# Data Types and Data Structures

Classification is an unexciting but necessary component of any field of learning. When learning a foreign language you typically have a prior grasp of the difference between, say, nouns and verbs, an understanding without which would make learning considerably more difficult. Programming languages are no different in this respect. You'll come to see the benefit of learning some fundamental ideas from computer science and how they are used in R. These benefits result from the fact that many coding mistakes arise from asking R to do things with certain types of data it doesn't (or shouldn't) know how to do. And in the event R doesn't know what to do when you run your code, you will know from the red R-speak returned in the Console. If you have a basic handle on these computer science terms, then you'll have a loose understanding of R-speak and will thus be able to troubleshoot the problem more effectively. Here we'll consider the most fundamental properties of a dataset: Its data structure and the data type of its contents.

## Data Structures

R organizes data in different ways, which can be a source of confusion. After all, Excel deals in spreadsheets, so why doesn't R organize all types of data in a spreadsheet format? The reason why R organizes data in different formats is a key reason programming languages are deemed preferable to their point-and-click/point-and-type alternatives: You gain more control over your workflow.

These different ways to organize data are called **data structures** and we'll only need to know the difference between two of them: vectors and data frames.

A **vector** is one-dimensional sequence of a particular data type. (See next section for the definition of data type.) To create a vector, use the combine function `c()` with each entry separated by a comma. Consider these vectors I'm naming `a`, `b`, and `c`.

```r
# Create 3 vectors of different data types
a <- c(0.77, 1, 98, -0.34, 5)
b <- c("How are you?", "Good.", "Thank you", "for", "asking.")
c <- c(TRUE, FALSE, FALSE, TRUE, TRUE)
```

Observe that after running these commands nothing returned in the Console. The 3 vectors were stored under the Values heading in the Environment pane (upper-right pane in RStudio). They were stored because we assigned names to them.

Extracting elements from a vector using square-bracket and indexing notation as follows `vector_name[index]`, where `vector_name` is the vector's name and `index` is an integer vector. In the following command, R extracts the second and third elements using the integer vector `c(2, 3)` from the vector `b` created above.

```r
# Extract the 2nd and 3rd elements from vector b
b[c(2, 3)]
```

R will then return `"Good."` and `"Thank you"` in the Console.

A **data frame** is what you think of when you think of a dataset. It's R's version of a spreadsheet and will be our preferred way to store data. An important aspect of data frames is that each of its columns has the same number of entries (observations). Also, just like a vector, the data in each column of a data frame must be of the same data type. But a data frame can, and most often will, consist of columns of different data types. To create a data frame from a set of vectors, use the `data.frame()` function with the vectors to be included listed as arguments of the function. Let's create a data frame named `df` with the 3 vectors created above:

```r
# Create data frame called df with vectors a, b, and c
df <- data.frame(a, b, c)
```

After creating `df` nothing returned in the Console. This is again because a name was assigned to the data frame and, as a result, `df` was added to the Environment pane. But notice that this time `df` is listed under the Data heading (not the Values heading). To view `df` you can either click on the spread sheet icon on the far right or else run the command `View(df)`. Also note that under the Data heading it indicates that `df` has `5 obs. of 3 variables`. The `obs.` is short for observations and refers to the number of rows in the data frame, while variables refers to its number of columns.

If you need to determine whether you're dealing with a data frame or a vector, the quickest way is to look at the Environment pane. As mentioned above, if your dataset is under the Data heading, then it's a data frame. If it's under the Values heading, then it's a vector.

### Identifying Columns in a Data Frame

Pretty much every time you use R, you'll need to work with particular columns from a data frame. To work with a particular column, first specify the data frame and then the column's name, with the data frame and column separated by `$`. For example, to extract column `a` from data frame `df` run the command:

```r
# Extract column a from data frame df
df$a
```

The Console then returns the contents of `a`. Say instead that we wanted to know the mean of the numbers in column `a`. Then we'd first extract `a` from `df` as before, and then wrap the `mean()` function around it:

```r
# Take the mean of column a
mean(df$a)
```

The mean of column `a`, as returned in the Console, is `20.886`.

Notice what's happening: We want to find the mean of the `a` column, but we need to first specify which data frame (here `df`) the `a` column belongs to. So when you work with a specific column from a data frame:

1. Specify the data frame that column belongs to.

2. Instruct R to extract a column by using the `$` symbol.

3. Specify the column you want from the data frame.

This, then, is how R interprets `df$a`: Extract the column `a` from the data frame `df`.

