ezplot: How to Easily Make *ggplot2* Graphics for Data Analysis

Guangming Lang

ezplot: How to Easily Make ggplot2 Graphics for Data Analysis

Guangming Lang

This book is for sale at http://leanpub.com/ezplot

This version was published on 2016-03-28

Tweet This Book!

Please help Guangming Lang by spreading the word about this book on Twitter!

The suggested tweet for this book is:

If you make data visualizations, you need to check out ezplot.

The suggested hashtag for this book is #ezplot.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#ezplot

Also By Guangming Lang

Score Personal Loan Applicants using R

Contents

Preface

This book will teach you two things: how to make good statistical charts for reports or presentations, and how to do it fast. The tool we'll use is a R package called ezplot, which I made to help me better do my own work. Three years ago, I started my own data science consulting company[1]. Since then, almost every project I took requires some sort of data visualization. A picture is worth a thousand words, and this is also true when showing insights from data. But making a correct and engaging "picture" is no easy feat. Hadley's ggplot is a great tool for that, but I didn't want to remember the numerous detailed commands for customizing and annotating a ggplot. So naturally, I did a lot of code recycling, but every time I copy-n-pasted a chunk of old code, I had to manually change the data frame or variable name to make it work for the new situation. This worked fine for one or two plots, but became very tedious and hence annoying when you had to make 60 of them. Even worse, as my code base piled up, it became a pain in the ass to find the right piece of code for the kind of customizations I needed to do. On top of that, code recycling lead my R scripts to grow large quickly, which made them hard to read, yet those repetitions could be eliminated if given enough thoughts. So finally one day, I sat down and wrote ezplot to change it all. I now use ezplot for making all my static charts, and it has made me much happier. A plot that used to take me 1 hour to finish now takes me less than 5 minutes. I truly love it, and I think you'll love it too.

After working through this book, you will be able to make the following 10 most used statistical charts in less than one-tenth of the time you use now.

- histogram
- density plot
- boxplot
- interval plot
- scatter plot
- barplot
- areaplot
- lineplot
- slopeplot
- heatmap

You'll also learn how to use color-blind friendly colors to the best effect. In addition, you'll be able to easily and quickly customize and anotate your charts, for example, change the scale of the axes to `log` or `percent`, enlarge the axis tick labels for web display, and etc.

You will get most out of this book by typing and running the code given in the book. Do NOT just copy and paste. Type the code. This will help you become a better R programmer. If you run into typos or errors, please let me know at gmlang@cabaceo.com.

Good luck and Happy Learning!

[1] http://www.cabaceo.com

Set up

1. Install R[2] and Rstudio[3].
2. Install a set of development tools:
    - On Windows, download and install Rtools[4].
    - On Mac, install the Xcode command line tools[5].
    - On Linux, install the R development package, usually called **r-devel** or **r-base-dev**.
3. Install the following R packages.

```
install.packages("dplyr")
install.packages("tidyr")
install.packages("devtools")
devtools::install_github("hadley/scales")
devtools::install_github("hadley/ggplot2")
devtools::install_github("jrnold/ggthemes")
devtools::install_github("gmlang/ezplot")
```

Note: throughout this book, if when displaying plots, you encounter an error like this, `Error in grid.Call(L_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : polygon edge not found`, just run `dev.off()` first, and then `print(p)`.

[2] http://www.r-project.org

[3] http://www.rstudio.com/products/rstudio/download/

[4] http://cran.r-project.org/bin/windows/Rtools/

[5] https://developer.apple.com/downloads

10 Most Used Charts

Previously, we learned when and how to use the function `plt_dist()`. It conveniently draws 3 of the 10 most common statistical charts in one figure: histogram, density plot, and boxplot. The other 7 are interval plot, scatter plot, bar plot, area plot, line (or timeseries) plot, slope plot, and heatmap. In this chapter, I'll show you how to use ezplot to make these charts with many nuances and variations. By the end of this chapter, you'll be able to make them yourself, and you'll see your productivity soaring, which will translate to a happy sensation that you can feel physically.

Let's get started.

Histogram

As we've already seen from the last chapter, histogram, density plot and boxplot are for displaying the distribution of a continuous variable. We'll look at histogram in this section, density plot and boxplot in the next two sections. Before we start, make sure you load the `ezplot` library. The ezplot function we'll use is `mk_distplot()`.

