Lunch Hour Learning Guide, Sessions 11-12, Spring 2025

Data Visualization in R - Parts 1 and 2

Sean Morey Smith

2025-04-23

Before Starting

1. Create a new, self-contained R project where you will store your work from this session. For guidance, see the instructions from Session 1.
2. Create a sub-directory (folder) called "data" in your project directory.
3. Unzip the .csvs in https://library.rice.edu/sites/default/files/materials/data.zip into the "data" sub-directory.

Session 11: Data Visualization in R - Part 1

What You Will Learn

- How the layered approach of the grammar of graphics contributes to plot structure
- How to create a histogram
- How to modify colors in the plot
- How to modify plot axes
- How to add annotations and titles
- How to use themes to customize the look of the plot

The Grammar of Graphics

Wilkinson (2005) proposed a series of "rules" for creating virtually all graphical forms. The package `ggplot`, developed by Hadley Wickham, is based on these rules and takes a layered approach to creating visualizations. The basic format involves specifying data, aesthetic mapping (i.e., how variables will be represented in the visual space), a coordinate system, and various layers including geoms (visual objects), scales, and so forth.

In these two sessions, we will focus on using `ggplot2` to create various types of graphs, including a histogram, a bar plot, a scatterplot, and more. You can create numerous other types of graphs with `ggplot2`, such as density plots, violin plots, maps, and many more.

The lesson will demonstrate the "layered" approach by beginning with the simplest form of a graph and then enhancing it with various changes to the code. Note that you will save your changes to new plot objects as you go along; however, once you become comfortable with the functions, you can combine numerous layers into the code for a single plot, thereby saving time and memory.

The Data: Netflix Movies and TV Shows

For this lesson, you will work with a dataset originally created by Soero (2022) and shared in Kaggle. The dataset has been cleaned and includes a limited number of variables for all movies and TV shows that were aired on Netflix in 2019. Of particular interest will be the movie/show genres, their length (i.e., runtime), and their average IMDB and TMDB scores.

Import the Dataset and Load the Tidyverse

```
library(tidyverse) # ggplot2 loads as part of tidyverse
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.0     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
titles <- read.csv("data/titles.csv", stringsAsFactors = FALSE)
```

Create a Histogram

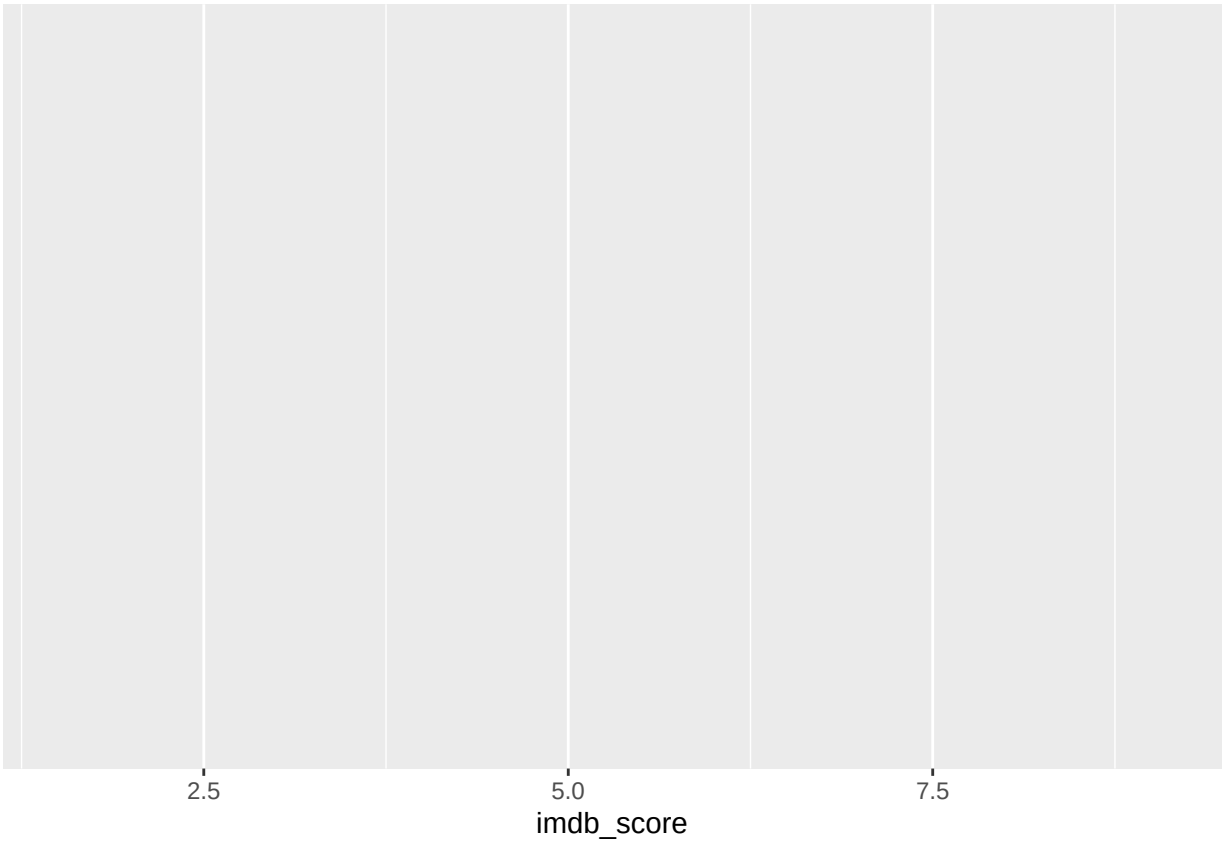
Begin by filtering the data to include only movies, save this as an object called movies, and look at the structure of the object.

```
movies <- titles %>%
        filter(type == "MOVIE")
str(movies)
```

```
## 'data.frame':    3744 obs. of  9 variables:
##  $ title       : chr  "Taxi Driver" "Deliverance" "Monty Python and the Holy Grail" "The Dirty Dozen
##  $ type        : chr  "MOVIE" "MOVIE" "MOVIE" "MOVIE" ...
##  $ release_year: int  1976 1972 1975 1967 1979 1971 1967 1980 1961 1966 ...
##  $ runtime     : int  114 109 91 150 94 102 110 104 158 117 ...
##  $ genre       : chr  "drama" "drama" "fantasy" "war" ...
##  $ country     : chr  "US" "US" "GB" "GB" ...
##  $ rating      : chr  "R" "R" "PG" "none" ...
##  $ imdb_score  : num  8.2 7.7 8.2 7.7 8 7.7 7.7 5.8 7.5 7.3 ...
##  $ tmdb_score  : num  8.18 7.3 7.81 7.6 7.8 ...
```

Next, create a basic plot that only contains the data and the aesthetic mappings with the x values set as `imdb_score`. Note that the plot will be empty because you've just told R what data to use but not how to display it; however, this step can save you a bit of time later.

```
base_plot <- ggplot(data = movies, mapping = aes(x = imdb_score))
base_plot
```

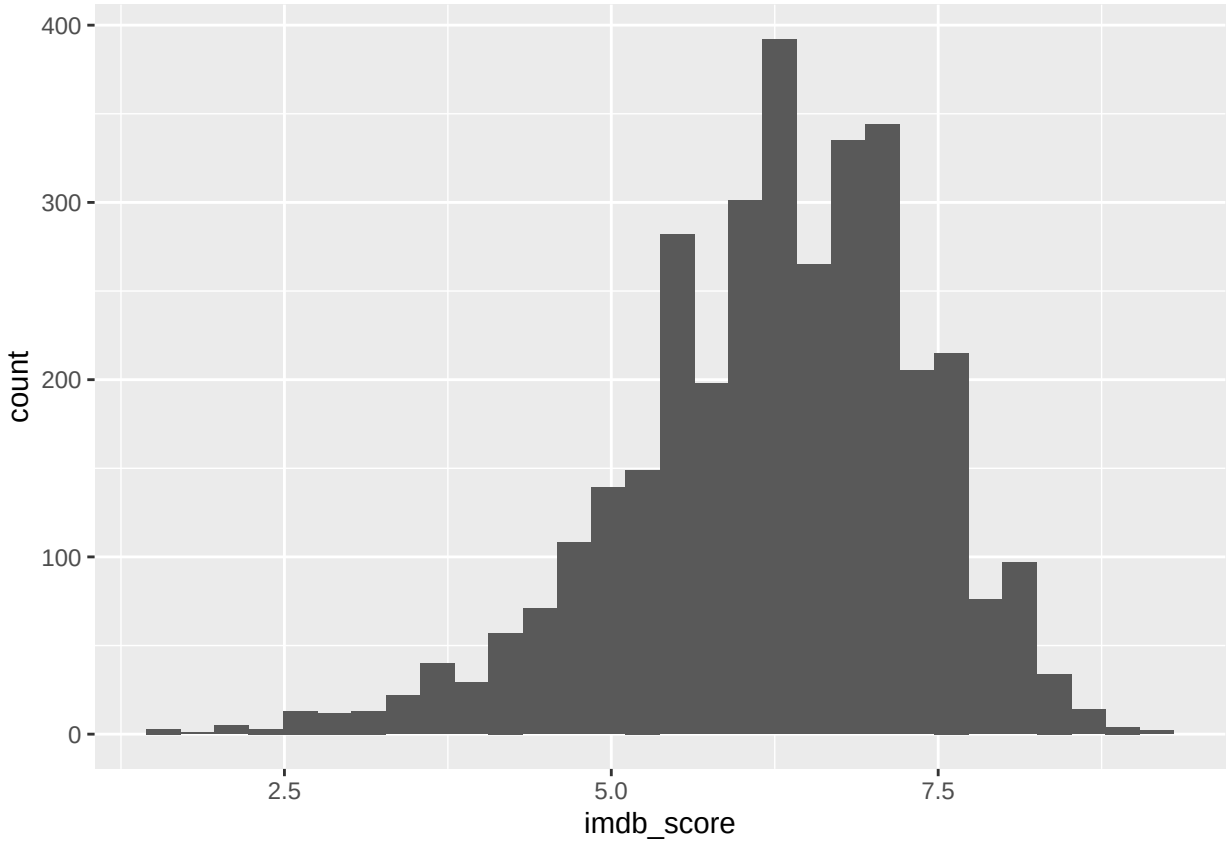


Add the `Geom` Layer

The next step is to add the `geom` layer. Unlike other `tidyverse` packages in which the pipe (`%>%`) is used to show subsequent steps, in `ggplot2` layers are added with a `+`. However, note that the main `ggplot()` function can be piped into a longer pipe using `%>%`. (An example is shown in the section on scatter plots.)

```
base_plot +
        geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

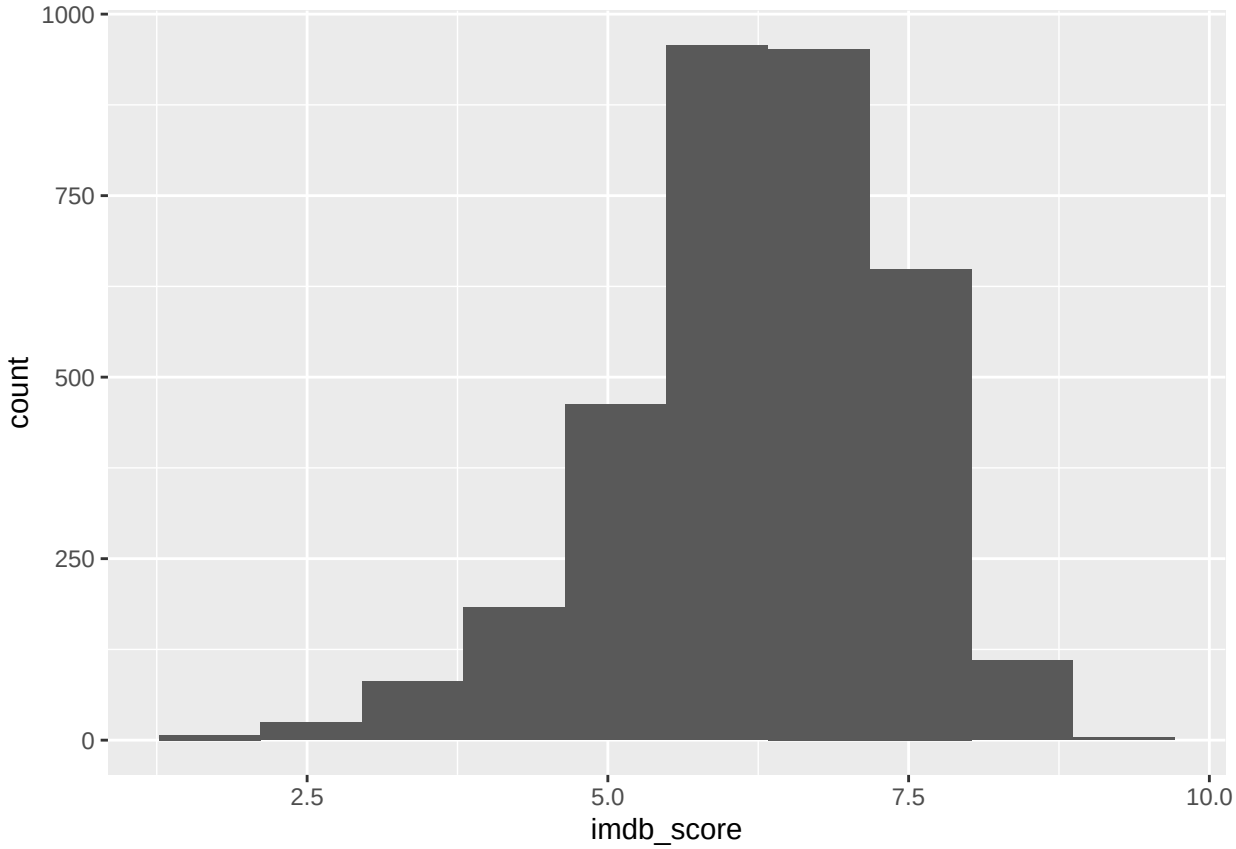


Adding the `geom` layer creates a histogram of the variable `imdb_score`, which was specified as `x` when we created `base_plot`.

Change the Appearance of the Distribution

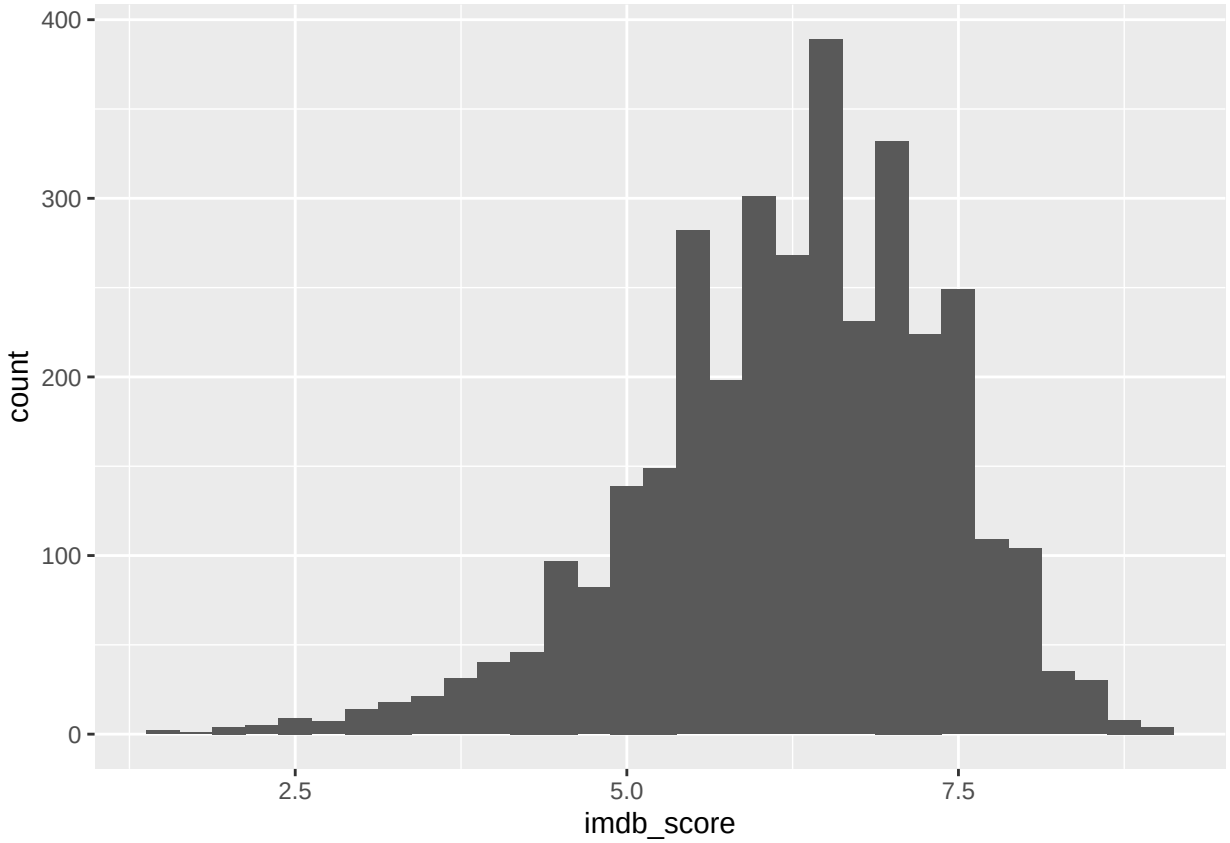
You can change the appearance of the distribution by adding an argument within `geom_histogram()`. There are two options. First, change the number of bins:

