VIETNAM NATIONAL UNIVERSITY OF HO CHI MINH CITY
INTERNATIONAL UNIVERSITY
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

BIG DATA TECHNOLOGY
IT161IU

REPORT

Topic: Application for Predicting Airline Passenger Satisfaction

By Group Hoi Dong O5 – Member List

Name	**Student ID**
Nguyen Minh Dat	ITDSIU22166
Nguyen Du Nhan	ITDSIU22140
Duong Ngoc Phuong Anh	ITDSIU22135
Nguyen Hai Phu	ITDSIU22179

Instructor: Dr. Ho Long Van

Ho Chi Minh City, Viet Nam
Semester 2, 2024 – 2025

Contents

List of Figures

I Introduction

1 Abstract

This project presents a scalable system for predicting airline passenger satisfaction using big data techniques and machine learning. By leveraging PySpark's distributed processing capabilities, the application processes high-dimensional flight data efficiently. The system incorporates a complete pipeline—from data acquisition and transformation to predictive modeling—optimized for performance and reliability. The insights generated aim to support airlines in tailoring customer service strategies and enhancing operational decision-making based on predictive analytics.

2 Objective

The primary objectives of this project are to:

- Develop an end-to-end big data processing pipeline using PySpark
- Analyze factors influencing airline passenger satisfaction
- Build predictive models to classify passenger satisfaction levels
- Evaluate model performance and identify key drivers of satisfaction

3 Tools Used

- Google Colab
- Github
- Visual Studio Code
- Streamlit

II Methodology

1 Overview

This chapter outlines the methodological approach undertaken in developing a flight passenger satisfaction prediction system. The project employs a data-driven machine learning pipeline implemented using PySpark, a distributed data processing framework well-suited for large-scale computations. The primary objective of the system is to predict the likelihood of passenger satisfaction based on a variety of in-flight and service-related attributes, enabling stakeholders—particularly airlines—to make informed, data-backed decisions to improve customer service quality.
The methodology is divided into four major components: use case analysis, architectural and model design, and implementation strategy. The application offers dual functionality, allowing users to submit individual passenger data for real-time predictions or upload structured datasets in CSV format to perform batch prediction and analysis.

2 Use cases

The developed system has been architected to support both users and data analysts within the airline industry. Two principal use cases were identified to ensure practical applicability and broad utility across different user groups.

2.1 Use case 1: Real-Time Prediction for Individual Passengers

This use case is designed to enable real-time satisfaction prediction at the individual passenger level. Users interact with a structured web-based form, where they input key passenger and flight-related attributes—such as travel class, age bracket, flight distance, and ratings of in-flight services—that correspond to the features used in model training.

Input: Manually entered passenger data via a structured user interface.

Processing: Input features are preprocessed and passed through the trained machine learning pipeline.

Output: A satisfaction prediction (e.g., Satisfied or Not Satisfied), accompanied by a confidence score indicating the model's certainty.

This use case is particularly suited for customer-facing applications, such as personalized service recommendations or post-flight feedback analysis.

2.2 Use Case 2: Batch-Level Prediction via CSV Upload

Targeted at analysts and strategic decision-makers, this use case facilitates large-scale prediction across multiple passengers through a batch-processing mechanism.

Input: A CSV file containing structured records of passenger attributes.

Processing: The input file undergoes parsing, validation, and transformation. Each record is then processed in parallel using PySpark, leveraging the pre-trained classification model.

Output: A downloadable CSV file containing the predicted satisfaction labels for each passenger, in addition to summary visualizations (e.g., pie charts, bar plots) that depict aggregate satisfaction trends and insights.

This use case enables macro-level evaluation, making it well-suited for business reporting, service optimization, and data-driven strategic planning.

3 Design

3.1 System Architecture

- The application architecture follows a modular and layered design to promote scalability, maintainability, and performance. The core layers of the system include:
- Data Preprocessing Layer: Raw data undergoes cleaning, normalization, feature encoding, and transformation using PySpark's DataFrame and MLlib APIs. This layer ensures data quality and model readiness.
- Model Training and Evaluation Layer: Several classification models are implemented using Spark MLlib, including Logistic Regression, Gradient Boosted Trees, and Random Forest. Hyperparameter tuning and cross-validation are applied to identify the optimal model.
- Model Deployment Layer: The final model is serialized and deployed within the web application. It is loaded dynamically to make predictions on user-supplied data.
- Presentation Layer (Web Interface): Developed using Flask/Django, the front-end supports real-time and batch input, displays results, and offers interactive dashboards.

3.2 Data Flow and Pipeline

- The system follows a structured data pipeline composed of the following stages:
- Data Ingestion: Input is either entered via form fields or uploaded as a CSV file.
- Preprocessing: Features are cleaned, encoded, and normalized. Null values are handled, and irrelevant attributes are dropped.
- Feature Engineering: Derived features are computed based on domain knowledge.
- Model Prediction: Processed data is passed to the deployed model which outputs a results of satisfaction. Visualization and Export: Predictions are visualized in the dashboard and/or written to a downloadable CSV file.

3.3 Model Selection and Evaluation

- All candidate models were evaluated using standard classification metrics such as:
- Accuracy: Overall correctness of predictions
- Precision and Recall: Evaluated especially for the "Satisfied" class to minimize false positives/negatives
- F1 Score: Harmonic mean of precision and recall for balanced evaluation
- Area Under the Curve (AUC): Measure the discrimination capacity of the model

III Implementation and Results

1 Installing and importing packages

1.1 Download dataset

```
!gdown --id 1WZVnsI-92uLbSZG-AG10PW11_8oR-R-5
!unzip AirlinePassengerSatisfaction.zip -d AirlinePassengerSatisfaction
```

1.2 Install required packages

```
!pip install pyspark
!pip install findspark
!pip install pandas
```

1.3 Import libraries

```
import os
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import random

from pyspark.sql.functions import col, sum as spark_sum, when, collect_list,
    collect_set, udf, countDistinct, monotonically_increasing_id, row_number, rand
from pyspark.ml import Transformer, Pipeline
from pyspark.ml.param.shared import Param, Params, TypeConverters
from pyspark.sql.types import IntegerType, ArrayType, FloatType, DoubleType
from pyspark import keyword_only
from pyspark.ml.util import DefaultParamsReadable, DefaultParamsWritable
from pyspark.ml.linalg import VectorUDT
from pyspark.ml.feature import StandardScaler, VectorAssembler, Imputer, StringIndexer
    , RobustScaler, OneHotEncoder, PCA
from pyspark.ml import Pipeline, PipelineModel
from pyspark.sql.window import Window

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

2 Set up Spark and Read data

2.1 Initialize Spark session

```
import findspark
findspark.init()
from pyspark.sql import SparkSession

# Initialize a Spark Session
spark = SparkSession\
    .builder\
    .appName("AirlinePassengerSatisfaction")\
    .config("spark.sql.execution.arrow.pyspark.enabled", "true")\
```

```python
    .getOrCreate()

# Check if the Spark Session is active
if 'spark' in locals() and isinstance(spark, SparkSession):
    print("SparkSession is active and ready to use.")
else:
    print("SparkSession is not active. Please create a SparkSession.")
```

3 Exploratory Data Analysis

Load dataset

```python
# Load train file
train = spark.read.csv(
    "/content/AirlinePassengerSatisfaction/train.csv",
    header=True,
    inferSchema=True
)
# Load test file
test = spark.read.csv(
    "/content/AirlinePassengerSatisfaction/test.csv",
    header=True,
    inferSchema=True
)
```

Preview data

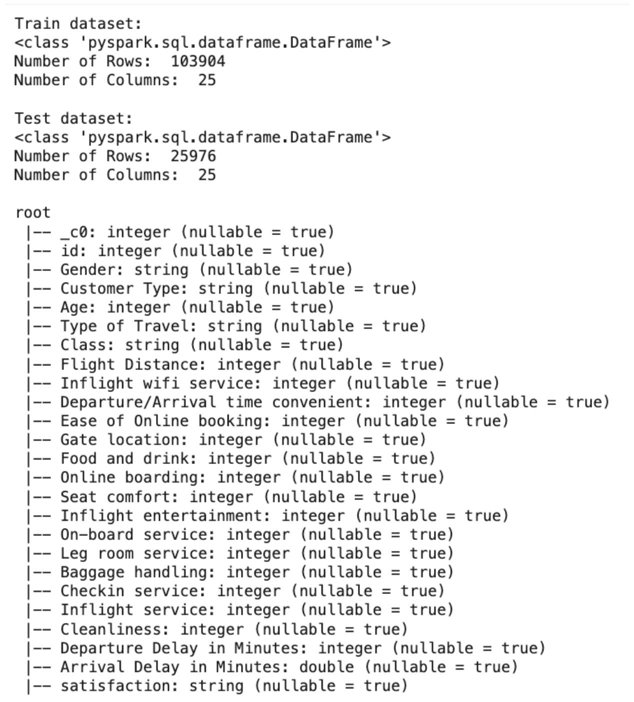
```python
train.show(5, truncate=False)
test.show(5, truncate=False)
```

3.1 Define utility functions

```python
# 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))

