Machine Learning Final Project: Passenger Satisfaction

Dataset link here

1. Problem Selection

Problem Definition: The goal is to predict airline passenger satisfaction based on various factors such as service quality, flight distance, and delays.

Importance:

- Understanding passenger satisfaction helps airlines improve their services.
- Identifying key dissatisfaction drivers can lead to targeted improvements.
- High satisfaction leads to customer loyalty and better business perception.

Role of Machine Learning: Machine Learning allows us to analyze complex patterns in the data and build a predictive model that can classify a passenger as 'satisfied' or 'neutral/dissatisfied' based on their feedback details.

In [ ]:

```
# Setup and Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

from imblearn.over_sampling import SMOTE

# Models
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

# Set Layout
sns.set(style="whitegrid")
%matplotlib inline
```

```
import warnings
warnings.filterwarnings('ignore')
```

2. Dataset Requirements & Loading

The selected dataset is `satisfaction.csv` located in the `data` folder. It contains over 100,000 records and more than 20 features, fulfilling the project requirements.

In [3]:
```
df = pd.read_csv('data/satisfaction.csv')
df.head()
```

Out[3]:

	id	satisfaction_v2	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Seat comfort
0	11112	satisfied	Female	Loyal Customer	65	Personal Travel	Eco	265	0
1	110278	satisfied	Male	Loyal Customer	47	Personal Travel	Business	2464	0
2	103199	satisfied	Female	Loyal Customer	15	Personal Travel	Eco	2138	0
3	47462	satisfied	Female	Loyal Customer	60	Personal Travel	Eco	623	0
4	120011	satisfied	Female	Loyal Customer	70	Personal Travel	Eco	354	0

5 rows × 24 columns

3. Exploratory Data Analysis (EDA) & Feature Understanding

i. Dataset Overview

In [4]:
```
print(f"Dataset Shape: {df.shape}")
print("\nColumn Info:")
df.info()
```

```
Dataset Shape: (129880, 24)

Column Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 129880 entries, 0 to 129879
Data columns (total 24 columns):
 #   Column                             Non-Null Count   Dtype  
---  ------                             --------------   -----  
 0   id                                 129880 non-null  int64  
 1   satisfaction_v2                    129880 non-null  object 
 2   Gender                             129880 non-null  object 
 3   Customer Type                      129880 non-null  object 
 4   Age                                129880 non-null  int64  
 5   Type of Travel                     129880 non-null  object 
 6   Class                              129880 non-null  object 
 7   Flight Distance                    129880 non-null  int64  
 8   Seat comfort                       129880 non-null  int64  
 9   Departure/Arrival time convenient  129880 non-null  int64  
 10  Food and drink                     129880 non-null  int64  
 11  Gate location                      129880 non-null  int64  
 12  Inflight wifi service              129880 non-null  int64  
 13  Inflight entertainment             129880 non-null  int64  
 14  Online support                     129880 non-null  int64  
 15  Ease of Online booking             129880 non-null  int64  
 16  On-board service                   129880 non-null  int64  
 17  Leg room service                   129880 non-null  int64  
 18  Baggage handling                   129880 non-null  int64  
 19  Checkin service                    129880 non-null  int64  
 20  Cleanliness                        129880 non-null  int64  
 21  Online boarding                    129880 non-null  int64  
 22  Departure Delay in Minutes         129880 non-null  int64  
 23  Arrival Delay in Minutes           129487 non-null  float64
dtypes: float64(1), int64(18), object(5)
memory usage: 23.8+ MB
```

In [ ]:
```
df.describe()  # Generate descriptive statistics for numerical features
```

Out[ ]:

	id	Age	Flight Distance	Seat comfort	Departure/Arrival time convenient	
count	129880.000000	129880.000000	129880.000000	129880.000000	129880.000000	129
mean	64940.500000	39.427957	1981.409055	2.838597	2.990645	
std	37493.270818	15.119360	1027.115606	1.392983	1.527224	
min	1.000000	7.000000	50.000000	0.000000	0.000000	
25%	32470.750000	27.000000	1359.000000	2.000000	2.000000	
50%	64940.500000	40.000000	1925.000000	3.000000	3.000000	
75%	97410.250000	51.000000	2544.000000	4.000000	4.000000	
max	129880.000000	85.000000	6951.000000	5.000000	5.000000	

In [6]:

```python
# Summary statistics for Categorical features
print(f"{'=' * 30}")
categorical_cols = df.select_dtypes(include=['object'])
print("Categorical Features Summary:")
print(f"{'=' * 30}")
categorical_cols.describe()
```

```
==============================
Categorical Features Summary:
==============================
```

Out[6]:

	satisfaction_v2	Gender	Customer Type	Type of Travel	Class
count	129880	129880	129880	129880	129880
unique	2	2	2	2	3
top	satisfied	Female	Loyal Customer	Business travel	Business
freq	71087	65899	106100	89693	62160

Feature	Description
id	Unique identifier for each passenger record (numerical).
Gender	Gender of the passenger (categorical: Male or Female).
Customer Type	Loyalty status of the customer (categorical: Loyal Customer or disloyal Customer).
Age	The age of the passenger (numerical, in years).
Type of Travel	Purpose of the travel (categorical: Business travel or Personal Travel).
Class	The travel class of the passenger (categorical: Business, Eco, Eco Plus).

Feature	Description
Flight Distance	The distance of the flight in miles (numerical).
Seat comfort	Passenger rating of seat comfort (ordinal, typically on a scale of 0-5).
Departure/Arrival time convenient	Passenger rating of how convenient the departure and arrival times were (scale 0-5).
Food and drink	Passenger rating of the food and beverages provided (scale 0-5).
Gate location	Passenger rating of the convenience of the gate location (scale 0-5).
Inflight wifi service	Passenger rating of the in-flight Wi-Fi service (scale 0-5).
Inflight entertainment	Passenger rating of in-flight entertainment options (scale 0-5).
Online support	Passenger rating of online customer support (scale 0-5).
Ease of Online booking	Passenger rating of how easy online booking was (scale 0-5).
On-board service	Passenger rating of the service provided by cabin crew on board (scale 0-5).
Leg room service	Passenger rating of leg room space and service (scale 0-5).
Baggage handling	Passenger rating of how baggage was handled (scale 0-5).
Checkin service	Passenger rating of the check-in process (scale 0-5).
Cleanliness	Passenger rating of overall cleanliness (scale 0-5).
Online boarding	Passenger rating of the online boarding process (scale 0-5).
Departure Delay in Minutes	Delay in departure time (numerical, in minutes).
Arrival Delay in Minutes	Delay in arrival time (numerical, in minutes).
satisfaction_v2 (Target variable)	The overall passenger satisfaction level (typically binary: satisfied or neutral/dissatisfied).

ii. Feature Type Identification

- **Target Variable**: `satisfaction_v2` (Binary Classification)
- **Nominal Categorical**: `Gender`, `Customer Type`, `Type of Travel`
- **Ordinal Categorical**: `Class` (Eco < Eco Plus < Business)
- **Numerical**: `Age`, `Flight Distance`, `Departure Delay in Minutes`, `Arrival Delay in Minutes`
- **Survey Features (Ordinal/Numerical)**: `Seat comfort`, `Cleanliness`, etc. (Rated 0-5)

Feature	Type	Explanation
id	Numerical	Unique identifier for each record; numeric values with no inherent order or meaning beyond identification.
Gender	Binary Categorical	Two categories: Male or Female; no natural order, just distinct groups.
Customer Type	Binary Categorical	Two categories: Loyal Customer or disloyal Customer; no inherent order.
Age	Numerical	Continuous (or discrete) integer values representing years; has meaningful magnitude and ratios.
Type of Travel	Binary Categorical	Two categories: Business travel or Personal Travel; no natural ordering.
Class	Ordinal Categorical	Three categories: Eco, Eco Plus, Business; clear natural ordering based on level of service/prestige (Eco < Eco Plus < Business).
Flight Distance	Numerical	Continuous positive values in miles; meaningful differences and ratios.
Seat comfort	Ordinal Categorical	Rated on a discrete scale (typically 0-5); higher values indicate better comfort, so order matters but intervals are not necessarily equal.
Departure/Arrival time convenient	Ordinal Categorical	Rated on a scale (0-5); ordered levels of satisfaction/convenience.
Food and drink	Ordinal Categorical	Rated on a scale (0-5); ordered quality/satisfaction levels.
Gate location	Ordinal Categorical	Rated on a scale (0-5); ordered convenience levels.
Inflight wifi service	Ordinal Categorical	Rated on a scale (0-5); ordered quality of service.
Inflight entertainment	Ordinal Categorical	Rated on a scale (0-5); ordered satisfaction with entertainment.
Online support	Ordinal Categorical	Rated on a scale (0-5); ordered quality of support.
Ease of Online booking	Ordinal Categorical	Rated on a scale (0-5); ordered ease of use.
On-board service	Ordinal Categorical	Rated on a scale (0-5); ordered quality of crew service.
Leg room service	Ordinal Categorical	Rated on a scale (0-5); ordered satisfaction with leg room.
Baggage handling	Ordinal Categorical	Rated on a scale (0-5); ordered quality of handling.

