CUNY SPS MSDS - DATA605 - Final

Keith Colella

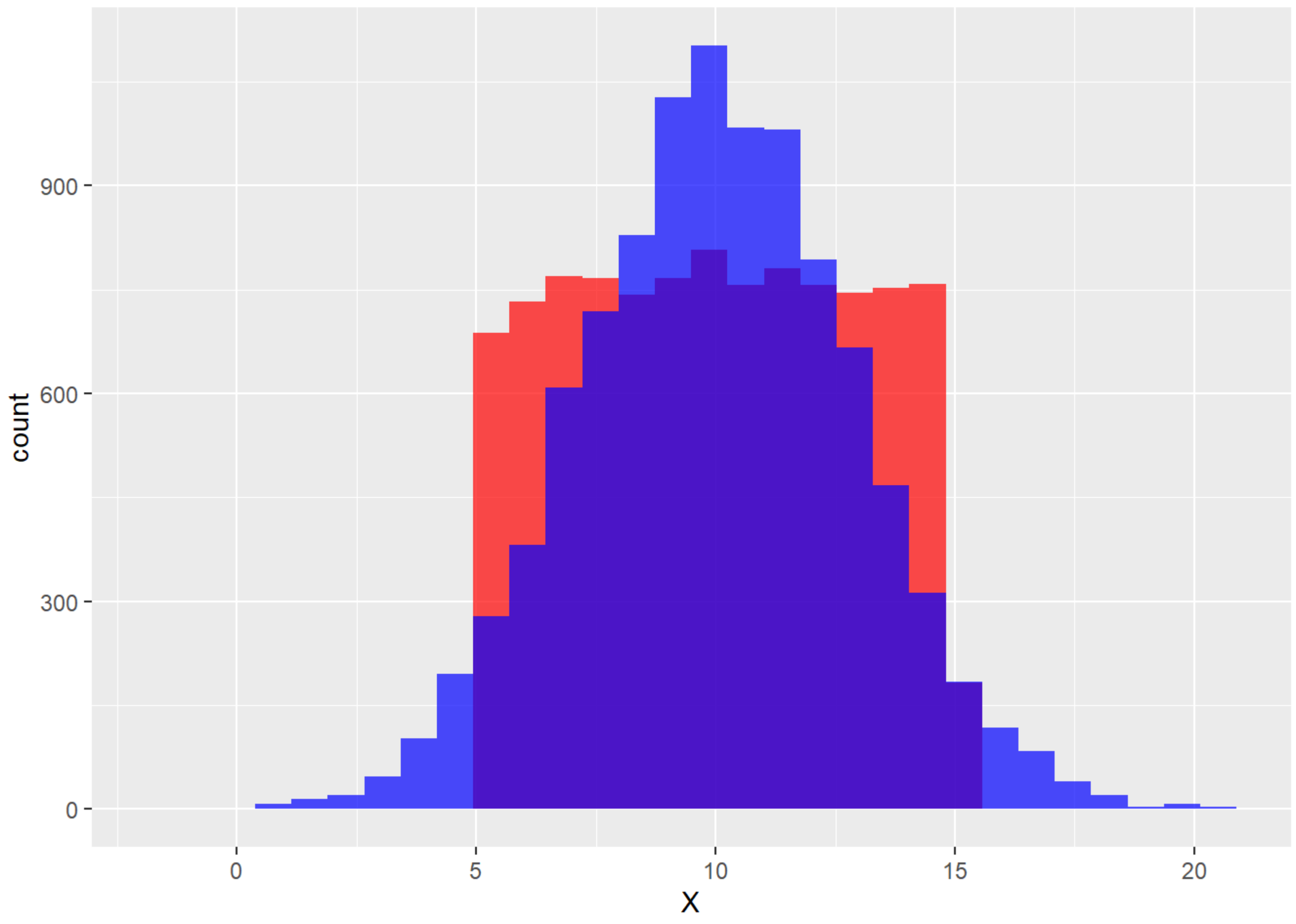
2023-12-17

```
library(tidyverse)
```

Problem 1

Using R, set a random seed equal to 1234 (i.e., set.seed(1234)). Generate a random variable X that has 10,000 continuous random uniform values between 5 and 15.Then generate a random variable Y that has 10,000 random normal values with a mean of 10 and a standard deviation of 2.89.

```
set.seed(1234)

X <- runif(n = 10^4, min = 5, max = 15)

Y <- rnorm(n = 10^4, mean = 10, sd = 2.89)

data.frame(X = X, Y = Y) %>%
  ggplot() +
  geom_histogram(aes(X), bins = 30, fill = 'red', alpha = 0.7) +
  geom_histogram(aes(Y), bins = 30, fill = 'blue', alpha = 0.7)
```



Part 1

Calculate as a minimum the below probabilities a through c. Assume the small letter “x” is estimated as the median of the X variable, and the small letter “y” is estimated as the median of the Y variable. Interpret the meaning of all probabilities.

a. P(X>x | X>y)

We can approach this question both conceptually and empirically, given the distributions we generated above. First, let’s set x and y.

```
x <- median(X)
y <- median(Y)

cat(
  'Median of X: ',x,'\n',
  'Median of Y: ',y,
  sep = ''
)
```

```
## Median of X: 10.01266
## Median of Y: 10.03154
```

These values, both close to 10, conform to expectations. The median of both uniform and normal distributions are very close to the means, and we know the mean of both distributions will be ~10. For X, it's the halfway point between the min and max (5 and 15), and for Y, it's given.

Let's reframe our question with this in mind. We are looking for the probability that our random variable is greater than x, and we already know that our random variable is greater than y. Since both of those values are ~10, our probability should be very close to 1.

We can confirm this by creating a subset of cases where $X > y$, then calculating the proportion of those cases where X is also less than x.

```
B <- X[X>y]
A <- B[B>x]

proportion <- length(A) / length(B)

cat('P(A|B):', proportion)
```

```
## P(A|B): 1
```

Within our random sample, 100% of the values of X within the subset $X > y$ are greater than x. In other words, $P(X > x|X > y) = 1$.

b. P(X>x & Y>y)

Whereas part (a) asked for a conditional probability, we're now looking at a joint probability. In other words, we're given nothing, and we have to find the probability that both of these conditions are met simultaneously.

Individually, the probability that a random variable is greater than the median of its respective distribution is 50%, since the median is, by definition, the 50th percentile. To be a bit more explicit, with a sample of 10,000, the probability would be

$$P(X > x) = \frac{(10,000/2)}{10,000} = \frac{5000}{10,000} = 0.5$$

And the same would apply to $P(Y > y)$. Moreover, we can estimate a joint probability $(PA \cap B)$ as the product of each individual probability: $P(A) \times P(B)$. So in this case, we would expect the probability to be

$$P(X > x \cap Y > y) = P(X > x) \times P(Y > y) = 0.4999 * 0.4999 = 0.2499$$

Let's check this empirically. We can calculate the proportion of cases where $X > x$ compared to the total length of X, and the same for $Y > y$, then multiply the two.

```
P_A <- length(X[X>x]) / length(X)
P_B <- length(Y[Y>y]) / length(Y)

P_AandB <- P_A * P_B

print(P_AandB)
```

```
## [1] 0.25
```

It matches! Another way to check this empirically is to randomly sample from both distributions, then calculate the proportion of cases where both samples are greater than their respective medians.

```
results <- data.frame()

for (n in 1:1000) {
  x_sample <- sample(X, 1)
  y_sample <- sample(Y, 1)
  result <- list(X = x_sample, Y = y_sample)
  results <- rbind(results, result)
}

proportion <- nrow(filter(results, X > x, Y > y)) / nrow(results)

print(proportion)
```

```
## [1] 0.25
```

Our estimate of 0.2499 holds true!

c. P(X<x | X>y)

We again have a condition probability, so we can employ a similar approach as our first problem. Moreover, given what we know about the first problem (i.e. that it's one), we can expect this probability to approach 0. Specifically, we know that x and y are both very close to 10. So if X is greater than 10 (the given condition), then the chances of it also being less than 10 should be zero.

Let's check it from our distributions.

```
B <- X[X>y]
A <- B[B<x]

proportion <- length(A) / length(B)

cat('P(A|B):', proportion)
```

```
## P(A|B): 0
```

Indeed, we find no cases where X is both greater than y and less than x, because both x and y are ~10.

Part 2

Investigate whether P(X>x & Y>y)=P(X>x)P(Y>y) by building a table and evaluating the marginal and joint probabilities.

We used this definition of joint probability above. Let's now build a contigency table to substantiate the definition. We'll use conditional subsets to count the number of cases the meet the conditions of interest ($X > x$ and $Y > y$, as well as their complements ($X \leq x$ and $Y \leq y$). From there, we can divide by $n = 10,000$ to convert the table to show the proportion that each subset represents of the total sample, which gives us their probabilities.

```
contingency <- table(
  factor(levels = c('X>x','X≤x','margin')),
  factor(levels = c('Y>y','Y≤y','margin'))
)

contingency['X>x', 'margin'] <- length(X[X>x])
contingency['margin', 'Y>y'] <- length(Y[Y>y])

contingency['X≤x', 'margin'] <- length(X[X<=x])
contingency['margin', 'Y≤y'] <- length(Y[Y<=y])

contingency['X>x', 'Y>y'] <- length(X[(X>x) & (Y>y)])
contingency['X>x', 'Y≤y'] <- length(X[(X>x) & (Y<=y)])

contingency['X≤x', 'Y>y'] <- length(X[(X<=x) & (Y>y)])
contingency['X≤x', 'Y≤y'] <- length(X[(X<=x) & (Y<=y)])

print(contingency)
```

```
##
##           Y>y  Y≤y margin
##   X>x    2507 2493   5000
##   X≤x    2493 2507   5000
##   margin 5000 5000      0
```

```
contingency_prop <- contingency/10^4

print(contingency_prop)
```

```
##
##             Y>y    Y≤y margin
##   X>x    0.2507 0.2493 0.5000
##   X≤x    0.2493 0.2507 0.5000
##   margin 0.5000 0.5000 0.0000
```

We can now see that the marginal probabilities $P(X > x)$ and $P(Y > y)$ both equal 0.5. This conforms to expectations, given that (as noted above), x and y represent the medians (or 50th percentiles) of their respective distributions. Moreover, we see that the joint probability $P(X > x \cap Y > y)$ is ~0.2507. Let’s see if the product of $P(X > x)$ and $P(Y > y)$ comes out to the same.

```
contingency_prop['X>x','margin'] * contingency_prop['margin', 'Y>y']
```

```
## [1] 0.25
```

Indeed, it matches. Our estimated probability in the contigency table doesn’t *exactly* equal 0.25 because our random sampling is, well, random!

Part 3

Check to see if independence holds by using Fisher's Exact Test and the Chi Square Test. What is the difference between the two? Which is most appropriate? Are you surprised at the results? Why or why not?

Both of these tests examine whether there are nonrandom associations between two variables. In the absence of such associations, we can conclude that the variables are independent. Both tests take as the null hypothesis (H_0) that the variables are independent, and the alternative hypothesis (H_1) is that they are not. Both produce a p-value that, as it approaches 0, gives us more confidence in rejecting alternative hypothesis. Finally, both require a contigency table as an input.

In terms of differences, one key difference is related to sample size. Whereas the Chi-Squared test assumes that samples are random and representative, and that there are at least ~5 observations in each square of the contigency table, the Fisher Exact Test is typically used for smaller sample sizes (although it technically still applies to larger samples, too). Moreover, the Chi-Squared test relies upon the chi-squared distribution (which becomes more accurate as sample size increases), whereas the Fisher Test relies upon no parametric distribution. Finally, the Fisher test is a bit more computationally intensive (especially with large sample sizes).

So, in terms of applicability, the Chi-Squared test is typically more appropriate for large sample sizes, while Fisher Test is often used with smaller samples. For situation, we have a very large sample of 10,000, so the Chi-Squared test seems most appropriate. Lucky for us, R has implementations for both of these tests, so let's check both!

```
p_chi <- chisq.test(contingency[1:2, 1:2])$p.value

p_fish <- fisher.test(contingency[1:2, 1:2])$p.value

cat(
  'Chi-Squared Test Result:', p_chi, '\n',
  'Fisher Exact Test Result:', p_fish,
  sep = ''
)
```

```
## Chi-Squared Test Result:0.7948638
## Fisher Exact Test Result:0.7948653
```

In both cases, our p-value is well above zero. So, with just about any reasonable significance level / α, we do not reject the null that the two variables are independent. Thus, we can conclude they *are* independent.

Given that we generated these variables with entirely independent processes (`rnorm` and `runif`), it is unsurprising that these tests helped us conclude that they are independent. Moreover, they come from very different different distributions, so again, one would expect independence.

Problem 2

You are to register for Kaggle.com (free) and compete in the Regression with a Crab Age Dataset competition: https://www.kaggle.com/competitions/playground-series-s3e16 (https://www.kaggle.com/competitions/playground-series-s3e16).

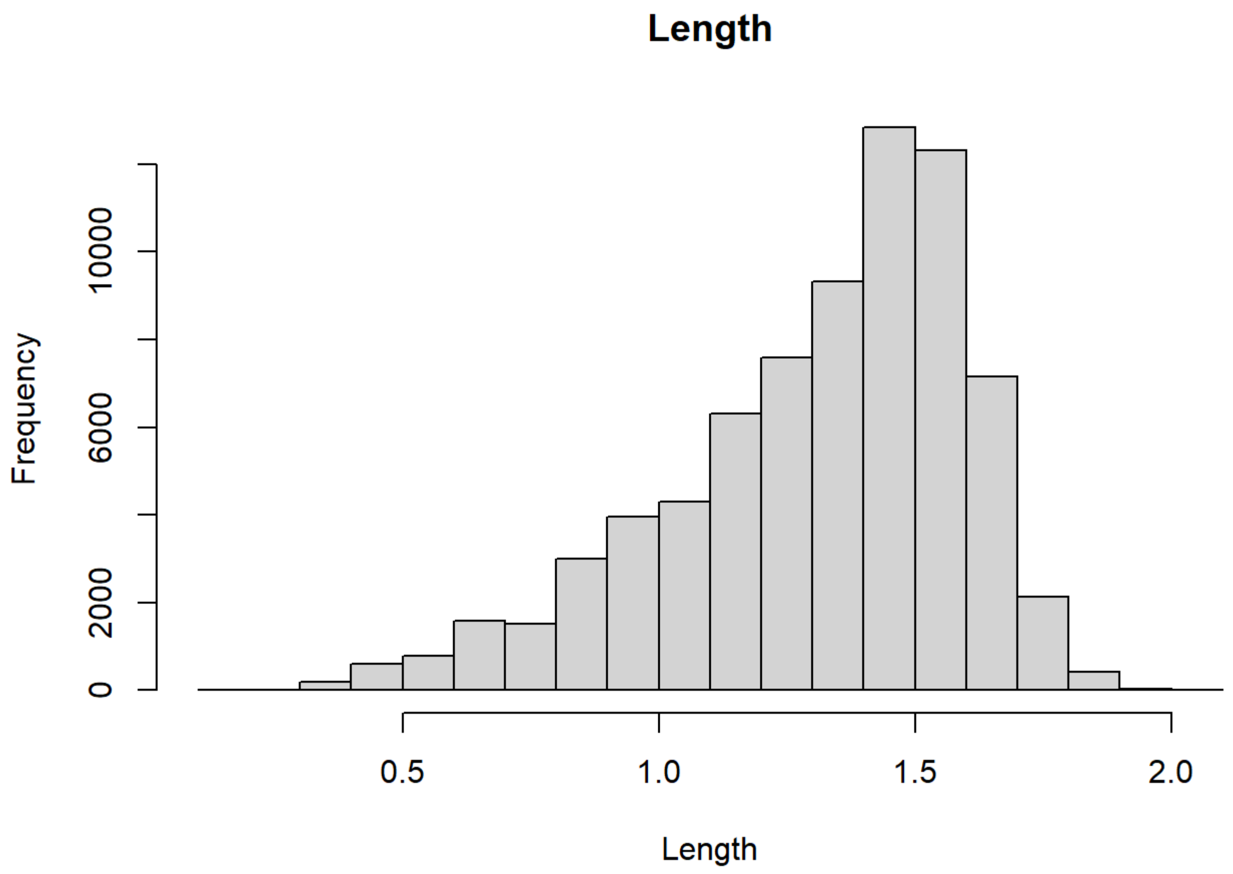
```
train <- read.csv('data/crab_train.csv')
test <- read.csv('data/crab_test.csv')
```

Descriptive and Inferential Statistics

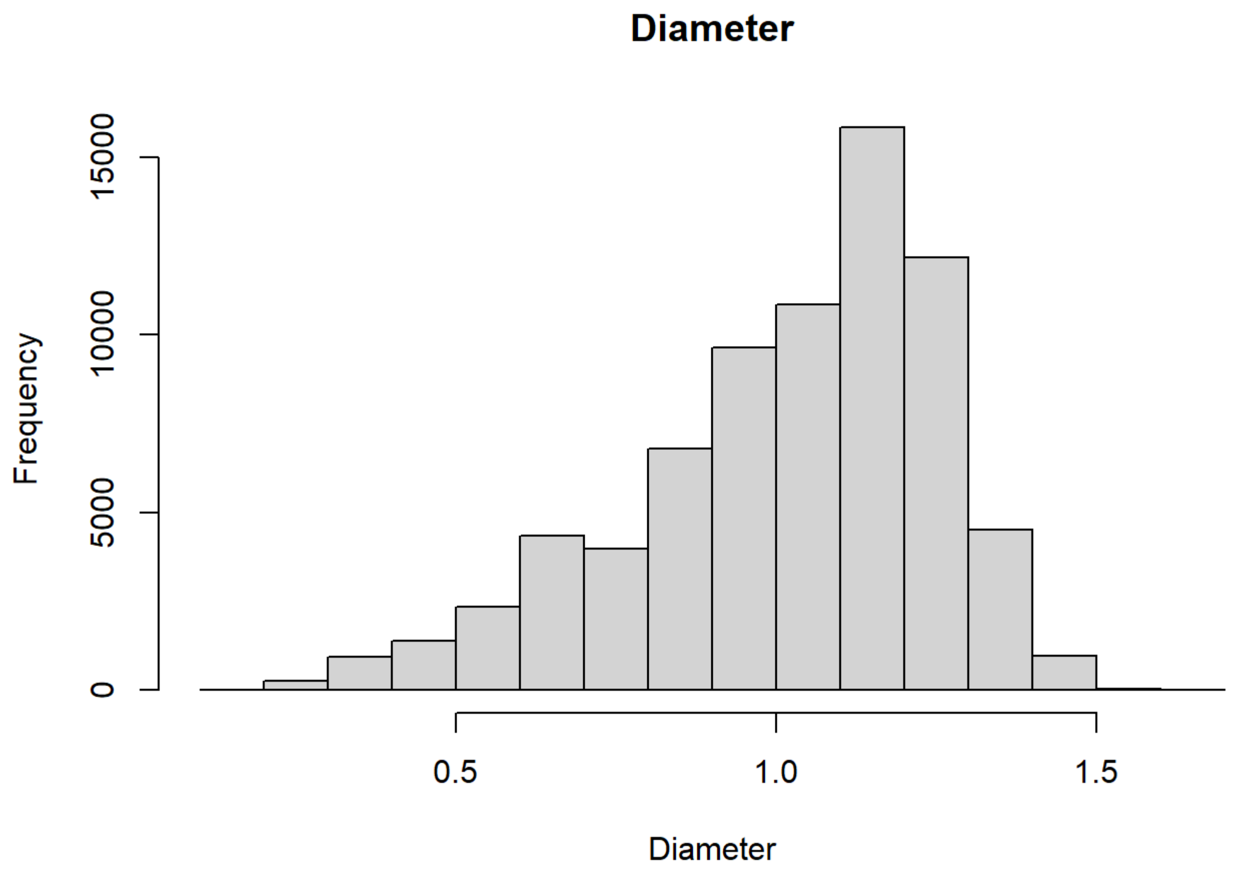
Provide univariate descriptive statistics and appropriate plots for the training data set.

```r
for (col in colnames(train)){
  if(col == 'id' | col == 'Sex') next
  cat('-----',col,'-----\n')
  print(summary(train[[col]]))
  hist(train[[col]], main = col, xlab = col)
}
```

