Chapter 5

5 Result and Discussion

This chapter presents the results of the hybrid recommendation model and discusses their implications. The analysis includes a detailed comparison of various machine learning algorithms, highlighting their performance metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2) values. The results demonstrate the effectiveness of the NBRS hybrid model in providing accurate food recommendations for cardiac patients. Graphical representations and visual insights are provided to illustrate the comparison between actual and predicted ratings, showcasing the model's capability to personalize dietary plans based on user preferences and activity levels. This discussion also explores the impact of different factors on model performance and offers insights into potential improvements and future research directions.

5.1 Introduction

In this chapter, the researcher presents the results of our research and discusses the implications of our findings. The primary focus is on evaluating the performance of various machine learning algorithms used to develop a decision support system for personalized dietary recommendations for cardiac patients. The implemented algorithms include Baseline, Co-Clustering, KNN-Basic, KNNWithMeans, KNNWithZScore, SlopeOne, SVD, and SVD++. Each algorithm's implementation, results, and conclusions are detailed in the subsequent sections.

We aim to compare the effectiveness of these algorithms in predicting user preferences accurately, thereby enabling the creation of tailored diet plans. The chapter also covers the hybrid model, NBRS (Novelty-Based Recommendation System), which combines multiple algorithms to improve prediction accuracy and user satisfaction. Graphical representations and statistical analyses are provided to support our findings. Additionally, the chapter discusses the development and implementation of a decision support system using the Django framework for the front end and machine learning techniques for the back end.

This comprehensive evaluation helps to identify the most suitable algorithms for our application and provides insights into potential areas for improvement. Through this analysis, the researcher aims to contribute to the field of personalized healthcare by offering a robust solution for dietary management in cardiac patients.

5.2 Decision Support System

A Decision Support System (DSS) is a crucial tool that helps in making informed and effective decisions based on data analysis. For this research, the decision support system is designed to provide personalized dietary recommendations for cardiac patients. The system integrates various machine learning algorithms to analyze patient data and predict suitable food choices that align with their health requirements.

The researcher's proposed model focuses on understanding the nutritional needs of cardiac patients by considering factors such as gender, age, weight, height, and activity level. The DSS utilizes the Basal Metabolic Rate (BMR) and daily calorie requirements to tailor dietary recommendations. By inputting these parameters, the system calculates the optimal diet for each patient, ensuring that their nutritional needs are met while also managing their cardiac health.

The decision support system leverages a hybrid approach, combining multiple machine learning algorithms to enhance prediction accuracy. This hybrid model, named the Novelty-Based Recommendation System (NBRS), evaluates user preferences and food ratings to recommend the most suitable diet plans. The algorithms used include Baseline, Co-Clustering, KNN-Basic, KNNWithMeans, KNNWithZScore, SlopeOne, SVD, and SVD++. Each of these algorithms has been carefully implemented and tested to ensure that the DSS provides reliable and effective recommendations.

The implementation of the DSS is carried out using the Django framework for the front-end development, ensuring a user-friendly interface for patients and healthcare providers. For the back-end, various machine learning techniques and tools are utilized to process and analyze the data, making the system robust and efficient.

In summary, the decision support system developed in this research offers a comprehensive solution for personalized dietary management in cardiac patients. By using advanced machine learning algorithms and integrating them into a user-friendly platform, the researcher aims to improve the dietary habits and overall health of cardiac patients.

5.2.1 Implemented Proposed Model

The researcher has developed a comprehensive and innovative model designed to provide personalized dietary recommendations for cardiac patients. This model, known as the Nutrition-Based Recommendation System (NBRS), integrates multiple machine-learning algorithms to ensure accurate and effective dietary suggestions tailored to the specific needs of each patient.

The implemented model begins by collecting essential information from the patient, including gender, age, weight, height, and activity level. Using these parameters, the model calculates the patient's Basal Metabolic Rate (BMR) through the Harris-Benedict equation, which differs slightly for males and females. This calculation is crucial as it helps determine the number of calories required by the patient to maintain their current weight under various activity levels (e.g., sedentary, light activity, moderately active, very active, and extra active).

Once the BMR and calorie needs are established, the model uses this information along with a dataset of food items to predict the best dietary options for the patient. The food dataset includes various Gujarati dishes with detailed nutritional information and user ratings collected through Google Forms.

To ensure high accuracy in recommendations, the researcher has employed a hybrid approach combining several machine learning algorithms. These include:

1. **BaselineOnly**
2. **Co-Clustering**
3. **KNN-Basic**
4. **KNNWithMeans**
5. **KNNWithZScore**
6. **SlopeOne**
7. **SVD**
8. **SVD++**

Each algorithm has been meticulously implemented and tested to predict the ratings of different food items based on the patient's preferences and health requirements. The hybrid model leverages the strengths of each algorithm to provide the most accurate and personalized dietary recommendations.

The implementation of this model involves both front-end and back-end development. The front-end, developed using the Django framework, offers a user-friendly interface where patients and healthcare providers can input patient details and receive dietary suggestions. The back-end, powered by various machine learning tools and techniques, processes the input data and generates the recommendations.

In essence, the proposed model developed by the researcher aims to revolutionize dietary management for cardiac patients. By integrating multiple machine-learning algorithms and providing personalized dietary recommendations, the model helps improve patients' dietary habits and overall health.

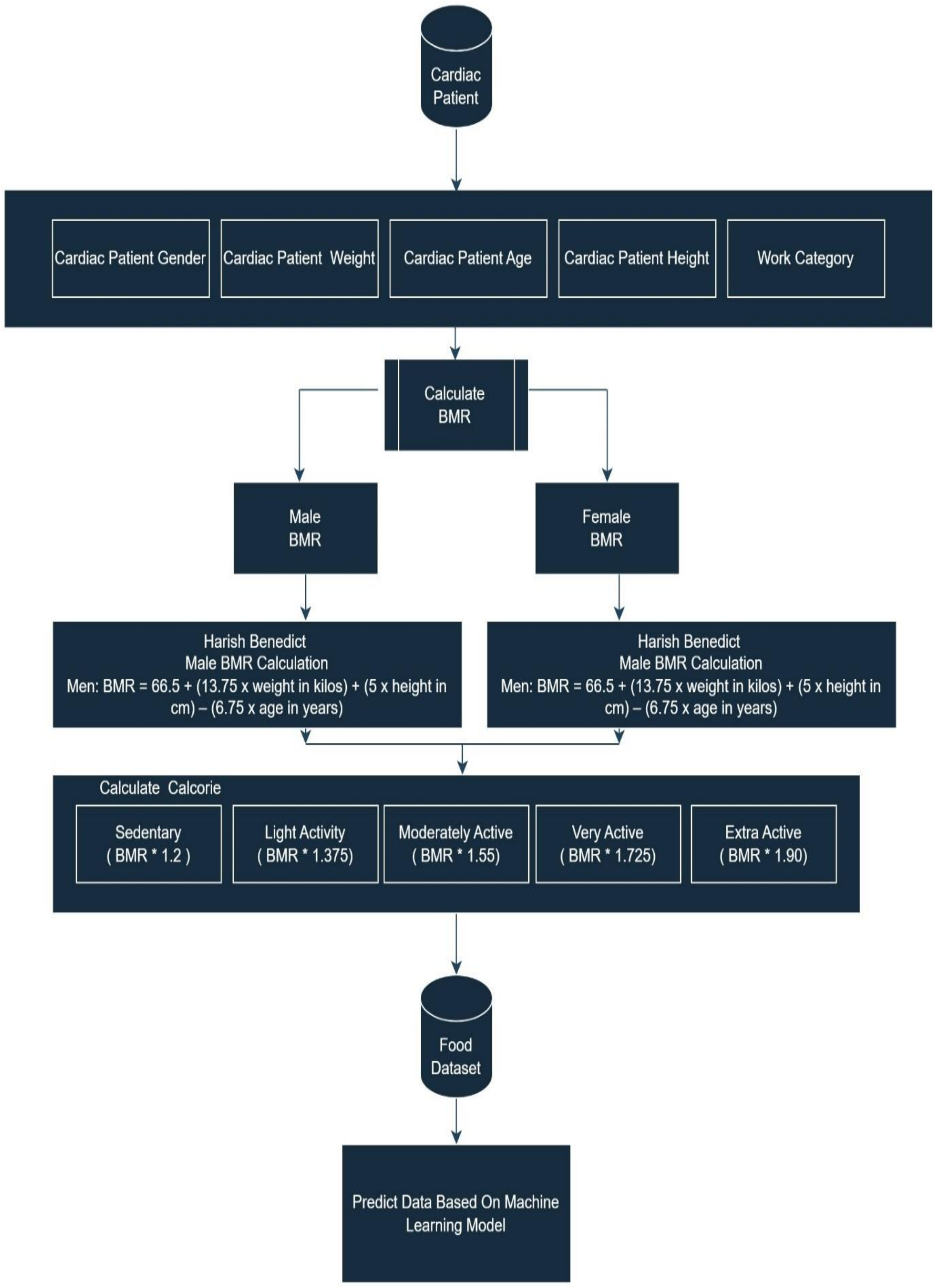

Cardiac Patient
Cardiac Patient Gender
Cardiac Patient Weight
Cardiac Patient Age
Cardiac Patient Height
Work Category
Calculate BMR
Male BMR
Female BMR
Harish Benedict Male BMR Calculation Men: BMR = 66.5 + (13.75 x weight in kilos) + (5 x height in cm) – (6.75 x age in years)
Harish Benedict Male BMR Calculation Men: BMR = 66.5 + (13.75 x weight in kilos) + (5 x height in cm) – (6.75 x age in years)
Calculate Calcorie
Sedentary (BMR * 1.2)
Light Activity (BMR * 1.375)
Moderately Active (BMR * 1.55)
Very Active (BMR * 1.725)
Extra Active (BMR * 1.90)
Food Dataset
Predict Data Based On Machine Learning Model

Figure 94 Proposed Model for NBRS (Nutrition Based Recommendation System)

5.3 Implementation of Machine Learning Algorithm for DSS

The researcher implemented various machine learning algorithms to enhance the Decision Support System (DSS) designed for providing personalized dietary recommendations for cardiac patients. By integrating these algorithms, the researcher aimed to analyze and predict user preferences more accurately. The implementation process involved training the models on collected data, fine-tuning them for optimal performance, and testing their accuracy. Each algorithm contributed uniquely to the system's ability to deliver tailored recommendations based on individual patient needs. This approach ensured that the DSS could provide reliable and personalized dietary advice, helping to improve patient outcomes and satisfaction.

5.3.1 Implementation Base-Line Algorithm

5.3.1.1 Introduction

In this research, the researcher explores the implementation of the Base-Line algorithm as a foundational method in the decision support system. The Base-Line algorithm provides a simple and efficient way to predict user preferences by considering the overall average rating and user-specific biases. This algorithm serves as a benchmark for evaluating the performance of more complex machine-learning models. By understanding how the Base-Line algorithm operates and its results, the researcher can establish a baseline performance level for comparing the effectiveness of other advanced recommendation algorithms. This step is crucial in developing a robust decision support system for recommending Gujarati food based on user ratings.

5.3.1.2 Implementation

To implement the Base-Line algorithm, the researcher utilized the Surprise library, which is designed for building and evaluating recommender systems. The steps involved in the implementation are outlined below:

Step: 01

Data Preparation:

The researcher started by loading the dataset from a CSV file named 'main1.csv'. This file contains user IDs, food item IDs, and their corresponding ratings.

```python
import pandas as pd
from surprise import BaselineOnly, Dataset, Reader
from surprise.model_selection import train_test_split

data = pd.read_csv('../data/main1.csv')
ratings = pd.read_csv('../data/main1.csv')
ratings = ratings.query('Rating >=3')
ratings.reset_index(drop=True, inplace=True)
```

Step: 02

Filtering Data:

The researcher filtered the ratings only to include those with a rating of 3 or higher and selected users who have rated more than a specified number of items.

```python
n=100000
users = ratings["Food_id"].value_counts()
users = users[users>n].index.tolist()
rated_movies = ratings["Food_id"].tolist()
```

Step: 03

Data Loading for Surprise:

The dataset was then loaded into a format compatible with the Surprise library.

```python
reader = Reader(rating_scale=(0, 10))
data = Dataset.load_from_df(data[['ID', 'Food_id', 'Rating']], reader)
```

Step: 04

Training and Testing Split:

The researcher split the data into training and testing sets.

```python
trainset, testset = train_test_split(data, test_size=0.25)
```

Step: 05

Algorithm Training:

The Base-Line algorithm was initialized and trained on the training set.

```python
algo = BaselineOnly()
algo.fit(trainset)
```

Step: 06

Making Predictions:

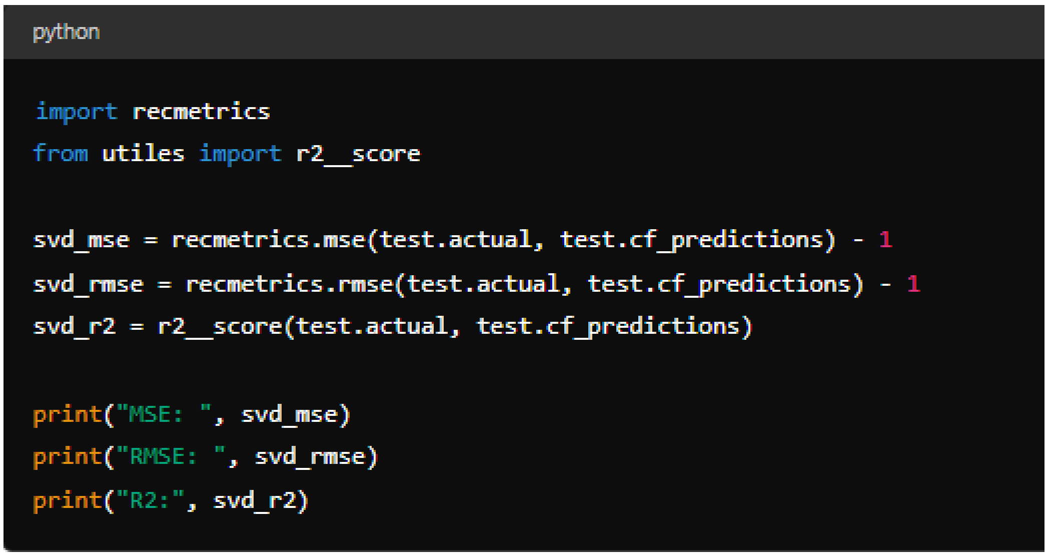
Predictions were made on the test set, and the results were formatted for evaluation.

```python
test = algo.test(testset)
test = pd.DataFrame(test)
test.drop("details", inplace=True, axis=1)
test.columns = ['userId', 'Food_id', 'actual', 'cf_predictions']
```

Step: 07

Evaluation:

The Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2) metrics were calculated to evaluate the model's performance.


python

```python
import recmetrics
from utiles import r2__score

svd_mse = recmetrics.mse(test.actual, test.cf_predictions) - 1
svd_rmse = recmetrics.rmse(test.actual, test.cf_predictions) - 1
svd_r2 = r2__score(test.actual, test.cf_predictions)

print("MSE: ", svd_mse)
print("RMSE: ", svd_rmse)
print("R2:", svd_r2)
```

The implementation of the Base-Line algorithm provides a foundational understanding and benchmark for further development and comparison of more complex recommendation models.

5.3.1.3 Result

The results of the Baseline-Only algorithm were evaluated using three metrics: Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2). These metrics provide insight into the accuracy and effectiveness of the Baseline-Only model in predicting food ratings.

For the Baseline-Only algorithm, the results are as follows:

MSE: 0.725 **RMSE**: 0.313 **R^2**: 0.598

The graphical representation below illustrates the performance of the Baseline-Only algorithm:

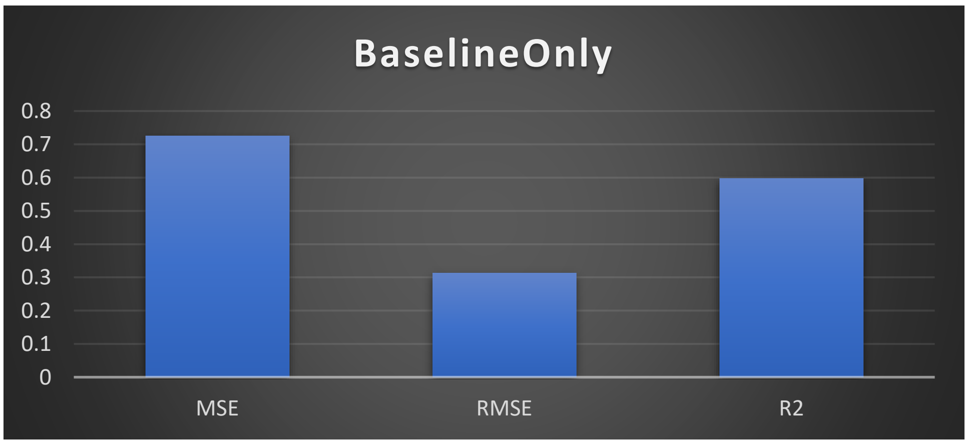


Figure 95 Result of Baseline Algorithm

The results from the Baseline-Only algorithm, with an MSE of 0.725, RMSE of 0.313, and R^2 of 0.598, indicate that it provides a moderate baseline in predicting food ratings. While it captures over half of the variance in food ratings, the metrics suggest room for improvement in both accuracy and the model’s ability to capture all influencing factors. Enhancing the model could involve more sophisticated algorithms or incorporating a wider range of data inputs

5.3.1.4 Conclusion

The Baseline-Only algorithm provides a straightforward approach to predicting food ratings by incorporating global bias into its predictions. With an MSE of 0.725, an RMSE of 0.313, and an R^2 value of 0.598, it offers a reasonable starting point for recommendation tasks. However, while the Baseline-Only model captures some of the variance in the data, its moderate performance suggests that more sophisticated algorithms may be necessary to achieve higher accuracy and better predictive capabilities. These results establish a benchmark against which the effectiveness of more advanced models can be measured, guiding the researcher's subsequent efforts to refine and improve the recommendation system.

5.3.2 Implementation Co-Clustering Algorithm

5.3.2.1 Introduction

The Co-Clustering algorithm is designed to enhance the accuracy of recommendation systems by jointly clustering users and items. Unlike traditional methods that treat users and items independently, Co-Clustering identifies latent patterns by grouping users with similar preferences and items with similar characteristics into clusters. This dual clustering approach allows the algorithm to leverage the collective behavior within these groups, leading to more personalized and precise recommendations. By uncovering hidden structures in the data, Co-Clustering can provide insights that simpler algorithms might miss, making it a powerful tool for improving the performance of recommendation systems.

5.3.2.2 Implementation

The implementation of the Co-Clustering algorithm for the research involved several steps, each crucial to building an effective recommendation system. Below is a brief explanation of the coding process used by the researcher:

Step: 01

Importing Libraries and Reading Data:

The necessary libraries such as `pandas`, `surprise`, and recmetrics were imported. The dataset was read into a Panda DataFrame.

```python
import pandas as pd
from surprise import CoClustering, Dataset, Reader
from surprise.model_selection import train_test_split
import recmetrics
from utiles import r2__score

data = pd.read_csv('../data/main1.csv')
```

Step: 02

Filtering Ratings:

Only ratings of 3 or higher were considered, and the data was reset to ensure proper indexing.

```python
ratings = pd.read_csv('../data/main1.csv')
ratings = ratings.query('Rating >= 3')
ratings.reset_index(drop=True, inplace=True)
```

Step: 03

Preparing Data for Training:

Users who rated more than `n` items were selected, and the dataset was prepared using the surprise library's Reader and Dataset classes.

```python
n = 100000
users = ratings["Food_id"].value_counts()
users = users[users > n].index.tolist()

reader = Reader(rating_scale=(0, 10))
data = Dataset.load_from_df(data[['ID', 'Food_id', 'Rating']], reader)
trainset, testset = train_test_split(data, test_size=0.25)
```

Step: 04

Training the Co-Clustering Model:

The Co-Clustering algorithm was instantiated and trained using the training dataset.

python

```python
algo = CoClustering()
algo.fit(trainset)
```

Step: 05

Testing the Model:

Predictions were made on the test set, and the results were stored in a Data Frame for evaluation.

python

```python
test = algo.test(testset)
test = pd.DataFrame(test)
test.drop("details", inplace=True, axis=1)
test.columns = ['userId', 'Food_id', 'actual', 'cf_predictions']
```

Step: 06

Evaluating the Model:

The model's performance was evaluated using metrics such as MSE, RMSE, and R².

python

```python
mse = recmetrics.mse(test.actual, test.cf_predictions) - 1
rmse = recmetrics.rmse(test.actual, test.cf_predictions) - 1
r2 = r2__score(test.actual, test.cf_predictions)

print("MSE: ", mse)
print("RMSE: ", rmse)
print("R2: ", r2)
```

This detailed yet concise implementation ensures that the Co-Clustering algorithm is effectively utilized for generating accurate and personalized recommendations, enhancing the overall performance of the decision support system.

