



Assessment Report

on

“Movie Watch Pattern Clustering”

submitted as partial fulfillment for the award of

BACHELOR OF TECHNOLOGY DEGREE

SESSION 2024-25

In

CSE(AI)

By

Name : Nikhil Kumar Singh

Roll Number : 202401100300161

Section: C

1. Introduction

This project analyzes user behavior in movie-watching habits by clustering viewers based on three key features:

1. **Time of Watching** – The hour of the day when users watch movies (0-23).
2. **Genre Preference** – The most frequently watched genre (action, comedy, drama, thriller).
3. **Rating Behavior** – The average rating given by users (1-5 scale)

2. Problem Statement

Movie Watch Pattern Clustering

Cluster users based on time of watching, genre preference, and rating behavior.

4. Methodology

The following steps were followed:

1. Load and preprocess the dataset
 - Standardized numerical features (watch_time_hour, avg_rating_given)
 - Applied One-Hot Encoding to categorical feature (genre_preference)
2. Determine the optimal number of clusters

- Used Elbow Method and Silhouette Score for evaluation
- Selected $k = 4$ as optimal number of clusters

3. Train the clustering model

- Applied K-Means algorithm
- Assigned each user to a cluster

4. Analyze clusters

- Plotted distributions of clusters
- Examined average values of features within each cluster
- Visualized genre preferences by cluster

5. Save and export results

CODE:

```

# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.compose import ColumnTransformer

# 1. Load and prepare the data
def load_and_preprocess_data():
    # Load the data
    df = pd.read_csv('/content/movie_watch.csv')

    # Data preprocessing
    preprocessor = ColumnTransformer(
        transformers=[
            ('num', StandardScaler(), ['watch_time_hour', 'avg_rating_given']),
            ('cat', OneHotEncoder(), ['genre_preference'])
        ])

    processed_data = preprocessor.fit_transform(df)
    return df, processed_data, preprocessor

# 2. Determine optimal number of clusters
def find_optimal_clusters(data):
    wcss = []
    silhouettes = []
    max_clusters = 8

    for k in range(2, max_clusters+1):
        kmeans = KMeans(n_clusters=k, random_state=42)

```

```

# 2. Determine optimal number of clusters
def find_optimal_clusters(data):
    wcss = []
    silhouettes = []
    max_clusters = 8

    for k in range(2, max_clusters+1):
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(data)
        wcss.append(kmeans.inertia_)
        silhouettes.append(silhouette_score(data, kmeans.labels_))

    # Plot the results
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(range(2, max_clusters+1), wcss, 'bo-')
    plt.title('Elbow Method')
    plt.xlabel('Number of clusters')
    plt.ylabel('WCSS')

    plt.subplot(1, 2, 2)
    plt.plot(range(2, max_clusters+1), silhouettes, 'go-')
    plt.title('Silhouette Scores')
    plt.xlabel('Number of clusters')
    plt.ylabel('Silhouette Score')
    plt.tight_layout()
    plt.show()

    return wcss, silhouettes

# 3. Perform clustering and analyze results
def perform_clustering(df, data, n_clusters=4):
    # Perform K-means clustering
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)

```

```

# 3. Perform clustering and analyze results
def perform_clustering(df, data, n_clusters=4):
    # Perform K-means clustering
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    clusters = kmeans.fit_predict(data)
    df['cluster'] = clusters

    # Cluster distribution
    plt.figure(figsize=(8, 5))
    sns.countplot(x='cluster', data=df, palette='viridis')
    plt.title('Distribution of Users Across Clusters')
    plt.show()

    # Analyze cluster characteristics
    cluster_profile = df.groupby('cluster').agg({
        'watch_time_hour': ['mean', 'std'],
        'genre_preference': lambda x: x.mode()[0],
        'avg_rating_given': ['mean', 'std']
    })

    print("Cluster Profiles:")
    print(cluster_profile)

    # Visualize watch times by cluster
    plt.figure(figsize=(10, 6))
    sns.boxplot(x='cluster', y='watch_time_hour', data=df, palette='viridis')
    plt.title('Watch Time Distribution by Cluster')
    plt.show()

    # Visualize ratings by cluster
    plt.figure(figsize=(10, 6))
    sns.boxplot(x='cluster', y='avg_rating_given', data=df, palette='viridis')
    plt.title('Rating Distribution by Cluster')
    plt.show()

