

# Streaming Content Dashboard

Harnur Singh  
20BCE1809  
SCOPE  
VIT Chennai  
Tamil Nadu  
India

[harnur.singh2020@vitstudent.ac.in](mailto:harnur.singh2020@vitstudent.ac.in)

Kashish Bajaj  
20BCE1790  
SCOPE  
VIT Chennai  
Tamil Nadu  
India

[kashish.bajaj2020@vitstudent.ac.in](mailto:kashish.bajaj2020@vitstudent.ac.in)

Ansh Goel  
20BCE1798  
SCOPE  
VIT Chennai  
Tamil Nadu  
India

[ansh.goel2020@vitstudent.ac.in](mailto:ansh.goel2020@vitstudent.ac.in)

**Abstract**— As we all know, data analysis and visualization are becoming increasingly essential in the modern world due to the way the human brain processes information. Using charts or diagrams to visualize vast quantities of complex data is simpler than poring over spreadsheets and reports. Data visualization is a fast and easy method to communicate concepts in a universal manner, and you can experiment with various scenarios by making minor modifications. People today do not want to squander time watching poor television, so they first check the ratings before deciding what to watch. Accordingly, we devised our project to create a streaming content interface that will enable us to visualize all of the most popular shows in every aspect that we can comprehend perfectly. Using K-means, we also clustered the combined data from Netflix, Hulu, Disney Plus, and Amazon Prime to construct a system for recommending movies similar to those the viewer has already viewed.

Netflix, Hulu, Disney Plus, Amazon Prime, recommendation system, text clustering, data visualisation and analytics, over-the-top (OTT) content, and k-means algorithms are included in this paper.

## I. INTRODUCTION

Recommender Systems (RSs) are characterized by the capability of filtering large information spaces and selecting the items that are likely to be more interesting and attractive to a user.

OTT Platforms are the biggest users of recommendation systems. So, in this Project we aim to visualize the content library of top OTT Platforms like Netflix, Disney Plus, Hulu and Amazon Prime. While doing this we will also discover correlations and recurring patterns in the dataset with interesting inferences.

Finally, we will see how the recommendation engine works to deliver similar content as quickly as possible.

## II. LITERATURE REVIEW

Shubhankar et al. [1] highlighted the main goal of ranking and modeling topic evolution, by an efficient algorithm. The topic, evolution, has become a challenging task over time for the researchers. They suggested the topic as a summary of its content, and introduced a unique approach, that assigns the rank to a topic by applying PageRank algorithm, without considering the time of research publication.

Song et al. [2] emphasized the use of text clustering for Topic Detection. The text clustering dominates other algorithms in

terms of time, computational complexity, and cost. There are many ways to transmit data over a network; still, we need methods to avoid noise and irrelevant information. They considered the unstructured and scattered data over the internet as text copra and introduced a two-step algorithm for clustering the text copra. The first step uses C-Process to create overlapping clusters with the help of Canopy Clustering. Canopy Clustering is usually applied before the K-mean algorithm to speed up clustering of large dataset. In the second step named K-means apply rough clustering on results based on the common points between the clusters. The K-Means uses the X-Means algorithm. The experiments have proved better performance than k-means clustering and single pass algorithms and proved to be more suitable for detection of online topics.

[3]. Furthermore, they have categorized topics based on the set of the topic and closed keyword-set. The closed keyword-set were formed such that; phrases were selected from topics of the publications along with a user-defined minimum support. PageRank algorithm has been applied in iterative fashion, to assign authorities to score on the research paper. The authority scores, based on popularity in the research community; however, the algorithm also identifies hot topics by evaluating them on a timeline and is shown as landmarks. They also tried to find fading topics; they first checked for topic detection, then evaluated landmark topics, and at the end tried to find fading topics. The algorithm proved as most effective and faster when tested on DBLP dataset.

Wu et al. [4] discussed CAR (Credible association rule), a new method to relate and track documents. The CAR does not use prior knowledge of category structure as compared to other automated procedures. The other traditional processes detect related documents based on the topic of documents, with the help of some predefined rules or categorization. This method makes use of the term frequency-inverse document frequency (TF-IDF) as a feature pre-selection set. TF-IDF is a numerical statistic which shows how important words are to a document. After the feature subset selection, CAR and minimal clique algorithms were applied. These two algorithms use an adjacency matrix to produce credible association rules. Refinements, remove noise and familiar words with high frequency that are not related to the topic of the document. An important level of reliability, availability and performance are achieved by applying refinements like Inverse Document Frequency (IDF) and quasi-maximal cliques.

Jo et al. in [5] proposed algorithms for the topic detection from linked textual corpus using the relationship of topics in

terms of the distribution and the link distribution. Their algorithm generates a ranked list of the topics the method has shown effective results with arXiv and Citeseer.

Jo et al. in [6] discovered rich patterns of topic evolution within built-in features of time-stamped documents. Instead of analyzing documents on the specific interval, the method focuses and treats topics separately as they appear over time in chronological order. The information is obtained such that; it qualifies the topic as either new or it has some similarity with existing topic. The result was visualized by chronological scanning on a graph known as topic evolution graph. The topological or time restrictions were not considered while building the graph. The nodes of the graph represent topics and the connection between the topic nodes represents the cross-citation relationship. This representation of information projects a huge amount of knowledge about topic evolution. Details about a single topic can be obtained by selecting a seed topic and studying its connections with other nodes. These connections change as time passes, the emergence of new topics adds new nodes to the graph and changes connections between them. The testing was carried out with the ACM repository.

Jayashri et al. [7] discussed retrieval of temporal and event-based knowledge from a vast collection of historical documents. The method uses temporal text mining (TTM) and Topic detection and tracking (TDT) techniques. The TTM extracts important patterns related to time, like collecting term frequency in a sequence of time; it also helps in keeping track of modification in the words with respect to time. In TDT, a clustering problem called Adaptive Resonance Theory (ART) is used, that tracks unknown topics in the system and assigns them to previously identified topics. The Evaluation of such information is usually carried at the time of its arrival, which helps to consider temporal properties of the topics. The Evolution is implemented using an incremental algorithm. The experiments helped to discover latest trends and identify trends that cannot be obtained from documents if analyzed individually.

Cui et al. [8] highlighted topic evolution in text data as an important task. The Text Flow has been introduced; which studies various patterns that appear from several topics. Three-level features selection was conducted namely keywords, critical events, and evolution trends, then these features were visualized via a coherent visualization technique that shows the complex and important relation between them. The Text Flow is an interactive visual analysis tool that helps users analyze how and why the correlated topics change over time. First patterns related to merging/splitting are discovered via hierarchical Dirichlet process employed in incremental fashion followed by extraction of keyword correlation and critical events. The result can be visualized by a three-level directed acyclic graph such that the user can deduce various information at any level of consideration. This method helps users visually study the relation between two topics that is best as it represents this relationship in an interactive way.

Jin [9] focused on detecting topics with immense information available over the internet. They introduced a new method by

combining Suffix Tree Clustering (STC) and Semantic analysis that approaches the problem in two steps. In the first step feature selection is done with the help of NLP algorithm, by selecting meaningful words for clustering and the weight is assigned to word using term frequency-inverse document frequenting (TF-IDF). The NLP results in parts of the sentences in the form of the noun, verb and named entity. The result of feature selection is supplied to STC to form clusters where the score is assigned to them. TDT is applied to track topics, focusing on topic drifting. This is an inherent difficulty of topic evolution, which occurs over time as added information emerges. The clustered contents are represented via VSM (Vector Space Model) by selecting only top K words that are added to the vector

[10]. Semantic analysis is used to add significance to the topics and the significance can be measured by applying filters to the words and analyzing structure of words under a cluster which share the same meaning. The experiments proved that topics can be tracked effectively.