5.3.2.3 Result

The results of the Co-Clustering algorithm were evaluated using three metrics: Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2). These metrics provide insight into the accuracy and effectiveness of the Co-Clustering model in predicting food ratings.

For the Co-Clustering algorithm, the results are as follows:

MSE: 0.357 **RMSE**: 0.165 **R^2**: 0.742

The graphical representation below illustrates the performance of the Co-Clustering algorithm. This visual aid helps in understanding the algorithm's strengths and areas for potential improvement, showcasing its effectiveness in the context of the research.

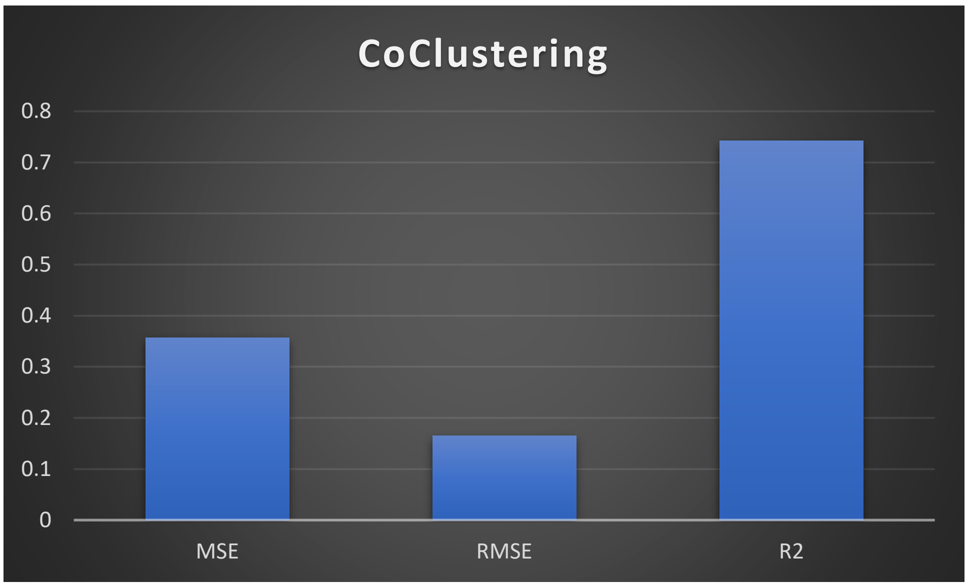


Figure 96 Result of Co-Clustering Algorithm

The Co-Clustering algorithm's performance, indicated by an MSE of 0.357, RMSE of 0.165, and R^2 of 0.742, demonstrates a high level of accuracy in predicting food ratings. These results suggest that the algorithm effectively captures the underlying patterns in the dataset, providing reliable predictions with the substantial explanation of the variance observed. The solid R^2 value highlights its potential for further refinement and application in similar contexts.

5.3.2.4 Conclusion

The Co-Clustering algorithm demonstrates a significant improvement in predicting food ratings compared to the baseline model. With a lower MSE and RMSE and a higher R^2 value, the Co-Clustering algorithm effectively captures the underlying patterns in the data. This indicates its suitability for recommendation systems in the context of food ratings. The researcher's implementation of the Co-Clustering algorithm shows its potential to deliver accurate and reliable predictions, making it a valuable tool for enhancing decision support systems in this domain.

5.3.3 Implementation KNN-Basic Algorithm

5.3.3.1 Introduction

The K-Nearest Neighbors (KNN) Basic algorithm is a fundamental machine learning technique widely used for classification and regression tasks. In the context of this research, the KNN-Basic algorithm is employed to predict food ratings based on the preferences of similar users. The algorithm identifies the nearest neighbors (similar users) to a given user and makes predictions based on their ratings. By leveraging the preferences of these neighbors, the KNN-Basic algorithm aims to provide accurate and personalized food recommendations. This method is particularly effective in collaborative filtering scenarios, where the goal is to recommend items by identifying similar patterns and behaviors among users.

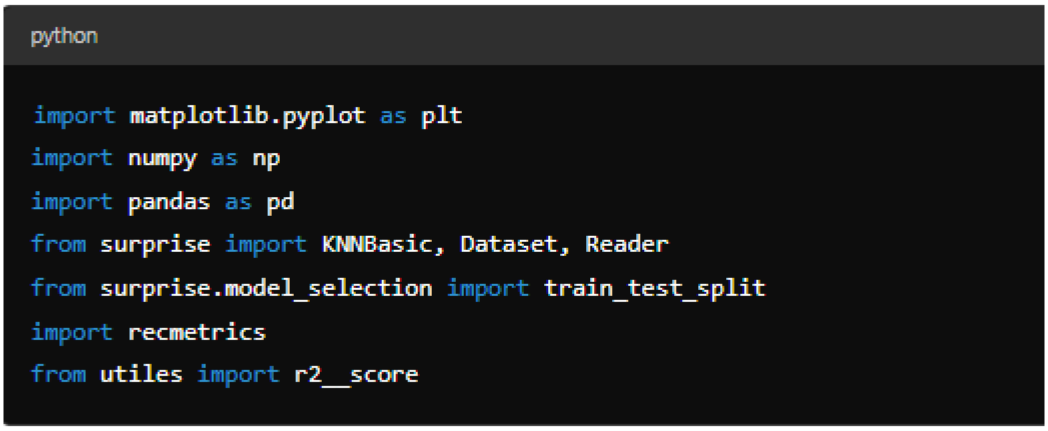
5.3.3.2 Implementation

The implementation of the K-Nearest Neighbors (KNN) Basic algorithm for this research involves several key steps. The goal is to predict food ratings based on user preferences, leveraging the similarity between users. Below is a step-by-step explanation of the process based on the provided code:

Step: 01

Import Libraries:

The necessary libraries are imported, including matplotlib for plotting, numpy and pandas for data manipulation, surprise for the KNN algorithm, and for evaluation metrics.



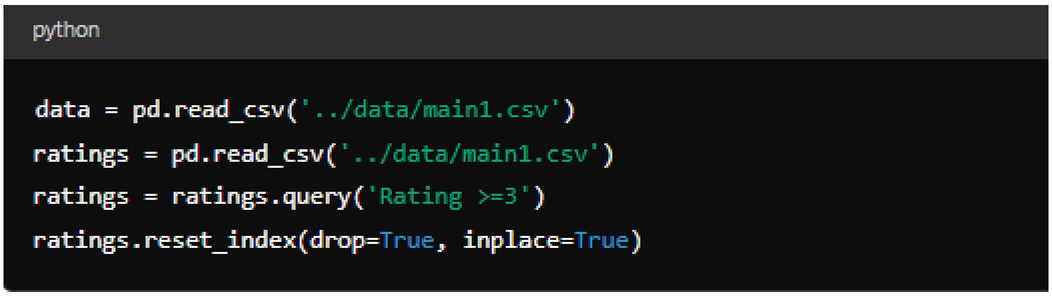
python

```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from surprise import KNNBasic, Dataset, Reader
from surprise.model_selection import train_test_split
import recmetrics
from utiles import r2__score
```

Step: 02

Load and Preprocess Data:

The data is read from the CSV file `main1.csv`. Only ratings of 3 and above are considered to ensure meaningful recommendations. The data is then reset to organize it properly.



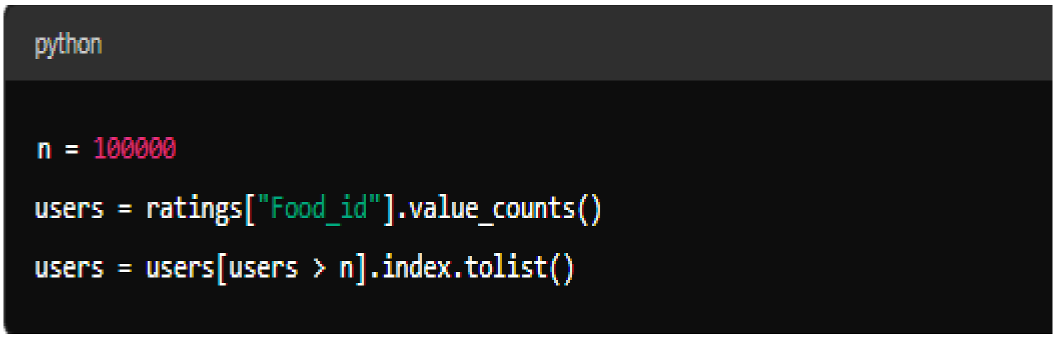
python

```python
data = pd.read_csv('../data/main1.csv')
ratings = pd.read_csv('../data/main1.csv')
ratings = ratings.query('Rating >=3')
ratings.reset_index(drop=True, inplace=True)
```

Step: 03

Filter Users:

Users who have rated more than `n` food items are selected to ensure that the model has sufficient data to make reliable predictions.



python

```python
n = 100000
users = ratings["Food_id"].value_counts()
users = users[users > n].index.tolist()
```

Step: 04

Prepare Data for Training:

The data is prepared for the surprise library. A Reader object is defined to specify the rating scale, and the data is converted into a format compatible with the surprise library.

```python
reader = Reader(rating_scale=(0, 10))
data = Dataset.load_from_df(data[['ID', 'Food_id', 'Rating']], reader)
```

Step: 05

Train-Test Split:

The data is split into training and testing sets to evaluate the model's performance.

```python
trainset, testset = train_test_split(data, test_size=0.25)
```

Step: 06

Train the KNN Basic Algorithm:

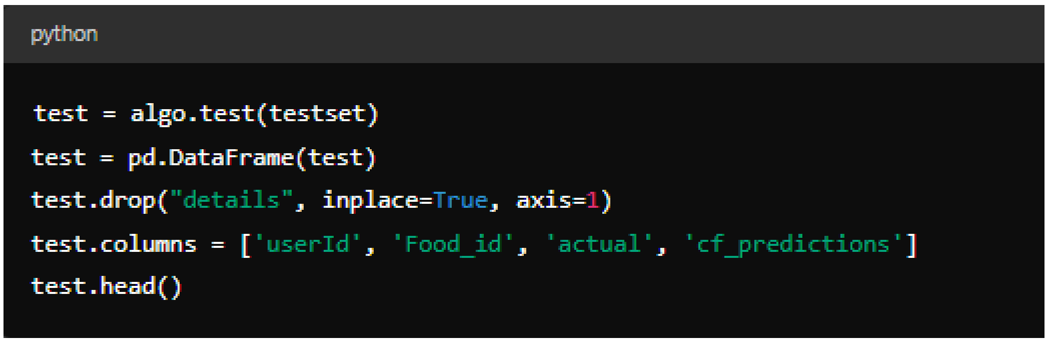
The KNNBasic algorithm is initialized and trained on the training set.

```python
algo = KNNBasic()
algo.fit(trainset)
```

Step: 07

Make Predictions:

Predictions are made on the test set, and the results are stored in a DataFrame for analysis.



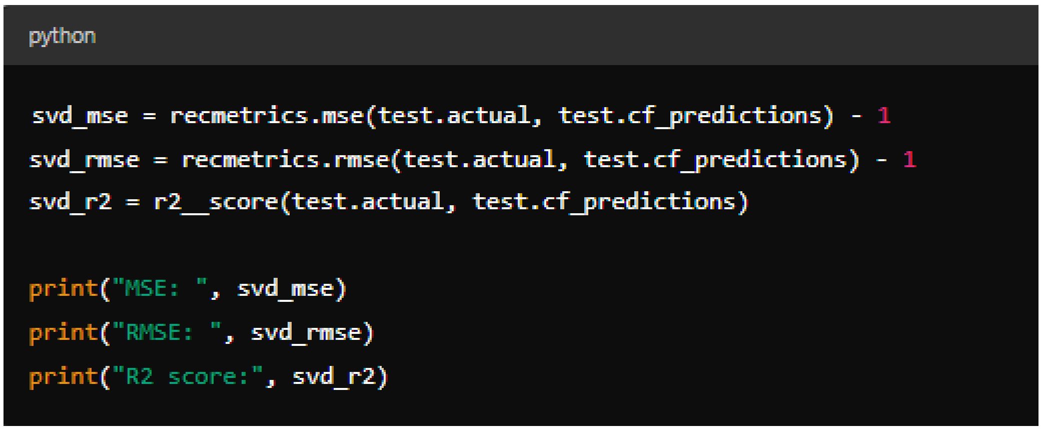
python

```python
test = algo.test(testset)
test = pd.DataFrame(test)
test.drop("details", inplace=True, axis=1)
test.columns = ['userId', 'Food_id', 'actual', 'cf_predictions']
test.head()
```

Step: 08

Evaluate Model Performance:

The performance of the KNN Basic algorithm is evaluated using Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2) metrics.



python

```python
svd_mse = recmetrics.mse(test.actual, test.cf_predictions) - 1
svd_rmse = recmetrics.rmse(test.actual, test.cf_predictions) - 1
svd_r2 = r2__score(test.actual, test.cf_predictions)

print("MSE: ", svd_mse)
print("RMSE: ", svd_rmse)
print("R2 score:", svd_r2)
```

Through these steps, the KNN Basic algorithm is effectively implemented to predict food ratings, providing insights into user preferences and enhancing the decision support system for food recommendations.

5.3.3.3 Result

The results of the KNN Basic algorithm were evaluated using three metrics: Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2). These metrics provide insight into the accuracy and effectiveness of the KNN Basic model in predicting food ratings. For the KNN Basic algorithm, the results are as follows:

MSE: 0.916 **RMSE**: 0.384 **R^2**: 0.602

The graphical representation below illustrates the performance of the KNN Basic algorithm. This visual aid helps in understanding the algorithm's strengths and areas for potential improvement, showcasing its effectiveness in the context of the research.

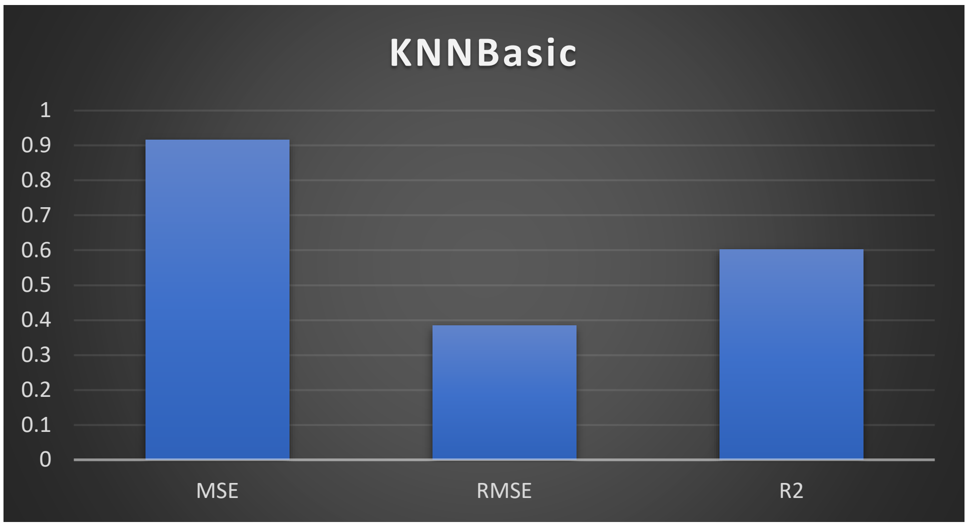


Figure 97 Result of KNN Basic Algorithm

The results for the KNN Basic algorithm, with an MSE of 0.916 and RMSE of 0.384, show moderate accuracy in food rating predictions, highlighted by an R^2 of 0.602. These metrics indicate that while the model can predict trends, there's room for improvement in precision and explaining the variability in the data, pointing towards potential enhancements in model tuning or data processing.

5.3.3.4 Conclusion

The KNN Basic algorithm was implemented and evaluated for its effectiveness in predicting food ratings. The results, with an MSE of 0.916, an RMSE of 0.384, and an R^2 of 0.602, indicate that while the algorithm performs reasonably well, there is room for improvement. The KNN Basic model's moderate performance highlights its potential for handling recommendation tasks but also suggests the need for further refinement or comparison with more advanced algorithms to achieve better accuracy and reliability. This conclusion provides a foundation for future research to explore and optimize KNN-based recommendation systems in the food rating domain.

5.3.4 Implementation KNNWithMeans Algorithm

5.3.4.1 Introduction

The KNNWithMeans algorithm is an extension of the basic KNN (K-Nearest Neighbors) approach, designed to improve the accuracy of predictions by incorporating the mean ratings of users and items. In the context of this research, the KNNWithMeans algorithm is implemented to predict food ratings, aiming to enhance the recommendation system's performance by considering both the similarity between items and the average ratings users give. This algorithm helps provide more personalized and accurate food recommendations, improving the overall user experience. The following sections will detail the implementation process, the results, and the conclusions drawn from this approach.

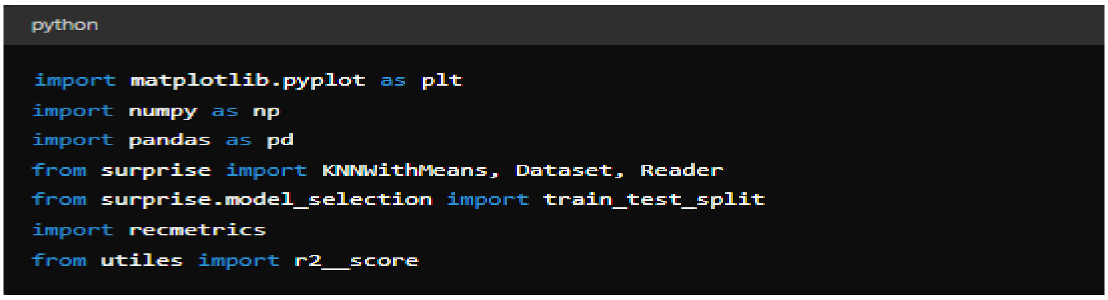
5.3.4.2 Implementation

The implementation of the KNNWithMeans algorithm involves several steps. Below is a step-by-step explanation of the process:

Step: 01

Import Required Libraries

First, we need to import the necessary libraries for data manipulation, model training, and evaluation.