```
base_plot +
        geom_histogram(bins = 10)
```



This will adjust the number of bins into which the data are sorted. Alternatively, change the bin width, relative to the scale of the x axis:

```
base_plot +
        geom_histogram(binwidth = 0.25)
```

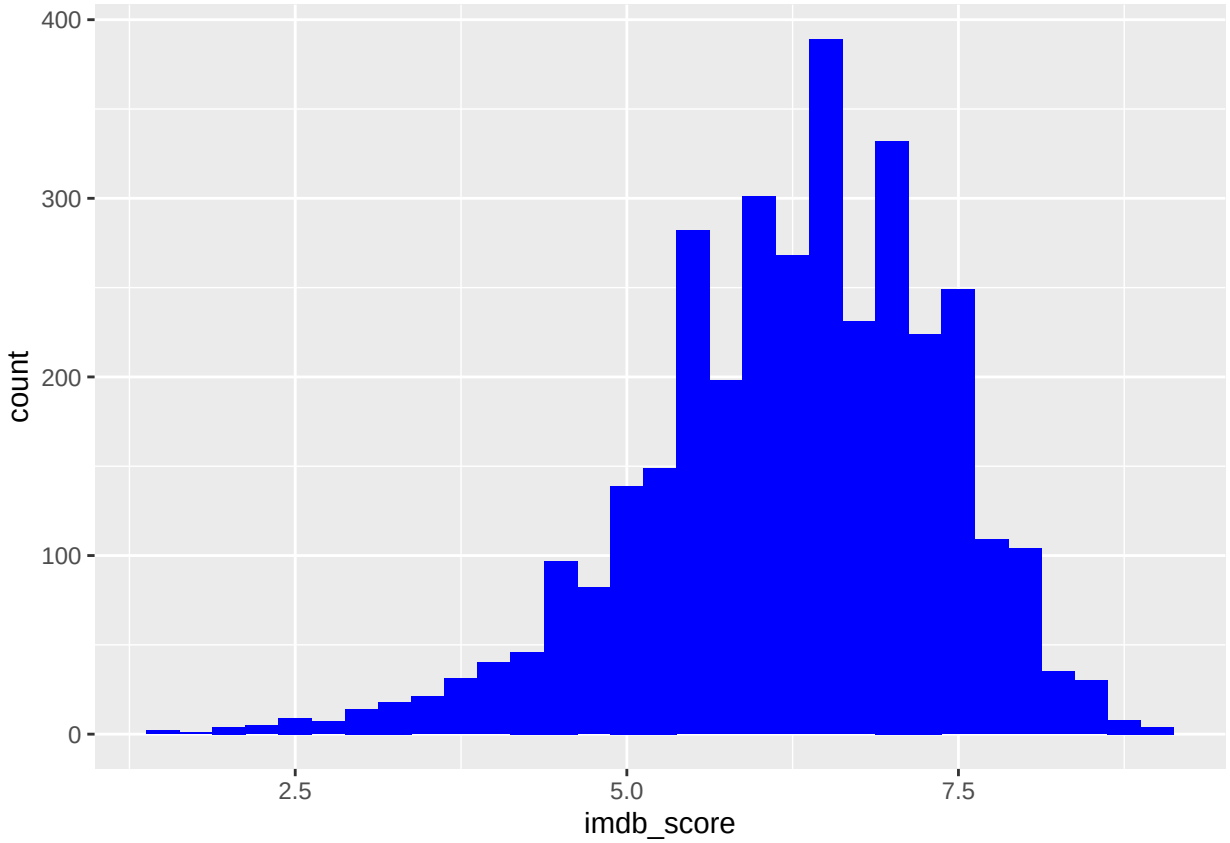


This will adjust the width of the bins. Practice varying the number of bins or bin width until you find one that you like best.

Change the Color of the Bars

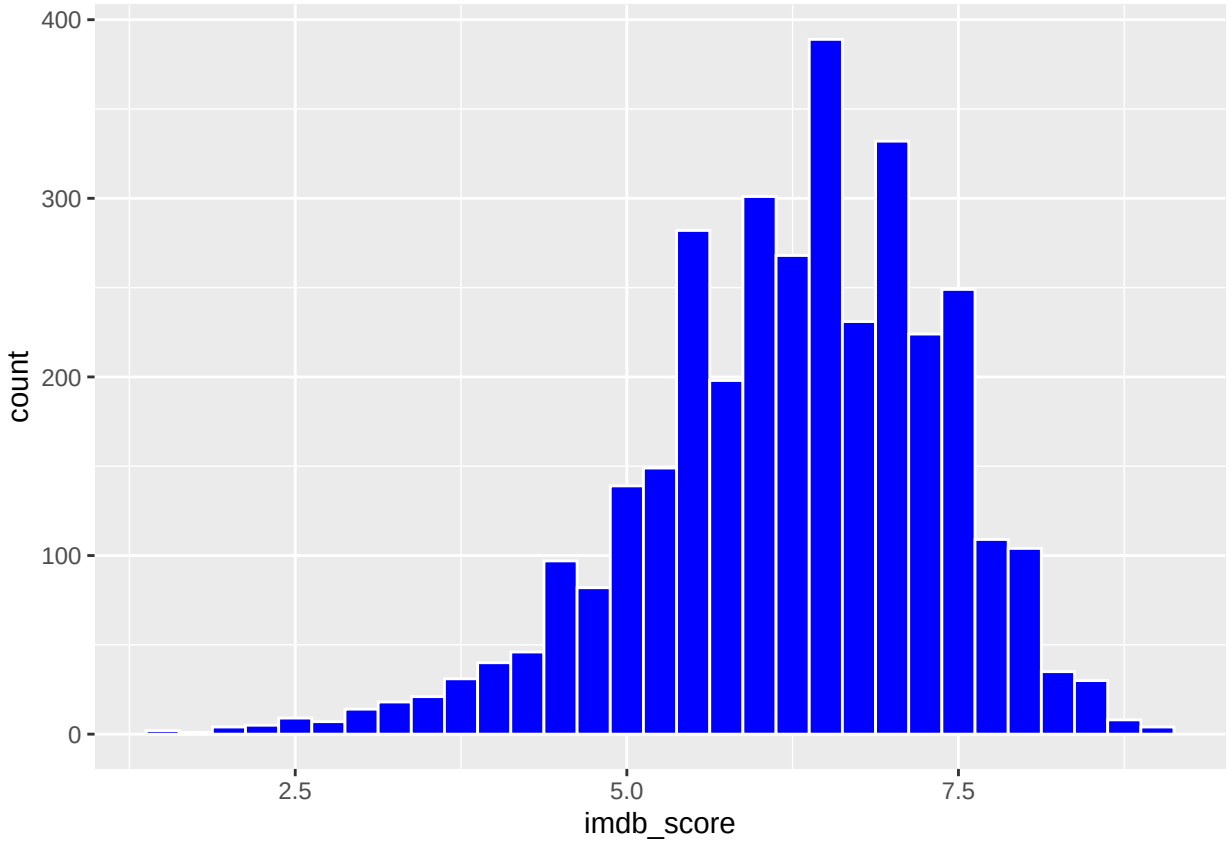
We will explore color in more depth in the next session, but let's take a look at some basic color changes to make the histogram look better. The default in `ggplot2` is grayscale, but you aren't limited to that! For color names, check out Wei's (2021) R Color Cheat Sheet.

