

eda

```
install.packages("GGally")
install.packages("moments")
install.packages("corrplot")

library(GGally)
library(ggplot2)
library(lmtest)
library(moments)
library(sandwich)
library(stargazer)
library(tidyverse)
library(corrplot)
library(data.table)

source('~/get_robust_se.R')

d <- read.csv('data/googleplaystore.csv')
summary(d)

##      App          Category       Rating      Reviews
##  Length:10841    Length:10841   Min.   : 1.000  Length:10841
##  Class :character Class :character  1st Qu.: 4.000  Class :character
##  Mode  :character Mode  :character  Median : 4.300  Mode  :character
##                                         Mean   : 4.193
##                                         3rd Qu.: 4.500
##                                         Max.   :19.000
##                                         NA's   :1474
##      Size          Installs        Type        Price
##  Length:10841    Length:10841   Length:10841  Length:10841
##  Class :character Class :character Class :character Class :character
##  Mode  :character Mode  :character Mode  :character Mode  :character
##                                         Type
##                                         Class :character
##                                         Class :character
##                                         Mode  :character
##                                         Mode  :character
##                                         NA's   :1474
##      Content.Rating     Genres      Last.Updated Current.Ver
##  Length:10841    Length:10841   Length:10841  Length:10841
##  Class :character Class :character Class :character Class :character
##  Mode  :character Mode  :character Mode  :character Mode  :character
##                                         Genres
##                                         Class :character
##                                         Class :character
##                                         Mode  :character
##                                         Mode  :character
##                                         NA's   :1474
##      Android.Ver
##  Length:10841
##  Class :character
##  Mode  :character
```

```

## 
## 
## 
## 

# Function to convert strings to numbers
str_to_num <- function(s) {
  n <- nchar(s)
  last <- substr(s, n, n + 1)
  if (last == "B") {
    number <- as.numeric(substr(s, 0, n - 1)) * 1.0e9
  } else if (last == "M") {
    number <- as.numeric(substr(s, 0, n - 1)) * 1.0e6
  } else if (last == "k") {
    number <- as.numeric(substr(s, 0, n - 1)) * 1.0e3
  } else if (last == "+") {
    number <- as.numeric(str_replace_all(s, "[+,]", ""))
  }
  else {
    number <- as.numeric(s)
  }
  return(number)
}

# Remove entries with bad values
d <- d[d$Installs != "Free", ]
d <- d[!is.nan(d$Rating), ]
d <- d[d$Size != "Varies with device", ]

# Convert strings to numbers
d$installs_clean <- sapply(d$Installs, str_to_num)
d$reviews_clean <- sapply(d$Reviews, str_to_num)
d$size_clean <- sapply(d$Size, str_to_num)
d$price_clean <- as.numeric(gsub('\\$', '', d$Price))

# Bins for install count

# install groups-- increments by 1, there are 19 of them.
install_groups <- data.table(table(d$installs_clean))
setnames(install_groups,c('V1','N'),c('installs_clean','count'))
install_groups[, install_group := 1:N]
install_groups <- install_groups[,c('installs_clean','install_group')]
install_groups

##      installs_clean install_group
## 1:              1             1
## 2:              5             2
## 3:             10             3
## 4:             50             4
## 5:            100             5
## 6:            500             6
## 7:           1000             7
## 8:           5000             8
## 9:          10000             9
## 10:          50000            10

```

```

## 11:      100000          11
## 12:      500000          12
## 13:     1000000          13
## 14:     5000000          14
## 15:    10000000          15
## 16:    50000000          16
## 17:   100000000          17
## 18:   500000000          18
## 19:  1000000000          19

d <- merge(d,install_groups,by='installs_clean')

# summary of dataset
summary(d)

## installs_clean          App           Category        Rating
## Min. :       1 Length:7729      Length:7729      Min. :1.000
## 1st Qu.: 10000 Class :character Class :character 1st Qu.:4.000
## Median : 100000 Mode  :character Mode  :character Median :4.300
## Mean   : 8417734
## 3rd Qu.: 1000000
## Max.  :1000000000          Reviews        Size           Installs        Type
## Length:7729      Length:7729      Length:7729      Length:7729
## Class :character Class :character Class :character Class :character
## Mode  :character Mode  :character Mode  :character Mode  :character
##
## 
## 
## 
## Price          Content.Rating      Genres        Last.Updated
## Length:7729      Length:7729      Length:7729      Length:7729
## Class :character Class :character Class :character Class :character
## Mode  :character Mode  :character Mode  :character Mode  :character
##
## 
## 
## 
## Current.Ver      Android.Ver      reviews_clean      size_clean
## Length:7729      Length:7729      Min. :       1 Min. : 8500
## Class :character Class :character 1st Qu.:     108 1st Qu.: 5300000
## Mode  :character Mode  :character Median :    2328 Median : 14000000
## 
## Mean   : 294673 Mean   : 22957607
## 3rd Qu.: 38961 3rd Qu.: 33000000
## Max.  :44893888 Max.  :100000000
## 
## price_clean      install_group
## Min. : 0.000 Min. : 1
## 1st Qu.: 0.000 1st Qu.: 9
## Median : 0.000 Median :11
## Mean   : 1.128 Mean   :11
## 3rd Qu.: 0.000 3rd Qu.:13
## Max.  :400.000 Max.  :19

# save a data.table version for some easier wrangling downstream
d_dt <- as.data.table(d)

```

```

numeric_cols <- colnames(d)[unlist(lapply(d, is.numeric))]

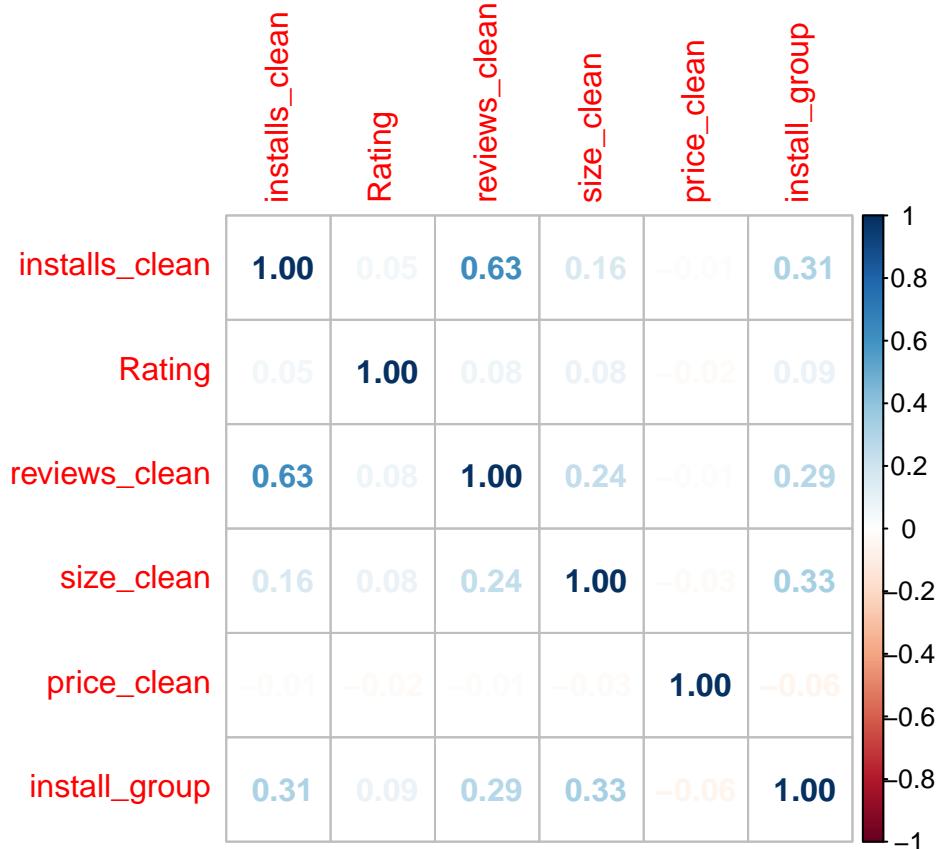
# show distribution of numeric columns
create_quantile_table <- function(column) {
  probs <- sort(c(seq(0, 1, 0.25), 0.05, 0.95))
  result <-
    as.data.table(t(data.table(quantile(d[, column], probs = probs))))
  setnames(result, paste0(probs * 100, '%'))
  result[, variable := column]
  result[, diff_min_vs_max := `100%` - `0%`]
  return(result)
}

table_quantile_numeric <- rbindlist(lapply(numeric_cols, create_quantile_table))
table_quantile_numeric

##      0%      5%     25%     50%     75%     95%    100%
## 1:   1  100.0 10000 1000000.0 1000000.0 50000000.00 10000000000
## 2:   1     3.1     4     4.3     4.5     4.80      5
## 3:   1     6.0    108   2328.0   38961.0   855727.00  44893888
## 4: 8500 1500000.0 5300000 14000000.0 33000000.0 76000000.00 100000000
## 5:   0     0.0     0     0.0     0.0     1.99     400
## 6:   1     5.0     9    11.0    13.0    16.00     19
##           variable diff_min_vs_max
## 1: installs_clean 999999999
## 2:          Rating        4
## 3: reviews_clean 44893887
## 4:    size_clean 99991500
## 5:   price_clean     400
## 6: install_group       18

# Corrplot across variables
corrplot(cor(d[, numeric_cols], use = "complete.obs"),
         method = 'number')

```



```
# Corrplot across variables (5th PCTL outliers removed)
corrplot(cor(d[d$reviews_clean >= 6, numeric_cols], use = "complete.obs"),
         method = 'number')
```

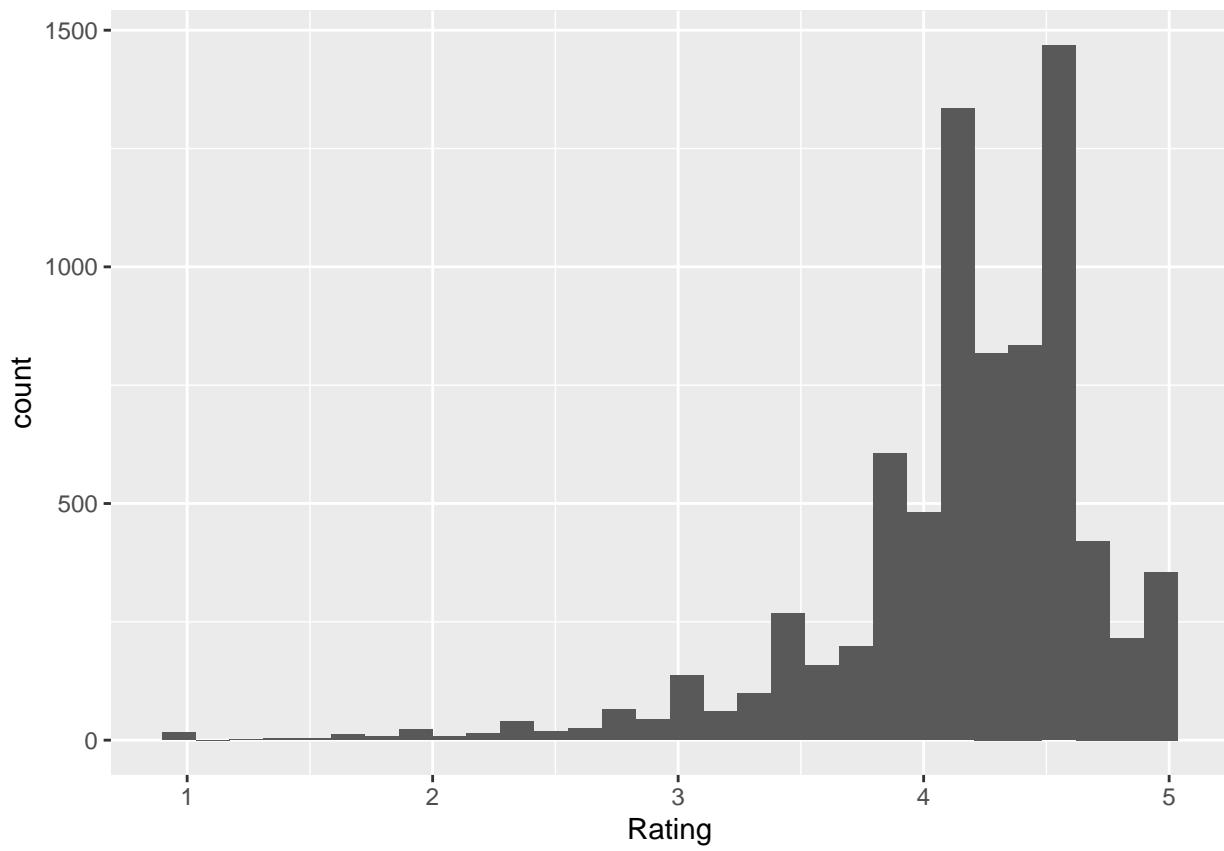


```
# Corrplot across variables (25th PCTL outliers removed)
corrplot(cor(d[d$reviews_clean >= 100, numeric_cols], use = "complete.obs"),
         method = 'number')
```

