📊 Indian TV Serial Analysis with Python

Objective: Uncover insights, trends, and hidden patterns from Indian television serial data using Python.

Step 1: Import Libraries & Load the Dataset

In [1]:
```
import pandas as pd
import numpy as np
df=pd.read_csv('D:/OWN PROJECT/TV Serial Project/Indian television Serial Data
df.head()
```

Out[1]:

	Sr Number	State	years	Title	Network	Genre	Start date	End date	No. of episodes
0	1	Assam	10	Beharbari Outpost	Rengoni	Sitcom	07/10/2013	04/04/2025	2,914
1	2	Assam	7	Bharaghar	Rang	Sitcom	26/11/2012	16/11/2019	1,446
2	3	Assam	5	Borola Kai	Rang	Sitcom	23/03/2015	28/11/2020	1,612
3	4	Assam	4	Oi Khapla	Rang	Sitcom	20/07/2015	02/02/2019	1,094
4	5	Bangali	19	Police Files	Akash Aath	Crime show	16/05/2004	04/04/2025	6,000

Step 2: Clean Episode Column

In [3]:
```
df['No. of episodes'] = df['No. of episodes'].astype(str).str.replace(',', '')
df['No. of episodes'] = pd.to_numeric(df['No. of episodes'], errors='coerce')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 353 entries, 0 to 352
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Sr Number        353 non-null    int64 
 1   State            353 non-null    object
 2   years            353 non-null    int64 
 3   Title            353 non-null    object
 4   Network          353 non-null    object
 5   Genre            353 non-null    object
 6   Start date       353 non-null    object
 7   End date         353 non-null    object
 8   No. of episodes  353 non-null    int64 
dtypes: int64(3), object(6)
memory usage: 24.9+ KB
```

Step 3: Convert Dates & Calculate Duration

In [5]:
```
df['Start date'] = pd.to_datetime(df['Start date'], errors='coerce')
df['End date'] = pd.to_datetime(df['End date'], errors='coerce')

df['Years'] = ((df['End date'] - df['Start date']).dt.days // 365).fillna(0).asty
df['Episodes_per_Year'] = df['No. of episodes'] / df['Years'].replace(0, 1)
```

Step 4: Check for Missing & Duplicate Data

In [7]:
```
print(df.isnull().sum())
print(df.duplicated().sum())
```

```
Sr Number              0
State                  0
years                  0
Title                  0
Network                0
Genre                  0
Start date           191
End date             160
No. of episodes        0
Years                  0
Episodes_per_Year      0
dtype: int64
0
```

Step 5: Longest Running Serial

In [8]:
```
df.loc[df['years'].idxmax()][['Title', 'years', 'No. of episodes']]
```

Out[8]:
```
Title              Krishi Darshan[1]
years                             57
No. of episodes                16780
Name: 84, dtype: object
```

Step 6: Top Networks by Average Episodes

In [9]:
```
df.groupby('Network')['No. of episodes'].mean().sort_values(ascending=False).head
```

Out[9]:
```
Network
DD National DD Kisan    16780.0
Jaya TV                  8522.0
Asianet News             8003.0
DD National              6237.5
DD Chandana              4413.0
Name: No. of episodes, dtype: float64
```

Step 7: Most Common Genres

In [10]:
```
df['Genre'].value_counts().head(5)
```

Out[10]:
```
Genre
Soap opera      188
Sitcom           19
Reality show     11
Cooking show     10
Talk show         8
Name: count, dtype: int64
```

Step 8: State-wise Production

In [12]:
```
df['State'].value_counts().head(5)
```

Out[12]:
```
State
Tamil        78
Hindi        66
Telugu       61
Malayalam    40
Kannada      38
Name: count, dtype: int64
```

Step 9: Most Intense Shows (Episodes per Year)

In [13]:
```
df.sort_values(by='Episodes_per_Year', ascending=False).head(5)[['Title', 'Episod
```

Out[13]:

	Title	Episodes_per_Year
84	Krishi Darshan[1]	16780.0
85	Chitrahaar	12000.0
177	Home Minister	6225.0
4	Police Files	6000.0
27	Rasoi Show	6000.0

Step 10: High Episode Outliers

In [14]:
```
df[df['No. of episodes'] > df['No. of episodes'].quantile(0.95)][['Title', 'No. o
```

Out[14]:

	Title	No. of episodes
4	Police Files	6000
5	Rannaghor	5166
27	Rasoi Show	6000
36	Taarak Mehta Ka Ooltah Chashmah	4310
37	Yeh Rishta Kya Kehlata Hai	4474
84	Krishi Darshan[1]	16780
85	Chitrahaar	12000
86	Rangoli	11500
99	Thatt Antha Heli	4413
138	Munshi	8003
144	Sandhya Deepam	5868
151	Udayamritham	4641
177	Home Minister	6225
215	Arul Neeram	8522
220	Olimayamana Ethirkaalam	5335
292	Aaradhana	5015
293	Annadata	5100

Step 11: Pivot Table – Network vs Genre

In [15]:
```
df.pivot_table(index='Network', columns='Genre', values='No. of episodes', aggfur
```

Out[15]:

Network	Genre	Agriculture	Astrological Show	Astrological show	Astrology show	Autobiography	Comedy	Comedy Show	Come ga sh
Aakash Aath		0	0	0	0	0	0	0	
Akash Aath		0	0	0	0	0	0	0	
Alpha Bangla		0	0	0	0	0	0	0	
Amrita TV		0	0	0	0	0	0	0	
And TV		0	0	0	0	0	0	0	

