Simulation Based Regression Notes

Lt Col Ken Horton Professor Bradley Warner

31 July, 2020

Objectives

1) Using the bootstrap generate confidence and estimates of standard error for estimates from a linear regression model.

2) Generate and interpret bootstrap confidence and prediction intervals for predicted values.

3) Generate bootstrap samples from sampling rows of the data or sampling residuals. Explain why you might prefer one over the other.

4) Interpret regression coefficients for a linear model with a categorical explanatory variable.

Introduction

There are at least two ways we can consider creating a bootstrap distribution for a linear model. We can easily fit a linear model to a resampled data set. But in some situations this may have undesirable features. Influential observations, for example, will appear duplicated in some resamples and be missing entirely from other resamples. Another option is to use "residual resampling". In residual resampling, the new data set has all of the predictor values from the original data set and a new response is created by adding to the fitted function a resampled residual.

Suppose we have n observations, each with Y and some number of X's, with each observation stored as a row in a data set. The two basic procedures when bootstrapping regression are: a. bootstrap observations, and
b. bootstrap residuals.
The latter is a special case of a more general rule:
- sample Y from its estimated conditional distribution given X.

In bootstrapping observations, we sample with replacement from the rows of the data; each Y comes with the corresponding X's. In any bootstrap sample some observations may be repeated multiple times, and others not included.

In bootstrapping residuals, we fit the regression model, compute predicted values $\hat{Y}_i$ and residuals $e_i = Y_i - \hat{Y}_i$, then create a bootstrap sampling using the same X values as in the original data, but with new Y values obtained using the prediction plus a random residual, $Y_i^* = \hat{Y}_i + e_i^*$, where the residuals e_i^* are sampled randomly with replacement from the original residuals.

Bootstrapping residuals corresponds to a designed experiment, where the x values are fixed and only Y is random, and bootstrapping observations to randomly sampled data where both X and Y are sampled. By the principle of sampling the way the data were drawn, we would bootstrap observations if the X's were random. But we don't have to.

Confidence intervals for parameters

To build a confidence interval for the slope parameter, we will resample the data or residuals and generate a new regression model. This process does not assume normality of the residuals. We will use functions from the `mosaic` package to complete this work. However, know that `tidymodels` and `purrr` are more sophisticated tools for doing this work.

Resampling

Let's use the Starbucks data again.

```
library(openintro)
```

```
star_mod <- lm(calories~carb,data=starbucks)
```

```
star_mod
```

```
##
## Call:
## lm(formula = calories ~ carb, data = starbucks)
##
## Coefficients:
## (Intercept)         carb
##     146.020        4.297
```

Let's see how `do()` treats a linear model object.

```
obs<-do(1)*star_mod
obs
```

```
##   Intercept     carb    sigma r.squared        F numdf dendf .row .index
## 1  146.0204 4.297084 78.25956 0.4556237 62.77234     1    75    1      1
```

Nice. To resample the data we use `do()` with `resample()`.

```
do(2)*lm(calories~carb,data=resample(starbucks))
```

```
##   Intercept     carb    sigma r.squared        F numdf dendf .row .index
## 1  176.8364 3.710569 80.63468 0.3278707 36.58567     1    75    1      1
## 2  183.9955 3.526139 85.13087 0.3002583 32.18241     1    75    1      2
```

We are ready to go.

```
set.seed(532)
results <- do(1000)*lm(calories~carb,data=resample(starbucks))
```