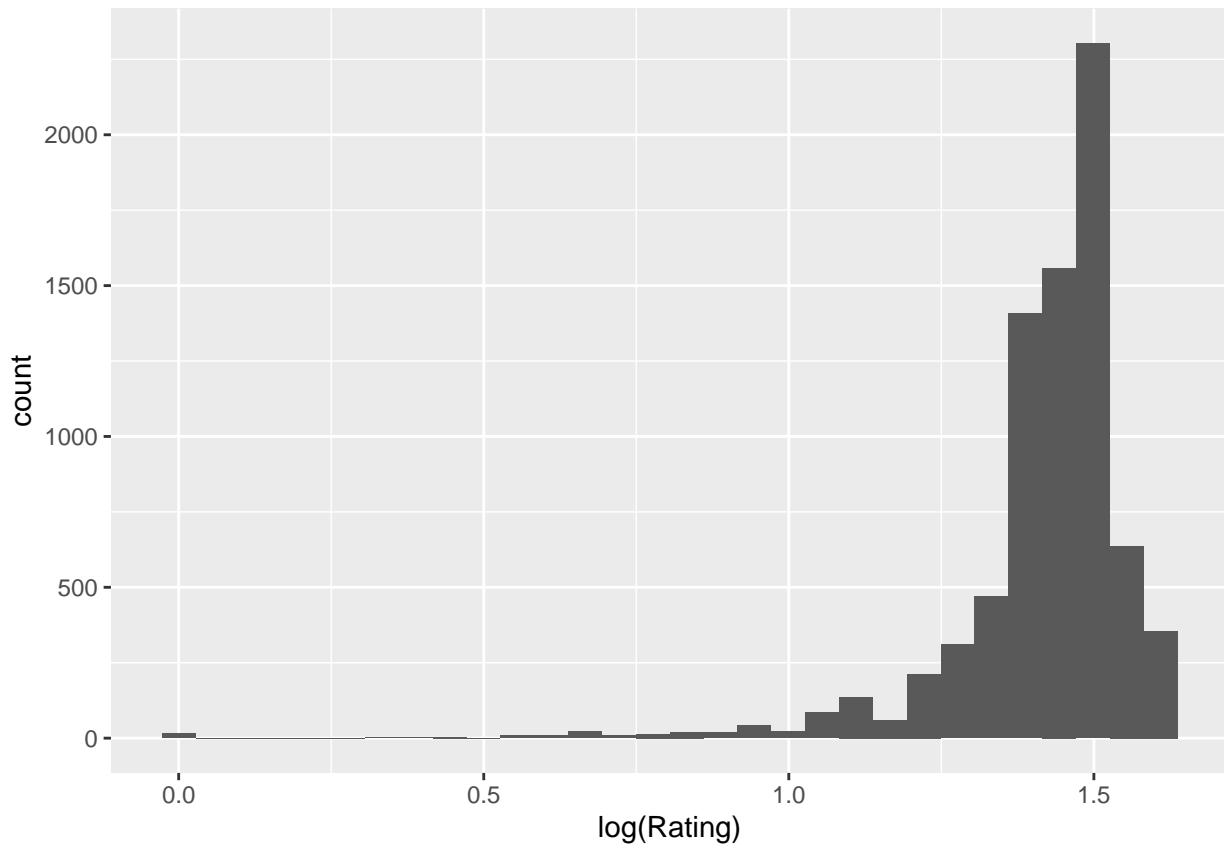


```
# Distribution of numeric columns
ggplot(data = d, aes(x = Rating)) +
  geom_histogram()

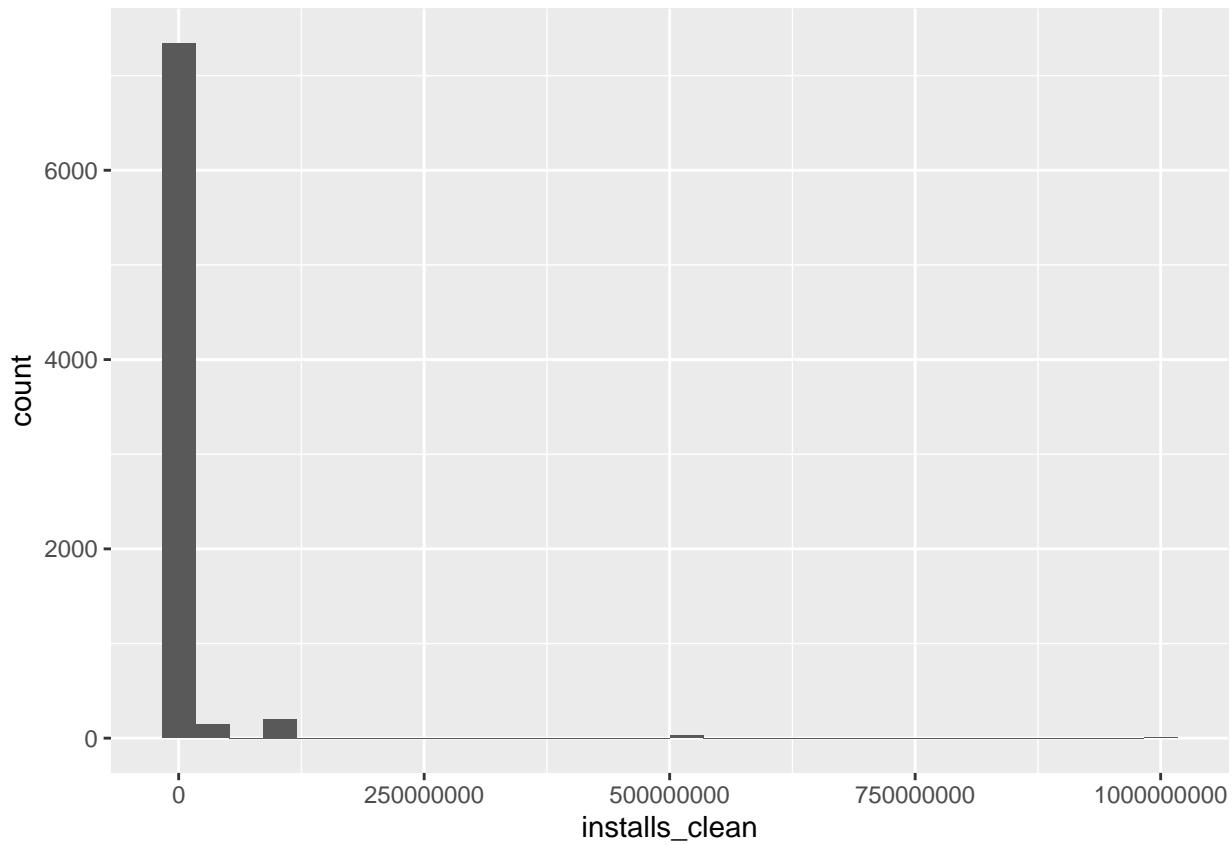
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



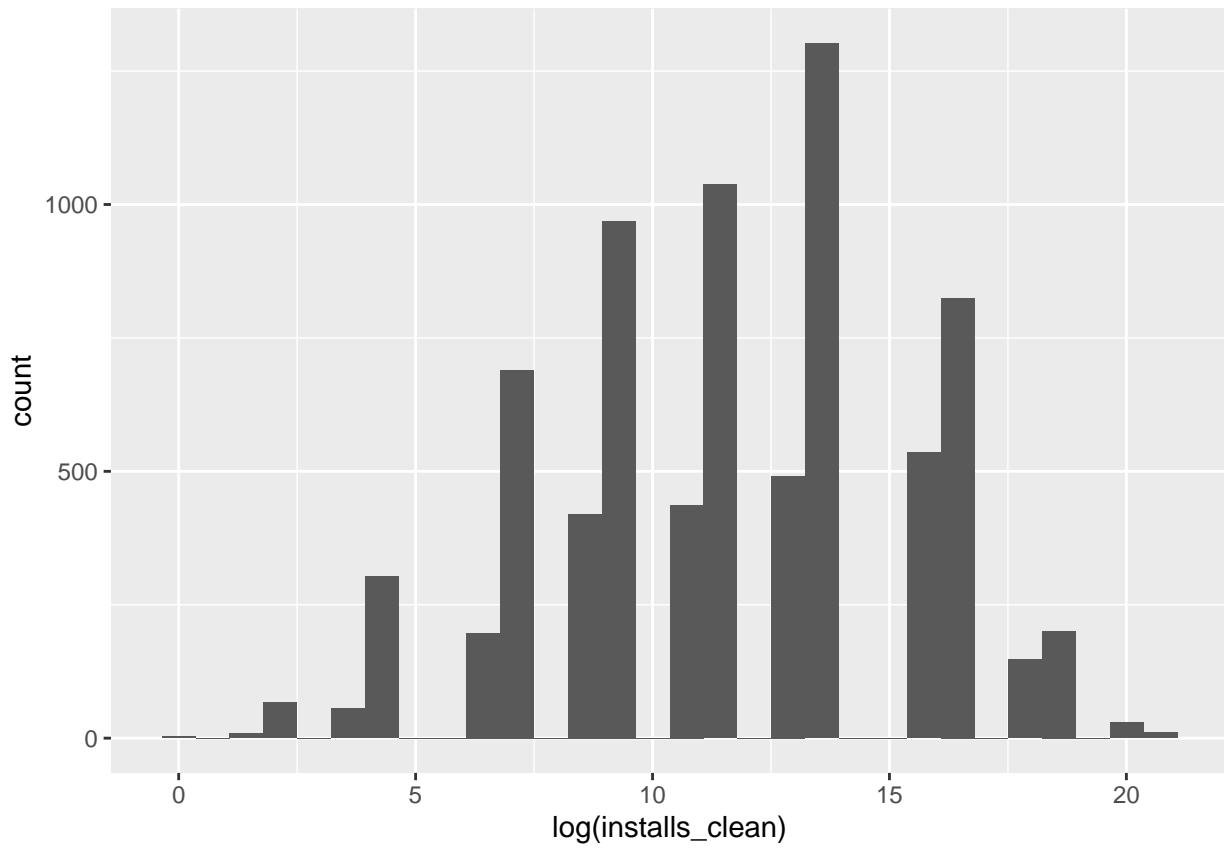
```
ggplot(data = d, aes(x = log(Rating))) +  
  geom_histogram()  
  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



