Mathematics for Business Analytics and Finance

Sami Najafi

MSIS2402/2502

Module 0

A Preliminary Introduction to R Programming

Some Basics

Entering Commands and Getting Help

Using R as A Calculator

```
> 1+1
[1] 2
> max(1,3,5)
[1] 5
> pi
[1] 3.141593
> sqrt(2)
[1] 1.414214
```

To know more about a function:

```
> help("mean") or  > ?mean
> example(mean)
```

Printing Something

Print

```
print(x,number of digits)
Print("expression")
```

```
> print(sqrt(2),4)
[1] 1.414
```

The only way to print multiple items is to print them one at a time

```
> print("The zero occurs at"); print(2*pi); print("radians")
[1] "The zero occurs at"
[1] 6.283185
[1] "radians"
```

Printing Something

Concatenate

cat(num1/expr1, num2/expr2,…) combines multiple items into a continuous output

```
> cat("The zero occurs at", 2*pi, "radians.", "\n")
The zero occurs at 6.283185 radians.

> fib = c(0,1,1,2,3,5,8,13,21,34)
> cat("The first few Fibonacci numbers are:", fib, "...\n")
The first few Fibonacci numbers are: 0 1 1 2 3 5 8 13 21 34 ...

> iter=1
> cat("iteration = ", iter = iter + 1, "\n")
iteration = 2
```

Variables and Vectors

Variables

Defining Variables

Assignment operators: <- , <<- , = , -> , ->>.

```
> x <- 3
> z <- sqrt(x^2)
> print(z)
[1] 9


> f = 3
> print(f)
[1] 3

> 5 -> u
> print(u)
[1] 5
```

Variables

Deleting a Variable

rm(x) removes the variable

```
> x <- 2*pi
> x
[1] 6.283185
> rm(x)
> x
Error: object "x" not found
```

Vectors

Creating and Combining Vectors

```
c(first number or string,…,last number or character)
```

```
> c(1,1,2,3,5,8,13,21)
[1] 1 1 2 3 5 8 13 21
> c(1*pi, 2*pi, 3*pi, 4*pi)
[1] 3.141593 6.283185 9.424778 12.566371
> c("Everyone", "loves", "stats.")
[1] "Everyone" "loves" "stats."
```

Combining two vectors:

```
> v1 = c(1,2,3)
> v2 = c(4,5,6)
> c(v1,v2)
[1] 1 2 3 4 5 6
```

Vectors

Computing Basic Statistics

mean(x), median(x), sd(x) var(x), cor(x,y), cov(x,y), range(x), quantile(x)

summary(x) gives some of the summary statistics

```
> x = c(0,1,1,2,3,5,8,13,21,34)
> y = log(x+1)
> mean(x)
[1] 8.8
> median(x)
[1] 4
> sd(x)
[1] 11.03328

> summary(x)
Min.   1st Qu.   Median   Mean   3rd Qu.   Max.
0.00     1.25     4.00    8.80    11.75   34.00
```

Vectors

Creating Sequences

```
seq(from, to) or from:to
seq(from, to, by= )
seq(from, to, length= )
rep(number, number of repetitions)
```

```
> 10:19
[1] 10 11 12 13 14 15 16 17 18 19
> 9:0
[1] 9 8 7 6 5 4 3 2 1 0
> seq(from=0, to=20)
[1] 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
> seq(from=0, to=20, by=2)
[1] 0 2 4 6 8 10 12 14 16 18 20
> seq(from=1.0, to=2.0, length=5)
[1] 1.00 1.25 1.50 1.75 2.00
```

Vectors

Comparing Numbers and Vectors

The comparison operators compare two values and return TRUE or FALSE.

For two vectors, the comparison is component-wise. The result is a logical vector.

```
> a = 3
> a == pi
[1] FALSE
> a != pi
[1] TRUE
> a < pi
[1] TRUE

> v = c( 3, pi, 4)
> w = c(pi, pi, pi)
> v == w
[1] FALSE TRUE FALSE
```

Vectors

Comparing Numbers and Vectors

any(v == w) Return TRUE if any element of v equals the same element in w

all(v == w) Return TRUE if all element of v equals the same element in w

any(v == num) Return TRUE if any element of v equals the value num

all(v == num) Return TRUE if all element of v equals the value num

```
> v = c(3,pi,4)
> w = c(pi,pi,pi)
> v != w
[1] TRUE FALSE TRUE
> v <= w
[1] TRUE TRUE FALSE
> v != 3
[1] FALSE TRUE TRUE
> any(v != 3)
[1] TRUE
```

Vectors

Choosing Vector Elements

vec[a] Select element a

vec[a:b] Select elements a through b

vec(c(a,b,c)) Select elements a,b, and c.

vec(-a:-b) or vec(-(a:b)) exclude elements a through b

```
> fib = c(0,1,1,2,3,5,8,13,21,34)
> fib[6]
[1] 05
> fib[4:9]
[1] 2 3 5 8 13 21
> fib[c(1,2,4,8)]
[1] 0 1 2 13
> fib[-1:-5]
[1] 5 8 13 21 34
```

