eda

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

```r
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')
```

```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             Installals           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
##
##
##
##
```

```r
d <- get_clean_dataset()

# summary of dataset
summary(d)
```

```
##     installs               size           reviews            rating
##   Min.   :       500   Min.   :  0.85   Min.   :      100   Min.   :1.600
##   1st Qu.:    100000   1st Qu.:  7.10   1st Qu.:      929   1st Qu.:4.000
##   Median :   1000000   Median : 19.00   Median :     9194   Median :4.300
##   Mean   :  10215340   Mean   : 26.89   Mean   :   373798   Mean   :4.198
##   3rd Qu.:   5000000   3rd Qu.: 40.00   3rd Qu.:    71762   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:305       1st Qu.:0.04931   1st Qu.:4.000
##   Median :  0.000   TRUE :5103      Median :0.19178   Median :4.100
##   Mean   :  1.036                   Mean   :0.71197   Mean   :3.845
##   3rd Qu.:  0.000                   3rd Qu.:0.83836   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:5408        Mode :logical      Mode :logical
##   1st Qu.:  1.200   Class :character   FALSE:4245         FALSE:4554
##   Median :  2.100   Mode  :character   TRUE :1163         TRUE :854
##   Mean   :  5.157
##   3rd Qu.:  4.100
##   Max.   :858.000
##   is_tools_category    genre            content_rating     is_content_everyone
##   Mode :logical     Length:5408        Length:5408        Mode :logical
##   FALSE:4996        Class :character   Class :character   FALSE:1228
##   TRUE :412         Mode  :character   Mode  :character   TRUE :4180
##
##
##
##      type           install_group    log_installs      log_size
##   Length:5408        Min.   : 1.00   Min.   :2.699   Min.   :-0.07058
##   Class :character   1st Qu.: 6.00   1st Qu.:5.000   1st Qu.: 0.85126
##   Mode  :character   Median : 8.00   Median :6.000   Median : 1.27875
##                      Mean   : 7.29   Mean   :5.704   Mean   : 1.21414
##                      3rd Qu.: 9.00   3rd Qu.:6.699   3rd Qu.: 1.60206
##                      Max.   :14.00   Max.   :9.000   Max.   : 2.00000
##   log_current_version log_last_updated   log_reviews
##   Min.   :0.0000      Min.   :0.00000   Min.   :2.000
##   1st Qu.:0.3424      1st Qu.:0.02091   1st Qu.:2.968
##   Median :0.4914      Median :0.07620   Median :3.964
##   Mean   :0.5673      Mean   :0.17347   Mean   :3.995
```

```
##  3rd Qu.:0.7076      3rd Qu.:0.26443   3rd Qu.:4.856
##  Max.   :2.9340      Max.   :0.96456   Max.   :7.652
```

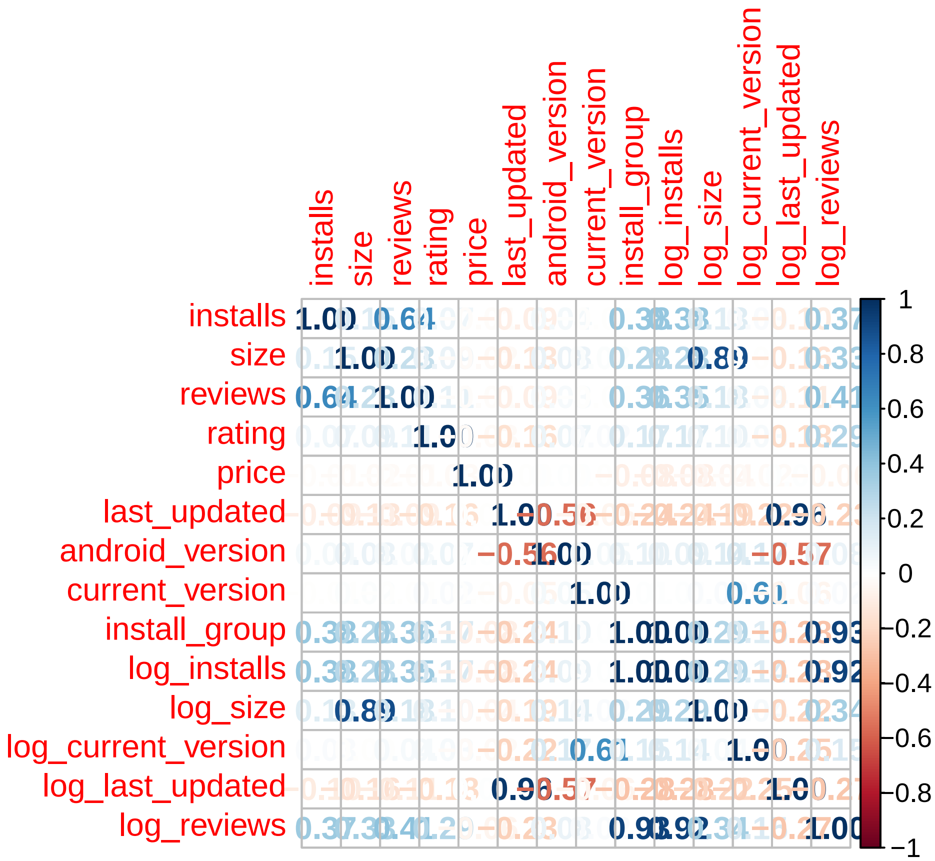
```
# save a data.table version for some easier wrangling downstream
d_dt <- as.data.table(d)
```

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

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

```
##               0%          5%           25%            50%
##  1: 500.00000000 10000.00000000 100000.00000000 1000000.00000000
##  2:   0.85000000    2.50000000      7.10000000      19.00000000
##  3: 100.00000000  156.00000000    928.75000000    9194.00000000
##  4:   1.60000000    3.40000000      4.00000000       4.30000000
##  5:   0.00000000    0.00000000      0.00000000       0.00000000
##  6:   0.00000000    0.01369863      0.04931507       0.19178082
##  7:   1.00000000    2.20000000      4.00000000       4.10000000
##  8:   0.00000000    1.00000000      1.20000000       2.10000000
##  9:   1.00000000    4.00000000      6.00000000       8.00000000
## 10:   2.69897000    4.00000000      5.00000000       6.00000000
## 11:  -0.07058107    0.39794001      0.85125835       1.27875360
## 12:   0.00000000    0.30103000      0.34242268       0.49136169
## 13:   0.00000000    0.00590886      0.02090591       0.07619639
## 14:   2.00000000    2.19312460      2.96789878       3.96350444
##                75%            95%            100%            variable
##  1: 5000000.0000000 50000000.0000000 1000000000.0000000         installs
##  2:      40.0000000       82.0000000        100.0000000             size
##  3:   71762.2500000  1224897.7000000   44893888.0000000          reviews
##  4:       4.5000000        4.7000000          5.0000000           rating
##  5:       0.0000000        0.9900000        400.0000000            price
##  6:       0.8383562        3.2884932          8.2164384     last_updated
##  7:       4.1000000        5.0000000          8.0000000  android_version
##  8:       4.1000000       10.0000000        858.0000000  current_version
##  9:       9.0000000       11.0000000         14.0000000    install_group
## 10:       6.6989700        7.6989700          9.0000000     log_installs
## 11:       1.6020600        1.9138139          2.0000000         log_size
## 12:       0.7075702        1.0413927          2.9339932 log_current_version
## 13:       0.2644297        0.6323046          0.9645631    log_last_updated
## 14:       4.8558960        6.0880991          7.6521872      log_reviews
##      diff_min_vs_max
##  1: 999999500.0000000
##  2:        99.1500000
##  3:  44893788.0000000
##  4:         3.4000000
##  5:       400.0000000
##  6:         8.2164384
##  7:         7.0000000
##  8:       858.0000000
##  9:        13.0000000
## 10:         6.3010300
## 11:         2.0705811
## 12:         2.9339932
## 13:         0.9645631
```

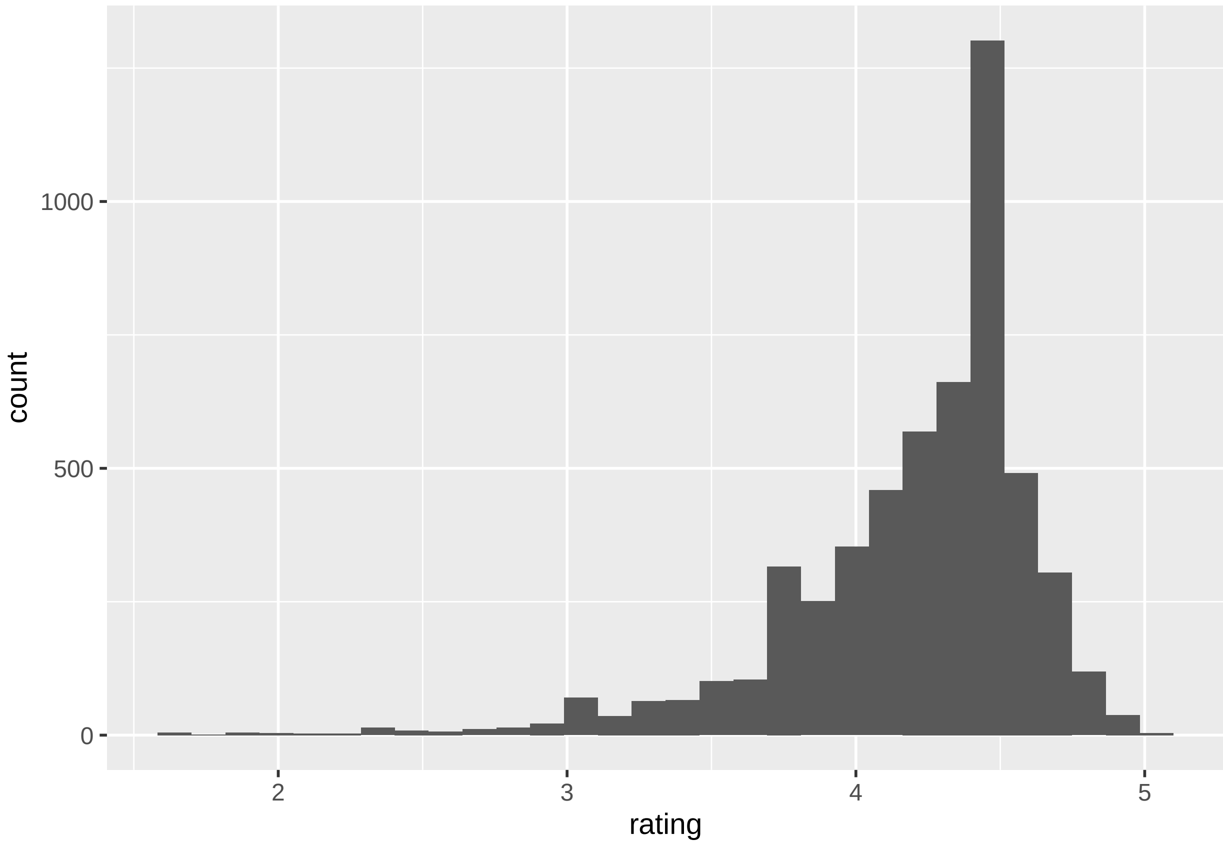
```
## 14:         5.6521872
```

```r
# 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')
```



