R literacy for digital soil mapping. Part 3

Soil Security Laboratory

2017

1 Data frames, data import, and data export

As described above, a data frame is a type of data structure in R with rows and columns, where different columns contain data with different modes. A data frame is probably the most common data structure that you will use for storing soil information data sets. Recall from before the data frame that we created.

```
dat <- data.frame(profile_id = c("Chromosol", "Vertosol", "Sodosol"),
FID = c("a1", "a10", "a11"),
easting = c(337859, 344059, 347034),
northing = c(6372415, 6376715, 6372740),
visted = c(TRUE, FALSE, TRUE))
dat
```

```
##   profile_id FID easting northing visted
## 1  Chromosol  a1  337859  6372415   TRUE
## 2   Vertosol a10  344059  6376715  FALSE
## 3    Sodosol a11  347034  6372740   TRUE
```

We can quickly assess the different modes of a data frame (or any other object for that matter) by using the **str**(structure) function.

```
str(dat)
```

```
## 'data.frame': 3 obs. of  5 variables:
##  $ profile_id: Factor w/ 3 levels "Chromosol","Sodosol",..: 1 3 2
##  $ FID       : Factor w/ 3 levels "a1","a10","a11": 1 2 3
##  $ easting   : num  337859 344059 347034
##  $ northing  : num  6372415 6376715 6372740
##  $ visted    : logi  TRUE FALSE TRUE
```

The **str** is probably one of the most used R functions. It is great for exploring the format and contents of any object created or imported.

1.1 Reading data from files

The easiest way to create a data frame is to read in data from a file—this is done using the function `read.table`, which works with ASCII text files. Data can be read in from other files as well, using different functions, but `read.table` is the most commonly used approach. `R` is very flexible in how it reads in data from text files.

Note that the column labels in the header have to be compatible with `R`'s variable naming convention, or else `R` will make some changes as they are read in (or will not read in the data correctly). So lets import some real soil data. These soil data are some chemical and physical properties from a collection of soil profiles sampled at various locations in New South Wales, Australia

```
## This will throw an error
soil.data <- read.table("USYD_soil1.txt", header = TRUE, sep = ",")
str(soil.data)
head(soil.data)
```

However, you may find that an error occurs, saying something like that the file does not exist. This is true as it has not been provided to you. Rather, to use this data you will need to load up the previously installed `ithir` package.

```
library(ithir)
data(USYD_soil1)
soil.data <- USYD_soil1
str(soil.data)
```

```
## 'data.frame': 166 obs. of  16 variables:
##  $ PROFILE     : int  1 1 1 1 1 1 2 2 2 2 ...
##  $ Landclass   : Factor w/ 4 levels "Cropping","Forest",..: 4 4 4 4 4 4 3 3 3 3 ...
##  $ Upper.Depth : num  0 0.02 0.05 0.1 0.2 0.7 0 0.02 0.05 0.1 ...
##  $ Lower.Depth : num  0.02 0.05 0.1 0.2 0.3 0.8 0.02 0.05 0.1 0.2 ...
##  $ clay        : int  8 8 8 8 NA 57 9 9 9 NA ...
##  $ silt        : int  9 9 10 10 10 8 10 10 10 10 ...
##  $ sand        : int  83 83 82 83 79 36 81 80 80 81 ...
##  $ pH_CaCl2    : num  6.35 6.34 4.76 4.51 4.64 6.49 5.91 5.94 5.63 4.22 ...
##  $ Total_Carbon: num  1.07 0.98 0.73 0.39 0.23 0.35 1.14 1.14 1.01 0.48 ...
##  $ EC          : num  0.168 0.137 0.072 0.034 NA 0.059 0.123 0.101 0.026 0.042 ...
##  $ ESP         : num  0.3 0.5 0.9 0.2 0.9 0.3 0.3 0.6 NA NA ...
##  $ ExchNa      : num  0.01 0.02 0.02 0 0.02 0.04 0.01 0.02 NA NA ...
##  $ ExchK       : num  0.71 0.47 0.52 0.38 0.43 0.46 0.7 0.56 NA NA ...
##  $ ExchCa      : num  3.17 3.5 1.34 1.03 1.5 9.13 2.92 3.2 NA NA ...
##  $ ExchMg      : num  0.59 0.6 0.22 0.22 0.5 5.02 0.51 0.5 NA NA ...
##  $ CEC         : num  5.29 3.7 2.86 2.92 2.6 ...
```

```
head(soil.data)
```

```
##   PROFILE      Landclass Upper.Depth Lower.Depth clay silt sand pH_CaCl2
## 1       1 native pasture        0.00        0.02    8    9   83     6.35
## 2       1 native pasture        0.02        0.05    8    9   83     6.34
## 3       1 native pasture        0.05        0.10    8   10   82     4.76
```

```
## 4       1 native pasture        0.10        0.20    8   10   83     4.51
## 5       1 native pasture        0.20        0.30   NA   10   79     4.64
## 6       1 native pasture        0.70        0.80   57    8   36     6.49
##   Total_Carbon    EC ESP ExchNa ExchK ExchCa ExchMg   CEC
## 1         1.07 0.168 0.3   0.01  0.71   3.17   0.59  5.29
## 2         0.98 0.137 0.5   0.02  0.47   3.50   0.60  3.70
## 3         0.73 0.072 0.9   0.02  0.52   1.34   0.22  2.86
## 4         0.39 0.034 0.2   0.00  0.38   1.03   0.22  2.92
## 5         0.23    NA 0.9   0.02  0.43   1.50   0.50  2.60
## 6         0.35 0.059 0.3   0.04  0.46   9.13   5.02 14.96
```

