eda

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

install.packages('GGally', repos='https://ftp.osuosl.org/pub/cran/')
install.packages('moments', repos='https://ftp.osuosl.org/pub/cran/')
install.packages("corrplot", repos = "http://cran.us.r-project.org")
```

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

# various functions for wrangling
source('./functions/get_robust_se.R')
source('./functions/get_clean_dataset.R')
source('./functions/eda_calculate_stats_by_group.R')
source('./functions/eda_build_quantile_table.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
##
##
##
##
##  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
##
##
##
##
##  Android.Ver
##  Length:10841
##  Class :character
##  Mode  :character
##
##
##
##
```

```
d <- get_clean_dataset(minimum_review_count = 0)

# summary of dataset
summary(d)
```

```
##     installs               size           reviews              rating
##  Min.   :         1   Min.   :  0.85   Min.   :       1   Min.   :1.000
##  1st Qu.:     10000   1st Qu.:  5.80   1st Qu.:      97   1st Qu.:4.000
##  Median :    100000   Median : 15.00   Median :    2039   Median :4.300
##  Mean   :   7646081   Mean   : 24.30   Mean   :  279761   Mean   :4.171
##  3rd Qu.:   1000000   3rd Qu.: 35.00   3rd Qu.:   35930   3rd Qu.:4.500
##  Max.   :1000000000   Max.   :100.00   Max.   :44893888   Max.   :5.000
##      price          is_free         last_updated     android_version
##  Min.   :  0.000   Mode :logical   Min.   :0.00000   Min.   :1.000
##  1st Qu.:  0.000   FALSE:531       1st Qu.:0.06301   1st Qu.:4.000
##  Median :  0.000   TRUE :6695      Median :0.25343   Median :4.000
##  Mean   :  1.139                   Mean   :0.79610   Mean   :3.837
##  3rd Qu.:  0.000                   3rd Qu.:1.06575   3rd Qu.:4.100
##  Max.   :400.000                   Max.   :8.21644   Max.   :8.000
##  current_version     category         is_family_category is_game_category
##  Min.   :  0.000   Length:7226        Mode :logical      Mode :logical
##  1st Qu.:  1.100   Class :character   FALSE:5655         FALSE:6292
##  Median :  2.000   Mode  :character   TRUE :1571         TRUE :934
##  Mean   :  5.241
##  3rd Qu.:  3.700
##  Max.   :858.000
##  is_tools_category    genre           content_rating     is_content_everyone
##  Mode :logical     Length:7226        Length:7226        Mode :logical
##  FALSE:6622        Class :character   Class :character   FALSE:1432
##  TRUE :604         Mode  :character   Mode  :character   TRUE :5794
##
##
##
##      type           install_group    log_installs      log_size
##  Length:7226        Min.   : 1.00   Min.   :0.000   Min.   :-0.07058
##  Class :character   1st Qu.: 9.00   1st Qu.:4.000   1st Qu.: 0.76343
##  Mode  :character   Median :11.00   Median :5.000   Median : 1.17609
##                     Mean   :10.92   Mean   :5.019   Mean   : 1.15074
##                     3rd Qu.:13.00   3rd Qu.:6.000   3rd Qu.: 1.54407
##                     Max.   :19.00   Max.   :9.000   Max.   : 2.00000
##    log_price       log_current_version log_last_updated   log_reviews
```

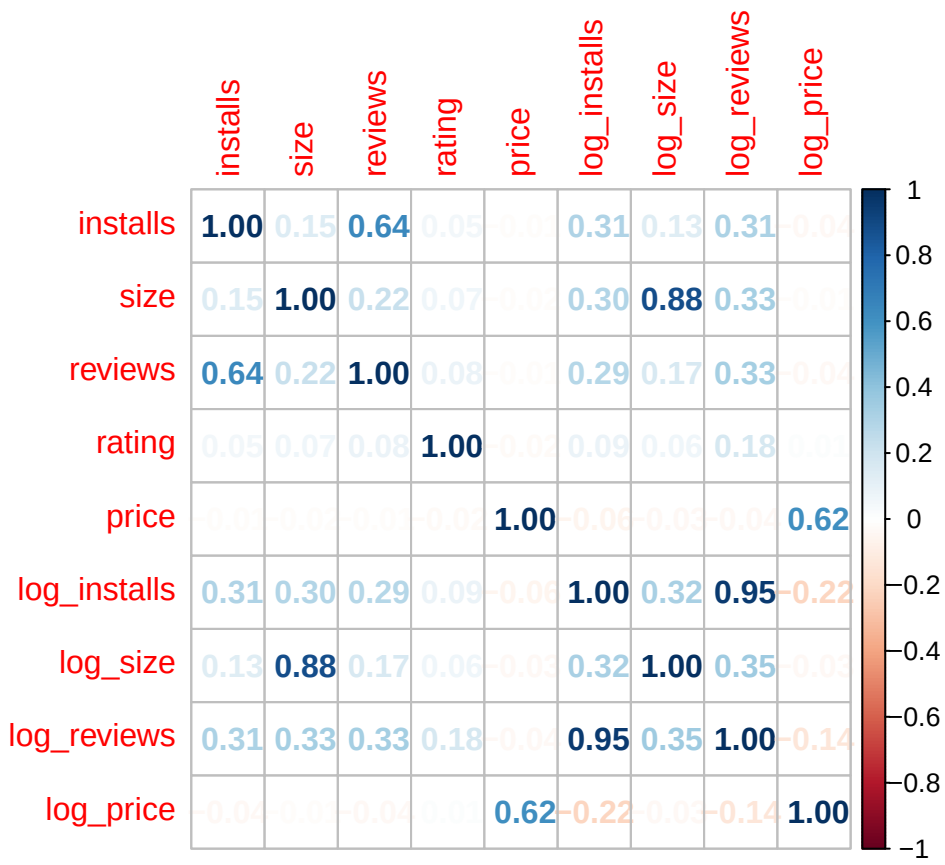
```
##  Min.   :0.00000   Min.   :0.0000      Min.   :0.00000   Min.   :0.000
##  1st Qu.:0.00000   1st Qu.:0.3222      1st Qu.:0.02654   1st Qu.:1.987
##  Median :0.00000   Median :0.4771      Median :0.09810   Median :3.309
##  Mean   :0.05045   Mean   :0.5417      Mean   :0.19366   Mean   :3.298
##  3rd Qu.:0.00000   3rd Qu.:0.6721      3rd Qu.:0.31508   3rd Qu.:4.555
##  Max.   :2.60314   Max.   :2.9340      Max.   :0.96456   Max.   :7.652
```

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

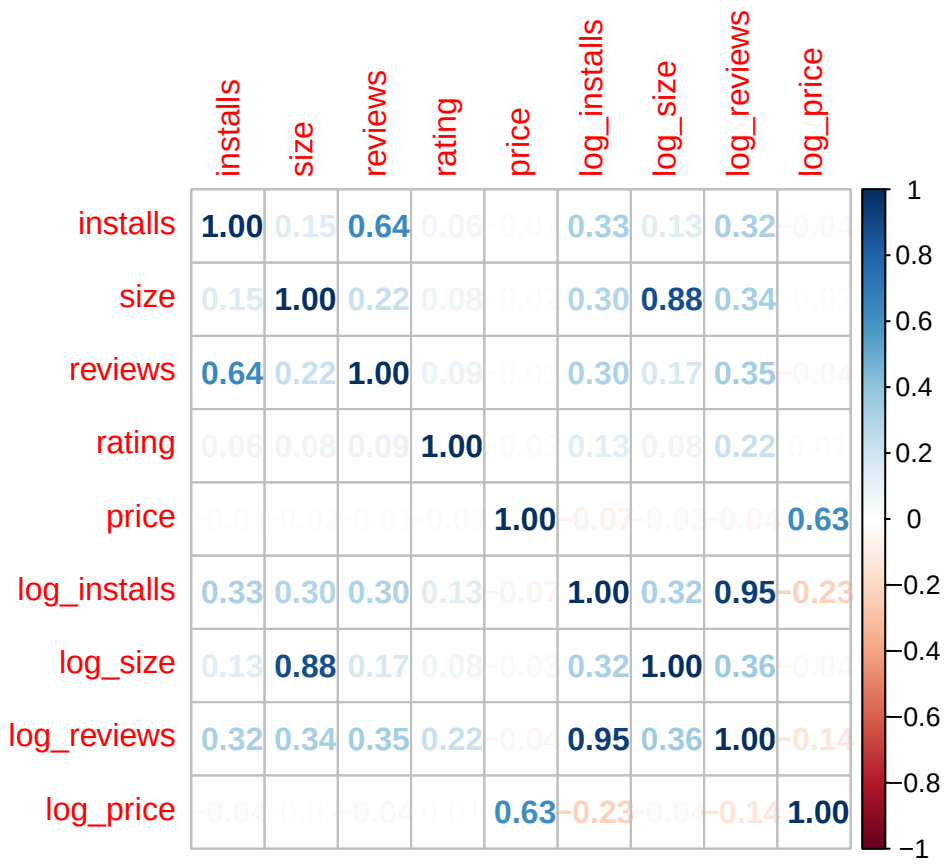
```
numeric_cols <- c(
  'installs',
  'size',
  'reviews',
  'rating',
  'price',
  'log_installs',
  'log_size',
  'log_reviews',
  'log_price'
)

table_quantile_numeric <- rbindlist(lapply(numeric_cols,eda_build_quantile_table))
table_quantile_numeric
```

```
##             0%          5%          25%           50%            75%
## 1:  1.00000000 100.0000000 10000.000000 100000.000000 1000000.000000
## 2:  0.85000000   2.2000000     5.800000     15.000000      35.000000
## 3:  1.00000000   6.0000000    97.000000   2039.000000   35930.250000
## 4:  1.00000000   3.1000000     4.000000      4.300000       4.500000
## 5:  0.00000000   0.0000000     0.000000      0.000000       0.000000
## 6:  0.00000000   2.0000000     4.000000      5.000000       6.000000
## 7: -0.07058107   0.3424227     0.763428      1.176091       1.544068
## 8:  0.00000000   0.7781513     1.986772      3.309417       4.555460
## 9:  0.00000000   0.0000000     0.000000      0.000000       0.000000
##                 95%              100%     variable  diff_min_vs_max
## 1: 10000000.0000000 1000000000.000000     installs 999999999.000000
## 2:       78.3500000        100.000000         size        99.150000
## 3:   837933.7500000   44893888.000000      reviews  44893887.000000
## 4:        4.8000000          5.000000       rating         4.000000
## 5:        1.9900000        400.000000        price       400.000000
## 6:        7.0000000          9.000000 log_installs         9.000000
## 7:        1.8940387          2.000000     log_size         2.070581
## 8:        5.9232082          7.652187  log_reviews         7.652187
## 9:        0.4756712          2.603144    log_price         2.603144
```