```
base_plot +
        geom_histogram(binwidth = .25, fill = "blue")
```



Note that the argument `fill = "blue"` will make both the bars and their borders blue. However, you may want to make the bars and their borders different colors for separation and emphasis.

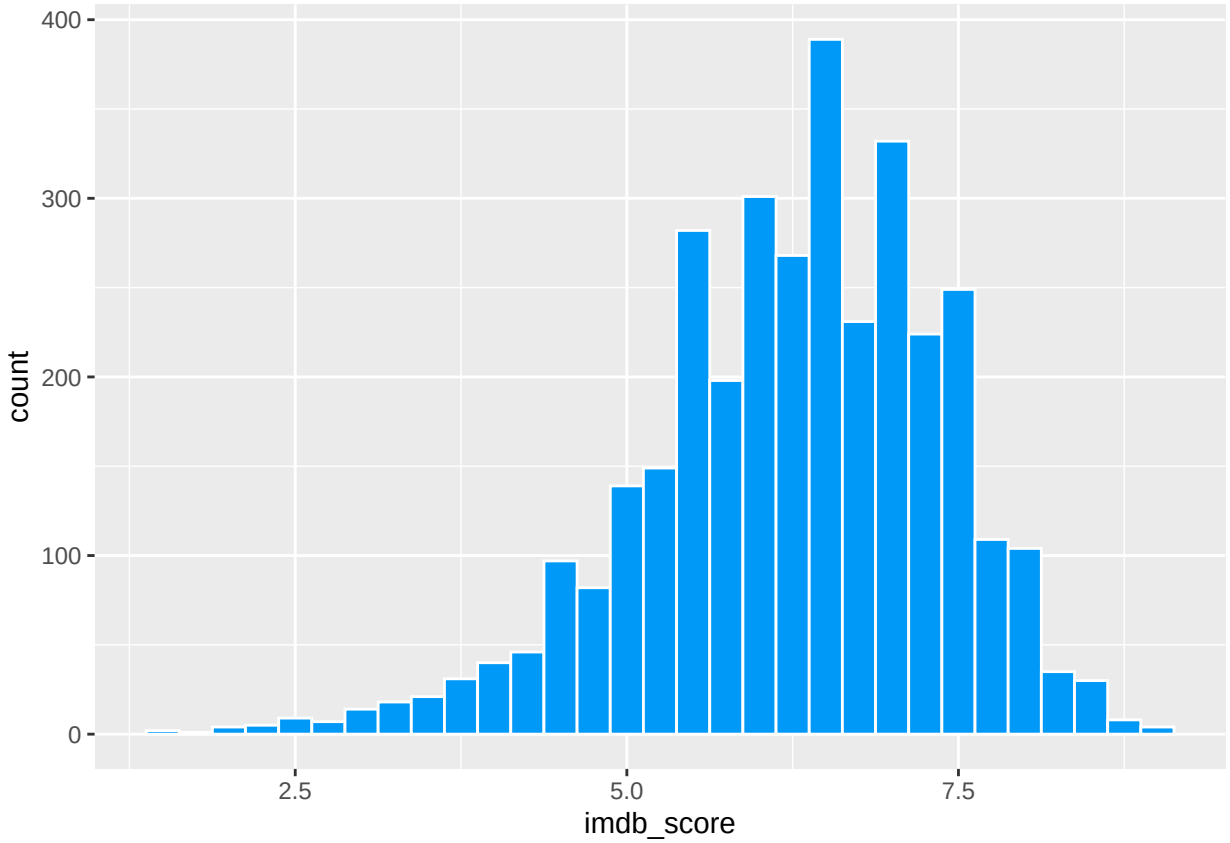
```
base_plot +
        geom_histogram(binwidth = .25, fill = "blue", color = "white")
```



Now the individual bars "pop" a bit more because of the different border color.

Finally, try adding a hex code instead of a color name and save the plot as `hist_base`. One resource for hexcodes is https://htmlcolorcodes.com.

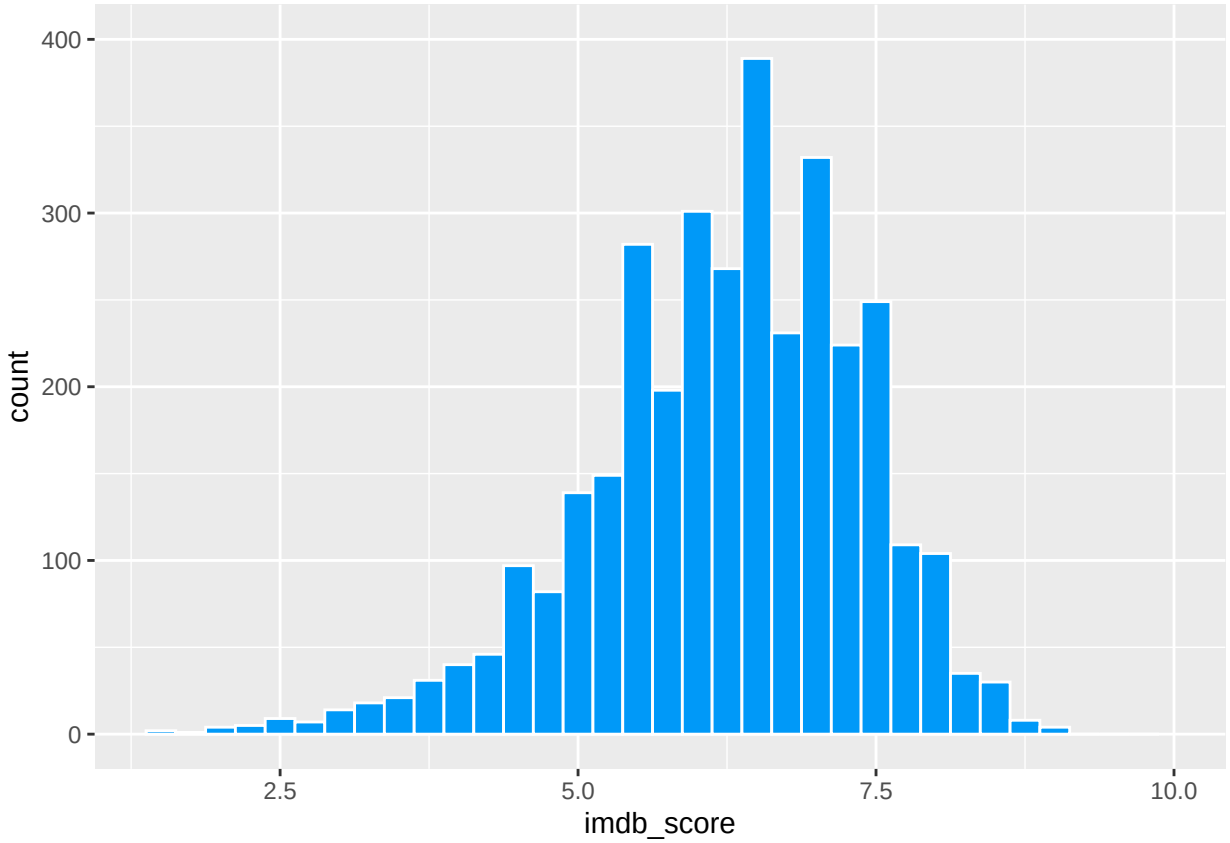
```
hist_base <- base_plot +
        geom_histogram(binwidth = .25, fill = "#0099F8", color = "white")
hist_base
```



Change the Axis Limits

Sometimes you will want to modify the limits of the x axis and/or y axis. This is a somewhat imprecise process for histograms, but the following example shows how to accomplish this. Keep in mind that the number/width of bins will contribute to how R organizes the x axis. In this example, you will set the lower limit of the x axis to `NA`, which tells R to set the lower limit according to the data. However, you will set the lower limit of y to 0. Save this plot as `hist_limits`.

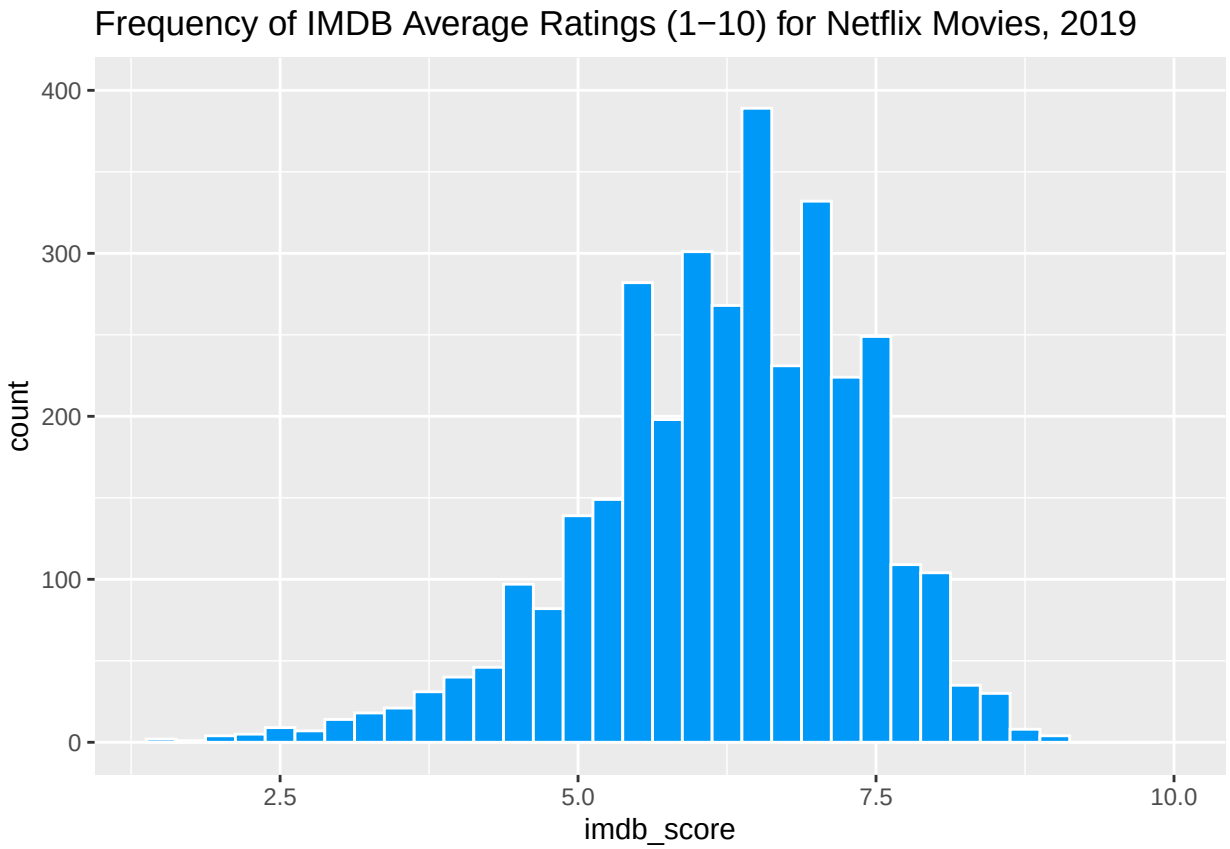
```
hist_limits <- hist_base +
        xlim(NA, 10) +
        ylim(0, 400)
hist_limits
```



Add a Main Title

No graph is complete without a title! A quick way to add a title is to use the `ggtitle()` function. Save the plot to `hist_title`.

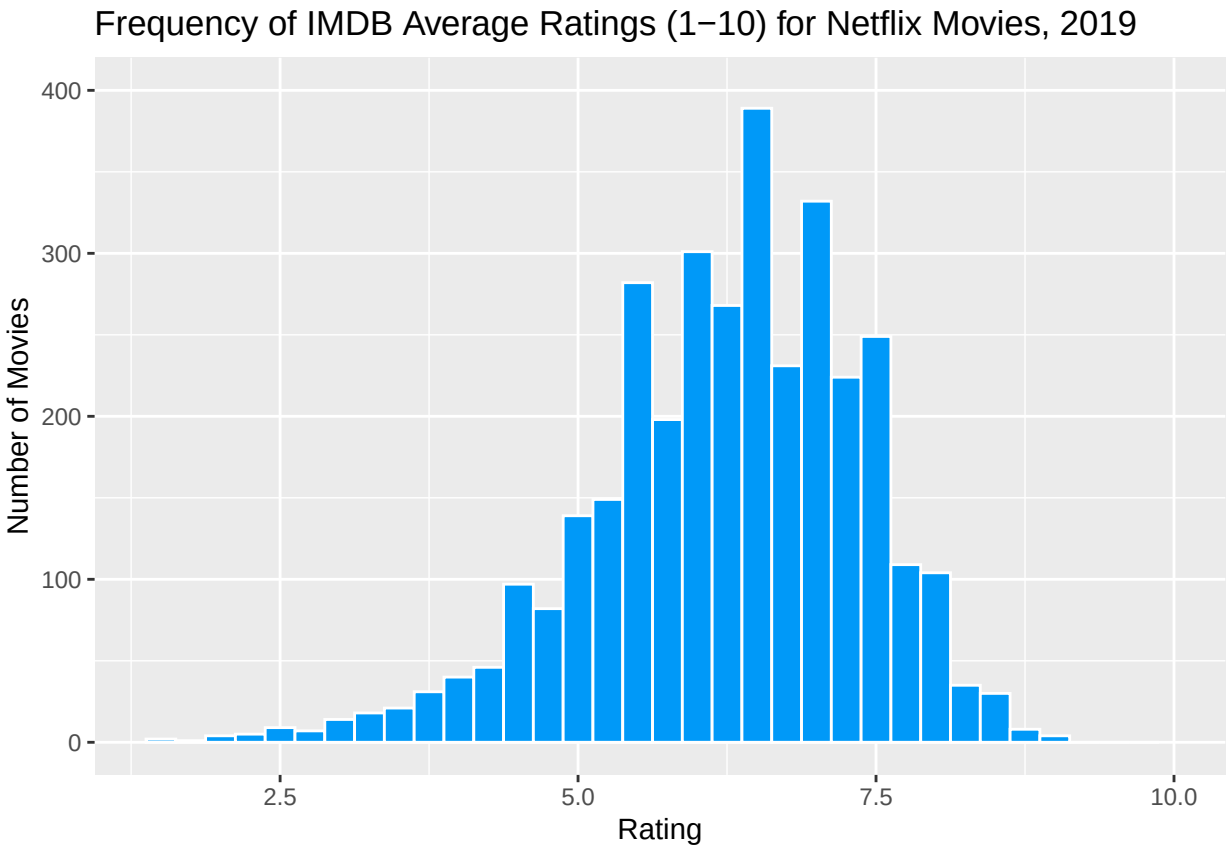
```
hist_title <- hist_limits +
        ggtitle("Frequency of IMDB Average Ratings (1-10) for Netflix Movies, 2019")
hist_title
```



Change the X and Y Axis Labels

The default in `ggplot2` is to use the variable names as x and y axis labels. Use the `labs()` function to change these:

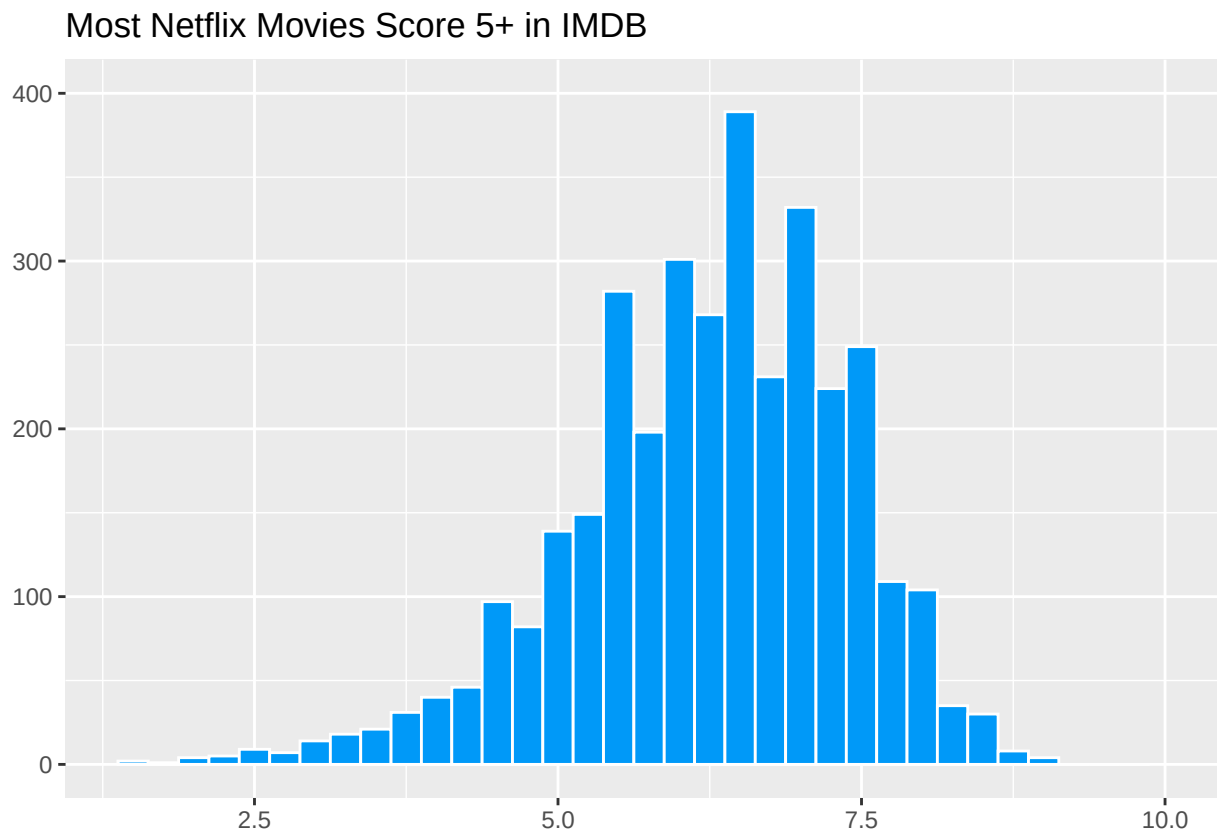
```
hist_title +
        labs(x = "Rating",
             y = "Number of Movies")
```



Remove X and Y Axis Labels

If you have already used `ggtitle()` in a base plot but want to modify the title, there is no need to create a new base plot. Simply override the title with a new title using the `labs` function. With a bit of tweaking to the title, you can remove the now-redundant x and y axis labels. Remove labels by setting their value with the `element_blank()` function. Save this plot as `hist_title_revised`.

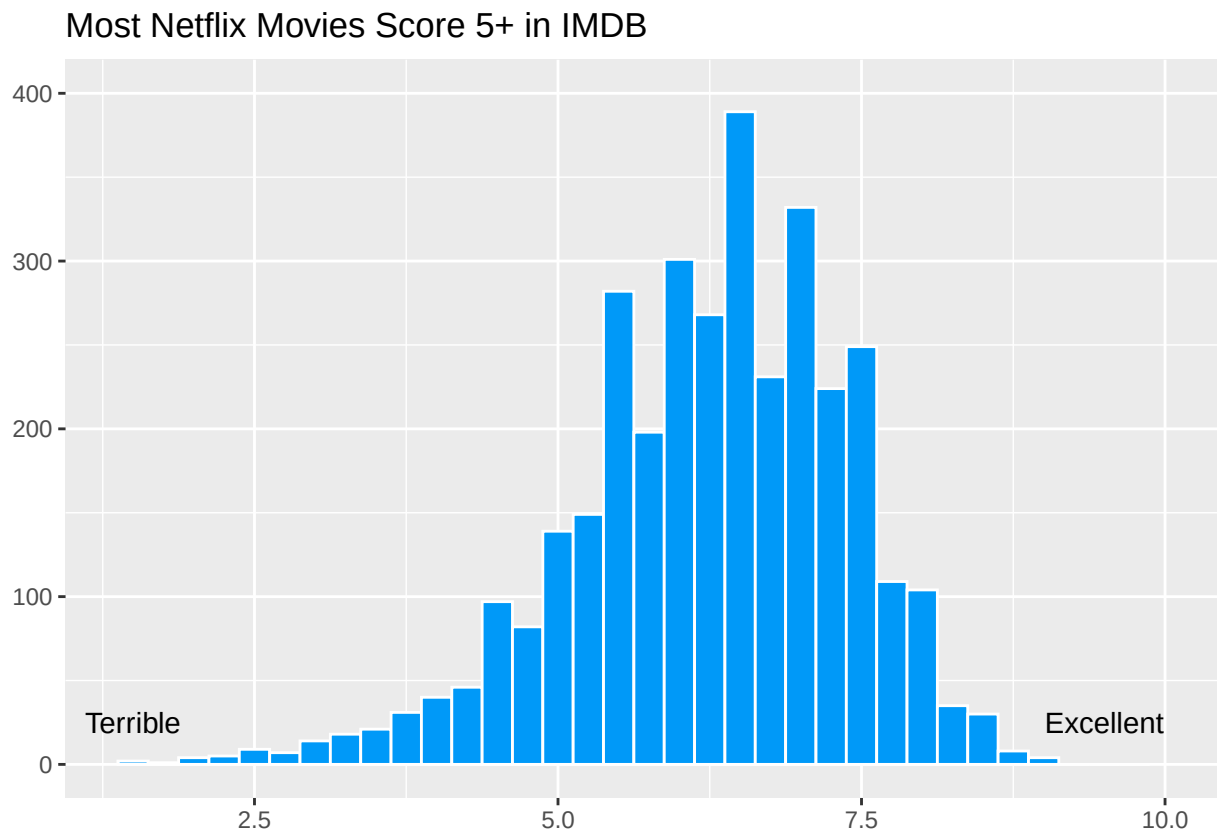
```
hist_title_revised <- hist_title +
        labs(title = "Most Netflix Movies Score 5+ in IMDB",
             x = element_blank(),
             y = element_blank())
hist_title_revised
```



Add Annotations

Perhaps you want to add a bit of context to the ratings. You can annotate parts of the plot by specifying the text and coordinates within the `annotate()` function. Save the plot to `hist_annotated`.

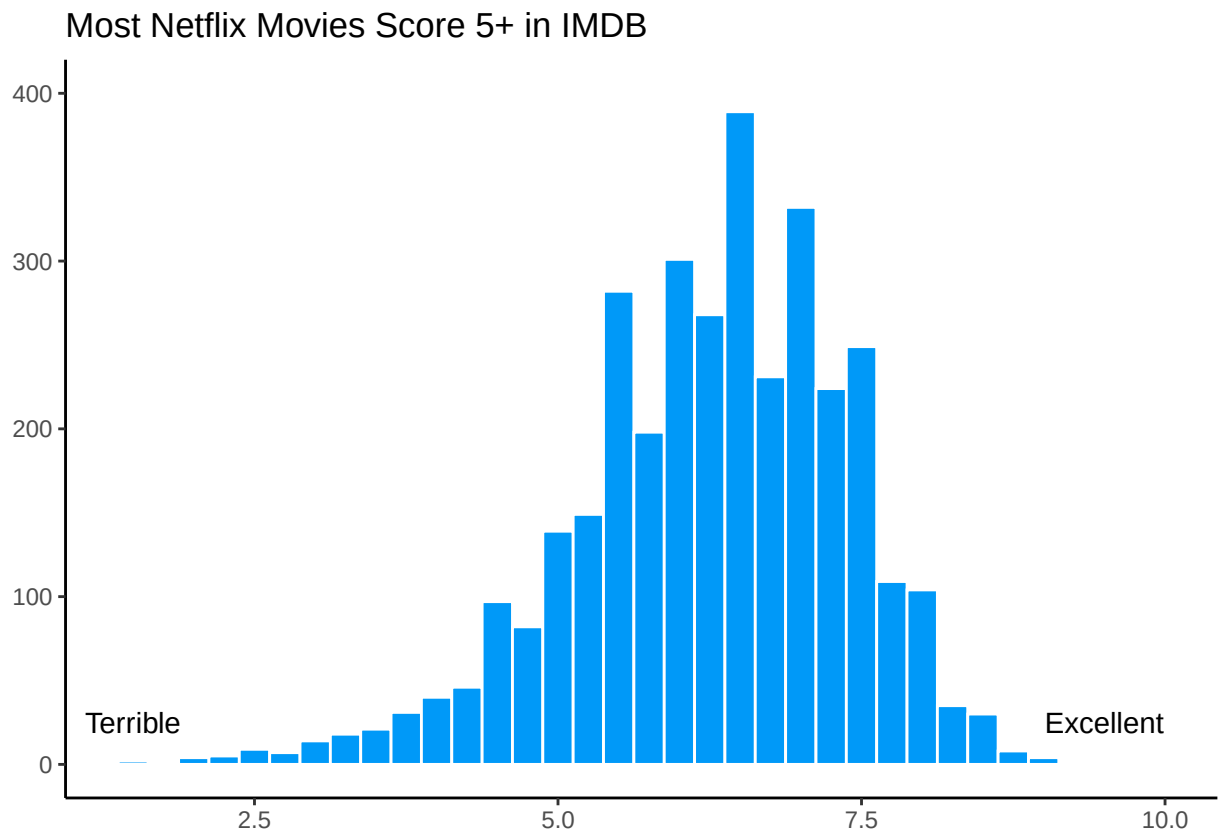
```
hist_annotated <- hist_title_revised +
        annotate("text",
                 x = c(1.5,9.5),
                 y = c(25,25),
                 label = c("Terrible", "Excellent"))
hist_annotated
```



Explore Themes

The `ggplot2` package contains several "themes" that allow for quick alterations to the default plot. Try modifying the plot you have just created by adding the `theme_classic()` layer.