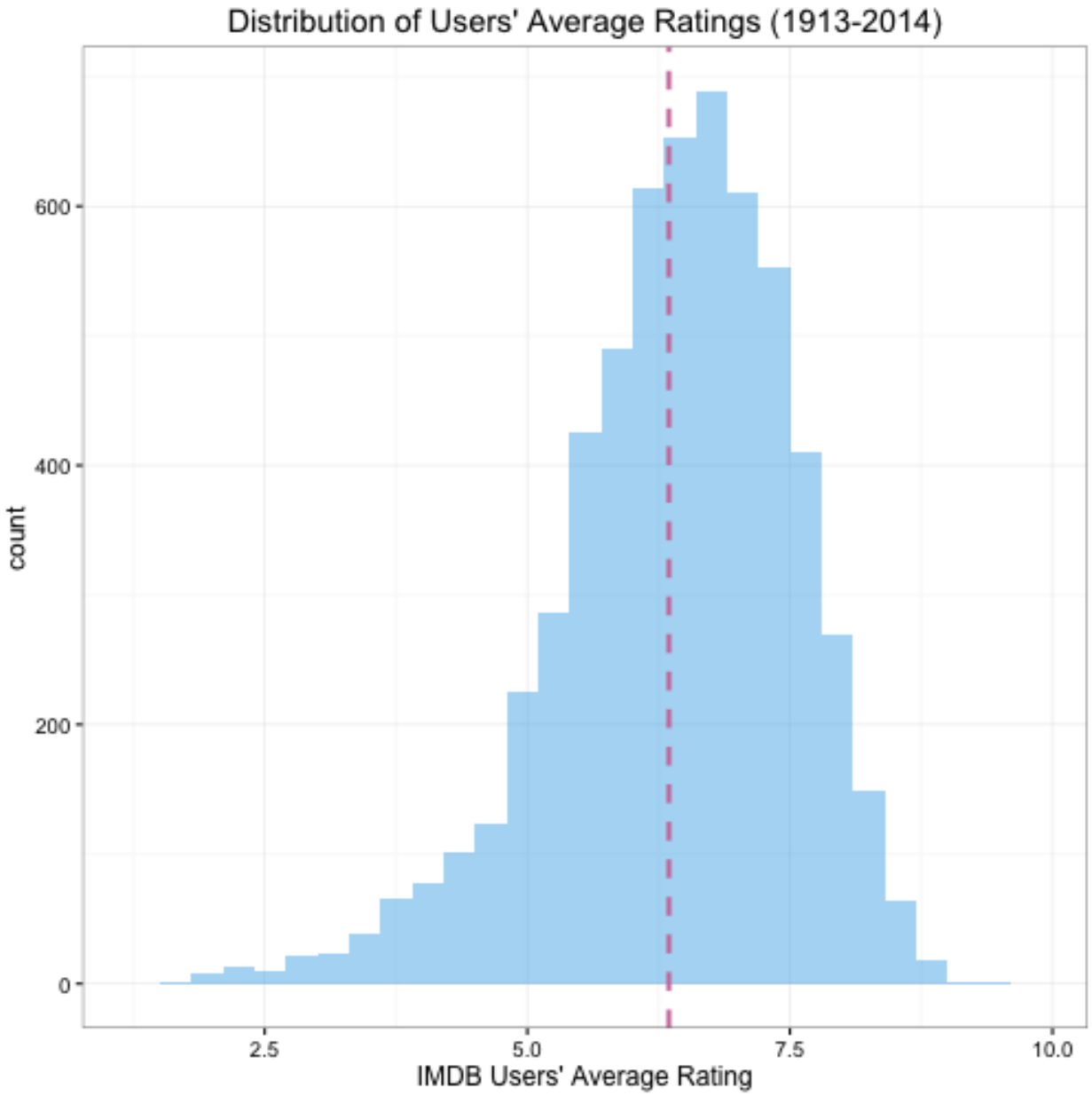
```
library(ezplot)
```

The `ezplot` library comes with a dataset of films. You can run the command `?films` to see the descriptions of its variables. If you do that, you'll see that it has a variable called `rating`, which measures the IMDB users' average ratings. Now, check its type by running the command `str()`.

```
str(films$rating)
 num [1:5944] 6.2 4.7 7.8 7.5 7.2 6.2 5 5.6 5.6 4.8 ...
```

The output `num` says that R treats it as a numeric variable. A numeric variable can be discrete (whole numbers) or continuous (decimal numbers). From the first few data values (6.2, 4.7, ...) shown in the output, it's clear that `rating` is continuous. And we can draw a histogram to help us understand its distribution.

```
plt = mk_distplot(films)
title = "Distribution of Users' Average Ratings (1913-2014)"
p = plt("rating", binw=0.3, xlab="IMDB Users' Average Rating", main=title,
        add_vline_mean=T)
print(p)
```



