BT1101

Basics of R Part I

Lab session contents

- Review lecture contents
- Discuss Part 1 of tutorial covering the previous week's lecture topic
- Hands-on coding in R

Basics of R

Setting up R & RStudio

Link to download R: https://www.r-project.org/

Link to download RStudio: https://www.rstudio.com/products/rstudio/

Data Types in R

A basic concept in programming is **variables.**

- Variables allow you to store information such as values (e.g. “2”) or objects (e.g. dataframes, functions) in R.
- Calling a variable’s name retrieves the stored information.
- Variable names are case-sensitive!
- Every variable has a data type (class):
  - Numeric
  - Integers
  - Logical
  - Character
  - Factor

Data Structures in R

1. **Vectors.** Can contain one datatype (e.g. numeric, character, logical), 1D.
    - y1 <- c(1, 2, 2, 3, 4, 5)
    - y2 <- c("small", "medium", "large", "large")

2. **Matrix.** Like vectors, can only contain one datatype (usually numeric). Data is arranged into a fixed number of rows and columns, 2D.
    - mat1 <- matrix(1:4, nrow=2, ncol=2)

3. **Arrays.** Multidimensional data structures. Matrices can be thought of as a special case of a 2D array.

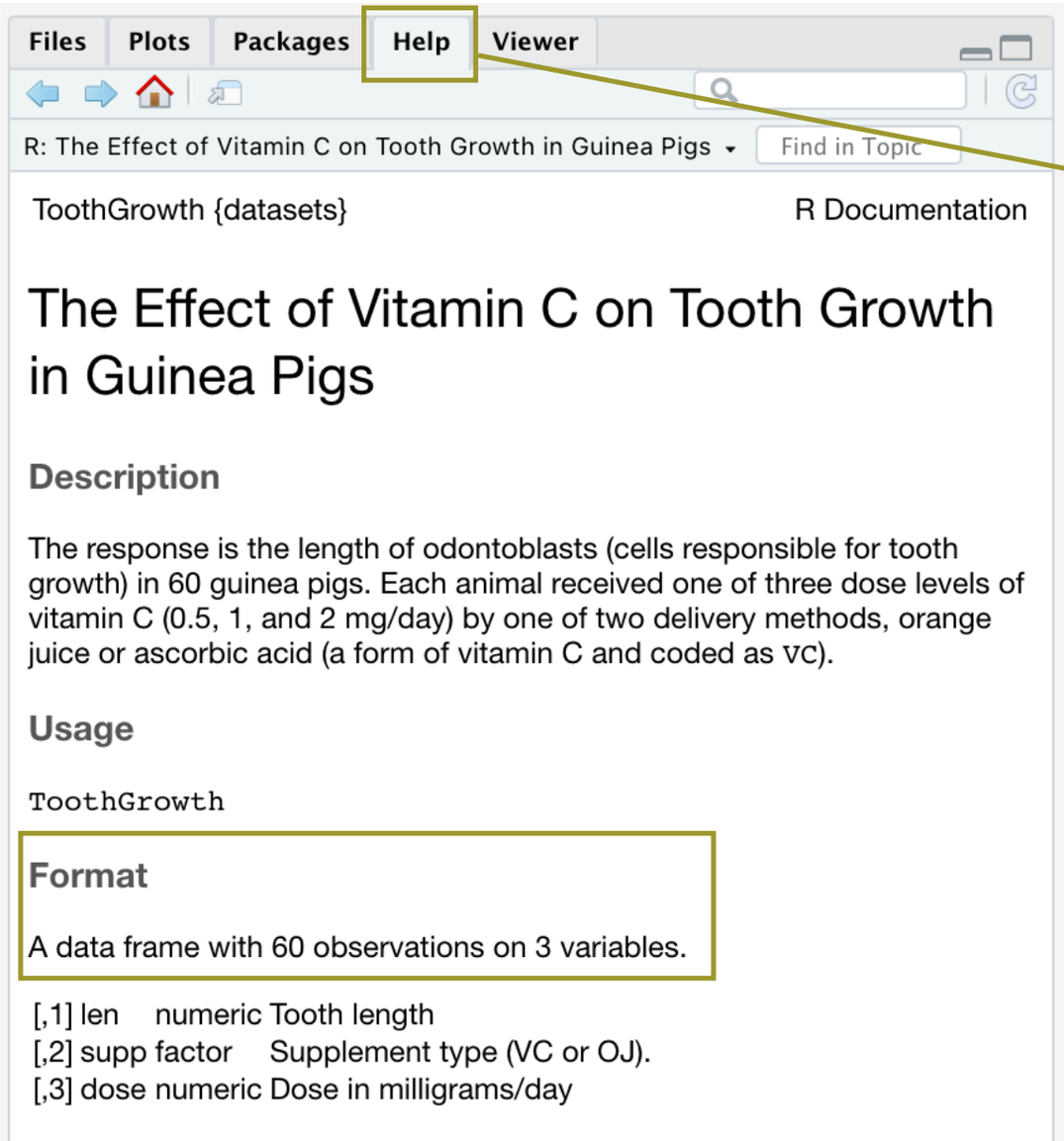
4. **Lists.** Can contain one or more datatypes.
    - list1 <- list(1, "two", y1, y2)