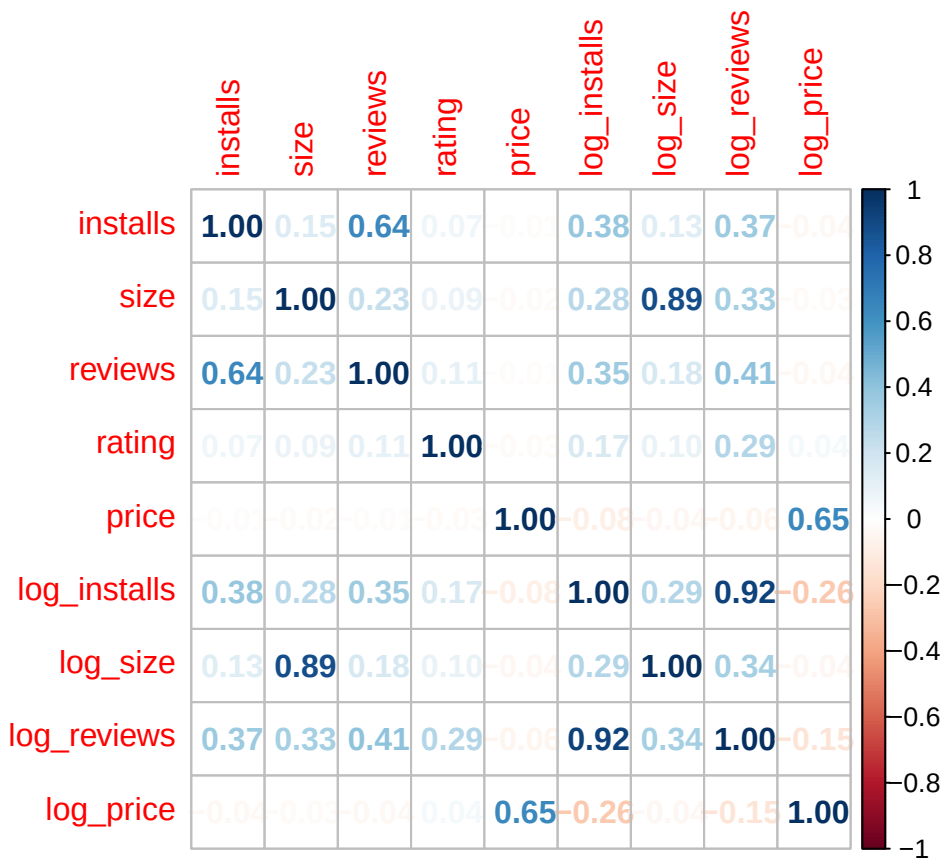
```
# Corrplot across variables
corrplot(cor(d[,numeric_cols], use = "complete.obs"),
         method = 'number')
```




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

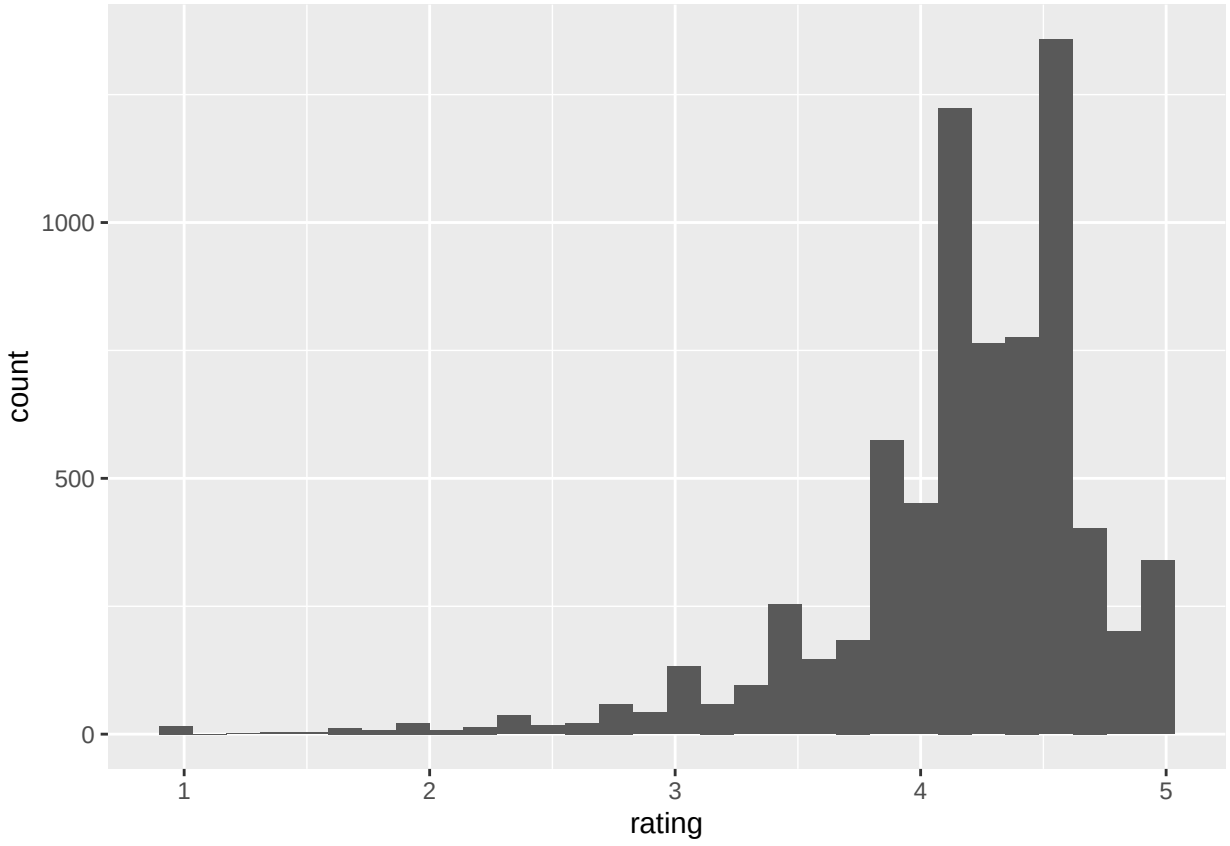



```
# Corrplot across variables (25th PCTL outliers removed)
corrplot(cor(d[d$reviews >= 100,numeric_cols], use = "complete.obs"),
         method = 'number')
```



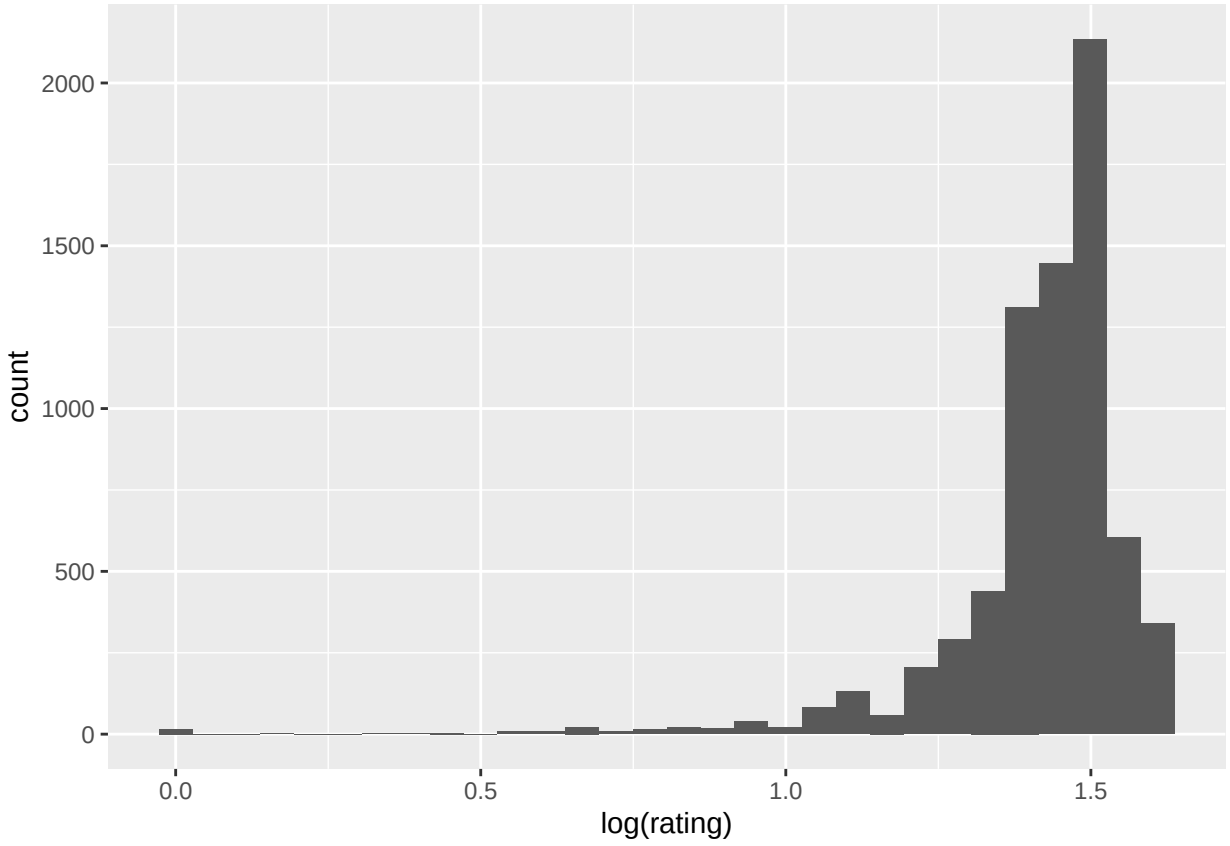

```
# Distribution of numeric columns
ggplot(data = d, aes(x = rating)) +
  geom_histogram()
```

