ISLR Chapter 2 Lab 2.3: Introduction to R

Roger H. French, JiQi Liu

02 September, 2018

Contents

3.1.1.1 Five sections to this

- ISLR 2.3.1 Basic Commands (included here)
- ISLR 2.3.2 Base Graphics (included here)
- ISLR 2.3.3 Indexing Data (in the .R file)
- ISLR 2.3.4 Loading Data (in the .R file)
- ISLR 2.3.5 Additional Graphics and Numerical Summaries (in the .R file)

3.1.1.1.1 License: CC-BY-SA 4.0

3.1.1.2 The datasets and code for Open Intro Stats are in

- The OIdata package
- The openintro package

```
library("openintro")
```

```
## Please visit openintro.org for free statistics materials
```

```
##
## Attaching package: 'openintro'
```

```
## The following objects are masked from 'package:datasets':
##
##     cars, trees
```

```
help("openintro")
```

```
## starting httpd help server ...
##  done
```

```
library("OIdata")
```

```
## Loading required package: RCurl
## Loading required package: bitops
## Loading required package: maps
```

```
help("OIdata")
```

The datasets for ISLR are in the ISLR packagelibray(ISLR)

```
library(ISLR)
help("ISLR")
```

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

```
??ISLR
```

3.1.1.3 ISLR 2.3.1 Basic Commands

To run a function called funcname,we type funcname(input1, input2), - where the inputs (or arguments) input1 and input2 tell R how to run the function.

3.1.1.3.1 c()

- For example, to create a vector of numbers, we use the function c() (for concatenate).
  - Any numbers inside the parentheses are joined together.
  - The following command instructs R to join together the numbers 1, 3, 2, and 5,
  - and to save them as a vector named x.
  - When we type x, it gives us back the vector.

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

```
## [1] 1 3 2 5
```

```
?c # will call up help for the funcname
```

Note that the > is not part of the command;

- rather, it is printed by R to indicate that it is ready for another command to be entered.

Don't use =, use <-, subtle hazards arise if you use =

```
x <- c(1,6,2)
x
```

```
## [1] 1 6 2
```

```
y <- c(1,4,3)
```

Hitting the up arrow multiple times will display the previous commands, which can then be edited.

- This is useful since one often wishes to repeat a similar command.

In addition, typing ?funcname will always cause R to open a new help file window

- with additional information about the function

3.1.1.3.2 length()

- We can tell R to add two sets of numbers together.
  - It will then add the first number from x to the first number from y, and so on.
  - However, x and y should be the same length.
  - We can check their length using the length() function.

```
length(x)
```

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

```
length(y)
```

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

```
x + y
```

```
## [1]  2 10  5
```

3.1.1.3.3 ls() and rm()

- The ls() function allows us to look at a list of all of the objects,
  - such as data and functions, that we have saved so far.
- The rm() function can be used to delete any that we don't want.

```
ls()
```

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

```
rm(x,y)
ls()
```

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

It's also possible to remove all objects at once:

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

3.1.1.3.4 matrix()

- The matrix() function can be used to create a matrix of numbers.
  - Before we use the matrix() function, we can learn more about it:

```
?matrix
```

The help file reveals that the matrix() function takes a number of inputs,

- but for now we focus on the first three:
  - the data (the entries in the matrix),
  - the number of rows, and
  - the number of columns.

First, we create a simple matrix.

```
x <- matrix(data = c(1,2,3,4), nrow = 2, ncol = 2)
x
```

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

Note that we could just as well omit typing data=, nrow=, and ncol= in the matrix() command above:

- that is, we could just type

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

and this would have the same effect.

However, it can sometimes be useful to specify the names of the arguments passed in,

- since otherwise R will assume that the function arguments are passed into the function
- in the same order that is given in the function's help file.

As this example illustrates, by default R creates matrices by successively filling in columns.

- Alternatively, the byrow=TRUE option can be used to populate the matrix in order of the rows.

```
matrix(c(1,2,3,4),2,2,byrow = TRUE)
```

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

Notice that in the above command we did not assign the matrix to a value such as x.

- In this case the matrix is printed to the screen
- but is not saved for future calculations.

