# Project Name - Netflix Movies & TV shows Clustering - Unsupervised Learning


**Project Type** - Unsupervised

**Contribution** - Individual

## Project Summary -

The goal of this project is to analyze and group Netflix shows and movies using clustering techniques. The dataset includes information like title, genre, release year, duration, and rating. We want to find patterns and similarities among the content and organize them into meaningful groups.

First, we'll clean the data by handling missing values, removing unnecessary columns, and converting categorical information into numbers. We may also create new features to make the data more useful.

Next, we'll explore the data using visualizations and summaries to understand the distribution and relationships between variables.

After that, we'll apply clustering methods like k-means or hierarchical clustering to group similar shows and movies. We'll determine the right number of groups using methods like the elbow technique or silhouette analysis.

Once the clusters are formed, we'll evaluate and interpret them to find common traits within each group. This analysis will help Netflix with content categorization, recommendations, and planning future content.

Finally, we'll summarize the findings using visualizations and suggest recommendations for improving user experience and content offerings. This project will give Netflix a clearer view of its content and assist in decision-making.

## GitHub Link -

↳ *1 cell hidden*

## Problem Statement

**This dataset includes TV shows and movies available on Netflix as of 2019, collected from Flixable, a third-party Netflix search engine.**

**In 2018, Flixable released a report showing that since 2010, the number of TV shows on Netflix has nearly tripled, while the number of movies has dropped by over 2,000 titles. It will be interesting to explore what other insights can be discovered from this dataset.**

## *Let's Begin !*

## *1. Know Your Data*

### Import Libraries

```
# Import Libraries

# Importing the libraries
import numpy as np
import pandas as pd
from numpy import math
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
from matplotlib.pyplot import figure
import plotly.graph_objects as go
import plotly.offline as py
import plotly.express as px
from datetime import datetime

import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
```

```
from plotly.offline import init_notebook_mode, iplot
import plotly.offline as po
import plotly.io as pio

from collections import Counter
from sklearn import preprocessing
from sklearn.feature_extraction.text import TfidfVectorize
from sklearn.model_selection import train_test_split, KFol
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
from sklearn.decomposition import PCA

import warnings
warnings.filterwarnings('ignore')
```

```
<ipython-input-1-1392662c5a3a>:6: DeprecationWarning: `np.math` is a deprecated alias
  from numpy import math
```

```
pip install -U kaleido
```

```
Collecting kaleido
  Downloading kaleido-0.2.1-py2.py3-none-manylinux1_x86_64.whl.metadata (15 kB)
Downloading kaleido-0.2.1-py2.py3-none-manylinux1_x86_64.whl (79.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 79.9/79.9 MB 6.5 MB/s eta 0:00:00
Installing collected packages: kaleido
Successfully installed kaleido-0.2.1
```

## Dataset Loading

```
# Load Dataset

df = pd.read_csv('/content/NETFLIX MOVIES AND TV SHOWS CLUSTERING.csv')
```

## Dataset First View

```
# Dataset First Look
df.head(2)
```

| | show_id | type | title | director | cast | country | date_added | release_year | rat |
|---|---|---|---|---|---|---|---|---|---|
| **0** | s1 | TV Show | 3% | NaN | João Miguel, Bianca Comparato, Michel Gomes | Brazil | August 14, 2020 | 2020 | TV- |

## Dataset Rows & Columns count

```
1 df.columns
```

```
Index(['show_id', 'type', 'title', 'director', 'cast', 'country', 'date_added',
       'release_year', 'rating', 'duration', 'listed_in', 'description'],
      dtype='object')
```

```
1 df.shape
```

```
(7787, 12)
```

## Dataset Information

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7787 entries, 0 to 7786
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   show_id       7787 non-null   object
 1   type          7787 non-null   object
 2   title         7787 non-null   object
 3   director      5398 non-null   object
 4   cast          7069 non-null   object
 5   country       7280 non-null   object
 6   date_added    7777 non-null   object
 7   release_year  7787 non-null   int64 
 8   rating        7780 non-null   object
 9   duration      7787 non-null   object
 10  listed_in     7787 non-null   object
 11  description   7787 non-null   object
dtypes: int64(1), object(11)
memory usage: 730.2+ KB
```

## Duplicate Values

```
1 df.duplicated().sum()
```

```
0
```

## Missing Values/Null Values

```
1 # Missing Values/Null Values Count
2 df.isnull().sum()
```

| | 0 |
|---|---|
| **show_id** | 0 |
| **type** | 0 |
| **title** | 0 |
| **director** | 2389 |
| **cast** | 718 |
| **country** | 507 |
| **date_added** | 10 |
| **release_year** | 0 |
| **rating** | 7 |
| **duration** | 0 |
| **listed_in** | 0 |
| **description** | 0 |

**dtype:** int64

```
df['cast'].fillna(value='No cast',inplace=True)
df['country'].fillna(value=df['country'].mode()[0],inplace=True)
```

## What did you know about your dataset?

This dataset contain information about various TV shows and movies available on Netflix, including details like the production country, release year, rating, duration, genre, and a description of each title. It consists of 12 columns and 7787 rows.

## *2. Understanding Your Variables*

```
# Dataset Describe
df.describe(include='all')
```

| | show_id | type | title | director | cast | country | date_added | release_year | rati |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 7787 | 7787 | 7787 | 5398 | 7787 | 7787 | 7777 | 7787.000000 | 77 |
| **unique** | 7787 | 2 | 7787 | 4049 | 6832 | 681 | 1565 | NaN | |
| **top** | s1 | Movie | 3% | Raúl Campos, Jan Suter | No cast | United States | January 1, 2020 | NaN | TV-I |
| **freq** | 1 | 5377 | 1 | 18 | 718 | 3062 | 118 | NaN | 28 |
| **mean** | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 2013.932580 | N |
| **std** | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 8.757395 | N |
| **min** | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 1925.000000 | N |
| **25%** | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 2013.000000 | N |
| **50%** | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 2017.000000 | N |

## Variables Description

show_id : Unique ID for every Movie / Tv Show

type : Identifier - A Movie or TV Show

title : Title of the Movie / Tv Show

director : Director of the Movie

cast : Actors involved in the movie / show

country : Country where the movie / show was produced

date_added : Date it was added on Netflix

release_year : Actual Releaseyear of the movie / show

rating : TV Rating of the movie / show

duration : Total Duration - in minutes or number of seasons

listed_in : Genere

description: The Summary descriptionAnswer Here

## Check Unique Values for each variable.

```
# Check Unique Values for each variable.
print(df.apply(lambda col: col.unique()))
```

```
show_id         [s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11,...
type                                          [TV Show, Movie]
title           [3%, 7:19, 23:59, 9, 21, 46, 122, 187, 706, 19...
director        [nan, Jorge Michel Grau, Gilbert Chan, Shane A...
cast            [João Miguel, Bianca Comparato, Michel Gomes, ...
country         [Brazil, Mexico, Singapore, United States, Tur...
date_added      [August 14, 2020, December 23, 2016, December ...
release_year    [2020, 2016, 2011, 2009, 2008, 2019, 1997, 201...
rating          [TV-MA, R, PG-13, TV-14, TV-PG, NR, TV-G, TV-Y...
duration        [4 Seasons, 93 min, 78 min, 80 min, 123 min, 1...
listed_in       [International TV Shows, TV Dramas, TV Sci-Fi ...
description     [In a future where the elite inhabit an island...
dtype: object
```

## 3. *Data Wrangling*

### Data Wrangling Code

```
# Create new features to store date, day, month and year seperately.
df["date_added"] = pd.to_datetime(df['date_added'],format='mixed')
df['day_added'] = df['date_added'].dt.day             # Compute day.
df['year_added'] = df['date_added'].dt.year           # Compute year.
df['month_added'] = df['date_added'].dt.month         # Compute month.
```

### What all manipulations have you done and insights you found?

Director: There are missing values in the "Director" column.

Country: There are missing values in the "Country" column, which have been filled with zero.

Cast: There are missing values in the "Cast" column, which have been filled with "No cast."

Date Added: There are missing values in the "Date Added" column.

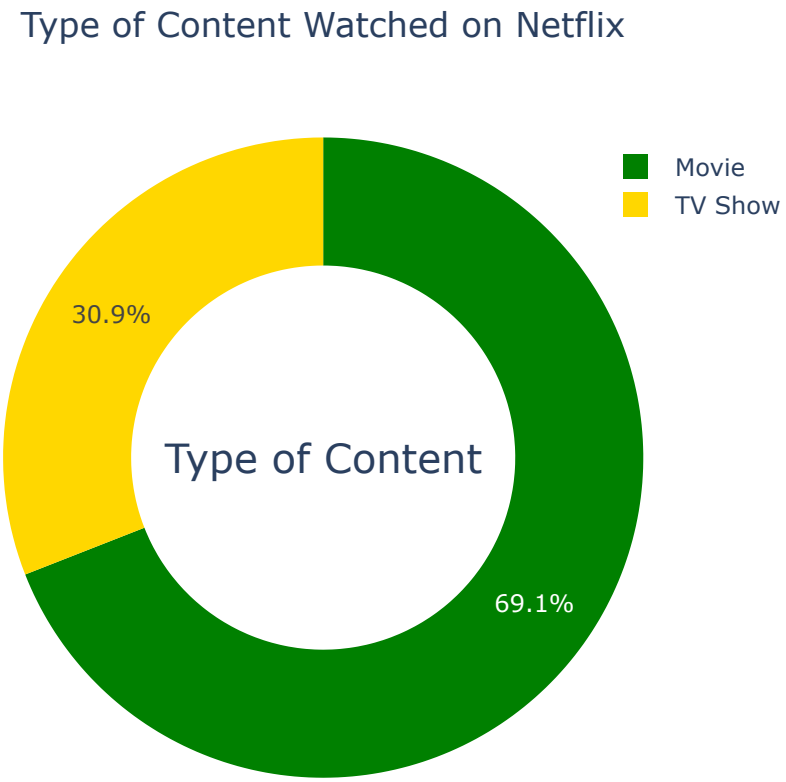
Duplicated entries have been identified in the dataset,sum is zero.Unique Values also in each column has to find unique items from different columns.

Date_addded Column: In the "Date Added" column, additional information has been extracted such as the day, month, and year.Answer Here.

## *4. Data Vizualization, Storytelling & Experimenting with charts : Understand the relationships between variables*

### Chart - 1

```
# Chart - 1 visualization code
labels = ['TV Show', 'Movie']
values = [df.type.value_counts()[1], df.type.value_counts()[0]]

# Colors
colors = ['#ffd700', '#008000']

# Create pie chart
fig = go.Figure(data=[go.Pie(labels=labels, values=values, hole=.6)])

# Customize layout
fig.update_layout(
    title_text='Type of Content Watched on Netflix',
    title_x=0.5,
    height=500,
    width=500,
    legend=dict(x=0.9),
    annotations=[dict(text='Type of Content', font_size=20, showarrow=False)]
)

# Set colors
fig.update_traces(marker=dict(colors=colors))

```

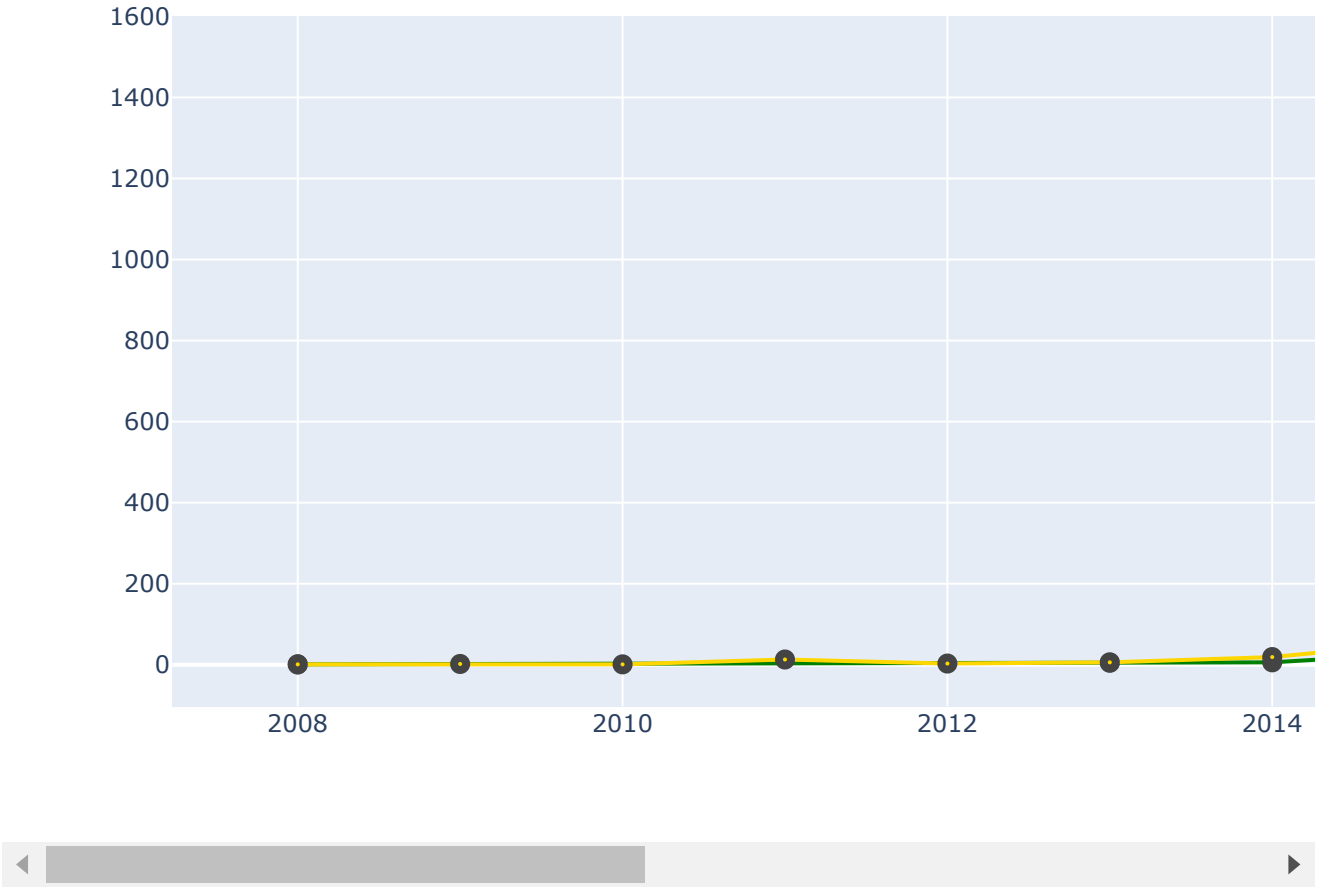


1. Why did you pick the specific chart?

The specific chart used in the code is a pie chart. I picked this chart because it is effective in visualizing the distribution of categorical data. In this case, the chart is used to represent the types of content watched on Netflix, which are categorized as "TV Show" and "Movie."Answer Here.

## Chart - 2

