

Introduction to Data Analytics:

R

Dr. Irene Vrbik

University of British Columbia Okanagan

`irene.vrbik@ubc.ca`

What is R?

R is a free and open source programming language for statistical computing and graphics.

1. One of the most widely used programming languages for statistical analysis.
2. Popular in academia and companies like Microsoft, Google, and Facebook.
3. There are currently over 8000 packages in R. cran.r-project.org/web/packages/available_packages_by_name.html
4. R creates high quality graphs and visualizations.

Why learn R?

R is built to handle and analyze data.

Example

Filtering a dataset to be within a lower and upper bound and then calculating summary statistics.

Python

```
1  for v in data:
2      if lower <= v <= upper:
3          maxdata = max(v, maxdata)
4          mindata = min(v, mindata)
5      sumdata += v
6      count += 1
```

Why learn R?

R is built to handle and analyze data.

Example

Filtering a dataset to be within a lower and upper bound and then calculating summary statistics.

R

```
1 new_data <- subset(data, x <= upper & x >= lower)
2 sumdata <- sum(new_data)
3 count <- length(new_data)
4 maxdata <- max(new_data)
5 mindata <- min(new_data)
```

Statistics Review: Types of Data

1. Qualitative (Categorical)

1.1 Descriptions or groups

1.2 Can be characters or numbers

1.3 Observed and not measured

1.4 i.e. names, labels, categories, properties

2. Quantitative (Numeric)

2.1 Strictly numeric

2.2 Can be measured

2.3 i.e. height, weight, speed, counts, temperature, volume

Numerical Summaries

A **numerical summary** provides an overview of data to help understand it without examining all data values.

A **measure of centre** and a **measure of spread** to describe quantitative data. There are no numerical summaries for qualitative data, only graphical summaries.

Measures of Centre

Definition (Mean)

The **mean** is the average of data values (sum of values divided by count):

$$\bar{y} = \frac{\sum_{i=1}^n y_i}{n}$$

Definition (Median)

The **median** is the value at which half of the data lies above that value and half lies below it.

1. Odd number of observations: \tilde{y} is the k th value where $k = \frac{n+1}{2}$.
2. Even number of observations: \tilde{y} is the mean of the k th and $(k+1)$ terms, where $k = \frac{n}{2}$.

Example

Let the data be given by $y = \{1, 3, 3, 7, 9\}$. The mean is

$$\bar{y} = \frac{1 + 3 + 3 + 7 + 9}{5} = 4.6$$

and the median is

$$\tilde{y} = 3$$

for which R provides `mean()` and `median()`:

```
> mean(y)
```

```
[1] 4.6
```

```
> median(y)
```

```
[1] 3
```


Measures of Spread

A measure of spread indicates how far apart the values are.

Definition (Variance)

Variance is the sum of the squares of each data point's distance from the mean:

$$s^2 = \frac{\sum_{i=1}^n (y_i - \bar{y})^2}{n - 1} = \frac{\sum_{i=1}^n y_i^2 - n\bar{y}^2}{n - 1}$$

Definition (Standard Deviation)

Standard Deviation is the square root of the variance: $s = \sqrt{s^2}$

Definition (Range)

Range is the maximum value minus minimum value: $\max(y) - \min(y)$.

Example (Standard Deviation)

Let $y = (1, 3, , 3, 7, 9)$ then the **variance** is

$$s^2 = \frac{(1 + 9 + 9 + 49 + 81) - (5 \cdot 4.6^2)}{5 - 1} = 10.8$$

and therefore the **standard deviation** is

$$s = 3.286$$

for which R provides `mean()` and `median()`:

```
> var(y)
```

```
[1] 10.8
```

```
> median(y)
```

```
[1] 3.286335
```

Question

Let $y = (1, 3, 4, 7, 11, 28)$. How many of the are following statements are true?

1. $\bar{y} = \tilde{y}$
2. $\bar{y} = 9$
3. $s^2 = 3.5$
4. $\text{range} = 6$.

A) 0 B) 1 C) 2 D) 3 E) 4

Question

Let $y = (1, 3, 4, 7, 11, 28)$. How many of the are following statements are true?

1. $\bar{y} = \tilde{y}$,
2. $\bar{y} = 9$,
3. $s^2 = 3.5$,
4. $\text{range} = 6$.

A) 0 B) 1 C) 2 D) 3 E) 4

Question

Let $y = (1, 3, 4, 7, 11, 28)$. How many of the are following statements are true?