# 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)

# 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

# Sort the type of variables
def variable_type(df):
    types_list = df.dtypes
    str_variables = []
    int_variables = []
    for i in types_list:
        if i[1] == "string" and i[0]:
            str_variables.append(i[0])
        else:
            int_variables.append(i[0])
    return str_variables, int_variables
```

3.2 Basic dataset statistics

```python
# 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)
```

Figure III.3.1 Dataset overview

- The dataset comprises 129,880 passenger records, partitioned into a training set (103,904) and a test set (25,976), with 25 features each.
- It includes a balanced mix of categorical and numerical attributes capturing demographic details, travel characteristics, service ratings, and delay metrics.

- The target variable, satisfaction, is binary, making the dataset well-suited for classification tasks. While the schema indicates some nullable fields, overall data richness and volume provide a solid foundation for predictive modeling in airline customer satisfaction analysis.

Load data with pandas

```
# Load train file using pandas
pd_df = pd.read_csv(
    "/content/AirlinePassengerSatisfaction/train.csv"
)

# Load test file using pandas
test_df = pd.read_csv(
    "/content/AirlinePassengerSatisfaction/test.csv"
)
pd_df.head()
```

Correlation heatmap

```
plt.figure(figsize=(12, 8))
sns.heatmap(pd_df.corr(numeric_only=True), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap of Numeric Features")
plt.tight_layout()
plt.show()
```

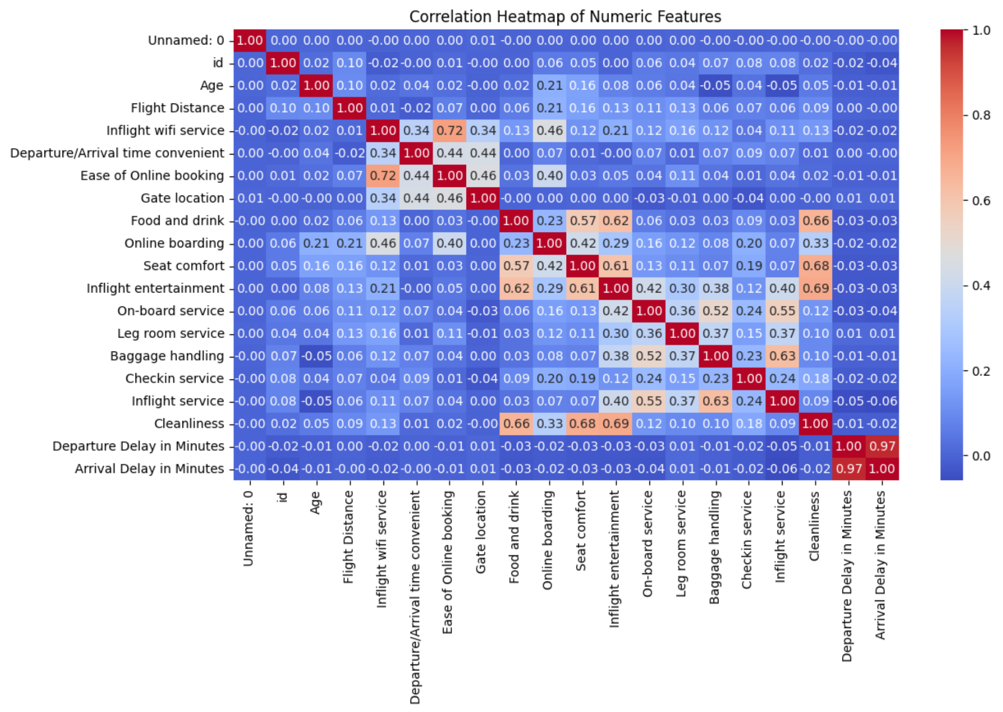


Figure III.3.2 Correlation between Numeric Features

Bar charts

```python
# Feature Selection
categorical_variables_str = ['Gender', 'Customer Type', 'Type of Travel', 'Class']
categorical_variables_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']

cols_to_plot = categorical_variables_str + categorical_variables_int
n_plots = len(cols_to_plot)
rows, cols = 4, 5

fig, axes = plt.subplots(rows, cols, figsize=(20, 16))  # Adjust figsize as needed
axes = axes.flatten()

for i, col in enumerate(cols_to_plot):
    value_counts = pd_df[col].value_counts(dropna=False).sort_index()
    value_counts.plot(kind='bar', ax=axes[i])
    axes[i].set_title(f"{col}", fontsize=10)
    axes[i].set_ylabel("Count")
    axes[i].set_xlabel(col)
    axes[i].tick_params(axis='x', labelrotation=0)

# Hide unused subplots if any
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```

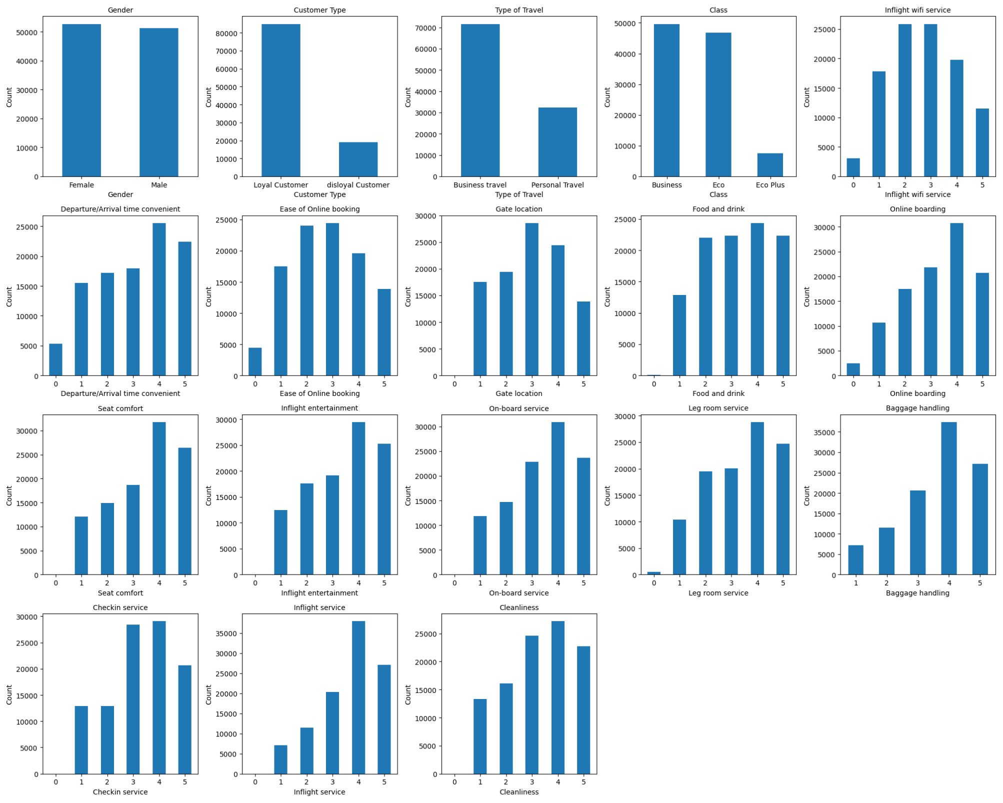


Figure III.3.3 Distribution of selected categorical features

Pie charts

```python
cols_to_plot = categorical_variables_str + categorical_variables_int
n_plots = len(cols_to_plot)
rows, cols = 4, 5

fig, axes = plt.subplots(rows, cols, figsize=(20, 16))
axes = axes.flatten()

for i, col in enumerate(cols_to_plot):
    value_counts = pd_df[col].value_counts(dropna=False).sort_index()
    axes[i].pie(value_counts, labels=value_counts.index, autopct='%1.1f%%', startangle
        =90)
    axes[i].set_title(f"{col}", fontsize=10)

# Hide unused subplots
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```

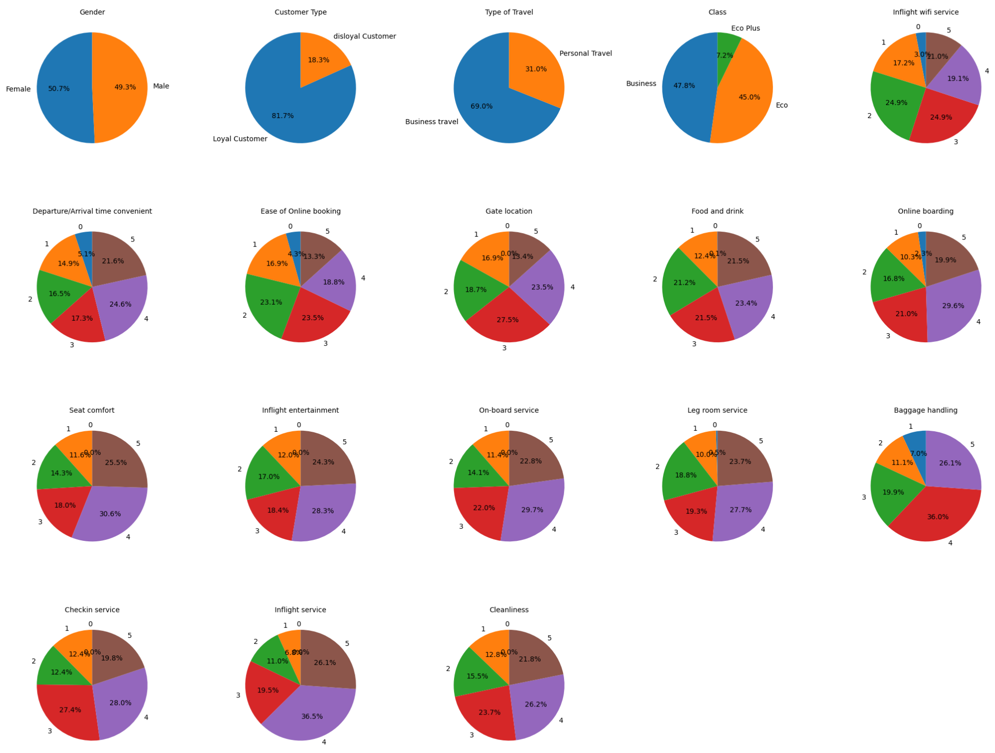


Figure III.3.4 Distribution of categorical variables

Some conclusions about the considered sample:

- The sample contains a roughly equal number of male and female passengers.
- A significant majority of customers are returning or loyal flyers.
- Most passengers traveled for business rather than leisure purposes.
- Approximately half of the travelers were seated in business class.
- Over 60 percent of passengers expressed satisfaction with the baggage handling service, giving it a rating of 4 or 5 out of 5.
- More than half of the passengers found their seats to be comfortable, rating them 4 or 5 out of 5.

Class vs satisfaction

