Lab 1: Introduction to R

Feng Qiu, Liyuan Xuan

0.1 Download and Install

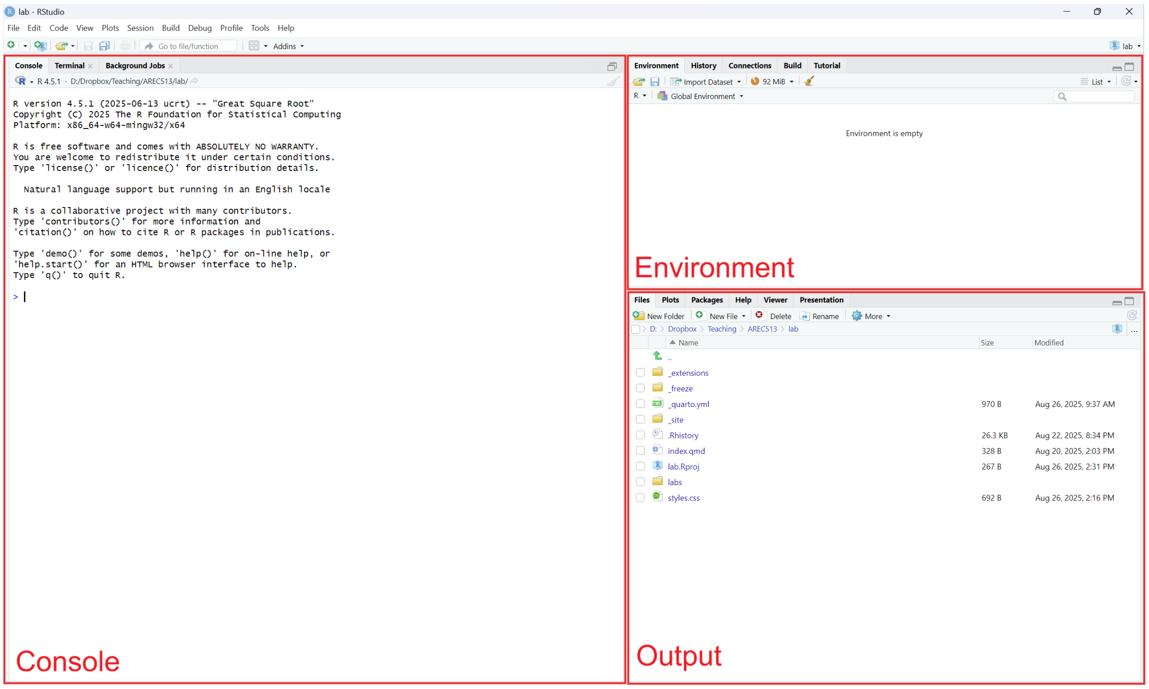
- **R**

  R is a free software environment for statistical computing and graphics. To download, go to CRAN and select the installer for your operating system (Windows, Mac, or Linux).

- **RStudio**

  RStudio is a user-friendly application that helps you write in R and enhances your programming experience. To download, visit this website and select the installer for your operating system.

After R and RStudio are installed, we will only need to use RStudio for this and future labs. The default RStudio layout has three panes: **Console**, **Environment**, and **Output**.



```
R version 4.5.1 (2025-06-13 ucrt) -- "Great Square Root"
Copyright (C) 2025 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

Environment is empty

Name	Size	Modified
..		
_extensions		
_freeze		
_quarto.yml	970 B	Aug 26, 2025, 9:37 AM
_site		
.Rhistory	26.3 KB	Aug 22, 2025, 8:34 PM
index.qmd	328 B	Aug 20, 2025, 2:03 PM
lab.Rproj	267 B	Aug 26, 2025, 2:31 PM
labs		
styles.css	692 B	Aug 26, 2025, 2:16 PM

Figure 1: screenshot of RStudio

You can customize your RStudio working environment via Tools > Global Options in the top menu bar.

0.2 Working Directory

The working directory is a folder path on your computer that sets the default location for files you read into R or save out of R. Think of it as a little "flag" on your computer tied to your project.

- **To find your current working directory**, type the code below in your Console pane and press *Enter*.

```
getwd()
```

```
[1] "D:/University of Alberta/0. Teaching/AREC513/Fall2025/PDF_only/lab1"
```

- **To change your current working directory**, you can

  1. Use the code below

```
setwd("C:/Program Files")
```

> 💡 Tip
>
> In R, file paths are separated by **forward slash** `/`, not backward slash `\`. The full path needs to be wrapped with the double quotations `" "`

  2. Use the top menu bar Session > Set Working Directory > Choose Directory
  3. To set a default working directory, go to Tools > Global Options > General > Default working directory

0.3 R Basics

0.3.1 Calculations

