Problem Statement —

What kind of movies and TV shows can be added to increase engagement, viewership and subscription?

Report Structure

This report is divide in to 7 sections —

1. Preparing Data
2. Basic Metrics
3. Data Cleaning
4. Graphical Summary
5. Exploration of Categorical columns.
6. Business Insights
7. Recommendations

Please note every section started from new page and, a section can span multiple pages.

Preparing Data

```
!gdown https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/000/940/original/netflix.csv
```

```
Downloading...
From: https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/000/940/original/netflix.csv
To: /content/netflix.csv
100% 3.40M/3.40M [00:00<00:00, 19.5MB/s]
```

```
import numpy as np
import pandas as pd
from pandas.api.types import CategoricalDtype
import matplotlib.pyplot as plt
import seaborn as sns
```

```
netflix = pd.read_csv('netflix.csv')
```

Basic metrics

In this section, we'll try to identify —

1. Number of rows and columns of data.
2. Data type of each column, non-null values in each column and memory usage by the dataset.
3. How data looks like, by taking sample of 5 rows out of it.
4. Null values percentage in each column.
5. If dataset contains duplicated rows.

```
#Number of rows and columns of data.
netflix.shape
```

```
(8807, 12)
```

```
# Data type of each column, non-null values in each column and memory usage by the dataset.
netflix.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8807 entries, 0 to 8806
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   show_id       8807 non-null   object
 1   type          8807 non-null   object
 2   title         8807 non-null   object
 3   director      6173 non-null   object
 4   cast          7982 non-null   object
 5   country       7976 non-null   object
 6   date_added    8797 non-null   object
 7   release_year  8807 non-null   int64 
 8   rating        8803 non-null   object
 9   duration      8804 non-null   object
 10  listed_in     8807 non-null   object
 11  description   8807 non-null   object
dtypes: int64(1), object(11)
memory usage: 825.8+ KB
```

```
# How data looks like, by taking sample of 5 rows out of it.
netflix.sample(5)
```

	show_id	type	title	director	cast	country	date_added	release_year	rating	duration	genre	description
1398	s1399	Movie	Death of Me	Darren Lynn Bousman	Maggie Q, Luke Hemsworth, Alex Essoe, Ingkarat...	United States, Thailand	2021-01-16	2020	R	94	Horror Movies	With no memory of the previous night, a vacati...
4865	s4866	TV Show	Bill Nye Saves the World	NaN	Bill Nye, Karlie Kloss, Derek Muller, Emily Ca...	United States	2018-05-11	2018	TV-14	3	Stand-Up Comedy & Talk Shows	Emmy-winning host Bill Nye brings experts and ...
8582	s8583	Movie	Thorne: Scaredy Cat	Benjamin Ross	David Morrissey, Eddie Marsan, Aidan Gillen, O...	United Kingdom	2016-11-02	2010	NR	125	Dramas, International Movies	Heading a new team whose aim is to crack the c...
			Stars in									This

```
# Null values percentage in each column.
netflix.isna().sum() / netflix.shape[0] * 100
```

	0
show_id	0.000000
type	0.000000
title	0.000000
director	29.908028
cast	9.367549
country	9.435676
date_added	0.113546
release_year	0.000000
rating	0.045418
duration	0.034064
listed_in	0.000000
description	0.000000

dtype: float64

```
# If dataset contains duplicated rows.
netflix.duplicated().sum()
```

```
0
```

Summary

`30%` of the movies are missing the **director**, `9.36%` of the movies are missing the **cast** and `9.43%` are missing **country** of origin and `0%` in **listed_in**.

Please note these are the percentage of ***movies*** which are missing the values, as these values will change when we normalize the data.

Data Cleaning

In this section —

1. Find categorical columns and convert them.
2. Convert the duration column to Integer.
3. Fill `NaN` values in `rating` and `release_year` column with `mode` value of that column.

```
# Nominal Variables – Country, Type, Genre
netflix['type'] = netflix['type'].astype('category')

# Ordinal – Rating
ratings_order = [
    'NC-17', 'R', 'TV-MA', 'UR', 'NR', 'PG-13', 'TV-14',
    'PG', 'TV-PG', 'TV-Y7-FV', 'TV-Y7', 'TV-G', 'G', 'TV-Y'
]
rating_order = CategoricalDtype(categories=ratings_order, ordered=True)
netflix['rating'] = np.where(netflix['rating'].str.contains('min'), 'PG-13', netflix['rating'])
netflix['rating'] = netflix['rating'].astype(rating_order)

# Rename the listed_in column to genre.
netflix.rename(columns={'listed_in': 'genre'}, inplace=True)

# Datetime
netflix['date_added'] = pd.to_datetime(netflix['date_added'], format = "mixed")
```

```
# Convert the duration column to Integer.
netflix['duration'] = netflix['duration'].str.extract('(\d+)')[0].astype('Int64')
```

