ITCS 6162: Data Mining - Programming Assignment

In this assignment, you will explore data analysis, recommendation algorithms, and graph-based techniques using the MovieLens dataset. Your tasks will range from basic data exploration to advanced recommendation models, including:

- Data manipulation with pandas
- User-item collaborative filtering
- Similarity-based recommendation models
- A Pixie-inspired Graph-based recommendation using adjacency lists with weighted random walks (without using NetworkX)

Dataset Files:

- **`u.data`** : User-movie ratings (`user_id  movie_id  rating  timestamp`)
- **`u.item`** : Movie metadata (`movie_id | title | release date | IMDB_website`)
- **`u.user`** : User demographics (`user_id | age | gender | occupation | zip_code`)

Part 1: Exploring and Cleaning Data

Inspecting the Dataset Format

The dataset is not in a traditional CSV format. To examine its structure, use the following shell command to display the first 10 lines of the file:

!head <file_name>

In the cells given below. Write the code to read the files.

In [1]:
```
# Please modify this to be the correct path if not using Google Drive (Uncom

# from google.colab import drive
# drive.mount('/content/drive')
#
# !head "drive/MyDrive/ProgrammingAssignment1KDD/u.data"
#
# !head "drive/MyDrive/ProgrammingAssignment1KDD/u.item"
#
# !head "drive/MyDrive/ProgrammingAssignment1KDD/u.user"
```

```
Mounted at /content/drive
196     242     3       881250949
186     302     3       891717742
22      377     1       878887116
244     51      2       880606923
166     346     1       886397596
298     474     4       884182806
115     265     2       881171488
253     465     5       891628467
305     451     3       886324817
6       86      3       883603013
1|Toy Story (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Toy%20Stor
y%20(1995)|0|0|0|1|1|1|0|0|0|0|0|0|0|0|0|0|0|0|0
2|GoldenEye (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?GoldenEye%2
0(1995)|0|1|1|0|0|0|0|0|0|0|0|0|0|0|0|0|1|0|0
3|Four Rooms (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Four%20Roo
ms%20(1995)|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|1|0|0
4|Get Shorty (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Get%20Shor
ty%20(1995)|0|1|0|0|0|1|0|0|1|0|0|0|0|0|0|0|0|0|0
5|Copycat (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Copycat%20(19
95)|0|0|0|0|0|0|1|0|1|0|0|0|0|0|0|0|1|0|0
6|Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)|01-Jan-1995||http://u
s.imdb.com/Title?Yao+a+yao+yao+dao+waipo+qiao+(1995)|0|0|0|0|0|0|0|0|1|0|0|0
|0|0|0|0|0|0|0
7|Twelve Monkeys (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Twelv
e%20Monkeys%20(1995)|0|0|0|0|0|0|0|0|1|0|0|0|0|0|0|1|0|0|0
8|Babe (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Babe%20(1995)|0|
0|0|0|1|1|0|0|1|0|0|0|0|0|0|0|0|0|0
9|Dead Man Walking (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Dea
d%20Man%20Walking%20(1995)|0|0|0|0|0|0|0|0|1|0|0|0|0|0|0|0|0|0|0
10|Richard III (1995)|22-Jan-1996||http://us.imdb.com/M/title-exact?Richard%
20III%20(1995)|0|0|0|0|0|0|0|0|1|0|0|0|0|0|0|0|0|1|0
1|24|M|technician|85711
2|53|F|other|94043
3|23|M|writer|32067
4|24|M|technician|43537
5|33|F|other|15213
6|42|M|executive|98101
7|57|M|administrator|91344
8|36|M|administrator|05201
9|29|M|student|01002
10|53|M|lawyer|90703
```

In [80]:

```python
# Modify this path to be the path to data folder. (In the repository, it is

# ===============
# PLEASE MODIFY
# ===============

#dataPath = "./data/"
dataPath = "drive/MyDrive/ProgrammingAssignment1KDD/" # My experiments were

dataFile = dataPath + "u.data"
itemFile = dataPath + "u.item"
userFile = dataPath + "u.user"
```

In [3]:
```
# u.data
dataString = ""
with open(dataFile, 'r') as file:
  dataString = file.read()

print(dataString[0:500] + "...")
```

```
196	242	3	881250949
186	302	3	891717742
22	377	1	878887116
244	51	2	880606923
166	346	1	886397596
298	474	4	884182806
115	265	2	881171488
253	465	5	891628467
305	451	3	886324817
6	86	3	883603013
62	257	2	879372434
286	1014	5	879781125
200	222	5	876042340
210	40	3	891035994
224	29	3	888104457
303	785	3	879485318
122	387	5	879270459
194	274	2	879539794
291	1042	4	874834944
234	1184	2	892079237
119	392	4	886176814
167	486	4	892738452
299	144	4	877881320
291	118	2	874833878
308	1	4	887736532
95	546	...
```

In [4]:
```
# u.item
itemString = ""
with open(itemFile, 'r', encoding='latin-1') as file:
  itemString = file.read()

print(itemString[0:700] + "...")
```

```
1|Toy Story (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Toy%20Stor
y%20(1995)|0|0|0|1|1|1|0|0|0|0|0|0|0|0|0|0|0|0|0
2|GoldenEye (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?GoldenEye%2
0(1995)|0|1|1|0|0|0|0|0|0|0|0|0|0|0|0|0|1|0|0
3|Four Rooms (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Four%20Roo
ms%20(1995)|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|1|0|0
4|Get Shorty (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Get%20Shor
ty%20(1995)|0|1|0|0|0|1|0|0|1|0|0|0|0|0|0|0|0|0|0
5|Copycat (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Copycat%20(19
95)|0|0|0|0|0|0|1|0|1|0|0|0|0|0|0|0|1|0|0
6|Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)|01-Jan-1995||http://u
s.imdb.c...
```

In [5]:
```
# u.user
userString = ""
```

```
with open(userFile, 'r') as file:
  userString = file.read()

print(userString[0:500] + "...")
```

```
1|24|M|technician|85711
2|53|F|other|94043
3|23|M|writer|32067
4|24|M|technician|43537
5|33|F|other|15213
6|42|M|executive|98101
7|57|M|administrator|91344
8|36|M|administrator|05201
9|29|M|student|01002
10|53|M|lawyer|90703
11|39|F|other|30329
12|28|F|other|06405
13|47|M|educator|29206
14|45|M|scientist|55106
15|49|F|educator|97301
16|21|M|entertainment|10309
17|30|M|programmer|06355
18|35|F|other|37212
19|40|M|librarian|02138
20|42|F|homemaker|95660
21|26|M|writer|30068
22|25|M|writer|40206
23...
```

Loading the Dataset with Pandas

Use **pandas** to load the dataset into a DataFrame for analysis. Follow these steps:

1. Import the necessary library: `pandas`.
2. Use `pd.read_csv()` (or an appropriate function) to read the dataset file.
3. Ensure the dataset is loaded with the correct delimiter (e.g., `','`, `'\t'`, `'|'`, or another separator if needed).
4. Select and display the first few rows using `.head()`.