```
hist_annotated +
        theme_classic()
```

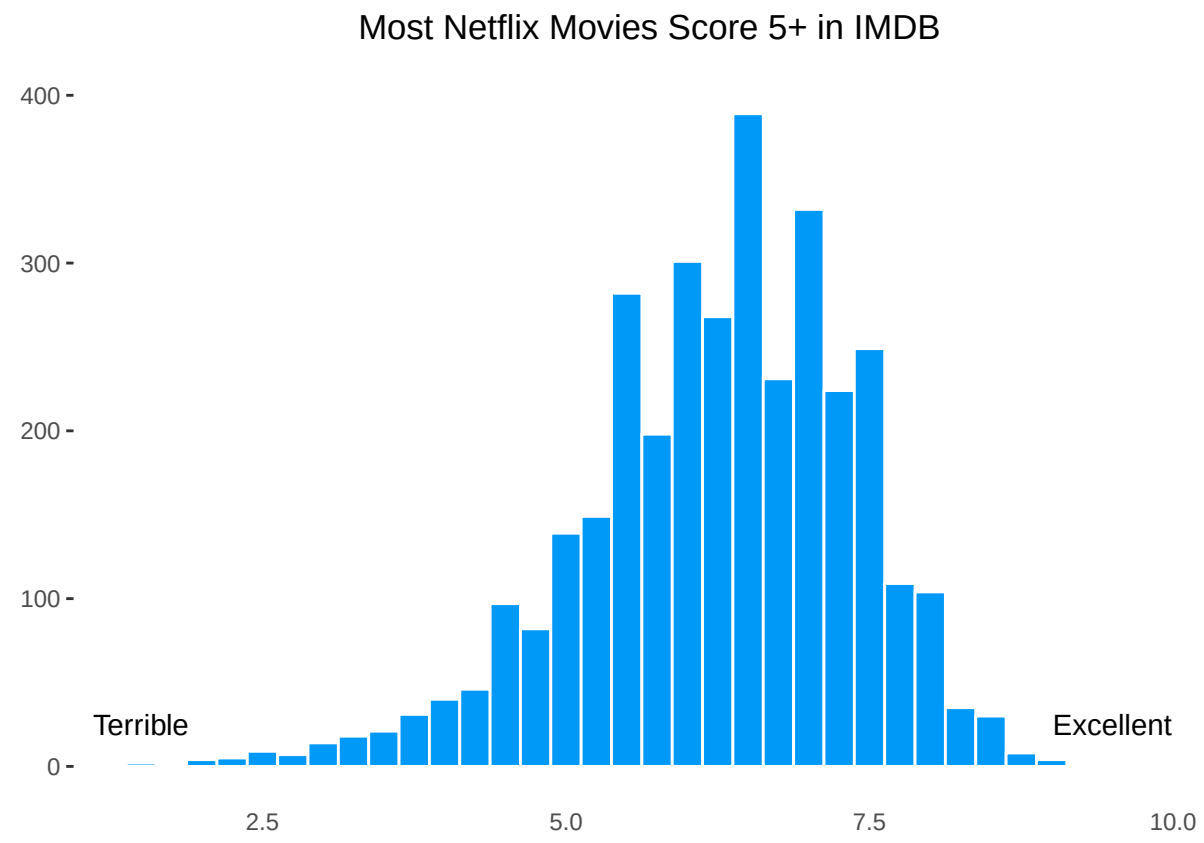


The main changes you will note are that the background is no longer a gray and white grid, and the axes are now black. The broader application of themes comes with the function `theme()`, which can take many arguments about all sorts of plot elements. See Henry Wang's ggplot2 Theme Elements Demonstration for more ideas!

Additional themes include `theme_gray()`, `theme_bw()`, `theme_linedraw()`, `theme_light()`, `theme_dark()`, `theme_minimal()`, and `theme_void()`. Check out how they change the look of the plot!

The final aspect of this session is to tinker with the elements of the plot. Note that you will start with the `hist_annotated` plot and remove the background, axis lines, and x-axis ticks manually. You will also center the title.

```
final_plot <- hist_annotated +
        theme(panel.border = element_blank(),
              panel.grid.major = element_blank(),
              panel.grid.minor = element_blank(),
              panel.background = element_blank(),
              axis.ticks.x = element_blank(),
              plot.title = element_text(hjust = 0.5))
final_plot
```



Export a Plot

Once you are satisfied with your plot and have saved it as a plot object, you can export it with the function `ggsave()`.

```
ggsave(filename = "netflix_histogram.png", plot = final_plot, width = 12,
       height = 10, dpi = 300, units = "cm")
```

Note that you can also use the "Export" button from the Plots tab in the bottom-right miscellaneous pane.

Bonus: Install and Import `ggthemes` Package

The `ggthemes` package provides custom ggplot themes, which you may find useful. Remember, you can install a package with `install.packages("ggthemes")`.