```
1 + 1
```

```
[1] 2
```

```
5 * 6
```

```
[1] 30
```

```
2 ^ 4
```

```
[1] 16
```

```
(8 + 5 * 6 / 3) / 2
```

```
[1] 9
```

0.3.2 Creating Objects

```
a <- 5 * 6
a
```

```
[1] 30
```

```
b <- 8 + a / 3
b
```

```
[1] 18
```

```
c <- "AREC 513"
c
```

```
[1] "AREC 513"
```

> Tip
>
> In RStudio, you can use the shortcut *Alt* + - *(hyphen)* to write the assignment operator `<-`.

0.3.3 Data Types

There are four main data types in R we will use in the labs: numeric, integer, logical, and character.

- **Numeric data**, also known as quantitative data

  ```
  is.numeric(b)     # test if the object "b" is numeric
  ```

  ```
  [1] TRUE
  ```

- **Integer data**, stores whole numbers without decimals

  ```
  d <- 8L     # to generate integer variable, add "L" after the number
  is.integer(d)
  ```

  ```
  [1] TRUE
  ```

- **Logical data**, stores only TRUE or FALSE

```
e <- TRUE
is.logical(e)
```

```
[1] TRUE
```

- **Character data**, stores text strings

```
is.character(c)
```

```
[1] TRUE
```

Tip

You can also use `class()` to check the class of an object. For example, try `class(a)`.

To convert your objects to a specific type, use `as.numeric()`, `as.integer()`, `as.logical()`, or `as.character()`.

0.3.4 Data Structures

When analyzing data, you rarely deal with objects that store one single value or datasets with one single variable. This section discusses some common data structures.

0.3.4.1 Vectors

0.3.4.1.1 Create a Vector

Vectors play a crucial role in R, R is a vectorized language. A vector is a collection of values/elements, all of the same type. We can use the function c() to create a vector, which means "combine".

```
grades_1 <- c(75, 76, 77, 78, 79, 80, 81, 82, 83, 84)
grades_1
```

```
 [1] 75 76 77 78 79 80 81 82 83 84
```

```
grades_2 <- c(75:84) # to generate continuous values from 75 to 80, you can use ":" directly.
grades_2
```

```
 [1] 75 76 77 78 79 80 81 82 83 84
```

```
names <- c("Marshall", "Ruby", "Peppa", "George", "Suzy", "Danny",
           "Pedro", "Rebecca", "Rubble", "Ryder", "Max", "Chase")
names
```

```
 [1] "Marshall" "Ruby"     "Peppa"    "George"   "Suzy"     "Danny"   
 [7] "Pedro"    "Rebecca"  "Rubble"   "Ryder"    "Max"      "Chase"   
```

0.3.4.1.2 Vector Operations

```
grades_1 + 5 # operations apply to each element in the vector
```

```
 [1] 80 81 82 83 84 85 86 87 88 89
```

```
grades_1 * 2
```

```
 [1] 150 152 154 156 158 160 162 164 166 168
```

```
sqrt(grades_1)
```

```
 [1] 8.660254 8.717798 8.774964 8.831761 8.888194 8.944272 9.000000 9.055385
 [9] 9.110434 9.165151
```

```
grades_1 >= 78
```

```
 [1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

0.3.4.1.3 Factor Vectors

Factor variables are important when building statistical models. There are two types of factor variable,

- **nominal**: categorical variable with no inherent order among the categories

```
jbp <- c("poor", "excellent", "fair", "poor", "good") # create a nominal job performance vector
jbp
```

```
[1] "poor"      "excellent" "fair"      "poor"      "good"     
```

```
class(jbp)
```

```
[1] "character"
```

```
jbp.1 <- as.factor(jbp)     # converting your character vector to factor
jbp.1
```

```
[1] poor      excellent fair      poor      good     
Levels: excellent fair good poor
```

```
class(jbp.1)
```

```
[1] "factor"
```

- **ordinal**: categorical variable with a defined ranking among the categories

```
# create a job performance vector, that is nominal
jbp.2 <- factor(jbp, levels = c("poor", "fair", "good", "excellent"))     # assigning an ordinal ranking
jbp.2
```

```
[1] poor      excellent fair      poor      good     
Levels: poor fair good excellent
```

0.3.4.1.4 Subset Vectors and Select Elements

To subset a vector or select certain elements from a vector, we use the square brackets `[ ]`.

- **By Position**

  ```
  grades_1[1]         # select the 1st element
  ```

  ```
  [1] 75
  ```

  ```
  grades_1[-2]        # exclude the 2nd element
  ```

  ```
  [1] 75 77 78 79 80 81 82 83 84
  ```

```
grades_1[3:5]       # select elements 3 to 5
```

```
[1] 77 78 79
```

```
grades_1[c(1,3,5)]  # select the 1st, 3rd, and 5th elements
```

```
[1] 75 77 79
```

- **By Condition**

```
grades_1[grades_1 >= 78]               # select grades_1 that are equal or above 78
```

```
[1] 78 79 80 81 82 83 84
```

```
grades_1[grades_1 != 78]               # select grades_1 that do not equal to 78
```

```
[1] 75 76 77 79 80 81 82 83 84
```

```
jbp.2[jbp.2 %in% c("poor", "fair")]    # select job performance that is "poor" or "fair"
```

```
[1] poor fair poor
Levels: poor fair good excellent
```

> Tip
>
> If you wish to apply multiple conditions, the common operators are `&` (Shift + 7) for AND and `|` (Shift + \) for OR. Try `grades_1[grades_1 >= 78 & grades_1 != 80]`.

