Netflix and Code

Olivia Ramos

2023-04-01

(https://www.distractify.com/p/best-netflix-original-series-2022) Source: distractify.com (https://www.distractify.com/p/best-netflix-original-series-2022)

Overview

The **Netflix TV Shows and Movies** dataset is adapted from Kaggle (https://www.kaggle.com/datasets/victorsoeiro/netflix-tv-shows-and-movies), it includes two files **credits.csv** and **titles.csv** which are used in this work for data exploration.

- titles.csv contains information of unique show/movie titles
- credits.csv contains the information of actors/directors in each corresponding title

  - *actors and directors can have a role in one or more motion pictures*

This dataset was chosen for its flexibility, as it has sufficient information to analyze the data in varying ways. For instance, one can determine attribute distributions, trends and predictions, differentiation, and so forth. Moreover, this dataset stands out because it offers new and exciting insights into the framework of a widely-used platform like Netflix (i.e., a favoured pastime of many individuals). In this work, I will be demonstrating geographic mapping, attribute counts, query-like sub-setting, top films/actors, and the relationship between attributes.

titles.csv attributes:

- id[1]
- title
- show_type[2]
- description
- release_year
- age_certification
- runtime
- genres
- production_countries
- seasons[3]
- imdb_id[4]
- imdb_score
- imdb_votes
- tmdb_popularity[5]
- tmdb_score

credits.csv attributes:

- person_id
- id
- name
- character
- role[6]

Libraries used in this work:

```
library(tidyverse)
library(dplyr)
library(patchwork)
library(countrycode)
library(maps)
library(RColorBrewer)
library(ggmosaic)
library(flextable)
```

The *Fun* Stuff

```
# Load data
titles <- read_csv("titles.csv")
```

```
glimpse(titles)
```

```
## Rows: 5,850
## Columns: 15
## $ id                   <chr> "ts300399", "tm84618", "tm154986", "tm127384", "t…
## $ title                <chr> "Five Came Back: The Reference Films", "Taxi Driv…
## $ type                 <chr> "SHOW", "MOVIE", "MOVIE", "MOVIE", "MOVIE", "SHOW…
## $ description          <chr> "This collection includes 12 World War II-era pro…
## $ release_year         <dbl> 1945, 1976, 1972, 1975, 1967, 1969, 1979, 1971, 1…
## $ age_certification    <chr> "TV-MA", "R", "R", "PG", NA, "TV-14", "R", "R", "…
## $ runtime              <dbl> 51, 114, 109, 91, 150, 30, 94, 102, 110, 104, 158…
## $ genres               <chr> "['documentation']", "['drama', 'crime']", "['dra…
## $ production_countries <chr> "['US']", "['US']", "['US']", "['GB']", "['GB', '…
## $ seasons              <dbl> 1, NA, NA, NA, NA, 4, NA, NA, NA, NA, NA, NA, NA,…
## $ imdb_id              <chr> NA, "tt0075314", "tt0068473", "tt0071853", "tt006…
## $ imdb_score           <dbl> NA, 8.2, 7.7, 8.2, 7.7, 8.8, 8.0, 7.7, 7.7, 5.8, …
## $ imdb_votes           <dbl> NA, 808582, 107673, 534486, 72662, 73424, 395024,…
## $ tmdb_popularity      <dbl> 0.600, 40.965, 10.010, 15.461, 20.398, 17.617, 17…
## $ tmdb_score           <dbl> NA, 8.179, 7.300, 7.811, 7.600, 8.306, 7.800, 7.5…
```

```
head(titles, 4)
```

id <chr>	title <chr>	type <chr>
ts300399	Five Came Back: The Reference Films	SHOW
tm84618	Taxi Driver	MOVIE
tm154986	Deliverance	MOVIE
tm127384	Monty Python and the Holy Grail	MOVIE

4 rows | 1-3 of 15 columns

```
# Find duplicate data
sum(duplicated(titles))
```

```
## [1] 0
```

```
# Consider the the magnitude of `NA` values in each attribute
colSums(is.na(titles))
```

```
##                  id                title                 type 
##                   0                    1                    0 
##         description         release_year    age_certification 
##                  18                    0                 2619 
##             runtime               genres production_countries 
##                   0                    0                    0 
##             seasons              imdb_id           imdb_score 
##                3744                  403                  482 
##          imdb_votes      tmdb_popularity           tmdb_score 
##                 498                   91                  311
```

Since our dataset only contains two types of motion pictures, let's consider the proportion of movies vs. TV shows using `prop.table()` [7] while resisting the urge to use a *pie chart*:

```
titles |>
  count(type) |>
  mutate(pct = scales::percent(prop.table(n)))
```

type <chr>	n <int>	pct <chr>
MOVIE	3744	64%
SHOW	2106	36%

2 rows

What is the difference between **IMDb** and **TMDB** scores across each type?

```
diff <- titles |>
  select(type, imdb_score, tmdb_score) |>
  group_by(type) |>
  summarise(imdb_avg_score = mean(imdb_score, na.rm = TRUE),
            tmdb_avg_score = mean(tmdb_score, na.rm = TRUE)) |>
  mutate(difference = scales::percent((tmdb_avg_score-imdb_avg_score)/imdb_avg_score))
diff
```

type <chr>	imdb_avg_score <dbl>	tmdb_avg_score <dbl>	difference <chr>
MOVIE	6.246748	6.464012	3.5%
SHOW	6.977927	7.480413	7.2%

2 rows

TMDB is less of a critic…

The Production Countries of Netflix Titles

Since the `production_countries` column looks like this:

```
## [1] "['US']"       "['US']"       "['US']"       "['GB']"       "['GB', 'US']"
```