Feature	Type	Explanation
Checkin service	Ordinal Categorical	Rated on a scale (0-5); ordered quality of check-in process.
Cleanliness	Ordinal Categorical	Rated on a scale (0-5); ordered level of cleanliness.
Online boarding	Ordinal Categorical	Rated on a scale (0-5); ordered ease/satisfaction with online boarding.
Departure Delay in Minutes	Numerical	Continuous non-negative values in minutes; meaningful magnitude and differences.
Arrival Delay in Minutes	Numerical	Continuous non-negative values in minutes; meaningful magnitude and differences.
satisfaction_v2 (Target)	Binary Categorical	Two classes: typically "satisfied" vs "neutral or dissatisfied"; used for binary classification, no inherent order beyond the label.

iii. Missing Values Analysis

In [7]:
```
missing_values = df.isnull().sum()
missing_values = missing_values[missing_values > 0]
missing_percentage = (missing_values / len(df)) * 100

print("Missing Values:")
print(pd.concat([missing_values, missing_percentage], axis=1, keys=['Count', 'Perce
```

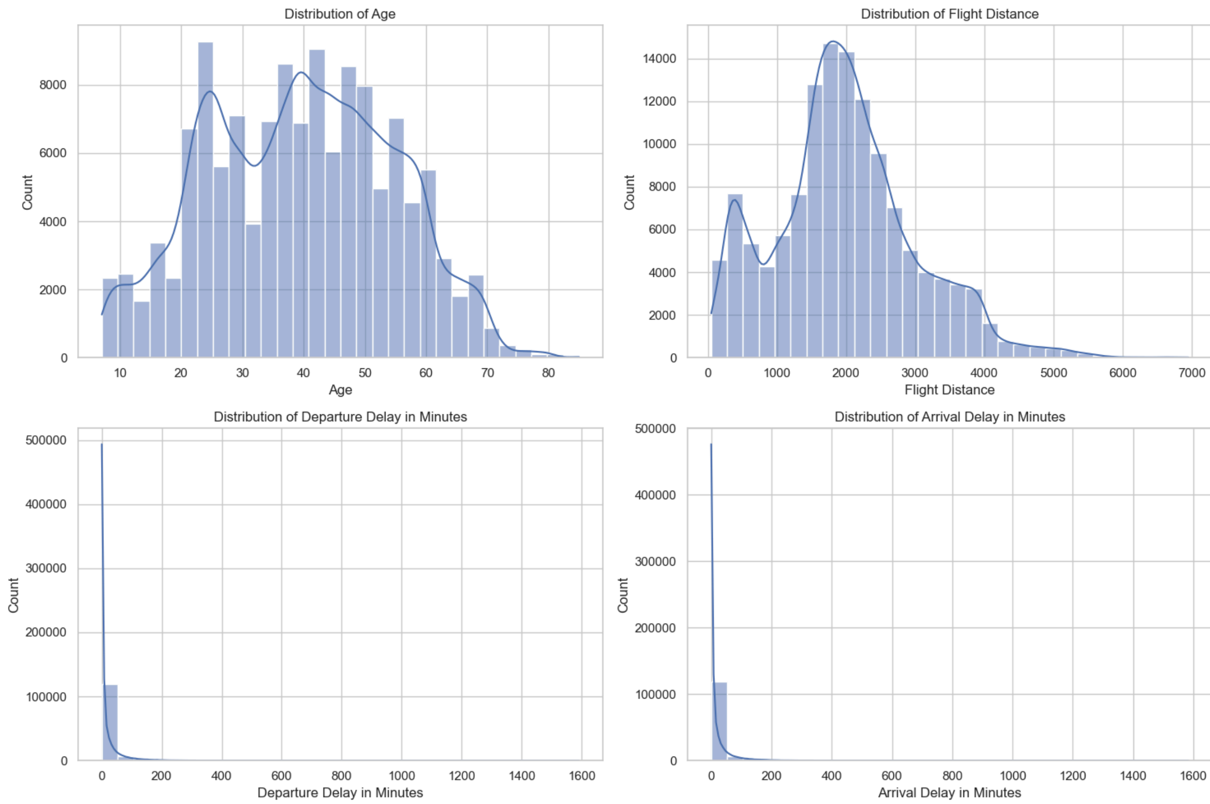
```
Missing Values:
                         Count  Percentage
Arrival Delay in Minutes    393    0.302587
```

Handling Strategy:

- `Arrival Delay in Minutes` has missing values. Since it is a numerical feature, we will impute it using the **Median** or **Mean**. Given delays might be skewed, Median is often safer.

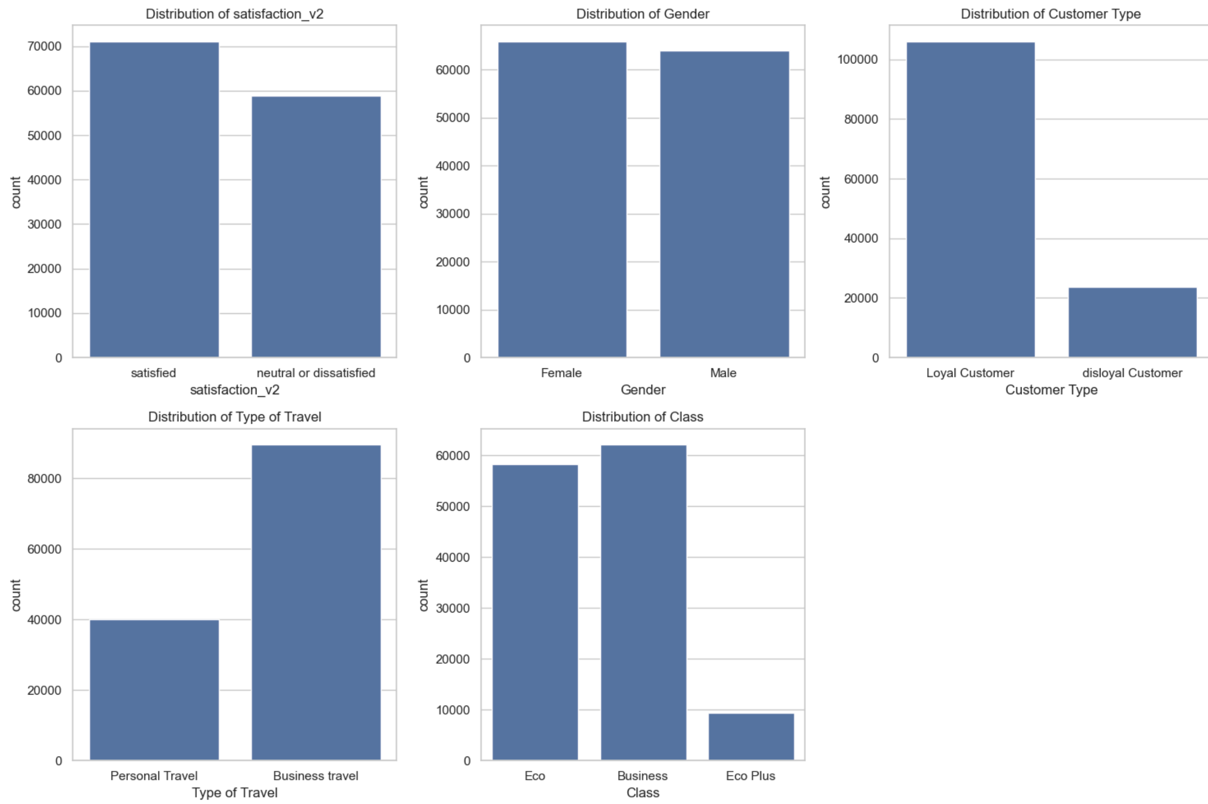
iv. Feature Distribution

In [8]:
```
# Numerical Distributions
numerical_cols = ['Age', 'Flight Distance', 'Departure Delay in Minutes', 'Arrival

plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical_cols, 1):
    plt.subplot(2, 2, i)
    sns.histplot(df[col], kde=True, bins=30)
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()
```



In [9]:

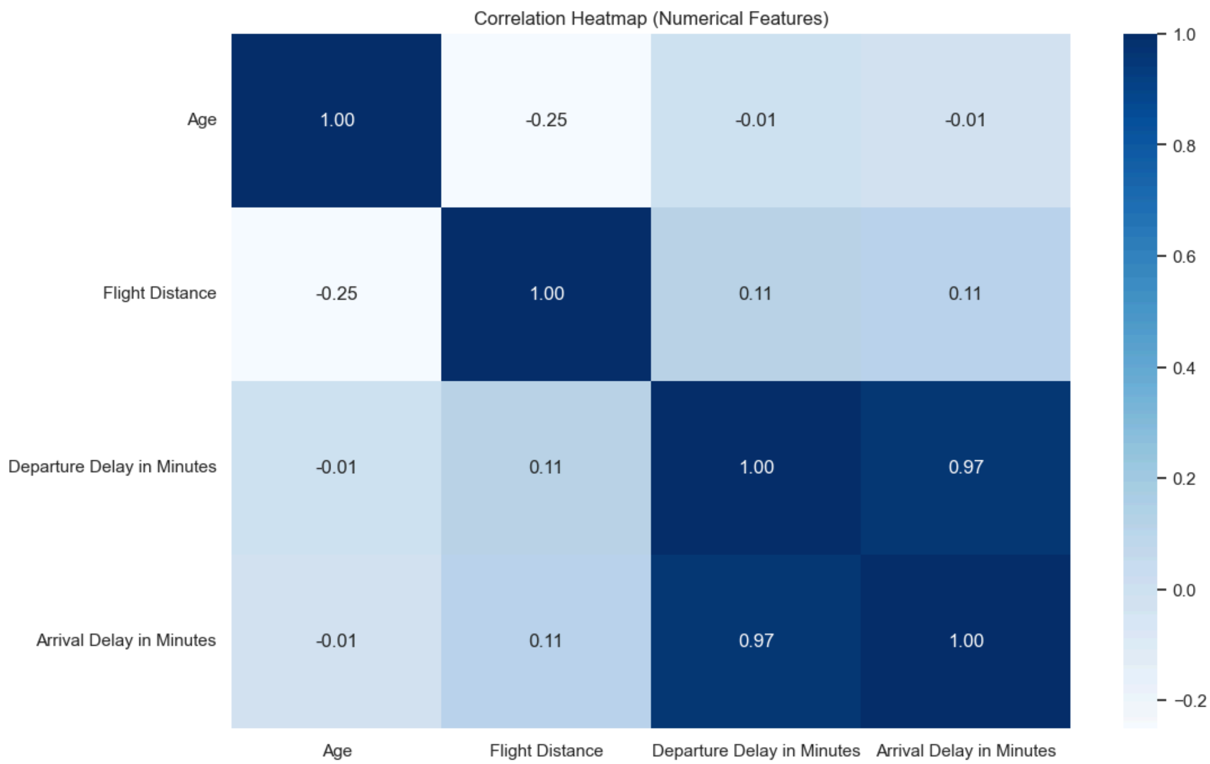
```
# Categorical Distributions
categorical_cols = df.select_dtypes(include=['object'])

plt.figure(figsize=(15, 10))
for i, col in enumerate(categorical_cols, 1):
    plt.subplot(2, 3, i)
    sns.countplot(x=col, data=df)
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()
```



v. Feature Relationships

In [10]:

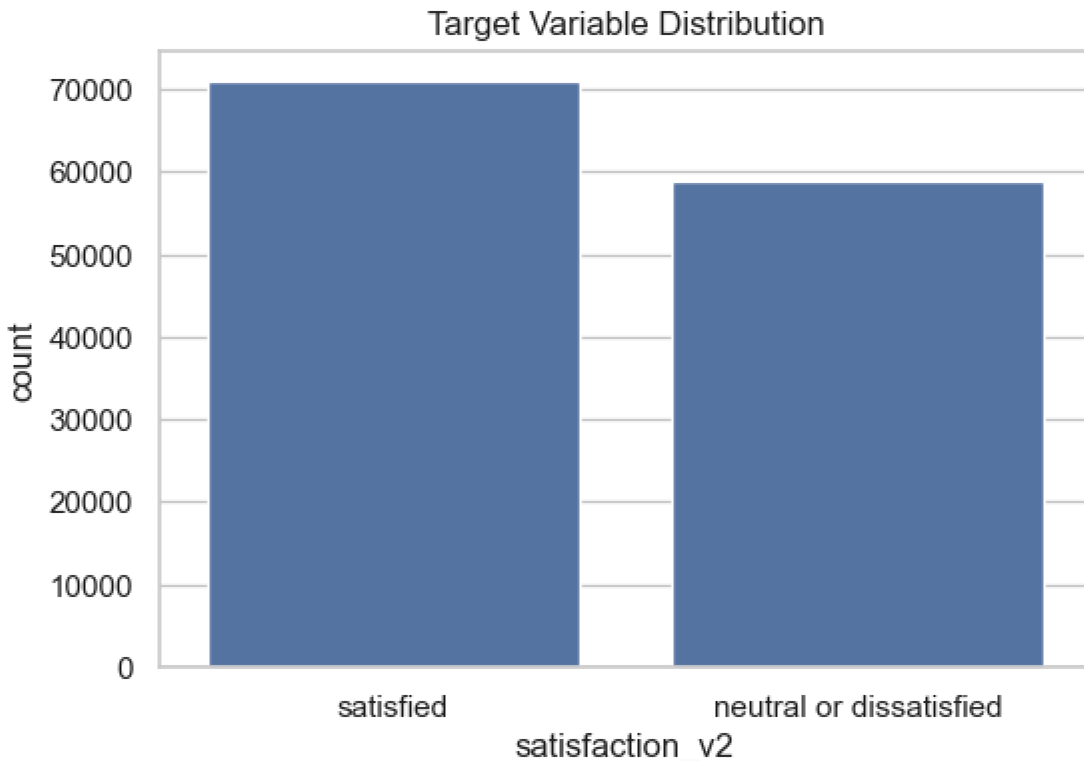
```
plt.figure(figsize=(12, 8))
# Correlation heatmap for numerical features
sns.heatmap(df[numerical_cols].corr(), annot=True, cmap='Blues', fmt='.2f')
plt.title('Correlation Heatmap (Numerical Features)')
plt.show()
```



vi. Class Distribution

In [11]:
```
target_counts = df['satisfaction_v2'].value_counts()
print("Class Distribution:")
print(target_counts)

plt.figure(figsize=(6, 4))
sns.countplot(x='satisfaction_v2', data=df)
plt.title('Target Variable Distribution')
plt.show()
```

```
Class Distribution:
satisfaction_v2
satisfied                  71087
neutral or dissatisfied    58793
Name: count, dtype: int64
```



Observation: Is the dataset balanced? Not really satisfied is a little bit more than dissatisfied, we might need balancing (SMOTE) but will come to it later.

Summary of Insights

Feature Types and Preprocessing Needs

- **Nominal Categorical** (no order): `Gender`, `Customer Type`, `Type of Travel` → Require **One-Hot Encoding** to convert into binary columns (drop_first=True to avoid multicollinearity).
- **Binary Categorical (Target)**: `satisfaction_v2` (`satisfied` / `neutral or dissatisfied`) → Map to 1/0 for binary classification.
- **Ordinal Categorical** (natural order, rated 0–5): `Class` (Eco < Eco Plus < Business), and all survey features (`Seat comfort`, `Cleanliness`, `Inflight wifi service`, `Online boarding`, etc.) → Use **Ordinal Encoding** for `Class`; survey ratings can be treated as ordinal/numerical (no encoding needed as they are already integer scales).
- **Numerical**: `Age`, `Flight Distance`, `Departure Delay in Minutes`, `Arrival Delay in Minutes` → Continuous; require **imputation** for missing values (e.g., median for `Arrival Delay in Minutes`, which has ~0.3% missing) and potential outlier handling (delays are heavily right-skewed).
- **ID column**: Unique identifier → Drop as it provides no predictive value.

Whether Scaling is Needed

Yes, **feature scaling is required**. Numerical features have vastly different ranges (e.g., `Flight Distance` up to 6951, `Age` up to 85, delays highly skewed). Models like KNN and Logistic Regression are sensitive to scale, so **StandardScaler** (mean=0, std=1) should be applied after train-test split (fit on training data only) to prevent features with larger magnitudes from dominating.

Whether Balancing is Needed

Yes, **class balancing is recommended**. The target `satisfaction_v2` is mildly imbalanced:

- Satisfied: 71,087 (~54.7%)
- Neutral or dissatisfied: 58,793 (~45.3%)

The minority class ratio is ~0.83, which can bias models toward the majority class. Techniques like **SMOTE** (oversampling the minority class) were applied to achieve a balanced dataset before modeling.

