Introduction to R - Part II

Matt Arthur

GradQuant
University of California Riverside

06 Oct 2022

Outline

Introduction

R is an open source software used for statistical computing and graphics.

- Easy to download, with no license restrictions.
- Provides support for powerful computing tasks, such as machine learning.
- Intuitive and flexible graphics support.
- Works well with other programs; i.e, SQL and Excel

Installing R

Go to `https://www.r-project.org`

- Click the "Download" link on the left.
- Select a mirror from which to download.
- Follow the download instructions for your operating system.

Installing R Studio

R Studio is a graphical user interface that makes R much easier to use.

To download R Studio, go to `https://www.rstudio.com`.

- R must be installed for R Studio to run.
- Go to Products > R Studio
- Follow the instructions for downloading the free version of R Studio Desktop.

Using R as a Calculator

One of the simplest ways to use R is as a calculator:

```
>  2 + 2
[1] 4
>  2 ** 3
[1] 8
>  2 + (3 ** 2 - sin(46))
[1] 10.09821
>  exp(1.3)
[1] 3.669297
```

R uses standard order of operations and has many predefined functions which make it easy to use in this way.

Testing for Equality

We can also use R to conduct logical tests:

```
>  2 == 3
[1] FALSE
>  2 < 3
[1] TRUE
```

As in other languages, we use "$==$" instead of "$=$" to test for equality.

Data Structures in R

R can store and use objects that we create in **variables**

```
> x <- 2 * 3
> print(x)
[1] 6
```

R can also store sequences of values, called **vectors**

```
> y <- c(2, 4, 6, 8)
> print(y)
[1] 2 4 6 8
> z <- 2:4
> print(z)
[1] 2 3 4
```

Note: The use of <- as an assignment operator in R.

We can access all or part of vectors defined in R by using square brackets: []

```
> y[1]
[1] 2
> y[2:4]
[1] 4 6 8
```

Vector Operations

Most mathematical operations that can be applied to scalars can also be applied to vectors in R:

- Operators to vectors are applied elementwise:

```
> x <- c(1, 2, 3, 4)
> y <- c(2, 4, 6, 8)
> print(x + y)
[1]  3  6  9 12
```

- When a shorter vector is combined with a longer vector, elements in the shorter vector are **recycled**:

```
> z <- c(1, 4)
> print(y + z)
[1]  3  8  7 12
```

Vector Operations

- Single numbers can be thought of as vectors of length-1

```
> y <- c(2, 4, 6, 8)
> y * 2
[1]  4  8 12 16
```

Matrices

A matrix is essentially a 2D array in R. Any vector can be put into matrix form using the `matrix()` function:

- Matrices can be ordered in different directions:

```
> m1 <- matrix(c(1:9), nrow = 3, ncol = 3)
> print(m1)
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> m2 <- matrix(c(1:9), nrow = 3, ncol = 3, byrow = T)
> print(m2)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

Matrices

- We can access matrix components using again using square brackets:

```
> m1[1,3]
[1] 7
> m2[,3]
[1] 3 6 9
> m1[1,]
[1] 1 4 7
```

Note: Vectors and matrices are indexed starting at 1.

Matrix Operations

- Elementwise multiplication:

```
> m1 * m2
     [,1] [,2] [,3]
[1,]    1    8   21
[2,]    8   25   48
[3,]   21   48   81
```

- Matrix Multiplication

```
> m1 %*% m2
     [,1] [,2] [,3]
[1,]   66   78   90
[2,]   78   93  108
[3,]   90  108  126
```

Matrix Operations

- Transpose

```
>  t(m1)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

- Diagonal Matrices

```
>  diag(c(1, 2, 3))
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    2    0
[3,]    0    0    3
```

Matrix Operations

- Inverse of a square Matrix:

```
> A <- matrix(c(1, 3, 5, 7), nrow = 2)
> solve(A)
       [,1]   [,2]
[1,] -0.875  0.625
[2,]  0.375 -0.125
```

- Row and Column Means

```
> rowMeans(A)
[1] 3 5
> colMeans(A)
[1] 2 6
```

The Working Directory

A working directory is the default location where R looks for files.

