Aim: To implement recommendation system on your dataset using the following machine learning techniques.

Types of Recommendation Systems

A Recommendation System suggests relevant items to users based on their preferences, behavior, or other factors. There are several types of recommendation techniques:

🔹 1. Content-Based Filtering

- **Idea:** Recommends items similar to those the user has liked before.
- **Works on:** Item features (attributes such as brand, price, category).

Example:

- If a user buys a Samsung phone, they might be recommended another Samsung device based on brand preference.

- Uses techniques like TF-IDF (for text data), Cosine Similarity, Decision Trees, etc.

🔹 2. Collaborative Filtering (CF)

- **Idea:** Recommends items based on similar users' preferences.
- **Works on:** User interactions rather than item features.

Example:

- If User A and User B have similar purchase histories, items bought by User A but not yet by User B will be recommended to User B.

- Uses methods like User-Based CF and Item-Based CF.

🔹 3. Hybrid Recommendation System

- **Idea:** Combines Content-Based Filtering and Collaborative Filtering for better accuracy.

Example:

- Netflix uses a hybrid approach, considering both user preferences and what similar users watch.

🔹 4. Knowledge-Based Recommendation

- **Idea:** Recommends items based on explicit domain knowledge rather than past user behavior.

Example:

- A car recommendation system suggests vehicles based on engine type, price, and fuel efficiency, regardless of past purchases.

2. Recommendation System Evaluation Measures

Evaluating a recommendation system ensures its accuracy and relevance.
Below are common evaluation metrics:

🔹 1. Accuracy-Based Metrics

(a) Precision:

- Measures how many of the recommended items are actually relevant.

- **Formula:**

$$Precision = \frac{Relevant\ Recommendations}{Total\ Recommendations}$$

- **Example:**
    - If 5 out of 10 recommended items are relevant, Precision = 5/10 = 0.5 (50%).

(b) Recall:

- Measures how many of the relevant items are actually recommended.
    - **Formula:**

$$Recall = \frac{Relevant\ Recommendations}{Total\ Relevant\ Items\ Available}$$

- **Example:**
    - If a user liked 8 items, but only 5 were recommended, Recall = 5/8 = 0.625 (62.5%).

(c) F1-Score:

- A balance between Precision and Recall.

  - **Formula:**

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

- Used when both Precision and Recall are important.

(d) Accuracy:

  - In classification-based recommendation systems (like Decision Trees), accuracy is measured as:
  - In our Decision Tree model, if Accuracy = 1.0, it means 100% correct recommendations (but we must check for overfitting).

$$Accuracy = \frac{Correct\ Predictions}{Total\ Predictions}$$

🔹 2. Ranking-Based Metrics

These measure how well the recommendation system ranks items:

(a) Mean Average Precision (MAP):

● Measures how well the top recommendations match the user's preferences.

(b) Normalized Discounted Cumulative Gain (NDCG):

● Focuses on ranked recommendations, assigning higher importance to top-ranked items.