1. $\bar{y} = \tilde{y}$, **X**

2. $\bar{y} = 9$,

3. $s^2 = 3.5$,

4. $\text{range} = 6$.

A) 0

B) 1


C) 2


D) 3

E) 4

Question

Let $y = (1, 3, 4, 7, 11, 28)$. How many of the are following statements are true?

1. $\bar{y} = \tilde{y}$, 

2. $\bar{y} = 9$, 

3. $s^2 = 3.5$,

4. $\text{range} = 6$.

A) 0

B) 1

C) 2

D) 3

E) 4

Question

Let $y = (1, 3, 4, 7, 11, 28)$. How many of the are following statements are true?

1. $\bar{y} = \tilde{y}$, ✗

2. $\bar{y} = 9$, ✓

3. $s^2 = 3.5$, ✗

4. $\text{range} = 6$.

A) 0

B) 1

C) 2

D) 3

E) 4

Question

Let $y = (1, 3, 4, 7, 11, 28)$. How many of the are following statements are true?

1. $\bar{y} = \tilde{y}$, **X**

2. $\bar{y} = 9$, **✓**

3. $s^2 = 3.5$, **X**

4. $\text{range} = 6$. **X**

A) 0

B) 1

C) **2**

D) 3

E) 4

Quantiles and Quartiles

The **q th quantile** is the point where at least $q \cdot 100\%$ of the data values are at or below the value.

There are some special quantiles called **quartiles** (quarters of the data).

Q1 First quartile. 0.25 quantile.

Q2 Second quartile. 0.5 quantile. Median.

Q3 Third quartile. 0.75 quantile.

The **interquartile range** is the difference between Q3 and Q1.

$$\text{IQR} = Q3 - Q1$$

It contains the center 50% of the data.

Example

Let the data $y = \{1, 2, 3, 4, 5, 6\}$ and median $\tilde{y} = \frac{3+4}{2} = 3.5$.

Q1 and Q3 are then the **medians** of the two subsets of data when divided at the median:

$$y_1 = \{1, 2, 3\} \quad y_2 = \{4, 5, 6\} \quad Q1 = 2 \quad Q3 = 5$$

The function `quantile()` is provided in R.

Question

Let $y = \{0, 1, \dots, 100\}$. How many of the following are true?

1. The median is 50 and Q3 is 75.
2. Each integer y_i is the $y_i/100$ th quantile. i.e. 4 is the 0.05th quantile.
3. For every data set, Q2 is strictly less than Q3.
4. If the data is reversed the quantile values remain unchanged.

A) 0 B) 1 C) 2 D) 3 E) 4

Question

Let $y = \{0, 1, \dots, 100\}$. Which of the following are true?

1. The median is 50 and Q3 is 75.
2. Each integer y_i is the $y_i/100$ th quantile. i.e. 4 is the 0.05th quantile.
3. For every data set, Q2 is strictly less than Q3.
4. If the data is reversed the quantile values remain unchanged.

A) 0 B) 1 C) 2 D) 3 E) 4

Question

Let $y = \{0, 1, \dots, 100\}$. Which of the following are true?

1. The median is 50 and Q3 is 75. ✓
2. Each integer y_i is the $y_i/100$ th quantile. i.e. 4 is the 0.05th quantile.
3. For every data set, Q2 is strictly less than Q3.
4. If the data is reversed the quantile values remain unchanged.

A) 0 B) 1 C) 2 D) 3 E) 4

Question

Let $y = \{0, 1, \dots, 100\}$. Which of the following are true?

1. The median is 50 and Q3 is 75. ✓
2. Each integer y_i is the $y_i/100$ th quantile. i.e. 4 is the 0.05th quantile. ✓
3. For every data set, Q2 is strictly less than Q3.
4. If the data is reversed the quantile values remain unchanged.

A) 0 B) 1 C) 2 D) 3 E) 4

Question

Let $y = \{0, 1, \dots, 100\}$. Which of the following are true?

1. The median is 50 and Q3 is 75. ✓
2. Each integer y_i is the $y_i/100$ th quantile. i.e. 4 is the 0.05th quantile. ✓
3. For every data set, Q2 is strictly less than Q3. ✗
4. If the data is reversed the quantile values remain unchanged.

A) 0 B) 1 C) 2 D) 3 E) 4

Question

Let $y = \{0, 1, \dots, 100\}$. Which of the following are true?

1. The median is 50 and Q3 is 75. ✓
2. Each integer y_i is the $y_i/100$ th quantile. i.e. 4 is the 0.05th quantile. ✓
3. For every data set, Q2 is strictly less than Q3. ✗
If data is comprised of the same number repeated then $Q2 = Q3$.
4. If the data is reversed the quantile values remain unchanged.

