IST 3420: Introduction to Data Science and Management

Langtao Chen, Fall 2016

Part 2: R Programming

Agenda

- Introduction to R Programming
- Data Structures in R
- R Functions
- Control Structures in R
- R Programming Style and Debug
- Dynamic Report: R Markdown

Introduction to R Programming

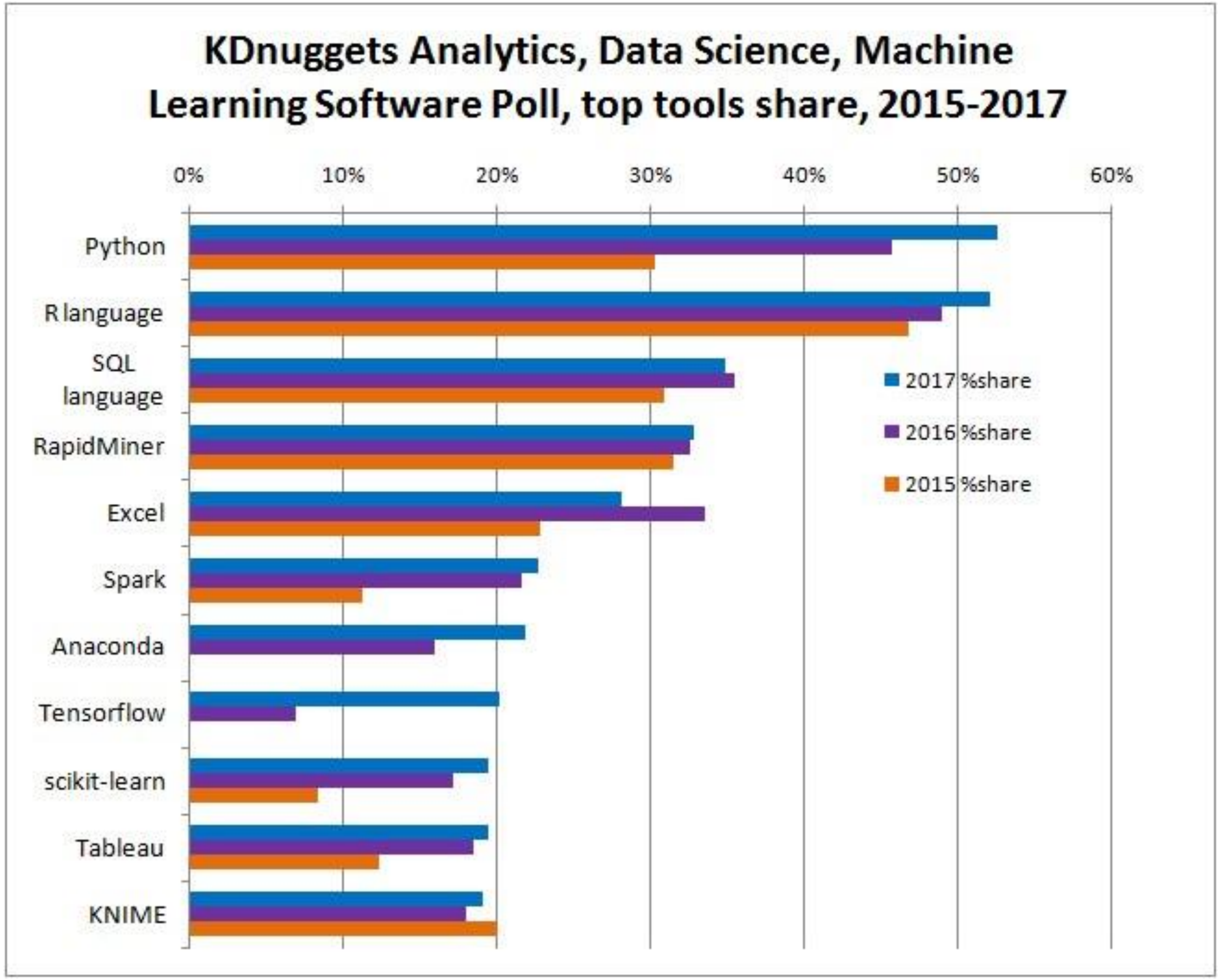
R

- A free, open-source programming language for statistical computing
- An interpreted language (executed directly, no compilation)
- R supports matrix arithmetic (like Matlab)
- R supports both procedural programming and object-oriented programming

Comprehensive R Archive Network

- Capability extended through a packaging system on CRAN, the Comprehensive R Archive Network
  - http://cran.r-project.org/
- So many useful packages available on CRAN
- You can contribute to CRAN by uploading your own package!

Revisit: R is popular; Don't get left behind.



To learn Python, choose "IST 5520 – Data Science and Machine Learning with Python" offered in Spring.

http://www.kdnuggets.com/2017/05/poll-analytics-data-science-machine-learning-software-leaders.html

Steep Learning Curve for R

- The "weird" syntax of R

"The best thing about R is that it was developed by statisticians. The worst thing about R is that ... it was developed by statisticians."

-- Bo Cowgill, Google

"Unlike other high-level scripting languages, such as Python or Ruby, R has a unique and somewhat prickly syntax and tends to have a steeper learning curve than other languages."

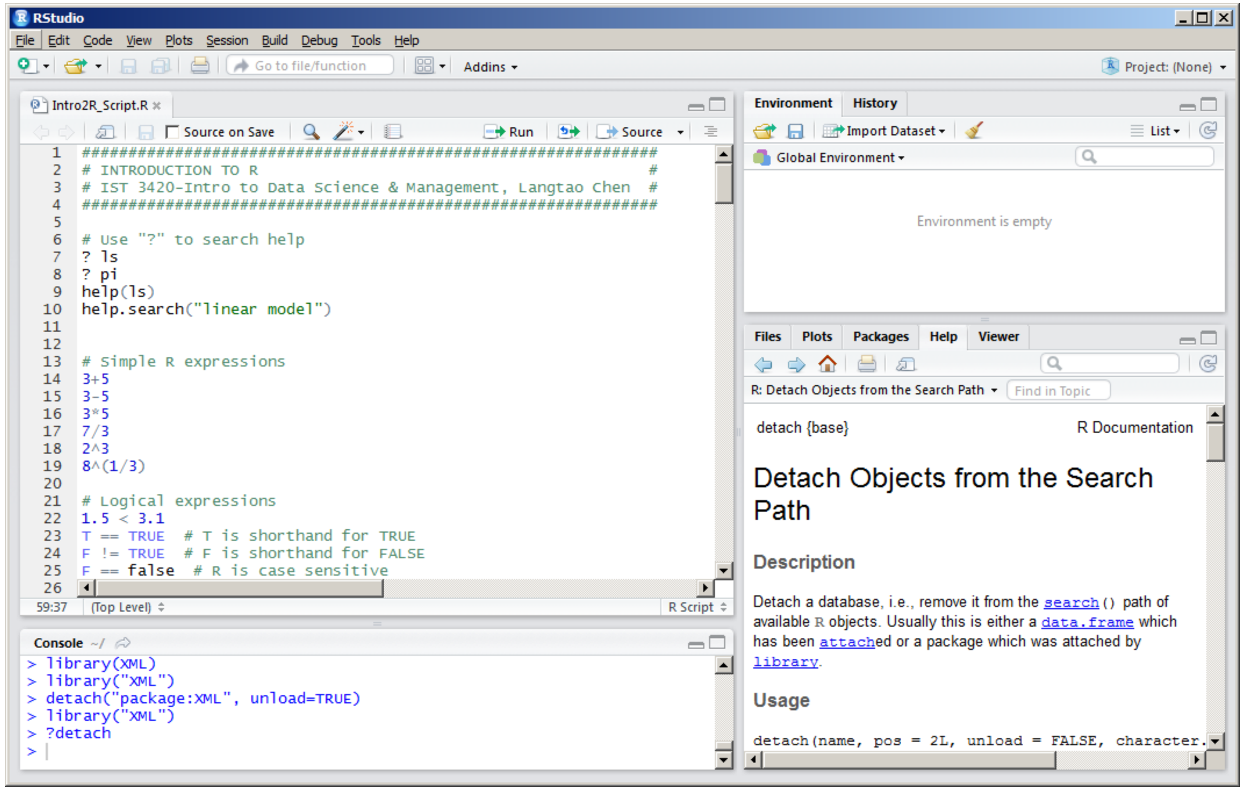
-- Drew Conway & John White, "Machine Learning for Hackers" P2.

Resources for Learning R

- Remember the Rseek (search engine for R language)!
  - http://rseek.org/
- "An Introduction to R"
  - https://cran.r-project.org/doc/manuals/R-intro.pdf
- R Language Definition
  - https://cran.r-project.org/doc/manuals/r-release/R-lang.pdf
- "R Reference Card" – quick reference for important tasks
  - https://cran.r-project.org/doc/contrib/Short-refcard.pdf
- A Step-by-Step R Tutorial
  - http://www.cyclismo.org/tutorial/R/
- Stack Overflow Q&A Site
  - http://stackoverflow.com/questions/tagged/r
- Commonly Used R Packages
  - https://support.rstudio.com/hc/en-us/articles/201057987-Quick-list-of-useful-R-packages

RStudio

- An open-source IDE for R
- Install the RStudio Desktop (open source edition) from https://www.rstudio.com/products/rstudio/download/



```
##################################################################
# INTRODUCTION TO R                                              #
# IST 3420-Intro to Data Science & Management, Langtao Chen       #
##################################################################

# Use "?" to search help
? ls
? pi
help(ls)
help.search("linear model")


# Simple R expressions
3+5
3-5
3*5
7/3
2^3
8^(1/3)

# Logical expressions
1.5 < 3.1
T == TRUE   # T is shorthand for TRUE
F != TRUE   # F is shorthand for FALSE
F == false  # R is case sensitive
```

```
> library(XML)
> library("XML")
> detach("package:XML", unload=TRUE)
> library("XML")
> ?detach
> 
```

detach {base} R Documentation

Detach Objects from the Search Path

Description

Detach a database, i.e., remove it from the search() path of available R objects. Usually this is either a data.frame which has been attached or a package which was attached by library.

Usage

```
detach(name, pos = 2L, unload = FALSE, character.
```

Some Useful RStudio Keyboard Shortcuts

- For a complete list, refer to https://support.rstudio.com/hc/en-us/articles/200711853-Keyboard-Shortcuts

Function	***Windows & Linux***	***Mac***
Move cursor to Source Editor	Ctrl + 1	Ctrl + 1
Move cursor to Console	Ctrl + 2	Ctrl + 2
Interrupt currently executing command	Esc	Esc
Navigate command history	Up/Down	Up/Down
Run current line/selection	Ctrl + Enter	Command + Enter
Save active document	Ctrl + S	Command + S

R Basic Concepts

- Everything that exists in R is an object.

- Everything that happens in R is a function call.

- Interfaces to other software are part of R.

Source: Chambers, John M. *Extending R*. CRC Press, 2016.

Attributes of an Object

- names
- dimnames
- dim
- class
- attributes (contain metadata)
- length (works on vectors and lists)
- nchar (number of characters in a string)

Basic Operations

- R is case sensitive!
- Use "?" to search help
- Constants and symbols
  - Any number typed directly is a constant.
  - The name of a variable is a symbol.
- Two assignment operators
  - Left assignment <- (for example, a <- 4)
  - Right assignment -> (for example, 4 -> b)
- List indexing: $

Atomic Data Types

- Character
  - “a”, “hello”
- Logical
  - TRUE, FALSE
- Integer
  - x <- 5L # Must add L at the end to explicitly denote integer
- Double
  - 4, 13.48
- Complex
  - 2 + 3i

R Basic Operators

- Arithmetic Operators

Operator	Meaning	Unary or Binary
+	Plus	Both
-	Minus	Both
*	Multiplication	Binary
/	Division	Binary
^	Exponentiation	Binary
%%	Modulus	Binary
%/%	Integer division	Binary
%*%	Matrix product	Binary
%o%	Outer product	Binary

(cont.)

- Comparison Operators

Operator	Meaning	Unary or Binary	Example (a is 4)	Result
<	Less than	Binary	a < 0	FALSE
>	Greater than	Binary	a > 0	TRUE
==	Equal to	Binary	a == 3	FALSE
>=	Greater than or equal to	Binary	a >= 0	TRUE
<=	Less than or equal to	Binary	a <= 0	FALSE
!=	Not equal to	Binary	a !=3	TRUE

(cont.)