Important Observations Before Modeling

- Distributions: `Age` is roughly normal (peak around 40); `Flight Distance` right-skewed; both delay features are heavily right-skewed with most values near 0 and long tails (outliers common).
- Strong correlation between `Departure Delay in Minutes` and `Arrival Delay in Minutes` (~0.97), indicating near redundancy.
- Minimal missing data (only 393 values in `Arrival Delay in Minutes`); easily handled with median imputation.
- Categorical features show clear dominance (e.g., most passengers are Loyal Customers, travel for Business, and fly Business class).
- No severe multicollinearity among other numerical features; survey ratings (0–5) behave like ordered categories with meaningful magnitude.

These insights guide robust preprocessing: drop ID, impute delays, encode categoricals appropriately, scale numerical features, and balance classes to ensure fair and effective model training.

5. Data Preprocessing

5.1 Handling Missing Values

We only have missing values in `Arrival Delay in Minutes` so we will replace the missing values with the median

In [12]:
```
# Drop ID as it is not a feature
if 'id' in df.columns:
    df = df.drop('id', axis=1)
```

```
# Impute Arrival Delay with Median (Since we saw above that it has missing values,
imputer = SimpleImputer(strategy='median')
df['Arrival Delay in Minutes'] = imputer.fit_transform(df[['Arrival Delay in Minute
print("We will impute using median since it's safer because delays are skewed and h
print("Missing values after imputation:", df.isnull().sum().sum())
```

```
We will impute using median since it's safer because delays are skewed and have outl
iers
Missing values after imputation: 0
```

5.2 Encoding Categorical Features

- **Target**: Satisfied -> 1, Neutral or dissatisfied -> 0
- **Class**: Eco -> 0, Eco Plus -> 1, Business -> 2 (Ordinal Encoding)
- **Other Categorical**: One-Hot Encoding

In [13]:
```
# Target Encoding
df['satisfaction_v2'] = df['satisfaction_v2'].map({'satisfied': 1, 'neutral or diss

# Ordinal Encoding for Class
class_mapping = {'Eco': 0, 'Eco Plus': 1, 'Business': 2}
df['Class'] = df['Class'].map(class_mapping)

# Identify nominal categorical columns for One-Hot Encoding
nominal_cols = ['Gender', 'Customer Type', 'Type of Travel']
# We will use pd.get_dummies for simplicity.
# Let's use get_dummies here to make the dataframe ready
df = pd.get_dummies(df, columns=nominal_cols, drop_first=True)

print("Dataframe shape after encoding:", df.shape)
df.head()
```

```
Dataframe shape after encoding: (129880, 23)
```

Out[13]:

	satisfaction_v2	Age	Class	Flight Distance	Seat comfort	Departure/Arrival time convenient	Food and drink	Gate location	Inflig w servi
0	1	65	0	265	0	0	0	2	
1	1	47	2	2464	0	0	0	3	
2	1	15	0	2138	0	0	0	3	
3	1	60	0	623	0	0	0	3	
4	1	70	0	354	0	0	0	3	

5 rows × 23 columns

5.3 Data Balancing

```
In [14]: X = df.drop('satisfaction_v2', axis=1)
y = df['satisfaction_v2']

count_0 = (y == 0).sum()
count_1 = (y == 1).sum()
ratio = min(count_0, count_1) / max(count_0, count_1)
print(f"Class 0: {count_0}, Class 1: {count_1}, Minority ratio: {ratio:.2f}")

if ratio < 0.95:  # Imbalanced if not close to 1:1
    print("Dataset is imbalanced. Applying SMOTE.")
    smote = SMOTE(random_state=42)
    X_balanced, y_balanced = smote.fit_resample(X, y)
    print(f"After SMOTE: Class 0: {(y_balanced == 0).sum()}, Class 1: {(y_balanced
    X = X_balanced
    y = y_balanced
else:
    print("Dataset is balanced; no balancing needed.")
```

```
Class 0: 58793, Class 1: 71087, Minority ratio: 0.83
Dataset is imbalanced. Applying SMOTE.
After SMOTE: Class 0: 71087, Class 1: 71087
```

How SMOTE Works

SMOTE (Synthetic Minority Oversampling Technique) is an oversampling method that generates synthetic samples for the minority class to balance the dataset. It works by:

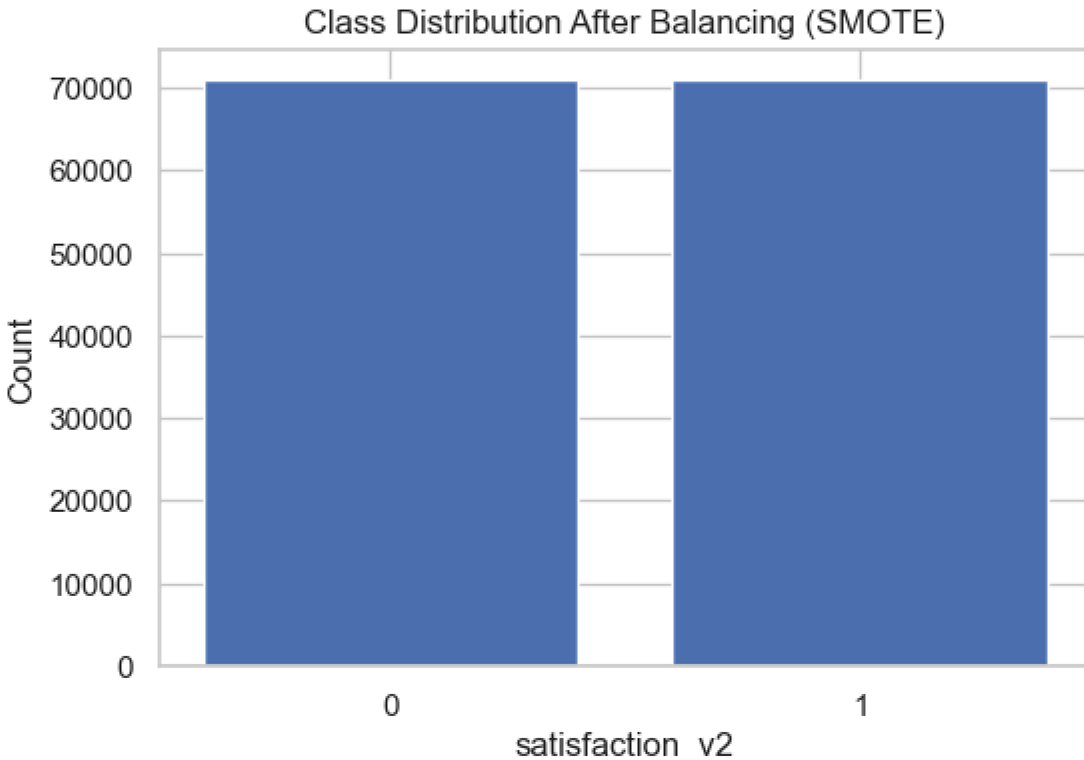
1. **Identify Minority Class**: For each sample in the minority class (e.g., 'neutral or dissatisfied'), find its k nearest neighbors (default k=5) in the feature space.
2. **Select Neighbors**: Randomly choose one of the k neighbors.
3. **Generate Synthetic Sample**: Create a new synthetic sample by interpolating between the original minority sample and the selected neighbor. This is done by taking a random point along the line connecting the two samples in the feature space.
4. **Repeat**: Generate as many synthetic samples as needed to balance the classes (e.g., match the majority class count).

This approach avoids simple duplication of minority samples, which can lead to overfitting, and creates diverse synthetic data that helps models learn better decision boundaries without introducing noise.

Target Distributions after Balancing

```
In [15]: # Count after balancing
after_counts = y_balanced.value_counts().sort_index()

plt.figure(figsize=(6,4))
plt.bar(after_counts.index, after_counts.values)
plt.title("Class Distribution After Balancing (SMOTE)")
plt.xlabel("satisfaction_v2")
plt.ylabel("Count")
```

```
plt.xticks([0, 1])
plt.show()
```



5.4 Dataset Splitting

In [16]:
```
# First, split into train+val and test
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.2, ra
# Then split train_val into train and val
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_si
```

5.5 Feature Scaling

Why Scaling?