5. **Dataframes.** Tabular data.
    - data_frame <- data.frame(int_vec, char_vec, bool_vec)

Part 1

1) We will start by exploring the built-in dataset called ToothGrowth**. To find out more about this dataset, type** ?ToothGrowth **in the R command line.**

```
?ToothGrowth #to learn more about ToothGrowth in the help menu
```



Where is the help menu and what information can you retrieve from it?

Part 1

1) We will start by exploring the built-in dataset called ToothGrowth**. To find out more about this dataset, type** ?ToothGrowth **in the R command line.**

```
summary(ToothGrowth)
```

```
##       len        supp         dose      
##  Min.   : 4.20   OJ:30   Min.   :0.500  
##  1st Qu.:13.07   VC:30   1st Qu.:0.500  
##  Median :19.25           Median :1.000  
##  Mean   :18.81           Mean   :1.167  
##  3rd Qu.:25.27           3rd Qu.:2.000  
##  Max.   :33.90           Max.   :2.000  
```

```
str(ToothGrowth)
```

Structure

```
## '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 2 ...
##  $ dose: num  0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...
```

What is the difference between the output of summary versus str?

The str() in R is a built-in function that can show the internal structure of large lists that are nested. The str() method accepts R Object as an argument and returns the internal information about that object.

The str() method is used as an alternative to summary() function but the str() method is more compact than summary() method.

To display the internal data structure of an R object, use the str() function. The str() function returns information about the rows(observations) and columns(variables) along with extra information like the names of the columns, class of each column, followed by some of the initial observations of each of the columns.

Part 1

1) We will start by exploring the built-in dataset called ToothGrowth**. To find out more about this dataset, type** ?ToothGrowth **in the R command line.**

```
head(ToothGrowth)
```

```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

```
tail(ToothGrowth)
```

```
##     len supp dose
## 55 24.8   OJ    2
## 56 30.9   OJ    2
## 57 26.4   OJ    2
## 58 27.3   OJ    2
## 59 29.4   OJ    2
## 60 23.0   OJ    2
```

How about the difference between **head** and **tail**?

Files | Plots | Packages | Help | Viewer

R: Return the First or Last Parts of an Object — Find in Topic

head {utils} — R Documentation

Return the First or Last Parts of an Object

Description

Returns the first or last parts of a vector, matrix, table, data frame or function. Since `head()` and `tail()` are generic functions, they may also have been extended to other classes.

Usage

```
head(x, ...)
## Default S3 method:
head(x, n = 6L, ...)