Ensure that:

- The `ratings` dataset is read from `"u.data"` using tab (`'\t'`) as a separator and column names (`"user_id"` , `"movie_id"` , `"rating"` and `"timestamp"`).
- The `movies` dataset is read from `"u.item"` using `'|'` as a separator, use columns (`0` , `1` , `2`), encoding (`"latin-1"`) and name the columns (`movie_id` , `title` , and `release_date`).
- The `users` dataset is read from `"u.user"` using `'|'` as a separator, use columns (`0` , `1` , `2` , `3`) and name the columns (`user_id` , `age` , `gender` , and `occupation`).

In [6]:

```
import pandas as pd
# ratings
ratings_data = pd.read_csv(dataFile, sep='\t', names=["user_id", "movie_id",

print(ratings_data)
```

```
       user_id  movie_id  rating  timestamp
0          196       242       3  881250949
1          186       302       3  891717742
2           22       377       1  878887116
3          244        51       2  880606923
4          166       346       1  886397596
...        ...       ...     ...        ...
99995      880       476       3  880175444
99996      716       204       5  879795543
99997      276      1090       1  874795795
99998       13       225       2  882399156
99999       12       203       3  879959583

[100000 rows x 4 columns]
```

In [7]:

```
# movies
movies_data = pd.read_csv(itemFile, sep='|', encoding="latin-1", usecols=[0,

print(movies_data)
```

```
      movie_id                                      title release_date
0            1                           Toy Story (1995)  01-Jan-1995
1            2                           GoldenEye (1995)  01-Jan-1995
2            3                          Four Rooms (1995)  01-Jan-1995
3            4                          Get Shorty (1995)  01-Jan-1995
4            5                             Copycat (1995)  01-Jan-1995
...        ...                                        ...          ...
1677      1678                          Mat' i syn (1997)  06-Feb-1998
1678      1679                           B. Monkey (1998)  06-Feb-1998
1679      1680                       Sliding Doors (1998)  01-Jan-1998
1680      1681                        You So Crazy (1994)  01-Jan-1994
1681      1682  Scream of Stone (Schrei aus Stein) (1991)  08-Mar-1996

[1682 rows x 3 columns]
```

In [8]:
```python
# users
users_data = pd.read_csv(userFile, sep='|', usecols=[0, 1, 2, 3], names=["us

print(users_data)
```

```
     user_id  age gender     occupation
0          1   24      M     technician
1          2   53      F          other
2          3   23      M         writer
3          4   24      M     technician
4          5   33      F          other
..       ...  ...    ...            ...
938      939   26      F        student
939      940   32      M  administrator
940      941   20      M        student
941      942   48      F      librarian
942      943   22      M        student

[943 rows x 4 columns]
```

Note: As a **Bonus** task save the `ratings`, `movies` and `users` dataframe created into a `.csv` file format.

Hint: Use the `to_csv()` function in pandas to save these DataFrames as CSV files.

In [9]:
```python
ratings_csv = dataPath + "ratings.csv"
movies_csv = dataPath + "movies.csv"
users_csv = dataPath + "users.csv"

# ratings
ratings_data.to_csv(ratings_csv, index=False)

# movies
movies_data.to_csv(movies_csv, index=False)

# users
users_data.to_csv(users_csv, index=False)
```

Display the first 10 rows of each file.

In [10]:
```python
# ratings
# !head "drive/MyDrive/ProgrammingAssignment1KDD/ratings.csv"
```

```
user_id,movie_id,rating,timestamp
196,242,3,881250949
186,302,3,891717742
22,377,1,878887116
244,51,2,880606923
166,346,1,886397596
298,474,4,884182806
115,265,2,881171488
253,465,5,891628467
305,451,3,886324817
```

In [11]:
```
# movies

# !head "drive/MyDrive/ProgrammingAssignment1KDD/movies.csv"
```

```
movie_id,title,release_date
1,Toy Story (1995),01-Jan-1995
2,GoldenEye (1995),01-Jan-1995
3,Four Rooms (1995),01-Jan-1995
4,Get Shorty (1995),01-Jan-1995
5,Copycat (1995),01-Jan-1995
6,Shanghai Triad (Yao a yao yao dao waipo qiao) (1995),01-Jan-1995
7,Twelve Monkeys (1995),01-Jan-1995
8,Babe (1995),01-Jan-1995
9,Dead Man Walking (1995),01-Jan-1995
```

In [12]:
```
# users


# !head "drive/MyDrive/ProgrammingAssignment1KDD/users.csv"
```

```
user_id,age,gender,occupation
1,24,M,technician
2,53,F,other
3,23,M,writer
4,24,M,technician
5,33,F,other
6,42,M,executive
7,57,M,administrator
8,36,M,administrator
9,29,M,student
```

Data Cleaning and Exploration with Pandas

After loading the dataset, it’s important to clean and explore the data to ensure consistency and accuracy. Below are key **pandas** functions for cleaning and understanding the dataset.

1. Handle Missing Values

- `df.dropna()` – Removes rows with missing values.
- `df.fillna(value)` – Fills missing values with a specified value.

2. Remove Duplicates

- `df.drop_duplicates()` – Drops duplicate rows from the dataset.

3. Handle Incorrect Data Types

- `df.astype(dtype)` – Converts columns to the appropriate data type.

