Visualizing Data with ggplot2 in R

Catherine Barber

2023-03-10

Goal and Learning Outcomes

Goal: The goal of this lesson is for you to create effective data visualizations using ggplot2 in R.

Learning Outcomes: During this lesson, you will demonstrate your ability to...

- Explain the layered approach of the grammar of graphics.
- Identify common types of plots and their associated geoms.
- Create basic and enhanced histograms.
- Create basic and enhanced scatter plots.
- Create basic and enhanced box plots.
- Create basic and enhanced bar plots.

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 this lesson, you will focus on using `ggplot2` to create four types of graphs: a histogram, a scatter plot, a bar plot, and a box plot. However, you can create numerous other types of graphs in ggplot2, such as density plots, box plots, violin plots, maps, and much 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)
library(tinytex)
```

```
titles_short <- read.csv("titles_short.csv")
```

Exercise 1: Create a Histogram

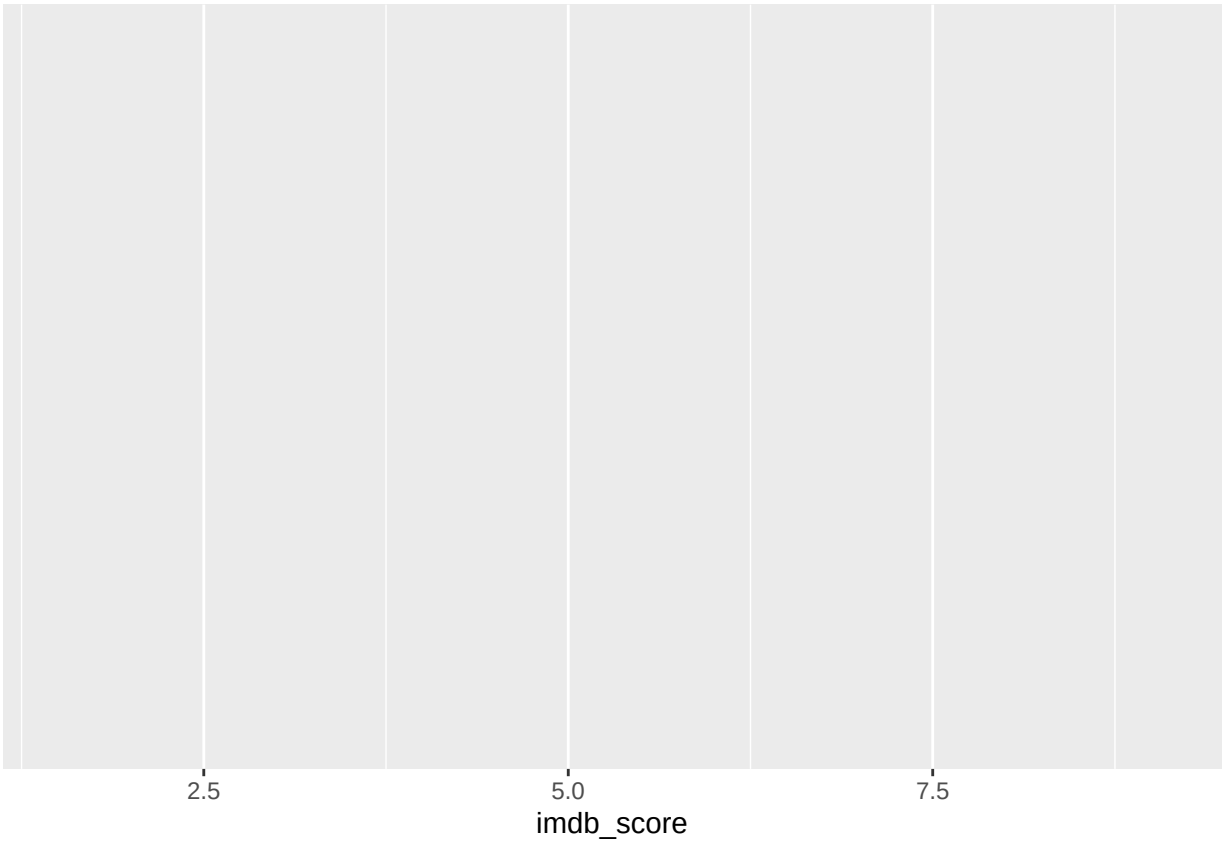
Begin by filtering the data to include only movies, save this as an object called `movies`, and look at the structure of the object.

```
movies <- titles_short %>%
  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" NA ...
##  $ 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. Note that nothing will be represented because you have not specified what type of plot you are creating; however, this step can save you a bit of time later.

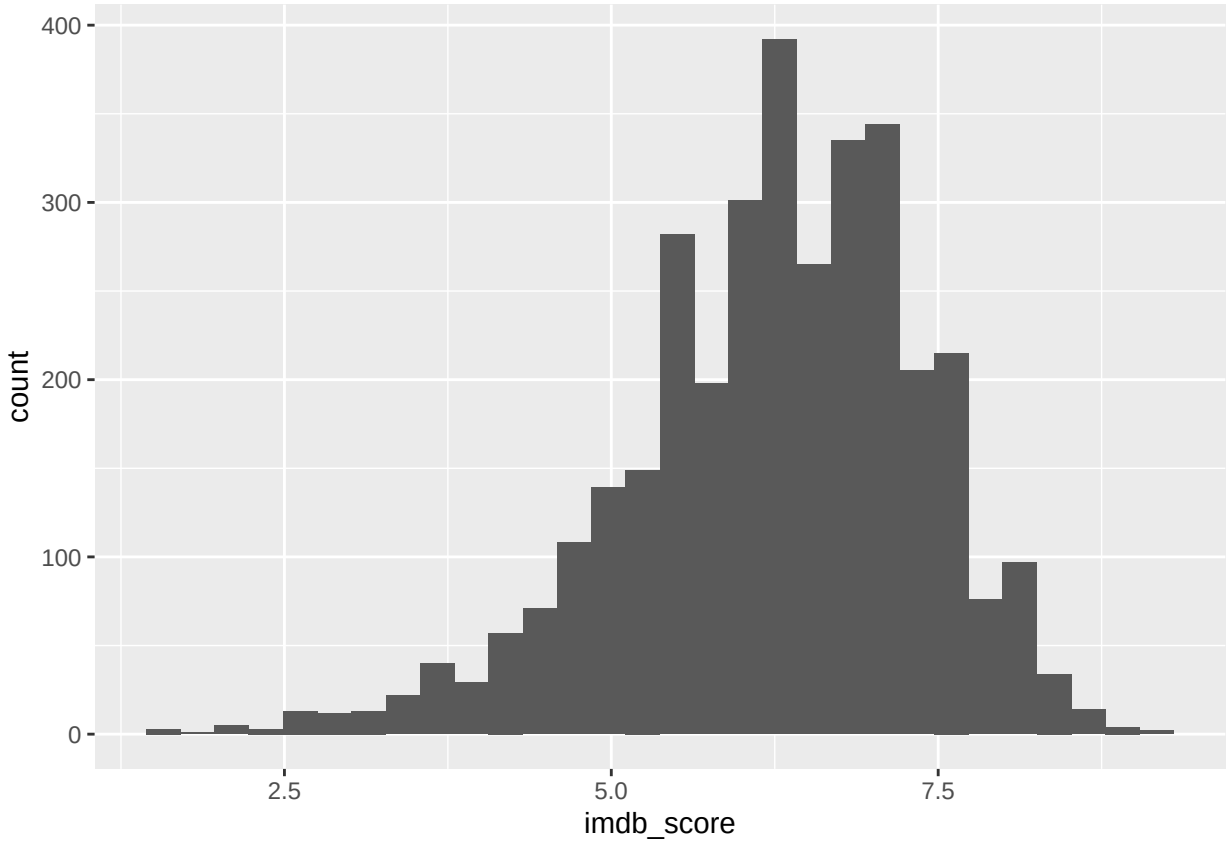
```
base_plot <- ggplot(data = movies, mapping = aes(x = imdb_score))
base_plot
```



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()
```

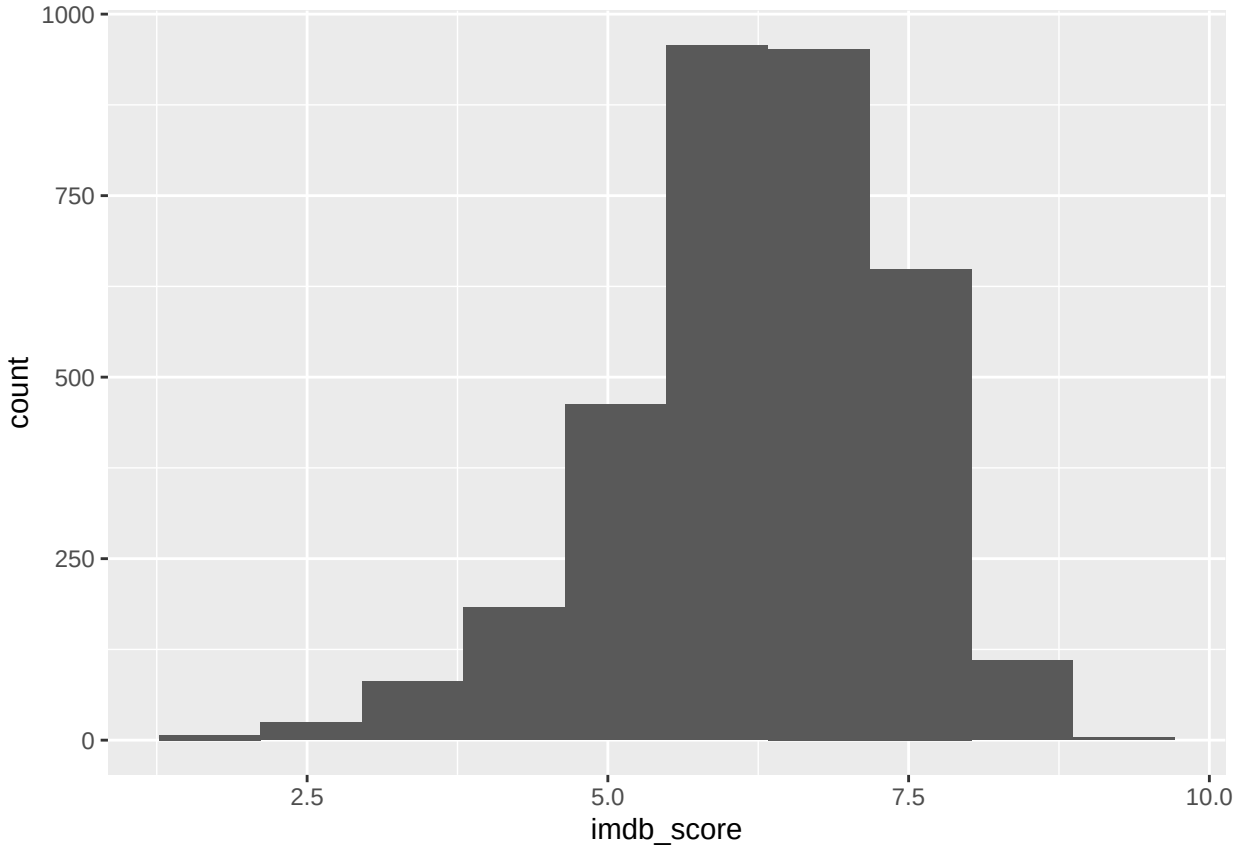


Adding the geom layer calls a histogram on the variable imdb_score, which was specified as x in the previous call.

Change the Appearance of the Distribution

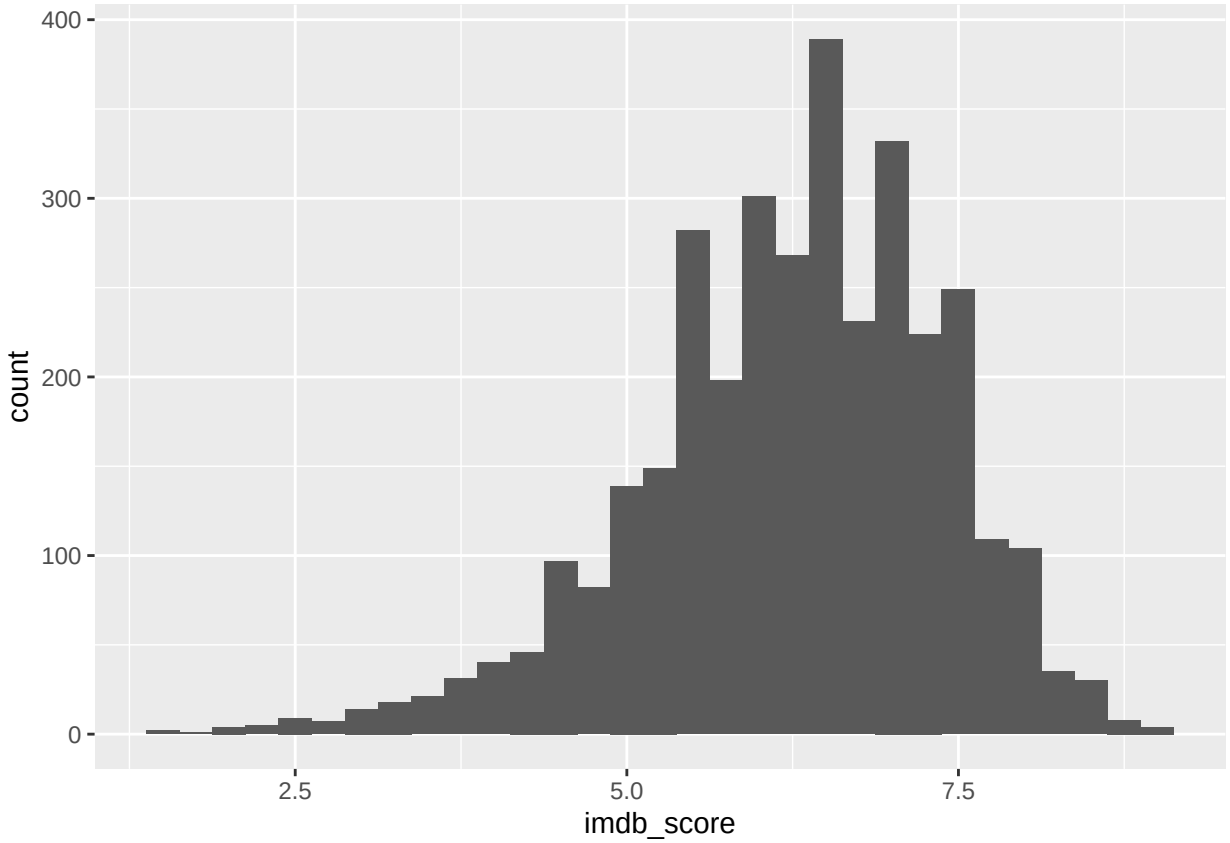
You can change the appearance of the distribution by adding an argument within `geom_histogram()`. There are two options. First, change the number of bins:

```
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)
```

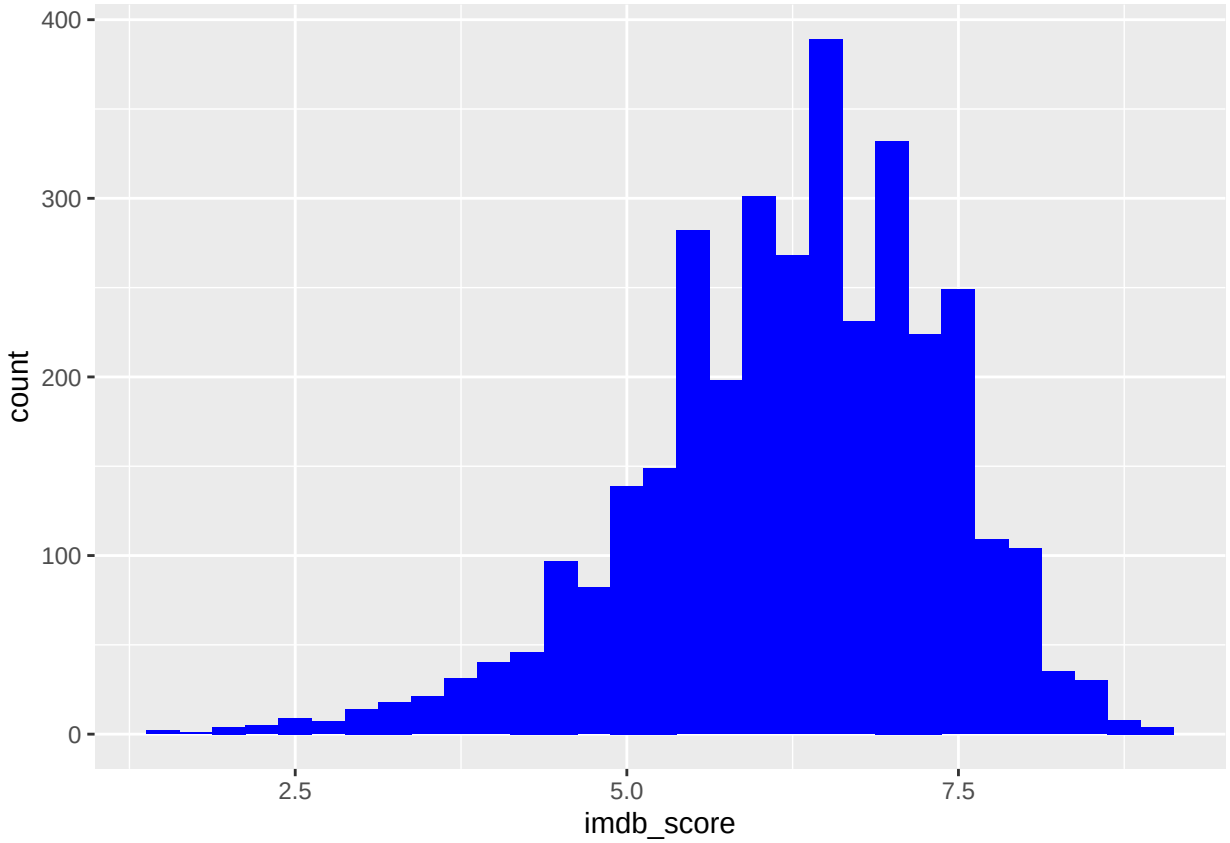


This will adjust the width of the bins. Practice varying the number of bins or bin width until you find one that you like best.

Change the Color of the Bars

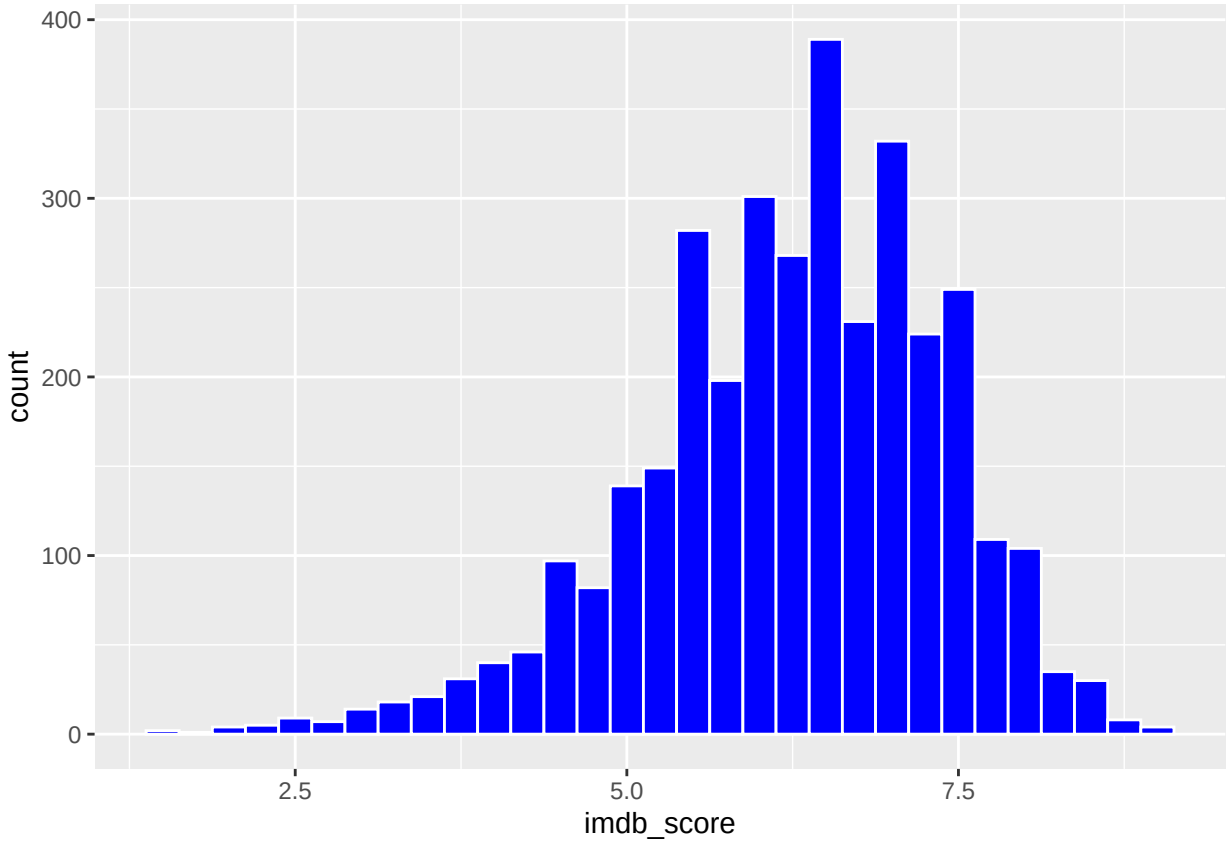
The default in `ggplot2` is grayscale, but you aren't limited to that! For color names, check out Wei's (2021) R Color Cheat Sheet.

```
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 emphasis.

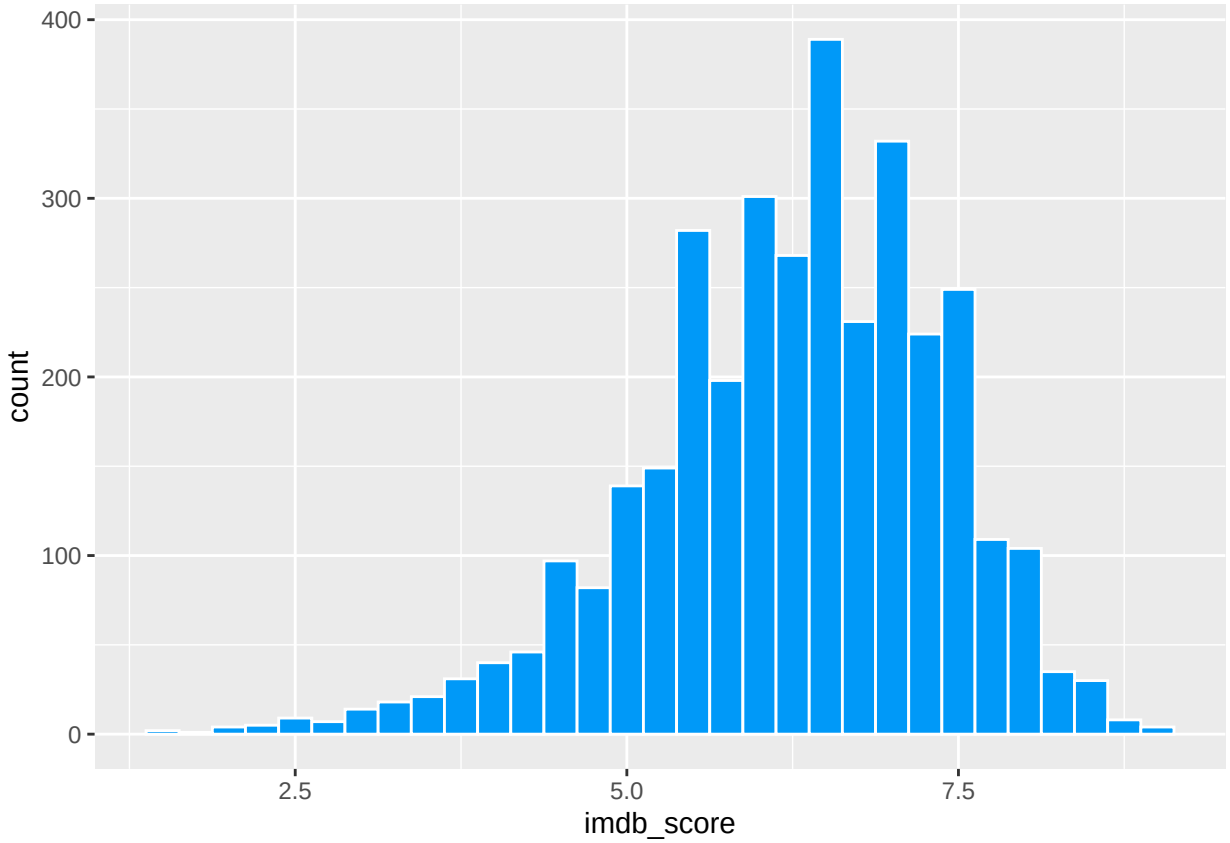
```
base_plot +
  geom_histogram(binwidth = .25, fill = "blue", color = "white")
```



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 htmlcolorcodes.com.