- Logic Operators

Operator	Meaning	Unary or Binary	Example (a is TRUE, b is FALSE)	Result
&	And, vectorized	Binary	a & b	FALSE
|	Or, vectorized	Binary	a | b	TRUE
&&	And, not vectorized	Binary	a && b	FALSE
||	Or, not vectorized	Binary	a || b	TRUE
!	Not	Unary	!a	TRUE
xor	Exclusive or	Binary	xor(a,b)	TRUE
isTrue()	Test if true	Unary	isTRUE(a)	FALSE

True Tables for Logical Operators

a	b	!a	a & b	a | b	a && b	a || b	xor(a,b)
TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE

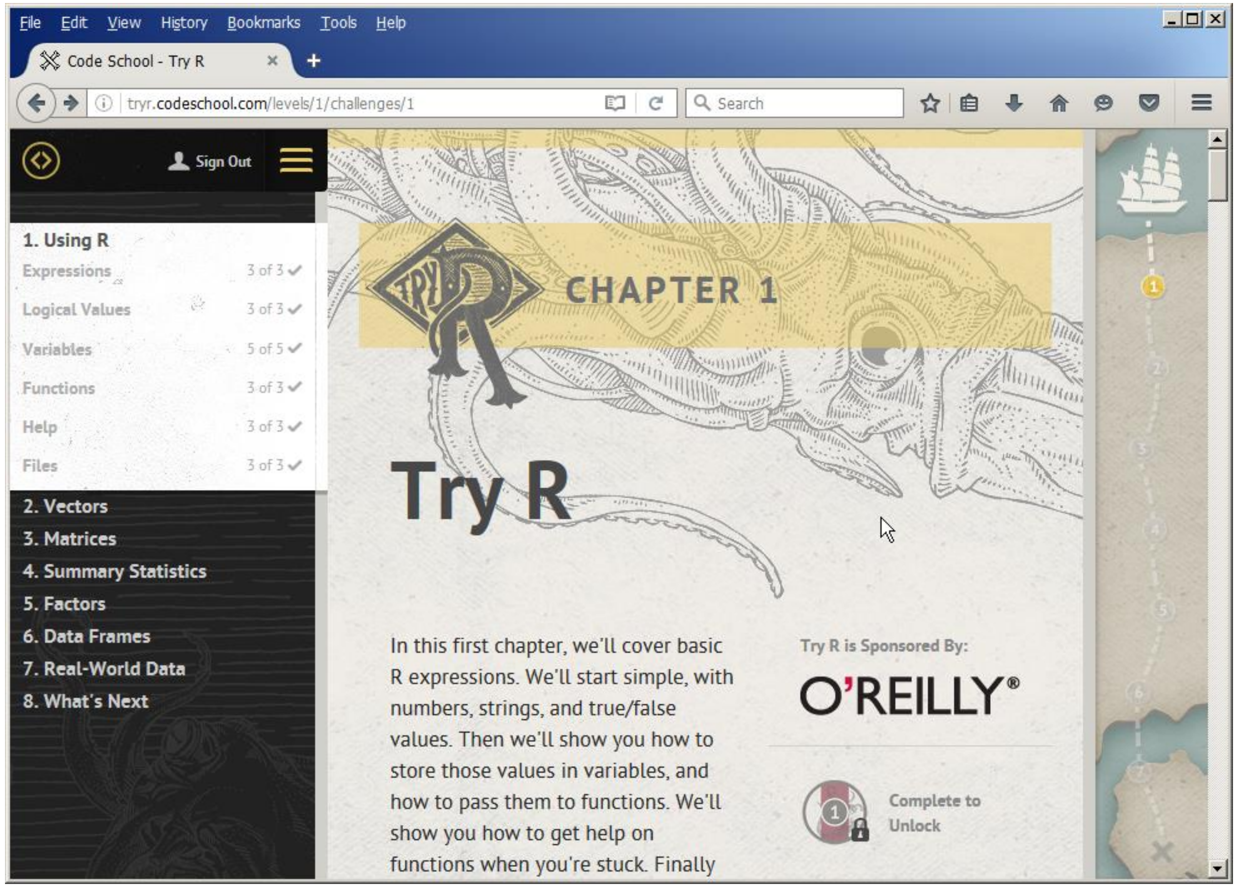
Inf, NaN, and NULL

- Inf and -Inf are positive and negative infinity
- NaN (not a number, an undefined value)
- NULL means that object does not exist

```
> 1/0
[1] Inf
> 0/0
[1] NaN
> 1/0 + 1/0
[1] Inf
> 1/0 - 1/0
[1] NaN
> sin(Inf)
[1] NaN
Warning message: In sin(Inf) : NaNs produced
> dim(1)
NULL
```

To be familiar with R

http://tryr.codeschool.com/levels/1/challenges/1


File Edit View History Bookmarks Tools Help
Code School - Try R
tryr.codeschool.com/levels/1/challenges/1
Search
Sign Out
CHAPTER 1
Try R

1. Using R

Expressions	3 of 3 ✓
Logical Values	3 of 3 ✓
Variables	5 of 5 ✓
Functions	3 of 3 ✓
Help	3 of 3 ✓
Files	3 of 3 ✓

2. Vectors
3. Matrices
4. Summary Statistics
5. Factors
6. Data Frames
7. Real-World Data
8. What's Next

In this first chapter, we'll cover basic R expressions. We'll start simple, with numbers, strings, and true/false values. Then we'll show you how to store those values in variables, and how to pass them to functions. We'll show you how to get help on functions when you're stuck. Finally

Try R is Sponsored By:

O'REILLY®

Complete to Unlock

Data Structures in R

R Data Structures

- Vectors, matrices, arrays, data frames (like tables in a RDBMS), and lists

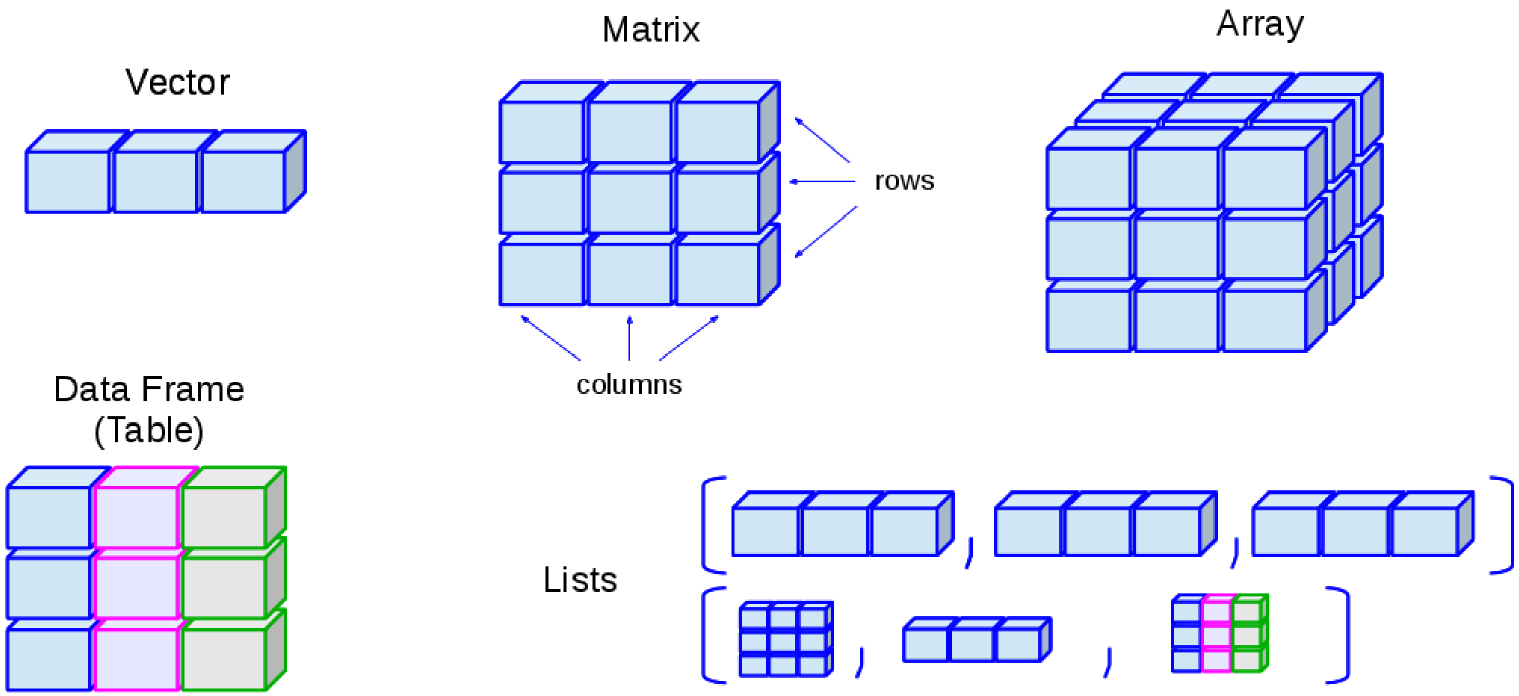
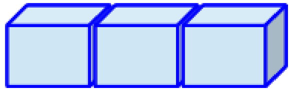


Image source: http://venus.ifca.unican.es/Rintro/dataStruct.html

Vectors

- An ordered collection of elements
- Create a vector of numbers
  - v1 <- c(1,2,3,4)
- Use [] to access vector elements
- Create a vector of strings
  - v2 <- c("a","b","c")
- Elements in a vector should be of the same type
  - v3 <- c(1, "a")
  - mode(v3) # Check the type of storage mode
    [1] "character"

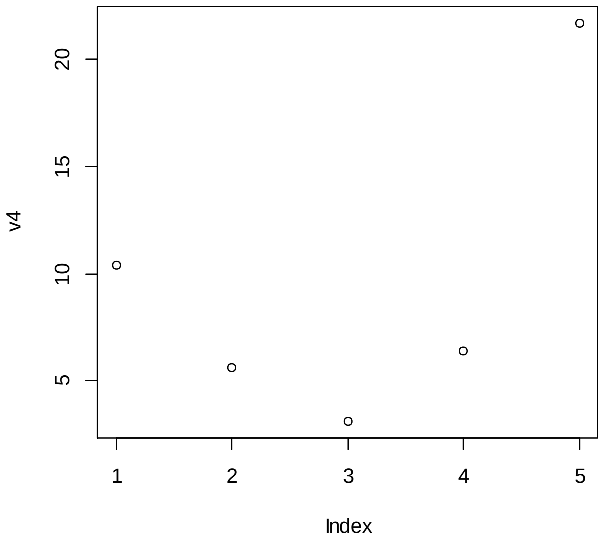
Vector

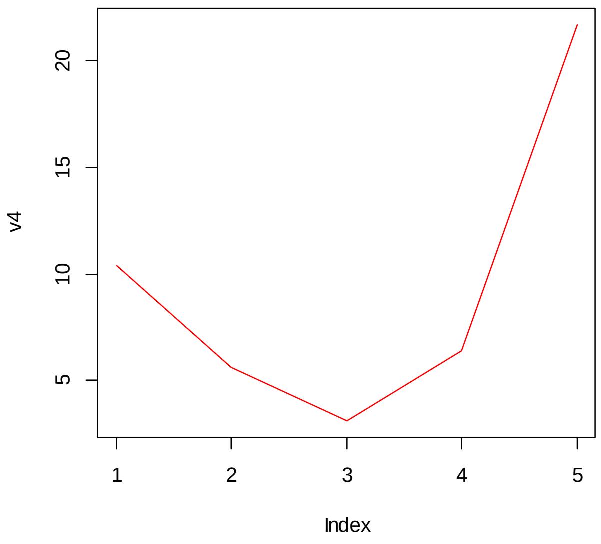


c function: c means "combine"

Plot Vectors

- Plot vectors
  - v4 <- c(10.4, 5.6, 3.1, 6.4, 21.7)
  - plot(v4) # plot the vector
  - plot(v4, type = "l",col = "red") # plot the vector in line graph





