

STA 141A: Fundamentals of Statistical Data Science

Course Material Summary *Part 1*

University of California at Davis

Last Edit Date: 06/23/2022

Disclaimer and Term of Use:

1. We do not guarantee the accuracy and completeness of the summary content. Some of the course material may not be included, and some of the content in the summary may not be correct. You should use this file properly and legally. We are not responsible for any results from using this file.
2. Although most of the content in this summary is originally written by the creator, there is still some of the content that is adapted (derived) from the slides and codes from *Professor Emanuela Furfaro*. We use those as references and quotes in this file. Please [contact us](#) to delete this file if you think your rights have been violated.
3. This work is licensed under a [Creative Commons Attribution 4.0 International License](#).

STA 141A Summary Part 1

I. R Basics

1. Working directory

```
getwd()
setwd("/path to working directory")
```

2. Arithmetic Operations

+ (addition), - (subtraction), * (multiplication), / (division), %% (remainder), %/% (integer division), ^ (exponent), log(x, base), factorial(x), exp(x), sin(), cos(), tan(), sqrt(), max(), min(), sum(), mean(), median(), quantile(x, prob), round(x, n), rank(), signif(x, n), var(), sd(), cor().

3. Assign value

We can use "=" or "<-" to assign value to a variable.

4. Print variable

```
print(x)
```

Output: 1

5. Print and assign

```
(x <- 2)
```

Output: 2

6. Workspace functions

Use ls() or objects() to show all the objects in the workspace.

```
ls()
objects()
```

rm(y, z) removes variables y, z,
rm(list=ls()) removes everything.

```
rm(y, z)
rm(list=ls())
```

7. R is case sensitive.

II. Vectors

1. Create vectors

```
c(2,4,6)
2:6
seq(2,3,by=0.5)
rep(1:2, times=3)
rep(1:2, each=3)
```

```
2 4 6
2 3 4 5 6
2.0 2.5 3.0
1 2 1 2 1 2
1 1 1 2 2 2
```

Join elements into a vector
An integer sequence
A complex sequence
Repeat a vector
Repeat elements of a vector

2. Types

We can use class() or mode() to check the type of a vector. Note that a vector has to have elements that are the same mode. The vector is coerced to the most flexible type. Character is the most flexible.

We can also use the following command to convert types:

as.logical	TRUE, FALSE, TRUE	Boolean values (TRUE or FALSE).
as.numeric	1, 0, 1	Integers or floating point numbers.
as.character	'1', '0', '1'	Character strings. Generally preferred to factors.
as.factor	'1', '0', '1', levels: '1', '0'	Character strings with preset levels. Needed for some statistical models.

3. Vector structure

```
length(x) #return the length of the vector x
head(x) #return the first 6 elements of vector x
head(x, n) #return the first n elements of vector x
tail(x) #return the last 6 elements of vector x
tail(x, n) #return the last n elements of vector x
```

4. Sort vector

```
sort(x) #sort increasingly
sort(x, decreasing = TRUE) #sort decreasingly
```

5. Sampling

```
sample(x, size, replace = T/F)
```

6. Recycle

Shorter vectors are recycled.

```
z = 1:8
x = 1:4
x+z
```

Output:
2 4 6 8 6 8 10 12

7. Logical vectors

OR: |

```
x <- 11:20
x<15|x>18
```

Output: TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE TRUE TRUE

AND: &

```
x<18&x>15
```

Output: FALSE FALSE FALSE FALSE FALSE TRUE TRUE FALSE FALSE FALSE

NOT: !

```
x != 15
```

Output: TRUE TRUE TRUE TRUE FALSE TRUE TRUE TRUE TRUE TRUE

```
!(x<18&x>15)
```

Output: TRUE TRUE TRUE TRUE FALSE TRUE TRUE TRUE TRUE TRUE

%in%: This operator is used to identify if an element belongs to a vector.

```
15 %in% x
```

Output: TRUE

8. Searching for elements in the vector

which() takes a logical vector, and returns an integer vector of positions, the position of the element/elements that satisfy a condition.

```
which(x<16|x>18)
```

