```
import pandas as pd
from google.colab import files
uploaded = files.upload()

df = pd.read_csv("movies.csv")
df.head()
```

Choose Files movies.csv

- **movies.csv**(text/csv) - 608 bytes, last modified: 5/8/2025 - 100% done

```
Saving movies.csv to movies (1).csv
```

	Customer	Age	Watched movie	Related movie	Start time	End time	websites	paid
0	Logeshkannan	19	kaththi	thupakki	03:00	06:00	JioHotstar	150
1	Dhamothiran	19	good bad ugly	vidamuyarchi	09:00	12:00	Amazon prime	100
2	Sakthi	19	kingston	gangers	12:00	03:00	JioHotstar	150
3	Sanjeev	18	beast	gurkha	03:00	06:00	Amazon prime	100
4	Naveen	18	Wrong turn	Thanksgiving	09:00	12:00	JioHotstar	150

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
import os
import zipfile
import urllib.request

# Download and extract MovieLens 100k if not already present
data_dir = "./ml-100k"
if not os.path.exists(data_dir):
    url = "http://files.grouplens.org/datasets/movielens/ml-100k.zip"
    urllib.request.urlretrieve(url, "ml-100k.zip")
    with zipfile.ZipFile("ml-100k.zip", "r") as zip_ref:
        zip_ref.extractall()


ratings_path = "ml-100k/u.data"
movies_path = "ml-100k/u.item"


!pip install scikit-surprise
```

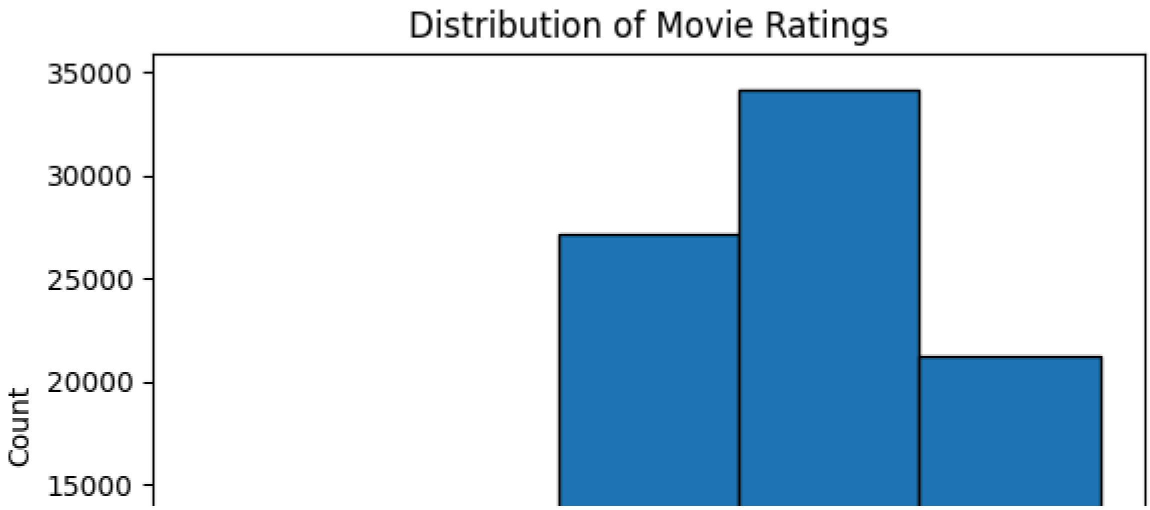
```
Collecting scikit-surprise
  Downloading scikit_surprise-1.1.4.tar.gz (154 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 154.4/154.4 kB 5.9 MB/s eta
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
```

```
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-pa
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (pyproject.toml) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.4-cp311-cp311-
  Stored in directory: /root/.cache/pip/wheels/2a/8f/6e/7e2899163e2d85d8266daab4
Successfully built scikit-surprise
Installing collected packages: scikit-surprise
Successfully installed scikit-surprise-1.1.4
```

```
"""
AI-Driven Matchmaking System for Movie Recommendation
"""

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from scipy.sparse import csr_matrix
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Load MovieLens 100k dataset
# Correcting the path to match where the data was extracted
ratings_path = "./ml-100k/u.data"
movies_path = "./ml-100k/u.item"

# Define column names
ratings_cols = ["user_id", "movie_id", "rating", "timestamp"]
movies_cols = [
    "movie_id", "title", "release_date", "video_release_date", "IMDb_URL",
    "unknown", "Action", "Adventure", "Animation", "Children's", "Comedy",
    "Crime", "Documentary", "Drama", "Fantasy", "Film-Noir", "Horror",
    "Musical", "Mystery", "Romance", "Sci-Fi", "Thriller", "War", "Western"
]

# Read data
ratings = pd.read_csv(ratings_path, sep="\t", names=ratings_cols)
movies = pd.read_csv(movies_path, sep="|", names=movies_cols, encoding="latin-1")

# Merge for convenience
ratings = ratings.merge(movies[["movie_id", "title"]], on="movie_id")

# Split data
train, test = train_test_split(ratings, test_size=0.2, random_state=42)

# Build user-item matrix
user_item_matrix = csr_matrix((train["rating"], (train["user_id"], train["movie_id"])))

# Compute user-user similarity
user_similarity = cosine_similarity(user_item_matrix)

# Predict function
def predict_rating(user_id: int, movie_id: int, k: int = 20):
    users_rated = train[train["movie_id"] == movie_id]["user_id"].values
    similarities = user_similarity[user_id, users_rated]
    ratings_vec = train[train["movie_id"] == movie_id]["rating"].values
```

```python
    if len(similarities) == 0:
        return train["rating"].mean()

    top_k_idx = np.argsort(similarities)[-k:]
    top_k_sims = similarities[top_k_idx]
    top_k_ratings = ratings_vec[top_k_idx]

    if top_k_sims.sum() == 0:
        return train["rating"].mean()

    return np.dot(top_k_sims, top_k_ratings) / top_k_sims.sum()

# Evaluate on test set
y_true, y_pred = [], []
for row in test.itertuples():
    y_true.append(row.rating)
    y_pred.append(predict_rating(row.user_id, row.movie_id))

rmse = np.sqrt(mean_squared_error(y_true, y_pred))
print(f"Test RMSE: {rmse:.4f}")

# Recommend top N movies for a user
def recommend_movies(user_id: int, N: int = 5):
    rated = train[train["user_id"] == user_id]["movie_id"].tolist()
    all_movies = ratings["movie_id"].unique()
    candidates = [m for m in all_movies if m not in rated]

    predictions = [predict_rating(user_id, m) for m in candidates]
    top_indices = np.argsort(predictions)[-N:][::-1]
    top_movie_ids = [candidates[i] for i in top_indices]

    return movies[movies["movie_id"].isin(top_movie_ids)][["title"]]

# Example: Recommend for user 1
print("Top recommendations for User 1:")
print(recommend_movies(1))

# Optional: Visualize rating distribution
plt.hist(ratings["rating"], bins=5, edgecolor="black")
plt.title("Distribution of Movie Ratings")
plt.xlabel("Rating")
plt.ylabel("Count")
plt.show()
```

```
Test RMSE: 1.0118
Top recommendations for User 1:
                                   title
849           Perfect Candidate, A (1996)
1188                    Prefontaine (1997)
1305                Delta of Venus (1994)
1466  Saint of Fort Washington, The (1993)
1641               Some Mother's Son (1996)
```




```
# Train-Test Split for collaborative filtering
train, test = train_test_split(ratings, test_size=0.2, random_state=42)


# Define a user ID and a movie ID for which to predict a rating
user_id = 1  # Replace with a valid user ID from your dataset
movie_id = 50  # Replace with a valid movie ID from your dataset

# Predict the rating for the specified user and movie
predicted_rating = predict_rating(user_id, movie_id)

# Print the predicted rating (optional)
print(f"Predicted rating for User {user_id} and Movie {movie_id}: {predicted_rating:.2f}")
```

```
Predicted rating for User 1 and Movie 50: 4.77
```

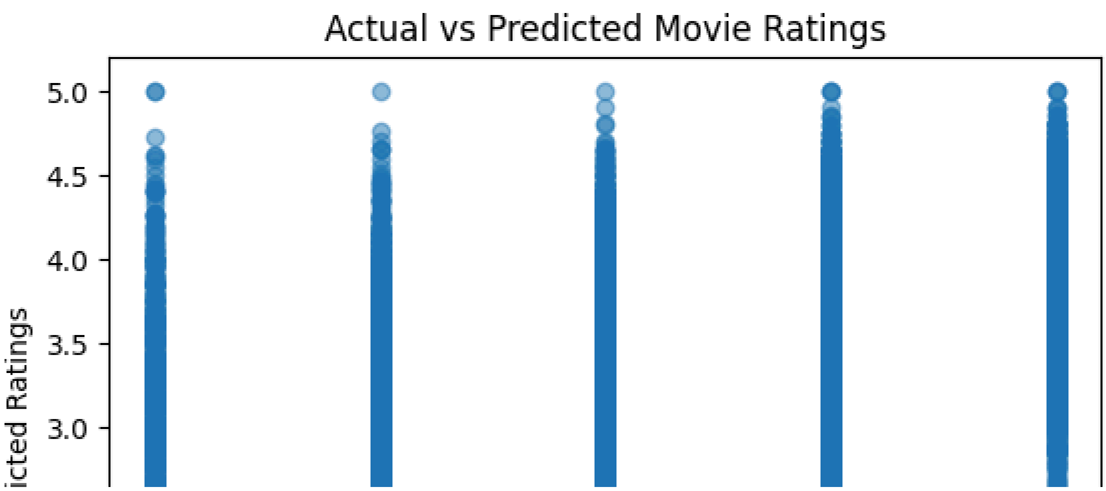
```
# --- Evaluation block already in the recommender script ---

from sklearn.metrics import mean_squared_error
y_true, y_pred = [], []

for row in test.itertuples():          # test is the 20 % split of ratings
    y_true.append(row.rating)          # actual rating
    y_pred.append(predict_rating(      # predicted rating from CF
        row.user_id, row.movie_id)
    )

rmse = np.sqrt(mean_squared_error(y_true, y_pred))
print(f"Test RMSE: {rmse:.4f}")
```

```
Test RMSE: 1.0118
```

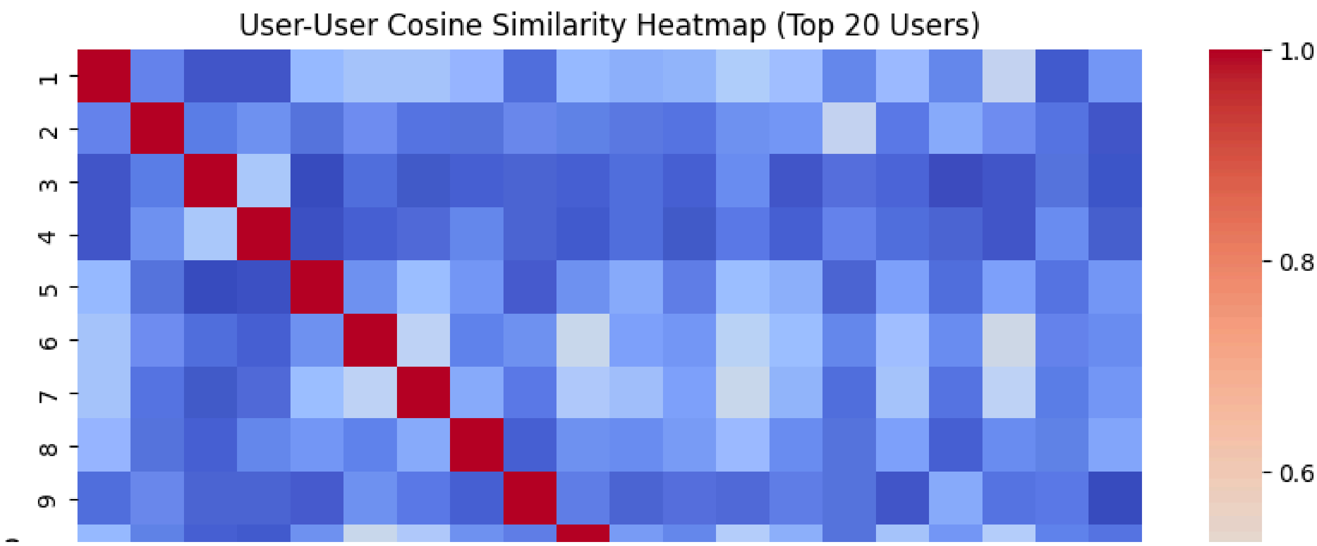
```
from sklearn.metrics import r2_score
r2 = r2_score(y_true, y_pred)
print(f"Pseudo-R²: {r2:.4f}")
```