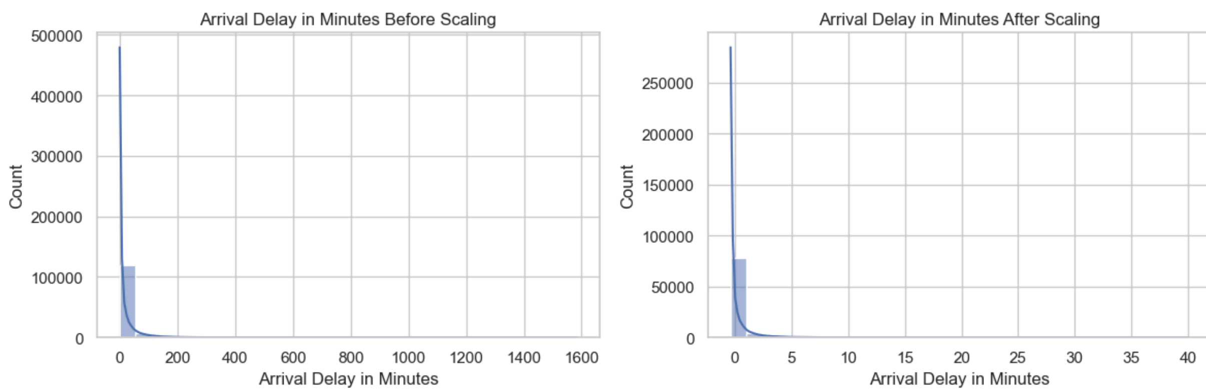
Scaling is applied using StandardScaler to standardize numerical features to mean=0, std=1. This is crucial for distance-based models like KNN and gradient-based models like Logistic Regression to prevent features with larger ranges (e.g., Flight Distance) from dominating.

In [17]:
```
# Scaling (fit on train, transform all)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

print("Training Set Shape:", X_train_scaled.shape)
print("Validation Set Shape:", X_val_scaled.shape)
print("Testing Set Shape:", X_test_scaled.shape)
```

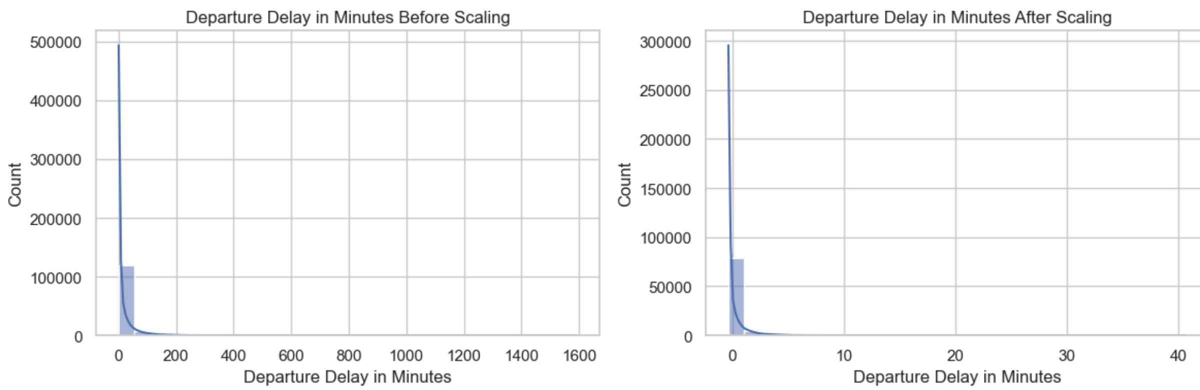
```
Training Set Shape: (85304, 22)
Validation Set Shape: (28435, 22)
Testing Set Shape: (28435, 22)
```

In [18]:
```python
# Create DataFrame for scaled training data
X_train_scaled_df = pd.DataFrame(
    X_train_scaled,
    columns=X_train.columns
)
```

In [19]:
```python
feature = 'Arrival Delay in Minutes'

plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
sns.histplot(df[feature], kde=True, bins=30)
plt.title(f'{feature} Before Scaling')

plt.subplot(1,2,2)
sns.histplot(X_train_scaled_df[feature], kde=True, bins=30)
plt.title(f'{feature} After Scaling')

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



In [20]:
```python
feature = 'Departure Delay in Minutes'

plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
sns.histplot(df[feature], kde=True, bins=30)
plt.title(f'{feature} Before Scaling')

plt.subplot(1,2,2)
sns.histplot(X_train_scaled_df[feature], kde=True, bins=30)
plt.title(f'{feature} After Scaling')

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



6. Model Building & 7. Hyperparameter Tuning

We will implement 5 models using GridSearchCV for hyperparameter tuning.

In [ ]:
```python
# Initialize results dictionary
model_results = {}

def train_evaluate_model(model, param_grid, name):
    print(f"\n--- Training {name} ---")

    # Evaluate with defaults on validation set
    model_default = model.__class__()  # Instantiate default model
    model_default.fit(X_train_scaled, y_train)  # Fit default model on training dat
    y_val_pred_default = model_default.predict(X_val_scaled)  # Predict on validati
    acc_default = accuracy_score(y_val, y_val_pred_default)  # Calculate accuracy
    f1_default = f1_score(y_val, y_val_pred_default)  # Calculate F1 score
    print(f"Default Params - Val Accuracy: {acc_default:.4f}, Val F1: {f1_default:.

    # GridSearchCV on train (uses internal CV)
    grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=3, scorin
    grid_search.fit(X_train_scaled, y_train)  # Perform grid search with cross-vali

    best_model = grid_search.best_estimator_  # Get best model from grid search
    print(f"Best Parameters: {grid_search.best_params_}")

    # Evaluate tuned model on validation
    y_val_pred_tuned = best_model.predict(X_val_scaled)  # Predict with tuned model
    acc_tuned = accuracy_score(y_val, y_val_pred_tuned)  # Tuned accuracy
    f1_tuned = f1_score(y_val, y_val_pred_tuned)  # Tuned F1
    print(f"Tuned - Val Accuracy: {acc_tuned:.4f}, Val F1: {f1_tuned:.4f}")

    # Final evaluation on test
    y_pred = best_model.predict(X_test_scaled)  # Predict on test set
    acc = accuracy_score(y_test, y_pred)  # Test accuracy
    prec = precision_score(y_test, y_pred)  # Test precision
    rec = recall_score(y_test, y_pred)  # Test recall
    f1 = f1_score(y_test, y_pred)  # Test F1
    cm = confusion_matrix(y_test, y_pred)  # Confusion matrix

    print(f"Test Accuracy: {acc:.4f}, Test F1: {f1:.4f}")
    print("Confusion Matrix:\n", cm)
```

```
    print("\nClassification Report:\n", classification_report(y_test, y_pred))

    # Plot Confusion Matrix
    plt.figure(figsize=(5,4))
    sns.heatmap(
        cm,
        annot=True,
        fmt='d',
        cmap='Blues',
        xticklabels=['Predicted 0', 'Predicted 1'],
        yticklabels=['Actual 0', 'Actual 1']
    )
    plt.title(f'Confusion Matrix - {name}')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.tight_layout()
    plt.show()

    model_results[name] = {
        'Default Val Acc': acc_default,
        'Tuned Val Acc': acc_tuned,
        'Test Accuracy': acc,
        'Test Precision': prec,
        'Test Recall': rec,
        'Test F1 Score': f1
    }
    return best_model
```

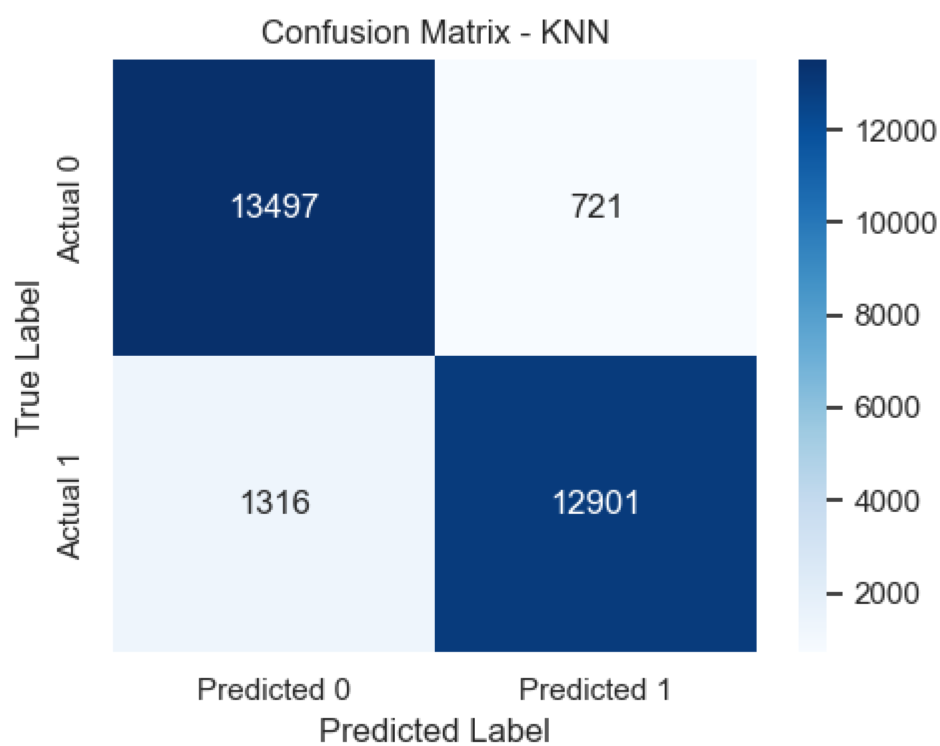
6.1 K-Nearest Neighbors (KNN)

Hyperparameter Selection and Justification for KNN

- **Important Hyperparameters**:
  - `n_neighbors` : Controls model complexity by determining how many neighbors influence a prediction. Fewer neighbors increase variance (overfitting), more increase bias (underfitting).
  - `weights` : 'uniform' gives equal weight to all neighbors; 'distance' weights closer neighbors more, reducing bias for non-linear boundaries.
- **Value Ranges**:
  - `n_neighbors` : [3,5,7,9] - Odd numbers to avoid ties; range balances simplicity (3) vs. stability (9), controlling bias-variance.
  - `weights` : ['uniform', 'distance'] - Tests both weighting schemes to see which reduces overfitting.
- **Selection Criteria**: Best model chosen based on validation accuracy, ensuring no overfitting (stable val/test scores).