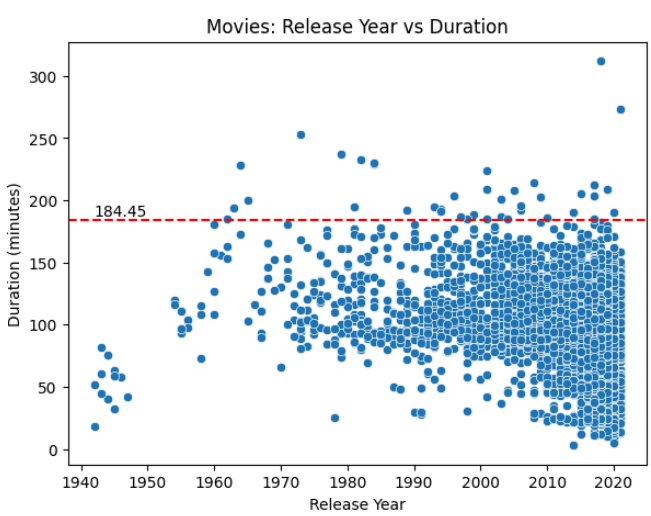
```
netflix[netflix['type'] == 'Movie']['duration'].describe().reset_index().T
```

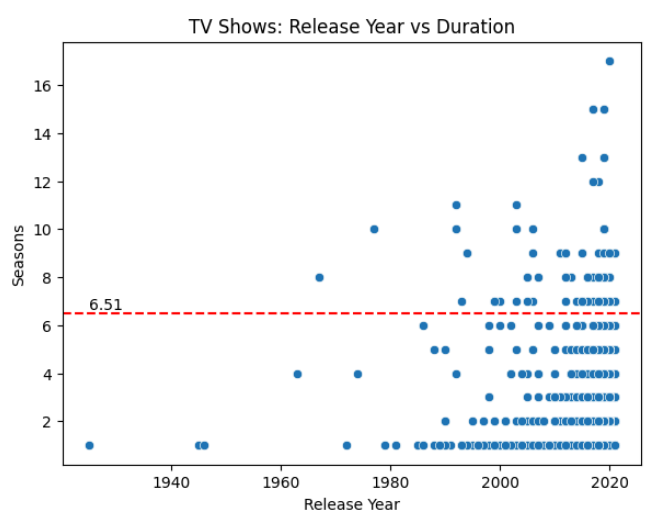
	0	1	2	3	4	5	6	7
index	count	mean	std	min	25%	50%	75%	max
duration	6128.0	99.577187	28.290593	3.0	87.0	98.0	114.0	312.0

```
netflix[netflix['type'] == 'TV Show']['duration'].describe().reset_index().T
```

	0	1	2	3	4	5	6	7
index	count	mean	std	min	25%	50%	75%	max
duration	2676.0	1.764948	1.582752	1.0	1.0	1.0	2.0	17.0

Detecting and Treating Outliers for *duration*.





From the above scatter plots —

- Movies started to deviate as the duration cross 180 minutes mark, wheareas an average length of movies is about an hour and a half.
- TV Shows started to deviate as number of seasons increases from around 6 or 7.

Here we'll treat outliers for the movies with the average length of the movies and, outliers for the TV shows with standard deviation of the seasons. *This decision best reflects the current trend of film industry.*

```
# Fill NaN values with mode value of the columns with less than 5% of missing data.
for column in ['rating', 'date_added']:
  netflix[column].fillna(netflix[column].mode()[0], inplace=True)

netflix['rating'] = netflix['rating'].apply(lambda x: netflix['rating'].mode()[0] if 'min' in x else x)
```

```
# Treating Outliers and NaN values in duration column based on it's TYPE.
from math import ceil

# Creating masks
movie_mask = netflix['type'] == 'Movie'
tv_show_mask = netflix['type'] == 'TV Show'

# Create separate DataFrames for movies and TV shows
movies = netflix[movie_mask]
tv_shows = netflix[tv_show_mask]

# Calculate the mean and standard deviation of duration for movies and TV shows
movie_duration_mean = movies['duration'].mean(skipna = True)
movie_duration_std = movies['duration'].std(skipna = True)
tv_show_duration_mean = tv_shows['duration'].mean(skipna = True)
tv_show_duration_std = tv_shows['duration'].std(skipna = True)
tv_show_fill = tv_show_duration_mean + 3 * tv_show_duration_std
movie_fill = movie_duration_mean + 3 * movie_duration_std

# Treating Outliers or NaNs
movie_outliers_or_nans = (movie_mask & (netflix['duration'] > movie_fill)) | (netflix['duration'].isna())
tv_show_outliers_or_nans = (tv_show_mask & (netflix['duration'] > tv_show_fill)) | (netflix['duration'].isna())

netflix['duration'] = np.where(movie_outliers_or_nans, ceil(movie_duration_mean), netflix['duration'])
netflix['duration'] = np.where(tv_show_outliers_or_nans, ceil(tv_show_fill) + 1, netflix['duration'])
netflix['duration'] = netflix['duration'].astype('Int64')
```

```
netflix.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8807 entries, 0 to 8806
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   show_id       8807 non-null   object
 1   type          8807 non-null   category
 2   title         8807 non-null   object
 3   director      6173 non-null   object
 4   cast          7982 non-null   object
 5   country       7976 non-null   object
 6   date_added    8807 non-null   datetime64[ns]
 7   release_year  8807 non-null   int64
 8   rating        8807 non-null   category
 9   duration      8807 non-null   Int64
 10  genre         8807 non-null   object
 11  description   8807 non-null   object
dtypes: Int64(1), category(2), datetime64[ns](1), int64(1), object(7)
memory usage: 714.7+ KB
```

