Sri Lanka Institute of Information Technology

Assignment 01

Airline Passenger Satisfaction Classification Models

Machine Learning – IT4060

B.Sc. (Hons) in Information Technology

Group Members

Name	IT Number	Email	ML algorithm Used
Arandara. S. D	IT21164330	It21164330@my.sliit.lk	Logistic Regression
Thuduwage I.M.H.G	IT21169380	it21169380@my.sliit.lk	SVM
Karunarathne R.Y.D.	IT21169144	it21169144@my.sliit.lk	Decision Tree
Kodithuwakku C.K.	IT21156960	it21156960@my.sliit.lk	Random Forest

Table of Contents

1.Introduction

1.1Problem Statement

In this project, we aim to develop machine learning models to predict airline passenger satisfaction based on various service and demographic-related features. The goal is to classify whether a passenger was satisfied or not with their flight experience.

1.2 Importance

Accurately predicting customer satisfaction helps airlines improve service quality, allocate resources efficiently, and enhance overall passenger experience. It's also a valuable tool for customer relationship management and loyalty programs.

1.3 Dataset Overview

We used the publicly available **Airline Passenger Satisfaction** dataset from Kaggle. The dataset includes features such as flight distance, seat comfort, inflight service ratings, and delays. Each record represents a passenger with an associated satisfaction label: **"satisfied"** or **"neutral or dissatisfied"**.

Dataset Source:
https://www.kaggle.com/datasets/teejmahal20/airline-passenger-satisfaction

- 6,000 +rows
- 25 features (numerical and categorical)

1.4 Project Objective

The objective of this project is to:

- Preprocess and clean the data effectively
- Train and evaluate **four different supervised learning models**
    - Random Forest
    - Decision Tree

    - Support Vector Machine (SVM)
    - Logistic Regression
- Compare the models using metrics like **accuracy**, **F1-score**, and **AUC**
- Provide insights on which algorithm performs best for this classification task

2. Dataset

The dataset used for this project is the **Airline Passenger Satisfaction** dataset sourced from Kaggle. It contains structured information collected through a survey of airline passengers, with the goal of predicting their satisfaction levels.

Dataset Dimensions are ,

- **Training Set Shape**: 35,827 rows × 25 columns
- **Testing Set Shape**: 25,976 rows × 25 columns

Each row represents a single passenger, and each column describes a feature related to the passenger's travel experience, demographics, or satisfaction rating.

Feature	**Description**
Unnamed: 0	Row index (not useful for modeling)
id	Unique identifier for each passenger (to be dropped)
Gender	Passenger gender (Male, Female)
Customer Type	Type of customer (Loyal Customer, Disloyal Customer)
Age	Age of the passenger
Type of Travel	Purpose of the trip (Personal Travel, Business travel)
Class	Class of travel (Eco, Eco Plus, Business)

Feature	Description
Flight Distance	Distance of the flight in miles
Inflight wifi service	Rating (0 = N/A, 1–5)
Departure/Arrival time convenient	Convenience rating (1–5)
Ease of Online booking	Rating (1–5)
Gate location	Rating (1–5)
Food and drink	Rating (1–5)
Online boarding	Rating (1–5)
Seat comfort	Rating (1–5)
Inflight entertainment	Rating (1–5)
On-board service	Rating (1–5)
Leg room service	Rating (1–5)
Baggage handling	Rating (1–5)
Checkin service	Rating (1–5)
Inflight service	Rating (1–5)
Cleanliness	Rating (1–5)
Departure Delay in Minutes	Delay in departure time (numeric)
Arrival Delay in Minutes	Delay in arrival time (numeric)
satisfaction	**Target variable** — 0 = "neutral or dissatisfied", 1 = "satisfied"

Missing Values

The dataset is relatively clean but contains some missing entries:

- **23 features** have exactly **1 missing value**
- Arrival Delay in Minutes has **106 missing values**
- Total rows with missing values: 106+

Since machine learning models cannot process missing data directly, appropriate preprocessing steps were applied (dropped or imputed as needed during model preparation).

Class Distribution (Target Variable)

The target variable satisfaction is binary:

- 0 – Neutral or Dissatisfied
- 1 – Satisfied

The distribution is moderately balanced, making it suitable for standard supervised learning algorithms. Visual plots (pie charts and count plots) are provided below for detailed understanding.

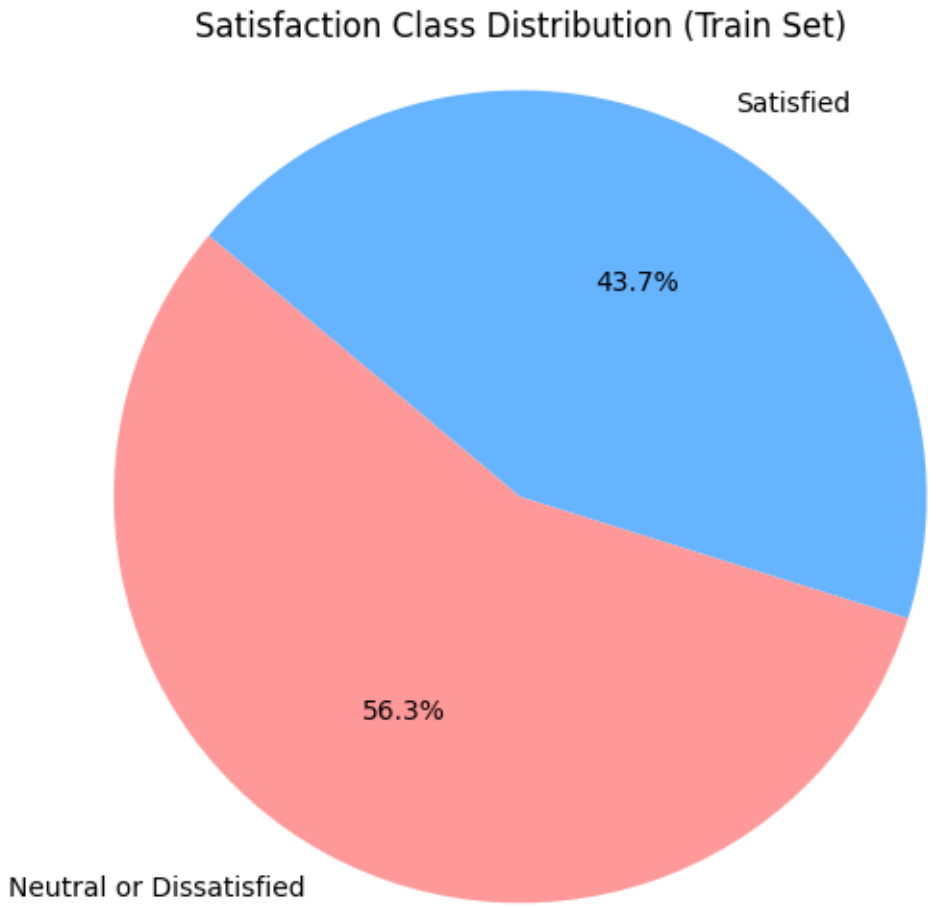
2.1 **Explore the Dataset**

```
Train data shape: (35827, 25)
Test data shape: (25976, 25)
```

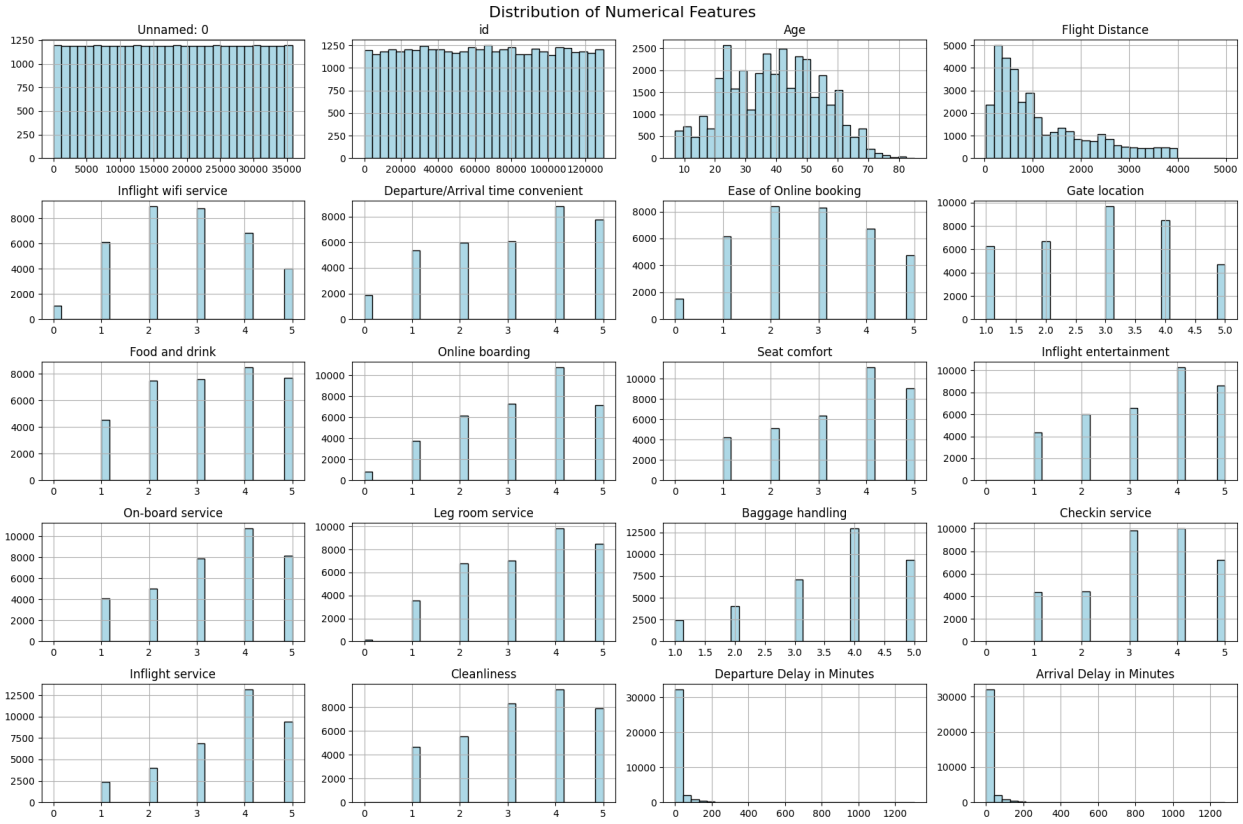
	Unnamed: 0	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	...	Inflight entertainment	On-board service	Leg room service	Baggage handling	Checkin service	Inflight service	Cleanliness	Departure Delay in Minutes	Arri De Minu
0	0	70172	Male	Loyal Customer	13.0	Personal Travel	Eco Plus	460.0	3.0	4.0	...	5.0	4.0	3.0	4.0	4.0	5.0	5.0	25.0	
1	1	5047	Male	disloyal Customer	25.0	Business travel	Business	235.0	3.0	2.0	...	1.0	1.0	5.0	3.0	1.0	4.0	1.0	1.0	
2	2	110028	Female	Loyal Customer	26.0	Business travel	Business	1142.0	2.0	2.0	...	5.0	4.0	3.0	4.0	4.0	4.0	5.0	0.0	
3	3	24026	Female	Loyal Customer	25.0	Business travel	Business	562.0	2.0	5.0	...	2.0	2.0	5.0	3.0	1.0	4.0	2.0	11.0	
4	4	119299	Male	Loyal Customer	61.0	Business travel	Business	214.0	3.0	3.0	...	3.0	3.0	4.0	4.0	3.0	3.0	3.0	0.0	

