R16

Importing data

Covered in R16

- Overview
- Exporting vector data to windows clipboard
- Exporting data to a text file
- Using write.table with data frames
- Using write.csv
- Other export commands

1 Overview

In this chapter we'll look at how to export data to text, csv or formats which can be used in other programs to, for example, produce reports.

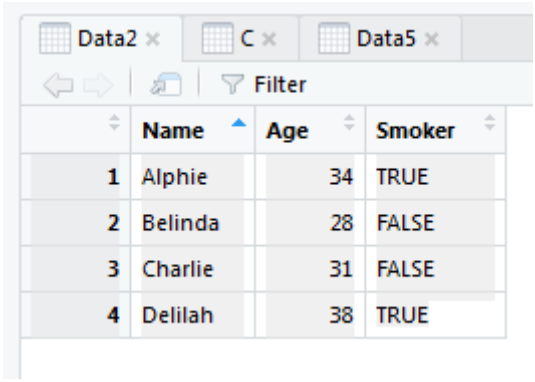
Exporting data to other programs

There are a variety of places that we can export data to. These include:

- windows clipboard
- a text document
- a CSV file
- Excel
- other statistical packages such as SAS or SPSS.

2 Exporting vector data to windows clipboard

If you only have a small amount data, it's possible to view the data, highlight it and just copy and paste it where you want it:


Data2 C Data5

Filter

	Name	Age	Smoker
1	Alphie	34	TRUE
2	Belinda	28	FALSE
3	Charlie	31	FALSE
4	Delilah	38	TRUE

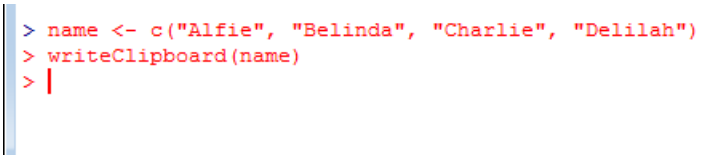
You can also copy data to the windows clipboard using the writeClipboard function. We can then paste this into any other program such as Notepad, Word or Excel. However, it only works well with vectors of character data.

writeClipboard(<character object>)

Let's create a quick character vector of policyholder's names:

```
name <- c("Alfie", "Belinda", "Charlie", "Delilah")
```

Then we'll use the writeClipboard command on object "name" to copy the contents of "name" into Windows clipboard:



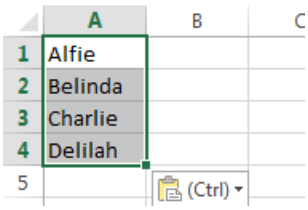
Now if we open Word or Excel and paste it using CTRL+V or Edit/Paste we find that we have the column of data from "name":


Alfie¶

Belinda¶

Charlie¶

Delilah¶



The writeClipboard command only works with character data. Let's look at what happens if we try to apply it to a numeric vector.

Let's create a vector of policyholder's ages:

```
age <- c(34, 28, 31, 38)
```

Then we'll use the writeClipboard command on object "age" to copy the contents of "age" into Windows clipboard:

```
> age <- c(34, 28, 31, 38)
> writeClipboard(age)
Error in writeClipboard(age) :
  argument must be a character vector or a raw vector
>
```

R returns an error telling us it's not a character vector and so it can't do it.

We can fix this by using the as.character command. This coerces the data type into character data:

```
> as.character(age)
[1] "34" "28" "31" "38"
>
```

We could put this in a new object and then apply writeClipboard on this new object. For example:

```
age2 <- as.character(age)
writeClipboard(age2)
```

Alternatively, we could just use as.character inside the writeClipboard function:

```
> writeClipboard(as.character(age))
>
```

We are now able to paste it in Word or Excel.

Incidentally, if our vectors have named elements they would not be included in the clipboard.

3 Exporting data to a text file

To write (*ie* export) data to a text file (*.txt) or CSV file (*.csv) we use the write.table( ) command. This function is particularly suited to exporting two dimensional objects. In this section we'll look at exporting it to a text file.

write.table(<object>,file="<filename>")

If the filename does not specify the file path then R will save the file in the current working directory.

Recall that you can find out the current working directory using the getwd( ) and you can change it using the setwd( ) command or the menu option Session/Set Working Directory.

