Lunch Hour Learning Guide, Session 2, Spring 2025

Importing and Indexing Data in R

Sean Morey Smith

2025-01-22

What You Will Learn

- Understanding data types and data structures
- Importing data from a delimited file (standard csv format, with variations)
- Viewing data
- Indexing (accessing) parts of a data frame

Before Starting

Create a new, self-contained R project in your chosen sub-directory, where you will store your work from this session. For guidance, see the instructions from Session 1.

Create a sub-directory called "data" in your project directory. Save the experiment_data.csv, no_header.csv, and data_file.txt files in your "data" directory. They are available in a zip at https://library.rice.edu/sites/default/files/materials/data.zip

Overview of Data Types and Data Structures

R is flexible and can be used with various types of data. In our sessions, we will just work with tabular data, which is what you typically find in a spreadsheet.

First, here is some terminology regarding data types, which describe the quality of values in a variable:

- *double*: numeric data that involve decimal points; AKA "float"
- *integer*: numeric data that are whole numbers
- *character*: non-numeric data encoded as alphanumeric characters; AKA "string"
- *logical*: non-numeric data encoded as `FALSE`, `TRUE`, or `NA` (missing); AKA "Boolean"

Two other data types are *complex* and *raw*, but we won't be using those.

Second, there are a few basic ways that R organizes data into structures. The ones you need to know for now include:

- *vector*: a one-dimensional structure consisting of one or more values of the same data type. The position of each value is meaningful, and the values can have names.
- *data frame*: a rectangular structure (like a spreadsheet) in which rows generally represent observations (e.g., respondents, participants, level of analysis) and columns generally represent variables. Data within each column must be the same data type, but various data types can occur across columns.
- *factor*: a special type of vector, in which each value represents a level of the associated variable. A factor is useful for categorical data or grouping variables.

Importing Data

Most of the time (but not always), data files will be in a tabular data format, with rows and columns. Often, this takes the form of a comma-separated values (csv) file, which you can import or "read in" by using the `read.csv()` function.

Note that, in the dataset we'll be using, we have two categorical variables that we want to treat as factors: "group" and "major". We will include an argument in the `read.csv()` function to treat those variables (which are encoded as character data) as factors.

```
scores <- read.csv("data/experiment_data.csv", stringsAsFactors = TRUE)
```

Note that we have assigned the output of the function `read.csv` and its arguments (the file name and the specification to treat character data as factors) to the variable `scores`, which is a data frame. The default is to read the first row as column headings.

The `scores` data frame object appears in the top-right Global Environment pane; it has 20 observations of 9 variables.

Sometimes you will have a data file that is not in a csv format but is in another format (such as a tab-delimited file). Alternatively, you may have a file that doesn't have column headings. Here are a few tips for reading those files:

- Use `read.table()`.
- Specify the nature of the data delimiter (e.g., "\t" for a tab delimiter or "|" for a pipe delimiter) with `sep`.
- Specify whether the first row contains column headings or not with `header` and supply your own with `col.names`.

Example of reading data from a tab-delimited file:

```
alt_data <- read.table("data/data_file.txt", sep = "\t")
```

Example of reading data from a csv file that has no header row:

```
alt_data2 <- read.table("data/no_header.csv",
                        sep = ",",
                        header = FALSE,
                        col.names = c("ID", "group", "pretest", "posttest"))
```

Viewing Data

There are several ways to look at your data: spreadsheet view, structure, first or last few rows, and summary. Let's go through each function and its output!

```
View(scores)
```

`View()` shows the data frame as a spreadsheet in a new tab in the scripting area.

```
str(scores)
```

```
## 'data.frame':    20 obs. of  9 variables:
##  $ ID           : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ group        : Factor w/ 2 levels "Control","Treat": 2 2 2 2 2 2 2 2 2 2 ...
##  $ pretest      : int  60 55 82 74 69 90 88 68 76 80 ...
##  $ posttest     : int  80 72 95 88 83 96 96 86 89 92 ...
##  $ trait_anxiety: int  2 18 6 10 12 3 2 12 9 1 ...
##  $ difference   : int  20 17 13 14 14 6 8 18 13 12 ...
##  $ major        : Factor w/ 3 levels "H","NS","SS": 3 2 1 3 2 1 3 2 1 3 ...
##  $ state_anxiety: int  20 26 5 12 6 1 4 10 11 5 ...
##  $ math_score   : int  40 35 90 70 92 97 90 88 86 92 ...
```