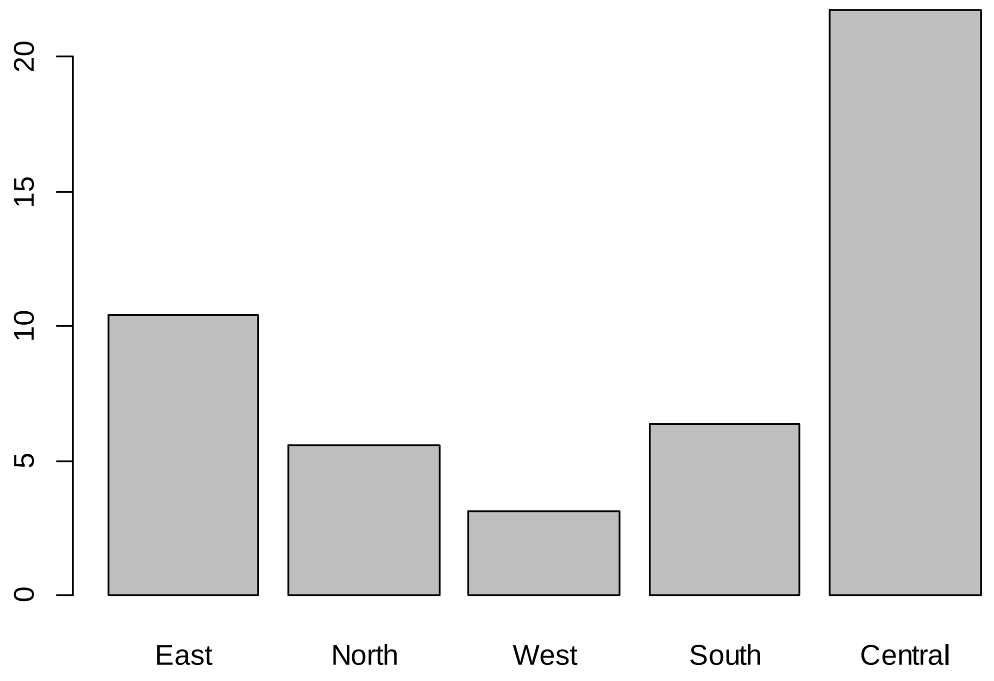
Names of Vectors

- Use names() to set or get names of an object

```
> v4
[1] 10.4 5.6 3.1 6.4 21.7
> names(v4) <- c("East","North","West","South","Central")
# To set vector name
> v4
  East North West South Central
  10.4 5.6 3.1 6.4 21.7
> names(v4) # To get vector name
[1] "East" "North" "West" "South" "Central"
```

Bar Plot

- barplot(v4)



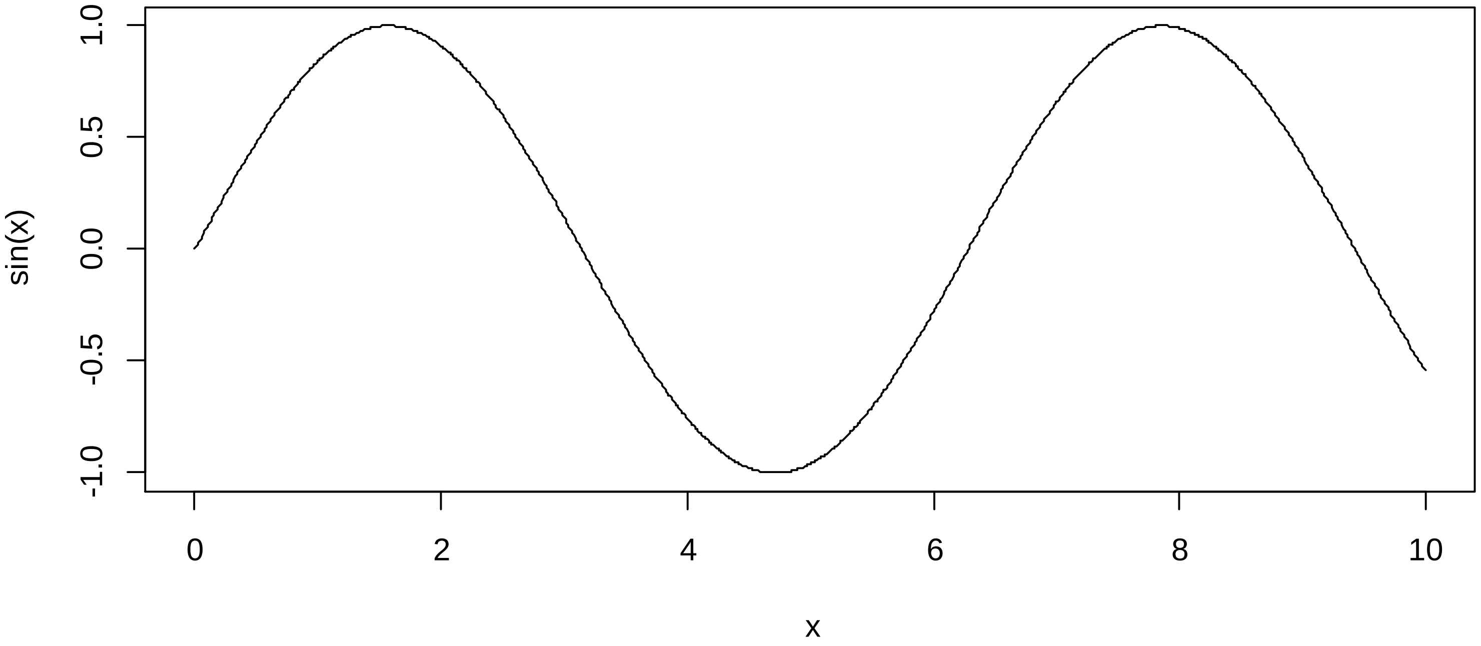
Sequences

- Use colon :
- Use seq() function

```
> 5:9
[1] 5 6 7 8 9
> seq(5,9)
[1] 5 6 7 8 9
> seq(5,9,by = 1)
[1] 5 6 7 8 9
> seq(5,9,by = 0.5)
[1] 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5 9.0
> seq(from = 5, to = 9, by = 0.4)
[1] 5.0 5.4 5.8 6.2 6.6 7.0 7.4 7.8 8.2 8.6 9.0
> seq(0, 1, length.out = 11)
[1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

Plot

```
> x <- seq(0,10, by=0.01)
> plot(x,sin(x),type ="l")
```



Repetitions

- Use rep( ) function

```
> rep(1:4, 3)
[1] 1 2 3 4 1 2 3 4 1 2 3 4
> rep(1:4, each = 3)
[1] 1 1 1 2 2 2 3 3 3 4 4 4
> rep(1:4, c(3,3,3,3))
[1] 1 1 1 2 2 2 3 3 3 4 4 4
> rep(1:4, c(1,2,3,4))
[1] 1 2 2 3 3 3 4 4 4 4
```

Vector Math

- Most arithmetic operations work as well

```
> a <- c(1,2,3,4)
> b <- a + 2
> a*2
[1] 2 4 6 8
> a/3
[1] 0.3333333 0.6666667 1.0000000 1.3333333
> a^2
[1] 1 4 9 16
> a<b
[1] TRUE TRUE TRUE TRUE
> sin(b)
[1] 0.1411200 -0.7568025 -0.9589243 -0.2794155
```

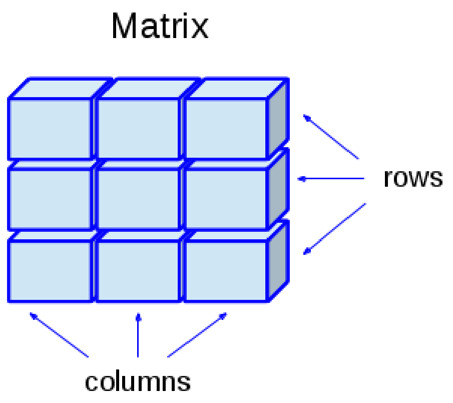
Logical Operators “&” vs. “&&” (“ | ” vs. “ | | ”)

- What’s the difference between “&” and “&&” or between “|” and “||”?
  - The longer form evaluates left to right examining only the first element of each vector.
  - (a<2)&&(b<4) is equivalent to (a[1]<2)&(b[1]<4)

```
> a
[1] 1 2 3 4
> b
[1] 3 4 5 6
> (a<2)&(b<4)
[1] TRUE FALSE FALSE FALSE
> (a<2)&&(b<4)
[1] TRUE
```

Matrices

- A matrix is a bi-dimensional array
  - Rows
  - Columns



Colum Names and Row Names

- Use colnames and rownames to retrieve or set the row and column names of a matrix-like object.

```
> m2 <- matrix(1:12,ncol = 4, byrow = TRUE)
> m2
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
> colnames(m2) <- c("a","b","c","d")
> rownames(m2) <- c("i","j","k")
> m2
  a  b  c  d
i 1  2  3  4
j 5  6  7  8
k 9 10 11 12
> colnames(m2)
[1] "a" "b" "c" "d"
> rownames(m2)
[1] "i" "j" "k"
```

Matrix Computations

- Addition
- Subtraction
- Scalar multiplication
- Element-wise multiplication
- Matrix multiplication
- Inverse
- Cholesky Decomposition

(cont.)

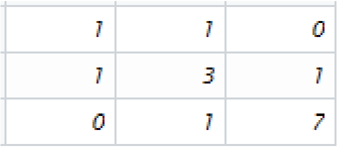
- Cholesky Decomposition (Cholesky Factorization)

Every positive definite matrix A can be decomposed as

$$A = LL^T$$

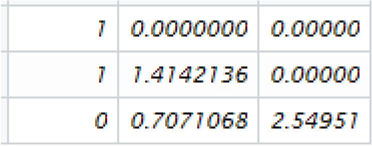
where L is a lower triangular matrix with positive diagonal elements

A=



1	1	0
1	3	1
0	1	7

L=



1	0.0000000	0.00000
1	1.4142136	0.00000
0	0.7071068	2.54951

For more information, refer to
https://en.wikipedia.org/wiki/Cholesky_decomposition

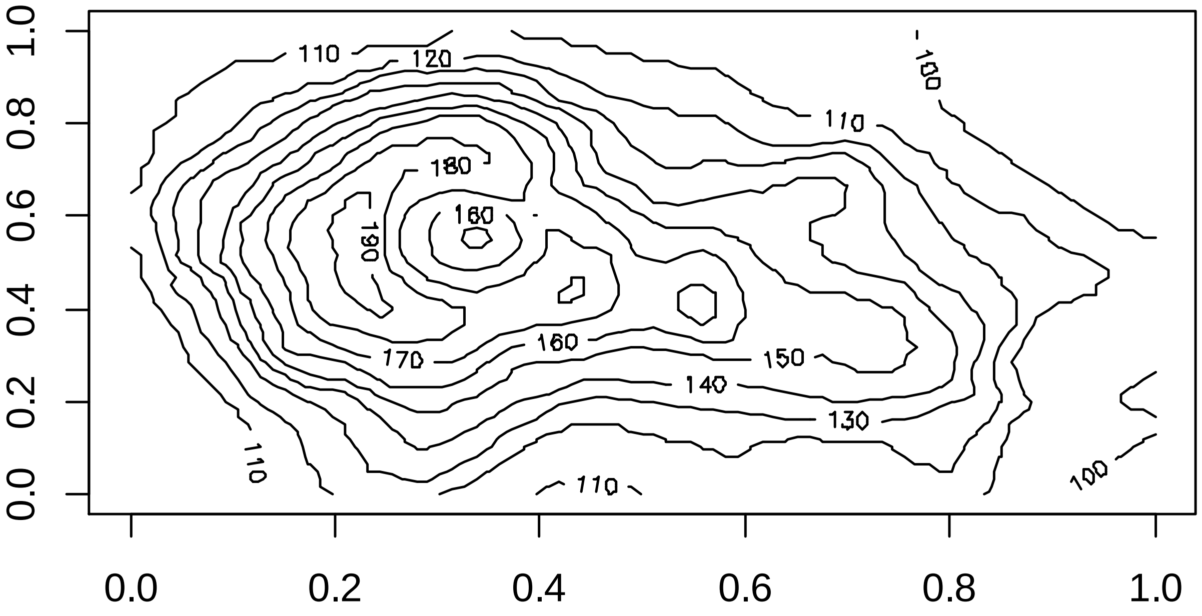
Plot Matrix: Maunga Whau Volcano

- Maunga Whau (Mt Eden) is one of about 50 volcanos in the Auckland volcanic field. This data set gives topographic information for Maunga Whau on a 10m by 10m grid.
- volcano is an elevation matrix with 87 rows and 61 columns, rows corresponding to grid lines running east to west and columns to grid lines running south to north.

Image source: http://www.teara.govt.nz/en/photograph/8706/mt-eden

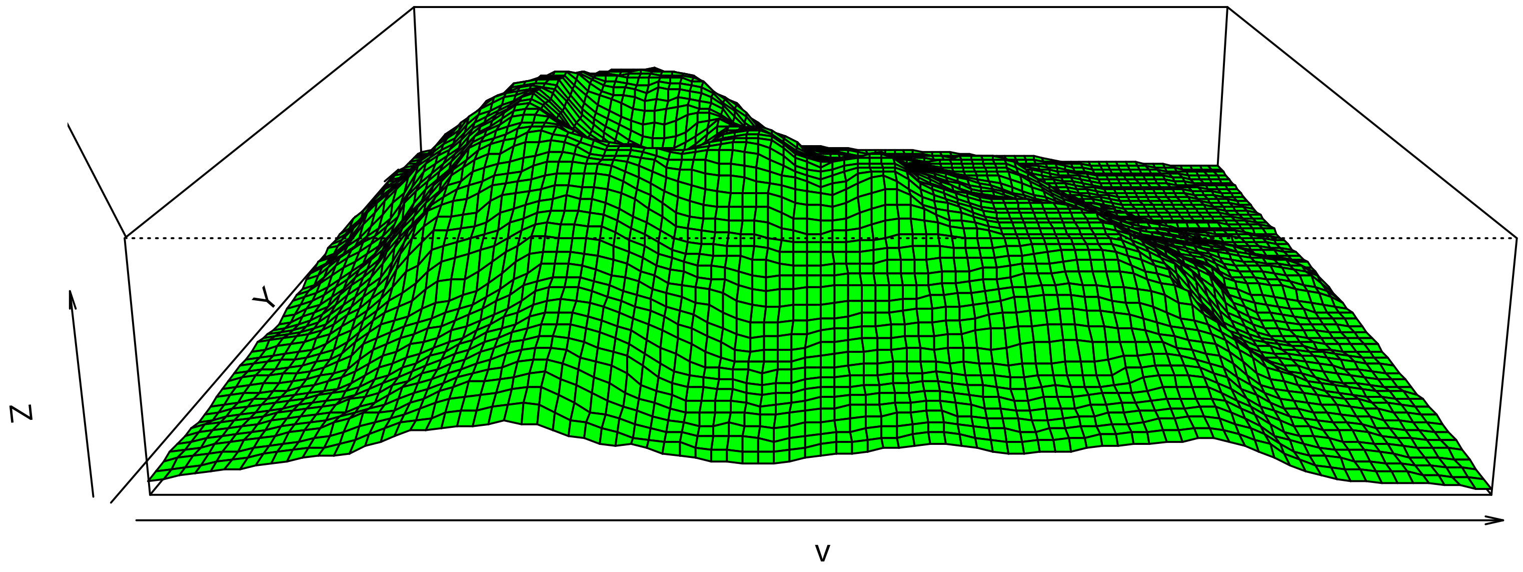
Draw a Contour Map

- v <- volcano
- contour(v)



Draw a Perspective Plot

- v <- volcano
- persp(v, expand=0.2, col = "green")

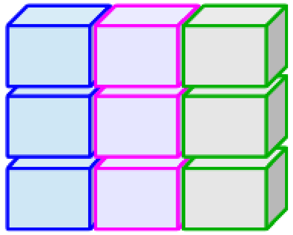


Data Frames

- Data frame is a list of vectors of equal length.
- Each column should be of the same type.
- Similar to tables in RDBMS, or data set in SAS or SPSS, i.e. a "cases by variables" matrix of data.

