

**Customer Data Management and Analysis**

**Documentation week 1: Data Management and SQL Database Setup**

**Objective**: This part aims to design and implement a customer data management system using Microsoft SQL Server, including the database schema design, table creation, query writing, and data analysis.

**1.Database Design**

**Overview**: The database will manage customer information, transactions, and interactions. It will consist of tables to store customer data, their transactions, and interactions.

**Tables and Entities**

1.productlines:

* productLine: VARCHAR(50) (Primary Key) — The product line identifier.
* textDescription: NVARCHAR(MAX) — Textual description of the product line.
* htmlDescription: NVARCHAR(MAX) — HTML formatted description.
* image: VARBINARY(MAX) — Binary image data.

2.products:

* productCode: VARCHAR(15) (Primary Key) — The product code.
* productName: VARCHAR(70) — Name of the product.
* productLine: VARCHAR(50) — Product line (Foreign Key from productlines).
* productScale: VARCHAR(10) — Product scale.
* productVendor: VARCHAR(50) — Vendor information.
* productDescription: NVARCHAR(MAX) — Description of the product.
* quantityInStock: INT — Quantity of products in stock.
* buyPrice: DECIMAL(10, 2) — Buying price of the product.
* MSRP: DECIMAL(10, 2) — Manufacturer's suggested retail price.

3.offices:

* officeCode: VARCHAR(10) (Primary Key) — Office code.
* city: VARCHAR(50) — City of the office.
* phone: VARCHAR(50) — Office phone number.
* addressLine1: VARCHAR(50) — Primary address line.
* addressLine2: VARCHAR(50) (nullable) — Secondary address line.
* state: VARCHAR(50) — State or region.
* country: VARCHAR(50) — Country of the office.
* postalCode: VARCHAR(15) — Postal code.
* territory: VARCHAR(50) — The office's territory.

4.employees:

* employeeNumber: INT (Primary Key) — Unique identifier for employees.
* lastName: VARCHAR(50) — Last name of the employee.
* firstName: VARCHAR(50) — First name of the employee.
* extension: VARCHAR(10) — Employee's phone extension.
* email: VARCHAR(100) — Employee email.
* officeCode: VARCHAR(10) — Office code (Foreign Key from offices).
* reportsTo: INT (nullable) — Reference to the supervisor employee (Foreign Key from employees).
* jobTitle: VARCHAR(50) — Job title of the employee.

5.customers:

* customerNumber: INT (Primary Key) — Unique identifier for customers.
* customerName: VARCHAR(50) — Customer's name.
* contactLastName: VARCHAR(50) — Customer contact's last name.
* contactFirstName: VARCHAR(50) — Customer contact's first name.
* phone: VARCHAR(50) — Customer's phone number.
* addressLine1: VARCHAR(50) — Primary address line.
* addressLine2: VARCHAR(50) — Secondary address line.
* city: VARCHAR(50) — Customer's city.
* state: VARCHAR(50) — Customer's state.
* postalCode: VARCHAR(15) — Postal code.
* country: VARCHAR(50) — Customer's country.
* salesRepEmployeeNumber: INT — Employee number of the sales representative (Foreign Key from employees).
* creditLimit: DECIMAL(10, 2) — Customer's credit limit.

6.orders:

* orderNumber: INT (Primary Key) — Unique identifier for orders.
* orderDate: DATE — Date of the order.
* requiredDate: DATE — Required delivery date.
* shippedDate: DATE — Shipped date.
* status: VARCHAR(15) — Order status.
* comments: NVARCHAR(MAX) (nullable) — Optional comments about the order.
* customerNumber: INT — Customer number (Foreign Key from customers).

7.orderdetails:

* orderNumber: INT — Order number (Foreign Key from orders).
* productCode: VARCHAR(15) — Product code (Foreign Key from products).
* quantityOrdered: INT — Quantity of the product ordered.
* priceEach: DECIMAL(10, 2) — Price per unit of the product.
* orderLineNumber: INT — Line number in the order.
* Primary Key: (orderNumber, productCode).