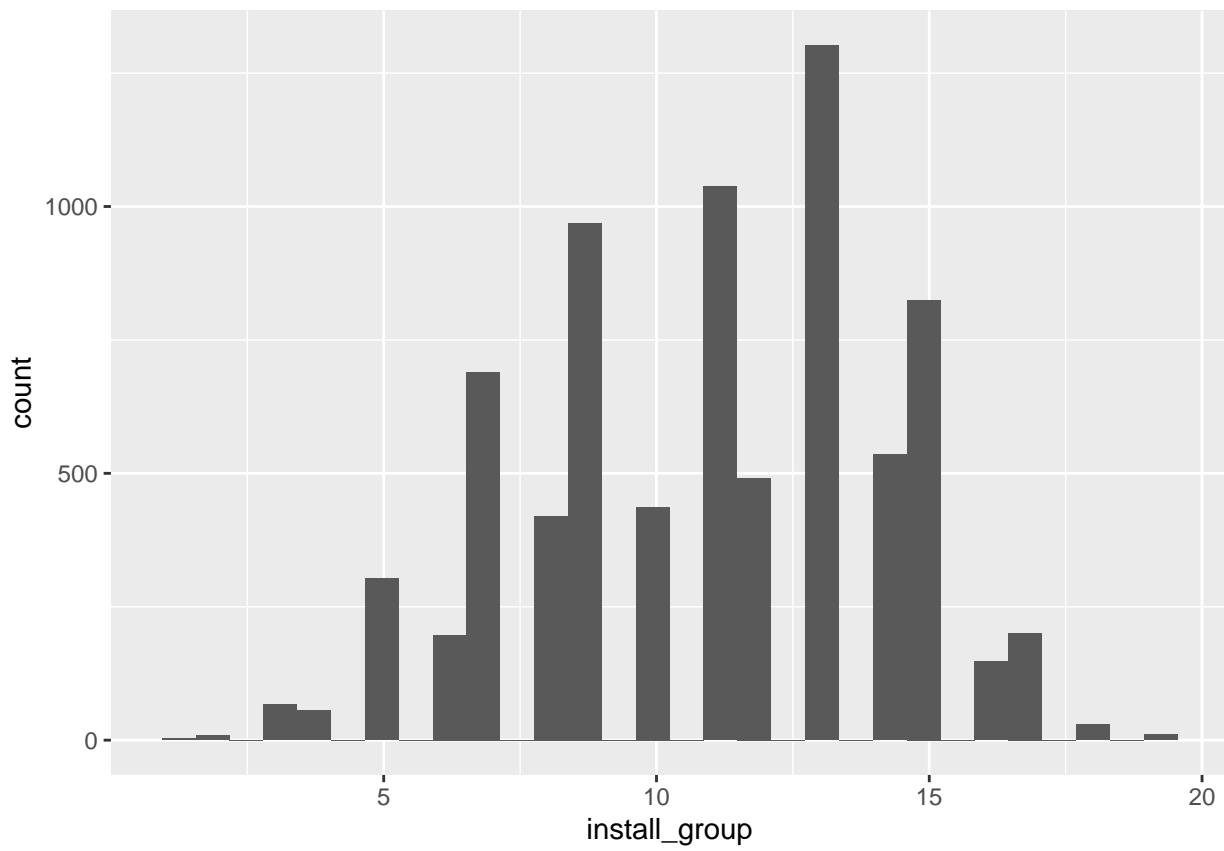
```
ggplot(data = d, aes(x = installs_clean)) +  
  geom_histogram()  
  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



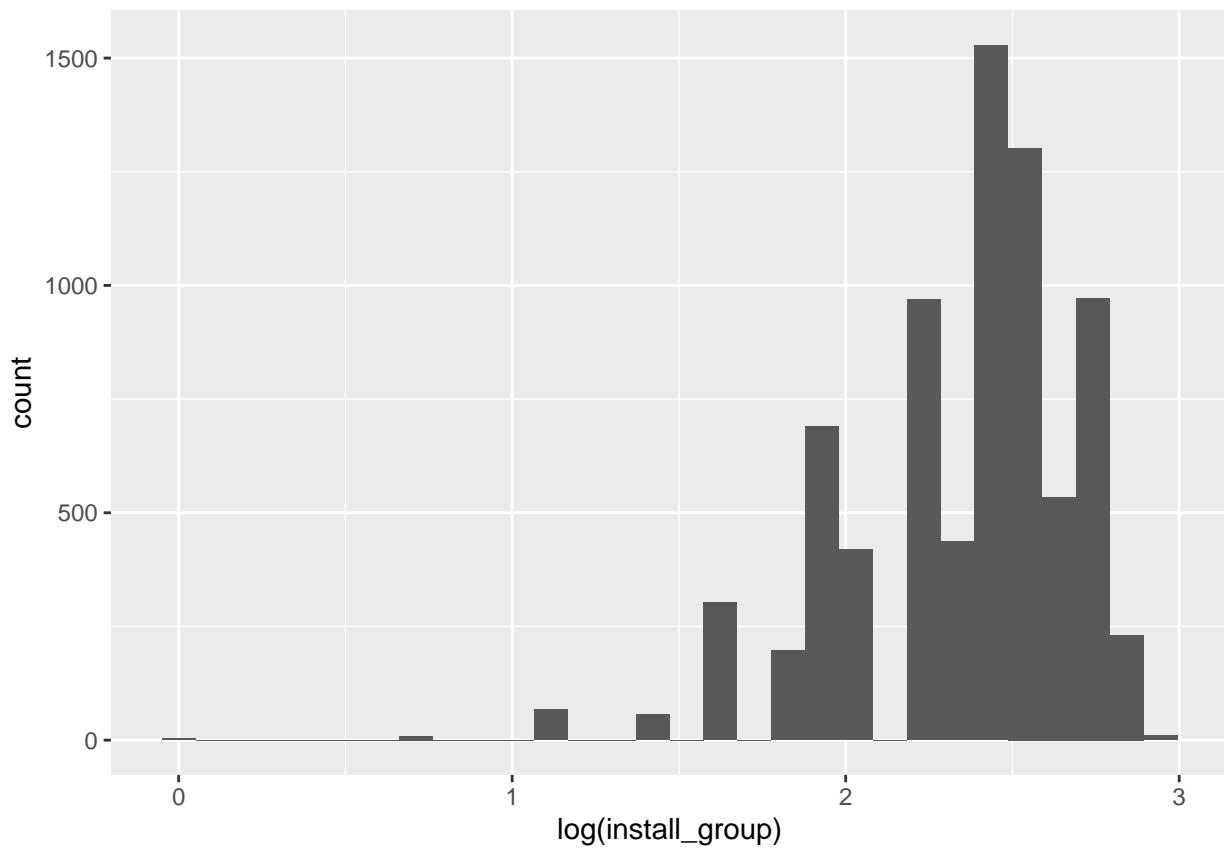
```
ggplot(data = d, aes(x = log(installs_clean))) +  
  geom_histogram()  
  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



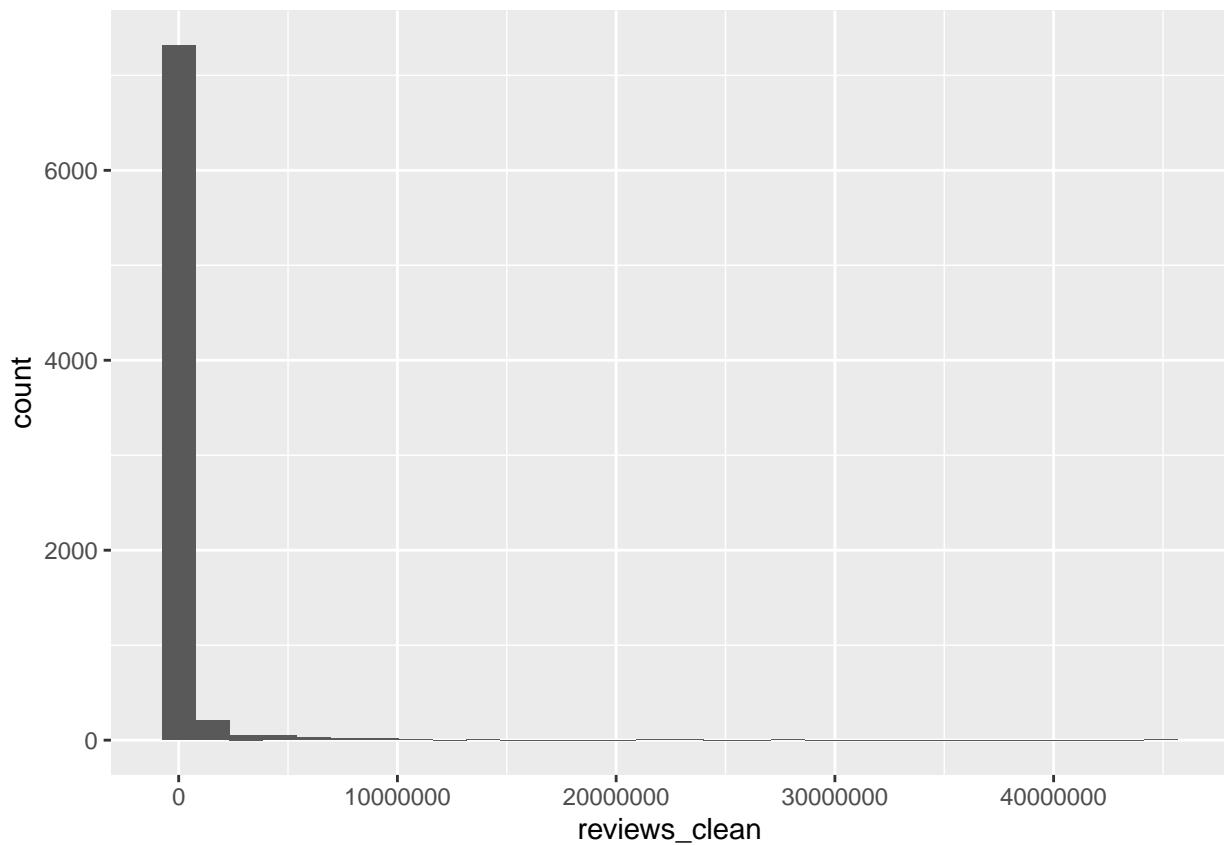
```
ggplot(data = d, aes(x = install_group)) +  
  geom_histogram()  
  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



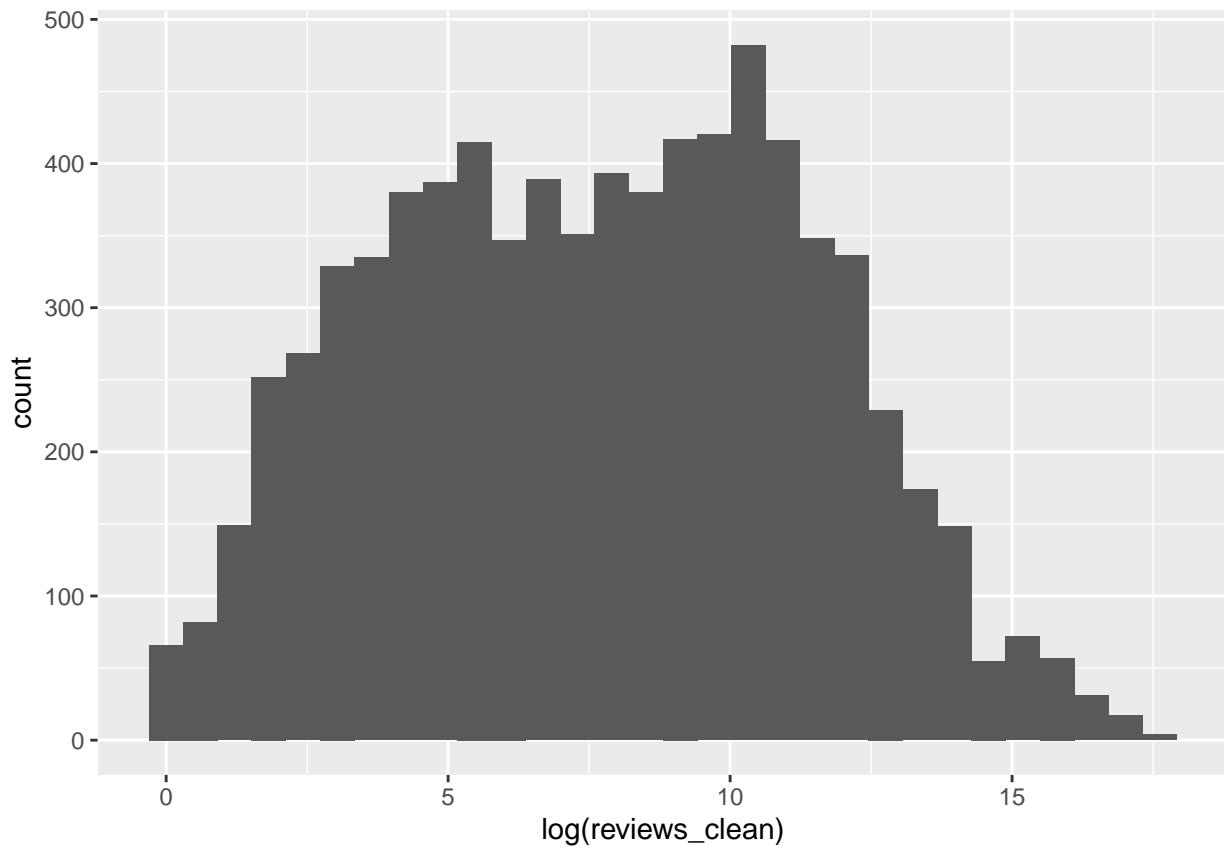
```
ggplot(data = d, aes(x = log(install_group))) +  
  geom_histogram()  
  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