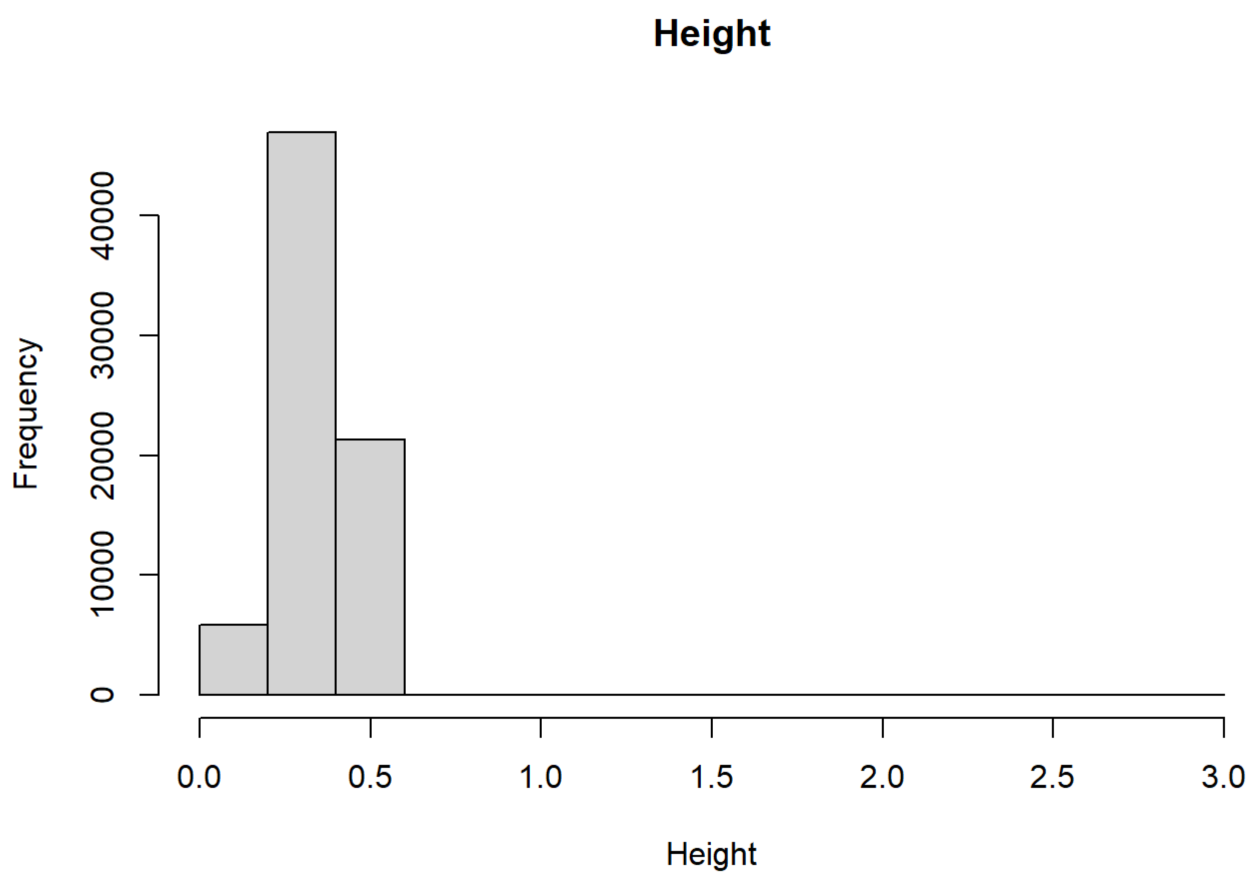
```
## ----- Length -----
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.1875  1.1500  1.3750  1.3175  1.5375  2.0128
```



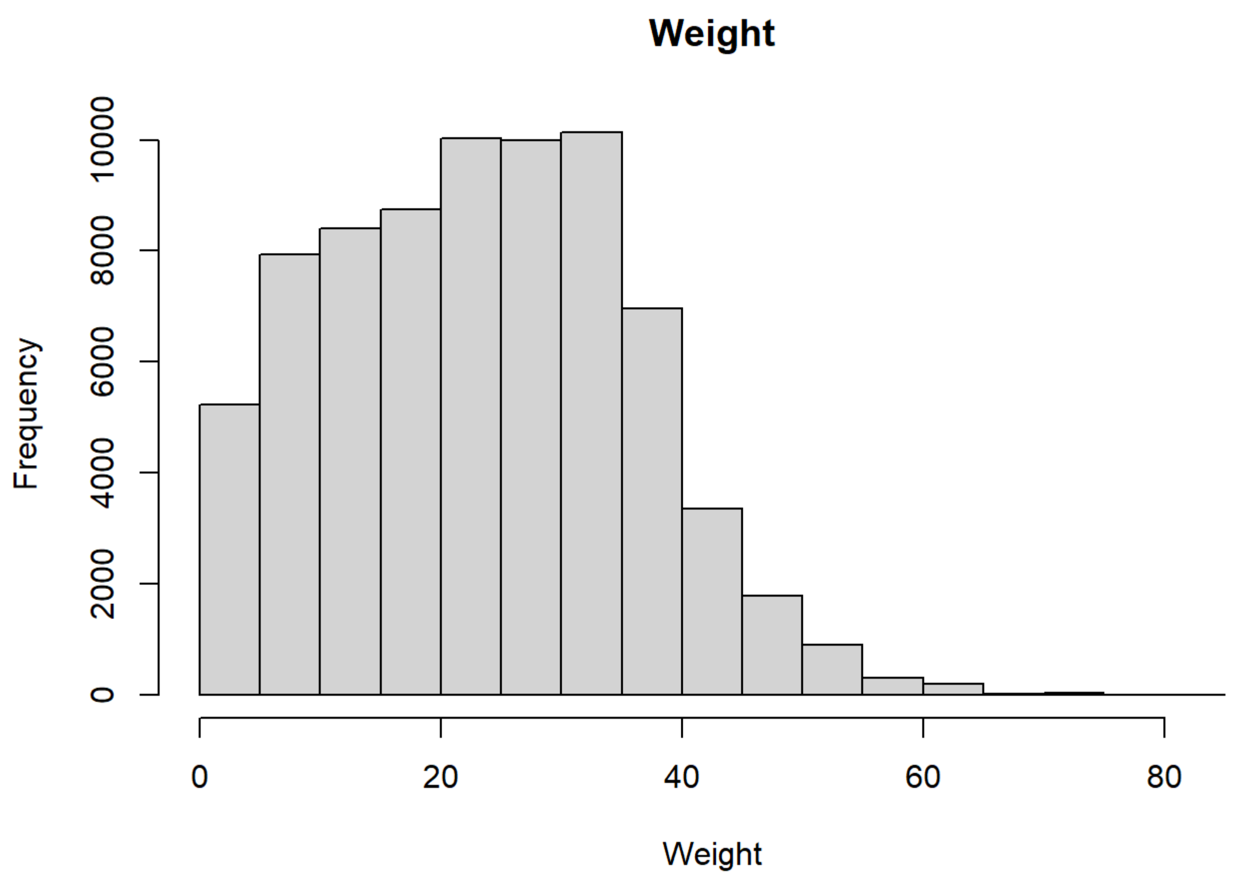
```
## ----- Diameter -----
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.1375  0.8875  1.0750  1.0245  1.2000  1.6125
```



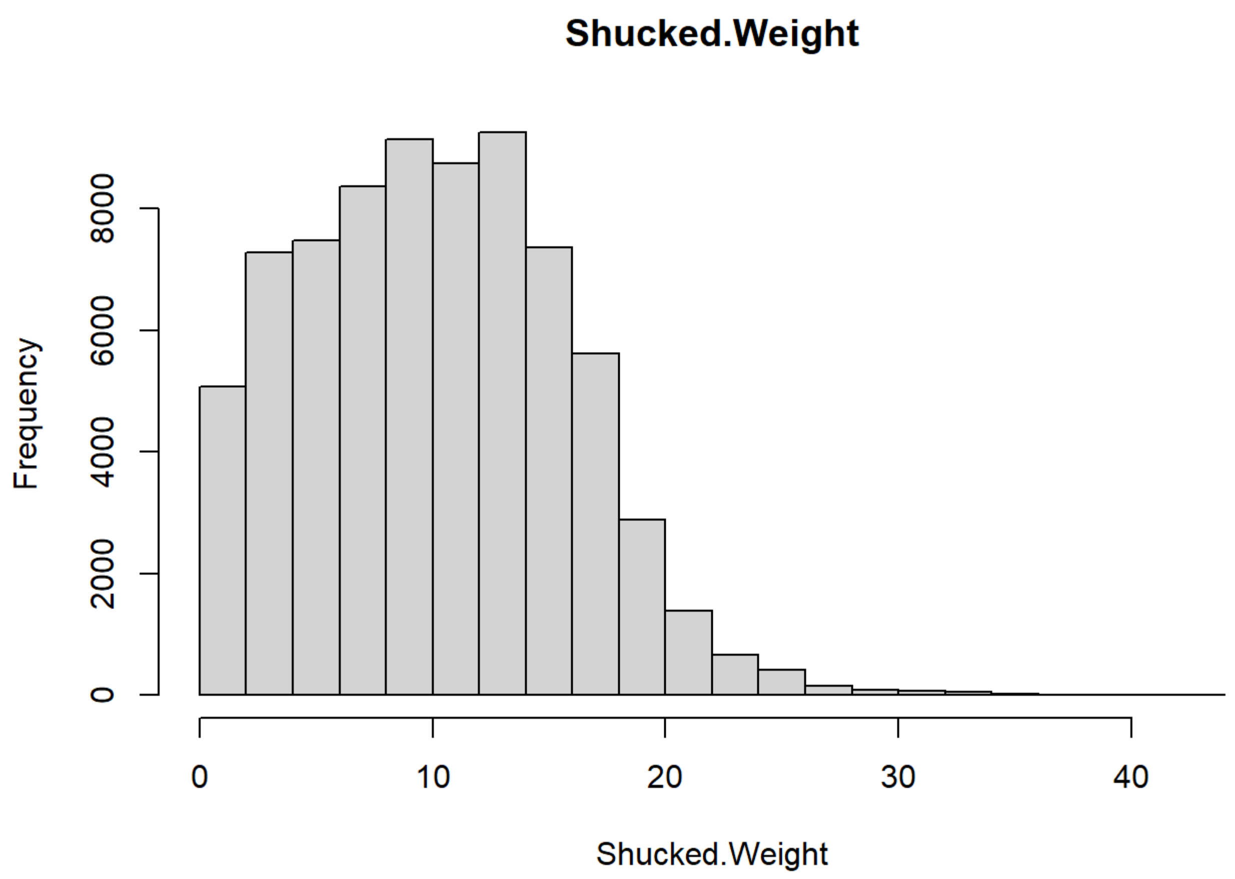
```
## ----- Height -----
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##  0.0000  0.3000  0.3625  0.3481  0.4125  2.8250
```



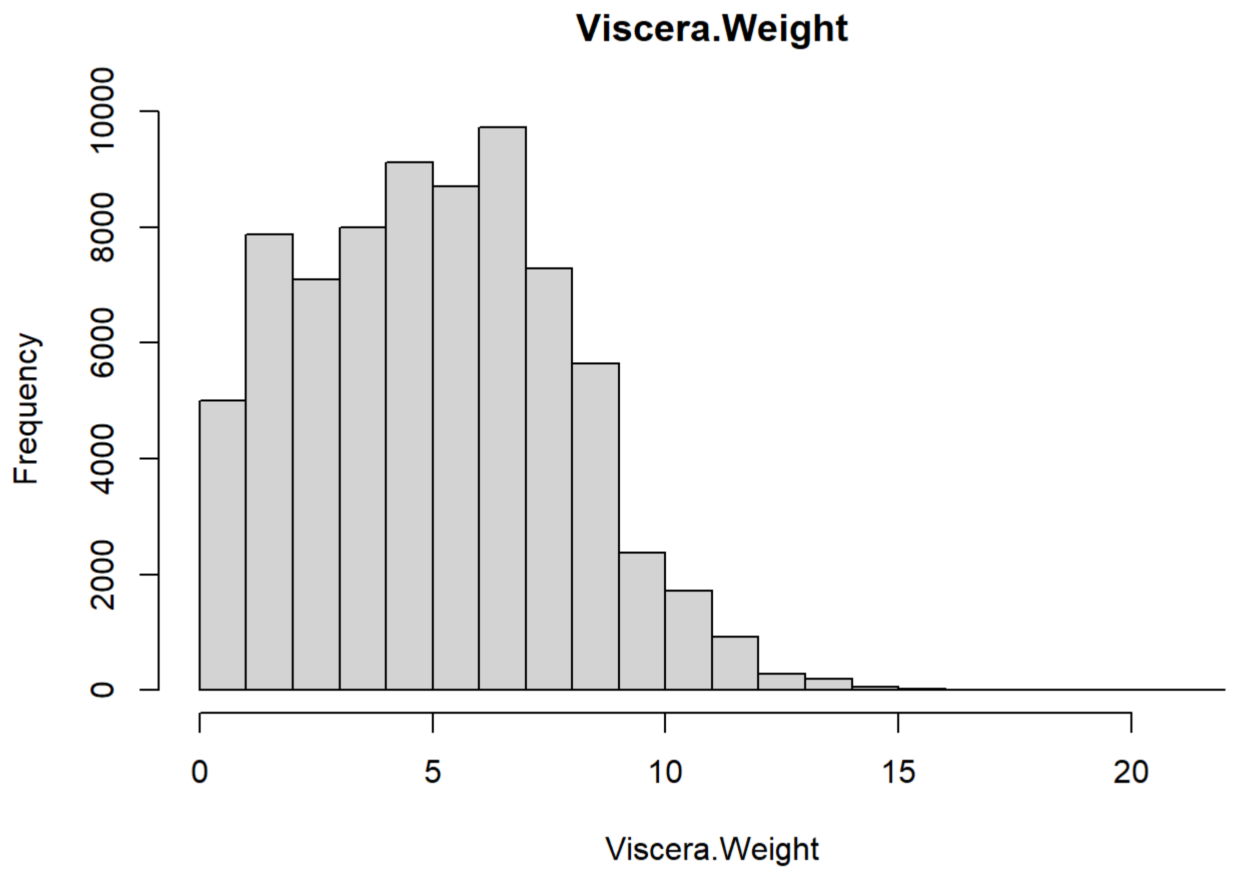
```
## ----- Weight -----
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0567 13.4377 23.7994 23.3852 32.1625 80.1015
```



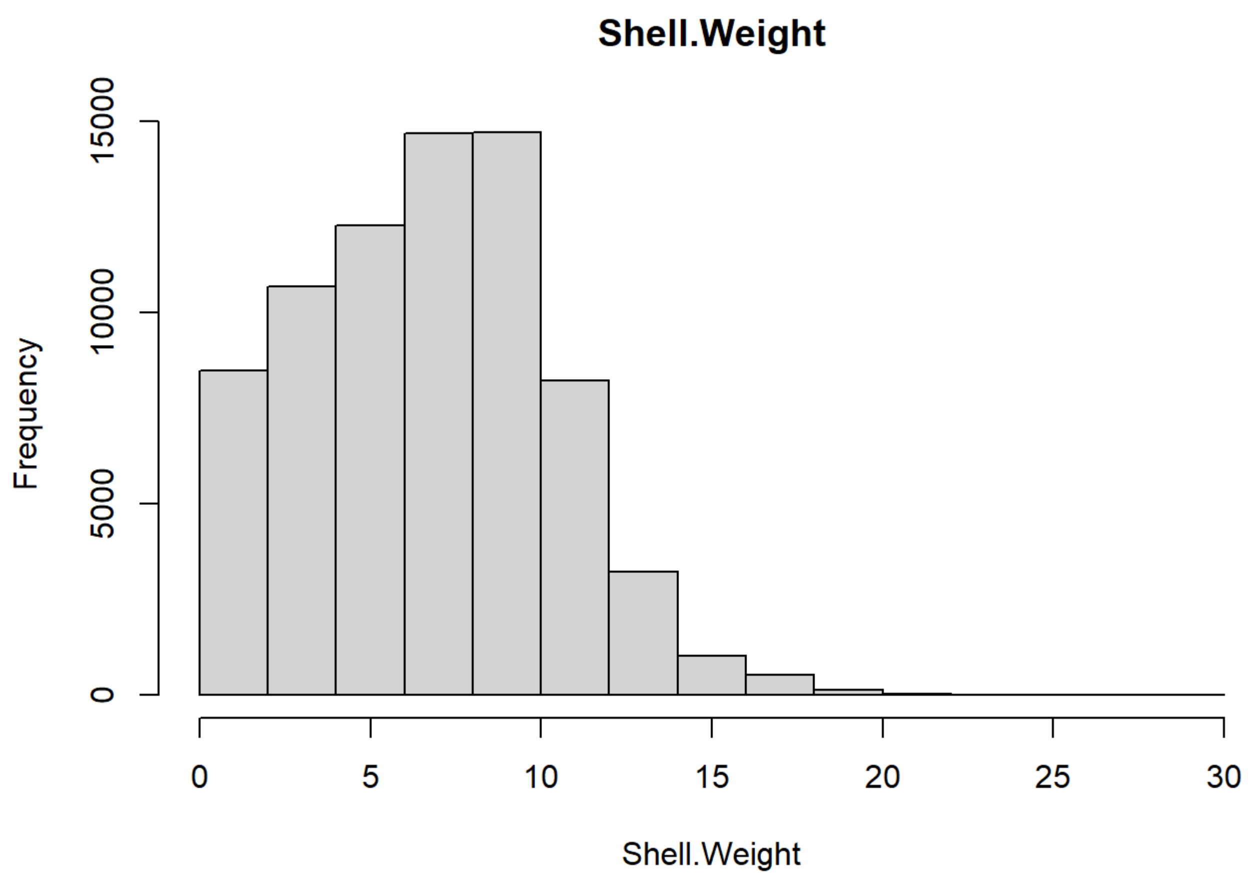
```
## ----- Shucked.Weight -----
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##  0.02835  5.71242  9.90815 10.10427 14.03300 42.18406
```



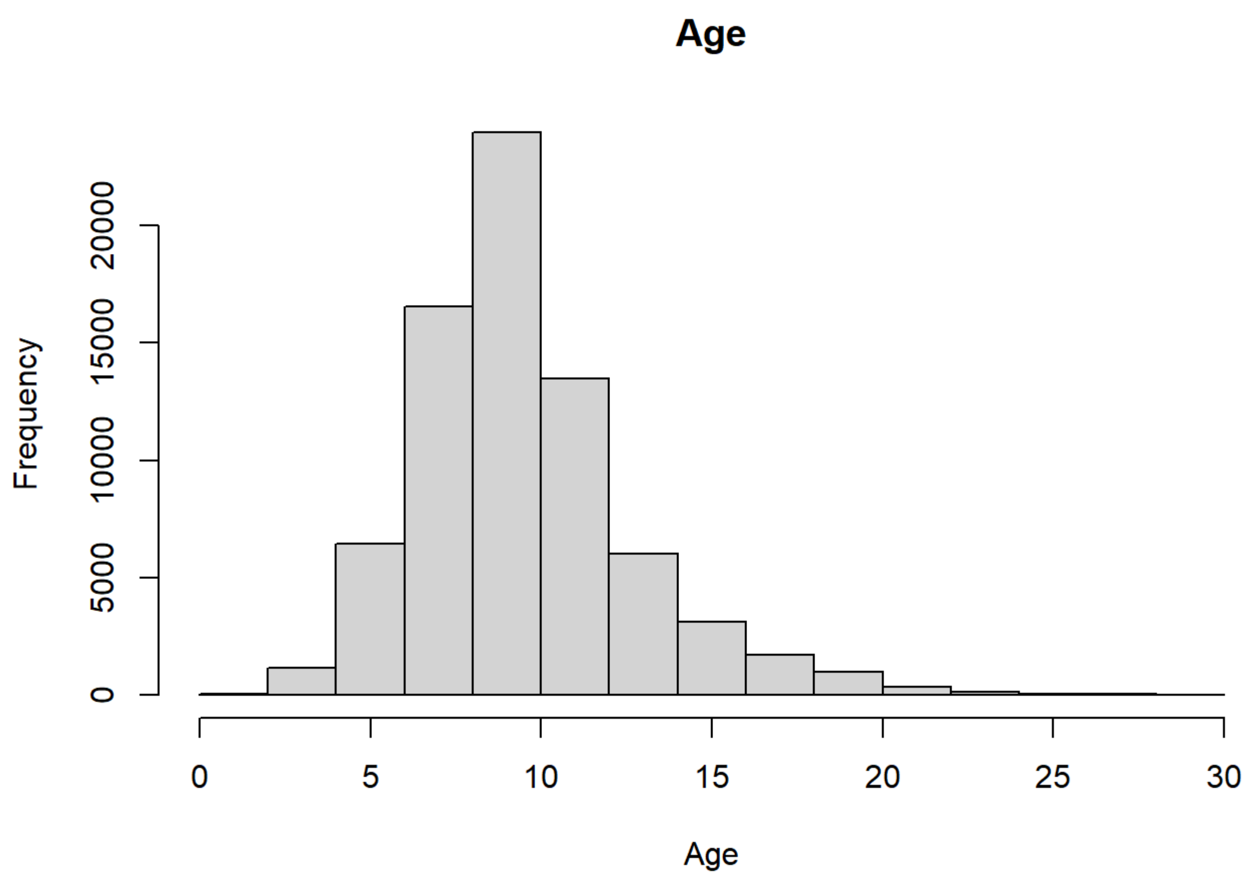
```
## ----- Viscera.Weight -----
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max. 
##  0.04252  2.86330  4.98951  5.05839  6.98815 21.54562
```