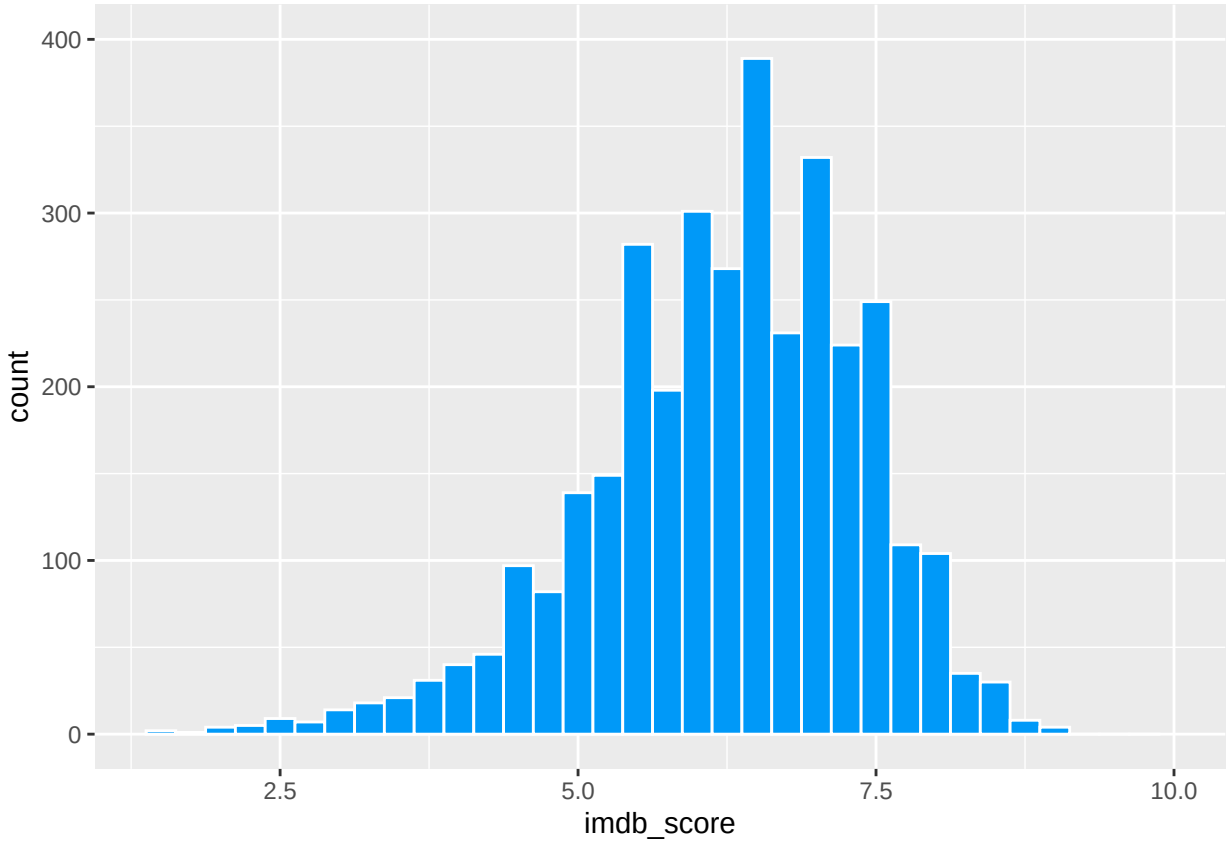
```
hist_base <- base_plot +
  geom_histogram(binwidth = .25, fill = "#0099F8", color = "white")
hist_base
```



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`.

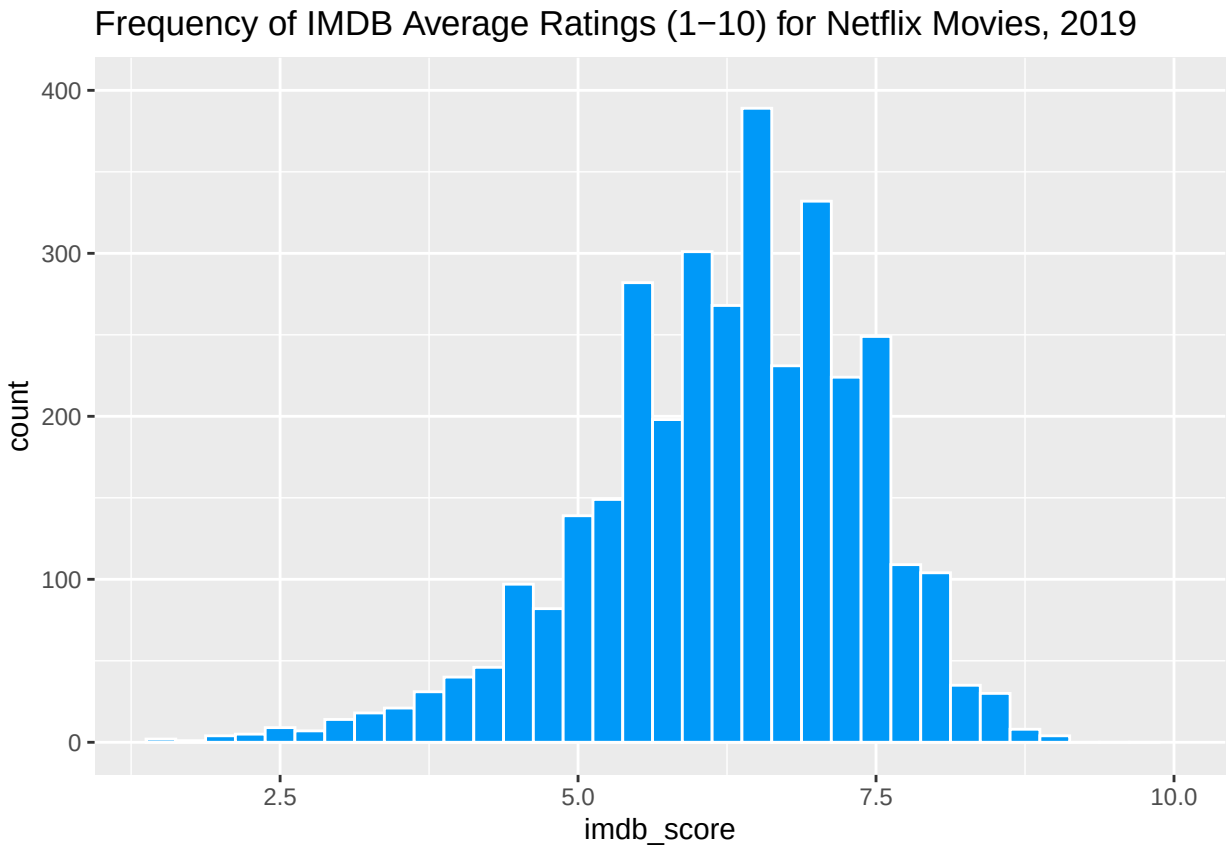
```
hist_limits <- hist_base +
  xlim(NA, 10) +
  ylim(0, 400)
hist_limits
```



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`.

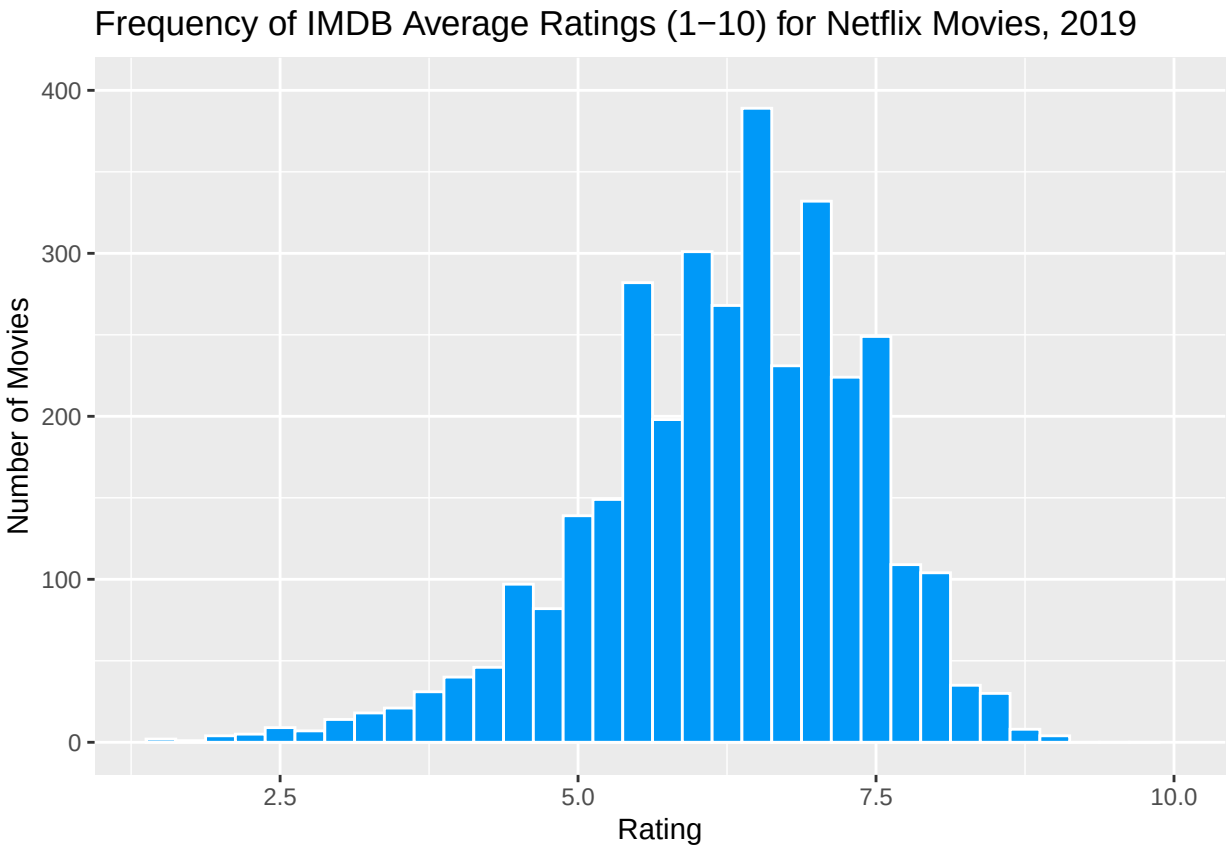
```
hist_title <- hist_limits +
  ggtitle("Frequency of IMDB Average Ratings (1-10) for Netflix Movies, 2019")
hist_title
```



