IDVE_EXAM_Q1

December 10, 2021

Riekert Holder 2517888 Question 1

[68]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[69]:
```
from IPython.display import Javascript
display(Javascript('''google.colab.output.setIframeHeight(0, true, {maxHeight:␣
↪8000})'''))
```

```
<IPython.core.display.Javascript object>
```

[70]:
```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import missingno
from mlxtend.preprocessing import TransactionEncoder
import plotly.express as px
```

0.1 1.1 Exploration of Netflix Data

0.1.1 1.1.1 Describe the dataset

[71]:
```
netflix_df = pd.read_csv("/content/drive/MyDrive/IDVE_Exam/netflix_titles.csv")
n_df = netflix_df.copy()
netflix_df.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
```

[72]:
```
netflix_df.nunique()
```

[72]:
```
show_id         8807
type               2
title           8807
director        4528
cast            7692
country          748
date_added      1767
release_year      74
rating            17
duration         220
listed_in        514
description     8775
dtype: int64
```

[73]:
```
netflix_df.describe()
```

[73]:
```
       release_year
count   8807.000000
mean    2014.180198
std        8.819312
min     1925.000000
25%     2013.000000
50%     2017.000000
75%     2019.000000
max     2021.000000
```

Dataset description: * 8807 entries with 12 columns/features * We have 2 unique types, Movies or TV Shows * 8807 unique content titles * 17 categories of ratings * Our content released_date has a range of 96 years

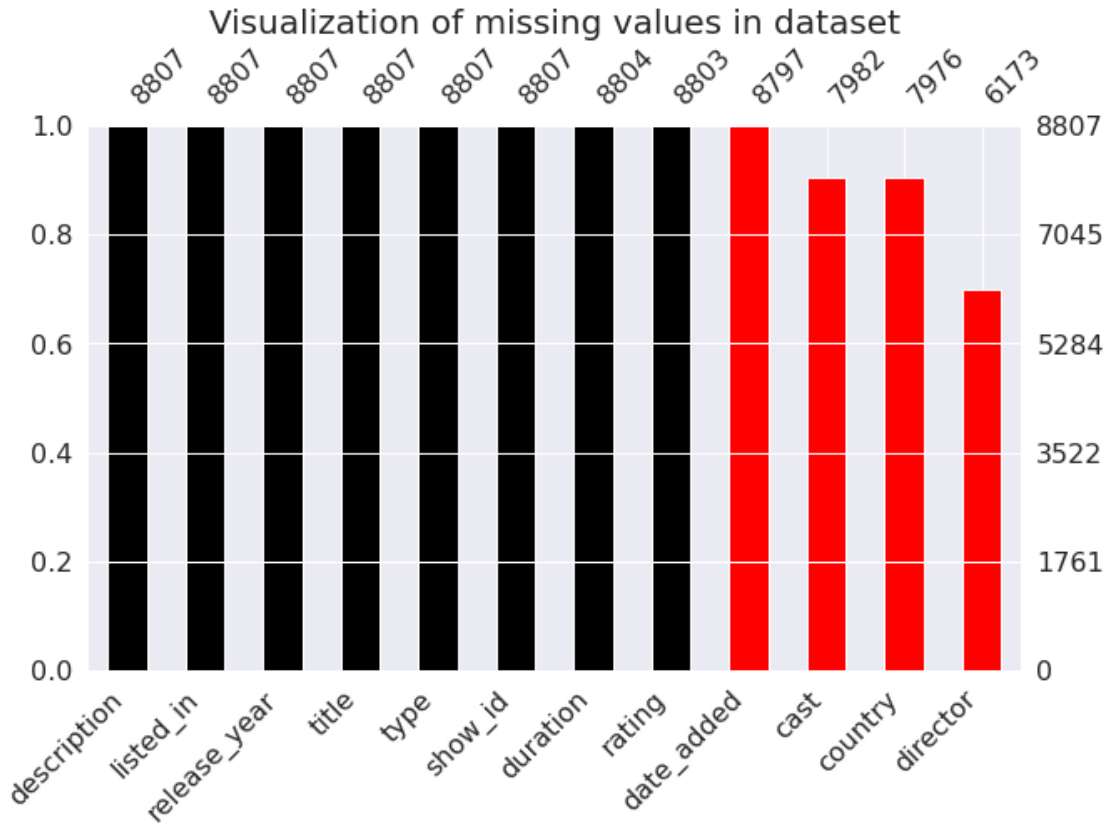
0.1.2 1.1.2 Identify missing values and duplicates

Missing values

[74]:
```
netflix_df.isnull().sum()
```

[74]:
```
show_id          0
type             0
title            0
director      2634
cast           825
country        831
date_added      10
release_year     0
rating           4
duration         3
listed_in        0
description      0
dtype: int64
```

[75]:
```
color =␣
 ↪['black','black','black','black','black','black','black','black','red','red','red','red']
missingno.bar(netflix_df, color = color, sort = 'descending', figsize = (10,6))
plt.title("Visualization of missing values in dataset",fontsize=20)
plt.show()
```



We have 4307 missing entries, the missing values are in the columns:

- **director - 2634 missing values**
- **cast - 825 missing values**
- **country - 831 missing values**
- **date_added - 10 missing dates**
- **rating - 4 missing values**
- **duration - 3 missing values**

Duplicates

[76]:
```
netflix_df.duplicated().sum()
```

[76]:
```
0
```

There is no duplicated data in the dataset

We drop all nan values except for director, cast and country because they have more than 10% missing data.We will also drop show_id because it contains no useful data

[77]:
```
#netflix_df['director'].replace(np.nan,'NO DATA',inplace=True)
#netflix_df['cast'].replace(np.nan,'NO DATA',inplace=True)
#netflix_df['country'].replace(np.nan,'NO DATA',inplace=True)

netflix_df.dropna(subset=["date_added"], inplace=True)
netflix_df.dropna(subset=["rating"], inplace=True)
netflix_df.dropna(subset=["duration"], inplace=True)
```

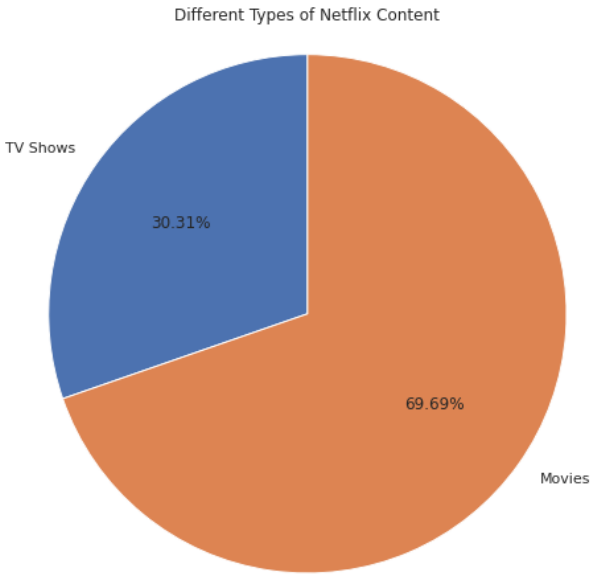
[78]:
```
netflix_df.isna().sum()
```

[78]:
```
show_id            0
type               0
title              0
director        2621
cast             825
country          829
date_added         0
release_year       0
rating             0
duration           0
listed_in          0
description        0
dtype: int64
```

