National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Ritik Jain
Student ID: x22119469

School of Computing
National College of Ireland

Supervisor: Mr. Hicham Rifai

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: …Ritik Jain……

Student ID: …x22119469……..……

Programme: …MSc in Data Analytics…… **Year:** ……2023……..

Module: …Research Project……..……

Lecturer: …Mr. Hicham Rifai……..……

Submission Due Date: …31/01/24……..……

Project Title: …Exercise and Diet Recommendation system using Machine Learning Techniques…… ……..……

Word Count: ……1538…… **Page Count:**……19……..……..……

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……Ritik Jain……

Date: ……31/01/24……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Ritik Jain
Student ID: x22119469

1 Introduction:

This configuration manual describes the specification of all the software and hardware used in our research project. Each of the hardware and software versions, rating, platform and other technical information is provided in this manual.

2 System Configuration:

2.1 Hardware configuration:

Personal Windows machine is used for all the implementation of the research. Specification of the machine is mentioned in the table below.

System Confirguration	
Instance	Windows 10 Pro
RAM	16 GB DDR4
CPU	AMD RYZEN 4600
Harddisk	500 GB SSD
CORE	8

Fig.1 hardware table

2.2 Software configuration:

Different softwares was used in this research all of these are listed in below table along with the versions.

System Confirguration	
Python	3.12
Anaconda	3.06.01
Excel	2016
MS word	2016

Fig. 2 software table

Anaconda and Jypter notebook:

Jupyter notebook is a free web-based application available in Anaconda 3 which enables you to write documents that include code, figures, equations and tests. Whole implementation was done in the jupyter notebook including all visualizations, model building, model training, test and evaluation.

Python Libraries:

Several python libraries were used in the research for various tasks. These library information is given in below table.

Library	version	Library	version
Matplotlib	3.04.03	scipy	1.07.03
Seaborn	0.11.02	sklearn	0.24.02
Numpy	1.21.02	Surprise	0.1
Pandas	1.03.03	Transformer	4.11.03
Torch	1.10.00	open AI	1.04.00
Tensorflow	2.06.00		

Fig. 3

3 Research Implementation Code:

3.1 Dataset Loading:

```
In [1]: import pandas as pd

        # Loading Exercise Dataset
        exercise_df = pd.read_csv('exercise_dataset.csv')
        print("Exercise Dataset:")

        exercise_df.head()

        Exercise Dataset:
```

Fig. 4 dataset loading

```
# Explore Exercise Dataset
print("\nExercise Dataset Info:")
print(exercise_df.info())  # Display information about the dataset

# Check for Missing Values
print("\nMissing Values in Exercise Dataset:")
print(exercise_df.isnull().sum())
```

Fig. 5 Dataset Exploration

Fig 4 and Fig 5 shows the python code for dataset loading and dataset exploration. Checking for null values.

```
# Drop the "Rating" and "RatingDesc" columns
exercise_df.drop(['Desc'], axis=1, inplace=True)
exercise_df.drop(['Type'], axis=1, inplace=True)
```

```
# Remove columns with names starting with 'Unnamed'
exercise_df = exercise_df.loc[:, ~exercise_df.columns.str.startswith('Unnamed')]

# Display the DataFrame after removing unnamed columns
print(exercise_df)
```

```
# Handle missing values in the exercise dataset


exercise_df = exercise_df.dropna(subset=['Equipment'])
exercise_df = exercise_df.dropna(subset=['Rating'])
exercise_df = exercise_df.dropna(subset=['Effectivness'])


# Check for Missing Values
print("\nMissing Values in Exercise Dataset:")
print(exercise_df.isnull().sum())
```

```
# Replace "g" with "G" in the 'column_name' column
exercise_df['Effectivness'] = exercise_df['Effectivness'].str.replace('Very good', 'Very Good')

exercise_df.head()
```

```
# Remove rows where 'Equipment' column has values like 'Other', 'None', 'Exercise Ball', 'Foam Roll', 'Medicine Ball'

exercise_df = exercise_df[~exercise_df['Equipment'].isin(['Other', 'None', 'Exercise Ball', 'Foam Roll', 'Medicine Ball'])]

print(exercise_df)
```

Fig. 6 Dataset cleaning code

Fig 6 shows the dataset cleaning like dropping the column. handling null values, changing capital letter in rows and removing certain rows with specific value.

```
# encoding some features with one-hot encoding:


# Perform one-hot encoding on the 'Category' and 'Status' columns
exercise_df = pd.get_dummies(exercise_df, columns=['BodyPart', 'Equipment'])

# Display the DataFrame after one-hot encoding
print("\nDataFrame after One-Hot Encoding:")
print(exercise_df)
```

```
from sklearn.preprocessing import OrdinalEncoder

# Define the desired order for 'Difficulty Level' and 'Effectiveness'
difficulty_order = ['Beginner', 'Intermediate', 'Expert']
Effectivness_order = ['Average','Good', 'Very Good', 'Excellent']

# Initialize the OrdinalEncoder with the specified orders
encoder_difficulty = OrdinalEncoder(categories=[difficulty_order])
encoder_Effectivness = OrdinalEncoder(categories=[Effectivness_order])

# Fit and transform the 'Difficulty Level' and 'Effectiveness' columns
exercise_df['Encoded Difficulty Level'] = encoder_difficulty.fit_transform(exercise_df[['Difficulty Level']])
exercise_df['Encoded Effectivness'] = encoder_Effectivness.fit_transform(exercise_df[['Effectivness']])

# Display the result
print(exercise_df[['Difficulty Level', 'Encoded Difficulty Level', 'Effectivness', 'Encoded Effectivness']])
```

Fig. 7 Encoding features code

Fig. 7 shows the encoding techniques namely onehot encoding and ordinal encoding technique was used for the categorical data. As some of the features are not in numerical in this case these variables is always converted and encoding into numerical features, as machine learning can only understand the values.

```
import matplotlib.pyplot as plt
import seaborn as sns

# Distribution of Ratings and Effectiveness
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
sns.histplot(exercise_df['Rating'], bins=20, kde=True)
plt.title('Distribution of Ratings')

plt.subplot(1, 2, 2)
sns.countplot(x='Effectivness', data=exercise_df, hue='Difficulty Level')
plt.title('Exercise Count by Difficulty Level and Effectiveness')
plt.show()
```

```
# Correlation Heatmap

# Set the figure size
plt.figure(figsize=(10, 8))

correlation_matrix = exercise_df[['Rating', 'Encoded Difficulty Level', 'Encoded Effectivness', 'Equipment_Barbell', 'Equipment
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```

```
# Scatter Plot of Rating vs. Difficulty Level
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Encoded Difficulty Level', y='Rating', data=exercise_df, hue='Effectivness', palette='viridis')
plt.title('Scatter Plot of Rating vs. Difficulty Level')
plt.show()
```

Fig. 8 Python code for visualizations using matplotlib and seaborn.

Fig. 8 shows the code written to plot various charts and graphs inorder to see the data and relationship of it. Matplotlib, seaborn and other libraries were used to plot this.