`str()` shows the structure of the data frame, including each variable, the type of data it contains, and the first few values.

```
head(scores)
```

```
##   ID group pretest posttest trait_anxiety difference major state_anxiety
## 1  1 Treat      60       80             2         20    SS            20
## 2  2 Treat      55       72            18         17    NS            26
## 3  3 Treat      82       95             6         13     H             5
## 4  4 Treat      74       88            10         14    SS            12
## 5  5 Treat      69       83            12         14    NS             6
## 6  6 Treat      90       96             3          6     H             1
##   math_score
## 1         40
## 2         35
## 3         90
## 4         70
## 5         92
## 6         97
```

`head()` shows all variables for the first few rows.

```
tail(scores, n = 3)
```

```
##    ID   group pretest posttest trait_anxiety difference major state_anxiety
## 18 18 Control      62       75            11         13     H            18
## 19 19 Control      77       87             6         10    SS            11
## 20 20 Control      81       93             4         12    NS             6
##    math_score
## 18         55
## 19         77
## 20         84
```

`tail()` shows all variables for the last few rows.
Note that we can specify the number of rows we want to see with `head()` or `tail()` by using the `n =` argument.

```
summary(scores)
```

```
##        ID            group       pretest         posttest     trait_anxiety
```

```
##  Min.   : 1.00   Control:10   Min.   :55.00   Min.   :66.00   Min.   : 1.00
##  1st Qu.: 5.75   Treat  :10   1st Qu.:63.50   1st Qu.:79.25   1st Qu.: 3.75
##  Median :10.50                Median :75.00   Median :86.50   Median : 7.50
##  Mean   :10.50                Mean   :73.65   Mean   :84.80   Mean   : 8.00
##  3rd Qu.:15.25                3rd Qu.:81.25   3rd Qu.:92.25   3rd Qu.:11.25
##  Max.   :20.00                Max.   :92.00   Max.   :96.00   Max.   :18.00
##    difference    major  state_anxiety    math_score
##  Min.   : 4.00   H :6   Min.   : 1.0   Min.   :32.00
##  1st Qu.: 6.75   NS:7   1st Qu.: 5.0   1st Qu.:52.75
##  Median :12.00   SS:7   Median :11.0   Median :80.50
##  Mean   :11.15          Mean   :12.4   Mean   :71.60
##  3rd Qu.:13.25          3rd Qu.:20.0   3rd Qu.:90.00
##  Max.   :20.00          Max.   :28.0   Max.   :97.00
```

`summary()` is a useful function for obtaining some summary statistics (min, max, 1st and 3rd quartiles, mean, and median) of each numeric variable. Note that this does not provide any measures of variability; however, you can easily calculate the interquartile range by subtracting Q1 from Q3.

Note that the two factors are summarized as the frequency of observations in each level of the factor.

Determining Variable Type

Although `str()` will give you the data type of each variable in your data frame, sometimes you might want to know the type of a specific variable without having to look at all the variables at once. (This is especially true when you are working with very large datasets with lots of variables.) The function `typeof()` is useful for this purpose:

```
typeof(scores$pretest)
```

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

```
typeof(scores$group)
```

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

The output for `scores$pretest` is not too surprising; the variable contains integer data because all of the values are whole numbers. However, why is `scores$group` integer, when we know that the possible values are "Treat" and "Control"? This is because R codes those two categories "behind the scenes" as integers (in this case, 1s and 2s).

If we instead wanted to check whether a variable is a factor or not, we can use the function `class()`, like this:

```
class(scores$pretest)
```

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

```
class(scores$group)
```

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

See that `pretest` is an integer but `group` is a factor.

Use `typeof()` to see what R is doing internally but `class()` to see how R expects you to interact with the data.

Indexing a Data Frame

Indexing, also known as subsetting or extracting, involves accessing specific pieces of a data frame. Each value has an index, or a position, in the data frame; the position is represented by its row number and its column number (or name).

Note that indexing begins with 1 (rather than 0).

To index a specific value, indicate the row number and column number in square brackets, separated by a comma. For example, index the math score for the first participant. This corresponds to the first row, ninth column:

```
scores[1,9]
```

```
## [1] 40
```

Practice: Index the trait anxiety score of participant 20. *Solution: Note that we have to check which column contains trait* ***anxiety*** *- column 5.*

```
scores[20, 5]
```

```
## [1] 4
```

To index a specific whole row (all variables), specify the row number and leave a blank after the comma. Perhaps we want all scores for participant 1 (i.e., row 1):

```
scores[1, ]
```

```
##   ID group pretest posttest trait_anxiety difference major state_anxiety
## 1  1 Treat      60       80             2         20    SS            20
##   math_score
## 1         40
```

Practice: Index all data for participant 2. Then index all data for participant 12. Add an explanatory comment to each code.