4. Filter Outliers (if applicable)

- `df[df['column_name'] > threshold]` – Filters rows based on a condition.

5. Rename Columns (if needed)

- `df.rename(columns={'old_name': 'new_name'})` – Renames columns for clarity.

6. Reset Index

- `df.reset_index(drop=True, inplace=True)` – Resets the index after cleaning.

Data Exploration Functions

To better understand the dataset, use these **pandas** functions:

- `df.shape` – Returns the number of rows and columns in the dataset.
- `df.nunique()` – Displays the number of unique values in each column.
- `df['column_name'].unique()` – Returns unique values in a specific column.

Example Usage in Pandas:

```python
import pandas as pd

# Load dataset
df = pd.read_csv("your_file.csv")

# Drop missing values
df_cleaned = df.dropna()

# Remove duplicate rows
df_cleaned = df_cleaned.drop_duplicates()

# Convert 'timestamp' column to datetime format
df_cleaned['timestamp'] = pd.to_datetime(df_cleaned['timestamp'])

# Display dataset shape
print("Dataset shape:", df_cleaned.shape)

# Display number of unique values in each column
print("Unique values per column:\n", df_cleaned.nunique())

# Display unique movie IDs
print("Unique movie IDs:", df_cleaned['movie_id'].unique()[:10])  # Show first 10 unique movie IDs
```

Note: The functions mentioned above are some of the widely used **pandas** functions for data cleaning and exploration. However, it is not necessary that all of these functions will be required in the exercises below. Use them as needed based on the dataset and the specific tasks.

Convert Timestamps into Readable dates.

In [13]:
```
# ratings
dirty_ratings = pd.read_csv(ratings_csv)
dirty_ratings['timestamp'] = pd.to_datetime(dirty_ratings['timestamp'], unit
dirty_ratings.reset_index(drop=True, inplace=True)
print(dirty_ratings.head())
```

```
   user_id  movie_id  rating           timestamp
0      196       242       3 1997-12-04 15:55:49
1      186       302       3 1998-04-04 19:22:22
2       22       377       1 1997-11-07 07:18:36
3      244        51       2 1997-11-27 05:02:03
4      166       346       1 1998-02-02 05:33:16
```

Check for Missing Values

In [21]:
```
# ratings
clean_ratings = dirty_ratings.drop_duplicates()

clean_ratings['user_id'].dropna()
clean_ratings['movie_id'].dropna()
clean_ratings = clean_ratings.fillna(0) # assume that a non-rating is 0.

clean_ratings.reset_index(drop=True, inplace=True)


print(clean_ratings.head())
```

```
   user_id  movie_id  rating           timestamp
0      196       242       3 1997-12-04 15:55:49
1      186       302       3 1998-04-04 19:22:22
2       22       377       1 1997-11-07 07:18:36
3      244        51       2 1997-11-27 05:02:03
4      166       346       1 1998-02-02 05:33:16
```

In [23]:
```
# movies
dirty_movies = pd.read_csv(movies_csv)

clean_movies = dirty_movies.drop_duplicates()
clean_movies['movie_id'].dropna()
clean_movies = clean_movies.fillna(0)
clean_movies.reset_index(drop=True, inplace=True)


print(clean_movies.head())
```

```
   movie_id              title release_date
0         1   Toy Story (1995)  01-Jan-1995
1         2   GoldenEye (1995)  01-Jan-1995
2         3  Four Rooms (1995)  01-Jan-1995
3         4  Get Shorty (1995)  01-Jan-1995
4         5     Copycat (1995)  01-Jan-1995
```

In [24]:
```
# users
dirty_users = pd.read_csv(users_csv)
```

```python
clean_users = dirty_users.drop_duplicates()
clean_users['user_id'].dropna()
clean_users.reset_index(drop=True, inplace=True)

print(clean_users.head())
```

```
   user_id  age gender  occupation
0        1   24      M  technician
1        2   53      F       other
2        3   23      M      writer
3        4   24      M  technician
4        5   33      F       other
```

Print the total number of users, movies, and ratings.

In [26]:
```python
print(f"Total Users: { clean_users.shape[0]}")
print(f"Total Movies: {clean_movies.shape[0]}")
print(f"Total Ratings: {clean_ratings.shape[0]}")
```

```
Total Users: 943
Total Movies: 1682
Total Ratings: 100000
```

Part 2: Collaborative Filtering-Based Recommendation

Create a User-Item Matrix

Instructions for Creating a User-Movie Rating Matrix

In this exercise, you will create a user-movie rating matrix using **pandas**. This matrix will represent the ratings that users have given to different movies.

1. **Dataset Overview**:
   The dataset has already been loaded. It includes the following key columns:
   - `user_id`: The ID of the user.
   - `movie_id`: The ID of the movie.
   - `ratings`: The rating the user gave to the movie.
2. **Create the User-Movie Rating Matrix**:
   Use the **`pivot()`** function in **pandas** to reshape the data. Your goal is to create a matrix where:
   - Each **row** represents a **user**.
   - Each **column** represents a **movie**.
   - Each **cell** contains the **rating** that the user has given to the movie.

   Specify the following parameters for the `pivot()` function:

- `index`: The `user_id` column (this will define the rows).
- `columns`: The `movie_id` column (this will define the columns).
- `values`: The `rating` column (this will fill the matrix with ratings).

3. **Inspect the Matrix**:
   After creating the matrix, examine the first few rows of the resulting matrix to ensure it has been constructed correctly.

4. **Handle Missing Values**:
   It's likely that some users have not rated every movie, resulting in `NaN` values in the matrix. You will need to handle these missing values. Consider the following options:

   - **Fill with 0**: If you wish to represent missing ratings as zeros (indicating no rating).
   - **Fill with the average rating**: Alternatively, replace missing values with the average rating for each movie.

Create the user-movie rating matrix using the `pivot()` function.

In [31]:
```
userToMovieMatrix = clean_ratings.pivot( index='user_id', columns='movie_id'
```

Display the matrix to verify the transformation.