## S3 method for class 'matrix'
head(x, n = 6L, ...) # is exported as head.matrix()
## NB: The methods for 'data.frame' and 'array'  are iden

## S3 method for class 'ftable'
head(x, n = 6L, ...)
## S3 method for class 'function'
head(x, n = 6L, ...)
```

Part 1

2) Selecting data

There are several variables in ToothGrowth**. Using Base R and dplyr functions, can you perform (i), (ii) and (iii)?**

i. **I. Extract the column** supp

ii. **II. Extract rows where** supp **is equal to "VC" and dose is less than 1 and assign the output to df2**

iii. **III. Extract the values of** len **where** supp **is equal to "VC"**

iv. **IV. Try to perform the above operations (i, ii, iii) again but this time, assign the output to** df2.1**,** df2.2 **and** df2.3 **respectively.**

v. **V. Use the class function to check the class attribute for each of the outputs. Use** is.data.frame **function to check whether the output is a dataframe or a vector.**

Indexing and selection with base R

Cheatsheet: https://github.com/rstudio/cheatsheets/blob/main/base-r.pdf

Selecting Vector Elements

By Position

`x[4]`	The fourth element.
`x[-4]`	All but the fourth.
`x[2:4]`	Elements two to four.
`x[-(2:4)]`	All elements except two to four.
`x[c(1, 5)]`	Elements one and five.

By Value

`x[x == 10]`	Elements which are equal to 10.
`x[x < 0]`	All elements less than zero.
`x[x %in% c(1, 2, 5)]`	Elements in the set 1, 2, 5.

Named Vectors

`x['apple']`	Element with name 'apple'.

Matrices

`m <- matrix(x, nrow = 3, ncol = 3)`
Create a matrix from x.

`m[2, ]` - Select a row

`m[ , 1]` - Select a column

`m[2, 3]` - Select an element

`t(m)`
Transpose

`m %*% n`
Matrix Multiplication

`solve(m, n)`
Find x in: m * x = n

Lists

`l <- list(x = 1:5, y = c('a', 'b'))`
A list is a collection of elements which can be of different types.

`l[[2]]`	`l[1]`	`l$x`	`l['y']`
Second element of l.	New list with only the first element.	Element named x.	New list with only element named y.

Also see the **dplyr** package.

Data Frames

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

x	y
1	a
2	b
3	c

List subsetting

`df$x`

`df[[2]]`

Understanding a data frame

`View(df)`	See the full data frame.
`head(df)`	See the first 6 rows.

Matrix subsetting

`df[ , 2]`

`df[2, ]`

`df[2, 2]`

`nrow(df)`
Number of rows.

`ncol(df)`
Number of columns.

`dim(df)`
Number of columns and rows.

cbind - Bind columns.

rbind - Bind rows.

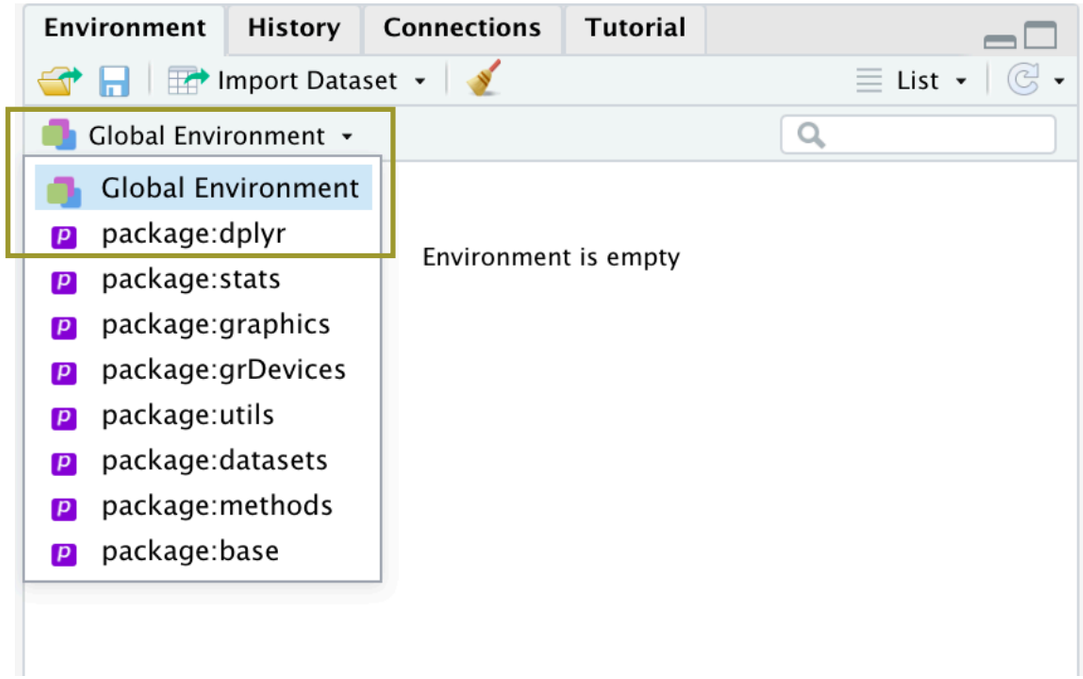
Hint: df [row, column]

dplyr Package

- Data manipulation library in R