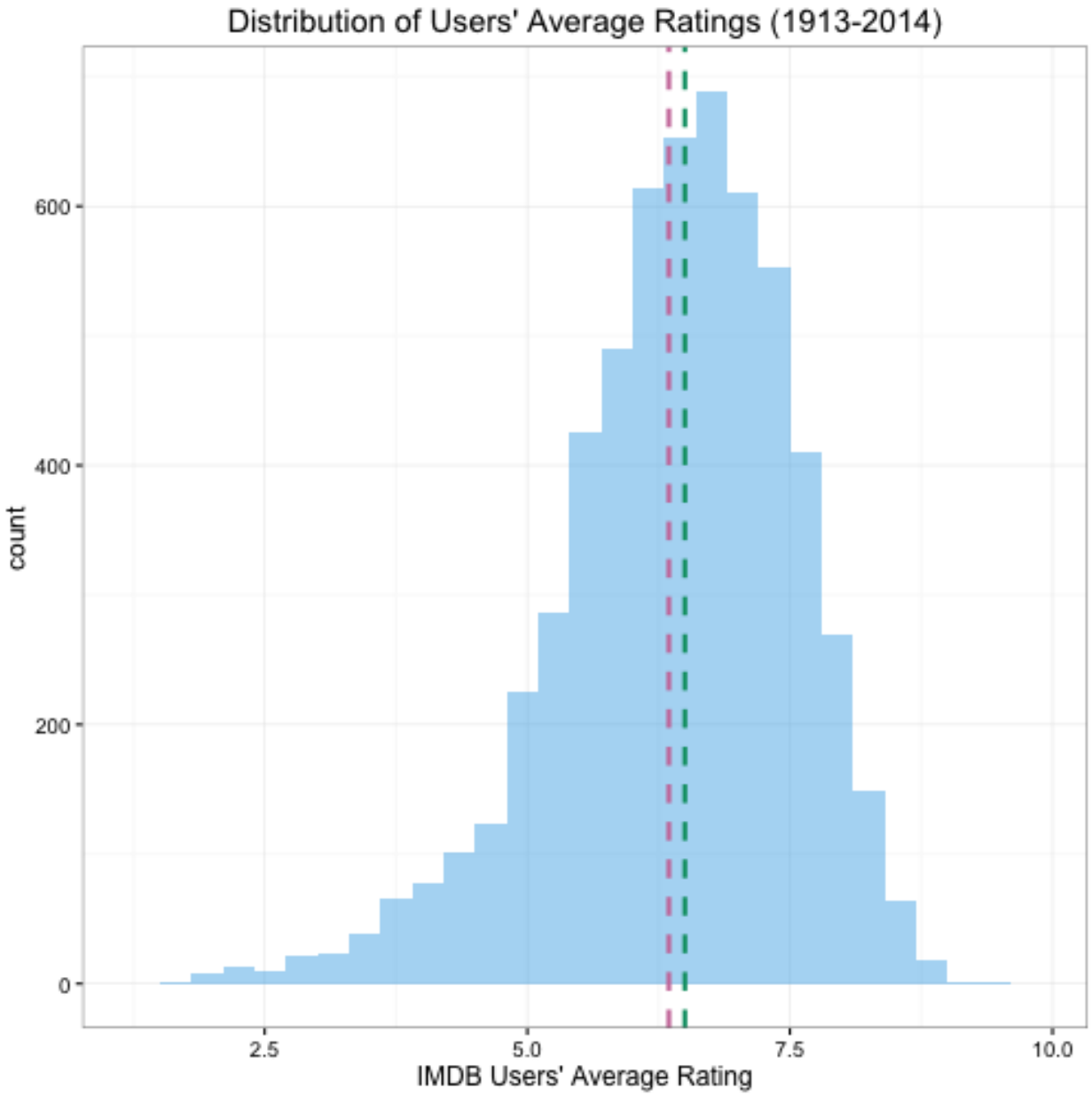
Avg. Ratings with Mean Line

Notice how we used `mk_distplot()`. We passed it the data frame `films` as input. It returned a function, and we assigned it to the variable `plt`. We then used the function `plt()` to draw the histogram. Note that once again, we had to feed `plt()` the variable name, in this case, `"rating"` in quotes, instead of the variable itself. Do you recall the same usage from the last chapter? Yes, this is how you use every plotting function in ezplot. I assure you this usage pattern will reappear again and again in later chapters.

Here're some explanations for the other parameters: a histogram is made of bins, and `binw` is the width of the bins; `xlab` is the x-axis label; `main` is the plot title; `add_vline_mean` decides whether

to add a vertical line at the mean or not, and when setting `add_vline_mean=T`, it draws a purple line vertically at the mean value of the x's, just like shown in the above histogram. We can also add a vertical line at the median by setting `add_vline_median=T`, and as shown by the plot below, the median vertical line is colored green.

```
p = plt("rating", binw=0.3, xlab="IMDB Users' Average Rating", main=title,
        add_vline_mean=T, add_vline_median=T)
print(p)
```



Avg. Ratings with Mean and Median Lines

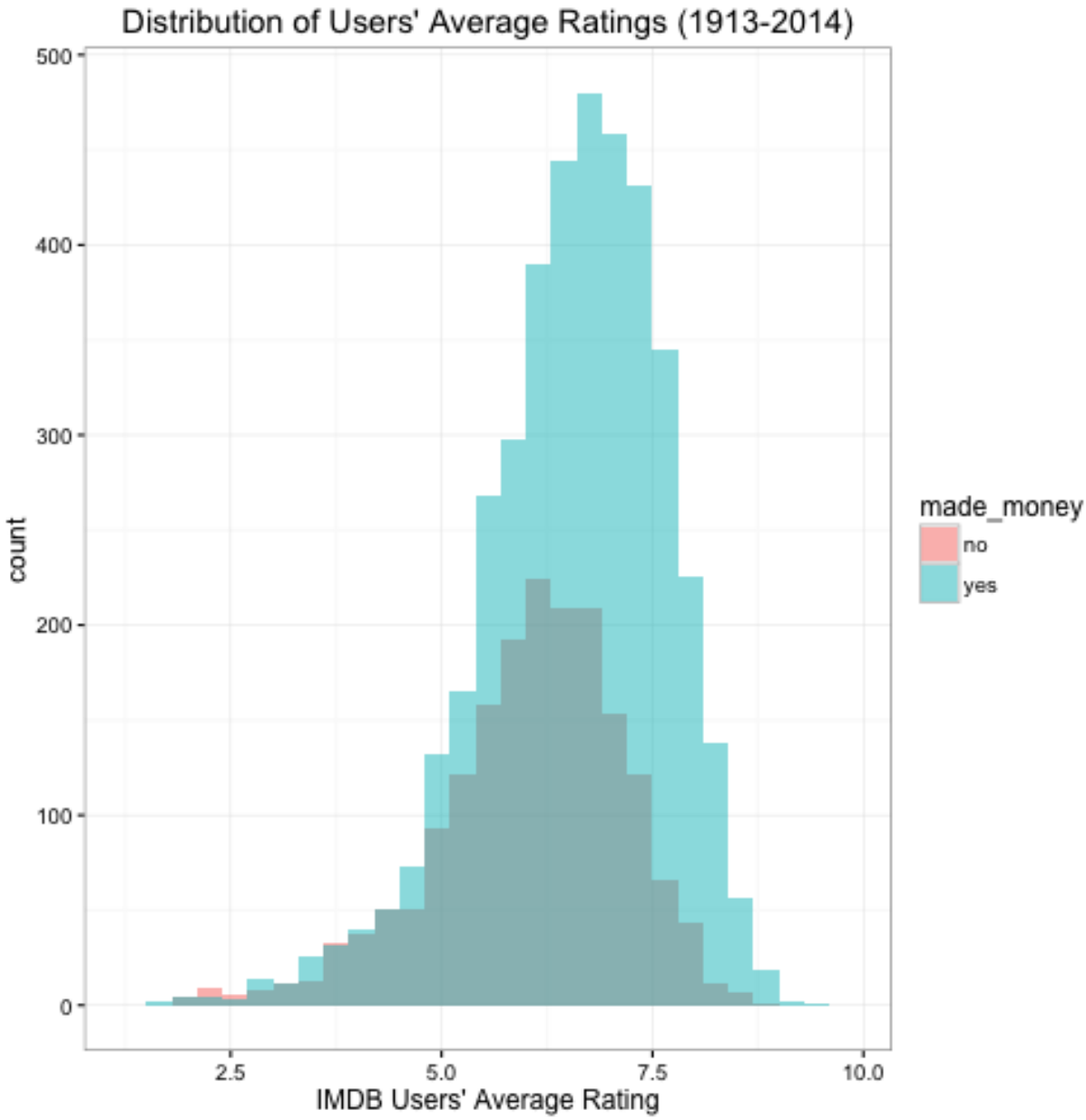
Now, I want you to run `?mk_distplot()` and read the complete description line by line. When you meet the word `fillby`, stop and read it again. Do you understand what it does? No? That's ok because that's what we're talking about next. The data frame `films` has a variable called `made_money`. Let's use `str()` to check its type.

```
str(films$made_money)
```

```
 Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 2 1 2 2 ...
```

We see `made_money` is a factor with two levels: yes, no. It indicates if a film made money or not. We've looked at the overall distribution of the users' average ratings. An interesting question is what do the distributions of the users' average ratings look like for films that made money vs. the ones that lost money. It turns out we can easily answer this question pictorially by setting `fillby = "made_money"`.