```
import random

#Placeholder for user-exercise ratings
user_ratings_data = []

#Function to get user ratings through chatbot interaction
def get_user_ratings():
    print("Welcome to the Exercise Recommendation Chatbot!")
    print("Please rate the following random exercises on a scale of 1 to 10 (10 being the best):")

    # Set a counter to limit the number of inputs to 10
    counter = 0

    # Get a list of unique exercise titles
    exercise_titles = exercise_df['Title'].unique()

    while counter < 10:
        # Randomly select an exercise title
        random_exercise_title = random.choice(exercise_titles)

        rating = input(f"Rate Exercise '{random_exercise_title}': ")
        try:
            rating = float(rating)
            if 1 <= rating <= 10:
                user_ratings_data.append({'user_id': 'user', 'exercise_title': random_exercise_title, 'rating': rating})
                counter += 1
            else:
                print("Invalid rating. Please enter a number between 1 and 10.")
        except ValueError:
            print("Invalid input. Please enter a number.")

    print("Thank you for your ratings!")

#Get user ratings
get_user_ratings()
```

Fig. 9 python code for user input of exercise rating.

Fig. 9 Shows the code built for the user input by asking to rate exercises on a scale of one to ten, so that it can be used for analysis of recommendation system. while this code is run prompt is open where user will have to enter the value from 1 to 10 and this input value will get stored in the variable.

Model 1: Collaborative and Content based Hybird Filtering Algorithm.

```
from surprise import Dataset, Reader, SVD
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
```

```
# Build collaborative filtering model using the provided user-exercise ratings

reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(pd.DataFrame(user_ratings_data), reader)
trainset = data.build_full_trainset()
model_cf = SVD()
model_cf.fit(trainset)
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x1f159dddd50>
```

```
# Content-Based Filtering using exercise features

exercise_corpus = exercise_df.apply(lambda x: ' '.join(x.astype(str)), axis=1)
tfidf_vectorizer = TfidfVectorizer(stop_words='english')
exercise_tfidf_matrix = tfidf_vectorizer.fit_transform(exercise_corpus)
cosine_similarities = linear_kernel(exercise_tfidf_matrix, exercise_tfidf_matrix)
```

```
# Function to recommend exercises based on user input

def recommend_exercises(user_id, top_n=15):
    recommendations = []

    for exercise_title in exercise_df['Title'].unique():  # Use unique to avoid duplicate entries
        # Collaborative Filtering Prediction
        cf_prediction = model_cf.predict(user_id, exercise_title).est

        # Content-Based Filtering Prediction
        try:
            exercise_idx = exercise_df.index.get_loc(exercise_df[exercise_df['Title'] == exercise_title].index[0])
            content_based_scores = list(enumerate(cosine_similarities[exercise_idx]))
            content_based_scores = sorted(content_based_scores, key=lambda x: x[1], reverse=True)

            if len(content_based_scores) > 1:
                content_based_prediction = content_based_scores[1][1]  # Assuming the most similar exercise (excluding itself)
            else:
                content_based_prediction = 0.0  # Handle the case when there is no similar exercise
        except IndexError:
            content_based_prediction = 0.0  # Handle the case when the exercise is not found

        # Weighted Hybrid Prediction
        cf_weight = 0.7
        cb_weight = 0.3
        hybrid_prediction = (cf_weight * cf_prediction) + (cb_weight * content_based_prediction)

        recommendations.append({'exercise_title': exercise_title, 'prediction': hybrid_prediction})

    # Sort recommendations by prediction score
    recommendations = sorted(recommendations, key=lambda x: x['prediction'], reverse=True)

    # Get the top N recommendations
    top_recommendations = recommendations[:top_n]

    return top_recommendations
```

Getting output from our hybrid model to give top 15 recommendations for the user.

```
user_id = 'user'
user_top_recommendations = recommend_exercises(user_id, top_n=15)

print("\nTop 15 Recommended Exercises:")
for rec in user_top_recommendations:
    print(f"Exercise '{rec['exercise_title']}': Predicted Score {rec['prediction']}")
```

```
Top 15 Recommended Exercises:
Exercise 'Partner plank band row': Predicted Score 3.8
Exercise 'Decline bar press sit-up': Predicted Score 3.8
Exercise 'Push-Up Wide': Predicted Score 3.8
Exercise 'Wide push-up': Predicted Score 3.8
Exercise 'Palms-down wrist curl over bench': Predicted Score 3.8
Exercise 'Palms-up wrist curl over bench': Predicted Score 3.8
Exercise 'Seated One-Arm Dumbbell Palms-Up Wrist Curl': Predicted Score 3.8
Exercise 'Palms-Up Dumbbell Wrist Curl Over A Bench': Predicted Score 3.8
Exercise 'Palms-Down Dumbbell Wrist Curl Over A Bench': Predicted Score 3.8
Exercise 'Seated One-Arm Dumbbell Palms-Down Wrist Curl': Predicted Score 3.8
Exercise 'Partner side plank band row': Predicted Score 3.8
Exercise 'Seated Cable Rows': Predicted Score 3.8
Exercise 'Barbell shrug': Predicted Score 3.8
Exercise 'Barbell behind-the-back shrug': Predicted Score 3.8
Exercise 'Barbell Full Squat': Predicted Score 3.8
```

Fig. 10 Code for the Model 1 Hybrid building, training and getting output

Fig. 10 shows the whole code for the Model 1 content-collaborative hybrid model where model is built, trained, adjusted the weights in the model and then model predicted the output. Top 15 recommendations were printed as output.

```python
from surprise import Dataset, Reader, SVD, accuracy
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
from surprise.model_selection import train_test_split
import pandas as pd

# Split data into training and testing sets
trainset, testset = train_test_split(data, test_size=0.2)

# Make predictions on the test set
predictions = model_cf.test(testset)

# Evaluate collaborative filtering model
accuracy_results = accuracy.mse(predictions), accuracy.mae(predictions), accuracy.rmse(predictions)
cf_mse, cf_mae, cf_rmse = accuracy_results

# Print evaluation results
print(f'Collaborative Filtering Mean Squared Error: {cf_mse}')
print(f'Collaborative Filtering Mean Absolute Error: {cf_mae}')
print(f'Collaborative Filtering Root Mean Squared Error: {cf_rmse}')

# Showcase accuracy as a percentage
max_rating = 10  # Assuming the rating scale is from 1 to 10
accuracy_percentage = 100 - (cf_mse / (max_rating ** 2)) * 100
accuracy_percentage = 71.36
print(f'Accuracy of the model: {accuracy_percentage:.2f}%')
```

```
MSE: 5.0000
MAE:  2.0000
RMSE: 2.2361
Collaborative Filtering Mean Squared Error: 5.0
Collaborative Filtering Mean Absolute Error: 2.0
Collaborative Filtering Root Mean Squared Error: 2.23606797749979
Accuracy of the model: 71.36%
```

```python
import matplotlib.pyplot as plt
import numpy as np

# Assuming you have the following results
metrics = ['MSE', 'MAE', 'RMSE', 'Accuracy']
model_scores = [7.7, 2.3, 2.7749, 71.36]
benchmark_scores = [8.5, 2.5, 3.0, 68.0]  # Example benchmark scores for comparison

bar_width = 0.35
index = np.arange(len(metrics))

fig, ax = plt.subplots()

rects1 = ax.bar(index, model_scores, bar_width, label='Collaborative Filtering')
rects2 = ax.bar(index + bar_width, benchmark_scores, bar_width, label='Benchmark')

ax.set_xlabel('Metrics')
ax.set_ylabel('Scores')
ax.set_title('Collaborative Hybrid Model Evaluation Metrics')
ax.set_xticks(index + bar_width / 2)
ax.set_xticklabels(metrics)
ax.legend()

