Regression Analysis

Fang Yu

2024-07-11

Contents

Learning Objective

1. Review Stats 413 lab content
2. Familiar with important functions and packages
3. Practical utilization of regression analysis

Basic R Markdown Operations

You can make text **bold** by surrounding it with two asterisks (**) and *italic* by surrounding it with one asterisk (*) - as seen throughout this document.

Demo #1: Hit *Cmd + Option + I* to insert a new R code chunk. Name this chunk `demo1` (no spaces!) and add code to print your name.

```
print("Fang Yu")
```

```
## [1] "Fang Yu"
```

Demo #2: Assignment in R by using `<-` and print the assigned variables

```
x <- 4 * 7 + 90 - 100
x
```

```
## [1] 18
```

Demo #3: Read in documents which is in csv format, using the function `read.csv()`, use `head()` to view the first 6 rows of the data

```
penguins <- read.csv("penguins.csv")
head(penguins)
```

```
##   species    island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
## 1  Adelie Torgersen           39.1          18.7               181        3750
## 2  Adelie Torgersen           39.5          17.4               186        3800
## 3  Adelie Torgersen           40.3          18.0               195        3250
## 4  Adelie Torgersen           36.7          19.3               193        3450
## 5  Adelie Torgersen           39.3          20.6               190        3650
## 6  Adelie Torgersen           38.9          17.8               181        3625
##      sex year
## 1   male 2007
## 2 female 2007
```

```
## 3 female 2007
## 4 female 2007
## 5   male 2007
## 6 female 2007
```

Frequency table

let's start to summarize the data. One way to do this for *categorical* variables is by creating a "frequency table". This counts the number of observations (rows) that correspond to each category of a specific variable. To make a frequency table, we use the `table()` function(*need to specify the data name penguins$*):

```
table(penguins$species)
```

```
##
##    Adelie Chinstrap    Gentoo
##       146        68       119
```

We can also make "two-way" frequency tables (also called **contingency tables**) to summarize counts for two categorical variables:

```
table(penguins$species, penguins$island)
```

```
##
##             Biscoe Dream Torgersen
##   Adelie        44    55        47
##   Chinstrap      0    68         0
##   Gentoo       119     0         0
```

Numerical Summaries

Using the `summary()` function, R returns 6 numbers: the minimum (shortest) flipper length, the first quartile, the median (middle) flipper length, the mean (average) flipper length, the third quartile, and the maximum (longest) flipper length:

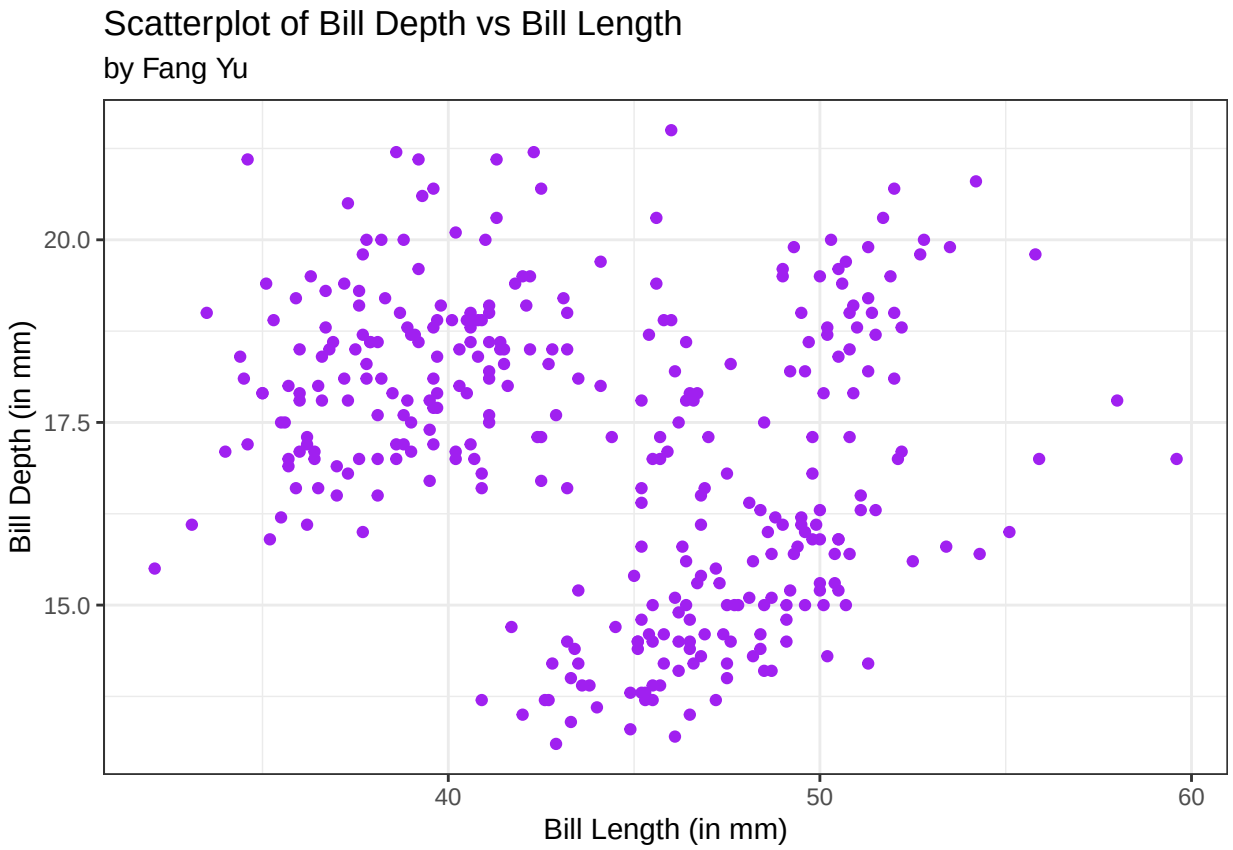
```
summary(penguins$flipper_length_mm)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     172     190     197     201     213     231
```

ggplot

Create a regression scatterplot using `ggplot()` function, by using the format of `ggplot(data = , aes(x = ,y = ) + geom_point(color = ) + labs(title = ,subtitle = ,x = ,y = ) + theme_bw())`, see the following chunks for detailed illustration of plotting bill_depth(y) against bill_length(x)

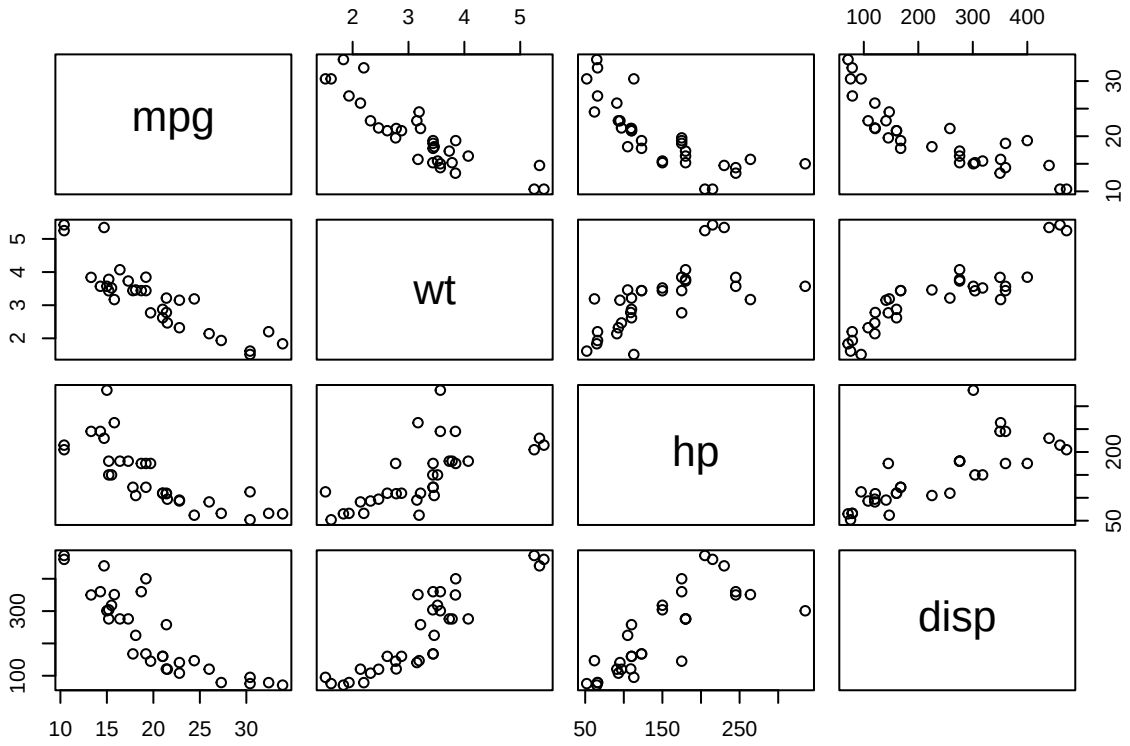
```
library(ggplot2)
ggplot(data = penguins, aes(x = bill_length_mm, y = bill_depth_mm)) +

  geom_point(color = "purple") +

  labs(title = "Scatterplot of Bill Depth vs Bill Length",
       subtitle = "by Fang Yu",
       x = "Bill Length (in mm)",
       y = "Bill Depth (in mm)") +

  theme_bw()
```