```
Pseudo-R²: 0.1894
```

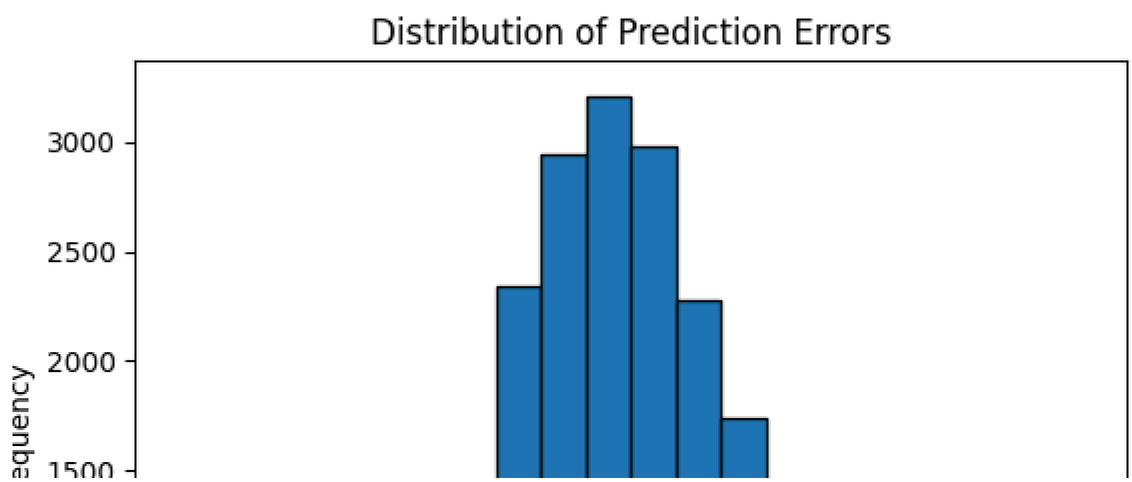
```
import matplotlib.pyplot as plt

# Actual vs Predicted Ratings scatter plot
plt.scatter(y_true, y_pred, alpha=0.5)
plt.xlabel("Actual Ratings")
plt.ylabel("Predicted Ratings")
plt.title("Actual vs Predicted Movie Ratings")
plt.show()
```




```
import seaborn as sns

# Sample: Use a subset of users for readability
subset_users = list(range(1, 21))  # First 20 users
similarity_subset = user_similarity[np.ix_(subset_users, subset_users)]

plt.figure(figsize=(10, 8))
sns.heatmap(similarity_subset, cmap="coolwarm", xticklabels=subset_users, yticklabels=subset_
plt.title("User-User Cosine Similarity Heatmap (Top 20 Users)")
plt.xlabel("User ID")
plt.ylabel("User ID")
plt.show()
```




```
errors = np.array(y_pred) - np.array(y_true)
plt.hist(errors, bins=20, edgecolor='black')
plt.title("Distribution of Prediction Errors")
plt.xlabel("Prediction Error")
plt.ylabel("Frequency")
plt.show()
```




```
user_id = 5  # Example user
recommended = recommend_movies(user_id, N=5)

print(f"Top 5 Movie Recommendations for User {user_id}:")
print(recommended)
```

```
Top 5 Movie Recommendations for User 5:
                                             title
849                       Perfect Candidate, A (1996)
1462                                Boys, Les (1997)
1466              Saint of Fort Washington, The (1993)
1499                        Santa with Muscles (1996)
1652  Entertaining Angels: The Dorothy Day Story (1996)
```

```
import pandas as pd
```

```
# Load MovieLens 100k dataset (assuming you've downloaded it as shown before)
ratings_path = "./ml-100k/u.data"  # user  item  rating  timestamp
movies_path = "./ml-100k/u.item"   # item|title|release date|...

# Define column names
ratings_cols = ["user_id", "movie_id", "rating", "timestamp"]
movies_cols = [
    "movie_id", "title", "release_date", "video_release_date", "IMDb_URL",
    "unknown", "Action", "Adventure", "Animation", "Children's", "Comedy",
    "Crime", "Documentary", "Drama", "Fantasy", "Film-Noir", "Horror",
    "Musical", "Mystery", "Romance", "Sci-Fi", "Thriller", "War", "Western"
]

# Load ratings and movies
ratings = pd.read_csv(ratings_path, sep="\t", names=ratings_cols)
movies = pd.read_csv(movies_path, sep="|", names=movies_cols, encoding="latin-1")

# Merge ratings with movie titles for easier analysis
df = ratings.merge(movies[["movie_id", "title"]], on="movie_id")

# Display the first few rows
df.head()
```

	user_id	movie_id	rating	timestamp	title
0	196	242	3	881250949	Kolya (1996)
1	186	302	3	891717742	L.A. Confidential (1997)
2	22	377	1	878887116	Heavyweights (1994)
3	244	51	2	880606923	Legends of the Fall (1994)
4	166	346	1	886397596	Jackie Brown (1997)

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
from sklearn.model_selection import train_test_split

# Feature selection: In your case, 'user_id', 'movie_id' are the features
# 'rating' is the target (what we want to predict)
X = df[['user_id', 'movie_id']]  # Features: user_id and movie_id
y = df['rating']  # Target: user ratings

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

# Display the first few rows of the training data
print(X_train.head(), y_train.head())
```

```
       user_id  movie_id
75220      807      1411
48955      474       659
44966      463       268
13568      139       286
92727      621       751 75220    1
48955    5
44966    4
13568    4
92727    4
Name: rating, dtype: int64
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Binarize the ratings (e.g., 1-3 = 'disliked', 4-5 = 'liked')
y_binary = (y >= 4).astype(int)  # 1 if liked (rating >= 4), 0 if disliked (rating < 4)

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y_binary, test_size=0.2, random_stat

# Logistic Regression Model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Evaluate the model (Accuracy)
accuracy = model.score(X_test, y_test)
print(f"Accuracy: {accuracy:.2f}")
```

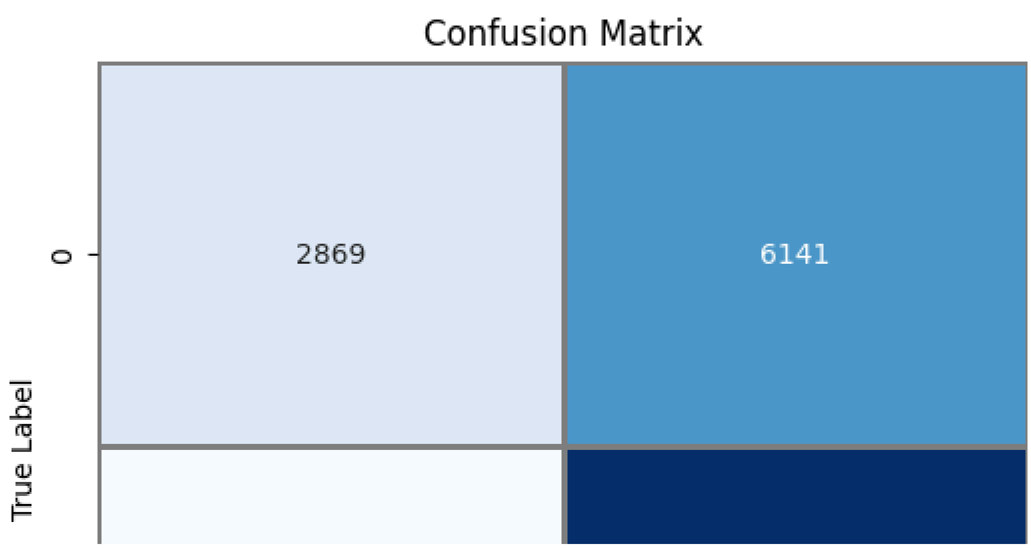
```
Accuracy: 0.60
```

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Predicting the labels for the test set
y_pred = model.predict(X_test)

# Performance Metrics
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Confusion Matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Classification Report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.60
Confusion Matrix:
[[2869 6141]
 [1940 9050]]
Classification Report:
              precision    recall  f1-score   support

           0       0.60      0.32      0.42      9010
```

```
              1       0.60      0.82      0.69     10990

       accuracy                           0.60     20000
      macro avg       0.60      0.57      0.55     20000
   weighted avg       0.60      0.60      0.57     20000
```

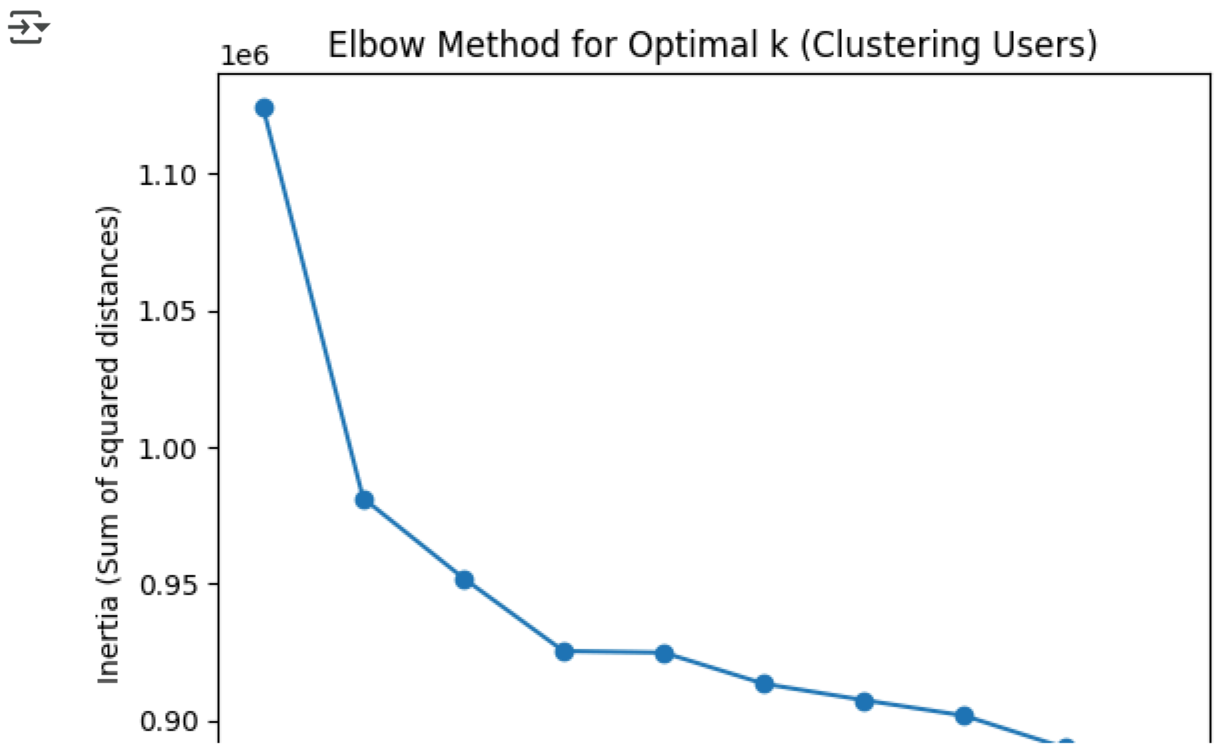
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix heatmap
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False, linewidths=1, linecolor='gra
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```




