SIOB 296 Introduction to Programming with R

Eric Archer (eric.archer@noaa.gov)

Week 09: March 2, 2020

Distributions

Functions are provided to calculate the density, distribution function, quantile function, and generate random numbers from a variety of parametric distributions. They have similar forms, where if `<stat>` is the name of the distribution (e.g., `norm` for Normal, `unif` for Uniform, `binom` for Binomial), `d<stat>` gives the density or probability mass function (likelihood), `p<stat>` gives the probability distribution (cumulative distribution function), `q<stat>` gives the quantile function, and `r<stat>` generates random numbers. Here are examples of all for for the Normal distribution:

```
# The likelihood of five values in a Normal distribution with a
#  mean of 10 and a standard deviation of 2:
x <- c(5, 8, 10, 12, 15)
dnorm(x, mean = 10, sd = 2)
```

```
[1] 0.00876415 0.12098536 0.19947114 0.12098536 0.00876415
```

```
# Cumulative probability of same values:
pnorm(x, mean = 10, sd = 2)
```

```
[1] 0.006209665 0.158655254 0.500000000 0.841344746 0.993790335
```

```
# Quantiles:
p <- c(0.05, 0.25, 0.5, 0.75, 0.95)
qnorm(p, mean = 10, sd = 2)
```

```
[1]  6.710293  8.651020 10.000000 11.348980 13.289707
```

```
# Five random draws:
rnorm(5, mean = 10, sd = 2)
```

```
[1]  8.737345 11.786523  8.225297 10.049129 12.409299
```

The random number seed is set with `set.seed()`. Setting this value ensures that the same random number sequence will be repeated:

```
# repeat the same random 5 numbers
set.seed(1)
rnorm(5, mean = 10, sd = 2)
```

```
[1]  8.747092 10.367287  8.328743 13.190562 10.659016
```

```
set.seed(1)
rnorm(5, mean = 10, sd = 2)
```

```
[1]  8.747092 10.367287  8.328743 13.190562 10.659016
```

```
# choose a different random 5
rnorm(5, mean = 10, sd = 2)
```

```
[1]  8.359063 10.974858 11.476649 11.151563  9.389223
```

Statistical tests

There are several functions for standard statistical tests that all have similar outputs. The most common ones are `binom.test`, `chisq.test`, `kruskal.test`, `ks.test`, and `t.test`. As an example, we'll simulate two sets of length measurements and conduct a t-test to test for differences between their means.

```
# choose a number of individuals from each species
n.ind <- sample(30:300, 1)

# simulate drawing some lengths from a normal distribution
spp1 <- rnorm(n.ind, 10, 3)
spp2 <- rnorm(n.ind, 11, 3)

# test difference between means
spp.ttest <- t.test(spp1, spp2)
spp.ttest
```

```

    Welch Two Sample t-test

data:  spp1 and spp2
t = -1.5726, df = 210.03, p-value = 0.1173
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.3877880  0.1561224
sample estimates:
mean of x mean of y
 10.29191  10.90774
```

```
str(spp.ttest)
```

```
List of 10
 $ statistic  : Named num -1.57
  ..- attr(*, "names")= chr "t"
 $ parameter  : Named num 210
  ..- attr(*, "names")= chr "df"
 $ p.value    : num 0.117
 $ conf.int   : num [1:2] -1.388 0.156
  ..- attr(*, "conf.level")= num 0.95
 $ estimate   : Named num [1:2] 10.3 10.9
  ..- attr(*, "names")= chr [1:2] "mean of x" "mean of y"
 $ null.value : Named num 0
  ..- attr(*, "names")= chr "difference in means"
 $ stderr     : num 0.392
 $ alternative: chr "two.sided"
 $ method     : chr "Welch Two Sample t-test"
 $ data.name  : chr "spp1 and spp2"
 - attr(*, "class")= chr "htest"
```

Note that the result of the `t.test` function is a list with various results and information about the test conducted. Because the function returns a named list, if we are only interested in one element, we can extract it directly:

```
t.test(spp1, spp2)$p.value
```

```
[1] 0.1173081
```

Another example of a common test is a chi-squared test of differences among frequencies, run with the function `chisq.test`. Lets also simulate some data of iris species occurence using the `sample` function. We'll give different weights to the `prob` argument to vary the occurence of species in each plot.

```
# collect samples from plot 1
plot1.n <- sample(30:100, 1)
plot1 <- sample(levels(iris$Species), plot1.n, T, c(1, 2, 5))

# collect samples from plot 2
plot2.n <- sample(30:100, 1)
plot2 <- sample(levels(iris$Species), plot2.n, T, c(2, 1, 5))

# create a data.frame combining plot samples
occ.df <- rbind(
  cbind(spp = plot1, plot = 1),
  cbind(spp = plot2, plot = 2)
)

table(occ.df[, "spp"], occ.df[, "plot"])
```

```
             1  2
  setosa     5 15
  versicolor  7  4
  virginica  20 28
```

```
occ.chisq <- chisq.test(occ.df[, "spp"], occ.df[, "plot"])
```

```
Warning in chisq.test(occ.df[, "spp"], occ.df[, "plot"]): Chi-squared
approximation may be incorrect
```

```
occ.chisq
```

```
    Pearson's Chi-squared test

data:  occ.df[, "spp"] and occ.df[, "plot"]
X-squared = 4.4644, df = 2, p-value = 0.1073
```

```
str(occ.chisq)
```

```
List of 9
 $ statistic: Named num 4.46
  ..- attr(*, "names")= chr "X-squared"
 $ parameter: Named int 2
  ..- attr(*, "names")= chr "df"
 $ p.value  : num 0.107
 $ method   : chr "Pearson's Chi-squared test"
 $ data.name: chr "occ.df[, \"spp\"] and occ.df[, \"plot\"]"
 $ observed : 'table' int [1:3, 1:2] 5 7 20 15 4 28
  ..- attr(*, "dimnames")=List of 2
  .. ..$ occ.df[, "spp"] : chr [1:3] "setosa" "versicolor" "virginica"
  .. ..$ occ.df[, "plot"]: chr [1:2] "1" "2"
 $ expected : num [1:3, 1:2] 8.1 4.46 19.44 11.9 6.54 ...
  ..- attr(*, "dimnames")=List of 2
```

```
  .. ..$ occ.df[, "spp"] : chr [1:3] "setosa" "versicolor" "virginica"
  .. ..$ occ.df[, "plot"]: chr [1:2] "1" "2"
 $ residuals: 'table' num [1:3, 1:2] -1.09 1.205 0.126 0.899 -0.995 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ occ.df[, "spp"] : chr [1:3] "setosa" "versicolor" "virginica"
  .. ..$ occ.df[, "plot"]: chr [1:2] "1" "2"
 $ stdres   : 'table' num [1:3, 1:2] -1.635 1.684 0.261 1.635 -1.684 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ occ.df[, "spp"] : chr [1:3] "setosa" "versicolor" "virginica"
  .. ..$ occ.df[, "plot"]: chr [1:2] "1" "2"
 - attr(*, "class")= chr "htest"
```

