Configuration Manual

MSc Research Project
MSc Data Analytics

Mohib Tariq
Student ID: x23259850

School of Computing
National College of Ireland

Supervisor: Teerath Kumar Menghwar

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: …….Mohib Tariq ……………………………………………………………………

Student ID: ……x23259850……………………………………………………………………………………..……

Programme: ………MSc Data Analytics……………………………**Year:** ………2024…………..

Module: ………MSc Research Project……………………………………………………………….………

Lecturer: ………Teerath Kumar……………………………………………………………………….………

Submission Due Date: ………12th December 2024……………………………………………………………….………

Project Title: ……Research Project Configuration Manual……………………..………

Word Count: ……………2017……………… **Page Count:** ……………………21………….…….………

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: ……………Mohib Tariq………………………………………………………………

Date: …………12th December 2024…………………………………………………

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

Mohib Tariq
x23259850

1 Introduction

This configuration manual is prepared to explain the technical implementation of the research study. All the tools, environment, and code configurations are mentioned in this manual. This configuration manual also contains code snippets of the implementation. Section 2 focuses on the environment which was used during the project. Section 3 focuses on how the data was collected. Section 4 contains the data pre-processing steps applied. Section 5 discusses the Experiments that were carried out.

2 Environment

Figure 1 shows the hardware configuration which was used for this research project. The project uses a 13th Gen Intel(R) Core™ i5-1335U @ 1.30 GHz processor and 16 GB (15.7 GB usable) installed RAM. The type of system used is a 64-bit operating system with an x64-based processor.

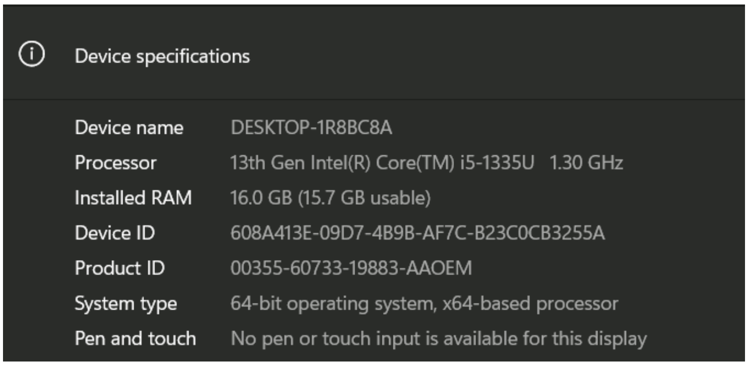

Figure 1: System Info

Furthermore, Jupyter Notebook was used as the Integrated Development Environment. Python was used as the programming language for the implementation of this project.

3 Data Collection

This research study utilized three different datasets. The first dataset contained information regarding Ireland's Tourist Attractions which was a public dataset taken from the data.gov.ie website. Figure 2 shows the data.gov.ie webpage from where the dataset was taken. The data set is available for download.

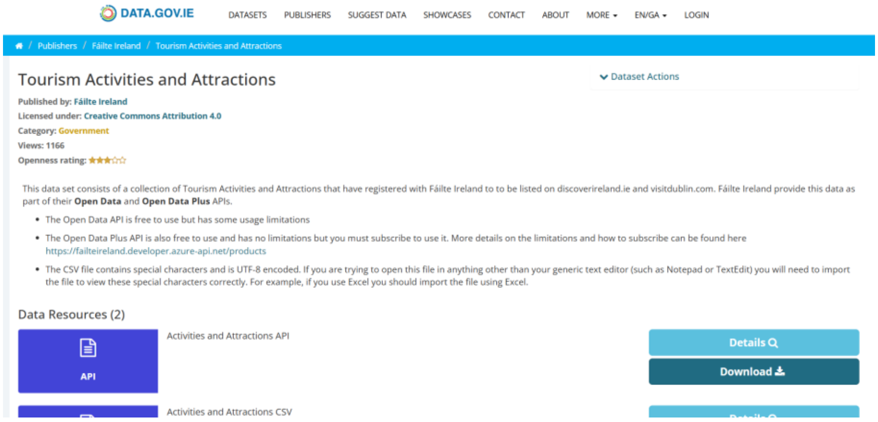

Figure 2: Source Of Dataset 1 (Ireland's Tourist Attractions)

Additionally, The Google Places API was used to fetch ratings of each of the attractions from Google Maps. The fetched ratings were added to the original dataset. Figure 3 shows the python script which was used to fetch the ratings and add them to the original dataset.

```python
import requests
import pandas as pd
import time

API_KEY = 'AIzaSyAxgmneAapeJSjltFsal61eDX4Mw4sluPs'

def get_place_rating(place_name):
    base_url = 'https://maps.googleapis.com/maps/api/place/textsearch/json'
    params = {
        'query': place_name,
        'key': API_KEY,
    }

    response = requests.get(base_url, params=params)

    if response.status_code != 200:
        print(f"Failed to fetch data for {place_name}, Status code: {response.status_code}")
        return None

    results = response.json().get('results', [])

    if not results:
        print(f"No results found for {place_name}")
        return None

    rating = results[0].get('rating', None)
    return rating

file_path = 'Attractions.csv'
attractions_df = pd.read_csv(file_path)

attractions_df['Rating'] = None

for index, row in attractions_df.iterrows():
    place_name = row['Name']
    print(f"Fetching rating for: {place_name}")

    rating = get_place_rating(place_name)

    attractions_df.at[index, 'Rating'] = rating

    time.sleep(2)

output_file = 'Attractions_with_ratings.csv'
attractions_df.to_csv(output_file, index=False)

print(f"Data saved to {output_file}")
```

Figure 3: Python Script For Adding ratings in Dataset 1

The second dataset that was used in this research was taken from the Failte Ireland Open Data. The dataset contains information about Ireland’s Accommodations including Hotels, Guest Houses etc.

