STATS 380 MID-SEMESTER TEST NOTES AND REVISION

R Basics

```
7 / 3                   # Decimal division
7 %/% 3                 # Integer division (floor)
7 %% 3                  # Modulus operation
list.files("Output")    #Gives all the files in the directory
\n                      # New line
\t                      # Tab
\\                      # Literal backslash
```

Vectors

All values have to be of the same mode().

```
length()                # Returns the number of elements in a vector
```

The Recycling Rule

The shorter vector will be repeated to make it the length of the longer vector.

```
vector_1 <- c(1, 1, 1, 1)
vector_2 <- c(5, 6, 7)
vector_1 + vector_2
[1] 6 7 8 6
```

Generating Vectors

```
seq(1, 10, by=2)
[1] 1 3 5 7 9
seq(1, by=2, length.out=10)
[1] 1 3 5 7 9 11 13 15 17 19
rep(c(5, 10), c(6, 3))
[1] 5 5 5 5 5 5 10 10 10
rep(c(5, 10), each=10)
[1] 5 5 5 5 5 5 5 5 5 5 10 10 10 10 10 10 10 10 10 10

minutes
hours
[1]  0 10 20 30 40 50
[1] 800 800 800 800 800 800 900 900 900 900 900 900
minutes + hours
[1] 800 810 820 830 840 850 900 910 920 930 940 950

rep(100*8:12, each=2) + c(0, 30)
[1] 800 830 900 930 1000 1030 1100 1130 1200 1230
```

Logical Vectors

```
grades <- c(80, 32, 92, 72, 54, 88, 49, 41, 63, 47)
low <- grades > 50
low
[1] TRUE FALSE TRUE TRUE TRUE TRUE FALSE FALSE TRUE FALSE
high <- grades < 50
high
[1] FALSE TRUE FALSE FALSE FALSE FALSE TRUE TRUE FALSE TRUE

%in%                    # Checks whether each item in a vector can be found in another vector
grades2 <- c(15, 53, 62, 76, 46, 83, 46, 64, 47, 80)
grades %in% grades2
[1] TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE
```

Subsetting # (1) Create a logical vector of days with moods > 5, (2) Pull out those days

```
moods <- c(10, 8, 2, 5, 5, 4, 7)
goodDays <- moods > 5
goodDays
[1] TRUE TRUE FALSE FALSE FALSE FALSE TRUE
moods[goodDays]
[1] 10 8 7
```

Custom Ordering

```
myName <- c("n", "a", "i", "B", "r", "n", "a")
o <- c(4, 5, 3, 2, 1, 6, 7)
myName[o]
[1] "B" "r" "i" "a" "n" "n" "a"
```

Appending to Vectors # The after= parameter refers to the INDEX not the element itself

```
animals <- c("Wolf", "Flamingo", "Seal", "Eagle")
append(animals , "Penguin", after=3)
[1] "Wolf" "Flamingo" "Seal" "Penguin" "Eagle"
```

Character Functions

substring()

```
brianna <- c("Animal Jam", "Business Analytics", "Crochet", "Deloitte", "Egg tart", "Fuck", "Gymnastics")
substring(brianna, 1, 1)
[1] "A" "B" "C" "D" "E" "F" "G"
```

strsplit()

```
strsplit(brianna, " ")
[[1]]
[1] "Animal" "Jam"
[[2]]
[1] "Business" "Analytics"
[[3]]
[1] "Crochet"
[[4]]
[1] "Deloitte"
[[5]]
[1] "Egg" "Tart"
[[6]]
[1] "Fuck"
[[7]]
[1] "Gymnastics"
```

