

MATH411 | Fall 2018 | Chapter 1: An Introduction to R

Dr. Yongtao Cao

August 27, 2018

Contents

1.1 The Background	1
1.2 R Basics	4
1.3 Data Types and Structures	11
1.4 Flow Control	25
1.5 Functions	31

1.1 The Background

A scientific research problem as statistical thinking, generally, includes

1. Formulate a real problem in statistical terms.

2. Collect data efficiently and effectively.
3. Analyze data to extract the maximum amount of useful information.
4. Interpret and report the results.

So, eventually we'll be working with **data** (soon or late, small or big).

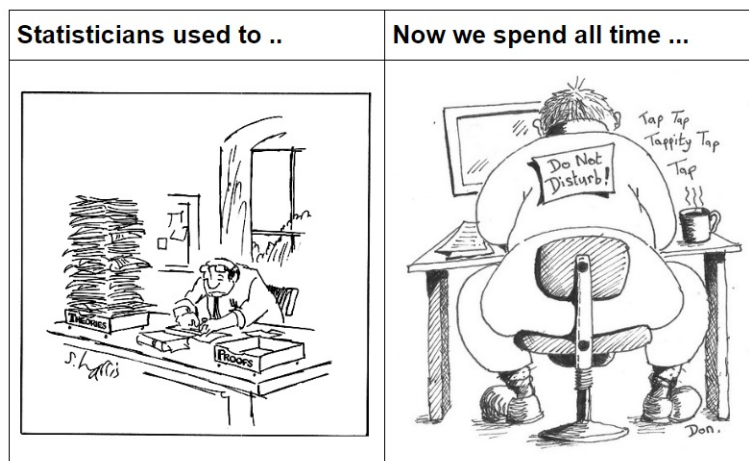


Figure 1: How do we work with data?