Also remember that we can use the tilde, ~, shortcut to specify the location. You can find out what the shortcut is on your computer by using path.expand("~").

Using write.table to export vectors to text file

Let's use this command to save the "name" vector with the name "name":

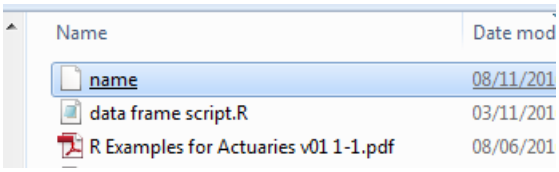
```
write.table(name, file="name")
```

or since file is the second argument we could omit the file= as long as we put it in the second position:

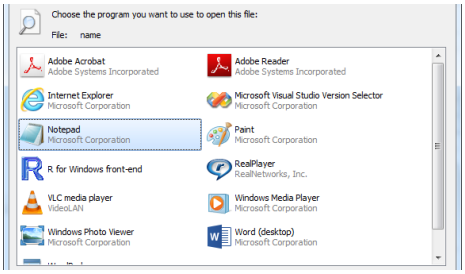
```
write.table(name, "name")
```

```
> write.table(name,"name")
>
```

If we look inside our working directory, we'll find the file:



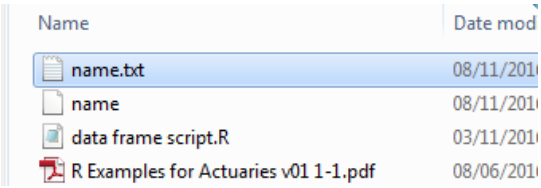
However, because we didn't specify the file type Windows doesn't have a clue what it is. We can still open it but we'll have to tell windows which program to use:



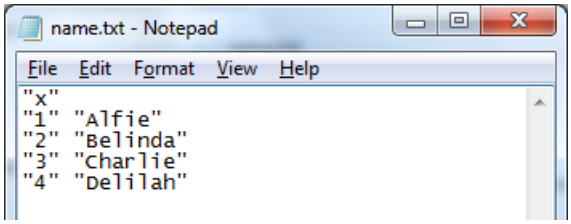
Since this is a bit annoying let's save the file correctly this time by adding the extension .txt:

```
> write.table(name,"name.txt")
> 
```

When we look in our working directory we can see this file correctly identified as a text file:



Clicking on it will open it in Notepad:



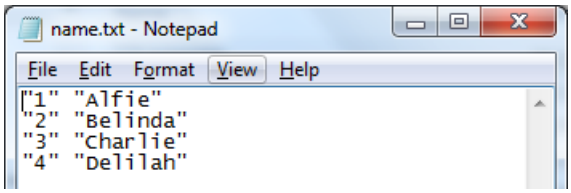
There are two odd things about our text file. First, we notice that there is an "x" at the top. This is because by default write.table adds column headings. Since our vector hasn't got a column heading it calls it "x". Secondly, it has added the row names "1", "2", "3" and "4". Had we named the elements in our vector these would have appeared here instead of the default row names.

Both of these features can be turned off by using the optional arguments col.names and row.names. By default both of these options are set to TRUE.

Let's experiment to see what happens if we try the logical options FALSE or NA. First, let's set col.names to FALSE:

```
write.table(name, "name.txt", col.names=FALSE)
```

Hopefully you are shocked that R has just overwritten our previous file without even a warning. Hopefully you'll be careful about saving files from now on. Once you've recovered from the shock and opened this file in Wordpad or Notepad you'll see the following:

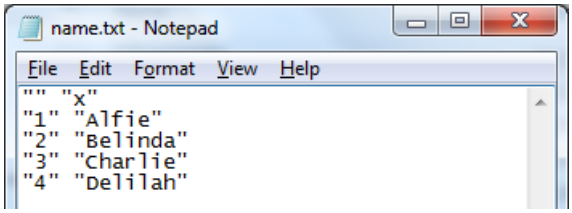


We can see that it has removed the default column name of "x".