- To view your current working directory, use the `getwd()` function:

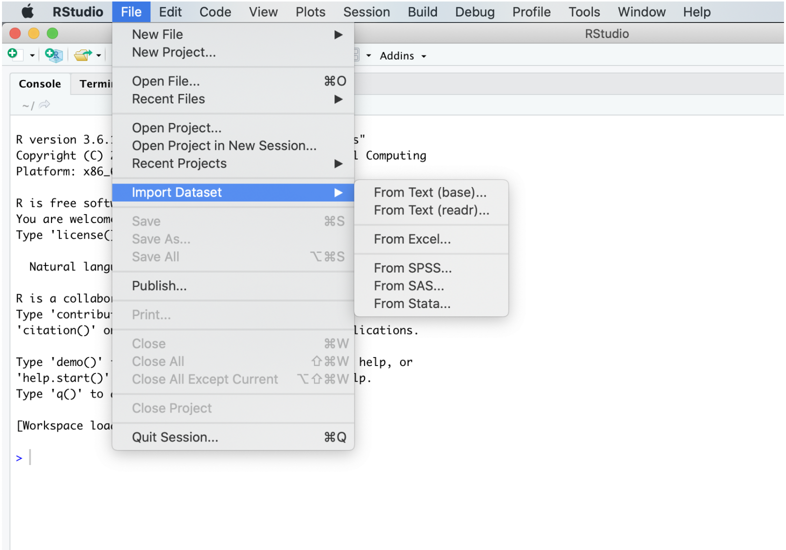
  ```
  > getwd()
  ```

- To change your working directory, use `setwd()`:

  ```
  > setwd("desired.path")
  ```

Importing Data: Using the File Tab

R Studio has a File tab that can be used to import data files:



Importing Data: Using R Directly

Alternatively, you can read data by using commands directly from R:

- From a Comma-Delimited File:

  ```
  > new_data <- read.table('your_file_path', sep = ',', heade
  ```

  or,

  ```
  > new_data <- read.csv('your_file_path', header = T)
  ```

- From Excel:

  ```
  > library(readxl)
  > new_data <- read_excel('your_file_path')
  ```

- From SPSS:

  ```
  > library(foreign)
  > new_data <- read.spss('your_file_path', to.data.frame =
  ```

Package Management

Many data-import utilities reside in auxiliary libraries which are not loaded in base R. To install these libraries, use the `install.packages()` function:

```
> install.packages("readxl")
```

R may need to be restarted prior to loading the packages you installed.

Outline

Concrete Example

We will do several hands-on exercises with the "Concrete Compressive Strength" dataset from the UC Irvine Machine Learning Data Repository[1]. You may access the data using the following link:

- `https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength`

Concrete Example

To download the data:

- Navigate to the link on the previous slide
- Click on the “Data Folder” link at the top of the page
- Download `Concrete_Data.xls`

Concrete Example

To load the data into R: First determine the directory to which you saved the Concrete Data.

- Usually will be in your "Downloads" folder
- "∼/Downloads" on Unix
- Something like "C:/Users/your-user-name/Downloads" on Windows

```
> library(readxl)
> concrete <- read_excel("~/Downloads/Concrete_Data.xls")
> View(concrete)
```

Exercises for Next Time

Next week, we'll continue working with the Concrete Dataset in R:

- Basic viewing/summarizing of the data
- Running a linear regression in R to predict concrete compressive strength
- Managing tabular data

Read in the Concrete Data

We read in our data using the `read_excel()` function, and we inspect the data using the `View()` function:

```
> concrete <- read_excel("~/Downloads/Concrete_Data.xls")
> concrete <- data.frame(concrete)
> View(concrete)
```

Read in the Concrete Data

Housekeeping: it will be good to choose shorter names for our columns. We can accomplish this using the `names()` function:

```
> names(concrete) <- c(
+       "cement", "slag", "ash", "water",
+       "plast", "aggr_c", "aggr_f", "age",
+       "strength"
+ )
> head(concrete)
  cement  slag ash water plast aggr_c aggr_f age strength
1  540.0   0.0   0   162   2.5 1040.0  676.0  28 79.98611
2  540.0   0.0   0   162   2.5 1055.0  676.0  28 61.88737
3  332.5 142.5   0   228   0.0  932.0  594.0 270 40.26954
4  332.5 142.5   0   228   0.0  932.0  594.0 365 41.05278
5  198.6 132.4   0   192   0.0  978.4  825.5 360 44.29608
6  266.0 114.0   0   228   0.0  932.0  670.0  90 47.02985
```