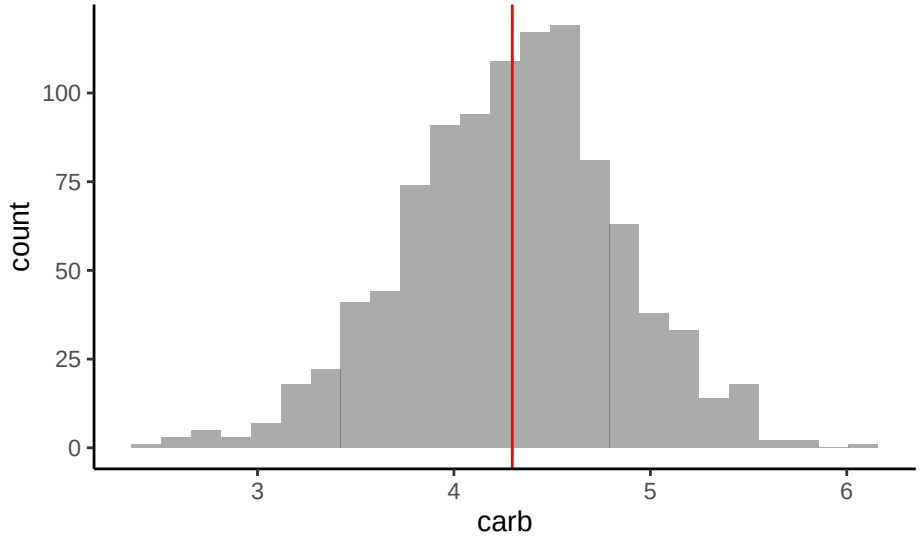
```
head(results)
```

```
##   Intercept      carb    sigma r.squared        F numdf dendf .row .index
## 1  154.7670 4.176327 78.94717 0.4127581 52.71568     1    75    1      1
## 2  166.8589 3.807697 72.09482 0.4032196 50.67437     1    75    1      2
## 3  105.3658 4.899956 77.62517 0.5310212 84.92195     1    75    1      3
## 4  227.4138 2.805156 79.97902 0.2467094 24.56317     1    75    1      4
## 5  194.9190 3.457191 83.74624 0.2670279 27.32313     1    75    1      5
## 6  183.1159 3.549460 73.90153 0.3931691 48.59292     1    75    1      6
```

With all this data, we can generate confidence intervals for the slope, R-squared, and F.

```
results %>%
  gf_histogram(~carb) %>%
  gf_vline(xintercept = obs$carb,color="red") %>%
  gf_theme(theme_classic())
```

```
## Warning: geom_vline(): Ignoring ‘mapping‘ because ‘xintercept‘ was provided.
```



The confidence interval is found using `cdata()`.

```
cdata(~carb,data=results)
```

```
##         lower    upper central.p
## 2.5% 3.166546 5.377743      0.95
```

We are 95% confident that the true slope is between 3.17 and 5.37. As a reminder, using the normality assumption we had a 95% confidence interval of $(3.21, 5.38)$.

```
confint(star_mod)
```

```
##                 2.5 %     97.5 %
## (Intercept) 94.387896 197.652967
## carb         3.216643   5.377526
```

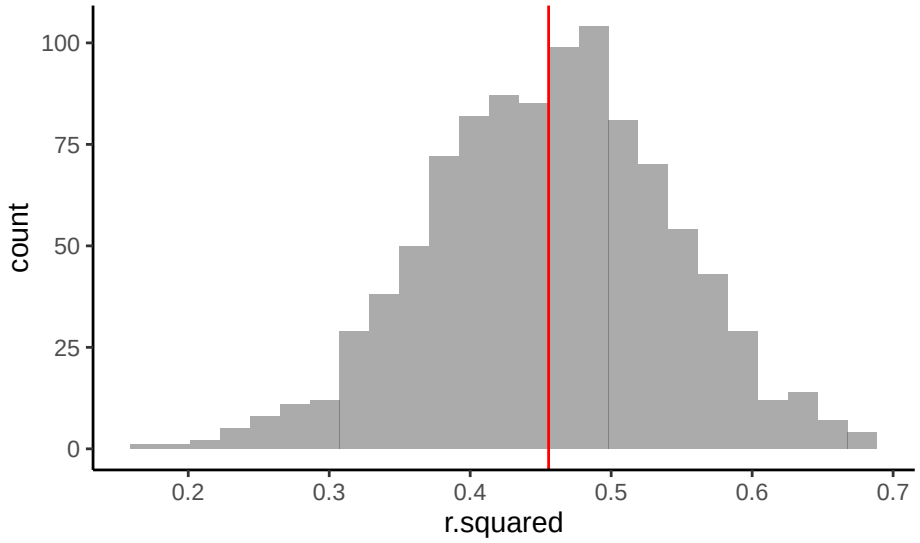
The confidence interval for R-squared is

```
cdata(~r.squared,data=results)
```

```
##          lower     upper central.p
## 2.5% 0.2837033 0.6234751      0.95
```

```
results %>%
  gf_histogram(~r.squared) %>%
  gf_vline(xintercept = obs$r.squared,color="red") %>%
  gf_theme(theme_classic())
```

```
## Warning: geom_vline(): Ignoring `mapping` because `xintercept` was provided.
```



This is nice.