paste()

```
context <- c("I play", "My major is", "I enjoy", "I work at", "I eat", "My favourite word is", "My sport is")
paste(context, brianna, sep=" ")
[1] "I play Animal Jam" "My major is Business Analytics" "I enjoy Crochet" "I work at Deloitte"
[5] "My favourite word is Fuck" "My sport is Gymnastics"

paste(context, brianna, sep=", ", collapse=" ")
[1] "I play, Animal Jam My major is, Business Analytics I enjoy, Crochet I work at, Deloitte I eat, Egg tart My favourite word is,
Fuck My sport is, Gymnastics"
```

nchar() # Returns the number of characters per string in a character vector

```
nchar(brianna)
```

```
[1] 10 18  7  8  8  4 10

items <- c("Worn Blankets", "Spiked Collars", "Fox Hats", "Cupid Wings")
nchar(items)
[1] 13 14 8 11
nchar(items)-1
[1] 12 13 7 10
substring(items , nchar(items)-1) # Substring without a third parameter assumes to the end of the character value
[1] "ts" "rs" "ts" "gs"
substring(items , 1, nchar(items)-1)
[1] "Worn Blanket"  "Spiked Collar" "Fox Hat" "Cupid Wing"

times <- paste(rep(8:12, each=2), sprintf("%02d", seq(0, 50, 10)), sep=":")
[1] "8:00" "8:10" "9:20" "9:30" "10:40" "10:50" "11:00" "11:10" "12:20" "12:30"

grep()                          # Gets the indexes of character(s) in a vector if it contains the specified character
grep(":00", times, fixed=TRUE)  # fixed=TRUE means the first argument is literal text
[1] 1 7

grepl()                         # Same as grep() but returns a logical vector
grepl(":00", times, fixed=TRUE)
[1] TRUE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE

gsub()                          # gsub() finds and removes (replaces)
gsub("s", "z", items, fixed=TRUE)
[1] "Worn Blanketz"  "Spiked Collarz" "Fox Hatz" "Cupid Wingz"
```

Lists # Instead of storing variables separately, we can store them in a list

```
name = "Brianna"
age = 21
birthday = "09-05-2003"
myFriends <- c("Mellisa", "Aulia", "Tanveer")
me <- list(name = "Brianna",
           age = 21,
           birthday = "09-05-2003",
           friends = myFriends)
me
$name
[1] "Brianna"

$age
[1] 21

$birthday
[1] "09-05-2003"

$friends
[1] "Mellisa" "Aulia"   "Tanveer"

me$friends[2]                   # Extracting an element in a vector stored in a list
[1] "Aulia"
```

R Graphics

cex()   # A size multiplier
col = hcl(240, 40, 40, alpha=0.5) # Changing the transparency of colours using alpha() and the HSL triplets

Manual Plotting

plot.new()	# Start a new empty plot
plot.window(x, y)	# Sets scale on axes
points(x, y)	# Adds data points
lines(x, y)	# Adds the trend line; Can make use of col, lty, pch
axis(1)	# Draws the x axis
axis(2)	# Draws the y axis
box()	# Adds a border

<u>Two plots in one</u>

```
grades <- c(87, 74, 95, 63)
attendance <- c(10, 9, 10, 5)

grades2 <- c(41, 56, 72, 77)
attendance2 <- c(2, 4, 8, 8)

plot.new()
plot.window(range(grades, grades2), range(attendance, attendance2))
points(grades, attendance, col=2)
lines(grades, attendance)
points(grades2, attendance2, col=4)
lines(grades2, attendance2)
axis(1)
axis(2)
box()
```

<u>Legends</u>

```
legend("topleft", legend = c("2025", "2024"), col=c(2, 4), pch=1, lwd=1)
```