Output: 1 2 3 4 5 9 10

```
which(x<16)
```

Output: 1 2 3 4 5

9. Calculate sum and mean of logical vector

```
sum(x<15)
```

Output: 4

```
mean(x<15)
```

Output: 0.4

10. Subset

1) Subset using a logical vector

```
x <- c(1,2,3,4,5)
x[c(TRUE, TRUE, FALSE, FALSE, TRUE)] #Output: 1,2,5
x[x<3] #Output: 1,2
x[x<3|x>4] #Output: 1,2,5
```

2) Subset using elements position

```
x <- c(1,2,3,4,5)
x[2] #Output: 2
x[-2] #Output: 1,3,4,5
x[c(1,5)] #Output: 1,5
x[-c(1,5)] #Output: 2,3,4
```

3) Subset using the name

```
x <- c(time1 = 2.5, time2 = 3.2)
x['time1'] #Output: ## time1 ## 2.5
```

11. Equality of vectors

```
x <- 1:3
y <- c(1,2,4)
x==y #Output: TRUE TRUE FALSE
x[x==y] #Output: 1 2
```

12. Removing NA

```
x<-c(1,2,NA,10,7)
sum(x,na.rm=TRUE) #Output: 20
mean(x, na.rm=TRUE) #Output: 5
is.na(x) #Output: FALSE FALSE TRUE FALSE FALSE
x[is.na(x)] #Output: 1 2 10 7
```

13. Frequency table

```
x <- sample(c("apple","orange","banana"), 10, replace = T)
table(x)
```

```
## x
## apple banana orange
## 5 3 2
```

```
y <- sample(c("red","yellow"), 10, replace = T)
table(x, y)
```

```
## y
## x red yellow
## apple 2 3
## banana 1 2
## orange 1 1
```

14. Factor

A factor is created using the factor() function.

```
education <- c('low', 'mid', 'high', 'high', 'mid')
educationf <- factor(education)

## [1] low  mid  high high mid
## Levels: high low mid
```

To simply print the levels of a factor the function levels() can be used. Factors can be useful to “filter out” typos or unacceptable values.

III. Matrix

1. Matrix Basics

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE,
        dimnames = NULL)
ncol() #shows the number of columns
nrow() #shows the number of rows
dim() #shows nrow first, ncol second
cbind() #add new vector of the same length of row
rbind() #add new vector of the same length of col
attributes() #shows the dimension
colnames() #add colnames
rownames() #add rownames
```

When adding two matrices that have the same dimension, they will add each other in the same position. Same for element-wise multiplication.

2. Matrix operations

```
t(A)%*%B # Matrix multiplication and transpose
sum(A) # add all elements
prod(A)
colSums(A) # sum columns
rowSums(A) # sum rows
```

3. Deal with NA

```
colSums(A, na.rm =T) # sum cols
rowSums(A, na.rm=T) # sum rows
colMeans(A, na.rm = T) # mean columns
rowMeans(A, na.rm = T) # mean rows
```

4. Using apply

```
apply(M, 1, sum, na.rm=T) # row sum without NA
apply(M, 2, sum, na.rm=T) # col sum without NA
```

5. Subset

Using the elements' position

```
M[n] # the n-th number in the matrix
M[,n] # the n-th row of the matrix
M[n,] # the n-th col of the matrix
M[n,m] # the n-th row, m-th col of the matrix
```

Using a logical vector

```
M[c(TRUE, FALSE, FALSE),c(TRUE, FALSE, FALSE)]
```

Using column names or row names

```
colnames(A) <- c("First","Second","Third")
rownames(A) <- c("a","b","C")
A["b","First"]
A[2,"First"]
```

6. Filtering

```
A <- matrix(100:105, ncol = 3, nrow = 2)
which(A==105) # return the index of element is 105
which(A==105,arr.ind=TRUE) # return 105 row and col
```

IV. List

1. List Basics

```
lst <- list(name="Fred", wife="Mary",
            no.children=3,
            child.ages=c(4,7,9), resident=TRUE)
str(lst)