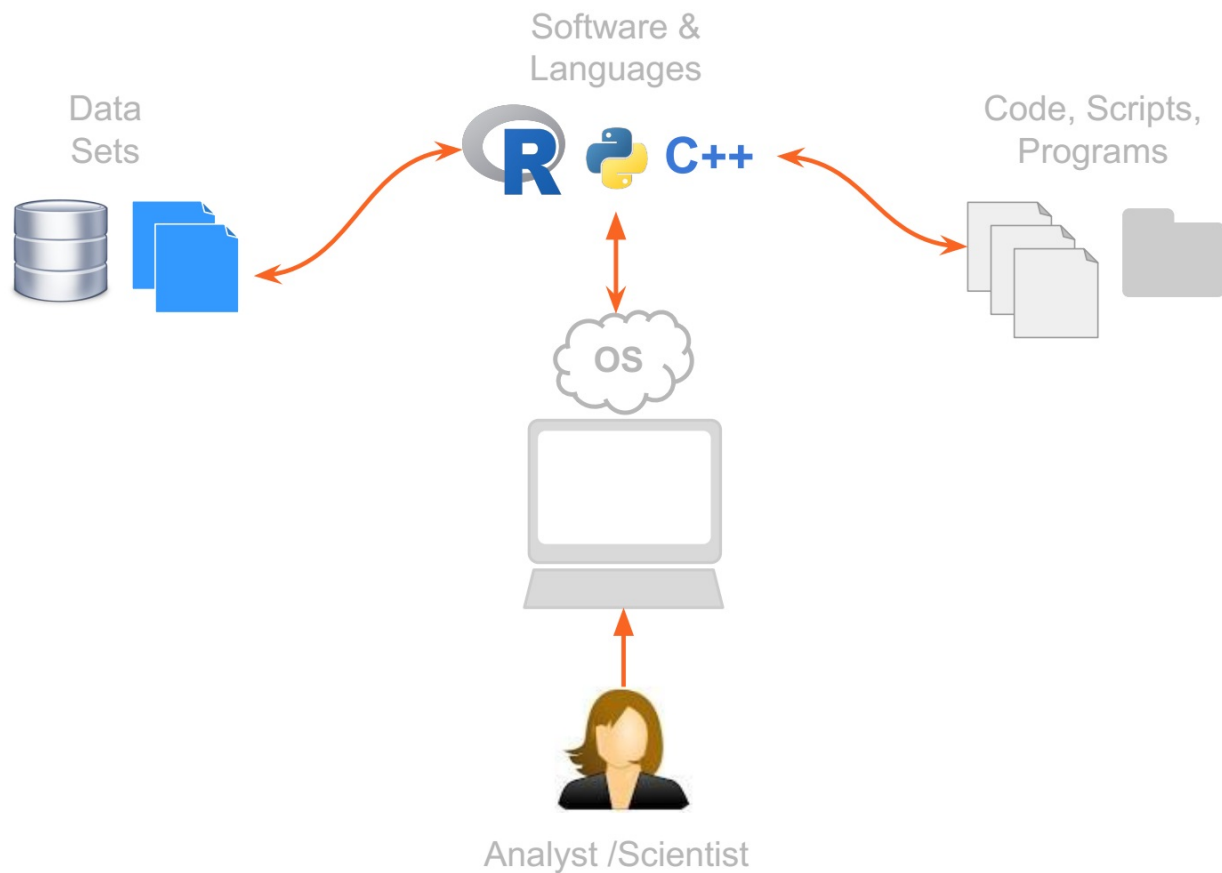


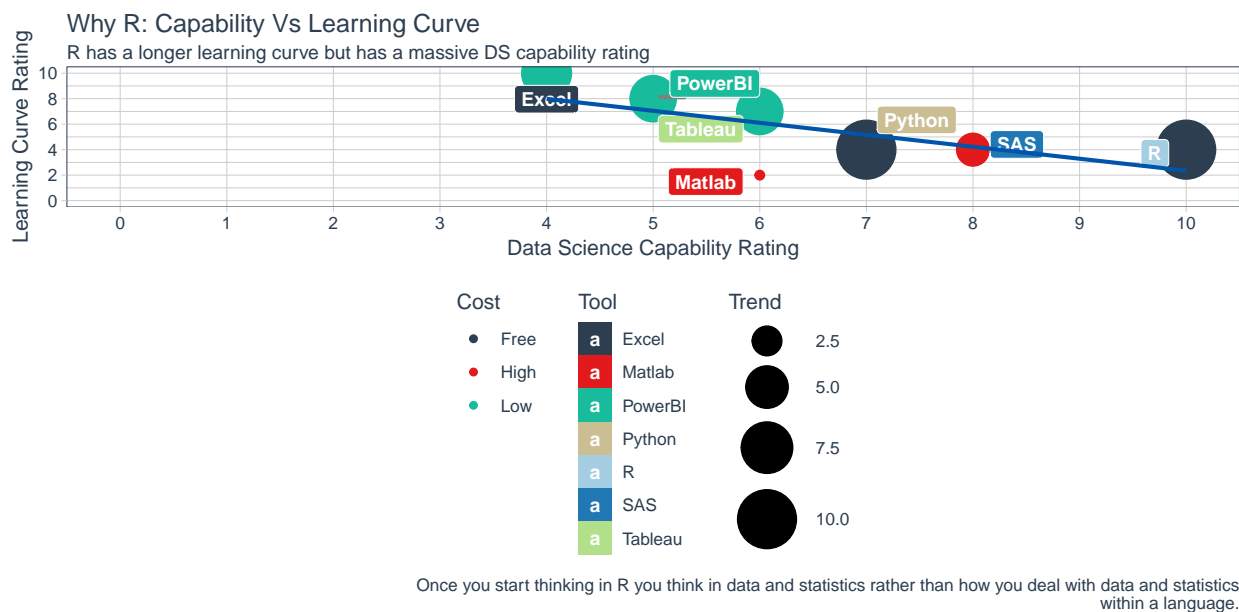
Figure 2: Essential skills for modern statisticians/data scientists

1.1.1 What is R?

R is a free **language** and software **environment** for **statistical computing and graphics**, and is highly **extensible**.

- Home: <https://www.r-project.org/>

1.1.2 Why use R?



- Read this article: <http://www.business-science.io/business/2017/12/27/six-reasons-to-use-R-for-business.html>.

1.2 R Basics

1.2.1 The R Console

The console window is the place where R is waiting for you to tell it what to do, and where it will show the results of a command. You can type commands directly into the console, but they will be forgotten when you close the session.

1. If R is ready to accept commands, the R console shows a >

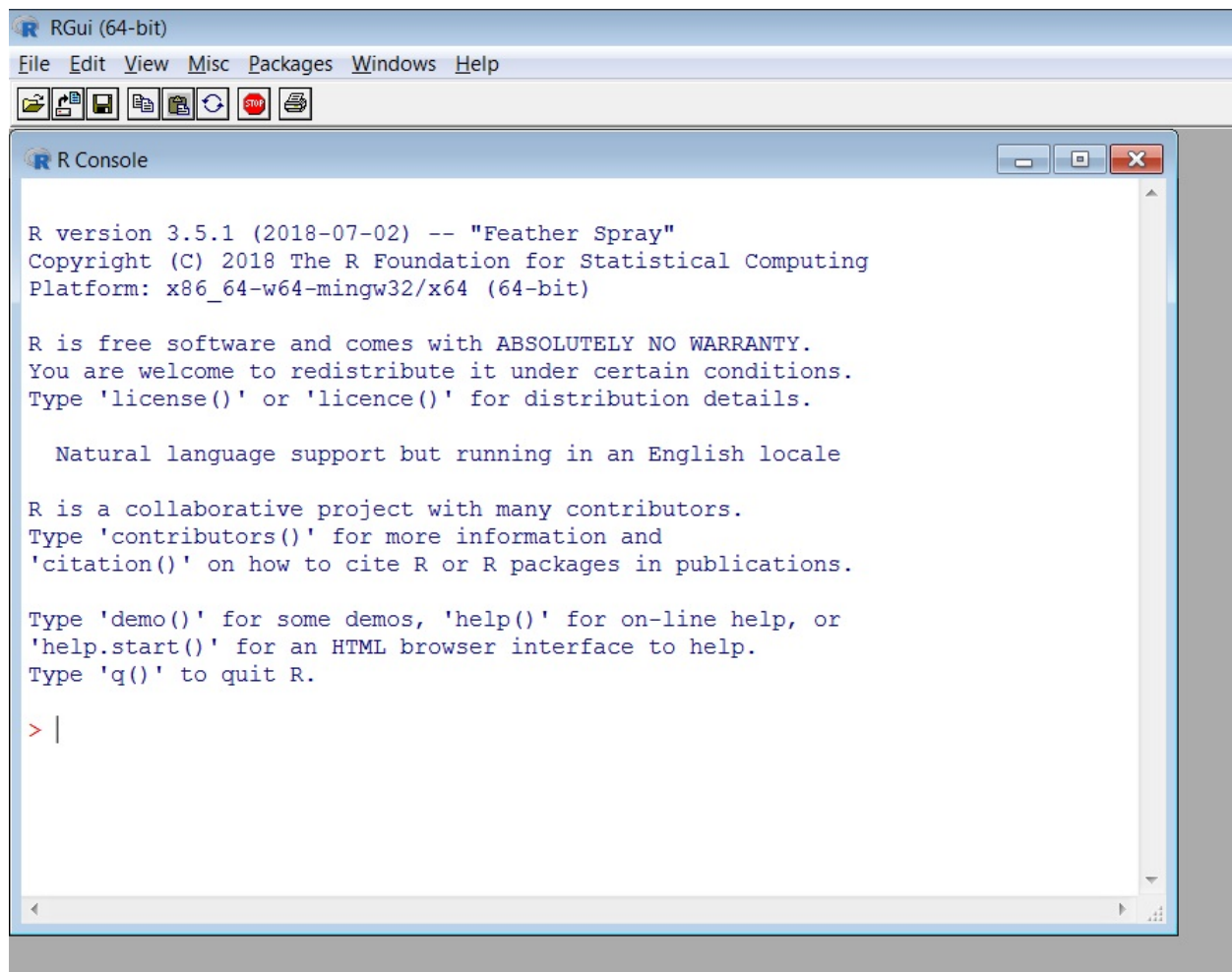


Figure 3: The R console

prompt. The result is given upon pressing the **Enter** key.

- When the console receives a command by directly typing into the console or running from the script editor, R will try to execute it.
- After running, the console will show the results and come back with a new `>` prompt to wait for new commands.

2. Console is waiting for you to enter more data: `+`.

- If R is still waiting for you to enter more commands because it isn't complete yet, the console will show a `+` prompt. It means that you haven't finished entering a complete command. Often this can be due to you having not 'closed' a parenthesis or quotation.

