Lecture 1: Introduction to R

STAT UN2102 *Applied Statistical Computing*

Gabriel Young
Columbia University

January 16, 2018

Some notes on the course

Notes

- You should have taken an introductory statistics course.
- Do not take UN2102 and GU4206. There is a **huge** overlap in those two courses.

Lecture

- You must have `R` and `R`Studio downloaded by next lecture.
- Bring your laptop to class every week to follow along.

Honor Code

"Columbia's intellectual community relies on academic integrity and responsibility as the cornerstone of its work. Graduate students are expected to exhibit the highest level of personal and academic honesty as they engage in scholarly discourse and research. **In practical terms, you must be responsible for the full and accurate attribution of the ideas of others in all of your research papers and projects; you must be honest when taking your examinations; you must always submit your own work and not that of another student, scholar, or internet source.** Graduate students are responsible for knowing and correctly utilizing referencing and bibliographical guidelines."

- Resources at `http://gsas.columbia.edu/academic-integrity`.
- Failure to observe these rules of conduct will have serious academic consequences, up to and including dismissal from the university.

A Word of Thanks

This course was developed for Columbia students with much guidance from the following course. Its web page is also a good resource for students.

Cosma Shalizi and Andrew Thomas (2014), "Statistical Computing 36-350: Beginning to Advanced Techniques in R".

A special thanks for guidance and advice from the following individuals and many others. John Cunningham, Yang Feng, Tian Zheng, Jennifer Hoeting, and Cynthia Rush.

Statistical Computing

It's essential for modern statisticians to be fluent in *statistical computing* (statistical programming).

At the end of this course, you will have:

- The ability to read and write code for statistical data analysis.
- An understanding of programming topics such as functions, objects, data structures, debugging.

What's the difference between data science and statistics?

"A data scientist is just a sexier word for statistician."

Nate Silver

"A data scientist is a better computer scientist than a statistician and is a better statistician than a computer scientist."

Unknown

Working with Data in R [1]

Steps

1. *Import* data from various sources: the web, a database, a stored file.
2. *Clean* and format the data. Usually this means rows are observations and columns are variables.
3. *Analyze* the data using visualizations, modelling, or other methods.
4. *Communicate* your results.

[1] Slide developed from G. Grolemund and H. Wickham.

Working with Data in R [1]

Steps

1. *Import* data from various sources: the web, a database, a stored file.
2. *Clean* and format the data. Usually this means rows are observations and columns are variables.
3. *Analyze* the data using visualizations, modelling, or other methods.
4. *Communicate* your results.

In this class, we learn tools and strategies for completing each step.

[1] Slide developed from G. Grolemund and H. Wickham.

Functional Programming [2]

Function Definition

A function is a machine which turns input objects (*arguments*) into an output object (*return value*), according to a definite rule.

- Programming is writing functions to transform inputs into outputs easily and correctly.
- Good programming takes big transformations and breaks them down into smaller and smaller ones until you come to tasks which the built-in functions can do.

[2] Slide developed from C.R. Shalizi and A.C. Thomas (2014).

What is R?

What is R?

R is an open-source statistical programming software used by industry professionals and academics alike.

This means that R is supported by a community of users.

Will use R extensively in this class

- Download R at: `https://www.r-project.org`
- Download RStudio at: `https://www.rstudio.com`

What is R?

R is an open-source statistical programming software used by industry professionals and academics alike.

This means that R is supported by a community of users.

Will use R extensively in this class

- Download R at: `https://www.r-project.org`
- Download RStudio at: `https://www.rstudio.com`

You must have R downloaded by next lecture!

Using R and RStudio

- The *editor* allows you to type and save code that you may want to reuse later.
- Basic interaction with R happens in the *console*. The is where you type R code.

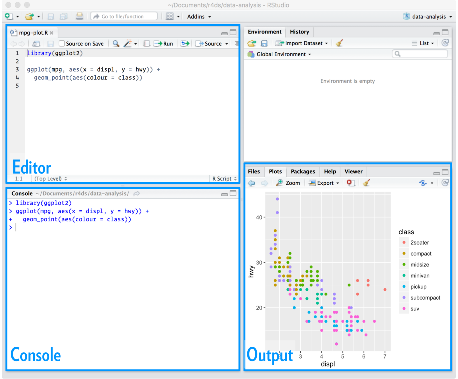


Figure 1: Image of RStudio from G. Grolemund and H. Wickham

If you'd like more information on the other functions and features of RStudio check out the short video tutorial on the Canvas page.

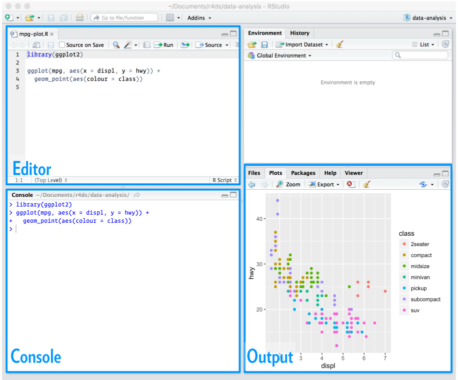


Figure 2: Image of RStudio from G. Grolemund and H. Wickham

Using R and RStudio

Code example.