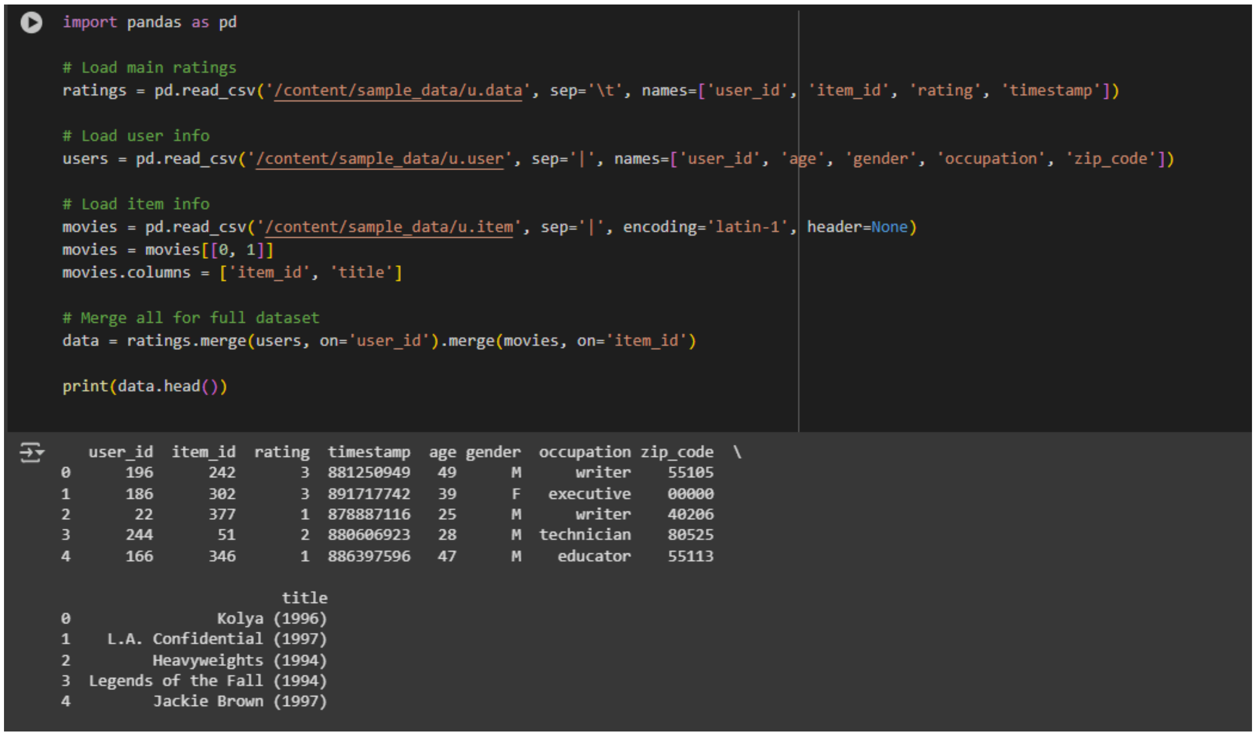
🔹 3. Diversity and Novelty Metrics

● **Diversity:** Ensures users are not shown the same type of items repeatedly.

Novelty: Measures if recommendations introduce new and unknown items

Implementation:

Load dataset



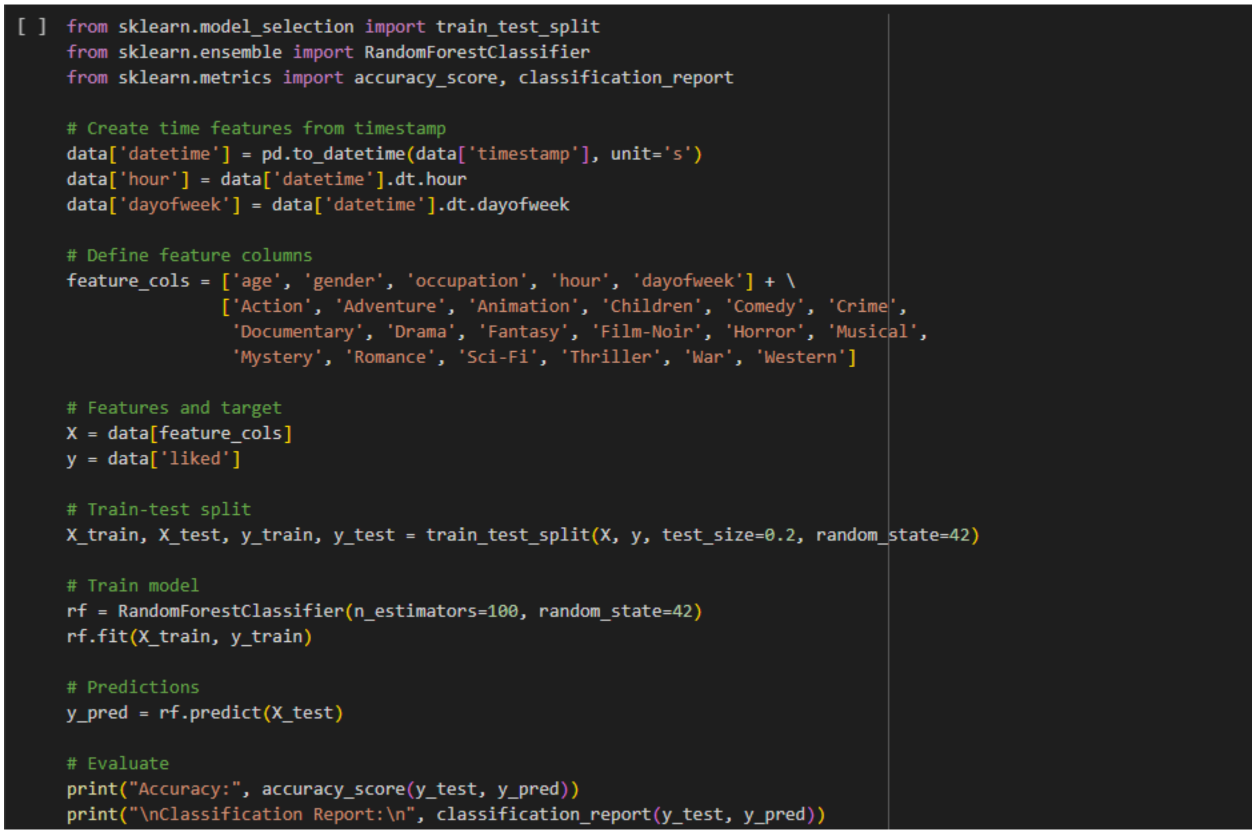
```
import pandas as pd

# Load main ratings
ratings = pd.read_csv('/content/sample_data/u.data', sep='\t', names=['user_id', 'item_id', 'rating', 'timestamp'])

# Load user info
users = pd.read_csv('/content/sample_data/u.user', sep='|', names=['user_id', 'age', 'gender', 'occupation', 'zip_code'])

# Load item info
movies = pd.read_csv('/content/sample_data/u.item', sep='|', encoding='latin-1', header=None)
movies = movies[[0, 1]]
movies.columns = ['item_id', 'title']

# Merge all for full dataset
data = ratings.merge(users, on='user_id').merge(movies, on='item_id')

print(data.head())
```

```
   user_id  item_id  rating  timestamp  age gender  occupation zip_code  \
0      196      242       3  881250949   49      M      writer    55105
1      186      302       3  891717742   39      F   executive    00000
2       22      377       1  878887116   25      M      writer    40206
3      244       51       2  880606923   28      M  technician    80525
4      166      346       1  886397596   47      M    educator    55113

                     title
0             Kolya (1996)
1    L.A. Confidential (1997)
2        Heavyweights (1994)
3  Legends of the Fall (1994)
4        Jackie Brown (1997)
```

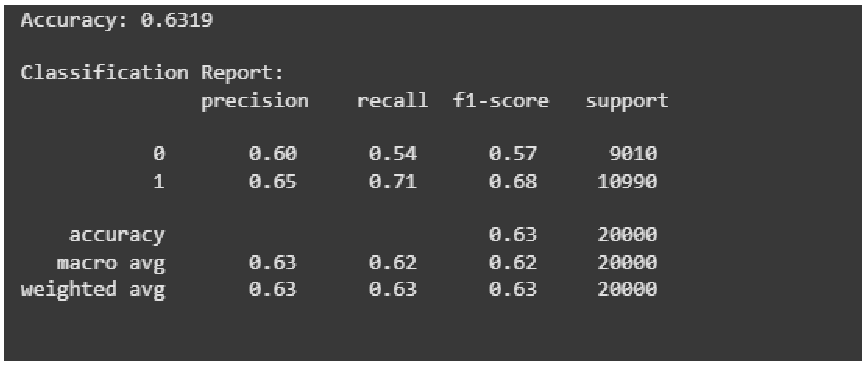
Dataset Used:
MovieLens 100K dataset, which includes:

- u.user: user demographic info.
- u.item: movie metadata.
- u.data: user ratings (user ID, movie ID, rating, timestamp)

Use classification:

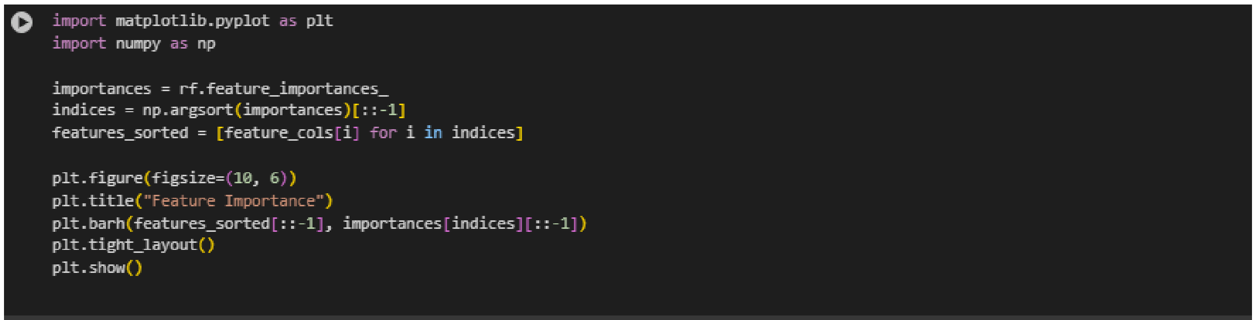


```
[ ] from sklearn.model_selection import train_test_split
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import accuracy_score, classification_report

    # Create time features from timestamp
    data['datetime'] = pd.to_datetime(data['timestamp'], unit='s')
    data['hour'] = data['datetime'].dt.hour
    data['dayofweek'] = data['datetime'].dt.dayofweek

    # Define feature columns
    feature_cols = ['age', 'gender', 'occupation', 'hour', 'dayofweek'] + \
                   ['Action', 'Adventure', 'Animation', 'Children', 'Comedy', 'Crime',
                    'Documentary', 'Drama', 'Fantasy', 'Film-Noir', 'Horror', 'Musical',
                    'Mystery', 'Romance', 'Sci-Fi', 'Thriller', 'War', 'Western']

    # Features and target
    X = data[feature_cols]
    y = data['liked']

    # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Train model
    rf = RandomForestClassifier(n_estimators=100, random_state=42)
    rf.fit(X_train, y_train)

    # Predictions
    y_pred = rf.predict(X_test)

    # Evaluate
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

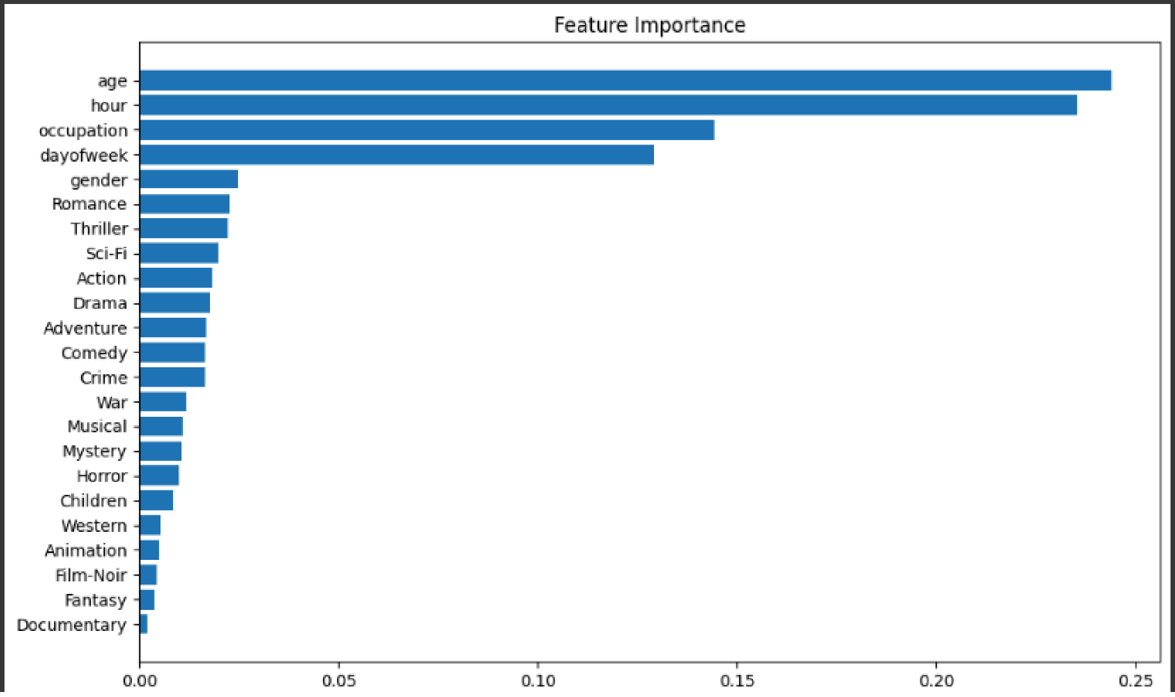