A) 0 B) 1 C) 2 D) 3 E) 4

Question

Let $y = \{0, 1, \dots, 100\}$. Which of the following are true?

1. The median is 50 and Q3 is 75. ✓
2. Each integer y_i is the $y_i/100$ th quantile. i.e. 4 is the 0.05th quantile. ✓
3. For every data set, Q2 is strictly less than Q3. ✗
If data is comprised of the same number repeated then $Q2 = Q3$.
4. If the data is reversed the quantile values remain unchanged. ✓

A) 0 B) 1 C) 2 D) 3 E) 4

Five Number Summary

A **five number summary** consists of the following **in this order**:

1. minimum,
2. Q1,
3. median,
4. Q3, and
5. maximum.

Example

Using $y = 1, 2, 3, 4, 5, 6$ the five number summary would be

```
> y <- c(1,2,3,4,5,6)
> fivesum(y)
[1] 1.0 2.0 3.5 5.0 6.0
```

Question

Which of the following are true?

1. Variance is always non negative.
2. Standard deviation can be zero.
3. If $a > b$ then $\text{quantile}(a) \geq \text{quantile}(b)$.
4. The five number summary provides the mean of the dataset.

A) 0

B) 1

C) 2

D) 3

E) 4

Question

Which of the following are true?

1. Variance is always non negative.
2. Standard deviation can be zero.
3. If $a > b$ then $\text{quantile}(a) \geq \text{quantile}(b)$.
4. The five number summary provides the mean of the dataset.

A) 0

B) 1

C) 2

D) 3

E) 4

Question

Which of the following are true?

1. Variance is always non negative. ✓
2. Standard deviation can be zero.
3. If $a > b$ then $\text{quantile}(a) \geq \text{quantile}(b)$.
4. The five number summary provides the mean of the dataset.

A) 0

B) 1

C) 2

D) 3

E) 4

Question

Which of the following are true?

1. Variance is always non negative. ✓
2. Standard deviation can be zero. ✓
3. If $a > b$ then $\text{quantile}(a) \geq \text{quantile}(b)$.
4. The five number summary provides the mean of the dataset.

A) 0

B) 1

C) 2

D) 3

E) 4

Question

Which of the following are true?

1. Variance is always non negative. ✓
2. Standard deviation can be zero. ✓
3. If $a > b$ then $\text{quantile}(a) \geq \text{quantile}(b)$. ✓
4. The five number summary provides the mean of the dataset.

A) 0

B) 1

C) 2

D) 3

E) 4

Question

Which of the following are true?

1. Variance is always non negative. ✓
2. Standard deviation can be zero. ✓
3. If $a > b$ then $\text{quantile}(a) \geq \text{quantile}(b)$. ✓
4. The five number summary provides the mean of the dataset. ✗

A) 0

B) 1

C) 2

D) 3

E) 4

Question

Which of the following are true?

1. Variance is always non negative. ✓
2. Standard deviation can be zero. ✓
3. If $a > b$ then $\text{quantile}(a) \geq \text{quantile}(b)$. ✓
4. The five number summary provides the mean of the dataset. ✗

A) 0 B) 1 C) 2 D) 3 E) 4

Note the \geq of 3.

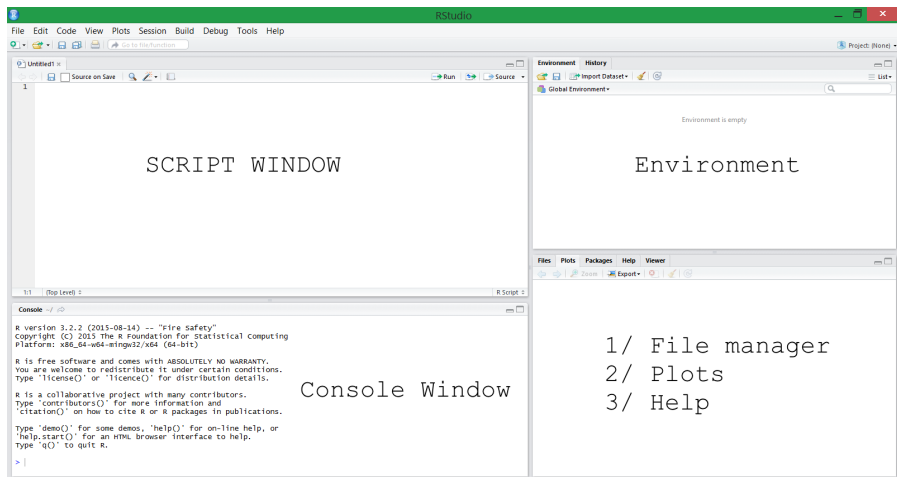
RStudio

RStudio is an integrated development environment (IDE) for R.