- Lets you subset, reshape, join and summarize data typically using less code than would be required in base R
- Part of the R tidyverse
- **Install** the package (if not already installed), then **load** the dplyr library.
  - install.packages("tidyverse")
  - library(tidyverse)

Packages need to be loaded each time your environment is restarted. If successful, the package should be reflected in Global Environment.

However, packages only need to be **installed** ONCE. Comment out the installation code afterwards, or use the console to run the installation code rather than writing it in your R scripts — this might save you some errors later on.



dplyr Package

Documentation: https://cran.r-project.org/web/packages/dplyr/dplyr.pdf

Cheatsheet: https://github.com/rstudio/cheatsheets/blob/main/data-transformation.pdf

You are going to learn the five key `dplyr` functions that allow you to solve the vast majority of your data manipulation challenges:

- `filter`: pick observations based on values
- `arrange`: reorder data
- `select`: pick variables
- `mutate`: create new variables
- `summarise`: summarize data by functions of choice

dplyr Package

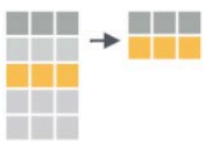
Documentation: https://cran.r-project.org/web/packages/dplyr/dplyr.pdf

Cheatsheet: https://github.com/rstudio/cheatsheets/blob/main/data-transformation.pdf

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.



filter(.data, …, .preserve = FALSE**)** Extract rows that meet logical criteria.
filter(mtcars, mpg > 20)

ARRANGE CASES

arrange(.data, …, .by_group = FALSE**)** Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

Summarise Cases

Apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …**)**
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

Manipulate Variables

EXTRACT VARIABLES

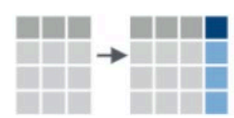
Column functions return a set of columns as a new vector or table.

select(.data, …**)** Extract columns as a table.
select(mtcars, mpg, wt)

MAKE NEW VARIABLES

Apply **vectorized functions** to columns. Vectorized functions take vectors as input and return vectors of the same length as output (see back).

vectorized function



mutate(.data, …, .keep = "all", .before = NULL, .after = NULL**)** Compute new column(s). Also **add_column(), add_count(),** and **add_tally().**
mutate(mtcars, gpm = 1 / mpg)

dplyr Package

- dplyr introduces **pipes**: %>% cmd+shift+M (Mac), ctrl+shift+M (Windows)
- Allows you to use the result of one function as the input to another function that comes after the pipe.
- Essentially, pipes allow you to chain several functions together

