An introduction to R for McGill Epidemiology and Public Health students

Fei Wang & Jay Brophy

2018-01-05

Contents

Chapter 1

Introduction

This a book written to help introduce the statistical language `R` for use in an "epidemiology" context. It is written using **Markdown** and `R` packages `rmarkdown`, `knitr` and `bookdown`.

Epidemiology has been defined (WHO) as "the study of the distribution and determinants of health-related states or events (including disease), and the application of this study to the control of diseases and other health problems".

Public health has been defined (CDC) as "the fulfillment of society's interest in assuring the conditions in which people can be healthy".

Epidemiology is the basic quantitative science underlining public health. Data analysis is the processing of information collected by observation or experimentation and is an essential element of all epidemiologic investigations (pre and post processing of data are as important as the actual analysis). Informed decision making in epidemiology and public health are therefore crucially dependent on high quality, reproducible data analysis.

This book hopes to help the reader move forward along the path of reproducible data analysis.

Chapter 2

An overview of R: part I {ch2}

2.1 Basic computations in R

R console can be used as an interactive calculator.

```
2+2
```

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

```
2*2
```

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

```
log(4)
```

```
## [1] 1.39
```

```
2*3+2
```

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

```
sqrt(4)#square root of 4
```

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

```
round(4/3,1)
```

```
## [1] 1.3
```

```
exp(1)
```

```
## [1] 2.72
```

Note: In a command line, the contents behind "#" will be ignored. This is good for comments for coding.

2.2 Create an object

The variables can be assigned values using leftward, rightward and equal to operator.

```
# Assignment using equal operator.
var1 = c(0,1,2,3)
var1
```

```
## [1] 0 1 2 3
```

```
# Assignment using leftward operator.
var2 <- 3:10
var2
```

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

```
# Assignment using rightward operator.
c(TRUE,1) -> var3
var3
```

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

Variables can be alphabets, alphanumeric but not numeric. It is not allowed to create numeric variables. There are no restrictions to the length of the variable name.

Note: Variable names are case sensitive.

```
a<-1.5*5+3
a
```

```
## [1] 10.5
```

```
A<-2.3+5
A
```

```
## [1] 7.3
```

If the object already exists, its previous value is erased.

```
x<-2+3
x
```

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

```
x<-2*9
x
```

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

NOTE: do NOT assgin the single letter names c, g, t, C, D, F, I and T as they are default names that are used by R. For instance, T and F are abbrevations for TRUE and FALSE in logical operations. We should avoid using names that are already used by the system.

To know all the variables currently available in the workspace:

```
ls()  # list current objects
```

```
## [1] "a"    "A"    "var1" "var2" "var3" "x"
```

```
rm(x) # delete an object
ls()
```

```
## [1] "a"    "A"    "var1" "var2" "var3"
```

2.3 Operators

2.3.1 2.3.1 Arithmetic operators

```
+: addition
-:subtraction
```

```
*:multiplication
/:division
^ or **:exponentiation
x %% y: modulus (x mod y) 5%%2 is 1
x %/%y: integer division 5%/%2 is 2
```

Examples:

```r
a <- c(2,4,6)
b <- c(8,2,4)
print(a+b)
```

```
## [1] 10  6 10
```

```r
print(a-b)
```

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

```r
print(a*b)
```

```
## [1] 16  8 24
```

```r
print(a/b)
```

```
## [1] 0.25 2.00 1.50
```

```r
print(a^b)
```

```
## [1]  256   16 1296
```

```r
print(a%%b)
```

```
## [1] 2 0 2
```

```r
print(a%/%b)
```

```
## [1] 0 2 1
```

2.3.2 Relational operators

```
>: greater than
>=: greater and equal to
<: less than
<=: less than and equal to
==: exactly equal to
!=: not equal to
```

Examples:

```r
#if a and b of the same length
a <- c(2,4,6,8)
b <- c(8,2,4,10)
a>b
```

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

```r
a>=b
```

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

```r
a<b
```

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

```
a<=b
```

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

```
a==b
```

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

```
a!=b
```

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

```
#if a and b of different length
a<-c(2,4,6,8)
b<-c(3,5)
a==b#recycling the values of the shortest one
```

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

Note: Two equal signs are used to assess euqality of two objects. If use only one euqal sign, the equal sign does the same as the assignment operatior "<-" so that the value of the object on the left may be replaced with the content of the object on the right.

From the above codes,the comparison operators operate on each element of the two objects being compared, and thus returns an object of the same size.

To compare 'wholly' two objects, two functions are available: identical and all.equal

```
a<-1:4
b<-1:4
a==b
```

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

```
identical(a,b)
```

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

```
#identical compares the internal representation of the data and returns TRUE if the objects are strictl
all.equal(a,b)
```

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

```
#all.equal compares the "near equality" of two objects, and returns TRUE or display a summary of the di
0.8==(1.0-0.2)
```

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

```
identical(0.8,1.0-0.2)
```

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

```
all.equal(0.8,1.0-0.2)
```

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

```
all.equal(0.8,1.0-0.2,tolerance=1e-30)
```

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

```
#The comparison of numeric values on a computer is sometimes surprising!
1.1==(1.2-0.1)
```

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

```r
identical(1.1,1.2-0.1)
```

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

```r
all.equal(1.1,1.2-0.1)
```

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

```r
all.equal(1.1,1.2-0.1,tolerance=1e-16)
```

```
## [1] "Mean relative difference: 2.02e-16"
```

2.3.3 Logical operators

```
!x: not x
x|y: x or y
x & y: x and y
```

Examples:

```r
a <- c(2,4,6)
b <- c(8,2,4)
!a
```

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

```r
a>4 & b>4
```

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

```r
a>4 | b>4
```

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

```r
a>4 && b>4#consider only the first element of the vectors
```

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

```r
a>4 ||b>4#consider only the first element of the vectors
```

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

2.4 Generate data

2.4.1 Regular sequences

A regular sequence of consecutive integers can be generated with:

```r
x<-1:10
print(x)
```

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

The resulting vector x has 10 elements. The operator ':' has priority on the arithmetic operators within an expression:

```r
1:10-1
```

```
##  [1] 0 1 2 3 4 5 6 7 8 9
```

```
1:(10-1)
```

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

The function seq is more flexible to generate sequences of real numbers.

```
seq(1, 10, by=2)#The first number indicates the beginning of the sequence, the second one the end, and
```

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

```
#another way to generate it
seq(length=5, from=1, to=9)
```

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

Another important function is rep() which allows the generation of repeating sequences:

```
rep(1, 30)
```

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

```
rep(seq(1,7,2),3)
```

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

```
rep(seq(1,7,2),each=3)
```

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

The function sequence creates a series of sequences of integers each ending by the numbers given as arguments:

```
sequence(2:4)
```

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

```
sequence(c(2,5))
```

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

2.4.2 Random sequences

If we want to do random sampling, we can use the sample function:

```
set.seed(2017)#In order to be able to replicate random sampling results, we should set a seed
sample(1:20,10,replace=T)##sample 10 numbers out of 1 to 20 with replacement
```

```
##  [1] 19 11 10  6 16 16  1  9 10  6
```

```
sample(1:20,10,replace=F)##sample 10 numbers out of 1 to 20 without replacement
```

```
##  [1] 14  1 19  8 17  6 15 10 12 13
```

2.5 Data types

R has a wide variety of data types including vectors, lists, factors, matrices and data frames.

2.5.1 Vectors

Vectors are the most basic R data objects. A vector usually contains object of same class.

```
x <- c(1,2,5.3,6,-2,4) # numeric vector
y <- c("one","two","three") # character vector
z <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE) #logical vector
```

To check the class of an object:

```
class(x)# class or type of an object
```

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

To convert the class of a vector

```
class(x)
```

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

```
as.character(x)
```

```
## [1] "1"   "2"   "5.3" "6"   "-2"  "4"
```

```
class(x)
```

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

Note: If trying to convert a "character" vector to "numeric" , NAs will be introduced. Hence, it should be with caution to use this command.

```
y <- c("one","two","three")
as.numeric(y)
```

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

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

To assign names to a vector: The names of a vector are stored in a vector of the same length of the object, and can be accessed with the function names.

```
a<-1:5
names(a)
```

```
## NULL
```

```
names(a)<-letters[1:5]
print(a)
```

```
## a b c d e
## 1 2 3 4 5
```

To do computations for two vectors:

```
# Create two vectors.
v1 <- c(3,8,4,5,0,11)
v2 <- c(4,11,2,8,1,2)
v1+v2# Vector addition.
```

```
## [1]  7 19  6 13  1 13
```

```
v1-v2# Vector substraction.
```

```
## [1] -1 -3  2 -3 -1  9
```

```r
v1*v2# Vector multiplication.
```

```
## [1] 12 88  8 40  0 22
```

```r
v1/v2# Vector division.
```

```
## [1] 0.750 0.727 2.000 0.625 0.000 5.500
```

```r
#if v1 and v2 are not the same length
v11 <- c(3,8,4,5,0,11)
v21 <- c(4,11)
v11+v21#V21 becomes c(4,11,4,11,4,11)
```

```
## [1]  7 19  8 16  4 22
```

```r
v11-v21#V21 becomes c(4,11,4,11,4,11)
```

```
## [1] -1 -3  0 -6 -4  0
```

2.5.2 Lists

A list is a special type of vector which contain elements of different data types.

```r
# Create a list containing strings,  vectors, numbers and logical values.
mylist <- list("Red", "Green", c(21,32,11),  51.23, 169.1,TRUE)
print(mylist)
```

```
## [[1]]
## [1] "Red"
##
## [[2]]
## [1] "Green"
##
## [[3]]
## [1] 21 32 11
##
## [[4]]
## [1] 51.2
##
## [[5]]
## [1] 169
##
## [[6]]
## [1] TRUE
```

```r
str(mylist)# structure of an object
```

```
## List of 6
##  $ : chr "Red"
##  $ : chr "Green"
##  $ : num [1:3] 21 32 11
##  $ : num 51.2
##  $ : num 169
##  $ : logi TRUE
```

```r
names(mylist)<-c("color1","color2","value","weight","height","index")
mylist
```

```
## $color1
## [1] "Red"
##
## $color2
## [1] "Green"
##
## $value
## [1] 21 32 11
##
## $weight
## [1] 51.2
##
## $height
## [1] 169
##
## $index
## [1] TRUE
```

2.5.3 Factors

The factor stores the nominal values as a vector of integers in the range [1... k], where k is the number of unique values in the nominal variable.

```
# variable color with 30 "green" entries, 10 "red" entries, 1 "blue"entry and 1 "pink"entry#
color <- c(rep("green",30), rep("red", 10),"blue","pink")
color <- factor(color)
# 1=blue, 2=green, 3=pink, 4=red internally (alphabetically)
# R now treats color as a nominal variable
summary(color)
```

```
##  blue green  pink   red
##     1    30     1    10
```

Generating Factor Levels:

```
gl(n, k, labels)
```

n is a integer giving the number of levels; k is a integer giving the number of replications; labels is a vector of labels for the resulting factor levels.

Examples:

```
color <- gl(3, 4, labels = c("green", "red","blue"))
print(color)
```

```
##  [1] green green green green red   red   red   red   blue  blue  blue
## [12] blue
## Levels: green red blue
```

2.5.4 Matrices

A matrix is a 2 dimensional data structure. It consists of elements of same class.

```
# Elements are arranged sequentially by row.
mymatrix <- matrix(c(3:14), nrow = 4, byrow = TRUE)
print(mymatrix)
```

```
##      [,1] [,2] [,3]
## [1,]    3    4    5
## [2,]    6    7    8
## [3,]    9   10   11
## [4,]   12   13   14
```

```
# Elements are arranged sequentially by column.
mymatrix <- matrix(c(3:14), nrow = 4, byrow = FALSE)
print(mymatrix)
```

```
##      [,1] [,2] [,3]
## [1,]    3    7   11
## [2,]    4    8   12
## [3,]    5    9   13
## [4,]    6   10   14
```

For matrices, colnames and rownames are labels of the columns and rows, respectively. They can be accessed either with their corresponding functions, or with dimnames which returns a list with both vectors.

```
# Define the column and row names.
rownames <- c("row1", "row2", "row3", "row4")
colnames <- c("col1", "col2", "col3")
rownames(mymatrix)<-rownames
colnames(mymatrix)<-colnames
print(mymatrix)
```

```
##      col1 col2 col3
## row1    3    7   11
## row2    4    8   12
## row3    5    9   13
## row4    6   10   14
```

```
#another way for defining the column and row names
mymatrix <- matrix(c(3:14), nrow = 4, byrow = F, dimnames = list(rownames, colnames))
print(mymatrix)
```

```
##      col1 col2 col3
## row1    3    7   11
## row2    4    8   12
## row3    5    9   13
## row4    6   10   14
```

*To perform various mathematical operations.*The dimensions should be same for the matrices involved in the operation.

```
# Create two 3x2 matrices.
matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 3)
matrix1
```

```
##      [,1] [,2]
## [1,]    3    4
## [2,]    9    2
## [3,]   -1    6
```

```
matrix2 <- matrix(c(5, 2, 2, 9, 3, 4), nrow = 3)
matrix2
```

```
##      [,1] [,2]
## [1,]    5    9
```

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

```
matrix1 + matrix2# Add the matrices.
```

```
##      [,1] [,2]
## [1,]    8   13
## [2,]   11    5
## [3,]    1   10
```

```
matrix1 - matrix2# Subtract the matrices
```

```
##      [,1] [,2]
## [1,]   -2   -5
## [2,]    7   -1
## [3,]   -3    2
```

To transpose a matrix:

```
t(matrix1)
```

```
##      [,1] [,2] [,3]
## [1,]    3    9   -1
## [2,]    4    2    6
```

To extract the diagonal elements of a matrix:

```
diag(matrix1)
```

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

2.5.5 Arrays

Arrays are similar to matrices but can have more than two dimensions.For instance, if we are going to create an array of dimension (3, 3, 2), it means that we are creating 2 rectangular matrices each with 3 rows and 3 columns.

```
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
vector2 <- 10:15

# Take these vectors as input to the array.
myarray<- array(c(vector1,vector2),dim = c(3,3,2))
myarray
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    5   10   13
## [2,]    9   11   14
## [3,]    3   12   15
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    5   10   13
## [2,]    9   11   14
## [3,]    3   12   15
```

To give names to the rows, columns and matrices in the array by using the dimnames parameter:

```
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
vector2 <- 10:15
column.names <- c("col1","col2","colL3")
row.names <- c("row1","row2","row3")
matrix.names <- c("matrix1","matrix2")
# Take these vectors as input to the array.
myarray <- array(c(vector1,vector2),dim = c(3,3,2),dimnames = list(row.names,column.names,matrix.names))
myarray
```

```
## , , matrix1
##
##      col1 col2 colL3
## row1    5   10    13
## row2    9   11    14
## row3    3   12    15
##
## , , matrix2
##
##      col1 col2 colL3
## row1    5   10    13
## row2    9   11    14
## row3    3   12    15
```

To do the computations: the operations on elements of array are carried out by accessing elements of the matrices.

```
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
vector2 <- 10:15

# Take these vectors as input to the array.
array1 <- array(c(vector1,vector2),dim = c(3,3,2))

# Create two vectors of different lengths.
vector3 <- c(9,1,0)
vector4 <- c(6,0,11,3,14,1,2,6,9)
array2 <- array(c(vector1,vector2),dim = c(3,3,2))

# create matrices from these arrays.
matrix1 <- array1[,,2]
matrix2 <- array2[,,2]

# Add the matrices.
result <- matrix1+matrix2
result
```

```
##      [,1] [,2] [,3]
## [1,]   10   20   26
## [2,]   18   22   28
## [3,]    6   24   30
```

2.5.6 Data Frames

Data frames are the most commonly used member of data types family. A data frame is a generalisation of a matrix, in which different columns may have different modes. All elements of any column must have the same mode, i.e. all numeric or all factor, or all character.

```
myframe <- data.frame(name = c("Lucy","John","Mark","Candy"), score = c(67,56,87,91))
myframe
```

```
##    name score
## 1  Lucy    67
## 2  John    56
## 3  Mark    87
## 4 Candy    91
```

```
dim(myframe)#the dimention of the data frame
```

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

```
str(myframe)#returns the structure of a data frame i.e. the list of variables stored in the data frame
```

```
## 'data.frame':    4 obs. of  2 variables:
##  $ name : Factor w/ 4 levels "Candy","John",..: 3 2 4 1
##  $ score: num  67 56 87 91
```

```
summary(myframe)#obtain the statistical summary and nature of the data
```

```
##     name       score
##  Candy:1   Min.   :56.0
##  John :1   1st Qu.:64.2
##  Lucy :1   Median :77.0
##  Mark :1   Mean   :75.2
##            3rd Qu.:88.0
##            Max.   :91.0
```

To convert a vector/matrix to a dataframe:

```
x <- 1:6
y <- seq(20,70,by=10)
#to convert the vector to a dataframe
df<-data.frame(x,y)
print(df)
```

```
##   x  y
## 1 1 20
## 2 2 30
## 3 3 40
## 4 4 50
## 5 5 60
## 6 6 70
```

```
is.data.frame(df)#check whether it is a dataframe
```

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

```
#to convert the vectors to a matrix
mymatrix<-cbind(x, y)
print(mymatrix)
```

```
##      x  y
```

```
## [1,] 1 20
## [2,] 2 30
## [3,] 3 40
## [4,] 4 50
## [5,] 5 60
## [6,] 6 70
```

```r
is.matrix(mymatrix)#check whether it is a matrix
```

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

```r
is.data.frame(mymatrix)#check whether it is a dataframe
```

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

```r
#to convert the matrix to a dataframe
a.df<-as.data.frame(mymatrix)
print(a.df)
```

```
##   x  y
## 1 1 20
## 2 2 30
## 3 3 40
## 4 4 50
## 5 5 60
## 6 6 70
```

```r
is.matrix(a.df)
```

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

```r
is.data.frame(a.df)
```

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

It shows that the new object a.df is not a matrix but a data frame. If the row and column names were defined beforehand, those names would have remained.

Chapter 3

An overview of R: Part II {ch3}

3.1 Assess the values of an object

3.1.1 Using the index system

The index system is an efficient and flexible way to access selectively the elements of an object.

To refer to elements of a vector:

```
a<-seq(2,12,by=1.5)
print(a)
```

```
## [1]  2.0  3.5  5.0  6.5  8.0  9.5 11.0
```

```
a[2]
```

```
## [1] 3.5
```

```
a[c(1,4)] # 1st and 4th elements of vector
```

```
## [1] 2.0 6.5
```

```
a[2]<-0#to replace the value in the 2nd line with 0
print(a)
```

```
## [1]  2.0  0.0  5.0  6.5  8.0  9.5 11.0
```

To refer to elements of a dataframe or matrix

We can use the bracket notation to access the indices for the observations and the variables. If a is a data frame or a matrix, the value of the ith line and jth column is accessed with a[i, j].

When a is a dataframe:

```
a <- data.frame(name=letters[1:4],score=61:64,grade=1:4)
print(a)
```

```
##   name score grade
## 1    a    61     1
## 2    b    62     2
## 3    c    63     3
## 4    d    64     4
```

```
a[1,3] # 1st row, 3rd column of the dataframe
```

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

```
a[2:4,1:3] # rows 2,3,4 of columns 1,2,3
```

```
##   name score grade
## 2    b    62     2
## 3    c    63     3
## 4    d    64     4
```

```
#To replace an element of the dataframe:
a[2,3]<-0# replace the value in the 2nd row and 3rd column with 0
print(a)
```

```
##   name score grade
## 1    a    61     1
## 2    b    62     0
## 3    c    63     3
## 4    d    64     4
```

This also applies to the matrix. When a is a matrix:

```
a<-matrix(1:10,2,5)
print(a)
```

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

```
a[1,2]
```

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

```
a[1:2,2:4]
```

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

To refer to elements of an array

This index system is easily generalized to arrays. The number of indices should be the same as the number of dimensions of the array (i.e., a three dimensional array: a[i, j, k], a[, , 3], a[,3,,], and so on).

```
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
vector2 <- 10:15
# Take these vectors as input to the array.
myarray<- array(c(vector1,vector2),dim = c(3,3,2))
print(myarray)
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    5   10   13
## [2,]    9   11   14
## [3,]    3   12   15
##
## , , 2
```

```
##
##      [,1] [,2] [,3]
## [1,]    5   10   13
## [2,]    9   11   14
## [3,]    3   12   15
```

```
# Print the third row of the second matrix of the array.
myarray[3,,2]
```

```
## [1]  3 12 15
```

```
# Print the element in the 1st row and 3rd column of the 1st matrix.
myarray[1,3,1]
```

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

```
# Print the 2nd Matrix.
myarray[,,2]
```

```
##      [,1] [,2] [,3]
## [1,]    5   10   13
## [2,]    9   11   14
## [3,]    3   12   15
```

To refer to elements of a list

```
mylist <- list("Red", "Green", c(21,32,11),  51.23, 169.1,TRUE)
print(mylist)
```

```
## [[1]]
## [1] "Red"
##
## [[2]]
## [1] "Green"
##
## [[3]]
## [1] 21 32 11
##
## [[4]]
## [1] 51.2
##
## [[5]]
## [1] 169
##
## [[6]]
## [1] TRUE
```

```
#double brackets extract the object from the list
mylist[[3]]#Identify elements of a list using the [[]] convention.
```

```
## [1] 21 32 11
```

```
#single brackets: a list is returned
mylist[3]
```

```
## [[1]]
## [1] 21 32 11
```

The result can then be itself indexed as previously seen for vectors, matrices, etc. For instance, if the third object of a list is a vector, its ith value can be accessed using mylist[[3]][i], if it is a three dimensional array

using my.list[[3]][i, j, k], and so on.In this case:

```
mylist[[3]][2]
```

```
## [1] 32
```

```
mylist[1:2]#return a list with the first and second elements of the original list
```

```
## [[1]]
## [1] "Red"
##
## [[2]]
## [1] "Green"
```

It is possible to access the values of an element with a comparison expression as the index:

```
#vector
a<-1:10
print(a)
```

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

```
a[a<=6]<-0
print(a)
```

```
##  [1]  0  0  0  0  0  0  7  8  9 10
```

```
#matrix
a <- matrix(c(3:14), nrow = 4, byrow = F)
a[a>=6]
```

```
## [1]  6  7  8  9 10 11 12 13 14
```

```
#To apply a conditional statement using the which() function and replace a whole set of number:
a[which(a>=6)]<-99
print(a)
```

```
##      [,1] [,2] [,3]
## [1,]    3   99   99
## [2,]    4   99   99
## [3,]    5   99   99
## [4,]   99   99   99
```

```
#The function which() is very useful when accessing and replacing data in vector or matrix format.
```

3.1.2 Using the names

If the elements of an object have names, they can be extracted by using them as indices.The names are labels of the elements of an object.There are several kinds of names (names, colnames, rownames, dimnames).

If a is a vector

```
a<-1:5
names(a)<-letters[1:5]
print(a)
```

```
## a b c d e
## 1 2 3 4 5
```

```
a["b"]
```

```
## b
## 2
```

If a is a matrix

```
a<-matrix(1:12,4,3)
colnames(a)<-c("col1","col2","col3")
rownames(a)<-c("row1","row2","row3","row4")
print(a)
```

```
##      col1 col2 col3
## row1    1    5    9
## row2    2    6   10
## row3    3    7   11
## row4    4    8   12
```

```
a["row1","col2"]#to identify the element with rowname "row1" and colname "col2"
```

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

```
a[c("row1","row4"),c("col1","col3")]
```

```
##      col1 col3
## row1    1    9
## row4    4   12
```

If a is a dataframe

```
a=as.data.frame(a)
print(a)
```

```
##      col1 col2 col3
## row1    1    5    9
## row2    2    6   10
## row3    3    7   11
## row4    4    8   12
```

```
a["row2",]#access the second row
```

```
##      col1 col2 col3
## row2    2    6   10
```

```
a$col1#To extract a vector or a factor from a data frame, we can use the operator $.
```

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

If a is a list

```
a <- list("Red", "Green", c(21,32,11),  51.23, 169.1,TRUE)
names(a)<-c("color1","color2","value","weight","height","index")
print(a)
```

```
## $color1
## [1] "Red"
##
## $color2
## [1] "Green"
##
## $value
```

```
## [1] 21 32 11
##
## $weight
## [1] 51.2
##
## $height
## [1] 169
##
## $index
## [1] TRUE
```

```
a["height"]
```

```
## $height
## [1] 169
```

```
a$height
```

```
## [1] 169
```

If a is an array

```
vector1 <- c(5,9,3)
vector2 <- 10:15
column.names <- c("col1","col2","colL3")
row.names <- c("row1","row2","row3")
matrix.names <- c("matrix1","matrix2")
a <- array(c(vector1,vector2),dim = c(3,3,2),dimnames = list(row.names,column.names,matrix.names))
print(a)
```

```
## , , matrix1
##
##      col1 col2 colL3
## row1    5   10    13
## row2    9   11    14
## row3    3   12    15
##
## , , matrix2
##
##      col1 col2 colL3
## row1    5   10    13
## row2    9   11    14
## row3    3   12    15
```

```
a["row1","col2","matrix1"]
```

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

```
a[,"col1","matrix2"]
```

```
## row1 row2 row3
##    5    9    3
```

3.2 Get the data into R

3.2.1 Direct data entering

```
mydata<-matrix(c(50,65,1,0,120,134),2,3)
colnames(mydata)<-c("Age","sex","BP")
rownames(mydata)<-c("ID=1","ID=2")
print(mydata)
```

```
##      Age sex  BP
## ID=1  50   1 120
## ID=2  65   0 134
```

3.2.2 Use dataset that come with R or R packages

If the aim is just to play with some test data to see how they load and what basic functions can be run, the default installation of R comes with several data sets.

```
data()
```

Example:

```
install.packages("ISwR")
library(ISwR)
data(stroke)
```

3.2.3 Read in data files

It is necessary to import data into R before we start working on our analysis. R offers wide range of packages for importing data available in any format. Some functions are available by default: read.table(), read.csv(), read.csv2(), read.delim() and read.delim2(). There are also a number of packages that will read files from Excel, SPSS, SAS, Stata and various relational databases.

Note:For reading and writing in files, R assumes the data files are located in the working directory. It is necessary to give the path to a file if it is not in the working directory.

```
#get current working directory
getwd()
# set working directory and thus avoid entering the full path of the data files
setwd("new path")
```

Table file (if the separator is a tab):

```
mydata<-read.table("data/stroke.txt",header=T)#read text file
head(mydata)
```

```
##      sex       died       dstr age dgn coma diab minf han  dead obsmonths
## 1   Male 1991-01-07 1991-01-02  76 INF   No   No  Yes  No  TRUE    0.1634
## 2   Male       <NA> 1991-01-03  58 INF   No   No   No  No FALSE   59.6078
## 3   Male 1991-06-02 1991-01-08  74 INF   No   No  Yes Yes  TRUE    4.7386
## 4 Female 1991-01-13 1991-01-11  77 ICH   No  Yes   No Yes  TRUE    0.0654
## 5 Female       <NA> 1991-01-13  76 INF   No  Yes   No Yes FALSE   59.2810
## 6   Male 1991-01-13 1991-01-13  48 ICH  Yes   No   No Yes  TRUE    0.1000
```

For further detail of the function read.table,

```
help(read.table)
```

CSV file (if the separater is a comma):

```
mydata<-read.csv("data/stroke.csv",header=T)
dim(mydata)
```

```
## [1] 829   9
```

```
head(mydata)
```

```
##      sex       dstr age coma diab minf han  dead obsmonths
## 1   Male 1991-01-02  76   No   No  Yes  No  TRUE    0.1634
## 2   Male 1991-01-03  58   No   No   No  No FALSE   59.6078
## 3   Male 1991-01-08  74   No   No  Yes Yes  TRUE    4.7386
## 4 Female 1991-01-11  78   No  Yes   No Yes  TRUE    0.0654
## 5 Female 1991-01-13  76   No  Yes   No Yes FALSE   59.2810
## 6   Male 1991-01-13  48  Yes   No   No Yes  TRUE    0.1000
```

The argument header = TRUE tells R that the first row of the data are the labels of every column. If set header =FALSE, it means the first row of the data are not the labels, but are considered as data points.

```
mydata<-read.csv("data/stroke.csv",header=F)
dim(mydata)
```

```
## [1] 830   9
```

```
head(mydata)
```

```
##       V1         V2  V3   V4   V5   V6  V7    V8          V9
## 1    sex       dstr age coma diab minf han  dead   obsmonths
## 2   Male 1991-01-02  76   No   No  Yes  No  TRUE 0.163398693
## 3   Male 1991-01-03  58   No   No   No  No FALSE 59.60784314
## 4   Male 1991-01-08  74   No   No  Yes Yes  TRUE 4.738562092
## 5 Female 1991-01-11  78   No  Yes   No Yes  TRUE 0.065359477
## 6 Female 1991-01-13  76   No  Yes   No Yes FALSE 59.28104575
```

In this case, R will read the first line as data, not column headers (and assigns default column header names we can change later).