R Markdown

For your homeworks and lab reports you will be using `R Markdown` which allows you to put your code, its outputs, and your thoughts all in one document.

Steps to Create an R Markdown Document

To create a new R Markdown document open RStudio and go to File -> New File -> R Markdown.

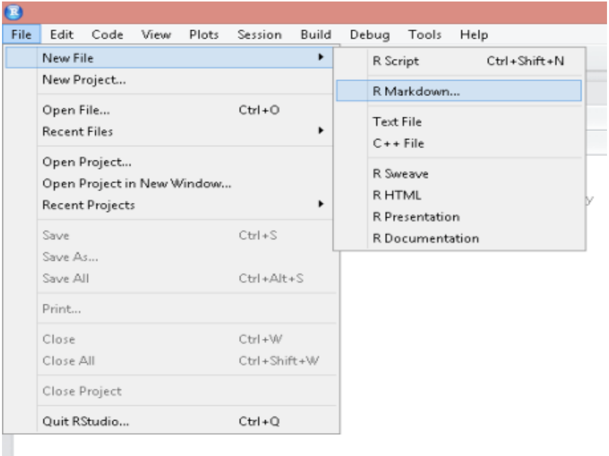


Figure 3: Image from https://www.r-bloggers.com

Steps to Create an R Markdown Document

Enter the title of your document and the author and hit OK.

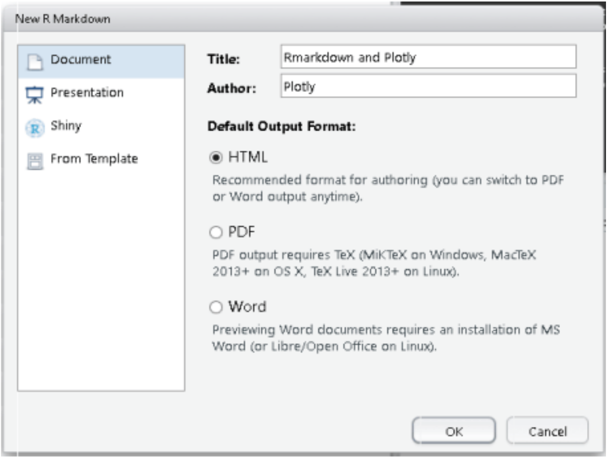


Figure 4: Image from https://www.r-bloggers.com

Steps to Create an R Markdown Document

Clicking the `Knit HTML` button in the editor window generates the document.

Editing the Markdown Document

- Writing equations uses LaTex code. So, for example, `$x^2$` produces x^2.
- Insert R code directly into the document using the following format:

````
```{r}
x <- rnorm(100)
```
````

- If you need help, go to `Help -> Markdown Quick Reference`.
- You'll get practice with this in lab.

Steps to Create an `R Markdown` Document

Code example.

A Quick Example...

Type the following into your console:

```
> # Create a vector in R names "x"
> x <- c(5, 29, 13, 87)
> x
```

```
[1]  5 29 13 87
```

Two important ideas:

1. Commenting
2. Assignment

A Quick Example...

Type the following into your console:

```
> # Create a vector in R
> x <- c(5, 29, 13, 87)
> x
```

```
[1]  5 29 13 87
```

Two important ideas:

1. Commenting
   - Anything after the # isn't evaluated by R.
   - Used to leave notes for humans reading your code.
   - Very important in our class. Comment your code!
2. Assignment

A Quick Example...

Type the following into your console:

```
> # Create a vector in R
> x <- c(5, 29, 13, 87)
> x
```

```
[1]  5 29 13 87
```

Two important ideas:

1. Commenting
2. Assignment
    - The `<-` symbol means assign `x` the value `c(5, 29, 13, 87)`.
    - Could use = instead of `<-` but this is discouraged.
    - All assignments take the same form: `object_name <- value`.
    - `c()` means "concatenate".
    - Type `x` into the console to print its assignment.

A Quick Example...

Type the following into your console:

```
> # Create a vector in R names "x"
> x <- c(5, 29, 13, 87)
> x
```

```
[1]  5 29 13 87
```

Note

The [1] tells us that 5 is the first element of the vector.

```
> # Create a vector in R names "x"
> x <- 1:50
> x
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
[19] 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
[37] 37 38 39 40 41 42 43 44 45 46 47 48 49 50
```

Variable Types, Vectors, & Matrices

Variable Types

`R` has a variety of variable types (or *modes*).

Modes

1. Numeric (3.7, 15907, 80.333)
2. Complex (1 + 2i)
3. Character ("Columbia", "Statistics is fun!", "HELLO WORLD")
4. Logical (TRUE, FALSE, 1, 0)

In this class, we are primarily concerned with numeric, character, and logical.

Let's check this out in R

'Numeric' variable type

```
> x <- 2
> mode(x)
```

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

```
> typeof(x)
```

```
[1] "double"
```

```
> y <- as.integer(3)
> typeof(y)
```

```
[1] "integer"
```

Let's check this out in R

'Complex' variable type

```
> z <- 1 - 2i
> z
```

```
[1] 1-2i
```

```
> typeof(z)
```

```
[1] "complex"
```

Let's check this out in R

'Character' variable type

```
> name <- "Columbia University"
> name
```

```
[1] "Columbia University"
```

```
> typeof(name)
```

```
[1] "character"
```

Let's check this out in R

'Logical' variable type

```
> a <- TRUE
> b <- F
> a
```

```
[1] TRUE
```

```
> b
```

```
[1] FALSE
```

```
> typeof(a)
```

```
[1] "logical"
```

Data Types

There are many data types in R.