I will be simply doing *this*:

```
# Extract all strings in production countries
countries <- titles$production_countries |>
  str_extract_all("[A-Z]+") |>
  unlist(recursive = TRUE)

# Create data frame of country counts
country_count <- as.data.frame(countries) |>
  count(countries, sort = TRUE)

head(country_count, 5)
```

	countries <chr>	n <int>
1	US	2323
2	IN	622
3	GB	404
4	JP	287
5	FR	248

5 rows

I am using the package countrycode (https://cran.r-project.org/web/packages/countrycode/countrycode.pdf) to map the country abbreviations to a region. This mapping will allow me to merge with `map_data("world")` :

```
# Create new column of regions
country_names <- country_count |>
  mutate(region = countrycode(sourcevar = country_count$countries, "iso2c", "country.name")) |>
  drop_na() |>
  rename("total" = "n")
```

```
head(country_names, 5)
```

	countries <chr>	total <int>	region <chr>
1	US	2323	United States
2	IN	622	India
3	GB	404	United Kingdom
4	JP	287	Japan
5	FR	248	France

5 rows

Creating a Data Frame of Map Data

Using the package maps (https://eriqande.github.io/rep-res-web/lectures/making-maps-with-R.html#:~:text=The%20maps%20package%20contains%20a,maps%20in%20the%20maps%20package.), I will join my `country_names` dataset with the `world` dataset from the package.

```
world <- map_data("world")
head(world, 3)
```

	long <dbl>	lat <dbl>	group <dbl>	order <int>	region <chr>	subregion <chr>
1	-69.89912	12.45200	1	1	Aruba	NA
2	-69.89571	12.42300	1	2	Aruba	NA
3	-69.94219	12.43853	1	3	Aruba	NA

3 rows

The following code is adapted from **Sarah Penir's** article: Making Maps with ggplot2 (https://sarahpenir.github.io/r/making-maps/)

```
# Determine the set difference between the two datasets on 'region'
setdiff(country_names$region, world$region) |> print()
```

```
## [1] "United States"                 "United Kingdom"
## [3] "Hong Kong SAR China"           "Palestinian Territories"
## [5] "Czechia"                       "British Indian Ocean Territory"
## [7] "Congo - Kinshasa"              "St. Kitts & Nevis"
## [9] "Vatican City"
```

We need `country_names$region` to match `world$region` in order to do the join, so I will recode the mismatched strings:

```
# Recoding the 'region' column
tedious <- country_names |>
  mutate(region = recode(str_trim(region),
                         "United States" = "USA",
                         "United Kingdom" = "UK",
                         "Hong Kong SAR China" = "China",
                         "Palestinian Territories" = "Palestine",
                         "Czechia" = "Czech Republic",
                         "British Indian Ocean Territory" = "UK", # ???
                         "Congo - Kinshasa" = "Democratic Republic of the Congo",
                         "St. Kitts & Nevis" = "Nevis",
                         "Vatican City" = "Vatican"))
```

I am going to bin each country's production counts before plotting:

```
tedious <- tedious |>
  mutate(Productions = cut(total, seq(0, 2400, 200), dig.lab = 5))
```

```
head(tedious, 5)
```

	countries	total	region	Productions
	<chr>	<int>	<chr>	<fct>
1	US	2323	USA	(2200,2400]
2	IN	622	India	(600,800]
3	GB	404	UK	(400,600]
4	JP	287	Japan	(200,400]
5	FR	248	France	(200,400]

5 rows

Before we proceed with the join, let's determine the set difference once again:

```
setdiff(tedious$region, world$region) |> print()
```

```
## character(0)
```

```
sub <- left_join(world, tedious, by = "region")
```

```
## Warning in left_join(world, tedious, by = "region"): Detected an unexpected many-to-many rela
tionship between `x` and `y`.
## ℹ Row 28281 of `x` matches multiple rows in `y`.
## ℹ Row 72 of `y` matches multiple rows in `x`.
## ℹ If a many-to-many relationship is expected, set `relationship =
##   "many-to-many"` to silence this warning.
```

Visualizing the Global Production Count

```
pinks <- c("#FBE6C5FF","#F5BA98FF", "#FA8A76FF", "#C8586CFF", "#70284AFF")

theme1 <- theme(plot.title = element_text(hjust = 0.5),
                panel.background = element_rect(fill = "white"),
                panel.border = element_blank(),
                panel.grid = element_blank(),
                axis.title = element_blank(),
                axis.text = element_blank(),
                axis.line = element_blank(),
                axis.ticks = element_blank())

w_pc <- ggplot(sub, aes(x = long, y = lat, group = group)) +
  coord_fixed(1.3) +
  ylim(-55, 84) +
  geom_polygon(aes(fill = Productions), color = "white") +
  scale_fill_manual(values = pinks, na.value = "lavenderblush3") +
  theme1 +
  labs(title = "Global Production Count")

w_pc
```