Next let's see what happens when we set col.names to NA:

write.table(name, "name.txt", col.names=NA)

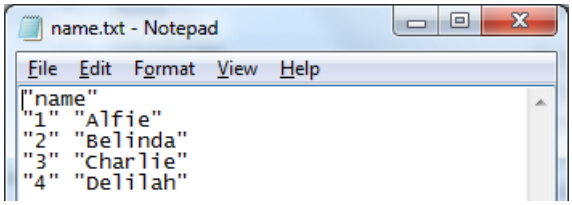
Opening this file in Wordpad reveals the following:



We can see that it puts in the default column name "x" but it also adds a blank column name for the row names. This is a useful output if we were going to paste this into, say, Excel as it ensures that everything lines up correctly in a grid.

Finally, suppose we want to call the column "name" rather than the default "x". we simply set the col.names option to this:

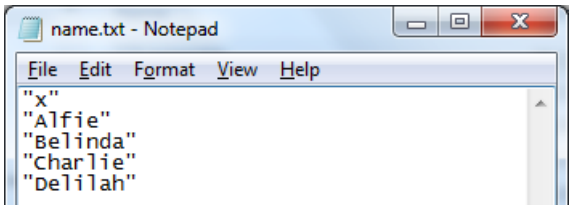
write.table(name, "name.txt", col.names="name")



Next let's see what happens when we set row.names to FALSE:

write.table(name, "name.txt", row.names=FALSE)

Opening this file in Wordpad reveals the following:



We can see that, as expected, it has removed the row names "1", "2", "3" and "4".

Let's see what happens when we set row.names to NA:

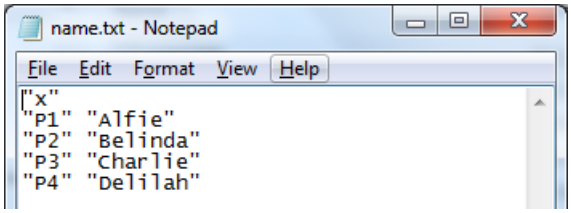
write.table(name, "name.txt", row.names=NA)

```
> write.table(name,"name.txt", row.names=NA)
Error in if (row.names) { : missing value where TRUE/FALSE needed
>
```

We get an error. Clearly NA is not an option here.

Suppose we want to name the rows “P1”, “P2”, “P3” and “P4” rather than the default “1”, “2”, “3” and “4”. We simply set the row.names option to equal this:

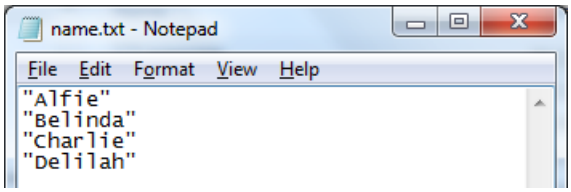
```
write.table(name, "name.txt", row.names=c("P1","P2","P3","P4"))
```



For a vector we will probably not actually want either the row or the column names. In which case we would type the following:

```
write.table(name, "name.txt", row.names=FALSE, col.names=FALSE)
```

This gives:



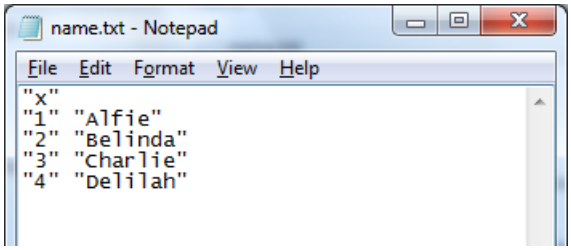
Before we move on to applying this function to matrices and data.frames let’s show a couple of other optional arguments.

The quote option

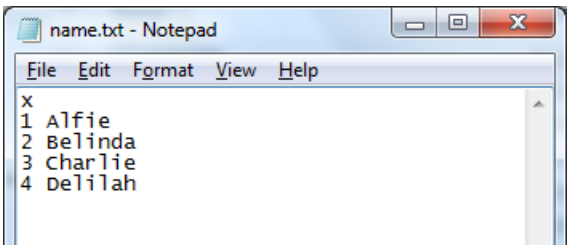
The first is quote, which refers to whether the character or factor data have double quote marks around them. It can take the logical values of TRUE or FALSE and by default this is set to TRUE.