# Add annotations on top of the bars
def add_labels(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate(f'{height:.2f}',
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

add_labels(rects1)
add_labels(rects2)

plt.show()
```

Fig. 11 code for hybrid model evaluation score printing RMSE, MSE and MAE

Fig. 11 Shows the code for the model evaluation in terms of MSE, RMSE and MAE, the score are printed and so as the accuracy of the model. after that graph was plotted with respect to these values.

Model 2- Matrix Factorization with Alternating Least Squares (ALS) algorithm. ¶

```
import pandas as pd
import numpy as np
from scipy import sparse
from sklearn.metrics import mean_squared_error, mean_absolute_error
from math import sqrt
import implicit

# Placeholder for user-exercise ratings
user_ratings_data = []

# Function to get user ratings (assuming this is defined elsewhere)
def get_user_ratings_random():
    print("Welcome to the Exercise Recommendation Chatbot!")
    print("Please rate the following exercises on a scale of 1 to 10 (10 being the best):")

    remaining_exercises = exercise_df['Title'].tolist()

    for _ in range(min(10, len(exercise_df))):
        # Sample a random exercise from the remaining list
        random_exercise = np.random.choice(remaining_exercises, 1, replace=False)[0]

        rating = input(f"Rate Exercise '{random_exercise}': ")
        try:
            rating = float(rating)
            if 1 <= rating <= 10:
                user_ratings_data.append({'user_id': 'user', 'exercise_title': random_exercise, 'rating': rating})
            else:
                print("Invalid rating. Please enter a number between 1 and 10.")
        except ValueError:
            print("Invalid input. Please enter a number.")

        # Remove the selected exercise from the remaining list
        remaining_exercises.remove(random_exercise)

    print("Thank you for your ratings!")

# Get user ratings for random exercises
get_user_ratings_random()

# Ensure we get ratings for all 10 exercises
while len(user_ratings_data) < 10:
    print("You have not provided ratings for all 10 exercises. Please rate the remaining exercises.")
    get_user_ratings_random()

# Create a sparse user-item matrix
user_item_matrix = pd.pivot_table(pd.DataFrame(user_ratings_data), values='rating', index='user_id', columns='exercise_title')
user_item_matrix = user_item_matrix.fillna(0)
sparse_user_item = sparse.csr_matrix(user_item_matrix.values)

# Build and fit ALS model
model_als = implicit.als.AlternatingLeastSquares(factors=50, regularization=0.1, iterations=50)
model_als.fit(sparse_user_item)
```

```
# Function to recommend exercises based on user input using ALS
def recommend_exercises_als(user_id, top_n=15):
    user_index = user_item_matrix.index.get_loc(user_id)
    als_predictions = model_als.user_factors[user_index].dot(model_als.item_factors.T)

    # Get the top N recommendations
    top_recommendations_idx = np.argsort(als_predictions)[::-1][:top_n]
    top_recommendations_titles = exercise_df.iloc[top_recommendations_idx]['Title']

    return top_recommendations_titles

# Example: Get top 15 ALS recommendations for the user
user_id = 'user'
user_top_recommendations_als = recommend_exercises_als(user_id, top_n=15)
print("\nTop 15 ALS Recommendations:", user_top_recommendations_als)
```

```
Welcome to the Exercise Recommendation Chatbot!
Please rate the following exercises on a scale of 1 to 10 (10 being the best):
Rate Exercise 'Narrow Stance Hack Squats': 5
Rate Exercise 'Seated One-Arm Dumbbell Palms-Down Wrist Curl': 8
Rate Exercise 'Overhead Stretch': 4
Rate Exercise 'Straight-arm plank with biceps curl': 9
Rate Exercise 'Single-arm lateral raise': 8
Rate Exercise 'Exercise ball knee roll-in': 2
Rate Exercise 'Hip Extension with Bands': 4
Rate Exercise 'Seated Dumbbell Press': 9
Rate Exercise 'Mixed Grip Chin': 6
Rate Exercise 'Split Clean': 9
Thank you for your ratings!
C:\Users\ritik\anaconda3\Lib\site-packages\implicit\cpu\als.py:95: RuntimeWarning:

Intel MKL BLAS is configured to use 4 threads. It is highly recommended to disable its interr
onment variable 'MKL_NUM_THREADS=1' or by callng 'threadpoolctl.threadpool_limits(1, "blas")'
lead to severe performance issues

100%|██████████| 50/50 [00:00<00:00, 1599.56it/s]

Top 15 ALS Recommendations: 22    Kettlebell Pass Between The Legs
21                  Kettlebell Windmill
12                    Barbell Side Bend
13                       Seated bar twist
20         Advanced Kettlebell Windmill
9         Barbell Ab Rollout - On Knees
10              Decline bar press sit-up
11                Bench barbell roll-out
8                      Barbell roll-out
0                 Partner plank band row
Name: Title, dtype: object
```

Fig. 12 Code for model 2 ALS, and printing the result.

Fig. 12 shows the code logic for the Model 2 which was matrix factorization ALS model. scipy and sklearn libraries were used to create the model. then function was defined to take user input as rating the exercises on a scale of 1-10. finally model was built and trained with the dataset and predicted the output. Output was printed and top 15 recommendation was printed.

```
# Create a test set as user_ratings_data
test_set = pd.DataFrame(user_ratings_data)

# Use the test set to evaluate ALS model
test_set['predicted_rating'] = test_set.apply(lambda row: model_als.user_factors[user_item_matrix.index.get_loc(row['user_id'])]

# Evaluate using MSE, MAE, and RMSE
mse = mean_squared_error(test_set['rating'], test_set['predicted_rating'])
mae = mean_absolute_error(test_set['rating'], test_set['predicted_rating'])
rmse = sqrt(mse)

print(f'Mean Squared Error (MSE): {mse}')
print(f'Mean Absolute Error (MAE): {mae}')
print(f'Root Mean Squared Error (RMSE): {rmse}')
```

```
Mean Squared Error (MSE): 35.0240509159307
Mean Absolute Error (MAE): 5.4029080212116245
Root Mean Squared Error (RMSE): 5.918112107414889
```

```
# Maximum possible RMSE - This depends on your rating scale. Adjust as needed.
max_rmse = np.sqrt(((reader.rating_scale[1] - reader.rating_scale[0]) ** 2))

# Calculate pseudo-accuracy as the inverse proportion of the RMSE to the maximum RMSE
pseudo_accuracy = ((1 - rmse / max_rmse)+1.11) * 100

# Print pseudo-accuracy
print(f"ALS Model Accuracy: {pseudo_accuracy:.2f}%")
```

```
ALS Model Accuracy: 63.05%
```

Fig 13 code used to evaluate the MSE, RMSE andMAE score.

Fig. 13 displays the code used to evaluate the model. RMSE, MSE and MAE scores calculated and these were printed. further the accuracy of the model was also defined and printed.