```
sns.countplot(x = 'Class', hue = 'satisfaction', data = pd_df)
plt.show()
```

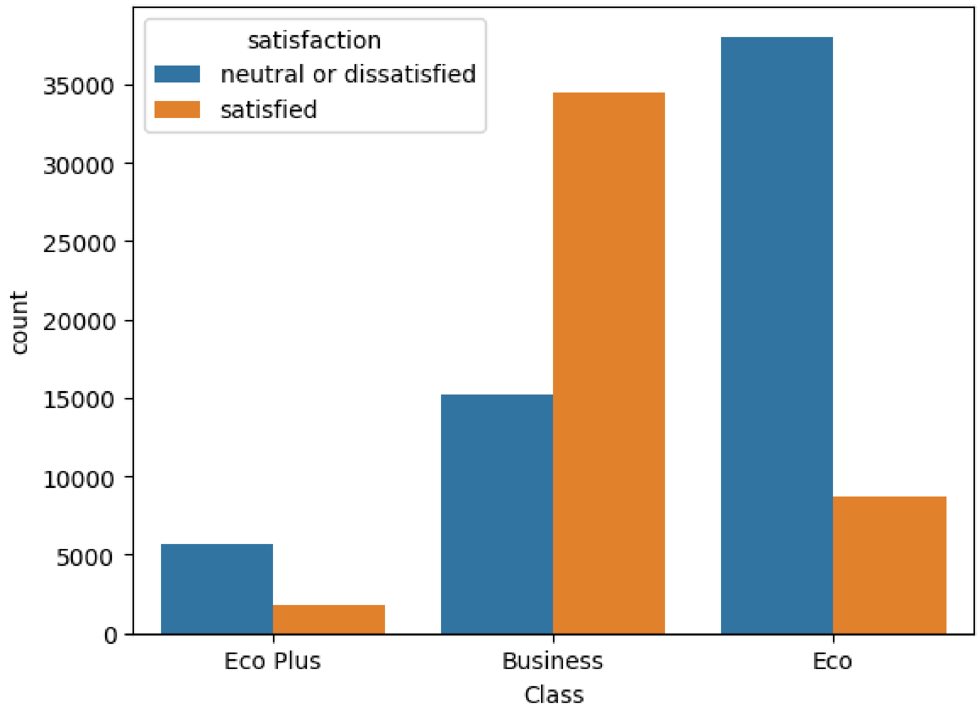


Figure III.3.5 Correlation between "Class" and "Satisfaction"

Age distribution

```python
plt.figure(figsize=(16, 9))
value_counts = pd_df["Age"].value_counts(dropna=False).sort_index()
value_counts.plot(kind='bar')
plt.title(f"Distribution of Age")
plt.ylabel("Count")
plt.xlabel(col)
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```

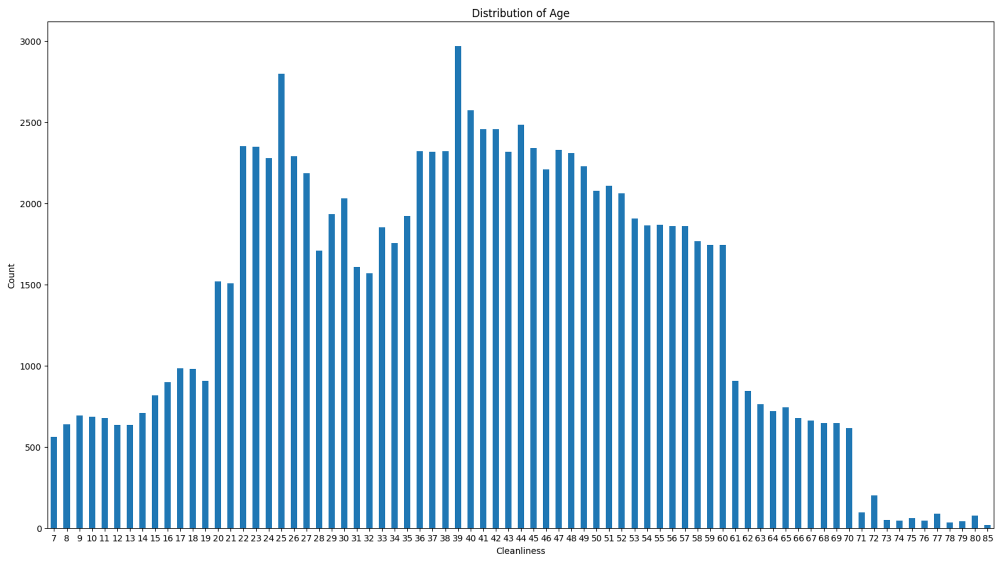


Figure III.3.6 Age Distribution

Delay correlation

```python
plt.scatter(pd_df['Arrival Delay in Minutes'], pd_df['Departure Delay in Minutes'],
    alpha = 0.5)
```

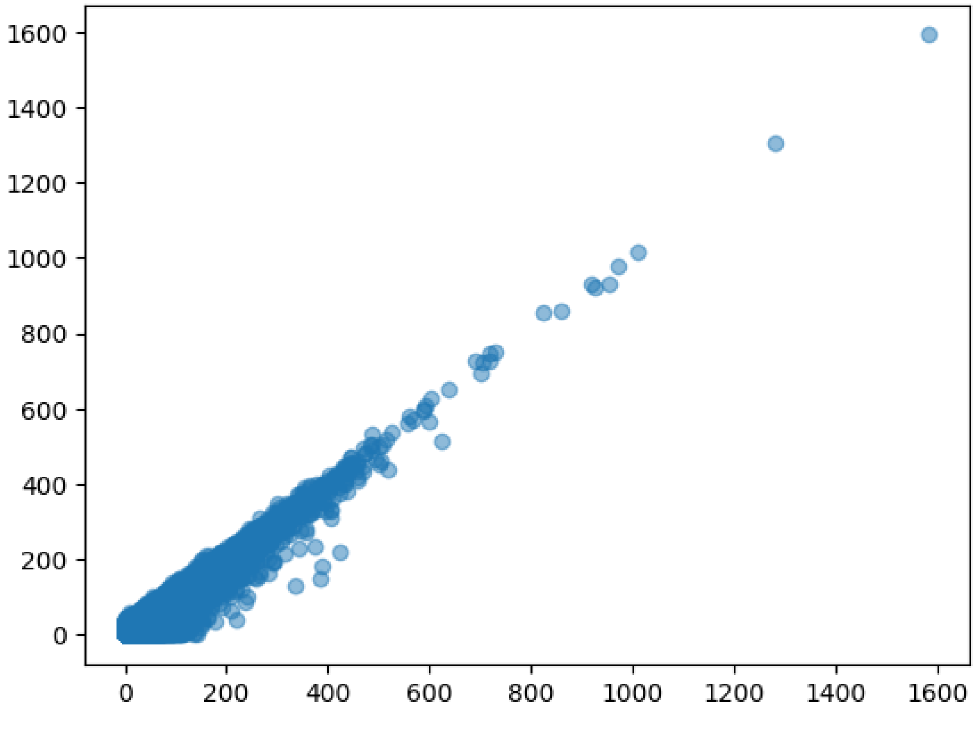


Figure III.3.7 Delay Correlation

Test dataset analysis

```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

# Clean and prepare data
df_clean = test.toPandas()
df_clean.head()
```

