DSA 8020 R Session 0: A Quick Introduction to R

Whitney Huang, Clemson University

Contents

This R session is modified from an R tutorial given by Dr. Doug Nychka at Colorado School of Mines [link].

The goal of this session is to help you become comfortable with basic R commands, including objects, arithmetic, vectors, plotting, subsetting, apply functions, and writing simple functions.

1. Basic R Objects and the Workspace

What is currently in the workspace?

The workspace contains the R objects that have been created during the current R session.

```
# List all objects currently stored in the workspace.
ls()
```

```
## character(0)
```

Where is my R working directory?

The working directory is the folder where R looks for files by default and where R saves output files unless another path is specified.

```
# Print the current working directory.
getwd()
```

```
## [1] "/Users/wkhuang/Desktop/Desktop - mass-mini19-huang/Teaching/STAT8020/Summer/R"
```

What files are in my working directory?

```
# List files and folders in the current working directory.
dir()
```

```
## [1] "BoulderTemperature.RData" "BT.RData"
## [3] "STAT8020_RCode0.Rmd"
```

Using c() to create a vector

The function c() combines values into a vector. A vector is one of the most common data structures in R.

```
# Create a numeric vector with three values.
x <- c(2, 3, 20)

# Print the vector.
x
```

```
## [1]  2  3 20
```

Note: `R` is case sensitive. The object `x` is different from `X`. Type `X` in the R console and press Enter to see what happens.

Recheck the workspace

```
# Check whether x has been added to the workspace.
ls()
```

```
## [1] "x"
```

```
# Print the value stored in x.
x
```

```
## [1]  2  3 20
```

Reassign `x` to another object name

```
# Create a new object x2 that stores the same values as x.
x2 <- x

# Check that both x and x2 are now in the workspace.
ls()
```

```
## [1] "x"  "x2"
```

Remove one object and create another

```
# Remove x from the workspace.
rm(x)

# Create a new vector x3.
x3 <- c(3, 4, 5)

# Check the objects currently in the workspace.
ls()
```

```
## [1] "x2" "x3"
```

Question: how would you combine `x2` and `x3` into a new vector?

```
# Try it here.
# Hint: use c().

# combined_x <- c(x2, x3)
# combined_x
```

2. Arithmetic in R

Adding numbers in R

R can perform arithmetic with individual numbers and with vectors.

```
# Add two single numbers.
A <- 2
B <- 10
Y <- A + B
Y
```

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

```
# Add two vectors element by element.
A <- c(2, 3, 4)
B <- c(10, 100, 1000)
Y <- A + B

# The first value of A is added to the first value of B, and so on.
Y
```

```
## [1]   12  103 1004
```

Other common arithmetic operations

```
# Exponentiation: 2 raised to the 4th power.
2^4
```

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

```
# Parentheses control the order of operations.
2 * (1 + 4)
```

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

```
# Square root.
sqrt(81)
```

```
## [1] 9
```

```
# Exponential function e^2.
exp(2)
```

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

Generating simple sequences

```r
# Generate integers from 1 to 10.
1:10
```

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

```r
# Generate integers from -5 to 5.
-5:5
```

```
##  [1] -5 -4 -3 -2 -1  0  1  2  3  4  5
```