In [ ]:
```
knn_params = {'n_neighbors': [3, 5, 7, 9], 'weights': ['uniform', 'distance']} # De
knn_model = train_evaluate_model(KNeighborsClassifier(), knn_params, 'KNN')
```

```
--- Training KNN ---
Default Params - Val Accuracy: 0.9287, Val F1: 0.9274
Fitting 3 folds for each of 8 candidates, totalling 24 fits
Best Parameters: {'n_neighbors': 7, 'weights': 'distance'}
Tuned - Val Accuracy: 0.9297, Val F1: 0.9284
Test Accuracy: 0.9284, Test F1: 0.9268
Confusion Matrix:
 [[13497   721]
 [ 1316 12901]]

Classification Report:
               precision    recall  f1-score   support

           0       0.91      0.95      0.93     14218
           1       0.95      0.91      0.93     14217

    accuracy                           0.93     28435
   macro avg       0.93      0.93      0.93     28435
weighted avg       0.93      0.93      0.93     28435
```



6.2 Naive Bayes

Hyperparameter Selection and Justification for Naive Bayes

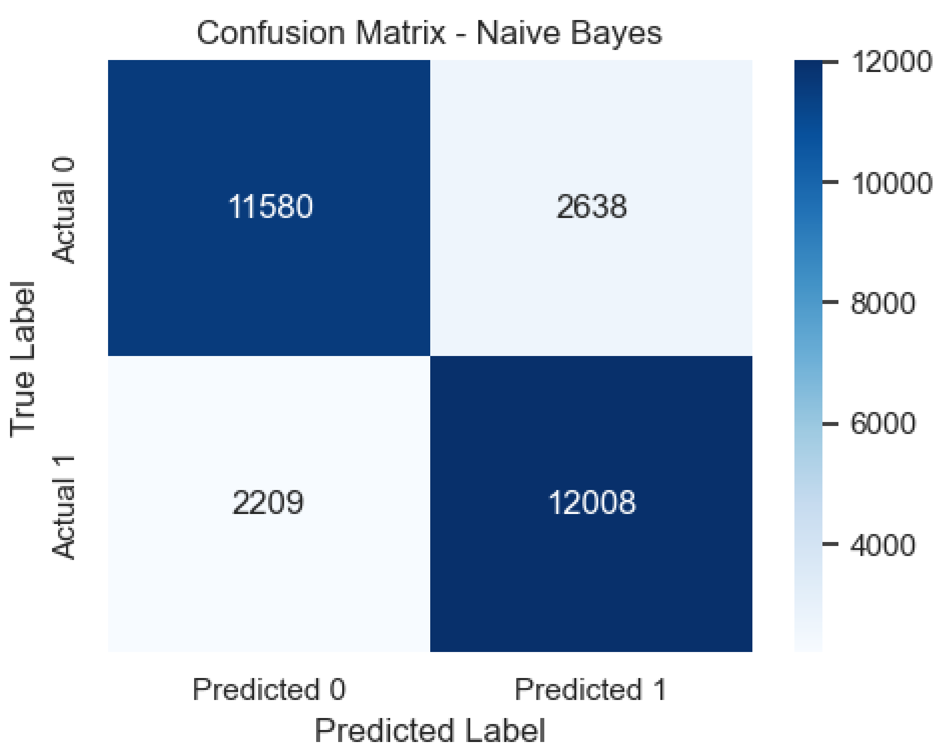
- **Important Hyperparameters**:
  - `var_smoothing` : Adds variance to features to handle zero-variance issues, controlling model stability and preventing overfitting in small datasets.
- **Value Ranges**:

- `var_smoothing`: [1e-9, 1e-8, 1e-7] - Small values (close to 0) for minimal smoothing; range tests sensitivity to variance, balancing bias (high smoothing) vs. variance (low smoothing).

- **Selection Criteria**: Best model chosen based on validation accuracy, ensuring stability between training and validation.

In [ ]:
```
nb_params = {'var_smoothing': [1e-9, 1e-8, 1e-7]} # Define hyperparameter grid for
nb_model = train_evaluate_model(GaussianNB(), nb_params, 'Naive Bayes')
```

```
--- Training Naive Bayes ---
Default Params - Val Accuracy: 0.8264, Val F1: 0.8289
Fitting 3 folds for each of 3 candidates, totalling 9 fits
Best Parameters: {'var_smoothing': 1e-09}
Tuned - Val Accuracy: 0.8264, Val F1: 0.8289
Test Accuracy: 0.8295, Test F1: 0.8321
Confusion Matrix:
 [[11580  2638]
 [ 2209 12008]]

Classification Report:
               precision    recall  f1-score   support

           0       0.84      0.81      0.83     14218
           1       0.82      0.84      0.83     14217

    accuracy                           0.83     28435
   macro avg       0.83      0.83      0.83     28435
weighted avg       0.83      0.83      0.83     28435
```



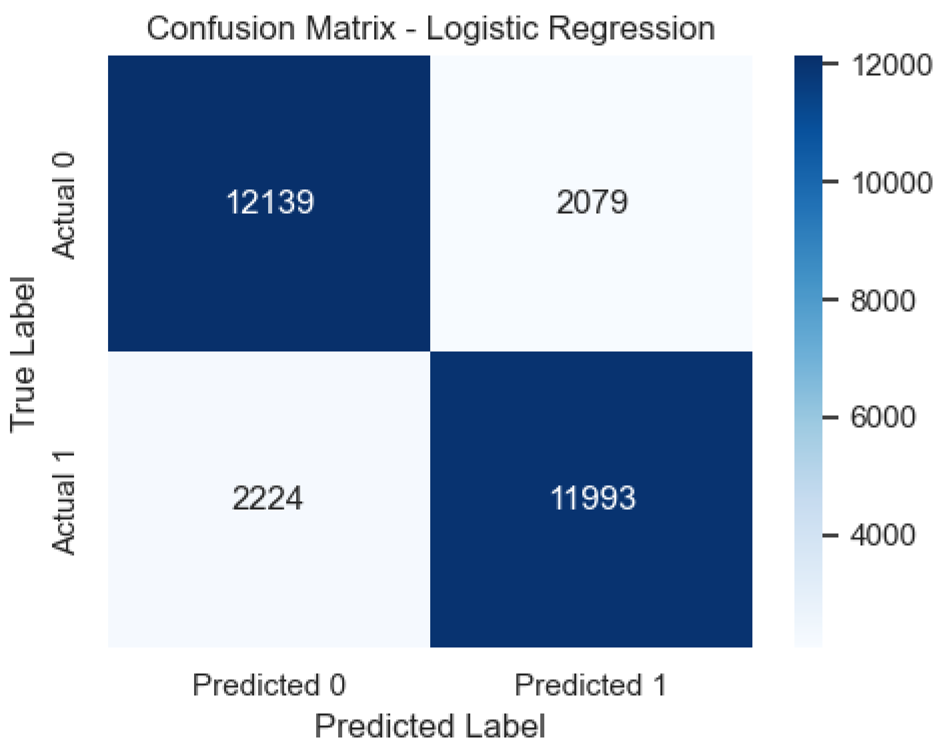
6.3 Logistic Regression

Hyperparameter Selection and Justification for Logistic Regression

- **Important Hyperparameters**:
  - `C` : Inverse regularization strength; lower C increases regularization (reduces overfitting), higher C allows more complexity (risks overfitting).
  - `solver` : Optimization algorithm; affects convergence and handling of regularization.
- **Value Ranges**:
  - `C` : [0.1, 1, 10] - Tests weak (0.1) to strong regularization (10), controlling bias-variance trade-off.
  - `solver` : ['liblinear', 'lbfgs'] - 'liblinear' for small datasets, 'lbfgs' for larger; chosen for compatibility and performance.
- **Selection Criteria**: Best model chosen based on validation accuracy, prioritizing reduction of overfitting via stable val/test scores.