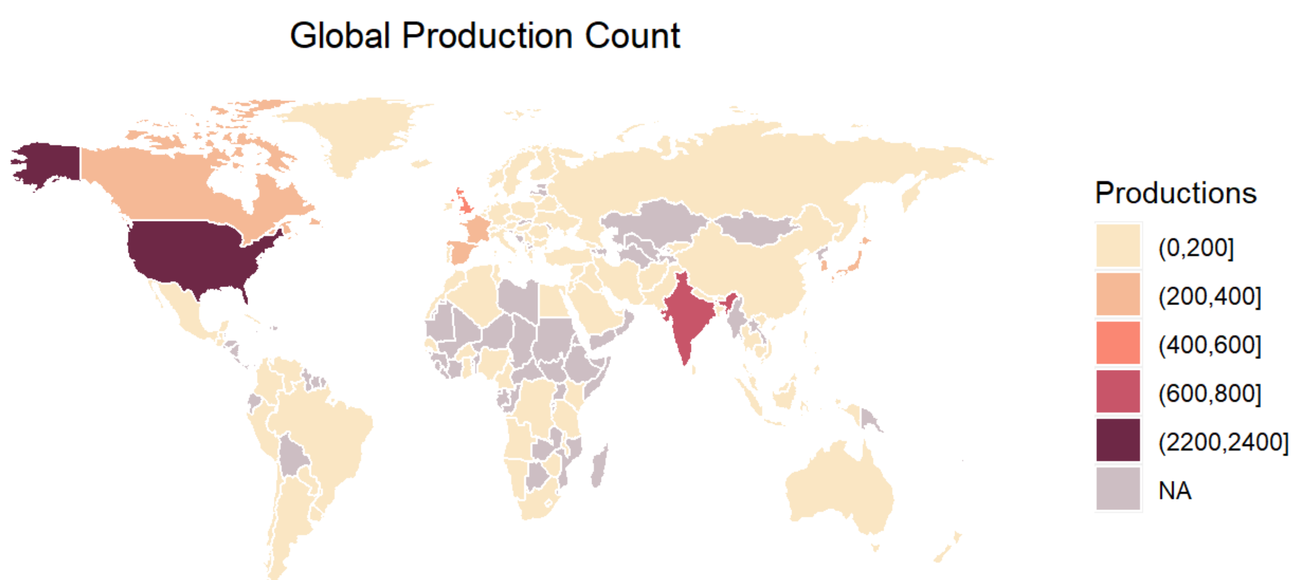

Fig 1. Count of motion pictures produced by country.

As we can see, the United States produced the most motion pictures (2200+), followed by India (600+) then the United Kingdom.

Visualizing the Distribution of Genres

```
# Extract all strings in genre column
genres <- titles$genres |>
  str_extract_all("[a-z]+") |>
  unlist(recursive = TRUE)

genre_count <- as.data.frame(genres) |>
  count(genres, sort = TRUE) |>
  mutate(pct = prop.table(n))

head(genre_count)
```

	genres <chr>	n <int>	pct <dbl>
1	drama	2968	0.19671262
2	comedy	2325	0.15409597
3	thriller	1228	0.08138918
4	action	1157	0.07668346
5	romance	971	0.06435578
6	documentation	952	0.06309650

6 rows

```r
# Extend colour palette
extendo <- colorRampPalette(brewer.pal(12, "Set3"))(nrow(genre_count))

theme2 <- theme(legend.position = "none",
                panel.grid.major.x = element_line(colour = "ivory2"),
                panel.grid.minor.x = element_line(colour = "white"),
                panel.background = element_rect(fill = "white"),
                plot.title = element_text(hjust = 0.5),
                axis.title.x = element_text(margin = margin(t = 20)),
                axis.title.y = element_blank())

# Plotting time
g <- ggplot(genre_count, aes(x = reorder(genres, n), y = n, fill = genres)) +
     geom_bar(stat = "identity") +
     coord_flip() +
     scale_fill_manual(values = extendo) +
     geom_text(size = 3.5,
               hjust = "inward",
               aes(label = scales::percent(pct, accuracy = 0.1))) +
     scale_y_continuous(breaks = scales::breaks_extended(n = 9),
                        labels = scales::label_comma()) +
     labs(title = "The Distribution of Genres on Netflix",
          y = "Count") +
  theme2

g
```