In [32]:
```
print(userToMovieMatrix.head())

print(f"\n\nUser 22 Rating of 377: {userToMovieMatrix[377][22]}") # User 22
```

```
movie_id  1     2     3     4     5     6     7     8     9     10    ...  \
user_id                                                             ...   
1          5.0   3.0   4.0   3.0   3.0   5.0   4.0   1.0   5.0   3.0  ...   
2          4.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   2.0  ...   
3          0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ...   
4          0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ...   
5          4.0   3.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ...   

movie_id  1673  1674  1675  1676  1677  1678  1679  1680  1681  1682  
user_id                                                               
1          0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  
2          0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  
3          0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  
4          0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  
5          0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  

[5 rows x 1682 columns]


User 22 Rating of 377: 1.0
```

User-Based Collaborative Filtering Recommender System

Objective

In this task, you will implement a **user-based collaborative filtering** movie recommendation system using the **Movie dataset**. The goal is to recommend movies to a user based on the preferences of similar users.

Step 1: Import Required Libraries

Before starting, ensure you have the necessary libraries installed. Use the following imports:

```python
import pandas as pd  # For handling data
import numpy as np   # For numerical computations
from sklearn.metrics.pairwise import cosine_similarity  # For
computing user similarity
```

Step 2: Compute User-User Similarity

- We will use **cosine similarity** to measure how similar each pair of users is based on their movie ratings.
- Since `cosine_similarity` does not handle missing values (NaN), replace them with `0` before computation.

Instructions:

1. Fill missing values with `0` using `.fillna(0)`.
2. Compute similarity using `cosine_similarity()`.
3. Convert the result into a **Pandas DataFrame**, with users as both row and column labels.

Hint:

You can achieve this using the following approach:

```python
user_similarity = cosine_similarity(user_movie_matrix.fillna(0))
user_sim_df = pd.DataFrame(user_similarity,
index=user_movie_matrix.index, columns=user_movie_matrix.index)
```

Step 3: Implement the Recommendation Function

Now, implement the function `recommend_movies_for_user(user_id, num=5)` to recommend movies for a given user.

Function Inputs:

- `user_id`: The target user for whom we need recommendations.
- `num`: The number of movies to recommend (default is 5).

Function Steps:

1. Find **similar users**:
   - Retrieve the similarity scores for the given `user_id`.
   - Sort them in **descending** order (highest similarity first).
   - Exclude the user themselves.
2. Get the **movie ratings** from these similar users.
3. Compute the **average rating** for each movie based on these users' preferences.
4. Sort the movies in **descending order** based on the computed average ratings.
5. Retrieve the **top `num` recommended movies**.
6. Map **movie IDs** to their **titles** using the `movies` DataFrame.
7. Return the results as a **Pandas DataFrame** with rankings.

Step 4: Return the Final Recommendation List

Your function should return a **DataFrame** structured as follows:

Ranking	Movie Name
1	Movie A
2	Movie B
3	Movie C
4	Movie D
5	Movie E

Hint: Your final DataFrame should be created like this:

```
result_df = pd.DataFrame({
    'Ranking': range(1, num+1),
    'Movie Name': movie_names
})
result_df.set_index('Ranking', inplace=True)
```

Example: User-Based Collaborative Filtering

```
recommend_movies_for_user(10, num = 5)
```

Output:

```
| Ranking | Movie Name                      |
|---------|---------------------------------|
| 1       | In the Company of Men (1997)    |
| 2       | Misérables, Les (1995)          |
| 3       | Thin Blue Line, The (1988)      |
```

```
| 4       | Braindead (1992)                |
| 5       | Boys, Les (1997)                |
```

In [54]:
```
# Code the function here

import pandas as pd  # For handling data
import numpy as np   # For numerical computations
from sklearn.metrics.pairwise import cosine_similarity

userSimilarity = cosine_similarity(userToMovieMatrix.fillna(0))
userSimilarityMtx = pd.DataFrame(userSimilarity, index=userToMovieMatrix.inc


def recommend_movies_for_user(userID, num):
  similarUsers = userSimilarityMtx[userID].sort_values(ascending=False).drop
  similarUsers = similarUsers[similarUsers > 0]
  #print(similarUsers)
  userRatings = userToMovieMatrix.loc[similarUsers.index]

  #averageMovieRatings = userRatings.mean()

  # Weighted mean (by similarity)
  weightedRatings = userRatings.T.mul(similarUsers, axis=1)
  sumWeights = similarUsers.sum()
  weightedAverageRatings = weightedRatings.sum(axis=1) / sumWeights

  # Don't want to recommend a movie that they have seen
  ratedByUser = userToMovieMatrix.loc[userID][userToMovieMatrix.loc[userID]
  unratedMovies = weightedAverageRatings.drop(ratedByUser, errors='ignore')


  topMovies = unratedMovies.sort_values(ascending=False)[0: num]
  movieNames = clean_movies[clean_movies['movie_id'].isin(topMovies.index)][




  result = pd.DataFrame({
      'Ranking': range(1, num+1),
      'Movie Name': movieNames

  })
  result.set_index('Ranking', inplace = True)
  return result;



print(recommend_movies_for_user(11, 5))
print(recommend_movies_for_user(10, 5))
print(recommend_movies_for_user(9, 10))
```

```
                              Movie Name
Ranking
1                       Toy Story (1995)
2                       Star Wars (1977)
3        Empire Strikes Back, The (1980)
4         Raiders of the Lost Ark (1981)
5              Return of the Jedi (1983)
                              Movie Name
Ranking
1                   Fugitive, The (1993)
2        Empire Strikes Back, The (1980)
3              Return of the Jedi (1983)
4              Back to the Future (1985)
5                Schindler's List (1993)
                               Movie Name
Ranking
1                        Toy Story (1995)
2                     Pulp Fiction (1994)
3        Silence of the Lambs, The (1991)
4                            Fargo (1996)
5                   Godfather, The (1972)
6         Empire Strikes Back, The (1980)
7          Raiders of the Lost Ark (1981)
8               Return of the Jedi (1983)
9                          Contact (1997)
10                          Scream (1996)
```

Code the function here## **Item-Based Collaborative Filtering Recommender System**

Objective

In this task, you will implement an **item-based collaborative filtering** recommendation system using the **Movie dataset**. The goal is to recommend movies similar to a given movie based on user rating patterns.