For further detail of `read.csv()`, type `help(read.csv)`

EXCEl file

we can use the function read.xls from the gdata package. It reads from an Excel spreadsheet and returns a data frame.

```
library(gdata)                    # load gdata package
help(read.xls)                    # documentation
mydata <- read.xls("myfile.xls")  # read from EXCEL
```

Minitab File

If the data file is in Minitab Portable Worksheet format, it can be opened with the function read.mtp from the foreign package.

```
library(foreign)                  # load the foreign package
help(read.mtp)                    # documentation
mydata <- read.mtp("myfile.mtp")  # read from .mtp file
```

SPSS File

For the data files in SPSS format, it can be opened with the function read.spss from the foreign package.

```
library(foreign)                # load the foreign package
help(read.spss)                 # documentation
mydata <- read.spss("myfile.sav", to.data.frame=TRUE)#"to.data.frame" option for choosing whether a data
```

SAS file

For the data files in in SAS format, it can be done by using function sasxport.get from the Hmisc package.

```
library(Hmisc) # load the  package
help(sasxport.get) # documentation
mydata <- sasxport.get("myfile.xpt") #read from SAS file
```

STATA file

```
library(foreign)# load the  package
help(read.dta)# documentation
mydata <- read.dta("myfile.dta")#read from STATA file
```

3.2.4 Read in data from the clipboard

If we've got just a small section of data already in a table, we can copy those data to the Windows clipboard and import them into R using the argument "clipboard" within the read.table() function.

```
mydata <- read.table(file = "clipboard", sep="\t", header=TRUE)#it handles clipboard data with a header
```

3.2.5 Read in remote data

The above functions work pretty much the same to access files from the Web as they do for local data.

```
mydata <- read.dta("https://stats.idre.ucla.edu/stat/data/test.dta")
```

3.3 Export data

Sometimes we may want to export the data from R to another format, such as TXT file and CSV file. There are numerous methods for exporting R objects into other formats.In the codes below, df is the name of data in R, and mydata is the new data name.

Exporting data to TXT (Tab Delimited Text File):

```
write.table(df, "mydata.txt", sep=",")
```

Exporting data to CSV:

```
write.csv(df, file="data/mydata.csv",row.names=F)
```

Exporting data to SPSS:

```
library(foreign)
write.foreign(df, "mydata.txt", "mydata.sps",   package="SPSS")
```

Exporting data to Stata:

```
library(foreign)
write.dta(df, "mydata.dta")
```

Export data to SAS:

```
library(foreign)
write.foreign(df, "mydata.txt", "mydata.sas", package="SAS")
```

3.4 Save and load data in R

The function used for saving the dataframe is:

```
save(objectlist, file="myfile")
```

where objectlist is the name of the current dataframe and myfile is the filename of RDATA we will save on the computer.

The function to upload the Rdata in R is load().

```
load("myfile.Rdata")
```

3.5 View data

Now we have imported the data into R. There are a few things that we need to do right after we import the data to R.

```r
mydata<-read.csv("data/stroke.csv",header=T)
#Check the dimensions (i.e number of rows and columns) of the dataset
dim(mydata)
```

```
## [1] 829   9
```

```r
#list the variables in mydata
names(mydata)
```

```
## [1] "sex"       "dstr"      "age"       "coma"      "diab"      "minf"
## [7] "han"       "dead"      "obsmonths"
```

```r
#To get information about the structure of dataset (i.e if variable is numeric or factor)
str(mydata)
```

```
## 'data.frame':    829 obs. of  9 variables:
##  $ sex      : Factor w/ 2 levels "Female","Male": 2 2 2 1 1 2 1 2 2 1 ...
##  $ dstr     : Factor w/ 575 levels "1991-01-02","1991-01-03",..: 1 2 3 4 5 5 6 6 7 7 ...
##  $ age      : int  76 58 74 78 76 48 81 53 78 69 ...
##  $ coma     : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 2 1 1 1 1 ...
##  $ diab     : Factor w/ 2 levels "No","Yes": 1 1 1 2 2 1 1 1 1 1 ...
##  $ minf     : Factor w/ 2 levels "No","Yes": 2 1 2 1 1 1 1 2 1 1 ...
##  $ han      : Factor w/ 2 levels "No","Yes": 1 1 2 2 2 2 2 2 2 2 ...
##  $ dead     : logi  TRUE FALSE TRUE TRUE FALSE TRUE ...
##  $ obsmonths: num  0.1634 59.6078 4.7386 0.0654 59.281 ...
```

```r
#Look the header of the dataset to get information about the variables and their values
head(mydata,n=10)
```

```
##       sex       dstr age coma diab minf han  dead obsmonths
## 1    Male 1991-01-02  76   No   No  Yes  No  TRUE    0.1634
## 2    Male 1991-01-03  58   No   No   No  No FALSE   59.6078
## 3    Male 1991-01-08  74   No   No  Yes Yes  TRUE    4.7386
## 4  Female 1991-01-11  78   No  Yes   No Yes  TRUE    0.0654
## 5  Female 1991-01-13  76   No  Yes   No Yes FALSE   59.2810
## 6    Male 1991-01-13  48  Yes   No   No Yes  TRUE    0.1000
## 7  Female 1991-01-14  81   No   No   No Yes  TRUE   34.3791
## 8    Male 1991-01-14  53   No   No  Yes Yes  TRUE   10.8497
## 9    Male 1991-01-15  78   No   No   No Yes FALSE   59.2157
```

```
## 10 Female 1991-01-15  69   No   No   No Yes  TRUE   33.6601
```

```r
## print last 5 rows of mydata
tail(mydata, n=5)
```

```
##        sex       dstr age coma diab minf han  dead obsmonths
## 825 Female 1993-12-26  55   No  Yes  Yes Yes FALSE    24.052
## 826 Female 1993-12-29  93   No   No   No  No  TRUE     5.654
## 827 Female 1993-12-31  81  Yes   No   No  No  TRUE     0.882
## 828 Female 1993-12-31  68   No   No   No Yes FALSE    23.889
## 829 Female 1993-12-31  74   No   No   No Yes  TRUE     0.425
```

```r
##test for missing values
sum(is.na(mydata))
```

```
## [1] 29
```

```r
sum(is.na(mydata$sex))
```

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

```r
sum(is.na(mydata$died))
```

```
## Warning in is.na(mydata$died): is.na() applied to non-(list or vector) of
## type 'NULL'
```

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

Chapter 4

Data management: Part I {ch4}

4.1 Control structure

R has the standard control structures.

4.1.1 For loop

It is used when a loop is to be executed a specific number of times.

```
for (var in seq) {statement}
```

Example:

```
#initialize a vector
y <- c(99,45,34,65,76,23)
#print the first 4 numbers of this vector
for(i in 1:4){
     print (y[i])
}
```

```
## [1] 99
## [1] 45
## [1] 34
## [1] 65
```

4.1.2 While Loop

The While loop executes the same code again and again until a stop condition is met.

```
while (test_expression) {
   statement
}
```

Example:

```
#initialize a condition
age <- 12
#check if age is less than 17
while(age < 17){
         print(age)
```

```
         age <- age + 2
}
```

```
## [1] 12
## [1] 14
## [1] 16
```

4.1.3 If-else Loop

This structure is used to test a condition.

```
if (<condition>){
         ##do something
} else {
         ##do something
}
```

Example:

```
#initialize a variable
Age<-30
#check if this variable * 2-5 is > 50
if (Age*2-5 >50 ){
       print("right")
} else {
       print ("do it again")
}
```

```
## [1] "right"
```

4.1.4 Repeat Loop

It executes an infinite loop.

```
repeat {
   commands
   if(condition) {
      break
   }
}
```

Example:

```
a<- c("Good","Morning")
b <- 2

repeat {
   print(a)
   b <-b+1

   if(b > 5) {
      break
   }
}
```

```
## [1] "Good"    "Morning"
## [1] "Good"    "Morning"
## [1] "Good"    "Morning"
## [1] "Good"    "Morning"
```

4.2 Missing values

Missing values in R are represented by NA (not available). Impossible values (e.g., dividing by zero) are represented by the symbol NaN (not a number).

```
myframe <- data.frame(name = c("Lucy","John","Mark","Candy"), score = c(67,56,87,91))
myframe[1:2,2] <- NA #injecting NA at 1st, 2nd row and 2nd column of df
myframe
```

```
##    name score
## 1  Lucy    NA
## 2  John    NA
## 3  Mark    87
## 4 Candy    91
```

```
is.na(myframe) #checks the entire data set for NAs and return logical output
```

```
##       name score
## [1,] FALSE  TRUE
## [2,] FALSE  TRUE
## [3,] FALSE FALSE
## [4,] FALSE FALSE
```

```
table(is.na(myframe)) #returns a table of logical output
```

```
##
## FALSE  TRUE
##     6     2
```

```
myframe[!complete.cases(myframe),] #returns a logical vector indicating which cases are not complete.
```

```
##   name score
## 1 Lucy    NA
## 2 John    NA
```

Missing values when using read.table()

When we are using the function read.table(), we expect missing values to be coded as NA. However, it is not always the case. For instance, if we have a text file that has been exported from SAS, the missing values are indicated by ".". In that case, we can use "na.strings"" to define the missing values (na.strings=c(".")).

It is also possible that there are multiple missing value indicators,we can use na.strings=c("NA",".","*","") to ensure that all these symbols are entered as NAs.

Missing values hinder normal calculations in a data set.

```
mean(myframe$score)
```

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

```
mean(myframe$score,na.rm =T)#ignore the NAs and compute the mean of remaining values in the selected co
```

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

```
new_myframe <- na.omit(myframe)#To remove rows with NA values in a data frame
new_myframe
```

```
##    name score
## 3  Mark    87
## 4 Candy    91
```

4.3 Dates

```
Sys.Date( )
```

```
## [1] "2018-01-05"
```

```
#returns today's date.
date()
```

```
## [1] "Fri Jan  5 14:00:49 2018"
```

```
#returns the current date and time.
```

Use as.Date( ) to convert strings to dates. The default is that Year-Month-Day. Therefore,

```
strdd<-c("2013/08/24","2013/11/23","2014/02/22","2014/05/23")
dd <- as.Date(strdd)
# number of days between two consecutive dates
diff(dd)
```

```
## Time differences in days
## [1] 91 91 90
```

```
dd[1]-dd[2]#calculations for the date
```

```
## Time difference of -91 days
```

Use format() to set or change the way that a date is formatted.The following symbols can be used with the format( ) function to print dates.The default format is "%Y-%m-%d".

```
%d: day, as number (i.e.,01-31)
%a: abbreviated weekday name (Mon)
%A: unabbreviated (Monday)
%m: month (00-12)
%b: month abbreviated name (Jan)
%B: unabbreviated (January)
%y: final two digits of year(17)
%Y: all four digits (2017)
```

Examples:

```
today <- Sys.Date()
format(today, format="%b %d %Y")
```

```
## [1] "Jan 05 2018"
```

```
format(today, format="%A %b %d %Y")
```

```
## [1] "Friday Jan 05 2018"
```

```
as.Date("1/12/1960",format="%d/%m/%Y")
```

```
## [1] "1960-12-01"
```

By default, dates are stored using 1970-01-01 as origin, with negative values for earlier dates.

```
#as.integer attempts to coerce its argument to be of integer type.
as.integer(as.Date("1/12/1960","%d/%m/%Y"))#attempt to coerce its argument to be of integer type.
```

```
## [1] -3318
```

```
as.integer(as.Date("1/1/1970","%d/%m/%Y"))
```

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

```
as.integer(as.Date("1/1/2017","%d/%m/%Y"))
```

```
## [1] 17167
```

4.4 Common useful functions

There are some useful functions that are widely used in regular R programming.

4.4.1 Numeric functions

Here are some examples:

```
abs(-3)#absolute value
```

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

```
sqrt(4) #square root
```

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

```
exp(5)#e^x
```

```
## [1] 148
```

```
log(3)  #natural logarithm
```

```
## [1] 1.1
```

```
log10(5)    #common logarithm
```

```
## [1] 0.699
```

```
ceiling(3.141592657)#takes a single numeric argument x and returns a numeric vector containing the smal
```

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

```
floor(3.141592657)#takes a single numeric argument x and returns a numeric vector containing the larges
```

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

```
trunc(3.141592657)#takes a single numeric argument x and returns a numeric vector containing the intege
```

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

```
round(3.141592657,3)#rounds the values in its first argument to the specified number of decimal places
```

```
## [1] 3.14
```

```r
signif(3.141592657,3)#rounds the values in its first argument to the specified number of significant di
```

```
## [1] 3.14
```

4.4.2 Character Functions

Extract or replace substrings in a character vector:

```
substr(x, start=n1, stop=n2)
```

Example:

```r
x <- "abcdef"
substr(x, 2, 4)
```

```
## [1] "bcd"
```

```r
substr(x, 2, 4) <- "22222"
x
```

```
## [1] "a222ef"
```

Search for pattern in x:

```
grep(pattern, x , ignore.case=FALSE, fixed=FALSE)
```

If fixed =FALSE then pattern is a regular expression. If fixed=TRUE then pattern is a text string. Returns matching indices.

Example:

```r
grep("A", c("b","A","c"), fixed=TRUE)
```

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

Find pattern in x and replace with replacement text:

```
sub(pattern, replacement, x, ignore.case =FALSE, fixed=FALSE)
```

If fixed=FALSE then pattern is a regular expression.If fixed = T then pattern is a text string.

Example:

```r
sub("\\s",".","Good morning")
```

```
## [1] "Good.morning"
```

```r
sub("ca","*","canada")
```

```
## [1] "*nada"
```

Split the elements of character vector x at split:

```
strsplit(x, split)
```

Example:

```r
strsplit("abc", "b")
```

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

```r
strsplit("abc", "")
```

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

Concatenate strings after using sep string to seperate them:

```
paste(..., sep="")
```

Example:

```r
paste("x",1:3,sep="")
```

```
## [1] "x1" "x2" "x3"
```

```r
paste("x",1:3,sep="M")
```

```
## [1] "xM1" "xM2" "xM3"
```

```r
paste("Today is", date())
```

```
## [1] "Today is Fri Jan  5 14:00:49 2018"
```

Force the characters to be uppercase or lowercase:

```
toupper(x)  Uppercase
tolower(x)  Lowercase
```

Example:

```r
x="abcdeT"
toupper(x)
```

```
## [1] "ABCDET"
```

```r
tolower(x)
```

```
## [1] "abcdet"
```

4.4.3 Apply functions over a list, array, dataframe or matrix

The function apply() can be used on data frames as well as matrices and arrays. *For dataframe*

```r
mydata <- data.frame(height=round(runif(5,160,180),2),weight=round(runif(5,40,70),2))
apply(mydata,2,mean)## All elements must be numeric!
```

```
## height weight
##  171.8   53.1
```

For matrix:

```r
mydata<-matrix(1:12,3,4)
colnames(mydata)=letters[1:4]
apply(mydata,2,summary)
```

```
##           a   b   c    d
## Min.    1.0 4.0 7.0 10.0
## 1st Qu. 1.5 4.5 7.5 10.5
## Median  2.0 5.0 8.0 11.0
## Mean    2.0 5.0 8.0 11.0
## 3rd Qu. 2.5 5.5 8.5 11.5
## Max.    3.0 6.0 9.0 12.0
```

For arrays:

```r
vector1 <- c(5,9,3)
vector2 <- 10:15
# Take these vectors as input to the array.
myarray <- array(c(vector1,vector2),dim = c(3,3,2))
print(myarray)
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    5   10   13
## [2,]    9   11   14
## [3,]    3   12   15
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    5   10   13
## [2,]    9   11   14
## [3,]    3   12   15
```

```r
# Use apply to calculate the sum/mean of the rows/columns across all the matrices.
result <- apply(myarray, c(1), sum)
print(result)
```

```
## [1] 56 68 60
```

```r
result <- apply(myarray, c(2), mean)
print(result)
```

```
## [1]  5.67 11.00 14.00
```

The function sapply() can be useful for getting information about the columns of a data frame.

```r
mydata<-read.csv("data/stroke.csv",header=T)
head(mydata)
```

```
##      sex       dstr age coma diab minf han  dead obsmonths
## 1   Male 1991-01-02  76   No   No  Yes  No  TRUE    0.1634
## 2   Male 1991-01-03  58   No   No   No  No FALSE   59.6078
## 3   Male 1991-01-08  74   No   No  Yes Yes  TRUE    4.7386
## 4 Female 1991-01-11  78   No  Yes   No Yes  TRUE    0.0654
## 5 Female 1991-01-13  76   No  Yes   No Yes FALSE   59.2810
## 6   Male 1991-01-13  48  Yes   No   No Yes  TRUE    0.1000
```

```r
sapply(mydata,is.factor)
```

```
##       sex      dstr       age      coma      diab      minf       han
##      TRUE      TRUE     FALSE      TRUE      TRUE      TRUE      TRUE
##      dead obsmonths
##     FALSE     FALSE
```

```r
#sex,dstr,coma,diab,minf and han are factors
sapply(mydata, function(x)if(!is.factor(x))return(0) else length(levels(x)))
```

```
##       sex      dstr       age      coma      diab      minf       han
##         2       575         0         2         2         2         2
##      dead obsmonths
##         0         0
```

```
# This answers the question that how many levels does each factor have?
```

4.4.4 Other useful functions

R has many functions for manipulating data. Some of them are listed:

```
age=c(1,6,4,5,8,5,4,3)
weight=c(45,65,34)
print(age)# Print a single R object
```

```
## [1] 1 6 4 5 8 5 4 3
```

```
cat(age,weight)# Print multiple objects, one after the other
```

```
## 1 6 4 5 8 5 4 3 45 65 34
```

```
mean(age)
```

```
## [1] 4.5
```

```
prod(age)#product of the elements
```

```
## [1] 57600
```

```
median(age)
```

```
## [1] 4.5
```

```
range(age)
```

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

```
var(age)#variance of the elements of x (calculated on n  1)
```

```
## [1] 4.29
```

```
sd(age)#standard deriviation
```

```
## [1] 2.07
```

```
max(age)
```

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

```
min(age)
```

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

```
which.max(age)#returns the index of the greatest element of x
```

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

```
which.min(age)#returns the index of the smallest element of x
```

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

```
quantile(age)#returns the minimum, 25%, 50%, 75% and maximum value
```

```
##   0%  25%  50%  75% 100%
## 1.00 3.75 4.50 5.25 8.00
```

```r
unique(age)# Gives the vector of distinct values
```

```
## [1] 1 6 4 5 8 3
```

```r
diff(age)# Replace a vector by the vector of first differences
```

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

```r
sort(age)# Sort elements into order
```

```
## [1] 1 3 4 4 5 5 6 8
```

```r
age[order(age)]#x[order(x)] orders elements of x
```

```
## [1] 1 3 4 4 5 5 6 8
```

```r
cumsum(age)#cumulative sums
```

```
## [1]  1  7 11 16 24 29 33 36
```

```r
cumprod(age)#cumulative products
```

```
## [1]     1     6    24   120   960  4800 19200 57600
```

```r
rev(age)# reverse the order of vector elements
```

```
## [1] 3 4 5 8 5 4 6 1
```

```r
cut(age, 5)#divide continuous variable in factor with n levels
```

```
## [1] (0.993,2.4] (5.2,6.6]   (3.8,5.2]   (3.8,5.2]   (6.6,8.01]  (3.8,5.2]
## [7] (3.8,5.2]   (2.4,3.8]
## Levels: (0.993,2.4] (2.4,3.8] (3.8,5.2] (5.2,6.6] (6.6,8.01]
```

The functions mean(), median(), range(), and a number of other functions, take the argument na.rm=T; i.e. remove NAs, then proceed with the calculation. Note: Function sort() and order() deal with missing data in different ways.

```r
x <- c(1, 15, 2, NA, 25)
sort(x)#The function sort() omits any NAs
```

```
## [1]  1  2 15 25
```

```r
order(x)#The function order() places NAs last.
```

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

```r
x[order(x)]
```

```
## [1]  1  2 15 25 NA
```

4.4.5 Write functions

In R, we can also write our own functions. This can be done by using Function() function.

```
func_name <- function (argument) {
   statement
}
```

Example 1:

```
mypow <- function(x, y) {
   # function to print x raised to the power y
   result <- x^y
   print(paste(x,"^", y, "is", result))
}
#Here, we created a function called mypow().It takes two arguments, finds the first argument raised to
```

To call this function:

```
mypow(2,4)
```

```
## [1] "2 ^ 4 is 16"
```

```
mypow(x=2,y=4)
```

```
## [1] "2 ^ 4 is 16"
```

Example 2:

```
basic <- function(x) {
  result=round(mean(x)/sd(x)+median(x),2)
   return(result)
}
```

To call the function:

```
basic(1:10)
```

```
## [1] 7.32
```

```
basic(c(2,3,4,5,6))
```

```
## [1] 6.53
```

```
#for a dataframe
mydata=data.frame(x1=c(2,3,0,-3),x2=c(0,3,4,5))
apply(mydata,2,basic)
```

```
##   x1   x2
## 1.19 4.89
```

Chapter 5

Data managements: Part II- Reshape data {ch4}

Data Reshaping in R is to change the way data is organized. There are situations when we need the data frame in a format that is different from the format in which we received it. R provides a variety of methods to split, merge and change the rows to columns and vice-versa in a data frame.

5.1 Subset Data

It is very common to subset a dataset from the original dataset.

The following codes are used to create the data frame "mydata"" containing the variables "name"",”sex"",“height"",and"weight".”

```
#create a dataframe
set.seed(2017)#this is to make sure the dataset can be replicated.
mydata <- data.frame(name=letters[6:10],  sex=sample(1:2,5,replace=T), height=round(runif(5,160,180),2)
print(mydata)
```

```
##   name sex height weight
## 1    f   2    175   60.2
## 2    g   2    161   40.1
## 3    h   1    169   40.8
## 4    i   1    169   53.0
## 5    j   2    165   55.0
```

5.1.1 Select variables

(1) As we described before, we can use index system to select variables.

In order to know which variables correspond to which number in the index, we use the names function, which will list the names of the variables in the order in which they appear in the data frame.

```
names(mydata)
```

```
## [1] "name"   "sex"    "height" "weight"
```

```
#From this list we see that name is variable 1, sex is variable 2, height is variable 3 and weight is v
```

When we only want to subset variables (or columns), we use the second index and leave the first index blank. Leaving an index blank indicates that we want to keep all the elements in that dimension.

```
b<-mydata[,1:2]#select name and sex
print(b)
```

```
##   name sex
## 1    f   2
## 2    g   2
## 3    h   1
## 4    i   1
## 5    j   2
```

```
b<-mydata[,2]
print(b)
```

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

Note that the last result is a vector but not a matrix. The default behaviour of R is to return an object of the lowest dimension possible. This can be altered with the option "drop"" (the default is TRUE):

```
b<-mydata[,2,drop=FALSE]
print(b)
```

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

(2) We can choose to access the specific columns by the column names:

```
b<-mydata[,c("name","sex")]
print(b)
```

```
##   name sex
## 1    f   2
## 2    g   2
## 3    h   1
## 4    i   1
## 5    j   2
```

```
b<-mydata[,"sex"]
print(b)
```

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

```
b<-mydata[,"sex",drop=F]
print(b)
```

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

(3) We can also use subset function to select variables:

```
#In the code below, we are telling R to select variables name and sex.
b<-subset(mydata,select=c(1,2))
print(b)
```

```
##   name sex
## 1    f   2
## 2    g   2
## 3    h   1
## 4    i   1
## 5    j   2
```

```
b<-subset(mydata,select=1:2)#select consecutive columns (the first and second column)
print(b)
```

```
##   name sex
## 1    f   2
## 2    g   2
## 3    h   1
## 4    i   1
## 5    j   2
```

5.1.2 Exclude variables

There are several ways to exclude variables.

(1) The most easiest way to drop columns is by using subset() function.The '-' sign indicates dropping variables.

```
df<-subset(mydata,select=-c(weight,height))
print(df)
```

```
##   name sex
## 1    f   2
## 2    g   2
## 3    h   1
## 4    i   1
## 5    j   2
```

Make sure the variable names would NOT be specified in quotes when using subset() function.

```
df.0<-subset(mydata,select=-c("weight","height"))
print(df.0)
Error in -c("weight", "height") : invalid argument to unary operator
```

(2) In the codes below, we are creating a character vector named drop in which we are storing column names "weight" and "height". Then we are telling R to select all the variables except the column names specified in the vector "drop".

```
drop<-c("weight","height")
df <- mydata[,!(names(mydata) %in% drop)]#The function names() returns all the column names and the '!'
print(df)
```

```
##   name sex
## 1    f   2
## 2    g   2
## 3    h   1
## 4    i   1
```

```
## 5    j   2
```

(3) Drop columns by column index numbers. In the following codes, we are telling R to drop variables that are positioned at third and fourth columns. The minus sign is to drop variables.

```
df <- mydata[,-c(3:4) ]
print(df)
```

```
##   name sex
## 1    f   2
## 2    g   2
## 3    h   1
## 4    i   1
## 5    j   2
```

```
df<-mydata[,-c(1,3)]
print(df)
```

```
##   sex weight
## 1   2   60.2
## 2   2   40.1
## 3   1   40.8
## 4   1   53.0
## 5   2   55.0
```

(4) set the column to NULL

```
df<-mydata
df[,3:4]<-list(NULL)
print(df)
```

```
##   name sex
## 1    f   2
## 2    g   2
## 3    h   1
## 4    i   1
## 5    j   2
```

```
df<-mydata
df[,3]<-NULL
print(df)
```

```
##   name sex weight
## 1    f   2   60.2
## 2    g   2   40.1
## 3    h   1   40.8
## 4    i   1   53.0
## 5    j   2   55.0
```

5.1.3 Select Observations

If we want to select specific observations:

```
# first 3 observations
df <- mydata[1:3,]
print(df)
```

```
##   name sex height weight
```

```
## 1    f   2    175   60.2
## 2    g   2    161   40.1
## 3    h   1    169   40.8
```

```
# use subset function
df<- subset(mydata, height >= 165 & sex == 2)#select observations with height more than 165 and sex equ
print(df)
```

```
##   name sex height weight
## 1    f   2    175   60.2
## 5    j   2    165   55.0
```

5.1.4 Select both variables and observations

In the following codes, we are creating the data frame df in which we keep only the variables name, sex and height and only the observations where height greater than or equal to 165 and sex equal to 2.

```
df<- subset(mydata, height >= 165 & sex == 2,select=c("name","sex","height"))#select observations with
print(df)
```

```
##   name sex height
## 1    f   2    175
## 5    j   2    165
```

```
#another way to do it#####
df<-mydata[which(mydata$height>=165 & mydata$sex==2),c("name","sex","height")]
print(df)
```

```
##   name sex height
## 1    f   2    175
## 5    j   2    165
```

5.1.5 Keep or delete variables using dplyr function

n R, the dplyr package is one of the most popular package for data manipulation (https://cran.r-project.org/web/packages/dplyr/index.html).

```
install.packages("dplyr")
```

Some examples for data manipulation with "dplyr" function:

```
library(dplyr)
#to delete the first, second and fouth column
mydata2 <- select(mydata, -1:-2, -4)
print(mydata2)
```

```
##   height
## 1    175
## 2    161
## 3    169
## 4    169
## 5    165
```

```
#to delete columns name, sex and weight

#Method 1
```

```r
mydata2 <- select(mydata, -name, -sex, -weight)
print(mydata2)
```

```
##   height
## 1    175
## 2    161
## 3    169
## 4    169
## 5    165
```

```r
#Method 2
mydata2 <- select(mydata, -c(name, sex, weight))
print(mydata2)
```

```
##   height
## 1    175
## 2    161
## 3    169
## 4    169
## 5    165
```

```r
#Method 3
mydata2 <- select(mydata, -name:-sex,-weight)
print(mydata2)
```

```
##   height
## 1    175
## 2    161
## 3    169
## 4    169
## 5    165
```

```r
#to keep columns height
mydata2 <- select(mydata, height)
print(mydata2)
```

```
##   height
## 1    175
## 2    161
## 3    169
## 4    169
## 5    165
```

There is another function "tidyr"(https://cran.r-project.org/web/packages/tidyr/index.html), which you can refer to for data manipulation.

