An introduction to R – Part 1

Harlley Lima & Manel Slokom

November 14, 2018

Before we start

Communication:

- Slack: AMS MADE Data 1 workspace
- Channels:
  - #general: communication and annoucementss
  - #lectures: everything related to the lectures
  - #tutorials: everything related to the tutorials

Download and Install R and RStudio

1. Download and install R (Link: `https://cran.r-project.org/`).
2. Download and install RStudio (Link: `https://www.rstudio.com/`)

Overview

1. Coding basics

2. Built-in Functions

3. Data types
   - Vectors
   - Factors
   - Matrices and Arrays
   - Lists and data frames

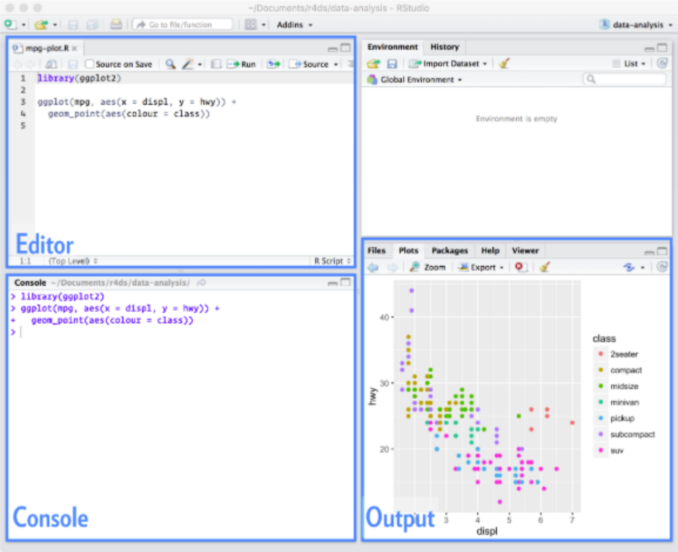
Overview

1 Coding basics

2 Built-in Functions

3 Data types
- Vectors
- Factors
- Matrices and Arrays
- Lists and data frames

Console



mpg-plot.R

```
library(ggplot2)

ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(colour = class))
```

Console ~/Documents/r4ds/data-analysis/

```
> library(ggplot2)
> ggplot(mpg, aes(x = displ, y = hwy)) +
+   geom_point(aes(colour = class))
> |
```

Environment is empty

Expression

Using R as a calculator (console):

- `1 / 200 * 13`
- `59 + 14 * 13`
- `20 / 4 / 5`

Tip: up and down arrow keys scroll through your command history.

Operators

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
%/%	is integer division (discards the fractional part)
^	exponentiation
x %% y	modulus (x mod y) 5 %% 2 is 1

Pay attention to the precedence of operators when executing an expression in R

Variables

All R statements where you assign values to variables have this form:

- `variable_name <- value`

Variable names must:

- start with a letter
- contain letters, numbers, _, and `.` (it's discouraged)
- case sensitive: `a` and `A` are different, `Number` and `number` also refer to different variables

Variables

You can inspect an variable by typing its name:

-
  ```
  x <- 4 * 4 # 16 is assigned to x
  x
  #>[1] 16
  ```

-
  ```
  this_is_a_long_name <- 2.5
  this_is_a_long_name
  #>[1] 2.5
  ```
- Type "this", press Tab, add characters until you have a unique prefix!

Tip keyboard shortcut for <-: Alt- (The minus sign)

Comments (#) are added with the purpose of making the code easier to understand, and are they ignored by R

Relational Operators

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
! =	Not equal to

Logical Operators

<table>
<tr><th>Operator</th><th>Description</th></tr>
<tr><td>!</td><td>Logical NOT</td></tr>
<tr><td>&</td><td>Element-wise logical AND</td></tr>
<tr><td>&&</td><td>Logical AND</td></tr>
<tr><td>|</td><td>Element-wise logical OR</td></tr>
<tr><td>||</td><td>Logical OR</td></tr>
</table>