Data Types

- Vectors
- Scalars
- Matrices
- Arrays
- Lists
- Dataframes

Data Types

There are many data types in R.

Data Types

Vectors

- All elements must be the same type (mode).
- More to come in this lecture!

- Scalars
- Matrices
- Arrays
- Lists
- Dataframes

Data Types

There are many data types in R.

Data Types

- Vectors

Scalars

- Treated as one-element vectors in R.

- Matrices
- Arrays
- Lists
- Dataframes

Data Types

There are many data types in R.

Data Types

- Vectors
- Scalars

Matrices

- An array (rows and columns) of values.
- All values must be the same type (mode).
- More to come this lecture!

- Arrays
- Lists
- Dataframes

Data Types

There are many data types in R.

Data Types

- Vectors
- Scalars
- Matrices

Arrays

- Similar to matrices, but with more than two dimensions.

- Lists
- Dataframes

Data Types

There are many data types in R.

Data Types

- Vectors
- Scalars
- Matrices
- Arrays

Lists

- Like a vector, but elements can be of different modes.

- Dataframes

Data Types

There are many data types in `R`.

Data Types

- Vectors
- Scalars
- Matrices
- Arrays
- Lists

Dataframes

- Like a matrix, but elements can be of different modes.
- More to come next week!

Check Your Understanding

What mode are the following variables?

1. `3*TRUE`?
2. `"147"`?

Check Your Understanding

What mode are the following variables?

1. 3*TRUE?
2. "147"?

Solutions

```
> 3*TRUE # Logicals in arithmetic
```

```
[1] 3
```

```
> mode(3*TRUE)
```

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

```
> mode("147")
```

```
[1] "character"
```

Vectors and Matrices in R

Recall: Vectors

- Variable types are called modes.
- All elements of a vector are the same mode.
- Scalars are just single-element vectors.

Recall: Matrices

- All elements of a matrix are the same mode.
- A matrix is treated like a vector in R with two additional attributes: number of rows and number of columns.

Building Vectors in R

- Use the *concatenate* function c() to define a vector.

Some Examples

Defining a numeric vector:

```
> x <- c(2, pi, 1/2, 3^2)
> x
```

```
[1] 2.000000 3.141593 0.500000 9.000000
```

A character vector:

```
> y <- c("NYC", "Boston", "Philadelphia")
> y
```

```
[1] "NYC"          "Boston"       "Philadelphia"
```

Building Vectors in R

- The syntax `a:b` produces a *sequence* of integers ranging from *a* to *b*.
- The *repetition* function `rep(val, num)` repeats the value `val` a total of `num` times.

Some Examples

A sequential list of integers:

```
> z <- 5:10
> z
```

```
[1]  5  6  7  8  9 10
```

Using `rep()` to create a 1's vector:

```
> u <- rep(1, 18)
> u
```

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

Building Vectors in R

- Alternately, could allocate space and then fill in element-wise.

Some Examples

```
> v <- c()
> v[1] <- TRUE
> v[2] <- TRUE
> v[3] <- FALSE
> v
```

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

Building Vectors in R

- The *concatenate* function `c()` can be nested.

Some Examples

```
> vec1 <- rep(-27, 3)
> vec1
```

```
[1] -27 -27 -27
```

```
> vec2 <- c(vec1, c(-26, -25, -24))
> vec2
```

```
[1] -27 -27 -27 -26 -25 -24
```

Building Matrices in R

- Use the function `matrix(values, nrow, ncol)` to define your matrix.
- In R, matrices are stored in *column-major order* (determines where the number go as in the following example).

Some Examples

Building a matrix that fills in by column:

```
> mat <- matrix(1:9, nrow = 3, ncol = 3)
> mat
```

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

Building Matrices in R

- Use the function `matrix(values, nrow, ncol)` to define your matrix.
- In R, matrices are stored in *column-major order* (determines where the number go as in the following example).

Some Examples

Building a matrix that fills in by row:

```
> new_mat <- matrix(1:9, nrow = 3, ncol = 3, byrow = TRUE)
> new_mat
```

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

Building Matrices in R

- Alternately, could allocate space and fill in element-wise.
- Tell R the size of the matrix beforehand.

Some Examples

Allocating the space for a matrix then filling it in:

```
> this_mat <- matrix(nrow = 2, ncol = 2)
> this_mat[1,1] <- sqrt(27)
> this_mat[1,2] <- round(sqrt(27), 3)
> this_mat[2,1] <- exp(1)
> this_mat[2,2] <- log(1)
> this_mat
```

```
         [,1]  [,2]
[1,] 5.196152 5.196
[2,] 2.718282 0.000
```

Building Matrices in R

- The *row bind* function `rbind()` also works, though it can be costly computationally. Similarly for *column bind* function `cbind()`.