5 rows × 25 columns

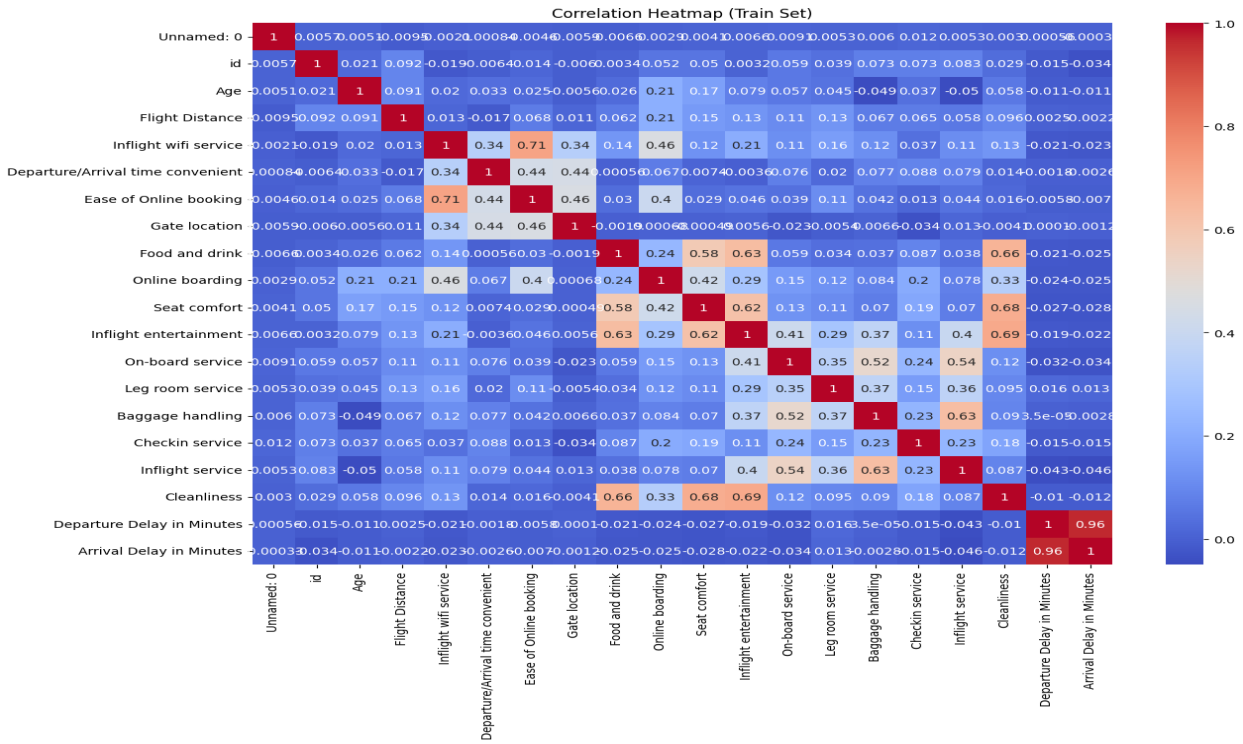
Img 01 : Data set head



Satisfaction Class Distribution (Train Set)



Distribution of Numerical Features



Correlation Heatmap (Train Set)

3. Methodology

3.1 Data Preprocessing

3.1.1 Data preprocessing – IT21164330

- Handling missing values

```
Missing Values (Test):
Unnamed: 0                            0
id                                    0
Gender                                0
Customer Type                         0
Age                                   0
Type of Travel                        0
Class                                 0
Flight Distance                       0
Inflight wifi service                 0
Departure/Arrival time convenient     0
Ease of Online booking                0
Gate location                         0
Food and drink                        0
Online boarding                       0
Seat comfort                          0
Inflight entertainment                0
On-board service                      0
Leg room service                      0
Baggage handling                      0
Checkin service                       0
Inflight service                      0
Cleanliness                           0
Departure Delay in Minutes            0
Arrival Delay in Minutes             83
satisfaction                          0
dtype: int64
```

Source code :

```
train_df.dropna(inplace=True)

test_df.dropna(inplace=True)


print("✅ Missing values removed.")

print("New train shape:", train_df.shape)

print("New test shape:", test_df.shape)
```

```
train_df.dropna(inplace=True)
test_df.dropna(inplace=True)

print("✅ Missing values removed.")
print("New train shape:", train_df.shape)
print("New test shape:", test_df.shape)
```

```
✅ Missing values removed.
New train shape: (44628, 25)
New test shape: (25893, 25)
```

- Label Encoding – IT21164330

```
[13] # Encode target column
     train_df['satisfaction'] = train_df['satisfaction'].replace({
         'satisfied': 1,
         'neutral or dissatisfied': 0
     })

     test_df['satisfaction'] = test_df['satisfaction'].replace({
         'satisfied': 1,
         'neutral or dissatisfied': 0
     })
```

```
[14] from sklearn.preprocessing import LabelEncoder
     import joblib
     import os

     # Create folder for encoders
     os.makedirs("logreg_encoders", exist_ok=True)

     # Encode each categorical column
     categorical_cols = train_df.select_dtypes(include='object').columns
     label_encoders = {}

     for col in categorical_cols:
         le = LabelEncoder()
         train_df[col] = le.fit_transform(train_df[col])
         test_df[col] = le.transform(test_df[col])
         joblib.dump(le, f"logreg_encoders/{col}_encoder.pkl")
         label_encoders[col] = le
```

Source Code :

- Feature Scaling – IT21164330

```
[15] from sklearn.preprocessing import StandardScaler

     # Separate features and labels
     X_train_full = train_df.drop('satisfaction', axis=1)
     y_train_full = train_df['satisfaction']

     X_test_full = test_df.drop('satisfaction', axis=1)
     y_test_full = test_df['satisfaction']

     # Apply StandardScaler
     scaler = StandardScaler()
     X_train_scaled = scaler.fit_transform(X_train_full)
     X_test_scaled = scaler.transform(X_test_full)
```

- Train-validation-test splitting

```
[16] from sklearn.model_selection import train_test_split

     # Stratified split of training data
     X_train, X_val, y_train, y_val = train_test_split(
         X_train_scaled,
         y_train_full,
         test_size=0.2,
         random_state=42,
         stratify=y_train_full
     )

     print("Training set shape:", X_train.shape)
     print("Validation set shape:", X_val.shape)

     Training set shape: (35702, 22)
     Validation set shape: (8926, 22)
```

3.1.2 Data preprocessing – IT21169380

- Handling missing values

```
Missing Values (Train):
Unnamed: 0                             0
id                                     0
Gender                                 0
Customer Type                          0
Age                                    1
Type of Travel                         1
Class                                  1
Flight Distance                        1
Inflight wifi service                  1
Departure/Arrival time convenient      1
Ease of Online booking                 1
Gate location                          1
Food and drink                         1
Online boarding                        1
Seat comfort                           1
Inflight entertainment                 1
On-board service                       1
Leg room service                       1
Baggage handling                       1
Checkin service                        1
Inflight service                       1
Cleanliness                            1
Departure Delay in Minutes             1
Arrival Delay in Minutes             106
satisfaction                           1
dtype: int64
```

- Drop Columns

```
[ ]  # Drop ID columns
     columns_to_drop = ['Unnamed: 0', 'id']
     train_df.drop(columns=columns_to_drop, axis=1, inplace=True)
     test_df.drop(columns=columns_to_drop, axis=1, inplace=True)

     print("Remaining columns after dropping ID columns:")
     train_df.columns.tolist()
```

```
Remaining columns after dropping ID columns:
['Gender',
 'Customer Type',
 'Age',
 'Type of Travel',
 'Class',
 'Flight Distance',
 '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',
 'Departure Delay in Minutes',
 'Arrival Delay in Minutes',
 'satisfaction']
```

- Label Encoding

```
[ ]  # Encode target variable ('satisfaction') using replace
     train_df['satisfaction'] = train_df['satisfaction'].replace({
         'satisfied': 1,
         'neutral or dissatisfied': 0
     })

     test_df['satisfaction'] = test_df['satisfaction'].replace({
         'satisfied': 1,
         'neutral or dissatisfied': 0
     })

     print("Target encoding done.")
```

```python
from sklearn.preprocessing import LabelEncoder
import joblib
import os

# Create directory to store encoders
os.makedirs("svm_encoders", exist_ok=True)

# Categorical columns
categorical_cols = train_df.select_dtypes(include='object').columns

# Encode each categorical column separately
label_encoders = {}

for col in categorical_cols:
    le = LabelEncoder()
    train_df[col] = le.fit_transform(train_df[col])
    test_df[col] = le.transform(test_df[col])

    # Save encoder for future use
    joblib.dump(le, f"svm_encoders/{col}_encoder.pkl")
    label_encoders[col] = le

print("Categorical features encoded and LabelEncoders saved.")
```

```
Categorical features encoded and LabelEncoders saved.
```

- Split Data set

```python
[ ] from sklearn.model_selection import train_test_split

    # Split the scaled training data
    X_train, X_val, y_train, y_val = train_test_split(
        X_train_scaled,
        y_train_full,
        test_size=0.2,
        random_state=42,
        # stratify=y_train_full
    )

    print("Training set shape:", X_train.shape)
    print("Validation set shape:", X_val.shape)
```

```
Training set shape: (28661, 22)
Validation set shape: (7166, 22)
```

3.1.3 Data preprocessing – IT21156960

- Handling missing values

```
[ ]  print(train_df.isnull().sum())
```

```
Unnamed: 0                            0
id                                    0
Gender                                0
Customer Type                         0
Age                                   0
Type of Travel                        0
Class                                 0
Flight Distance                       0
Inflight wifi service                 0
Departure/Arrival time convenient     0
Ease of Online booking                0
Gate location                         0
Food and drink                        0
Online boarding                       0
Seat comfort                          0
Inflight entertainment                0
On-board service                      0
Leg room service                      0
Baggage handling                      0
Checkin service                       0
Inflight service                      0
Cleanliness                           0
Departure Delay in Minutes            0
Arrival Delay in Minutes            310
satisfaction                          0
dtype: int64
```

```
[ ]  print("Missing values before handling:\n")
     print(train_df.isnull().sum())

     numeric_columns = train_df.select_dtypes(include=['float64', 'int64']).columns
     train_df[numeric_columns] = train_df[numeric_columns].fillna(train_df[numeric_columns].median())

     test_df[numeric_columns] = test_df[numeric_columns].fillna(test_df[numeric_columns].median())

     print("\nMissing values after handling:\n")
     print(train_df.isnull().sum())
```

```
Missing values before handling:
```