0.1.3 1.1.4 Split between movies and TV shows

[79]:
```
plt.figure(figsize=(15,8))
label=['TV Shows', 'Movies']
plt.pie(netflix_df['type'].value_counts().sort_values(), labels=label,
        autopct='%1.2f%%', startangle=90)
plt.title('Different Types of Netflix Content')
plt.axis('equal')
```

[79]:
```
(-1.1136604528891605,
 1.111690438440619,
 -1.1167378672922192,
 1.1007970412996295)
```



We can see from the graph above that most of netflix's content are of the type 'Movie'

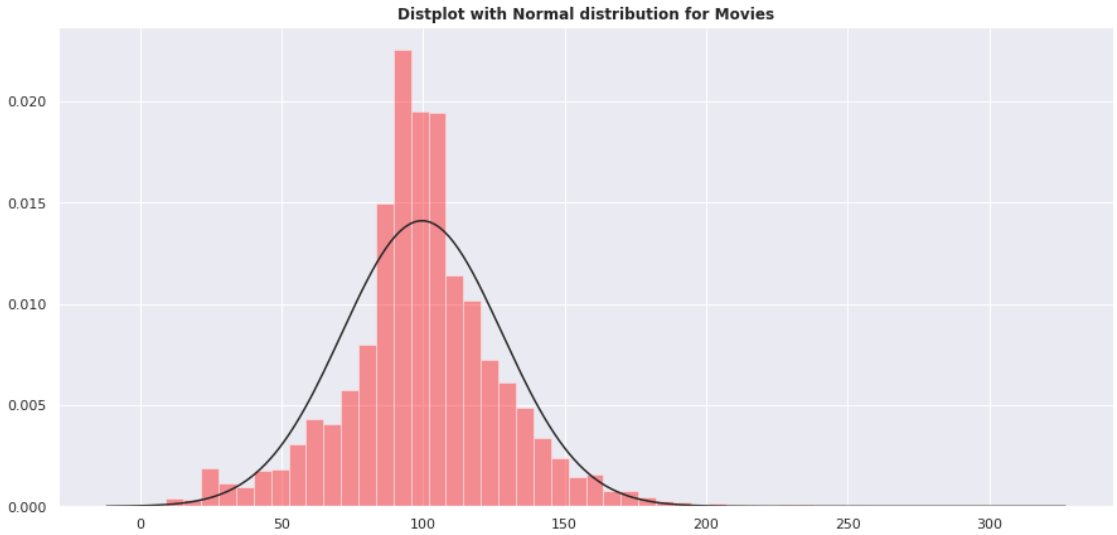
0.1.4 1.1.5 Distribution of movie duration (or number of seasons of TV shows)

[80]:
```
df_tv = netflix_df[netflix_df["type"] == "TV Show"]
df_movies = netflix_df[netflix_df["type"] == "Movie"]
```

[81]:
```
from scipy.stats import norm

plt.figure(figsize=(15,7))
sns.distplot(df_movies['duration'].str.
 ↪extract('(\d+)'),fit=norm,kde=False,color=['red'])
plt.title('Distplot with Normal distribution for Movies',fontweight="bold")
plt.show()
```

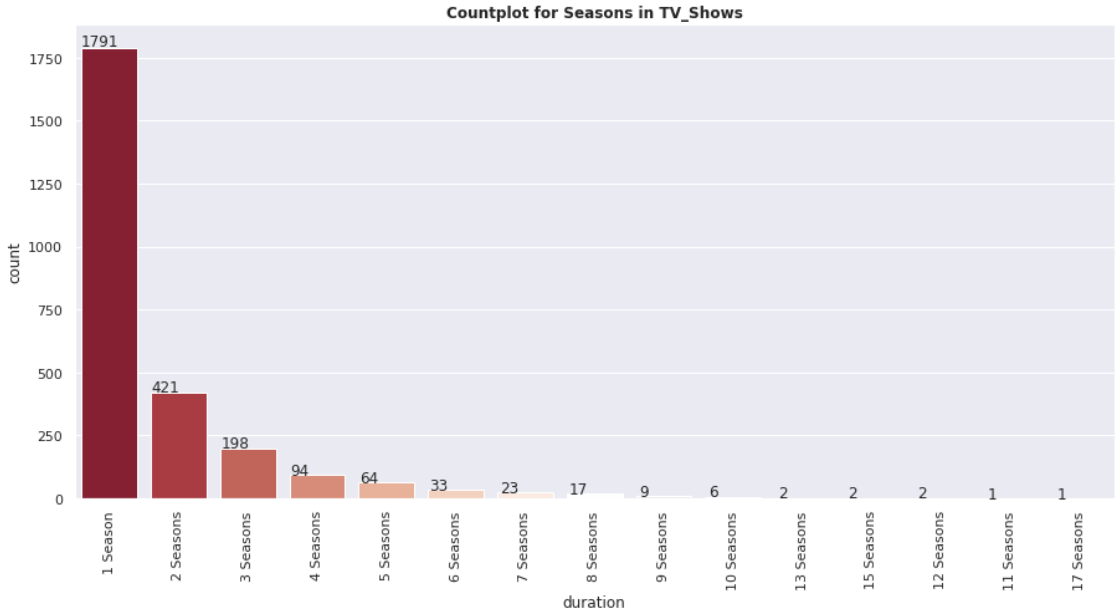
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619:
FutureWarning:
```

```
`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
```



[82]:
```
plt.figure(figsize=(15,7))
ax = sns.countplot(df_tv['duration'],order = df_tv['duration'].value_counts().
 ↪index,palette="RdGy")
plt.title('Countplot for Seasons in TV_Shows',fontweight="bold")
plt.xticks(rotation=90)
for p in ax.patches:
        ax.annotate(str(p.get_height()), (p.get_x() * 1.005, (p.get_height() *␣
 ↪1.005)))
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning:

Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
```



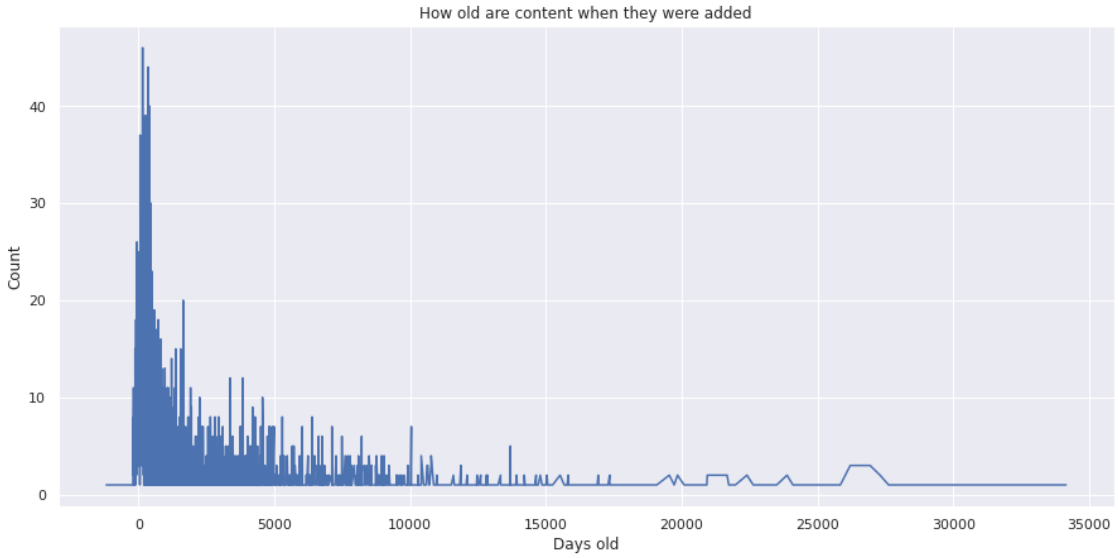
Analysis: * Movies: The Distribution is Normal, the mean length of the distribution is about 90-110 minutes * Tv shows: Majority of the TV shows on netflix have 1 season.The distribution is skewed to the right and about 90% of the TV Shows have 3 seasons or fewer

0.1.5 1.1.6 Plot of the age of content when it is released on Netflix

[83]:
```
df_copy = netflix_df.copy()
df_copy["date_added"] = pd.to_datetime(df_copy["date_added"])
```

[84]:
```
release_year = df_copy["release_year"].apply(lambda x: pd.Timestamp(x, 7, 1))
# now get the difference between date added and date released for each movie
df_copy["time_to_add"] = (df_copy["date_added"] - release_year).dt.days
years = df_copy.groupby("time_to_add").size()
plt.figure(figsize=(15,7))
plt.title('How old are content when they were added')
plt.xlabel('Days old')
plt.ylabel('Count')
sns.lineplot(data=years)
```

[84]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6756b04bd0>
```

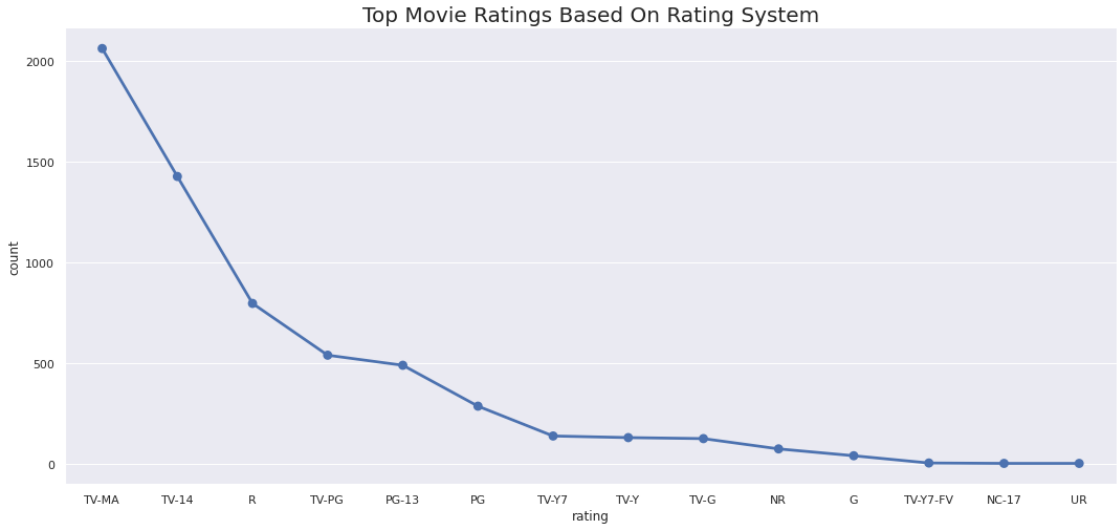


We can see that the majority of all the content is less than 5000 days old before the were added to Netflix's library.

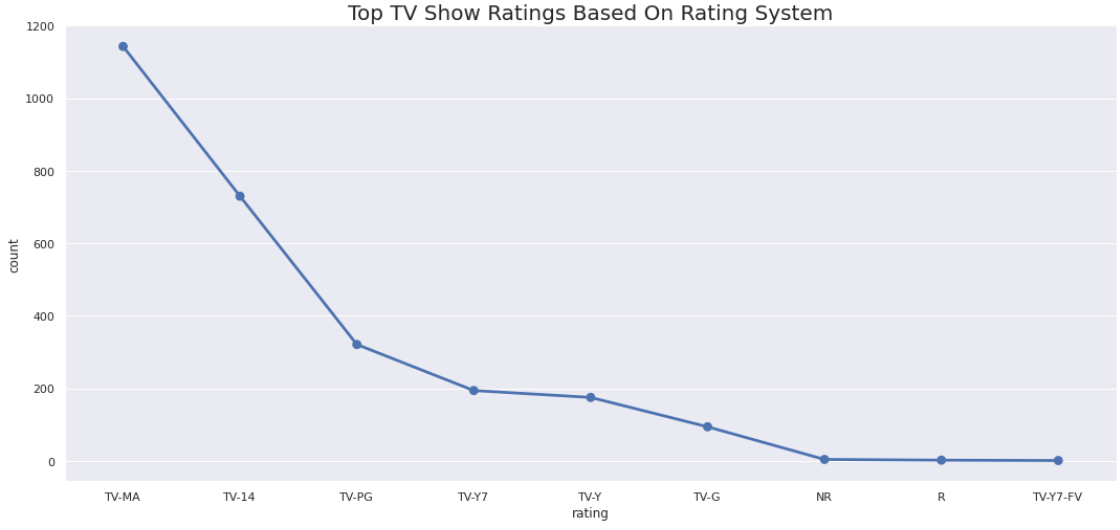
0.1.6 1.1.7 How is it distributed by age/maturity rating?