```
> id <- c(11,12,13)
> name <- c("Lily","Jim","Tom")
> credit <- c(710,700,680)
> df <- data.frame(id,name,credit)
> df
  id name credit
1 11 Lily    710
2 12  Jim    700
3 13  Tom    680
> df["name"]  # Show the name column
  name
1 Lily
2  Jim
3  Tom
> df[["name"]]
[1] Lily Jim  Tom
Levels: Jim Lily Tom
```

Data Frame
(Table)



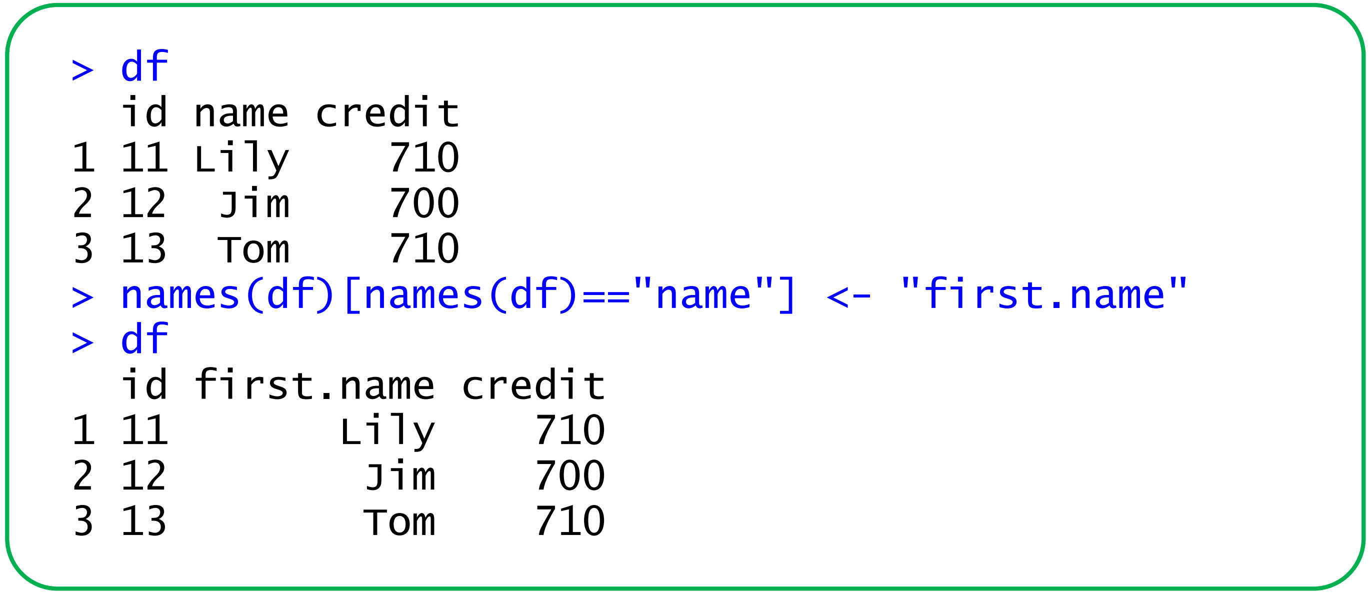
(cont.)

- Single brackets vs. double brackets
  - Single brackets mean subsetting, the result is a data frame
  - Double brackets return a vector (or use $ sign)

```
> a <- df["credit"]
> attributes(a)
$names
[1] "credit"

$row.names
[1] 1 2 3

$class
[1] "data.frame"

> b <- df[["credit"]]
> attributes(b)
NULL
> c <- df$credit  # A shorthand for double brackets
> attributes(c)
NULL
```

(cont.)

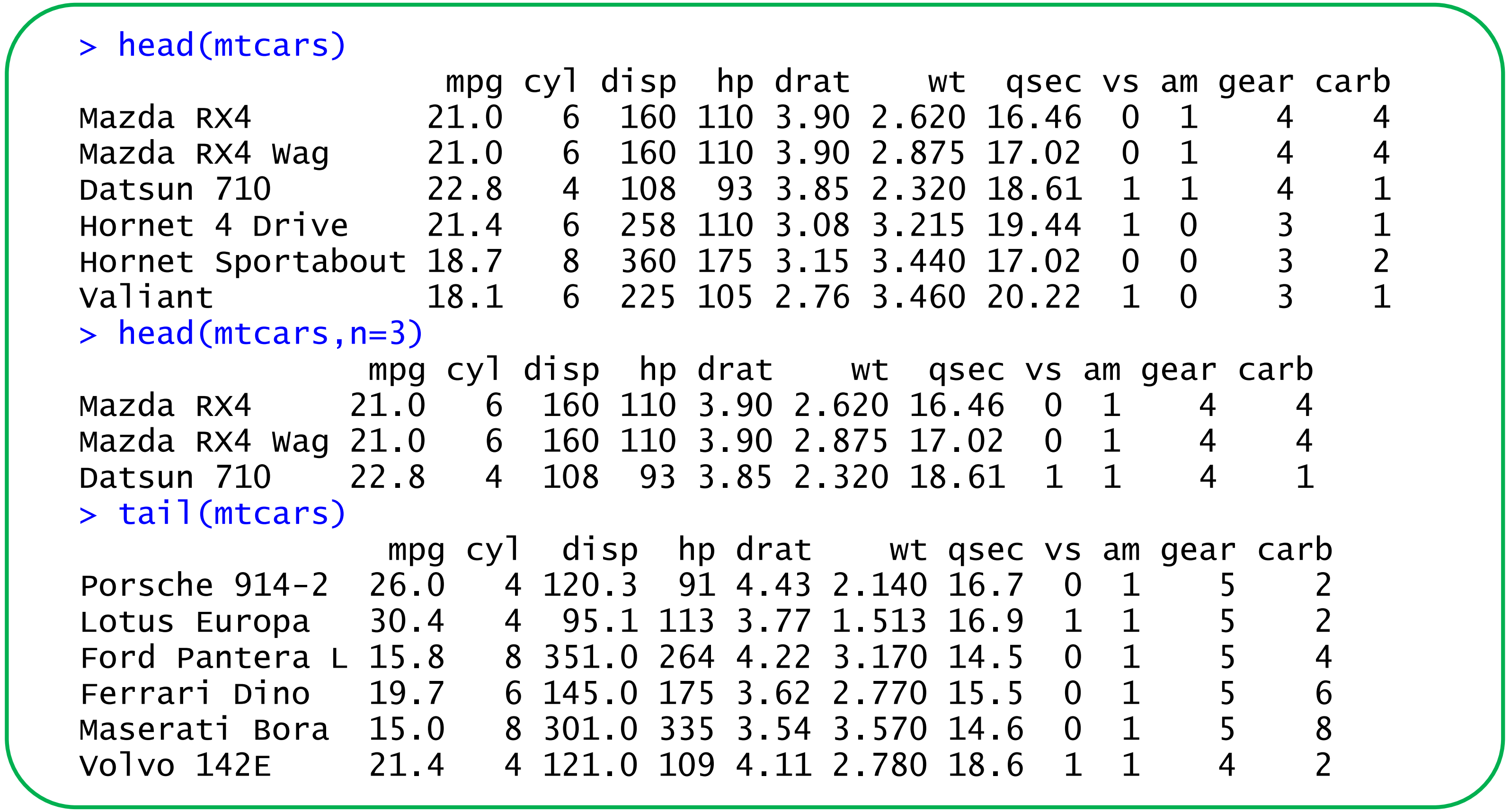
- Rename Columns
  - names(data.frame)[names(data.frame)==“old.name”] <- “new.name”



```
> df
  id name credit
1 11 Lily    710
2 12  Jim    700
3 13  Tom    710
> names(df)[names(df)=="name"] <- "first.name"
> df
  id first.name credit
1 11       Lily    710
2 12        Jim    700
3 13        Tom    710
```

Motor Trend Data Built in R

- The mtcars is a built-in data frame which comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).



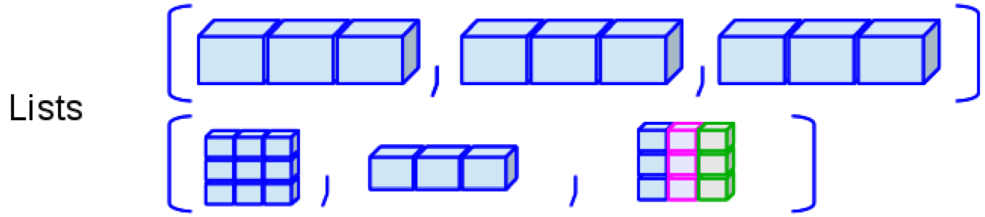
```
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
> head(mtcars,n=3)
               mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4     21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710    22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
> tail(mtcars)
                mpg cyl  disp  hp drat    wt qsec vs am gear carb
Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

Summary of All Variables(Columns)

```
> summary(mtcars)  # Summary of all variables(columns)
      mpg             cyl             disp             hp             drat      
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0   Min.   :2.760  
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5   1st Qu.:3.080  
 Median :19.20   Median :6.000   Median :196.3   Median :123.0   Median :3.695  
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7   Mean   :3.597  
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0   3rd Qu.:3.920  
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0   Max.   :4.930  
       wt             qsec             vs               am        
 Min.   :1.513   Min.   :14.50   Min.   :0.0000   Min.   :0.0000  
 1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000   1st Qu.:0.0000  
 Median :3.325   Median :17.71   Median :0.0000   Median :0.0000  
 Mean   :3.217   Mean   :17.85   Mean   :0.4375   Mean   :0.4062  
 3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000   3rd Qu.:1.0000  
 Max.   :5.424   Max.   :22.90   Max.   :1.0000   Max.   :1.0000  
      gear            carb      
 Min.   :3.000   Min.   :1.000  
 1st Qu.:3.000   1st Qu.:2.000  
 Median :4.000   Median :2.000  
 Mean   :3.688   Mean   :2.812  
 3rd Qu.:4.000   3rd Qu.:4.000  
 Max.   :5.000   Max.   :8.000  
```

Lists

- A list is a special type of vector. Elements can be of different types.
- Use lists act as containers.



