Merge, intro to optimization and variable transformation with R

Olalla Díaz-Yáñez

4/8/2018

Contents

Merge

Merge function help us to merge two data frames by common columns or row names.

It is always best to explicitly state the identifiers on which you want to merge; it's safer if the input data.frames change unexpectedly and easier to read later on. By using the merge function and its optional parameters you can specify by which variable you want to merge the frames to be sure that the matching is in the fields you desire.

Lets see now some examples of different merges.

We are going to create two simple data.frames:

```
dfA <- data.frame (TreeId = c(1:6), Specie = c(rep ("Pine", 3), rep ("Spruce", 3)))
dfB <- data.frame (TreeId = c(2, 4, 6), DevelopmentClass = c(rep ("Young", 2),
                                                             rep ("Old", 1)))

print(dfA)
```

```
##   TreeId Specie
## 1      1   Pine
## 2      2   Pine
## 3      3   Pine
## 4      4 Spruce
## 5      5 Spruce
## 6      6 Spruce
```

```
print(dfB)
```

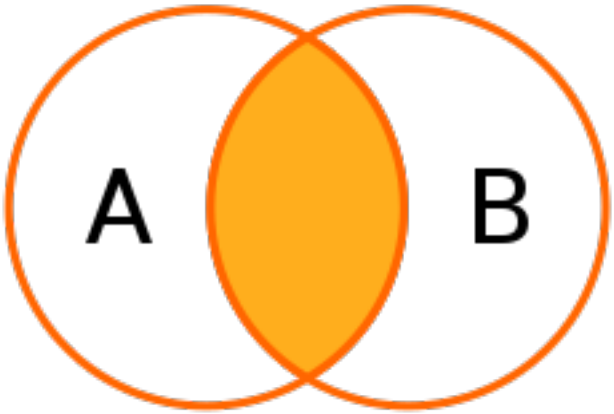


Figure 1: Inner merge (Source: wikipedia)

```
##   TreeId DevelopmentClass
## 1      2            Young
## 2      4            Young
## 3      6              Old
```

Inner merge

An inner merge requires each row in the two merged tables to have matching column values. The inner merge will create a new table with those trees that have same ID number in A and B, it will compare each row of A with each row of B to find all pairs of rows that have same tree ID.

```
merge(dfA, dfB, by = "TreeId")
```

```
##   TreeId Specie DevelopmentClass
## 1      2   Pine            Young
## 2      4 Spruce            Young
## 3      6 Spruce              Old
```

You can also use the by.x and by.y parameters if the matching variables have different names in the different data frames, for example:

```
dfC <- data.frame (TreeID = c(1:6), Specie = c(rep ("Pine", 3), rep ("Spruce", 3)))
dfD <- data.frame (TreeId = c(2, 4, 6), DevelopmentClass = c(rep ("Young", 2),

print(dfC)
```

```
##   TreeID Specie
## 1      1   Pine
## 2      2   Pine
## 3      3   Pine
## 4      4 Spruce
## 5      5 Spruce
## 6      6 Spruce
```

```
print(dfD)
```

```
##   TreeId DevelopmentClass
## 1      2            Young
## 2      4            Young
```

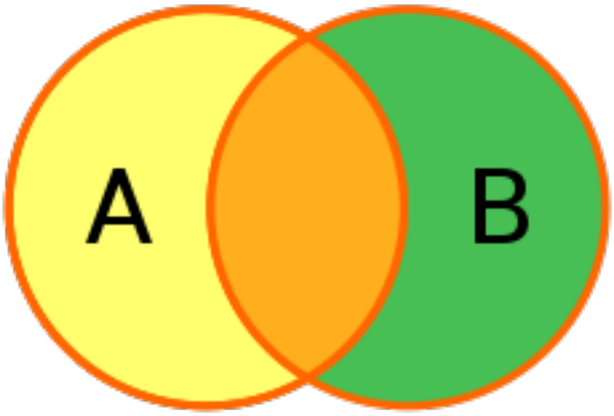


Figure 2: Full outer merge (Source: wikipedia)

```
## 3      6              Old
```

```
merge(dfC, dfD, by.x = "TreeID", by.y = "TreeId")
```

```
##   TreeID Specie DevelopmentClass
## 1      2   Pine            Young
## 2      4 Spruce            Young
## 3      6 Spruce              Old
```

Outer merge

An Outer merge could be a full outer merge, a left outer merge or right outer merge.

Full outer merge

A full outer merge combines the effect of applying both left and right outer merges. In this case we will have all rows from both tables, including those that do not match, in those unmatching cases we will have NULL value for the new common variables:

```
merge(x = dfA, y = dfB, by = "TreeId", all = TRUE)
```

```
##   TreeId Specie DevelopmentClass
## 1      1   Pine             <NA>
## 2      2   Pine            Young
## 3      3   Pine             <NA>
## 4      4 Spruce            Young
## 5      5 Spruce             <NA>
## 6      6 Spruce              Old
```

Left outer merge

A left outer merge for tables A and B always contains all rows of the "left" table (A), even if the merge-condition does not find any matching row in the "right" table (B). Here we will also have NULL values in rows (TreeIds) that are in A but not in B.

```
merge(x = dfA, y = dfB, by = "TreeId", all.x = TRUE)
```

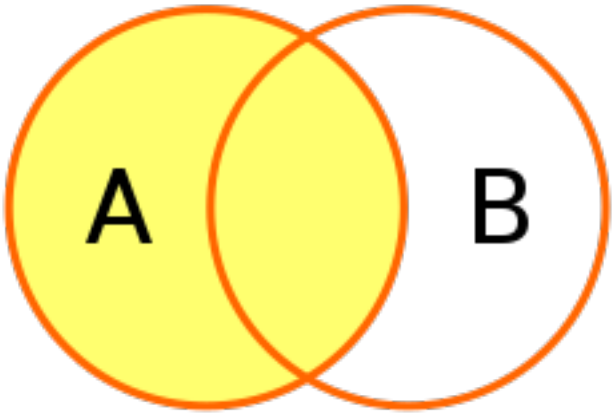


Figure 3: Left outer merge (Source: wikipedia)