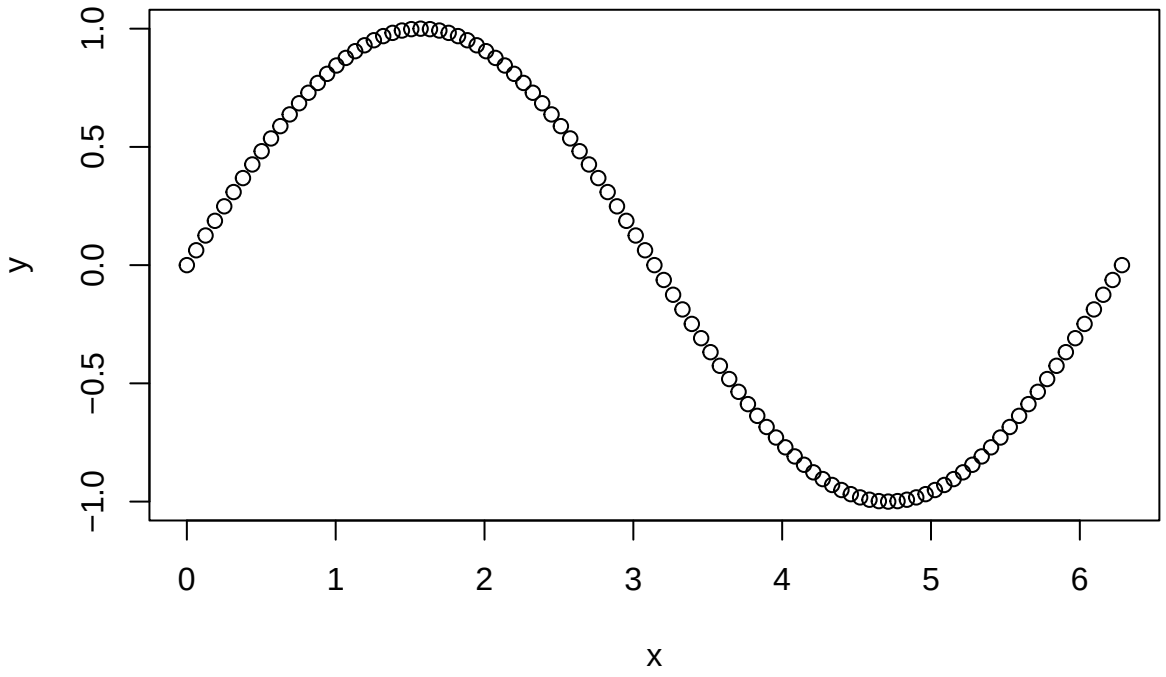
Question: How would you generate the values from 5 down to 1?

```r
# Try it here.
# Hint: the colon operator can count down as well as up.

# 5:1
```

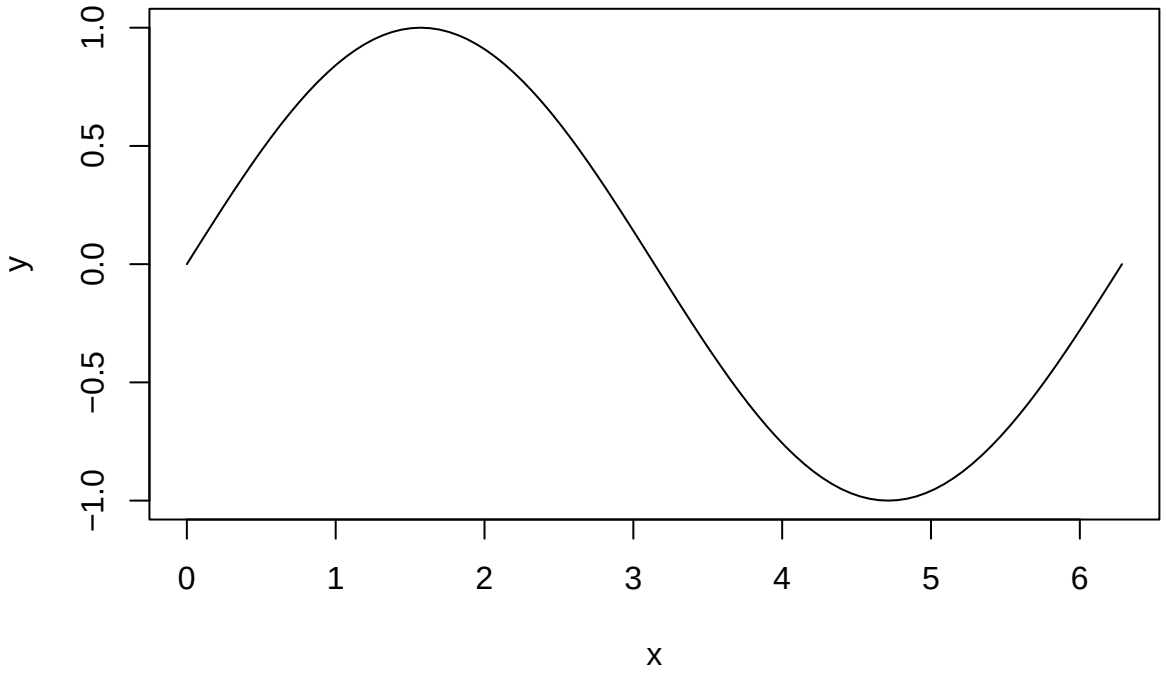
Generating and plotting a sine wave

This example creates values between 0 and 2π, evaluates the sine function at those values, and then plots the result.

```r
# Start with integers from 0 to 100.
x <- 0:100

# Convert these integers into 101 equally spaced values from 0 to 2*pi.
x <- 2 * pi * (x / 100)

# Compute sin(x) for each value of x.
y <- sin(x)

# Plot the points.
plot(x, y)
```




```r
# Plot the same data as a connected line.
plot(x, y, type = "l")
```




```r
# A more direct way to create 101 equally spaced values from 0 to 2*pi.
x <- seq(from = 0, to = 2 * pi, length.out = 101)
```

