Swiggy Restaurant Recommendation System Using K-Means Clustering & Streamlit

1. Introduction

Food delivery platforms like Swiggy host thousands of restaurants across India, making personalized recommendations crucial for user satisfaction. Traditional systems either rely on collaborative filtering or manual tagging, which often leads to inaccurate or non-personalized suggestions.

This project aims to build an end-to-end restaurant recommendation system that uses unsupervised machine learning (K-Means Clustering) combined with content-based filtering and presents real-time results using a Streamlit web application.

This work demonstrates practical skills in:

- Data preprocessing
- Feature engineering
- Machine learning model building
- Clustering
- Web deployment
- Python scripting and UI development

2. Problem Statement

Given a large dataset of Swiggy restaurants, the objective is to recommend the most suitable restaurants based on user preferences such as:

- City
- Cuisine type
- Minimum rating
- Maximum cost for two

The system should:

1. Organize restaurants into meaningful groups using K-Means clustering
2. Accept user inputs via a web UI

3. Recommend top relevant restaurants from the appropriate cluster

3. Objectives

Primary Objectives

- To analyze, clean, and preprocess Swiggy's dataset
- To encode complex categorical features for ML compatibility
- To build a recommendation pipeline using K-Means clustering
- To deploy the system using Streamlit for user interaction

Secondary Objectives

- Improve recommendation speed using MiniBatchKMeans
- Ensure the system handles missing or inconsistent data
- Provide fallback recommendations when filters are too strict

4. Dataset Description

Dataset:

swiggy.csv
Total Records: ~148,541

Columns:

Column	Description
id	Restaurant ID
name	Restaurant name
city	City & locality
rating	Swiggy rating (0–5)
rating_count	Number of reviews
cost	Cost for two (₹)
cuisine	Comma-separated cuisines

Column	Description
lic_no	License number
address	Exact restaurant address
link	Swiggy restaurant link
menu	Menu description

Data Challenges

- Missing values (rating, cost, rating_count)
- Inconsistent strings (e.g., "--" instead of rating)
- Mixed cuisine types in a single string
- Large dataset causing slow processing

5. Exploratory Data Analysis (EDA)

Key observations:

- Many ratings were "--" and required conversion
- Cost had significant missing values (~148k)
- Cuisine had diverse formatting, multiple labels
- Ratings mostly concentrated between 3.0–4.5
- Cities had irregular naming and needed normalization

EDA was performed in Jupyter Notebook, covering:

- Data type inspection
- Null value detection
- Unique value exploration
- Distribution understanding

6. Data Preprocessing (preprocess.py)

Preprocessing steps included:

6.1 Cleaning

- Converted rating "--" → NaN → replaced with median
- Cost missing values → filled with ₹300 (logical baseline)
- Rating count NaN → filled with 0
- Converted cuisine strings to lowercase, normalized commas
- Created cuisine_list column using str.split()

6.2 One-Hot Encoding (City)

Used OneHotEncoder for high-cardinality city values:

- Created binary columns like city_Chennai, city_Delhi, etc.
- Saved encoder as city_encoder.pkl

6.3 Multi-Hot Encoding (Cuisine)

- Extracted unique cuisines
- Created 1/0 flags for each cuisine (multi-label encoding)
- Saved cuisine categories as cuisine_list.pkl

6.4 Final Encoded Dataset

Created encoded_data.csv containing:

- One-hot cities
- Multi-hot cuisines
- Rating, rating_count, cost

7. Model Selection

Why K-Means Clustering?

- Dataset is large (150K+ rows)
- No target label (unsupervised task)
- Clustering groups restaurants with similar characteristics
- Faster & more scalable than collaborative filtering

Model Used:

MiniBatchKMeans

- Faster than standard KMeans
- Suitable for large-scale datasets
- Partial fitting (batch processing) improves speed

Hyperparameters

Parameter	Value
n_clusters	**25**
batch_size	**1024**
max_iter	**200**
reassignment_ratio	**0.01**

Output Saved

- kmeans.pkl (trained model)
- Updated cleaned_data.csv with cluster labels

8. Recommendation Engine (recommendation.py)

Process Flow

1. User input → city, cuisine, min rating, max cost
2. Create feature vector identical to training structure
3. Use K-Means to assign user to nearest cluster
4. Retrieve all restaurants from that cluster
5. Filter by:
   - City
   - Cuisine match
   - Rating ≥ min_rating
   - Cost ≤ max_cost

1. If strict match fails → show fallback recommendations

2. Sort and return top N results

9. Streamlit Web Application (app.py)

Features

- **Sidebar panel with filters:**
    - City
    - Cuisine
    - Min Rating slider
    - Max Cost slider
    - Recommendations count
- **Restaurant cards showing:**
    - Name
    - City
    - Cuisines
    - Rating (Stars displayed)
    - Cost
    - Swiggy link
- **Fallback recommendation logic**
- **Clean modern layout with customized styles**

UI Enhancements Done

- Left sidebar for filters
- Reduced text size for readability
- Restaurant card layout using st.container()
- K-Means logo replaced with dining icon

10. Results

Output Accuracy

- **Recommendations match correctly for city + cuisine**
- **Fast prediction due to clustering**
- **Meaningful fallback suggestions prevent blank results**

Example Output (American, Adyar, Chennai)

- BurgerMan
- CHORD V
- Meat & Eat
- KFC
- English Breakfast Box

✓ Output is accurate
✓ All filters affect results correctly

11. Evaluation

1. **Quality Metrics**
2. **Since K-Means is unsupervised, intrinsic evaluation is used:**

- Cluster Cohesion (similar restaurants grouped)
- Recommendation Relevance (manual inspection)
- Speed Performance (real-time suggestion**)**

3. **Performance**

- Encoding: Fast
- Model training: ~10–20 seconds
- Recommendation: <100ms
- Streamlit app: Smooth and responsive

12. Conclusion

This project successfully builds an efficient Restaurant Recommendation System using:

- Extensive pre-processing
- Multi-hot & one-hot encoding
- K-Means clustering for grouping
- Streamlit for web-based interaction

The system:

- Handles large-scale data
- Provides personalized, fast, accurate recommendations
- Has clean UI with fallback logic
- Is scalable for future improvements