GSP Workshop: Intermediate R: Statistics for Graduate Students

Nnenna Asidianya

University of Toronto

March 2, 2021

Outline

Part I

- Conditionals and Control Work flows
  - Equality (or not)
  - & and |
  - The if statement

Outline

Part I

- Conditionals and Control Work flows
  - Equality (or not)
  - & and |
  - The if statement
- Loops
  - Write a while loop
  - Write a for loop
  - Mix up loops with control flow

Outline

Part I

- Conditionals and Control Work flows
  - Equality (or not)
  - & and |
  - The if statement
- Loops
  - Write a while loop
  - Write a for loop
  - Mix up loops with control flow

Part II

- Functions and Arguments: Review

Outline

Part I

- Conditionals and Control Work flows
  - Equality (or not)
  - & and |
  - The if statement
- Loops
  - Write a while loop
  - Write a for loop
  - Mix up loops with control flow

Part II

- Functions and Arguments: Review
- Subset functions in R
  - Demonstration with data set
  - Resources for developing packages for tidying data

Outline

Part I

- Conditionals and Control Work flows
  - Equality (or not)
  - & and |
  - The if statement
- Loops
  - Write a while loop
  - Write a for loop
  - Mix up loops with control flow

Link to slides: Intermediate-R

Part II

- Functions and Arguments: Review
- Subset functions in R
  - Demonstration with data set
  - Resources for developing packages for tidying data
- Advanced: Tidyverse

Prelude: R Markdown

R Markdown provides an patform for data science. In an R Markdown file you can both

- Both save and execute code
- create reports and/or articles that can be shared to various audiences.

First:

```
```{r}
install.packages("rmarkdown")
```
```

Here is a link to the code used : R Code
Here is a link to an R Markdown cheet sheet: R Markdown

Conditionals and Control Flow

A **logical statement** is a declarative sentence which conveys information about the truth of a statement.

Figure: Figure credit: Boolean

Conditionals and Control Flow

A **logical statement** is a declarative sentence which conveys information about the truth of a statement.

- Examples 1 and 2: Equality (or not): If $x = 3, y = 4$ then we can assess the truth about the following statements:

```
> #Example 1: Equality
>
> x=3
> y=4
> x==y
[1] FALSE
```

Conditionals and Control Flow

A **logical statement** is a declarative sentence which conveys information about the truth of a statement.

- Examples 1 and 2: Equality (or not): If $x = 3, y = 4$ then we can assess the truth about the following statements:

```
> #Example 1: Equality
>
> x=3
> y=4
> x==y
[1] FALSE
```

```
> #Example 2: Not equality
>
> x!=y
[1] TRUE
>
```

Conditionals and Control Flow

- Example 3: Equality (or not): Comparison of strings.
  - Set the two vectors below equal to each other
  - This creates a logical vector (with TRUE/FALSE)

```
user1<-c("userR", "user", "UserR")
user2<-c("userR", "useR", "UserR")

user1==user2 #this is a logical vector
[1]  TRUE FALSE  TRUE
```

Conditionals and Control Flow

- Example 3: Equality (or not): Comparison of strings.
  - Set the two vectors below equal to each other
  - This creates a logical vector (with TRUE/FALSE)

```
user1<-c("userR", "user", "UserR")
user2<-c("userR", "useR", "UserR")

user1==user2 #this is a logical vector
[1]  TRUE FALSE  TRUE
```

Demonstration: Let's use logical operators to subset our data.

Conditionals and Control Flow

- Example 5: The greater than ($>$), and less than ($<$) operators. Dealing with unusual points:

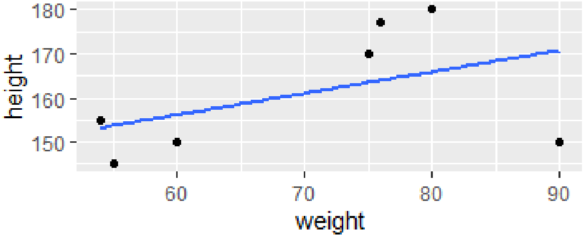


Figure: There appears to be an unusual point at (90, 150)

.