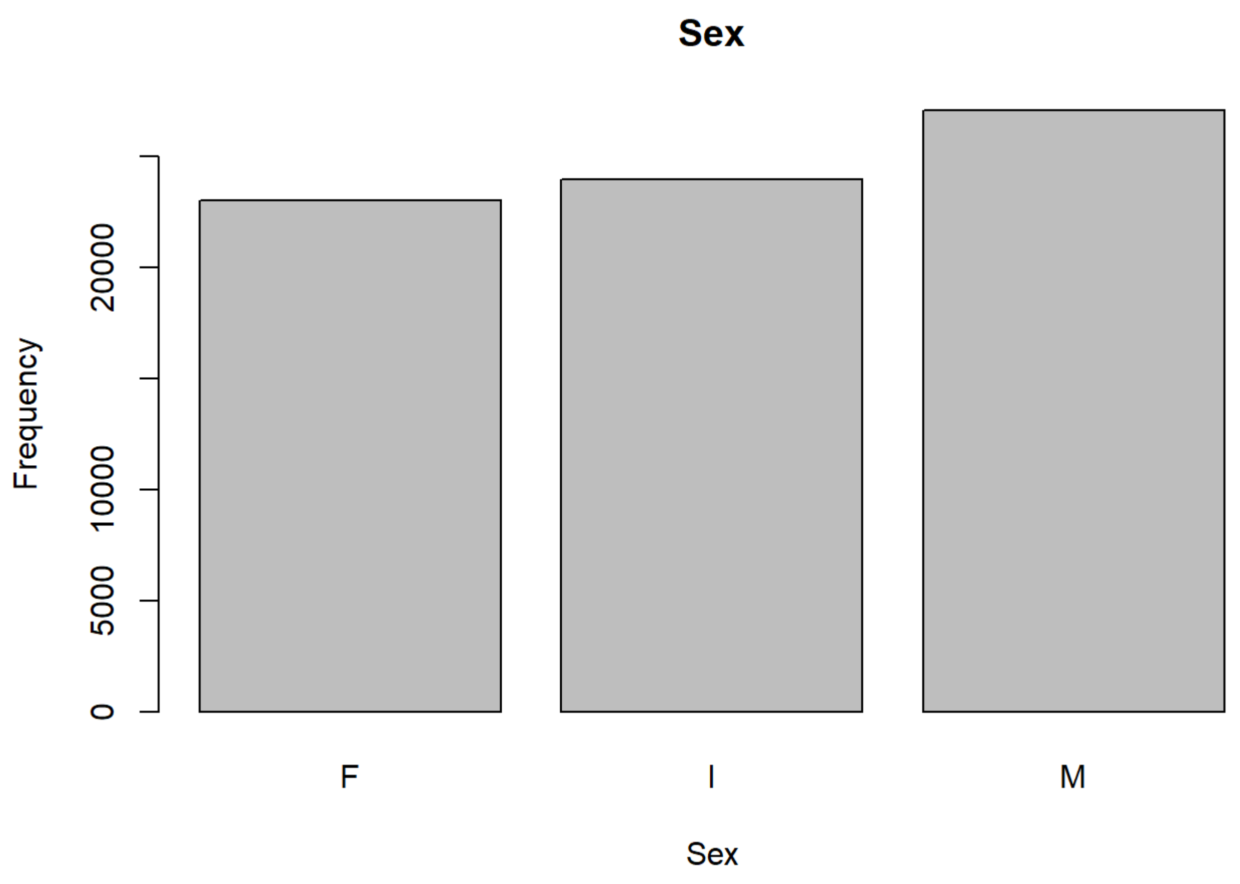
```
## ----- Shell.Weight -----
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max. 
##  0.04252  3.96893  6.93145  6.72387  9.07184 28.49125
```



```
## ----- Age -----
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   8.000  10.000   9.968  11.000  29.000
```



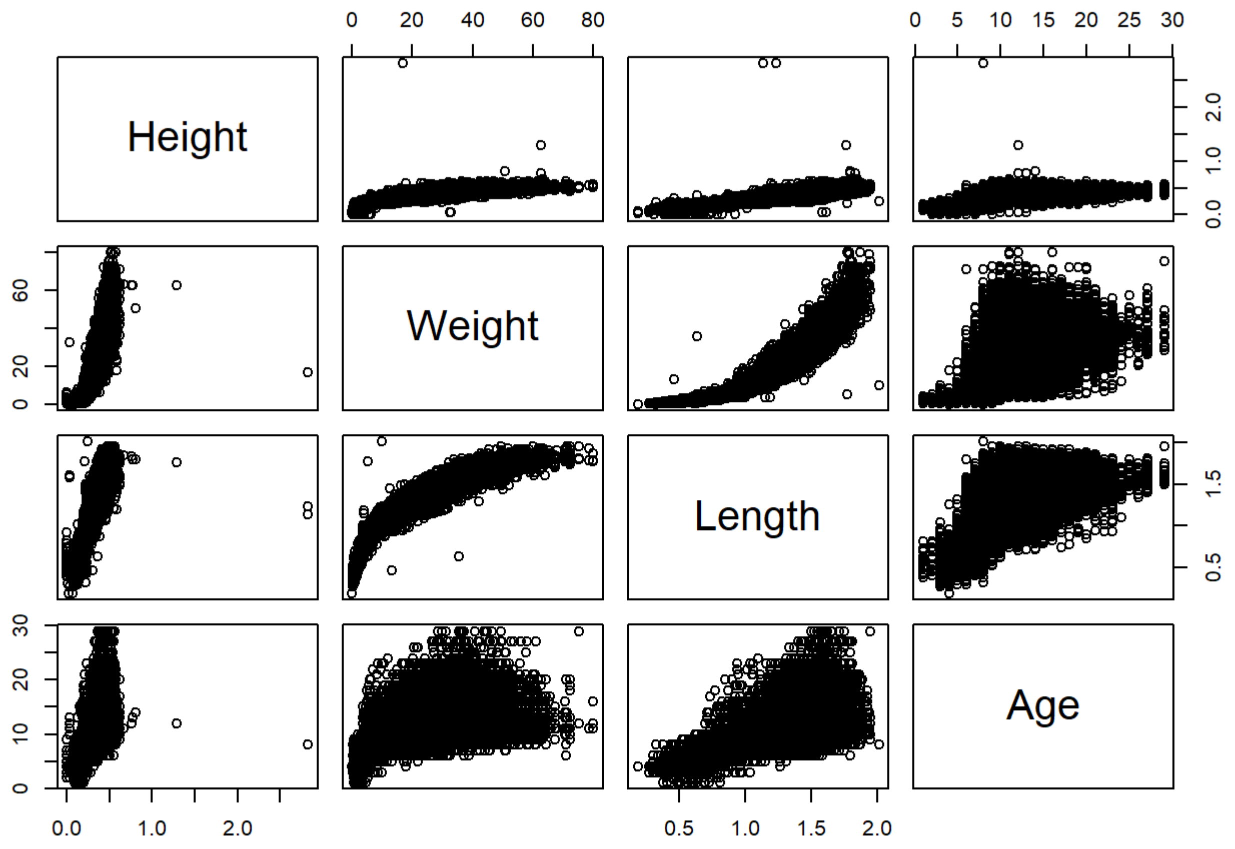
```
barplot(table(train$Sex), main = "Sex", xlab = "Sex", ylab = "Frequency")
```



Across nearly all of our variables, we see quite a bit of skew. Specifically, Length and Diameter exhibit left skew, whereas Height, Age and the various Weight measures exhibit right skew. When modeling, these variables may benefit from transformation. Sex, on the other hand, is relatively well balanced across the three classes.

Provide a scatterplot matrix for at least two of the independent variables and the dependent variable.

```
pairs(~ Height + Weight + Length + Age, data = train)
```



We see some evidence of linear relationships between these three primary predictors and Age. The relationships don’t appear perfectly linear, but they seem sufficient for explore in modeling. Moreover, some transformation may reinforce the linearity of the relationships.

Derive a correlation matrix for any three quantitative variables in the dataset.

```
cor(train[, c('Height', 'Weight', 'Length')])
```

```
##           Height    Weight    Length
## Height 1.0000000 0.9017748 0.9183517
## Weight 0.9017748 1.0000000 0.9363738
## Length 0.9183517 0.9363738 1.0000000
```

We see high correlation across these primary independent variables. This conforms to expectations, as all three relate to size. We’ll need to consider whether this multicollinearity could impact our modeling.

Test the hypotheses that the correlations between each pairwise set of variables is 0 and provide an 80% confidence interval. Discuss the meaning of your analysis. Would you be worried about familywise error? Why or why not?

```r
variables <- colnames(train)[-c(1,2)]
variables <- c(variables[length(variables)], variables[-length(variables)])
combos <- combn(variables, 2, simplify = FALSE)

results <- data.frame()
for (combo in combos) {
  cor_test <- cor.test(train[[combo[1]]], train[[combo[2]]], conf.level = 0.80)
  result <- list(
    Var1 = combo[1], Var2 = combo[2], p = cor_test$p.value,
    ci_lower = cor_test$conf.int[1], ci_lower = cor_test$conf.int[2]
  )
  results <- rbind(results, result)
}

results
```

```
##              Var1           Var2 p  ci_lower ci_lower.1
## 1             Age         Length 0 0.6098938  0.6157754
## 2             Age       Diameter 0 0.6183556  0.6241393
## 3             Age         Height 0 0.6352664  0.6408507
## 4             Age         Weight 0 0.5981791  0.6041938
## 5             Age Shucked.Weight 0 0.4997954  0.5068284
## 6             Age Viscera.Weight 0 0.5736566  0.5799419
## 7             Age   Shell.Weight 0 0.6608286  0.6661015
## 8          Length       Diameter 0 0.9893379  0.9895359
## 9          Length         Height 0 0.9176108  0.9190862
## 10         Length         Weight 0 0.9357910  0.9369515
## 11         Length Shucked.Weight 0 0.9147504  0.9162747
## 12         Length Viscera.Weight 0 0.9171100  0.9185939
## 13         Length   Shell.Weight 0 0.9162044  0.9177038
## 14       Diameter         Height 0 0.9206384  0.9220617
## 15       Diameter         Weight 0 0.9376824  0.9388098
## 16       Diameter Shucked.Weight 0 0.9134223  0.9149693
## 17       Diameter Viscera.Weight 0 0.9176101  0.9190854
## 18       Diameter   Shell.Weight 0 0.9219851  0.9233852
## 19         Height         Weight 0 0.9008913  0.9026508
## 20         Height Shucked.Weight 0 0.8628846  0.8652710
## 21         Height Viscera.Weight 0 0.8820858  0.8841588
## 22         Height   Shell.Weight 0 0.9025285  0.9042605
## 23         Weight Shucked.Weight 0 0.9709993  0.9715328
## 24         Weight Viscera.Weight 0 0.9707920  0.9713293
## 25         Weight   Shell.Weight 0 0.9652040  0.9658423
## 26 Shucked.Weight Viscera.Weight 0 0.9420988  0.9431486
## 27 Shucked.Weight   Shell.Weight 0 0.9095880  0.9112004
## 28 Viscera.Weight   Shell.Weight 0 0.9333145  0.9345182
```

Our testing indicates that **all** pairs of variables have correlations greater than 0. I'll note that the p-values from these tests are not actually zero. Instead, they are exceedingly small and approach zero; the `cor.test` function simply rounds it to zero because it's so small. Still, they all indicate that we should reject the null that correlations are zero, instead concluding they are non-zero.

While this is a “good” thing for variables’ relationship with Age (since it indicates they are likely good predictors), it also indicates that we’ll need to account for multicollinearity in modeling.

Because we are conducting a series of statistically tests concurrently, we should consider familywise error rate, which is the probability of false positives in our findings. With such a large series of tests (28 in total), the methods for adjusting our testing (e.g. Bonferonni correction) may drive overly conservative alphas. However, our p-values are already near zero, so we can apply a correction anyway and see if it impacts any conclusions.

The Bonferonni correction requires us to adjust our α by dividing by on the number of tests conducted (28). We can then subtract this adjusted α from 1 to get our new confidence level.

```r
results <- data.frame()
for (combo in combos) {
  cor_test <- cor.test(train[[combo[1]]], train[[combo[2]]], conf.level = 1 - (0.20 / 28))
  result <- list(
    Var1 = combo[1], Var2 = combo[2], p = cor_test$p.value,
    ci_lower = cor_test$conf.int[1], ci_lower = cor_test$conf.int[2]
  )
  results <- rbind(results, result)
}

results
```

```
##              Var1           Var2 p  ci_lower ci_lower.1
## 1             Age         Length 0 0.6066327  0.6189787
## 2             Age       Diameter 0 0.6151483  0.6272890
## 3             Age         Height 0 0.6321689  0.6438910
## 4             Age         Weight 0 0.5948447  0.6074702
## 5             Age Shucked.Weight 0 0.4959021  0.5106649
## 6             Age Viscera.Weight 0 0.5701734  0.5833669
## 7             Age   Shell.Weight 0 0.6579028  0.6689712
## 8          Length       Diameter 0 0.9892276  0.9896431
## 9          Length         Height 0 0.9167891  0.9198861
## 10         Length         Weight 0 0.9351445  0.9375806
## 11         Length Shucked.Weight 0 0.9139014  0.9171013
## 12         Length Viscera.Weight 0 0.9162835  0.9193986
## 13         Length   Shell.Weight 0 0.9153693  0.9185169
## 14       Diameter         Height 0 0.9198456  0.9228334
## 15       Diameter         Weight 0 0.9370544  0.9394209
## 16       Diameter Shucked.Weight 0 0.9125607  0.9158082
## 17       Diameter Viscera.Weight 0 0.9167884  0.9198854
## 18       Diameter   Shell.Weight 0 0.9212052  0.9241444
## 19         Height         Weight 0 0.8999116  0.9036051
## 20         Height Shucked.Weight 0 0.8615565  0.8665660
## 21         Height Viscera.Weight 0 0.8809318  0.8852835
## 22         Height   Shell.Weight 0 0.9015641  0.9051998
## 23         Weight Shucked.Weight 0 0.9707019  0.9718218
## 24         Weight Viscera.Weight 0 0.9704926  0.9716204
## 25         Weight   Shell.Weight 0 0.9648483  0.9661881
## 26 Shucked.Weight Viscera.Weight 0 0.9415138  0.9437176
## 27 Shucked.Weight   Shell.Weight 0 0.9086901  0.9120747
## 28 Viscera.Weight   Shell.Weight 0 0.9326439  0.9351708
```

Even with the correction, our conclusion remains the same — all pairs of variables have non-zero correlation. Our confidence intervals are slightly wider, but in all cases, the correlation appears quite strong (>0.9) for pairs of predictor variables. When paired with our target variable (Age), predictor variables have slightly lower levels of correlation (~0.5 to ~0.7), but still enough to indicate that they may be good predictors.

Linear Algebra and Correlation

Invert your correlation matrix from above. (This is known as the precision matrix and contains variance inflation factors on the diagonal.) Multiply the correlation matrix by the precision matrix, and then multiply the precision matrix by the correlation matrix. Conduct LDU decomposition on the matrix.

I didn’t have an explicit correlation matrix with all variables, so I’ll first generate that. Note that I’ll remove the ID column (since it’s not actually a numerical value, just a unique key), and the Sex column (since its not numerical).

```
cor_matrix <- cor(select(train, -id, -Sex))
cor_matrix
```

```
##                   Length  Diameter    Height    Weight Shucked.Weight
## Length         1.0000000 0.9894374 0.9183517 0.9363738      0.9155158
## Diameter       0.9894374 1.0000000 0.9213531 0.9382486      0.9141991
## Height         0.9183517 0.9213531 1.0000000 0.9017748      0.8640826
## Weight         0.9363738 0.9382486 0.9017748 1.0000000      0.9712672
## Shucked.Weight 0.9155158 0.9141991 0.8640826 0.9712672      1.0000000
## Viscera.Weight 0.9178552 0.9183510 0.8831266 0.9710619      0.9426260
## Shell.Weight   0.9169573 0.9226882 0.9033982 0.9655246      0.9103977
## Age            0.6128431 0.6212559 0.6380669 0.6011950      0.5033202
##                Viscera.Weight Shell.Weight       Age
## Length              0.9178552    0.9169573 0.6128431
## Diameter            0.9183510    0.9226882 0.6212559
## Height              0.8831266    0.9033982 0.6380669
## Weight              0.9710619    0.9655246 0.6011950
## Shucked.Weight      0.9426260    0.9103977 0.5033202
## Viscera.Weight      1.0000000    0.9339190 0.5768078
## Shell.Weight        0.9339190    1.0000000 0.6634733
## Age                 0.5768078    0.6634733 1.0000000
```

Next, we can compute the precision matrix by inverting this correlation matrix. We'll use the `inv` function from `pracma`.

```
precision_matrix <- pracma::inv(cor_matrix)
precision_matrix
```

```
##                     Length    Diameter     Height      Weight Shucked.Weight
## Length          50.0363646 -45.1097381 -1.9464374   0.1100257     -2.9284116
## Diameter       -45.1097381  52.3551380 -2.5463119  -1.4747766     -0.4008903
## Height          -1.9464374  -2.5463119  7.8112768  -0.1561810      0.2354580
## Weight           0.1100257  -1.4747766 -0.1561810  78.0530360    -32.1967124
## Shucked.Weight  -2.9284116  -0.4008903  0.2354580 -32.1967124     25.3018917
## Viscera.Weight  -1.4824486   0.2206767 -0.5969465 -19.0009603      1.5006489
## Shell.Weight     1.5536462  -2.7474053 -2.0985613 -25.7645353      7.8914648
## Age             -0.1657241  -0.4720841 -0.4972981  -1.7174146      2.4136599
##                Viscera.Weight Shell.Weight        Age
## Length             -1.4824486     1.553646 -0.1657241
## Diameter            0.2206767    -2.747405 -0.4720841
## Height             -0.5969465    -2.098561 -0.4972981
## Weight            -19.0009603   -25.764535 -1.7174146
## Shucked.Weight      1.5006489     7.891465  2.4136599
## Viscera.Weight     17.9689552     1.656625  0.3565189
## Shell.Weight        1.6566248    20.997071 -1.2752517
## Age                 0.3565189    -1.275252  2.1702671
```

Finally, we can multiply these two and perform our LDU decomposition. We'll start by calculating the L and U components with `lu.decomposition` function from `matrixcalc`. We can then pull out the diagonals from the U component to form our D component. With that, we need to then adjust our U component and scale its diagonal down to 1 by dividing by the diagonal of D.

```
product_matrix <- cor_matrix %*% precision_matrix

lu_decomp <- matrixcalc::lu.decomposition(product_matrix)

L <- lu_decomp$L
U <- lu_decomp$U
D <- diag(diag(U))
U <- U / diag(D)

cat('L component:\n')
```

```
## L component:
```

```
print(L)
```

```
##               [,1]          [,2]          [,3]          [,4]          [,5]
## [1,]  1.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
## [2,]  8.743006e-16  1.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
## [3,] -1.373901e-15 -4.607426e-15  1.000000e+00  0.000000e+00  0.000000e+00
## [4,] -6.702972e-15 -5.995204e-15 -2.775558e-16  1.000000e+00  0.000000e+00
## [5,] -2.178813e-15  1.970646e-15 -1.221245e-15 -1.687539e-14  1.000000e+00
## [6,] -8.187895e-15  1.554312e-15 -1.387779e-15 -2.009504e-14 -4.218847e-15
## [7,]  4.163336e-17 -1.998401e-15 -6.661338e-16 -1.243450e-14 -3.552714e-15
## [8,]  9.159340e-16 -4.773959e-15 -7.216450e-16 -7.993606e-15 -6.661338e-15
##               [,6]         [,7] [,8]
## [1,]  0.000000e+00 0.000000e+00    0
## [2,]  0.000000e+00 0.000000e+00    0
## [3,]  0.000000e+00 0.000000e+00    0
## [4,]  0.000000e+00 0.000000e+00    0
## [5,]  0.000000e+00 0.000000e+00    0
## [6,]  1.000000e+00 0.000000e+00    0
## [7,] -3.080869e-15 1.000000e+00    0
## [8,] -1.776357e-15 7.105427e-15    1
```

```
cat('D component:\n')
```

```
## D component:
```

```
print(D)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    1    0    0    0    0    0    0    0
## [2,]    0    1    0    0    0    0    0    0
## [3,]    0    0    1    0    0    0    0    0
## [4,]    0    0    0    1    0    0    0    0
## [5,]    0    0    0    0    1    0    0    0
## [6,]    0    0    0    0    0    1    0    0
## [7,]    0    0    0    0    0    0    1    0
## [8,]    0    0    0    0    0    0    0    1
```

```
cat('U component:\n')
```

```
## U component:
```

```
print(U)
```

```
##      [,1]          [,2]          [,3]          [,4]          [,5]          [,6]
## [1,]    1 -2.664535e-15  2.775558e-16 -3.552714e-15 -3.996803e-15 -5.273559e-16
## [2,]    0  1.000000e+00 -1.054712e-15 -1.443290e-14 -1.065814e-14 -3.358425e-15
## [3,]    0  0.000000e+00  1.000000e+00 -1.088019e-14 -5.329071e-15  2.220446e-16
## [4,]    0  0.000000e+00  0.000000e+00  1.000000e+00 -1.776357e-15  3.608225e-16
## [5,]    0  0.000000e+00  0.000000e+00  0.000000e+00  1.000000e+00 -1.415534e-15
## [6,]    0  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  1.000000e+00
## [7,]    0  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
## [8,]    0  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
##              [,7]          [,8]
## [1,] 5.107026e-15  2.220446e-16
## [2,] 7.549517e-15 -1.941337e-31
## [3,] 1.021405e-14 -2.220446e-16
## [4,] 3.330669e-15  2.220446e-16
## [5,] 4.107825e-15  4.440892e-16
## [6,] 3.330669e-15  4.440892e-16
## [7,] 1.000000e+00  4.440892e-16
## [8,] 0.000000e+00  1.000000e+00
```