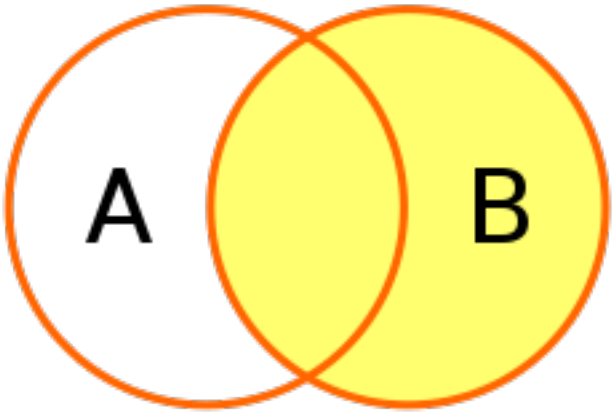


Figure 4: Left outer merge (Source: wikipedia)

```
##   TreeId Specie DevelopmentClass
## 1      1   Pine             <NA>
## 2      2   Pine            Young
## 3      3   Pine             <NA>
## 4      4 Spruce            Young
## 5      5 Spruce             <NA>
## 6      6 Spruce              Old
```

Right outer merge

A right outer merge for tables A and B always contains all rows of the "right" table (B), even if the merge-condition does not find any matching row in the "left" table (A). Here we will also have NULL values in rows (TreeIds) that are in B but not in A.

```
merge(x = dfA, y = dfB, by = "TreeId", all.y = TRUE)
```

```
##   TreeId Specie DevelopmentClass
## 1      2   Pine            Young
## 2      4 Spruce            Young
## 3      6 Spruce              Old
```

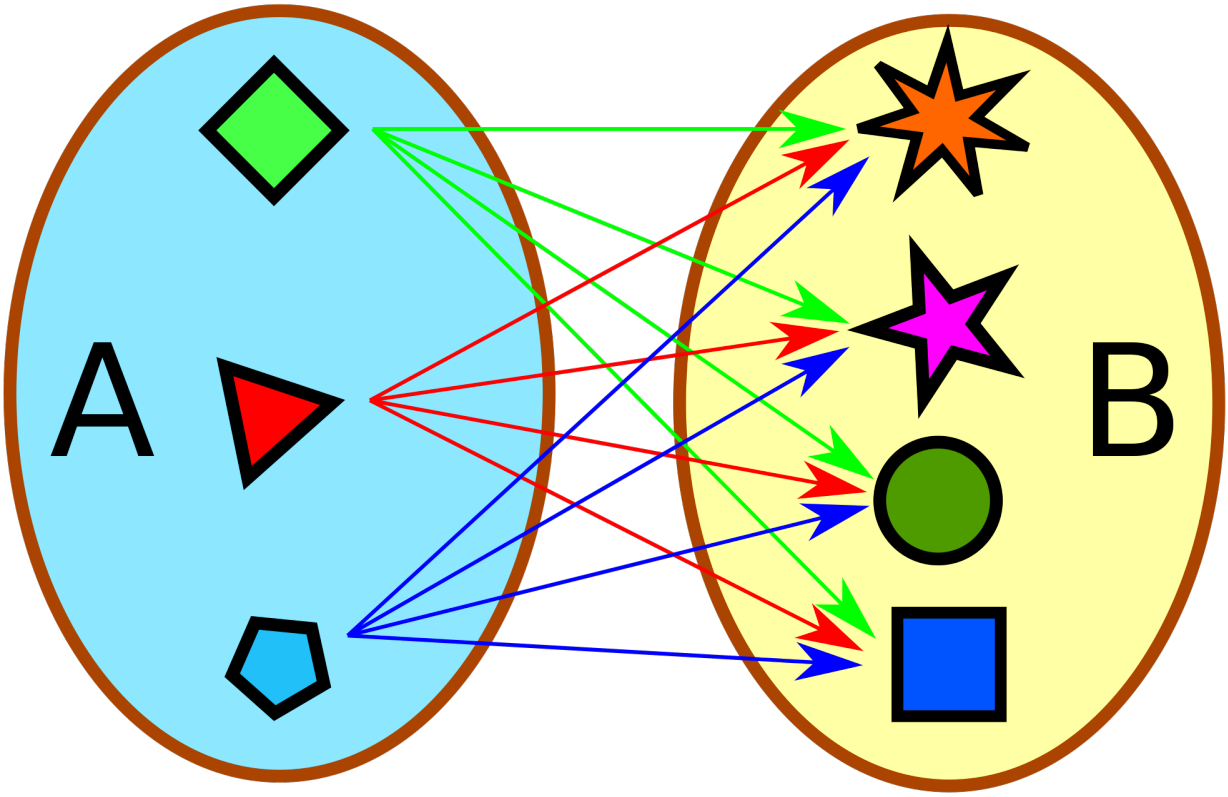


Figure 5: Left outer merge (Source: wikipedia)

Cross merge

A cross merge is just as the inner merge but without specifiying what is the matching variable, it is kind of "all with all" matching where the resulting table will show all posible combinations.

```
merge(x = dfA, y = dfB, by = NULL)
```

```
##    TreeId.x Specie TreeId.y DevelopmentClass
## 1         1   Pine        2            Young
## 2         2   Pine        2            Young
## 3         3   Pine        2            Young
## 4         4 Spruce        2            Young
## 5         5 Spruce        2            Young
## 6         6 Spruce        2            Young
## 7         1   Pine        4            Young
## 8         2   Pine        4            Young
## 9         3   Pine        4            Young
## 10        4 Spruce        4            Young
## 11        5 Spruce        4            Young
## 12        6 Spruce        4            Young
## 13        1   Pine        6              Old
## 14        2   Pine        6              Old
## 15        3   Pine        6              Old
## 16        4 Spruce        6              Old
## 17        5 Spruce        6              Old
```

```
## 18        6 Spruce        6              Old
```

Intro to optimization with R

A mathematical optimization uses a rigorous mathematical model to determine the most efficient solution to a problem. First is necessary to identify what is the objective, the objective is a quantitative measure of the performance (e.g. cubic metres of wood or kg of berries), in general, any quantity (or combination thereof) represented as a single number.

Problem type	Package	Routine
General purpose (1-dim.)	Built-in	optimize(...)
General purpose (n-dim.)	Built-in	optim(...)
Linear Programming	lpSolve	lp(...)
Quadratic Programming	quadprog	solve.QP(...)
Quadratic Programming	quadprog	solve.QP(...)
Non-Linear Programming	optimize	optimize(...)
optimx	optimx(...)	
General interface	ROI	ROI_solve(...)