```
ToothGrowth %>% select(c(len, supp))
```

```
##     len supp
## 1   4.2   VC
## 2  11.5   VC
## 3   7.3   VC
## 4   5.8   VC
## 5   6.4   VC
## 6  10.0   VC
```

Part 1

2i) Extract the column supp.

```
#i extracting variable values or columns
ToothGrowth$supp
```

```
##  [1] VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC
## [26] VC VC VC VC VC OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ
## [51] OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ
## Levels: OJ VC
```

```
ToothGrowth[,2]
```

```
##  [1] VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC
## [26] VC VC VC VC VC OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ
## [51] OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ
## Levels: OJ VC
```

```
ToothGrowth[,"supp"]
```

```
##  [1] VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC
## [26] VC VC VC VC VC OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ
## [51] OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ
## Levels: OJ VC
```

Part 1

2i) Extract the column supp.

```
ToothGrowth %>% select(supp)  #using dplyr
```

```
##    supp
## 1    VC
## 2    VC
## 3    VC
## 4    VC
## 5    VC
## 6    VC
## 7    VC
```

What is the dplyr %>% doing here?

Part 1

2ii) Extract rows where supp is equal to "VC" and dose is less than 1 and assign the output to df2

```
#ii extracting rows
df2<-ToothGrowth[ToothGrowth$supp=="VC" & ToothGrowth$dose<1,]
df2<- subset(ToothGrowth, supp=="VC" & dose <1)
df2<- ToothGrowth %>% filter(supp=="VC" & dose <1)  #using dplyr
```

```
    len supp dose
1   4.2   VC  0.5
2  11.5   VC  0.5
3   7.3   VC  0.5
4   5.8   VC  0.5
5   6.4   VC  0.5
6  10.0   VC  0.5
7  11.2   VC  0.5
8  11.2   VC  0.5
9   5.2   VC  0.5
10  7.0   VC  0.5
```

Part 1

2iii) Extract the values of len where supp is equal to "VC"

```
ToothGrowth$len[ToothGrowth$supp=="VC"]
```

```
##  [1]  4.2 11.5  7.3  5.8  6.4 10.0 11.2 11.2  5.2  7.0 16.5 16.5 15.2 17.3 22.5
## [16] 17.3 13.6 14.5 18.8 15.5 23.6 18.5 33.9 25.5 26.4 32.5 26.7 21.5 23.3 29.5
```

```
subset(ToothGrowth, select = "len", supp=="VC")
```

```
##     len
## 1   4.2
## 2  11.5
## 3   7.3
## 4   5.8
## 5   6.4
## 6  10.0
## 7  11.2
## 8  11.2
## 9   5.2
```

```
ToothGrowth %>% filter(supp=="VC") %>% select(len) #using dplyr
```

```
##     len
## 1   4.2
## 2  11.5
## 3   7.3
## 4   5.8
## 5   6.4
## 6  10.0
## 7  11.2
## 8  11.2
```

Part 1

2iv) Try to perform the above operations (i, ii, iii) again but this time, assign the output to df2.1, df2.2 and df2.3 respectively.

2v) Use the class function to check the class attribute for each of the ouputs. Use is.data.frame function to check whether the output is a dataframe or a vector.

```
#iv
df2.1 <- ToothGrowth$len[ToothGrowth$supp=="VC"]
df2.2 <- subset(ToothGrowth, select = "len", supp=="VC")
df2.3 <- ToothGrowth %>% filter(supp=="VC") %>% select(len) #using dplyr
class(df2.1)
```

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

```
class(df2.2)
```

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

```
class(df2.3)
```

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

Why do you need to assign the output to a label/name?

Part 1

```
is.data.frame(df2.1)
```

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

```
is.data.frame(df2.2)
```

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

```
is.data.frame(df2.3)
```

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

Why do you need to assign the output to a label/name?

Part 1

3) Adding/Removing/Changing data columns for Toothgrowth data.

i. Change the variable name from len to length and assign the output to df3.1

ii. Increase the value of len by 0.5 if supp is equal to OJ and assign the output to df3.2

iii. Remove the column dose from the data and assign the output to df3.3

iv. Increase the value of dose by 0.1 for all records and rename dose to dose.new and assign output to df3.4

v. Create a new variable high.dose and assign it a value of "TRUE" if dose is more than 1 and "FALSE" if dose is less than or equal to 1. Assign the dataframe with the new variable high.dose to df3.5. Export df3.5 to a csv file. Discuss what is the r code to export as an excel file (.xlsx).

Part 1

3i) Change the variable name from len to length and assign the output to df3.1

```
df3.1 <- ToothGrowth
colnames(df3.1)[colnames(df3.1) == "len"] <- "length"  # base R
```

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

With the base R option, the results are stored.