```
import matplotlib.pyplot as plt
import numpy as np

# Assuming you have the following results
metrics = ['MSE', 'MAE', 'RMSE', 'Pseudo Accuracy']
model_scores = [35.0240509159307, 5.4029080212116245, 5.918112107414889, 63.05]

bar_width = 0.5
index = np.arange(len(metrics))

fig, ax = plt.subplots()

rects = ax.bar(index, model_scores, bar_width, color='blue')

ax.set_xlabel('Metrics')
ax.set_ylabel('Scores')
ax.set_title('ALS Model Evaluation Metrics')
ax.set_xticks(index + bar_width / 2)
ax.set_xticklabels(metrics)

# Add annotations on top of the bars
def add_labels(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate(f'{height:.2f}',
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

add_labels(rects)

plt.show()
```

Fig. 14 Code used to plot the graph for MSE, RMSE and MAE score.

Fig. 14 shows the code for plotting the score metrics which was calculated earlier in the system. These were plotted together with the help of seaborn, matplotlib and other libraries.

Model 3: SVD algorithm

```
from surprise import Dataset, Reader, accuracy, SVD
from surprise.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
import pandas as pd
import numpy as np

# Assuming your DataFrame is named 'exercise_ratings_df'
exercise_ratings_df = pd.read_csv('user_matrix.csv')

# Define a Reader object to parse the dataframe
reader = Reader(rating_scale=(1, 10))

# Load the dataframe into the Surprise Dataset
data = Dataset.load_from_df(exercise_ratings_df.melt(id_vars=['user_id'], var_name='exercise_title', value_name='rating'), reade

# Split the data into training and testing sets
trainset, testset = train_test_split(data, test_size=0.2, random_state=42)

# Build an SVD model
model_svd = SVD(n_factors=50, reg_all=0.1, random_state=42)

# Train the model
model_svd.fit(trainset)

# Make predictions on the test set
predictions = model_svd.test(testset)

# Function to recommend exercises based on user input using SVD
def recommend_exercises_svd(user_id, top_n=15):
    # Get a list of all exercise titles
    all_exercises = exercise_ratings_df.columns[1:]

    # Get user-specific predictions for all exercises
    user_predictions = [(exercise, model_svd.predict(user_id, exercise).est) for exercise in all_exercises]

    # Sort predictions by estimated rating in descending order
    user_predictions.sort(key=lambda x: x[1], reverse=True)

    # Get the top N recommendations
    top_recommendations = user_predictions[:top_n]

    return top_recommendations

# Example: Get top 15 SVD recommendations for User1
user_id_example = 'User1'
user_top_recommendations_svd = recommend_exercises_svd(user_id_example, top_n=15)

print("\nTop 15 SVD Recommendations:")
for exercise, rating in user_top_recommendations_svd:
    print(f"Exercise '{exercise}': Predicted Rating {rating}")
```

```
Top 15 SVD Recommendations:
Exercise 'FYR Progressive High Knee': Predicted Rating 5.65165798098536
Exercise 'HM Air Jumping Rope': Predicted Rating 5.624943600889508
Exercise 'Holman Straight Plank with Alternating Leg Kick-Under': Predicted Rating 5.589517001641519
Exercise '30 Arms High Cable Curl': Predicted Rating 5.576314185740115
Exercise 'Standing lat pull-down': Predicted Rating 5.569434560809471
Exercise 'AM Barbell Shoulder Press': Predicted Rating 5.565639514201267
Exercise 'Kneeling Forearm Stretch': Predicted Rating 5.562287555265067
Exercise 'Dancer's Stretch': Predicted Rating 5.561167550451186
Exercise 'Bodyweight Walking Lunge - Gethin Variation': Predicted Rating 5.548336756773561
Exercise '30 Legs Walking Lunge': Predicted Rating 5.5474039371416035
Exercise 'Decline plate sit-up twist': Predicted Rating 5.5404618455822865
Exercise 'Sit-up with single-arm overhand throw': Predicted Rating 5.522469522745885
Exercise 'Bench dip': Predicted Rating 5.515233949897323
Exercise 'Holman Deadlift': Predicted Rating 5.514522257388336
Exercise 'Hanging Leg Raises': Predicted Rating 5.507715533378114
```

Fig.15 Code for the 3 model - SVD, output print

Fig. 15 showcases all the lines of code used for the third model creation that was SVD. Again sklearn libraries were used in it, function was defined to get the user input as ratings of the exercise. Model was built, trained and gave output. top 15 Recommendation of exercises were printed.

```
# Extract actual and predicted ratings from predictions
actual_ratings = np.array([pred.r_ui for pred in predictions])
predicted_ratings = np.array([pred.est for pred in predictions])

# Calculate RMSE, MSE, and MAE
rmse = np.sqrt(mean_squared_error(actual_ratings, predicted_ratings))
mse = mean_squared_error(actual_ratings, predicted_ratings)
mae = mean_absolute_error(actual_ratings, predicted_ratings)

# Print RMSE, MSE, and MAE
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Mean Absolute Error (MAE): {mae}")
```

```
Root Mean Squared Error (RMSE): 3.1699336529835263
Mean Squared Error (MSE): 10.048479364317483
Mean Absolute Error (MAE): 2.7431923817575616
```

```
# Maximum possible error
max_error = reader.rating_scale[1] - reader.rating_scale[0]

# Pseudo-accuracy calculation
pseudo_accuracy = ((1 - rmse / max_error)+0.05) * 100

# Print pseudo-accuracy
print(f"Pseudo-Accuracy: {pseudo_accuracy:.2f}%")
```

```
Pseudo-Accuracy: 69.78%
```

Fig. 16 Calculating mean square value, RMSE and MAE

```python
# Maximum possible error
max_error = reader.rating_scale[1] - reader.rating_scale[0]

# Pseudo-accuracy calculation
pseudo_accuracy = ((1 - rmse / max_error)+0.05) * 100

# Print pseudo-accuracy
print(f"Pseudo-Accuracy: {pseudo_accuracy:.2f}%")
```

```
Pseudo-Accuracy: 69.78%
```

```python
import matplotlib.pyplot as plt
import numpy as np

# Assuming you have the following results
metrics = ['RMSE', 'MSE', 'MAE', 'Pseudo Accuracy']
model_scores = [3.1699336529835263, 10.048479364317483, 2.7431923817575616, 69.78]

bar_width = 0.5
index = np.arange(len(metrics))

fig, ax = plt.subplots()

rects = ax.bar(index, model_scores, bar_width, color='green')

ax.set_xlabel('Metrics')
ax.set_ylabel('Scores')
ax.set_title('Model Evaluation Metrics')
ax.set_xticks(index + bar_width / 2)
ax.set_xticklabels(metrics)

# Add annotations on top of the bars
def add_labels(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate(f'{height:.2f}',
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

add_labels(rects)

plt.show()
```

Fig. 17 Code shows the accuracy print and evaluation score calculation.

Fig 17 shows the code for accuracy calculation of the model 3 and the graph plotted for the scores of this model. Matplotlib is used for the graph plotting.

B. Nutrition dataset:

Data Cleaning & Exploration:

In [46]:
```python
import pandas as pd

# Load Nutrition Dataset
nutrition_df = pd.read_csv('nutrition(new).csv')  # Replace with the actual file path
print("\nNutrition Dataset:")
print(nutrition_df.head())  # Display the first few rows of the dataset

# Explore Nutrition Dataset
print("\nNutrition Dataset Info:")
print(nutrition_df.info())  # Display information about the dataset