```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

# Define the hyperparameters to tune
param_grid = {'C': [0.1, 1, 10, 100]}

# Create GridSearchCV object
grid = GridSearchCV(LogisticRegression(max_iter=1000), param_grid, cv=5)

# Fit the model to the training data
grid.fit(X_train, y_train)

# Print best parameters and best accuracy score
print(f"Best Parameters: {grid.best_params_}")
print(f"Best Accuracy: {grid.best_score_:.2f}")
```

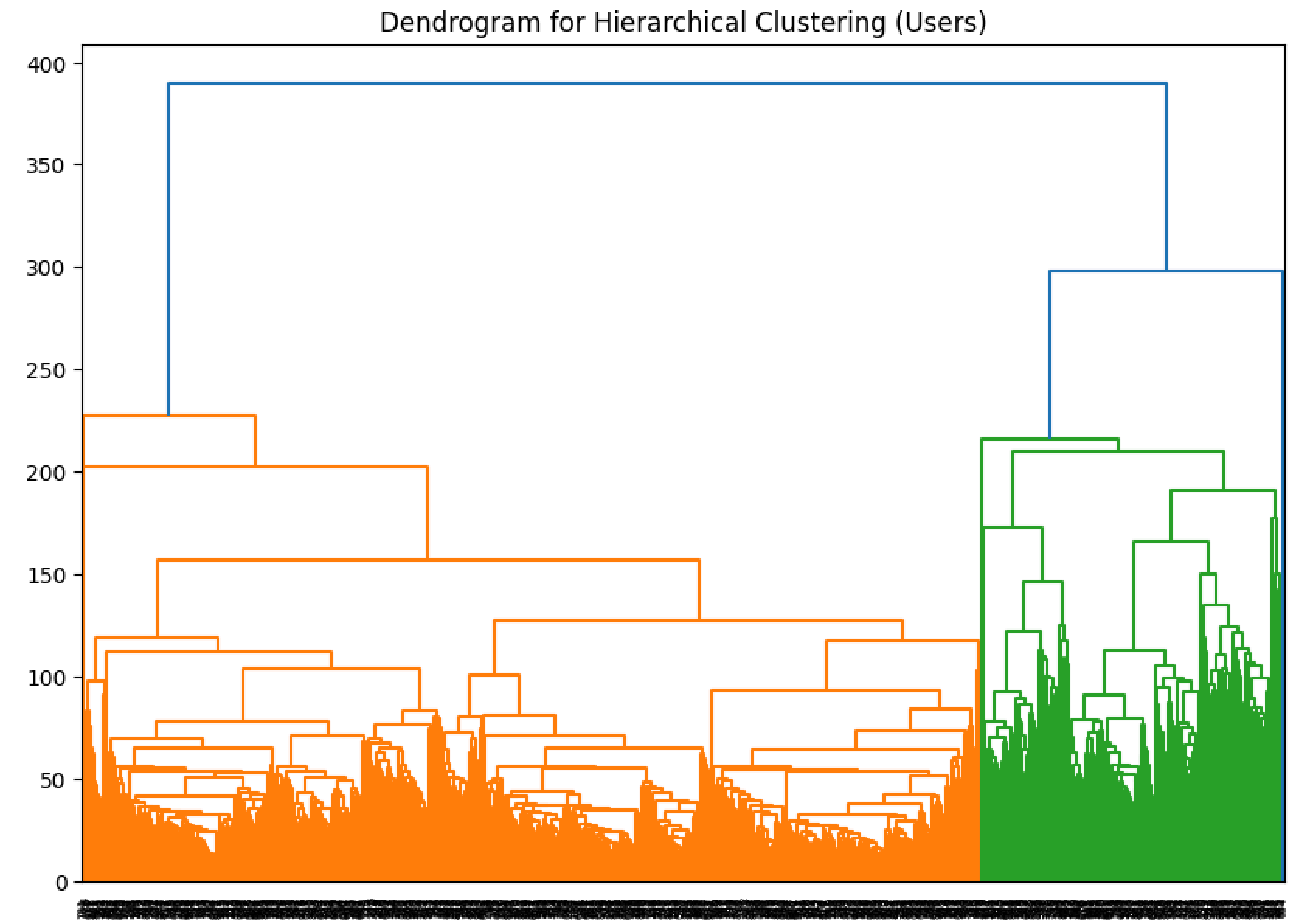
```
Best Parameters: {'C': 0.1}
Best Accuracy: 0.59
```

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split

# Load MovieLens 100k dataset
ratings_path = "./ml-100k/u.data"  # user  item  rating  timestamp
movies_path = "./ml-100k/u.item"   # item|title|release date|...

# Define column names for ratings and movies
ratings_cols = ["user_id", "movie_id", "rating", "timestamp"]
movies_cols = [
    "movie_id", "title", "release_date", "video_release_date", "IMDb_URL",
    "unknown", "Action", "Adventure", "Animation", "Children's", "Comedy",
    "Crime", "Documentary", "Drama", "Fantasy", "Film-Noir", "Horror",
    "Musical", "Mystery", "Romance", "Sci-Fi", "Thriller", "War", "Western"
]

# Load the data
ratings = pd.read_csv(ratings_path, sep="\t", names=ratings_cols)
movies = pd.read_csv(movies_path, sep="|", names=movies_cols, encoding="latin-1")

# Pivot ratings to create a user-item matrix
user_movie_ratings = ratings.pivot(index='user_id', columns='movie_id', values='rating').fi

# Apply KMeans clustering on user-movie ratings matrix
inertia = []
k_range = range(1, 11)  # Test for k values from 1 to 10

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(user_movie_ratings)
    inertia.append(kmeans.inertia_)

# Plot the Elbow graph to find the optimal k
plt.plot(k_range, inertia, marker='o')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia (Sum of squared distances)')
plt.title('Elbow Method for Optimal k (Clustering Users)')
plt.show()

# Let's assume after inspecting the elbow graph, we choose k=4 as optimal
optimal_k = 4
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
user_clusters = kmeans.fit_predict(user_movie_ratings)

# Add cluster labels to the original user data
ratings['user_cluster'] = user_clusters[ratings['user_id'] - 1]

# Display the first few rows of the data with cluster labels
print(ratings.head())
```




```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as sch
from sklearn.preprocessing import StandardScaler

# Load MovieLens 100k dataset
ratings_path = "./ml-100k/u.data"  # user  item  rating  timestamp
movies_path = "./ml-100k/u.item"   # item|title|release date|...

# Define column names for ratings and movies
ratings_cols = ["user_id", "movie_id", "rating", "timestamp"]
movies_cols = [
    "movie_id", "title", "release_date", "video_release_date", "IMDb_URL",
    "unknown", "Action", "Adventure", "Animation", "Children's", "Comedy",
    "Crime", "Documentary", "Drama", "Fantasy", "Film-Noir", "Horror",
    "Musical", "Mystery", "Romance", "Sci-Fi", "Thriller", "War", "Western"
]

# Load the data
ratings = pd.read_csv(ratings_path, sep="\t", names=ratings_cols)
movies = pd.read_csv(movies_path, sep="|", names=movies_cols, encoding="latin-1")

# Pivot ratings to create a user-item matrix (users as rows, movies as columns)
user_movie_ratings = ratings.pivot(index='user_id', columns='movie_id', values='rating').fi

# Standardize the data (optional but can improve results)
scaler = StandardScaler()
user_movie_ratings_scaled = scaler.fit_transform(user_movie_ratings)

# Create the Dendrogram
plt.figure(figsize=(10, 7))
dendrogram = sch.dendrogram(sch.linkage(user_movie_ratings_scaled, method='ward'))
plt.title("Dendrogram for Hierarchical Clustering (Users)")
plt.show()
```

```
# Apply Agglomerative Clustering with n_clusters=4
hc = AgglomerativeClustering(n_clusters=4, affinity='euclidean', linkage='ward')
y_hc = hc.fit_predict(user_movie_ratings_scaled)

# Add cluster labels to the data
ratings['user_cluster'] = y_hc[ratings['user_id'] - 1]

# Visualize the clusters
plt.scatter(user_movie_ratings_scaled[:, 0], user_movie_ratings_scaled[:, 1], c=y_hc, cmap=
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as sch
from sklearn.preprocessing import StandardScaler

# Load MovieLens 100k dataset
ratings_path = "./ml-100k/u.data"  # user  item  rating  timestamp
movies_path = "./ml-100k/u.item"   # item|title|release date|...

# Define column names for ratings and movies
ratings_cols = ["user_id", "movie_id", "rating", "timestamp"]
movies_cols = [
    "movie_id", "title", "release_date", "video_release_date", "IMDb_URL",
    "unknown", "Action", "Adventure", "Animation", "Children's", "Comedy",
    "Crime", "Documentary", "Drama", "Fantasy", "Film-Noir", "Horror",
    "Musical", "Mystery", "Romance", "Sci-Fi", "Thriller", "War", "Western"
]

# Load the data
ratings = pd.read_csv(ratings_path, sep="\t", names=ratings_cols)
movies = pd.read_csv(movies_path, sep="|", names=movies_cols, encoding="latin-1")

# Pivot ratings to create a user-item matrix (users as rows, movies as columns)
user_movie_ratings = ratings.pivot(index='user_id', columns='movie_id', values='rating').fi

# Standardize the data (optional but can improve results)
scaler = StandardScaler()
user_movie_ratings_scaled = scaler.fit_transform(user_movie_ratings)

# Create the Dendrogram
plt.figure(figsize=(10, 7))
dendrogram = sch.dendrogram(sch.linkage(user_movie_ratings_scaled, method='ward'))
plt.title("Dendrogram for Hierarchical Clustering (Users)")
plt.show()

# Apply Agglomerative Clustering with n_clusters=4
hc = AgglomerativeClustering(n_clusters=4, affinity='euclidean', linkage='ward')
y_hc = hc.fit_predict(user_movie_ratings_scaled)

# Add cluster labels to the data
ratings['user_cluster'] = y_hc[ratings['user_id'] - 1]

# Visualize the clusters
plt.scatter(user_movie_ratings_scaled[:, 0], user_movie_ratings_scaled[:, 1], c=y_hc, cmap=
plt.title("Hierarchical Clustering of Users")
plt.xlabel("Feature 1 (scaled)")
plt.ylabel("Feature 2 (scaled)")
plt.show()
```

