Project Data Cleaning

Katrina Truebebach

March 4, 2019

```
rm(list = ls())
```

Discovery: The budget and revenue data are not in consistently US or local currency and there is no way to tell. Thus instead of the original plan to use an exchange rate and local deflator to get real US dollars, we are only using US films.

Data Quality Note: If IMDB lists multiple values for a variable, this dataset only gives the first or last listed. For example, duration often has multiple versions like director's cut - IMDB gives the last listed. Language also often has multiple - IMDB gives the first language. Same problem for country.

Download Data

Keep different file paths for different people

```
# Qiang's File Paths
#movie <- read_csv("D:/academic/DS 5110 Introduction to Data Management and Processing/project/movie_me
#oscar <- read_csv("D:/academic/DS 5110 Introduction to Data Management and Processing/project/academy_
#deflator <- read_csv("D:/academic/DS 5110 Introduction to Data Management and Processing/project/API_N
# Katrina's File Paths
movie <- read_csv("~/DS5110/data/movie_metadata.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   color = col_character(),
##   director_name = col_character(),
##   actor_2_name = col_character(),
##   genres = col_character(),
##   actor_1_name = col_character(),
##   movie_title = col_character(),
##   actor_3_name = col_character(),
##   plot_keywords = col_character(),
##   movie_imdb_link = col_character(),
##   language = col_character(),
##   country = col_character(),
##   content_rating = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

```
oscar <- read_csv("~/DS5110/data/academy_awards.csv")
```

```
## Parsed with column specification:
## cols(
##   Year = col_character(),
##   Ceremony = col_double(),
##   Award = col_character(),
##   Winner = col_double(),
##   Name = col_character(),
##   Film = col_character()
```

```
## )
```

```
deflator <- read_csv("~/DS5110/data/GDPDEF.csv")
```

```
## Parsed with column specification:
## cols(
##   DATE = col_character(),
##   GDPDEF = col_double()
## )
```

Movie Data Cleaning

Drop some variables we are not using. Rename some variables

```
# drop variables we aren't using
movie <- movie %>% select(-c(color, starts_with('num_voted'),
                             contains('link'), aspect_ratio, facenumber_in_poster,
                             plot_keywords))
# rename year variable so matches with keys in other datasets
# also rename country to country_name.
  # When merge with othere datasets later, need to differentiate between country abbreviations and full
movie <- movie %>% rename(year = title_year)
```

Clean up string variables: trim white space from all character variables

```
movie <- movie %>%
  mutate_if(is.character, str_trim)
```

Drop rows that are duplicated: just base on title and year. Other variables have small errors (small difference in number of facebook likes, difference in who is actor3 etc.) but those movies are still duplicates

```
sum(duplicated(movie %>% select(movie_title, year))) # 124
```

```
## [1] 124
```

```
movie <- movie[!duplicated(movie %>% select(movie_title, year)),]
```

Exclude observations with missing revenue as this is our dependent variable. Keep other missings for now - if variables are included in a model, then any missing rows will be dropped. But don't want to drop now in case those rows have useful information about other variables.

```
movie <- filter(movie, !is.na(gross))
```

Exclude non-US movies. Tried to include these and apply a country-specific deflator to make real and then use exchange rates to get US dollars. However, after manually checking some IMDB pages, we realized that some movies that are made in foreign countries still report US Dollars and some don't. The unit isn't recorded, so there is no way to check this.
Thus, we are excluding non-US movies to avoid misleading relationships between variables.

```
movie <- movie %>% filter(country == "USA")
```

Tidy genre column. Create one column per genre such that each is a dummy variable. One movie can be multiple genres. This is tidy data, but for visualizations we will likely have to gather the various genre columns and make the data untidy to use genre as an aesthetic.

```
# use mtabulate from qdapTools package to create dummies for each genre
genre_dummies <- mtabulate(strsplit(movie$genres, '\\|'))
# cbind with original data
movie <- cbind(movie, genre_dummies)
# make a tibble again
```

```
movie <- as.tibble(movie)
# rename Film-Noir and sci-Fi columns. The dashes make some later manipulation hard
movie <- movie %>%
  rename(FilmNoir = 'Film-Noir', SciFi = 'Sci-Fi')
```

Recode some movie content ratings.

X replaced by NC-17 in 1990, M replaced by PG.
Source: https://www.cnn.com/2014/08/06/showbiz/sixties-movie-ratings/index.html

Not rated and Unrated are effectively the same as missing

Approved and Passed: Only found definition for Approved, but Passed around the same time period, and thus assuming the same thing. 13 movies, all pre 1965. Replacing as missing "Approved - Pre-1968 titles only (from the MPAA site) Under the Hays Code, films were simply approved or disapproved based on whether they were deemed 'moral' or 'immoral'.")" Source: https://help.imdb.com/article/contribution/titles/certificates/GU757M8ZJ9ZPXB39?ref_=helpart_nav_27#

```
movie <- movie %>%
  mutate(content_rating = ifelse(content_rating == "M", "PG", content_rating),
         content_rating = ifelse(content_rating == "X", "NC-17", content_rating),
         content_rating = ifelse(content_rating == 'Not Rated' | content_rating == 'Unrated' |
                                   content_rating == 'Approved' | content_rating == 'Passed',
                                 NA, content_rating))
```

Deflator Data Cleaning

Basic cleaning: rename columns and create date variable

```
deflator <- deflator %>%
  # rename columns
  rename(date = DATE, gdpd = GDPDEF) %>%
  # convert date column into a datetime
  mutate(date = mdy(date))
```

Data is quarterly, but movie data is annual. Take average across quarters to get an annual deflator.

```
deflator <- deflator %>%
  # get year variable
  mutate(year = year(date)) %>%
  # mean across quarters
  group_by(year) %>%
  summarize(gdpd = mean(gdpd)) %>%
  # divide GDPD by 100
  # (calculated as real / nominal * 100, so need to divide by 100 to reverse the conversion)
  mutate(gdpd = gdpd / 100)
```

Oscar Data Cleaning

Rename all variables to be lower case

```
# rename variables to lower case
oscar <- oscar %>% rename_all(tolower)
```

Clean the year variable: first few Oscars were two years (1930/1931 for example). Just take first year.

Year matters so that we can see how many Oscars they had won when each movie in the movie dataset was released. An actor earlier in their career with no Oscars would liekly have less of an impact on movie revenue

than that same actor later in their career after 3 Oscars.

Does not matter which film they won the award for.

```
oscar <- oscar %>%
  # get first year if year variable has a / aka has two years
  mutate(year = ifelse(str_detect(year, '/'), str_split(year, '/', simplify = T)[,1], year),
    # make numeric instead of character
    year = as.numeric(year))
```

Filter: only include winners and acting and directing awards. One person can be an actor and a director, so make sure to differentiate.
There are several types of acting and directing awards: make sure to get them all (but not art director etc.)

```
actors <- oscar %>%
  # winners only
  filter(winner == 1) %>%
  # acting and directing awards only
  filter(str_detect(award, 'Actor') | str_detect(award, 'Actress'))
directors <- oscar %>%
  # winners only
  filter(winner == 1) %>%
  # acting and directing awards only
  filter(str_detect(award, 'Directing'))
# check these are the awards we want
unique(actors$award)
```

```
## [1] "Actor"                         "Actress"
## [3] "Actor in a Supporting Role"    "Actress in a Supporting Role"
## [5] "Actor in a Leading Role"       "Actress in a Leading Role"
```

```
unique(directors$award)
```

```
## [1] "Directing (Comedy Picture)"    "Directing (Dramatic Picture)"
## [3] "Directing"
```

Cleaning issue: For the first few years, the dataset lists director name under 'name' for the directing awards, as is logical. However, it switches in 1930 such that 'name' is the film name and 'film' is the director name

```
directors <- directors %>%
  mutate(name = ifelse(year >= 1930, film, name))
```

Expand the actor and director datasets so have full time series for each Actor. Mark when in that time series they earned each Oscar. Then can cumulatively sum so know number of Oscars at each year.

```
# actors
actors <- actors %>%
  # drop unnecessary columns
  select(-c(ceremony, award, winner, film)) %>%
  # indicate that they won an Oscar in this year
  mutate(oscar = 1) %>%
  # expand dataset so one year per actor
  complete(year, name) %>%
  # fill in oscar dummy with zero when it is not 1
  mutate(oscar = ifelse(is.na(oscar), 0, oscar))
# directors
directors <- directors %>%
  # drop unnecessary columns
```

```
  select(-c(ceremony, award, winner, film)) %>%
  # indicate that they won an Oscar in this year
  mutate(oscar = 1) %>%
  # expand dataset so one year per actor
  complete(year, name) %>%
  # fill in oscar dummy with zero when it is not 1
  mutate(oscar = ifelse(is.na(oscar), 0, oscar))
```

Cumulatively sum number of Oscars per actor/actress and director

```
actors <- actors %>%
  group_by(name) %>%
  mutate(total_oscars = cumsum(oscar)) %>%
  select(-oscar)
directors <- directors %>%
  group_by(name) %>%
  mutate(total_oscars = cumsum(oscar)) %>%
  select(-oscar)
```

Merge Movie and Deflator Data by Year

```
# check if there are any non-matches
nonmatch <- movie %>% anti_join(deflator, by = 'year')
unique(nonmatch$year)
```

```
##  [1] 1940 1946 1939 1936 1937 1942 1935 1933 1929 1920
```

```
  # years before 1947, which is when the deflator data starts.
nonmatch %>% count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1    13
```

```
  # 13 observations. OK to drop.
# merge using an inner join. Want to drop any observations in movie that do not have a year in deflator
movie <- movie %>% inner_join(deflator, by = 'year')
```

Convert nominal US dollars to real dollars

Use the deflator. Real = nominal / GDPD

```
movie <- movie %>%
  mutate(real_budget = budget / gdpd,
         real_gross = gross / gdpd)
```

Merge Movie and Actor Oscar Data

Merge 4 times: once for each of the 3 actor columns in the movie dataset and once for director. Get the total number of oscars for that person at the time of the movie release.
Left join so that we keep the rows with actors and directors that do not have Oscars

```
movie <- movie %>%
  # actor merge
  # rename because will merge in more variables with identical name
```

```
  left_join(actors, by = c('actor_1_name' = 'name', 'year')) %>%
  rename(total_oscars_actor1 = total_oscars) %>%
  left_join(actors, by = c('actor_2_name' = 'name', 'year')) %>%
  rename(total_oscars_actor2 = total_oscars) %>%
  left_join(actors, by = c('actor_3_name' = 'name', 'year'), suffix = c('', '_actor3')) %>%
  rename(total_oscars_actor3 = total_oscars) %>%
  # director merge
  left_join(directors, by = c('director_name' = 'name', 'year'), suffix = c('', '_director')) %>%
  rename(total_oscars_director = total_oscars) %>%
  # replace missings for these 4 new variables with zero. Otherwise messes up math manipulations
  mutate_at(vars(total_oscars_actor1, total_oscars_actor2, total_oscars_actor3, total_oscars_director),
            funs(ifelse(is.na(.), 0, .)))
```

Add together the number of oscars that the actors/actresses in that movie have earned. Only one director so that is all set.

In the report, need to acknowledge that this is not at all a good measure of actor popularity. Many popular actors and actresses (and directors) never win Oscars. But an interesting measure none-the-less. Is the Academy in tune with what regular people want to see and will pay for?

```
movie <- movie %>%
  mutate(total_oscars_actor = total_oscars_actor1 + total_oscars_actor2 + total_oscars_actor3)
```

There are 2 movies with cumulative # actor Oscars == 4. 6 movies with 3.
There are 25 movies with cumulative # actor Oscars == 2.

Make factors and recode categories: actors: 0, 1, 2 or more. Directors: 0, 1 or more.
These numbers make sense too. If two major actors with Oscars. Usually only have one director, so having one Oscar holding director is good.

```
movie <- movie %>%
  mutate_at(vars(total_oscars_actor, total_oscars_director), funs(as.factor(.))) %>%
  # recode
  mutate(total_oscars_actor = fct_collapse(total_oscars_actor,
                                           '0' = '0', '1' = '1',
                                           '2 or more' = c('2', '3', '4')),
         total_oscars_director = fct_collapse(total_oscars_director,
                                              '0' = '0',
                                              '1 or more' = c('1', '2')))
```

Outliers?

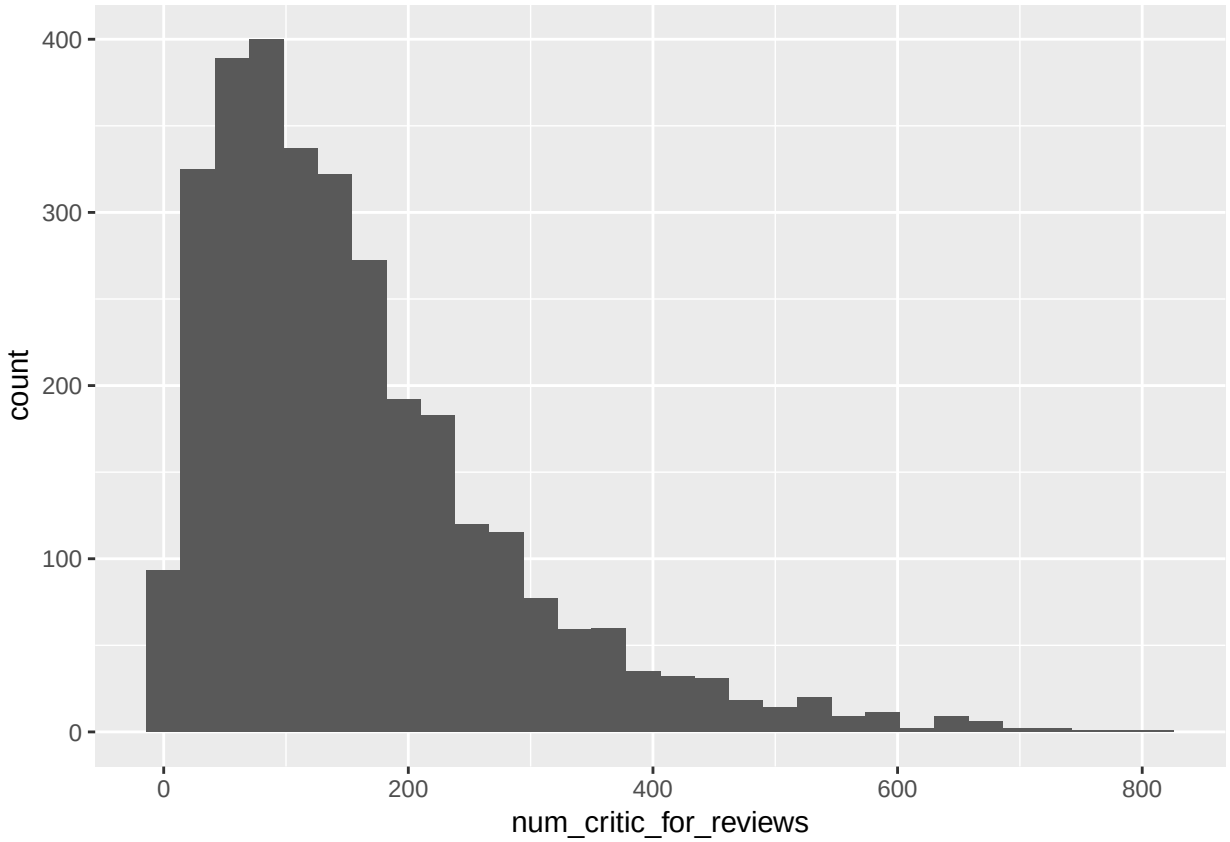
Basic distribution graphs for each variable: major outliers?

```
# numeric: histograms
num_cols <- movie[unlist(lapply(movie, is.numeric))]
lapply(names(num_cols), function(var) {
  ggplot(movie, aes_string(x = var)) + geom_histogram()
})
```

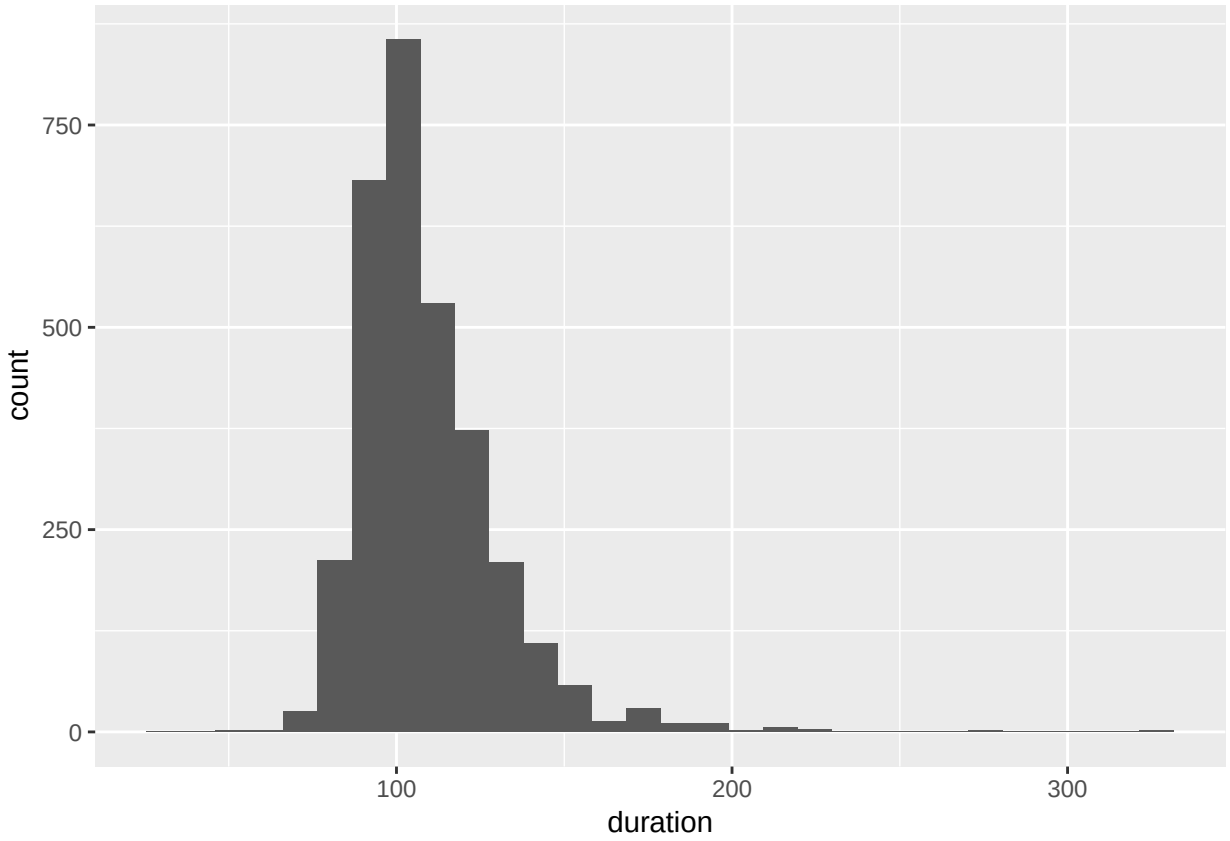
```
## [[1]]
```

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

```
## Warning: Removed 3 rows containing non-finite values (stat_bin).
```

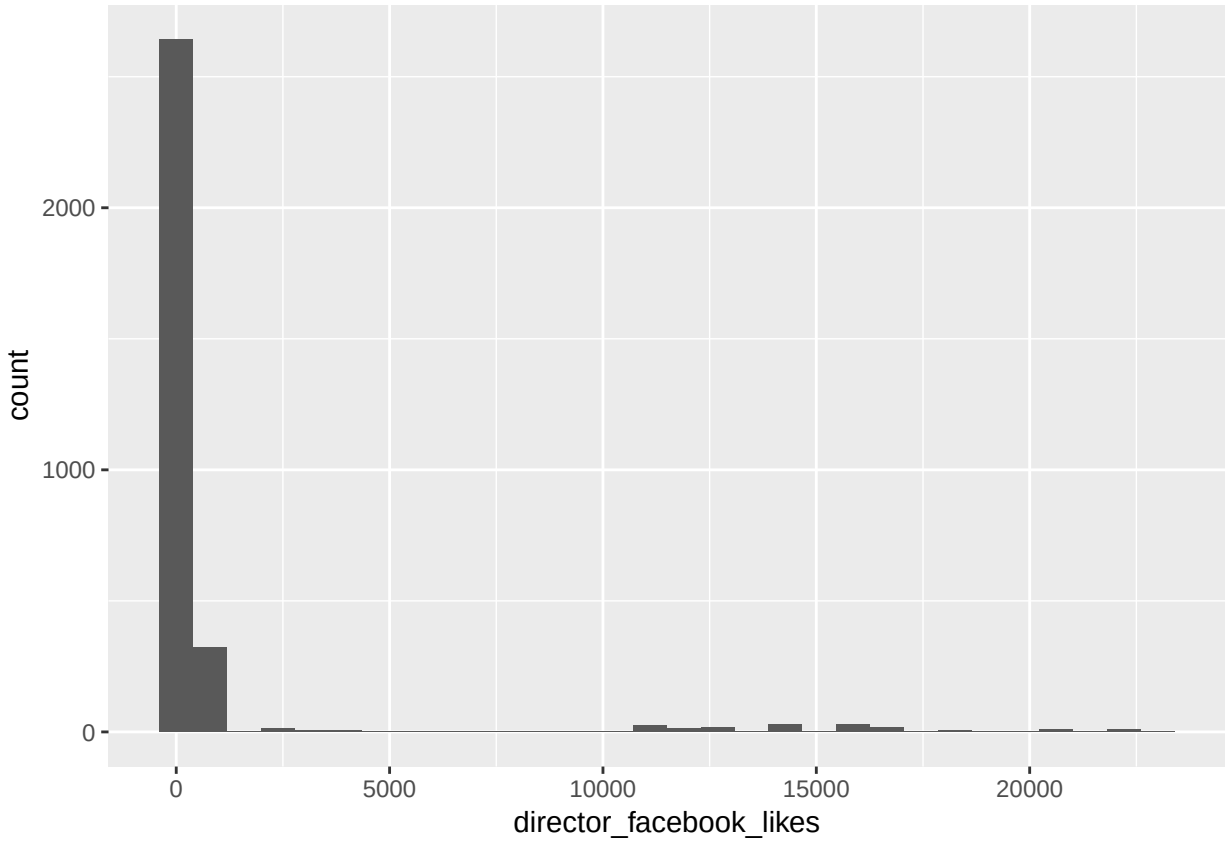



