

R Workshop

Lecture 2: Basics of R (continued) and tidyverse functions

1 Quick review

printing, vectors, `data.frames`, conditionals, Discuss use of `paste()` and `assign()` to create new variable names in loops ([see here for more](#)).

2 Working with pre-collected data using tidyverse

The functions below are part of the `dplyr` library, which is loaded when you load `tidyverse`.

2.1 The right packages/libraries need to be installed and used

Packages/libraries are repositories of other functions that might be useful for us. There are literally thousands of such libraries for R. We are going to use a collection of packages called the `tidyverse` - when we install and load this library, we will be able to use the many libraries and functions developed or inspired by the work of **Hadley Wickham**.

The following command installs packages/libraries. [**Note:** the quotes are necessary.]

```
install.packages("tidyverse")
```

It is not enough to install a library in R; you also have to “load” it in every session you want to use it. When you run the command, you will get a set of lines that look like the following, don’t worry, it’s just loading the relevant libraries associated with `tidyverse`. [**Note:** No quotes here.]

```
library(tidyverse)

## -- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
## v dplyr      1.1.4      v readr      2.1.5
## v forcats   1.0.0      v stringr    1.5.1
## v ggplot2   3.5.2      v tibble     3.3.0
## v lubridate 1.9.4      v tidyr      1.3.1
## v purrr     1.0.4
## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

Advice: Put all the `install.packages` and `library` commands at the beginning of your script. This way, you will know what all libraries are used in your script.

2.2 Reading in data

Advice: You should try to store your data in a `csv` file (comma separated values), if possible. The file format is easy to work with it, and can really be opened by a variety of different programs.

To open a `csv` file that is in a particular directory, you will need to “set the working directory” with the function `setwd()`, and then use the `read_csv()` file to open the data. If you do this correctly, you will see a `data.frame` names `measurements` in your environment list.

Note: There are other functions like `read.csv()` that can be used too. I am just teaching you one of them.

```
#setting the working directory
#Replace content with the relevant directory address
setwd("/.../.../.../")

#Opening the csv file and storing it to a (data.frame) variable
#The variable can be called anything, I am just calling it 'measurements'
measurements = read_csv("Lecture2-Measurements.csv")
```

```
## Rows: 36 Columns: 5
## -- Column specification -----
## Delimiter: ","
## chr (2): Vowel, Speed
## dbl (3): Subject, Consonant1Duration, Consonant2Duration
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Above, I encouraged you to use the `csv` file type, but every now-and-then, it is easier to work with other file types (`xlsx`, `spss data`, ...). In those cases, there are other libraries or functions that you can use. If you want to open an `xlsx` file, then you need to use the following function:

```
#Loading the relevant library (Note: this is installed with the tidyverse,
#but still has to be loaded separately.)
library(readxl)

#Setting the working directory
setwd("/.../.../.../")

#Opening the first sheet in an excel file and storing it to a (data.frame) variable
measurements = read_excel("Measurements.xlsx",sheet=1)
```

```
#Loading the relevant library (Note: this is installed with the tidyverse,
#but still has to be loaded separately.)
library(readxl)

#Opening the first sheet in an excel file and storing it to a (data.frame) variable
measurements = read_excel("Measurements.xlsx",sheet=1)
```

2.3 Getting information about your `data.frame`

There are many ways you can view or get information about your `data.frame`

```
#Viewing only the first "n" rows of your data
head(measurements,10)
```

```

## # A tibble: 10 x 5
##   Subject Vowel Speed Consonant1Duration Consonant2Duration
##   <dbl> <chr> <chr>           <dbl>           <dbl>
## 1     1     1 ae FAST             60.0             52.4
## 2     1     1 ae NORMAL         65.6             64.0
## 3     1     1 ae SLOW           64.6             56.2
## 4     1     1 o  FAST             69.8             62.2
## 5     1     1 o  NORMAL         74.0             69.8
## 6     1     1 o  SLOW           76.0             69.4
## 7     3     3 ae FAST             70.2             62.2
## 8     3     3 ae NORMAL         70.6             58.2
## 9     3     3 ae SLOW           65.3             59.5
## 10    3     3 o  FAST             65.7             67.3

#The default is 6 rows
head(measurements)

## # A tibble: 6 x 5
##   Subject Vowel Speed Consonant1Duration Consonant2Duration
##   <dbl> <chr> <chr>           <dbl>           <dbl>
## 1     1     1 ae FAST             60.0             52.4
## 2     1     1 ae NORMAL         65.6             64.0
## 3     1     1 ae SLOW           64.6             56.2
## 4     1     1 o  FAST             69.8             62.2
## 5     1     1 o  NORMAL         74.0             69.8
## 6     1     1 o  SLOW           76.0             69.4

#If you want to see some general information about each column
summary(measurements)

##   Subject      Vowel      Speed      Consonant1Duration
## Min.   :1  Length:36  Length:36  Min.   :56.52
## 1st Qu.:3  Class :character  Class :character  1st Qu.:63.07
## Median :5  Mode  :character  Mode  :character  Median :65.68
## Mean   :5
## 3rd Qu.:7
## Max.   :9
##                               Mean   :66.34
##                               3rd Qu.:69.92
##                               Max.   :77.08
## Consonant2Duration
## Min.   :52.39
## 1st Qu.:58.94
## Median :62.09
## Mean   :62.27
## 3rd Qu.:66.83
## Max.   :73.50