R Data Structures

Matrices

```
matrix(1:4, nrow=2)
1 3
2 4
matrix(1:20, nrow=3, ncol=7)
1 4 7 10 13 16 19
2 5 8 11 14 17 20
3 6 9 12 15 18 01
```
# Recycling rule applies where the final element = 1

```
matrix(1:9, nrow=3, byrow=TRUE)
1 2 3
4 5 6
7 8 9
cbind(1:3, 4:6, 7:9, 10:12)
1 4 7 10
2 5 8 11
3 6 9 12
```

```
rbind(1:3, 4:6, 7:9, 10:12)
  1   2   3
  4   5   6
  7   8   9
10 11 12
hours <- 0:11                                  # [1] 0 1 2 3 4 5 6 7 8 9
minutes <- seq(0, 50, 10)                      # [1] 0 10 20 30 40 50
as.numeric(outer(minutes, hours*100, "+"))     #hours*100 is 0 100 200 300 400 500 600 700 800 900
00 100 200 300 400 500 600 700 800 900
10 110 210 310 410 510 610 710 810 910
20 120 220 320 420 520 620 720 820 920
30 130 230 330 430 530 630 730 830 930
40 140 240 340 440 540 640 740 840 940
50 150 250 350 450 550 650 750 850 950

myMat <- matrix(1:9, nrow=3)
myMat
1 4 7
2 5 8
3 6 9
myMat * 2                 # Mathematical operations apply to all elements, similar to vector arithmetic
1   8 14
4 10 16
6 12 18
```

<u>Transpose</u>

```
t(myMat )                 # Flips rows and columns
1 2 3
4 5 6
7 8 9
```

Naming Matrices Rows and Columns

```
animalJam <- matrix(c("Eagle", "Flamingo", "Seal", "Spike", "Worn", "Necklace", "Bear", "Owl", "Mouse"), nrow=3)
        [,1]          [,2]          [,3]
[1,]    "Eagle"       "Spike"       "Bear"
[2,]    "Flamingo"    "Worn"        "Owl"
[3,]    "Seal"        "Necklace" "  Mouse"

colnames(animalJam) <- c("Animals", "Clothing", "Pets")
rownames(animalJam) <- c("Avatar 1", "Avatar 2", "Avatar 3")
             Animals       Clothing      Pets
Avatar 1     "Eagle"       "Spike"       "Bear"
Avatar 2     "Flamingo"    "Worn"        "Owl"
Avatar 3     "Seal"        "Necklace"    "Mouse"
```

Subsetting Matrices

```
animalJam[2, 3]
[1] "Owl"
animalJam[3, 1:3]
[1] "Seal" "Necklace" "Mouse"
animalJam[, "Animals"]
[1] "Eagle" "Flamingo" "Seal"
```

```
animalJam["Avatar 1", "Animals"]
[1] "Eagle"
```

Factors

```
longSpikes <- c("Black", "Red", "Pink", "Purple","Blue", "Yellow", "Green", "Orange")
haveSpikes <- c(1, 0, 0, 0, 0, 1, 0, 0)
spikes_df <- data.frame(longSpikes, haveSpikes)
spikes_df <- data.frame("longSpikes " = longSpikes, "haveSpikes" = haveSpikes)

spikeState <- factor(haveSpikes, labels=c("Has spike", "No spike"))
[1] No spike  Has spike Has spike Has spike Has spike No spike  Has spike Has spike
Levels: Has spike No spike
```

In R, levels automatically use the unique character values in alphabetical order

```
daysOfWeek <- c("SA", "TH", "TU", "MO", "WE", "MO", "TH", "SA", "WE", "WE", "FR", "SU")
daysFactor <- factor(daysOfWeek )
[1] SA TH TU MO WE MO TH SA WE WE FR SU
Levels: FR MO SA SU TH TU WE                    # An unnatural order, sorted alphabetically
```

So, we can specify a custom order using label

```
daysFactor <- factor(daysOfWeek, label=c("MO", "TU", "WE", "TH", "FR", "SA", "SU"))
[1] WE FR SA TU SU TU FR WE SU SU MO TH
Levels: MO TU WE TH FR SA SU                    # Factors no sorted in a natural order
```

Factor Functions: table()

```
table(daysFactor )                              # table() returns a table of counts automatically
MO TU WE TH FR SA SU
 1   2   2   1  2   1  3
```