0.3.4.2 Matrices

A matrix is a collection of data elements arranged in a two-dimensional rectangular layout. The elements of a matrix must be of the same type of data. Matrices are commonly used in mathematics and statistics. Math matrices and R matrices are different concepts. Matrices in R are broader.

```
ma.1 <- matrix(1:15, nrow = 5, ncol =3)
ma.1
```

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

```
ma.2 <- matrix(names, nrow = 6, ncol = 2)
ma.2
```

```
     [,1]       [,2]
[1,] "Marshall" "Pedro"
[2,] "Ruby"     "Rebecca"
[3,] "Peppa"    "Rubble"
[4,] "George"   "Ryder"
[5,] "Suzy"     "Max"
[6,] "Danny"    "Chase"
```

Since matrices are two-dimensional, to subset or select elements from a matrix using `[ ]`, you need to define row and/or column index in `[ROW, COL]`.

```
ma.1[2, ]   # select all elements in the 2nd row of ma.1
```

```
[1]  2  7 12
```

```
ma.1[, 2]   # select all elements in the 2nd column of ma.1
```

```
[1]  6  7  8  9 10
```

```
ma.1[2, 2]  # select the element in row 2 and column 2 of ma.1
```

```
[1] 7
```

```
ma.2[3, 2]  # select the element in row 3 and column 2 of ma.2
```

```
[1] "Rubble"
```

0.3.4.3 Arrays

In R, arrays are the data objects that can store data in more than two dimensions. Data need to be the same type.

```
array.1 <- array(1:24, dim = c(2, 3, 4))      # 2 rows * 3 columns * 4 matrices
array.1
```

```
, , 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12

, , 3

     [,1] [,2] [,3]
[1,]   13   15   17
[2,]   14   16   18

, , 4

     [,1] [,2] [,3]
[1,]   19   21   23
[2,]   20   22   24
```

0.3.4.4 Lists

Lists are R objects that can contain multiple components of different data types, data structures, and dimensions.

```
names
```

```
 [1] "Marshall" "Ruby"     "Peppa"    "George"   "Suzy"     "Danny"   
 [7] "Pedro"    "Rebecca"  "Rubble"   "Ryder"    "Max"      "Chase"   
```

```
grades_1
```

```
 [1] 75 76 77 78 79 80 81 82 83 84
```

```
list.1 <- list(name = names, grade = grades_1, random_num = 9)      # create a list with 3 components
list.1
```

```
$name
 [1] "Marshall" "Ruby"     "Peppa"    "George"   "Suzy"     "Danny"
 [7] "Pedro"    "Rebecca"  "Rubble"   "Ryder"    "Max"      "Chase"

$grade
 [1] 75 76 77 78 79 80 81 82 83 84

$random_num
[1] 9
```

```
names(list.1)
```

```
[1] "name"       "grade"      "random_num"
```

To select from a list, you can keep using index numbers with square brackets `[ ]`, or you can use the extractor operator `$`.

```
list.1[1]     # create a new list with only the 1st component from list.1
```

```
$name
 [1] "Marshall" "Ruby"     "Peppa"    "George"   "Suzy"     "Danny"
 [7] "Pedro"    "Rebecca"  "Rubble"   "Ryder"    "Max"      "Chase"
```

```
list.1[[1]]   # extract all elements from the 1st component of list.1 (i.e. the name vector)
```

```
 [1] "Marshall" "Ruby"     "Peppa"    "George"   "Suzy"     "Danny"
 [7] "Pedro"    "Rebecca"  "Rubble"   "Ryder"    "Max"      "Chase"
```

```
list.1$name   # extract all the elements from the component named "name" from list.1
```

```
 [1] "Marshall" "Ruby"     "Peppa"    "George"   "Suzy"     "Danny"
 [7] "Pedro"    "Rebecca"  "Rubble"   "Ryder"    "Max"      "Chase"
```

```
list.1[[1]][1]
```

```
[1] "Marshall"
```

```
list.1$name[1]
```

```
[1] "Marshall"
```