We can check our decomposition by multiplying these three component matrixes and confirming that their product matches the original `product_matrix`.

```
reformed_matrix <- L %*% D %*% U

cat('Original Product Matrix:\n')
```

```
## Original Product Matrix:
```

```
print(matrix(scales::scientific(product_matrix), nrow = 8))
```

```
##      [,1]        [,2]        [,3]        [,4]        [,5]        [,6]       
## [1,] "1.00e+00"  "-2.66e-15" "2.78e-16"  "-3.55e-15" "-4.00e-15" "-5.27e-16"
## [2,] "8.74e-16"  "1.00e+00"  "-1.05e-15" "-1.44e-14" "-1.07e-14" "-3.36e-15"
## [3,] "-1.37e-15" "-4.61e-15" "1.00e+00"  "-1.09e-14" "-5.33e-15" "2.22e-16" 
## [4,] "-6.70e-15" "-6.00e-15" "-2.78e-16" "1.00e+00"  "-1.78e-15" "3.61e-16" 
## [5,] "-2.18e-15" "1.97e-15"  "-1.22e-15" "-1.69e-14" "1.00e+00"  "-1.42e-15"
## [6,] "-8.19e-15" "1.55e-15"  "-1.39e-15" "-2.01e-14" "-4.22e-15" "1.00e+00" 
## [7,] "4.16e-17"  "-2.00e-15" "-6.66e-16" "-1.24e-14" "-3.55e-15" "-3.08e-15"
## [8,] "9.16e-16"  "-4.77e-15" "-7.22e-16" "-7.99e-15" "-6.66e-15" "-1.78e-15"
##      [,7]       [,8]       
## [1,] "5.11e-15" "2.22e-16" 
## [2,] "7.55e-15" "0.00e+00" 
## [3,] "1.02e-14" "-2.22e-16"
## [4,] "3.33e-15" "2.22e-16" 
## [5,] "4.11e-15" "4.44e-16" 
## [6,] "3.33e-15" "4.44e-16" 
## [7,] "1.00e+00" "4.44e-16" 
## [8,] "7.11e-15" "1.00e+00" 
```

```
cat('Reformed Matrix from LDU:\n')
```

```
## Reformed Matrix from LDU:
```

```
print(matrix(scales::scientific(reformed_matrix), nrow = 8))
```

```
##      [,1]        [,2]        [,3]        [,4]        [,5]        [,6]       
## [1,] "1.00e+00"  "-2.66e-15" "2.78e-16"  "-3.55e-15" "-4.00e-15" "-5.27e-16"
## [2,] "8.74e-16"  "1.00e+00"  "-1.05e-15" "-1.44e-14" "-1.07e-14" "-3.36e-15"
## [3,] "-1.37e-15" "-4.61e-15" "1.00e+00"  "-1.09e-14" "-5.33e-15" "2.22e-16" 
## [4,] "-6.70e-15" "-6.00e-15" "-2.78e-16" "1.00e+00"  "-1.78e-15" "3.61e-16" 
## [5,] "-2.18e-15" "1.97e-15"  "-1.22e-15" "-1.69e-14" "1.00e+00"  "-1.42e-15"
## [6,] "-8.19e-15" "1.55e-15"  "-1.39e-15" "-2.01e-14" "-4.22e-15" "1.00e+00" 
## [7,] "4.16e-17"  "-2.00e-15" "-6.66e-16" "-1.24e-14" "-3.55e-15" "-3.08e-15"
## [8,] "9.16e-16"  "-4.77e-15" "-7.22e-16" "-7.99e-15" "-6.66e-15" "-1.78e-15"
##      [,7]       [,8]       
## [1,] "5.11e-15" "2.22e-16" 
## [2,] "7.55e-15" "0.00e+00" 
## [3,] "1.02e-14" "-2.22e-16"
## [4,] "3.33e-15" "2.22e-16" 
## [5,] "4.11e-15" "4.44e-16" 
## [6,] "3.33e-15" "4.44e-16" 
## [7,] "1.00e+00" "4.44e-16" 
## [8,] "7.11e-15" "1.00e+00" 
```

```
cat('Match Check:',all.equal(round(matrix(product_matrix,nrow = 8),20),round(reformed_matrix,2
0)))
```

```
## Match Check: TRUE
```

Our reconstruction was successful! Note that the value in `[7,8]` is actually zero in both matrices, but appears as -3.50e-46 in the reconstruction due to floating point precision.

One thing to note here is that our initial `product_matrix` is symmetrical and has unit diagonals (i.e. 1s along the diagonal). As a result, the U component is simply the transpose of L. Moreover, the D component is simply an identity matrix. We can demonstrate this with an alternative deconstruction.

```
lu_decomp <- matrixcalc::lu.decomposition(product_matrix)

L_alt <- lu_decomp$L
U_alt <- t(L_alt)
D_alt <- diag(8)

cat('L component:\n')
```

```
## L component:
```

```
print(L_alt)
```

```
##               [,1]          [,2]          [,3]          [,4]          [,5]
## [1,]  1.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
## [2,]  8.743006e-16  1.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
## [3,] -1.373901e-15 -4.607426e-15  1.000000e+00  0.000000e+00  0.000000e+00
## [4,] -6.702972e-15 -5.995204e-15 -2.775558e-16  1.000000e+00  0.000000e+00
## [5,] -2.178813e-15  1.970646e-15 -1.221245e-15 -1.687539e-14  1.000000e+00
## [6,] -8.187895e-15  1.554312e-15 -1.387779e-15 -2.009504e-14 -4.218847e-15
## [7,]  4.163336e-17 -1.998401e-15 -6.661338e-16 -1.243450e-14 -3.552714e-15
## [8,]  9.159340e-16 -4.773959e-15 -7.216450e-16 -7.993606e-15 -6.661338e-15
##               [,6]         [,7] [,8]
## [1,]  0.000000e+00 0.000000e+00    0
## [2,]  0.000000e+00 0.000000e+00    0
## [3,]  0.000000e+00 0.000000e+00    0
## [4,]  0.000000e+00 0.000000e+00    0
## [5,]  0.000000e+00 0.000000e+00    0
## [6,]  1.000000e+00 0.000000e+00    0
## [7,] -3.080869e-15 1.000000e+00    0
## [8,] -1.776357e-15 7.105427e-15    1
```

```
cat('D component:\n')
```

```
## D component:
```

```
print(U_alt)
```

```
##      [,1]         [,2]          [,3]          [,4]          [,5]          [,6]
## [1,]    1 8.743006e-16 -1.373901e-15 -6.702972e-15 -2.178813e-15 -8.187895e-15
## [2,]    0 1.000000e+00 -4.607426e-15 -5.995204e-15  1.970646e-15  1.554312e-15
## [3,]    0 0.000000e+00  1.000000e+00 -2.775558e-16 -1.221245e-15 -1.387779e-15
## [4,]    0 0.000000e+00  0.000000e+00  1.000000e+00 -1.687539e-14 -2.009504e-14
## [5,]    0 0.000000e+00  0.000000e+00  0.000000e+00  1.000000e+00 -4.218847e-15
## [6,]    0 0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  1.000000e+00
## [7,]    0 0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
## [8,]    0 0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
##               [,7]          [,8]
## [1,]  4.163336e-17  9.159340e-16
## [2,] -1.998401e-15 -4.773959e-15
## [3,] -6.661338e-16 -7.216450e-16
## [4,] -1.243450e-14 -7.993606e-15
## [5,] -3.552714e-15 -6.661338e-15
## [6,] -3.080869e-15 -1.776357e-15
## [7,]  1.000000e+00  7.105427e-15
## [8,]  0.000000e+00  1.000000e+00
```

```
cat('U component:\n')
```

```
## U component:
```

```
print(D_alt)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    1    0    0    0    0    0    0    0
## [2,]    0    1    0    0    0    0    0    0
## [3,]    0    0    1    0    0    0    0    0
## [4,]    0    0    0    1    0    0    0    0
## [5,]    0    0    0    0    1    0    0    0
## [6,]    0    0    0    0    0    1    0    0
## [7,]    0    0    0    0    0    0    1    0
## [8,]    0    0    0    0    0    0    0    1
```

```
reformed_matrix_alt <- L %*% D %*% U

cat('Original Product Matrix:\n')
```

```
## Original Product Matrix:
```

```
print(matrix(scales::scientific(product_matrix), nrow = 8))
```

```
##      [,1]        [,2]        [,3]        [,4]        [,5]        [,6]       
## [1,] "1.00e+00"  "-2.66e-15" "2.78e-16"  "-3.55e-15" "-4.00e-15" "-5.27e-16"
## [2,] "8.74e-16"  "1.00e+00"  "-1.05e-15" "-1.44e-14" "-1.07e-14" "-3.36e-15"
## [3,] "-1.37e-15" "-4.61e-15" "1.00e+00"  "-1.09e-14" "-5.33e-15" "2.22e-16" 
## [4,] "-6.70e-15" "-6.00e-15" "-2.78e-16" "1.00e+00"  "-1.78e-15" "3.61e-16" 
## [5,] "-2.18e-15" "1.97e-15"  "-1.22e-15" "-1.69e-14" "1.00e+00"  "-1.42e-15"
## [6,] "-8.19e-15" "1.55e-15"  "-1.39e-15" "-2.01e-14" "-4.22e-15" "1.00e+00" 
## [7,] "4.16e-17"  "-2.00e-15" "-6.66e-16" "-1.24e-14" "-3.55e-15" "-3.08e-15"
## [8,] "9.16e-16"  "-4.77e-15" "-7.22e-16" "-7.99e-15" "-6.66e-15" "-1.78e-15"
##      [,7]       [,8]       
## [1,] "5.11e-15" "2.22e-16" 
## [2,] "7.55e-15" "0.00e+00" 
## [3,] "1.02e-14" "-2.22e-16"
## [4,] "3.33e-15" "2.22e-16" 
## [5,] "4.11e-15" "4.44e-16" 
## [6,] "3.33e-15" "4.44e-16" 
## [7,] "1.00e+00" "4.44e-16" 
## [8,] "7.11e-15" "1.00e+00" 
```

```
cat('Reformed Matrix from LDU:\n')
```

```
## Reformed Matrix from LDU:
```

```
print(matrix(scales::scientific(reformed_matrix_alt), nrow = 8))
```

```
##      [,1]        [,2]        [,3]        [,4]        [,5]        [,6]       
## [1,] "1.00e+00"  "-2.66e-15" "2.78e-16"  "-3.55e-15" "-4.00e-15" "-5.27e-16"
## [2,] "8.74e-16"  "1.00e+00"  "-1.05e-15" "-1.44e-14" "-1.07e-14" "-3.36e-15"
## [3,] "-1.37e-15" "-4.61e-15" "1.00e+00"  "-1.09e-14" "-5.33e-15" "2.22e-16" 
## [4,] "-6.70e-15" "-6.00e-15" "-2.78e-16" "1.00e+00"  "-1.78e-15" "3.61e-16" 
## [5,] "-2.18e-15" "1.97e-15"  "-1.22e-15" "-1.69e-14" "1.00e+00"  "-1.42e-15"
## [6,] "-8.19e-15" "1.55e-15"  "-1.39e-15" "-2.01e-14" "-4.22e-15" "1.00e+00" 
## [7,] "4.16e-17"  "-2.00e-15" "-6.66e-16" "-1.24e-14" "-3.55e-15" "-3.08e-15"
## [8,] "9.16e-16"  "-4.77e-15" "-7.22e-16" "-7.99e-15" "-6.66e-15" "-1.78e-15"
##      [,7]       [,8]       
## [1,] "5.11e-15" "2.22e-16" 
## [2,] "7.55e-15" "0.00e+00" 
## [3,] "1.02e-14" "-2.22e-16"
## [4,] "3.33e-15" "2.22e-16" 
## [5,] "4.11e-15" "4.44e-16" 
## [6,] "3.33e-15" "4.44e-16" 
## [7,] "1.00e+00" "4.44e-16" 
## [8,] "7.11e-15" "1.00e+00" 
```

```
cat('Match Check:',all.equal(round(matrix(product_matrix,nrow = 8),20),round(reformed_matrix_al
t,20)))
```

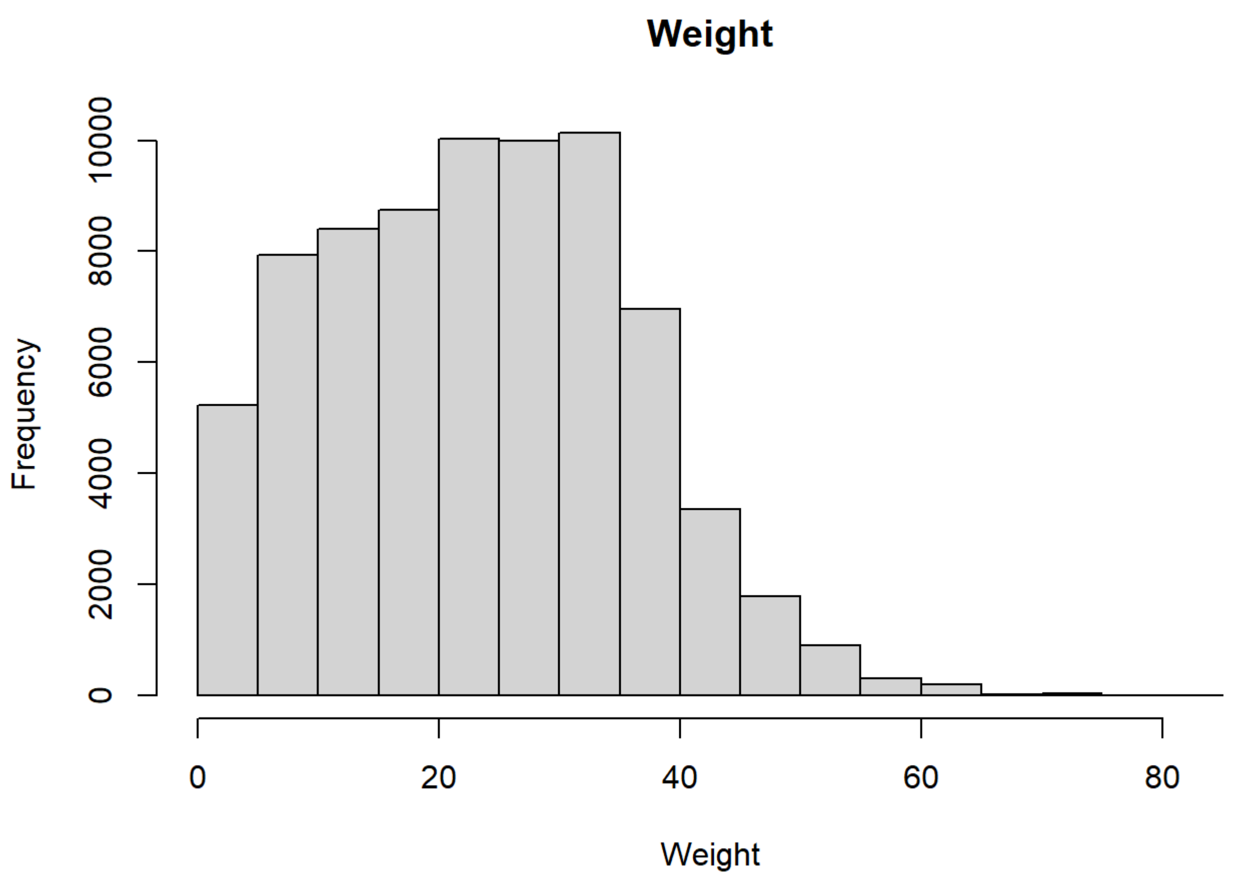
```
## Match Check: TRUE
```

Calculus-Based Probability & Statistics

Many times, it makes sense to fit a closed form distribution to data. Select a variable in the Kaggle.com training dataset that is skewed to the right, shift it so that the minimum value is absolutely above zero if necessary. Then load the MASS package and run fitdistr to fit an exponential probability density function. Find the optimal value of λ for this distribution, and then take 1000 samples from this exponential distribution using this value (e.g., rexp(1000, λ)). Plot a histogram and compare it with a histogram of your original variable. Using the exponential pdf, find the 5th and 95th percentiles using the cumulative distribution function (CDF). Also generate a 95% confidence interval from the empirical data, assuming normality. Finally, provide the empirical 5th percentile and 95th percentile of the data. Discuss.

We'll use the `Weight` variable for this exercise, as it demonstrates right skew.

```
hist(train$Weight, main = 'Weight', xlab = 'Weight')
```



A shift is not needed here, as the minimum value for Weight is already greater than 0.

```
min(train$Weight)
```

```
## [1] 0.056699
```

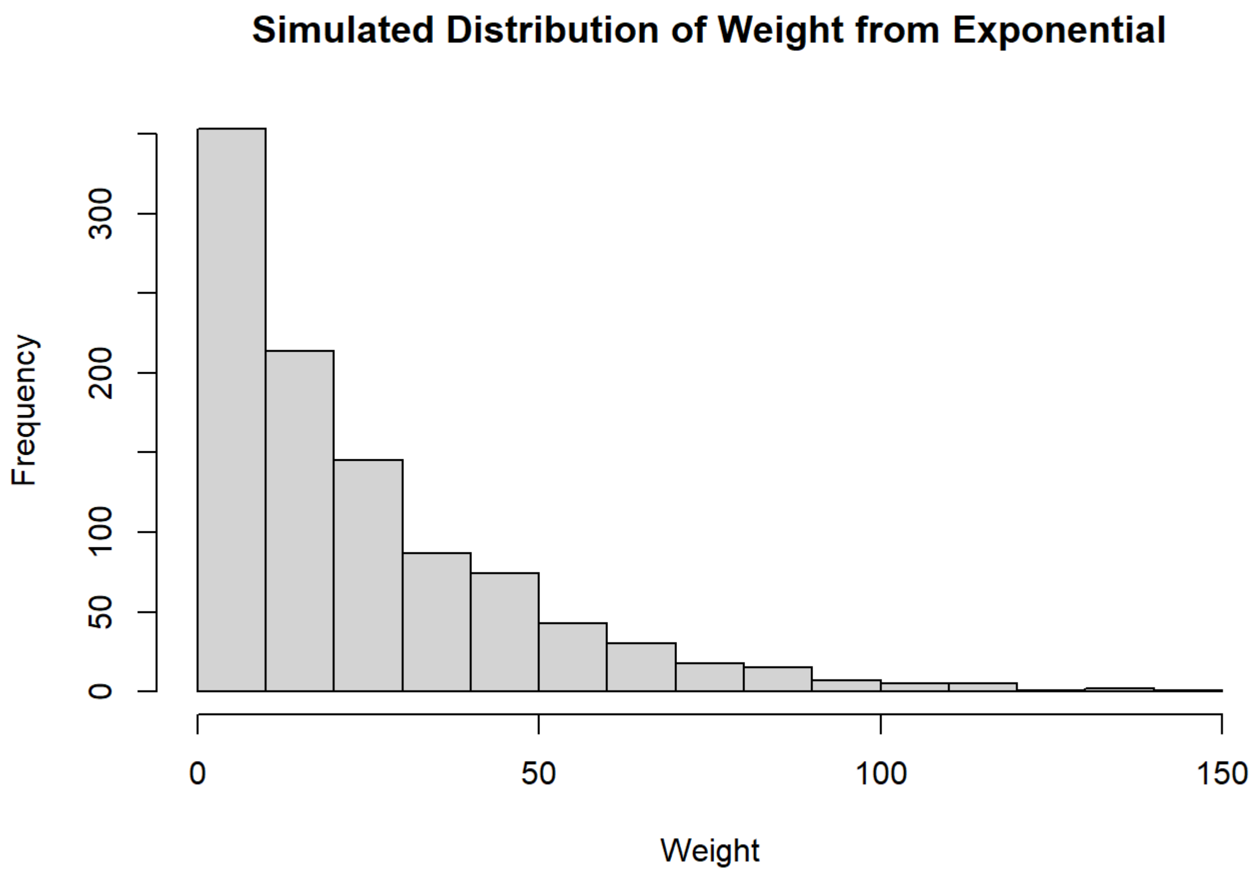
Now, we can use the `fitdistr` function from `MASS` to fit an exponential distribution. This function provides an estimate for the optimal parameter of whichever distribution you choose, which this case is λ for an exponential.

```
weight_exp <- MASS::fitdistr(train$Weight, "exponential")
cat('Estimated Optimal Lambda:', weight_exp$estimate)
```