The third dataset was sourced from Google Maps by using the Google Places API. Figure 4 shows the Python script which was used for the automation of fetching and aggregating data. The results were stored in a structure csv for further analysis.

```python
import requests
import pandas as pd
import time

API_KEY = 'AIzaSyAxgmneAapeJSjltFsal61eDX4Mw4sluPs'
base_url = 'https://maps.googleapis.com/maps/api/place/textsearch/json'

def fetch_tourist_attractions(query, max_results=10000):
    attractions = []
    next_page_token = None
    total_fetched = 0

    while total_fetched < max_results:
        params = {
            'query': query,
            'key': API_KEY,
        }
        if next_page_token:
            params['pagetoken'] = next_page_token

        response = requests.get(base_url, params=params)

        if response.status_code != 200:
            print(f"Request failed with status code: {response.status_code}")
            break

        results = response.json()

        if 'results' not in results or not results['results']:
            print("No more results found.")
            break

        for place in results['results']:
            name = place.get('name', 'No name')
            rating = place.get('rating', 'No rating')
            latitude = place['geometry']['location'].get('lat') if 'geometry' in place else None
            longitude = place['geometry']['location'].get('lng') if 'geometry' in place else None
            address = place.get('formatted_address', 'No address')
            types = ', '.join(place.get('types', []))

            attractions.append({
                'Name': name,
                'Rating': rating,
                'Latitude': latitude,
                'Longitude': longitude,
                'Address': address,
                'Tags': types
            })

        total_fetched += len(results['results'])

        next_page_token = results.get('next_page_token', None)
        if not next_page_token:
            print(f"Total fetched for query '{query}': {total_fetched}")
            break

        time.sleep(2)

    return attractions
```

```python
queries = [
    'tourist attractions in Ireland',
    'landmarks in Ireland',
    'historical sites in Ireland',
    'tourist attractions in Dublin',
    'tourist attractions in Galway',
    'tourist attractions in Cork',
    'most visited places in ireland',
    'adventurous places in ireland',
    'Museums in ireland',
    'amusement parks in ireland',
    'famous bars in ireland',
    'famous activities in ireland',
    'famous parks in ireland',
    'cultural sights in Ireland',
    'Pubs in Ireland',
    'Christmas markets in Ireland',
    'Pubs in Ireland',
    'Clubs In Ireland',
    'Architecture in Ireland',
    'Cathedrals in Ireland',
    'Parks in Ireland',
    'Parties in Ireland',
    'Cinemas in Ireland',
    'Shopping Malls in Ireland'
]

all_attractions = []

for query in queries:
    print(f"Fetching data for query: {query}")
    attractions_data = fetch_tourist_attractions(query, max_results=10000)
    all_attractions.extend(attractions_data)
    print(f"Total records after query '{query}': {len(all_attractions)}")

    if len(all_attractions) >= 10000:
        break

if all_attractions:
    df = pd.DataFrame(all_attractions)
    df.to_csv('tourist_attractions_ireland_full_2.csv', index=False)
    print(f"Data successfully saved to 'tourist_attractions_ireland_full.csv' with {len(all_attractions)} records.")
else:
    print("No data to save.")
```

Figure 4: Python Script For Data Collection For Dataset 3

4 Installation Of Packages and Libraries

There were many libraries that were required to install and import for each of the techniques of filtering. Figure 5 shows all the imported libraries which were used throughout the project.

```python
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import TruncatedSVD
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from scipy.sparse import hstack, csr_matrix
```

```python
import pandas as pd
import numpy as np
from sklearn.decomposition import TruncatedSVD
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split
from scipy.sparse import csr_matrix
import matplotlib.pyplot as plt
```

```python
import pandas as pd
import numpy as np
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split
from scipy.sparse import hstack, csr_matrix
from sklearn.linear_model import Ridge
from sklearn.cluster import KMeans
```

Figure 5: Imported libraries

5 Data Cleaning and Processing

Data Cleaning and Processing were performed separately on each of the three datasets. Figure 6 shows data cleaning and processing performed on dataset 1. Moving on, Figure 7 shows Data Cleaning applied on Dataset 2.

```
maps_ratings['Rating'].fillna(maps_ratings['Rating'].mean(), inplace=True)
```

```
maps_ratings['County'].fillna(maps_ratings['County'].mode()[0], inplace=True)
```

```
maps_ratings.drop(['Url', 'Telephone'], axis=1, inplace=True)
```

Figure 6: Dataset 1 - Filling Null Values and Dropping Columns

```
accomodations_data = accomodations_data.drop(columns=['Proprietor Description'])
```

```
accomodations_data = accomodations_data.dropna(subset=['Latitude', 'Longitude'])
```

```
accomodations_data = accomodations_data.drop(columns=['Address Line 1'])
```

```
accomodations_data = accomodations_data.dropna(subset=['Rating', 'Eircode/Postal code'])
```

```
accomodations_data['Rating'].fillna(accomodations_data['Rating'].mode()[0], inplace=True)
```

Figure 7: Dataset 2 - Data Cleaning

From the figures it can be seen that the following steps were applied for data pre-processing:

- Replacing null values with the mean of the particular column
- Replacing null values with the mode of the particular column
- Dropping unnecessary columns
- Dropping rows with null values

Figure 9 shows the application of label Encoding on some of the columns of Dataset 2. The user ratings were simulated using Gaussian distribution to make the ratings more realistic. Figure 8 shows the code snippet for the simulation of user ratings in Dataset 1. The same method of simulating user ratings was also implemented in Datasets 2 and 3.

```python
np.random.seed(42)
num_users = 200

user_ratings = []

for attraction_id in range(len(tourist_data)):
    num_ratings = np.random.randint(1, 10)
    for _ in range(num_ratings):
        user_id = np.random.randint(1, num_users + 1)
        rating = np.clip(np.random.normal(loc=3.5, scale=1.0), 0, 5)
        user_ratings.append({
            'User_ID': user_id,
            'Attraction': tourist_data['Name'][attraction_id],
            'User_Rating': rating
        })
```

Figure 8: Dataset 1 - Simulation Of User Ratings

```python
rating_mapping = {
    'Hotel - 5 Star': 5,
    'Hotel - 4 Star': 4,
    'Welcome Standard - Approved': 3,
    'Holiday Cottage Approved': 3,
}

accomodations_data['Rating'] = accomodations_data['Rating'].replace(rating_mapping)

accomodations_data['Rating'] = pd.to_numeric(accomodations_data['Rating'], errors='coerce')

print("\nStandardized Ratings:")
print(accomodations_data['Rating'].head())
```

```python
county_mapping = {
    'Wicklow': 1,
    'Galway': 2,
    'Cork': 3,
    'Dublin': 4,
    'Wexford': 5,
    'Kerry': 6,
    'Mayo': 7,
    'Tipperary': 8,
    'Clare': 9,
    'Waterford': 10,
    'Kilkenny': 11,
    'Westmeath': 12,
    'Roscommon': 13,
    'Donegal': 14,
    'Laois': 15,
    'Limerick': 16,
    'Kildare': 17,
    'Sligo': 18,
    'Leitrim': 19,
    'Louth': 20,
    'Cavan': 21,
    'Monaghan': 22,
    'Meath': 23,
    'Longford': 24,
    'Carlow': 25,
    'Offaly' : 26,
}

accomodations_data['Address County'] = accomodations_data['Address County'].replace(rating_mapping)

accomodations_data['Address County'] = pd.to_numeric(accomodations_data['Rating'], errors='coerce')

print("\nStandardized Counties:")
print(accomodations_data['Address County'].head())
```

```python
sector_mapping = {
    'Hotel' : 1,
    'Guest House' : 2,
    'Holiday Hostel': 3,
    'Historic House': 4,
    'Holiday Apartment': 5,
    'Caravan and Camping Park' : 6,
    "Fáilte Ireland's Welcome Standard": 7,
    'B&Bs' : 13,
    'Holiday Cottage': 8,
    'Youth Hostel': 9,
    'Individual Self Catering Cottage': 10,
    'Individual Self Catering Apartment': 11,
    'Holiday Camp': 12,
}

accomodations_data['Sector'] = accomodations_data['Sector'].replace(sector_mapping)

accomodations_data['Sector'] = pd.to_numeric(accomodations_data['Sector'], errors='coerce')

print("\nStandardized Sectors:")
print(accomodations_data['Sector'].head())
```