```
# i)
df3.1 <- ToothGrowth %>% rename (length=len) # using dplyr
df3.1
```

```
##    length supp dose
## 1     4.2   VC  0.5
## 2    11.5   VC  0.5
## 3     7.3   VC  0.5
## 4     5.8   VC  0.5
## 5     6.4   VC  0.5
## 6    10.0   VC  0.5
## 7    11.2   VC  0.5
## 8    11.2   VC  0.5
```

Recommended

Part 1

3ii) Increase the value of len by 0.5 if supp is equal to OJ and assign the output to df3.2

```
# ii)
# to retain all observations
df3.2 <- ToothGrowth
df3.2$len[df3.2[,"supp"] == "OJ"] <- df3.2$len[df3.2[,"supp"] == "OJ"] + 0.5

# using dplyrs with ifelse or case_when

df3.2 %>%
    mutate(len = ifelse(supp == "OJ", len+0.5, len))
```

Use dplyr please, dont use base R

```
##     len supp dose
## 1   4.2   VC  0.5
## 2  11.5   VC  0.5
## 3   7.3   VC  0.5
## 4   5.8   VC  0.5
## 5   6.4   VC  0.5
## 6  10.0   VC  0.5
## 7  11.2   VC  0.5
## 8  11.2   VC  0.5
## 9   5.2   VC  0.5
```

```
df3.2 %>%
    mutate(len = case_when(supp == "OJ" ~ len+0.5, TRUE ~ len))
```

Part 1

3iii) Remove the column dose from the data and assign the output to df3.3

```
# iii)
df3.3 <- ToothGrowth[, -3]
df3.3 <- subset(ToothGrowth, select = -c(dose))
df3.3
```

```
##     len supp
## 1   4.2   VC
## 2  11.5   VC
## 3   7.3   VC
## 4   5.8   VC
## 5   6.4   VC
## 6  10.0   VC
## 7  11.2   VC
## 8  11.2   VC
## 9   5.2   VC
## 10  7.0   VC
```

```
#dplyr
df3.3 <- ToothGrowth %>% select(-c(dose))
df3.3
```

```
##     len supp
## 1   4.2   VC
## 2  11.5   VC
## 3   7.3   VC
## 4   5.8   VC
## 5   6.4   VC
## 6  10.0   VC
## 7  11.2   VC
## 8  11.2   VC
## 9   5.2   VC
## 10  7.0   VC
## 11 16.5   VC
```

Part 1

3iv) Increase the value of dose by 0.1 for all records and rename dose to dose.new and assign output to df3.4

```
# iv Increase the value of `dose` by 0.1 for all records and rename `dose` to `dose.new` and assign output to df3.4

df3.4 <- ToothGrowth
df3.4$dose.new <- df3.4$dose + 0.1
df3.4 <- df3.4[, -3] # or use subset

df3.4
```

```
##     len supp dose.new
## 1   4.2   VC      0.6
## 2  11.5   VC      0.6
## 3   7.3   VC      0.6
## 4   5.8   VC      0.6
## 5   6.4   VC      0.6
## 6  10.0   VC      0.6
## 7  11.2   VC      0.6
## 8  11.2   VC      0.6
## 9   5.2   VC      0.6
## 10  7.0   VC      0.6
## 11 16.5   VC      1.1
## 12 16.5   VC      1.1
## 13 15.2   VC      1.1
## 14 17.3   VC      1.1
## 15 22.5   VC      1.1
## 16 17.3   VC      1.1
## 17 13.6   VC      1.1
```

```
# using dplyr
df3.4 <- ToothGrowth %>% mutate(dose=dose+0.10) %>% rename (new.dose=dose)
df3.4
```

*dose.new

Part 1

3v) Create a new variable high.dose and assign it a value of "TRUE" if dose is more than 1 and "FALSE" if dose is less than or equal to 1. Assign the dataframe with the new variable high.dose to df3.5. Export df3.5 to a csv file. Discuss what is the r code to export as an excel file (.xlsx).