Conditionals and Control Flow

- Example 5: The greater than ($>$), and less than ($<$) operators. Dealing with unusual points:

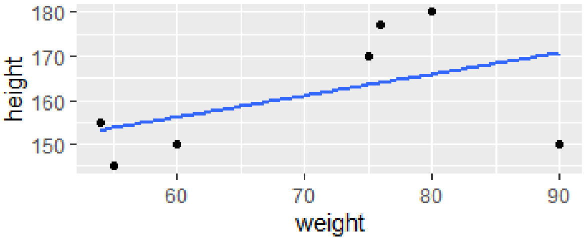


Figure: There appears to be an unusual point at (90, 150)

. **Demonstration:** Let's use logical operators to subset our data.

Conditionals and Control Flow

- Example 6: The & and | operators. Dealing with unusual points:

```{r}
#recall
features
```

height <dbl>	weight <dbl>
150	60
145	55
150	90
155	54
170	75
180	80
177	76

7 rows

Figure: There appears to be an unusual point at (90, 150)

Conditionals and Control Flow

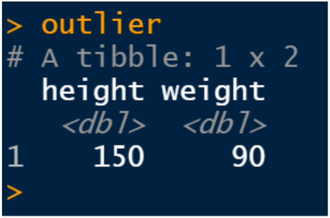
- Example 6: The & and | operators:

```
outlier <-features[ which(weight==90 & height==150), ]
outlier

features3<-features[ which(weight==90 | height==150), ]
features3
```

Conditionals and Control Flow

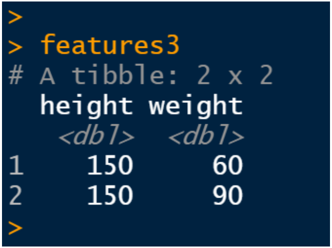
- Example 6: The & and | operators:

```
outlier <-features[ which(weight==90 & height==150), ]
outlier

features3<-features[ which(weight==90 | height==150), ]
features3
```



```
> outlier
# A tibble: 1 x 2
  height weight
   <dbl>  <dbl>
1    150     90
>
```

(a) The subset with & operator



```
>
> features3
# A tibble: 2 x 2
  height weight
   <dbl>  <dbl>
1    150     60
2    150     90
>
```

(b) The subset with | operator.

Figure: Data frames created from the AND and OR logical operators

Loops

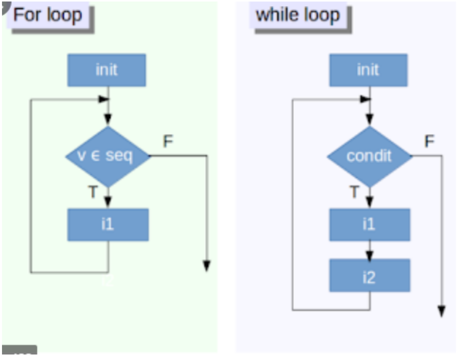


Figure: Credit: A figure of a for and while loop loops

A **loop** is a replication of instructions. It's a mini-step process that organizes a sequence of actions into parts that need to be repeated. Diamonds, on the other hand, are called **decision symbols**, and translate into questions which only have two possible logical answers; TRUE or FALSE.

Loops

Example 7: For loop

- Start with an intialization (e.g. vector or matrix) of empties.
- Tests to whether a current value is within a specified defined range (i.e. within 1:100).
- If the condition is not met and the resulting outcome is False, the loop is never executed.

Loops

Example 7: For loop

- Start with an intialization (e.g. vector or matrix) of empties.
- Tests to whether a current value is within a specified defined range (i.e. within 1:100).
- If the condition is not met and the resulting outcome is False, the loop is never executed.

```
set.seed(1)
rv <- rnorm(1000, 0, 1)
usq<-matrix("NA", 100, 1)

for(i in 1:100) {
usq[i] <- rv[i]*rv[i]
print(usq[i])
}
usq<-data.frame(usq=as.numeric(unlist(usq)))
attach(usq)
```

Loops

Example 7: For loop

- Once the condition is past 100, the evaluation is False, and the loop ends:

```
 > dim(usq)
[1] 100   1
>
```

- The data frame is entries for the ______ distribution.

Loops

Example 7: For loop

- Once the condition is past 100, the evaluation is False, and the loop ends:

```
 > dim(usq)
[1] 100   1
>
```

- The data frame is entries for the ______ distribution.
- The data frame is entries for the chi distribution (denoted χ^2_1).