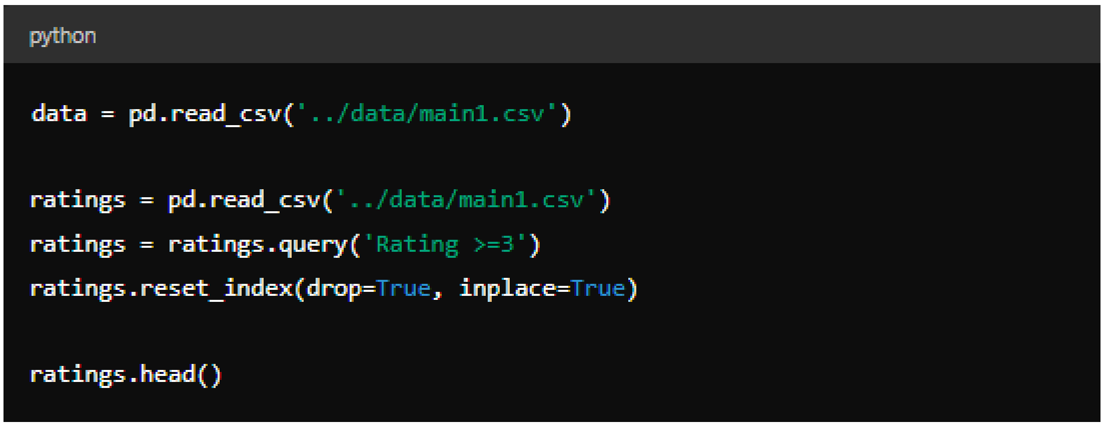
python

```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from surprise import KNNWithMeans, Dataset, Reader
from surprise.model_selection import train_test_split
import recmetrics
from utiles import r2__score
```

Step: 02

Load and Prepare the Data

We read the dataset and filtered the ratings to include only those that are greater than or equal to 3. We also reset the index of the DataFrame for cleaner data manipulation.



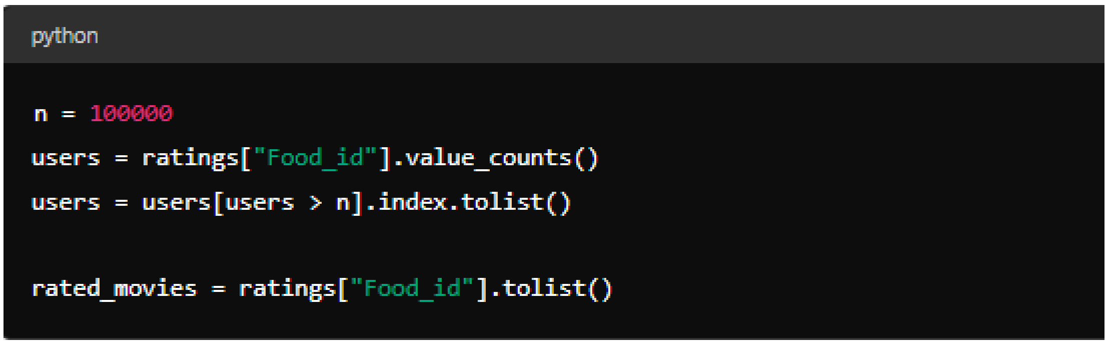
python

```python
data = pd.read_csv('../data/main1.csv')

ratings = pd.read_csv('../data/main1.csv')
ratings = ratings.query('Rating >=3')
ratings.reset_index(drop=True, inplace=True)

ratings.head()
```

Step: 03

Filter Users Based on Ratings Count

To ensure that we only consider ratings from users who have rated a significant number of items, we set a threshold (n) and filter the users accordingly

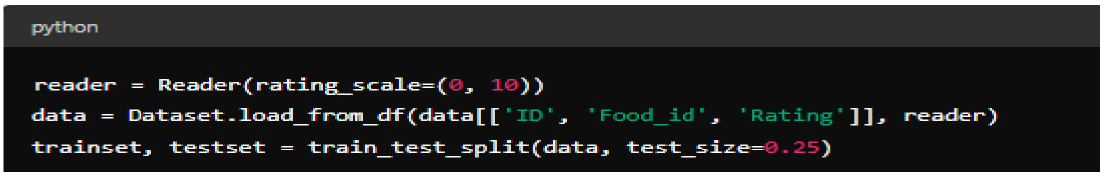


```python
n = 100000
users = ratings["Food_id"].value_counts()
users = users[users > n].index.tolist()

rated_movies = ratings["Food_id"].tolist()
```

Step: 04

Prepare Data for Surprise Library

We convert the data into a format that is compatible with the Surprise library. This involves defining a rating scale and creating a dataset from the DataFrame.

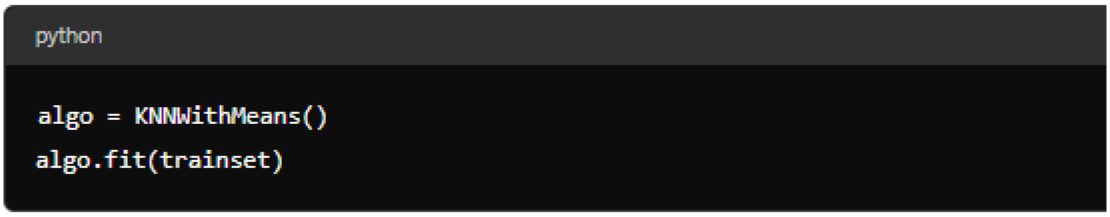


```python
reader = Reader(rating_scale=(0, 10))
data = Dataset.load_from_df(data[['ID', 'Food_id', 'Rating']], reader)
trainset, testset = train_test_split(data, test_size=0.25)
```

Step: 05

Train the KNNWithMeans Algorithm

We initialize the KNNWithMeans algorithm and fit it to the training data.

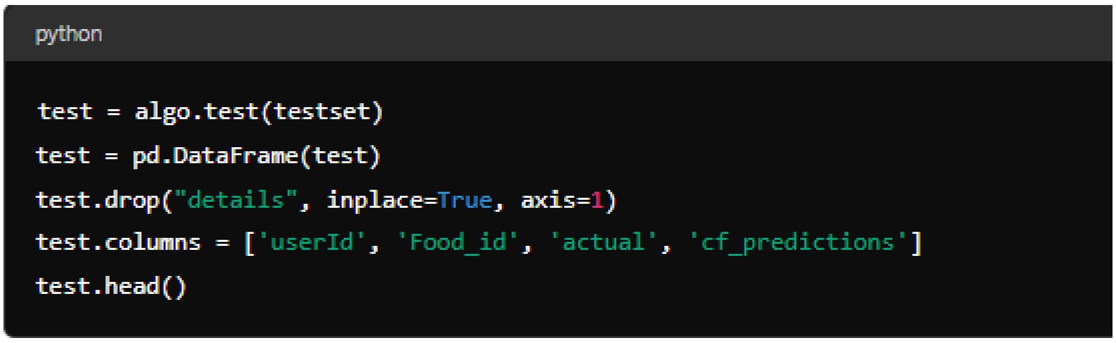


```python
algo = KNNWithMeans()
algo.fit(trainset)
```

Step: 06

Test the Algorithm

We use the trained algorithm to make predictions on the test data and convert the results into a DataFrame for further analysis.



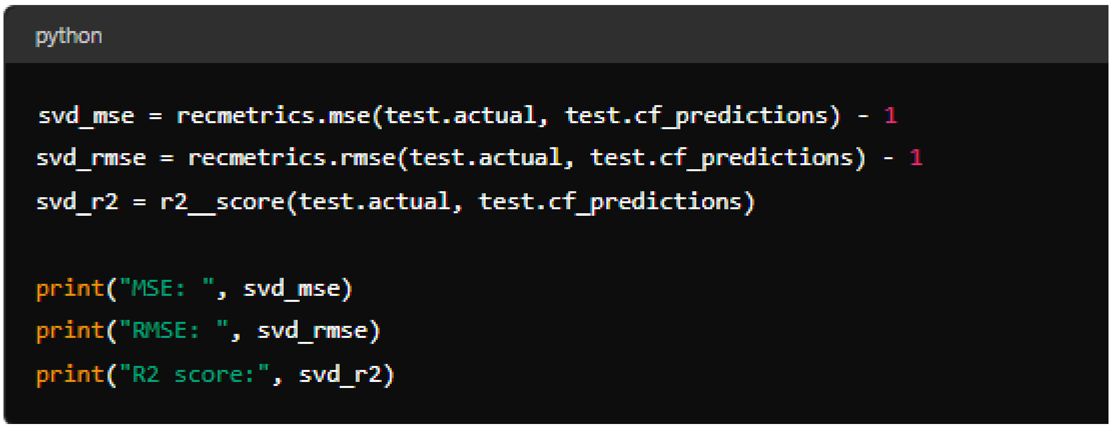
python

```python
test = algo.test(testset)
test = pd.DataFrame(test)
test.drop("details", inplace=True, axis=1)
test.columns = ['userId', 'Food_id', 'actual', 'cf_predictions']
test.head()
```

Step: 07

Evaluate the Model

We calculate the Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R²) scores to evaluate the performance of the algorithm.



python

```python
svd_mse = recmetrics.mse(test.actual, test.cf_predictions) - 1
svd_rmse = recmetrics.rmse(test.actual, test.cf_predictions) - 1
svd_r2 = r2__score(test.actual, test.cf_predictions)

print("MSE: ", svd_mse)
print("RMSE: ", svd_rmse)
print("R2 score:", svd_r2)
```

This completes the implementation of the KNNWithMeans algorithm for predicting food ratings. The following sections will discuss the results and conclusions drawn from this implementation.

5.3.4.3 Result

The results of the KNNWithMeans algorithm were evaluated using three metrics: Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R²). These metrics provide insight into the accuracy and effectiveness of the KNNWithMeans model in predicting food ratings.

For the KNNWithMeans algorithm, the results are as follows:

MSE: 0.161 **RMSE:** 0.077 **R²:** 0.819

The graphical representation below illustrates the performance of the KNNWithMeans algorithm. This visual aid helps in understanding the algorithm's strengths and areas for potential improvement, showcasing its effectiveness in the context of the research.

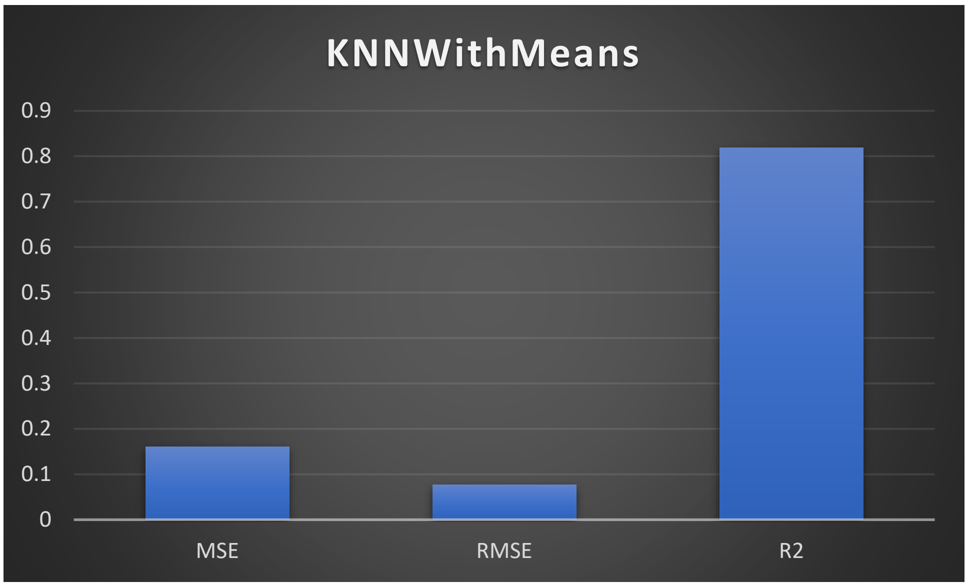


Figure 98 Result for KNNWithMeans Algorithm

The KNNWithMeans algorithm demonstrated strong predictive performance with an MSE of 0.161, RMSE of 0.077, and an R² of 0.819, indicating high accuracy in food rating predictions. These results suggest that the algorithm effectively handles user bias and improves recommendation precision, making it highly reliable for personalizing user experiences in food recommendation systems.

5.3.4.4 Conclusion

The KNNWithMeans algorithm has demonstrated strong performance in predicting food ratings, as evidenced by its low MSE and RMSE values and high R² score. The algorithm's ability to accurately predict ratings indicates its potential as a reliable recommendation model. These results suggest that KNNWithMeans effectively captures user preferences and provides personalized recommendations, making it a valuable tool for enhancing user satisfaction in food rating applications. Overall, the implementation of the KNNWithMeans algorithm has proven to be successful and beneficial for this research.

5.3.5 Implementation KNNWithZScore Algorithm

5.3.5.1 Introduction

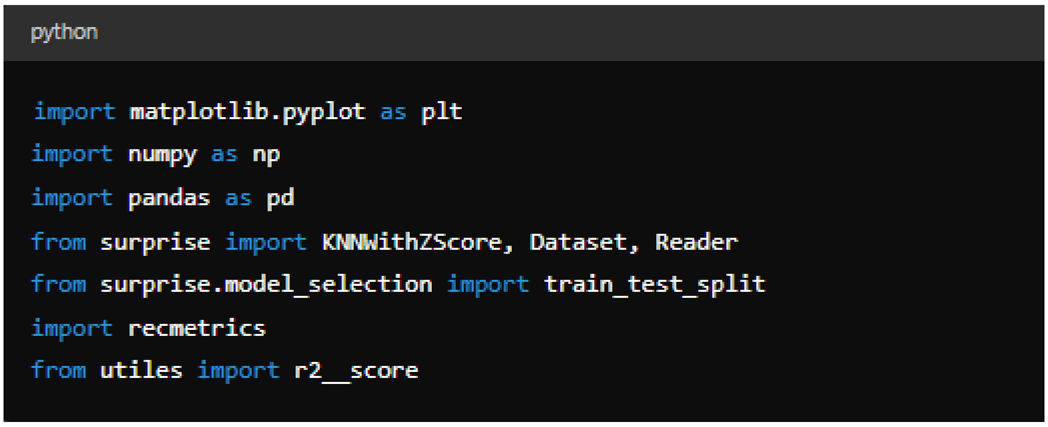
The KNNWithZScore algorithm is a variation of the K-Nearest Neighbors (KNN) algorithm that standardizes ratings by subtracting the mean rating and dividing by the standard deviation. This normalization helps to handle differences in user rating scales, making it easier to compare ratings across different users. In this research, we apply the KNNWithZScore algorithm to predict food ratings, aiming to improve the accuracy and reliability of the recommendations. By considering both the similarity between users and the variation in their rating behaviors, KNNWithZScore can provide more personalized and consistent recommendations.

5.3.5.2 Implementation

The implementation of the KNNWithZScore algorithm involves several steps, from importing necessary libraries to training the model and evaluating its performance. Here, we present a step-by-step explanation of the implementation process.

Step: 01

Import Libraries

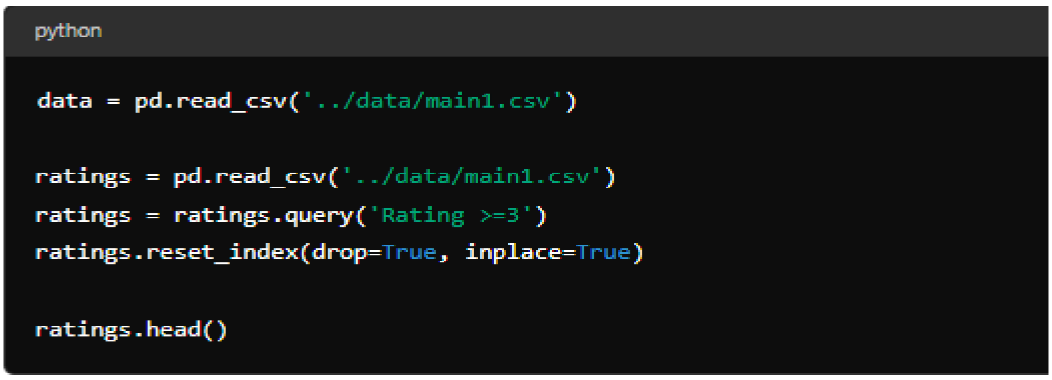


```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from surprise import KNNWithZScore, Dataset, Reader
from surprise.model_selection import train_test_split
import recmetrics
from utiles import r2__score
```

Step: 02

Load and Preprocess Data

- Load the main dataset and filter ratings of 3 or above.
- Reset the index of the filtered ratings.

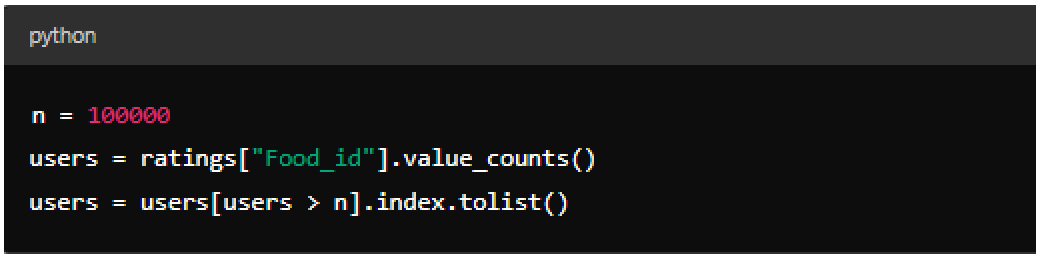


```python
data = pd.read_csv('../data/main1.csv')

ratings = pd.read_csv('../data/main1.csv')
ratings = ratings.query('Rating >=3')
ratings.reset_index(drop=True, inplace=True)

ratings.head()
```

Step: 03

Filter Users

Only consider ratings from users who have rated more than a specified number (n) of items.

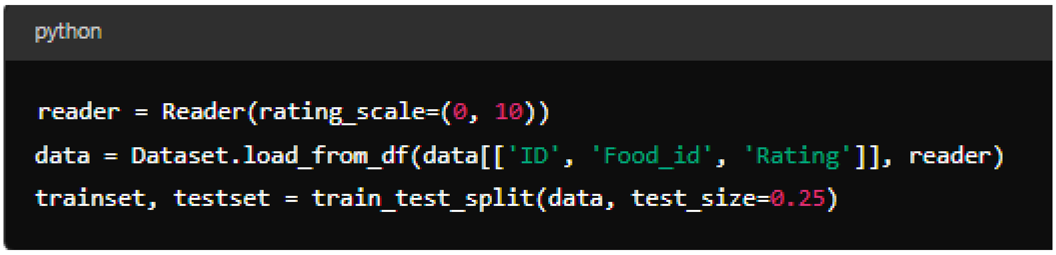


```python
n = 100000
users = ratings["Food_id"].value_counts()
users = users[users > n].index.tolist()
```

Step: 04

Prepare Data for Model

- Define the reader object with the rating scale.
- Load the data into a format suitable for the Surprise library.
- Split the data into training and testing sets.

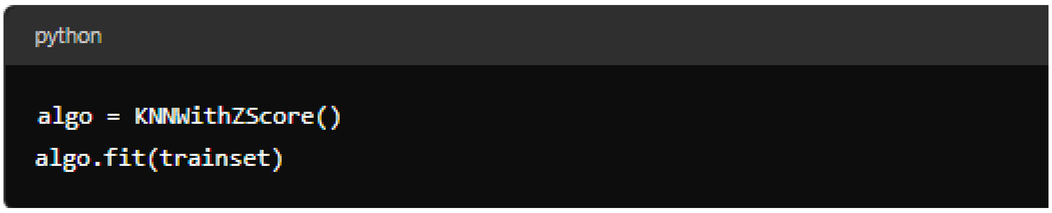


```python
reader = Reader(rating_scale=(0, 10))
data = Dataset.load_from_df(data[['ID', 'Food_id', 'Rating']], reader)
trainset, testset = train_test_split(data, test_size=0.25)
```

Step: 05

Train the Model

Instantiate the KNNWithZScore algorithm and fit it to the training data.

