

Introduction to R: Core Language Tutorial

Dr Jeromy Anglim

```
library(ProjectTemplate); load.project()
```

Basic Arithmetic and Logical Operations

```
# You can use R like a calculator
```

```
1 + 1 # addition
```

```
## [1] 2
```

```
10 - 9 # subtraction
```

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

```
10 * 10 # multiplication
```

```
## [1] 100
```

```
100 / 10 # division
```

```
## [1] 10
```

```
10 ^ 2 # exponentiation
```

```
## [1] 100
```

```
abs(-10) # absolute value
```

```
## [1] 10
```

```
ceiling(3.5) # round up to next integer
```

```
## [1] 4
```

```
floor(3.5) # round down to next integer
```

```
## [1] 3
```

```
sqrt(100) # square roots
```

```
## [1] 10
```

```
exp(2) # exponents
```

```
## [1] 7.389056
```

```
pi # mathematical constant pi
```

```
## [1] 3.141593
```

```
exp(1) # mathematical constant e
```

```
## [1] 2.718282
```

```
log(100) # natural logs (i.e., base e)
```

```
## [1] 4.60517
```

```
log(100, base= 10) # base 10 logs
```

```
## [1] 2
```

```
# Use parentheses to clarify order of operations  
(1 + 1) * 2
```

```
## [1] 4
```

```
1 + (1 * 2)
```

```
## [1] 3
```

```
# You can test for equality  
# TRUE and FALSE are keywords  
# T and F are synonyms, but are generally discouraged  
TRUE
```

```
## [1] TRUE
```

```
FALSE
```

```
## [1] FALSE
```

```
1 == 2 # Equality (Return TRUE if equal)
```

```
## [1] FALSE
```

```
1 != 2 # Inequality (Return FALSE if unequal)
```

```
## [1] TRUE
```

```
10 > 9 # Greater than
```

```
## [1] TRUE
```

```
9 < 10 # Less than
```

```
## [1] TRUE
```

```
10 <= 10 # Less than or equal
```

```
## [1] TRUE
```

```
2 %in% c(1, 2, 3) # is the number in the vector
```

```
## [1] TRUE
```

```
# TRUE and FALSE coerces to 1 and 0 respectively  
as.numeric(TRUE)
```

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

```
as.numeric(FALSE)
```

```
## [1] 0
```

```
# Logical converting to 0, 1 is useful  
x <- c(2, 5, 7, 10, 15)  
x > 5
```

```
## [1] FALSE FALSE TRUE TRUE TRUE
```

```
sum(x > 5) # sum of a 0-1 variable is a count
```

```
## [1] 3
```

```
mean(x > 5) # mean of a 0-1 variable is a proportion
```

```
## [1] 0.6
```

Basic language features

```
#####
# Assignment:
# To assign values to a variable either use <- or =
# <- is the more common convention in R
x <- 1 + 1
x
```

```
## [1] 2
```

```
# = is the common assignment operator in other programming
# languages. It does work in R, but is not the convention.
y = 1 + 1
y
```

```
## [1] 2
```

```
#####
# Variable name rules:
# Variable names generally
# 1. Start with a letter (lower or uppercase)
# 2. Followed by letters, numbers, underscore (_), or period (.)
# 3. No spaces

# These do not work
# my variable <- 1234
# 1234variable <- 1234
# 1234 <- 1234

# This works
myvariable <- 1234
my_variable <- 1234
my.variable <- 1234
myvariable123 <- 1234
myVariable <- 1234
my.variable <- 1234

# R has many naming conventions
# As a matter of preference, style, and convenience, I prefer:
# 1. Short but descriptive names
#   * Less than 8 characters for names of lists and data.frames
#   * Less than 15 characters for variables names in data.frames
# 2. Use underscore to separate words within a variable name
# 3. Avoid upper case letters

# Names starting with a period are hidden
.myvariable <- 1234
ls()
```

```
## [1] "config"          "csurvey"          "helper.function"
## [4] "my_variable"     "my.variable"      "myvariable"
## [7] "myVariable"      "myvariable123"    "project.info"
## [10] "x"               "y"
```

```
ls(all.names = TRUE)
```

```
## [1] ".myvariable"      ".Random.seed"      "config"
## [4] "csurvey"           "helper.function"   "my_variable"
## [7] "my.variable"       "myvariable"        "myVariable"
## [10] "myvariable123"     "project.info"      "x"
## [13] "y"
```

```
#####
# Comments:
# Comments are any text on a line following a hash #
# 1. They often appear as the first character of a line
#    to present a whole line comment
# 2. At the end of a common on a line
mean(c(1,2,3,4)) # Example of end of line comment
```

```
## [1] 2.5
```

```
# 3. Half way through a command at the end of a line
c(1, # Example comment
  2,3, # Another comment
  4)
```

```
## [1] 1 2 3 4
```

```
#####
# Spaces:
# R will generally permits zero, one or more spaces between
# variables, operators, and other syntactic elements.
# However, appropriate and consistent spacing improves
# the readability of you scripts.
# See Hadley Wickham's style guide:
# http://adv-r.had.co.nz/Style.html
```

```
# This is bad but works
x<-c(1,2,3,400)*2
x<-  c ( 1,2,3, 400)* 2
```

```
# This is more readable:
# Add spaces after variables, operators, commas
x <- c(1, 2, 3, 400) * 2
```

```
#####
# Multiline line commands
# Commands can generally span multiple lines
# as long as R does not think the command has finished
```

```
# This works
x <- c("apple",
      "banana")
x
```

```
## [1] "apple" "banana"
```

```
y <- 10 +  
  10 #this works  
y
```

```
## [1] 20
```

```
# This does not work  
y <- 10  
  + 10
```

```
## [1] 10
```

```
y
```

```
## [1] 10
```

```
#####  
# Multiple commands on one line  
# You can include more than one command on one line  
# by separat the commands by a semicolon.  
# But generally, you should avoid doing this as it is not  
# very readable.  
x <- c(1, 2); y <- c(3, 4); z <- rnorm(10)  
x;y;z
```

```
## [1] 1 2
```

```
## [1] 3 4
```

```
## [1] -0.6135865 -0.8046023 -0.1913762 1.6264101 0.9699219 -1.4279235  
## [7] -0.3623782 0.2020919 0.3325194 -0.8352546
```

```
#####  
# # R is case sensitive  
test <- "lower case"  
TEST <- "upper case"  
TEST
```

```
## [1] "upper case"
```

```
test # The original value was not lost
```

```
## [1] "lower case"
```

```
# because test is different to TEST  
Test # This variable does not exist
```

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

```
Test <- "title case"
Test
```

```
## [1] "title case"
```

```
# tip: It's often simpler to make variables lower case
#      so that you don't have to think about case.
```

Understanding directories

```
# R has a working directory.
# This is important when loading and saving files to disk
getwd() # show the current working directory
```

```
## [1] "/Users/jeromy/teaching/r-training/r-training/training-exercises"
```

```
# you can use setwd to change the working directory
# setwd("~/blah/myproject")
```

```
# Tip: Open RStudio with the Rproj file then the working directory
# will be the directory containing the Rproj file.
```

```
# Tips:
# * Try to avoid spaces in file names
#   (use hyphen or underscore instead)
# * If on Windows, then disable "hide extensions of
#   known file types" (see folder options )
# * If you do use spaces, then you'll need to escape the space with
#   a slash (e.g., ("my\ documents"))
# * Use backslash as the directory separator
# * Store all relevant files for a project within
#   the project working directory
```

The Workspace

```
#####
# R Sessions:
```

```
# Quitting R
# You can end the R Session using the q function
# q()
```

```
# But if you are in Rstudio, it is simpler to:
# * Just quit RStudio and this will quit the R session
# * Use the session menu in RStudio to Restart or Terminate
#   an R session
```

```
#####
```

```
# Workspaces and environments:
```

```
# list environments
```

```
search()
```

```
## [1] ".GlobalEnv"          "package:MASS"
## [3] "package:Hmisc"       "package:ggplot2"
## [5] "package:Formula"     "package:survival"
## [7] "package:grid"        "package:lattice"
## [9] "package:psych"       "package:ProjectTemplate"
## [11] "package:stats"       "package:graphics"
## [13] "package:grDevices"   "package:utils"
## [15] "package:datasets"    "package:methods"
## [17] "Autoloads"           "package:base"
```

```
# Create some objects in the global environment
```

```
x <- 1:10
```

```
y <- 1:20
```

```
data(mtcars) # Add a built-in dataset mtcars
```

```
# Show objects in the global environment
```

```
ls()
```

```
## [1] "config"          "csurvey"          "helper.function"
## [4] "mtcars"          "my_variable"      "my.variable"
## [7] "myvariable"      "myVariable"       "myvariable123"
## [10] "project.info"    "test"             "Test"
## [13] "TEST"            "x"                 "y"
## [16] "z"
```

```
# or look at the environment pane in RStudio
```

```
#####
```

```
# Removing objects:
```

```
# Removing named objects with the rm function
```

```
rm(x)
```

```
ls()
```

```
## [1] "config"          "csurvey"          "helper.function"
## [4] "mtcars"          "my_variable"      "my.variable"
## [7] "myvariable"      "myVariable"       "myvariable123"
## [10] "project.info"    "test"             "Test"
## [13] "TEST"            "y"                 "z"
```

```
rm(y, mtcars)
```

```
# Remove all objects from global workspace
```

```
# Option 1. Use the following command
```

```
rm(list = ls())
```

```
# Option 2. Click the broom object in RStudio Environment pane
```



```
#####
# Saving objects
# Save all objects in the workspace
save.image()

x <- 30
y <- 1:10
# Save specific named objects using save function.
# rdata or RData is the standard file extension.
save(x, y, file = "output/y.rdata")

# Let's remove x and change y
rm(x)
y <- "changed"
y
```

```
## [1] "changed"
```

```
# load variables stored in rdata file
load(file = "output/y.rdata")
x
```

```
## [1] 30
```

```
y
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

```
# Tips:
# * Try to avoid using save.image() to store temporary calculations
# * Instead, try to write scripts that can be run to return you to
# your current state of analyses.
```

Data types: Logical, character, numeric

```
#####
# Basic data types
# The most common basic vector types are
x <- c(FALSE, TRUE) # logical vector
y <- c("a", "b", "cat", "dog") # character vector
z1 <- c(100, 1, 2, 3) # numeric integer vector
z2 <- c(100.2, 0.4, 0.9) # numeric real/double vector
class(x); typeof(x); mode(x)
```