## List of 5
## $ name      : chr "Fred"
## $ wife      : chr "Mary"
## $ no.children: num 3
## $ child.ages: num [1:3] 4 7 9
## $ resident  : logi TRUE

class(lst)

## [1] "list"
```

```
attributes(lst)

## $names
## [1] "name"      "wife"      "no.children" "child.ages" "resident"
```

2. Subset

```
class(lst[[1]])

## [1] "character"

class(lst[1])

## [1] "list"

lst[["name"]]

## [1] "Fred"

lst[["child.ages"]]

## [1] 4 7 9
```

3. Extract the first element of the fourth element of the list

```
lst[[4]][1]

## [1] 4
```

4. Select a sublist

```
lst[c(1,2)]

## $name
## [1] "Fred"
##
## $wife
## [1] "Mary"

lst[c("name","wife")]

## $name
## [1] "Fred"
##
## $wife
## [1] "Mary"
```

V. Arrays

1. One vector argument to describe all dimensions

```
A<-matrix(1:6,2,3)
B<-array(1:18,c(2,3,3))
C<-array(2,c(2,3,3))
dim(B)

## [1] 2 3 3
```

2. Subsetting an array is like subsetting a matrix but you will have more dimensions to specify

```
B[1,,]

##      [,1] [,2] [,3]
## [1,]   1   7  13
## [2,]   3   9  15
## [3,]   5  11  17
```

3. Select the last row of the third matrix

```
B[, ,3][2,]

## [1] 14 16 18
```

4. Select the first element of the third matrix

```
B[, ,3][1,1]

## [1] 13
```

VI. apply(), lapply(), sapply(), and tapply()

1. apply()

```
apply(X, MARGIN, FUN, ..., simplify = TRUE)
```

X: an array, including a matrix.
MARGIN: a vector giving the subscripts which the function will be applied over. E.g., for a matrix 1 indicates rows, 2 indicates columns, c(1, 2) indicates rows and columns. Where X has named dimnames, it can be a character vector selecting dimension names.

FUN: the function to be applied: see ‘Details’. In the case of functions like +, %*%, etc., the function name must be backquoted or quoted.

...: optional arguments to FUN.

simplify: a logical indicating whether results should be simplified if possible.

2. lapply()

```
lapply(X, FUN, ...)
```

lapply returns a list of the same length as X, each element of which is the result of applying FUN to the corresponding element of X.

```
lst <- list(group1 = sample(seq(0, 10, 0.1), 10),
            group2 = sample(seq(5, 10, 0.1), 10),
            group3 = sample(seq(10, 15, 0.1), 10))
lapply(lst, mean)
```

```
## $group1
## [1] 4.56
##
## $group2
## [1] 7.44
##
## $group3
## [1] 12.87
```

3. sapply()

```
sapply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)
```

USE.NAMES: logical; if TRUE and if X is character, use X as names for the result unless it had names already. Since this argument follows ... its name cannot be abbreviated.

sapply is a user-friendly version and wrapper of lapply by default returning a vector, matrix or, if simplify = "array", an array if appropriate, by applying simplify2array(). sapply(x, f, simplify = FALSE, USE.NAMES = FALSE) is the same as lapply(x, f).

```
sapply(lst, mean)
```

```
## group1 group2 group3
## 4.56 7.44 12.87
```

4. tapply()

```
tapply(X, INDEX, FUN = NULL, ..., default = NA, simplify = TRUE)
```

```
tapply(lst[[1]], lst[[2]], mean)
```

```
## 5.1 5.7 6.2 6.7 7.2 8 8.2 8.8 8.9 9.6
## 9.9 0.5 1.5 4.5 7.0 3.6 5.9 5.6 0.6 6.5
```

VII. Splitting according to a factor

```
str(ToothGrowth)
```

```
## 'data.frame': 60 obs. of 3 variables:
## $ len : num 4.2 11.5 7.3 5.8 6.4 10 11.2 11.2 5.2 7 ...
## $ supp: Factor w/ 2 levels "OJ","VC": 2 2 2 2 2 2 2 2 2 ...
## $ dose: num 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...
```