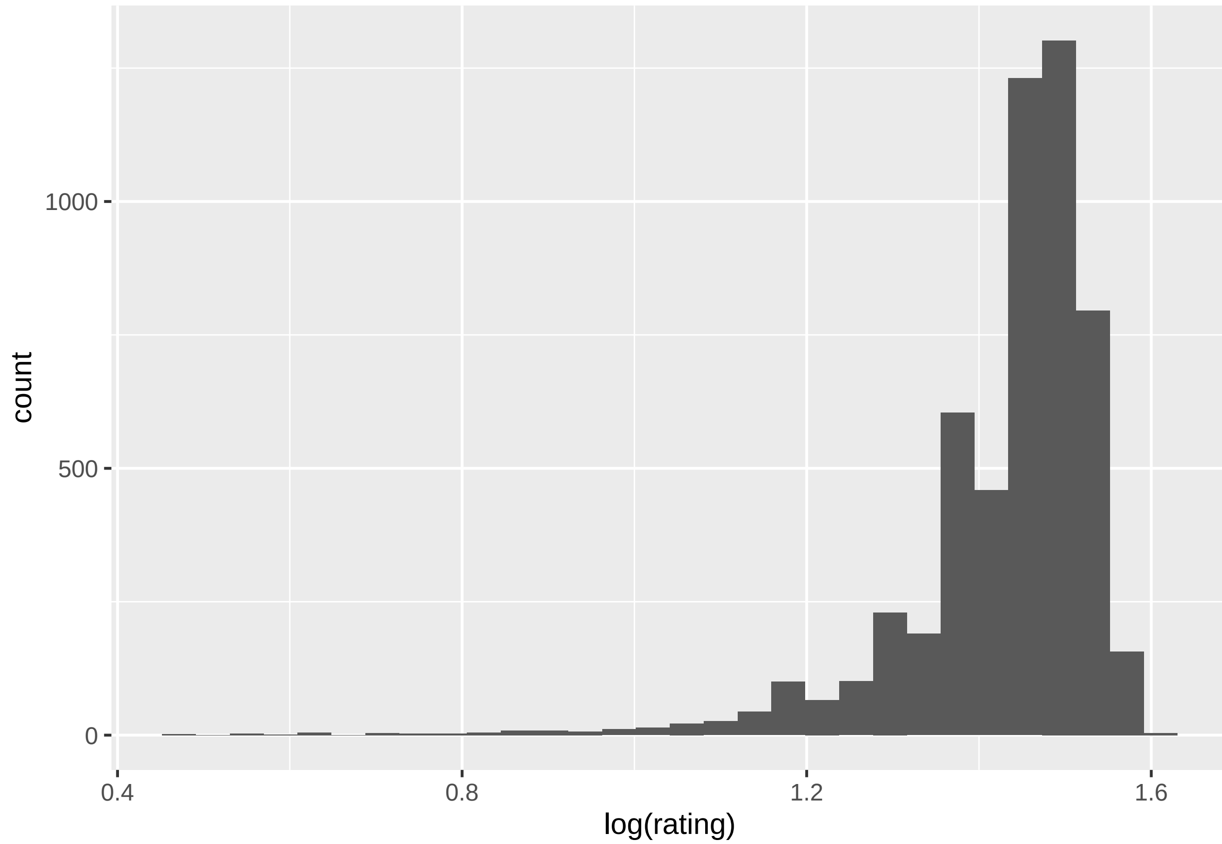

```r
# Corrplot across variables (25th PCTL outliers removed)
corrplot(cor(d[d$reviews >= 100,numeric_cols], use = "complete.obs"),
         method = 'number')

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

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



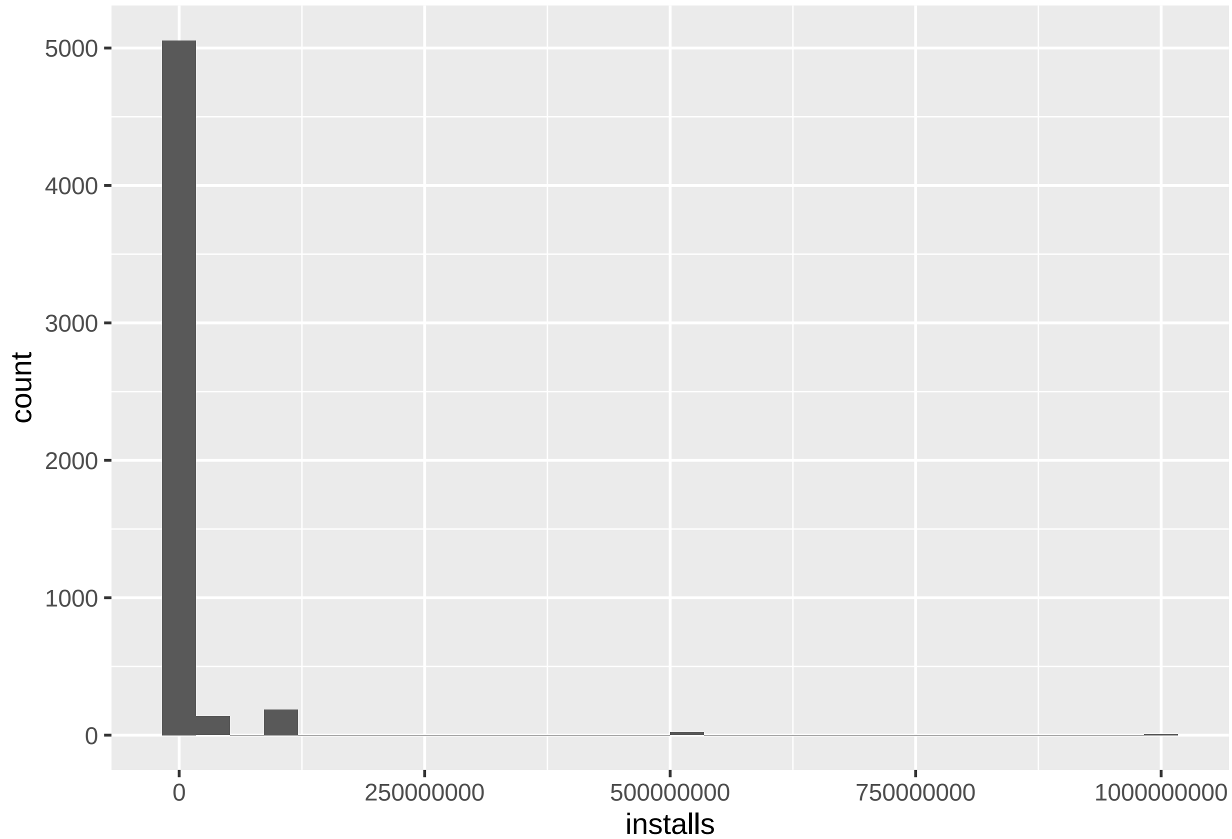

```r
ggplot(data = d, aes(x = log(rating))) +
  geom_histogram()
```

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



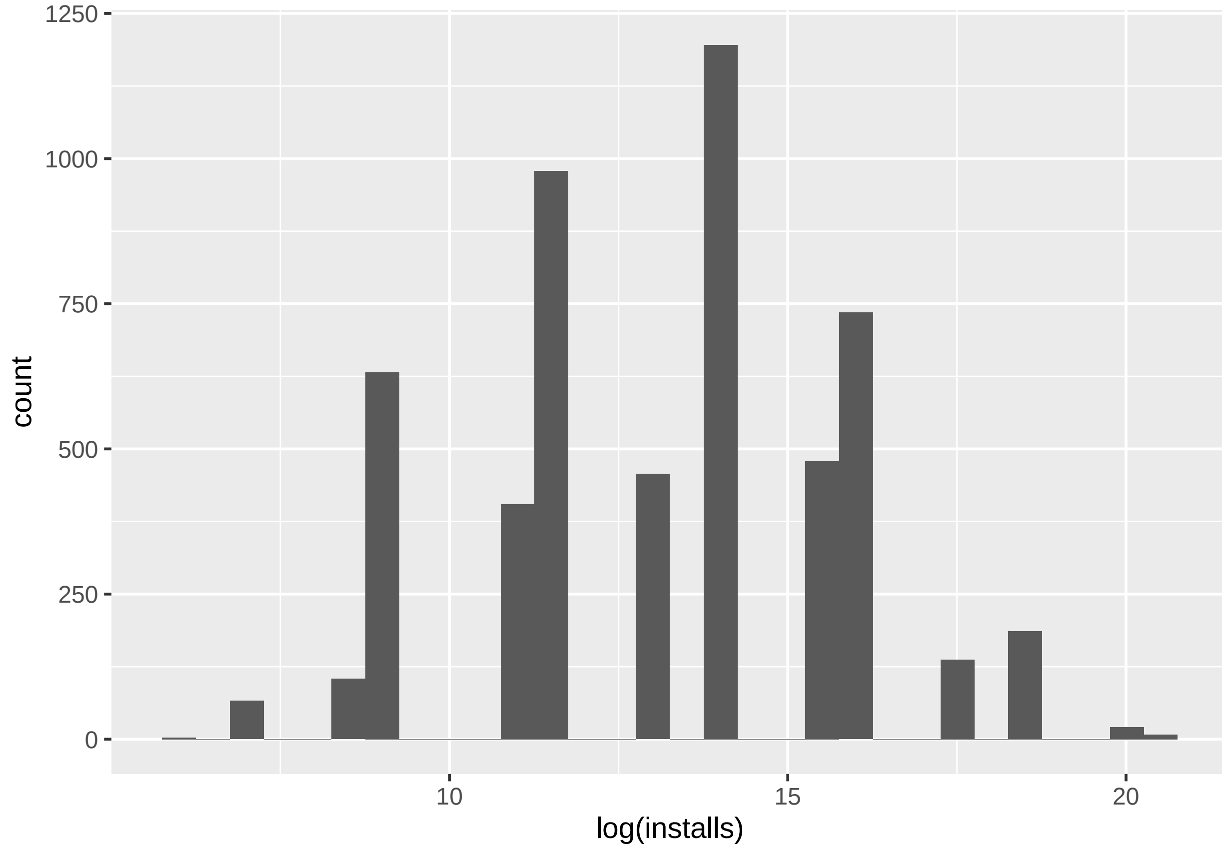