[85]:
```
# we are using px here to see the smaller ratings too
x = netflix_df['rating'].value_counts()
fig = px.pie(values = x.values,
             names = x.index,
             color_discrete_sequence=px.colors.sequential.Reds)
fig.show()
```

[86]:
```
df_tv = netflix_df[netflix_df["type"] == "TV Show"]
df_movie = netflix_df[netflix_df["type"] == "Movie"]

movie_ratings = df_movie.groupby(['rating'])['show_id'].count().
 ↪reset_index(name='count').sort_values(by='count',ascending=False)
fig_dims = (18,8)
fig, ax = plt.subplots(figsize=fig_dims)
sns.pointplot(x='rating',y='count',data=movie_ratings)
plt.title('Top Movie Ratings Based On Rating System',size='20')
plt.show()
```



[87]:
```
tv_ratings = df_tv.groupby(['rating'])['show_id'].count().
 ↪reset_index(name='count').sort_values(by='count',ascending=False)
fig_dims = (18,8)
fig, ax = plt.subplots(figsize=fig_dims)
sns.pointplot(x='rating',y='count',data=tv_ratings)
plt.title('Top TV Show Ratings Based On Rating System',size='20')
plt.show()
```



The top 3 ratings for Movies and TV Shows are:

1. **TV-MA**
2. **TV-14**

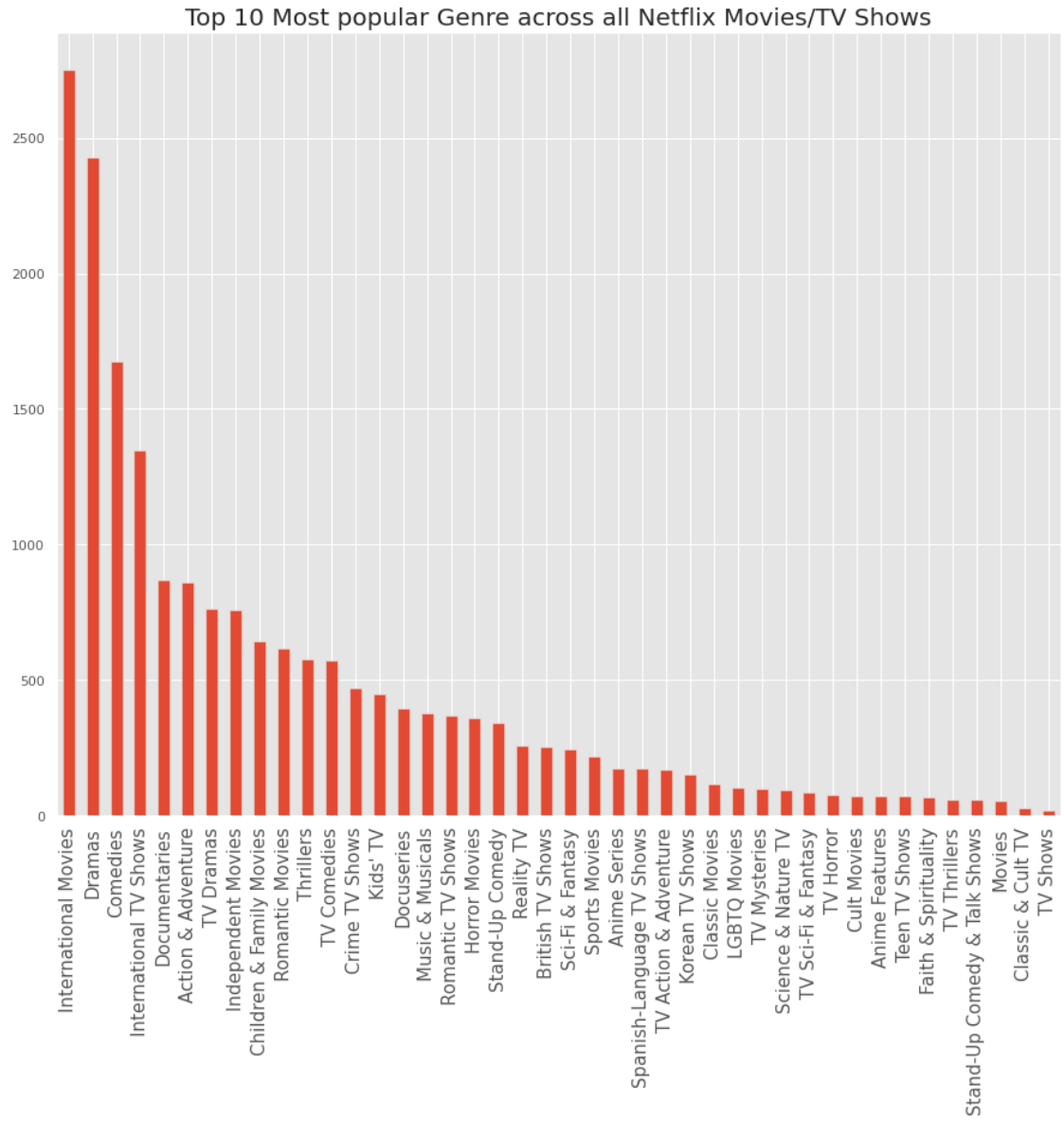
3. **TV-PG The top 3 ratings for Movies are:**
4. **TV-MA**
5. **TV-14**
6. **R The top 3 ratings for TV Shows are:**
7. **TV-MA**
8. **TV-14**
9. **TV-PG**

0.1.7 1.1.8 How is it distributed by genre? Which genres are the most popular?

[88]:
```
#Here we seperate multiple genres in each Movie/TV show's genre, and only keep␣
 ↪the one that is first

genre = netflix_df['listed_in'].apply(lambda t: t.split(', '))
genre = list(genre)


encoder = TransactionEncoder().fit(genre)
onehot = encoder.transform(genre)
onehot_genre = pd.DataFrame(onehot, columns = encoder.columns_,␣
 ↪index=netflix_df['show_id'])

genre_count = onehot_genre.sum().sort_values(ascending=False)
genre_count.head()
```

[88]:
```
International Movies      2752
Dramas                    2426
Comedies                  1674
International TV Shows    1349
Documentaries              869
dtype: int64
```

[89]:
```
plt.style.use('ggplot')
plt.figure(figsize=(15, 12))
genre_count.plot(kind='bar')
plt.xticks(rotation='90')
plt.tick_params(axis='x', labelsize=15)
plt.title('Top 10 Most popular Genre across all Netflix Movies/TV Shows',␣
 ↪fontsize=20)
plt.show()
```