3. Escaping a command and getting a new prompt: press **esc** or the **stop** button.

- If you're in R console and you can't figure out why your command isn't running, you can press **esc** to escape the command and bring back a new prompt `>`.

4. Comments: text behind a `#` is a comment.

5. Clear the current console by using **Ctrl+L**.

6. Can use `up arrow` (`down arrow`) to see you previous command lines.

Some very useful functions you want to get familiar with

1. Here are some standard commands for managing your work space.

- `getwd()` print the current working directory.
- `ls()` list the objects in the current work space.
- `setwd("c:/docs/mydir")` change to mydirectory
- `rm()` removes objects from your work space; `rm(list = ls())` removes them all.
- `sessionInfo()` gives information about your session, i.e., loaded packages, R version, etc.

2. View and set options for the session.

- `?options`(or `help(options)`) learn about available options.
- `options()` view current option settings.
- `options(digits = 4)` number of digits to print on output.

3. Memory usage

- `memory.size()` gives the total amount of memory currently used by R.
- `memory.limit()` without any argument gives the limit of memory used by R. This can also be used to increase the limit. The maximum amount is limited by the memory of the computer.

4. The Package System

- `install.packages("package_name")` download and install a package (required once per computer).
- `library(package_name)` load a package in R (required once per R session).

1.2.2 R as a Scientific Calculator

The simplest use of R is as a calculator. Basic operations that you can do in R are listed in the table below:

Command	Meaning
<code>+, -, *, \</code>	add, subtract, multiply, divide
<code>^</code>	raise to the power of
<code>%/%</code>	integer division (ex: <code>6000 %/% 365 = 16</code>)
<code>%%</code>	remainder after division (ex: <code>6000 %% 365 = 160</code>)

Command	Meaning
<hr/>	
<code>()</code>	change the order of operations
<code>log()</code> , <code>exp()</code>	logarithms and exponents (ex: <code>log(10) = 2.302</code>)
<code>cos()</code> , <code>sin()</code> , <code>tan()</code>	trigonometry (ex: <code>sin(pi/2)</code>)
<code>sqrt()</code>	square root
<code>abs()</code>	absolute value
<code>floor()</code> , <code>ceiling()</code>	round down or round up
<code>round()</code>	round to the nearest whole number (ex: <code>round(2.3)</code>)

1.2.3 Using an Editor

When you write a program, it normally does not consist of only one or two lines. Therefore it is a good idea to write the code of the program in a text editor and then run the whole code or parts of it in R. In this class, we'll mostly enter the commands in the R script editor.

- As an example, let's open the **Chapter 1 R code** file and start to learn and use R.

1.2.4 Objects in R

In order to do something with data, we will need to tell R what to call it, so that we can refer to it in our code. In programming, we typically have **variables (objects)** (things that may vary) and **values** (our data).

In R, assigning of values to variables takes the following form

```
variable.name <- value
```

- The assignment operator `<-` can be thought of as storing the value/thing on the RHS into the name/thing on the LHS.
- If you've read some R code already, you've possibly seen that both `<-` and `=` are used to assign values to objects, and this tends to cause some confusion. Technically, **R will accept either when assigning variables**, so in that respect it comes down to a matter of style – I will use and highly recommend you to use `=`.
- The big difference comes when using functions that take **arguments** – there you should only use `=` to specify what the value of the **argument**.

What to Name Objects?

`?make.names`

You can see object names cannot:

- Start with a number
- Contain a space
- Contain **reserved** words (`?Reserved`)

Object names must **start with a letter** and can contain **letters**, **numbers**, **_** and **.**

Case Matters

`my_names != My_names`

1.3 Data Types and Structures

As mentioned before, we'll be working with **Data**. Thus, we need to understand

1. How do statisticians/data scientists think about data?

Hierarchy of Data Types

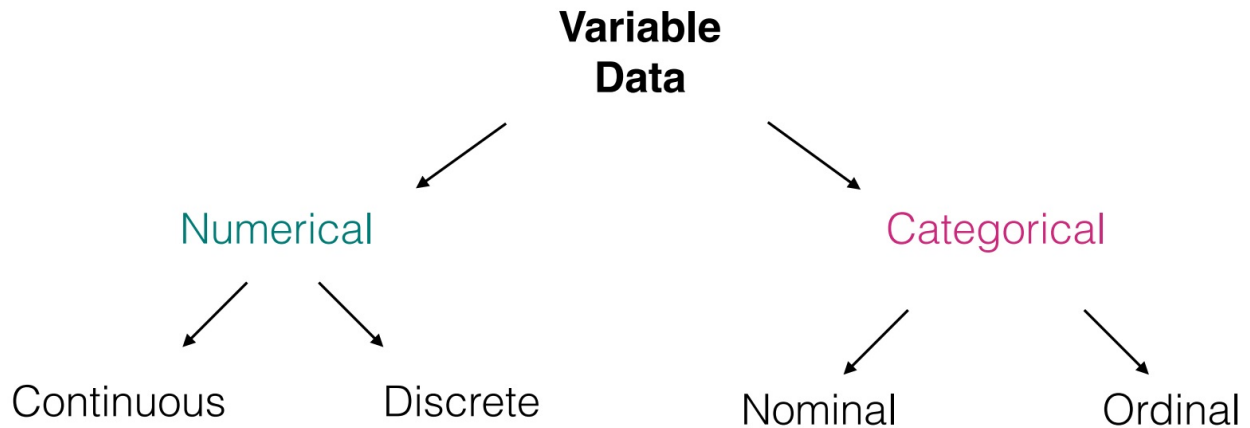


Figure 4: Data Types

2. How do computers treat data?

Data (for computers): at the lowest level, computers treat all kinds of data in binary format: 0's and 1's.