Scatterplot Matrix

Using the `mtcars` data set, create a scatterplot matrix that includes fuel efficiency (mpg), vehicle weight, horsepower, and engine displacement.

```
data(mtcars)
plot(~mpg + wt + hp + disp, data = mtcars)
```



Estimated Model

To estimate the coefficients of a linear regression model, we use the `lm()` function. The `data` argument will reference the data set that we want to use. The `formula` argument will take the following structure:

response ~ predictor_1 + predictor_2 + ... + predictor_p

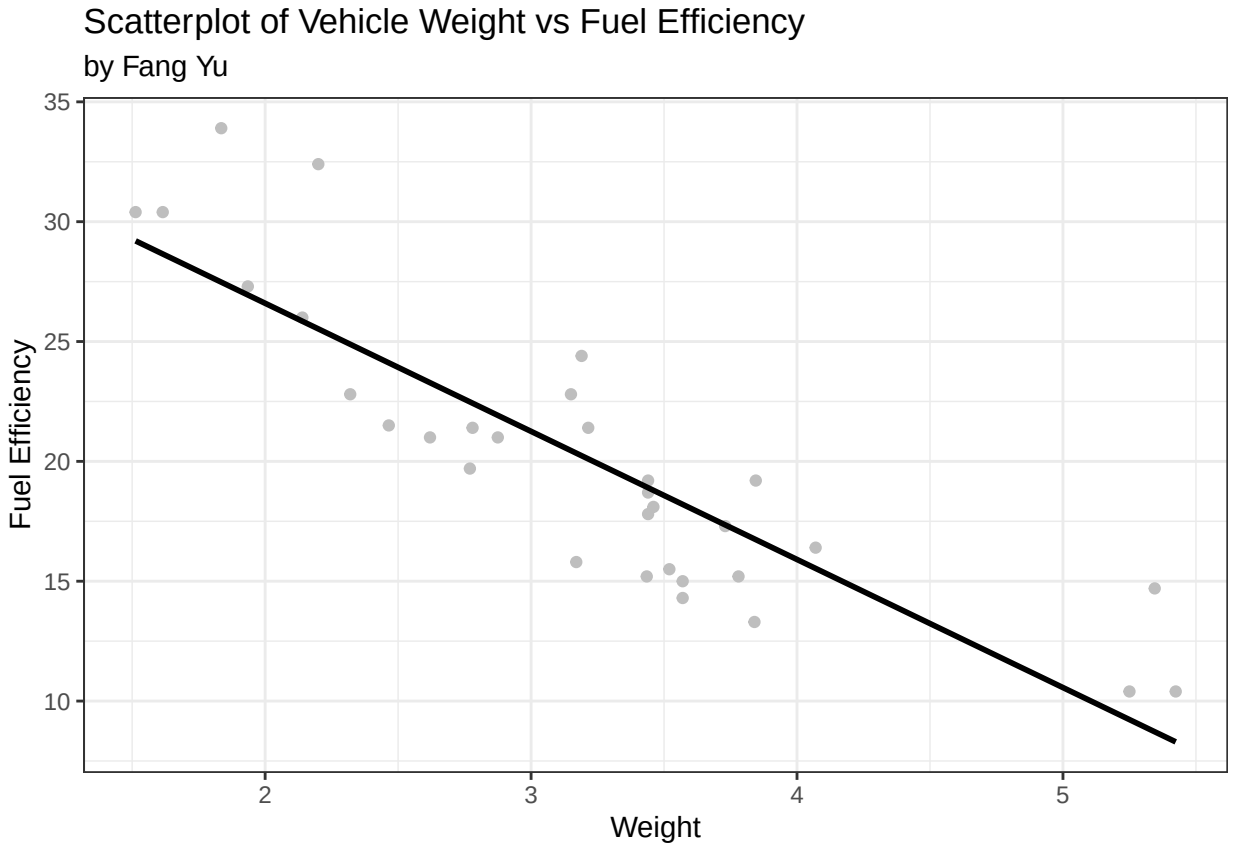
Create a linear regression model with mpg against wt using the data `mtcars`

```
lm(mpg ~ wt, data = mtcars)
```

```
##
## Call:
## lm(formula = mpg ~ wt, data = mtcars)
##
## Coefficients:
## (Intercept)           wt
##      37.285       -5.344
```

To visualize the **estimated relationship**, we will add `geom_smooth()` to our scatterplot code:

```
ggplot(data = mtcars, aes(x = wt, y = mpg)) +

  geom_point(color = "grey") +

  geom_smooth(method = "lm", formula = y ~ x, se = FALSE, color = "black") +

  labs(title = "Scatterplot of Vehicle Weight vs Fuel Efficiency",
       subtitle = "by Fang Yu",
       x = "Weight",
       y = "Fuel Efficiency") +

  theme_bw()
```



Testing the overall model

To use our linear model in a variety of additional ways, we can store the linear model as an object in our global environment. We simple use a left facing arrow (<-) and give the model a name.With this stored model, we can retrieve additional summary information, the design matrix, create diagnostic plots, run additional tests, etc.

First, let's pass our stored model through the `summary()` function.

```
lm_penguins <- lm(body_mass_g ~ flipper_length_mm + bill_length_mm + bill_depth_mm, data = penguins)
summary(lm_penguins)
```

```
##
## Call:
## lm(formula = body_mass_g ~ flipper_length_mm + bill_length_mm +
##     bill_depth_mm, data = penguins)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1051.37  -284.50   -20.37   241.03  1283.51
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -6445.476    566.130 -11.385   <2e-16 ***
## flipper_length_mm    50.762      2.497  20.327   <2e-16 ***
## bill_length_mm        3.293      5.366   0.614    0.540
## bill_depth_mm        17.836     13.826   1.290    0.198
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 393 on 329 degrees of freedom
## Multiple R-squared:  0.7639, Adjusted R-squared:  0.7618
## F-statistic: 354.9 on 3 and 329 DF,  p-value: < 2.2e-16
```

Then, we can pass the stored model through the `anova()` function.

```
anova(lm_penguins)
```

```
## Analysis of Variance Table
##
## Response: body_mass_g
##                     Df    Sum Sq   Mean Sq   F value Pr(>F)
## flipper_length_mm    1 164047703 164047703 1062.1232 <2e-16 ***
## bill_length_mm       1    140000    140000    0.9064 0.3418
## bill_depth_mm        1    257051    257051    1.6643 0.1979
## Residuals          329  50814912    154453
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

If we want to run a test for a linear combination of predictors, we need the estimated variance-covariance matrix for the coefficients. To compute this matrix, pass the stored model through the `vcov()` function.

```
vcov(lm_penguins)
```

```
##                    (Intercept) flipper_length_mm bill_length_mm bill_depth_mm
## (Intercept)       320503.2363      -1211.206658     805.525086   -6528.69599
## flipper_length_mm  -1211.2067          6.236320      -8.786474      20.06738
## bill_length_mm       805.5251         -8.786474      28.793240     -17.85211
## bill_depth_mm      -6528.6960         20.067377     -17.852112     191.15678
```

Prediction Interval

To create a prediction interval, we use the `predict` function, but one of its arguments requires us to input a data frame with the value of each predictor variable. When creating this data frame, **we must type in the variable names exactly as they appear in the original data set**. Once we have *correctly* created the data frame, we pass it through the `predict.lm()` function along with some other necessary arguments.

The `predict.lm()` function has four important arguments:

- The first argument is our stored regression model (`lm_penguins`)
- `newdata`: the data frame for our new observation
- `interval`: the type of interval ("prediction" or "confidence")
- `level`: the desired confidence level

```
new_penguin <- data.frame(flipper_length_mm = 220,
                          bill_length_mm = 45,
                          bill_depth_mm = 15.4)
predict.lm(lm_penguins,
           newdata = new_penguin,
           interval = "prediction",
           level = 0.90)
```

```
##        fit     lwr      upr
## 1 5145.052 4493.18 5796.925
```

For an individual penguin with a flipper length of 220 mm, a bill length of 45 mm, and a bill depth of 15.4 mm, we would predict their body mass to be between (4493, 5797).

Confidence Interval

We want an interval for the *average* response of *all* observations with the same set of given values.

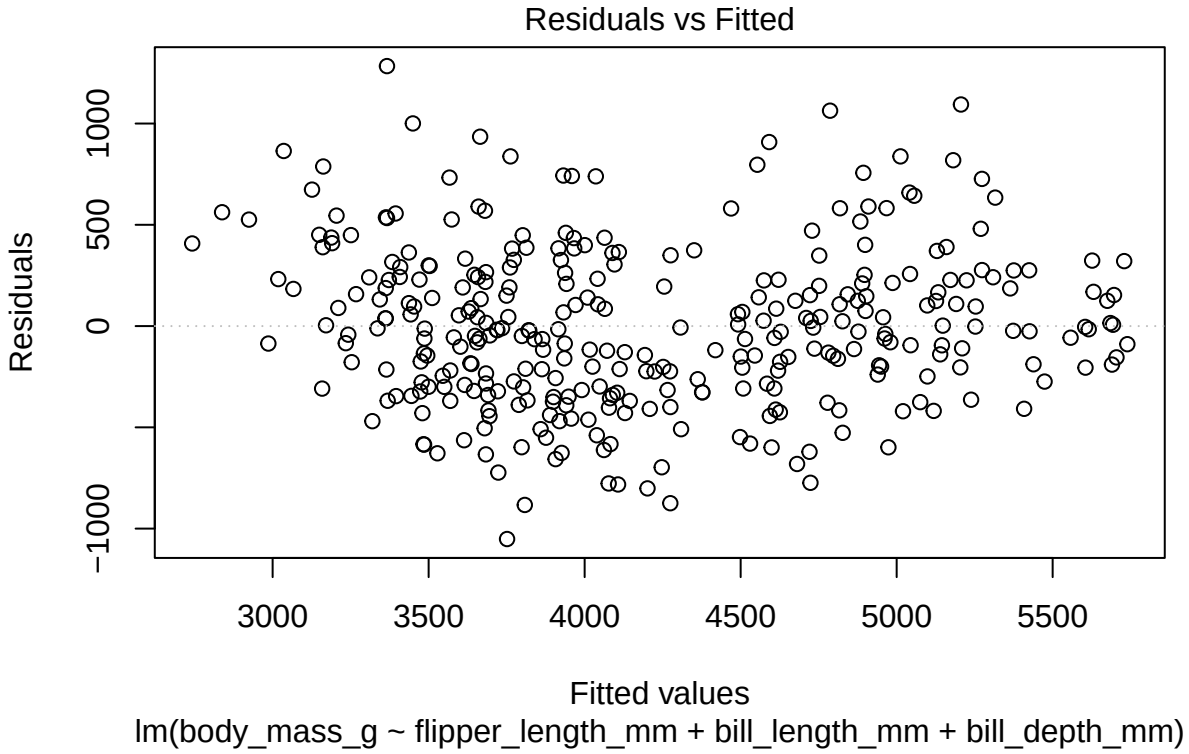
```
predict.lm(lm_penguins,
           newdata = new_penguin,
           interval = "confidence",
           level = 0.90)
```