The `chisq.test` function also has the ability to estimate significance via a bootstrap, which is selected by setting `simulate.p.value = TRUE`:

```
chisq.test(occ.df[, "spp"], occ.df[, "plot"], sim = TRUE)
```

```
    Pearson's Chi-squared test with simulated p-value (based on 2000
    replicates)

data:  occ.df[, "spp"] and occ.df[, "plot"]
X-squared = 4.4644, df = NA, p-value = 0.1259
```

We can directly calculate the Chi-squared statistic using some matrix algebra and a few summary functions. Recall that Chi-squared = sum((observed - expected)^2 / expected)

```
# observed frequencies
obs <- table(occ.df[, "spp"], occ.df[, "plot"])
obs
```

```
             1  2
  setosa     5 15
  versicolor  7  4
  virginica  20 28
```

```
# row sums and column sums
row.sums <- rowSums(obs)
row.sums
```

```
    setosa versicolor  virginica
        20         11         48
```

```
col.sums <- colSums(obs)
col.sums
```

```
 1  2
32 47
```

```
# expected frequencies are the matrix product of these two divided by the total
exp <- outer(row.sums, col.sums) / sum(freq)
```

```
Error in eval(expr, envir, enclos): object 'freq' not found
```

```
chi.squared <- sum((obs - exp) ^ 2 / exp)
```

```
Error in obs - exp: non-numeric argument to binary operator
```

```
chi.squared
```

```
Error in eval(expr, envir, enclos): object 'chi.squared' not found
```

```
# compared to value from chisq.test function:
occ.chisq$statistic
```

```
X-squared
 4.464363
```

```
# the p-value of this can be looked up from the chisq.distribution
1 - pchisq(chi.squared, df = 2)
```

```
Error in pchisq(chi.squared, df = 2): object 'chi.squared' not found
```

Formula

In R, models are usually based on `formula` objects. Formulae are constructed using the tilde (`~`) operator. The syntax is `y ~ x`, which is translated as `y` is a function of `x`. As an example, we can create a data frame of our simulated length data and run the t-test using a formula structure instead.

```
# create data frame
length.df <- data.frame(
  length = c(spp1, spp2),
  spp = rep(c(1, 2), each = n.ind)
)

# run t-test with formula based on columns in data frame
t.test(length ~ spp, data = length.df)
```

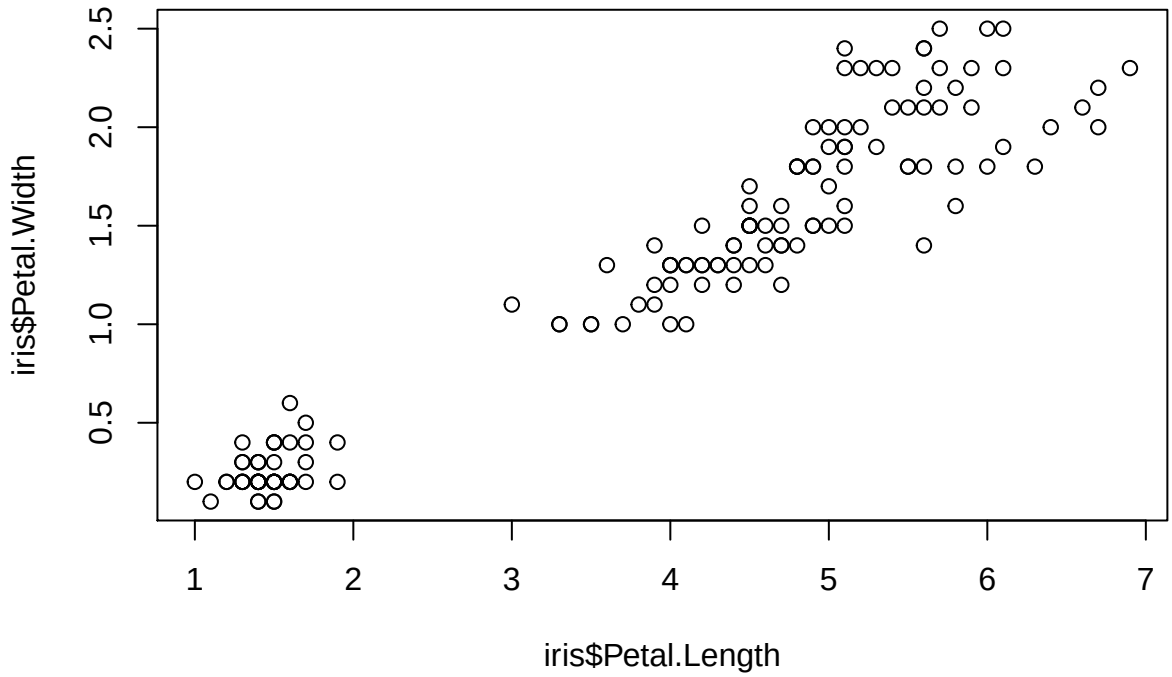
```
    Welch Two Sample t-test

data:  length by spp
t = -1.5726, df = 210.03, p-value = 0.1173
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.3877880  0.1561224
sample estimates:
mean in group 1 mean in group 2
       10.29191        10.90774
```