3. How do programming languages handle data?

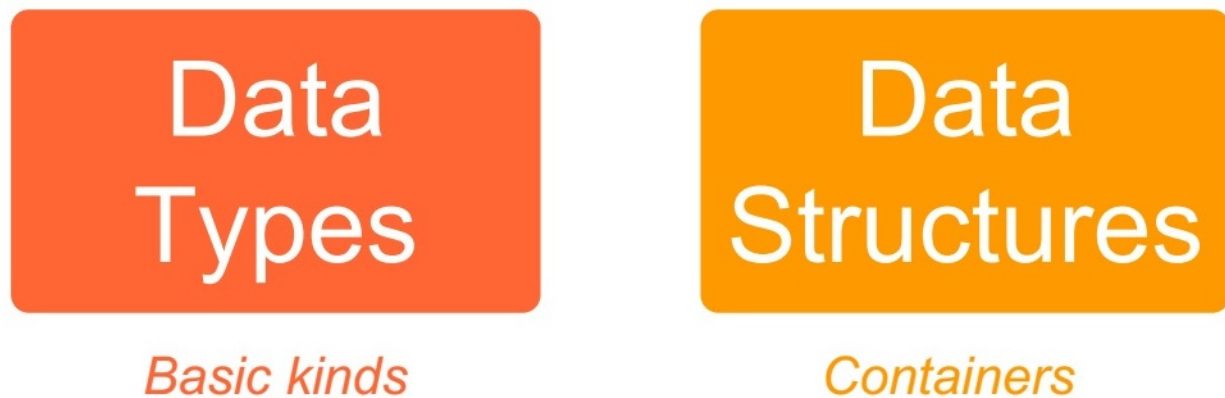


Figure 5: Data for Software / Languages?

Data Types (for programming languages): also refer to as **data**

primitives or **primitive types**. They serve as the building blocks (i.e. they are like the **atoms**)

The six data types that R uses include:

1. **integer** for **whole numbers** (e.g., 2L, the L indicates to R that it's an integer).
2. **Double** for **real, decimal numbers**.
3. **character** for **text values**, denoted by using quotes (” or “”) around value.
4. **logical** for **TRUE** and **FALSE** (the **Boolean** data type).
5. **complex** to represent complex numbers with real and imaginary parts (e.g., 1+4i) and that's all we're going to say about them.
6. **raw** that we won't discuss further.

The table below provides examples of each of the commonly used data types:

Data Type	Examples
Integer	2L, 500L, -17L
Double	1, 1.5, 20, pi

Data Type	Examples
Character	“anytext”, “5”, “TRUE”
Logical	TRUE, FALSE, T, F

There are some special data values in R

- `NULL` = null object
- `NA` = Not Available (missing value)
- `Inf` = infinity
- `NaN` = Not a Number (different from `NA`)

Data Structures in R

1.3.1 Vectors

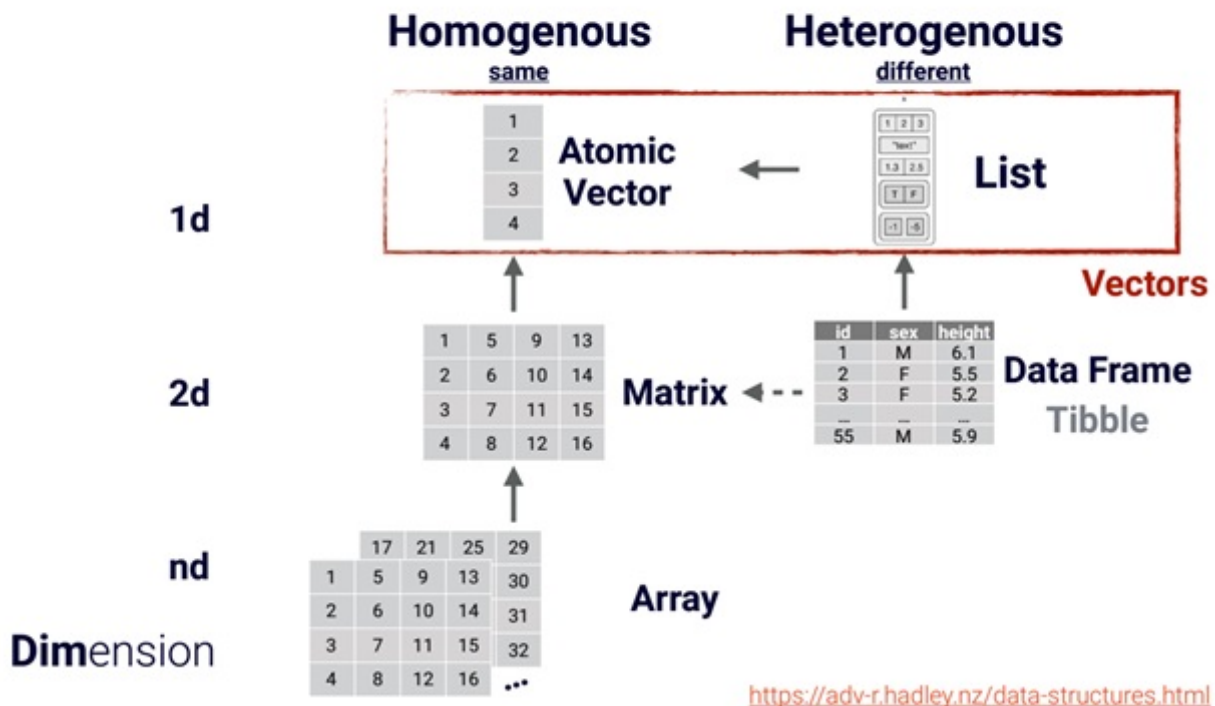
A **vector** (or one-dimensional array) is a **collection** of values (or elements) of the **same type**, each identified by an **index** in the range `1` to `length(vec)`.

1.3.1.1 Creating vectors

See R code.

Data Structures

... how you can build data ...



1.3.1.2 Functions on Vectors

Command	Description
<code>length(vec)</code>	the number of elements in <code>vec</code>
<code>sum(vec)</code>	sums up all the elements of <code>vec</code>
<code>mean(vec)</code>	mean of <code>vec</code>
<code>median(vec)</code>	median of <code>vec</code>
<code>min(vec), max(vec)</code>	the largest or smallest element of <code>vec</code>
<code>range(vec)</code>	returns the smallest and largest values together
<code>sd(vec), var(vec)</code>	the standard deviation and variance of <code>vec</code>
<code>sort(vec)</code>	returns the <code>vec</code> in sorted order
<code>order(vec)</code>	returns the index that sorts the vector <code>vec</code>
<code>unique(vec)</code>	lists the unique elements of <code>vec</code>
<code>summary(vec)</code>	gives a five-number summary
<code>any(vec), all(vec)</code>	useful on Boolean vectors
<code>which.max(vec)</code>	The index of the maximum value (first index if ties)
<code>which.min(vec)</code>	The index of the minimum value (first index if ties)
<code>duplicated(vec)</code>	lists the duplicated elements of <code>vec</code>

1.3.1.3 Subsetting and Indexing

Subsets are a way to take a selection of values in a larger collection and create a smaller collection with them.