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



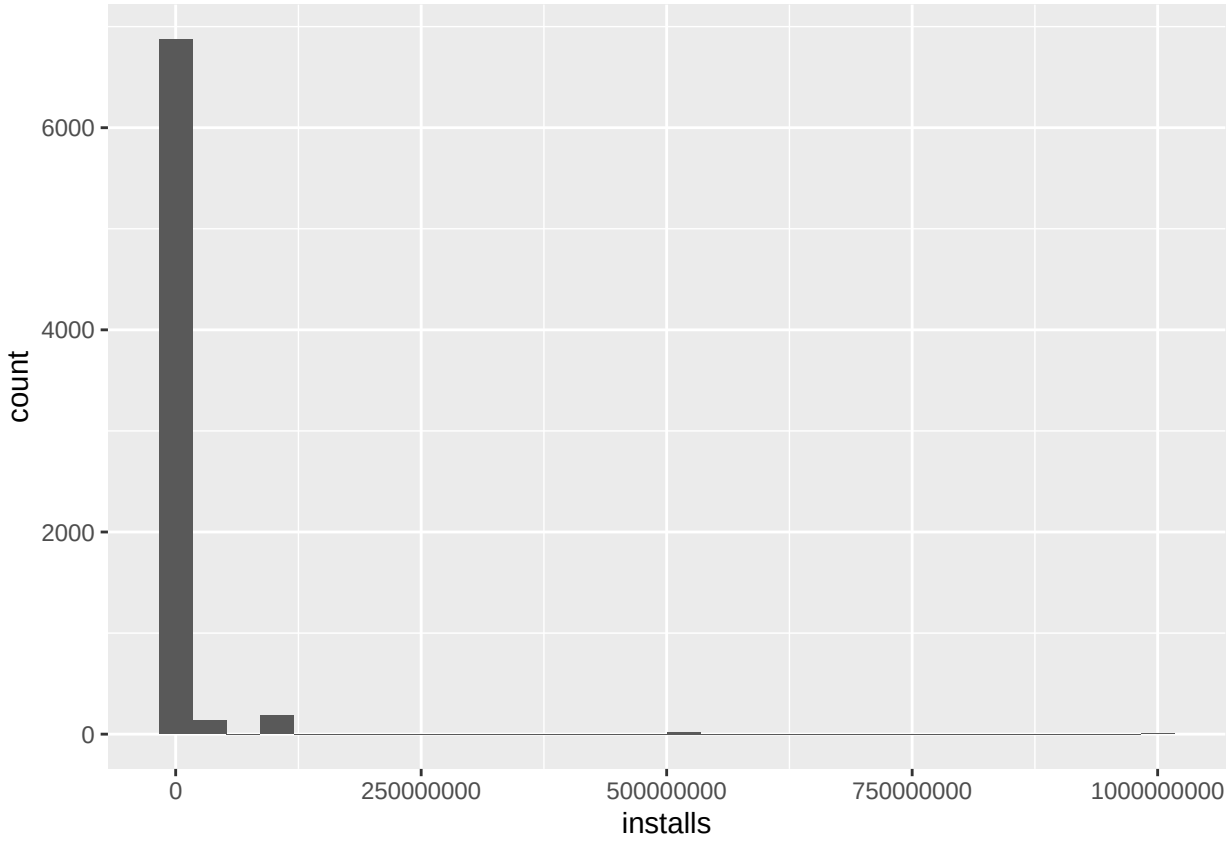

```
ggplot(data = d, aes(x = log(rating))) +
  geom_histogram()
```

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



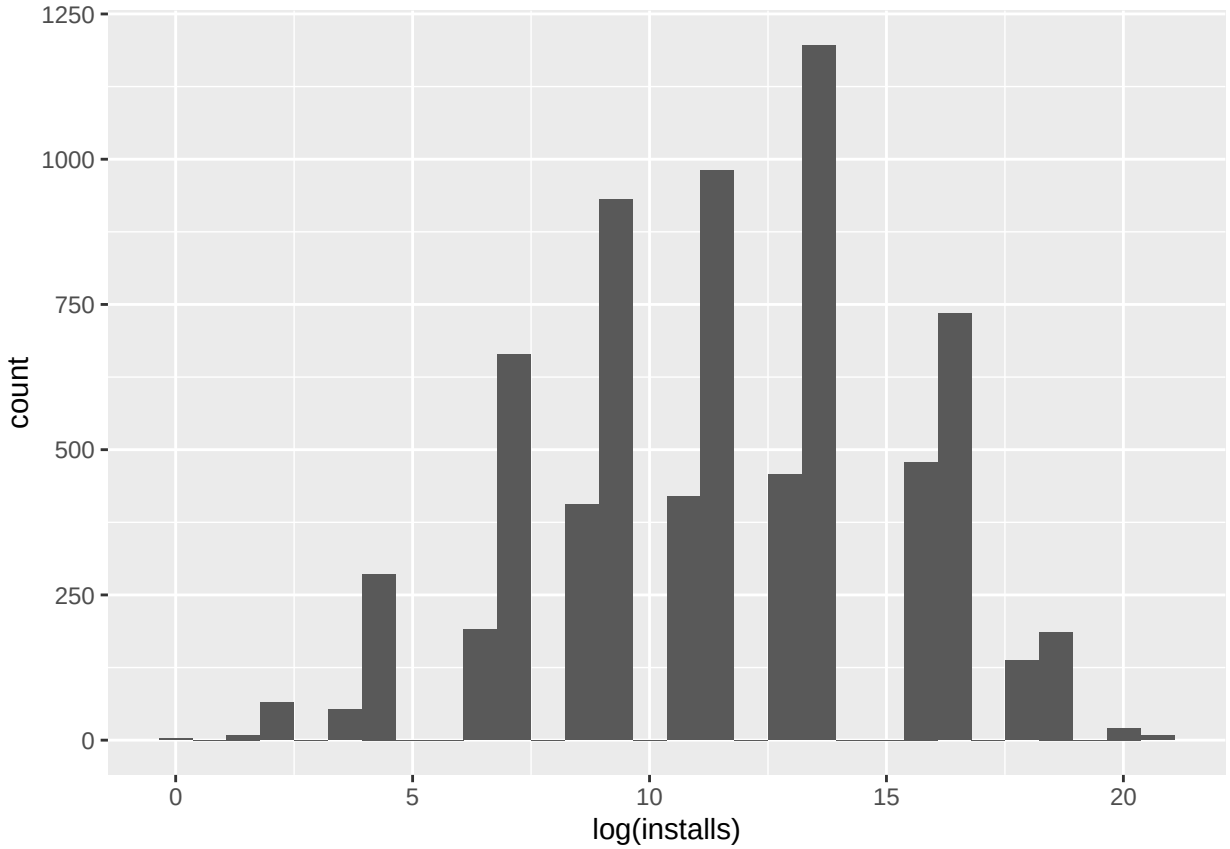

```
ggplot(data = d, aes(x = installs)) +
  geom_histogram()
```

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



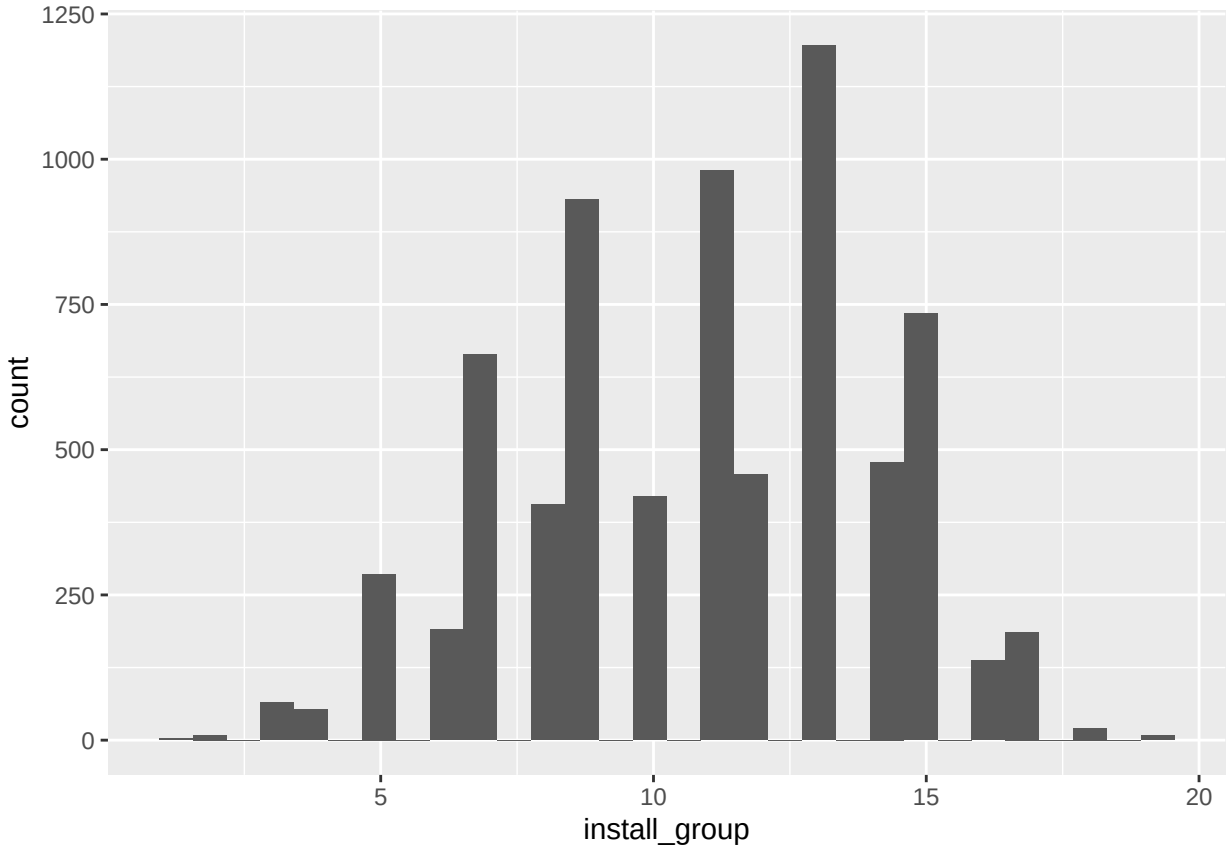

```
ggplot(data = d, aes(x = log(installs))) +
  geom_histogram()
```