5.1.6 Keep/drop variables by name pattern

The codes below are creating data for 5 variables: age,sex,test_blood,test_pressure, height_morning,weight_morning.

```r
mydata <- read.table(text="age sex test_blood test_pressure height_morning weight_morning
25 1 30 120 165 70
34 1 38 134 170 65
45 2 28 132 175 50", header=TRUE)
print(mydata)
```

```
##   age sex test_blood test_pressure height_morning weight_morning
## 1  25   1         30           120            165             70
## 2  34   1         38           134            170             65
## 3  45   2         28           132            175             50
```

To Keep columns whose name starts with "test":

```
nam<-grepl("^test",names(mydata))#The grepl() function is used to search for matches to a pattern.In th
print(nam)#It returns TRUE for "test_blood"" and "test_pressure".
```

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

```
mydata1 <- mydata[,nam]
print(mydata1)
```

```
##   test_blood test_pressure
## 1         30           120
## 2         38           134
## 3         28           132
```

To keep columns whose name contains "morning" at the end:

```
mydata12 <- mydata[,grepl("morning$",names(mydata))]
print(mydata12)
```

```
##   height_morning weight_morning
## 1            165             70
## 2            170             65
## 3            175             50
```

The "$" is used to search for the sub-strings at the end of string. It returns "height_morning" and "weight_morning".

To drop columns whose name contains "morning" at the end:

```
mydata22 <- mydata[,!grepl("morning$",names(mydata))]
print(mydata22)
```

```
##   age sex test_blood test_pressure
## 1  25   1         30           120
## 2  34   1         38           134
## 3  45   2         28           132
```

To Keep columns whose name contains the letter "s":

```
mydata32 <- mydata[,grepl("*s",names(mydata))]
print(mydata32)
```

```
##   sex test_blood test_pressure
## 1   1         30           120
## 2   1         38           134
## 3   2         28           132
```

To drop columns whose name contains the letter "s":

```
mydata33 <- mydata[,!grepl("*s",names(mydata))]
print(mydata33)
```

```
##   age height_morning weight_morning
## 1  25            165             70
## 2  34            170             65
```

```
## 3  45            175             50
```

5.2 Merge Data

5.2.1 Add cases/observations to a dataset

We can join two matrix/dataframe using the rbind() function. Appending two datasets require that both have variables with exactly the same name.

```
#rbind function
a<-matrix(1:10,2,5)
colnames(a)<-paste("col",1:5,sep="")
rownames(a)<-paste("row",1:2,sep="")
print(a)
```

```
##      col1 col2 col3 col4 col5
## row1    1    3    5    7    9
## row2    2    4    6    8   10
```

```
b<-matrix(1:15,3,5)
colnames(b)<-paste("col",1:5,sep="")
rownames(b)<-paste("row",1:3,sep="")
print(b)
```

```
##      col1 col2 col3 col4 col5
## row1    1    4    7   10   13
## row2    2    5    8   11   14
## row3    3    6    9   12   15
```

```
mydata<-rbind(a,b)#the number of columns must be the same for the two matrix
print(mydata)
```

```
##      col1 col2 col3 col4 col5
## row1    1    3    5    7    9
## row2    2    4    6    8   10
## row1    1    4    7   10   13
## row2    2    5    8   11   14
## row3    3    6    9   12   15
```

```
#dataframe is similar to matrix
a<-as.data.frame(a)
b<-as.data.frame(b)
mydata<-rbind(a,b)
print(mydata)
```

```
##       col1 col2 col3 col4 col5
## row1     1    3    5    7    9
## row2     2    4    6    8   10
## row11    1    4    7   10   13
## row21    2    5    8   11   14
## row3     3    6    9   12   15
```

When there is one dataset missing one varible:

```
a<-matrix(1:8,2,4)
colnames(a)<-paste("col",1:4,sep="")
```

```
rownames(a)<-paste("row",1:2,sep="")
a=as.data.frame(a)
print(a)
```

```
##      col1 col2 col3 col4
## row1    1    3    5    7
## row2    2    4    6    8
```

```
b<-matrix(1:15,3,5)
colnames(b)<-paste("col",1:5,sep="")
rownames(b)<-paste("row",1:3,sep="")
b=as.data.frame(b)
print(b)
```

```
##      col1 col2 col3 col4 col5
## row1    1    4    7   10   13
## row2    2    5    8   11   14
## row3    3    6    9   12   15
```

```
mydata<-rbind(a,b)
Returs:Error in rbind(deparse.level, ...) : numbers of columns of arguments do not match
```

There are some possible solutions to this:

Option A: Drop the extra variable from one of the datasets (in this case b)

```
b$col5<-NULL
mydata<-rbind(a,b)
```

Option B: Create the variable with missing values in the incomplete dataset (in this case a)

```
a$col5<-NA
mydata<-rbind(a,b)
```

5.2.2 Merge Data Frames

We can merge two data frames by using the merge() function. Merging two datasets require that both have at least one variable in common (either string or numeric). If string make sure the categories have the same spelling (i.e.country names, etc.).

```
#we import two datasets about diabetes: diabetes_1 and diabetes_2.
mydata1<-read.csv("data/diabetes_1.csv",header=T)
mydata2<-read.csv("data/diabetes_2.csv",header=T)
dim(mydata1)
```

```
## [1] 332   5
```

```
head(mydata1)
```

```
##   glu bp  bmi age type
## 1 148 72 33.6  50  Yes
## 2  85 66 26.6  31   No
## 3  89 66 28.1  21   No
## 4  78 50 31.0  26  Yes
## 5 197 70 30.5  53  Yes
## 6 166 72 25.8  51  Yes
```

```r
dim(mydata2)
```

```
## [1] 200   5
```

```r
head(mydata2)
```

```
##   glu bp  bmi age type
## 1  86 68 30.2  24   No
## 2 195 70 25.1  55  Yes
## 3  77 82 35.8  35   No
## 4 165 76 47.9  26   No
## 5 107 60 26.4  23   No
## 6  97 76 35.6  52  Yes
```

```r
#We then merge the two data sets based on the values of blood pressure("bp") and body mass index("bmi")
mydata <- merge(x = mydata1, y = mydata2,
   by.x = c("bp", "bmi"),
   by.y = c("bp", "bmi")
)
dim(mydata)
```

```
## [1] 17  8
```

```r
head(mydata,n=10)
```

```
##    bp  bmi glu.x age.x type.x glu.y age.y type.y
## 1  60 33.8   117    27     No   125    31     No
## 2  64 29.7    75    33     No   100    21     No
## 3  64 31.2   189    29    Yes   158    24     No
## 4  64 33.2   117    24     No    96    21     No
## 5  66 38.1   115    28     No   114    21     No
## 6  68 38.5   100    26     No   129    43    Yes
## 7  70 27.4   116    21     No   124    36    Yes
## 8  70 33.1    91    22     No   123    40     No
## 9  70 35.4   124    34     No   134    29    Yes
## 10 72 25.6   157    24     No    99    28     No
```

```r
#In the example above, on choosing these two columns for merging, the records where values of these two
```

When there are NAs in the matching variable ('incomparables'):

```r
x <- data.frame(k1 = c(NA,NA,3,4,5), k2 = c(1,NA,NA,4,5), data = 1:5)
y <- data.frame(k1 = c(NA,2,NA,4,5), k2 = c(NA,NA,3,4,5), data = 1:5)
merge(x, y, by = c("k1","k2")) # NA's match
```

```
##   k1 k2 data.x data.y
## 1  4  4      4      4
## 2  5  5      5      5
## 3 NA NA      2      1
```

```r
merge(x, y, by = "k1") # NA's match, so 6 rows
```

```
##   k1 k2.x data.x k2.y data.y
## 1  4    4      4    4      4
## 2  5    5      5    5      5
## 3 NA    1      1   NA      1
## 4 NA    1      1    3      3
## 5 NA   NA      2   NA      1
```

```
## 6 NA   NA      2    3      3
```

```
merge(x, y, by = "k2", incomparables = NA) # 2 rows
```

```
##   k2 k1.x data.x k1.y data.y
## 1  4    4      4    4      4
## 2  5    5      5    5      5
```

5.2.3 Add more variables to a dataset

```
#cbind function
a<-matrix(1:10,2,5)
b<-matrix(20:31,2,6)
mydata<-cbind(a,b)#the number of rows must be the same for the two matrix
print(mydata)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
## [1,]    1    3    5    7    9   20   22   24   26    28    30
## [2,]    2    4    6    8   10   21   23   25   27    29    31
```

5.3 The Reshape package

Reshape package is a comprehensive package to massage data.

`install.packages("reshape")`

In this package, we can "melt" data so that each row is a unique id-variable combination. Then we "cast" the melted data into any shape we would like.

```
mydata<-data.frame(ID=rep(letters[1:3],c(1,2,3)),year=c(65,60,65,60,65,70),x1=c(1,1,2,1,2,3),x2=c(2,3,2
print(mydata)
```

```
##   ID year x1 x2
## 1  a   65  1  2
## 2  b   60  1  3
## 3  b   65  2  2
## 4  c   60  1  5
## 5  c   65  2  4
## 6  c   70  3  6
```

```
#Now we melt the data to organize it, converting all columns other than ID and year into multiple rows.
library(reshape)
mdata<-melt(mydata, id = c("ID","year"))
print(mdata)
```

```
##    ID year variable value
## 1   a   65       x1     1
## 2   b   60       x1     1
## 3   b   65       x1     2
## 4   c   60       x1     1
## 5   c   65       x1     2
## 6   c   70       x1     3
## 7   a   65       x2     2
## 8   b   60       x2     3
## 9   b   65       x2     2
```

```
## 10  c   60       x2     5
## 11  c   65       x2     4
## 12  c   70       x2     6
```

Cast the Molten Data:

```
#casted the melt data
#cast(data, formula, function)
subjmeans <- cast(mdata, ID~variable, mean)
print(subjmeans)
```

```
##   ID  x1  x2
## 1  a 1.0 2.0
## 2  b 1.5 2.5
## 3  c 2.0 5.0
```

```
timemeans <- cast(mdata, year~variable, mean)
print(timemeans)
```

```
##   year   x1   x2
## 1   60 1.00 4.00
## 2   65 1.67 2.67
## 3   70 3.00 6.00
```

aggregate() is also a useful function for this:

```
aggregate(mydata$x1,by=list(mydata$ID),mean)#the mean of x1 by ID
```

```
##   Group.1   x
## 1       a 1.0
## 2       b 1.5
## 3       c 2.0
```

```
aggregate(mydata$x2,by=list(mydata$ID),mean)#the mean of x2 by ID
```

```
##   Group.1   x
## 1       a 2.0
## 2       b 2.5
## 3       c 5.0
```

```
aggregate(mydata$x1,by=list(mydata$year),mean)#the mean of x1 by year
```

```
##   Group.1    x
## 1      60 1.00
## 2      65 1.67
## 3      70 3.00
```

```
aggregate(mydata$x2,by=list(mydata$year),mean)#the mean of x2 by year
```

```
##   Group.1    x
## 1      60 4.00
## 2      65 2.67
## 3      70 6.00
```

```
#When using the aggregate() function, the by variables must be in a list.
```

There is much more that we can do with these functions. Please refer to:

```
help(melt)
help(cast)
help(aggregate)
```

Chapter 6

Base Graphics with R {ch5}

One of the main reasons data analysts turn to R is for its strong graphic capabilities.To see some of the possibilities that R offers, enter:

```
demo(graphics)
```

There are two kinds of graphical functions: **the high-level plotting functions** which create a new graph, and **the low-level plotting functions** which add elements to an existing graph. The graphs are produced with respect to **graphical parameters** which are defined by default and can be modified with the function **par**.

6.1 Graphical functions

6.1.1 (1) High-level graphical functions

Here are some of the high-level graphical functions. For each function, the options may be found with the on-line help in R.

Some of these options are identical for several graphical functions; here are the main ones (with their possible default values):

Table 6.1: High-level graphical functions

names	annotation
plot(x)	plot of the values of x (on the y-axis) ordered on the x-axis
plot(x,y)	bivariate plot of x (on the x-axis) and y (on the y-axis)
pie(x)	pie-chart
boxplot(x)	"box-and-whiskers" plot
stripchart(x)	plot of the values of x on a line (an alternative to boxplot() for small sample sizes)
coplot(x~y|z	bivariate plot of x and y for each value (or interval of values) of z
pairs(x)	if x is a matrix or a data frame, draws all possible bivariate plots between the columns of x
hist(x)	histogram of the frequencies of x
barplot(x)	histogram of the values of x
qqnorm(x)	quantiles of x with respect to the values expected under a normal law
qqplot(x,y)	quantiles of y with respect to the quantiles of x

Table 6.2: Graphical Parameters

names	annotation
add=FALSE	if TRUE superposes the plot on the previous one (if it exists)
axes=TRUE	if FALSE does not draw the axes and the box
type=p	specifies the type of plot, 'p': points, 'l': lines, 'b': points connected by lines, 'o': similar to 'b', but the lin
xlim=, ylim=	specifies the lower and upper limits of the axes, for ex- ample with xlim=c(1, 10) or xlim=range(x)
xlab=, ylab=	annotates the axes, must be variables of mode character
main=	main title, must be a variable of mode character
sub=	sub-title (written in a smaller font)

Table 6.3: Low-level Functions

names	annotation
points(x, y)	add points (the option type= can be used)
lines(x, y)	add lines
text(x, y, labels,...)	add text given by labels at coordinates (x,y)
mtext(text,side=3, line=0,...)	add text given by text in the margin specified by side; line specifies the line fro
segments(x0, y0, x1, y1)	draw lines from points (x0,y0) to points (x1,y1)
arrows(x0, y0,x1, y1, angle= 30, code=2)	draw arrows at points (x0,y0) if code=2, at points (x1,y1) if code=1, or both i
abline(a,b)	draw a line of slope b and intercept a
abline(h=y)	draw a horizontal line at ordinate y
abline(v=x)	draw a vertical line at abcissa x
abline(lm.obj)	draw the regression line given by lm.obj
rect(x1, y1, x2,y2)	draw a rectangle which left, right, bottom, and top limits are x1, x2, y1, and y
polygon(x, y)	draw a polygon linking the points with coordinates given by x and y
legend(x, y,legend)	add the legend at the point (x,y) with the symbols given by legend
title()	add a title and optionally a sub-title
axis(side, vect)	add an axis at the bottom (side=1), on the left (2), at the top (3), or on the r
box()	add a box around the current plot
rug(x)	draw the data x on the x-axis as small vertical lines
locator(n,type='n', ...)	return the coordinates (x,y) after the user has clicked n times on the plot with

6.1.2 (2) Low-level functions

R has a set of graphical functions which affect an already existing graph:

Example 1:

```
x <- rnorm(10)
y <- rnorm(10)
plot(x,y)#The plot( ) function opens a graph window and plots x vs.y
abline(lm(y~x))#This code adds a regression line to this graph.
```

bookdown-demo_files/figure-latex/unnamed-chunk-111-1.pdf

Example 2:

```
x <- rnorm(10)
y <- rnorm(10)
plot(x,y,axes=F,xlim=c(-2,2),ylim=c(-2,2))#do not draw axes
axis(1,at=c(-2,-1.5,-1,-0.5,0,0.5,1,1.5,2))#axis at the bottom
axis(2,at=c(-2,-.15,-1,-0.5,0,0.5,1,1.5,2))#axis on the left
abline(h=0)#draw a horizontal line at y=0
```

bookdown-demo_files/figure-latex/unnamed-chunk-112-1.pdf

Example 3:

Sometimes we may want to overlay the plots in order to compare the results. However, calling plot() multiple times will have the effect of plotting the current graph on the same window replacing the previous one.To overlay the plots, we can create the initial plot and then add additional information to the plot.

```
plot(x,
 main="Overlaying Graphs",
 ylab="",
 type="l",
 col="blue")

lines(y, col="red")

legend("topleft",
      c("x","y"),
      fill=c("blue","red")
)
```

bookdown-demo_files/figure-latex/unnamed-chunk-113-1.pdf

6.2 Graphical parameters

In addition to plotting commands, the presentation of graphics can be improved with graphical parameters. We can customize many features of the graphs (fonts, colors, axes, titles) through graphic options.

6.2.1 6.2.1 Specify within the high-level function

One way to specify graphical parameters is by providing the optionname=value pairs directly to a high level plotting function. In this case, the options are only in effect for that specific graph.

```
# Set a graphical parameter within the plotting function
plot(x, col.axis="blue")
```

bookdown-demo_files/figure-latex/unnamed-chunk-114-1.pdf

But it does not work for all. See the help for a specific high level plotting function (e.g. plot, hist, boxplot) to determine which graphical parameters can be set this way.

6.2.2 Function par()

Another way to change the graphical parameters is with the function par(). If we set parameter values here, the changes will be in effect for the rest of the session or until we change them again.

```
par()              # view current settings
```

```
# Set a graphical parameter using par()
opar <- par()# make a copy of current settings
par(bg="lightyellow", col.axis="blue", mar=c(4, 4, 2.5, 0.25)) #Three parameters are then modified: bg
plot(x, y, xlab="Ten random values", ylab="Ten other values",
xlim=c(-2, 2), ylim=c(-2, 2), pch=22, col="red", bg="yellow",
bty="l", tcl=-.25, las=1, cex=1.5)# create a plot with these new settings
title("Hypothetical Example", font.main=3, adj=1)
```

bookdown-demo_files/figure-latex/unnamed-chunk-115-1.pdf

```
par(opar)          # restore original settings
```

```
## Warning in par(opar): graphical parameter "cin" cannot be set

## Warning in par(opar): graphical parameter "cra" cannot be set

## Warning in par(opar): graphical parameter "csi" cannot be set

## Warning in par(opar): graphical parameter "cxy" cannot be set

## Warning in par(opar): graphical parameter "din" cannot be set

## Warning in par(opar): graphical parameter "page" cannot be set
```

There are 73 graphical parameters, some of them have very similar functions. The exhaustive list of these parameters can be read with **?par**. Now some usual codes are shown below.

(1)Text and Symbol Size

```
cex # number indicating the amount by which plotting text and symbols should be scaled relative to the
cex.axis #magnification of axis annotation relative to cex
cex.lab #magnification of x and y labels relative to cex
cex.main    #magnification of titles relative to cex
cex.sub #magnification of subtitles relative to cex
```

(2) Plotting symbols

Use the pch= option to control the type of symbols.

```
x <- rep(1,25)
plot(x, pch = 1:25, axes = F, xlab = "", ylab = "")
text(1:25,.95,labels = 1:25)
```

bookdown-demo_files/figure-latex/unnamed-chunk-116-1.pdf

(3) Lines

We can change lines using the following options. This is particularly useful for reference lines, axes, and fit lines.

```
lty #controls the type of lines, can be an integer (1: solid, 2: dashed, 3: dotted, 4: dotdash, 5: longd
lwd #a numeric which controls the width of lines
```

(4) Fonts

It is also possible to set font size and style

```
font #Integer specifying font to use for text. 1=plain, 2=bold, 3=italic, 4=bold italic, 5=symbol
font.axis #font for axis annotation
font.lab    #font for x and y labels
font.main   #font for titles
font.sub    #font for subtitles
ps  #font point size (roughly 1/72 inch) text size=ps*cex
family  #font family for drawing text. Standard values are "serif", "sans", "mono", "symbol". Mapping is
```

(5) Colors

We can visually improve our plots by coloring them. This is generally done with the col graphical parameter.To set the colors:

```
col #controls the colour of symbols
col.axis #color of axis
col.lab #color of x and y labels
col.main    #color of titles
col.sub #color of subtitles
```

657 colors are available in R. To see their names, just type :

```
colors()
```

This returns a vector of all the color names in alphabetical order with the first element being white. We can color the plot by indexing this vector.

Then it is easy to call one of these colors for a plot:

```
plot(c(1,2) , c(1,1) , axes=F , col=c("blue" , "Darkgreen") , pch=20 , cex=14 , xlim=c(0,3),xlab="",ylab
```

bookdown-demo_files/figure-latex/unnamed-chunk-117-1.pdf

If we need severals colors in the same palette, we can use one of the folowing functions :

```
rainbow()
heat.colors()
terrain.colors()
topo.colors()
cm.colors()
```

For example:

```
heat.colors(5 , alpha=0.4)#we have to indicate the number of different colors we want, and the transpar
```

```
## [1] "#FF000066" "#FF550066" "#FFAA0066" "#FFFF0066" "#FFFF8066"
```

```
n <- 20
y <- -sin(3*pi*((1:n)-1/2)/n)
plot(y, axes = FALSE, frame.plot = TRUE, xlab = "", ylab = "", pch = 21, cex = 30,
     bg = rainbow(n, alpha=0.4))
```

bookdown-demo_files/figure-latex/unnamed-chunk-119-1.pdf

Use the rgb() function

The function rgb() allows us to specify red, green and blue component with a number between 0 and 1.

```
rgb(red, green, blue, alpha) # The arguments indicate: quantity of red (between 0 and 1), quantity of gr
```

For example:

```
plot(c(1,2) , c(1,1) , axes=F , col=c(rgb(1,0.5,0.1,0.5),rgb(0.5,1,1,0.5)) , pch=20 , cex=14 , xlim=c(0
```

bookdown-demo_files/figure-latex/unnamed-chunk-120-1.pdf

Some examples for the graphic parameters:

```
# Create data:
x <- rnorm(10)
y <- rnorm(10)
#plot
plot(x, y, xlab="Ten random values", ylab="Ten other values", ##xlab and ylab change the axis labels whi
     xlim=c(-2, 2), ylim=c(-2, 2), #xlim and ylim allow us to define the limits on both axes
     pch=22,      #pch is used  as an option: pch=22 specifies a square
     col="red",   #controls the colour of symbols
     bg="yellow", #specifies the colour of the background
     bty="l",     #controls the type of box drawn around the plot
main="Hypothetical Example",# a title is added
las=1,   #an integer which controls the orientation of the axis labels (0: parallel to the axes, 1: hor
cex=1.5  #a value controlling the size of texts and symbols
)
```

bookdown-demo_files/figure-latex/unnamed-chunk-121-1.pdf

Some handy plotting parameters

```
attach(mtcars)
```

```
## The following object is masked from package:ggplot2:
##
##     mpg
```

```
plot(disp,mpg,
     main = "MPG vs. Displacement",    # Add a title
     type = "p",
     col = "grey",                     # Change the color of the points
     pch = 16,                         # Change the plotting symbol  see help(points)
     cex = 1,                          # Change size of plotting symbol
     xlab = "Displacement (cu. in)",   # Add a label on the x-axis
     ylab = "Miles per Gallon",        # Add a label on the y-axis
     bty = "n",                        # Remove the box around the plot
     #asp = 1,                         # Change the y/x aspect ratio see help(plot)
     font.axis = 1,                    # Change axis font to bold italic
     col.axis = "black",               # Set the color of the axis
     xlim = c(85,500),                 # Set limits on x axis
     ylim = c(10,35),                  # Set limits on y axis
     las=1)                            # Make axis labels parallel to x-axis

abline(lm(mpg ~ disp),                 # Add regression line y ~ x
       col="red",                      # regression line color
       lty = 2,                        # use dashed line
       lwd = 2)                        # Set thickness of the line

lines(lowess(mpg ~ disp),              # Add lowess line y ~ x
      col="dark blue",                 # Set color of lowess line
      lwd= 2)                          # Set thickness of the lowess line

leg.txt <- c("red = lm", "blue = lowess") # Text for legend
legend(list(x = 180,y = 35),           # Set location of the legend
       legend = leg.txt,               # Specify text
       col = c("red","dark blue"),     # Set colors for legend
       lty = c(2,1),                   # Set type of lines in legend
       merge = TRUE)                   # merge points and lines
```

bookdown-demo_files/figure-latex/unnamed-chunk-122-1.pdf

```
detach(mtcars)
```

6.3 Multiple plots on the one page

There are two ways for combining a graphic:one with function layout() and the other with function par().

6.3.1 Function layout()

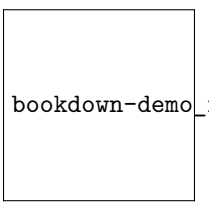
The function layout partitions the active graphic window in several parts where the graphs will be displayed successively. Its main argument is a matrix with integer numbers indicating the numbers of the "sub-windows". For example, to divide the device into four equal parts:layout(matrix(1:4, 2, 2)). We can divide the device into as many rows and columns as we want, and specify the column-widths and the row-heights.

```
#create the data
set.seed(2017)
a<-seq(129,1)+4*runif(129,0.4)
b<-seq(1,129)^2+runif(129,0.98)
```

Let's say we want to divide the device in 3 parts : a big graph on top and 2 smalls below. we give a matrix of 2 columns and 2 rows, and attribute each part for a graph. So the 2 first parts for graph number 1, and the 2 other for graph 2 and 3.

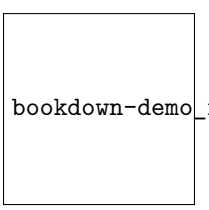
Calling layout.show() results in the layout being displayed for the reference. This allows us to experiment with the parameter options and immediately get a sense of how a given layout will be rendered.

```
nf<-layout(matrix(c(1,1,2,3), 2, 2, byrow = TRUE))
layout.show(nf)
```



bookdown-demo_files/figure-latex/unnamed-chunk-124-1.pdf

```
#add plots on each of the screen
hist(a , breaks=30 , border=F , col=rgb(0.1,0.8,0.3,0.5) , xlab="distribution of a" , main="")
boxplot(a , xlab="a" , col=rgb(0.8,0.8,0.3,0.5) , las=2)
boxplot(b , xlab="b" , col=rgb(0.4,0.2,0.3,0.5) , las=2)
```



bookdown-demo_files/figure-latex/unnamed-chunk-124-2.pdf

By default, layout() partitions the device with regular heights and widths: this can be modified with the options widths and heights. These dimensions are given relatively.

If we want to custom the size of each screen:

```
# Set the layout
nf<-layout(matrix(c(1,1,2,3),2,2,byrow=TRUE), widths=c(2.5,1.5), heights=c(2,2),TRUE)
layout.show(nf)
```

bookdown-demo_files/figure-latex/unnamed-chunk-125-1.pdf

```
#Add the plots
hist(a , breaks=30 , border=F , col=rgb(0.1,0.8,0.3,0.5) , xlab="distribution of a" , main="")
boxplot(a , xlab="a" , col=rgb(0.8,0.8,0.3,0.5) , las=2)
boxplot(b , xlab="b" , col=rgb(0.4,0.2,0.3,0.5) , las=2)
```

bookdown-demo_files/figure-latex/unnamed-chunk-125-2.pdf

6.3.2 Function par()

We can put multiple graphs in a single plot by setting some graphical parameters with the help of par() function.

The parameter mfrow can be used to configure the graphics sheet so that subsequent plots appear row by row, one after the other in a rectangular layout, on the one page.

```
#create the data
month.temp=c(10,12,5,11,7,10,9,6,4,7,9,10)
names(month.temp)=letters[1:12]
print(month.temp)
```

```
##  a  b  c  d  e  f  g  h  i  j  k  l
## 10 12  5 11  7 10  9  6  4  7  9 10
```

```
#plot
par(mfrow=c(1,2))     # mfrow takes in a vector of form c(m, n) which divides the given plot into m*n ar
barplot(month.temp, main="Barplot")
pie(month.temp, main="Piechart", radius=1)
```

bookdown-demo_files/figure-latex/unnamed-chunk-126-1.pdf

This same phenomenon can be achieved with the graphical parameter mfcol.The only difference between the two is that, mfrow fills in the subplot region row wise while mfcol fills it column wise.

```
Temperature <- airquality$Temp
Ozone <- airquality$Ozone
par(mfrow=c(2,2))
hist(Temperature)
boxplot(Temperature, horizontal=TRUE)
hist(Ozone)
boxplot(Ozone, horizontal=TRUE)
```

bookdown-demo_files/figure-latex/unnamed-chunk-127-1.pdf

Same plot with the change par(mfcol = c(2, 2)) would look as follows. Note that only the ordering of the subplot is different.

```
Temperature <- airquality$Temp
Ozone <- airquality$Ozone
par(mfcol=c(2,2))
hist(Temperature)
boxplot(Temperature, horizontal=TRUE)
hist(Ozone)
boxplot(Ozone, horizontal=TRUE)
```

bookdown-demo_files/figure-latex/unnamed-chunk-128-1.pdf

We can control the location of a figure more precisely with graphical parameter fig(). We need to provide the coordinates in a normalized form as c(x1, x2, y1, y2). For example, the whole plot area would be c(0, 1, 0, 1) with (x1, y1) = (0, 0) being the lower-left corner and (x2, y2) = (1, 1) being the upper-right corner.

```
# make labels and margins smaller
par(cex=0.7, mai=c(0.1,0.1,0.2,0.1))#mai for the sizes of the margins around the plotting region

Temperature <- airquality$Temp

# define area for the histogram
par(fig=c(0.1,0.7,0.3,0.9))
hist(Temperature)

# define area for the boxplot
par(fig=c(0.8,1,0,1), new=TRUE)
boxplot(Temperature)

# define area for the stripchart
par(fig=c(0.1,0.67,0.1,0.25), new=TRUE)
stripchart(Temperature, method="jitter")
```

bookdown-demo_files/figure-latex/unnamed-chunk-129-1.pdf

Table 6.4: Guidance for the formats

functions	format	annotations
JPG	jpeg	Bitmap image. Have a fixed resolution and are pixelated when zoomed enough
PNG	png	Bitmap image. Have a fixed resolution and are pixelated when zoomed enough
BMP	bmp	Bitmap image. Have a fixed resolution and are pixelated when zoomed enough
TIFF	tiff	Bitmap image. Have a fixed resolution and are pixelated when zoomed enough
PDF	pdf	Vector images.Easily resizable. Zooming on the image will not compromise its quality
Postscript	postscript	Vector images.Easily resizable. Zooming on the image will not compromise its quality

6.4 Save a plot

6.4.1 Save plots by menu click

All the graphs (bar graph, pie chart, histogram, etc.) we plot in R are displayed on the screen by default.RStudio has a nice feature in that it saves all of the plots in the plotting pane (we introducted in the first lecture). We can save the graph in a variety of formats from the menu: Export–> Save as. It is no problem if we just produce the plots one after one and save each one individually because it keeps all of the plots in the pane. However,imagine that we are running a loop and have 1000 plots inside the loop. This manual-saving method becomes impractical quickly.Therefore, we need automatically save plots to a folder without spending too much time.