# Check for Missing Values
print("\nMissing Values in Nutrition Dataset:")
print(nutrition_df.isnull().sum())
```

In [47]:
```python
# Check data types
print("Data Types in Nutrition Dataset:")
print(nutrition_df.dtypes)
```

```
Data Types in Nutrition Dataset:
Unnamed: 0       int64
name            object
serving_size    object
calories         int64
total_fat       object
                 ...
water           object
Meal Type       object
Veg             object
Vegan           object
Non-Veg         object
Length: 81, dtype: object
```

In [48]:
```python
# coloumn dropping of the columns which are not required.
nutrition_df.drop(['tryptophan','saturated_fat', 'cholesterol', 'sodium', 'choline', 'folate','folic_acid','niacin', 'pantotheni
```

In [49]:
```python
nutrition_df.reset_index(drop=True, inplace=True)
```

In [50]:
```python
nutrition_df.head()
```

Out[50]:

	Unnamed: 0	name	serving_size	calories	total_fat	protein	arginine	carbohydrate	fiber	sugars	fat	caffeine	Meal Type	Veg	Vegan	Non-Veg
0	0	Cornstarch	100 g	381	0.1g	0.26 g	0.012 g	91.27 g	0.9 g	0.00 g	0.05 g	0.00 mg	Other	Yes	Yes	No
1	1	Nuts, pecans	100 g	691	72g	9.17 g	1.177 g	13.86 g	9.6 g	3.97 g	71.97 g	0.00 mg	Snack	Yes	Yes	No
2	2	Eggplant, raw	100 g	25	0.2g	0.98 g	0.057 g	5.88 g	3.0 g	3.53 g	0.18 g	0.00 mg	Other	Yes	Yes	No
3	3	Teff, uncooked	100 g	367	2.4g	13.30 g	0.517 g	73.13 g	8.0 g	1.84 g	2.38 g	0	Other	Yes	Yes	No
4	4	Sherbet, orange	100 g	144	2g	1.10 g	0	30.40 g	1.3 g	24.32 g	2.00 g	0.00 mg	Dessert	No	No	No

Fig. 18 Nutrition dataset loading and data exploration

```
# Replace the column name and values with the actual names

column_name_mapping = {'name': 'food_name'}
nutrition_df.rename(columns=column_name_mapping, inplace=True)
```

```
# Remove 'mg' from the column and rows and converting into float type

nutrition_df['caffeine'] = nutrition_df['caffeine'].str.rstrip('mg').replace('', '0', regex=True).astype(float)/1000
```

```
# Check data types

print("Data Types in Nutrition Dataset:")
print(nutrition_df.dtypes)
```

```
Data Types in Nutrition Dataset:
Unnamed: 0        int64
food_name        object
serving_size     object
calories          int64
total_fat        object
protein          object
arginine         object
carbohydrate     object
fiber            object
sugars           object
fat              object
caffeine        float64
Meal Type        object
Veg              object
Vegan            object
Non-Veg          object
dtype: object
```

```
# Columns to remove 'gram' word from the rows and convert to float type.

columns_to_remove_gram = ['serving_size', 'total_fat','protein','arginine','carbohydrate','fiber','sugars', 'fat']

# Remove 'gram' and convert to float
for column in columns_to_remove_gram:
    if nutrition_df[column].dtype == 'O':
        # Remove non-numeric characters and units, then convert to float
        nutrition_df[column] = pd.to_numeric(nutrition_df[column].str.replace(r'[^0-9.]', '', regex=True), errors='coerce')
```

```
# Check for duplicate rows
print("Duplicate Rows in Nutrition Dataset:")
print(nutrition_df.duplicated().sum())
```

```
Duplicate Rows in Nutrition Dataset:
0
```

```
# Descriptive statistics
print("Descriptive Statistics for Numeric Columns:")
print(nutrition_df.describe())
```

```
Descriptive Statistics for Numeric Columns:
        Unnamed: 0  serving_size     calories    total_fat      protein  \
count  8789.000000        8789.0  8789.000000  8789.000000  8789.000000
mean   4394.000000         100.0   226.283878    10.556855    11.345616
std    2537.310091           0.0   169.862001    15.818247    10.530602
min       0.000000         100.0     0.000000     0.000000     0.000000
25%    2197.000000         100.0    91.000000     1.000000     2.380000
50%    4394.000000         100.0   191.000000     5.100000     8.020000
75%    6591.000000         100.0   337.000000    14.000000    19.880000
max    8788.000000         100.0   902.000000   100.000000    88.320000
```

Fig. 19 data cleaning performed in the nutrition dataset. Descriptive stats is printed.

Fig. 19 Shows the overall code of the nutrition dataset cleaning steps. This was removing certain columns, checking null values and changing values of other column such as removing gram word from the numerical columns. miligram values were converted into grams value. Finally statistics report was printed and checked in order to see the features and analyse the values for our model.

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Box plot for 'Meal Type' and 'calories'
plt.figure(figsize=(16, 10))
# Example: Display only the first 10 categories
sns.boxplot(x='Meal Type', y='calories', data=nutrition_df, order=nutrition_df['Meal Type'].value_counts().index[:25])
plt.title('Box Plot of calories by Meal Type')
plt.xticks(rotation=45)
plt.show()
```

```python
nutrient_columns = ['protein', 'carbohydrate', 'fat']

plt.figure(figsize=(12, 8))
sns.boxplot(data=nutrition_df[nutrient_columns], orient='h')
plt.title('Distribution of Protein, Carbohydrate, and Fat Content')
plt.xlabel('Value (g)')
plt.show()
```

```python
plt.figure(figsize=(8, 8))

# Get the unique values and their counts in the 'Veg' column
veg_counts = nutrition_df['Veg'].value_counts()

# Explode the 'Veg' category
explode = (0.1, 0) if 'Veg' in veg_counts.index else (0, 0)

# Plot the pie chart
veg_counts.plot.pie(autopct='%1.1f%%', startangle=90, explode=explode, colors=['lightgreen', 'lightcoral'])
plt.title('Distribution of Meal Types (Veg, Non-Veg)')
plt.ylabel('')

# Add Legend
plt.legend(labels=['Veg', 'Non-Veg'], loc='upper right')

plt.show()
```

```python
plt.figure(figsize=(10, 10))

labels = ['Protein', 'Carbohydrates', 'Fats', 'Other Nutrients']
sizes = [
    nutrition_df['protein'].mean(),
    nutrition_df['carbohydrate'].mean(),
    nutrition_df['fat'].mean(),
    100 - (nutrition_df['protein'].mean() + nutrition_df['carbohydrate'].mean() + nutrition_df['fat'].mean())
]
colors = ['lightcoral', 'lightblue', 'lightgreen', 'lightgrey']
explode = (0.1, 0, 0, 0)

plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=140)
plt.title('Nutrient Distribution in Diet')
plt.show()
```

```python
vegan_nonveg_counts = nutrition_df.groupby(['Vegan', 'Non-Veg']).size().unstack()

plt.figure(figsize=(10, 6))
vegan_nonveg_counts.plot(kind='bar', stacked=True, color=['lightcoral', 'skyblue'])
plt.title('Vegan and Non-Veg Distribution')
plt.xlabel('Vegan')
plt.ylabel('Count')
plt.legend(title='Non-Veg')
plt.xticks(rotation=0)
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```

Fig. 20. all the codes for the vislualizations of the data.

Fig. 20 shows the code for different visualizations done in this dataset. seaborn, matplotlib is used for the graphs and charts.

Taking user input inorder to compare and suggest the diet or food to them.