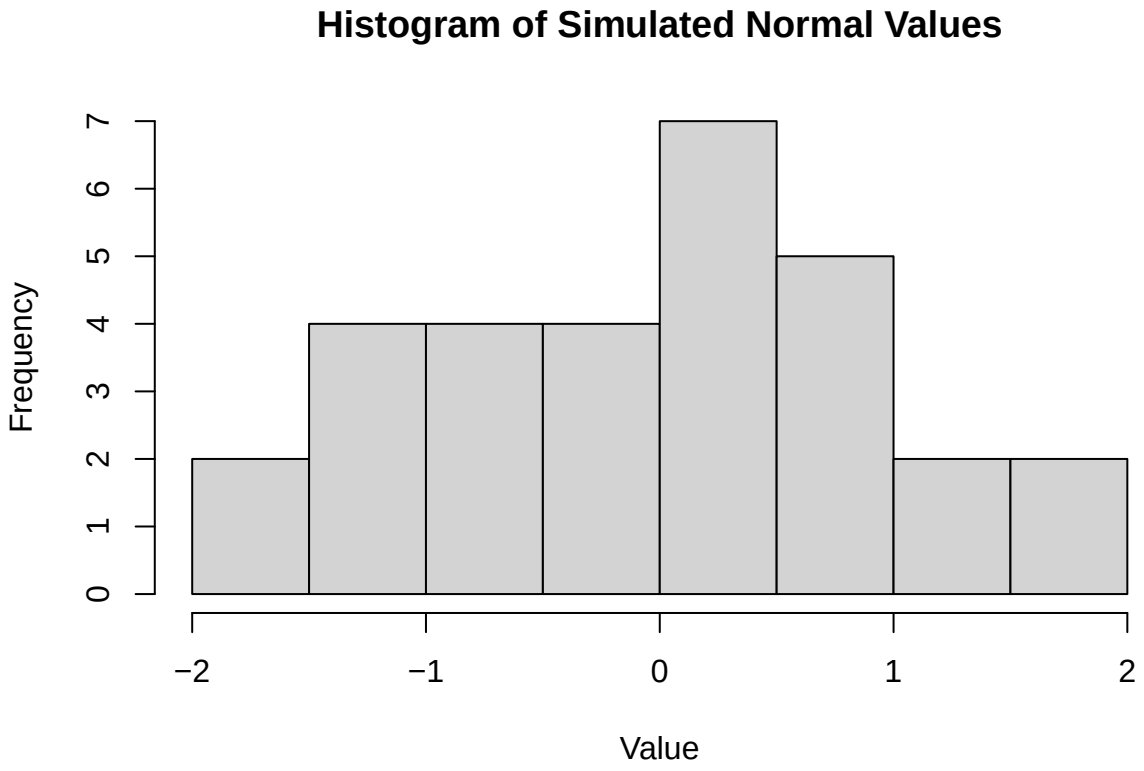
Generating random values

Random numbers are useful for simulation, resampling, and statistical modeling.