Let’s set it to FALSE to see what happens:

```
write.table(name, "name.txt", quote=FALSE)
```



default (quote=TRUE)



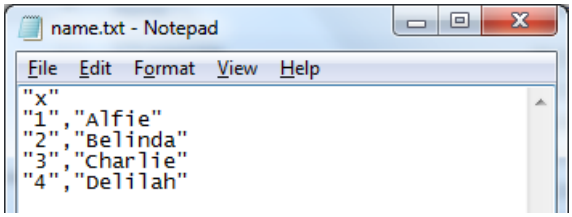
quote=FALSE

The sep option

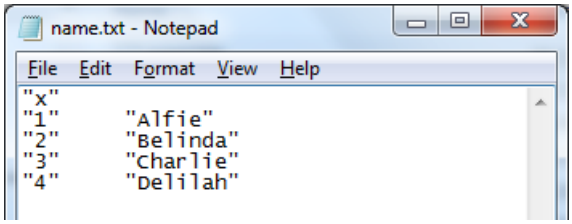
Just as with the read.table function we can specify the separator that goes between different entries in each row. By default this is set to a single space " ". Other options could include a comma "," a semi-colon ";" or a tab "\t".

Here are examples of a comma and a tab separator:

write.table(name, "name.txt", sep=",")



write.table(name, "name.txt", sep="\t")



4 Using write.table with data frames

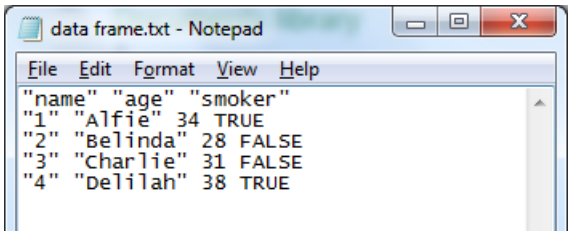
First let's create a data frame, A:

```
> name <- c("Alfie", "Belinda", "Charlie", "Delilah")
> age <- c(34, 28, 31, 38)
> smoker <- c(TRUE, FALSE, FALSE, TRUE)
> A <- data.frame(name,age,smoker)
> A
     name age smoker
1   Alfie  34   TRUE
2 Belinda  28  FALSE
3 Charlie  31  FALSE
4 Delilah  38   TRUE
>
```

Let's now export this using the following:

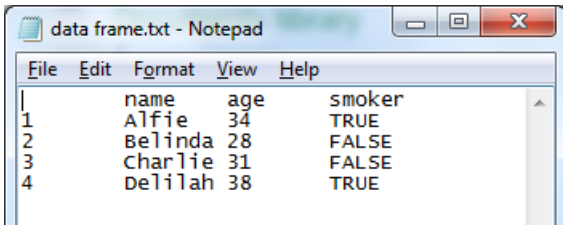
write.table(A, "data frame.txt")

This gives the following:



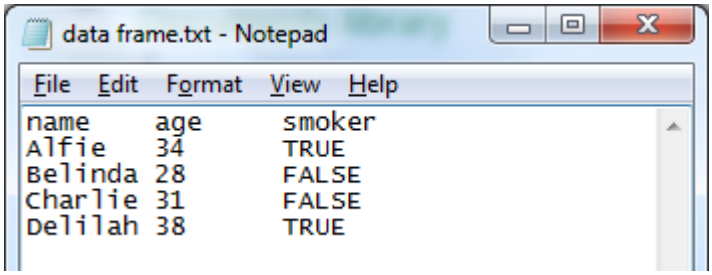
By now, we know how to get rid of the quote marks, set col.names to NA and separators to tabs to make it all look pretty as follows:

write.table(A, "data frame.txt", quote=FALSE, sep="\t", col.names=NA)



Or we could get rid of the row names, in which case it would be pointless setting col.names to NA (and you'll get a lovely error message from R if you try):

write.table(A, "data frame.txt", quote=FALSE, sep="\t", row.names=FALSE)



5 Using write.csv

Exporting a vector to a csv file

In the previous chapter we said that CSV stands for Comma Separated Value and is to Excel what NotePad is to Word. It is a stripped down excel file that removes all the formatting and separates the values in the cells with commas.