In [ ]:
```
lr_params = {'C': [0.1, 1, 10], 'solver': ['liblinear', 'lbfgs']}  # Define hyperpa
lr_model = train_evaluate_model(LogisticRegression(max_iter=1000), lr_params, 'Logi
```

```
--- Training Logistic Regression ---
Default Params - Val Accuracy: 0.8440, Val F1: 0.8432
Fitting 3 folds for each of 6 candidates, totalling 18 fits
Best Parameters: {'C': 10, 'solver': 'lbfgs'}
Tuned - Val Accuracy: 0.8440, Val F1: 0.8432
Test Accuracy: 0.8487, Test F1: 0.8479
Confusion Matrix:
 [[12139  2079]
 [ 2224 11993]]

Classification Report:
               precision    recall  f1-score   support

           0       0.85      0.85      0.85     14218
           1       0.85      0.84      0.85     14217

    accuracy                           0.85     28435
   macro avg       0.85      0.85      0.85     28435
weighted avg       0.85      0.85      0.85     28435
```



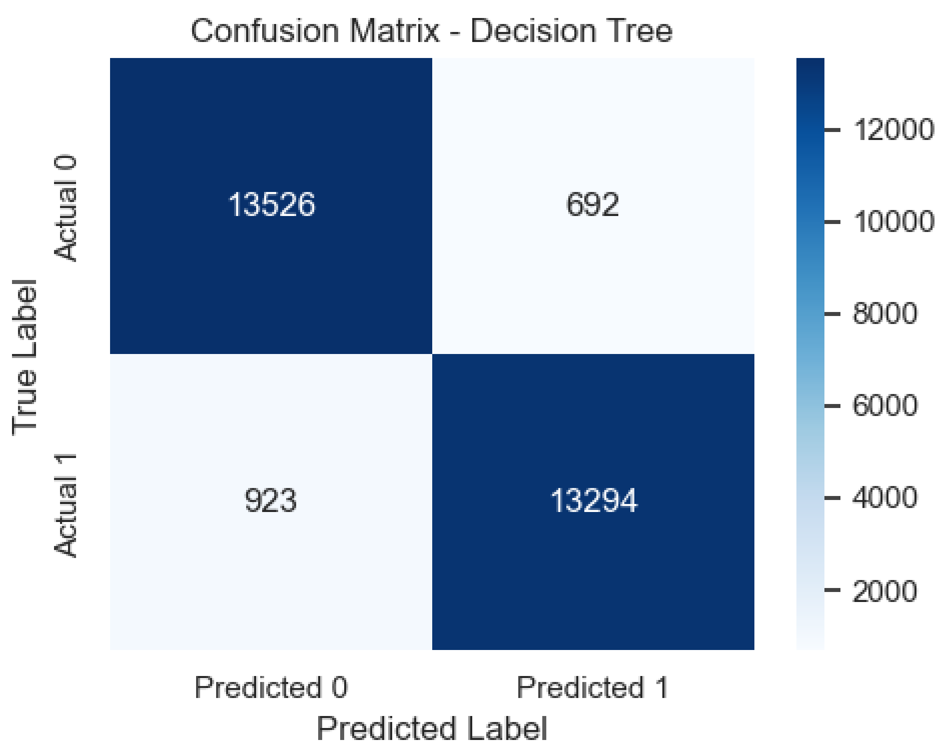
6.4 Decision Tree

Hyperparameter Selection and Justification for Decision Tree

- **Important Hyperparameters**:
  - `max_depth` : Limits tree depth; shallower trees reduce overfitting (higher bias), deeper allow complexity (higher variance).
  - `min_samples_split` : Minimum samples to split; higher values prevent overfitting by requiring more data for splits.
  - `criterion` : Splitting metric ('gini' or 'entropy'); affects how splits are chosen, influencing model fit.
- **Value Ranges**:
  - `max_depth` : [None, 10, 20] - None for unlimited (potential overfitting), 10/20 for controlled depth, balancing complexity.
  - `min_samples_split` : [2, 5, 10] - Tests from minimal splits (2) to conservative (10), controlling overfitting.
  - `criterion` : ['gini', 'entropy'] - Compares impurity measures for best fit.
- **Selection Criteria**: Best model chosen based on validation accuracy, ensuring no overfitting (val/test stability).

In [ ]:
```
dt_params = {'max_depth': [None, 10, 20], 'min_samples_split': [2, 5, 10], 'criteri
dt_model = train_evaluate_model(DecisionTreeClassifier(random_state=42), dt_params,
```

```
--- Training Decision Tree ---
Default Params - Val Accuracy: 0.9434, Val F1: 0.9435
Fitting 3 folds for each of 18 candidates, totalling 54 fits
Best Parameters: {'criterion': 'entropy', 'max_depth': 20, 'min_samples_split': 10}
Tuned - Val Accuracy: 0.9481, Val F1: 0.9478
Test Accuracy: 0.9432, Test F1: 0.9427
Confusion Matrix:
 [[13526   692]
 [  923 13294]]

Classification Report:
               precision    recall  f1-score   support

           0       0.94      0.95      0.94     14218
           1       0.95      0.94      0.94     14217

    accuracy                           0.94     28435
   macro avg       0.94      0.94      0.94     28435
weighted avg       0.94      0.94      0.94     28435
```



6.5 Random Forest

Hyperparameter Selection and Justification for Random Forest

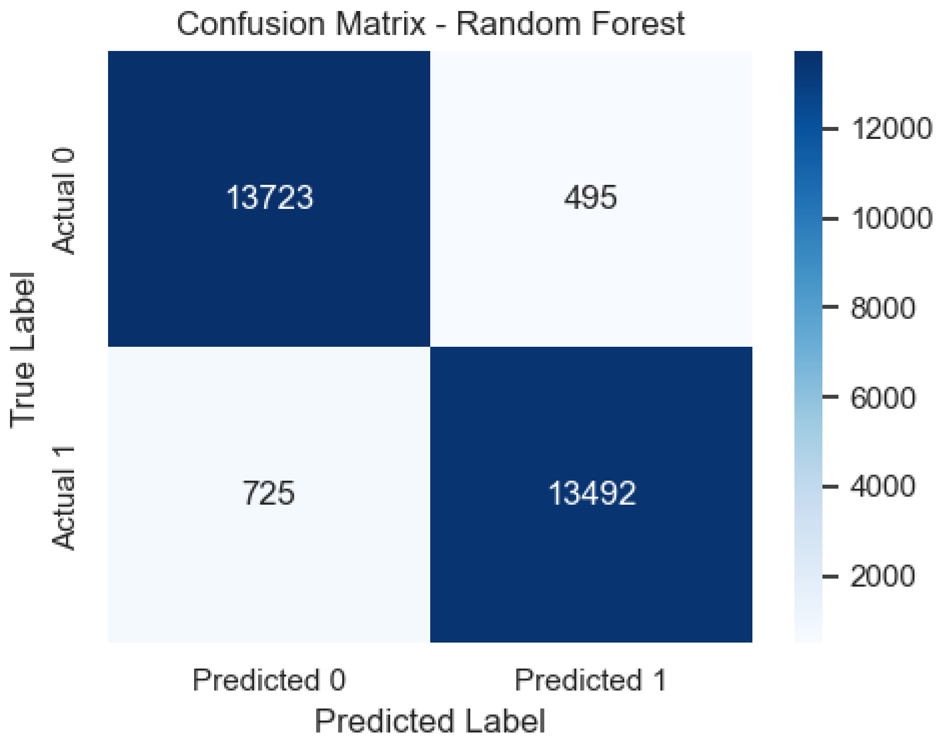
- **Important Hyperparameters**:
  - `n_estimators` : Number of trees; more trees reduce variance (less overfitting), but increase computation.
  - `max_depth` : Limits depth per tree; controls complexity, similar to Decision Tree.
  - `min_samples_split` : Minimum samples to split; higher values reduce overfitting.

- **Value Ranges**:
  - `n_estimators` : [50, 100] - Balances performance (100) vs. speed (50), controlling variance.
  - `max_depth` : [None, 10, 20] - None for full depth, 10/20 for regularization, managing bias-variance.
  - `min_samples_split` : [2, 5] - Tests minimal (2) vs. conservative (5) splits to prevent overfitting.
- **Selection Criteria**: Best model chosen based on validation accuracy, focusing on stability and overfitting reduction.