```
## [1] "logical"
```

```
## [1] "logical"
```

```
## [1] "logical"
```

```
class(y); typeof(y); mode(y)
```

```
## [1] "character"
```

```
## [1] "character"
```

```
## [1] "character"
```

```
class(z1); typeof(z1); mode(z1)
```

```
## [1] "numeric"
```

```
## [1] "double"
```

```
## [1] "numeric"
```

```
class(z2); typeof(z2); mode(z2)
```

```
## [1] "numeric"
```

```
## [1] "double"
```

```
## [1] "numeric"
```

```
# Checking type of object  
# there are a range of "is." functions for that return TRUE  
# if object is of corresponding type  
# apropos("^is\\.")  
is.logical(c(TRUE, TRUE))
```

```
## [1] TRUE
```

```
is.numeric(c("a", "b"))
```

```
## [1] FALSE
```

```
is.character(c(1, 2, 3))
```

```
## [1] FALSE
```

```
#####  
# Conversion of Types:  
# R has functions that explicitly convert data types  
# apropos("^as\\.")  
as.character(c(1, 2, 3, 4))
```

```
## [1] "1" "2" "3" "4"
```

```
as.numeric(c("1", "2a", "3", "four"))
```

```
## Warning: NAs introduced by coercion
```

```
## [1] 1 NA 3 NA
```

```
as.numeric(c(FALSE, FALSE, TRUE, TRUE))
```

```
## [1] 0 0 1 1
```

```
# R often performs conversions implicitly  
sum(c(FALSE, TRUE, TRUE)) # converts logical to 0, 1 numeric
```

```
## [1] 2
```

```
paste0("v", c(1, 2, 3)) # converts numeric vector to character
```

```
## [1] "v1" "v2" "v3"
```

Basic data structures: Vectors, Matrices, Lists, Data.frames

```
#####
```

```
# Vectors:
```

```
# In R, a single value (scalar) is a vector.
```

```
x <- 1 # I.e., x is a vector of length 1
```

```
# In addition to importing data,
```

```
# R has various functions for creating vectors.
```

```
c(1, 2, 3, 4) # c stands for combine
```

```
## [1] 1 2 3 4
```

```
1:10 # create an integer sequence 1 to 10
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

```
seq(1, 10) # alternative way of creating a sequence
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

```
seq(1, 10, by = 2) # The function has additional options
```

```
## [1] 1 3 5 7 9
```

```
rep(1, 5) # repeat a value a certain number of times
```

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

```
rep(c(1,2,3), 5) # repeat a value a certain number of times
```

```
## [1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
```

```
# as well as many simulation functions which we'll cover later
```

```
# Initial examples:
```

```
# Sample 10 items with replacement from
```

```
sample(x = c("happy", "funny", "silly"), size = 10, replace = TRUE)
```

```
## [1] "silly" "funny" "happy" "silly" "silly" "silly" "silly" "silly"
```

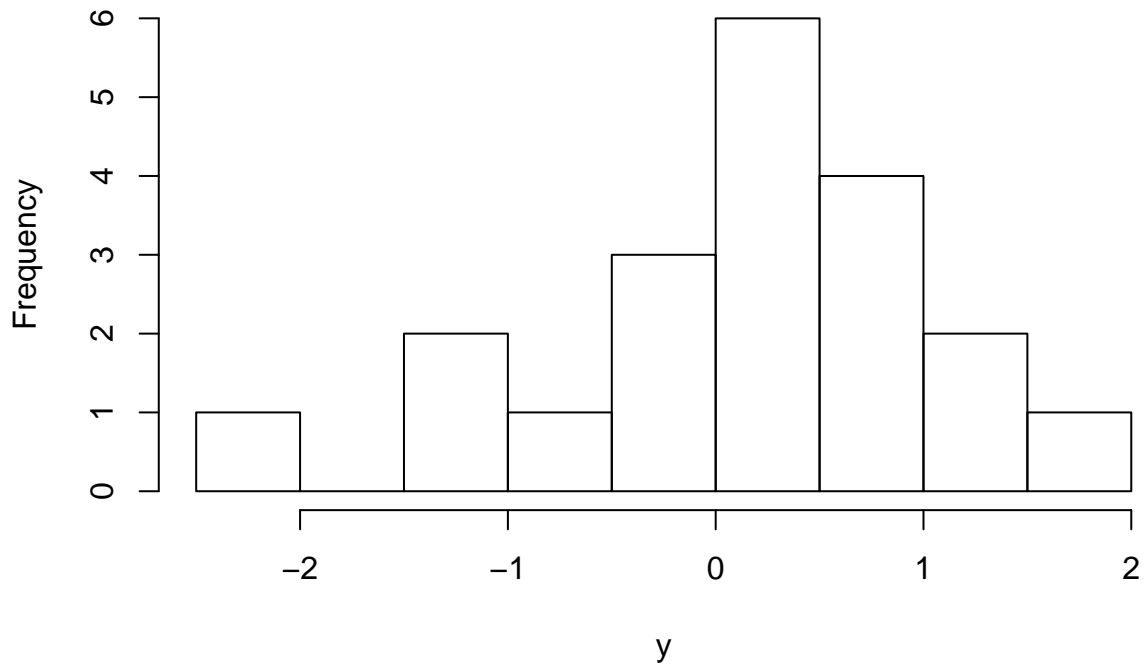
```
## [9] "silly" "happy"
```

```
# Sample 20 values from a normal distribution
```

```
y <- rnorm(n = 20, mean = 0, sd = 1)
```

```
hist(y) # show values in histogram
```

Histogram of y



```
# Vectors can have names
```

```
x <- c(1,2,3,4,5)
```

```
names(x) <- c("a", "b", "c", "d", "e")
```

```
x
```

```
## a b c d e
```

```
## 1 2 3 4 5
```

```
# Extracting vectors
x[c(1,2)] # by numeric position
```

```
## a b
## 1 2
```

```
x[x < 3] # by logical vector
```

```
## a b
## 1 2
```

```
x[c("b", "c")] # by name
```

```
## b c
## 2 3
```

```
#####
# Matrices:
# All data must be of same type (e.g., numeric, character, logical)
y <- matrix(c(1, 2,
              4, 5,
              7, 8 ),
            byrow = TRUE, ncol = 2)
y
```

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

```
class(y)
```

```
## [1] "matrix"
```

```
# number of rows and columns
dim(y) # Number of rows and columns
```

```
## [1] 3 2
```

```
nrow(y) # Number of rows
```

```
## [1] 3
```

```
ncol(y) # Number of columns
```

```
## [1] 2
```

```
# Rows and columns can be given names
rownames(y) <- c("a", "b", "c")
colnames(y) <- c("col1", "col2")
```

```
# Rows and columns can be indexed
y["a", ] # By rowname
```

```
## col1 col2
##      1      2
```

```
y[, "col1"] # By column name
```

```
## a b c
## 1 4 7
```

```
y["a", "col1"] # By both
```

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

```
y[c(1,2), ] # By row position
```

```
##      col1 col2
## a         1      2
## b         4      5
```

```
y[,1] # By column position
```

```
## a b c
## 1 4 7
```

```
y[c(2,3), 2] # By column position
```

```
## b c
## 5 8
```

```
#####
# Lists
# Store arbitrary structures of one or more named elements.
# Elements can be of different lengths
# Lists can contain lists can be nested to create tree like structures
# Lists are commonly used for representing results of analyses
```

```
w <- list(apple = c("a", "b", "c"),
          banana = c(1,2),
          carrot = FALSE,
          animals = list(dog = c("dog1", "dog2"),
                        cat = c(TRUE, FALSE)))
```

```
class(w)
```

```
## [1] "list"
```

```
# Accessing one element of list  
w$apple # using dollar notation
```

```
## [1] "a" "b" "c"
```

```
w[[1]] # by position
```

```
## [1] "a" "b" "c"
```

```
w[["apple"]] # by name (double brackets)
```

```
## [1] "a" "b" "c"
```

```
# Accessing subset of list  
w[c(1, 2)] # by position (single bracket)
```

```
## $apple  
## [1] "a" "b" "c"  
##  
## $banana  
## [1] 1 2
```

```
w[c("apple", "banana")] # by name
```

```
## $apple  
## [1] "a" "b" "c"  
##  
## $banana  
## [1] 1 2
```

```
w[c(FALSE, FALSE, TRUE, TRUE)] # by logical vector
```

```
## $carrot  
## [1] FALSE  
##  
## $animals  
## $animals$dog  
## [1] "dog1" "dog2"  
##  
## $animals$cat  
## [1] TRUE FALSE
```

```
# Quick illustration of a list object returned by  
# a statistical function
```

```
# We'll simulate some data for two hypothetical groups x and y  
# and perform an independent samples t-test.
```

```
x <- rnorm(10, mean = 0, sd = 1)
y <- rnorm(10, mean = 1, sd = 1)
fit <- t.test(x, y)
```

```
# The function
class(fit) # class does not say list, but it is a list
```

```
## [1] "htest"
```

```
mode(fit)
```

```
## [1] "list"
```

```
str(fit) # show structure of object
```

```
## List of 9
## $ statistic : Named num -2.29
## .. attr(*, "names")= chr "t"
## $ parameter : Named num 13.8
## .. attr(*, "names")= chr "df"
## $ p.value : num 0.0386
## $ conf.int : atomic [1:2] -2.1775 -0.0681
## .. attr(*, "conf.level")= num 0.95
## $ estimate : Named num [1:2] -0.121 1.002
## .. attr(*, "names")= chr [1:2] "mean of x" "mean of y"
## $ 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 "x and y"
## - attr(*, "class")= chr "htest"
```

```
names(fit) # show names of elements
```

```
## [1] "statistic" "parameter" "p.value" "conf.int" "estimate"
## [6] "null.value" "alternative" "method" "data.name"
```

```
# we can view particular elements
fit$statistic
```

```
## t
## -2.286389
```

```
fit$parameter
```

```
## df
## 13.80062
```



```
fit$p.value
```

```
## [1] 0.03857336
```

```
# or extract subsets of the list  
fit[c("statistic", "parameter", "p.value")]
```

```
## $statistic  
##      t  
## -2.286389  
##  
## $parameter  
##      df  
## 13.80062  
##  
## $p.value  
## [1] 0.03857336
```

```
#####  
# Data Frames:  
# Data frames are the standard data structure used for storing  
# data. If you have used other software (e.g., SPSS, Excel, etc.),  
# this is what you may think of as a "dataset".  
# Columns can be of different data types (e.g., character, numeric, logical, etc.)  
z <- data.frame(var1 = 1:9, var2 = letters[1:9])  
z
```

```
##   var1 var2  
## 1    1    a  
## 2    2    b  
## 3    3    c  
## 4    4    d  
## 5    5    e  
## 6    6    f  
## 7    7    g  
## 8    8    h  
## 9    9    i
```