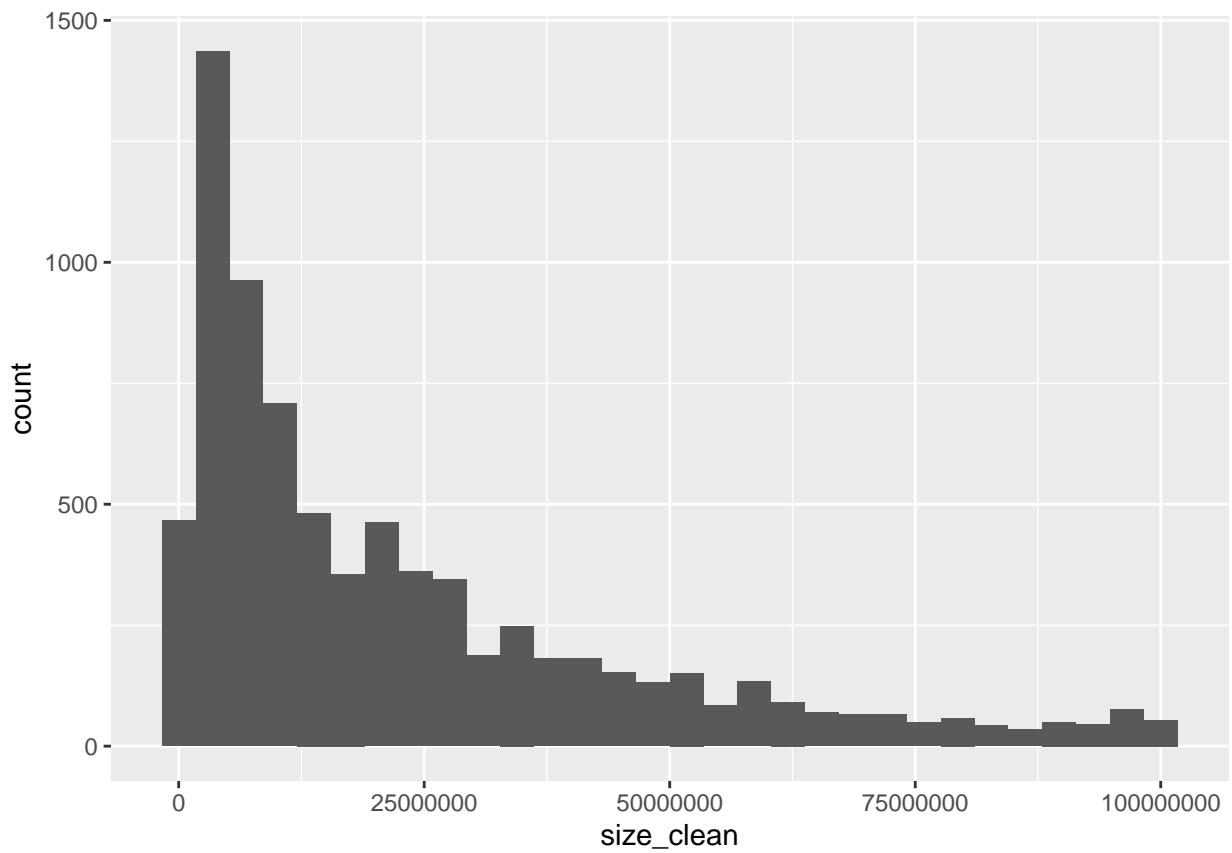
```
ggplot(data = d, aes(x = reviews_clean)) +  
  geom_histogram()  
  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



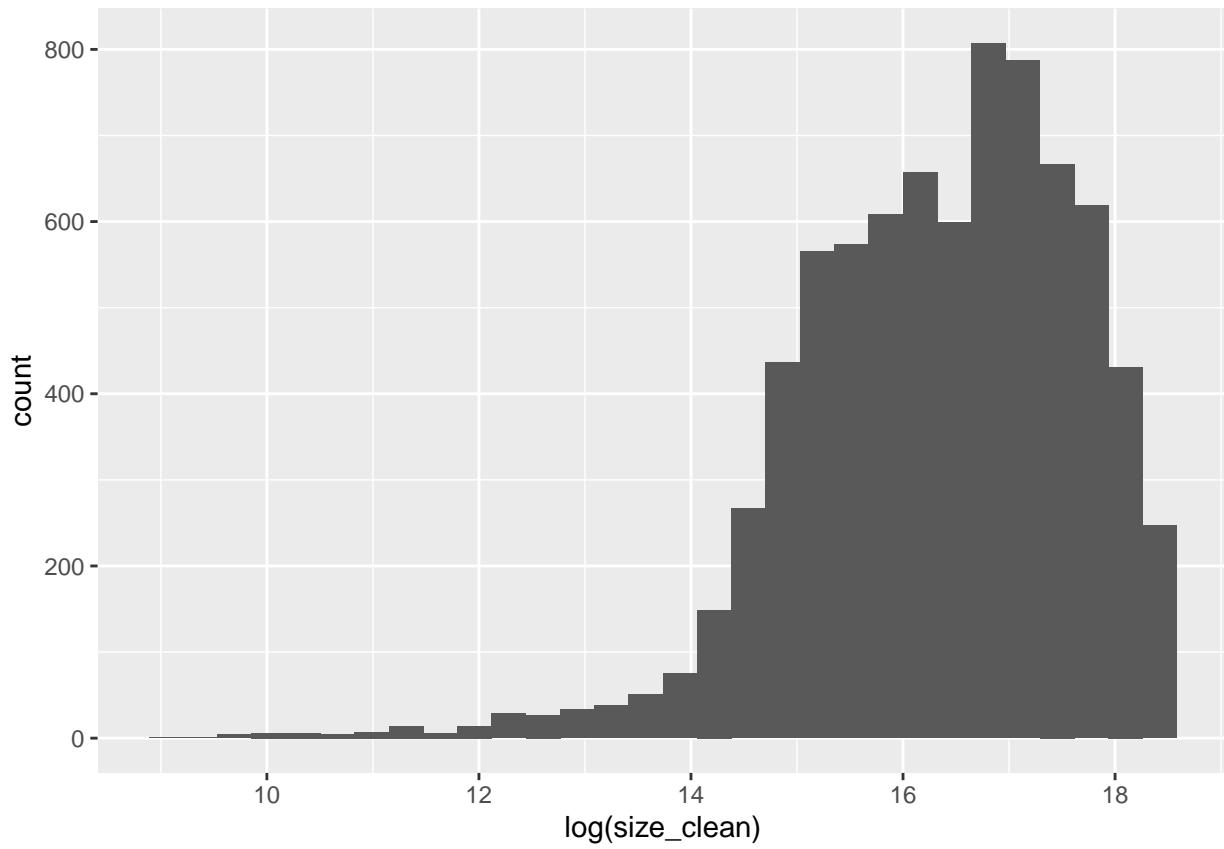
```
ggplot(data = d, aes(x = log(reviews_clean))) +  
  geom_histogram()  
  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



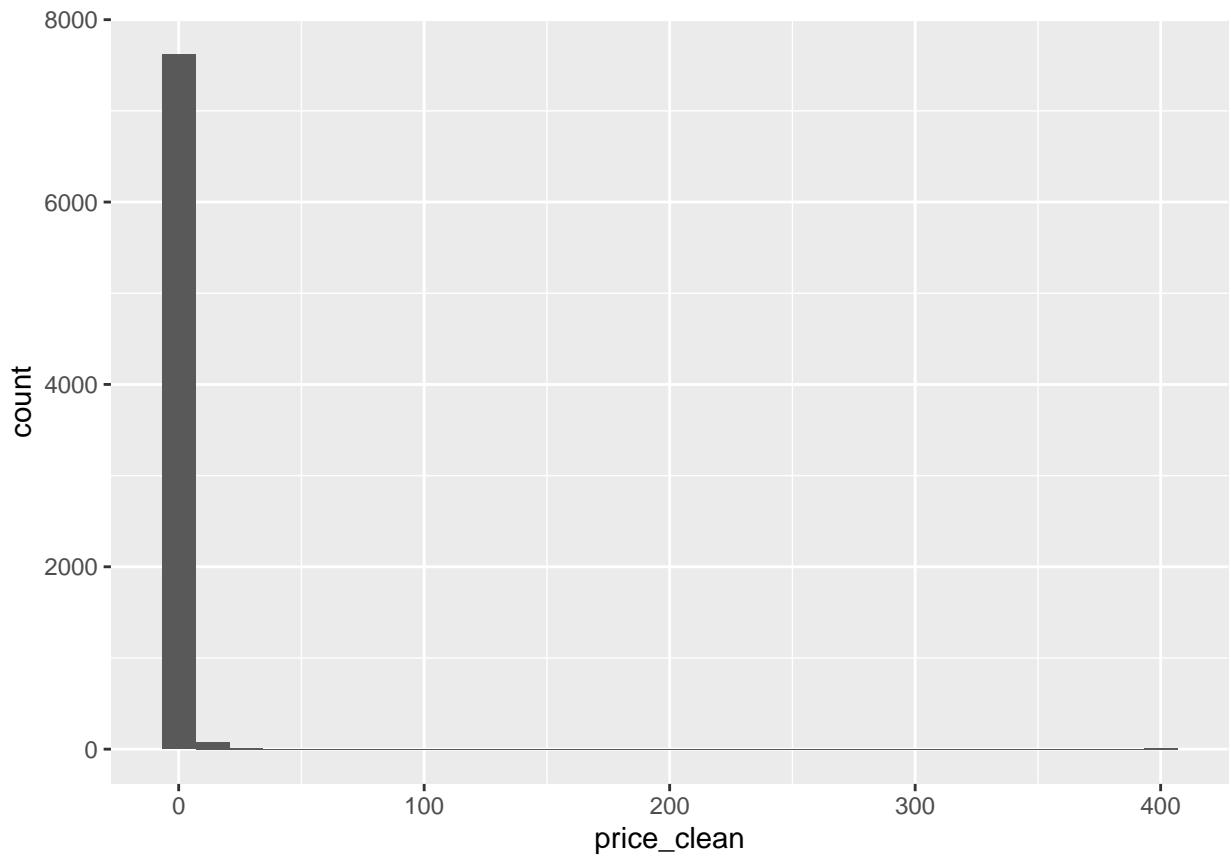
```
ggplot(data = d, aes(x = size_clean)) +  
  geom_histogram()  
  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



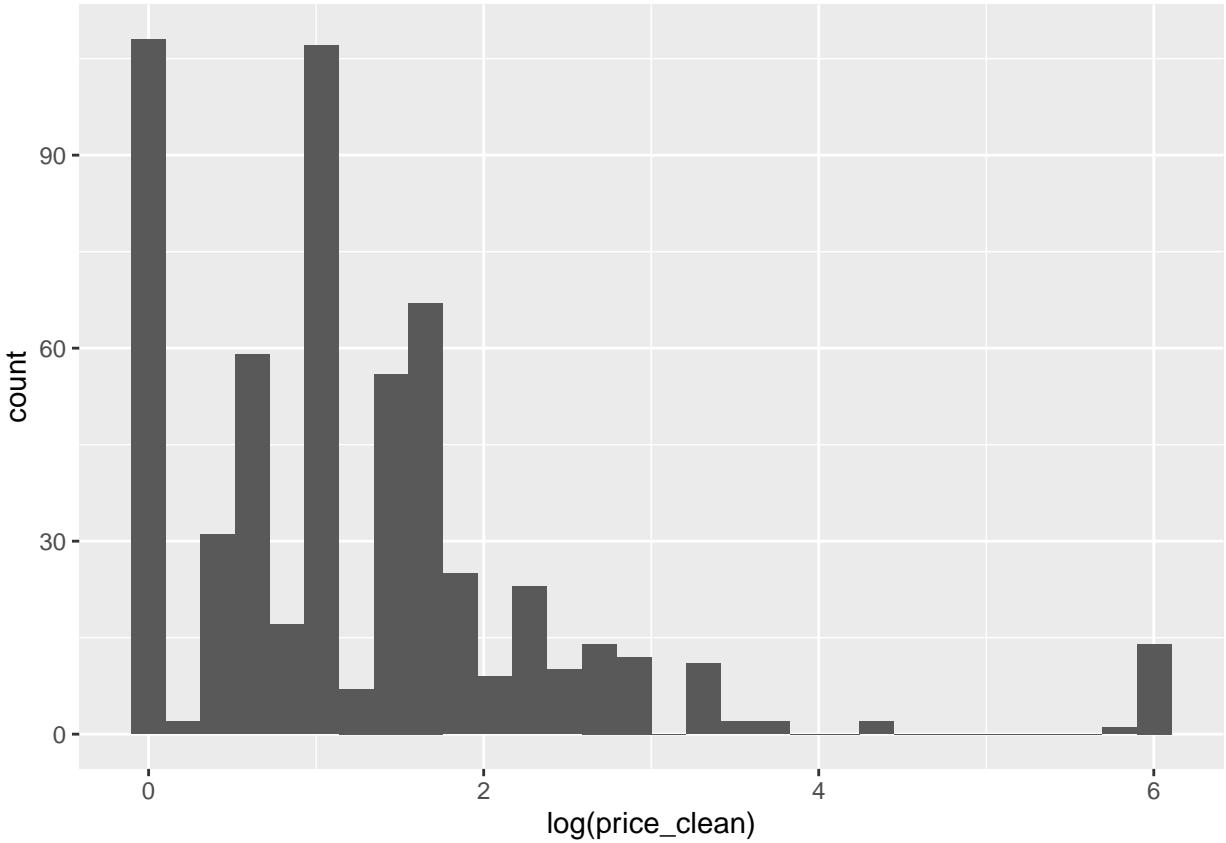
```
ggplot(data = d, aes(x = log(size_clean))) +  
  geom_histogram()  
  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
ggplot(data = d, aes(x = price_clean)) +  
  geom_histogram()  
  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
ggplot(data = d, aes(x = log(price_clean))) +  
  geom_histogram()  
  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.  
## Warning: Removed 7150 rows containing non-finite values (stat_bin).
```