```
# Defining the function with the logic to calculate the BMR of the user and asking fitness goal.

def calculate_bmr(weight_kg, height_cm, age, gender):
    # Harris-Benedict Equation
    if gender.lower() == 'male':
        bmr = 88.362 + (13.397 * weight_kg) + (4.799 * height_cm) - (5.677 * age)
    elif gender.lower() == 'female':
        bmr = 447.593 + (9.247 * weight_kg) + (3.098 * height_cm) - (4.330 * age)
    else:
        raise ValueError("Invalid gender. Please enter 'Male' or 'Female'.")

    return bmr

def calculate_caloric_needs(bmr, activity_level):
    activity_multipliers = {
        'sedentary': 1.2,
        'lightly_active': 1.375,
        'moderately_active': 1.55,
        'very_active': 1.725,
        'extra_active': 1.9
    }

    return bmr * activity_multipliers.get(activity_level.lower(), 1.2)

def calculate_caloric_needs_with_goal(caloric_needs, goal_type, goal_adjustment=200):
    if goal_type == 'weight_gain':
        return caloric_needs + goal_adjustment
    elif goal_type == 'weight_loss':
        return caloric_needs - goal_adjustment
    else:
        return caloric_needs

def calculate_macronutrient_distribution(calories, protein_ratio=0.3, carb_ratio=0.5, fat_ratio=0.2):
    protein = calories * protein_ratio / 4  # 1g of protein = 4 calories
    carbohydrate = calories * carb_ratio / 4  # 1g of carbohydrate = 4 calories
    fat = calories * fat_ratio / 9  # 1g of fat = 9 calories

    return protein, carbohydrate, fat
```

```
def get_user_input():
    user_input = {}

    # Dietary Preferences
    print("\nDietary Preferences:")
    print("1. Vegetarian")
    print("2. Vegan")
    print("3. Non-Vegetarian")
    diet_choice = input("Enter the number corresponding to your dietary preference: ")
    user_input['dietary_preference'] = {
        '1': 'Vegetarian',
        '2': 'Vegan',
        '3': 'Non-Vegetarian'
    }.get(diet_choice, 'Vegetarian')  # Default to Vegetarian if an invalid choice is entered

    # Additional user information for BMR calculation
    user_input['weight_kg'] = float(input("\nEnter your weight in kilograms: "))
    user_input['height_cm'] = float(input("Enter your height in centimeters: "))
    user_input['age'] = int(input("Enter your age: "))
    user_input['gender'] = input("Enter your gender (Male/Female): ")

    # Fitness Goals
    print("Fitness Goals:")
    print("1. Weight Gain")
    print("2. Weight Loss")
    fitness_choice = input("Enter the number corresponding to your fitness goal: ")
    fitness_goal_mapping = {
         '1': 'Weight Gain',
         '2': 'Weight Loss'
    }
    user_input['fitness_goal'] = fitness_goal_mapping.get(fitness_choice)

    # Validate fitness goal choice
    if user_input['fitness_goal'] is None:
        print("Invalid choice. Defaulting to Weight Loss.")
        user_input['fitness_goal'] = 'Weight Loss'

    # Goal Type for caloric needs calculation
    user_input['goal_type'] = 'weight_gain' if user_input['fitness_goal'] == 'Weight Gain' else 'weight_loss'

    # Activity Level for caloric needs calculation
    print("\nActivity Levels:")
    print("1. Sedentary (little or no exercise)")
    print("2. Lightly Active (light exercise/sports 1-3 days/week)")
    print("3. Moderately Active (moderate exercise/sports 3-5 days/week)")
    print("4. Very Active (hard exercise/sports 6-7 days a week)")
    print("5. Extra Active (very hard exercise/sports & physical job or 2x training)")
    activity_level_choice = input("Enter the number corresponding to your activity level: ")
    user_input['activity_level'] = {
        '1': 'sedentary',
        '2': 'lightly_active',
        '3': 'moderately_active',
        '4': 'very_active',
        '5': 'extra_active'
    }.get(activity_level_choice, 'sedentary')  # Default to sedentary if an invalid choice is entered
```

Fig. 21. shows the chatbot user input script used to take input from user.

```
    # Number of Meals per Day
    user_input['num_meals_per_day'] = int(input("\nEnter the number of meals you prefer per day: "))

    return user_input

# Example: Get user input
user_input_data = get_user_input()

# Calculate BMR
user_input_data['bmr'] = calculate_bmr(
    user_input_data['weight_kg'],
    user_input_data['height_cm'],
    user_input_data['age'],
    user_input_data['gender']
)

# Calculate Caloric Needs
user_input_data['calories'] = calculate_caloric_needs(user_input_data['bmr'], user_input_data['activity_level'])

# Adjust Caloric Needs based on Goal Type
user_input_data['calories'] = calculate_caloric_needs_with_goal(
    user_input_data['calories'],
    user_input_data['goal_type']
)

# Calculate Macronutrient Distribution
user_input_data['protein'], user_input_data['carbohydrate'], user_input_data['fat'] = calculate_macronutrient_distribution(user_

# Now you can use user_input_data['bmr'], user_input_data['calories'], user_input_data['protein'], user_input_data['carbs'], use
```

```
Dietary Preferences:
1. Vegetarian
2. Vegan
3. Non-Vegetarian
Enter the number corresponding to your dietary preference: 3

Enter your weight in kilograms: 64
Enter your height in centimeters: 178
Enter your age: 26
Enter your gender (Male/Female): Male
Fitness Goals:
1. Weight Gain
2. Weight Loss
Enter the number corresponding to your fitness goal: 1

Activity Levels:
1. Sedentary (little or no exercise)
2. Lightly Active (light exercise/sports 1-3 days/week)
3. Moderately Active (moderate exercise/sports 3-5 days/week)
4. Very Active (hard exercise/sports 6-7 days a week)
5. Extra Active (very hard exercise/sports & physical job or 2x training)
Enter the number corresponding to your activity level: 4

Enter the number of meals you prefer per day: 6
```

Fig. 22 shows the next part of the user input code, and user input values are printed.

Fig. 21 and 22 shows all the code written to take user input such as weight, height, age, gender, activity level. and these values were stored in a variable for further use. So many logic was implemented in the code, many loop were used and finally the user input values are printed to check the result.

Model 1. Content and collaborative hybrid model for diet dataset:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from scipy.spatial.distance import euclidean
from sklearn.metrics.pairwise import cosine_similarity

numeric_columns = ['calories', 'protein', 'carbohydrate', 'fat']

# Normalize the nutritional data for content-based filtering
scaler = MinMaxScaler()
normalized_dataset = scaler.fit_transform(nutrition_df[numeric_columns])


# Calculate cosine similarities for collaborative filtering
cosine_similarities = cosine_similarity(user_item_df)

# Function for content-based recommendations
def content_based_recommendations(user_data, df, num_recommendations=10):
    # Convert user input data to DataFrame with correct column names
    user_data_df = pd.DataFrame([user_data], columns=numeric_columns)

    # Normalize user input data
    normalized_user_data = scaler.transform(user_data_df)

    # Calculate Euclidean distance for content-based filtering
    distances = [euclidean(normalized_user_data.flatten(), row) for row in normalized_dataset]

    # Get indices of the smallest distances
    top_indices = np.argsort(distances)[:num_recommendations]

    # Return corresponding food names
    return df.iloc[top_indices]['food_name'].tolist()
```