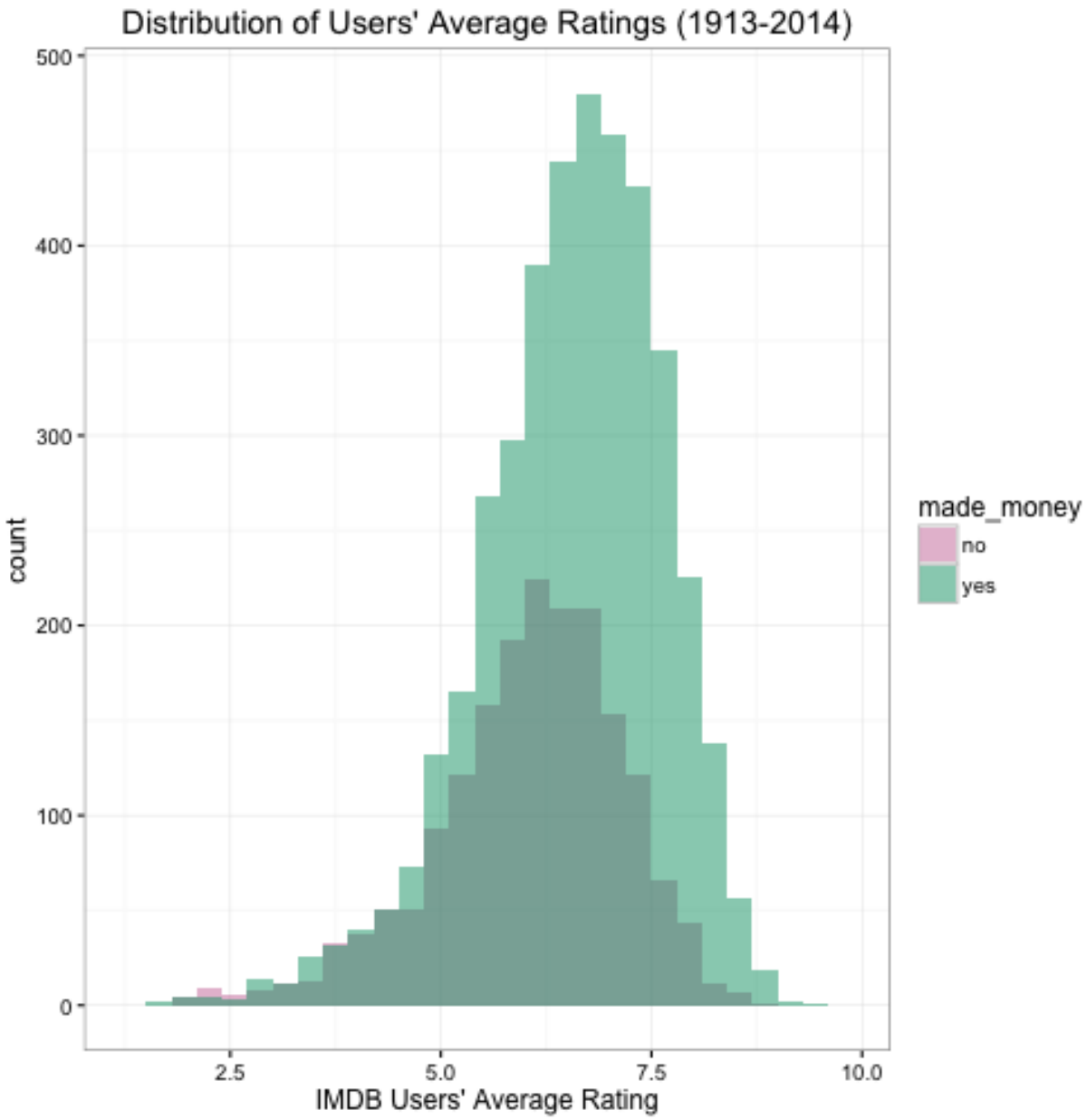
```
p = plt("rating", fillby="made_money", binw=0.3,
        xlab="IMDB Users' Average Rating", main=title)
print(p)
```



Avg. Ratings of Two Film Groups

One thing I'd like to point out: the default red and green colors in the above plot are not color-blind friendly. And it's good to use color-blind friendly colors whenever possible. Luckily, the ezplot package comes with 9 color-blind friendly colors, and we'll now use the color-blind friendly versions of red and green to re-do the above chart.

```
red = cb_color("reddish_purple")
green = cb_color("bluish_green")
p = p + ggplot2::scale_fill_manual(values = c(red, green))
print(p)
```



Avg. Ratings of Two Film Groups, Color-blind Friendly

We see the green histogram (films that made money) completely overshadows the red histograms (films that didn't make money or lost money). What conclusion can you draw? Write it down. Now, let's check the size of each group.

```
table(films$made_money)
  no  yes
1831 4113
```

We see the number of films that made money is more than twice of the number of films that didn't make money or lost money. Does this change your initial conclusion? What's your new conclusion?