6.4.2 Save plots using functions

We can save the graph via code using specific functions.Please note that we need to call the function dev.off() after all the plotting, to save the file and return output to the terminal.

The first step in saving plots is to decide the output format that we want to use. Here lists some of the available formats, along with guidance as to when they may be useful.

6.4.3 (1) A common method

The following methods work on any computer with R, regardless of operating system or the way that we are connecting.

Save as Jpeg image

```
jpeg(file="saving_plot1.jpeg")
hist(Temperature, col="gray")
dev.off()
```

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

```
#We will not actually see the plot. We can find this plot in the current directory. We can also specify
#The resolution of the image by default will be 480x480 pixel.
```

Save as png image

We can specify the resolution we want with arguments width and height.

```
png(file="saving_plot2.png",width=600, height=350)
hist(Temperature, col="gold")
dev.off()
```

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

```
#We are saving a png file with resolution 600x350
```

Save as bmp image

We can specify the size of our image in inch, cm or mm with the argument units and specify ppi with res.

```
bmp(file="saving_plot3.bmp",
   width=6, height=4, units="in", res=100)
hist(Temperature, col="steelblue")
dev.off()
```

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

```
#We are saving a bmp file of size 6x4 inch and 100 ppi.
```

Save as tiff image

```
tiff(file="saving_plot4.tiff",
   width=6, height=4, units="in", res=100)
hist(Temperature, col="steelblue")
dev.off()
```

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

Save as pdf file

```
pdf(file="saving_plot5.pdf")
hist(Temperature, col="violet")
dev.off()
```

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

Save as postscript file

```
postscript(file="saving_plot6.ps")
hist(Temperature, col="violet")
dev.off()
```

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

6.4.4 (2)Another option

R also provides the dev.copy command, to copy the contents of the graph window to a file without having to re-enter the commands.

```
hist(Temperature, col="tan")
```

bookdown-demo_files/figure-latex/unnamed-chunk-137-1.pdf

```
dev.copy(png,'myplot.png')#to create a png file called myplot.png from a graph that is displayed by R
```

```
## quartz_off_screen
##                 3
```

```
dev.off()
```

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

```
#For most plots, things will be fine, but sometimes translating what was on the screen into a different
```

6.4.5 (3) Saving plots in a loop

We can see the beauty of the automatically saving through the following loop:

```
names <- letters[1:26] ## Gives a sequence of the letters of the alphabet
set.seed(2017)
beta1 <- rnorm(26, 5, 2) ## A vector of slopes (one for each letter)
beta0 <-10 ## A common intercept

for(i in 1:26){
 x <- rnorm(500, 105, 10)
 y <- beta0 + beta1[i]*x + 15*rnorm(500)

 myfile <- paste("myplot_", names[i], ".jpg", sep = "")

 jpeg(file=myfile)
    mytitle = paste("my title is", names[i])
    plot(x,y, main = mytitle)
 dev.off()
}
```

Chapter 7

Descriptive statistics with R: Part I-Quantitative variables {ch6}

Before going into the actual statistical modelling and analysis of a data set, it is often useful to make some simple characterizations of the data.

The data set diabetes_1 will be used as the example, which contains a population of women who were at least 21 years old, of Pima Indian heritage and living near Phoenix, Arizona, was tested for diabetes according to World Health Organization criteria. There are four quantitative variables: glu, bp, bmi and age; one categorical variable: type.

glu: plasma glucose concentration in an oral glucose tolerance test bp:diastolic blood pressure (mm Hg) bmi:body mass index (weight in kg/(height in m)^2) age: age in years type:Yes or No, for diabetic according to WHO criteria.

```
#read in the dataset
mydata<-read.csv("data/diabetes_1.csv",header=T)
dim(mydata)
```

```
## [1] 332   5
```

```
head(mydata)
```

```
##   glu bp  bmi age type
## 1 148 72 33.6  50  Yes
## 2  85 66 26.6  31   No
## 3  89 66 28.1  21   No
## 4  78 50 31.0  26  Yes
## 5 197 70 30.5  53  Yes
## 6 166 72 25.8  51  Yes
```

```
#create another variable age_cat which is a categorical variable of age#
mydata$age_cat[mydata$age<=30]<-1
mydata$age_cat[mydata$age>30 & mydata$age<=50]<-2
mydata$age_cat[mydata$age>50 ]<-3
mydata$age_cat=factor(mydata$age_cat,levels=c(1,2,3),labels=c("<=30","30-50",">50"))
write.csv(mydata,file="diabetes_new.csv",row.names=F)#prevent rownames to be written.
```

Now the dataset mydata contains four quantitative variables: glu, bp, bmi and age; two categorical variables: type and age_cat.

7.1 For the whole dataset

```
# mean,median,25th and 75th quartiles,min,max
summary(mydata)
```

```
##       glu            bp             bmi            age        type
##  Min.   : 65   Min.   : 24.0   Min.   :19.4   Min.   :21.0   No :223
##  1st Qu.: 96   1st Qu.: 64.0   1st Qu.:28.2   1st Qu.:23.0   Yes:109
##  Median :112   Median : 72.0   Median :32.9   Median :27.0
##  Mean   :119   Mean   : 71.7   Mean   :33.2   Mean   :31.3
##  3rd Qu.:136   3rd Qu.: 80.0   3rd Qu.:37.2   3rd Qu.:37.0
##  Max.   :197   Max.   :110.0   Max.   :67.1   Max.   :81.0
##   age_cat
##  <=30 :205
##  30-50:105
##  >50  : 22
##
##
##
```

That returns some basic calculations for each column. For continous variable, we'll see the minimum and maximum values along with median, mean, 1st quartile and 3rd quartile. For categorical variable, we will see the counts in each category group.

There are also numerous R functions designed to provide a range of descriptive statistics. For example:

```
install.packages("psych")
```

```
library(psych)
describe(mydata)
```

```
##          vars   n   mean    sd median trimmed   mad  min   max range  skew
## glu         1 332 119.26 30.50  112.0  116.58 26.69 65.0 197.0 132.0  0.70
## bp          2 332  71.65 12.80   72.0   71.79 11.86 24.0 110.0  86.0 -0.08
## bmi         3 332  33.24  7.28   32.9   32.79  6.75 19.4  67.1  47.7  0.80
## age         4 332  31.32 10.64   27.0   29.65  7.41 21.0  81.0  60.0  1.38
## type*       5 332   1.33  0.47    1.0    1.29  0.00  1.0   2.0   1.0  0.73
## age_cat*    6 332   1.45  0.62    1.0    1.35  0.00  1.0   3.0   2.0  1.04
##          kurtosis   se
## glu         -0.31 1.67
## bp           0.81 0.70
## bmi          1.46 0.40
## age          1.76 0.58
## type*       -1.47 0.03
## age_cat*     0.02 0.03
```

This returns several more statistics from the data including standard deviation, "mad" (mean absolute deviation), skew (measuring whether or not the data distribution is symmetrical) and kurtosis (whether the data have a sharp or flatter peak near its mean).

```
install.packages("pastecs")
```

```
library(pastecs)
stat.desc(mydata) #Compute a table giving various descriptive statistics about the series
```

```
##                    glu       bp      bmi      age type age_cat
## nbr.val      3.32e+02 3.32e+02 3.32e+02 3.32e+02   NA      NA
```

```
## nbr.null     0.00e+00 0.00e+00 0.00e+00 0.00e+00   NA      NA
## nbr.na       0.00e+00 0.00e+00 0.00e+00 0.00e+00   NA      NA
## min          6.50e+01 2.40e+01 1.94e+01 2.10e+01   NA      NA
## max          1.97e+02 1.10e+02 6.71e+01 8.10e+01   NA      NA
## range        1.32e+02 8.60e+01 4.77e+01 6.00e+01   NA      NA
## sum          3.96e+04 2.38e+04 1.10e+04 1.04e+04   NA      NA
## median       1.12e+02 7.20e+01 3.29e+01 2.70e+01   NA      NA
## mean         1.19e+02 7.17e+01 3.32e+01 3.13e+01   NA      NA
## SE.mean      1.67e+00 7.02e-01 4.00e-01 5.84e-01   NA      NA
## CI.mean.0.95 3.29e+00 1.38e+00 7.86e-01 1.15e+00   NA      NA
## var          9.30e+02 1.64e+02 5.30e+01 1.13e+02   NA      NA
## std.dev      3.05e+01 1.28e+01 7.28e+00 1.06e+01   NA      NA
## coef.var     2.56e-01 1.79e-01 2.19e-01 3.40e-01   NA      NA
```

7.2 For a single quantitative variable

7.2.1 Numerical representation

As we described in previous lectures, R provides a wide range of functions for obtaining summary statistics, i.e.,mean, sd, var, min, max, median, range, and quantile.

```r
mean(mydata$age)
```

```
## [1] 31.3
```

```r
median(mydata$age)
```

```
## [1] 27
```

```r
sd(mydata$age)
```

```
## [1] 10.6
```

```r
range(mydata$age)
```

```
## [1] 21 81
```

```r
quantile(mydata$age)
```

```
##   0%  25%  50%  75% 100%
##   21   23   27   37   81
```

```r
# min,25th quartile, median,75th quartile,max
fivenum(mydata$age)
```

```
## [1] 21 23 27 37 81
```

We can apply trim parameter to exclude some values if they are outliers. When trim parameter is supplied, the values of the variable get sorted and then the required numbers of observations are dropped from calculating the mean.

```r
mean(mydata$age,trim=0.3)#The age variable will first get sorted and 3 values from each end will be dro
```

```
## [1] 27.8
```

```r
#for the above functions, they only work for the vectors, so we have to select a specified column.
```

To combine the results into one summary table:

```
sumtab<-rbind(mean(mydata$age),median(mydata$age),sd(mydata$age))
rownames(sumtab)<-c("mean","median","sd")
sumtab
```

```
##        [,1]
## mean   31.3
## median 27.0
## sd     10.6
```

7.2.2 Graphical representation

7.2.2.1 (1) Histograms

We can get a reasonable impression of the shape of a distribution by drawing a histogram, which, is a count of how many observations fall within specified divisions of the x-axis.

```
hist(mydata$age, col="skyblue4")
```

bookdown-demo_files/figure-latex/unnamed-chunk-147-1.pdf

```
#The column height is the raw number in each interval so that we can see how many observations have gon
```

If we want to get the density plot, where the area of a column is proportional to the number, we can set freq=F. This is helpful if we want to compare with other populations.

```
hist(mydata$age, freq=F,col="skyblue4")
```

bookdown-demo_files/figure-latex/unnamed-chunk-148-1.pdf

```
#The y-axis is in density units , so that the total area of the histogram will be 1.
```

If we want to obtain a count in age groups 20-30,30-40,40-50,50-60,60-70,70-81:

```
brk<-c(20,30,40,50,60,70,81)
hist(mydata$age,breaks=brk,freq=F,col="skyblue4")
```

bookdown-demo_files/figure-latex/unnamed-chunk-149-1.pdf

To improve it:

```
hist(mydata$age,breaks=brk,col="skyblue4",freq=F,xlab="Age",main="Age Distribution")
```

bookdown-demo_files/figure-latex/unnamed-chunk-150-1.pdf

Histograms can be a poor method for determining the shape of a distribution because it is so strongly affected by the number of bins used. We may want to add estimated smooth density curves to the plot using the density() function.

```
hist(mydata$age,breaks=brk,col="skyblue4",freq=F,xlab="Age",ylim=c(0,0.065),main="Age Distribution")
d <- density(mydata$age)
lines(d,col=2,lwd=2)
```

bookdown-demo_files/figure-latex/unnamed-chunk-151-1.pdf

We may want a filled density plot alone:

```
plot(d, main="Density plot of age")
polygon(d, col="red", border="blue")
```

bookdown-demo_files/figure-latex/unnamed-chunk-152-1.pdf

We may also want to compare the observed density with a theoretical density e.g. a normal distribution. We can simply add a corresponding density curve using the lines() and dnorm().

```
hist(mydata$age,breaks=brk,col="skyblue4",freq=F,xlab="Age",ylim=c(0,0.065),xlim=c(10,100),main="Age Di
lines(d,col=2,lwd=2)
lines(dnorm(1:100,mean(mydata$age),sd=sd(mydata$age)),lwd=2,lty=2,col=3)
legend("topright",lty=c(1,2),col=c(2,3),lwd=2,legend=c("estimated density","expected density under Norma
```

bookdown-demo_files/figure-latex/unnamed-chunk-153-1.pdf

We can improve the legend by changing some parameters:

```
hist(mydata$age,breaks=brk,col="skyblue4",freq=F,xlab="Age",ylim=c(0,0.065),xlim=c(10,100),main="Age Di
lines(d,col=2,lwd=2)
lines(dnorm(1:100,mean(mydata$age),sd=sd(mydata$age)),lwd=2,lty=2,col=3)
legend(60,0.04,lty=c(1,2),col=c(2,3),lwd=2,legend=c("estimated density","expected density under \n Norma
```

bookdown-demo_files/figure-latex/unnamed-chunk-154-1.pdf

We may want to display a single value for each individual. In this case, we can use rug() function, which adds vertical bars showing the distribution of values of x along the x-axis of the current plot. It is particularly useful for showing the actual values along the side of a boxplot.

```
hist(mydata$age,breaks=brk,col="skyblue4",freq=F,xlab="Age",ylim=c(0,0.065),xlim=c(20,100),main="Age Di
rug(mydata$age,col=2)
```

bookdown-demo_files/figure-latex/unnamed-chunk-155-1.pdf

7.2.2.2 (2) Boxplot

The box plot of an observation variable is a graphical representation based on its quartiles, as well as its smallest and largest values. It attempts to provide a visual shape of the data distribution.

```
boxplot(mydata$age,ylab="age")
```

bookdown-demo_files/figure-latex/unnamed-chunk-156-1.pdf

```
#The box in the middle indicates "hinges" (nearly quartiles) and median. The lines show the largest or
boxplot(mydata$age,xlab="age",horizontal=T)
```

bookdown-demo_files/figure-latex/unnamed-chunk-156-2.pdf

We can add means to the plot:

```
boxplot(mydata$age,ylab="age")
points(1,mean(mydata$age),pch="x",cex=1.6, col=2)
```

bookdown-demo_files/figure-latex/unnamed-chunk-157-1.pdf

7.2.2.3 (3) Empirical cumulative distribution

The empirical cumulative distribution function (ECDF) is defined as the fraction of data smaller than or equal to x. That is, if x is the kth smallest observation, then the proportion k/n of the data is smaller than or equal to x .

The ECDF provides an alternative visualisation of distribution and we can see whether data is normally distributed.

```
n <- length(mydata$age)
plot(sort(mydata$age),(1:n)/n,type="s",ylim=c(0,1),xlab="age",ylab="cumulative distribution function",ma
```

bookdown-demo_files/figure-latex/unnamed-chunk-158-1.pdf

```
#another way for plotting ECDF
ecdf.age<-ecdf(mydata$age)
plot(ecdf.age,xlab="age",ylab="cumulative distribution function",main="Empirical cumulative distributio
```

bookdown-demo_files/figure-latex/unnamed-chunk-158-2.pdf

```
#This is also more precise regarding the mathematical definition of the step function.
```

7.2.2.4 (4) Q-Q plots

T see whether data can be assumed normally distributed, Q-Q plot is another way except the ECDF. In Q-Q plot, we are plotting the kth smallest observation against the expected value of the kth smallest observation out of n in a standard normal distribution. The point is that in this way we would expect to obtain a straight line if data come from a normal distribution with any mean and standard deviation.

```
qqnorm(mydata$age,col="blue")
qqline(mydata$age,col="red",lwd=2)
```

bookdown-demo_files/figure-latex/unnamed-chunk-159-1.pdf

```
#the observed values are now drawn along the y-axis.
```

7.3 For a quantitative variable by groups

7.3.1 Numerical representation

When dealing with grouped data, we will often want to have various summary statistics computed within groups; To this end, we can use tapply().

```
tapply(mydata$glu,mydata$age_cat,mean)#The tapply call takes the glu variable, splits it according to a
```

```
##  <=30 30-50   >50
##   115   123   144
```

```
xbar <- tapply(mydata$glu,mydata$age_cat, mean)
s <- tapply(mydata$glu,mydata$age_cat, sd)
n <- tapply(mydata$glu,mydata$age_cat, length)
cbind(mean=xbar,std.dev=s,n=n)
```

```
##       mean std.dev   n
## <=30   115    28.0 205
## 30-50  123    31.7 105
## >50    144    33.6  22
```

Function aggregate() is very similar to tapply() except that aggregate() works on an entire data frame and presents its results as a data frame.

```
#calculate the mean for the whole dataset mydata.
aggregate(mydata,by=list(mydata$age_cat),mean)#Notice that the grouping argument in this case must be a
```

```
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA

## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA

## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA

## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA

## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA

## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
```

```
##   Group.1 glu   bp  bmi  age type age_cat
## 1    <=30 115 68.8 33.0 24.4   NA      NA
## 2   30-50 123 75.3 33.7 39.2   NA      NA
## 3     >50 144 80.5 33.0 58.0   NA      NA
```

```
#Note that it will return NA if argument is not numeric or logical
```

The function by() can also be used for grouped data.The by() function is similar to aggregate except that it only allows functions that are applicable to entire data frames.

```
by(data, INDICES, FUN, ..., simplify = TRUE)#data is the data frame we wish to analyse, INDICES is the
```

Example:

```r
by(mydata,mydata$age_cat, summary)
```

```
## mydata$age_cat: <=30
##       glu            bp             bmi            age        type
##  Min.   : 68   Min.   : 24.0   Min.   :19.4   Min.   :21.0   No :160
##  1st Qu.: 94   1st Qu.: 62.0   1st Qu.:27.6   1st Qu.:22.0   Yes: 45
##  Median :108   Median : 68.0   Median :32.5   Median :24.0
##  Mean   :115   Mean   : 68.8   Mean   :33.0   Mean   :24.4
##  3rd Qu.:128   3rd Qu.: 78.0   3rd Qu.:37.2   3rd Qu.:26.0
##  Max.   :196   Max.   :110.0   Max.   :67.1   Max.   :30.0
##   age_cat
##  <=30 :205
##  30-50:  0
##  >50  :  0
##
##
##
## ------------------------------------------------------
## mydata$age_cat: 30-50
##       glu            bp             bmi            age        type
##  Min.   : 65   Min.   : 30.0   Min.   :20.8   Min.   :31.0   No :53
##  1st Qu.: 99   1st Qu.: 70.0   1st Qu.:29.6   1st Qu.:35.0   Yes:52
##  Median :118   Median : 74.0   Median :33.3   Median :38.0
##  Mean   :123   Mean   : 75.3   Mean   :33.7   Mean   :39.2
##  3rd Qu.:148   3rd Qu.: 84.0   3rd Qu.:36.7   3rd Qu.:43.0
##  Max.   :197   Max.   :106.0   Max.   :50.0   Max.   :50.0
##   age_cat
##  <=30 :  0
##  30-50:105
##  >50  :  0
##
##
##
## ------------------------------------------------------
## mydata$age_cat: >50
##       glu            bp             bmi            age        type
##  Min.   : 90   Min.   : 60.0   Min.   :19.6   Min.   :51.0   No :10
##  1st Qu.:116   1st Qu.: 74.0   1st Qu.:28.2   1st Qu.:53.0   Yes:12
##  Median :144   Median : 76.0   Median :33.1   Median :56.0
##  Mean   :144   Mean   : 80.5   Mean   :33.0   Mean   :58.0
##  3rd Qu.:172   3rd Qu.: 85.8   3rd Qu.:36.8   3rd Qu.:60.8
##  Max.   :197   Max.   :110.0   Max.   :46.5   Max.   :81.0
##   age_cat
##  <=30 : 0
##  30-50: 0
##  >50  :22
##
##
##
```

The result of the call to by is actually a list of objects that has been wrapped as an object of class "by" and printed using a print method for that class. Therefore we can access the result for each subgroup using standard list indexing.

```
result<-by(mydata,mydata$age_cat, summary)
result[1]
```

```
## $`<=30`
##       glu            bp             bmi            age        type
##  Min.   : 68   Min.   : 24.0   Min.   :19.4   Min.   :21.0   No :160
##  1st Qu.: 94   1st Qu.: 62.0   1st Qu.:27.6   1st Qu.:22.0   Yes: 45
##  Median :108   Median : 68.0   Median :32.5   Median :24.0
##  Mean   :115   Mean   : 68.8   Mean   :33.0   Mean   :24.4
##  3rd Qu.:128   3rd Qu.: 78.0   3rd Qu.:37.2   3rd Qu.:26.0
##  Max.   :196   Max.   :110.0   Max.   :67.1   Max.   :30.0
##   age_cat
##  <=30 :205
##  30-50:  0
##  >50  :  0
##
##
##
```

Note that the default R functions mean(), sd(), median() and quantile() are not applicable to entire data frames (only to individual vectors or variables). Therefore they cannot be combined with the by() function.

We can also choose to use:

```
summary(mydata[mydata$age_cat==levels(mydata$age_cat)[1],])
```

```
##       glu            bp             bmi            age        type
##  Min.   : 68   Min.   : 24.0   Min.   :19.4   Min.   :21.0   No :160
##  1st Qu.: 94   1st Qu.: 62.0   1st Qu.:27.6   1st Qu.:22.0   Yes: 45
##  Median :108   Median : 68.0   Median :32.5   Median :24.0
##  Mean   :115   Mean   : 68.8   Mean   :33.0   Mean   :24.4
##  3rd Qu.:128   3rd Qu.: 78.0   3rd Qu.:37.2   3rd Qu.:26.0
##  Max.   :196   Max.   :110.0   Max.   :67.1   Max.   :30.0
##   age_cat
##  <=30 :205
##  30-50:  0
##  >50  :  0
##
##
##
```

```
summary(mydata[mydata$age_cat==levels(mydata$age_cat)[2],])
```

```
##       glu            bp             bmi            age        type
##  Min.   : 65   Min.   : 30.0   Min.   :20.8   Min.   :31.0   No :53
##  1st Qu.: 99   1st Qu.: 70.0   1st Qu.:29.6   1st Qu.:35.0   Yes:52
##  Median :118   Median : 74.0   Median :33.3   Median :38.0
##  Mean   :123   Mean   : 75.3   Mean   :33.7   Mean   :39.2
##  3rd Qu.:148   3rd Qu.: 84.0   3rd Qu.:36.7   3rd Qu.:43.0
##  Max.   :197   Max.   :106.0   Max.   :50.0   Max.   :50.0
##   age_cat
##  <=30 :  0
##  30-50:105
##  >50  :  0
##
##
```

```
##
```

```r
summary(mydata[mydata$age_cat==levels(mydata$age_cat)[3],])
```

```
##       glu            bp             bmi            age        type
##  Min.   : 90   Min.   : 60.0   Min.   :19.6   Min.   :51.0   No :10
##  1st Qu.:116   1st Qu.: 74.0   1st Qu.:28.2   1st Qu.:53.0   Yes:12
##  Median :144   Median : 76.0   Median :33.1   Median :56.0
##  Mean   :144   Mean   : 80.5   Mean   :33.0   Mean   :58.0
##  3rd Qu.:172   3rd Qu.: 85.8   3rd Qu.:36.8   3rd Qu.:60.8
##  Max.   :197   Max.   :110.0   Max.   :46.5   Max.   :81.0
##   age_cat
##  <=30 : 0
##  30-50: 0
##  >50  :22
##
##
##
```

```r
#or
describeBy(mydata, mydata$age_cat)
```

```
##
##  Descriptive statistics by group
## group: <=30
##          vars   n   mean    sd median trimmed   mad  min   max range  skew
## glu         1 205 114.69 28.03  108.0  111.68 25.20 68.0 196.0 128.0  0.88
## bp          2 205  68.82 12.35   68.0   68.88 11.86 24.0 110.0  86.0 -0.06
## bmi         3 205  33.01  7.79   32.5   32.48  7.26 19.4  67.1  47.7  0.93
## age         4 205  24.43  2.65   24.0   24.25  2.97 21.0  30.0   9.0  0.46
## type*       5 205   1.22  0.41    1.0    1.15  0.00  1.0   2.0   1.0  1.35
## age_cat*    6 205   1.00  0.00    1.0    1.00  0.00  1.0   1.0   0.0   NaN
##          kurtosis   se
## glu          0.18 1.96
## bp           0.68 0.86
## bmi          1.80 0.54
## age         -0.89 0.19
## type*       -0.19 0.03
## age_cat*      NaN 0.00
## ------------------------------------------------------------
## group: 30-50
##          vars   n  mean    sd median trimmed   mad  min max range  skew
## glu         1 105 123.0 31.69  118.0  121.20 31.13 65.0 197 132.0  0.48
## bp          2 105  75.3 12.21   74.0   75.68  8.90 30.0 106  76.0 -0.37
## bmi         3 105  33.8  6.28   33.3   33.38  5.49 20.8  50  29.2  0.54
## age         4 105  39.2  5.45   38.0   38.96  5.93 31.0  50  19.0  0.26
## type*       5 105   1.5  0.50    1.0    1.49  0.00  1.0   2   1.0  0.02
## age_cat*    6 105   2.0  0.00    2.0    2.00  0.00  2.0   2   0.0   NaN
##          kurtosis   se
## glu         -0.67 3.09
## bp           1.32 1.19
## bmi         -0.16 0.61
## age         -1.03 0.53
## type*       -2.02 0.05
## age_cat*      NaN 0.00
```

```
## ------------------------------------------------------------
## group: >50
##          vars  n   mean    sd median trimmed   mad  min   max range  skew
## glu         1 22 144.23 33.61  143.5  144.22 45.96 90.0 197.0 107.0 -0.03
## bp          2 22  80.45 11.58   76.0   79.00  7.41 60.0 110.0  50.0  1.05
## bmi         3 22  33.00  7.00   33.1   32.86  6.45 19.6  46.5  26.9  0.17
## age         4 22  57.95  7.32   56.0   56.78  5.93 51.0  81.0  30.0  1.46
## type*       5 22   1.55  0.51    2.0    1.56  0.00  1.0   2.0   1.0 -0.17
## age_cat*    6 22   3.00  0.00    3.0    3.00  0.00  3.0   3.0   0.0   NaN
##          kurtosis   se
## glu         -1.33 7.17
## bp           0.94 2.47
## bmi         -0.69 1.49
## age          2.06 1.56
## type*       -2.06 0.11
## age_cat*      NaN 0.00
```

7.3.2 Graphical representation

In dealing with grouped data, the purpose is not only to create plots for each group but also to compare the plots between groups.