```
import plotly.graph_objects as go
import pandas as pd
tv_show = df[df["type"] == "TV Show"]
movie = df[df["type"] == "Movie"]

col = "year_added"

content_1 = tv_show["year_added"].value_counts().sort_index()
content_2 = movie["year_added"].value_counts().sort_index()

trace1 = go.Scatter(x=content_1.index, y=content_1.values, name="TV Shows", marker=dic
trace2 = go.Scatter(x=content_2.index, y=content_2.values, name="Movies", marker=dict(

fig = go.Figure(data=[trace1, trace2], layout=go.Layout(title="Content added over the
# Display chart
fig.show()
```




## 1. Why did you pick the specific chart?

The line chart is suitable for showing the trend and distribution of data over a continuous axis (in this case, the years). It allows for easy comparison between the two categories (TV shows and movies) and how their counts vary over time.Answer Here.

## Chart - 3

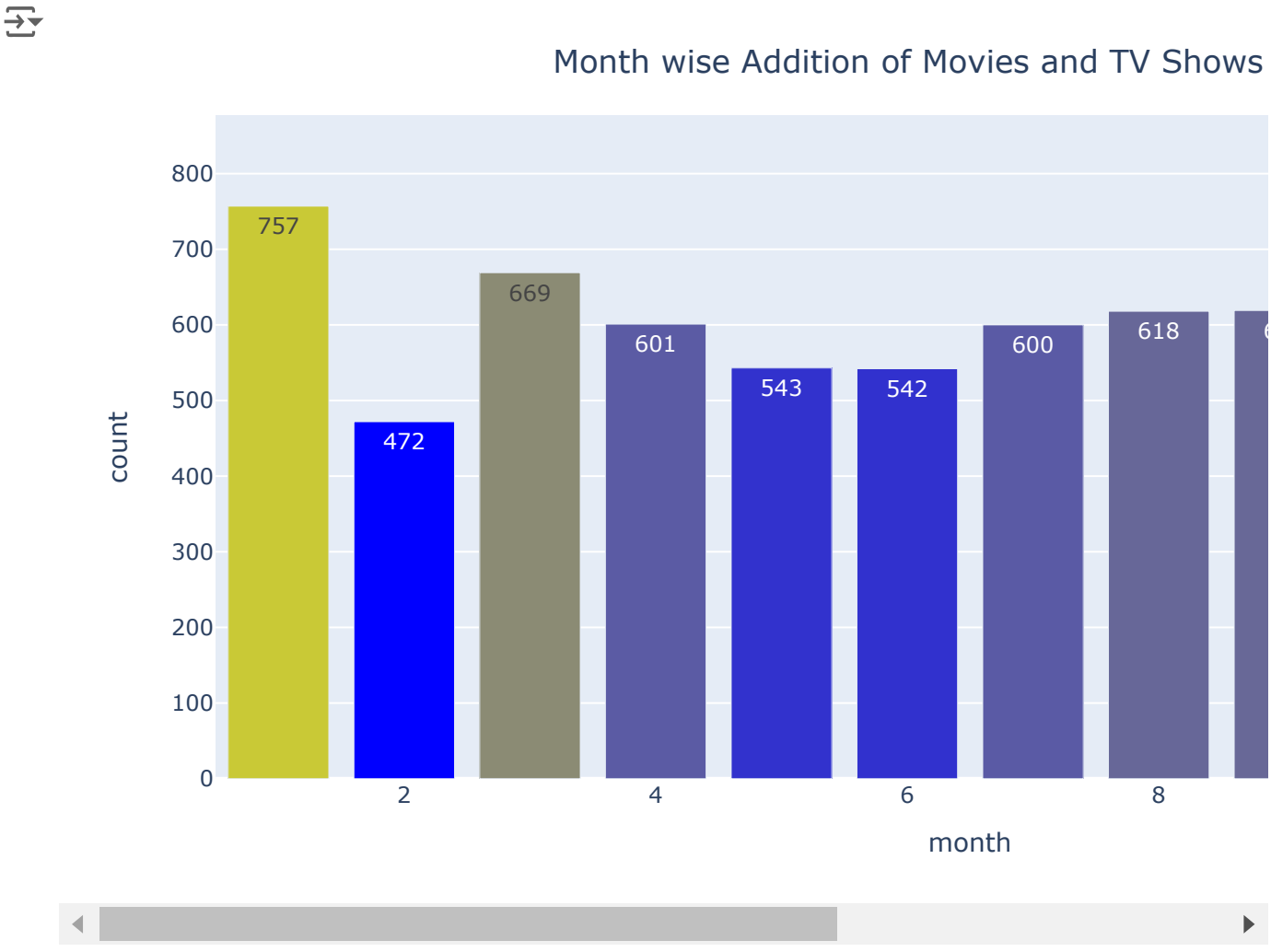
```
months_df = df['month_added'].value_counts().reset_index()

# Rename the columns to "month" and "count"
months_df.columns = ['month', 'count']

```

```
print(months_df)
```

```
   month  count
0   12.0    833
1   10.0    785
2    1.0    757
```

```
3    11.0    738
4     3.0    669
5     9.0    619
6     8.0    618
7     4.0    601
8     7.0    600
9     5.0    543
10    6.0    542
11    2.0    472
```

```
fig = px.bar(months_df, x="month", y="count", text_auto=True, color='count', color_con
fig.update_layout(
    title={
        'text': 'Month wise Addition of Movies and TV Shows on Netflix',
        'y':0.95,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'},
        autosize=False,
        width=1000,
        height=500,
        showlegend=True)
# fig.show()
fig.show()
```



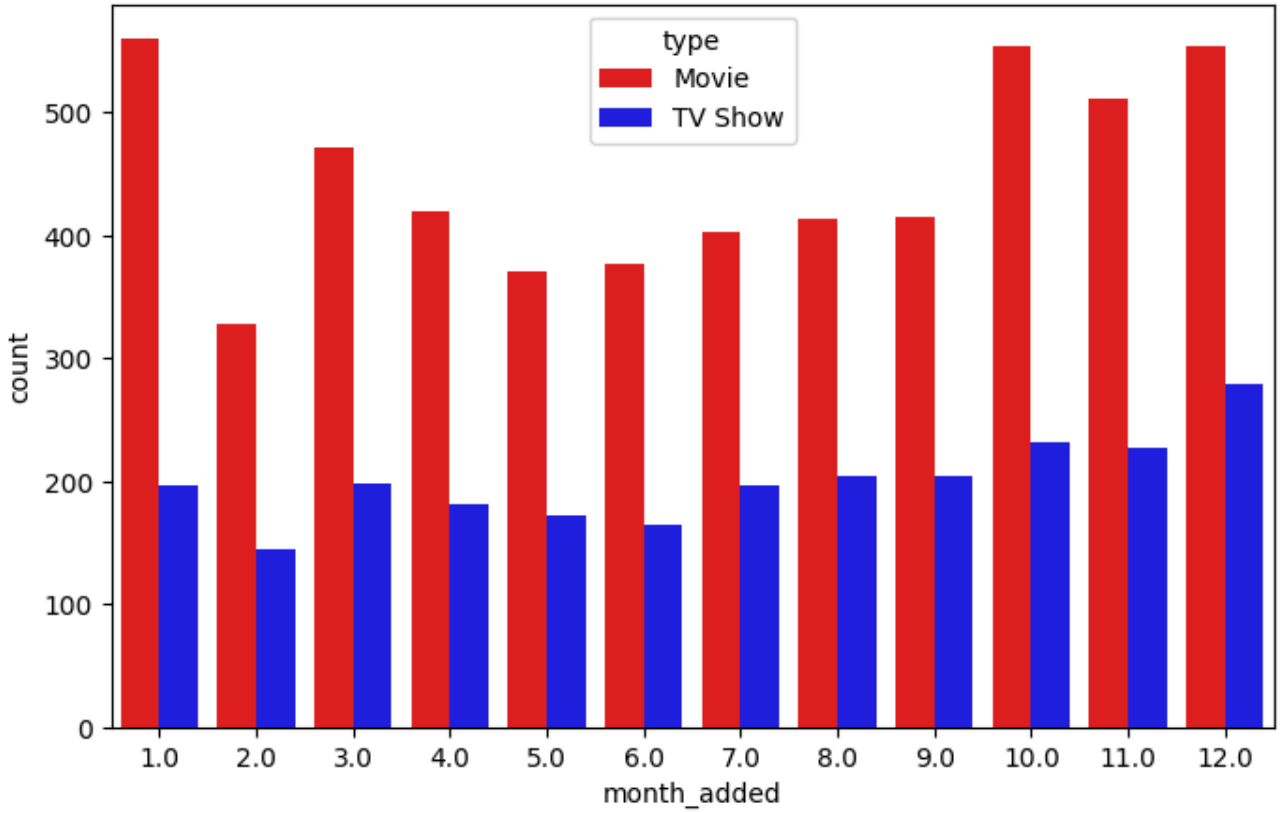
1. Why did you pick the specific chart?

The bar chart is suitable for comparing and displaying categorical data (months) and their corresponding counts. The chart helps in understanding the distribution of content additions across different months and identifying any patterns or trends.Answer Here.

## Chart - 4 Count Plot

```
# Chart - 4 visualization code
import matplotlib.pyplot as plt
import seaborn as sns
fig, ax = plt.subplots(figsize=(8,5))
sns.countplot(x='month_added', hue='type',lw=5, data=df, ax=ax,palette=['#FF0000' ,'#0
```

```
<Axes: xlabel='month_added', ylabel='count'>
```

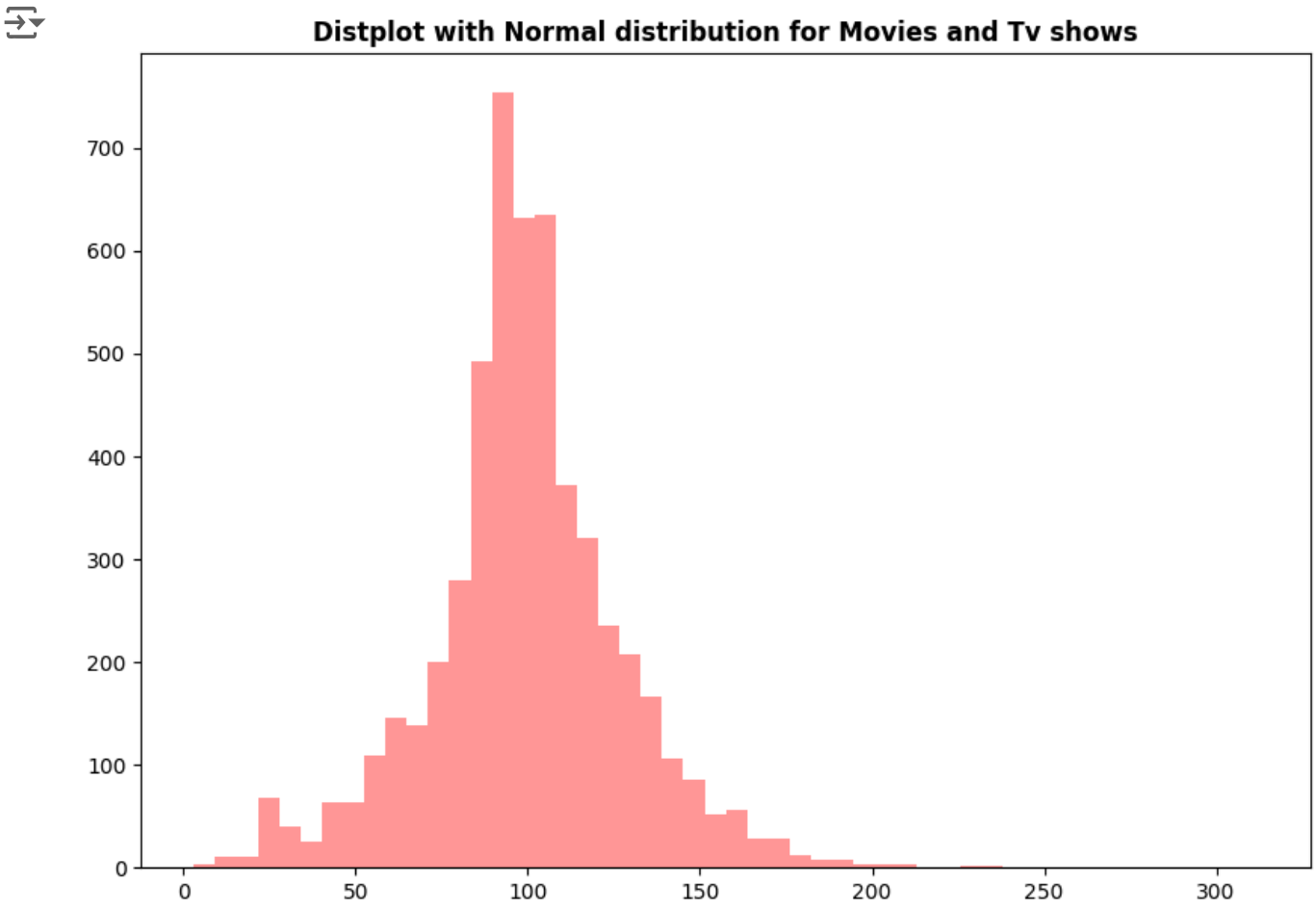



1. Why did you pick the specific chart?

By using a countplot, we can easily see and compare the frequencies of TV show and movie additions for each month.Answer Here.

## Chart - 5 - Distribution plot