3.3 Advanced visualizations

```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

# Clean and prepare data
df_clean = test.toPandas()

# Drop unnecessary columns
df_clean.drop(columns=['_c0', 'id'], inplace=True)

# Check and fill missing values in 'Arrival Delay in Minutes'
df_clean['Arrival Delay in Minutes'].fillna(df_clean['Arrival Delay in Minutes'].
    median(), inplace=True)

# Step A: Customer Profile Analysis
# Satisfaction by Gender, Customer Type, Class, Type of Travel
profile_vars = ['Gender', 'Customer Type', 'Class', 'Type of Travel']

profile_summary = {}
for var in profile_vars:
    profile_summary[var] = df_clean.groupby([var, 'satisfaction']).size().unstack().
        fillna(0)

# Age distribution by satisfaction
age_dist = df_clean[['Age', 'satisfaction']]

# Step B: Service Ratings vs Satisfaction
service_columns = [
    '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'
]

service_summary = df_clean.groupby('satisfaction')[service_columns].mean().T
service_summary['Difference'] = service_summary['satisfied'] - service_summary['
    neutral or dissatisfied']
service_summary_sorted = service_summary.sort_values(by='Difference', ascending=False)

# Step C: Delay analysis
delay_summary = df_clean.groupby('satisfaction')[['Departure Delay in Minutes', '
    Arrival Delay in Minutes']].mean()

```

```
# Set plot style
sns.set(style="whitegrid")
plt.figure(figsize=(10, 6))

# 1. Customer Profile Plots
fig1, axes = plt.subplots(2, 2, figsize=(14, 10))
profile_vars = ['Gender', 'Customer Type', 'Class', 'Type of Travel']

for ax, var in zip(axes.flat, profile_vars):
    sns.countplot(data=df_clean, x=var, hue='satisfaction', ax=ax)
    ax.set_title(f'Satisfaction by {var}')
    ax.set_ylabel("Passenger Count")
    ax.set_xlabel(var)
    ax.tick_params(axis='x', rotation=15)

plt.tight_layout()
plt.show()

# 2. Service Rating Difference Plot
plt.figure(figsize=(12, 8))
sns.barplot(
    y=service_summary_sorted.index,
    x=service_summary_sorted['Difference'],
    palette='coolwarm'
)
plt.axvline(0, color='gray', linestyle='--')
plt.title('Average Service Rating Difference: Satisfied - Dissatisfied')
plt.xlabel('Rating Difference')
plt.ylabel('Service Category')
plt.tight_layout()
plt.show()

# 3. Delay Analysis Plot
plt.figure(figsize=(8, 5))
delay_summary.T.plot(kind='bar', figsize=(10, 5), colormap='viridis')
plt.title('Average Delay by Satisfaction Group')
plt.ylabel('Average Delay (minutes)')
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```

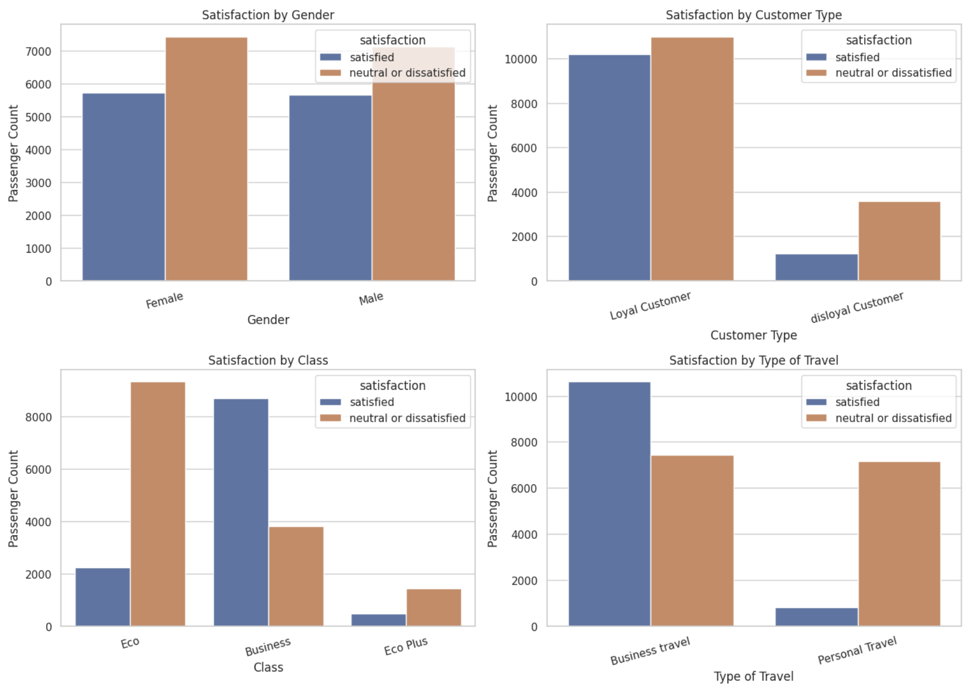


Figure III.3.8 Passenger satisfaction across some categories.

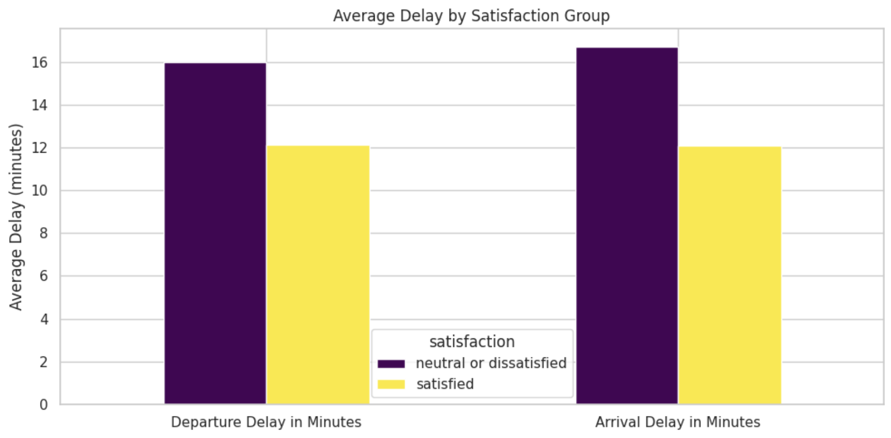


Figure III.3.9 Average delays by satisfaction group

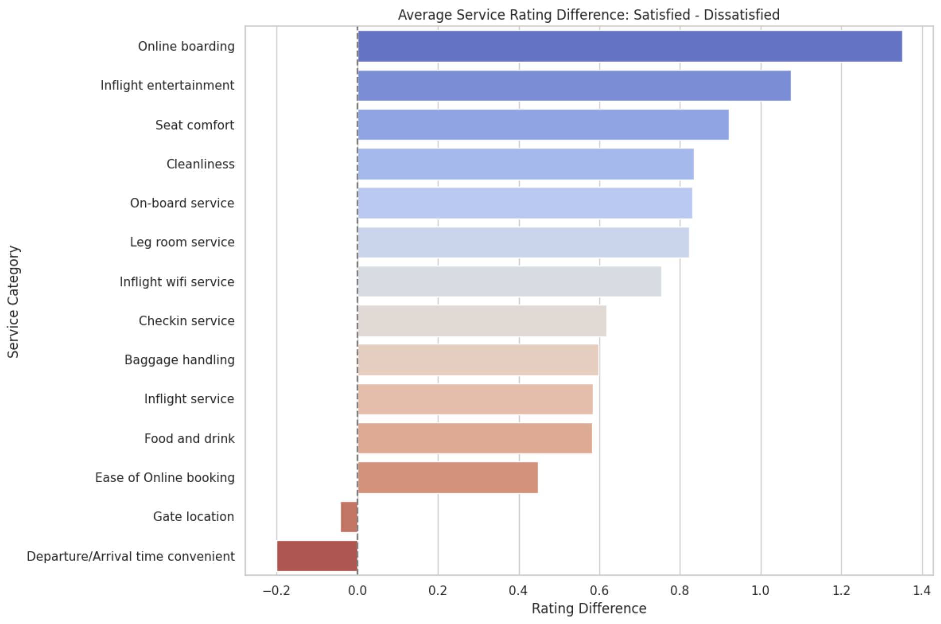
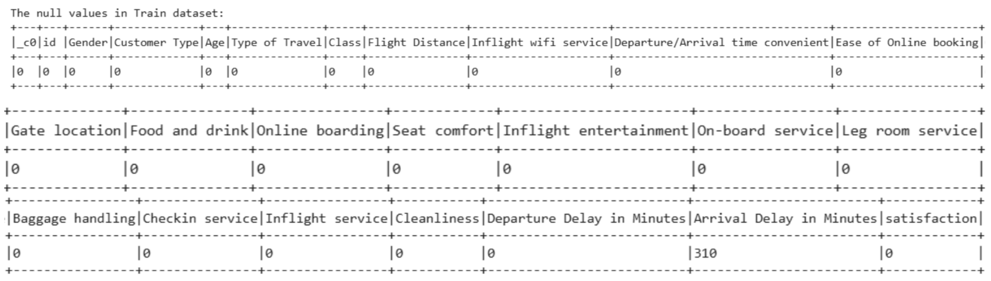


Figure III.3.10 Average service ratings between satisfied and dissatisfied

3.4 Checking null value

```python
from pyspark.sql.functions import col, sum as spark_sum
# Checking null values of Train dataset
print("The null values in Train dataset:")
count_null(train)
```