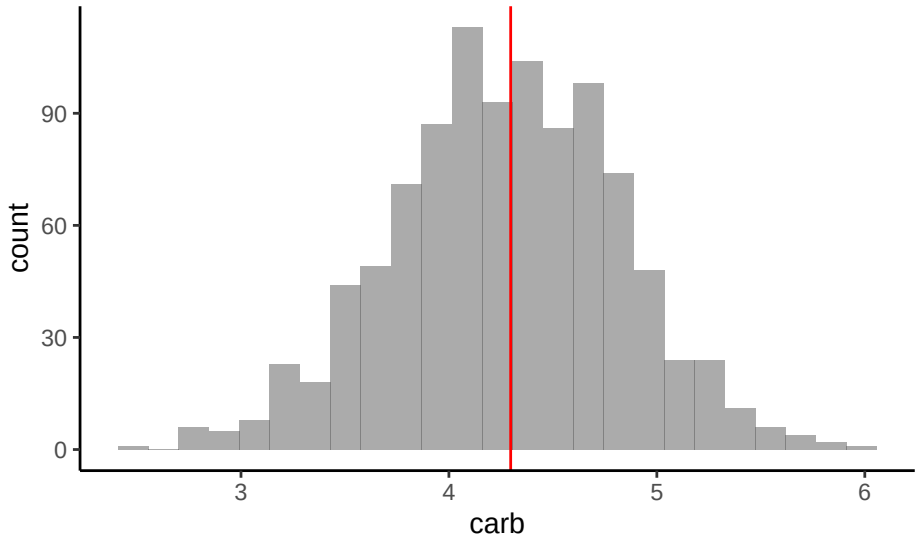
Resample residuals

We could also resample the residuals instead of the data. This makes a stronger assumption about the applicability of the linear model. However, it guarantees that every X value is in the resample dataframe.

```
results_resid <- do(1000) * lm( calories~carb, data = resample(star_mod)) # resampled residuals
```

```
## Warning: `select_()` is deprecated as of dplyr 0.7.0.
## Please use `select()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
results_resid %>%
  gf_histogram(~carb) %>%
  gf_vline(xintercept = obs$carb,color="red") %>%
  gf_theme(theme_classic())
```




```
cdata(~carb,data=results_resid)
```

```
##         lower    upper central.p
## 2.5% 3.192279 5.311577      0.95
```

Similar to the previous bootstrap confidence interval just a little narrower.

Confidence intervals for prediction

We now want to generate a confidence interval for the average calories from a 60 grams of carbohydrates.

Using the normal assumption, we had

```
predict(star_mod,newdata = data.frame(carb=60),interval="confidence")
```

```
##        fit      lwr      upr
## 1 403.8455 379.7027 427.9883
```

We have the slope and intercept in the `results` object. We can use `tidyverse` functions to find the confidence interval.

```
head(results)
```

```
##   Intercept     carb    sigma r.squared        F numdf dendf .row .index
## 1  154.7670 4.176327 78.94717 0.4127581 52.71568     1    75    1      1
## 2  166.8589 3.807697 72.09482 0.4032196 50.67437     1    75    1      2
## 3  105.3658 4.899956 77.62517 0.5310212 84.92195     1    75    1      3
## 4  227.4138 2.805156 79.97902 0.2467094 24.56317     1    75    1      4
## 5  194.9190 3.457191 83.74624 0.2670279 27.32313     1    75    1      5
## 6  183.1159 3.549460 73.90153 0.3931691 48.59292     1    75    1      6
```

```
results %>%
  mutate(pred=Intercept+carb*60) %>%
  cdata(~pred,data=.)
```

```
##         lower    upper central.p
## 2.5% 385.2706 423.6689      0.95
```

This is similar to the interval we found last lesson. We are 95% confident that the average calorie content for a menu item with 60 grams of carbohydrates is between 380.8 and 425.7.

