PROBLEM STATEMENT:

Meal Recommender System: Based on historical purchases of a buyer and/or his personal information (age, gender, health conditions etc.), define buyer persona. When the buyer prepares his current cart, the meal recommendation wizard could provide him with food choices, i.e. the meal this buyer is most likely to prepare. And then accordingly he can either directly add the prepared meal to his cart or add the missing ingredients to his cart.

DATSET:

Dataset is synthetically generated by using different sources mentioned below:

a. Users' personal data collection: Data like favorite cuisines, food spiciness preferred, food choices, allergies, etc. is collected by circulating form.
b. Recipe and Ingredients mapping: CulinaryDB dataset
c. Historical purchases of users: SalesDB dataset

APPROACH:

1. ASSOCIATIVE RULE MINNING:
    a. BRIEF:
        i. Association rules can be thought of as an IF-THEN relationship. Suppose item **A (Antecedent)** is being bought by the customer, then the chances of item **B (Consequent)** being picked by the customer too.
        ii. Apriori algorithm uses frequent itemsets to generate association rules. It is based on the concept that a subset of a frequent itemset must also be a frequent itemset. Frequent Itemset is an itemset whose support value is greater than a threshold value(support). We apply an iterative approach or level-wise search where k-frequent itemsets are used to find k+1 itemsets.
        iii. Metrics:
            1. Support: frequently bought items $Support = \frac{freq(A,B)}{N}$
            2. Confidence: how often the items A and B occur together, given the number times A occurs. $Confidence = \frac{freq(A,B)}{freq(A)}$
            3. Lift: indicates the strength of a rule over the random occurrence of A and B. $Lift(X \rightarrow Y) = \frac{support(X \cup Y)}{support(X).support(Y)}$

$$\text{Support (A)} = \frac{\text{Number of transaction in which A appears}}{\text{Total number of transactions}}$$

$$\text{Confidence (A}\rightarrow\text{B)} = \frac{\text{Support(AUB)}}{\text{Support(A)}}$$

    b. MODEL1 (PURCHASE HISTORY-->NEXT_PRODUCTS):
        i. Training:

1. Applied Apriori on purchased products in previous transaction to generate Associative Rules. These rules are further filtered and ranked according to their support, confidence and lift scores.

ii. <u>Prediction:</u>

1. Used current cart items as one transaction and applied learned rules on them to predict products user is most likely to buy next.

c. MODEL2 (PERSONAL INFORMATION-->FAV_CUISINE) :

i. <u>Training:</u>

1. Applied Apriori on personal information of users related to food preferences to generate associative rules. These rules are further filtered according to their support and confidence scores.
2. Rules are further filtered and kept only those which has "Cuisine" as consequent, because we are predicting cuisines that user might like based on user's personal information.

ii. <u>Prediction:</u>

1. Rules are sorted in fixed order of features and input data is also sorted in similar order.
2. Generate subsets of input data and check if that subset exists in antecedents if the trained rules and then return its corresponding consequent. (Here rule which matches with max length subset of input is returned).

2. COLLABORATIVE FILTERING:

a. BRIEF:

i. Collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating).

ii. The underlying assumption of the collaborative filtering approach is that if a person *A* has the same opinion as a person *B* on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person.

1. <u>Metrics:</u>

a. Clustering: Silhouette Coefficient, Davies-Bouldin index

b. Silhouette Coefficient:

i. A higher Silhouette Coefficient score relates to a model with better defined clusters. The Silhouette Coefficient is defined for each sample and is composed of two scores:

ii. **a**: The mean distance between a sample and all other points in the same class.

iii. **b**: The mean distance between a sample and all other points in the *next nearest cluster*.

iv. The Silhouette Coefficient *s* for a single sample is then given as: $s=(b-a)/max(a,b)$

v. The Silhouette Coefficient for a set of samples is given as the mean of the Silhouette Coefficient for each sample.
vi. The score is bounded between -1 for incorrect clustering and +1 for highly dense clustering. Scores around zero indicate overlapping clusters.

c. Davies-Bouldin index:
i. A lower Davies-Bouldin index relates to a model with better separation between the clusters.
ii. This index signifies the average 'similarity' between clusters, where the similarity is a measure that compares the distance between clusters with the size of the clusters themselves.
iii. Zero is the lowest possible score. Values closer to zero indicate a better partition.

b. TRAINING:
i. User_Product Matrix Creation:
1. Matrix is created on purchase history of users where one row in matrix represents purchase history of one user and columns represents all products in the dataset.
2.

	Product1	Product2	Product3					Product M
User1	0	1	0	.	..	..	1	0
User2	1	0	0	.	..	..	0	1
.	.	.	.	.	..	..	..	..
.	.	.	.	.	..	..	..	..
User N	0	0	1	.	..	..	..	1

In this User-Product Matrix :
Cell[u][p] = 1 ;
if User 'u' has purchased product 'p'
Cell[u][p] = 0 ;
Otherwise

ii. Buyer Persona Creation:
1. Users are divided into clusters based on their purchase history where each cluster represents a persona.
2. User_Product Matrix created above is given to different clustering algorithms to divide users into different personas.
3. PCA applied to reduce dimensions of above matrix.
4. Clustering algorithms K-means, Hierarchical and Spectral are tried with different number of clusters.
5. Results of above are compared using Silhouette score and Davis Bouldin index to choose best model's clustering results.
6. Best model: K-Means with 9 clusters and dimensions reduced to 100

iii. Auto-encoder training:
1. Auto-encoder is trained on User_Product matrix computed above to predict products user is most likely to buy from purchase history.