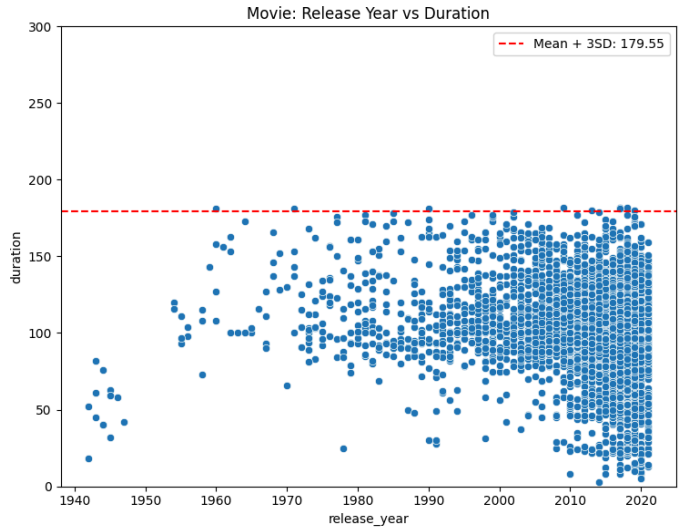
```
# Verifying the changes
netflix[netflix['type'] == 'Movie']['duration'].describe().reset_index().T
```

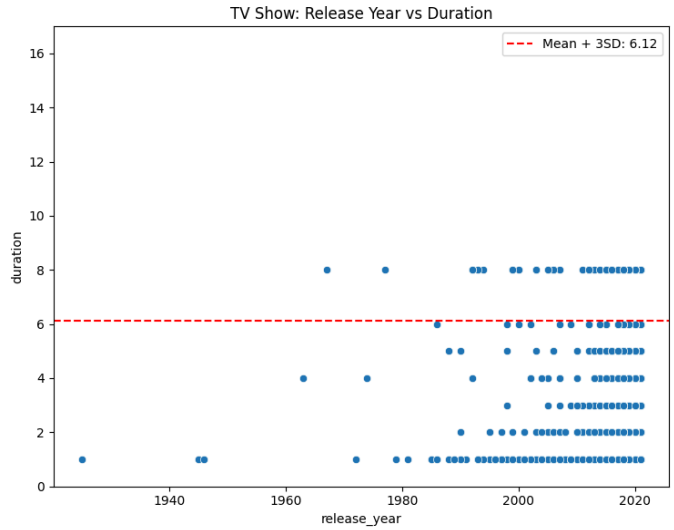
	0	1	2	3	4	5	6	7
index	count	mean	std	min	25%	50%	75%	max
duration	6131.0	98.8351	26.90543	3.0	87.0	98.0	114.0	182.0

```
# Verifying the changes
netflix[netflix['type'] == 'TV Show']['duration'].describe().reset_index().T
```

	0	1	2	3	4	5	6	7
index	count	mean	std	min	25%	50%	75%	max
duration	2676.0	1.745516	1.458164	1.0	1.0	1.0	2.0	8.0

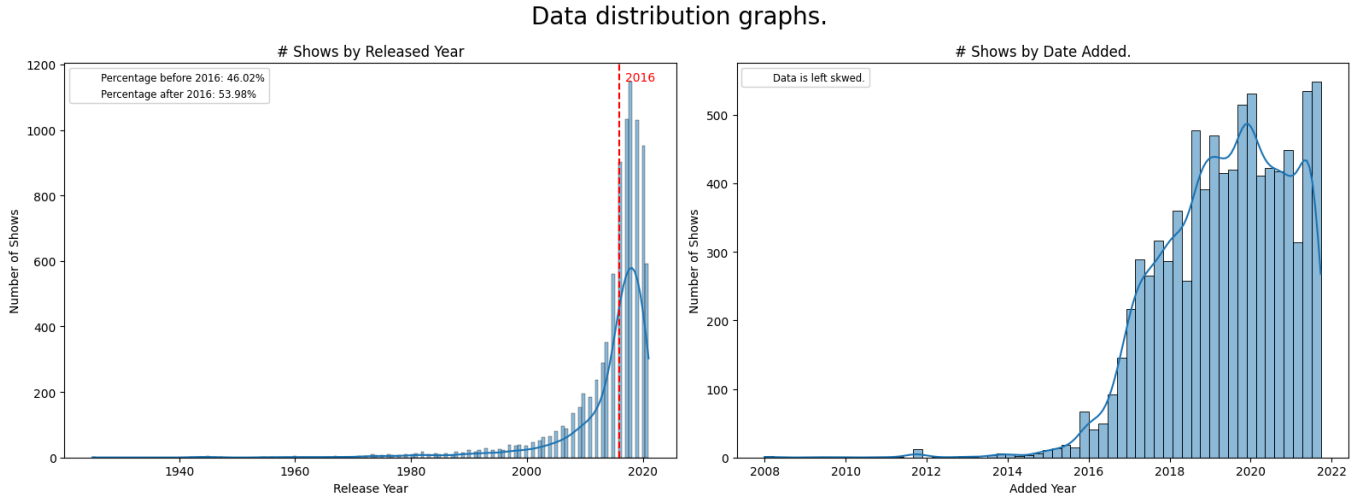
Numerical summary above and Scatter plot shows the Data Distribution