3.1.1.3.5 sqrt()

- The sqrt() function returns the square root of each element of a vector or matrix

```
sqrt(x)
```

```
##          [,1]     [,2]
## [1,] 1.000000 1.732051
## [2,] 1.414214 2.000000
```

x^2

The command x^2 raises each element of x to the power 2;

- any powers are possible, including fractional or negative powers.

```
x^2
```

```
##      [,1] [,2]
## [1,]    1    9
## [2,]    4   16
```

3.1.1.3.6 rnorm()

- The rnorm() function generates a vector of random normal variables,
  - with first argument n the sample size.
- Each time we call this function, we will get a different answer.

3.1.1.3.7 cor()

- Here we create two correlated sets of numbers, x and y, and
- use the cor() function
  - to compute the correlation between them.

```
x <- rnorm(50)
y <- x + rnorm(50,mean = 50,sd = .1)
cor(x,y)
```

```
## [1] 0.996379
```

By default, rnorm() creates standard normal random variables with a mean of 0 and a standard deviation of 1.

- However, the mean and standard deviation can be altered using the mean and sd arguments, as illustrated above.

Sometimes we want our code to reproduce the exact same set of random numbers;

- we can use the set.seed() function to do this.
- The set.seed() function takes an (arbitrary) integer argument.

```
set.seed(1303)
rnorm(50)
```

```
##  [1] -1.1439763145  1.3421293656  2.1853904757  0.5363925179  0.0631929665
##  [6]  0.5022344825 -0.0004167247  0.5658198405 -0.5725226890 -1.1102250073
## [11] -0.0486871234 -0.6956562176  0.8289174803  0.2066528551 -0.2356745091
## [16] -0.5563104914 -0.3647543571  0.8623550343 -0.6307715354  0.3136021252
## [21] -0.9314953177  0.8238676185  0.5233707021  0.7069214120  0.4202043256
## [26] -0.2690521547 -1.5103172999 -0.6902124766 -0.1434719524 -1.0135274099
## [31]  1.5732737361  0.0127465055  0.8726470499  0.4220661905 -0.0188157917
## [36]  2.6157489689 -0.6931401748 -0.2663217810 -0.7206364412  1.3677342065
## [41]  0.2640073322  0.6321868074 -1.3306509858  0.0268888182  1.0406363208
## [46]  1.3120237985 -0.0300020767 -0.2500257125  0.0234144857  1.6598706557
```

We use set.seed() throughout the labs whenever we perform calculations involving random quantities.

- In general this should allow the user to reproduce our results.
- However, it should be noted that as new versions of R become available
  - it is possible that some small discrepancies may form between the book and the output from R.

3.1.1.3.8 mean(), var(), sd()

- The mean() and var() functions can be used to
  - compute the mean and variance of a vector of numbers.
- Applying sqrt() to the output of var()will give the standard deviation.
  - Or we can simply use the sd() function.

```
set.seed(3)
y <- rnorm(100)
mean(y)
```

```
## [1] 0.01103557
```

```
var(y)
```

```
## [1] 0.7328675
```

```
sqrt(var(y))
```

```
## [1] 0.8560768
```

```
sd(y)
```

