The purpose of this exercise is to repeat some of earlier exercise on data manipulation and to set you up for Data Pre-processing.

Topics covered:

- Data types in R
- Factors in R
- Reading and manipulating data from csv files.

R—The Basics

If you use R often you will likely need to read in data at some point. While R can read excel .xls and .xlsx files these filetypes often cause problems. Comma separated files (.csv) are much easier to work with. It is best to save these files as csv before reading them into R. If you need to read in a csv with R the best way to do it is with the command read.csv. In this exercise we shall be making use of the ”weather” dataset which is a .csv dataset.

Understanding ‘Data Types’ in R

For many who have used other data analysis software or who have a programming background, you will be familiar with the concept of ‘data types’.

R strictly stores data in several different data types, called ‘classes’:

- `Numeric – e.g. 3.1415, 1.618`
- `Integer – e.g. -1, 0, 1, 2, 3`
- `Character – e.g. “vancomycin”, “metronidazole”`
- `Logical – TRUE, FALSE`
- `Factors/categorical – e.g. male or female under variable, gender`

R also usually does not allow mixing of data types for a variable, except in a:

- `List – as a one dimensional vector, e.g. c(“vancomycin”, 1.618, “red”)`
- `Data-frame – as a two dimensional table with rows (observations) and columns (variables)`

Lists and data-frames are treated as their own ‘class’ in R.

Query output from our dataset will be in the form of data tables with different data types in different columns. Therefore, R usually stores these tables as ‘data-frames’ when they are read into R.

Special Values in R

- NA – 'not available', usually a default placeholder for missing values.
- NAN – 'not a number', only applying to numeric vectors.
- NULL – 'empty' value or set. Often returned by expressions where the value is undefined.
- Inf – value for 'infinity' and only applies to numeric vectors.

For this exercise you will need to install a package called 'reshape2' (case sensitive). We shall be needing this package to create/manipulate our data frame later on in this exercise.

Setting Working Directory

This step tells R where to read in the source files.

Command: setwd("directory_path")

In this case: (your data file (weather) is saved in a folder called "test" on the Desktop)

```
setwd("~/Desktop/test")

#List files in directory:

list.files()  #Command

#Output

[1] "~$ta Pre-Processing Exercise.R" "Data Pre-Processing Exercise.R"
[3] "weather.csv"
```

Reading in .csv Files from test Query Results

The data read into R is assigned 'test' for reference.

test <- read.csv("weather.csv")

Viewing the Dataset

There are several commands in R that are very useful for getting a 'feel' of your datasets and see what they look like before you start manipulating them.

- View the first and last 2 rows. E.g.:

```
head(test, 2)
```

```
head(test, 2) #command

#output

  outlook temperature humidity windy play
1   sunny         hot     high    no   no
2   sunny         hot     high   yes   no



#output

tail(test, 2)
    outlook temperature humidity windy play
13 overcast         hot   normal    no  yes
14    rainy        mild     high   yes   no
```

View summary statistics. E.g.:

```
summary(test) #command


#output
     outlook  temperature   humidity windy    play        id
 overcast:4   cool:4      high  :7   no :8   no :5   Length:14
 rainy   :5   hot :4      normal:7   yes:6   yes:9   Class :character
 sunny   :5   mild:6                                 Mode  :character
```

- View structure of data set (obs = number of rows). E.g.: (NOTE: obs means observation)

```
str(test) #command

#output
'data.frame':  14 obs. of  5 variables:
 $ outlook    : Factor w/ 3 levels "overcast","rainy",..: 3 3 1 2 2 2 1 3
3 2 ...
 $ temperature: Factor w/ 3 levels "cool","hot","mild": 2 2 2 3 1 1 1 3 1
3 ...
 $ humidity   : Factor w/ 2 levels "high","normal": 1 1 1 1 2 2 2 1 2 2 ..
.
 $ windy      : Factor w/ 2 levels "no","yes": 1 2 1 1 1 2 2 1 1 1 ...
 $ play       : Factor w/ 2 levels "no","yes": 1 1 2 2 2 1 2 1 2 2 ...
```

- Find out the ‘class’ of a variable or dataset. E.g.:

```
class(test)   #command

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

- View number of rows and column, or alternatively, the dimension of the dataset. E.g.:

```
nrow(test)  #command

#output
[1] 14

ncol(test)  #command

#output
[1] 5

dim(test)  #command

#output
[1] 14  5
```

- View the class of the dataset

```
class(test)  #command

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

Subsetting a Dataset

Aim: Sometimes, it may be useful to look at only some columns or some rows in a dataset/data-frame—this is called subsetting.

