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)

source('./get_robust_se.R')
```

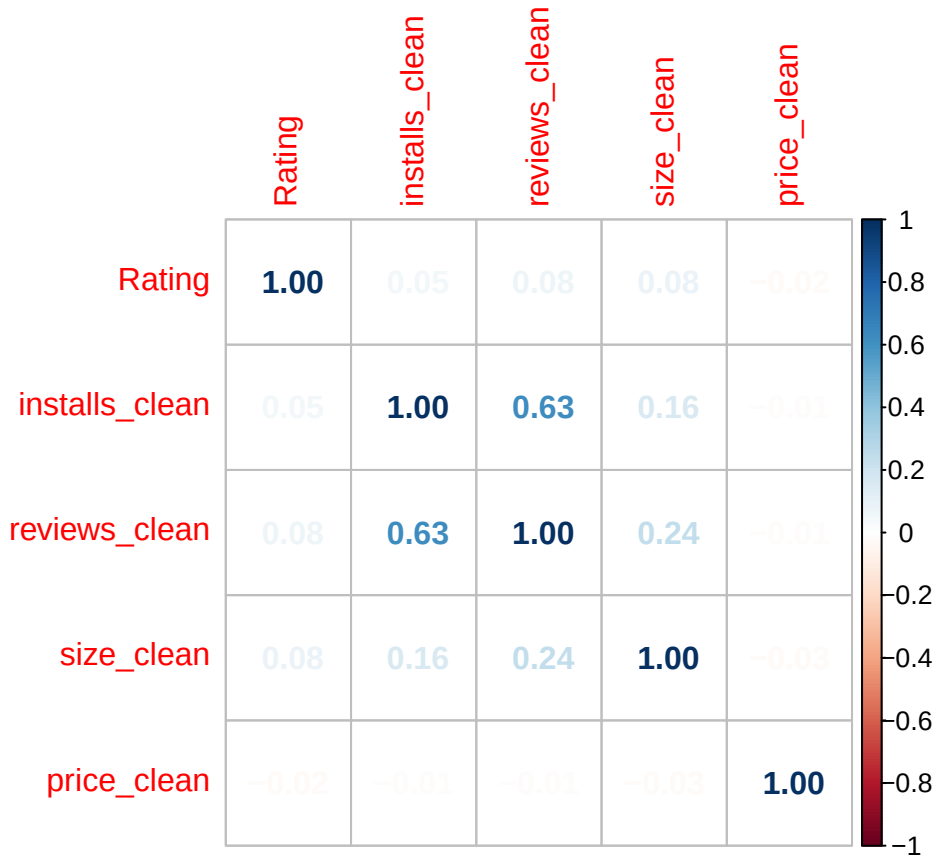
```
d <- read.csv('data/googleplaystore.csv')
summary(d)
```

```
##      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
##
##
##
##
##  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
##
```

```
##
##
##
```

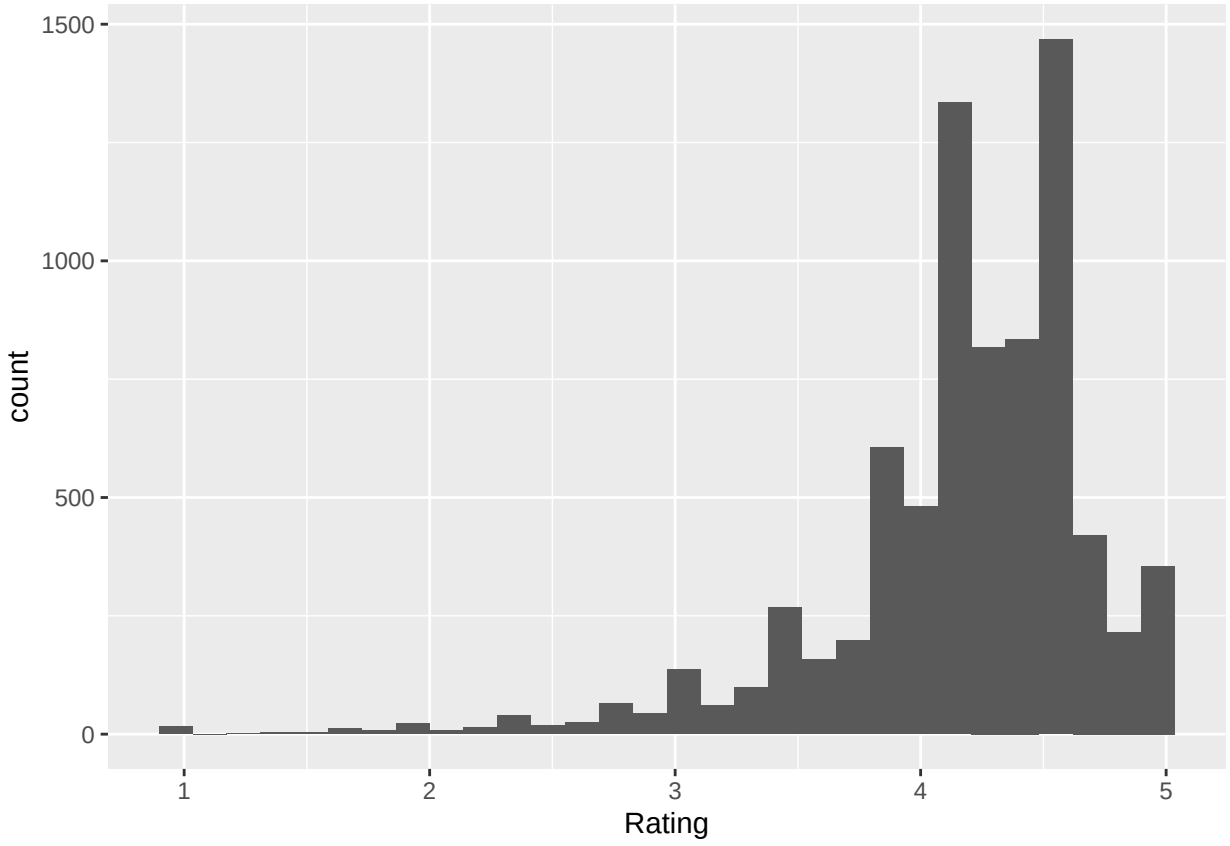
```
# Function to convert strings to numbers
str_to_num <- function(s) {
  n <- nchar(s)
  last <- substr(s, n, n + 1)
  if (last == "B") {
    number <- as.numeric(substr(s, 0, n - 1)) * 1.0e9
  } else if (last == "M") {
    number <- as.numeric(substr(s, 0, n - 1)) * 1.0e6
  } else if (last == "k")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

