# SIOB 296 Introduction to Programming with R

*Eric Archer (eric.archer@noaa.gov)*

*Week 7 (May 16, 2017) - Graphics*

---

## Reading

*The Book of R*:
Chapter 18 - Hypothesis testing
Chapter 20 - Simple linear regression

*The Art of R*:
Chapter 8.2 - Functions for statistical distributions
Chapter 8.6 - Simulation programming in R

### 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]  9.900195 10.892659 10.708232 12.521264 10.287962
```

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
```

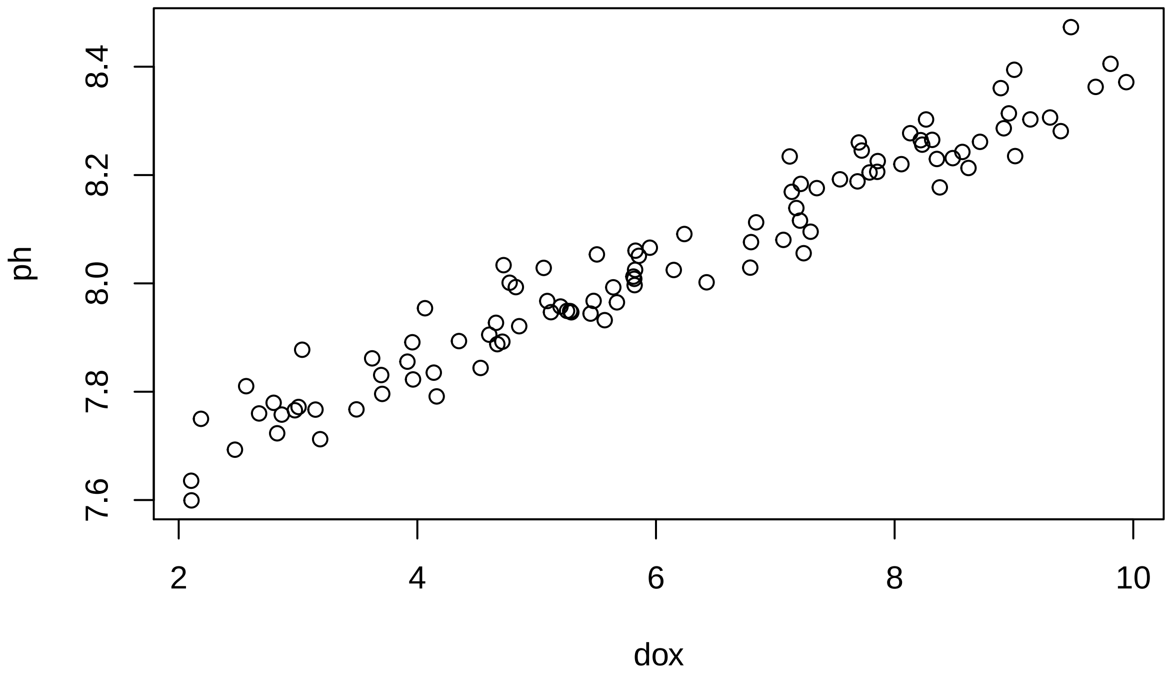
```
# choose a differnt random 5
rnorm(5, mean = 10, sd = 2)
```

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

### Simulated data

We can use these random number generators to simulate a linear relationship between dissolved oxygen and ph that we can then model:

```
dox <- runif(100, 2, 10)
ph <- 7.5 + (dox * 0.09) + rnorm(100, 0, 0.05)
plot(dox, ph)
```



### Linear models

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`. Here's an example for our pH and dissolved oxygen simulated data:

```
pd.form <- ph ~ dox
str(pd.form)
```

```
Class 'formula'  language ph ~ dox
  ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
```

The function `lm` fits a linear model to a formula and returns the intercept and slope estimates as well as diagnostics of the fit:

```r
pd.lm <- lm(ph ~ dox)
# Here's a simple summary of the fit
print(pd.lm)
```

```

Call:
lm(formula = ph ~ dox)

Coefficients:
(Intercept)          dox
    7.49528      0.09031
```