```
# Tip: Some functions work with matrices,  
# some work with data.frames,  
# and some work with both.  
# * If you are wanting to store data like you might store in  
# a database, then you'll generally want a data.frame.  
# * If you are dealing with a mathematical object that you  
# want to perform a mathematical operation on, then you generally  
# want a matrix (e.g., correlation matrix, covariance matrix,  
# distance matrix in MDS, matrices used for matrix algebra).
```

Working with data frames

```
# Let's use the built-in survey data.frame dataset
library(MASS)
data(survey)
?survey
mydata <- na.omit(survey) # for simplicity I'll exclude missing data
shortdata <- mydata[1:6, 1:5]
shortdata
```

```
##      Sex Wr.Hnd NW.Hnd W.Hnd      Fold
## 1 Female  18.5   18.0 Right R on L
## 2  Male  19.5   20.5 Left  R on L
## 5  Male  20.0   20.0 Right Neither
## 6 Female  18.0   17.7 Right L on R
## 7  Male  17.7   17.7 Right L on R
## 8 Female  17.0   17.3 Right R on L
```

```
#####
# Extracting observations (i.e., rows) and
# variables (i.e., columns).
# There are similarities to matrices and lists
# Select observations
shortdata[1:5, ] # by row number
```

```
##      Sex Wr.Hnd NW.Hnd W.Hnd      Fold
## 1 Female  18.5   18.0 Right R on L
## 2  Male  19.5   20.5 Left  R on L
## 5  Male  20.0   20.0 Right Neither
## 6 Female  18.0   17.7 Right L on R
## 7  Male  17.7   17.7 Right L on R
```

```
shortdata[c(5,4,3,2,1), ] # re-order
```

```
##      Sex Wr.Hnd NW.Hnd W.Hnd      Fold
## 7  Male  17.7   17.7 Right L on R
## 6 Female  18.0   17.7 Right L on R
## 5  Male  20.0   20.0 Right Neither
## 2  Male  19.5   20.5 Left  R on L
## 1 Female  18.5   18.0 Right R on L
```

```
shortdata[ shortdata$Sex == "Female", ] # by logical vector
```

```
##      Sex Wr.Hnd NW.Hnd W.Hnd      Fold
## 1 Female  18.5   18.0 Right R on L
## 6 Female  18.0   17.7 Right L on R
## 8 Female  17.0   17.3 Right R on L
```

```
shortdata[c("1", "2"), ] # by rownames
```

```
##      Sex Wr.Hnd NW.Hnd W.Hnd  Fold
## 1 Female  18.5   18.0 Right R on L
## 2  Male  19.5   20.5  Left R on L
```

```
# Select variables
```

```
shortdata[, c(1,2)] # by position like a matrix
```

```
##      Sex Wr.Hnd
## 1 Female  18.5
## 2  Male  19.5
## 5  Male  20.0
## 6 Female  18.0
## 7  Male  17.7
## 8 Female  17.0
```

```
shortdata[c(1,2)] # by position like a list
```

```
##      Sex Wr.Hnd
## 1 Female  18.5
## 2  Male  19.5
## 5  Male  20.0
## 6 Female  18.0
## 7  Male  17.7
## 8 Female  17.0
```

```
shortdata[, c("Sex", "Fold")] # by name like a matrix
```

```
##      Sex      Fold
## 1 Female R on L
## 2  Male R on L
## 5  Male Neither
## 6 Female L on R
## 7  Male L on R
## 8 Female R on L
```

```
shortdata[c("Sex", "Fold")] #
```

```
##      Sex      Fold
## 1 Female R on L
## 2  Male R on L
## 5  Male Neither
## 6 Female L on R
## 7  Male L on R
## 8 Female R on L
```

```
shortdata$Sex # by name to get a single variable
```

```
## [1] Female Male Male Female Male Female
## Levels: Female Male
```

```
#####
# Names
names(shortdata) # get variable names
```

```
## [1] "Sex" "Wr.Hnd" "NW.Hnd" "W.Hnd" "Fold"
```

```
colnames(shortdata) # but this also works
```

```
## [1] "Sex" "Wr.Hnd" "NW.Hnd" "W.Hnd" "Fold"
```

```
rownames(shortdata) # rows can also have names
```

```
## [1] "1" "2" "5" "6" "7" "8"
```

```
# Tip: Avoid row names.
# Add another variable to the data.frame to store this information.
```

```
# Examine first few rows
head(mydata) # first 6 rows
```

```
##      Sex Wr.Hnd NW.Hnd W.Hnd Fold Pulse Clap Exer Smoke Height
## 1 Female  18.5  18.0 Right R on L   92 Left Some Never 173.00
## 2 Male   19.5  20.5 Left  R on L  104 Left None Regul 177.80
## 5 Male   20.0  20.0 Right Neither 35 Right Some Never 165.00
## 6 Female  18.0  17.7 Right L on R   64 Right Some Never 172.72
## 7 Male   17.7  17.7 Right L on R   83 Right Freq Never 182.88
## 8 Female  17.0  17.3 Right R on L   74 Right Freq Never 157.00
##      M.I Age
## 1 Metric 18.250
## 2 Imperial 17.583
## 5 Metric 23.667
## 6 Imperial 21.000
## 7 Imperial 18.833
## 8 Metric 35.833
```

```
head(mydata, n = 10) # first 7 rows
```

```
##      Sex Wr.Hnd NW.Hnd W.Hnd Fold Pulse Clap Exer Smoke Height
## 1 Female  18.5  18.0 Right R on L   92 Left Some Never 173.00
## 2 Male   19.5  20.5 Left  R on L  104 Left None Regul 177.80
## 5 Male   20.0  20.0 Right Neither 35 Right Some Never 165.00
## 6 Female  18.0  17.7 Right L on R   64 Right Some Never 172.72
## 7 Male   17.7  17.7 Right L on R   83 Right Freq Never 182.88
## 8 Female  17.0  17.3 Right R on L   74 Right Freq Never 157.00
## 9 Male   20.0  19.5 Right R on L   72 Right Some Never 175.00
## 10 Male  18.5  18.5 Right R on L   90 Right Some Never 167.00
```

```
## 11 Female 17.0 17.2 Right L on R 80 Right Freq Never 156.20
## 14 Female 19.5 20.2 Right L on R 66 Neither Some Never 155.00
##      M.I    Age
## 1      Metric 18.250
## 2 Imperial 17.583
## 5      Metric 23.667
## 6 Imperial 21.000
## 7 Imperial 18.833
## 8      Metric 35.833
## 9      Metric 19.000
## 10     Metric 22.333
## 11 Imperial 28.500
## 14     Metric 17.500
```

```
tail(mydata) # last few rows
```

```
##      Sex Wr.Hnd NW.Hnd W.Hnd  Fold Pulse  Clap Exer Smoke Height
## 230  Male  18.6  19.6 Right L on R   71 Right Freq Occas  185.0
## 231 Female  18.8  18.5 Right R on L   80 Right Some Never  169.0
## 233 Female  18.0  18.0 Right L on R   85 Right Some Never  165.1
## 234 Female  18.5  18.0 Right L on R   88 Right Some Never  160.0
## 236  Male  21.0  21.5 Right R on L   90 Right Some Never  183.0
## 237 Female  17.6  17.3 Right R on L   85 Right Freq Never  168.5
##      M.I    Age
## 230      Metric 19.333
## 231      Metric 18.167
## 233 Imperial 17.667
## 234      Metric 16.917
## 236      Metric 17.167
## 237      Metric 17.750
```

```
# View(mydata) # Rstudio function to open data in viewer
# or click on the icon in the Environment pane
```

```
# How many rows and columns?
dim(mydata) # rows and column counts
```

```
## [1] 168 12
```

```
nrow(mydata) # row count
```

```
## [1] 168
```

```
ncol(mydata) # column count
```

```
## [1] 12
```

```
# Examine structure
str(mydata)
```

```
## 'data.frame': 168 obs. of 12 variables:
## $ Sex : Factor w/ 2 levels "Female","Male": 1 2 2 1 2 1 2 2 1 1 ...
## $ Wr.Hnd: num 18.5 19.5 20 18 17.7 17 20 18.5 17 19.5 ...
## $ NW.Hnd: num 18 20.5 20 17.7 17.7 17.3 19.5 18.5 17.2 20.2 ...
## $ W.Hnd : Factor w/ 2 levels "Left","Right": 2 1 2 2 2 2 2 2 2 2 ...
## $ Fold : Factor w/ 3 levels "L on R","Neither",...: 3 3 2 1 1 3 3 3 1 1 ...
## $ Pulse : int 92 104 35 64 83 74 72 90 80 66 ...
## $ Clap : Factor w/ 3 levels "Left","Neither",...: 1 1 3 3 3 3 3 3 3 2 ...
## $ Exer : Factor w/ 3 levels "Freq","None",...: 3 2 3 3 1 1 3 3 1 3 ...
## $ Smoke : Factor w/ 4 levels "Heavy","Never",...: 2 4 2 2 2 2 2 2 2 2 ...
## $ Height: num 173 178 165 173 183 ...
## $ M.I : Factor w/ 2 levels "Imperial","Metric": 2 1 2 1 1 2 2 2 1 2 ...
## $ Age : num 18.2 17.6 23.7 21 18.8 ...
## - attr(*, "na.action")=Class 'omit' Named int [1:69] 3 4 12 13 15 16 19 25 26 29 ...
## .. ..- attr(*, "names")= chr [1:69] "3" "4" "12" "13" ...
```

Getting help

```
# Use question mark (i.e., ?) followed by command name
# to lookup specific command
?mean
help(mean) # or use help function

# to look up package
help(package = "MASS")

# Press F1 in RStudio on the command name
# mean

# Use double question mark to do a full-text search on R help
??"factor analysis"

# Search google
# e.g., how to get the mean of a vector using r

# Ask question on Stackoverflow with the R tag
# http://stackoverflow.com/questions/tagged/r
```

Exercise 1

```
# 1. Working with vectors
# 1.1 Create a variable called x with 10 values drawn from a
# normal distribution (see rnorm)

# 1.2 Use the sum and > operator to work out how many values in x
# are larger than 1
```

```

# 3. Using the cats dataset in the MASS package
library(MASS)
data(cats)
# 3.1 Look up the help file on cats

# 3.2 How many observations are there?

# 3.3 Show the first 10 rows of the cats data.frame

# 3.4 Show the structure of cats using the str function

# 3.5 Extract the female cats and assign to variable fcats

# 3.6 How many rows is in fcats?

```

Answers 1

```

# 1. Working with vectors
# 1.1 Create a variable called x with 10 values drawn from a
#     normal distribution (see rnorm)
x <- rnorm(10)

# 1.2 Use the sum and > operator to work out how many values in x
#     are larger than 1
sum(x > 1)

```

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