### III. PROPOSED METHODOLOGY

1. Define the objectives: Determine the streaming dashboard's data, insights, and audience.
2. Select data sources for your streaming dashboard. Real-time data streams, databases, APIs, and other updated data sources are examples.
3. Pre-process data in R Studio for Tableau visualisation. This may require cleaning, converting, calculating, and gathering data for your dashboard.
4. produce visualisations: Use Tableau to produce streaming-friendly line charts, bar charts, gauges, and maps. Choose dynamic, real-time visualisations.
5. Connect streaming data sources to Tableau. Tableau supports live connections to databases, APIs, and streaming services.
6. Customize dashboard layout: Arrange visualizations, filters, and other components in Tableau. Customize the dashboard for your audience and project goals.
7. Publish the dashboard: Tableau Server or Tableau Public, depending on your needs and accessibility. Depending on your platform, you may share the live dashboard with your team, stakeholders, or the public.
8. Configure Tableau to automatically update data at your streaming frequency. This ensures that the dashboard shows real-time or near-real-time data.
9. Optimize streaming dashboard performance: Monitor data refresh rate, visualisation responsiveness, and user experience. To stream dashboard material smoothly, optimise performance.

10. Improve your streaming dashboard depending on comments, insights, and evolving needs. To maintain the dashboard current and useful, update visuals, data sources, and other components.

Following this strategy, you may use R Studio and Tableau to construct a streaming dashboard that shows real-time or near-real-time data, offering insights and visualisations for decision-making and data-driven analysis.

#### IV. NOVELTY

Our project's primary objective, or what we call "Novelty," lies in the implementation of sophisticated sub-algorithms such as Feature Engineering and Hierarchical Clustering in the data pre-processing process. This sets our project apart from others as it addresses the labour-intensive nature of most dashboard content creation. By incorporating the aforementioned algorithms, we have significantly reduced the time complexity involved, making it compatible with any user-friendly UI for seamless integration.

In today's modern world, k-means clustering accounts for 95% of implementation in data analysis, with the remaining 5% utilizing various other algorithms. This underscores the significance of our project, as it goes beyond conventional approaches and leverages cutting-edge techniques to streamline data processing and improve efficiency. The inclusion of Feature Engineering and Hierarchical Clustering elevates our project to a new level of sophistication, offering a novel and robust solution for data pre-processing in dashboard content creation.

#### V. DATA SET USED

For this project we will use 4 datasets containing of listings of all the movies and tv shows available on Netflix, Hulu, Disney Plus and Amazon Prime, along with details such as - cast, directors, ratings, release year, duration, etc. In total there are approximately 22k observations. It is obtained from Kaggle Open-Source Dataset Library.

##### A. Feature components for Analysis and Visualisation

For this visualization and analysis, we use feature attributes from the dataset, namely,

- Type\_
- Title\_
- Director\_
- Cast\_
- Country\_
- Date Added\_

- Release Year
- Rating
- Duration
- Listed In
- Description

Each individual dataset contains all the following attributes. During the project we will combine all 4 datasets into one and then we will append a column denoting the OTT platform.

#### VI. SYSTEM ARCHITECTURE AND DESIGN (Parameters and Discussion)

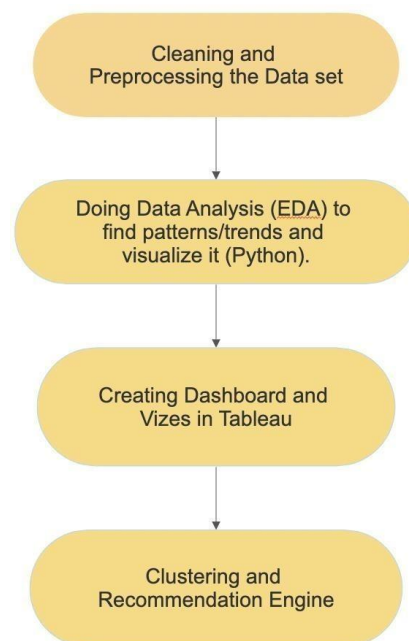


Fig A – Structural flow for this project

##### A. Module 1: Data Cleaning and Dataset Analysis

After Importing the data set, we need to clean it and analyze what data we were able to collect. After this we can easily plan which parameters to visualize.

##### B. Module 2: Data Analysis to find patterns/trends and visualize it

The attributes from the obtained data set are compared with each other to find correlations and dependencies and then these are visualized using distinct types of graphs. We can use these graphs to visualize common trends in the dataset.

### C. Module 3: Creating Dashboard and Visualize in Tableau.

We then use Tableau to further Visualize the Dataset and create interactive Dashboards. We found Tableau to be an incredibly versatile and powerful tool for this purpose.

### D. Module 4: Clustering and Recommendation Engine.

We will use K-Means clustering to cluster similar data. We then append the cluster id generated to the combined dataset to facilitate the recommendation engine

#### D.1. K-Means Algorithm

K-means algorithm is an iterative algorithm that tries to partition the dataset into  $K$  pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible.

The way K-means algorithm works is as follows :

1. Specify number of clusters  $K$ .
2. Initialize centroids by first shuffling the dataset and then randomly selecting  $K$  data points for the centroids without replacement.
3. Keep iterating until there is no change to the centroids.
4. Compute the sum of the squared distance between data points and all centroids.
5. Assign each data point to the closest cluster (centroid).

Compute the centroids for the clusters by taking the average of the all- data points that belong to each cluster.

The objective function is:

$$J = \sum_{i=1}^m \sum_{k=1}^K w_{ik} \|x^i - \mu_k\|^2$$

#### D.2. Recommendation Engine

The recommendation Engine takes a Movie or Show Title as an input. It then finds the cluster id of that entry. It uses the cluster id to reduce the search space.

Now it runs a text similarity check between the description of entered show or movie to find similar content from that cluster.

Thus, using K-Means and Text Similarity, it achieves fast and accurate results.