```
##
## [[2]]
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## Warning: Removed 1 rows containing non-finite values (stat_bin).
```




```
##
## [[3]]
```

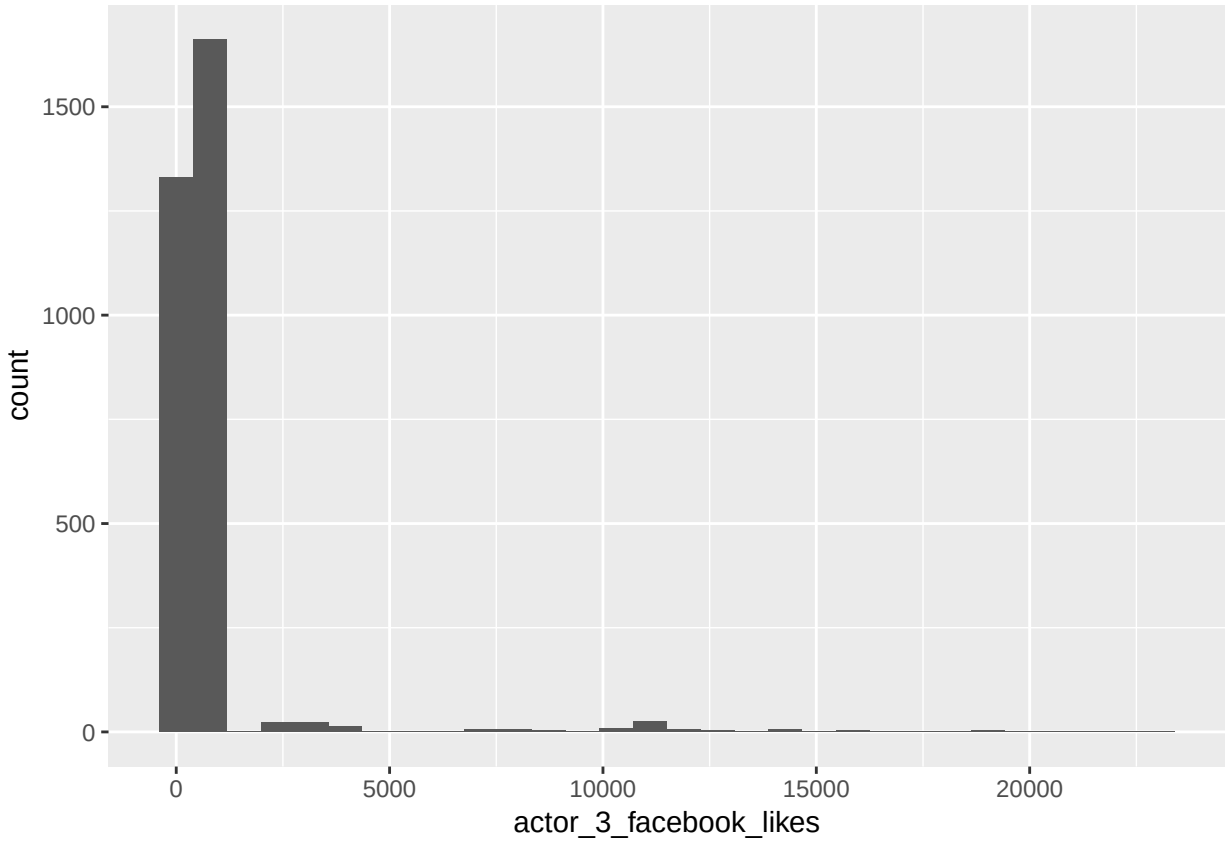
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
##
## [[4]]
```

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

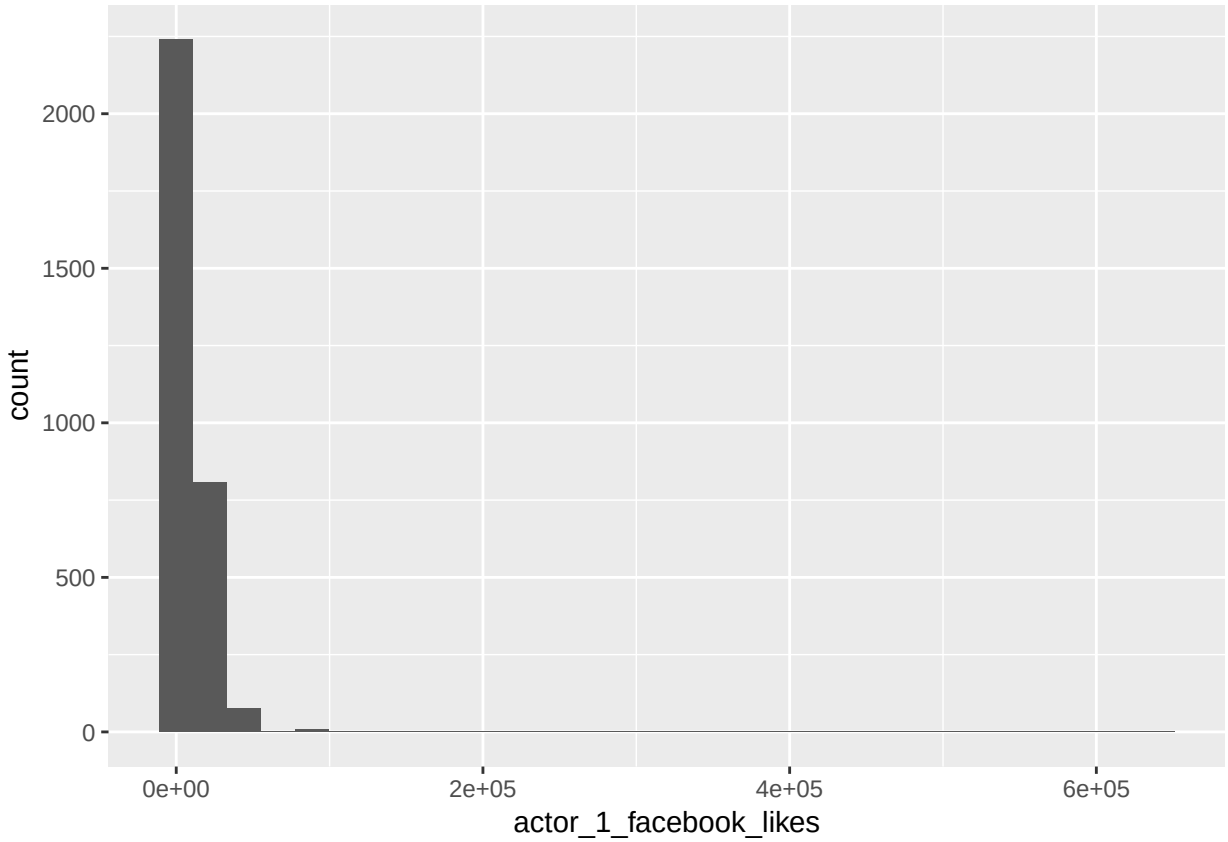
```
## Warning: Removed 6 rows containing non-finite values (stat_bin).
```




```
##
## [[5]]
```

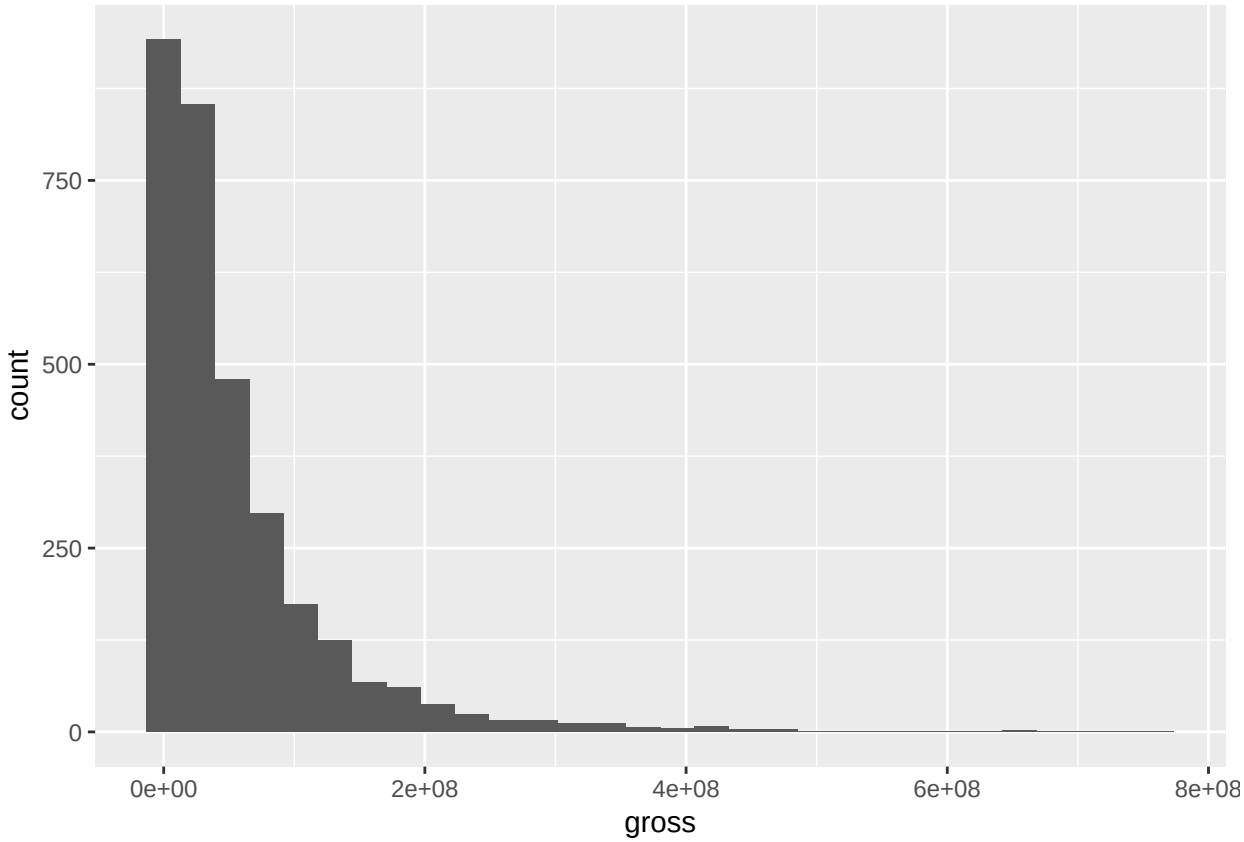
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 2 rows containing non-finite values (stat_bin).
```



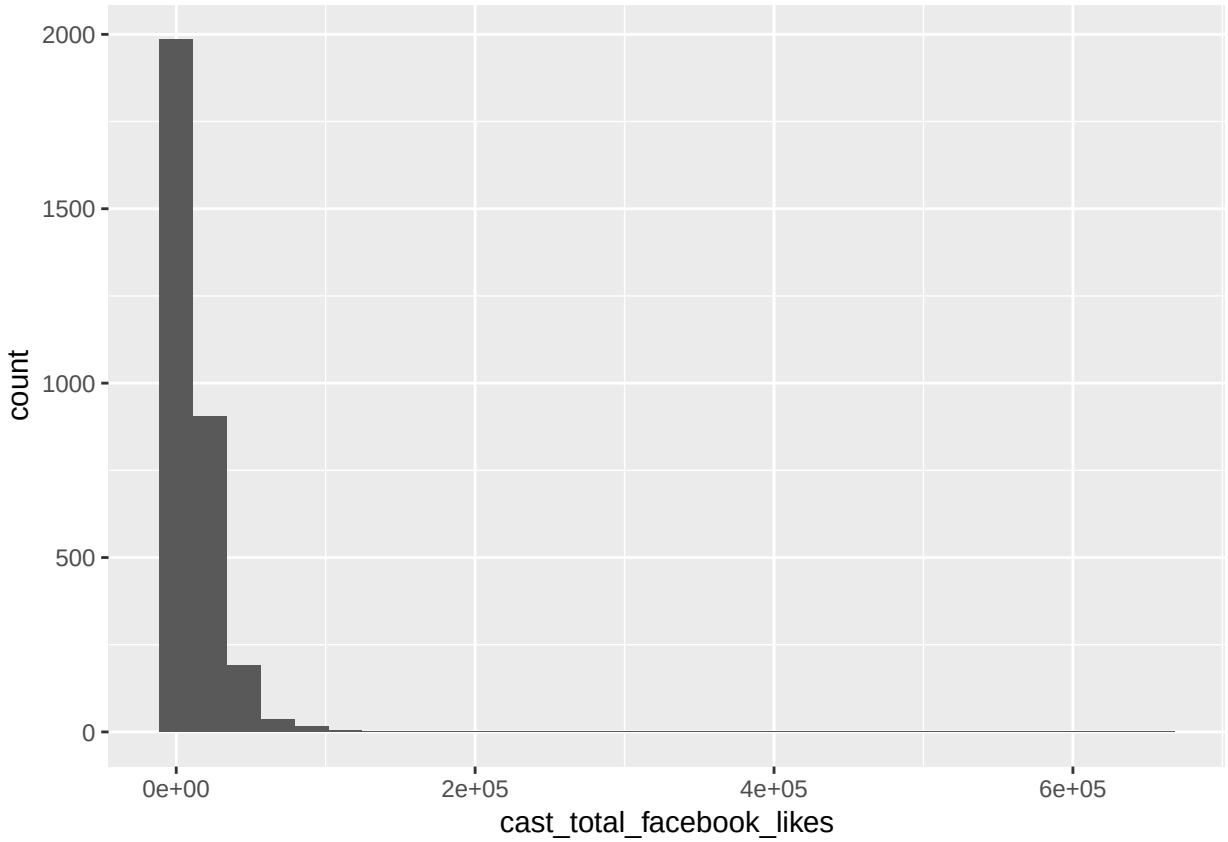

```
##
## [[6]]
```

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




```
##
## [[7]]
```

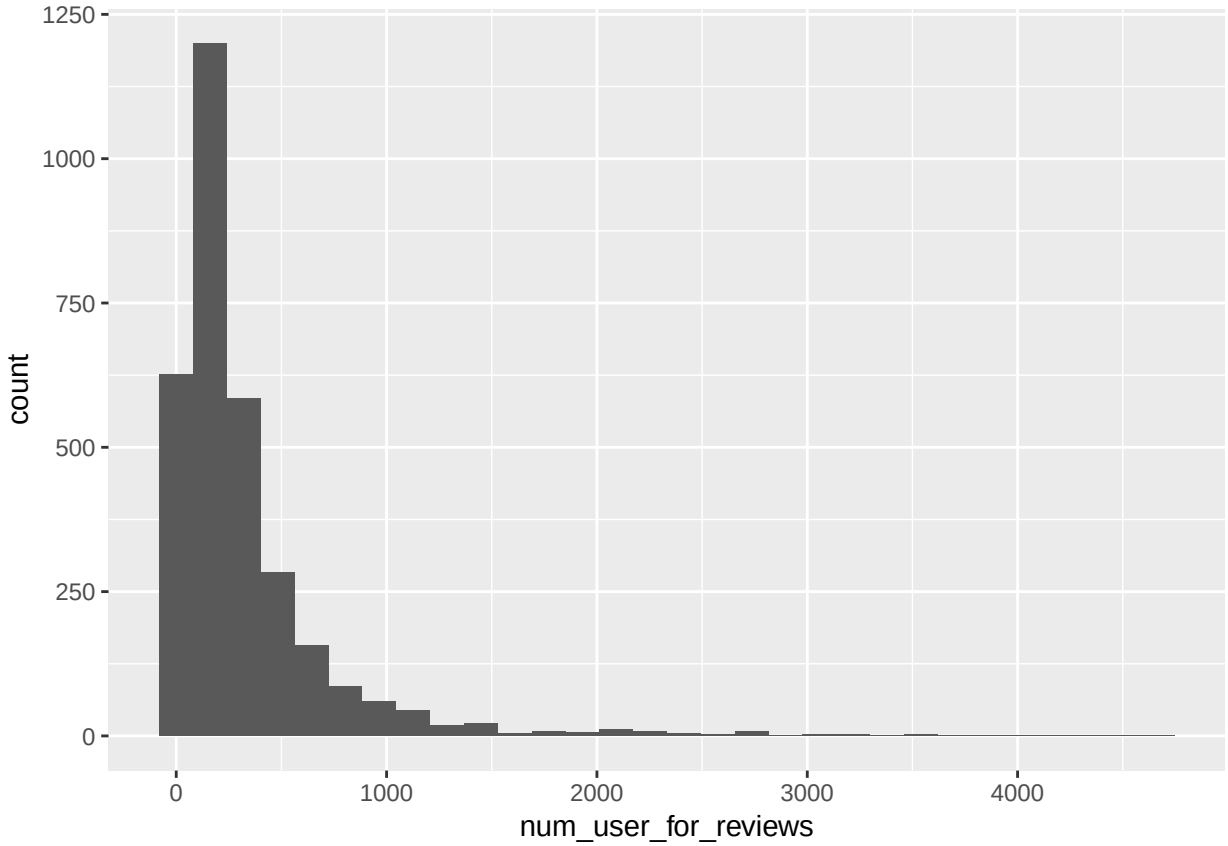
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



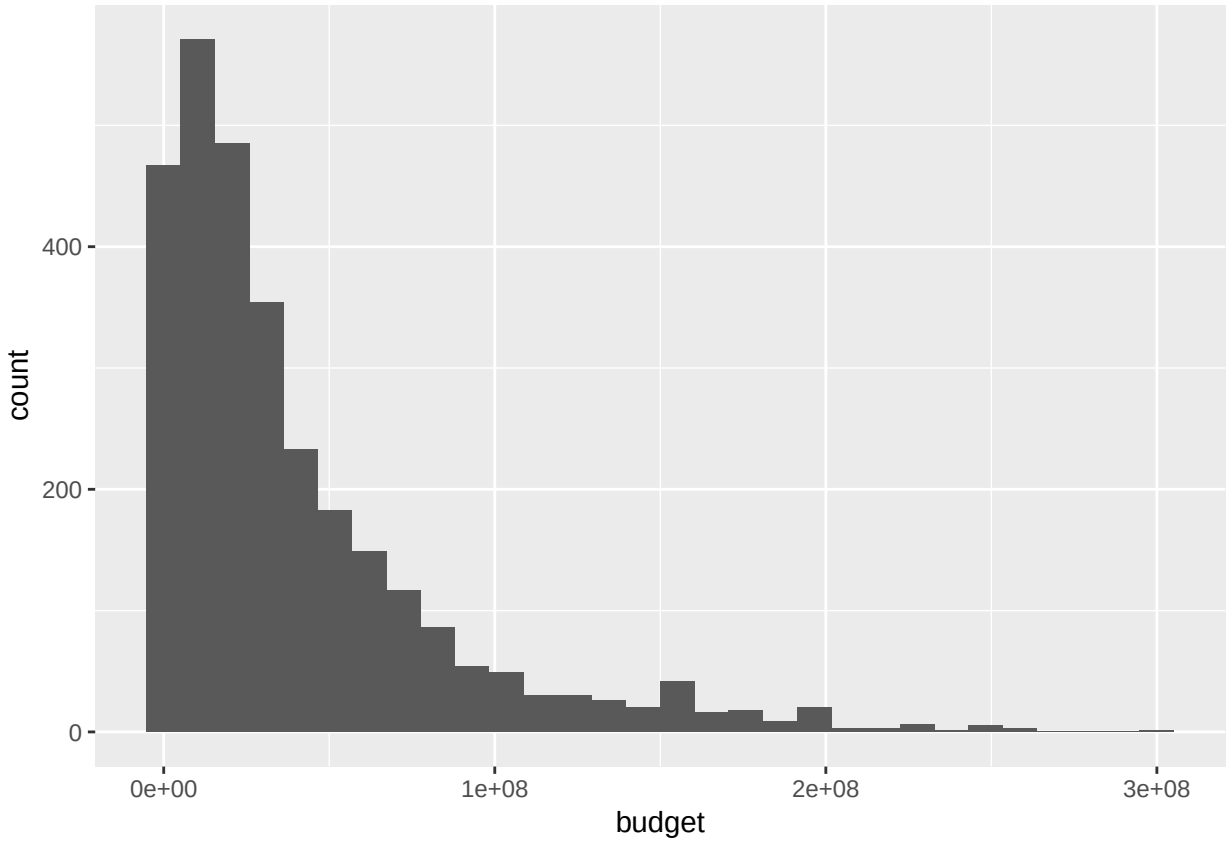