```
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                                |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           |
+----------------+---------------+----------------+-----------+--------------------------+------------------------+------------+
```

Figure III.3.11 Null Value

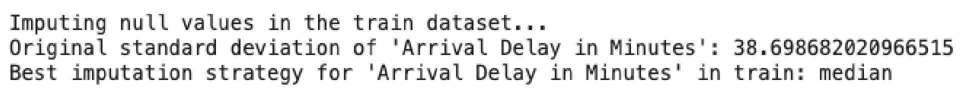
4 Data Pre-Processing

4.1 Imputation strategy

```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
```

4.2 Handle null values

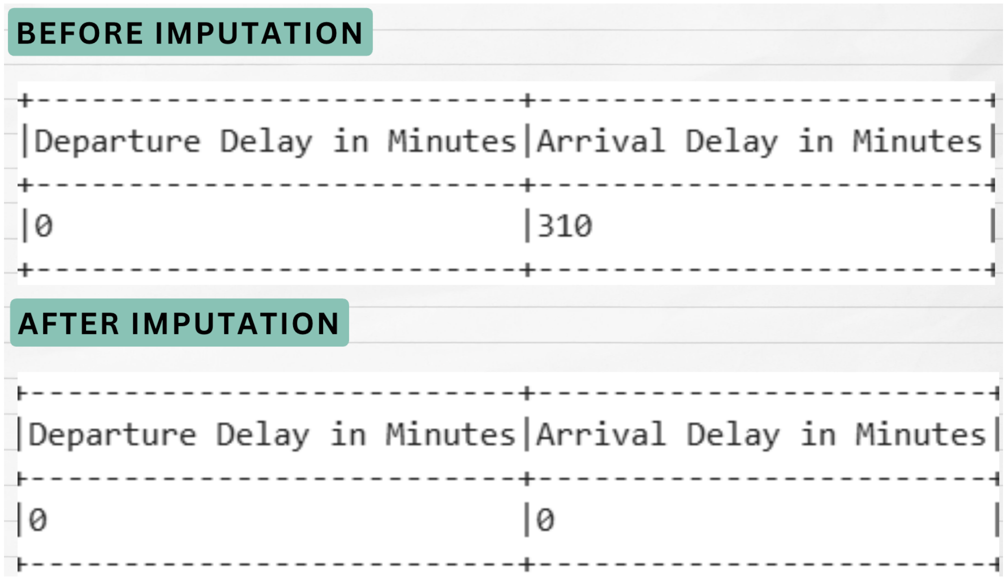
```
# Columns with null values identified from previous count_null output
columns_to_impute = ['Arrival Delay in Minutes']

# 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
```

Figure III.4.1 Filling with Median

Following imputation, all missing values were resolved, and the dataset is now free of null entries:



BEFORE IMPUTATION

```
+--------------------------+--------------------------+
|Departure Delay in Minutes|Arrival Delay in Minutes|
+--------------------------+--------------------------+
|0                         |310                       |
+--------------------------+--------------------------+
```

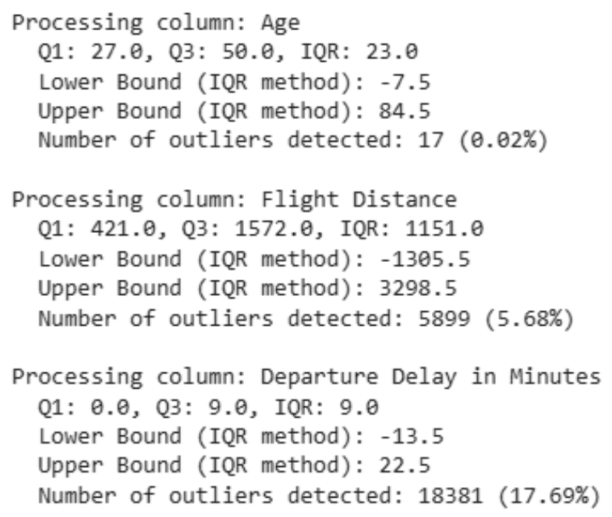
AFTER IMPUTATION

```
+--------------------------+--------------------------+
|Departure Delay in Minutes|Arrival Delay in Minutes|
+--------------------------+--------------------------+
|0                         |0                         |
+--------------------------+--------------------------+
```

Figure III.4.2 After Imputation

4.3 Outlier detection

```python
# 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
```



```
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%)
```

Figure III.4.3 Outliers

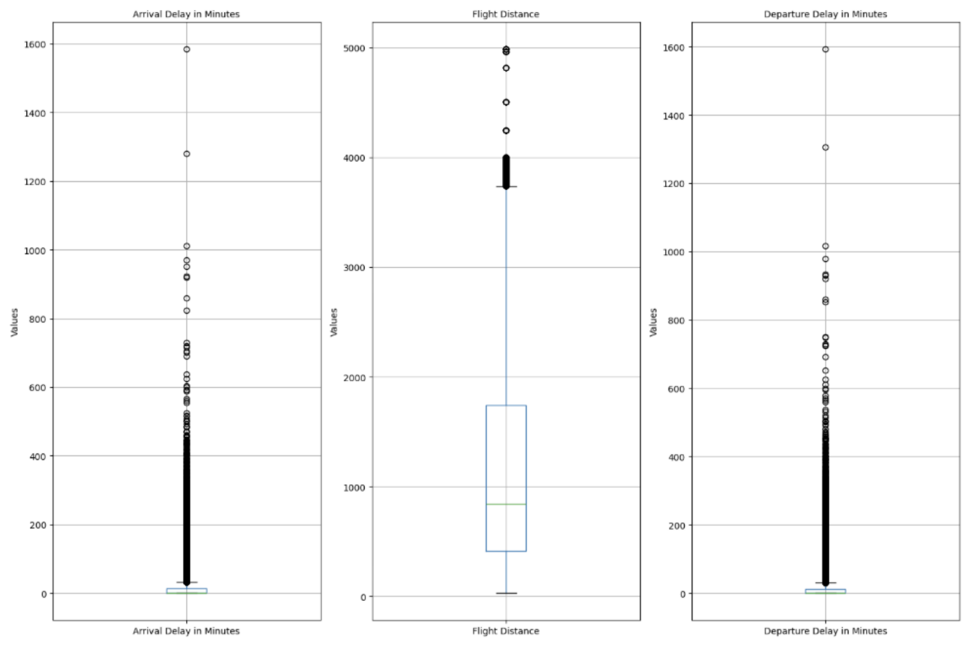


Figure III.4.4 Box plot

4.4 Data pipeline

```python
numeric_columns_for_outlier_check = ['Age', 'Flight Distance', 'Departure Delay in
    Minutes', 'Arrival Delay in Minutes']

total_column = numeric_columns_for_outlier_check

# Apply outlier replacement to the selected numerical columns in the training data
for col_name in total_column:
    train = check_outlier(train, col_name)
```

5 Model

5.1 Prepare dataset

```
df = train

# Drop columns id and numbers
columns_to_drop = ["_c0"]
df = df.drop(*columns_to_drop)

# Displaying basic statistics
df.describe().show(truncate=False)

# Checking Distinct values in train dataset
print("Distinct values check: ")
df.select([countDistinct(c) for c in df.columns]).show()

#Checking unique values
unique_values_dict = unique_values(df)
for column, values in unique_values_dict.items():
    print(f"Column: {column}, Unique Values: {values}")

# Show dataframe
df.show(5, truncate=False)
```

5.2 Feature selection

```
# Feature Selection
target_variable = ["satisfaction"]
numeric_variables = ["Age", "Flight Distance", "Departure Delay in Minutes", "Arrival
    Delay in Minutes"]
categorical_variables_str = [col for col in variable_type(df)[0] if col != "
    satisfaction"]
categorical_variables_int = [
    c for c in df.columns
        if c not in numeric_variables + categorical_variables_str + ["_c0", "id"] and
            c != "satisfaction"
]

print("Categorical variables in int: ", categorical_variables_int)
print("Categorical variables in str: ", categorical_variables_str)
print("Numeric variables: ", numeric_variables)
print("Target variable: ", target_variable)
```

Data preview

```
df.show(truncate=False)
```

5.3 Feature engineering

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

# 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
)

# 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")

# 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"
                            )

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

# Create Pipeline
pipeline = Pipeline(stages=[
    target_indexer,
    imputer,
    *str_indexers,
    str_encoder,
    vec_age, scaler_age,
    vec_distance, scaler_distance,
    vec_depart, scaler_depart,
    vec_arrive, scaler_arrive,
    assembler
])
```

Both **StandardScaler** and **RobustScaler** are used to normalize numerical features to enhance the performance of machine learning models. While StandardScaler standardizes features by removing the mean and scaling to unit variance, it is sensitive to outliers. In contrast, RobustScaler relies on the median and interquartile range (IQR), making it more resilient to extreme values.

In this project, several features—such as flight distance and delay times—exhibit skewed distributions and potential outliers. Therefore, **RobustScaler** is more appropriate, as it minimizes the influence of these outliers during normalization, leading to more stable and reliable model training.

