Telco Customer Churn Prediction Flask App

📁 Required Directory Structure

Create the following directory structure for your Flask application:

```
churn_prediction_app/
├── app.py                    # Main Flask application
├── requirements.txt          # Python dependencies
├── README.md                 # This file
├── templates/                # HTML templates
│   ├── base.html             # Base template with common layout
│   ├── index.html            # Home page template
│   ├── predict.html          # Prediction page template
│   └── model_info.html       # Model information template
├── model/                    # Model files (from training script)
│   ├── best_churn_model.joblib   # Trained ML model
│   ├── model_metadata.json       # Model metadata
│   └── evaluation_metrics.png    # Model evaluation plots
├── uploads/                  # Temporary file uploads (auto-created)
└── customers.csv             # Your customer data file
```

🚀 Setup Instructions

1. Prerequisites

- Python 3.8 or higher
- Trained model files from the training script

2. Install Dependencies

```bash
# Create virtual environment (recommended)
python -m venv churn_env
source churn_env/bin/activate  # On Windows: churn_env\Scripts\activate

# Install required packages
pip install -r requirements.txt
```

3. Prepare Model Files

Ensure you have run the training script (`train.py`) which should create:

- `model/best_churn_model.joblib` - The trained machine learning model
- `model/model_metadata.json` - Model features and performance metrics
- `model/evaluation_metrics.png` - Performance visualization plots

4. Prepare Customer Data

Place your customer CSV file in the project root directory. The system will:

- Automatically remove unnecessary columns
- Handle missing values
- Convert data types appropriately

5. Run the Application

```bash
python app.py
```

The application will start on `http://localhost:5000`

📋 File Explanations

Core Application Files

`app.py` - Main Flask Application

- Handles file uploads and CSV processing
- Provides REST API endpoints for customer data and predictions
- Manages model loading and prediction logic
- Implements automatic column filtering based on training script
- Includes error handling and validation

`requirements.txt` - Python Dependencies

- Lists all required Python packages with specific versions
- Includes Flask, scikit-learn, pandas, and visualization libraries
- Ensures consistent environment setup across deployments

HTML Templates

`templates/base.html` - Base Template

- Common HTML structure and navigation
- Bootstrap CSS framework for responsive design
- Font Awesome icons for better UX
- Flash message handling
- Custom CSS for styling prediction cards and risk levels

`templates/index.html` - Home Page

- Welcome section with usage instructions
- File upload form for customer CSV data
- Status indicators for data and model loading
- Feature requirements and format guidelines

`templates/predict.html` - Prediction Interface

- Customer selection dropdown populated from uploaded data
- Two-column customer data display for easy viewing
- Real-time AJAX calls for data loading and predictions
- Interactive prediction results with risk levels and recommendations
- Responsive design with loading indicators

`templates/model_info.html` - Model Information

- Displays model type, performance metrics, and features
- Shows hyperparameters used during training
- Lists numeric and categorical features
- Technical information about preprocessing steps

🔧 Key Features

Automatic Data Processing

- **Column Filtering**: Automatically removes 20+ unnecessary columns including:
  - Identifiers (Customer ID)
  - Geographic data (Lat/Long, Zip, City, State)
  - Temporal features (Quarter)
  - Target leakage (Churn Reason, Churn Score)
  - Low-value features identified during EDA
- **Data Cleaning**:
  - Converts Total Charges to numeric
  - Imputes missing values with median (numeric) or mode (categorical)
  - Handles data type conversions automatically

User Experience

- **Responsive Design**: Works on desktop, tablet, and mobile devices
- **Real-time Updates**: AJAX-powered interface for smooth interactions
- **Visual Feedback**: Color-coded risk levels (High=Red, Medium=Yellow, Low=Green)
- **Actionable Insights**: Provides specific recommendations based on churn risk

Model Integration

- **Seamless Prediction**: Uses the exact same preprocessing pipeline as training
- **Confidence Scores**: Shows prediction probability and model confidence
- **Feature Importance**: Displays top contributing factors (when available)
- **Error Handling**: Graceful handling of missing models or corrupted data

🎯 Usage Workflow

1. **Upload Data**: Select and upload your customer CSV file
2. **Select Customer**: Choose a customer from the dropdown list
3. **View Details**: Examine customer information in organized columns
4. **Get Prediction**: Click "Predict Churn Risk" for ML analysis
5. **Review Results**: See churn probability, risk level, and recommendations
6. **Take Action**: Follow suggested retention strategies based on risk level

📊 Risk Level Interpretations

- 🟢 **Low Risk (0-30%)**: Stable customers, focus on upselling
- 🟡 **Medium Risk (30-70%)**: Proactive engagement needed
- 🔴 **High Risk (70-100%)**: Immediate retention efforts required

🔒 Security Notes

- File uploads are validated for CSV format only
- Temporary files are stored securely
- No sensitive data is logged or permanently stored
- Model predictions are generated in real-time without data persistence

🐛 Troubleshooting

Model Not Loading:

- Ensure `model/best_churn_model.joblib` exists
- Check that training script completed successfully
- Verify file permissions

CSV Upload Issues:

- Confirm file is in proper CSV format with headers
- Check for special characters in column names
- Ensure file size is reasonable (<100MB recommended)

Prediction Errors:

- Verify customer data contains expected features
- Check for missing critical columns
- Ensure numeric columns contain valid numbers

🚀 Production Deployment

For production deployment, consider:

- Use a production WSGI server (Gunicorn, uWSGI)
- Set up proper environment variables for configuration
- Implement proper logging and monitoring
- Add authentication if handling sensitive data
- Set up database for persistent storage (optional)
- Configure load balancing for high traffic scenarios