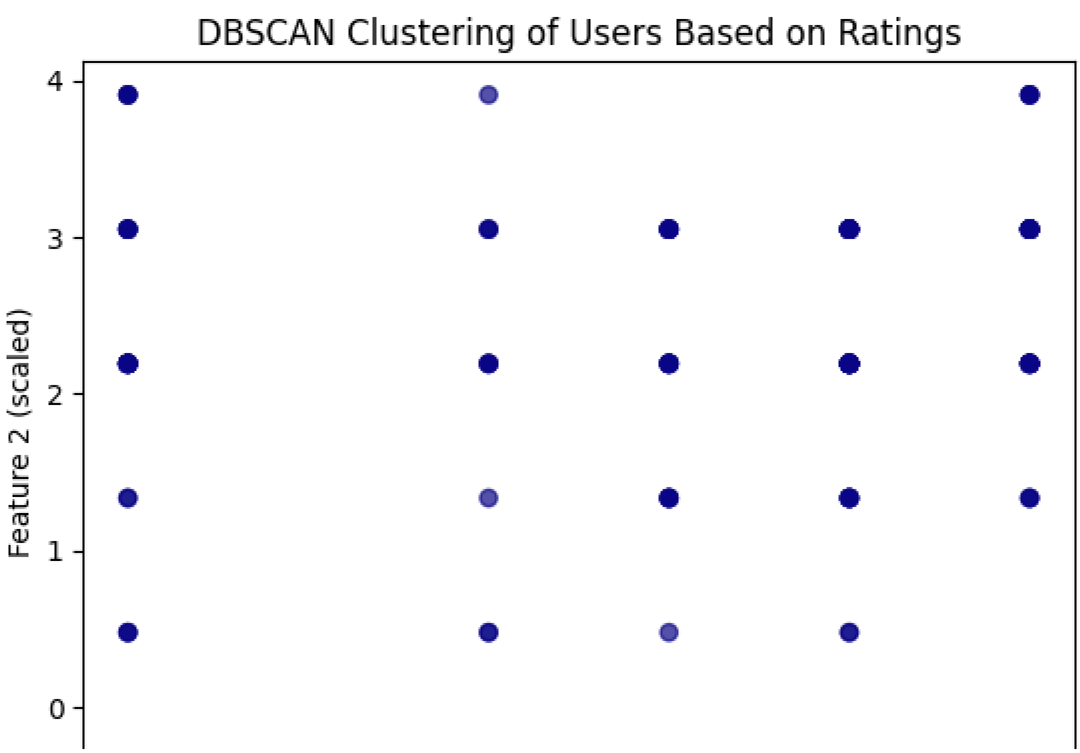



```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-28-1ecb86e74439> in <cell line: 0>()
     37 
     38 # Apply Agglomerative Clustering with n_clusters=4
---> 39 hc = AgglomerativeClustering(n_clusters=4, affinity='euclidean', 
linkage='ward')
     40 y_hc = hc.fit_predict(user_movie_ratings_scaled)
     41 
```

Next steps: Explain error

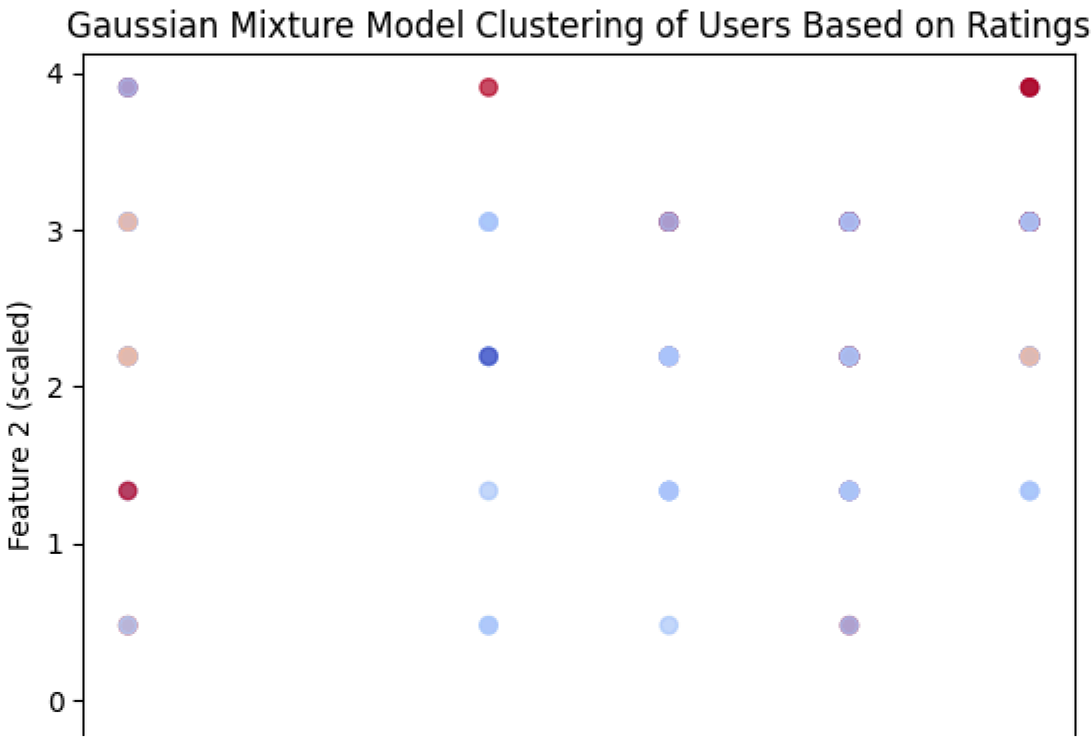
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

# Load MovieLens 100k dataset
ratings_path = "./ml-100k/u.data"   # user  item  rating  timestamp
movies_path = "./ml-100k/u.item"    # item|title|release date|...

# Define column names for ratings and movies
ratings_cols = ["user_id", "movie_id", "rating", "timestamp"]
movies_cols = [
    "movie_id", "title", "release_date", "video_release_date", "IMDb_URL",
    "unknown", "Action", "Adventure", "Animation", "Children's", "Comedy",
    "Crime", "Documentary", "Drama", "Fantasy", "Film-Noir", "Horror",
```

```
    "Musical", "Mystery", "Romance", "Sci-Fi", "Thriller", "War", "Western"
]

# Load the data
ratings = pd.read_csv(ratings_path, sep="\t", names=ratings_cols)
movies = pd.read_csv(movies_path, sep="|", names=movies_cols, encoding="latin-1")

# Pivot ratings to create a user-item matrix (users as rows, movies as columns)
user_movie_ratings = ratings.pivot(index='user_id', columns='movie_id', values='rating').fi

# Standardize the data (optional but can improve results)
scaler = StandardScaler()
user_movie_ratings_scaled = scaler.fit_transform(user_movie_ratings)

# Apply DBSCAN clustering
dbscan = DBSCAN(eps=0.5, min_samples=5)  # eps: maximum distance between samples in a clust
y_dbscan = dbscan.fit_predict(user_movie_ratings_scaled)

# Visualize the clusters
plt.scatter(user_movie_ratings_scaled[:, 0], user_movie_ratings_scaled[:, 1], c=y_dbscan, c
plt.title("DBSCAN Clustering of Users Based on Ratings")
plt.xlabel("Feature 1 (scaled)")
plt.ylabel("Feature 2 (scaled)")
plt.show()

# Add cluster labels to the original ratings data
ratings['user_cluster'] = y_dbscan[ratings['user_id'] - 1]

# Display the first few rows with cluster labels
print(ratings.head())
```



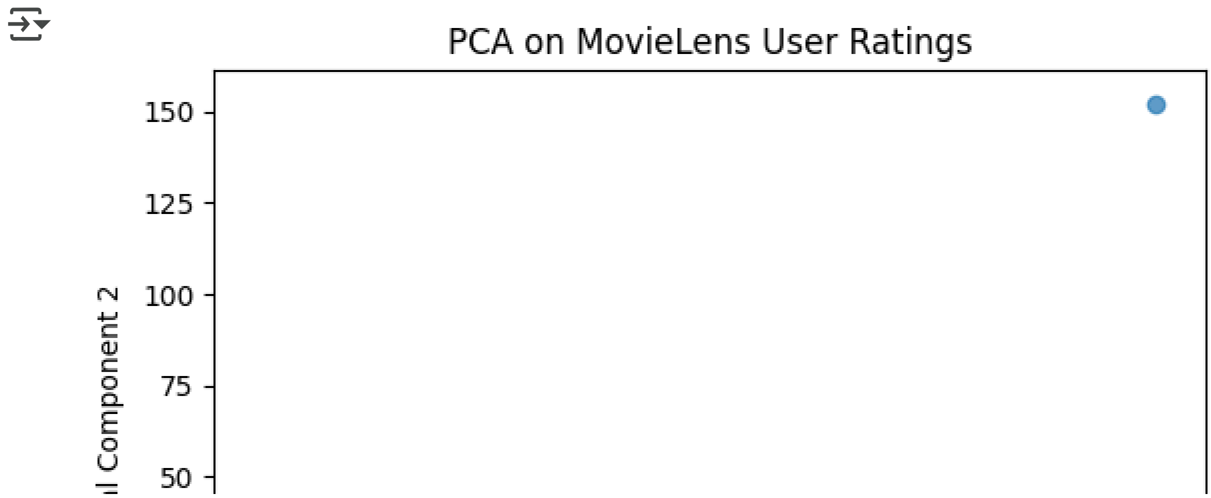

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.preprocessing import StandardScaler
```

```python
# Load MovieLens 100k dataset
ratings_path = "./ml-100k/u.data"  # user  item  rating  timestamp
movies_path = "./ml-100k/u.item"   # item|title|release date|...

# Define column names for ratings and movies
ratings_cols = ["user_id", "movie_id", "rating", "timestamp"]
movies_cols = [
    "movie_id", "title", "release_date", "video_release_date", "IMDb_URL",
    "unknown", "Action", "Adventure", "Animation", "Children's", "Comedy",
    "Crime", "Documentary", "Drama", "Fantasy", "Film-Noir", "Horror",
    "Musical", "Mystery", "Romance", "Sci-Fi", "Thriller", "War", "Western"
]

# Load the data
ratings = pd.read_csv(ratings_path, sep="\t", names=ratings_cols)
movies = pd.read_csv(movies_path, sep="|", names=movies_cols, encoding="latin-1")

# Pivot ratings to create a user-item matrix (users as rows, movies as columns)
user_movie_ratings = ratings.pivot(index='user_id', columns='movie_id', values='rating').fi

# Standardize the data (optional but can improve results)
scaler = StandardScaler()
user_movie_ratings_scaled = scaler.fit_transform(user_movie_ratings)

# Apply GMM (Gaussian Mixture Model) clustering
gmm = GaussianMixture(n_components=4, random_state=42)
y_gmm = gmm.fit_predict(user_movie_ratings_scaled)

# Visualize the clusters
plt.scatter(user_movie_ratings_scaled[:, 0], user_movie_ratings_scaled[:, 1], c=y_gmm, cmap
plt.title("Gaussian Mixture Model Clustering of Users Based on Ratings")
plt.xlabel("Feature 1 (scaled)")
plt.ylabel("Feature 2 (scaled)")
plt.show()

# Add cluster labels to the original ratings data
ratings['user_cluster'] = y_gmm[ratings['user_id'] - 1]

# Display the first few rows with cluster labels
print(ratings.head())
```



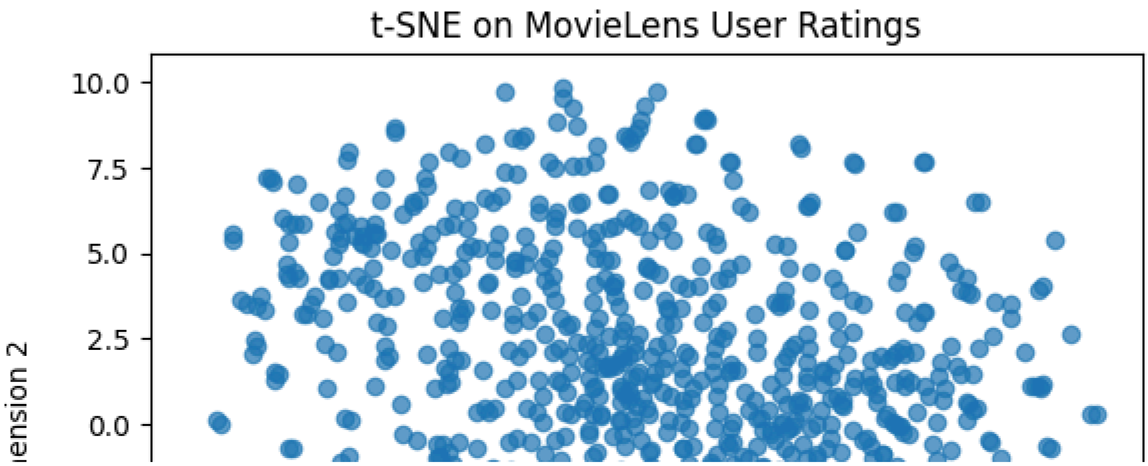

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Load MovieLens 100k dataset
ratings_path = "./ml-100k/u.data"  # user  item  rating  timestamp
movies_path = "./ml-100k/u.item"   # item|title|release date|...