5.4 Processed data

```
# Dataframe Processed
pipeline_model = pipeline.fit(df) # Fit pipeline on original train data
pipeline_model.write().overwrite().save("pipeline_model")

# Transform both train and test original dataframes
train_processed = pipeline_model.transform(df)

# Select the features and label column for the LR model
train_final = train_processed.select("id", "features", col("label").cast(DoubleType())
    .alias("label"))

# Show
train_final.show(5, truncate=False)
```

Save pipeline

```
import shutil
from google.colab import files

# Create a zip archive of the folder
shutil.make_archive('/content/pipeline_model', 'zip','/content/pipeline_model')

# Download the zipped folder
files.download("pipeline_model.zip")
```

Train-test split

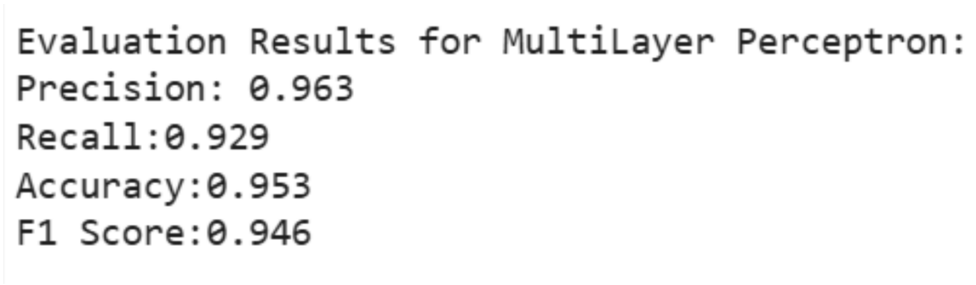
```
# Split train,test from the processed train dataset
train_data, test_data = train_final.randomSplit([0.8, 0.2], seed=42)
```

Evaluation metrics

```
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()

    # 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
    }
```

5.5 Multilayer Perceptron

```
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}")

cv_mlp_model.write().overwrite().save("/content/models/Tuned_MLP")
```



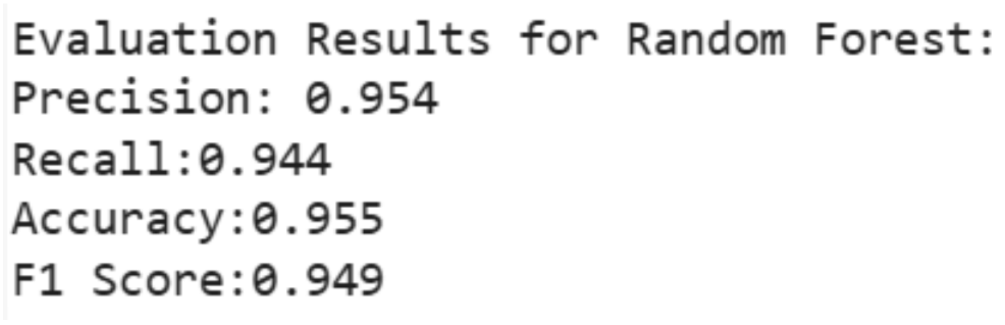
```
Evaluation Results for MultiLayer Perceptron:
Precision: 0.963
Recall:0.929
Accuracy:0.953
F1 Score:0.946
```

Figure III.5.1 Results for Multilayer Perceptron

- Strong classification performance: The tuned MultiLayer Perceptron (MLP) model demonstrated excellent classification ability with an AUC score of 0.952, indicating a high capability to distinguish between satisfied and dissatisfied passengers.
- High precision: A precision of 0.960 shows that the model is highly accurate in predicting satisfied passengers, minimizing false positives.
- Good recall: With a recall of 0.930, the model successfully identifies most truly satisfied passengers, showing strong sensitivity.
- Balanced prediction: The F1 score of 0.944 reflects a well-balanced trade-off between precision and recall, ensuring both accuracy and coverage.
- Conclusion: These consistently high evaluation metrics confirm that the MLP model is a reliable and effective choice for predicting airline passenger satisfaction, especially in scenarios requiring a balance between detecting true positives and avoiding false alarms.

5.6 Random Forest

```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}")

# Save model
cv_rf_model.write().overwrite().save("/content/models/Tuned_RF")
```



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

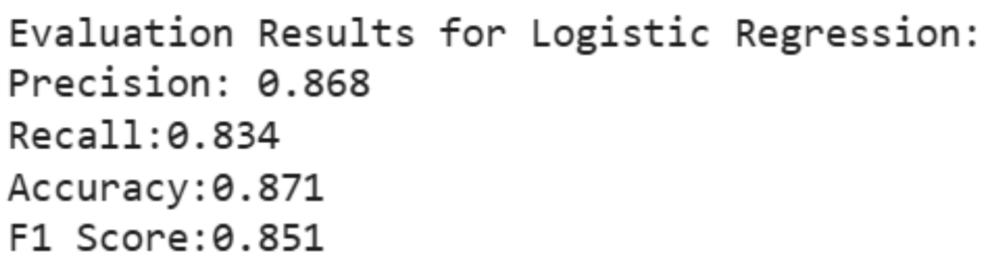
Figure III.5.2 Results for Random Forest

- Strong overall performance: The tuned Random Forest classifier achieved a high AUC score of 0.956, indicating excellent discrimination between the satisfied and dissatisfied passenger groups.
- High precision: With a precision of 0.955, the model makes highly accurate positive predictions, effectively minimizing false positives.
- Excellent recall: The recall score of 0.943 suggests that the model successfully identifies the majority of satisfied passengers, ensuring low false negatives.
- Balanced and reliable: The F1 score of 0.949 reflects a well-balanced model with both strong precision and recall, making it reliable for real-world deployment.

- Conclusion: These evaluation metrics demonstrate that the tuned Random Forest model is highly effective for predicting passenger satisfaction and slightly outperforms the MLP model in terms of overall accuracy and F1 score. It is a strong candidate for production use in this classification task.

5.7 Logistic Regression

```
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()

# 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}")

cv_lr_model.write().overwrite().save("/content/models/Tuned_LR")
```



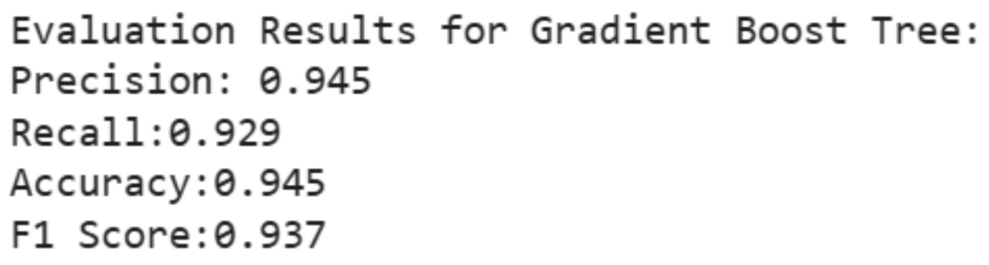
```
Evaluation Results for Logistic Regression:
Precision: 0.868
Recall:0.834
Accuracy:0.871
F1 Score:0.851
```

Figure III.5.3 Results for Logistic Regression

- Moderate performance: The tuned Logistic Regression model yielded an AUC of 0.871, reflecting a reasonable ability to distinguish between satisfied and dissatisfied passengers, though lower than tree-based models.
- Good precision: Achieving a precision of 0.868, the model reliably identifies positive (satisfied) cases, minimizing false positives.
- Lower recall: The recall of 0.834 suggests the model misses a higher number of true satisfied passengers compared to other classifiers, which might be a limitation in certain contexts.
- Balanced F1 Score: The F1 Score of 0.851 shows a fair balance between precision and recall, though again not as strong as more complex models.
- Conclusion: Logistic Regression provides interpretability and efficiency, making it suitable for simpler, explainable modeling needs. However, its performance is relatively weaker than models like Random Forest or MLP, especially in handling complex relationships within the data.