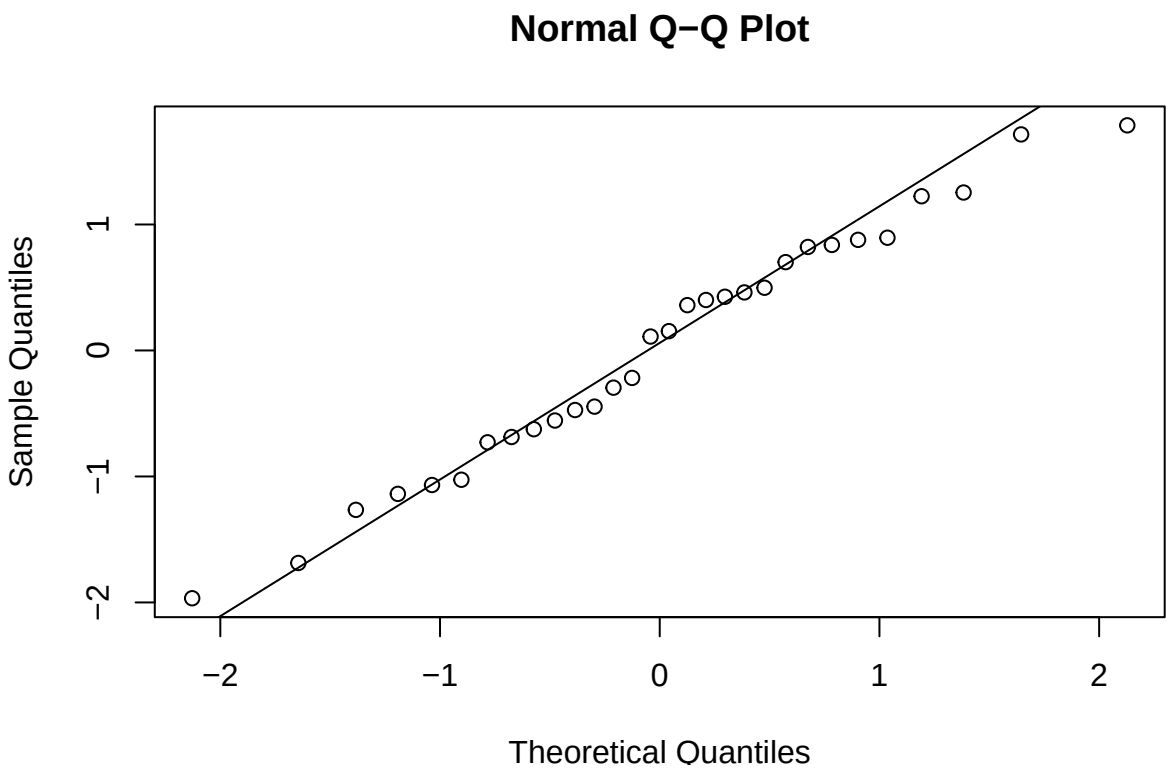
```r
# Set a seed so the random numbers are reproducible.
set.seed(123)

# Generate 10 random numbers from a Uniform(0, 1) distribution.
U <- runif(n = 10)
U
```

```
##  [1] 0.2875775 0.7883051 0.4089769 0.8830174 0.9404673 0.0455565 0.5281055
##  [8] 0.8924190 0.5514350 0.4566147
```

```r
# Generate 30 random numbers from a standard normal distribution.
V <- rnorm(n = 30, mean = 0, sd = 1)

# Create a histogram to visualize the distribution of V.
hist(V, main = "Histogram of Simulated Normal Values", xlab = "Value")
```




```r
# Create a normal Q-Q plot to check whether the data look approximately normal.
qqnorm(V)
qqline(V)
```



3. Subsetting Data

Load a data set

This section uses a Boulder temperature data set stored in an `.RData` file. Make sure the file `BT.RData` is in your working directory.

```
# Load the data set into R.
load("BT.RData")

# Extract the temperature column for easier typing.
BT <- BoulderJuneTemperature$Temp

# Keep the full data set as a separate object.
BAll <- BoulderJuneTemperature

# Preview the first few values of the temperature vector.
head(BT)
```

```
## [1] 65.51667 68.58333 69.21667 68.58333 70.91667 64.25000
```

```
# Preview the first 10 rows of the full data set.
head(BAll, 10)
```

```
##    Year     Temp
## 1  1984 65.51667
## 2  1985 68.58333
```

```
## 3  1986 69.21667
## 4  1987 68.58333
## 5  1988 70.91667
## 6  1989 64.25000
## 7  1990 69.95000
## 8  1991 66.56667
## 9  1992 62.90000
## 10 1993 64.66667
```

Print the first 10 temperature values

```
# Use square brackets to select elements 1 through 10.
BT[1:10]
```

```
##  [1] 65.51667 68.58333 69.21667 68.58333 70.91667 64.25000 69.95000 66.56667
##  [9] 62.90000 64.66667
```

