Getting Started with R

Setting up the environment

Please download and install R (3.4.3):

- Windows: https://cran.r-project.org/bin/windows/base/R-3.4.3-win.exe
- Mac: https://cran.r-project.org/bin/macosx/R-3.4.3.pkg

Install the latest version of R Studio for Windows/Mac:

- https://www.rstudio.com/products/rstudio/download/#download

Once both are installed open the RStudio application and follow along with this short video to get familiar with the console.

Getting Started

Using Packages

```
install.packages("data.table") #Downloads and installs a package from CRAN
library(data.table) #Load the package into the session, making all its functions available to use
```

Working Directory

```
getwd() #Find the current working directory (where inputs are found and outputs are sent)
setwd("C://new/file/path') #Change the current working directory

#Note: Use Projects (New Project... in File menu) in RStudio to set the working directory to the folder you are w
orking in.
```

Getting Help

```
?sum #Get information of a function
help.search("Weighted sum") #Search R help files for a word or phrase
help(package = "data.table") #Find help for a package
```

Creating Variables

Type the following in the console

```
x = c(1,2,3)
x
```

Data Types

Data types in R. Converting between data types can always occurr from a higher values in the list to a lower value.

- Boolean | e.g TRUE, FALSE | Conversion function: *as.logical*
- Integers, Floats | e.g 1,3.4,3...10 | Conversion function: *as.numeric*
- Characters, Factors | e.g "1", "bear" | Conversion function: *as.character* / *as.factor*

```
#Logical: Boolean values (TRUE or FALSE)
logical = c(TRUE,FALSE,TRUE)
class(logical)
#Numeric: Integers or floating point numbers
numeric = c(1,2,3,4)
class(numeric)
#Character: Character strings
character = c("this","is","a","character")
class(character)
#Factor:Character strings with preset levels
factor = as.factor(c("this","is","a","factor"))
class(factor)
```

Vectors

A vector is a one-dimensional collection of elements of numbers, strings or logical values. Elements in a vector are indexed by position. Mathematical operations can be directly performed on vectors.

Creating Vectors

```
c(2, 4, 6) #Join elements into a vector
2:6 #An integer sequence
seq(2, 3, by=0.5) #A complex sequence
rep(1:2, times=3) #Repeat a vector
rep(1:2, each=3) #Repeat elements of a vector
```

Vector Functions

```
x = c(1,2,3,3,4,5)

sort(x) #Return x sorted.
rev(x)  #Return x reversed.
table(x) #See counts of values.
unique(x) #See unique values.
length(x) #See number of elements.
```

Selecting Vector Elements

```
x = c("f","e","d","c","b","a")


x[1] #Select the first element
x[-4] #Select all but the forth element
x[2:4] #Select the second to fourth element
x[-(2:4)] #Select everything except the second to fourth elements
x[c(1, 5)] #Select elements one and five
x[x == "a"] #Select elements equal to "a"
x[x < "c"] #Select all elements before "c"
x[x %in%c("a","b","c")] #Select all element in "a","b","c"
```

Vector operations

```
num = c(1,2,3,4,5)
char = c("a","b","c","d")
#Can perform arithmetic directly on numeric vectors
num+5
num-5
num*5
num/5
num%%5
#Can perform arithmetic directly on vectors
num==5
num>=5
num<=5
num!=5
char!="z"
```

Mathematical Vector Functions

```
x = 1:100
y = 1:100
n = 10

log(x) #Natural log
sum(x) #Sum
exp(x) #Exponential
mean(x) #Mean
max(x) #Largest element
median(x) #Median
min(x) #Smallest element
quantile(x) #Percentage quantiles
round(x, n) #Round to n decimal places
rank(x) #Rank of elements
signif(x, n) #Round to n significant digits
var(x) #Variance.
cor(x, y) #Correlation.
sd(x) #Standard deviation.
```

Matrices

Two-dimentional collection of elements. Indexed by rows or columns. All columns have to be of the same data type. Arithmetic and logical operations can be directly performed on columns (type permitting).

```
x = 1:9
m = matrix(x, nrow = 3, ncol = 3)

m[2, ] #Select a row
m[ , 1] #Select a column
m[,3] #Select an element


t(m) #Transpose (rows to columns, columns to rows)
m %*% x[1:3] #Matrix multiplication
```

Lists

A list is a collection of elements which can be of different types.

```
l = list(x = 1:5, y = c('a', 'b'))

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

DataFrames / DataTables

Two-dimentional collection of elements. Indexed by rows or columns. All columns do not have to be of the same data type and can be indexed by a label.

```
dt=data.table(Col1=1:10,Col2=letters[1:10],Col3=as.factor(letters[11:20]))


#Indexing and operations
dt[1,,] #Indexing by row
dt[,Col1,] #Indexing by column
dt[1:5,Col1,] #Indexing by row and column
dt$Col1*10 #Arithmetic operations
dt$Col1==10 ##Logical operations

#Methods
nrow(dt) #Number of rows
ncol(dt) #Number of columns
dim(dt) #Number of cols and rows
cbind(dt[,.(Col1,Col2)],dt[,.(newCol=Col3)]) #Bind columns
rbind(dt[1:5],dt[6:10]) #Bind Rows
```

Importing / Exporting Data

```
###Importing text or csv files
dt = fread('c://path//to//file.csv')

###Exporting text or csv files
fwrite(df,'c://path//to//file_output.csv')

#See the readr or data.table package for more information.
```

Programming

For-Loops

```
for (element in vector){
  Do something
}

#Example
for (i in 1:10){
  j = i * 10
  print(j)
}
```

While-Loops

```
While (element in vector){
  Do something
}

#Example
j=0
while (j<100){
  j = j * 10
  print(j)
}
```

If Statements

```
if (condition){
  Do something
} else {
  Try something different
}

#Example
x = 1
if (x > 3){
  print('Yes')
} else{
  print('No')
}
```

Functions

```
function_name <- function(parameter){
  Do something
  return(output)
}

#Example
vec_length = function(vec){
  length = len(vec)
  return(length)
}
```