```
# ==========================
# Install & import packages
# ==========================
!pip install requests pandas numpy scikit-learn seaborn matplotlib
!pip install fuzzywuzzy[speedup]

import requests
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import time
import itertools

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
from sklearn.decomposition import PCA

from fuzzywuzzy import process

from google.colab import drive
from google.colab import userdata

sns.set(style="whitegrid")

# ==========================
# API Key
# ==========================
API_KEY = userdata.get('TMDB_Key')  # or hardcode your key here
```

```
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (2.32.3)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.0.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-packages (0.13.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests) (2025.6.15)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.15.3)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.5.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.58.4)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.
Collecting fuzzywuzzy[speedup]
  Downloading fuzzywuzzy-0.18.0-py2.py3-none-any.whl.metadata (4.9 kB)
Collecting python-levenshtein>=0.12 (from fuzzywuzzy[speedup])
  Downloading python_levenshtein-0.27.1-py3-none-any.whl.metadata (3.7 kB)
Collecting Levenshtein==0.27.1 (from python-levenshtein>=0.12->fuzzywuzzy[speedup])
  Downloading levenshtein-0.27.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.6 kB)
Collecting rapidfuzz<4.0.0,>=3.9.0 (from Levenshtein==0.27.1->python-levenshtein>=0.12->fuzzywuzzy[speedup])
  Downloading rapidfuzz-3.13.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (12 kB)
Downloading python_levenshtein-0.27.1-py3-none-any.whl (9.4 kB)
Downloading levenshtein-0.27.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (161 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 161.7/161.7 kB 4.0 MB/s eta 0:00:00
Downloading fuzzywuzzy-0.18.0-py2.py3-none-any.whl (18 kB)
Downloading rapidfuzz-3.13.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.1/3.1 MB 40.1 MB/s eta 0:00:00
Installing collected packages: fuzzywuzzy, rapidfuzz, Levenshtein, python-levenshtein
Successfully installed Levenshtein-0.27.1 fuzzywuzzy-0.18.0 python-levenshtein-0.27.1 rapidfuzz-3.13.0
```

Data collection

```
# ==========================
# Get TV shows data from TMDB
# ==========================
def get_tv_shows(api_key, pages=5, endpoint="tv/popular"):
    BASE_URL = 'https://api.themoviedb.org/3'
```

```
    dataset = []

    for page in range(1, pages+1):
        url = f"{BASE_URL}/{endpoint}?api_key={api_key}&language=en-US&page={page}"
        response = requests.get(url)
        if response.status_code != 200:
            print("Error:", response.json())
            continue
        shows = response.json()['results']

        for show in shows:
            show_id = show['id']
            details_url = f"{BASE_URL}/tv/{show_id}?api_key={api_key}&language=en-US"
            details = requests.get(details_url).json()

            genres = [g['name'] for g in details.get('genres', [])]
            episode_time = details.get('episode_run_time', [0])[0] if details.get('episode_run_time') else 0

            dataset.append({
                'name': show.get('name'),
                'rating': show.get('vote_average'),
                'episode_run_time': episode_time,
                'genres': genres,
                'popularity': show.get('popularity')
            })

            time.sleep(0.2)  # avoid hitting rate limits

    return pd.DataFrame(dataset)

# Fetch data
df = get_tv_shows(API_KEY, pages=5)
print(df.head())
```

```
                                  name  rating  episode_run_time  \
0                           Squid Game   7.900                 0
1  The Late Late Show with Craig Ferguson   6.785                60
2      The Late Show with Stephen Colbert   6.362                 0
3                        The Daily Show   6.367                30
4                         Among Friends   3.200                12

                               genres  popularity
0  [Action & Adventure, Mystery, Drama]   2111.7522
1                      [Comedy, Talk]    805.4902
2                      [Comedy, Talk]    708.1901
3                      [News, Comedy]    659.9982
4                     [Drama, Family]    633.2599
```