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

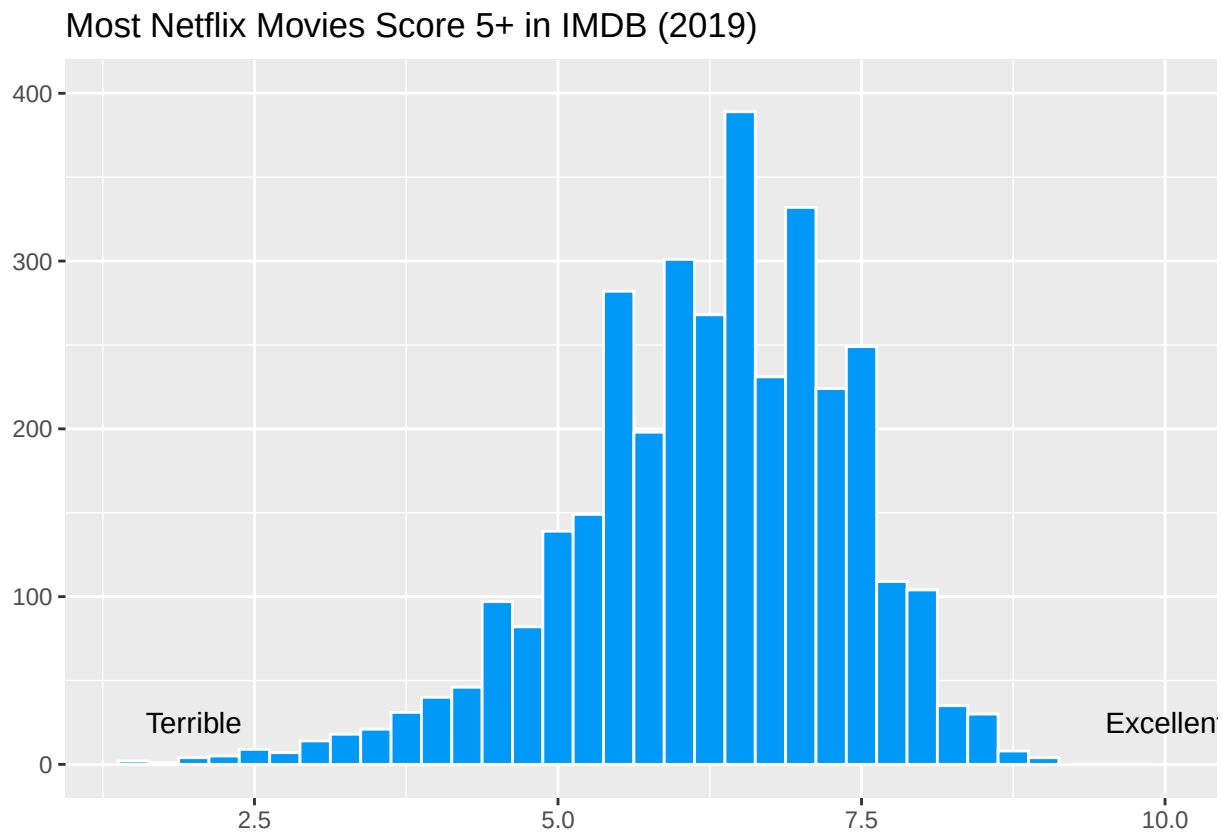
If you have already used `ggtitle()` in a base plot but want to modify the title, there is no need to create a new base plot. Simply override the title with a new title using the `labs` function. With a bit of tweaking to the title, you can remove the now-redundant x and y axis labels. Accomplish this by specifying 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 (2019)",
       x = element_blank(),
       y = element_blank())
```

Add Annotation

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`.

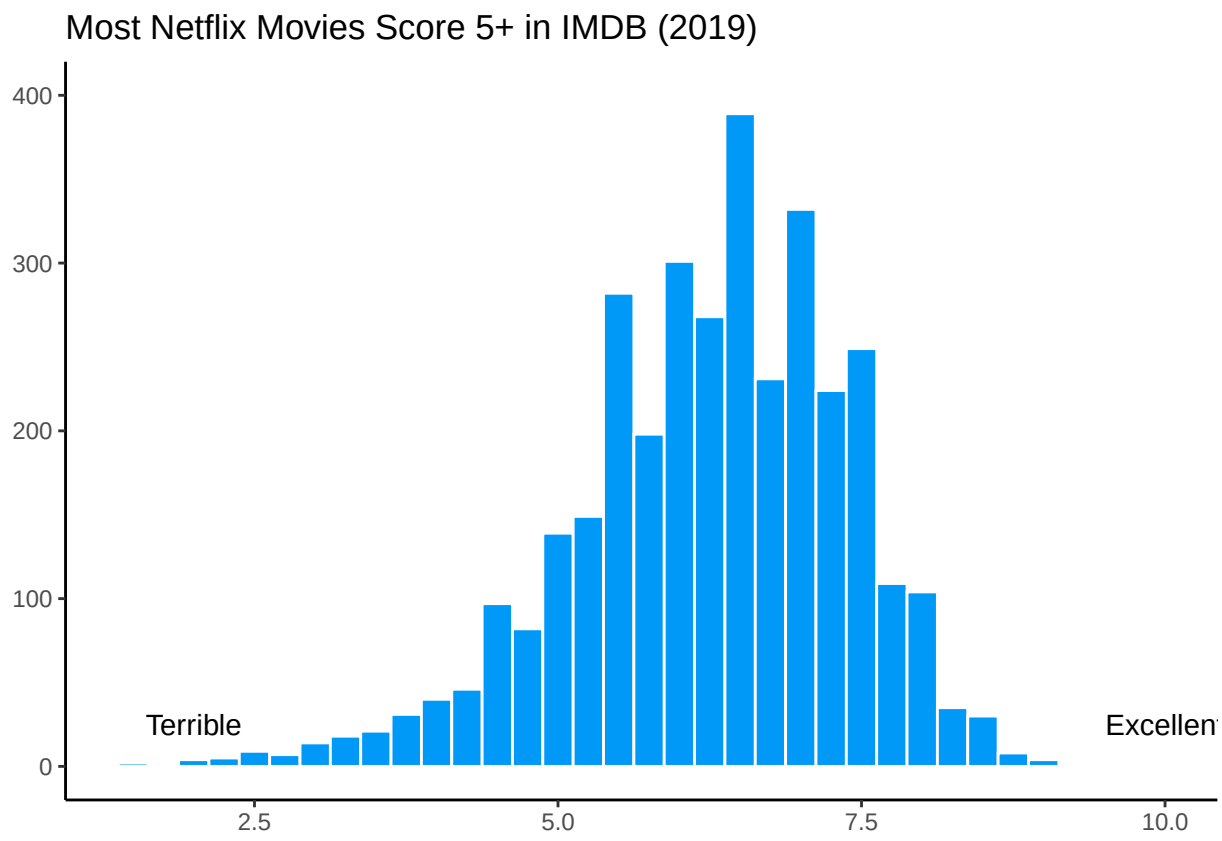
```
hist_annotated <- hist_title_revised +
  annotate("text", x = c(2,10), y = c(25,25), label = c("Terrible", "Excellent"))
hist_annotated
```



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.

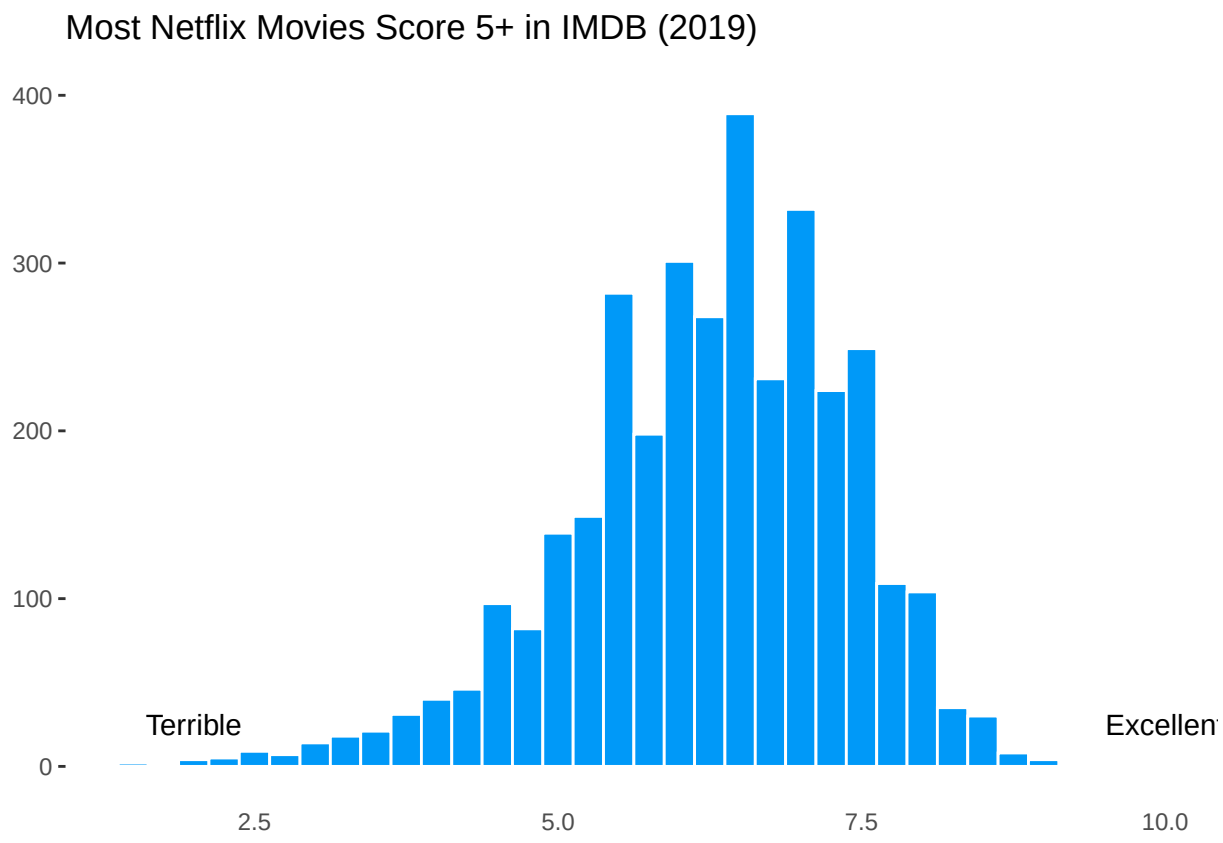
```
hist_annotated +
  theme_classic()
```



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!

The final aspect of this exercise 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.

```
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())
```



Exercise 2: 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_short` dataset for these four genres of TV shows and for `runtime` less than 100. Note that the `%in%` function finds all observations within `genre` that match the strings specified in the vector of genres `c("drama", "comedy", "documentation", "animation")`.

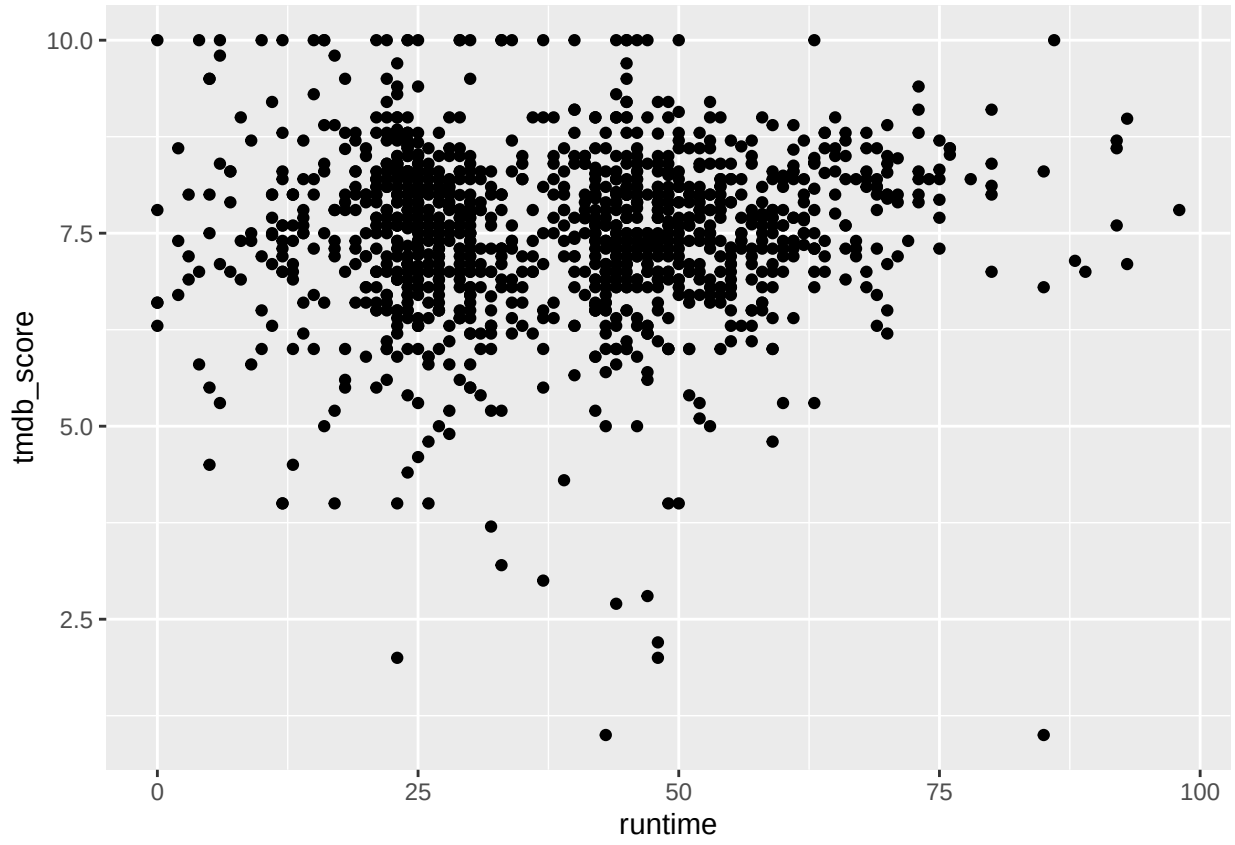
```
top_show_genres <- titles_short %>%
  filter(type == "SHOW" & genre %in% c("drama", "comedy", "documentation", "animation") & runtime < 100)
```

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 function (`%>%`) rather than using the argument `data = data` within the `ggplot()` function. This also helps when your plot immediately follows one or more `dplyr` functions.

Take a look:

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

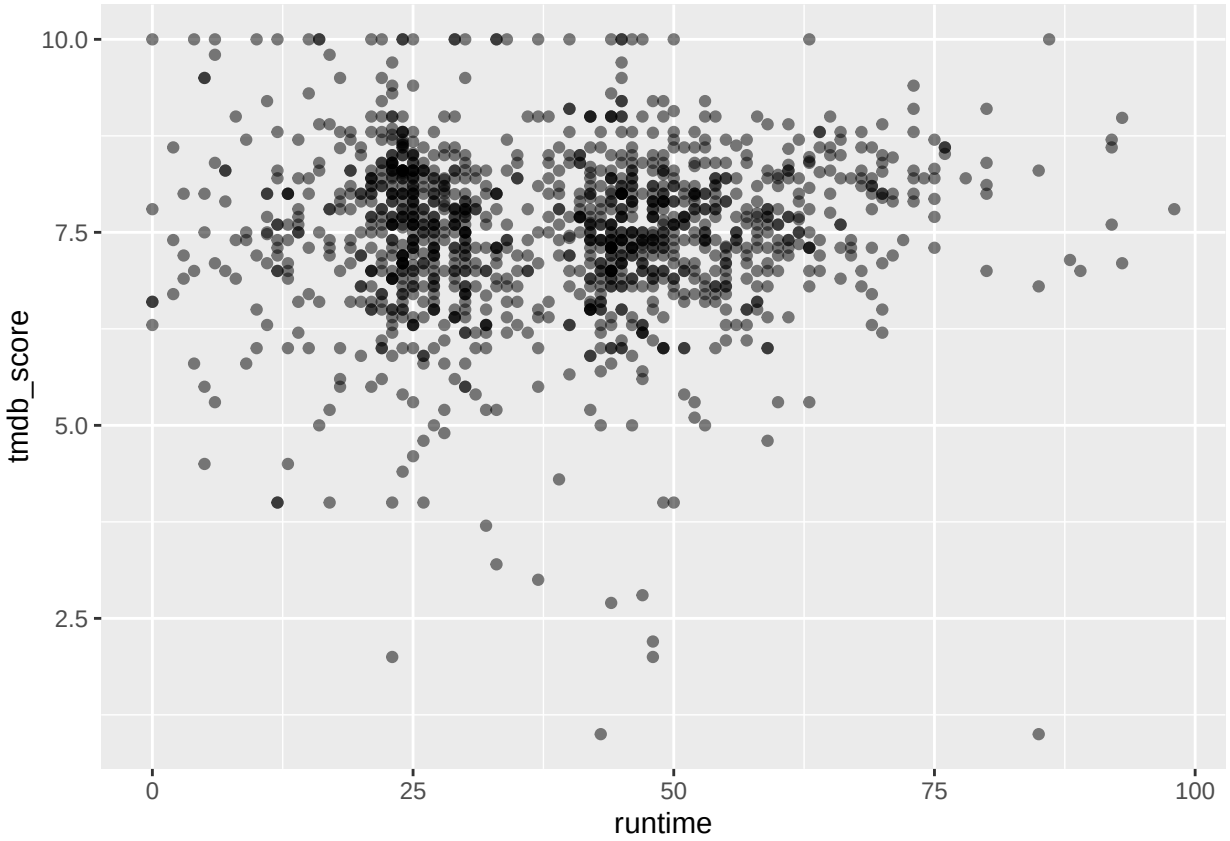


The plot shows little apparent relationship between the length of TV shows and their TMDB scores. However, there is an issue with overplotting. You can address this problem with the argument `alpha`.

Adjust Opacity

To make the points lighter or darker, 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 = 0.5)
time_tmdb_scatter1
```