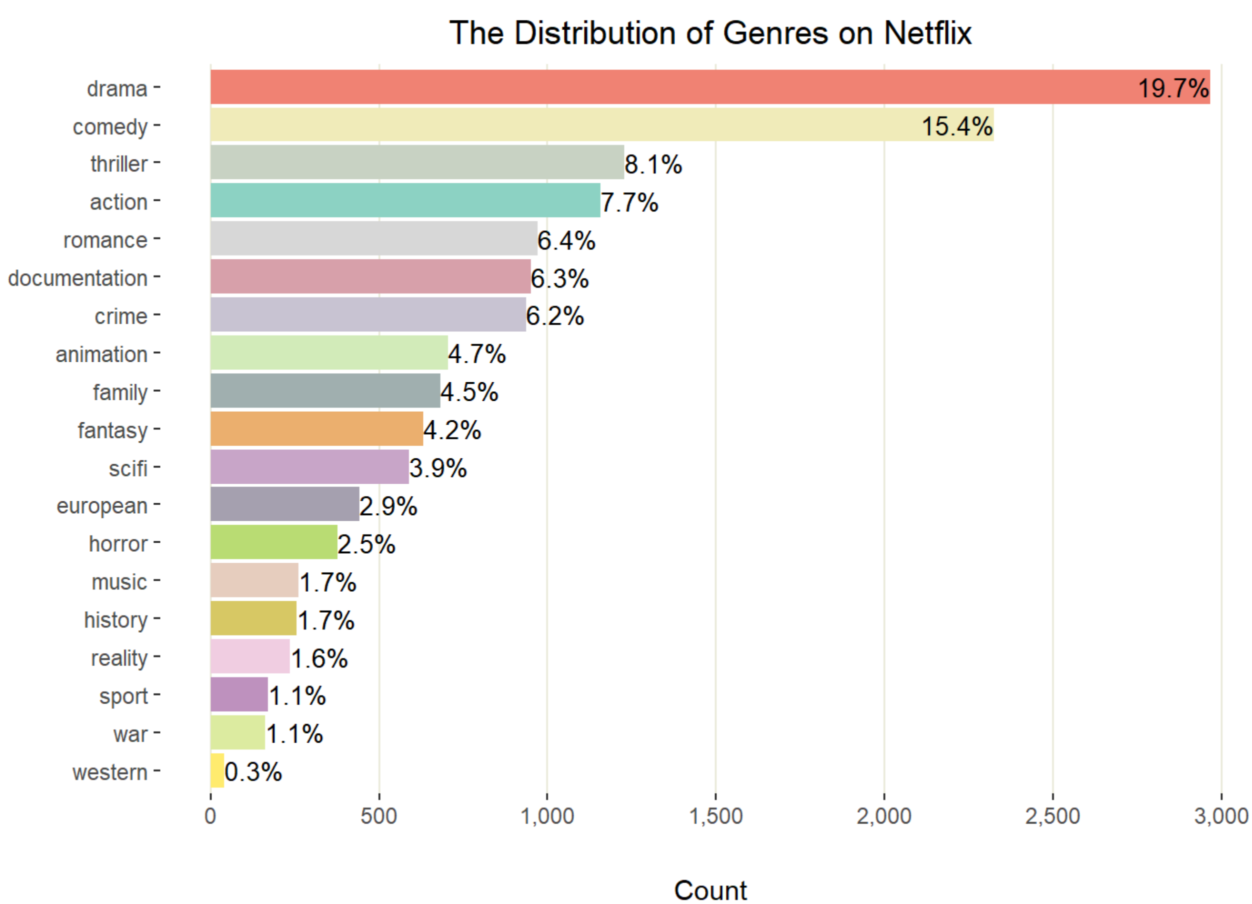

Fig 2. The distribution of unique genres on Netflix.
Drama dramatically accounts for nearly 20% of titles on Netflix. It is also funny how comedy is nearly double the proportion of the two genres preceding it!

Exploring Two Datasets with One Code

```
# Load some more data
credits <- read_csv("credits.csv")
```

```
glimpse(credits)
```

```
## Rows: 77,801
## Columns: 5
## $ person_id <dbl> 3748, 14658, 7064, 3739, 48933, 32267, 519612, 29068, 519613…
## $ id        <chr> "tm84618", "tm84618", "tm84618", "tm84618", "tm84618", "tm84…
## $ name      <chr> "Robert De Niro", "Jodie Foster", "Albert Brooks", "Harvey K…
## $ character <chr> "Travis Bickle", "Iris Steensma", "Tom", "Matthew 'Sport' Hi…
## $ role      <chr> "ACTOR", "ACTOR", "ACTOR", "ACTOR", "ACTOR", "ACTOR", "ACTOR…
```

```
head(credits, 4)
```

person_id <dbl>	id <chr>	name <chr>	character <chr>	role <chr>
3748	tm84618	Robert De Niro	Travis Bickle	ACTOR
14658	tm84618	Jodie Foster	Iris Steensma	ACTOR
7064	tm84618	Albert Brooks	Tom	ACTOR
3739	tm84618	Harvey Keitel	Matthew 'Sport' Higgins	ACTOR

4 rows

```
sum(duplicated(credits))
```

```
## [1] 0
```

```
colSums(is.na(credits))
```

```
## person_id        id      name character      role
##         0         0         0      9772         0
```

Moving on to merging **titles.csv** and **credits.csv**:

```
# Merge on matching attribute column 'id'
friends <- left_join(credits, titles, by = "id")
```

I am creating a list of the top movie actors on the basis that they played a role in at least **3** movies with an average rating greater than **7**:

```
# Filter the movies by the biggest production country and a score > 7
p1 <- friends |>
  filter(str_detect(production_countries, "US")) |>
  mutate(avg_score = (tmdb_score + imdb_score) / 2 ) |>
  filter(avg_score >= 7 & type == "MOVIE")

# Filter with actors appearing at least 3 times
p2 <- p1 |>
  group_by(person_id) |>
  filter(sum(role %in% "ACTOR") >= 3) |>
  select(person_id, name, title, avg_score)

# Get the actors
p3 <- p2 |>
  select(person_id, name, title, avg_score) |>
  group_by(name) |>
  summarise(potential = mean(avg_score)) |>
  arrange(desc(potential))

head(p3, 5)
```

name <chr>	**potential** <dbl>
Joe Pesci	8.219333
Paul Herman	8.219333
Marion Cotillard	8.189500
Robert De Niro	8.129500
Cillian Murphy	8.111333

5 rows

```
# Extract top 10 rows
top10 <- p3 |> slice(1:10)
```

The Highest Rated Movie of the Top 10 Actors

We can format data frames nicely with the flextable (https://ardata-fr.github.io/flextable-book/) package:

```r
q1 <- filter(p2, name %in% top10$name) |>
      group_by(name) |>
      filter(avg_score == max(avg_score)) |>
      rename("Movie Score" = avg_score,
             "Actor" = name,
             "Top Film" = title) |>
      mutate_if(is.numeric, round, digits = 2)

flextable::flextable(q1[,2:4], cwidth = c(2, 2, 2))
```

Actor	Top Film	Movie Score
Robert De Niro	GoodFellas	8.58
Joe Pesci	GoodFellas	8.58
Paul Herman	GoodFellas	8.58
James Russo	Once Upon a Time in America	8.38
Leonardo DiCaprio	Inception	8.60
Tom Hardy	Inception	8.60
Cillian Murphy	Inception	8.60
Marion Cotillard	Inception	8.60
Michael Caine	Inception	8.60
Miranda Nolan	Inception	8.60