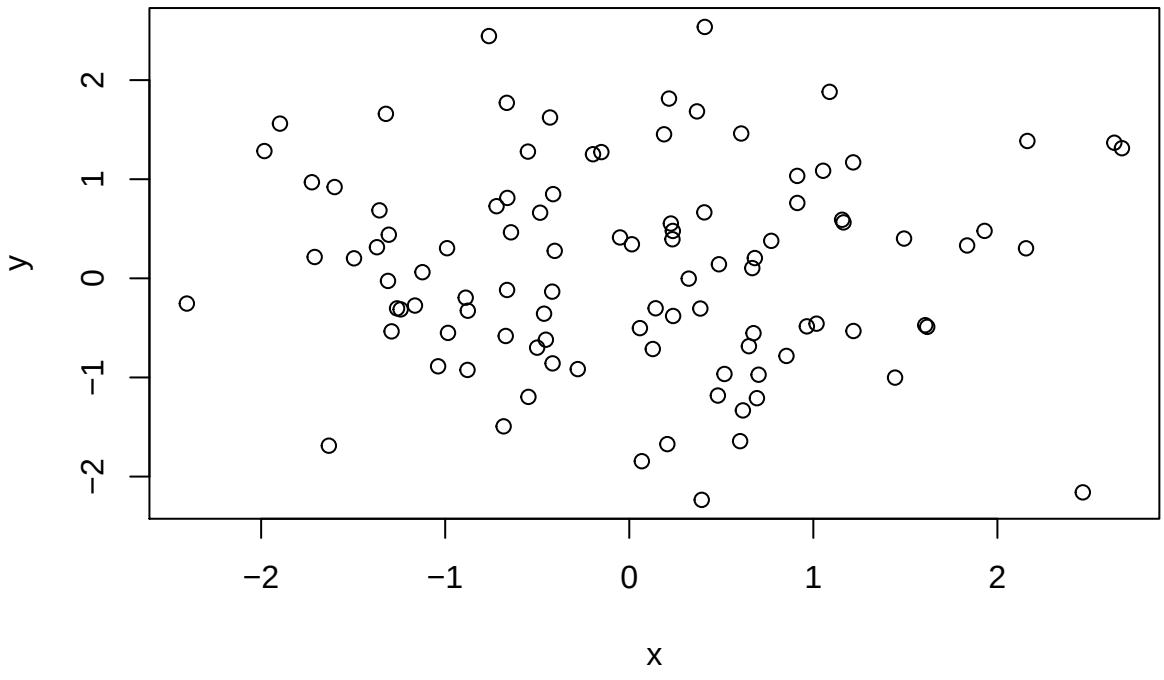
```
## [1] 0.8560768
```

3.1.1.4 ISLR 2.3.2 Base Graphics

3.1.1.4.1 plot()

- The plot() function is the primary way to plot data in R.
  - plot() is part of base R, so its base graphics.
  - For instance, plot(x,y) produces a scatterplot of the numbers in x versus the numbers in y.
- There are many additional options that can be passed in to the plot() function.
  - For example, passing in the argument xlab will result in a label on the x-axis.
- To find out more information about the plot() function, type ?plot.

```
x <- rnorm(100)
y <- rnorm(100)
plot(x,y)
```




```
plot(x,y,xlab = "this is the x-axis",ylab = "this is the y-axis",main = "Plot of X vs Y")
```

pdf(), jpeg()

- We will often want to save the output of an R plot.
  - The command that we use to do this will depend on the file type that we would like to create.
- For instance, to create a pdf, we use the pdf() function,
  - and to create a jpeg, we use the jpeg() function.

```
pdf("Figure.pdf")
plot(x,y,col = "green")
dev.off()
```

```
## pdf
##   2
```

The function dev.off() indicates to R that we are done creating the plot.

- Alternatively, we can simply copy the plot window
  - and paste it into an appropriate file type, such as a Word document.
- But this is not a useful practice to use
  - Better to caputure your figure as an individual file in your repo.

3.1.1.4.2 seq()

- The function seq() can be used to create a sequence of numbers.
  - For instance, seq(a,b) makes a vector of integers between a and b.
- There are many other options: - for instance, seq(0,1,length=10) makes a sequence of10 numbers that are equally spaced between 0 and 1. - Typing 3:11 is a shorthand for seq(3,11) for integer arguments.

```
x <- seq(1,10)
x
```