# Additional sanitizations
d$price_clean     <- as.numeric(gsub('\\$','',d$Price))
```

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

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



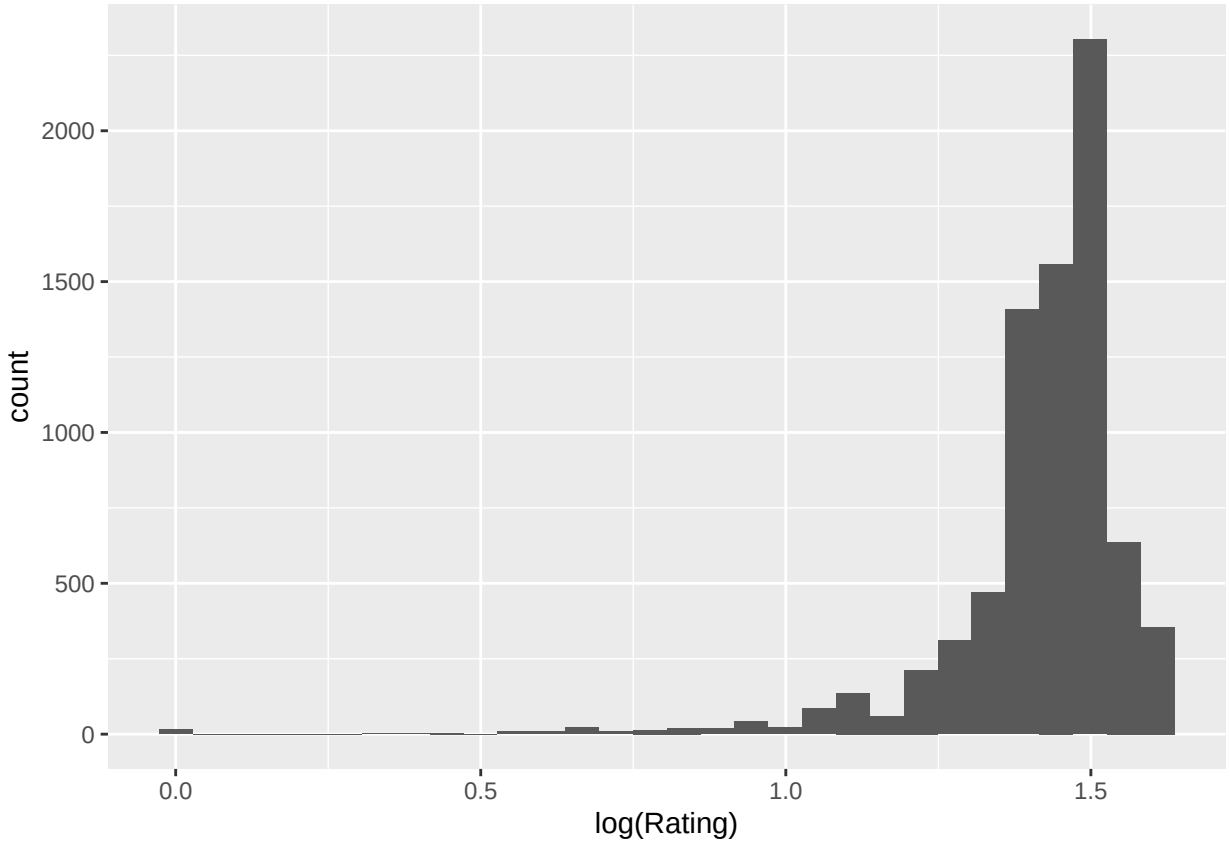

```
# Distribution of numeric columns
ggplot(data = d, aes(x = Rating)) +
  geom_histogram()
```

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



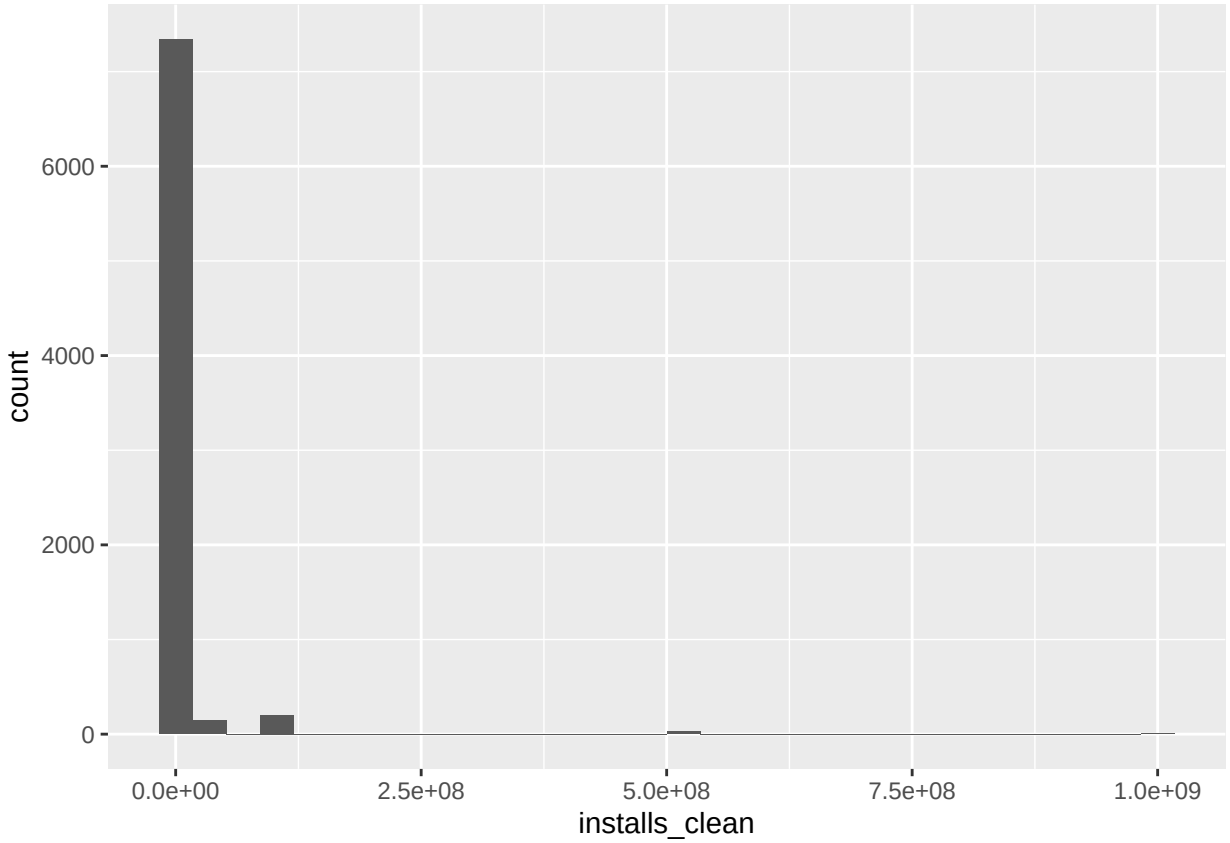

```
ggplot(data = d, aes(x = log(Rating))) +
  geom_histogram()
```

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



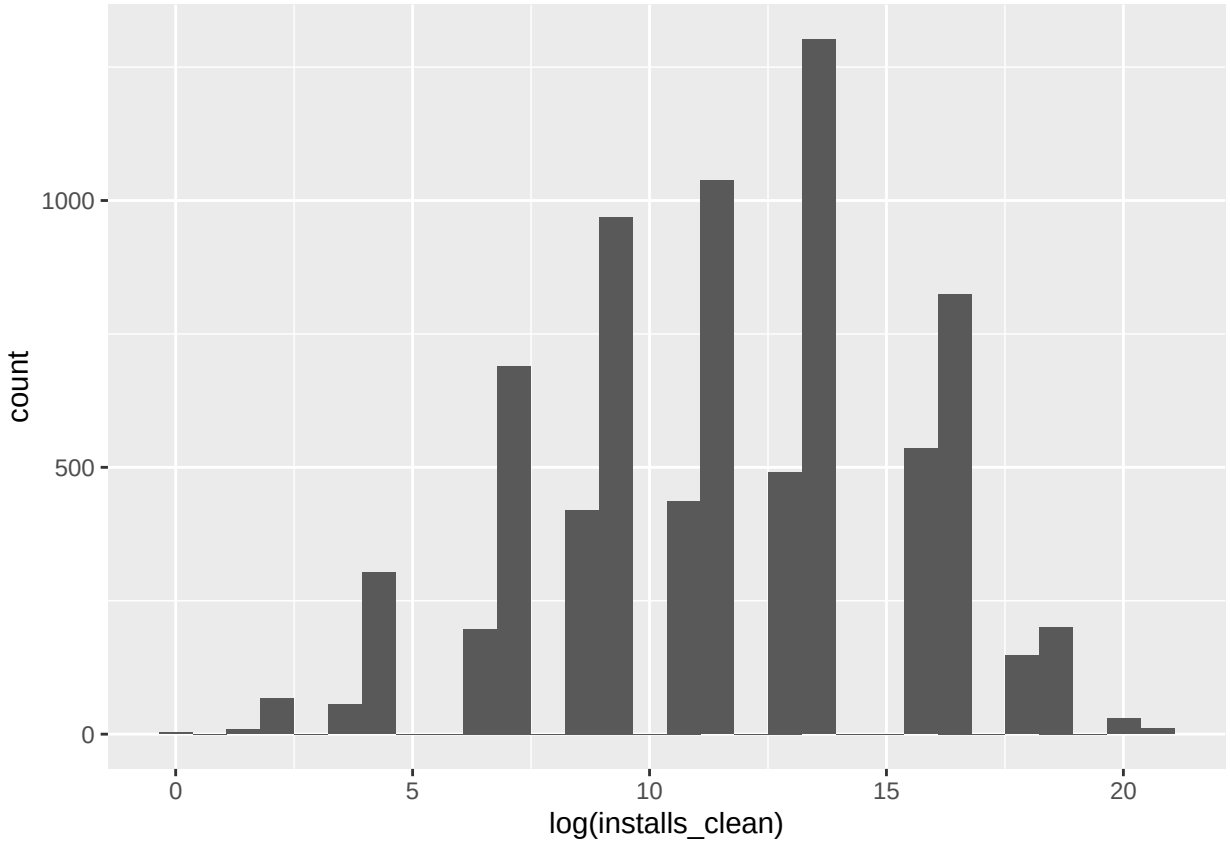