## VII. STATISTICAL MODELLING & MODELLING

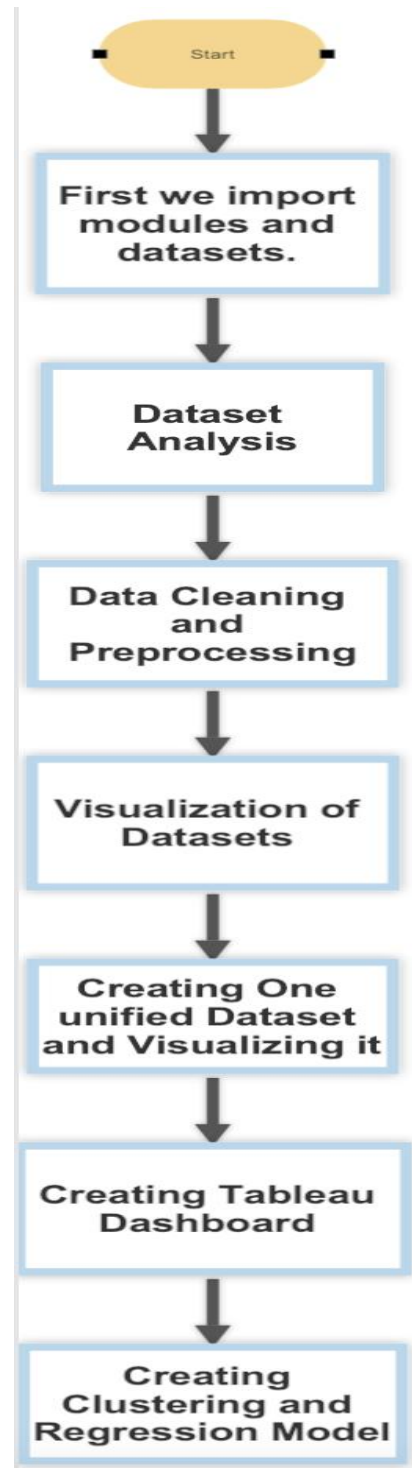
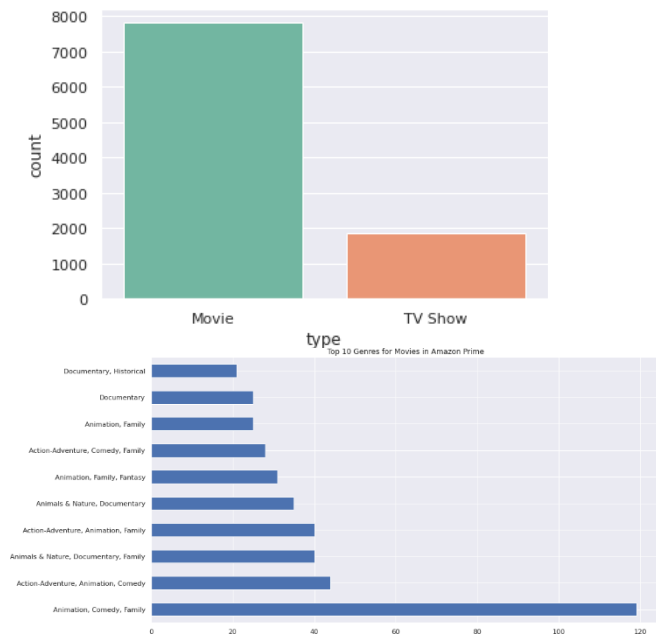


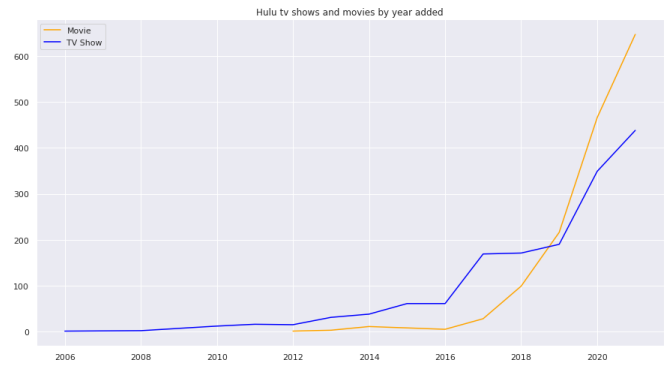
Fig B – Systematic Road Map for Streaming Dashboard Content

## VIII. RESULT (Discussion)

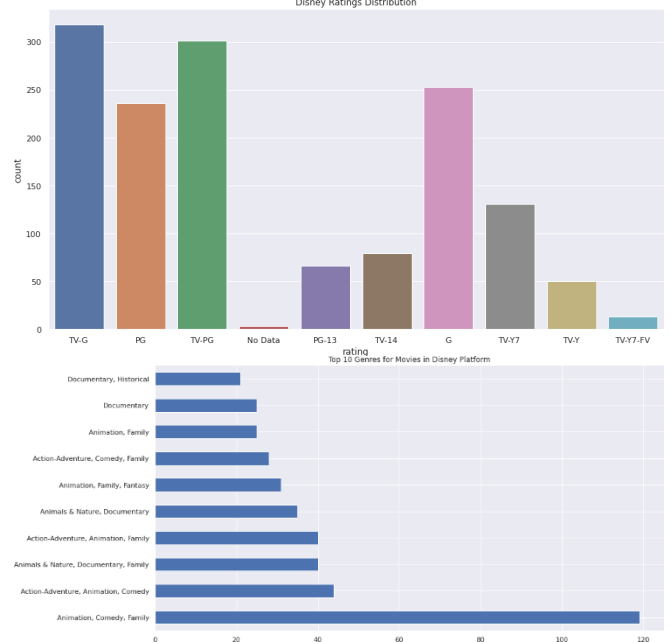
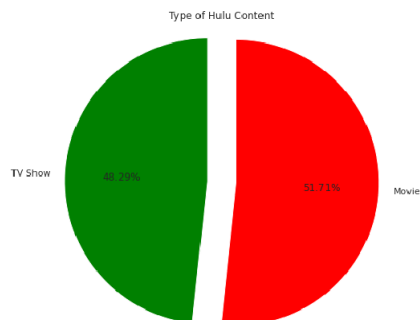
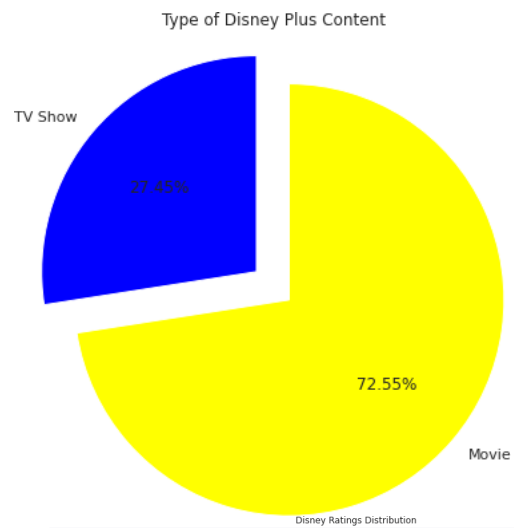
**Types of Amazon Prime Content**



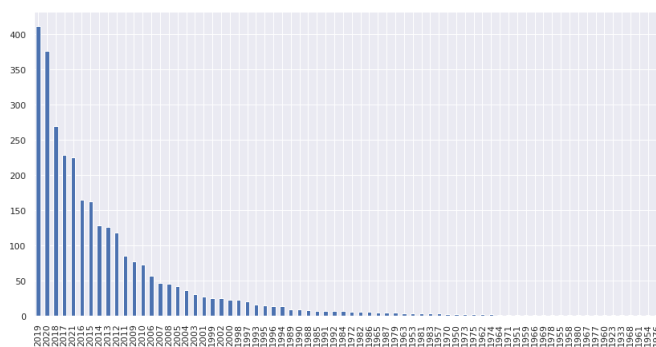
*Fig 1 – Types of Amazon Prime Content*



*Fig 2 – Types of Hulu Content*



*Fig 3 – Types of Disney Plus Content*





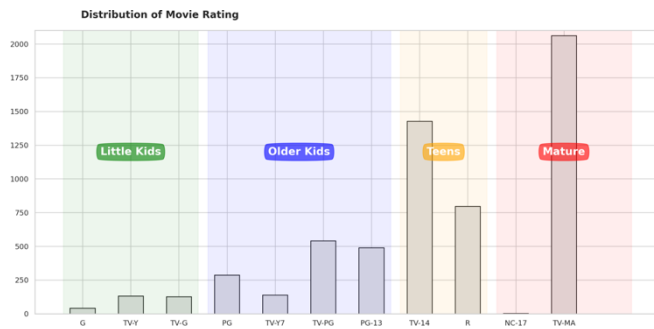


Fig 4 – Distribution of Movie Rating

Distribution of release by country (top 10)