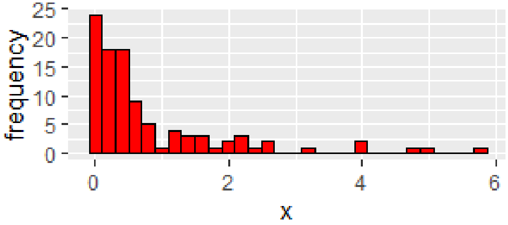


Figure: The square of a standard normal is chi

Loops

Example 8: While loop

- Start with an intialization (e.g. vector or matrix) of empties.
- Now followed by a logical comparison between a control variable and a value (using earlier defined conditionals).
- If the condition is not met and the resulting outcome is False, the loop is never executed.

Loops

Example 8: While loop

- Start with an intialization (e.g. vector or matrix) of empties.
- Now followed by a logical comparison between a control variable and a value (using earlier defined conditionals).
- If the condition is not met and the resulting outcome is False, the loop is never executed.

```
set.seed(1)
usq<-matrix("NA", 100, 1)
n = length(usq)
x=0
while (x<=n) {
x<-sum(x, 1)
i<-sum(i, 1)
usq[i] <- rv[i]*rv[i]
print(usq[i])
}
usq<-data.frame(usq=as.numeric(unlist(usq)))
attach(usq)
```

Loops

Example 8 : While loop

- Once the condition is past 100, the evaluation is False, and the loop ends:

```
> dim(usq)
[1] 100   1
>
```

.

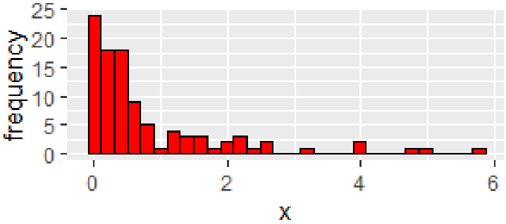


Figure: The square of a standard normal is chi

Note: The while loop was constructed to yield the same output as the for loop. This was less efficient.

Example 8: Mix up loops with control flow

- Example 4 (revisted): Outlier point in *weight* and height.

```
weight
height

# Code the for loop with conditionals
for (i in weight) {
if (i > 80) {
print("You're an outlier!")
}
else {
print("Nothing to see here!")
}
print(i)
}
```

Loops

Example 8: Mix up loops with control flow

- Example 4 (revisted): Outlier point in *weight* and height.

```
weight
height

# Code the for loop with conditionals
for (i in weight) {
if (i > 80) {
print("You're an outlier!")
}
else {
print("Nothing to see here!")
}
print(i)
}
```

Demonstration: This is a bit silly but let's see what we get.

Part II

Question: What have we examined and why?

Part II

Question: What have we examined and why?

1. Conditionals and logical statements
2. Loops and the role of logical statements/condititons
3. The role of if statements in executing logicals

Part II

Question: What have we examined and why?

1. Conditionals and logical statements
2. Loops and the role of logical statements/condititons
3. The role of if statements in executing logicals

Answer: All of these components are important features of tidying data in R.

Functions: Review

An **R function** is created by using the keyword *function*. The basic syntax of an R function definition is as follows:

```
function_name <- function(arg_1,..) {
Function_body
}
```

The different parts of a function are:

- Function Name = This is the actual name of the function. It is stored in the R environment as its name.
- Arguments - This is an placeholder to pass an input value into your function (i.e. $f(x)$).
- Function Body - The function contains the statements that determines what the function does.

Example 9: A perfect square in R

```
>
> perfect.squares<-function(x){for (x in 1:10){
+     if(x%%x == 0)
+         b<-x^2
+     print (b)
+ }}
> perfect.squares(10)
```

Figure: A function in R to compute perfect squares of all the numbers in the range of 1 to 10

Example 9: A perfect square in R

```
>
> perfect.squares<-function(x){for (x in 1:10){
+     if(x%%x == 0)
+         b<-x^2
+     print (b)
+ }}
> perfect.squares(10)
```

Figure: A function in R to compute perfect squares of all the numbers in the range of 1 to 10

Demonstration: Let's see what we have printed.

Example 9: A perfect square in R

Here is how the function is stored as an object in the R environment:

Function: perfect.squares (.GlobalEnv) (Read-only)

```
function(x){for (x in 1:10){
    if(x%%x == 0)
        b<-x^2
    print (b)
}}
```

Figure: A function in R stored in the global environment.

Example 10: Processing Grades in R

Here is a function that evaluates at two inputs:

```
translate<-function(x,y){
if (x=="NA"){x=0}
else if (x!="NA"){x=x}
print(x)
if (y== -1){y="NA"}
else if (y!=-1){y=y}
print(y)
}
```

- If x is "NA" then it will assign a value of 0.
- else if y is equal to -1 then it will assign a value of "NA"
- else let y=y.