- Operators & and | perform **element-wise** operation producing result having length of the longer operand.
- && and || examine only the **first element** of the operands resulting into a single length logical vector.
- Zero is considered *FALSE* and non-zero numbers are taken as *TRUE*.
- $! =$ comes from the maths symbol $\neq$ ($a! = b$)

Examples

```
> x <- 5
> y <- 16
> x < y
[1] TRUE

> x > y
> FALSE

> x <= 5
[1] TRUE

> y >= 20
[1] FALSE

> y == 16
[1] TRUE

> x != 5
[1] FALSE
```

```
> x <- c(TRUE, FALSE, 0,6)
> y <- c(FALSE, TRUE, FALSE,
TRUE)
> !x
[1] FALSE TRUE TRUE FALSE

> x & y
[1] FALSE FALSE FALSE TRUE

> x && y
[1] FALSE

> x | y
[1] TRUE TRUE FALSE TRUE

> x || y
[1] TRUE
```

Examples

```
> x <- c(3, 5, 1, 2, 7, 6, 4)
> x

> (x > 2) & (x <= 6) # is x greater than 2 and less than or equal to 6

 [1] TRUE TRUE FALSE FALSE FALSE TRUE TRUE

> (x < 2) | (x > 5) # is x less than 2 or greater than 5
 [1] FALSE FALSE TRUE FALSE TRUE TRUE FALSE

> !(x > 3) # not [x greater than 3]
 [1] TRUE FALSE TRUE TRUE FALSE FALSE FALSE
```

Basic data types

R has many types. These include:

- character: "a", "swc"
- numeric: 2, 15.5
- logical: TRUE, FALSE

Different data types can be assigned to the same variable:

-
  ```
  a <- 13
  a <- 13.3
  a <- "thirteen"
  a <- TRUE
  ```

Exercises

1. Write a code to calculate the distance between two points.
   $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Overview

1. Coding basics

2. Built-in Functions

3. Data types
   - Vectors
   - Factors
   - Matrices and Arrays
   - Lists and data frames

Calling functions

R has a large collection of pre-defined functions that are called like this:

- `function_name(arg1 = val1, arg2 = val2, ...)`

Examples:

- `seq(1, 10)`: 1 2 3 4 5 6 7 8 9 10
  operator colon `:` - Generate regular sequences. Ex. `x <- 1:10`
- `sqrt(x)`: square root
- `sin(x)`: computes the sine of the given value
- `cos(x)`: computes the cosine of the given value
- **help()**: provides access to the documentation pages for R
  Ex.: `help(sin)` or `?sin`
- `??`: it allows for searching the help system for documentation matching a given character string
  `Ex.: ??seq`

Overview

1. Coding basics

2. Built-in Functions

3. Data types
   - Vectors
   - Factors
   - Matrices and Arrays
   - Lists and data frames

Vectors

Vector

- Vector is a basic data structure in R
- A vector is a sequence of data elements of the same basic type
- Vectors are created using the `c()` function. Examples:
    - `x <- c(1, 5, 4, 9, 0)`
    - `x <- c(FALSE, TRUE, FALSE)`
    - `x <- c("a", "b", "c", "d", "e")`
    - `x <- c(10.4, 5.6, 3.1, 6.4, 21.7)`
- `length`: get the length of vectors:

  ```
  x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
  length(x)
  [1] 5
  ```
- Single number is also a vector, of length one.

Indexing vectors

-
  ```
  x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
  x[1]
  [1] 10.4
  ```

-
  ```
  x[c(2,4)]
  [1] 5.6 6.4
  ```

-
  ```
  x[-1] # access all but 1st element
  [1] 5.6 3.1 6.4 21.7
  ```

-
  ```
  x[c(-1,-3)] # access all but 1st and 3rd element
  [1] 5.6 6.4 21.7
  ```

-
  ```
  x[c(2.4, 3.54)] # real numbers are truncated to
  integers
  [1] 5.6 3.1
  ```