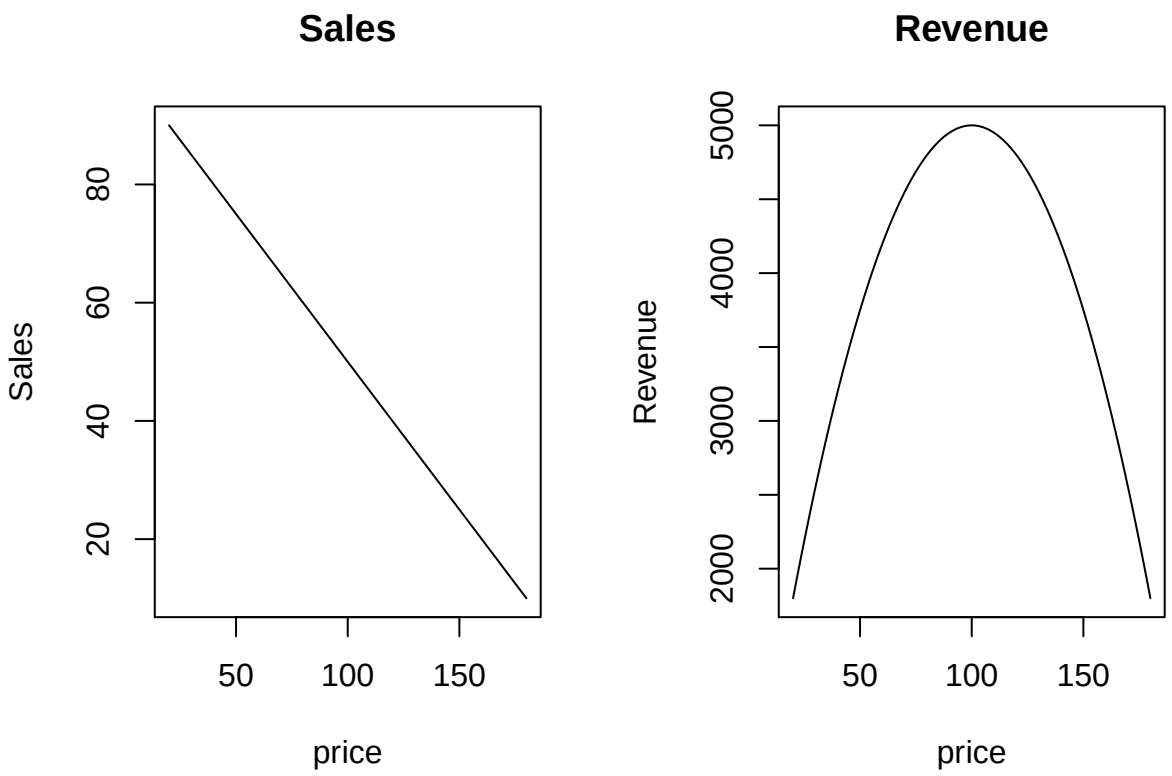
All available packages are listed in the CRAN task view for optimization and mathematical programming: "Optimization and mathematical programming" https://cran.r-project.org/web/views/Optimization.html

The format is generic to all the functions: optimizer(objective, constraints, bounds, types, maximum)

We can see how the function optimize() is used with a simple example:

```r
# These are the sales and revenue functions for one product
sales <- function(price) {100 - 0.5 * price }
revenue <- function(price) {price * sales(price)}

# We can plot this functions to see how they behave
par(mfrow=c(1, 2))
curve(sales, from = 20, to = 180, xname = "price", ylab = "Sales", main = "Sales")
curve(revenue, from = 20, to = 180, xname = "price", ylab = "Revenue", main = "Revenue")
```



We can now use the optimize function to find which price value will give the highest sales.

```
optimize(revenue, interval=c(20, 180), maximum=TRUE)
```

```
## $maximum
## [1] 100
##
## $objective
## [1] 5000
```

As we observed graphically when we charge a price of 100 EUR, and expect to get 5.000 EUR in revenue.

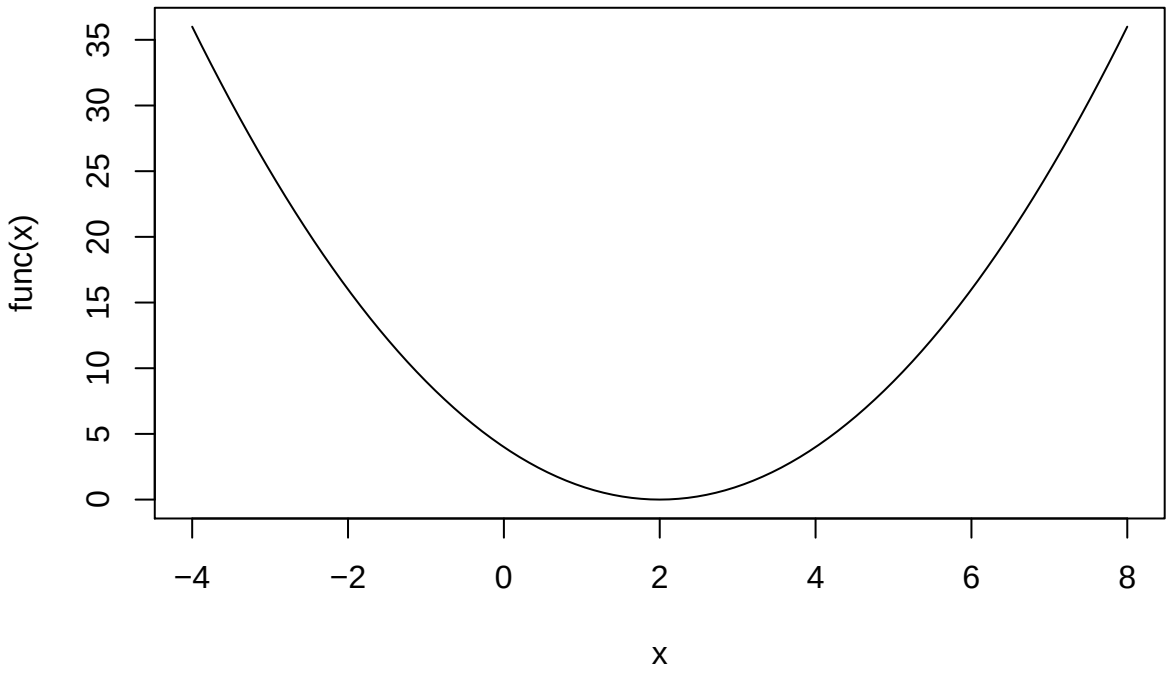
Another example but this time looking for the minimun:

```
func <- function(x){
  return ( (x-2)^2 )}

# What is the value of the functions when x = -2
func(-2)
```

```
## [1] 16
```

```
# plot the function
curve(func,-4,8)
```




```
# you can find the minimum using again the optimize function
optimize (f = func, interval = c(-10, 10))
```

```
## $minimum
## [1] 2
##
## $objective
## [1] 0
```

The challenge is that in forestry we will typically have situations way more complex where multiple values would be involved and several restrictions aplied. In those cases first you will have to define which method is the best approach for each case, as problems vary in size and complexity. Depending on the problem optimization methods have specific advantages.