```
# Chart - 5 visualization code
#Checking the distribution of Movie Durations
plt.figure(figsize=(10,7))
#Regular Expression pattern \d is a regex pattern for digit + is a regex pattern for a
sns.distplot(movie['duration'].str.extract('(\d+)'),kde=False, color=['red'])
plt.title('Distplot with Normal distribution for Movies and Tv shows',fontweight="bold
plt.show()
```




### 1. Why did you pick the specific chart?

The Distplot is a suitable choice for this analysis because it allows us to observe the frequency or count of movies falling into different duration ranges.Answer Here.

## Chart - 6 H-Bargraph

```
df['cast']
```

| | cast |
|---|---|
| **0** | João Miguel, Bianca Comparato, Michel Gomes, R... |
| **1** | Demián Bichir, Héctor Bonilla, Oscar Serrano, ... |
| **2** | Tedd Chan, Stella Chung, Henley Hii, Lawrence ... |
| **3** | Elijah Wood, John C. Reilly, Jennifer Connelly... |
| **4** | Jim Sturgess, Kevin Spacey, Kate Bosworth, Aar... |
| **...** | ... |
| **7782** | Imad Creidi, Antoinette Turk, Elias Gergi, Car... |
| **7783** | Vicky Kaushal, Sarah-Jane Dias, Raaghav Chanan... |
| **7784** | Nasty C |
| **7785** | Adriano Zumbo, Rachel Khoo |
| **7786** | No cast |

7787 rows × 1 columns

**dtype:** object

```
# seperating actors from cast column
cast = df['cast'].str.split(', ', expand=True).stack()

# top actors name who play highest role in movie/show.
cast.value_counts()

```

| | count |
|---|---|
| **No cast** | 718 |
| **Anupam Kher** | 42 |
| **Shah Rukh Khan** | 35 |
| **Om Puri** | 30 |
| **Naseeruddin Shah** | 30 |
| **...** | ... |
| **Archie Alemania** | 1 |
| **Demore Barnes** | 1 |
| **Marty Adams** | 1 |
| **Nicole Boivin** | 1 |
| **Rachel Khoo** | 1 |

32882 rows × 1 columns

**dtype:** int64

```
cast =cast[cast != 'No cast']
```

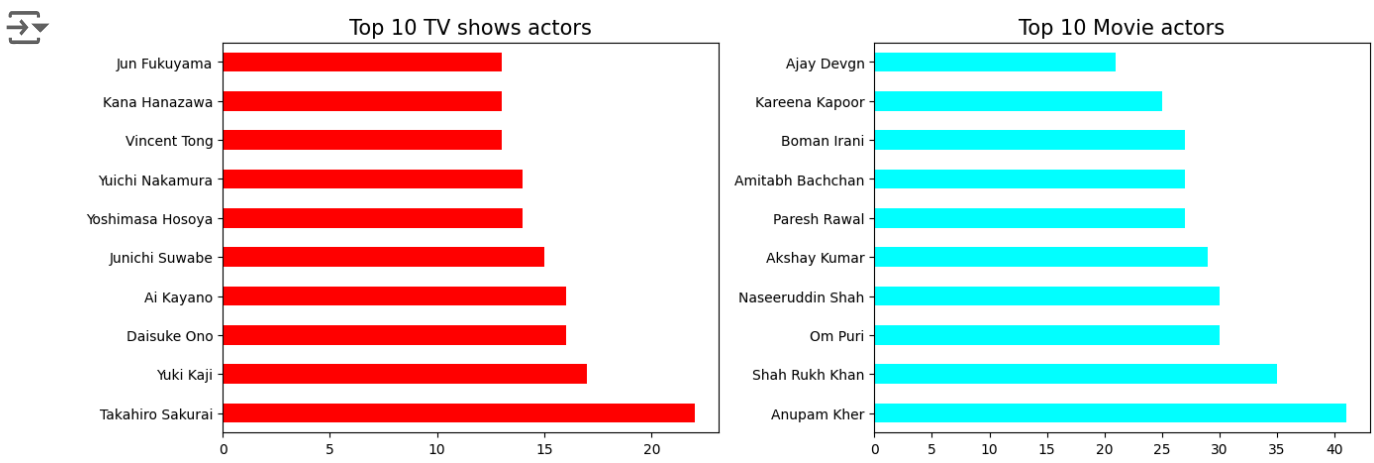
```
cast.value_counts()
```

| | count |
|---|---|
| **Anupam Kher** | 42 |
| **Shah Rukh Khan** | 35 |
| **Naseeruddin Shah** | 30 |
| **Om Puri** | 30 |
| **Akshay Kumar** | 29 |
| **...** | ... |
| **Archie Alemania** | 1 |
| **Demore Barnes** | 1 |
| **Marty Adams** | 1 |
| **Nicole Boivin** | 1 |
| **Rachel Khoo** | 1 |

32881 rows × 1 columns

**dtype:** int64

```
# Chart - 7 visualization code
fig,ax = plt.subplots(1,2, figsize=(14,5))

# seperating TV shows actor from cast column
top_TVshows_actor = df[df['type']=='TV Show']['cast'].str.split(', ', expand=True).sta
top_TVshows_actor =top_TVshows_actor[top_TVshows_actor != 'No cast']
# plotting actor who appeared in highest number of TV Show
a = top_TVshows_actor.value_counts().head(10).plot(kind='barh', ax=ax[0],color='red')
a.set_title('Top 10 TV shows actors', size=15)

# seperating movie actor from cast column
top_movie_actor = df[df['type']=='Movie']['cast'].str.split(', ', expand=True).stack()
top_movie_actor =top_movie_actor[top_movie_actor != 'No cast']
# plotting actor who appeared in highest number of Movie
b = top_movie_actor.value_counts().head(10).plot(kind='barh', ax=ax[1],color='Cyan')
b.set_title('Top 10 Movie actors', size=15)

plt.tight_layout(pad=1.2, rect=[0, 0, 0.95, 0.95])
plt.show()

```




## 1. Why did you pick the specific chart?

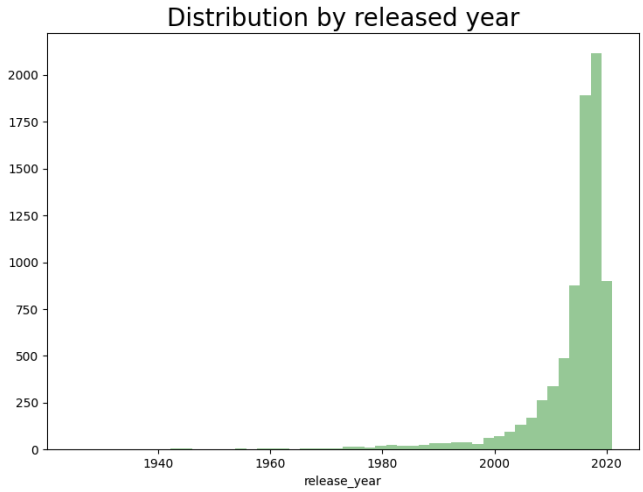
The horizontal orientation of the bars allows for easier reading and comparison of the values. The length of each bar represents the number of TV shows or movies an actor has appeared in. The chart also includes titles and is divided into two subplots, making it clear that one subplot represents TV shows and the other represents movies.Answer Here.

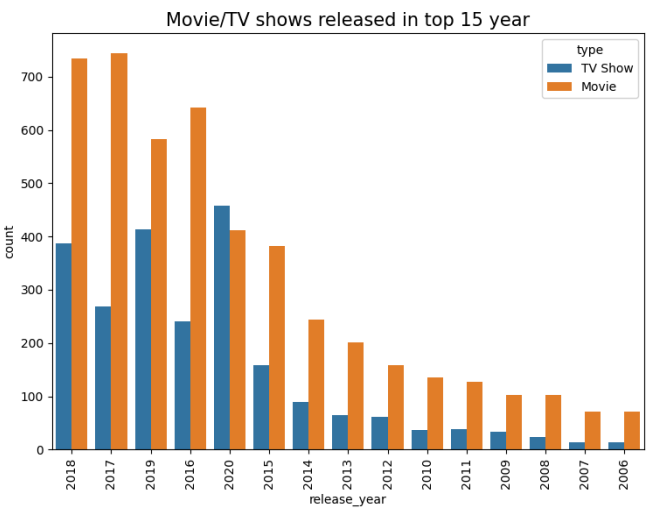
## Chart - 7 - Histo and Bar Graph

```
df['release_year'].nunique()
print(f'Oldest release year : {df.release_year.min()}')
print(f'Latest release year : {df.release_year.max()}')
```

```
Oldest release year : 1925
Latest release year : 2021
```

```
fig,ax = plt.subplots(1,2, figsize=(15,6))

# Univariate analysis
hist = sns.distplot(df['release_year'], ax=ax[0], kde=False,color='green')
hist.set_title('Distribution by released year', size=20)

# Bivariate analysis
count = sns.countplot(x="release_year", hue='type', data=df, order=df['release_year'].
count.set_title('Movie/TV shows released in top 15 year', size=15)
plt.xticks(rotation=90)

plt.tight_layout()
plt.show()

```




## Chart - 8 - Correlation Heatmap

```
ratings = {
    'TV-PG': 'Older Kids',
    'TV-MA': 'Adults',
    'TV-Y7-FV': 'Older Kids',
    'TV-Y7': 'Older Kids',
    'TV-14': 'Teens',
    'R': 'Adults',
    'TV-Y': 'Kids',
    'NR': 'Adults',
    'PG-13': 'Teens',
    'TV-G': 'Kids',
    'PG': 'Older Kids',
    'G': 'Kids',
    'UR': 'Adults',
    'NC-17': 'Adults'
}
df['target_ages'] = df['rating'].replace(ratings)
```

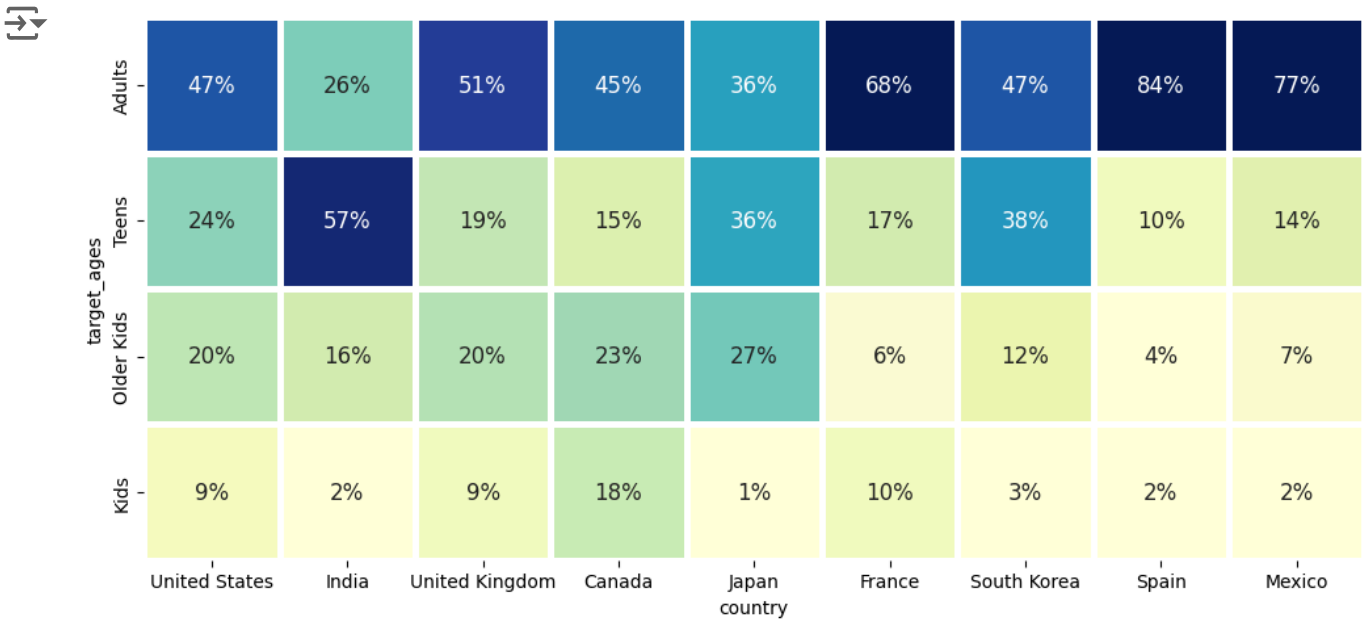
```
df.head()
```

| | show_id | type | title | director | cast | country | date_added | release_year | ra |
|---|---|---|---|---|---|---|---|---|---|
| **0** | s1 | TV Show | 3% | NaN | João Miguel, Bianca Comparato, Michel Gomes, R... | Brazil | 2020-08-14 | 2020 | T |
| **1** | s2 | Movie | 7:19 | Jorge Michel Grau | Demián Bichir, Héctor Bonilla, Oscar Serrano, ... | Mexico | 2016-12-23 | 2016 | T |
| **2** | s3 | Movie | 23:59 | Gilbert Chan | Tedd Chan, Stella Chung, Henley Hii, Lawrence ... | Singapore | 2018-12-20 | 2011 | |
| **3** | s4 | Movie | 9 | Shane Acker | Elijah Wood, John C. Reilly, Jennifer Connelly... | United States | 2017-11-16 | 2009 | P |
| **4** | s5 | Movie | 21 | Robert Luketic | Jim Sturgess, Kevin Spacey, Kate Bosworth, Aar | United States | 2020-01-01 | 2008 | P |

```
# Preparing data for heatmap
df['count'] = 1

# Group by country and sum the count, no need to include 'country' inside the groupby
data = df.groupby('country')['count'].sum().sort_values(ascending=False).reset_index()

# Extracting the top 10 countries
data = data['country']

# Filter the main dataframe for the top 10 countries
df_heatmap = df[df['country'].isin(data)]

# Create a crosstab for the heatmap data
df_heatmap = pd.crosstab(df_heatmap['country'], df_heatmap['target_ages'], normalize="

df_heatmap
```

| country | Canada | Egypt | France | India | Japan | Mexico | South Korea | Spai |
|---|---|---|---|---|---|---|---|---|
| target_ages | | | | | | | | |
| **Adults** | 0.446328 | 0.277228 | 0.678261 | 0.255688 | 0.364444 | 0.77 | 0.469945 | 0.83582 |
| **Kids** | 0.180791 | 0.000000 | 0.095652 | 0.016251 | 0.008889 | 0.02 | 0.027322 | 0.02238 |
| **Older Kids** | 0.225989 | 0.039604 | 0.060870 | 0.160347 | 0.271111 | 0.07 | 0.120219 | 0.04477 |