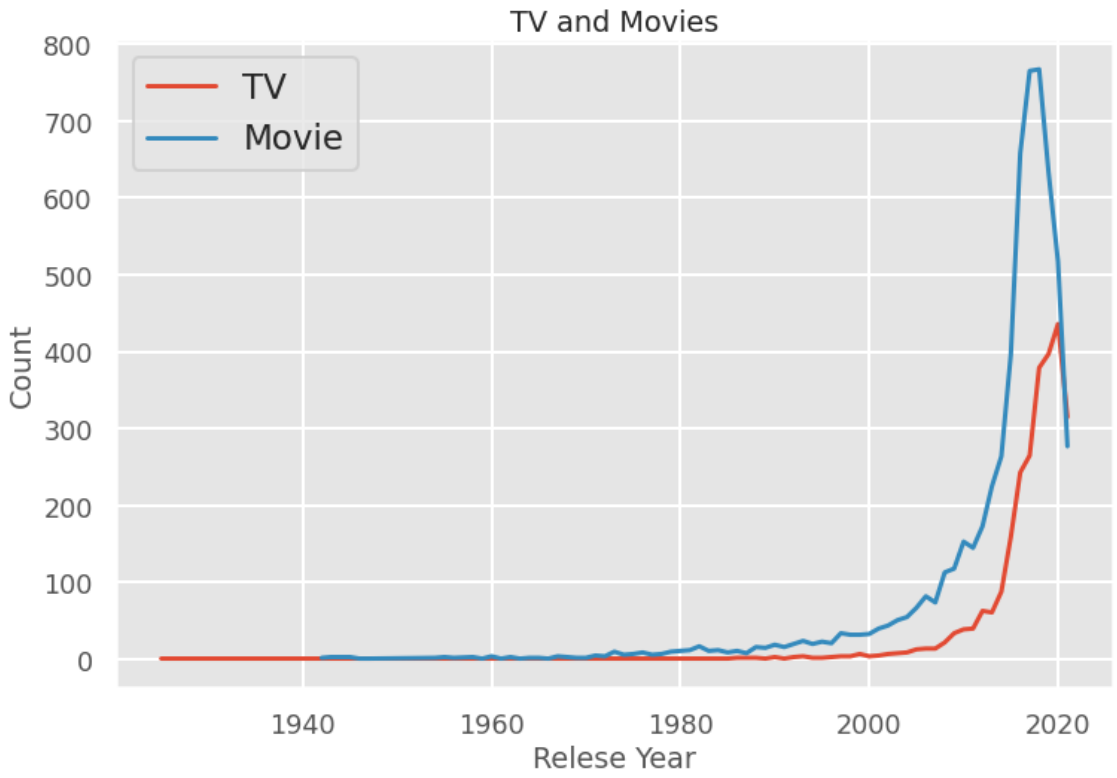


Most popular genre counts across Netflix: 1. **International Movies - 2752** 2. **Dramas - 2426** 3. **Comedies - 1674**

0.1.8 1.1.9 Are TV shows or movies added more regularly?

[90]:
```
df_tv = netflix_df[netflix_df['type'] == 'TV Show'].groupby('release_year').
 ↪count()
df_movies = netflix_df[netflix_df['type'] == 'Movie'].groupby('release_year').
 ↪count()

plt.figure(figsize=(12,8))
sns.set_context('poster', font_scale=0.8)
```

```
sns.lineplot(data = df_tv['show_id'], sizes=10)
sns.lineplot(data = df_movies['show_id'])
plt.ylabel('Count')
plt.xlabel('Relese Year')
plt.legend(['TV', 'Movie'], fontsize='large')
plt.title('TV and Movies')
plt.show()
```

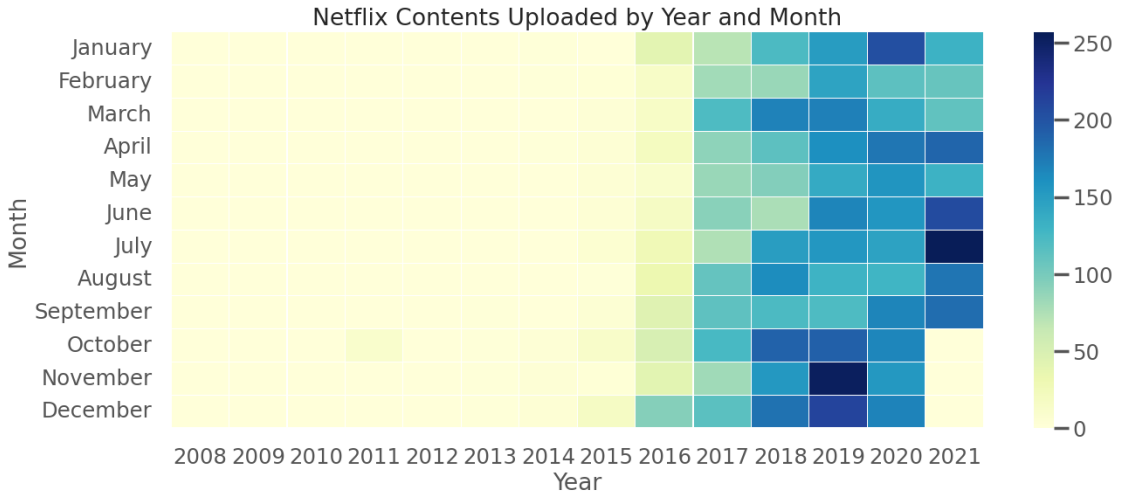


Movies and TV Shows were added on a regular increasing rate over the years until 2019, this may be due to the impact of Coronavirus, therefore they werent able toupload more content from 2019-2020 and it decreased drastically

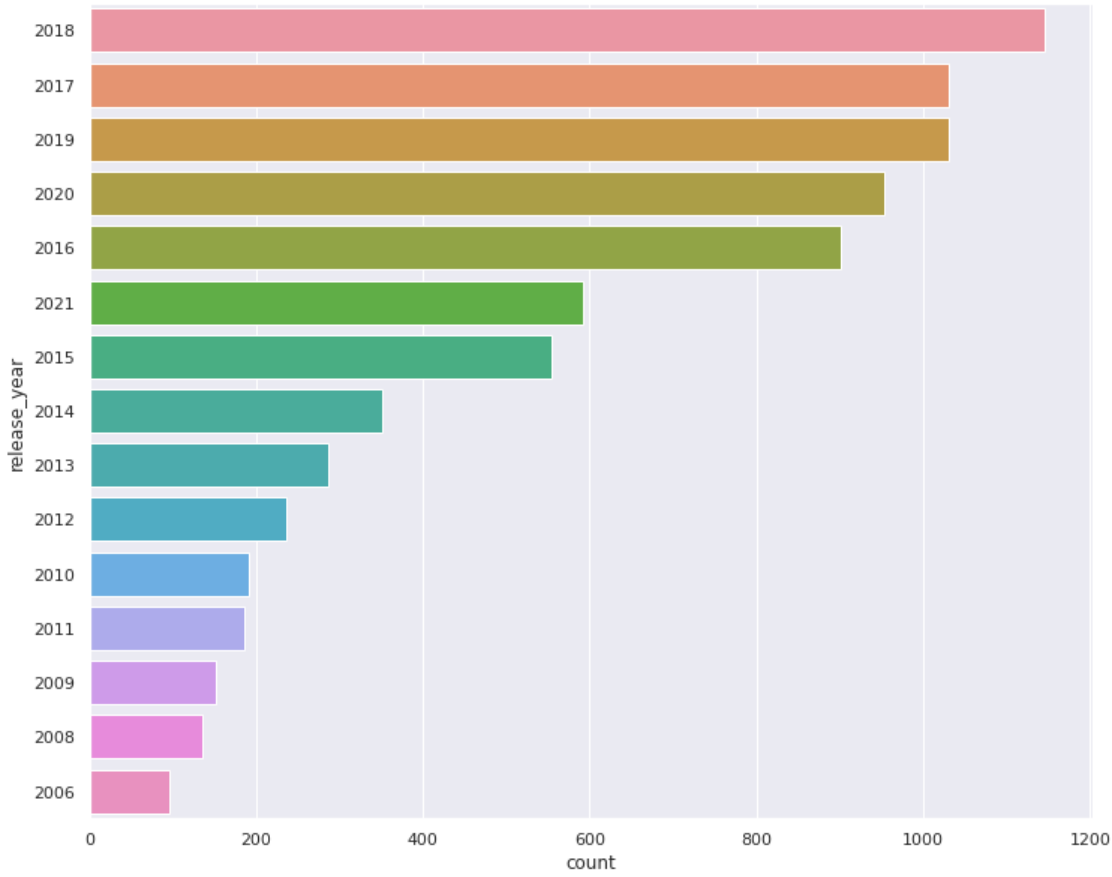
0.1.9 1.1.10 Is there a particular time of week/year when content gets uploaded?