Another cool optimization problem is the circus tent problem, you can find a visual example here.

NOTE: Subsetting is not the same as optimizing.

Data transformation in R

During the research methods course we saw that when we model continuous response variables using linear models we made several important assumptions about its behaviors:

- constant variance
- normal errors
- independent errors
- random sampling

"Data transformation" is a fancy term for changing the values of observations through some mathematical operation. Such transformations are simple in R and it is simply to assign a variable the value of the transformed variable.

I found a nice review in this page

Reciprocal

The reciprocal, x to 1/x, with its sibling the negative reciprocal, x to -1/x, is a very strong transformation with a drastic effect on distribution shape. It can not be applied to zero values. Although it can be applied to negative values, it is not useful unless all values arepositive. The reciprocal of a ratio may often be interpreted as easily as the ratio itself: e.g.

- population density (people per unit area) becomes area per person;
- persons per doctor becomes doctors per person;
- rates of erosion become time to erode a unit depth.

The reciprocal reverses order among values of the same sign: largest becomes smallest, etc. The negative reciprocal preserves order among values of the same sign.

Logarithm

The logarithm, x to log base 10 of x, or x to log base e of x (ln x), or x to log base 2 of x, is a strong transformation with a major effect on distribution shape. It is commonly used for reducing right skewness and is often appropriate for measured variables. It can not be applied to zero or negative values. One unit on a logarithmic scale means a multiplication by the base of logarithms being used.

Cube root

The cube root, x to x^(1/3). This is a fairly strong transformation with a substantial effect on distribution shape: it is weaker than the logarithm. It is also used for reducing right skewness, and has the advantage that it can be applied to zero and negative values. Note that for example the cube root of a volume has the units of a length. It is commonly applied to rainfall data.

Square root

The square root, x to x^(1/2) = sqrt(x), is a transformation with a moderate effect on distribution shape: it is weaker than the logarithm and the cube root. It is also used for reducing right skewness, and also has the advantage that it can be applied to zero values. Note that the square root of an area has the units of a length. It is commonly applied to counted data, especially if the values are mostly rather small.

Square

The square, x to x^2, has a moderate effect on distribution shape and it could be used to reduce left skewness. In practice, the main reason for using it is to fit a response by a quadratic function y = a + b x + c x^2. Quadratics have a turning point, either a maximum or a minimum, although the turning point in a function fitted to data might be far beyond the limits of the observations. The distance of a body from an origin is a quadratic if that body is moving under constant acceleration, which gives a very clear physical justification for using a quadratic. Otherwise quadratics are typically used solely because they can mimic a relationship within the data region. Outside that region they may behave very poorly, because they take on arbitrarily large values for extreme values of x, and unless the intercept a is constrained to be 0, they may behave unrealistically close to the origin.

Which transformation?

The main criterion in choosing a transformation is: what works with the data? As examples above indicate, it is important to consider as well two questions.

What makes physical (biological, economic, whatever) sense, for example in terms of limiting behaviour as values get very small or very large? This question often leads to the use of logarithms.

Can we keep dimensions and units simple and convenient? If possible, we prefer measurement scales that are easy to think about. The cube root of a volume and the square root of an area both have the dimensions of length, so far from complicating matters, such transformations may simplify them. Reciprocals usually have simple units, as mentioned earlier. Often, however, somewhat complicated units are a sacrifice that has to be made.

Lets see a simple example on how to transfomr variables and create a model with and withour transformation on the variables:

I obtained this data and example from here and here.

```
# Load the data
shortleaf <- read.table("Data/shortleaf.txt", header=T)
head(shortleaf)
```

```
##   Diam Vol
## 1  4.4 2.0
## 2  4.6 2.2
## 3  5.0 3.0
## 4  5.1 4.3
## 5  5.1 3.0
## 6  5.2 2.9
```

```
# Lets transform the variable Diam
shortleaf$trans_diam <- (shortleaf$Diam)^3     #Raises Y to the power of 3
shortleaf$trans_diam <- (shortleaf$Diam)^(1/9) #Takes the ninth root of Y
shortleaf$trans_diam <- log(shortleaf$Diam)    #Takes the natural logarithm (ln) of Y
shortleaf$trans_diam <- log10(shortleaf$Diam)  #Takes the base-10 logarithm of Y
shortleaf$trans_diam <- exp(shortleaf$Diam)    #Raises the constant e to the power of Y
shortleaf$trans_diam <- abs(shortleaf$Diam)    #Finds the absolute value of Y
shortleaf$trans_diam <- sin(shortleaf$Diam)    #Calculates the sine of Y
shortleaf$trans_diam <- asin(shortleaf$Diam)   #Calculates the inverse sine (arcsine) of Y
```

```
## Warning in asin(shortleaf$Diam): NaNs produced
```

Fit a simple linear regression model of Vol on Diam.