The Highest Rated Movies of the Top 10 Actors

```r
# Listing all the movies
q2 <- filter(p2, name %in% top10$name) |>
      group_by(name) |>
      summarize(title = paste(sort(unique(title)), collapse = ", "))

q3 <- left_join(top10, q2, by = "name")

# Create column of total movies
q3$movie_count <- str_count(q3$title, ",") + 1

q4 <- q3 |>
  rename("Total" = movie_count,
         "Actor" = name,
         "Top Films" = title,
         "Avg. Movie Score" = potential) |>
  mutate_if(is.numeric, round, digits = 2)

flextable::flextable(q4[order(-q4$Total),], cwidth = c(1, 1, 6, 1))
```

Actor	Avg. Movie Score	Top Films	Total
Leonardo DiCaprio	8.02	Blood Diamond, Catch Me If You Can, Django Unchained, Don't Look Up, Inception, The Departed, Titanic	7
Robert De Niro	8.13	Awakenings, GoodFellas, Once Upon a Time in America, Taxi Driver, The Irishman	5
Michael Caine	7.97	Dunkirk, Inception, Quincy, The Dark Knight Rises	4
Joe Pesci	8.22	GoodFellas, Once Upon a Time in America, The Irishman	3
Paul Herman	8.22	GoodFellas, Once Upon a Time in America, The Irishman	3
Marion Cotillard	8.19	Big Fish, Inception, The Dark Knight Rises	3
Cillian Murphy	8.11	Dunkirk, Inception, The Dark Knight Rises	3
Miranda Nolan	8.11	Dunkirk, Inception, The Dark Knight Rises	3
Tom Hardy	8.11	Dunkirk, Inception, The Dark Knight Rises	3
James Russo	8.08	Django Unchained, Donnie Brasco, Once Upon a Time in America	3

The actors starring in the top movies are generally those who are in the **same** set of movies.

Just for fun, let's see the rating for a great actor and director:

```
friends |>
  filter(name == "Keanu Reeves" | name == "Quentin Tarantino") |>
  group_by(name) |>
  summarise(rating = mean(imdb_score))
```

name <chr>	rating <dbl>
Keanu Reeves	5.80
Quentin Tarantino	7.78

2 rows

disappointing…

Finding the Unnecessarily Long Titles on Netflix

```
too_long <- titles |>
  group_by(title) |>
  summarise(runtime = max(runtime)) |>
  arrange(desc(runtime)) |>
  slice(1:20)

too_long <- too_long |>
  mutate(hours = runtime/60,
         title = fct_reorder(title, hours)) |>
  rename("Runtime (hours)" = hours,
         "Runtime (mins)" = runtime,
         "Motion Picture" = title) |>
  mutate_if(is.numeric, round, digits = 1)

flextable::flextable(too_long, cwidth = c(2, 2, 2))
```

Motion Picture	Runtime (mins)	Runtime (hours)
Bonnie & Clyde	240	4.0
A Lion in the House	225	3.8
Lagaan: Once Upon a Time in India	224	3.7
Jodhaa Akbar	214	3.6
Kabhi Khushi Kabhie Gham	210	3.5
The Irishman	209	3.5
No Direction Home: Bob Dylan	208	3.5
Hum Aapke Hain Koun..!	206	3.4
Jab Harry Met Sejal	200	3.3
Apocalypse Now Redux	196	3.3
Titanic	194	3.2
What's Your Raashee?	192	3.2
Wyatt Earp	191	3.2
Sivaji: The Boss	189	3.1
Swades	189	3.1
The Hateful Eight	188	3.1
RRR	187	3.1
Kal Ho Naa Ho	186	3.1
Lakshya	186	3.1

Motion Picture	Runtime (mins)	Runtime (hours)
Saladin the Victorious	186	3.1

Visualizing the Top Netflix Titles

```
# Finding the counts per year
titles |>
  count(release_year) |>
  arrange(desc(n))
```

release_year <dbl>	n <int>
2019	836
2020	814
2021	787
2018	773
2017	563
2022	371
2016	362
2015	223
2014	153
2013	135

1-10 of 63 rows Previous **1** 2 3 4 5 6 7 Next

Since 2019-2021 has the most releases, I will explore a subset of the data:

```r
# Subsetting by year and the biggest production country
top_titles <- titles |>
  filter(release_year >= 2019,
         str_detect(production_countries, "US")) |>
  arrange(desc(imdb_score)) |>
  slice(1:20)


top_titles |>
  mutate(title = fct_reorder(title, imdb_score)) |>
  ggplot(mapping = aes(x = imdb_score, y = title, group = 1,
                       color = type, label = round(imdb_score, 2))) +
  geom_segment(aes(x = 8, xend = imdb_score, yend = title), size = 0.9) +
  geom_point(size = 4) +
  scale_colour_manual(values = c("#FA7E5CFF", "#D17DF9FF"), name = "Type") +
  scale_x_continuous(breaks = seq(8.0, 9.2, 0.2), limits = c(8.0, 9.2)) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5),
        axis.title.y = element_blank(),
        legend.position = c(0.9, 0.2)) +
  labs(title = "Top 20 Motion Pictures from 2019-2022",
       x = "IMDb Score")
```