Step 1: Import Required Libraries

Although we have done this part already in the previous task but just to emphasize the importance reiterrating this part.

Before starting, ensure you have the necessary libraries installed. Use the following imports:

```python
import pandas as pd  # For handling data
import numpy as np   # For numerical computations
from sklearn.metrics.pairwise import cosine_similarity  # For computing item similarity
```

Step 2: Compute Item-Item Similarity

- We will use **cosine similarity** to measure how similar each pair of movies is based on their user ratings.
- Since `cosine_similarity` does not handle missing values (NaN), replace them with `0` before computation.
- Unlike user-based filtering, we need to **transpose** (`.T`) the `user_movie_matrix` because we want similarity between movies (columns) instead of users (rows).

Instructions:

1. Transpose the user-movie matrix using `.T` to make movies the rows.
2. Fill missing values with `0` using `.fillna(0)`.
3. Compute similarity using `cosine_similarity()`.
4. Convert the result into a **Pandas DataFrame**, with movies as both row and column labels.

Hint:

You can achieve this using the following approach:

```
item_similarity = cosine_similarity(user_movie_matrix.T.fillna(0))
item_sim_df = pd.DataFrame(item_similarity,
index=user_movie_matrix.columns, columns=user_movie_matrix.columns)
```

Step 3: Implement the Recommendation Function

Now, implement the function `recommend_movies(movie_name, num=5)` to recommend movies similar to a given movie.

Function Inputs:

- `movie_name`: The target movie for which we need recommendations.
- `num`: The number of similar movies to recommend (default is 5).

Function Steps:

1. Find the **movie_id** corresponding to the given `movie_name` in the `movies` DataFrame.
2. If the movie is not found, return an appropriate message.
3. Extract the **similarity scores** for this movie from `item_sim_df`.
4. Sort the movies in **descending order** based on similarity (excluding the movie itself).
5. Retrieve the **top `num` similar movies**.
6. Map **movie IDs** to their **titles** using the `movies` DataFrame.
7. Return the results as a **Pandas DataFrame** with rankings.

Step 4: Return the Final Recommendation List

Your function should return a **DataFrame** structured as follows:

Ranking	Movie Name
1	Movie A
2	Movie B
3	Movie C
4	Movie D
5	Movie E

Hint: Your final DataFrame should be created like this:

```
result_df = pd.DataFrame({
    'ranking': range(1, num+1),
    'movie_name': movie_names
})
result_df.set_index('ranking', inplace=True)
```

Example: Item-Based Collaborative Filtering

```
recommend_movies("Jurassic Park (1993)", num=5)
```

Output:

```
| Ranking | Movie Name                                |
|---------|-------------------------------------------|
| 1       | Top Gun (1986)                            |
| 2       | Empire Strikes Back, The (1980)           |
| 3       | Raiders of the Lost Ark (1981)            |
| 4       | Indiana Jones and the Last Crusade (1989)|
| 5       | Speed (1994)                              |
```

In [52]:

```
# Code the function here
import pandas as pd  # For handling data
import numpy as np   # For numerical computations
from sklearn.metrics.pairwise import cosine_similarity  # For computing item

item_similarity = cosine_similarity(userToMovieMatrix.T.fillna(0))
item_sim_df = pd.DataFrame(item_similarity, index=userToMovieMatrix.columns,


def recommend_movies(movieName, num):
  movieID = clean_movies.loc[clean_movies['title'] == movieName]['movie_id']
  if (len(movieID) <= 0):
    print(f'Movie, {movieName}, not found')
    return;
  movieID = movieID[0]


  similarMovies = item_sim_df[movieID].sort_values( ascending=False).drop(mo
  topMovies = similarMovies[0:num]
```

```python
    topMovieNames = clean_movies[clean_movies['movie_id'].isin(topMovies.index


    result = pd.DataFrame({
        'Ranking': range(1, num+1),
        'Movie Title': topMovieNames
    })
    result.set_index('Ranking', inplace=True)
    return result


print(recommend_movies('Return of the Jedi (1983)', 5))
print(recommend_movies('Empire Strikes Back, The (1980)', 5))
print(recommend_movies("Jurassic Park (1993)", num=5))
```

```
                             Movie Title
Ranking
1                       Toy Story (1995)
2                       Star Wars (1977)
3            Independence Day (ID4) (1996)
4          Empire Strikes Back, The (1980)
5           Raiders of the Lost Ark (1981)
                                       Movie Title
Ranking
1                                 Star Wars (1977)
2                   Raiders of the Lost Ark (1981)
3                           Terminator, The (1984)
4                        Back to the Future (1985)
5         Indiana Jones and the Last Crusade (1989)
                                       Movie Title
Ranking
1                                   Top Gun (1986)
2                  Empire Strikes Back, The (1980)
3                   Raiders of the Lost Ark (1981)
4         Indiana Jones and the Last Crusade (1989)
5                                     Speed (1994)
```

Part 3: Graph-Based Recommender (Pixie-Inspired Algorithm)

Adjacency List

Objective

In this task, you will preprocess the Movie dataset and construct a **graph representation** where:

- **Users** are connected to the movies they have rated.
- **Movies** are connected to users who have rated them.

This graph structure will help in exploring **user-movie relationships** for recommendations.

Step 1: Merge Ratings with Movie Titles

Since we have **movie IDs** in the ratings dataset but need human-readable movie titles, we will:

1. Merge the `ratings` DataFrame with the `movies` DataFrame using the `'movie_id'` column.
2. This allows each rating to be associated with a **movie title**.

Hint:

Use the following Pandas operation to merge:

```
ratings = ratings.merge(movies, on='movie_id')
```

Step 2: Aggregate Ratings

Since multiple users may rate the same movie multiple times, we:

1. Group the dataset by `['user_id', 'movie_id', 'title']`.
2. Compute the **mean rating** for each movie by each user.
3. Reset the index to ensure we maintain a clean DataFrame structure.

Hint:

Use `groupby()` and `mean()` as follows:

```
ratings = ratings.groupby(['user_id', 'movie_id', 'title'])
['rating'].mean().reset_index()
```

Step 3: Normalize Ratings

Since different users have different rating biases, we normalize ratings by:

1. **Computing each user's mean rating**.
2. **Subtracting the mean rating** from each individual rating.