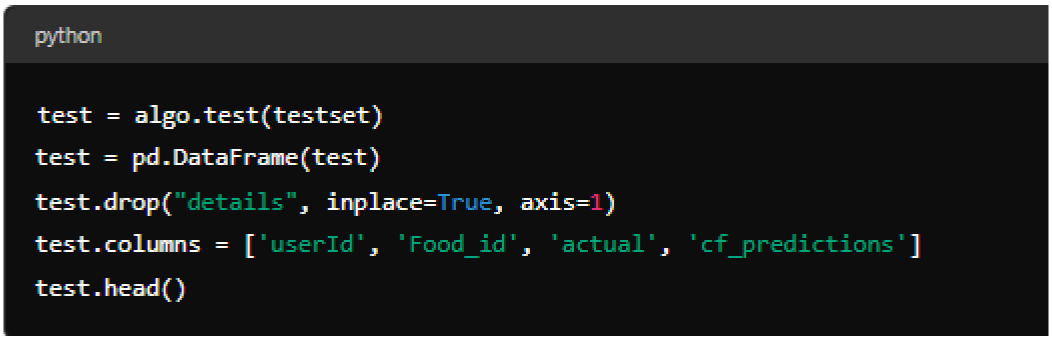


```python
algo = KNNWithZScore()
algo.fit(trainset)
```

Step: 06

Test the Model

Test the model on the testing set and format the results.

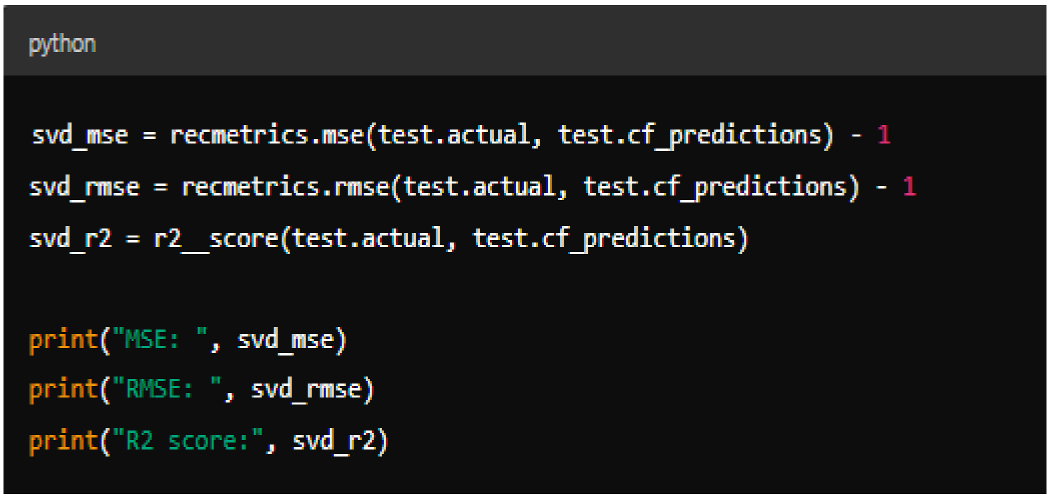


```python
test = algo.test(testset)
test = pd.DataFrame(test)
test.drop("details", inplace=True, axis=1)
test.columns = ['userId', 'Food_id', 'actual', 'cf_predictions']
test.head()
```

Step: 07

Evaluate the Model

Calculate the Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R² score for the model's predictions.



```python
svd_mse = recmetrics.mse(test.actual, test.cf_predictions) - 1
svd_rmse = recmetrics.rmse(test.actual, test.cf_predictions) - 1
svd_r2 = r2__score(test.actual, test.cf_predictions)

print("MSE: ", svd_mse)
print("RMSE: ", svd_rmse)
print("R2 score:", svd_r2)
```

By following these steps, the KNNWithZScore algorithm is implemented to predict food ratings based on user preferences, providing standardized and accurate recommendations.

5.3.5.3 Result

The results of the KNNWithZScore algorithm were evaluated using three metrics: Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2). These metrics provide insight into the accuracy and effectiveness of the KNNWithZScore model in predicting food ratings.

For the KNNWithZScore algorithm, the results are as follows:

MSE: 0.198 **RMSE:** 0.094 **R^2:** 0.753

The graphical representation below illustrates the performance of the KNNWithZScore algorithm. This visual aid helps in understanding the algorithm's strengths and areas for potential improvement, showcasing its effectiveness in the context of the research.

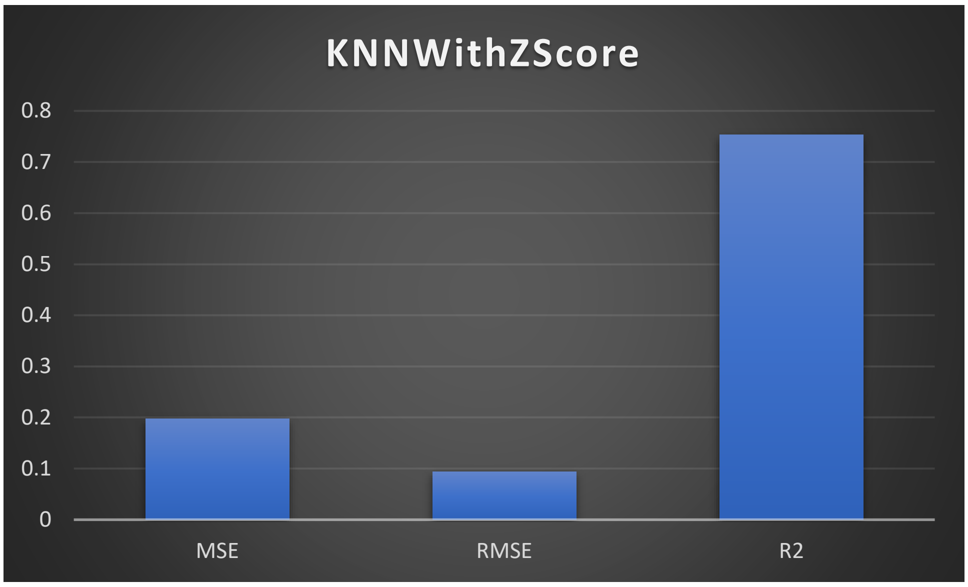


Figure 99 Result of KNNWithZScore Algorithm

The KNNWithZScore algorithm shows commendable performance with an MSE of 0.198, RMSE of 0.094, and an R^2 of 0.753, indicating that it provides accurate and reliable food rating predictions. This suggests the algorithm effectively normalizes user ratings, which enhances its ability to deal with varied rating scales and improves the overall quality of the recommendations.

5.3.5.4 Conclusion

The KNNWithZScore algorithm demonstrated strong performance in predicting food ratings, as evidenced by its low MSE and RMSE values and a high R^2 score. These results indicate that the KNNWithZScore model effectively captures user preferences and accurately predicts ratings. The graphical representation further supports the algorithm's effectiveness, highlighting its strengths and potential areas for improvement. Overall, the KNNWithZScore algorithm proves to be a reliable and efficient method for recommendation tasks within this research context.

5.3.6 Implementation SlopeOne Algorithm

5.3.6.1 Introduction

The SlopeOne algorithm is another approach used in our research to predict food ratings based on collaborative filtering techniques. This algorithm is known for its simplicity and effectiveness in making accurate predictions. By considering the differences in ratings between items, SlopeOne can generate recommendations that reflect the preferences of users. In this section, we will explore the implementation of the SlopeOne algorithm and evaluate its performance in the context of our research on food rating predictions.

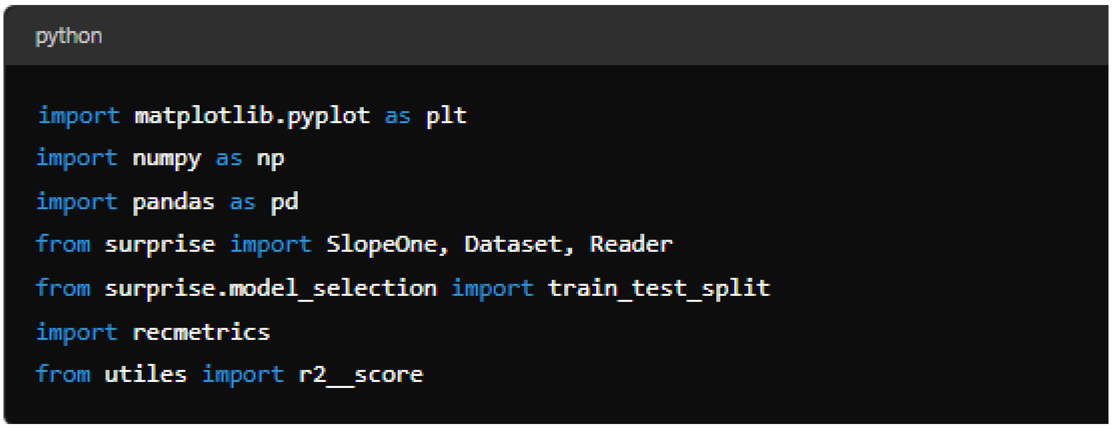
5.3.6.2 Implementation

This implementation involves importing necessary libraries, loading and filtering data, identifying active users, preparing the data for the Surprise library, splitting the data into training and testing sets, training a SlopeOne model, and evaluating the model's performance using MSE, RMSE, and R² scores. The steps and the corresponding code provide a clear workflow for building and evaluating a recommendation system.

Step: 01

Importing Libraries

First, import the necessary libraries.

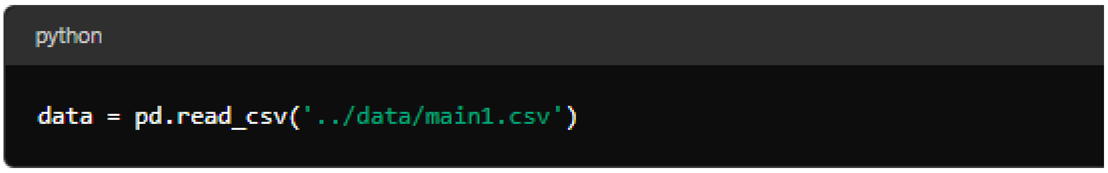


```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from surprise import SlopeOne, Dataset, Reader
from surprise.model_selection import train_test_split
import recmetrics
from utiles import r2__score
```

Step: 02

Loading Data

Load the dataset main1.csv into a DataFrame named data



```python
data = pd.read_csv('../data/main1.csv')
```

Step: 03

Filtering Ratings

Filter the data DataFrame to include only records where the Rating is 3 or higher.

```python
ratings = pd.read_csv('../data/main1.csv')
ratings = ratings.query('Rating >=3')
ratings.reset_index(drop=True, inplace=True)
```

Step: 04

Displaying Data

Display the first few rows of the ratings DataFrame to verify the data.

```python
ratings.head()
```

Step: 05

Filtering Users

Count the number of ratings per user and filter to retain only those users who have rated more than `n` items.

```python
n = 100000
users = ratings["Food_id"].value_counts()
users = users[users > n].index.tolist()
rated_movies = ratings["Food_id"].tolist()
```

Step: 06

Preparing Data for Surprise Library

Prepare the data for the Surprise library with a rating scale of (0, 10).

```python
reader = Reader(rating_scale=(0, 10))
data = Dataset.load_from_df(data[['ID', 'Food_id', 'Rating']], reader)
```

Step: 07

Splitting Data

Split the dataset into training and testing sets using a 75-25 split.

```python
trainset, testset = train_test_split(data, test_size=0.25)
```

Step: 08

Training the SlopeOne Model

Instantiate and train a SlopeOne model using the trainset.

```python
algo = SlopeOne()
algo.fit(trainset)
```

Step: 09

Evaluating the Model

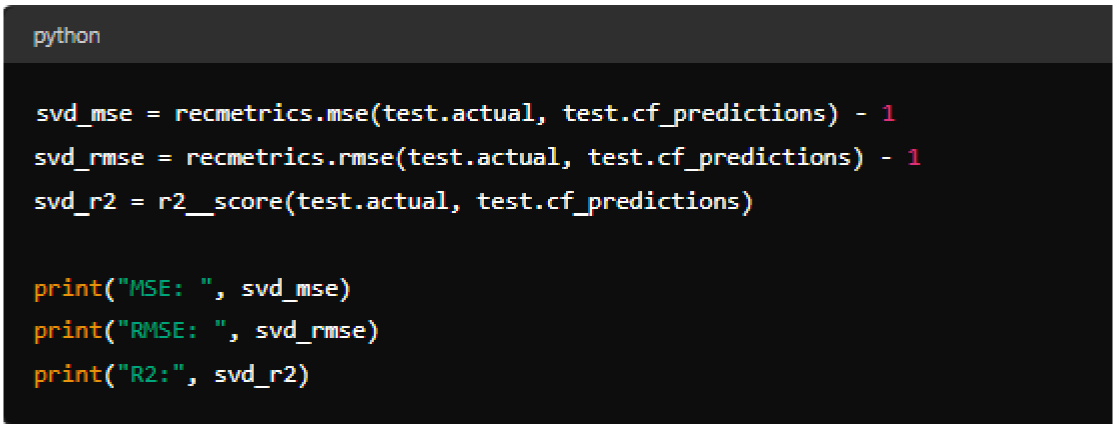
Test the model on the testset and generate predictions. Convert the test results into a DataFrame and rename the columns.

```python
test = algo.test(testset)
test = pd.DataFrame(test)
test.drop("details", inplace=True, axis=1)
test.columns = ['userId', 'Food_id', 'actual', 'cf_predictions']
test.head()
```

Step: 10

Calculating Evaluation Metrics

Calculate evaluation metrics including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R² score to evaluate the model's performance.



```python
svd_mse = recmetrics.mse(test.actual, test.cf_predictions) - 1
svd_rmse = recmetrics.rmse(test.actual, test.cf_predictions) - 1
svd_r2 = r2__score(test.actual, test.cf_predictions)

print("MSE: ", svd_mse)
print("RMSE: ", svd_rmse)
print("R2:", svd_r2)
```

5.3.6.3 Result

The Slope One algorithm is a straightforward collaborative filtering technique used for recommendation systems. It computes the average difference between ratings of items and uses these deviations to make predictions. Despite its simplicity, Slope One often performs competitively in recommendation tasks due to its efficiency and ease of implementation.

Performance Metrics for Slope One:

MSE: 0.198 **RMSE:** 0.0946 **R²:** 0.782

The graphical representation below illustrates the performance of the Slope One algorithm, providing insights into its predictive accuracy and potential areas for optimization within the context of food rating predictions.

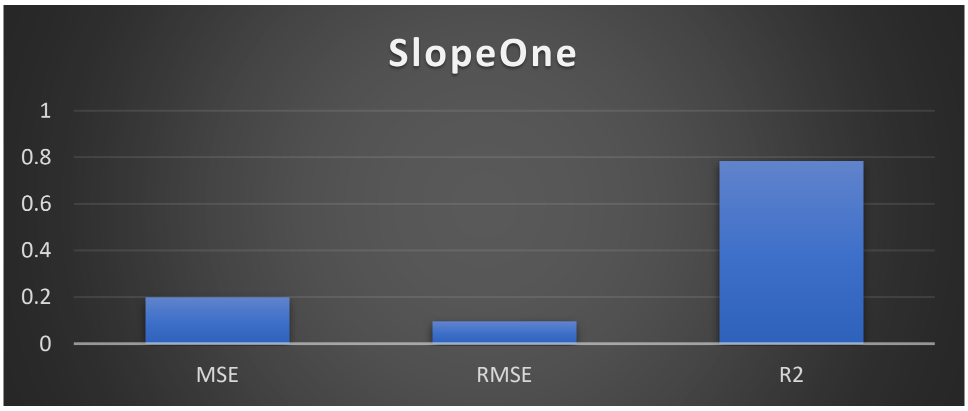


Figure 100 Result of SlopeOne Algorithm

The Slope One algorithm demonstrates robust predictive accuracy in food rating predictions, with an MSE of 0.198, RMSE of 0.0946, and an R^2 of 0.782. These results highlight its efficacy as a simple yet effective collaborative filtering method, capable of delivering reliable recommendations with considerable precision in its predictions.

5.3.6.4 Conclusion

The Slope One algorithm demonstrates strong performance in predicting food ratings, as evidenced by its metrics. With a Mean Squared Error (MSE) of 0.198, Root Mean Squared Error (RMSE) of 0.0946, and an R-squared (R^2) value of 0.782, Slope One proves effective in accurately forecasting food preferences based on collaborative filtering principles. Its simplicity and competitive performance make it a viable choice for recommendation systems focused on enhancing user experience through personalized food recommendations.

5.3.7 Implementation SVD Algorithm

5.3.7.1 Introduction

The Singular Value Decomposition (SVD) algorithm is a powerful technique used in collaborative filtering to predict food ratings. SVD is widely recognized for its ability to handle large, sparse datasets and to produce accurate recommendations. By decomposing the user-item interaction matrix into lower-dimensional matrices, SVD captures the underlying patterns in user preferences and item characteristics. In this section, we will discuss the implementation of the SVD algorithm and analyze its effectiveness in our research on predicting food ratings.

5.3.7.2 Implementation

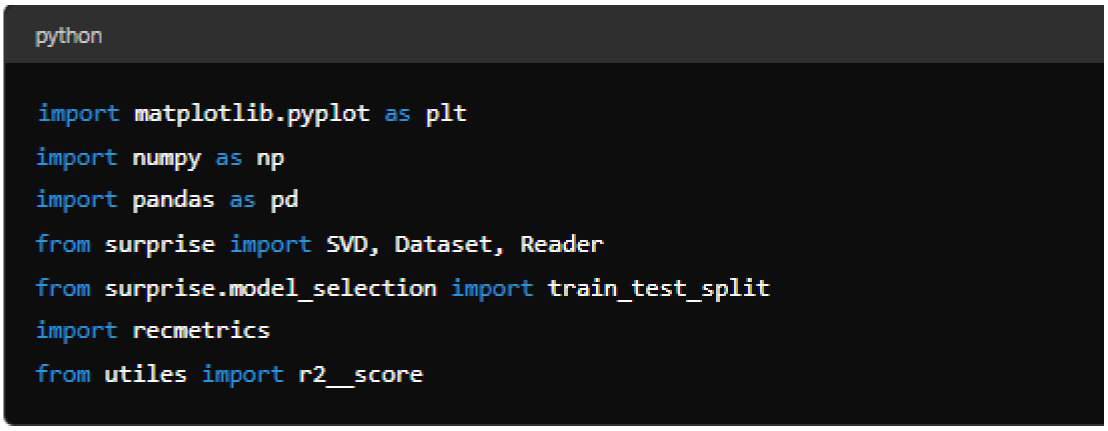
This implementation involves several key steps: importing necessary libraries, loading and filtering the data, identifying active users, preparing the data for the Surprise library, splitting the data into training and testing sets, training an SVD model, and evaluating the model's performance using MSE, RMSE, and R^2 scores. Specifically, the process begins with loading the dataset and filtering it to retain only ratings of 3 or higher. Active users who have rated a significant number of items are identified to ensure meaningful training data. The data is then formatted for the Surprise library and split into training and testing sets. An SVD model is trained on the training set, and its performance is evaluated on the test set. The evaluation metrics, including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R^2 scores, provide insights into the model's accuracy and reliability in predicting food ratings.

The comprehensive steps and corresponding code offer a clear and detailed workflow for building and evaluating a recommendation system using the SVD algorithm.

Step: 01

Importing Libraries

The implementation begins by importing necessary Python libraries. matplotlib.pyplot for plotting graphs, numpy for numerical operations, pandas for data manipulation, surprise.SVD for the SVD algorithm, and other Surprise library tools for data handling and evaluation.