```
fig, ax = plt.subplots(1, 1, figsize=(12, 12))

country_order2 = ['United States', 'India', 'United Kingdom', 'Canada', 'Japan', 'Fran
age_order = ['Adults', 'Teens', 'Older Kids', 'Kids']

sns.heatmap(data=df_heatmap.loc[age_order, country_order2],
            cmap='YlGnBu',
            square=True,
            linewidth=2.5,
            cbar=False,
            annot=True,
            fmt='1.0%',
            vmax=.6,
            vmin=0.05,
            ax=ax,
            annot_kws={"fontsize": 12})
plt.show()
```




1. Why did you pick the specific chart?

A heatmap is a suitable choice when visualizing the relationships between two categorical variables, in this case, countries and age groups. It allows for a clear representation of patterns, trends, and comparisons across different categories.Answer Here.

## *5. Hypothesis Testing*

```
#making copy of df_clean_frame
netflix_hypothesis=df.copy()
#head of df_hypothesis
netflix_hypothesis.head()
```

| | show_id | type | title | director | cast | country | date_added | release_year | ra |
|---|---|---|---|---|---|---|---|---|---|
| **0** | s1 | TV Show | 3% | NaN | João Miguel, Bianca Comparato, Michel Gomes, R... | Brazil | 2020-08-14 | 2020 | T |
| **1** | s2 | Movie | 7:19 | Jorge Michel Grau | Demián Bichir, Héctor Bonilla, Oscar Serrano, ... | Mexico | 2016-12-23 | 2016 | T |
| **2** | s3 | Movie | 23:59 | Gilbert Chan | Tedd Chan, Stella Chung, Henley Hii, Lawrence ... | Singapore | 2018-12-20 | 2011 | |
| **3** | s4 | Movie | 9 | Shane Acker | Elijah Wood, John C. Reilly, Jennifer Connelly... | United States | 2017-11-16 | 2009 | P |
| **4** | s5 | Movie | 21 | Robert Luketic | Jim Sturgess, Kevin Spacey, Kate Bosworth, Aar... | United States | 2020-01-01 | 2008 | P |

```
netflix_hypothesis['duration']= netflix_hypothesis['duration'].str.extract('(\d+)')
netflix_hypothesis['duration'] = pd.to_numeric(netflix_hypothesis['duration'])
```

```
netflix_hypothesis['type'] = pd.Categorical(netflix_hypothesis['type'], categories=['M
#from duration feature extractin string part and after extracting Changing the object
#df_hypothesis['duration']= df_hypothesis['duration'].str.extract('(\d+)')
#df_hypothesis['duration'] = pd.to_numeric(df_hypothesis['duration'])
#head of df_
netflix_hypothesis.head(3)
```

| | show_id | type | title | director | cast | country | date_added | release_year | ra |
|---|---|---|---|---|---|---|---|---|---|
| **0** | s1 | TV Show | 3% | NaN | João Miguel, Bianca Comparato, Michel Gomes, R... | Brazil | 2020-08-14 | 2020 | T |
| **1** | s2 | Movie | 7:19 | Jorge Michel Grau | Demián Bichir, Héctor Bonilla, Oscar Serrano, ... | Mexico | 2016-12-23 | 2016 | T |
| **2** | s3 | Movie | 23:59 | Gilbert Chan | Tedd Chan, Stella Chung, Henley Hii, Lawrence ... | Singapore | 2018-12-20 | 2011 | |

```
netflix_hypothesis['type'] = pd.Categorical(netflix_hypothesis['type'], categories=['M
```

```
# Perform Statistical Test to obtain P-Value
#group_by duration and TYPE
group_by_= netflix_hypothesis[['duration','type']].groupby(by='type')
#mean of group_by variable
group1=group_by_.mean().reset_index()
group1
```

| | type | duration |
|---|---|---|
| **0** | Movie | 99.307978 |
| **1** | TV Show | 1.775934 |

```
netflix_hypothesis.groupby('type')['duration'].count()
```

| type | duration |
|---|---|
| **Movie** | 5377 |
| **TV Show** | 2410 |

**dtype:** int64

```
#In A and B variable grouping values
A= group_by_.get_group('Movie')
B= group_by_.get_group('TV Show')
#mean and std
M1 = A.select_dtypes(include='number').mean()
S1 = A.select_dtypes(include='number').std()

M2 = B.select_dtypes(include='number').mean()
S2 = B.select_dtypes(include='number').std()

print('Mean  {}'.format(M1,M2))
print('Std  {}'.format(S2,S1))
```

```
Mean  duration    99.307978
dtype: float64
Std  duration    1.596359
dtype: float64
```

```
#import stats
from scipy import stats
#length of groups and DOF
n1 = len(A)
n2= len(B)
print(n1,n2)

dof = n1+n2-2
print('dof',dof)

sp_2 = ((n2-1)*S1**2  + (n1-1)*S2**2) / dof
print('SP_2 =',sp_2)

sp = np.sqrt(sp_2)
print('SP',sp)

#tvalue
t_val = (M1-M2)/(sp * np.sqrt(1/n1 + 1/n2))
print('tvalue',t_val[0])
```

```
5377 2410
dof 7785
SP_2 = duration    253.64841
dtype: float64
SP duration    15.926343
dtype: float64
tvalue 249.81856492927665
```

Which statistical test have you done to obtain P-Value?

t-distribution

```
#t-distribution
stats.t.ppf(0.025,dof)
```

```
-1.9602687544602204
```

```
#t-distribution
stats.t.ppf(0.975,dof)
```

```
1.96026875446022
```

**Answer Here.**

Because the t-value is not in the range, the null hypothesis is rejected.

**As a result, The duration which is more than 90 mins are movies**

## *6. Feature Engineering & Data Pre-processing*

```
df2 = df.copy()
```

```
# Combining all the clustering attributes into a single column
df['clustering'] = (df['director'] + ' ' + df['cast'] +' ' +
                                df['country'] +' ' + df['listed_in'] +
                                ' ' + df['description'])

```

```
df['clustering'][25]
```

```
'Lyric R. Cabral, David Felix Sutcliffe No cast United States Documentaries This rea
l-life look at FBI counterterrorism operations features access to both sides of a st
ing: the government informant and the radicalized target.'
```

### Textual Data Preprocessing

```
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import string

nltk.download('all', quiet=True)

def transform_text(text):
    # Check if the input is a string; if not, return an empty string
    if not isinstance(text, str):
        return ''

    # Convert text to lowercase
    text = text.lower()

    # Remove URLs
```

```
    text = re.sub(r'http\S+', '', text)

    # Tokenize text into words
    words = nltk.word_tokenize(text)

    # Remove non-alphanumeric characters
    words = [word for word in words if word.isalnum()]

    # Remove stopwords and punctuation
    stopwords_set = set(stopwords.words('english'))
    words = [word for word in words if word not in stopwords_set]

    # Lemmatize words
    lemmatizer = WordNetLemmatizer()
    lemmatized_words = [lemmatizer.lemmatize(word) for word in words

    # Join words into a string and return
    return ' '.join(lemmatized_words)

# Apply the function to the 'clustering' column
df['Clean_Text'] = df['clustering'].apply(transform_text)

```

```
df["Clean_Text"][50]
```

```
'roland emmerich steven strait camilla belle cliff curtis joel virgel affif ben badr
a mo zinal nathanael baring mona hammond omar sharif united state south africa actio
n adventure fierce mammoth hunter set impossible journey rescue woman love vicious w
```

## Text Vectorization

**TF-IDF combines two metrics: Term frequency (TF) and inverse document frequency (IDF).**

Term Frequency (TF): This metric measures the frequency of a term in a document. It assumes that the more often a term appears in a document, the more relevant it is to that document. It is calculated using the formula:

**TF(t, d) = (Number of times term t appears in document d) / (Total number of terms in document d)**

**Inverse Document Frequency (IDF): This metric measures the importance of a term across a collection of documents. It gives higher weight to terms that appear less frequently in the entire collection. It is calculated using the formula:**

**IDF(t) = log_e(Total number of documents / Number of documents containing term t)**

```
bag_of_words = df.Clean_Text
```

```
t_vectorizer = TfidfVectorizer(max_features=20000)
X= t_vectorizer.fit_transform(bag_of_words)
```

```
print(X.shape)
```

```
(7787, 20000)
```

```
t_vectorizer.get_feature_names_out()
```

```
array(['007', '10', '100', ..., 'şimşek', 'şinasi', 'şükran'],
      dtype=object)
```

Which text vectorization technique have you used and why?

Answer Here.

## Dimesionality Reduction

**Do you think that dimensionality reduction is needed? Explain Why**

**Answer Here.**

PCA to reduce the dimensionality of the dataset. PCA identifies the directions (principal components) along which the data varies the most. These components are ordered by the amount of variance they explain in the data.

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

transformer = TruncatedSVD(n_components=4800)  # Adjust n_components as needed
transformer.fit(X)  # No need to convert sparse matrix to dense
```

```
TruncatedSVD
TruncatedSVD(n_components=4800)
```

```
X_sample = X[:5000:12]  # Select a subset of 1000 samples
transformer.fit(X_sample)
```

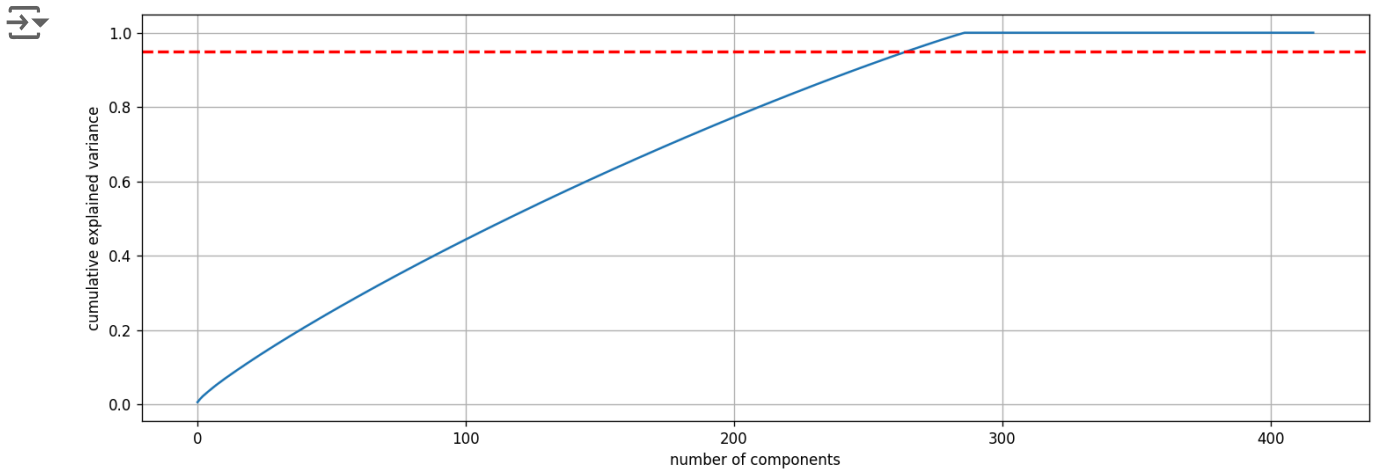
```
TruncatedSVD
TruncatedSVD(n_components=4800)
```

**Which dimensionality reduction technique have you used and why? (If dimensionality reduction done on dataset.)**

**Answer Here**

PCA can extract the most relevant features from a dataset. It transforms the original features into a new set of uncorrelated variables called principal components. These components are linear combinations of the original features and capture the maximum amount of variation present in the data.

```
# Lets plot explained var v/s comp to check how many components to be considered.
 #explained var v/s comp
# Add a grid to the plot
import matplotlib.pyplot as plt
plt.figure(figsize=(15,5), dpi=120)
plt.plot(np.cumsum(transformer.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axhline(y=0.95, color='r', linestyle='--',linewidth=2, label='95% Explained Varian
plt.grid()
plt.show()
```




**The plot helps in determining the number of components to consider for dimensionality reduction. You can select the number of components where the cumulative explained variance reaches a satisfactory threshold, such as 95%. The point where the curve intersects or is closest to the threshold line can guide you in choosing the appropriate number of components for your analysis.**

```
# Import the necessary libraries
from sklearn.decomposition import PCA
# Create an instance of PCA with the desired explained variance ratio
pca_tuned = PCA(n_components=0.65)
# Fit the PCA model on the input data, X, which is converted to a dense array
pca_tuned.fit(X.toarray())
# Transform the input data, X, to its reduced dimensional representation
X_transformed = pca_tuned.transform(X.toarray())
# Print the shape of the transformed data to see the number of samples and transformed
print(X_transformed.shape)