0.3.4.5 Data Frames

A data frame is a list whose elements are equal-length vectors, and vectors can be different data types. Basically, a data frame is a limited version of a list, or a flexible version of a matrix. In a data frame, vectors/variables can be different types, but the length needs to be the same.

```
df1 <- data.frame(Name = names[1:10], Grade = grades_1)
df1
```

```
       Name Grade
1  Marshall    75
2      Ruby    76
3     Peppa    77
4    George    78
5      Suzy    79
6     Danny    80
7     Pedro    81
8   Rebecca    82
9    Rubble    83
10    Ryder    84
```

```
df1$Name
```

```
 [1] "Marshall" "Ruby"     "Peppa"    "George"   "Suzy"     "Danny"
 [7] "Pedro"    "Rebecca"  "Rubble"   "Ryder"
```

```
df1$Grade[1:5]
```

```
[1] 75 76 77 78 79
```

```
df1$Grade[df1$Grade >= 80]
```

```
[1] 80 81 82 83 84
```

```
df1$Name[df1$Grade >= 80]
```

```
[1] "Danny"   "Pedro"   "Rebecca" "Rubble"  "Ryder"
```

```
df1[df1$Grade >= 80, ]
```

```
      Name Grade
6    Danny    80
7    Pedro    81
8  Rebecca    82
9   Rubble    83
10   Ryder    84
```

```
mean(df1$Grade)
```

```
[1] 79.5
```

0.4 R Packages

To extend the capabilities of R, various packages are developed to handle different tasks: data manipulation, analysis, and visualization.

0.4.1 Base Packages

The base packages providing basic functions and datasets are pre-included with R installation. For example, the package *datasets*, which provides a collection of datasets.

- Try `data()`, which returns the list of datasets within this package.
- To load one of the listed datasets, e.g. mtcars, type `data(mtcars)`. This dataset provides statistics from road tests on multiple models of cars.

0.4.2 Contributed Packages

Many other contributed packages, designed to implement specific operations, are not included with R by default. To utilize these packages, we need to install and load them individually.

- Install a package, type `install.packages("tidyverse")`
- Load the installed package, type `library(tidyverse)`

⚠ Warning

When installing, the package names must be in quotes `" "`. When loading, they are not necessary.

After loading the package, we can use the functions in *tidyverse*. *tidyverse* is a collection of packages, such as *dplyr* and *tidyr* for data manipulation, *ggplot2* for data visualization, *lubridate* for processing time-series data.

- Example with `select` from *dplyr*, type `?select` first to see the R Documentation for this function.

```r
library(tidyverse)

data(mpg)     # new dataset mpg also comes with tidyverse

head(mpg)     # head() displays the first 6 observations of a dataset
```

```
# A tibble: 6 x 11
  manufacturer model displ  year   cyl trans      drv     cty   hwy fl    class
  <chr>        <chr> <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
1 audi         a4      1.8  1999     4 auto(l5)   f        18    29 p     compa~
2 audi         a4      1.8  1999     4 manual(m5) f        21    29 p     compa~
3 audi         a4      2    2008     4 manual(m6) f        20    31 p     compa~
4 audi         a4      2    2008     4 auto(av)   f        21    30 p     compa~
5 audi         a4      2.8  1999     6 auto(l5)   f        16    26 p     compa~
6 audi         a4      2.8  1999     6 manual(m5) f        18    26 p     compa~
```

```r
head(select(mpg, c(manufacturer, model, year)))      # select() selects named variables from a data frame
```

```
# A tibble: 6 x 3
  manufacturer model  year
  <chr>        <chr> <int>
1 audi         a4     1999
2 audi         a4     1999
3 audi         a4     2008
4 audi         a4     2008
5 audi         a4     1999
6 audi         a4     1999
```

```r
head(filter(mpg, year >= 2000))      # filter() subsets a data frame based on defined conditions
```

```
# A tibble: 6 x 11
```