Solution:

```
scores[2, ] # All data for participant 2
```

```
##   ID group pretest posttest trait_anxiety difference major state_anxiety
## 2  2 Treat      55       72            18         17    NS            26
##   math_score
## 2         35
```

```
scores[12, ] # All data for participant 12
```

```
##    ID   group pretest posttest trait_anxiety difference major state_anxiety
## 12 12 Control      58       70            14         12     H            25
##    math_score
## 12         38
```

To index a specific variable (all rows), leave a blank before the comma and specify the column number. Example: Index the fourth column, all rows.

```
scores[, 4]
```

```
##  [1] 80 72 95 88 83 96 96 86 89 92 85 70 66 90 96 80 77 75 87 93
```

Practice: Index all difference scores. Then index all math scores. Add an explanatory comment to each code.

Solution:

```
scores[ , 6] # All difference scores
```

```
##  [1] 20 17 13 14 14  6  8 18 13 12  6 12  6  5  4  7 13 13 10 12
```

```
scores[ , 9] # All math scores
```

```
##  [1] 40 35 90 70 92 97 90 88 86 92 46 38 32 90 95 75 60 55 77 84
```

To index a series of rows, use a colon between the first and last index in the series.
Example: Index the first five rows, all columns.

```
scores[1:5, ]
```

```
##   ID group pretest posttest trait_anxiety difference major state_anxiety
## 1  1 Treat      60       80             2         20    SS            20
## 2  2 Treat      55       72            18         17    NS            26
## 3  3 Treat      82       95             6         13     H             5
## 4  4 Treat      74       88            10         14    SS            12
## 5  5 Treat      69       83            12         14    NS             6
##   math_score
## 1         40
## 2         35
## 3         90
## 4         70
## 5         92
```

We can index a series of columns the same way.
If we want all rows, columns 2 through 4:

```
scores[, 2:4]
```

```
##      group pretest posttest
## 1    Treat      60       80
## 2    Treat      55       72
## 3    Treat      82       95
## 4    Treat      74       88
## 5    Treat      69       83
## 6    Treat      90       96
## 7    Treat      88       96
## 8    Treat      68       86
## 9    Treat      76       89
```

```
## 10   Treat      80       92
## 11 Control      79       85
## 12 Control      58       70
## 13 Control      60       66
## 14 Control      85       90
## 15 Control      92       96
## 16 Control      73       80
## 17 Control      64       77
## 18 Control      62       75
## 19 Control      77       87
## 20 Control      81       93
```

As you may have experienced, it can be challenging remembering what number each column corresponds to; thus, it is often more convenient to refer to columns by the variable name. You can index using variable name instead. Be sure to wrap it in quotation marks.
For example, the pretest scores for the first ten participants:

```
scores[1:10, "pretest"]
```

```
##  [1] 60 55 82 74 69 90 88 68 76 80
```

For multiple columns by name, use the concatenate (`c()`) function to gather the variable names together. For example, the "pretest" and "posttest" scores of the 11th through 20th participants:

```
scores[11:20, c("pretest", "posttest")]
```

```
##    pretest posttest
## 11      79       85
## 12      58       70
## 13      60       66
## 14      85       90
## 15      92       96
## 16      73       80
## 17      64       77
## 18      62       75
## 19      77       87
## 20      81       93
```

Alternatively, you can use dollar sign notation to index a column:

```
scores$pretest[1:10]
```

```
##  [1] 60 55 82 74 69 90 88 68 76 80
```

This will return the first 10 rows of pretest scores.

Indexing by Citeria

Remember the logical operators: "==", ">=", "<=", "!=", "&", "|"? These can be used to index by specific criteria.
Example: Find all rows that have a pretest score of greater than or equal to 70:

```
scores[scores$pretest >= 70, ]
```

```
##    ID   group pretest posttest trait_anxiety difference major state_anxiety
## 3   3   Treat      82       95             6         13     H             5
## 4   4   Treat      74       88            10         14    SS            12
## 6   6   Treat      90       96             3          6     H             1
## 7   7   Treat      88       96             2          8    SS             4
## 9   9   Treat      76       89             9         13     H            11
## 10 10   Treat      80       92             1         12    SS             5
## 11 11 Control      79       85             4          6    NS            20
## 14 14 Control      85       90             6          5    NS             2
## 15 15 Control      92       96             2          4     H             3
## 16 16 Control      73       80             9          7    SS            15
## 19 19 Control      77       87             6         10    SS            11
## 20 20 Control      81       93             4         12    NS             6
##    math_score
## 3          90
## 4          70
## 6          97
## 7          90
## 9          86
## 10         92
## 11         46
## 14         90
## 15         95
## 16         75
## 19         77
## 20         84
```