```
Accuracy: 0.6319

Classification Report:
              precision    recall  f1-score   support

           0       0.60      0.54      0.57      9010
           1       0.65      0.71      0.68     10990

    accuracy                           0.63     20000
   macro avg       0.63      0.62      0.62     20000
weighted avg       0.63      0.63      0.63     20000
```

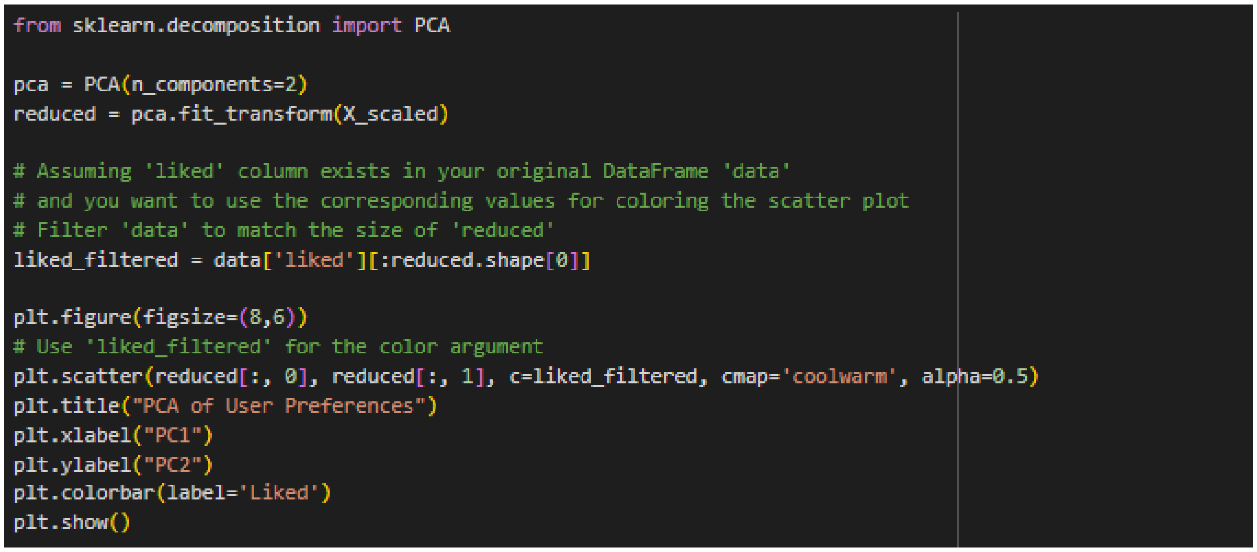
Visualization to show which attribute affects the choice most:



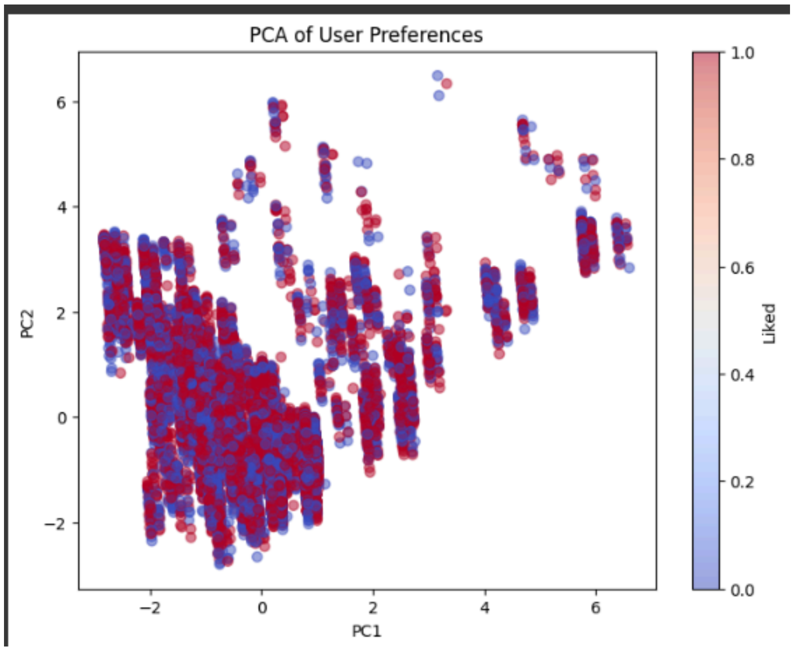
```
import matplotlib.pyplot as plt
import numpy as np