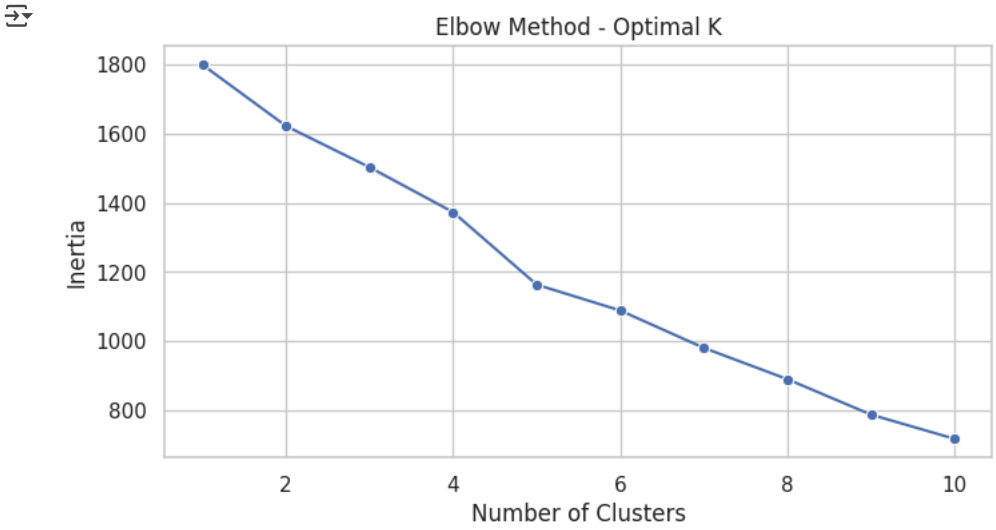
Data preparation & feature engineering

```
# ==========================
# Handle missing genres & one-hot encoding
# ==========================
df['genres'] = df['genres'].apply(lambda x: x if isinstance(x, list) else [])

all_genres = set(g for sublist in df['genres'] for g in sublist)
for genre in all_genres:
    df[genre] = df['genres'].apply(lambda x: 1 if genre in x else 0)

# ==========================
# Prepare features for clustering
# ==========================
X = df[['rating', 'episode_run_time', 'popularity'] + list(all_genres)]
X.fillna(X.mean(), inplace=True)

# ==========================
# Scale data
# ==========================
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
/tmp/ipython-input-3-4145276992.py:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  X.fillna(X.mean(), inplace=True)
```

EDA & Elbow method

```
# =========================
# Elbow Method to find optimal K
# =========================
inertia = []
K_range = range(1, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

plt.figure(figsize=(8,4))
sns.lineplot(x=K_range, y=inertia, marker='o')
plt.title('Elbow Method - Optimal K')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.show()
```