- Label Encoding

```
[ ]  label_encoders = {}

     categorical_cols = ['Gender', 'Customer Type', 'Type of Travel', 'Class']

     for col in categorical_cols:
         le = LabelEncoder()
         train_df[col] = le.fit_transform(train_df[col])
         test_df[col] = le.transform(test_df[col])
         label_encoders[col] = le
```

```
[ ]  train_df['satisfaction'] = train_df['satisfaction'].map({
         'satisfied': 1,
         'neutral or dissatisfied': 0
     })
     test_df['satisfaction'] = test_df['satisfaction'].map({
         'satisfied': 1,
         'neutral or dissatisfied': 0
     })

     print(train_df['satisfaction'].value_counts(normalize=True) * 100)
     print(test_df['satisfaction'].value_counts(normalize=True) * 100)
```

```
satisfaction
0    56.666731
1    43.333269
Name: proportion, dtype: float64
satisfaction
0    56.107828
1    43.892172
Name: proportion, dtype: float64
```

- Split Data set

```
[ ]  X = train_df.drop('satisfaction', axis=1)
     y = train_df['satisfaction']

     X_train, X_val, y_train, y_val = train_test_split(
         X, y, test_size=0.2, random_state=42, stratify=y
     )


     print("Training set shape:", X_train.shape)
     print("Validation set shape:", X_val.shape)
```

```
Training set shape: (83123, 22)
Validation set shape: (20781, 22)
```

3.1.4 Data preprocessing – IT21169144

- Handling missing values and drop columns

```
[ ]  print(train_df.isnull().sum())

Unnamed: 0                            0
id                                    0
Gender                                0
Customer Type                         0
Age                                   0
Type of Travel                        0
Class                                 0
Flight Distance                       0
Inflight wifi service                 0
Departure/Arrival time convenient     0
Ease of Online booking                0
Gate location                         0
Food and drink                        0
Online boarding                       0
Seat comfort                          0
Inflight entertainment                0
On-board service                      0
Leg room service                      0
Baggage handling                      0
Checkin service                       0
Inflight service                      0
Cleanliness                           0
Departure Delay in Minutes            0
Arrival Delay in Minutes            310
satisfaction                          0
dtype: int64
```

```
[ ]  columns_to_drop = ['Unnamed: 0', 'id']
     train_df.drop(columns=columns_to_drop, axis=1, inplace=True)
     test_df.drop(columns=columns_to_drop, axis=1, inplace=True)
```

```
[ ]  numeric_cols = train_df.select_dtypes(include=['float64', 'int64']).columns

     for col in numeric_cols:
         train_df[col].fillna(train_df[col].mean(), inplace=True)
         test_df[col].fillna(test_df[col].mean(), inplace=True)
```

- Label Encoding

```
[ ]  # Encode satisfaction using .replace() instead of map/lambda
     train_df['satisfaction'] = train_df['satisfaction'].replace({
         'satisfied': 1,
         'neutral or dissatisfied': 0
     })

     test_df['satisfaction'] = test_df['satisfaction'].replace({
         'satisfied': 1,
         'neutral or dissatisfied': 0
     })
```

```
[ ]  # Step 4: Encode categorical features using LabelEncoder
     label_encoders = {}
     categorical_cols = train_df.select_dtypes(include='object').columns


     for col in categorical_cols:
         le = LabelEncoder()
         train_df[col] = le.fit_transform(train_df[col])
         test_df[col] = le.transform(test_df[col])

         label_encoders[col] = le

     print("✅ Categorical features encoded and encoders saved.")
```

```
✅ Categorical features encoded and encoders saved.
```

- Split Data set

```
[ ]  from sklearn.model_selection import train_test_split

     # Separate features and target
     X = train_df.drop('satisfaction', axis=1)
     y = train_df['satisfaction']

     # Split into training and validation sets
     X_train, X_val, y_train, y_val = train_test_split(
         X, y, test_size=0.2, random_state=42, stratify=y
     )

     print("Training set shape:", X_train.shape)
     print("Validation set shape:", X_val.shape)
```

```
Training set shape: (83123, 22)
Validation set shape: (20781, 22)
```

3.2 Machine Learning Algorithms Used

In this project, we applied four supervised learning algorithms to predict airline passenger satisfaction. Each model was carefully selected to provide a balance between accuracy, interpretability, and computational efficiency.

- **Logistic Regression – IT21164330**

Logistic Regression is a statistical model that uses a logistic function to model binary dependent variables. It's a fundamental baseline model for classification tasks.

- **Strengths**:
  - Very interpretable (coefficients indicate feature importance).
  - Fast to train and predict.
  - Good performance when classes are linearly separable.
- **Limitations**:
  - Assumes a linear relationship between features and log-odds.
  - Struggles with complex non-linear patterns without feature engineering.

In this project: Logistic Regression served as a simple, interpretable baseline and showed competitive accuracy after scaling and tuning.

- **Support Vector Machine (SVM)**

SVM is a powerful classifier that finds the optimal hyperplane separating data into classes. It is effective even in high-dimensional spaces.

- **Strengths**:
  - Works well in cases where classes are not linearly separable (using kernel trick).
  - Effective in high-dimensional feature spaces.
- **Limitations**:
  - Requires careful feature scaling.

- Computationally intensive for large datasets.
- Choosing the right kernel and hyperparameters is critical.

In this project: SVM achieved strong classification results after proper scaling and hyperparameter tuning, particularly benefiting from using an RBF kernel.

- **Random Forest Classifier -IT21156960**

Random Forest is an **ensemble learning method** that builds multiple decision trees and combines their outputs for a more accurate and robust prediction.

- **Strengths**:
  - Handles both categorical and numerical features well.
  - Reduces overfitting compared to a single decision tree.
  - Capable of modeling complex nonlinear relationships.
- **Limitations**:
  - Can be computationally heavy for very large datasets.
  - Less interpretable than a single tree.

In this project: Random Forest provided high accuracy and good generalization, making it one of the strongest performers.

- **Decision Tree Classifier - IT21169144**

A Decision Tree creates a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

- **Strengths**:
  - Highly interpretable (can visualize the tree structure).
  - Fast to train and predict.
  - Can capture nonlinear decision boundaries.
- **Limitations**:
  - Prone to overfitting if not properly controlled (depth, pruning).

- Can be unstable — small variations in data might change the structure.

In this project: The Decision Tree offered good interpretability but slightly lower performance compared to ensemble methods.

3.2.1 Model Architecture – IT21164330

```
[ ] from sklearn.model_selection import GridSearchCV

    # Define parameter grid
    param_grid = {
        'C': [0.01, 0.1, 1, 10],
        'penalty': ['l1', 'l2'],
        'solver': ['liblinear'],  # liblinear supports both L1 and L2
    }

    # Initialize base model
    log_base = LogisticRegression(max_iter=1000, random_state=42)

    # Grid search with cross-validation
    grid_search = GridSearchCV(log_base, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
    grid_search.fit(X_train, y_train)

    # Best model and parameters
    best_log_model = grid_search.best_estimator_
    print("✅ Best Parameters Found:", grid_search.best_params_)
```

```
✅ Best Parameters Found: {'C': 1, 'penalty': 'l2', 'solver': 'liblinear'}
```

```
[ ] # Use the best hyperparameters
    best_log_model = LogisticRegression(
        C=1,
        penalty='l2',
        solver='liblinear',
        max_iter=1000,
        random_state=42
    )

    # Train the model
    best_log_model.fit(X_train, y_train)

    print("✅ Best Logistic Regression model trained.")
```

```
✅ Best Logistic Regression model trained.
```

3.2.2 Model Architecture -IT21169380

```
from sklearn.svm import SVC

# Initialize SVM model
svm_model = SVC(
    kernel='rbf',
    C=1.0,
    probability=True,
    random_state=42
)

# Train the model
svm_model.fit(X_train, y_train)

print("✅ SVM model training complete.")
```

```
✅ SVM model training complete.
```

3.2.3 Model Architecture - IT21156960

```
[ ]  rf_model = RandomForestClassifier(
        n_estimators=100,
        max_depth=None,
        random_state=42,
        class_weight='balanced'
    )

    rf_model.fit(X_train, y_train)

    y_pred = rf_model.predict(X_val)

    accuracy = accuracy_score(y_val, y_pred)
    print("Validation Accuracy:", round(accuracy * 100, 2), "%")
```

```
Validation Accuracy: 96.18 %
```

3.2.4 Model Architecture – IT21169144

```
[ ]  from sklearn.tree import DecisionTreeClassifier
     from sklearn.metrics import accuracy_score

     # Initialize the Decision Tree model
     dt_model = DecisionTreeClassifier(
         random_state=42,
         class_weight='balanced'  # helpful for slight imbalance
     )

     # Train the model
     dt_model.fit(X_train, y_train)

     # Predict on validation set
     y_val_pred = dt_model.predict(X_val)

     # Evaluate accuracy
     val_accuracy = accuracy_score(y_val, y_val_pred)
     print(f"Decision Tree - Validation Accuracy: {val_accuracy * 100:.2f}%")

     Decision Tree - Validation Accuracy: 94.74%
```

4. Evaluation & Results

All of the group four member has used following

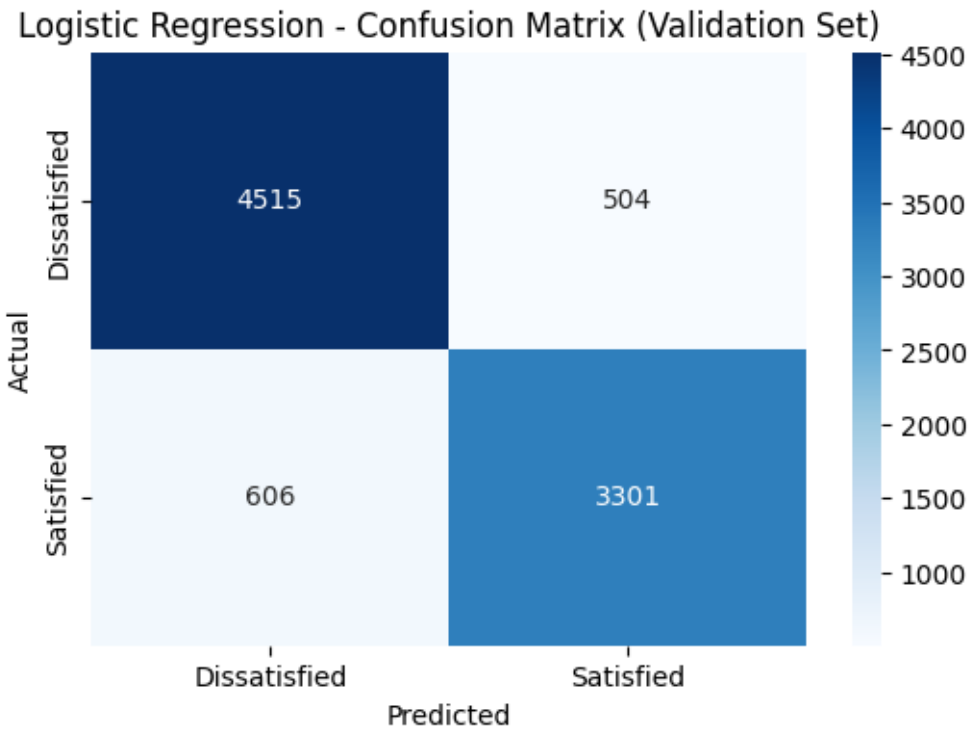
Evaluation Metrics

- Accuracy
- Confusion Matrix
- Precision, Recall, F1-score
- ROC Curve & AUC

4.1 Evaluation Metrices – IT21164330

```
[ ]  from sklearn.metrics import accuracy_score

     # Predict on training data
     y_train_pred = best_log_model.predict(X_train)

     # Accuracy
     train_acc = accuracy_score(y_train, y_train_pred)
     print(f"🔍 Training Accuracy (Tuned Model): {train_acc * 100:.2f}%")

     🔍 Training Accuracy (Tuned Model): 87.30%
```