importances = rf.feature_importances_
indices = np.argsort(importances)[::-1]
features_sorted = [feature_cols[i] for i in indices]

plt.figure(figsize=(10, 6))
plt.title("Feature Importance")
plt.barh(features_sorted[::-1], importances[indices][::-1])
plt.tight_layout()
plt.show()
```



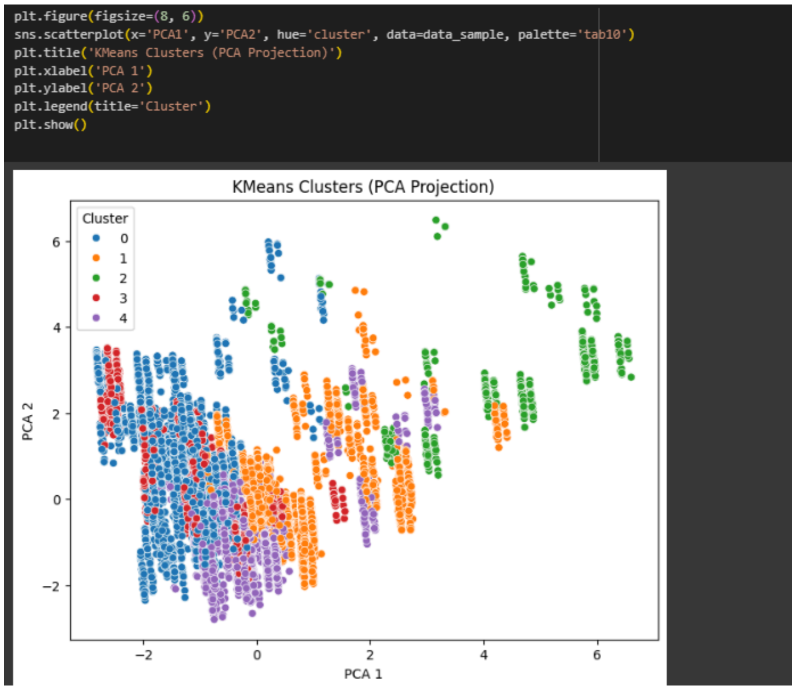
Dimensionality reduction:



```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
reduced = pca.fit_transform(X_scaled)

# Assuming 'liked' column exists in your original DataFrame 'data'
# and you want to use the corresponding values for coloring the scatter plot
# Filter 'data' to match the size of 'reduced'
liked_filtered = data['liked'][:reduced.shape[0]]