(cont.)

```
> list1 <- list("a",5,TRUE)
> list1
[[1]]
[1] "a"

[[2]]
[1] 5

[[3]]
[1] TRUE

> list1[[1]]
[1] "a"
> list2 <- list(list1,1+2i)  # list2 contains list1
> list2[[1]][[2]]
[1] 5
```

Two List Index Forms

- Difference between [[and [notations
  - [[form allows only a single element to be selected using integer or character indices.
  - [allows indexing by vectors.

R Functions

Functions: Closure Type Objects

- So many built-in functions available
- You can define your own functions
  - Function name
  - Input (argument list)
  - Output

```
> f2c <- function(f){
+ # Fahrenheit to Celsius conversion
+ c <- (f-32)*5/9
+ return(c)
+ }
> f2c(90)
[1] 32.22222
> f2c(32)
[1] 0
> typeof(f2c)
[1] "closure"
```

Writing Your Own Functions

- Syntax

  function (arglist) body

  - The keyword function indicates that you want to create a function.
  - An argument list is a comma separated list of formal arguments. A formal argument can be a symbol, a statement of the form 'symbol = expression', or the special formal argument '...'.
  - The body can be any valid R expression. Generally, the body is a group of expressions contained in curly braces ('{' and '}') called block.
  - Generally functions are assigned to symbols but they don't need to be (anonymous functions).

Source: https://cran.r-project.org/doc/manuals/r-release/R-lang.html

(cont.)

- Formal arguments define the variables whose values will be supplied at the time the function is invoked. The names of these arguments can be used within the function body.

- Default values for arguments can be specified using the special form ‘name = expression’. In this case, if the user does not specify a value for the argument when the function is invoked the expression will be associated with the corresponding symbol.

Source: https://cran.r-project.org/doc/manuals/r-release/R-lang.html

Control Structures in R

Statement

- Computation in R consists of sequentially evaluating statements.
- Statements, such as x<-1:10 or mean(y), can be separated by either a semi-colon or a new line.
- Whenever the evaluator is presented with a syntactically complete statement that statement is evaluated and the value returned.
- The result of evaluating a statement can be referred to as the value of the statement. The value can always be assigned to a symbol.
- Both semicolons and new lines can be used to separate statements. A semicolon always indicates the end of a statement while a new line may indicate the end of a statement. If the current statement is not syntactically complete new lines are simply ignored by the evaluator. If the session is interactive the prompt changes from '>' to '+'.

```
> x <- 0; x + 5          > y <- c(1,2,3,
[1] 5                    + 4,5,6)
                         > y
                         [1] 1 2 3 4 5 6
```

Source: https://cran.r-project.org/doc/manuals/r-release/R-lang.html

Block (Grouped Expression)

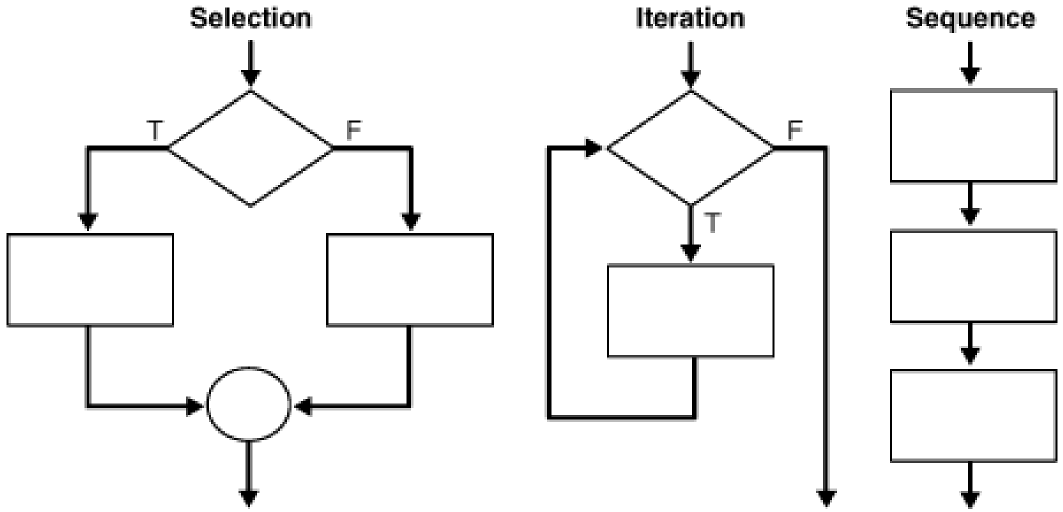
- Statements can be grouped together in braces, {expr_1; ...; expr_m}
- A group of statements is sometimes called a block.
- Blocks are not evaluated until a new line is entered after the closing brace “}”.
- The value of the block is the result of the last expression in the block evaluated.

```
b = {a <- 0
  a + 5}
```

```
> b = {a <- 0
+ a + 5}
> b
[1] 5
```

Structure Theorem

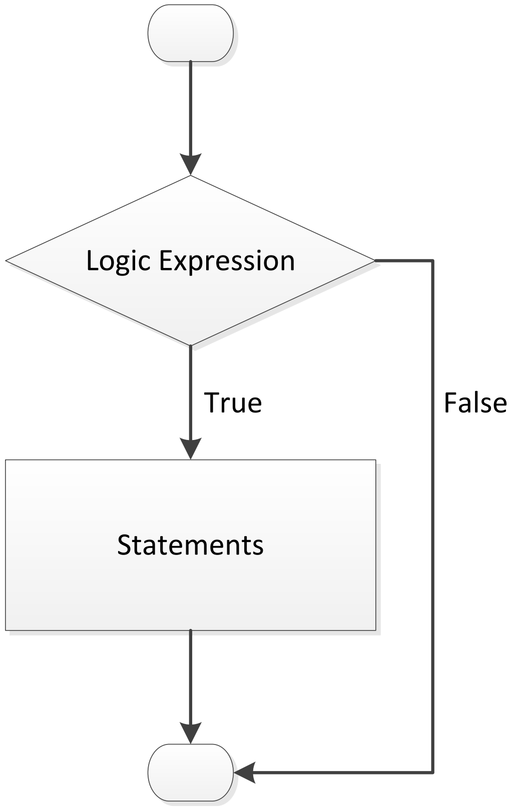
- According to the *structure theorem*, any computable program can be written using three basic control structures:
  - Sequence: executing one subprogram, and then another subprogram
  - Selection: executing one of two subprograms according to the value of a boolean expression
  - Iteration (loop): executing a subprogram until a boolean expression is true



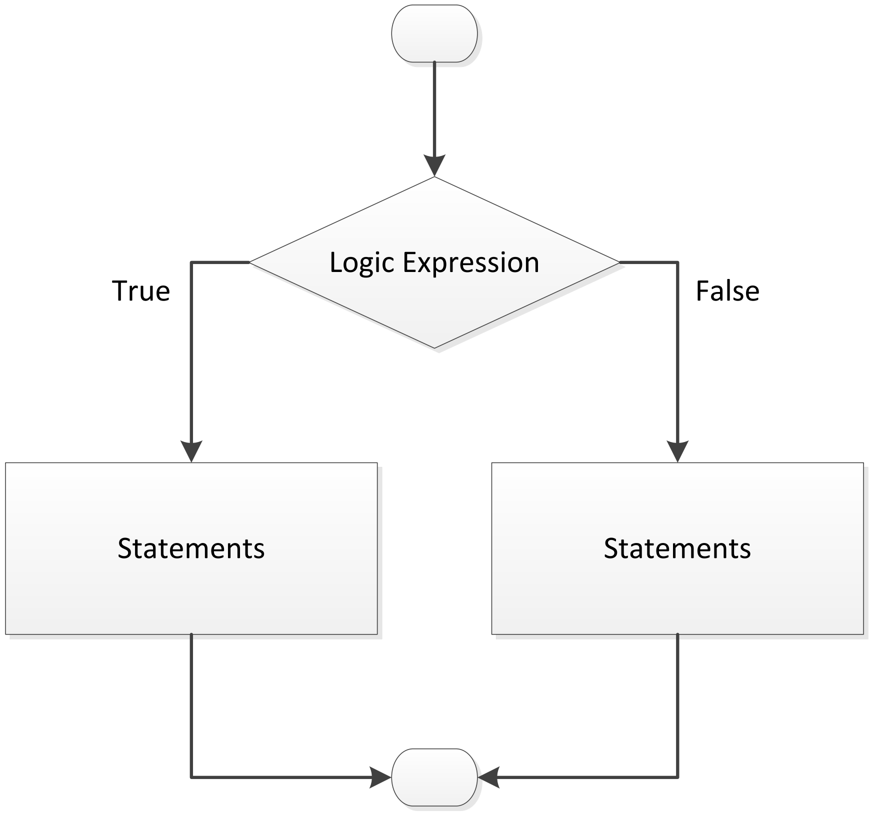
Reading: http://en.wikipedia.org/wiki/Structured_program_theorem

Selection Structure

One-way selection structure



Two-way selection structure



One-Way Selection Structure in R

- Syntax

  if(logic expression) {…}

Function

```
is.even <- function(x){
  if(x%%2==0){
    return(TRUE)
  }
}
```

Test

```
> is.even(24)
[1] TRUE
> is.even(23)
> is.even(10.5)
```

Two-Way Selection Structure in R

- Syntax

  if(logic expression) {…} else {…}

Function

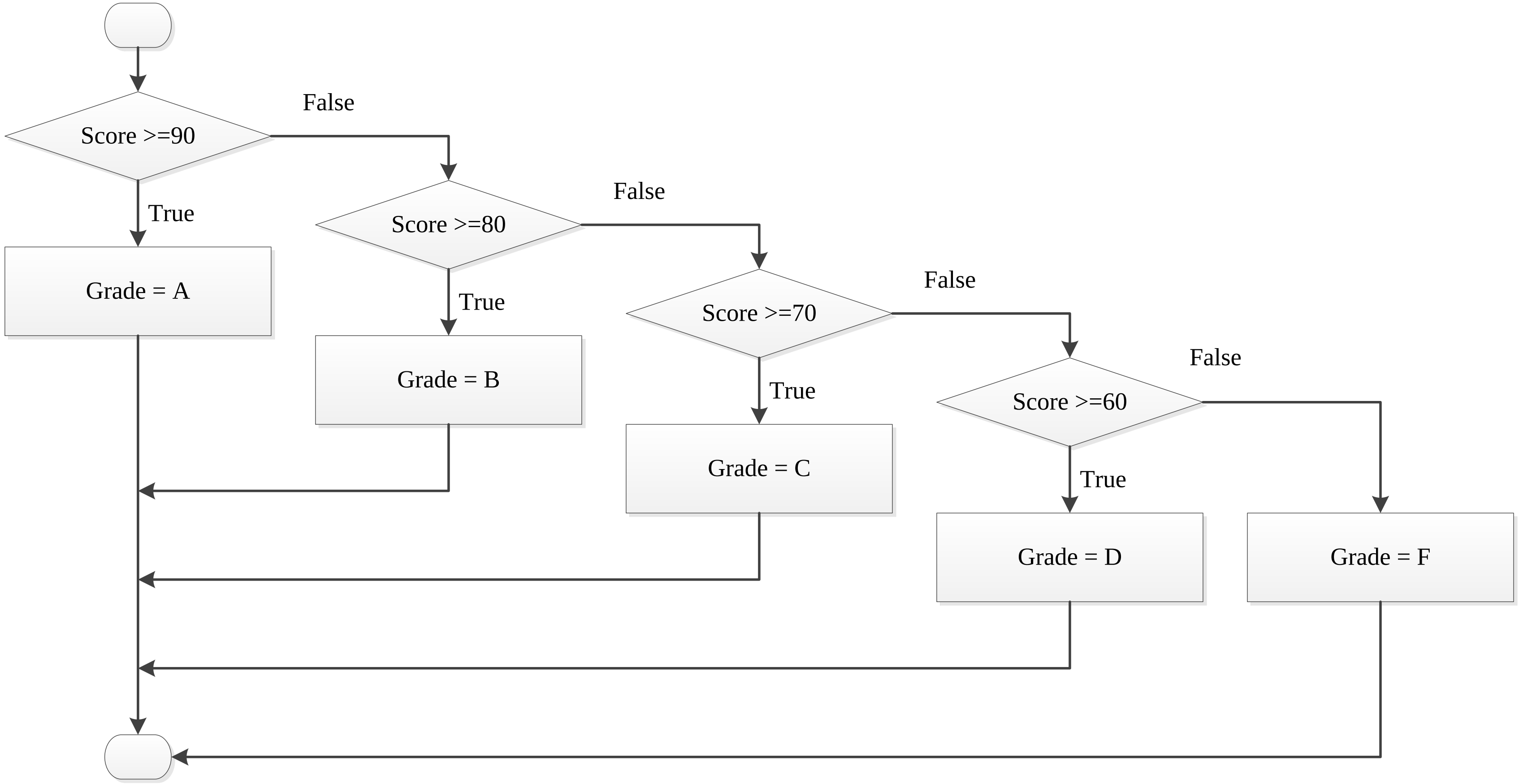
```
is.even2 <- function(x){
  if(x%%2==0){
    return(TRUE)
  } else{
    return(FALSE)
  }
}
```

Test

```
> is.even2(24)
[1] TRUE
> is.even2(23)
[1] FALSE
> is.even2(10.5)
[1] FALSE
```

Multi-Way Selection Structure

- Convert score to letter grade



(cont.)