Graphical Summary

Data distribution Summary based on Release Year and Date Added —




```
netflix[['date_added', 'release_year']].describe().reset_index().round(2)
```

	index	date_added	release_year
0	count	8807	8807.00
1	mean	2019-05-17 12:13:09.735437824	2014.18
2	min	2008-01-01 00:00:00	1925.00
3	25%	2018-04-06 00:00:00	2013.00
4	50%	2019-07-04 00:00:00	2017.00
5	75%	2020-08-18 00:00:00	2019.00
6	max	2021-09-25 00:00:00	2021.00
7	std	NaN	8.82

```
netflix['release_year'].mode()
```

	release_year
0	2018

dtype: int64

From the above graphical and non-graphical summary we can see that,

- *Netflix* started to add shows to its library from `2008` and it took *10 years* to acquire `25%` of shows, from then on there is a subtle increase of `25%` every year for 4 consecutive years.
- The above hypothesis is also supported by graph for release year and `mode` (maximum frequency count) value of the release year (`2018`), i.e, this dataset contains maximum number of shows from `2018`.

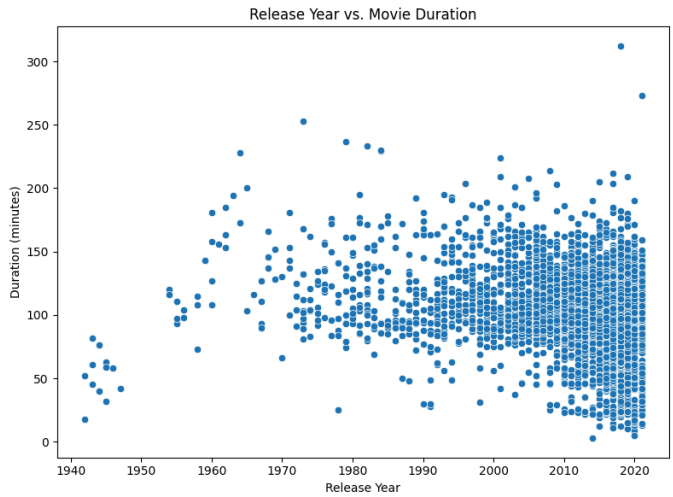
duration summary based on raw data —

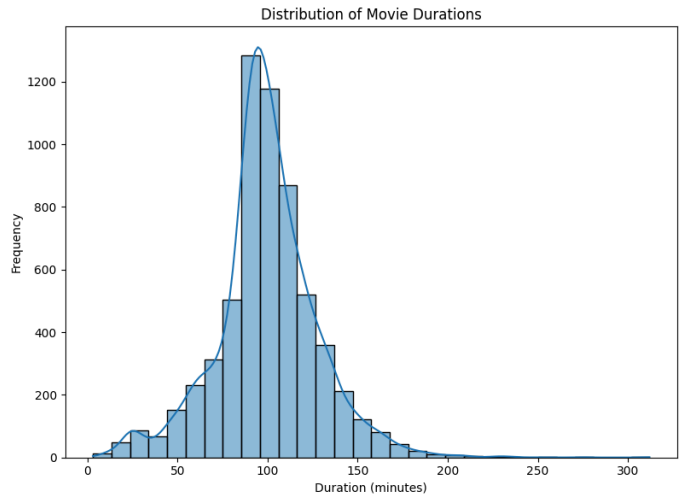
```
netflix.groupby('type', observed = True).describe()['duration'].reset_index().round(2)
```

	type	count	mean	min	25%	50%	75%	max	std
0	Movie	6128.0	99.58	3.0	87.0	98.0	114.0	312.0	28.29
1	TV Show	2676.0	1.76	1.0	1.0	1.0	2.0	17.0	1.58