Training Accuracy

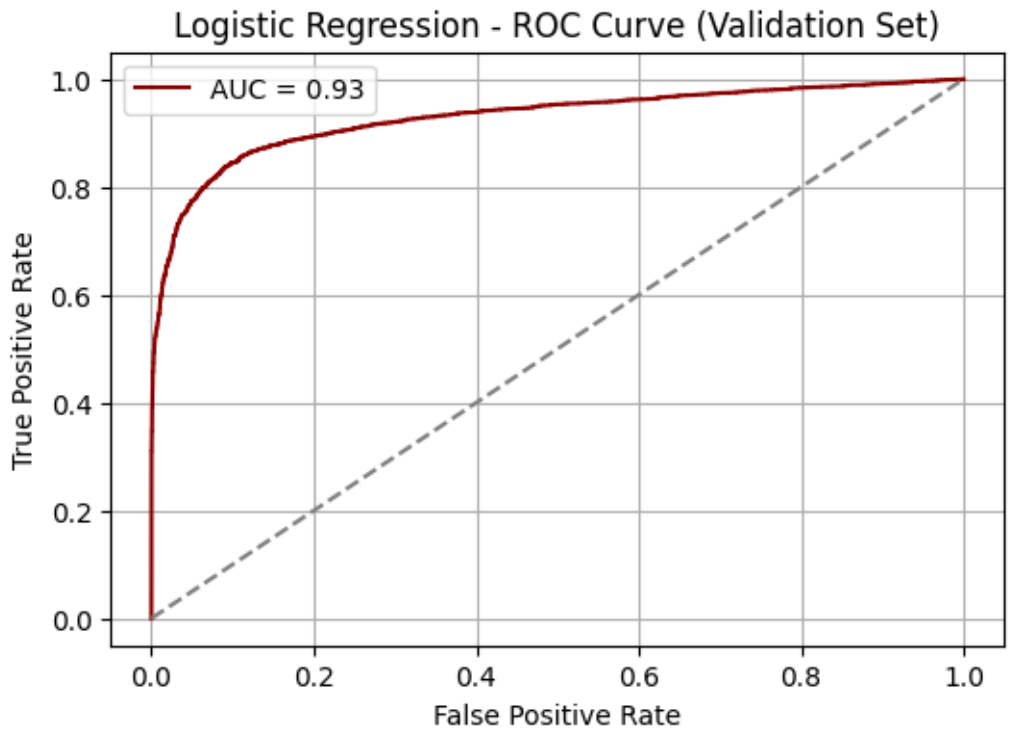


Confusion Matrix

```
[ ] from sklearn.metrics import classification_report

    print("Logistic Regression - Classification Report (Validation Set):\n")
    print(classification_report(y_val, y_val_pred, target_names=['Dissatisfied', 'Satisfied']))
```

```
Logistic Regression - Classification Report (Validation Set):

              precision    recall  f1-score   support

Dissatisfied       0.88      0.90      0.89      5019
   Satisfied       0.87      0.84      0.86      3907

    accuracy                           0.88      8926
   macro avg       0.87      0.87      0.87      8926
weighted avg       0.88      0.88      0.88      8926
```

Classification Report



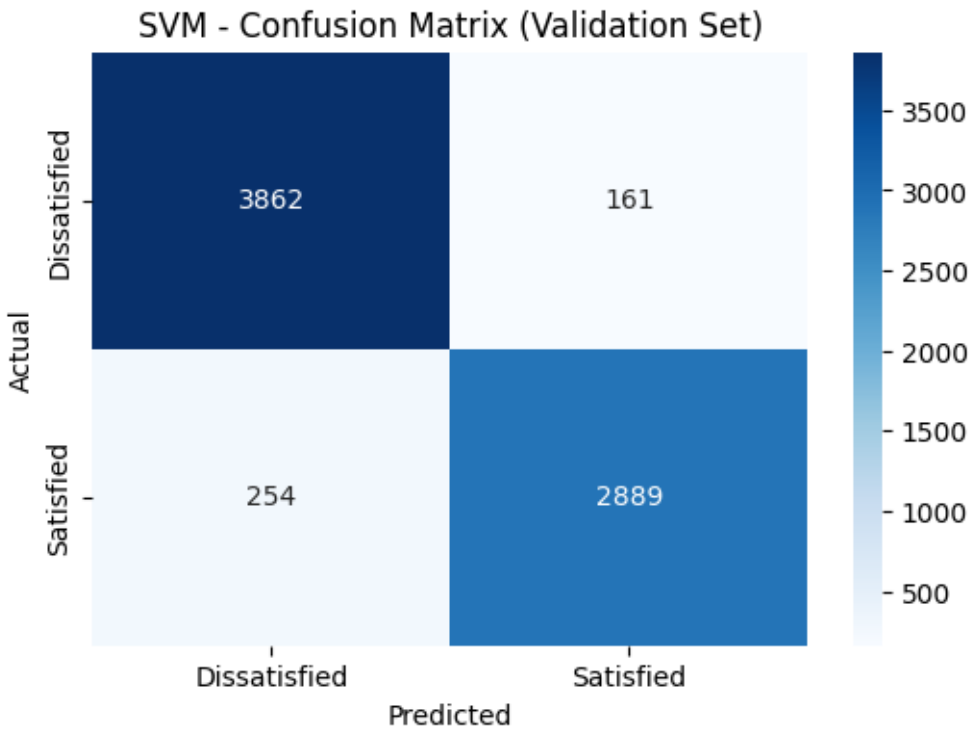
ROC curves

4.2 Evaluation Metrices – IT21169380

```
from sklearn.metrics import accuracy_score

# Predict on training set itself
y_train_pred = svm_model.predict(X_train)

# Training Accuracy
train_accuracy = accuracy_score(y_train, y_train_pred)
print(f"✅ SVM - Training Accuracy: {train_accuracy * 100:.2f}%")
```

```
✅ SVM - Training Accuracy: 95.59%
```

Training Accuracy

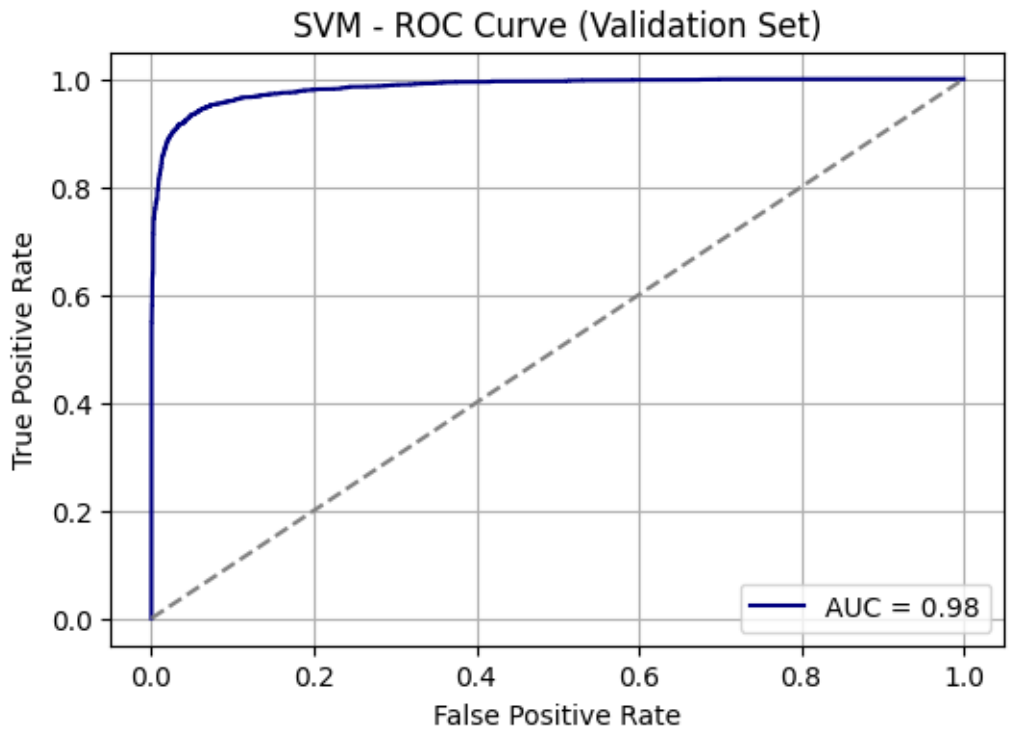


Confusion Matrix

```
[ ]  from sklearn.metrics import classification_report

     # Print classification report
     print("SVM - Classification Report (Validation Set):\n")
     print(classification_report(y_val, y_val_pred, target_names=['Dissatisfied', 'Satisfied']))
```

```
SVM - Classification Report (Validation Set):

                precision    recall  f1-score   support

  Dissatisfied       0.94      0.96      0.95      4023
     Satisfied       0.95      0.92      0.93      3143

      accuracy                           0.94      7166
     macro avg       0.94      0.94      0.94      7166
  weighted avg       0.94      0.94      0.94      7166
```