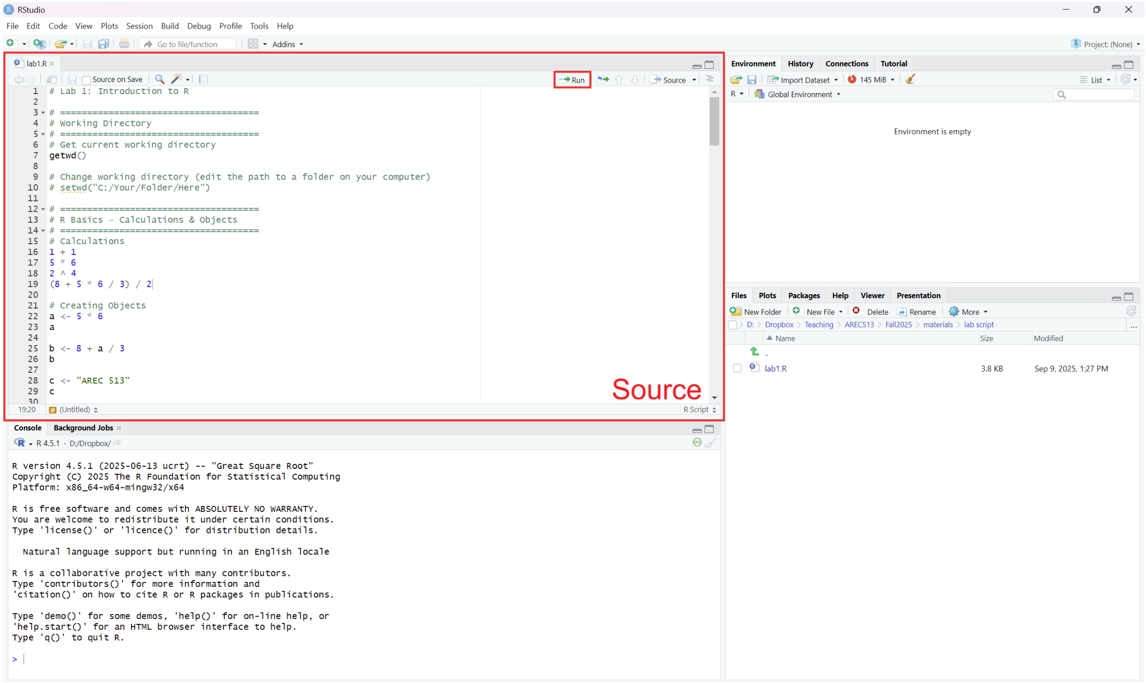
```
  manufacturer model      displ  year   cyl trans  drv     cty   hwy fl    class
  <chr>        <chr>      <dbl> <int> <int> <chr>  <chr> <int> <int> <chr> <chr>
1 audi         a4           2    2008     4 manua~ f        20    31 p     comp~
2 audi         a4           2    2008     4 auto(~ f        21    30 p     comp~
3 audi         a4           3.1  2008     6 auto(~ f        18    27 p     comp~
4 audi         a4 quattro   2    2008     4 manua~ 4        20    28 p     comp~
5 audi         a4 quattro   2    2008     4 auto(~ 4        19    27 p     comp~
6 audi         a4 quattro   3.1  2008     6 auto(~ 4        17    25 p     comp~
```

See how packages simplify the code and make it more intuitive. More details about *tidyverse* will be discussed in Lab 2.

0.5 R Script

R script is simply a text file containing a set of commands and comments, which allows you to save, edit, and execute your code. To create an R script, in the top toolbar, select File > New File > R Script.

