# R-cheat sheet from Data Camp course

*Shashank Sule*

*9/7/2019*

## R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

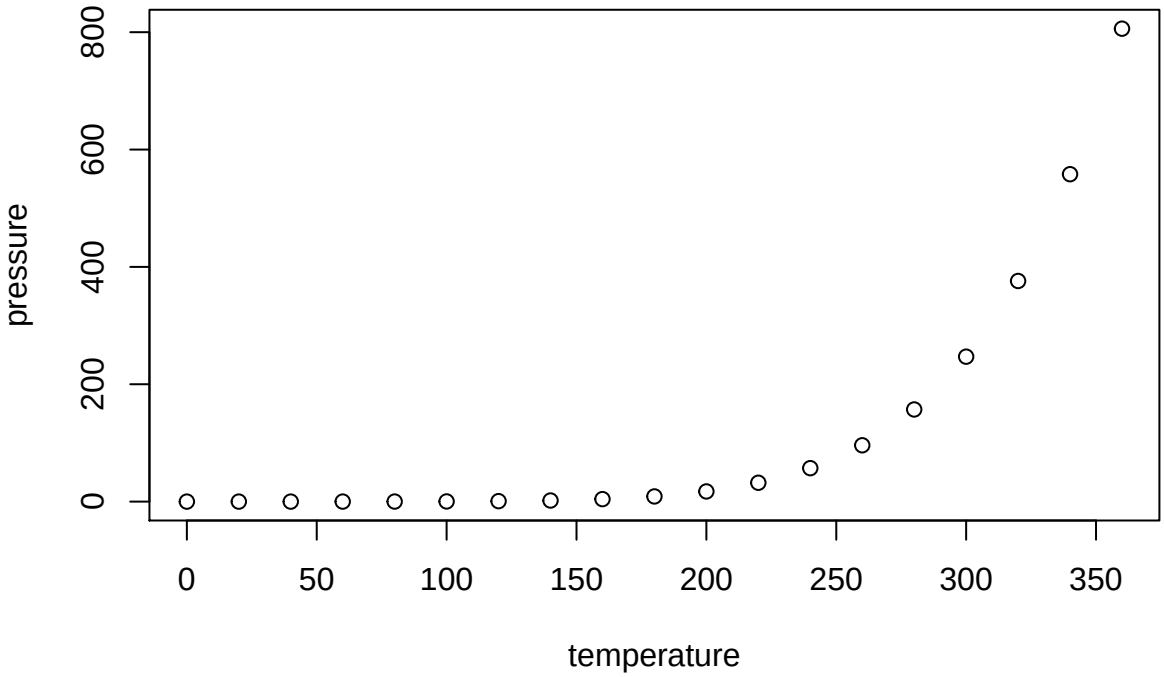
When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00
```

## Including Plots

You can also embed plots, for example:




Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

## R syntax from Data Camp

### General

How to ask for help about a function

```
?mean
```

### Vectors

Vectors can be generated in two ways:

```
Vec1 <- c(1,2,3)
Vec2 <- 1:3
Vec3 <- c("Monday", "Wednesday", "Tuesday")
Vec1
```

```
## [1] 1 2 3
```

```
Vec2
```

```
## [1] 1 2 3
```

```
Vec3
```

```
## [1] "Monday"    "Wednesday" "Tuesday"
```

Here $c$ is the concatentation operator. Note that $<-$ is the assignment operator.

We can use the following function to assign names to indices in a vector:

```
names(Vec1) <- Vec3
Vec1["Tuesday"]
```

```
## Tuesday
##       3
```

We can also create a sub-vector using vectors of logical values:

```
Vals = c(TRUE, FALSE, TRUE)
Vec1[Vals]
```

```
##  Monday Tuesday
##       1       3
```

Lastly, a `list` is a "generic" data type because it can store any sorts of objects you want in an ordered tuple:

```
A = list(c("Monday", "Tuesday", "Wednesday"), c(1,2))
A[1]
```

```
## [[1]]
## [1] "Monday"    "Tuesday"   "Wednesday"
```

```
A[2]
```

```
## [[1]]
## [1] 1 2
```

The `order` function orders values in a vector. It returns indices ordered by the ascending order of the elements they represent. Note that for numerical vectors it orders them by comparison but for string vectors it does that by lexicographic order. You can order a vector `v` by doing `v[order(v)]`

```
G = c("H", "Hell", "Hello","He", "Hel")
order(G)
```

```
## [1] 1 4 5 2 3
```

```
G[order(G)]
```

```
## [1] "H"     "He"    "Hel"   "Hell"  "Hello"
```

### Matrices

The `matrix` function takes three values and organizes a vector into a matrix specified by row or column:

```
NumMat = matrix(1:9, byrow=TRUE, nrow = 3)
NumMat
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

The `rownames` and `colnames` functions allow you to name rows and columns:

```
# Box office Star Wars (in millions!)
new_hope <- c(460.998, 314.4)
empire_strikes <- c(290.475, 247.900)
return_jedi <- c(309.306, 165.8)

# Construct matrix
star_wars_matrix <- matrix(c(new_hope, empire_strikes, return_jedi), nrow = 3, byrow = T

# Vectors region and titles, used for naming
region <- c("US", "non-US")
titles <- c("A New Hope", "The Empire Strikes Back", "Return of the Jedi")

# Name the columns with region
colnames(star_wars_matrix) <- region
rownames(star_wars_matrix) <- titles

star_wars_matrix
```

```
##                              US non-US
## A New Hope              460.998  314.4
## The Empire Strikes Back 290.475  247.9
## Return of the Jedi      309.306  165.8
```