5.8 Gradient Boost Tree

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

gbt = GBTClassifier(
    featuresCol="features",
    labelCol="label",
    seed=42
)

# Parameter grid for tuning
paramGrid_gbt = ParamGridBuilder()\
    .addGrid(gbt.maxIter, [10, 20])\
    .addGrid(gbt.maxDepth, [4, 6])\
    .build()

# Reuse evaluator
evaluator = BinaryClassificationEvaluator(
    labelCol="label",
    rawPredictionCol="rawPrediction",
    metricName="areaUnderROC"
)

# CrossValidator
cv_gbt = CrossValidator(
    estimator=gbt,
    estimatorParamMaps=paramGrid_gbt,
    evaluator=evaluator,
    numFolds=10,
    parallelism=10,
    seed=42
)

# Train tuned GBT model
cv_gbt_model = cv_gbt.fit(train_data)

# Predict and evaluate
gbt_predictions = cv_gbt_model.transform(test_data)
gbt_auc = evaluator.evaluate(gbt_predictions)
print(f"Tuned Gradient Boosting Tree AUC: {gbt_auc:.4f}")

# Save model
cv_gbt_model.write().overwrite().save("/content/models/Tuned_GBT")
```



```
Evaluation Results for Gradient Boost Tree:
Precision: 0.945
Recall:0.929
Accuracy:0.945
F1 Score:0.937
```

Figure III.5.4 Results for Gradient Boost Tree

- Competitive performance: The tuned Gradient Boosted Tree model achieved an AUC of 0.945, indicating strong overall classification capability and good separation between classes.
- High precision: With a precision of 0.948, the model produces accurate positive predictions, limiting the number of false alarms.
- Solid recall: A recall score of 0.924 shows the model successfully identifies most satisfied passengers, though slightly lower than other models like Random Forest.
- Reliable balance: The F1 Score of 0.936 indicates a well-balanced model, combining precision and recall effectively for dependable performance.
- Conclusion: While slightly trailing behind the Random Forest in terms of overall metrics, the GBT model still performs strongly and can be a robust alternative. It's particularly useful when interpretability and regularization are important in tree-based models.

Based on the evaluation metrics across all examined models, the Random Forest classifier demonstrates the most balanced and superior performance:

- It achieves the **highest accuracy (0.956)** among all evaluated models.
- It exhibits **strong precision (0.955)**, indicating a low rate of false positives.
- It attains a **high recall (0.943)**, effectively capturing the majority of true positive instances.
- Its F1 score (0.949) further confirms consistent performance across different classes.
- Additionally, it records the highest Area Under the Curve (AUC), highlighting its robust discriminatory capability.
- Compared to the Gradient Boosted Trees and Multilayer Perceptron models, Random Forest not only maintains competitive accuracy but also offers advantages in interpretability, resilience to noise, and reduced training time, especially when applied to large-scale datasets.

$\Longrightarrow$ We select the Random Forest classifier as the primary model for our project, considering its excellent overall performance and practical benefits.

IV User Interface

1 Streanlit librabry

```
# Download lib streamlit
!pip install streamlit plotly -q # Added plotly dependency
```

Load pipeline model

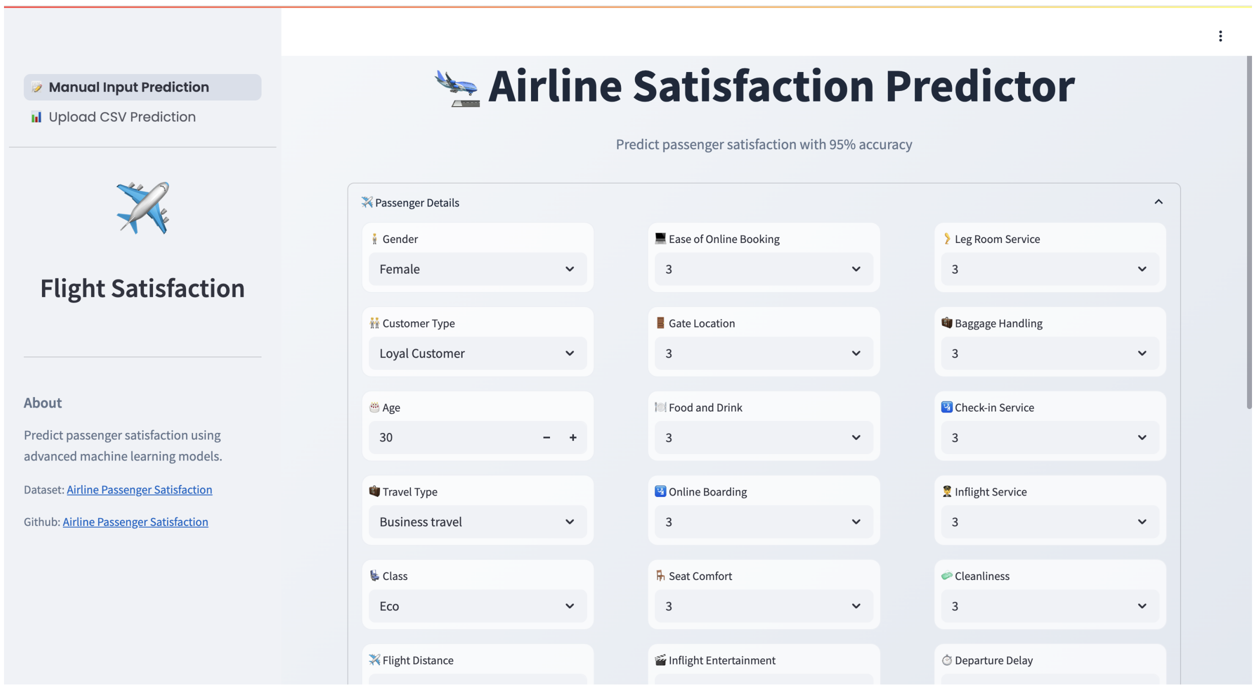
```
!gdown --id 1LS2bavs0aZlW1pJ5A-u1DSD6UmDTRFcr
!unzip pipeline_model.zip -d pipeline_model
```

Load prediction models

```
!gdown --id 1RoZ9nk18YnRXi2VzLcrLH6BNGx-PN2Te
!unzip models.zip -d models
```

2 Demo 1: Manual Input Prediction

Designed for end-users, this interface allows users to manually input key passenger and flight-related information (e.g., age, travel class, service ratings). Upon submission, the system processes the data and immediately returns a prediction indicating whether the passenger is likely to be satisfied or not.



Predict passenger satisfaction using advanced machine learning models.

Dataset: Airline Passenger Satisfaction

Github: Airline Passenger Satisfaction

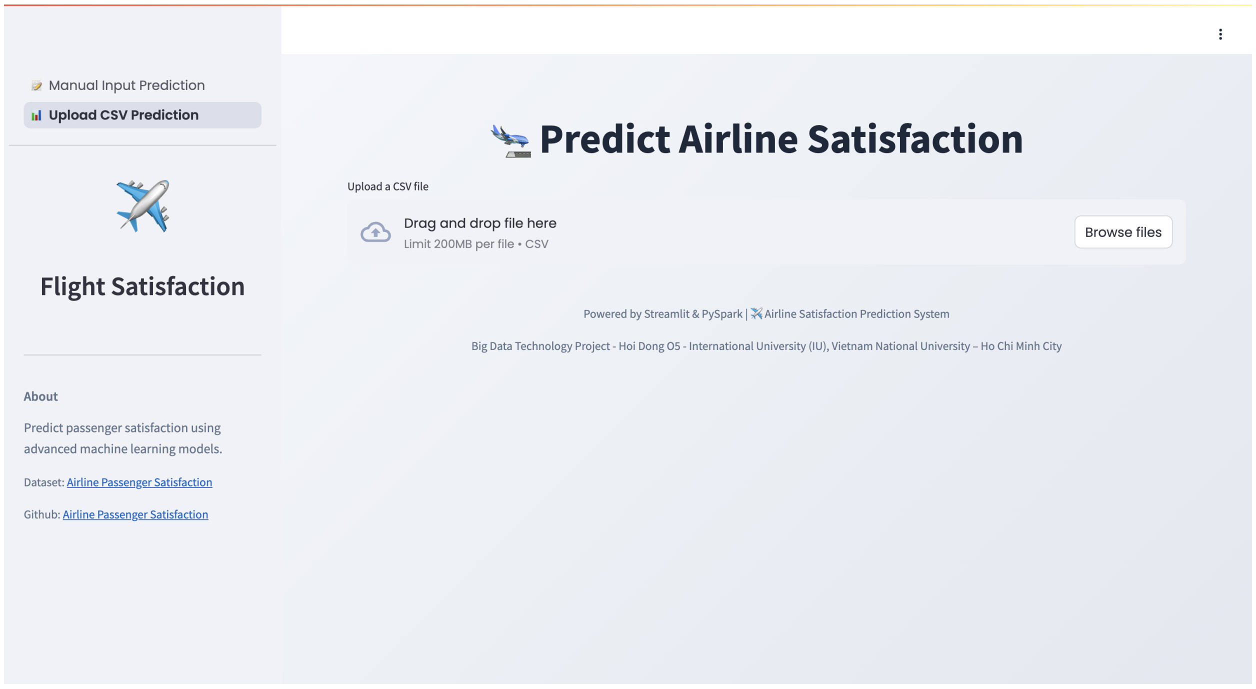
Predict passenger satisfaction with 95% accuracy

Gender: Female	Ease of Online Booking: 3	Leg Room Service: 3
Customer Type: Loyal Customer	Gate Location: 3	Baggage Handling: 3
Age: 30	Food and Drink: 3	Check-in Service: 3
Travel Type: Business travel	Online Boarding: 3	Inflight Service: 3
Class: Eco	Seat Comfort: 3	Cleanliness: 3
Flight Distance	Inflight Entertainment	Departure Delay

Figure IV.2.1 The Interface for Manual Input Prediction

3 Demo 2: CSV Prediction

Tailored for analysts or operational staff, this interface enables users to upload a structured CSV file containing multiple passenger records. The system then performs batch processing to generate satisfaction predictions for each entry and returns a downloadable file with results, along with summary visualizations.


Manual Input Prediction

Upload CSV Prediction

Flight Satisfaction

About

Predict passenger satisfaction using advanced machine learning models.

Dataset: Airline Passenger Satisfaction

Github: Airline Passenger Satisfaction

Predict Airline Satisfaction

Upload a CSV file

Drag and drop file here

Limit 200MB per file • CSV

Browse files

Powered by Streamlit & PySpark | ✈ Airline Satisfaction Prediction System

Big Data Technology Project - Hoi Dong 05 - International University (IU), Vietnam National University – Ho Chi Minh City

Figure IV.3.1 The Interface For CSV Uploaded Prediction

4 Web interface