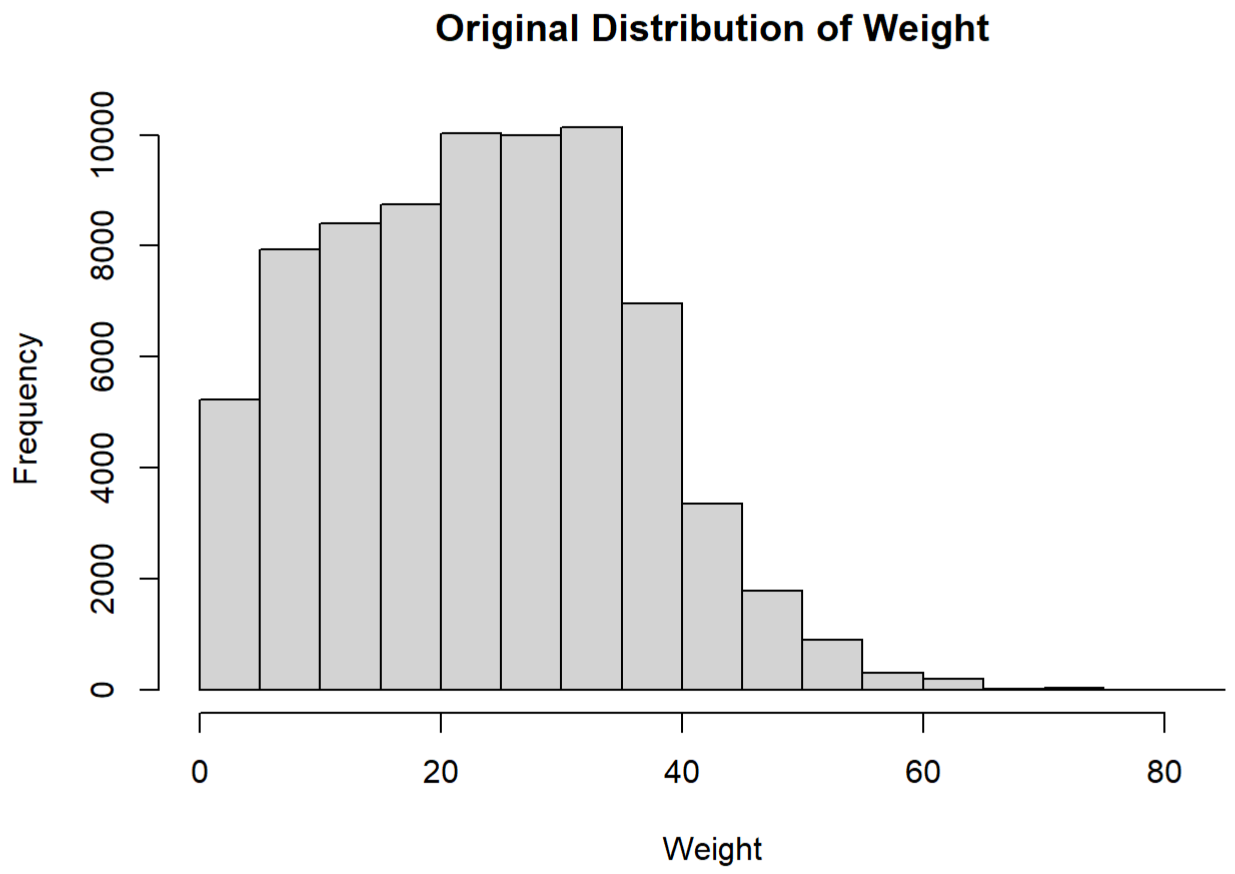
```
## Estimated Optimal Lambda: 0.04276206
```

Let's sample from this distribution and compare the resulting plot to the original variable.

```
weight_exp_sample <- rexp(1000, weight_exp$estimate)

hist(weight_exp_sample, main = 'Simulated Distribution of Weight from Exponential', xlab = 'Weig
ht')
```



```
hist(train$Weight, main = 'Original Distribution of Weight', xlab = 'Weight')
```



Our simulated distribution appears to be very rough approximation of the actual `Weight` variable. It exhibits the same right skew, as expected, but it also produced much greater tail values. The max in our distribution is greater than 150, but the true max weight is closer to 80.

Let’s compare the 5th and 95th quantiles from each distribution, along with a 95% confidence interval. For the confidence interval, we’ll use the following formula:

$$\bar{x} \pm (\frac{s}{\sqrt{n}} \times zscore)$$

The mean and standard deviation of our sample will serve as $\bar{x}$ and s, respectively; our sample size will serve as n; and we can generate a z-score with the `qnorm` function (which comes out to ~1.96 for a 95% CI).

```
exp_5th <- qexp(0.05, rate = weight_exp$estimate)
exp_95th <- qexp(0.95, rate = weight_exp$estimate)

emp_5th <- quantile(train$Weight, 0.05)
emp_95th <- quantile(train$Weight, 0.95)

ci <- mean(train$Weight) + c(-1, 1) * sd(train$Weight) / sqrt(length(train$Weight)) * qnorm(0.97
5)

cat(
  'Exponential 5th percentile:', exp_5th,
  '\nExponential 95th percentile:', exp_95th,
  '\nEmpirical 5th percentile:', emp_5th,
  '\nEmpirical 95th percentile:', emp_95th,
  '\n95% CI:', ci,
  '\nSimulated distribution mean:',mean(weight_exp_sample)
)
```

```
## Exponential 5th percentile: 1.199505
## Exponential 95th percentile: 70.05585
## Empirical 5th percentile: 3.600386
## Empirical 95th percentile: 44.21105
## 95% CI: 23.29412 23.47632
## Simulated distribution mean: 23.86306
```

As with the plots above, we see significant differences between our empirical and parametric distributions. The 5th and 95th percentiles of the parametric distribution are lower and higher than the observed 5th and 95th percentiles. The mean of our simulated distribution also sits outside of the 95% confidence interval for the true population mean of `Weight` (although it is quite close).

Modeling

Build some type of multiple regression model and submit your model to the competition board. Provide your complete model summary and results with analysis. Report your Kaggle.com user name and score.

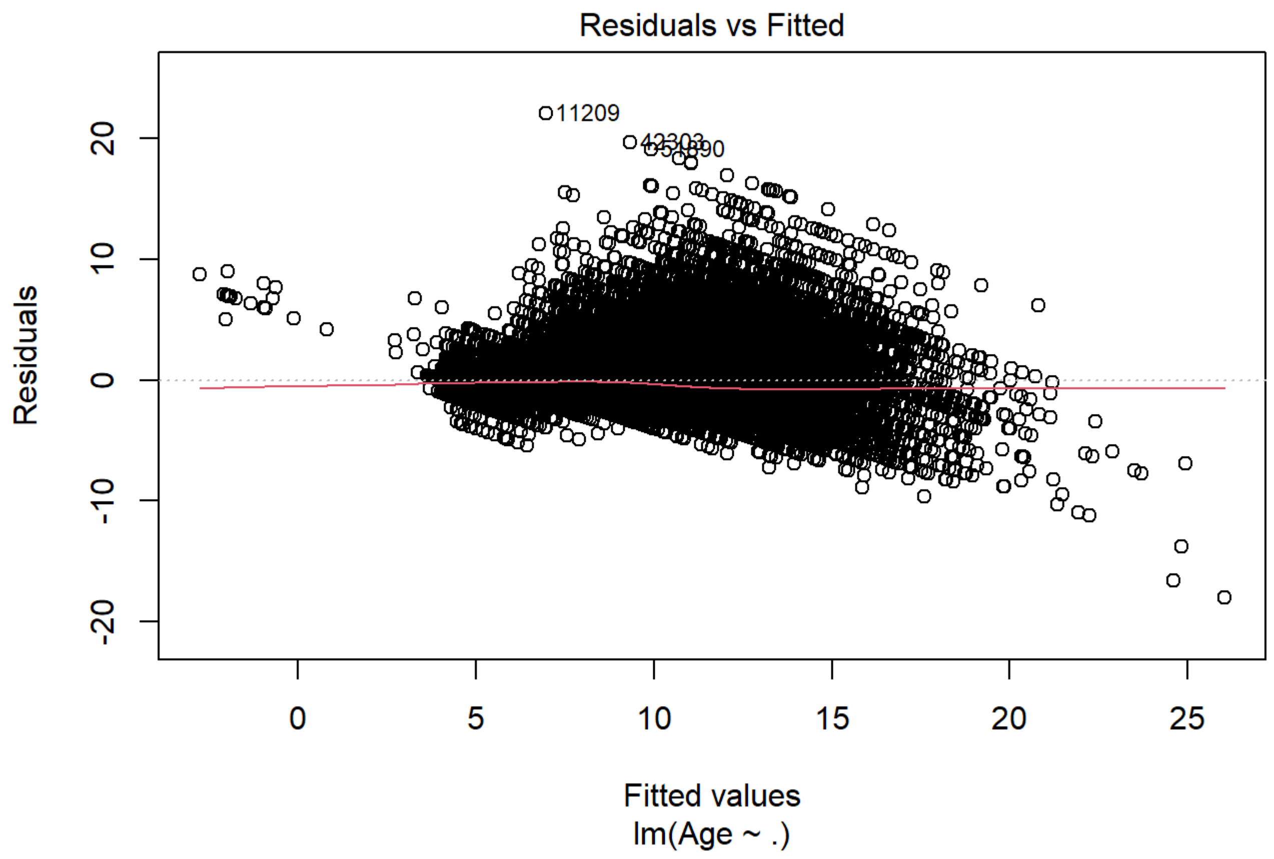
Baseline

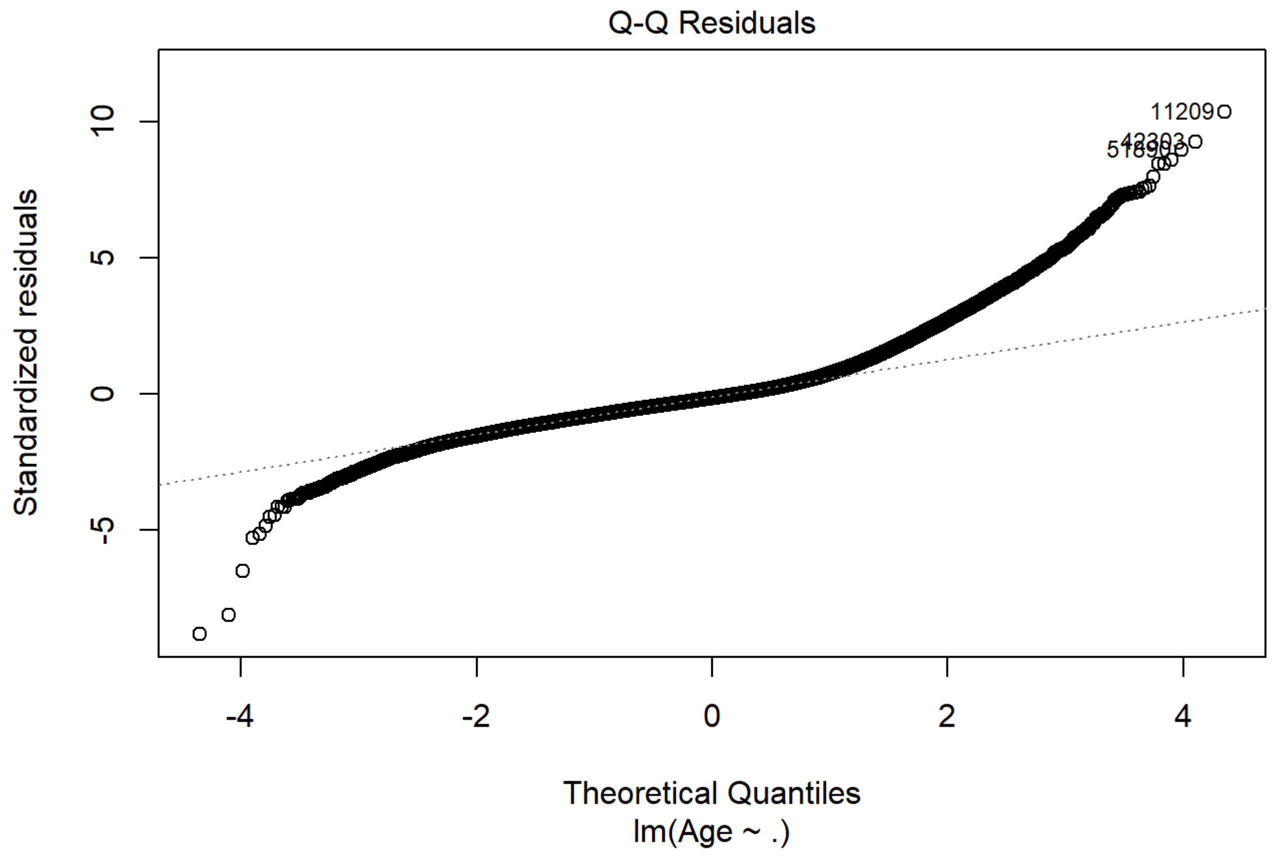
Let’s start with an initial linear fit to serve as a baseline.

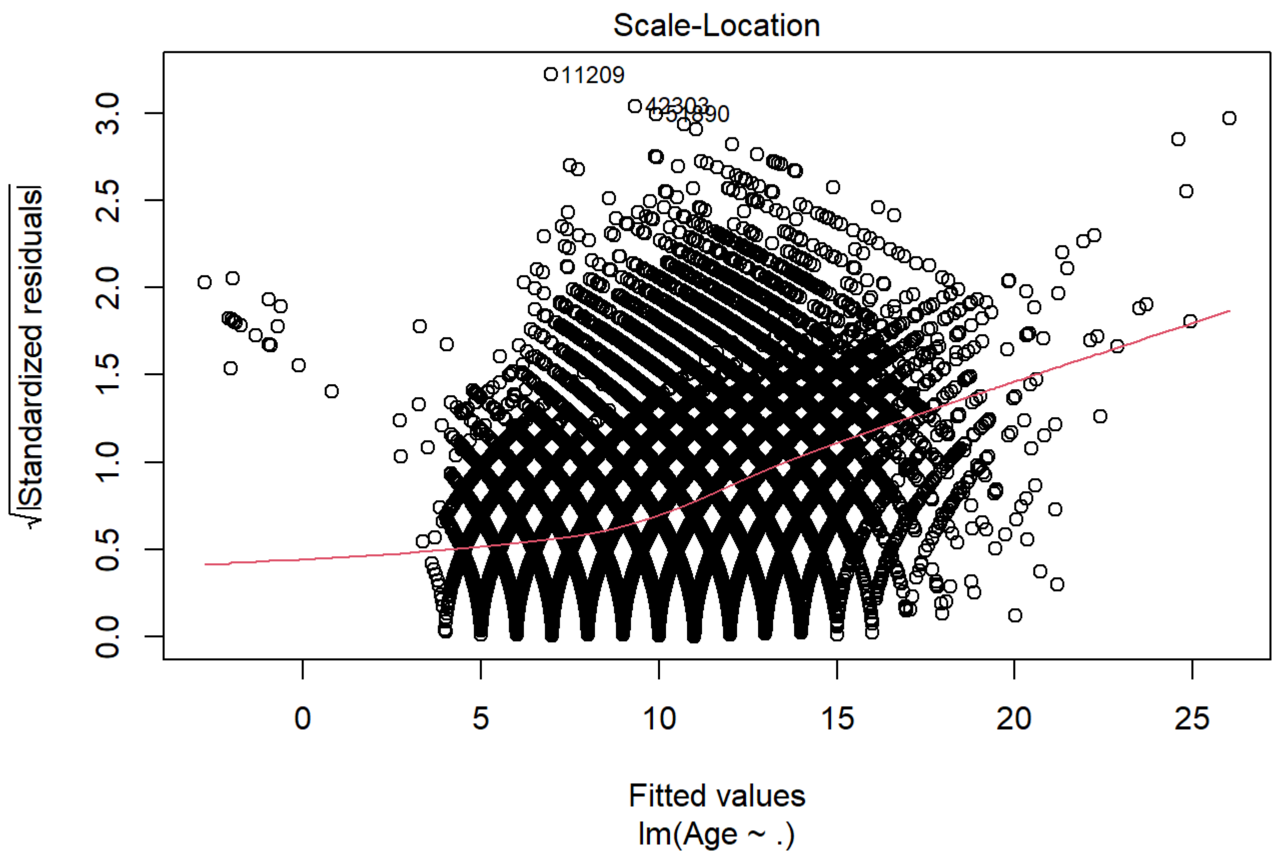
```
lm_base <- lm(Age ~ ., data = select(train, -id))
summary(lm_base)
```

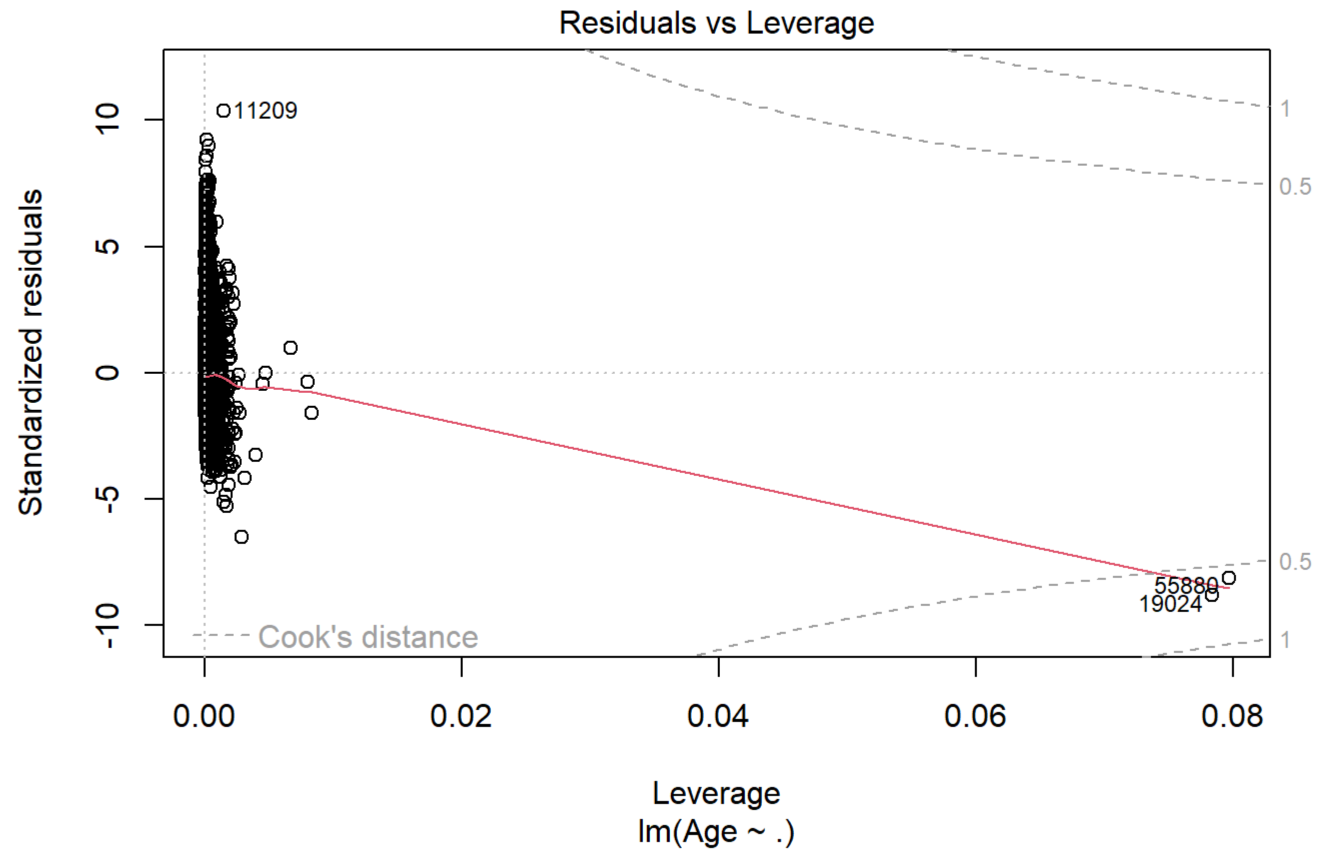
```
##
## Call:
## lm(formula = Age ~ ., data = select(train, -id))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18.0347  -1.2228  -0.3320   0.7537  22.0418
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)     3.755758   0.075897  49.485  < 2e-16 ***
## SexI           -1.040735   0.025209 -41.284  < 2e-16 ***
## SexM           -0.115212   0.019157  -6.014 1.82e-09 ***
## Length          0.910211   0.192234   4.735 2.20e-06 ***
## Diameter        2.138507   0.238786   8.956  < 2e-16 ***
## Height          7.222272   0.236275  30.567  < 2e-16 ***
## Weight          0.194265   0.005416  35.867  < 2e-16 ***
## Shucked.Weight -0.614115   0.006632 -92.603  < 2e-16 ***
## Viscera.Weight -0.216351   0.011872 -18.224  < 2e-16 ***
## Shell.Weight    0.512127   0.009820  52.152  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.128 on 74041 degrees of freedom
## Multiple R-squared:  0.5508, Adjusted R-squared:  0.5508
## F-statistic: 1.009e+04 on 9 and 74041 DF,  p-value: < 2.2e-16
```

```
plot(lm_base)
```


Residuals vs Fitted
Residuals
Fitted values
lm(Age ~ .)
11209
42303
51890
20
10
0
-10
-20
0
5
10
15
20
25


Q-Q Residuals
Standardized residuals
Theoretical Quantiles
lm(Age ~ .)
11209
42303
51890
10
5
0
-5
-4
-2
0
2
4


Scale-Location
√|Standardized residuals|
Fitted values
lm(Age ~ .)
11209
42303
51890
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
5
10
15
20
25


Residuals vs Leverage
Standardized residuals
Leverage
lm(Age ~ .)
11209
55880
19024
Cook's distance
1
0.5
0.5
1
-10
-5
0
5
10
0.00
0.02
0.04
0.06
0.08

These plots highlight some potential issues. The first and third plots indicate heteroskedasticity. We noted before significant skew in most of our variables, which is likely driving inconsistent variance in residuals. We'll consider some transformations in our next iteration of the model.