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



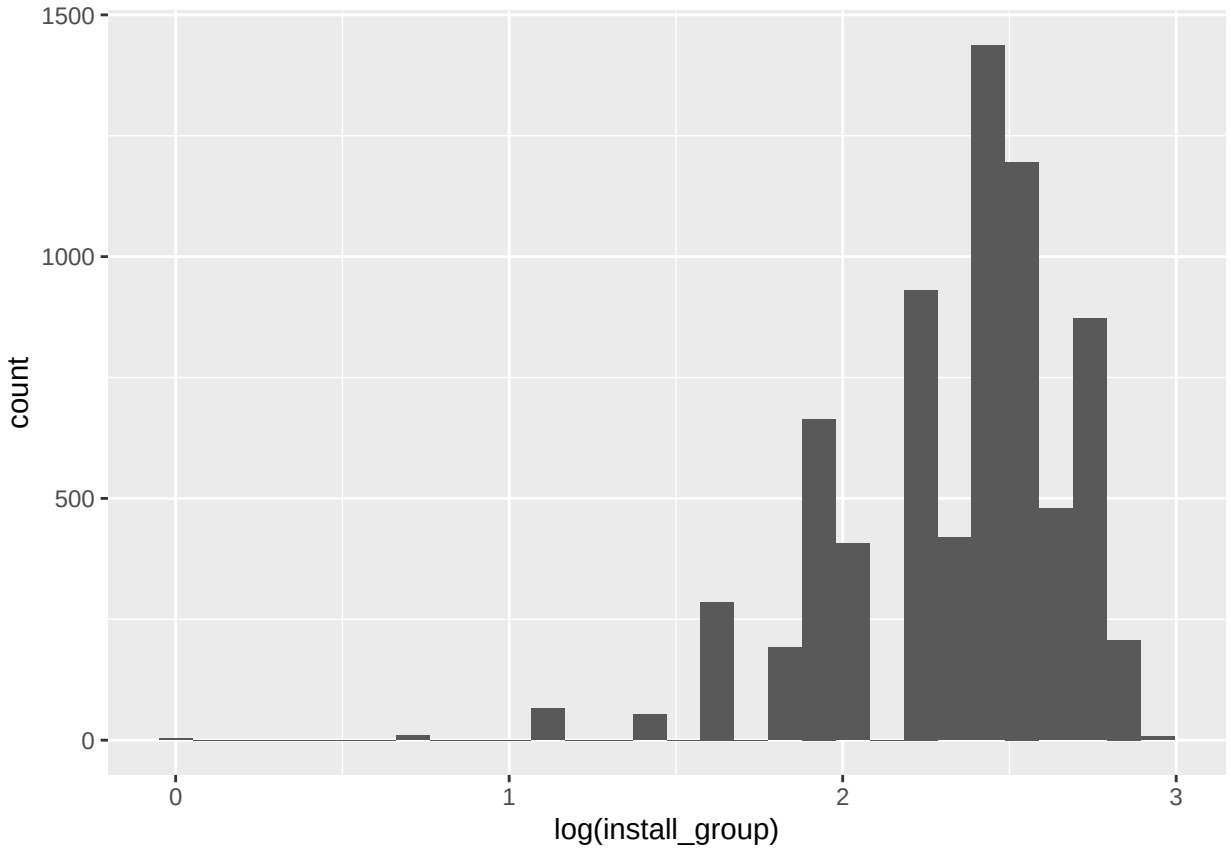

```
ggplot(data = d, aes(x = install_group)) +
  geom_histogram()
```

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



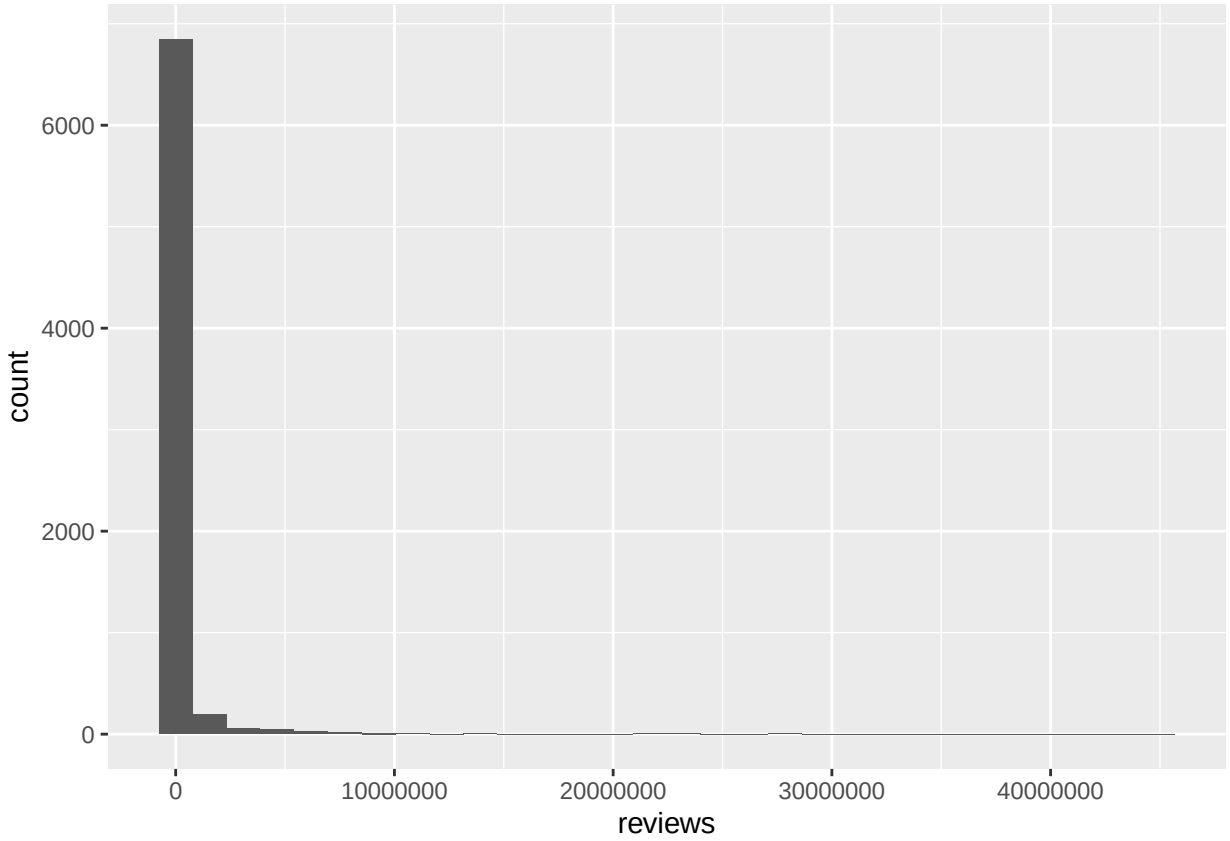

```
ggplot(data = d, aes(x = log(install_group))) +
  geom_histogram()
```

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



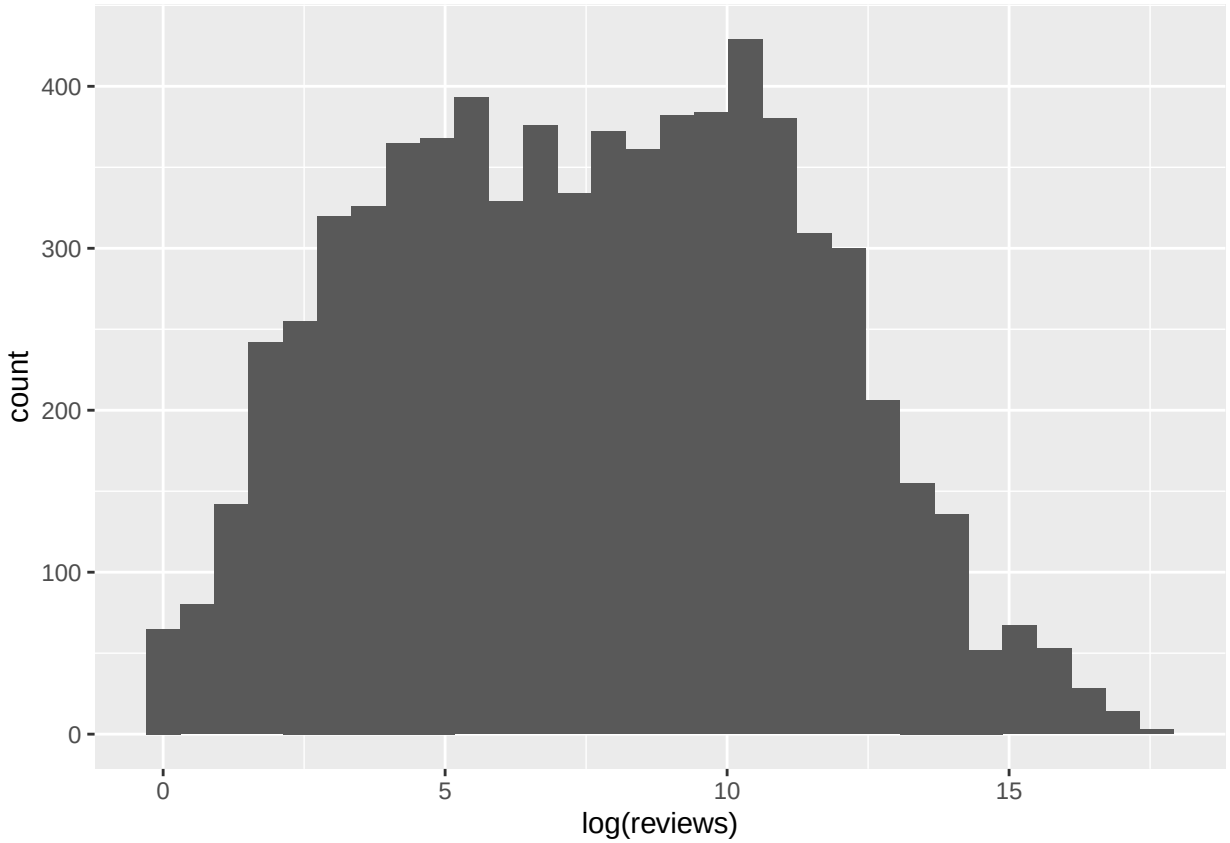

```
ggplot(data = d, aes(x = reviews)) +
  geom_histogram()
```

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



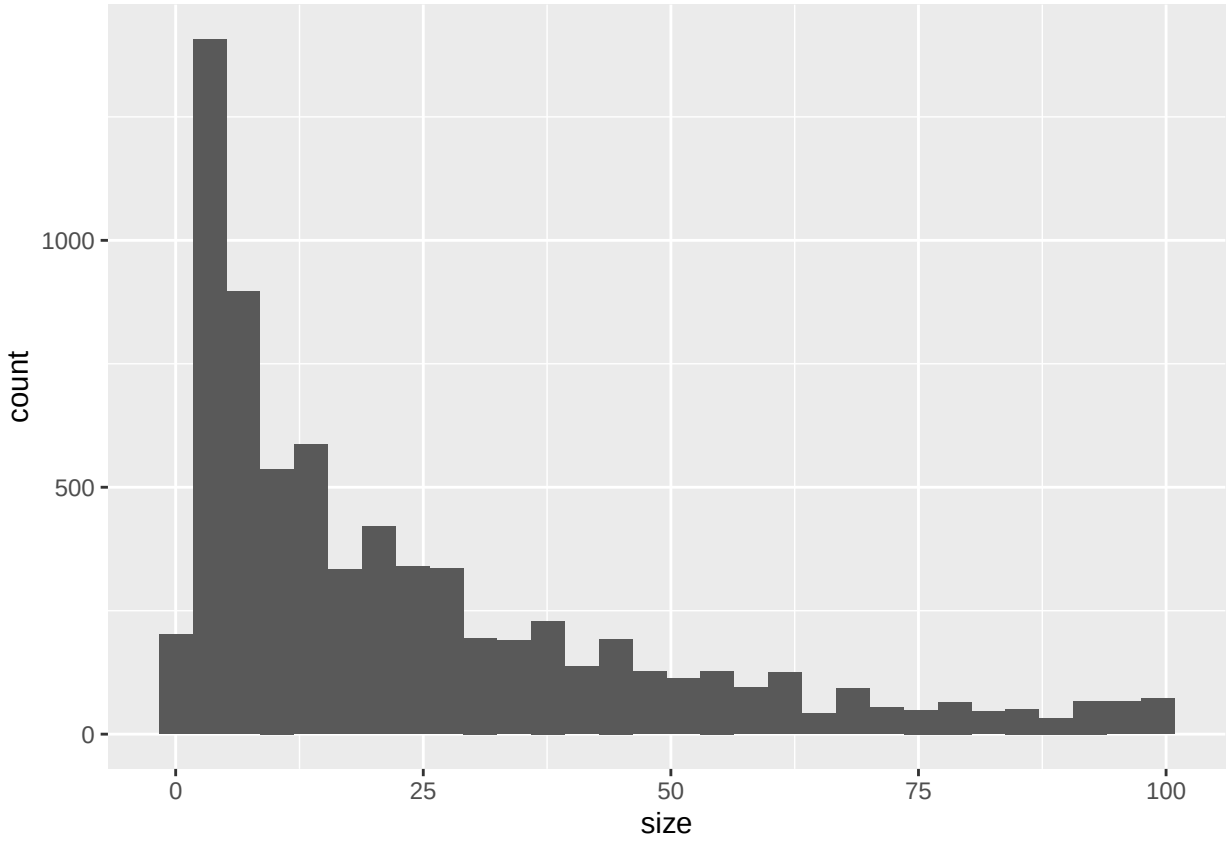

```
ggplot(data = d, aes(x = log(reviews))) +
  geom_histogram()
```