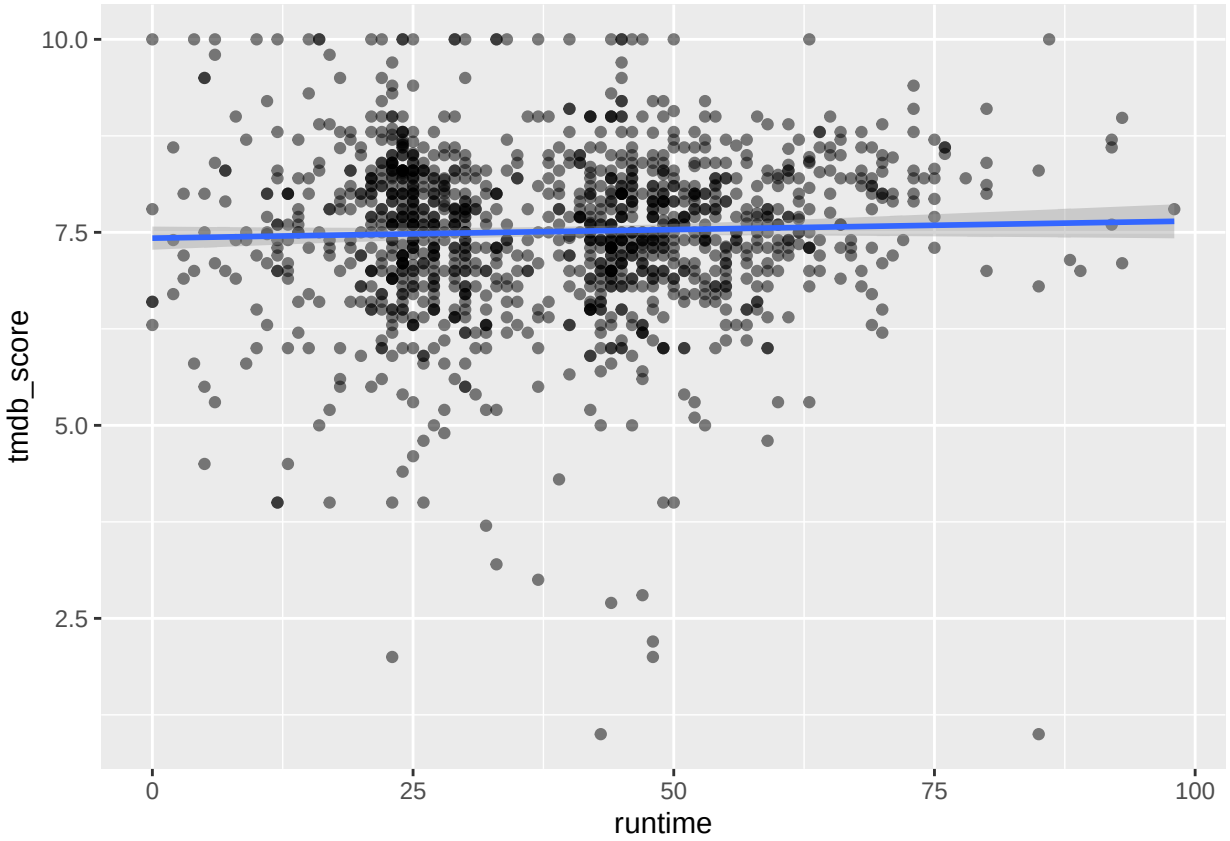


Notice that some of the points are darker, indicating more than one TV show represented by that point.

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.

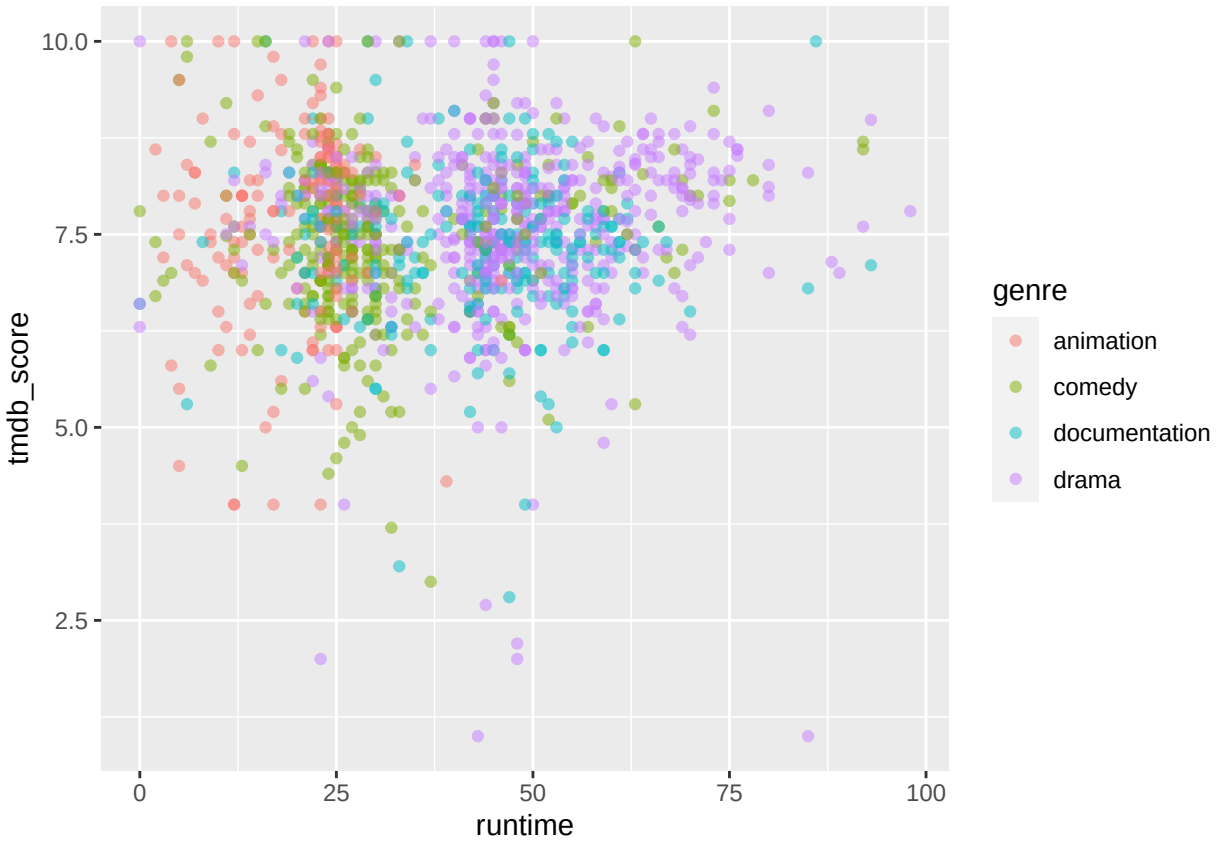
```
regression1 <- time_tmdb_scatter1 +
  geom_smooth(method = "lm")
regression1
```



Add Color

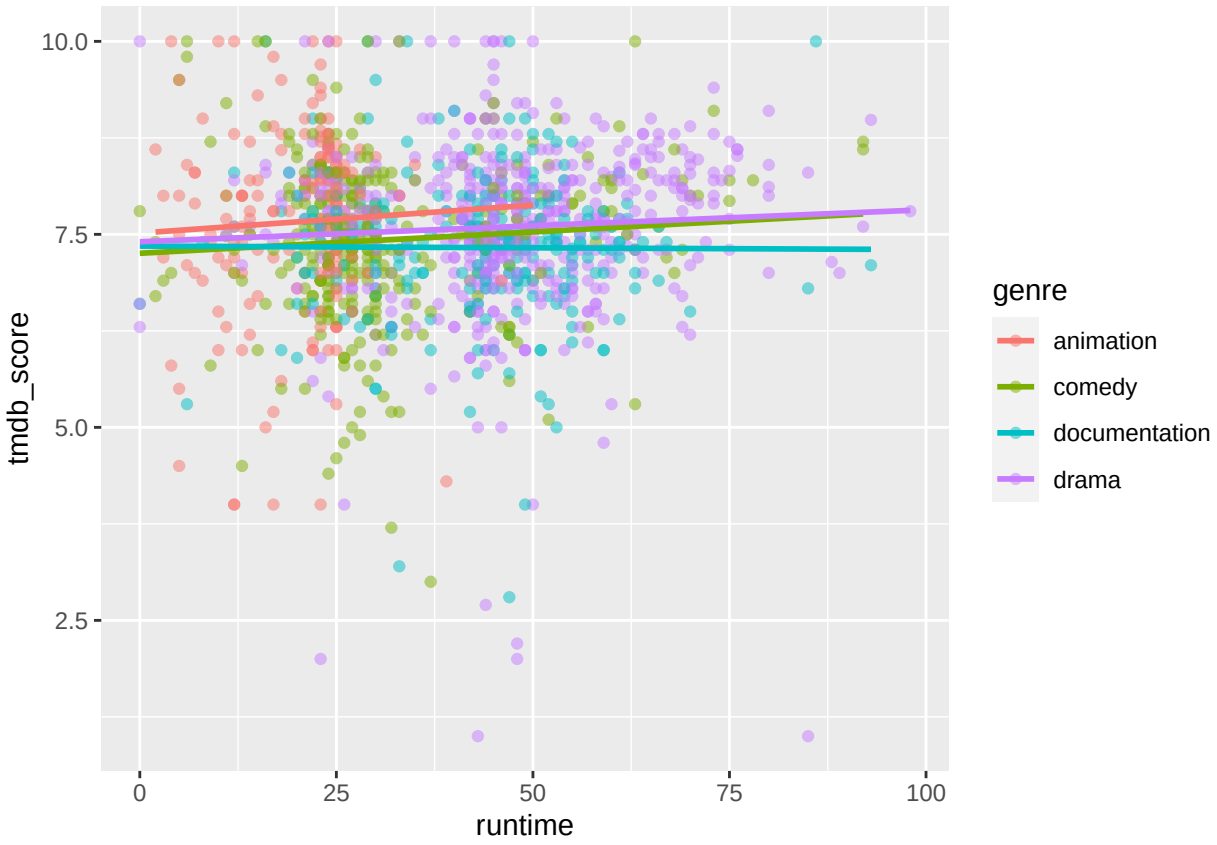
Although you can make all of the points the same color by passing a color name or hexcode to the argument `fill =` within `geom_point`, it can be useful to assign a variable to the color aesthetic. For example, assign `genre` to color:

```
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
```