```
ggplot(data = d, aes(x = installs)) +
  geom_histogram()
```

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



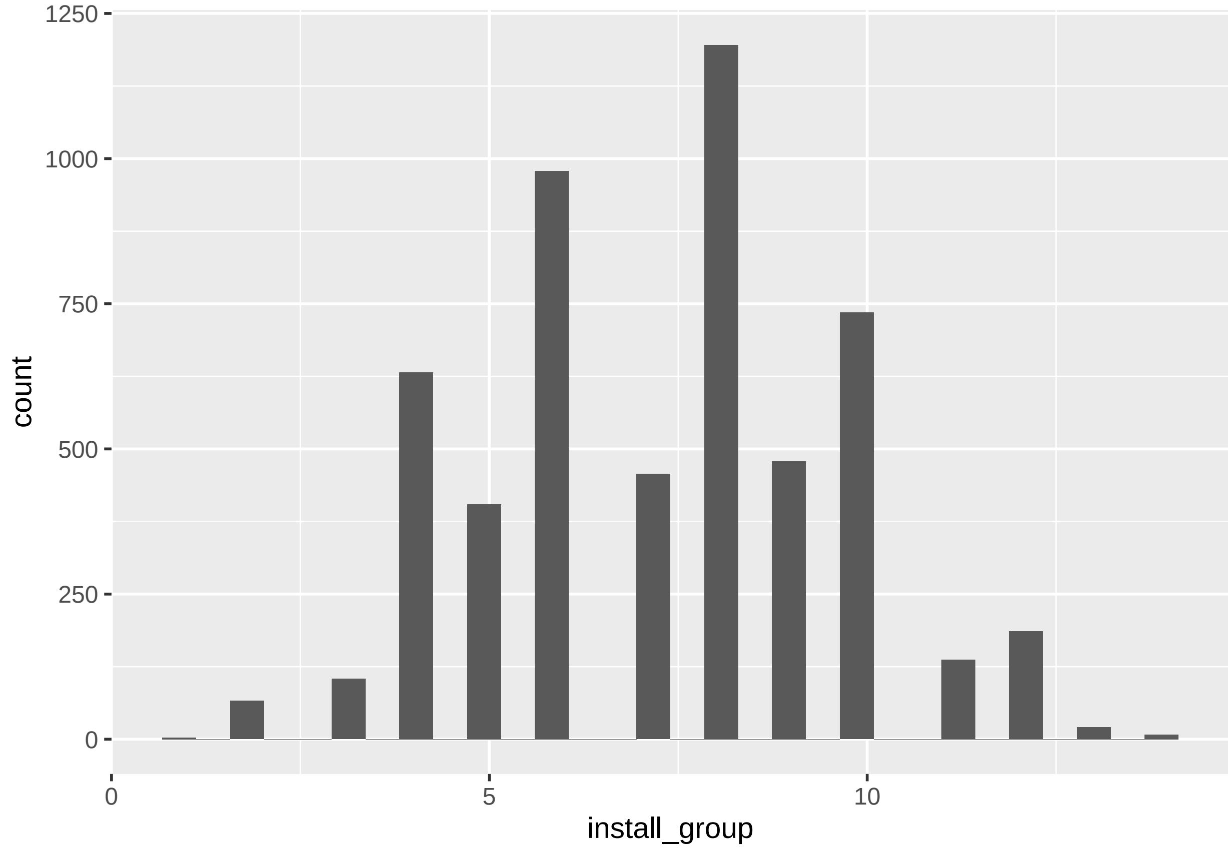

```
ggplot(data = d, aes(x = log(installs))) +
  geom_histogram()
```

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



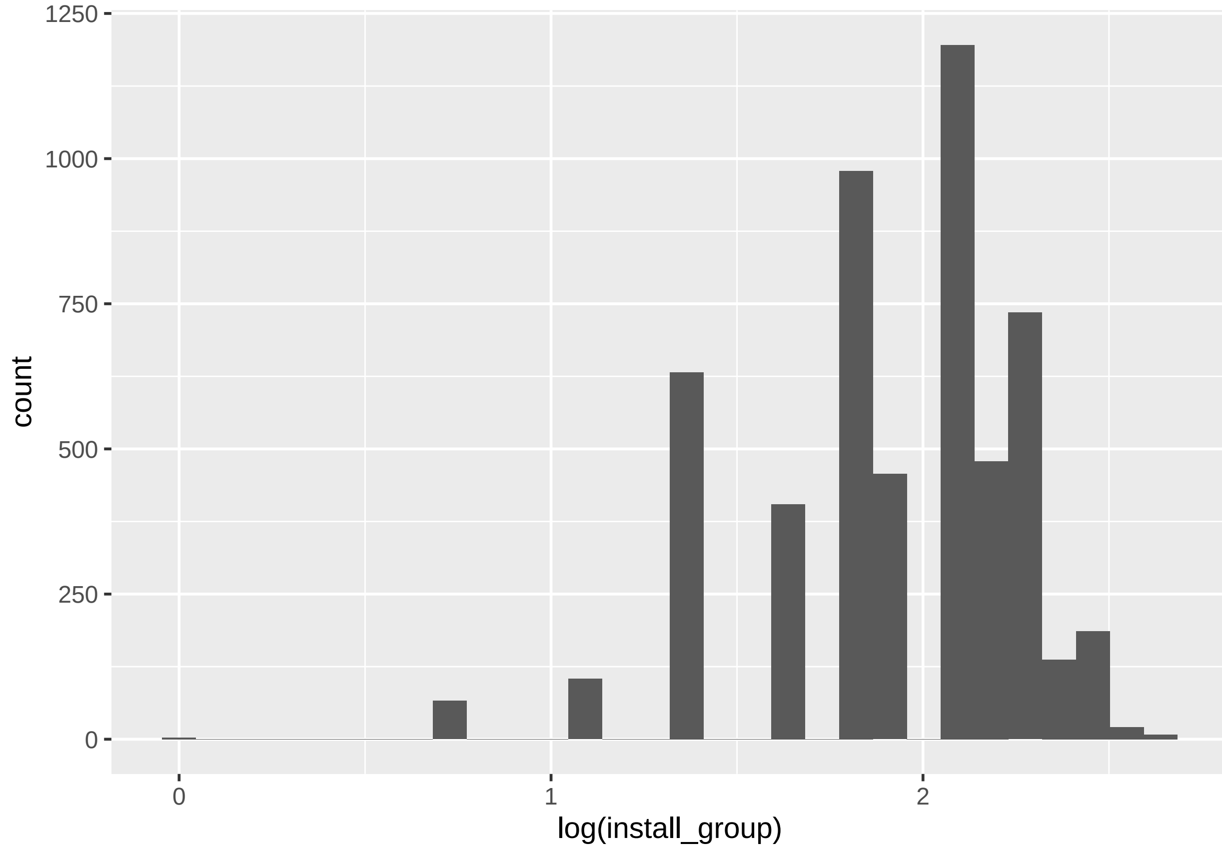

```
ggplot(data = d, aes(x = install_group)) +
  geom_histogram()
```

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



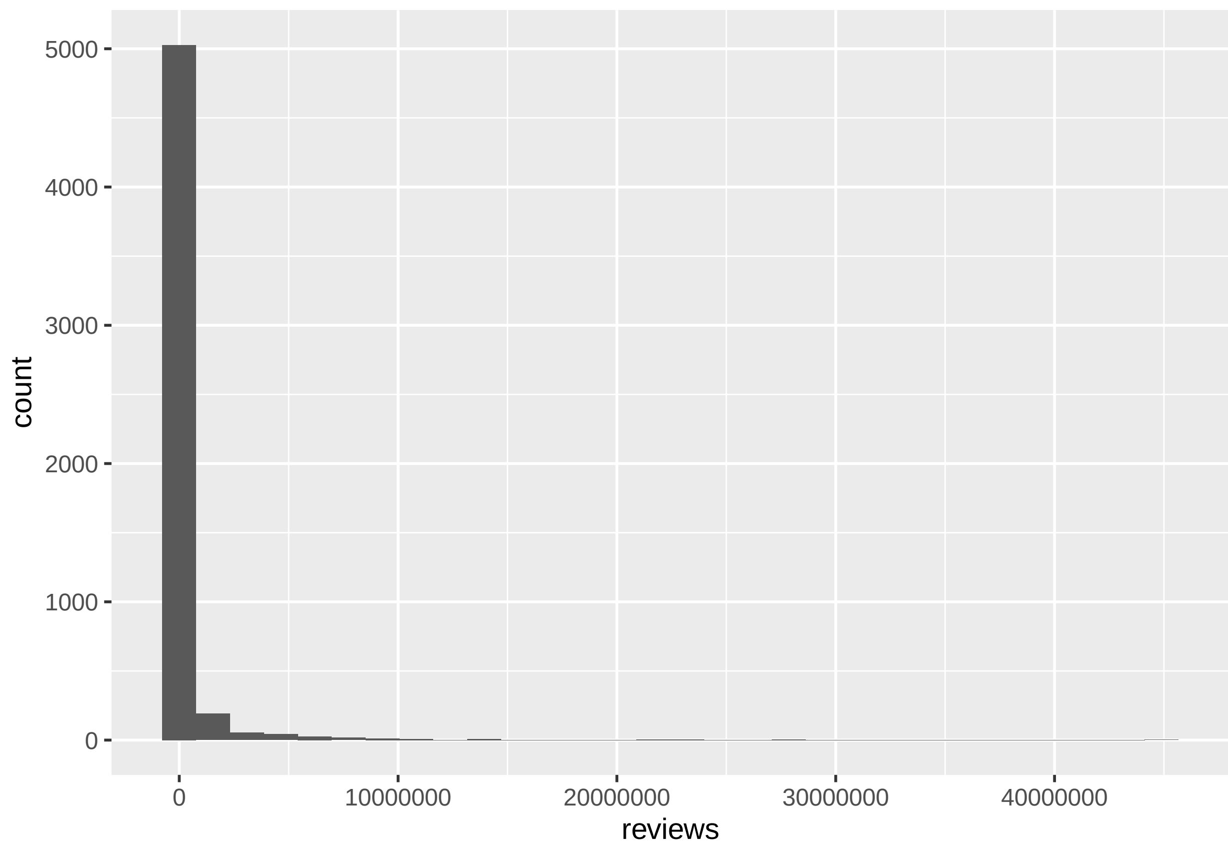