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



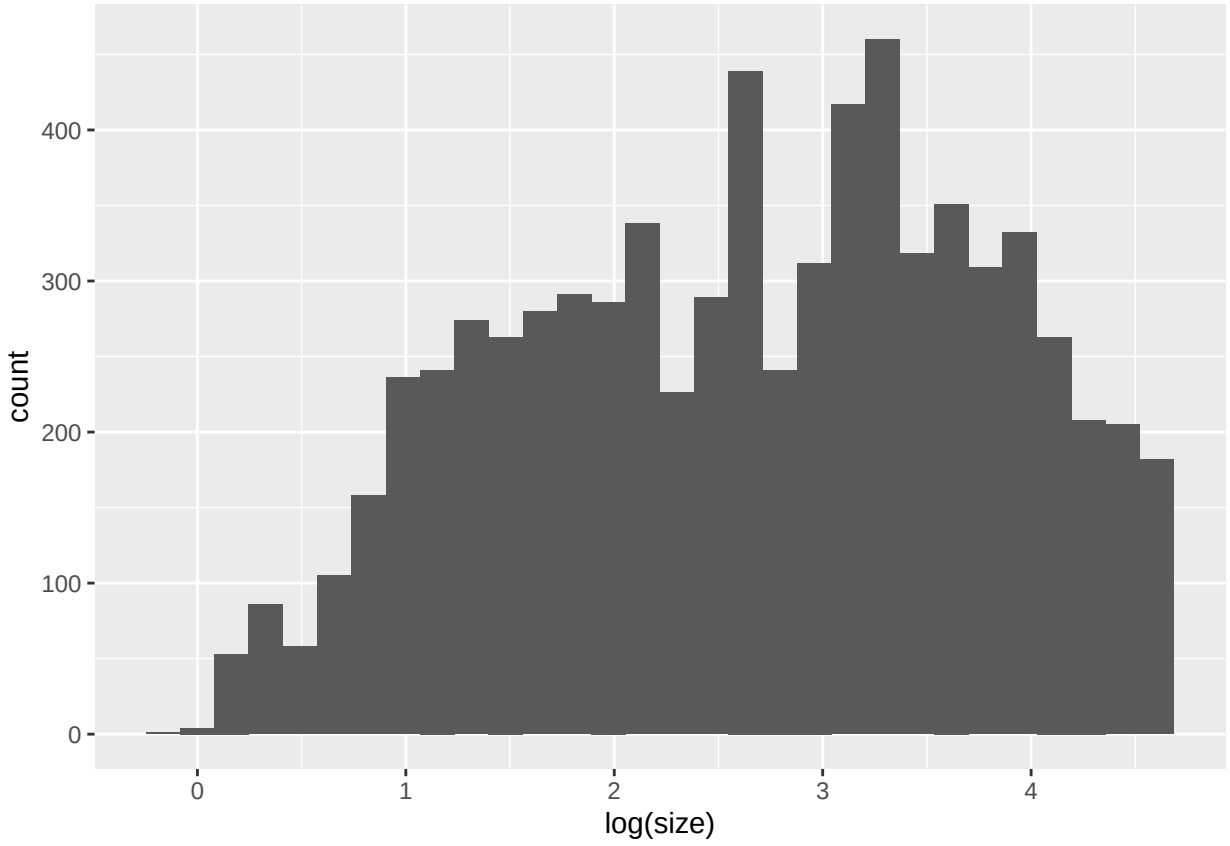

```
ggplot(data = d, aes(x = size)) +
  geom_histogram()
```

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



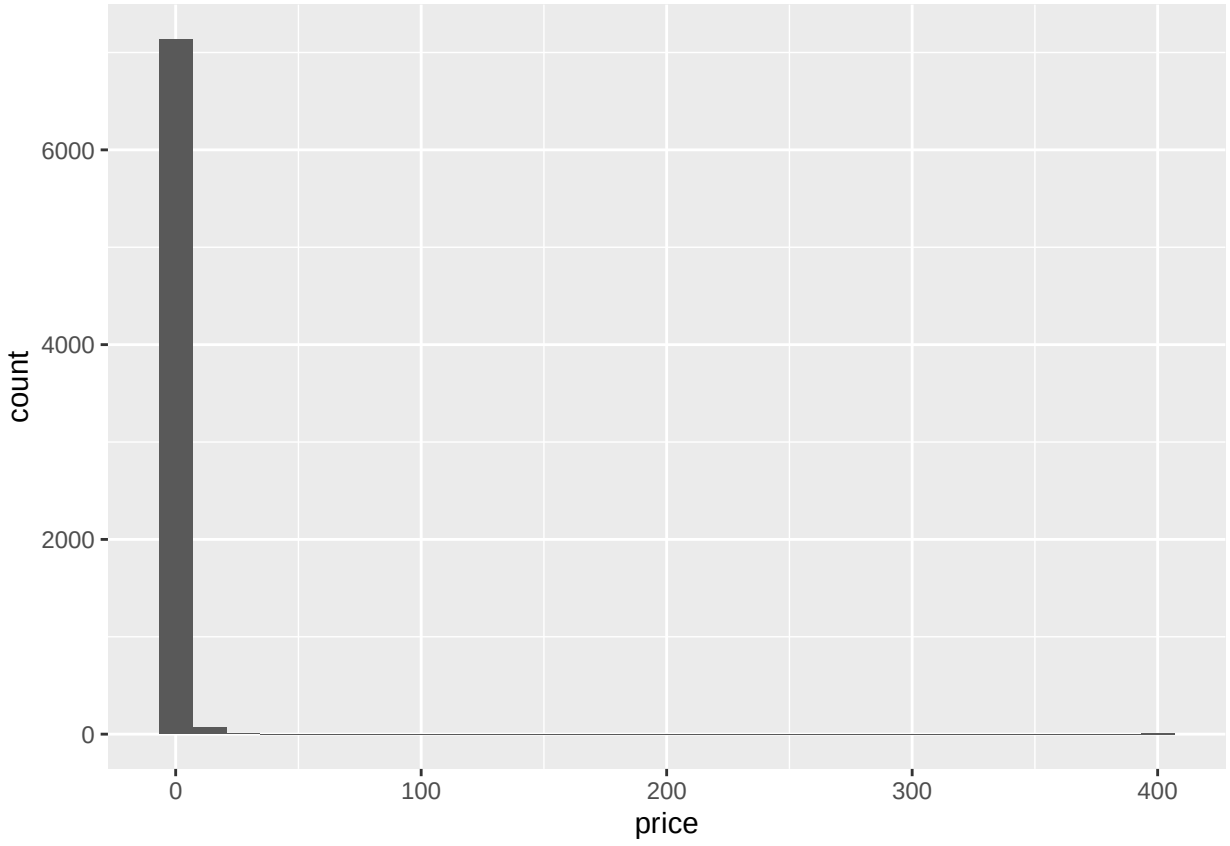

```
ggplot(data = d, aes(x = log(size))) +
  geom_histogram()
```

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




```
ggplot(data = d, aes(x = price)) +
  geom_histogram()
```

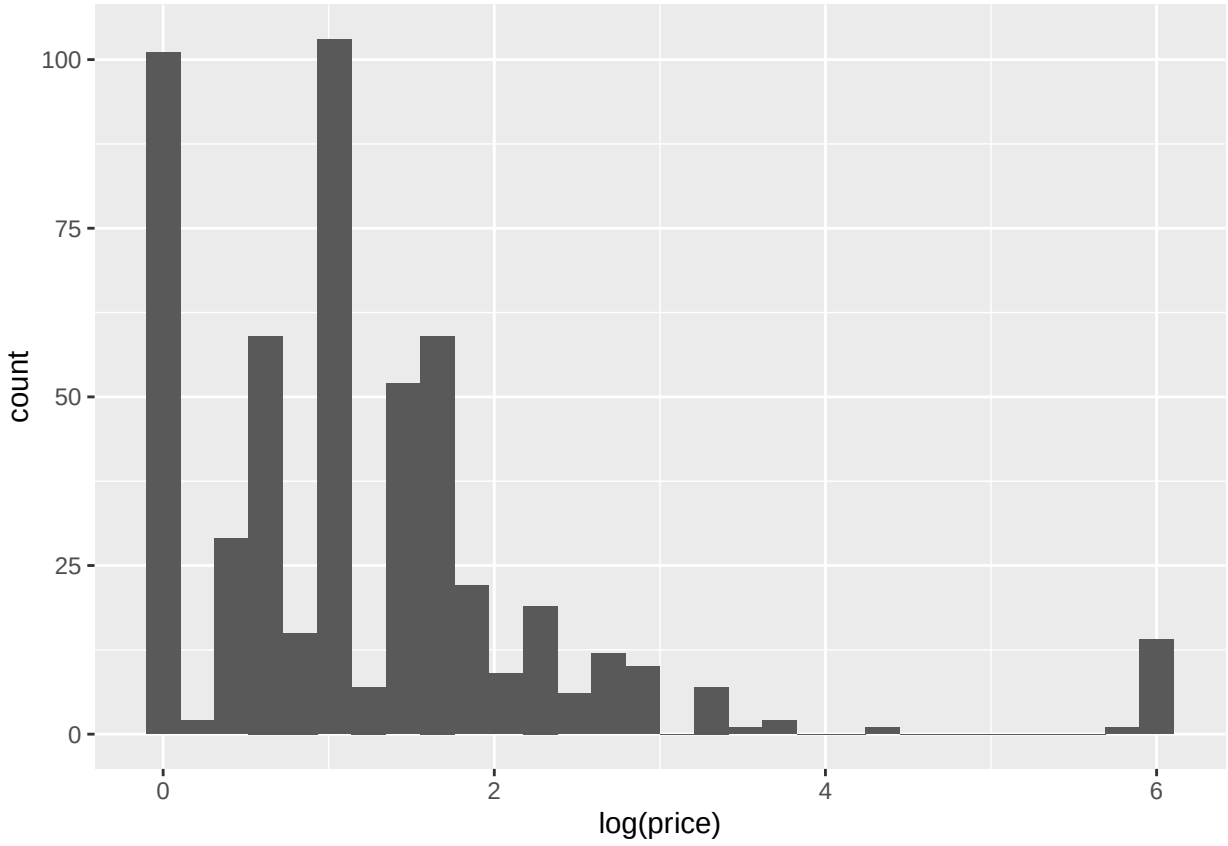
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
ggplot(data = d, aes(x = log(price))) +
  geom_histogram()
```

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