Let's create a simple data-frame to demonstrate basic subsetting and other **command** functions in R. One simple way to do this is to create each column of the data-frame separately then combine them into a dataframe later. Note the different kinds of data types for the columns/variables created, and beware that R is case-sensitive.

Examples: Note that comments appearing after the hash sign (#) will not be evaluated.

```
subject_id <- c(1:6) #integer
outlook <- as.factor(c("overcast", "rainy", "sunny", "overcast", "rainy",
"sunny"))  #factor
temperature <- as.factor(c("hot", "cool", "mild", "hot", "cool", "mild"))
humidity <- as.factor(c("high", "normal", "high", "normal", "high", "norma
l"))  #factor
windy <- as.factor(c("no", "yes", "no", "yes", "no", "yes")) #factor
play <- as.factor(c("no", "yes", "no", "yes", "no", "yes"))  #factor

#Combine all the columns created above into a single data frame
myframe <- data.frame(subject_id, outlook, temperature, humidity, windy, p
lay)

#Print myframe.
myframe
```

```
#Combine all the columns created above into a single data frame

data.frame(test) #command (#This command creates the complete data fr
ame. With the data frame, R offers you a great first step by allo
wing you to store your data in overviewable, rectangular grids. E
ach row of these grids corresponds to measurements or values of a
n instance, while each column is a vector containing data for a s
pecific variable).
```

Note, that the above command is redundant in that, read.csv already create s a data frame; this is why we have not assigned the return value to any variable.

Question: can you verify if the above statement is indeed true?

```
#output
    outlook temperature humidity windy play
1     sunny         hot     high    no   no
2     sunny         hot     high   yes   no
3  overcast         hot     high    no  yes
4     rainy        mild     high    no  yes
5     rainy        cool   normal    no  yes
6     rainy        cool   normal   yes   no
7  overcast        cool   normal   yes  yes
8     sunny        mild     high    no   no
9     sunny        cool   normal    no  yes
10    rainy        mild   normal    no  yes
11    sunny        mild   normal   yes  yes
12 overcast        mild     high   yes  yes
13 overcast         hot   normal    no  yes
14    rainy        mild     high   yes   no

head(test, 5)  #command  (view only the 5 first rows)

#output
    outlook temperature humidity windy play id
1    sunny         hot     high    no   no  1
2    sunny         hot     high   yes   no  2
3 overcast         hot     high    no  yes  3
4    rainy        mild     high    no  yes  4
5    rainy        cool   normal    no  yes  5

str(test)  #command     (#Note the class of each variable/column)

#output
'data.frame':  14 obs. of  6 variables:
 $ outlook    : Factor w/ 3 levels "overcast","rainy",..: 3 3 1 2 2 2 1 3
3 2 ...
 $ temperature: Factor w/ 3 levels "cool","hot","mild": 2 2 2 3 1 1 1 3 1
3 ...
 $ humidity   : Factor w/ 2 levels "high","normal": 1 1 1 1 2 2 2 1 2 2 ..
.
 $ windy      : Factor w/ 2 levels "no","yes": 1 2 1 1 1 2 2 1 1 1 ...
 $ play       : Factor w/ 2 levels "no","yes": 1 1 2 2 2 1 2 1 2 2 ...
 $ id         : chr  "1" "2" "3" "4" ...
```

To subset or extract only e.g., temperature (*t*) and humidity (*h*), we can use either the dollar sign ($) after the dataset, data, or use the square brackets []. The square brackets [] is used in this case to select column t*emperature* and *humidity*.

```
t2 <- test[, 2]; t2  #command

#output
 [1] hot  hot  hot  mild cool cool cool mild cool mild mild mild hot  mild
Levels: cool hot mild

h2 <- test[, 3]; h2  #command

#output
 [1] high   high   high   high   normal normal normal high   normal normal
normal high   normal
[14] high
Levels: high normal
```

Using Packages in R

There are many packages that make life so much easier when manipulating data in R. They need to be installed on your computer and loaded at the start of your R script before you can call the functions in them. We will introduce examples of a couple of useful packages later in this module.

For now, the command for installing packages is:

```
install.packages("name_of_package_case_sensitive")
```

The command for loading the package into the R working environment:

```
library(name_of_package_case_sensitive)
```

Note—there are no quotation marks when loading packages as compared to installing; you will get an error message otherwise.

Getting Help in R

There are various online tutorials and Q&A forums for getting help in R. Stackoverflow, Cran and Quick-R are some good examples. Within the R console, a question mark, ?, followed by the name of the function of interest will bring up the help menu for the function, e.g.

```
?head
```