```
##
## [[8]]
```

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

```
## Warning: Removed 1 rows containing non-finite values (stat_bin).
```

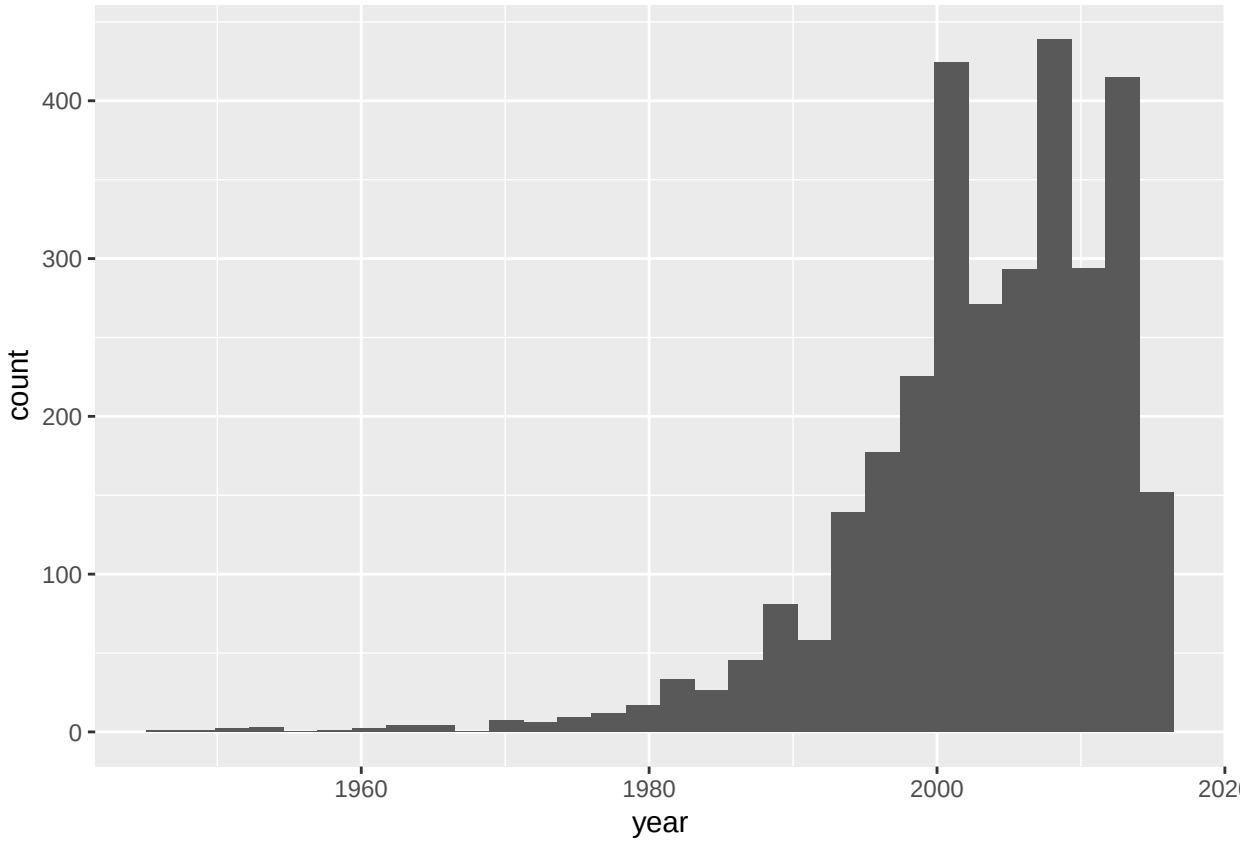



```
##
## [[9]]
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## Warning: Removed 160 rows containing non-finite values (stat_bin).
```

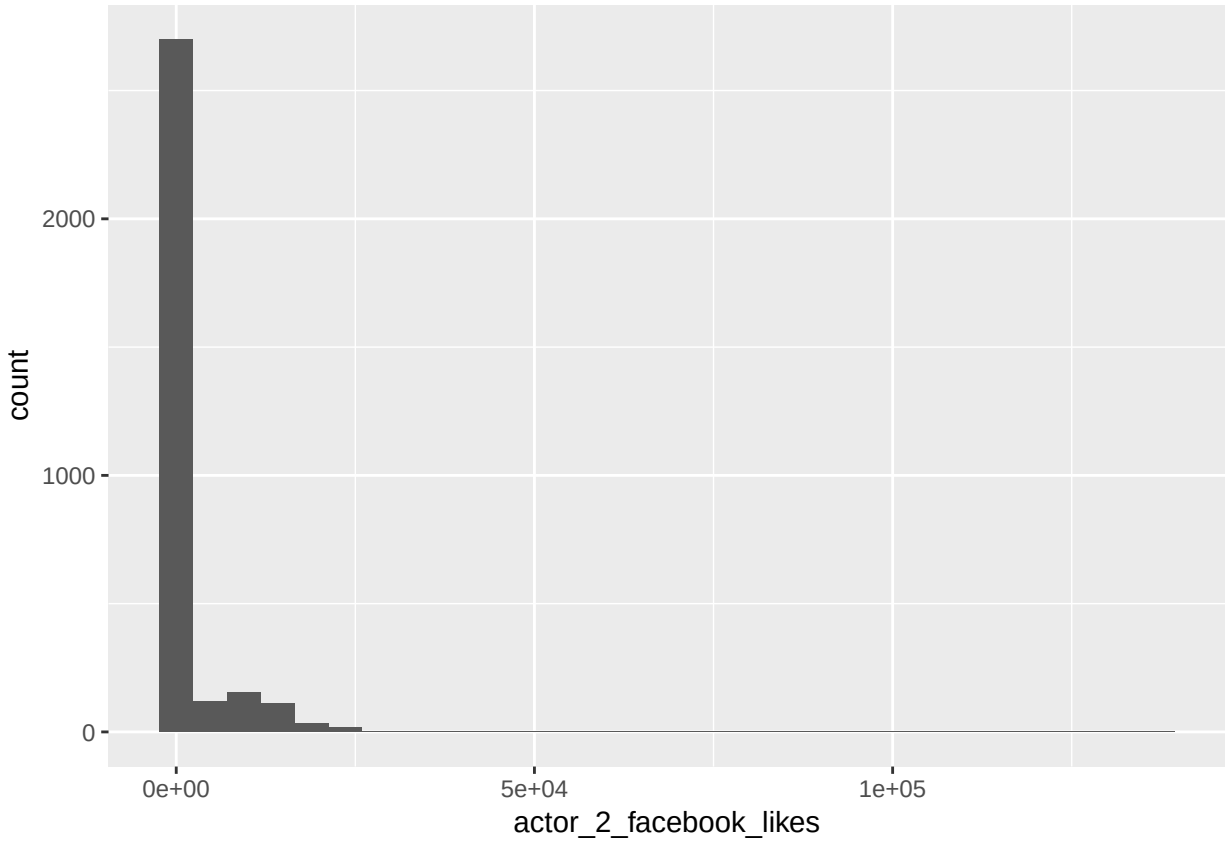



```
##
## [[10]]
```

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

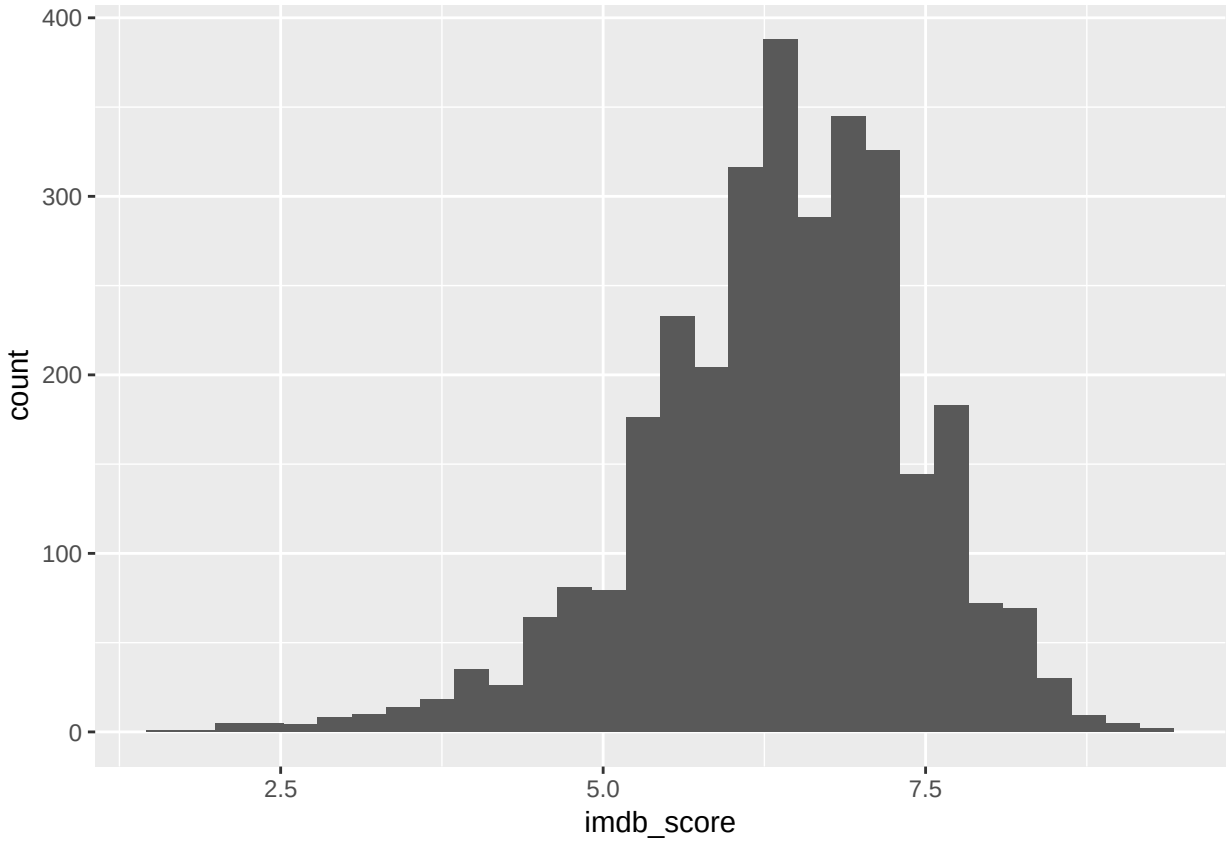



```
##
## [[11]]
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## Warning: Removed 3 rows containing non-finite values (stat_bin).
```



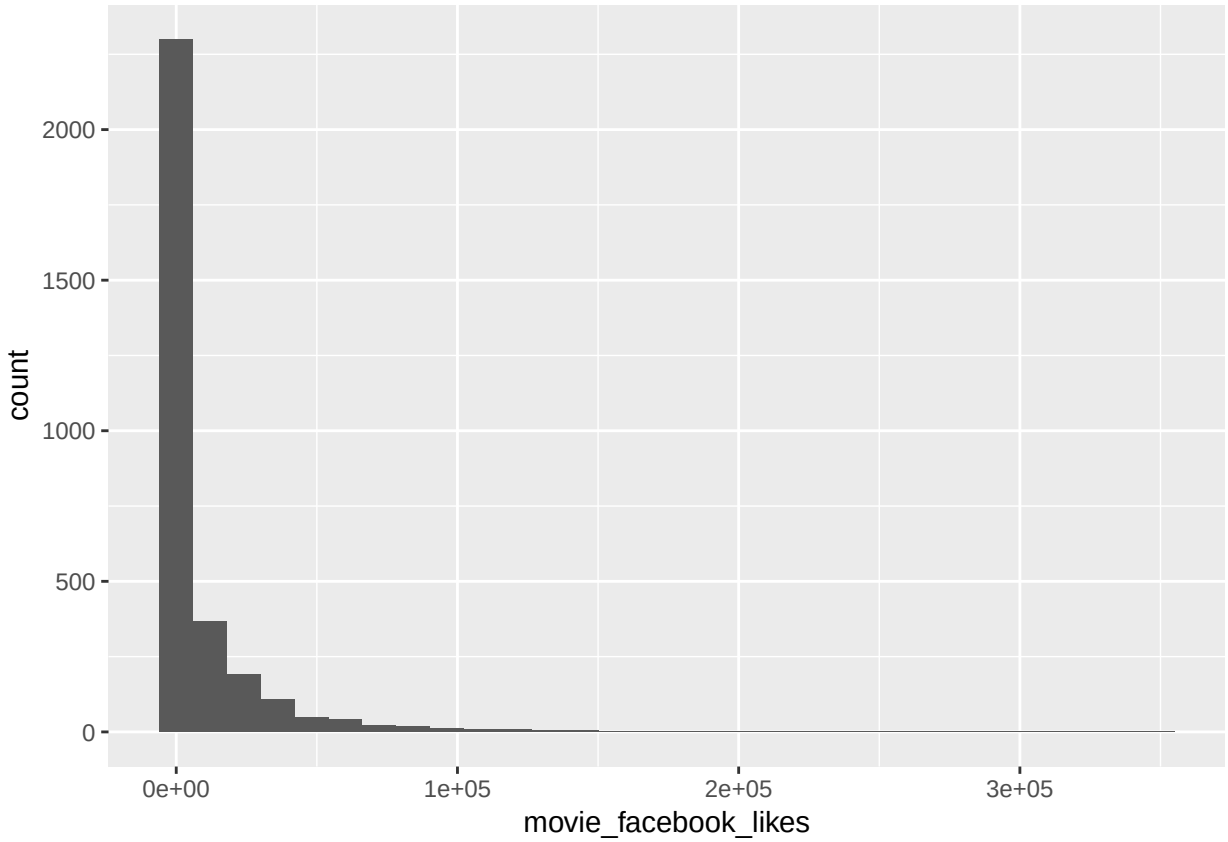

```
##
## [[12]]
```

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



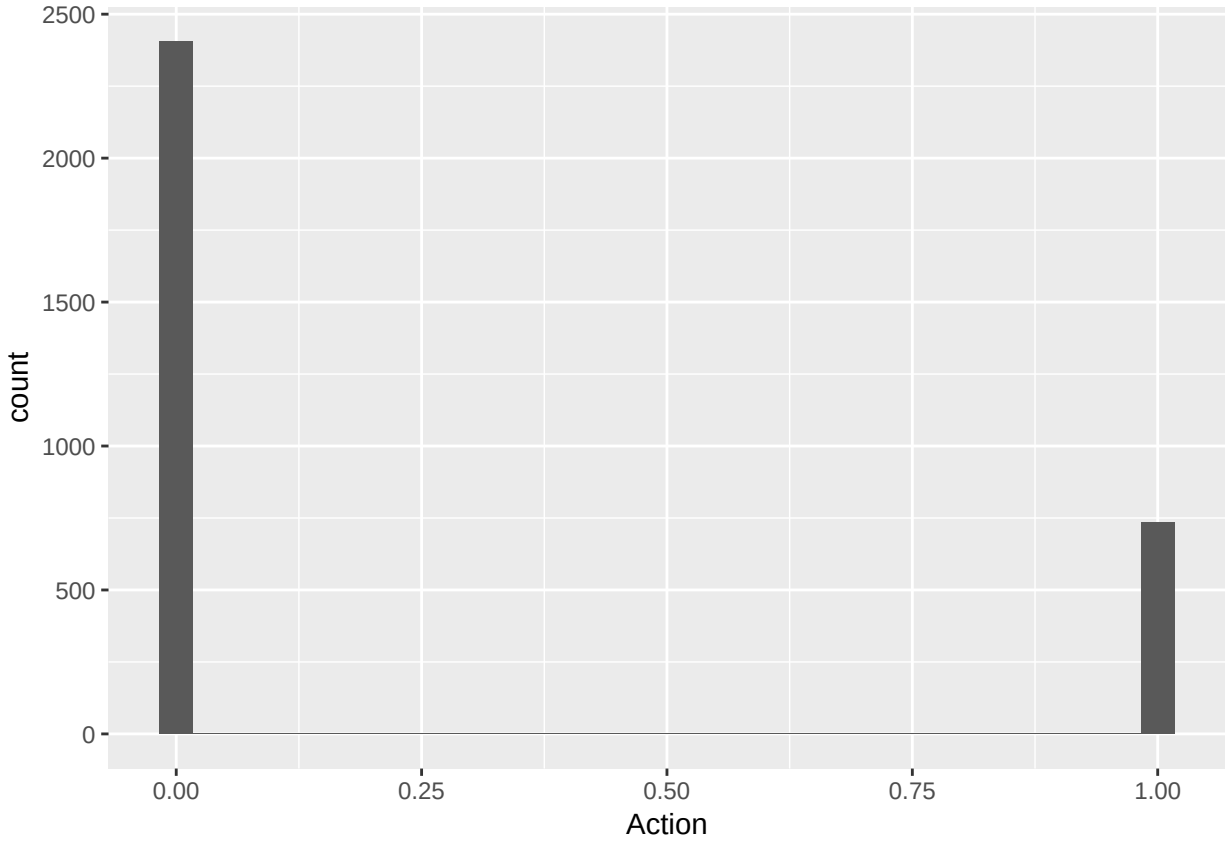

```
##
## [[13]]
```

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



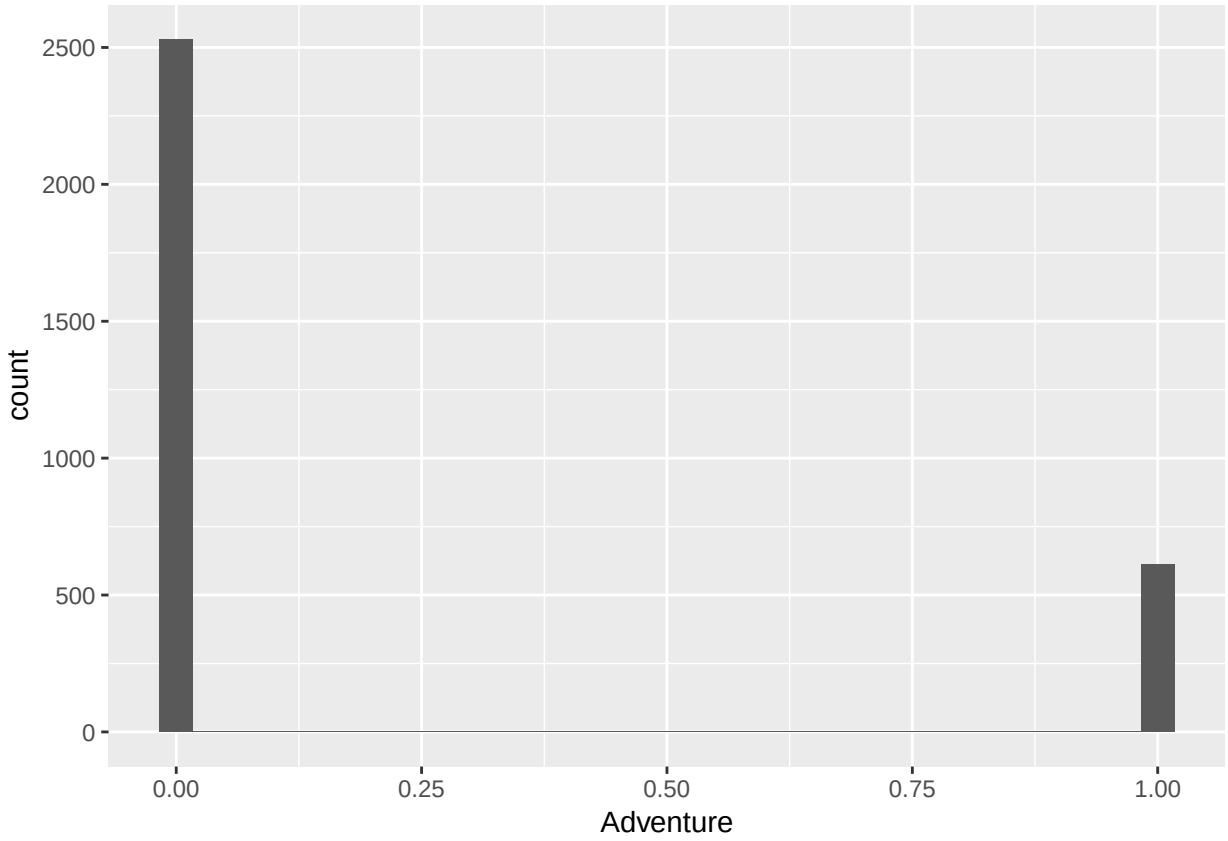

```
##
## [[14]]
```

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



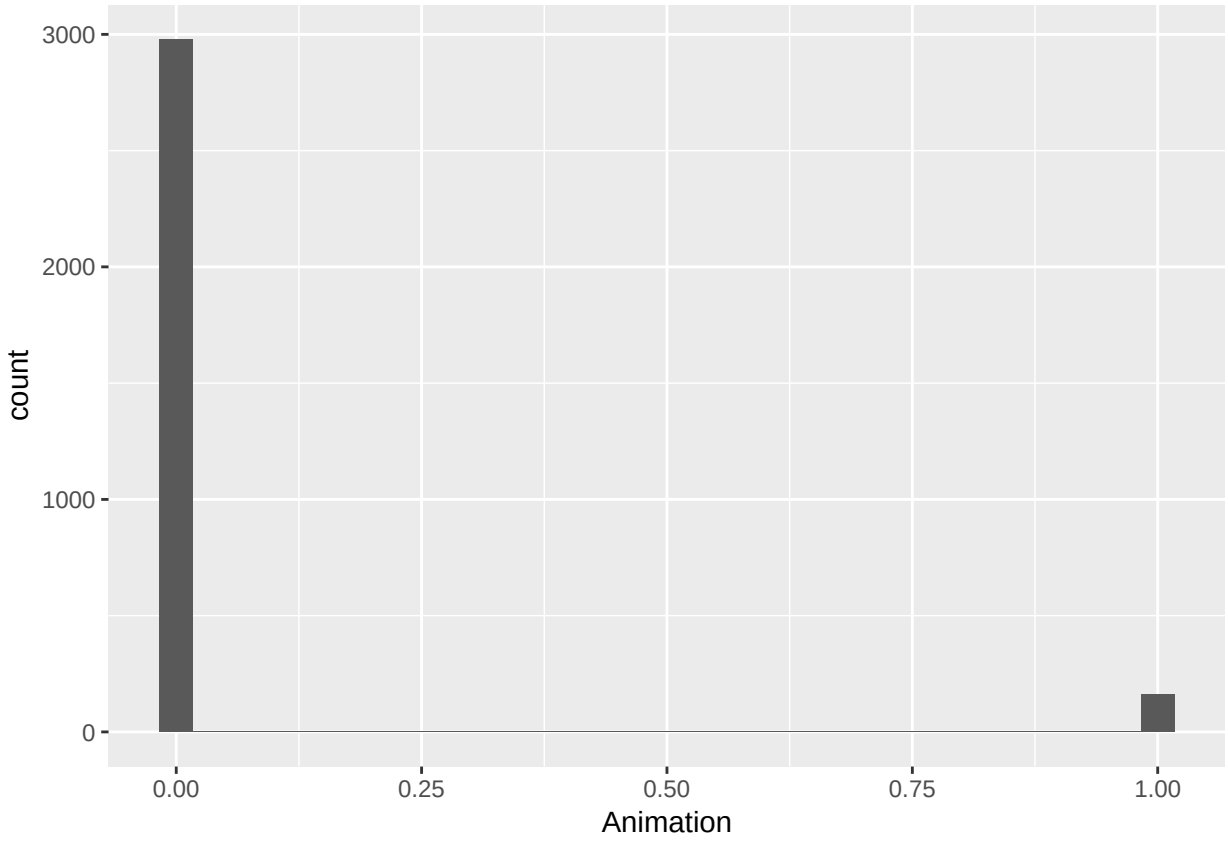