Instructions:

- Use `groupby('user_id')` to group ratings by users.
- Apply `transform(lambda x: x - x.mean())` to adjust ratings.

Hint:

Normalize ratings using:

```
ratings['rating'] = ratings.groupby('user_id')
['rating'].transform(lambda x: x - x.mean())
```

This ensures each user's ratings are centered around zero, making similarity calculations fairer.

Step 4: Construct the Graph Representation

We represent the user-movie interactions as an **undirected graph** using an **adjacency list**:

- Each **user** is a node connected to movies they rated.
- Each **movie** is a node connected to users who rated it.

Graph Construction Steps:

1. Initialize an empty dictionary `graph = {}`.
2. Iterate through the **ratings dataset**.
3. For each `user_id` and `movie_id` pair:
    - Add the movie to the user's set of connections.
    - Add the user to the movie's set of connections.

Hint:

The following code builds the graph:

```
graph = {}
for _, row in ratings.iterrows():
    user, movie = row['user_id'], row['movie_id']
    if user not in graph:
        graph[user] = set()
    if movie not in graph:
        graph[movie] = set()
    graph[user].add(movie)
    graph[movie].add(user)
```

This results in a **bipartite graph**, where:

- **Users** are connected to multiple movies.
- **Movies** are connected to multiple users.

Step 5: Understanding the Graph

- **Nodes** in the graph represent **users and movies**.

- **Edges** exist between a user and a movie **if the user has rated the movie**.
- This structure allows us to find **users with similar movie tastes** and **movies frequently watched together**.

Exploring the Graph

- **Find a user's rated movies:**

```
user_id = 1
print(graph[user_id])  # Movies rated by user 1
```

- **Find users who rated a movie:**

```
movie_id = 50
print(graph[movie_id])  # Users who rated movie 50
```

In [56]:

```
# Code the function here

ratings = clean_ratings.merge(clean_movies, on='movie_id', how='left').drop_
ratings = ratings.groupby(['user_id', 'movie_id', 'title'])['rating'].mean()
ratings['rating'] = ratings.groupby('user_id')['rating'].transform(lambda x:

graph = {}
for _, row in ratings.iterrows():
    user, movie = row['user_id'], row['movie_id']
    if user not in graph:
        graph[user] = set()
    if movie not in graph:
        graph[movie] = set()
    graph[user].add(movie)
    graph[movie].add(user)

def moviesRatedByUser(userID):
  return graph[userID]
def usersWhoRatedMovie(movieID):
  return graph[movieID]
```

In [57]:

```
print(moviesRatedByUser(12))

print(usersWhoRatedMovie(300))
```

```
{1, 514, 4, 6, 7, 521, 10, 11, 522, 13, 14, 15, 16, 524, 18, 527, 532, 533,
24, 537, 538, 28, 29, 542, 543, 548, 551, 42, 43, 556, 557, 559, 560, 49, 5
0, 561, 566, 567, 58, 59, 60, 62, 64, 577, 69, 71, 72, 73, 583, 76, 588, 59
1, 592, 82, 84, 88, 601, 90, 603, 92, 605, 94, 606, 96, 97, 98, 99, 610, 61
3, 106, 618, 109, 110, 622, 115, 627, 117, 629, 119, 630, 121, 632, 127, 63
9, 640, 130, 643, 132, 133, 135, 138, 654, 143, 144, 145, 655, 151, 663, 66
4, 665, 666, 156, 157, 669, 159, 671, 161, 673, 168, 170, 682, 172, 684, 17
4, 175, 686, 177, 178, 690, 180, 693, 186, 191, 194, 195, 196, 708, 707, 71
0, 200, 201, 202, 203, 204, 715, 207, 213, 214, 215, 216, 727, 218, 221, 22
2, 735, 737, 226, 228, 233, 234, 745, 747, 238, 239, 753, 242, 754, 246, 75
8, 249, 250, 251, 763, 253, 256, 259, 773, 774, 264, 267, 268, 271, 272, 78
5, 276, 788, 279, 280, 282, 795, 796, 288, 291, 293, 805, 806, 297, 299, 30
0, 301, 303, 305, 308, 311, 823, 825, 314, 315, 318, 831, 833, 322, 836, 83
8, 327, 328, 329, 332, 844, 334, 846, 339, 851, 342, 343, 344, 345, 346, 34
7, 854, 862, 352, 864, 867, 868, 870, 361, 363, 875, 880, 370, 883, 372, 37
3, 374, 886, 889, 378, 379, 380, 381, 892, 894, 896, 385, 387, 901, 391, 39
2, 393, 394, 903, 908, 397, 398, 399, 910, 913, 402, 916, 405, 406, 919, 40
9, 924, 416, 417, 929, 421, 933, 425, 940, 429, 430, 943, 433, 435, 437, 44
2, 443, 445, 447, 450, 453, 454, 455, 456, 457, 458, 464, 465, 468, 471, 47
2, 474, 478, 480, 483, 487, 491, 493, 497, 498, 499, 503, 506}
{2, 3, 4, 7, 11, 12, 13, 15, 16, 21, 24, 26, 29, 33, 35, 39, 40, 43, 46, 49,
56, 58, 61, 63, 64, 66, 69, 70, 74, 83, 84, 85, 86, 87, 88, 90, 91, 99, 100,
102, 103, 104, 107, 110, 112, 113, 116, 119, 121, 125, 126, 127, 128, 129, 1
30, 133, 134, 137, 141, 144, 145, 146, 149, 151, 155, 163, 164, 166, 168, 16
9, 170, 173, 177, 178, 179, 181, 186, 187, 188, 190, 191, 193, 195, 197, 19
8, 204, 205, 206, 210, 211, 215, 217, 220, 222, 223, 224, 229, 231, 234, 23
8, 239, 240, 241, 243, 245, 247, 249, 251, 252, 253, 255, 257, 258, 260, 26
1, 263, 264, 265, 271, 274, 275, 276, 281, 282, 284, 285, 288, 292, 293, 29
4, 297, 299, 300, 301, 303, 304, 305, 309, 311, 313, 317, 320, 322, 323, 32
4, 327, 328, 329, 332, 333, 334, 335, 345, 346, 347, 351, 353, 355, 356, 36
2, 375, 378, 379, 380, 384, 388, 389, 390, 391, 392, 395, 396, 400, 404, 40
8, 409, 410, 413, 414, 416, 418, 419, 423, 424, 425, 427, 428, 429, 430, 43
1, 432, 433, 435, 438, 439, 440, 441, 444, 445, 446, 447, 450, 451, 454, 45
5, 456, 459, 462, 464, 465, 466, 476, 478, 479, 484, 486, 487, 488, 489, 49
3, 494, 497, 499, 500, 502, 504, 505, 506, 507, 509, 510, 511, 515, 517, 51
8, 520, 521, 525, 526, 529, 531, 532, 533, 534, 535, 537, 540, 544, 546, 54
8, 550, 551, 552, 557, 559, 564, 569, 572, 574, 578, 580, 582, 587, 589, 59
1, 596, 597, 598, 602, 605, 608, 611, 612, 615, 616, 619, 620, 621, 624, 62
5, 627, 628, 629, 630, 633, 634, 635, 637, 639, 644, 646, 647, 652, 653, 65
4, 655, 656, 657, 661, 663, 666, 668, 669, 673, 674, 676, 677, 678, 682, 68
3, 687, 689, 692, 693, 694, 695, 697, 698, 699, 701, 702, 703, 704, 705, 70
8, 710, 713, 714, 716, 717, 718, 719, 721, 722, 724, 725, 729, 730, 732, 73
5, 740, 743, 748, 749, 750, 751, 752, 753, 755, 756, 758, 759, 760, 767, 76
8, 770, 772, 774, 775, 779, 780, 782, 783, 784, 787, 788, 791, 796, 797, 80
0, 801, 802, 803, 807, 808, 809, 810, 811, 812, 813, 816, 817, 818, 819, 82
7, 831, 833, 834, 838, 840, 841, 843, 844, 850, 853, 856, 857, 860, 863, 86
6, 867, 871, 872, 873, 875, 876, 877, 879, 880, 881, 884, 885, 889, 892, 89
4, 896, 898, 902, 904, 905, 906, 908, 909, 910, 915, 919, 920, 924, 926, 92
7, 930, 931, 935, 936, 937, 938, 940, 941, 942, 948, 1012, 1094}
```