```
library(ggthemes)
```

Functions of this package include `theme_tufte()`, `theme_economist()`, `theme_stata()`, and `theme_hc()`. I recommend applying these to plots that have not already been heavily customized, as it is easier to modify a plot that has been created with a theme simply by using the `element_blank()` function to remove unwanted changes.

Session 12: Data Visualization in R - Part 2

What You Will Learn

- How to create additional plot types
- How to add dimensions to a plot
- How to add statistical models and transformations
- How to facet on a variable

Import the Dataset and Load the Tidyverse

```
library(tidyverse)
titles <- read.csv("data/titles.csv", stringsAsFactors = FALSE)
```

Create a Scatter Plot

One of the most common plots in data science is the scatter plot, which examines the relationship between two continuously-measured variables. In this exercise, you will explore whether there is a linear relationship between the length of a TV show (i.e., runtime) and its TMDB rating.

Prepare the Data

You will limit the analysis to the top four TV show genres (by frequency), which previous work has shown to be drama, comedy, documentation, and animation. Begin by filtering the `titles` dataset for these four genres of TV shows and for `runtime` less than 100 and more than 9. Note that the `%in%` operator finds all the rows where the `genre` value is *in* the vector of strings specifying which genres we're interested in: `c("drama", "comedy", "documentation", "animation")`.

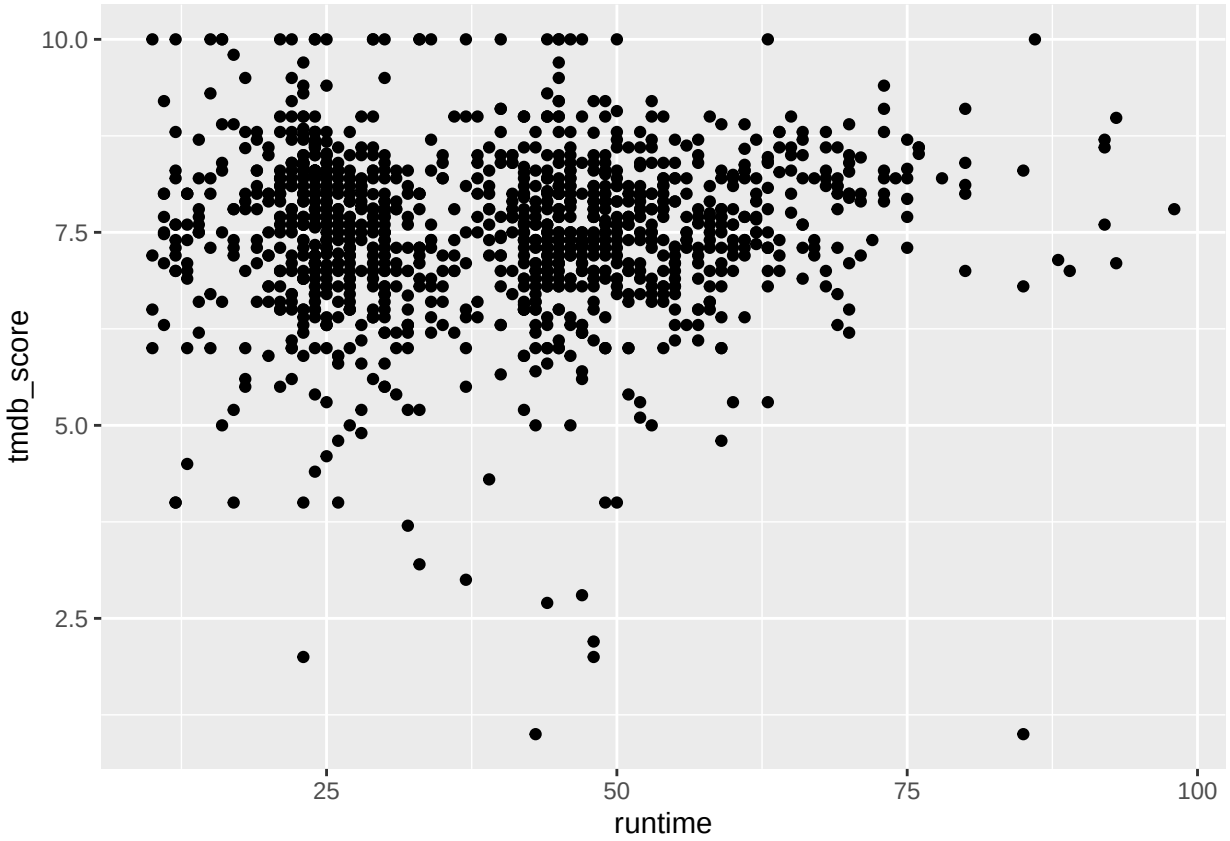
```
top_show_genres <- titles %>%
        filter(type == "SHOW"
               & genre %in% c("drama", "comedy", "documentation", "animation")
               & (runtime < 100 & runtime > 9))
```

Create the Basic Plot

From this point on, you will use a shortcut to save some time typing! You can begin a line of `ggplot2` code with the dataset name and the pipe operator (`%>%`) rather than using the argument `data = data` within the `ggplot()` function. This also helps when your plot immediately follows one or more `dplyr` functions. In addition, you can drop the mapping prefix `mapping =` before `aes()` as long as the `aes()` function appears first.

Take a look:

```
top_show_genres %>%
        ggplot(aes(x = runtime, y = tmdb_score)) +
        geom_point()
```

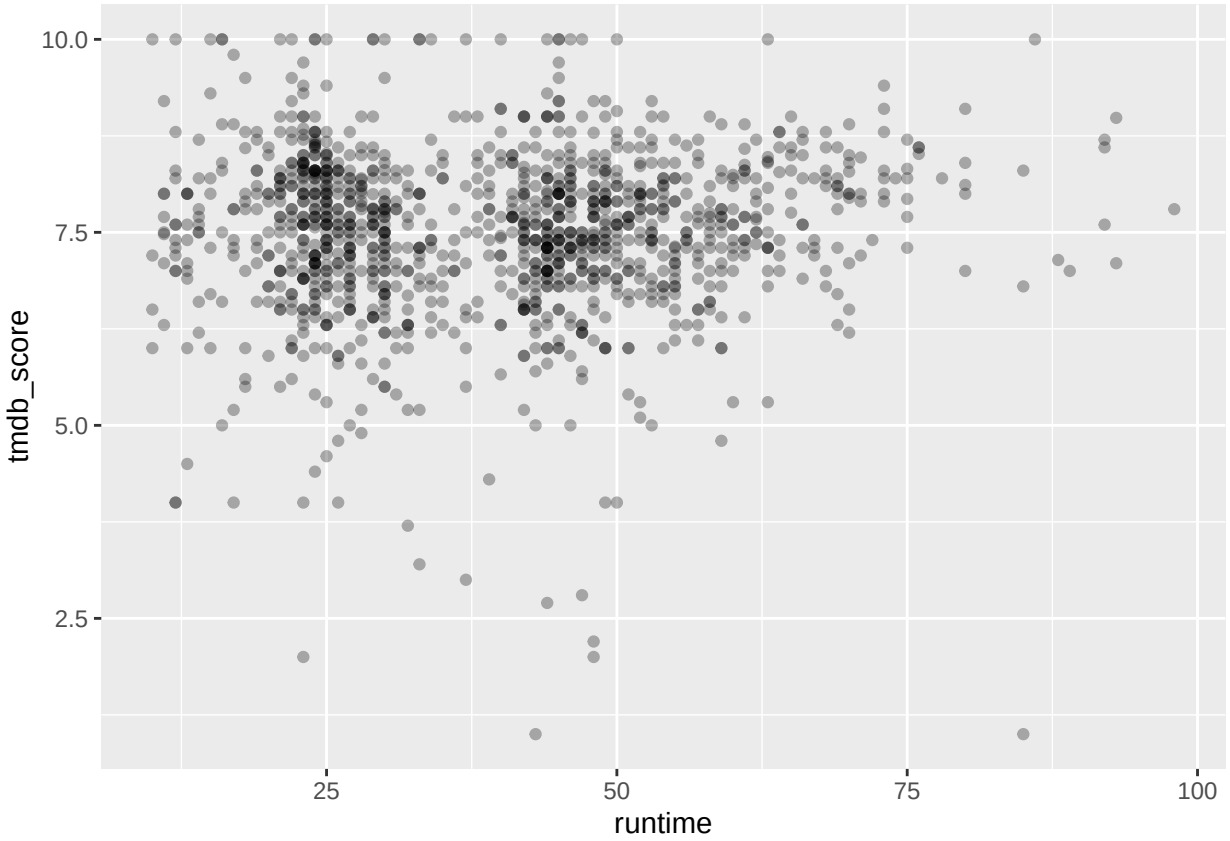


The plot shows little apparent relationship between the length of TV shows and their TMDB scores. However, there is an issue with over-plotting. You can address this problem with the argument `alpha`.

Adjust Opacity

To make the points partially transparent, the argument `alpha` can be set as a decimal anywhere between 0 and 1, with smaller numbers indicating lighter points. Try adjusting the alpha value and saving the plot as `time_tmdb_scatter1`.

```
time_tmdb_scatter1 <- top_show_genres %>%
        ggplot(aes(x = runtime, y = tmdb_score)) +
        geom_point(alpha = 0.3)
time_tmdb_scatter1
```



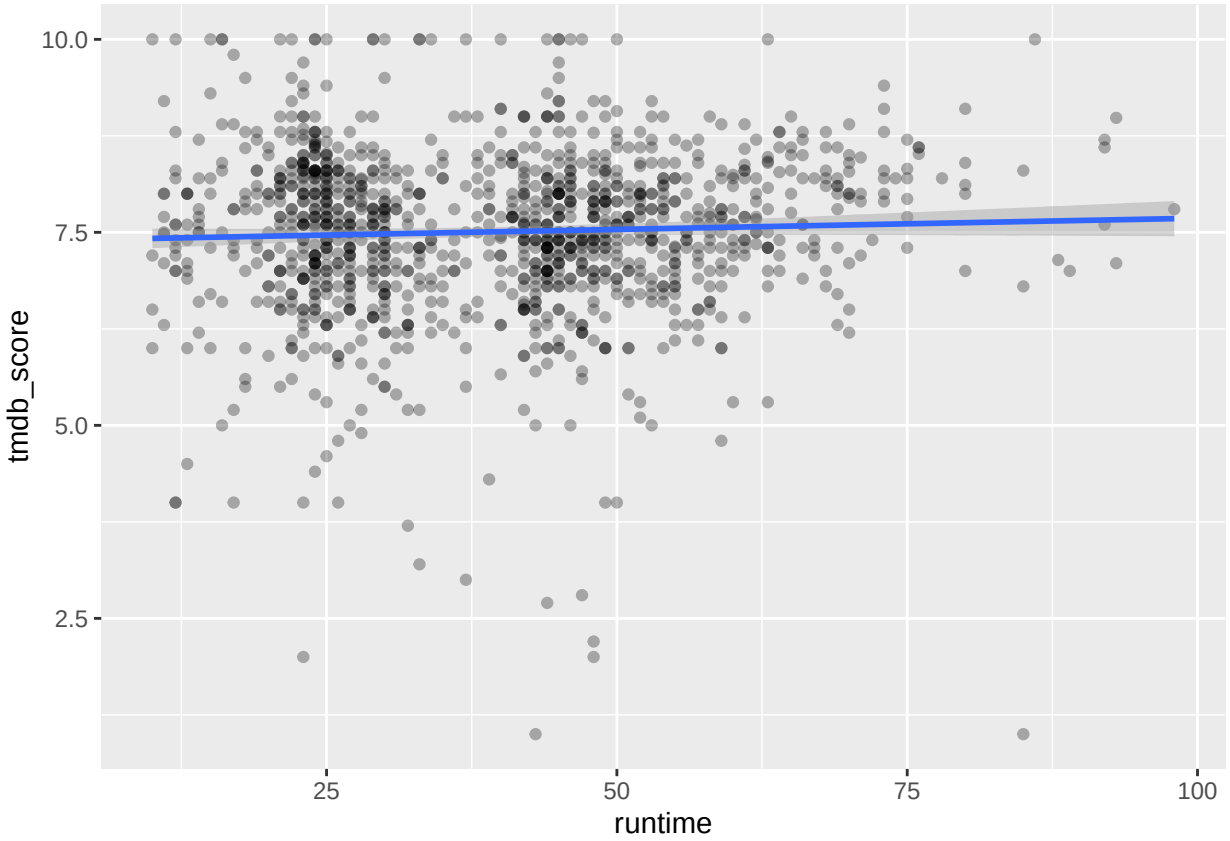
Notice that some of the points are darker, indicating an overlap where more than one TV show is plotted.

Add a Regression Line

It can be helpful to see the actual line of best fit for the relationship between x and y. To add a regression line, create an additional geom layer called `geom_smooth`; the default will include an error band around the line.