```
##
## [[15]]
```

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



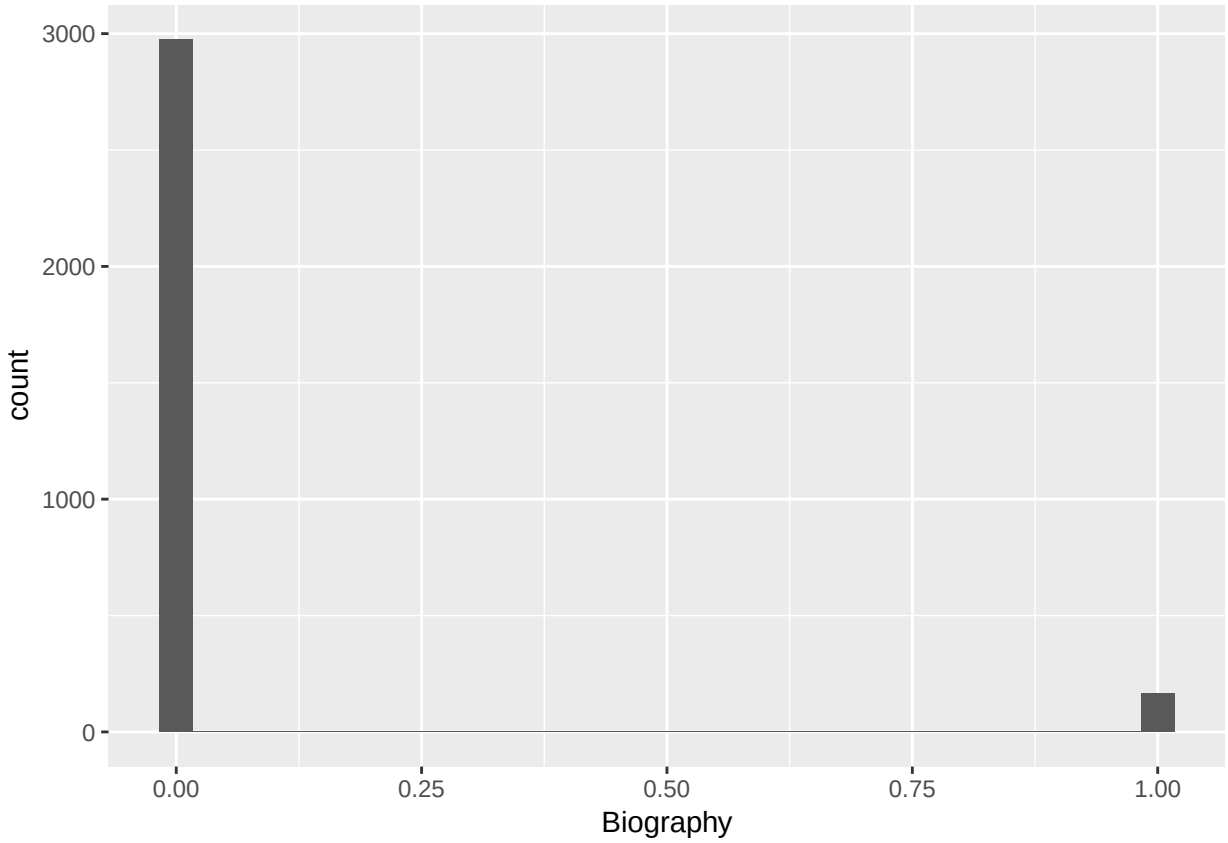

```
##
## [[16]]
```

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



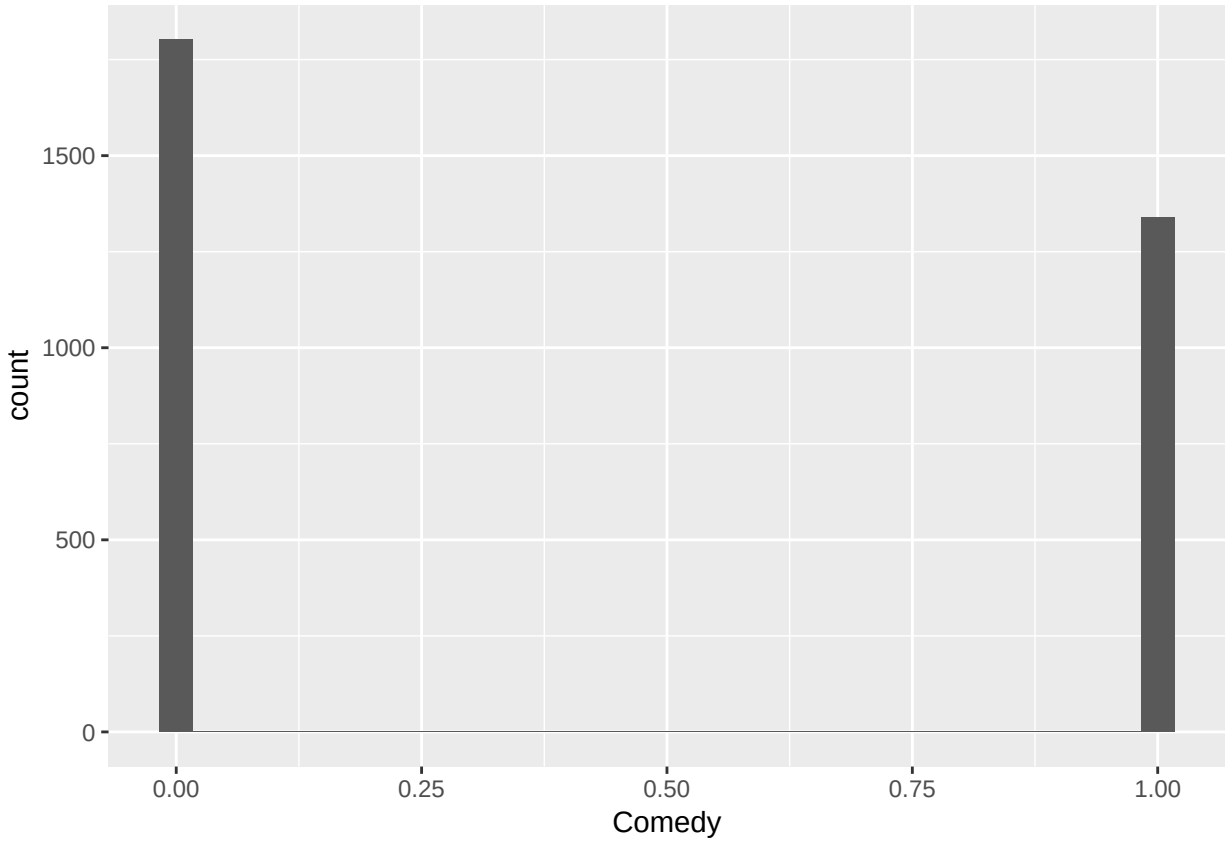

```
##
## [[17]]
```

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



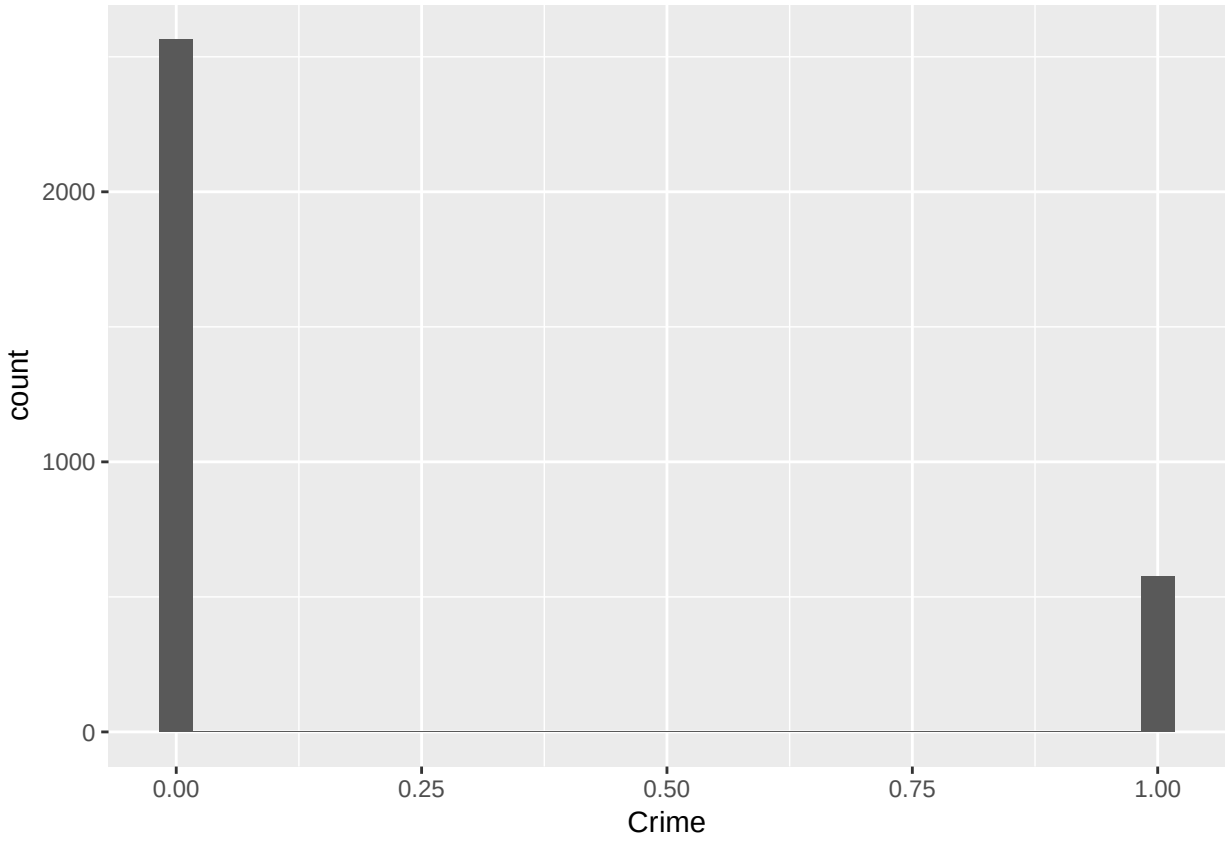

```
##
## [[18]]
```

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



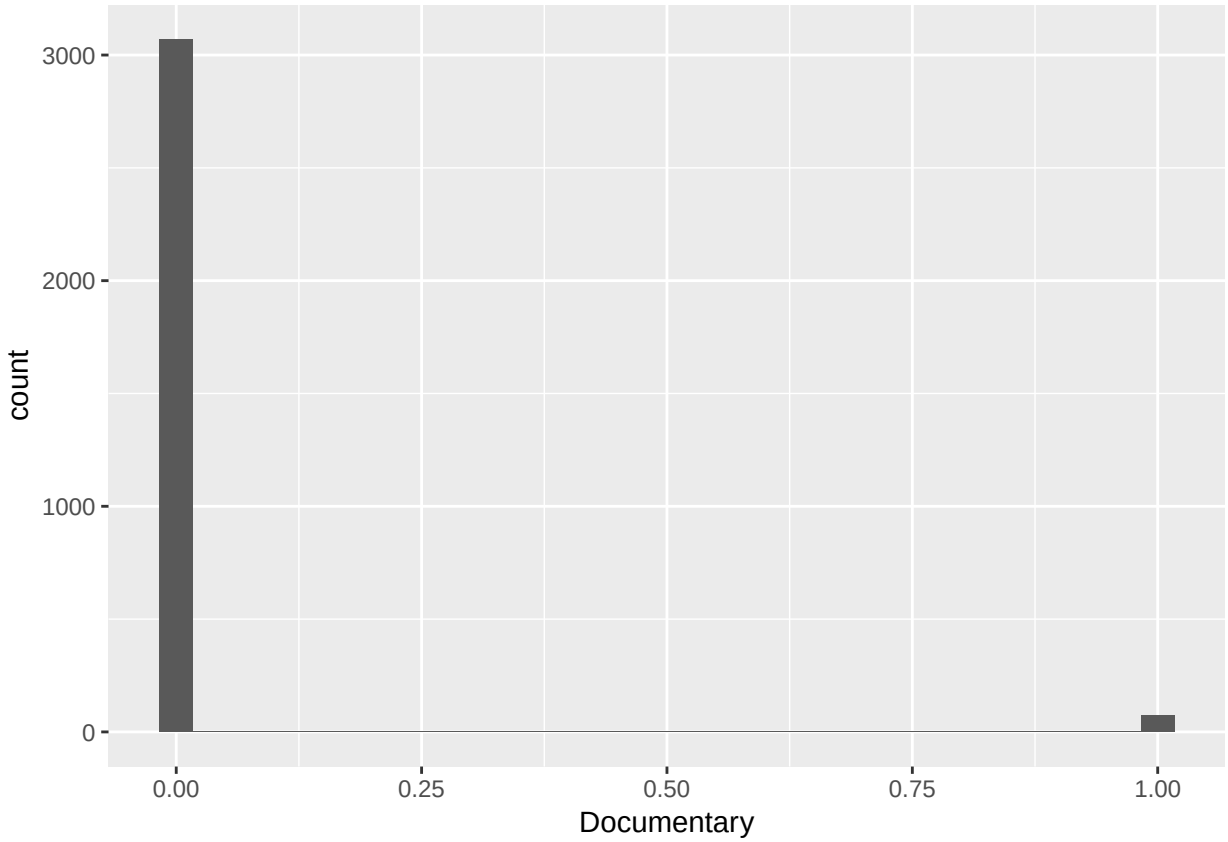

```
##
## [[19]]
```

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



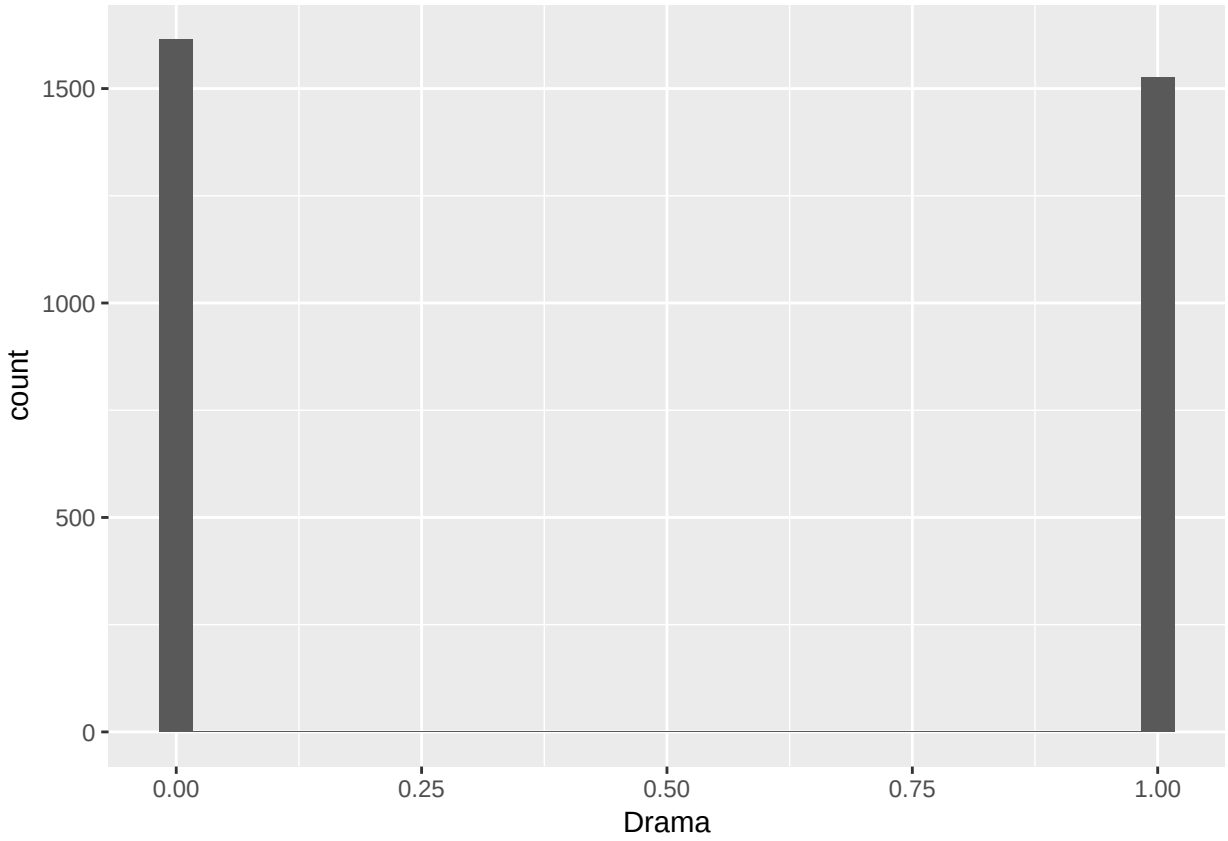

```
##
## [[20]]
```

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



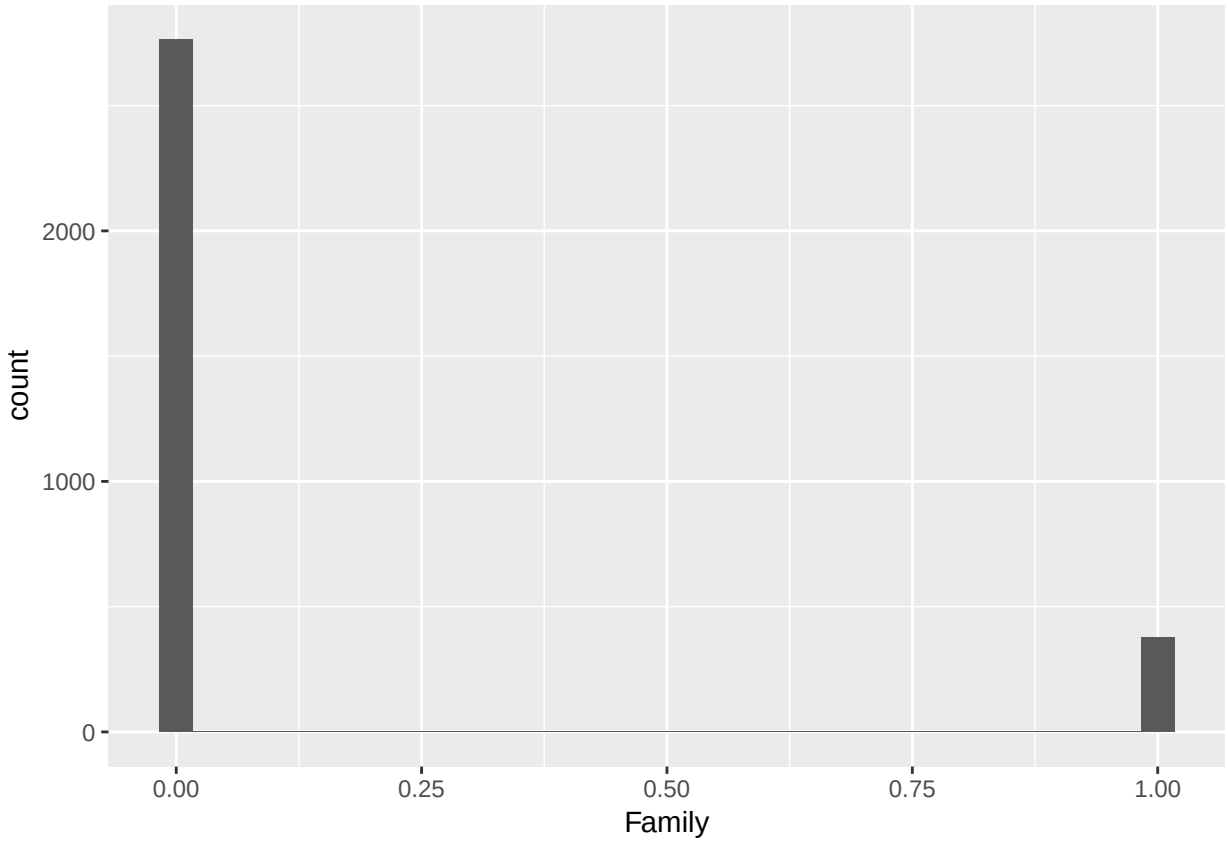