```
ggplot(data = d, aes(x = installs_clean)) +
  geom_histogram()
```

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



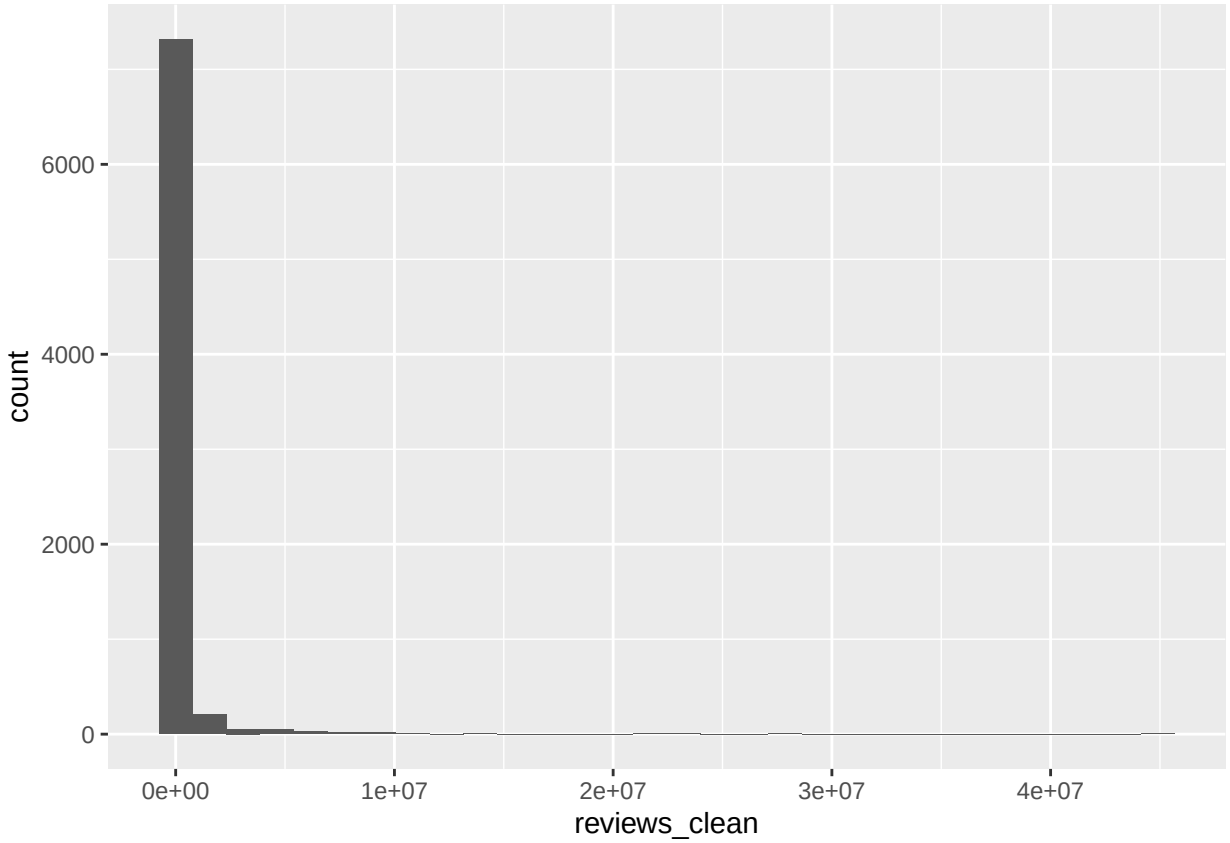

```
ggplot(data = d, aes(x = log(installs_clean))) +
  geom_histogram()
```

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



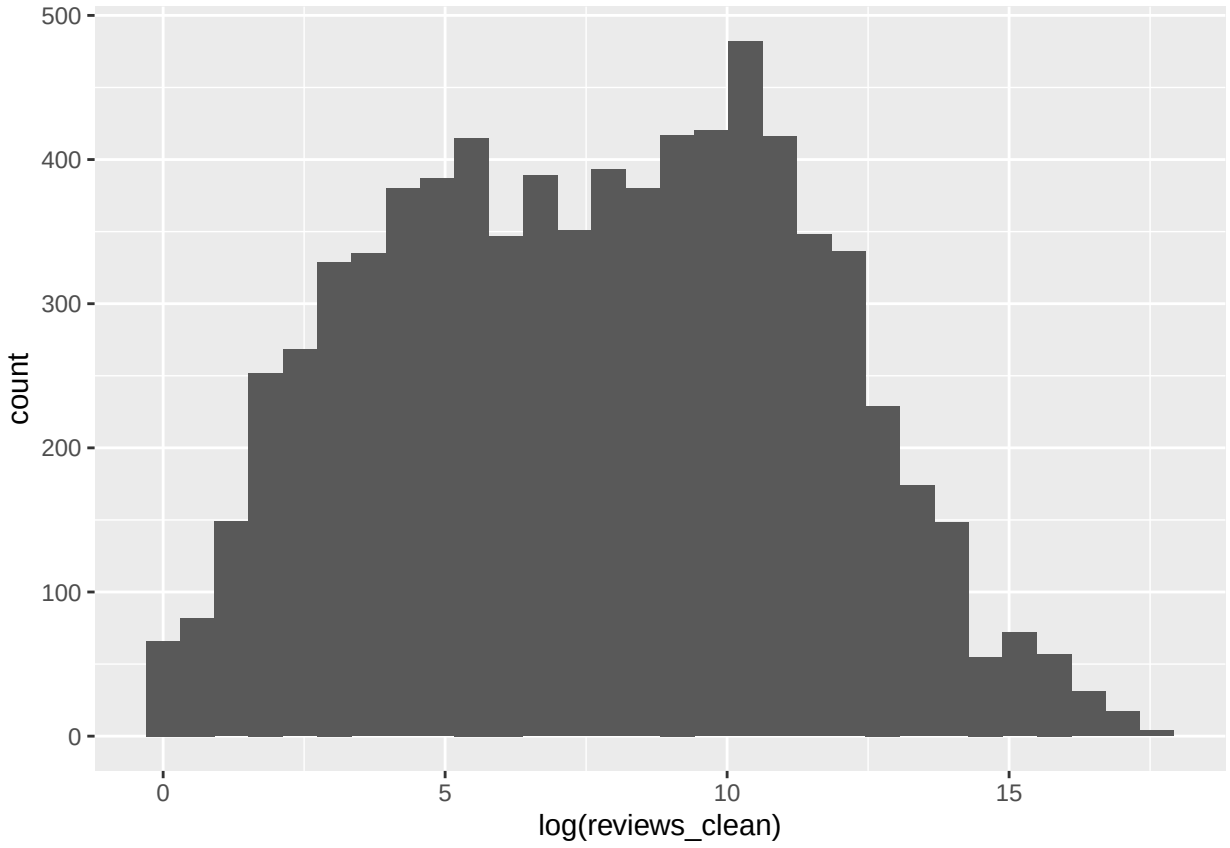