```r
ggplot(data = d, aes(x = log(install_group))) +
  geom_histogram()
```

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



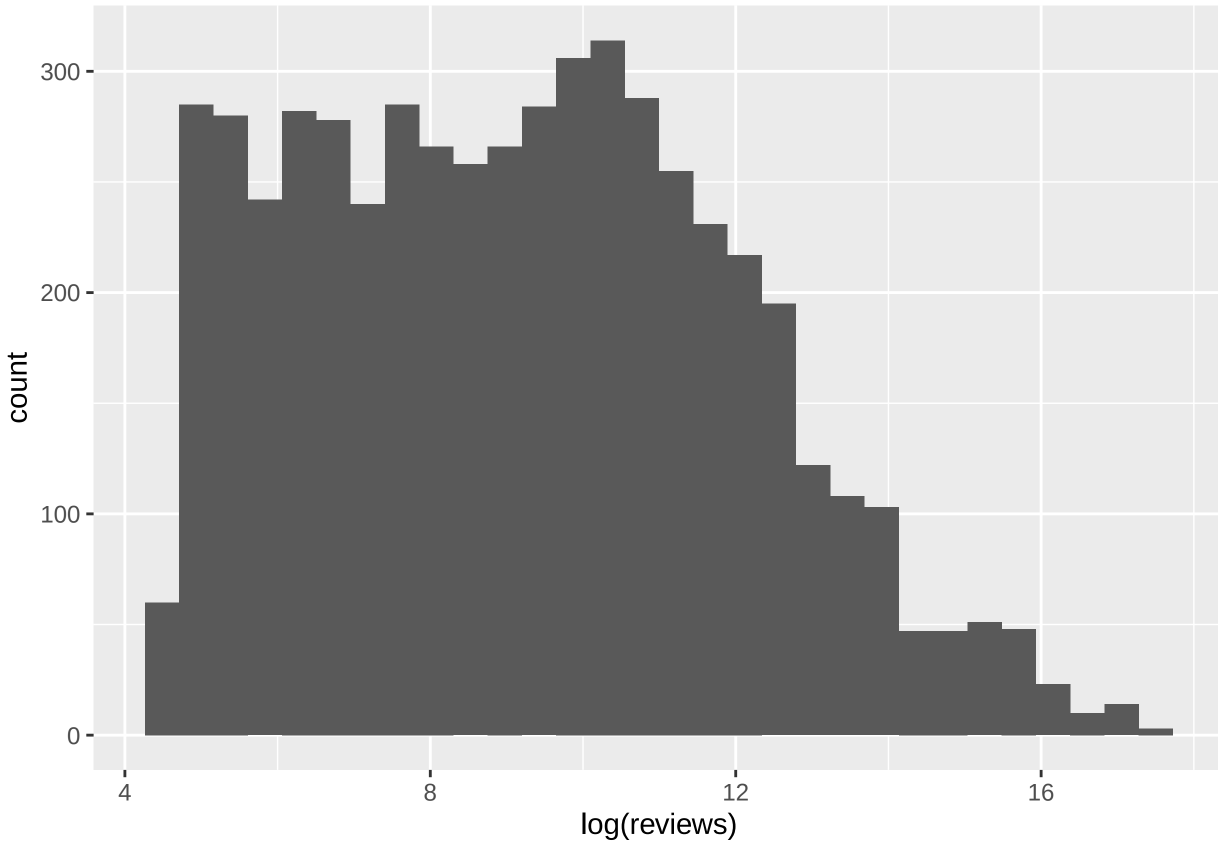

```
ggplot(data = d, aes(x = reviews)) +
  geom_histogram()
```

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



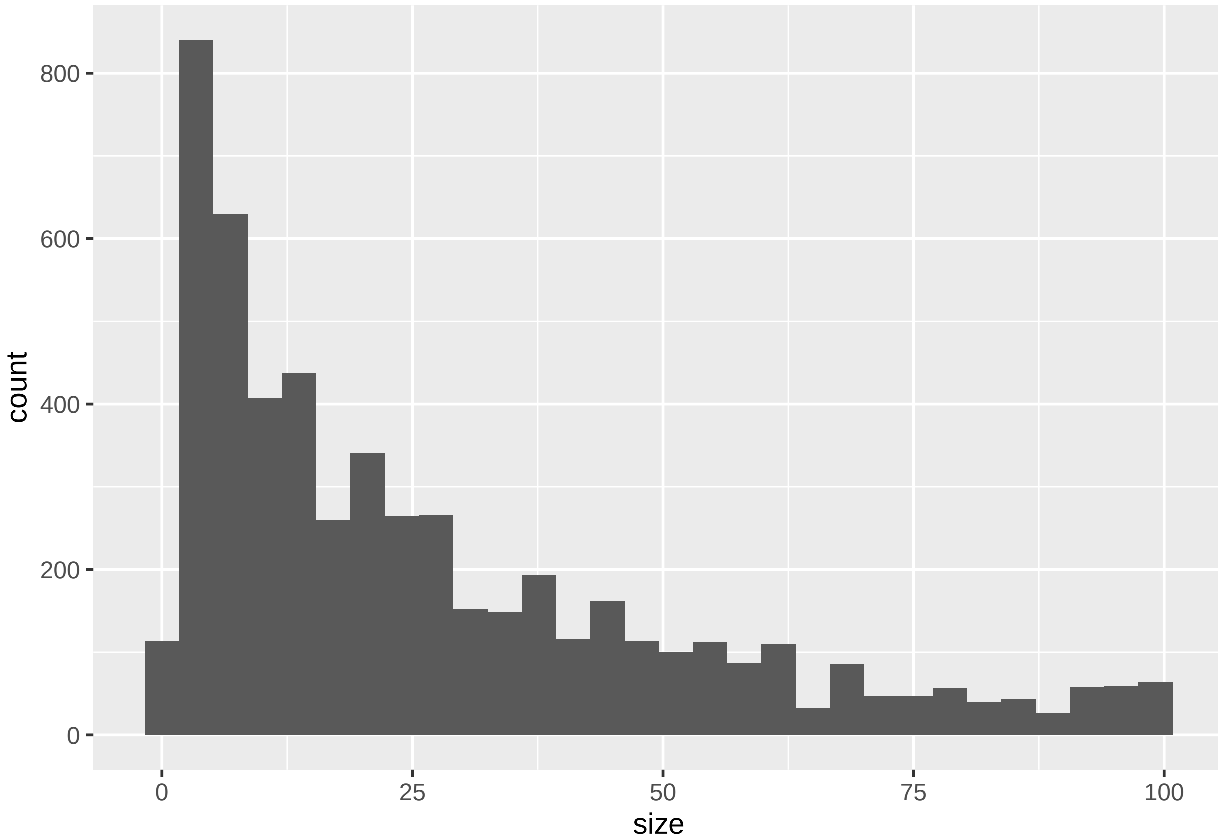

```r
ggplot(data = d, aes(x = log(reviews))) +
  geom_histogram()
```

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



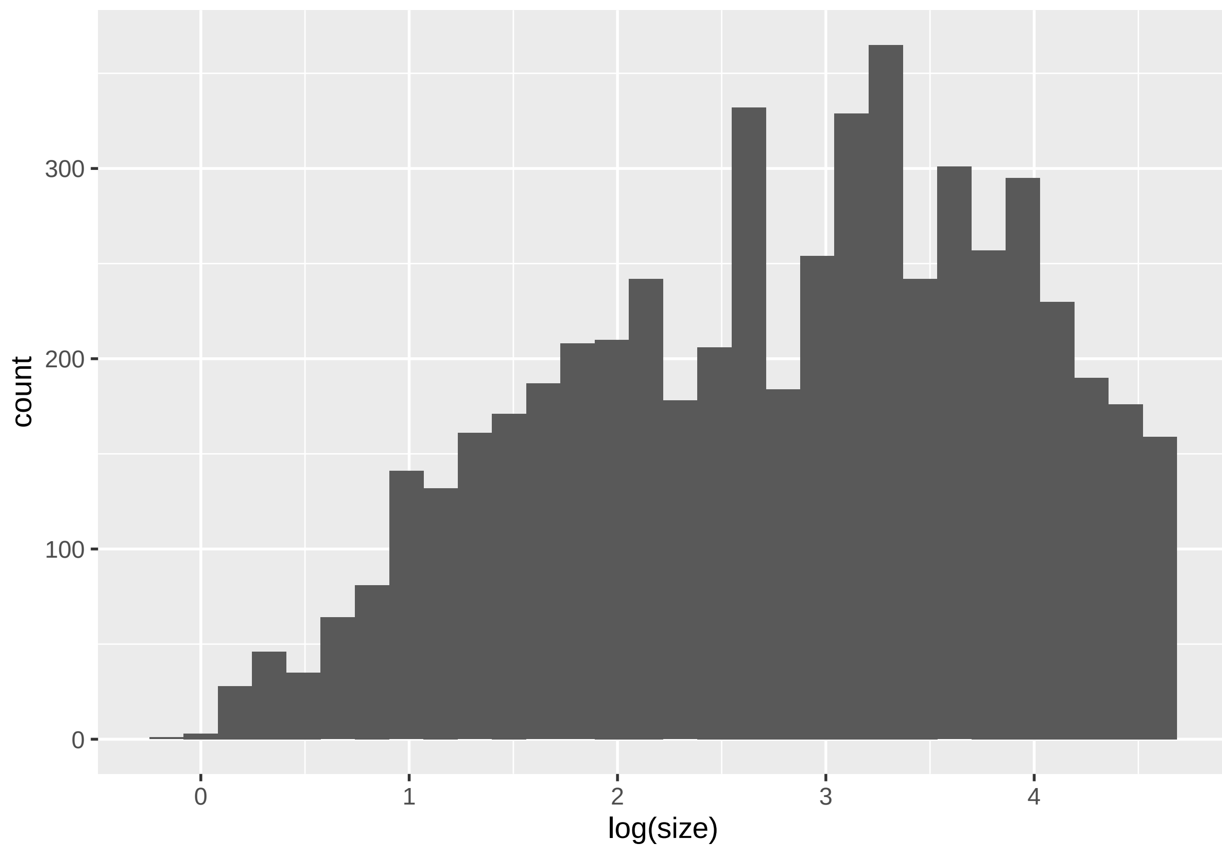