Figure 9: Dataset 2 - Label Encoding

6 Experiments

6.1 Content-Based Filtering

6.1.1 Dataset 1:

Figures 10 and 11 show the application of Content-Based Filtering on Dataset 1. Figure 10 shows the feature extraction part and model development. Figure 11 shows the methods of evaluation performed.

```
tourist_data = pd.read_csv('final_final_tourist_data.csv')
```

```
#Vectorizing Tags column using TF-IDF

tags_vectorizer = TfidfVectorizer(stop_words='english')
tags_tfidf_matrix = tags_vectorizer.fit_transform(tourist_data['Tags'].fillna(''))
```

```
# print(tags_tfidf_matrix)
```

```
# Reducing dimensionality of the TF-IDF matrix using Truncated SVD

svd = TruncatedSVD(n_components=100, random_state=42)
tags_tfidf_reduced = svd.fit_transform(tags_tfidf_matrix)
```

```
# Scale the Rating, Latitude, and Longitude columns to bring them to the same range

scaler = MinMaxScaler()
tourist_data[['Rating', 'Latitude', 'Longitude']] = scaler.fit_transform(tourist_data[['Rating', 'Latitude', 'Longitude']])
```

```
# MErging the user ratings with the original data
user_ratings_merged = user_ratings_df.merge(tourist_data, left_on='Attraction', right_on='Name')
```

```
# Features from textual and numerical attributes are combined
tags_features = svd.transform(tags_vectorizer.transform(user_ratings_merged['Tags'].fillna('')))
other_features = user_ratings_merged[['Rating', 'Latitude', 'Longitude']].values
combined_features = np.hstack([tags_features, other_features])
```

```
X = combined_features
y = user_ratings_merged['User_Rating'].values
```

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Train a KNeighborsRegressor to predict user ratings
knn_model = KNeighborsRegressor(n_neighbors=5, metric='cosine')
knn_model.fit(X_train, y_train)
```

Figure 10: Dataset 1 - Content-Based Filtering Part 1

```
y_knn_pred = []
batch_size = 1000
for i in range(0, X_test.shape[0], batch_size):
    X_test_batch = X_test[i:i + batch_size]
    y_knn_pred_batch = knn_model.predict(X_test_batch)
    y_knn_pred.extend(y_knn_pred_batch)
```

```
y_knn_pred = np.array(y_knn_pred)
```

```
rmse_knn = np.sqrt(mean_squared_error(y_test, y_knn_pred))
mae_knn = mean_absolute_error(y_test, y_knn_pred)
r2_knn = r2_score(y_test, y_knn_pred)

print(f"KNN Regressor - RMSE: {rmse_knn:.4f}")
print(f"KNN Regressor - MAE: {mae_knn:.4f}")
print(f"KNN Regressor - R^2 Score: {r2_knn:.4f}")
```

```
plt.figure(figsize=(10, 6))
x_values = range(len(y_test))
plt.plot(x_values, y_test, label='Actual Ratings', alpha=0.7, marker='o', linestyle='--')
plt.plot(x_values, y_knn_pred, label='Predicted Ratings', alpha=0.7, marker='x', linestyle='-')
plt.xlabel('Test Data Index')
plt.ylabel('Rating')
plt.title('Actual Vs Predicted Ratings  (Dataset 1 : Content Based Filtering)')
plt.legend()
plt.grid(True)
plt.show()
```

```
plt.figure(figsize=(10, 6))
errors = y_test - y_knn_pred
sns.histplot(errors, kde=True)
plt.xlabel('Prediction Error')
plt.title('Distribution of Prediction Errors')
plt.grid(True)
plt.show()
```

```
plt.figure(figsize=(10, 6))
sns.histplot(y_test, bins=50, color='blue', alpha=0.5, label='Original Ratings')
sns.histplot(y_knn_pred, bins=50, color='red', alpha=0.5, label='Predicted Ratings')
plt.xlabel('Rating Value')
plt.ylabel('Frequency')
plt.title('Distribution of Original and Predicted Ratings - Dataset 2 (Content Based Filtering)')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 11: Dataset 1 - Content-Based Filtering Part 2 (Evaluation)

6.1.2 Dataset 2:

Figures 12 and 13 show the application of content-based filtering on dataset 2. The following steps are performed:

- TF-IDF Vectorization of Account Name (Textual Feature)
- Scaling Of Numerical Features
- Both the TF-IDF vectors and the numeric features are combined into the form of one feature matrix.
- The combined feature matrix undergoes dimensionality reduction using SVD
- The user-item matrix is reconstructed
- Predictions are made based on the reconstructed matrix

```
df = pd.read_csv('final_accomodations_data.csv')

df['Account Name'] = df['Account Name'].fillna('')
df['Owner(s) as it appears on Register'] = df['Owner(s) as it appears on Register'].fillna('')

vectorizer_account = TfidfVectorizer(stop_words='english')
vectorizer_owner = TfidfVectorizer(stop_words='english')

account_tfidf_matrix = vectorizer_account.fit_transform(df['Account Name'])
owner_tfidf_matrix = vectorizer_owner.fit_transform(df['Owner(s) as it appears on Register'])

combined_tfidf_matrix = hstack([account_tfidf_matrix, owner_tfidf_matrix])

n_components = 20
svd = TruncatedSVD(n_components=n_components, random_state=42)
combined_tfidf_reduced = svd.fit_transform(combined_tfidf_matrix)

np.random.seed(42)
num_users = 50
user_ids = np.arange(1, num_users + 1)
```

```
scaler = MinMaxScaler()
df[['Rating', 'Latitude', 'Longitude', 'Sector', 'Total Units']] = scaler.fit_transform(df[['Rating', 'Latitude', 'Longitude', 'S

account_tfidf_matrix = vectorizer_account.transform(df['Account Name'])
owner_tfidf_matrix = vectorizer_owner.transform(df['Owner(s) as it appears on Register'])

combined_tfidf_matrix = hstack([account_tfidf_matrix, owner_tfidf_matrix])

combined_tfidf_reduced = svd.fit_transform(combined_tfidf_matrix)

content_matrix = hstack([csr_matrix(combined_tfidf_reduced), csr_matrix(df[['Rating', 'Latitude', 'Longitude', 'Sector', 'Total U

X_train, X_test, y_train, y_test = train_test_split(content_matrix, df['User_Rating'], test_size=0.2, random_state=42)

model = KNeighborsRegressor(n_neighbors=5, algorithm='brute', metric='cosine')
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

rmse_content = np.sqrt(mean_squared_error(y_test, y_pred))
mae_content = mean_absolute_error(y_test, y_pred)
r2_content = r2_score(y_test, y_pred)

rmse_content, mae_content, r2_content
```

