Working with Data in R

Command History

Hitting the up arrow multiple times will display the previous commands, which can then be edited. This is useful since one often wishes to repeat a similar command.

Matrices

A vector stores data in a 1-dimensional structure with a single index (item number). However R can also be used to work with matrices. Note that in R each item in either a vector and matrix objects must be of a single type, e.g. `int`, `numeric`, `character`.

$$\text{vector } v = [1, 2, 6, 3]$$

$$\text{matrix } m = \begin{bmatrix} 1 & 2 \\ 6 & 3 \end{bmatrix}$$

To create a matrix in R use the `matrix()` function. Before we use the `matrix()` function, we can learn more about it using the help functionality:

```
?matrix
```

The help file reveals that the `matrix()` function takes a number of inputs, but for now we focus on the first three: the data (the entries in the matrix), the number of rows, and the number of columns. First, we create a simple matrix.

```
x <- matrix(data=c(1,2,3,4), nrow=2, ncol=2)
x
```

```
     [ ,1] [ ,2]
[1,]    1     3
[2,]    2     4
```

Note that we could just as well omit typing `data=, nrow=, and ncol=` in the `matrix()` command above: that is, we could just type

```
x <- matrix(c(1,2,3,4) ,2,2)
```

and this would have the same effect. However, it can sometimes be useful to specify the names of the arguments passed in, since otherwise R will assume that the function arguments are passed into the function in the same order that is given in the function's help file. As this example illustrates, by default R creates matrices by successively filling in columns. Alternatively, the `byrow=TRUE` option can be used to populate the matrix in order of the rows.

```
matrix(c(1,2,3,4),2,2,byrow=TRUE)
```

```
     [ ,1] [ ,2]
[1,]    1     2
[2,]    3     4
```

Working with Matrices

Suppose that our data is stored in the matrix `A`.

```
A <- matrix(1:16,4,4)
A
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
```

The `dim()` function outputs the number of rows followed by the number of columns of a given matrix.

```
dim(A)
```

```
[1] 4 4
```

We can make a selection from a matrix just as we did for vectors, but must specify two values. For example:

```
A[2,3]
```

```
[1] 10
```

will select the element corresponding to the second row and the third column. The first number after the open-bracket symbol `[` always refers to the row, and the second number

always refers to the column. We can also select multiple rows and columns at a time, by providing vectors as the indices.

Try to work out what the following commands will select. Check your understanding by running them in R:

1. `A[c(1,3),c(2,4)]`
2. `A[1:3,2:4]`

The next examples include either no index for the columns or no index for the rows. These indicate that R should include all columns or all rows, respectively. R treats a single row or column of a matrix as a vector.

3. `A[1:2,]`
4. `A[,1:2]`
5. `A[1,]`

The use of a negative sign `-` in the index tells R to keep all rows or columns except those indicated in the index.

6. `A[-c(1,3),]`
7. `A[-c(1,3),-c(1,3,4)]`

Data frames

In general our dataset may contain a mixture of number, text and category variables. To be able to work with this R provides the *data frame* object class.

For example suppose we have the following data set of student grades:

name	student_id	grade	gender	teacher	age
Lauren	100213	78.6	F	Mr Smith	20
Marcelino	100214	83	M	Mr Smith	19
Ramona	100215	60.2	F	Mr Smith	18
Lonna	100216	65.6	F	Mr Smith	22

We see each column contains a single type of data, and so we could store them individually as vectors:

```
name <- c("Lauren", "Marcelino", "Ramona")
student_id <- c(100213, 100214, 100215)
grade <- c(78.6, 83, 60.2)
gender <- c("F", "M", "F")
teacher <- c("Mr Smith", "Ms Clarke", "Mr Smith")
age <- c(20, 19, 18)
```

We can then use these to build a data frame object:

```
grades_df <- data.frame(name, student_id, grade, gender, teacher, age)
```

Here we use the `data.frame()` function and pass the set of vectors that form the columns of the dataset. To view the data frame we have created we can use the print command:

```
print(grades_df)
```

```
       name student_id grade  gender   teacher age
1    Lauren     100213   78.6      F  Mr Smith  20
2 Marcelino     100214   83.0      M Ms Clarke  19
3    Ramona     100215   60.2      F  Mr Smith  18
```

Other useful functions to view data frame information are:

`head()`	prints only the first few rows (useful for big data sets.
`View()`	opens a view of the data frame in an RStudio panel (similar to a spreadsheet view).
`names()`	show the column names
`str()`	shows the structure of the data frame.