Classification Report



ROC curve

4.3 Evaluation Metrices – IT21156960

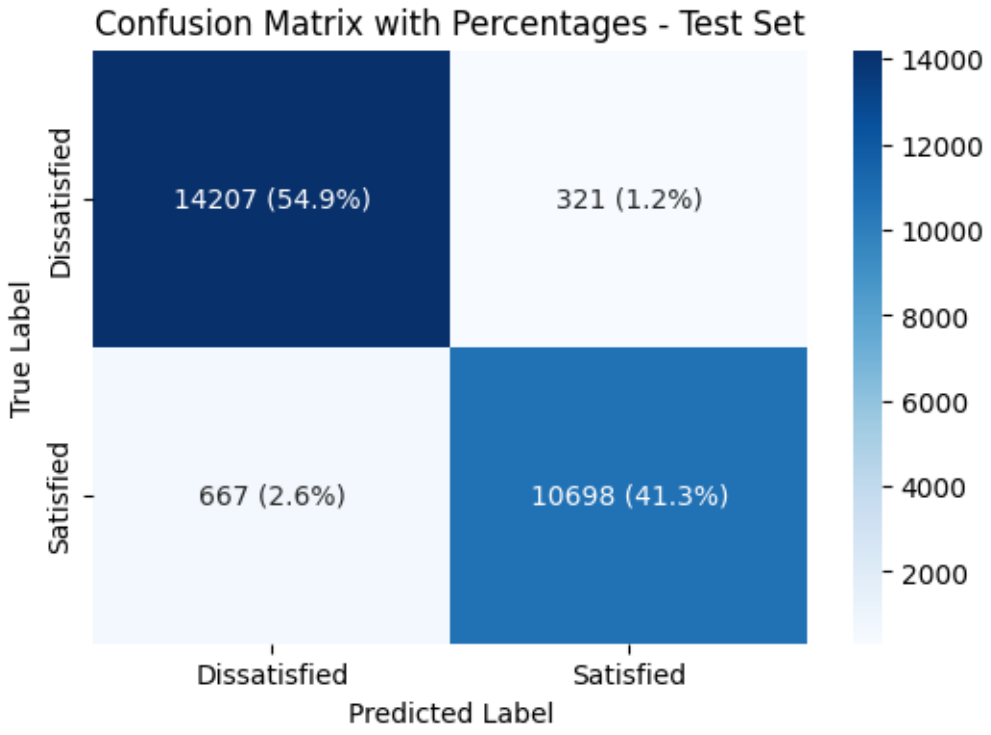
```
[ ]  rf_model = RandomForestClassifier(
         n_estimators=100,
         max_depth=None,
         random_state=42,
         class_weight='balanced'
     )

     rf_model.fit(X_train, y_train)

     y_pred = rf_model.predict(X_val)

     accuracy = accuracy_score(y_val, y_pred)
     print("Validation Accuracy:", round(accuracy * 100, 2), "%")

     Validation Accuracy: 96.18 %
```

Validation Accuracy

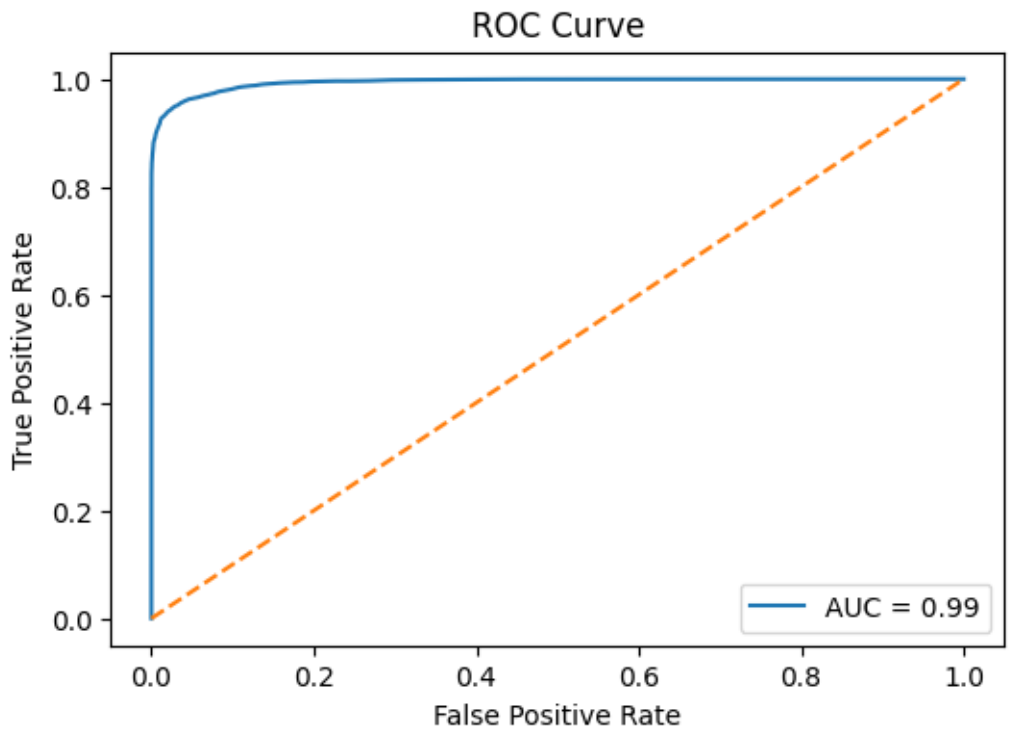
```
[ ]  print(classification_report(y_val, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97     11776
           1       0.97      0.94      0.96      9005

    accuracy                           0.96     20781
   macro avg       0.96      0.96      0.96     20781
weighted avg       0.96      0.96      0.96     20781
```

Classification Report



Confusion Matrix



ROC curve

4.4 Evaluation Metrices – IT24469144

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Initialize the Decision Tree model
dt_model = DecisionTreeClassifier(
    random_state=42,
    class_weight='balanced'  # helpful for slight imbalance
)

# Train the model
dt_model.fit(X_train, y_train)

# Predict on validation set
y_val_pred = dt_model.predict(X_val)

# Evaluate accuracy
val_accuracy = accuracy_score(y_val, y_val_pred)
print(f"Decision Tree - Validation Accuracy: {val_accuracy * 100:.2f}%")
```

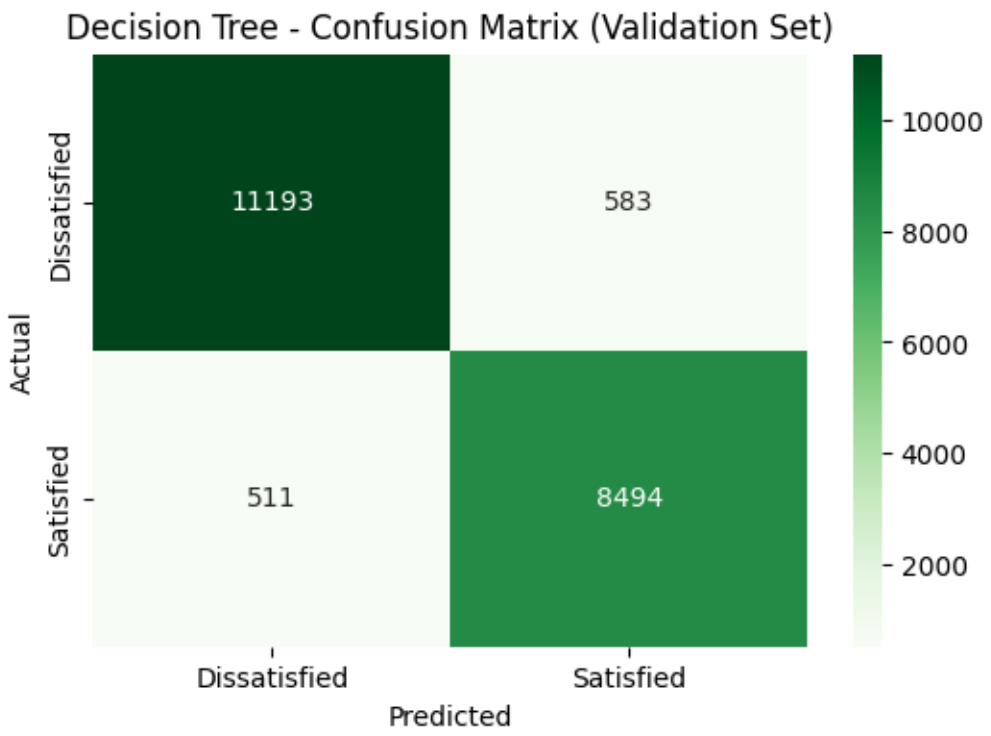
```
Decision Tree - Validation Accuracy: 94.74%
```

Validation Accuracy

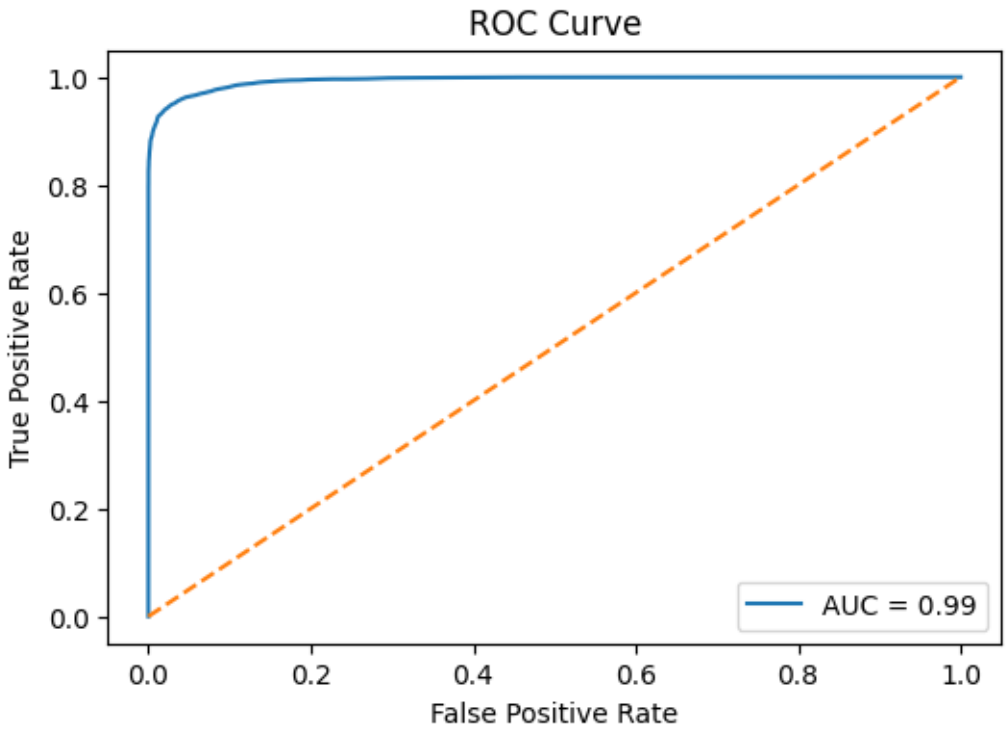
```
[ ]  from sklearn.metrics import classification_report

     # Textual evaluation metrics
     print("Decision Tree - Classification Report (Validation Set):\n")
     print(classification_report(y_val, y_val_pred, target_names=['Dissatisfied', 'Satisfied']))
```