Prediction interval

The prediction interval is more difficult. We have to account for the variability of the slope but also the residual since this is an individual observation. What we are going to do is sample with replacement from the residuals and then add this value to the predicted value in the last step.

First as a reminder, the prediction interval at 60 grams of `carb`.

```
predict(star_mod,newdata = data.frame(carb=60),interval="prediction")
```

```
##        fit      lwr      upr
## 1 403.8455 246.0862 561.6048
```

If we are generating a bootstrap of size 1000, we will resample from the residuals 1000 times.

```
results %>%
  mutate(pred=Intercept+carb*60) %>%
  cbind(resid=sample(star_mod$residuals,size=1000,replace = TRUE)) %>%
  mutate(pred_ind=pred+resid) %>%
  cdata(~pred_ind,data=.)
```

```
##         lower    upper central.p
## 2.5% 276.9905 577.8671      0.95
```

Close, just a little bit different.

Categorical predictor

We want to finish up simple linear regression by discussing a categorical predictor. It changes our interpretation somewhat.

Thus far, we have only discussed regression in the context of a quantitative, continuous response AND a quantitative, continuous predictor. We can build linear models with categorical predictor variables as well.

In the case of a binary covariate, nothing about the linear model changes. The two levels of the binary covariate are typically coded as 1 and 0, and the model is built, evaluated and interpreted in an analogous fashion as before.

In the case of a categorical covariate with k levels, where $k > 2$, we need to include $k - 1$ *dummy variables* in the model. Each of these dummy variables takes the value 0 or 1. For example, if a covariate has $k = 3$ categories or levels (say A, B or C), we create two dummy variables, X_1 and X_2, each of which can only

take values 1 or 0. If $X_1 = 1$, the covariate takes the value A. If $x_2 = 1$, the covariate takes the value B. If both $X_1 = 0$ and $X_2 = 0$, this is known as the reference category, and in this case the covariate takes the value C. The arrangement of the levels of the categorical covariate are arbitrary and can be adjusted by the user. This coding is called **contrasts** and again is typically taught in a course on linear models.

The linear model is $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + e$.

When the covariate takes the value A, $\mathrm{E}(Y) = \beta_0 + \beta_1$.

When the covariate takes the value B, $\mathrm{E}(Y) = \beta_0 + \beta_2$.

When the covariate takes the value C, $\mathrm{E}(Y) = \beta_0$.

Based on this, think about how you would interpret the coefficients β_0, β_1, and β_2.

Lending Club

This data set represents thousands of loans made through the Lending Club platform, which is a platform that allows individuals to lend to other individuals. Of course, not all loans are created equal. Someone who is a essentially a sure bet to pay back a loan will have an easier time getting a loan with a low interest rate than someone who appears to be riskier. And for people who are very risky? They may not even get a loan offer, or they may not have accepted the loan offer due to a high interest rate. It is important to keep that last part in mind, since this data set only represents loans actually made, i.e. do not mistake this data for loan applications! The data set is `loans_full_schema` from the `openintro` package.

```
library(openintro)
```

```
dim(loans_full_schema)
```

```
## [1] 10000    55
```

This is a big data set. For educational purposes, we will sample 100 points from the original data. We need to drop the extra factor levels in `homeownership` that have zero observations.

```
tally(~homeownership,data=loans_full_schema,format="proportion")
```

```
## homeownership
##                ANY MORTGAGE      OWN     RENT
##    0.0000   0.0000   0.4789   0.1353   0.3858
```

We to sample the data so that each level of home ownership has the same proportion as the original, a stratified sample.