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

```
List of 12
 $ coefficients : Named num [1:2] 7.4953 0.0903
  ..- attr(*, "names")= chr [1:2] "(Intercept)" "dox"
 $ residuals    : Named num [1:100] 0.12187 0.00161 0.03698 0.00519 -0.03372 ...
  ..- attr(*, "names")= chr [1:100] "1" "2" "3" "4" ...
 $ effects      : Named num [1:100] -80.40688 1.93363 0.02556 -0.00457 -0.04393 ...
  ..- attr(*, "names")= chr [1:100] "(Intercept)" "dox" "" "" ...
 $ rank         : int 2
 $ fitted.values: Named num [1:100] 8.35 7.83 8.15 7.77 7.87 ...
  ..- attr(*, "names")= chr [1:100] "1" "2" "3" "4" ...
 $ assign       : int [1:2] 0 1
 $ qr           :List of 5
  ..$ qr   : num [1:100, 1:2] -10 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:100] "1" "2" "3" "4" ...
  .. .. ..$ : chr [1:2] "(Intercept)" "dox"
  .. ..- attr(*, "assign")= int [1:2] 0 1
  ..$ qraux: num [1:2] 1.1 1.12
  ..$ pivot: int [1:2] 1 2
  ..$ tol  : num 1e-07
  ..$ rank : int 2
  ..- attr(*, "class")= chr "qr"
 $ df.residual  : int 98
 $ xlevels      : Named list()
 $ call         : language lm(formula = ph ~ dox)
 $ terms        :Classes 'terms', 'formula'  language ph ~ dox
  .. ..- attr(*, "variables")= language list(ph, dox)
  .. ..- attr(*, "factors")= int [1:2, 1] 0 1
  .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. ..$ : chr [1:2] "ph" "dox"
  .. .. .. ..$ : chr "dox"
  .. ..- attr(*, "term.labels")= chr "dox"
  .. ..- attr(*, "order")= int 1
  .. ..- attr(*, "intercept")= int 1
  .. ..- attr(*, "response")= int 1
  .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
  .. ..- attr(*, "predvars")= language list(ph, dox)
  .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
  .. .. ..- attr(*, "names")= chr [1:2] "ph" "dox"
 $ model        :'data.frame':  100 obs. of  2 variables:
  ..$ ph : num [1:100] 8.47 7.83 8.18 7.77 7.84 ...
  ..$ dox: num [1:100] 9.48 3.7 7.21 3 4.14 ...
  ..- attr(*, "terms")=Classes 'terms', 'formula'  language ph ~ dox
  .. .. ..- attr(*, "variables")= language list(ph, dox)
  .. .. ..- attr(*, "factors")= int [1:2, 1] 0 1
  .. .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. .. ..$ : chr [1:2] "ph" "dox"
  .. .. .. .. ..$ : chr "dox"
  .. .. ..- attr(*, "term.labels")= chr "dox"
  .. .. ..- attr(*, "order")= int 1
  .. .. ..- attr(*, "intercept")= int 1
  .. .. ..- attr(*, "response")= int 1
  .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
  .. .. ..- attr(*, "predvars")= language list(ph, dox)
  .. .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
  .. .. .. ..- attr(*, "names")= chr [1:2] "ph" "dox"
 - attr(*, "class")= chr "lm"
```

The estimated coefficients are stored in the `$coefficients` element:

```
pd.lm$coefficients
```

```
(Intercept)         dox
  7.4952814   0.0903088
```

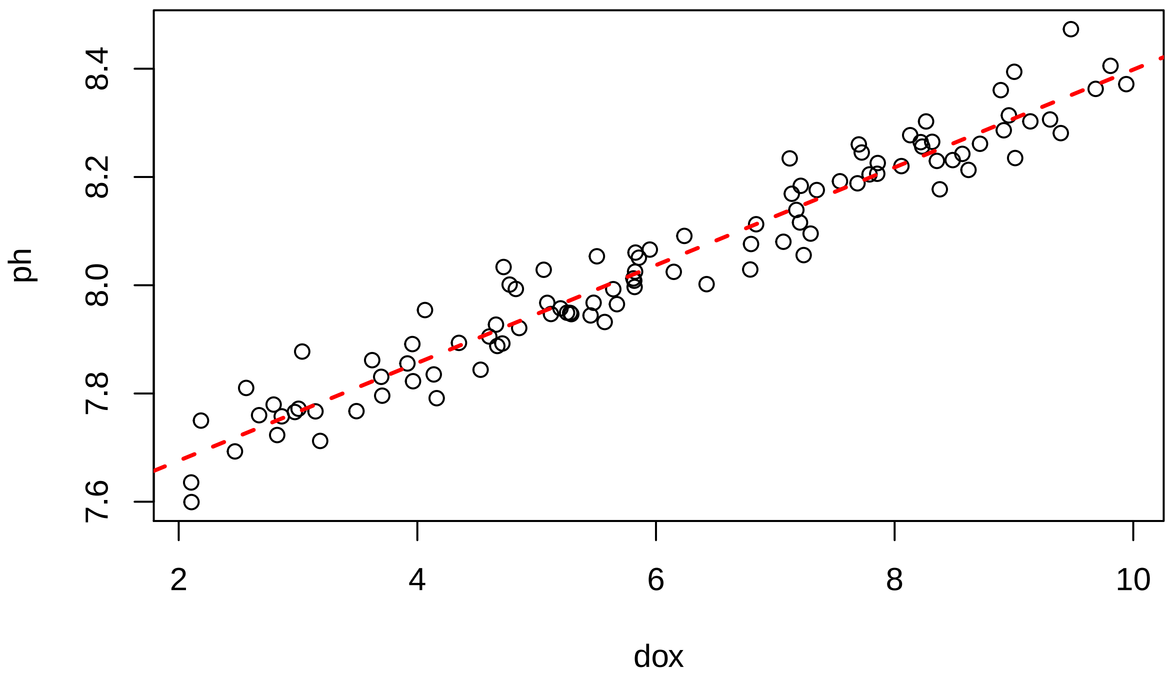
They can also be obtained with a call to the `coef()` function:

```
coef(pd.lm)
```

```
(Intercept)         dox
  7.4952814   0.0903088
```

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

```
plot(dox, ph)
abline(pd.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:

```
x <- summary(pd.lm)
print(x)
```

```

Call:
lm(formula = ph ~ dox)

Residuals:
      Min        1Q    Median        3Q       Max 
-0.093234 -0.029194 -0.004968  0.027440  0.121873 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 7.495281   0.014262  525.53   <2e-16 ***
dox         0.090309   0.002226   40.57   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.04766 on 98 degrees of freedom
Multiple R-squared:  0.9438,    Adjusted R-squared:  0.9432 
F-statistic:  1646 on 1 and 98 DF,  p-value: < 2.2e-16
```