```
# split() creates a list, in this case it separates the vector len into subgroups
# based on the variable dose and stores the new split vectors into a list
split(ToothGrowth$len, ToothGrowth$dose)
```

```
## $'0.5'
## [1] 4.2 11.5 7.3 5.8 6.4 10.0 11.2 11.2 5.2 7.0 15.2 21.5 17.6 9.7 14.5
## [16] 10.0 8.2 9.4 16.5 9.7
##
## $'1'
## [1] 16.5 16.5 15.2 17.3 22.5 17.3 13.6 14.5 18.8 15.5 19.7 23.3 23.6 26.4 20.0
## [16] 25.2 25.8 21.2 14.5 27.3
##
## $'2'
## [1] 23.6 18.5 33.9 25.5 26.4 32.5 26.7 21.5 23.3 29.5 25.5 26.4 22.4 24.5 24.8
## [16] 30.9 26.4 27.3 29.4 23.0
```

```
class(split(ToothGrowth$len, ToothGrowth$dose))
```

```
## [1] "list"
```

The lapply() function applies a function to a list, returns a list in this case applies mean to len by dose

```
lapply(split(ToothGrowth$len, ToothGrowth$dose), mean)
```

```
## $'0.5'
## [1] 10.605
##
## $'1'
## [1] 19.735
##
## $'2'
## [1] 26.1
```

sapply() is a user friendly version of lapply(), it returns a vector

```
sapply(split(ToothGrowth$len, ToothGrowth$dose), mean)
```

```
## 0.5 1 2
## 10.605 19.735 26.100
```

tapply() apply a function to each cell of a ragged array, that is to each (non-empty) group of values given by a unique combination of the levels of certain factors.

```
tapply(ToothGrowth$len, ToothGrowth$dose, mean)
```

```
## 0.5 1 2
## 10.605 19.735 26.100
```

two way cross classification

```
sapply(split(ToothGrowth$len,
            list(ToothGrowth$supp, ToothGrowth$dose)), mean)
```

```
## OJ.0.5 VC.0.5 OJ.1 VC.1 OJ.2 VC.2
## 13.23 7.98 22.70 16.77 26.06 26.14
```

```
tapply(ToothGrowth$len,
      list(ToothGrowth$supp, ToothGrowth$dose), mean)
```

```
## 0.5 1 2
## OJ 13.23 22.70 26.06
## VC 7.98 16.77 26.14
```

VIII. Control structures: if, for, repeat, while

1. If statement

```
if (CON) {ACTION1} else {ACTION2}
if (CON) ACTION1 else ACTION2
ifelse(CON, ACTION1, ACTION2)
```

2. Nesting if statements

```
ifelse(CON1, ACTION1,
      ifelse(CON2, ACTION2,
            ACTION3))
```

3. For loop

```
for (variable in sequence){
  Do something
}
```

```
for (i in 1:4){
  j <- i + 10
  print(j)
}
```

4. While loop

```
while (condition){
  Do something
}
```

```
while (i < 5){
  print(i)
  i <- i + 1
}
```

5. Repeat

```
x <- 1
repeat
{
  print(x)
  x<-x+1
  if (x==10) break
}
```

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

IX. Function