Vectors

More Complex Element Selections

Vec[vec<=a] Select elements that are not greater than a

Vec[vec>=a&vec<=b] Select elements that are between a and b inclusively

More complex element selections can be coded similarly.

```
> fib = c(0,1,1,2,3,5,8,13,21,34)
> fib[fib < 10] # Use that vector to select elements less than 10
[1] 0 1 1 2 3 5 8
> fib[fib > mean(fib)] # Use that vector to select elements less than 10
[1] 13 21 34
> fib[fib<=5|fib>=21] # | means “or”
[1] 0 1 1 2 3 5 21 34
> fib[fib>5&fib<21] # | means “and”
[1] 8 13
> w=c(1,2,4,3,5,6,8,2,7,0)
> fib[w>5]
[1] 5 8 21
```

Vectors

Arithmetic Operations

+, -, *, /, ^: These arithmetic operations are component-wise

```
> v = c(11,12,13,14,15)
> w = c(1,2,3,4,5)
> v + w
[1] 12 14 16 18 20
> v - w
[1] 10 10 10 10 10
> v * w
[1] 11 24 39 56 75
> v / w
[1] 11.000000 6.000000 4.333333 3.500000 3.000000
> w ^ v
[1] 1 4096 1594323 268435456 30517578125
```

Vectors

Arithmetic Operations between a Vector and a Scalar

The operation is performed between every vector element and the scalar

```
> w + 2
[1] 3 4 5 6 7
> w - 2
[1] -1 0 1 2 3
> w * 2
[1] 2 4 6 8 10
> w / 2
[1] 0.5 1.0 1.5 2.0 2.5
> w ^ 2
[1] 1 4 9 16 25
> 2 ^ w
[1] 2 4 8 16 32
```

Performing Vector Arithmetic

Operation of Functions on Vectors

Functions operate on every element. The result is a vector.

```
> w
[1] 1 2 3 4 5
> sqrt(w)
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
> log(w)
[1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379
> sin(w)
[1] 0.8414710 0.9092974 0.1411200 -0.7568025 -0.9589243
```

Vectors

More Vector Arithmetic

m%%n Modulo operator (gives the remainder of m/n)

%/% Integer division (gives the integer part of m/n)

%*% Matrix multiplication (to be studied later)

%in% Returns TRUE if the left operand occurs in its right operand; FALSE otherwise

```
> 14%%5
[1] 4
> 14%%5
[1] 2
> 5%in%14
[1] FALSE
> 5%in%c(5,4)
[1] TRUE
```

Functions, Loops, Matrices, and Lists

Functions

Defining a Function

```
function(param1, ...., paramN) expr

function(param1, ..., paramN) {
expr1
.
.
.
exprM
}
```

```
> cv = function(x) sd(x)/mean(x)
> cv(1:10)
[1] 0.5504819

s=function(n){
  if(n>=5) return (2*n)
  if((n>=0)&&(n<5))return (n)
  else return (3*n)
}
> s(3)
[1] 3
```

Loops

Creating Loops

```
for (variable in vector) {
expressions
}

while (condition) {

}
```

```
> factorial=1
> for (i in 1:5) {factorial = factorial *i}
> factorial
[1] 120

> while(i>=1){print(i);i=i-1}
[1] 3
[1] 2
[1] 1
```

Matrices

Creating a Matrix

```
Mat=matrix(c(a1,…,an),nrow=,ncol=,byrow=)
dim(Mat) given the matrix dimension
```

```
> mat=matrix(c(1,2,3,4),2,2,byrow=1)
> mat
     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

Lists

Creating a List

```
ls=list(num1/exp1,num2/exp2,…)
```

```
> m= matrix(list(1,2,"x",3,"y",4),2,3,byrow=1)
> m
     [,1] [,2] [,3]
[1,]  1    2    "x"
[2,]  3    "y"   4
m[[1,1]]+3
[1] 4

> s=c(m[[1,3]],"t")
> s
[1] "x" "t"
```

Input and Output

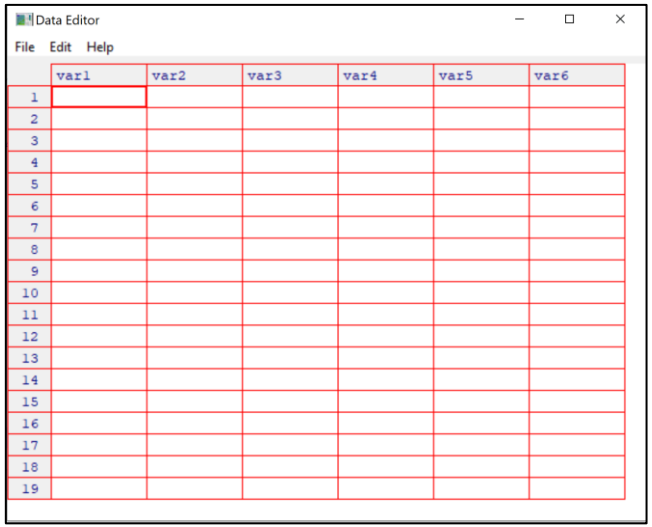
Input and Output

Entering Data from the Keyboard

For very small datasets, use the c() for vectors.

You can also create an empty dataset (data frame) and then use "edit" to fill it