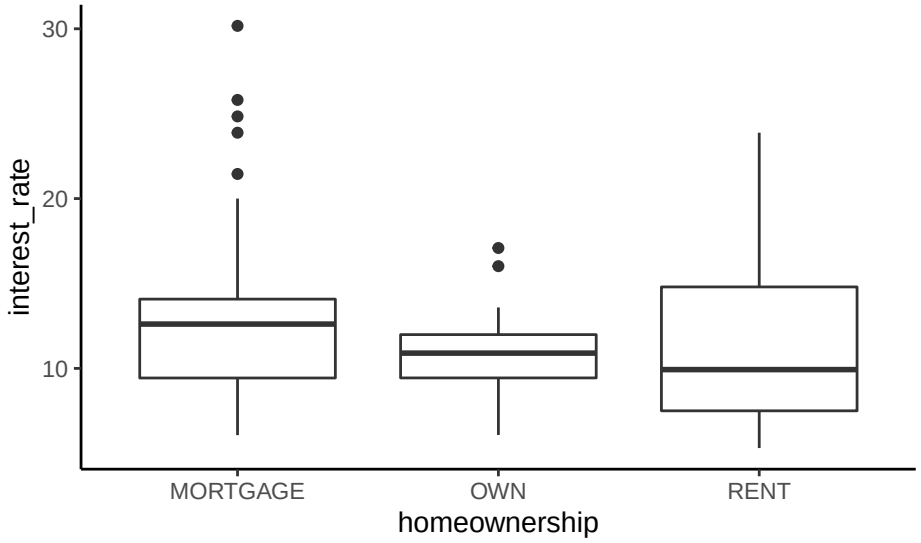
```
loans100 <- loans_full_schema %>%
  select(interest_rate,homeownership) %>%
  droplevels() %>%
  group_by(homeownership) %>%
  slice_sample(prop=0.01) %>%
  ungroup()
```

```
tally(~homeownership,data=loans100,format="proportion")
```

```
## homeownership
##  MORTGAGE       OWN      RENT
## 0.4795918 0.1326531 0.3877551
```

Let's look at the data.

```
loans100 %>%
  gf_boxplot(interest_rate~homeownership) %>%
  gf_theme(theme_classic())
```



It appears that there is some evidence that home ownership impacts the interest rate. We can build a linear model to explore whether this difference in significant. We can use the `lm()` function in `R`, but in order to include a categorical predictor, we need to make sure that variable is stored as a "factor" type. If it is not, we'll need to convert it.

```
str(loans100)
```

```
## tibble [98 x 2] (S3: tbl_df/tbl/data.frame)
##  $ interest_rate: num [1:98] 9.93 15.05 10.9 14.08 12.62 ...
##  $ homeownership: Factor w/ 3 levels "MORTGAGE","OWN",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Now we can build the model:

```
loan_mod<-lm(interest_rate ~ homeownership,data=loans100)
summary(loan_mod)
```

```
##
## Call:
## lm(formula = interest_rate ~ homeownership, data = loans100)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -6.869 -3.545 -0.755  1.861 17.231
```

```
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)        12.9391     0.7471  17.320   <2e-16 ***
## homeownershipOWN   -1.7691     1.6049  -1.102    0.273
## homeownershipRENT  -1.4244     1.1173  -1.275    0.205
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.122 on 95 degrees of freedom
## Multiple R-squared:  0.0224, Adjusted R-squared:  0.001816
## F-statistic: 1.088 on 2 and 95 DF,  p-value: 0.341
```

Note that by default, `R` set the `MORTGAGE` level as the reference category. This is because it is first alphabetically. You can control this by changing the order of the factor levels. The package `forcats` helps with this effort.

How would we interpret this output? Since `MORTGAGE` is reference category, the intercept is effectively the estimated, average, interest rate for home owners with a mortgage.

```
loans100 %>%
  filter(homeownership == "MORTGAGE") %>%
  summarise(average=mean(interest_rate))
```

```
## # A tibble: 1 x 1
##   average
##     <dbl>
## 1    12.9
```

The other terms represent the expected difference in delivery times for the other locations.