Create an indicator for temperatures above 70

A logical indicator has values `TRUE` or `FALSE`. Here, `TRUE` means that the temperature is above 70.

```
# Identify which temperature values are greater than 70.
ind70 <- BT > 70
ind70
```

```
##  [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
## [13] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE
## [25] FALSE FALSE FALSE FALSE  TRUE FALSE
```

Use the indicator to subset the data

```
# Select temperatures above 70.
BT[ind70]
```

```
## [1] 70.91667 70.05000 70.36667 71.56667 74.13333
```

```
# Select the years corresponding to temperatures above 70.
BAll$Year[ind70]
```

```
## [1] 1988 1994 2002 2006 2012
```

Question: How many years had temperatures above 70 degrees?

```
# Try it here.
# Hint: TRUE is treated as 1 and FALSE is treated as 0 when using sum().

# sum(ind70)
```

Working with the data as a matrix or data frame

The object `BAll` has rows and columns. We can use `[row, column]` notation to select specific parts of the data.

```
# Check the number of rows and columns.
dim(BAll)
```

```
## [1] 30  2
```

```
# Select the first row and first column.
BAll[1, 1]
```

```
## [1] 1984
```

```
# Select the first row and all columns.
BAll[1, ]
```

```
##   Year     Temp
## 1 1984 65.51667
```

```
# Select all rows and the first column.
BAll[, 1]
```

```
##  [1] 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998
## [16] 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013
```

```
# Select the column named Year.
BAll[, "Year"]
```

```
##  [1] 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998
## [16] 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013
```

```
# Select the second column. This is the same as selecting the Temp column.
BAll[, 2]
```

```
##  [1] 65.51667 68.58333 69.21667 68.58333 70.91667 64.25000 69.95000 66.56667
##  [9] 62.90000 64.66667 70.05000 62.33333 66.93333 66.40000 62.68333 64.85000
## [17] 67.40000 68.85000 70.36667 62.83333 62.70000 65.41667 71.56667 67.65000
## [25] 66.01667 63.16667 66.35000 67.56667 74.13333 69.82000
```

```
# Select rows 10 through 20 and all columns.
BAll[10:20, ]
```

```
##    Year     Temp
## 10 1993 64.66667
## 11 1994 70.05000
## 12 1995 62.33333
## 13 1996 66.93333
## 14 1997 66.40000
```

```
## 15 1998 62.68333
## 16 1999 64.85000
## 17 2000 67.40000
## 18 2001 68.85000
## 19 2002 70.36667
## 20 2003 62.83333
```

Exercise: Plot the temperatures by year.

```
# Try it here.
# plot(BAll$Year, BAll$Temp, type = "l",
#      xlab = "Year", ylab = "June Temperature")
```

4. Apply Functions in R

What are apply functions?

The `apply` family of functions allows you to repeat the same calculation over rows, columns, or groups of data. These functions often require less code than explicit loops.