```
Decision Tree - Classification Report (Validation Set):

              precision    recall  f1-score   support

Dissatisfied       0.96      0.95      0.95     11776
   Satisfied       0.94      0.94      0.94      9005

    accuracy                           0.95     20781
   macro avg       0.95      0.95      0.95     20781
weighted avg       0.95      0.95      0.95     20781
```

Classification Report



Confusion Matrix


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.99

ROC curve

5. Discussion and conclusions

This project explored and compared four popular supervised learning algorithms — Random Forest, Decision Tree, Support Vector Machine (SVM), and Logistic Regression — to predict airline passenger satisfaction using a real-world dataset.

All models were trained using the same data preprocessing pipeline, which included label encoding, missing value handling, and feature scaling (where required). Each model was evaluated using multiple metrics including **accuracy**, **precision**, **recall**, **F1-score**, and **ROC-AUC**, both on a validation set and a final test set.

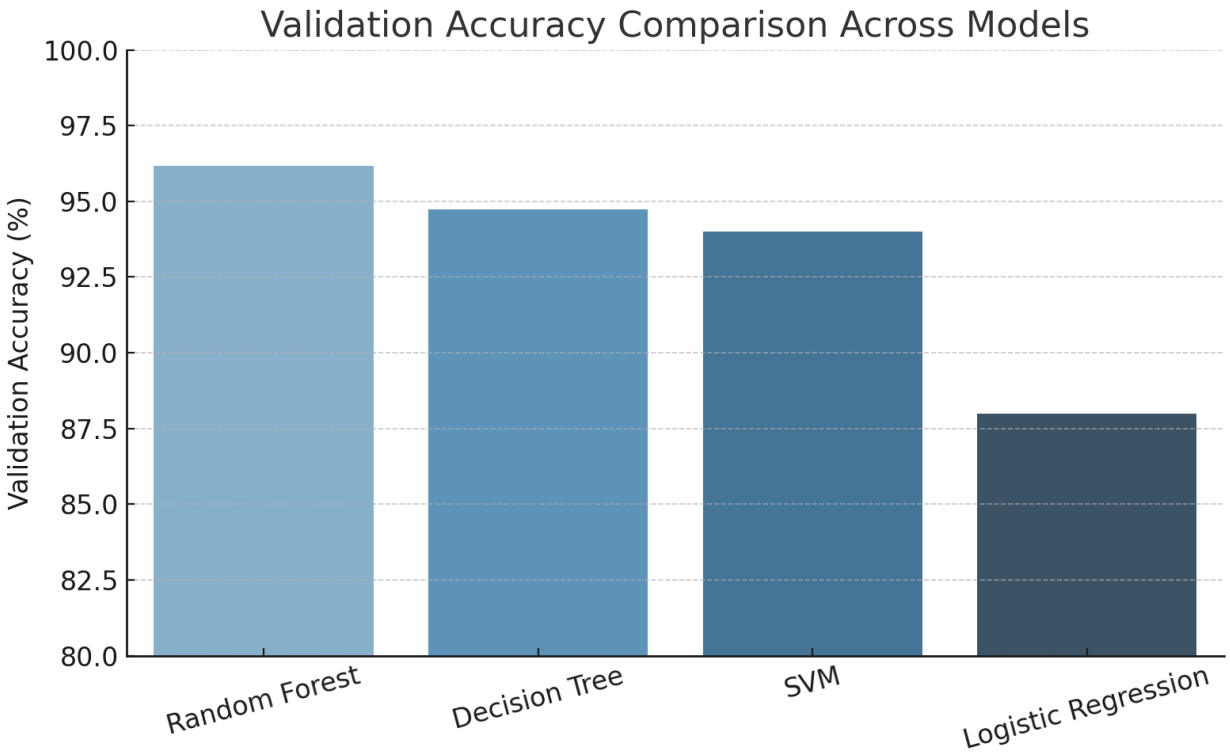
- **Model Comparison Summary:**
  - **Random Forest** consistently delivered the highest accuracy (96.18%) on the validation set and performed equally well on the test set. It effectively balanced both classes and showed robustness against noise and outliers.
  - **Decision Tree** followed closely with a validation accuracy of 94.74%. It showed strong performance but had a slightly higher misclassification rate for satisfied passengers.
  - **SVM** achieved high precision and recall with a validation accuracy of around 94%. It required proper scaling but handled class separation well due to its kernel-based structure.
  - **Logistic Regression**, being a linear model, performed as a strong baseline with 88% validation accuracy. It was the simplest and fastest model but struggled slightly with complex relationships in the data.

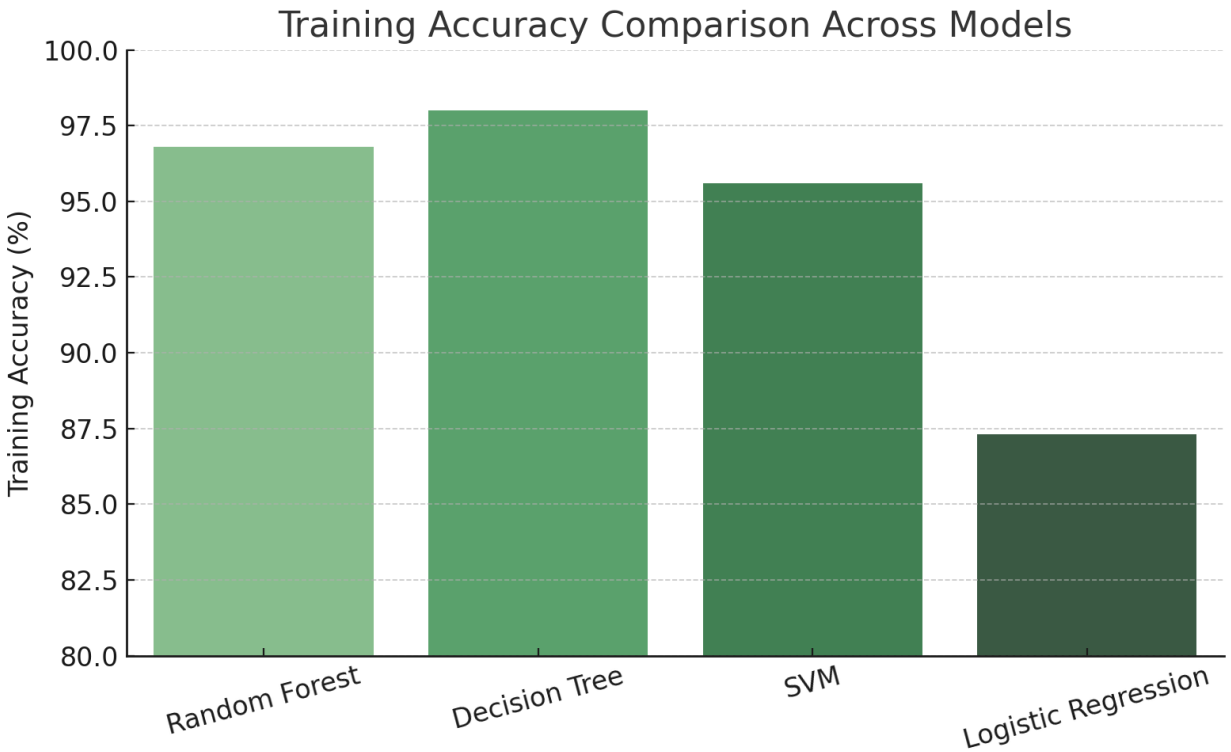
The **confusion matrices** supported these metrics, showing that Random Forest and SVM had the best class separation with fewer false negatives and false positives. Logistic Regression, while interpretable, had relatively more misclassifications, especially in predicting satisfied passengers

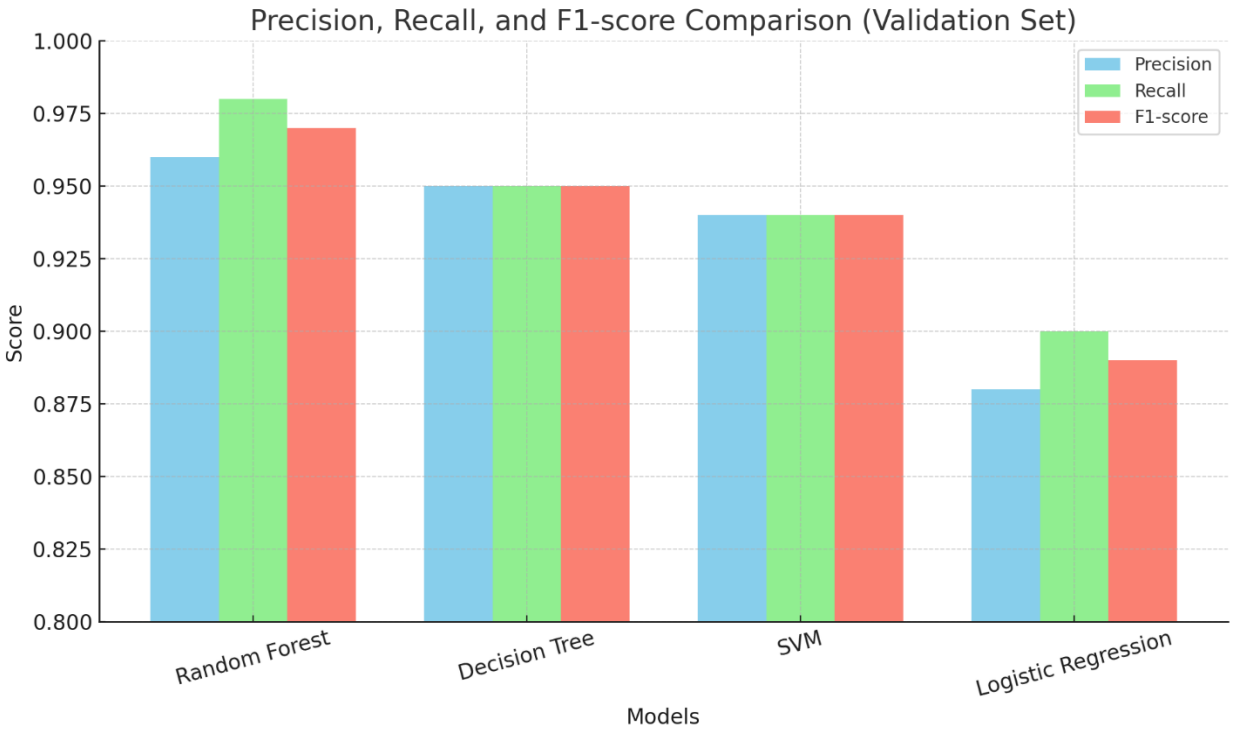
Conclusion

Based on the comprehensive evaluation of all four models, the **Random Forest Classifier** emerged as the best-performing model for predicting passenger satisfaction. Its ensemble nature, ability to handle nonlinear relationships, and resistance to overfitting allowed it to outperform other models across all key metrics.

This project demonstrates that machine learning can effectively model passenger satisfaction using real-world airline data. The outcome can be valuable for airlines in identifying service areas that influence customer satisfaction and improving overall passenger experience.


Validation Accuracy Comparison Across Models
Validation Accuracy (%)
100.0
97.5
95.0
92.5
90.0
87.5
85.0
82.5
80.0
Random Forest
Decision Tree
SVM
Logistic Regression


Training Accuracy Comparison Across Models
Training Accuracy (%)
100.0
97.5
95.0
92.5
90.0
87.5
85.0
82.5
80.0
Random Forest
Decision Tree
SVM
Logistic Regression


Precision, Recall, and F1-score Comparison (Validation Set)
Precision
Recall
F1-score
Score
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0.800
Random Forest
Decision Tree
SVM
Logistic Regression
Models

6. Source Code

IT21164330