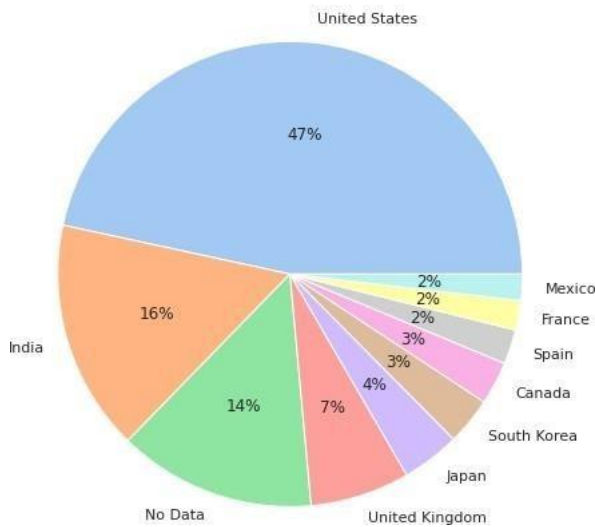


Fig 5 – Distribution of release by country

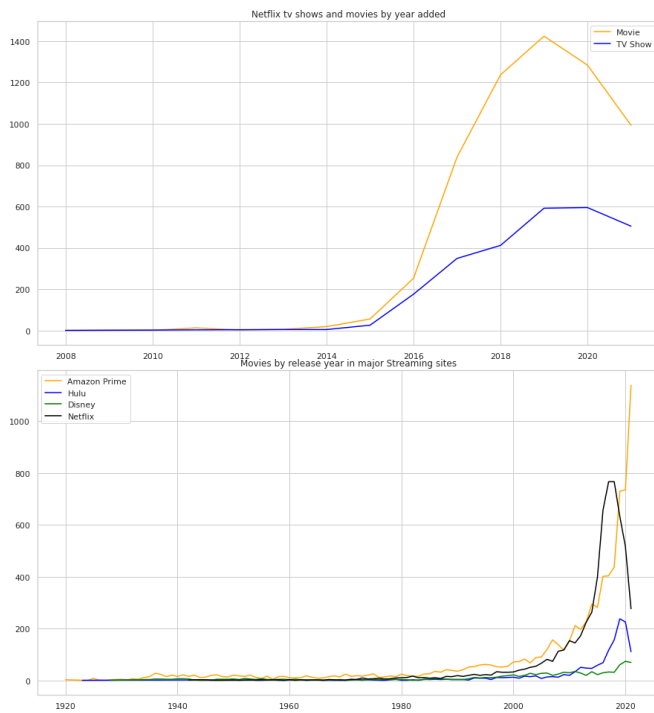


Fig 6 – Releases of movie year by year



Fig 7 – Region-wise distribution for Content

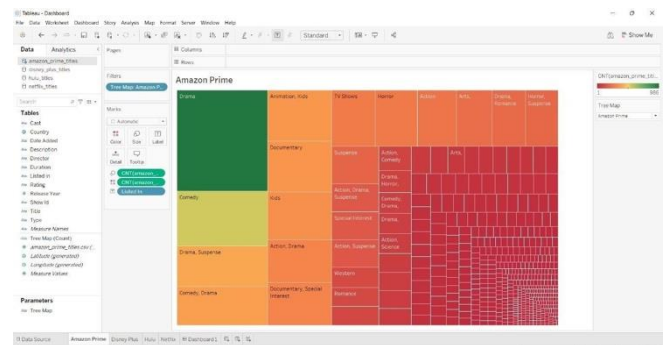


Fig 7.1 – Heat Map for Netflix

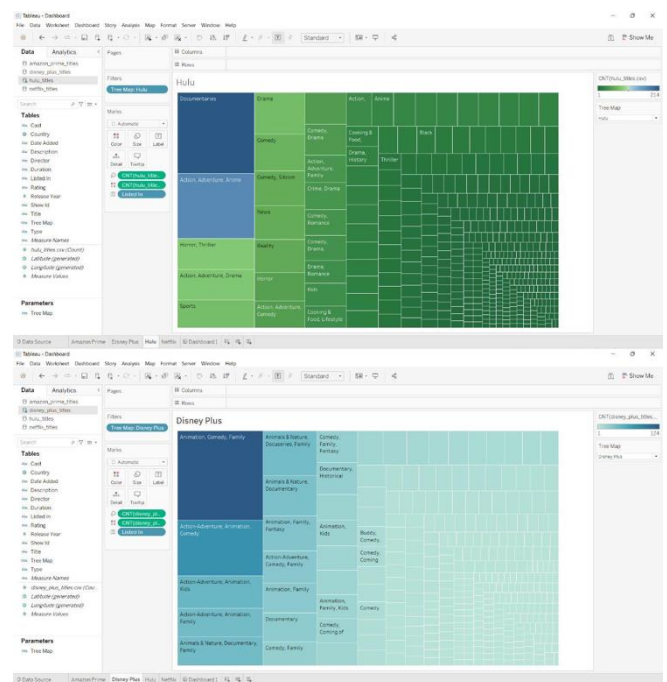


Fig 7.2 – Heat Map for Hulu & Disney Plus

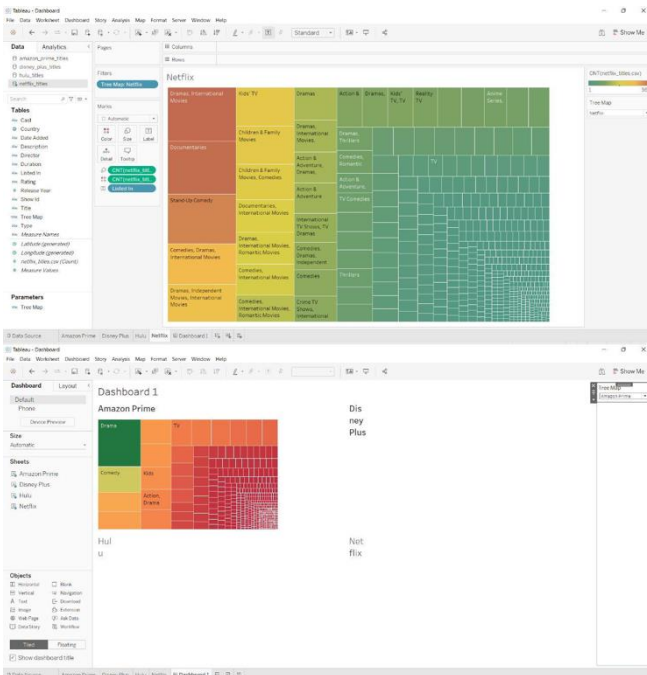


Fig 7.3 – Heat Map for Amazon Prime

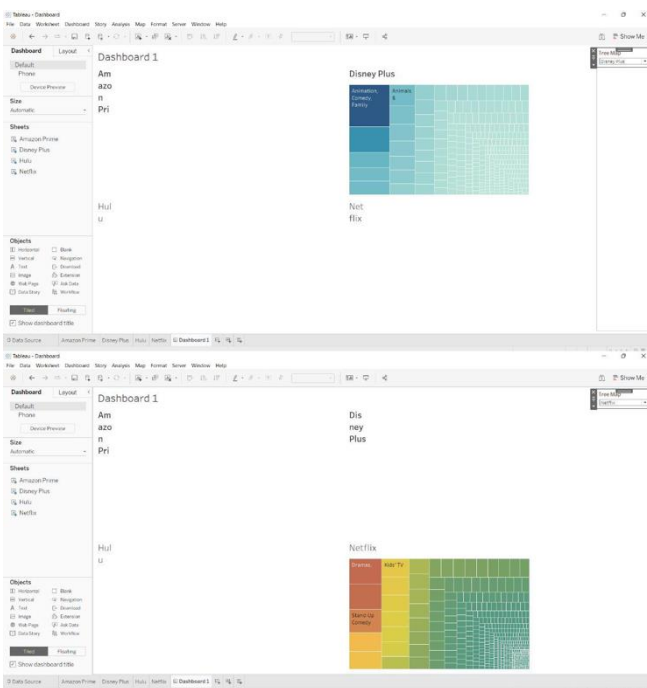


Fig 8 – Comparison between the various Heat Maps

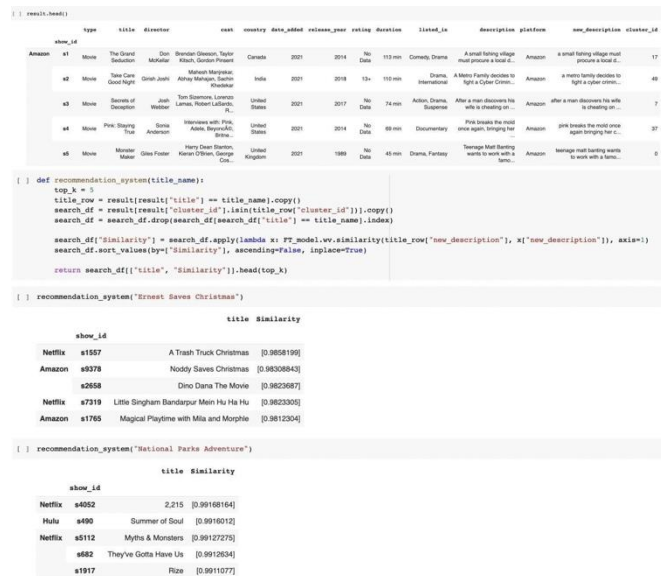


Fig 9 – Comparing the accuracy between the Algorithms

## IX. CONCLUSION

From the Visualization we gained a lot of Inferences. Like how each platform values movies more than tv shows. We also found that Amazon and Netflix have the biggest content library with Disney & Hulu slowly building their catalogues. We also saw how US is the biggest producer of OTT Content with India coming at a close Second. We also inferred how the growth of OTT Content Libraries has been meteoric in recent years, almost growing exponentially. We also saw the rating distribution between the OTTs and how they favor older teens/Adult markets as their main customer segment.

Finally, we created and tested the recommendation engine. We can see how such engines use clustering to reduce runtime dramatically while producing high quality results. This also highlights the importance of clustering data in large corporate environments like multinational OTT providers.

This proves that clustering is not just a mere visualization tool but also an especially important machine learning implementation that reduces runtimes in such demanding workloads drastically.

## X. FUTURE WORK

This report briefly describes simple models that predicts ratings. There are two others widely adopted approaches not discussed here: content-based and collaborative filtering. The recommender Lab package implements these methods and provides an environment to build and test recommendation systems.

