Chapter 4 Working with Data

This chapter focuses on working with real datasets used throughout the course. You will learn how course data are made available, how to inspect and understand variables, and how to perform basic data transformations using the `dplyr` package.

The goal is to help you move from small, isolated examples to working confidently with full datasets.

4.1 Course data vs importing data

In this course, many datasets are provided directly through the **MKT4320BGSU** package. These datasets are accessed using the `data()` function. For the next two chapters, we'll be using the `directmktg` dataset.

```
data("directmktg")
```

Using `data()` has several advantages:

- Everyone is working with the same dataset
- No file paths are required
- Fewer import-related errors

When possible, labs will rely on datasets loaded with `data()`.

4.1.1 Importing external data

In some cases, you may work with your own data files (e.g., CSV files).

```
mydata <- read.csv("mydata.csv")
```

Imported data behave the same way as course datasets once they are loaded into R. The key difference is how they enter your workspace.

As a general rule:

- Use `data()` for course-provided datasets
- Use `read.csv()` (or similar functions) for your own files

4.2 Inspecting datasets and variables

Before analyzing data, it is critical to understand what is in the dataset.

4.2.1 Viewing the data

The `head(data_object)` first few rows (the default is `n=6`) and helps you understand the structure of the data. There is a similar function, `tail()`, that looks at the last `n` rows.

```
head(directmktg)
```

```
    userid age buy gender salary
1 15624510  19  No   Male     19
2 15810944  35  No   Male     20
3 15668575  26  No Female     43
4 15603246  27  No Female     57
5 15804002  19  No   Male     76
6 15728773  27  No   Male     58
```

```
head(directmktg, n=5)
```

```
    userid age buy gender salary
1 15624510  19  No   Male     19
2 15810944  35  No   Male     20
3 15668575  26  No Female     43
4 15603246  27  No Female     57
5 15804002  19  No   Male     76
```

```
tail(directmktg, n=5)
```

```
      userid age buy gender salary
396 15691863  46 Yes Female     41
397 15706071  51 Yes   Male     23
398 15654296  50 Yes Female     20
399 15755018  36  No   Male     33
400 15594041  49 Yes Female     36
```

To see the entire data object, you can use the `View(data_object)` function, which will open the data in a spreadsheet-like format in the Source pane.

```
View(directmktg)
```

4.2.2 Dataset structure

The `str(object)` function is one of the most important commands in R. When used with a dataset object, it shows:

- Variable names
- Variable types (numeric, factor, character)
- A preview of values

```
str(directmktg)
```

```
'data.frame':   400 obs. of  5 variables:
 $ userid: num  15624510 15810944 15668575 15603246 15804002 ...
 $ age   : num  19 35 26 27 19 27 27 32 25 35 ...
 $ buy   : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 2 1 1 ...
 $ gender: Factor w/ 2 levels "Male","Female": 1 1 2 2 1 1 2 2 1 2 ...
 $ salary: num  19 20 43 57 76 58 84 150 33 65 ...
```

4.2.3 Variable summaries

When used on the entire dataset, the `summary(object_name)` function provides a summary of all variables. The output depends on the variable type:

- Numeric variables: min, max, mean, quartiles
- Factor variables: counts by level

```r
summary(directmktg)
```

```
     userid              age         buy          gender        salary
 Min.   :15566689   Min.   :18.00   No :257   Male  :196   Min.   : 15.00
 1st Qu.:15626764   1st Qu.:29.75   Yes:143   Female:204   1st Qu.: 43.00
 Median :15694342   Median :37.00                          Median : 70.00
 Mean   :15691540   Mean   :37.66                          Mean   : 69.74
 3rd Qu.:15750363   3rd Qu.:46.00                          3rd Qu.: 88.00
 Max.   :15815236   Max.   :60.00                          Max.   :150.00
```

4.3 Data frames, variables, and indexing

A **data frame** is a table of rows and columns. In R, you will often need to reference a specific variable (column) or select a subset of rows/columns.

4.3.1 Accessing a variable (a column)

Use the `$` operator to access a column by name.

```
directmktg$age
```

```
  [1] 19 35 26 27 19 27 27 32 25 35 26 26 20 32 18 29 47 45 46 48 45 47 48 45 46 47 49 47 2
 [39] 26 27 27 33 35 30 28 23 25 27 30 31 24 18 29 35 27 24 23 28 22 32 27 25 23 32 59 24 2
 [77] 18 22 28 26 30 39 20 35 30 31 24 28 26 35 22 30 26 29 29 35 35 28 35 28 27 28 32 33 1
[115] 42 40 35 36 40 41 36 37 40 35 41 39 42 26 30 26 31 33 30 21 28 23 20 30 28 19 19 18 3
[153] 31 36 40 31 46 29 26 32 32 25 37 35 33 18 22 35 29 29 21 34 26 34 34 23 35 25 24 31 2
[191] 24 19 29 19 28 34 30 20 26 35 35 49 39 41 58 47 55 52 40 46 48 52 59 35 47 60 49 40 4
[229] 40 42 35 39 40 49 38 46 40 37 46 53 42 38 50 56 41 51 35 57 41 35 44 37 48 37 50 52 4
[267] 40 37 47 40 43 59 60 39 57 57 38 49 52 50 59 35 37 52 48 37 37 48 41 37 39 49 55 37 3
[305] 40 42 51 47 36 38 42 39 38 49 39 39 54 35 45 36 52 53 41 48 48 41 41 42 36 47 38 48 4
[343] 38 47 47 41 53 54 39 38 38 37 42 37 36 60 54 41 40 42 43 53 47 42 42 59 58 46 38 54 6
[381] 42 48 44 49 57 56 49 39 47 48 48 47 45 60 39 46 51 50 36 49
```