```

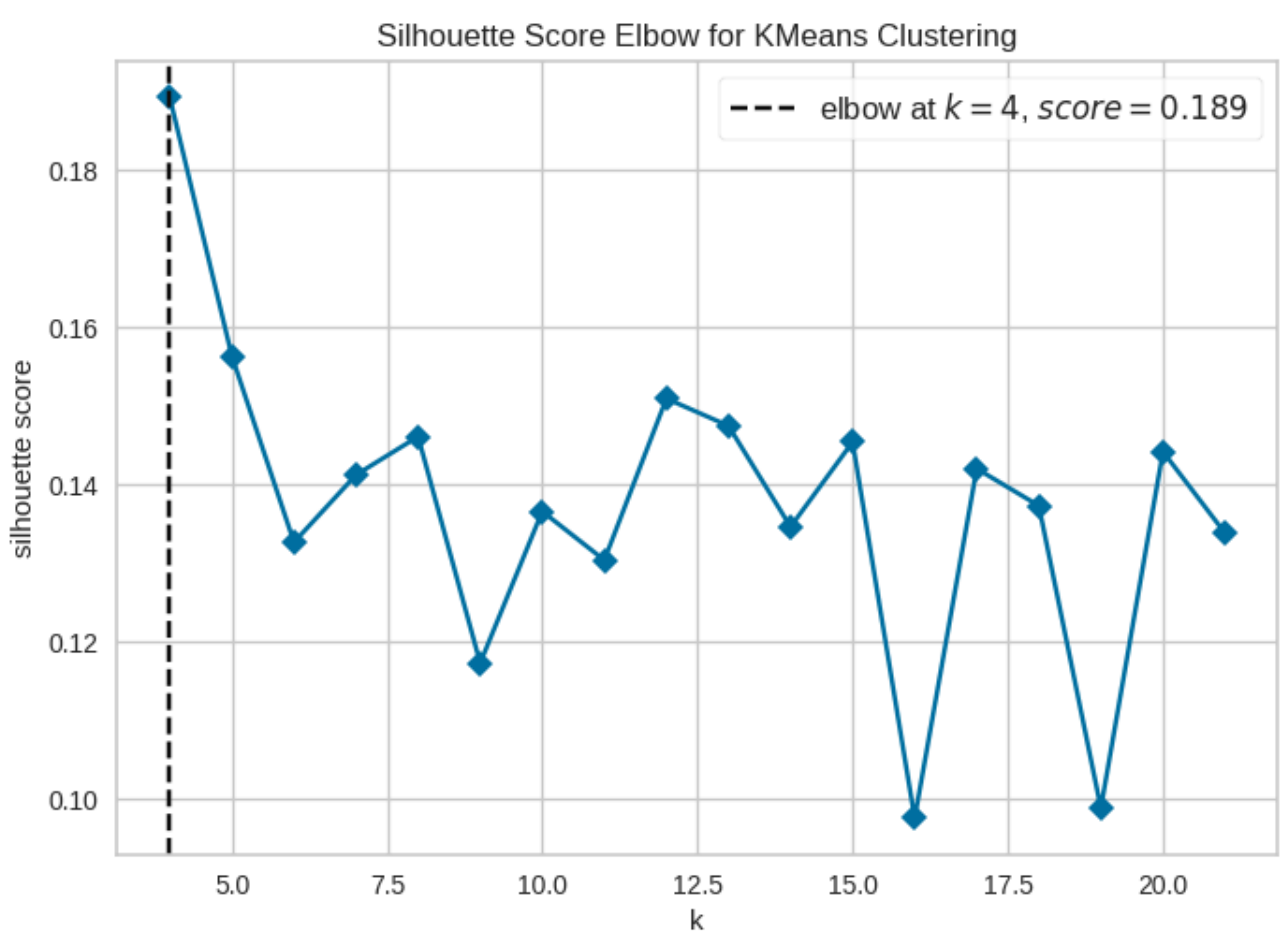
```
(7787, 1688)
```

```
X_transformed
```

```
array([[-1.65740427e-02, -6.40278678e-02,  3.58402088e-02, ...,
         2.50795095e-06, -2.09956326e-05, -5.92865061e-05],
       [-3.01357610e-02,  9.92646649e-03,  2.06066640e-02, ...,
        -4.72485577e-03, -2.75152605e-02, -1.25072990e-02],
       [-2.71113465e-02,  1.06339311e-02, -2.60965538e-02, ...,
         1.54202028e-02, -1.20993410e-02,  1.16357108e-02],
       ...,
       [-1.65740427e-02, -6.40278678e-02,  3.58402088e-02, ...,
         2.50795095e-06, -2.09956326e-05, -5.92865061e-05],
       [-1.65740427e-02, -6.40278678e-02,  3.58402088e-02, ...,
         2.50795095e-06, -2.09956326e-05, -5.92865061e-05],
       [ 3.17269207e-01,  1.43858130e-01,  1.37812482e-01, ...,
         8.47587621e-03,  8.97860703e-03, -9.24177368e-03]])
```

# 7. ML Model Implementation

```
from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer

# Initialize the KMeans model with a random_state of 5
model = KMeans(random_state=5)

# Initialize the KElbowVisualizer with the KMeans model and desired parameters
visualizer = KElbowVisualizer(model, k=(4, 22), metric='silhouette', timings=False, lo

# Fit the visualizer on the transformed data
visualizer.fit(X_transformed)

# Display the elbow plot
visualizer.show()
```



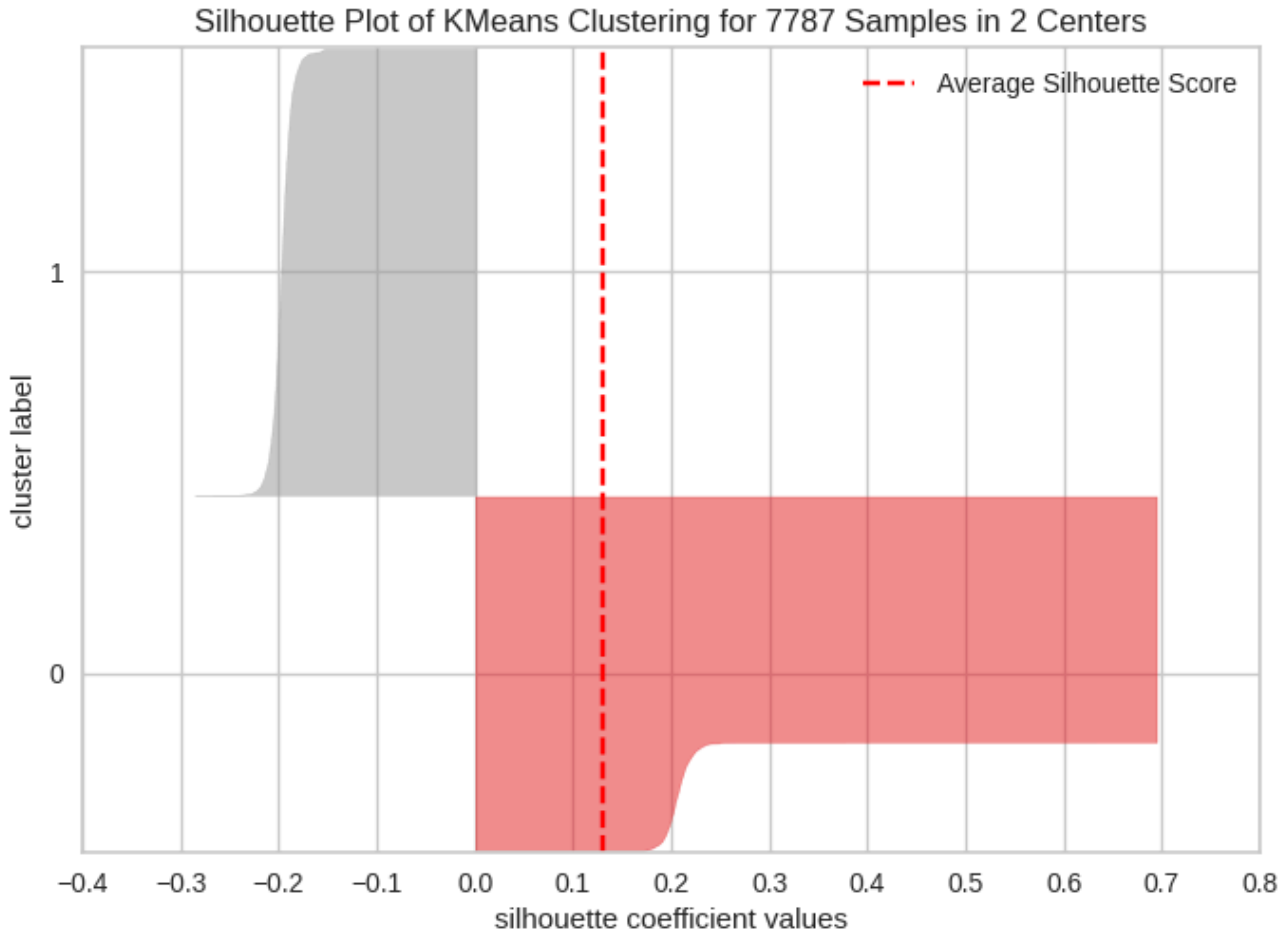

```
<Axes: title={'center': 'Silhouette Score Elbow for KMeans Clustering'}, xlabel='k',
ylabel='silhouette score'>
```

**The plot will also indicate the "elbow" point, which represents the recommended number of clusters based on the selected metric.Using elbow plot with the optimal number of 5 clusters** .

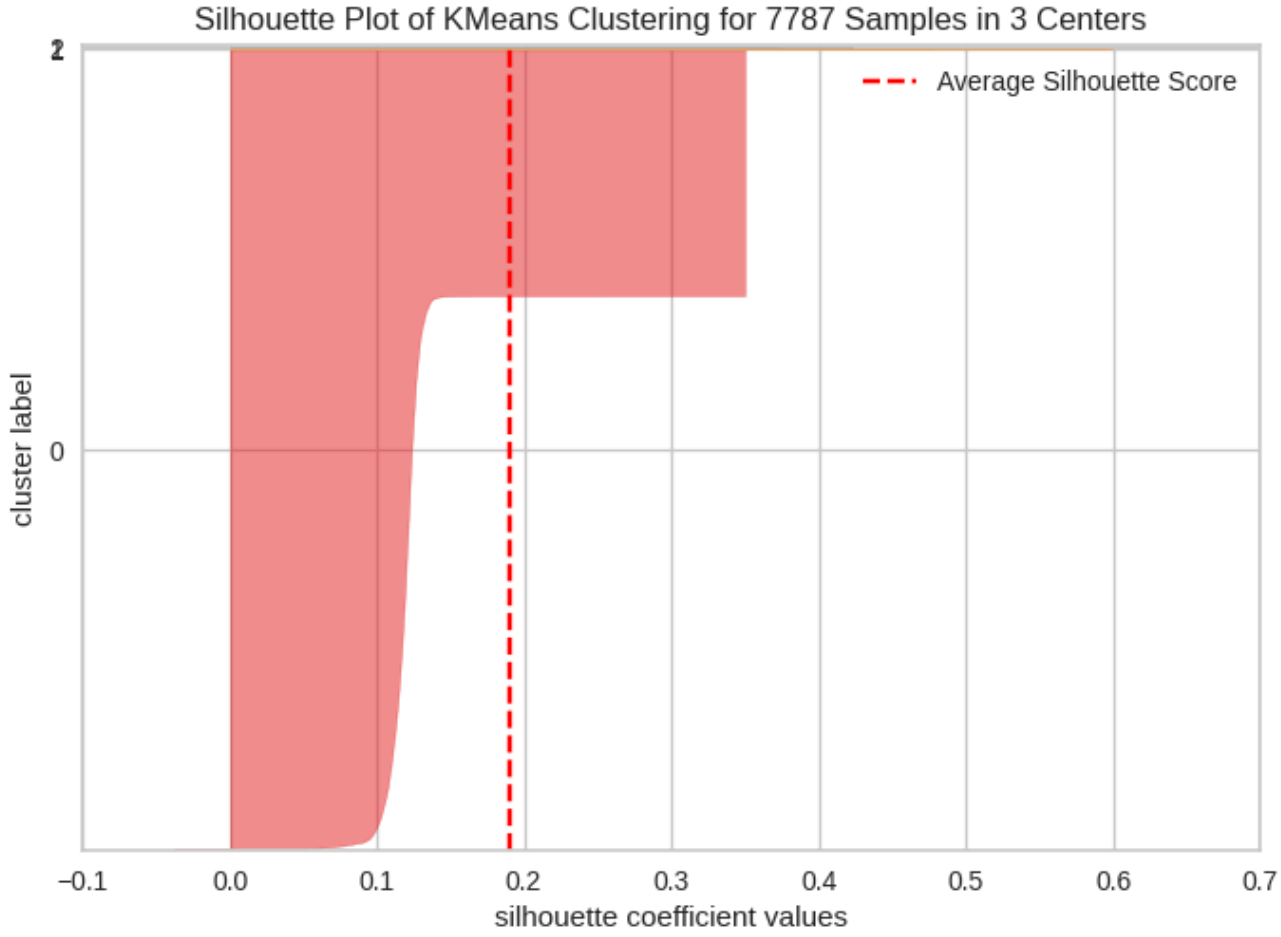
```
from yellowbrick.cluster import SilhouetteVisualizer
from sklearn.metrics import silhouette_score, silhouette_samples

def silhouette_score_analysis(n):

  for n_clusters in range(2,n):
      km = KMeans (n_clusters=n_clusters, random_state=5)
      preds = km.fit_predict(X_transformed)
      centers = km.cluster_centers_

      score = silhouette_score(X_transformed, preds, metric='euclidean')
      print ("For n_clusters = {}, silhouette score is {}".format(n_clusters, score))

      visualizer = SilhouetteVisualizer(km)

      visualizer.fit(X_transformed) # Fit the training data to the visualizer
      visualizer.poof() # Draw/show/poof the data
```

```
silhouette_score_analysis(15)
```

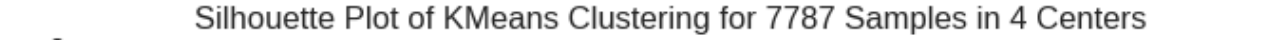
For n_clusters = 2, silhouette score is 0.13084475553157582