Using logical vector as index

```
x <- c(10,20,30,40,50,60)
```

- `x[c(TRUE, FALSE, FALSE, TRUE, TRUE, TRUE)]`
  ```
  [1] 10 40 50 60
  ```
- `x[x < 20]`
  ```
  [1] 10
  ```
- `x[x > 20]`
  ```
  [1] 30 40 50 60
  ```
- Range index:
  ```
  x[2:4]
  [1] 20 30 40
  ```

Combining Vectors

-
  ```
  n = c(2, 3, 5)
  s = c("aa", "bb", "cc", "dd", "ee")
  c(n, s)
  [1] "2" "3" "5" "aa" "bb" "cc" "dd" "ee"
  ```

Notice how the numeric values are being changed into character strings when the two vectors are combined. This is necessary so as to maintain the same data type for members in the same vector

How to modify a vector

```
x <- c(10,20,30,40,50,60)
```

- Index in R stars with 1.
  `x[1] <- 80`: modify 1st element
- `x[2] <- 0`: modify 2nd element
- `x[x<30] <- 50 # modify elements less than 3.`

  ```
  x
  [1] 50 50 30 40 50 60
  ```

Vector arithmetics

```
a <- c(1, 3, 5, 7)
b <- c(1, 2, 4, 8)
```

- `5 * a`
  ```
  [1] 5 15 25 35]
  ```
- `a + b`
  ```
  [1] 2 5 9 15
  ```
- `a - b`
  ```
  [1] 0 1 1 -1
  ```
- `a * b`
  ```
  [1] 1 6 20 56
  ```
- `a / b`
  ```
  [1] 1.000 1.500 1.250 0.875
  ```

Vector arithmetics

Recycling Rule

-
  ```
  u <- c(10, 20, 30)
  v <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
  u + v
  [1] 11 22 33 14 25 36 17 28 39
  ```

Named vector members

```
v <- c("Mary", "Jane")
```

- We now name the first member as First, and the second as Last:

  ```
  names(v) = c("First", "Last")
  v
  First Last
  "Mary" "Jane"
  ```

- Short syntax `v <- c(First = "Mary", Last = "Jane")`
- Now we can index using the names:

  ```
  > v["First"]
  First
  "Mary"
  > v["Last"]
  Last
  "Jane"
  ```

Examples

```
> x <- 0:6
> class(x)
 [1] "integer"
```

-
  ```
  > as.numeric(x)
  [1] 0 1 2 3 4 5 6
  ```

-
  ```
  > as.logical(x)
  [1] FALSE TRUE TRUE TRUE TRUE TRUE TRUE
  ```

-
  ```
  > as.character(x)
  [1] "0" "1" "2" "3" "4" "5" "6"
  ```

Examples

```
> x <- c("a", "b", "c")
```

-
  ```
  > as.numeric(x)
   [1] NA NA NA
  ```

-
  ```
  > as.logical(x)
  [1] NA NA NA
  ```

-
  ```
  > as.complex(x)
  [1] NA NA NA
  ```

```
> is.numeric(x)
 [1] FALSE
```

Special values used in R

Missing values

Missing values are denoted by `NA` (not available) or `NaN` (Not a Number) for undefined mathematical operations.

- `is.na()` is used to test variables if they are *NA*
- `is.nan()` is used to test for *NaN*

In R, NaN stands for "Not a Number"

-
  ```
  > sqrt(-4)
  Warning:  NaNs produced
  [1] NaN
  ```

Missing values

```
## Create a vector with NAs in it
> x <- c(1, 2, NA, 10, 3)

## Return a logical vector indicating which elements are
NA
> is.na(x)
[1] FALSE FALSE TRUE FALSE FALSE

## returning a logical vector indicating which elements
are NaN
> is.nan(x)
[1] FALSE FALSE FALSE FALSE FALSE
```

Missing values

```
## Now let’s create a vector with both NA and NaN values
> x <- c(1, 2, NaN, NA, 4)
> is.na(x)
[1] FALSE FALSE TRUE TRUE FALSE

> is.nan(x)
[1] FALSE FALSE TRUE FALSE FALSE
```

Infinity

Positive and negative infinity are represented with `Inf` and `-Inf`, respectively:

-
  ```
  > 1 / 0
  [1] Inf
  ```

-
  ```
  > -5 / 0
  [1] - Inf
  ```

Factors

- Factors are used to work with **categorical** variables, variables that have a fixed and known set of possible values. Like countries, programming language, etc.