```
str(x)
```

```
List of 11
 $ call         : language lm(formula = ph ~ dox)
 $ terms        :Classes 'terms', 'formula'  language ph ~ dox
  .. ..- attr(*, "variables")= language list(ph, dox)
  .. ..- attr(*, "factors")= int [1:2, 1] 0 1
  .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. ..$ : chr [1:2] "ph" "dox"
  .. .. .. ..$ : chr "dox"
  .. ..- attr(*, "term.labels")= chr "dox"
  .. ..- attr(*, "order")= int 1
  .. ..- attr(*, "intercept")= int 1
  .. ..- attr(*, "response")= int 1
  .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
  .. ..- attr(*, "predvars")= language list(ph, dox)
  .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
  .. .. ..- attr(*, "names")= chr [1:2] "ph" "dox"
 $ residuals    : Named num [1:100] 0.12187 0.00161 0.03698 0.00519 -0.03372 ...
  ..- attr(*, "names")= chr [1:100] "1" "2" "3" "4" ...
 $ coefficients : num [1:2, 1:4] 7.50 9.03e-02 1.43e-02 2.23e-03 5.26e+02 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:2] "(Intercept)" "dox"
  .. ..$ : chr [1:4] "Estimate" "Std. Error" "t value" "Pr(>|t|)"
 $ aliased      : Named logi [1:2] FALSE FALSE
  ..- attr(*, "names")= chr [1:2] "(Intercept)" "dox"
 $ sigma        : num 0.0477
 $ df           : int [1:3] 2 98 2
 $ r.squared    : num 0.944
 $ adj.r.squared: num 0.943
 $ fstatistic   : Named num [1:3] 1646 1 98
  ..- attr(*, "names")= chr [1:3] "value" "numdf" "dendf"
 $ cov.unscaled : num [1:2, 1:2] 0.08956 -0.01317 -0.01317 0.00218
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:2] "(Intercept)" "dox"
  .. ..$ : chr [1:2] "(Intercept)" "dox"
 - 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:

```r
new.dox <- data.frame(dox = runif(5, 2, 10))
predict(pd.lm, new.dox)
```

```
       1        2        3        4        5 
8.156066 7.901664 7.968998 8.395535 8.293671 
```

```r
# include confidence intervals
predict(pd.lm, new.dox, interval = "confidence")
```

```
       fit      lwr      upr
1 8.156066 8.145053 8.167079
2 7.901664 7.890015 7.913312
3 7.968998 7.958911 7.979084
4 8.395535 8.375770 8.415301
5 8.293671 8.278097 8.309245
```

Models can be built on categorical predictors as well. In this example we test whether or not surface temperature differs among the first five stations:

```r
ctd <- read.csv("ctd.csv", stringsAsFactors = FALSE)
surf <- subset(ctd, depth == 1 & station %in% paste0("Station.", 1:5))

temp.lm <- lm(temp ~ station, surf)
summary(temp.lm)
```

```
Call:
lm(formula = temp ~ station, data = surf)

Residuals:
   Min     1Q Median     3Q    Max 
-4.583 -1.838 -0.160  1.367  6.331 

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)    
(Intercept)       17.2263     0.3090  55.749  < 2e-16 ***
stationStation.2  -0.3656     0.4370  -0.837  0.40349    
stationStation.3  -0.4963     0.4370  -1.136  0.25703    
stationStation.4  -0.8671     0.4370  -1.984  0.04816 *  
stationStation.5  -1.2236     0.4370  -2.800  0.00545 ** 
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.373 on 290 degrees of freedom
Multiple R-squared:  0.03099,   Adjusted R-squared:  0.01762 
F-statistic: 2.318 on 4 and 290 DF,  p-value: 0.05723
```