Linear models

We also use formula for linear and non-linear modelling. The standard function `lm` fits a linear model and returns the intercept and slope estimates as well as diagnostics of the fit. Below, we'll fit a model to estimate iris petal width from length.

```
plot(iris$Petal.Length, iris$Petal.Width)
```




```r
# Fit the model
petal.lm <- lm(Petal.Width ~ Petal.Length, data = iris)
# Here's a simple summary of the fit
petal.lm
```

```
Call:
lm(formula = Petal.Width ~ Petal.Length, data = iris)

Coefficients:
 (Intercept)  Petal.Length
     -0.3631        0.4158
```

```r
# Here are all of the elements in the fitted object:
str(petal.lm)
```

```
List of 12
 $ coefficients : Named num [1:2] -0.363 0.416
  ..- attr(*, "names")= chr [1:2] "(Intercept)" "Petal.Length"
 $ residuals    : Named num [1:150] -0.019 -0.019 0.0226 -0.0606 -0.019 ...
  ..- attr(*, "names")= chr [1:150] "1" "2" "3" "4" ...
 $ effects      : Named num [1:150] -14.6888 8.9588 0.0257 -0.0576 -0.0159 ...
  ..- attr(*, "names")= chr [1:150] "(Intercept)" "Petal.Length" "" "" ...
 $ rank         : int 2
 $ fitted.values: Named num [1:150] 0.219 0.219 0.177 0.261 0.219 ...
  ..- attr(*, "names")= chr [1:150] "1" "2" "3" "4" ...
 $ assign       : int [1:2] 0 1
 $ qr           :List of 5
  ..$ qr   : num [1:150, 1:2] -12.2474 0.0816 0.0816 0.0816 0.0816 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:150] "1" "2" "3" "4" ...
  .. .. ..$ : chr [1:2] "(Intercept)" "Petal.Length"
  .. ..- attr(*, "assign")= int [1:2] 0 1
```

```
  ..$ qraux: num [1:2] 1.08 1.1
  ..$ pivot: int [1:2] 1 2
  ..$ tol  : num 1e-07
  ..$ rank : int 2
  ..- attr(*, "class")= chr "qr"
 $ df.residual  : int 148
 $ xlevels      : Named list()
 $ call         : language lm(formula = Petal.Width ~ Petal.Length, data = iris)
 $ terms        :Classes 'terms', 'formula'  language Petal.Width ~ Petal.Length
  .. ..- attr(*, "variables")= language list(Petal.Width, Petal.Length)
  .. ..- attr(*, "factors")= int [1:2, 1] 0 1
  .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. ..$ : chr [1:2] "Petal.Width" "Petal.Length"
  .. .. .. ..$ : chr "Petal.Length"
  .. ..- attr(*, "term.labels")= chr "Petal.Length"
  .. ..- attr(*, "order")= int 1
  .. ..- attr(*, "intercept")= int 1
  .. ..- attr(*, "response")= int 1
  .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
  .. ..- attr(*, "predvars")= language list(Petal.Width, Petal.Length)
  .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
  .. .. ..- attr(*, "names")= chr [1:2] "Petal.Width" "Petal.Length"
 $ model        :'data.frame':  150 obs. of  2 variables:
  ..$ Petal.Width : num [1:150] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
  ..$ Petal.Length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
  ..- attr(*, "terms")=Classes 'terms', 'formula'  language Petal.Width ~ Petal.Length
  .. .. ..- attr(*, "variables")= language list(Petal.Width, Petal.Length)
  .. .. ..- attr(*, "factors")= int [1:2, 1] 0 1
  .. .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. .. ..$ : chr [1:2] "Petal.Width" "Petal.Length"
  .. .. .. .. ..$ : chr "Petal.Length"
  .. .. ..- attr(*, "term.labels")= chr "Petal.Length"
  .. .. ..- attr(*, "order")= int 1
  .. .. ..- attr(*, "intercept")= int 1
  .. .. ..- attr(*, "response")= int 1
  .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
  .. .. ..- attr(*, "predvars")= language list(Petal.Width, Petal.Length)
  .. .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
  .. .. .. ..- attr(*, "names")= chr [1:2] "Petal.Width" "Petal.Length"
 - attr(*, "class")= chr "lm"
```

Elements can be extracted from this list by name. For example, the estimated coefficients are stored in the `$coefficients` element:

```
petal.lm$coefficients
```

```
 (Intercept) Petal.Length
  -0.3630755    0.4157554
```

However, there are a set of functions for extracting common elements from model fits. An example is the `coef()` function, which will also extract the coefficients:

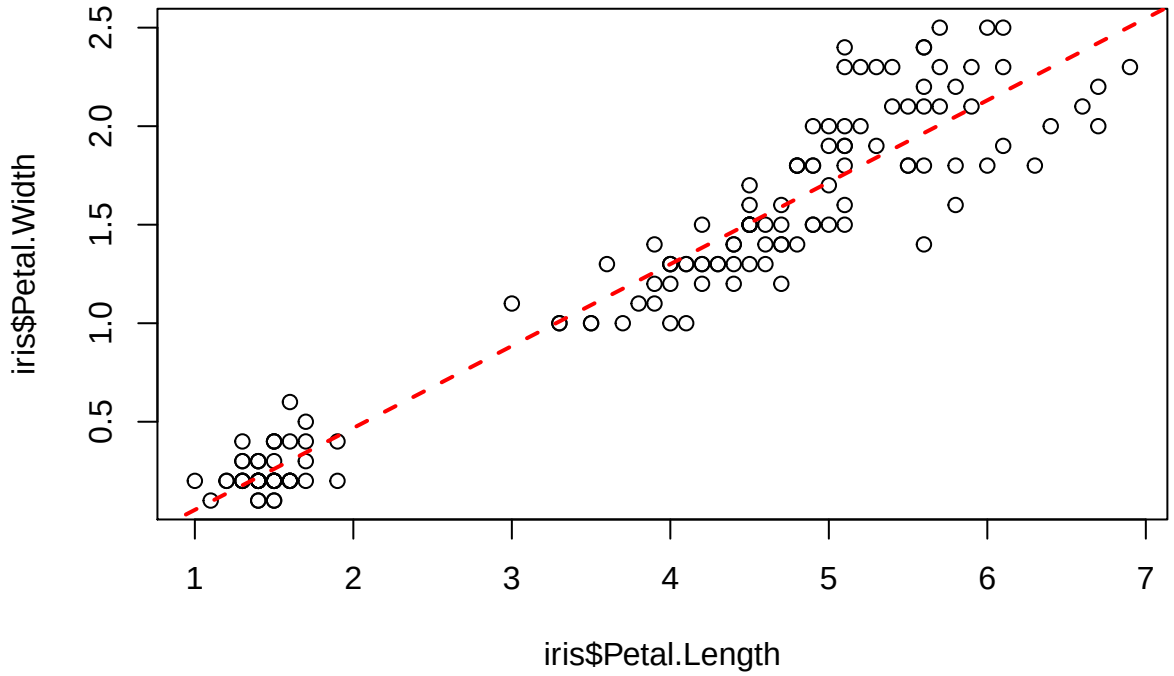
```
coef(petal.lm)
```

```
 (Intercept) Petal.Length
  -0.3630755    0.4157554
```

Others are `residuals` and `fitted.values`, which will often work with model objects from other routines such as `glm` and `nls`.