Read in the Concrete Data

Inspect the dimensions and first few rows of the dataset using the `nrow()`, `ncol()`, and `head()` functions:

```
> nrow(concrete)
[1] 1030
> ncol(concrete)
[1] 9
> head(concrete)
  cement  slag ash water plast aggr_c aggr_f age strength
1  540.0   0.0   0   162   2.5 1040.0  676.0  28 79.98611
2  540.0   0.0   0   162   2.5 1055.0  676.0  28 61.88737
3  332.5 142.5   0   228   0.0  932.0  594.0 270 40.26954
4  332.5 142.5   0   228   0.0  932.0  594.0 365 41.05278
5  198.6 132.4   0   192   0.0  978.4  825.5 360 44.29608
6  266.0 114.0   0   228   0.0  932.0  670.0  90 47.02985
```

Read in the Concrete Data

Index Data Frames just like Matrices:

```
> concrete[1,1]
[1] 540
> head(concrete[1,])
  cement slag ash water plast aggr_c aggr_f age strength
1    540    0   0   162   2.5   1040    676  28 79.98611
> head(concrete[,1])
[1] 540.0 540.0 332.5 332.5 198.6 266.0
> head(concrete[,"cement"])
[1] 540.0 540.0 332.5 332.5 198.6 266.0
```

Filter the Concrete Data

Or, index columns using string vectors:

```
> interesting_columns <- c("cement", "strength")
> head(concrete[, interesting_columns])
  cement strength
1  540.0 79.98611
2  540.0 61.88737
3  332.5 40.26954
4  332.5 41.05278
5  198.6 44.29608
6  266.0 47.02985
```

Filter the Concrete Data

Subset row data using boolean statements for filtering. Remember: 2 == 3 returns FALSE:

```
> head(concrete[,"ash"]) == 0
[1] TRUE TRUE TRUE TRUE TRUE TRUE
> interesting_rows <- concrete[, "ash"] != 0
> interesting_data_frame <- concrete[interesting_rows, interes
> head(interesting_data_frame)
    cement strength
185 222.36 11.57630
186 222.36 24.44882
187 222.36 24.89008
188 222.36 29.44752
189 222.36 40.71356
190 233.81 10.38351
```

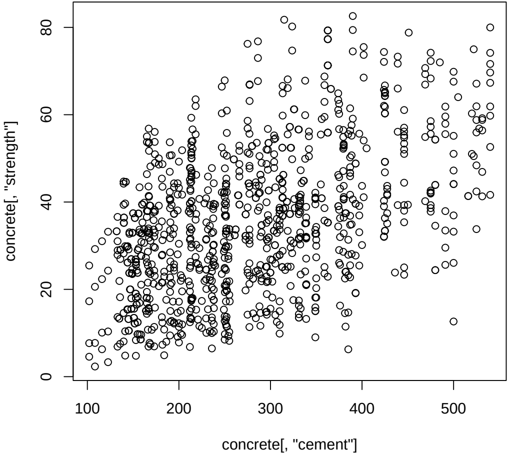
Plotting the Concrete Data

Plot data using the `plot()` function. This function takes two arguments:

- `x` = a vector of x-values
- `y` = a vector of y-values

Plotting the Concrete Data

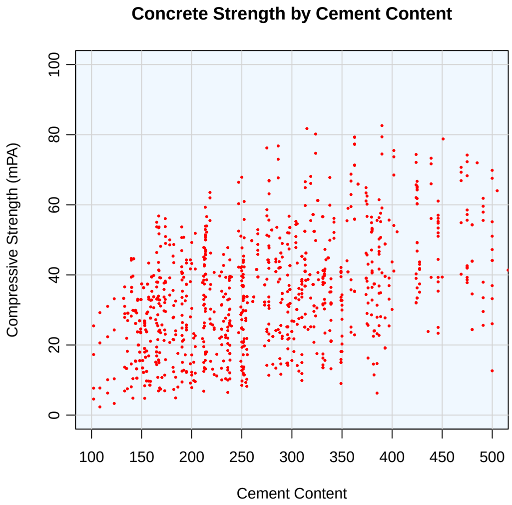
```
> plot(x = concrete[,"cement"], y = concrete[,"strength"])
```