Opening an R script creates a new pane to your RStudio, the **Source** pane. In this pane, you can edit and save your code. To run your code, you can hold *Ctrl* and press *Enter*, or click the Run button. This will run through your entire R script, or your selected lines of code.



```
# Lab 1: Introduction to R

# ====================================
# Working Directory
# ====================================
# Get current working directory
getwd()

# Change working directory (edit the path to a folder on your computer)
# setwd("C:/Your/Folder/Here")

# ====================================
# R Basics – Calculations & Objects
# ====================================
# Calculations
1 + 1
5 * 6
2 ^ 4
(8 + 5 * 6 / 3) / 2

# Creating Objects
a <- 5 * 6
a

b <- 8 + a / 3
b

c <- "AREC 513"
c
```

```
R version 4.5.1 (2025-06-13 ucrt) -- "Great Square Root"
Copyright (C) 2025 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

Figure 2: The Source Pane and R Sript

ℹ Note

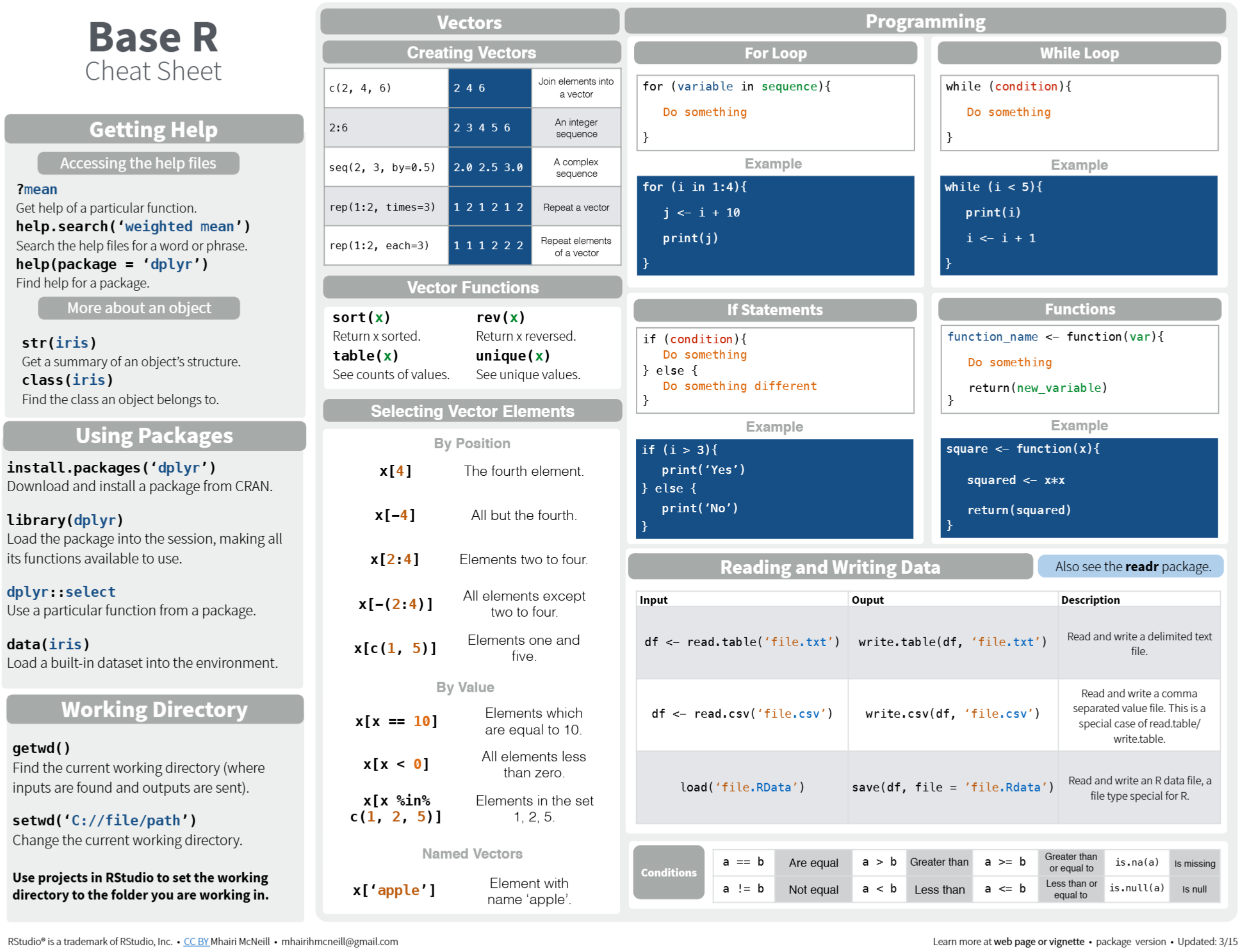
All lines entered in an R script are interpreted as code to be run by R, unless they begin with `#` (Shift + 3). Using `#` is useful for documenting and explaining your code, or for temporarily disabling sections of code.

0.6 Useful Resources

This section lists some useful resources for your exploration of R.

0.6.1 Base R Cheat Sheet

Click here for more information



Base R
Cheat Sheet

Getting Help

Accessing the help files

`?mean`
Get help of a particular function.
`help.search('weighted mean')`
Search the help files for a word or phrase.
`help(package = 'dplyr')`
Find help for a package.

More about an object

`str(iris)`
Get a summary of an object's structure.
`class(iris)`
Find the class an object belongs to.

Using Packages

`install.packages('dplyr')`
Download and install a package from CRAN.

`library(dplyr)`
Load the package into the session, making all its functions available to use.

`dplyr::select`
Use a particular function from a package.

`data(iris)`
Load a built-in dataset into the environment.

Working Directory

`getwd()`
Find the current working directory (where inputs are found and outputs are sent).

`setwd('C://file/path')`
Change the current working directory.

Use projects in RStudio to set the working directory to the folder you are working in.

Vectors

Creating Vectors

`c(2, 4, 6)`	2 4 6	Join elements into a vector
`2:6`	2 3 4 5 6	An integer sequence
`seq(2, 3, by=0.5)`	2.0 2.5 3.0	A complex sequence
`rep(1:2, times=3)`	1 2 1 2 1 2	Repeat a vector
`rep(1:2, each=3)`	1 1 1 2 2 2	Repeat elements of a vector

Vector Functions

`sort(x)` Return x sorted.	`rev(x)` Return x reversed.
`table(x)` See counts of values.	`unique(x)` See unique values.

Selecting Vector Elements

By Position

`x[4]`	The fourth element.
`x[-4]`	All but the fourth.
`x[2:4]`	Elements two to four.
`x[-(2:4)]`	All elements except two to four.
`x[c(1, 5)]`	Elements one and five.

By Value

`x[x == 10]`	Elements which are equal to 10.
`x[x < 0]`	All elements less than zero.
`x[x %in% c(1, 2, 5)]`	Elements in the set 1, 2, 5.

Named Vectors

`x['apple']`	Element with name 'apple'.

Programming

For Loop

```
for (variable in sequence){

   Do something

}
```

Example

```
for (i in 1:4){

   j <- i + 10

   print(j)

}
```

While Loop

```
while (condition){

   Do something

}
```

Example

```
while (i < 5){

   print(i)

   i <- i + 1

}
```

If Statements

```
if (condition){
   Do something
} else {
   Do something different
}
```

Example