```
ggplot(data = d, aes(x = reviews_clean)) +
  geom_histogram()
```

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



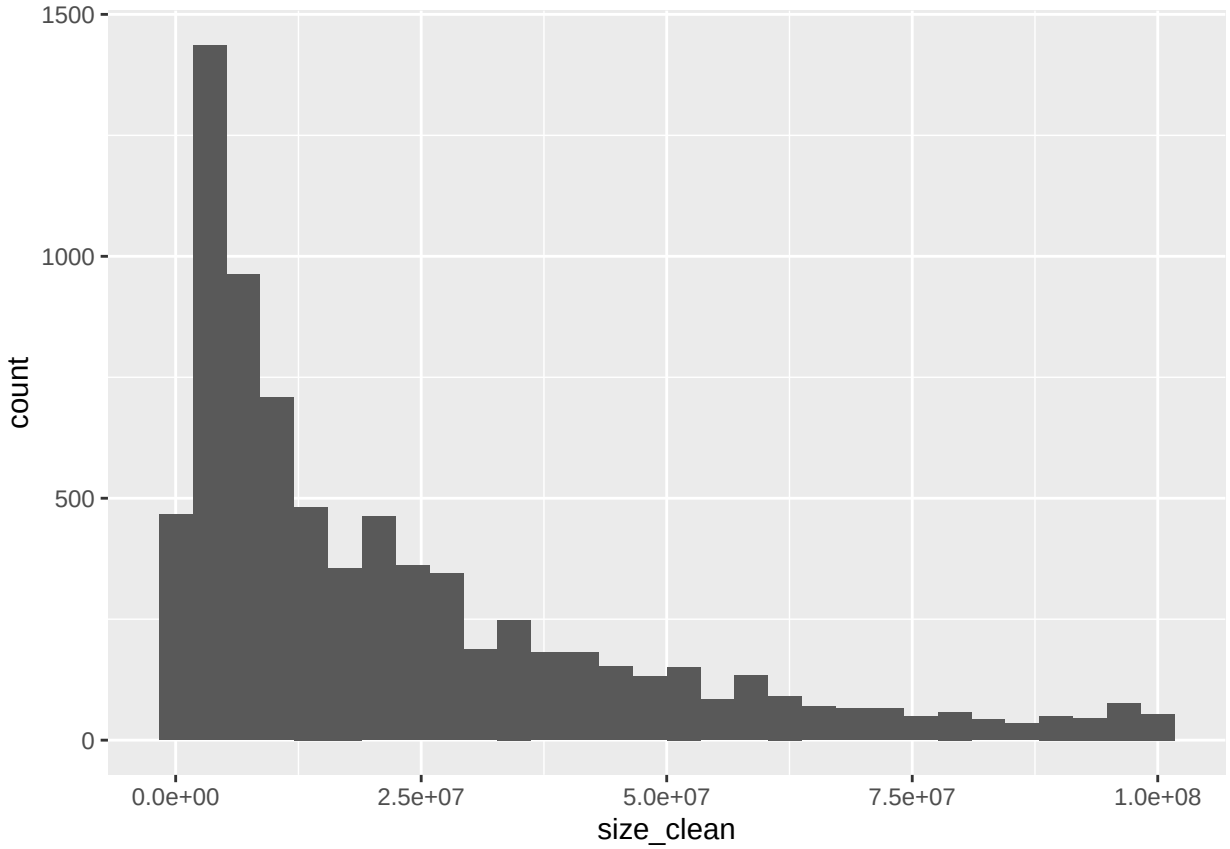

```
ggplot(data = d, aes(x = log(reviews_clean))) +
  geom_histogram()
```

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



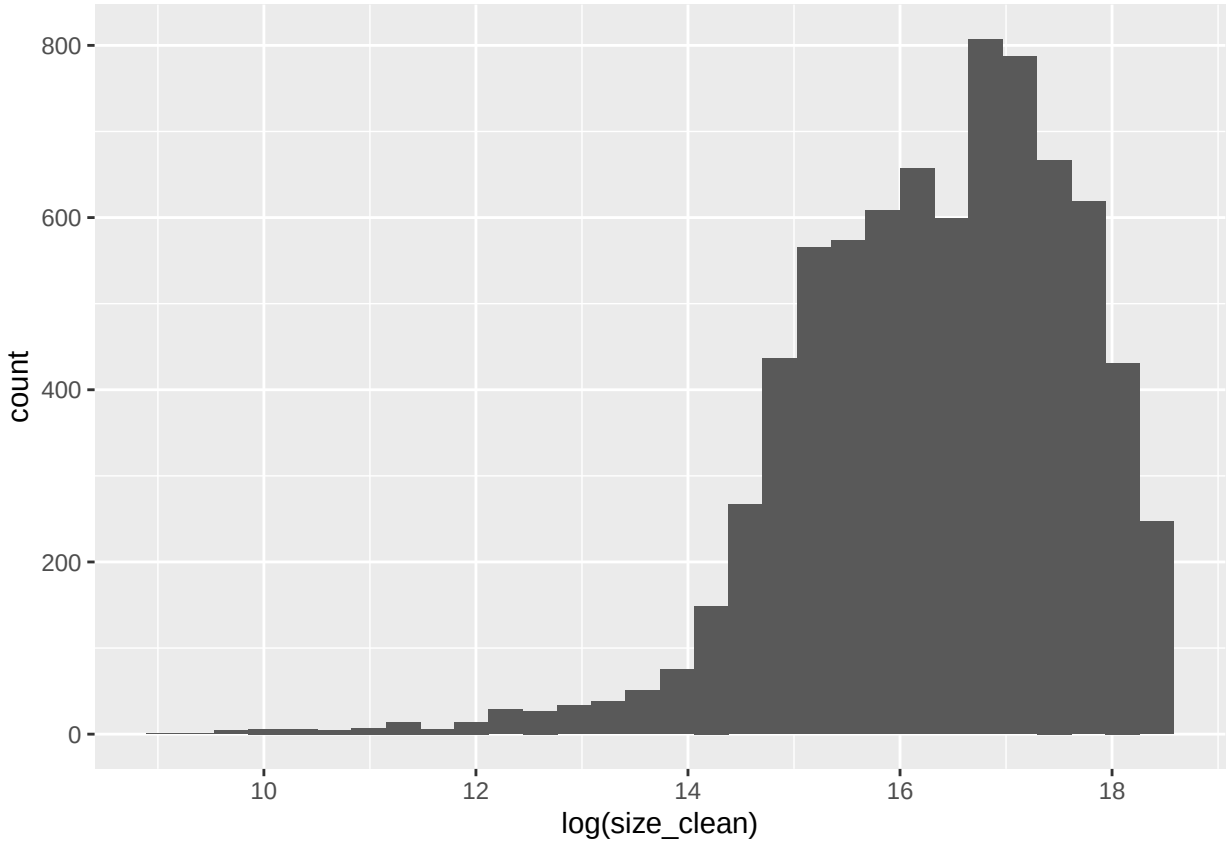