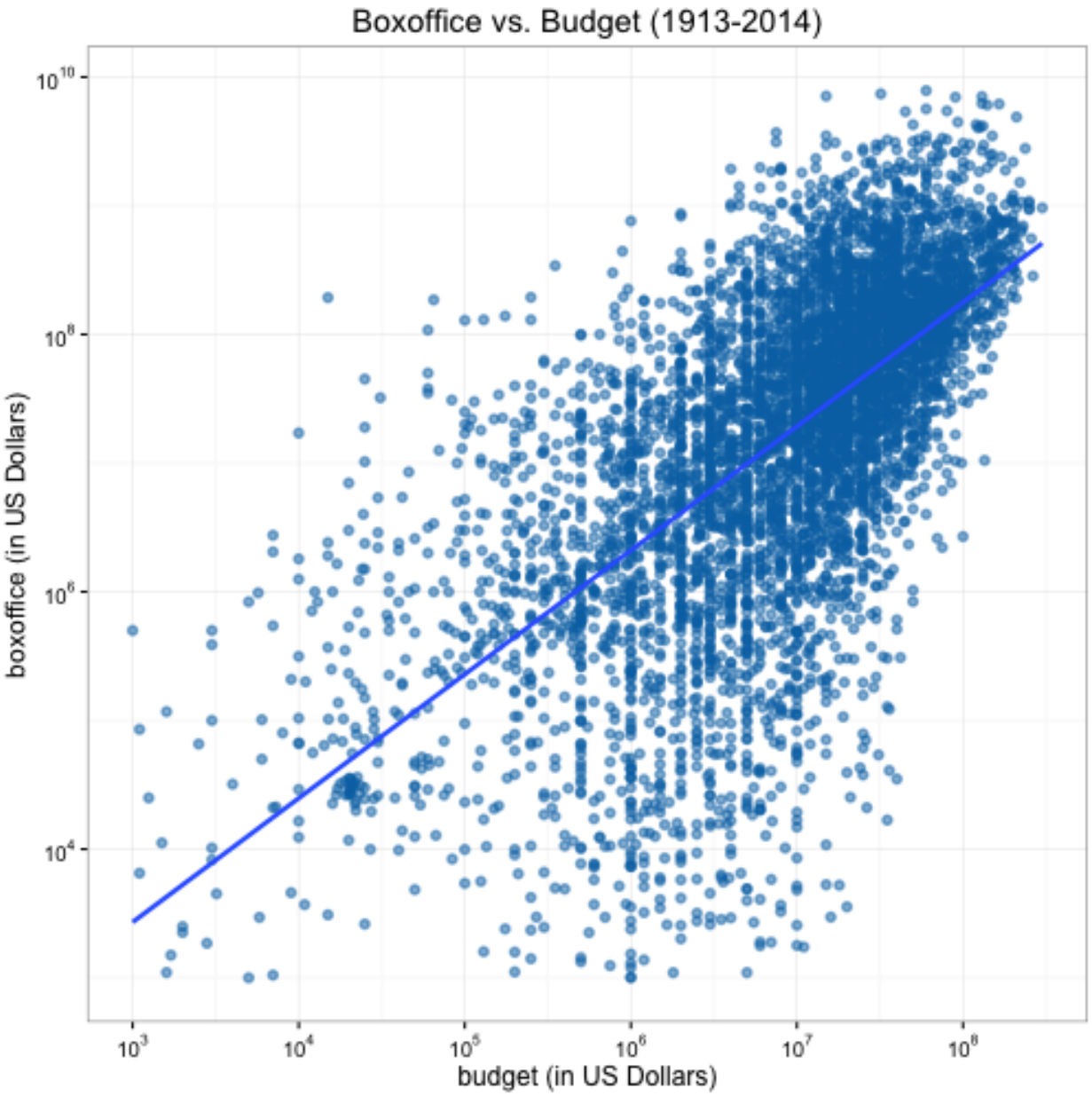
Scatter Plot

A scatter plot is good for showing the relationship between two continuous variables. Let's call the `mk_scatterplot()` function on the data frame `films` to obtain a function that we can use to make scatter plots for any two continuous variables in `films`.

```
library(ezplot)
plt = mk_scatterplot(films)
# get color-blind friendly colors
purple = cb_color("reddish_purple")
green = cb_color("bluish_green")
```

For example, we can use `plt()` to draw a scatter plot to reveal if there's a relationship between `boxoffice` and `budget`, and if there is, what kind of relationship it is.