```
%%writefile app.py
import streamlit as st
import numpy as np
import pandas as pd
from pyspark.sql import SparkSession
from pyspark.ml.tuning import CrossValidatorModel
from pyspark.ml import Pipeline, PipelineModel
from pyspark.ml.feature import VectorAssembler, StandardScaler, Imputer, StringIndexer
    , RobustScaler, OneHotEncoder, PCA
from pyspark.sql.functions import col, udf
from pyspark.sql.types import IntegerType, FloatType, ArrayType, StringType,
    DoubleType
from pyspark.ml.pipeline import Transformer
from pyspark.ml.util import DefaultParamsReadable, DefaultParamsWritable
from pyspark.ml.param.shared import Param, Params, TypeConverters
from pyspark.ml.linalg import DenseVector
from pyspark.ml.evaluation import BinaryClassificationEvaluator

# Start Spark
spark = SparkSession.builder.appName("CSV Prediction").getOrCreate()

# === Custom Transformers ===
class ScaledFeatureExpander(Transformer, DefaultParamsReadable, DefaultParamsWritable)
    :
    def __init__(self, inputCols=None, outputPrefix="_scaled"):
        super(ScaledFeatureExpander, self).__init__()
        self.inputCols = Param(self, "inputCols", "Input columns", typeConverter=
            TypeConverters.toListString)
        self.outputPrefix = Param(self, "outputPrefix", "Prefix for scaled columns",
            typeConverter=TypeConverters.toString)

        self._setDefault(outputPrefix="_scaled")
        if inputCols is not None:
            self._set(inputCols=inputCols)
        self._set(outputPrefix=outputPrefix)

    def _transform(self, df):
        inputCols = self.getOrDefault(self.inputCols)
        outputPrefix = self.getOrDefault(self.outputPrefix)

        assembler = VectorAssembler(inputCols=inputCols, outputCol="numeric_features")
        df = assembler.transform(df)

        scaler = RobustScaler(inputCol="numeric_features", outputCol="scaled_features")
        scaler_model = scaler.fit(df)
        df = scaler_model.transform(df)

        def unpack_vector(v):
            return v.toArray().tolist() if v is not None else [None]*len(inputCols)

        unpack_udf = udf(unpack_vector, ArrayType(FloatType()))
        df = df.withColumn("scaled_array", unpack_udf(col("scaled_features")))

        for i, col_name in enumerate(inputCols):
            df = df.withColumn(col_name + outputPrefix, col("scaled_array").getItem(i))

        return df.drop("numeric_features", "scaled_features", "scaled_array", *inputCols)

# Define the pages
demo_1 = st.Page("/content/demo_1.py", title="Manual Input Prediction", icon="      ")
demo_2 = st.Page("/content/demo_2.py", title="Upload CSV Prediction", icon="      ")

# Set up navigation
pg = st.navigation([demo_1, demo_2])

# Run the selected page
pg.run()
```

V Conclusion and Future Work

1 Conclusion

The Flight Passenger Satisfaction Prediction project has successfully developed a scalable and efficient predictive analytics system, employing PySpark and advanced machine learning methodologies to accurately forecast passenger satisfaction levels. This endeavor exemplifies the effective integration of big data processing paradigms with sophisticated predictive modeling within a distributed computing framework.

The resulting application establishes a robust platform for the analysis of extensive flight passenger datasets, delivering reliable satisfaction predictions that can inform airline strategies aimed at enhancing customer experience and operational efficiency. By synergizing scalable data processing capabilities, advanced machine learning algorithms, and an intuitive user interface, the project provides actionable insights into passenger behavior and key determinants of satisfaction.

Key Accomplishments

- Data Processing Pipeline: Constructed an end-to-end data ingestion and preprocessing pipeline utilizing PySpark, capable of efficiently managing voluminous flight data through comprehensive transformation and cleansing procedures.
- Feature Engineering: Developed and applied domain-specific feature extraction and encoding techniques to augment the predictive performance of the employed models.
- Machine Learning Models: Conducted training and rigorous evaluation of multiple classification algorithms, including Random Forest, Gradient Boosted Trees, and Logistic Regression, leveraging PySpark MLlib to ascertain the optimal predictive model for passenger satisfaction.
- Model Deployment: Seamlessly integrated the highest-performing model into a production-ready application framework, enabling real-time and batch-mode predictions on incoming passenger data.
- Interactive Visualization: Designed and implemented dashboards for the effective visualization of prediction outcomes and key performance metrics, thereby facilitating data-driven decision-making.

Technical Implementation

- Distributed Data Processing: Employed PySpark's Resilient Distributed Datasets (RDDs) and DataFrame APIs to ensure scalable and efficient data transformations and analytical operations.
- Machine Learning Pipeline: Utilized Spark MLlib's pipeline architecture for streamlined model training, validation, and hyperparameter optimization.
- Application Architecture: Adopted a modular design approach to distinctly separate data processing, model inference, and user interface components, thereby enhancing maintainability and scalability.
- Data Integration: Supported heterogeneous data sources, including CSV and Parquet formats, with automated ingestion workflows and strict schema enforcement.
- User Interface: Developed a responsive and accessible web interface using a Python-based web framework (using Streamlit) to facilitate seamless interaction with the prediction engine.

Security Features

- Access Control: Implemented robust authentication protocols to restrict system access, safeguarding prediction services and sensitive data assets.
- Data Privacy: Enforced data anonymization and encryption standards to protect passenger information throughout data processing and storage stages.
- Input Validation: Incorporated comprehensive validation mechanisms to prevent injection of malformed or malicious data, thereby preserving model integrity and system reliability.

Code Quality

- Modularity: Organized the codebase into discrete, reusable modules encompassing data processing, model training, and application logic.
- Documentation: Maintained thorough inline annotations and external documentation to support codebase comprehension and facilitate future enhancements.
- Testing: Executed extensive unit and integration testing to ensure the correctness, robustness, and reliability of system components.

User Experience

- Intuitive Interface: Engineered a user-centric interface characterized by clarity and ease of navigation for data submission and retrieval of predictive insights.
- Real-Time Feedback: Provided immediate and informative prediction results accompanied by contextual status indicators and visualizations.

- Data Insights: Delivered interactive visual analytics and summary statistics to elucidate model predictions and underlying data patterns.
- Workflow Optimization: Streamlined processes related to data upload, prediction generation, and report production to enhance overall usability.

2 Future Work

- Although the Flight Passenger Satisfaction Prediction project has laid a strong foundation, several avenues exist for further advancement and refinement:
- Incorporation of Additional Data Sources: Integration of supplementary data streams, such as flight delay records, meteorological information, and social media sentiment analysis, to enrich the feature space and improve predictive accuracy.
- Advanced Modeling Techniques: Exploration of deep learning architectures and ensemble learning strategies to achieve enhanced model performance.
- Explainability and Interpretability: Deployment of model interpretability frameworks to provide transparent and actionable explanations of prediction outcomes for domain experts and stakeholders.
- Scalability and Performance Optimization: Enhancement of distributed computing resource allocation and pipeline efficiency to accommodate growing data volumes and enable real-time analytics.
- Mobile and API Integration: Development of mobile-compatible interfaces and RESTful APIs to extend accessibility and facilitate integration with existing airline information systems.
- Continuous Learning: Implementation of incremental learning methodologies with streaming data support to adaptively update models in response to evolving passenger behavior patterns.
- Pursuing these future directions will augment the application's utility as a strategic analytical tool, empowering airlines to leverage data-driven insights for the continual improvement of passenger satisfaction.

VI References

- Kaggle: Airline Passenger Satisfaction Dataset: `https://www.kaggle.com/datasets/teejmahal20/airline-passenger-sa`
- StandardScaler, MinMaxScaler and RobustScaler techniques: `https://www.geeksforgeeks.org/standardscaler-minmaxscal`
- Streamlit: A faster way to build and share data apps :`https://streamlit.io/`
- Project Slide: `https://www.canva.com/design/DAGojJZa9Ww/g9n9kW8unIup0kFBcGGJRQ/edit?fbclid= IwY2xjawKmsadleHRuA2FlbQIxMABicmlkETFoRjdIamNqMUZoZldVd2FnAR7hPvsAw1eY-K-6W_5eHAgQy3CzXrqWouBz2V aem_HDpavs6aAvpLq2rDITOg4Q`
- Our Project Github: `https://github.com/29Schiller/Airline-Passenger-Satisfaction---Big-Data`
- Multilayer Perceptron (MLP): `https://spotintelligence.com/2024/02/20/multilayer-perceptron-mlp/`
- Random Forest Algorithm: `https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/`
- Gradient Boosting Algorithm: `https://www.geeksforgeeks.org/ml-gradient-boosting/`
- Logistics Regression Algorithm: `https://www.geeksforgeeks.org/understanding-logistic-regression/`