```

# 3. Using the cats dataset in the MASS package
library(MASS)
data(cats)
# 3.1 Look up the help file on cats
?cats

# 3.2 How many observations are there?
nrow(cats)

```

```
## [1] 144
```

```

# 3.3 Show the first 10 rows of the cats data.frame
head(cats, 10)

```

```

##      Sex Bwt Hwt
## 1    F 2.0 7.0
## 2    F 2.0 7.4
## 3    F 2.0 9.5
## 4    F 2.1 7.2
## 5    F 2.1 7.3
## 6    F 2.1 7.6

```

```
## 7    F 2.1 8.1
## 8    F 2.1 8.2
## 9    F 2.1 8.3
## 10   F 2.1 8.5
```

```
# 3.4 Show the structure of cats using the str function
str(cats)
```

```
## 'data.frame':   144 obs. of  3 variables:
##  $ Sex: Factor w/ 2 levels "F","M": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Bwt: num   2 2 2 2.1 2.1 2.1 2.1 2.1 2.1 2.1 ...
##  $ Hwt: num   7 7.4 9.5 7.2 7.3 7.6 8.1 8.2 8.3 8.5 ...
```

```
# 3.5 Extract the female cats and assign to variable fcats
fcats <- cats[ cats$Sex == "F", ]
```

```
# 3.6 How many rows is in fcats?
nrow(fcats)
```

```
## [1] 47
```

Packages

```
# R has many additional packages
# To use a package it needs to be installed.
# You only need to install a package once.
# To use a package, you need to load the package each time
# you use R.
```

```
#####
# Installation
# Option 1. Use the install.packages function.
# install.packages("psych")
# Note that some packages rely on other packages.
# dependencies = TRUE ensures that dependencies are also installed.
# install.packages("psych", dependencies = TRUE)

# Option 2. Use the package tab in R Studio
# Click install and enter package details
```

```
#####
# Loading an installed package
# Option 1. Use the library function
library(psych) # I.e., put this at the start of your script
```



```

# Other options
# 2. We'll talk about ProjectTemplate later
#
# 3. Put it in your R startup file
#    (not recommended as it reduces reproducibility)

#####
# Common errors
# Not having a package installed is a common error
# If you try to load a package that is not installed.
# e.g.,
library(foo)

```

```
## Error in library(foo): there is no package called 'foo'
```

```

# You will get an error
# Error in library(foo) : there is no package called 'foo'
# This means:
# 1. You mistyped the name of the package, or
# 2. You need to install the package
#    install.packages("foo")
# Note foo is just an example. There is no package called foo.

# Not having a package loaded is a common error
# E.g., there is a function you want to use
detach(package:psych) # used for example to ensure psych is not attached
# say we wanted to use the fisherz function from the psych package
# but the psych package is not loaded
fisherz(.3)

```

```
## Error in eval(expr, envir, enclos): could not find function "fisherz"
```

```

# We get the error:
# "Error: object 'fisherz' not found"
# Thus we need to run
library(psych)

```

```

##
## Attaching package: 'psych'
##
## The following object is masked from 'package:Hmisc':
##
##     describe
##
## The following object is masked from 'package:ggplot2':
##
##     %+%

```

```
fisherz(.3)
```

```
## [1] 0.3095196
```

```
#####
# Packages contain additional functions.
# Once the package is loaded, functions are added to the workspace
# list workspace
search()
```

```
## [1] ".GlobalEnv"          "package:psych"
## [3] "package:MASS"         "package:Hmisc"
## [5] "package:ggplot2"      "package:Formula"
## [7] "package:survival"     "package:grid"
## [9] "package:lattice"      "package:ProjectTemplate"
## [11] "package:stats"        "package:graphics"
## [13] "package:grDevices"    "package:utils"
## [15] "package:datasets"     "package:methods"
## [17] "Autoloads"            "package:base"
```

```
# To make it clear that a function comes from a particular package
# or to overcome the issue where two packages have functions with the same names
# use double colon (i.e., package::function).
# RStudio also permits auto-completion of function names.
# psych::alpha() # alpha is a function in the psych package
```

Missing data

```
# Missing data is represented in R by NA
x <- c(1, 2, NA, 4)
y <- c("a", "b", NA, "c")
x
```

```
## [1] 1 2 NA 4
```

```
y
```

```
## [1] "a" "b" NA "c"
```

```
# To see whether a value is missing
is.na(x)
```

```
## [1] FALSE FALSE TRUE FALSE
```

```
# If you have missing data, some functions will return NA by default
# rather than returning a value
mean(x)
```

```
## [1] NA
```

```
sd(x)
```

```
## [1] NA
```

```
# Many functions have a na.rm argument
```

```
mean(x, na.rm=TRUE)
```

```
## [1] 2.333333
```

```
sd(x, na.rm=TRUE)
```

```
## [1] 1.527525
```

```
# or you remove the missing data
```

```
na.omit(x)
```

```
## [1] 1 2 4
```

```
## attr("na.action")
```

```
## [1] 3
```

```
## attr("class")
```

```
## [1] "omit"
```

```
mean(na.omit(x))
```

```
## [1] 2.333333
```

```
# na.omit also works on data frames performing listwise deletion
```

```
head(survey)
```

```
##      Sex Wr.Hnd NW.Hnd W.Hnd  Fold Pulse  Clap Exer Smoke Height
## 1 Female  18.5  18.0 Right R on L   92   Left Some Never 173.00
## 2 Male   19.5  20.5 Left  R on L  104   Left None Regul 177.80
## 3 Male   18.0  13.3 Right L on R   87 Neither None Occas    NA
## 4 Male   18.8  18.9 Right R on L   NA Neither None Never 160.00
## 5 Male   20.0  20.0 Right Neither  35   Right Some Never 165.00
## 6 Female  18.0  17.7 Right L on R   64   Right Some Never 172.72
##      M.I    Age
## 1 Metric 18.250
## 2 Imperial 17.583
## 3 <NA> 16.917
## 4 Metric 20.333
## 5 Metric 23.667
## 6 Imperial 21.000
```

```
dim(survey)
```

```
## [1] 237 12
```

```
cleaned_survey <- na.omit(survey)
dim(cleaned_survey)
```

```
## [1] 168 12
```

Getting summaries of data frames

```
library(MASS) # user survey data from MASS package
data(survey) # load an internal dataset
mydata <- survey
```

```
# Variable Names
names(mydata)
```

```
## [1] "Sex"      "Wr.Hnd" "NW.Hnd" "W.Hnd"  "Fold"   "Pulse"  "Clap"
## [8] "Exer"     "Smoke"  "Height" "M.I"    "Age"
```

```
# Show structure
str(mydata)
```

```
## 'data.frame': 237 obs. of 12 variables:
## $ Sex : Factor w/ 2 levels "Female","Male": 1 2 2 2 2 1 2 1 2 2 ...
## $ Wr.Hnd: num 18.5 19.5 18 18.8 20 18 17.7 17 20 18.5 ...
## $ NW.Hnd: num 18 20.5 13.3 18.9 20 17.7 17.7 17.3 19.5 18.5 ...
## $ W.Hnd : Factor w/ 2 levels "Left","Right": 2 1 2 2 2 2 2 2 2 ...
## $ Fold : Factor w/ 3 levels "L on R","Neither",...: 3 3 1 3 2 1 1 3 3 3 ...
## $ Pulse : int 92 104 87 NA 35 64 83 74 72 90 ...
## $ Clap : Factor w/ 3 levels "Left","Neither",...: 1 1 2 2 3 3 3 3 3 ...
## $ Exer : Factor w/ 3 levels "Freq","None",...: 3 2 2 2 3 3 1 1 3 3 ...
## $ Smoke : Factor w/ 4 levels "Heavy","Never",...: 2 4 3 2 2 2 2 2 2 ...
## $ Height: num 173 178 NA 160 165 ...
## $ M.I : Factor w/ 2 levels "Imperial","Metric": 2 1 NA 2 2 1 1 2 2 2 ...
## $ Age : num 18.2 17.6 16.9 20.3 23.7 ...
```

```
# Useful summary of numeric and categorical variables
Hmisc::describe(mydata)
```

```
## mydata
##
## 12 Variables      237 Observations
## -----
## Sex
##      n missing  unique
##    236      1      2
##
## Female (118, 50%), Male (118, 50%)
## -----
## Wr.Hnd
```

```

##      n missing  unique    Info    Mean    .05    .10    .25    .50
##    236         1      60      1  18.67  16.00  16.50  17.50  18.50
##      .75      .90      .95
##    19.80    21.15    22.05
##
## lowest : 13.0 14.0 15.0 15.4 15.5, highest: 22.5 22.8 23.0 23.1 23.2
## -----
## NW.Hnd
##      n missing  unique    Info    Mean    .05    .10    .25    .50
##    236         1      68      1  18.58  15.50  16.30  17.50  18.50
##      .75      .90      .95
##    19.72    21.00    22.22
##
## lowest : 12.5 13.0 13.3 13.5 15.0, highest: 22.7 23.0 23.2 23.3 23.5
## -----
## W.Hnd
##      n missing  unique
##    236         1      2
##
## Left (18, 8%), Right (218, 92%)
## -----
## Fold
##      n missing  unique
##    237         0      3
##
## L on R (99, 42%), Neither (18, 8%), R on L (120, 51%)
## -----
## Pulse
##      n missing  unique    Info    Mean    .05    .10    .25    .50
##    192         45      43      1  74.15  59.55  60.00  66.00  72.50
##      .75      .90      .95
##    80.00    90.00    92.00
##
## lowest : 35 40 48 50 54, highest: 96 97 98 100 104
## -----
## Clap
##      n missing  unique
##    236         1      3
##
## Left (39, 17%), Neither (50, 21%), Right (147, 62%)
## -----
## Exer
##      n missing  unique
##    237         0      3
##
## Freq (115, 49%), None (24, 10%), Some (98, 41%)
## -----
## Smoke
##      n missing  unique
##    236         1      4
##
## Heavy (11, 5%), Never (189, 80%), Occas (19, 8%)
## Regul (17, 7%)
## -----

```

```
## Height
##      n missing  unique    Info    Mean    .05    .10    .25    .50
##    209      28     67      1  172.4  157.0  160.0  165.0  171.0
##      .75     .90     .95
##    180.0  185.4  189.6
##
## lowest : 150.0 152.0 152.4 153.5 154.9
## highest: 191.8 193.0 195.0 196.0 200.0
## -----
## M.I
##      n missing  unique
##    209      28     2
##
## Imperial (68, 33%), Metric (141, 67%)
## -----
## Age
##      n missing  unique    Info    Mean    .05    .10    .25    .50
##    237      0     88      1  20.37  17.08  17.22  17.67  18.58
##      .75     .90     .95
##    20.17  23.58  30.68
##
## lowest : 16.75 16.92 17.00 17.08 17.17
## highest: 41.58 43.83 44.25 70.42 73.00
## -----
```