Some Examples

```
> vec1 <- rep(0, 4)
> vec2 <- c("We're", "making", "matrices", "!")
> final_mat <- rbind(vec1, vec2)
> final_mat
```

```
     [,1]    [,2]     [,3]       [,4]
vec1 "0"     "0"      "0"        "0"
vec2 "We're" "making" "matrices" "!"
```

Recall, matrix entries must all be the same type.

Building Matrices in R

- Name columns (rows) of a matrix using `colnames()` (`rownames()`).

Some Examples

```
> this_mat # Defined previously
```

```
        [,1]  [,2]
[1,] 5.196152 5.196
[2,] 2.718282 0.000
```

```
> colnames(this_mat) # Nothing there yet
```

```
NULL
```

Building Matrices in R

- Name columns (rows) of a matrix using `colnames()` (`rownames()`).

Some Examples

```
> colnames(this_mat) <- c("Column1", "Column2")
> this_mat
```

```
      Column1 Column2
[1,] 5.196152   5.196
[2,] 2.718282   0.000
```

Mixing Variable Modes

- When variable modes are mixed in vectors or matrices, `R` picks the 'least common denominator'.
- Use the *structure* function `str()` to display the internal structure of an `R` object.

Example

```
> vec <- c(1.75, TRUE, "abc")
> vec
```

```
[1] "1.75" "TRUE" "abc"
```

```
> str(vec)
```

```
 chr [1:3] "1.75" "TRUE" "abc"
```

Help in R

- Use a single question mark `?` to get help about a specific function using form `?function name`.
  - Provides a description, lists the arguments (to the function), gives an example, etc.
- Use the double question mark to get help with a topic using form `??topic`.

How to get help in R

```
> # What does the str() function do?
>
> # Function help:
> ?str
> # Fuzzy matching:
> ??"structure"
```

Help in R

Code example.

Subsetting Vectors

- Use square brackets [] to extract elements or subsets of elements.

Example

```
> y <- c(27, -34, 19, 7, 61)
> y[2]
```

```
[1] -34
```

```
> y[3:5]
```

```
[1] 19  7 61
```

```
> y[c(1, 4)]
```

```
[1] 27  7
```

Subsetting Vectors

- Use the same strategy to reassign elements of a vector.

Example

```
> y <- c(27, -34, 19, 7, 61)
> y
```

```
[1]  27 -34  19   7  61
```

```
> y[c(1, 4)] <- 0
> y
```

```
[1]   0 -34  19   0  61
```

Subsetting Vectors

- Negative values can be used to exclude elements.

Example

```
> y <- c(27, -34, 19, 7, 61)
> y
```

```
[1]  27 -34  19   7  61
```

```
> y[-c(1, 4)]
```

```
[1] -34  19  61
```

```
> y <- y[-1]
> y
```

```
[1] -34  19   7  61
```

Subsetting Matrices

- `mat[i,j]` returns the $(i,j)th$ element of `mat`.
- `mat[i, ]` returns the i^{th} row of `mat`.
- `mat[ ,j]` returns the j^{th} column of `mat`.

```
> mat <- matrix(1:8, ncol = 4)
> mat
```

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

```
> mat[, 2:3]
```

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

Subsetting Matrices

- Can use column names or row names to subset as well.
- Negative values are used to exclude elements.

```
> this_mat
```

```
     Column1 Column2
[1,] 5.196152   5.196
[2,] 2.718282   0.000
```

```
> this_mat[, "Column2"]
```

```
[1] 5.196 0.000
```

```
> this_mat[, -1]
```

```
[1] 5.196 0.000
```

An Extended Example: Image Data

But First...Packages!

What are packages?

- Packages are collections of functions, data, or code that extend the capabilities of base R.
- Some packages come pre-loaded but others must be downloaded and installed using function `install.packages("package name")`.
- An installed R package must be loaded in each session it is to be used using function `library("package name")`.

```
> # Installing the "pixmap" package.
> install.packages("pixmap")
> library("pixmap")
```

But First...Packages!

A quick video!

Image Data Example [3]

- Images are made up of pixels which are arranged in rows and columns (like a matrix).
- Image data are matrices where each element is a number representing the intensity or brightness of the corresponding pixel.
- We will work with a greyscale image with numbers ranging from 0 (black) to 1 (white).

[3] Example developed from N. Matloff, "The Art of R Programming: A Tour of Statistical Software Design".

Image Data Example (cont.)

```
> library(pixmap)
> casablanca_pic <- read.pnm("casablanca.pgm")
> casablanca_pic
```

```
Pixmap image
  Type          : pixmapGrey
  Size          : 360x460
  Resolution    : 1x1
  Bounding box  : 0 0 460 360
```

```
> plot(casablanca_pic)
```

Image Data Example (cont.)

Image Data Example (cont.)

```
> dim(casablanca_pic@grey)
```

```
[1] 360 460
```

```
> casablanca_pic@grey[360, 100]
```

```
[1] 0.4431373
```

```
> casablanca_pic@grey[180, 10]
```

```
[1] 0.9882353
```

Image Data Example (cont.)

Let's erase Rick from the image.

```
> casablanca_pic@grey[15:70, 220:265] <- 1
```

```
> plot(casablanca_pic)
```

Image Data Example (cont.)

Image Data Example (cont.)

- Use `R`'s `locator()` function to find the rows and columns corresponding to Rick's face.
- A call to the function allows the user to click on a point in a plot and then the function returns the coordinates of the click.