Factors

- Creating factors
  - `countries <- c("Brazil", "Netherlands", "Tunisia", "Spain", "Brazil", "Tunisia", "Netherlands")`
  - Now you can create a factor:
    `y1 <- factor(countries)`

Matrices and Arrays

Matrices

- A matrix is a collection of data elements arranged in a two-dimensional rectangular layout.
- Matrices are vectors with a dimension attribute.
- The dimension attribute is itself an integer vector length 2 (#*nrow*, #*ncol*)

```
> m <- matrix(nrow = 2, ncol = 3)
> dim(m)
[1] 2 3
```

- Matrices are constructed column-wise by default
- Entries can be thought of starting in the “upper left” corner and running down the columns.

```
> m <- matrix(1:6, nrow = 2, ncol = 3)
```

Matrices

- Matrices can be created by column-binding or row-binding with the `cbind()` and `rbind()` functions

- `x <- 1:3` # sequence 1
- `y <- 10:12` # sequence 2
- `cbind(x, y)` # By columns
- `rbind(x, y)` # By rows

How to get access to elements in the matrix?

Example

```
> A = matrix(
c(2, 4, 3, 1, 5, 7), # the data elements
nrow=2, # number of rows
ncol=3, # number of columns
byrow = TRUE) # fill matrix by rows
> A # print the matrix
```

- An element at the m^{th} row, n^{th} column of A can be accessed by the expression `A[m, n]`:

```
A[2, 3] # element at 2nd row, 3rd column
```

- The entire m^{th} row can be extracted as `A[m, ]`.

```
> A[2, ] # the 2nd row
[1] 1 5 7
```

How to get access to elements in the matrix?

- Similarly, the entire n^{th} column A can be extracted as `A[ ,n]`.

```
> A[ ,3] # the 3rd column
[1] 3 7
```

- We can extract more than one rows or columns at a time.

```
> A[ ,c(1,3)] # the 1st and 3rd columns
```

- We assign names to the rows and columns of the matrix.
- We can access the elements by names.

-
  ```
  > colnames(A)
  NULL
  ```

-
  ```
  > colnames(A) <- c('a1', 'a2', 'a3')
  ```

Matrix manipulation

- **Transpose**: We construct the *transpose* of a matrix by interchanging its columns and rows with the function t:

  ```
  t(A) # transpose of A
  ```

- **Combining Matrices**: The columns of two matrices having the same number of rows can be combined into a larger matrix.

Matrix B

```
> B = matrix(
c(2, 4, 3, 1, 5, 7),
nrow=3,
ncol=2)
```

Matrix C

```
> C = matrix(
c(7, 4, 2),
nrow=3,
ncol=1)
```

Matrix manipulation

- We can combine the columns of B and C with **cbind**:

```
> cbind(B, C)
```

- Similarly, we can combine the rows of two matrices if they have the same number of columns with the **rbind** function.

```
> D <- matrix(
+ c(6, 2),
+ nrow=1,
+ ncol=2)
> rbind(B, D)
```

Matrix manipulation

Matrices may be used in arithmetic expressions and the result is a matrix formed by element-by-element operations.

```
> a <- matrix(10:15, nrow = 2, ncol= 3)
> class(a)
[1] "matrix"
> typepf(a)
[1] "integer"
```

```
> b <- matrix(10:15, nrow = 2, ncol = 3, byrow = TRUE)
dim(b)
[1] 2 3
```

```
a + b
a * b # element-by-element product
```

Matrix operators and functions

- R offers functions designed to work on a matrix `A`.