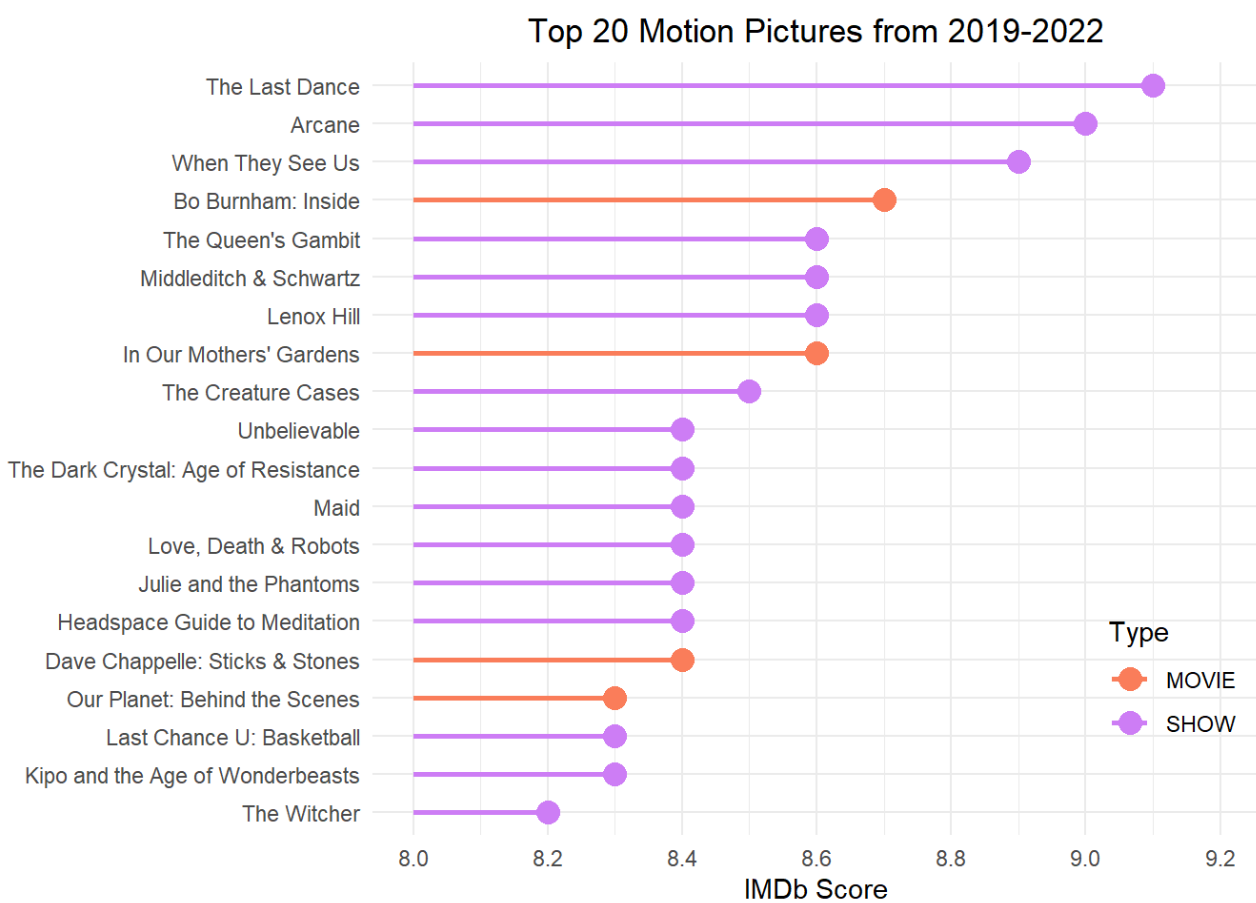

Fig 3. Top 20 titles on Netflix produced in the U.S. from 2019-2022.

Visualizing Rating vs. Runtime

I am separating the data into show releases before and after 2000. For simplicity, I am categorizing the data as pre/post Gen z. *I am aware* that Gen Z starts from 1997…

```
# Filter and categorize data
rnr <- titles |>
  filter(type == "SHOW") |>
  mutate(turning_point = factor(release_year < 2000,
                                levels = c(TRUE, FALSE),
                                labels = c("Pre Gen Z", "Post Gen Z")),
         mins = cut(runtime, breaks = c(0, 30, 60, 100),
                     labels = c("Short\n (up to 30)", "Long\n (up to 60)",
                                "Too long\n (up to 100)"), na.rm = TRUE),
         rating = cut(imdb_score, breaks = 4,
                     labels = c("Horrible", "Bad", "Okay",
                                "Good"), na.rm = TRUE))
```

Using the ggmosaic (https://cran.r-project.org/web/packages/ggmosaic/ggmosaic.pdf) package to plot:

```
traffic_lights <- c("#CF597EFF", "#DE8A5AFF", "#E9E29CFF", "#9CCB86FF")

theme3 <- theme(plot.title = element_text(hjust = 0.5),
                plot.background = element_blank(),
                panel.background = element_blank(),
                panel.border = element_blank(),
                panel.grid.major = element_blank(),
                panel.grid.minor = element_blank(),
                strip.background = element_blank(),
                strip.text = element_text(face = "italic"),
                axis.title.x = element_text(vjust = -1.2))

mos <- ggplot(rnr) +
  geom_mosaic(aes(x = product(mins), fill = rating),
              na.rm = TRUE, offset = 0, show.legend = FALSE) +
  facet_grid(.~turning_point, scales = "free_x") +
  scale_fill_manual(values = traffic_lights) +
  theme3 +
  labs(title = "The Distribution of Rating and Runtime",
       x = "Runtime (mins)",
       y = "Rating")

mos
```