We can use the `abline` function to plot the estimated fit over the data:

```
plot(iris$Petal.Length, iris$Petal.Width)
abline(petal.lm, col = "red", lwd = 2, lty = "dashed")
```



More detail about the fit can be extracted with the `summary` function. In particular, we can see a summary of the residuals to inspect normality of the errors, as well as the standard errors and p-values for tests of significant deviation of the estimated parameters from zero:

```
lm.smry <- summary(petal.lm)
print(lm.smry)
```

```
Call:
lm(formula = Petal.Width ~ Petal.Length, data = iris)

Residuals:
     Min       1Q   Median       3Q      Max
-0.56515 -0.12358 -0.01898  0.13288  0.64272

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  -0.363076   0.039762  -9.131  4.7e-16 ***
Petal.Length  0.415755   0.009582  43.387  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2065 on 148 degrees of freedom
Multiple R-squared:  0.9271,    Adjusted R-squared:  0.9266
F-statistic:  1882 on 1 and 148 DF,  p-value: < 2.2e-16
```

```
str(lm.smry)
```

```
List of 11
 $ call         : language lm(formula = Petal.Width ~ Petal.Length, data = iris)
 $ terms        :Classes 'terms', 'formula'  language Petal.Width ~ Petal.Length
  .. ..- attr(*, "variables")= language list(Petal.Width, Petal.Length)
  .. ..- attr(*, "factors")= int [1:2, 1] 0 1
  .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. ..$ : chr [1:2] "Petal.Width" "Petal.Length"
  .. .. .. ..$ : chr "Petal.Length"
  .. ..- attr(*, "term.labels")= chr "Petal.Length"
  .. ..- attr(*, "order")= int 1
  .. ..- attr(*, "intercept")= int 1
  .. ..- attr(*, "response")= int 1
  .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
  .. ..- attr(*, "predvars")= language list(Petal.Width, Petal.Length)
  .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
  .. .. ..- attr(*, "names")= chr [1:2] "Petal.Width" "Petal.Length"
 $ residuals    : Named num [1:150] -0.019 -0.019 0.0226 -0.0606 -0.019 ...
  ..- attr(*, "names")= chr [1:150] "1" "2" "3" "4" ...
 $ coefficients : num [1:2, 1:4] -0.36308 0.41576 0.03976 0.00958 -9.13122 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:2] "(Intercept)" "Petal.Length"
  .. ..$ : chr [1:4] "Estimate" "Std. Error" "t value" "Pr(>|t|)"
 $ aliased      : Named logi [1:2] FALSE FALSE
  ..- attr(*, "names")= chr [1:2] "(Intercept)" "Petal.Length"
 $ sigma        : num 0.206
 $ df           : int [1:3] 2 148 2
 $ r.squared    : num 0.927
 $ adj.r.squared: num 0.927
 $ fstatistic   : Named num [1:3] 1882 1 148
  ..- attr(*, "names")= chr [1:3] "value" "numdf" "dendf"
 $ cov.unscaled : num [1:2, 1:2] 0.03708 -0.00809 -0.00809 0.00215
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:2] "(Intercept)" "Petal.Length"
  .. ..$ : chr [1:2] "(Intercept)" "Petal.Length"
 - attr(*, "class")= chr "summary.lm"
```

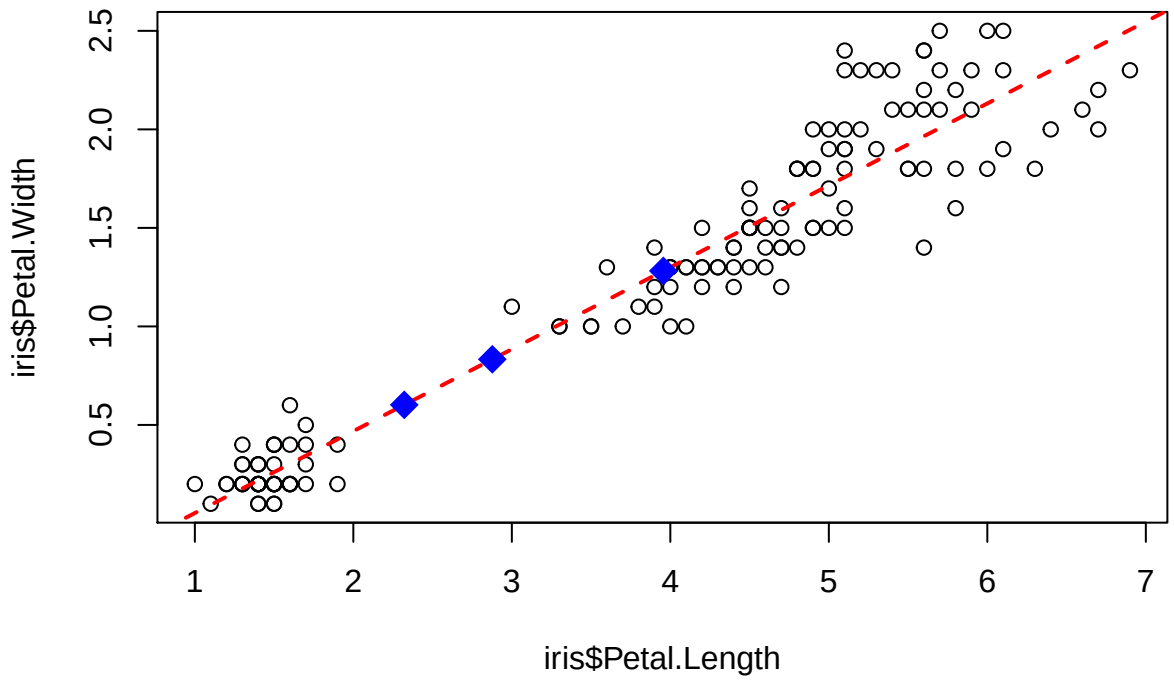
We can use the model fit object to pedict new data too. We just need a data frame of the new values with column names of the independent values the same as those in the original model:

```
new.petals <- data.frame(Petal.Length = runif(3, 2, 4))
new.petal.pred <- predict(petal.lm, new.petals)
new.petal.pred
```

```
        1         2         3
1.2815940 0.8330151 0.6019327
```

```
plot(iris$Petal.Length, iris$Petal.Width)
abline(petal.lm, col = "red", lwd = 2, lty = "dashed")
points(
  new.petals$Petal.Length,
  new.petal.pred,
  pch = 18,
  col = "blue",
```

```
  cex = 2
)
```