Figure 12: Dataset 2 - Content-Based Filtering Part 1

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

plt.figure(figsize=(10, 6))
plt.plot(range(1, n_components + 1), svd.singular_values_, marker='o', linestyle='--')
plt.xlabel('Latent Factor Index')
plt.ylabel('Singular Value')
plt.title('Scree Plot of Singular Values')
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
sns.histplot(y_test, bins=50, color='blue', alpha=0.5, label='Original Ratings')
sns.histplot(y_pred, bins=50, color='red', alpha=0.5, label='Predicted Ratings')
plt.xlabel('Rating Value')
plt.ylabel('Frequency')
plt.title('Distribution of Original and Predicted Ratings - Dataset 2 (Content Based Filtering)')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.6, color='green')
plt.xlabel('Actual Ratings')
plt.ylabel('Predicted Ratings')
plt.title('Actual vs Predicted Ratings - Dataset 2 (Content Based Filtering)')
plt.grid
```

Figure 13: Dataset 2 - Content-Based Filtering Part 2

6.1.3 Dataset 3:

Figure 14 shows the first part of content-based filtering where TF-IDF is applied to the tags

```
dataset3 = pd.read_csv("tourist_attractions_ireland_full_2.csv")

tags_vectorizer = TfidfVectorizer(stop_words='english')
tags_tfidf_matrix = tags_vectorizer.fit_transform(dataset3['Tags'].fillna(''))

n_components = min(tags_tfidf_matrix.shape[1], 35)
svd = TruncatedSVD(n_components=n_components, random_state=42)
tags_tfidf_reduced = svd.fit_transform(tags_tfidf_matrix)

item_similarity = cosine_similarity(tags_tfidf_reduced)
```

Figure 14: Dataset 3 - Content-Based Filtering Part 1

Figure 14 shows the application of TF-IDF vectorization on the tags. SVD is applied to the tf-idf matrix to reduce the dimensionality of the matrix. Cosine similarities of the reduced tf-idf matrixes are calculated.

```
def train_knn_and_predict(X, user_ratings, n_neighbors=5):
    observed_indices = np.where(user_ratings > 0)[0]
    missing_indices = np.where(user_ratings == 0)[0]

    if len(observed_indices) == 0:
        return np.zeros(len(user_ratings))

    X_train, X_test, y_train, y_test = train_test_split(
        X[observed_indices], user_ratings[observed_indices], test_size=0.2, random_state=42
    )


    knn = KNeighborsRegressor(n_neighbors=n_neighbors, metric="cosine")
    knn.fit(X_train, y_train)


    y_pred_test = knn.predict(X_test)
    mse = mean_squared_error(y_test, y_pred_test)
    print(f"User MSE: {mse}")

    full_predictions = knn.predict(X)
    return full_predictions
```

Figure 15: Dataset 3 - Content-Based Filtering Part 2

Figure 15 shows the modeling part where a KNN Regressor is being trained.

```python
for user_idx in range(user_ratings_matrix.shape[0]):
    print(f"Processing User {user_idx + 1}/{user_ratings_matrix.shape[0]}")
    user_ratings = user_ratings_matrix[user_idx]
    predicted_ratings[user_idx] = train_knn_and_predict(tags_tfidf_reduced, user_ratings)

predicted_ratings_df = pd.DataFrame(predicted_ratings, index=user_ratings_df['User_ID'].unique(), columns=attractions)

predicted_ratings_file_path = "predicted_user_ratings_dataset3.csv"
predicted_ratings_df.to_csv(predicted_ratings_file_path)

true_ratings = []
pred_ratings = []

for user_idx in range(user_ratings_matrix.shape[0]):
    observed_indices = np.where(user_ratings_matrix[user_idx] > 0)[0]
    true_ratings.extend(user_ratings_matrix[user_idx][observed_indices])
    pred_ratings.extend(predicted_ratings[user_idx][observed_indices])

rmse = np.sqrt(mean_squared_error(true_ratings, pred_ratings))
mae = mean_absolute_error(true_ratings, pred_ratings)
r2 = r2_score(true_ratings, pred_ratings)

print("Evaluation Metrics:")
print(f"RMSE: {rmse}")
print(f"MAE: {mae}")
print(f"R²: {r2}")
```

Figure 16: Dataset 3 - Content-Based Filtering Part 3

Figure 17 shows the recommend_items function which generates recommendations to test the model.

```python
def recommend_items(user_index, user_ratings, similarity_matrix, top_n=5):
    user_sim_scores = similarity_matrix.dot(user_ratings)
    norm_sim_scores = user_sim_scores / np.array([np.abs(similarity_matrix).sum(axis=1)]).T.flatten()
    recommended_indices = np.argsort(-norm_sim_scores)[:top_n]
    return recommended_indices

user_index = 0
user_ratings = predicted_ratings[user_index]
top_recommendations = recommend_items(user_index, user_ratings, item_similarity, top_n=5)

recommended_items = dataset3.iloc[top_recommendations][['Name', 'Rating', 'Tags']]
print(recommended_items.reset_index(drop=True))
```

Figure 17: Dataset 3 - Generate Recommendations Function

6.2 Collaborative Filtering:

6.2.1 Dataset 1:

Figure 18 shows the application of collaborative filtering on dataset 1. A user-item matrix is created and the dimensionality is reduced by applying SVD to the matrix.