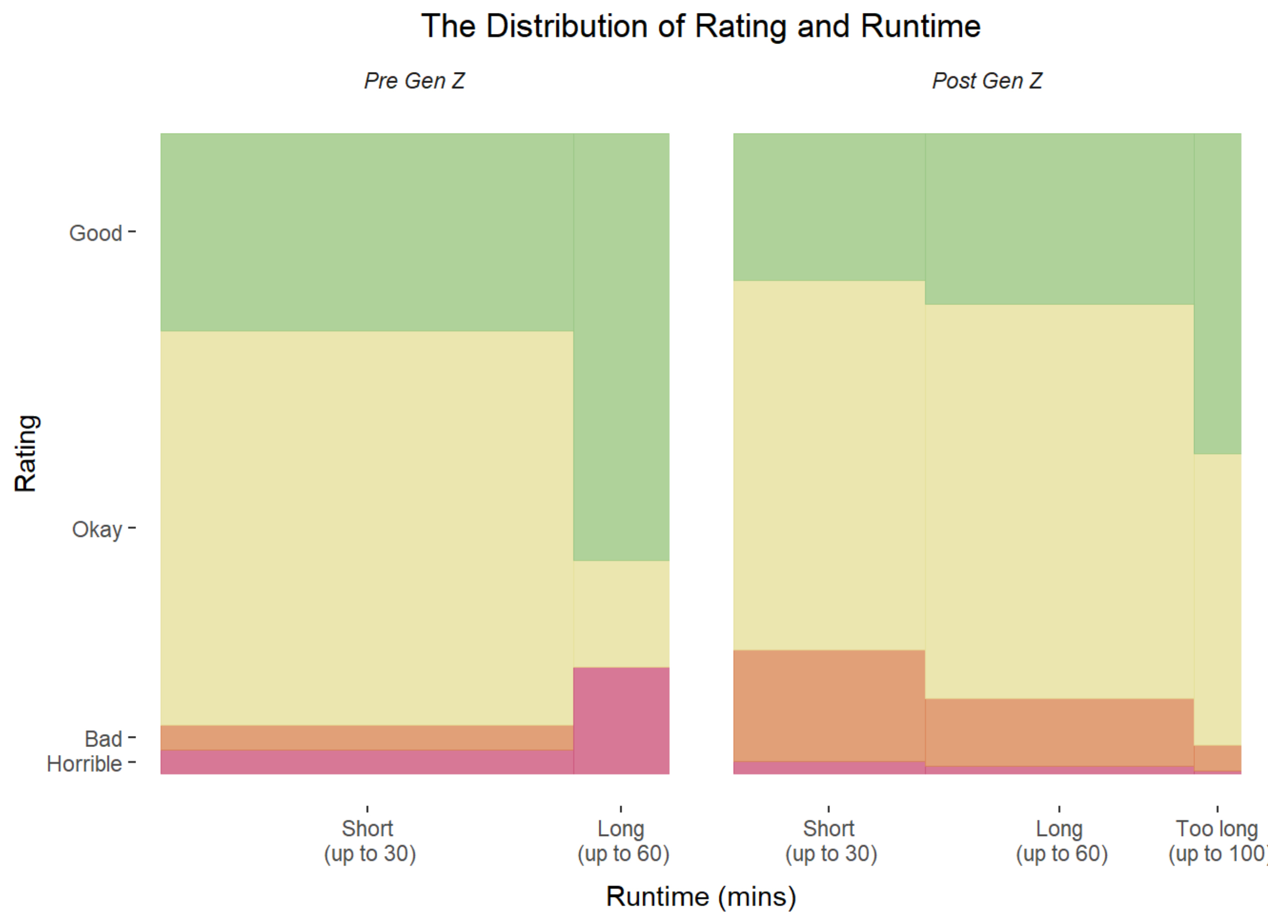

Fig 4. The distribution of the number of ratings and runtime of shows released before and after year 2000.

As we can see, the majority of shows released before 2000 are under 30 minutes, with a small portion of shows ranging from 30-60 minutes. In contrast, after 2000, we can see that we have a larger majority of shows up to 60 minutes in length, and shows over 1 hour in runtime begin to appear. Although the runtime and rating do not have a strong correlation, we can see that the addition of more shows in post *Gen Z* account for more **bad** ratings and less **horrible** ratings.

Visualizing the Runtime Over the Years

```
titles$release_year |> range()
```

```
## [1] 1945 2022
```

Finding the average runtime across each type over the year range:

```
# Store avg. runtime mean of movies and shows
runtime_mean <- titles |>
  group_by(type) |>
  summarise(r_mean = mean(runtime, na.rm = TRUE)) |>
  print()
```

```
## # A tibble: 2 × 2
##   type  r_mean
##   <chr>  <dbl>
## 1 MOVIE   98.2
## 2 SHOW    39.0
```