```
ggplot(data = d, aes(x = size_clean)) +
  geom_histogram()
```

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



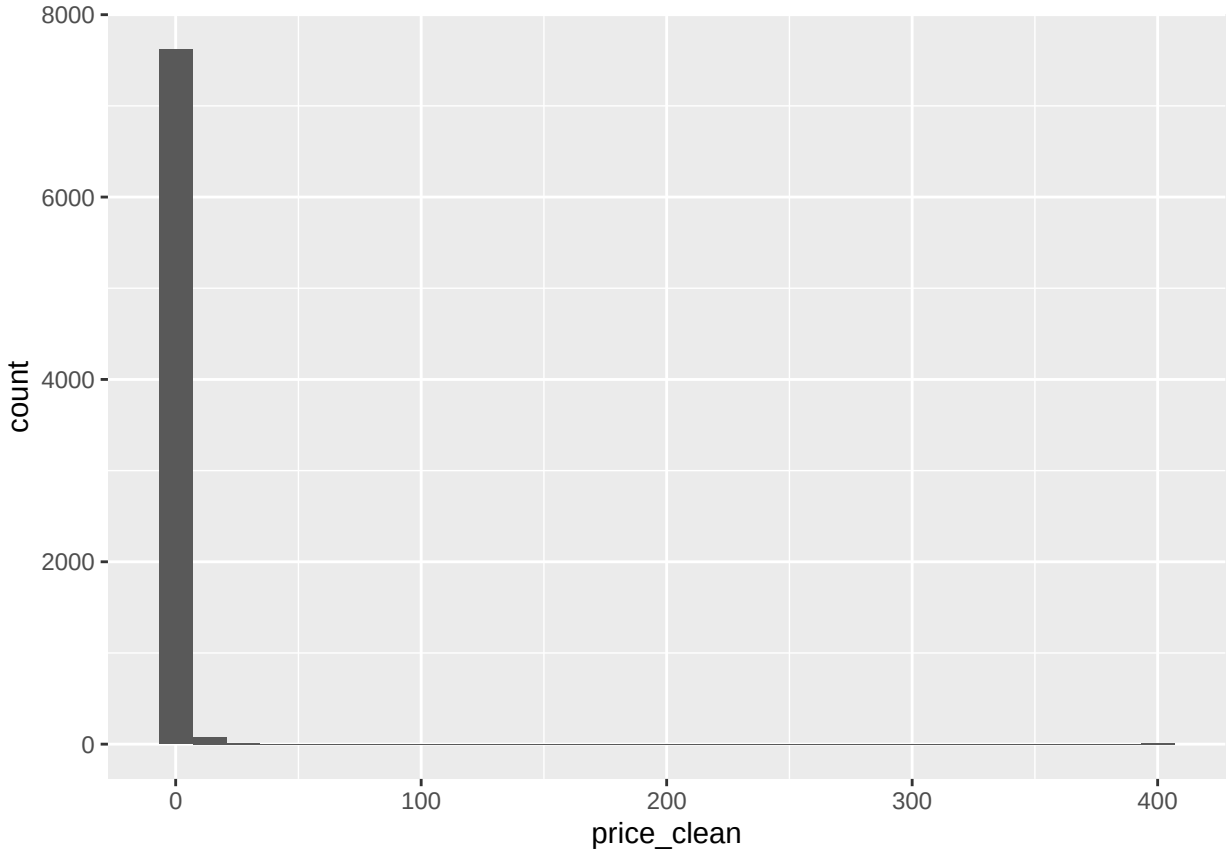

```
ggplot(data = d, aes(x = log(size_clean))) +
  geom_histogram()
```

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




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

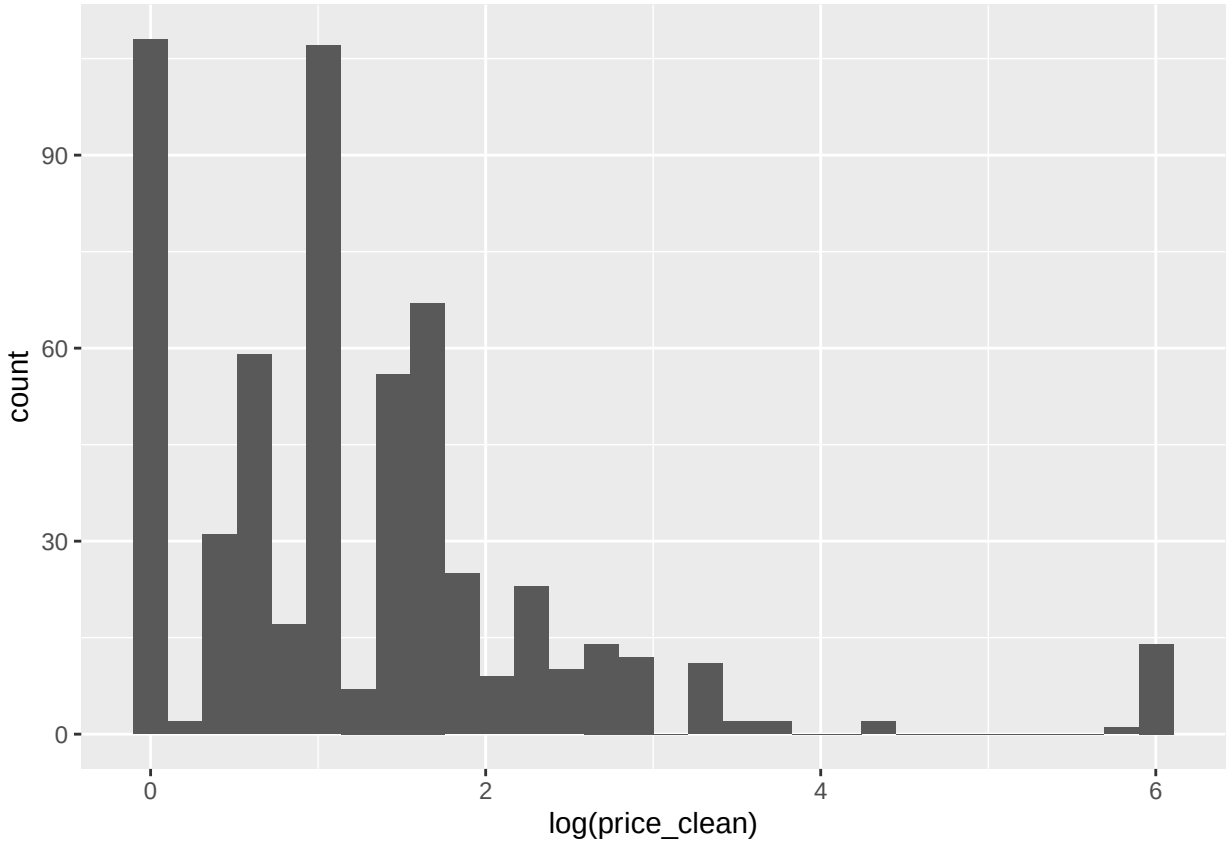
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




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

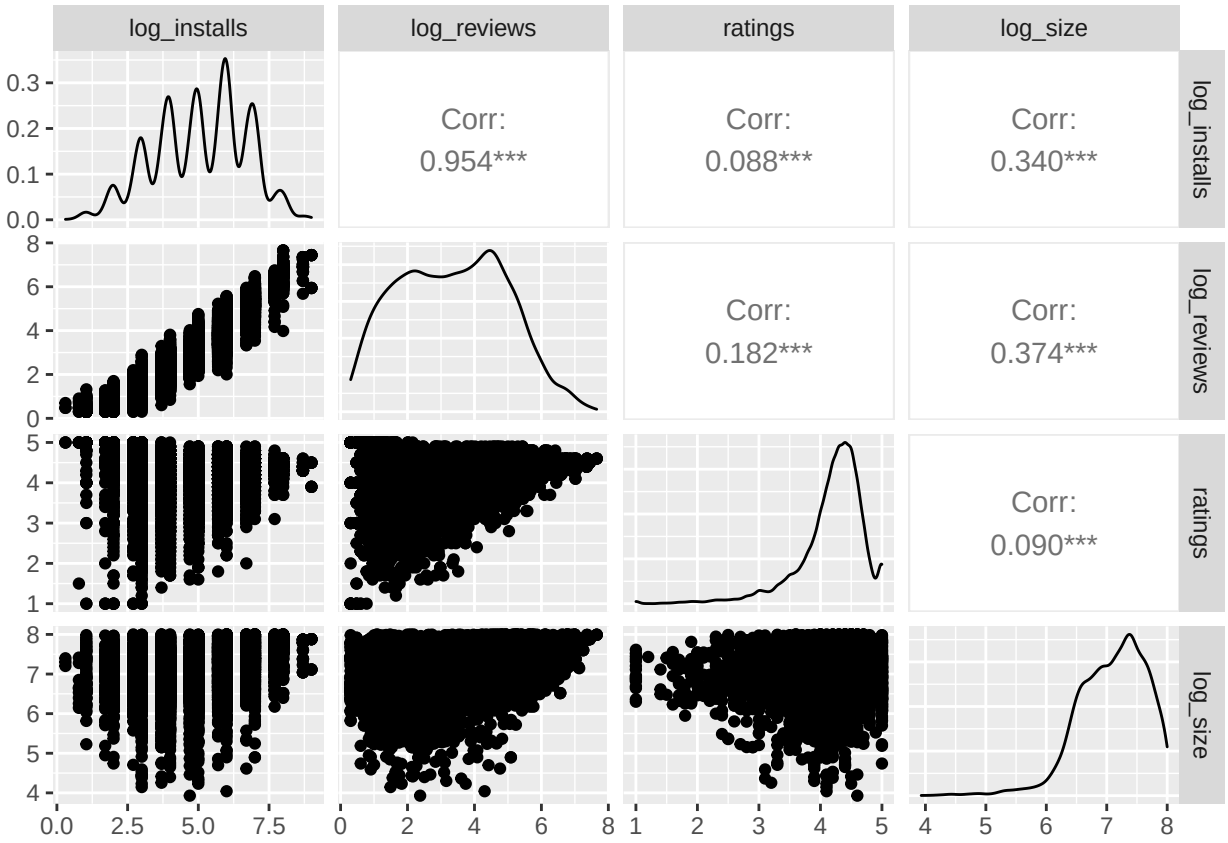
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 7150 rows containing non-finite values (stat_bin).
```