7.3.2.1 (1) Histograms

We have already mentioned earlier in this lecture how to obtain a histogram by typing hist(x), where x is the variable containing the data. And we have also seen in previous lecture how to use par() to combine two plots in one image.

```
opar<-par(mfrow=c(3,1))
hist(mydata$glu[mydata$age_cat==levels(mydata$age_cat)[1]],breaks=10,col="violet",freq=F)
hist(mydata$glu[mydata$age_cat==levels(mydata$age_cat)[2]],breaks=10,col="tan1",freq=F)
hist(mydata$glu[mydata$age_cat==levels(mydata$age_cat)[3]],breaks=10,col="skyblue",freq=F)
```

bookdown-demo_files/figure-latex/unnamed-chunk-165-1.pdf

```
par(opar)
```

We can also compare groups via density plot:

```
install.packages("sm")
```

```
library(sm)
sm.density.compare(mydata$glu, mydata$age_cat, xlab="Glucose level",col=c(1,2,3),lty=c(1,2,3),lwd=2)
title(main="Glucose distribution by age")
legend(175,0.014,legend=c("age<=30","30<age<=50","age>50"),bty="n",col=c(1,2,3),lty=c(1,2,3),lwd=2)
```

bookdown-demo_files/figure-latex/unnamed-chunk-166-1.pdf

####(2) Boxplot

Boxplot can be used by y~group where a separate boxplot for numeric variable y is generated for each value of group.

```
#Draw the boxplot, with the number of individuals per group
boxplot(mydata$glu ~ mydata$age_cat , col=rgb(0.1,0.9,0.3,0.4),main="glucose distribution by age")
#add means to the plot
points(c(1,2,3),tapply(mydata$glu,mydata$age_cat,mean),pch="x",cex=1.6, col=2)
```

bookdown-demo_files/figure-latex/unnamed-chunk-167-1.pdf

We can also plot against two crossed factors:

```
boxplot(mydata$glu~mydata$type*mydata$age_cat, notch=F,
  col=(c("gold","tan")),
  main="glucose level in different age and type groups")
```

bookdown-demo_files/figure-latex/unnamed-chunk-168-1.pdf

```
#we can compare medians using the notch=T statement in the boxplot() function.
```

7.3.2.2 (3) Violin Plots

A violin plot is a combination of a boxplot and a kernel density plot.

```
install.packages("vioplot")
```

```
library(vioplot)
x1 <- mydata$glu[mydata$age_cat==levels(mydata$age_cat)[1]]
x2 <- mydata$glu[mydata$age_cat==levels(mydata$age_cat)[2]]
x3 <- mydata$glu[mydata$age_cat==levels(mydata$age_cat)[3]]
vioplot(x1, x2, x3, names=c("age<=30", "30<age<=50", "age>50"),
   col="gold")
title("Violin Plots of glucose level")
title(ylab="glucose level")
```

bookdown-demo_files/figure-latex/unnamed-chunk-169-1.pdf

7.4 For two quantitative variables

7.4.1 Numerical representation

For two quantitative variables, we may want to know the correlation between the two variables.We can use the cor( ) function to produce correlations and the cov( ) function to produces covariances.For cor() and cov() functions, x must be numeric.

```
# Correlations/covariances among numeric variables in  mydata. Use listwise deletion of missing data if
cor(subset(mydata,select=-c(age_cat,type)), use="complete.obs", method="kendall")
```

```
##       glu    bp    bmi    age
## glu 1.000 0.149 0.1690 0.1721
## bp  0.149 1.000 0.2370 0.2360
## bmi 0.169 0.237 1.0000 0.0908
## age 0.172 0.236 0.0908 1.0000
```

```
cov(subset(mydata,select=-c(age_cat,type)), use="complete.obs")
```

```
##       glu    bp   bmi    age
## glu 930.3  76.0 60.90  76.10
## bp   76.0 163.8 31.52  44.23
## bmi  60.9  31.5 53.04   3.67
## age  76.1  44.2  3.67 113.13
```

```
#generate correlations between the columns of X and the columns of Y
x <- mydata[1:2]
y <- mydata[3:4]
cor(x, y)
```

```
##       bmi   age
## glu 0.274 0.235
## bp  0.338 0.325
```

7.4.2 Graphical representation

A scatter plot pairs up values of two quantitative variables in a data set and display them as geometric points inside a Cartesian diagram. One variable is chosen in the horizontal axis and another in the vertical axis.

```
#a simple scatter plot
attach(mydata)#the R knows where to find the following variables#
```

```
## The following object is masked _by_ .GlobalEnv:
##
##     age
```

```
## The following object is masked from mydata (pos = 10):
##
```

```
##     age
```

```
plot(bmi, glu, main="Scatterplot Example",
    xlab="BMI ", ylab="glucose level ", pch=19)
# Add fit lines
abline(lm(glu~bmi), col="red") # regression line (y~x)
lines(lowess(bmi,glu), col="blue") # lowess line (x,y)
```

bookdown-demo_files/figure-latex/unnamed-chunk-171-1.pdf

```
detach(mydata)
```

The scatterplot( ) function in the car package offers many enhanced features.

```
# Enhanced Scatterplot
# by age category
library(car) #scatterplot() is built in the car()package
scatterplot(glu ~ bmi | type, data=mydata,
    xlab="BMI", ylab="glucose level",
   main="Enhanced Scatter Plot",
   labels=row.names(mydata),smoother=F,reg.line=F,col=c("green","violet"),pch=c(1,19))
```

bookdown-demo_files/figure-latex/unnamed-chunk-172-1.pdf

```
#If we want to draw a regression line on the plot
scatterplot(glu ~ bmi | type, data=mydata,
    xlab="BMI", ylab="glucose level",
   main="Enhanced Scatter Plot",
   labels=row.names(mydata),smoother=F,col=c("green","violet"),pch=c(1,19))
```

bookdown-demo_files/figure-latex/unnamed-chunk-172-2.pdf

```
#If we want to add a nonparametric-regression smooth by loessLine
scatterplot(glu ~ bmi | type, data=mydata,
    xlab="BMI", ylab="glucose level",
   main="Enhanced Scatter Plot",
   labels=row.names(mydata),reg.line=F,col=c("green","violet"),pch=c(1,19))
```

bookdown-demo_files/figure-latex/unnamed-chunk-172-3.pdf

When we have more than two variables and we want to find the correlation between one variable versus the remaining ones we use scatterplot matrix. We use pairs() function to create matrices of scatterplots.

```
#If we want to a scatterplot matrix
pairs(~glu+bp+bmi+age,data=mydata,
   main="Scatterplot Matrix")
```

bookdown-demo_files/figure-latex/unnamed-chunk-173-1.pdf

Chapter 8

Descriptive statistics with R: Part II-Categorical variables and ggplot

As the previous lecture, the data set diabetes_new will be used as the example, which contains four quantitative variables: glu, bp, bmi and age; two categorical variable: type and age_cat.

```
#read in the dataset
mydata<-read.csv("diabetes_new.csv",header=T)
dim(mydata)
```

```
## [1] 332   6
```

```
head(mydata)
```

```
##   glu bp  bmi age type age_cat
## 1 148 72 33.6  50  Yes   30-50
## 2  85 66 26.6  31   No   30-50
## 3  89 66 28.1  21   No    <=30
## 4  78 50 31.0  26  Yes    <=30
## 5 197 70 30.5  53  Yes     >50
## 6 166 72 25.8  51  Yes     >50
```

8.1 Numerical representation

Categorical data are usually described in the form of tables.

```
table(mydata$age_cat)
```

```
##
##  <=30   >50 30-50
##   205    22   105
```

Relative frequencies in a table are expressed as proportions of the row or column totals. Tables of relative frequencies can be constructed using the following two codes:

```
#Method 1
round(table(mydata$age_cat)/length(mydata$age_cat),2)
```

```
##
##  <=30   >50 30-50
```

```
##  0.62  0.07  0.32
```

```
#Method 2
mytab<-table(mydata$age_cat)
round(prop.table(mytab),2)
```

```
##
##  <=30   >50 30-50
##  0.62  0.07  0.32
```

If we want to investigate the bivariate frequency distribution of two variables:

```
df<-table(mydata$age_cat,mydata$type)
print(df)
```

```
##
##          No Yes
##   <=30  160  45
##   >50    10  12
##   30-50  53  52
```

```
prop.table(df)
```

```
##
##             No    Yes
##   <=30  0.4819 0.1355
##   >50   0.0301 0.0361
##   30-50 0.1596 0.1566
```

```
prop.table(df,1)
```

```
##
##            No   Yes
##   <=30  0.780 0.220
##   >50   0.455 0.545
##   30-50 0.505 0.495
```

```
#Note that the rows (1st index) sum to 1
prop.table(df,2)
```

```
##
##             No    Yes
##   <=30  0.7175 0.4128
##   >50   0.0448 0.1101
##   30-50 0.2377 0.4771
```

```
##Note that the columns (2nd index) sum to 1
```

Marginal tables:

```
margin.table(df,1) #The second argument  is the number of the marginal index: 1  give row totals
```

```
##
##  <=30   >50 30-50
##   205    22   105
```

```
margin.table(df,2)#The second argument  is the number of the marginal index: 2 give column totals
```

```
##
##  No Yes
```

```
## 223 109
```

8.2 Graphical representation

For presentation purposes, it may be desirable to display a graph rather than a table of counts or percentages.

8.2.1 (1) Barplots

```r
#Simple Barplots for counts
counts <- table(mydata$age_cat)
barplot(counts, main="Age Distribution",
    xlab="age",ylab="Counts")
```

bookdown-demo_files/figure-latex/unnamed-chunk-180-1.pdf

```r
#Simple Barplots for frequency
freq <- prop.table(counts)
bp<-barplot(freq, main="Age Distribution",
    xlab="age",ylab="relative frequency")
```

bookdown-demo_files/figure-latex/unnamed-chunk-180-2.pdf

```r
# Simple Horizontal Bar Plot with Added Labels
counts <- table(mydata$age_cat)
barplot(counts, main="Age Distribution", horiz=TRUE,
  names.arg=c("age>=30", "30<age<=50", "age>50"),xlab="Counts")
```

bookdown-demo_files/figure-latex/unnamed-chunk-180-3.pdf

```r
# Stacked Bar Plot with Colors and Legend
counts <- table(mydata$age_cat, mydata$type)
barplot(counts, main="Age Distribution by type",
 col=c("darkblue","seagreen3","tan3"),
    legend = rownames(counts),xlab="type",ylab="counts")
```

bookdown-demo_files/figure-latex/unnamed-chunk-180-4.pdf

```
# Grouped Bar Plot
counts <- table(mydata$age_cat, mydata$type)
barplot(counts, main="Age Distribution by type",
  col=c("darkblue","seagreen3","tan3"),
    legend = rownames(counts), beside=TRUE,xlab="type",ylab="counts")
```

bookdown-demo_files/figure-latex/unnamed-chunk-180-5.pdf

8.2.2 (2) Dotcharts

Dotcharts contain the same information as barplots with beside=T but give quite a different visual impression.

```
counts <- table(mydata$age_cat, mydata$type)
dotchart(counts,labels=row.names(counts),cex=.7,lcol="gray")
```

bookdown-demo_files/figure-latex/unnamed-chunk-181-1.pdf

8.2.3 (3) Piecharts

Pie charts are created with the function pie(x, labels=) where x is a non-negative numeric vector indicating the area of each slice and labels= notes a character vector of names for the slices.

```
opar <- par(mfrow=c(1,2),mex=0.8, mar=c(1,1,2,1))
pie(table(mydata$age_cat[mydata$type=="No"]), main="Age distribution for type==No", col=rainbow(3))
pie(table(mydata$age_cat[mydata$type=="Yes"]),main="Age distribution for type==Yes", col=rainbow(3))
```

bookdown-demo_files/figure-latex/unnamed-chunk-182-1.pdf

```
par(opar)
```

3D pie plots:

```
install.packages("plotrix")
```

```
# 3D Exploded Pie Chart
library(plotrix)
counts <- table(mydata$age_cat)
lbls <- c("age>=30", "30<age<=50", "age>50")
pie3D(counts,labels=names(counts),explode=0.1,
    main="Pie Chart of age")
```

bookdown-demo_files/figure-latex/unnamed-chunk-183-1.pdf

8.3 ggplot2 function

ggplot2() is a commonly used package for graphing purpose in R. Compared to base graphics, ggplot2 is more flexible for graphics. The package author Hadley Wickham describes ggplot2 as: "a plotting system for R, based on the grammar of graphics, which tries to take the good parts of base and lattice graphics and none of the bad parts. It takes care of many of the fiddly details that make plotting a hassle (like drawing legends) as well as providing a powerful model of graphics that makes it easy to produce complex multi-layered graphics."

Please note that ggplot2() uses a different system for adding plot elements and the data should always be in a dataframe.

```
install.packages("ggplot2")
```

```
#load the package
library(ggplot2)
```

All ggplot2 plots with a call to ggplot(), supplying default data and aesthethic mappings, specified by aes(). We can then add layers, scales, coords and facets with +.

```
ggplot      #Create a new plot
aes         #Construct aesthetic mappings
+.gg        #Add components to a plot
```

There are two major functions that we will use in ggplot2(): qplot() and ggplot(). qplot() is for quick plot.

```
geom_point            #scatter plots, dot plots,etc
geom_line             #time series, trend line,etc
geom_histogram        #histogram
geom_violin           #violion plot
```

For more reference, please refer to (http://ggplot2.tidyverse.org/reference/) and (http://tutorials.iq.harvard.edu/R/Rgraphics/Rgraphics.html).

Some examples:

8.3.1 (1)a simple histogram for a single quantative variable

```
qplot(data=mydata,x=glu)#For a histogram, all we need to tell qplot()is which dataframe to look in and
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

bookdown-demo_files/figure-latex/unnamed-chunk-185-1.pdf

```
ggplot(mydata, aes(x = glu)) +geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

bookdown-demo_files/figure-latex/unnamed-chunk-185-2.pdf

8.3.2 (2)for a quantative variable by groups

```
# If we want to see how the raw values of glucose are distributed over different age groups.
qplot(data=mydata,x=age_cat,y=glu)
```

bookdown-demo_files/figure-latex/unnamed-chunk-186-1.pdf

```
ggplot(mydata,aes(x=age_cat,y=glu))+geom_point()
```

bookdown-demo_files/figure-latex/unnamed-chunk-186-2.pdf

8.3.3 (3) for two quantitative variables

```
#a simple scatter lot
qplot(data=mydata,x=bmi,y=glu)
```

bookdown-demo_files/figure-latex/unnamed-chunk-187-1.pdf

```
p1<-ggplot(mydata,aes(x=bmi,y=glu))
p1+geom_point()
```

bookdown-demo_files/figure-latex/unnamed-chunk-187-2.pdf

```
p1+geom_point(colour="violet")#colour: "outside" color
```

bookdown-demo_files/figure-latex/unnamed-chunk-187-3.pdf

```
#add a nonparametric-regression smooth line
p1+geom_point()+geom_smooth()
```

```
## `geom_smooth()` using method = 'loess'
```

bookdown-demo_files/figure-latex/unnamed-chunk-187-4.pdf

```
p1+geom_point(shape=3)+geom_smooth(colour="red")#shape: shape of the points
```

```
## `geom_smooth()` using method = 'loess'
```

bookdown-demo_files/figure-latex/unnamed-chunk-187-5.pdf

```
#a scatter plot by type category
qplot(data=mydata,x=bmi,y=glu,col=type)
```

bookdown-demo_files/figure-latex/unnamed-chunk-187-6.pdf

```
p1+geom_point(aes(color=type))
```

bookdown-demo_files/figure-latex/unnamed-chunk-187-7.pdf

```
#add a smooth line
p1 +geom_point(aes(color = type)) +geom_smooth()
```

```
## `geom_smooth()` using method = 'loess'
```

bookdown-demo_files/figure-latex/unnamed-chunk-187-8.pdf

```
p1 +geom_point(aes(color = type)) +geom_smooth(linetype=2)#linetype:type of the line
```

```
## `geom_smooth()` using method = 'loess'
```

bookdown-demo_files/figure-latex/unnamed-chunk-187-9.pdf

We can plot the relationship between bmi and glucose for each type, with each type separated into the various age categories.

```
qplot(data=mydata,x=bmi,y=glu,color=type,facets = age_cat~type)
```

bookdown-demo_files/figure-latex/unnamed-chunk-188-1.pdf

```
p1 +geom_point(aes(color = type))+facet_grid(age_cat~type)
```

bookdown-demo_files/figure-latex/unnamed-chunk-188-2.pdf

8.3.4 (4) for categorial variable

```
#a simple boxplot
qplot(data=mydata,x=age_cat,y=glu,geom="boxplot")
```

bookdown-demo_files/figure-latex/unnamed-chunk-189-1.pdf

```
p1+geom_boxplot(aes(x=age_cat,y=glu))
```

bookdown-demo_files/figure-latex/unnamed-chunk-189-2.pdf

```
p1+geom_boxplot(aes(x=age_cat,y=glu),fill="yellow")#fill parameter: "inside" color
```

bookdown-demo_files/figure-latex/unnamed-chunk-189-3.pdf

```
#a jitter plot
qplot(data=mydata,x=age_cat,y=glu,geom="jitter")
```

bookdown-demo_files/figure-latex/unnamed-chunk-189-4.pdf

```
p1+geom_jitter(aes(x=age_cat,y=glu))
```

bookdown-demo_files/figure-latex/unnamed-chunk-189-5.pdf

Chapter 9

Basic statistical tests with R

R provides a number of functions for standard statistical tests comparing means, medians and proportions.In this lecture, we will learn how to perform simple statistical tests like the t-test, u-test, chi-squared.

9.1 Two-sample tests

9.1.1 t-test

The t-test is used to determine statistical differences between two samples.

```
#independent two group t-test
x1<-1:10
x2<-7:20
t.test(x1,x2)
```

```
##
##  Welch Two Sample t-test
##
## data:  x1 and x2
## t = -5, df = 20, p-value = 2e-05
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -11.05  -4.95
## sample estimates:
## mean of x mean of y
##       5.5      13.5
```

```
t.test(x1,x2,alternative="greater")#greater
```

```
##
##  Welch Two Sample t-test
##
## data:  x1 and x2
## t = -5, df = 20, p-value = 1
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  -10.5   Inf
## sample estimates:
```

```
## mean of x mean of y
##       5.5      13.5
```

```r
#paired t-test
x1<1:10
```

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

```r
x2<-sample(21:30,10,replace=T)
t.test(x1,x2,paired=TRUE) # where x1 & x2 are numeric
```

```
##
##  Paired t-test
##
## data:  x1 and x2
## t = -20, df = 9, p-value = 5e-08
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -21.4 -16.2
## sample estimates:
## mean of the differences
##                   -18.8
```

```r
#One sample test
t.test(x1,mu=6) # Ho: mu=6
```

```
##
##  One Sample t-test
##
## data:  x1
## t = -0.5, df = 9, p-value = 0.6
## alternative hypothesis: true mean is not equal to 6
## 95 percent confidence interval:
##  3.33 7.67
## sample estimates:
## mean of x
##       5.5
```

This version of the test does not assume that the variance of the two samples is equal.We can use the var.equal = TRUE option to specify equal variances and a pooled variance estimate.

```r
x1<-1:10
x2<-7:20
t.test(x1,x2,var.equal=T)
```

```
##
##  Two Sample t-test
##
## data:  x1 and x2
## t = -5, df = 20, p-value = 4e-05
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -11.22  -4.78
## sample estimates:
## mean of x mean of y
##       5.5      13.5
```

9.1.2 Nonparametric tests

A popular alternative to parametric tests are non-parametric (rank-based) tests.Wilcoxon tests can be used for two samples.

```
x <- c(0.80, 0.83, 1.89, 1.04, 1.45, 1.38, 1.91, 1.64, 0.73, 1.46)
y <- c(1.15, 0.88, 0.90, 0.74, 1.21)
wilcox.test(x, y)
```

```
##
##  Wilcoxon rank sum test
##
## data:  x and y
## W = 40, p-value = 0.3
## alternative hypothesis: true location shift is not equal to 0
```

```
wilcox.test(x, y, alternative = "g",exat=T) #For small sample sizes we should  set the argument exact=T
```

```
##
##  Wilcoxon rank sum test
##
## data:  x and y
## W = 40, p-value = 0.1
## alternative hypothesis: true location shift is greater than 0
```

The standard test is rank-based and only the p-value is reported. Reporting p-values without an appropriate effect measures is not good for scientific practice. The wilcox.test() function allows to compute an effect measure by setting the argument conf.int=T.

```
x <- c(0.80, 0.83, 1.89, 1.04, 1.45, 1.38, 1.91, 1.64, 0.73, 1.46)
y <- c(1.15, 0.88, 0.90, 0.74, 1.21)
wilcox.test(x, y, conf.int=T)
```

```
##
##  Wilcoxon rank sum test
##
## data:  x and y
## W = 40, p-value = 0.3
## alternative hypothesis: true location shift is not equal to 0
## 95 percent confidence interval:
##  -0.15  0.76
## sample estimates:
## difference in location
##                  0.305
```

9.2 Analysis Of Variance

When we have more than two samples to compare, we would usually use analysis of variance, which is a global test for equality of means. We are assuming normality and constant variance for the model error term when we attempt to use parametric tesets. We may need to assess the assumptions so that we can decide to use parametric or non-parametric methods.

As the previous lecture, the data set diabetes_new will be used as the example, which contains four quantitative variables: glu, bp, bmi and age; two categorical variable: type and age_cat.

```r
#read in the dataset
mydata<-read.csv("data/diabetes_new.csv",header=T)
dim(mydata)
```

```
## [1] 332   6
```

```r
head(mydata)
```

```
##   glu bp  bmi age type age_cat
## 1 148 72 33.6  50  Yes   30-50
## 2  85 66 26.6  31   No   30-50
## 3  89 66 28.1  21   No    <=30
## 4  78 50 31.0  26  Yes    <=30
## 5 197 70 30.5  53  Yes     >50
## 6 166 72 25.8  51  Yes     >50
```

9.2.1 Assumptions assessment

9.2.1.1 (1) Normality

As we mentioned before, we can use Q-Q plot to assess the normality.

```r
# Q-Q Plot for variable glu
attach(mydata)
```

```
## The following objects are masked _by_ .GlobalEnv:
##
##     age, bp
```

```
## The following object is masked from mydata (pos = 10):
##
##     age
```

```r
qqnorm(glu)
qqline(glu)
```

bookdown-demo_files/figure-latex/unnamed-chunk-196-1.pdf

```r
#Significant departures from the line suggest violations of normality.
```

For statistical test:

```r
shapiro.test(glu)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  glu
## W = 0.9, p-value = 6e-10
```

We can see that the glucose level is not normally distributed.

9.2.1.2 (2) Homogeneity of Variances

```
# Test whether the distribution of a variable has the same variance in all groups
bartlett.test(glu~age_cat)#the tilde symbol (~) should be read as "described by".
```

```
##
##  Bartlett test of homogeneity of variances
##
## data:  glu by age_cat
## Bartlett's K-squared = 3, df = 2, p-value = 0.2
```

```
# non-parametric test of the equality of variances
fligner.test(glu~age_cat)
```

```
##
##  Fligner-Killeen test of homogeneity of variances
##
## data:  glu by age_cat
## Fligner-Killeen:med chi-squared = 4, df = 2, p-value = 0.1
```

9.2.2 Parametric test

If the above assumptions can be met, we can choose the parametric tests to compare the central tendencies of several independent groups

```
myanova<-aov(glu~age_cat)
summary(myanova)
```

```
##              Df Sum Sq Mean Sq F value  Pr(>F)
## age_cat       2  19431    9716    11.1 2.2e-05 ***
## Residuals   329 288505     877
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
coefficients(myanova)
```

```
##  (Intercept)   age_cat>50 age_cat30-50
##       114.69        29.54         8.26
```

The results above returns a significant overall p-value. However, there are 3 age groups, now the question quickly arises of where the difference lies. It becomes necessary to compare the individual groups. Therefore we need a post-hoc test.

```
TukeyHSD(myanova)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = glu ~ age_cat)
##
## $age_cat
##              diff     lwr   upr p adj
## >50-<=30    29.54  13.898 45.18 0.000
## 30-50-<=30   8.26  -0.102 16.63 0.054
## 30-50->50  -21.27 -37.622 -4.93 0.007
```

```
#The table/output shows us the difference between pairs, the 95% confidence interval(s) and the p-value
```

We can use box plots and line plots to visualize group differences

```
plot(TukeyHSD(myanova))
```

bookdown-demo_files/figure-latex/unnamed-chunk-201-1.pdf

```
# Plot Means with Error Bars
library(gplots)
plotmeans(glu~age_cat,xlab="age category",
  ylab="glucose level", main="Mean Plot\nwith 95% CI")
```

bookdown-demo_files/figure-latex/unnamed-chunk-201-2.pdf

```
#produce mean plots for single factors, and includes confidence intervals.
```

We can also choose to use a function called pairwise.t.test to compute all possible two-group comparisons. pairwise.t.test also allows to make adjustments for multiple comparisons.As we know, performing many tests will increase the probability of finding one of them to be significant; that is, the p-values tend to be exaggerated. A common adjustment method is the Bonferroni correction, which is based on the fact that the probability of observing at least one of n events is less than the sum of the probabilities for each event.

```
pairwise.t.test(glu, age_cat, p.adj="bonferroni")
```

```
##
##  Pairwise comparisons using t tests with pooled SD
##
## data:  glu and age_cat
##
##       <=30  >50
## >50   4e-05 -
## 30-50 0.062 0.007
##
## P value adjustment method: bonferroni
```

The output is a table of p-values for the pairwise comparisons. Here, the p-values have been adjusted by the Bonferroni method. If that results in a value bigger than 1, then the adjustment procedure sets the adjusted p-value to 1.

The default method for pairwise.t.test is actually not the Bonferroni correction but a variant due to Holm. In this method, only the smallest p needs to be corrected by the full number of tests, the second smallest is corrected by n 1, etc., unless that would make it smaller than the previous one, since the order of the p-values should be unaffected by the adjustment.

```
pairwise.t.test(glu, age_cat)
```

```
##
```

```
##  Pairwise comparisons using t tests with pooled SD
##
## data:  glu and age_cat
##
##       <=30  >50
## >50   4e-05 -
## 30-50 0.021 0.005
##
## P value adjustment method: holm
```

To know more about adjustment methods:

```
?p.adjust
```

9.2.3 Nonparametric test

There are some non-parametric tests that do not require the above assumptions.

```r
kruskal.test(glu~age_cat)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  glu by age_cat
## Kruskal-Wallis chi-squared = 20, df = 2, p-value = 1e-04
```

We can see that the global test indicates evidence for statistically significant group difference. However, at this stage we cannot tell which of the levels is different from which. We can get an overview by creating a simple boxplot of the data.

```r
boxplot(glu~age_cat)
```

bookdown-demo_files/figure-latex/unnamed-chunk-205-1.pdf

We can also perform a post-hoc analysis to determine which levels of the independent variable significantly differ from each other level.

It is also possible to perform the pairwise t tests so that they do not use a common pooled standard deviation. This is controlled by the argument pool.sd.

```r
pairwise.t.test(glu,age_cat,pool.sd=F, p.adj="bonferroni")
```

```
##
##  Pairwise comparisons using t tests with non-pooled SD
##
## data:  glu and age_cat
##
##       <=30  >50
## >50   0.002 -
## 30-50 0.075 0.032
##
## P value adjustment method: bonferroni
```

Or we can choose the dunn.test from the FSA package:

```
install.packages("FSA")
install.packages("dunn.test")
```

```
library(FSA)
dunnTest(glu~age_cat,method="bonferroni")#use bonferroni for the adjustment of multiple testing
```

```
## Dunn (1964) Kruskal-Wallis multiple comparison

##   p-values adjusted with the Bonferroni method.

##     Comparison     Z  P.unadj    P.adj
## 1   <=30 - >50 -3.93 8.65e-05 0.000259
## 2 <=30 - 30-50 -2.24 2.48e-02 0.074472
## 3  >50 - 30-50  2.61 9.12e-03 0.027359
```

9.2.4 Multivariate analysis of variance

If there is more than one dependent variable, we can test them simultaneously using a multivariate analysis of variance (MANOVA).

```
y<-cbind(glu,bp,bmi)
fit <- manova(y~age_cat)
summary(fit, test="Pillai")
summary.aov(fit ) #get univariate statistics
```

9.2.5 Two-way analysis of variance

One-way analysis of variance deals with one-way classifications of data. It is also possible to analyze data that are cross-classified according to several criteria.

```
anova(lm(glu~age_cat+type))
```

```
## Analysis of Variance Table
##
## Response: glu
##            Df Sum Sq Mean Sq F value  Pr(>F)
## age_cat     2  19431    9716    14.7 7.9e-07 ***
## type        1  71301   71301   107.7 < 2e-16 ***
## Residuals 328 217204     662
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

9.3 Correlation

A correlation coefficient is a symmetric, scale-invariant measure of association between two random variables. It ranges from 1 to +1, where the extremes indicate perfect correlation and 0 means no correlation. The sign is negative when large values of one variable are associated with small values of the other and positive if both variables tend to be large or small at the same time.

For the correlation between two continuous variables, it is suggested that we should start with a visualization of the relationship to gain some understanding of the general nature of the potential relationship:

```
library(ggplot2)
p1<-ggplot(mydata,aes(x=bmi,y=glu))
p1+geom_point()
```

bookdown-demo_files/figure-latex/unnamed-chunk-209-1.pdf

The Pearson correlation

The Pearson correlation is parametric and rooted in the two-dimensional normal distribution where the theoretical correlation describes the contour ellipses for the density.