## XI. APPENDIX

```
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
import matplotlib.pyplot as plt import seaborn as sns
import plotly.io as pio
from plotly.offline import iplot
from plotly.subplots import make_subplots
from wordcloud import WordCloud, STOPWORDS
import random
import re

df1 =
    pd.read_csv("amazon_prime_titles.csv",
                delimiter=";", encoding="latin-1",
                parse_dates=["date_added"], index_col=["show_id"])
df2 =
    pd.read_csv("hulu_titles.csv",
                delimiter=";", encoding="latin-1",
                parse_dates=["date_added"], index_col=["show_id"])
df3 =
    pd.read_csv("disney_plus_titles.csv",
                delimiter=";", encoding="latin-1",
                parse_dates=["date_added"], index_col=["show_id"])
df4 =
    pd.read_csv("netflix_titles.csv",
                delimiter=";", encoding="latin-1",
                parse_dates=["date_added"], index_col=["show_id"])
print("The size and shape of dataset 1") print(df1.size)
print(df1.shape)

print("The size and shape of dataset 2") print(df2.size)
print(df2.shape)

print("The size and shape of dataset 3") print(df3.size)
print(df3.shape)

print("The size and shape of dataset 4") print(df4.size)
print(df4.shape) df1.dtypes df2.dtypes

df3.dtypes

df4.dtypes

df1["date_added"] = df1["date_added"].dt.year
df1["date_added"].unique()

df1["date_added"].fillna(0, inplace=True)
df1["date_added"] = df1["date_added"].astype(int)
df1.head()

df1.loc[df1["date_added"]==0, ] df1.info()

df1.duplicated().sum()

df1.fillna("No Data", inplace=True) df1.isnull().sum()

df2['cast'] = df2['cast'].astype(str)
```

```
df2["date_added"] = df2["date_added"].dt.year
df2["date_added"].unique()

df2["date_added"].fillna(0, inplace=True)
df2["date_added"] = df2["date_added"].astype(int)

df2.loc[df2["date_added"]==0, ] df2.info()
df2.duplicated().sum()

df2.fillna("No Data", inplace=True) df2.isnull().sum()

df3["date_added"] = df3["date_added"].dt.year
df3["date_added"].unique()

df3["date_added"].fillna(0, inplace=True)
df3["date_added"] = df3["date_added"].astype(int)
df3.head()

df3.loc[df3["date_added"]==0, ] df3.info()
df3.duplicated().sum()

df3.fillna("No Data", inplace=True) df3.isnull().sum()

df4["date_added"] = df4["date_added"].dt.year
df4["date_added"].unique()

df4["date_added"].fillna(0, inplace=True)
df4["date_added"] = df4["date_added"].astype(int)
df4.head()

df4.loc[df4["date_added"]==0, ] df4.info()
df4.duplicated().sum()

df4.fillna("No Data", inplace=True) df4.isnull().sum()

df1.head() df2.head() df3.head()

sns.set(style="darkgrid")
ax = sns.countplot(x="type", data=df1, palette="Set2")
ax.set_title(f'Types of AmazonPrime
              Content', fontsize=15,
              fontweight='bold', position=(0.20, 1.03))

plt.figure(figsize = (15,8))
plt.title('Top 10 Genres for Movies in Amazon Prime')
df3[df3["type"]=="Movie"]["listed_in"].value_counts()[
:10].plot(kind='barh') plt.show()

d1 = df1[df1["type"] == "TV Show"] d2 =
df1[df1["type"] == "Movie"]

col = "release_year"

vc1 = d1[col].value_counts().reset_index()
vc1 = vc1.rename(columns = {col : "count", "index" :
col}) vc1['percent'] = vc1['count'].apply(lambda x :
100*x/sum(vc1['count'])) vc1 = vc1.sort_values(col)

vc2 = d2[col].value_counts().reset_index()
```



```
vc2 = vc2.rename(columns = {col : "count", "index" :
col}) vc2['percent'] = vc2['count'].apply(lambda x :
100*x/sum(vc2['count'])) vc2 = vc2.sort_values(col)
```

```
trace1 = go.Bar(x=vc1[col],
y=vc1['count'], name="TV Shows",
marker=dict(color="#a678de"))
trace2 = go.Bar(x=vc2[col],
y=vc2['count'], name="Movies",
marker=dict(color="#6ad49b"))
data = [trace1, trace2]
layout = go.Layout(title="Content added over the
years", legend=dict(x=0.1, y=1.1, orientation="h"))
fig = go.Figure(data, layout=layout) fig.show()
```

```
col = "rating"
```

```
vc1 = d1[col].value_counts().reset_index()
vc1 = vc1.rename(columns = {col : "count", "index" :
col}) vc1['percent'] = vc1['count'].apply(lambda x :
100*x/sum(vc1['count'])) vc1 = vc1.sort_values(col)
```

```
vc2 = d2[col].value_counts().reset_index()
vc2 = vc2.rename(columns = {col : "count", "index" :
col}) vc2['percent'] = vc2['count'].apply(lambda x :
100*x/sum(vc2['count'])) vc2 = vc2.sort_values(col)
```

```
trace1 = go.Bar(x=vc1[col],
y=vc1['count'], name="TV Shows",
marker=dict(color="#a678de"))
trace2 = go.Bar(x=vc2[col],
y=vc2['count'], name="Movies",
marker=dict(color="#6ad49b"))
data = [trace1, trace2]
layout = go.Layout(title="Content added over the
years", legend=dict(x=0.1, y=1.1, orientation="h"))
fig = go.Figure(data, layout=layout) fig.show()
df1['date_added'].value_counts().plot(kind='bar')
plt.figure(figsize=(14, 7))
labels=['TV Show', 'Movie']
plt.pie(df2['type'].value_counts().sort_values(),labels=lab
els,explode=[0.1,0.1],autopct='
%1.2f%%',colors=['green','red'], startangle=90)
plt.title('Type of Hulu Content')
plt.axis('equal') plt.show()
```

```
plt.figure(figsize=(14, 7))
df2['release_year'].value_counts().plot(kind='bar')
```

```
plt.figure(figsize = (45,30))
plt.title('Hulu ratings distribution seperated by type of
release') sns.countplot(x='rating', data=df2)
plt.show()
```

```
plt.figure(figsize=(15,8))
movie = df2[df2['type'] == 'Movie' ] tv = df2[df2['type']
== 'TV Show']
movie = movie[movie['date_added']>2000] tv =
tv[tv['date_added']>2000]
added_counts= movie['date_added'].value_counts()
added_tv_counts= tv['date_added'].value_counts()
```

```
sns.lineplot(x=added_counts.index,y=added_counts.valu
es, color="orange", label='Movie')
sns.lineplot(x=added_tv_counts.index,y=added_tv_count
s.values,color="blue", label='TV Show')
plt.title('Hulu tv shows and movies by year added')
plt.legend()
plt.show()
```

```
plt.figure(figsize=(14, 7)) labels=['TV Show', 'Movie']
plt.pie(df3['type'].value_counts().sort_values(),labels=lab
els,explode=[0.1,0.1],autopct='
%1.2f%%',colors=['blue','yellow'], startangle=90)
```

```
plt.title('Type of Disney Plus Content') plt.axis('equal')
plt.show()
```

```
plt.figure(figsize = (15,8)) plt.title('Disney Ratings
Distribution') sns.countplot(x='rating', data = df3)
plt.show()
```

```
plt.figure(figsize = (15,8))
plt.title('Top 10 Genres for Movies in Disney Platform')
df3[df3['type']=="Movie"][['listed_in']].value_counts()[
:10].plot(kind='barh') plt.show()
```

```
plt.figure(figsize = (15,8))
plt.title('Top 10 Genres for TV SHOW in Disney
Platform') df3[df3['type']=="TV
Show"][['listed_in']].value_counts()[:10].plot(kind='barh
',color='blue') plt.show()
```

```
plt.figure(figsize = (15,8))
plt.title('Disney ratings distribution seperated by type of
release') sns.countplot(x='rating', data=df3, hue='type')
plt.show()
```

```
df3['date_added'].value_counts().plot(kind='bar')
```

```
df_tv_show = df3[df3['type']=="TV Show"]
df_tv_show['duration'].value_counts().plot(kind='bar')
```

```
sns.set(style="darkgrid")
ax = sns.countplot(x="type", data=df4, palette="Set2")
ax.set_title(f'Types of Netflix Content', fontsize=15,
fontweight='bold', position=(0.20, 1.03))
```

```
sns.set_style('whitegrid') # plot with grid movie =
df4[df4['type'] == 'Movie']
rating_order = ['G', 'TV-Y', 'TV-G', 'PG', 'TV-Y7', 'TV-
PG', 'PG-13', 'TV-14', 'R', 'NC- 17', 'TV-MA']
movie_rating =
movie['rating'].value_counts()[rating_order] def
rating_barplot(data, title, height, h_lim=None):
fig, ax = plt.subplots(1,1, figsize=(14, 7), dpi=200) if
h_lim :
ax.set_ylim(0, h_lim)
ax.bar(data.index, data, color="#e0e0e0", width=0.52,
edgecolor='black') color = ['green', 'blue', 'orange',
'red']
span_range = [[0, 2], [3, 6], [7, 8], [9, 11]]
```

```

for idx, sub_title in enumerate(['Little Kids', 'Older Kids', 'Teens', 'Mature']): ax.annotate(sub_title, xy=(sum(span_range[idx])/2,height), xytext=(0,0), textcoords='offset points', va="center", ha="center", color="w", fontsize=16, fontweight='bold', bbox=dict(boxstyle='round4', pad=0.4, color=color[idx], alpha=0.6))
ax.axvspan(span_range[idx][0]-0.4,span_range[idx][1]+0.4, color=color[idx], alpha=0.07)
ax.set_title(f'Distribution of {title} Rating', fontsize=15, fontweight='bold', position=(0.20, 1.0+0.03))
fig.tight_layout() plt.show()
rating_barplot(movie_rating, 'Movie', 1200, 2100)

```

```

plt.figure(figsize = (15,8))
plt.title('Netflix country distribution separated by type of release') sns.countplot(x='country', data=df4, hue='type', order=df4.country.value_counts().iloc[:5].index)
plt.show()

```

```

country_counts = df4['country'].value_counts(sort=True) country_df = pd.DataFrame(country_counts)
country_df = country_df.reset_index()
country_df.columns = ['country', 'counts']
country_pie_df = country_df.head(10) plt.figure(figsize = (15,8))
colors = sns.color_palette('pastel')[0:10]
plt.title('Distribution of release by country (top 10)')
plt.pie(country_pie_df['counts'], labels = country_pie_df['country'], colors = colors, autopct='%0.0f%%')
plt.show()

```

```

plt.figure(figsize=(15,8))
movie = df4[df4['type'] == 'Movie' ] tv = df4[df4['type'] == 'TV Show']
movie = movie[movie['date_added']>2000] tv = tv[tv['date_added']>2000]
added_counts= movie['date_added'].value_counts()
added_tv_counts= tv['date_added'].value_counts()
sns.lineplot(x=added_counts.index,y=added_counts.values, color="orange", label='Movie')
sns.lineplot(x=added_tv_counts.index,y=added_tv_counts.values,color="blue", label='TV Show')
plt.title('Netflix tv shows and movies by year added')

```

```
plt.legend() plt.show()
```

```

plt.figure(figsize=(15,8))
dd1 = df1[df1['type'] == 'Movie' ] dd2 = df2[df2['type'] == 'Movie' ] dd3 = df3[df3['type'] == 'Movie' ] dd4 = df4[df4['type'] == 'Movie' ]

```

```

added_counts1= dd1['release_year'].value_counts()
added_counts2= dd2['release_year'].value_counts()
added_counts3= dd3['release_year'].value_counts()
added_counts4= dd4['release_year'].value_counts()
sns.lineplot(x=added_counts1.index,y=added_counts1.values, color="orange", label='Amazon Prime')
sns.lineplot(x=added_counts2.index,y=added_counts2.values, color="blue", label='Hulu')
sns.lineplot(x=added_counts3.index,y=added_counts3.values, color="green", label='Disney')
sns.lineplot(x=added_counts4.index,y=added_counts4.values, color="black", label='Netflix')
plt.title('Movies by release year in major Streaming sites') plt.legend()
plt.show()