[91]:
```
net_date = netflix_df[['date_added']].dropna()
net_date['Year'] = net_date.date_added.apply(lambda y : y.split(' ')[-1])
net_date['Month'] = net_date.date_added.apply(lambda y : y.split(' ')[0])

month_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July',␣
 ↪'August', 'September', 'October', 'November', 'December']
df = net_date.groupby('Year')['Month'].value_counts().unstack().
 ↪fillna(0)[month_order].T
```

```
ax = plt.subplots(figsize = (15, 6), dpi=100)
plt.title('Netflix Contents Uploaded by Year and Month')
ax = sns.heatmap(df, cmap="YlGnBu", linewidths=.1)
```



[92]:
```
plt.figure(figsize=(12,10))
sns.set(style="darkgrid")
ax = sns.countplot(y="release_year", data=netflix_df,␣
 ↪order=netflix_df['release_year'].value_counts().index[0:15])
```



As the years increased we can see that more content have been added frequetly,with 2018 being the year where the most content was added.October, November and december is the months where the most content was uploaded, with December having the most uploads.This might be becuase its the time of vacation

0.2 1.2 IMDB Ratings

0.2.1 1.2.1 Joining the datasets

[93]:
```
#reading our data
basics_df = pd.read_csv('/content/drive/MyDrive/IDVE_Exam/title.basics_small.
 ↪tsv.gz',sep='\t')
ratings_df = pd.read_csv('/content/drive/MyDrive/IDVE_Exam/title.ratings_small.
 ↪tsv.gz',sep='\t')
basics_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 138070 entries, 0 to 138069
Data columns (total 9 columns):
 #   Column          Non-Null Count   Dtype
```

```
---  ------           --------------   -----
 0   tconst           138070 non-null  object
 1   titleType        138070 non-null  object
 2   primaryTitle     138070 non-null  object
 3   originalTitle    138070 non-null  object
 4   isAdult          138070 non-null  int64
 5   startYear        138070 non-null  int64
 6   endYear          138070 non-null  int64
 7   runtimeMinutes   138070 non-null  int64
 8   genres           138070 non-null  object
dtypes: int64(4), object(5)
memory usage: 9.5+ MB
```

[94]:
```
ratings_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 114155 entries, 0 to 114154
Data columns (total 3 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   tconst         114155 non-null  object
 1   averageRating  114155 non-null  float64
 2   numVotes       114155 non-null  int64
dtypes: float64(1), int64(1), object(1)
memory usage: 2.6+ MB
```

[95]:
```
basics_df.isna().sum()
```

[95]:
```
tconst            0
titleType         0
primaryTitle      0
originalTitle     0
isAdult           0
startYear         0
endYear           0
runtimeMinutes    0
genres            0
dtype: int64
```

[96]:
```
ratings_df.isna().sum()
```

[96]:
```
tconst           0
averageRating    0
numVotes         0
dtype: int64
```

[97]:
```
#Merging our raings and basics datasets
rated_titles = pd.merge(basics_df.set_index('tconst'), ratings_df.
 ↪set_index('tconst'), left_index=True, right_index=True).drop_duplicates()
```

```
rated_titles.shape
```

[97]:
```
(114155, 10)
```

[98]:
```
rated_titles.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 114155 entries, tt0000004 to tt9916580
Data columns (total 10 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   titleType       114155 non-null  object
 1   primaryTitle    114155 non-null  object
 2   originalTitle   114155 non-null  object
 3   isAdult         114155 non-null  int64
 4   startYear       114155 non-null  int64
 5   endYear         114155 non-null  int64
 6   runtimeMinutes  114155 non-null  int64
 7   genres          114155 non-null  object
 8   averageRating   114155 non-null  float64
 9   numVotes        114155 non-null  int64
dtypes: float64(1), int64(5), object(4)
memory usage: 9.6+ MB
```

[99]:
```
rated_titles.duplicated().sum()
```

[99]:
```
0
```

[100]:
```
rated_titles.head()
```

[100]:

tconst	titleType	primaryTitle	...	averageRating	numVotes
			...		
tt0000004	short	Un bon bock	...	6.0	153
tt0000020	short	The Derby 1895	...	5.0	319
tt0000023	short	The Sea	...	5.7	1293
tt0000031	short	Jumping the Blanket	...	5.5	936
tt0000051	short	The Bohemian Encampment	...	3.8	32

[5 rows x 10 columns]

[101]:
```
rated_titles_clean = rated_titles.copy()

net_clean = n_df[n_df['type'] == 'Movie']
net_clean.shape
```

[101]:
```
(6131, 12)
```

[102]:
```
net_clean['title']= net_clean['title'].str.lower()

rated_titles_clean['primaryTitle'] = rated_titles_clean['primaryTitle'].str.
 ↪lower()
```

```
rated_titles_clean['originalTitle'] = rated_titles_clean['originalTitle'].str.
 ↪lower()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1:
SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[103]:
```
rated_titles_clean = rated_titles_clean[rated_titles_clean.startYear.
 ↪apply(lambda x: str(x).isnumeric())]
rated_titles_clean['startYear'] = rated_titles_clean['startYear'].astype(int)
rated_titles_clean.shape
```

[103]:
```
(114155, 10)
```

[104]:
```
rated_titles_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 114155 entries, tt0000004 to tt9916580
Data columns (total 10 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   titleType       114155 non-null  object
 1   primaryTitle    114155 non-null  object
 2   originalTitle   114155 non-null  object
 3   isAdult         114155 non-null  int64
 4   startYear       114155 non-null  int64
 5   endYear         114155 non-null  int64
 6   runtimeMinutes  114155 non-null  int64
 7   genres          114155 non-null  object
 8   averageRating   114155 non-null  float64
 9   numVotes        114155 non-null  int64
dtypes: float64(1), int64(5), object(4)
memory usage: 9.6+ MB
```

[105]:
```
print(net_clean.columns,"\n",rated_titles_clean.columns)
```

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