```
##        fit      lwr      upr
## 1 5145.052 5076.523 5213.581
```

Diagnostic Plots

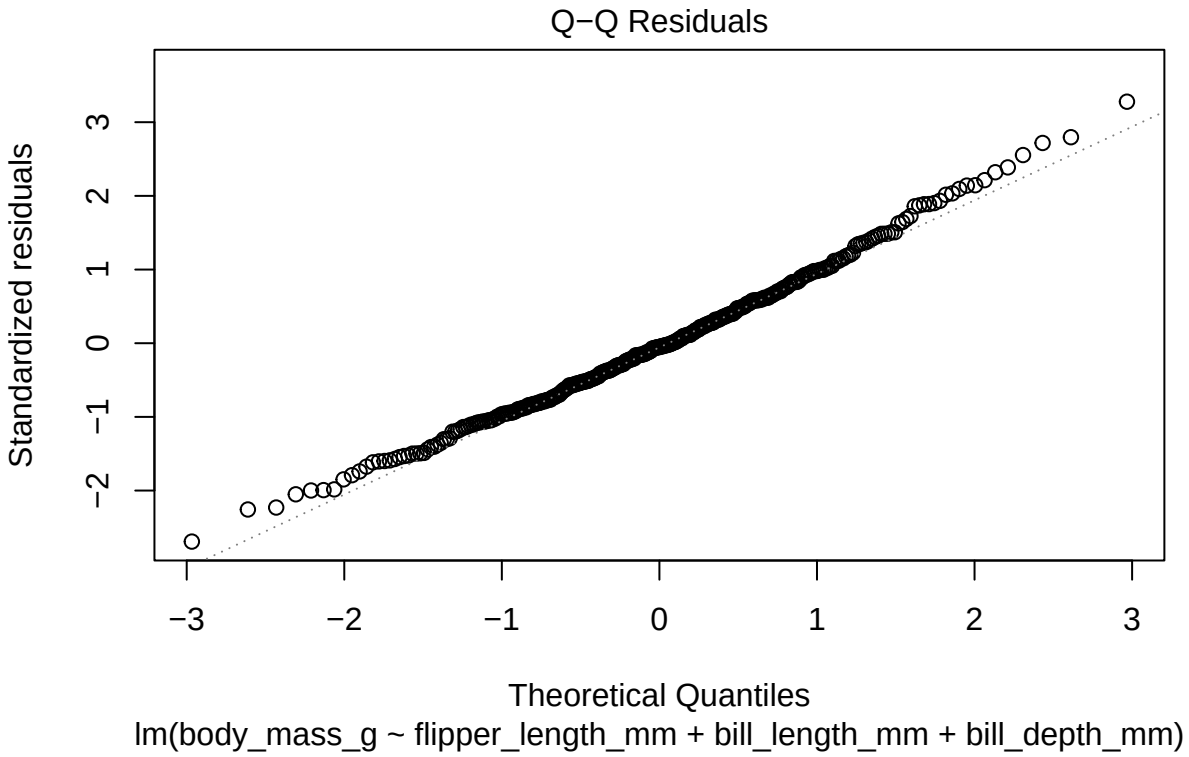
To create a residual plot, using the following code:

```
plot(lm_penguins, which = 1, id.n = 0, add.smooth = FALSE)
```



To create a QQ-plot, using the following code:

```
plot(lm_penguins, which = 2, id.n = 0, add.smooth = FALSE)
```



Factor Variables

We can use the `as.factor()` function to convert `species`, `island`, and `sex` to factor variables by overwriting the existing variables.

```
penguins$species <- as.factor(penguins$species)
penguins$island <- as.factor(penguins$island)
penguins$sex <- as.factor(penguins$sex)
```

If you are interested in the specific levels (or groups) of a factor variable, use the `levels()` function. The first level listed is the *reference category*:

```
levels(penguins$species)
```

```
## [1] "Adelie"    "Chinstrap" "Gentoo"
```

```
levels(penguins$island)
```

```
## [1] "Biscoe"    "Dream"     "Torgersen"
```

```
levels(penguins$sex)
```

```
## [1] "female" "male"
```

The default ordering is alphabetical. If we wish to change the first level of a factor variable (to give the factor variable a new reference category), we can use the `relevel()` function. To make "Chinstrap" the reference category for the `species` variable, we use the following code:

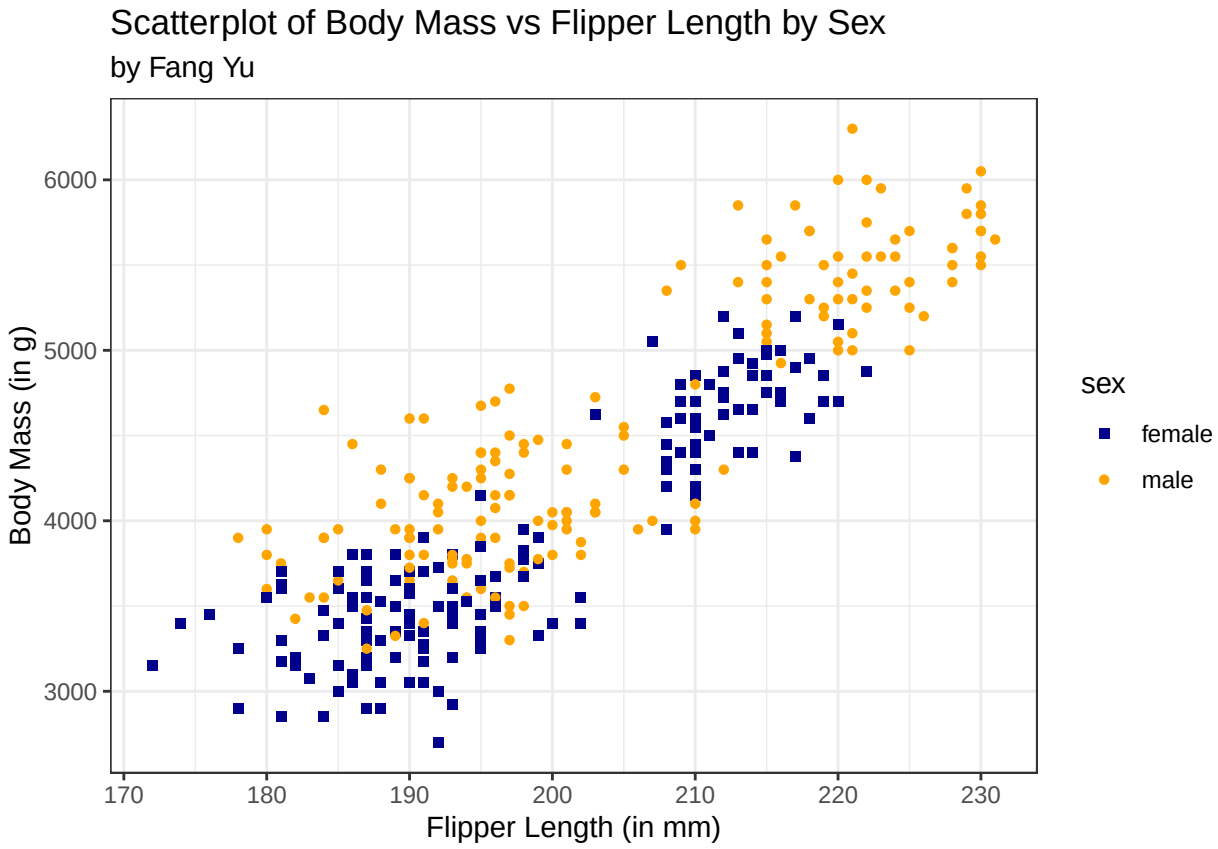
```
penguins$species <- relevel(penguins$species, "Chinstrap")
levels(penguins$species)
```

```
## [1] "Chinstrap" "Adelie"    "Gentoo"
```

Plotting by group

Create a scatterplot of body mass versus flipper length *by sex*. Use different colors and shapes to denote male and female penguins:

```
ggplot(data = penguins, aes(x = flipper_length_mm,
                            y = body_mass_g,
                            color = sex,
                            shape = sex)) +

  geom_point() +

  labs(title = "Scatterplot of Body Mass vs Flipper Length by Sex",
       subtitle = "by Fang Yu",
       x = "Flipper Length (in mm)",
       y = "Body Mass (in g)") +

  theme_bw() +
  scale_color_manual(values = c("darkblue", "orange", "grey40")) +
  scale_shape_manual(values = c(15, 16, 17))
```



Interactions

To include an interaction term, we use the `*` operator instead of the `+` operator when creating our regression model.

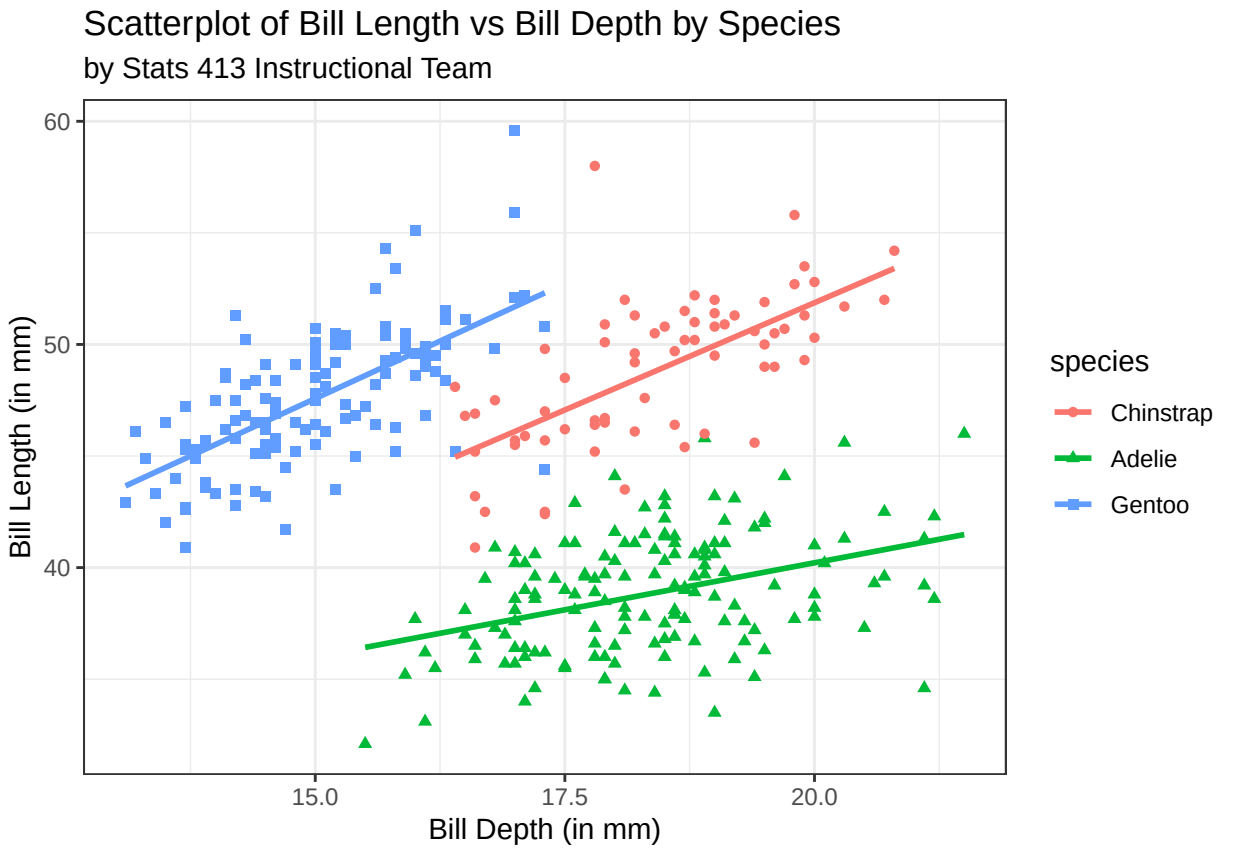
```
lm_interaction <- lm(bill_length_mm ~ species * bill_depth_mm, data = penguins)
```

```r
summary(lm_interaction)
```

```
##
## Call:
## lm(formula = bill_length_mm ~ species * bill_depth_mm, data = penguins)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.9137 -1.5150  0.0587  1.5733 10.3590
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   13.4279     4.8548   2.766 0.005999 **
## speciesAdelie                  9.9389     5.7395   1.732 0.084277 .
## speciesGentoo                  3.2423     5.9444   0.545 0.585829
## bill_depth_mm                  1.9221     0.2631   7.307 2.11e-12 ***
## speciesAdelie:bill_depth_mm   -1.0796     0.3113  -3.468 0.000595 ***
## speciesGentoo:bill_depth_mm    0.1382     0.3483   0.397 0.691692
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.445 on 327 degrees of freedom
## Multiple R-squared:  0.8032, Adjusted R-squared:  0.8001
## F-statistic: 266.8 on 5 and 327 DF,  p-value: < 2.2e-16
```

We can visualize the estimated model of each species using the following code.

```r
ggplot(data = penguins, aes(x = bill_depth_mm,
                            y = bill_length_mm,
                            color = species,
                            shape = species)) +

  geom_point() +

  geom_smooth(method = "lm", formula = y ~ x, se = FALSE) +

  labs(title = "Scatterplot of Bill Length vs Bill Depth by Species",
       subtitle = "by Stats 413 Instructional Team",
       x = "Bill Depth (in mm)",
       y = "Bill Length (in mm)") +

  theme_bw()
```



Quadratic Fit

When the relationship between the response and a predictor is non-linear, we can attempt a quadratic fit to improve the model by including the `I()` function

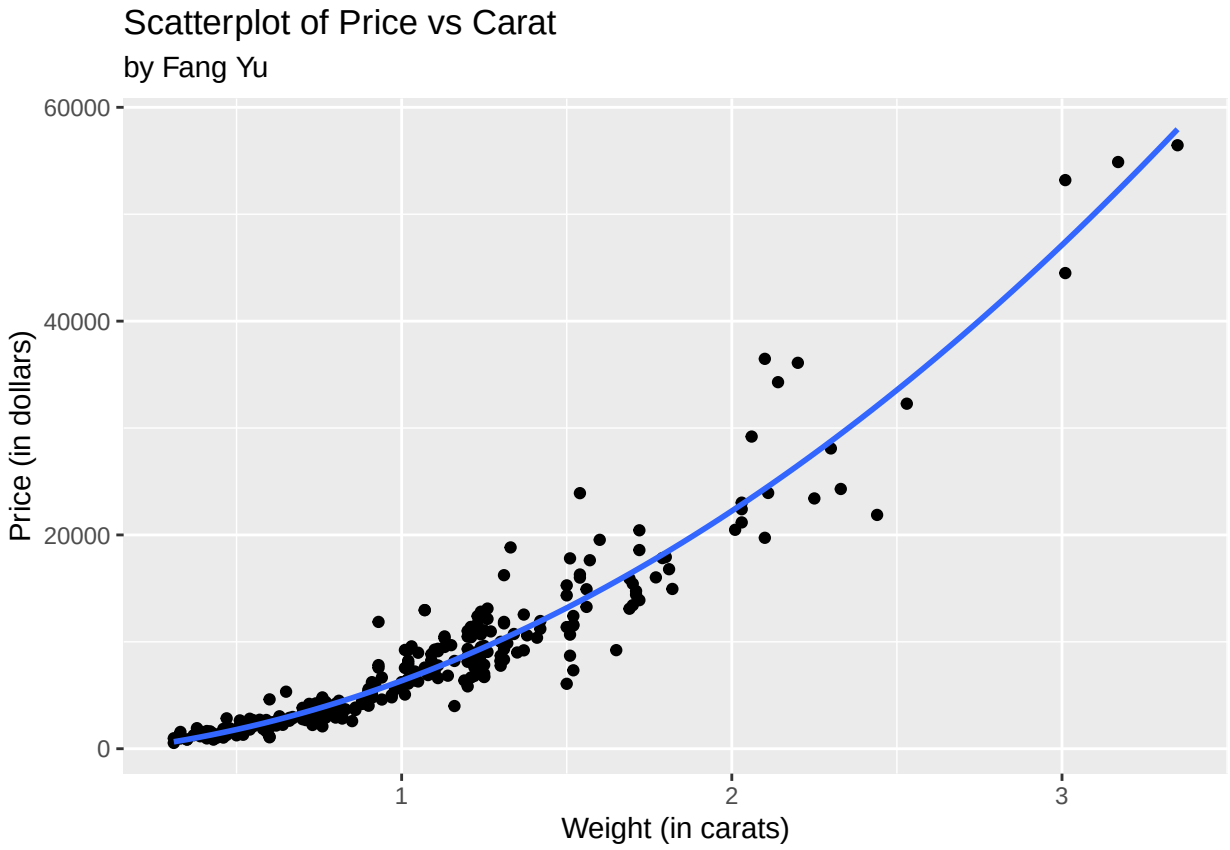
```
diamonds <- read.csv("diamonds.csv")
lm_diamonds <- lm(Price ~ Carat + I(Carat^2), data = diamonds)
summary(lm_diamonds)
```

```
##
## Call:
## lm(formula = Price ~ Carat + I(Carat^2), data = diamonds)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10207.4   -711.6   -167.9    355.0  12147.3
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -522.7      466.3  -1.121  0.26307
## Carat         2386.0      752.5   3.171  0.00166 **
## I(Carat^2)    4498.2      263.0  17.101  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2127 on 348 degrees of freedom
## Multiple R-squared:  0.9257, Adjusted R-squared:  0.9253
```

```
## F-statistic:  2168 on 2 and 348 DF,  p-value: < 2.2e-16
```

To plot the estimated regression model with a quadratic fit, we use a different `formula` in the `geom_smooth()` portion of our code.