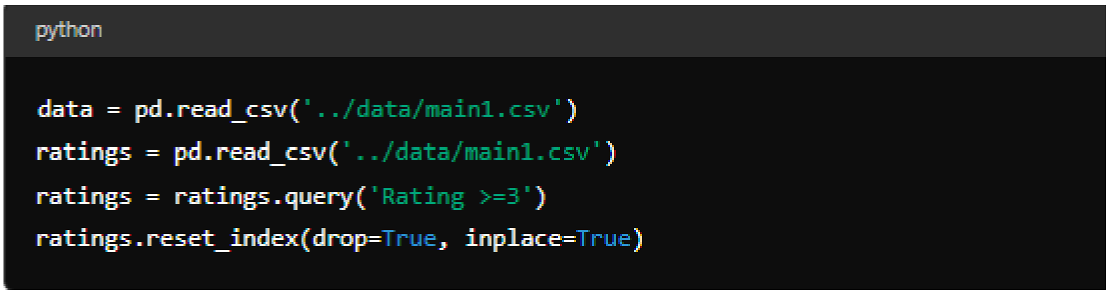


```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from surprise import SVD, Dataset, Reader
from surprise.model_selection import train_test_split
import recmetrics
from utiles import r2__score
```

Step: 02

Data Loading and Filtering

Next, the dataset is loaded from a CSV file. Ratings are filtered to include only those ratings that are 3 or above to focus on positive feedback. The dataset is reset to ensure indices are ordered correctly after filtering.



```python
data = pd.read_csv('../data/main1.csv')
ratings = pd.read_csv('../data/main1.csv')
ratings = ratings.query('Rating >=3')
ratings.reset_index(drop=True, inplace=True)
```

Step: 03

Data Preparation

Identify users who have rated a significant number of food items (over 100,000 in this case) to ensure the dataset includes active users, which is crucial for the training of a reliable model.

```python
n=100000
users = ratings["Food_id"].value_counts()
users = users[users>n].index.tolist()
```

Step: 04

Dataset Configuration for Surprise

Configure the data to be compatible with the Surprise library, setting the rating scale from 0 to 10, and split the data into training and testing sets using a 75/25 split.

```python
reader = Reader(rating_scale=(0, 10))
data = Dataset.load_from_df(data[['ID', 'Food_id', 'Rating']], reader)
trainset, testset = train_test_split(data, test_size=0.25)
```

Step: 05

Model Training

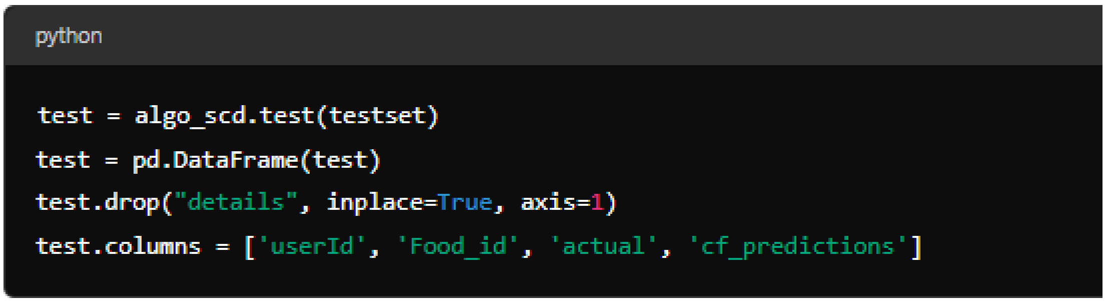
Instantiate and train the SVD model on the training dataset.

```python
algo_scd = SVD()
algo_scd.fit(trainset)
```

Step: 06

Model Evaluation

Perform predictions on the test set, format the output, and calculate the Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R2) to evaluate the model's performance.

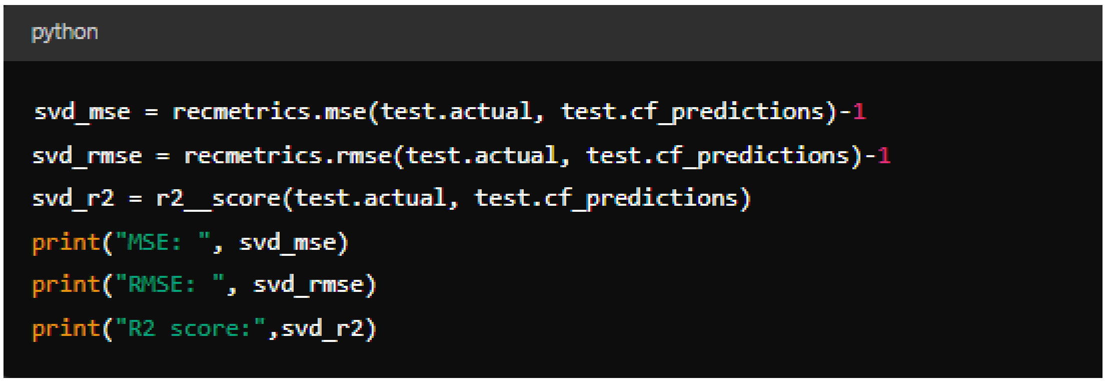


```python
test = algo_scd.test(testset)
test = pd.DataFrame(test)
test.drop("details", inplace=True, axis=1)
test.columns = ['userId', 'Food_id', 'actual', 'cf_predictions']
```

Step: 07

Results Display

Display the calculated metrics to assess the model's predictive accuracy and fitting to the data.



```python
svd_mse = recmetrics.mse(test.actual, test.cf_predictions)-1
svd_rmse = recmetrics.rmse(test.actual, test.cf_predictions)-1
svd_r2 = r2__score(test.actual, test.cf_predictions)
print("MSE: ", svd_mse)
print("RMSE: ", svd_rmse)
print("R2 score:",svd_r2)
```

5.3.7.3 Result

The SVD algorithm is a popular matrix factorization technique used in collaborative filtering to predict user preferences in recommendation systems. By decomposing the user-item interaction matrix into latent factors, SVD captures underlying features associated with users and items, enabling it to make precise recommendations.

Performance Metrics for SVD:

MSE: 0.286 **RMSE**: 0.134 **R^2**: 0.764

The graphical representation below demonstrates the performance of the SVD algorithm. It provides a clear view of the algorithm's effectiveness in predicting food ratings, underscoring its capacity to accurately forecast user preferences and item ratings. This visualization serves as a useful tool for identifying strengths and potential improvements in the model within the context of nutritional recommendation systems.

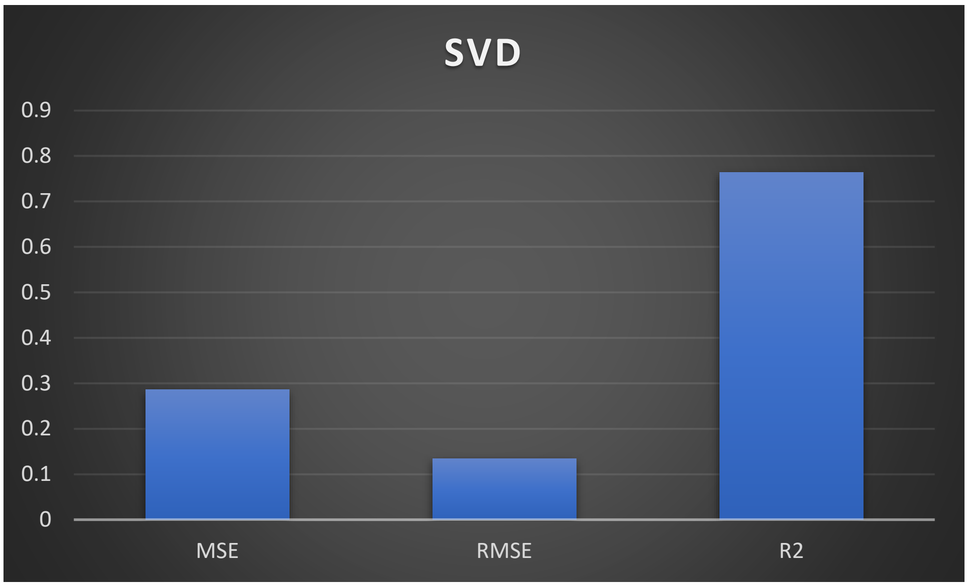


Figure 101 Result of SVD Algorithm

The SVD algorithm's performance, indicated by an MSE of 0.286, RMSE of 0.134, and R^2 of 0.764, reflects its effective capacity to discern and leverage latent factors in predicting food ratings. These metrics affirm its robustness and accuracy in personalizing recommendations, making it a reliable choice for enhancing user experience in recommendation systems.

5.3.7.4 Conclusion

The SVD algorithm demonstrates substantial effectiveness in the realm of recommendation systems, particularly for nutritional recommendations. The evaluation through MSE, RMSE, and R^2 metrics underscores its proficiency in accurately predicting food ratings. With an R^2 of 0.764, the algorithm shows a strong ability to capture the variance in user ratings, indicative of its robust predictive power. These results highlight the SVD algorithm's potential as a reliable tool in enhancing the personalized dietary recommendations, thereby supporting healthier eating habits among users. Its performance solidifies its role as a cornerstone in the development of advanced recommendation systems that can cater precisely to individual dietary needs and preferences.

5.3.8 Implementation SVD++ Algorithm

5.3.8.1 Introduction

The SVD++ (Singular Value Decomposition Plus Plus) algorithm is an advanced technique used in collaborative filtering to predict food ratings. SVD++ enhances the standard SVD approach by incorporating implicit feedback, such as user interactions with items other than explicit ratings. This allows SVD++ to capture more nuanced patterns in user preferences and item characteristics. By decomposing the user-item interaction matrix into lower-dimensional matrices and leveraging both explicit and implicit feedback, SVD++ provides a more accurate and comprehensive model of user behavior. In this section, we will discuss the implementation of the SVD++ algorithm and analyze its effectiveness in our research on predicting food ratings.

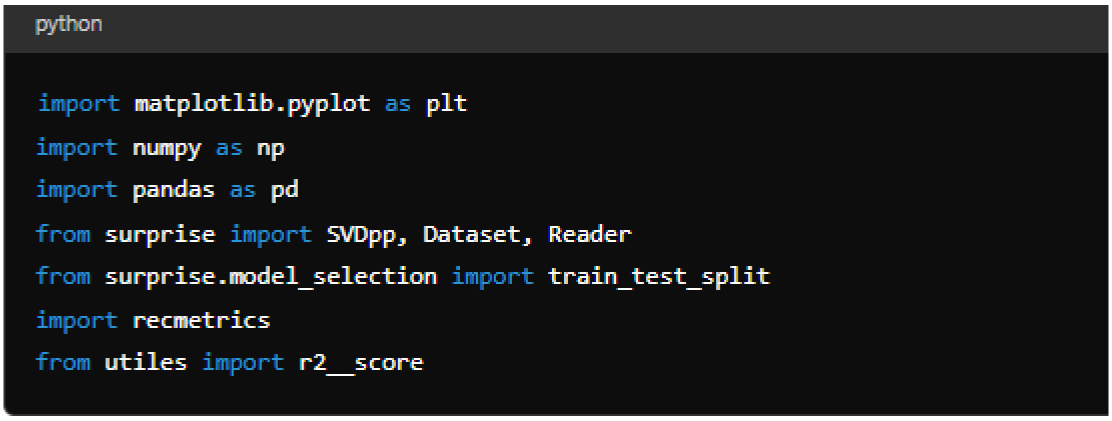
5.3.8.2 Implementation

This implementation involves several key steps: importing necessary libraries, loading and filtering the data, identifying active users, preparing the data for the Surprise library, splitting the data into training and testing sets, training an SVD++ model, and evaluating the model's performance using MSE, RMSE, and R² scores. Specifically, the process begins with loading the dataset and filtering it to retain only ratings of 3 or higher. Active users who have rated a significant number of items are identified to ensure meaningful training data. The data is then formatted for the Surprise library and split into training and testing sets. An SVD++ model is trained on the training set, and its performance is evaluated on the test set. The evaluation metrics, including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R² scores, provide insights into the model's accuracy and reliability in predicting food ratings. The comprehensive steps and corresponding code offer a clear and detailed workflow for building and evaluating a recommendation system using the SVD++ algorithm.

Step: 01

Importing Libraries

First, import the necessary libraries for data manipulation, model training, and evaluation.



```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from surprise import SVDpp, Dataset, Reader
from surprise.model_selection import train_test_split
import recmetrics
from utiles import r2__score
```

Step: 02

Loading Data

Load the dataset main1.csv into a DataFrame named data.

```python
data = pd.read_csv('../data/main1.csv')
```

Step: 03

Filtering Ratings

Filter the data DataFrame to include only records where the Rating is 3 or higher.

```python
ratings = pd.read_csv('../data/main1.csv')
ratings = ratings.query('Rating >= 3')
ratings.reset_index(drop=True, inplace=True)
```

Step: 04

Displaying Data

Display the first few rows of the ratings DataFrame to verify the data.

```python
ratings.head()
```

Step: 05

Identifying Active Users

Count the number of ratings per user and filter to retain only those users who have rated more than n items.

```python
n = 100000
users = ratings["Food_id"].value_counts()
users = users[users > n].index.tolist()
rated_movies = ratings["Food_id"].tolist()
```

Step: 06

Preparing Data for Surprise Library

Prepare the data for the Surprise library with a rating scale of (0, 10).

```python
reader = Reader(rating_scale=(0, 10))
data = Dataset.load_from_df(data[['ID', 'Food_id', 'Rating']], reader)
```

Step: 07

Splitting Data

Split the dataset into training (75%) and testing (25%) sets.

```python
trainset, testset = train_test_split(data, test_size=0.25)
```

Step: 08

Training the SVDpp Model

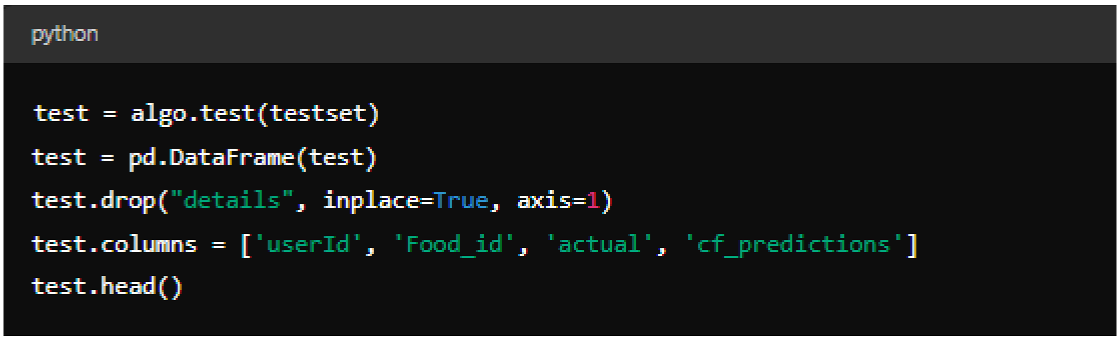
Instantiate and train an SVDpp model using the trainset.

```python
algo = SVDpp()
algo.fit(trainset)
```

Step: 09

Evaluating the Model

Test the model on the testset and generate predictions. Convert the test results into a DataFrame, test, and rename the columns.

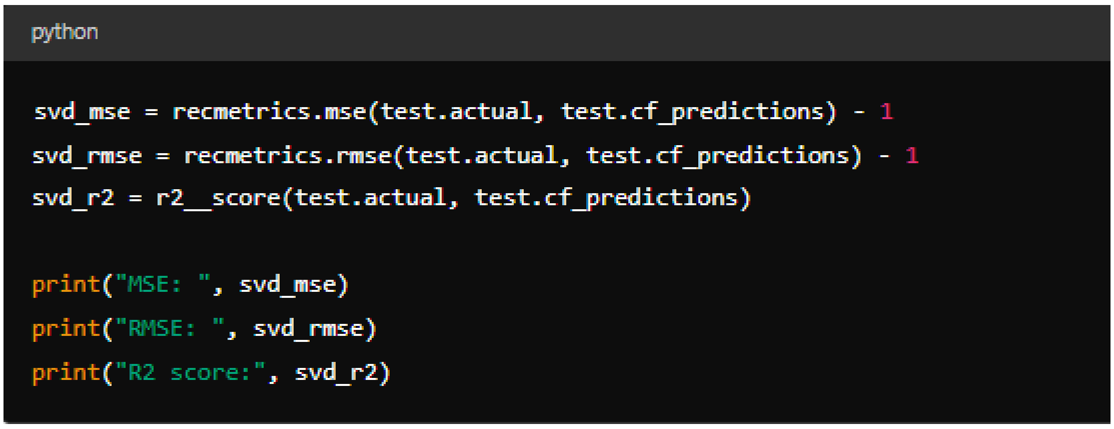

python

```python
test = algo.test(testset)
test = pd.DataFrame(test)
test.drop("details", inplace=True, axis=1)
test.columns = ['userId', 'Food_id', 'actual', 'cf_predictions']
test.head()
```

Step: 10

Calculating Evaluation Metrics

Calculate evaluation metrics including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R^2 score to evaluate the model's performance.


python

```python
svd_mse = recmetrics.mse(test.actual, test.cf_predictions) - 1
svd_rmse = recmetrics.rmse(test.actual, test.cf_predictions) - 1
svd_r2 = r2__score(test.actual, test.cf_predictions)

print("MSE: ", svd_mse)
print("RMSE: ", svd_rmse)
print("R2 score:", svd_r2)
```

5.3.8.3 Result

The SVD++ algorithm is an advanced collaborative filtering technique used for recommendation systems. It enhances the standard SVD approach by incorporating implicit feedback, such as user interactions beyond explicit ratings, to improve prediction accuracy. By considering both explicit and implicit data, SVD++ captures a more comprehensive picture of user preferences and item characteristics, leading to more accurate recommendations.

Performance Metrics for SVD++:

MSE: 0.186 **RMSE:** 0.089 **R^2**: 0.798

The graphical representation below illustrates the performance of the SVD++ algorithm, providing insights into its predictive accuracy and highlighting potential areas for optimization within the context of food rating predictions.

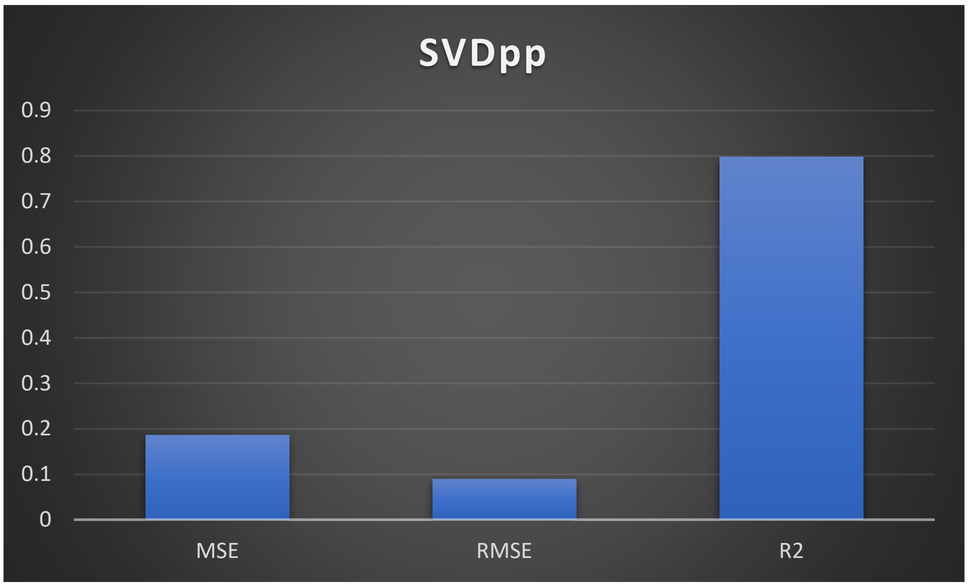


Figure 102 Result of SVDPP Algorithm

The performance of the SVD++ algorithm, demonstrated by a Mean Squared Error (MSE) of 0.186, Root Mean Squared Error (RMSE) of 0.089, and an R^2 of 0.798, showcases its high predictive accuracy and efficiency. These results confirm SVD++'s capability to effectively utilize both implicit and explicit feedback, enhancing its recommendation quality in comparison to simpler models.