```
       'endYear', 'runtimeMinutes', 'genres', 'averageRating', 'numVotes'],
      dtype='object')
```

[106]:
```
df = pd.merge(net_clean, rated_titles_clean, left_on=['title','release_year'],␣
 ↪right_on=['primaryTitle','startYear'])
df.sample(5)
```

[106]:
```
     show_id   type  ... averageRating numVotes
605    s1252  Movie  ...           8.0   252040
1195   s2383  Movie  ...           5.6      632
665    s1377  Movie  ...           6.4     1439
4692   s8509  Movie  ...           5.9     4030
1947   s3917  Movie  ...           7.1      130

[5 rows x 22 columns]
```

[107]:
```
df.shape
```

[107]:
```
(4884, 22)
```

[108]:
```
df.duplicated().sum()
```

[108]:
```
0
```

[109]:
```
df.isna().sum()
```

[109]:
```
show_id            0
type               0
title              0
director          97
cast             354
country          168
date_added         0
release_year       0
rating             1
duration           3
listed_in          0
description        0
titleType          0
primaryTitle       0
originalTitle      0
isAdult            0
startYear          0
endYear            0
runtimeMinutes     0
genres             0
averageRating      0
numVotes           0
dtype: int64
```

After combining the movies with their ratings our data makes sense, the title columns content are the same as the primaryTitle and originalTitle columns from the imdb dataset that were

merged.The runtimes from are similar in most cases, but we do have some runtime differences by 1-4 min when we compare.We also have the number of votes and the average rating for each movie

0.2.2 1.2.2 Missing joins

[109]:
```
```

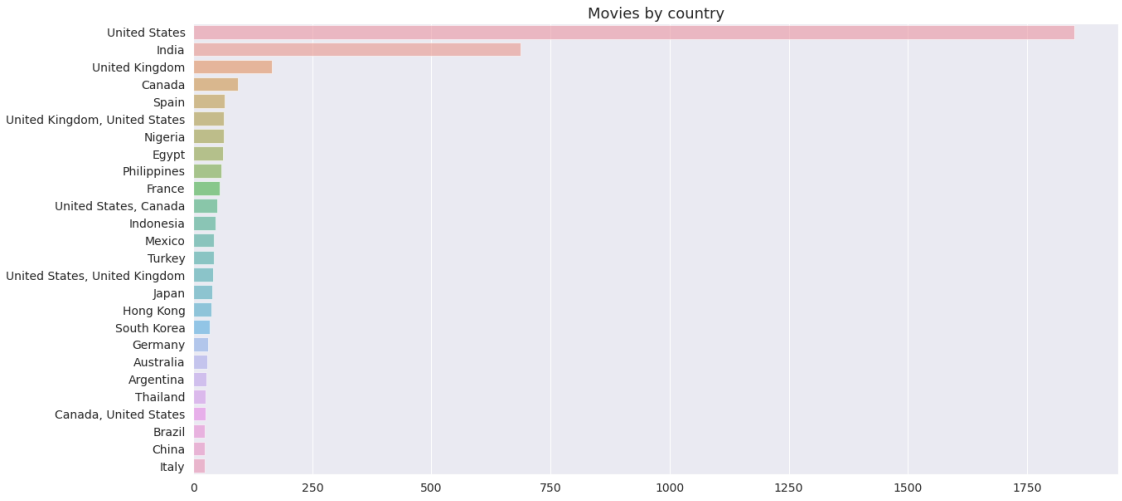
0.2.3 1.2.3 Plot Ratings

[110]:
```
fig = px.histogram(data_frame=df, x=df["averageRating"], title="Ratings of␣
↪Netflix Movies")
fig.show()
```

From the averageRating plot above we can see the data is negative skew.Therefore netflix can be seen as having a good quality library.The highest count rating is 6.5

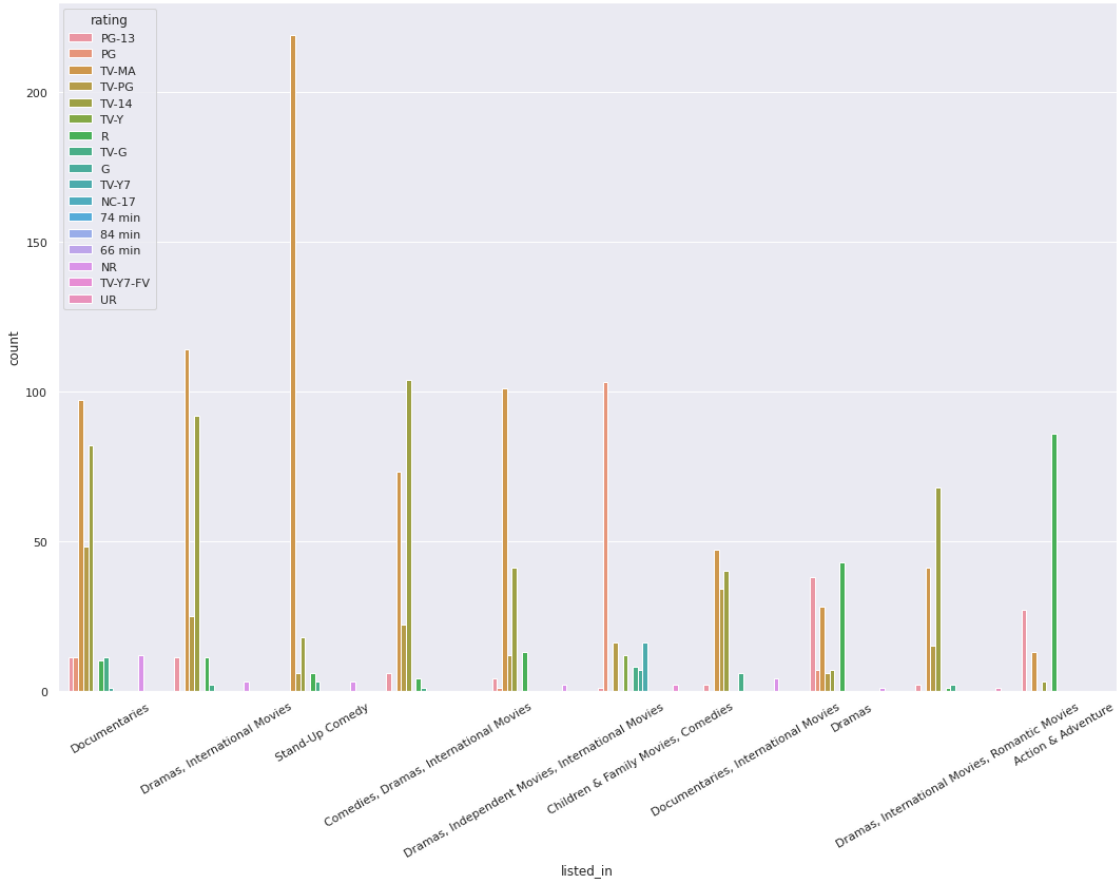
0.2.4 1.2.4 Plot Relationships between movie and other features

[111]:
```
df_movies = df[df['type'] == 'Movie']
countries = df_movies['country'].value_counts()[df_movies['country'].
↪value_counts(normalize=True)> 0.005]
list_countries = list(countries.index)
plt.figure(figsize=(20,10))
plt.title('Movies by country', fontsize=18)
plt.tick_params(labelsize=14)
sns.barplot(y=countries.index, x=countries.values, alpha=0.6)
plt.show()
```



The top 3 countries that produced the most movies are: USA, India and the united Kingdom

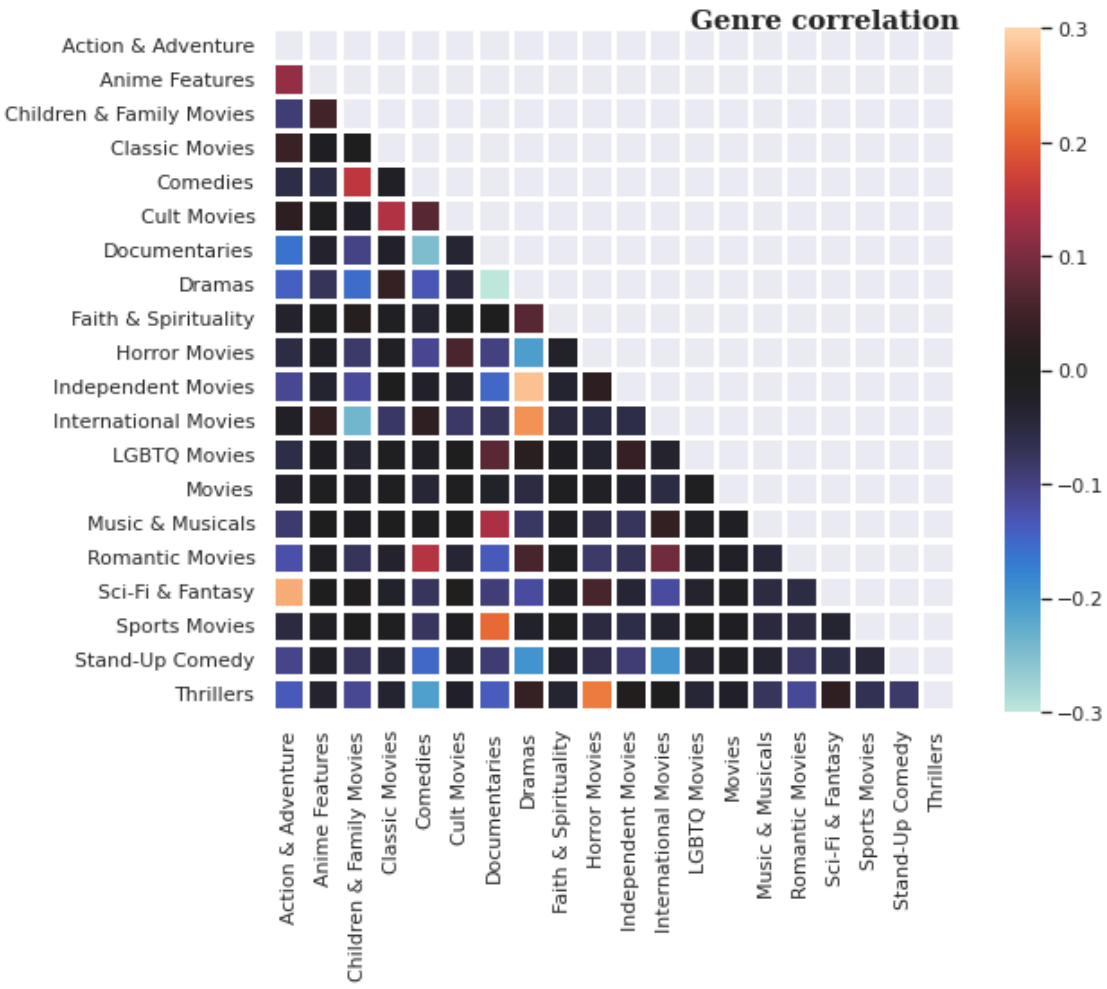
[112]:
```
sns.set()
plt.figure(figsize=(18,12))
sns.countplot(x='listed_in',hue='rating',data = df_movies,order␣
 ↪=df_movies["listed_in"].value_counts().index[0:10])
plt.xticks(rotation = 30)
plt.show()
```



Drama type movies are mostly rated by TV-MA

[113]:
```
from sklearn.preprocessing import MultiLabelBinarizer