## Data Types

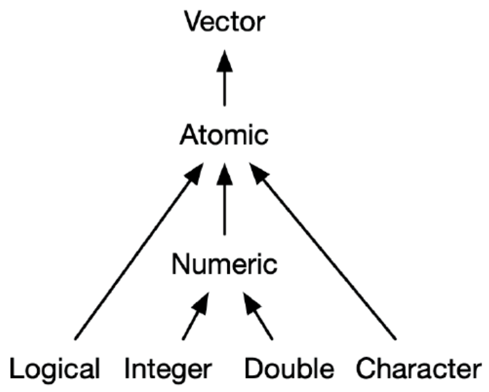
Popular usage of the word data suggests data are just a bunch of (real) numbers. But data come in many different forms. In this course we'll need to work with dates of the year, letters of the alphabet, alphanumeric strings and numbers.

A **data type** is a broad classification of the kinds of data R works with. The 3 most commonly used data types commonly are:

1. Numeric: 77 or 37.39178
2. Character: "a7" or "Heads"
3. Logical: TRUE or FALSE

The fundamental object in R is called an *atomic vector*, but we'll refer to them simply as vectors. The building blocks of data frames are vectors, and, because of this, the terms "vector," "variable" and "column" are used synonymously. The elements of a vector have the same data type and so we speak of numeric vectors or character vectors or logical vectors. In this sense a vector is homogeneous, contrasting with the more complex *list*, which can contain different data types. We'll have more to say about lists later on. Here the focus is on vectors.

Numeric vectors are further subdivided into double vectors (real numbers) and integer vectors. Though the distinction between real numbers and integers can be critical in certain circumstances, we'll only deal with double vectors. For us, then, a numeric vector means a double vector.[1]



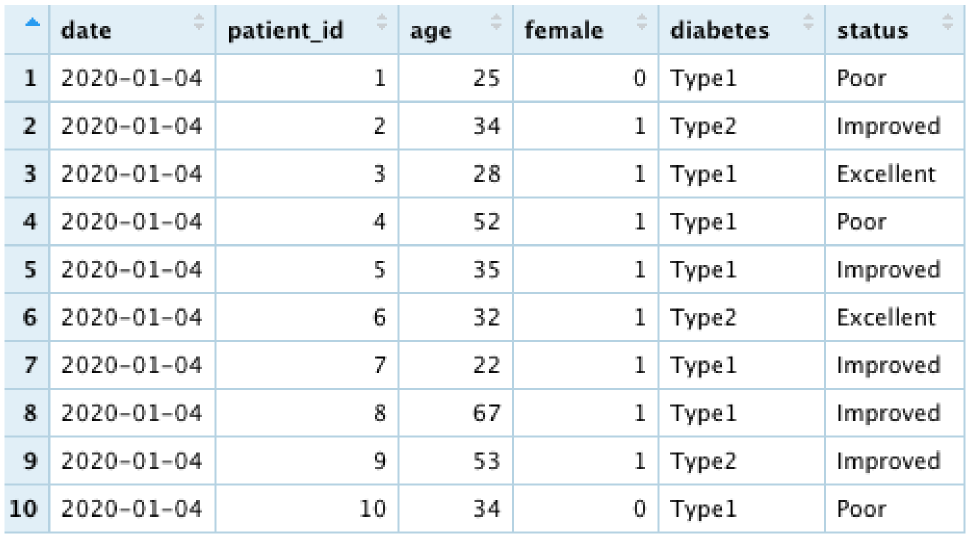
These 3 data types are broad categories. In actual applications though, you'll need a deeper understanding of the data you're working with. Determining whether a variable is continuous or nominal or ordinal guides choices over how data are analyzed and visually presented. (See Appendix on scales of measurement for definitions of continuous, nominal and ordinal variables.)

But let's get concrete. Look at the data below of 10 (fictitious) patients. The `age` variable is a continuous variable; for example, patient 2 is 9 years older than patient 1. The gender variable, `female`, consists of numbers, but that doesn't mean it's a continuous variable. Is each female (the 1's) in the dataset 1 better than every male (the 0's)? No, because gender is a nominal variable. The values a gender variable can take on are *names*: "male" and "female". If, instead of using 0's and 1's to designate males and females, we used the words "male" and "female" nothing fundamentally alters the information embedded in the gender variable. In fact, isn't that what's going on with the `diabetes` variable? We could create a new variable called `type1` with the follow assignment:

$$type1_i = \begin{cases} 0, & \text{Patient } i \text{ has Type 2 diabetes} \\ 1, & \text{Patient } i \text{ has Type 1 diabetes,} \end{cases}$$

and the information, although in different form, is duplicated.