Movies

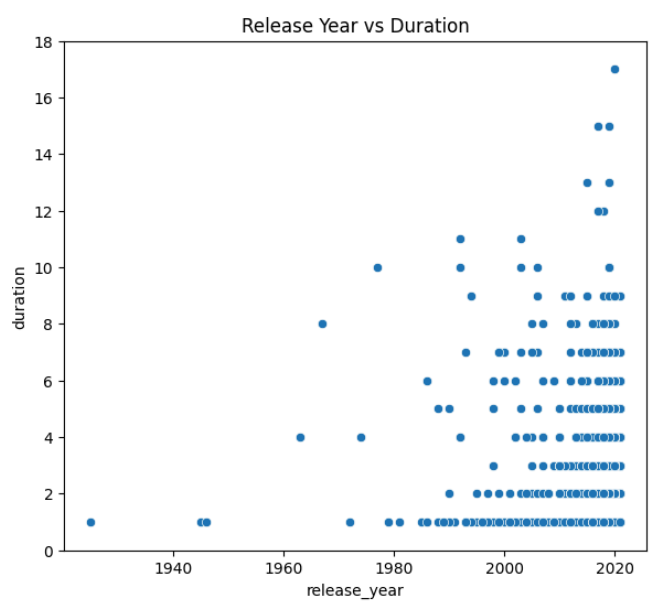
- `Average` movie length is *an hour and a half*.
- `minimum` length of 1 minute and `maximum` length of *6 hours*.

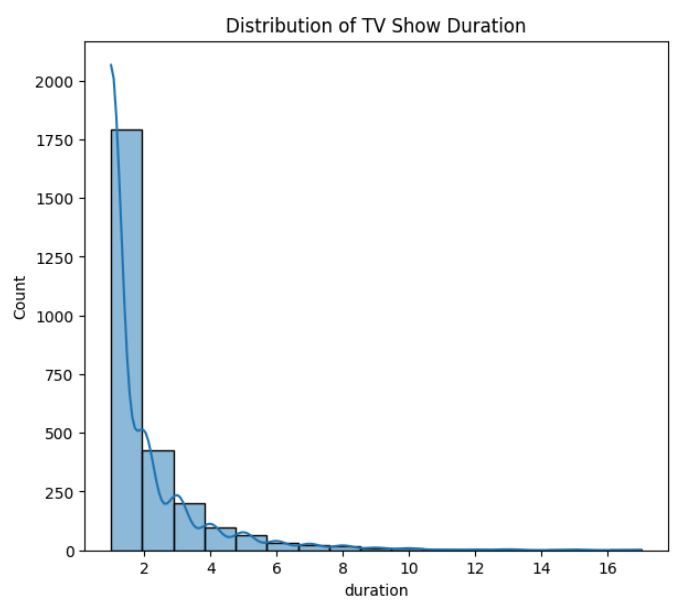




TV Shows

- `Average` TV season length is around *two*.
- `minimum` of 1 season and `maximum` of *17 seasons*.