The fourth indicates two very influential points at indexes 55880 and 19024. If we take a look at these points, nothing appears especially significant. All predictor variables are roughly around the first quartile in terms of their respective distributions. The exceptions are Shucked and Viscera Weight, which are relatively high compared to overall Weight. So, if anything, this observation seems for be for a crab that has relatively high Shucked and Viscera Weight. I don't think this warrants removal at these points, but we'll keep an eye out as we adjust the model.

```
train[c('55880','19024'), ]
```

```
##          id Sex Length Diameter Height  Weight Shucked.Weight Viscera.Weight
## 55880 55879   I 1.1375   0.8875  2.825 16.8396       9.412034       4.068153
## 19024 19023   F 1.2375   0.9500  2.825 16.8396       9.412034       3.288542
##       Shell.Weight Age
## 55880     3.784658   8
## 19024     3.784658   8
```

Previously, we had also identified high correlation among predictors. Let's check the Variance Infaltion Factor to see if the model exhibits high multicollinearity.

```
car::vif(lm_base)
```

```
##                     GVIF Df GVIF^(1/(2*Df))
## Sex             1.815731  2        1.160815
## Length         50.029756  1        7.073172
## Diameter       52.539130  1        7.248388
## Height          7.731224  1        2.780508
## Weight         76.731546  1        8.759654
## Shucked.Weight 22.695250  1        4.763953
## Viscera.Weight 17.972270  1        4.239371
## Shell.Weight   20.256450  1        4.500717
```

Looking at the adjusted VIF (the third column). It seems there is some significant multicollinearity with Length, Diameter and Weight. If this condition persists after transformation, I'll consider feature selection techniques to remove correlated pairs.

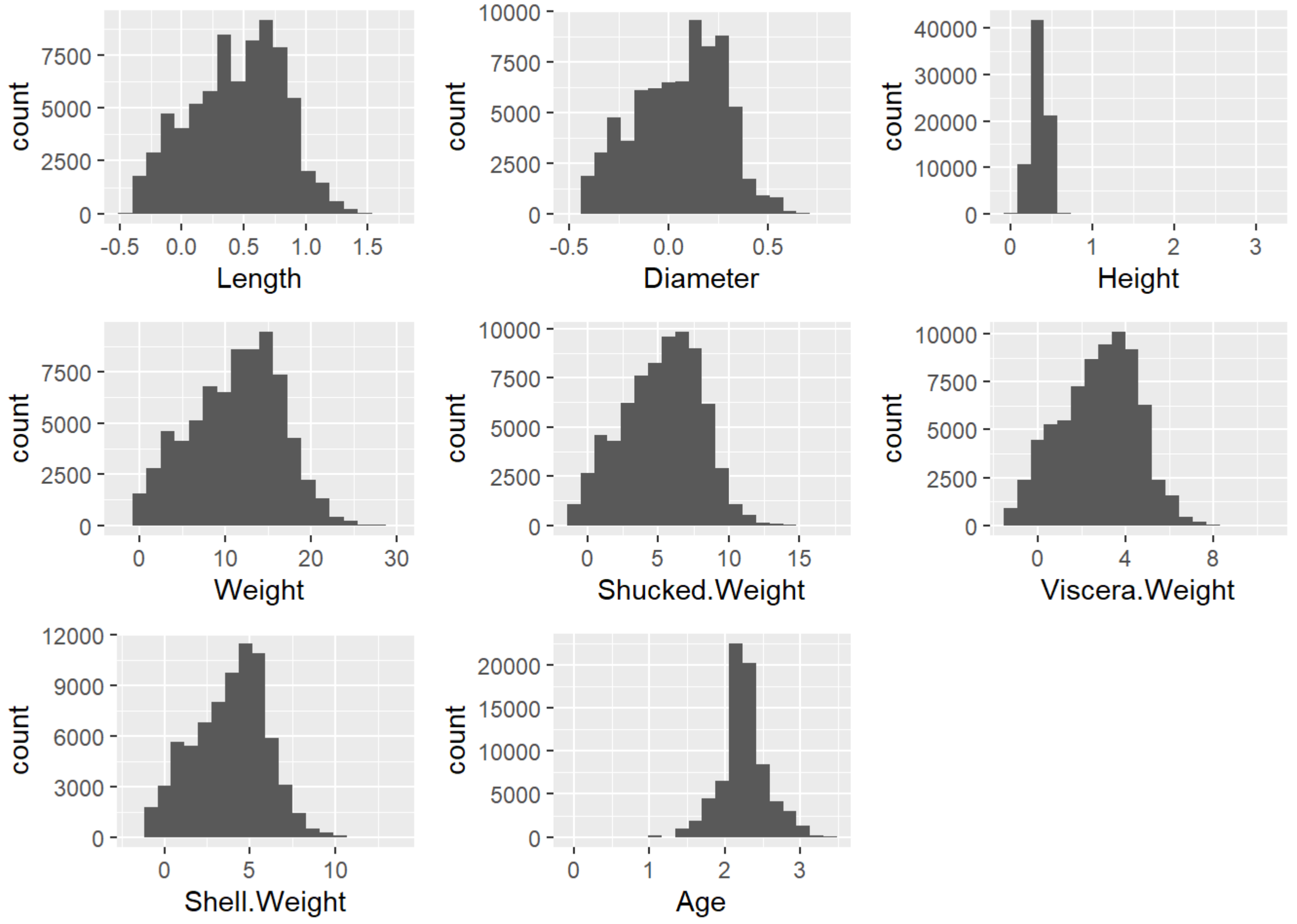
Power Transformation

I'll use the `powerTransform` function from `car` to address the skew in our variables.

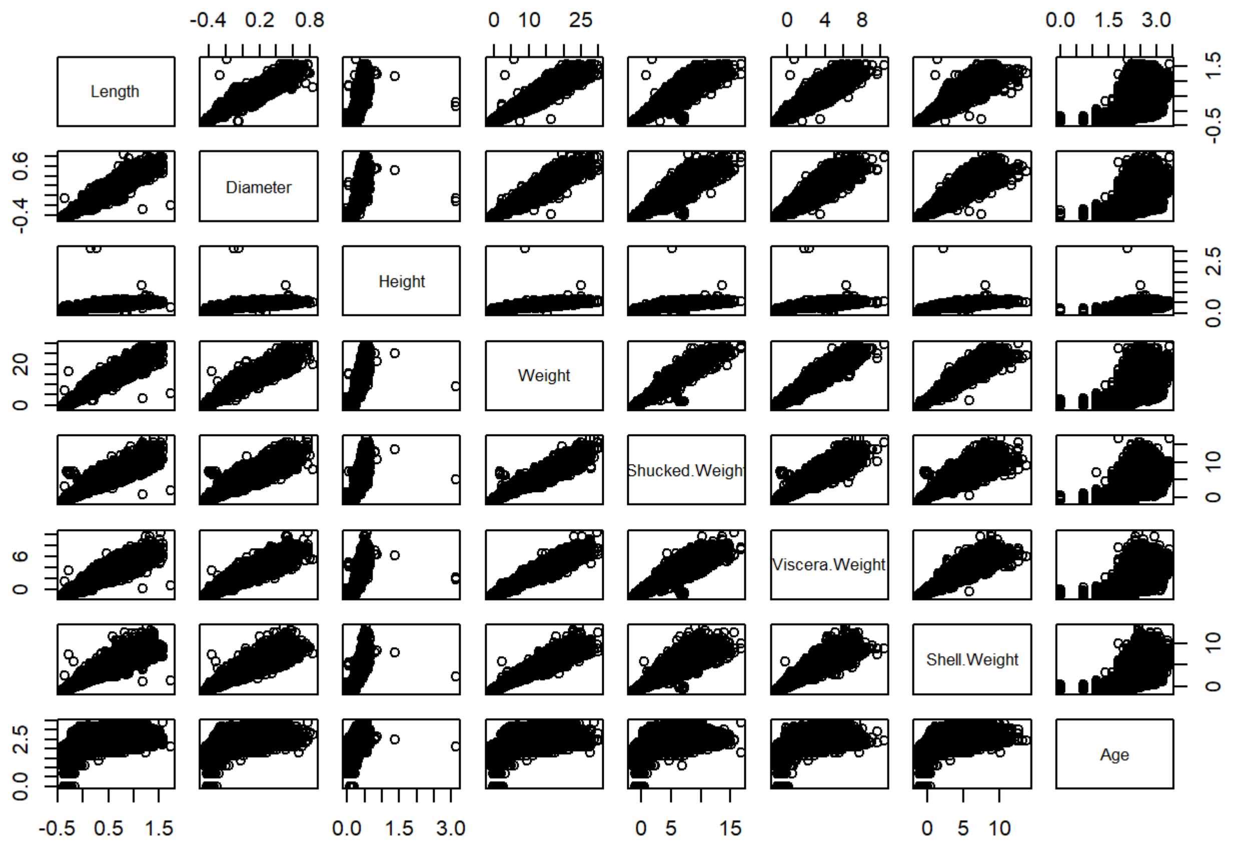
```
train_transform <- train

train_transform[['Height']] <- train_transform[['Height']] + 1

for (predictor in colnames(select(train_transform, -id, -Sex, -Age))) {
  formula <- as.formula(paste(predictor,' ~ 1'))
  transformation <- car::powerTransform(formula, data = train_transform)
  transformed <- car::bcPower(train_transform[predictor], transformation$lambda)
  train_transform[predictor] <- transformed
}

transformation <- car::powerTransform(Age ~ ., data = train_transform)
transformed <- car::bcPower(train_transform['Age'], transformation$lambda)
train_transform['Age'] <- transformed
```

Let’s re-examine these variables now that they’re transformed.

```
histograms <- function(df) {
 plots <- list()

 for (i in 1:ncol(df)) {
 col <- colnames(df)[i]
 p <- df %>%
 ggplot(aes(!!sym(col))) +
 geom_histogram(bins = 20)
 plots[[i]] <- p
 }

 return(cowplot::plot_grid(plotlist = plots, nrow = 3))
}

histograms(select(train_transform, -id, -Sex))
```



```
pairs(select(train_transform, -id, -Sex))
```



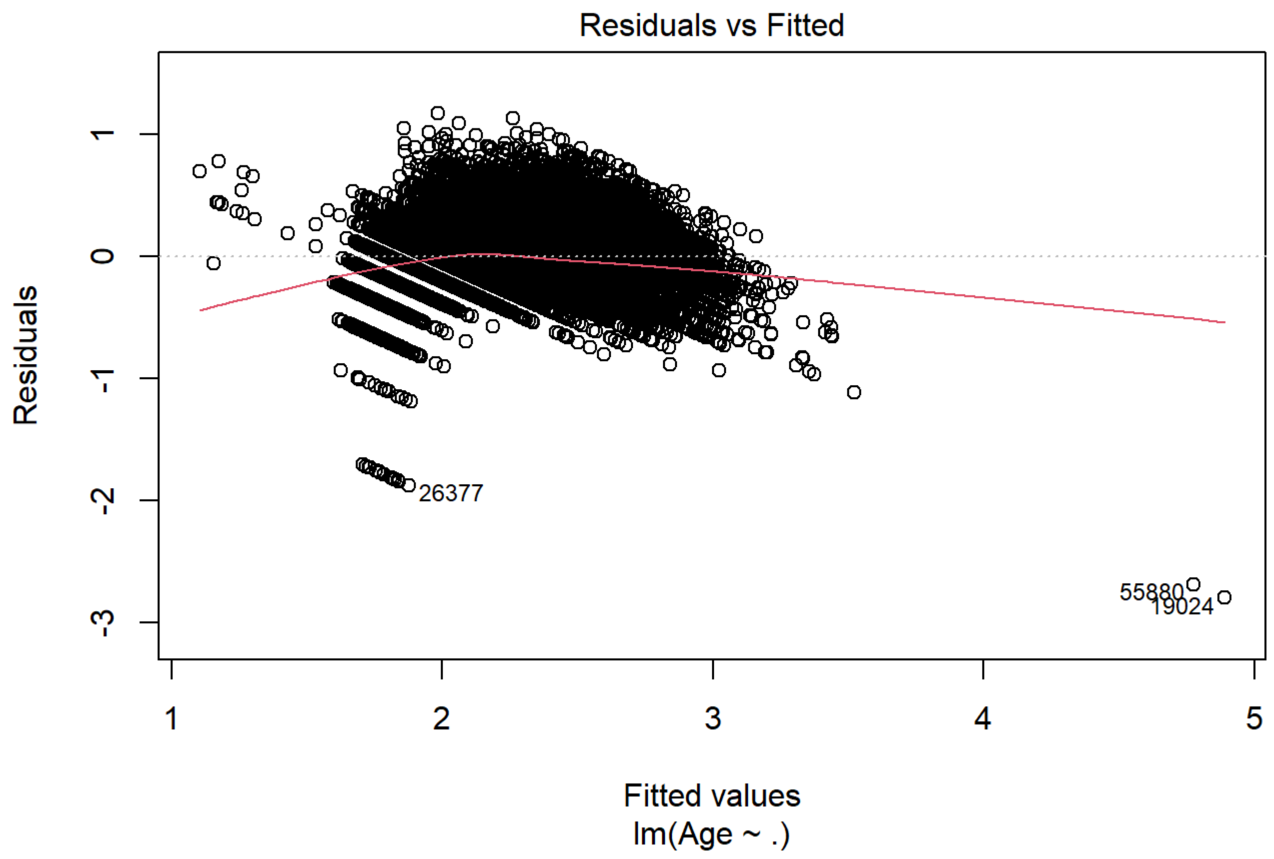
Our distributions appear much more normal, and the linear relationships are much clearer. Multicollinearity persists, but we'll address that shortly.

For now, let's refit the model.

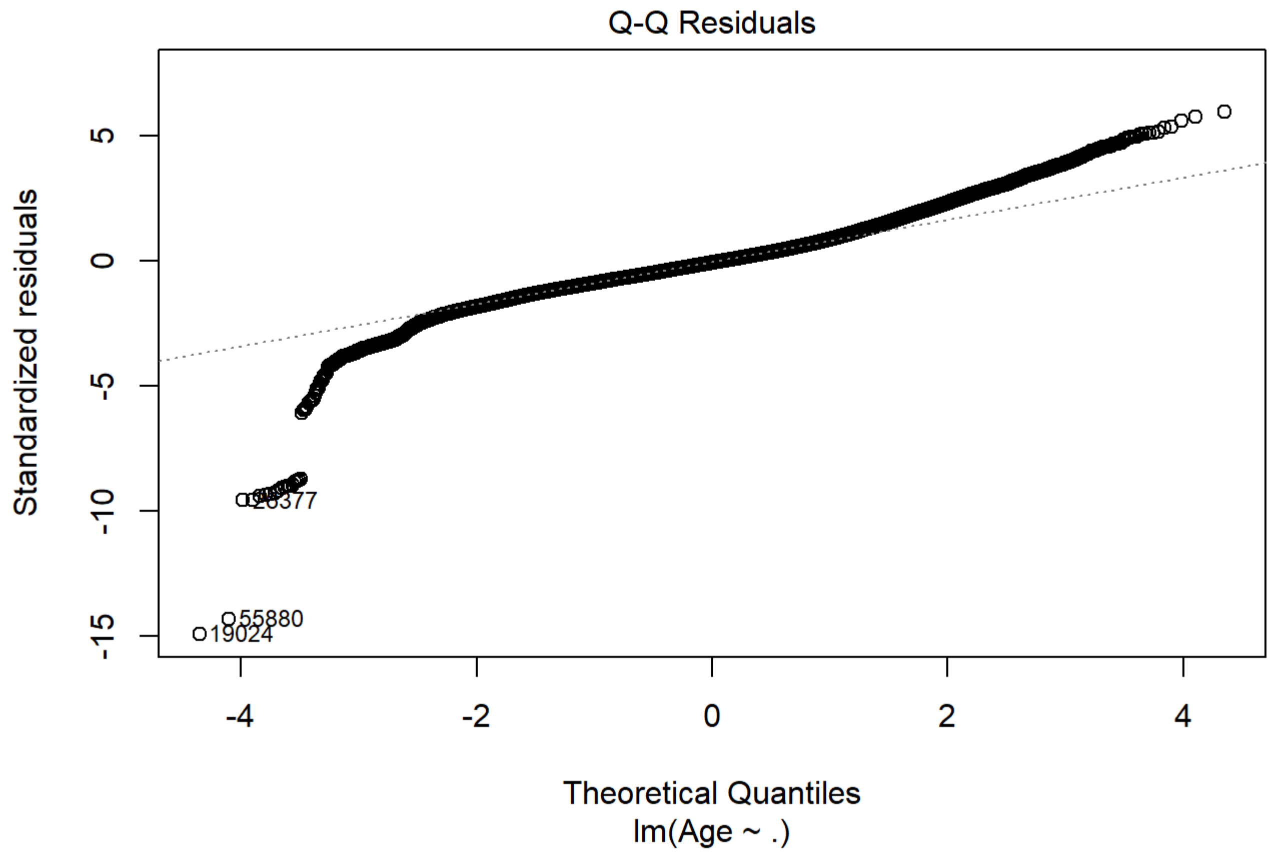
```
lm_pt <- lm(Age ~ ., data = select(train_transform, -id))
summary(lm_pt)
```

```
##
## Call:
## lm(formula = Age ~ ., data = select(train_transform, -id))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.79901 -0.12065 -0.01213  0.10232  1.17416
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)     1.728739   0.008301 208.255  < 2e-16 ***
## SexI           -0.115230   0.002334 -49.380  < 2e-16 ***
## SexM           -0.008225   0.001768  -4.653 3.28e-06 ***
## Length         -0.056057   0.012412  -4.516 6.30e-06 ***
## Diameter       -0.002909   0.020781  -0.140    0.889
## Height          1.012449   0.020462  49.479  < 2e-16 ***
## Weight          0.030603   0.001376  22.237  < 2e-16 ***
## Shucked.Weight -0.093403   0.001420 -65.767  < 2e-16 ***
## Viscera.Weight -0.009426   0.002006  -4.699 2.62e-06 ***
## Shell.Weight    0.106829   0.001745  61.237  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1964 on 74041 degrees of freedom
## Multiple R-squared:  0.6168, Adjusted R-squared:  0.6168
## F-statistic: 1.324e+04 on 9 and 74041 DF,  p-value: < 2.2e-16
```