Function

```
score2grade <- function(score){
  if(score >= 90) return("A")
  else if (score >=80) return("B")
        else if (score >= 70) return("C")
              else if (score >= 60) return("D")
                   else return("F")
}
```

Test

```
> score2grade(99)
[1] "A“
> score2grade(90.1)
[1] "A“
> score2grade(89.9)
[1] "B“
> score2grade(70.1)
[1] "C“
> score2grade(68.6)
[1] "D“
> score2grade(57)
[1] "F"
```

The Nearest Rule (if else ambiguity)

The else clause matches the nearest preceding if clause in the same block.

Motivations for Looping

- Suppose that you are given scores for 100 students, you need to calculate letter grades for them.
  - Scores are stored in a vector called score_v.
  - Grades should be stored in a vector called grade_v.
- How do you solve this problem?

Opening Problem

- Problem: it would be tedious to write 100 statements

100
times

```
grade_v[1] = score2grade(score_v[1])
grade_v[2] = score2grade(score_v[2])
grade_v[3] = score2grade(score_v[3])
grade_v[4] = score2grade(score_v[4])
grade_v[5] = score2grade(score_v[5])
grade_v[6] = score2grade(score_v[6])
...

...

...
grade_v[98] = score2grade(score_v[98])
grade_v[99] = score2grade(score_v[99])
grade_v[100] = score2grade(score_v[100])
```

Introducing Loops

```
grade_v <- NULL

for (i in 1:100)
  grade_v[i] = score2grade(score_v[i])
```

- In general, loop constructs control repeated executions of a block of statements.

Loop Structure in R

- R provides three statements to support looping
  - for statement
  - while statement
  - repeat statement
- Two statements used to explicitly control looping
  - break statement
  - next statement

for Loop

- Syntax

```
for (name in vector)
    statement
```

```
# Generate random scores for 100 students
score_v <- sample(50:100, 100, replace=T)
print(score_v)

# Use for loop
grade_v <- NULL  # Initiate a grade vector
for (i in 1:100)
    grade_v[i] = score2grade(score_v[i])
print(grade_v)  # Show the grades calculated
```

while Loop

- Syntax

```
while (logic expression)
    statement
```

```
# Use while loop
grade_v <- NULL
i <- 1
while (i <= 100){
  grade_v[i] = score2grade(score_v[i])
  i <- i + 1
}
print(grade_v)  # Show the grades calculated
```

repeat Loop

- Syntax

  repeat statement

```
# Use repeat loop
grade_v <- NULL
i <- 1
repeat {
  grade_v[i] = score2grade(score_v[i])
  i <- i + 1
  if (i == 101) break
}
print(grade_v)  # Show the grades calculated
```

Which Loop to Use?

- The three forms of loop statements, for, while, and repeat, are expressively equivalent.
- You can write a loop in any of these three forms.

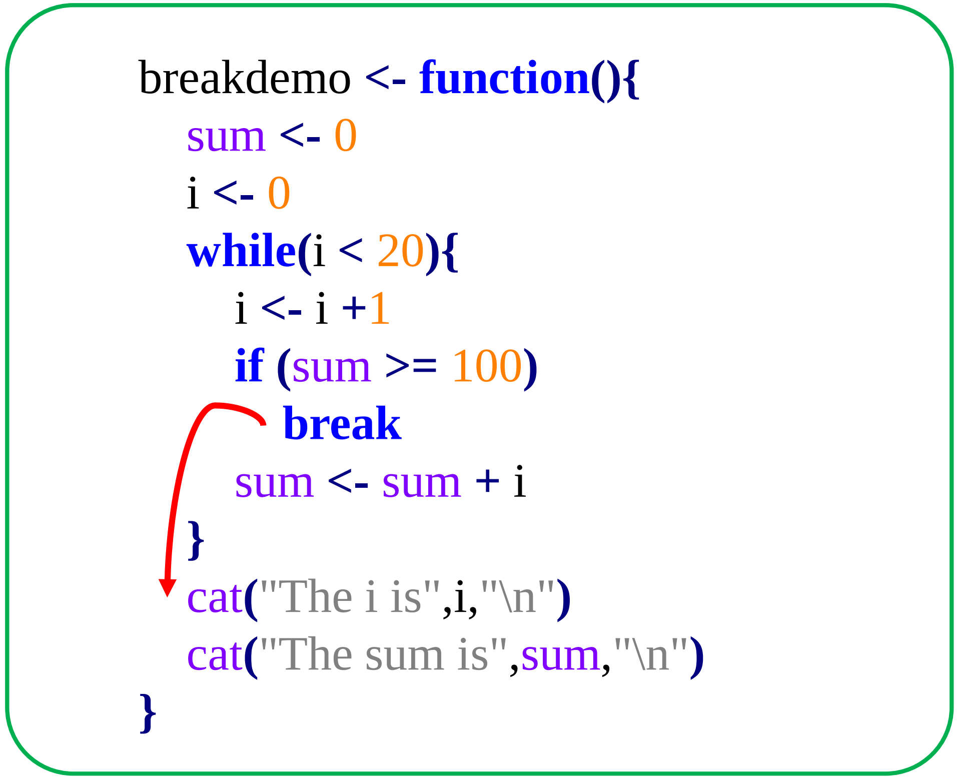
Guidelines for Choosing Loop Structures

- Use the one that is most intuitive and comfortable for you.
- In general, a for loop may be used if the number of repetitions is known, as, for example, when you need to print a message 100 times.
- A while loop may be used if the number of repetitions is not known, as in the case of reading the numbers until the input is 0.
- A repeat loop can be used to replace a while loop if the loop body has to be executed before testing the continuation condition.

Using break and next

- The break and next keywords provide additional controls in a loop.
- break statement breaks out of the loop.
- continue statement bypasses the current iteration.

break Statement



```
breakdemo <- function(){
  sum <- 0
  i <- 0
  while(i < 20){
    i <- i +1
    if (sum >= 100)
      break
    sum <- sum + i
  }
  cat("The i is",i,"\n")
  cat("The sum is",sum,"\n")
}
```

```
> breakdemo()
The i is 15
The sum is 105
```

break statement breaks out of the loop.

sum = 1 + 2 + 3 + ...+14 = 105

next Statement

```
nextdemo <- function(){
    sum <- 0
    i <- 0
    while(i < 20){
        i <- i + 1
        if (i == 10 | i ==11)
            next
        sum <- sum + i
    }
    cat("The i is",i,"\n")
    cat("The sum is",sum,"\n")
}
```

```
> nextdemo()
The i is 20
The sum is 189
```

next statement bypasses the current iteration.

sum = 1 + 2 + ...+ 8 + 9 + 12 + 13 + ... + 20 = 189

R Programming Style and Debug

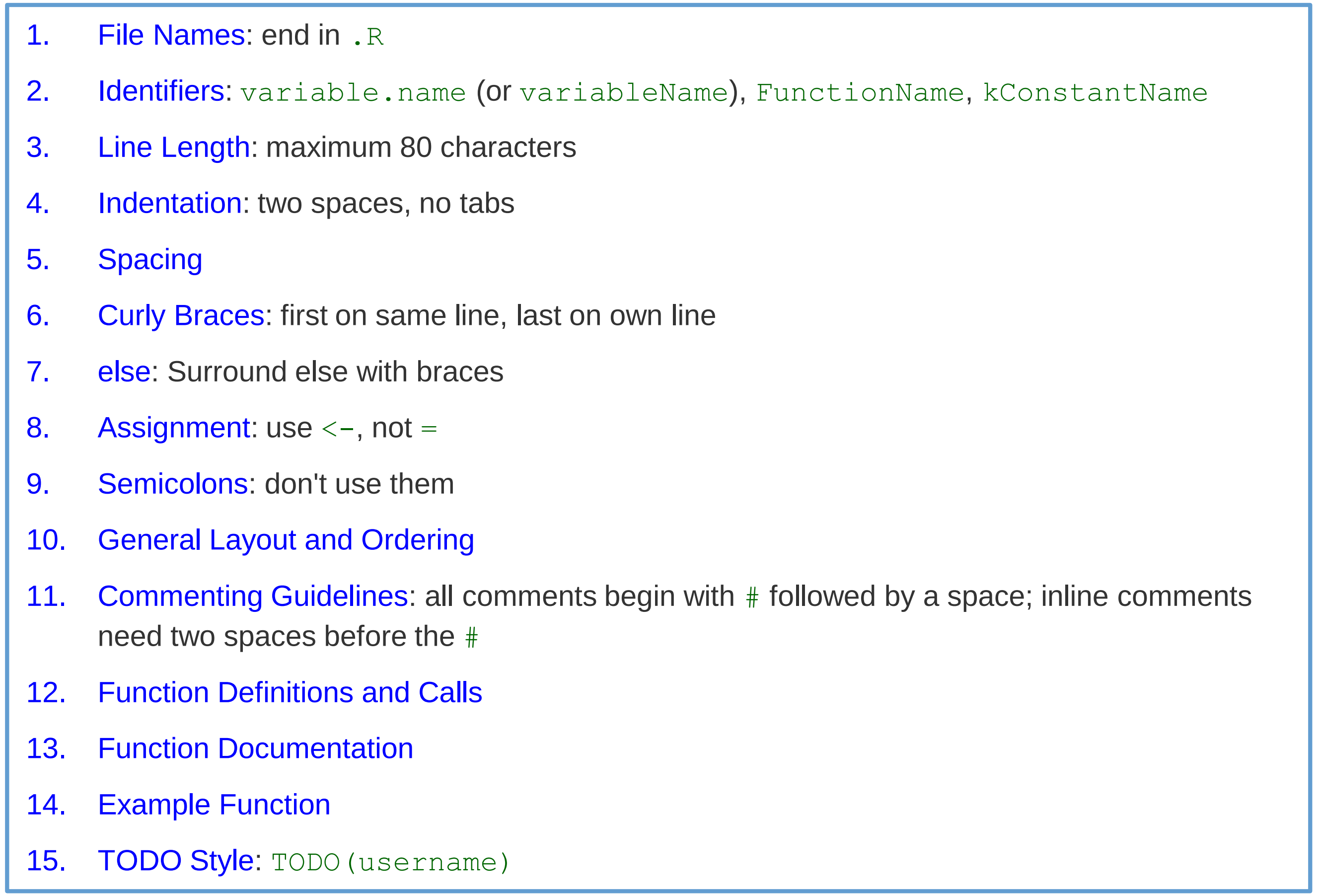
Programming Style and Documentation

- Programming style is important
  - Good programming style makes a program more readable
  - Good programming style helps reduce programming errors

- Several guidelines
  - Appropriate Comments
  - Naming Conventions
  - Proper Indentation and Spacing Lines