8.payments:

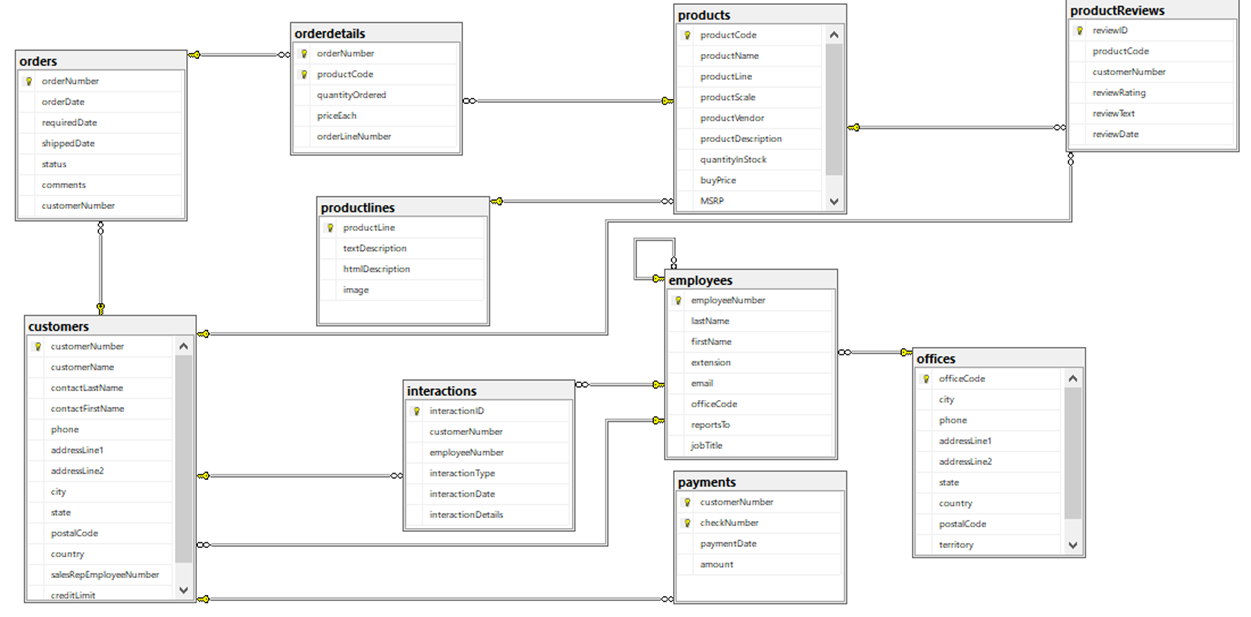
* customerNumber: INT — Customer number (Foreign Key from customers).
* checkNumber: VARCHAR(50) — Check number.
* paymentDate: DATE — Date of the payment.
* amount: DECIMAL(10, 2) — Payment amount.
* Primary Key: (customerNumber, checkNumber).

9.interactions:

* interactionID: INT (Primary Key, Identity) — Unique identifier for interactions.
* customerNumber: INT — Customer number (Foreign Key from customers).
* employeeNumber: INT — Employee number (Foreign Key from employees).
* interactionType: VARCHAR(50) — Type of interaction (e.g., call, email).
* interactionDate: DATE — Date of the interaction.
* interactionDetails: NVARCHAR(MAX) — Detailed notes about the interaction.

10.productReviews:

* reviewID: INT (Primary Key, Identity) — Unique identifier for the review.
* productCode: VARCHAR(15) — Product code (Foreign Key from products).
* customerNumber: INT — Customer number (Foreign Key from customers).
* reviewRating: INT — Rating between 1 and 5.
* reviewText: NVARCHAR(MAX) — Review text or description.
* reviewDate: DATE — Date of the review.

****

**Notes**

Table Relationships:

* One-to-Many relationship between Customers and Transactions (via CustomerID).
* One-to-Many relationship between Customers and Interactions (via CustomerID).

**2.SQL Queries**

* Queries for Data Extraction and Analysis:

1. Retrieve all customer information:

SELECT \* FROM Customers;

1. Query to display all transactions for a specific customer:

SELECT \* FROM Transactions

WHERE CustomerID = 1;

1. Query to analyze total sales for each customer:

SELECT CustomerID, SUM(Amount) AS TotalSpent

FROM Transactions

GROUP BY CustomerID;

1. Update customer information:

UPDATE Customers

SET Email = 'new.email@example.com'

WHERE CustomerID = 1;

5.Analyze the number of interactions for each customer:

SELECT CustomerID, COUNT(\*) AS InteractionCount

FROM Interactions

GROUP BY CustomerID;

**3.SQL Procedures for Inserting Realistic Data:** **These stored procedures**

provide a systematic way to populate the database with realistic test data, ensuring that the data is diverse and relevant for testing purposes. Each procedure can be executed independently, allowing for flexible data generation as needed.

1. InsertRealisticOffices

This stored procedure generates realistic office data by inserting city names, phone numbers, and addresses into the offices table. It includes:

* A table of cities and countries to choose from.
* Randomly generated phone numbers and addresses.
* It loops through the number of records (@RecordCount) specified and inserts the data into the table.