When we import a file into `R`, it is good habit to look at its structure (`str`) to ensure the data is as it should be. As can be seen, this data set (frame) has 166 observations, and 15 columns. The `head` function is also a useful exploratory function, which simply allows us to print out the data frame, but only the first 6 rows of it (good for checking data frame integrity). Note that you must specify `header=TRUE`, or else `R` will interpret the row of labels as data. If the file you are loading is not in the directory that `R` is working in (the working directory, which can be checked with `getwd()` and changed with `setwd(file = ‘‘filename’’)`). When setting the working directory (`setwd()`), you can include the file path, but note that the path should have forward, not backward slashes (or double backward slashes, if you prefer).

The column separator function `sep` (an argument of `read.table`) lets you tell `R`, where the column breaks or delimiters occur. In the `soil.data` object, we specify that the data is comma separated. If you do not specify a field separator, `R` assumes that any spaces or tabs separate the data in your text file. However, any character data that contain spaces must be surrounded by quotes (otherwise, `R` interprets the data on either side of the white spaces as different elements).

Besides comma separators, tab separators are also common `sep=‘‘\t’’`, so is `sep=‘‘.’’` (full stop separator). Two consecutive separators will be interpreted as a missing value. Conversely, with the default options, you need to explicitly identify missing values in your data file with `NA` (or any other character, as long as you tell `R` what it is with the `na.strings` argument).

For some field separators, there are alternate functions that can be used with the default arguments, e.g. `read.csv`, which is identical to `read.table`, except default arguments differ. Also, `R` does not care what the name of your file is, or what its extension is, as long as it is an ASCII text file. A few other handy bits of information for using `read.table` follow. You can include comments at the end of rows in your data file—just precede them with a `#`. Also `R` will recognize `NaN, Inf`, and `-Inf` in input files.

Probably the easiest approach to handling missing values is to indicate their presence with `NA` in the text file. `R` will automatically recognize these as missing values. Alternatively, just leave the missing values as they are (blank spaces), and let `R` do the rest.

```
which(is.na(soil.data$CEC))
```

```
## [1]   9  10  45  63 115
```

```
soil.data[8:11, ]
```

```
##    PROFILE        Landclass Upper.Depth Lower.Depth clay silt sand
## 8        2 improved pasture        0.02        0.05    9   10   80
## 9        2 improved pasture        0.05        0.10    9   10   80
## 10       2 improved pasture        0.10        0.20   NA   10   81
## 11       2 improved pasture        0.38        0.50   36    8   56
##    pH_CaCl2 Total_Carbon    EC ESP ExchNa ExchK ExchCa ExchMg CEC
## 8      5.94         1.14 0.101 0.6   0.02  0.56    3.2    0.5 4.0
## 9      5.63         1.01 0.026  NA     NA    NA     NA     NA  NA
## 10     4.22         0.48 0.042  NA     NA    NA     NA     NA  NA
## 11     6.48         0.18 0.053 0.4   0.04  0.49    6.0    2.3 8.6
```

In most cases, it makes sense to put your data into a text file for reading into R. This can be done in various ways. Data download from the internet are often in text files to begin with. Data can be entered directly into a text file using a text editor. For data that are in spreadsheet program such as Excel or JMP, there are facilities available for saving these tabular frames to text files for reading into R.