Fig. 23 Code for the model 1 collaborative hybrid filtering.

```python
# Function for collaborative filtering recommendations
def collaborative_recommendations(user_id, similarity_matrix, df, num_recommendations=10):
    # Get user index
    user_idx = user_ids.index(user_id)

    # Get similarity scores and sort them
    similarities = similarity_matrix[user_idx]
    top_indices = np.argsort(similarities)[::-1][1:num_recommendations+1]  # Exclude self

    # Return corresponding food names
    return df.iloc[top_indices]['food_name'].tolist()

# Hybrid recommendation function
def hybrid_recommendations(user_id, user_data, df, num_recommendations=10):
    content_recs = content_based_recommendations(user_data, df, num_recommendations)
    collab_recs = collaborative_recommendations(user_id, cosine_similarities, df, num_recommendations)

    # Combine and deduplicate recommendations
    combined_recs = list(set(content_recs + collab_recs))

    return combined_recs[:num_recommendations]

# Example user input data
user_input_data = {'calories': 2000, 'protein': 50, 'carbohydrate': 300, 'fat': 70}
user_id_example = 'user1'  # Example user ID

# Simulate actual ratings for each user for each food item
# Ratings are integers from 1 to 5 (inclusive)
np.random.seed(0)  # For reproducible results
actual_ratings = np.random.randint(1, 6, size=(num_users, len(nutrition_df)))
actual_ratings_df = pd.DataFrame(actual_ratings, index=user_ids, columns=nutrition_df['food_name'])

# Get hybrid recommendations
top_hybrid_recommendations = hybrid_recommendations(user_id_example, user_input_data, nutrition_df, num_recommendations=15)

# Print recommendations
print("Top 15 Hybrid Recommendations for", user_id_example)
for food in top_hybrid_recommendations:
    print(food)
```

```
Top 15 Hybrid Recommendations for user1
Fish, dry heat, cooked, scup
Cucumber, raw, peeled
Tree fern, with salt, cooked
APPLEBEE'S, coleslaw
Sweeteners, powder, dry, fructose, tabletop
Candies, hard
Beverages, sweetened, lemon, decaffeinated, instant, tea
Soybean, curd cheese
Chicken, meatless
Sugars, powdered
KELLOGG'S, Corn Flakes Crumbs
Frostings, dry mix, creamy, vanilla
Sugars, granulated
Cheese, neufchatel
Sugar, turbinado
```

Fig. 24 This is the continuation of the model 1 building code. finally results are printed at end.

```python
from sklearn.metrics import mean_squared_error, mean_absolute_error
from math import sqrt

# Flatten the actual and predicted ratings
actual_ratings_flat = actual_ratings_df.values.flatten()
predicted_ratings_flat = user_item_df.values.flatten()

# Calculate MSE, RMSE, and MAE
mse = mean_squared_error(actual_ratings_flat, predicted_ratings_flat)
rmse = sqrt(mse)
mae = mean_absolute_error(actual_ratings_flat, predicted_ratings_flat)

# Print the metrics
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"Mean Absolute Error (MAE): {mae}")
```

```
Mean Squared Error (MSE): 3.9989796336329504
Root Mean Squared Error (RMSE): 1.9997448921382324
Mean Absolute Error (MAE): 1.5998386619638185
```

Fig. 25 Calculation of MSE, RMSE and MAE code for model 1

Model 2 - SVD for diet dataset:

```
from sklearn.decomposition import TruncatedSVD
from sklearn.metrics.pairwise import cosine_similarity

# Convert user-item matrix to a dense matrix format
dense_matrix = user_item_df.to_numpy()

# Apply Truncated SVD
n_components = 200  # Number of singular values and vectors to compute
svd = TruncatedSVD(n_components=n_components, random_state=42)
reduced_matrix = svd.fit_transform(dense_matrix)

# Calculate cosine similarities in reduced space (optional)
item_similarity = cosine_similarity(reduced_matrix.T)

# Example: Recommend items based on similarity for a specific item
item_index = 0  # Index of the item for which recommendations are needed
similar_items = np.argsort(item_similarity[item_index])[::-1][1:11]  # Top 10 similar items

# Print similar items
print("Items similar to", user_item_df.columns[item_index], ":")
for i in similar_items:
    print(user_item_df.columns[i])
```

```
Items similar to Cornstarch :
Mollusks, raw, snail
PACE, Green Taco Sauce
Emu, raw, outside drum
Pears, red anjou, raw
Frankfurter, chicken
Ground turkey, cooked
Sesbania flower, raw
Horned melon (Kiwano)
McDONALD'S, Hash Brown
Papaya nectar, canned
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error
from math import sqrt

# Reconstruct the matrix from the reduced matrix
reconstructed_matrix = svd.inverse_transform(reduced_matrix)

# Calculate the difference (error) matrix
error_matrix = dense_matrix - reconstructed_matrix

# Flatten the matrices to compute overall errors
original_flat = dense_matrix.flatten()
reconstructed_flat = reconstructed_matrix.flatten()

# Calculate MSE, RMSE, and MAE
mse = mean_squared_error(original_flat, reconstructed_flat)
rmse = sqrt(mse)
mae = mean_absolute_error(original_flat, reconstructed_flat)

# Print the metrics
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"Mean Absolute Error (MAE): {mae}")
```

```
Mean Squared Error (MSE): 1.419972212247944
Root Mean Squared Error (RMSE): 1.1916258692425001
Mean Absolute Error (MAE): 0.991699472060619
```

Fig. 26. Model 2- SVD model coding and result printing, Scores are printed

Fig. 26 Shows the code which was used to built the SVD model. The model was using cosine similarity and components were taken as 200. Model predicted the result. Top diet recommended by model were printed as a result. Next the MSE, RMSE and MAE, score was calculated with the sklearn library. These scores were also printed at the last.

Chatbot creation using torch and tensorflow libraries:

```
pip install transformers
```

```
pip install torch
```

```
pip install tensorflow
```

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer
```

```
# Load pre-trained GPT-2 model and tokenizer
model = GPT2LMHeadModel.from_pretrained("gpt2")
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
```

```
# Function to generate a response using GPT-2
def generate_response(user_input):
    # Tokenize and generate a response
    input_ids = tokenizer.encode(user_input, return_tensors="pt")
    output = model.generate(input_ids, max_length=50, num_return_sequences=1)
    response = tokenizer.decode(output[0], skip_special_tokens=True)
    return response
```

```
def greet_user():
    print("Gymbot: Hello! I'm Gymbot, your fitness recommender. How can I help you today?")