```
##
## [[21]]
```

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



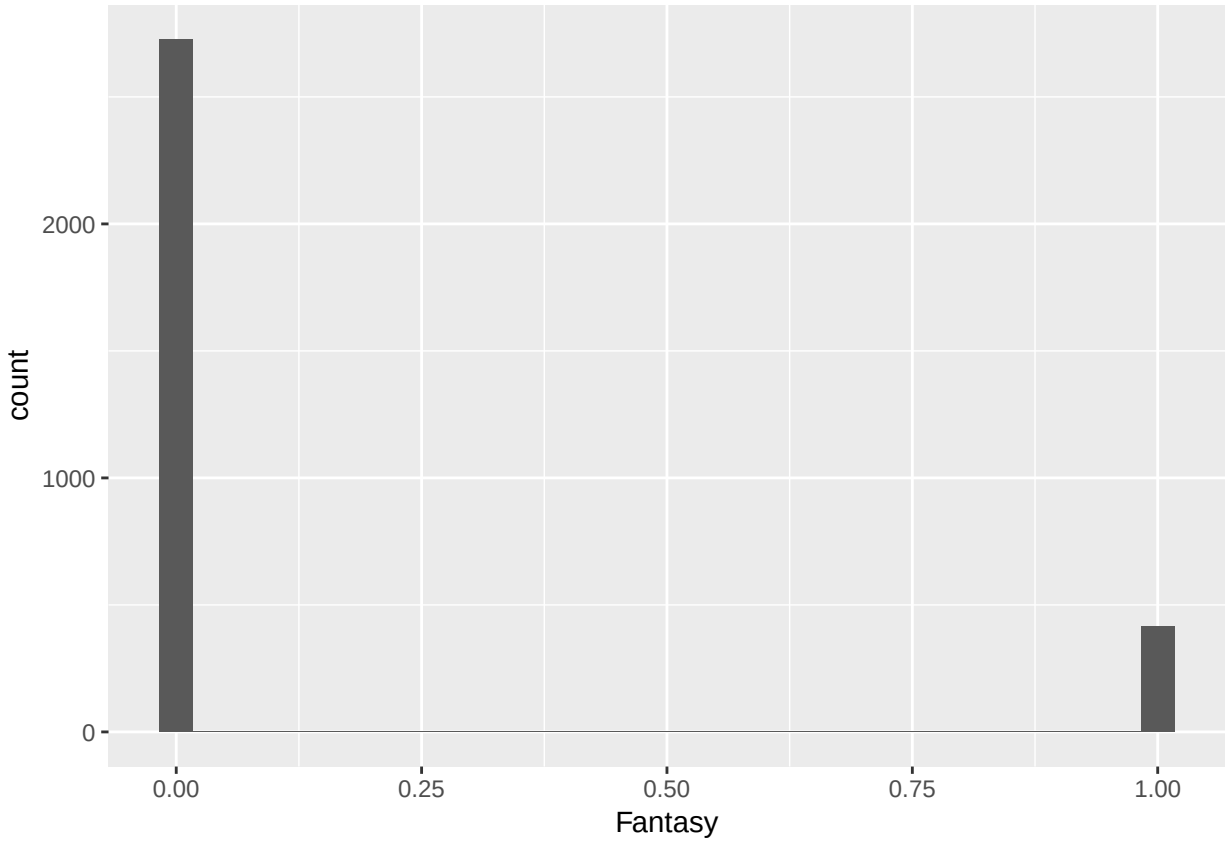

```
##
## [[22]]
```

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



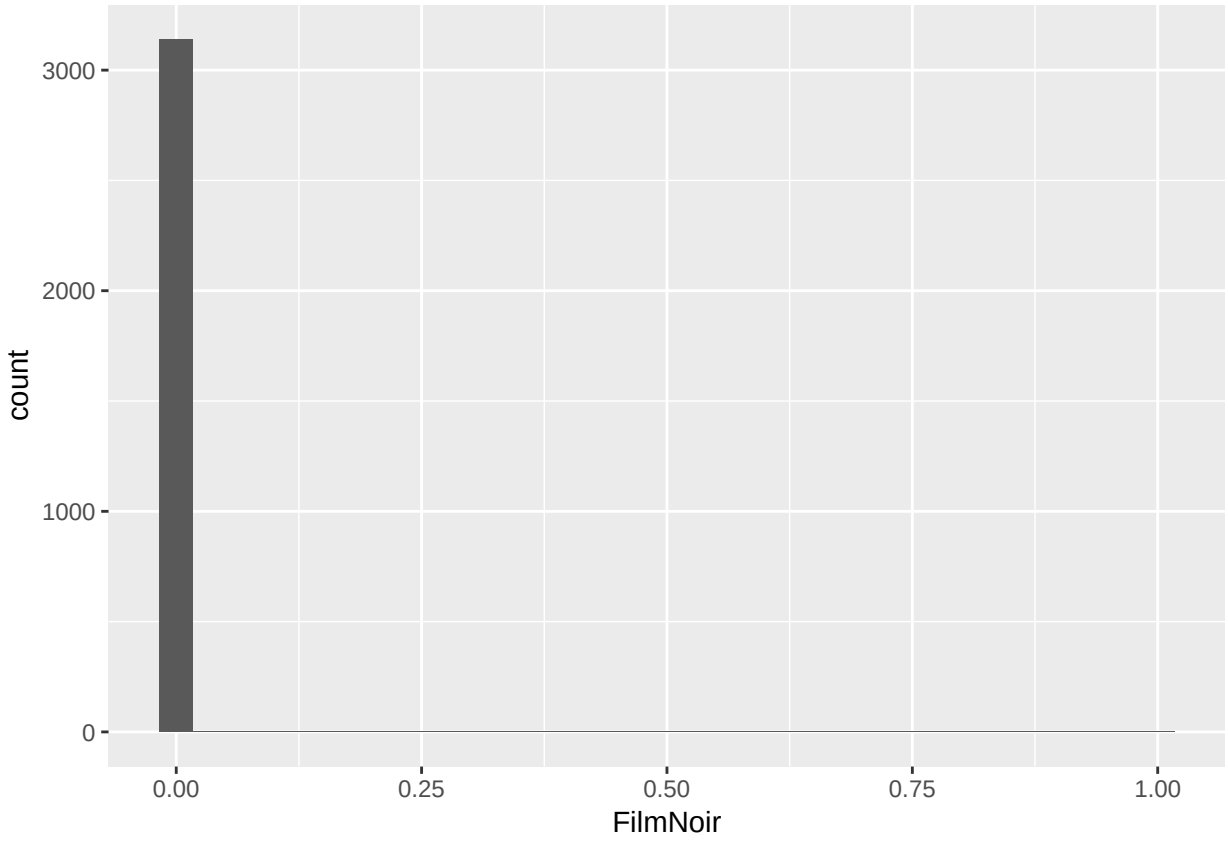

```
##
## [[23]]
```

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



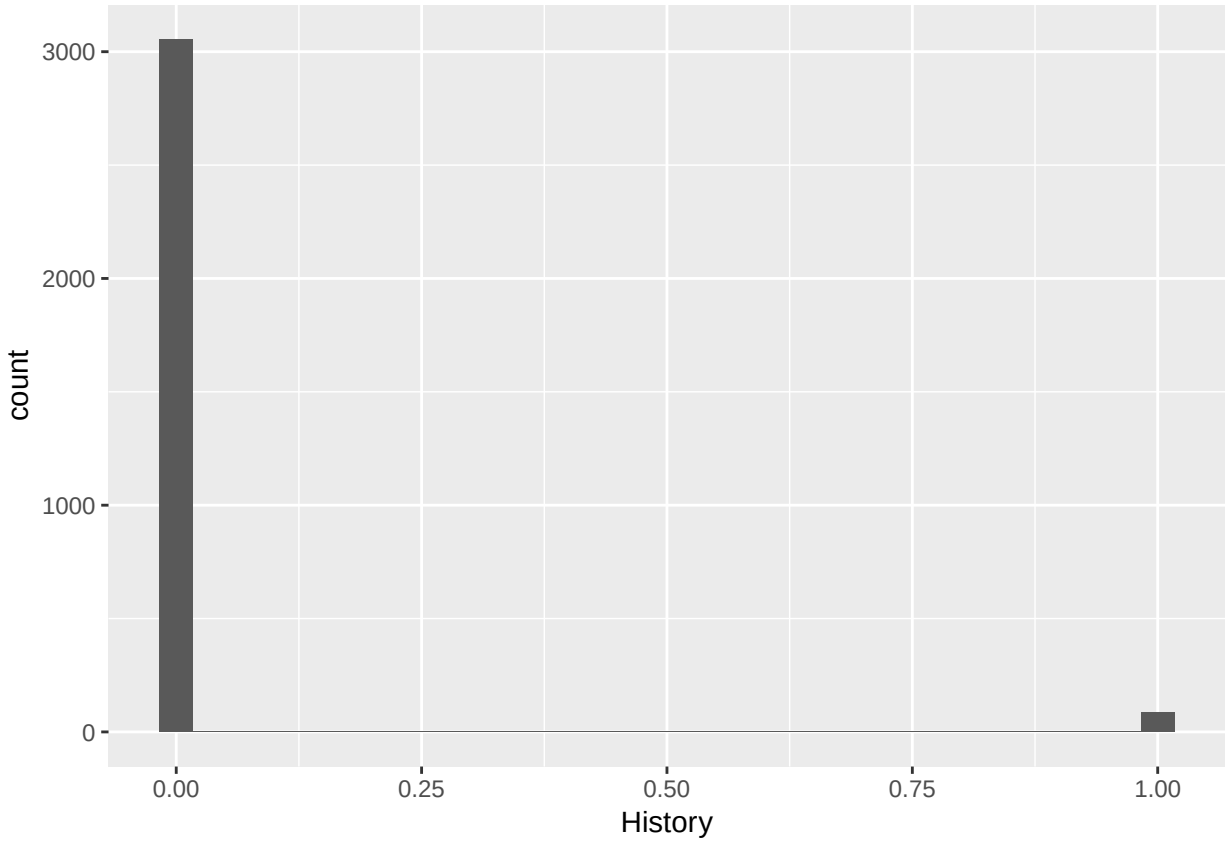

```
##
## [[24]]
```

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



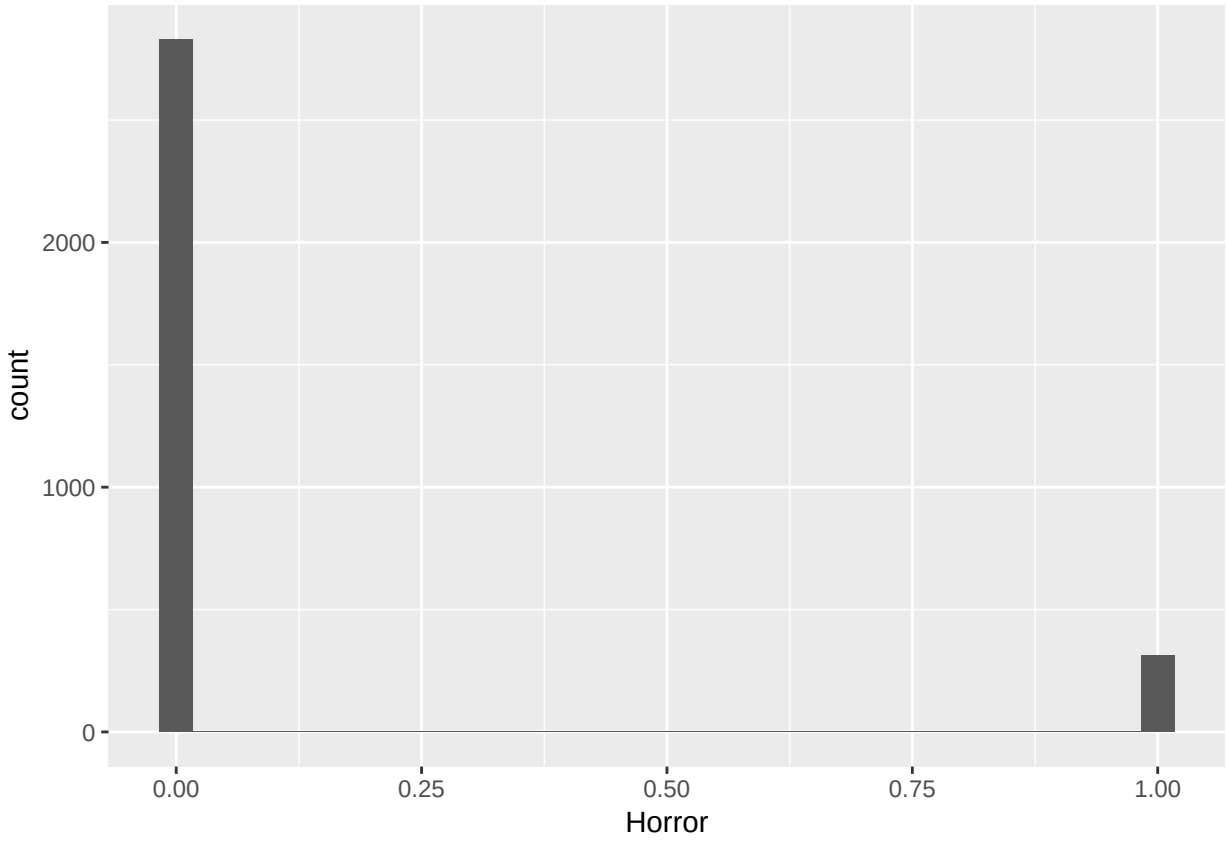

```
##
## [[25]]
```

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



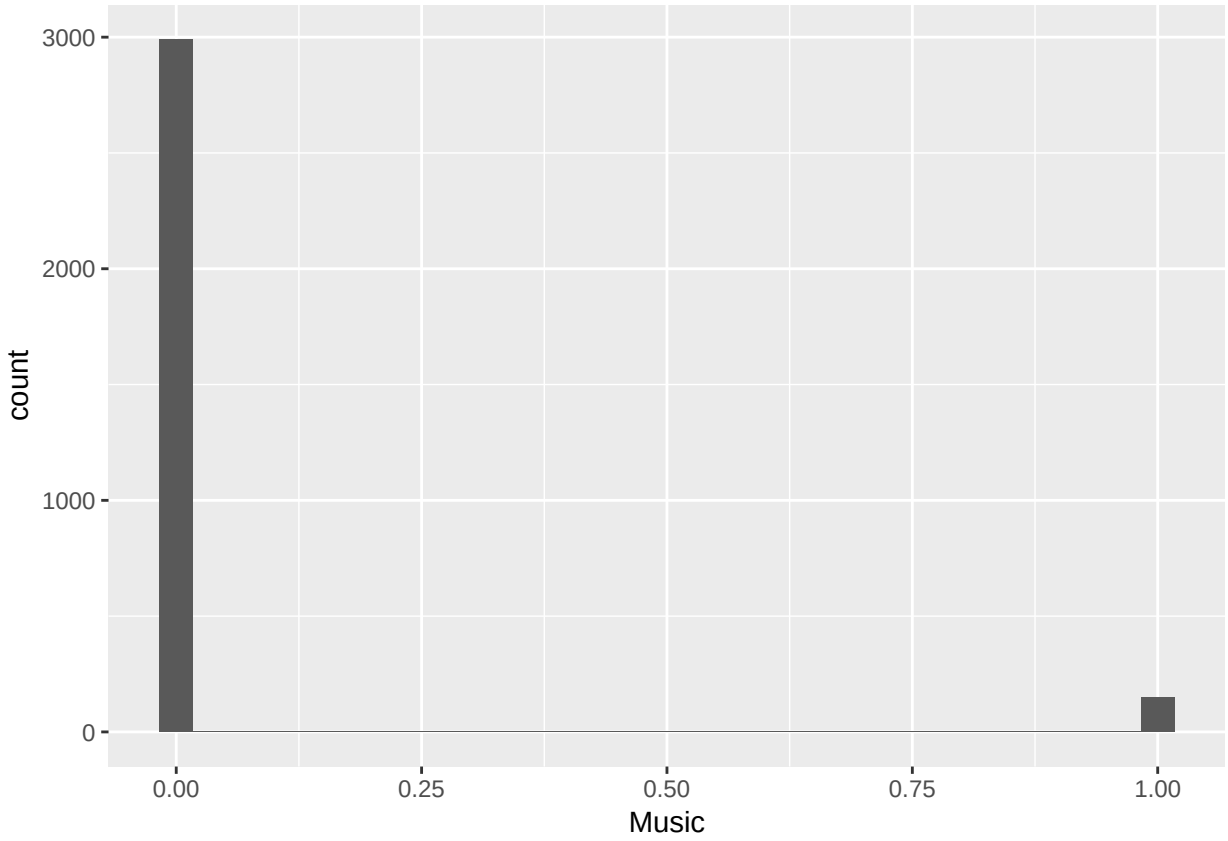

```
##
## [[26]]
```

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



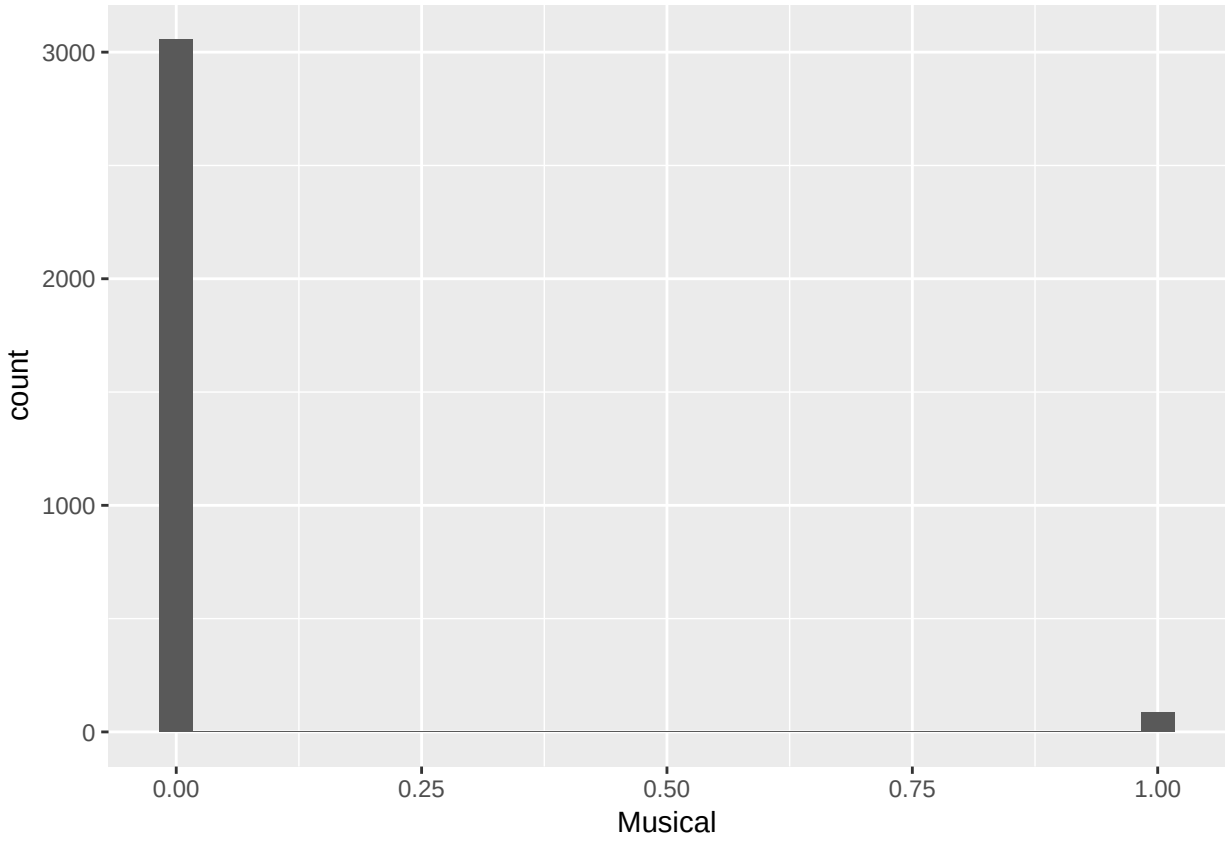

```
##
## [[27]]
```

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



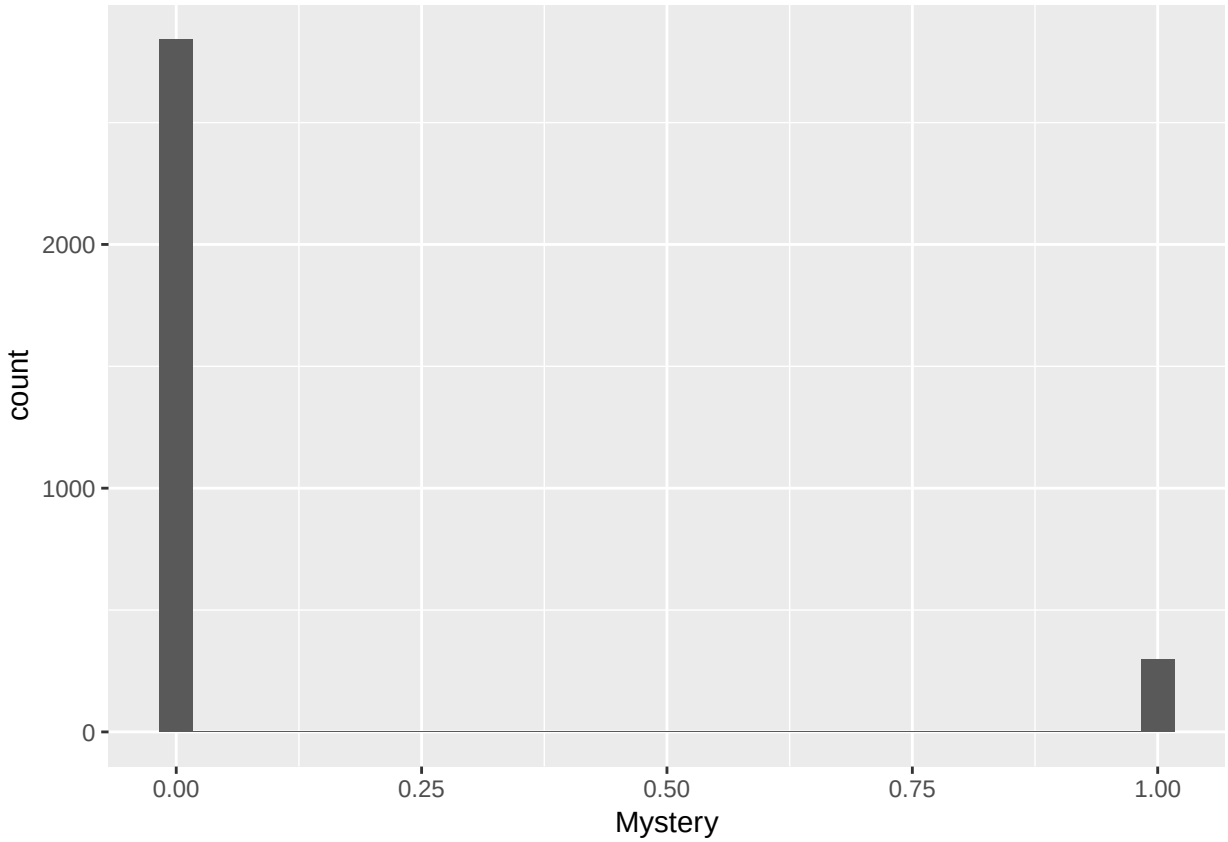