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])
```




```
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***
##                                                     (0.013)                  (0.009)
##
## log_size                                            -0.048***                -0.053***
```

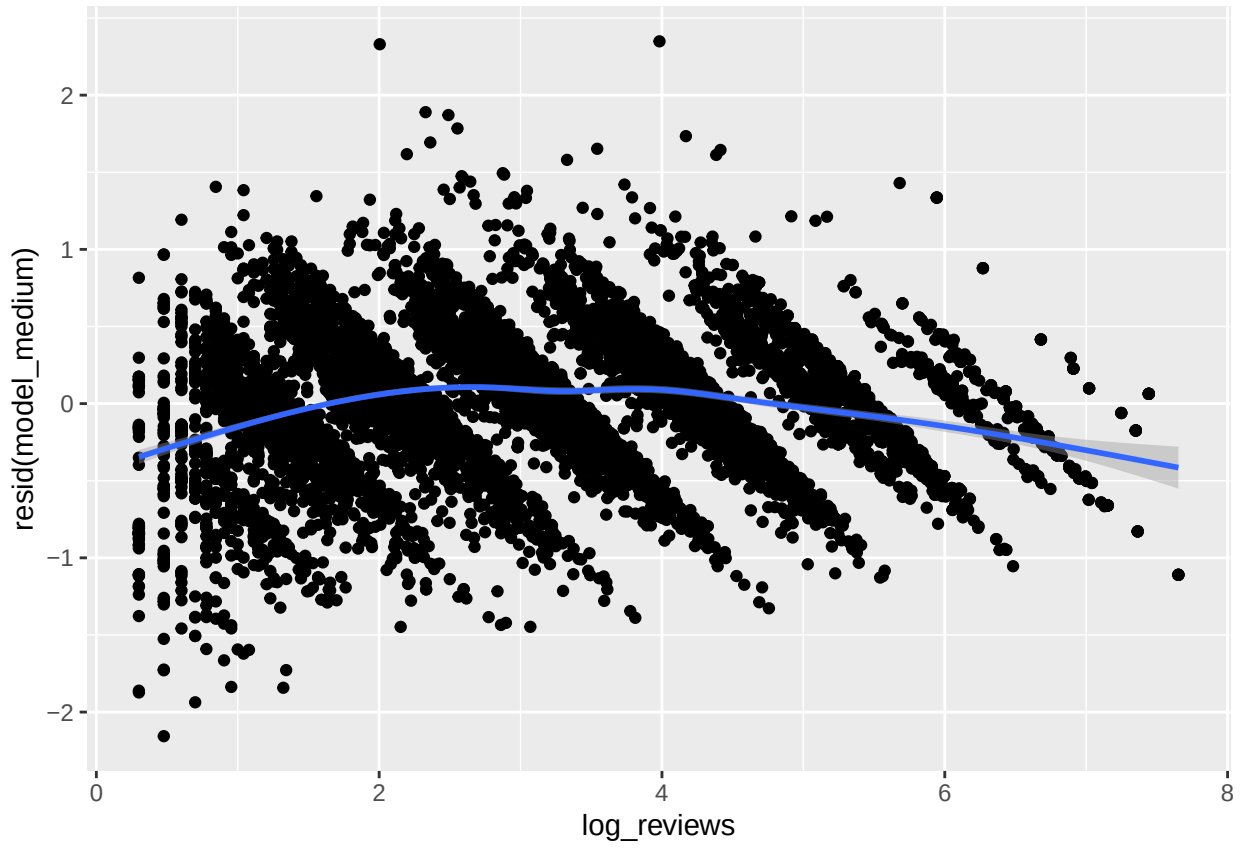
```
##                                                (0.010)                    (0.009)
##
## factor(Type)Paid                                                         -0.611***
##                                                                           (0.019)
##
## Constant                          1.860***                 3.213***      3.264***
##                                   (0.015)                  (0.083)       (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; ***
```

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.

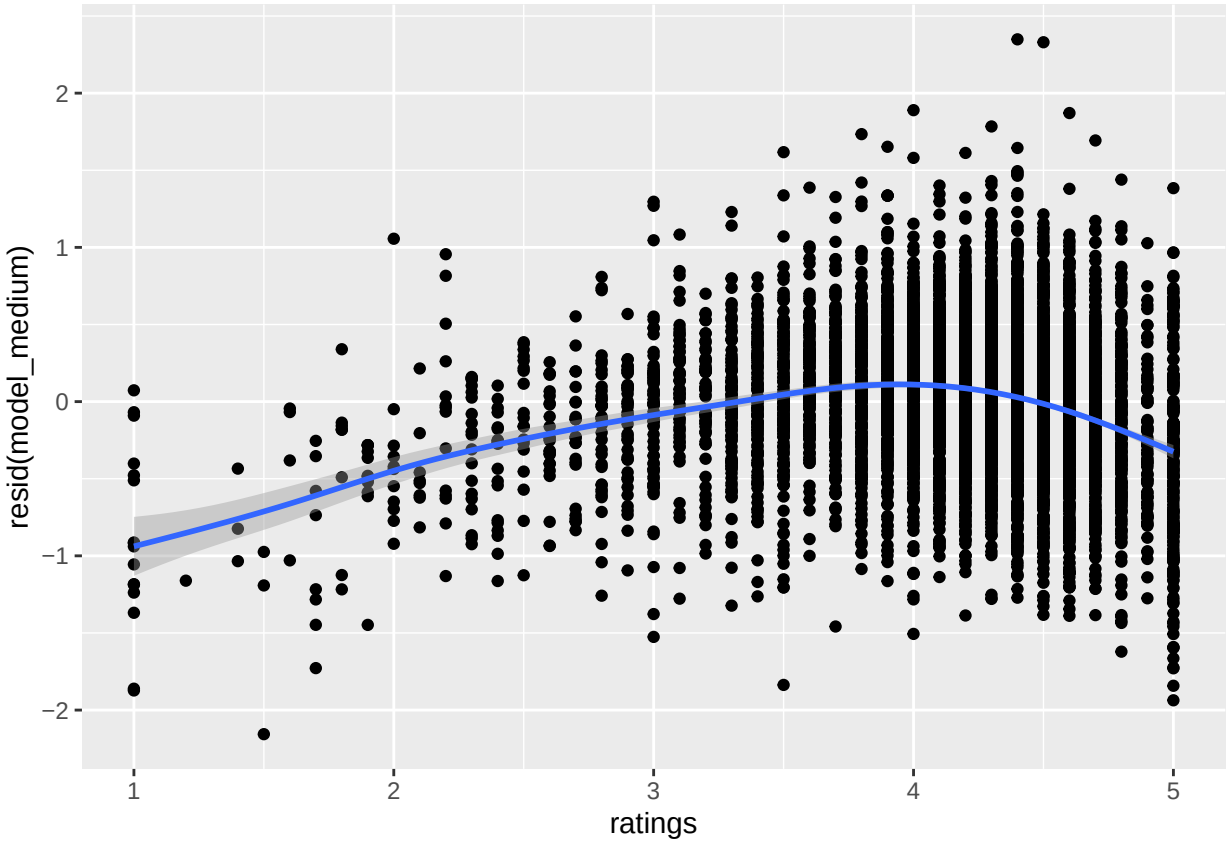
```
# Reviews versus residuals
plot_1 <- ggplot(data = d, mapping = aes(x = log_reviews, y = resid(model_medium))) +
  geom_point() + stat_smooth()
plot_1
```

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



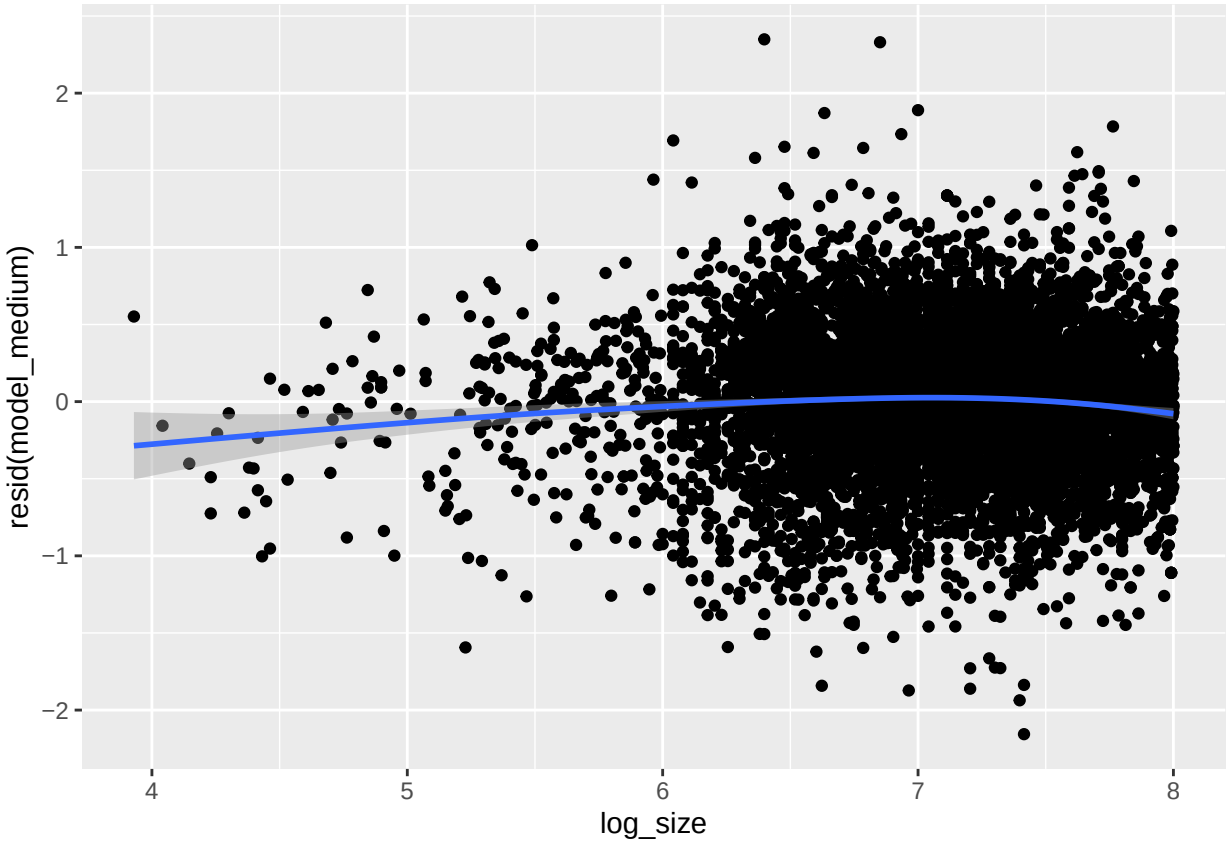