```
regression1 <- time_tmdb_scatter1 +
        geom_smooth(method = "lm") # "lm" indicates a linear regression
regression1
```

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

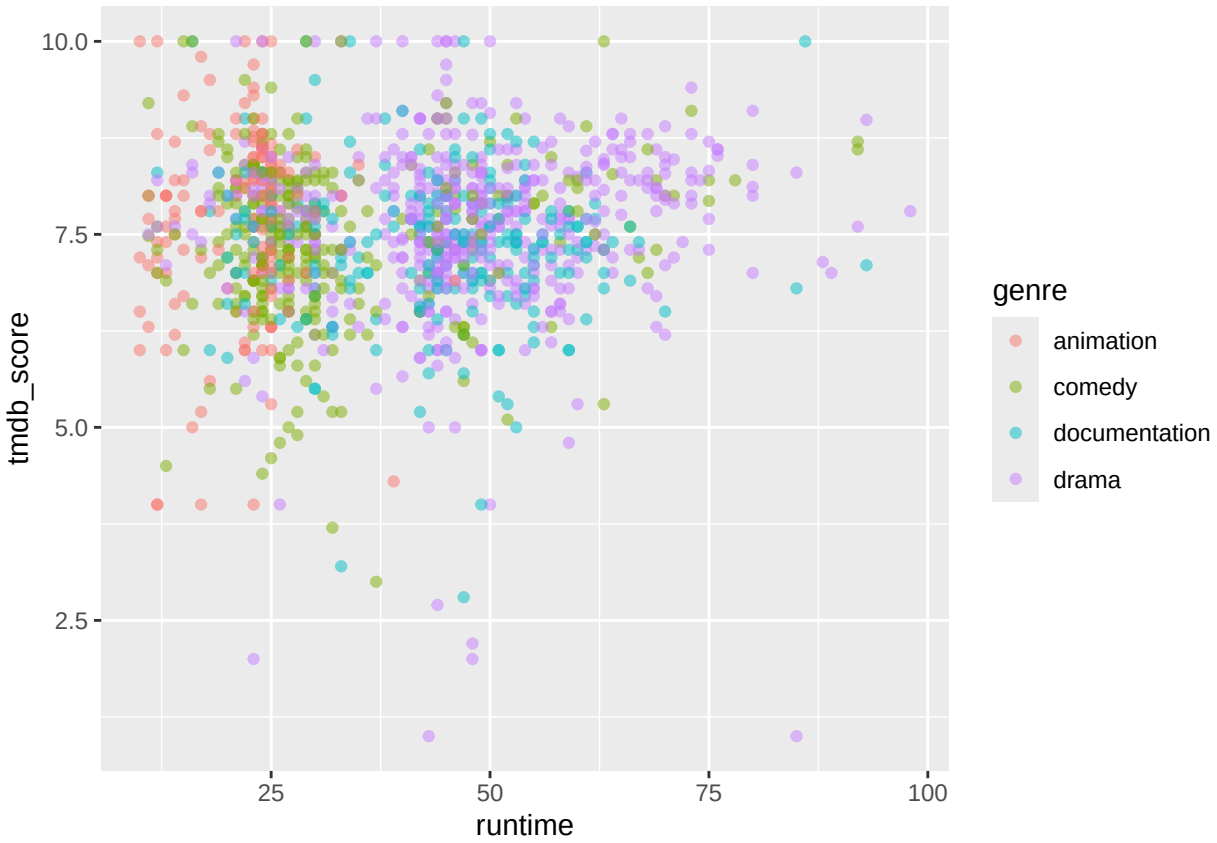


Add a Dimension

Although you can make all of the points the same color by passing a color name or hexcode to the argument `fill =` within `geom_point`, it can be useful to add a dimension by assigning a variable name to the color aesthetic.

For example, assign `genre` to color:

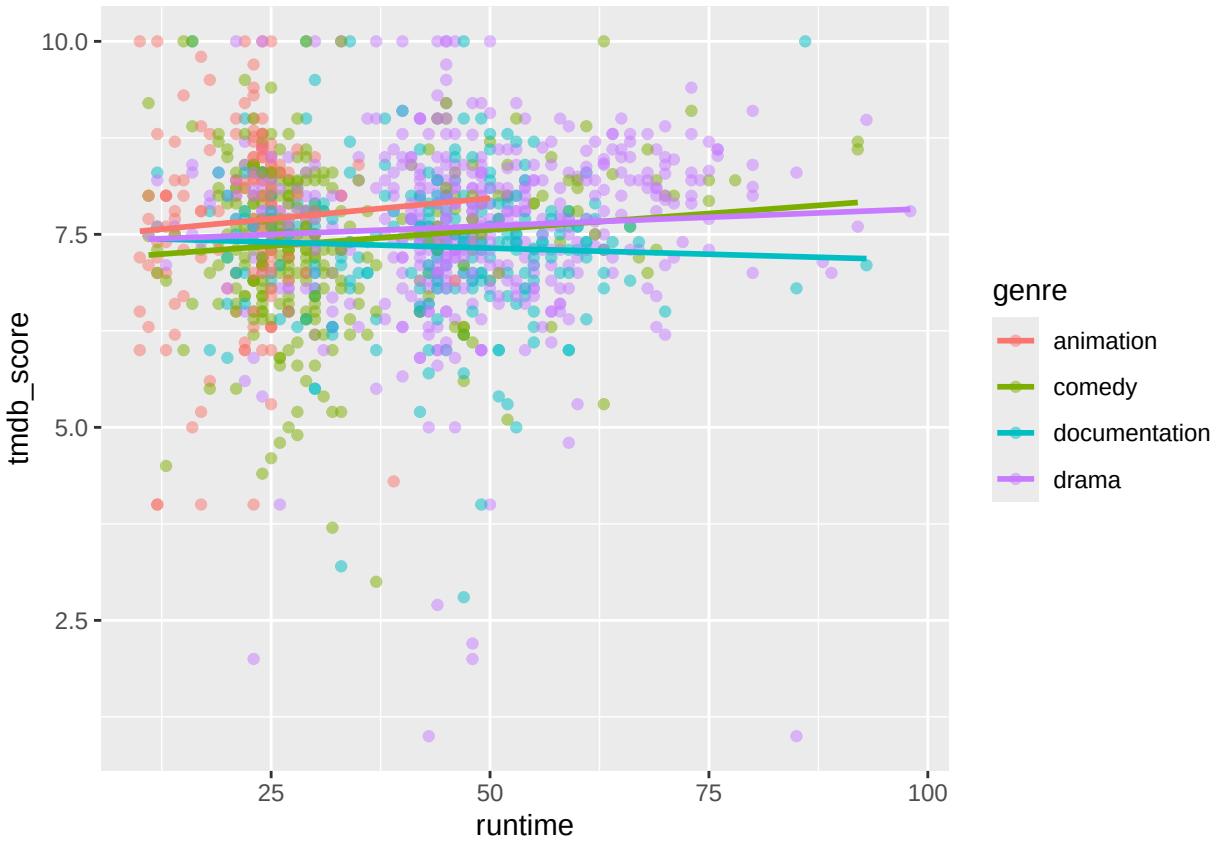
```
time_tmdb_scatter2 <- top_show_genres %>%
        ggplot(aes(x = runtime, y = tmdb_score, color = genre)) +
        geom_point(alpha = 0.5)
time_tmdb_scatter2
```



Similarly, you may want to see separate regression lines for each genre depicted on the plot. Note that you can remove the error bands by specifying the `se = FALSE` argument.

```
regression2 <- time_tmdb_scatter2 +
        geom_smooth(method = "lm", se = FALSE)
regression2
```

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



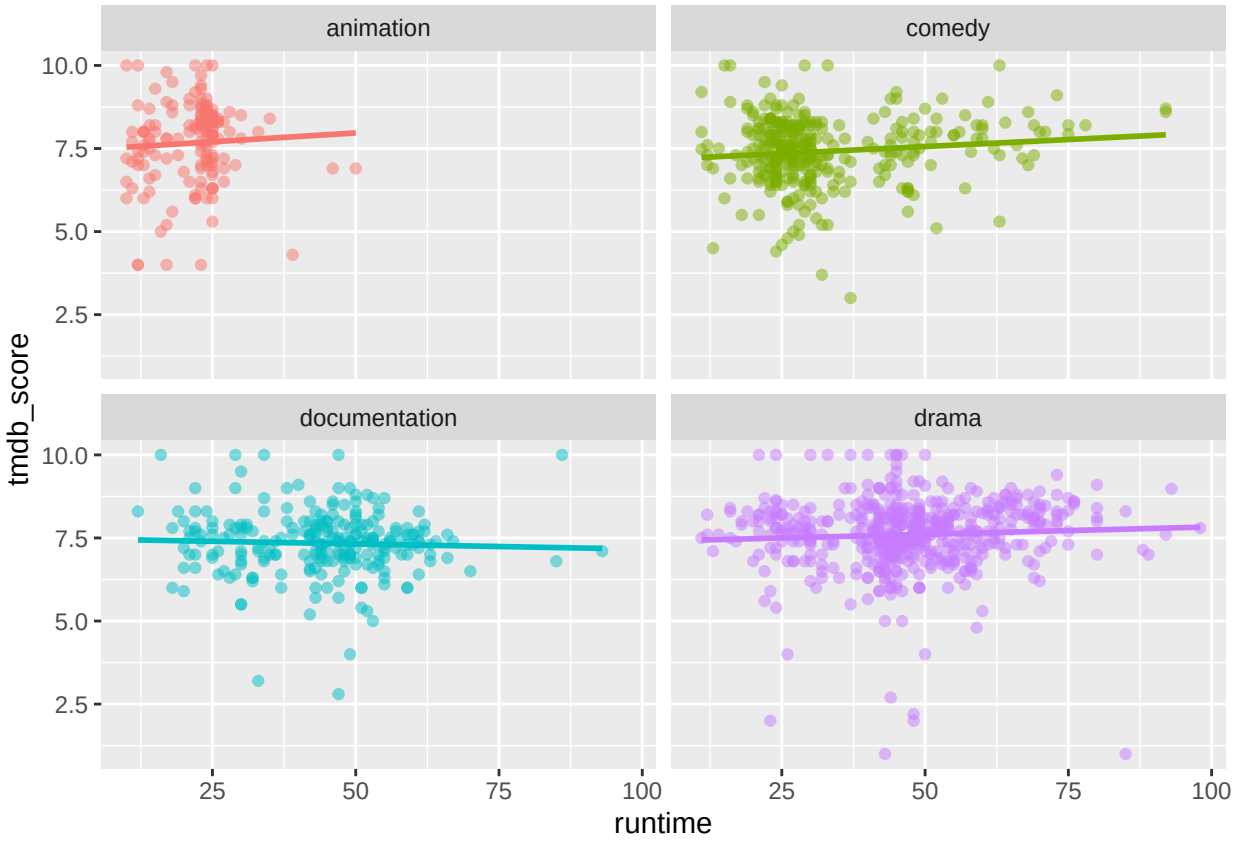
Unfortunately, this results in a rather messy plot. Although using color and separate lines may be helpful for a variable with just two levels, with four levels, it's too chaotic. Fortunately, an alternative exists: small multiples!

Create Small Multiples

A great function called `facet_wrap()` will replicate the plot for each level of a specified variable. Here, you will `facet_wrap` the `regression2` plot on the variable `genre` and remove the redundant legend by passing the argument `legend.position = "none"` argument to the `theme()` function.

```
regression2 +
        facet_wrap(vars(genre)) +
        theme(legend.position = "none")
```

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



The results are much cleaner. Practice adjusting the layout and labeling, using the arguments you learned in the first lesson.

Create a Bar Plot

A bar plot typically represents frequencies, means, or other statistics for various levels of a categorical variable. You want to create a bar plot comparing the mean IMDB scores for movies and TV shows in four genres: comedy, documentary, drama, and action.

Prepare the Data

First, you will use `dplyr` functions to summarize the IMDB scores for each genre and type.

```
imdb_means <- titles %>%
        group_by(genre, type) %>%
        summarize(mean = mean(imdb_score, na.rm = TRUE), n = n())
```

```
## `summarise()` has grouped output by 'genre'. You can override using the
## `.groups` argument.
```

```
imdb_means
```

```
## # A tibble: 38 x 4
## # Groups:   genre [19]
```

```
##    genre         type   mean     n
##    <chr>         <chr> <dbl> <int>
##  1 action        MOVIE  5.94   232
##  2 action        SHOW   6.87   133
##  3 animation     MOVIE  6.32   130
##  4 animation     SHOW   6.68   187
##  5 comedy        MOVIE  6.09   961
##  6 comedy        SHOW   6.98   344
##  7 crime         MOVIE  6.39   122
##  8 crime         SHOW   7.09   116
##  9 documentation MOVIE  6.99   414
## 10 documentation SHOW   7.18   251
## # i 28 more rows
```

Now, pick just the genres we're interested in: comedy, documentary, drama, and action.

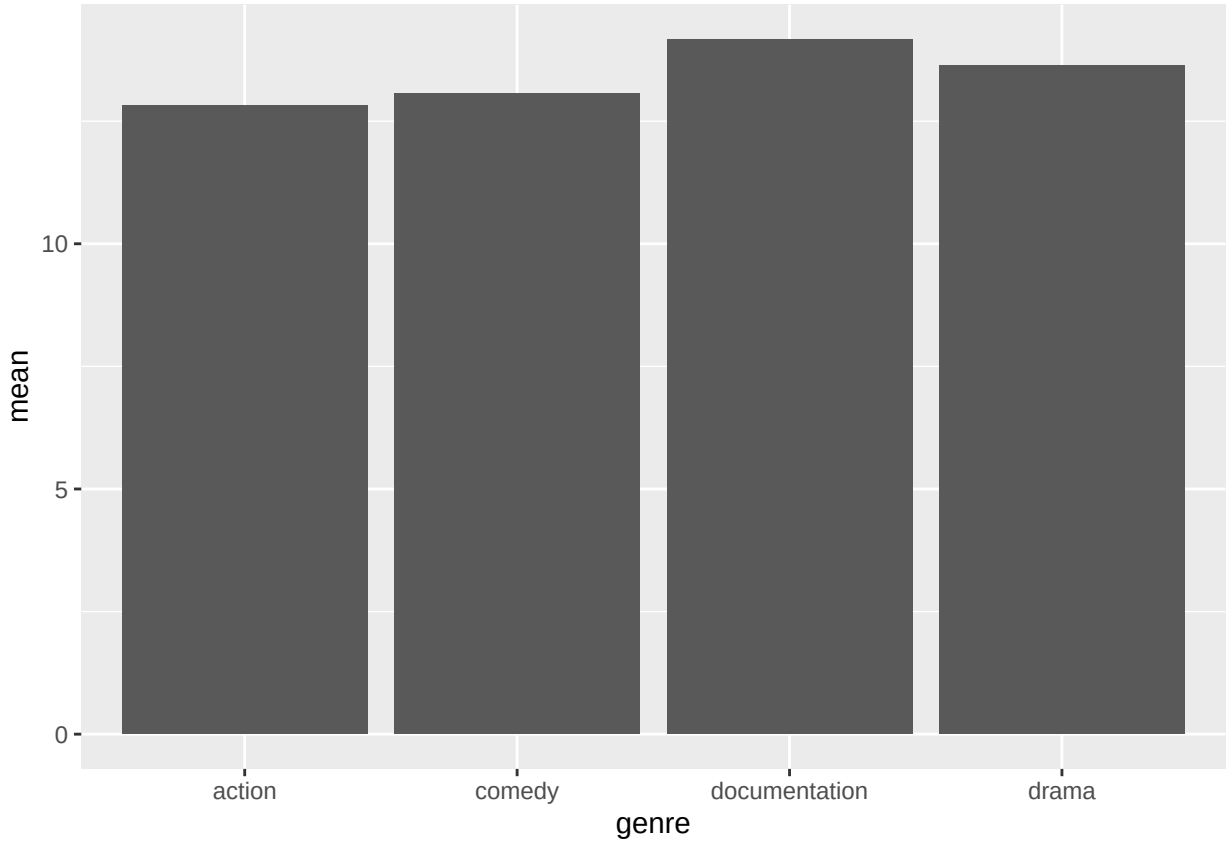
```
imdb_four <- imdb_means %>%
        filter(genre %in% c("comedy", "documentation", "drama", "action"))
imdb_four
```

```
## # A tibble: 8 x 4
## # Groups:   genre [4]
##   genre         type   mean     n
##   <chr>         <chr> <dbl> <int>
## 1 action        MOVIE  5.94   232
## 2 action        SHOW   6.87   133
## 3 comedy        MOVIE  6.09   961
## 4 comedy        SHOW   6.98   344
## 5 documentation MOVIE  6.99   414
## 6 documentation SHOW   7.18   251
## 7 drama         MOVIE  6.46   882
## 8 drama         SHOW   7.19   539
```

Create a Bar Plot using `geom_col()`

Next, create a bar plot of the means. Note that the aesthetic mapping includes two variables: x for genre and y for mean. (You can also drop the `data =` and mapping =`prefixes as long as the data argument is the first and the`aes()`is the second argument in the`ggplot‘ function.)