Install R first! `cran.rstudio.com/`

Download RStudio at: `rstudio.com/products/rstudio/download/`

RStudio Environment



RStudio IDE

Script Window

Draft and save code. Write a script to run in the console by CTRL+R or CTRL+Enter, or pressing Run.

Console

Where the code goes once run. Shows **input (blue)**, output (black) and any **errors or warnings (red)**. (different "Themes" can be chosen to display different colours.

Environment

Shows saved variables and datasets

File Browser/Plots/Help

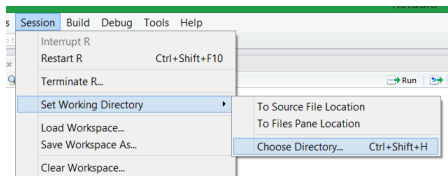
Show files in working directory and generated plots, help, open window.

Working Directory

The **working directory** is the “home base” of your R program. All files are written to and read from the working directory. There are two ways to do this:

1. Using the **interface** by doing

Session → Set Working Directory → Choose Directory



2. Doing `setwd()` as in `"setwd(/Users/ivrbik)"`. Note the working directory is retrieved with `getwd()`.

The print function will **print** to the console

```
_____ print _____  
1 > print("Hello world!")  
2 [1] "Hello world!"
```

Print statements are often used in conjunction with paste() functions

```
_____ print and paste _____  
1 > x <- 5  
2 > print(paste("x =", x))  
3 [1] "x = 5"  
4 > print(paste("x", x, sep=" = "))  
5 [1] "x = 5"  
6 > print("x=",x) # doesn't work like python  
7 [1] "x="
```

Some notes

- ▶ The `>` character is used to denote R *input*. *Output* is given directly below (the number in the square brackets denotes corresponding element number in the object appearing beside it).

```
> print(x) # where x holds the numbers 1 to 28
```

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

```
[14] 14 15 16 17 18
```

- ▶ Like Python, R is case sensitive.
- ▶ Unlike Python, R is not that particular about white space and indent.

```
>      x <- 5 # both OK!
```

```
> x <- 7
```

Basics of R

Lines can be terminated by a semicolon (but they don't need to be)

```
> x <- 5; # both OK!
```

```
> x <- 5
```

R commands may be separated either by a semi-colon or a newline.

```
> x <- 7
```

```
> y <- 3
```

```
> #or
```

```
> x <- 7; y<- 3
```

Curly brackets { } are used to group commands together.

```
for (i in c(1,2,3)) {  
  print(i)  
  print(i + 1)  
print(i - 1) # indent doesn't matter  
}
```


Basics of R

Comment lines begin with `#`. There is no multiline commenting in R.

```
> #This is a comment.
```

Variables are assigned with `<-`. N.B. We can either use the `=` or `<-` for the assignment operator. There is some debate on the preferred syntax

```
> x <- 4
```

```
> y <- 10
```

To get **help** do

```
> help(c)
```

```
> ?c
```

Calculations in R

R has the standard math operations $+$, $-$, $*$, $/$, and $^$. Trigonometric functions \sin , \cos , and \tan . Exponential \exp , \log (base ten), \log_{10} .

```
> 1+2
```

```
[1] 3
```

```
> 2*3
```

```
[1] 6
```

```
> 6/2
```

```
[1] 3
```

```
> 2^3
```

```
8
```

```
> sin(pi)
```

```
[1] 1.224647e-16
```

```
> log10(100)
```

```
[1] 2
```

Note π has a value but is **not a protected name**. Further note π 's value is **approximate**.

Basic Operations for Vectors

We can also perform basic operations on vectors:

Operation	Command
natural log	<code>log(x)</code>
Exponent	<code>exp(x)</code>
Log base 10	<code>log10(x)</code>
Absolute value	<code>abs(x)</code>
Square root	<code>sqrt(x)</code>
Sum	<code>sum(x)</code>
Number of elements in x	<code>length(x)</code>

Unique elements of x	<code>unique(x)</code>
Mean	<code>mean(x)</code>
Variance	<code>var(x)</code>
Standard Deviation	<code>sd(x)</code>
Minimum value	<code>min(x)</code>
Maximum value	<code>max(x)</code>
Smallest and largest values	<code>range(x)</code>
median	<code>median(x)</code>
Basic statistical summary	<code>summary(x)</code>
Sort	<code>sort(x)</code>

Question

How many of the following are true?