Check Your Understanding

Using matrix z, what is the output of the following?

```
> z
```

```
     First Second Third
[1,]     1      4     7
[2,]     2      5     8
[3,]     3      6     9
```

1. `z[2:3, "Third"]?`
2. `c(z[,-(2:3)], "abc")?`
3. `rbind(z[1,], 1:3)?`

Check Your Understanding

```
> z
```

```
     First Second Third
[1,]     1      4     7
[2,]     2      5     8
[3,]     3      6     9
```

Solutions

```
> z[2:3, "Third"]
```

```
[1] 8 9
```

```
> c(z[,-(2:3)], "abc")
```

```
[1] "1"   "2"   "3"   "abc"
```

Check Your Understanding

```
> z
```

```
     First Second Third
[1,]     1      4     7
[2,]     2      5     8
[3,]     3      6     9
```

Solutions

```
> rbind(z[1,], 1:3)
```

```
     First Second Third
[1,]     1      4     7
[2,]     1      2     3
```

More with Vectors & Matrices and Linear Algebra Review

Reminder: Vector Algebra

Define vectors:

$$A = (a_1, a_2, \ldots, a_N), \quad \text{and} \quad B = (b_1, b_2, \ldots, b_N).$$

Then for c a scalar,

- $A + B = (a_1 + b_1, a_2 + b_2, \ldots, a_N + b_n)$.
- $cA = (ca_1, ca_2, \ldots, ca_N)$.
- Dot product: $A \cdot B = a_1 b_1 + a_2 b_2 + \ldots + a_N b_N$.
- Norm: $\|A\|^2 = A \cdot A = {a_1}^2 + {a_2}^2 + \ldots + {a_N}^2$.

Reminder: Matrix Algebra

Define matrices:

$$A = \begin{pmatrix} a_1 & a_3 \\ a_2 & a_4 \end{pmatrix}, \quad \text{and} \quad B = \begin{pmatrix} b_1 & b_3 \\ b_2 & b_4 \end{pmatrix}.$$

Then for c a scalar,

- $A + B = \begin{pmatrix} a_1 + b_1 & a_3 + b_3 \\ a_2 + b_2 & a_4 + b_4 \end{pmatrix}.$
- $cA = \begin{pmatrix} ca_1 & ca_3 \\ ca_2 & ca_4 \end{pmatrix}.$
- Matrix Multiplication: $AB = \begin{pmatrix} a_1 b_1 + a_3 b_2 & a_1 b_3 + a_3 b_4 \\ a_2 b_1 + a_4 b_2 & a_2 b_3 + a_4 b_4 \end{pmatrix}.$
  What if the dimensions of A and B are different?

Reminder: Matrix Operations

Define matrices:

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{pmatrix}, \quad \text{and } B = \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix}.$$

- The *transpose* of A is a $m \times n$ matrix:

$$t(A) = \begin{pmatrix} a_{1,1} & a_{2,2} & \cdots & a_{n,1} \\ a_{1,2} & a_{2,2} & \cdots & a_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1,m} & a_{2,m} & \cdots & a_{n,m} \end{pmatrix}.$$

- The *trace* of the square matrix B is the sum of the diagonal elements: $tr(B) = b_{1,1} + b_{2,2}$.

Reminder: Matrix Operations

Define matrices:

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{pmatrix}, \quad \text{and } B = \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix}.$$

- The *determinant* of square matrix B is $det(B) = b_{1,1}b_{2,2} - b_{1,2}b_{2,1}$.
  How do you find the determinant for an $n \times n$ matrix?
- The *inverse* of square matrix B is denoted B^{-1} and
  $BB^{-1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ and

$$B^{-1} = \frac{1}{det(B)} \begin{pmatrix} b_{2,2} & -b_{1,2} \\ -b_{2,1} & b_{1,1} \end{pmatrix}.$$

How do you find the inverse of an $n \times n$ matrix?

Functions on Numeric Vectors

Useful R Functions

R function	Description
`length(x)`	Length of a vector x
`sum(x)`	Sum of a vector x
`mean(x)`	Arithmetic mean of a vector x
`quantiles(x)`	Sample quantiles of a vector x
`max(x)`	Maximum of a vector x
`min(x)`	Minimum of a vector x
`sd(x)`	Sample standard deviation of a vector x
`var(x)`	Sample variance of a vector x
`summary(x)`	Summary statistics of vector x

Reminder...

To access the help documentation of a known R function, use syntax `?function`.

Example: Functions on Numeric Vectors

Example

To investigate the dependence of energy expenditure (y) on body build, researches used underwater weighing techniques to determine the fat-free body mass (x) for each of seven men. They also measured the total 24-hour energy expenditure for each man during conditions of quiet sedentary activity. The results are shown in the table.