Implement Weighted Random Walks

Random Walk-Based Movie Recommendation System (Weighted Pixie)

Objective

In this task, you will implement a **random-walk-based recommendation algorithm** using the **Weighted Pixie** method. This technique uses a **user-movie bipartite graph** to recommend movies by simulating a random walk from a given user or movie.

Step 1: Import Required Libraries

Make sure you have the necessary libraries:

```python
import random  # For random walks
import pandas as pd  # For handling data
```

Step 2: Implement the Random Walk Algorithm

Your task is to **simulate a random walk** from a given starting point in the **bipartite user-movie graph**.

Hints for Implementation

- Start from **either a user or a movie**.
- At each step, **randomly move** to a connected node.
- Keep track of **how many times each movie is visited**.
- After completing the walk, **rank movies by visit count**.

Step 3: Implement User-Based Recommendation

Hints:

- Check if the `user_id` exists in the `graph`.
- Start a loop that runs for `walk_length` steps.
- Randomly pick a **connected node** (user or movie).
- Track how many times each **movie** is visited.
- Sort movies by visit frequency and return the **top N**.

Step 4: Implement Movie-Based Recommendation

Hints:

- Find the `movie_id` corresponding to the given `movie_name`.
- Ensure the movie exists in the `graph`.
- Start a random walk from that movie.
- Follow the same **tracking and ranking** process as the user-based version.

Note:
Your task: Implement a function `weighted_pixie_recommend(user_id,`

`walk_length=15, num=5)` or `weighted_pixie_recommend(movie_name, walk_length=15, num=5)`.

Implement either Step 3 or Step 4.

Step 5: Running Your Recommendation System

Once your function is implemented, test it by calling:

Example: User-Based Recommendation

```
weighted_pixie_recommend(1, walk_length=15, num=5)
```

Ranking	Movie Name
1	My Own Private Idaho (1991)
2	Aladdin (1992)
3	12 Angry Men (1957)
4	Happy Gilmore (1996)
5	Copycat (1995)

Example: Movie-Based Recommendation

```
weighted_pixie_recommend("Jurassic Park (1993)", walk_length=10,
num=5)
```

Ranking	Movie Name
1	Rear Window (1954)
2	Great Dictator, The (1940)
3	Field of Dreams (1989)
4	Casablanca (1942)
5	Nightmare Before Christmas, The (1993)

Step 6: Understanding the Results

Your function should return a **DataFrame** structured as follows:

Ranking	Movie Name
1	Movie A
2	Movie B
3	Movie C
4	Movie D
5	Movie E

Each movie is ranked based on **how frequently it was visited** during the walk.

Experiment with Different Parameters

- Try different `walk_length` values and observe how it changes recommendations.
- Adjust the number of recommended movies (`num`).

In [73]:
```
# Code the function here
import random  # For random walks
import pandas as pd  # For handling data


def weighted_pixie_recommend_by_user(userID, walkLen=10, num=5):

  if (userID not in graph):
    print(f'User: {userID} does not exist')
    return []


  current = userID
  counts = {}

  isUserID = True #Each time we traverse, we will be alternating between a u

  for _ in range(walkLen):
    neighbors = list(graph[current])

    if not neighbors:
      break

    next = random.choice(neighbors)

    if isUserID: # if we are a user then the next one is a movie
      counts[next] = counts.get(next, 0) + 1

    isUserID = not isUserID
    current = next;

  recommended = sorted(counts.items(), key=lambda x: x[1], reverse=True)[0:r
  recommendedIDS = [n for n, _ in recommended]
  recommendedNames = clean_movies.loc[recommendedIDS]['title'].tolist()
  res = pd.DataFrame({
      'Ranking': range(1, num+1),
      'Title':recommendedNames

  })
  res.set_index('Ranking', inplace=True)

  return res

print(weighted_pixie_recommend_by_user(1, 20, 5))
```

```
                                  Title
Ranking
1                     All About Eve (1950)
2         Replacement Killers, The (1998)
3                      Carried Away (1996)
4              Crossing Guard, The (1995)
5                         Quiz Show (1994)
```