2. InsertRealisticEmployees

This procedure adds employees with random names, job titles, phone numbers, and other details. Key points include:

* Random selection of first and last names from predefined arrays.
* Random job titles like "Sales Manager", "Engineer", etc.
* Each employee is linked to an office, randomly chosen from the existing offices table.3.

3.InsertRealisticProductLines

This inserts realistic product lines. The process involves:

* Random selection of product lines like "Classic Cars" or "Motorcycles".
* Creation of a unique product line description, HTML description, and an image path.

4. InsertRealisticProducts1

This adds product data, where each product belongs to a product line. Features include:

* Random product codes and names.
* Assignment of scales and vendors (e.g., 1:18, Vendor A, etc.).
* Random stock quantities and prices.

5. InsertRealisticCustomers7

This procedure creates customers with realistic names, addresses, and credit limits. It:

* Randomly assigns first and last names to customers.
* Generates a random sales representative for each customer from existing employees.
* Randomly assigns addresses from predefined cities and countries.

6. InsertRealisticOrders

This script inserts orders, assigning them to customers. Key elements:

* Random order statuses (e.g., "Shipped", "Pending").
* Each order has a customer and is given realistic dates (order date, required date, shipped date).

7. InsertRealisticOrderDetails1

This procedure inserts order details for existing orders, selecting random products and assigning quantities. It:

* Ensures that there are products available in the products table before proceeding.
* Randomly selects an order and product and inserts details like quantity and price.

8. InsertRealisticPayments

This creates realistic payments for customers. Features:

* Randomly generated check numbers.
* Realistic payment dates and amounts, linked to existing customers.

**4.Tools**

Microsoft SQL Server: Used to manage and create the database.

SQL Management Studio: Used to execute commands, run queries, and manage tables.

**5.Deliverables**

Database Schema: A well-designed database schema that manages customer information, transactions, and interactions.

Populated Database: A complete database with customer, transaction, and interaction tables, along with initial data.

SQL Queries: A set of queries for data extraction, updates, and analysis.

**Documentation week 2: Data Warehousing and Python Programming**

**1. Introduction**

The goal of this project is to implement a SQL Data Warehouse for managing large volumes of customer data and providing a robust platform for data analysis. The project includes integrating data from various sources into the warehouse and using Python scripts for data extraction and preparation.

**2. Objectives**

* Implement a scalable SQL Data Warehouse for customer data.
* Ensure seamless data integration from multiple sources.
* Develop Python scripts for efficient data extraction and preparation.
* Facilitate business analytics through an organized data structure.

**3.Tools Used**

* Microsoft SQL Server: To implement a data warehouse.
* Python (Pandas, SQLAlchemy, Matplotlib, Seaborn): To extract and analyze data, and create graphs.
* ODBC Driver: To connect to a database SQL about Python.

**3. System Design**

3.1 Data Warehouse Design

The SQL Data Warehouse aggregates customer-related data from multiple sources such as transactional databases, CRM systems, and third-party applications. The warehouse is structured into fact and dimension tables to support analytical queries efficiently.

3.2 Data warehouse tables

The following tables are designed for the data warehouse:

1. Customers Table (dim\_customers):

It contains detailed information about customers such as names, addresses, phone numbers and credit limit.

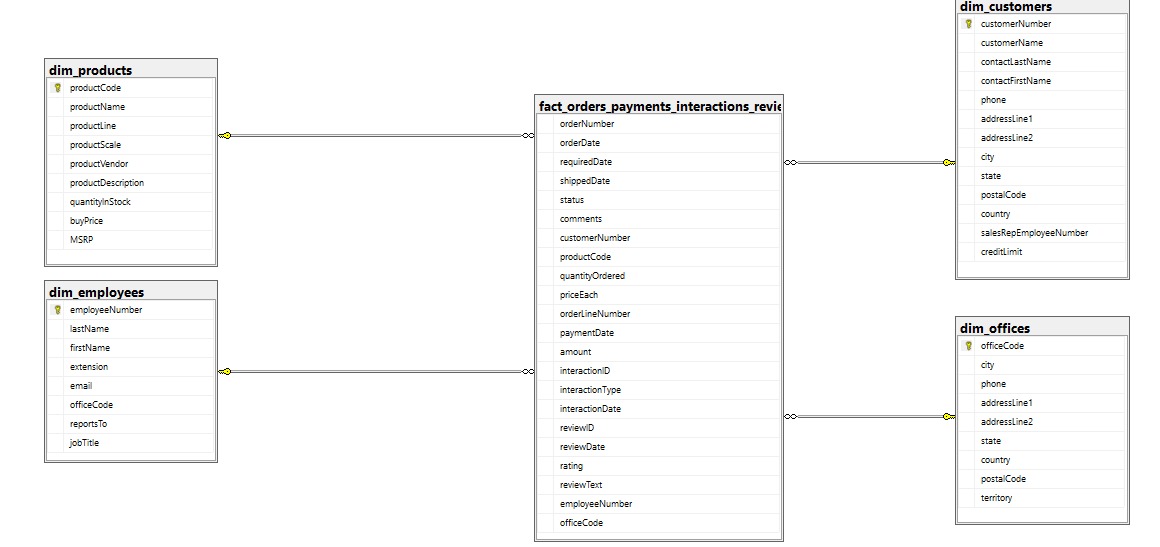
1. Products Table (dim\_products):