Load the Boulder monthly temperature data

Make sure the file `BoulderTemperature.RData` is in your working directory.

```
# Load monthly mean temperatures.
load("BoulderTemperature.RData")

# Check the dimensions of the data set.
dim(BoulderTemperature)
```

```
## [1] 118  12
```

```
# Check the first row.
BoulderTemperature[1, ]
```

```
##      jan feb mar apr      may   jun      jul      aug      sep      oct
## 1897 NaN NaN NaN NaN 60.25806 64.65 70.56452 69.06452 66.81667 52.41935
##           nov      dec
## 1897 41.86667 30.40323
```

```
# Extract the row names, which contain the years.
yr <- rownames(BoulderTemperature)

# Find rows corresponding to years 1991 through 2010.
index <- which(yr %in% 1991:2010)

# Keep only the selected years.
tempData <- BoulderTemperature[index, ]

# View the selected data.
tempData
```

```
##           jan      feb      mar      apr      may      jun      jul      aug
## 1991 29.61290 40.96429 42.69355 47.73333 58.22581 66.56667 70.46774 69.11290
## 1992 35.28571 40.56897 43.25806 54.21667 59.06452 62.90000 68.14516 66.29032
## 1993 28.33871 30.58929 42.30645 47.56667 57.91935 64.66667 69.46774 67.37097
## 1994 35.50000 32.10714 43.82258 47.61667 60.80645 70.05000 71.14516 71.01613
## 1995 34.67742 38.28571 42.11290 44.51667 50.85484 62.33333 70.48387 73.96774
## 1996 29.70968 37.67241 37.62903 50.41667 58.87097 66.93333 71.45161 69.48387
## 1997 31.37097 32.96429 45.53226 42.81667 57.17742 66.40000 71.40323 68.85484
## 1998 36.50000 36.39286 38.54839 46.50000 58.61290 62.68333 72.75806 70.37097
## 1999 36.20968 42.10714 45.98387 44.55000 55.58065 64.85000 73.33871 69.30645
## 2000 36.41935 41.06897 42.85484 51.23333 60.98387 67.40000 74.66129 73.03226
## 2001 32.93548 32.32143 40.75806 50.63333 58.40323 68.85000 75.14516 71.85484
## 2002 33.11290 35.98214 37.27419 53.01667 56.16129 70.36667 76.90323 71.30645
## 2003 40.20968 32.08929 43.66129 50.60000 57.35484 62.83333 75.66129 72.79032
## 2004 35.40323 33.67241 48.17742 49.18333 59.96774 62.70000 69.16129 66.40323
## 2005 35.43548 37.87500 41.96774 48.40000 57.72581 65.41667 75.04839 69.70968
## 2006 40.66129 33.67857 39.38710 53.88333 60.95161 71.56667 74.37097 71.56452
## 2007 27.22581 34.58929 47.56452 47.81667 58.00000 67.65000 74.75806 73.56452
## 2008 31.62903 36.10345 40.75806 47.80000 57.03226 66.01667 75.01613 69.62903
## 2009 38.19355 39.33929 44.20968 47.30000 59.30645 63.16667 69.53226 69.48387
## 2010 33.01613 30.05357 42.37097 48.75000 53.90323 66.35000 72.45161 72.41935
##           sep      oct      nov      dec
## 1991 61.56667 52.14516 37.05000 35.48387
## 1992 64.41667 53.87097 34.00000 29.77419
## 1993 59.01667 48.61290 35.61667 35.41935
## 1994 64.83333 50.69355 36.53333 36.08065
## 1995 60.38333 51.33871 44.95000 36.24194
## 1996 60.76667 53.01613 40.58333 36.46774
## 1997 64.01667 52.66129 37.86667 33.83871
## 1998 67.20000 50.32258 44.01667 32.16129
## 1999 58.48333 51.90323 47.98333 36.91935
## 2000 63.10000 49.59677 31.31667 31.20968
## 2001 65.00000 53.83871 43.85000 34.98387
## 2002 64.06667 45.75806 40.26667 36.58065
## 2003 60.50000 57.38710 38.91667 36.35484
## 2004 62.85000 51.85484 39.66667 36.45161
## 2005 66.35000 53.09677 44.93333 33.30645
## 2006 58.40000 50.98387 43.36667 35.29032
## 2007 64.43333 55.17742 44.86667 30.06452
## 2008 60.90000 51.80645 46.20000 31.09677
## 2009 63.10000 44.46774 43.76667 26.66129
## 2010 66.55000 54.77419 39.76667 37.19355
```

Using `apply()` to compute means

In `apply(X, MARGIN, FUN)`, the argument `MARGIN = 1` means apply the function by row, and `MARGIN = 2` means apply the function by column.

```r
# Compute the mean temperature for each year.
byYear <- apply(tempData, MARGIN = 1, FUN = mean)
byYear
```

```
##     1991     1992     1993     1994     1995     1996     1997     1998
```

```
##     1991     1992     1993     1994     1995     1996     1997     1998
## 50.96857 50.98260 48.90762 51.68375 50.84554 51.08345 50.40858 51.33892
##     1999     2000     2001     2002     2003     2004     2005     2006
## 52.26798 51.90642 52.38118 51.73297 52.36322 51.29098 52.43878 52.84208
##     2007     2008     2009     2010
## 52.14257 51.16565 50.71062 51.46661
```

```
# rowMeans() gives the same result for row averages.
rowMeans(tempData)
```

```
##     1991     1992     1993     1994     1995     1996     1997     1998
## 50.96857 50.98260 48.90762 51.68375 50.84554 51.08345 50.40858 51.33892
##     1999     2000     2001     2002     2003     2004     2005     2006
## 52.26798 51.90642 52.38118 51.73297 52.36322 51.29098 52.43878 52.84208
##     2007     2008     2009     2010
## 52.14257 51.16565 50.71062 51.46661
```

```
# Compute the mean temperature for each month across years.
byMonth <- apply(tempData, MARGIN = 2, FUN = mean)
byMonth
```

```
##      jan      feb      mar      apr      may      jun      jul      aug
## 34.07235 35.92127 42.54355 48.72750 57.84516 65.98500 72.56855 70.37661
##      sep      oct      nov      dec
## 62.79667 51.66532 40.77583 34.07903
```

```
# colMeans() gives the same result for column averages.
colMeans(tempData)
```

```
##      jan      feb      mar      apr      may      jun      jul      aug
## 34.07235 35.92127 42.54355 48.72750 57.84516 65.98500 72.56855 70.37661
##      sep      oct      nov      dec
## 62.79667 51.66532 40.77583 34.07903
```