```
if (i > 3){
   print('Yes')
} else {
   print('No')
}
```

Functions

```
function_name <- function(var){

   Do something

   return(new_variable)
}
```

Example

```
square <- function(x){

   squared <- x*x

   return(squared)
}
```

Reading and Writing Data

Also see the **readr** package.

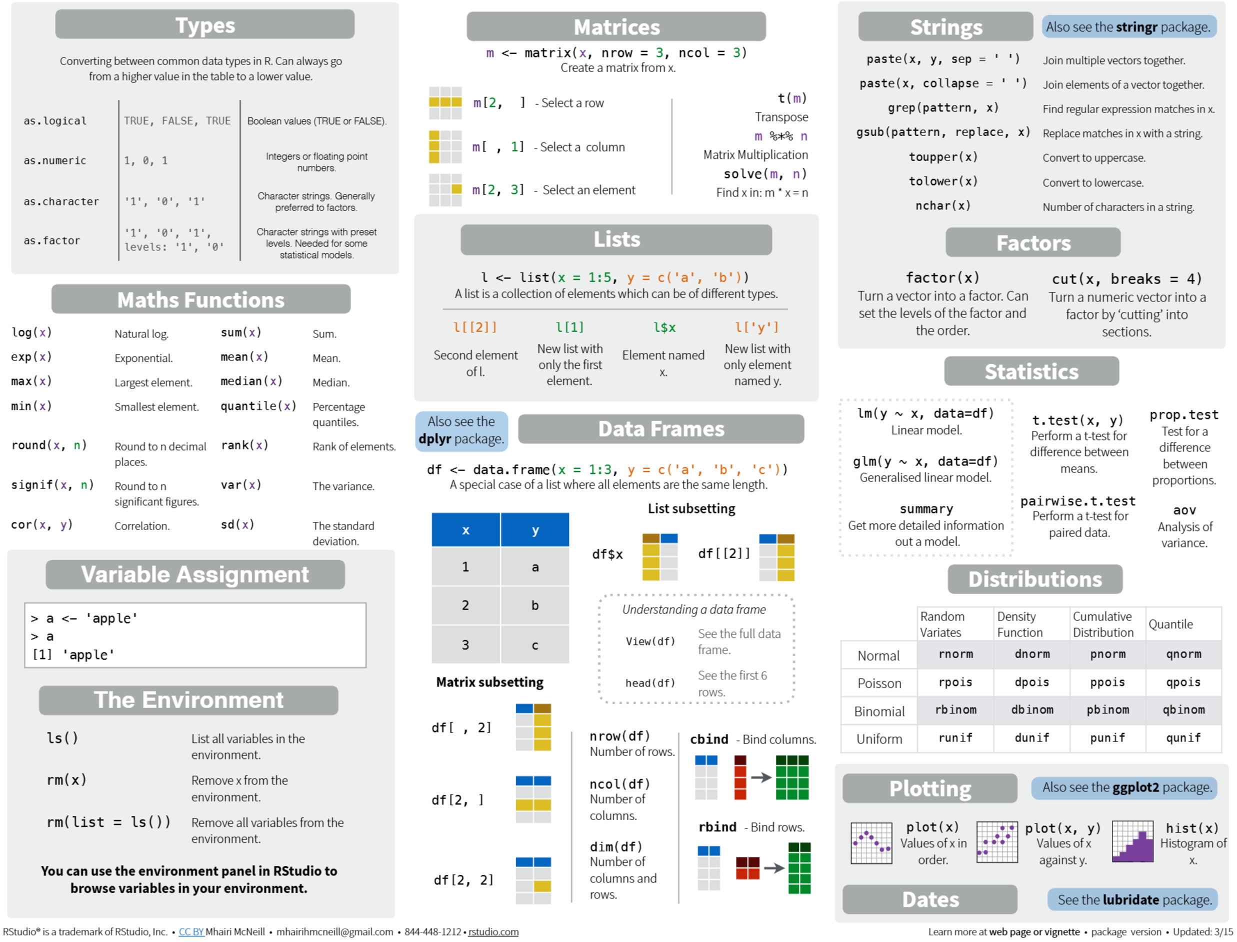
Input	Ouput	Description
`df <- read.table('file.txt')`	`write.table(df, 'file.txt')`	Read and write a delimited text file.
`df <- read.csv('file.csv')`	`write.csv(df, 'file.csv')`	Read and write a comma separated value file. This is a special case of read.table/ write.table.
`load('file.RData')`	`save(df, file = 'file.Rdata')`	Read and write an R data file, a file type special for R.

Conditions

`a == b`	Are equal	`a > b`	Greater than	`a >= b`	Greater than or equal to	`is.na(a)`	Is missing
`a != b`	Not equal	`a < b`	Less than	`a <= b`	Less than or equal to	`is.null(a)`	Is null

RStudio® is a trademark of RStudio, Inc. • CC BY Mhairi McNeill • mhairihmcneill@gmail.com

Learn more at **web page or vignette** • package version • Updated: 3/15



Types

Converting between common data types in R. Can always go from a higher value in the table to a lower value.

`as.logical`	`TRUE, FALSE, TRUE`	Boolean values (TRUE or FALSE).
`as.numeric`	`1, 0, 1`	Integers or floating point numbers.
`as.character`	`'1', '0', '1'`	Character strings. Generally preferred to factors.

`as.factor`	`'1', '0', '1',`
	`levels: '1', '0'`
		Character strings with preset levels. Needed for some statistical models.

Maths Functions

`log(x)`	Natural log.	`sum(x)`	Sum.
`exp(x)`	Exponential.	`mean(x)`	Mean.
`max(x)`	Largest element.	`median(x)`	Median.
`min(x)`	Smallest element.	`quantile(x)`	Percentage quantiles.
`round(x, n)`	Round to n decimal places.	`rank(x)`	Rank of elements.
`signif(x, n)`	Round to n significant figures.	`var(x)`	The variance.
`cor(x, y)`	Correlation.	`sd(x)`	The standard deviation.

Variable Assignment