```
import pandas as pd


train_df = pd.read_csv("/content/train.csv")

test_df = pd.read_csv("/content/test.csv")



print("Train set shape:", train_df.shape)

print("Test set shape:", test_df.shape)



train_df.head()


train_df.tail()


# Step 1.1: Check basic dataset info

print("Train Dataset Info:")

train_df.info()


print("\nMissing Values (Train):")

print(train_df.isnull().sum())
```

```
print("\nTest Dataset Info:")
test_df.info()

print("\nMissing Values (Test):")
print(test_df.isnull().sum())



import matplotlib.pyplot as plt

# Pie chart for class distribution
target_counts = train_df['satisfaction'].value_counts()
labels = ['Neutral or Dissatisfied', 'Satisfied']
sizes = target_counts.values

plt.figure(figsize=(6, 6))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140, colors=['#ffcc99', '#66b3ff'])
plt.title('Satisfaction Class Distribution (Train Set)')
plt.axis('equal')
plt.show()



# Step 1.4: Drop missing value rows in both train and test sets
train_df.dropna(inplace=True)
test_df.dropna(inplace=True)
```

```
print(" ✅ Missing values removed.")
print("New train shape:", train_df.shape)
print("New test shape:", test_df.shape)



import seaborn as sns
import matplotlib.pyplot as plt

# Select numeric columns
numeric_cols = train_df.select_dtypes(include=['float64', 'int64']).columns

# Plot in a 3-column grid
num_plots = len(numeric_cols)
rows = (num_plots // 3) + int(num_plots % 3 != 0)

fig, axes = plt.subplots(rows, 3, figsize=(18, 5 * rows))
axes = axes.flatten()

for i, col in enumerate(numeric_cols):
    sns.boxplot(data=train_df, x='satisfaction', y=col, ax=axes[i])
    axes[i].set_title(f"Boxplot of {col}")
    axes[i].set_xlabel("Satisfaction")
    axes[i].set_ylabel(col)
```

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

plt.tight_layout()
plt.show()



# Drop ID columns
columns_to_drop = ['Unnamed: 0', 'id']
train_df.drop(columns=columns_to_drop, axis=1, inplace=True)
test_df.drop(columns=columns_to_drop, axis=1, inplace=True)



# Encode target column
train_df['satisfaction'] = train_df['satisfaction'].replace({
    'satisfied': 1,
    'neutral or dissatisfied': 0
})

test_df['satisfaction'] = test_df['satisfaction'].replace({
    'satisfied': 1,
    'neutral or dissatisfied': 0
})
```

```
from sklearn.preprocessing import LabelEncoder
import joblib
import os

# Create folder for encoders
os.makedirs("logreg_encoders", exist_ok=True)

# Encode each categorical column
categorical_cols = train_df.select_dtypes(include='object').columns
label_encoders = {}

for col in categorical_cols:
    le = LabelEncoder()
    train_df[col] = le.fit_transform(train_df[col])
    test_df[col] = le.transform(test_df[col])
    joblib.dump(le, f"logreg_encoders/{col}_encoder.pkl")
    label_encoders[col] = le



from sklearn.preprocessing import StandardScaler

# Separate features and labels
X_train_full = train_df.drop('satisfaction', axis=1)
y_train_full = train_df['satisfaction']
```

```
X_test_full = test_df.drop('satisfaction', axis=1)
y_test_full = test_df['satisfaction']

# Apply StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_full)
X_test_scaled = scaler.transform(X_test_full)



from sklearn.model_selection import train_test_split

# Stratified split of training data
X_train, X_val, y_train, y_val = train_test_split(
    X_train_scaled,
    y_train_full,
    test_size=0.2,
    random_state=42,
    stratify=y_train_full
)

print("Training set shape:", X_train.shape)
print("Validation set shape:", X_val.shape)



from sklearn.linear_model import LogisticRegression
```

```
log_model = LogisticRegression(
    solver='liblinear',
    C=1.0,
    max_iter=1000,
    random_state=42
)

# Train the model
log_model.fit(X_train, y_train)

print(" ✅ Logistic Regression model trained.")



from sklearn.metrics import accuracy_score

# Predict on training set
y_train_pred = log_model.predict(X_train)

# Training accuracy
train_acc = accuracy_score(y_train, y_train_pred)
print(f" 🔍 Training Accuracy: {train_acc * 100:.2f}%")



from sklearn.model_selection import GridSearchCV
```

```
# Define parameter grid
param_grid = {
    'C': [0.01, 0.1, 1, 10],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear'],  # liblinear supports both L1 and L2
}

# Initialize base model
log_base = LogisticRegression(max_iter=1000, random_state=42)

# Grid search with cross-validation
grid_search = GridSearchCV(log_base, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)

# Best model and parameters
best_log_model = grid_search.best_estimator_
print(" ✅ Best Parameters Found:", grid_search.best_params_)


# Use the best hyperparameters
best_log_model = LogisticRegression(
    C=1,
    penalty='l2',
    solver='liblinear',
```

```
    max_iter=1000,
    random_state=42
)

# Train the model
best_log_model.fit(X_train, y_train)

print("✅ Best Logistic Regression model trained.")


from sklearn.metrics import accuracy_score

# Predict on training data
y_train_pred = best_log_model.predict(X_train)

# Accuracy
train_acc = accuracy_score(y_train, y_train_pred)
print(f"🔍 Training Accuracy (Tuned Model): {train_acc * 100:.2f}%")


#  Predict labels and probabilities for validation set
y_val_pred = best_log_model.predict(X_val)
y_val_proba = best_log_model.predict_proba(X_val)[:, 1]

from sklearn.metrics import confusion_matrix
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# Confusion matrix
cm_val = confusion_matrix(y_val, y_val_pred)

plt.figure(figsize=(6, 4))
sns.heatmap(cm_val, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Dissatisfied', 'Satisfied'],
            yticklabels=['Dissatisfied', 'Satisfied'])
plt.title("Logistic Regression - Confusion Matrix (Validation Set)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()


from sklearn.metrics import classification_report

print("Logistic Regression - Classification Report (Validation Set):\n")
print(classification_report(y_val, y_val_pred, target_names=['Dissatisfied', 'Satisfied']))


from sklearn.metrics import roc_curve, auc

# ROC curve
```

```
fpr, tpr, _ = roc_curve(y_val, y_val_proba)

roc_auc = auc(fpr, tpr)


plt.figure(figsize=(6, 4))

plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}", color='darkred')

plt.plot([0, 1], [0, 1], linestyle='--', color='gray')

plt.title("Logistic Regression - ROC Curve (Validation Set)")

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.legend()

plt.grid()

plt.show()
```

IT21169380

```
!ls


import pandas as pd


train_df = pd.read_csv("/content/train.csv")

test_df = pd.read_csv("/content/test.csv")


print("Train data shape:", train_df.shape)

print("Test data shape:", test_df.shape)
```

```
train_df.head()



# Basic structure and null value checking

print("Dataset Info:")

train_df.info()


print("\nMissing Values (Train):")

print(train_df.isnull().sum())


print("\nMissing Values (Test):")

print(test_df.isnull().sum())


# Quick statistical summary

train_df.describe()



import matplotlib.pyplot as plt


# Pie chart for target variable distribution

target_counts = train_df['satisfaction'].value_counts()

labels = ['Neutral or Dissatisfied', 'Satisfied']

sizes = target_counts.values
```

```
plt.figure(figsize=(6, 6))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140, colors=['#ff9999','#66b3ff'])
plt.title('Satisfaction Class Distribution (Train Set)')
plt.axis('equal')
plt.show()



# Histograms for numerical features
numeric_cols = train_df.select_dtypes(include=['int64', 'float64']).columns

train_df[numeric_cols].hist(bins=30, figsize=(18, 12), color='lightblue', edgecolor='black')
plt.suptitle("Distribution of Numerical Features", fontsize=16)
plt.tight_layout()
plt.show()



import seaborn as sns

# Correlation heatmap for numeric variables
plt.figure(figsize=(14, 10))
sns.heatmap(train_df.select_dtypes(include='number').corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap (Train Set)")
plt.show()
```

```
# Drop ID columns
columns_to_drop = ['Unnamed: 0', 'id']
train_df.drop(columns=columns_to_drop, axis=1, inplace=True)
test_df.drop(columns=columns_to_drop, axis=1, inplace=True)

print("Remaining columns after dropping ID columns:")
train_df.columns.tolist()



# Encode target variable ('satisfaction') using replace
train_df['satisfaction'] = train_df['satisfaction'].replace({
    'satisfied': 1,
    'neutral or dissatisfied': 0
})

test_df['satisfaction'] = test_df['satisfaction'].replace({
    'satisfied': 1,
    'neutral or dissatisfied': 0
})

print("Target encoding done.")



# Handle missing values using mean
numeric_cols = train_df.select_dtypes(include=['int64', 'float64']).columns
```

```
for col in numeric_cols:
    train_df[col].fillna(train_df[col].mean(), inplace=True)
    test_df[col].fillna(test_df[col].mean(), inplace=True)

print("Missing values handled using mean imputation.")



from sklearn.preprocessing import LabelEncoder
import joblib
import os

# Create directory to store encoders
os.makedirs("svm_encoders", exist_ok=True)

# Categorical columns
categorical_cols = train_df.select_dtypes(include='object').columns

# Encode each categorical column separately
label_encoders = {}

for col in categorical_cols:
    le = LabelEncoder()
    train_df[col] = le.fit_transform(train_df[col])
    test_df[col] = le.transform(test_df[col])
```

```
    # Save encoder for future use
    joblib.dump(le, f"svm_encoders/{col}_encoder.pkl")
    label_encoders[col] = le

print("Categorical features encoded and LabelEncoders saved.")



from sklearn.preprocessing import StandardScaler

# Separate features and labels
X_train_full = train_df.drop('satisfaction', axis=1)
y_train_full = train_df['satisfaction']

X_test_full = test_df.drop('satisfaction', axis=1)
y_test_full = test_df['satisfaction']

# Apply StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_full)
X_test_scaled = scaler.transform(X_test_full)

print("Feature scaling (normalization) complete.")
```

```
from sklearn.model_selection import train_test_split

# Split the scaled training data
X_train, X_val, y_train, y_val = train_test_split(
    X_train_scaled,
    y_train_full,
    test_size=0.2,
    random_state=42,
    # stratify=y_train_full
)

print("Training set shape:", X_train.shape)
print("Validation set shape:", X_val.shape)


from sklearn.svm import SVC

# Initialize SVM model
svm_model = SVC(
    kernel='rbf',       # RBF Kernel (good for most non-linear data)
    C=1.0,              # Regularization parameter (can tune later)
    probability=True,   # Enable probability outputs for ROC curve
    random_state=42
)
```

```
# Train the model
svm_model.fit(X_train, y_train)

print(" ✅ SVM model training complete.")



from sklearn.metrics import accuracy_score

# Predict on training set itself
y_train_pred = svm_model.predict(X_train)

# Training Accuracy
train_accuracy = accuracy_score(y_train, y_train_pred)
print(f" ✅ SVM - Training Accuracy: {train_accuracy * 100:.2f}%")



from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Confusion matrix for validation set
cm_val = confusion_matrix(y_val, y_val_pred)

plt.figure(figsize=(6, 4))
sns.heatmap(cm_val, annot=True, fmt='d', cmap='Blues',
```

```
        xticklabels=['Dissatisfied', 'Satisfied'],
        yticklabels=['Dissatisfied', 'Satisfied'])
plt.title("SVM - Confusion Matrix (Validation Set)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()



from sklearn.metrics import classification_report

# Print classification report
print("SVM - Classification Report (Validation Set):\n")
print(classification_report(y_val, y_val_pred, target_names=['Dissatisfied', 'Satisfied']))



from sklearn.metrics import roc_curve, auc

# ROC curve and AUC
fpr_val, tpr_val, _ = roc_curve(y_val, y_val_proba)
roc_auc_val = auc(fpr_val, tpr_val)

plt.figure(figsize=(6, 4))
plt.plot(fpr_val, tpr_val, label=f"AUC = {roc_auc_val:.2f}", color='navy')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.title("SVM - ROC Curve (Validation Set)")
```

```
plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.legend()

plt.grid()

plt.show()
```

