Model Optimization Process

1. Data Preprocessing

Handling Missing Values:

For categorical columns like 'Payment Method' and 'Preferred Visit Time', missing values are replaced with 'Unknown'.

Feature Selection:

The target variable is 'Satisfaction Score', which is converted into a binary classification based on its median value.

Features are divided into categorical and numerical columns.

Splitting Data:

The dataset is split into training (80%) and test (20%) sets using train_test_split with a random seed of 42 for reproducibility.

2. Preprocessing Pipelines

Categorical Features:

Handled using a pipeline with:

SimpleImputer (most frequent strategy) for missing values.

OneHotEncoder for converting categories into dummy variables, ignoring unknown categories.

Numerical Features:

Processed using a pipeline that applies StandardScaler to normalize the data.

3. Model Pipeline

The preprocessing steps for both categorical and numerical features are combined using a ColumnTransformer.

The final pipeline consists of:

preprocessor: Responsible for data transformation.

classifier: A LogisticRegression model with the following hyperparameters:

random_state=42: Ensures reproducibility.

max_iter=1000: Increased to ensure convergence.

4. Training

The model is trained using the fit method on the training dataset.

5. Evaluation

Predictions are made on the test set.

Performance is evaluated using classification_report, which provides precision, recall, F1-score, and accuracy for each class.

6. Model Persistence

The trained model is saved as logistic_model.pkl using joblib for later use in a production environment.

Initial Flask App Architecture

Project Directory Structure:

```
├── app.py               # Flask application
├── templates/           # HTML templates for web UI
├── static/              # CSS, JavaScript, images, etc.
├── logistic_model.pkl   # Pre-trained model file
└── requirements.txt     # Dependencies
```

app.py: Flask Application

```
from flask import Flask, request, render_template
import joblib
import numpy as np
import pandas as pd

# Load the trained model
model = joblib.load('logistic_model.pkl')

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('index.html')
```

```
@app.route('/predict', methods=['POST'])
def predict():
    if request.method == 'POST':
        # Extract input data from the form
        features = [request.form[key] for key in request.form.keys()]

        # Convert features to DataFrame for compatibility with the pipeline
        input_df = pd.DataFrame([features], columns=request.form.keys())

        # Predict using the loaded model
        prediction = model.predict(input_df)[0]

        # Render the result page
        return render_template('result.html', prediction=prediction)

if __name__ == '__main__':
    app.run(debug=True)
```

Templates (templates/)

index.html: Form to input feature values.

result.html: Displays prediction results.

This architecture allows seamless integration of the model into a Flask app, providing a user-friendly interface for making predictions in real-time. Would you like detailed code for any specific section or suggestions for further optimization?