Function	Description
`dim(A)`	dimension of A
`t(A)`	transpose the matrix A
`rowMeans(A)`	row means
`rowSums(A)`	row sums
`colMeans(A)`	column means
`colSums(A)`	column sums
`colnames(A)`	column names
`rownames(A)`	row names
`ncol(A)`	number of columns
`nrow(A)`	number of rows

Arrays

- An array is an extension of a vector to more than two dimensions.
- While matrices are comfined to two dimensions, arrays can be of any number of dimensions.
- In R we can generate an array with the array function.
- The `array()` function takes a **dim** attributes which creates the required number of dimension.

-
  ```
  # The array above has dimensions 3 x 4 x 2
  > a <- array(data = 1:24, dim = c(3, 4, 2))
  ```

Lists and data frames

Lists

- Lists are special type of vectors that can contain elements of different classes.
- Lists are very important data type in R.
- The following variable x is a **list** containing copies of three vectors n, s, b, and a numeric value 3.

```
> n <- c(2, 3, 5)
> s <- c("aa", "bb", "cc", "dd", "ee")
> b <- c(TRUE, FALSE, TRUE, FALSE, FALSE)
> x <- list(n, s, b, 3) # x contains copies of n, s, b
```

- **List Slicing**: the following is a slice returning a list with the 2nd element, which is a copy of s:

```
> x[2]
[[1]]
[1] "aa" "bb" "cc" "dd" "ee"
```

Lists

- We can retrieve a slice with multiple members. Here a slice containing the second and fourth members of x.

```
> x[c(2, 4)]
[[1]]
[1] "aa" "bb" "cc" "dd" "ee"

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

- We can modify its content directly:

```
> x[[2]][1] <- "ta"
> x[[2]]
[1] "ta" "bb" "cc" "dd" "ee"
> s
[1] "aa" "bb" "cc" "dd" "ee" # s is unaffected
```

Lists

```
> Lst <- list(name = "Fred", wife = "Mary", no.children =
3, child.ages= c(4, 7, 9))
```

- The function `length()` gives the number of components in the list

  ```
  > length(Lst)
  [1] 4
  ```
- Component names: > `name$component-name`

```
• > Lst$name
  [1] "Fred" # is the same as Lst[[1]]
• > Lst$child.ages[1]
  [1] 4 # is the same as Lst[[4]][1]
```

Lists

It is important to distinguish `Lst[[1]]` from `Lst[1]`

- ‘`[[]]`’ is the operator used to return element of the list.
  It is the first element in the list.
- ‘`[]`’ is a general subscripting operator.
  It is a sublist of the list.

Data frames

- A data frame is used for storing data tables.
- Unlike a matrix in data frame each column can contain different types of data:
  - The first column can be ‘numeric’
  - The second can be ‘character’
  - The third column can be ‘logical’
- It is a list of vectors of equal length (we can index just like we indexed lists).
- Data frames are created using the data.frame() function.

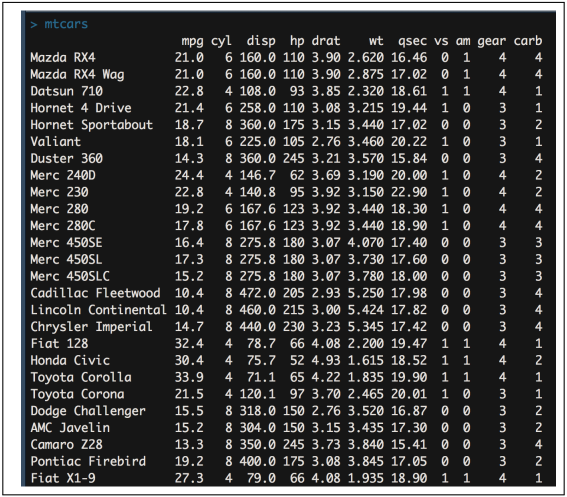
Data frames