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 similar to the multi-category linear model and gets specified with the same formula using the `aov` function:

```
temp.aov <- aov(temp ~ station, surf)
summary(temp.aov)
```

```
             Df Sum Sq Mean Sq F value Pr(>F)  
station       4   52.2  13.060   2.318 0.0572 .
Residuals   290 1633.6   5.633                 
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
str(temp.aov)
```

```
List of 13
 $ coefficients : Named num [1:5] 17.226 -0.366 -0.496 -0.867 -1.224
  ..- attr(*, "names")= chr [1:5] "(Intercept)" "stationStation.2" "stationStation.3" "stationStation.4
 $ residuals    : Named num [1:295] 3.09 -2.99 -1.52 -3.03 -2.74 ...
  ..- attr(*, "names")= chr [1:295] "18" "40" "73" "103" ...
 $ effects      : Named num [1:295] -285.73 1.93 1.33 -1.6 -6.65 ...
  ..- attr(*, "names")= chr [1:295] "(Intercept)" "stationStation.2" "stationStation.3" "stationStation
 $ rank         : int 5
 $ fitted.values: Named num [1:295] 17.2 17.2 17.2 17.2 17.2 ...
  ..- attr(*, "names")= chr [1:295] "18" "40" "73" "103" ...
 $ assign       : int [1:5] 0 1 1 1 1
 $ qr           :List of 5
  ..$ qr   : num [1:295, 1:5] -17.1756 0.0582 0.0582 0.0582 0.0582 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:295] "18" "40" "73" "103" ...
  .. .. ..$ : chr [1:5] "(Intercept)" "stationStation.2" "stationStation.3" "stationStation.4" ...
  .. ..- attr(*, "assign")= int [1:5] 0 1 1 1 1
  .. ..- attr(*, "contrasts")=List of 1
  .. .. ..$ station: chr "contr.treatment"
  ..$ qraux: num [1:5] 1.06 1.03 1.03 1.05 1.08
  ..$ pivot: int [1:5] 1 2 3 4 5
  ..$ tol  : num 1e-07
  ..$ rank : int 5
  ..- attr(*, "class")= chr "qr"
 $ df.residual  : int 290
 $ contrasts    :List of 1
  ..$ station: chr "contr.treatment"
 $ xlevels      :List of 1
  ..$ station: chr [1:5] "Station.1" "Station.2" "Station.3" "Station.4" ...
 $ call         : language aov(formula = temp ~ station, data = surf)
 $ terms        :Classes 'terms', 'formula'  language temp ~ station
  .. ..- attr(*, "variables")= language list(temp, station)
  .. ..- attr(*, "factors")= int [1:2, 1] 0 1
  .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. ..$ : chr [1:2] "temp" "station"
  .. .. .. ..$ : chr "station"
  .. ..- attr(*, "term.labels")= chr "station"
  .. ..- attr(*, "order")= int 1
  .. ..- attr(*, "intercept")= int 1
  .. ..- attr(*, "response")= int 1
  .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
  .. ..- attr(*, "predvars")= language list(temp, station)
  .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "character"
  .. .. ..- attr(*, "names")= chr [1:2] "temp" "station"
 $ model        :'data.frame':  295 obs. of  2 variables:
  ..$ temp   : num [1:295] 20.3 14.2 15.7 14.2 14.5 ...
  ..$ station: chr [1:295] "Station.1" "Station.1" "Station.1" "Station.1" ...
  ..- attr(*, "terms")=Classes 'terms', 'formula'  language temp ~ station
  .. .. ..- attr(*, "variables")= language list(temp, station)
  .. .. ..- attr(*, "factors")= int [1:2, 1] 0 1
  .. .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. .. ..$ : chr [1:2] "temp" "station"
  .. .. .. .. ..$ : chr "station"
  .. .. ..- attr(*, "term.labels")= chr "station"
  .. .. ..- attr(*, "order")= int 1
  .. .. ..- attr(*, "intercept")= int 1
  .. .. ..- attr(*, "response")= int 1
  .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
  .. .. ..- attr(*, "predvars")= language list(temp, station)
  .. .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "character"
  .. .. .. ..- attr(*, "names")= chr [1:2] "temp" "station"
 - attr(*, "class")= chr [1:2] "aov" "lm"
```

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

```r
anova(temp.lm)
```

```
Analysis of Variance Table
```

```
Response: temp
           Df  Sum Sq Mean Sq F value  Pr(>F)
station     4   52.24 13.0602  2.3184 0.05723 .
Residuals 290 1633.62  5.6332
---
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(temp.aov)
```

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

Fit: aov(formula = temp ~ station, data = surf)

$station
                          diff       lwr         upr     p adj
Station.2-Station.1 -0.3655932 -1.565124  0.83393726 0.9190166
Station.3-Station.1 -0.4962712 -1.695802  0.70325929 0.7874740
Station.4-Station.1 -0.8671186 -2.066649  0.33241184 0.2762382
Station.5-Station.1 -1.2235593 -2.423090 -0.02402884 0.0430545
Station.3-Station.2 -0.1306780 -1.330208  1.06885251 0.9982509
Station.4-Station.2 -0.5015254 -1.701056  0.69800506 0.7809047
Station.5-Station.2 -0.8579661 -2.057497  0.34156438 0.2866747
Station.4-Station.3 -0.3708475 -1.570378  0.82868302 0.9150329
Station.5-Station.3 -0.7272881 -1.926819  0.47224234 0.4577427
Station.5-Station.4 -0.3564407 -1.555971  0.84308980 0.9256832
```

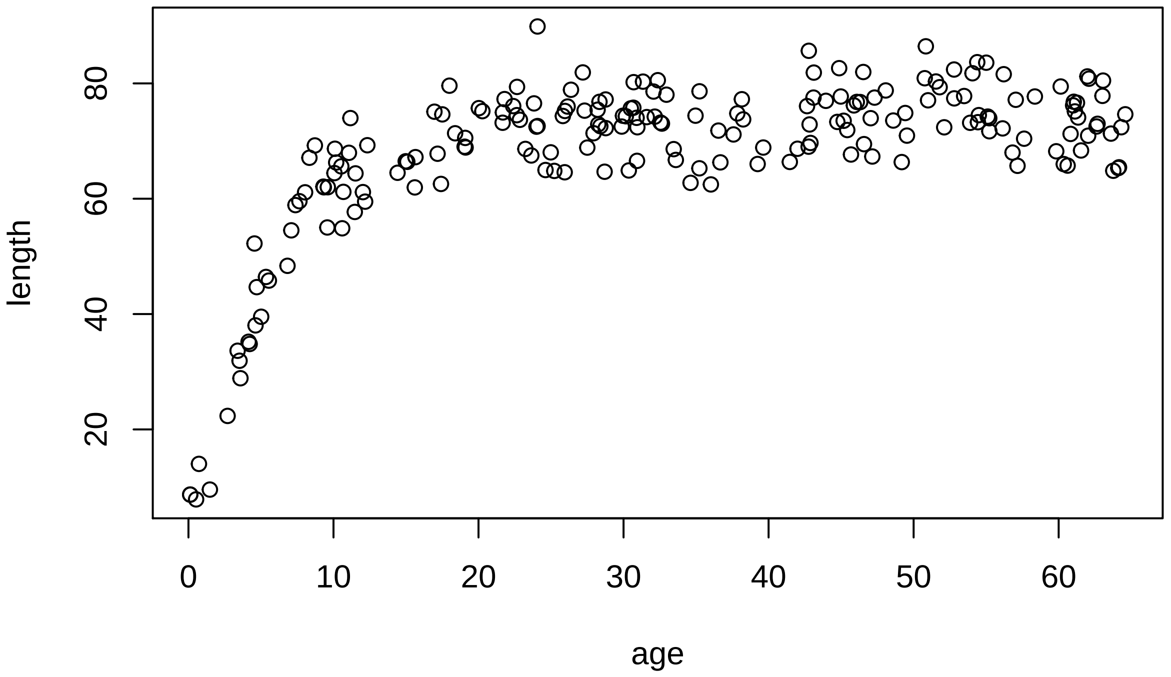
## Non-linear models

To illustrate non-linear model fitting and statistal tests, 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) {
  # Check to make sure age.range is a reasonable vector
  if (!is.numeric(age.range) || !is.vector(age.range))
    stop("'age.range' is not a numeric vector")
  if (any(age.range < 0)) stop("'age.range' < 0")
  if (age.range[1] >= age.range[2])
    stop("'age.range[1]' >= 'age.range[2]'")
  # 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`, which 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 numericDeriv(form[[3L]], names(ind), env): Missing value or an infinity produced when evaluati
```

```
# 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
8.309 2.182 0.272
```

```
 residual sum-of-squares: 5498

Number of iterations to convergence: 6
Achieved convergence tolerance: 2.8e-06
```