# Define column names for ratings and movies
ratings_cols = ["user_id", "movie_id", "rating", "timestamp"]
movies_cols = [
    "movie_id", "title", "release_date", "video_release_date", "IMDb_URL",
    "unknown", "Action", "Adventure", "Animation", "Children's", "Comedy",
    "Crime", "Documentary", "Drama", "Fantasy", "Film-Noir", "Horror",
    "Musical", "Mystery", "Romance", "Sci-Fi", "Thriller", "War", "Western"
]

# Load the data
ratings = pd.read_csv(ratings_path, sep="\t", names=ratings_cols)
movies = pd.read_csv(movies_path, sep="|", names=movies_cols, encoding="latin-1")

# Pivot ratings to create a user-item matrix (users as rows, movies as columns)
user_movie_ratings = ratings.pivot(index='user_id', columns='movie_id', values='rating').fi

# Standardize the data (important for PCA)
scaler = StandardScaler()
user_movie_ratings_scaled = scaler.fit_transform(user_movie_ratings)

# Apply PCA to reduce the data to 2 dimensions
pca = PCA(n_components=2)
X_pca = pca.fit_transform(user_movie_ratings_scaled)

# Scatter plot of PCA results (users in 2D space)
```

```
plt.scatter(X_pca[:, 0], X_pca[:, 1], alpha=0.7)
plt.title("PCA on MovieLens User Ratings")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```



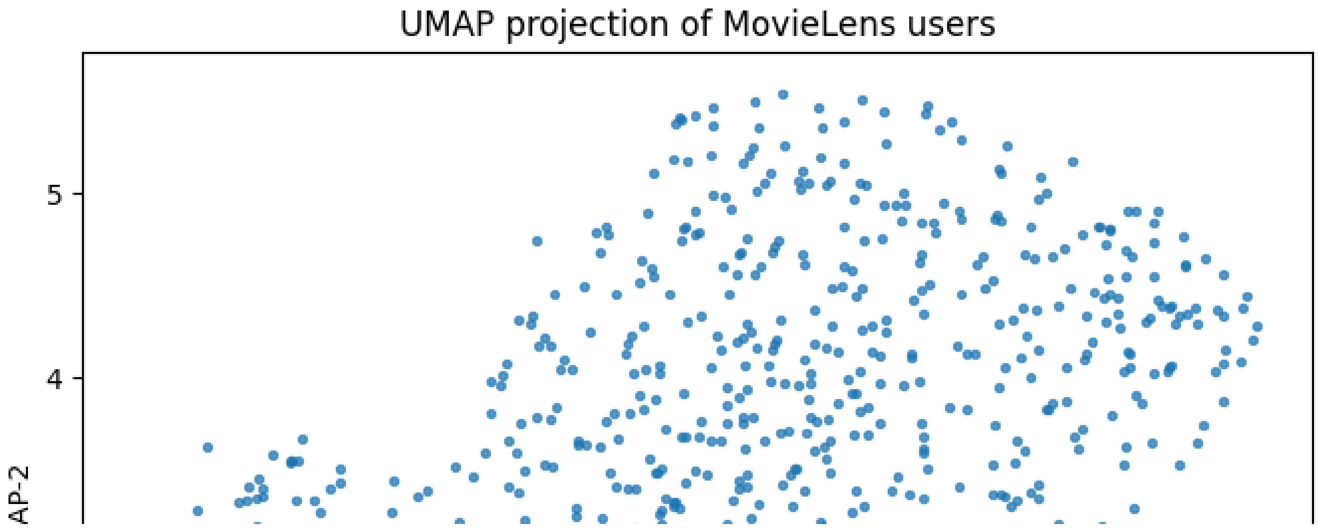

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler

# Load MovieLens 100k dataset
ratings_path = "./ml-100k/u.data"  # user  item  rating  timestamp
movies_path = "./ml-100k/u.item"   # item|title|release date|...

# Define column names for ratings and movies
ratings_cols = ["user_id", "movie_id", "rating", "timestamp"]
movies_cols = [
    "movie_id", "title", "release_date", "video_release_date", "IMDb_URL",
    "unknown", "Action", "Adventure", "Animation", "Children's", "Comedy",
    "Crime", "Documentary", "Drama", "Fantasy", "Film-Noir", "Horror",
    "Musical", "Mystery", "Romance", "Sci-Fi", "Thriller", "War", "Western"
]

# Load the data
ratings = pd.read_csv(ratings_path, sep="\t", names=ratings_cols)
movies = pd.read_csv(movies_path, sep="|", names=movies_cols, encoding="latin-1")

# Pivot ratings to create a user-item matrix (users as rows, movies as columns)
user_movie_ratings = ratings.pivot(index='user_id', columns='movie_id', values='rating').fi

# Standardize the data (important for t-SNE)
scaler = StandardScaler()
user_movie_ratings_scaled = scaler.fit_transform(user_movie_ratings)

# Apply t-SNE to reduce the data to 2 dimensions
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(user_movie_ratings_scaled)

# Scatter plot of t-SNE results (users in 2D space)
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], alpha=0.7)
plt.title("t-SNE on MovieLens User Ratings")
plt.xlabel("Dimension 1")
```

```
plt.ylabel("Dimension 2")
plt.show()
```




```
import pandas as pd
import matplotlib.pyplot as plt
import umap
from sklearn.preprocessing import StandardScaler

# ── 1. Load MovieLens-100k ──────────────────────────────────────────────
ratings_path = "./ml-100k/u.data"          # user  item  rating  timestamp
movies_path  = "./ml-100k/u.item"          # item|title|…

ratings_cols = ["user_id", "movie_id", "rating", "timestamp"]
movies_cols  = [
    "movie_id","title","release_date","video_release_date","IMDb_URL",
    "unknown","Action","Adventure","Animation","Children's","Comedy",
    "Crime","Documentary","Drama","Fantasy","Film-Noir","Horror",
    "Musical","Mystery","Romance","Sci-Fi","Thriller","War","Western"
]

ratings = pd.read_csv(ratings_path, sep="\t", names=ratings_cols)
movies  = pd.read_csv(movies_path,  sep="|", names=movies_cols, encoding="latin-1")

# ── 2. Build user-item matrix ──────────────────────────────────────────
user_movie = ratings.pivot(index="user_id",
                           columns="movie_id",
                           values="rating").fillna(0)

# ── 3. Standardize (helps UMAP) ────────────────────────────────────────
X_scaled = StandardScaler().fit_transform(user_movie)

# ── 4. Run UMAP to 2D ──────────────────────────────────────────────────
reducer = umap.UMAP(n_components=2, random_state=42)
X_umap   = reducer.fit_transform(X_scaled)

# ── 5. Visualize users in 2-D preference space ────────────────────────
plt.figure(figsize=(8,6))
plt.scatter(X_umap[:,0], X_umap[:,1], s=8, alpha=0.7)
plt.title("UMAP projection of MovieLens users")
plt.xlabel("UMAP-1"); plt.ylabel("UMAP-2")
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: Future
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/umap/umap_.py:1952: UserWarning: n_jobs
  warn(
```

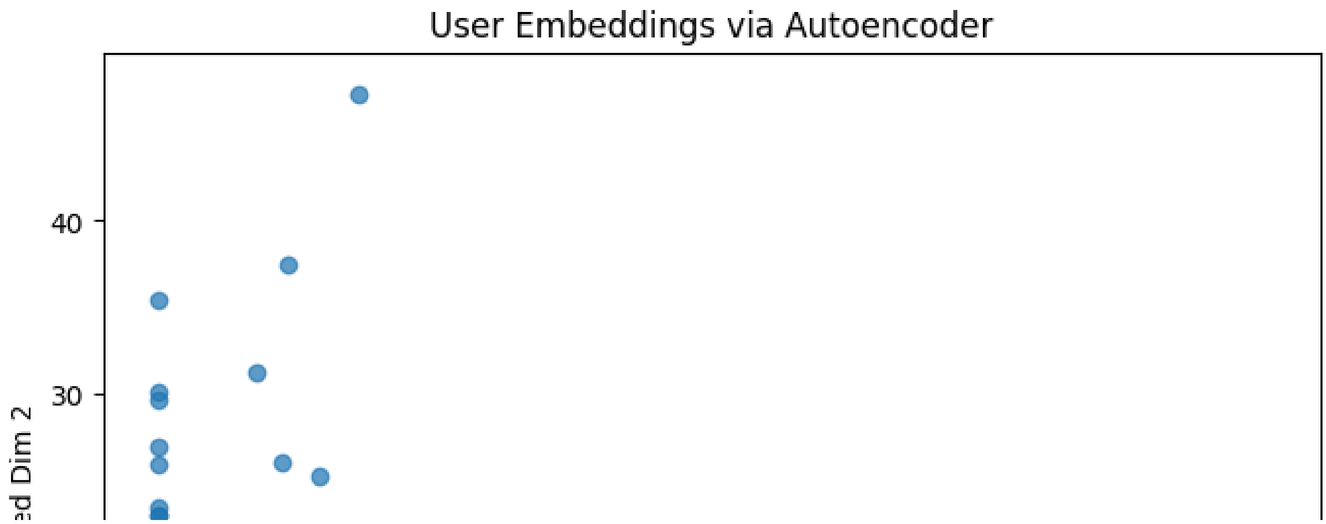



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from sklearn.preprocessing import StandardScaler

# 1. Load MovieLens dataset
ratings_path = './ml-100k/u.data'
ratings_cols = ['user_id', 'movie_id', 'rating', 'timestamp']
ratings = pd.read_csv(ratings_path, sep='\t', names=ratings_cols)

# 2. Create user-movie matrix
user_movie_matrix = ratings.pivot(index='user_id', columns='movie_id', values='rating').fil
X = user_movie_matrix.values  # Users as rows, movies as features

# 3. Normalize data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. Define Autoencoder architecture
input_dim = X_scaled.shape[1]
encoding_dim = 32

input_layer = keras.layers.Input(shape=(input_dim,))
encoded = keras.layers.Dense(encoding_dim, activation='relu')(input_layer)
decoded = keras.layers.Dense(input_dim, activation='sigmoid')(encoded)

autoencoder = keras.models.Model(input_layer, decoded)
autoencoder.compile(optimizer='adam', loss='mse')

# 5. Train the Autoencoder
autoencoder.fit(X_scaled, X_scaled, epochs=30, batch_size=32, shuffle=True, verbose=1)

# 6. Encode user representations
encoder = keras.models.Model(input_layer, encoded)
X_encoded = encoder.predict(X_scaled)
```