```
# Common univariate statistics for numeric variables
psych::describe(mydata)
```

```
##      vars  n  mean  sd median trimmed  mad  min  max range skew
## Sex*    1 236   NaN   NA    NA    NaN   NA   Inf -Inf -Inf  NA
## Wr.Hnd   2 236 18.67 1.88 18.50 18.61 1.48 13.00 23.2 10.20 0.18
## NW.Hnd   3 236 18.58 1.97 18.50 18.55 1.63 12.50 23.5 11.00 0.02
## W.Hnd*   4 236   NaN   NA    NA    NaN   NA   Inf -Inf -Inf  NA
## Fold*    5 237   NaN   NA    NA    NaN   NA   Inf -Inf -Inf  NA
## Pulse    6 192 74.15 11.69 72.50 74.02 11.12 35.00 104.0 69.00 -0.02
## Clap*    7 236   NaN   NA    NA    NaN   NA   Inf -Inf -Inf  NA
## Exer*    8 237   NaN   NA    NA    NaN   NA   Inf -Inf -Inf  NA
## Smoke*   9 236   NaN   NA    NA    NaN   NA   Inf -Inf -Inf  NA
## Height  10 209 172.38 9.85 171.00 172.19 10.08 150.00 200.0 50.00 0.22
## M.I*    11 209   NaN   NA    NA    NaN   NA   Inf -Inf -Inf  NA
## Age     12 237 20.37 6.47 18.58 18.99 1.61 16.75 73.0 56.25 5.16
##      kurtosis  se
## Sex*          NA  NA
## Wr.Hnd        0.30 0.12
## NW.Hnd        0.44 0.13
## W.Hnd*        NA  NA
## Fold*         NA  NA
## Pulse         0.33 0.84
## Clap*         NA  NA
## Exer*         NA  NA
## Smoke*        NA  NA
## Height       -0.44 0.68
## M.I*          NA  NA
## Age          33.47 0.42
```

```
summary(mydata)
```

```
##      Sex      Wr.Hnd      NW.Hnd      W.Hnd      Fold
## Female:118  Min.   :13.00  Min.   :12.50  Left  : 18  L on R : 99
## Male   :118  1st Qu.:17.50  1st Qu.:17.50  Right:218  Neither: 18
## NA's   : 1   Median :18.50  Median :18.50  NA's : 1   R on L :120
##                               Mean   :18.67  Mean   :18.58
##                               3rd Qu.:19.80  3rd Qu.:19.73
##                               Max.   :23.20  Max.   :23.50
##                               NA's   :1     NA's   :1
##      Pulse      Clap      Exer      Smoke      Height
## Min.   : 35.00  Left   : 39  Freq:115  Heavy: 11  Min.   :150.0
## 1st Qu.: 66.00  Neither: 50  None: 24  Never:189  1st Qu.:165.0
## Median : 72.50  Right  :147  Some: 98  Occas: 19  Median :171.0
## Mean   : 74.15  NA's   : 1   Regul: 17  Mean   :172.4
## 3rd Qu.: 80.00                NA's : 1   3rd Qu.:180.0
## Max.   :104.00                NA's : 1   Max.   :200.0
## NA's   :45                  NA's   :28
##      M.I      Age
## Imperial: 68  Min.   :16.75
## Metric   :141  1st Qu.:17.67
## NA's     : 28  Median :18.58
##                               Mean   :20.37
##                               3rd Qu.:20.17
##                               Max.   :73.00
##
```

Summaries of numeric vectors (or data frame variables)

```
x <- c(1, 2, 3, 4,5)
```

```
# Total
sum(x) # sum of vector
```

```
## [1] 15
```

```
prod(x) # product of vector
```

```
## [1] 120
```

```
# Central tendency
mean(x) # mean of vector
```

```
## [1] 3
```

```
median(x) # median of vector
```

```
## [1] 3
```

```
length(x) # length of vector
```

```
## [1] 5
```

```
# Spread  
sd(x) # standard deviation
```

```
## [1] 1.581139
```

```
var(x) # variance
```

```
## [1] 2.5
```

```
range(x) # min and max of vector
```

```
## [1] 1 5
```

```
min(x) # minimum of vector
```

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

```
max(x) # max of vector
```

```
## [1] 5
```

```
# Other distributional features  
psych::skew(x) # skewness
```

```
## [1] 0
```

```
psych::kurtosi(x) # kurtosis
```

```
## [1] -1.912
```

```
dat <- data.frame(x = c(1, 2, 3, 4, 5),  
                  y = c(0, 0, 1, 1, 1))  
dat
```

```
##   x y  
## 1 1 0  
## 2 2 0  
## 3 3 1  
## 4 4 1  
## 5 5 1
```



```
# Vector operations typically operate element wise
dat$z <- dat$x + dat$y
dat
```

```
##   x y z
## 1 1 0 1
## 2 2 0 2
## 3 3 1 4
## 4 4 1 5
## 5 5 1 6
```

```
dat$z <- dat$x * dat$y
dat
```

```
##   x y z
## 1 1 0 0
## 2 2 0 0
## 3 3 1 3
## 4 4 1 4
## 5 5 1 5
```

```
# A single value is recycled through the vector
dat$z <- dat$x + 10
dat
```

```
##   x y z
## 1 1 0 11
## 2 2 0 12
## 3 3 1 13
## 4 4 1 14
## 5 5 1 15
```

Exercise 2 - Data.frames

```
# For this exercise will use the GSS7402 dataset
library(AER)
```

```
## Loading required package: car
##
## Attaching package: 'car'
##
## The following object is masked from 'package:psych':
##
##   logit
##
## Loading required package: lmtest
## Loading required package: zoo
##
## Attaching package: 'zoo'
```

```
##
## The following objects are masked from 'package:base':
##
##   as.Date, as.Date.numeric
##
## Loading required package: sandwich

help(package = AER)
data("GSS7402")
?GSS7402 # to learn about the dataset
# It might be easier to work with a shorter variable name
gss <- GSS7402

# 1. List the variable names in the gss dataset

# 2. Show the first few rows (hint: the head) of the dataset?

# 3. How many cases are there?

# 4. What is the mean, sd, and range age of the sample

# 5. Use the psych and Hmisc describe functions to describe the samples

# 6. Extract a data.frame with only people over the age of 80

# 7. Get the mean number of children ("kids") for participants
#    over the age of 80

# 8. Use the mean function to get the mean age at first birth.
#    Hint: there is missing data.
```

Answers Exercise 2 - Data.frames

```
# For this exercise will use the GSS7402 dataset
library(AER)
help(package = AER)
data("GSS7402")
?GSS7402 # to learn about the dataset
# It might be easier to work with a shorter variable name
gss <- GSS7402

# 1. List the variable names in the gss dataset
names(gss)
```

```
## [1] "kids"          "age"           "education"     "year"
## [5] "siblings"      "agefirstbirth" "ethnicity"     "city16"
## [9] "lowincome16"   "immigrant"
```

```
# 2. Show the first few rows (hint: the head) of the dataset?
head(gss)
```

```
## kids age education year siblings agefirstbirth ethnicity city16
## 1 0 25 14 2002 1 NA cauc no
## 2 1 30 13 2002 4 19 cauc yes
## 3 1 55 2 2002 1 27 cauc no
## 4 2 57 16 2002 1 22 cauc no
## 5 2 71 12 2002 6 29 cauc yes
## 6 0 19 13 2002 1 NA other yes
## lowincome16 immigrant
## 1 no no
## 2 no no
## 3 no yes
## 4 no no
## 5 no no
## 6 no no
```

3. How many cases are there?

```
nrow(gss)
```

```
## [1] 9120
```

4. What is the mean, sd, and range age of the sample

```
mean(gss$age)
```

```
## [1] 46.08202
```

```
sd(gss$age)
```

```
## [1] 17.92389
```

```
range(gss$age)
```

```
## [1] 18 89
```

5. Use the psych and Hmisc describe functions to describe the samples

```
psych::describe(gss)
```

```
## vars n mean sd median trimmed mad min max range
## kids 1 9120 2.08 1.81 2 1.86 1.48 0 8 8
## age 2 9120 46.08 17.92 43 44.94 19.27 18 89 71
## education 3 9120 12.64 2.96 12 12.70 2.97 0 20 20
## year 4 9120 1990.29 9.10 1994 1990.79 11.86 1974 2002 28
## siblings 5 9120 4.05 3.25 3 3.60 2.97 0 35 35
## agefirstbirth 6 3312 22.63 4.86 22 22.18 4.45 9 42 33
## ethnicity* 7 9120 NaN NA NA NaN NA Inf -Inf -Inf
## city16* 8 9120 NaN NA NA NaN NA Inf -Inf -Inf
## lowincome16* 9 9120 NaN NA NA NaN NA Inf -Inf -Inf
## immigrant* 10 9120 NaN NA NA NaN NA Inf -Inf -Inf
## skew kurtosis se
## kids 1.00 1.03 0.02
## age 0.48 -0.78 0.19
```

```
## education      -0.26      1.03 0.03
## year           -0.36     -1.16 0.10
## siblings        1.67      4.78 0.03
## agefirstbirth  0.87       0.59 0.08
## ethnicity*      NA        NA   NA
## city16*         NA        NA   NA
## lowincome16*    NA        NA   NA
## immigrant*      NA        NA   NA
```

```
Hmisc::describe(gss)
```

```
## gss
##
## 10 Variables      9120 Observations
## -----
## kids
##      n missing  unique    Info    Mean
##    9120      0      9    0.96    2.076
##
##           0   1   2   3   4   5   6   7   8
## Frequency 2127 1544 2338 1474 790 376 208 100 163
## %          23   17   26   16   9   4   2   1   2
## -----
## age
##      n missing  unique    Info    Mean    .05    .10    .25    .50
##    9120      0      72     1    46.08    22    25    31    43
##      .75    .90    .95
##      59     73     79
##
## lowest : 18 19 20 21 22, highest: 85 86 87 88 89
## -----
## education
##      n missing  unique    Info    Mean    .05    .10    .25    .50
##    9120      0      21    0.96    12.64     8     9    12    12
##      .75    .90    .95
##      14     16     18
##
## lowest : 0 1 2 3 4, highest: 16 17 18 19 20
## -----
## year
##      n missing  unique    Info    Mean
##    9120      0      8    0.98    1990
##
##           1974 1978 1982 1986 1990 1994 1998 2002
## Frequency  785  877 1064  842  767 1688 1580 1517
## %           9   10   12    9    8   19   17   17
## -----
## siblings
##      n missing  unique    Info    Mean    .05    .10    .25    .50
##    9120      0      27    0.98    4.051     1     1     2     3
##      .75    .90    .95
##      6      8     10
##
## lowest : 0 1 2 3 4, highest: 22 23 25 27 35
```