Subject	1	2	3	4	5	6	7
x	49.3	59.3	68.3	48.1	57.61	78.1	76.1
y	1,894	2,050	2,353	1,838	1,948	2,528	2,568

```
> # Define covariate and response variable
> x <- c(49.3,59.3,68.3,48.1,57.61,78.1,76.1)
> y <- c(1894,2050,2353,1838,1948,2528,2568)
```

Example: Functions on Numeric Vectors (continued)

Example

Subject	1	2	3	4	5	6	7
x	49.3	59.3	68.3	48.1	57.61	78.1	76.1
y	1,894	2,050	2,353	1,838	1,948	2,528	2,568

```
> n <- length(x) # Sample size
> n
```

```
[1] 7
```

```
> max(x)
```

```
[1] 78.1
```

```
> sd(x)
```

```
[1] 12.09438
```

Example: Functions on Numeric Vectors (continued)

Example

Subject	1	2	3	4	5	6	7
x	49.3	59.3	68.3	48.1	57.61	78.1	76.1
y	1,894	2,050	2,353	1,838	1,948	2,528	2,568

```
> summary(x) # Summary statistics
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  48.10   53.46   59.30   62.40   72.20   78.10
```

```
> summary(y)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1838    1921    2050    2168    2440    2568
```

Element-Wise Operations for Vectors

Vectors x and y must have the same length. Let a be a scalar.

Element-Wise Operators

Operator	Description
`a + x`	Element-wise scalar addition
`a * x`	Element-wise scalar multiplication
`x + y`	Element-wise addition
`x * y`	Element-wise multiplication
`x ^ a`	Element-wise power
`a ^ x`	Element-wise exponentiation
`x ^ y`	Element-wise exponentiation

Recycling

Recall that a scalar is just a vector of length 1. When a shorter vector is added to a longer one, the elements in the shorter vectored are repeated. This is *recycling*.

Some Examples

```
> u <- c(1,3,5)
> v <- c(1,3,5)
> v + 4 # Recycling
```

```
[1] 5 7 9
```

```
> v + c(1,3) # Recycling
```

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

```
> v + u
```

```
[1]  2  6 10
```

Some Examples

Note: Operators are functions in R.

```
> u <- c(1,3,5)
> v <- c(1,3,5)
> '+'(v,u)
```

```
[1]  2  6 10
```

```
> '*'(v,u)
```

```
[1]  1  9 25
```

Line of Best Fit Example

Recall the energy expenditure versus fat-free body mass example.

Example

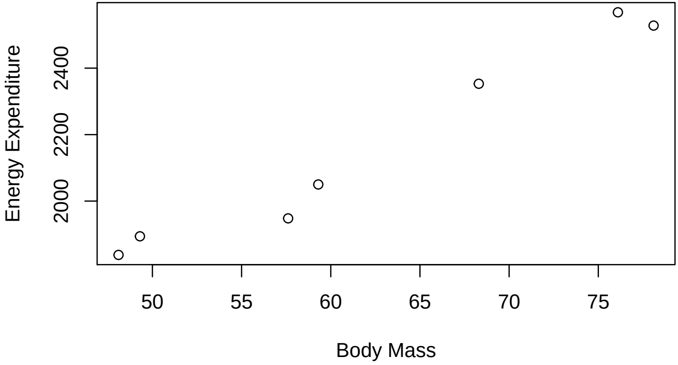
To investigate the dependence of energy expenditure (y) on body build, researches used underwater weighing techniques to determine the fat-free body mass (x) for each of seven men. They also measured the total 24-hour energy expenditure for each man during conditions of quiet sedentary activity. The results are shown in the table.

Subject	1	2	3	4	5	6	7
x	49.3	59.3	68.3	48.1	57.61	78.1	76.1
y	1,894	2,050	2,353	1,838	1,948	2,528	2,568

Line of Best Fit Example (cont.)

Let's find the line of best fit.

```
> plot(x,y, xlab = "Body Mass", ylab = "Energy Expenditure")
```



Line of Best Fit Example (cont.)

Recall:

For the line of best fit, $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$ where

$$\hat{\beta}_1 = \frac{S_{xy}}{S_{xx}} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}, \quad \text{and} \quad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}.$$

Solution:

```
> # First, compute x and y deviations
>     dev_x <- x - mean(x)
>     dev_y <- y - mean(y)
> # Next, compute sum of squares of xy and xx
>     Sxy <- sum(dev_x * dev_y)
>     Sxx <- sum(dev_x * dev_x)
```

Line of Best Fit Example (cont.)

Recall:

For the line of best fit, $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$ where

$$\hat{\beta}_1 = \frac{S_{xy}}{S_{xx}} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}, \quad \text{and} \quad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}.$$

Solution:

```
> # Compute the estimated slope
>     Sxy/Sxx
```

```
[1] 25.01184
```

```
> # Compute the estimated intercept
>     mean(y) - (Sxy/Sxx) * mean(x)
```

```
[1] 607.6539
```

Functions for Numeric Matrices

Useful R Functions

R Function	Description
`A %*% B`	Matrix multiplication for compatible matrices A, B.
`dim(A)`	Dimension of matrix A.
`t(A)`	Transpose of matrix A.
`diag(x)`	Returns a diagonal matrix with elements x along the diagonal.
`diag(A)`	Returns a vector of the diagonal elements of A.
`solve(A,b)`	Returns x in the equation $b = Ax$.
`solve(A)`	Inverse of A where A is a square matrix.
`cbind(A,B)`	Combine matrices horizontally for compatible matrices A, B.
`rbind(A,B)`	Combine matrices vertically for compatible matrices A, B.