This all may seem confusing, but it is really not that bad. Your best bet is to play around with the different options, find one that you like, and stick with it. Lastly, data frames can also be edited interactively in R using the `edit` function. This is really only useful for small data sets, and the function is not supported by RStudio (you could try with using Tinn-R instead if you want to explore using this function)

```
soil.data <- edit(soil.data)
```

1.2 Creating data frames manually

Data frames can be made manually using the `data.frame` function:

```
soil <- c("Chromosol", "Vertosol", "Organosol", "Anthroposol")
carbon <- c(2.1, 2.9, 5.5, 0.2)
dat <- data.frame(soil.type = soil, soil.OC = carbon)
dat
```

```
##     soil.type soil.OC
## 1   Chromosol     2.1
## 2    Vertosol     2.9
## 3   Organosol     5.5
## 4 Anthroposol     0.2
```

While this approach is not an efficient way to enter data that could be read in directly, it can be very handy for some applications, such as the creation of customized summary tables. Note that column names are specified using an equal sign. It is also possible to specify (or change, or check) column names for

an existing data frame using the function `names`.

```
names(dat) <- c("soil", "SOC")
dat
```

```
##          soil SOC
## 1   Chromosol 2.1
## 2    Vertosol 2.9
## 3   Organosol 5.5
## 4 Anthroposol 0.2
```

Row names can be specified in the `data.frame` function with the `row.names` argument.

```
dat <- data.frame(soil.type = soil, soil.OC = carbon,
row.names = c("Ch", "Ve", "Or", "An"))
dat
```

```
##      soil.type soil.OC
## Ch   Chromosol     2.1
## Ve    Vertosol     2.9
## Or   Organosol     5.5
## An Anthroposol     0.2
```

Specifying row names can be useful if you want to index data, which will be covered later. Row names can also be specified for an existing data frame with the `rownames` function (not to be confused with the `row.names` argument).

1.3 Working with data frames

So what do you do with data in `R` once it is in a data frame? Commonly, the data in a data frame will be used in some type of analysis or plotting procedure. It is usually necessary to be able to select and identify specific columns (i.e. vectors) within data frames. There are two ways to specify a given column of data within a data frame. The first is to use the `$` notation. To see what the column names are, we can use the `names` function. Using our `soil.data` set:

```
names(soil.data)
```

```
##  [1] "PROFILE"      "Landclass"    "Upper.Depth"  "Lower.Depth"
##  [5] "clay"         "silt"         "sand"         "pH_CaCl2"
##  [9] "Total_Carbon" "EC"           "ESP"          "ExchNa"
## [13] "ExchK"        "ExchCa"       "ExchMg"       "CEC"
```