```
## Warning: Removed 6695 rows containing non-finite values (stat_bin).
```




```
categorical_cols <- c(
  'category',
  'type',
  'content_rating',
  'current_version',
  'android_version'
)

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

# compare mean install grp across variable values
table_long_cat
```

```
##         group_by_column count_apps install_group_avg install_count_med
##  1:                GAME        934         12.933619           1000000
##  2:         PHOTOGRAPHY        220         12.277273           1000000
##  3:            SHOPPING        154         12.194805           1000000
##  4:       VIDEO_PLAYERS        107         11.635514            500000
##  5:              SPORTS        230         11.282609            100000
##  6:       COMMUNICATION        197         11.274112            500000
##  7:              SOCIAL        162         11.012346            100000
##  8:  HEALTH_AND_FITNESS        190         10.900000            500000
##  9:    TRAVEL_AND_LOCAL        143         10.804196            100000
## 10:        PRODUCTIVITY        223         10.699552            100000
```

```
## 11:              FAMILY       1571         10.654360            100000
## 12:  NEWS_AND_MAGAZINES        157         10.496815             50000
## 13:               TOOLS        604         10.435430             50000
## 14:              DATING        141         10.354610            100000
## 15:     PERSONALIZATION        268         10.287313             50000
## 16: BOOKS_AND_REFERENCE        141         10.092199             50000
## 17:             FINANCE        260          9.996154             10000
## 18:           LIFESTYLE        265          9.833962             50000
## 19:            BUSINESS        221          9.090498             10000
## 20:             MEDICAL        272          8.426471             10000
## 21:                Free       6695         11.156236            100000
## 22:                Paid        531          7.919021              5000
## 23:        Everyone 10+        293         12.679181           1000000
## 24:                Teen        810         11.907407           1000000
## 25:          Mature 17+        326         11.269939            500000
## 26:            Everyone       5794         10.670866            100000
## 27:                   4        108         12.222222           1000000
## 28:                 4.1        108         11.953704           1000000
## 29:                 1.9        103         11.776699            500000
## 30:                 1.7        126         11.706349            500000
## 31:                 3.1        158         11.689873            500000
## 32:                 2.2        163         11.674847            500000
## 33:                 1.4        218         11.582569            500000
## 34:                 2.4        112         11.482143            300000
## 35:                 2.3        126         11.365079            100000
## 36:                 1.8        117         11.094017            100000
## 37:                 2.1        259         10.930502            100000
## 38:                 1.6        140         10.885714            100000
## 39:                   3        164         10.780488            100000
## 40:                 1.5        176         10.715909            100000
## 41:                 1.2        439         10.587699            100000
## 42:                 1.3        281         10.533808            100000
## 43:                 1.1        591         10.345178            100000
## 44:                   2        327          9.963303             50000
## 45:                   1       1173          9.128730             10000
## 46:                 4.1       1839         11.374116            500000
## 47:                 4.4        723         11.128631            100000
## 48:                   4       2180         10.920642            100000
## 49:                 4.2        305         10.888525            100000
## 50:                   5        433         10.872979            100000
## 51:                 2.3        784         10.838010            100000
## 52:                 2.1        111         10.612613             50000
## 53:                 4.3        178         10.556180            100000
## 54:                   3        205         10.107317             50000
## 55:                 2.2        197          9.375635             10000
##         group_by_column count_apps install_group_avg install_count_med
##            variable
##  1:        category
##  2:        category
##  3:        category
##  4:        category
##  5:        category
##  6:        category
##  7:        category
```

```
##  8:        category
##  9:        category
## 10:        category
## 11:        category
## 12:        category
## 13:        category
## 14:        category
## 15:        category
## 16:        category
## 17:        category
## 18:        category
## 19:        category
## 20:        category
## 21:            type
## 22:            type
## 23:  content_rating
## 24:  content_rating
## 25:  content_rating
## 26:  content_rating
## 27: current_version
## 28: current_version
## 29: current_version
## 30: current_version
## 31: current_version
## 32: current_version
## 33: current_version
## 34: current_version
## 35: current_version
## 36: current_version
## 37: current_version
## 38: current_version
## 39: current_version
## 40: current_version
## 41: current_version
## 42: current_version
## 43: current_version
## 44: current_version
## 45: current_version
## 46: android_version
## 47: android_version
## 48: android_version
## 49: android_version
## 50: android_version
## 51: android_version
## 52: android_version
## 53: android_version
## 54: android_version
## 55: android_version
##            variable
```

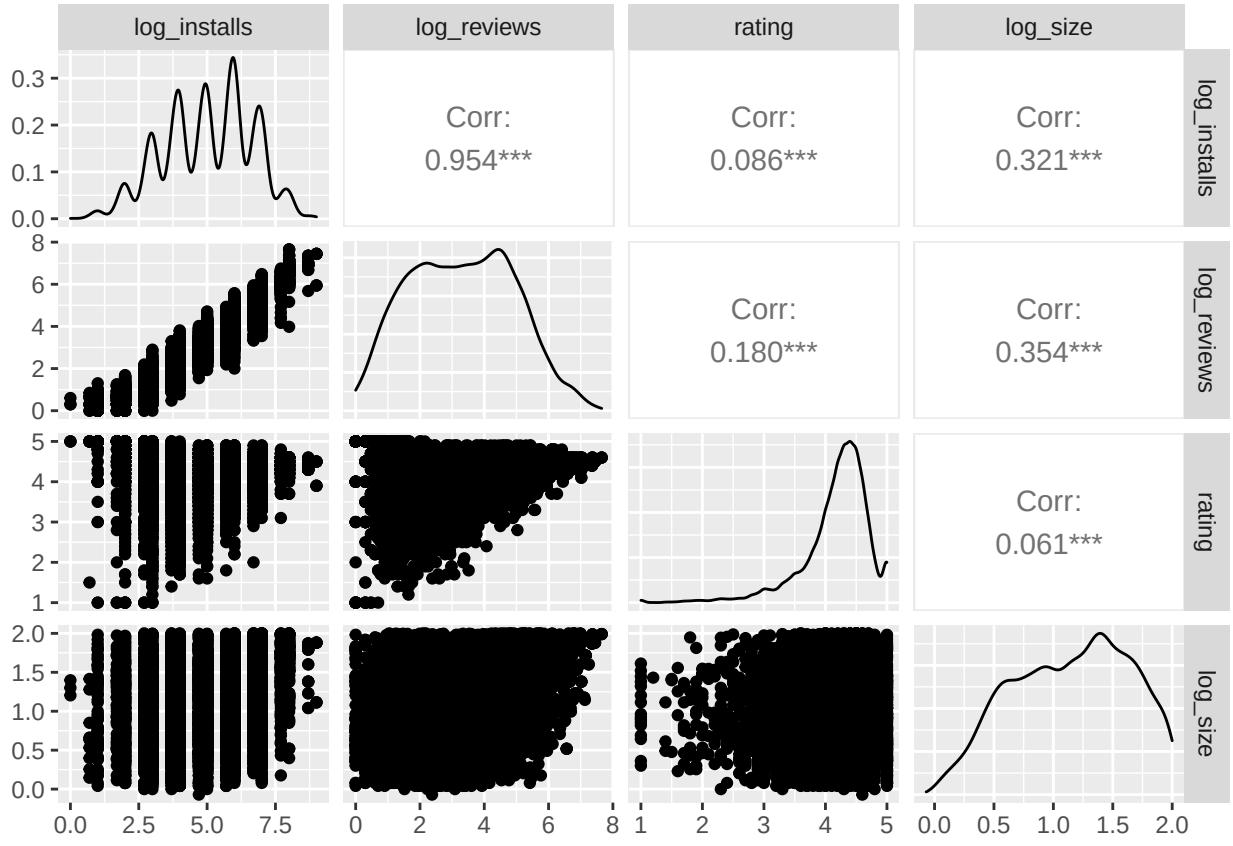
```
# compare distribution of mean install grp across variable values
table_quantile_cat
```

```
##          0%       25%       50%      75%     100%        variable
## 1:  8.426471 10.238535 10.676956 11.27624 12.93362        category
```

```
## 2:  7.919021  8.728325  9.537628 10.34693 11.15624            type
## 3: 10.670866 11.120171 11.588673 12.10035 12.67918  content_rating
## 4:  9.128730 10.651804 11.094017 11.68236 12.22222 current_version
## 5:  9.375635 10.570288 10.855495 10.91261 11.37412 android_version
##    diff_min_vs_max
## 1:        4.507148
## 2:        3.237215
## 3:        2.008314
## 4:        3.093492
## 5:        1.998482
```

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 (7226 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.