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

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

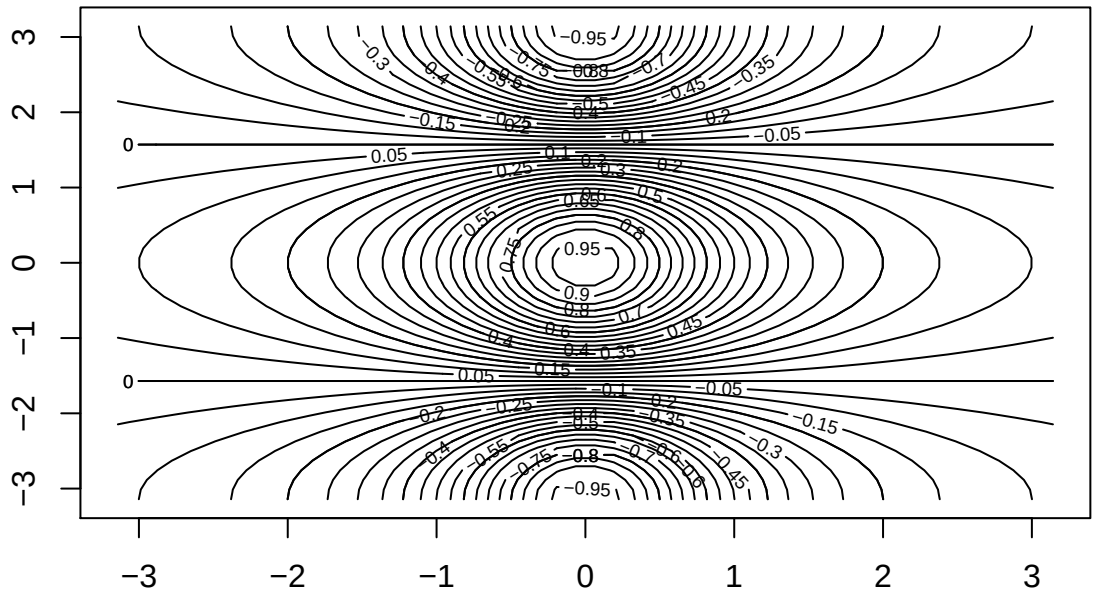
```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
x <- seq(-pi,pi,length = 50)
```

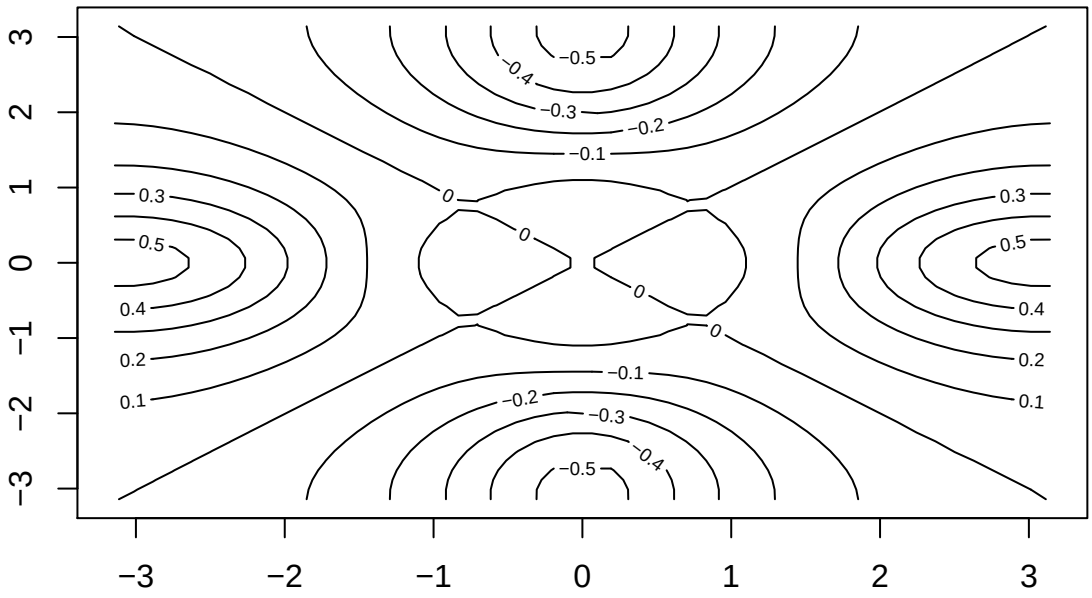
3.1.1.4.3 contour()

- We will now create some more sophisticated plots.
- The contour() function produces a contour plot
  - in order to represent three-dimensional data;
  - it is like a topographical map.
- It takes three arguments:
  - 1. A vector of the x values (the first dimension),
  - 2. A vector of the y values (the second dimension), and
  - 3. A matrix whose elements correspond to the z value (the third dimension) for each pair of (x,y) coordinates.
- As with the plot() function,
  - there are many other inputs that can be used to fine-tune the output of the contour() function.
  - To learn more about these, take a look at the help file by typing ?contour.

```
y <- x
f <- outer(x,y,function(x,y)cos(y)/(1 + x^2))
contour(x,y,f)
contour(x,y,f,nlevels = 45,add = T)
```