1. R is case-sensitive.
2. A command in R can be terminated by a semi-colon.
3. Indentation is the syntax used to group statements together.
4. A single line comment starts with #.

A) 0 B) 1 C) 2 D) 3 E) 4

Question

How many of the following are true?

1. R is case-sensitive.
2. A command in R can be terminated by a semi-colon.
3. Indentation is the syntax used to group statements together.
4. A single line comment starts with #.

A) 0 B) 1 C) 2 D) 3 E) 4

Question

How many of the following are true?

1. R is case-sensitive. ✓
2. A command in R can be terminated by a semi-colon.
3. Indentation is the syntax used to group statements together.
4. A single line comment starts with #.

A) 0 B) 1 C) 2 D) 3 E) 4

Question

How many of the following are true?

1. R is case-sensitive. ✓
2. A command in R can be terminated by a semi-colon. ✓
3. Indentation is the syntax used to group statements together.
4. A single line comment starts with #.

A) 0 B) 1 C) 2 D) 3 E) 4

Question

How many of the following are true?

1. R is case-sensitive. ✓
2. A command in R can be terminated by a semi-colon. ✓
3. Indentation is the syntax used to group statements together. ✗
4. A single line comment starts with #.

A) 0 B) 1 C) 2 D) 3 E) 4

Question

How many of the following are true?

1. R is case-sensitive. ✓
2. A command in R can be terminated by a semi-colon. ✓
3. Indentation is the syntax used to group statements together. ✗
4. A single line comment starts with #. ✓

A) 0 B) 1 C) 2 D) 3 E) 4

Question

How many of the following are true?

1. R is case-sensitive. ✓
2. A command in R can be terminated by a semi-colon. ✓
3. Indentation is the syntax used to group statements together. ✗
4. A single line comment starts with #. ✓

A) 0 B) 1 C) 2 D) 3 E) 4

R Data Types

Numeric Decimal values

Integer Can be created using `as.integer()`

Complex Complex values (i.e. $a+bi$)

Logical TRUE/FALSE. Can be denoted using T/F (Boolean variables)

Character String values denoted with single or double quotes.

R Data Types

Typing checking in R

```
1 > class(1)
2 [1] "numeric"
3 > class(as.integer(1))
4 [1] "integer"
5 > z = 1 + 2i; class(z)
6 [1] "complex"
7 > class(TRUE) # notice all in caps!
8 [1] "logical"
9 > class("irene")
10 [1] "character"
```

We could also use `typeof()`, see the R Language Definition manual, and more about their differences [here](#)

Try it: in R

Question

In an R program

1. Make a comment with your name and student number.
2. Calculate the following $4 \times 5 - 12^3$, $e^{3 \times 4}$, $\sin(4\pi - 6)$.
3. Guess then check with `class` the types of the following variables

```
> var1 <- FALSE  
> var2 <- T  
> var2b <- "TRUE"  
> var3 <- 3^4 - 10  
> var4 <- "Hello World"
```

Data structures

- ▶ R has a wide variety of data structures including:
 - ▶ scalars
 - ▶ vectors (numerical, character, logical)
 - ▶ matrices
 - ▶ data frames, and
 - ▶ lists.

Vectors

- ▶ A vector is the most basic data structure in R and can be of two types:
 - ▶ atomic vectors (containing all the same type of data)
 - ▶ lists (can contain a mixture of different types of data)
- ▶ As we did in the examples above, you can create a vector using `c` (see `?c` for more details on this *combine* function)

Vectors

Atomic Vectors can be vectors characters, logical, integers or numeric.

```
# example of a numeric vector:
```

```
y=c(2,3,0,3,1,0,0,1)
```

```
# example of a character vector:
```

```
letters<-c('A','B','C')
```

```
# example of a logical vector:
```

```
lvec <- c(TRUE, FALSE, FALSE)
```

Vectors

Numeric vectors can be created a number of different ways.

```
# Ceate a vector of size 10, where each element is a 2:
```

```
y=rep(2,10)
```

```
# This specifies a sequence of integers:
```

```
y=3:12
```

```
#This specifies a sequence of real numbers:
```

```
z=seq(from=1,to=2,by=0.1)
```

```
#This specifies a sequence of real numbers:
```

```
z=seq(from=2,to=1,by=-0.1)
```

The end number (specified by the to argument) *is* inclusive!