In this case let's look at the structure of the data frame we have created:

```
str(grades_df)
```

```
'data.frame':   3 obs. of  6 variables:
 $ name      : Factor w/ 3 levels "Lauren","Marcelino",..: 1 2 3
 $ student_id: num  1e+05 1e+05 1e+05
```

```
 $ grade     : num  78.6 83 60.2
 $ gender    : Factor w/ 2 levels "F","M": 1 2 1
 $ teacher   : Factor w/ 2 levels "Mr Smith","Ms Clarke": 1 2 1
 $ age       : num  20 19 18
```

We can see that R has created the columns with numerical data types for our numerical data, and Factor (categorical) data types from the text vectors. The category types are noted as the Factors **`levels`**.

Note in this case `name` is not a category that can be used to group students (unlike `gender` or `teacher`) so would be better stored as a `character` data type.

Its important to store data in appropriate data types, for example R will use this information to produce appropriate plots types when displaying data.

Data frames also have row labels. We can see these using the `rownames` function:

```
rownames(grades_df)
```

```
[1] "1" "2" "3"
```

By default we see these row labels were set to be text strings `"1"`, `"2"`, `"3"`, etc,

Selecting data

Data frames are 2-dimensional so can make selections using row and column indices in the same way as we did for `matrix` objects.

Try out and check you understand the output of the following commands:

```
grades_df[2,3]
```

```
grades_df[1,1:3]
```

```
grades_df[3,]
```

```
grades_df[,-2]
```

Where selections return a set of data the result is given as another data frame object (displayed with column names, and row names) or a vector if the selection is of a single column.

We can also make selections using column or row labels. Try the following:

```
grades_df["2","grade"]
```

```
grades_df[,"teacher"]
```

```
grades_df[,c("name","grade")]
```

Note the following command does not work:

```
grades_df[,-c("teacher","student_id")]
```

$

R provides a useful shortcut for accessing columns from a data frame using `$`:

```
grades_df$grade
```

```
[1] 78.6 83.0 60.2
```

Now that we know how to select a column we can select the student id numbers and use these to overwrite the default row labels:

```
rownames(grades_df) <- grades_df$student_id
head(grades_df)
```

```
            name student_id grade gender   teacher age
100213    Lauren     100213  78.6      F  Mr Smith  20
100214 Marcelino     100214  83.0      M Ms Clarke  19
100215    Ramona     100215  60.2      F  Mr Smith  18
```

We can now remove the `student_id` column. We do this by selecting all columns except column 2, and storing this back into `grade_df` (this overwrites the original data frame).

```
grade_df <- grade_df[,-2]
head(grades_df)
```

```
            name grade gender   teacher age
100213    Lauren  78.6      F  Mr Smith  20
100214 Marcelino  83.0      M Ms Clarke  19
100215    Ramona  60.2      F  Mr Smith  18
```

Finally let's correct the data type of the names column, so it is set to store `character` rather than `Factor` data.

We can convert a data type using a command like the following:

```
as.character(grades_df$name)
```

```
[1] "Lauren"    "Marcelino" "Ramona"
```

and to convert a column, we overwrite it with the corrected data:

```
grades_df$name  <- as.character(grades_df$name)
str(grades_df)
```

```
'data.frame':   3 obs. of  5 variables:
 $ name   : chr  "Lauren" "Marcelino" "Ramona"
 $ grade  : num  78.6 83 60.2
 $ gender : Factor w/ 2 levels "F","M": 1 2 1
 $ teacher: Factor w/ 2 levels "Mr Smith","Ms Clarke": 1 2 1
 $ age    : num  20 19 18
```

Graphics

The `plot()` function is the default command to plot data in R. For instance, `plot(x,y)` produces a scatterplot of the numbers in `x` versus the numbers in `y`. There are many additional options that can be passed in to the `plot()` function. For example, passing in the argument `xlab` will result in a label on the x-axis.

```
x <- rnorm(100)
y <- rnorm(100)
plot(x,y)
plot(x, y, xlab = "this is the x-axis",
       ylab = "this is the y-axis",
       main = "Plot of X vs Y")
```

We will often want to save the output of an R plot. One easy way to do this is using the **export** option in RStudio's plots pane menu. From this we can save the image or copy it to the clipboard, so it can be pasted into another application.

We can also save an image to disk. The command that we use to do this will depend on the file type that we would like to create. For instance, to create a `png`, we use the `png()` function, and to create a pdf, we use the `pdf()` function.

```
png("my_firstigure.png")
plot(x,y,col="green")
dev.off()
```

The function `dev.off()` indicates to R that we are done creating the plot.

The `par` command lets us specify graphics parameters. One useful parameter is `mfrow` which defines a plot arrangment in the figure:

```
par(mfrow=c(1,2))
```

The two numbers here specify the number of plots in our figure by row and column. Now if we draw two plots they appear side by side:

```
plot(x, col="green")
plot(y, col="blue")
```

Note to reset to a single plot in each figure run command `par(mfrow=c(1,1))`.