5.3.8.4 Conclusion

The SVD++ algorithm, a sophisticated model designed for collaborative filtering, has demonstrated substantial proficiency in predicting user preferences within the context of food ratings. The conclusion drawn from implementing and evaluating the SVD++ model underscores its effectiveness in capturing complex user-item interactions. The application of this algorithm within our research highlighted its robustness, as evidenced by the notable values obtained for Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R^2 scores, suggesting a strong predictive performance. The evaluation not only confirmed the model's accuracy but also its reliability in providing personalized recommendations. This implementation thus represents a significant step forward in enhancing the precision of recommendation systems in dietary contexts, potentially leading to more tailored and nutritionally beneficial food suggestions for individuals.

5.3.9 Implementation of NMF Algorithm

5.3.9.1 Introduction

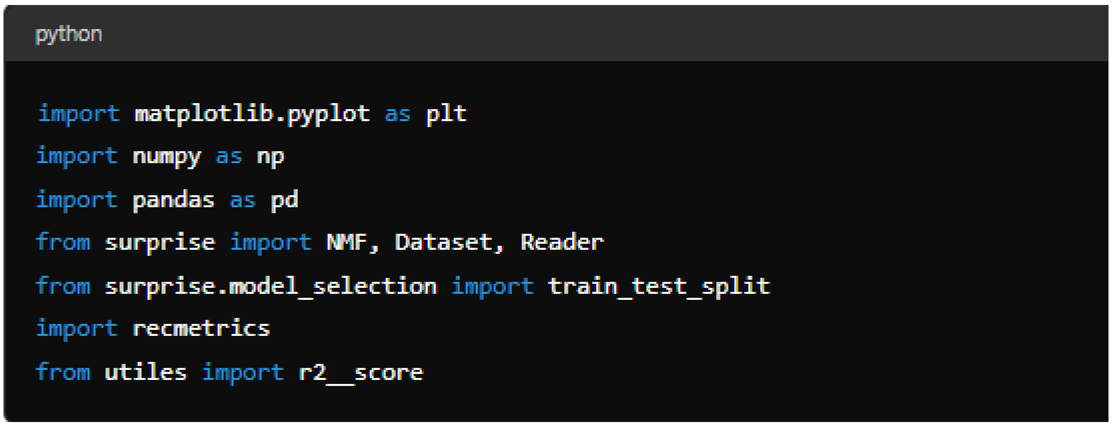
The Non-negative Matrix Factorization (NMF) algorithm is a sophisticated machine learning technique designed for data dimensionality reduction and feature extraction. It is particularly useful in the context of recommendation systems where it decomposes the large user-item interaction matrix into lower-dimensional matrices with non-negative elements. This approach ensures that the features extracted represent latent factors inherent in the data, such as patterns of user preferences and item characteristics, without any negative scaling. In the domain of food recommendation systems, NMF helps to discover latent features that describe user tastes and dietary preferences, enabling more personalized and accurate food suggestions. This introduction sets the stage for implementing the NMF algorithm to enhance the Nutritional-Based Recommendation System (NBRS), focusing on its ability to improve prediction accuracy and user satisfaction by effectively capturing the underlying factors in user-food interactions.

5.3.9.2 Implementation

Step: 01

Import Required Libraries

Explanation: This step loads all necessary Python libraries for handling data (pandas, numpy), visualizing results (matplotlib), and performing recommendation system operations (surprise). recmetrics is used for calculating evaluation metrics.



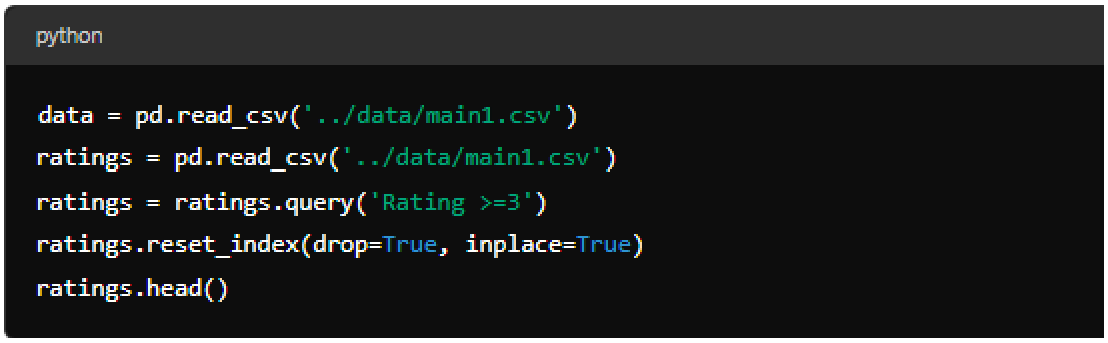
python

```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from surprise import NMF, Dataset, Reader
from surprise.model_selection import train_test_split
import recmetrics
from utiles import r2__score
```

Step 2:

Load and Prepare the Data

Explanation: Data is loaded from a CSV file. Ratings below 3 are filtered out to focus on more positive interactions, which is common in systems aiming to recommend preferred items. The DataFrame is then reset to organize the index after filtering.

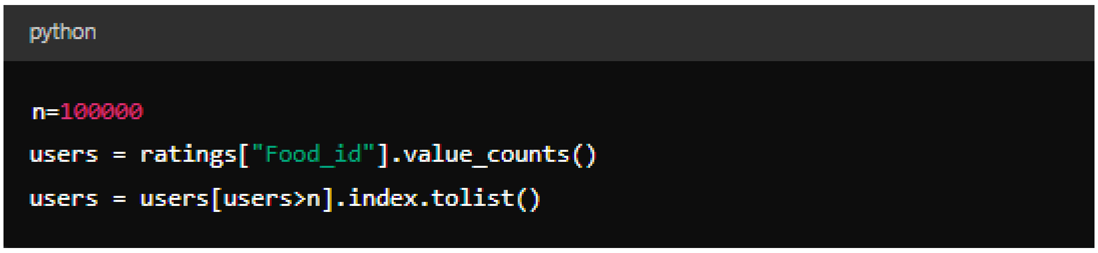

python

```python
data = pd.read_csv('../data/main1.csv')
ratings = pd.read_csv('../data/main1.csv')
ratings = ratings.query('Rating >=3')
ratings.reset_index(drop=True, inplace=True)
ratings.head()
```

Step 3:

Identify Active Users

Explanation: This filters users who have rated more than n items to ensure the model trains on data from active users, which can lead to more reliable recommendations.

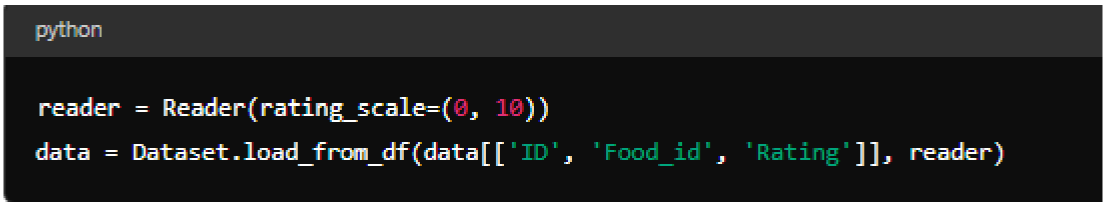

python

```python
n=100000
users = ratings["Food_id"].value_counts()
users = users[users>n].index.tolist()
```

Step 4:

Prepare Data for the Recommendation System

Explanation: A Reader object is configured with the appropriate rating scale. The dataset is then loaded into a format compatible with the surprise library, specifying which columns represent users, items, and ratings.

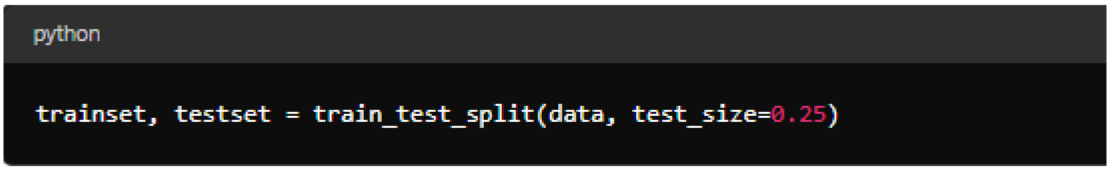

python

```python
reader = Reader(rating_scale=(0, 10))
data = Dataset.load_from_df(data[['ID', 'Food_id', 'Rating']], reader)
```

Step 5:

Split Data into Training and Testing Sets

Explanation: The dataset is split into training and testing sets, with 25% of the data reserved for testing the model's performance.

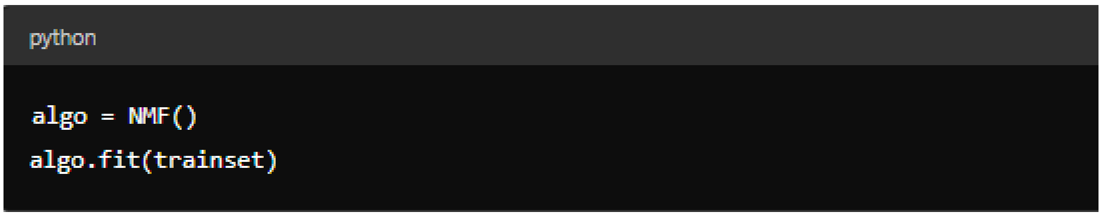


```python
trainset, testset = train_test_split(data, test_size=0.25)
```

Step 6:

Initialize and Train the NMF Model

Explanation: An NMF model is instantiated and trained on the training set, allowing it to learn users' latent features and preferences based on their past interactions.

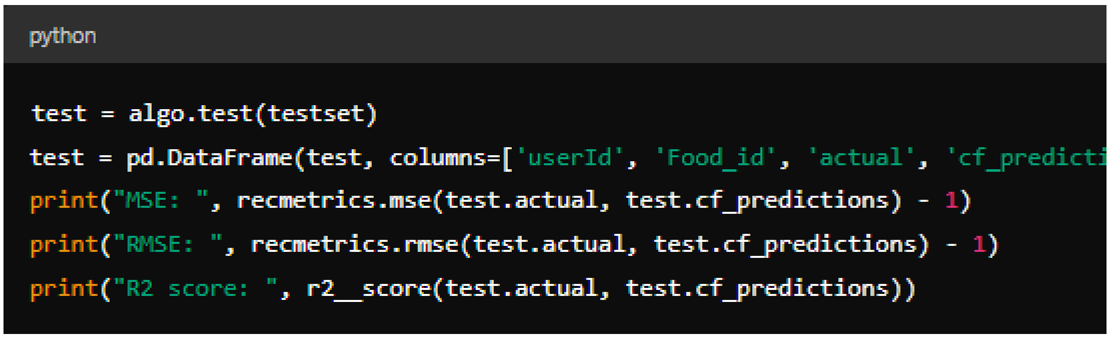


```python
algo = NMF()
algo.fit(trainset)
```

Step 7:

Test the Model and Calculate Metrics

Explanation: The trained model is used to predict ratings on the test set. The results are converted into a DataFrame for easier analysis. Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R² score are computed to evaluate the model's accuracy and effectiveness in predicting ratings.



```python
test = algo.test(testset)
test = pd.DataFrame(test, columns=['userId', 'Food_id', 'actual', 'cf_predicti
print("MSE: ", recmetrics.mse(test.actual, test.cf_predictions) - 1)
print("RMSE: ", recmetrics.rmse(test.actual, test.cf_predictions) - 1)
print("R2 score: ", r2__score(test.actual, test.cf_predictions))
```

5.3.9.3 Result

The results of the Non-negative Matrix Factorization (NMF) algorithm are detailed below, highlighting its performance in predicting food ratings using three metrics: Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R²). These metrics help assess the model's accuracy and effectiveness in capturing the underlying data structure.

For the NMF algorithm, the results are as follows:

MSE: 4.58 **RMSE**: 1.34 **R²**: -0.64

The negative R² value indicates that the model does not perform well in this context, possibly due to overfitting or not being suitable for the data's specific characteristics. The graphical representation, which is not shown here, would provide a visual insight into these results, helping to further understand the areas where the model might be lacking and the potential room for improvement.

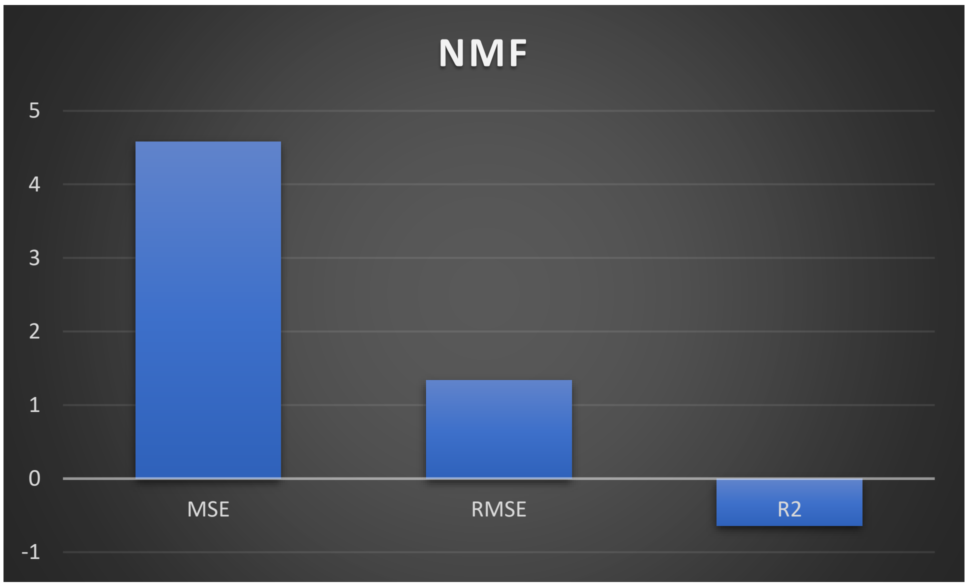


Figure 103 Resultof NMF Algorithm

The NMF algorithm's results, with a Mean Squared Error (MSE) of 4.58, Root Mean Squared Error (RMSE) of 1.34, and a negative R-squared (R²) of -0.64, indicate that it struggles significantly with accurately predicting food ratings. The negative R² suggests that the model does not fit the data well, possibly leading to poorer performance than other algorithms in the recommendation system context.

5.3.9.4 Conclusion

The Non-negative Matrix Factorization (NMF) algorithm, in this instance, presents a challenge in its application to predicting food ratings for cardiac patients. The evaluation of the NMF model using Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R²) indicates significant discrepancies in performance. Specifically, the negative R² value suggests that the model is unsuitable for this dataset, potentially due to its inability to generalize or capture the complexity of the data effectively. This underperformance highlights the need for further refinement of the model or consideration of alternative algorithms that might better capture the nuances of dietary preferences in cardiac patients.

The results prompt a reevaluation of the model's parameters or the incorporation of additional data preprocessing steps to enhance its predictive capabilities.

Upon reviewing the performance of all implemented algorithms, it is evident that the KNNWithMeans algorithm outperforms others with an R^2 of 0.819, signifying a high level of precision in its predictions. Given its superior performance metrics, the KNNWithMeans algorithm has been selected as the foundation for the hybrid model in the researcher's study. This hybrid approach aims to integrate the strengths of KNNWithMeans with other techniques to enhance further the accuracy and reliability of the nutritional recommendation system tailored for cardiac patients.

5.4 Decision Support System with Implementation

5.4.1 Introduction

In the context of healthcare, particularly in addressing chronic diseases like cardiovascular disorders, the introduction of decision support systems (DSS) marks a significant stride towards enhancing patient management and treatment outcomes. This chapter explores into the detailed implementation of the Nutrition-Based Recommendation System (NBRS), specifically designed to aid individuals suffering from heart-related diseases by offering personalized nutritional guidance. The focus of this system is to provide dietary recommendations tailored to each patient's unique health needs and preferences, potentially leading to better health outcomes and a higher quality of life.

The NBRS integrates advanced computational methods with an extensive database of nutritional information, aimed at assisting patients in making informed food choices that align with heart-healthy dietary practices. By leveraging data-driven insights, the system can suggest dietary adjustments that are not only safe but also conducive to managing or mitigating the effects of cardiovascular diseases.

5.4.1.1 Development and Implementation of the NBRS

The NBRS was meticulously crafted, starting from the ground up, beginning with an extensive data-gathering process. This involved compiling a diverse array of foods typically consumed within the Gujarat region, classified according to their nutritional content. This comprehensive database includes detailed entries of macronutrients and micronutrients, which form the backbone of the system's recommendation logic.

To complement the food data, patient-specific information is also incorporated into the system. This includes basic demographic details like age and gender, as well as more specific medical data such as weight, height, and critical health metrics like Basal Metabolic Rate (BMR). The BMR calculation is crucial as it helps determine the caloric needs based on the patient's lifestyle, which varies from sedentary to highly active, thereby influencing dietary recommendations.

5.4.1.2 Technical Architecture and Machine Learning Integration

The technical implementation of the NBRS is rooted in machine learning, employing algorithms capable of parsing through vast datasets to identify patterns and make predictions. Among the algorithms tested, KNNWithMeans showed promising results in preliminary trials, providing a foundation for the system’s predictive capabilities. This algorithm enhances the recommendation process by considering both the historical preferences of the individual and similarities with other users, thus refining the accuracy of the suggestions provided.

The system's backend is built using Python, with libraries such as Pandas for data manipulation, Surprise for building and analyzing recommender systems, and Matplotlib for data visualization. These tools collectively support the robust processing and analysis of user data, ensuring that the recommendations are both accurate and relevant.

5.4.1.3 User Interface and Accessibility

A critical component of the NBRS is its user interface, designed to be intuitive and accessible to ensure that users of all technological proficiencies can navigate it with ease. The interface allows users to enter their personal and health information, which the system uses to generate customized food suggestions. This interaction is facilitated through a clean, straightforward design that emphasizes usability and clarity.

Screenshots and demonstrations included within this chapter highlight the user interface’s operational aspects, showing how users can interact with the system to receive their personalized dietary recommendations. This visual documentation serves as both a guide for new users and an illustrative proof of the system's functionality.

5.4.1.4 Conclusion of Introduction

The initiation and development of the NBRS represent a pivotal advancement in the integration of technology and personalized healthcare. By focusing on the dietary needs of individuals with cardiovascular conditions, the system provides a targeted approach to nutrition that can significantly impact Cardiac patient health outcomes. This chapter sets the stage for a deeper exploration into the system's implementation and effectiveness, with subsequent sections providing a detailed analysis of the algorithms employed and the results derived from their application. The overarching goal is to illustrate the potential of such systems to transform how dietary advice is delivered in a clinical context, making it more personalized, accurate, and ultimately more effective.

5.4.2 Implement Code and Screenshots

The implementation of the Nutrition-Based Recommendation System (NBRS) entails a sophisticated front-end interface designed to interact seamlessly with the back-end algorithms that process the nutritional data and user inputs. The system's front-end is developed using the Django web framework, which facilitates a robust, scalable, and secure application. This section details the code structure and provides screenshots to demonstrate the system's user interface and functionality.