Indexing Vectors

To index an element from a vector, use single square brackets

```
> z = c("apples","bananas","oranges", "pineapples")
> z[1]
[1] "apples"
> z[2:3]
[1] "bananas" "oranges"
> z[c(4,2)] # for selecting non-contiguous elements
[1] "pineapples" "bananas"
> z[-1] # removes the first item
[1] "bananas"      "oranges"      "pineapples"
```

Notice that the index begins at 1

Also notice the difference in the [-1] notation.

We have the option of including names for our vector elements (similar to a Python dictionary)

```
z = c(123456, 25, 2019)
names(z) <- c("studentID", "age", "year")
z["studentID"] # same as z[1]
z[c("age", "year")] # wont work like z[c(2,3)]
z[c(FALSE, TRUE, TRUE)] # will work
z["-year"] # wont work like z[-3]
```

To add a new element/delete/replace an element:

```
> z = c("first","second","third", "fourth")
> # adds a fith element and replaces the second
> z[5] = "fifth" # doesn't need to exist before we assign
> z[2] = "2nd"
> z
[1] "first"  "2nd"    "third"  "fourth" "fifth"
> #removes the second element
> (z = z[-2])
[1] "first"  "third"  "fourth" "fifth"
```

Notice what happens if we add a 10th element to `z` (without specifying elements 6–9):

```
> z[10] = "tenth"
> z
[1] "first"  "third"  "fourth" "fifth"  NA
[6] NA       NA       NA       NA       "tenth"
```

We can also add new named elements

```
> z = c(123456, 25, 2019)
> names(z) <- c("studentID", "age", "year")
> # adds a forth element with the students major
> z["major"] = "COSC"
> z
studentID      age      year      major
"123456"      "25"     "2019"     "COSC"
```

Vector operations

Notice that operations performed on vectors get computed on each element of the vector (similar to `map` in Python)

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

```
> y <- 11:20
```

```
> x*2
```

```
[1]  2  4  6  8 10 12 14 16 18 20
```

```
> x+y
```

```
[1] 12 14 16 18 20 22 24 26 28 30
```

Matrices

- ▶ While we can view matrices as column vectors stacked side by side or row vectors stack one on top of another, matrices are really just a vector in \mathbb{R}^n .
- ▶ The main difference between matrices and the vectors discussed in the previous slides is that a matrix has a **dimension** (like numpy arrays in Python)
- ▶ The dimensions can be found using the `dim()` function.

Matrices

```
> (m <- matrix(1:8, nrow = 2, ncol = 4))  
      [,1] [,2] [,3] [,4]  
[1,]    1    3    5    7  
[2,]    2    4    6    8  
  
> dim(m) # gives the number of rows and columns  
[1] 2 4  
  
> class(m)  
[1] "matrix"
```

Matrices

- ▶ By default, matrices are constructed columnwise.
- ▶ You can always change to row-wise by specifying `byrow=TRUE`

```
(m <- matrix(1:6, nrow=2, ncol =3)) # fill columnwise
```

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

```
(m <- matrix(1:6, nrow=2, ncol =3, byrow=TRUE)) #fill row-wise
```

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

Named Matrices

We may also choose to name our rows and columns

```
rownames(m) = c("row1", "row2")  
colnames(m) = c("col1", "col2", "col3")
```

```
m
```

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

Indexing

- ▶ To index an element from a matrix, we can again use square brackets, but we need to specify TWO numbers `[row, col]`.
- ▶ If we want to extract an entire row, we'll leave the `col` field blank (indicating we want all columns).
- ▶ If we want to extract an entire column, we'll leave the `row` field blank (indicating we want all rows).

Indexing

The following extracts a column, row, and cell from matrix m ,

```
# extract the third column of m
```

```
m[,3]
```

```
## row1 row2
```

```
##      3      6
```

```
# extract the first row of m
```

```
m[1,]
```

```
## col1 col2 col3
```

```
##      1      2      3
```

```
# extract the element in the 1st row and 3rd column
```

```
m[1,3]
```

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

Alternatively we could call them by name

```
# extract the third column of m
m[, "col3"] # same as m[,3]

## row1 row2
##      3      6

# extract the first row of m
m["row1",] # same as m[1,]

## col1 col2 col3
##      1      2      3

# extract the element in the 1st row and 3rd column
m["row1", "col3"] # same as m[1,3]

## [1] 3
```