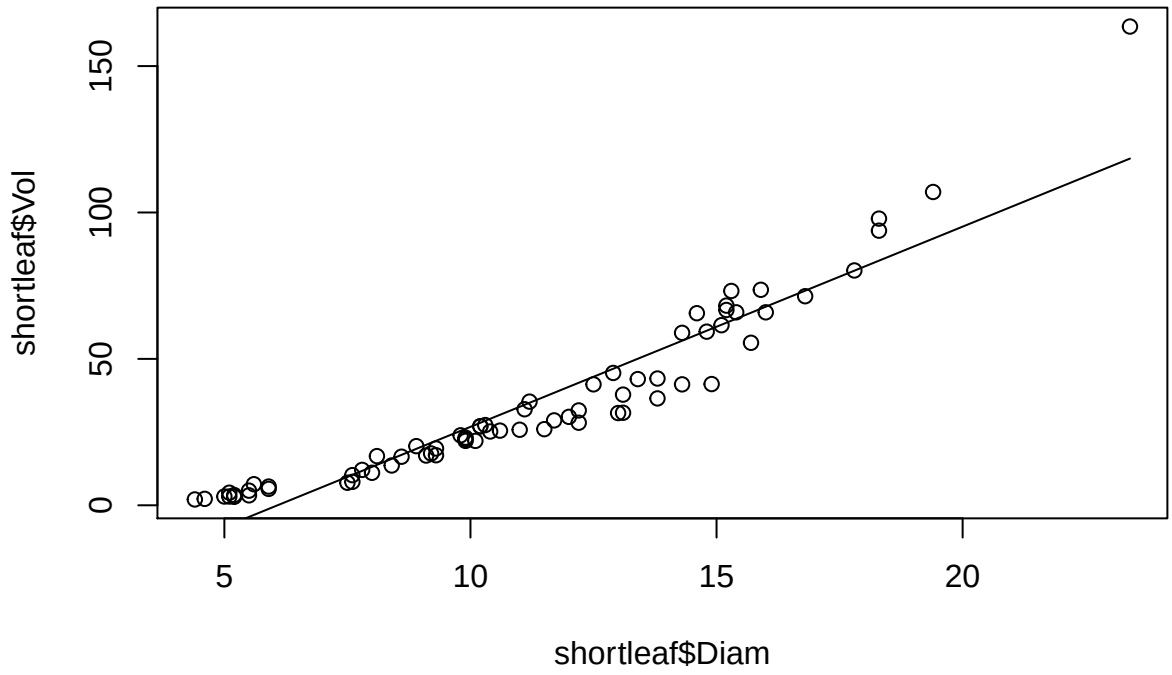
```
model.1 <- lm(shortleaf$Vol ~ shortleaf$Diam)
summary(model.1)
```

```
##
## Call:
## lm(formula = shortleaf$Vol ~ shortleaf$Diam)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -18.899  -4.768  -1.438   6.740  45.089
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -41.5681     3.4269  -12.13   <2e-16 ***
## shortleaf$Diam   6.8367     0.2877   23.77   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 9.875 on 68 degrees of freedom
## Multiple R-squared:  0.8926, Adjusted R-squared:  0.891
## F-statistic: 564.9 on 1 and 68 DF,  p-value: < 2.2e-16
```

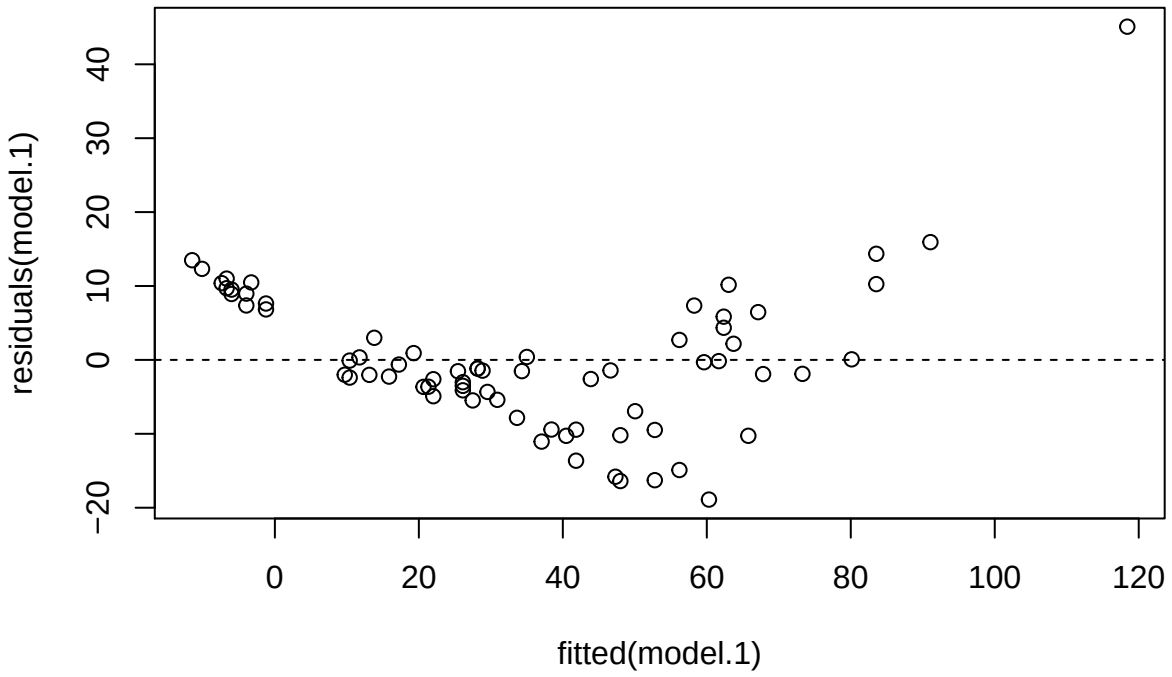
Display scatterplot of the data and add the regression line.

```
plot(x = shortleaf$Diam, y = shortleaf$Vol,
     panel.last = lines(sort(shortleaf$Diam), fitted(model.1)[order(shortleaf$Diam)]))
```



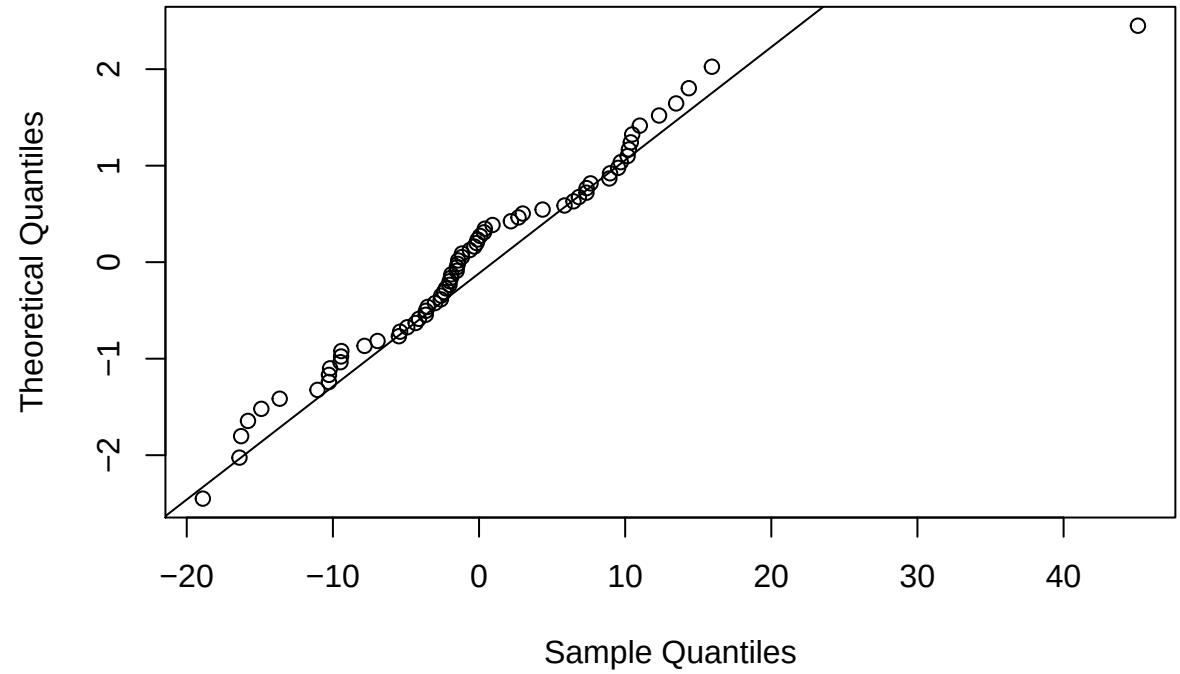
Display residual plot with fitted (predicted) values on the horizontal axis.

```
plot(x = fitted(model.1), y = residuals(model.1),
     panel.last = abline(h=0, lty=2))
```