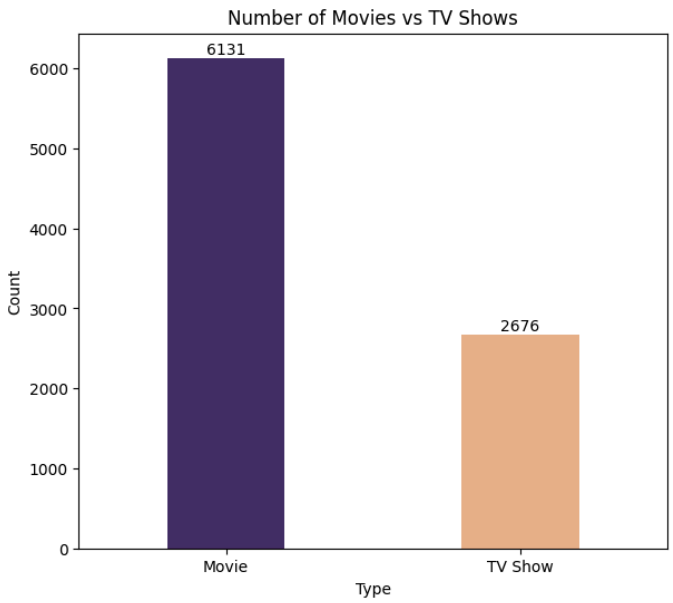


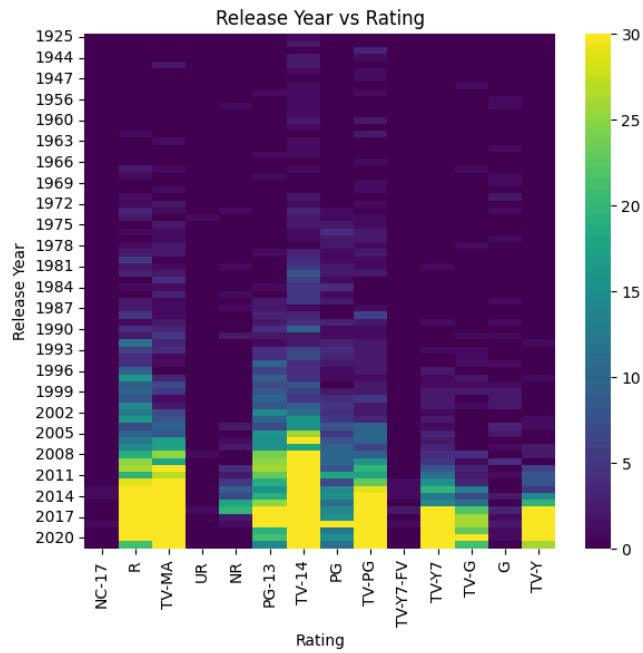


Summary based on **Type** and **Rating** —

```
netflix['rating'].value_counts().reset_index().T
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
rating	TV-MA	TV-14	TV-PG	R	PG-13	TV-Y7	TV-Y	PG	TV-G	NR	G	TV-Y7-FV	NC-17	UR
count	3207	2160	863	799	497	334	307	287	220	80	41	6	3	3





- Netflix has around 6k movies which makes `70%` of its total libray and, around 2.5k TV Shows which makes `30%` of its total library.
- The Heatmap shows how movies are made over the years across different ratings.

Exploration of other **Categorical** columns —

Here we'll explore `cast`, `director`, `genre` and `country` columns.

```
# Defining new dataset
stacked = netflix.copy()

# Splitting Cast, Director, Genere and Country
stacked['cast'] = stacked['cast'].str.split(',')
stacked['director'] = stacked['director'].str.split(',')
stacked['genre'] = stacked['genre'].str.split(',')
stacked['country'] = stacked['country'].str.split(',')

# Stacking lists to rows
stacked = stacked.explode('cast').explode('director').explode('genre').explode('country')

stacked.reset_index(drop=True, inplace=True)
```

Basic metrics

In this section, we'll try to identify —

1. Number of rows and columns of data.
2. Data type of each column, non-null values in each column and memory usage by the dataset.
3. How data looks like, by taking sample of 10 rows out of it.
4. Null values percentage in each column.
5. If dataset contains duplicated rows.

```
# Number of rows and columns of data.
stacked.shape
```

```
(202065, 12)
```

```
# Data type of each column, non-null values in each column and memory usage by the dataset.
stacked.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 202065 entries, 0 to 202064
Data columns (total 12 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   show_id       202065 non-null  object
 1   type          202065 non-null  category
 2   title         202065 non-null  object
 3   director      151422 non-null  object
 4   cast          199916 non-null  object
 5   country       190168 non-null  object
 6   date_added    202065 non-null  datetime64[ns]
 7   release_year  202065 non-null  int64
 8   rating        202065 non-null  category
 9   duration      202065 non-null  Int64
 10  genre         202065 non-null  object
 11  description   202065 non-null  object
dtypes: Int64(1), category(2), datetime64[ns](1), int64(1), object(7)
memory usage: 16.0+ MB
```

```
# How data looks like, by taking sample of 5 rows out of it.
stacked.head(5)
```

	show_id	type	title	director	cast	country	date_added	release_year	rating	duration	genre	description
0	s1	Movie	Dick Johnson Is Dead	Kirsten Johnson	NaN	United States	2021-09-25	2020	PG-13	90	Documentaries	As her father nears the end of his life, filmm...
1	s2	TV Show	Blood & Water	NaN	Ama Qamata	South Africa	2021-09-24	2021	TV-MA	2	International TV Shows	After crossing paths at a party, a Cape Town t...
2	s2	TV Show	Blood & Water	NaN	Ama Qamata	South Africa	2021-09-24	2021	TV-MA	2	TV Dramas	After crossing paths at a party, a Cape Town t...
3	s2	TV Show	Blood & Water	NaN	Ama Qamata	South Africa	2021-09-24	2021	TV-MA	2	TV Mysteries	After crossing paths at a party, a Cape Town t...

```
# Null values percentage in each column.
stacked.isna().sum() / stacked.shape[0] * 100
```

	0
show_id	0.000000
type	0.000000
title	0.000000
director	25.062727
cast	1.063519
country	5.887709
date_added	0.000000
release_year	0.000000
rating	0.000000
duration	0.000000
genre	0.000000
description	0.000000

dtype: float64

```
# If dataset contains duplicated rows.
stacked.duplicated().sum()
```

```
7
```

Summary

As we can see from the above statistics the *percentages* has changed from the inital statistics, and they no longer corresponds `%` w.r.t. number of movies.

Note — *`stacked` dataframe is as normalized as a `netflix` dataframe / data can get. But we'll not perform any operations on this, because data fetching is costly this way.*

Exploration

In this section we'll consider some iteresting question about the `cast`, `director`, `genre` and `country` of the movies.

Questions —

1. Actors / Actress worked in least and most shows.
2. Directors with most and least shows.
3. Most worked actors in a year.
4. Shows by country.
5. Cast / Director and movie ratings.
6. How shows are made based on genre?
7. Most frequently worked Cast + Director duo.
8. Most frequently worked Cast + Cast duo.
9. Famous genre of a country.

Note: *Tie-breaks will happen lexicographically.*

```
# Defining new dataset
stacked = netflix.copy()

# Splitting Cast, Director, Genere and Country
stacked['cast'] = stacked['cast'].str.split(',')
stacked['director'] = stacked['director'].str.split(',')
stacked['genre'] = stacked['genre'].str.split(',')
stacked['country'] = stacked['country'].str.split(',')

# Renaming the show_id column to show_count
stacked.rename(columns={'show_id': 'show_count'}, inplace=True)
```

Actors / Actress worked in least and most movies.