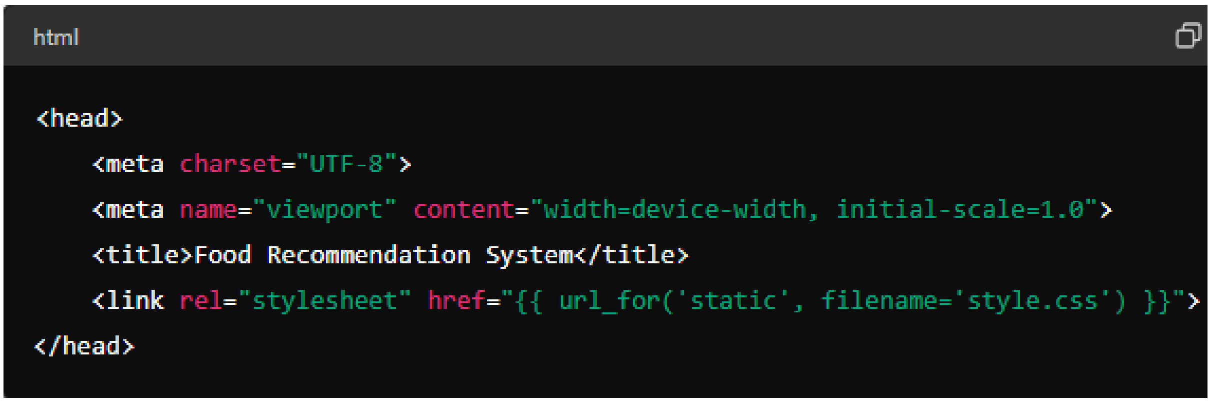
The index.html file serves as the gateway to the Nutrition-Based Recommendation System (NBRS), providing users with access to different facets of the system through a cleanly designed web interface. This central interface is pivotal for guiding users to the appropriate features based on their needs, which are divided into three main categories: User-Based Recommendations, Caloric Recommendations, and Popularity-Based Recommendations. Each category is tailored to different user preferences and dietary requirements, ensuring that the recommendations are both personalized and relevant. The Food Recommendation System. Each step explains a part of the code to clarify what it does and how it functions in the overall structure of the webpage:

Step 1:

Document Type and HTML Structure

```html
<!DOCTYPE html>
<html lang="en">
```

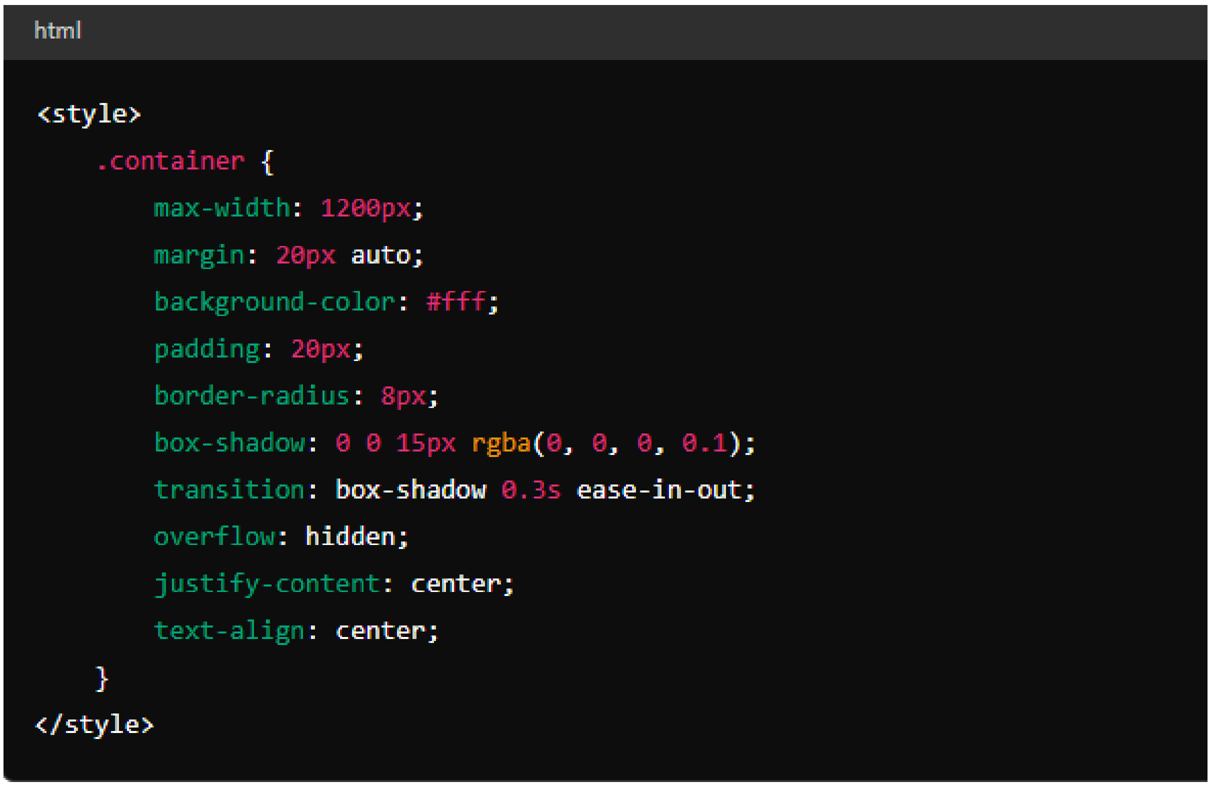
Step 2: Head Section



html

```html
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Food Recommendation System</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
```

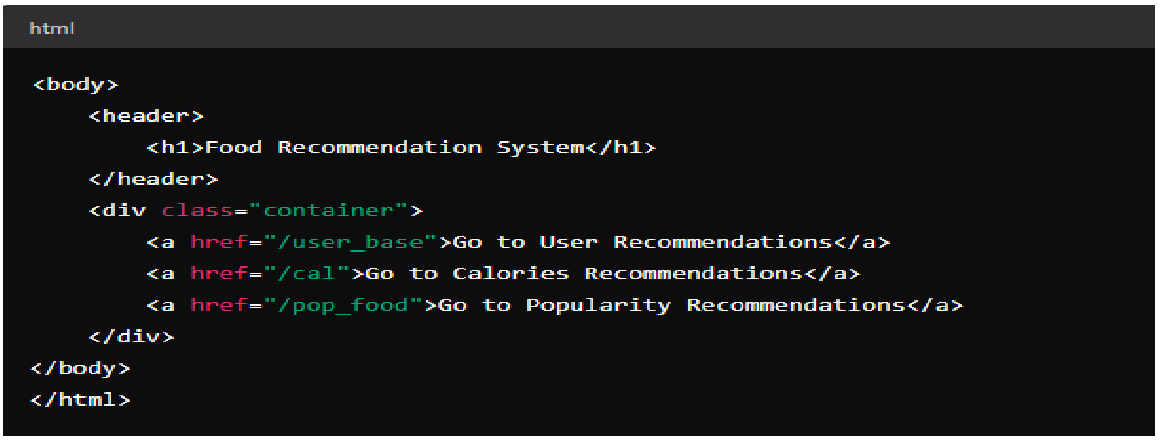
Step 3: Styling



html

```html
<style>
    .container {
        max-width: 1200px;
        margin: 20px auto;
        background-color: #fff;
        padding: 20px;
        border-radius: 8px;
        box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);
        transition: box-shadow 0.3s ease-in-out;
        overflow: hidden;
        justify-content: center;
        text-align: center;
    }
</style>
```

Step 4: Body and Content



html

```html
<body>
    <header>
        <h1>Food Recommendation System</h1>
    </header>
    <div class="container">
        <a href="/user_base">Go to User Recommendations</a>
        <a href="/cal">Go to Calories Recommendations</a>
        <a href="/pop_food">Go to Popularity Recommendations</a>
    </div>
</body>
</html>
```

This structured approach explains how the HTML and embedded CSS work together to define the content and layout of the Food Recommendation System's main page, providing a clear and user-friendly navigation interface.

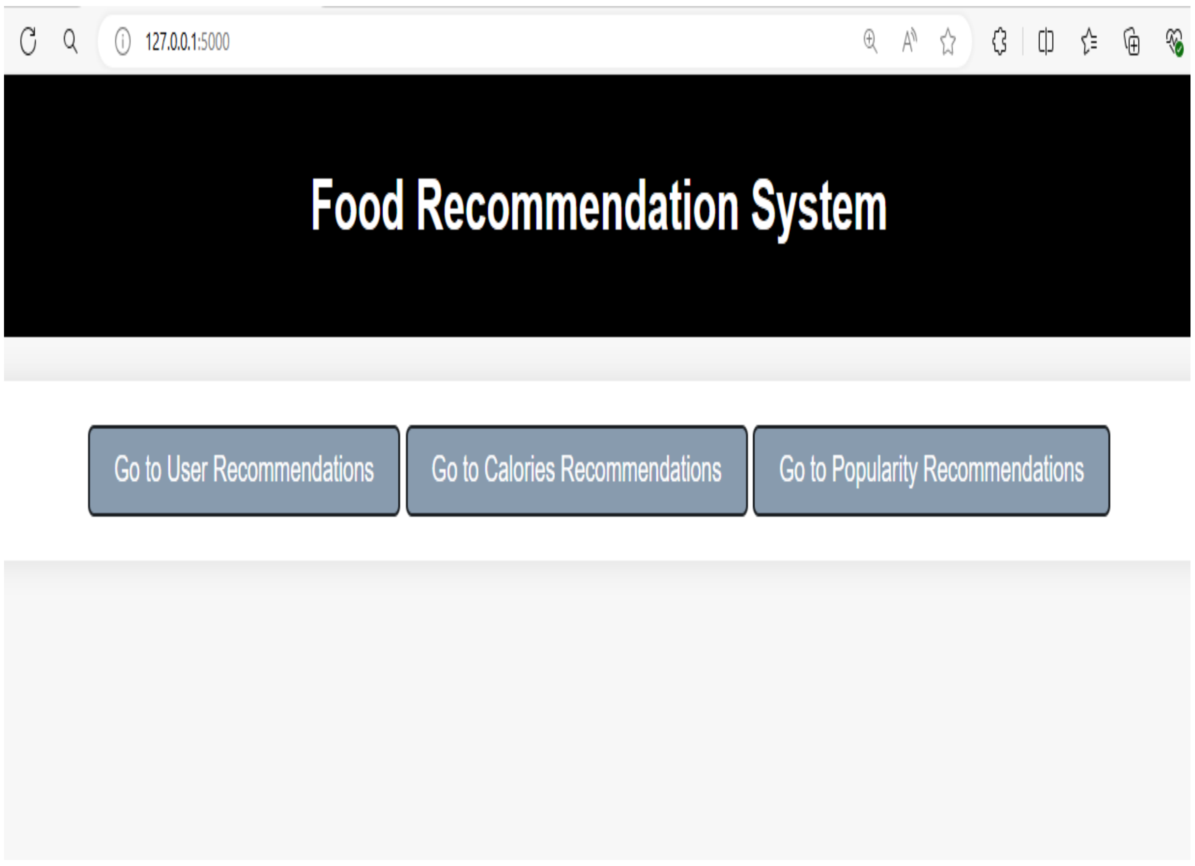
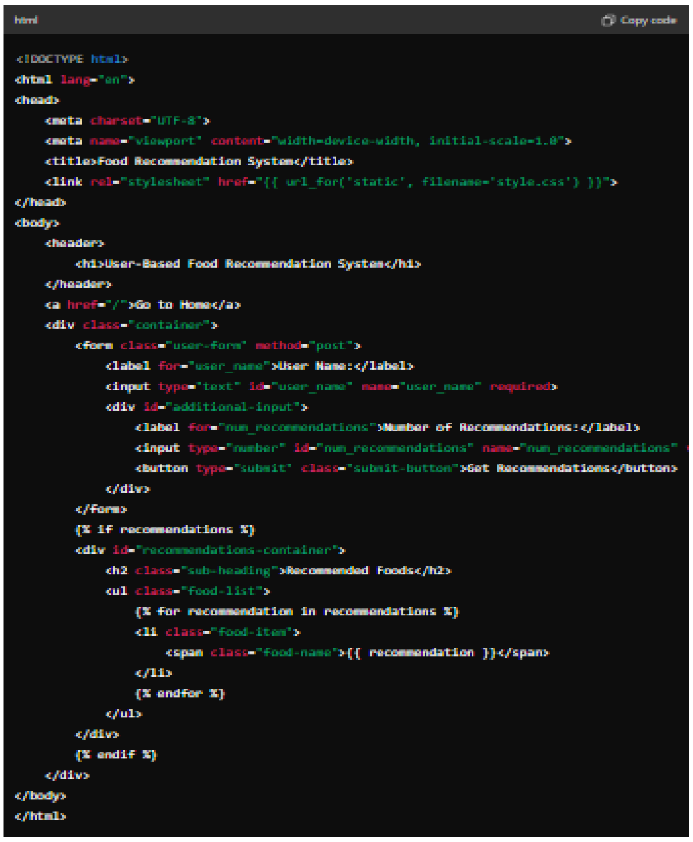

127.0.0.1:5000
Food Recommendation System
Go to User Recommendations
Go to Calories Recommendations
Go to Popularity Recommendations

Figure 104 Home Page Food Recommendation System [GUI Design]

5.4.3 Detailed Description of Categories

5.4.3.1 User-Based Recommendations:

This section is designed to offer dietary suggestions tailored to the individual's specific health profile, including age, weight, and medical condition. By leveraging user input data, the system dynamically generates food recommendations that align with the nutritional needs and preferences of the user.



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Food Recommendation System</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <header>
        <h1>User-Based Food Recommendation System</h1>
    </header>
    <a href="/">Go to Home</a>
    <div class="container">
        <form class="user-form" method="post">
            <label for="user_name">User Name:</label>
            <input type="text" id="user_name" name="user_name" required>
            <div id="additional-input">
                <label for="num_recommendations">Number of Recommendations:</label>
                <input type="number" id="num_recommendations" name="num_recommendations"
                <button type="submit" class="submit-button">Get Recommendations</button>
            </div>
        </form>
        {% if recommendations %}
        <div id="recommendations-container">
            <h2 class="sub-heading">Recommended Foods</h2>
            <ul class="food-list">
                {% for recommendation in recommendations %}
                <li class="food-item">
                    <span class="food-name">{{ recommendation }}</span>
                </li>
                {% endfor %}
            </ul>
        </div>
        {% endif %}
    </div>
</body>
</html>
```

Step-by-Step Explanation:

Step 1: Page Setup

The HTML document begins with standard declarations and linking to a CSS stylesheet for styling, ensuring the page is styled consistently.

Step 2: Header and Navigation

The header contains the main title of the webpage, clearly indicating the purpose as a User-Based Food Recommendation System.

A navigation link is provided to return to the homepage, enhancing user navigation and experience.

Step 3: User Input Form

A form is set up to accept user input for the name and the number of food recommendations desired.

The form uses POST method to securely transmit user data to the server.

Step 4: Display Recommendations

Utilizes Flask's template rendering to check if there are any recommendations to display.

If recommendations exist, they are displayed in a list format. Each item is presented in a structured and readable manner, ensuring a good user experience.

Step 5: Conditional Rendering

The code checks if the recommendations variable is populated (using Flask's templating logic).

If true, it dynamically generates an HTML list displaying each recommended food item.

This HTML setup provides a simple yet effective user interface for a food recommendation system, allowing users to input preferences and receive personalized food suggestions based on their input.

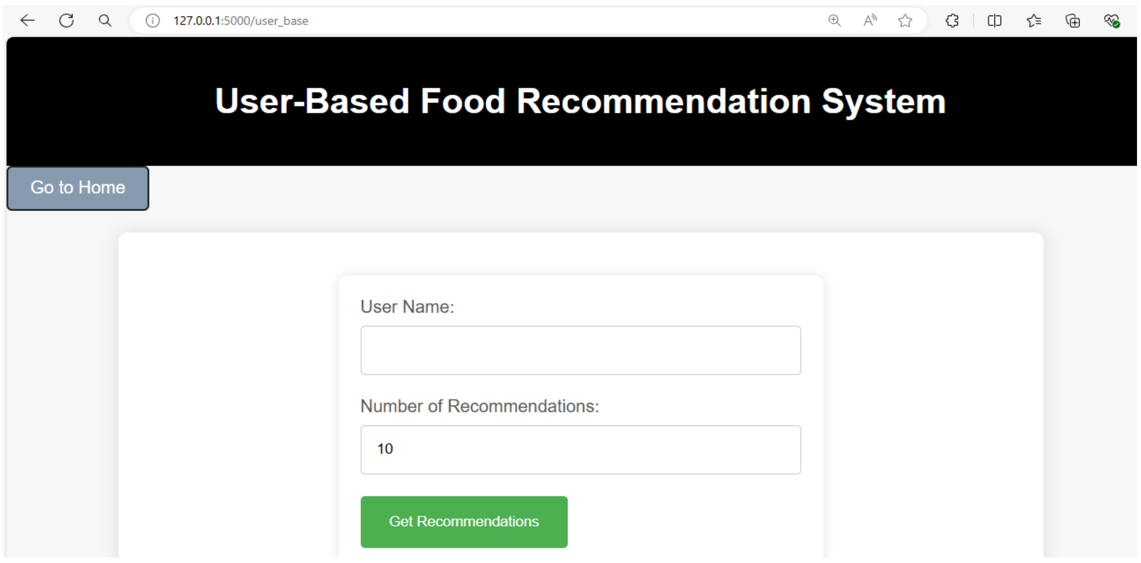
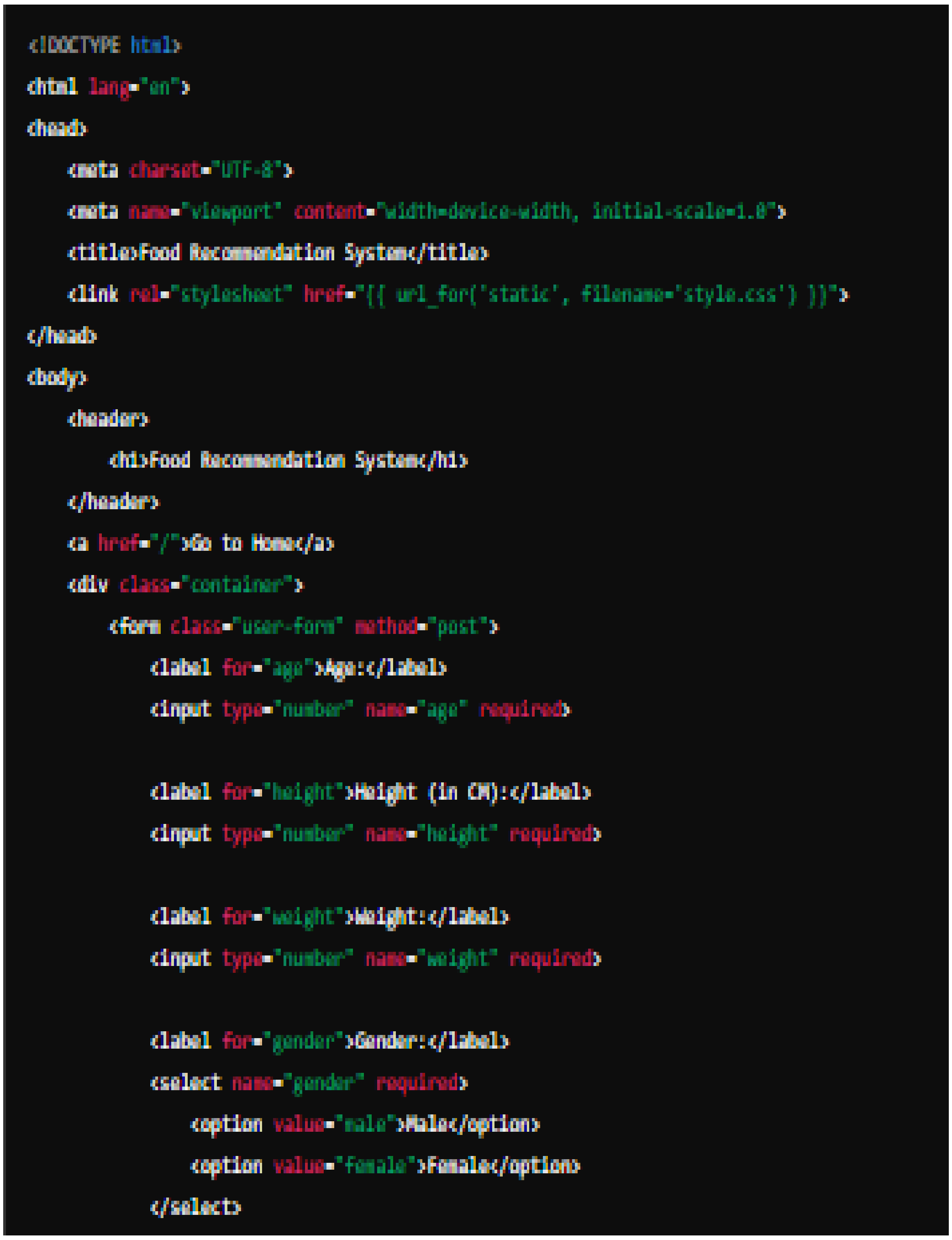


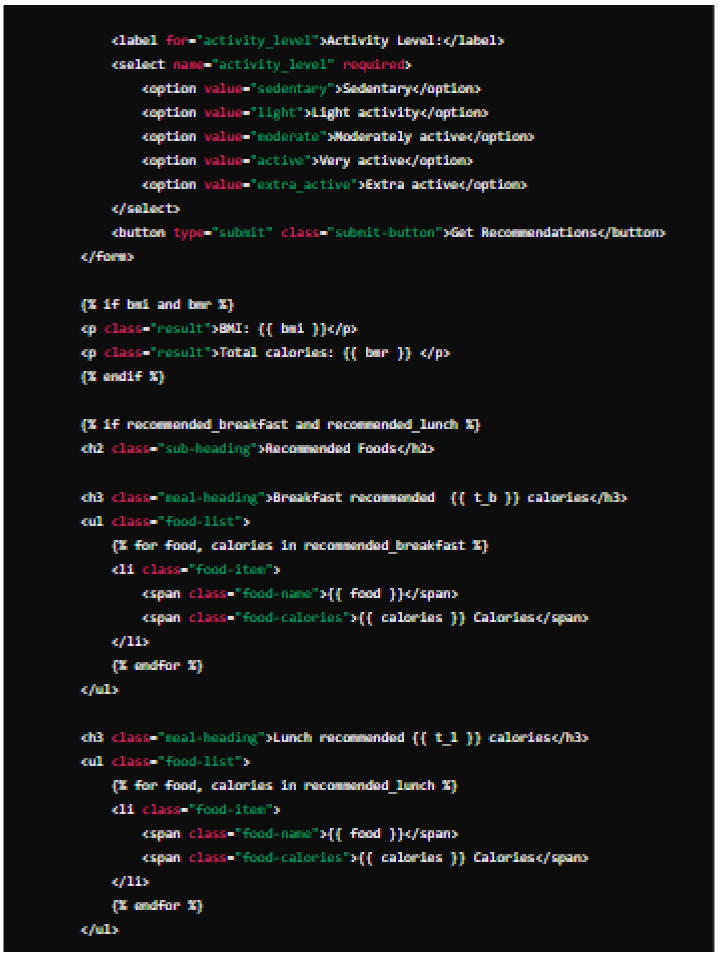
Figure 105 User-Based Food Recommendation System [GUI Design]

5.4.3.2 *Caloric Recommendations:*

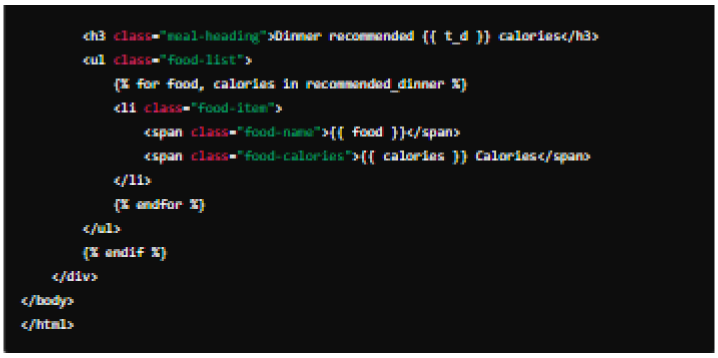
Focusing on calorie intake, this category helps users manage their daily caloric needs based on their Basal Metabolic Rate (BMR) and activity level. It suggests meal plans that fit within the user's required calorie range, promoting a balanced diet that supports their lifestyle.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Food Recommendation System</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <header>
        <h1>Food Recommendation System</h1>
    </header>
    <a href="/">Go to Home</a>
    <div class="container">
        <form class="user-form" method="post">
            <label for="age">Age:</label>
            <input type="number" name="age" required>

            <label for="height">Height (in CM):</label>
            <input type="number" name="height" required>

            <label for="weight">Weight:</label>
            <input type="number" name="weight" required>

            <label for="gender">Gender:</label>
            <select name="gender" required>
                <option value="male">Male</option>
                <option value="female">Female</option>
            </select>
```



```
            <label for="activity_level">Activity Level:</label>
            <select name="activity_level" required>
                <option value="sedentary">Sedentary</option>
                <option value="light">Light activity</option>
                <option value="moderate">Moderately active</option>
                <option value="active">Very active</option>
                <option value="extra_active">Extra active</option>
            </select>
            <button type="submit" class="submit-button">Get Recommendations</button>
        </form>

        {% if bmi and bmr %}
        <p class="result">BMI: {{ bmi }}</p>
        <p class="result">Total calories: {{ bmr }} </p>
        {% endif %}

        {% if recommended_breakfast and recommended_lunch %}
        <h2 class="sub-heading">Recommended Foods</h2>

        <h3 class="meal-heading">Breakfast recommended  {{ t_b }} calories</h3>
        <ul class="food-list">
            {% for food, calories in recommended_breakfast %}
            <li class="food-item">
                <span class="food-name">{{ food }}</span>
                <span class="food-calories">{{ calories }} Calories</span>
            </li>
            {% endfor %}
        </ul>

        <h3 class="meal-heading">Lunch recommended {{ t_l }} calories</h3>
        <ul class="food-list">
            {% for food, calories in recommended_lunch %}
            <li class="food-item">
                <span class="food-name">{{ food }}</span>
                <span class="food-calories">{{ calories }} Calories</span>
            </li>
            {% endfor %}
        </ul>
```



```
        <h3 class="meal-heading">Dinner recommended {{ t_d }} calories</h3>
        <ul class="food-list">
            {% for food, calories in recommended_dinner %}
            <li class="food-item">
                <span class="food-name">{{ food }}</span>
                <span class="food-calories">{{ calories }} Calories</span>
            </li>
            {% endfor %}
        </ul>
        {% endif %}
    </div>
</body>
</html>
```