Display normal probability plot of the residuals and add a diagonal line to the plot.

```
qqnorm(residuals(model.1), main="", datax=TRUE)
qqline(residuals(model.1), datax=TRUE)
```



We can log-transforming both response and predictor to see what happens:

Lets start with log(Diam) variable and fit a simple linear regression model of Vol on log(Diam).

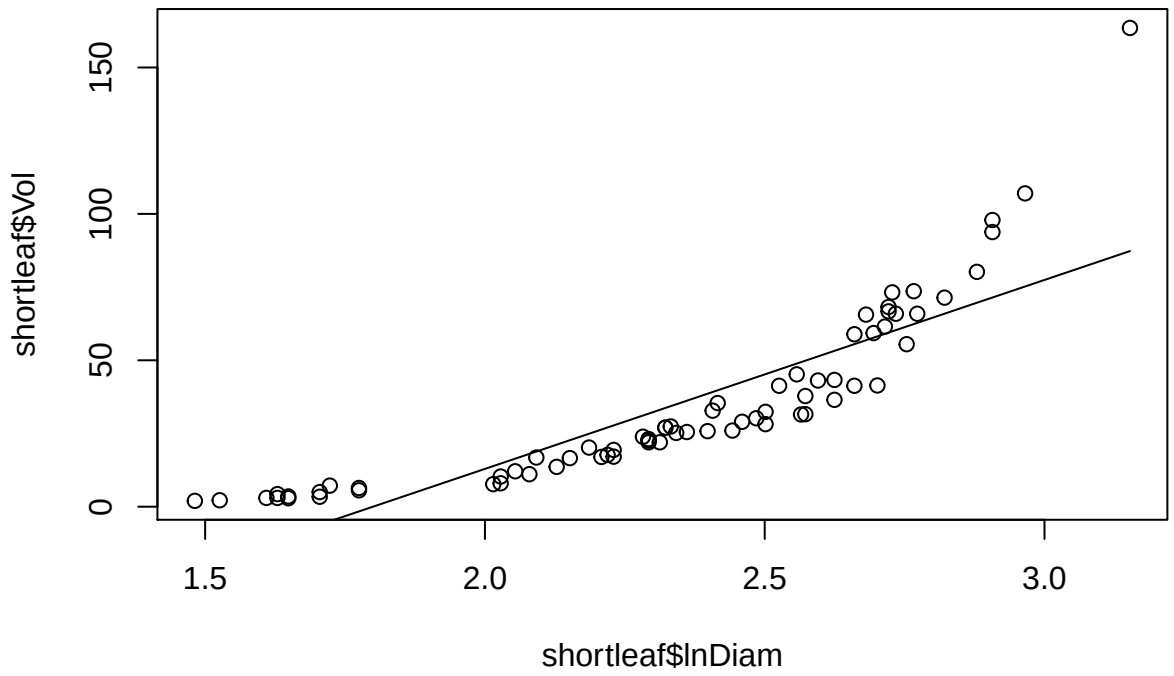
```
shortleaf$lnDiam <- log(shortleaf$Diam)

model.2 <- lm(shortleaf$Vol ~ shortleaf$lnDiam)
```

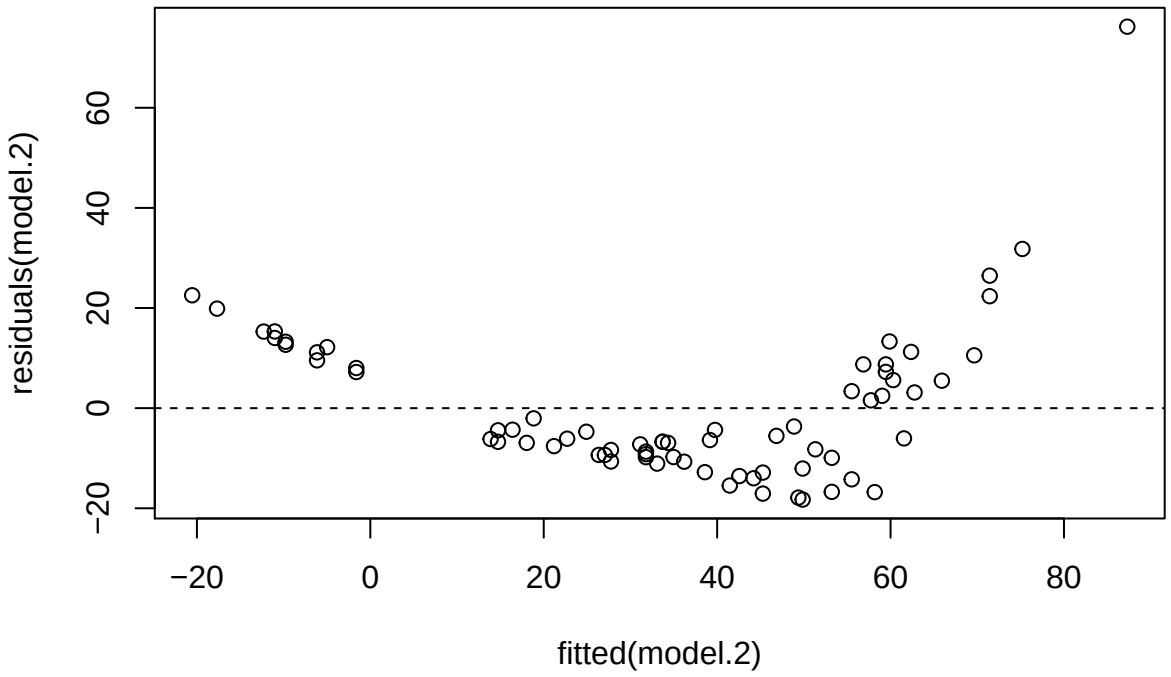
```
summary(model.2)
```

```
##
## Call:
## lm(formula = shortleaf$Vol ~ shortleaf$lnDiam)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -18.264  -9.665  -5.793   8.741  76.198
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      -116.162     10.830  -10.73 2.88e-16 ***
## shortleaf$lnDiam   64.536      4.562   14.15  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.17 on 68 degrees of freedom
## Multiple R-squared:  0.7464, Adjusted R-squared:  0.7427
## F-statistic: 200.2 on 1 and 68 DF,  p-value: < 2.2e-16
```