```

categorical_cols <- c(
  'Category',
  'Type',
  'Content.Rating',
  'Current.Ver',
  'Android.Ver'
)

calc_stats_by_group <-
  function(column, quantile_table = FALSE) {

    # calc summary statistics per unique element in categorical column
    result <- d_dt[, .(count_apps = .N,
                      install_group_avg = mean(install_group, na.rm = TRUE),
                      rating_avg = mean(Rating, na.rm = TRUE)),
                  by = column][count_apps > 100][order(-install_group_avg)]
    setnames(result, column, 'group_by_column')
    result[, variable := column]

    # create a table that shows the distribution across group averages
    if (quantile_table) {
      result <- as.data.table(t(data.table(quantile(
        result$install_group_avg
      ))))
      setnames(result, c('0%', '25%', '50%', '75%', '100%'))
      result[, variable := column]
    }
  }

```

```

        result[, diff_min_vs_max := `100%` - `0%`]
    }
    return(result)
}

# perform function across all categorical columns
table_long_cat <- rbindlist(lapply(categorical_cols, calc_stats_by_group))
table_quantile_cat <- rbindlist(lapply(categorical_cols, calc_stats_by_group, quantile_table=TRUE))

# compare avg install grp across variable values
table_long_cat

```

	group_by_column	count_apps	install_group_avg	rating_avg	variable
## 1:	GAME	974	12.975359	4.269507	Category
## 2:	EDUCATION	110	12.627273	4.387273	Category
## 3:	PHOTOGRAPHY	236	12.389831	4.147034	Category
## 4:	SHOPPING	179	12.307263	4.227933	Category
## 5:	VIDEO_PLAYERS	116	11.655172	4.025862	Category
## 6:	SPORTS	247	11.404858	4.204858	Category
## 7:	COMMUNICATION	211	11.350711	4.102844	Category
## 8:	SOCIAL	177	11.203390	4.257062	Category
## 9:	TRAVEL_AND_LOCAL	160	11.181250	4.043125	Category
## 10:	HEALTH_AND_FITNESS	223	11.103139	4.223767	Category
## 11:	PRODUCTIVITY	235	10.872340	4.143830	Category
## 12:	FAMILY	1617	10.679654	4.190229	Category
## 13:	NEWS_AND_MAGAZINES	169	10.656805	4.143787	Category
## 14:	TOOLS	634	10.517350	4.011041	Category
## 15:	DATING	173	10.352601	3.957803	Category
## 16:	PERSONALIZATION	280	10.275000	4.324286	Category
## 17:	BOOKS_AND_REFERENCE	144	10.159722	4.320139	Category
## 18:	FINANCE	266	10.026316	4.112030	Category
## 19:	LIFESTYLE	280	9.914286	4.093571	Category
## 20:	BUSINESS	246	9.394309	4.119919	Category
## 21:	MEDICAL	324	8.521605	4.184259	Category
## 22:	Free	7150	11.243636	4.167469	Type
## 23:	Paid	579	7.981002	4.252677	Type
## 24:	Everyone	318	12.764151	4.261006	Content.Rating
## 25:	Teen	868	11.995392	4.227074	Content.Rating
## 26:	Mature	368	11.334239	4.095380	Content.Rating
## 27:	10+	6172	10.747894	4.166445	Content.Rating
## 28:	Everyone	1.3	9.533333	4.209167	Current.Ver
## 29:	Teen	2.0	9.067797	4.138983	Current.Ver
## 30:	Mature	1.2	8.714286	4.115873	Current.Ver
## 31:	10+	1.1	8.446154	4.164615	Current.Ver
## 32:	Everyone	1.0	7.967249	4.133843	Current.Ver
## 33:	1.3	458	4.248083	Android.Ver	
## 34:	2.0	1930	11.427979	4.4.4 and up	
## 35:	2.3	806	11.330025	4.5.0 and up	
## 36:	2.6	490	11.038776	4.4.0 and up	
## 37:	2.9	566	11.025248	4.5.3 and up	
## 38:	3.2	318	11.014134	4.4.2 and up	
## 39:	3.5	1194	10.933962	4.4.0.3 and up	
## 40:	3.8	113	10.619469	4.2.1 and up	
## 41:	4.1	195	10.600000	4.2.3 and up	

```

## 42:      2.3.3 and up      235      10.510638  4.052766  Android.Ver
## 43:      3.0 and up       211      10.071090  4.125118  Android.Ver
## 44:      2.2 and up       206      9.325243  3.985437  Android.Ver
##   group_by_column count_apps install_group_avg rating_avg      variable
# compare distribution of avg install grp across variable values
table_quantile_cat

##          0%        25%        50%        75%        100%      variable
## 1: 8.521605 10.275000 10.872340 11.404858 12.975359      Category
## 2: 7.981002 8.796660 9.612319 10.427978 11.243636      Type
## 3: 10.747894 11.187653 11.664815 12.187582 12.764151 Content.Rating
## 4: 7.967249 8.446154 8.714286 9.067797 9.533333  Current.Ver
## 5: 9.325243 10.577660 10.921755 11.028630 11.427979  Android.Ver
##   diff_min_vs_max
## 1:        4.453754
## 2:        3.262635
## 3:        2.016257
## 4:        1.566084
## 5:        2.102737

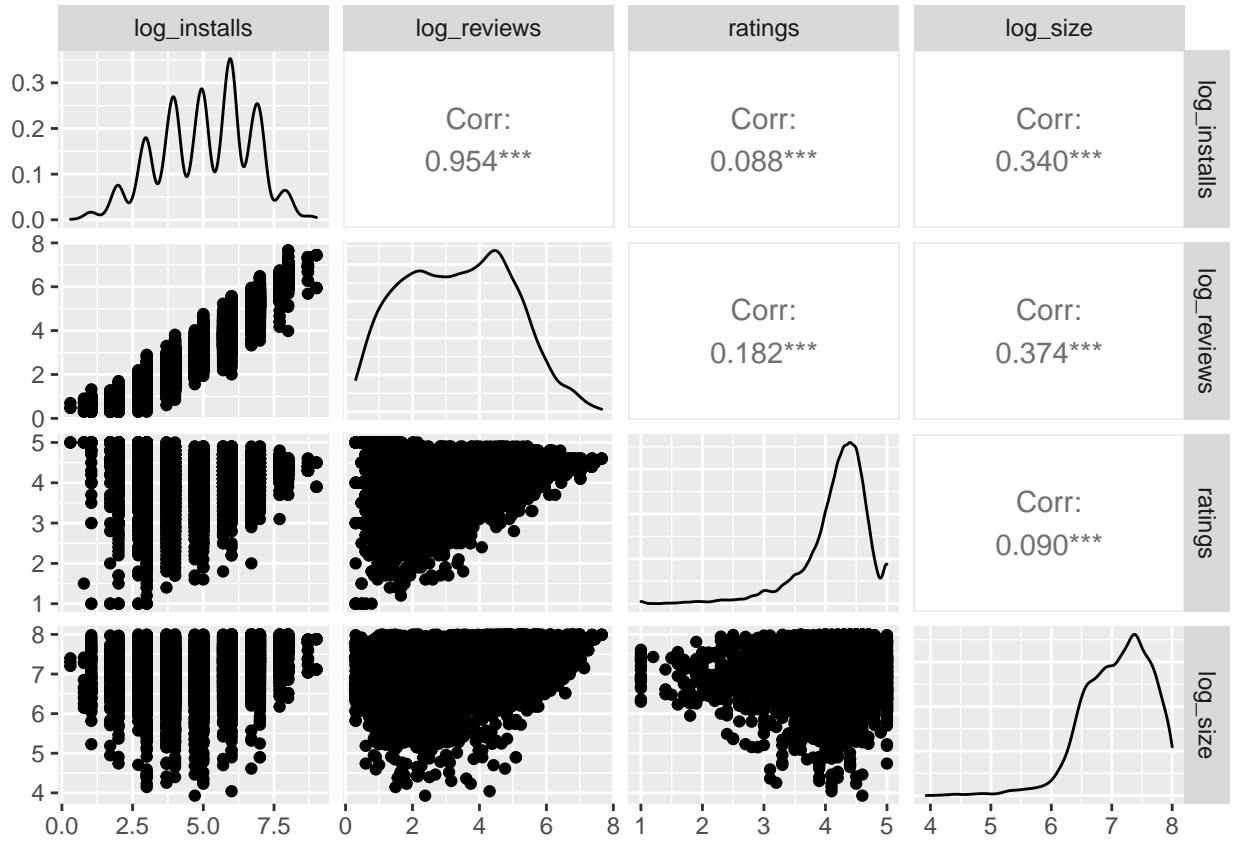
```

I.I.D. data: According to the Kaggle authors, this data set was collected by randomly scraping the Google Play Store. Since no clusters of applications were specifically targeted, we can reasonably use the entirety of the store as our reference population. We recognize that applications likely have some degree of interdependence, especially within genres. For example, the success of one application probably has a negative impact on other applications of the same type. Due to the large size of this data set (7729 records), however, we expect any dependencies to be negligible. We also have reason to believe that the data are identically distributed, as they are drawn from the same population of applications. One could argue that since the Google Play Store changes over time, the distribution also shifts in response. Because the authors do not mention the time frame across which the data was collected, we will assume that they originated from a single snapshot of the Play Store and that no shifts in the underlying distribution occurred.