```

```

# Main execution
if __name__ == "__main__":
    # Step 1: Load and preprocess data
    df, processed_data, preprocessor = load_and_preprocess_data()

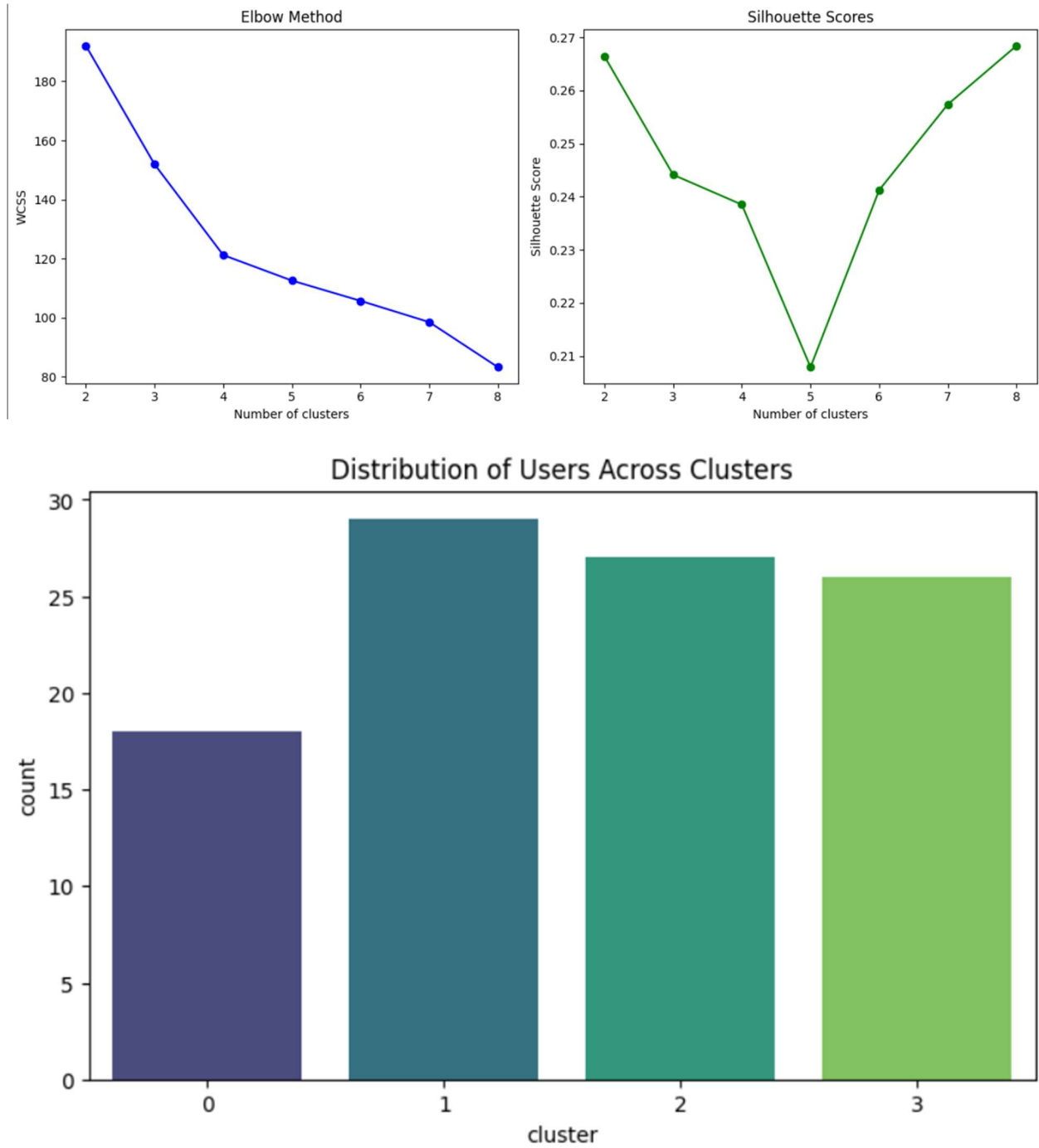
    # Step 2: Determine optimal number of clusters