In [79]:
```
# Code the function here
import random  # For random walks
import pandas as pd  # For handling data


# This is essentially the same function as by user, just has to convert the
def weighted_pixie_recommend_by_movie(movieName, walkLen=10, num=5):

  movieID = clean_movies.loc[clean_movies['title'] == movieName]['movie_id']

  if (len(movieID) <= 0):
    print(f'Movie: {movieName} does not exist')
    return []

  movieID = movieID[0]



  current = movieID
  counts = {}

  isUserID = False #Each time we traverse, we will be alternating between a

  for _ in range(walkLen):
    neighbors = list(graph[current])

    if not neighbors:
      break

    next = random.choice(neighbors)

    if isUserID: # if we are a user then the next one is a movie
      counts[next] = counts.get(next, 0) + 1

    isUserID = not isUserID
    current = next;

  recommended = sorted(counts.items(), key=lambda x: x[1], reverse=True)[0:r
  recommendedIDS = [n for n, _ in recommended]
  recommendedNames = clean_movies.loc[recommendedIDS]['title'].tolist()
  res = pd.DataFrame({
      'Ranking': range(1, num+1),
      'Title':recommendedNames

  })
  res.set_index('Ranking', inplace=True)

  return res
```

```
print(weighted_pixie_recommend_by_movie("Jurassic Park (1993)", 15, 5))
```

```
                                          Title
Ranking
1                           Blade Runner (1982)
2         Nightmare on Elm Street, A (1984)
3                       Princess Caraboo (1994)
4                   My Own Private Idaho (1991)
5         Die Hard: With a Vengeance (1995)
```

Submission Requirements:

To successfully complete this assignment, ensure that you submit the following:

1. Jupyter Notebook Submission

- Submit a **fully completed Jupyter Notebook** that includes:
  - **All implemented recommendation functions** (user-based, item-based, and random walk-based recommendations).
  - **Code explanations** in markdown cells to describe each step.
  - **Results and insights** from running your recommendation models.

2. Explanation of Pixie-Inspired Algorithms (3-5 Paragraphs)

- Write a **detailed explanation** of **Pixie-inspired random walk algorithms** used for recommendations.
- Your explanation should cover:
  - What **Pixie-inspired recommendation systems** are.
  - How **random walks** help in identifying relevant recommendations.
  - Any real-world applications of such algorithms in industry.

3. Report for the Submitted Notebook

Your report should be structured as follows:

Title: Movie Recommendation System Report

1. Introduction

- Briefly introduce **movie recommendation systems** and why they are important.
- Explain the **different approaches used** (user-based, item-based, random-walk).

2. Dataset Description

- Describe the **MovieLens 100K dataset**:
  - Number of users, movies, and ratings.
  - What features were used.
  - Any preprocessing performed.

3. Methodology

- Explain the three recommendation techniques implemented:
  - **User-based collaborative filtering** (how user similarity was calculated).
  - **Item-based collaborative filtering** (how item similarity was determined).
  - **Random-walk-based Pixie algorithm** (why graph-based approaches are effective).

4. Implementation Details

- Discuss the steps taken to build the functions.
- Describe how the **adjacency list graph** was created.
- Explain how **random walks** were performed and how visited movies were ranked.

5. Results and Evaluation

- Present **example outputs** from each recommendation approach.
- Compare the different methods in terms of accuracy and usefulness.
- Discuss any **limitations** in the implementation.

6. Conclusion

- Summarize the key takeaways from the project.
- Discuss potential improvements (e.g., **hybrid models, additional features**).
- Suggest real-world applications of the methods used.

Submission Instructions

- Submit `.zip` file consisting of Jupyter Notebook and all the datafiles (provided) and the ones saved [i.e. `users.csv`, `movies.csv` and `ratings.csv`]. Also, include the Report and Pixie Algorithm explanation document.
- [`Bonus 10 Points`] **Upload your Jupyter Notebook, Explanation Document, and Report** to your GitHub repository.

- Ensure the repository is public and contains:
  - `users.csv`, `movies.csv` and `ratings.csv` [These are the Dataframes which were created in part 1. Save and export them as a `.csv` file]
  - `Movie_Recommendation.ipynb`
  - `Pixie_Algorithm_Explanation.pdf` or `.md`
  - `Recommendation_Report.pdf` or `.md`
- **Submit the GitHub repository link in the cell below.**

Example Submission Format

```
GitHub Repository: https://github.com/username/Movie-Recommendation
```

In [ ]:
```
# Submit the Github Link here:
```

Grading Rubric: ITCS 6162 - Data Mining Assignment

Category	Criteria	Points
Part 1: Exploring and Cleaning Data (15 pts)	Properly loads `u.user`, `u.movies`, and `u.item` datasets into DataFrames	5
	Handles missing values, duplicates, and inconsistencies appropriately	5
	Saves the cleaned datasets into CSV files: `users.csv`, `movies.csv`, `ratings.csv`	5
Part 2: Collaborative Filtering-Based Recommendation (30 pts)	Implements user-based collaborative filtering correctly	10
	Implements item-based collaborative filtering correctly	10
	Computes similarity measures accurately and provides valid recommendations	10
Part 3: Graph-Based Recommender (Pixie-Inspired Algorithm) (35 pts)	Constructs adjacency lists properly from user-movie interactions	10
	Implements weighted random walk-based recommendation correctly	15
	Explains and justifies the algorithm design choices (Pixie-inspired)	10
Code Quality & Documentation (10 pts)	Code is well-structured, efficient, and follows best practices	5
	Markdown explanations and comments are clear and enhance understanding	5

Category	Criteria	Points
Results & Interpretation (5 pts)	Provides meaningful insights from the recommendation system's output	5
Submission & Report (5 pts)	Submits all required files in the correct format (ZIP file with Jupyter notebook, processed CSV files, and project report)	5
Total		100

Bonus (10 pts)

Category	Criteria	Points
GitHub Submission	Provides a well-documented GitHub repository with CSV files, a structured README, and a properly formatted Jupyter Notebook	10