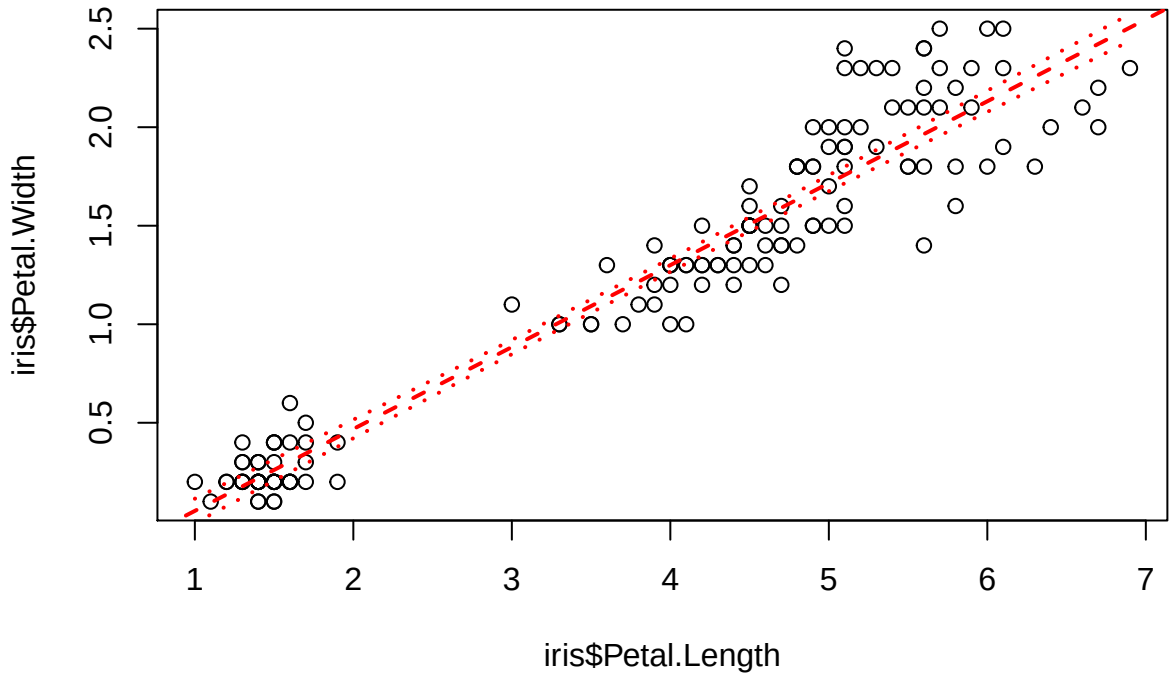


Predictions can also include confidence intervals:

```
# a data frame of 100 evenly spaced points from the max to min petal length
ci.df <- data.frame(
  Petal.Length = seq(min(iris$Petal.Length), max(iris$Petal.Length), length.out = 100)
)
new.petal.pred <- predict(petal.lm, ci.df, interval = "confidence")
head(new.petal.pred)
```

```
         fit          lwr       upr
1 0.05267990 -0.009267579 0.1146274
2 0.07745724  0.016458153 0.1384563
3 0.10223458  0.042177655 0.1622915
4 0.12701192  0.067890631 0.1861332
5 0.15178927  0.093596764 0.2099818
6 0.17656661  0.119295722 0.2338375
```

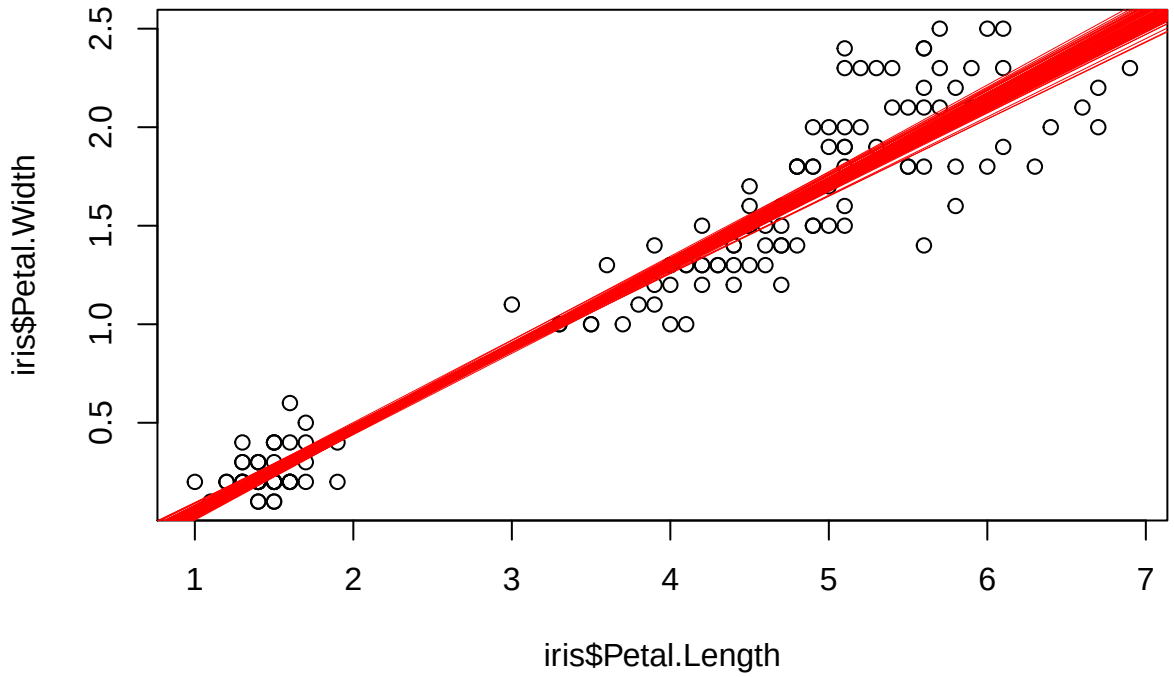
```
# plot points, fit, and confidence intervals
plot(iris$Petal.Length, iris$Petal.Width)
abline(petal.lm, col = "red", lwd = 2, lty = "dashed")
lines(ci.df$Petal.Length, new.petal.pred[, "lwr"], col = "red", lwd = 2, lty = "dotted")
lines(ci.df$Petal.Length, new.petal.pred[, "upr"], col = "red", lwd = 2, lty = "dotted")
```



Is that really the confidence interval? Most of the points lie outside of it! Yes, this is the confidence interval of the *fit*, not the variance around the fit (the residuals). We can prove that by bootstrapping the data and showing the distribution of the bootstrapped fits:

```
boot.lm <- lapply(1:100, function(i) {
  lm(
    Petal.Width ~ Petal.Length,
    data = iris[sample(1:nrow(iris), nrow(iris), T), ]
  )
})

plot(iris$Petal.Length, iris$Petal.Width)
for(x in boot.lm) abline(x, col = "red", lwd = 0.5)
```



Models can be built on categorical predictors as well. In this example we test whether or not petal length differs among species:

```
length.lm <- lm(Petal.Length ~ Species, iris)
summary(length.lm)
```

```
Call:
lm(formula = Petal.Length ~ Species, data = iris)

Residuals:
   Min     1Q Median     3Q    Max
-1.260 -0.258  0.038  0.240  1.348

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)       1.46200    0.06086   24.02   <2e-16 ***
Speciesversicolor 2.79800    0.08607   32.51   <2e-16 ***
Speciesvirginica  4.09000    0.08607   47.52   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4303 on 147 degrees of freedom
Multiple R-squared:  0.9414,    Adjusted R-squared:  0.9406
F-statistic:  1180 on 2 and 147 DF,  p-value: < 2.2e-16
```

Note that the results list the dummy variables representing the levels of the categorical station predictor. Their estimated effects are expressed as being relative to the first level.