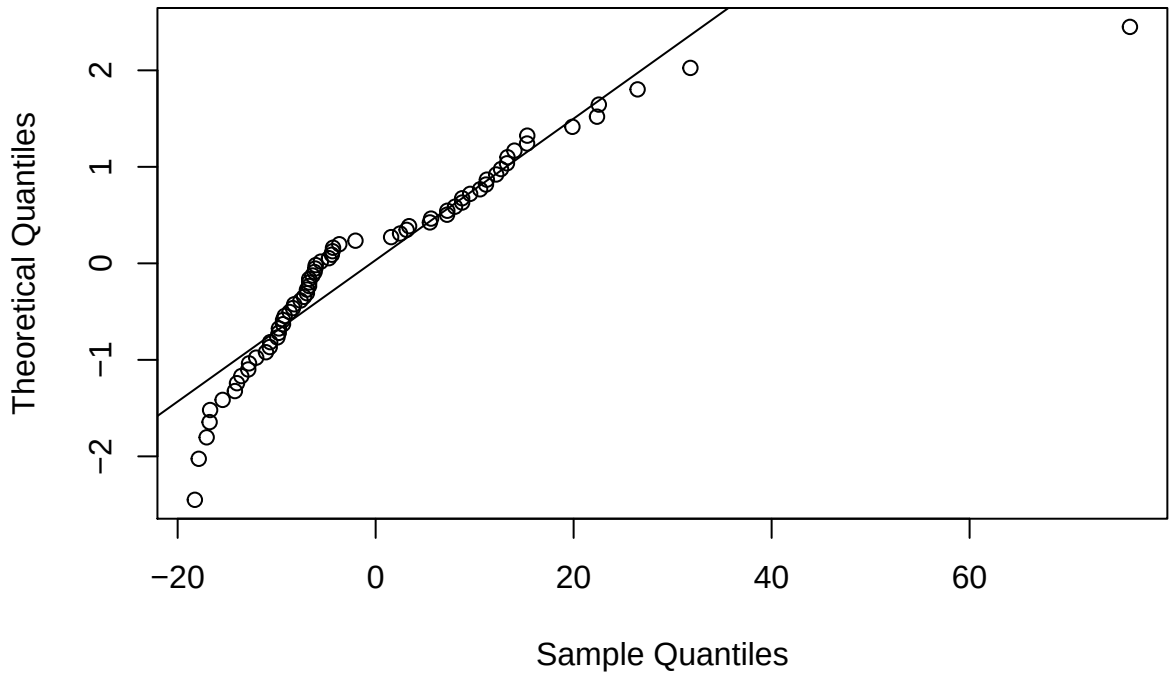
```
plot(x=shortleaf$lnDiam, y=shortleaf$Vol,
     panel.last = lines(sort(shortleaf$lnDiam), fitted(model.2)[order(shortleaf$lnDiam)]))
```




```
plot(x=fitted(model.2), y=residuals(model.2),
     panel.last = abline(h=0, lty=2))
```




```
qqnorm(residuals(model.2), main="", datax=TRUE)
qqline(residuals(model.2), datax=TRUE)
```



Create log(Vol) variable and fit a simple linear regression model of log(Vol) on log(Diam).

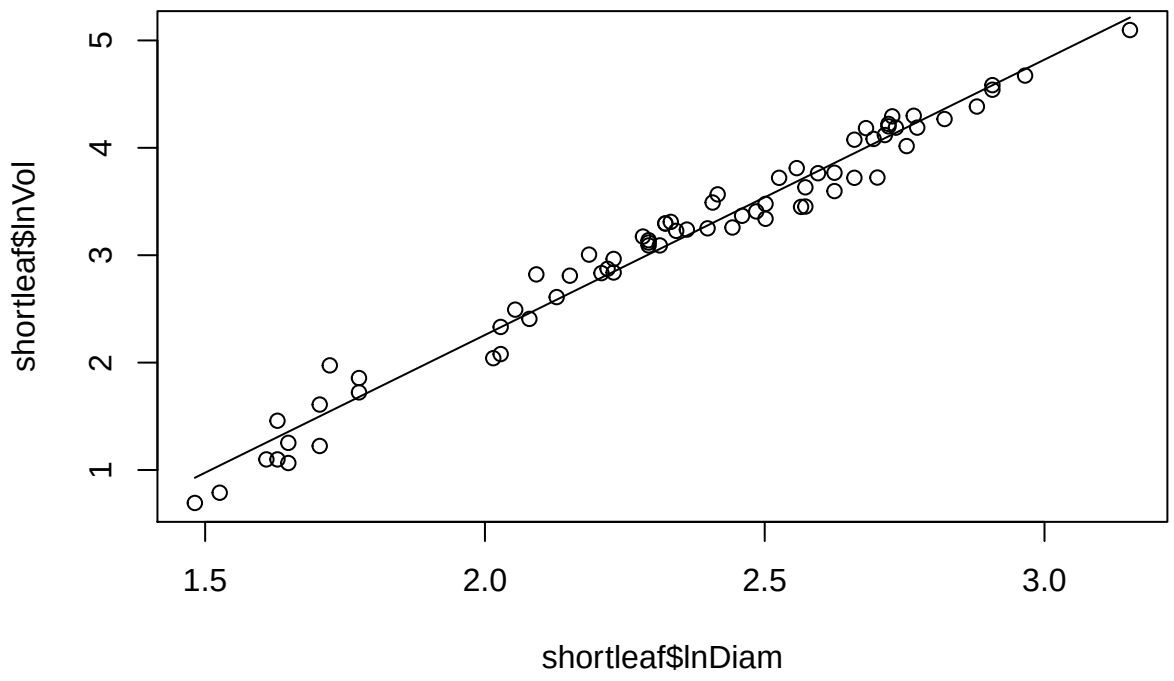
```
shortleaf$lnVol <- log(shortleaf$Vol)
shortleaf$lnDiam <- log(shortleaf$Diam)

model.3 <- lm(shortleaf$lnVol ~ shortleaf$lnDiam)
summary(model.3)
```