```
# create the data frame
BMI <- data.frame (
      gender = c("Male", "Male", "Female"),
      height = c(152, 171.5, 165),
      weight = c(81, 93, 78),
      age = c(42, 38, 26)
)
```

Build-in Data Frame

We will use a built-in data frames in R for our tutorials, called:

> mtcars



```
> mtcars
                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
```

Build-in Data Frame

- The top line of the table, called the **header**, contains the column names.
- Each horizontal line afterward denotes a **data row**, which begins with the name of the row, and then followed by the actual data.
- Each data member of a row is called a **cell**.

Data Frame

- To retrieve data in a cell
- We would enter its row and column coordinates in the single square bracket "[]" operator

Data Frame

- The cell value from the first row, second column:

```
> mtcars[1, 2]
[1] 6
```

- We can use the row and column names instead of the numeric coordinates.

```
> mtcars["Mazda RX4", "cyl"]
[1] 6
```

- We can combine column names and the numeric coordinates.

```
> mtcars[1, "cyl"]
[1] 6
```

Data Frame

- The number of data rows in the data frame is given by the **nrow** function.

  ```
  > nrow(mtcars)   # number of data rows
  [1] 32
  ```

- The number of columns of a data frame is given by the **ncol** function.

  ```
  > ncol(mtcars)   # number of columns
  [1] 11
  ```

- Further details of the mtcars data set: `> help(mtcars)`

Instead of printing out the entire data frame, it is often desirable to preview it with

```
>  head(mtcars)
```

Data Frame Column Vector

- We can also retrieve with the “$” operator

```
> mtcars$am
[1] 1 1 1 0 0 0 0 0 0 0 0 ...
```

Data frame column slice

We retrieve a data frame column slice with the single square bracket “[]” operator.

The output is a data.frame.

Numeric Indexing

The following is a slice containing the first column of the built-in data set:

```
> mtcars[1]
```

Name Indexing

We can retrieve the same column slice by its name.

```
> mtcars["mpg"]
```

To retrieve a data frame slice with the two columns mpg and hp:

```
> mtcars[c("mpg", "hp")]
```

Data frame row slice

We retrieve rows from a data frame with the single square bracket operator.

```
>  mtcars[24,]
[1]          mpg cyl disp hp drat wt ...
Camaro Z28 13.3 8 350 245 3.73 3.84 ...
```

Note that in addition to an index vector of row positions, we append an extra comma character. This is important, as the extra comma signals a wildcard match for the second coordinate for column positions.

To retrieve more than one rows, we use a numeric index vector.

```
>  mtcars[c(3, 24),]
```

Data frame row slice

Name Indexing

- We can retrieve a row by its name.

```
> mtcars["Camaro Z28",]
[1]         mpg cyl disp hp drat wt ...
Camaro Z28 13.3 8 350 245 3.73 3.84 ...
```

Subset of a data.frame

- Let's select the rows of the data.frame `mtcars` where the `cyl` column is greater than 6

```
> mtcars [mtcars$cyl > 6,]
# 2nd possibility
> subset(mtcars, cyl > 6)
```

- now let's select the rows of `mtcars` where the `cyl` column is greater than 6 or is equal to 8.

```
> mtcars [mtcars$cyl > 6 | mtcars$cyl == 8, ]
> subset(mtcars, cyl> 6 | cyl == 8)
```

Subset of a data.frame

We can subset both rows and columns at the same time:

```
> mtcars [mtcars$cyl > 6 | mtcars$cyl == 8, c("hp", "drat",
"wt")]
# or
> subset(mtcars, cyl > 6 | cyl == 8, select = hp:wt)
```

References

Hadley Wickham & Garrett Grolemund (2017)
R for data science: import, tidy, transform, visualize, and model data
O'Reilly.

Roger D. Peng (2015)
R Programming for Data Science

Venables W.N. & Smith D. M. (2018)
An introduction to R: Notes on R - A programming for Data Analysis and Graphics
Version 3.5.1.

Emmanuel Paradis (2005)
R for beginners

Link:
Advanced R
https://adv-r.hadley.nz/

The End