```
ggplot(data = diamonds, aes(x = Carat, y = Price)) +

  geom_point() +

  geom_smooth(method = "lm", formula = y ~ x + I(x^2), se = FALSE) +

  labs(title = "Scatterplot of Price vs Carat",
       subtitle = "by Fang Yu",
       x = "Weight (in carats)",
       y = "Price (in dollars)")
```



Log Transformation

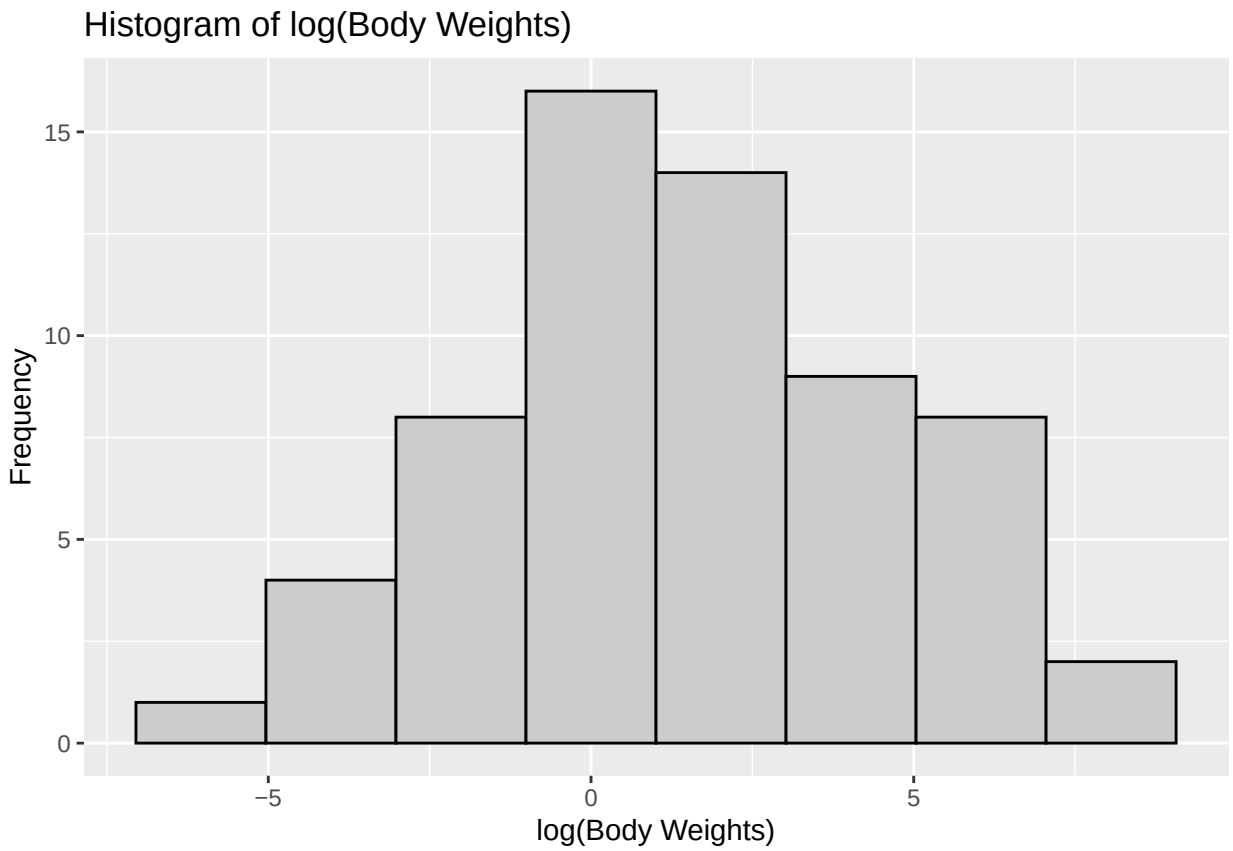
Use log transformation to fix right-skewed scenario, by including `log()` function and create the histogram for the log transformed variable `body`:

```
weights <- read.csv("mammals.csv", row.names = 1)
lm_weight <- lm(log(brain) ~ log(body), data = weights)
summary(lm_weight)
```

```
##
## Call:
## lm(formula = log(brain) ~ log(body), data = weights)
##
```

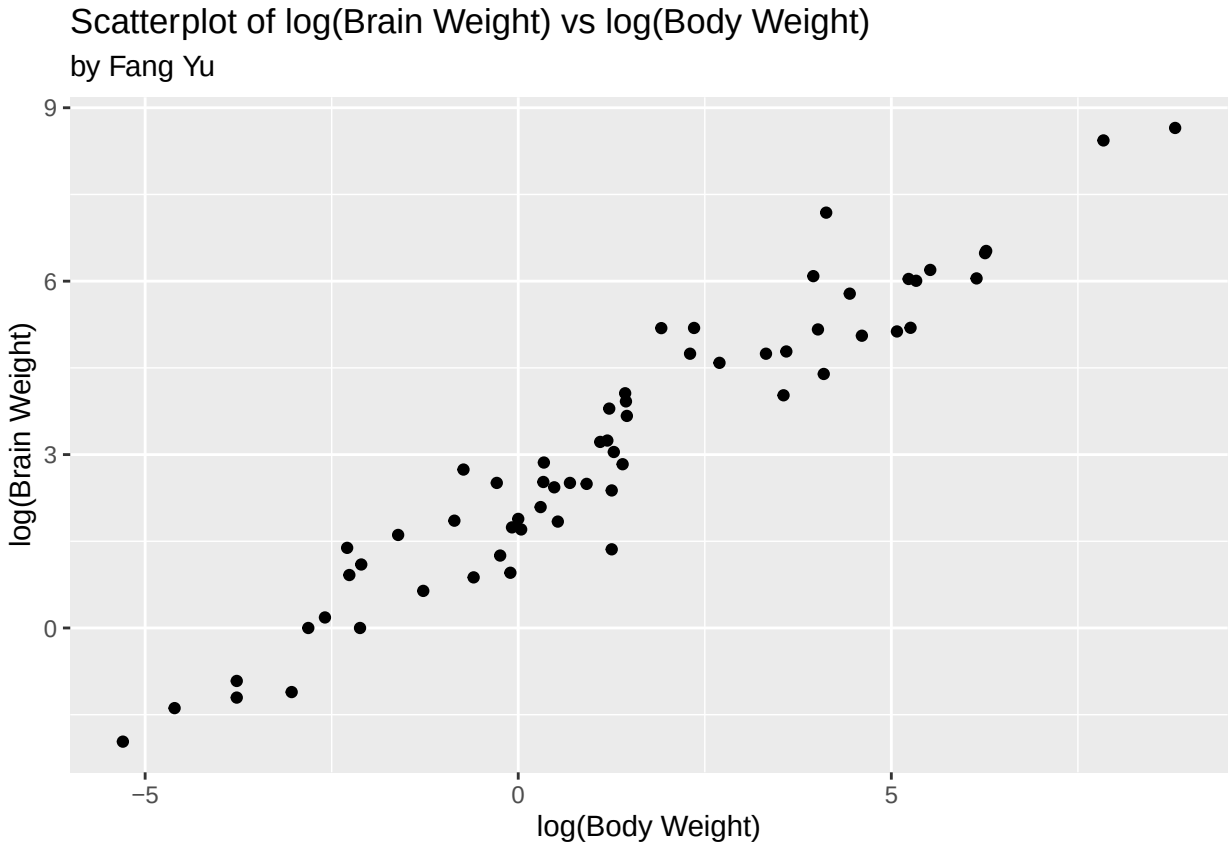
```
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.71550 -0.49228 -0.06162  0.43598  1.94833
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.13479    0.09604   22.23   <2e-16 ***
## log(body)    0.75169    0.02846   26.41   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6943 on 60 degrees of freedom
## Multiple R-squared:  0.9208, Adjusted R-squared:  0.9195
## F-statistic: 697.4 on 1 and 60 DF,  p-value: < 2.2e-16
```

```
ggplot(data = weights, aes(x = log(body))) +

  geom_histogram(bins = 8, color = "black", fill = "grey80") +

  labs(title = "Histogram of log(Body Weights)",
       x = "log(Body Weights)",
       y = "Frequency")
```




```
ggplot(data = weights, aes(x = log(body), y = log(brain))) +

  geom_point() +

  labs(title = "Scatterplot of log(Brain Weight) vs log(Body Weight)",
```

```
       subtitle = "by Fang Yu",
       x = "log(Body Weight)",
       y = "log(Brain Weight)")
```



Multicollinearity

Multicollinearity occurs when one predictor has a strong relationship with another predictor or a linear combination of other predictors. We can check for multicollinearity by using the `vif()` function from the `car` package.