```
# Stacking Cast column
cast = stacked.explode('cast')
cast['cast'] = cast['cast'].str.strip()

# Number of shows worked by a cast member
cast.groupby('cast')['show_count'].unique().transform(lambda x: len(x)).sort_values(ascending=False).reset_index().iloc[[0, -1]]
```

	cast	show_count
0	Anupam Kher	43
36438	Ṣọpẹ́ Dìrísù	1

Most worked actor in a year.

```
# Number of shows worked by a cast member in a single year.
year_popularity = cast.groupby(['cast', 'release_year'])['show_count'].unique()\
                  .transform(lambda x: len(x)).reset_index().sort_values(by=['show_count', 'cast'], ascending=[False, False])
year_popularity['cast'] = year_popularity['cast'].str.strip()

# In case of a tie select lexicographically first cast member.
year_popularity.groupby('release_year').agg({'cast': 'first', 'show_count': 'max'}).sort_index(ascending=False).reset_index()
```

	release_year	cast	show_count
0	2021	Fortune Feimster	11
1	2020	Keith Wickham	6
2	2019	Vincent Tong	8
3	2018	Andrea Libman	8
4	2017	Craig Sechler	6
...	...	...	...
67	1947	Lloyd Bridges	1
68	1946	Walter Huston	1
69	1945	Walter Huston	1
70	1944	Burgess Meredith	1
71	1942	Jane Darwell	1

72 rows × 3 columns

Probably Jane Darwell and Henry Fonda were the first few actors to work in a commercial motion picture.

Director with least and most shows.

```
# Stacking director column
directors = stacked.explode('director')
directors['director'] = directors['director'].str.strip()

# Number of shows directed by a director
directors.groupby('director')['show_count'].unique()\
            .transform(lambda x: len(x)).reset_index().sort_values(by=['show_count', 'director'], ascending=[False, False]).iloc[[-1, 0]]
```

	director	show_count
1	A. Raajdheep	1
3749	Rajiv Chilaka	22

Shows by country.

```
# Stacking country column.
country = stacked.explode('country')
country['country'] = country['country'].str.strip()

# Number of movies from country
country.groupby('country')['show_count'].unique().transform(lambda x: len(x)).reset_index()\
        .sort_values(by=['show_count', 'country'], ascending=[False, False]).iloc[[0, 1, 2, -1]]
```

	country	show_count
116	United States	3690
46	India	1046
115	United Kingdom	806
1	Afghanistan	1

Shows count from top 3 countries —

1. United States
2. India
3. United Kingdom

Cast / Director and show ratings.

We'll work with two question here, for both Cast and Director —

- Cast
  - As *Anupam Kher* has worked in most shows, we'll find what are his shows' ratings.
  - Then we'll look at, in every rating who is the top cast.
- Directors
  - As *Rajiv Chilaka* has directed most shows, we'll find what are his shows' ratings.
  - Then we'll look at, in every rating who is the top director.

Cast

```
cast_and_ratings = cast.groupby(['cast', 'rating'], observed=False)['show_count'].unique().dropna()\
                    .transform(lambda x: len(x)).reset_index().sort_values(by=['show_count', 'cast'], ascending=[False, False])

# Anupam Kher's shows distribution across ratings.
cast_and_ratings[cast_and_ratings['cast'] == 'Anupam Kher']
```

	cast	rating	show_count
3647	Anupam Kher	TV-14	28
3646	Anupam Kher	TV-MA	6
3649	Anupam Kher	TV-PG	6
3645	Anupam Kher	R	2
3648	Anupam Kher	PG	1

```
# Top cast in every rating.
cast_and_ratings.groupby('rating', observed = False).agg({'cast': 'first', 'show_count': 'max'})\
                .sort_values(by='rating', ascending=False).reset_index()
```

	rating	cast	show_count
0	TV-Y	Andrea Libman	21
1	G	Dom DeLuise	5
2	TV-G	David Attenborough	6
3	TV-Y7	Julie Tejwani	29
4	TV-Y7-FV	Yuri Lowenthal	1
5	TV-PG	David Attenborough	12
6	PG	Sylvester Stallone	6
7	TV-14	Anupam Kher	28
8	PG-13	Pierce Brosnan	10
9	NR	Russell Peters	4
10	UR	Rob Schneider	1
11	TV-MA	Takahiro Sakurai	17
12	R	Bruce Willis	15
13	NC-17	Tom Green	1

Director

```
director_and_ratings = directors.groupby(['director', 'rating'], observed=False)['show_count'].unique().dropna()\
                    .transform(lambda x: len(x)).reset_index().sort_values(by=['show_count', 'director'], ascending=[False, False])

# Rajiv Chilaka's shows distribution across ratings.
director_and_ratings[director_and_ratings['director'] == 'Rajiv Chilaka']
```

	director	rating	show_count
4271	Rajiv Chilaka	TV-Y7	20
4272	Rajiv Chilaka	TV-Y	2