```python
df = pd.read_csv('tourist_data_with_ratings.csv')

df = df.groupby(['User_ID', 'Attraction']).agg({'User_Rating': 'mean'}).reset_index()

num_users = df['User_ID'].nunique()
num_items = df['Attraction'].nunique()

user_item_matrix = df.pivot(index='User_ID', columns='Attraction', values='User_Rating').fillna(0).values
user_item_matrix = csr_matrix(user_item_matrix)

n_components = 25
svd = TruncatedSVD(n_components=n_components, random_state=42)
user_item_reduced = svd.fit_transform(user_item_matrix)

user_item_approx = svd.inverse_transform(user_item_reduced)

X_train, X_test, y_train, y_test = train_test_split(user_item_matrix.toarray(), user_item_approx, test_size=0.2, random_state=42)

y_pred = X_test

rmse_svd = np.sqrt(mean_squared_error(y_test, y_pred))
mae_svd = mean_absolute_error(y_test, y_pred)
r2_svd = r2_score(y_test, y_pred)
```

Figure 18: Dataset 1 - Collaborative Filtering Part 1

Figure 19 shows the recommend_items function which generates recommendations to test the collaborative filtering model.

```python
def recommend_items(user_preferences, df, svd, num_recommendations=5):
    user_latent = svd.transform([user_preferences])
    user_approx_ratings = svd.inverse_transform(user_latent).flatten()

    recommendations = pd.DataFrame({'item': df['Attraction'].unique(), 'predicted_rating': user_approx_ratings})3
    recommended_items = recommendations.sort_values(by='predicted_rating', ascending=False).head(num_recommendations)
    return recommended_items

user_preferences = [np.random.uniform(0, 5) for _ in range(num_items)]  # Simulated user preferences

top_recommended_items = recommend_items(user_preferences, df, svd)
```

Figure 19: Dataset 1 - Collaborative Filtering Recommend Items Function

Figure 20 shows the Python code for the evaluation plots which were generated to visualize the performance of the model.

```python
plt.figure(figsize=(10, 6))
plt.plot(range(1, n_components + 1), svd.singular_values_, marker='o', linestyle='--')
plt.xlabel('Latent Factor Index')
plt.ylabel('Singular Value')
plt.title('Scree Plot of Singular Values')
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.hist(user_item_matrix.toarray().flatten(), bins=50, alpha=0.5, label='Original Ratings')
plt.hist(user_item_approx.flatten(), bins=50, alpha=0.5, label='Reconstructed Ratings')
plt.xlabel('Rating Value')
plt.ylabel('Frequency')
plt.title('Distribution of Original and Reconstructed Ratings')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
x_values = range(len(y_test.flatten()))
plt.plot(x_values, y_test.flatten(), label='Actual Ratings', alpha=0.7)
plt.plot(x_values, y_pred.flatten(), label='Predicted Ratings', alpha=0.7)
plt.xlabel('Test Data Index')
plt.ylabel('Rating')
plt.title('Actual vs Predicted Ratings')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 20: Dataset 1 - Collaborative Filtering Part 2 (Evaluation)

6.2.2 Dataset 2:

Figure 21 shows the implementation of the second dataset. A user-item matrix is created and Truncated SVD is applied. The user-item matrix is recreated and is used to make predictions. It can also be seen that data is split in the ratio of 80/20 where 80% is used for training and 20% is used for testing.

```
user_ratings_df = pd.read_csv('user_ratings_accommodations.csv')

user_ratings_df = user_ratings_df.groupby(['User_ID', 'Accommodation_ID']).agg({'User_Rating': 'mean'}).reset_index()

num_users = user_ratings_df['User_ID'].nunique()
num_items = user_ratings_df['Accommodation_ID'].nunique()

user_item_matrix = user_ratings_df.pivot(index='User_ID', columns='Accommodation_ID', values='User_Rating').fillna(0).values
user_item_matrix = csr_matrix(user_item_matrix)

n_components = 20
svd = TruncatedSVD(n_components=n_components, random_state=42)
user_item_reduced = svd.fit_transform(user_item_matrix)

user_item_approx = svd.inverse_transform(user_item_reduced)

X_train, X_test, y_train, y_test = train_test_split(user_item_matrix.toarray(), user_item_approx, test_size=0.2, random_state=42)

rmse_svd = np.sqrt(mean_squared_error(y_test, X_test))
mae_svd = mean_absolute_error(y_test, X_test)
r2_svd = r2_score(y_test, X_test)

print(f"Collaborative Filtering with Truncated SVD - RMSE: {rmse_svd:.4f}")
print(f"Collaborative Filtering with Truncated SVD - MAE: {mae_svd:.4f}")
print(f"Collaborative Filtering with Truncated SVD - R^2 Score: {r2_svd:.4f}")
```

Figure 21: Dataset 2 - Collaborative Filtering Part 1

Figure 22 shows the recommend_items() function which generates recommendations to test the collaborative filtering model.

```
def recommend_items(user_preferences, df, svd, num_recommendations=5):
    user_latent = svd.transform([user_preferences])
    user_approx_ratings = svd.inverse_transform(user_latent).flatten()

    recommendations = pd.DataFrame({'item': df['Accommodation_ID'].unique(), 'predicted_rating': user_approx_ratings})
    recommended_items = recommendations.sort_values(by='predicted_rating', ascending=False).head(num_recommendations)
    return recommended_items

user_preferences = [np.random.uniform(0, 5) for _ in range(num_items)]

top_recommended_items = recommend_items(user_preferences, user_ratings_df, svd)

print("\nTop Recommended Accommodations for You:")
print(top_recommended_items[['item', 'predicted_rating']])
```

Figure 22: Dataset 2 - Collaborative Filtering Recommend Items Function

```python
plt.figure(figsize=(10, 6))
plt.plot(range(1, n_components + 1), svd.singular_values_, marker='o', linestyle='--')
plt.xlabel('Latent Factor Index')
plt.ylabel('Singular Value')
plt.title('Scree Plot of Singular Values for Collaborative Filtering - Dataset 2')
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.hist(user_item_matrix.toarray().flatten(), bins=50, alpha=0.5, label='Original Ratings', color='blue')
plt.hist(user_item_approx.flatten(), bins=50, alpha=0.5, label='Reconstructed Ratings', color='orange')
plt.xlabel('Rating Value')
plt.ylabel('Frequency')
plt.title('Distribution of Original and Reconstructed Ratings')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
x_values = range(len(y_test.flatten()))
plt.plot(x_values, y_test.flatten(), label='Actual Ratings', alpha=0.7, color='blue')
plt.plot(x_values, X_test.flatten(), label='Predicted Ratings', alpha=0.7, color='red')
plt.xlabel('Test Data Index')
plt.ylabel('Rating')
plt.title('Actual vs Predicted Ratings')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 23: Dataset 2 - Collaborative Filtering (Evaluation)

The code of evaluation plots in figure 23 were used to generate visualizations for evaluation of collaborative filtering on the second dataset.

6.2.3 Dataset 3

This section covers the implementation of collaborative filtering on Dataset 3. Figure 24 shows the python code for the first part of collaborative filtering where a user-item matrix is created and SVD is performed to predict the user ratings.