```
# Ratings versus residuals
plot_2 <- ggplot(data = d, mapping = aes(x = ratings, y = resid(model_medium))) +
  geom_point() + stat_smooth()
plot_2
```

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



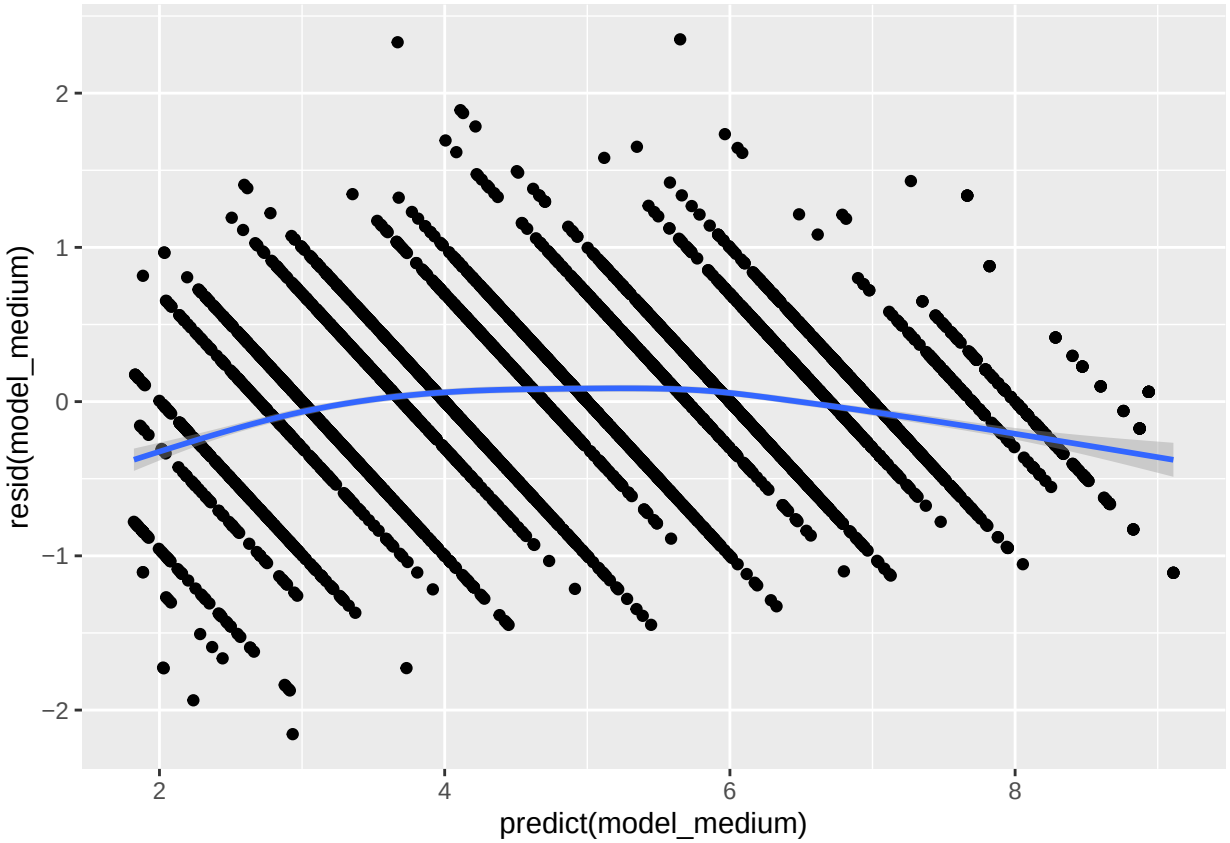