Practice: Index all treatment group participants with trait anxiety scores < 10. *Solution: We have to use a logical operator (&) for this one!*

```
scores[(scores$group == "Treat") & (scores$trait_anxiety < 10), ]
```

```
##    ID group pretest posttest trait_anxiety difference major state_anxiety
## 1   1 Treat      60       80             2         20    SS            20
## 3   3 Treat      82       95             6         13     H             5
## 6   6 Treat      90       96             3          6     H             1
## 7   7 Treat      88       96             2          8    SS             4
## 9   9 Treat      76       89             9         13     H            11
## 10 10 Treat      80       92             1         12    SS             5
##    math_score
## 1          40
## 3          90
## 6          97
## 7          90
## 9          86
## 10         92
```

The use of `&` allows you to specify two or more criteria.

Use not equals - with the exclamation point - to index all values except the one(s) specified.
Example: We want all participants with a "posttest" score that was not 93.

```
scores[scores$posttest != 93, ]
```

```
##    ID   group pretest posttest trait_anxiety difference major state_anxiety
## 1   1   Treat      60       80             2         20    SS             20
## 2   2   Treat      55       72            18         17    NS             26
## 3   3   Treat      82       95             6         13     H              5
## 4   4   Treat      74       88            10         14    SS             12
## 5   5   Treat      69       83            12         14    NS              6
## 6   6   Treat      90       96             3          6     H              1
## 7   7   Treat      88       96             2          8    SS              4
## 8   8   Treat      68       86            12         18    NS             10
## 9   9   Treat      76       89             9         13     H             11
## 10 10   Treat      80       92             1         12    SS              5
## 11 11 Control      79       85             4          6    NS             20
## 12 12 Control      58       70            14         12     H             25
## 13 13 Control      60       66            18          6    SS             28
## 14 14 Control      85       90             6          5    NS              2
## 15 15 Control      92       96             2          4     H              3
## 16 16 Control      73       80             9          7    SS             15
## 17 17 Control      64       77            11         13    NS             20
## 18 18 Control      62       75            11         13     H             18
## 19 19 Control      77       87             6         10    SS             11
##    math_score
## 1          40
## 2          35
## 3          90
## 4          70
## 5          92
## 6          97
## 7          90
## 8          88
## 9          86
## 10         92
## 11         46
## 12         38
## 13         32
## 14         90
## 15         95
## 16         75
## 17         60
## 18         55
## 19         77
```

If the criterion is a character data type, specify it in quotation marks.
Example - Obtain all data for “H” (Humanities) majors:

```
scores[scores$major == "H", ]
```

```
##    ID   group pretest posttest trait_anxiety difference major state_anxiety
## 3   3   Treat      82       95             6         13     H              5
## 6   6   Treat      90       96             3          6     H              1
## 9   9   Treat      76       89             9         13     H             11
```

```
## 12 12 Control      58       70            14          12     H            25
## 15 15 Control      92       96             2           4     H             3
## 18 18 Control      62       75            11          13     H            18
##    math_score
## 3          90
## 6          97
## 9          86
## 12         38
## 15         95
## 18         55
```

If the criterion involves more than one value, use `%in%` and `c()`.
Example: Obtain all data for both "SS" (Social Science) and "NS" (Natural Science) majors:

```
scores[scores$major %in% c("SS", "NS"),]
```

```
##    ID   group pretest posttest trait_anxiety difference major state_anxiety
## 1   1   Treat      60       80             2          20    SS            20
## 2   2   Treat      55       72            18          17    NS            26
## 4   4   Treat      74       88            10          14    SS            12
## 5   5   Treat      69       83            12          14    NS             6
## 7   7   Treat      88       96             2           8    SS             4
## 8   8   Treat      68       86            12          18    NS            10
## 10 10   Treat      80       92             1          12    SS             5
## 11 11 Control      79       85             4           6    NS            20
## 13 13 Control      60       66            18           6    SS            28
## 14 14 Control      85       90             6           5    NS             2
## 16 16 Control      73       80             9           7    SS            15
## 17 17 Control      64       77            11          13    NS            20
## 19 19 Control      77       87             6          10    SS            11
## 20 20 Control      81       93             4          12    NS             6
##    math_score
## 1          40
## 2          35
## 4          70
## 5          92
## 7          90
## 8          88
## 10         92
## 11         46
## 13         32
## 14         90
## 16         75
## 17         60
## 19         77
## 20         84
```

Bonus practice:

1. Develop a question about this dataset that you can answer by indexing.
2. Write your question as a comment above your code.
3. Create an index to answer the question.
4. Write the answer to your question as a comment below your code.