We can do direct assignment within the `matrix` function as well, via the `dimnames` parameter. Furthermore, we can calculae the rowsum using `rowSums()`:

```
# Construct star_wars_matrix
box_office <- c(460.998, 314.4, 290.475, 247.900, 309.306, 165.8)
star_wars_matrix <- matrix(box_office, nrow = 3, byrow = TRUE,
                           dimnames = list(c("A New Hope", "The Empire Strikes Back", "R
                                           c("US", "non-US")))

# The worldwide box office figures
```

```
worldwide_vector <- rowSums(star_wars_matrix)

print(worldwide_vector)
```

```
##              A New Hope The Empire Strikes Back      Return of the Jedi
##                 775.398                 538.375                 475.106
```

Similarly, there is a `colsums()` function too. Furthermore, horizontal concatentation is done using `cbind()` and vertical concatentation is done using `rbind()`.

Finally, matrix selection works pretty logically. To say select the bottom right hand corner 2 by 2 matrix of the `star_wars_matrix` we do the following

```
bottom_right_hand_2by2 <- star_wars_matrix[2:3,1:2]
bottom_right_hand_2by2
```

```
##                              US non-US
## The Empire Strikes Back 290.475  247.9
## Return of the Jedi      309.306  165.8
```

Note that we selected all the columns of the matrix so we could have also done the following:

```
star_wars_matrix[2:3,]
```

```
##                              US non-US
## The Empire Strikes Back 290.475  247.9
## Return of the Jedi      309.306  165.8
```

The syntax works similarly if you want to select particular columns and all rows. Lastly, to recover the dimensions of a matrix, we can use the `dim` function which returns an integer vector.

```
dim(star_wars_matrix)
```

```
## [1] 3 2
```

```
class(dim(star_wars_matrix))
```

```
## [1] "integer"
```

```
length(dim(star_wars_matrix))
```

```
## [1] 2
```

### Factors

Factors are pretty high level data structures in R. Basically, a data set may comprise categorical variables such as “Male” and “Female”. The factors tool bascially helps R understand that a given variable is actually a dataset of categorical variables. This is a very data-specific tool.

Here's a simple example. The `cars` vector stores the speeds of 5 cars (indexed as `1:5`) as `"slow"`, `"medium"`, or `"fast"`. Now, I want R to know that. So I use the `factor` function with ordering.

```
cars = c("slow", "medium", "medium", "slow", "fast")
cars_ordered = factor(cars, ordered=TRUE, levels=c("fast", "medium","slow"))
summary(cars_ordered)
```

```
##   fast medium   slow
##      1      2      2
```

```
summary(cars)
```

```
##    Length     Class      Mode
##         5 character character
```

As you can see, R views `cars_ordered` differently than `cars` because according to R, `cars` is just another vector of strings whereas `cars_ordered` is actually a dataset that stores data on which car is slow medium or fast. (This is also what you do when you mark a multiple choice exam. You first grade, and then count how many questions were right, wrong, and not attempted).

In another perspective, `factor` picks up distinct values from a multiset. So to recover the distinct values, we use the levels function:

```
C =
  levels(cars_ordered)
C
```

```
## [1] "fast"   "medium" "slow"
```

```
C[1] > C[2]
```

```
## [1] FALSE
```

Only note that when the factor is ordered, the variables in the levels vector carry extra information, namely information on their ordering. That is why `C[1] > C[2]` returned `TRUE` because `"fast"` is bigger than `"slow"`.

### data.frames

Data frames are R's flagship for storing datasets. You can create a dataset using row/column vectors and literally entering them as arguments in the `data.frame()` function:

```
# Definition of vectors
name <- c("Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune")
type <- c("Terrestrial planet", "Terrestrial planet", "Terrestrial planet",
          "Terrestrial planet", "Gas giant", "Gas giant", "Gas giant", "Gas giant")
diameter <- c(0.382, 0.949, 1, 0.532, 11.209, 9.449, 4.007, 3.883)
```

```
rotation <- c(58.64, -243.02, 1, 1.03, 0.41, 0.43, -0.72, 0.67)
rings <- c(FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, TRUE)

# Create a data frame from the vectors
planets_df <- data.frame(name,type, diameter, rotation, rings)
planets_df
```

```
##      name               type diameter rotation rings
## 1 Mercury Terrestrial planet    0.382    58.64 FALSE
## 2   Venus Terrestrial planet    0.949  -243.02 FALSE
## 3   Earth Terrestrial planet    1.000     1.00 FALSE
## 4    Mars Terrestrial planet    0.532     1.03 FALSE
## 5 Jupiter          Gas giant   11.209     0.41  TRUE
## 6  Saturn          Gas giant    9.449     0.43  TRUE
## 7  Uranus          Gas giant    4.007    -0.72  TRUE
## 8 Neptune          Gas giant    3.883     0.67  TRUE
```

Data frames are better than just matrices because R has additional knowledge about their entries. For example, it understands that the top rows are actually labels for the table. However, you can refer to elements in the data frame just like matrices using index notation:

```
# Print out diameter of Mercury (row 1, column 3)

planets_df[1,3]
```

```
## [1] 0.382
```

```
# Print out data for Mars (entire fourth row)
planets_df[4,]
```

```
##   name               type diameter rotation rings
## 4 Mars Terrestrial planet    0.532     1.03 FALSE
```

You can also extract a category from the frame using the $ operator. This is an additional feature of data frames:

```
planets_df$diameter
```

```
## [1]  0.382  0.949  1.000  0.532 11.209  9.449  4.007  3.883
```

Additionally, you can always extract this into a vector and operate on it like you operate on vectors.