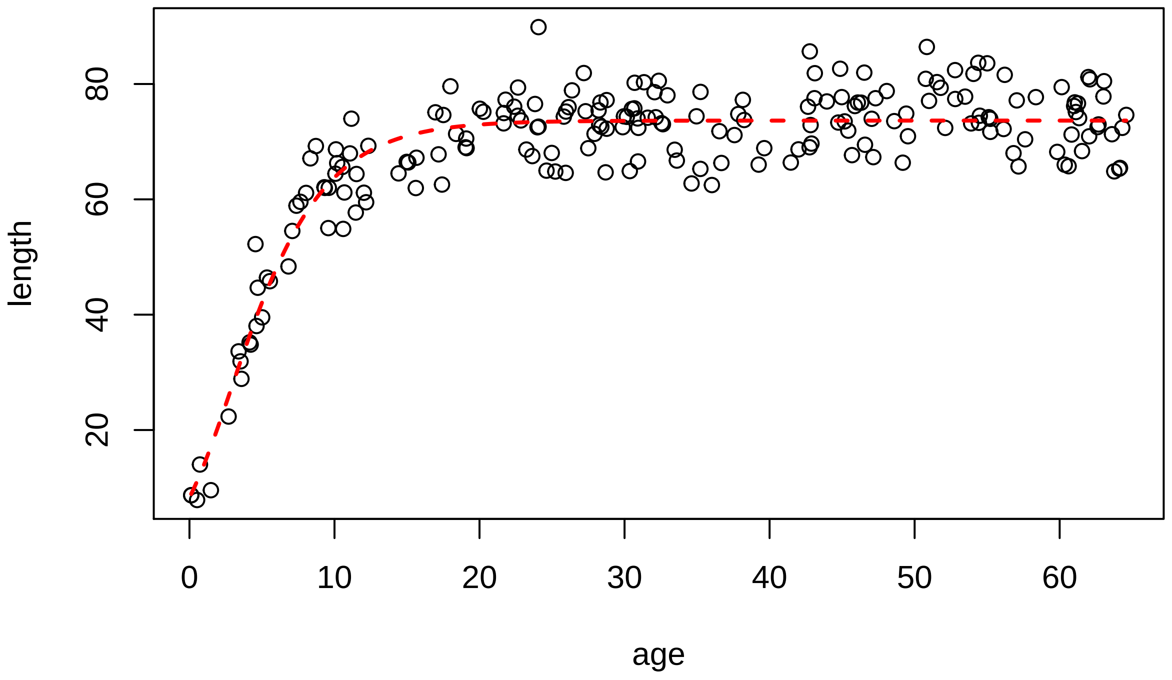
…and here are the estimated coefficients:

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

```
      lab         k         g
8.3093886 2.1819649 0.2720253
```

We'll plot the fitted curve:

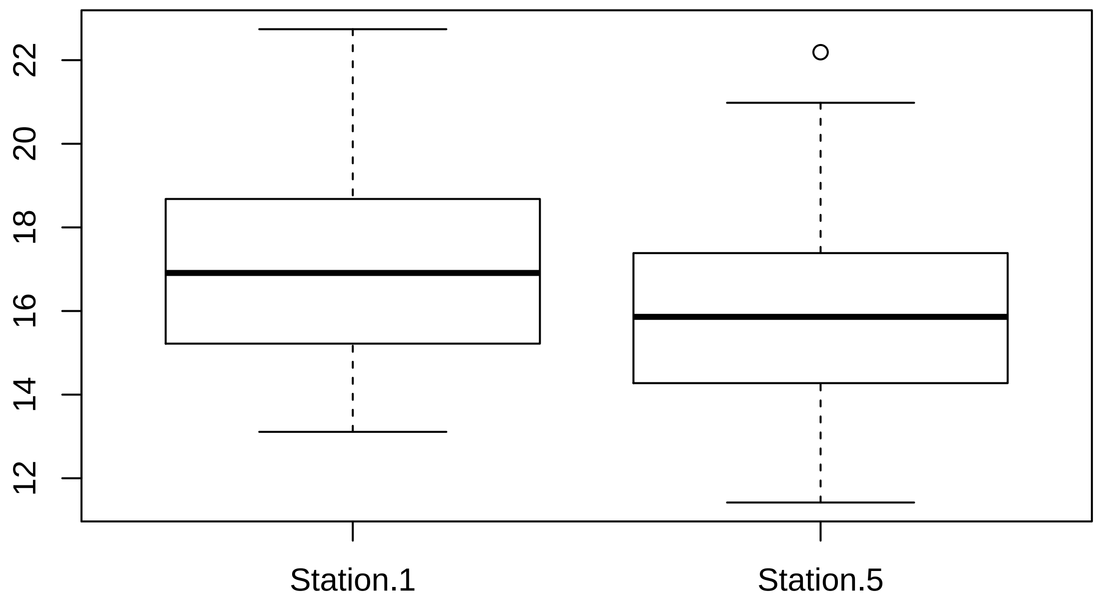
```r
grow.fit <- data.frame(
  age = seq(
    min(growth.df$age),
    max(growth.df$age),
    length.out = 1000
  )
)
grow.fit$length <- predict(gr.nls, grow.fit)
plot(length ~ age, growth.df)
lines(grow.fit$age, grow.fit$length, col = "red", lwd = 2, lty = "dashed")
```



### 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 conduct a t-test of surface temperatures between stations 1 and 5.

```r
surf15 <- subset(surf, station %in% c("Station.1", "Station.5"))
boxplot(temp ~ station, surf15)
```



```
surf15.t <- t.test(temp ~ station, surf15)
surf15.t
```

```

    Welch Two Sample t-test

data:  temp by station
t = 2.7601, df = 115.31, p-value = 0.006725
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.3454737 2.1016449
sample estimates:
mean in group Station.1 mean in group Station.5 
               17.22627                16.00271 
```

```
str(surf15.t)
```

```
List of 9
 $ statistic  : Named num 2.76
  ..- attr(*, "names")= chr "t"
 $ parameter  : Named num 115
  ..- attr(*, "names")= chr "df"
 $ p.value    : num 0.00672
 $ conf.int   : atomic [1:2] 0.345 2.102
  ..- attr(*, "conf.level")= num 0.95
 $ estimate   : Named num [1:2] 17.2 16
  ..- attr(*, "names")= chr [1:2] "mean in group Station.1" "mean in group Station.5"
 $ null.value : Named num 0
  ..- attr(*, "names")= chr "difference in means"
 $ alternative: chr "two.sided"
 $ method     : chr "Welch Two Sample t-test"
 $ data.name  : chr "temp by station"
 - attr(*, "class")= chr "htest"
```

For a chi-squared test, there's the function `chisq.test`. As an example, we'll see if ctd sampling is random across years and months. Let's first examine years, using the output from `table`:

```r
# Extract year, month, and day as columns:
ctd$year <- as.numeric(substr(ctd$sample_date, 1, 4))
ctd$month <- as.numeric(substr(ctd$sample_date, 6, 7))
ctd$day <- as.numeric(substr(ctd$sample_date, 9, 10))

yr.freq <- table(ctd$year)
yr.chisq <- chisq.test(yr.freq)
str(yr.chisq)
```

```
List of 9
 $ statistic: Named num 5658
  ..- attr(*, "names")= chr "X-squared"
 $ parameter: Named num 6
  ..- attr(*, "names")= chr "df"
 $ p.value  : num 0
 $ method   : chr "Chi-squared test for given probabilities"
 $ data.name: chr "yr.freq"
 $ observed : 'table' int [1:7(1d)] 12886 13908 13865 13907 6760 8195 8120
  ..- attr(*, "dimnames")=List of 1
  .. ..$ : chr [1:7] "2010" "2011" "2012" "2013" ...
 $ expected : Named num [1:7] 11092 11092 11092 11092 11092 ...
  ..- attr(*, "names")= chr [1:7] "2010" "2011" "2012" "2013" ...
 $ residuals: table [1:7(1d)] 17 26.7 26.3 26.7 -41.1 ...
  ..- attr(*, "dimnames")=List of 1
  .. ..$ : chr [1:7] "2010" "2011" "2012" "2013" ...
 $ stdres   : table [1:7(1d)] 18.4 28.9 28.4 28.9 -44.4 ...
  ..- attr(*, "dimnames")=List of 1
  .. ..$ : chr [1:7] "2010" "2011" "2012" "2013" ...
 - attr(*, "class")= chr "htest"
```