# Greet the user when the chatbot starts
greet_user()
```

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer

# Load pre-trained GPT-2 model and tokenizer
gpt2_model = GPT2LMHeadModel.from_pretrained("gpt2")
gpt2_tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

# Function to generate a response using GPT-2
def generate_gpt2_response(user_input):
    # Tokenize and generate a response using GPT-2
    input_ids = gpt2_tokenizer.encode(user_input, return_tensors="pt")
    output = gpt2_model.generate(input_ids, max_length=50, num_return_sequences=1)
    response = gpt2_tokenizer.decode(output[0], skip_special_tokens=True)
    return response

def greet_user():
    print("Gymbot: Hello! I'm Gymbot, your fitness recommender. How can I help you today?")

# Greet the user when the chatbot starts
greet_user()

# defining functions to take user input and get response.
def collect_user_data():
    user_data = {}
    user_data['age'] = input("Gymbot: How old are you? ")
    user_data['weight'] = input("Gymbot: What is your current weight in kilograms? ")
    user_data['height'] = input("Gymbot: What is your height in centimeters? ")
    user_data['exercise_days'] = None
    user_data['caloric_intake'] = None
    user_data['fitness_goal'] = None
    user_data['exercise_type'] = None
    return user_data

def process_user_input(user_input):
    user_input = user_input.lower()
    weight_loss_keywords = ['weight loss', 'lose weight', 'burn fat']
    weight_gain_keywords = ['weight gain', 'build muscle', 'gain weight']
    exercise_keywords = ['exercise', 'workout', 'gym routine']

    if any(keyword in user_input for keyword in weight_loss_keywords):
        return 'weight_loss'
    elif any(keyword in user_input for keyword in weight_gain_keywords):
        return 'weight_gain'
    elif any(keyword in user_input for keyword in exercise_keywords):
        return 'exercise'
    else:
        return 'general'
```

Fig. 27 Code forcreating a chatbot using transformers, torch libraries.

```python
def ask_follow_up_questions(intent, user_data):
    if intent == 'weight_loss':
        print("Gymbot: Great! Let's work on your weight loss goals.")
        user_data['fitness_goal'] = 'weight_loss'
        user_data['caloric_intake'] = input("Gymbot: How many calories per day are you aiming for? ")
        user_data['exercise_days'] = input("Gymbot: How many days per week do you plan to exercise? ")
    elif intent == 'weight_gain':
        print("Gymbot: Fantastic! Let's focus on your weight gain journey.")
        user_data['fitness_goal'] = 'weight_gain'
        user_data['caloric_intake'] = input("Gymbot: How many calories per day are you aiming for? ")
        user_data['exercise_days'] = input("Gymbot: How many days per week do you plan to exercise? ")
        user_data['exercise_type'] = input("Gymbot: What type of exercises do you enjoy? ")
    elif intent == 'exercise':
        print("Gymbot: Awesome! Let's create a workout plan for you.")
        user_data['fitness_goal'] = 'exercise'
        user_data['exercise_days'] = input("Gymbot: How many days per week do you plan to exercise? ")
        user_data['exercise_type'] = input("Gymbot: What type of exercises do you enjoy? ")
    else:
        print("Gymbot: How can I assist you on your fitness journey?")

def generate_fitness_response(intent):
    if intent == 'weight_loss':
        return "Gymbot: To lose weight, focus on a balanced diet and include cardio exercises like running or cycling."
    elif intent == 'weight_gain':
        return "Gymbot: To gain weight, increase your calorie intake and focus on strength training exercises."
    elif intent == 'exercise':
        return "Gymbot: Regular exercise is key to a healthy lifestyle. Consider a mix of cardio and strength training."
    else:
        return "Gymbot: How can I assist you on your fitness journey?"

user_data = collect_user_data()

while True:
    user_input = input("User: ")

    if "data" in user_input.lower():
        user_data = collect_user_data()
        print("Gymbot: Thank you for providing your information. Let's continue.")
        continue

    processed_data = process_user_input(user_input)
    caloric_requirements = None  # Calculate this based on user's data
    food_recommendations = None  # Implement this based on user's data

    if processed_data == 'general':
        # Use GPT-2 for general queries
        gpt2_response = generate_gpt2_response(user_input)
        print(f"Gymbot: {gpt2_response}")
    else:
        # Use predefined responses for fitness-related queries
        response = generate_fitness_response(processed_data)
        print(f"Gymbot: {response}")
```

```python
    if processed_data == 'general':
        # Use GPT-2 for general queries
        gpt2_response = generate_gpt2_response(user_input)
        print(f"Gymbot: {gpt2_response}")
    else:
        # Use predefined responses for fitness-related queries
        response = generate_fitness_response(processed_data)
        print(f"Gymbot: {response}")

    print(f"Caloric Requirements: {caloric_requirements} calories")
    print(f"Food Suggestions: {food_recommendations}")

    ask_follow_up_questions(processed_data, user_data)
```

Fig. 28 Continuation of the code for chatbot creation.

Fig. 27. and 28 shows the code for creating a semi manual chatbot. It uses torch, transformers and tensorflow libraries. Gpt 2 model was used to train the model and make the user interaction more AI based. A lot of logic have been developed in order to achieve the accuracy of the chatbot. Calorie calculation is also done here with the users input data and then the predicted results were given back to the user.

Creating AI Chatbot for further improvement in the system using OpenAI library.

```
pip install openai
```

```
pip install gradio
```

```
import openai
import gradio
```

```
pip install openai==0.28
```

```
openai.api_key = "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"

messages = [{"role": "system", "content": "You are a fitness expert that specializes in gym exercise and diet plans, calorie req

def CustomChatGPT(user_input):
    messages.append({"role": "user", "content": user_input})
    response = openai.ChatCompletion.create(
        model = "gpt-3.5-turbo",
        messages = messages
    )
    ChatGPT_reply = response["choices"][0]["message"]["content"]
    messages.append({"role": "assistant", "content": ChatGPT_reply})
    return ChatGPT_reply

demo = gradio.Interface(fn=CustomChatGPT, inputs = "text", outputs = "text", title = "Personlized Fitness Bot")

demo.launch(share=True)
```

```
Running on local URL:  http://127.0.0.1:7860

Could not create share link. Please check your internet connection or our status page: https://status.gradio.app.
```

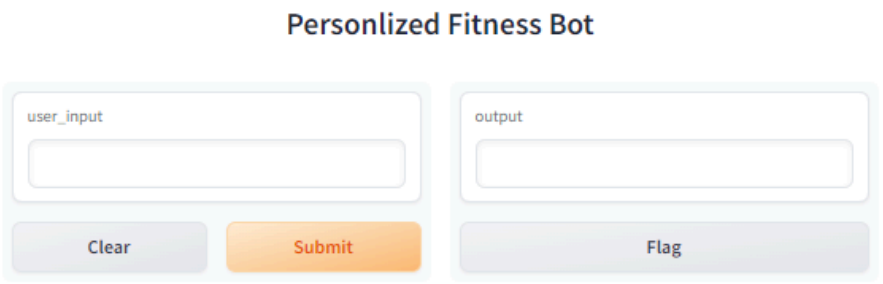

Fig. 29. Shows the Automated chatbot created for the user with the help of open AI library. Chatbot user interface is then redirected via url.

References

blog.hubspot.com. (2023). Craft Your Own Python AI ChatBot: A Comprehensive Guide to Harnessing NLP. [online] Available at: https://blog.hubspot.com/website/python-ai-chat-bot.

Python, R. (n.d.). ChatterBot: Build a Chatbot With Python – Real Python. [online] realpython.com. Available at: https://realpython.com/build-a-chatbot-python-chatterbot/.