2. Auto-encoder tries to do dimensionality reduction in encoder phase and reconstruction of data in decoder phase. We are using reconstructed data by decoder and predict products user is most likely to buy by analyzing where decoder predicted values close to 1 but it was 0 in original data.
3. User persona is also taken into consideration during prediction by training separate auto-encoder for each persona/cluster.
4. Due to persona, similar users will get similar recommendations of products.

c. PREDICTION:
   i. Case 1: User is already in database (User from training data)
      1. Fetch cluster, fetch results from trained model of auto-encoder for that cluster
   ii. Case 2: New user (Not in database already)
      1. Assign cluster using cart products of user and applying trained model of cluster
      2. Find most similar user from that cluster using cosine similarity.
      3. Fetch Auto-encoder's result for that similar user and return it as prediction for current user

3. RECIPE EMBEDDINGS
   a. BRIEF:

We have used word2vec skipgram model with negative sampling. (If what is negative sampling is asked!!!, for every word we are predicting the probability of a word being context word, here we change it a little bit. We predict whether a word can be input word's context word or not. So say we have 1000 unique words, so instead of building a classifier with 1000 classes, we have changed our problem to 1000 binary classifiers. Now how weights are updated? For all true cases, we update the weight, but for negative values, we pick k random values and update weights for them only.)

   b. TRAINING:

We have generated embeddings for each ingredient by training Word2vec skipgram model. Input to this model is list of recipes where each recipe is list of ingredients. The model generates two vectors for each ingredient, Target vector(ingredient's own embedding) and context vector(ingredient's context embedding). The context for a ingredient are the other ingredients within the same recipe.

   c. PREDICTIONS
      i. To find similar ingredients to a given ingredient, we have computed cosine similarity of target vectors.
      ii. To find complementary ingredients of a given ingredient, we have computed cosine similarity of target vectors and context vectors.
      iii. To find recipe recommendations, we use output vectors (dot product of target and context vector)

4. RECIPE RECOMMENDER
    a. CURRENT INGREDIENTS:
        i. Current cart Items
        ii. Predictions obtained after applying Collaborative filtering on purchase history
        iii. Predictions obtained after applying Associative rule mining on purchase history
        iv. Complementary ingredients of current cart items using recipe embeddings.
        v. Taken intersection of above 3 predictions and passed it to recipe embeddings along with current cart items
    b. RECIPE RECOMMENDATION
        i. FROM MODEL
            1. Recipe vectors: we compute vectors for each recipe by taking average of output vector of ingredients present in recipe
            2. Average of output vectors of current ingredients
            3. Compute cosine similarity between vectors obtained from above two averages.
        ii. FROM DATABASE
            1. SQL query to database with current cart items

PERFORMANCE METRICS:

1. Coverage:
    a. Coverage is the percent of items in the training data the model can recommend on a test set.
    b. Computed on similar products results:
        i. For each user is test data,
            1. Assume user's purchase history as current cart items and apply our recommender on it
            2. Similar products predicted from recommender are stored.
        ii. For all users in test data:
            1. Compute average as unique products predicted as above for all users divided by total number of unique products in the database
    c. Value = 7.7 (Generally, recommendation systems show around 7-8% of coverage)
2. Personalization:
    a. Indicates how much personalized experience the model is offering to each user. It is the dissimilarity (1- cosine similarity) between user's lists of recommendations. A high personalization score indicates user's recommendations are different, meaning the model is offering a personalized experience to each user.
    b. Computed on recipe recommendations
        i. Created a User_Recommended_Recipe Matrix for all the users in the test data where each row represents a user and columns are all unique recipes stored in the database. Matrix will have value 1 if particular recipe is recommended to the user, otherwise 0.
        ii. Calculated cosine similarity between the recommendations
3. Intra-List Similarity:

a. The average cosine similarity of all items in a list of recommendations. This calculation uses features of the recommended items (such as favorite cuisine) to calculate the similarity
b. Computed on recipe recommendations
    i. In our case, For each recipe in recommendation list, perform One Hot Encoding on it where columns of the vector are all cuisines in the dataset and value is 1 if recommended recipe belongs to the cuisine.
    ii. Calculate cosine similarity of each recipe with other recipes in the recommendation list.
    iii. Take average of all cosine similarities calculated above which will represent Intra-List Similarity for that user's recommendation.
    iv. Intra-list similarity for the model: We can calculate average Intra-List Similarity for all users in test set by taking average of Intra-List Similarity score of all users in the test set calculated in the above step.
    v. If a recommender system is recommending lists of very similar recipes to single users (for example, a user receives only recommendations of Italian cuisine), then the intra-list similarity will be high.

Programming Paradigm: (Both are type of Imperative)

Procedural: Functions

Object Oriented: Classes wherever required

Design Pattern: (Creational, Structural, Behavioral)

Object Oriented design: (Identify objects, their responsibility, how they interact and behave)

In our case training is one-time thing and no independent modules, so no interactions between objects of these

Flask framework used for UI: Designed database tables and their relationships. Created class for each table and mentioned their interactions with each other in Model part of the UI.

SOLID PRINCIPLES

1. Single responsibility: a class should have one and only one reason to change
2. Open Closed : a class should be open for extension and closed for modification
3. Liskov substitution: subclasses should be substitutable by their base classes
4. Interface Segregation: client should not be forced to use interfaces that they do not use

5. Dependency Inversion: High level modules should not depend on low level modules