```
function_name <- function(var){
  Do something
  return(new_variable)
}
```

```
square <- function(x){
  squared <- x*x
  return(squared)
}
```

Data structures: Vectors

- Manual creation: c(element1, element2, ...)
- Functions like seq(), rep(), rnorm(), rbinom(), sample(), etc, return a vector.
- Vectors contain elements of the same type (coercion to same type).
- Numeric vectors.
- Character vectors: insert string value using "" or "
- Logical vectors: F, T, or FALSE, TRUE
 - They are coerced to 0, 1 when used in functions like sum(), mean(), etc
- Factors: vectors with the 'levels' attribute: created using factor()
 - Levels can be sorted and labelled.
- Sub-setting vectors:
 - using []
 - position (also negative), logical vector, or name (in named vectors).
- Vectors recycling.

Data structures: matrices

- Creation from scratch: `matrix(x, ncol, nrow)`.
- Creation by combining vectors:
 - `cbind(vector1, vector2, ...)`
 - `rbind(vector1, vector2, ...)`
- All elements must be of the same type (e. g. all numeric, all character, etc).
- Sub-setting:
 - Using `[]` + Position of rows and columns (, indicates all), logical vector, or name (in named matrices)
- Adding names:
 - `colnames() <- c(name1, name2, ...)`
 - `rownames() <- c(name1, name2, ...)`

Data structures: Lists

- Creation from scratch: `list(element1, element2, ...)`
- Elements can be of the different type and/or different class.
- Adding elements:
 - `list_name[["new_element_name"]] <- new_element_values`
 - `list_name[[new_element_number]] <- new_element_values`
- Sub-setting:
 - Using `[]` + elements position or elements name or logical vector >> returns a list.
 - Using `[[ ]]` + element position or element name >> returns the element with its class.
 - Using `$` + element name >> returns the element with its class.

Data objects: Data frames

- Creation from scratch: `data.frame(col1, col2, ...)`.
- Columns are vectors that can be of the different type.
- Adding elements:
 - `df_name[["new_element_name"]] <- new_element_values`
 - `df_name$new_element_number <- new_element_values`
- Sub-setting:
 - `[]` + columns position (and rows position) >> may return a data frame or a vector
 - `[]` + logical vector as long as number of columns (or number of rows) >> returns a data frame
 - `[]` + name of columns (and/or rows) >> returns a data frame
 - `[[ ]]` + column position or column name >> returns a data frame
 - `$` + column name >> returns a vector

	Vector	Matrix	List	Dataframe	Tibble
Elements type	Same	Same	Different	Different	Different
Elements length	Same	Same	Different	Same	Same
Dimensions	Number of elements	Number of columns (elements) and rows.	Number of elements, length of each element	Number of columns (elements) and rows.	Number of columns (elements) and rows.
Sub-setting	<code>[]</code>	<code>[]</code>	<code>[]</code> <code>[[]]</code> <code>\$</code>	<code>[]</code> <code>[[]]</code> <code>\$</code>	<code>[]</code> <code>[[]]</code> <code>\$</code> Always returns tibbles
Printing method	Prints everything	Prints everything	Prints everything	Prints everything	Only prints the first 10 rows.
Recycling	-	Yes	-	Yes	No

General functions for objects

- `is` functions
- `as` functions
- `length()`
- `dim()`
- `class()`
- `str()`
- `attributes()`
- `names()`, `colnames()`, `rownames()`

Control flows: Repetitive execution

- `apply`
 - `apply(x, MARGIN, fun)`
 - X is a matrix
- `sapply()`
 - `sapply(x, fun)`
 - X is a list
 - Applies a function to all the elements of a list
- `lapply()`
 - `lapply(x, fun)`
 - X is a list
 - Applies a function to all the elements of a list
- `tapply()`
 - `tapply(vec1, factor, function)`
- Loops:
 - for loops


```
for (variable in vector) {
  commands to be repeated
}
```
 - while loop


```
while(condition) {
  commands to be repeated
}
```

Control flows: conditional execution

- Conditional execution: if statements
 - `if (condition) {`
 expression1
 }`else {`
 expression2
 }`}`
- Vectorized version:
 - `ifelse(condition, expression1, expression2)`

Advanced topics in Writing functions

- Name masking.
 - Search path
- Returning multiple values:
 - The `return()` function only takes one argument, thus can only return one object. If more objects need to be returned, they need to be embedded in a list
- Default values:
 - Arguments can be set equal to default values.
- The `...` argument:
 - For passing any admissible argument.
- The `stop()` function:
 - Stops functions execution and prints a message.
- The `switch()` function:
 - For conditional execution.

Data visualization with ggplot

- First `ggplot()` function.
- Then geom functions with aesthetics > to map variables from the data frame to the plot.

`ggplot(data = <DATA>) +`
`<GEOM FUNCTION>(mapping = aes(<MAPPINGS>))`

- Local vs global mappings.
- Plotting subsets of the dataset.
- Plotting more than two variables (colour, size, group, facets).
- Graphics for communication.

`df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))`
A special case of a list where all elements are the same length.