```
ggpairs(d[, c('log_installs', 'log_reviews', 'rating', 'log_size')])
```




```
model_small  <- lm(log_installs ~ 1 + log_reviews, data = d)
model_medium <- lm(log_installs ~ 1 + log_reviews + rating + log_size, data = d)
```

```r
model_large <- lm(log_installs ~ 1 + log_reviews + rating + 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                  0.944***                 0.966***                 0.947***
##                              (0.004)                  (0.004)                  (0.003)
##
## rating                                                -0.256***                -0.235***
##                                                       (0.013)                  (0.009)
##
## log_size                                              -0.066***                -0.053***
##                                                       (0.012)                  (0.011)
##
## factor(type)Paid                                                               -0.617***
##                                                                                (0.020)
##
## Constant                     1.907***                 2.974***                 2.984***
##                              (0.015)                  (0.053)                  (0.040)
##
## ------------------------------------------------------------------------------------------------
## Observations                  7,226                    7,226                    7,226
## R2                            0.910                    0.918                    0.927
## Adjusted R2                   0.910                    0.918                    0.927
## Residual Std. Error      0.481 (df = 7224)        0.460 (df = 7222)        0.432 (df = 7221)
## F Statistic         72,962.100*** (df = 1; 7224) 26,842.640*** (df = 3; 7222) 23,068.790*** (df = 4;
## ================================================================================================
## Note:                                                                *p<0.1; **p<0.05; ***
```

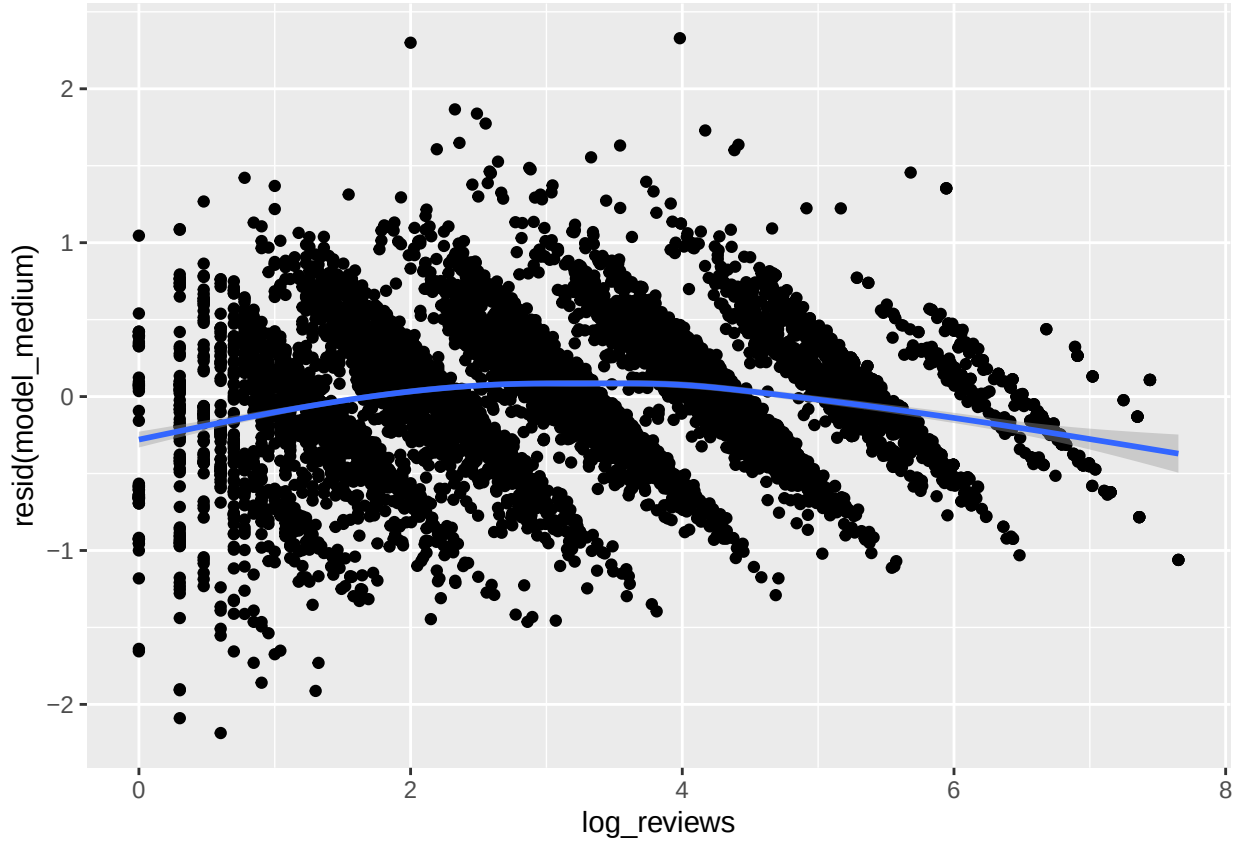
2. **No Perfect Colinearity:** We can immediately conclude that `log_installs`, `log_reviews`, `rating`, and `log_size` are not perfectly colinear as otherwise the regression above would have failed. We can also assess near perfect colinearity for these variables by observing the robust standard errors 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 `rating` 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.

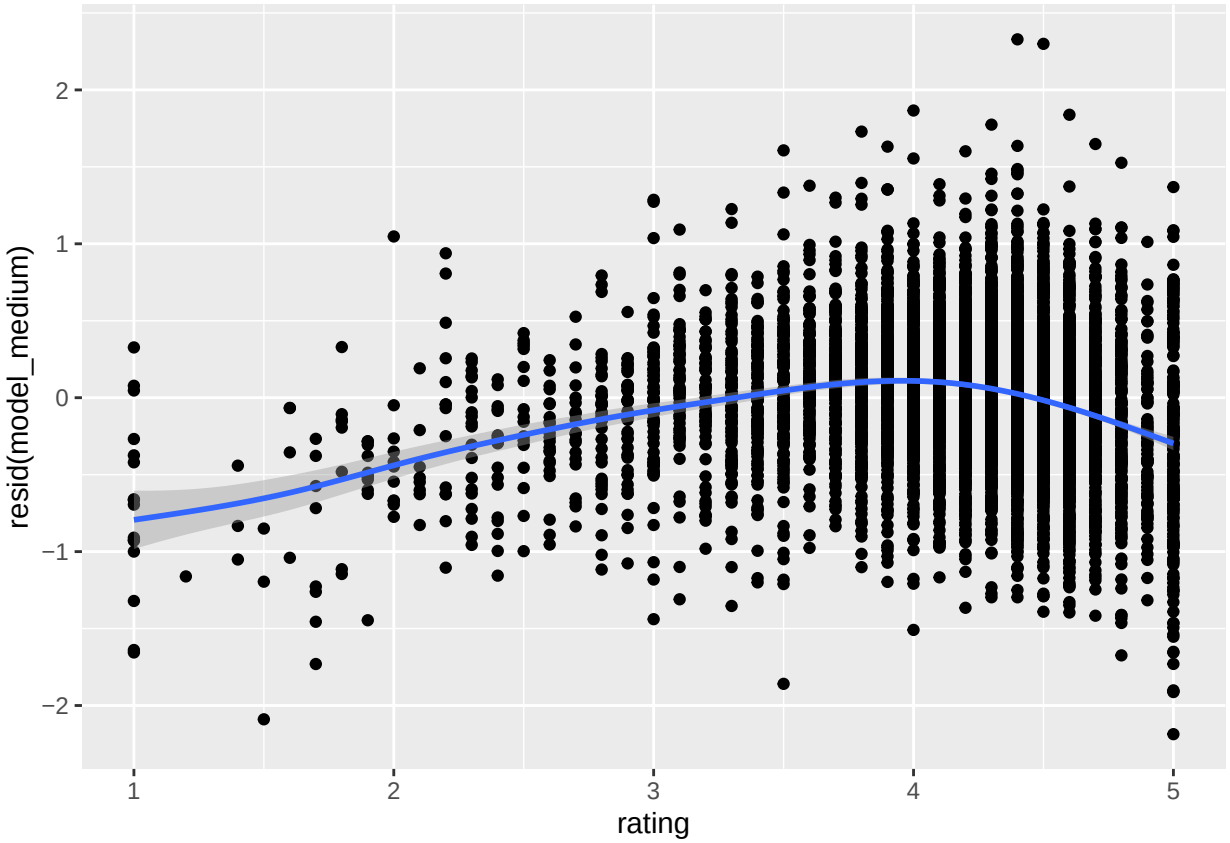
```
# Reviews versus residuals
plot_1 <- ggplot(data = d, mapping = aes(x = log_reviews, y = resid(model_medium))) +
  geom_point() + stat_smooth()
plot_1
```

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



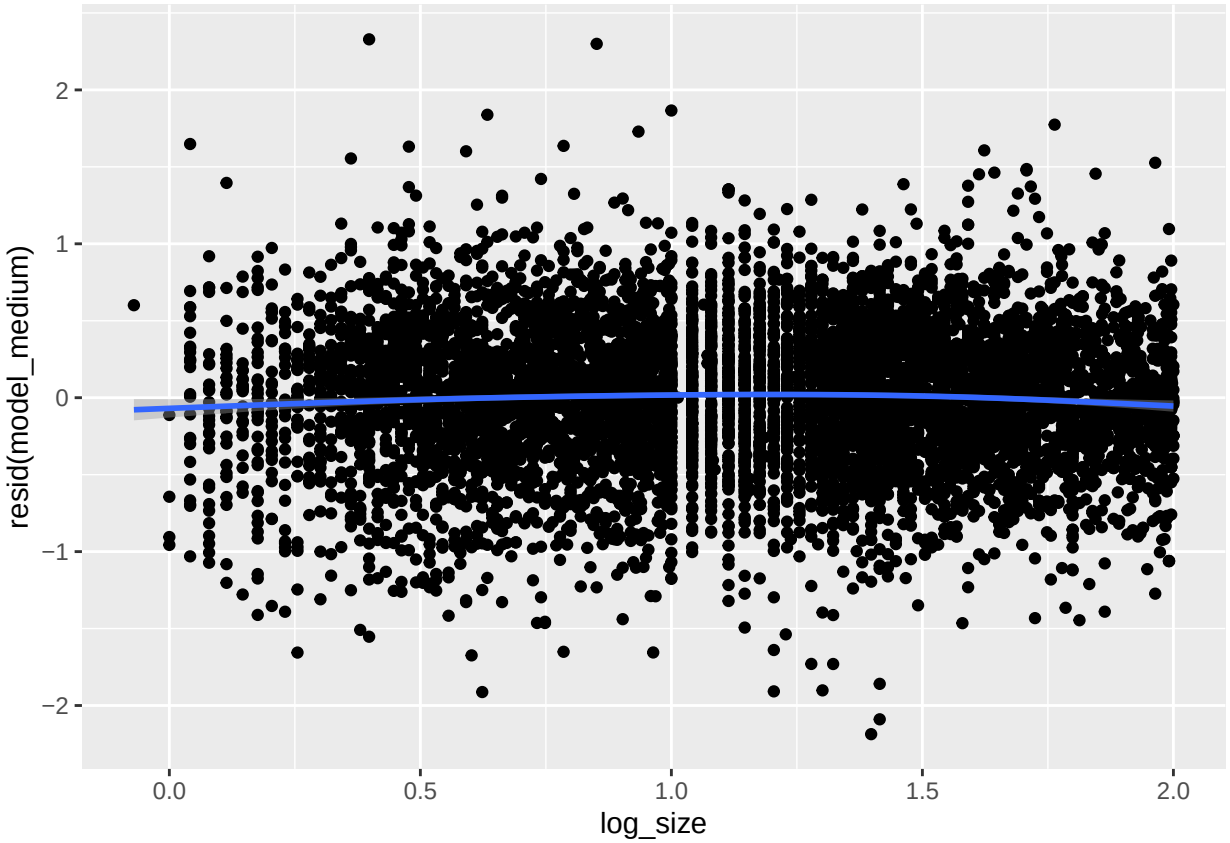