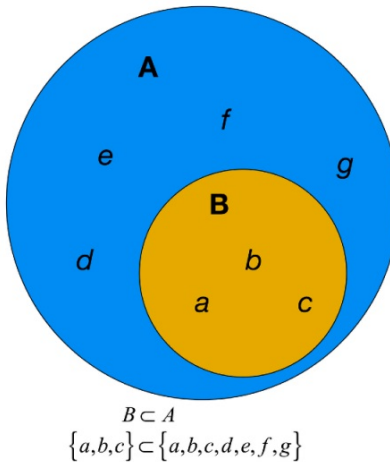


Figure 6: The idea of subsetting

We can extract elements from vectors using square brackets.

- See R code for examples.

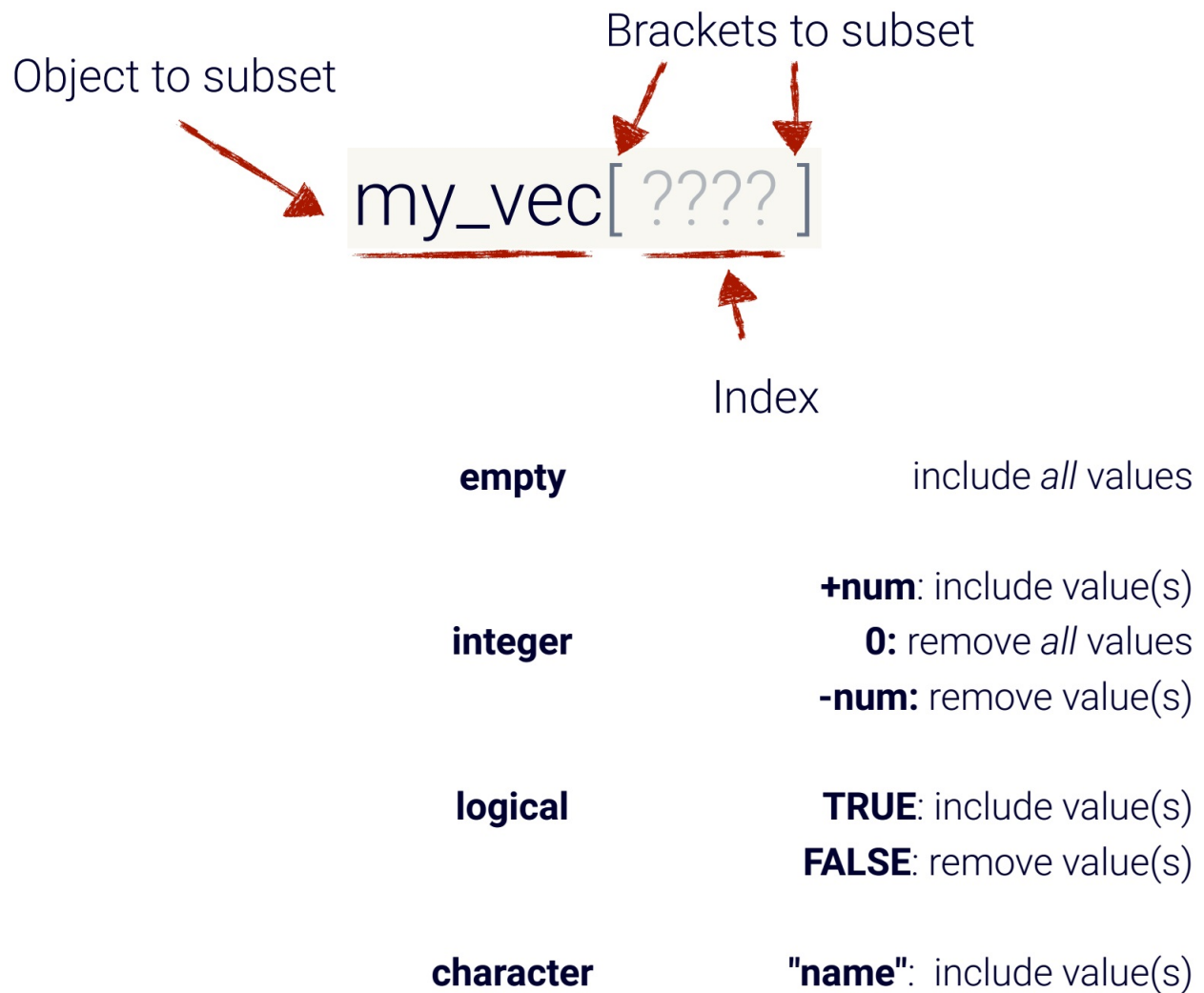


Figure 7: Brackets Subset

1.3.1.4 Vectorization

Vectorization refers to performing work on multiple items at the same time.


$$f\left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_{n \times 1}\right) = \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{bmatrix}_{n \times 1}$$

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}_{4 \times 1} + \begin{bmatrix} 5 \\ 6 \\ 7 \\ 8 \end{bmatrix}_{4 \times 1} = \begin{bmatrix} 6 \\ 8 \\ 10 \\ 12 \end{bmatrix}_{4 \times 1}$$
$$\begin{array}{rclcl} 1 & + & 5 & = & 6 \\ 2 & + & 6 & = & 8 \\ 3 & + & 7 & = & 10 \\ 4 & + & 8 & = & 12 \end{array}$$

Figure 8: Vectorization: vectors in, vectors out

- Binary Vectorized Operators

```
x = c(1, 2, 3, 4); y = c(5, 6, 7, 8)
x + y                                # Addition
# [1] 6 8 10 12
x - y                                # Subtraction
# [1] -4 -4 -4 -4
x * y                                # Multiplication
# [1] 5 12 21 32
x / y                                # Division
# [1] 0.20 0.333 0.429 0.500
x ^ y                                # Exponentiation
# [1] 1 32 729 65536
x %/% y                              # Integer Division
# [1] 0 0 0 0
x %% y                               # Modulus
# [1] 1 2 3 4
```

Figure 9: Element-wise Changes

- Comparisons








$x == y$	equal to	
$x != y$	not equal to	
$x < y$	less than	
$x > y$	greater than	
$x \leq y$	less than or equal to	
$x \geq y$	Greater than or equal to	
$x \%in\% y$	x is apart of y	

Figure 10: Kinds of Comparisons

- See R code for examples.