```

```

plt.figure(figsize=(15,8))
dt1 = df1[df1['type'] == 'TV Show' ] dt2 = df2[df2['type'] == 'TV Show' ] dt3 = df3[df3['type'] == 'TV Show' ] dt4 = df4[df4['type'] == 'TV Show' ] dt1 = dt1[dt1['release_year']>1980] dt2 = dt2[dt2['release_year']>1980] dt3 = dt3[dt3['release_year']>1980] dt4 = dt4[dt4['release_year']>1980]
added_countst1= dt1['release_year'].value_counts()
added_countst2= dt2['release_year'].value_counts()
added_countst3= dt3['release_year'].value_counts()
added_countst4= dt4['release_year'].value_counts()
sns.lineplot(x=added_countst1.index,y=added_countst1.values, color="orange", label='Amazon Prime')
sns.lineplot(x=added_countst2.index,y=added_countst2.values, color="blue", label='Hulu')
sns.lineplot(x=added_countst3.index,y=added_countst3.values, color="green", label='Disney')
sns.lineplot(x=added_countst4.index,y=added_countst4.values, color="black", label='Netflix')
plt.title('TV SHOWS by release year in major Streaming sites') plt.legend()

```

```
plt.show()
```

```

df1['platform'] = 'Amazon' df2['platform'] = 'Hulu'
df3['platform'] = 'Disney' df4['platform'] = 'Netflix'

```

```

frames = [df1, df2, df3, df4]
result = pd.concat(frames, keys=["Amazon", "Hulu", "Disney", "Netflix"])

```

```

platform_counts = result['platform'].value_counts(sort=True) platform_df = pd.DataFrame(platform_counts)
platform_df = platform_df.reset_index()
platform_df.columns = ['platform', 'counts']
platform_pie_df = platform_df plt.figure(figsize = (15,8))
colors = sns.color_palette('pastel')[0:10]
plt.title('Distribution of release by platforms')

```

```
plt.pie(platform_pie_df['counts'], labels =
platform_pie_df['platform'], colors = colors,
autopct='%0.0f%%')
plt.show()
```

```
text = ' '.join(result['listed_in'])
plt.rcParams['figure.figsize'] = (12,12)
wordcloud = WordCloud(background_color =
'black', colormap='vlag', width = 1200, height = 1200,
max_words = 121).generate(text)
plt.imshow(wordcloud) plt.axis('off') plt.show()
```

```
country_codes = {'afghanistan': 'AFG', 'albania': 'ALB',
'algeria': 'DZA', 'american samoa': 'ASM', 'andorra':
'AND', 'angola': 'AGO', 'anguilla': 'AIA',
'antiguaand barbuda': 'ATG', 'argentina':
'ARG', 'armenia': 'ARM', 'aruba':
'ABW', 'australia': 'AUS', 'austria': 'AUT', 'azerbaijan':
'AZE', 'bahamas': 'BHM',
'bahrain': 'BHR', 'bangladesh': 'BGD', 'barbados':
'BRB', 'belarus': 'BLR', 'belgium':
'BEL', 'belize': 'BLZ', 'benin': 'BEN', 'bermuda':
'BMU', 'bhutan': 'BTN',
'bolivia': 'BOL', 'bosnia and herzegovina':
'BIH', 'botswana': 'BWA', 'brazil':
'BRA', 'british virgin islands': 'VGB', 'brunei':
'BRN', 'bulgaria': 'BGR',
'burkina faso': 'BFA', 'burma':
'MMR', 'burundi': 'BDI', 'cabo
verde':
'CPV', 'cambodia': 'KHM', 'cameroon':
'CMR', 'canada': 'CAN', 'cayman islands':
'CYM',
'central african republic': 'CAF', 'chad': 'TCD', 'chile':
'CHL', 'china': 'CHN', 'colombia': 'COL', 'comoros':
'COM', 'congo democratic': 'COD',
```

```
'Congo republic': 'COG', 'cook islands': 'COK', 'costa
rica': 'CRI', 'cote d'ivoire': 'CIV', 'croatia':
'HRV', 'cuba': 'CUB', 'curacao': 'CUW', 'cyprus': 'CYP',
'czech republic': 'CZE', 'denmark':
'DNK', 'djibouti': 'DJI', 'dominica':
'DMA', 'dominican republic': 'DOM', 'ecuador':
'ECU', 'egypt': 'EGY',
'el salvador': 'SLV', 'equatorial
guinea': 'GNQ', 'eritrea': 'ERI', 'estonia':
'EST', 'ethiopia': 'ETH', 'falkland islands': 'FLK', 'faroe
islands': 'FRO',
'fiji': 'FJI', 'finland': 'FIN', 'france': 'FRA', 'french
polynesia': 'PYF', 'gabon':
'GAB', 'gambia, the': 'GMB', 'georgia': 'GEO', 'germany':
'DEU',
'ghana': 'GHA', 'gibraltar': 'GIB', 'greece':
'GRC', 'greenland': 'GRL', 'grenada':
'GRD', 'guam': 'GUM', 'guatemala': 'GTM', 'guernsey':
'GGY',
'guinea-bissau': 'GNB', 'guinea': 'GIN', 'guyana':
'GUY', 'haiti': 'HTI', 'honduras':
```

```
'HND', 'hong kong': 'HKG', 'hungary': 'HUN', 'iceland':
'ISL', 'india': 'IND',
'indonesia': 'IDN', 'iran': 'IRN', 'iraq': 'IRQ', 'ireland':
'IRL', 'isle of man': 'IMN', 'israel':
'ISR', 'italy': 'ITA', 'jamaica': 'JAM', 'japan':
'JPN', 'jersey': 'JEY',
'jordan': 'JOR', 'kazakhstan': 'KAZ', 'kenya':
'KEN', 'kiribati': 'KIR', 'north korea':
'PRK', 'south korea': 'KOR', 'kosovo': 'KSV', 'kuwait':
'KWT',
'kyrgyzstan': 'KGZ', 'laos': 'LAO', 'latvia':
'LVA', 'lebanon': 'LBN', 'lesotho':
'LSO', 'liberia': 'LBR', 'libya': 'LBY', 'liechtenstein':
'LIE', 'lithuania': 'LTU',
'luxembourg': 'LUX', 'macau': 'MAC', 'macedonia':
'MKD', 'madagascar': 'MDG', 'malawi':
'MWI', 'malaysia': 'MYS', 'maldives': 'MDV', 'mali':
'MLI', 'malta': 'MLT',
'marshall islands': 'MHL', 'mauritania':
'MRT', 'mauritius': 'MUS', 'mexico':
'MEX', 'micronesia': 'FSM', 'moldova': 'MDA', 'monaco':
'MCO', 'mongolia': 'MNG',
'montenegro': 'MNE', 'morocco':
'MAR', 'mozambique': 'MOZ', 'namibia':