```
ggplot(data = d, aes(x = size)) +
  geom_histogram()
```

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



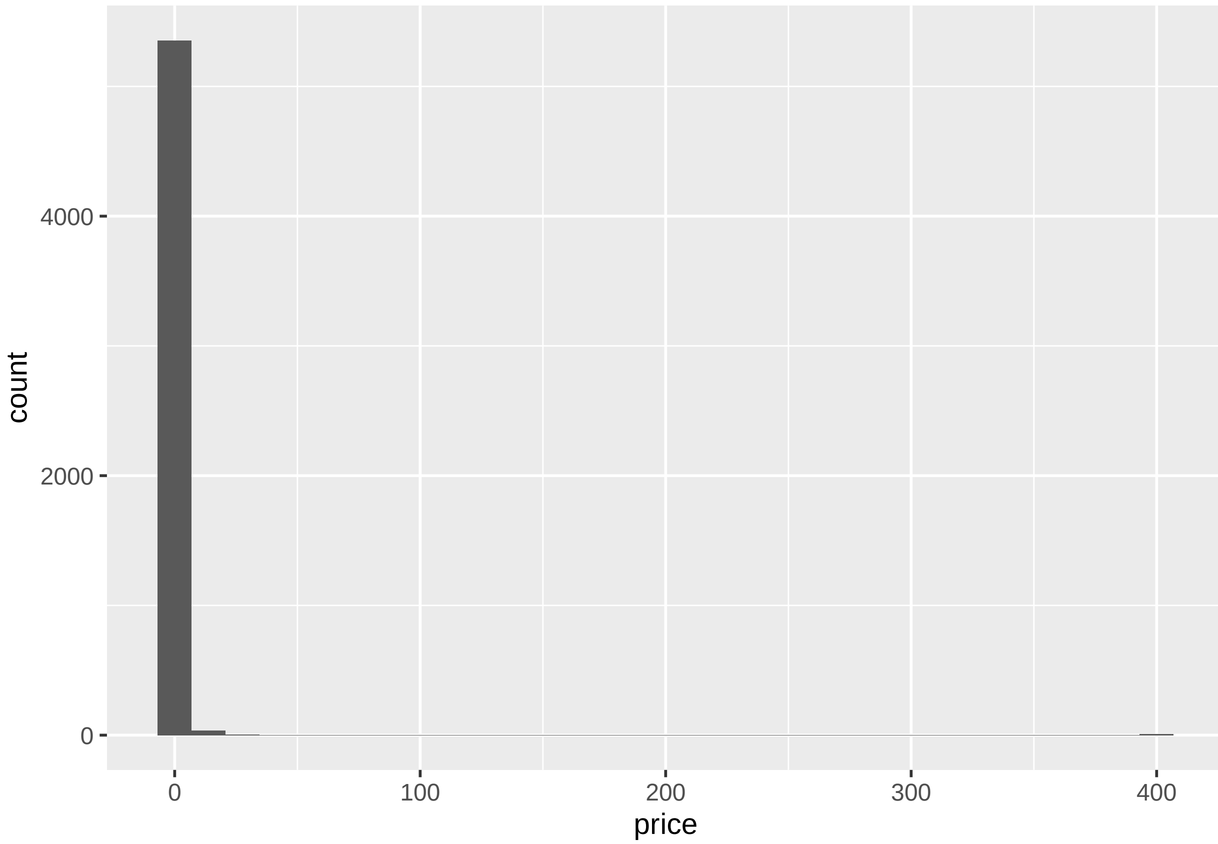

```
ggplot(data = d, aes(x = log(size))) +
  geom_histogram()
```

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




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

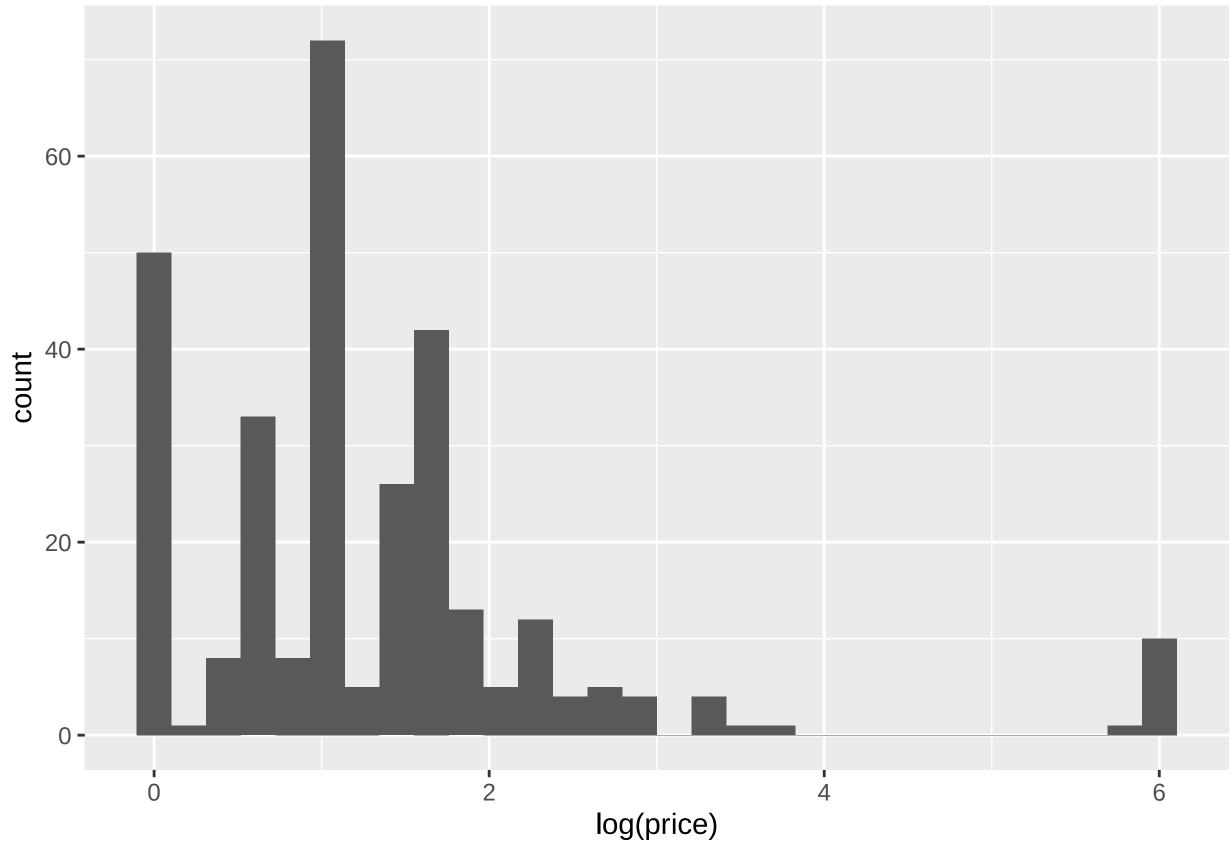
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




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

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

```
## Warning: Removed 5103 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        854          8.553864           5000000
##  2:        PHOTOGRAPHY        182          8.456044           1000000
##  3:           SHOPPING        132          8.196970           1000000
##  4:      COMMUNICATION        144          7.972222           1000000
##  5:       PRODUCTIVITY        156          7.474359           1000000
##  6:             SOCIAL        123          7.398374           1000000
##  7:             SPORTS        195          7.158974           1000000
##  8: HEALTH_AND_FITNESS        150          7.140000            500000
##  9: NEWS_AND_MAGAZINES        108          7.083333           1000000
## 10:    PERSONALIZATION        184          7.027174            750000
```