System of Linear Equations Example

Solve the system of equations: $\begin{cases} 3x - 2y + z = -1 \\ x + \frac{1}{2}y - 12z = 2 \\ x + y + z = 3 \end{cases}$

Recall,

We can represent the system using matrices as follows:

$$\begin{pmatrix} 3 & -2 & 1 \\ 1 & \frac{1}{2} & -12 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -1 \\ 2 \\ 3 \end{pmatrix}.$$

Then we would like to solve for vector (x, y, z).

System of Linear Equations Example (cont.)

Recall,

$$\begin{pmatrix} 3 & -2 & 1 \\ 1 & \frac{1}{2} & -12 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -1 \\ 2 \\ 3 \end{pmatrix}$$

Solution:

```
> # Define matrix A
>     A <- matrix(c(3,1,1,-2,1/2,1,1,-12,3), nrow = 3)
> # Define vector b
>     b <- c(-1, 2, 3)
> # Use the solve function
>     solve(A, b)
```

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

System of Linear Equations Example (cont.)

Recall,

$$\begin{pmatrix} 3 & -2 & 1 \\ 1 & \frac{1}{2} & -12 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -1 \\ 2 \\ 3 \end{pmatrix}$$

Let's use matrix multiplication to check that $\mathbf{x} = \begin{pmatrix} 1 & 2 & 0 \end{pmatrix}^T$ is the correct solution to our system of equations.

Solution

```
> x <- c(1, 2, 0) # Define solution vector x
> A %*% x         # Then check with matrix multiplication
```

```
     [,1]
[1,]   -1
[2,]    2
[3,]    3
```

Element-wise Operations for Matrices

Let A and B be matrices of the same dimensions. Let a be a scalar.

Element-wise Operators

Operator	Description
`a + A`	Element-wise scalar addition
`a * A`	Element-wise scalar multiplication
`A + B`	Element-wise addition
`A * B`	Element-wise multiplication
`A ^ a`	Element-wise power
`a ^ A`	Element-wise exponentiation
`A ^ B`	Element-wise exponentiation

Filtering

Logical and Relational Operators

Logical Operator	Description
!	Negation (NOT)
&	AND
|	OR

Relational Operator	Description
<, >	Less than, greater than
<=, >=	Less than or equal to, greater than or equal to
==	Equal to
!=	Not equal to

Examples of Logical and Relational Operators

Some Basic Examples

```
> 1 > 3
```

```
[1] FALSE
```

```
> 1 == 3
```

```
[1] FALSE
```

```
> 1 != 3
```

```
[1] TRUE
```

Examples of Logical and Relational Operators

Some Basic Examples

```
> (1 > 3) & (4*5 == 20)
```

```
[1] FALSE
```

```
> (1 > 3) | (4*5 == 20)
```

```
[1] TRUE
```

Examples of Logical and Relational Operators

Some Basic Examples

```
> c(0,1,4) < 3
```

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

```
> which(c(0,1,4) < 3)
```

```
[1] 1 2
```

```
> which(c(TRUE, TRUE, FALSE))
```

```
[1] 1 2
```

Examples of Logical and Relational Operators

Some Basic Examples

```
> c(0,1,4) >= c(1,1,3)
```

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

```
> c("Cat","Dog") == "Dog"
```

```
[1] FALSE  TRUE
```

Filtering Examples

Sometimes we would like to extract elements form a vector or matrix that satisfy certain criteria.

Extracting Elements from a Vector

```
> w <- c(-3, 20, 9, 2)
> w[w > 3] ### Extract elements of w greater than 3
```

```
[1] 20  9
```

```
> ### What's going on here?
> w > 3
```

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

```
> w[c(FALSE, TRUE, TRUE, FALSE)]
```

```
[1] 20  9
```

Filtering Examples

```
> w <- c(-3, 20, 9, 2)
> ### Extract elements of w with squares between 3 and 10
> w[w*w >= 3 & w*w <= 10]
```

```
[1] -3  2
```

```
> w*w >= 3 ### What's going on here?
```

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

```
> w*w <= 10
```

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

```
> w*w >= 3 & w*w <= 10
```

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

Filtering Examples

Extracting Elements from a Vector

```
> w <- c(-1, 20, 9, 2)
> v <- c(0, 17, 10, 1)
> ### Extract elements of w greater than elements from v
> w[w > v]
```

```
[1] 20  2
```

```
> ### What's going on here?
> w > v
```

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

```
> w[c(FALSE, TRUE, FALSE, TRUE)]
```

```
[1] 20  2
```

Filtering Examples

Filtering Elements of a Matrix

```
> M <- matrix(c(rep(4,5), 5:8), ncol=3, nrow=3)
> M
```

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

```
> ### We can do element-wise comparisons with matrices too.
> M > 5
```

```
      [,1]  [,2] [,3]
[1,] FALSE FALSE TRUE
[2,] FALSE FALSE TRUE
[3,] FALSE FALSE TRUE
```

Filtering Examples

```
> M
```

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

```
> M[,3] < 8
```

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

```
> M[M[,3] < 8, ]
```

```
     [,1] [,2] [,3]
[1,]    4    4    6
[2,]    4    4    7
```

Filtering Examples

Reassigning Elements of a Matrix

```
> M
```

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

```
> ### Assign elements greater than 5 with zero
> M[M > 5] <- 0
> M
```

```
     [,1] [,2] [,3]
[1,]    4    4    0
[2,]    4    4    0
[3,]    4    5    0
```

Check Your Understanding

Using matrix z, what is the output of the following?

```
> z
```

```
     First Second Third
[1,]     1      1     9
[2,]     2      0    16
[3,]     3      1    25
```

1. `z[z[, "Second"], ]`?
2. `z[, 1] != 1`?
3. `z[(z[, 1] != 1), 3]`?

