**Swiggy’s Restaurant Recommendation System using Streamlit**

**STEP 1: IMPORTING NEEDED LIBRARIES**

>>>import pandas as pd import numpy as np import >>>pickle from sklearn.preprocessing import >>>OneHotEncoder from sklearn.cluster import >>>KMeans from sklearn.metrics.pairwise import >>>cosine\_similarity from sklearn.decomposition >>>import PCA from sklearn.preprocessing import StandardScaler

**STEP 2: Data Preprocessing and Encoding for Swiggy Restaurant Dataset.**

Explanation: This script performs data cleaning and preprocessing on a raw Swiggy restaurant dataset to prepare it for analysis or model building.

1. Loading Data: The raw dataset is loaded from a CSV file.
2. Handling Missing Values: It checks and displays unique values for specific columns, drops rows with missing values in essential columns (city, cuisine, and name), and handles invalid numeric values (e.g., converting ratings and cost to numeric).
3. Text Standardization: Cleans up text columns (city and cuisine) by stripping spaces and ensuring proper capitalization.
4. One-Hot Encoding: Converts categorical columns (city and cuisine) into binary representations using OneHotEncoder, and saves the encoder as a pickle file for future use.
5. Saving Cleaned Data: The cleaned data (with both original and encoded features) is saved into separate CSV files (cleaned\_data.csv and encoded\_data.csv).

**STEP 3: City Name Cleaning and Standardization for Swiggy Dataset.**

Explanation: This script processes and standardizes the city names from the Swiggy dataset to ensure uniformity and remove unnecessary information.

1. Loading Data: The dataset is loaded from a specified file path (swiggy.csv).
2. Extracting City Column: The city column is extracted for further processing.
3. Cleaning and Standardizing City Names: A function (clean\_city\_name) is defined to:
   * Remove any extra details after a comma (e.g., 'phase', 'sector') to retain only the main city name.
   * Remove unwanted terms like "phase", "sector", "vihar", etc., if they are part of the city name.
4. Applying the Cleaning Function: The function is applied to all city names to clean and standardize them.
5. Removing Empty Cities: Any empty city names are removed from the list.
6. Getting Unique Cities: The unique, cleaned city names are extracted and sorted for better readability.
7. Output: The cleaned and sorted unique city names are printed.

This process helps in standardizing city names by removing unwanted text, making it easier for analysis and further processing.

**STEP 4: Restaurant Recommendation System Using PCA and Cosine Similarity.**

Explanation:

This script implements a restaurant recommendation system that leverages dimensionality reduction (PCA) and cosine similarity to recommend restaurants based on cuisine and city input.

1. Loading Data:
   * The encoded and cleaned datasets (encoded\_data.csv and cleaned\_data.csv) are loaded into memory.
2. Checking for Missing Values:
   * Missing values in the encoded dataset are checked using isnull().sum().
3. Standardizing Data:
   * The data is standardized using StandardScaler() to ensure each feature has a mean of 0 and a standard deviation of 1, which is important before applying PCA.
4. PCA for Dimensionality Reduction:
   * Principal Component Analysis (PCA) is applied to the standardized data to reduce the number of features while retaining the most important information. The number of components is set to 10, but it can be adjusted based on the requirements.
   * The explained variance ratio of the PCA components is printed to evaluate how much variance is captured by each principal component.
5. Restaurant Recommendation Function:
   * The function recommend\_restaurant() takes user input for cuisine and city and recommends similar restaurants based on cosine similarity:
     + It identifies the index of the input restaurant in the dataset based on the provided cuisine and city.
     + The input restaurant's feature vector is standardized using the same scaler and transformed using PCA.
     + Cosine similarity is calculated between the input restaurant and all other restaurants.
     + The top N most similar restaurants are selected if their similarity score meets the defined threshold.
6. Recommendation Output:
   * The function returns the recommended restaurants, which meet the similarity threshold, and their corresponding similarity scores.
7. Example Usage:
   * An example input of "Indian" cuisine and "Abohar" city is used to recommend similar restaurants, and the results are printed, including similarity scores and PCA-transformed features.

This method allows for efficient restaurant recommendations based on similarity in a reduced feature space, leveraging PCA for dimensionality reduction and cosine similarity for measuring similarity between restaurant feature vectors.  
  
**Conclusion:**

In conclusion, the restaurant recommendation system implemented in this script effectively combines data preprocessing, dimensionality reduction (via PCA), and similarity-based techniques (via cosine similarity) to offer personalized recommendations. By standardizing the data and reducing its dimensionality, the system efficiently identifies and ranks restaurants that closely match the user’s preferences in terms of cuisine and city. The ability to filter recommendations based on similarity thresholds ensures that only the most relevant suggestions are presented, improving user experience and decision-making. This approach can be easily adapted or extended to various other recommendation systems across different domains, making it a versatile and scalable solution for personalized recommendations.