```

d$log_installs <- log10(d$installs_clean + 1)
d$log_reviews <- log10(d$reviews_clean + 1)
d$ratings <- d$Rating
d$log_size <- log10(d$size_clean + 1)
cols <- c('log_installs', 'log_reviews', 'ratings', 'log_size')
ggpairs(d[, cols])

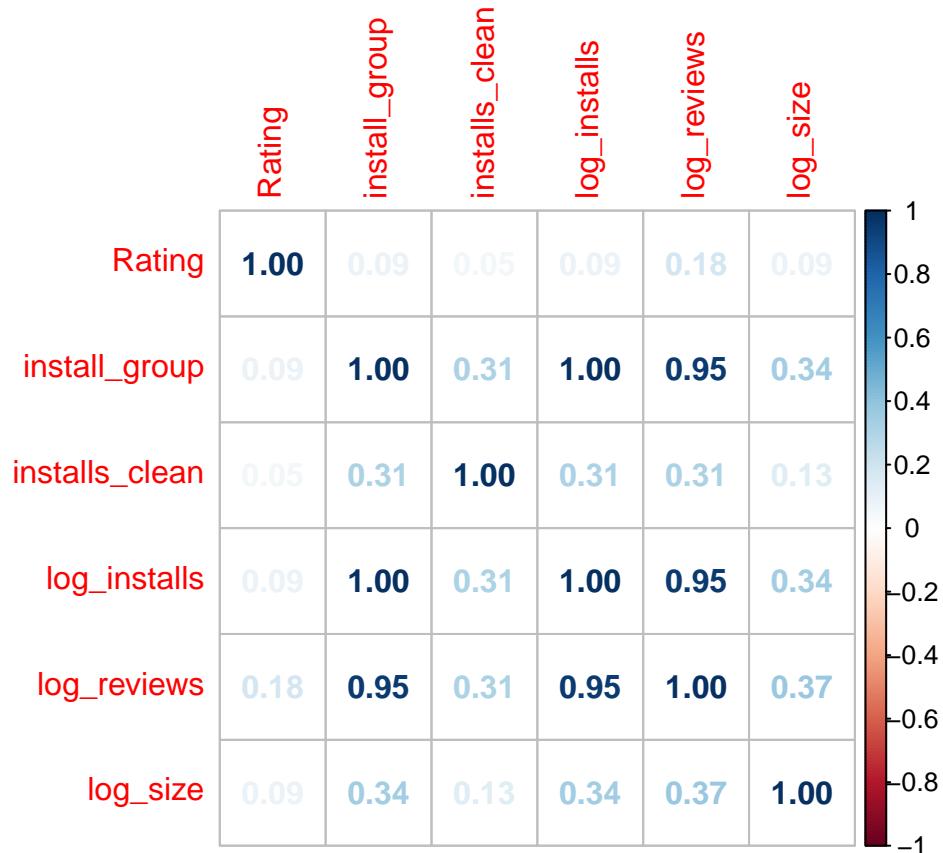
```



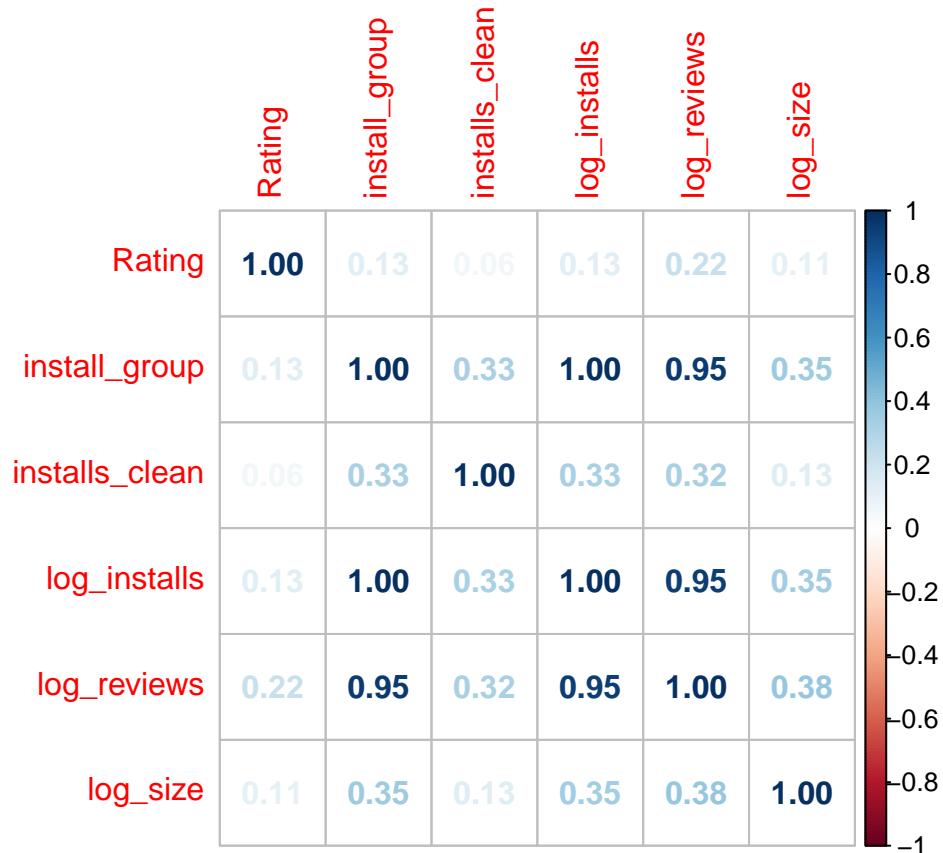
```
# save a data.table version for some easier wrangling downstream
d_dt <- as.data.table(d)

cols <- c('Rating','install_group','installs_clean', 'log_installs', 'log_reviews', 'log_size')

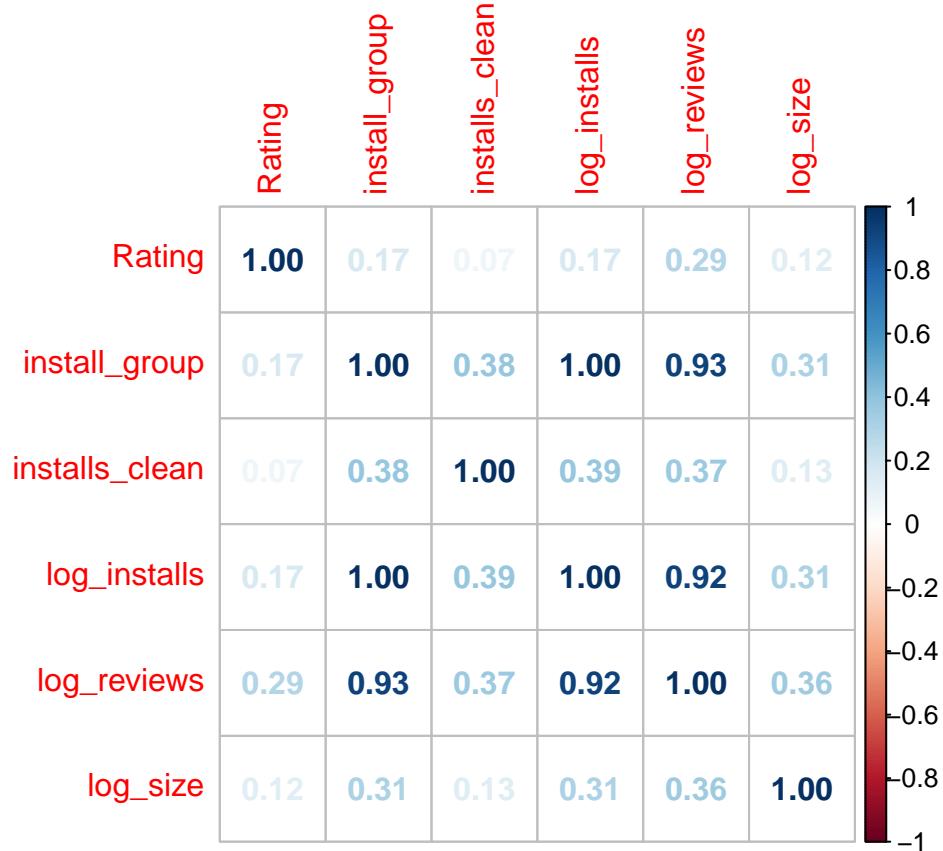
# Corrplot across variables
corrplot(cor(d[, cols], use = "complete.obs"),
         method = 'number')
```



```
# Corrplot across variables (5th PCTL outliers removed)
corrplot(cor(d[d$reviews_clean >= 6,cols], use = "complete.obs"),
         method = 'number')
```



```
# Corrplot across variables (25th PCTL outliers removed)
corrplot(cor(d[d$reviews_clean >= 100,cols], use = "complete.obs"),
         method = 'number')
```



```

model_small <- lm(log_installs ~ 1 + log_reviews, data = d)
model_medium <- lm(log_installs ~ 1 + log_reviews + ratings + log_size, data = d)
model_large <- lm(log_installs ~ 1 + log_reviews + ratings + log_size + factor(Type), data = d)

stargazer(
  model_small,
  model_medium,
  model_large,
  type = 'text',
  se = list(get_robust_se(model_small), get_robust_se(model_medium))
)

##
## -----
##                                     Dependent variable:
##                                     -----
##                                     log_installs
##                                     (1)          (2)          (3)
## -----
## ## log_reviews           0.955***      0.977***      0.959***  