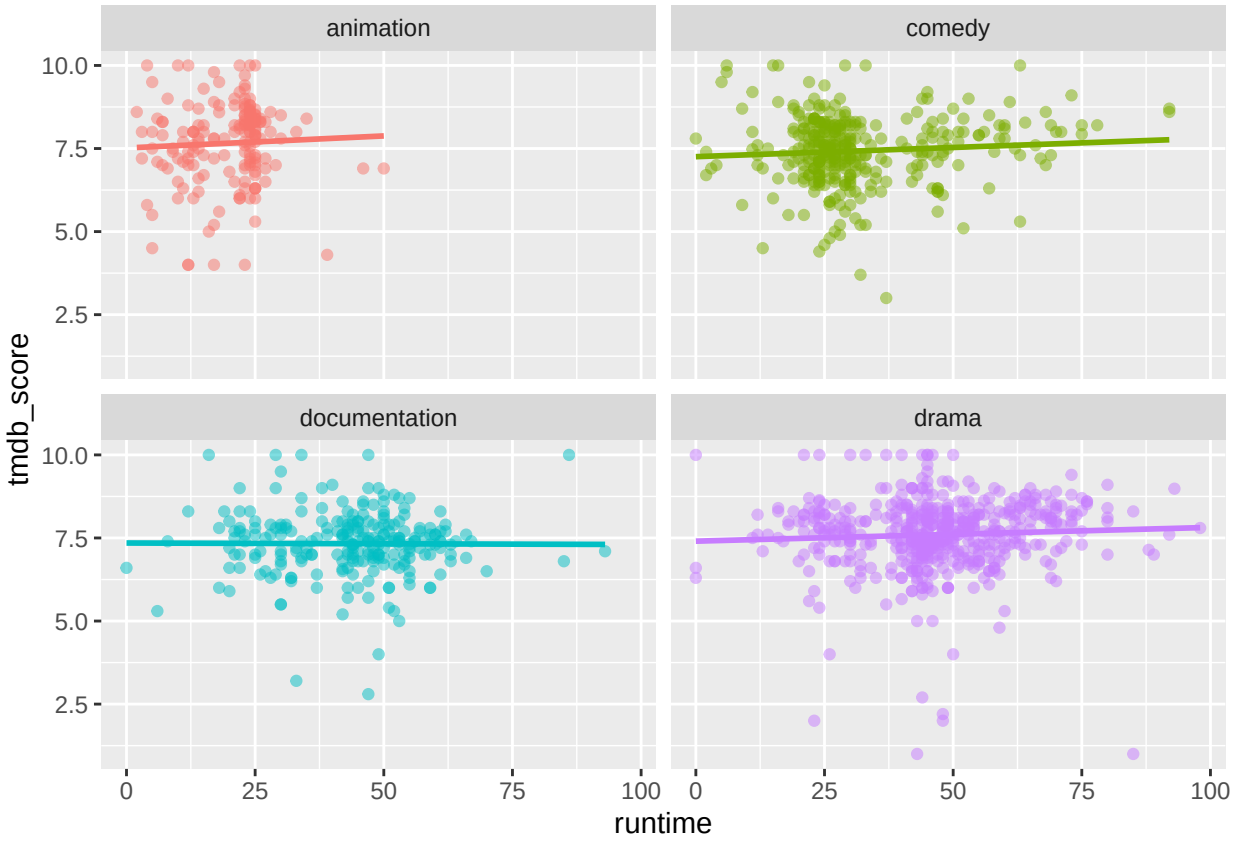


Unfortunately, this results in a rather messy plot. Although using color and separate lines may be helpful for a variable with just two levels, with four levels, it's too chaotic. Fortunately, an alternative exists: small multiples!

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")
```



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

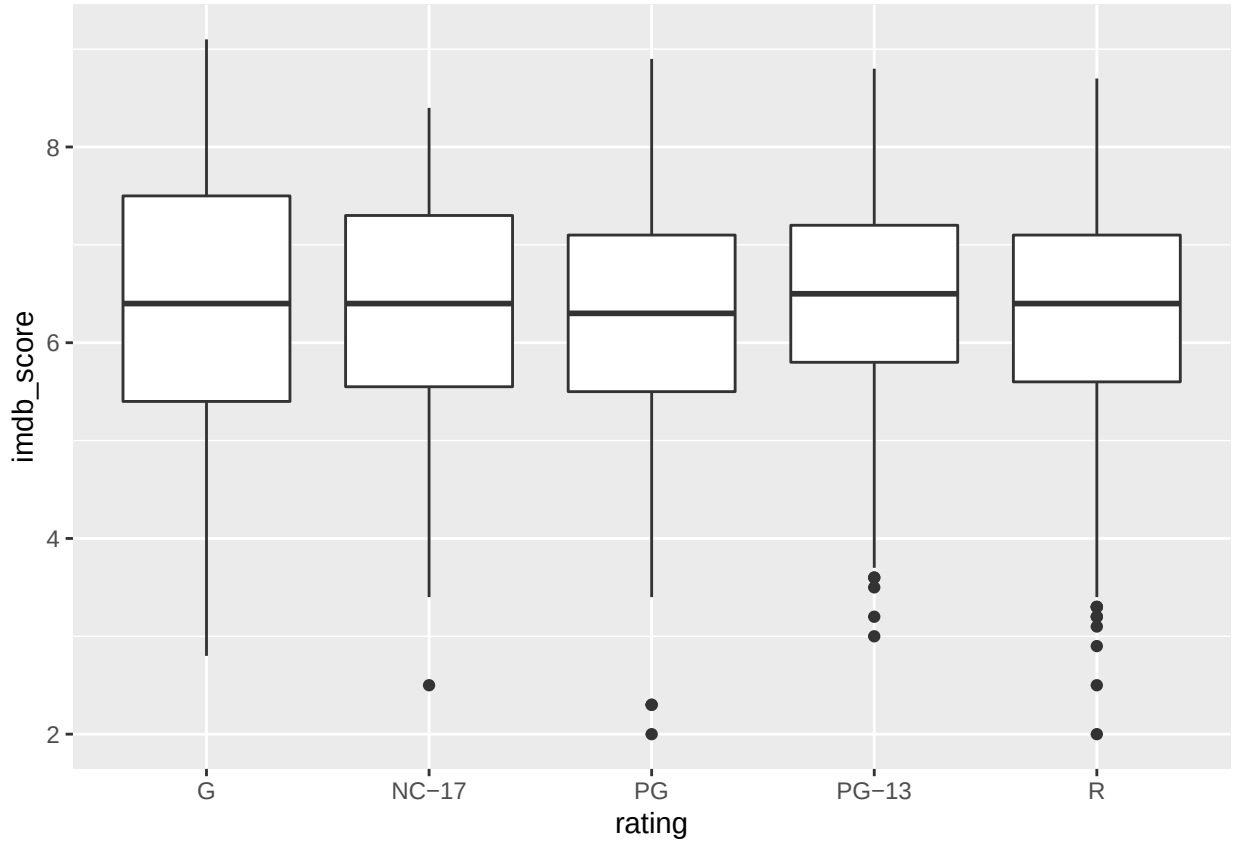
Exercise 3: Create a Box Plot

You have explored plotting the distribution of a single variable (histogram), two continuous variables (scatter plot), and two continuous variables in relationship to a categorical variable (scatter plot with color and small multiples). What if you want to see the difference between two or more levels of a categorical variable in terms of a continuous variable? For example, you want to know whether median IMDB scores and their variability differ by movie rating (G, PG, PG-13, etc.). One option is to create a boxplot.

Filter and Plot the Data

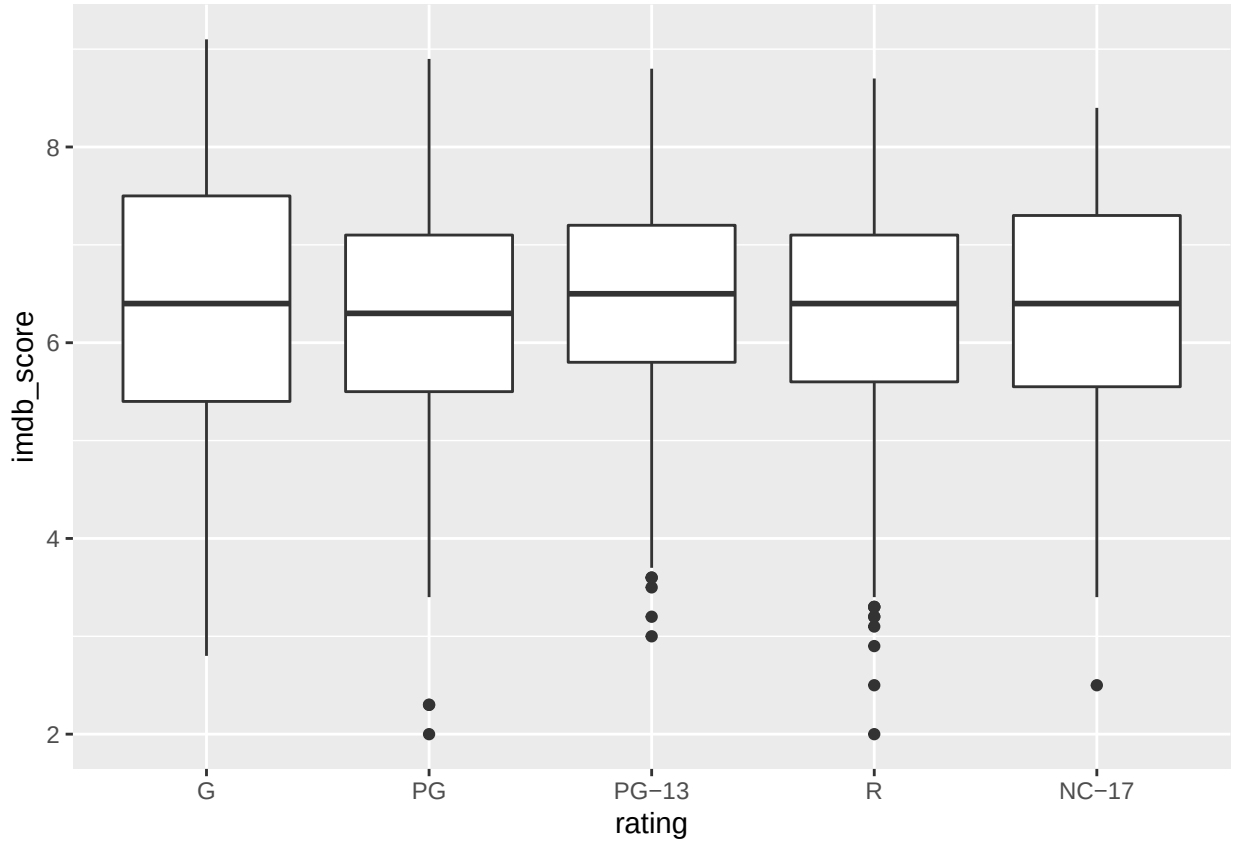
First filter out any movies that do not have a rating using `!is.na()`, then create the plot.

```
movies %>%
  filter(!is.na(rating)) %>%
  ggplot(aes(x = rating, y = imdb_score)) +
  geom_boxplot()
```



The default is for boxes to be ordered alphabetically based on the value of x. However, sometimes you will want them ordered in a specific way. One option is to convert the x variable to a factor using `mutate`.