Fancy Plotting

```
> par(bg = "white")
> plot(x = NULL, y = NULL, xaxt = "n", yaxt = "n",
+       xlim = c(100, 500), ylim = c(0, 100),
+       xlab = "Cement Content",
+       ylab = "Compressive Strength (mPA)",
+       main = "Concrete Strength by Cement Content")
> rect(par("usr")[1], par("usr")[3], par("usr")[2],
+       par("usr")[4], col = "aliceblue")
> axis(side = 1, at = seq(100, 500, by = 50), labels = T,
+       hadj = 0.5)
> axis(side = 2, at = seq(0, 100, by = 20), labels = T,
+       padj = 0.5, gap.axis = 0.10)
> abline(h = seq(0, 100, by = 20), col = "lightgrey")
> abline(v = seq(100, 500, by = 50), col = "lightgrey")
> points(x = concrete$cement, y = concrete$strength,
+       pch = 19, cex = 0.3, col = "red")
```

Fancy Plotting - Results



Model the Concrete Data

Linear Regression is accomplished using the `lm()` function:

```
> my_model <- lm(strength ~ cement, data = concrete)
> summary(my_model)
```

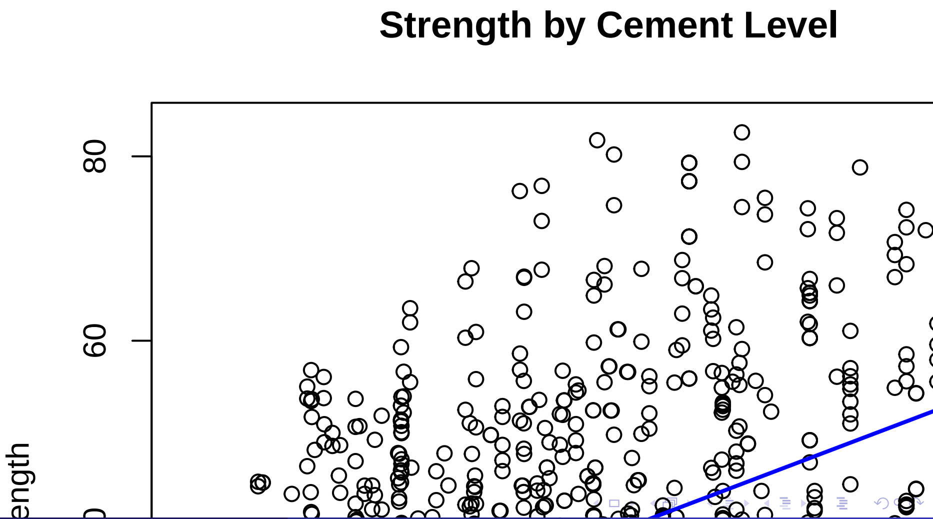
	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	13.4428	1.2969	10.37	0.0000
cement	0.0796	0.0043	18.40	0.0000

Adding the Trendline

Add a trendline using the "coefficients" vector, stored in the linear model object:

```
> print(my_model$coefficients)
(Intercept)      cement
13.44279487  0.07957957
> plot(concrete$cement, concrete$strength,
+      xlab = "Cement", ylab = "Strength",
+      main = "Strength by Cement Level")
> abline(a = my_model$coefficients[1],
+      b = my_model$coefficients[2],
+      lwd = 2, col = "blue")
```

Adding the Trendline



Outline

Exercises

Consider the following exercises:

1. Run a linear regression with the Concrete Data. This time, use `water` as the response variable and `strength` as the predictor. Calculate the regression coefficients and R^2.
2. Run the same regression as above, but this time only include rows for which `slag > 0`
3. Make a scatterplot of `strength` against `water`. Add both trend lines from (1) and (2) above.