```python
df = pd.read_csv("tourist_data_with_ratings_dataset3.csv")

df = df.groupby(['User_ID', 'Attraction']).agg({'User_Rating': 'mean'}).reset_index()

num_users = df['User_ID'].nunique()
num_items = df['Attraction'].nunique()

user_item_matrix = df.pivot(index='User_ID', columns='Attraction', values='User_Rating').fillna(0).values
user_item_matrix = csr_matrix(user_item_matrix)

n_components = 25
svd = TruncatedSVD(n_components=n_components, random_state=42)
user_item_reduced = svd.fit_transform(user_item_matrix)

user_item_approx = svd.inverse_transform(user_item_reduced)

X_train, X_test, y_train, y_test = train_test_split(user_item_matrix.toarray(), user_item_approx, test_size=0.2, random_state=42)

y_pred = X_test

rmse_svd = np.sqrt(mean_squared_error(y_test, y_pred))
mae_svd = mean_absolute_error(y_test, y_pred)
r2_svd = r2_score(y_test, y_pred)

print("Collaborative Filtering with Truncated SVD - Evaluation Metrics:")
print(f"RMSE: {rmse_svd:.4f}")
print(f"MAE: {mae_svd:.4f}")
print(f"R²: {r2_svd:.4f}")
```

Figure 24: Dataset 3 - Collaborative Filtering Part 1

```
def recommend_items(user_preferences, df, svd, num_recommendations=5):
    user_latent = svd.transform([user_preferences])

    user_approx_ratings = svd.inverse_transform(user_latent).flatten()

    recommendations = pd.DataFrame({
        'item': df['Attraction'].unique(),
        'predicted_rating': user_approx_ratings
    })

    recommended_items = recommendations.sort_values(by='predicted_rating', ascending=False).head(num_recommendations)
    return recommended_items

user_preferences = [np.random.uniform(0, 5) for _ in range(num_items)]

top_recommended_items = recommend_items(user_preferences, df, svd)

print("\nTop Recommended Tourist Attractions for You:")
print(top_recommended_items[['item', 'predicted_rating']])
```

Figure 25: Dataset 3 - Recommend Items Function

```
plt.figure(figsize=(10, 6))
plt.plot(range(1, n_components + 1), svd.singular_values_, marker='o', linestyle='--')
plt.xlabel('Latent Factor Index')
plt.ylabel('Singular Value')
plt.title('Scree Plot of Singular Values - Dataset 3 (Collaborative Filtering)')
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.hist(user_item_matrix.toarray().flatten(), bins=50, alpha=0.5, label='Original Ratings')
plt.hist(user_item_approx.flatten(), bins=50, alpha=0.5, label='Reconstructed Ratings')
plt.xlabel('Rating Value')
plt.ylabel('Frequency')
plt.title('Distribution of Original and Reconstructed Ratings')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
x_values = range(len(y_test.flatten()))
plt.plot(x_values, y_test.flatten(), label='Actual Ratings', alpha=0.7)
plt.plot(x_values, y_pred.flatten(), label='Predicted Ratings', alpha=0.7)
plt.xlabel('Test Data Index')
plt.ylabel('Rating')
plt.title('Actual vs Predicted Ratings')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 26: Dataset 3 - Collaborative Filtering Part 3 – Evaluation

Figure 25 shows the function to generate recommendations to test the model while Figure 26 shows the code for the plots which were used for evaluation.

6.3 Weighted Hybrid Approach:

This sections explains the shows the code configuration for the implementation of the weighted hybrid approach on each of the three datasets: Ireland’s Tourist Attractions, Ireland’s Accommodations, and Ireland’s famous places.

6.3.1 Dataset 1

This subsection shows the code for the implementation of the weighted hybrid approach on dataset 1.

```
final_tourist_data = pd.read_csv('final_final_tourist_data.csv')
tourist_data_with_ratings = pd.read_csv('tourist_data_with_ratings.csv')
```

```
df = final_tourist_data.merge(tourist_data_with_ratings, left_on='Name', right_on='Attraction', how='inner')
```

Figure 27: Dataset 1 - Weighted Hybrid Approach (Loading Data)

Figure 27 shows the code where the data is being loaded. The original data and the ratings data are merged together.

```
df['Rating'] = pd.to_numeric(df['Rating'], errors='coerce')
df['Rating'].fillna(df['Rating'].mean(), inplace=True)
```

```
df['Tags'] = df['Tags'].fillna('')
```

```
vectorizer = TfidfVectorizer(stop_words='english')
tags_matrix = vectorizer.fit_transform(df['Tags'])
```

```
scaler = MinMaxScaler()
df[['Rating', 'Latitude', 'Longitude']] = scaler.fit_transform(df[['Rating', 'Latitude', 'Longitude']])
```

```
content_matrix = hstack([tags_matrix, csr_matrix(df[['Rating', 'Latitude', 'Longitude']].values)])
```

Figure 28: Dataset 1- Weighted Hybrid Approach (Content Based Filtering Part)

Figure 28 shows the Content-Based Filtering Module where content-based filtering is being implemented. The content matrix combines the tf-idf features and the scaled numerical features to be used further in the hybrid approach.

```
tourist_data_with_ratings = tourist_data_with_ratings.groupby(['User_ID', 'Attraction']).agg({'User_Rating': 'mean'}).reset_index
user_item_matrix = tourist_data_with_ratings.pivot(index='User_ID', columns='Attraction', values='User_Rating').fillna(0).values
user_item_matrix = csr_matrix(user_item_matrix)
```

```
n_components = 20
svd = TruncatedSVD(n_components=n_components, random_state=42)
user_item_reduced = svd.fit_transform(user_item_matrix)
```

```
user_item_approx_reduced_expanded = np.tile(user_item_reduced, (int(np.ceil(content_matrix.shape[0] / user_item_reduced.shape[0])
user_item_approx_reduced_expanded = user_item_approx_reduced_expanded[:content_matrix.shape[0], :]
```

```
user_item_approx_reduced_expanded_sparse = csr_matrix(user_item_approx_reduced_expanded)
```

Figure 29: Dataset 1 - Weighted Hybrid Approach (Collaborative Filtering Part)

Figure 29 shows the collaborative filtering module where the reduced user-item matrix is saved as the collaborative filtering feature matrix.

```
kmeans = KMeans(n_clusters=10, random_state=42)
df['Cluster'] = kmeans.fit_predict(tags_matrix)
```

```
cluster_features = pd.get_dummies(df['Cluster'])
cluster_matrix = csr_matrix(cluster_features.values)
```

Figure 30: Dataset 1 - Weighted Hybrid Approach (Cluster-Based Features Part)