    print("Determining optimal number of clusters...")
    wcss, silhouettes = find_optimal_clusters(processed_data)

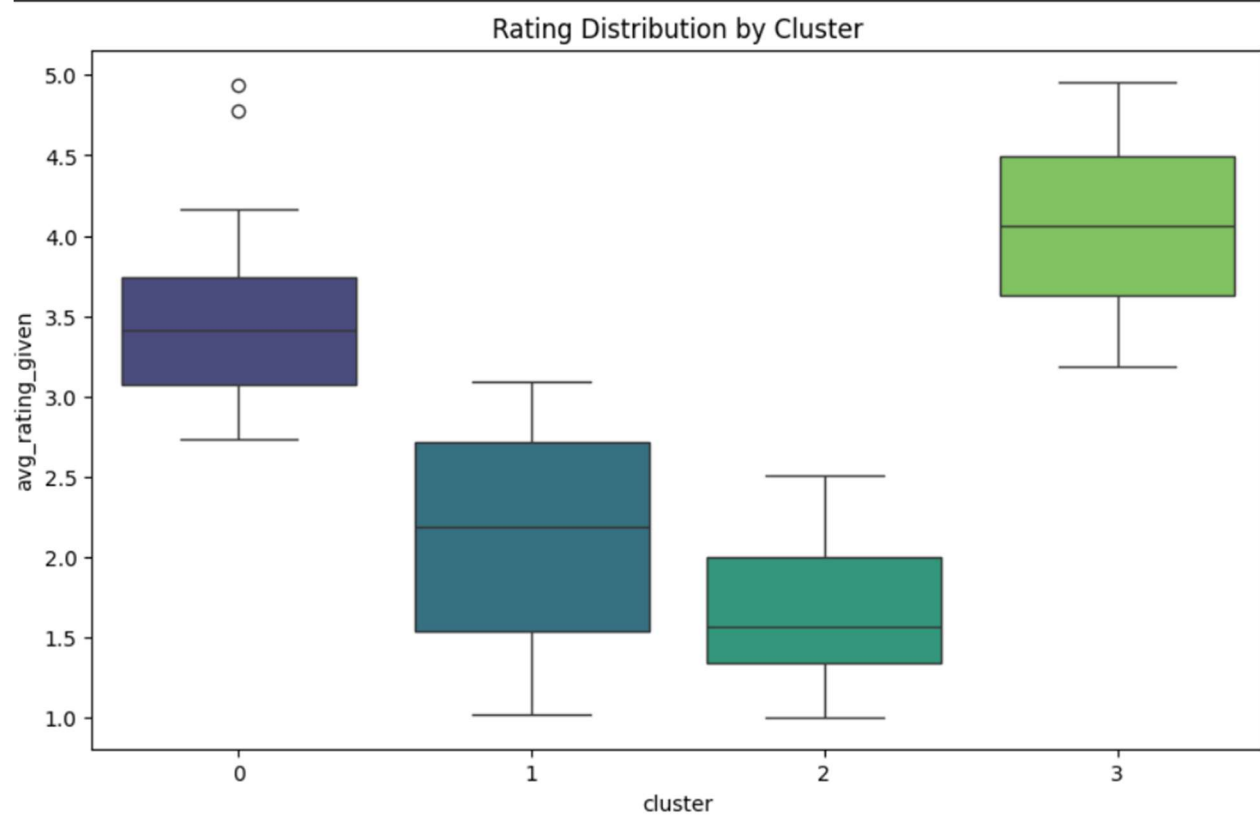
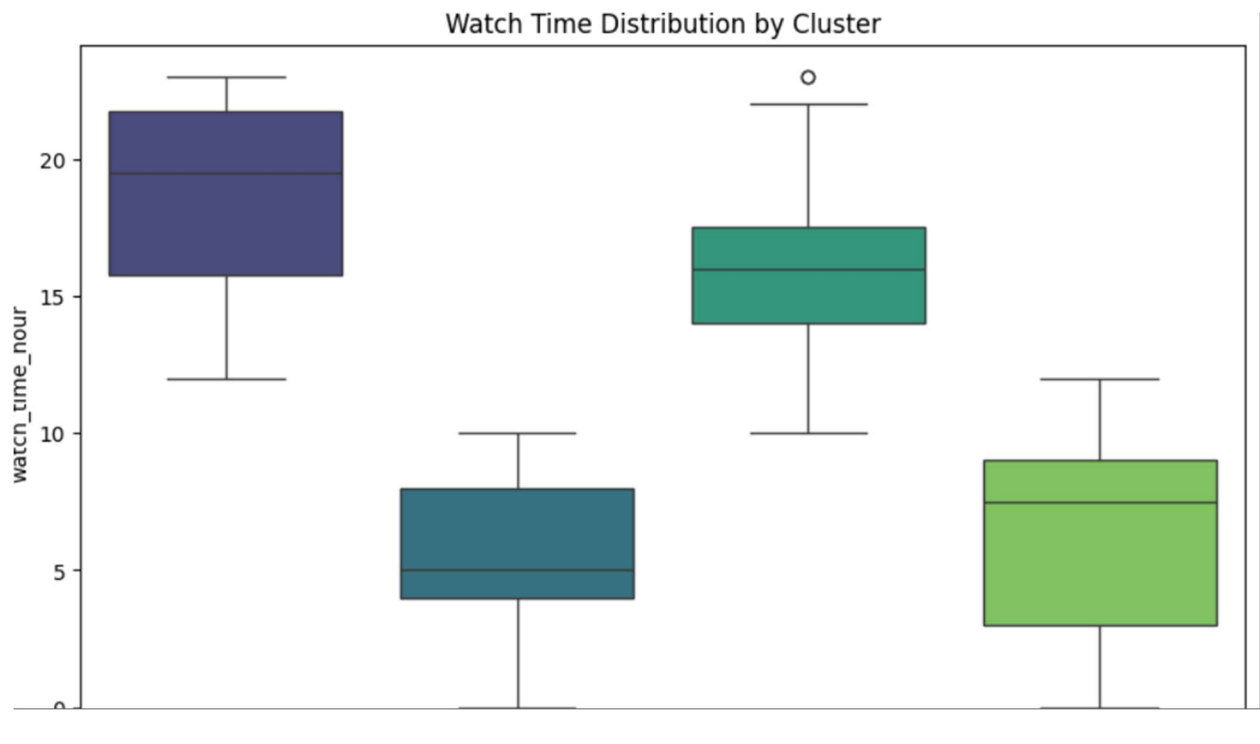
    # Step 3: Perform clustering with optimal k (4 in this case)