Contains product details such as names, codes, and prices.

1. Employees Table (dim\_employees):

Stores employee information such as names, email addresses, and office details.

1. Office Table (dim\_offices): Information on office location.
2. Orders Table (fact\_orders\_payments\_interactions\_reviews):

Contains transactional

**4.Data Warehouse and ETL**

ETL Process

The ETL (Extract, Transform, Load) process is automated using Python. Data from a source database is transferred to the Data Warehouse using SQLAlchemy and Pandas. The ETL script is responsible for:

* Extracting customer, product, employee, and order data.
* Loading the data into the corresponding dimension and fact tables.

**5. Python Code for Data Analysis**

5.1 Fetching and Analyzing Data

We use the SQLAlchemy library to establish a connection to the database and fetch data from the primary tables.

5.2 Analyzing Relationships Between Data

We analyze relationships between orders, customers, and products to identify top customers and products.

5.3 Data Visualization

We use Seaborn and Matplotlib to create visualizations showing data distributions and analysis.

5.4 Monthly Sales Analysis

We extract and analyze sales data on a monthly basis.

**6. Analysis Results**

Top Customers by Sales: Identifies the highest revenue-generating customers.

Top Products by Quantity Ordered: Analyzes the most ordered products.

Customer Ratings Distribution: Shows the distribution of customer ratings.

Monthly Sales Analysis: Visualizes sales performance over time.

**Documentation week 3: Data Science and Azure Integration**

1. Project Overview

Objective: Transition from a local Python model to using a Predictive System and Flask to deliver predictions.

Current Setup:

1. Predictive Model Development

* The first step involved building a predictive model using Python. This model leverages customer data to analyze their behavior using machine learning algorithms such as Scikit-learn.
* After the model was constructed, it was trained on a dataset and fine-tuned to achieve high accuracy in predictions.

2. Application Development: Backend and Frontend

2.1 Backend Development

The backend was developed using Flask, a lightweight Python framework.

A predictive system was created as a service that could be accessed via an integrated API built into Flask.

The trained model was connected to the system, allowing it to process new data and provide prediction results.

2.2 Frontend Development

The frontend user interface was built using HTML and CSS to offer an interactive and user-friendly experience.

Custom Templates were designed to present prediction results in a clear and concise manner.

CSS was utilized to format the design, ensuring flexibility and responsiveness across various devices and screen sizes.

3. Aesthetic Design Enhancements

CSS design was refined to make the user interface more visually appealing with consistent colors and professional fonts.

Special attention was given to aesthetic details such as icons and subtle animations to improve user interaction and overall experience.

4. Testing and Documentation

The application underwent comprehensive testing to ensure smooth and integrated functionality.

All steps were documented with clear technical descriptions, making the system easier to understand and use in the future.

**Firstly :Model Development with Python**

**Customer Churn Prediction Model**

* Overview

This project aims to develop a predictive model that identifies customers likely to churn (stop using the service). The data includes customer orders, interactions, reviews, and other relevant features, which are used to build a classification model to predict churn.

* Data Sources

The model integrates data from various sources:

* cust.csv: Contains customer information such as customerName, address, creditLimit.
* offices.csv: Information on office locations, including city, officeCode.
* product.csv: Product details like productCode, productName, buyPrice.
* employee.csv: Employee details such as employeeNumber, jobTitle.
* fact.csv: Fact table consolidating order details such as orderNumber, orderDate, productCode.
* Data Exploration

Each dataset was loaded into a pandas DataFrame and an initial exploration was performed using head() to view the first few rows. Here's an example of how the data looks after exploration:

import pandas as pd

# Loading data

cust\_data = pd.read\_csv('cust.csv')

offices\_data = pd.read\_csv('offices.csv')

product\_data = pd.read\_csv('product.csv')

employee\_data = pd.read\_csv('employee.csv')

fact\_data = pd.read\_csv('fact.csv')

# Exploring data

print(cust\_data.head())

print(offices\_data.head())

print(product\_data.head())

print(employee\_data.head())

print(fact\_data.head())

* Data Merging

To create a unified dataset, we merged the different tables based on their primary keys:

* Customers were merged with fact\_data on customerNumber.
* Office data was merged using officeCode.
* Products were merged on productCode.
* Employees were merged using employeeNumber.