The `$` just uses a `$` between the data frame name and column name to specify a particular column. Say we want to look at the `ESP` column, which is the acronym for exchangeable sodium percentage.

```
soil.data$ESP
```

```
##   [1]  0.3  0.5  0.9  0.2  0.9  0.3  0.3  0.6   NA   NA  0.4  0.9  0.2  0.1
```

```
##  [15]   NA  0.4  0.5  0.7  0.2  0.1   NA  0.2  0.3   NA  0.8  0.6  0.8  0.9
##  [29]  0.9  1.1  0.5  0.6  1.1  0.2  0.6  1.1  0.4  0.3  0.5  1.0  2.2  0.1
##  [43]  0.1  0.1   NA  0.1  0.1  0.4   NA  0.2  0.1  0.3  0.4  0.1  0.4  0.1
##  [57]  0.1   NA  0.1  0.3   NA  0.1   NA  0.2  1.8  2.6  0.2 13.0  0.0  0.1
##  [71]  0.3  0.1  0.1  0.3  0.0  0.3  0.6  0.9  0.4   NA   NA  2.4  0.2  0.3
##  [85]  0.2  0.0  0.1  0.4   NA  0.3  0.3  0.2  0.3  0.6  0.3  0.2  0.7  0.3
##  [99]  0.4  1.0  7.9  6.1  5.7  5.2  4.7  2.9  5.8  7.2  9.6   NA 17.4   NA
## [113] 11.1  6.4   NA  4.0 12.1 21.2  2.2  1.9   NA  4.0 13.2  0.9  0.8  0.5
## [127]  0.2  0.4   NA  1.2  0.6  0.2  1.0  0.4  0.6  0.1  0.4   NA  0.7  0.5
## [141]  0.7  0.9  4.8  3.8  4.9  6.2 10.4 16.4  2.7   NA  1.2  0.5  1.9  2.0
## [155]  2.1  1.9  1.8  3.5  7.7  2.7  1.8  0.8  0.5  0.5  0.3  0.9
```

Although it is handy to think of data frame columns to have a vertical orientation, this orientation is not present when they are printed individually—instead, elements are printed from left to right, and then top to bottom. The expression `soil.data$ESP` could be used just as you would for any other vector. For example:

```
mean(soil.data$ESP)
```

```
## [1] NA
```

`R` can not calculate the mean because of the `NA` values in the vector. Lets remove them first using the `na.omit` function.

```
mean(na.omit(soil.data$ESP))
```

```
## [1] 1.99863
```

The second option for working with individual columns within a data frame is to use the commands `attach` and `detach`. Both of these functions take a data frame as an argument. `attach`ing a data frame puts all the columns within the that data frame into `R`'s search path, and they can be called by using their names alone without the `$` notation.

```
attach(soil.data)
ESP
```

```
##   [1]  0.3  0.5  0.9  0.2  0.9  0.3  0.3  0.6   NA   NA  0.4  0.9  0.2  0.1
##  [15]   NA  0.4  0.5  0.7  0.2  0.1   NA  0.2  0.3   NA  0.8  0.6  0.8  0.9
##  [29]  0.9  1.1  0.5  0.6  1.1  0.2  0.6  1.1  0.4  0.3  0.5  1.0  2.2  0.1
##  [43]  0.1  0.1   NA  0.1  0.1  0.4   NA  0.2  0.1  0.3  0.4  0.1  0.4  0.1
##  [57]  0.1   NA  0.1  0.3   NA  0.1   NA  0.2  1.8  2.6  0.2 13.0  0.0  0.1
##  [71]  0.3  0.1  0.1  0.3  0.0  0.3  0.6  0.9  0.4   NA   NA  2.4  0.2  0.3
##  [85]  0.2  0.0  0.1  0.4   NA  0.3  0.3  0.2  0.3  0.6  0.3  0.2  0.7  0.3
##  [99]  0.4  1.0  7.9  6.1  5.7  5.2  4.7  2.9  5.8  7.2  9.6   NA 17.4   NA
## [113] 11.1  6.4   NA  4.0 12.1 21.2  2.2  1.9   NA  4.0 13.2  0.9  0.8  0.5
## [127]  0.2  0.4   NA  1.2  0.6  0.2  1.0  0.4  0.6  0.1  0.4   NA  0.7  0.5
## [141]  0.7  0.9  4.8  3.8  4.9  6.2 10.4 16.4  2.7   NA  1.2  0.5  1.9  2.0
## [155]  2.1  1.9  1.8  3.5  7.7  2.7  1.8  0.8  0.5  0.5  0.3  0.9
```

Note that when you are done using the individual columns, it is good practice to `detach` your data frame. Once the data frame is `detach`ed, `R` will no longer

know what you mean when you specify the name of the column alone:

```
## This will throw an error
detach(soil.data)
ESP
```

If you modify a variable that is part of an attached data frame, the data within the data frame remain unchanged; you are actually working with a copy of the data frame.