5 rows × 70 columns

Step 12: Compare State-wise Episode Totals

In [16]:
```
df.groupby('State')['No. of episodes'].sum().sort_values(ascending=False).head(10
```

Out[16]:
```
State
Hindi        129163
Tamil        109558
Telugu       101926
Malayalam     61135
Kannada       58186
Marathi       45427
Bangali       45013
Odisha        17898
Gujrati       13119
Assam          7066
Name: No. of episodes, dtype: int64
```

Step 13: Longest Running Genres

In [17]:
```
df.groupby('Genre')['Years'].mean().sort_values(ascending=False).head(10)
```

Out[17]:
```
Genre
Astrological Show      40.000000
Music                  37.250000
Reality, Music         30.000000
Reality, music         28.000000
Astrology show         27.600000
Game show              25.285714
Travel show            24.000000
Quiz show              23.000000
Supernatural horror    19.000000
Reality, Dance         18.000000
Name: Years, dtype: float64
```

Step 14: Feature Scaling + Clustering

In [18]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

features = df[['No. of episodes', 'Years', 'Episodes_per_Year']].dropna()
scaled = StandardScaler().fit_transform(features)
kmeans = KMeans(n_clusters=3, random_state=42).fit(scaled)

df['Cluster'] = kmeans.labels_
df.groupby('Cluster')[['No. of episodes', 'Years', 'Episodes_per_Year']].mean()
```

```
C:\Users\PIXEL-FC93\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:141
2: FutureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\PIXEL-FC93\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:143
6: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the e
nvironment variable OMP_NUM_THREADS=2.
  warnings.warn(
```

Out[18]:

Cluster	No. of episodes	Years	Episodes_per_Year
0	1409.906907	2.162162	1058.992104
1	6643.066667	3.000000	5591.148000
2	3869.200000	123.000000	30.023950

Step 15: Rename Cluster Groups

In [19]:
```
def label_cluster(row):
    if row['Cluster'] == 1:
        return 'Mega Show (High Episodes & Intensity)'
    elif row['Cluster'] == 2:
        return 'Old Legacy Show (Long Duration)'
    else:
        return 'Short-Term or Seasonal Show'

df['Cluster_Label'] = df.apply(label_cluster, axis=1)
```

Step 16: Cluster Distribution & Share

In [21]:
```
df['Cluster_Label'].value_counts()
(df['Cluster_Label'].value_counts(normalize=True) * 100).round(2)
```

Out[21]:
```
Cluster_Label
Short-Term or Seasonal Show              94.33
Mega Show (High Episodes & Intensity)     4.25
Old Legacy Show (Long Duration)           1.42
Name: proportion, dtype: float64
```

Step 17: Top Genres per Cluster

In [22]:
```
df.groupby('Cluster_Label')['Genre'].value_counts().groupby(level=0).head(3)
```

Out[22]:
```
Cluster_Label                           Genre
Mega Show (High Episodes & Intensity)   Cooking show          2
                                        Agriculture           1
                                        Astrology show        1
Old Legacy Show (Long Duration)         Game show             1
                                        Astrological Show     1
                                        Astrology show        1
Short-Term or Seasonal Show             Soap opera          187
                                        Sitcom               18
                                        Reality show         10
Name: count, dtype: int64
```

Step 18: Network-Wise Cluster Strength

In [23]:
```
network_cluster = df.groupby('Network')['Cluster_Label'].value_counts().unstack()
network_cluster.sort_values(by='Mega Show (High Episodes & Intensity)', ascending
```

Out[23]:

Cluster_Label	Mega Show (High Episodes & Intensity)	Old Legacy Show (Long Duration)	Short-Term or Seasonal Show
Network			
ETV	3	0	20
Aakash Aath	1	0	0
DD National	1	1	2
Zee Marathi	1	0	6
Zee Bangla	1	1	8
Sony SAB	1	0	5
Akash Aath	1	0	1
Jaya TV	1	0	0
DD National DD Kisan	1	0	0
Zee Telugu	1	0	19

Step 19: Cluster-wise Averages

In [24]:
```
df.groupby('Cluster_Label')[['No. of episodes', 'Years', 'Episodes_per_Year']].me
```

Out[24]:

Cluster_Label	No. of episodes	Years	Episodes_per_Year
Mega Show (High Episodes & Intensity)	6643.1	3.0	5591.1
Old Legacy Show (Long Duration)	3869.2	123.0	30.0
Short-Term or Seasonal Show	1409.9	2.2	1059.0

Step 20: Export Cluster-wise CSVs

In [25]:
```
for label in df['Cluster_Label'].unique():
    df[df['Cluster_Label'] == label].to_csv(f"{label.replace(' ', '_')}.csv", inc
```

Conclusion

1. The project analyzed 353 Indian TV serials spanning from 1 Jan 1889 to 31 Jan 2025.
2. Data was cleaned and prepared by handling dates, missing values, and converting episode counts to numeric format.
3. The longest-running show is Krishi Darshan with 57 years and 16,780 episodes.
4. Most common genre is Soap opera, followed by Sitcom and Reality shows.
5. KMeans clustering grouped serials into 3 types:
    -Mega Shows (High Episodes & Intensity)
    -Old Legacy Shows (Long Duration)
    -Short-Term or Seasonal Shows
6. About 94% of shows are short-term, while only 1.4% are legacy shows.
7. Outlier detection revealed shows with exceptionally high episodes, like Chitrahaar and Home Minister.
8. The project concluded with exporting categorized data, building pivot insights, and performing cluster-based behavioral analysis.