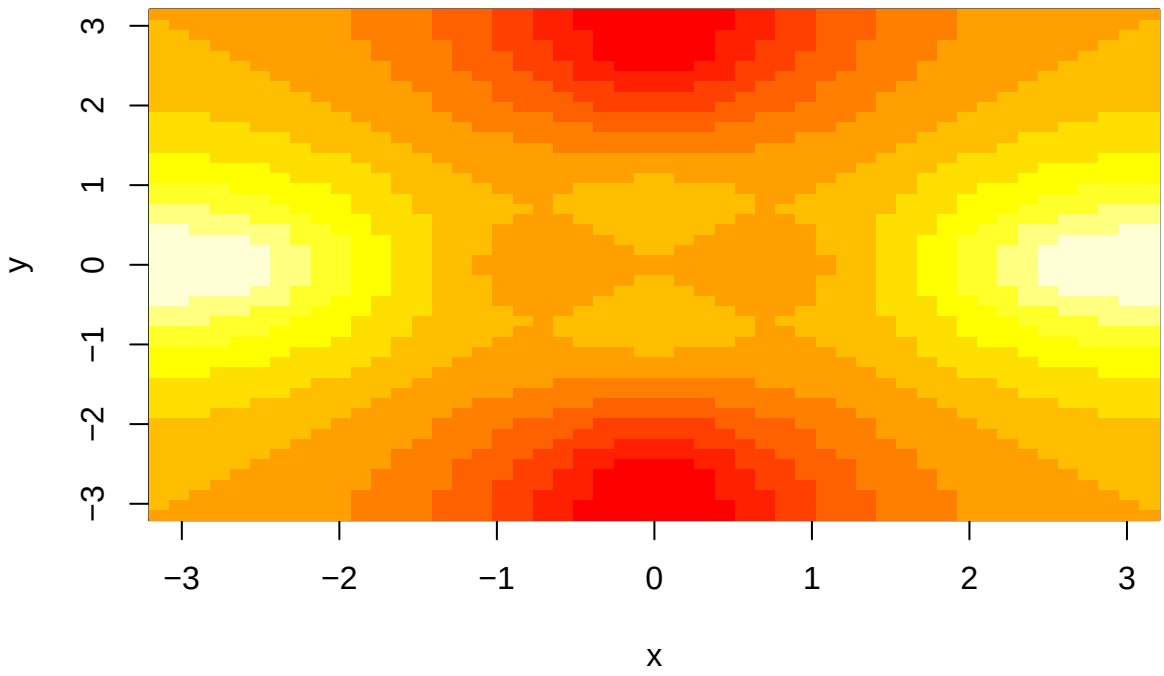
```
fa <- (f - t(f))/2
contour(x,y,fa,nlevels = 15)
```



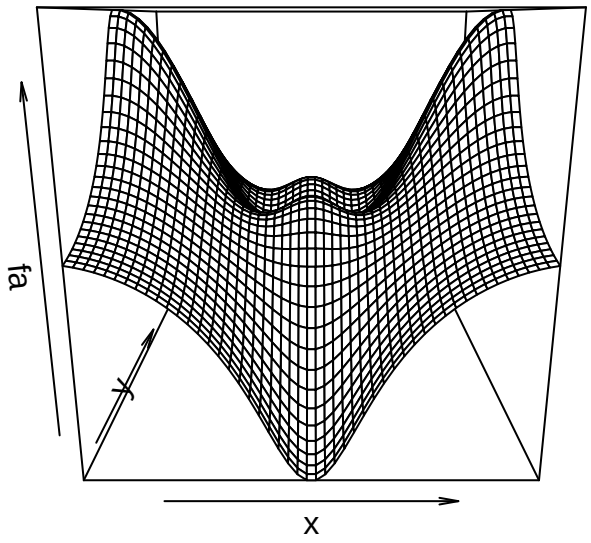
3.1.1.4.4 image(), heatmap(), persp()

- The image() function works the same way as contour(),
  - except that it produces a color-coded plot whose colors depend on the z value.
- This is known as a heatmap,
  - and is sometimes used to plot temperature in weatherforecasts.
- Alternatively, persp() can be used to produce a three-dimensional plot.
  - The arguments theta and phi control
    - the angles at which the plot is viewed.