'NAM', 'nepal': 'NPL', 'netherlands': 'NLD', 'new
caledonia': 'NCL', 'new zealand': 'NZL',
'nicaragua': 'NIC', 'nigeria': 'NGA', 'niger': 'NER', 'niue':
'NIU', 'northern mariana
islands': 'MNP', 'norway': 'NOR', 'oman':
'OMN', 'pakistan': 'PAK', 'palau': 'PLW',
'panama': 'PAN', 'papua new guinea':
'PNG', 'paraguay': 'PRY', 'peru':
'PER', 'philippines': 'PHL', 'poland': 'POL', 'portugal':
'PRT', 'puerto rico': 'PRI',
'qatar': 'QAT', 'romania': 'ROU', 'russia':
'RUS', 'rwanda': 'RWA', 'saint kitts and nevis':
'KNA', 'saint lucia': 'LCA', 'saint martin': 'MAF',
'saint pierre and miquelon': 'SPM', 'saint vincent and the
grenadines': 'VCT', 'samoa': 'WSM', 'san marino':
'SMR', 'sao tome and principe': 'STP',
'saudi arabia': 'SAU', 'senegal': 'SEN', 'serbia':
'SRB', 'seychelles': 'SYC', 'sierra leone':
'SLE', 'singapore': 'SGP', 'sint maarten':
'SXM', 'slovakia': 'SVK',
'slovenia': 'SVN', 'solomon islands': 'SLB', 'somalia':
'SOM', 'south africa': 'ZAF', 'south
sudan': 'SSD', 'spain': 'ESP', 'sri lanka': 'LKA', 'sudan':
'SDN',
'suriname': 'SUR', 'swaziland': 'SWZ', 'sweden':
'SWE', 'switzerland': 'CHE', 'syria':
'SYR', 'taiwan': 'TWN', 'tajikistan': 'TJK', 'tanzania':
'TZA',
'thailand': 'THA', 'timor-leste': 'TLS', 'togo':
'TGO', 'tonga': 'TON', 'trinidad and
tobago': 'TTO', 'tunisia': 'TUN', 'turkey':
'TUR', 'turkmenistan': 'TKM',
'tuvalu': 'TUV', 'uganda': 'UGA', 'ukraine':
'UKR', 'united arab emirates': 'ARE', 'united kingdom':
'GBR', 'united states': 'USA', 'uruguay': 'URY',
'uzbekistan': 'UZB', 'vanuatu': 'VUT', 'venezuela':
'VEN', 'vietnam': 'VNM', 'virgin
```

```
islands': 'VGB', 'west bank': 'WBG', 'yemen':
'YEM', 'zambia': 'ZMB', 'zimbabwe': 'ZWE'}
```

```
## countries
```

```
from collections import Counter
colorscale = ['#f7fbff', '#ebf3fb', '#deebf7',
'#d2e3f3', '#c6dbef', '#b3d2e9', '#9ecae1',
'#85bcd6', '#6baed6', '#57a0ce', '#4292c6',
'#3082be', '#2171b5', '#1361a9', '#08519c',
'#0b4083', '#08306b'
]

def geoplots(ddf):
country_with_code, country = {}, {}
shows_countries = "",
''.join(result['country'].dropna().split(", ") for c,v in
dict(Counter(shows_countries)).items()):
code = ""
if c.lower() in country_codes: code =
country_codes[c.lower()]
country_with_code[code] = v country[c] = v
```

```
data = [dict(
type = 'choropleth',
locations = list(country_with_code.keys()), z =
list(country_with_code.values()),
colorscale = [[0,"rgb(5, 10, 172)"],[0.65,"rgb(40, 60,
190)"],[0.75,"rgb(70, 100, 245)"],\
[0.80,"rgb(90, 120, 245)"],[0.9,"rgb(106, 137,
247)"],[1,"rgb(220, 220,
220)"]],
autocolorscale = False, reversescale = True, marker =
dict(
line = dict ( color = 'gray', width = 0.5
) ),
colorbar = dict( autotick = False, title = ''),
) ]
```

```
layout = dict( title = "", geo = dict(
showframe = False, showcoastlines = False, projection =
dict(
type = 'Mercator'
)
)
)
)
```

```
fig = dict( data=data, layout=layout )
iplot( fig, validate=False, filename='d3-world-map' )
return country
```

```
country_vals = geoplots(result)
tabs = Counter(country_vals).most_common(25)
```

```
labels = [_[0] for _ in tabs][::-1]
values = [_[1] for _ in tabs][::-1]
trace1 = go.Bar(y=labels, x=values,
orientation="h", name="",
marker=dict(color="#a678de"))
```

```
data = [trace1]
```

```
layout = go.Layout(title="Countries with most content",
height=700, legend=dict(x=0.1, y=1.1, orientation="h"))
fig = go.Figure(data, layout=layout) fig.show()
```

```
result.head()
```

```
def preprocessing(desc): desc = desc.lower()
desc = re.sub('[-=+,#/\?:^$.@%*~&% \.!\]
\\'\(\)\[\]\<\>\'...> ]', ' ', desc) desc = "
".join(desc.split())
```

```
return desc
```

```
result["new_description"] =
result["description"].apply(lambda x: preprocessing(x))
print(result.shape)
result.head()
```

```
from gensim.models.fasttext import FastText as
FT_gensim corpus = result["new_description"].tolist()
sentences = [re.split(' ', str(sentence)) for sentence in
corpus] print(corpus[0])
print(sentences[0])
```

```
embedding_size = 30
```

```
FT_model = FT_gensim(size=embedding_size,
min_count=2, min_n=2, max_n=5, sg=1, negative=10,
sample=0.001, window=5, alpha=0.025,
min_alpha=0.0001) FT_model.build_vocab(sentences)
print('corpus_count: ', FT_model.corpus_count)
print('corpus_total_words: ',
FT_model.corpus_total_words)
```

```
FT_model.train(sentences, epochs=FT_model.epochs,
total_examples=FT_model.corpus_count,
total_words=FT_model.corpus_total_words)
print(FT_model)
```

```
FT_vector = []
```

```
for item in corpus:
FT_vector.append(FT_model.wv[str(item)])
FT_vector = np.asarray(FT_vector)
```

```
from sklearn.cluster import KMeans from
scipy.spatial.distance import cdist
```

```
kmeanModel = KMeans(n_clusters=50,
random_state=42).fit(FT_vector) cluster_id =
kmeanModel.predict(FT_vector)
result["cluster_id"] = cluster_id
```