Data Frames

subset()

The subset() function can be used to select rows and/or columns

```
grades <- c(95, 84, 76, 87, 80)
courses <- c("STATS330", "COMPSCI345", "COMPSCI399", "BUSAN300", "BUSAN302")
semOne <- data.frame(Grades=grades, Courses = courses)
subset(semOne , grades > 80)
 Grades    Courses
1     95   STATS330
2     84   COMPSCI345
4     87   BUSN300
```

<u>Ordering a Data Frame</u>

```
o <- order(semOne$Grades, decreasing=TRUE)
semOne[o, ]
 Grades    Courses
1     95   STATS330
4     87   BUSAN300
2     84   COMPSCI345
5     80   BUSAN302
3     76   COMPSCI399
```

Combining Data Frames

```
grades2 <- c(72, 89, 76, 88)
courses2 <- c("COMPSCI335", "STATS220", "BUSINESS350", "BUSAN305")
semTwo <- data.frame(Grades=grades2 , Courses = courses2 )      # New data frame
```

To apply a ID column so we can identify which rows are from which semester:

```
s1 <- cbind(ID="S1", semOne )
s2 <- cbind(ID="S2", semTwo )
wholeYear <- rbind(s1, s2)
ID        Grades     Courses
S1        95         STATS330
S1        84         COMPSCI345
S1        76         COMPSCI399
S1        87         BUSAN300
S1        80         BUSAN302
S2        72         COMPSCI335
S2        89         STATS220
S2        76         BUSINESS350
S2        88         BUSAN305
```

R Programming

Control Flow

For Loops

```
wishList<- c("Headdress", "Party Hat", "Snow Leopard Slippers", "Furry Hat", "Glitched Ring")
for (item in wishList){
        print(item)
}
[1] "Headdress"
[1] "Party Hat"
[1] "Snow Leopard Slippers"
[1] "Furry Hat"
[1] "Glitched Ring"
```

For Loops with conditions

```
numbers <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
for (num in numbers){
        if (num %% 2 == 0){
                print(paste0(num, " is even"))
        }else{
                print(paste0(num, " is odd"))}
        }
[1] "1 is odd"
[1] "2 is even"
[1] "3 is odd"
[1] "4 is even"
[1] "5 is odd"
[1] "6 is even"
[1] "7 is odd"
[1] "8 is even"
[1] "9 is odd"
[1] "10 is even"
```

While Loops

```
count <- 1
while (count < 11){
        if (numbers[count] %% 2 == 0){
                print(paste0(numbers[count], " is even"))
                count <- count + 1
        }else {
                print(paste0(numbers[count], " is odd"))
                count <- count + 1
        }
}
[1] "1 is odd"
[1] "2 is even"
[1] "3 is odd"
[1] "4 is even"
[1] "5 is odd"
[1] "6 is even"
[1] "7 is odd"
[1] "8 is even"
[1] "9 is odd"
[1] "10 is even"
```

The ifelse() Function

```
numbers <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
oddEven <- ifelse(numbers %% 2 == 0, "EVEN", "ODD")
[1] "ODD"  "EVEN" "ODD"  "EVEN" "ODD"  "EVEN" "ODD"  "EVEN" "ODD"  "EVEN"
```

Functions

```
countEvenOddNumbers <- function(numbers){
        evenCount <- 0
        oddCount <- 0
        for (num in numbers){
                if (num %% 2 == 1) {
                        oddCount <- oddCount + 1
                } else {
                        evenCount <- evenCount + 1
                }
        }
        return(c(evenCount, oddCount ))
}
numbers <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
countEvenOddNumbers(numbers)
[1] 5 5
```

Statistical Computing

Split-Apply-Combine

```
studyHours <- c(840, 950, 1030, 850, 920, 750, 1940, 2030, 2310, 2330)
mood <- c(5, 6, 3, 7, 3, 8, 9, 8, 7, 8)
```