```
# v
df3.5 <- ToothGrowth
df3.5$high.dose[df3.5[,"dose"] > 1 ] <- "TRUE"
df3.5$high.dose[df3.5[,"dose"] <= 1 ] <- "FALSE"
df3.5
```

```
##     len supp dose high.dose
## 1   4.2   VC  0.5     FALSE
## 2  11.5   VC  0.5     FALSE
## 3   7.3   VC  0.5     FALSE
## 4   5.8   VC  0.5     FALSE
## 5   6.4   VC  0.5     FALSE
## 6  10.0   VC  0.5     FALSE
## 7  11.2   VC  0.5     FALSE
## 8  11.2   VC  0.5     FALSE
## 9   5.2   VC  0.5     FALSE
## 10  7.0   VC  0.5     FALSE
```

```
write.csv(df3.5, "df3.5.csv") # write.xlsx(df3.5, "df3.5.xlsx")
```

Food for thought:

We can use dplyr too! How would you do it?

Part 1

4) Sorting

i. **There are two functions in Base R "sort" and "order" to perform sorting. How do these two functions differ? Try to do a sort with each function on ToothGrowth$len.**

ii. **Using a base R function (e.g. order), how can you sort the dataframe ToothGrowth in decreasing order of len?**

iii. **What dplyr functions can you use to sort ToothGrowth in increasing order of len? Can you also sort the dataframe in decreasing order of len?**

Sorting

Base R

- **sort** returns the original object, sorted in ascending order by default.
- **order** returns the indices of the sorted object, also in ascending order by default.

```
myvector <- c(10,30,15,40)
sort(myvector)
```

```
## [1] 10 15 30 40
```

```
order(myvector)
```

```
## [1] 1 3 2 4
```

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, …, .preserve = FALSE**)** Extract rows that meet logical criteria.
filter(mtcars, mpg > 20)

ARRANGE CASES

arrange(.data, …, .by_group = FALSE**)** Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

In dplyr, **arrange()** orders the rows of a data frame by the values of selected columns in ascending order by default.

Part 1

4i) There are two functions in Base R "sort" and "order" to perform sorting. How do these two functions differ? Try to do a sort with each function on ToothGrowth$len.

```
# i)
sort(ToothGrowth$len)
```

```
##  [1]  4.2  5.2  5.8  6.4  7.0  7.3  8.2  9.4  9.7  9.7 10.0 10.0 11.2 11.2 11.5
## [16] 13.6 14.5 14.5 14.5 15.2 15.2 15.5 16.5 16.5 16.5 17.3 17.3 17.6 18.5 18.8
## [31] 19.7 20.0 21.2 21.5 21.5 22.4 22.5 23.0 23.3 23.3 23.6 23.6 24.5 24.8 25.2
## [46] 25.5 25.5 25.8 26.4 26.4 26.4 26.4 26.7 27.3 27.3 29.4 29.5 30.9 32.5 33.9
```

```
order(ToothGrowth$len)
```

```
##  [1]  1  9  4  5 10  3 37 38 34 40  6 36  7  8  2 17 18 35 49 13 31 20 11 12 39
## [26] 14 16 33 22 19 41 45 48 28 32 53 15 60 29 42 21 43 54 55 46 24 51 47 25 44
## [51] 52 57 27 50 58 59 30 56 26 23
```

Hint: How sorting can be done using Base R and dplyr

Part 1

4ii) Using a base R function (e.g. order), how can you sort the dataframe ToothGrowth in decreasing order of len?