Loading the `Auto` dataset

We will be working with a dataset used in the *Introduction to Statistical Learning in R* textbook. The authors have provided an R *package* that can be used to access these. To install this package type the following:

```
install.packages("ISLR")
```

You should see the following that indicates the package has been downloaded and installed successfully.

```
trying URL 'https://cran.rstudio.com/bin/macosx/el-capitan/contrib/3.5/ISLR_1.2.
Content type 'application/x-gzip' length 2924917 bytes (2.8 MB)
==================================================
downloaded 2.8 MB
```

You only need to download a package once, but each time you want to use it you must tell R to load the package into the workspace environment using the `library` command:

```
library(ISLR)
```

This then let's us to load the example *ISLR* `Auto` dataset:

```
data(Auto)
```

Let's explore the dataset. First we check what it contains:

```
dim(Auto)
```

```
[1] 392 9
```

The `dim()` function tells us that the data has 392 observations, or rows, and 9 variables, or columns.
Let's use `names()` to check the column names.

```
names(Auto)
```

```
[1] "mpg"          "cylinders"    "displacement"
[4] "horsepower"   "weight"       "acceleration"
[7] "year"         "origin"       "name"
```

As this has been loaded from a package we can use the `help` function to read more about the data set.

```
?Auto
```

and we can use the `View` a function to display it in an RStudio panel.

```
View(Auto)
```

Additional Graphical and Numerical Summaries

We can use the `plot()` function to produce scatterplots of the quantitative variables. However, simply typing the column names will produce an error message, because R does not know to look in the `Auto` data set for those column names.

```
plot(cylinders, mpg)
```

```
Error in plot(cylinders , mpg) : object 'cylinders ' not found
```

To refer to a column, we must type the data set and the variable name joined with a `$` symbol.

```
plot(Auto$cylinders, Auto$mpg )
```

Note: use autocomplete to save typing and avoid typos.

The `cylinders` column is stored as a numeric vector, so R has treated it as quantitative. However, since there are only a small number of possible values for cylinders, one may prefer to treat it as a qualitative variable. The `as.factor()` function converts quantitative variables into qualitative variables (i.e. stored as a set of categories).

```
Auto$cylinders=as.factor(Auto$cylinders)
```

If the variable plotted on the x-axis is categorial, then boxplots will automatically be produced by the `plot()` function. As usual, a number of options can be specified in order to customize the plots.

Try out the following commands:

```
plot(Auto$cylinders, Auto$mpg, ylab ="MPG")
plot(Auto$cylinders, Auto$mpg, col="red")
plot(Auto$cylinders, Auto$mpg, col="red", varwidth=T)
plot(Auto$cylinders, Auto$mpg, col="red", varwidth=T,horizontal=T)
plot(Auto$cylinders, Auto$mpg, col="red", varwidth=T, xlab="cylinders",
```

The `hist()` function can be used to plot a histogram. Note that `col=2` has the same effect as `col="red"`.

```
hist(Auto$mpg)
hist(Auto$mpg,col=2)
hist(Auto$mpg,col=2,breaks=15)
```

The `pairs()` function creates a scatterplot matrix i.e. a scatterplot for every pair of variables for any given data set.

```
pairs(Auto)
```

We can also produce scatterplots for just a subset of the variables.

```
pairs(~ mpg + displacement + horsepower + weight +
acceleration, Auto)
```

In conjunction with the `plot()` function, `identify()` provides a useful interactive method for identifying the value for a particular variable for points on a plot. We pass in three arguments to `identify()`:

- the x-axis variable,
- the y-axis variable,
- and the variable whose values we would like to see printed.

After we call `identify` we can click on the points we are interested on the displayed plot. Once we are finished selecting points, we end by pressing the `escape` key, and R then adds the annotated information onto the plot. In the console it also displays the row numbers of the selected data points.

Note: *This can be particularly useful to identify any outliers in our data.*

```
plot(Auto$horsepower ,Auto$mpg)
identify(Auto$horsepower, Auto$mpg, Auto$name)
```

The `summary()` function produces a numerical summary of each variable in a particular data set

```
summary(Auto)
```

```
      mpg          cylinders      displacement
 Min.   : 9.00   Min.   :3.000   Min.   : 68.0
 1st Qu.:17.00   1st Qu.:4.000   1st Qu.:105.0
 Median :22.75   Median :4.000   Median :151.0
 Mean   :23.45   Mean   :5.472   Mean   :194.4
 3rd Qu.:29.00   3rd Qu.:8.000   3rd Qu.:275.8
 Max.   :46.60   Max.   :8.000   Max.   :455.0

...
```

We can also produce a summary of just a single column.

```
summary(Auto$mpg)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   9.00   17.00   22.75   23.45   29.00   46.60
```

For qualitative variables such as `name` , R will list the number of observations that fall in each category.