Figure 30 shows the implementation of cluster-based features where every similar attraction is grouped together into similar clusters.

```
weight_content = 0.6
weight_collaborative = 0.3
weight_cluster = 0.1
hybrid_matrix = hstack([
    weight_content * content_matrix,
    weight_collaborative * user_item_approx_reduced_expanded_sparse,
    weight_cluster * cluster_matrix
])
```

```
y = df['User_Rating']

X = hybrid_matrix
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
ridge_model = Ridge(alpha=1.0, random_state=42)
ridge_model.fit(X_train, y_train)

y_pred = ridge_model.predict(X_test)

rmse_hybrid = np.sqrt(mean_squared_error(y_test, y_pred))
mae_hybrid = mean_absolute_error(y_test, y_pred)
r2_hybrid = r2_score(y_test, y_pred)

print(f"Weighted Hybrid Model with KMeans Clustering - RMSE: {rmse_hybrid:.4f}")
print(f"Weighted Hybrid Model with KMeans Clustering - MAE: {mae_hybrid:.4f}")
print(f"Weighted Hybrid Model with KMeans Clustering - R^2 Score: {r2_hybrid:.4f}")
```

Figure 31: Dataset 1 - Weighted Hybrid Approach (Weights Aggregation + Model Training)

Figure 31 shows the application of weights to each individual type of filtering (Content-Based, Collaborative, and Cluster-Based). The matrixes with weights are combined to form a hybrid matrix which is passed to the ridge regression model. Data is split into the ratio of 80/20 where 80% is the training data and 20% is the testing data.

```
def generate_recommendations(user_id, ridge_model, hybrid_matrix, df, top_n=5):

    user_index = user_id - 1
    user_features = hybrid_matrix[user_index].toarray()

    attraction_features = hybrid_matrix.toarray()  # All attraction features
    predicted_ratings = ridge_model.predict(attraction_features)

    recommendations = pd.DataFrame({
        'Attraction': df['Name'].values,

    })

    recommendations = recommendations.drop_duplicates(subset='Attraction', keep='first')

    return recommendations.head(top_n)
```

Figure 32: Dataset 1 - Weighted Hybrid Approach Generate Recommendations

Figure 32 shows the generate_recommendations function which generates recommendations.

6.3.2 Dataset 2

This section shows code for the implementation of weighted hybrid approach on dataset 3.

```
df = pd.read_csv('final_accomodations_data.csv')
user_ratings_df = pd.read_csv('user_ratings_accommodations.csv')

df = df.merge(user_ratings_df, left_on='accommodation_id', right_on='Accommodation_ID', how='inner')
```

Figure 33: Dataset 2 - Weighted Hybrid Approach (Loading Data)

Figure 34 shows the content-based filtering module.

```
df['Rating'] = pd.to_numeric(df['Rating'], errors='coerce')
df['Rating'].fillna(df['Rating'].mean(), inplace=True)

df['Account Name'] = df['Account Name'].fillna('')

vectorizer = TfidfVectorizer(stop_words='english', max_features=500)
account_name_matrix = vectorizer.fit_transform(df['Account Name'])

scaler = MinMaxScaler()
df[['Rating', 'Latitude', 'Longitude']] = scaler.fit_transform(df[['Rating', 'Latitude', 'Longitude']])

content_matrix = hstack([account_name_matrix, csr_matrix(df[['Rating', 'Latitude', 'Longitude']].values)])
```

Figure 34: Dataset 2 - Weighted Hybrid Approach (Content-Based Filtering Part)

Figure 35 shows the collaborative filtering module.

```
n_components = 20
content_svd = TruncatedSVD(n_components=n_components, random_state=42)
content_matrix_reduced = content_svd.fit_transform(content_matrix)

user_item_matrix = user_ratings_df.pivot(index='User_ID', columns='Accommodation_ID', values='User_Rating').fillna(0)
user_item_matrix = csr_matrix(user_item_matrix.values)

svd = TruncatedSVD(n_components=n_components, random_state=42)
user_item_reduced = svd.fit_transform(user_item_matrix)

user_item_approx_reduced_expanded = np.tile(user_item_reduced, (int(np.ceil(content_matrix.shape[0] / user_item_reduced.shape[0]
user_item_approx_reduced_expanded = user_item_approx_reduced_expanded[:content_matrix.shape[0], :]

user_item_approx_reduced_expanded_sparse = csr_matrix(user_item_approx_reduced_expanded)
```

Figure 35: Dataset 2 - Weighted Hybrid Approach (Collaborative Filtering Part)

Figure 36 shows the cluster-based approach on dataset 2.

```
kmeans = KMeans(n_clusters=10, random_state=42)
df['Cluster'] = kmeans.fit_predict(account_name_matrix)

cluster_features = pd.get_dummies(df['Cluster'])
cluster_matrix = csr_matrix(cluster_features.values)
```

Figure 36: Dataset 2 - Weighted Hybrid Approach (Cluster-Based features Part)

```python
weight_content = 0.6
weight_collaborative = 0.3
weight_cluster = 0.1

hybrid_matrix = hstack([
    weight_content * content_matrix,
    weight_collaborative * user_item_approx_reduced_expanded_sparse,
    weight_cluster * cluster_matrix
])

y = df['User_Rating'].values

X_train, X_test, y_train, y_test = train_test_split(hybrid_matrix, y, test_size=0.2, random_state=42)

ridge_model = Ridge(alpha=1.0, random_state=42)
ridge_model.fit(X_train.toarray(), y_train)

y_pred = ridge_model.predict(X_test.toarray())

rmse_hybrid = np.sqrt(mean_squared_error(y_test, y_pred))
mae_hybrid = mean_absolute_error(y_test, y_pred)
r2_hybrid = r2_score(y_test, y_pred)

print(f"Weighted Hybrid Model - RMSE: {rmse_hybrid:.4f}")
print(f"Weighted Hybrid Model - MAE: {mae_hybrid:.4f}")
print(f"Weighted Hybrid Model - R² Score: {r2_hybrid:.4f}")
```

Figure 37: Dataset 2 - Weighted Hybrid Approach (Weights Aggregation + Model Training)

Figure 37 shows the application of weights integration and model training.

```python
plt.figure(figsize=(10, 6))
plt.plot(range(1, n_components + 1), svd.singular_values_, marker='o', linestyle='--')
plt.xlabel('Latent Factor Index')
plt.ylabel('Singular Value')
plt.title('Scree Plot of Singular Values for Hybrid Model')
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
sns.histplot(y_test, bins=50, color='blue', alpha=0.5, label='Original Ratings')
sns.histplot(y_pred, bins=50, color='red', alpha=0.5, label='Predicted Ratings')
plt.xlabel('Rating Value')
plt.ylabel('Frequency')
plt.title('Distribution of Original and Predicted Ratings for Hybrid Model - Dataset 2')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.6, color='green')
plt.xlabel('Actual Ratings')
plt.ylabel('Predicted Ratings')
plt.title('Actual vs Predicted Ratings for Hybrid Model')
plt.grid(True)
plt.show()
```

Figure 38: Dataset 2 - Weighted Hybrid Approach (Evaluation Graphs)

```python
def generate_recommendations(user_id, ridge_model, hybrid_matrix, df, top_n=5):

    df = df.drop_duplicates(subset=['accommodation_id'], keep='first')

    user_index = user_id - 1
    user_features = hybrid_matrix[user_index].toarray()

    repeated_user_features = np.tile(user_features, (df.shape[0], 1))


    recommendations = pd.DataFrame({
        'Accommodation': df['Account Name'].values,
    })


    return recommendations.head(top_n)
```

Figure 39: Dataset 2 - Weighted Hybrid Approach (Generate Recommendations Function)

Figure 39 shows the generate_recommendations () function which generates the recommendations.