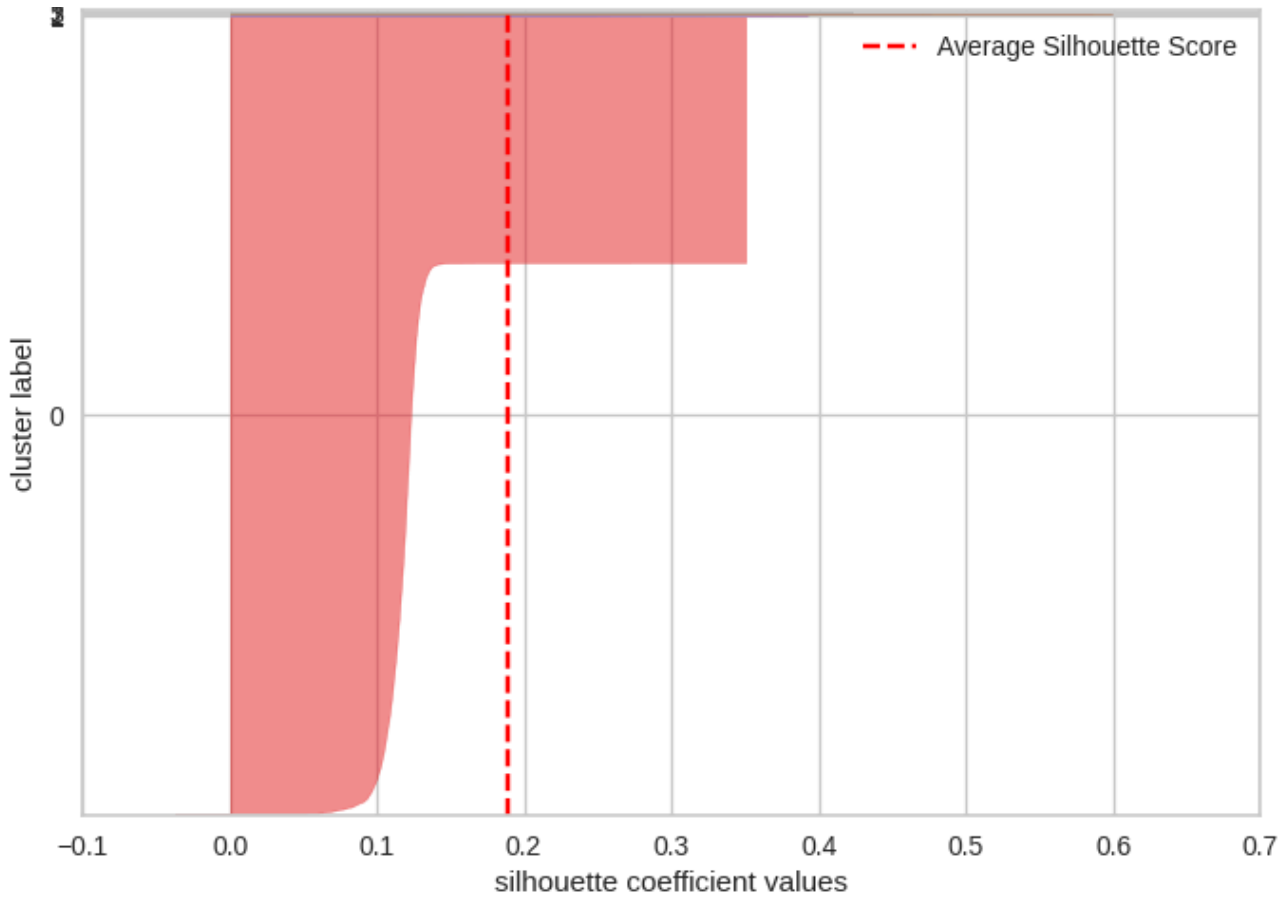



For n_clusters = 3, silhouette score is 0.18982719265623804

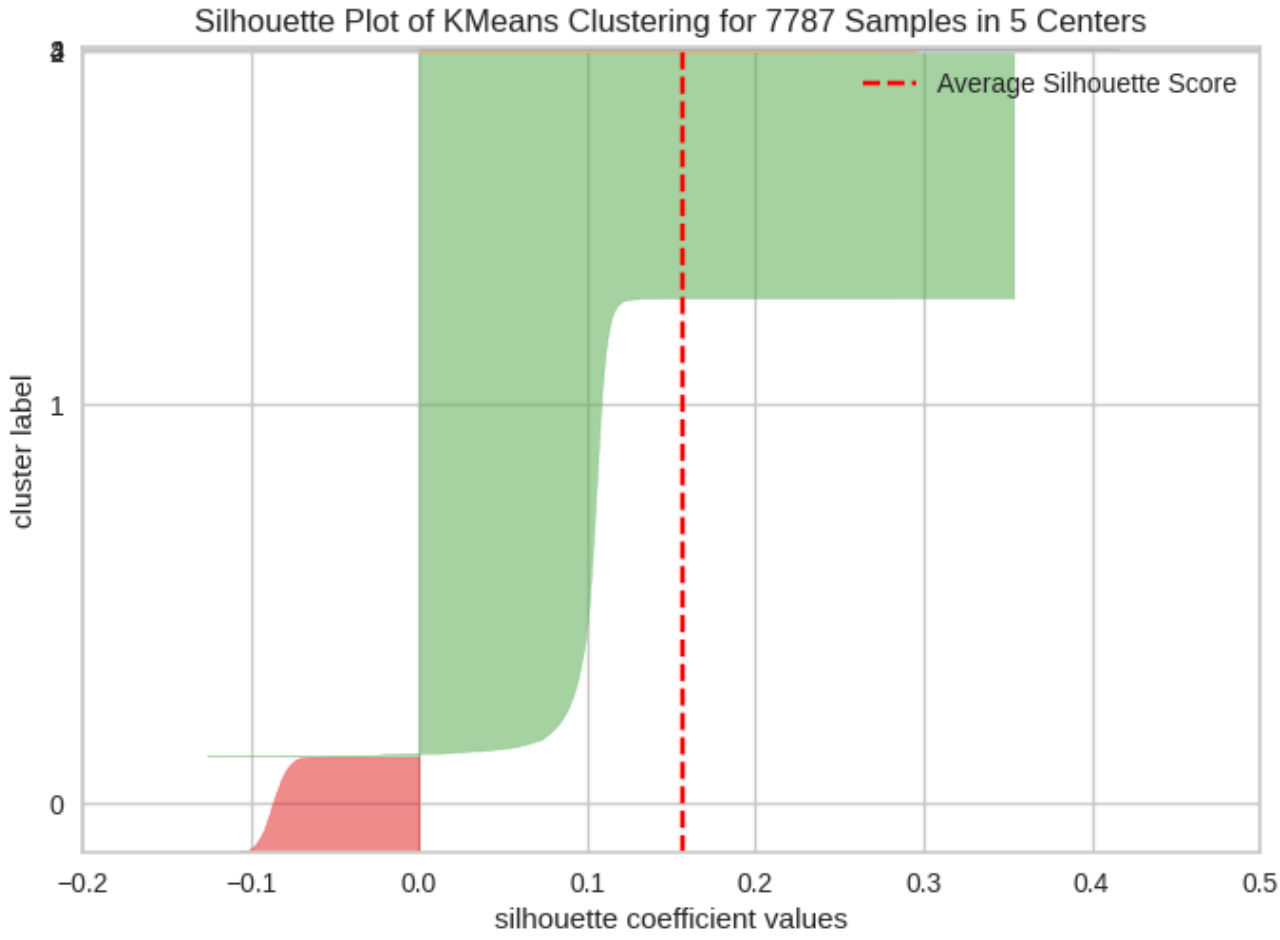



For n_clusters = 4, silhouette score is 0.18915068754624137

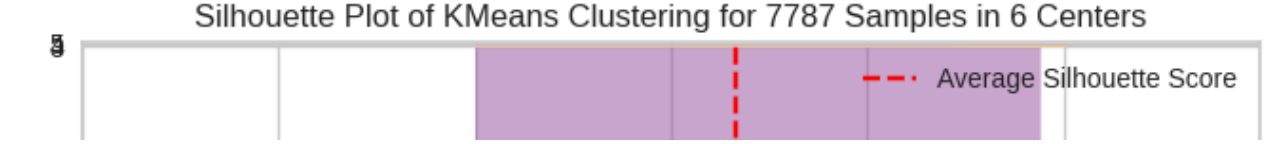


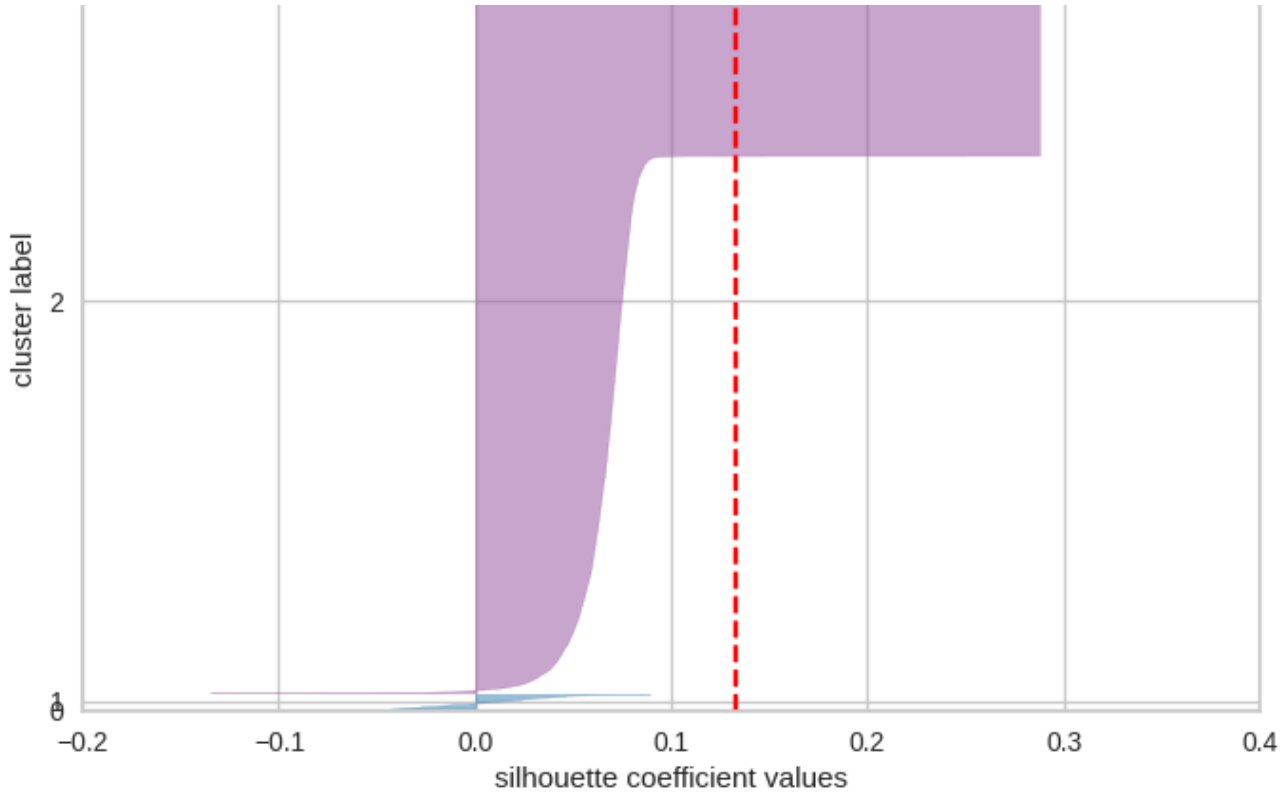



For n_clusters = 5, silhouette score is 0.15629223022760141



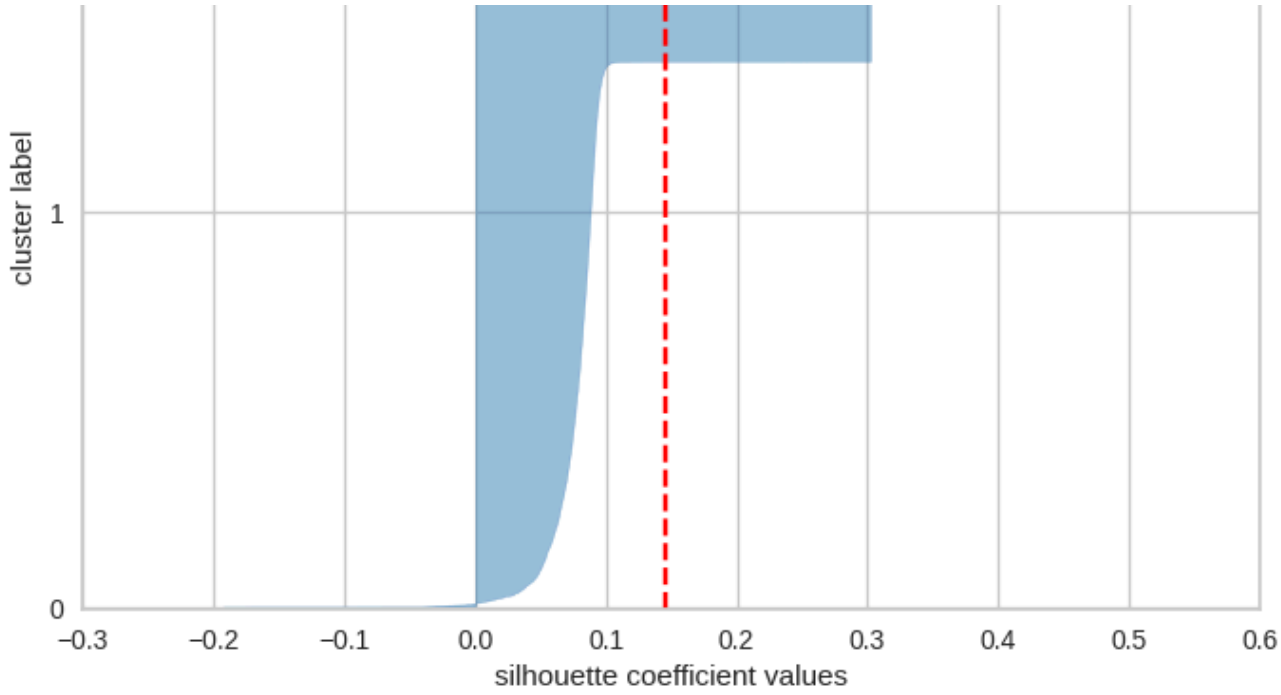

For n_clusters = 6, silhouette score is 0.13265033236965473



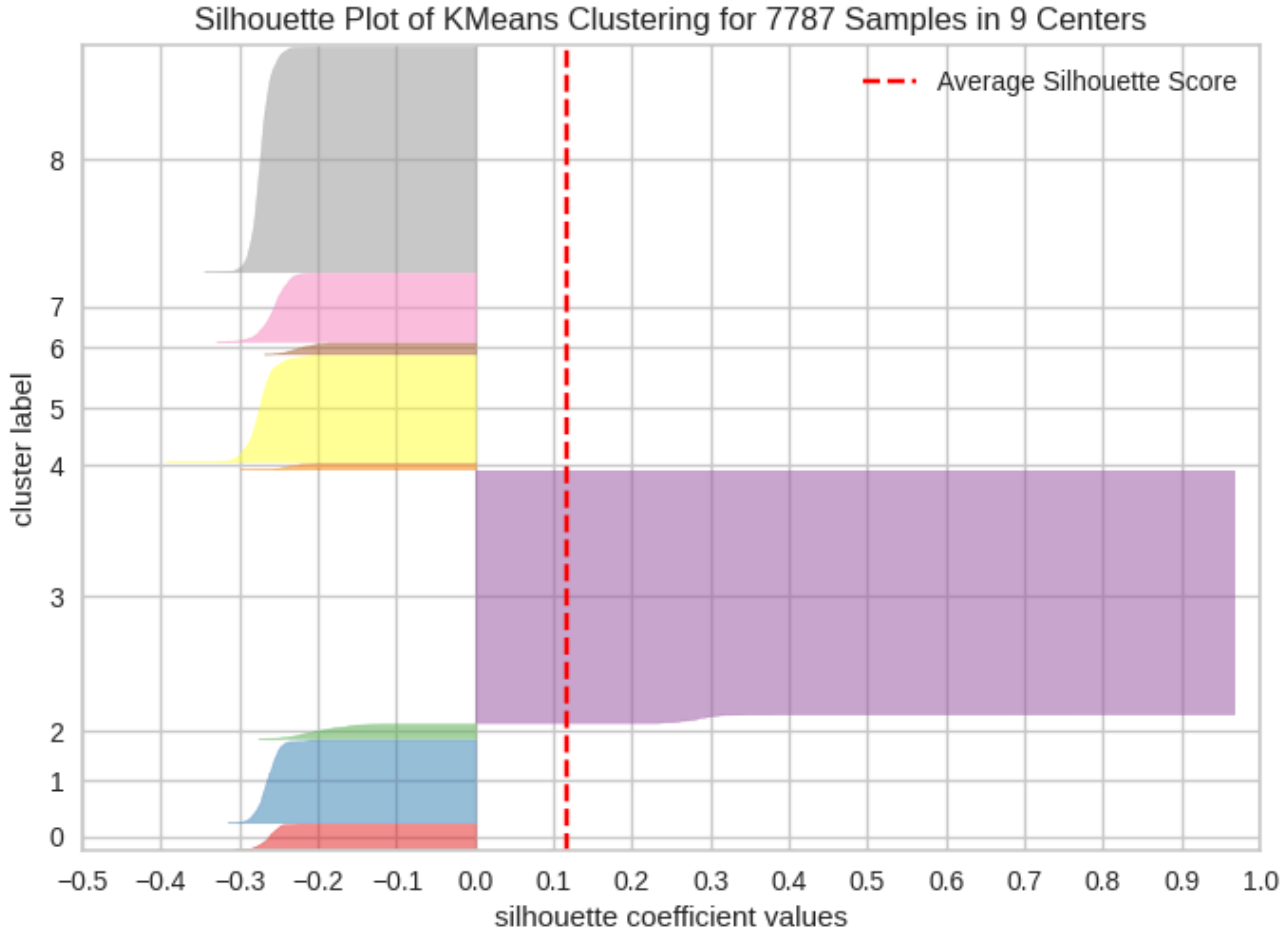



```
For n_clusters = 7, silhouette score is 0.14121649283112092
```

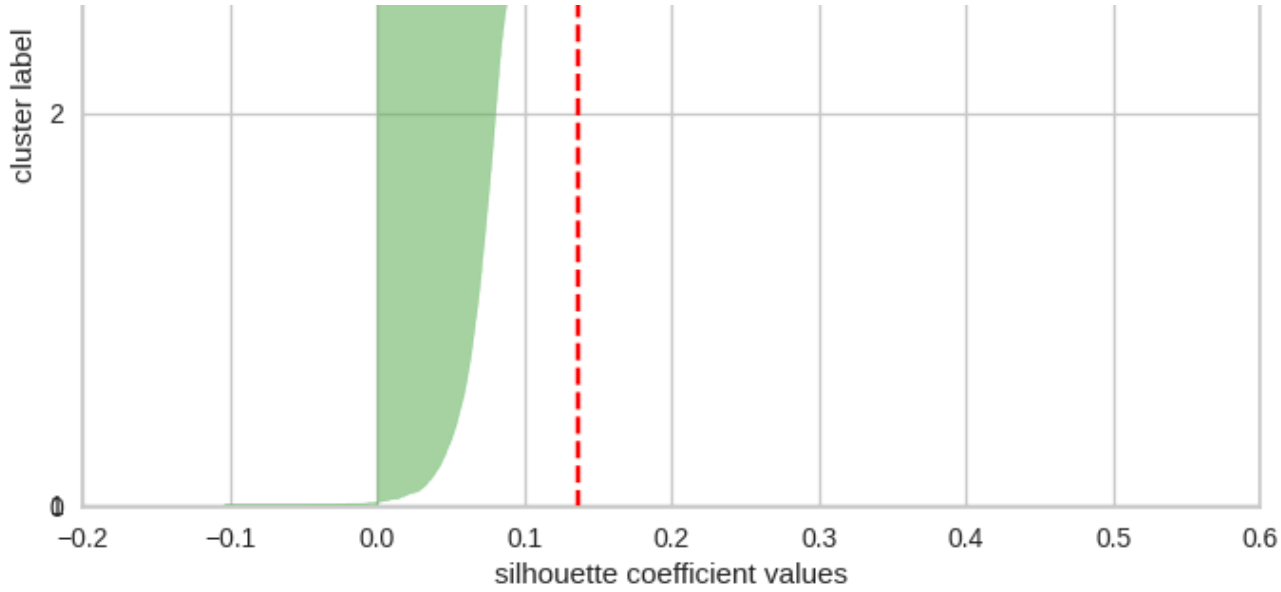

```
For n_clusters = 8, silhouette score is 0.14607911059936912
```