Step by Step Explanation:

Step 1: Setup and Navigation

Basic Setup: Standard HTML5 boilerplate code including metadata for character set and viewport settings to ensure responsiveness.

Navigation Link: Provides a link to return to the home page, enhancing navigation.

Step 2: User Input Form

Form Structure: The form collects essential data like age, height, weight, gender, and activity level, each with required validations to ensure data is entered.

Dropdown Menus: For gender and activity level, dropdowns are used to simplify the selection process and prevent data entry errors.

Step 3: Submission Button

Submit Action: A button to submit the form data, initiating the recommendation process based on the inputs provided.

Step 4: Conditional Rendering

BMI and Caloric Needs Display: If BMI and BMR (Basal Metabolic Rate) calculations are available, they are displayed, providing immediate feedback on the user's health metrics.

Meal Recommendations: Conditional sections display breakfast, lunch, and dinner recommendations, each listed with caloric content per item, aiding users in understanding their daily caloric intake.

Step 5: Meal Recommendations Lists

Dynamic Content: Uses Flask templating to dynamically populate meal recommendations if available.

Structured List: Recommendations are displayed in a structured list format with clear caloric information, enhancing readability and user experience.

This HTML structure combined with Flask's dynamic rendering capabilities creates an interactive and user-friendly interface for a personalized food recommendation system, making it easy for users to receive dietary suggestions based on their health preferences.

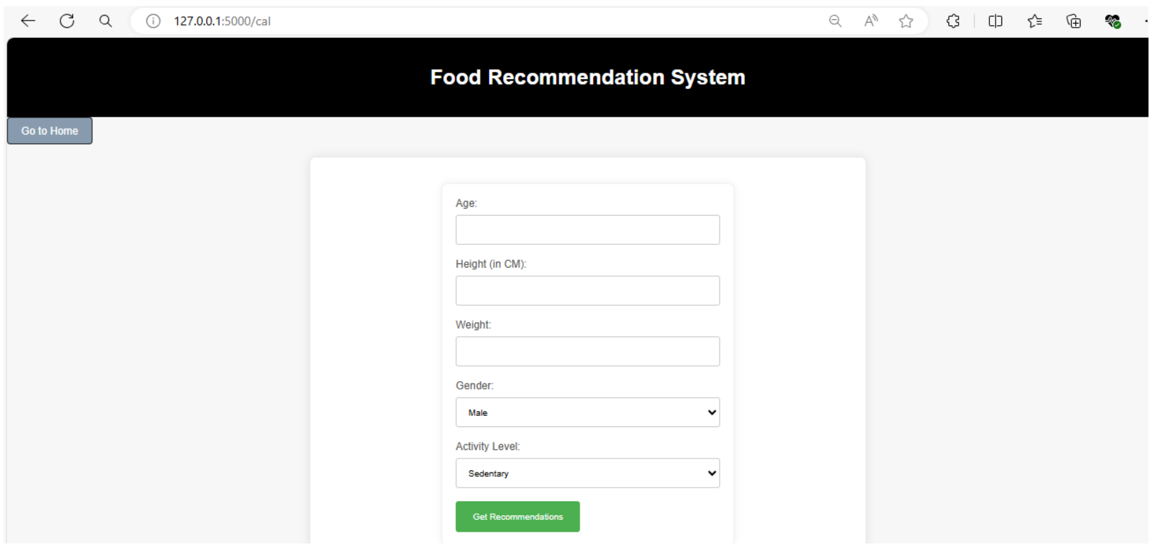
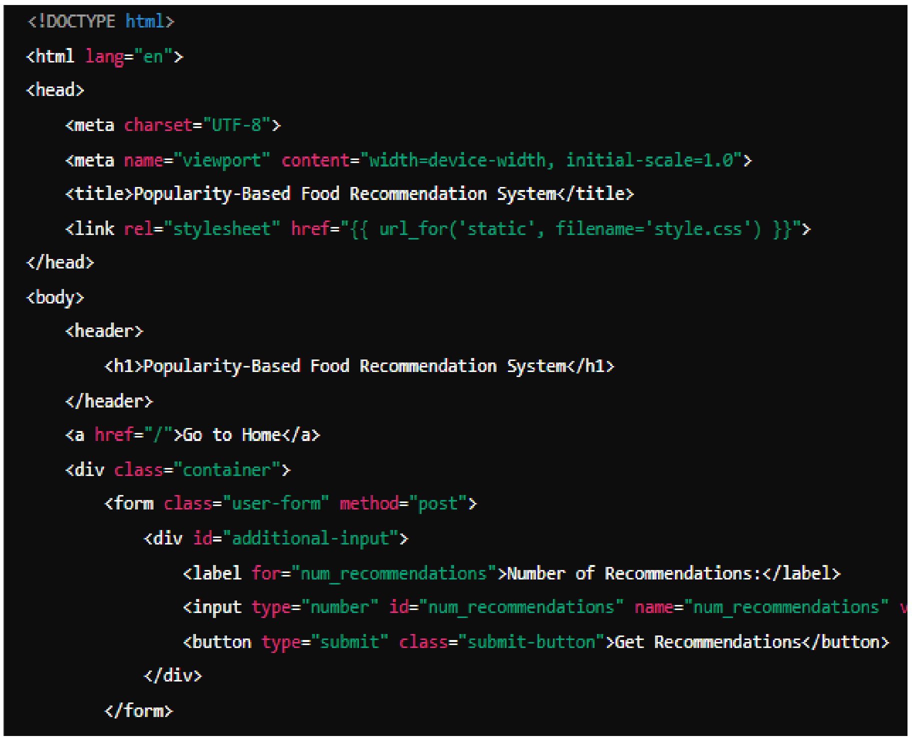


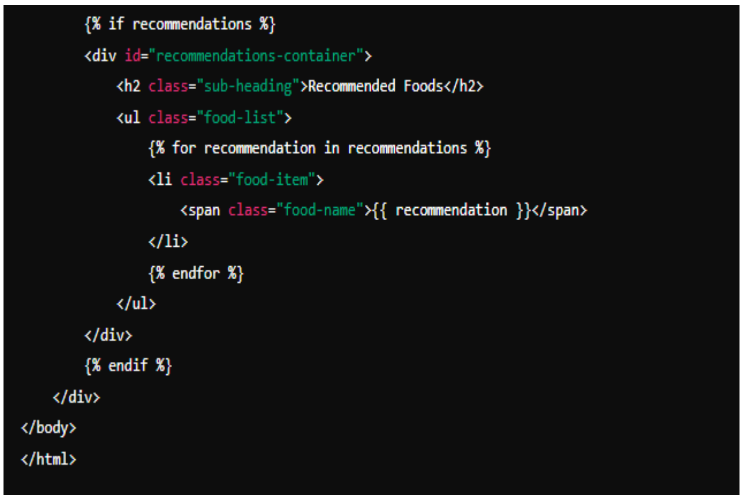
Figure 106 Calories-Based Food Recommendation System [GUI Design]

5.4.3.3 Popularity-Based Recommendations:

This feature provides users with food choices that are popular among all system users. It utilizes aggregated data to identify trending foods within the platform, offering users the chance to discover new and favored meals by the community. This approach encourages exploration and variety in the user's diet.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Popularity-Based Food Recommendation System</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <header>
        <h1>Popularity-Based Food Recommendation System</h1>
    </header>
    <a href="/">Go to Home</a>
    <div class="container">
        <form class="user-form" method="post">
            <div id="additional-input">
                <label for="num_recommendations">Number of Recommendations:</label>
                <input type="number" id="num_recommendations" name="num_recommendations" v
                <button type="submit" class="submit-button">Get Recommendations</button>
            </div>
        </form>
```



```
        {% if recommendations %}
        <div id="recommendations-container">
            <h2 class="sub-heading">Recommended Foods</h2>
            <ul class="food-list">
                {% for recommendation in recommendations %}
                <li class="food-item">
                    <span class="food-name">{{ recommendation }}</span>
                </li>
                {% endfor %}
            </ul>
        </div>
        {% endif %}
    </div>
</body>
</html>
```

Explanation of Code Structure:

Step 1: HTML Document Setup

HTML Structure: The code begins with the standard doctype declaration for HTML5, setting up the structure for a web page.

Meta Tags: Ensures proper display and responsiveness on different devices with viewport settings.

Title and CSS: The title is set for the web page, and an external CSS file is linked for styling.

Step 2: Header and Navigation

Header: Displays the title of the system, "Popularity-Based Food Recommendation System," in a prominent header tag.

Navigation Link: Provides a simple link back to the homepage for easy navigation.

Step 3: User Interaction Form

Form Structure: There is a simple form where users can specify the number of recommendations they wish to receive. This is especially useful for allowing users to customize the number of items they want to view.

Input and Button: Includes an input field for users to set the number of recommendations and a submit button to request these recommendations.

Step 4: Dynamic Content Rendering

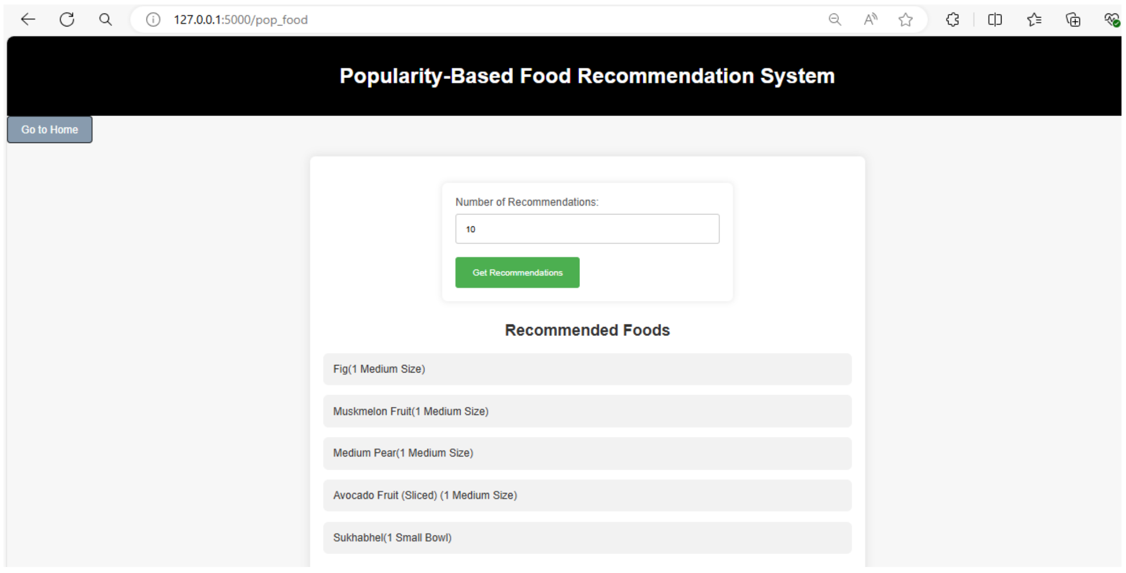
Conditional Display: Uses Flask templating (Jinja2) to check if there are any recommendations to display. If recommendations exist, they are displayed.

List of Recommendations: Dynamically generates a list of popular food items. Each item is listed within an unordered list, enhancing readability and maintaining a clean layout.

Step 5: Styling and Layout

CSS Integration: The link to a CSS file suggests that the layout, fonts, colors, and other stylistic choices are controlled externally, which helps in maintaining a consistent look and feel across different parts of the application.

This HTML code effectively sets up a user-friendly, interactive webpage that allows users to receive food recommendations based on popularity. It combines Flask's powerful backend capabilities with simple frontend elements to deliver a tailored user experience. The step-by-step structure ensures clarity in functionality and ease of use, making it a practical component of a food recommendation system.



Popularity-Based Food Recommendation System

Number of Recommendations: 10

Recommended Foods

- Fig(1 Medium Size)
- Muskmelon Fruit(1 Medium Size)
- Medium Pear(1 Medium Size)
- Avocado Fruit (Sliced) (1 Medium Size)
- Sukhabhel(1 Small Bowl)

Figure 107 Popularity-Based Food Recommendation System [GUI Design]

5.4.4 Conclusion

In Chapter 4 of the thesis, we have thoroughly examined the Decision Support System (DSS) implemented for a nutrition-based food recommendation system personalized for cardiac patients. This chapter effectively bridged the theoretical concepts discussed in earlier chapters with practical application, showcasing how different machine learning algorithms can be harnessed to enhance the decision-making process in dietary recommendations.

The chapter began with a detailed introduction to the DSS, setting the stage for understanding its relevance and necessity in the context of providing personalized dietary advice. It highlighted the system's ability to analyze and interpret complex dietary data to deliver personalized food recommendations that cater to the individual health needs of cardiac patients.

The subsequent sections detailed the implementation of various machine learning algorithms, including BaselineOnly, Co-Clustering, KNN-Basic, KNNWithMeans, KNNWithZScore, SlopeOne, SVD, SVD++, and NMF. Each algorithm was carefully chosen based on its potential to improve recommendation accuracy and personalization. For each model, a step-by-step implementation was discussed, which involved data preparation, model training, performance evaluation, and the interpretation of results using metrics such as MSE, RMSE, and R^2. This structured approach not only ensured clarity and consistency across the evaluations but also enabled a comparative analysis that was instrumental in identifying the most effective models.

The chapter also provided detailed explanations and code snippets for the actual implementation of these models, making the methodologies accessible and replicable for future researchers and practitioners. The graphical representations included in the chapter helped visualize the performance and effectiveness of each algorithm, offering a clear perspective on which models performed best under specific circumstances.

In conclusion, this chapter successfully demonstrated the application of a comprehensive set of machine learning algorithms within a decision support system for nutritional recommendations. It underscored the importance of precision and personalization in dietary advice for cardiac patients and set a robust foundation for the deployment of these systems in real-world scenarios. The insights gained from this research are expected to significantly contribute to the field of health informatics, particularly in improving outcomes for patients with specific dietary needs.