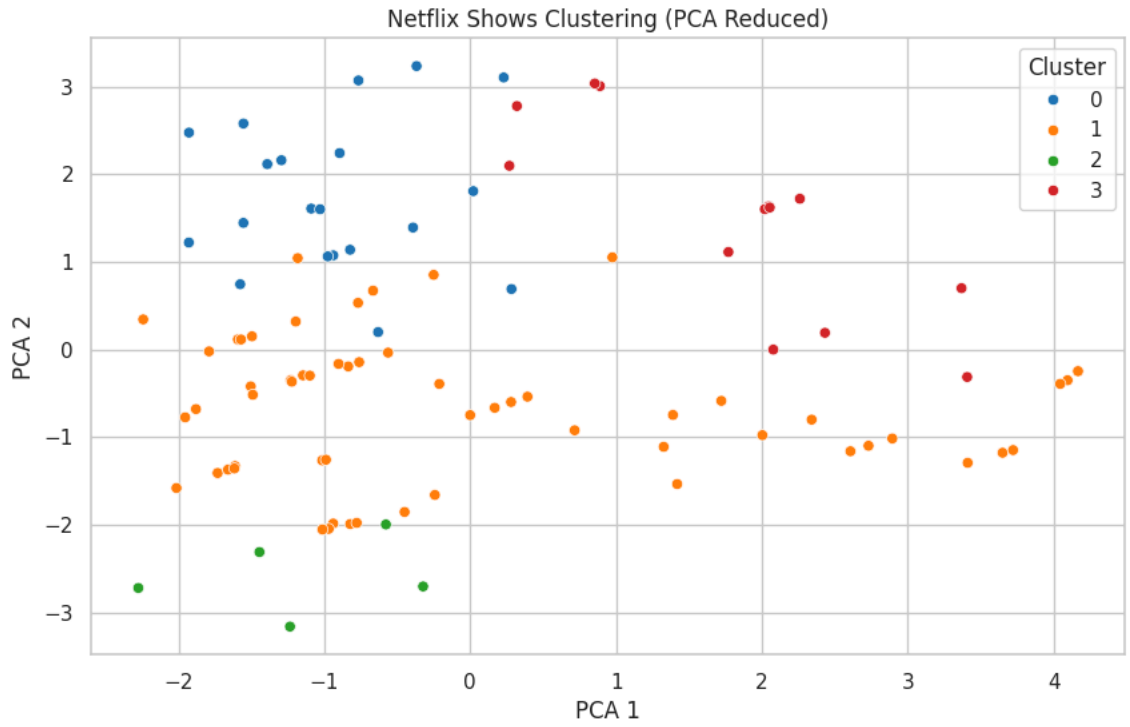


Clustering

```
# =========================
# K-Means
# =========================
kmeans = KMeans(n_clusters=4, random_state=42)
df['Cluster'] = kmeans.fit_predict(X_scaled)

# =========================
# Hierarchical Clustering
# =========================
agg = AgglomerativeClustering(n_clusters=4)
df['AggloCluster'] = agg.fit_predict(X_scaled)

# =========================
# DBSCAN
# =========================
dbscan = DBSCAN(eps=2, min_samples=5)
df['DBSCANCluster'] = dbscan.fit_predict(X_scaled)
```

PCA Visualization

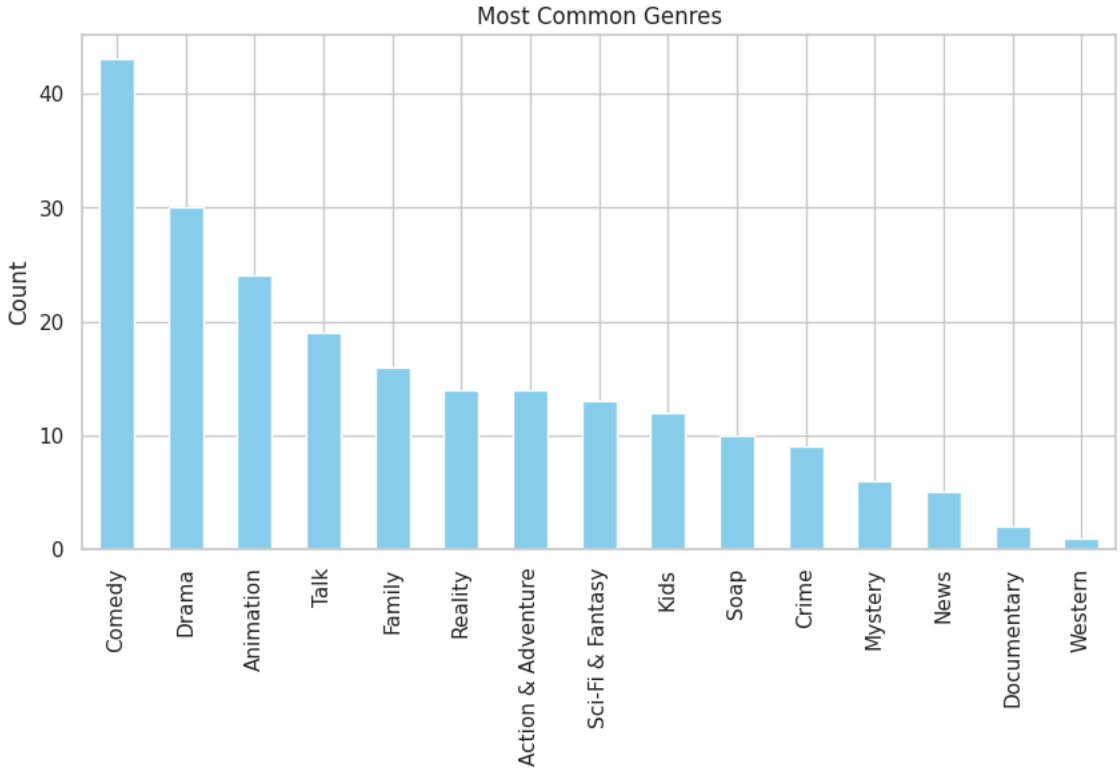
```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(10,6))
sns.scatterplot(x=X_pca[:,0], y=X_pca[:,1], hue=df['Cluster'], palette='tab10')
plt.title('Netflix Shows Clustering (PCA Reduced)')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.show()
```

