# R Cheat Sheet

As a general rule spaces in commands/code are completely ignored by R and thus do not matter
Use '#' at the beginning of a line to tell R to skip that line; use these to add comments to your source file to document what your code is doing

**objects**:

- everything you want to save in R (have available to use within your workspace) is an object
- you must name an object at the time of creation
- objects could be: a vector of numbers, a vector of words, results from running a function (e.g. aov()), or a data.frame
- a data.frame is the most common type of object you will use – it is a standard data set with a individuals (sometimes called 'cases') in rows (only one individual per row) and variables in columns (treatment category, response measure, etc.)

**<-** use this arrow symbol ('<' + '-') to create an object and its name; technically called a 'redirect' symbol
e.g. var1 <- c(5,3,2,5) creates an object that is a vector is 4 values in it
*anytime you run a function that has a return value, it will print it to some output; if you provide an object name, it will print it to that object; it you do not, it will use the 'standard ouput' (stdout), which is the console window

**$** use this to pull out variables from a data.frame; called 'string'
e.g. var1 <- my.data$var1 will create the vector 'var1' and put into it the same values that are in var1 found in the data.frame called 'my.data'

**[ ]** use square brackets to indicate any item in a vector, any row or column in a matrix or data.frame
e.g. in the data.frame 'my.data', my.data[1,3] will return (give the value of) the item in row 1, column 3; using my.data[1,] will give all the values (every column) in row 1, my.data[,3] will give all the values (every row) in column 3

<u>Frequently Used Functions in R</u>

Each of these functions requires an argument inside the parentheses that is the name of some object that exists in your workspace

A brief note on functions. Functions are simply a saved collection of commands; you use the function name to run those commands. Most functions require some objects – you indicate the objects inside the parentheses that identify an name as a function. Most objects return some value (an answer from running the commands); you can save this value to an object. If you do not, it will simply print the output to your console screen.

```
e.g. my.function(object1, object2) {
        lines of code that are saved together to be executed when function called
        do something with object1
        do something with object2
        return some.value  }        **the curly braces enclose the function code
```

Now call the function, sending it object1 and object2 and asking it to save the returned some.value to an object named result:

result <- my.function(object1, object2)

Functions that bring your data into and out of R

**read.csv()** – read a comma-delimited file saved in your working directory; give filename in quotes as argument; assign result of function to data frame object
e.g. my.data <- read.csv("myDataFile.csv")

**write.csv()** – write your data frame data to a comma-delimited file on your computer (in your working directory); provide name of data.frame and a name for the file
e.g. write.csv(my.data, "outputFile.csv")

Functions that provide information about your object

**str**() – 'str' is an abbreviation for 'structure'; this will return (prints to screen by default) the type of object and the kinds of data in the object

**head**() – returns the 'head' of the object = first 6 values (if vector) or lines (if matrix or data.frame)

**tail**() – returns the 'tail' of the object = last 6 values (if vector) or lines (if matrix or data.frame)

**nrow**() – returns the number of rows in a matrix or data.frame

**length**() – returns the number of items in a vector or the number of columns in a data.frame

**names**(my.df) – print the column names of the data.frame called 'my.df' that is in my workspace

**names**(my.2col.df) **<-** c("col1", "col2") – assign variable names 'col1' and 'col2' to the two columns that make up the data.frame called 'my.2col.df' in my workspace

Functions that manipulate data

**c()** – combine values into a single vector
e.g. vector1 <- **c**(value1, value2, value3) – create a vector containing the values inside the parentheses and assign it the object name 'vector1

**data.frame()** – put objects together into a data.frame (objects must have the same number of rows)
e.g. my.data.df <- c("factor1", "factor2", "data1", "data2")

**cbind()** – combine columns of data together into matrix (columns must have same number of rows)

**rbind()** – combine rows of data together (column names must match)

**as.numeric()** – change type of data in object to be 'numeric'

**as.character()** – change type of data in object to be 'character' (i.e. strings)

**as.factor()** – change type of data in object to be a factor; only available for objects (variables) within a data frame; has unique characteristics and required for use as categorical factor in ANOVA

**Elements of matrices and data.frames are identified using the form my.data[row, column]**

my.data[row_numbers,] – return only rows indicated by 'row_numbers'; all columns included because no value after comma

my.data[, col_numbers] – return only columns indicated by 'col_numbers'; all rows included because no value before comma

my.data[row#, col#] – return value of element found at specific row (=row#) and column (=col#)

Using formula argument in a function

Many functions, including plot(), t.test(), aov(), lm() will take a formula as input

- a formula is something that occurs in the form: y ~ a + b

This symbol (called 'tilde') is typically found on the key beside the '1' key on your keyboard. It means 'varies with'.

- e.g. To run an anova (aov function), you will have something in the form:
  anova.result <- aov(response ~ factor1 + factor2, data = my.data)
- An alternative is to use the '$' feature and the data.frame name:
  anova.result <- aov(my.data$response ~ my.data$factor1 + my.data$factor2)

This option is to tell R where to find the variables you've used in your formula

Summary Statistics

Any easy way to frequencies, means, etc. by grouping factors is to use the function **aggregate()**

- **aggregate()** takes as arguments
  - the variable you wish to summarize (typically a variable (column) within a data.frame
  - how you want the data to be subset, as a list of factor columns in your data.frame
  - a function to carry out on each group; standard functions available are: mean, var (variance), sd (standard deviation), sum, length (count of number in each group)

**mean()** – will take the mean value of numbers in object given to the function
**var()** – variance of values in object provided
**sd()** – standard deviation of objects provided
**summary()** – for numeric vector, returns minimum value, 1st quartile value, median, mean, 3rd quartile value, and maximum value; for character vector returns length of vector, its class and its mode