[1]In case you're wondering why the word "double" designates a vector of real numbers, it's short for "double-precision floating-point numbers." To simplify matters greatly, because real numbers have infinite decimal expansions, they must have approximate representations to be used in computing. In R, real numbers are represented by a double-precision floating-point format.



| | date | patient_id | age | female | diabetes | status |
|---|---|---|---|---|---|---|
| 1 | 2020-01-04 | 1 | 25 | 0 | Type1 | Poor |
| 2 | 2020-01-04 | 2 | 34 | 1 | Type2 | Improved |
| 3 | 2020-01-04 | 3 | 28 | 1 | Type1 | Excellent |
| 4 | 2020-01-04 | 4 | 52 | 1 | Type1 | Poor |
| 5 | 2020-01-04 | 5 | 35 | 1 | Type1 | Improved |
| 6 | 2020-01-04 | 6 | 32 | 1 | Type2 | Excellent |
| 7 | 2020-01-04 | 7 | 22 | 1 | Type1 | Improved |
| 8 | 2020-01-04 | 8 | 67 | 1 | Type1 | Improved |
| 9 | 2020-01-04 | 9 | 53 | 1 | Type2 | Improved |
| 10 | 2020-01-04 | 10 | 34 | 0 | Type1 | Poor |

What about the `status` variable? Like `diabetes` it's a string of letters. However, unlike diabetes the *order* of each patient's status matters. That is, although we can't say how much better off patient 2 is than patient 1, we can say that patient 2 is doing better than patient 1. The `diabetes` variable is an example of an ordinal variable in that the values it can assume imply numerical differences but we don't how the magnitude of these differences. We only know one value is more or less than another.

### Factors

Categorical variables (i.e., nominal and ordinal variables) are called **factors**. The function `factor()` stores the categorical values as a vector of integers $(1, 2, \ldots, k)$, where $k$ is the number of unique values. For example, consider this vector of nominal data:

```r
# Create a vector of nominal data called diabetes
diabetes <- c("Type1", "Type2", "Type1", "Type1")
```

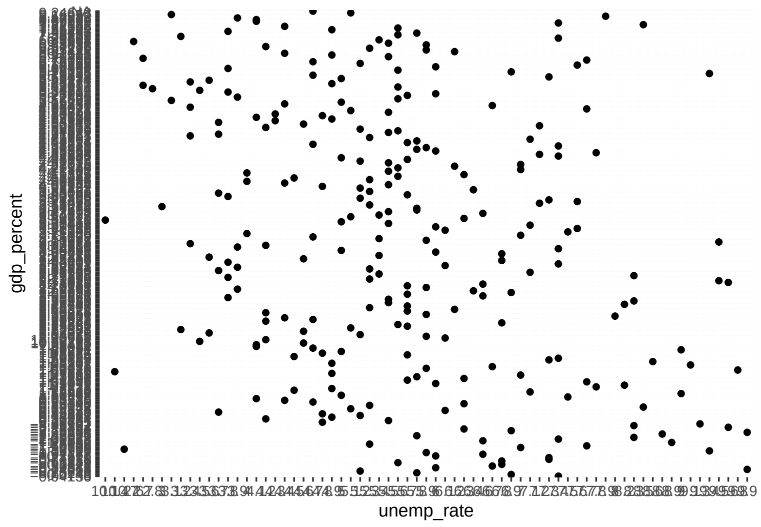
Then by running `diabetes <- factor(diabetes)` a vector $(1, 2, 1, 1)$ is stored in R.

## Know Your Data

This is all to say: *Know your data.* You're susceptible of committing massive blunders — and often without knowing it — if you don't have a good sense of the nature of your data.