How does the runtime of TV shows and movies change over the years?

```
# Plot with y-line representing the mean runtime
rs <- ggplot(titles, aes(x = release_year, y = runtime, colour = type)) +
  geom_point(aes(colour = type, alpha = 0.5)) +
  geom_smooth(method = "loess", span = 0.7, show.legend = FALSE) +
  geom_hline(data = runtime_mean, aes(yintercept = r_mean, col = type),
             linetype = "solid", size = 1, color = c("black")) +
  scale_x_continuous(breaks = seq(1970, 2020, 10), limits = c(1970, 2022)) +
  scale_colour_manual(values = c("coral", "mediumorchid")) +
  facet_wrap(.~type, scales = "free_y") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5, vjust = 1.2),
        axis.title.x = element_text(vjust = -1.2),
        legend.position = "none") +
  labs(x = "Release Year",
       y = "Runtime (mins)",
       colour = "Type",
       title = "Runtime of Motion Pictures by Year")

rs
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

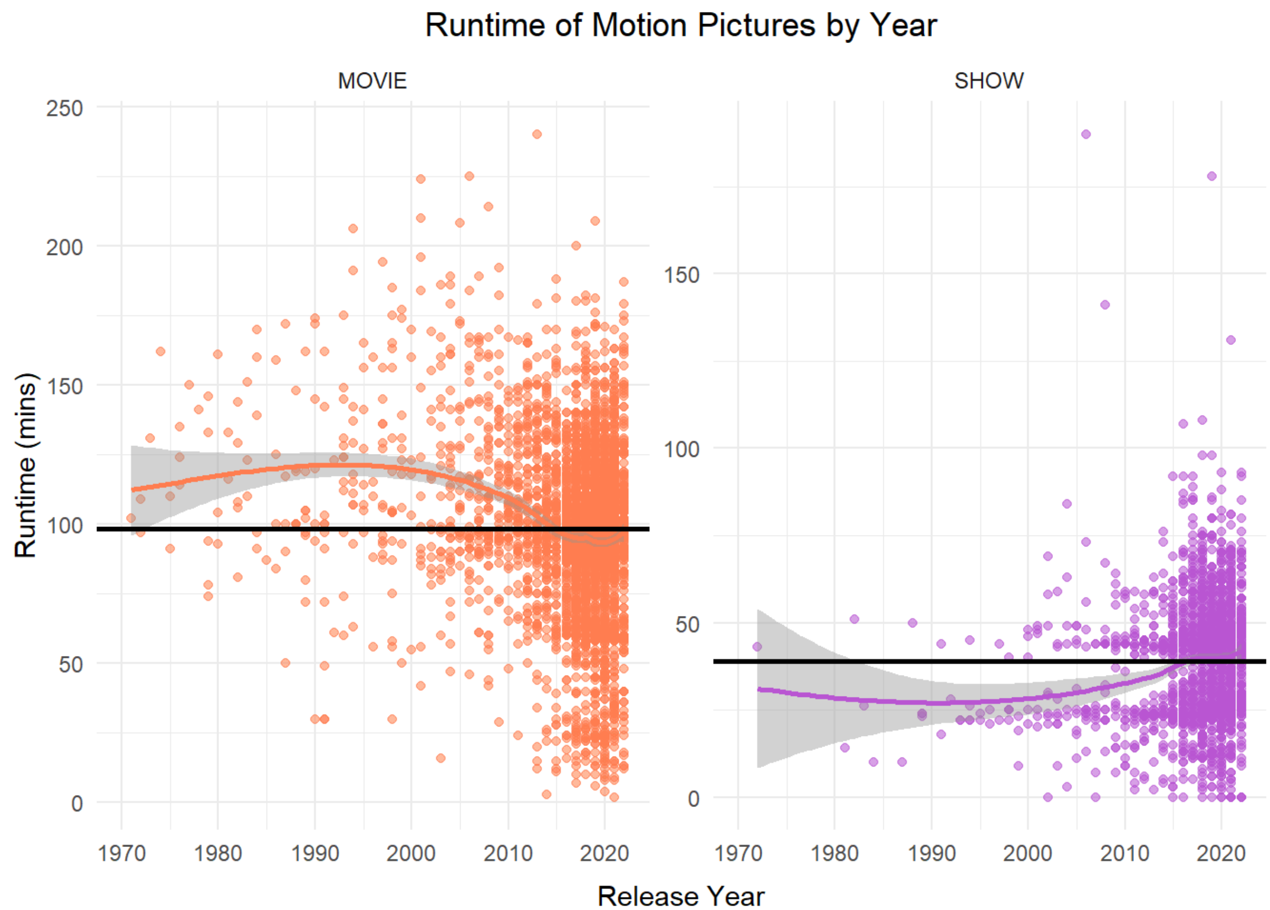

Fig 5. The changes in runtime of shows and movies by release year from 1970-2022.

- In this visual, I have used a loess regression line[8] to fit the scattered data points.
- I have also included a black `geom_hline()` on the y-axis to represent the average runtime for each facet.
- Evidently, **movies** released from 1970-2010 are longer than the total average runtime. The runtime peaks around 2000 and by ~2015, the runtime falls below the average and increases again closer to 2022.
- In contrast, we can see that the runtime for **TV shows** gradually increases around the year 2000 and surpasses the average as we get closer to 2020.
- If we compare this observation to **Fig 4.** in Visualizing Rating vs. Runtime, the facet for runtime after year 2000 conveys the same trend: a significant portion of TV shows released after 2000 have longer runtimes (around 60 minutes).

References

1. **Dataset:** Soeiro, V. (2022, July 26). Netflix TV shows and Movies. Kaggle. Retrieved March 29, 2023, from https://www.kaggle.com/datasets/victorsoeiro/netflix-tv-shows-and-movies (https://www.kaggle.com/datasets/victorsoeiro/netflix-tv-shows-and-movies)
2. **Source code for map plot:** Penir, S. (2019, January 6). Making maps with GGPLOT2. Sarah's Notes. Retrieved March 29, 2023, from https://sarahpenir.github.io/r/making-maps/ (https://sarahpenir.github.io/r/making-maps/)

---

1. The title ID on JustWatch (https://www.justwatch.com/)↩
2. SHOW or MOVIE↩
3. The number of seasons of a show↩

4. The IMDb ID on IMDb (https://www.imdb.com/)↩
5. The TMDB rating on TMDB (https://www.themoviedb.org/?language=en-CA)↩
6. ACTOR or DIRECTOR↩
7. The function calculates the value of each entry in a table as a proportion of all values↩
8. Locally weighted smoothing: used in regression analysis to fit a line through a plot (e.g. scatter) to define a relationship between attributes and predict trends↩