```
loans100 %>%
  group_by(homeownership) %>%
  summarise(average=mean(interest_rate),std_dev=sd(interest_rate))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 3 x 3
##   homeownership average std_dev
##   <fct>           <dbl>   <dbl>
## 1 MORTGAGE         12.9    5.52
## 2 OWN              11.2    3.18
## 3 RENT             11.5    5.12
```

Specifically, on average, interest rates for home owners who own the house is 3.193 percent less than those with a mortgage those who rent is 0.8564 percent higher on average.

> **Exercise**:
> Using the coefficient from the regression model, how do we find the difference in average interest rates between home owners and renters?

The first coefficient $\beta_{homeownershipOWN} = \mu_{OWN} - \mu_{MORTGAGE}$ and $\beta_{homeownershipRENT} = \mu_{RENT} - \mu_{MORTGAGE}$. Thus $\mu_{OWN} - \mu_{RENT} = \beta_{homeownershipOWN} - \beta_{homeownershipRENT}$, the difference in coefficients.

The model is not fitting a line to the data but just estimating average with the coefficients representing difference from the reference level.

The `Std.Error`, `t value`, and `Pr(>|t|)` values can be used to conduct inference about the respective estimates. It appears that a significant difference in mean interest rates between owners and those with a mortgage exists.

Bootstrap

From the boxplots, the biggest difference in means is between home owners and renters. However, in the regression output there is not p-value to test this difference. An easy solution would be to change the reference level but what if you had many levels? How would you know which ones to test? In the next section we will look at multiple comparisons but before then we can use the bootstrap to help us.

Let's bootstrap the regression.

```
set.seed(532)
results <- do(1000)*lm(interest_rate ~ homeownership,data=resample(loans100))
```

```
head(results)
```

```
##   Intercept homeownershipOWN homeownershipRENT    sigma   r.squared          F
## 1  12.32660      -1.67660000        -1.2723500 5.079923 0.018042801 0.87278044
## 2  13.25458       0.02541667        -1.1535833 4.538518 0.015965588 0.77066963
## 3  11.51904       0.25971154        -0.1024595 3.949203 0.000610356 0.02900962
## 4  12.76354      -1.53154167        -3.0700417 4.786713 0.086382191 4.49110559
## 5  12.17980      -1.02434137        -0.6771643 4.819335 0.006751215 0.32286241
## 6  13.34440      -0.60973333        -1.1931879 5.167354 0.011124003 0.53433408
##   numdf dendf .row .index
## 1     2    95    1      1
## 2     2    95    1      2
## 3     2    95    1      3
## 4     2    95    1      4
## 5     2    95    1      5
## 6     2    95    1      6
```

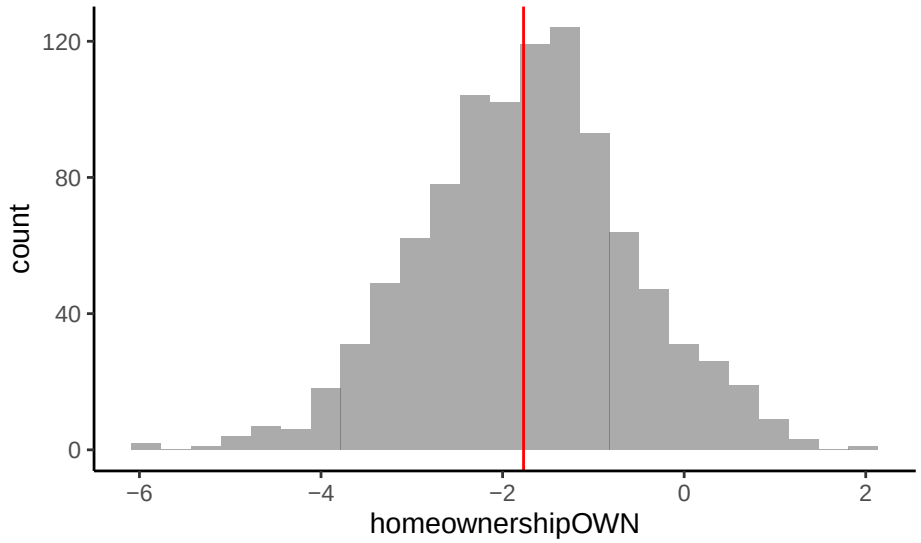
We of course can generate a confidence interval on either of the coefficients in the `results` object.