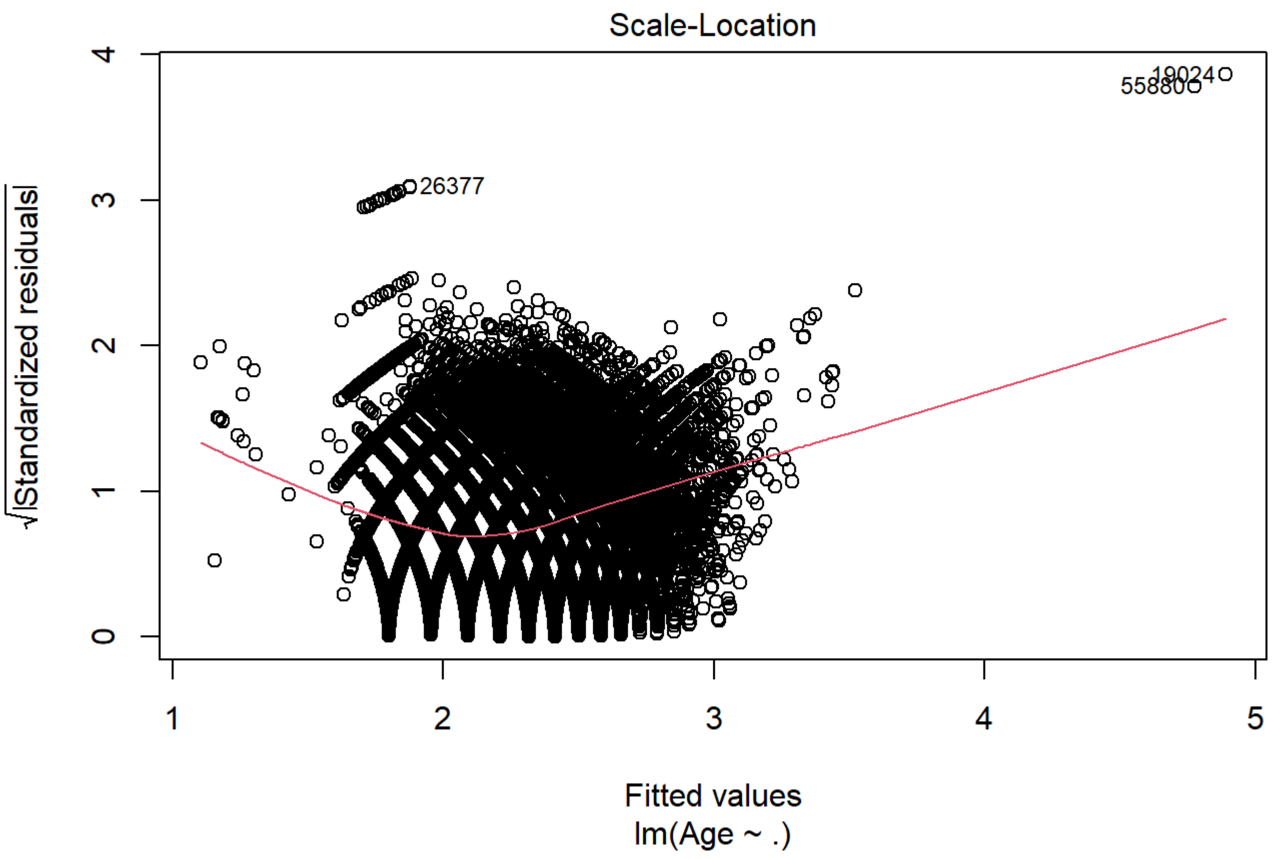
```
plot(lm_pt)
```



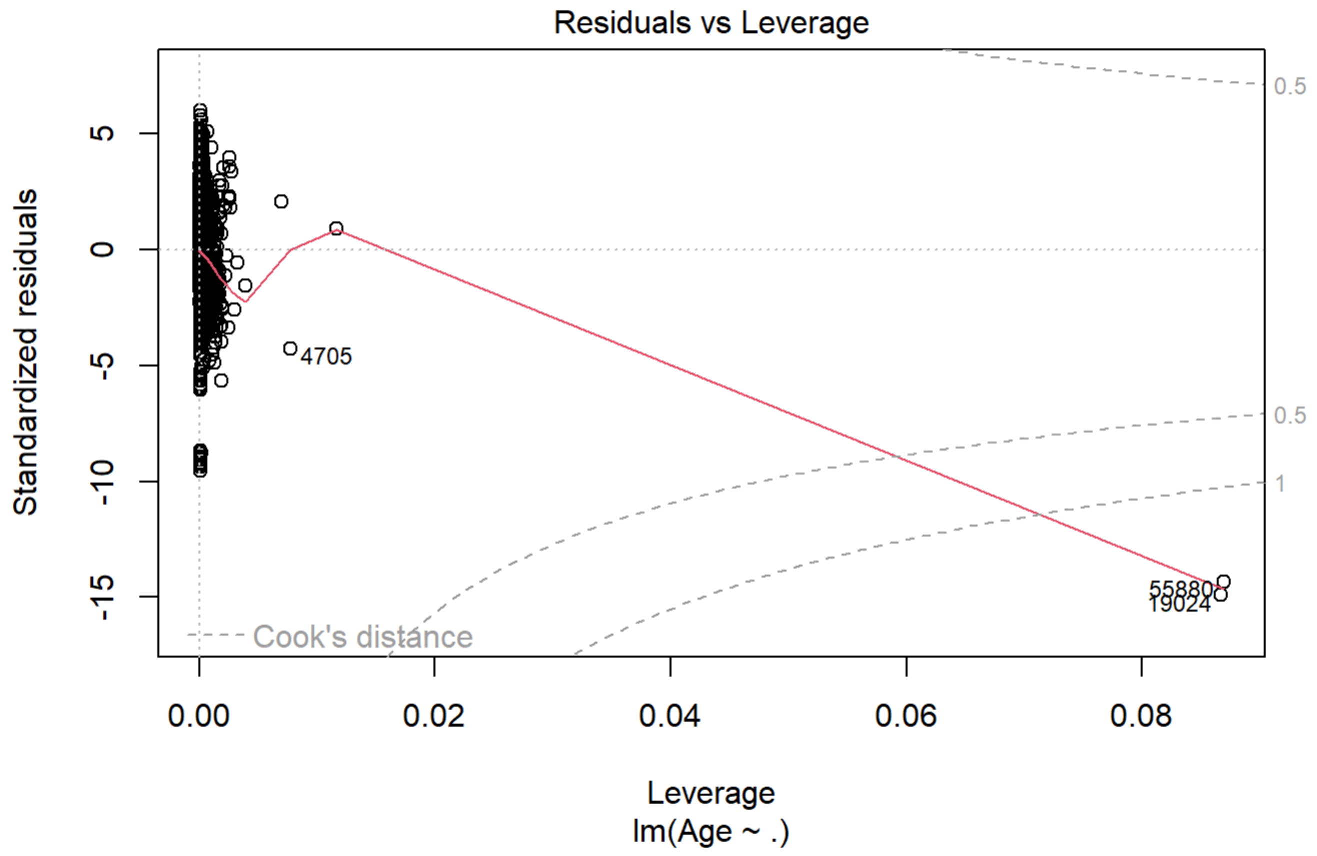
Residuals vs Fitted
Residuals
Fitted values
lm(Age ~ .)
26377
55880
19024



Q-Q Residuals
Standardized residuals
Theoretical Quantiles
lm(Age ~ .)
26377
55880
19024



Scale-Location
√|Standardized residuals|
Fitted values
lm(Age ~ .)
19024
55880
26377



Residuals vs Leverage
Standardized residuals
Leverage
lm(Age ~ .)
Cook's distance
0.5
0.5
1
4705
55880
19024

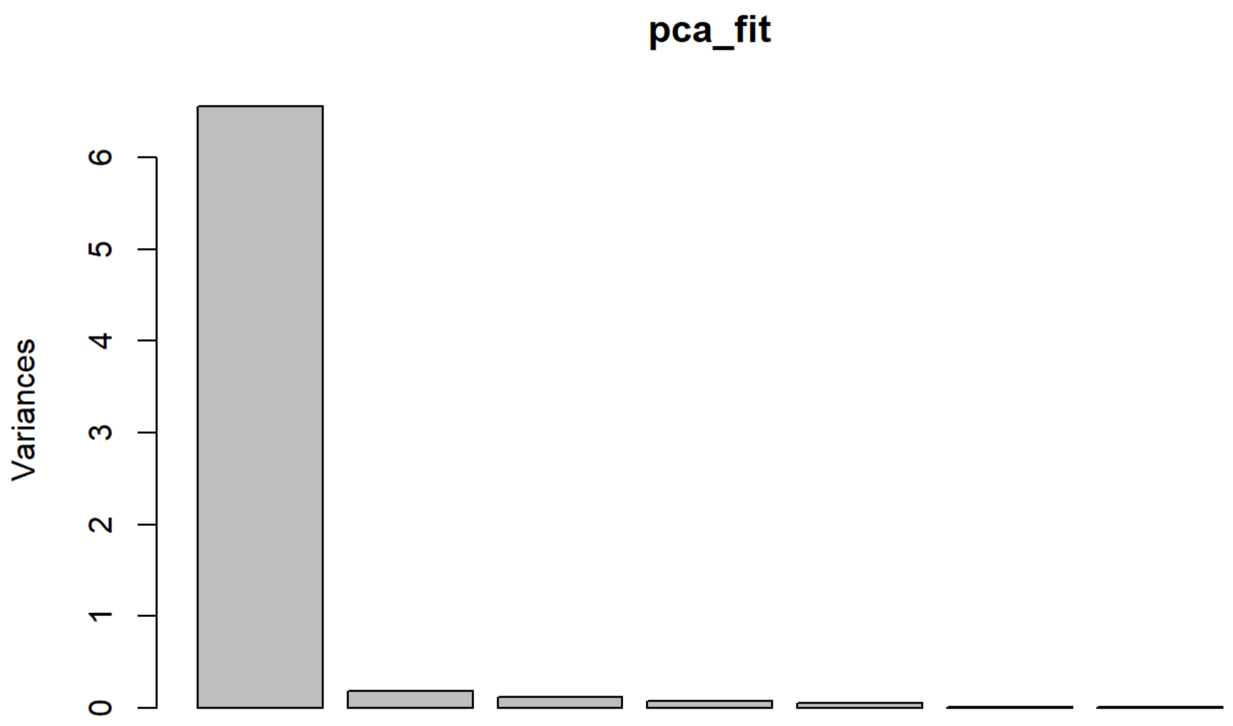
We see some of the original issues persist. Given the persistence of these issues, along with multicollinearity, I believe Principal Component Analysis (PCA) may be a viable option. Let’s give it a go!

PCA

```
pca_inputs <- scale(select(train, -id, -Age, -Sex))
pca_fit <- prcomp(pca_inputs, center = TRUE, scale. = TRUE)

summary(pca_fit)
```

```
## Importance of components:
##                           PC1     PC2     PC3    PC4     PC5     PC6     PC7
## Standard deviation     2.5613 0.42673 0.33302 0.2737 0.22806 0.10188 0.09812
## Proportion of Variance 0.9372 0.02601 0.01584 0.0107 0.00743 0.00148 0.00138
## Cumulative Proportion  0.9372 0.96317 0.97901 0.9897 0.99714 0.99862 1.00000
```

```
plot(pca_fit)
```



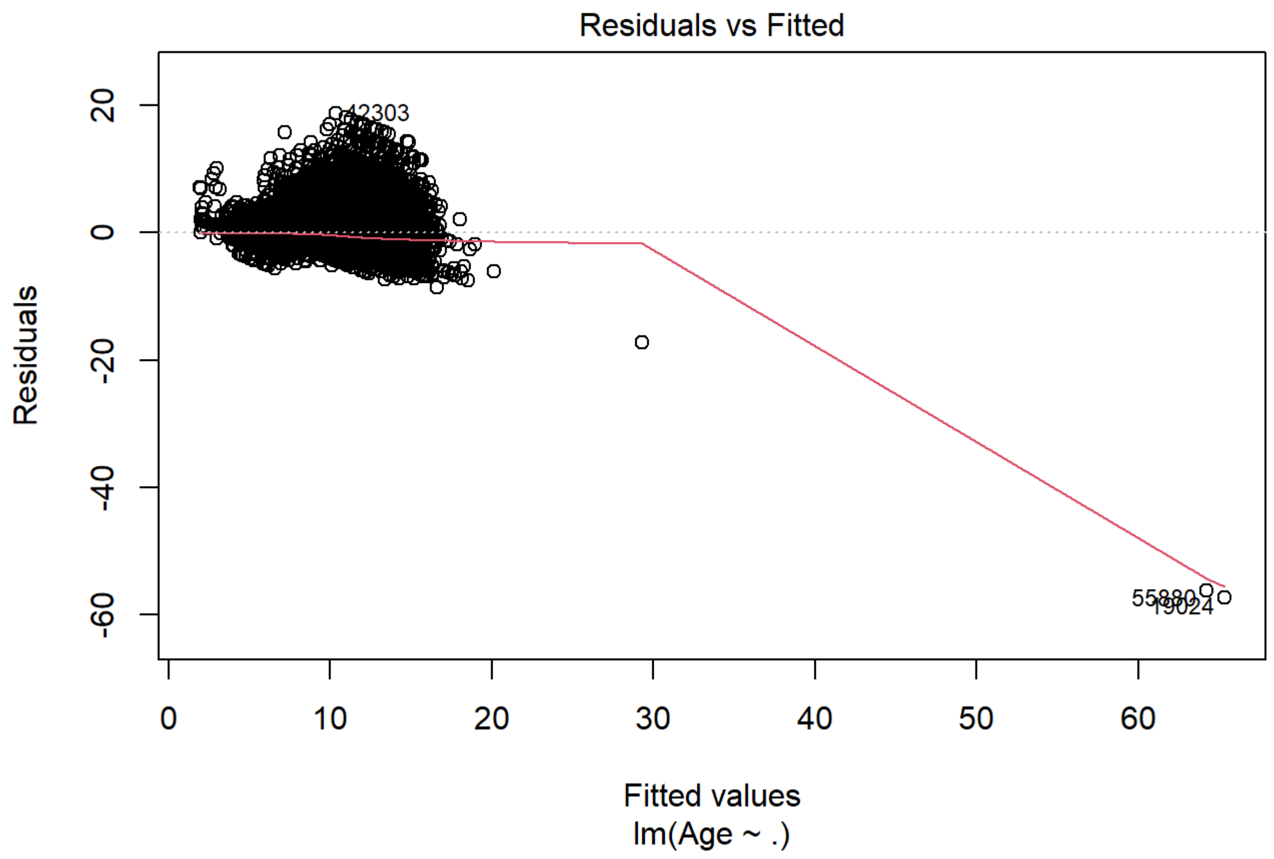
We end up with 7 principal components, although the first drives explains the vast majority of the variance. This is somewhat expected, given the high degree of multicollieanrity and redundancy across predictors. I’ll therefore only take the first three components to use in modeling.

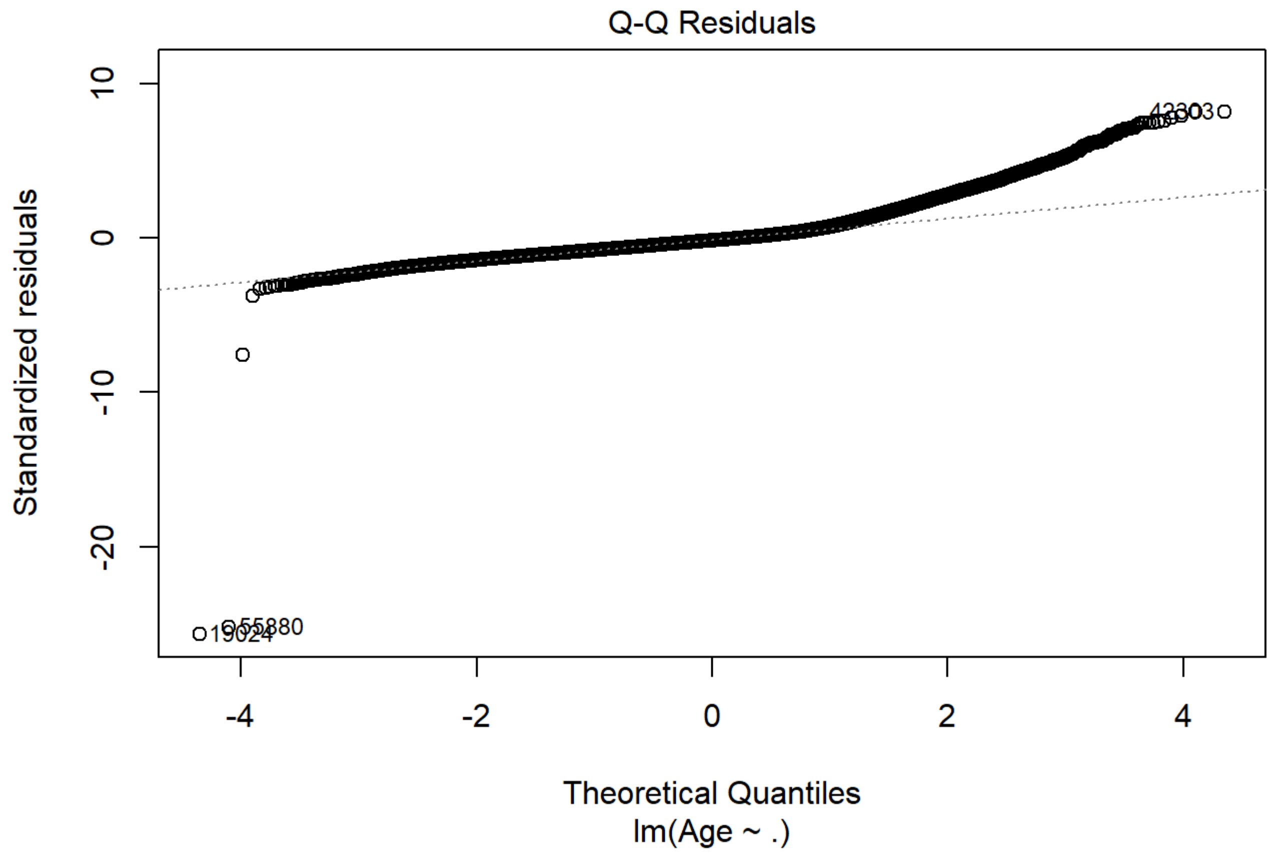
We’ll add back the categorical Sex variable, along with our target variable, then fit another model.

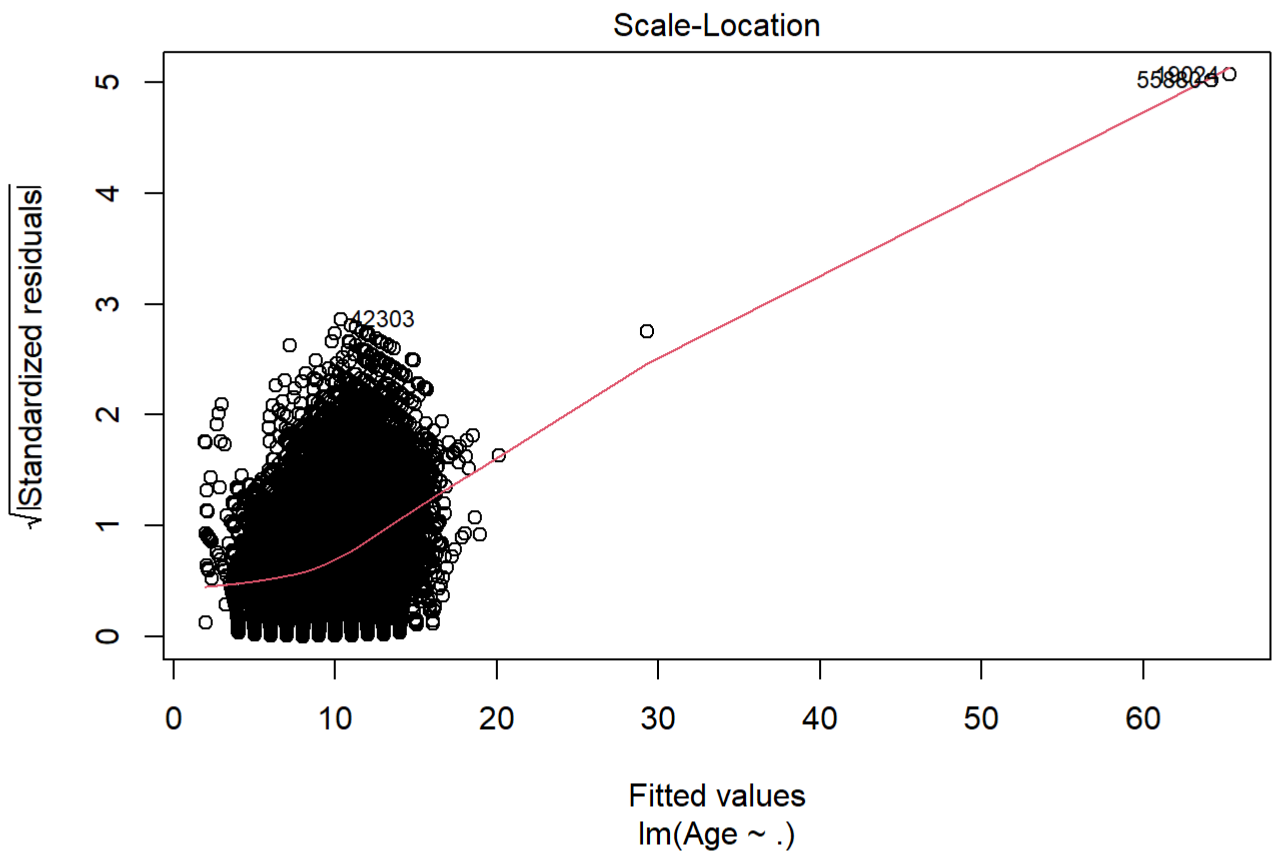
```r
train_pca <- data.frame(pca_fit$x[, 1:3]) %>%
  mutate(Sex = train$Sex, Age = train$Age)

lm_pca <- lm(Age ~ ., data = train_pca)
summary(lm_pca)
```