```
ggplot(imdb_four, aes(x = genre, y = mean)) +
        geom_col()
```

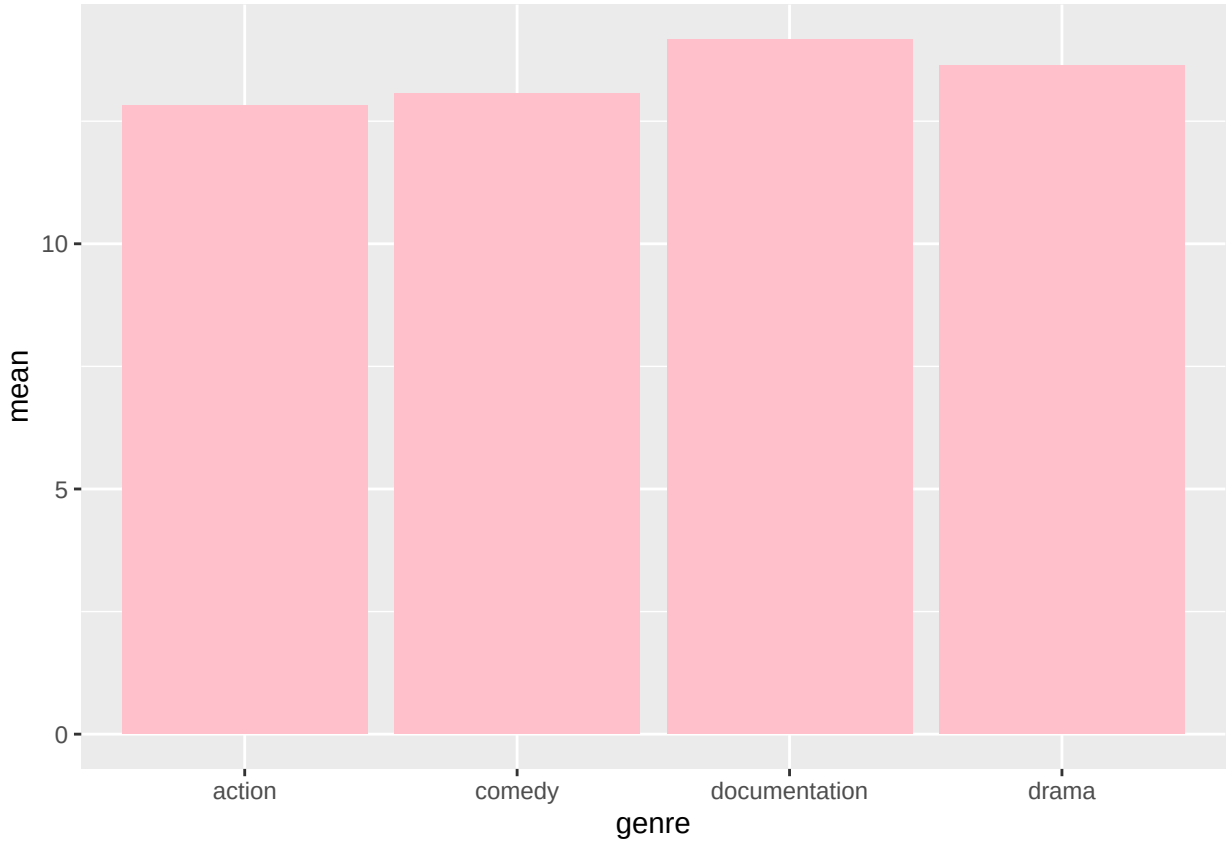


The result is functional but rather boring, so let's add some visual interest with color.

Add Color to the Bar Plot

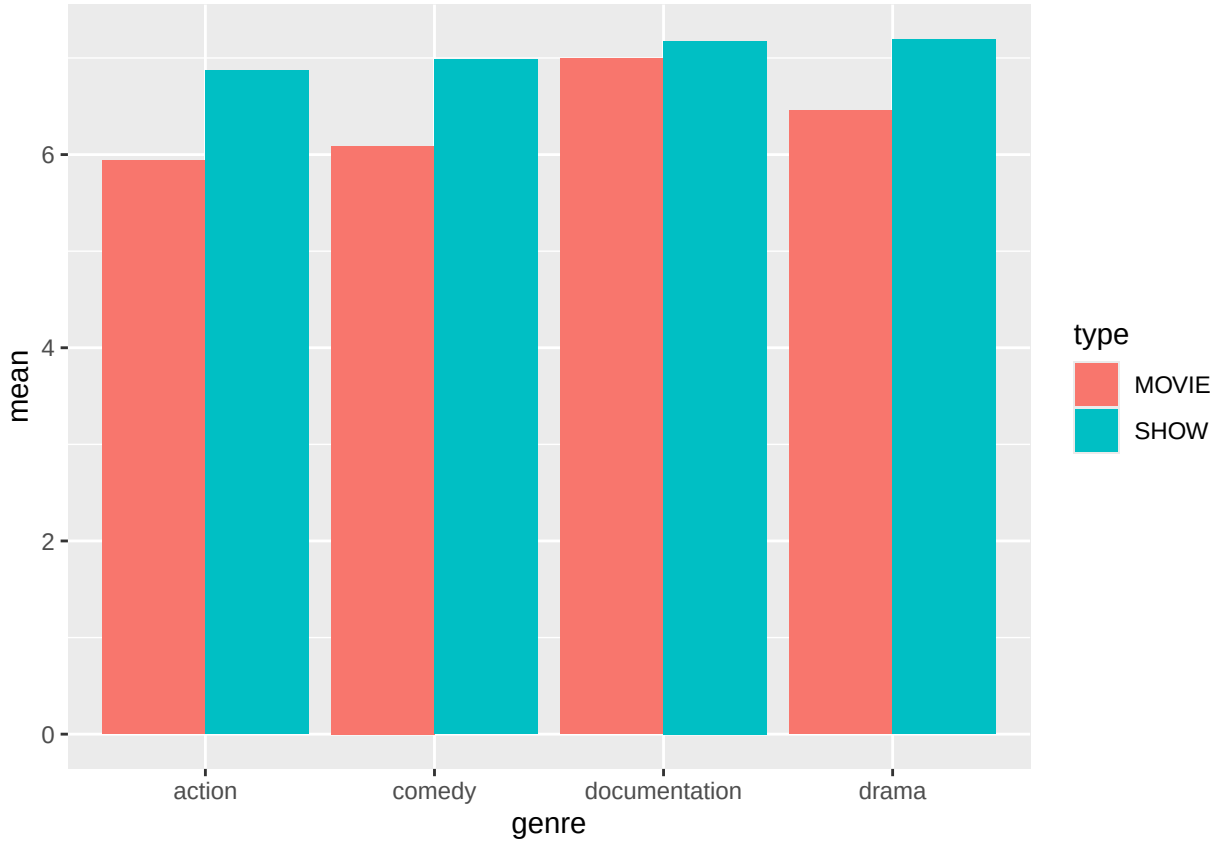
The basic method for adding color to the bars is to specify the `fill` argument within the `geom_col()` function:

```
ggplot(imdb_four, aes(x = genre, y = mean)) +
        geom_col(fill = "pink")
```



However, it might be more interesting to compare TV shows and movies, with each type shown as a different color. In that case, you can map a variable to the aesthetic `fill`. Note that you will also specify an attribute within the `geom_col()` layer, indicating that the bars should be side-by-side. (The default is stacked bars.)

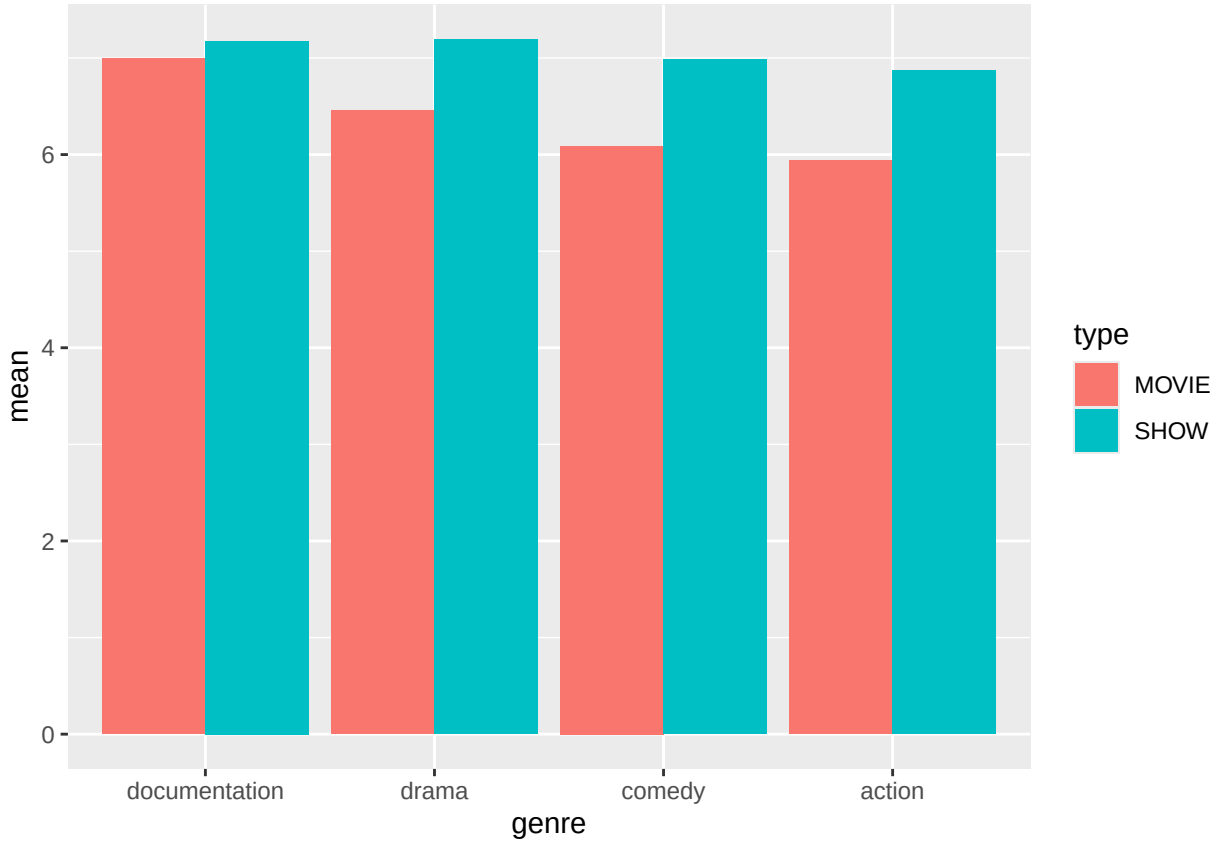
```
four_genres_bar <- ggplot(data = imdb_four, aes(x = genre, y = mean, fill = type)) +
        geom_col(position = "dodge")
four_genres_bar
```



Re-order the X Axis

By default, the categorical variable levels are ordered alphabetically. You might want to re-order the bars in some other way. To do this, use the `x_scale_discrete()` function and save the plot to `ordered_genres_bar`. (Note that `x_scale_discrete()` is for categorical variables. You will see transformation of an x axis for continuous variables momentarily.)