IT21156960

```
from google.colab import files

uploaded = files.upload()
```

```
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
```

```
train_df = pd.read_csv("train.csv")

test_df = pd.read_csv("test.csv")
```

```
print("Train data shape:", train_df.shape)
print("Test data shape:", test_df.shape)



train_df.head()


test_df.tail()


train_df.info()


print(train_df.isnull().sum())


train_df['satisfaction'].value_counts(normalize=True) * 100


plt.figure(figsize=(6, 4))
sns.countplot(data=train_df, x='satisfaction', palette='viridis')
plt.title("Target Variable Distribution")
plt.show()



plt.figure(figsize=(14, 10))
sns.heatmap(train_df.select_dtypes(include='number').corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```

```
columns_to_drop = ['Unnamed: 0', 'id']
train_df.drop(columns=columns_to_drop, axis=1, inplace=True)
test_df.drop(columns=columns_to_drop, axis=1, inplace=True)

print("Remaining columns:", train_df.columns.tolist())



test_df.dropna(inplace=True)

print(train_df['satisfaction'].value_counts(normalize=True) * 100)



train_df['satisfaction'] = train_df['satisfaction'].map({
    'satisfied': 1,
    'neutral or dissatisfied': 0
})
test_df['satisfaction'] = test_df['satisfaction'].map({
    'satisfied': 1,
    'neutral or dissatisfied': 0
})

print(train_df['satisfaction'].value_counts(normalize=True) * 100)
print(test_df['satisfaction'].value_counts(normalize=True) * 100)
```

```
print("Missing values before handling:\n")
print(train_df.isnull().sum())

numeric_columns = train_df.select_dtypes(include=['float64', 'int64']).columns
train_df[numeric_columns] = train_df[numeric_columns].fillna(train_df[numeric_columns].median())

test_df[numeric_columns] = test_df[numeric_columns].fillna(test_df[numeric_columns].median())

print("\nMissing values after handling:\n")
print(train_df.isnull().sum())



label_encoders = {}

categorical_cols = ['Gender', 'Customer Type', 'Type of Travel', 'Class']

for col in categorical_cols:
    le = LabelEncoder()
    train_df[col] = le.fit_transform(train_df[col])
    test_df[col] = le.transform(test_df[col])
    label_encoders[col] = le



X = train_df.drop('satisfaction', axis=1)
```

```
y = train_df['satisfaction']


X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)



print("Training set shape:", X_train.shape)
print("Validation set shape:", X_val.shape)



rf_model = RandomForestClassifier(
    n_estimators=100,
    max_depth=None,
    random_state=42,
    class_weight='balanced'
)

rf_model.fit(X_train, y_train)

y_pred = rf_model.predict(X_val)

accuracy = accuracy_score(y_val, y_pred)
print("Validation Accuracy:", round(accuracy * 100, 2), "%")
```

```
print(classification_report(y_val, y_pred))




y_proba = rf_model.predict_proba(X_val)[:, 1]

fpr, tpr, _ = roc_curve(y_val, y_proba)

roc_auc = auc(fpr, tpr)


plt.figure(figsize=(6, 4))

plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}")

plt.plot([0, 1], [0, 1], linestyle='--')

plt.title("ROC Curve")

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.legend()

plt.show()



X_test = test_df.drop('satisfaction', axis=1)

y_test = test_df['satisfaction']


y_test_pred = rf_model.predict(X_test)


print("Final Evaluation on Test Set:")
```

```
print("Accuracy:", round(accuracy_score(y_test, y_test_pred) * 100, 2), "%")

print("\nClassification Report:\n")

print(classification_report(y_test, y_test_pred))



import numpy as np


cm = confusion_matrix(y_test, y_test_pred)

cm_sum = np.sum(cm)

cm_percent = cm / cm_sum * 100


labels = np.array([

    [f"{cm[0,0]} ({cm_percent[0,0]:.1f}%)", f"{cm[0,1]} ({cm_percent[0,1]:.1f}%)"],

    [f"{cm[1,0]} ({cm_percent[1,0]:.1f}%)", f"{cm[1,1]} ({cm_percent[1,1]:.1f}%)"]

])


plt.figure(figsize=(6, 4))

sns.heatmap(cm, annot=labels, fmt='', cmap='Blues', xticklabels=['Dissatisfied', 'Satisfied'],
yticklabels=['Dissatisfied', 'Satisfied'])

plt.title("Confusion Matrix with Percentages - Test Set")

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.show()
```

IT24469144

```
from google.colab import drive

import pandas as pd


drive.mount('/content/drive')


!ls 'drive/MyDrive/ML project'


train_path = "/content/drive/MyDrive/ML project/train.csv"

test_path = "/content/drive/MyDrive/ML project/test.csv"


train_df = pd.read_csv(train_path)

test_df = pd.read_csv(test_path)


print("Train shape:", train_df.shape)

print("Test shape:", test_df.shape)


print(train_df.info())


print(train_df.describe())


print(train_df['satisfaction'].value_counts())


import seaborn as sns
```

```
import matplotlib.pyplot as plt



# Get all numerical columns
numerical_cols = train_df.select_dtypes(include=['int64', 'float64']).columns

# Plot histograms
train_df[numerical_cols].hist(bins=30, figsize=(18, 12), color='steelblue', edgecolor='black')
plt.suptitle("Histograms of Numerical Features", fontsize=16)
plt.tight_layout()
plt.show()



columns_to_drop = ['Unnamed: 0', 'id']
train_df.drop(columns=columns_to_drop, axis=1, inplace=True)
test_df.drop(columns=columns_to_drop, axis=1, inplace=True)



# Encode satisfaction using .replace() instead of map/lambda
train_df['satisfaction'] = train_df['satisfaction'].replace({
    'satisfied': 1,
    'neutral or dissatisfied': 0
})

test_df['satisfaction'] = test_df['satisfaction'].replace({
```

```
    'satisfied': 1,
    'neutral or dissatisfied': 0
})


numeric_cols = train_df.select_dtypes(include=['float64', 'int64']).columns

for col in numeric_cols:
    train_df[col].fillna(train_df[col].mean(), inplace=True)
    test_df[col].fillna(test_df[col].mean(), inplace=True)


from sklearn.preprocessing import LabelEncoder
import joblib

# Step 4: Encode categorical features using LabelEncoder
label_encoders = {}
categorical_cols = train_df.select_dtypes(include='object').columns


for col in categorical_cols:
    le = LabelEncoder()
    train_df[col] = le.fit_transform(train_df[col])
    test_df[col] = le.transform(test_df[col])
```

```
    label_encoders[col] = le

print(" ✅ Categorical features encoded and encoders saved.")

from sklearn.model_selection import train_test_split

# Separate features and target
X = train_df.drop('satisfaction', axis=1)
y = train_df['satisfaction']

# Split into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print("Training set shape:", X_train.shape)
print("Validation set shape:", X_val.shape)


from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Initialize the Decision Tree model
dt_model = DecisionTreeClassifier(
    random_state=42,
```

```
    class_weight='balanced'  # helpful for slight imbalance
)

# Train the model
dt_model.fit(X_train, y_train)

# Predict on validation set
y_val_pred = dt_model.predict(X_val)

# Evaluate accuracy
val_accuracy = accuracy_score(y_val, y_val_pred)
print(f"Decision Tree - Validation Accuracy: {val_accuracy * 100:.2f}%")



import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Confusion matrix
cm = confusion_matrix(y_val, y_val_pred)

plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens',
            xticklabels=['Dissatisfied', 'Satisfied'],
            yticklabels=['Dissatisfied', 'Satisfied'])
```

```
plt.title("Decision Tree - Confusion Matrix (Validation Set)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()



from sklearn.metrics import classification_report

# Textual evaluation metrics
print("Decision Tree - Classification Report (Validation Set):\n")
print(classification_report(y_val, y_val_pred, target_names=['Dissatisfied', 'Satisfied']))



from sklearn.metrics import roc_curve, auc

# Predict probabilities for ROC
y_val_proba = dt_model.predict_proba(X_val)[:, 1]

fpr, tpr, _ = roc_curve(y_val, y_val_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}", color='darkgreen')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.title("Decision Tree - ROC Curve (Validation Set)")
```

```
plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.legend()

plt.grid()

plt.show()



# Separate features and labels in test set

X_test = test_df.drop('satisfaction', axis=1)

y_test = test_df['satisfaction']



# Predict on the test set

y_test_pred = dt_model.predict(X_test)

# Accuracy score

from sklearn.metrics import accuracy_score

test_acc = accuracy_score(y_test, y_test_pred)

print(f"Decision Tree - Test Accuracy: {test_acc * 100:.2f}%")



from sklearn.metrics import confusion_matrix

import seaborn as sns

import matplotlib.pyplot as plt
```

```
cm_test = confusion_matrix(y_test, y_test_pred)


plt.figure(figsize=(6, 4))
sns.heatmap(cm_test, annot=True, fmt='d', cmap='YlGn',
        xticklabels=['Dissatisfied', 'Satisfied'],
        yticklabels=['Dissatisfied', 'Satisfied'])
plt.title("Decision Tree - Confusion Matrix (Test Set)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()



from sklearn.metrics import classification_report

print("Decision Tree - Classification Report (Test Set):\n")
print(classification_report(y_test, y_test_pred, target_names=['Dissatisfied', 'Satisfied']))



from sklearn.metrics import roc_curve, auc

y_test_proba = dt_model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_test_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6, 4))
```

```
plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}", color='forestgreen')

plt.plot([0, 1], [0, 1], linestyle='--', color='gray')

plt.title("Decision Tree - ROC Curve (Test Set)")

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.legend()

plt.grid()

plt.show()
```