```
##
## [[28]]
```

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



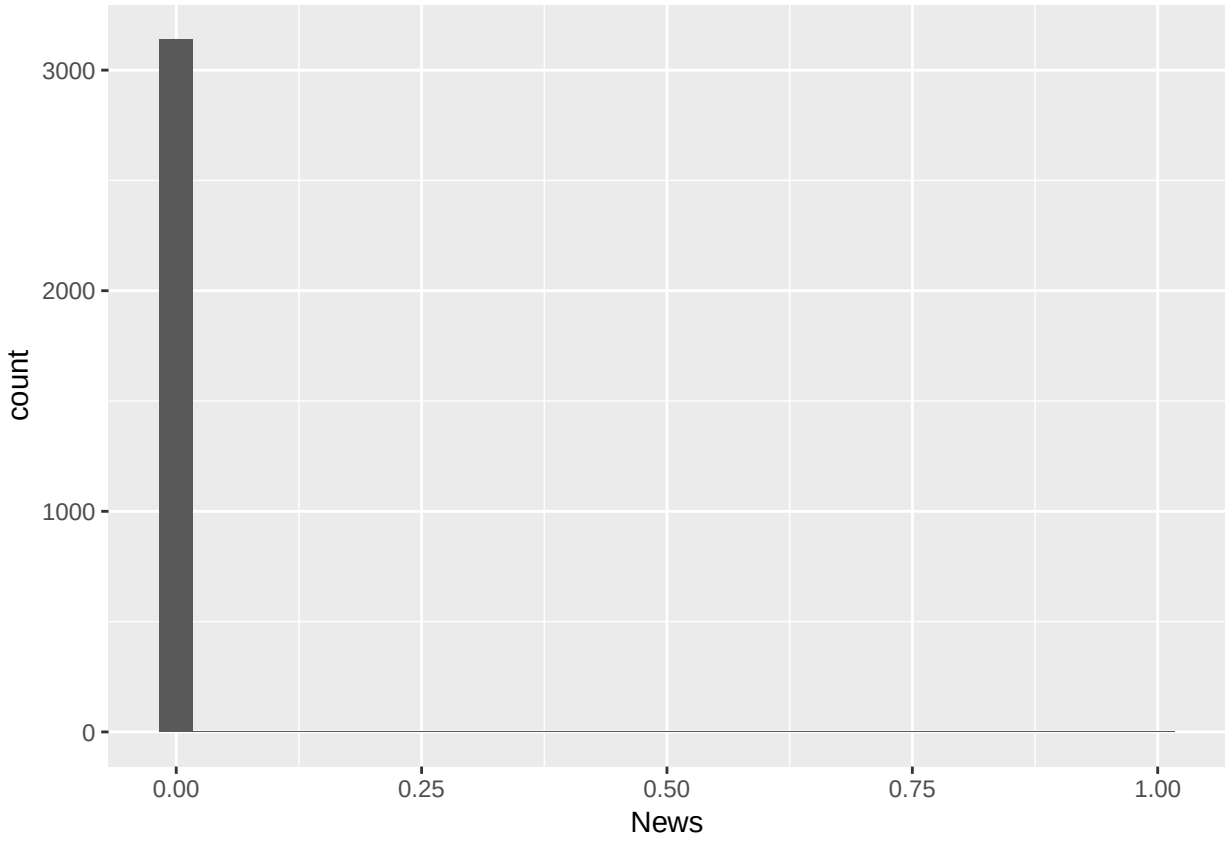

```
##
## [[29]]
```

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



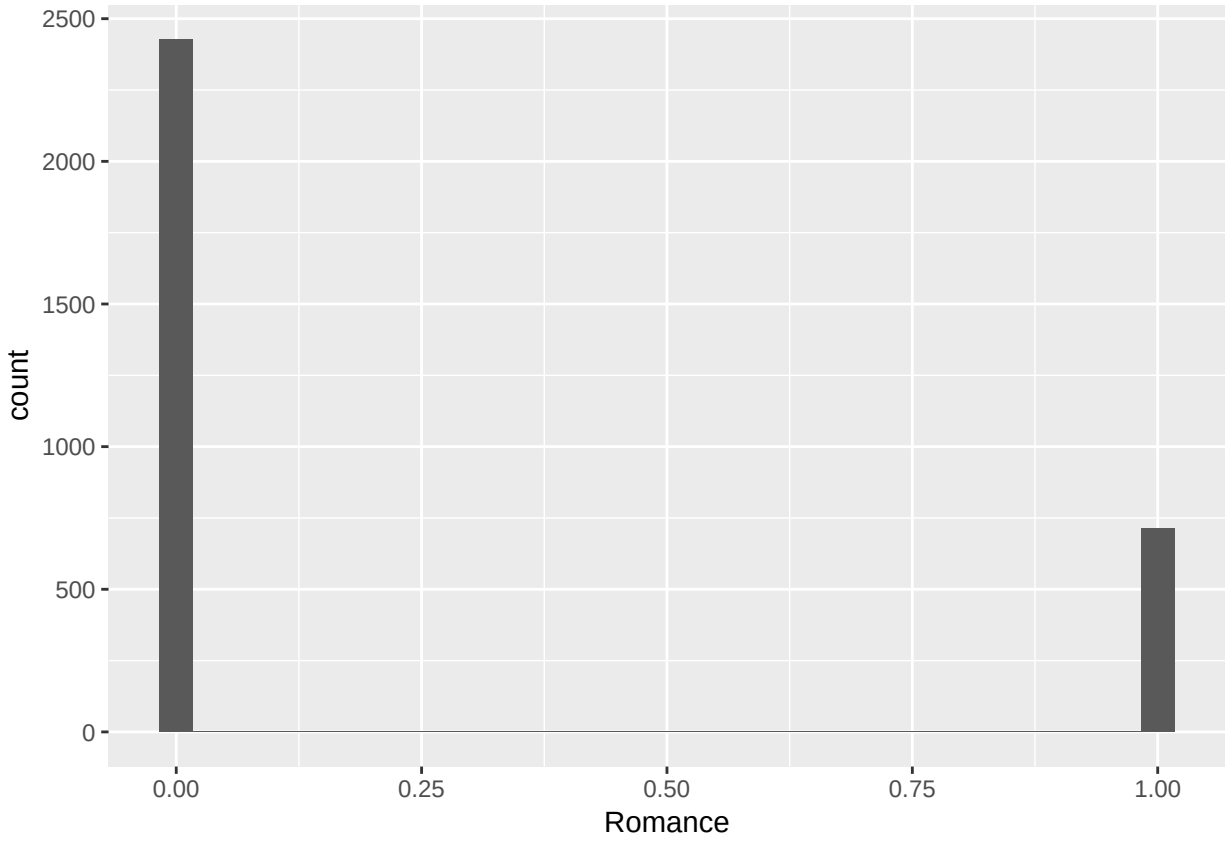

```
##
## [[30]]
```

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



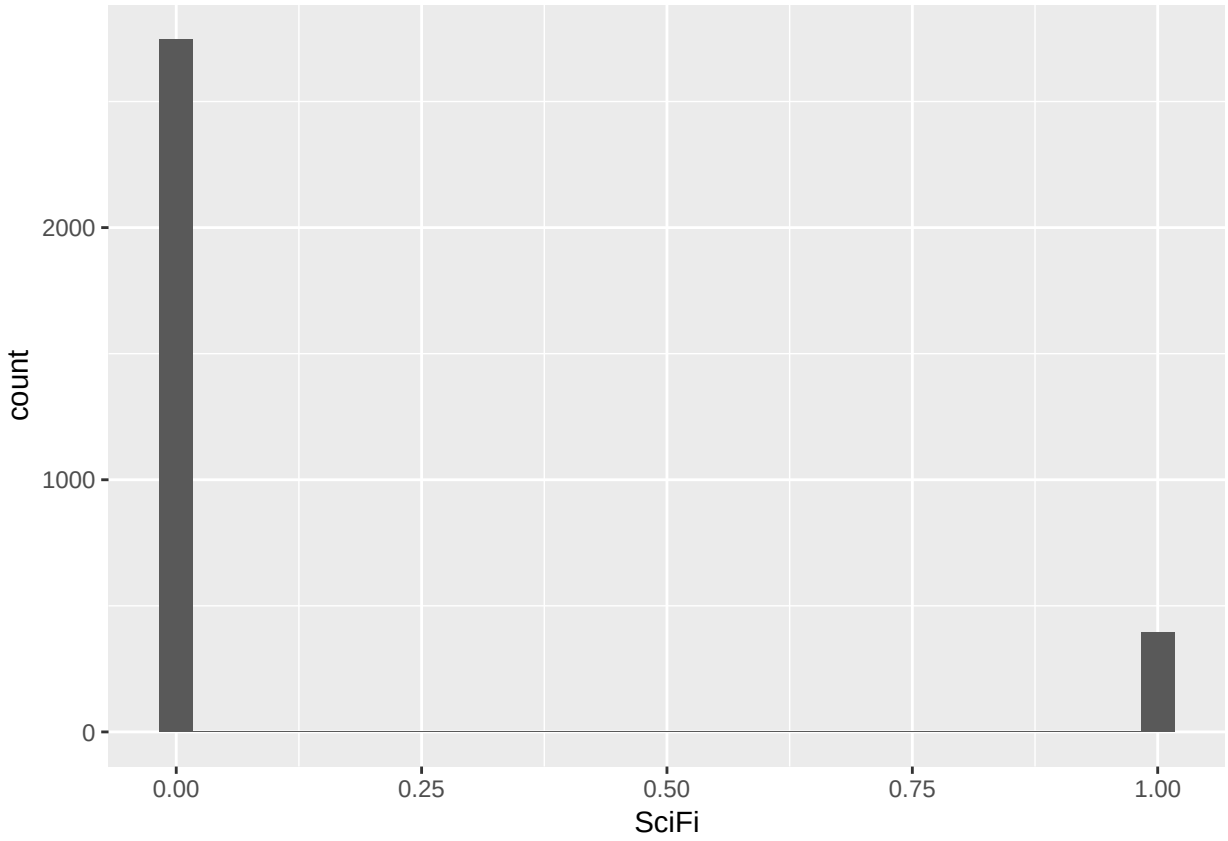

```
##
## [[31]]
```

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



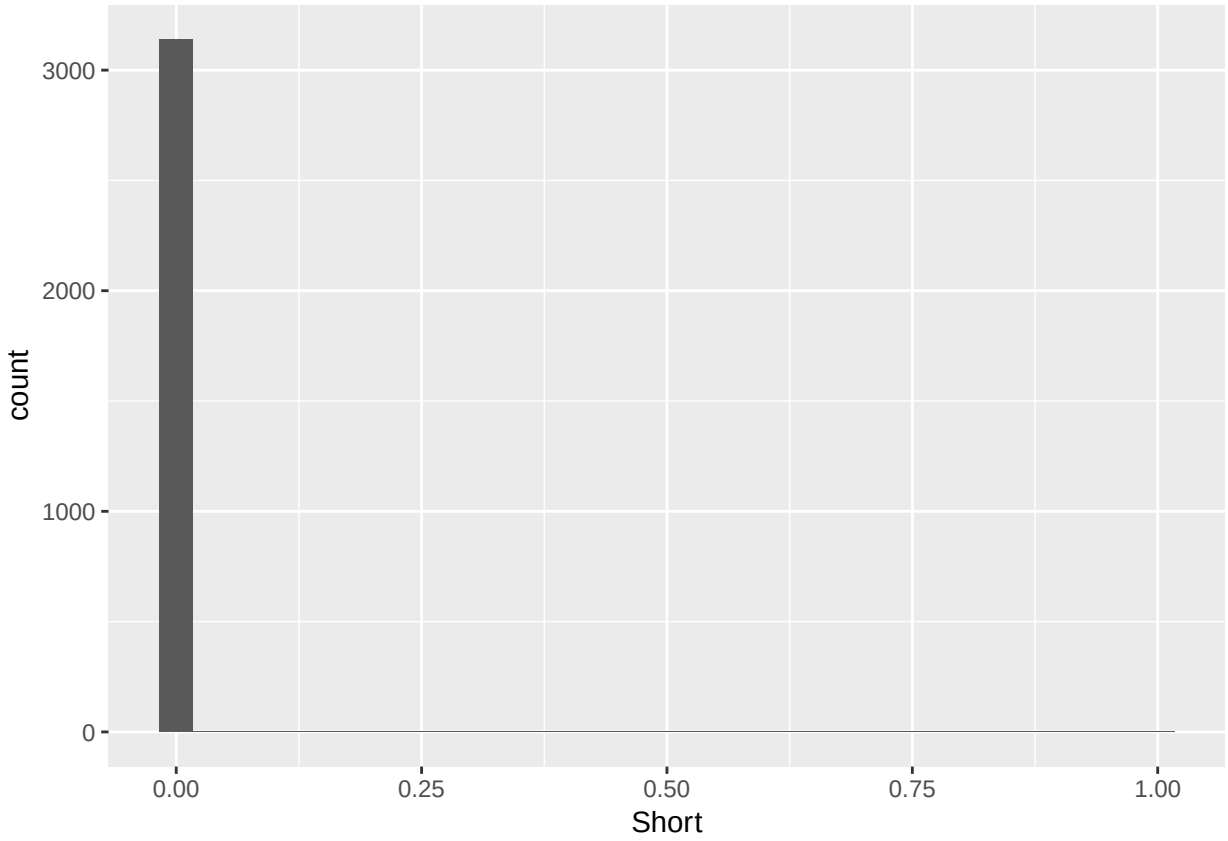

```
##
## [[32]]
```

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



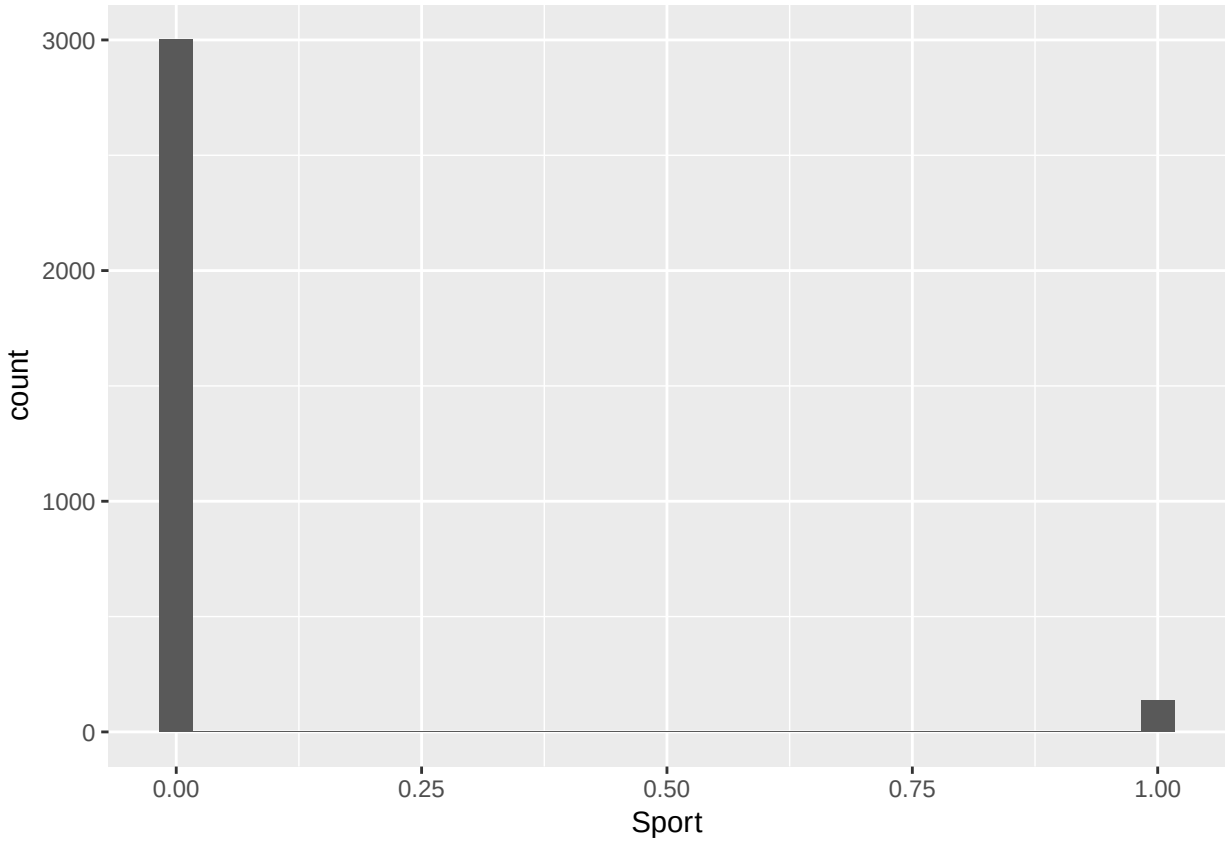

```
##
## [[33]]
```

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



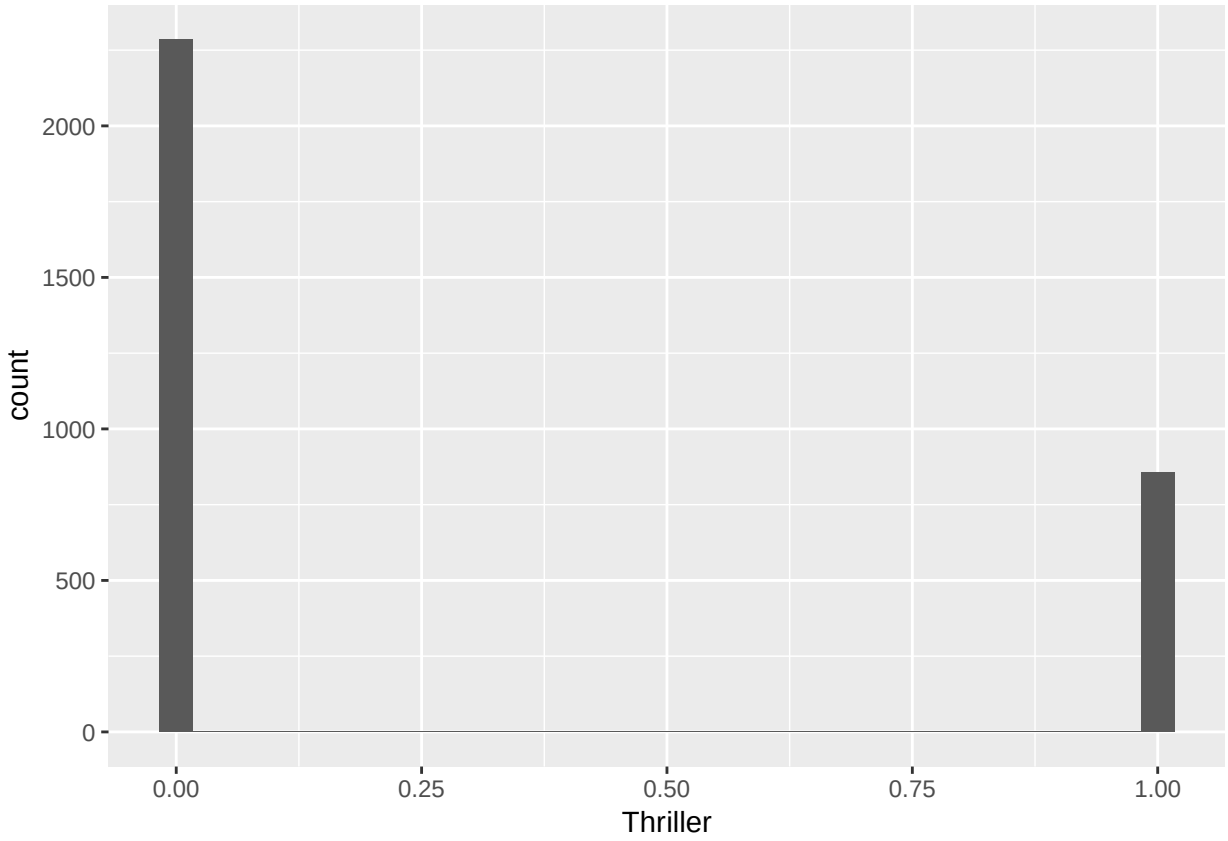