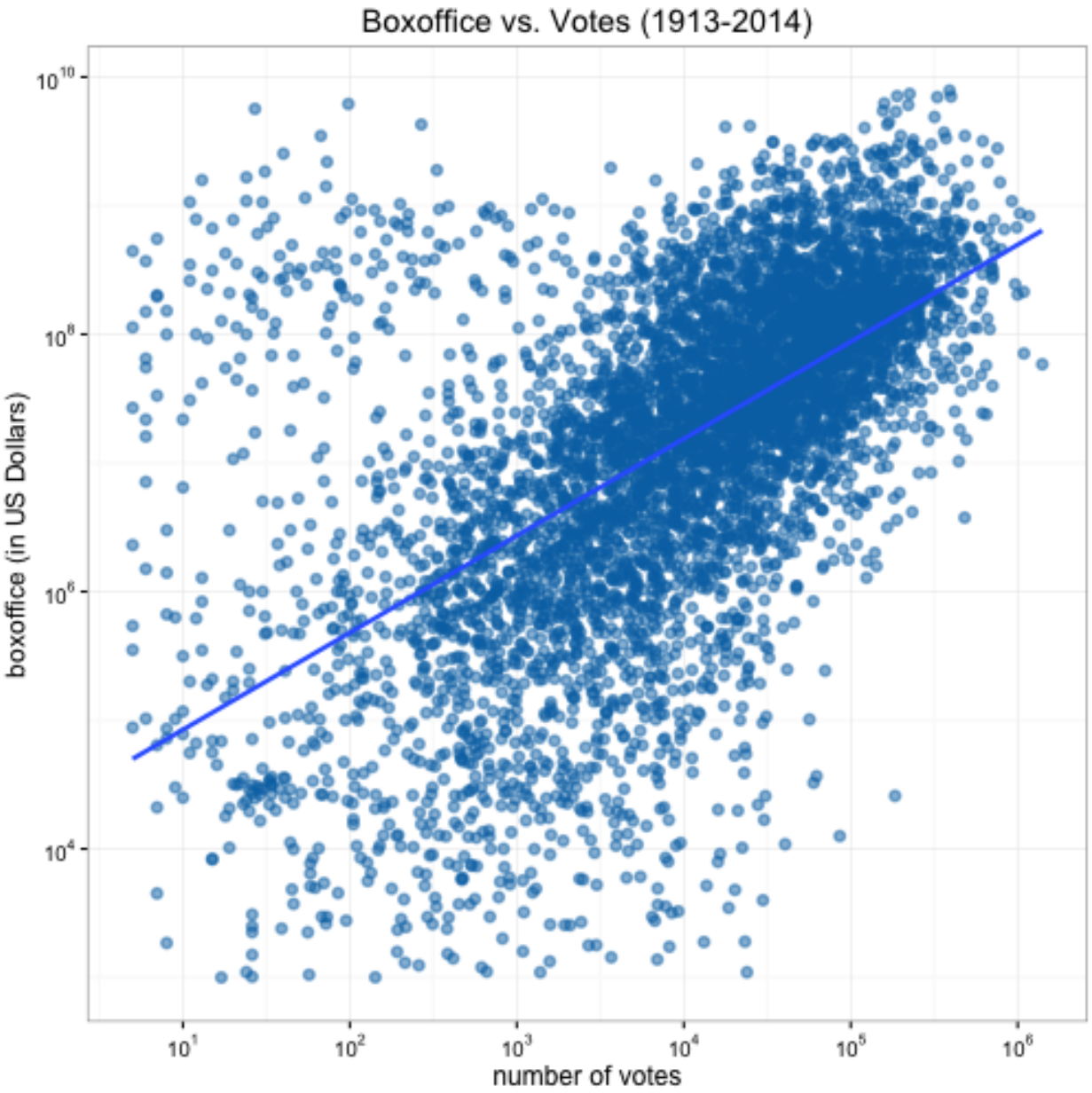
```
# make plot title
title = "Boxoffice vs. Budget (1913-2014)"
# plot boxoffice vs. budget
p = plt(xvar="budget", yvar="boxoffice", xlab="budget (in US Dollars)",
        ylab="boxoffice (in US Dollars)", main=title,
        pt_size=1.2, pt_alpha=0.5, add_line=T)
# use log10 scale on both axes
p = scale_axis(p, scale="log10") # default is y-axis
p = scale_axis(p, axis="x", scale="log10")
# display plot
print(p)
```



Boxoffice vs. Budget

Like mentioned in previous chapters, the function `plt()` can be re-used. Just give it two different variable names. For example, we now use it to draw a scatter plot of `boxoffice` vs. `votes`.

```
title = "Boxoffice vs. Votes (1913-2014)"
p = plt("votes", "boxoffice", xlab="number of votes",
        ylab="boxoffice (in US Dollars)", main=title,
        pt_size=1.5, pt_alpha=0.5, add_line=T)
p = ezplot::scale_axis(p, scale="log10")
p = ezplot::scale_axis(p, "x", scale="log10")
print(p)
```