```
> a <- 'apple'
> a
[1] 'apple'
```

The Environment

`ls()`	List all variables in the environment.
`rm(x)`	Remove x from the environment.
`rm(list = ls())`	Remove all variables from the environment.

You can use the environment panel in RStudio to browse variables in your environment.

Matrices

`m <- matrix(x, nrow = 3, ncol = 3)`
Create a matrix from x.

`m[2, ]` - Select a row

`m[ , 1]` - Select a column

`m[2, 3]` - Select an element

`t(m)`
Transpose

`m %*% n`
Matrix Multiplication

`solve(m, n)`
Find x in: m * x = n

Lists

`l <- list(x = 1:5, y = c('a', 'b'))`
A list is a collection of elements which can be of different types.

`l[[2]]`	`l[1]`	`l$x`	`l['y']`
Second element of l.	New list with only the first element.	Element named x.	New list with only element named y.

Also see the **dplyr** package.

Data Frames

`df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))`
A special case of a list where all elements are the same length.

x	y
1	a
2	b
3	c

List subsetting

`df$x` `df[[2]]`

Understanding a data frame

`View(df)`	See the full data frame.
`head(df)`	See the first 6 rows.

Matrix subsetting

`df[ , 2]`

`df[2, ]`

`df[2, 2]`

`nrow(df)`
Number of rows.

`ncol(df)`
Number of columns.

`dim(df)`
Number of columns and rows.

`cbind` - Bind columns.

`rbind` - Bind rows.

Strings

Also see the **stringr** package.

`paste(x, y, sep = ' ')`	Join multiple vectors together.
`paste(x, collapse = ' ')`	Join elements of a vector together.
`grep(pattern, x)`	Find regular expression matches in x.
`gsub(pattern, replace, x)`	Replace matches in x with a string.
`toupper(x)`	Convert to uppercase.
`tolower(x)`	Convert to lowercase.
`nchar(x)`	Number of characters in a string.

Factors

`factor(x)`
Turn a vector into a factor. Can set the levels of the factor and the order.

`cut(x, breaks = 4)`
Turn a numeric vector into a factor by 'cutting' into sections.

Statistics

`lm(y ~ x, data=df)`
Linear model.

`glm(y ~ x, data=df)`
Generalised linear model.

`summary`
Get more detailed information out a model.

`t.test(x, y)`
Perform a t-test for difference between means.

`pairwise.t.test`
Perform a t-test for paired data.

`prop.test`
Test for a difference between proportions.

`aov`
Analysis of variance.

Distributions

	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	`rnorm`	`dnorm`	`pnorm`	`qnorm`
Poisson	`rpois`	`dpois`	`ppois`	`qpois`
Binomial	`rbinom`	`dbinom`	`pbinom`	`qbinom`
Uniform	`runif`	`dunif`	`punif`	`qunif`

Plotting

Also see the **ggplot2** package.

`plot(x)`
Values of x in order.

`plot(x, y)`
Values of x against y.

`hist(x)`
Histogram of x.

Dates

See the **lubridate** package.

RStudio® is a trademark of RStudio, Inc. • CC BY Mhairi McNeill • mhairihmcneill@gmail.com • 844-448-1212 • rstudio.com

Learn more at **web page or vignette** • package version • Updated: 3/15

0.6.2 An Introduction to R

An Introduction to R, by Venables, W. N., Smith, D. M., & R Development Core Team

0.7 Exercise

Load the dataset “mpg” and work through the exercises below. Note, “mpg” is included in the tidyverse package, so you will need to load the package first.

1. Calculate the **mean**, **range**, **minimum**, and **maximum** of the variable “hwy” across all models. Then, combine these statistics into one vector. (Tip: look up the RDocumentation for the functions `mean`, `range`, `min`, and `max`).
2. Since “hwy” is measured in miles per gallon, create a new variable in mpg that expresses “hwy” in litres per 100 km.
3. Identify the models of cars that are most fuel-efficient. Which classes of cars are least fuel-efficient? (Tip: to remove duplicates, use the function `unique`).
4. Compute the quantiles of “hwy”. Can you also calculate the tertiles instead? (Tip: look up for the RDocumentation for the function `quantile`).
5. Now, based on the tertiles you calculated, assign “least efficient”, “medium”, and “most efficient” labels to all models. Try using both base R indexing functions and the function `ifelse`.