```
## -----
## agefirstbirth
##      n missing  unique    Info    Mean    .05    .10    .25    .50
##    3312    5808     33    0.99    22.63     16     17     19     22
##      .75     .90     .95
##      25      30      32
##
## lowest :  9 11 12 13 14, highest: 38 39 40 41 42
## -----
## ethnicity
##      n missing  unique
##    9120         0       2
##
## other (1785, 20%), cauc (7335, 80%)
## -----
## city16
##      n missing  unique
##    9120         0       2
##
## no (5246, 58%), yes (3874, 42%)
## -----
## lowincome16
##      n missing  unique
##    9120         0       2
##
## no (7182, 79%), yes (1938, 21%)
## -----
## immigrant
##      n missing  unique
##    9120         0       2
##
## no (8122, 89%), yes (998, 11%)
## -----
```

```
# 6. Extract a data.frame with only people over the age of 80
gss_over80 <- gss[ gss$age > 80, ]
```

```
# 7. Get the mean number of children ("kids") for participants
#    over the age of 80
mean(gss[ gss$age > 80, "kids"])
```

```
## [1] 2.394737
```

```
# 8. Use the mean function to get the mean age at first birth.
#    Hint: there is missing data.
mean(gss$agefirstbirth) # doesn't work because there is missing data
```

```
## [1] NA
```

```
mean(gss$agefirstbirth, na.rm = TRUE) # doesn't work because there is missing data
```

```
## [1] 22.63074
```

String functions

```
paste("hello", "how", "are", "You") # defaults to space separator
```

```
## [1] "hello how are You"
```

```
paste0("hello", "how", "are", "You") # no separator
```

```
## [1] "hellohowareYou"
```

```
paste("apple", "banana", "carrot", "date", sep = ", ") # specify arbitrary separator
```

```
## [1] "apple, banana, carrot, date"
```

```
paste0("v", 1:10) # paticularly useful with vectors
```

```
## [1] "v1" "v2" "v3" "v4" "v5" "v6" "v7" "v8" "v9" "v10"
```

```
# Extract substring
```

```
substr("abcdefghijklmnop", 4, 6)
```

```
## [1] "def"
```

```
# Change case
```

```
toupper("abcd") # make upper case
```

```
## [1] "ABCD"
```

```
tolower("ABCD") # make lower case
```

```
## [1] "abcd"
```

```
mystring <- c("apple", "banana", "carrot", "date", "egg", "fig")
```

```
# Identify which strings match a pattern
```

```
grep("a", mystring) # index of objects with "a"
```

```
## [1] 1 2 3 4
```

```
grep("a", mystring, value = TRUE) # value of objects with "a"
```

```
## [1] "apple" "banana" "carrot" "date"
```

```
# get count of number of characters
```

```
nchar(mystring)
```

```
## [1] 5 6 6 4 3 3
```

```
data.frame(mystring, nchar(mystring))
```

```
##   mystring nchar.mystring.  
## 1   apple           5  
## 2  banana           6  
## 3  carrot           6  
## 4    date           4  
## 5    egg            3  
## 6    fig            3
```

```
# Substitute a mystringreplacement text that matches a pattern  
questions <- c("How are you?", "What is going on?")  
gsub(" ", "_", questions) # replace space with underscore
```

```
## [1] "How_are_you?"      "What_is_going_on?"
```

```
# R string manipulation tools are very powerful  
# For more information see  
?grep  
?"regular expression"
```

```
# see also Hadley Wickham's package for string manipulation  
# It attempts to introduce greater consistency in notation.  
# install.packages("stringr")  
library(stringr)  
help(package = "stringr")  
# all functions begin with str_  
str_length(mystring) # see nchar
```

```
## [1] 5 6 6 4 3 3
```

```
str_sub(mystring, start = 1, end = 3)
```

```
## [1] "app" "ban" "car" "dat" "egg" "fig"
```

```
# writing output to the console  
cat("Hello World!")
```

```
## Hello World!
```

```
# Tab is \t and new line is \n  
cat("Hello\t World\nSome more text")
```

```
## Hello      World  
## Some more text
```

Importing data

```
# A simple option is to export data from your external data
# in csv format and then import the data using csv
# csv
medals <- read.csv("data/practice/medals.csv")
head(medals)
```

```
##   Year   City   Sport   Discipline NOC   Event Event.gender
## 1 1924 Chamonix   Skating Figure skating AUT   individual      M
## 2 1924 Chamonix   Skating Figure skating AUT   individual      W
## 3 1924 Chamonix   Skating Figure skating AUT   pairs          X
## 4 1924 Chamonix Bobsleigh Bobsleigh BEL   four-man      M
## 5 1924 Chamonix Ice Hockey Ice Hockey CAN   ice hockey      M
## 6 1924 Chamonix Biathlon Biathlon FIN military patrol      M
##   Medal
## 1 Silver
## 2  Gold
## 3  Gold
## 4 Bronze
## 5  Gold
## 6 Silver
```

```
tail(medals)
```

```
##      Year City Sport Discipline NOC   Event Event.gender Medal
## 2306 2006 Turin Skiing Snowboard USA   Half-pipe      M   Gold
## 2307 2006 Turin Skiing Snowboard USA   Half-pipe      M Silver
## 2308 2006 Turin Skiing Snowboard USA   Half-pipe      W   Gold
## 2309 2006 Turin Skiing Snowboard USA   Half-pipe      W Silver
## 2310 2006 Turin Skiing Snowboard USA Snowboard Cross      M   Gold
## 2311 2006 Turin Skiing Snowboard USA Snowboard Cross      W Silver
```

```
dim(medals)
```

```
## [1] 2311    8
```

```
# Other delimited formats
medals <- read.table("data/practice/medals.tsv", sep = "\t")

# Read Excel:
# Read xls files using xls
# Requires that Perl is installed and on the path
# You may need to install Perl if on Windows
# https://www.perl.org/get.html
# library(gdata)
# medals <- gdata::read.xls("data/practice/medals.xls")

# requires Java
# library(xlsx)
# x <- xlsx::read.xlsx("data/practice/medals.xlsx", sheetIndex = 1)

# More recent package that has no dependencies on external packages
```



```

# readxl
library(readxl)
medals <- readxl::read_excel("data/practice/medals.xls")
medals <- readxl::read_excel("data/practice/medals.xlsx")

# SPSS
library(foreign)
cas <- foreign::read.spss("data/practice/cas.sav", to.data.frame = TRUE)

## Warning in foreign::read.spss("data/practice/cas.sav", to.data.frame
## = TRUE): data/practice/cas.sav: Unrecognized record type 7, subtype 18
## encountered in system file

## Warning in foreign::read.spss("data/practice/cas.sav", to.data.frame
## = TRUE): data/practice/cas.sav: Unrecognized record type 7, subtype 24
## encountered in system file

```

```
attr(cas, "variable.labels")
```

```

##                district
##                "District code"
##                school
##                "School name"
##                county
##                "County"
##                grades
##                "grade span of district"
##                students
##                "Total enrollment"
##                teachers
##                "Number of teachers"
##                calworks
## "Percent qualifying for CalWorks (income assistance)"
##                lunch
##                "Percent qualifying for reduced-price lunch"
##                computer
##                "Number of computers"
##                expenditure
##                "Expenditure per student"
##                income
##                "District average income (in USD 1,000)"
##                english
##                "Percent of English learners"
##                read
##                "Average reading score"
##                math
##                "Average math score"

```

```

# tip: You may need to think about value labels in your SPSS file
# Specifically, if you have numeric variables that have variable labels, you may
# want to remove the value labels in SPSS or

```

```
# import stata
?read.dta
?read.sas
```

```
## No documentation for 'read.sas' in specified packages and libraries:
## you could try '??read.sas'
```

```
# General purpose packages
# The haven package also can read and write SPSS, SAS, and Stata files

# rio package: General purpose import and export
# General purpose import and export tools
# It's a fairly new package so there may still be a few bugs.
# https://github.com/leeper/rio
# github version is currently a little bit ahead of the CRAN version
library(rio)
cas <- rio::import(file="data/practice/cas.sav")
rio::export(cas, file="output/cas.xlsx")
medals <- rio::import(file="data/practice/medals.csv")

# Use ProjectTemplate to auto-import (see discussion later)
```

Exporting data

```
mydata <- data.frame(a = c(1,2,3), b = c("a", "b", "c"))

# Internal R format
# Good option if you need to re-open data in R
save(mydata, file="output/mydata.rdata")
# load("output/mydata.rdata")

# csv
# Good option if you need to get data into other software
# This should open in almost all other software (e.g. Excel, SPSS, etc.)
write.csv(mydata, file = "output/mydata.csv")
write.csv(mydata, file = "output/mydata-2.csv", row.names = FALSE) # exclude row.names

# If you need more flexibility in terms of delimiters, etc.
write.table(mydata, file = "output/mydata.tsv", sep = "\t") # e.g., tab delimiter

# Exporting to other formats
# There are a range of options for exporting to other formats
# Functionality is often spread around
# Given that the csv option is usually sufficient
library(foreign)
?foreign::write.foreign # options for exporting to SAS, SPSS, and Stata directly
?rio::export
```

Exercise 3

```
# 1. Open medals.csv in the data/practice/ directory
#    and assign to variable medals

# 2. Check that the file imported correctly
#    (a) look at the first few rows,
#    (b) look at the last few rows,
#    (b) check the structure (i.e., str),
#    (c) Use the Hmisc describe function to check basic properties

# 3. Create a new variable in medals that indicates
#    whether the medals was Gold (TRUE) or Silver/Bronze (FALSE)
#    and call it isgold

# 4. How the number of sum of gold medals

# 5. Export the medals data.frame to the output folder
#    (a) as a csv file
#    (b) as a native rdata file

# 6. Remove the medals dataset from the workspace
#    and then load it again from the csv file.
#    Check that it imported correctly.
# Then remove medals and repeat for the rdata file
```

Answers for Exercise 3

```
# 1. Open medals.csv in the data/practice/ directory
#    and assign to variable medals
medals <- read.csv("data/practice/medals.csv")

# 2. Check that the file imported correctly
#    (a) look at the first few rows,
#    (b) look at the last few rows,
#    (b) check the structure (i.e., str),
#    (c) Use the Hmisc describe function to check basic properties
head(medals)
```

```
##   Year   City      Sport      Discipline NOC      Event Event.gender
## 1 1924 Chamonix   Skating Figure skating AUT      individual      M
## 2 1924 Chamonix   Skating Figure skating AUT      individual      W
## 3 1924 Chamonix   Skating Figure skating AUT      pairs        X
## 4 1924 Chamonix Bobsleigh      Bobsleigh BEL      four-man      M
## 5 1924 Chamonix Ice Hockey      Ice Hockey CAN      ice hockey      M
## 6 1924 Chamonix Biathlon      Biathlon FIN military patrol      M
##   Medal
## 1 Silver
## 2  Gold
```