Solution to Exercises

First, take note of the rows for which `slag > 0`:

```
> p_slag <- concrete[, "slag"] > 0
```

For (1), just repeat the previous set of code, but use `water` as the response and `strength` as the predictor:

```
> w_resp_1 <- lm(water ~ strength, data = concrete)
> w_resp_2 <- lm(water ~ strength,
+       data = concrete[p_slag,])
```

For (1), inspect the output of `w_resp_1`:

```
> summary(w_resp_1)
> library(xtable)
> print(xtable(w_resp_1))
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	194.8270	1.5081	129.19	0.0000
strength	-0.3702	0.0382	-9.70	0.0000

Solution to Exercises

For (2), same thing with `w_resp_2`:

```
> summary(w_resp_2)
```

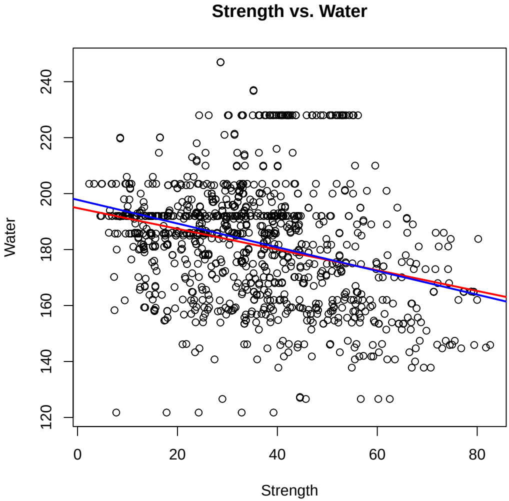
	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	197.7776	2.3501	84.16	0.0000
strength	-0.4236	0.0547	-7.74	0.0000

For plotting, use the `plot()` function, with two uses of `abline(...)`.

Solution to Exercises

```
> plot(x = concrete[,"strength"], y = concrete[,"water"],
+      main = "Strength vs. Water", xlab = "Strength",
+      ylab = "Water")
> abline(a = w_resp_1$coefficients["(Intercept)"],
+      b = w_resp_1$coefficients["strength"],
+      col = "red", lwd = 2)
> abline(a = w_resp_2$coefficients["(Intercept)"],
+      b = w_resp_2$coefficients["strength"],
+      col = "blue", lwd = 2)
```

Solution to Exercises



Outline

Bonus: Functions in R

So far, we have encountered many R Functions: `c()`, `matrix()`, `plot()`, `lm()`, and more!

R allows user-defined functions. The basic syntax is as follows:

```
> my_function <- function(arg1, arg2, ...) {
+       # make a difference
+       # in the world
+       # with arg1, arg2, ...
+       # return(a_value)
+ }
```

Bonus: Functions in R

In the example below, the function `f()` takes two arguments, and returns the sum of the arguments:

```
> f <- function(a, b) {
+     s <- a + b
+     return(s)
+ }
> f(3,2)
[1] 5
> f(4,7)
[1] 11
```

Bonus: Functions in R

We could also define a function to do more complicated tasks. The following function runs a simple linear regression using Concrete Compressive Strength as a response variable, and the function argument as a predictor:

```
> reg <- function(predictor) {
+     model <- lm(concrete[,"strength"] ~
+         concrete[,predictor])
+     s <- summary(model)
+     return(s$r.squared)
+ }
> reg("cement")
[1] 0.2478374
> reg("slag")
[1] 0.01817763
```

Related Workshops

If you are interested in learning more about R, please consider the following workshops, which are offered later this quarter:

- Linear Models in R: Monday 10/3 & 10/10: 2:00PM - 2:50PM
- Data Visualization in R: Tuesday 10/11 & 10/18: 1:00PM - 1:50PM
- Remember to register!

Credits

This presentation was adapted from a previous workshop delivered by Lead Consultant Ruihan Lu in 2020.

The Concrete Data were owned and donated to the UCI Machine Learning Repository by I-Cheng Yeh:

I-Cheng Yeh.
Modeling of strength of high performance concrete using artificial neural networks.
Cement and Concrete Research, 28(12):1797–1808, 1998.

Questions?