```
##
## [[34]]
```

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



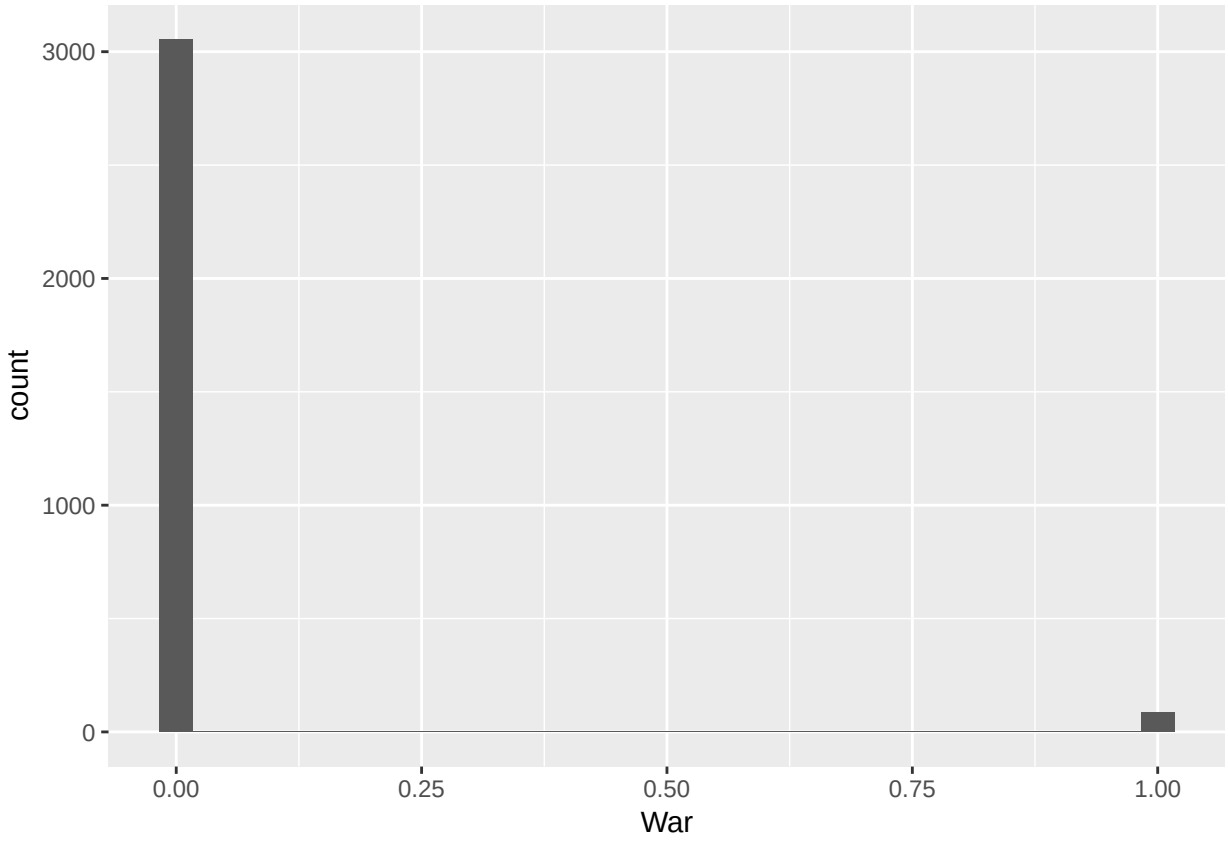

```
##
## [[35]]
```

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



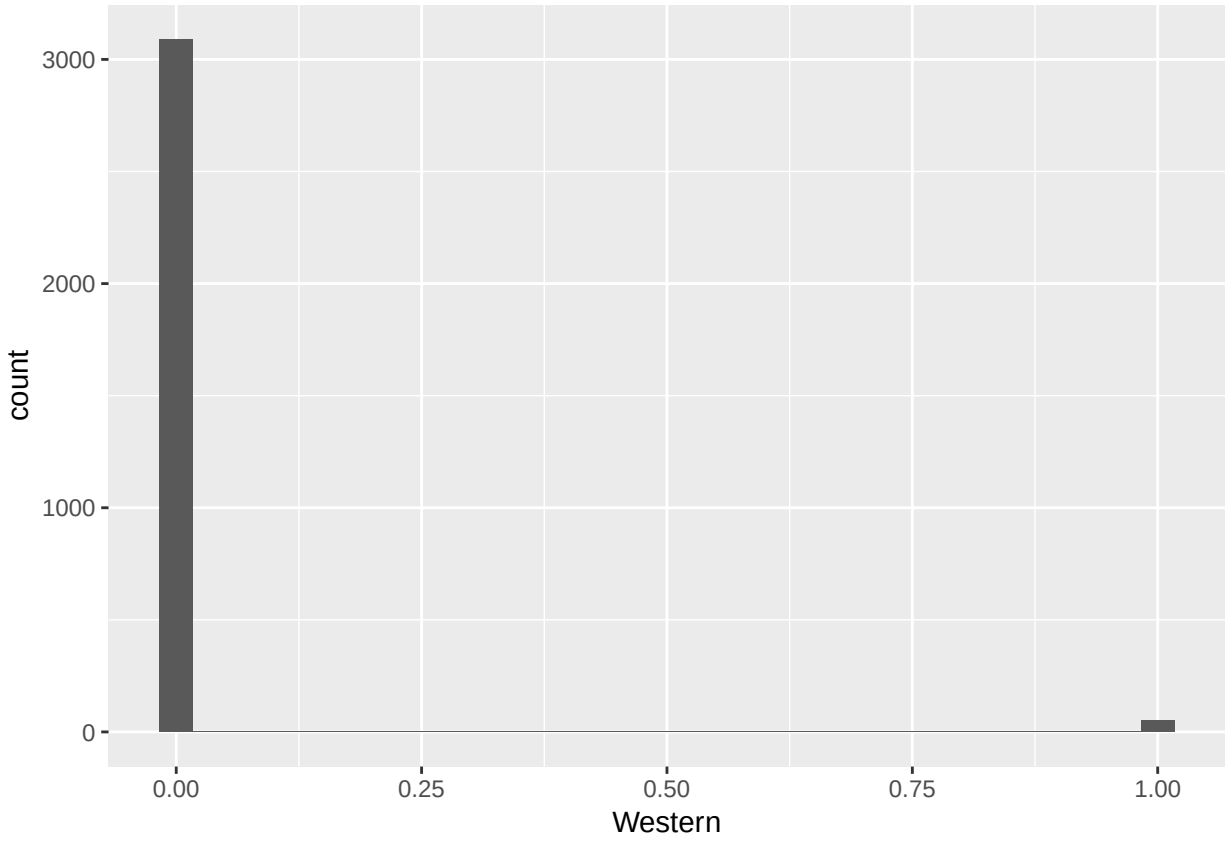

```
##
## [[36]]
```

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



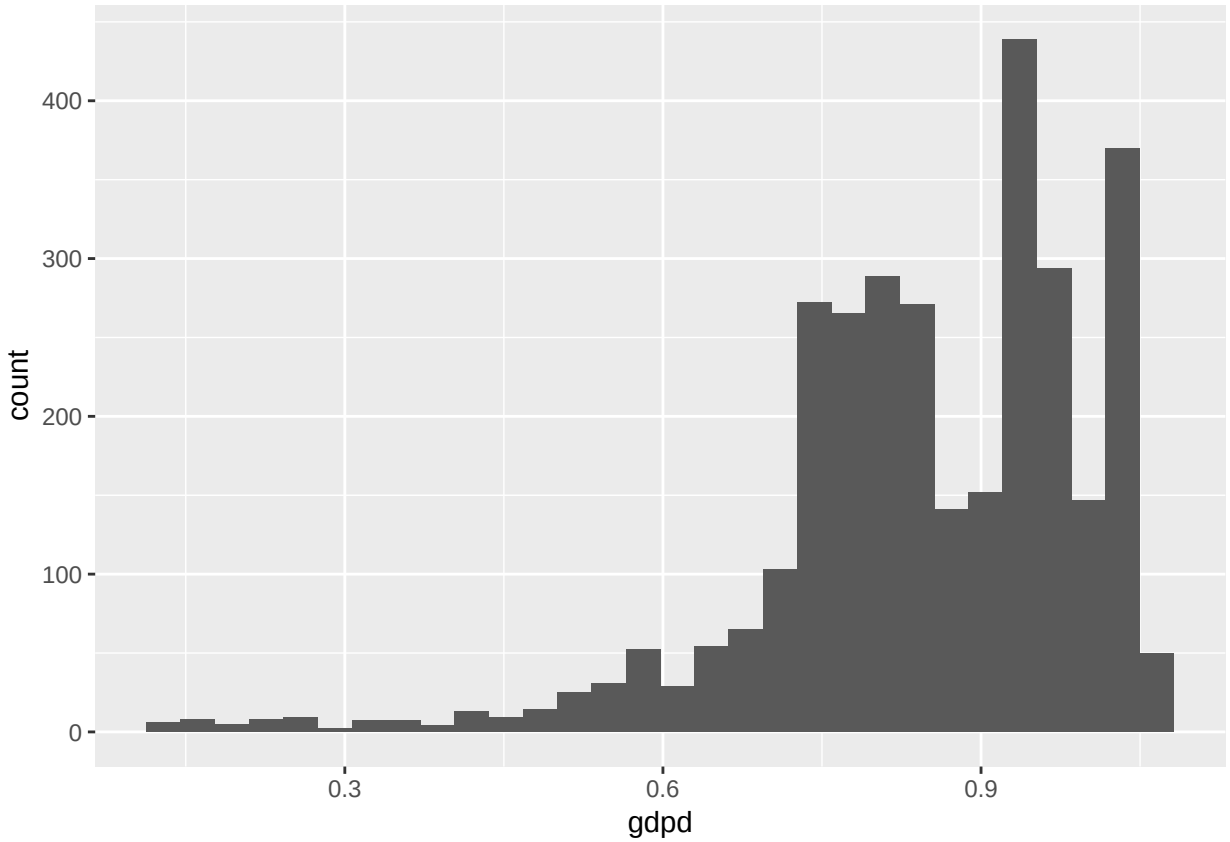

```
##
## [[37]]
```

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

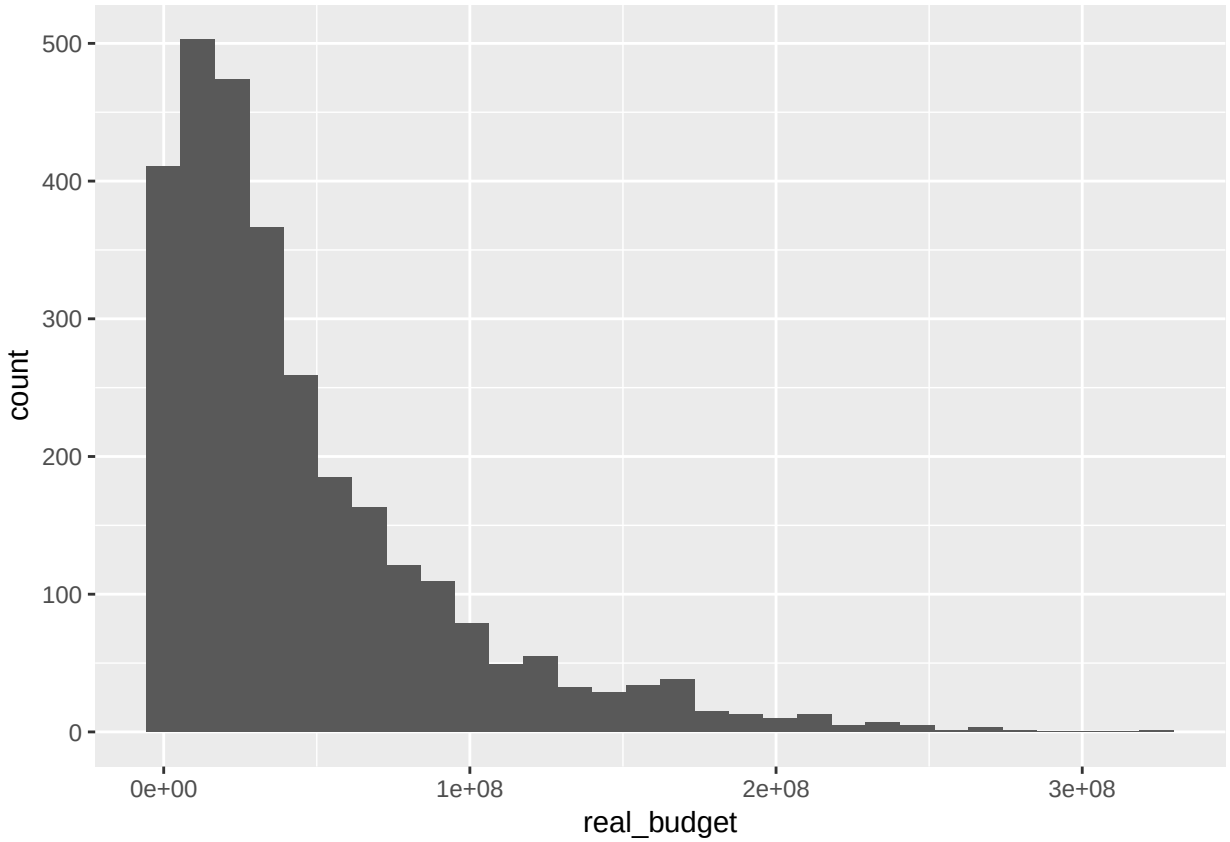



```
##
## [[38]]
```

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

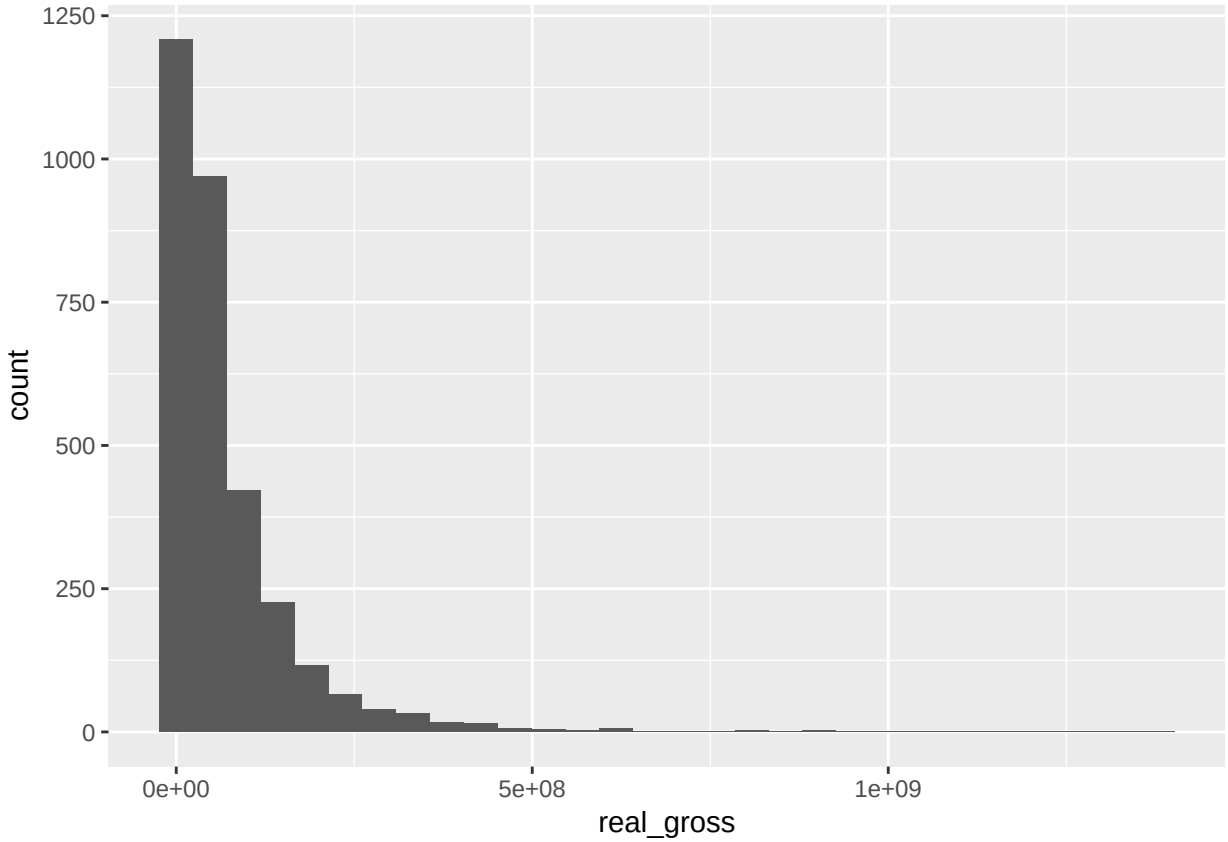



```
##
## [[39]]
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## Warning: Removed 160 rows containing non-finite values (stat_bin).
```



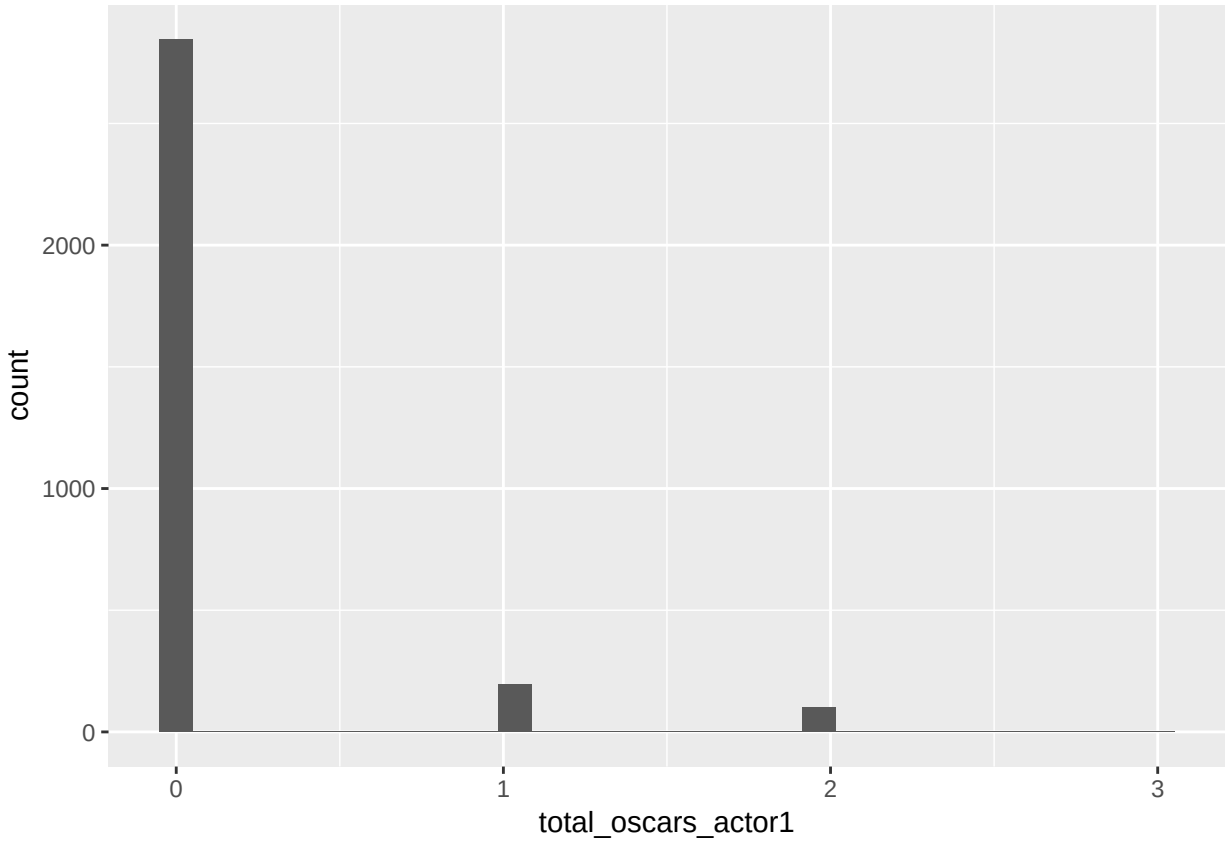

```
##
## [[40]]
```

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



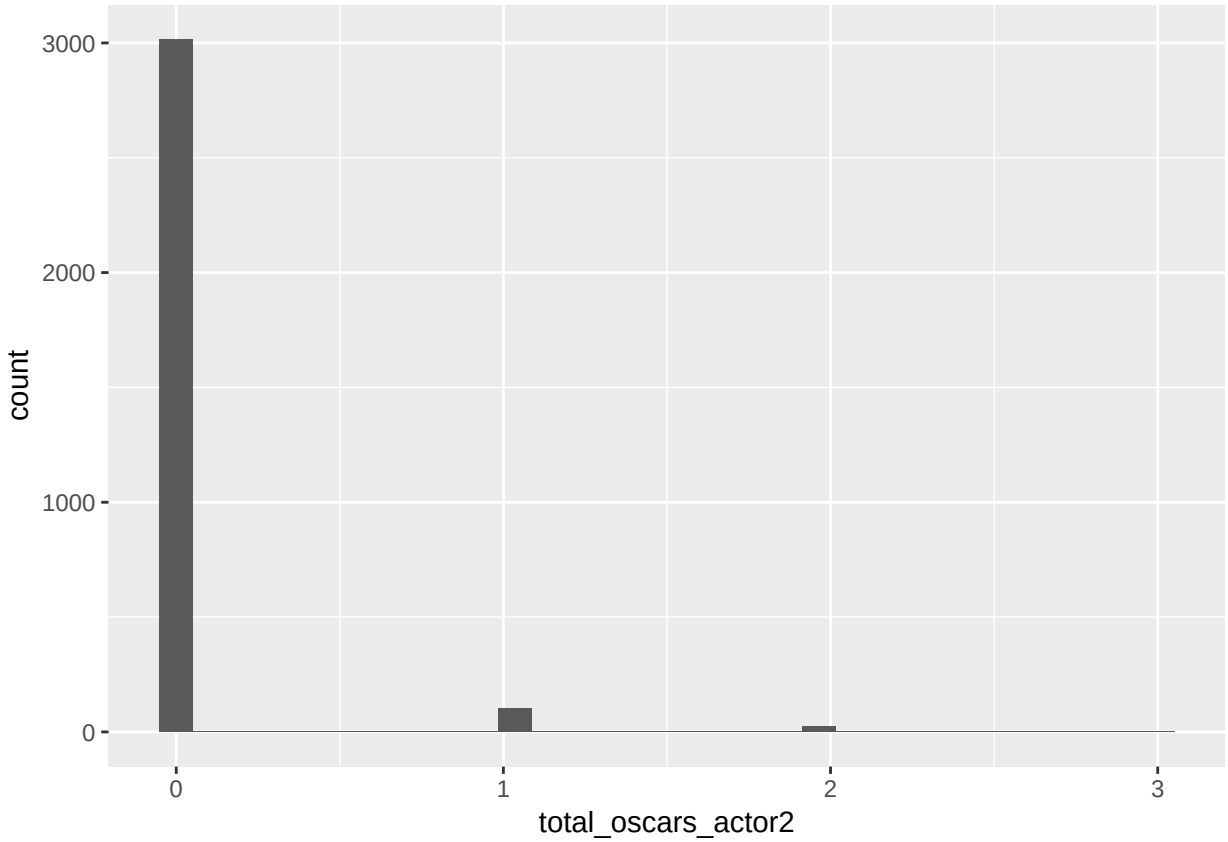