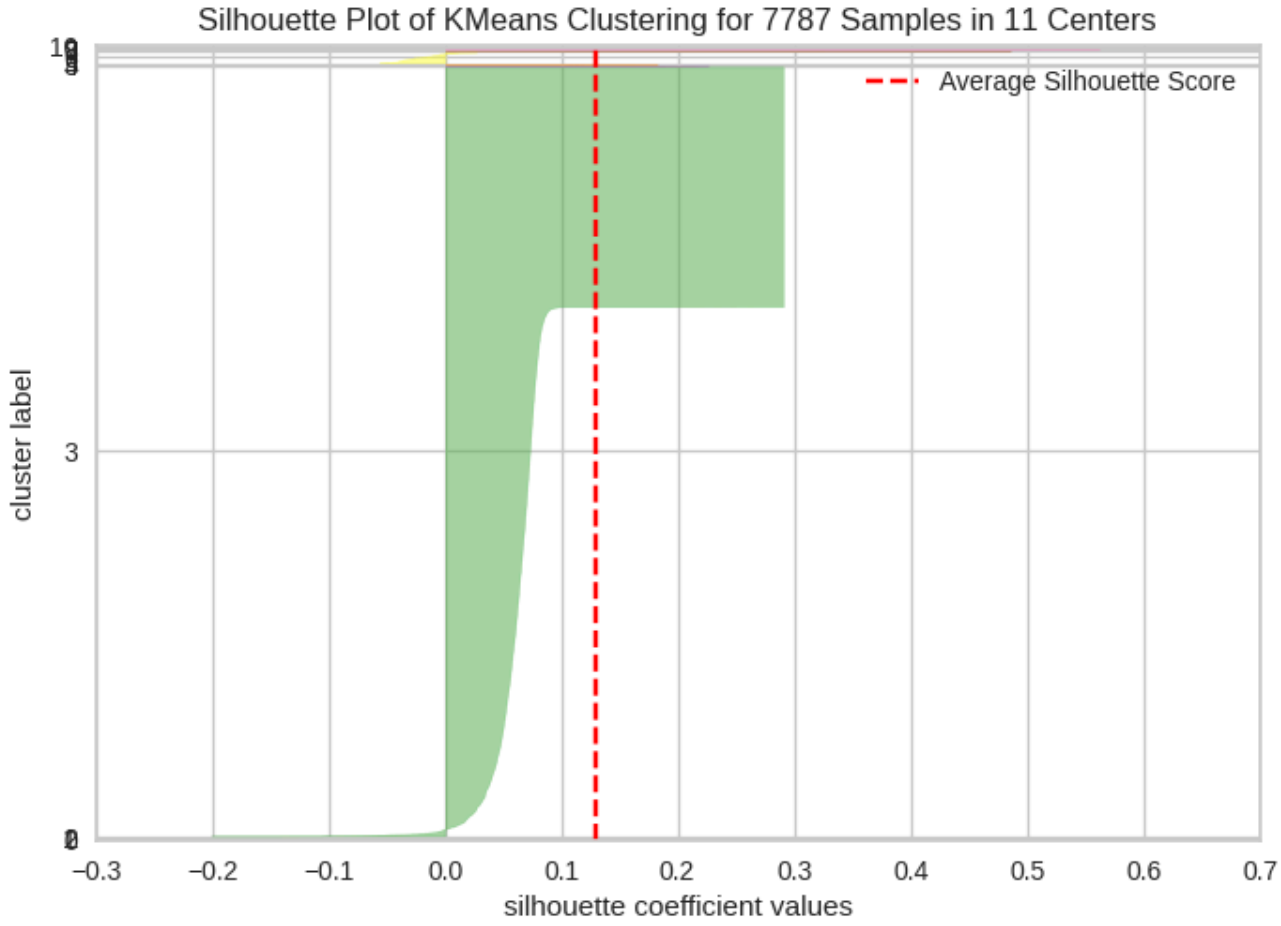

For n_clusters = 9, silhouette score is 0.11730444656614080



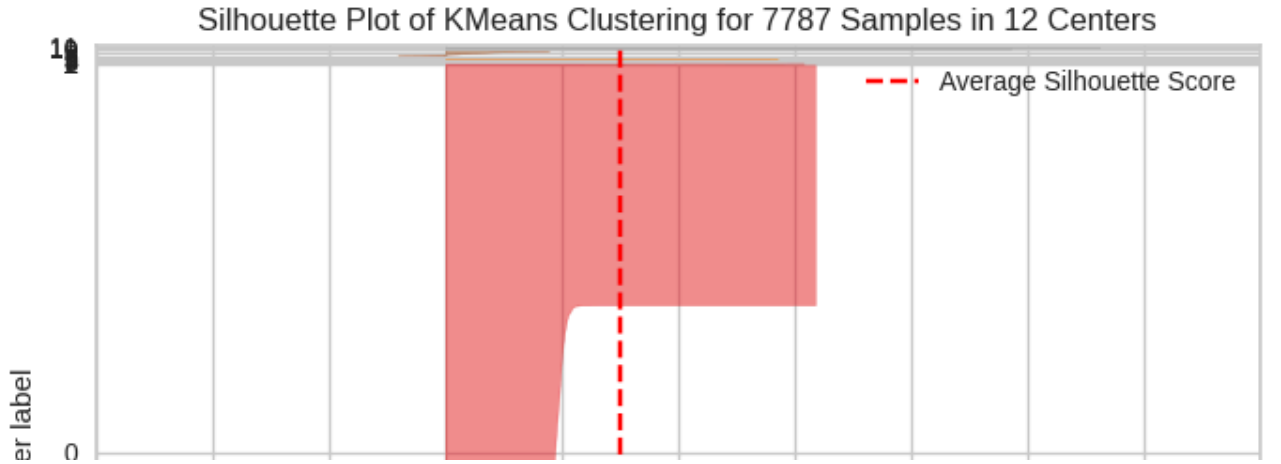

For n_clusters = 10, silhouette score is 0.13657591753039514

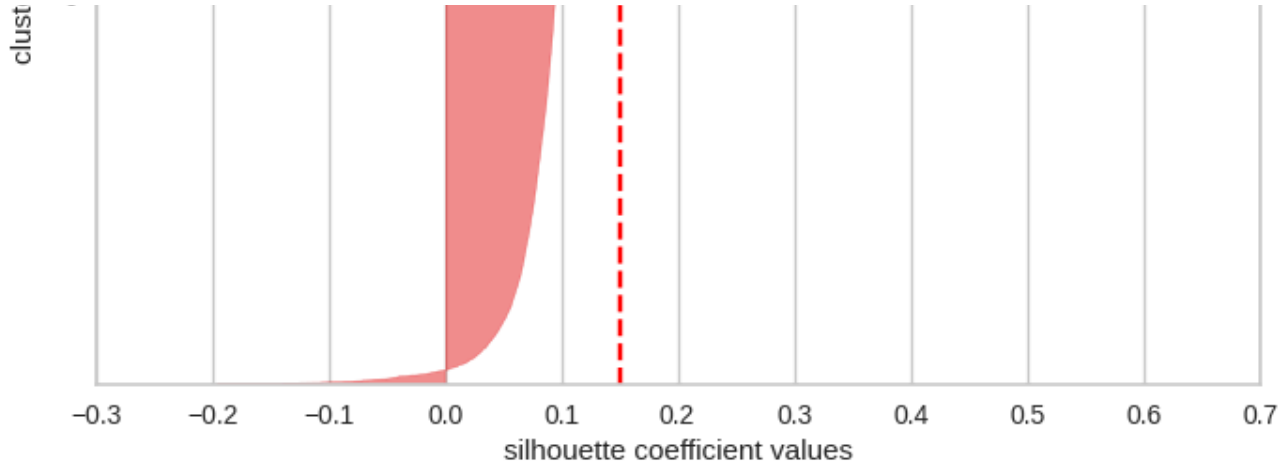



For n_clusters = 11, silhouette score is 0.13038424133679777

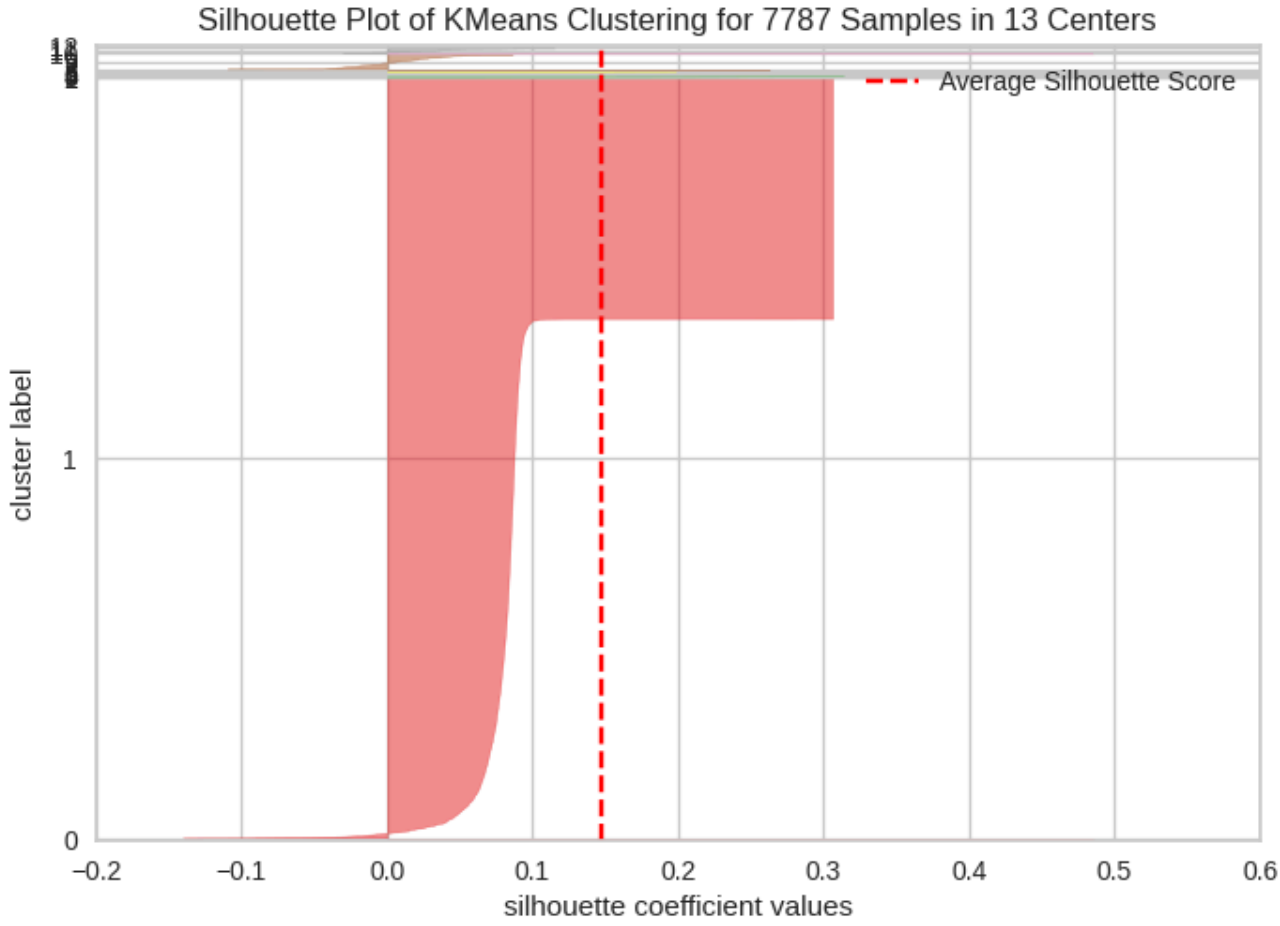



For n_clusters = 12, silhouette score is 0.150912020628163

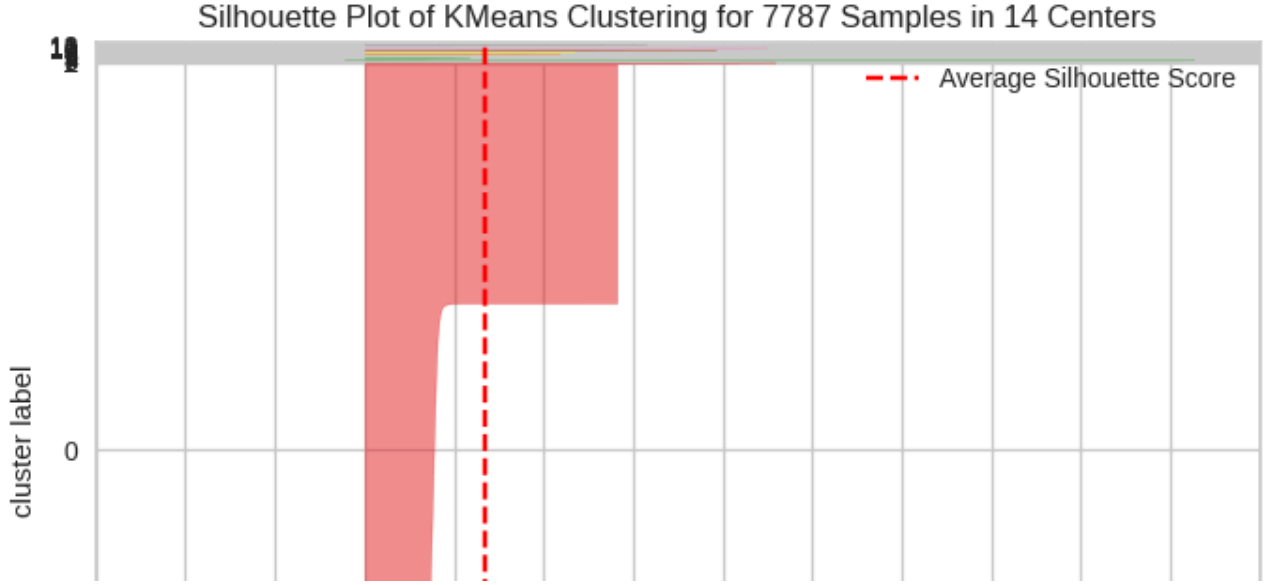


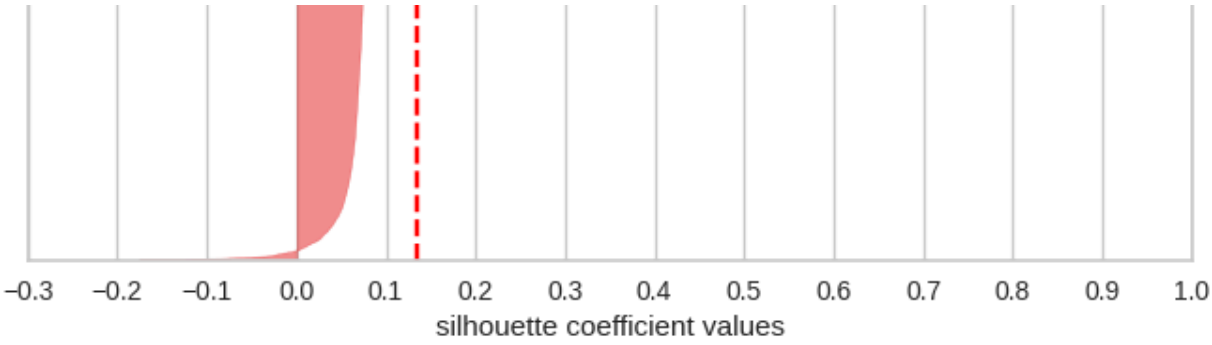



For n_clusters = 13, silhouette score is 0.147444873039125



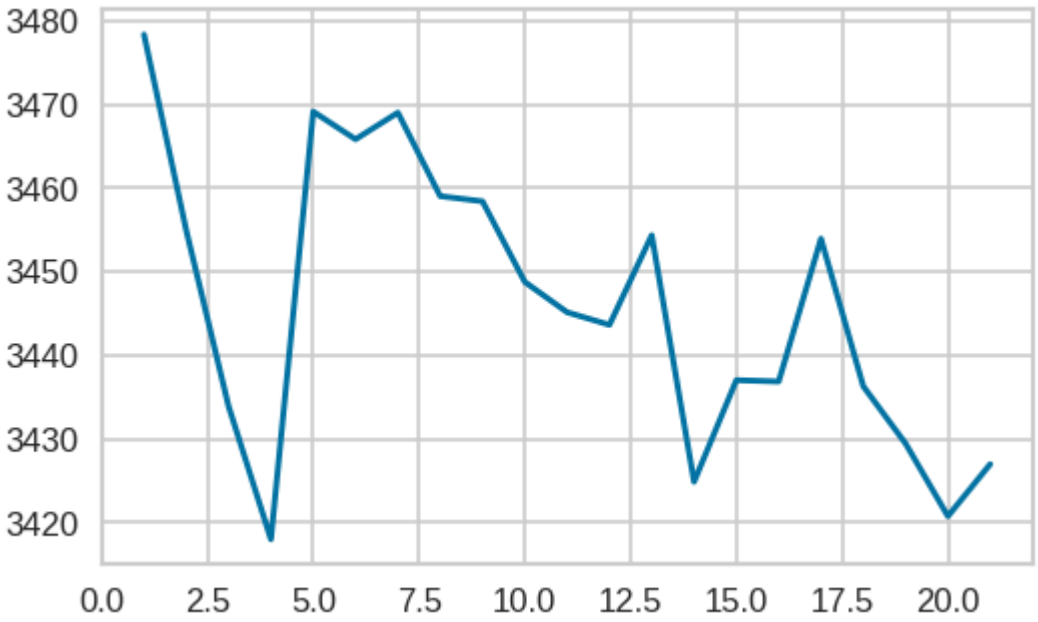

For n_clusters = 14, silhouette score is 0.1346113734311055




−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
silhouette coefficient values

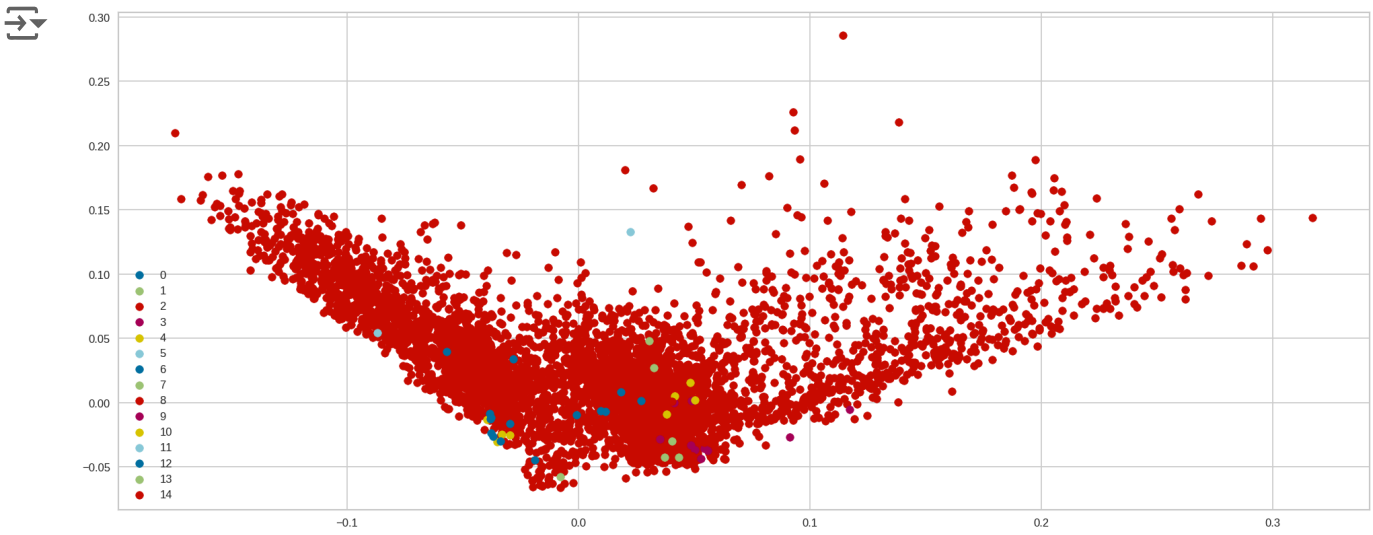
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Create a figure with a specific size and resolution
plt.figure(figsize=(5, 3), dpi=120)

# Initialize an empty list to store the within-cluster sum of squares (WCSS)
wcss = []

# Iterate over different numbers of clusters
for i in range(1, 22):
    # Create a KMeans model with default parameters
    model = KMeans(random_state=0)

    # Initialize the KMeans algorithm with specific parameters
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_st

    # Fit the KMeans algorithm to the transformed data
    kmeans.fit(X_transformed)

    # Append the WCSS to the list
    wcss.append(kmeans.inertia_)

# Plot the number of clusters against the WCSS
plt.plot(range(1, 22), wcss)
```

```
[<matplotlib.lines.Line2D at 0x7e81192fe020>]
```




```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import numpy as np

```

```
# Create a figure with a larger size and resolution
plt.figure(figsize=(20, 8), dpi=120)

