**Project Report**

**Swiggy Restaurant Recommendation System**

**1. Project Objective**

The primary objective of this project was to develop a functional restaurant recommendation system using a dataset of Swiggy listings. The goal was to build an interactive web application that allows users to input their preferences—such as city, area, cuisine, and cost—and receive a tailored list of recommended restaurants. This project demonstrates a complete end-to-end data science workflow, from raw data processing to a user-facing application.

**2. Our Step-by-Step Approach**

The project was executed in three distinct, sequential phases to ensure a structured and organized workflow:

**Step 1: Data Cleaning and Preprocessing**

* **Goal:** To transform the raw swiggy.csv dataset into a clean, reliable, and structured format suitable for analysis.
* **Output:** A clean dataset named cleaned\_swiggy.csv.
* **Script:** datacleaning.py

**Step 2: Feature Engineering and Model Building**

* **Goal:** To convert the cleaned data into a numerical (encoded) format that a machine learning model can understand and to build the core recommendation logic.
* **Output:** A preprocessor model (preprocessor.pkl) and a complete data file (restaurant\_data.pkl) for the app and encoded\_data.csv.
* **Script:** feature\_engineering.py

**Step 3: Application Development**

* **Goal:** To build an interactive and user-friendly web interface using Streamlit, allowing users to interact with the recommendation engine.
* **Output:** A fully functional web application.
* **Script:** app.py

**3. Data Analysis and Cleaning Steps**

The initial dataset contained several issues that needed to be addressed before it could be used. The datacleaning.py script performed the following critical tasks:

* **Handling Duplicates:** The first step was to identify and remove any duplicate rows to prevent skewed analysis.
* **Location Data Normalization:** The city column often contained both the area and the city (e.g., "Vastrapur, Ahmedabad"). This was processed to create two separate, clean columns: city and area. This is crucial for allowing users to filter by both a broad city and a specific neighborhood.
* **Cleaning rating\_count:** This column had inconsistent string formats like "50+ ratings", "1K+ ratings", and "Too Few Ratings". A custom function was implemented to convert these into consistent numerical values (e.g., "50+" became 51, "1K+" became 1000) and to discard entries with too few ratings, as they are not reliable for recommendations.
* **Cleaning cost:** The cost column included the currency symbol '₹' (e.g., "₹ 200"), which prevented it from being treated as a number. The script removed this symbol to allow for numerical analysis.
* **Data Type Conversion:** The rating, cost, and rating\_count columns were converted to a numeric data type. Any values that could not be converted were treated as errors and subsequently removed.
* **Handling Missing Values:** After the cleaning steps, any rows that still had missing critical information (like rating, cost, or cuisine) were dropped using dropna() to ensure the final dataset was complete and robust.

This rigorous cleaning process was essential for the accuracy and reliability of the final recommendation model.

**4. Recommendation Methodology**

Our recommendation system is a **Content-Based Filtering** model. In simple terms, it recommends restaurants by finding others that are most similar to a user's stated preferences. The core of this methodology is broken down into two parts:

**4.1. Feature Engineering: Teaching the Model to Understand Data**

A machine learning model only understands numbers. The goal of feature\_engineering.py was to convert our dataset into a purely numerical format.

* **Numerical Features (rating, cost, rating\_count):** These were scaled using MinMaxScaler. This technique rescales all numbers to be within a 0-to-1 range. This is vital because it prevents a feature with large values (like cost) from unfairly dominating features with smaller values (like rating).
* **Categorical Features (city, area, 'cuisine'):** These text-based features were processed using OneHotEncoder. This technique converts each unique category (e.g., "Chennai", "Adyar", "Chinese") into a new numerical column with a 1 or a 0, effectively turning text into numbers the model can process.
* **ColumnTransformer:** We used this powerful tool to apply the correct transformation (MinMaxScaler or OneHotEncoder) to the correct columns automatically. The "fitted" ColumnTransformer was saved as our preprocessor.pkl model.

**4.2. Similarity Calculation: The "Filter and Rank" Logic**

Initially, a pure similarity search returned irrelevant results. We implemented a more intelligent and intuitive **"Filter and Rank"** approach in the final app.py:

1. **Filter First:** When a user makes a selection, the application first performs a hard filter on the dataset. It creates a smaller "candidate pool" of restaurants that strictly match the user's selected **City** and **Cuisine**. This guarantees that every recommendation is fundamentally relevant.
2. **Rank by Similarity:** Next, the system uses the preprocessor model and **Cosine Similarity** to rank the restaurants within this candidate pool. Cosine Similarity measures the "angle" between two restaurants based on their features (cost, rating, area, etc.). A smaller angle means a higher similarity. The app ranks the candidates based on how similar they are to the user's complete profile and returns the top 5.

This hybrid approach provides the best of both worlds: the relevance of filtering and the nuanced ranking of a machine learning model.

**5. Final Application and Key Results**

The final deliverable is a fully functional web application built with Streamlit.

* **User Interface:** The application features a clean, two-column layout with intuitive dropdown menus for selecting city, area, and cuisine, along with sliders for specifying a preferred cost and minimum rating.
* **Functionality:** The app successfully integrates the pre-built model to provide real-time restaurant recommendations based on user input.
* **Outcome:** We have successfully built a personalized recommendation engine that helps users discover new restaurants tailored to their specific tastes and preferences, fulfilling all the core requirements of the project.

**6. Insights and Future Improvements**

* **Key Insight:** The project underscored that effective data cleaning is the most critical phase of any data science project. The initial model was useless until the inconsistencies in the cost, rating\_count, and city columns were resolved.
* **Future Work:**
  + **Incorporate NLP:** Analyze user reviews to add another layer of similarity (e.g., recommend based on restaurant "vibe" or specific popular dishes).
  + **Hybrid Model:** Integrate **Collaborative Filtering** to recommend restaurants based on what other users with similar tastes have liked.
  + **More Filters:** Add more filtering options, such as delivery time, specific offers, or vegetarian/non-vegetarian options.

This project serves as a strong foundation for a more advanced and feature-rich recommendation platform.