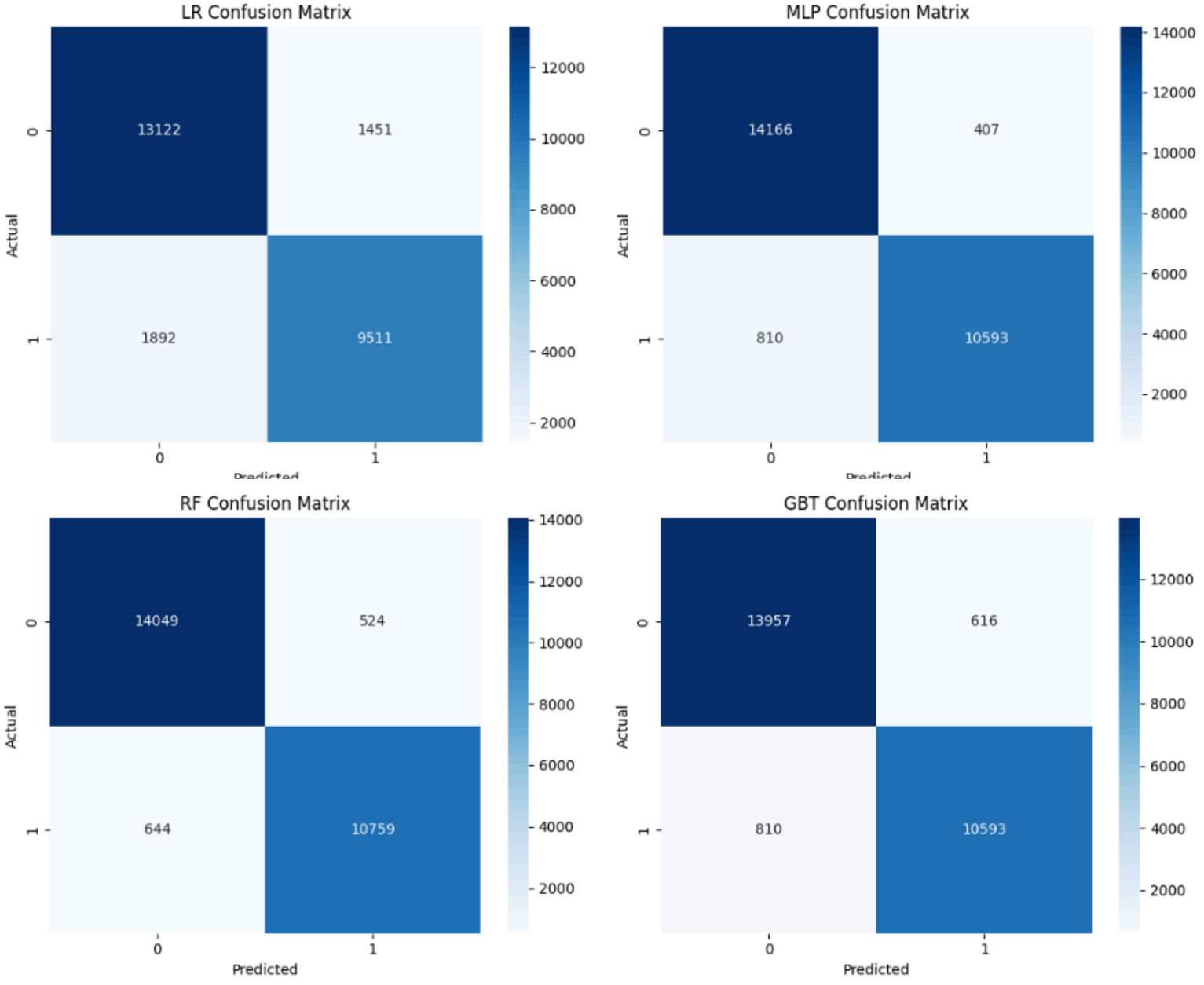
```python
# Show MLP sample predictions
mlp_predictions.orderBy(rand(seed=42)).select(
    "label", "prediction", "probability"
).show(10, truncate=False)
```

```python
# Show MLP metrics
mlp_result = evaluate_classification_metrics(mlp_predictions, "MultiLayer Perceptron")
```

```
Evaluation Results for MultiLayer Perceptron:
Precision: 0.963
Recall:0.929
Accuracy:0.953
F1 Score:0.946
```



	Model	Accuracy	AUC
0	LR	0.871	0.925
1	MLP	0.953	0.991
2	RF	0.955	0.992
3	GBT	0.945	0.989


LR Confusion Matrix
13122
1451
1892
9511
Actual
Predicted
MLP Confusion Matrix
14166
407
810
10593
RF Confusion Matrix
14049
524
644
10759
GBT Confusion Matrix
13957
616
810
10593

WEBSITE DEMO WORK

METHODOLOGY: PROCEDURES AND TOOLS

Streamlit

FUTURE WORK

- **Real-time feedback analysis** – Predict satisfaction using live reviews and social media.
- **Personalized suggestions** – Recommend specific service improvements for different passengers.
- **Explainable AI dashboards** – Show key reasons behind predictions for better decisions.
- **Multimodal data (text, audio, images)** – Improve accuracy with more data types.
- **Link to flight delays/maintenance** – Connect satisfaction with operational issues.

THANK YOU VERY MUCH!