ANOVA

An analysis of variance (ANOVA) is related to the multi-category linear model and gets specified with the same formula using the `aov` function:

```r
length.aov <- aov(Petal.Length ~ Species, iris)
summary(length.aov)
```

```
             Df Sum Sq Mean Sq F value Pr(>F)
Species       2  437.1  218.55    1180 <2e-16 ***
Residuals   147   27.2    0.19
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
str(length.aov)
```

```
List of 13
 $ coefficients : Named num [1:3] 1.46 2.8 4.09
  ..- attr(*, "names")= chr [1:3] "(Intercept)" "Speciesversicolor" "Speciesvirginica"
 $ residuals    : Named num [1:150] -0.062 -0.062 -0.162 0.038 -0.062 ...
  ..- attr(*, "names")= chr [1:150] "1" "2" "3" "4" ...
 $ effects      : Named num [1:150] -46.026 4.347 20.45 0.058 -0.042 ...
  ..- attr(*, "names")= chr [1:150] "(Intercept)" "Speciesversicolor" "Speciesvirginica" "" ...
 $ rank         : int 3
 $ fitted.values: Named num [1:150] 1.46 1.46 1.46 1.46 1.46 ...
  ..- attr(*, "names")= chr [1:150] "1" "2" "3" "4" ...
 $ assign       : int [1:3] 0 1 1
 $ qr           :List of 5
  ..$ qr   : num [1:150, 1:3] -12.2474 0.0816 0.0816 0.0816 0.0816 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:150] "1" "2" "3" "4" ...
  .. .. ..$ : chr [1:3] "(Intercept)" "Speciesversicolor" "Speciesvirginica"
  .. ..- attr(*, "assign")= int [1:3] 0 1 1
  .. ..- attr(*, "contrasts")=List of 1
  .. .. ..$ Species: chr "contr.treatment"
  ..$ qraux: num [1:3] 1.08 1.05 1.09
  ..$ pivot: int [1:3] 1 2 3
  ..$ tol  : num 1e-07
  ..$ rank : int 3
  ..- attr(*, "class")= chr "qr"
 $ df.residual  : int 147
 $ contrasts    :List of 1
  ..$ Species: chr "contr.treatment"
 $ xlevels      :List of 1
  ..$ Species: chr [1:3] "setosa" "versicolor" "virginica"
 $ call         : language aov(formula = Petal.Length ~ Species, data = iris)
 $ terms        :Classes 'terms', 'formula'  language Petal.Length ~ Species
  .. ..- attr(*, "variables")= language list(Petal.Length, Species)
  .. ..- attr(*, "factors")= int [1:2, 1] 0 1
  .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. ..$ : chr [1:2] "Petal.Length" "Species"
  .. .. .. ..$ : chr "Species"
  .. ..- attr(*, "term.labels")= chr "Species"
  .. ..- attr(*, "order")= int 1
  .. ..- attr(*, "intercept")= int 1
  .. ..- attr(*, "response")= int 1
  .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
  .. ..- attr(*, "predvars")= language list(Petal.Length, Species)
  .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "factor"
```

```
  .. .. ..- attr(*, "names")= chr [1:2] "Petal.Length" "Species"
 $ model        :'data.frame':  150 obs. of  2 variables:
  ..$ Petal.Length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
  ..$ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
  ..- attr(*, "terms")=Classes 'terms', 'formula'  language Petal.Length ~ Species
  .. .. ..- attr(*, "variables")= language list(Petal.Length, Species)
  .. .. ..- attr(*, "factors")= int [1:2, 1] 0 1
  .. .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. .. ..$ : chr [1:2] "Petal.Length" "Species"
  .. .. .. .. ..$ : chr "Species"
  .. .. ..- attr(*, "term.labels")= chr "Species"
  .. .. ..- attr(*, "order")= int 1
  .. .. ..- attr(*, "intercept")= int 1
  .. .. ..- attr(*, "response")= int 1
  .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
  .. .. ..- attr(*, "predvars")= language list(Petal.Length, Species)
  .. .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "factor"
  .. .. .. ..- attr(*, "names")= chr [1:2] "Petal.Length" "Species"
 - attr(*, "class")= chr [1:2] "aov" "lm"
```

The analysis of variance table can also be computed from an `lm` object using `anova`:

```
anova(length.lm)
```

```
Analysis of Variance Table

Response: Petal.Length
           Df Sum Sq Mean Sq F value    Pr(>F)
Species     2 437.10 218.551  1180.2 < 2.2e-16 ***
Residuals 147  27.22   0.185
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Differences between levels of the predictor can be tested with the `TukeyHSD()` function:

```
TukeyHSD(length.aov)
```

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = Petal.Length ~ Species, data = iris)

$Species
                      diff     lwr     upr p adj
versicolor-setosa    2.798 2.59422 3.00178     0
virginica-setosa     4.090 3.88622 4.29378     0
virginica-versicolor 1.292 1.08822 1.49578     0
```

Permutation tests

If we don't want to rely on canned parametric assessments of differences among groups, we can construct the null distributions by randomly permuting group assignments and comparing the distribution of the test statistic with the observed value. As an example here's the observed t-statistic for our simulated length measurements from the beginning:

```
spp.ttest$statistic
```

```
        t
-1.572637
```

The t-statistic for one random permutation of species can be calculated by permuting the species column of the data frame and running the t-test again:

```
perm.spp <- sample(length.df$spp)
t.test(length.df$length ~ perm.spp)$statistic
```

```
        t
0.2401946
```

We want to run this a number of times and create a vector of the permuted t-statistics. Let's use `sapply` to create a matrix of permutations and then `apply` to walk through that matrix and run the t.test for each permutation using an anonymous function:

```
# matrix of permuted species designations
perm.spp.mat <- sapply(1:1000, function(i) sample(length.df$spp))
str(perm.spp.mat)
```

```
 num [1:216, 1:1000] 1 2 1 2 2 2 1 2 2 2 ...
```

```
# vector of t-statistics for each permutation
perm.t <- apply(perm.spp.mat, 2, function(spp) {
  t.test(length.df$length ~ spp)$statistic
})
str(perm.t)
```

```
 num [1:1000] -0.89599 0.27409 0.71318 0.00269 1.63731 ...
```

If we don't want to save the permutations, we can both permute and run the t-test with `sapply`:

```
perm.t <- sapply(1:1000, function(i) {
  perm.spp <- sample(length.df$spp)
  t.test(length.df$length ~ perm.spp)$statistic
})
str(perm.t)
```

```
 Named num [1:1000] -0.76627 -0.31312 1.53265 0.08491 -0.00443 ...
 - attr(*, "names")= chr [1:1000] "t" "t" "t" "t" ...
```