    print("\nPerforming clustering with k=4...")
    clustered_df = perform_clustering(df, processed_data, n_clusters=4)

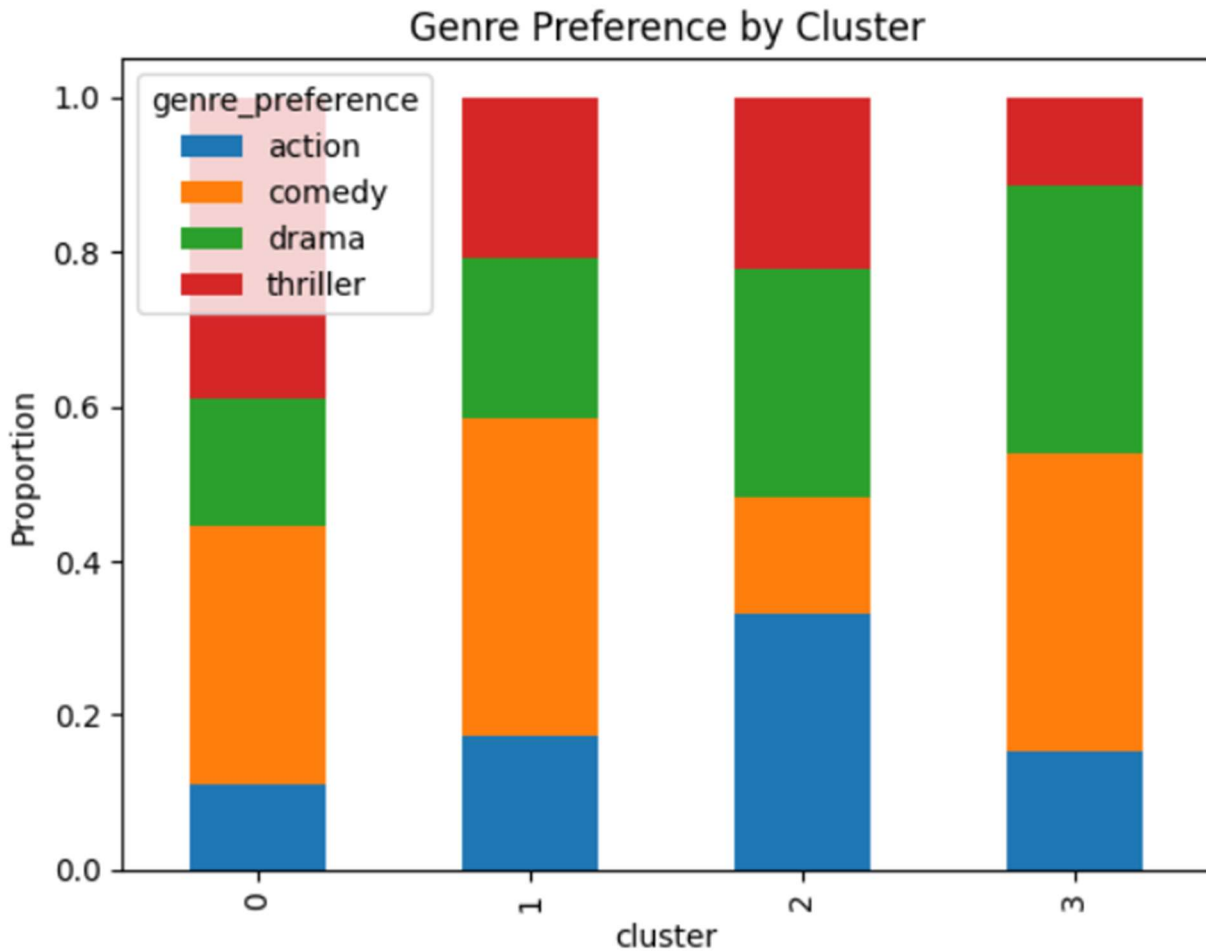
    # Save results if needed
    clustered_df.to_csv('clustered_movie_watch.csv', index=False)
    print("\nAnalysis complete. Results saved to 'clustered_movie_watch.csv'")

```

OUTPUT:







REFERENCES:

1. Dataset provided for the exam
2. Scikit-learn documentation: <https://scikit-learn.org>
3. Matplotlib and Seaborn for visualization