Indexing

Note that we can also use one number (tricker this way)

```
(m = matrix(11:4, 2,4))

##      [,1] [,2] [,3] [,4]
## [1,]   11    9    7    5
## [2,]   10    8    6    4

m[8] # extracts the 8th element in m

## [1] 4

(mVector = as.numeric(m))

## [1] 11 10  9  8  7  6  5  4

mVector[8]

## [1] 4
```

Indexing

Like the vectors discussed earlier, matrices fall under the atomic vector category and therefore can only store data of one type:

```
m = matrix(c("bananas", 2, TRUE, 0), nrow=2, ncol=2)
```

```
# notice how all coerce to characters:
```

```
m
```

```
##      [,1]      [,2]  
## [1,] "bananas" "TRUE"  
## [2,] "2"       "0"
```


lists

- ▶ If we want our vector to contain objects of different data types we need to create a list.
- ▶ We can think of lists as a special type of vector that can mix data types and structures

```
(x = list("apple", "banana", 2, TRUE, 4, "four"))
```

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

```
## [1] "apple"
```

```
##
```

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

```
## [1] "banana"
```

```
##
```

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

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

```
##
```

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

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

```
##
```

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

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

```
##
```

```
## [[6]]
```

```
## [1] "four"
```

lists

Unlike vectors, the member of a list are accessed using double square brackets:

```
x[[1]] # first member
```

```
## [1] "apple"
```

```
x[[2]] # second member, and so on ...
```

```
## [1] "banana"
```

lists

- ▶ While I have specified each member to be a single word/number/logical, we could also have specified them to be vectors or matrices.
- ▶ To index the elements within the members of a list we use single square brackets (usually nested after double square brackets)

lists

```
y = list(c("apples", "banana", "four"), c(2,4), TRUE)
```

```
y
```

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

```
## [1] "apples" "banana" "four"
```

```
##
```

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

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

```
##
```

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

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

lists

```
y[[1]][3] # the third element of the first member of y
```

```
## [1] "four"
```

```
y[[2]][1] # the first element of the second member of y
```

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

```
y[[1]][4] # calling something that does not exist
```

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

Named lists

Alternatively we could have names for our list members

```
numbers= c(2,4) # can define the member outside first
y = list(words=c("apples", "banana", "four"),
          numbers, logicals=TRUE)
y$words # same this as y[[1]]

## [1] "apples" "banana" "four"

y$numbers # same this as y[[2]]

## NULL

y$logicals # same this as y[[3]]

## [1] TRUE
```

We could also name the list members after they have been assigned

```
y = list(c("apples", "banana", "four"),  
        c(2,4), TRUE)  
names(y) <- c("words", "numbers", "logicals")
```

```
y
```

```
## $words
```

```
## [1] "apples" "banana" "four"
```

```
##
```

```
## $numbers
```

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

```
##
```

```
## $logicals
```

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


lists

We can index members of lists using the integer index numbers OR their member names (in quotations)

```
y$words
```

```
## [1] "apples" "banana" "four"
```

```
y[["words"]]
```

```
## [1] "apples" "banana" "four"
```

```
y[[4]]
```

```
## Error in y[[4]]: subscript out of bounds
```

```
# the following are not valid:
```

```
# y$4
```

```
# y[[words]]
```

lists

We can modify its content/add new content directly either using `[[]]` with numbers or `$` with member names.

```
y$words[3] = "oranges" #replace the 3rd element of the 1st member  
y[[1]][4] = "pineapples" #add a 4th element to the 1st member  
y[[4]] = c(0,1,1,0) # add a forth member to y  
y$"fifth" = 3:12 # add a fifth member to y (named "fifth")
```

Data Frames

- ▶ Data frames are perhaps the most used data structure used in statistics.
- ▶ Like list, they can store different data types.
- ▶ Furthermore, they can contain additional attributes such as column and row names.
- ▶ When combining vectors in a data frame all must have the same size (i.e. length).
- ▶ Missing observations will be recorded as NA.