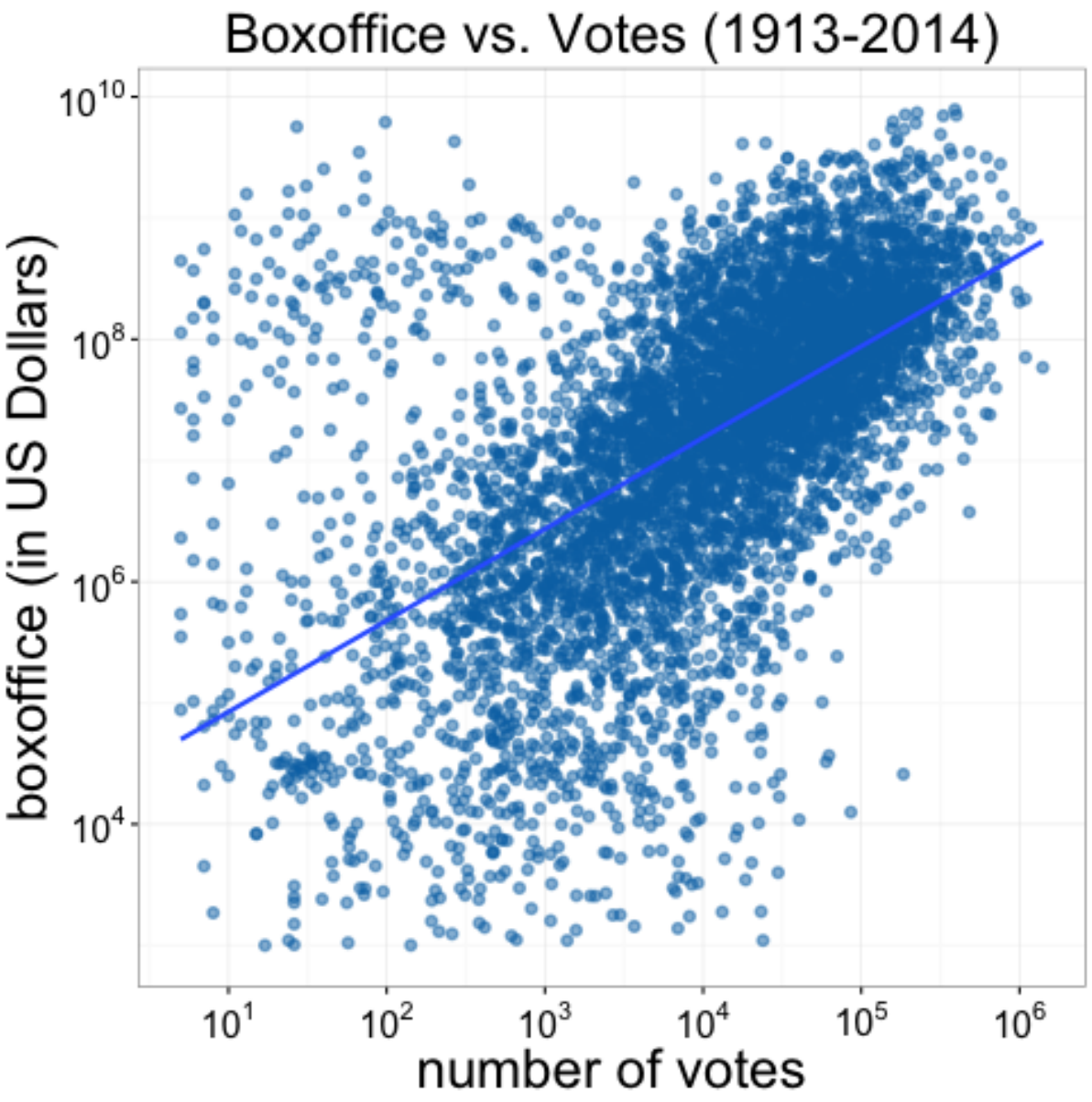


Boxoffice vs. Votes

One thing I want to point out is that the plots created by default is too small for web display. If you are making a analytic web app[6], you want bigger font size, bigger point size and bigger everything. Luckily, the `web_display()` function does that for you. Many times, all you need to do is to pass the plot object to it, and it'll return a plot object that just works.

[6] http://app.cabaceo.com/ocpu/github/gmlang/imdb/www/#/bo

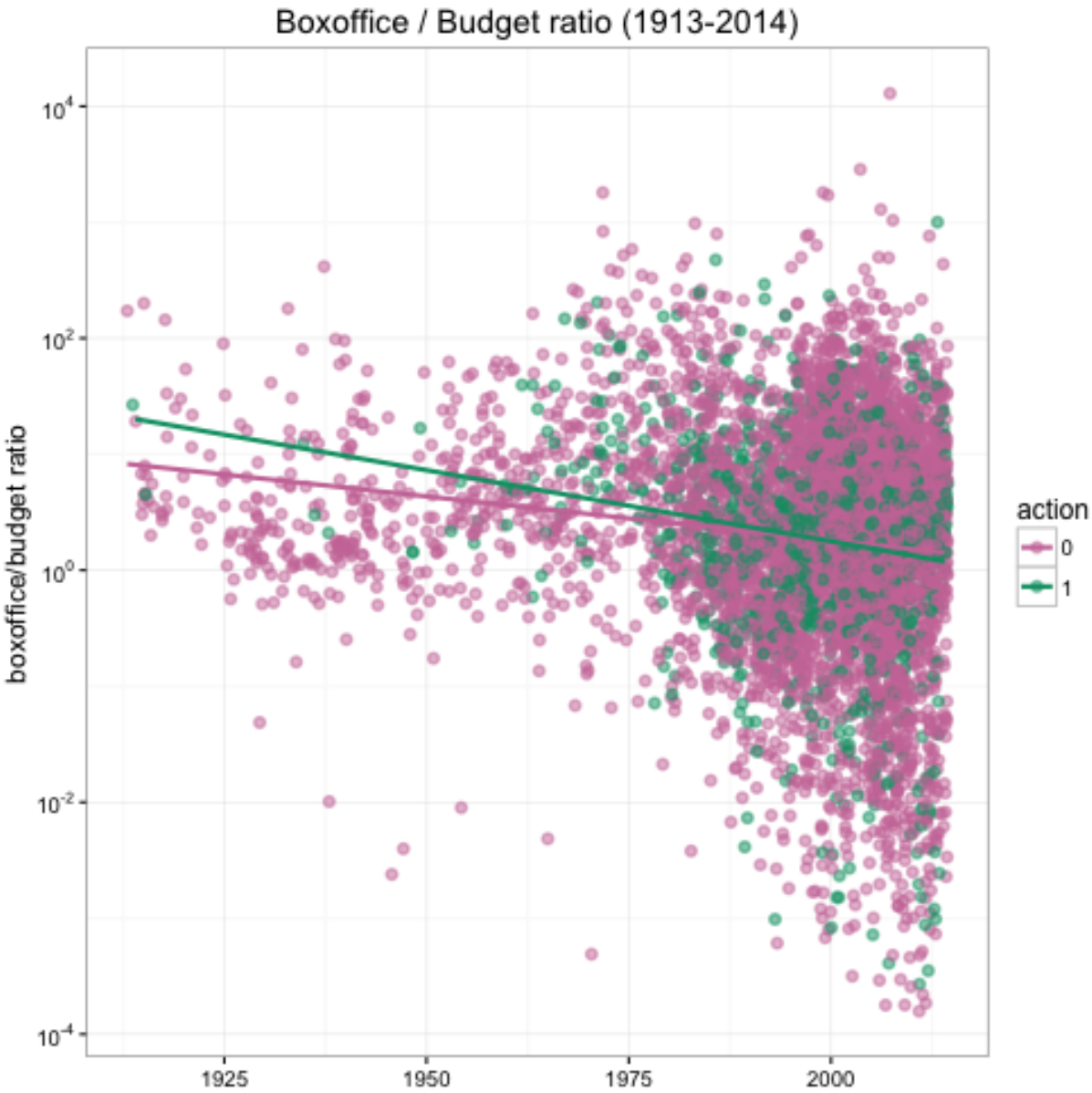
```
web_display(p)
```



Boxoffice vs. Votes for Web

We can also use a categorical variable to color the points. To demo this, let's first ask ourselves an interesting question: did action movies make money year after year? To answer it, we'll need to draw a scatter plot of `bo_bt_ratio` vs. `year` and color the points by the binary factor `action`.

```
p = plt(xvar="year", yvar="bo_bt_ratio", fillby="action",
        ylab="boxoffice/budget ratio",
        main="Boxoffice / Budget ratio (1913-2014)",
        pt_size=1.5, pt_alpha=0.5, add_line=T)
# use log10 scale on y-axis
p = scale_axis(p, scale="log10")
# use color-blind friendly colors
p = p + ggplot2::scale_color_manual(values = c(purple, green))
# display plot
print(p)
```



Boxoffice vs. Budget colored by action film indicator

The green dots are action films, while the purple dots are non-action films. First, notice there are more purple dots than green dots. Second, our impression is that the green line has a steeper negative slope, and if we pay attention to the green dots before 1962, we'll see the reason is because action films always made money before 1962 (none of the green dots before 1962 are below the y = 1 line), while non-action films weren't as lucky as the action films. And after 1962, some action films also started losing money.