1.3.1.5 Character Vectors

Character is a single symbol that is displayed. Examples like, “i”, “U”, “P”, “1”, “*”, ...

String is multiple characters combined together. Ex: “IUP”, “Hello World!”

- Using special characters

Symbol	Description	Symbol	Description
<code>\n</code>	<i>newline</i>	<code>\\</code>	<i>backslash \</i>
<code>\r</code>	carriage return	<code>\'</code>	ASCII apostrophe '
<code>\t</code>	tab	<code>\"</code>	ASCII quotation mark "
<code>\b</code>	backspace	<code>\`</code>	ASCII grave accent (backtick) `
<code>\a</code>	alert (bell)	<code>\nnn</code>	character with given octal code (1, 2 or 3 digits)
<code>\f</code>	form feed	<code>\xnn</code>	character with given hex code (1 or 2 hex digits)
<code>\v</code>	vertical tab	<code>\unnnn</code>	Unicode character with given code (1–4 hex digits)

Figure 11: Escape Characters

- See R code for examples.

Regular Expressions (**regex**) are sequences of characters that defines a search pattern to find or replace values contained in a collection of character (string) values.

- A much nicer way to work with regex is to use the **stringr** package.

Function	Description
<code>str_detect(x, pattern)</code>	Any pattern matches?
<code>str_count(x, pattern)</code>	How many matches?
<code>str_subset(x, pattern)</code>	What are the matching patterns?
<code>str_locate(x, pattern)</code>	Where are the matching patterns?
<code>str_extract(x, pattern)</code>	What is the matching value?
<code>str_match(x, pattern)</code>	What is the matching group?
<code>str_replace(x, pattern, replacement)</code>	What should replace the pattern?
<code>str_split(x, pattern)</code>	How should the string be split?

Figure 12: Some functions in stringr package

- See R code for examples.

1.3.2 Matrix

A **matrix** in R is a collection of vectors of **same length and identical data type**. Vectors can be combined as columns in the matrix or by row, to create a 2-dimensional structure.

Basic Matrix Functions and Operations

In the following examples, \mathbf{A} and \mathbf{B} are matrices and \mathbf{x} and \mathbf{b} are a vectors.

Operator or Function	Description
<hr/>	
<code>dim(A)</code>	returns the number of rows and columns of \mathbf{A}
<code>nrow(A)</code>	returns the number of rows of \mathbf{A}
<code>ncol(A)</code>	returns the number of columns of \mathbf{A}
<code>A * B</code>	Element-wise multiplication
<code>A %*% B</code>	Matrix multiplication
<code>A %o% B</code>	Outer product. \mathbf{AB}'
<code>crossprod(A,B)</code>	$\mathbf{A}'\mathbf{B}$
<code>crossprod(A)</code>	$\mathbf{A}'\mathbf{A}$
<code>t(A)</code>	Transpose
<code>diag(x)</code>	Creates diagonal matrix with elements of \mathbf{x} in the principal diagonal
<code>diag(A)</code>	Returns a vector containing the elements of the principal diagonal
<code>diag(k)</code>	If k is a scalar, this creates a $k \times k$ identity matrix. Go figure.

Operator or Function	Description
<code>solve(A, b)</code>	Returns vector \mathbf{x} in the equation $\mathbf{b} = \mathbf{Ax}$ (i.e., $\mathbf{A} - \mathbf{1b}$)
<code>solve(A)</code>	Inverse of \mathbf{A} where \mathbf{A} is a square matrix.
<code>ginv(A)</code>	Moore-Penrose Generalized Inverse of \mathbf{A} . (requires loading the MASS package)
<code>cbind(A,B,...)</code>	Combine matrices(vectors) horizontally. Returns a matrix.
<code>rbind(A,B,...)</code>	Combine matrices(vectors) vertically. Returns a matrix.
<code>rowMeans(A)</code>	Returns vector of row means.
<code>rowSums(A)</code>	Returns vector of row sums.
<code>colMeans(A)</code>	Returns vector of column means.
<code>colSums(A)</code>	Returns vector of column sums.
<code>R = chol(A)</code>	Choleski factorization of \mathbf{A} . Returns the upper triangular factor, such that $\mathbf{R}'\mathbf{R} = \mathbf{A}$.
<code>y = qr(A)</code>	QR decomposition of \mathbf{A} .
<code>y = eigen(A)</code>	<code>y\$val</code> contains eigenvalues and <code>y\$vec</code> contains eigenvectors

Operator or

Function

Description

`y = svd(A)`

Single value decomposition of **A**.

- See R code for more details.

1.3.3 Lists

A **list** is an ordered collection of components **not necessarily of the same type**.

- See R code for more details.

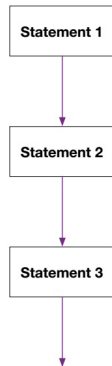
1.4 Flow Control

Flow of Control specifies the direction the program takes or flows when given conditions and parameters.

A **control structure** is a piece of code whose analysis of a value results in a choice being made as to the direction the program should go.

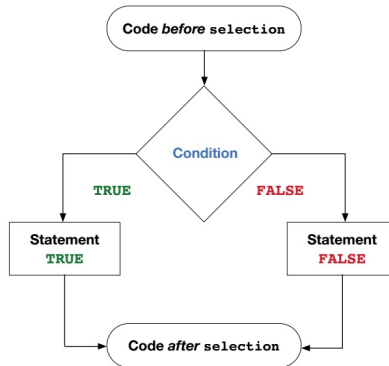
There are 3 kinds of structures in practice.