```
if (!requireNamespace("car", quietly = TRUE)) {
    install.packages("car")
}
library(car)
```

```
## Loading required package: carData
```

```
sat <- read.csv("sat.csv", row.names = 1)
lm_salary <- lm(salary ~ expend + ratio + takers + verbal + math, data = sat)
vif(lm_salary)
```

```
##    expend     ratio    takers    verbal      math
##  2.050826  1.273655  7.506217 22.690255 19.470915
```

Let's see if there are any strong relationships between any pair of predictors by computing the pairwise correlations for the predictors.

```
cor(sat[, c(1,2,4,5,6)])
```

```
##            expend       ratio     takers      verbal        math
## expend  1.0000000 -0.37102539  0.5926274 -0.41004987 -0.34941409
## ratio  -0.3710254  1.00000000 -0.2130536  0.06376664  0.09542173
## takers  0.5926274 -0.21305361  1.0000000 -0.89326296 -0.86938393
## verbal -0.4100499  0.06376664 -0.8932630  1.00000000  0.97025604
## math   -0.3494141  0.09542173 -0.8693839  0.97025604  1.00000000
```

Likelihood Ratio Test & AIT

To run a Likelihood Ratio Test, we use the `anova()` function and specify `LRT` as the `test`. The model with fewer predictors should go first.

```
lm_salary1 <- lm(salary ~ expend + ratio + total, data = sat)
anova(lm_salary1, lm_salary, test = "LRT")
```

```
## Analysis of Variance Table
##
## Model 1: salary ~ expend + ratio + total
## Model 2: salary ~ expend + ratio + takers + verbal + math
##   Res.Df    RSS Df Sum of Sq Pr(>Chi)
## 1     46 198.44
## 2     44 185.33  2    13.108    0.211
```

Use `AIC()` function for model comparison, always choose models with lower AIC values

```
AIC(lm_salary1, lm_salary)
```

```
##            df      AIC
## lm_salary1  5 220.8165
## lm_salary   7 221.3996
```

Outliers

We can access the leverages by using `hatvalues()`:

```
hatvalues(lm_salary1)
```

```
##         AL         AK         AZ         AR         CA         CO         CT
## 0.05026032 0.16076106 0.05538142 0.04516584 0.24831690 0.02985429 0.11518635
##         DE         FL         GA         HI         ID         IL         IN
## 0.04322945 0.06593190 0.10154284 0.04780364 0.05981488 0.05551379 0.05207054
##         IA         KS         KY         LA         ME         MD         MA
## 0.09292880 0.06845269 0.02696936 0.04078500 0.07573210 0.04710176 0.05228809
##         MI         MN         MS         MO         MT         NE         NV
## 0.14542230 0.08681648 0.06176208 0.05372360 0.02856856 0.07219163 0.05101186
##         NH         NJ         NM         NY         NC         ND         OH
## 0.03284911 0.18709292 0.04194007 0.17385705 0.09645613 0.11148983 0.02163564
##         OK         OR         PA         RI         SC         SD         TN
## 0.05605154 0.07540646 0.05505580 0.06329372 0.13294596 0.10164451 0.05666836
##         TX         UT         VT         VA         WA         WV         WI
## 0.07195861 0.26985704 0.06956678 0.08835052 0.07340723 0.04266591 0.10192885
##         WY
## 0.04129040
```

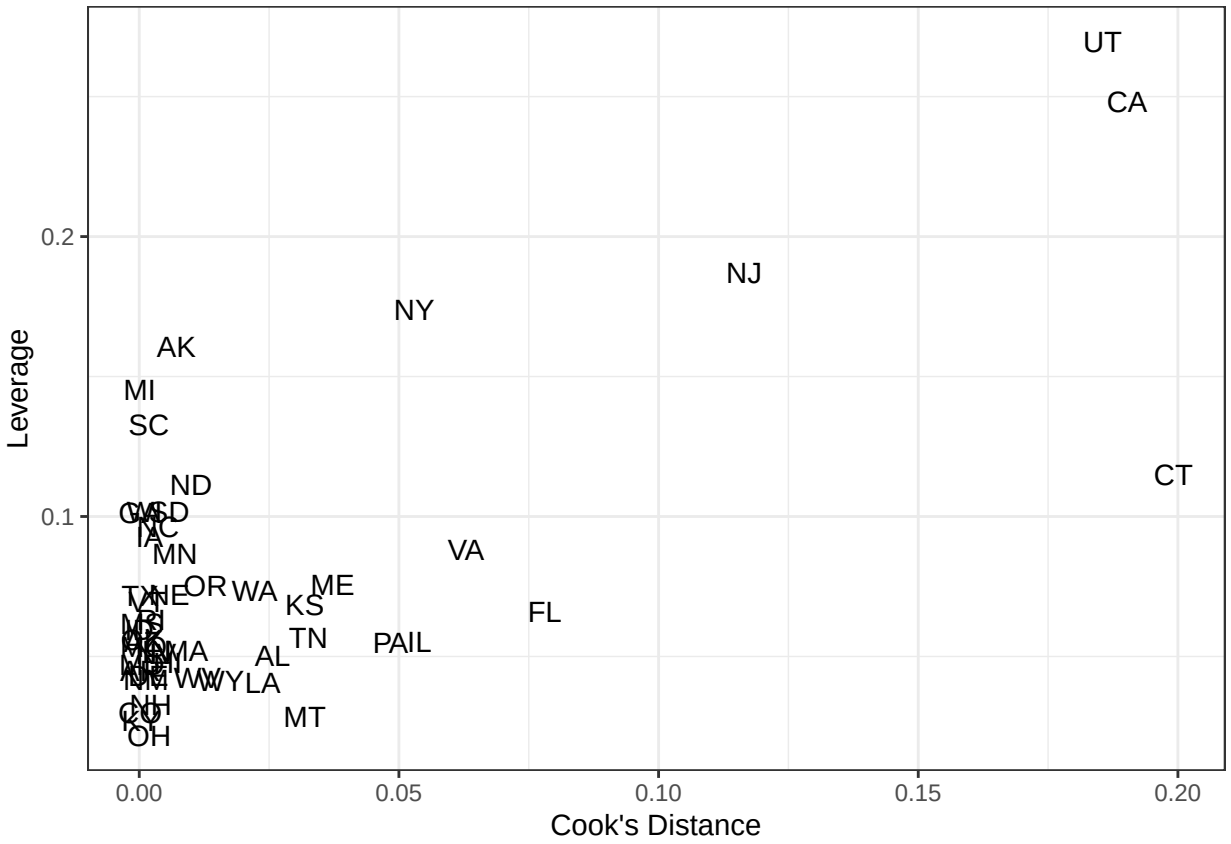
We can access the Cook's distances by using `cooks.distance()`:

```
cooks.distance(lm_salary1)
```

```
##           AL           AK           AZ           AR           CA           CO
## 2.568089e-02 7.159750e-03 6.390312e-04 2.324890e-04 1.902074e-01 1.945901e-04
##           CT           DE           FL           GA           HI           ID
## 1.991353e-01 1.763848e-03 7.809787e-02 6.672188e-05 5.490728e-03 1.188901e-05
##           IL           IN           IA           KS           KY           LA
## 5.396910e-02 3.336811e-03 1.988661e-03 3.188040e-02 3.493695e-04 2.381002e-02
##           ME           MD           MA           MI           MN           MS
## 3.723416e-02 5.072101e-04 9.073922e-03 9.461246e-05 6.850959e-03 5.483519e-04
##           MO           MT           NE           NV           NH           NJ
## 8.775059e-04 3.186878e-02 5.794353e-03 3.196340e-03 2.213457e-03 1.164791e-01
##           NM           NY           NC           ND           OH           OK
## 1.323198e-03 5.295890e-02 3.635416e-03 9.983205e-03 1.923143e-03 5.610484e-04
##           OR           PA           RI           SC           SD           TN
## 1.278072e-02 4.842742e-02 2.347249e-03 1.841017e-03 5.771393e-03 3.257290e-02
##           TX           UT           VT           VA           WA           WV
## 2.019262e-04 1.854943e-01 1.539535e-03 6.294269e-02 2.226314e-02 1.117743e-02
##           WI           WY
## 1.064380e-03 1.571104e-02
```

By using the following code, plot the leverages against Cook's Distance

```
ggplot(mapping = aes(x = cooks.distance(lm_salary1),
                     y = hatvalues(lm_salary1))) +

         geom_text(aes(label = row.names(sat))) +

         labs(x = "Cook's Distance",
              y = "Leverage") +

         theme_bw()
```



Because the leverages and Cook's distances do not live within the `sat` data set, we use the `mapping` feature of `ggplot()` and supply the vectors to `x` and `y` - Instead of `geom_point()`, we can use `geom_text()` and use the row names as the labels of the points

Cross Validation

Within this package is a function called `cvFit()`. This function has several arguments that we need to specify:

- object: the linear model we wish to test
- data: the data set
- y: the response variable
- cost: cost function - we will use RMSPE (root mean squared prediction error) as discussed in lecture
- K: number of folds (or groups)
- R: number of replications

Let's start with the smaller model and run 5-fold cross validation with 10 replications. Note: because these results will change from run-to-run, we will use `set.seed(1234)` in order to remove the randomness of these trials (and to ensure we all get the same answer). To see the individual RMSPE values for each replication, we can use `$reps` to access them.