5. Writing Functions in R

Finding the interquartile range (IQR)

The interquartile range is the difference between the 75th percentile and the 25th percentile.

```
# Compute the 75th percentile of BT.
BT75 <- quantile(BT, probs = 0.75)
BT75
```

```
##    75%
## 69.125
```

```
# Compute the 25th percentile of BT.
BT25 <- quantile(BT, probs = 0.25)
BT25
```

```
##     25%
## 64.7125
```

```
# Compute the interquartile range manually.
BT75 - BT25
```

```
##    75%
## 4.4125
```

```
# Compare with the built-in IQR() function.
IQR(BT)
```

```
## [1] 4.4125
```

Building your own function

The function below adds the squares of two inputs. A function usually has three main parts: input arguments, a body where the calculation is performed, and a returned result.

```
# Define a function that computes a^2 + b^2.
myFun <- function(a, b) {
  result <- a^2 + b^2
  return(result)
}

# Test the function with two single numbers.
test1 <- myFun(a = 2, b = 3)
test1
```

```
## [1] 13
```

```
# Test the function with two vectors.
# R performs the calculation element by element.
test2 <- myFun(a = 1:5, b = 11:15)
test2
```

```
## [1] 122 148 178 212 250
```

The objects `a`, `b`, and `result` are local to the function. This means they are used inside the function but do not appear as separate objects in the global workspace.

Building your own IQR function

```
# Define a function to compute the interquartile range.
myIQR <- function(y) {
  q75 <- quantile(y, probs = 0.75, names = FALSE)
  q25 <- quantile(y, probs = 0.25, names = FALSE)
  iqr_value <- q75 - q25
  return(iqr_value)
```

```
}

# Apply the function to the Boulder June temperature vector.
myIQR(BT)
```

```
## [1] 4.4125
```

Modify the function to work with missing values

Missing values are represented by `NA` in R. Many functions require the argument `na.rm = TRUE` to remove missing values before calculation.

```
# Add one missing value to the temperature vector.
test <- c(BT, NA)

# Define a version of myIQR that can optionally remove missing values.
myIQR <- function(y, na.rm = FALSE) {
  q75 <- quantile(y, probs = 0.75, names = FALSE, na.rm = na.rm)
  q25 <- quantile(y, probs = 0.25, names = FALSE, na.rm = na.rm)
  iqr_value <- q75 - q25
  return(iqr_value)
}

# Remove missing values before computing the IQR.
myIQR(test, na.rm = TRUE)
```

```
## [1] 4.4125
```

Adding a warning message

Warnings are useful when a function still runs but the user should be aware of something important.

```
# Add a warning message when missing values are removed.
myIQR <- function(y, na.rm = FALSE) {
  if (na.rm) {
    warning("Missing values were removed before computing the IQR.")
  }

  q75 <- quantile(y, probs = 0.75, names = FALSE, na.rm = na.rm)
  q25 <- quantile(y, probs = 0.25, names = FALSE, na.rm = na.rm)
  iqr_value <- q75 - q25
  return(iqr_value)
}

# Compute the IQR after removing missing values.
myIQR(test, na.rm = TRUE)
```

```
## Warning in myIQR(test, na.rm = TRUE): Missing values were removed before
## computing the IQR.
```

```
## [1] 4.4125
```

6. Summary of Key Ideas

By the end of this session, you should be able to:

- create and inspect objects in `R`;
- understand the working directory and workspace;
- perform basic arithmetic with numbers and vectors;
- generate sequences and random values;
- make simple plots;
- subset vectors and data frames;
- use `apply()` to summarize rows and columns;
- write simple `R` functions; and
- handle missing values using `na.rm = TRUE`.