Sequential



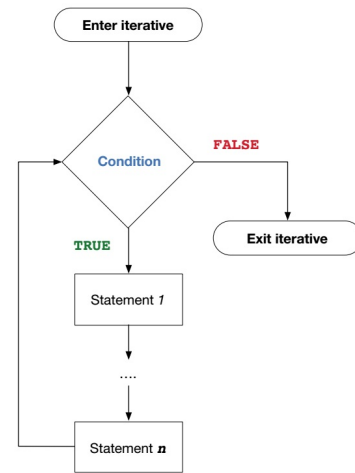
"One foot in front of the other"

Selection



"Making a choice"

Iterative



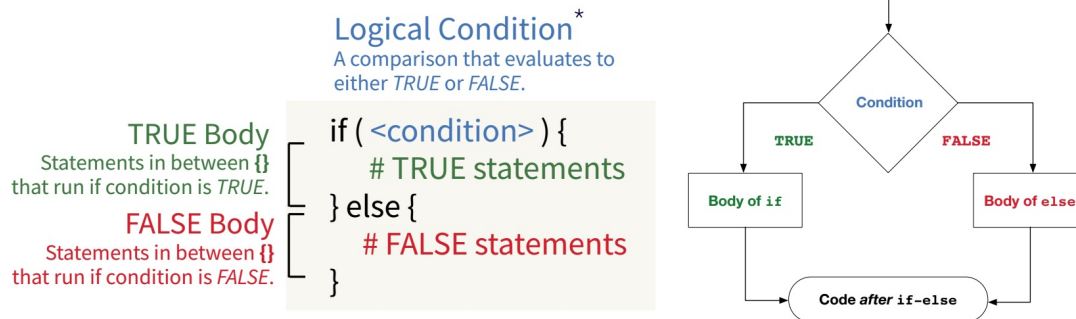
"Repeating yourself"

Figure 13: How Computers Process Information

1.4.1 Conditional Execution

Conditional execution is performed in R using the command/function `if`.

The if-else Template



* The **logical condition** must be of length 1 for the if-else structure. You may wish to use **&&** and **||** to ensure *logical* values of length 1 or use **ifelse(condition, true-statement, false-statement)** for a *vectorized* version.

Figure 14: Allowing for a Choice

The if-else ... if-else Template

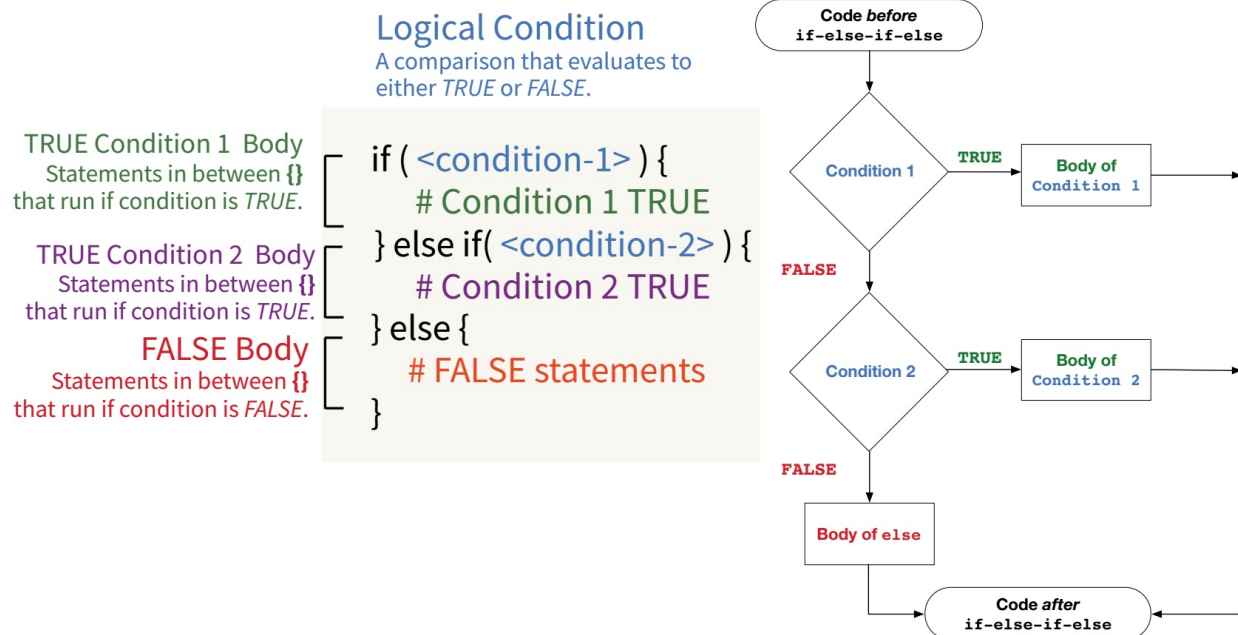


Figure 15: Allowing for Multiple Choices

- **Practice 1:** VAT has different rate according to the product purchased. Imagine we have three different kind of products with different VAT applied:

Categories	Products	VAT
A	Book, magazine, newspaper	8%
B	Vegetable, meat, beverage	10%
C	Tee-shirt, jean, pant	20%

Write a chain to apply the correct VAT rate to the product a customer bought.

- **Practice 2:** Classifying Blood Pressure. **Systolic** measures the amount of pressure in the veins when the heart beats, which can be classified as

Systolic	BP Type
Below 120	Normal
120-129	Elevated
130-139	Stage 1 / Hypertension
140+	Stage 2 / Hypertension

Write an **if-else if-else** statement for the BP type classification.

Vectorized ifelse Template

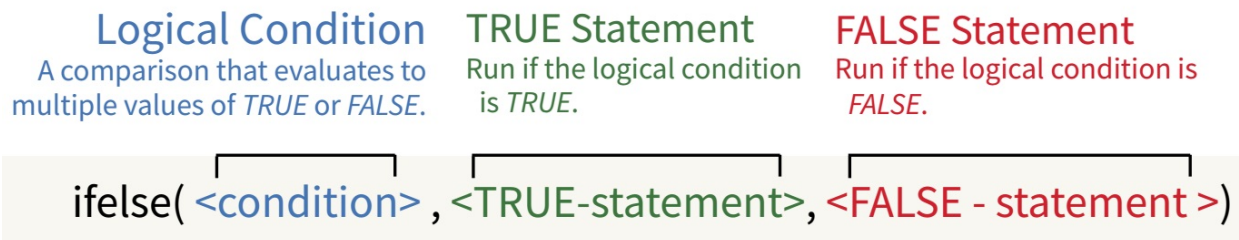


Figure 16: Allowing Simultaneous Choice on Data

1.4.2 Repetitive Execution

Iteration is a computational structure that allows the computer to repeat the same instruction multiple times.

There are three basic iteration structures in R.