```
## 11:          FAMILY       1163          7.015477            500000
## 12:           TOOLS        412          7.002427            500000
## 13:       LIFESTYLE        171          6.520468            100000
## 14:          DATING        108          6.518519            500000
## 15:        BUSINESS        114          6.447368            100000
## 16:         FINANCE        186          6.107527            100000
## 17:         MEDICAL        125          5.408000            100000
## 18:            Free       5103          7.461885           1000000
## 19:            Paid        305          4.419672             10000
## 20:    Everyone 10+        269          8.189591           1000000
## 21:            Teen        685          7.854015           1000000
## 22:      Mature 17+        271          7.236162            500000
## 23:        Everyone       4180          7.144019            500000
## 24:             1.4        175          7.708571           1000000
## 25:             3.1        131          7.610687           1000000
## 26:             1.7        107          7.560748           1000000
## 27:             2.2        138          7.557971           1000000
## 28:             1.5        129          7.054264            500000
## 29:             1.2        320          7.046875            500000
## 30:             1.1        403          7.019851            500000
## 31:             1.6        106          6.962264            500000
## 32:             2.1        207          6.888889            500000
## 33:             1.3        207          6.879227            500000
## 34:               3        128          6.804688            100000
## 35:               2        211          6.625592            500000
## 36:               1        641          6.262090            100000
## 37:             4.1       1417          7.783345           1000000
## 38:               4       1616          7.321782           1000000
## 39:             4.4        572          7.295455           1000000
## 40:             4.2        230          7.265217           1000000
## 41:               5        324          7.166667           1000000
## 42:             4.3        126          7.039683            500000
## 43:             2.3        611          6.978723            500000
## 44:               3        137          6.583942            100000
## 45:             2.2        124          5.750000            100000
##     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
```

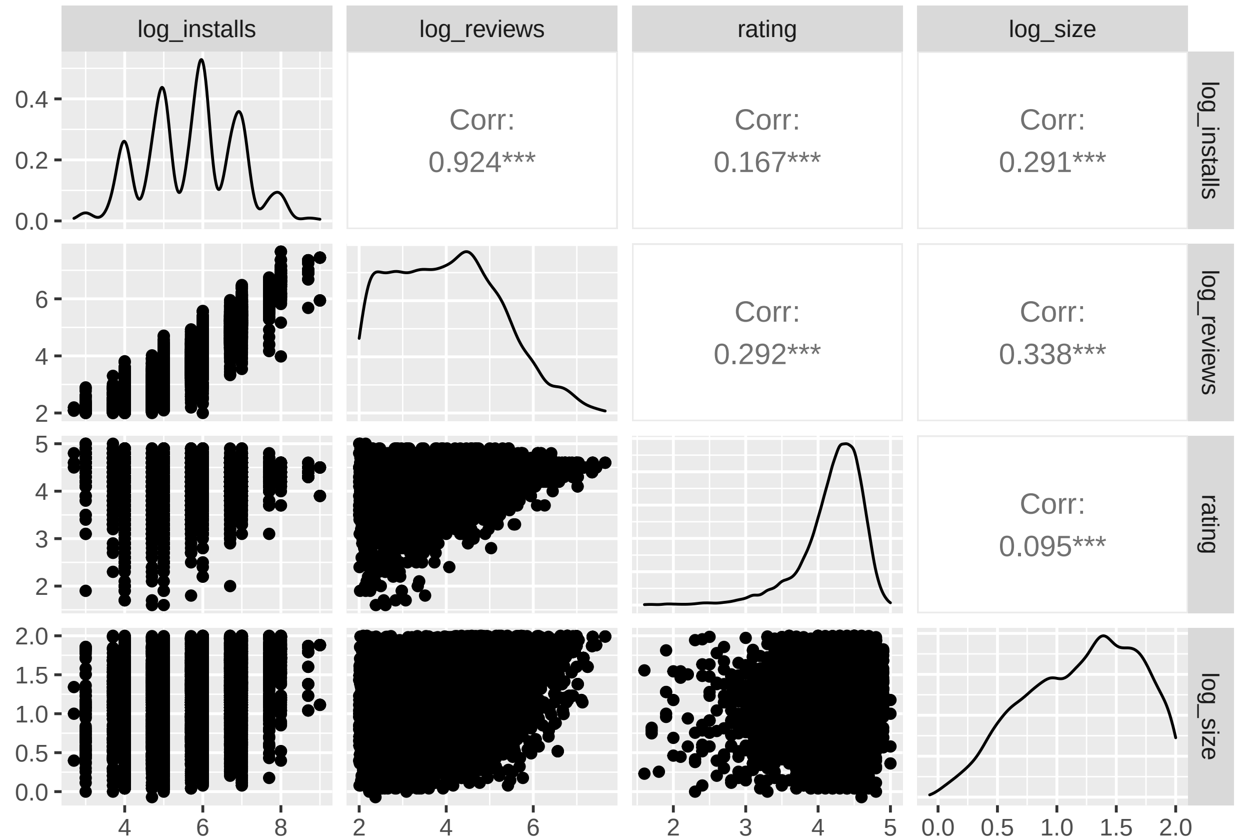
```
## 18:            type
## 19:            type
## 20:  content_rating
## 21:  content_rating
## 22:  content_rating
## 23:  content_rating
## 24: current_version
## 25: current_version
## 26: current_version
## 27: current_version
## 28: current_version
## 29: current_version
## 30: current_version
## 31: current_version
## 32: current_version
## 33: current_version
## 34: current_version
## 35: current_version
## 36: current_version
## 37: android_version
## 38: android_version
## 39: android_version
## 40: android_version
## 41: android_version
## 42: android_version
## 43: android_version
## 44: android_version
## 45: android_version
##            variable
```

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

```
##          0%      25%      50%      75%     100%        variable diff_min_vs_max
## 1: 5.408000 6.520468 7.083333 7.474359 8.553864        category        3.145864
## 2: 4.419672 5.180225 5.940779 6.701332 7.461885            type        3.042213
## 3: 7.144019 7.213127 7.545088 7.937909 8.189591  content_rating        1.045572
## 4: 6.262090 6.879227 7.019851 7.557971 7.708571 current_version        1.446481
## 5: 5.750000 6.978723 7.166667 7.295455 7.783345 android_version        2.033345
```

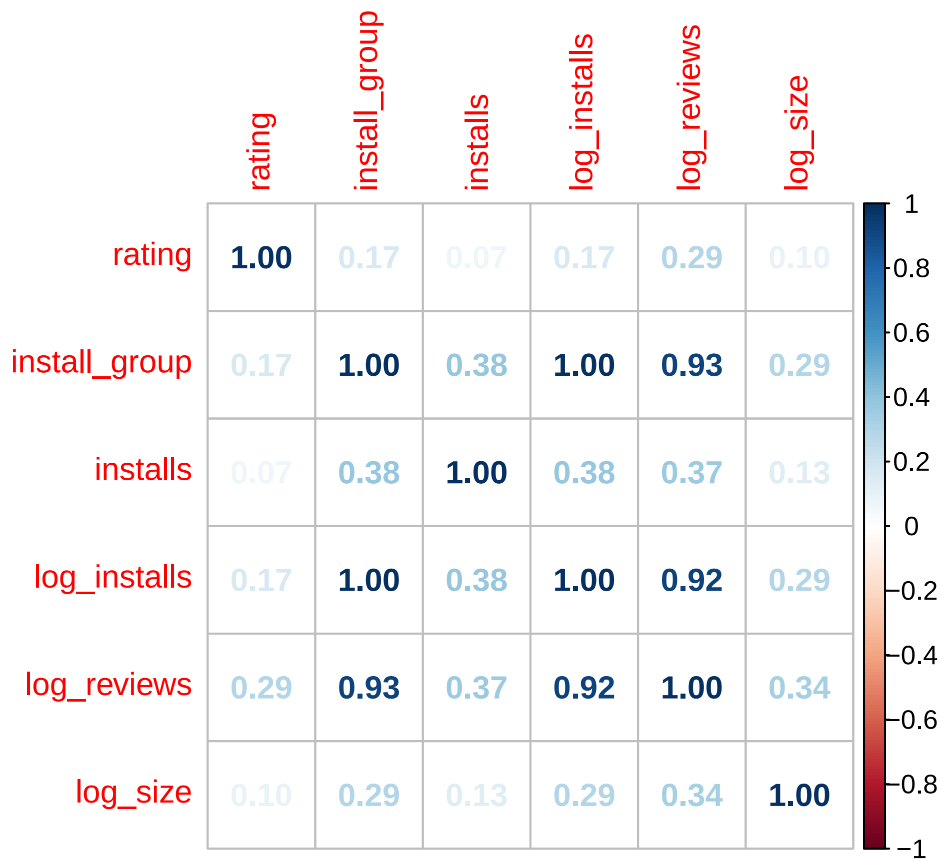
I.I.D. data: According to the Kaggle authors, this data set was collected by randomly scraping the Google Play Store. Since no clusters of applications were specifically targeted, we can reasonably use the entirety of the store as our reference population. We recognize that applications likely have some 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 (5408 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.

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




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

```r
cols <- c('rating','install_group','installs', '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 >= 6,cols], use = "complete.obs"),
         method = 'number')
```