```
> scores <- c(61, 66, 90, 88, 100)

> scores <- data.frame() # Create empty data frame
> scores <- edit(scores) # edit the data frame
```



Input and Output

Creating a Dataset on Keyboard

```
nameofdataframe = data.frame(
  var1=c(…),
  var2=c(…),
  var3=c(…)
  )
```

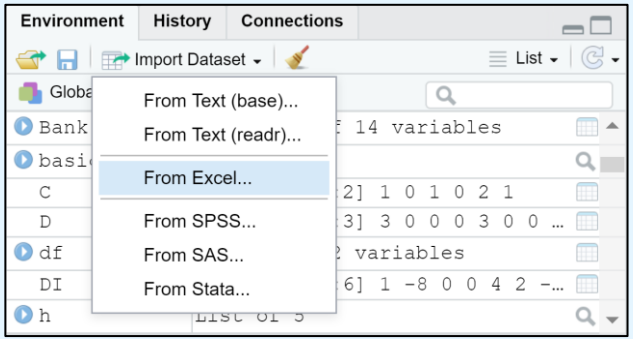
```
scores = data.frame(
  label=c("Low", "Mid", "High"),
  lbound=c( 0, 0.67, 1.64),
  ubound=c(0.674, 1.64, 2.33)
  )

> scores
  label   lbound      ubound
1 Low      0.00       0.674
2 Mid      0.67       1.640
3 High     1.64       2.330
```

Input and Output

Importing a Dataset From Excel

```
library(readxl)
Data_name = read_excel("enter the address of the excel file", + sheet = "enter the sheet name")
```

An easier way: Import Dataset → From Excel




```
> library(readxl)
> Bank <- read_excel("address…", + sheet = "mysheet")
> View(Bank)
```

(Use the posted UniversalBank.xlsx to practice these codes)

Input and Output

Exporting a Dataset (Data Frame) in Excel

```
library(writexl)

write_xlsx(list(Nameofsheet1 = dataforsheet1,
                Nameofsheet2 = dataforsheet2,
                Nameofsheet3 = dataforsheet3),
           "filename.xlsx")
```

```
> library(writexl)
> write_xlsx(list(mysheet=df, mysheet2=df),"C:/Users/Vortex/Dropbox/Math course/Bank5.xlsx")
```

Some Other Statistical Tools

Some Other Statistical Tools

Choose a Random Sample

```
sample(x, size, replace = FALSE)

x: the vector
size: size of the sample
relace: False/True whether the sample is with replacement or not
```

In the UniversalBank.xlsx case, after importing the dataset as a new data frame Bank, we can code like this:

```
> sample(Bank$ID,10)
[1] 1981 3717 2055 1622 868 2893 4634 1231 1202 385
```

Some Other Statistical Tools

Generating a Random Sequence

```
sample(x, size, replace = TRUE,Prob)

x: the vector
size: size of the sample
relace: False/True whether the sample is with replacement or not
Prob: a vector of probability weights for obtaining the elements of the vector being sampled.
```

```
> sample(c("H","T"),10,replace=TRUE,c(0.5,0.5))
[1] "T" "T" "T" "H" "H" "T" "H" "T" "T" "H"
```

Some Other Statistical Tools

Tabulating and Creating Contingency Tables

table(x) create a summary table based on different categories in x

Table(x,y) creates a contingency table

x, y: categorical variables

```
> table(Bank$Family)
   1       2          3          4
1472    1296       1010       1222
```

```
> table(Bank$Family,Bank$Education)
    1   2   3
1 678 326 468
2 657 265 374
3 349 383 278
4 412 429 381
```

Some Other Statistical Tools

Testing Categorical Variables for Independence

summary(table(x,y)) creates a contingency table

x, y: categorical variables

```
> summary(table(Bank$Family,Bank$Education))
Number of cases in table: 5000
Number of factors: 2
Test for independence of all factors: Chisq = 169.01, df = 6, p-value = 7.288e-34
```

Remark: The small *p*-value (e.g., p-value<0.05) indicates that the two factors are probably not independent. Practically speaking, we conclude there is some relationship between the two variables.

Some Other Statistical Tools

Converting Data to Z-Scores

```
scale(x)

x: a vector
```

```
> v=scale(Bank$Income)
> v[1:5]
[1] -0.5381750 -0.8640230 -1.3636566 0.5697084 -0.6250678
```

Some Other Statistical Tools

Testing the Mean of a Sample (t Test)

```
t.test(x, mu=m)

x: a vector
m: the hypothesized mean value for the data
```

```
> t.test(Bank$Income,mu=50)

One Sample t-test
data: Bank$Income
t = 36.519, df = 4999, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 50
95 percent confidence interval: 72.49792 75.05048
sample estimates:
mean of x 73.7742
```

Remark: The small *p*-value (e.g., p-value<0.05) indicates that the population mean is different from the hypothesized value mu.