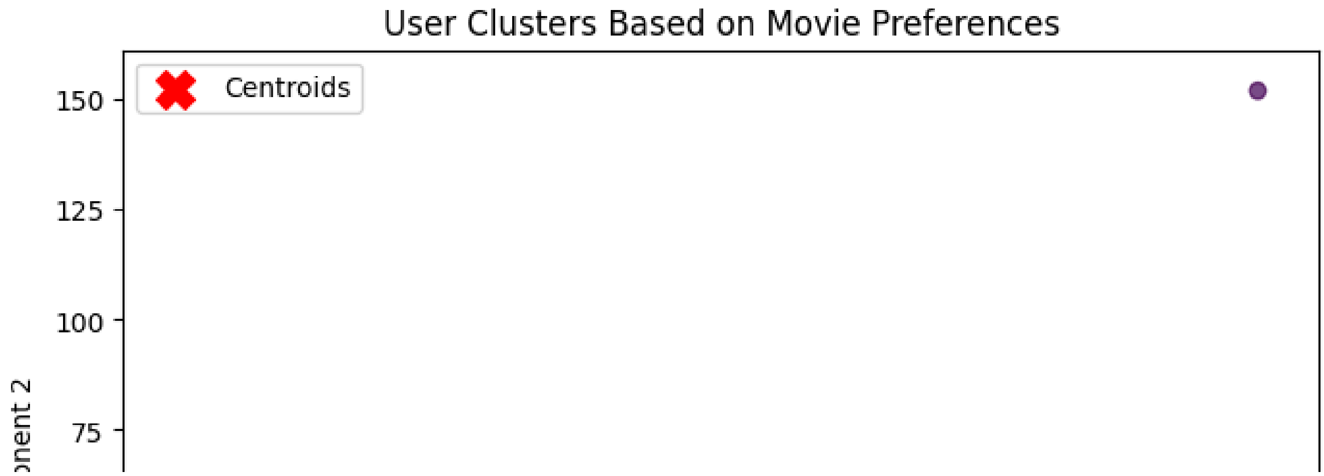
```
# 7. Visualize compressed user embeddings (first 2 components)
plt.figure(figsize=(8,6))
plt.scatter(X_encoded[:, 0], X_encoded[:, 1], alpha=0.7, cmap='plasma')
plt.title("User Embeddings via Autoencoder")
plt.xlabel("Encoded Dim 1")
plt.ylabel("Encoded Dim 2")
plt.show()
```

```
Epoch 1/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 1.1969
Epoch 2/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.9311
Epoch 3/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.9068
Epoch 4/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.9600
Epoch 5/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.8835
Epoch 6/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.9062
Epoch 7/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.9157
Epoch 8/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.8511
Epoch 9/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.9387
Epoch 10/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.9184
Epoch 11/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.8731
Epoch 12/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.8263
Epoch 13/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.9123
Epoch 14/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.8077
Epoch 15/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.8451
Epoch 16/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.8462
Epoch 17/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.7953
Epoch 18/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.8629
Epoch 19/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.8861
Epoch 20/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.8386
Epoch 21/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.7782
Epoch 22/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.8489
Epoch 23/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.8183
Epoch 24/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.8331
Epoch 25/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.8659
Epoch 26/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.8148
Epoch 27/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.8371
Epoch 28/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.8243
```

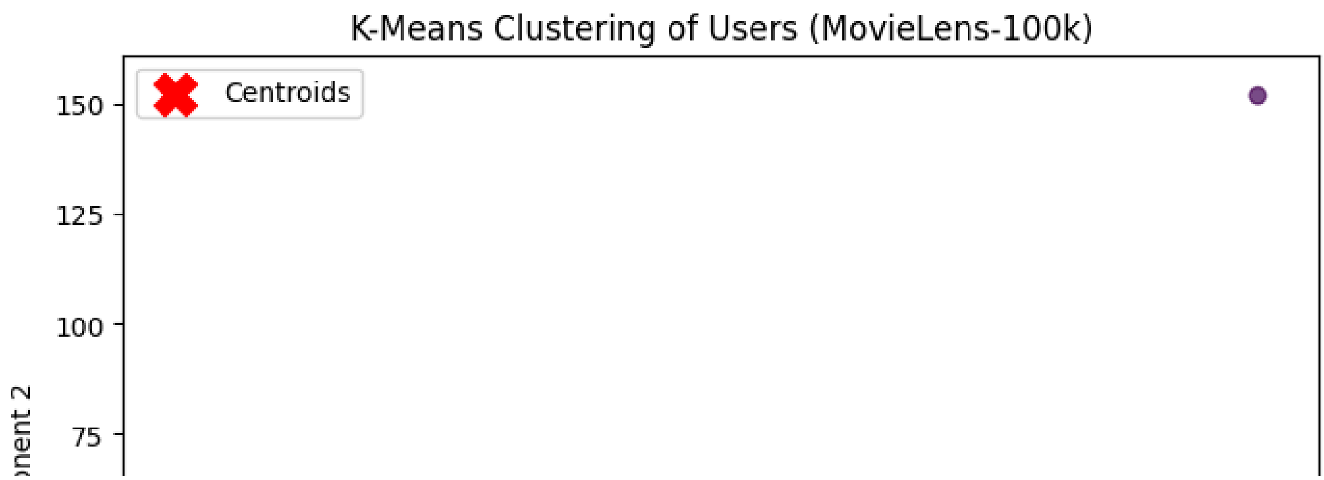
```
Epoch 29/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.7806
Epoch 30/30
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.8996
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
<ipython-input-34-8edf82438b2d>:41: UserWarning: No data for colormapping provid
  plt.scatter(X_encoded[:, 0], X_encoded[:, 1], alpha=0.7, cmap='plasma')
```




```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# 1. Load MovieLens ratings data
ratings = pd.read_csv('./ml-100k/u.data', sep='\t', names=['user_id', 'movie_id', 'rating',

# 2. Create a user-movie rating matrix
user_movie_matrix = ratings.pivot(index='user_id', columns='movie_id', values='rating').fil
X = user_movie_matrix.values  # rows = users, columns = movie ratings

# 3. Standardize data
X_scaled = StandardScaler().fit_transform(X)

# 4. Apply K-Means clustering
kmeans = KMeans(n_clusters=5, random_state=42)
clusters = kmeans.fit_predict(X_scaled)

# 5. Visualize clusters using PCA (for 2D plot)
from sklearn.decomposition import PCA
X_pca = PCA(n_components=2).fit_transform(X_scaled)

plt.figure(figsize=(8,6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters, cmap='viridis', alpha=0.7)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            s=200, c='red', marker='X', label='Centroids')
plt.title("User Clusters Based on Movie Preferences")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.legend()
plt.show()
```




```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# ── 1. Load MovieLens-100k ratings ────────────────────────────────
ratings = pd.read_csv(
    "./ml-100k/u.data",
    sep="\t",
    names=["user_id", "movie_id", "rating", "timestamp"]
)

# ── 2. Build a user-movie matrix (rows = users, cols = movies) ─
user_movie = ratings.pivot(index="user_id",
                           columns="movie_id",
                           values="rating").fillna(0)

# ── 3. Prepare features (X) ───────────────────────────────────────
X = user_movie.values               # shape: (n_users, n_movies)
X_scaled = StandardScaler().fit_transform(X)

# ── 4. K-Means clustering (choose k=4 to mirror your 4 blobs) ─
kmeans = KMeans(n_clusters=4, random_state=42)
clusters = kmeans.fit_predict(X_scaled)

# ── 5. 2-D visualization via PCA ──────────────────────────────────
X_pca = PCA(n_components=2).fit_transform(X_scaled)

plt.figure(figsize=(8,6))
plt.scatter(X_pca[:,0], X_pca[:,1], c=clusters, cmap="viridis", alpha=0.7)
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1],
            s=250, c="red", marker="X", label="Centroids")
plt.title("K-Means Clustering of Users (MovieLens-100k)")
plt.xlabel("PCA component 1")
plt.ylabel("PCA component 2")
plt.legend()
plt.show()
```

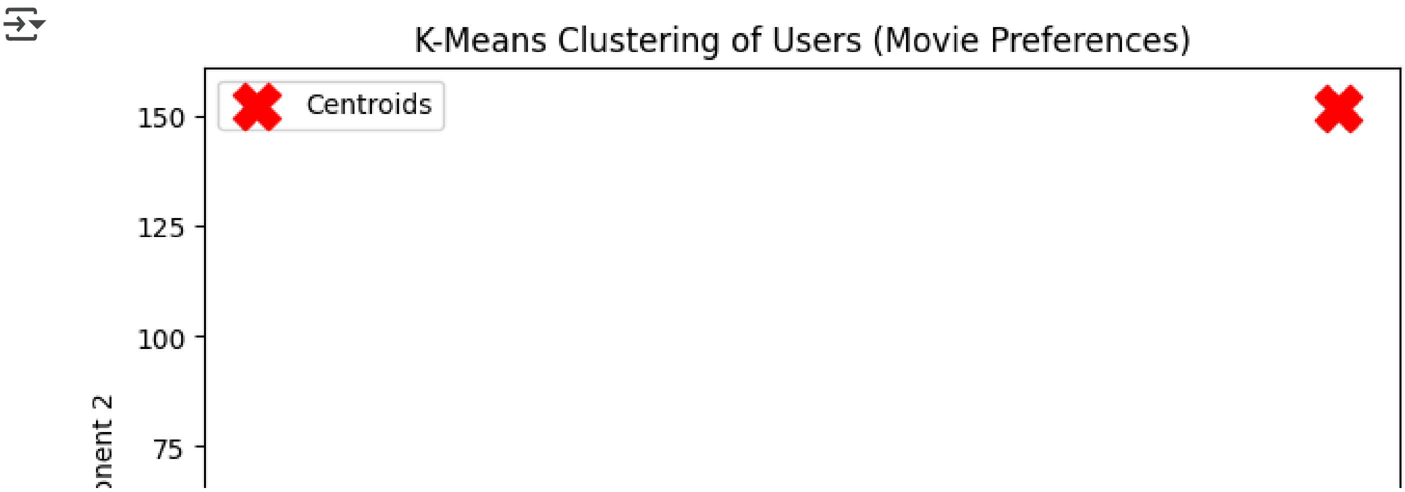



```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Step 1: Load MovieLens-100k dataset
ratings = pd.read_csv('./ml-100k/u.data', sep='\t', names=['user_id', 'movie_id', 'rating',

# Step 2: Create user-movie matrix (users as rows, movies as columns)
user_movie_matrix = ratings.pivot(index='user_id', columns='movie_id', values='rating').fil

# Step 3: Prepare feature matrix
X = user_movie_matrix.values

# Step 4: Normalize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 5: Apply K-Means clustering
kmeans = KMeans(n_clusters=4, random_state=42)
y_kmeans = kmeans.fit_predict(X_scaled)

# Optional: Add cluster labels to users
user_clusters = pd.DataFrame({'user_id': user_movie_matrix.index, 'cluster': y_kmeans})
print(user_clusters.head())
```

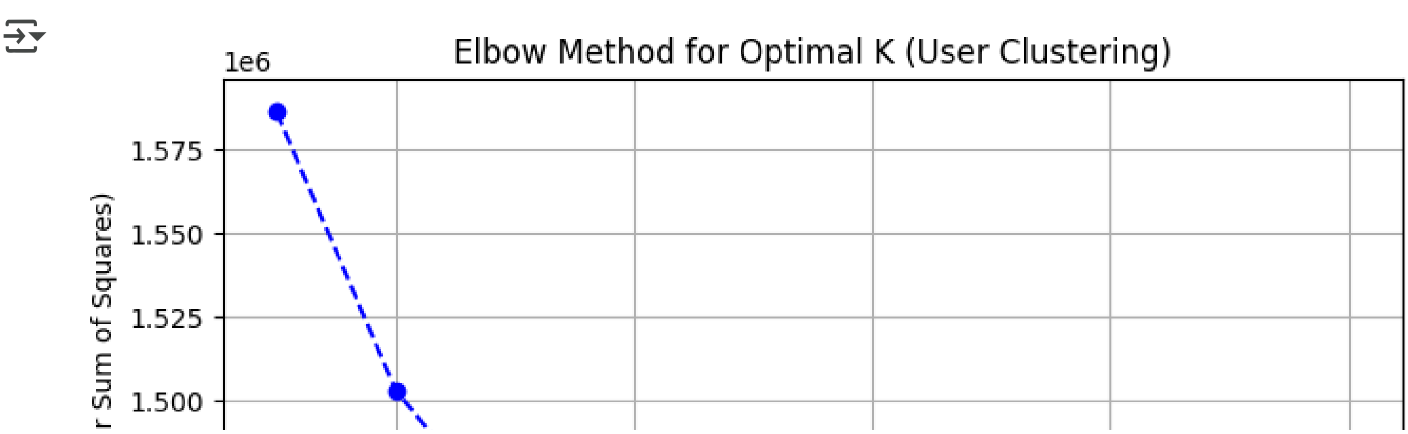
	user_id	cluster
0	1	1
1	2	2
2	3	2
3	4	2
4	5	1