```
cor(glu,bmi,method="pearson")
```

```
## [1] 0.274
```

However, the calculations above give no indication of whether the correlation is significantly different from zero. To that end, we need cor.test:

```
cor.test(glu,bmi,method="pearson")#the default is pearson.
```

```
##
##  Pearson's product-moment correlation
##
## data:  glu and bmi
## t = 5, df = 300, p-value = 4e-07
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.172 0.371
## sample estimates:
##   cor
## 0.274
```

```
#This returns us p-value,confidence interval as well as the correlation coefficient.
```

9.3.1 Spearman's

Spearman's is non-parametric, not depending on the normal distribution and, indeed

```
cor.test(glu,bmi,method="spearman")
```

```
## Warning in cor.test.default(glu, bmi, method = "spearman"): Cannot compute
## exact p-value with ties
```

```
##
##  Spearman's rank correlation rho
##
## data:  glu and bmi
## S = 5e+06, p-value = 3e-06
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##   rho
## 0.253
```

9.3.2 Kendall's

```
cor.test(glu,bmi,method="kendall")
```

```
##
##  Kendall's rank correlation tau
##
## data:  glu and bmi
## z = 5, p-value = 5e-06
## alternative hypothesis: true tau is not equal to 0
## sample estimates:
##   tau
## 0.169
```

All of the the three methods show significant results.

```
#as we attach the data before, we need to detach the data now since we will not use it temporarily.
detach(mydata)
```

9.4 Tabular data

9.4.1 Single proportion

Tests of single proportions are generally based on the binomial distribution with size parameter N and probability parameter p.

```
prop.test(45,300,0.15)
```

```
##
##  1-sample proportions test without continuity correction
##
## data:  45 out of 300, null probability 0.15
## X-squared = 0, df = 1, p-value = 1
## alternative hypothesis: true p is not equal to 0.15
## 95 percent confidence interval:
##  0.114 0.195
## sample estimates:
##    p
## 0.15
```

The three arguments are the number of positive outcomes, the total number, and the theoretical probability parameter that we want to test for.

We can also use binom.test to obtain a test in the binomial distribution.

```
binom.test(45,300,0.15)
```

```
##
##  Exact binomial test
##
## data:  45 and 300
## number of successes = 40, number of trials = 300, p-value = 1
## alternative hypothesis: true probability of success is not equal to 0.15
## 95 percent confidence interval:
##  0.112 0.196
```

```
## sample estimates:
## probability of success
##                   0.15
```

9.4.2 Two independent proportions

prop.test:

The function prop.test can also be used to compare two or more pro- portions. For that purpose, the arguments should be given as two vectors, where the first contains the number of positive outcomes and the second the total number for each group.

```r
x <- c(9,4)
n <- c(12,13)
prop.test(x,n)
```

```
##
##  2-sample test for equality of proportions with continuity
##  correction
##
## data:  x out of n
## X-squared = 3, df = 1, p-value = 0.07
## alternative hypothesis: two.sided
## 95 percent confidence interval:
##  0.0115 0.8731
## sample estimates:
## prop 1 prop 2
##  0.750  0.308
```

```r
#Please note that the confidence interval given is for the difference in proportions.
```

fisher.test: If we want to be sure that the p-value is correct, we can use Fisher's exact test.We illustrate this using the same data as in the preceding section.

```r
y <- matrix(c(9,4,3,9),2)
print(y)# the second column of the table is the number of negative outcomes
```

```
##      [,1] [,2]
## [1,]    9    3
## [2,]    4    9
```

```r
fisher.test(y)
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  y
## p-value = 0.05
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##   0.901 57.255
## sample estimates:
## odds ratio
##       6.18
```

Pearson's Chi-squared Test

The chi-square test is used to compare the observed distribution to an expected distribution, in a situation where we have two or more categories in a discrete data. In other words, it compares multiple observed proportions to expected probabilities.

```
chisq.test(x, y = NULL, correct = TRUE,
           p = rep(1/length(x), length(x)), rescale.p = FALSE,
           simulate.p.value = FALSE, B = 2000)
```

p=() is a vector of probabilities of the same length of x. rescale.p is a logical scalar; if TRUE then p is rescaled to sum to 1. If rescale.p is FALSE, and p does not sum to 1, an error is given.simulate.p.value is a logical indicating whether to compute p-values by Monte Carlo simulation. B is an integer specifying the number of replicates used in the Monte Carlo test.

```r
chisq.test(y)
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  y
## X-squared = 3, df = 1, p-value = 0.07
```

9.4.3 k proportions

In many cases, we want to compare more than two proportions.Let's first create a new table:

```r
temp<-data.frame(x=rep(c("yes","no"),200),y=sample(c(1,2,3),200,replace=T))
x<-table(temp$x,temp$y)
print(x)
```

```
##
##        1  2  3
##   no  68 62 70
##   yes 80 62 58
```

To use prop.test on a table like x, we need to convert it to a vector of "successes".

```r
x.yes <- x["yes",]
x.total <- margin.table(x,2)
print(x.yes)
```

```
##  1  2  3
## 80 62 58
```

```r
print(x.total)
```

```
##
##   1   2   3
## 148 124 128
```

```r
#conduct the test
prop.test(x.yes,x.total)
```

```
##
##  3-sample test for equality of proportions without continuity
##  correction
##
## data:  x.yes out of x.total
## X-squared = 2, df = 2, p-value = 0.4
```

```
## alternative hypothesis: two.sided
## sample estimates:
## prop 1 prop 2 prop 3
##  0.541  0.500  0.453
```

9.4.4 r × c tables

For the analysis of tables with more than two classes on both sides, we can use chisq.test or fisher.test.We should note that the fisher test can be very computationally demanding if the cell counts are large and there are more than two rows or columns.

```
#Let's create a hypothetical dataframe with caffeine consumption and weight
x <- matrix(c(652,1537,598,242,36,46,38,21,218,327,106,67),nrow=3,byrow=T)
colnames(x) <- c("0","1-150","151-300",">300")
rownames(x) <- c("underweight","normal","overweight")
x
```

```
##               0 1-150 151-300 >300
## underweight 652  1537     598  242
## normal       36    46      38   21
## overweight  218   327     106   67
```

```
#chi-square test
mychi<-chisq.test(x)
#There are more we can get from the chi-square test
names(mychi)
```

```
## [1] "statistic" "parameter" "p.value"   "method"    "data.name" "observed"
## [7] "expected"  "residuals" "stdres"
```

```
#To see the observed values :
mychi$observed
```

```
##               0 1-150 151-300 >300
## underweight 652  1537     598  242
## normal       36    46      38   21
## overweight  218   327     106   67
```

```
#To see the expected values:
mychi$expected
```

```
##                 0  1-150 151-300  >300
## underweight 705.8 1488.0   578.1 257.1
## normal       32.9   69.3    26.9  12.0
## overweight  167.3  352.7   137.0  60.9
```

```
#To see the residuals
mychi$residuals
```

```
##                  0 1-150 151-300   >300
## underweight -2.026  1.27   0.829 -0.941
## normal       0.548 -2.80   2.138  2.611
## overweight   3.919 -1.37  -2.650  0.776
```

It is often useful to see where the differences lie. Such a table cannot be directly extracted, but it is easy to calculate:

```
E <- mychi$expected
O <- mychi$observed
(O-E)^2/E
```

```
##                  0 1-150 151-300  >300
## underweight  4.106  1.61   0.687 0.886
## normal       0.301  7.82   4.571 6.817
## overweight  15.356  1.88   7.025 0.602
```

We can also use the Chi-square test for the goodness of fit test. For goodness of fit test, we are actually doing some statistical testing to see if the reference distribution of the data is different from the primary distribution. Reference distribution is defined as a distribution which we assume fits the data the best. Our hypothesis tests if this assumption is correct or not.Primary distribution is defined as actual distribution that the data is sampled from. In practice this distribution is unknown and we try to estimate and find that distribution.

Say if we have four colors-red,green,black,blue, the counts of each color is 81.50,27,49. We want to know that whether the colors are equally common.

```
counts <- c(81, 50, 27,49)
res <- chisq.test(counts, p = c(1/4, 1/4, 1/4,1/4))
print(res)
```

```
##
##  Chi-squared test for given probabilities
##
## data:  counts
## X-squared = 30, df = 3, p-value = 3e-06
```

```
#The p-value of the test is 2.751^{-6}, which is less than the significance level alpha = 0.05. We can
```

Chapter 10

Regressions with R-Linear Regression

In the previous lecture, we described some basic statistical tests in R. The main object of this lecture is to show how to perform simple linear regression and multiple regression analysis.

As the previous lectures, the data set diabetes_new will be used as the example, which contains four quantitative variables: glu, bp, bmi and age; two categorical variable: type and age_cat.

```
#read in the dataset
mydata<-read.csv("data/diabetes_new.csv",header=T)
dim(mydata)
```

```
## [1] 332   6
```

```
head(mydata)
```

```
##   glu bp  bmi age type age_cat
## 1 148 72 33.6  50  Yes   30-50
## 2  85 66 26.6  31   No   30-50
## 3  89 66 28.1  21   No    <=30
## 4  78 50 31.0  26  Yes    <=30
## 5 197 70 30.5  53  Yes     >50
## 6 166 72 25.8  51  Yes     >50
```

```
#for simplicity, we sample 100 observations from the original dataset.
set.seed(2017)
sample.id<-sample(rownames(mydata),100,replace=F)
sample<-mydata[sample.id,]
dim(sample)
```

```
## [1] 100   6
```

```
head(sample)
```

```
##     glu bp  bmi age type age_cat
## 307  94 72 23.1  56   No     >50
## 178 136 84 35.0  35  Yes   30-50
## 155 125 50 33.3  28  Yes    <=30
## 95  162 52 37.2  24  Yes    <=30
## 253  98 60 34.7  22   No    <=30
## 328  88 58 28.4  22   No    <=30
```

10.1 Simple linear regression

10.1.1 Function

```
# For linear regression analysis, the function lm (linear model) is used
fit<-lm (sample$glu~sample$age)
print(fit)
```

```
##
## Call:
## lm(formula = sample$glu ~ sample$age)
##
## Coefficients:
## (Intercept)   sample$age
##      90.278        0.942
```

We can see that the output of lm is very brief. All we see is the estimated intercept () and the estimated slope (). The best-fitting straight line is seen to be glu = 90.2780 + 0.9417 × age, but no tests of significance are given.

The result of lm is a model object. This is a distinctive concept of the S language (of which R is a dialect). Whereas other statistical systems focus on generating printed output that can be controlled by setting options, An lm object does in fact contain much more information than we see when it is printed.

```
summary(fit)
```

```
##
## Call:
## lm(formula = sample$glu ~ sample$age)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -54.06 -21.81  -3.56  26.04  78.41
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   90.278      9.088    9.93  < 2e-16 ***
## sample$age     0.942      0.274    3.44  0.00085 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 29.8 on 98 degrees of freedom
## Multiple R-squared:  0.108,  Adjusted R-squared:  0.0987
## F-statistic: 11.8 on 1 and 98 DF,  p-value: 0.000851
```

```
#The format above looks more like what other statistical packages would output.
```

10.1.2 Output explanation

Let's dissect the output.

```
Call:
lm(formula = glu ~ age)
```

As we can see, the first item shown in the output is the formula R used to fit the data.

```
Residuals:
    Min      1Q  Median      3Q     Max
-54.064 -21.810  -3.564  26.045  78.411
```

The next item in the model output talks about the residuals. Residuals are essentially the difference between the actual observed response values (glucose level in our case) and the response values that the model predicted. The Residuals section of the model output gives a superficial view of the distribution of the residuals that may be used as a quick check of the distributional assumptions. It breaks it down into 5 summary points. The average of the residuals is zero by definition, so when assessing how well the model fit the data, we should look for a symmetrical distribution across these points on zero. In our example, we can see that the distribution of the residuals do not appear to be symmetrical. That means that the model predicts certain points that fall far away from the actual observed points. We could take this further consider plotting the residuals to see whether this normally distributed.

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  90.2780     9.0877   9.934  < 2e-16 ***
age           0.9417     0.2736   3.442 0.000851 ***
```

The next section in the model output talks about the coefficients of the model. Theoretically, in simple linear regression, the coefficients are two unknown constants that represent the intercept and slope terms in the linear model. Except the regression coefficient and the intercept, this time we also have standard errors, t tests, and p-values. The symbols to the right are graphical indicators of the level of significance. The line below the table shows the definition of these indicators; one star means $0.01 < p < 0.05$.

The coefficient t-value is a measure of how many standard deviations our coefficient estimate is far away from 0. We want it to be far away from zero as this would indicate we could reject the null hypothesis - that is, we could declare a relationship between age and glu exist. In our example, the t-statistic values are relatively far away from zero and are large relative to the standard error, which could indicate a relationship exists. In general, t-values are also used to compute p-values.

```
Residual standard error: 29.76 on 98 degrees of freedom
```

Residual Standard Error is measure of the quality of a linear regression fit. Theoretically, every linear model is assumed to contain an error term E. Due to the presence of this error term, we are not capable of perfectly predicting our response variable (glu) from the predictor (age) one. The Residual Standard Error is the average amount that the response will deviate from the true regression line. In our example, the actual bmi for glu can deviate from the true regression line by approximately 29.76 on average. It's also worth noting that the Residual Standard Error was calculated with 98 degrees of freedom. Simplistically, degrees of freedom are the number of data points that went into the estimation of the parameters used after taking into account these parameters (restriction). In our case, we had 100 data points and two parameters (intercept and slope).

```
Multiple R-squared:  0.1078,    Adjusted R-squared:  0.09873
```

The first item above is R2, which in a simple linear regression may be recognized as the squared Pearson correlation coefficient. The other one is the adjusted R2. Adjusted R^2 measures the goodness of fit of a regression model. The R2 statistic can be recognized as a measure of how well the model is fitting the actual data. It takes the form of a proportion of variance. It always lies between 0 and 1. Higher the R^2, better is the model. Our $R^2 = 0.09873$. It represents our regression that does not explain the variance in the response variable well. Roughly only 9.9% of the variance found in the response variable (glu) can be explained by the predictor variable (age). Please note that in multiple regression settings, the R2 will always increase as more variables are included in the model. That's why the adjusted R2 is the preferred measure as it adjusts for the number of variables considered.

```
F-statistic: 11.85 on 1 and 98 DF,  p-value: 0.0008513
```

F-statistic is a good indicator of whether there is a relationship between our predictor and the response variables (whether the regression coefficient is zero). The further the F-statistic is from 1, the better it is.

However, how much larger the F-statistic needs to be depends on both the number of data points and the number of predictors. Generally, when the number of data points is large, an F-statistic that is only a little bit larger than 1 is already sufficient to reject the null hypothesis (H0 : There is no relationship between age and glu). If the number of data points is small, a large F-statistic is required to be able to ascertain that there may be a relationship between predictor and response variables. In our example the F-statistic is 11.85 which is relatively larger than 1 given the size of our data.

10.1.3 Other useful functions

```
coefficients(fit) # model coefficients
```

```
## (Intercept)  sample$age
##      90.278       0.942
```

```
confint(fit, level=0.95) # CIs for model parameters
```

```
##              2.5 % 97.5 %
## (Intercept) 72.244 108.31
## sample$age   0.399   1.48
```

```
anova(fit) # anova table
```

```
## Analysis of Variance Table
##
## Response: sample$glu
##            Df Sum Sq Mean Sq F value  Pr(>F)
## sample$age  1  10492   10492    11.8 0.00085 ***
## Residuals  98  86801     886
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
vcov(fit) # covariance matrix for model parameters
```

```
##             (Intercept) sample$age
## (Intercept)       82.59    -2.3495
## sample$age        -2.35     0.0749
```

Let's visualize the regression graphically

```
plot(sample$age,sample$glu,col = "blue",main = "Simple Regression",cex = 1.3,pch = 16,xlab = "age distri
abline(fit,col="red",lwd=3)
```

bookdown-demo_files/figure-latex/ok-1.pdf

10.1.4 Fitted values and residuals

As shown above, summary can be used to extract information about the results of a regression analysis. Two further extraction functions are fitted and resid.

```
fitted(fit)
```

```
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
## 143 123 117 113 111 111 111 110 111 114 113 137 116 114 119 138 118 129
##  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
## 118 113 114 114 115 110 111 111 124 134 111 121 116 138 131 126 112 111
##  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
## 124 116 127 118 110 110 128 111 116 134 121 126 133 112 111 130 122 126
##  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
## 125 126 111 137 114 111 130 110 111 167 131 115 112 120 111 127 110 117
##  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
## 111 134 110 113 138 135 118 113 113 110 114 117 136 113 111 133 120 141
##  91  92  93  94  95  96  97  98  99 100
## 111 118 110 118 115 126 124 125 114 110
```

```r
#This returns fitted values - the y-values that we would expect for the given x-values according to the
resid(fit)
```

```
##         1         2         3         4         5         6         7
## -49.01583  12.76087   8.35311  49.12010 -12.99641 -22.99641 -22.99641
##         8         9        10        11        12        13        14
##   1.94534  28.00359  -4.82165 -29.87990  10.63466  10.29485  -0.82165
##        15        16        17        18        19        20        21
##  28.46961  -2.30709   1.41136 -34.88961 -10.58864  -0.87990  66.17835
##        22        23        24        25        26        27        28
## -13.82165  47.23660 -23.05466 -22.99641  68.00359  26.81913   0.40165
##        29        30        31        32        33        34        35
##  -7.99641   6.64437  36.29485  27.69291  39.22689 -20.06437 -43.93815
##        36        37        38        39        40        41        42
## -30.99641 -41.18087  30.29485  32.99388  71.41136 -26.05466  20.94534
##        43        44        45        46        47        48        49
## -10.94786 -39.99641 -12.70515 -15.59835 -18.35563 -37.06437  -7.65660
##        50        51        52        53        54        55        56
## -27.93815 -15.99641 -26.83136  64.70262 -54.06437 -37.12262   2.93563
##        57        58        59        60        61        62        63
## -13.99641   0.63466 -26.82165   0.00359  26.16864 -19.05466 -29.99641
##        64        65        66        67        68        69        70
## -32.55952  11.22689 -18.76340 -18.93815 -21.41388  12.00359 -31.00612
##        71        72        73        74        75        76        77
## -21.05466 -14.64689  -9.99641 -49.59835 -16.05466   4.12010  42.69291
##        78        79        80        81        82        83        84
##  26.45990  28.41136 -35.87990  -0.87990  28.94534  -6.82165  46.35311
##        85        86        87        88        89        90        91
##  28.57641 -31.87990   2.00359 -26.65660   4.58612  29.86767  26.00359
##        92        93        94        95        96        97        98
##  78.41136 -11.05466  28.41136   5.23660 -17.06437  49.81913  18.87738
##        99       100
## -16.82165   5.94534
```

```r
#This returns the difference between the observed glu value and the fitted values shown above.

plot(sample$age, sample$glu,col="skyblue4",pch=19)
lines(sample$age,fitted(fit),col="red",lwd=3)
segments(sample$age,fitted(fit),sample$age,sample$glu,col="springgreen4")
```

bookdown-demo_files/figure-latex/ok1-1.pdf

```
#This plot is drawn where residuals are displayed by connecting observations to corresponding points on
```

When conducting a residual analysis, a "residuals versus fits plot" is frequently created. It is a scatter plot of residuals on the y axis and fitted values on the x axis. The plot is used to detect non-linearity, unequal error variances, and outliers.

```
plot(fitted(fit), resid(fit), pch=19,col="skyblue4",ylab="Residuals", xlab="Fitted values")
abline(0, 0,lty=2,col="green",lwd=2)                    # the horizon
```

bookdown-demo_files/figure-latex/ok2-1.pdf

If we are comparing the above two plots, we can find that any data point that falls directly on the estimated regression line has a residual of 0. Therefore, the residual = 0 line corresponds to the estimated regression line.

Some characteristics of a well-behaved residual vs. fits plot:

(1)The residuals "bounce randomly" around the 0 line. This suggests that the assumption that the relationship is linear is reasonable.

(2) The residuals roughly form a "horizontal band" around the 0 line. This suggests that the variances of the error terms are equal.

(3) No one residual "stands out" from the basic random pattern of residuals. This suggests that there are no outliers.

Note:Interpreting these plots is subjective. It should be with caution to interprete the results based on residual vs. fits plots for small data sets.

Additionally, we can also find out whether residuals come from a normal distribution by checking for a straight line on a Q–Q plot:

```
qqnorm(resid(fit),pch=19)
qqline(resid(fit),col="skyblue4",lwd=2)
```

bookdown-demo_files/figure-latex/ok3-1.pdf

10.1.5 Prediction and confidence intervals

Assume that the error term in the simple linear regression model is independent of x, and is normally distributed, with zero mean and constant variance.

The basic syntax for predict() in linear regression is:

```
predict(object, newdata)
```

Where, object is the formula which is already created using the lm() function, newdata is the vector containing the new value for predictor variable.

If we apply the predict function with no arguments, it just gives the fitted values:

```r
predict(fit)
```

```
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
## 143 123 117 113 111 111 111 110 111 114 113 137 116 114 119 138 118 129
##  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
## 118 113 114 114 115 110 111 111 124 134 111 121 116 138 131 126 112 111
##  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
## 124 116 127 118 110 110 128 111 116 134 121 126 133 112 111 130 122 126
##  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
## 125 126 111 137 114 111 130 110 111 167 131 115 112 120 111 127 110 117
##  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
## 111 134 110 113 138 135 118 113 113 110 114 117 136 113 111 133 120 141
##  91  92  93  94  95  96  97  98  99 100
## 111 118 110 118 115 126 124 125 114 110
```

If we add interval="confidence" or interval="prediction", then we get the vector of predicted values augmented with limits. For a given value of x, the interval estimate of the dependent variable (glucose level) is called the prediction interval.For a given value of x, the interval estimate for the mean of the dependent variable, is called the confidence interval.

```r
predict(fit,interval="predict")
```

```
## Warning in predict.lm(fit, interval = "predict"): predictions on current data refer to _future_ respo
```

```
##     fit   lwr upr
## 1   143  82.2 204
## 2   123  63.9 183
## 3   117  57.3 176
## 4   113  53.4 172
## 5   111  51.4 171
## 6   111  51.4 171
## 7   111  51.4 171
## 8   110  50.4 170
## 9   111  51.4 171
## 10  114  54.4 173
## 11  113  53.4 172
## 12  137  77.2 198
## 13  116  56.3 175
## 14  114  54.4 173
## 15  119  59.2 178
## 16  138  78.0 199
## 17  118  58.2 177
## 18  129  69.3 188
## 19  118  58.2 177
## 20  113  53.4 172
## 21  114  54.4 173
## 22  114  54.4 173
## 23  115  55.3 174
## 24  110  50.4 170
```

```
## 25  111  51.4 171
## 26  111  51.4 171
## 27  124  64.8 184
## 28  134  73.7 193
## 29  111  51.4 171
## 30  121  62.0 181
## 31  116  56.3 175
## 32  138  78.0 199
## 33  131  71.1 190
## 34  126  66.6 186
## 35  112  52.4 171
## 36  111  51.4 171
## 37  124  64.8 184
## 38  116  56.3 175
## 39  127  67.5 187
## 40  118  58.2 177
## 41  110  50.4 170
## 42  110  50.4 170
## 43  128  68.4 187
## 44  111  51.4 171
## 45  116  56.3 175
## 46  134  73.7 193
## 47  121  62.0 181
## 48  126  66.6 186
## 49  133  72.8 192
## 50  112  52.4 171
## 51  111  51.4 171
## 52  130  70.2 189
## 53  122  62.9 182
## 54  126  66.6 186
## 55  125  65.7 185
## 56  126  66.6 186
## 57  111  51.4 171
## 58  137  77.2 198
## 59  114  54.4 173
## 60  111  51.4 171
## 61  130  70.2 189
## 62  110  50.4 170
## 63  111  51.4 171
## 64  167 101.4 232
## 65  131  71.1 190
## 66  115  55.3 174
## 67  112  52.4 171
## 68  120  61.1 180
## 69  111  51.4 171
## 70  127  67.5 187
## 71  110  50.4 170
## 72  117  57.3 176
## 73  111  51.4 171
## 74  134  73.7 193
## 75  110  50.4 170
## 76  113  53.4 172
## 77  138  78.0 199
## 78  135  74.6 194
```

```
## 79  118  58.2 177
## 80  113  53.4 172
## 81  113  53.4 172
## 82  110  50.4 170
## 83  114  54.4 173
## 84  117  57.3 176
## 85  136  76.3 197
## 86  113  53.4 172
## 87  111  51.4 171
## 88  133  72.8 192
## 89  120  61.1 180
## 90  141  80.5 202
## 91  111  51.4 171
## 92  118  58.2 177
## 93  110  50.4 170
## 94  118  58.2 177
## 95  115  55.3 174
## 96  126  66.6 186
## 97  124  64.8 184
## 98  125  65.7 185
## 99  114  54.4 173
## 100 110  50.4 170
```

"fit" denotes the expected values, here identical to the fitted values. lwr and upr are the lower and upper confidence limits for the expected values, respectively, the prediction limits for glucose level for new persons with these values of age. The warning in this case does not really mean that anything is wrong, but there is a pitfall: The limits should not be used for evaluating the observed data to which the line has been fitted. Therefore, let's predict in a new data frame in which the variable wt contains the values at which we want predictions to be made.

```r
library(ggplot2)
#we create a new data frame
newdata<-data.frame(age=seq(20,80,by=5))
##We now apply the predict function and set the predictor variable in the newdata argument. We also set
p1 <- predict(fit, interval="predict", newdata=newdata)
```

```
## Warning: 'newdata' had 13 rows but variables found have 100 rows
```

```r
##we may also want to know the confidence interval.
p2 <- predict(fit, interval="confidence", newdata=newdata)
```

```
## Warning: 'newdata' had 13 rows but variables found have 100 rows
```

```r
plot(sample$age,sample$glu,ylim=range(50,200),pch=19, col="skyblue4") #create a standard scatterplot, a
pred.age <- newdata$age
matlines(sample$age, p2, lty=c(1,2,2), col="black",lwd=2)
matlines(sample$age, p1, lty=c(1,3,3), col="red",lwd=2)
```

bookdown-demo_files/figure-latex/ok6-1.pdf

Please note from the above figure that the black line is the confidence limits (narrow bands) and the red line is the prediction limits (wide bands).The confidence bands reflect the uncertainty about the line itself. If

there are many observations, the bands will be quite narrow, reflecting a well-determined line. These bands often show a marked curvature since the line is better determined near the center of the point cloud. Let's try:

```
x<-data.frame(age=mean(sample$age))
predict(fit,newdata=x)
```

```
## Warning: 'newdata' had 1 row but variables found have 100 rows
```

```
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
## 143 123 117 113 111 111 111 110 111 114 113 137 116 114 119 138 118 129
##  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
## 118 113 114 114 115 110 111 111 124 134 111 121 116 138 131 126 112 111
##  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
## 124 116 127 118 110 110 128 111 116 134 121 126 133 112 111 130 122 126
##  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
## 125 126 111 137 114 111 130 110 111 167 131 115 112 120 111 127 110 117
##  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
## 111 134 110 113 138 135 118 113 113 110 114 117 136 113 111 133 120 141
##  91  92  93  94  95  96  97  98  99 100
## 111 118 110 118 115 126 124 125 114 110
```

```
y<-mean(sample$glu)
print(y)
```

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

This shows that the predicted value at mean value of x will be the mean value of y , whatever the slope is, and hence the standard error of the fitted value at that point is the standard effor of the mean of the independent variable. At other values of x, there will also be a contribution from the variability of the estimated slope, having increasing influence as you move away from the mean value of x.

The prediction bands include the uncertainty about future observations. These bands should capture the majority of the observed points and will not collapse to a line as the number of observations increases. Rather, the limits approach the true line ± 2 standard deviations (for 95% limits).