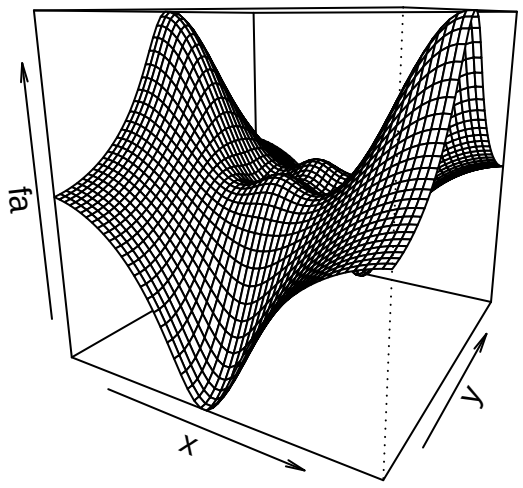
```
image(x,y,fa)
```



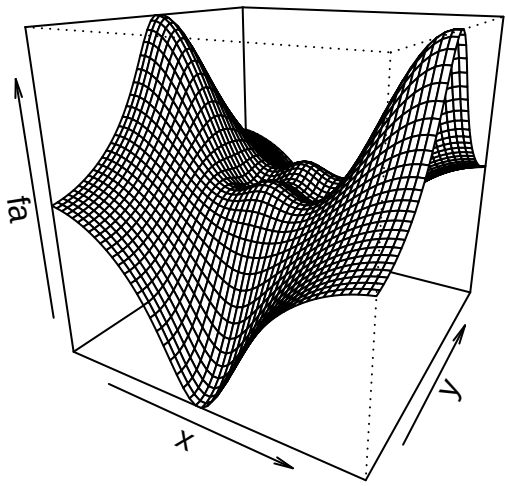

```
persp(x,y,fa)
```



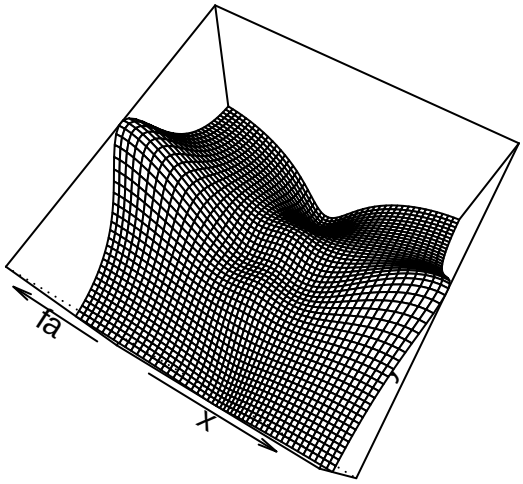

```
persp(x,y,fa,theta = 30)
```



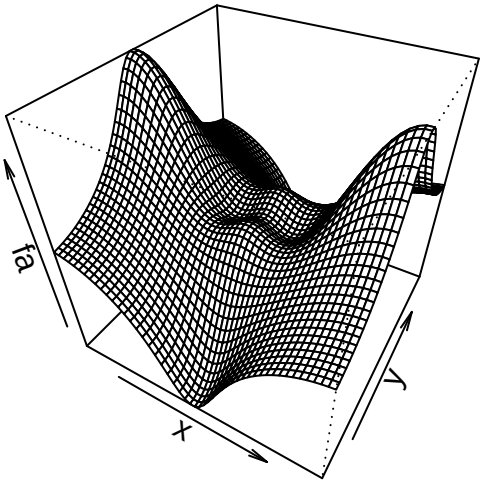

```
persp(x,y,fa,theta = 30,phi = 20)
```




```
persp(x,y,fa,theta = 30,phi = 70)
```




```
persp(x,y,fa,theta = 30,phi = 40)
```



3.1.1.5 Three ISLR sections remaining

- ISLR 2.3.1 Basic Commands (included here)
- ISLR 2.3.2 Base Graphics (included here)
- ISLR 2.3.3 Indexing Data (in the .R file)
- ISLR 2.3.4 Loading Data (in the .R file)
- ISLR 2.3.5 Additional Graphics and Numerical Summaries (in the .R file)