Google's R Style Guide

- https://google.github.io/styleguide/Rguide.xml
- Summary

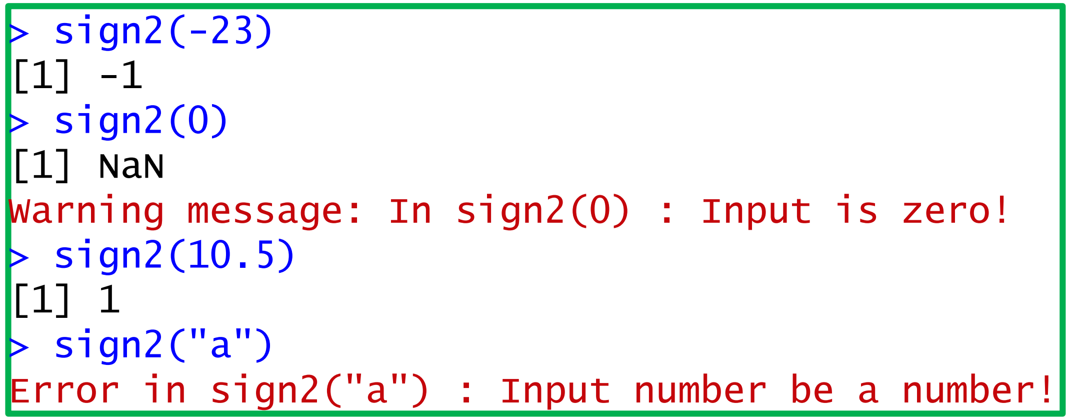


1. File Names: end in `.R`
2. Identifiers: `variable.name` (or `variableName`), `FunctionName`, `kConstantName`
3. Line Length: maximum 80 characters
4. Indentation: two spaces, no tabs
5. Spacing
6. Curly Braces: first on same line, last on own line
7. else: Surround else with braces
8. Assignment: use `<-`, not `=`
9. Semicolons: don't use them
10. General Layout and Ordering
11. Commenting Guidelines: all comments begin with `#` followed by a space; inline comments need two spaces before the `#`
12. Function Definitions and Calls
13. Function Documentation
14. Example Function
15. TODO Style: `TODO(username)`

Display Messages in R

- stop(): provide error messages and stop execution
- warning(): provide warning message and continue execution
- message(): provide a general message and continue execution

```
sign2 <- function(x){
  if(is.numeric(x)){
    if (x>0){
      return(1);
    }
    if (x==0){
      warning("Input is zero!")
      return(NaN)
    } else {
      return(-1)
    }

  } else{
    stop("Input number be a number!")
  }
}
```



```
> sign2(-23)
[1] -1
> sign2(0)
[1] NaN
Warning message: In sign2(0) : Input is zero!
> sign2(10.5)
[1] 1
> sign2("a")
Error in sign2("a") : Input number be a number!
```

Programming Errors

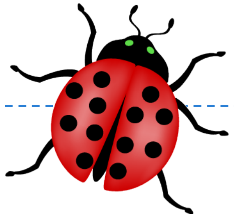
- Syntax Errors
  - You type a command that R cannot understand
  - For example, missing commas, unmatched parentheses, wrong function name etc.

```
> a <- c(0,1,2,3]
Error: unexpected ']' in "a <- c(0,1,2,3]“
> a <- c(0,1,2 3)
Error: unexpected numeric constant in "a <- c(0,1,2 3"
```

- Runtime Errors
  - Causes the program to abort

- Logic Errors
  - The program can run, but produce wrong results
  - Difficult to find

Debugging



- Logic errors are called *bugs*.
- The process of finding and correcting errors is called debugging.
- A common approach to debugging is to use a combination of methods to narrow down to the part of the program where the bug is located.
    - For a short and simple program, you can hand-trace the program (i.e., catch errors by reading the program), or you can insert print statements in order to show the values of the variables or the execution flow of the program.
    - For a large, complex program, the most effective approach for debugging is to use a debugger utility.

Debugger

- Debugger is a program that facilitates debugging. You can use a debugger to
  - Execute a single statement at a time.
  - Trace into or stepping over a function.
  - Set breakpoints.
  - Display variables.
  - Display call stack.

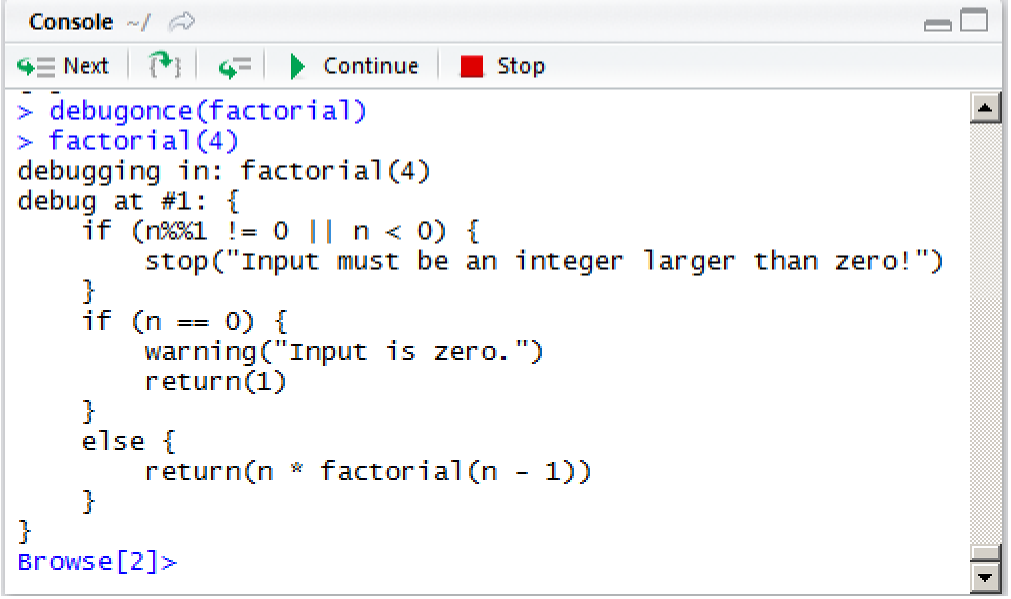
Debugging in R

- Call debug() or debugonce() function to go to debug mode

Function Definition

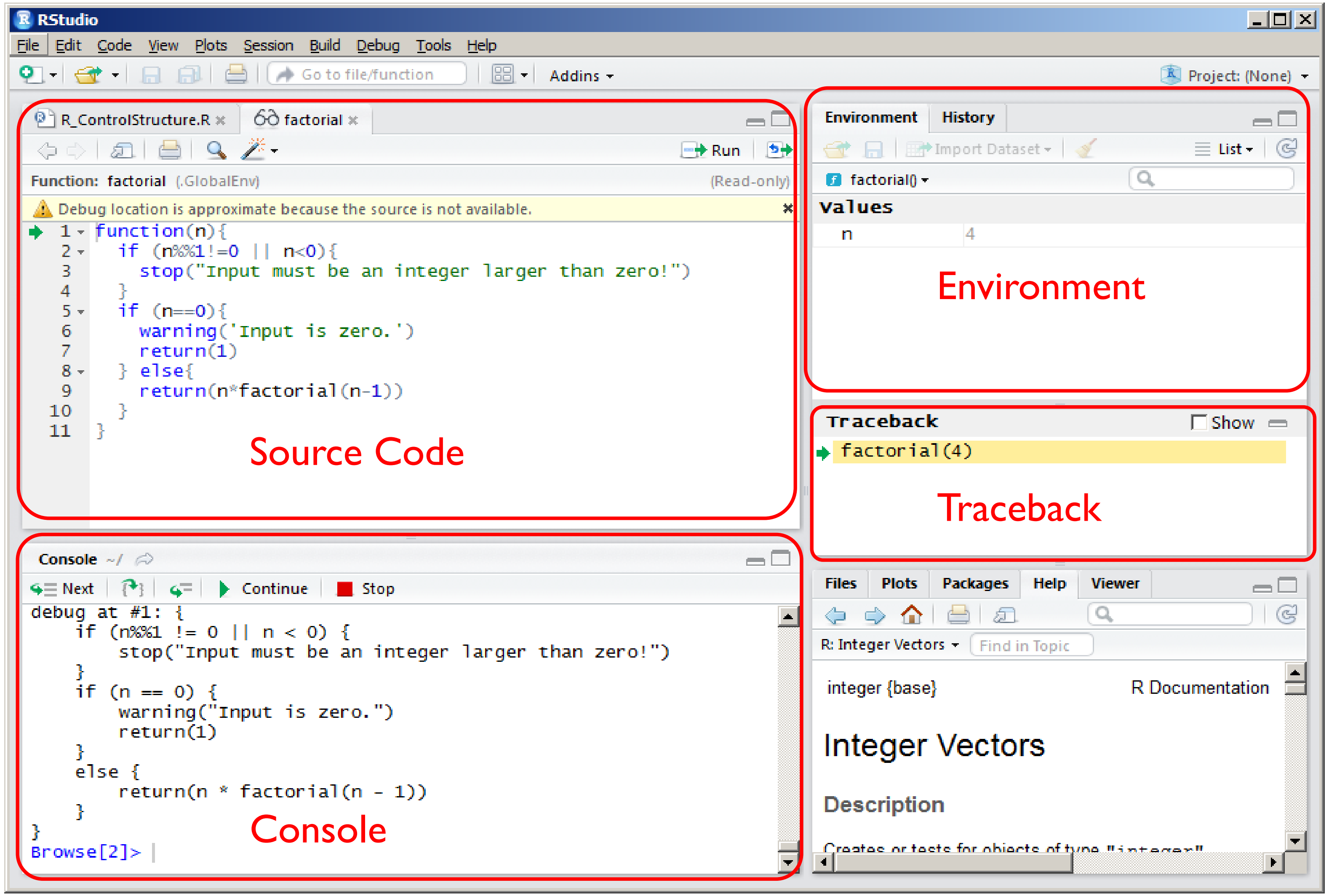
```
factorial <- function(n=3){
  if (n%%1!=0 || n<0){
    stop("Input must be an
integer larger than zero!")
  }
  if (n==0){
    return(1)
  } else{
    return(n*factorial(n-
1))
  }
}
```

Debug Mode



```
> debugonce(factorial)
> factorial(4)
debugging in: factorial(4)
debug at #1: {
    if (n%%1 != 0 || n < 0) {
        stop("Input must be an integer larger than zero!")
    }
    if (n == 0) {
        warning("Input is zero.")
        return(1)
    }
    else {
        return(n * factorial(n - 1))
    }
}
Browse[2]>
```

R Debugger Overview



Function: factorial (.GlobalEnv) (Read-only)

Debug location is approximate because the source is not available.

```
function(n){
  if (n%%1!=0 || n<0){
    stop("Input must be an integer larger than zero!")
  }
  if (n==0){
    warning('Input is zero.')
    return(1)
  } else{
    return(n*factorial(n-1))
  }
}
```

Values

n	4

```
factorial(4)
```

```
debug at #1: {
    if (n%%1 != 0 || n < 0) {
        stop("Input must be an integer larger than zero!")
    }
    if (n == 0) {
        warning("Input is zero.")
        return(1)
    }
    else {
        return(n * factorial(n - 1))
    }
}
Browse[2]>
```

integer {base} R Documentation

Integer Vectors

Description

Creates or tests for objects of type "integer".

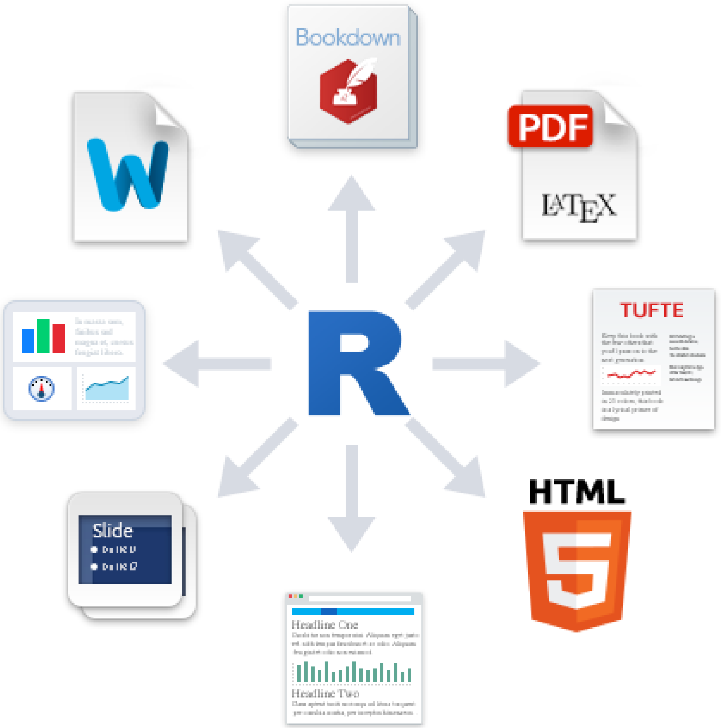
Dynamic Report: R Markdown

Dynamic Documents in R

- "R Markdown is an authoring format that enables easy creation of dynamic documents, presentations, and reports from R".