In [ ]:

```
rf_params = {'n_estimators': [50, 100], 'max_depth': [None, 10, 20], 'min_samples_s
rf_model = train_evaluate_model(RandomForestClassifier(random_state=42), rf_params,
```

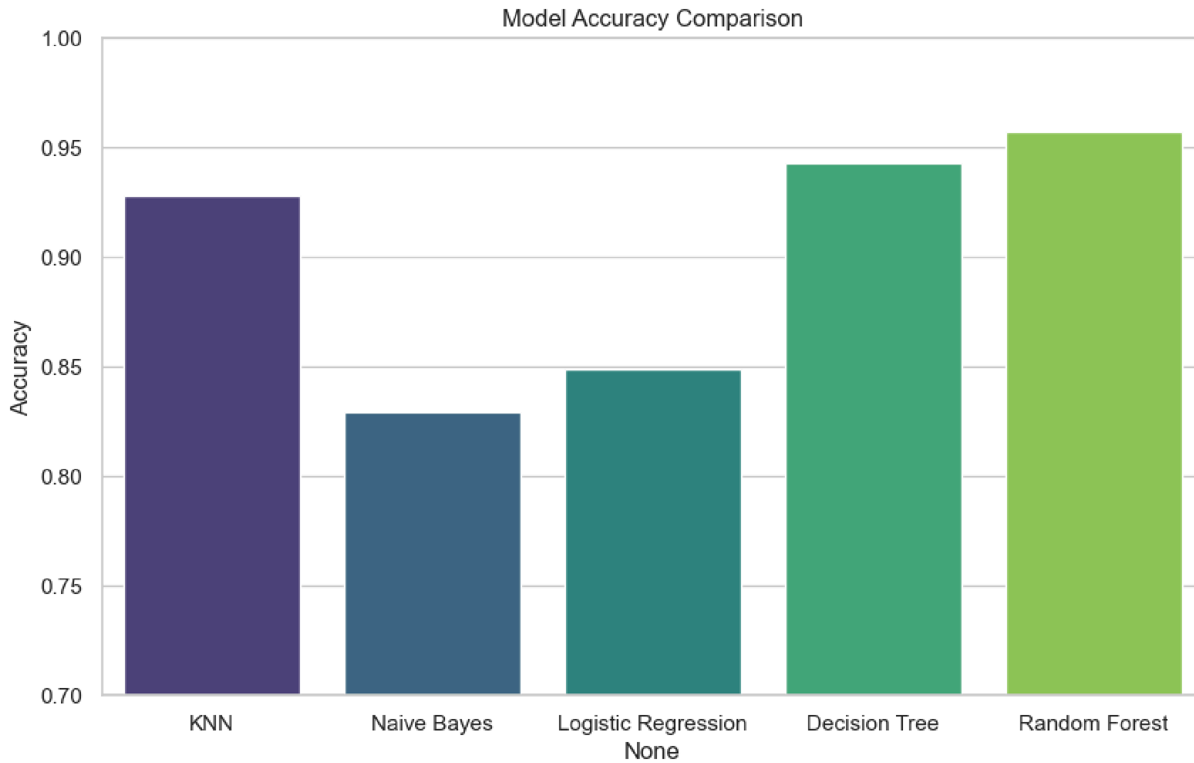
```
--- Training Random Forest ---
Default Params - Val Accuracy: 0.9605, Val F1: 0.9602
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Best Parameters: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}
Tuned - Val Accuracy: 0.9603, Val F1: 0.9600
Test Accuracy: 0.9571, Test F1: 0.9567
Confusion Matrix:
 [[13723   495]
 [  725 13492]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.97      0.96     14218
           1       0.96      0.95      0.96     14217

    accuracy                           0.96     28435
   macro avg       0.96      0.96      0.96     28435
weighted avg       0.96      0.96      0.96     28435
```



8. Model Evaluation & 9. Comparison

In [ ]:
```
# Create Summary DataFrame
results_df = pd.DataFrame(model_results).T  # Transpose for better readability
print("Model Performace Comparison:")
print(results_df)

# Visualization
plt.figure(figsize=(10, 6))
sns.barplot(x=results_df.index, y=results_df['Test Accuracy'], palette='viridis')
plt.title('Model Accuracy Comparison')
plt.ylim(0.7, 1.0)
plt.ylabel('Accuracy')
plt.show()
```

```
Model Performace Comparison:
                     Default Val Acc  Tuned Val Acc  Test Accuracy  \
KNN                         0.928750       0.929699       0.928363
Naive Bayes                 0.826376       0.826376       0.829541
Logistic Regression         0.843960       0.843960       0.848672
Decision Tree               0.943380       0.948127       0.943204
Random Forest               0.960471       0.960260       0.957095

                     Test Precision  Test Recall  Test F1 Score
KNN                        0.947071     0.907435       0.926829
Naive Bayes                0.819883     0.844623       0.832069
Logistic Regression        0.852260     0.843568       0.847891
Decision Tree              0.950522     0.935078       0.942737
Random Forest              0.964610     0.949005       0.956744
```



10. Conclusion

Based on the analysis, we can observe which model performed the best.

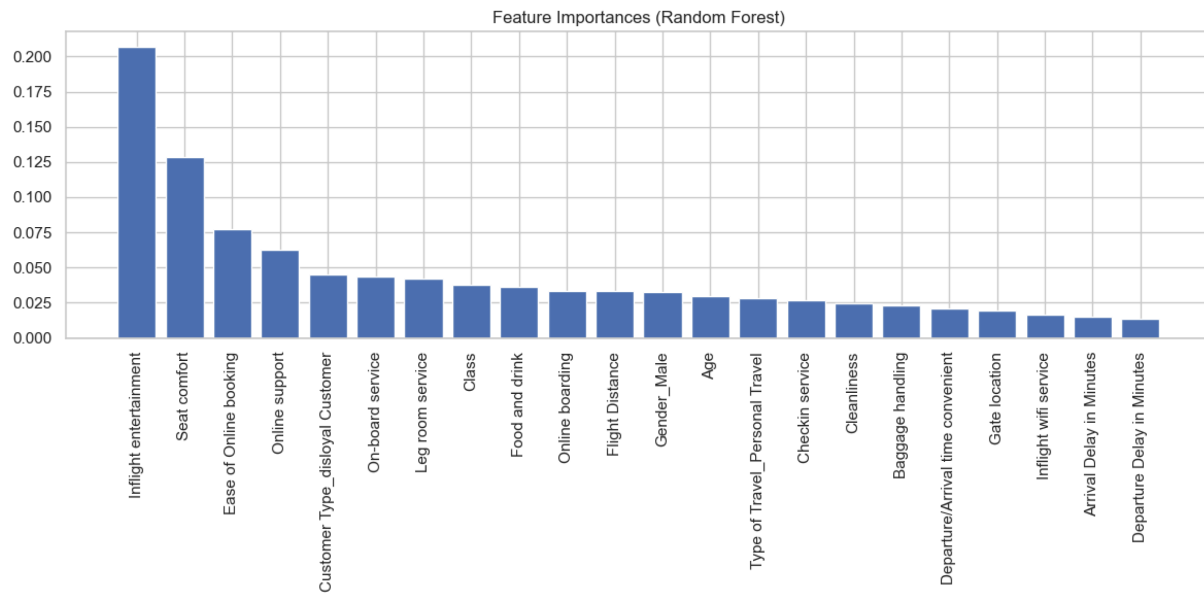
- **Random Forest** typically performs very well on this dataset due to its ability to capture complex non-linear relationships and interactions between features.
- **Logistic Regression** and **Naive Bayes** might struggle if the decision boundary is highly non-linear.

The best model should be selected for deployment based on the Validation Accuracy and F1-Score.

Bonus: Feature Importance (Random Forest)

In [ ]:

```
if 'Random Forest' in model_results:
    importances = rf_model.feature_importances_  # Get feature importances from Ran
    feature_names = X.columns  # Feature names
    indices = np.argsort(importances)[::-1]  # Sort indices by importance descendin

    plt.figure(figsize=(12, 6))
    plt.title("Feature Importances (Random Forest)")
    plt.bar(range(X.shape[1]), importances[indices], align="center")  # Bar plot of
    plt.xticks(range(X.shape[1]), feature_names[indices], rotation=90)  # Feature n
    plt.tight_layout()
    plt.show()
```


Feature Importances (Random Forest)
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
Inflight entertainment
Seat comfort
Ease of Online booking
Online support
Customer Type_disloyal Customer
On-board service
Leg room service
Class
Food and drink
Online boarding
Flight Distance
Gender_Male
Age
Type of Travel_Personal Travel
Checkin service
Cleanliness
Baggage handling
Departure/Arrival time convenient
Gate location
Inflight wifi service
Arrival Delay in Minutes
Departure Delay in Minutes