```
# Corrplot across variables (25th PCTL outliers removed)
corrplot(cor(d[d$reviews >= 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 + rating + log_size, data = d)
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                     0.879***                   0.918***                   0.889***
##                                 (0.005)                    (0.005)                    (0.005)
##
## rating                                                    -0.302***                  -0.253***
```

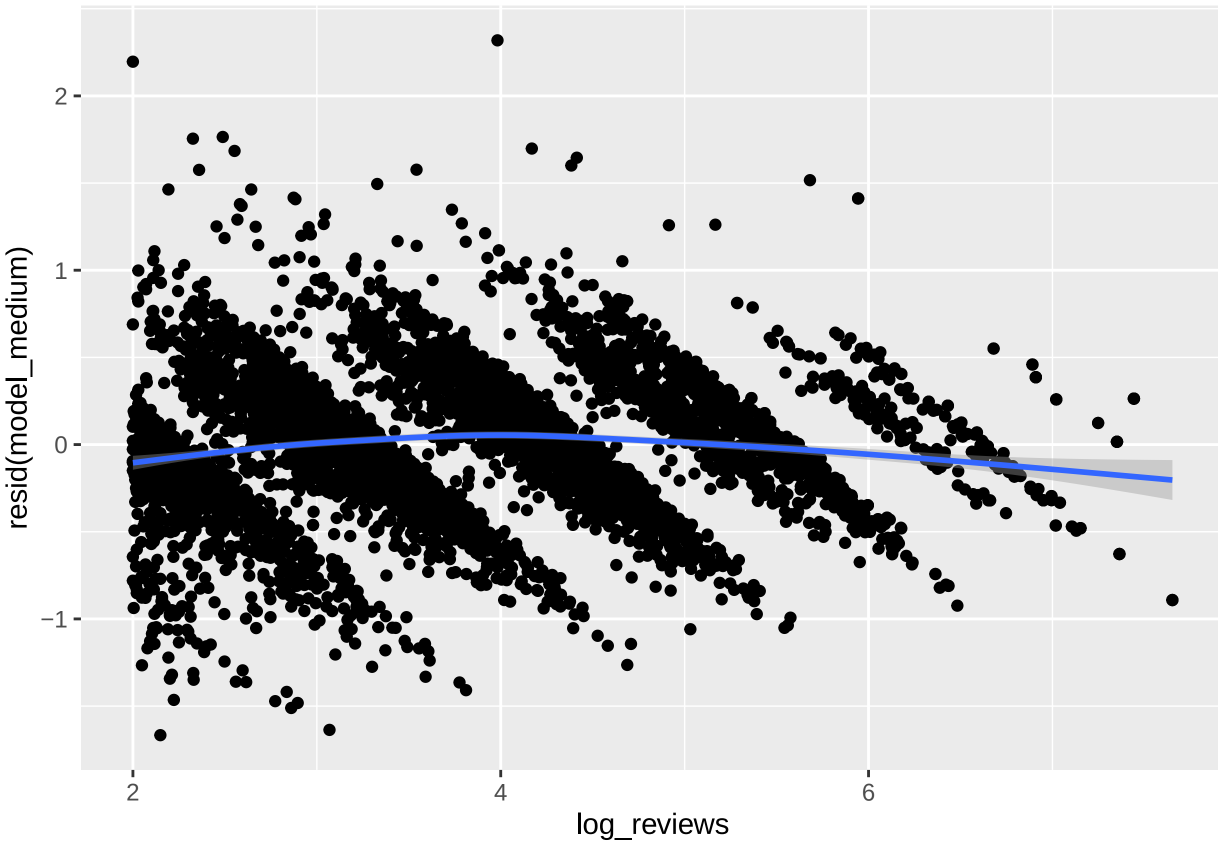
```
##                                                    (0.017)                   (0.013)
##
## log_size                                         -0.061***                 -0.049***
##                                                    (0.013)                   (0.012)
##
## factor(type)Paid                                                           -0.720***
##                                                                              (0.024)
##
## Constant                        2.194***          3.379***                  3.314***
##                                 (0.021)           (0.069)                   (0.053)
##
## ---------------------------------------------------------------------------------------------
## Observations                     5,408             5,408                     5,408
## R2                               0.854             0.866                     0.886
## Adjusted R2                      0.854             0.866                     0.885
## Residual Std. Error      0.443 (df = 5406)     0.425 (df = 5404)         0.392 (df = 5403)
## F Statistic         31,576.970*** (df = 1; 5406) 11,635.130*** (df = 3; 5404) 10,452.460*** (df = 4;
## =============================================================================================
## Note:                                                          *p<0.1; **p<0.05; ***p
```

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.

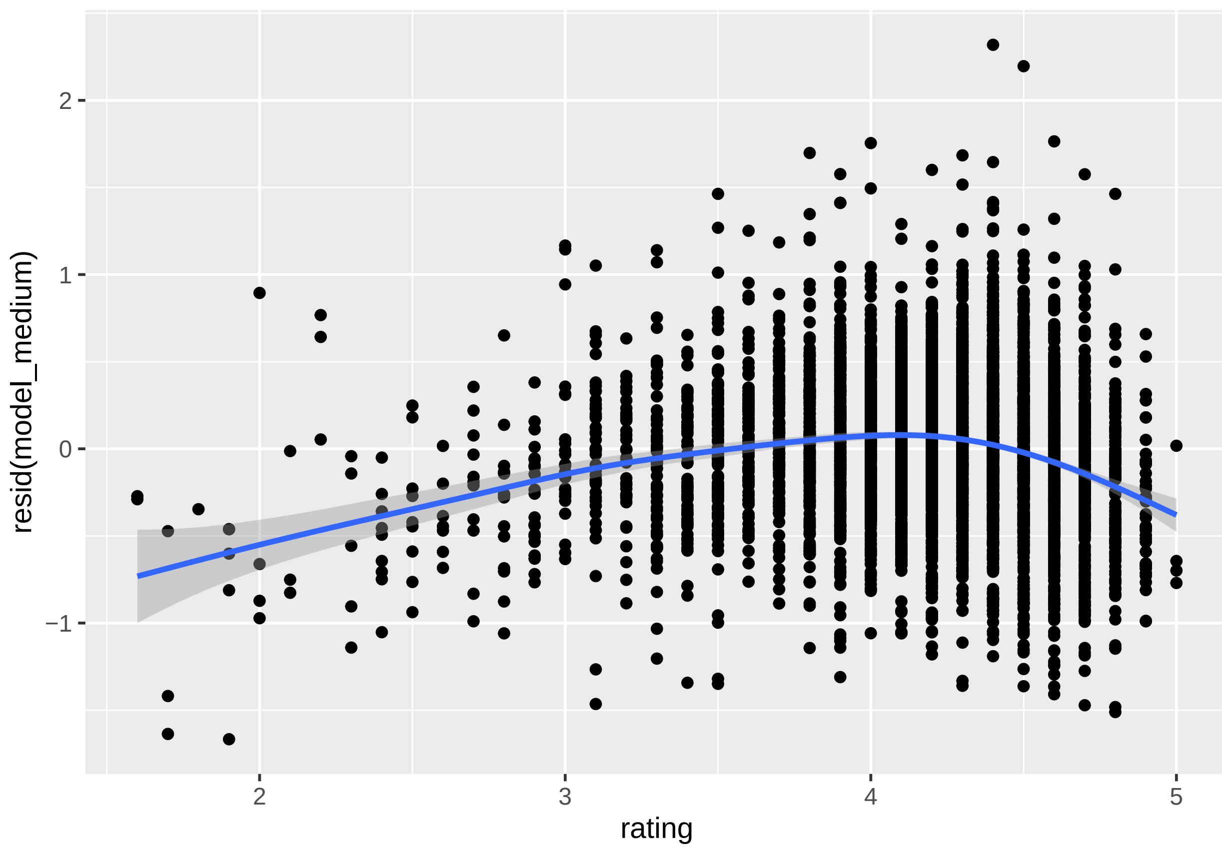
```
# Reviews versus residuals
plot_1 <- ggplot(data = d, mapping = aes(x = log_reviews, y = resid(model_medium))) +
  geom_point() + stat_smooth()
plot_1
```

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



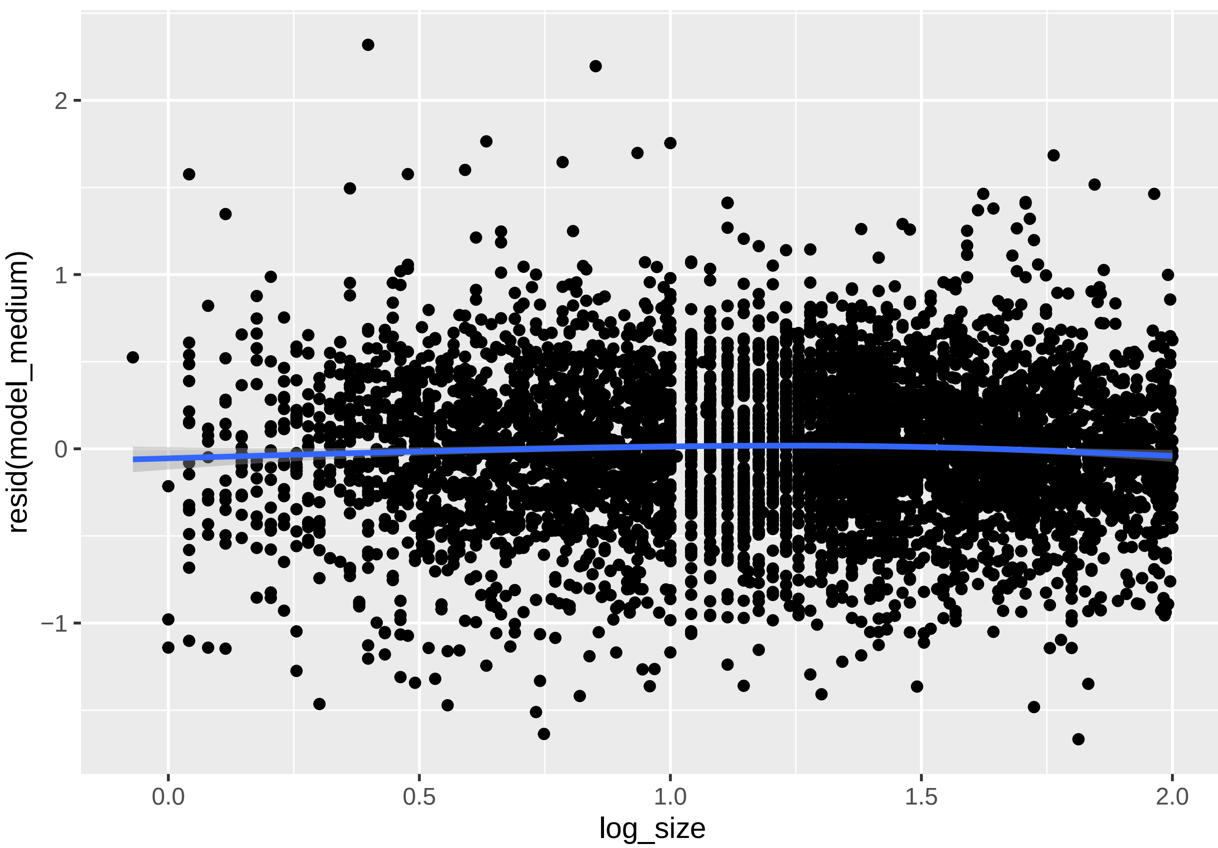