10.2 Multiple linear regression

In the preceding sections, we are only describing one dependent variable and one independent variable. when we conduct statistical tests, we want to make sure that differences in the estimated parameters are 'real differences' and not a result of a spurious association i.e. due to the confounding variable. This is can be done by multiple linear regression. R provides comprehensive support for multiple linear regression.

Let's conduct a multiple regression analysis:

```
myfit<-lm(sample$glu~sample$bp + sample$bmi + sample$age + sample$type)
#As the simple linear model, we can obtain more outputs with the aid of summary
summary(myfit)
```

```
##
## Call:
## lm(formula = sample$glu ~ sample$bp + sample$bmi + sample$age +
##     sample$type)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -63.50 -17.58  -2.22  14.52  51.59
```

```
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)      78.956     15.246    5.18  1.2e-06 ***
## sample$bp        -0.221      0.208   -1.07    0.289
## sample$bmi        0.889      0.366    2.43    0.017 *
## sample$age        0.485      0.247    1.97    0.052 .
## sample$typeYes   35.739      5.667    6.31  9.0e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 23.4 on 95 degrees of freedom
## Multiple R-squared:  0.464,  Adjusted R-squared:  0.441
## F-statistic: 20.6 on 4 and 95 DF,  p-value: 3.13e-12
```

We can compare nested models with the anova( ) function. We need to ensure that the two models are actually nested.

```
fit1<-lm(glu~bp+bmi+age+type, data = sample)
fit2<-lm(glu~type, data = sample)
summary(fit2)
```

```
##
## Call:
## lm(formula = glu ~ type, data = sample)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -64.61 -17.66  -2.66  15.13  59.34
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   105.66       2.93   36.02  < 2e-16 ***
## typeYes        42.95       5.11    8.41  3.4e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 24 on 98 degrees of freedom
## Multiple R-squared:  0.419,  Adjusted R-squared:  0.413
## F-statistic: 70.7 on 1 and 98 DF,  p-value: 3.35e-13
```

```
anova(fit1,fit2)
```

```
## Analysis of Variance Table
##
## Model 1: glu ~ bp + bmi + age + type
## Model 2: glu ~ type
##   Res.Df   RSS Df Sum of Sq    F Pr(>F)
## 1     95 52142
## 2     98 56507 -3     -4365 2.65  0.053 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#The result returns that  there is no significant improvement of the model once type and bmi are includ
```

Chapter 11

Survival analysis

In this lecture, we will describe the survival analysis in R. The R package named survival is used to carry out survival analysis. The package implements a large number of advanced techniques. It contains the function Surv() which defines a survival object, coxph() which runs a cox proportional hazards regression and survfit() to fit a survival curve to a model or formula. To be specific, we will illustrate how to create the response variable, the Kaplan-Meier estimate, the cumulative hazard, the log-rank test and the regularly used Cox proportional hazards model

```
library(survival)
```

We use the data set lung in package survival.It depicts survival in patients with advanced lung cancer from the North Central Cancer Treatment Group. The variables are listed below:

inst: Institution code time: Survival time in days status: censoring status 1=censored, 2=dead age: Age in years sex: Male=1 Female=2 ph.ecog: ECOG performance score (0=good 5=dead) ph.karno: Karnofsky performance score (bad=0-good=100) rated by physician pat.karno: Karnofsky performance score as rated by patient meal.cal: Calories consumed at meals wt.loss: Weight loss in last six months

```
names(lung)
```

```
##  [1] "inst"      "time"      "status"    "age"       "sex"
##  [6] "ph.ecog"   "ph.karno"  "pat.karno" "meal.cal"  "wt.loss"
```

```
head(lung)
```

```
##   inst time status age sex ph.ecog ph.karno pat.karno meal.cal wt.loss
## 1    3  306      2  74   1       1       90       100     1175      NA
## 2    3  455      2  68   1       0       90        90     1225      15
## 3    3 1010      1  56   1       0       90        90       NA      15
## 4    5  210      2  57   1       1       90        60     1150      11
## 5    1  883      2  60   1       0      100        90       NA       0
## 6   12 1022      1  74   1       1       50        80      513       0
```

```
dim(lung)
```

```
## [1] 228  10
```

```
#Let's recode the status to make 1 for death and 0 for censored
mydata<-lung
mydata$status<-ifelse(lung$status==1,0,1)
#For illustration purpose, we only use the complete cases in the dataset
mydata<-mydata[complete.cases(mydata),]
head(mydata)
```

```
##   inst time status age sex ph.ecog ph.karno pat.karno meal.cal wt.loss
## 2    3  455      1  68   1       0       90        90     1225      15
## 4    5  210      1  57   1       1       90        60     1150      11
## 6   12 1022      0  74   1       1       50        80      513       0
## 7    7  310      1  68   2       2       70        60      384      10
## 8   11  361      1  71   2       2       60        80      538       1
## 9    1  218      1  53   1       1       70        80      825      16
```

```
attach(mydata)
```

```
## The following object is masked _by_ .GlobalEnv:
##
##     age

## The following objects are masked from mydata (pos = 10):
##
##     age, inst, meal.cal, pat.karno, ph.ecog, ph.karno, sex,
##     status, time, wt.loss
```

11.1 Create the response variable

Before complex functions may be performed, the data has to be put into the proper format: a survival object.

```
Surv(time, time2, event, type)
```

Where time indicates start time, time 2 indicates stop time and type indicates whether or not an event occurred. In this case, we do not have the start and stop time. Instead, we can also use the follow-up time as an argument in the function Surv().

```
Surv(time,status)
```

```
##   [1]  455   210  1022+  310   361   218   166   170   567   613   707
##  [12]   61   301    81   371   520   574   118   390    12   473    26
##  [23]  107    53   814   965+   93   731   460   153   433   583    95
##  [34]  303   519   643   765    53   246   689     5   687   345   444
##  [45]  223    60   163    65   821+  428   230   840+  305    11   226
##  [56]  426   705   363   176   791    95   196+  167   806+  284   641
##  [67]  147   740+  163   655    88   245    30   477   559+  450   156
##  [78]  529+  429   351    15   181   283    13   212   524   288   363
##  [89]  199   550    54   558   207    92    60   551+  293   353   267
## [100]  511+  457   337   201   404+  222    62   458+  353   163    31
## [111]  229   156   291   179   376+  384+  268   292+  142   413+  266+
## [122]  320   181   285   301+  348   197   382+  303+  296+  180   145
## [133]  269+  300+  284+  292+  332+  285   259+  110   286   270   225+
## [144]  269   225+  243+  276+  135    79    59   240+  202+  235+  239
## [155]  252+  221+  185+  222+  183   211+  175+  197+  203+  191+  105+
## [166]  174+  177+
```

We can find that the time is put into a survival format, with "+" indicating right censoring data.

11.2 Kaplan-Meier estimate

The Kaplan–Meier estimator can be used to estimate the survival function in the presence of right censoring. The function survfit() is used to find the Kaplan-Meier estimate of the survival function. There are three arguments: formula, conf.int, and conf.type. "formula" refers to a survival object , and it is the only required input.

```
fit1<-survfit(Surv(time,status)~1)#The formula Surv(time,status)~1 instructs the survfit() function to
print(fit1)
```

```
## Call: survfit(formula = Surv(time, status) ~ 1)
##
##       n  events  median 0.95LCL 0.95UCL
##     167     120     310     285     371
```

```
##The output is not very informative. We can see a couple of summary statistics and an estimate of the

#To see the actual Kaplan-Meier estimate, we can use summary() function.
summary (fit1)#for simplicity, we only show the first 10 results for summary().
```

```
## Call: survfit(formula = Surv(time, status) ~ 1)
##
##  time n.risk n.event survival std.err lower 95% CI upper 95% CI
##     5    167       1   0.9940 0.00597       0.9824        1.000
##    11    166       1   0.9880 0.00842       0.9717        1.000
##    12    165       1   0.9820 0.01028       0.9621        1.000
##    13    164       1   0.9760 0.01183       0.9531        1.000
##    15    163       1   0.9701 0.01319       0.9446        0.996
##    26    162       1   0.9641 0.01440       0.9363        0.993
##    30    161       1   0.9581 0.01551       0.9282        0.989
##    31    160       1   0.9521 0.01653       0.9203        0.985
##    53    159       2   0.9401 0.01836       0.9048        0.977
##    54    157       1   0.9341 0.01919       0.8973        0.973
##    59    156       1   0.9281 0.01998       0.8898        0.968
##    60    155       2   0.9162 0.02145       0.8751        0.959
##    61    153       1   0.9102 0.02213       0.8678        0.955
##    62    152       1   0.9042 0.02278       0.8606        0.950
##    65    151       1   0.8982 0.02340       0.8535        0.945
##    79    150       1   0.8922 0.02400       0.8464        0.941
##    81    149       1   0.8862 0.02457       0.8394        0.936
##    88    148       1   0.8802 0.02512       0.8323        0.931
##    92    147       1   0.8743 0.02566       0.8254        0.926
##    93    146       1   0.8683 0.02617       0.8185        0.921
##    95    145       2   0.8563 0.02715       0.8047        0.911
##   107    142       1   0.8503 0.02762       0.7978        0.906
##   110    141       1   0.8442 0.02807       0.7910        0.901
##   118    140       1   0.8382 0.02851       0.7841        0.896
##   135    139       1   0.8322 0.02894       0.7773        0.891
##   142    138       1   0.8261 0.02935       0.7706        0.886
##   145    137       1   0.8201 0.02975       0.7638        0.881
##   147    136       1   0.8141 0.03013       0.7571        0.875
##   153    135       1   0.8080 0.03051       0.7504        0.870
##   156    134       2   0.7960 0.03122       0.7371        0.860
##   163    132       3   0.7779 0.03221       0.7173        0.844
##   166    129       1   0.7719 0.03252       0.7107        0.838
```

```
##   167    128       1   0.7658 0.03282       0.7041        0.833
##   170    127       1   0.7598 0.03311       0.6976        0.828
##   176    124       1   0.7537 0.03341       0.6910        0.822
##   179    122       1   0.7475 0.03370       0.6843        0.817
##   180    121       1   0.7413 0.03398       0.6776        0.811
##   181    120       2   0.7290 0.03452       0.6644        0.800
##   183    118       1   0.7228 0.03478       0.6577        0.794
##   197    114       1   0.7164 0.03505       0.6510        0.789
##   199    112       1   0.7101 0.03531       0.6441        0.783
##   201    111       1   0.7037 0.03557       0.6373        0.777
##   207    108       1   0.6971 0.03583       0.6303        0.771
##   210    107       1   0.6906 0.03608       0.6234        0.765
##   212    105       1   0.6840 0.03633       0.6164        0.759
##   218    104       1   0.6775 0.03658       0.6094        0.753
##   222    102       1   0.6708 0.03681       0.6024        0.747
##   223    100       1   0.6641 0.03705       0.5953        0.741
##   226     97       1   0.6573 0.03730       0.5881        0.735
##   229     96       1   0.6504 0.03753       0.5809        0.728
##   230     95       1   0.6436 0.03776       0.5737        0.722
##   239     93       1   0.6367 0.03798       0.5664        0.716
##   245     90       1   0.6296 0.03821       0.5590        0.709
##   246     89       1   0.6225 0.03843       0.5516        0.703
##   267     85       1   0.6152 0.03867       0.5439        0.696
##   268     84       1   0.6079 0.03890       0.5362        0.689
##   269     83       1   0.6005 0.03911       0.5286        0.682
##   270     81       1   0.5931 0.03933       0.5208        0.675
##   283     79       1   0.5856 0.03954       0.5130        0.668
##   284     78       1   0.5781 0.03974       0.5052        0.661
##   285     76       2   0.5629 0.04012       0.4895        0.647
##   286     74       1   0.5553 0.04029       0.4817        0.640
##   288     73       1   0.5477 0.04045       0.4739        0.633
##   291     72       1   0.5401 0.04060       0.4661        0.626
##   293     69       1   0.5322 0.04076       0.4581        0.618
##   301     66       1   0.5242 0.04093       0.4498        0.611
##   303     64       1   0.5160 0.04110       0.4414        0.603
##   305     62       1   0.5077 0.04127       0.4329        0.595
##   310     61       1   0.4993 0.04143       0.4244        0.588
##   320     60       1   0.4910 0.04157       0.4160        0.580
##   337     58       1   0.4826 0.04170       0.4074        0.572
##   345     57       1   0.4741 0.04182       0.3988        0.564
##   348     56       1   0.4656 0.04192       0.3903        0.555
##   351     55       1   0.4572 0.04201       0.3818        0.547
##   353     54       2   0.4402 0.04212       0.3650        0.531
##   361     52       1   0.4318 0.04215       0.3566        0.523
##   363     51       2   0.4148 0.04217       0.3399        0.506
##   371     49       1   0.4064 0.04215       0.3316        0.498
##   390     45       1   0.3973 0.04217       0.3227        0.489
##   426     42       1   0.3879 0.04221       0.3134        0.480
##   428     41       1   0.3784 0.04223       0.3041        0.471
##   429     40       1   0.3690 0.04222       0.2948        0.462
##   433     39       1   0.3595 0.04218       0.2856        0.452
##   444     38       1   0.3500 0.04212       0.2765        0.443
##   450     37       1   0.3406 0.04203       0.2674        0.434
##   455     36       1   0.3311 0.04192       0.2584        0.424
```

```
##   457     35       1   0.3217 0.04177       0.2494        0.415
##   460     33       1   0.3119 0.04163       0.2401        0.405
##   473     32       1   0.3022 0.04145       0.2309        0.395
##   477     31       1   0.2924 0.04124       0.2218        0.386
##   519     29       1   0.2823 0.04104       0.2123        0.375
##   520     28       1   0.2722 0.04079       0.2030        0.365
##   524     27       1   0.2622 0.04051       0.1937        0.355
##   550     25       1   0.2517 0.04022       0.1840        0.344
##   558     23       1   0.2407 0.03993       0.1739        0.333
##   567     21       1   0.2293 0.03964       0.1634        0.322
##   574     20       1   0.2178 0.03928       0.1529        0.310
##   583     19       1   0.2063 0.03885       0.1427        0.298
##   613     18       1   0.1949 0.03835       0.1325        0.287
##   641     17       1   0.1834 0.03777       0.1225        0.275
##   643     16       1   0.1720 0.03711       0.1126        0.262
##   655     15       1   0.1605 0.03636       0.1029        0.250
##   687     14       1   0.1490 0.03552       0.0934        0.238
##   689     13       1   0.1376 0.03459       0.0840        0.225
##   705     12       1   0.1261 0.03355       0.0749        0.212
##   707     11       1   0.1146 0.03240       0.0659        0.199
##   731     10       1   0.1032 0.03112       0.0571        0.186
##   765      8       1   0.0903 0.02979       0.0473        0.172
##   791      7       1   0.0774 0.02818       0.0379        0.158
##   814      5       1   0.0619 0.02646       0.0268        0.143
```

This returns the Kaplan-Meier estimator and its estimated std, and the 95% confidence interval using the log transform. Sometimes, we would be more interested in showing the Kaplan–Meier estimate graphically than numerically.

```
plot(fit1,main = 'Kaplan Meier Plot',xlab="Follow-up time",ylab="Survival Probability")
```

bookdown-demo_files/figure-latex/unnamed-chunk-237-1.pdf

```
#If we look closely, we will see that the bands are not symmetrical around the estimate.This is because

plot(fit1,conf.int=F,main = 'Kaplan Meier Plot',xlab="Follow-up time",ylab="Survival Probability")
```

bookdown-demo_files/figure-latex/unnamed-chunk-237-2.pdf

```
#If we want a 99% confidence interval
fit2<-survfit(Surv(time,status)~1,conf.int=0.99)
plot(fit2,main = 'Kaplan Meier Plot with 99% Confidence Interval',xlab="Follow-up time",ylab="Survival
```

bookdown-demo_files/figure-latex/unnamed-chunk-237-3.pdf

Notice that the confidence intervals constructed above are only pointwise confidence intervals. Confidence bands, which are bounds on an entire range of time, are a bit more generalized. That is, for a 95% confidence band, the probability that any part of the true curve is out of the confidence bands is 0.05. The confBands() function from the OIsurv package is employed.

```
install.packages("OIsurv")
```

```
library(OIsurv)
cb <- confBands(Surv(time,status), type = "hall")
plot(fit1,main = 'Kaplan Meyer Plot with confidence bands',xlab="Follow-up time",ylab="Survival Probabil
lines(cb, col = "red",lty =3)
legend(500, 0.99, legend = c('K-M survival estimate',
'pointwise intervals', 'Hall-Werner conf bands'), col=c("blue","blue","red"),lty = 1:3)
```

bookdown-demo_files/figure-latex/unnamed-chunk-238-1.pdf

Often, we wish to plot two or more survival functions on the same plot so that we can compare them directly. For instance, if we want to obtain survival functions split by sex, do the following:

```
surv.bysex <- survfit(Surv(time,status)~sex)
plot(surv.bysex,lty=c(1,2),xlab="Follow-up time",ylab="Survival Probability",main="Comparing Survival fu
legend(700,0.6,legend=c("male","female"),lty=c(1,2))
```

bookdown-demo_files/figure-latex/unnamed-chunk-239-1.pdf

```
#It indicates that the females have a higher survival probability than the males at each time point dur
```

Notice that there are no confidence intervals on the curves. By default, they are not shown when there are more than two curves to avoid confusing display. However, we can show them by passing conf.int=T to plot.

```
surv.bysex <- survfit(Surv(time,status)~sex)
plot(surv.bysex,conf.int=T,col=c("red","green"),lty=c(1,2))
legend(700,0.6,legend=c("male","female"),col=c("red","green"),lty=c(1,2))
```

bookdown-demo_files/figure-latex/unnamed-chunk-240-1.pdf

11.3 Cumulative hazard

To obtain the cumulative hazard:

```
fit1<-survfit(Surv(time,status)~1)
myfit<-summary (fit1)
H.hat <- -log(myfit$surv)
H.hat <- c(H.hat, H.hat[length(H.hat)])
plot(c(myfit$time, 1000), H.hat, xlab="Follow-up Time", ylab="Cumulative Hazard",main="Cumulative Hazar
```

bookdown-demo_files/figure-latex/unnamed-chunk-241-1.pdf

11.4 The log-rank test

The log-rank test is used to test whether there is a difference between the survival times between two or more samples. It is based on looking at the population at each death time and computing the expected number of deaths in proportion to the number of individuals at risk in each group. This is then summed over all death times and compared with the observed number of deaths by a procedure similar to the 2 test. This is done by:

```
survdiff(formula, rho=0)
```

The first argument is a survival object against a categorical covariate variable that is typically a variable designating which groups correspond to which survival times. The second argument designates the weights. The default is rho=0, which corresponds to the log-rank test. When rho=1, this is the "Peto & Peto modification of the Gehan-Wilcoxon test".

```
survdiff(Surv(time,status)~sex)
```

```
## Call:
## survdiff(formula = Surv(time, status) ~ sex)
##
##         N Observed Expected (O-E)^2/E (O-E)^2/V
## sex=1 103       82     68.7      2.57      6.05
## sex=2  64       38     51.3      3.44      6.05
##
##  Chisq= 6  on 1 degrees of freedom, p= 0.0139
```

```
#This shows that there is a significantly statistical difference between survival time for male and fem
```

11.5 Cox Proportional Hazards Model

The proportional hazards model allows the analysis of survival data by regression models similar to those of lm and glm. The scale on which linearity is assumed is the log-hazard scale. The function coxph() fits a Cox PH model to the supplied data. The two arguments of particular interest are formula and method. formula will be almost identical to fitting a linear model except that the response variable will be a survival object instead of a vector.

11.5.1 Regular Cox proportional hazards model

Let's first consider a model with single regressor sex:

```
mycox<-coxph(Surv(time,status)~sex)
summary(mycox)
```

```
## Call:
## coxph(formula = Surv(time, status) ~ sex)
##
##   n= 167, number of events= 120
##
##       coef exp(coef) se(coef)     z Pr(>|z|)
## sex -0.479     0.619    0.197 -2.44    0.015 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##     exp(coef) exp(-coef) lower .95 upper .95
## sex     0.619       1.61     0.421      0.91
##
## Concordance= 0.567  (se = 0.026 )
## Rsquare= 0.037   (max possible= 0.998 )
## Likelihood ratio test= 6.25  on 1 df,   p=0.0124
## Wald test            = 5.94  on 1 df,   p=0.0148
## Score (logrank) test = 6.05  on 1 df,   p=0.0139
```

```
mycox$coefficients
```

```
##    sex
## -0.479
```

```
mycox$var #  estimated cov matrix of the estimates
```

```
##        [,1]
## [1,] 0.0387
```

The coef is the estimated logarithm of the hazard ratio between the two groups, the exp(coef) denotes the hazard ratio. Next it also shows the standard errors, z values and p-values for each test. The line following that gives the inverted ratio and confidence intervals for the hazard ratio. Finally, three overall tests for significant effects in the model are given. Notice that the Wald test is identical to the z test based on the estimated coefficient divided by its standard error, whereas the score test is equivalent to the log-rank test when the model involves only a simple grouping.

```
#The default method to handle ties is Efron, we can also choose to use Breslow:
summary(coxph(Surv(time,status)~sex, method="breslow"))
```

```
## Call:
## coxph(formula = Surv(time, status) ~ sex, method = "breslow")
##
##   n= 167, number of events= 120
##
##       coef exp(coef) se(coef)     z Pr(>|z|)
## sex -0.478     0.620    0.197 -2.43    0.015 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##     exp(coef) exp(-coef) lower .95 upper .95
```

```
## sex      0.62       1.61     0.422     0.911
##
## Concordance= 0.567  (se = 0.026 )
## Rsquare= 0.037   (max possible= 0.998 )
## Likelihood ratio test= 6.23  on 1 df,   p=0.0126
## Wald test            = 5.92  on 1 df,   p=0.015
## Score (logrank) test = 6.03  on 1 df,   p=0.014
```

To obtain the baseline survival function from a Cox PH model, apply survfit() to coxph():

```
my.survfit.object <- survfit(mycox)
#Plot the baseline survival function
plot(my.survfit.object,xlab="Follow-up time",ylab = "Proportion survived",
     main = "Baseline Survival Curve")
```

bookdown-demo_files/figure-latex/unnamed-chunk-245-1.pdf

Let's next fit a more coxplex model:

```
names(mydata)
```

```
##  [1] "inst"      "time"      "status"    "age"       "sex"
##  [6] "ph.ecog"   "ph.karno"  "pat.karno" "meal.cal"  "wt.loss"
```

```
mydata$ph.ecog<-factor(mydata$ph.ecog)
mycox.2<-coxph(Surv(time,status)~sex+age+ph.ecog+ph.karno+meal.cal+wt.loss, data = mydata)
summary(mycox.2)
```

```
## Call:
## coxph(formula = Surv(time, status) ~ sex + age + ph.ecog + ph.karno +
##     meal.cal + wt.loss, data = mydata)
##
##   n= 167, number of events= 120
##
##               coef exp(coef)  se(coef)     z Pr(>|z|)
## sex      -5.63e-01  5.70e-01  2.02e-01 -2.79  0.00533 **
## age       9.85e-03  1.01e+00  1.17e-02  0.84  0.39909
## ph.ecog1  6.64e-01  1.94e+00  2.81e-01  2.36  0.01812 *
## ph.ecog2  1.65e+00  5.20e+00  4.40e-01  3.75  0.00018 ***
## ph.ecog3  2.85e+00  1.73e+01  1.12e+00  2.54  0.01099 *
## ph.karno  2.03e-02  1.02e+00  1.12e-02  1.80  0.07109 .
## meal.cal -2.48e-05  1.00e+00  2.59e-04 -0.10  0.92388
## wt.loss  -1.24e-02  9.88e-01  7.76e-03 -1.60  0.10942
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##          exp(coef) exp(-coef) lower .95 upper .95
## sex          0.570     1.7555     0.383     0.846
## age          1.010     0.9902     0.987     1.033
## ph.ecog1     1.942     0.5150     1.120     3.366
## ph.ecog2     5.201     0.1923     2.197    12.313
## ph.ecog3    17.293     0.0578     1.922   155.586
```

```
## ph.karno     1.020     0.9799     0.998     1.043
## meal.cal     1.000     1.0000     0.999     1.000
## wt.loss      0.988     1.0125     0.973     1.003
##
## Concordance= 0.641  (se = 0.031 )
## Rsquare= 0.149   (max possible= 0.998 )
## Likelihood ratio test= 26.9  on 8 df,   p=0.000749
## Wald test            = 27.4  on 8 df,   p=0.000611
## Score (logrank) test = 29.3  on 8 df,   p=0.000276
```

```
#We can find that only sex, ph.ecog are significantly meaningful variables.
mycox.3<-coxph(Surv(time,status)~sex+ph.ecog, data = mydata)
summary(mycox.3)
```

```
## Call:
## coxph(formula = Surv(time, status) ~ sex + ph.ecog, data = mydata)
##
##   n= 167, number of events= 120
##
##            coef exp(coef) se(coef)     z Pr(>|z|)
## sex      -0.500     0.607    0.197 -2.53  0.01135 *
## ph.ecog1  0.321     1.378    0.233  1.38  0.16898
## ph.ecog2  0.919     2.506    0.261  3.52  0.00043 ***
## ph.ecog3  1.997     7.369    1.036  1.93  0.05379 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##          exp(coef) exp(-coef) lower .95 upper .95
## sex          0.607      1.648     0.412     0.893
## ph.ecog1     1.378      0.726     0.873     2.176
## ph.ecog2     2.506      0.399     1.503     4.178
## ph.ecog3     7.369      0.136     0.968    56.106
##
## Concordance= 0.646  (se = 0.03 )
## Rsquare= 0.115   (max possible= 0.998 )
## Likelihood ratio test= 20.4  on 4 df,   p=0.000418
## Wald test            = 21.9  on 4 df,   p=0.000214
## Score (logrank) test = 23.5  on 4 df,   p=9.95e-05
```

```
#Let's see whether the above two models make a lot difference
anova(mycox.2,mycox.3)
```

```
## Analysis of Deviance Table
##  Cox model: response is  Surv(time, status)
##  Model 1: ~ sex + age + ph.ecog + ph.karno + meal.cal + wt.loss
##  Model 2: ~ sex + ph.ecog
##   loglik Chisq Df P(>|Chi|)
## 1   -495
## 2   -498  6.46  4      0.17
```

```
#it shows that model 2(the full model) is not significantly better than model 3(the reduced model)
plot(survfit(mycox.3),main="Baseline Survival Curve with 95% Confidence Interval")
```

bookdown-demo_files/figure-latex/test1-1.pdf

Let's plot the effect of sex:

```
#we need to first build a dataframe for the effect levels we want to look at holding other covariates c
#new<- data.frame(sex = 1:2,ph.ecog = rep(levels(ph.ecog)[1],2))
new <- data.frame(sex = 1:2,ph.ecog = c(0,0))
print(new)
```

```
##   sex ph.ecog
## 1   1       0
## 2   2       0
```

```
new$ph.ecog <- as.factor(new$ph.ecog)
str(new)
```

```
## 'data.frame':    2 obs. of  2 variables:
##  $ sex    : int  1 2
##  $ ph.ecog: Factor w/ 1 level "0": 1 1
```

```
#plot the effects
plot(survfit(mycox.3, newdata = new),conf.int = TRUE,xlab = "Follow-up time",ylab = "Proportion survive
legend(120, 0.4, legend = c("female", "male"), lty = 1,col=c("green","red"))
```

bookdown-demo_files/figure-latex/unnamed-chunk-246-1.pdf

11.5.2 Test for proportional hazards

When we are using the Cox proportional hazards model, we are assuming that the hazards are proportional.To test this, we can use:

```
cox.zph(mycox.3)
```

```
##              rho  chisq      p
## sex       0.1160 1.5517 0.2129
## ph.ecog1 -0.0560 0.3702 0.5429
## ph.ecog2 -0.2027 4.7314 0.0296
## ph.ecog3  0.0107 0.0138 0.9066
## GLOBAL        NA 7.0279 0.1344
```

```
## Note this is very conservative!
```

It is often better to check the graphs for systematic trends:

```
#To view easily, let's show them on two plots.
a <- cox.zph(mycox.3)
par(mfrow = c(1, 2))
```

```r
plot(a[1], main = "Sex")
plot(a[2], main = "ECOG performance score=1")
```

bookdown-demo_files/figure-latex/unnamed-chunk-248-1.pdf

```r
par(mfrow = c(1, 2))
plot(a[3], main = "ECOG performance score=2")
plot(a[4], main = "ECOG performance score=3")
```

bookdown-demo_files/figure-latex/unnamed-chunk-248-2.pdf

As we can see in above results, all of the variables seem to satisfy the proportional hazards assumption. To explain the non-proportional hazards, let's hypothetically assume that the variable sex cannot meet the proportional hazards assumption. There are a number of approaches to account for the non-proportional hazards.

```r
#Option 1: Include an interaction with time for the variables
mycox.4<-coxph(Surv(time,status)~sex+ph.ecog+sex*time)
```

```
## Warning in coxph(Surv(time, status) ~ sex + ph.ecog + sex * time): a
## variable appears on both the left and right sides of the formula
```

```
## Warning in fitter(X, Y, strats, offset, init, control, weights = weights, :
## Loglik converged before variable 1,2,3,4 ; beta may be infinite.
```

```r
summary(mycox.4)
```

```
## Call:
## coxph(formula = Surv(time, status) ~ sex + ph.ecog + sex * time)
##
##   n= 167, number of events= 120
##
##              coef exp(coef) se(coef)     z Pr(>|z|)
## sex      -0.86873   0.41948  0.82263 -1.06     0.29
## ph.ecog   0.26011   1.29707  0.26058  1.00     0.32
## time     -0.99957   0.36804  0.15688 -6.37  1.9e-10 ***
## sex:time  0.00201   1.00202  0.00263  0.77     0.44
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##          exp(coef) exp(-coef) lower .95 upper .95
## sex          0.419      2.384    0.0837     2.103
## ph.ecog      1.297      0.771    0.7783     2.162
## time         0.368      2.717    0.2706     0.501
## sex:time     1.002      0.998    0.9969     1.007
##
## Concordance= 1  (se = 0.031 )
```

```
## Rsquare= 0.997   (max possible= 0.998 )
## Likelihood ratio test= 961  on 4 df,   p=0
## Wald test            = 41.8  on 4 df,   p=1.85e-08
## Score (logrank) test = 190  on 4 df,   p=0
```

```
#As actually sex is proportional, the interaction term returns a p value>0.05.
#It only makes sense when they have a linear interaction between the covariate and time. We can also tr

#Option 2: Stratified by the variate
mycox.5<-coxph(Surv(time,status)~ph.ecog+strata(sex))
summary(mycox.5)
```

```
## Call:
## coxph(formula = Surv(time, status) ~ ph.ecog + strata(sex))
##
##   n= 167, number of events= 120
##
##          coef exp(coef) se(coef)    z Pr(>|z|)
## ph.ecog 0.483     1.621    0.134 3.61  0.00031 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##         exp(coef) exp(-coef) lower .95 upper .95
## ph.ecog      1.62      0.617      1.25      2.11
##
## Concordance= 0.622  (se = 0.038 )
## Rsquare= 0.075   (max possible= 0.994 )
## Likelihood ratio test= 13  on 1 df,   p=0.000312
## Wald test            = 13  on 1 df,   p=0.000311
## Score (logrank) test = 13.2  on 1 df,   p=0.00028
```

```
#Each strata has a different baseline hazard function but the remaining covariates are assumed to be co

#However, we can plot it
plot(survfit(mycox.5), col = 1:2,main="Survival Probability stratified by sex")
legend(700, 0.6, legend = c("male","female"), lty = 1, col = 1:2)
```

bookdown-demo_files/figure-latex/unnamed-chunk-249-1.pdf

There are many advanced methods we can use to handle proportional hazards assumption. We will not explain here in detail.