```
##
## [[41]]
```

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



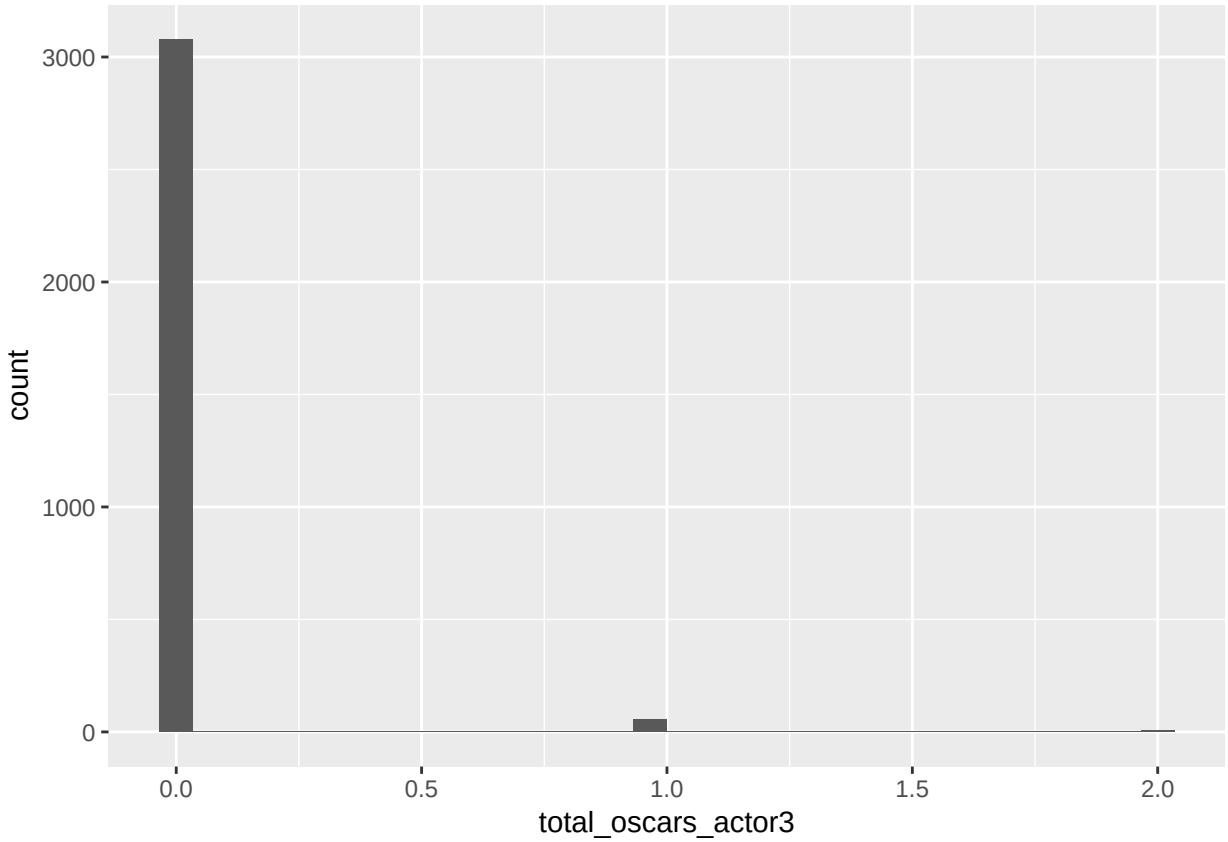

```
##
## [[42]]
```

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



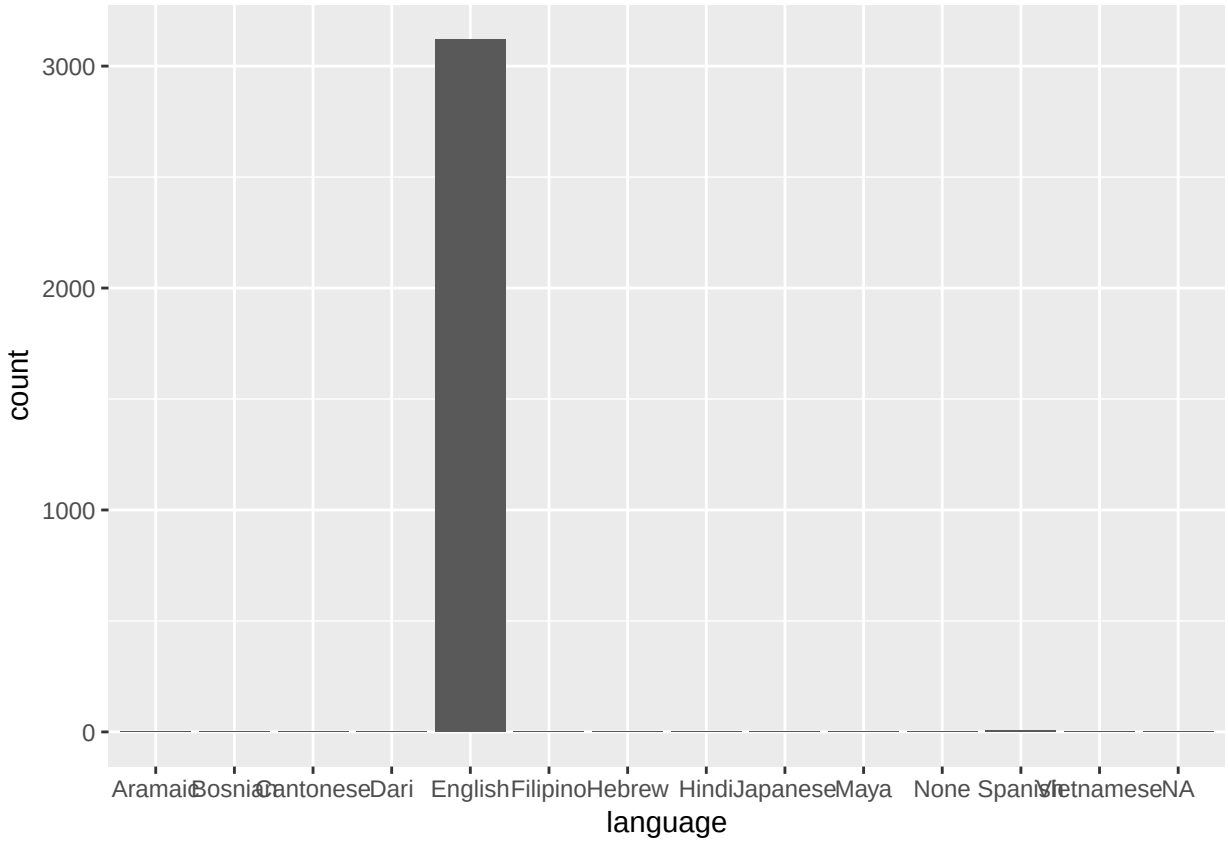

```
##
## [[43]]
```

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

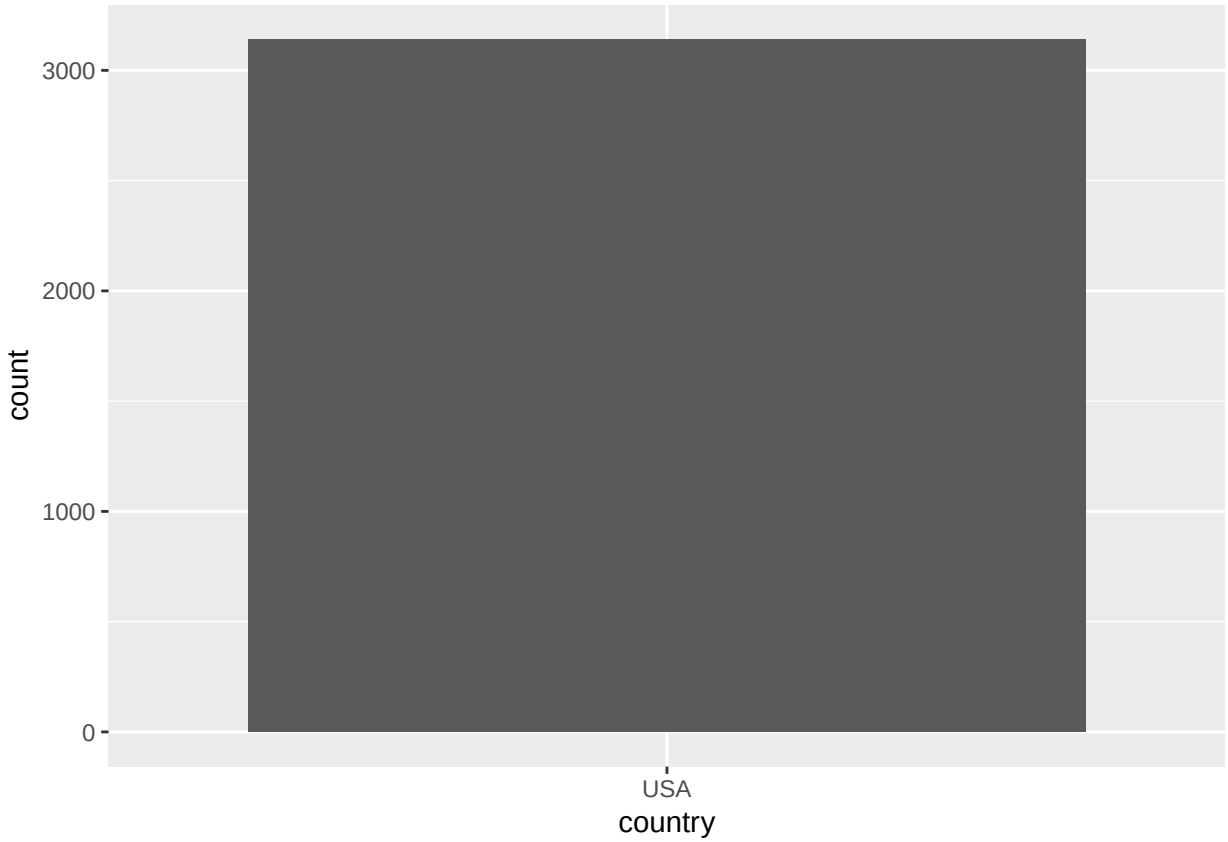



```
# character: bar plot
# specify these manually: dont want actors names, original genre column etc
lapply(c('language', 'country', 'content_rating'), function(var) {
  ggplot(movie, aes_string(x = var)) + geom_bar()
})
```

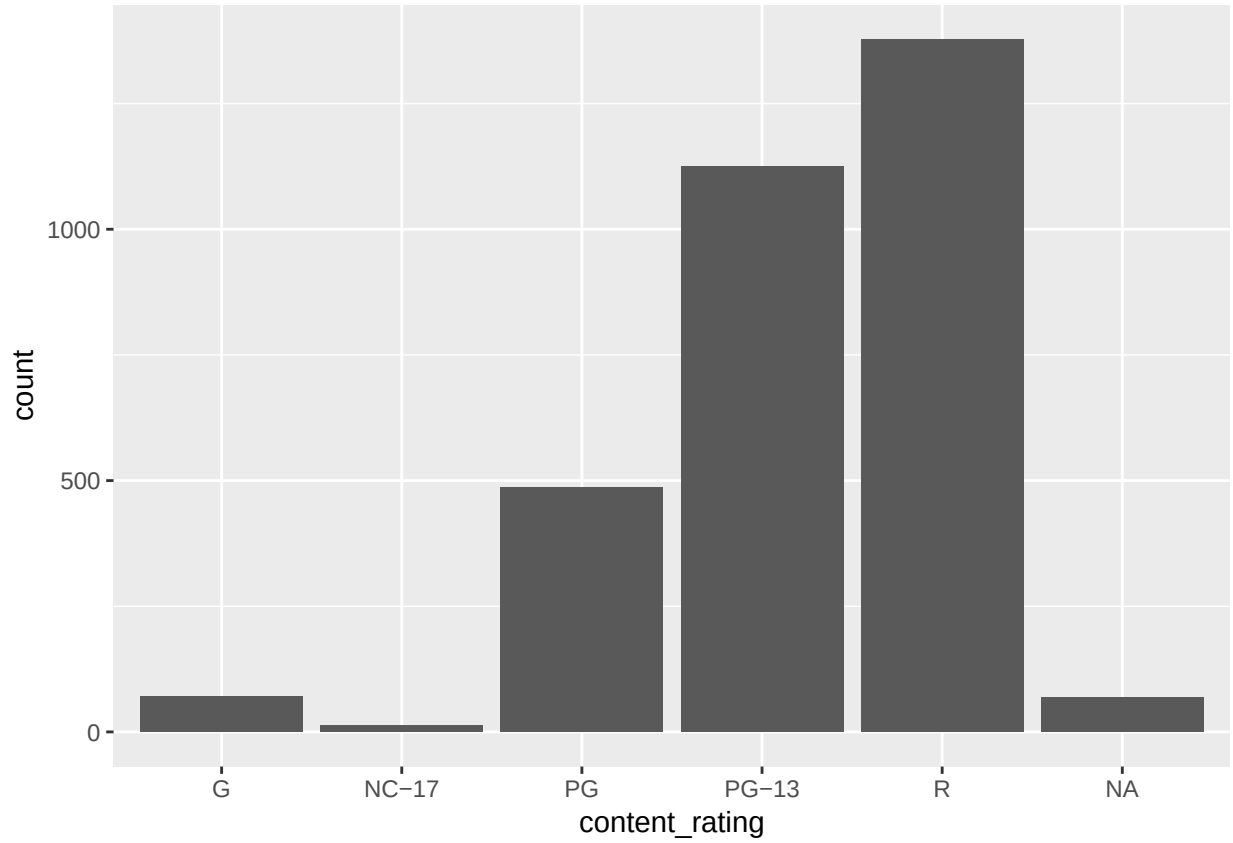
```
## [[1]]
```


3000
2000
1000
0
count
Aramaic
Bosnian
Cantonese
Dari
English
Filipino
Hebrew
Hindi
Japanese
Maya
None
Spanish
Vietnamese
NA
language

```
##
## [[2]]
```




```
##
## [[3]]
```



Look at outliers based on above visualizations

```
# duraton outliers
movie %>% filter(duration > 200 | duration < 60) %>% select(movie_title, duration, year)
# duration: pulled extended cut, director's cut etc from IMDB if available. (usually last run time list
# Example: Heaven's Gate original rough cut was 325 minutes/over 5 hours. However, final cut was around
# could trim the outliers, but even for non-outlier movies, I found several examples where the official

# facebook likes. Big movies and directors with lots of likes. Seems correct
movie %>% filter(director_facebook_likes > 10000) %>% select(director_name)
movie %>% filter(cast_total_facebook_likes > 100000) %>% select(movie_title)

# revenue and budget outliers
movie %>% arrange(real_gross) %>% select(movie_title, real_gross, year) %>% head()
movie %>% arrange(desc(real_gross)) %>% select(movie_title, real_gross, gross, budget, year) %>% head()
# gross is generally looking at gross USA not worldwide gross
# the outliers for gross appear to be legit
movie %>% arrange(real_budget) %>% select(movie_title, real_budget, year, gross) %>% head()
movie %>% arrange(desc(real_budget)) %>% select(movie_title, real_budget, budget, gross, year) %>% head
# outliers appear to be legit

# genres with very few films
movie %>% filter(FilmNoir == 1) # 1
movie %>% filter(News == 1) # 1
movie %>% filter(Short == 1) # 1
# drop these three genres. All 3 of these movies also have other genres indicated
movie %>% filter(Western == 1) # 52
```

```
movie %>% filter(War == 1) # 88
# these have enough that they are ok.
```

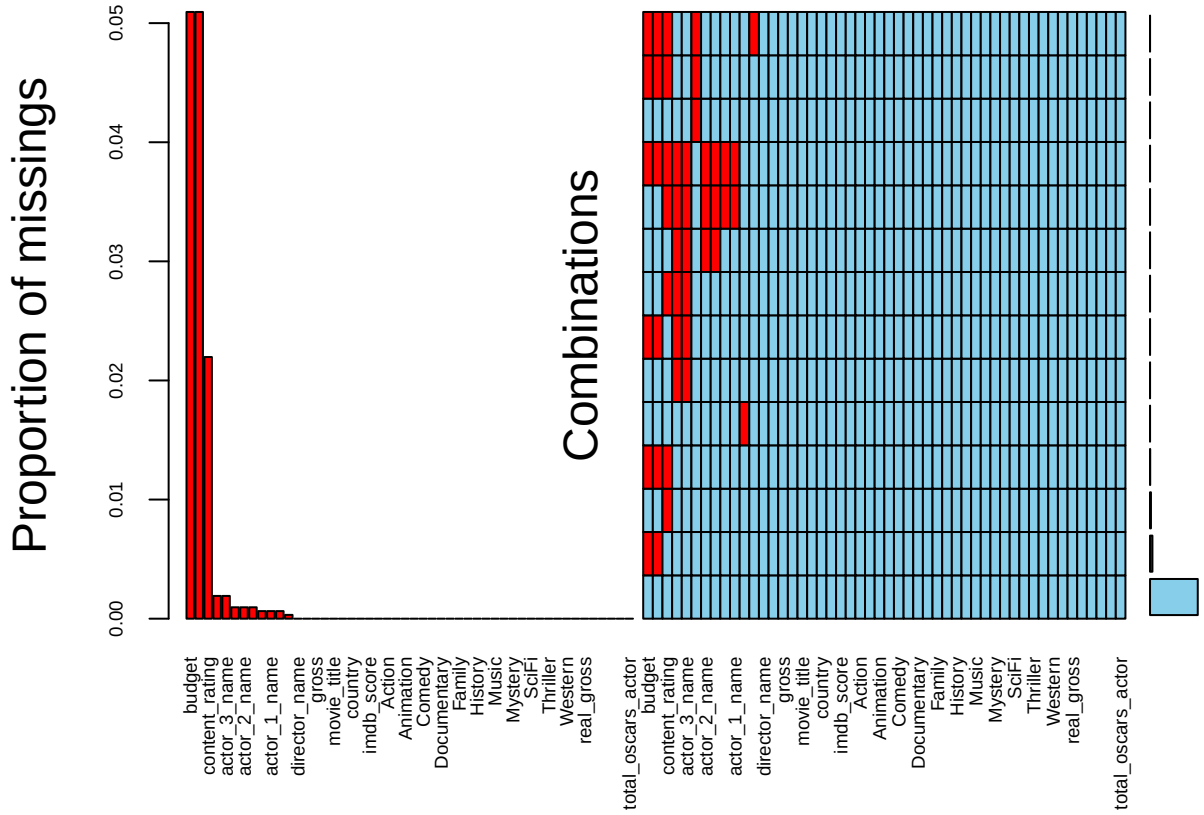
Drop some variables based on the above: genres with only one movie (FilmNoir, News, Short) - not helpful for analysis since only one data point. Also dropping duration variable because of the data quality issues that are not fixable and would lead to misleading reuslts.

```
movie <- movie %>% select(-c(FilmNoir, News, Short, duration))
```

Missing Variable Visualization

After all of this cleaning, look at a visualization of the missing values for each variable **I don't have all of the data on my work computer, so this has not been tested. Will run on home later**

```
missing_values <- aggr(movie, sortVars = T, prop = T, sortCombs = T, cex.lab = 1.5, cex.axis = .6, cex.
```




```
##
##  Variables sorted by number of missings:
##                    Variable        Count
##                      budget 0.0509391913
##                 real_budget 0.0509391913
##              content_rating 0.0219675263
##     actor_3_facebook_likes 0.0019102197
##                actor_3_name 0.0019102197
##     num_critic_for_reviews 0.0009551098
##                actor_2_name 0.0009551098
##     actor_2_facebook_likes 0.0009551098
```

```
##     actor_1_facebook_likes 0.0006367399
##               actor_1_name 0.0006367399
##                   language 0.0006367399
##       num_user_for_reviews 0.0003183699
##              director_name 0.0000000000
##    director_facebook_likes 0.0000000000
##                      gross 0.0000000000
##                     genres 0.0000000000
##                movie_title 0.0000000000
##  cast_total_facebook_likes 0.0000000000
##                    country 0.0000000000
##                       year 0.0000000000
##                 imdb_score 0.0000000000
##       movie_facebook_likes 0.0000000000
##                     Action 0.0000000000
##                  Adventure 0.0000000000
##                  Animation 0.0000000000
##                  Biography 0.0000000000
##                     Comedy 0.0000000000
##                      Crime 0.0000000000
##                Documentary 0.0000000000
##                      Drama 0.0000000000
##                     Family 0.0000000000
##                    Fantasy 0.0000000000
##                    History 0.0000000000
##                     Horror 0.0000000000
##                      Music 0.0000000000
##                    Musical 0.0000000000
##                    Mystery 0.0000000000
##                    Romance 0.0000000000
##                      SciFi 0.0000000000
##                      Sport 0.0000000000
##                   Thriller 0.0000000000
##                        War 0.0000000000
##                    Western 0.0000000000
##                       gdpd 0.0000000000
##                 real_gross 0.0000000000
##        total_oscars_actor1 0.0000000000
##        total_oscars_actor2 0.0000000000
##        total_oscars_actor3 0.0000000000
##      total_oscars_director 0.0000000000
##         total_oscars_actor 0.0000000000
```

Partition data

Choosing 60% train because data not that big to begin with so want to make sure valid and test are big enough

```
set.seed(5)
movie_sets <- resample_partition(movie, c(train = .6, valid = .2, test = .2))
train <- as_tibble(movie_sets$train) # 1885
valid <- as_tibble(movie_sets$valid) # 629
test <- as_tibble(movie_sets$test) # 629
```

Save data

```
save(train, valid, test, file = '~/DS5110/data/proj_cleaned_dta.RData')
```