Data Frames

Here is an example of how to create a data frame and add to it:

```
Person=c('John', 'Jill', 'Jack')  
Grade=c('45', '92', '91')  
(Lab=data.frame(Person, Grade))
```

```
##   Person Grade  
## 1   John    45  
## 2   Jill    92  
## 3   Jack    91
```

Data Frames

Like matrices, we can create column and row names

```
colnames(Lab) # adopted from the variable names inputed
```

```
## [1] "Person" "Grade"
```

```
rownames(Lab) # default to increasing integers
```

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

```
# we could also add row names if we choose
```

```
rownames(Lab) = c("Student1", "Student2", "Student3")
```

```
Lab
```

```
##           Person Grade
```

```
## Student1   John    45
```

```
## Student2   Jill    92
```

```
## Student3   Jack    91
```

Data Frames

Unlike matrices, we can extract a column of this data.frame we use the \$ notation:

```
Lab$Person
```

```
## [1] John Jill Jack
```

```
## Levels: Jack Jill John
```

```
Lab$Grade
```

```
## [1] 45 92 91
```

```
## Levels: 45 91 92
```

```
# N.B. This does not work for rows:
```

```
Lab$Student1
```

```
## NULL
```

Aside

Smart Autofill

- ▶ In some situations, R will try to autocomplete commands for us. When you begin to type `Lab$` for instance, you will see the options for the column names appear.
- ▶ You can use the arrow keys on your keyboard to navigate through the options.
- ▶ Press ENTER when you land on the option you want.

Data Frames

Adding a column to the data frame is as easy as typing:

```
Lab$Passed=c(FALSE,TRUE,TRUE)
```

```
Lab
```

```
##           Person Grade Passed
## Student1   John    45  FALSE
## Student2   Jill    92   TRUE
## Student3   Jack    91   TRUE
```


Attributes

- ▶ Each of these objects have so-called **attributes**.
- ▶ Here are some examples of attributes:
 1. names, dimension names
 2. dimensions
 3. class
 4. length

Attributes

```
attributes(Lab) # data vector defined in previous example
```

```
## $names
```

```
## [1] "Person" "Grade"  "Passed"
```

```
##
```

```
## $row.names
```

```
## [1] "Student1" "Student2" "Student3"
```

```
##
```

```
## $class
```

```
## [1] "data.frame"
```

```
attributes(m) # matrix defined previously
```

```
## $dim
```

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

Data Entry

- ▶ Data entry can be either done by hand (impractical), or loaded from an existing file.
- ▶ `read.table` is a convenient way of reading data into R and storing it to a data frame.
- ▶ The data should be formatted such that each row contains information regarding one subject with data separated by white space/commas/tabs/.
- ▶ The first line may (or may not) contain a header with the names of the variables in each column.

Data Entry

Let's take the data `intake.txt` data available on CANVAS which looks like this:

"pre"	"post"
-------	--------

5260	3910
------	------

5470	4220
------	------

5640	3885
------	------

6180	5160
------	------

6390	5645
------	------

6515	4680
------	------

6805	5265
------	------

7515	5975
------	------

7515	6790
------	------

Data Entry

To read the data in:

```
read.table("intake.txt", header=TRUE)
```

```
##      pre post
## 1  5260 3910
## 2  5470 4220
## 3  5640 3885
## 4  6180 5160
## 5  6390 5645
## 6  6515 4680
## 7  6805 5265
## 8  7515 5975
## 9  7515 6790
## 10 8230 6900
## 11 8770 7335
```

Data Entry

- ▶ `header=TRUE` indicates that the first line of file contains the names of the variables (rather than data values).
 - ▶ `header=TRUE` allows `pre` and `post` to be read in as column names.
- ▶ (`header=FALSE` by default)

Data Entry

Since we didn't save it to a variable, it simply prints it to the screen. This won't be very useful in practice, so save it to a variable with a name that makes sense.

```
intake = read.table("intake.txt", header=TRUE)
```

Aside

Name Rules

There are rules on the types of names we can give variables:

- ▶ No spaces
- ▶ No special characters (apart from periods and underscores)
- ▶ Can't begin with a number

Data Entry

If we instead want to import the `data.csv` file available on CANVAS, we simply specify that our separator is a comma.

```
data = read.table("data.csv", sep=",", header=TRUE)
```

Alternatively, we could have used the `read.csv()` function

```
data = read.csv("data.csv") # header=TRUE by default in read.csv
```

Data Entry

If you want to specify a file that is outside of your working directory, simply specify the path name preceeding the filename. For example:

```
data <- read.csv("/Users/ivrbik/DATA101/data.csv", header=T)
```

Data Exporting

There is also a function that will allow you to write the contents of a data frame to a file. For instance, we could save our simple Lab data frame to a textfile called Lab.txt or Lab.csv using the following:

```
write.table(Lab, file="Lab.txt")  
write.csv(Lab, file="Lab.csv")
```

Like the read functions, write will, by default, save these files to your working direction. To specify otherwise use:

```
write.table(Lab, file="<path>/Lab.txt")
```

Tip of the Day

To obtain a previous calculation in R , navigate to the console, and press the 'Up/Down Arrow' key on your keyboard.