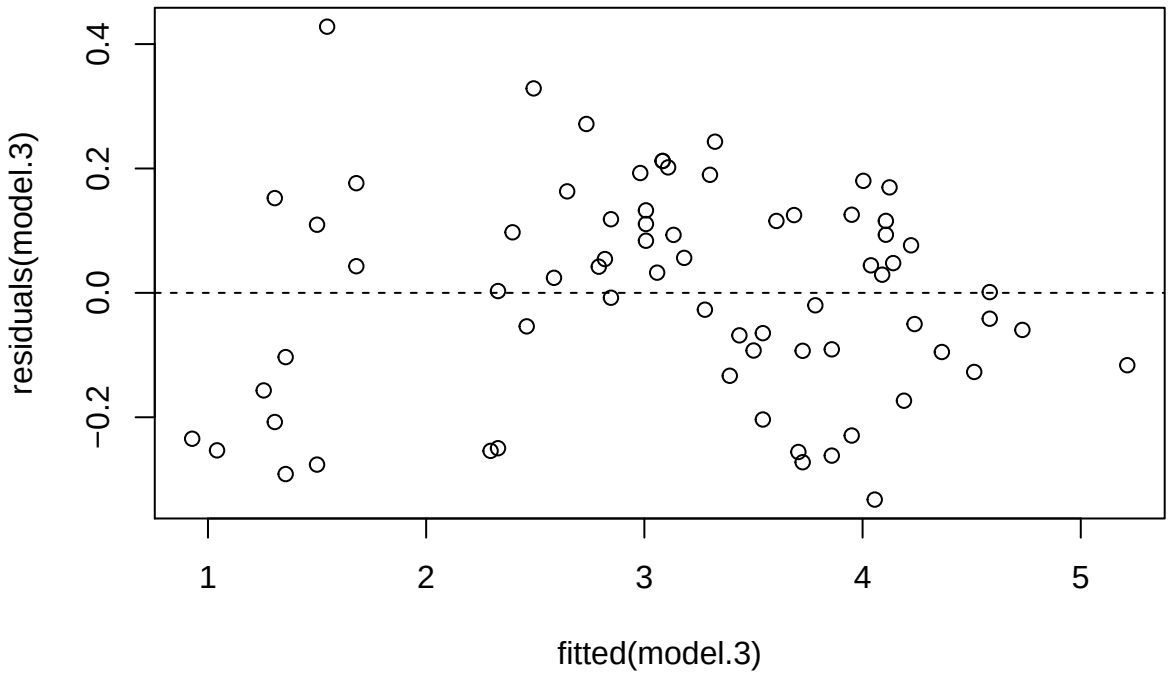
```
##
```

```
## Call:
## lm(formula = shortleaf$lnVol ~ shortleaf$lnDiam)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.3323 -0.1131  0.0267  0.1177  0.4280
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -2.8718     0.1216  -23.63   <2e-16 ***
## shortleaf$lnDiam   2.5644     0.0512   50.09   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1703 on 68 degrees of freedom
## Multiple R-squared:  0.9736, Adjusted R-squared:  0.9732
## F-statistic:  2509 on 1 and 68 DF,  p-value: < 2.2e-16
```

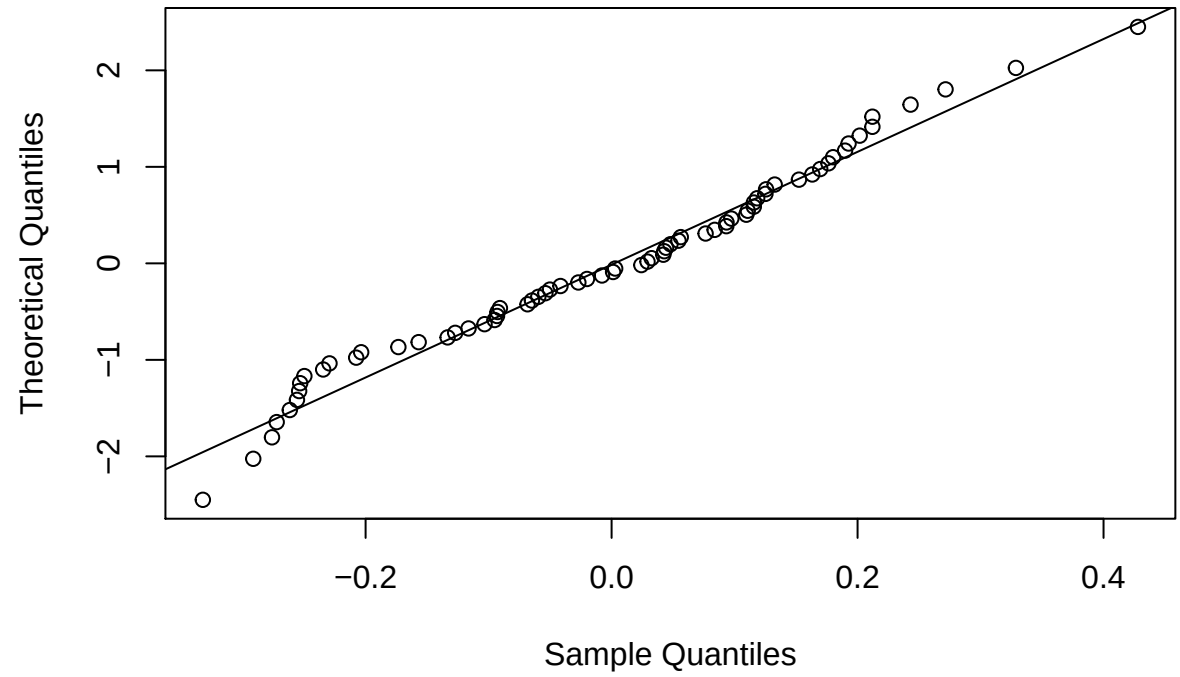
```
plot(x = shortleaf$lnDiam, y = shortleaf$lnVol,
     panel.last = lines(sort(shortleaf$lnDiam), fitted(model.3)[order(shortleaf$lnDiam)]))
```




```
plot(x = fitted(model.3), y = residuals(model.3),
     panel.last = abline(h = 0, lty = 2))
```




```
qqnorm(residuals(model.3), main = "", datax = TRUE)
qqline(residuals(model.3), datax = TRUE)
```




```
#exp(predict(model.3, interval = "confidence", newdata = data.frame (lnDiam = log(10))))
```