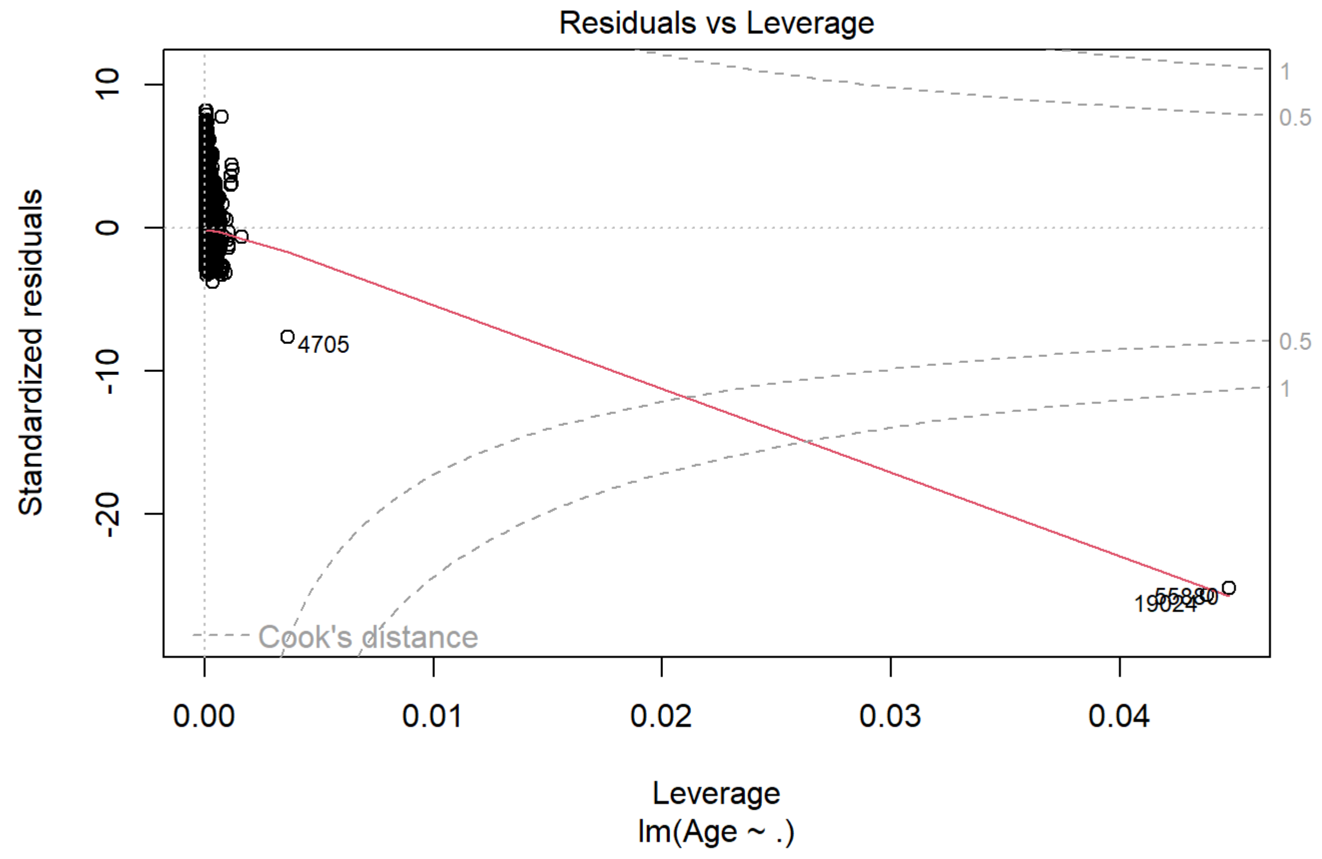
```
##
## Call:
## lm(formula = Age ~ ., data = train_pca)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -57.337  -1.363  -0.391   0.741  18.684
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 10.424254   0.016194  643.71   <2e-16 ***
## PC1         -0.637052   0.004381 -145.41   <2e-16 ***
## PC2          1.419566   0.019758   71.85   <2e-16 ***
## PC3         -1.883732   0.025214  -74.71   <2e-16 ***
## SexI        -1.201214   0.026930  -44.60   <2e-16 ***
## SexM        -0.185460   0.020537   -9.03   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.284 on 74045 degrees of freedom
## Multiple R-squared:  0.4827, Adjusted R-squared:  0.4827
## F-statistic: 1.382e+04 on 5 and 74045 DF,  p-value: < 2.2e-16
```

```r
plot(lm_pca)
```


Residuals vs Fitted
Residuals
Fitted values
lm(Age ~ .)
42303
55880
19024
20
0
-20
-40
-60
0
10
20
30
40
50
60


Q-Q Residuals
Standardized residuals
Theoretical Quantiles
lm(Age ~ .)
42303
55880
19024
10
0
-10
-20
-4
-2
0
2
4


Scale-Location
√|Standardized residuals|
0
1
2
3
4
5
42303
19024
55880
0
10
20
30
40
50
60
Fitted values
lm(Age ~ .)

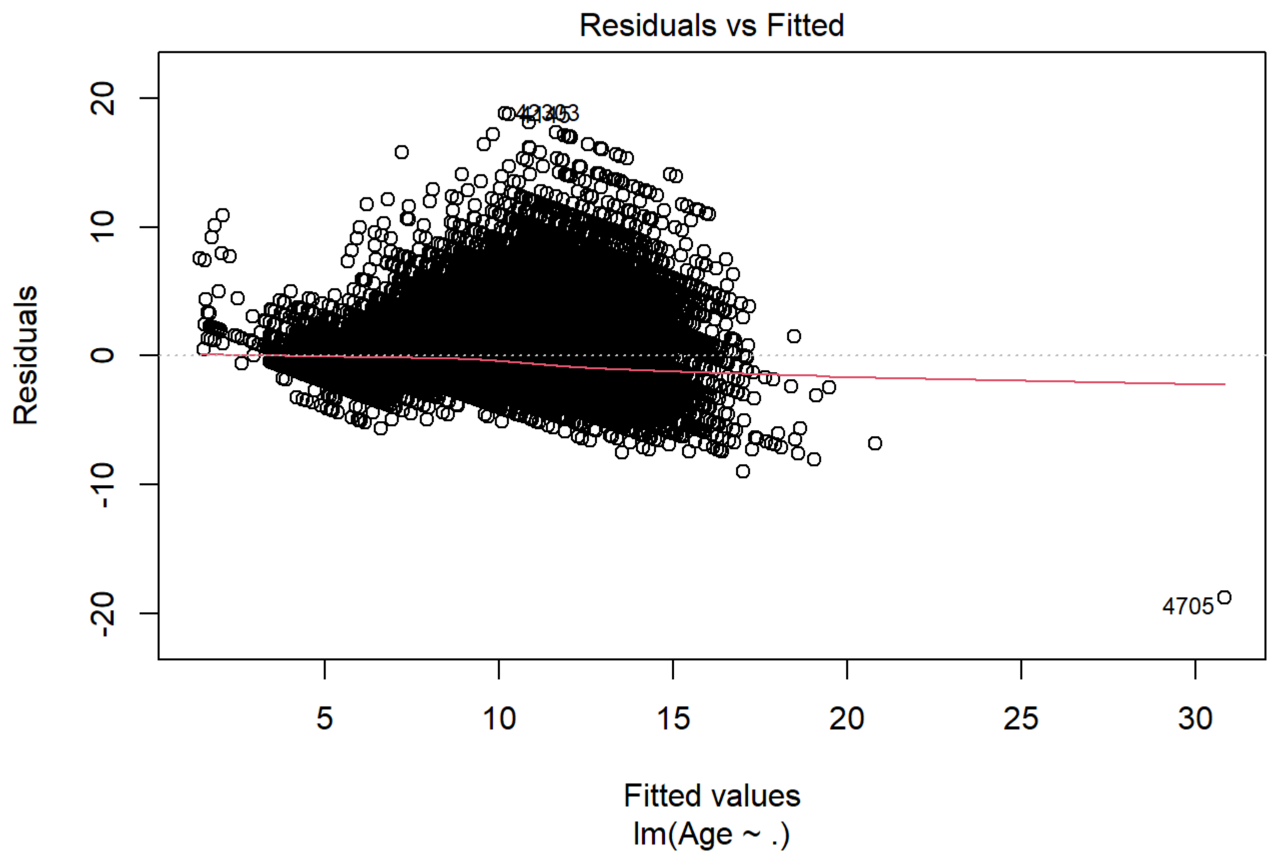

Residuals vs Leverage
Standardized residuals
10
0
-10
-20
4705
55880
19024
Cook's distance
1
0.5
0.5
1
0.00
0.01
0.02
0.03
0.04
Leverage
lm(Age ~ .)

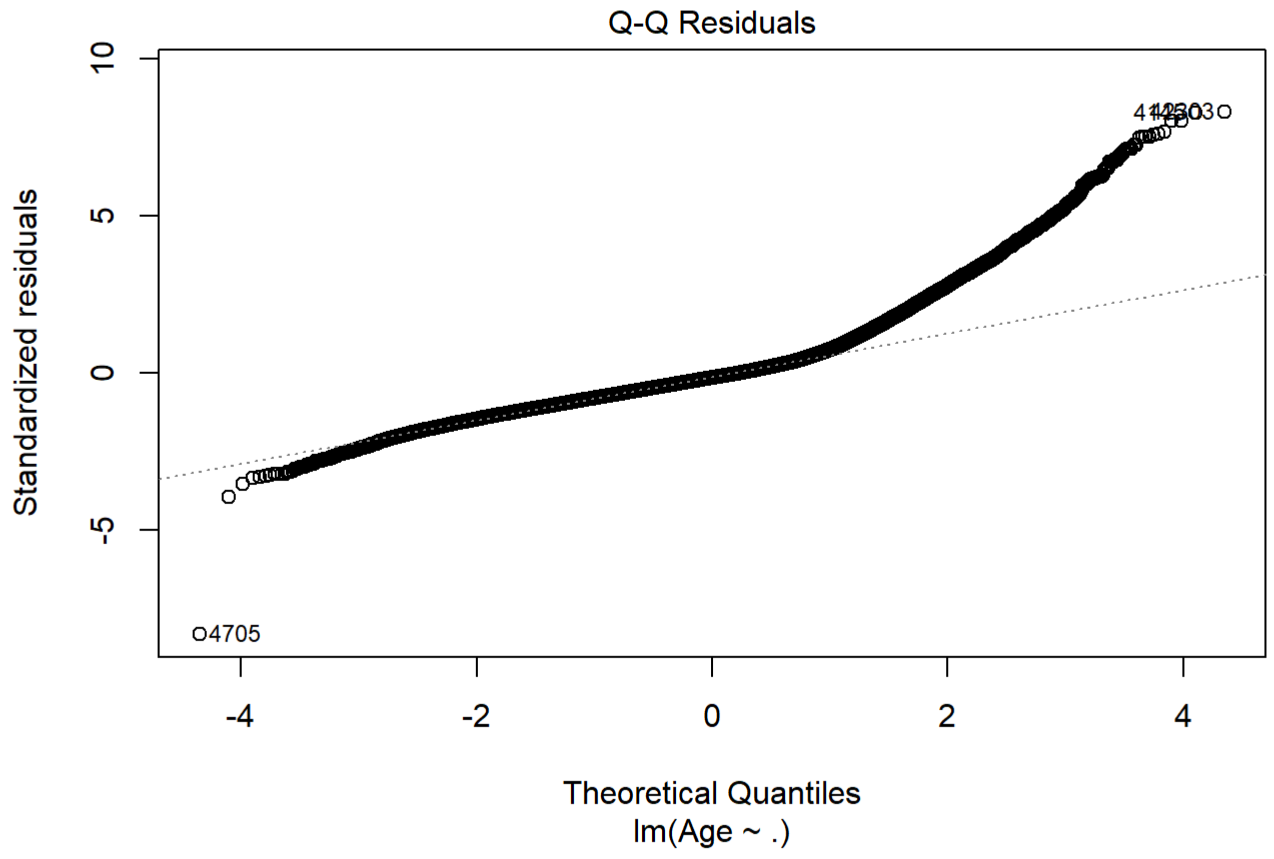
Our residual plots look much better. However, our R-squared is relatively low at ~50%, and the two influential points remains. Let’s fit this model without that point and see how things change.

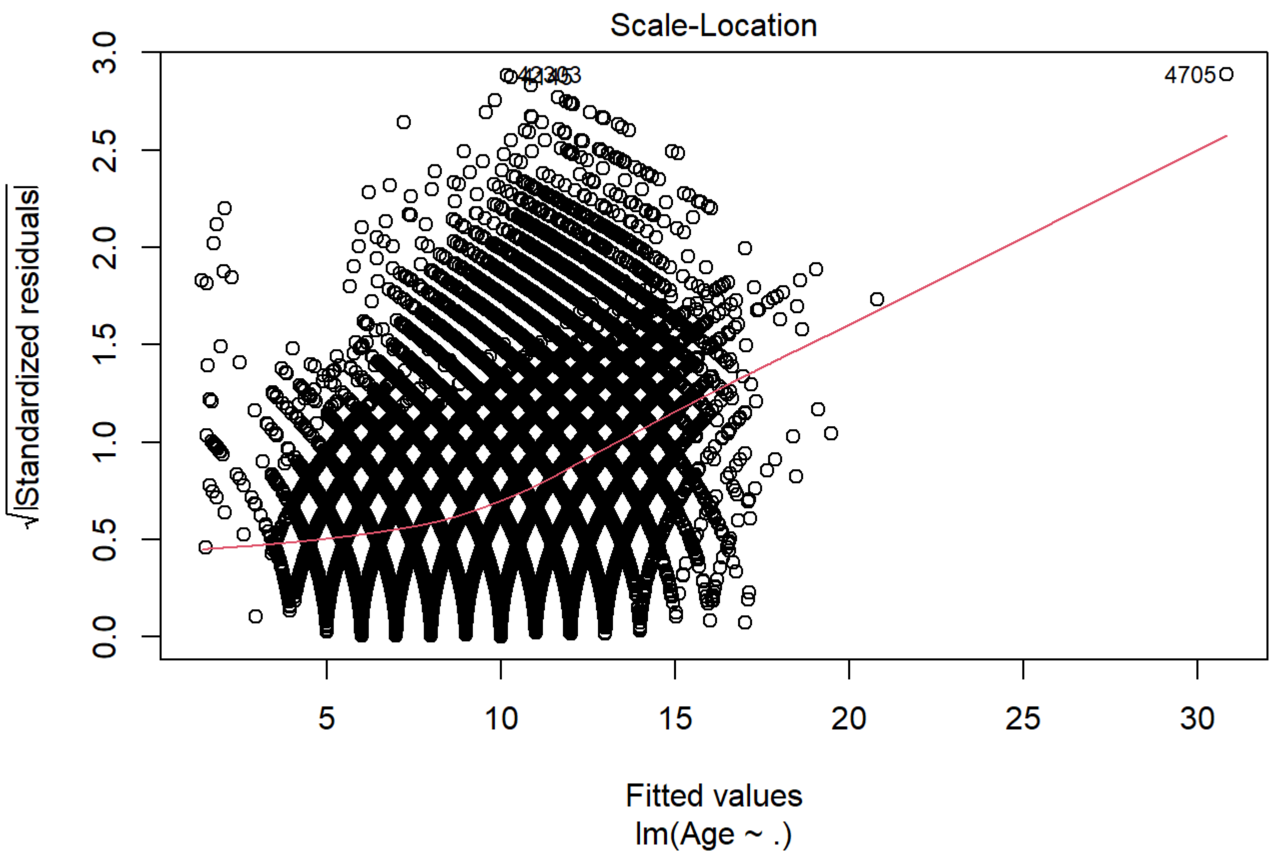
```
train_pca2 <- train_pca[-which(rownames(train_pca) == '55880'),]
train_pca2 <- train_pca2[-which(rownames(train_pca2) == '19024'),]

lm_pca <- lm(Age ~ ., data = train_pca2)
summary(lm_pca)
```

```
##
## Call:
## lm(formula = Age ~ ., data = train_pca2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18.8189  -1.3569  -0.3730   0.7585  18.8129
##
## Coefficients:
##              Estimate Std. Error  t value Pr(>|t|)
## (Intercept) 10.411749   0.016049  648.744   <2e-16 ***
## PC1         -0.643641   0.004345 -148.145   <2e-16 ***
## PC2          1.568261   0.019981   78.488   <2e-16 ***
## PC3         -2.100777   0.025656  -81.882   <2e-16 ***
## SexI        -1.162985   0.026704  -43.551   <2e-16 ***
## SexM        -0.180485   0.020349   -8.869   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.263 on 74043 degrees of freedom
## Multiple R-squared:  0.4922, Adjusted R-squared:  0.4922
## F-statistic: 1.435e+04 on 5 and 74043 DF,  p-value: < 2.2e-16
```

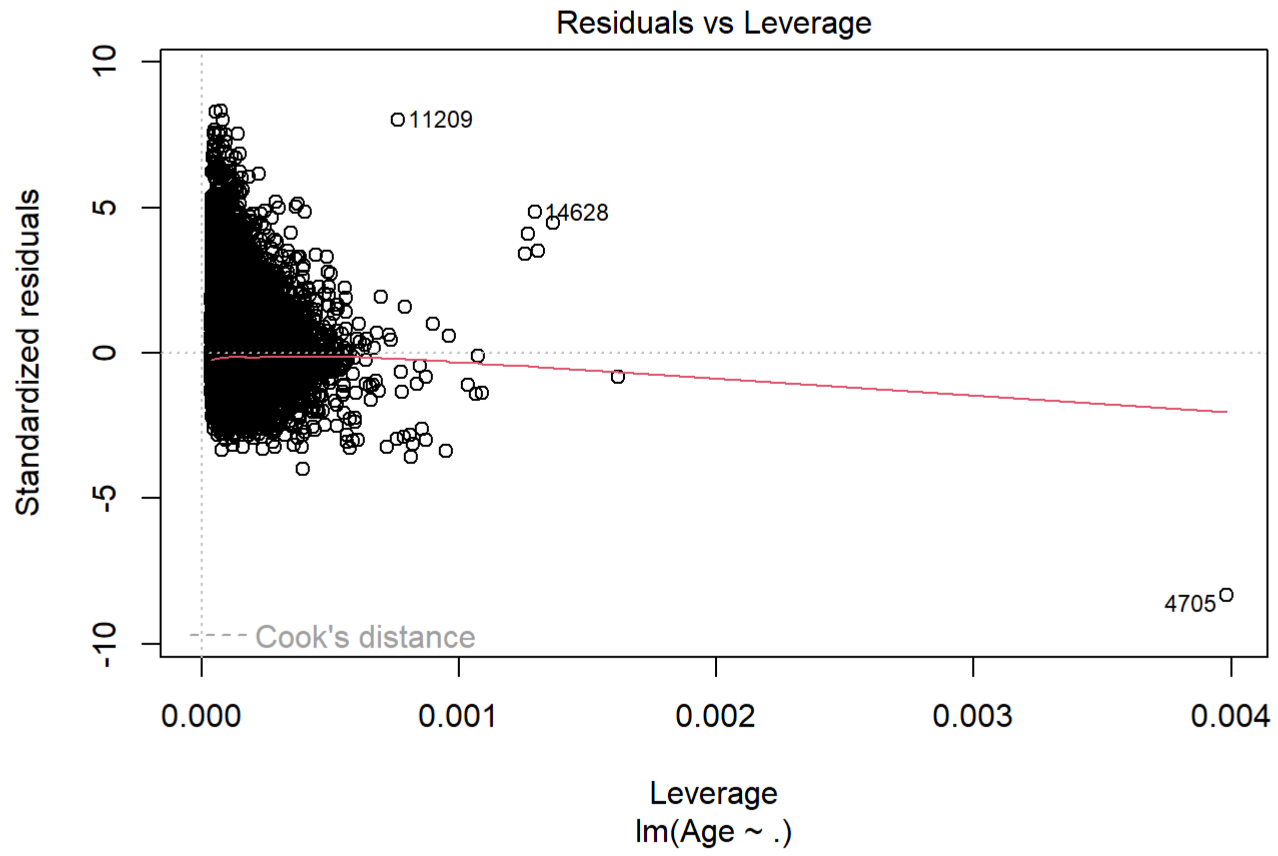
```
plot(lm_pca)
```


Residuals vs Fitted
Residuals
Fitted values
lm(Age ~ .)
4705
20
10
0
-10
-20
5
10
15
20
25
30


Q-Q Residuals
Standardized residuals
Theoretical Quantiles
lm(Age ~ .)
4705
10
5
0
-5
-4
-2
0
2
4



Scale-Location
√|Standardized residuals|
Fitted values
lm(Age ~ .)
4705
42303



Residuals vs Leverage
Standardized residuals
Leverage
lm(Age ~ .)
11209
14628
4705
Cook's distance

I don’t actually see a big difference in coefficients, so it appears we are fine to leave out these points.

Ultimately, it seems there is some non-linearity in the model. So, modeling approaches that handle non-linearity may provide better fits. Regarding the linear models we’ve considered, however, this appears to provide the best while observing the key assumptions of regression.

With this final model, we can generate our predictions for submission on Kaggle!

```r
test_scaled <- scale(select(test, -id, -Sex))
test_pca <- predict(pca_fit, newdata = test_scaled)
test_pca<- data.frame(test_pca[, 1:3]) %>%
  mutate(Sex = test$Sex)
predictions <- predict(lm_pca, newdata = test_pca)
predictions <- data.frame(id = test$id, Age = predictions)

write.csv(predictions, 'data/crab_kaggle_submission_kac624.csv', row.names = FALSE)
```

My submission had a score (the MAE) of 1.57413. Not bad! Curious to see how this stacks up to other submissions.