```
# Ratings versus residuals
plot_2 <- ggplot(data = d, mapping = aes(x = rating, y = resid(model_medium))) +
  geom_point() + stat_smooth()
plot_2
```

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



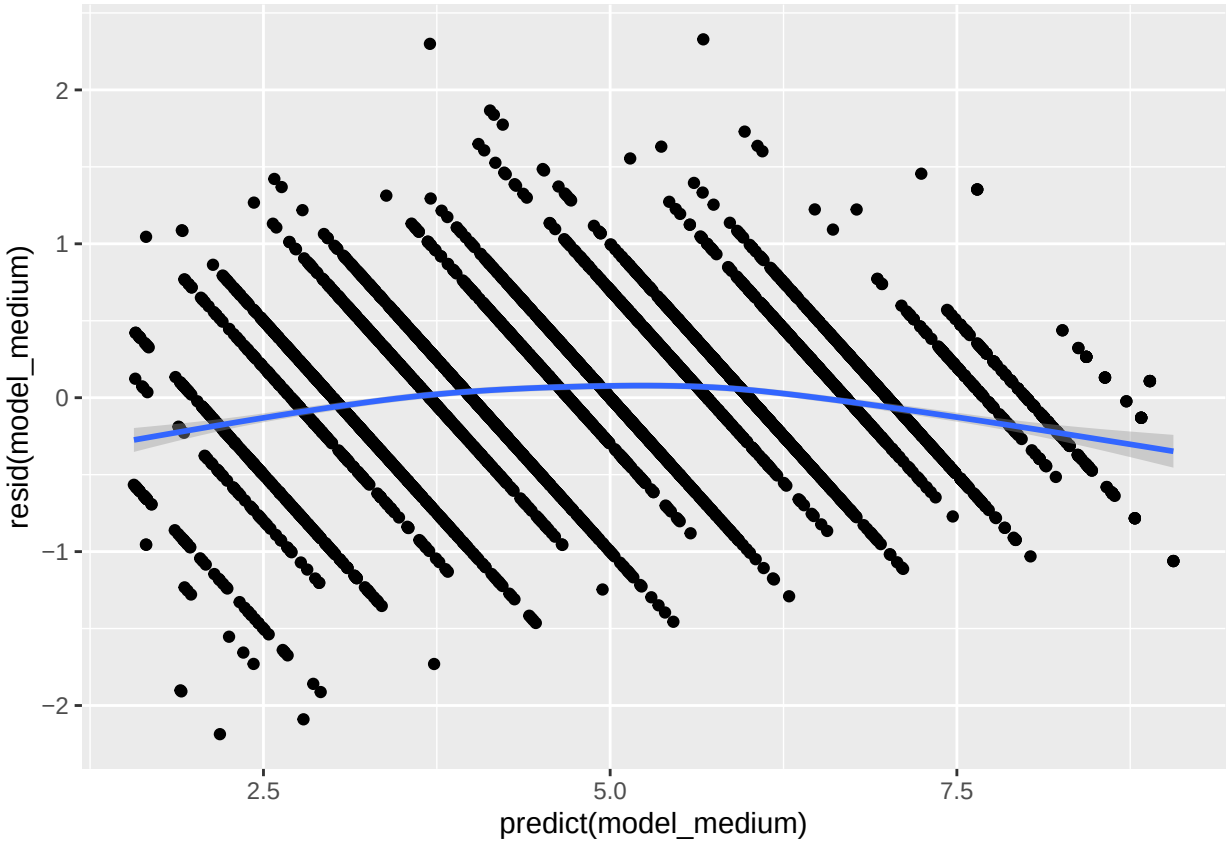

```
# Size versus residuals
plot_3 <- ggplot(data = d, mapping = aes(x = log_size, y = resid(model_medium))) +
  geom_point() + stat_smooth()
plot_3
```

```
## `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")'
```



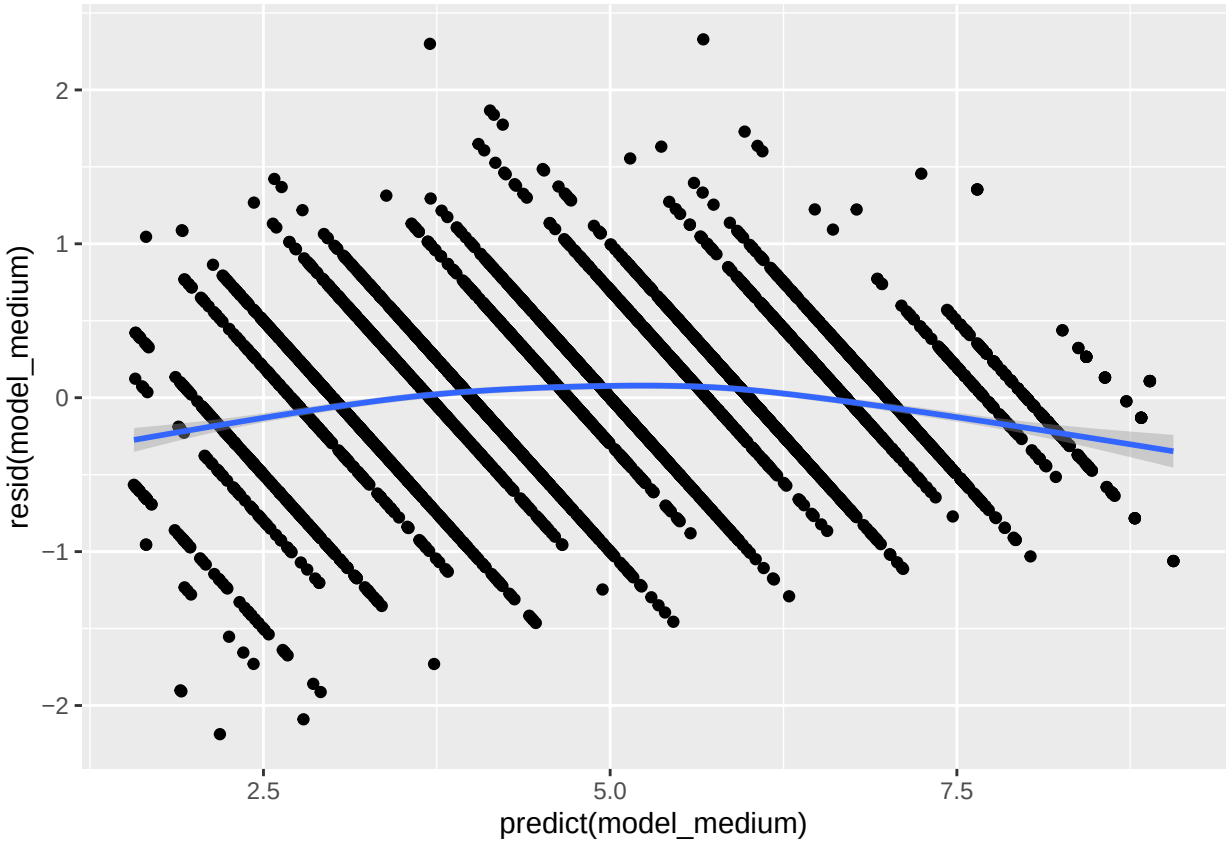
4. **Homoskedastic Errors:** When assessing homoskedastic errors, we seek to determine if there is a relationship between the variance of the model residuals and the predictors. If the homoskedastic assumption is satisfied, then we should observe a lack of relationship; conversely, if the data are heteroskedastic 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 Breush-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 = 206.05, df = 1, p-value < 0.00000000000000022
```

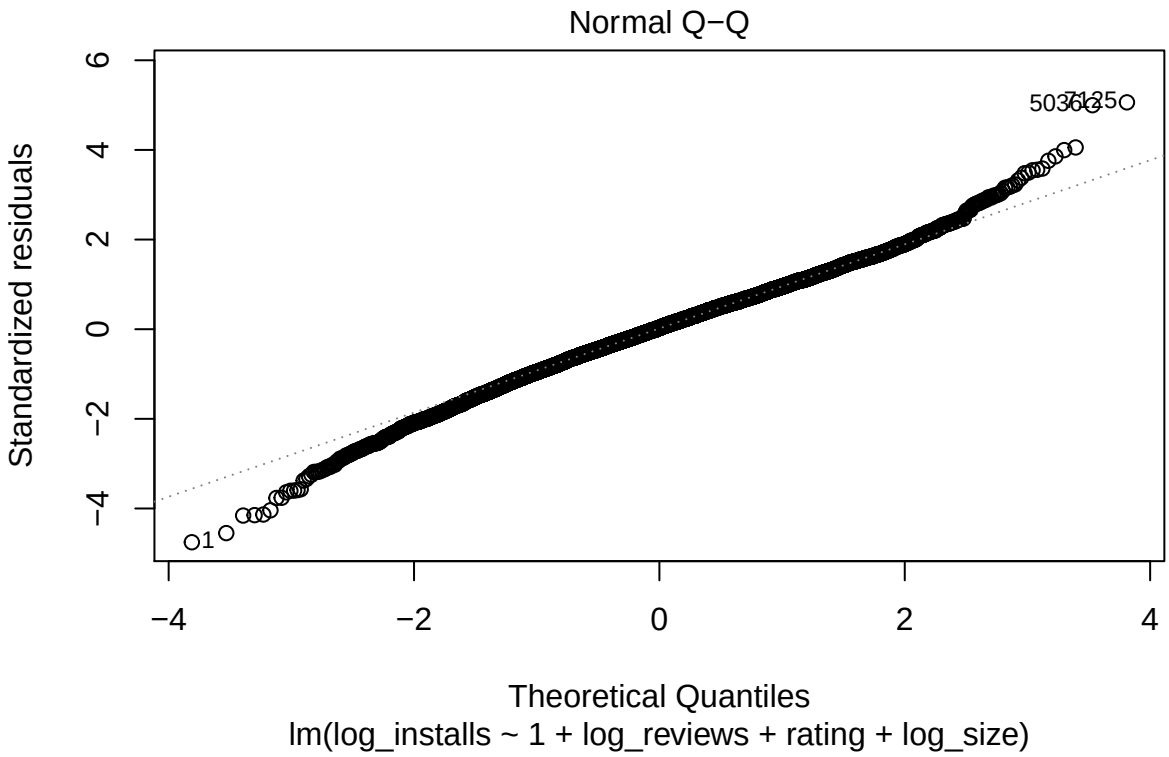
```
plot_4
```

```
## `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 coefficent) > 0, we know higher average review leads to more installs. Since Gamma1 (installs slope cofficent for reverse causality) is > 0, this leads to positive feedback. Given we have two potentially positive coefficents, 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                        0.944***
##                                    (0.004)
##
## log_installs                                        0.964
##
##
## Constant                           1.907***      -1.542***
##                                    (0.015)        (0.053)
##
## ------------------------------------------------------------
## Observations                        7,226          7,226
## R2                                  0.910          0.910
## Adjusted R2                         0.910          0.910
## Residual Std. Error (df = 7224)     0.481          0.487
## F Statistic (df = 1; 7224)      72,962.100***  72,962.100***
## ============================================================
## Note:                           *p<0.1; **p<0.05; ***p<0.01
```