Say we want to plot the study hours by mood, but have a separate trendline for day and night time hours.

```
late <- studyHours > 1800
early <- studyHours < 1800
night <- ifelse(late, "night", "day")
night
[1] "day" "day" "day" "day" "day" "day" "night" "night" "night" "night"
```

split()

The split() function takes a vector or data frame and breaks it into pieces, the first argument is a vector to split, and the second argument is a vector of values that indicate group membership.

```
studyHoursDayNight <- split(studyHours, night)
$day
[1] 840 950 1030 850 920 750
$night
[1] 1940 2030 2310 2330
```

lapply()

The lapply() function helps to perform action on each "group" of a list more efficiently. For example, if we wanted to compute the mean of day and night, we would simply call mean() on both groups, although if we have many groups, the code becomes very long and repetitive.

```
studyHoursNightAvg <- lapply(studyHoursDayNight , mean)
$day
[1] 890
$night
[1] 2152.5
```

sapply()

Works the same way as lapply() but attempts to simplify the result into a vector is possible

unlist()

The unlist() function reduces a list into a vector.

```
unlist(studyHoursNightAvg )
  day    night
890.0 2152.5
```

do.call()

Say we want to use lapply() to derive the ranges.

```
studyHoursRange <- lapply(studyHoursDayNight, range)
unlist(studyHoursRange )
day1  day2 night1 night2
750   1030   1940   2330
```

Now we end up with a meaningless vector. A better approach would be to use do.call().

```
do.call(rbind, studyHoursRange)
        [,1] [,2]
day    750 1030
night 1940 2330
```

apply()

We can also use apply() using a matrix

```
m <- matrix(1:12, nrow=3)
apply(m, 1, sum)
[1] 22 26 30
apply(m, 1, mean)
[1] 2 5 8 11
```

Anonymous Functions

```
lapply(studyHoursDayNight, function(x) mean(x, na.rm=TRUE))
```

Statistical Functions

Distributions

```
dnorm(x), pnrom(q), qnorm(p)          # Normal Distributions
dt(x), pt(q), qt(p)                   # t Distributions
dF(x), pF(q), qF(p)                   # F Distributions
dbinom(x), pbinom(q), qbinom(p)       # Binomial Distributions
dpois(x), ppois(q), qpois(p)          # Poisson Distributions
```

LECTURE NOTES

T = TRUE = 1

F = FALSE = 0

Order of Types

Logical ⟶ Numerical ⟶ Character

```
a <- c(8e2, '1e1')          # The 8e2 is evaluated first before converting into a String
[1] "800" "1e1"

b <- c(1, 2, 3, NA)
length(b)
[1] 4

c <- c(1, 2, 3, NULL)
length(c)
[1] 3
c[-1]
[1] 2 3

d <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20)
d[-1]
[1] 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
d[d[-1]]
[1] 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 NA
d[-length(d)]
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

fruit <- c("apple", "banana", "orange", "lemon", "grape")
grep("a", fruit)
[1] 1 2 3 5
grepl("a", fruit)
[1] TRUE TRUE TRUE FALSE TRUE
```

```
grep("^a", fruit)
[1] 1
grepl("a$", fruit)
[1] FALSE TRUE FALSE FALSE
```

Operations

```
0^0             # Nan
0/0             # NaN
1/0             # Inf
-1/0            # -Inf
log(0)          # -Inf
log(-1)         # NaN
Inf - Inf       # NaN
Inf / Inf       # NaN
Inf + Inf       # Inf
Inf * 0         # Nan
1/Inf           # 0
Inf ^ 0         # 1
0 ^ Inf         # 0
NaN + 1         # NaN
NA == NA        # NA
NA + 1          # NA
NA == NULL      # NA
TRUE + 1        # 2
FALSE + 1       # 1
NA > 1          # NA
sqrt(NA)        # NA
sqrt(-1)        # NaN
```