plt.figure(figsize=(8,6))
# Use 'liked_filtered' for the color argument
plt.scatter(reduced[:, 0], reduced[:, 1], c=liked_filtered, cmap='coolwarm', alpha=0.5)
plt.title("PCA of User Preferences")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.colorbar(label='Liked')
plt.show()
```

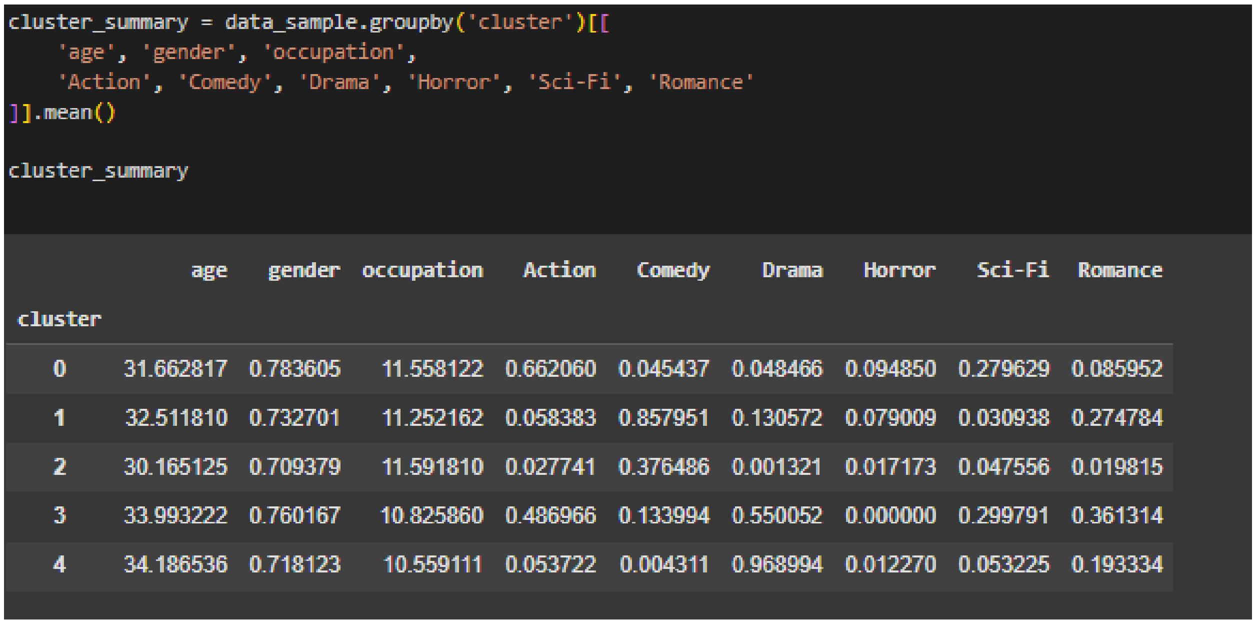


Kmeans clustering:



```
plt.figure(figsize=(8, 6))
sns.scatterplot(x='PCA1', y='PCA2', hue='cluster', data=data_sample, palette='tab10')
plt.title('KMeans Clusters (PCA Projection)')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.legend(title='Cluster')
plt.show()
```

This output shows the **cluster-wise summary statistics** of user preferences from a movie dataset, grouped by `cluster` (which likely came from a clustering algorithm like KMeans).



```python
cluster_summary = data_sample.groupby('cluster')[[
    'age', 'gender', 'occupation',
    'Action', 'Comedy', 'Drama', 'Horror', 'Sci-Fi', 'Romance'
]].mean()

cluster_summary
```

cluster	age	gender	occupation	Action	Comedy	Drama	Horror	Sci-Fi	Romance
0	31.662817	0.783605	11.558122	0.662060	0.045437	0.048466	0.094850	0.279629	0.085952
1	32.511810	0.732701	11.252162	0.058383	0.857951	0.130572	0.079009	0.030938	0.274784
2	30.165125	0.709379	11.591810	0.027741	0.376486	0.001321	0.017173	0.047556	0.019815
3	33.993222	0.760167	10.825860	0.486966	0.133994	0.550052	0.000000	0.299791	0.361314
4	34.186536	0.718123	10.559111	0.053722	0.004311	0.968994	0.012270	0.053225	0.193334

Conclusion:

In this experiment, we used K-Means Clustering to group users based on their demographics and movie genre preferences. The clustering revealed distinct patterns—some groups preferred action-packed genres like Action and Sci-Fi, while others leaned towards Drama and Romance.

This segmentation helps us understand user behavior and can be applied in areas like personalized recommendations and targeted marketing. Overall, the experiment highlights how unsupervised learning can uncover hidden insights from data and aid in effective decision-making.