Another option (for selecting particular columns) is to use the square braces `[]` to specify the column you want. Using the square braces to select the `ESP` column from our data set you would use:

```
soil.data[, 10]
```

Here you are specifying the column in the tenth position, which as you should check is the `ESP` column. To use the square braces the row position precedes to comma, and the column position proceeds to comma. By leaving a blank space in front of the comma, we are essentially instruction `R` to print out the whole column. You may be able to surmise that it is also possible to subset a selection of columns quite efficiently with this square brace method. We will use the square braces more a little later on.

The `$` notation can also be used to add columns to a data frame. For example, if we want to express our `Upper.Depth` and `Lower.Depth` columns in *cm* rather than *m* we could do the following.

```
soil.data$Upper <- soil.data$Upper.Depth * 100
soil.data$Lower <- soil.data$Lower.Depth * 100
head(soil.data)
```

```
##   PROFILE      Landclass Upper.Depth Lower.Depth clay silt sand pH_CaCl2
## 1       1 native pasture        0.00        0.02    8    9   83     6.35
## 2       1 native pasture        0.02        0.05    8    9   83     6.34
## 3       1 native pasture        0.05        0.10    8   10   82     4.76
## 4       1 native pasture        0.10        0.20    8   10   83     4.51
## 5       1 native pasture        0.20        0.30   NA   10   79     4.64
## 6       1 native pasture        0.70        0.80   57    8   36     6.49
##   Total_Carbon    EC ESP ExchNa ExchK ExchCa ExchMg   CEC Upper Lower
## 1         1.07 0.168 0.3   0.01  0.71   3.17   0.59  5.29     0     2
## 2         0.98 0.137 0.5   0.02  0.47   3.50   0.60  3.70     2     5
## 3         0.73 0.072 0.9   0.02  0.52   1.34   0.22  2.86     5    10
## 4         0.39 0.034 0.2   0.00  0.38   1.03   0.22  2.92    10    20
## 5         0.23    NA 0.9   0.02  0.43   1.50   0.50  2.60    20    30
## 6         0.35 0.059 0.3   0.04  0.46   9.13   5.02 14.96    70    80
```

Many data frames that contain real data will have some missing observations. `R` has several tools for working with these observations. For starters, the `na.omit` function can be used for removing `NA`s from a vector. Working again with the `ESP` column of our `soil.data` set:

```
soil.data$ESP
```

```
##   [1]  0.3  0.5  0.9  0.2  0.9  0.3  0.3  0.6   NA   NA  0.4  0.9  0.2  0.1
```

```
##  [15]   NA  0.4  0.5  0.7  0.2  0.1   NA  0.2  0.3   NA  0.8  0.6  0.8  0.9
##  [29]  0.9  1.1  0.5  0.6  1.1  0.2  0.6  1.1  0.4  0.3  0.5  1.0  2.2  0.1
##  [43]  0.1  0.1   NA  0.1  0.1  0.4   NA  0.2  0.1  0.3  0.4  0.1  0.4  0.1
##  [57]  0.1   NA  0.1  0.3   NA  0.1   NA  0.2  1.8  2.6  0.2 13.0  0.0  0.1
##  [71]  0.3  0.1  0.1  0.3  0.0  0.3  0.6  0.9  0.4   NA   NA  2.4  0.2  0.3
##  [85]  0.2  0.0  0.1  0.4   NA  0.3  0.3  0.2  0.3  0.6  0.3  0.2  0.7  0.3
##  [99]  0.4  1.0  7.9  6.1  5.7  5.2  4.7  2.9  5.8  7.2  9.6   NA 17.4   NA
## [113] 11.1  6.4   NA  4.0 12.1 21.2  2.2  1.9   NA  4.0 13.2  0.9  0.8  0.5
## [127]  0.2  0.4   NA  1.2  0.6  0.2  1.0  0.4  0.6  0.1  0.4   NA  0.7  0.5
## [141]  0.7  0.9  4.8  3.8  4.9  6.2 10.4 16.4  2.7   NA  1.2  0.5  1.9  2.0
## [155]  2.1  1.9  1.8  3.5  7.7  2.7  1.8  0.8  0.5  0.5  0.3  0.9
```

```
na.omit(soil.data$ESP)
```

```
##   [1]  0.3  0.5  0.9  0.2  0.9  0.3  0.3  0.6  0.4  0.9  0.2  0.1  0.4  0.5
##  [15]  0.7  0.2  0.1  0.2  0.3  0.8  0.6  0.8  0.9  0.9  1.1  0.5  0.6  1.1
##  [29]  0.2  0.6  1.1  0.4  0.3  0.5  1.0  2.2  0.1  0.1  0.1  0.1  0.1  0.4
##  [43]  0.2  0.1  0.3  0.4  0.1  0.4  0.1  0.1  0.1  0.3  0.1  0.2  1.8  2.6
##  [57]  0.2 13.0  0.0  0.1  0.3  0.1  0.1  0.3  0.0  0.3  0.6  0.9  0.4  2.4
##  [71]  0.2  0.3  0.2  0.0  0.1  0.4  0.3  0.3  0.2  0.3  0.6  0.3  0.2  0.7
##  [85]  0.3  0.4  1.0  7.9  6.1  5.7  5.2  4.7  2.9  5.8  7.2  9.6 17.4 11.1
##  [99]  6.4  4.0 12.1 21.2  2.2  1.9  4.0 13.2  0.9  0.8  0.5  0.2  0.4  1.2
## [113]  0.6  0.2  1.0  0.4  0.6  0.1  0.4  0.7  0.5  0.7  0.9  4.8  3.8  4.9
## [127]  6.2 10.4 16.4  2.7  1.2  0.5  1.9  2.0  2.1  1.9  1.8  3.5  7.7  2.7
## [141]  1.8  0.8  0.5  0.5  0.3  0.9
## attr(,"na.action")
##  [1]   9  10  15  21  24  45  49  58  61  63  80  81  89 110 112 115 121
## [18] 129 138 150
## attr(,"class")
## [1] "omit"
```