6.3.3 Dataset 3

The below figures show code configurations for the implementation of the weighted hybrid approach on the third dataset.

```python
final_tourist_data = pd.read_csv("tourist_attractions_ireland_full_2.csv")
tourist_data_with_ratings = pd.read_csv("tourist_data_with_ratings_dataset3.csv")

df = final_tourist_data.merge(tourist_data_with_ratings, left_on='Name', right_on='Attraction', how='inner')
```

Figure 40: Dataset 3 - Weighted Hybrid Approach (Loading Data)

```python
df['Rating'] = pd.to_numeric(df['Rating'], errors='coerce')
df['Rating'].fillna(df['Rating'].mean(), inplace=True)

df['Tags'] = df['Tags'].fillna('')

vectorizer = TfidfVectorizer(stop_words='english')
tags_matrix = vectorizer.fit_transform(df['Tags'])

scaler = MinMaxScaler()
df[['Rating', 'Latitude', 'Longitude']] = scaler.fit_transform(df[['Rating', 'Latitude', 'Longitude']])

content_matrix = hstack([tags_matrix, csr_matrix(df[['Rating', 'Latitude', 'Longitude']].values)])
```

Figure 41: Dataset 3 - Weighted Hybrid Approach (Content-Based Filtering Part)

Figure 41 shows the code for the content-based filtering part on dataset 3.

```python
tourist_data_with_ratings = tourist_data_with_ratings.groupby(['User_ID', 'Attraction']).agg({'User_Rating': 'mean'}).reset_index
user_item_matrix = tourist_data_with_ratings.pivot(index='User_ID', columns='Attraction', values='User_Rating').fillna(0).values
user_item_matrix = csr_matrix(user_item_matrix)

n_components = 20
svd = TruncatedSVD(n_components=n_components, random_state=42)
user_item_reduced = svd.fit_transform(user_item_matrix)

user_item_approx_reduced_expanded = np.tile(user_item_reduced, (int(np.ceil(content_matrix.shape[0] / user_item_reduced.shape[0])
user_item_approx_reduced_expanded = user_item_approx_reduced_expanded[:content_matrix.shape[0], :]

user_item_approx_reduced_expanded_sparse = csr_matrix(user_item_approx_reduced_expanded)
```

Figure 42: Dataset 3 - Weighted Hybrid Approach (Collaborative FIltering Part)

```
kmeans = KMeans(n_clusters=10, random_state=42)
df['Cluster'] = kmeans.fit_predict(tags_matrix)

cluster_features = pd.get_dummies(df['Cluster'])
cluster_matrix = csr_matrix(cluster_features.values)
```

Figure 43: Dataset 3 - Weighted Hybrid Approach (Cluster-Based Features)

Figure 43 shows the cluster-based filtering on dataset 3.

```
weight_content = 0.6
weight_collaborative = 0.3
weight_cluster = 0.1

hybrid_matrix = hstack([
    weight_content * content_matrix,
    weight_collaborative * user_item_approx_reduced_expanded_sparse,
    weight_cluster * cluster_matrix
])

y = df['User_Rating']

X = hybrid_matrix
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

ridge_model = Ridge(alpha=1.0, random_state=42)
ridge_model.fit(X_train, y_train)

y_pred = ridge_model.predict(X_test)

rmse_hybrid = np.sqrt(mean_squared_error(y_test, y_pred))
mae_hybrid = mean_absolute_error(y_test, y_pred)
r2_hybrid = r2_score(y_test, y_pred)

print(f"Weighted Hybrid Model - RMSE: {rmse_hybrid:.4f}")
print(f"Weighted Hybrid Model - MAE: {mae_hybrid:.4f}")
print(f"Weighted Hybrid Model - R² Score: {r2_hybrid:.4f}")
```

Figure 44: Dataset 3 - Weighted Hybrid Approach (Weights Aggregation + Model Training)

Figure 44 shows the application of weights aggregation and model training.

```
plt.figure(figsize=(10, 6))
plt.plot(range(1, n_components + 1), svd.singular_values_, marker='o', linestyle='--')
plt.xlabel('Latent Factor Index')
plt.ylabel('Singular Value')
plt.title('Scree Plot of Singular Values')
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.hist(y_test, bins=50, alpha=0.5, label='Original Ratings', color='blue')
plt.hist(y_pred, bins=50, alpha=0.5, label='Predicted Ratings', color='red')
plt.xlabel('Rating Value')
plt.ylabel('Frequency')
plt.title('Distribution of Original and Predicted Ratings - Dataset 3 (Weighted Hybrid Approach)')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.6, color='green')
plt.xlabel('Actual Ratings')
plt.ylabel('Predicted Ratings')
plt.title('Actual vs Predicted Ratings')
plt.grid(True)
plt.show()
```

Figure 45: Dataset 3 - Weighted Hybrid Approach (Evaluation Plots)

```
def generate_recommendations(user_id, ridge_model, hybrid_matrix, df, top_n=5):

    df = df.drop_duplicates(subset=['Name'], keep='first')

    user_index = user_id - 1
    user_features = hybrid_matrix[user_index].toarray()

    repeated_user_features = np.tile(user_features, (df.shape[0], 1))

    recommendations = pd.DataFrame({
        'Attraction': df['Name'].values,

    })

    return recommendations.head(top_n)
```

Figure 46: Dataset 3 - Weighted Hybrid Approach (Generate recommendations Function)

7 References

Tourism Activities and Attractions Dataset
Government of Ireland. (n.d.). Tourism activities and attractions. Retrieved from https://data.gov.ie/dataset/tourism-activities-and-attractions

Accommodation Dataset
Fáilte Ireland. (n.d.). Open data API - Accommodation. Retrieved from https://failteireland.azure-api.net/opendata-api/v2/accommodation/csv