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Reduce to 2D using PCA for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Plot the clusters
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_kmeans, cmap='viridis', alpha=0.7)
```

```
# Reduce centroids to 2D as well
centroids_2d = pca.transform(kmeans.cluster_centers_)
plt.scatter(centroids_2d[:, 0], centroids_2d[:, 1],
            s=300, c='red', marker='X', label="Centroids")

plt.legend()
plt.title("K-Means Clustering of Users (Movie Preferences)")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.show()
```

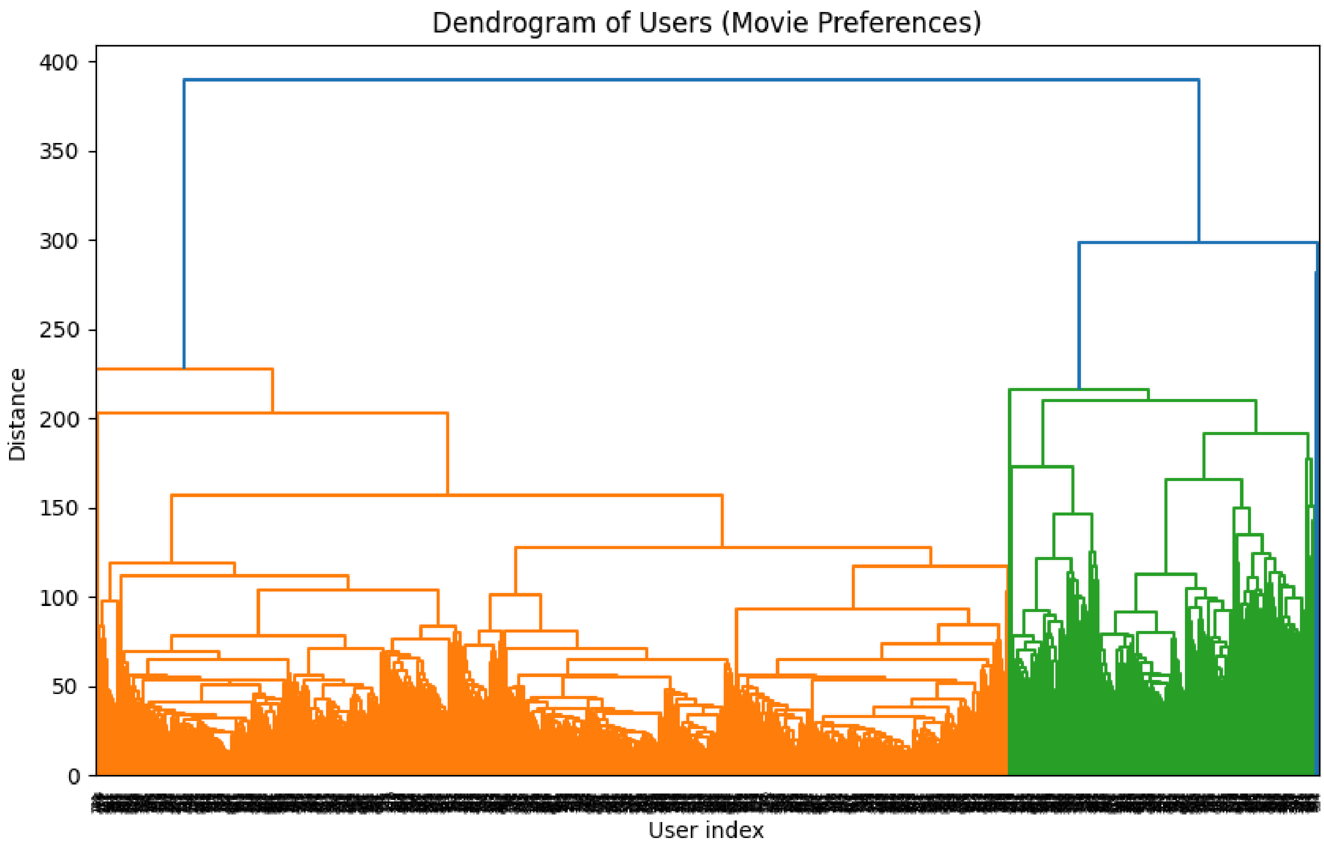



```
wcss = []  # Within-cluster sum of squares
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)  # Use normalized data
    wcss.append(kmeans.inertia_)

# Plot the elbow graph
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--', color='blue')
plt.xlabel("Number of Clusters (K)")
plt.ylabel("WCSS (Within-Cluster Sum of Squares)")
plt.title("Elbow Method for Optimal K (User Clustering)")
plt.grid(True)
plt.show()
```




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

```python
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# ── 1. Load MovieLens-100k ratings ───────────────────────────────────
ratings = pd.read_csv(
    "./ml-100k/u.data",
    sep="\t",
    names=["user_id", "movie_id", "rating", "timestamp"]
)

# ── 2. Build user-movie matrix (rows=user, cols=movie) ────────────
user_movie = ratings.pivot(index="user_id",
                           columns="movie_id",
                           values="rating").fillna(0)

# ── 3. Scale features ──────────────────────────────────────────────────
X = StandardScaler().fit_transform(user_movie.values)

# ── 4. Dendrogram (Ward linkage) ──────────────────────────────────────
plt.figure(figsize=(10, 6))
sch.dendrogram(sch.linkage(X, method="ward"))
plt.title("Dendrogram of Users (Movie Preferences)")
plt.xlabel("User index")
plt.ylabel("Distance")
plt.show()

# ── 5. Agglomerative clustering: choose, e.g., 4 clusters ────────
hc = AgglomerativeClustering(n_clusters=4, affinity="euclidean", linkage="ward")
user_labels = hc.fit_predict(X)

# ── 6. Visualize clusters in 2-D with PCA ───────────────────────────
X_pca = PCA(n_components=2).fit_transform(X)

plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=user_labels, cmap="rainbow", alpha=0.7)
plt.title("Hierarchical Clustering of Users (2-D PCA view)")
plt.xlabel("PCA-1"); plt.ylabel("PCA-2")
plt.show()

# ── 7. Attach cluster IDs back to user IDs (optional) ─────────────
user_clusters = pd.DataFrame({
    "user_id": user_movie.index,
    "cluster": user_labels
})
print(user_clusters.head())
```

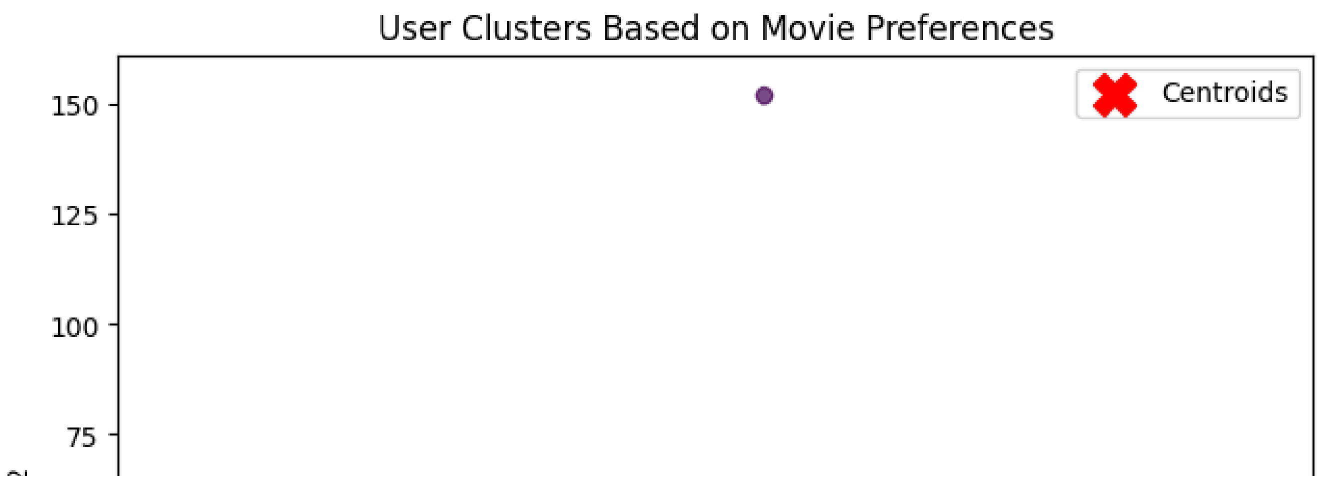



```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-40-862806633d43> in <cell line: 0>()
     31 
     32 # ── 5. Agglomerative clustering: choose, e.g., 4 clusters ─────────
---> 33 hc = AgglomerativeClustering(n_clusters=4, affinity="euclidean", 
linkage="ward")
     34 user_labels = hc.fit_predict(X)
     35 

TypeError: AgglomerativeClustering.__init__() got an unexpected keyword argument 
'affinity'
```

Next steps: Explain error

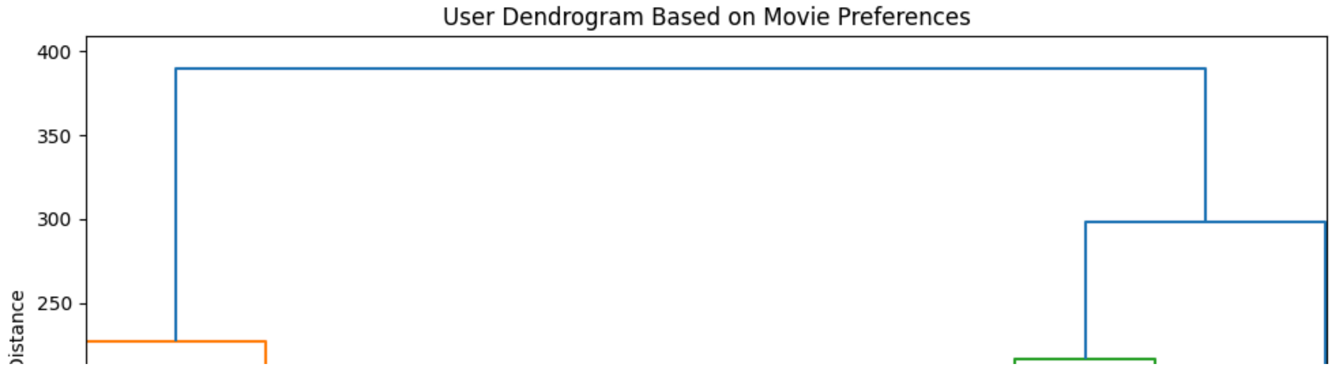
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# 1 )  Load ratings ────────────────────────────────────────────────────
ratings = pd.read_csv(
    "./ml-100k/u.data",
    sep="\t",
    names=["user_id", "movie_id", "rating", "timestamp"]
)

# 2 )  Build user-movie matrix (rows = users, cols = movies) ────────
user_movie = ratings.pivot(index="user_id",
                           columns="movie_id",
```

```
                                   values="rating").fillna(0)

# 3 )  Scale features (very important for clustering) ───────────────
X = StandardScaler().fit_transform(user_movie.values)

# 4 )  Run K-Means (k = 4 to mirror your 4 synthetic blobs) ────────
kmeans   = KMeans(n_clusters=4, random_state=42)
clusters = kmeans.fit_predict(X)

# 5 )  2-D plot with PCA for visualization only  ──────────────────────
X_2d = PCA(n_components=2).fit_transform(X)

plt.figure(figsize=(8,6))
plt.scatter(X_2d[:,0], X_2d[:,1], c=clusters, cmap="viridis", alpha=0.7)
centroids_2d = PCA(n_components=2).fit_transform(kmeans.cluster_centers_)
plt.scatter(centroids_2d[:,0], centroids_2d[:,1],
            s=250, c="red", marker="X", label="Centroids")
plt.title("User Clusters Based on Movie Preferences")
plt.xlabel("PCA-1"); plt.ylabel("PCA-2")
plt.legend(); plt.show()
```