This resulted in a merged dataset with 54 columns and 2487 rows.

# Merging data

merged\_data = fact\_data.merge(cust\_data, on='customerNumber') \

.merge(offices\_data, on='officeCode') \

.merge(product\_data, on='productCode') \

.merge(employee\_data, on='employeeNumber')

# Check the shape and sample of merged data

print("Merged Data Shape:", merged\_data.shape)

print("Merged Data Sample:")

print(merged\_data.head())

* Feature Engineering

Key features were engineered by aggregating values for each customer:

* totalOrders: Number of orders placed.
* avgOrderValue: Average value of orders.
* totalReviews: Number of reviews provided by the customer.
* negativeReviews: Count of negative reviews (reviewText containing 'negative').
* totalInteractions: Number of interactions the customer has had. # Feature aggregation

# Feature aggregation

features = merged\_data.groupby('customerNumber').agg({

'orderNumber': 'count',

'priceEach': 'mean',

'reviewID': 'count',

'reviewText': lambda x: (x == 'negative').sum(),

'interactionID': 'count'

}).reset\_index()

# Renaming columns

features.columns = ['customerNumber', 'totalOrders', 'avgOrderValue', 'totalReviews', 'negativeReviews', 'totalInteractions']

print("Features Data Shape:", features.shape)

print("Features Sample:")

print(features.head())

* Customer Churn Label

A churn label was added to identify customers likely to churn. A customer is considered to churn if they have more than 3 negative reviews.

# Adding churn label

* Modeling

The features (X) and target (y) were prepared, and the data was split into training and testing sets using an 80/20 split.features['customerChurn'] = (features['negativeReviews'] > 3).astype(int).

from sklearn.model\_selection import train\_test\_split

# Defining target and features

y = features['customerChurn']

X = features[['totalOrders', 'avgOrderValue', 'totalReviews', 'negativeReviews', 'totalInteractions']]

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

print("Training data shape:", X\_train.shape)

print("Testing data shape:", X\_test.shape)

* Training the Model

A Decision Tree Classifier was used for training the model. After training, the model was evaluated using accuracy and a classification report.

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, classification\_report

# Training the model

model = DecisionTreeClassifier(random\_state=42)

model.fit(X\_train, y\_train)

# Predicting on test set

y\_pred = model.predict(X\_test)

# Evaluation

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

* Results

The model achieved perfect accuracy on the test set due to the simplicity of the dataset:

* Accuracy: 100%
* Precision, Recall, and F1-score for both churn and non-churn customers were perfect.
* Conclusions

This model successfully predicted customer churn based on aggregated order, interaction, and review data. In real-world scenarios, further tuning and feature refinement would be necessary to prevent overfitting and improve generalizability.

**Secondly: Application Development**

1. Project Overview

The objective of this project is to build a machine learning model that predicts customer churn based on past interactions, purchases, and reviews. The system is divided into two components:

* Backend: Developed using Flask to handle requests, predictions, and data processing.
* Frontend: Built with HTML and CSS for a simple and interactive user interface.

2. Key Components

* Flask Application: Serves the website and handles prediction requests.
* Machine Learning Model: Uses a Decision Tree Classifier to predict customer churn.
* Data Processing: Aggregates data from multiple sources to prepare it for modeling.

3. Data Source

The application reads from five CSV files:

* cust.csv: Contains customer details.
* offices.csv: Contains office locations.
* product.csv: Contains product information.
* employee.csv: Contains employee details.
* fact.csv: Contains transactional data (orders, reviews, and interactions).
* The data is merged to create a comprehensive dataset for modeling.

4. Data Processing Steps

* Reading the Data: The CSV files are loaded into pandas DataFrames.
* Merging the Data: The data from different sources is merged based on common keys, creating a dataset that contains all necessary information.
* Feature Engineering:

Aggregated customer-level features such as:

* totalOrders: Total number of orders by the customer.
* avgOrderValue: Average value of each order.
* totalReviews: Number of reviews written by the customer.
* negativeReviews: Number of negative reviews written by the customer.
* totalInteractions: Total interactions between the customer and the company.