```
ordered_box <- movies %>%
  filter(!is.na(rating)) %>%
  mutate(rating = factor(rating, levels=c("G", "PG", "PG-13", "R", "NC-17"))) %>%
  ggplot(aes(x = rating, y = imdb_score)) +
  geom_boxplot()
ordered_box
```

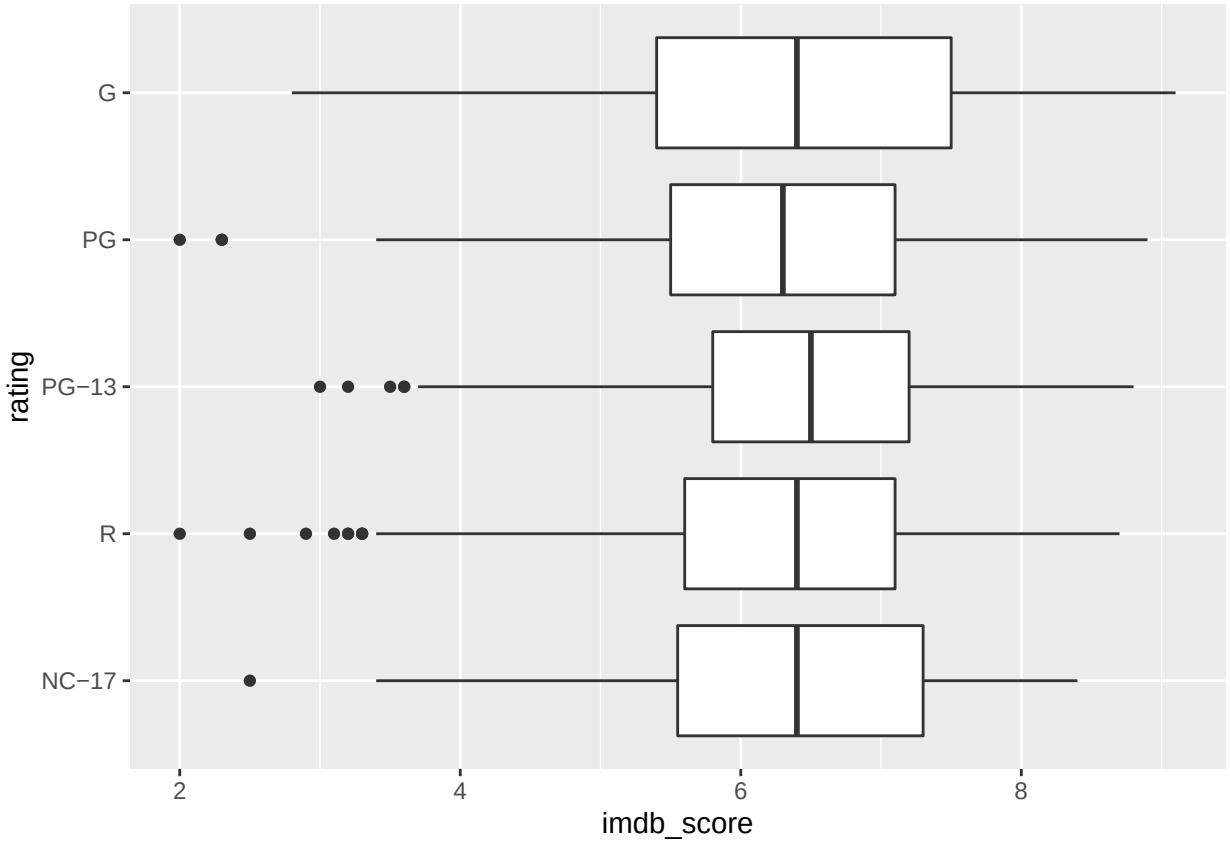


It appears that IMDB scores are pretty consistent across age certification ratings.

Flip Coordinates

Although not evident in the current box plot, sometimes the values on the x axis become overcrowded. One solution is to flip the coordinates so that the categorical variable appears on the y axis and the continuous variable on the x axis. Note that you may need to adjust the order of the categorical variable levels by passing the argument `limits = rev` to the function `scale_x_discrete`. This will ensure that the first factor appears at the top of the y scale.

```
ordered_box +
  coord_flip() +
  scale_x_discrete(limits = rev)
```



Exercise 4: Create a Bar Graph

Unlike a histogram, a bar graph typically represents frequencies, means, or other statistics for various levels of a categorical variable. In the next exercise, you will identify the top four movie genres (by frequency) and plot their mean IMDB ratings.

Prepare the Data

To identify the top four most common movie genres and calculate their means, you will use `dplyr` functions and save the results to `top_genres`.

```
top_genres<- movies %>%
  group_by(genre) %>%
  summarize(mean = mean(imdb_score, na.rm = TRUE), n = n()) %>%
  arrange(desc(n)) %>%
  slice(1:4)
top_genres
```

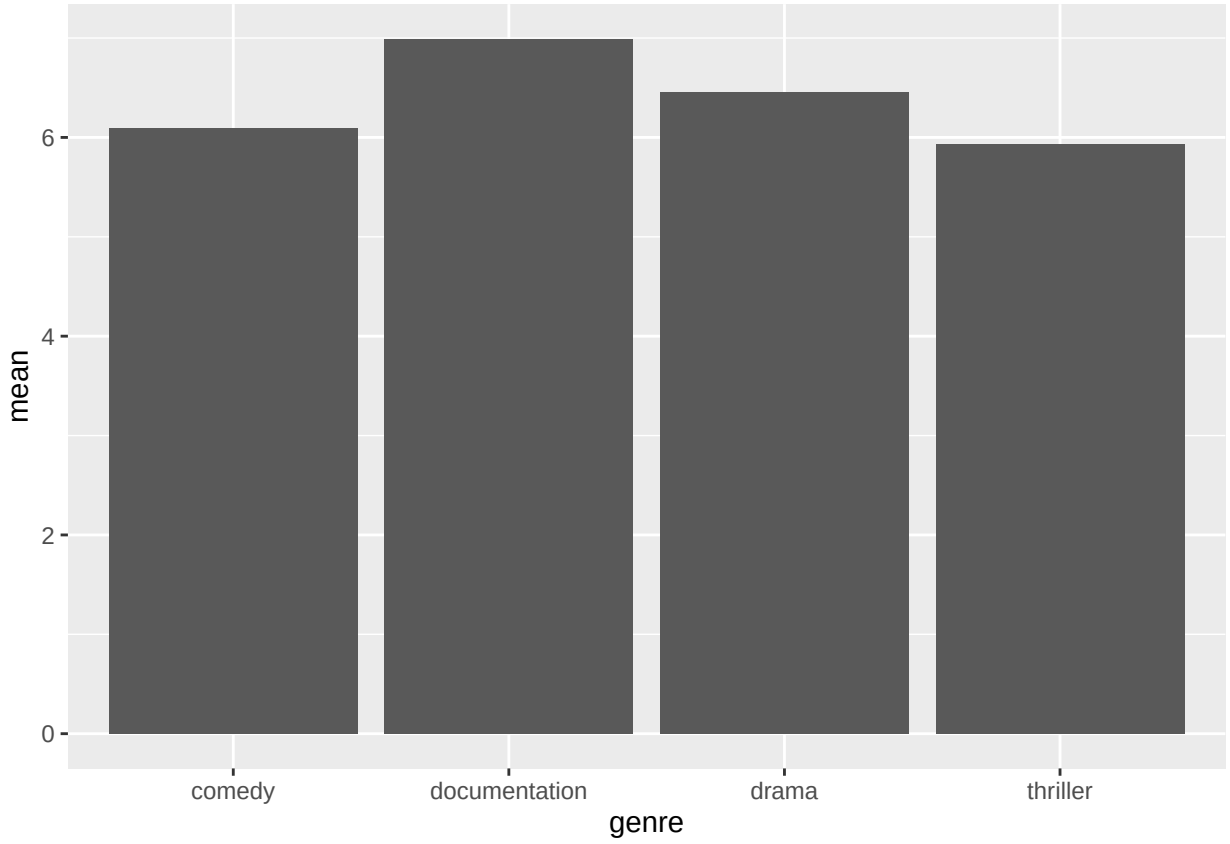
```
## # A tibble: 4 x 3
##   genre          mean     n
##   <chr>         <dbl> <int>
## 1 comedy         6.09   961
## 2 drama          6.46   882
## 3 documentation  6.99   414
```