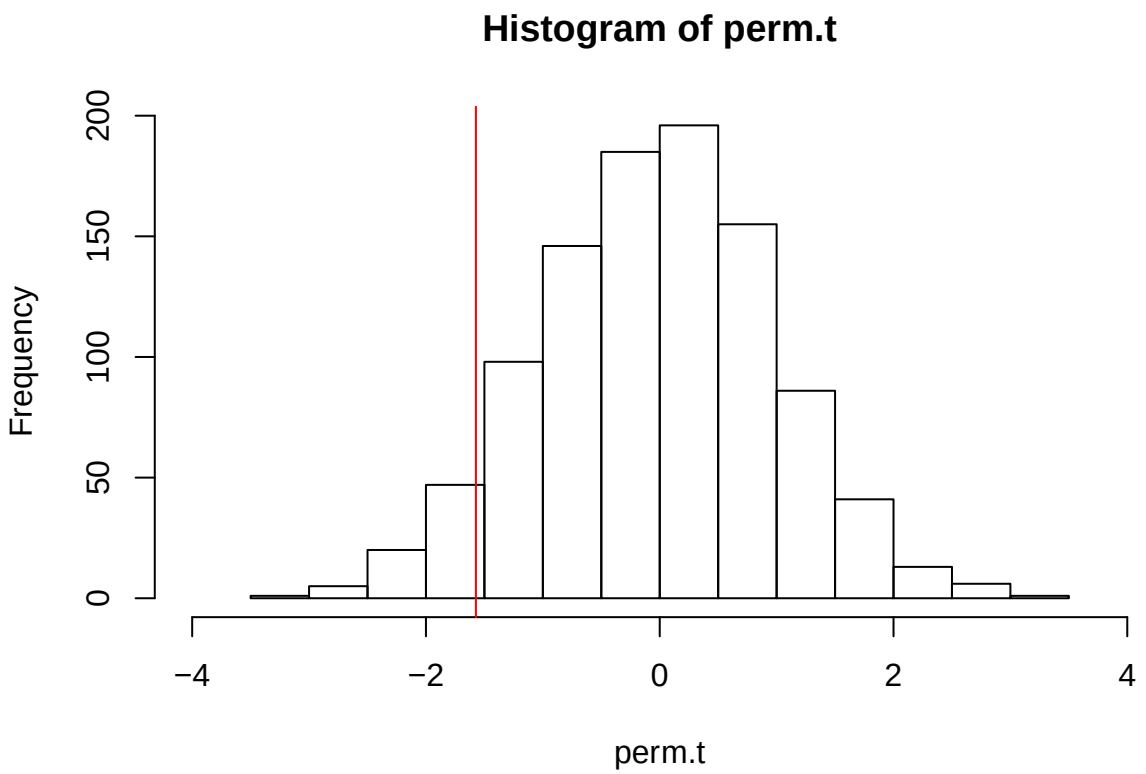
Now let's summarize the results and compare the distribution to the observed value.

```
summary(perm.t)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-3.07313 -0.71061 -0.00845 -0.02241  0.64964  3.35464
```

```
obs.t <- spp.ttest$statistic

# a histogram of the permutation t-distribution
hist(perm.t, xlim = range(pretty(c(perm.t, obs.t))))
abline(v = obs.t, col = "red")
```




```
# what percentage of the distribution is >= the observed?
mean(perm.t >= obs.t)
```

```
[1] 0.937
```

```
# compared to t-test p-value:
spp.ttest$p.value
```

```
[1] 0.1173081
```

A more proper permutation t-test is to create a null distribution of the value we're interested in - the difference betwee means, rather than the distribution of the test statistic. Let's make our lives easier by creating a function that returns the difference among means given a data frame like `length.df`, then running the permutation test with this function.

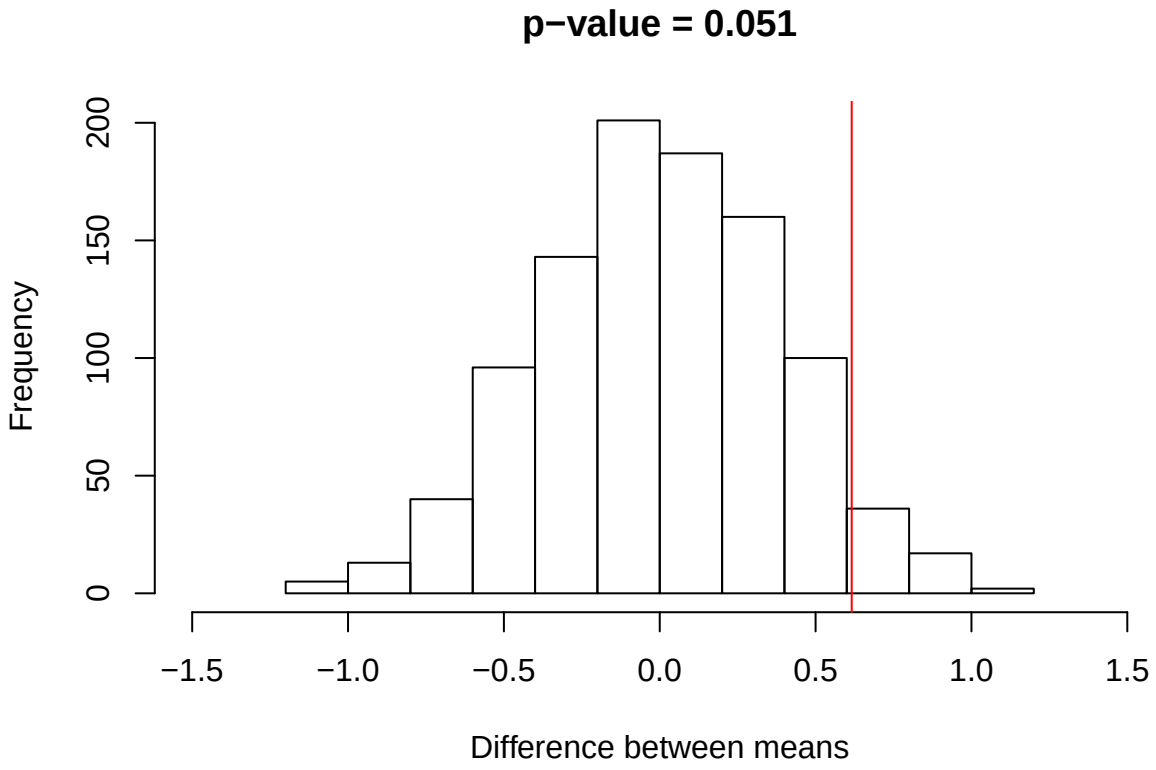
```
meanDiff <- function(x) {
  # calculate mean length for both groups
  spp.mean <- tapply(x$length, x$spp, mean)
  # return difference
  diff(spp.mean)
}

# the observed difference
obs.diff <- meanDiff(length.df)

# collect vector of differences from permutations
perm.df <- length.df # a copy that we'll be modifying
perm.diff <- sapply(1:1000, function(i) {
  perm.df$spp <- sample(perm.df$spp)
  meanDiff(perm.df)
})
```

```r
# calculate p-value based on sign of observed difference
p.value <- if(obs.diff < 0) {
  mean(perm.diff <= obs.diff)
} else {
  mean(perm.diff >= obs.diff)
}

# show relationship of difference to null distribution
hist(
  perm.diff,
  xlab = "Difference between means",
  xlim = range(pretty(c(perm.diff, obs.diff))),
  main = paste("p-value =", round(p.value, 3))
)
abline(v = obs.diff, col = "red")
```