```
ordered_genres_bar <- four_genres_bar +
        scale_x_discrete(limits = c("documentation", "drama", "comedy", "action"))
ordered_genres_bar
```

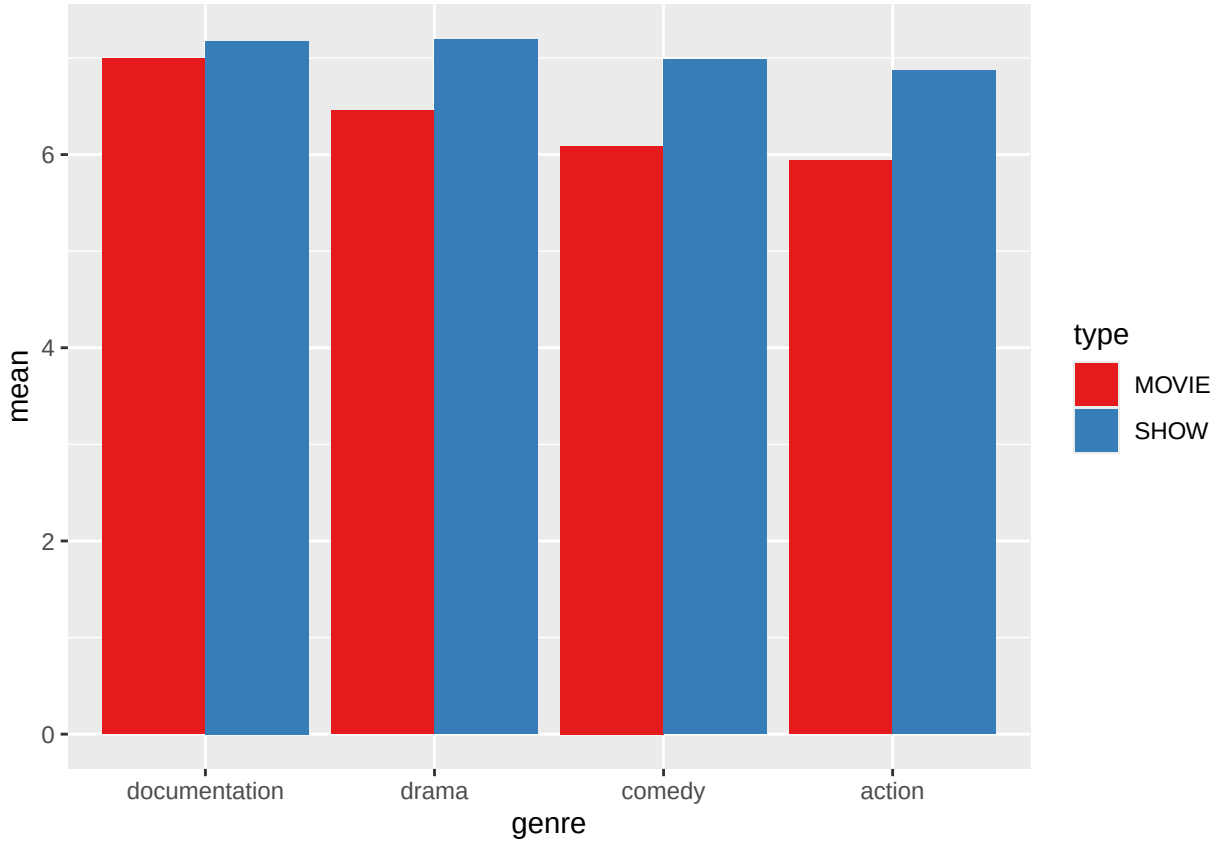


The result shows the genre arranged with documentation on the far left.

Use a Different Color Palette

If the color palette is not to your liking, you can use one of several included options, and there are more in the `RColorBrewer` package.

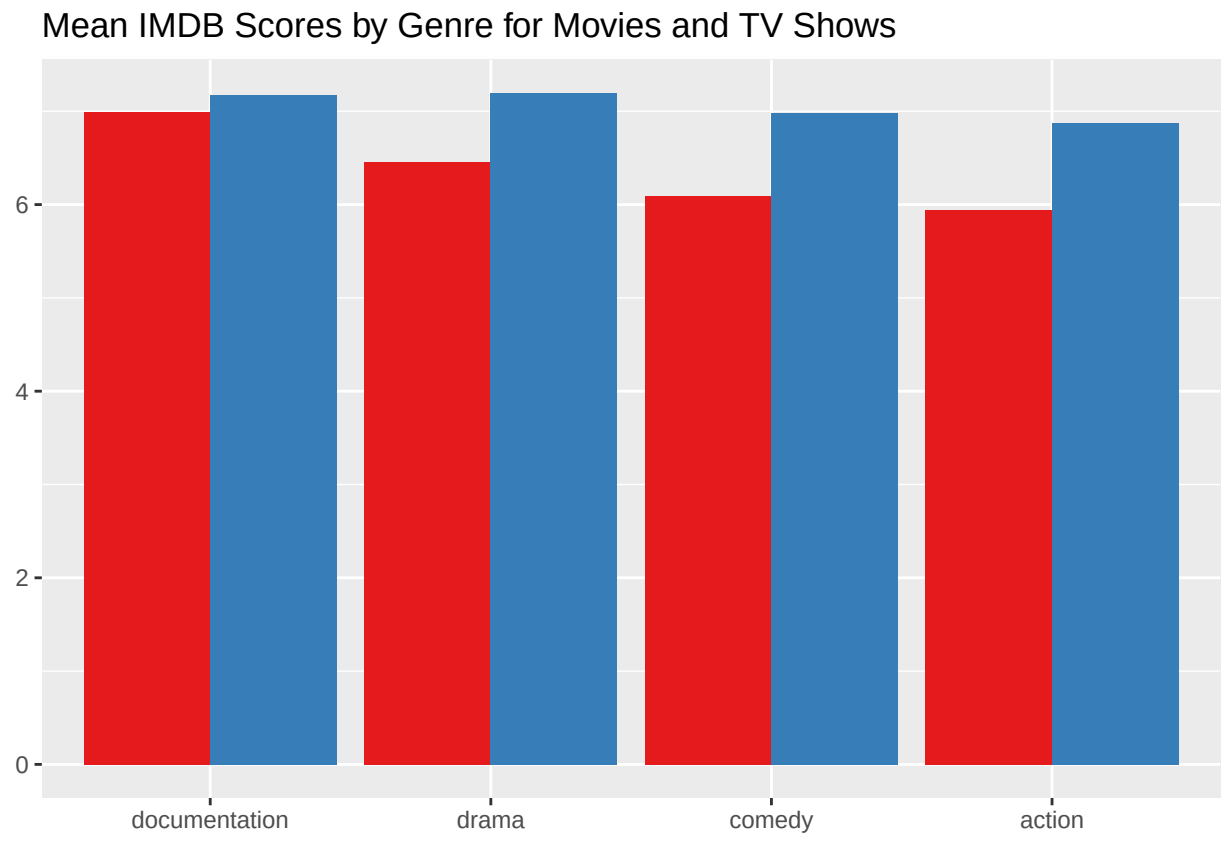
```
ordered_genres_bar_bold <- ordered_genres_bar +
        scale_fill_brewer(palette = "Set1")
ordered_genres_bar_bold
```



Remove the Legend and Modify Titles/Axis Labels

Finally, because the legend is somewhat redundant (the x axis bar labels provide the same information), you can remove the legend with `theme()`:

```
ordered_genres_bar_bold +
        theme(legend.position = "none") +
        labs(title = "Mean IMDB Scores by Genre for Movies and TV Shows",
             x = element_blank(),
             y = element_blank())
```



Create a Line Plot

The final type of plot you will explore in this session is a line plot, which is typically used for charting change over time. For example, you might be interested in how IMDB scores have changed from 2012 to 2022, comparing TV shows to movies.

Prepare the Data

First filter the data for movies and shows released in 2012 and beyond. Then group the data by release year and type before calculating mean IMDB scores for each.

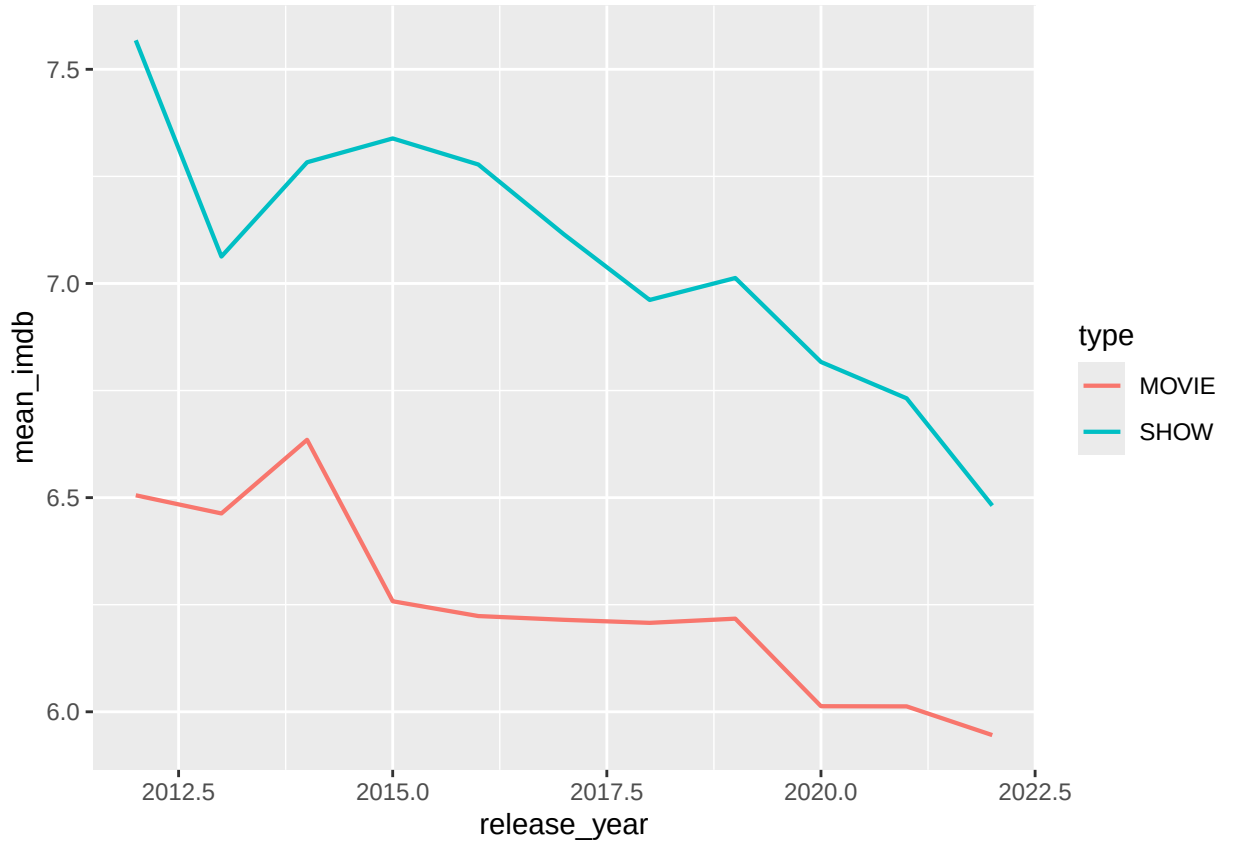
```
imdb_over_time <- titles %>%
        filter(release_year >= 2012) %>%
        group_by(release_year, type) %>%
        summarize(mean_imdb = mean(imdb_score, na.rm = TRUE))
```

```
## `summarise()` has grouped output by 'release_year'. You can override using the
## `.groups` argument.
```

Create the plot

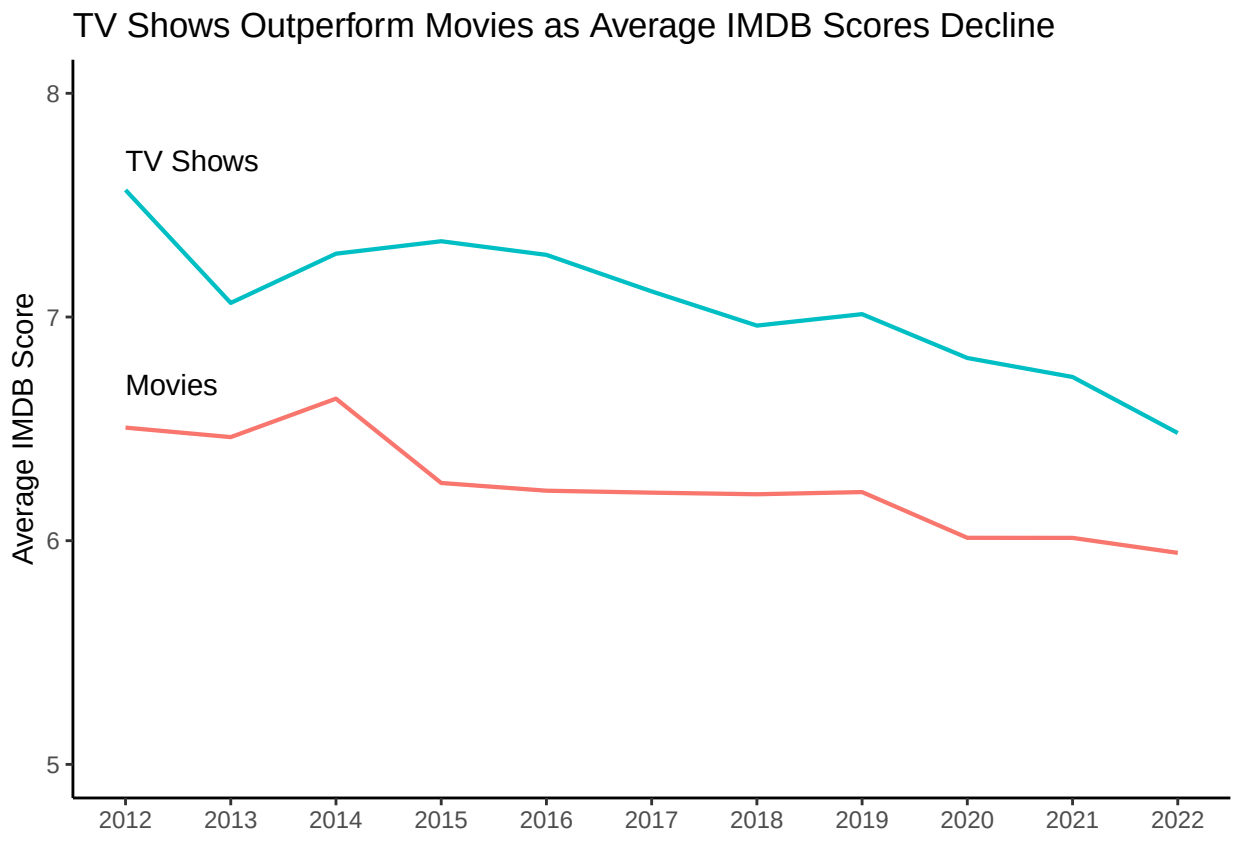
Now, we can plot these scores over time.

```
imdb_trends <- imdb_over_time %>%
        ggplot(aes(x = release_year, y = mean_imdb, group = type)) +
                geom_line(aes(color = type), size = .75)
imdb_trends
```



The mean IMDB scores for shows and movies have declined with shows consistently beating the movies. Let's add a title stating that and use some of our other new `gglot` skills to clean up `imdb_trends`.

```
imdb_trends <- imdb_trends +
        scale_x_continuous(breaks = seq(2012, 2022, 1)) +
        ylim(5, 8) +
        labs(title = "TV Shows Outperform Movies as Average IMDB Scores Decline",
             x = element_blank(),
             y = "Average IMDB Score") +
        annotate("text", x = c(2012, 2012),
                 y = c(7.7, 6.7),
                 label = c("TV Shows", "Movies"),
                 hjust = 0) +
        theme_classic() +
        theme(legend.position = "none")
imdb_trends
```



Using `ggplot` takes practice, and there are endless variations on these and other plots. Challenge yourself to make small adjustments to the defaults to improve the beauty and effectiveness of your graphics!

References

- Chang, W. (2022). *R graphics cookbook*. O'Reilly.
- R Color Brewer palettes. (n.d.).
- R Graph Gallery. (n.d.).
- R Studio ggplot Cheatsheet. (n.d.).
- Soero, V. (2022). Netflix TV Shows and Movies.
- Wang, H. (n.d.). `ggplot2` Theme Elements Demonstration.
- Wei, Y. (2021). R Color Cheat Sheet.
- Wickham, H., & Grolemund, G. (2017). *R for data science*. O'Reilly.
- Wilkinson, L. (2005). *The grammar of graphics*. Springer.