```
set.seed(1234)
if (!requireNamespace("cvTools", quietly = TRUE)) {
    install.packages("cvTools")
}
library(cvTools)
```

```
## Loading required package: lattice
```

```
## Loading required package: robustbase
```

```
cv5_salary1 <- cvFit(lm_salary1,
                     data = sat,
                     y = sat$salary,
                     cost = rmspe,
                     K = 5,
                     R = 10)
cv5_salary1
```

```
## 5-fold CV results:
##       CV
## 2.256004
```

```
cv5_salary1$reps
```

```
##             CV
##  [1,] 2.268797
##  [2,] 2.263793
##  [3,] 2.229512
##  [4,] 2.255425
##  [5,] 2.216956
##  [6,] 2.247881
##  [7,] 2.176517
##  [8,] 2.370415
##  [9,] 2.197702
## [10,] 2.333040
```

We can also run Leave-One-Out Cross-Validation (LOOCV) by setting the number of folds to the number of rows in the data set.

```
loocv_salary1 <- cvFit(lm_salary1,
                       data = sat,
                       y = sat$salary,
                       cost = rmspe,
                       K = nrow(sat))
loocv_salary1
```

```
## Leave-one-out CV results:
##       CV
## 2.193253
```

Variable Selection

We can perform all of the selection methods, including Forward Selection, Backward Elimination, and Stepwise Regression, using the `step()` function:

- object: the *linear model* of the "starting" point
- scope: the *formula* of the "ending" point
- direction: "forward", "backward", or "both" (stepwise)
- trace: TRUE or FALSE (TRUE outputs the individual steps, FALSE does not)

```
fs_model1 <- step(object = lm(mpg ~ 1, data = mtcars),
                  scope = mpg ~ cyl + disp + hp + drat + wt + qsec + vs + am + gear + carb,
                  direction = "forward",
                  trace = FALSE)

summary(fs_model1)
```

```
##
## Call:
## lm(formula = mpg ~ wt + cyl + hp, data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.9290 -1.5598 -0.5311  1.1850  5.8986
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 38.75179    1.78686  21.687  < 2e-16 ***
## wt          -3.16697    0.74058  -4.276 0.000199 ***
## cyl         -0.94162    0.55092  -1.709 0.098480 .
## hp          -0.01804    0.01188  -1.519 0.140015
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.512 on 28 degrees of freedom
## Multiple R-squared:  0.8431, Adjusted R-squared:  0.8263
## F-statistic: 50.17 on 3 and 28 DF,  p-value: 2.184e-11
```

```
be_model <- step(object = lm(mpg ~ ., data = mtcars),
                 scope = mpg ~ 1,
                 direction = "backward",
                 trace = FALSE)

summary(be_model)
```

```
##
## Call:
## lm(formula = mpg ~ wt + qsec + am, data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.4811 -1.5555 -0.7257  1.4110  4.6610
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   9.6178     6.9596   1.382 0.177915
## wt           -3.9165     0.7112  -5.507 6.95e-06 ***
## qsec          1.2259     0.2887   4.247 0.000216 ***
## am            2.9358     1.4109   2.081 0.046716 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.459 on 28 degrees of freedom
## Multiple R-squared:  0.8497, Adjusted R-squared:  0.8336
## F-statistic: 52.75 on 3 and 28 DF,  p-value: 1.21e-11
```

To run stepwise regression, change the direction to both:

```
sr_model <- step(object = lm(mpg ~ ., data = mtcars),
                 scope = mpg ~ 1,
                 direction = "both",
                 trace = FALSE)
summary(sr_model)
```

```
##
## Call:
## lm(formula = mpg ~ wt + qsec + am, data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.4811 -1.5555 -0.7257  1.4110  4.6610
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   9.6178     6.9596   1.382 0.177915
## wt           -3.9165     0.7112  -5.507 6.95e-06 ***
## qsec          1.2259     0.2887   4.247 0.000216 ***
## am            2.9358     1.4109   2.081 0.046716 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.459 on 28 degrees of freedom
## Multiple R-squared:  0.8497, Adjusted R-squared:  0.8336
## F-statistic: 52.75 on 3 and 28 DF,  p-value: 1.21e-11
```

Shrinkage Method

To perform ridge regression, we will utilize the `glmnet` function. This function takes in our response variable, the design matrix, and a value for alpha. Alpha is a tuning parameter for elastic net.

- When set to 0, the function runs Ridge Regression.
- When set to 1, the function runs Lasso

The `model.matrix()` function creates the design matrix for the specified linear model.

```
if (!requireNamespace("glmnet", quietly = TRUE)) {
    install.packages("glmnet")
}
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

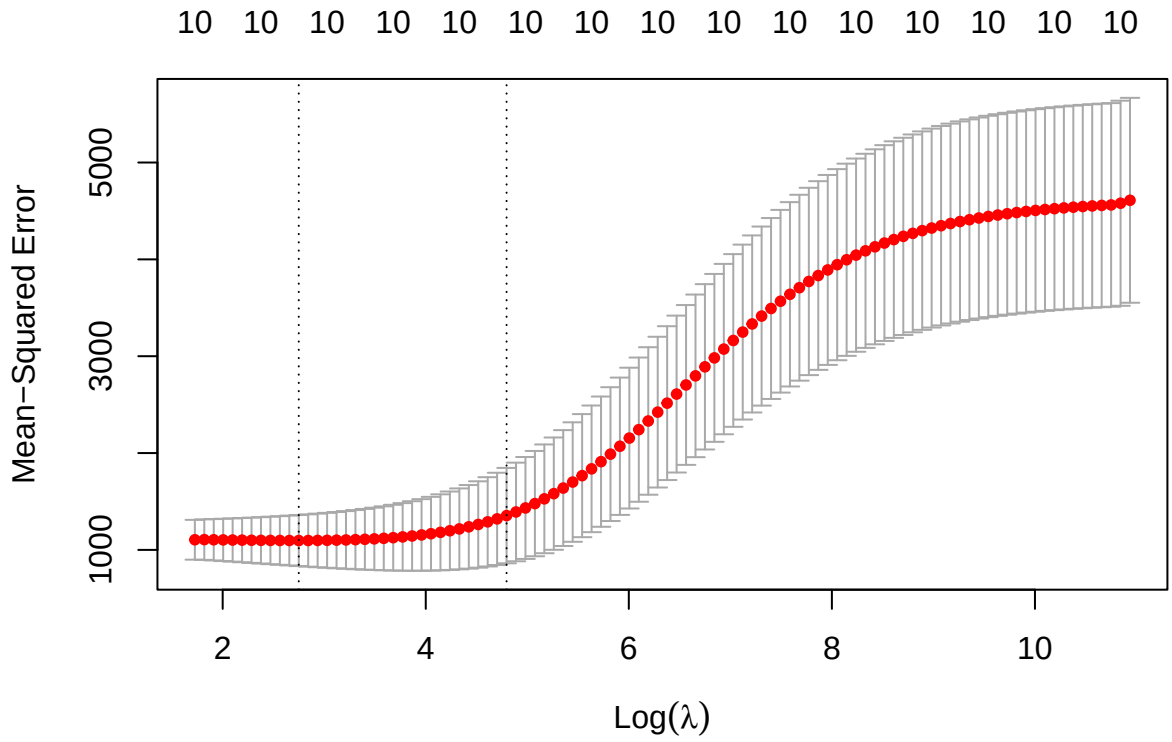
```
y <- mtcars$hp
x <- model.matrix(lm(hp ~ ., data = mtcars))
ridge_model <- glmnet(x, y, alpha = 0)
coef(ridge_model, c(0.1, 1, 10, 100))
```

```
## 12 x 4 sparse Matrix of class "dgCMatrix"
##                       s1          s2          s3           s4
## (Intercept) 145.4197146 145.4197146 161.5416116 194.02044277
## (Intercept)   .           .           .           .
## mpg          -1.6039712  -1.6039712  -1.5920122  -1.27933732
## cyl           8.9076274   8.9076274   8.1023254   4.87460970
## disp          0.2084335   0.2084335   0.1682053   0.07116354
## drat         -3.3550844  -3.3550844  -3.5301394  -4.78016475
## wt           -0.9026725  -0.9026725   2.4467465   5.72257253
## qsec         -7.3573035  -7.3573035  -7.5755964  -5.28184174
## vs           13.3744893  13.3744893   6.8295149 -11.41215937
```

```
## am           4.6382009   4.6382009   3.5020311  -0.17041756
## gear         9.5782001   9.5782001   8.9873768   3.09196804
## carb        11.9185074  11.9185074  10.9072187   6.57956186
```

To find the optimal value for lambda, we can use the package's `cv.glmnet()` function to run cross validation. By default, the function will generate its own sequence for lambda (which is recommended). Passing the stored results through the plot function provides us with a good visual for the model fit versus log(lambda).

```
cv_ridge <- cv.glmnet(x, y, alpha = 0)

plot(cv_ridge)
```



To retrieve the optimal lambda value, using the following code:

```
set.seed(1234)

opt_ridge_lambda <- cv_ridge$lambda.min

opt_ridge_lambda
```

```
## [1] 15.63134
```

```
coef(ridge_model, opt_ridge_lambda)
```

```
## 12 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept) 172.3242459
## (Intercept)   .
## mpg          -1.5785075
## cyl           7.4041494
## disp          0.1413014
## drat         -3.7471821
## wt            4.2244414
```

```
## qsec         -7.4546376
## vs            1.4481843
## am            2.6942526
## gear          8.1937352
## carb         10.1906801
```