To illustrate the point, think of this example. I download some macroeconomic data on GDP growth rates and unemployment rates from FRED (the Federal Reserve's repository of economic data). After downloading the dataset I do some reformatting in Excel, making whatever edits I deem necessary. I then import it into R, call the data frame `gdp_unemp_df`, and generate a scatterplot of GDP growth rates and unemployment rates. Here it is:

```r
# Create scatterplot
ggplot(data = gdp_unemp_df) +
  geom_point(mapping = aes(x = unemp_rate, y = gdp_percent))
```



To R's credit it did generate the plot. But something's clearly gone wrong. Apart from the crazy labels on the axes (called "tick labels"), note that the plot's theme changed from `ggplot()`'s default gray background to white.[2] As a first step to fix the plot, check the data types. My preferred way to do this quickly is to use the `glimpse()` function from the **tidyverse** package. The `glimpse()` function takes a data frame as input and returns it transposed with the data type of each variable as the first column. Let's see the data types for this data frame:

```
glimpse(gdp_unemp_df)
```

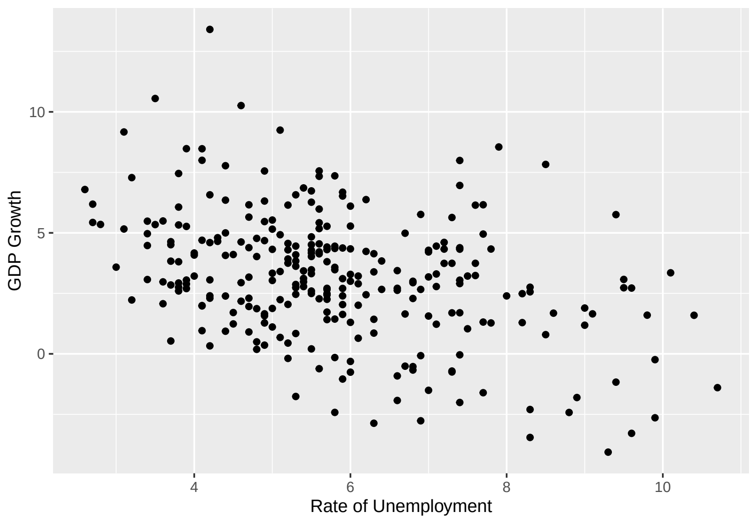
Upon importation into R, all 3 variables were interpreted as character vectors.[3] How do we know that? Note the `<chr>` after each variable's name. A quick fix[4] to display the data correctly is to (temporarily) convert the data types of `unemp_rate` and `gdp_percent` to double (i.e., real numbers). Use the `as.double()` function as follows:

```
# Create scatterplot with variables as doubles and new axis labels
ggplot(data = gdp_unemp_df) +
  geom_point(mapping = aes(x = as.double(unemp_rate),
                           y = as.double(gdp_percent))) +
  labs(x = "Rate of Unemployment", y = "GDP Growth")
```

[2]Also `ggplot()` automatically adjusts the size of plots to include all data; the half points are then another clue something's amiss.

[3]This often occurs when you edit data in Excel and then imported the edited file into R. It also results when a spreadsheet is saved as an Excel file (.xls or .xlsx) and not as a .csv file. Your default download format should be a .csv file if possible.

[4]Once you gain familiarity with **dplyr** functions you'll know how to permanently convert variables to different data types, which is what is needed here.



This is much better, and it makes economic-sense: The faster the economy is growing, the lower the unemployment rate. Also notice that on the last line of code I changed the labels displayed on the axes so that they're more descriptive. If I didn't change the labels, R puts what's specified inside `aes()` for the x and y variable. So it would've returned, for example, `as.double(unemp_rate)` on the x-axis. Not a very indicative description of what's being measured.

## Appendix: Scales of Measurement

All variables you'll encounter will be either continuous or categorical variables. A **continuous** variables take the form of numbers that you can perform arithmetic operations on and they'll retain their meaning. For example, if some person is 5 feet 8 inches tall and another is 5 feet 10 inches, we not only that one person has greater height than the other, we know by how much one is taller than the other by subtracting.

Categorical variables come in 2 forms:

1. Nominal variables
2. Ordinal variables.

As its name suggests, **nominal** data consist of names of things. Eye color is a nominal variable and assumes values like "Brown", "Blue", "Hazel" and so on. It's often helpful to assign numbers to these categories, but you must remember that these numbers are arbitrary assignments. If all people in some dataset with brown eyes are assigned values of 0 and all people with blue eyes the value of 1, then it's meaningless to claim that people with brown eyes are 1 less than blue-eyed people.

The second type of categorical variable is an **ordinal** variable. Ordinal measurements are ranked (i.e., ordered) from low to high, but we don't know how much higher or lower one individual is in relation to another. We just know the ranking. Ordinal data may appear as either numbers or strings of letters. But regardless of their specific form, they need to be treated accordingly and not as either nominal or continuous variables.