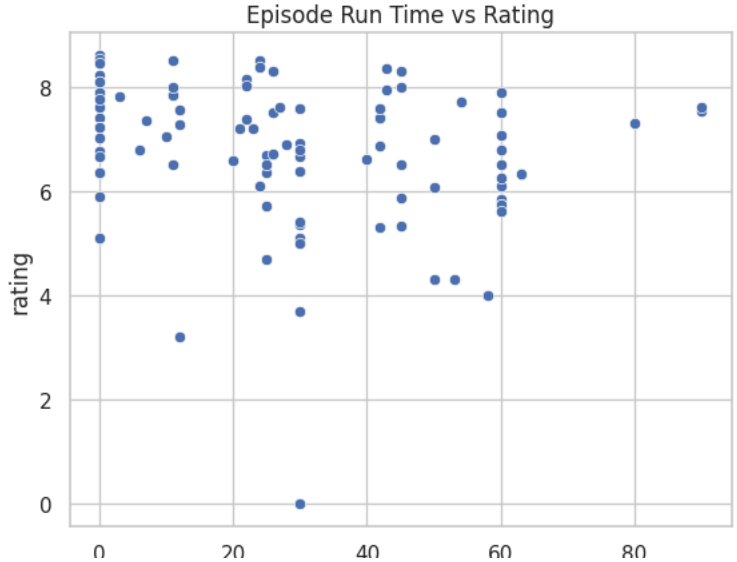


Cluster analysis & EDA plots

```
# ==========================
# 🔍 Cluster summaries
# ==========================
print(df.groupby('Cluster')[['rating', 'episode_run_time', 'popularity']].mean())

# ==========================
# 🔍 Dominant genres by cluster
# ==========================
print(df.groupby('Cluster')[list(all_genres)].mean())

# ==========================
# 🔍 Popular genres
# ==========================
all_genres_flat = list(itertools.chain(*df['genres']))
pd.Series(all_genres_flat).value_counts().plot(kind='bar', figsize=(10,5), color='skyblue')
plt.title("Most Common Genres")
plt.ylabel("Count")
plt.show()

# ==========================
# 🔍 Runtime vs Rating
# ==========================
sns.scatterplot(x='episode_run_time', y='rating', data=df)
plt.title("Episode Run Time vs Rating")
plt.show()

# ==========================
# 🔍 Correlation heatmap
# ==========================
corr_features = ['rating', 'episode_run_time', 'popularity'] + list(all_genres)
plt.figure(figsize=(14,10))
sns.heatmap(df[corr_features].corr(), cmap='coolwarm', annot=True)
plt.title("Feature Correlation Heatmap")
plt.show()
```

```
           rating  episode_run_time  popularity
Cluster
0        7.136571         31.428571  400.401948
1        6.357361         30.852459  311.084820
2        6.150200         28.600000  434.324080
3        7.996692         16.615385  247.956562
         Documentary     Drama   Reality  Animation      Talk  \
Cluster
0           0.000000  0.952381  0.047619   0.000000  0.000000
1           0.032787  0.081967  0.213115   0.245902  0.245902
2           0.000000  0.000000  0.000000   0.000000  0.800000
3           0.000000  0.384615  0.000000   0.692308  0.000000

         Action & Adventure     Crime  Sci-Fi & Fantasy  News   Mystery  \
Cluster
0                  0.238095  0.428571               0.0   0.0  0.095238
1                  0.049180  0.000000               0.0   0.0  0.032787
2                  0.000000  0.000000               0.0   1.0  0.000000
3                  0.461538  0.000000               1.0   0.0  0.153846

             Kids    Family   Western      Soap    Comedy
Cluster
0        0.000000  0.000000  0.047619  0.142857  0.142857
1        0.180328  0.245902  0.000000  0.114754  0.524590
2        0.000000  0.000000  0.000000  0.000000  0.600000
3        0.076923  0.076923  0.000000  0.000000  0.384615
```





episode_run_time