To write (*ie* export) data to a csv file (*.csv) we use the write.csv( ) command:

write.csv(<object>, file="<filename>")

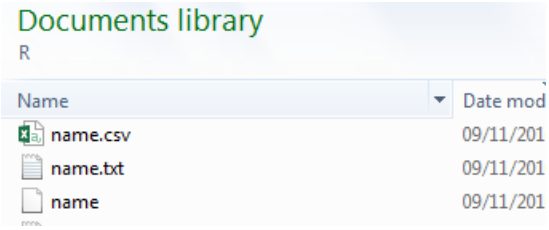
The write.csv function is actually the same as the write.table command, it just has fixed defaults for some of the optional arguments to ensure that it is properly formatted as a csv file that can be read by any spreadsheet program such as Excel.

These defaults include commas for the separator (which tells spreadsheets to put the value in the next column). It always has a header row (*ie* column names) and will assign default ones if none are specified. If row.names is TRUE then it sets col.names=NA so that everything lines up beautifully as we saw earlier.

Let's apply this to our vector name.

```
write.csv(name, "name.csv")
```

Because we gave the file extension .csv we can see in our working directory that Windows recognises it as a spreadsheet file:



If we click on this file it opens in Excel to give:

	A	B	C	D	E	F	G
1		x					
2	1	Alfie					
3	2	Belinda					
4	3	Charlie					
5	4	Delilah					

We can see that we have the default row names "1", "2", "3" and "4" and the default vector column name "x".

We can remove the row names by setting row.names to FALSE:

```
write.csv(name, "name.csv", row.names=FALSE)
```

If you get an error message it's because we are saving a file called "name.csv" whilst that file is open in Excel. Simply close the Excel file and re-execute the command and you'll get the following:

	A	B	C	D
1	x			
2	Alfie			
3	Belinda			
4	Charlie			
5	Delilah			

However if we try to get rid of the column name as well:

```
write.csv(name, "name.csv", row.names=FALSE, col.names=FALSE)
```

We get a warning that R ignores this setting:

```
> write.csv(name, "name.csv", row.names=FALSE, col.names=FALSE)
Warning message:
In write.csv(name, "name.csv", row.names = FALSE, col.names = FALSE) :
  attempt to set 'col.names' ignored
>
```

This is because the write.csv *always* has a header row to match the convention. That's why when we used read.csv in the previous chapter it always assumed the file had a header row.

Similarly if you tried to change the separator to something other than a comma you'd get an error too.

Exporting a data frame to a csv file

Finally let's write our data frame A from earlier to a csv file:

```
write.csv(A, "data frame.csv")
```

	A	B	C	D	E	F	G
1		name	age	smoker			
2	1	Alfie	34	TRUE			
3	2	Belinda	28	FALSE			
4	3	Charlie	31	FALSE			
5	4	Delilah	38	TRUE			

Again, everything is beautifully lined up with the row and column names.

If we had no row or column names, the function would automatically assign them as we have seen.

6 Other export commands

Exporting data to Excel files

We have already been able to export data from R into Excel by saving it as a csv file. It is possible to export it directly to an xlsx Excel file by using the package "openxlsx".

You can then use the new command:

write.xlsx(<object>, "<filename>", sheetname= …)

to export the object to an xlsx file. Additionally you can even specify the formatting and other features.

Exporting data to other statistical packages

It is possible to export data to other statistical packages such as SAS, SPSS or Stata by using the package "foreign". You can then use the new command:

write.foreign(<data object>, "<data filename>", "<code filename>",
package="<package name>")

where package="SAS" or package="SPSS" or package="Stata".

7 Summary

Key terms

Export	Move data from R into another format, *eg* a text or Excel file

Key commands

writeClipboard(<character object>)

Copies data stored in a character vector to the clipboard

write.table(<object>,file="<filename>")

Exports data into a text file. Optional arguments include row.names, col.names, quote and sep

write.csv(<object>,file="<filename>")

Exports data into a csv file

write.xlsx(<object>, "<filename>", sheetname= …)

Exports data into an Excel file – needs the openxlsx package

write.foreign(<data object>, "<data filename>", "<code filename>", package="<package name>")

Exports data into an Excel file – needs the foreign package

There is no 'Have a go' section in this chapter.