```
## 3    Gold
## 4 Bronze
## 5    Gold
## 6 Silver
```

```
tail(medals)
```

```
##      Year City Sport Discipline NOC      Event Event.gender Medal
## 2306 2006 Turin Skiing  Snowboard USA      Half-pipe      M    Gold
## 2307 2006 Turin Skiing  Snowboard USA      Half-pipe      M Silver
## 2308 2006 Turin Skiing  Snowboard USA      Half-pipe      W    Gold
## 2309 2006 Turin Skiing  Snowboard USA      Half-pipe      W Silver
## 2310 2006 Turin Skiing  Snowboard USA Snowboard Cross      M    Gold
## 2311 2006 Turin Skiing  Snowboard USA Snowboard Cross      W Silver
```

```
str(medals)
```

```
## 'data.frame':    2311 obs. of  8 variables:
## $ Year      : int  1924 1924 1924 1924 1924 1924 1924 1924 1924 ...
## $ City      : chr   "Chamonix" "Chamonix" "Chamonix" "Chamonix" ...
## $ Sport     : chr   "Skating" "Skating" "Skating" "Bobsleigh" ...
## $ Discipline : chr   "Figure skating" "Figure skating" "Figure skating" "Bobsleigh" ...
## $ NOC       : chr   "AUT" "AUT" "AUT" "BEL" ...
## $ Event     : chr   "individual" "individual" "pairs" "four-man" ...
## $ Event.gender: chr   "M" "W" "X" "M" ...
## $ Medal     : chr   "Silver" "Gold" "Gold" "Bronze" ...
```

```
Hmisc::describe(medals)
```

```
## medals
##
## 8 Variables      2311 Observations
## -----
## Year
##      n missing  unique    Info    Mean    .05    .10    .25    .50
##    2311      0     20      1    1980    1932    1948    1968    1988
##      .75     .90     .95
##    1998     2006     2006
##
## lowest : 1924 1928 1932 1936 1948, highest: 1992 1994 1998 2002 2006
## -----
## City
##      n missing  unique
##    2311      0     17
##
##      Albertville Calgary Chamonix Cortina d'Ampezzo
## Frequency      171     138     49              72
## %              7       6       2              3
##
##      Garmisch-Partenkirchen Grenoble Innsbruck Lake Placid
## Frequency           51     106     214     157
## %              2       5       9       7
##
##      Lillehammer Nagano Oslo Salt Lake City Sapporo Sarajevo
```

```

## Frequency      183      205      67          234      105      117
## %              8       9       3          10       5       5
##              Squaw Valley St. Moritz Turin
## Frequency      81          109      252
## %              4           5       11
## -----
## Sport
##      n missing  unique
## 2311         0       7
##
##      Biathlon Bobsleigh Curling Ice Hockey Luge Skating Skiing
## Frequency      162          133      21          69      108      758      1060
## %              7           6       1           3       5       33       46
## -----
## Discipline
##      n missing  unique
## 2311         0       15
##
## Alpine Skiing (367, 16%), Biathlon (162, 7%)
## Bobsleigh (115, 5%), Cross Country S (399, 17%)
## Curling (21, 1%), Figure skating (207, 9%)
## Freestyle Ski. (54, 2%), Ice Hockey (69, 3%)
## Luge (108, 5%), Nordic Combined (84, 4%)
## Short Track S. (96, 4%), Skeleton (18, 1%)
## Ski Jumping (114, 5%), Snowboard (42, 2%)
## Speed skating (455, 20%)
## -----
## NOC
##      n missing  unique
## 2311         0       45
##
## lowest : AUS AUT BEL BLR BUL, highest: UKR URS USA UZB YUG
## -----
## Event
##      n missing  unique
## 2311         0       67
##
## lowest : 10000m          1000m          10km          10km pursuit          12,5km mass start
## highest: super-G          Team          Team pursuit          Team sprint          two-man
## -----
## Event.gender
##      n missing  unique
## 2311         0       3
##
## M (1386, 60%), W (802, 35%), X (123, 5%)
## -----
## Medal
##      n missing  unique
## 2311         0       3
##
## Bronze (764, 33%), Gold (774, 33%), Silver (773, 33%)
## -----

```

```
# 3. Create a new variable in medals that indicates
#   whether the medals was Gold (TRUE) or Silver/Bronze (FALSE)
#   and call it isgold
medals$isgold <- medals$Medal == "Gold"
```

```
# 4. How the number of sum of gold medals
sum(medals$isgold)
```

```
## [1] 774
```

```
# 5. Export the medals data.frame to the output folder
#   (a) as a csv file
#   (b) as a native rdata file
write.csv(medals, "output/medals.csv")
# or technically you may want to do
write.csv(medals, "output/medals.csv", row.names = FALSE)
save(medals, file = "output/medals.rdata")
```

```
# 6. Remove the medals dataset from the workspace
#   and then load it again from the csv file.
#   Check that it imported correctly.
# Then remove medals and repeat for the rdata file
rm(medals)
medals <- read.csv("output/medals.csv")
head(medals)
```

```
##   Year   City      Sport   Discipline NOC      Event Event.gender
## 1 1924 Chamonix   Skating Figure skating AUT      individual      M
## 2 1924 Chamonix   Skating Figure skating AUT      individual      W
## 3 1924 Chamonix   Skating Figure skating AUT      pairs        X
## 4 1924 Chamonix Bobsleigh Bobsleigh BEL      four-man      M
## 5 1924 Chamonix Ice Hockey Ice Hockey CAN      ice hockey      M
## 6 1924 Chamonix Biathlon Biathlon FIN military patrol      M
##   Medal isgold
## 1 Silver FALSE
## 2   Gold  TRUE
## 3   Gold  TRUE
## 4 Bronze FALSE
## 5   Gold  TRUE
## 6 Silver FALSE
```

```
rm(medals)
load("output/medals.rdata")
head(medals)
```

```
##   Year   City      Sport   Discipline NOC      Event Event.gender
## 1 1924 Chamonix   Skating Figure skating AUT      individual      M
## 2 1924 Chamonix   Skating Figure skating AUT      individual      W
## 3 1924 Chamonix   Skating Figure skating AUT      pairs        X
## 4 1924 Chamonix Bobsleigh Bobsleigh BEL      four-man      M
## 5 1924 Chamonix Ice Hockey Ice Hockey CAN      ice hockey      M
## 6 1924 Chamonix Biathlon Biathlon FIN military patrol      M
```

```
##      Medal isgold
## 1 Silver  FALSE
## 2   Gold   TRUE
## 3   Gold   TRUE
## 4 Bronze  FALSE
## 5   Gold   TRUE
## 6 Silver  FALSE
```

Random variables and distributions

```
# In statistics, we often want to generate random data with certain properties
# or looking up features of statistical distributions.
# See the following help for list of common distributions in base R
?Distributions

# and see http://cran.r-project.org/web/views/Distributions.html for many more distributions

# Each distribution has four functions that differ in terms of the first letter
# For example, for the normal distribution, you have
dnorm(1) # Density of the value 1 of a standard normal distribution
```

```
## [1] 0.2419707
```

```
pnorm(1) # Cumulative distribution function for value of 1 on standard normal distribution
```

```
## [1] 0.8413447
```

```
qnorm(.975) # Inverse cumulative distribution function for value of .975
```

```
## [1] 1.959964
```

```
rnorm(5) # Generate 5 random draws from normal distribution
```

```
## [1] 0.6958336 -0.7036630 0.8755151 -1.3879961 -1.3162740
```

```
#
dunif(1) # Density of the value 1 of a uniform distribution (0, 1)
```

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

```
punif(.5) # Cumulative distribution function for value of 1 on uniform distribution
```

```
## [1] 0.5
```

```
qunif(.975) # Inverse cumulative distribution function for value of .975
```

```
## [1] 0.975
```

```
runif(5) # Generate 5 random draws from uniform distribution
```

```
## [1] 0.766837264 0.144183603 0.009785924 0.408310513 0.313770549
```

```
# Distributions have parameters that can be specified
```

```
x <- rnorm(10, mean = 10, sd = 1) # draw 10 from mean of 10
```

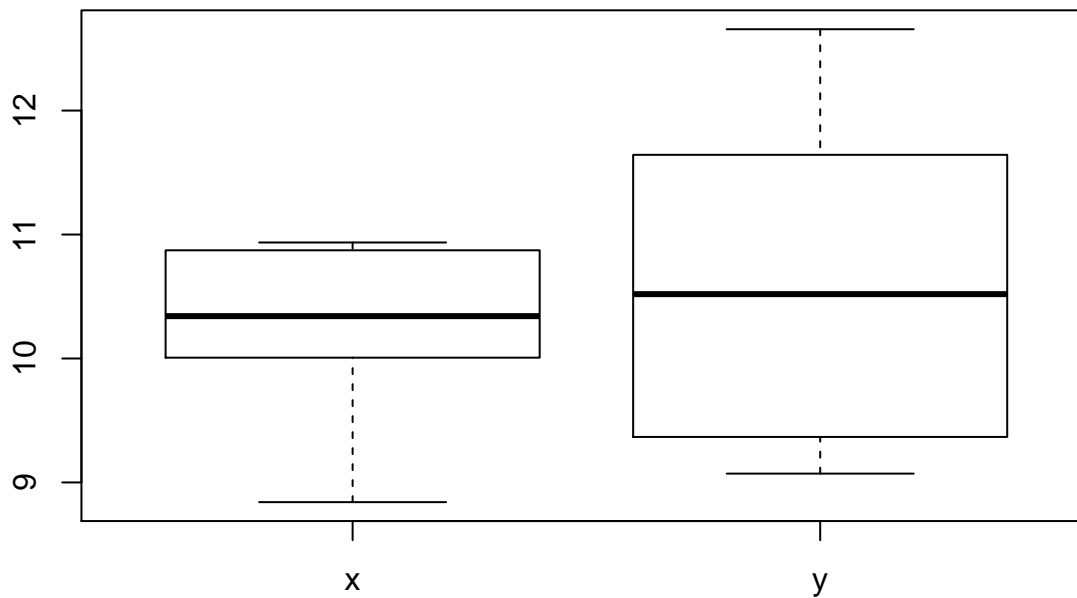
```
y <- rnorm(10, mean = 11, sd = 1) # draw 10 from mean of 11
```