```
obs<-do(1)*loan_mod
obs
```

```
##   Intercept homeownershipOWN homeownershipRENT    sigma  r.squared       F
## 1  12.93915        -1.769149         -1.424412 5.121521 0.02239738 1.08825
##   numdf dendf .row .index
## 1     2    95    1      1
```

```
results %>%
  gf_histogram(~homeownershipOWN) %>%
  gf_vline(xintercept = obs$homeownershipOWN,color="red") %>%
  gf_theme(theme_classic())
```

```
## Warning: geom_vline(): Ignoring `mapping` because `xintercept` was provided.
```




```
cdata(~homeownershipOWN,data=results)
```

```
##          lower     upper central.p
## 2.5% -3.981076 0.6392392      0.95
```

Which is similar to the results assuming normality.

```
confint(loan_mod)
```

```
##                        2.5 %     97.5 %
## (Intercept)       11.456066 14.4222320
## homeownershipOWN  -4.955322  1.4170246
## homeownershipRENT -3.642522  0.7936982
```

However, we want a confidence interval for the difference between home owners and renters.

```
results %>%
  mutate(own_rent=homeownershipOWN - homeownershipRENT) %>%
  cdata(~own_rent,data=.)
```

```
##          lower    upper central.p
## 2.5% -2.734185 2.033604      0.95
```

Done! From this interval we can infer that home owners have a significantly lower interest rate than renters.

ANOVA Table

As a reminder, we could also report the results of loans analysis using an *analysis of variance*, or ANOVA, table.

```r
anova(loan_mod)
```

```
## Analysis of Variance Table
##
## Response: interest_rate
##               Df  Sum Sq Mean Sq F value Pr(>F)
## homeownership  2   57.09  28.545  1.0882  0.341
## Residuals     95 2491.85  26.230
```

This table lays out how the variation between observations is broken down. This is a simultaneous test of equal of the three means. Using the F-statistic, we would reject the null hypothesis of no differences in mean response across levels of the categorical variable. Notice it is the same p-value reported for the F distribution in the regression summary.

Pairwise Comparisons

The ANOVA table above (along with the summary of the linear model output before that) merely tells you whether any difference exists in the mean response across the levels of the categorical predictor. It does not tell you where that difference lies. In the case of using regression we can compare `MORTGAGE` to the other two levels but can't conduct a hypothesis of `OWN` vs `RENT`. In order to make all pairwise comparisons, we need another tool. A common one is the Tukey method. Essentially, the Tukey method conducts three hypothesis tests (each under the null of no difference in mean) but corrects the p-values based on the understanding that we are conducting three simultaneous hypothesis tests with the same set of data and we don't want to inflate the Type 1 error.

We can obtain these pairwise comparisons using the `TukeyHSD()` function in `R`. The "HSD" stands for "Honest Significant Differences". This function requires an `anova` object, which is obtained by using the `aov()` function:

```r
TukeyHSD(aov(interest_rate~homeownership, data=loans100))
```

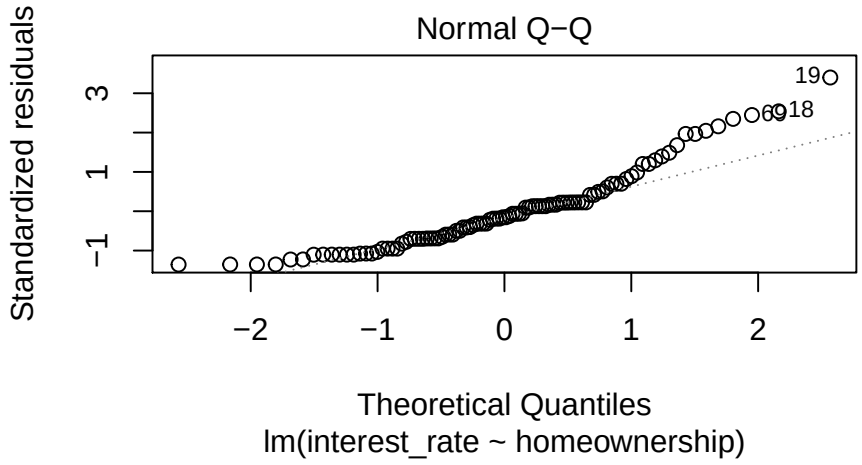
```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = interest_rate ~ homeownership, data = loans100)
##
## $homeownership
##                       diff       lwr      upr     p adj
## OWN-MORTGAGE  -1.7691489 -5.590469 2.052171 0.5150087
## RENT-MORTGAGE -1.4244121 -4.084691 1.235867 0.4128101
## RENT-OWN       0.3447368 -3.573404 4.262877 0.9761025
```

According to this output, only the average interest rate for owners is different from renters.

Assumptions

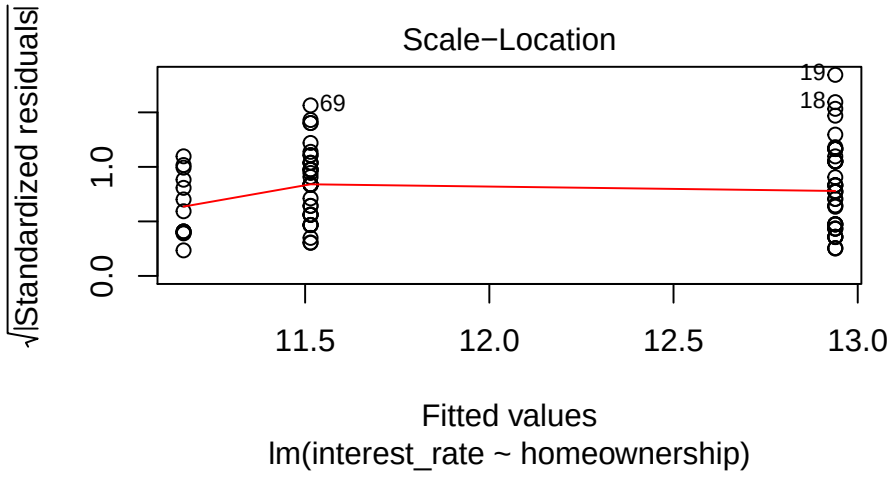
Keep in mind that ANOVA is a special case of a simple linear model. Therefore, all of the assumptions remain the same except for the linearity. The order of the levels is irrelevant and thus a line does not need to go through the three levels. In order to evaluate these assumptions, we would need to obtain the appropriate diagnostic plots:

```
plot(loan_mod,2)
```



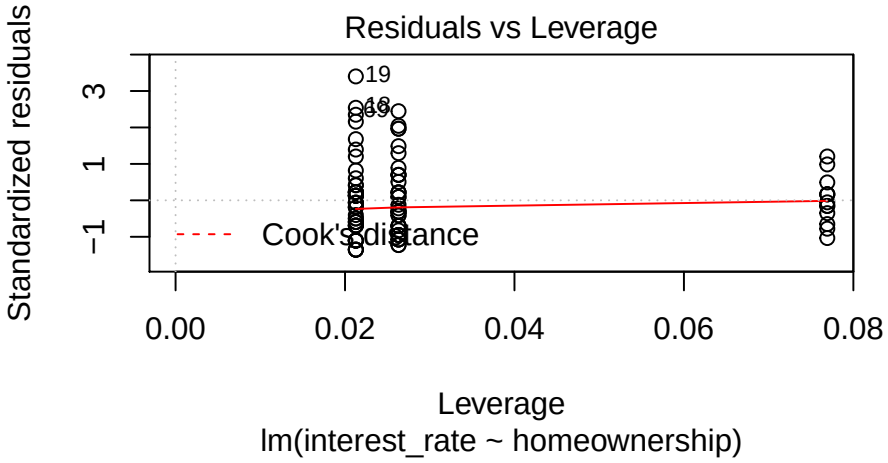
Normality is suspect but we have a large sample size and thus we did not get much of a difference in results from the bootstrap which does not assume normality.

```
plot(loan_mod,3)
```



The assumption of equal variance is also suspect. The variance for the home owners might be less than that for the other two.

```
plot(loan_mod,5)
```



We have three points that might be outliers but they are not too extreme. In general, nothing in this plot is concerning to us.

File Creation Information

- File creation date: 2020-07-31
- Windows version: Windows 10 x64 (build 18362)
- R version 3.6.3 (2020-02-29)
- `mosaic` package version: 1.7.0
- `tidyverse` package version: 1.3.0