```
# Top director in every rating.
director_and_ratings.groupby('rating', observed = False).agg({'director': 'first', 'show_count': 'max'})\
                    .sort_index(ascending=False).reset_index()
```

	rating	director	show_count
0	TV-Y	Joey So	6
1	G	Simon Wells	2
2	TV-G	Lucas Margutti	3
3	TV-Y7	Rajiv Chilaka	20
4	TV-Y7-FV	Suhas Kadav	1
5	TV-PG	Riri Riza	4
6	PG	Robert Rodriguez	7
7	TV-14	Umesh Mehra	8
8	PG-13	Steven Spielberg	5
9	NR	Jerry Rothwell	2
10	UR	Walerian Borowczyk	1
11	TV-MA	Jan Suter	20
12	R	Quentin Tarantino	8
13	NC-17	Warren P. Sonoda	1

How shows are made based on genre?

```
# Stacking genre column
genre = stacked.explode('genre')
genre['genre'] = genre['genre'].str.strip()

# Show count by genre
genre.groupby('genre')['show_count'].unique().transform(lambda x: len(x)).sort_values(ascending=False)\
     .reset_index().iloc[np.r_[0:5, -15:0]]
```

	genre	show_count
0	International Movies	2752
1	Dramas	2427
2	Comedies	1674
3	International TV Shows	1351
4	Documentaries	869
27	Classic Movies	116
28	LGBTQ Movies	102
29	TV Mysteries	98
30	Science & Nature TV	92
31	TV Sci-Fi & Fantasy	84
32	TV Horror	75
33	Anime Features	71
34	Cult Movies	71
35	Teen TV Shows	69
36	Faith & Spirituality	65
37	TV Thrillers	57
38	Movies	57
39	Stand-Up Comedy & Talk Shows	56
40	Classic & Cult TV	28
41	TV Shows	16

- Genre with most shows — International Movies → 2752
- Genre with least shows — Classic & Cult TV → 28

Also, some other observations —

1. *Stand-Up Comedy & Talk Shows*, *Faith & Spirituality*, *Anime Features*, *LGBTQ Movies* stands 3rd, 6th, 9th and 14th from last.
2. There are only two lables *Spanish-Language TV Shows* and *Korean TV Shows* which identifies the language of the show, better cataloging can help to showcase the diversity in Netflix library.

Most frequently worked Cast + Director duo.

```
# Stacking cast and directors.
cast_and_director_duo = cast.explode('director')
cast_and_director_duo['cast'] = cast_and_director_duo['cast'].str.strip()
cast_and_director_duo['director'] = cast_and_director_duo['director'].str.strip()

# Most frequently worked Cast + Director duo
cast_and_director_duo.groupby(['cast', 'director'])['show_count'].unique().transform(lambda x: len(x)).reset_index() \
                  .sort_values(by=['show_count', 'cast'], ascending=[False, False]).iloc[0]
```

	36158
cast	Rajesh Kava
director	Rajiv Chilaka
show_count	19

dtype: object

Most frequently worked Cast + Cast duo.

```
# Merging cast with cast
cast_and_cast_duo = cast.merge(cast, how='inner', on='show_count', suffixes=('_left', '_right'))[['show_count', 'cast_left', 'cast_right']]
cast_and_cast_duo = cast_and_cast_duo[cast_and_cast_duo['cast_left'] != cast_and_cast_duo['cast_right']]

# Dropping NaN values
cast_and_cast_duo.dropna(inplace = True)

# Most frequently worked Cast + Cast duo
cast_and_cast_duo.groupby(['cast_left', 'cast_right'])['show_count'].unique()\
                  .transform(lambda x: len(x)).reset_index().sort_values(by='show_count', ascending=False).iloc[0]
```

	461888
cast_left	Rupa Bhimani
cast_right	Julie Tejwani
show_count	31

dtype: object

Famous genre of a country.

```
# Stacking country on genre
genre_and_country = genre.explode('country')

genre_and_country['country'] = genre_and_country['country'].str.strip()

# Famous genre of a country
genre_and_country.groupby(['country', 'genre'])['show_count'].unique()\
                  .transform(lambda x: len(x)).reset_index().sort_values(by=['show_count', 'country'], ascending=[False, False]).iloc[0]
```

	526
country	India
genre	International Movies
show_count	864

dtype: object

Business Insights

- There are handful of shows from `1942` to `1978`.
- There are `70%` Movies as compared to `30%` TV Shows.
- Only *two* foreign language genre, *Spanish* and *Korean* are catecategorized, given the fact that `46%` of the Movies and TV Shows are of *International* category.
- Categorizing *sequels* is also very important which is also not present in data.

Recommendations

1. Netflix can add more Movies and TV Shows in categories like *Stand-Up Comedy & Talk Shows*, *Faith & Spirituality*, *Anime Features* and *LGBTQ Movies* to attract all kind of audience, which goes to other streaming services to find the same content.
2. Identify and label the native language of every show.

These actions will encourage adults to buy family plan, because every age group, *young* and *old* will have something to watch.

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.