# Initialize a KMeans model with 15 clusters
kmeans = KMeans(n_clusters=15, init='k-means++', random_state=9)

# Fit the KMeans algorithm to the transformed data
kmeans.fit(X_transformed)

# Predict the labels of the clusters
label = kmeans.fit_predict(X_transformed)

# Get unique labels from the predictions
unique_labels = np.unique(label)

# Plot the results
for i in unique_labels:
    # Scatter plot the points belonging to each cluster
    plt.scatter(X_transformed[label == i, 0], X_transformed[label == i, 1], label=i)

# Display a legend to identify the clusters
plt.legend()

# Show the plot
plt.show()

```




```
# Add cluster values to the dateframe.
df['cluster_number'] = kmeans.labels_
```

```
df.head(1)
```

| | show_id | type | title | director | cast | country | date_added | release_year | rat |
|---|---|---|---|---|---|---|---|---|---|
| **0** | s1 | TV Show | 3% | NaN | João Miguel, Bianca Comparato, Michel Gomes, R... | Brazil | 2020-08-14 | 2020 | TV- |

```
# Count the number of movies or TV shows in each cluster
cluster_content_count = df['cluster_number'].value_counts().reset_index().rename(colum

# Print the cluster content count
print(cluster_content_count)

```

```
    cluster_number  count
0                2   7732
1                9      8
2               14      8
3               12      8
4                6      7
5               10      4
6                4      4
7                0      4
8               13      3
9                5      3
10               1      2
11               7      2
12               8      1
```

# 8- Recommender system

A **recommender system is a type of information filtering system that suggests items to users based on their preferences**, interests, or past behavior. **It is commonly used in various applications such as e-commerce websites, streaming platforms, social media, and more.** The