A column is often a vector, which means you can use vector functions on it.

```
mean(directmktg$age)
```

```
[1] 37.655
```

```
summary(directmktg$age)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  18.00   29.75   37.00   37.66   46.00   60.00 
```

4.3.2 Indexing a data frame

Data frames can be indexed (or subsetted) using:

- `data[rows, cols]`
- leave rows blank to keep all rows: `data[, cols]`
- leave cols blank to keep all columns: `data[rows, ]`

In addition, sequencing can be used to get multiple rows and/or columns. By default, subsetting a single column returns a vector. To retain it as a data frame, use the `drop = FALSE` argument.

```
directmktg[1, ]              # first row, all columns
directmktg[1:5, ]            # first 5 rows, all columns
directmktg[, 2]              # all rows, column 2 only (age), as a vector
directmktg[, 2, drop=FALSE]  # all rows, column 2 only (age), as a data frame
```

You can also access index columns using the column name columns. in two other ways. First, if you know the column number, you can using `[[ ]]` (useful when column names are stored in an object).

```
directmktg[1, "age"]                # first row, age column
directmktg[1:5, c("age", "gender")]  # first 5 rows, age and gender columns
```

4.3.3 Filtering rows using a condition (base R)

You can filter rows by using a logical condition inside the row index.

```
directmktg[directmktg$age >= 55, ]
```

```
      userid age buy gender salary
65  15605000  59  No Female     83
205 15660866  58 Yes Female    101
207 15654230  55 Yes Female    130
213 15707596  59  No Female     42
216 15779529  60 Yes Female    108
220 15732987  59 Yes   Male    143
224 15593715  60 Yes   Male    102
228 15685346  56 Yes   Male    133
244 15769596  56 Yes Female    104
248 15775590  57 Yes Female    122
259 15569641  58 Yes Female     95
263 15672821  55 Yes Female    125
272 15688172  59 Yes Female     76
273 15791373  60 Yes   Male     42
275 15692819  57 Yes Female     26
276 15727467  57 Yes   Male     74
281 15609669  59 Yes Female     88
293 15625395  55 Yes   Male     39
301 15736397  58 Yes Female     38
335 15814553  57 Yes   Male     60
337 15664907  58 Yes   Male    144
356 15606472  60 Yes   Male     34
366 15807525  59 Yes Female     29
367 15574372  58 Yes Female     47
371 15611430  60 Yes Female     46
372 15774744  60 Yes   Male     83
374 15708791  59 Yes   Male    130
380 15749381  58 Yes Female     23
385 15806901  57 Yes Female     33
386 15775335  56 Yes   Male     60
394 15635893  60 Yes   Male     42
```

This keeps only rows where `age >= 55`.

4.3.4 Specific variable summaries (base R)

You can use variable naming or indexing to get specific variable summaries using the `summary()` function.

```
summary(directmktg$age)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  18.00   29.75   37.00   37.66   46.00   60.00 
```

```
summary(directmktg[,2:3])
```

```
      age          buy       
 Min.   :18.00   No :257  
 1st Qu.:29.75   Yes:143  
 Median :37.00            
 Mean   :37.66            
 3rd Qu.:46.00            
 Max.   :60.00            
```

4.4 Data transformations (base R)

Data transformation refers to modifying or creating variables so they are more useful for analysis. This includes:

- creating new variables
- recoding variables

Base R can do all of these. Even if you prefer `dplyr` (covered later), it is helpful to understand what is happening “under the hood.”

4.4.1 Creating a new variable

This example creates a **new variable** in the data frame using base R. The expression on the right-hand side, `directmktg$age / 10`, is evaluated first. Because `directmktg$age` is a vector, the division is applied to **each observation**, and the result is a new vector of the same length.

Using the assignment operator `<-`, this new vector is stored as a column named `age10` in the `directmktg` data frame.

```
directmktg$age10 <- directmktg$age / 10
head(directmktg, n=5)
```

```
    userid age buy gender salary age10
1 15624510  19  No   Male     19   1.9
2 15810944  35  No   Male     20   3.5
3 15668575  26  No Female     43   2.6
4 15603246  27  No Female     57   2.7
5 15804002  19  No   Male     76   1.9
```

4.4.2 Recoding with `ifelse()`

A common transformation is converting a categorical outcome into a numeric indicator.

```
directmktg$buy_binary <- ifelse(directmktg$buy == "Yes", 1, 0)
table(directmktg$buy, directmktg$buy_binary)
```

```
       0   1
  No  257   0
  Yes   0 143
```

Another common recoding is converting a numeric value into categories, which can be done with nested `ifelse()` statements.

```r
directmktg$salary_cat <- ifelse(directmktg$salary <=50, "Low",
                                ifelse(directmktg$salary <=80, "Med", "High"))
head(directmktg[,c("salary","salary_cat")], 10)
```

```
   salary salary_cat
1      19        Low
2      20        Low
3      43        Low
4      57        Med
5      76        Med
6      58        Med
7      84       High
8     150       High
9      33        Low
10     65        Med
```

4.5 What's next

In the next chapter, we introduce the `dplyr` **package** for data manipulation. While the tasks performed with `dplyr` are similar to those you have already seen using base R, `dplyr` provides a more consistent and expressive way to work with data. These tools will be used extensively in later chapters for descriptive analysis, visualization, and modeling.