- R code embedded in text
  - You can write R code in plain text and generate data analysis reports in various formats such as HTML, PDF, Word, HTML5 slides.

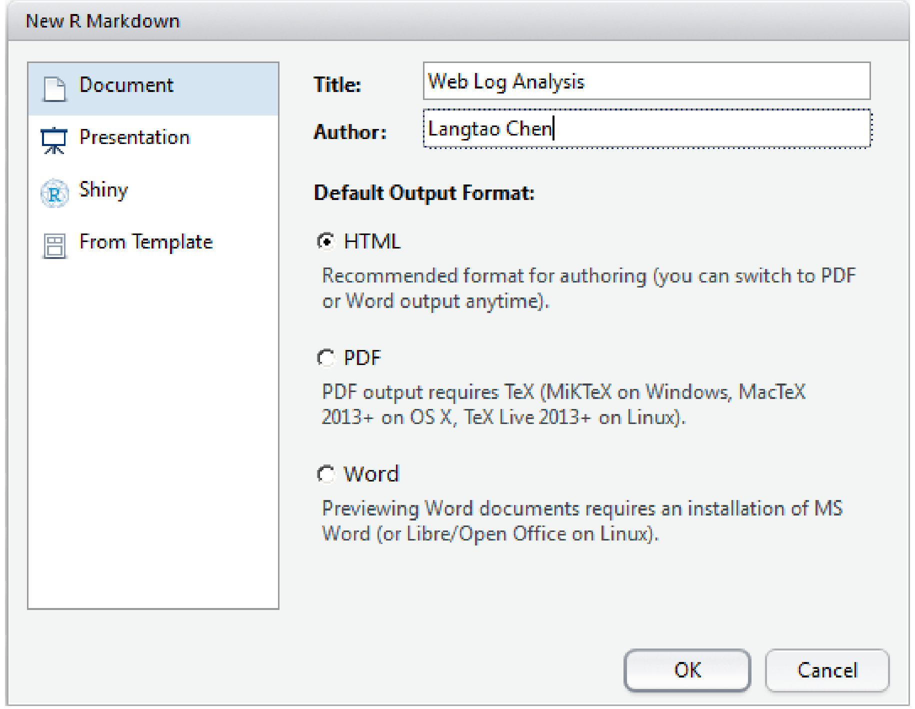
- Reproducible analysis
  - You can easily reproduce the data analysis results after the data and/or code change.



Source: http://rmarkdown.rstudio.com/

Use R Markdown

- Install R markdown package
  - install.packages("rmarkdown")
- In Rstudio, click "File -> New File -> R Markdown…" menu
- In the popup window, input header information, then click "OK" button



New R Markdown

Title: Web Log Analysis

Author: Langtao Chen

Default Output Format:

● HTML

Recommended format for authoring (you can switch to PDF or Word output anytime).

○ PDF

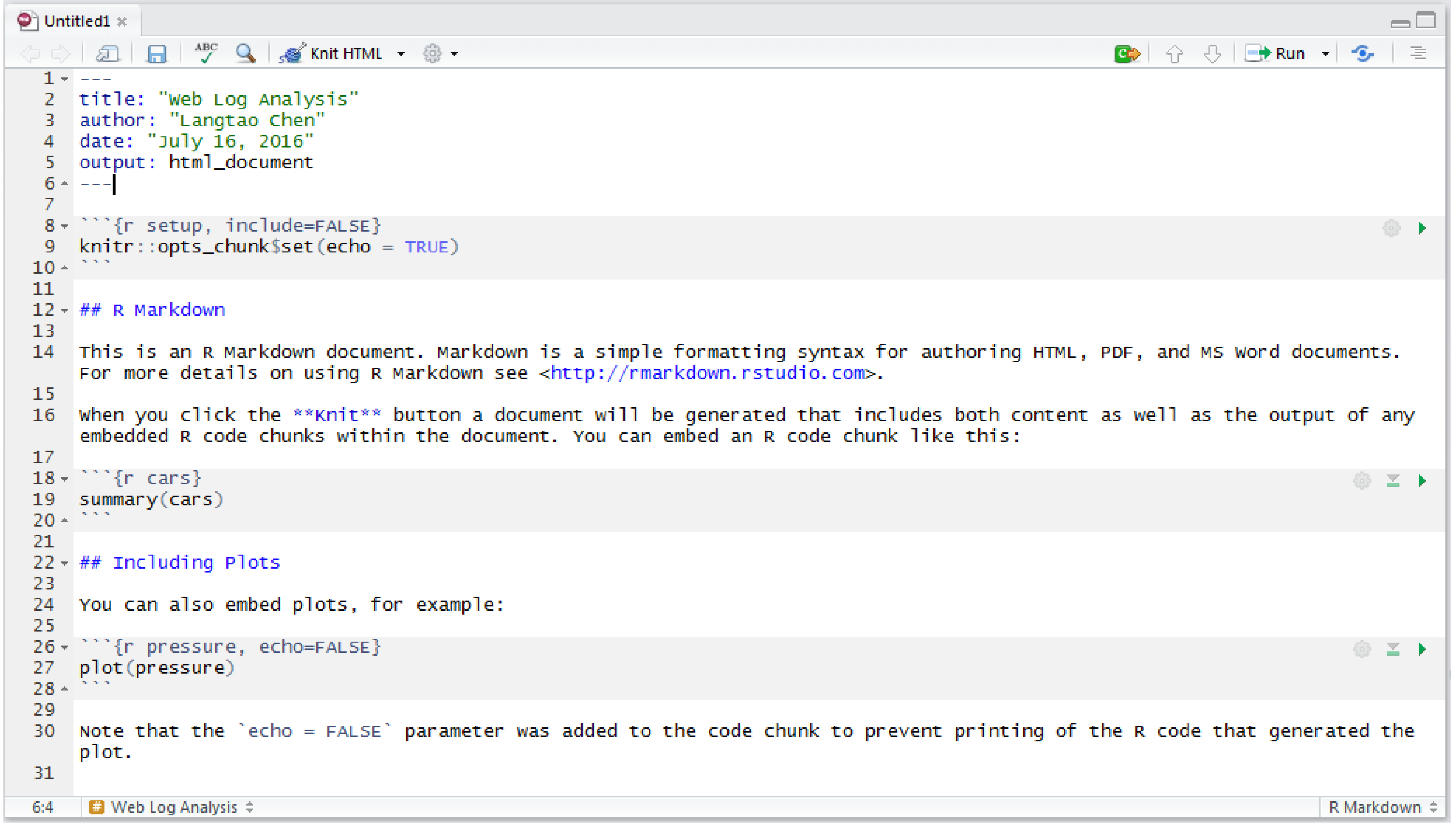
PDF output requires TeX (MiKTeX on Windows, MacTeX 2013+ on OS X, TeX Live 2013+ on Linux).

○ Word

Previewing Word documents requires an installation of MS Word (or Libre/Open Office on Linux).

(cont.)

- RStudio generates a sample R markdown (.Rmd) file for you



~~~
---
title: "Web Log Analysis"
author: "Langtao Chen"
date: "July 16, 2016"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
~~~

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```{r cars}
summary(cars)
```

Including Plots

You can also embed plots, for example:

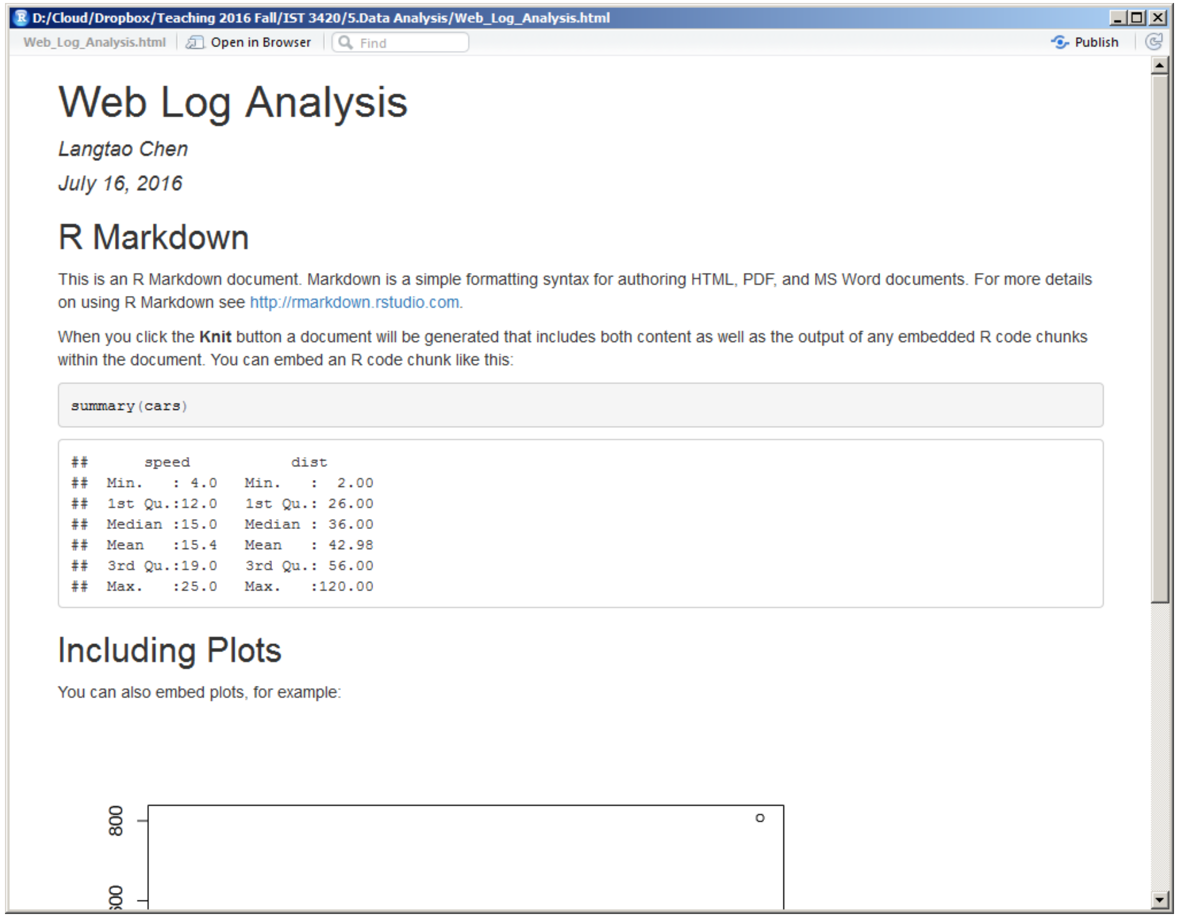
```{r pressure, echo=FALSE}
plot(pressure)
```

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

```
```

(cont.)

- Click the "Knit HTML" button Knit HTML on the toolbar to generate the HTML report


D:/Cloud/Dropbox/Teaching 2016 Fall/IST 3420/5.Data Analysis/Web_Log_Analysis.html
Web_Log_Analysis.html
Open in Browser
Find
Publish
800
600

Web Log Analysis

Langtao Chen

July 16, 2016

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00  
```

Including Plots

You can also embed plots, for example:

(cont.)

- Follow the R markdown syntax demonstrated in the sample file, write your own data analysis by editing the Rmd template.

R Markdown Syntax Summary

- YAML Header (key: value pairs)
  - At the beginning of Rmd file
  - Between lines of ---
- Plain Text Format
  - Headers: Begin with #
  - Lists: Begin with -
  - LaTex or MathML equations: Enclosed within $
- Embedded R Code
  - R Code Chunks: Begin with ```{r} and end with ```
  - Inline R Code: Begin with `r and end with `

An Example

- R Markdown File

  R Markdown Sample.Rmd

- PDF Output

  R Markdown Sample.pdf

Reference

- "An Introduction to R"
  - https://cran.r-project.org/doc/manuals/R-intro.pdf
- R Language Definition
  - https://cran.r-project.org/doc/manuals/r-release/R-lang.pdf
- Base R Cheat Sheet
  - http://www.rstudio.com/wp-content/uploads/2016/06/r-cheat-sheet.pdf
- Advanced R Cheat Sheet
  - http://www.rstudio.com/wp-content/uploads/2016/02/advancedR.pdf
- R Markdown Cheat Sheet
  - http://www.rstudio.com/wp-content/uploads/2016/03/rmarkdown-cheatsheet-2.0.pdf
- R Markdown Reference Guide
  - http://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf

Q & A