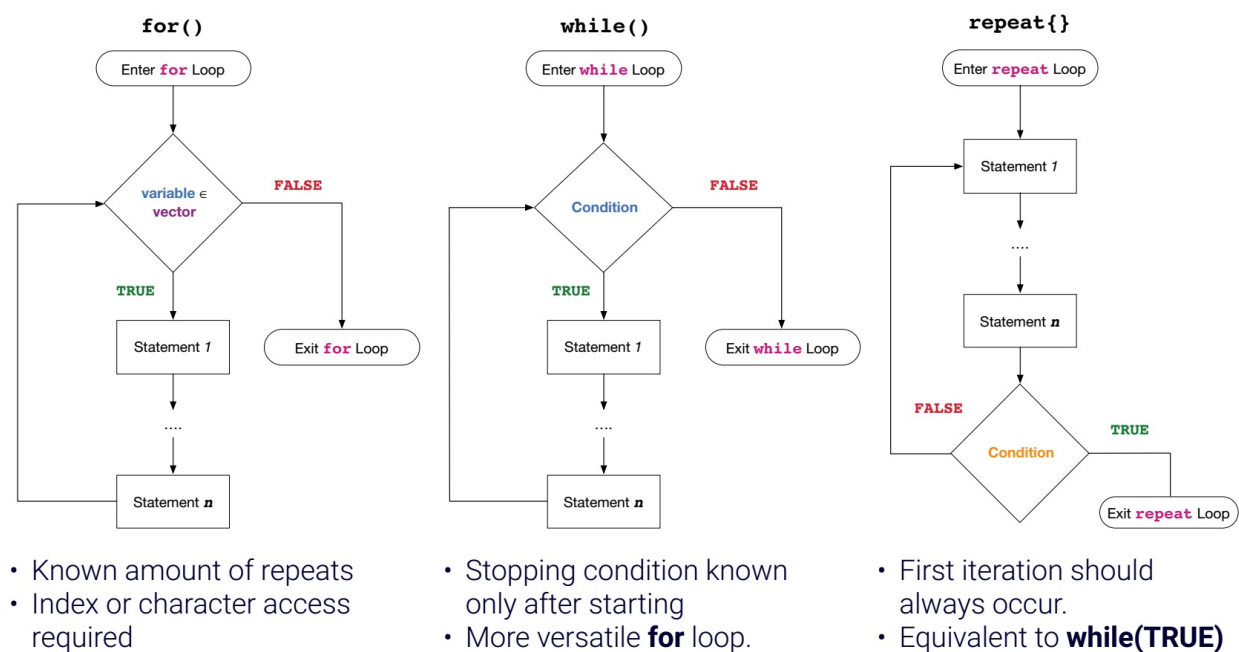


Figure 17: An Overview of Looping

In this class, we only focus on the **for** loop.

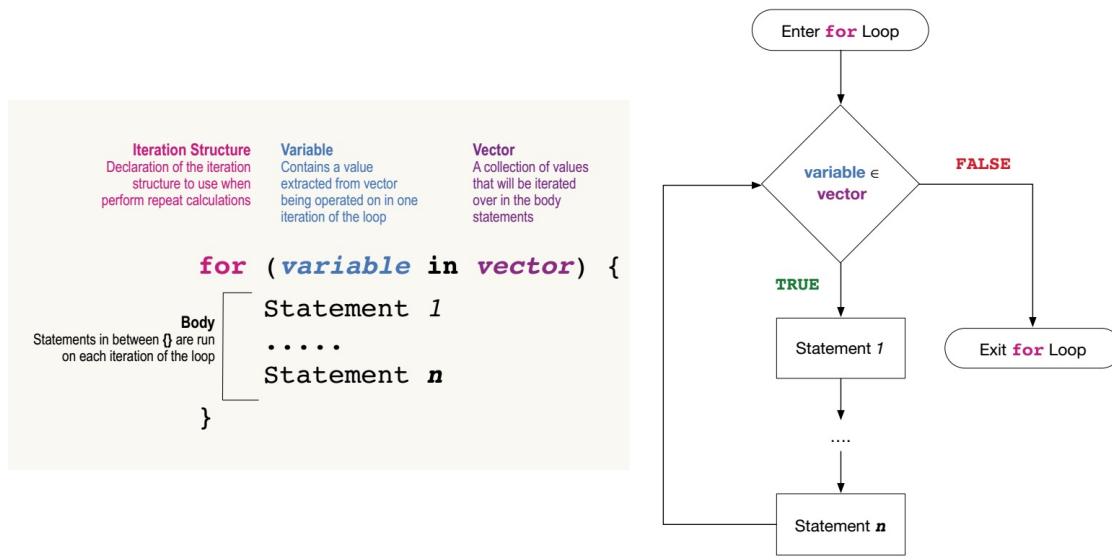
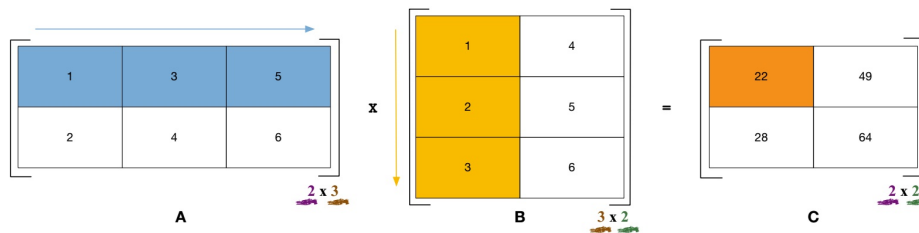


Figure 18: For Template

• Practice: Matrix Multiplication



If **A** is an $n \times m$ matrix and **B** is an $m \times p$ matrix, then **C = AB** is an $n \times p$ matrix. The elements of **C** are computed with

$$c_{ij} = \sum_{k=1}^m a_{ik} b_{kj}$$

Figure 19: Multiplying Matrices

1.5 Functions

One of the biggest advantages of R is its immense flexibility: if you are unhappy with a way something is implemented or if you think something is missing, you can simply define your own **function**.

A function should be

- written to carry out a specified task
- may or may not include arguments
- **contain a body**
- may or may not return one or more values

A general approach to a function is to use the **argument part** as **inputs**, feed the **body part** and finally return an **output**. See R code for more details. . .

To write a function:

```
function.name = function (arguments) {  
## do something with arguments  
## return results  
}
```

- **Practice 1:** Write a function to compute the Pearson correlation coefficient, r . Given two vectors **x** and **y**, we want to

apply the following formula

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

- **Practice 2:** Write a function to calculate a student's letter grade, `letter.grade` for a given vector of scores between 0 and 100 based on the following rules.

- If grade ≥ 90 , assign A
- Otherwise, if grade ≥ 80 , assign B
- Otherwise, if grade ≥ 70 , assign C
- In all other cases, assign F