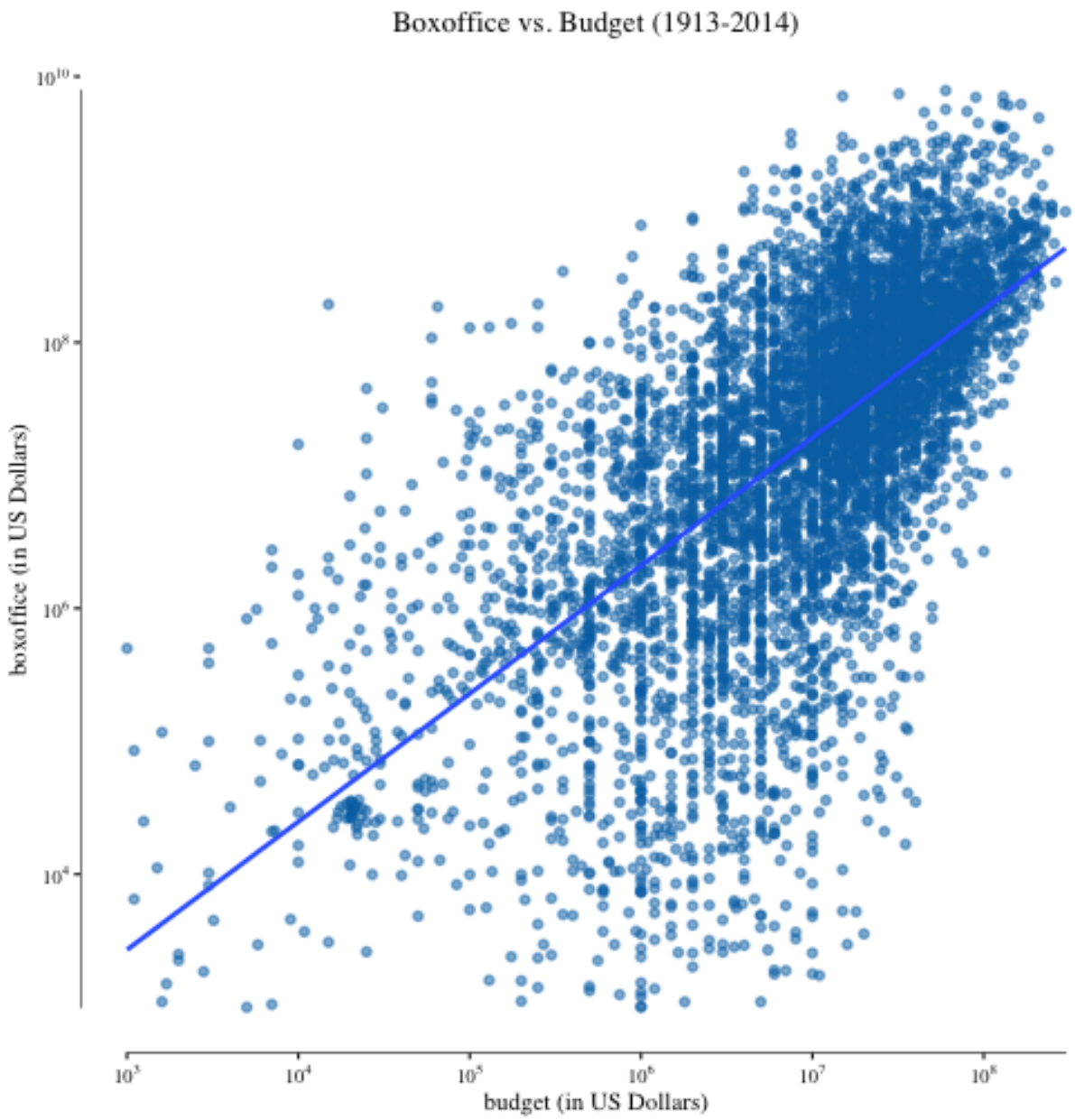
Now, it's your turn. Make scatter plots to answer the following questions: 1. Did drama make money year after year? What about comedy? 2. Was it true that the higher the `rating`, the bigger

the boxoffice/budget ratio? What about when looking at romance films vs. non-romance films? 3. Was it true that the more `votes` a films got, the bigger its the boxoffice/budget ratio? What about when looking at drama vs. non-drama?

Finally, let's talk about style and taste. There's a well known name in information visualization, Edward Tufte[7], and he's famous for his minimalistic designs. For example, here's how it looks if we apply Tufte's design principles to the scatter plot of boxoffice vs. budget.

```
library(ggplot2)
library(ggthemes)
p = plt(xvar="budget", yvar="boxoffice", xlab="budget (in US Dollars)",
        ylab="boxoffice (in US Dollars)",
        main="Boxoffice vs. Budget (1913-2014)",
        pt_size=1.2, pt_alpha=0.5, add_line=T)
# use log10 scale on both axes
p = scale_axis(p, scale="log10") # default is y-axis
p = scale_axis(p, axis="x", scale="log10")
# apply Tufte's style
p = p + geom_rangeframe() + theme_tufte()
print(p)
```

[7] https://en.wikipedia.org/wiki/Edward_Tufte



Boxoffice vs. Budget, Tufte Style

Compare this plot with the first plot we drew in the beginning of this chapter, what differences do you notice? We'll talk more about Tufte styles in later chapters. For now, let's take a break. In the next chapter, we'll talk about bar chart, which is commonly used for showing the distribution of a quantitative variable. See you then.