```
dat <- data.frame(x=x, y=y)
```

```
dat
```

```
##           x           y
## 1  10.006721  10.302188
## 2   9.905732  12.656132
## 3  10.328294  10.735812
## 4   8.841139  10.078750
## 5  10.135187   9.236849
## 6  10.935702  11.855231
## 7  10.933639   9.367044
## 8  10.354740  11.642860
## 9  10.872766   9.071333
## 10 10.603060  10.857407
```

```
boxplot(dat)
```



Functions

```
# You can write functions and these are generally the same as  
# the functions you use in R
```

```
# For example, I could create a function that printed some text
```



```
print_some_text <- function(x = "Hello World") {  
  print(x)  
}
```

```
# If I run the above command, I can then use it  
print_some_text() # using the default argument
```

```
## [1] "Hello World"
```

```
print_some_text("blah blah blah") # or to print some other text
```

```
## [1] "blah blah blah"
```

```
# Anatomy of a function  
# Functions have a name  
# They take one or more arguments  
# Arguments may have default values
```

```
# Let's take a more interesting example: Power analysis  
# The following data simulates data for two groups and  
# examines whether there is a significant difference at .05  
# It repeats the process 1000 times and calculates the  
# proportion of times it is statistically significant  
# (i.e., simulation estimate of the statistical power)
```

```
significant <- NULL  
for (i in 1:1000) {  
  x <- rnorm(30, mean = 0.0, sd = 1)  
  y <- rnorm(30, mean = 0.3, sd = 1)  
  fit <- t.test(x, y)  
  fit  
  significant[i] <- (fit$p.value < .05)  
}  
statistical_power <- mean(significant)  
statistical_power
```

```
## [1] 0.209
```

```
# we could convert this to a function  
power_group_dif1 <- function() {  
  significant <- NULL  
  for (i in 1:1000) {  
    x <- rnorm(30, mean = 0.0, sd = 1)  
    y <- rnorm(30, mean = 0.3, sd = 1)  
    fit <- t.test(x, y)  
    fit  
    significant[i] <- (fit$p.value < .05)  
  }  
  statistical_power <- mean(significant)  
  statistical_power  
}
```

```
}
```

```
power_group_dif1()
```

```
## [1] 0.218
```

```
# but the beauty of function is that they can make things general  
# Let's make the mean of group 2 an argument that can be specified
```

```
power_group_dif2 <- function(mean2 = 0.3) {  
  significant <- NULL  
  for (i in 1:1000) {  
    x <- rnorm(30, mean = 0.0, sd = 1)  
    y <- rnorm(30, mean = mean2, sd = 1)  
    fit <- t.test(x, y)  
    fit  
    significant[i] <- (fit$p.value < .05)  
  }  
  statistical_power <- mean(significant)  
  statistical_power  
}
```

```
# now we can specify different values  
power_group_dif2(0)
```

```
## [1] 0.055
```

```
power_group_dif2(.3)
```

```
## [1] 0.224
```

```
power_group_dif2(.5)
```

```
## [1] 0.477
```

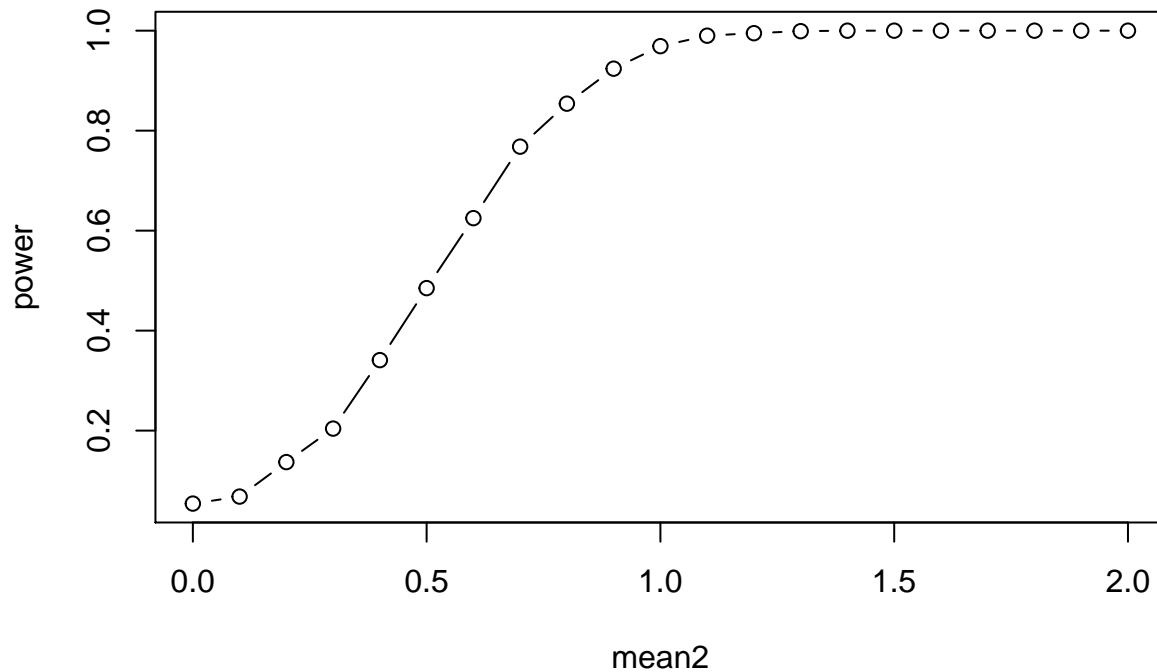
```
power_group_dif2(.8)
```

```
## [1] 0.852
```

```
power_group_dif2(1)
```

```
## [1] 0.968
```

```
settings <- seq(from = 0, to = 2, by = .1)  
results <- data.frame(mean2= settings)  
results$power <- sapply(results$mean2, function(X) power_group_dif2(X))  
plot(results, type = "b")
```



```
# obviously it could be made a whole lot more general
power_group_dif3 <- function(mean1 = 0, mean2 = 0.3, sd1 = 1, sd2 = 1,
                             n1 = 30 , n2 = 30, ksimulations = 1000,
                             alpha_criterion = .05) {

  significant <- NULL
  for (i in 1:ksimulations) {
    x <- rnorm(30, mean = mean1, sd = sd1)
    y <- rnorm(30, mean = mean2, sd = sd2)
    fit <- t.test(x, y)
    fit
    significant[i] <- (fit$p.value < alpha_criterion)
  }
  statistical_power <- mean(significant)
  statistical_power
}

power_group_dif3(mean1 = 10, mean2 = 11, sd1 = 1, sd2 = 1,
                  n1 = 100 , n2 = 100, ksimulations = 1000,
                  alpha_criterion = .01)
```

```
## [1] 0.889
```

Debugging functions

```
# debugging functions
print_some_text <- function(x = "Hello World") {
  print(x)
}
debugonce(print_some_text) # activates debugging on the function
print_some_text()
```

```
## debugging in: print_some_text()
## debug at <text>#2: {
##   print(x)
## }
## debug at <text>#3: print(x)
## [1] "Hello World"
```

```
# many other useful functions
?traceback # provide further information when an error occurs
?browser # place in function
```

Viewing source code for internal functions

```
# Option 1: type function name
t.test
cor
power.t.test

# Option 2:
# S3 Methods
# Some functions are generic and operate differently depending
# on the class of the first argument
# mean
# print
# summary

# Methods will list the actual function names called
methods(mean)
methods(print)
methods(summary)

mean.default
summary.table

# Option 3:
# Some functions are part of packages but are not exported
# I.e., they are intended for internal use, but
# they are often quite useful
library(ProjectTemplate)

# Double colon shows the functions exported from a package
# i.e., packagename::function
ProjectTemplate::run.project

# Triple colon shows internal functions
# i.e., packagename:::function
ProjectTemplate:::xls.reader
```

```
# Also, see the getAnywhere function  
xls.reader # this doesn't work
```

```
## Error in eval(expr, envir, enclos): object 'xls.reader' not found
```

```
getAnywhere(xls.reader) # this does work
```

Exercise 4

```
library(MASS)  
data(mammals)  
?mammals  
head(mammals)
```

```
##           body brain  
## Arctic fox    3.385  44.5  
## Owl monkey   0.480  15.5  
## Mountain beaver 1.350   8.1  
## Cow          465.000 423.0  
## Grey wolf    36.330 119.5  
## Goat         27.660 115.0
```

```
# 1. Create a function that takes a single argument x  
#     and prints that value twice.  
#     use the function to print "hello world" twice
```

```
# 2. Divide mammall brain weight (g) by body weight (kg) and  
#     get the mean of this value
```

```
# 3. Write a function that takes arguments x and y  
#     and returns the mean of x divided by y
```

```
# 4. Apply the function to get the mean ratio of brain to body size
```

```
# 5. Modify the ratio function to return a list with  
#     (a) the mean of x divided by y, and  
#     (b) the sd of x divided by y.  
#     Then apply to mammals data as above.
```

```
# 6. Step through the code for the correlation function
```

```
# 7. Show the source code for  
#     (a) the t.test function,
```

```
# (b) the summary method for lm objects
# (c) the alpha function in the psych package
```

Answers 4

```
library(MASS)
data(mammals)
?mammals
head(mammals)
```

```
##               body brain
## Arctic fox      3.385  44.5
## Owl monkey      0.480  15.5
## Mountain beaver 1.350   8.1
## Cow             465.000 423.0
## Grey wolf       36.330 119.5
## Goat           27.660 115.0
```

```
# 1. Create a function that takes a single argument x
#     and prints that value twice.
#     use the function to print "hello world" twice
print_twice <- function(x) {
  print(x)
  print(x)
}
print_twice("hello world")
```

```
## [1] "hello world"
## [1] "hello world"
```

```
# 2. Divide mammall brain weight (g) by body weight (kg) and
#     get the mean of this value
mean(mammals$brain / mammals$body )
```

```
## [1] 9.624214
```

```
# 3. Write a function that takes arguments x and y
#     and returns the mean of x divided by y
mean_ratio <- function(x, y) {
  mean(x / y)
}
```

```
# 4. Apply the function to get the mean ratio of brain to body size
mean_ratio(mammals$brain, mammals$body)
```

```
## [1] 9.624214
```

```

# 5. Modify the ratio function to return a list with
#   (a) the mean of x divided by y, and
#   (b) the sd of x divided by y.
#   Then apply to mammals data as above.
mean_ratio <- function(x, y) {
  ratioxy <- x / y
  list(mean_ratio = mean(ratioxy),
        sd_ratio = sd(ratioxy))
}

# 6. Step through the code for the correlation function
# debugonce(cor)
cor(mammals$brain, mammals$body, method = "spearman")

```

```
## [1] 0.9534986
```

```

# 7. Show the source code for
#   (a) the t.test function,
#   (b) the summary method for lm objects
#   (c) the alpha function in the psych package
# t.test
# summary.lm
# psych::alpha

```