## ##                         (0.004)        (0.004)        (0.003)  

## ##  

## ## ratings              -0.261***     -0.239***     -0.239***  

## ##                         (0.013)        (0.009)  

## ##  

## ## log_size             -0.048***     -0.053***     -0.053***  

## ##                         (0.004)        (0.003)

```

```

##                                     (0.010)          (0.009)
## factor(Type)Paid                  -0.611***      (0.019)
##                                     (0.015)          (0.083)
## Constant                         1.860***      3.213***      3.264***      (0.073)
##                                     (0.019)          (0.073)
## -----
## Observations                      7,729          7,729          7,729
## R2                                0.909          0.917          0.927
## Adjusted R2                        0.909          0.917          0.927
## Residual Std. Error               0.484 (df = 7727)   0.462 (df = 7725)   0.435 (df = 7724)
## F Statistic                       77,422.710*** (df = 1; 7727) 28,504.560*** (df = 3; 7725) 24,455.410*** (df = 4;
## Note:                                         *p<0.1; **p<0.05; ***p<0.01

```

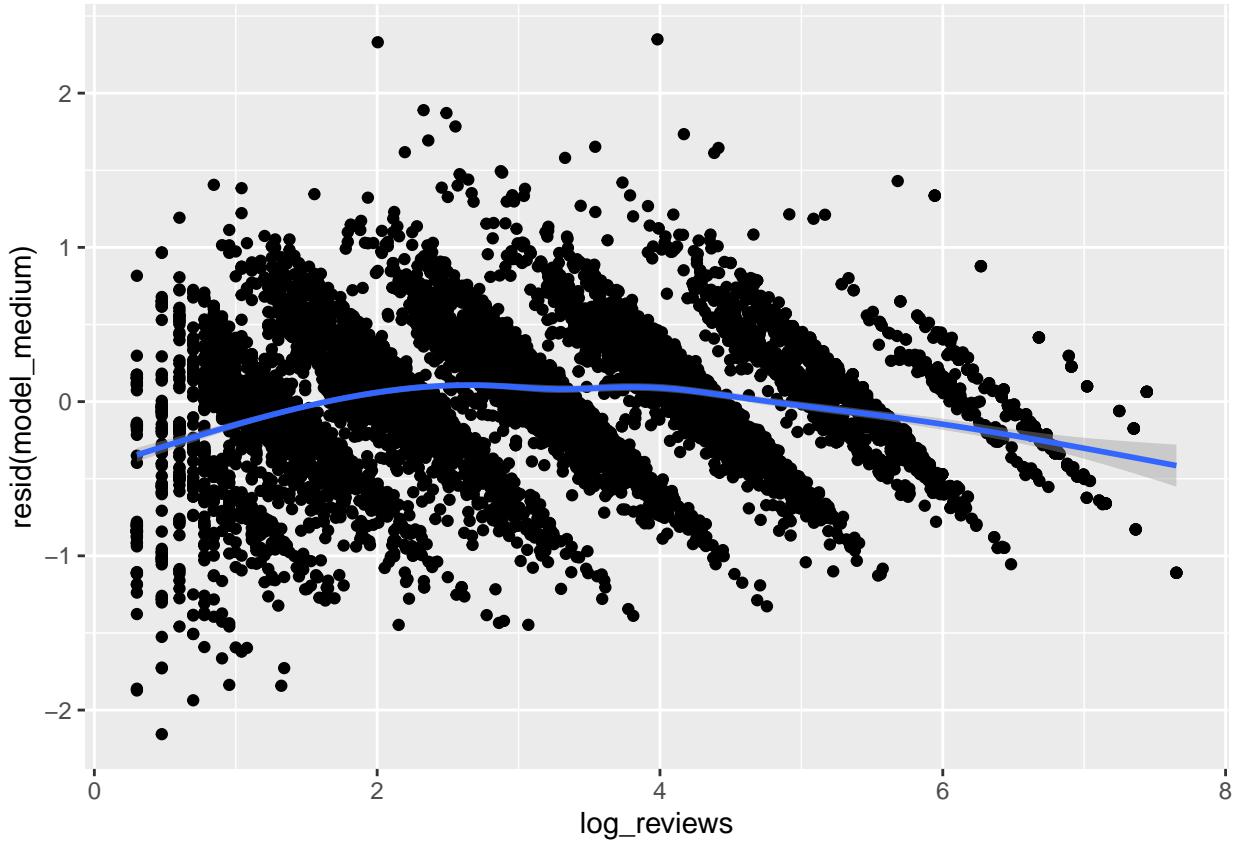
2. **No Perfect Colinearity:** We can immediately conclude that `log_installs`, `log_reviews`, `ratings`, and `log_size` are not perfectly colinear as otherwise the regression above would have failed. We can also assess near perfect colinearity for these variables by observing the robust standard errors returned by the regression model. In general, highly colinear features will have large standard errors. Since the standard error of the coefficients are small relative to their magnitude, we can reasonably conclude that they are not nearly colinear.
3. **Linear Conditional Expectation:** To verify the assumption of linear conditional expectations, we seek to show that there is no relationship between the model residuals and any of the predictor variables. That is, the model does not systematically underpredict or overpredict in certain regions of the input space. Plots 1 through 3 show the relationships between the model residuals and individual predictors. The residuals are generally well-centered around zero, although the model seems to underpredict when `log_reviews` is high and `ratings` is low. The fourth plot shows the model residuals as a function of the model predictions. Here, the model seems to underpredict in the left-most and right-most regions, and slightly overpredict in the middle. Overall, there are no strong non-linear relationships between the model residuals and the input features, and we do not find enough evidence to reject the assumption of linear conditional expectation.

```

# Reviews versus residuals
plot_1 <- ggplot(data = d, mapping = aes(x = log_reviews, y = resid(model_medium))) +
  geom_point() + stat_smooth()
plot_1

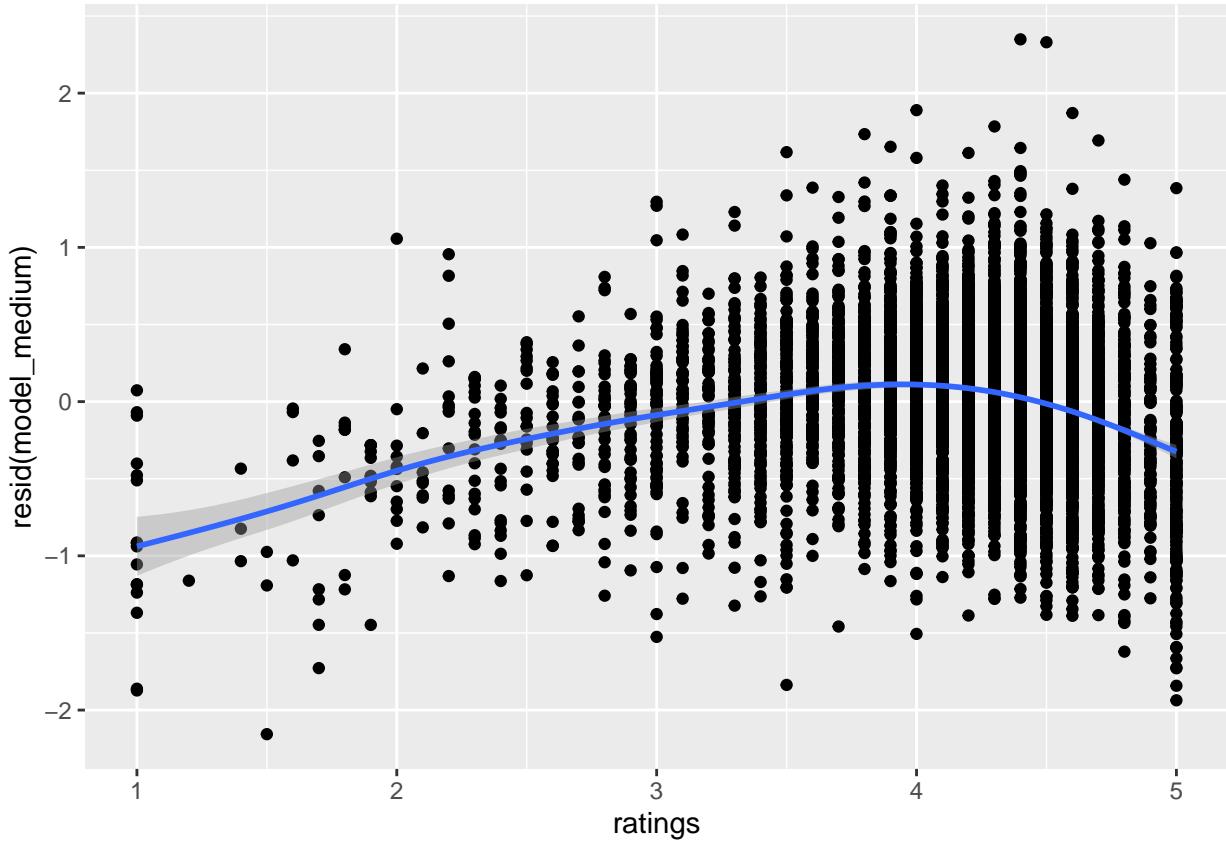
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'

```



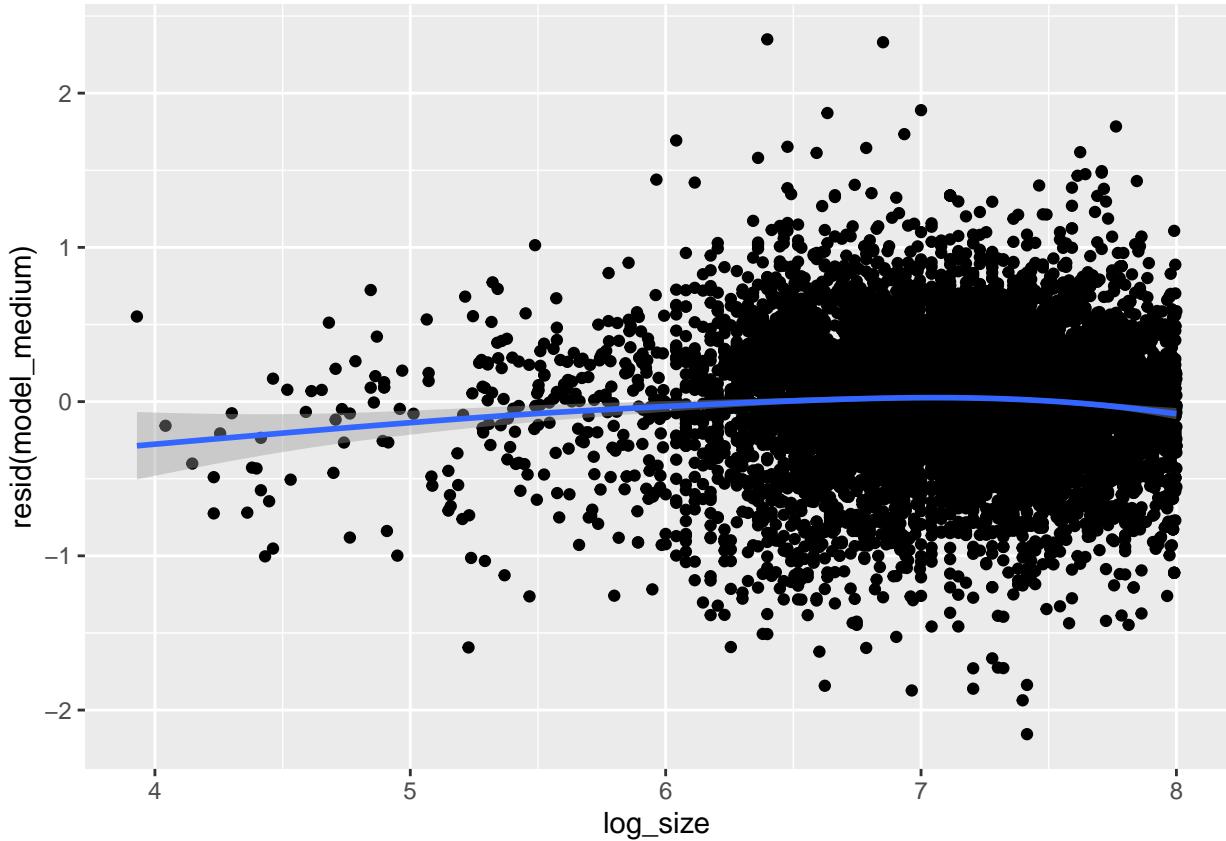
```
# Ratings versus residuals
plot_2 <- ggplot(data = d, mapping = aes(x = ratings, y = resid(model_medium))) +
  geom_point() + stat_smooth()
plot_2

## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



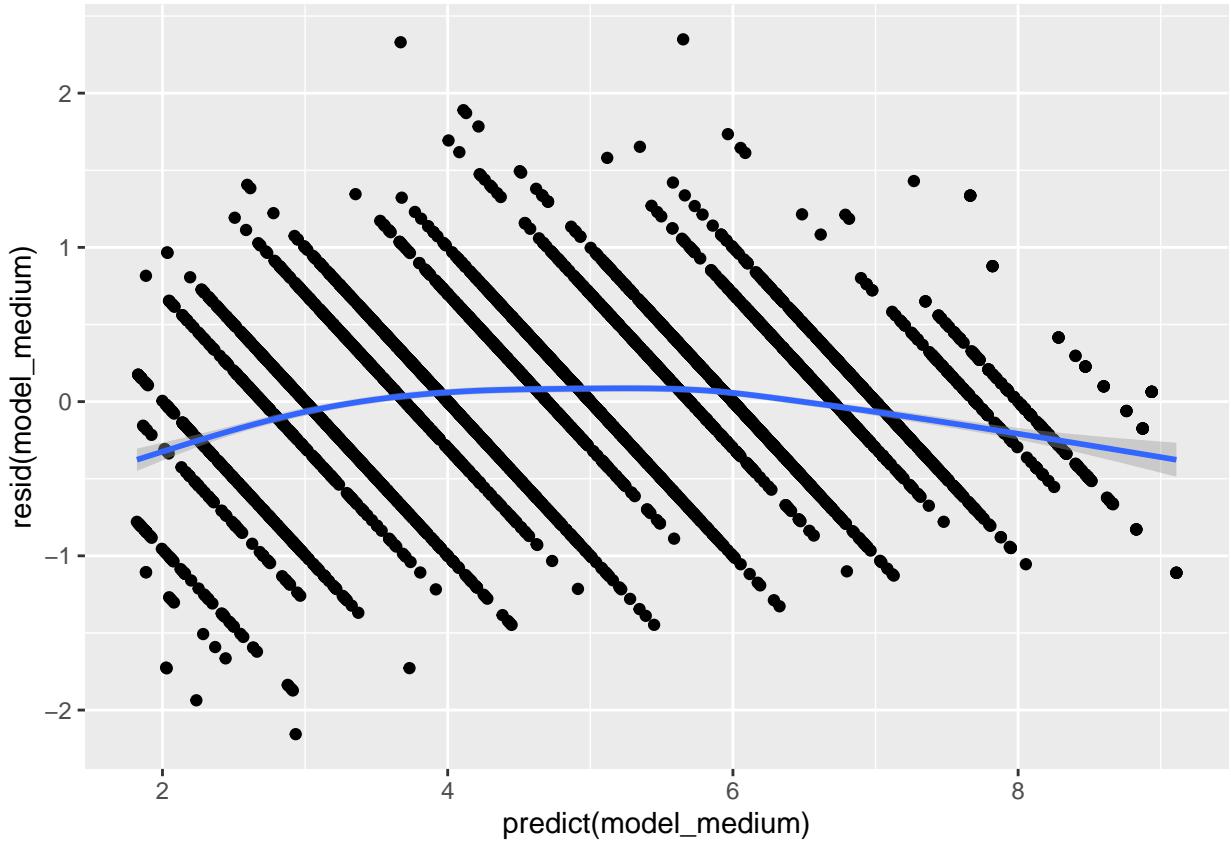
```
# Size versus residuals
plot_3 <- ggplot(data = d, mapping = aes(x = log_size, y = resid(model_medium))) +
  geom_point() + stat_smooth()
plot_3

## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



```
# Model predictions versus residuals
plot_4 <- ggplot(data = d, mapping = aes(x = predict(model_medium), y = resid(model_medium))) +
  geom_point() + stat_smooth()
plot_4

## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

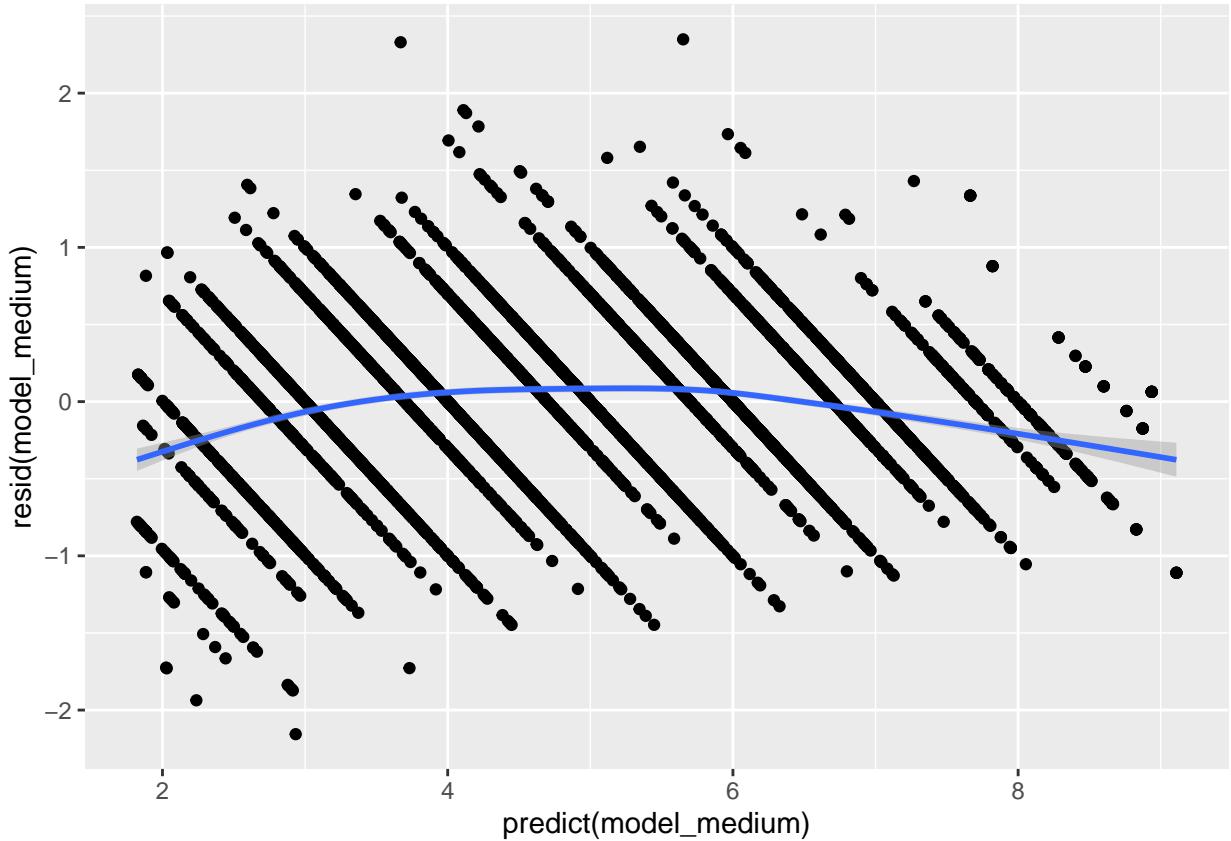


4. **Homoskedastic Errors:** When assessing homoskedastic errors, we seek to determine if there is a relationship between the variance of the model residuals and the predictors. If the homoskedastic assumption is satisfied, then we should observe a lack of relationship; conversely, if the data are heteroskedastic then the conditional variance will depend on the predictors. The first plot is an eyeball test of homoskedasticity, showing the model residuals as a function of the model predictions. We notice that the spread of the residuals is mostly consistent throughout the data, although the right-hand side is somewhat narrower. As a more concrete assessment, we also perform a Breusch-Pagan test with the null hypothesis that there are no heteroskedastic errors in the model. Since the p -value falls below our significance threshold of 0.001, we find enough evidence to reject the null hypothesis. In response to this failed assumption, we report robust standard errors (adjusted for heteroskedasticity) instead of non-adjusted errors.

```
# Breusch-Pagan test
bp_test <- bptest(model_small)
bp_test

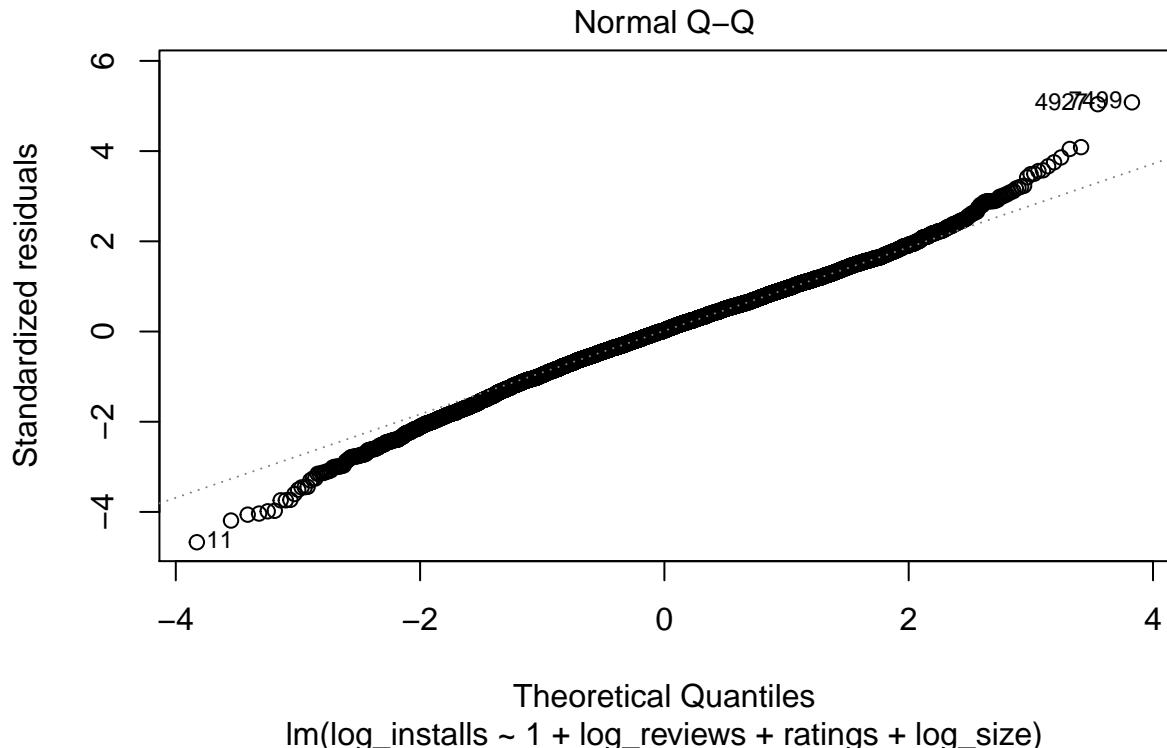
##
## studentized Breusch-Pagan test
##
## data: model_small
## BP = 235.66, df = 1, p-value < 0.0000000000000022
plot_4

## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



5. **Normally Distributed Errors:** When assessing the normality of the error distribution, we seek to determine if the model residuals are approximately Gaussian. If so, then the sample quantiles of the residuals should closely match the theoretical quantiles of a normal distribution in a Q-Q plot. Below, we plot the Q-Q plot associated with our model. In general, the residuals seem to follow a normal distribution, as the middle quantiles match the corresponding theoretical quantiles. However, the tails of the residual distribution are fatter than expected; the first quantiles occur at smaller than expected values, and the last quantiles occur at larger than expected values. Overall, the assumption of normally distributed errors seems imperfect but reasonably justified.

```
# Q-Q plot
plot_5 <- plot(model_medium, which = 2)
```



plot_5

NULL

** Reverse Causality: ** We have to consider the possibility that high average reviews could lead to a higher number of installations which could lead to a higher average review. We will want to test for a reverse causality relationship between these two variables to determine if the best linear predictor is valid. If we regress average reviews on installs, the installs coefficient (Gamma1) will have a positive slope. Since Beta1 (average review slope coefficient) > 0, we know higher average review leads to more installs. Since Gamma1 (installs slope coefficient for reverse causality) is > 0, this leads to positive feedback. Given we have two potentially positive coefficients, this could be a bias away from zero which is a concern that a reverse causality relationship exists between the two variables. We could consider dropping average reviews as a variable and determine if there are other leading variables that can explain the number of installs for an app.

```

model_small <- lm(log_installs ~ 1 + log_reviews, data = d)
model_reverse <- lm(log_reviews ~ 1 + log_installs, data = d)

stargazer(
  model_small,
  model_reverse,
  type = 'text',
  se = list(get_robust_se(model_small), get_robust_se(model_medium))
)

##
## =====
##                               Dependent variable:
```

```

## -----
##          log_installs   log_reviews
##                  (1)           (2)
## -----
## log_reviews          0.955*** 
##                      (0.004)
##
## log_installs          0.952
##
## 
## Constant            1.860***   -1.467*** 
##                      (0.015)     (0.083)
## 
## -----
## Observations        7,729       7,729
## R2                  0.909       0.909
## Adjusted R2         0.909       0.909
## Residual Std. Error (df = 7727) 0.484       0.483
## F Statistic (df = 1; 7727)    77,422.710*** 77,422.710*** 
## =====
## Note:               *p<0.1; **p<0.05; ***p<0.01

```