Here's a test of sampling frequency by year and month:

```r
yr.mo.chisq <- chisq.test(table(ctd$month, ctd$year))
yr.mo.chisq
```

```

    Pearson's Chi-squared test

data:  table(ctd$month, ctd$year)
X-squared = 7509.3, df = 66, p-value < 2.2e-16
```

```r
str(yr.mo.chisq)
```

```
List of 9
 $ statistic: Named num 7509
  ..- attr(*, "names")= chr "X-squared"
 $ parameter: Named int 66
  ..- attr(*, "names")= chr "df"
 $ p.value  : num 0
 $ method   : chr "Pearson's Chi-squared test"
 $ data.name: chr "table(ctd$month, ctd$year)"
 $ observed : 'table' int [1:12, 1:7] 1154 1136 1157 179 1153 1147 1157 1158 1164 1163 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:12] "1" "2" "3" "4" ...
  .. ..$ : chr [1:7] "2010" "2011" "2012" "2013" ...
 $ expected : num [1:12, 1:7] 919 1390 926 775 1393 ...
```

```
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:12] "1" "2" "3" "4" ...
  .. ..$ : chr [1:7] "2010" "2011" "2012" "2013" ...
 $ residuals: table [1:12, 1:7] 7.77 -6.81 7.59 -21.42 -6.42 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:12] "1" "2" "3" "4" ...
  .. ..$ : chr [1:7] "2010" "2011" "2012" "2013" ...
 $ stdres   : table [1:12, 1:7] 8.82 -7.89 8.62 -24.19 -7.45 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:12] "1" "2" "3" "4" ...
  .. ..$ : chr [1:7] "2010" "2011" "2012" "2013" ...
 - attr(*, "class")= chr "htest"
```

…we can get also the same result by setting the `x` and `y` arguments of `chisq.test` to the original columns:

```
chisq.test(ctd$month, ctd$year)
```

```

    Pearson's Chi-squared test

data:  ctd$month and ctd$year
X-squared = 7509.3, df = 66, p-value < 2.2e-16
```

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

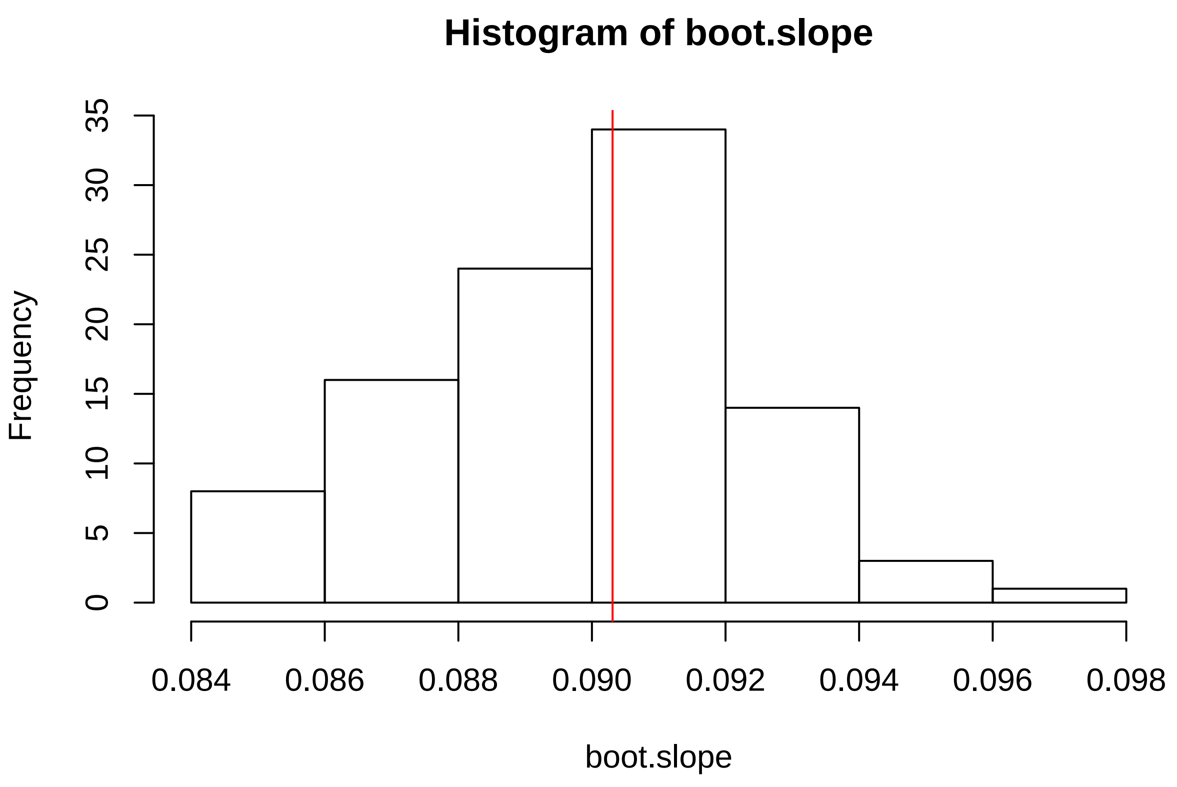
```
chisq.test(ctd$month, ctd$year, simulate.p.value = TRUE)
```

```

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

data:  ctd$month and ctd$year
X-squared = 7509.3, df = NA, p-value = 0.0004998
```

### Bootstrapping

We can get bootstrap estimates of the varianc of a parameter by randomly selecting rows of our original data with replacement. As an example, here's some code for getting the variance of the slope parameter from our simulated ph/dox data from 100 bootstrap samples:

```
# calculate the observed value
obs.slope <- coef(lm(ph ~ dox))[2]
# conduct 100 bootstrap replicates
boot.slope <- sapply(1:100, function(i) {
  boot.i <- sample(1:length(ph), length(ph), replace = TRUE)
  boot.ph <- ph[boot.i]
  boot.dox <- dox[boot.i]
  boot.lm <- lm(boot.ph ~ boot.dox)
  coef(boot.lm)[2]
})
# a histogram of the boostrap slopes with the observed value in red
hist(boot.slope)
abline(v = obs.slope, col = "red")
```



```r
# the observed value and a summary of the bootstrap replicates
obs.slope
```

```
      dox
0.0903088
```

```r
summary(boot.slope)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.08427 0.08836 0.09017 0.08992 0.09170 0.09655
```