Weighted Least Squares

```
peas <- read.csv("galtonpeas.csv")
wls_peas <- lm(Progeny ~ Parent, data = peas, weights = 1/SD^2)
summary(wls_peas)
```

```
##
## Call:
## lm(formula = Progeny ~ Parent, data = peas, weights = 1/SD^2)
##
## Weighted Residuals:
##       1       2       3       4       5       6       7
##  0.08187  0.09162 -0.16753 -0.04067 -0.08950  0.06071  0.06328
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 12.79642    0.68112  18.787 7.87e-06 ***
## Parent       0.20480    0.03815   5.368  0.00302 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.11 on 5 degrees of freedom
## Multiple R-squared:  0.8521, Adjusted R-squared:  0.8225
## F-statistic: 28.81 on 1 and 5 DF,  p-value: 0.003021
```

Logistic Regression

Create and store a model called `logistic_diabetes` that uses *both* `chol` *and* `age` to predict the response `diabetic`. Additional variables get added into the model the same as before (just use the `+` operator). Pass the stored model through the `summary()` function to examine the effect estimates and the significance.

```
diabetes <- read.csv("diabetes.csv")
logistic_diabetes <- glm(diabetic ~ chol + age, data = diabetes, family = "binomial")
summary(logistic_diabetes)
```

```
##
## Call:
## glm(formula = diabetic ~ chol + age, family = "binomial", data = diabetes)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -7.834223   1.586990  -4.937 7.95e-07 ***
## chol         0.016310   0.005434   3.001  0.00269 **
## age          0.054101   0.018173   2.977  0.00291 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 140.45  on 129  degrees of freedom
## Residual deviance: 112.96  on 127  degrees of freedom
## AIC: 118.96
##
## Number of Fisher Scoring iterations: 5
```

The effect estimates get interpreted the same as above, with the caveat that we are now controlling for the other variables in the model.

Because the second logistic regression model is nested within the first model, we can test the deviance to see if including `age` results in a significant improvement.

```
logistic_chol <- glm(diabetic ~ chol, data = diabetes, family = "binomial")
anova(logistic_chol, logistic_diabetes, test = "LRT")
```

```
## Analysis of Deviance Table
##
## Model 1: diabetic ~ chol
## Model 2: diabetic ~ chol + age
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1       128     123.23
## 2       127     112.96  1   10.272 0.001351 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We find that the results are statistically significant!

Now that there are two predictor variables, plotting can be a little trickier - so we use the `predictorEffects()` function within the `effects()` package to help us out. This function plots the effect of one variable while holding the other variable(s) constant.

```
plot(predictorEffects(logistic_diabetes), rescale.axis = FALSE, grid = TRUE)
```

```
## Error in h(simpleError(msg, call)): error in evaluating the argument 'x' in selecting a method for fu
```

To calculate the predicted/fitted value for the a new observation, we now have the following code:

```
predict(logistic_diabetes, data.frame(chol = 350, age = 40), type = "response")
```

```
##         1
## 0.5095578
```

Bootstrapping

The code below creates a linear that predicts horsepower from fuel efficiency. Using the `summary()` function, we find the point estimate for the slope coefficient (-8.83). We would use this as our estimate of the true slope.

```
lm_cars <- lm(hp ~ mpg, data = mtcars)

summary(lm_cars)
```

```
##
## Call:
## lm(formula = hp ~ mpg, data = mtcars)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -59.26 -28.93 -13.45  25.65 143.36
##
```

```
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   324.08      27.43  11.813 8.25e-13 ***
## mpg            -8.83       1.31  -6.742 1.79e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 43.95 on 30 degrees of freedom
## Multiple R-squared:  0.6024, Adjusted R-squared:  0.5892
## F-statistic: 45.46 on 1 and 30 DF,  p-value: 1.788e-07
```

If we are unsure about the variability around this estimate (or the distribution it follows), bootstrapping can help us define that uncertainty.

In Section 1 of the code below, we first define how many replications we wish to run (this should be a very large number) and then we create an empty vector (of the same size) to store the results of each replication.

In Section 2 of the code below, we run a "for" loop which cycles through the code inside as many times as we specify (here we are running this from an index of 1 to the number of replications specified in Section 1). Inside the for loop, the first line of code takes a random sample (with replacement) from our data set. The `sample()` function use three arguments:

- The first argument is the range of what to sample. Here, we choose the entire `mtcars` data set (i.e. from 1 to the number of rows in the data set).
- The second argument is how many observations we wish to randomly take. Here, we would like to take the same size sample as the data set.
- The last argument is to specify that we wish to sample *with* replacement.

This function only returns random integers so we must pass it through the `mtcars[]` data frame to extract the data of the corresponding rows. The second line of code inside the for loop retrieves the estimate of the slope coefficient for the linear model run on the sample randomly drawn on the first line. This is then stored in index "i" of the empty vector we created in Section 1. Try it out! (Note: sometimes it takes a little bit of time to run.)

```
# Section 1
reps <- 10000
bootstrap_dist <- vector(length = reps)

# Section 2
for (i in 1:reps) {
  bootstrap_resample <- mtcars[sample(1:nrow(mtcars), nrow(mtcars), replace = TRUE), ]
  bootstrap_dist[i] <- lm(hp ~ mpg, data = bootstrap_resample)$coefficients[2]
}
```

To quantify the uncertainty of our estimate, we take the standard deviation of the stored bootstrapped estimates.

```
sd(bootstrap_dist)
```

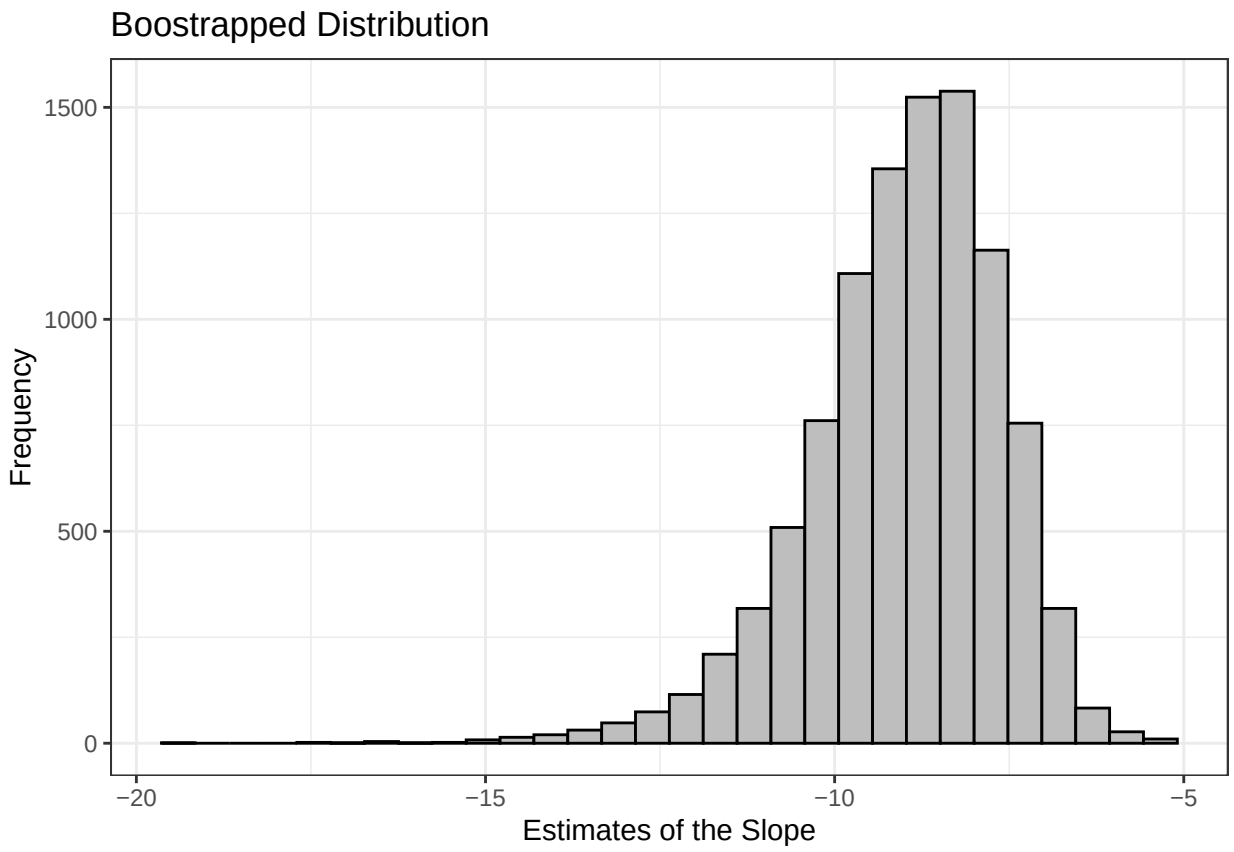
```
## [1] 1.373593
```

From the original model, the standard error of the slope estimate was 1.31. This value shouldn't be too far off.

Finally, we can plot the bootstrapped distribution to get an idea of its shape.

```
# First store the bootstrap vector as a data frame
bootstrapped_data <- data.frame(estimates = bootstrap_dist)

# Then use ggplot to visualize the bootstrapped data
```

```r
ggplot(data = bootstrapped_data, aes(x = estimates)) +
  geom_histogram(bins = 30, color = "black", fill = "grey")+
  labs(title = "Boostrapped Distribution",
       x = "Estimates of the Slope",
       y = "Frequency")+
  theme_bw()
```



The distribution is roughly symmetric with a slight left skew.

Bootstrapping is a very helpful technique that can be used to quantify the uncertainty for any statistic (mean, median, variance, slope estimate!