Finally, let's plot the survival curves computed for Kaplan-Meier and cox proportional hazards models (model on the same graph.

```
fit1<-survfit(Surv(time,status)~1)
km <- rep("KM", length(fit1$time))
km_df <- data.frame(fit1$time,fit1$surv,km)
names(km_df) <- c("Time","Surv","Model")

fit2<-survfit(mycox.2)
Coxfullmodel <- rep("Coxfullmodel",length(fit2$time))
```

```
cox_df.full <- data.frame(fit2$time,fit2$surv,Coxfullmodel)
names(cox_df.full) <- c("Time","Surv","Model")

fit3<-survfit(mycox.3)
Coxreducedmodel <- rep("Coxreducedmodel",length(fit3$time))
cox_df.reduced <- data.frame(fit3$time,fit3$surv,Coxreducedmodel)
names(cox_df.reduced) <- c("Time","Surv","Model")
plot_df <- rbind(km_df,cox_df.full,cox_df.reduced)

library(ggplot2)
p <- ggplot(plot_df, aes(x = Time, y = Surv, color = Model))
p + geom_line() + ggtitle("Comparison of Survival Curves")
```

bookdown-demo_files/figure-latex/unnamed-chunk-250-1.pdf

We may see few differences between the three survival curves.

Chapter 12

Bootstrap, analysis of clustered data and non-linear effects

12.1 Bootstrap

Bootstrapping is a nonparametric method which lets us compute estimated standard errors, confidence intervals and hypothesis testing.It is very useful in statistics and can be easily implemented in R.

```
install.packages("boot")
```

```
library(boot)
```

Let's use the dataset city from the boot package as an example oto understand bootstrap.

```
city
```

```
##      u   x
## 1  138 143
## 2   93 104
## 3   61  69
## 4  179 260
## 5   48  75
## 6   37  63
## 7   29  50
## 8   23  48
## 9   30 111
## 10   2  50
```

Let's say we want to know the correlation between u and x. Let's first define a function that will return the statistics that we would like to bootstrap.

```
mycor<-function(d, i) {#The first argument of the function is the dataset.  The second argument can be
  d0<-d[i,]
return( cor(d0$u,d0$x) ) #Here, the statistic of interest  is the mean of the variable
}
```

We can do the bootstrap by using loops.

```
n = dim(city)[1]
B = 1000
result = rep(NA, B)
```

```
for (i in 1:B) {
boot.sample = sample(n, replace = TRUE)
result[i] = mycor(city[boot.sample,])
}
result[1:10]
```

```
##  [1] 0.994 0.940 0.940 0.964 0.692 0.960 0.946 0.931 0.885 0.956
```

```
#To get the 90% confidence interval
quantile(result,c(0.05,0.95))
```

```
##    5%   95%
## 0.598 0.984
```

```
#To get the 95% confidence interval
quantile(result,c(0.025,0.975))
```

```
##  2.5% 97.5%
## 0.463 0.990
```

```
#To get the 99% confidence interval
quantile(result,c(0.005,0.995))
```

```
##  0.5% 99.5%
## 0.192 0.996
```

There is a package boot with a function boot() that does the bootstrap for many situations. We can use the boot command which executes the resampling of the dataset and calculation of the statistics of interest on these samples. Before calling boot, we also need to define a function first. Next, we can use the boot command, providing our dataset name, our function, and the number of bootstrap samples to be drawn.

```
myboot<- boot(city, mycor, 1000)#the three arguments: the data from the original sample; a function to
myboot
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = city, statistic = mycor, R = 1000)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*    0.904 -0.0349       0.149
```

```
#The object u.boot is a list with many elements.
summary(myboot)
```

```
##      R original bootBias bootSE bootMed
## 1 1000    0.904  -0.0349  0.149   0.918
```

```
myboot$t0#the sample mean of the original data.
```

```
## [1] 0.904
```

```
#To get the bias
mean(myboot$t)-myboot$t0
```

```
## [1] -0.0349
```

```r
#To get the standard error
sd(myboot$t)
```

```
## [1] 0.149
```

We can also use the built-in function boot.ci() to calculate bootstrap confidence intervals using multiple methods.

```r
boot.ci(myboot)
```

```
## Warning in boot.ci(myboot): bootstrap variances needed for studentized
## intervals
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = myboot)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 0.647,  1.231 )   ( 0.818,  1.420 )
##
## Level     Percentile            BCa
## 95%   ( 0.388,  0.990 )   ( 0.128,  0.976 )
## Calculations and Intervals on Original Scale
## Some BCa intervals may be unstable
```

```r
#Basic uses the estimated standard error. Percentile uses percentiles. BCa also uses percentiles, but a
quantile(myboot$t,c(0.025,0.975))
```

```
##  2.5% 97.5%
## 0.389 0.990
```

To look at the histogram and normal quantile-quantile plot of the bootstrap estimates, we can use plot with the "boot" object we created. The histogram includes a dotted vertical line indicating the location of the original statistic.

```r
plot(myboot)
```

bookdown-demo_files/figure-latex/unnamed-chunk-258-1.pdf

12.2 Analysis of clustered data

In many situations we may come across data that include repeated measurements of individuals or measurements of subjects that are expected to be not entirely independent (e.g. family members, individuals living in communities).Clustering can introduce severe problems for statistical inference, in particular in regard to estimated standard errors and associated p-values and confidence intervals.Let's use the smoking dataset.

```r
mydata<-read.table("data/smokingw.txt",header=T)
head(mydata)
```

```
##   id currsmoker male age fev0 fev3 fev6 fev9
## 1  1          0    1  29  3.4 3.40 3.45 3.20
## 2  2          1    1  34  3.1 3.15 3.50 2.95
## 3  3          1    0  19  3.6 3.45 3.45 3.10
## 4  5          1    0  17  3.4 3.30 2.93 2.30
## 5  6          0    0  17  3.3 3.75 3.50 2.95
## 6  8          1    0  36  3.9 4.00 4.05 3.75
```

```
NROW(unique(mydata[,1]))
```

```
## [1] 78
```

```
names(mydata)
```

```
## [1] "id"         "currsmoker" "male"       "age"        "fev0"
## [6] "fev3"       "fev6"       "fev9"
```

```
#Let's plot it
pairs(mydata[,5:8])
```

bookdown-demo_files/figure-latex/unnamed-chunk-259-1.pdf

We may find that different measurements of the FEV are related. In order to conduct the following analysis, we need to reshape the data.

```
smoking.long<-reshape(mydata,varying=c("fev0","fev3","fev6","fev9"),v.names="fev",timevar="year",times=c
smoking.long<-smoking.long[order(smoking.long$id),]
head(smoking.long)
```

```
##     id currsmoker male age year  fev
## 1    1          0    1  29    0 3.40
## 79   1          0    1  29    1 3.40
## 157  1          0    1  29    2 3.45
## 235  1          0    1  29    3 3.20
## 2    2          1    1  34    0 3.10
## 80   2          1    1  34    1 3.15
```

Let's first try a naive model without consideration of the correlation.

```
model1<-lm(fev~currsmoker+male+age, data = smoking.long)
summary(model1)
```

```
##
## Call:
## lm(formula = fev ~ currsmoker + male + age, data = smoking.long)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.5405 -0.3313  0.0086  0.3524  1.6190
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.52803    0.12278   28.73  < 2e-16 ***
## currsmoker  -0.28875    0.07621   -3.79  0.00018 ***
## male         0.13998    0.06490    2.16  0.03178 *
```

```
## age         -0.00661    0.00394   -1.68  0.09447 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.545 on 308 degrees of freedom
## Multiple R-squared:  0.0632, Adjusted R-squared:  0.0541
## F-statistic: 6.93 on 3 and 308 DF,  p-value: 0.000159
```

```r
AIC(model1)
```

```
## [1] 512
```

However, this model assumes independence. We therefore want to account for the repeated measurement nature of the data. We can do this by fitting a linear mixed effects model.

12.2.1 Linear Mixed Effects Models

```
install.packages("lme4")
```

Let's first fit a random intercept model:

```r
library(nlme)
model2=lme(fev~male+age+currsmoker+year,random=~1|id,data=smoking.long)
summary(model2)
```

```
## Linear mixed-effects model fit by REML
##  Data: smoking.long
##   AIC BIC logLik
##   179 205  -82.3
##
## Random effects:
##  Formula: ~1 | id
##         (Intercept) Residual
## StdDev:       0.497    0.205
##
## Fixed effects: fev ~ male + age + currsmoker + year
##             Value Std.Error  DF t-value p-value
## (Intercept)  3.70    0.2293 233   16.15  0.0000
## male         0.14    0.1209  74    1.16  0.2508
## age         -0.01    0.0073  74   -0.90  0.3710
## currsmoker  -0.29    0.1420  74   -2.03  0.0456
## year        -0.12    0.0104 233  -11.28  0.0000
##  Correlation:
##            (Intr) male   age    crrsmk
## male       -0.122
## age        -0.805 -0.150
## currsmoker -0.326 -0.135 -0.126
## year       -0.068  0.000  0.000  0.000
##
## Standardized Within-Group Residuals:
##       Min        Q1       Med        Q3       Max
## -3.896824 -0.478778 -0.000248  0.559734  3.385840
##
## Number of Observations: 312
## Number of Groups: 78
```

Let's next fit a random intercept and random slope model:

```
model3=lme(fev~male+age+currsmoker+year,random=~1+year|id,data=smoking.long)
summary(model3)
```

```
## Linear mixed-effects model fit by REML
##  Data: smoking.long
##   AIC BIC logLik
##   179 212  -80.3
##
## Random effects:
##  Formula: ~1 + year | id
##  Structure: General positive-definite, Log-Cholesky parametrization
##             StdDev Corr
## (Intercept) 0.514  (Intr)
## year        0.058  -0.257
## Residual    0.191
##
## Fixed effects: fev ~ male + age + currsmoker + year
##             Value Std.Error  DF t-value p-value
## (Intercept)  3.71    0.2294 233   16.15  0.0000
## male         0.13    0.1208  74    1.09  0.2771
## age         -0.01    0.0073  74   -0.87  0.3896
## currsmoker  -0.30    0.1419  74   -2.09  0.0405
## year        -0.12    0.0117 233  -10.01  0.0000
##  Correlation:
##            (Intr) male   age    crrsmk
## male       -0.122
## age        -0.803 -0.150
## currsmoker -0.325 -0.135 -0.126
## year       -0.089  0.000  0.000  0.000
##
## Standardized Within-Group Residuals:
##     Min      Q1     Med      Q3     Max
## -3.4638 -0.4646 -0.0242  0.5180  2.8970
##
## Number of Observations: 312
## Number of Groups: 78
```

We can see that for model1, the AIC is 512.1198; For model 2, the AIC is 178.5472; For model 3, the AIC is 178.5033.Generally, the smaller the AIC, the better the model. Model 2 and model 3 are much better than model 1. But model 3 is not so much better than model 2. We can also do a Chi-square test to compare the goodness of fit for model 2 and model3.

```
anova(model2,model3)
```

```
##        Model df AIC BIC logLik   Test L.Ratio p-value
## model2     1  7 178 205  -82.3
## model3     2  9 178 212  -80.3 1 vs 2    4.04   0.132
```

```
#We can also find that model 3 is not statistically significant better than model 2.
```

We can also see that it doesn't make much sense to include a random slope since the average FEV over time is varied by person but the FEV trajectory (the slope) is assumed to be homogeneous.

12.2.2 Generalized Estimation Equations (GEE) models

By using the linear mixed effects model, we are estimating the conditional effects (individual or cluster-specific effects). However, if we are interested in the marginal effects (average population effects), we may want to use the GEE model.

```
library(geepack)
model4<-geeglm(fev~male+age+currsmoker+year,id=id, data=smoking.long)
summary(model4)
```

```
##
## Call:
## geeglm(formula = fev ~ male + age + currsmoker + year, data = smoking.long,
##     id = id)
##
##  Coefficients:
##             Estimate  Std.err   Wald Pr(>|W|)
## (Intercept)  3.70357  0.19437 363.05   <2e-16 ***
## male         0.13998  0.12744   1.21    0.272
## age         -0.00661  0.00776   0.73    0.394
## currsmoker  -0.28875  0.14311   4.07    0.044 *
## year        -0.11703  0.01161 101.59   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Estimated Scale Parameters:
##             Estimate Std.err
## (Intercept)    0.276  0.0379
##
## Correlation: Structure = independenceNumber of clusters:   78   Maximum cluster size: 4
```

If we explore more about geeglm, we can do:

```
?geeglm
```

We can specify different correlation structure by using:

```
model5<-geeglm(fev~male+age+currsmoker+year,id=id, data=smoking.long,corstr="ar1")
summary(model5)
```

```
##
## Call:
## geeglm(formula = fev ~ male + age + currsmoker + year, data = smoking.long,
##     id = id, corstr = "ar1")
##
##  Coefficients:
##             Estimate  Std.err   Wald Pr(>|W|)
## (Intercept)  3.65631  0.20411 320.88   <2e-16 ***
## male         0.16787  0.12927   1.69    0.194
## age         -0.00539  0.00803   0.45    0.502
## currsmoker  -0.31763  0.13966   5.17    0.023 *
## year        -0.11971  0.01221  96.06   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Estimated Scale Parameters:
##             Estimate Std.err
```

```
## (Intercept)    0.276  0.0379
##
## Correlation: Structure = ar1  Link = identity
##
## Estimated Correlation Parameters:
##       Estimate Std.err
## alpha    0.918  0.0151
## Number of clusters:   78   Maximum cluster size: 4
```

Please note that geeglm can also work for the binary response variable by specifying the family as binomial. We will not discuss this in detail.

If we want to compare these models, we can use QIC.

```r
library(MuMIn)
model.sel(model4,model5,rank = QIC)
```

```
## Model selection table
##        (Intrc)      age  crrsm  male   year corstr  qLik  QIC delta weight
## model4    3.70 -0.00661 -0.289 0.140 -0.117        -43.0 -371  0.00  0.627
## model5    3.66 -0.00539 -0.318 0.168 -0.120    ar1 -43.1 -370  1.04  0.373
## Models ranked by QIC(x)
```

Notice that smaller QIC values are better.

12.3 Non-linear effects

When we are fitting the regression models for continuous indepdendent variables, we are assuming the linearity of the effects.In this section, we use a hypothetical data set.

```r
x <- data.frame(time = c(0, 1, 2, 4, 6, 8, 9, 10, 11, 12, 13,
14, 15, 16, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30),
counts = c(126.6, 101.8, 71.6, 101.6, 68.1, 62.9, 45.5, 41.9,
46.3, 34.1, 38.2, 41.7, 24.7, 41.5, 36.6, 19.6,
22.8, 29.6, 23.5, 15.3, 13.4, 26.8, 9.8, 18.8, 25.9, 19.3))
#To do a simple regression
m1<-lm(counts ~ time,data=x)
summary(m1)
```

```
##
## Call:
## lm(formula = counts ~ time, data = x)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -20.08  -9.88  -1.88   8.49  39.44
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   87.155      6.019   14.48  2.3e-13 ***
## time          -2.825      0.332   -8.51  1.0e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.2 on 24 degrees of freedom
```

```
## Multiple R-squared:  0.751,  Adjusted R-squared:  0.741
## F-statistic: 72.5 on 1 and 24 DF,  p-value: 1.03e-08
```

```
#The model explains over 74% of the variance and has highly significant coefficients for the intercept
```

Next, let's plot the counts over time and superpose our linear model.

```
plot(x$time,x$counts)
abline(lm(counts ~ time,data=x), col = "blue")
```

bookdown-demo_files/figure-latex/unnamed-chunk-269-1.pdf

The model looks good, but we can see that the plot has curvature that is not explained well by a linear model. Now let's fit a quadratic regression splines for time:

```
x$time2<-x$time^2
m2<-lm(counts ~ time+time2,data=x)
summary(m2)
```

```
##
## Call:
## lm(formula = counts ~ time + time2, data = x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -24.265  -4.921  -0.952   5.586  18.773
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 110.1075     5.4803   20.09  4.4e-16 ***
## time         -7.4225     0.8058   -9.21  3.5e-09 ***
## time2         0.1506     0.0255    5.92  5.0e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.75 on 23 degrees of freedom
## Multiple R-squared:  0.901,  Adjusted R-squared:  0.893
## F-statistic:  105 on 2 and 23 DF,  p-value: 2.7e-12
```

```
#We can also use the following codes to obtain the same results.
m22<-lm(counts ~ poly(time,2),data=x)
summary(m22)
```

```
##
## Call:
## lm(formula = counts ~ poly(time, 2), data = x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -24.265  -4.921  -0.952   5.586  18.773
##
## Coefficients:
```

```
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)       42.61       1.91   22.28  < 2e-16 ***
## poly(time, 2)1  -129.09       9.75  -13.24  3.1e-12 ***
## poly(time, 2)2    57.71       9.75    5.92  5.0e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.75 on 23 degrees of freedom
## Multiple R-squared:  0.901,  Adjusted R-squared:  0.893
## F-statistic:  105 on 2 and 23 DF,  p-value: 2.7e-12
```

We can see that the quadratic model performs even better, explaining 89.3% of the variance(an additional 15% of the variance). Let's compare the two models:

```r
AIC(m1)
```

```
## [1] 219
```

```r
AIC(m2)
```

```
## [1] 197
```

```r
anova(m1,m2)
```

```
## Analysis of Variance Table
##
## Model 1: counts ~ time
## Model 2: counts ~ time + time2
##   Res.Df  RSS Df Sum of Sq  F Pr(>F)
## 1     24 5519
## 2     23 2188  1      3331 35  5e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We can see that the quadratic model is statistically significant better than the simple linear model. Now, let's plot the quadratic model.

```r
intervals <- seq(0, 30, 1)
new<-data.frame(time=intervals, time2=intervals^2)
new$pre <- predict(m2,newdata=new)
plot(x$time, x$counts, pch=16, xlab = "Time (s)", ylab = "Counts", cex.lab = 1.3)
lines(new$time, new$pre, col = "darkgreen", lwd = 3)
```

bookdown-demo_files/figure-latex/unnamed-chunk-272-1.pdf

The quadratic model appears to fit the data better than the linear model.

If we want to further fit a cubic model:

```r
x$time3<-x$time^3
m3<-lm(counts ~ time+time2+time3,data=x)
summary(m3)
```

```
##
## Call:
## lm(formula = counts ~ time + time2 + time3, data = x)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -25.058  -6.395  -0.006   6.458  20.203
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 114.37287    6.45922   17.71  1.7e-14 ***
## time         -9.50357    1.88791   -5.03  4.9e-05 ***
## time2         0.33104    0.15049    2.20    0.039 *
## time3        -0.00403    0.00332   -1.22    0.237
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.65 on 22 degrees of freedom
## Multiple R-squared:  0.908,  Adjusted R-squared:  0.895
## F-statistic:   72 on 3 and 22 DF,  p-value: 1.56e-11
```

```r
anova(m2,m3)
```

```
## Analysis of Variance Table
##
## Model 1: counts ~ time + time2
## Model 2: counts ~ time + time2 + time3
##   Res.Df  RSS Df Sum of Sq    F Pr(>F)
## 1     23 2188
## 2     22 2050  1       138 1.48   0.24
```

We can find that the cubic regression does not improve the model a lot.

There are some other approaches we can choose for the non-linear fitting.For instance, the B-spline basis function.

```r
library(splines)
m4<-lm(counts ~ bs(time),data=x)
summary(m4)
```

```
##
## Call:
## lm(formula = counts ~ bs(time), data = x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -25.058  -6.395  -0.006   6.458  20.203
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   114.37       6.46   17.71  1.7e-14 ***
## bs(time)1     -95.04      18.88   -5.03  4.9e-05 ***
## bs(time)2     -90.76      13.60   -6.68  1.0e-06 ***
## bs(time)3     -96.05       9.70   -9.90  1.5e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.65 on 22 degrees of freedom
## Multiple R-squared:  0.908,  Adjusted R-squared:  0.895
```

```
## F-statistic:   72 on 3 and 22 DF,  p-value: 1.56e-11
```

We can specify the degree of freedom and knots by using the df and knots arguments in the bs() function.To know more about the bs() function, please refer to:

```
?bs
```

12.4 Epi packages- Epitab()

There are some R packages specifically for the epidemiologists, which can be used easily and efficiently. For example, epicalc package and epitools package. We will only discuss the epitools package here.

```
install.packages("epitools")
```

Package epitools is for training and practicing epidemiologists including methods for twoway and multi-way contingency tables.There are a number of useful functions built-in this package. For instance, the function epitab() can be used to calculates risks, risk ratios, odds ratios and their associated confidence intervals. Here are a few examples.

```
library(epitools)
dig<-read.csv("data/dig.csv",stringsAsFactors=F) #digitalis data
names(dig)
```

```
##  [1] "ID"       "TRTMT"    "AGE"      "RACE"     "SEX"      "EJF_PER"
##  [7] "EJFMETH"  "CHESTX"   "BMI"      "KLEVEL"   "CREAT"    "DIGDOSER"
## [13] "CHFDUR"   "RALES"    "ELEVJVP"  "PEDEMA"   "RESTDYS"  "EXERTDYS"
## [19] "ACTLIMIT" "S3"       "PULCONG"  "NSYM"     "HEARTRTE" "DIABP"
## [25] "SYSBP"    "FUNCTCLS" "CHFETIOL" "PREVMI"   "ANGINA"   "DIABETES"
## [31] "HYPERTEN" "DIGUSE"   "DIURETK"  "DIURET"   "KSUPP"    "ACEINHIB"
## [37] "NITRATES" "HYDRAL"   "VASOD"    "DIGDOSE"  "CVD"      "CVDDAYS"
## [43] "WHF"      "WHFDAYS"  "DIG"      "DIGDAYS"  "MI"       "MIDAYS"
## [49] "UANG"     "UANGDAYS" "STRK"     "STRKDAYS" "SVA"      "SVADAYS"
## [55] "VENA"     "VENADAYS" "CREV"     "CREVDAYS" "OCVD"     "OCVDDAYS"
## [61] "RINF"     "RINFDAYS" "OTH"      "OTHDAYS"  "HOSP"     "HOSPDAYS"
## [67] "NHOSP"    "DEATH"    "DEATHDAY" "REASON"   "DWHF"     "DWHFDAYS"
```

```
mytab<-xtabs(~TRTMT + DEATH, data=dig)
#If we want to get the odds ratio, we can use:
OR<-mytab[1,1]*mytab[2,2]/(mytab[2,1]*mytab[1,2])
print(OR)
```

```
## [1] 0.986
```

```
#Or, we can choose to use the epitab()function:
epitab(mytab)
```

```
## $tab
##      DEATH
## TRTMT    0    p0    1    p1 oddsratio lower upper p.value
##     0 2209 0.499 1194 0.503     1.000    NA    NA      NA
##     1 2216 0.501 1181 0.497     0.986 0.892  1.09   0.799
##
## $measure
## [1] "wald"
##
## $conf.level
## [1] 0.95
```

```
##
## $pvalue
## [1] "fisher.exact"
```

```r
#There are other ways to use the epitab function:
epitab(dig$TRTMT,dig$DEATH)
```

```
## $tab
##          Outcome
## Predictor    0    p0    1    p1 oddsratio lower upper p.value
##         0 2209 0.499 1194 0.503     1.000    NA    NA      NA
##         1 2216 0.501 1181 0.497     0.986 0.892  1.09   0.799
##
## $measure
## [1] "wald"
##
## $conf.level
## [1] 0.95
##
## $pvalue
## [1] "fisher.exact"
```

```r
epitab(c(2209, 1194, 2216, 1181))
```

```
## $tab
##           Outcome
## Predictor  Disease1    p0 Disease2    p1 oddsratio lower upper p.value
##   Exposed1     2209 0.499     1194 0.503     1.000    NA    NA      NA
##   Exposed2     2216 0.501     1181 0.497     0.986 0.892  1.09   0.799
##
## $measure
## [1] "wald"
##
## $conf.level
## [1] 0.95
##
## $pvalue
## [1] "fisher.exact"
```

```r
#We can also use the epitab function to get the risk ratio:
epitab(mytab,method="riskratio")
```

```
## $tab
##      DEATH
## TRTMT    0    p0    1    p1 riskratio lower upper p.value
##     0 2209 0.649 1194 0.351     1.000    NA    NA      NA
##     1 2216 0.652 1181 0.348     0.991 0.929  1.06   0.799
##
## $measure
## [1] "wald"
##
## $conf.level
## [1] 0.95
##
## $pvalue
## [1] "fisher.exact"
```

```r
#Or we can use directly riskration()function:
riskratio(mytab)
```

```
## $data
##        DEATH
## TRTMT      0    1 Total
##   0     2209 1194  3403
##   1     2216 1181  3397
##   Total 4425 2375  6800
##
## $measure
##      risk ratio with 95% C.I.
## TRTMT estimate lower upper
##     0    1.000    NA    NA
##     1    0.991 0.929  1.06
##
## $p.value
##      two-sided
## TRTMT midp.exact fisher.exact chi.square
##     0         NA           NA         NA
##     1      0.782        0.799      0.781
##
## $correction
## [1] FALSE
##
## attr(,"method")
## [1] "Unconditional MLE & normal approximation (Wald) CI"
```

For more information about epitab or riskratio, we can use:

```
?epitab
?riskratio
```

For more inforamtion about the epitools package, please refer to (https://cran.r-project.org/web/packages/epitools/epitools.pdf).