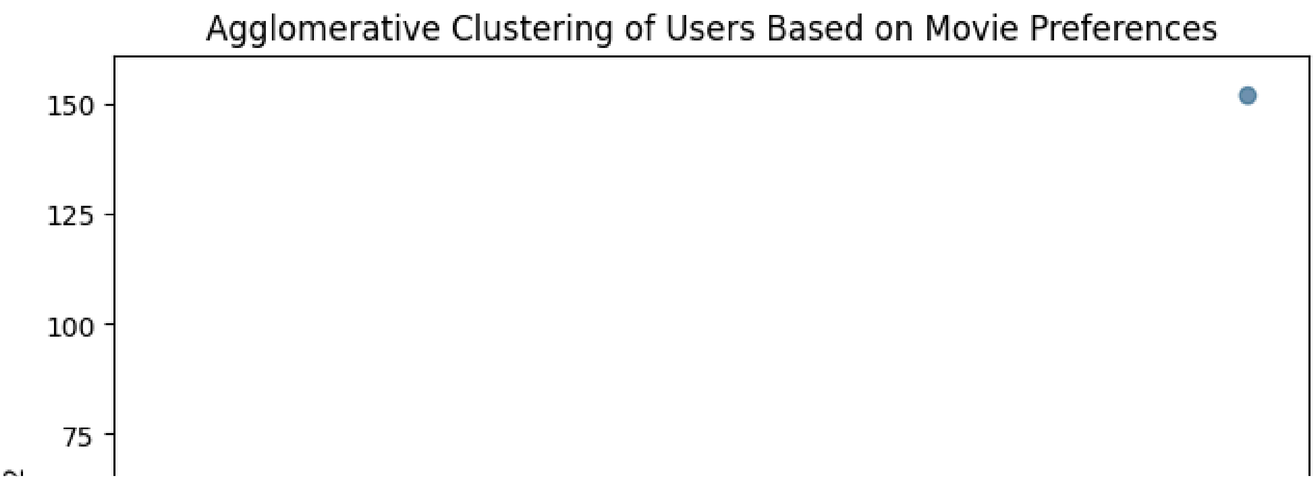



```
import pandas as pd
import matplotlib.pyplot as plt
import scipy.cluster.hierarchy as sch
from sklearn.preprocessing import StandardScaler

# 1. Load MovieLens-100k ratings
ratings = pd.read_csv(
    "./ml-100k/u.data",
    sep="\t",
    names=["user_id", "movie_id", "rating", "timestamp"]
)

# 2. Build user-movie matrix
user_movie = ratings.pivot(index="user_id", columns="movie_id", values="rating").fillna(0)

# 3. Normalize the matrix
X_scaled = StandardScaler().fit_transform(user_movie)

# 4. Plot dendrogram
plt.figure(figsize=(10, 6))
dendrogram = sch.dendrogram(sch.linkage(X_scaled, method='ward'))
plt.title("User Dendrogram Based on Movie Preferences")
plt.xlabel("User Index")
```

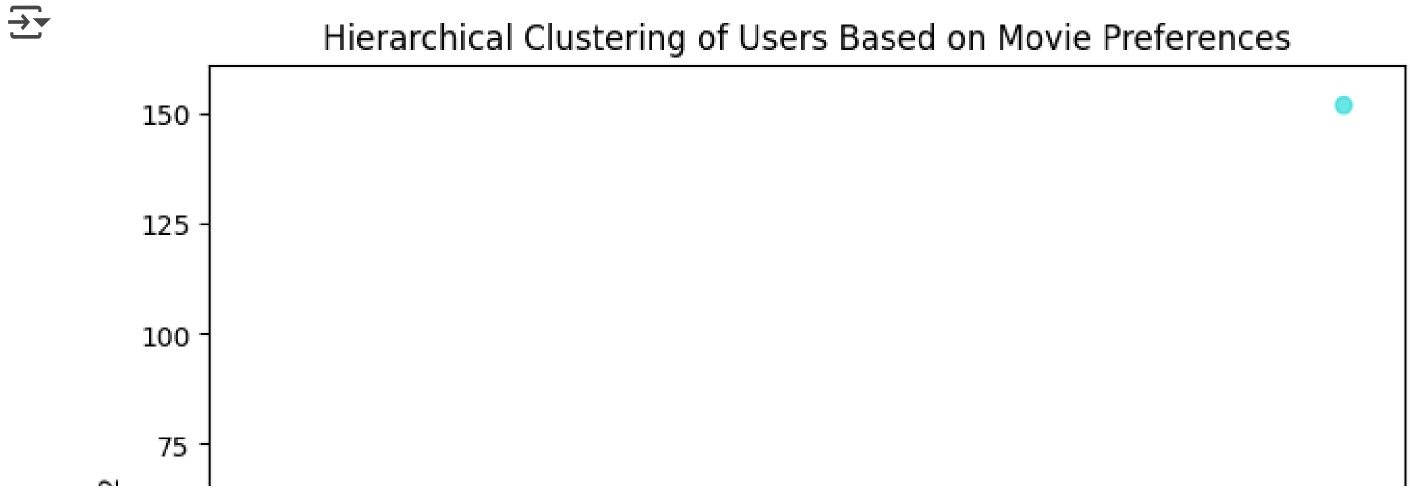
```
plt.ylabel("Euclidean Distance")
plt.tight_layout()
plt.show()
```



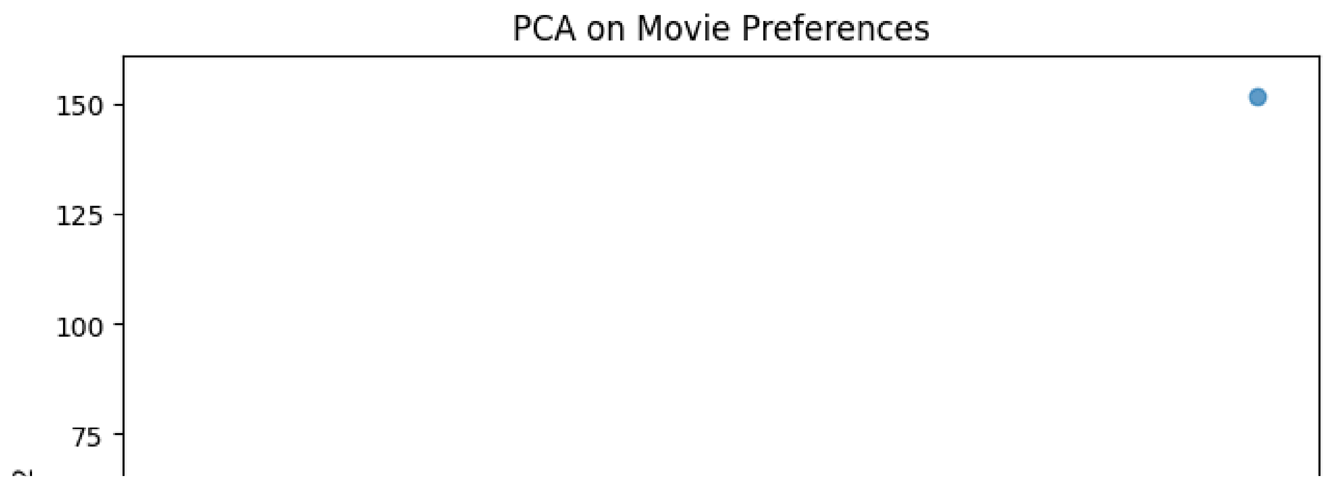

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler

# 1. Load MovieLens-100k ratings
ratings = pd.read_csv(
    "./ml-100k/u.data",
    sep="\t",
    names=["user_id", "movie_id", "rating", "timestamp"]
)

# 2. Create user-movie matrix
user_movie = ratings.pivot(index="user_id", columns="movie_id", values="rating").fillna(0)

# 3. Normalize the matrix
X_scaled = StandardScaler().fit_transform(user_movie)

# 4. Apply Agglomerative Clustering
hc = AgglomerativeClustering(n_clusters=4, linkage='ward')
y_hc = hc.fit_predict(X_scaled)

# 5. Visualize clusters using PCA (reduce to 2D for plotting)
from sklearn.decomposition import PCA
X_pca = PCA(n_components=2).fit_transform(X_scaled)

plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_hc, cmap="viridis", alpha=0.7)
plt.title("Agglomerative Clustering of Users Based on Movie Preferences")
plt.xlabel("PCA-1")
plt.ylabel("PCA-2")
plt.show()
```




```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# 1. Load MovieLens-100k ratings
ratings = pd.read_csv(
    "./ml-100k/u.data",
    sep="\t",
    names=["user_id", "movie_id", "rating", "timestamp"]
)

# 2. Create user-movie matrix
user_movie = ratings.pivot(index="user_id", columns="movie_id", values="rating").fillna(0)

# 3. Normalize the matrix
X_scaled = StandardScaler().fit_transform(user_movie)

# 4. Apply Agglomerative Clustering
hc = AgglomerativeClustering(n_clusters=4, linkage='ward')
y_hc = hc.fit_predict(X_scaled)

# 5. Visualize clusters using PCA (reduce to 2D for plotting)
X_pca = PCA(n_components=2).fit_transform(X_scaled)

# 6. Plot the clusters with Hierarchical Clustering labels
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_hc, cmap='rainbow', alpha=0.7)
plt.title("Hierarchical Clustering of Users Based on Movie Preferences")
plt.xlabel("PCA-1")
plt.ylabel("PCA-2")
plt.show()
```




```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# 1. Load MovieLens-100k ratings data
ratings = pd.read_csv(
    "./ml-100k/u.data",  # Adjust this path if needed
    sep="\t",
    names=["user_id", "movie_id", "rating", "timestamp"]
)

# 2. Build user-movie matrix (rows = users, columns = movies)
user_movie = ratings.pivot(index="user_id", columns="movie_id", values="rating").fillna(0)

# 3. Normalize the matrix (important for PCA)
X_scaled = StandardScaler().fit_transform(user_movie)

# 4. Apply PCA (reduce to 2D for visualization)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# 5. Plot the 2D representation
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], alpha=0.7)
plt.title("PCA on Movie Preferences")
plt.xlabel("PCA-1")
plt.ylabel("PCA-2")
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import load_digits
from sklearn.preprocessing import StandardScaler

# 1. Load the Digits dataset
digits = load_digits()
X = digits.data  # Features (64-dimensional)
y = digits.target  # Labels (digits 0-9)

# 2. Normalize the data (important for PCA)
X_scaled = StandardScaler().fit_transform(X)

# 3. Apply PCA (reduce to 2D for visualization)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# 4. Scatter plot of the PCA result (2D representation)
plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', alpha=0.7)

# 5. Add a color bar for digit labels
plt.colorbar(scatter, label="Digit Label")
plt.title("PCA of Digits Dataset")
plt.xlabel("PCA-1")
plt.ylabel("PCA-2")
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import load_digits
from sklearn.preprocessing import StandardScaler

# 1. Load the Digits dataset
digits = load_digits()
X = digits.data  # Features (64-dimensional)
y = digits.target  # Labels (digits 0-9)

# 2. Standardize the features (important for PCA)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```