```r
# Ratings versus residuals
plot_2 <- ggplot(data = d, mapping = aes(x = rating, y = resid(model_medium))) +
  geom_point() + stat_smooth()
plot_2
```

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



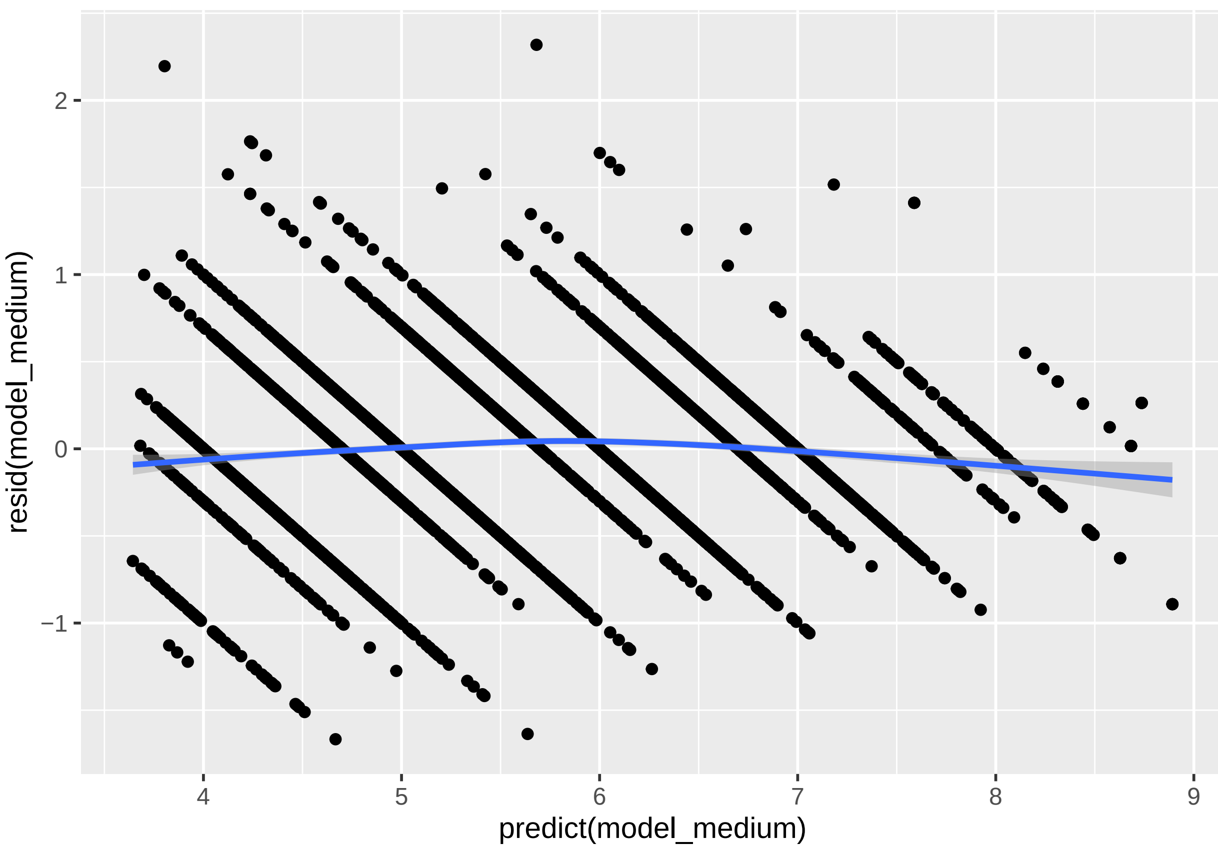

```
# Size versus residuals
plot_3 <- ggplot(data = d, mapping = aes(x = log_size, y = resid(model_medium))) +
  geom_point() + stat_smooth()
plot_3
```

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



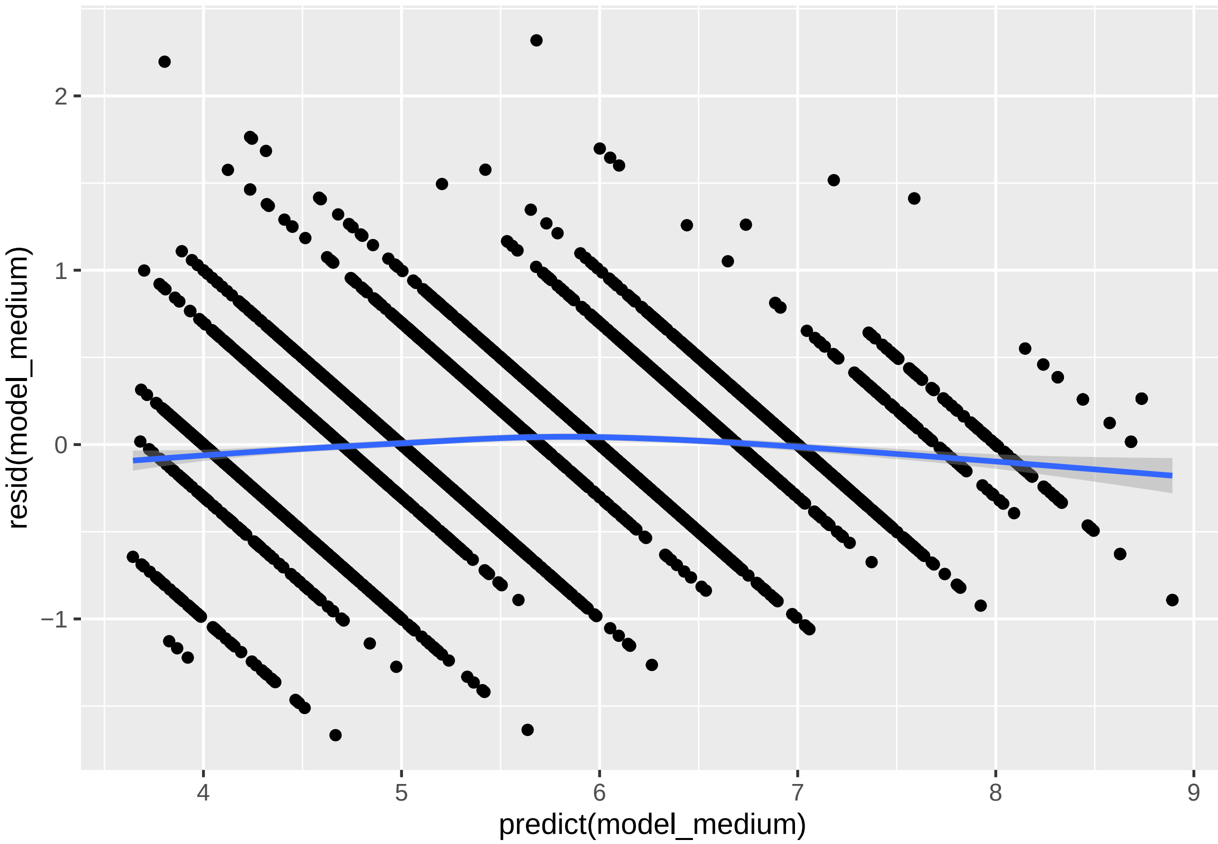
4. **Homoskedastic Errors:** When assessing homoskedastic errors, we seek to determine if there is a relationship between the variance of the model residuals and the predictors. If the homoskedastic assumption is satisfied, then we should observe a lack of relationship; conversely, if the data are heteroskedastic 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 = 107.58, df = 1, p-value < 0.00000000000000022
```

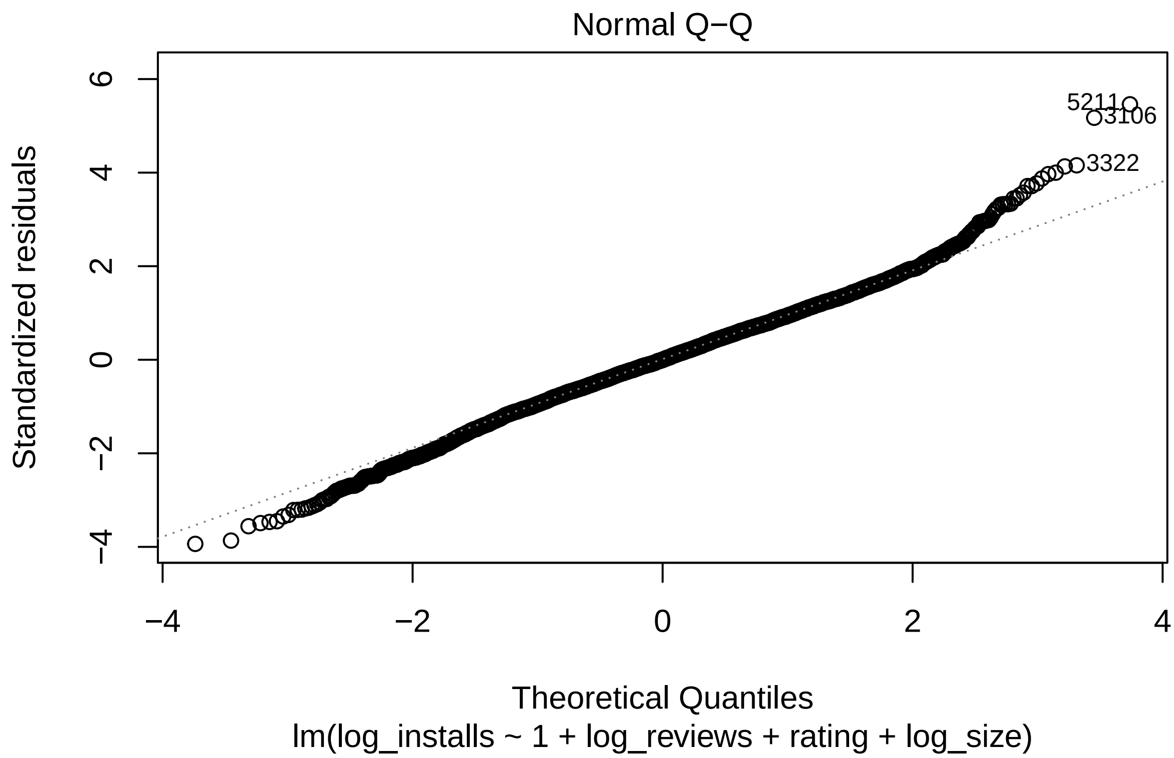
```
plot_4
```

```
## `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.

```r
# 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                         0.879***
##                                     (0.005)
##
## log_installs                                        0.972
##
##
## Constant                            2.194***      -1.548***
##                                     (0.021)        (0.069)
##
## ------------------------------------------------------------
## Observations                         5,408          5,408
## R2                                   0.854          0.854
## Adjusted R2                          0.854          0.854
## Residual Std. Error (df = 5406)      0.443          0.466
## F Statistic (df = 1; 5406)      31,576.970*** 31,576.970***
## ============================================================
## Note:                            *p<0.1; **p<0.05; ***p<0.01
```