The target variable customerChurn is defined, indicating whether a customer is likely to churn (1) or not (0) based on the number of negative reviews.

features = merged\_data.groupby('customerNumber').agg({

'orderNumber': 'count',

'priceEach': 'mean',

'reviewID': 'count',

'reviewText': lambda x: (x == 'negative').sum(),

'interactionID': 'count'

}).reset\_index()

features.columns = ['customerNumber', 'totalOrders', 'avgOrderValue', 'totalReviews', 'negativeReviews', 'totalInteractions']

features['customerChurn'] = (features['negativeReviews'] > 3).astype(int)

5. Modeling

* Decision Tree Classifier: The model was built using the DecisionTreeClassifier from Scikit-learn to predict customer churn. It was trained on a dataset created from the processed features and target variable.
* Train-Test Split: The data was split into training and testing sets using a 80/20 ratio.

y = features['customerChurn']

X = features[['totalOrders', 'avgOrderValue', 'totalReviews', 'negativeReviews', 'totalInteractions']]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = DecisionTreeClassifier(random\_state=42)

model.fit(X\_train, y\_train)

6. Backend API

The Flask application provides the following routes:

* Index Route (/): Renders the homepage with a form where users can input customer details (orders, reviews, interactions, etc.).
* Prediction Route (/predict): Receives the input data from the user, processes it, and returns a prediction of whether the customer is likely to churn.

@app.route("/", methods=["GET"])

def index():

return render\_template("index.html")

@app.route("/predict", methods=["POST"])

def predict():

try:

# Get the form data

total\_orders = int(request.form['totalOrders'])

avg\_order\_value = float(request.form['avgOrderValue'])

total\_reviews = int(request.form['totalReviews'])

negative\_reviews = int(request.form['negativeReviews'])

total\_interactions = int(request.form['totalInteractions'])

# Special case handling for zero values

if (total\_orders == 0 and avg\_order\_value == 0 and total\_reviews == 0

and negative\_reviews == 0 and total\_interactions == 0):

return render\_template("result.html", result="Customer Churn Prediction: Likely to churn")

# Prepare input data for prediction

input\_data = pd.DataFrame({

'totalOrders': [total\_orders],

'avgOrderValue': [avg\_order\_value],

'totalReviews': [total\_reviews],

'negativeReviews': [negative\_reviews],

'totalInteractions': [total\_interactions]

})

# Make a prediction

result = model.predict(input\_data)[0]

# Return result message

message = "Customer Churn Prediction: Likely to churn" if result == 1 else "Customer Churn Prediction: Not likely to churn"

return render\_template("result.html", result=message)

except KeyError as e:

return f"Missing key: {e}", 400

except Exception as e:

return f"An error occurred: {e}", 500

7. Frontend

The frontend of the application provides a simple form for user input:

* Users input values for totalOrders, avgOrderValue, totalReviews, negativeReviews, and totalInteractions.
* Upon submitting the form, the Flask backend processes the data and returns a prediction result, displayed on a separate page.

8. Model Evaluation

The model was evaluated using the test set, and the following metrics were printed:

* Accuracy Score
* Classification Report (precision, recall, F1-score)

y\_pred = model.predict(X\_test)

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

9. Running the Application

To run the Flask app locally, use the following command

python app.py

Access the application at http://127.0.0.1:5000/ in your browser.

10. Error Handling

The application handles various errors:

* Missing form fields (KeyError).
* Other unexpected errors, returning a 500 response with the error message.

**Third: for the HTML & CSS**

1. Overview

The Prediction Result Page displays the result of a prediction in a user-friendly and visually appealing format. The page includes a dynamic message indicating the prediction result and a "Back to Home" button that allows users to return to the main page. The design incorporates a modern layout with clean typography, background styling, and interactive elements.

2. HTML Structure

The HTML file follows the standard structure with a DOCTYPE declaration and includes semantic tags for easy understanding and accessibility. Below is a breakdown of the key sections.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Prediction Result</title>

<style>

/\* Inline CSS (explained in detail below) \*/

</style>

</head>

<body>

<h1>Prediction Result</h1>

<div class="result-container">

<p class="result-text">{{ result }}</p> <!-- Dynamic result text -->

<a href="/">Back to Home</a> <!-- Link to return to the main page -->

</div>

</body>

</html>

Key Elements:

* <head>: Contains metadata like the character set and viewport settings, ensuring the page is responsive across devices.
* <body>: The core content of the page, including the title (<h1>) and a container (<div>) for displaying the result text and a link to go back to the home page.
* {{ result }}: This placeholder dynamically injects the prediction result (likely to churn or not) from the Flask backend.