Although the result does contain more than just the non-`NA` values, only the non-`NA` values will be used in subsequent operations. Note that the result of `na.omit` contains more information than just the non-`NA` values. This function can also be applied to complete data frames. In this case, any row with an `NA` is removed (so be careful with its usage).

```
soil.data.cleaned <- na.omit(soil.data)
```

It is often necessary to identify `NA`s present in a data structure. The `is.na` function can be used for this—it can also be negated using the "!" character.

```
is.na(soil.data$ESP)
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE
##  [12] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
##  [23] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [34] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [45]  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
##  [56] FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE
##  [67] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
##  [78] FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [89]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [100] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
## [111] FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE
## [122] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
## [133] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
## [144] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
## [155] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [166] FALSE
```

1.4 Writing data to files

With `R`, it is easy to write data to files. The function `write.table` is usually the best function for this purpose. Given only a data frame and a file name, this function will write the data contained in the data frame to a text file. There are a number of arguments that can be controlled with this function as shown below (also look at the help file).

write.table(x, file = ””, append = FALSE, quote = TRUE, sep = ” ”, eol = ”\”, na = ”NA”, dec = ”.”, row.names = TRUE, col.names = TRUE, qmethod = c(”escape”, ”double”), fileEncoding = ””)

The important ones (or most frequently used) are the column separator `sep` argument, and whether of not you want to keep the column and row names (`col.names` and `row.names` respectively). For example, if we want to write `soil.data` to text file (“file name.txt”), retaining the column names, not retaining row names, and having a tab delimited column separator, we would use:

```
write.table(soil.data, file = "file name.txt", col.names = TRUE,
row.names = FALSE, sep = "\t")
```

Setting the `append` argument to `TRUE` lets you add data to a file that already exists.

The `write.table` function can not be used with all data structures in `R` (like lists for example). However, it can be used for such things as vectors and matrices.

1.5 Exercises

1. Using the `soil.data` object, determine the minimum and maximum soil pH (`PH_CaCl2`) in the data frame. Next add a new column to the dataframe that contains the log_{10} of soil carbon `Total_Carbon`.
2. Create a new data frame that contains the mean SOC, pH, and clay of the data set. Write out the summary to a new file using the default options. Finally, try changing the separator to a tab and write to a new file.
3. There are a number of `NA` values in the data set. We want to remove

them. Could this be done in one step i.e. delete every row that contains an `NA`? Is this appropriate? How would you go about ensuring that no data is lost? Can you do this? or perhaps—do this!