Example 10: Processing Grades in R

Here is the output at different pairs of (x, y).

```
> #Testing the function with input values
> x<-"NA"
> y<-"-1"
>
> x2<-1
> y2<-0
>
> translate(x, y)
[1] 0
[1] "NA"
> translate(x2, y2)
[1] 1
[1] 0
> 
```

Figure: A function in R stored in the global environment.

Question: Does anyone see a problem with the way in which the function is evaluating the output?

Example 10: Processing Grades in R

Here is the output at different pairs of (x, y).

```
> #Testing the function with input values
> x<-"NA"
> y<-"-1"
>
> x2<-1
> y2<-0
>
> translate(x, y)
[1] 0
[1] "NA"
> translate(x2, y2)
[1] 1
[1] 0
> 
```

Figure: A function in R stored in the global environment.

Question: Does anyone see a problem with the way in which the function is evaluating the output?

- Ans: It does so term by term, which is inefficient if we have a vector of entries that we want to process (let's see).

Example 11: ifelse function in R

In order to understand the brilliance of this function let's go back to Example 10.

```
> x3<-c(0, "NA", 1)
> y3<-c(-1, "NA", 80)
>
> translate(x3,y3)
[1] "0"  "NA" "1"
[1] "NA"
Warning messages:
1: In if (x == "NA") { :
  the condition has length > 1 and only the first e
lement will be used
2: In if (x != "NA") { :
  the condition has length > 1 and only the first e
lement will be used
3: In if (y == -1) { :
  the condition has length > 1 and only the first e
lement will be used
> |
```

Figure: The translate function in R with vector inputs.

Example 11: ifelse function in R

The **'ifelse'** is a built in base R function that returns a value with the same length as the test, rather than the evaluation at the first element.

```
x3<-c(0, "NA", 1)
> z<-x3
> ifelse(z==0, "NA", z)
[1] "NA" "NA" "1"
>
```

Aside:

- In statistics education it is common to use placeholders of "-1" when a student has a valid documentation for absence.
- If no valid documentation is received, the student receives a value of 0.
- The ifelse function evaluates these condition term by term on the entire vector.

Subset function in R

```
library(myPackage)
attach(Grades)
view(Grades)
```

```
head(Grades)

TT1 = `Term Test 1 (/505)`

```
```

	Term Test 1 (/505) <dbl>
1	34
2	36
3	26
4	45
5	18
6	34

6 rows | 1-2 of 4 columns

Figure: Processed marks for an undergraduate statistics course.
```

Subset function in R

The **subset** function is the easiest way to select variables and observations (at least within base R).

Example 10-11 revisted: Grade Processing: Suppose that I want to select only those students who have no noted absences in Term Test 1 (TT1) and Term Test 2 (TT2)

```
dim(Grades)

newdata <-newdata <- subset(Grades, TT1 !=-1 | TT2!= -1,)
dim(newdata)

newgrades <- subset(Grades, TT1 !=-1 & TT2!= -1,)
dim(newgrades)
```

Subset function in R

The **subset** function is the easiest way to select variables and observations (at least within base R).

Example 10-11 revisted: Grade Processing: Suppose that I want to select only those students who have no noted absences in Term Test 1 (TT1) and Term Test 2 (TT2)

```
dim(Grades)
```

```
newdata <-newdata <- subset(Grades, TT1 !=-1 | TT2!= -1,)
dim(newdata)
```

```
newgrades <- subset(Grades, TT1 !=-1 & TT2!= -1,)
dim(newgrades)
```

Demonstration: What is the difference?

new.function() function in myPackage

The data set *newgrades* is closer to what would use to process grades for undergraduate students.

- This is still not the ideal realization of my grades vectors (I need to account for 0s).

```
> view(new.grades)
> new.grades
function(x)
{x=ifelse(x=="NA", 0,
          ifelse(x==-1, "NA", x))
}
<bytecode: 0x0000023167a2b188>
<environment: namespace:myPackage>
> |
```

Figure: The new.grades function translates "-1" to "NA" and "NA" to "0".

Demonstration: Revisit the TT1 and TT2 grades in the Statistics data set.

Tidyverse

Next information session: The **tidyverse** package is a collection of R packages designed for data science.

- Efficiency of data clean up
- Tidy pipeline structures
- Sophisticated plots (in fact I used ggplot here)

Resource: Tidyverse