df_movies['genre'] = df_movies['listed_in'].apply(lambda x :  x.replace('␣
 ↪',',',').replace(', ',',').split(','))
Types = []
for i in df_movies['genre']: Types += i
Types = set(Types)
test = df_movies['genre']
mlb = MultiLabelBinarizer()
res = pd.DataFrame(mlb.fit_transform(test), columns=mlb.classes_, index=test.
 ↪index)
```

```
corr = res.corr()
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
fig, ax = plt.subplots(figsize=(10, 7))
fig.text(.54,.88,'Genre correlation',␣
 ↪fontfamily='serif',fontweight='bold',fontsize=15)
pl = sns.heatmap(corr, mask=mask, vmax=.3, vmin=-.3, center=0, square=True,␣
 ↪linewidths=2.5)

plt.show()
```



Independant Movies and Drama's are the strongest correlated movie genre's followed by Scifi&Fantasy and Action and adventure

0.3 1.3 Actos and Directors

0.3.1 1.3.1 Missing directors

[114]:
```
df_movies.isna().sum()
```

[114]:
```
show_id             0
type                0
title               0
director           97
cast              354
country           168
date_added          0
release_year        0
rating              1
duration            3
listed_in           0
description         0
titleType           0
primaryTitle        0
originalTitle       0
isAdult             0
startYear           0
endYear             0
runtimeMinutes      0
genres              0
averageRating       0
numVotes            0
genre               0
dtype: int64
```

There are 97 missing directors

0.3.2 1.3.2 Director counts: Before vs After filling data

[114]:
```

```

0.3.3 1.3.3 Best director

[114]:
```

```

IGNORE BELOW

[117]:
```
%cd 'FOLDER'
```

```
/content/drive/My Drive/FOLDER
```

[118]:
```
!ls
```

```
IDVE_EXAM_Q1.ipynb  IDVE_EXAM_Q2.ipynb  IDVE_EXAM_Q3.ipynb
```

[119]:
```
!sudo apt-get install texlive-xetex texlive-fonts-recommended␣
↪texlive-generic-recommended
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
texlive-fonts-recommended is already the newest version (2017.20180305-1).
texlive-generic-recommended is already the newest version (2017.20180305-1).
texlive-xetex is already the newest version (2017.20180305-1).
0 upgraded, 0 newly installed, 0 to remove and 58 not upgraded.
```

[ ]:
```
!jupyter nbconvert --to pdf fileName.ipynb
```