```
# ii)
ToothGrowth[order(ToothGrowth$len, decreasing=TRUE),]
```

```
##     len supp dose
## 23 33.9   VC  2.0
## 26 32.5   VC  2.0
## 56 30.9   OJ  2.0
## 30 29.5   VC  2.0
## 59 29.4   OJ  2.0
## 50 27.3   OJ  1.0
```

Part 1

4iii) What dplyr function can you use to sort ToothGrowth in increasing order of len? Can you also sort the dataframe in decreasing order of len?

```
#iii)
ToothGrowth %>% arrange(len)
```

```
##     len supp dose
## 1   4.2   VC  0.5
## 2   5.2   VC  0.5
## 3   5.8   VC  0.5
## 4   6.4   VC  0.5
## 5   7.0   VC  0.5
## 6   7.3   VC  0.5
```

```
ToothGrowth %>% arrange(desc(len))
```

```
##     len supp dose
## 1  33.9   VC  2.0
## 2  32.5   VC  2.0
## 3  30.9   OJ  2.0
## 4  29.5   VC  2.0
## 5  29.4   OJ  2.0
## 6  27.3   OJ  1.0
```

Part 1

5) Factors

i. **Check if** supp **is a factor vector. First type** ToothGrowth$supp**. What do you observe with the output?**

ii. **Next use** is.factor() **and** is.ordered() **to check if** supp **is a factor and is so whether it is an ordered factor.**

iii. **Now supposed we find that vitamin C (VC) is a superior supplement compared to orange juice (OJ), and we want to order** supp **such that VC is a higher level than OJ, how could we do this?**

3

Part 1

5i) Check if supp is a factor vector. First type ToothGrowth$supp. What do you observe with the output?

```
#i
ToothGrowth$supp
```

```
##  [1] VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC
## [26] VC VC VC VC VC OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ
## [51] OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ
## Levels: OJ VC
```

What are factors and their levels? Why do you think factors might be helpful when performing data analysis?

Factor is a data structure used for fields that take only predefined, finite number of values (CATEGORICAL data)

Levels is the different type of values in the field.

Eg. Marital status is a factor that may contain values from single, married, separated, divorced, or widowed.

single, married, separated, divorced, or widowed are levels for the factor.

Part 1

5ii) Next use is.factor() and is.ordered() to check if supp is a factor and is so whether it is an ordered factor.

```
#ii
is.factor(ToothGrowth$supp)
```

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

```
is.ordered(ToothGrowth$supp)
```

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

Example of an ordered factor
Big Medium Small

Part 1

5ii) Now supposed we find that vitamin C (VC) is a superior supplement compared to orange juice (OJ), and we want to order supp such that VC is a higher level than OJ, how could we do this? (Hint: Assign factor_supp to ToothGrowth$supp)

```
#iii
factor_supp <- factor(ToothGrowth$supp, levels=c("OJ","VC"),ordered=TRUE )
factor_supp
```

```
##  [1] VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC
## [26] VC VC VC VC VC OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ
## [51] OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ
## Levels: OJ < VC
```

```
is.factor(factor_supp)
```

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

```
is.ordered(factor_supp)
```

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

R & RStudio tips

Clearing your workspace can prevent it from becoming messy.

- To clear the console: **ctrl + L**
- To clear a variable from your environment: **rm(variable)**
- To clear all variables (use with caution!): **rm(list=ls())** Recommend not using this TBH

Keyboard shortcuts help you work more efficiently. Some common ones are:

- Run the current line of code: **cmd+enter** (Mac), **ctrl+enter** (Windows)
- Insert the <- operator: **option + -** (Mac), **Alt + -** (Windows)
- Insert the %>% operator: **cmd+shift+M** (Mac), **ctrl+shift+M** (Windows)

ctrl/cmd + S to save your file

Documentation provides information about functions in R and examples of how they're used.

- To read R documentation about a function: **help(function_name)** or **?function_name**

Next week

- Lecture and tutorials as per usual next week
- Basics of R Part 2 due **5th Sept (Mon), 9am** — answers **must** be submitted in the form of an R script.