```

If you want to view **all** the data in a separate tab in **RStudio**, then use `View(name of data.frame)`.

2.4 Subsetting to only some rows of your data.frame

Sometimes, you want to remove some of the data because it is not relevant to the analysis you are performing. Let's say that in our dataset, I want to get only the "SLOW" values. Then, we do the following. First, we select the relevant `data.frame`, and then **pipe/chain** a `filter()` function to it with the piping function `%>%`.

```
#Selects only the "SLOW" values
measurements2 = measurements %>%
  filter(Speed == "SLOW")
```

If you want everything but the "SLOW" values, then:

```
#Selects only the "SLOW" values
measurements2 = measurements %>%
  filter(Speed != "SLOW")
```

2.5 Selecting only some columns of your data.frame

Let's say you have a gigantic `data.frame`, with lots of columns, but you are interested only in some columns, then it makes sense to remove everything else for current purposes. It would be a terrible idea to delete it from the original `csv` file, as we might lose the data forever; it is better to do it in **R**, so that the elimination is just temporary.

```
#Selects only the relevant columns
measurements2 = measurements %>%
  select(Subject, Vowel, Speed, Consonant1Duration)
```

If you want to both filter rows and select columns, then use the pipe twice:

```
#Filter and then select only the relevant columns
measurements2 = measurements %>%
  filter(Speed != "SLOW") %>%
  select(Subject, Vowel, Speed, Consonant1Duration)

#If all the selected columns are adjacent to one another,
#you can use the ":" notation
measurements2 = measurements %>%
  filter(Speed != "SLOW") %>%
  select(Subject:Consonant1Duration)
```

2.6 Arranging your data in descending or ascending order according to some column of your data.frame

To view the data, sometimes it makes sense to arrange it in descending or ascending order according to some column(s) in your `data.frame`. In which case, we can use `arrange`.

```
#Arranges in ascending order according to the column "Speed"
measurements3 = measurements2 %>%
  arrange(Speed)
head(measurements3)

## # A tibble: 6 x 4
##   Subject Vowel Speed Consonant1Duration
##   <dbl> <chr> <chr>         <dbl>
## 1     1   ae   FAST           60.0
## 2     1   o    FAST           69.8
## 3     3   ae   FAST           70.2
```

```

## 4      3 o      FAST      65.7
## 5      4 ae     FAST      64.2
## 6      4 o      FAST      66.7

#Arranges in descending order according to the column "Speed"
measurements3 = measurements2 %>%
  arrange(desc(Speed))
head(measurements3)

## # A tibble: 6 x 4
##   Subject Vowel Speed  Consonant1Duration
##   <dbl> <chr> <chr>          <dbl>
## 1      1 ae   NORMAL          65.6
## 2      1 o    NORMAL          74.0
## 3      3 ae   NORMAL          70.6
## 4      3 o    NORMAL          66.6
## 5      4 ae   NORMAL          67.3
## 6      4 o    NORMAL          77.1

#You can do more complex arrangements
#descending (alphabetic) order for "Speed", and then ascending order for "Consonant1Duration"
#Note: the order matters
measurements3 = measurements2 %>%
  arrange(desc(Speed), Consonant1Duration)
head(measurements3)

## # A tibble: 6 x 4
##   Subject Vowel Speed  Consonant1Duration
##   <dbl> <chr> <chr>          <dbl>
## 1      9 ae   NORMAL          56.5
## 2      9 o    NORMAL          57.7
## 3      7 o    NORMAL          61.5
## 4      7 ae   NORMAL          63.6
## 5      6 ae   NORMAL          64.4
## 6      1 ae   NORMAL          65.6