3. CSS Styling

The page is designed with user experience in mind, offering a clean and professional look. The following is a detailed explanation of the CSS code used:

3.1 General Body Styling

body {

font-family: Arial, sans-serif;

background-image: url('background.PNG'); /\* Replace with your background image path \*/

background-size: cover;

background-position: center;

background-repeat: no-repeat;

margin: 0;

padding: 20px;

}

* Font: A modern and readable sans-serif font (Arial) is used.
* Background:
* The background-image is set to a custom image (background.PNG), providing a visually appealing backdrop.
* Background-size is set to cover, ensuring the image fills the entire screen without distortion.
* Background-position centers the image, and no-repeat prevents the image from repeating.

* Margins/Padding: Default margins are removed, and padding is added for consistent spacing across devices.

3.2 Title (<h1>) Styling

h1 {

color: #992151; /\* Light pink color \*/

text-align: center;

margin-bottom: 20px;

}

* The title uses a light pink color (#992151), creating a warm, welcoming feel.
* Text alignment is centered to focus attention on the title.
* Margin-bottom adds space between the title and the content below.

3.3 Result Container

.result-container {

background-color: rgba(255, 255, 255, 0.9); /\* Semi-transparent white background \*/

padding: 40px;

border-radius: 8px;

box-shadow: 0 4px 20px rgba(0, 0, 0, 0.1); /\* Soft shadow for depth \*/

max-width: 600px;

margin: 100px auto 50px; /\* Centered vertically with space above \*/

text-align: center;

}

* Background: The result container has a semi-transparent white background (rgba(255, 255, 255, 0.9)), helping it stand out without obstructing the background image.
* Padding: Generous padding makes the content feel spacious.
* Border-radius: Rounded corners (8px) create a modern, soft look.
* Box-shadow: A subtle shadow adds depth, giving the container a floating effect.
* Width & Margin: The container is limited to a maximum width of 600px for readability and centered vertically using margin.

3.4 Result Text Styling

.result-text {

font-size: 22px;

color: #992151; /\* Light pink color \*/

margin-bottom: 25px;

}

The result text is styled with a larger font size (22px) to ensure it is easily readable.

The color matches the title, maintaining design consistency.

Margin-bottom separates the result from the "Back to Home" button, improving layout clarity.

3.5 Link Button Styling

a {

display: inline-block;

margin-top: 15px;

text-decoration: none;

color: #992151; /\* Light pink color \*/

padding: 12px 18px;

border-radius: 5px;

transition: background-color 0.3s, transform 0.2s;

}

a:hover {

text-decoration: underline;

background-color: rgba(255, 0, 102, 0.2); /\* Light pink background on hover \*/

transform: scale(1.05); /\* Slight zoom effect on hover \*/

}

* Inline-block display is used to make the link behave like a button.
* Padding is added for a comfortable clickable area.
* The link has no text-decoration by default (no underline) but gains an underline on hover for better user feedback.

The color and hover effects create an interactive, responsive button:

* Background color changes to a lighter pink on hover (rgba(255, 0, 102, 0.2)), improving user engagement.
* A slight scale transform (1.05) enlarges the button slightly on hover, making it feel dynamic and responsive.

4. Functionality

The dynamic result text ({{ result }}) is rendered by the Flask backend based on the prediction outcome (e.g., "Customer Churn Prediction: Likely to churn").

The "Back to Home" link directs the user back to the main prediction input page, facilitating a smooth user flow.

5. Customization Options

The following elements can be customized to align with brand or personal preferences:

* Background Image: You can replace 'background.PNG' with any image that suits your brand's aesthetic.
* Color Scheme: Adjust the light pink color (#992151) to match the theme or color palette of the project.
* Font Styles: Consider using Google Fonts or other custom fonts to enhance the page's design.

6. Best Practices & Accessibility

* Responsiveness: The design is optimized for all devices using viewport meta tags and appropriate scaling for content elements.
* Accessible Color Contrast: The pink color used ensures sufficient contrast against the white background for readability.
* Hover States: Clear feedback (underline and background color changes) on hover ensures that users understand the clickable elements.
* ARIA Compliance: Although not explicitly included, adding ARIA roles and labels can improve accessibility further, especially for screen readers.