Check Your Understanding

```
> z
```

```
     First Second Third
[1,]     1      1     9
[2,]     2      0    16
[3,]     3      1    25
```

Solutions

```
> z[z[, "Second"], ]
```

```
     First Second Third
[1,]     1      1     9
[2,]     1      1     9
```

Check Your Understanding

```
> z
```

```
     First Second Third
[1,]     1      1     9
[2,]     2      0    16
[3,]     3      1    25
```

Solutions

```
> z[, 1] != 1
```

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

```
> z[(z[, 1] != 1), 3]
```

```
[1] 16 25
```

A Quick Note

```
> z
```

```
     First Second Third
[1,]     1      1     9
[2,]     2      0    16
[3,]     3      1    25
```

```
> z[(z[, 1] != 1), 3]
```

```
[1] 16 25
```

```
> z[(z[, 1] != 1), 3, drop = FALSE]
```

```
     Third
[1,]    16
[2,]    25
```

NA and NULL Values

NA and NULL

- `NA` indicates a missing value in a dataset.
- `NULL` is a value that doesn't exist and is often returned by expressions and functions whose value is undefined.

Example

```
> length(c(-1, 0, NA, 5))
```

```
[1] 4
```

```
> length(c(-1, 0, NULL, 5))
```

```
[1] 3
```

NA and NULL

Example

```
> ### Use na.rm = TRUE to remove NA values
> t <- c(-1,0,NA,5)
> mean(t)
```

```
[1] NA
```

```
> mean(t, na.rm = TRUE)
```

```
[1] 1.333333
```

```
> ### NA values are missing, but NULL values don't exist.
> s <- c(-1, 0, NULL, 5)
> mean(s)
```

```
[1] 1.333333
```

NA and NULL

NULL can be used is to build a vector in the following way:

```
> # Define an empty vector
>     x <- NULL
> # Fill in the vector
>     x[1] <- "Blue"
>     x[2] <- "Green"
>     x[3] <- "Red"
>     x
```

```
[1] "Blue"  "Green" "Red"
```

- NULL is commonly used to build vectors in loops with each iteration adding another element.
- Filling in pre-allocated space is less expensive (computationally) than adding an element at each step.
- Loops will be introduced next lecture.

A Note on Lists

Lists

A list structure combines objects of different modes.

Recall, in vectors and matrices all elements must have the same mode.

To define a list:

- Use the function "`list()`":

```
list(name1 = object1, name2 = object2, ...)
```

List component names (called tags) are optional.

Line of Best Fit Example

Recall, the energy expenditure versus fat-free body mass example one more time.

Example

To investigate the dependence of energy expenditure (y) on body build, researches used underwater weighing techniques to determine the fat-free body mass (x) for each of seven men. They also measured the total 24-hour energy expenditure for each man during conditions of quiet sedentary activity. The results are shown in the table.

Subject	1	2	3	4	5	6	7
x	49.3	59.3	68.3	48.1	57.61	78.1	76.1
y	1,894	2,050	2,353	1,838	1,948	2,528	2,568

Lists

Let's make a list of the values we've calculated for this example.

```
> # Combine data into single matrix
>   data <- cbind(x, y)
> # Summary values for x and y
>   sum_x <- summary(x)
>   sum_y <- summary(y)
> # We computed Sxy and Sxx previously
>   est_vals <- c(Sxy/Sxx, mean(y) - Sxy/Sxx*mean(x))
```

Lists

```
> # Define a list with different objects for each element
>   body_fat <- list(variable_data = data,
+                    summary_x = sum_x, summary_y = sum_y,
+                    LOBF_est = est_vals)
```

Extracting Components of Lists

Extract an individual component "c" from a list called `lst` in the following ways:

- `lst$c`
- `lst[[i]]` where "c" is the i^{th} component.
- `lst[["c"]]`

Extracting Components of Lists

Energy expenditure versus fat-free body mass example

```
> # Extract the first list element
> body_fat[[1]]
```

```
         x    y
[1,] 49.30 1894
[2,] 59.30 2050
[3,] 68.30 2353
[4,] 48.10 1838
[5,] 57.61 1948
[6,] 78.10 2528
[7,] 76.10 2568
```

Lists

Energy expenditure versus fat-free body mass example

```
> # Extract the Line of Best Fit estimates
> body_fat$LOBF_est
```

```
[1]  25.01184 607.65386
```

```
> # Extract the summary of x
> body_fat[["summary_x"]]
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  48.10   53.46   59.30   62.40   72.20   78.10
```

Optional Reading

- (Chapters 1-2) Phil Spector, *Data Manipulation with R,* Springer, 2008.
- (Chapters 1-4) Norman Matloff, *The Art of R Programming: A Tour of Statistical Software Design.*