```

2.7 Creating a new column in your data.frame

Sometimes, it is useful to create a new column in your `data.frame()`: maybe you want to create some new column, or you want to keep track of some information in the data.

```

#Creates a new column with the same value
measurements3 = measurements2 %>%
  mutate(NewValue = 1)
head(measurements3)

## # A tibble: 6 x 5
##   Subject Vowel Speed  Consonant1Duration NewValue
##   <dbl> <chr> <chr>          <dbl>     <dbl>
## 1      1 ae   FAST          60.0         1
## 2      1 ae   NORMAL        65.6         1
## 3      1 o    FAST          69.8         1
## 4      1 o    NORMAL        74.0         1
## 5      3 ae   FAST          70.2         1

```

```
## 6      3 ae    NORMAL      70.6      1

#Creates a new column where the value depends on another column.
#In this case, using the function "ifelse()" inside mutate is super useful in the long run.
measurements3 = measurements2 %>%
  mutate(NewValue = ifelse(Consonant1Duration<65, yes="Low", no="High"))
head(measurements3)

## # A tibble: 6 x 5
##   Subject Vowel Speed  Consonant1Duration NewValue
##   <dbl> <chr> <chr>                <dbl> <chr>
## 1     1   ae    FAST                 60.0 Low
## 2     1   ae    NORMAL               65.6 High
## 3     1   o    FAST                 69.8 High
## 4     1   o    NORMAL               74.0 High
## 5     3   ae    FAST                 70.2 High
## 6     3   ae    NORMAL               70.6 High
```

2.8 Summarising your data.frame

This is extremely useful, if you want to get average values for participants or some combination of column values. It requires the use of two functions `group_by()` and `summarise()/summarize()`.

```
#Summarising the data with the mean value for each participant
measurements3 = measurements2 %>%
  group_by(Subject) %>%
  summarise(MeanSubjectValue = mean(Consonant1Duration))
head(measurements3)

## # A tibble: 6 x 2
##   Subject MeanSubjectValue
##   <dbl>          <dbl>
## 1     1             67.4
## 2     3             68.3
## 3     4             68.8
## 4     6             69.9
## 5     7             61.1
## 6     9             59.3

#Summarising the data with the mean value for each participant for each speed
measurements3 = measurements2 %>%
  group_by(Subject,Speed) %>%
  summarise(MeanSubjectValue = mean(Consonant1Duration))

## `summarise()` has grouped output by 'Subject'. You can override using the
## `groups` argument.

head(measurements3)

## # A tibble: 6 x 3
## # Groups:   Subject [3]
##   Subject Speed  MeanSubjectValue
##   <dbl> <chr>          <dbl>
## 1     1 FAST             64.9
```

```
## 2      1 NORMAL      69.8
## 3      3 FAST       68.0
## 4      3 NORMAL     68.6
## 5      4 FAST       65.4
## 6      4 NORMAL     72.2
```

3 Some useful references

You should keep a copy of the [dplyr cheatsheet](#) with you — it’s super helpful!

4 Homework

1. Find the error in each of the following pieces of code:

(a) When trying to print “Hello World!” 6 times:

```
for(i within c(1:6)){
  print("Hello World!")
}
```

(b) When trying to exclude subjects numbered 10 or above from the *Subject* column from a `data.frame` named *ExperimentData*:

```
ExperimentData %>
  filter(Subject < 10)
```

(c) When trying to include just some specific values from a column named *Subject* column from a `data.frame` named *ExperimentData*:

```
ExperimentData %>%
  filter(Subject = 1)
```

(d) This is not quite an ‘error’, but it is still not what you should be doing. When trying to get the mean value for each *Subject* of a column named *ReactionTime* from a `data.frame` named *ExperimentData*:

```
ExperimentData %>%
  group_by(Subject) %>%
  mutate(MeanReactionTime = mean(ReactionTime))
```

2. Open the `data.frame` that we used in class (uploaded to D2L), and then exclude the *Consonant1Duration* column from the `data.frame`. Note, write the code using `tidyverse` functions and the piping function.

3. Write a script using `tidyverse` functions that does the following things. Make sure to comment your script properly.

i. Create a data frame named *Data* with 3 columns and a 100 rows

- 1st column named *Subject*: the values 1 to 100.
- 2nd column named *Position*: Repeat two items “A” and “B”, so that all even subjects are A, and all odd subjects are B. So toggle between the two.
- 3rd column named *Values*: the values 1001 to 1100 in that order.

ii. Use `tidyverse` functions to subset to only the rows that have *Position* as “A”.

- iii. Now, use `tidyverse` functions to create a new column named *NewValues* with the values 2001 to 2050.
- iv. Now, use `tidyverse` functions to find the mean of all the remaining values in the column.
- v. Combine the above steps ii-iv into one long chain of commands with the piping function, and assign it to a new data frame named *Data2*.