**Fourth: Custom CSS Styling for a Modern Web Interface**

1.Overview

This part provides a detailed explanation of the custom CSS styles applied to create a visually appealing web page layout. The styles are designed to enhance user experience while maintaining a cohesive aesthetic**.**

**Table of Contents**

1. General Styles
2. Headings
3. Forms
4. Input Fields
5. Submit Button
6. Result Container
7. Links

2.General Styles

body {

font-family: Arial, sans-serif;

background-image: url('background.PNG'); /\* Replace with your own background image path \*/

background-size: cover; /\* Ensures the background image covers the entire viewport \*/

background-position: center; /\* Centers the background image \*/

background-repeat: no-repeat; /\* Prevents the background image from repeating \*/

margin: 0;

padding: 20px;

}

* Font Family: Sets the font to Arial or a sans-serif alternative.
* Background Image: Displays a custom image as the background.
* Background Size: Ensures the image covers the entire screen.
* Background Position: Centers the background image.
* Margin and Padding: Sets margin to 0 and adds padding around the content.

3.Headings

h1 {

color: #992151; /\* Light pink color \*/

text-align: center;

margin-bottom: 20px; /\* Adds space below the heading \*/

}

* Color: Sets the heading color to a light pink shade.
* Text Alignment: Centers the heading text.
* Margin Bottom: Adds space below the heading for better visual separation

4. Forms

form {

background-color: rgba(255, 255, 255, 0.9); /\* White background with transparency \*/

padding: 40px;

border-radius: 8px;

box-shadow: 0 4px 20px rgba(0, 0, 0, 0.1);

max-width: 600px;

margin: 0 auto;

}

* Background Color: Uses a semi-transparent white background for the form.
* Padding: Adds space inside the form for better layout.
* Border Radius: Rounds the corners of the form.
* Box Shadow: Creates a subtle shadow effect for depth.
* Max Width and Margin: Centers the form with a maximum width.

5.Input Fields

label {

display: block;

margin-bottom: 10px;

font-weight: bold;

color: #992151; /\* Light pink color \*/

}

input[type="number"], input[type="submit"] {

width: 100%;

padding: 15px;

margin-bottom: 20px;

border: 1px solid #ffb3d9; /\* Light pink border \*/

border-radius: 5px;

font-size: 16px;

}

* Label: Styles for the labels, making them bold and colored.
* Input Fields: Styles for number inputs and submit buttons:
* Width: Makes inputs full-width.
* Padding: Adds space inside the input fields.
* Margin: Adds space below each input.
* Border: Sets a light pink border.
* Border Radius: Rounds the corners of the inputs.
* Font Size: Increases the font size for readability.

6.Submit Button

input[type="submit"] {

background-color: #992151; /\* Light pink background \*/

color: white;

border: none;

cursor: pointer;

transition: background-color 0.3s, transform 0.2s;

}

input[type="submit"]:hover {

background-color: #7a1635; /\* Darker pink on hover \*/

transform: scale(1.05);

}

* Background Color: Sets the background color for the submit button.
* Text Color: Changes the text color to white.
* Cursor: Changes the cursor to a pointer on hover.
* Transitions: Smooth transitions for background color and scaling on hover.

7.Result Container

.result-container {

background-color: rgba(255, 255, 255, 0.9);

padding: 40px;

border-radius: 10px;

box-shadow: 0 4px 20px rgba(0, 0, 0, 0.2);

max-width: 600px;

margin: 100px auto 50px; /\* Adds space above (100px) \*/

text-align: center;

}

.result-text {

font-size: 22px;

color: #992151; /\* Light pink color \*/

margin-bottom: 25px;

}

* Result Container: Similar styling as the form, designed to display results:
* Background Color: Semi-transparent white background.
* Box Shadow: Darker shadow for emphasis.
* Margin: Adds space around the container.
* Text Alignment: Centers the text.
* Result Text: Increases the font size for results and adds color.

8.Links

a {

display: inline-block;

margin-top: 15px;

text-decoration: none;

color: #992151; /\* Light pink color \*/

padding: 12px 18px;

border-radius: 5px;

transition: background-color 0.3s, transform 0.2s;

}

a:hover {

text-decoration: underline;

background-color: rgba(255, 0, 102, 0.2); /\* Light pink background on hover \*/

transform: scale(1.05);

}

* Links: Styles for anchor tags:
* Display: Sets display to inline-block for proper padding.
* Text Decoration: Removes the underline by default.
* Padding: Adds space around the text.
* Hover Effect: Underlines the text and changes the background color on hover.

9.Conclusion

This CSS code enhances the user interface of a web application by providing a clean and modern design. By customizing the styles, developers can improve user engagement and accessibility. Make sure to adjust the background image path and any color values to fit the desired branding.