```
## 4 thriller       5.94   311
```

The top four genres are comedy, drama, documentation (i.e., documentary), and thriller. Their means range from 5.93 (thriller) to 6.99 (documentation). Next, create a bar plot for the means. Note that the aesthetic mapping includes two variables: x for genre and y for mean. (You can also drop the `mapping =` prefix to the `aes()` function as long as it is the second argument in the `ggplot` function.)

Create a Bar Plot using `geom_col()`

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

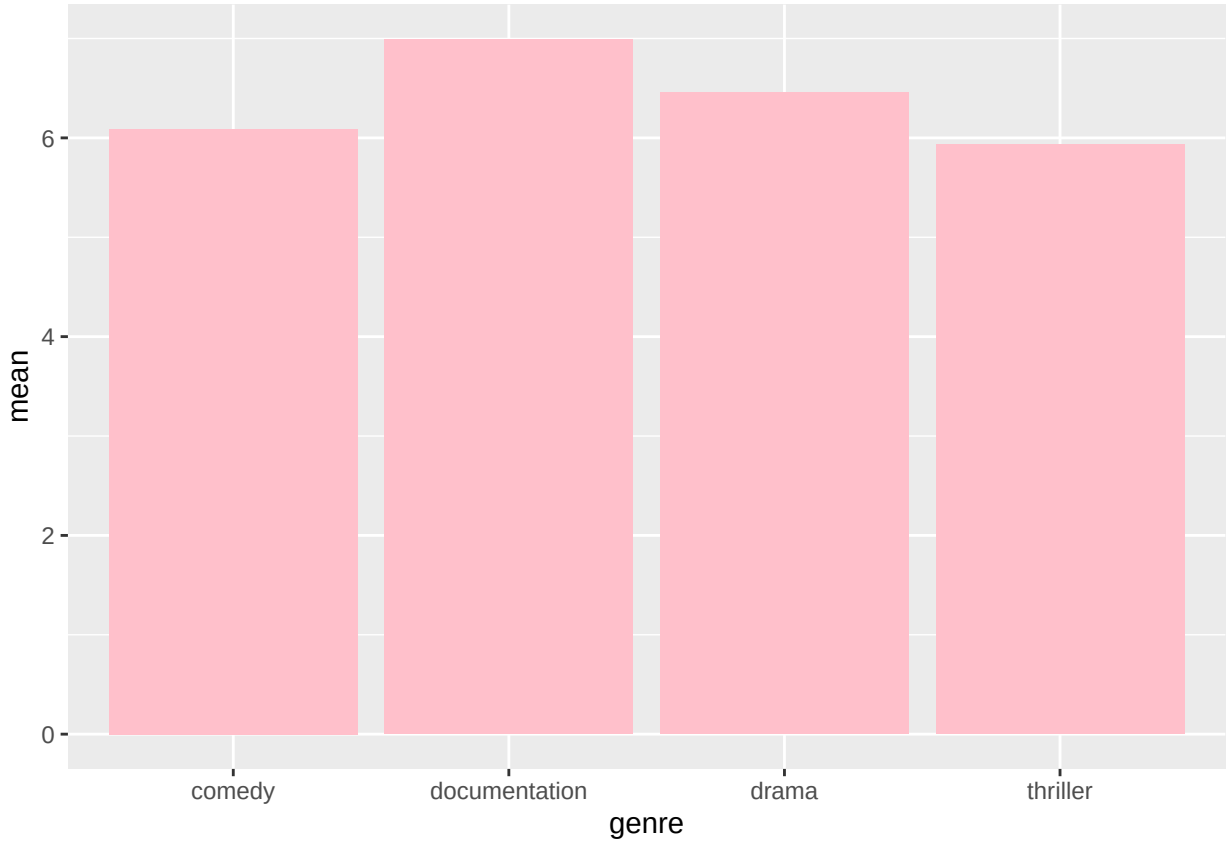


The result is functional but rather boring. Add some visual interest with color.

Add Color to the Bar Plot

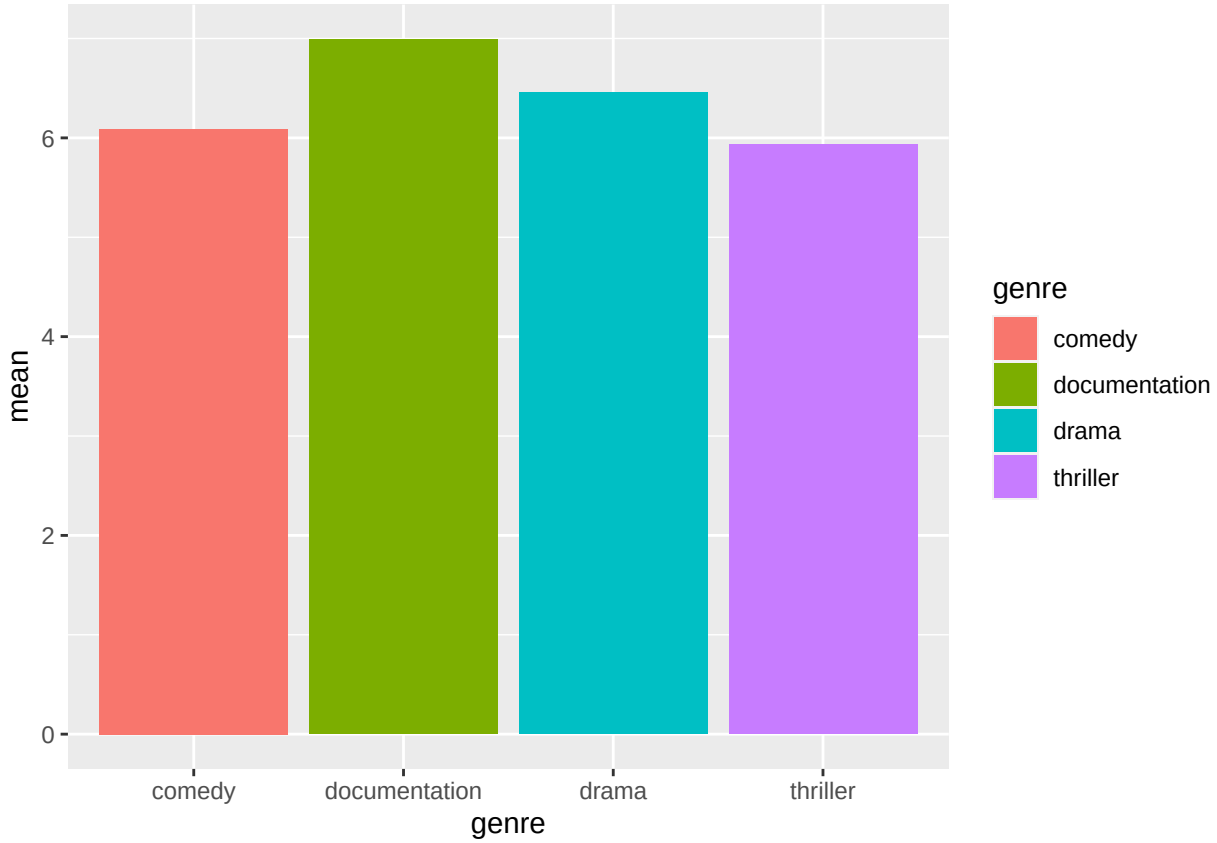
The basic method for adding color to the bars is to specify the `fill` argument within the `geom_col()` function:

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



However, if you want each bar to be a different color, you must map a variable to a third aesthetic: `fill`:

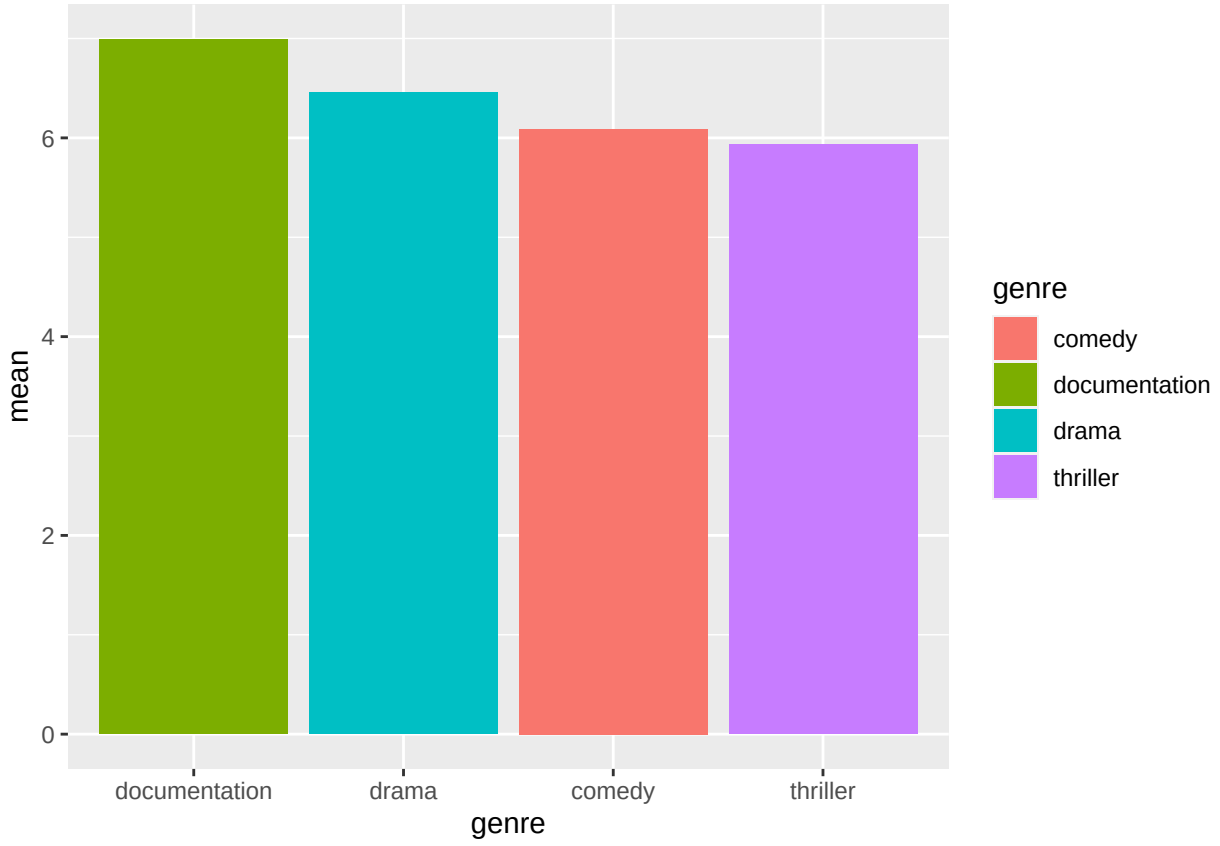
```
top_genres_bar <- ggplot(data = top_genres, aes(x = genre, y = mean, fill = genre)) +
  geom_col()
top_genres_bar
```



Re-order the X Axis

By default, the x axis is ordered alphabetically by the levels of the categorical variable. You might want to re-order the bars in some other way, such as by value. To do this, use the `x_scale_discrete()` function and save the plot to `ordered_genres_bar`.

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

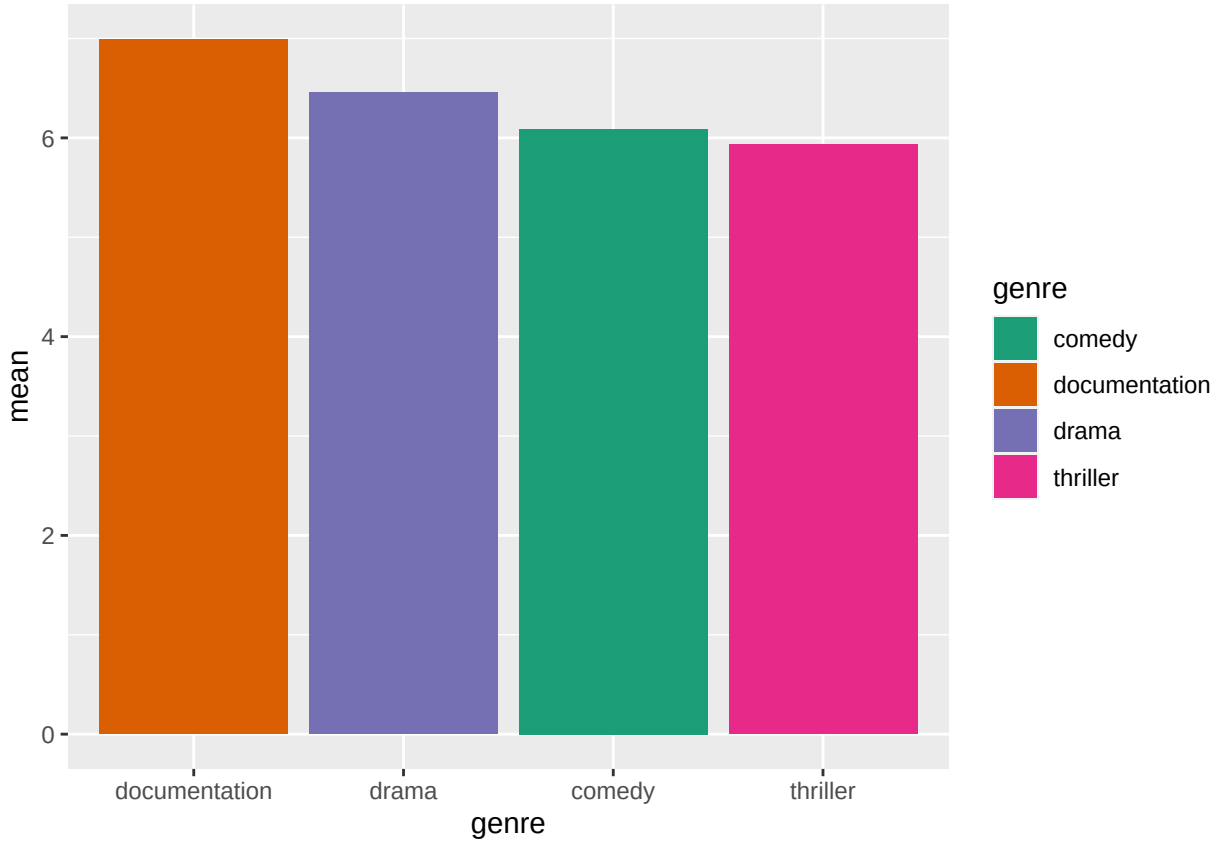


The result shows the genre with the largest mean (comedy) on the far left.

Use a Different Color Palette

If the color palette is not to your liking, you can use one of several options within the `RColorBrewer` packages, which is installed in base R.

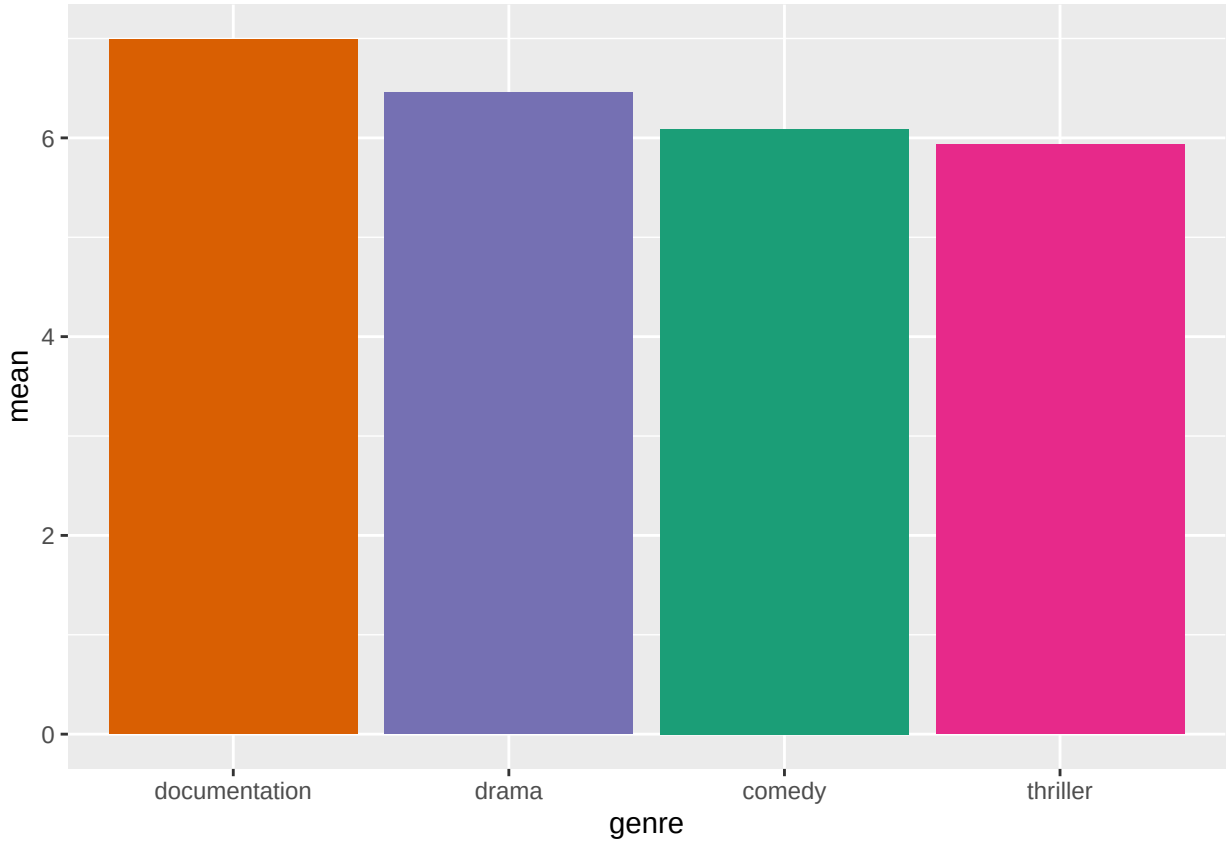
```
ordered_genres_bar_bold <- ordered_genres_bar +
  scale_fill_brewer(palette = "Dark2")
ordered_genres_bar_bold
```



Remove the Legend

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

```
ordered_genres_bar_bold +
theme(legend.position = "none")
```



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.