```
# Size versus residuals
plot_3 <- ggplot(data = d, mapping = aes(x = log_size, y = resid(model_medium))) +
  geom_point() + stat_smooth()
plot_3
```

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



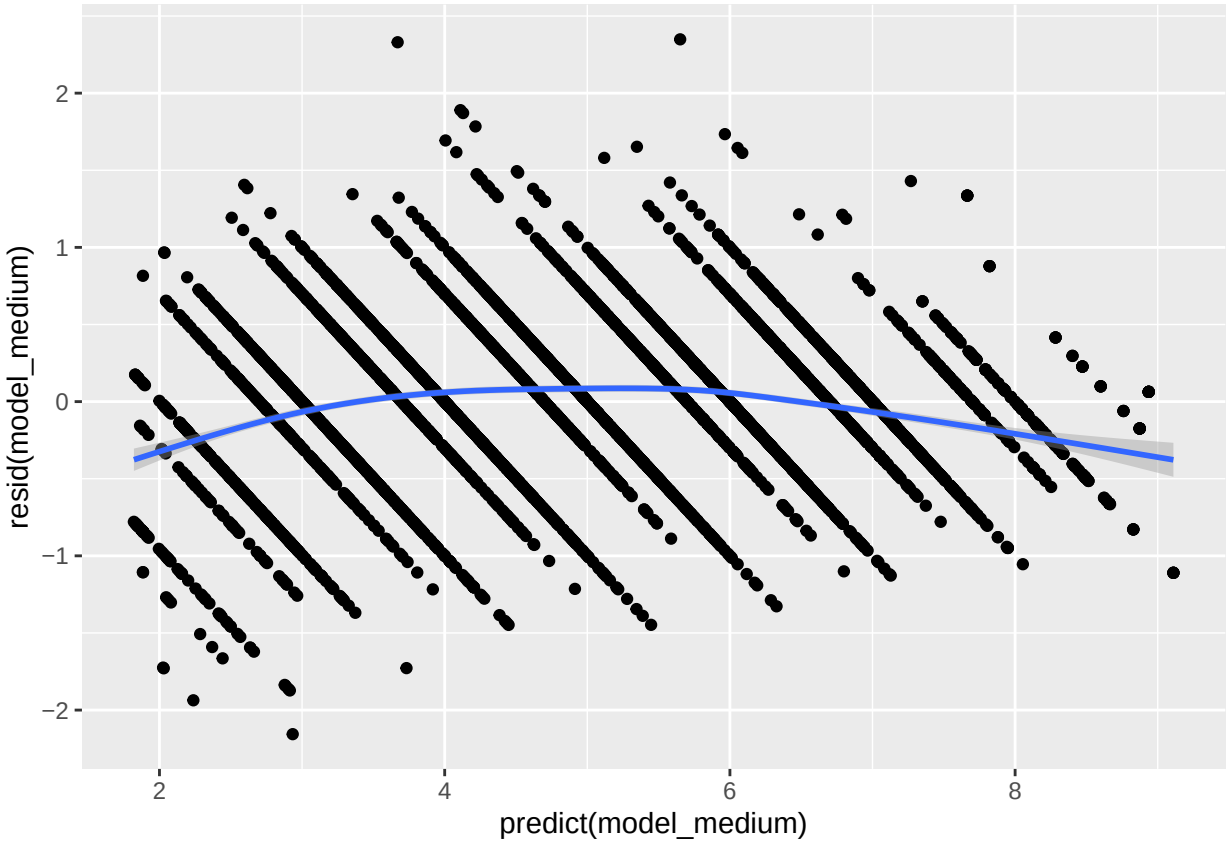
4. **Homoskedastic Errors:** When assessing homoskedastic errors, we seek to determine if there is a relationship between the variance of the model residuals and the predictors. If the homoskedastic assumption is satisfied, then we should observe a lack of relationship; conversely, if the data are heteroskedastic 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 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 = 235.66, df = 1, p-value < 2.2e-16
```

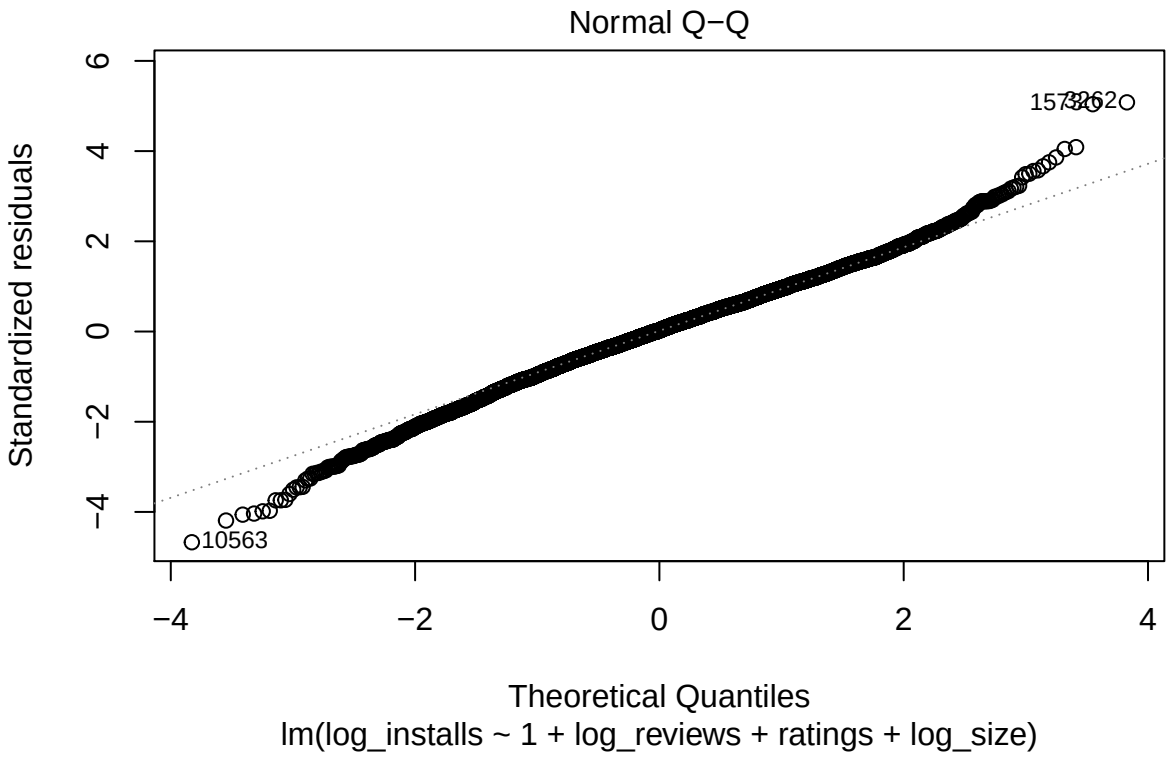
```
plot_4
```

```
## `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                        0.955***
##                                    (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
```