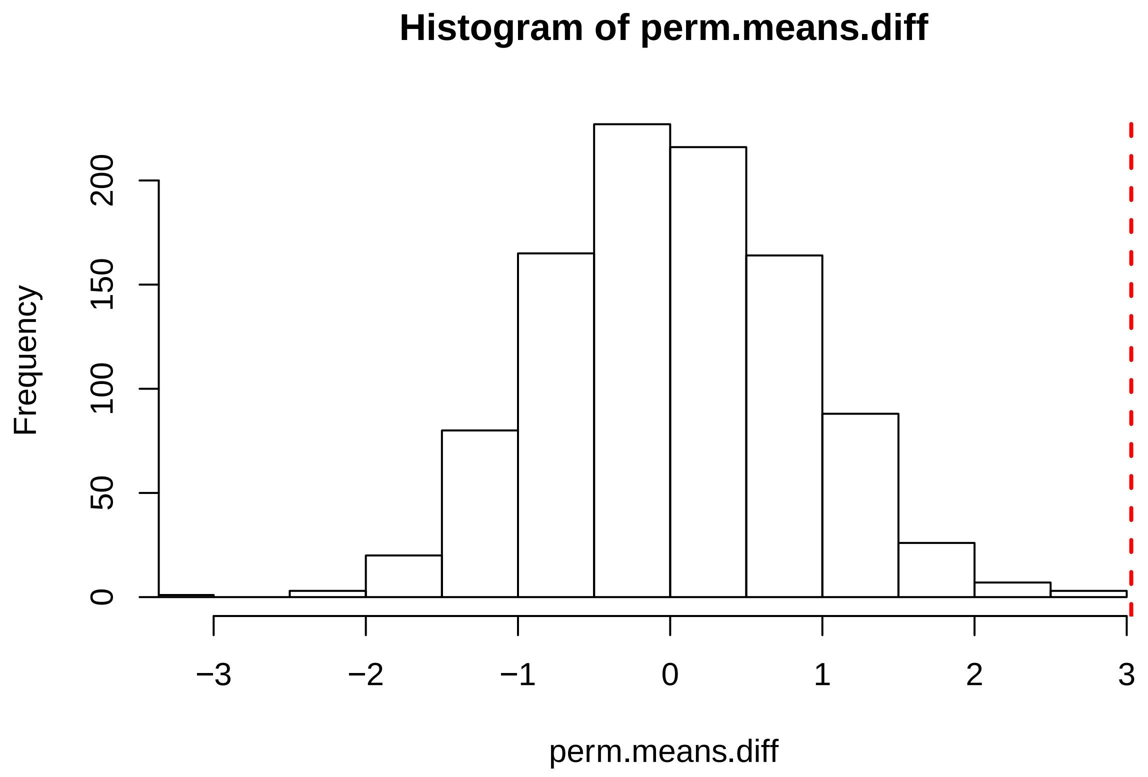
### Permutation tests

We can also use resampling to conduct permutation tests where we create a null distribution by randomly permuting the predictor variable. Here we simulate separate male and female growth and test if adults (> 18 y/o) have different lengths.

```r
# Simulate data for the two sexes
m.growth <- sim.growth.func(c(0, 65), 10, 2.05, 0.27, 5, 100)
f.growth <- sim.growth.func(c(0, 65), 10, 2.02, 0.25, 4, 100)
# Extract adults
adult.m <- m.growth[m.growth[, "age"] > 18, ]
adult.f <- f.growth[f.growth[, "age"] > 18, ]

# Create combined sex data.frame
adult.df <- rbind(adult.m, adult.f)
adult.df$sex <- rep(c("m", "f"), c(nrow(adult.m), nrow(adult.f)))

# the t-test result
t.test(length ~ sex, adult.df)
```

```
    Welch Two Sample t-test

data:  length by sex
t = -3.9812, df = 137.6, p-value = 0.0001105
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -4.534991 -1.525090
sample estimates:
mean in group f mean in group m 
       74.62510        77.65514 
```

```
# calculate observed difference of means
obs.means <- tapply(adult.df$length, adult.df$sex, mean)
obs.means.diff <- diff(obs.means)
# permute sexes and calculate difference of means for null distribution
perm.means <- sapply(1:1000, function(i) {
  tapply(adult.df$length, sample(adult.df$sex), mean)
})
perm.means.diff <- apply(perm.means, 2, diff)

# plot histogram of permutation differences with observed difference
hist(perm.means.diff, xlim = range(c(obs.means.diff, perm.means.diff)))
abline(v = obs.means.diff, col = "red", lty = "dashed", lwd = 2)
```



```
# observed difference
obs.means.diff
```

```
       m 
3.030041 
```

```
# summary of permutation differences
summary(perm.means.diff)
```

```
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max. 
-3.115000 -0.535300  0.008564  0.021450  0.595000  2.574000 
```

```
# fraction of permutation >= observed
mean(perm.means.diff >= obs.means.diff)
```

```
[1] 0
```