result.head()

```
def recommendation_system(title_name): top_k = 5
title_row = result[result["title"] == title_name].copy()
search_df =
result[result["cluster_id"].isin(title_row["cluster_id"])].
copy() search_df =
search_df.drop(search_df[search_df["title"] ==
title_name].index)
```

```
search_df["Similarity"] =
search_df.apply(lambda x:
FT_model.wv.similarity(title_row["new_description"],
x["new_description"]), axis=1)
```

```
search_df.sort_values(by=["Similarity"],
ascending=False, inplace=True) return
search_df[["title", "Similarity"]].head(top_k)
recommendation_system("Aakhri Adaalat")
```

recommendation\_system("National Parks Adventure")

recommendation\_system("Secrets of Deception")

## XII. REFERENCES

- Rafael A. Irizarry (2019), [Introduction to Data Science: Data Analysis and Prediction Algorithms with R](#)
- Yixuan Qiu (2017), [recosystem: recommendation System Using Parallel Matrix Factorization](#)
- Michael Hahsler (2019), [recommendationlab: Lab for Developing and Testing recommendation Algorithms. R package version 0.2-5.](#)
- Georgios Drakos, [How to select the Right Evaluation Metric for Machine Learning Models: Part 1 Regression Metrics](#)
- <https://www.edx.org/professional-certificate/harvardx-data-science>
- <https://www.netflixprize.com/>
- <https://www.edx.org/professional-certificate/harvardx-data-science>
- <https://grouplens.org/>
- <https://movielens.org/>

- <https://grouplens.org/datasets/movielens/latest/>
- <https://grouplens.org/datasets/movielens/10m/>
- A Matrix-factorization Library for Recommender Systems <https://www.csie.ntu.edu.tw/~cjlin/libmf/>
- <https://cran.r-project.org/web/packages/recosystem/index.html>
- <https://www.netflixprize.com/>
- <https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html>
- <https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html>
- [https://cran.r-project.org/web/packages/available\\_packages\\_by\\_name.html](https://cran.r-project.org/web/packages/available_packages_by_name.html)