Pairwise analyses

One way to do pairwise analyses in R is to use the function `combn` which will generate all possible combinations of length `m` of elements of a vector. For example, here's all possible combinations of 2 species:

```r
combn(levels(iris$Species), 2)
```

```
     [,1]         [,2]        [,3]
[1,] "setosa"     "setosa"    "versicolor"
[2,] "versicolor" "virginica" "virginica"
```

The `combn` function will also supply each combination to a function of your choosing, for example, we can paste the names together:

```
combn(levels(iris$Species), 2, paste, collapse = " v. ")
```

```
[1] "setosa v. versicolor"    "setosa v. virginica"
[3] "versicolor v. virginica"
```

. . . or we can use the value delivered by the function to extract some data and calculate a value between each pair, like the difference in mean petal length:

```
combn(levels(iris$Species), 2, function(x) {
  x.df <- droplevels(iris[iris$Species %in% x, ])
  petal.length.means <- tapply(x.df$Petal.Length, x.df$Species, mean)
  diff(petal.length.means)
})
```

```
[1] 2.798 4.090 1.292
```

Non-linear models

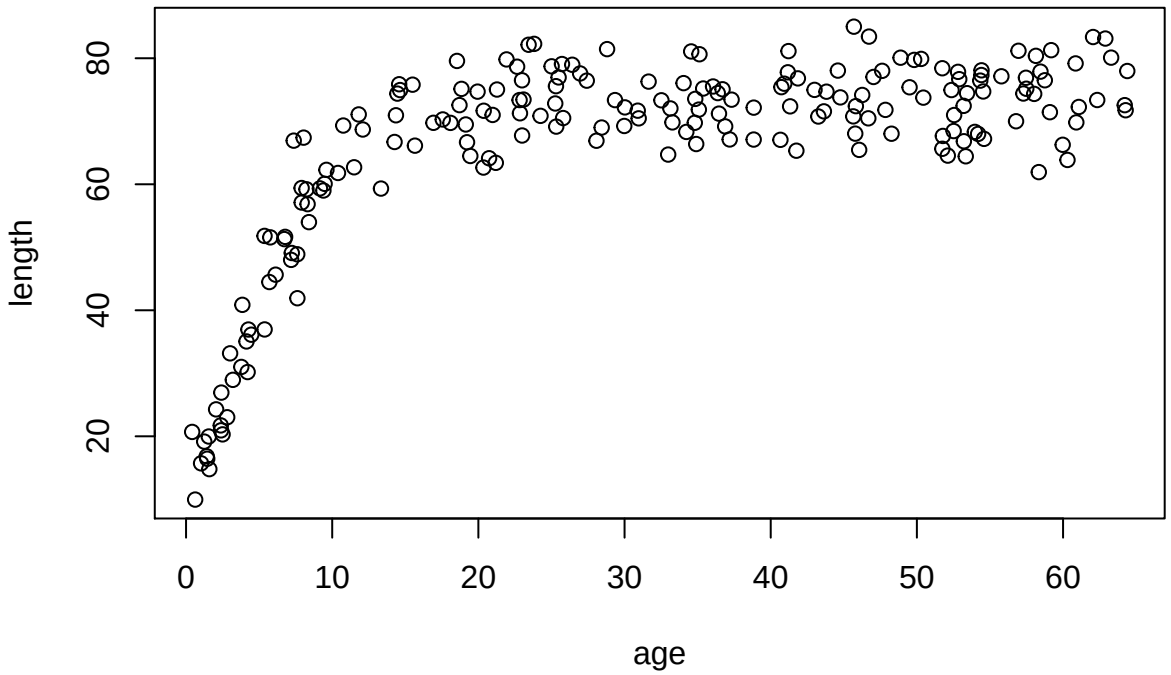
To illustrate non-linear model fitting, we'll first create a function to simulate growth data (length ~ age) based on a Gompertz curve. The Gompertz function is $length = L_0 \cdot e^{k(1-e^{-g \cdot age})}$, where L_0 is the length at birth (LAB). Here's the function to create simulated growth data:

```
# age.range - a two element vector giving the minimum and maximum ages
# lab - the length at birth
# k, g - displacement and rate parameters
# std.dev - standard deviation for the error term
# sample.size - number of points to simulate

sim.growth.func <- function(age.range, lab, k, g, std.dev, sample.size) {
  # Generate some random ages between min and max of age.range
  ages <- runif(sample.size, age.range[1], age.range[2])
  # Calculate the expected length for those ages from the Gompertz equation
  expected.length <- lab * exp(k * (1 - exp(-g * ages)))
  # Add some error to the lengths and return the named array
  length.err <- rnorm(sample.size, 0, std.dev)
  as.data.frame(cbind(age = ages, length = expected.length + length.err))
}
```

With this function, we can now simulate some growth data:

```
growth.df <- sim.growth.func(
  age.range = c(0, 65),
  lab = 10,
  k = 2,
  g = 0.25,
  std.dev = 5,
  sample.size = 200
)
plot(length ~ age, growth.df)
```



Now let's use nonlinear least squares to estimate the parameters from this simulated data. We can do that with the `nls` function, which behaves very similarly to `lm`. The main difference is that we need to supply initial values, which are specified in the third argument, `start`. These values should be chosen carefully so as to ensure convergence.

```
gr.form <- length ~ lab * exp(k * (1 - exp(-g * age)))
# starting values for k and g are too far off for default number of iterations
gr.nls <- nls(gr.form, growth.df, start = c(lab = 15, k = 10, g = 10))
```

```
Error in nls(gr.form, growth.df, start = c(lab = 15, k = 10, g = 10)): singular gradient
```

```
# this should work
gr.nls <- nls(gr.form, growth.df, start = c(lab = 15, k = 1, g = 0.5))
print(gr.nls)
```

```
Nonlinear regression model
  model: length ~ lab * exp(k * (1 - exp(-g * age)))
   data: growth.df
  lab     k     g
9.494 2.048 0.253
 residual sum-of-squares: 4950

Number of iterations to convergence: 6
Achieved convergence tolerance: 1.046e-06
```

...and here are the estimated coefficients:

```
gr.coef <- coef(gr.nls)
gr.coef
```

```
      lab         k         g
9.4936600 2.0477619 0.2530218
```

We'll plot the fitted curve:

```r
grow.fit <- data.frame(
  age = seq(
    min(growth.df$age),
    max(growth.df$age),
    length.out = 1000
  )
)
grow.fit$length <- predict(gr.nls, grow.fit)
plot(length ~ age, growth.df)
lines(grow.fit$age, grow.fit$length, col = "red", lwd = 2, lty = "dashed")
```