

## 2

# Data for Plotting

I am a firm believer in the idea that the first step before embarking into producing a plot is knowing the characteristics of the data. This may seem pretty obvious for the experienced analyst, but it may not be so for the novice, especially if this new apprentice is using default plots and lacks some formal training on basic methodological, statistical, and visual concepts. It is also important to keep in mind that some plotting functions require particular data structures as input, which are not necessarily the ones you have in hand.

In the next sections, I will provide different ways of thinking about the data in your hands, which will allow you to identify their characteristics and format, so that you can profit from these in the coming chapters.

### 2.1 Data-Types

When you have information about the society you are analyzing, you will definitely run into very different data types. In general, if we have a traditional spreadsheet in mind where rows are the individuals or units of study and the columns are their variables, the data type is a characteristic of the column. Considering the types managed by **R** or **Python**, let me summarize those into these groups:

- **Text.** A column where its set of values is a candidate for a *key* field: values that are not repeated because they are uniquely identifying the unit of analysis (i.e. “full name”). If it is not unique, the text or *string* or *character* values are generally turned into a categorical column. However, if the text is, for example, a “tweet” it will remain as text to extract its meaning (it will not be a categorical, and it will not be a key).
- **Categorical.** These data are produced by organizing or categorizing a variable into statuses or levels. These statuses have names to differentiate

one from the other, and they can represent an ordering or not. They are limited and discrete; that is, each represents one particular characteristic. Notice that categories can be represented with numbers<sup>1</sup>.

- **Numerical.** You have numerical data when your data values are potentially unlimited. These values appear when a variable can be measured or counted. Values from counting are discrete (not allowing decimals). Measurement allows for decimal values.
- **Boolean.** Values that are used in logical operations, as they only accept two values: True and False.
- **Date.** They are simple numbers behind the scenes, but when a column has a date format it allows for particular operations that can give, for example, a result in terms of days or months.
- **Missing.** We generally consider a missing value as the cell that has no information, and most programs will recognize them like that. However, when doing surveys, there are people that do not want to give answers to a particular question (which will then produce a missing value) but you still collect the reason for that (i.e. “do not know”, “not interested”). You need to be aware of this, because those answers may be present in categorical and numerical values<sup>2</sup>.

**R** or **Python** will “decide” the data types by default, but this does not mean you have to rely on that default judgment. Most of the time the data types need to be formatted to the right type by the user. For example, when visiting the Wikipage of Freedom Indices<sup>3</sup> you see text in every column, but that text is just a label to identify the levels of a scale (Stevens, 1946). Those texts represent values that may need further mathematical treatment. You also see the text “n/a” being used to represent a missing value. Part of this data table is shown in Figure 2.1.

You will need to turn the text into the right data type. Notice that each column should represent a single data type, but the exception is that a missing value can always be present in any of them.

**R** and **Python** can manage the data types presented in this book very well. **R** will directly implement any of these, while **Python** will need libraries<sup>4</sup> beyond its basic functionality to represent those values. Let me give you further details and examples.

<sup>1</sup> You can find data where 1 represents “single”; 2 represents married, 3 represents divorced, and so on.

<sup>2</sup> Imagine that you have the value -9 in a variable like *age*, that may not be a typing error, but a code telling you “I refuse to give my age.” To avoid such errors, you must always read the methodological section of all data you collect.

<sup>3</sup> [https://en.wikipedia.org/wiki/List\\_of\\_freedom\\_indices](https://en.wikipedia.org/wiki/List_of_freedom_indices).

<sup>4</sup> I will mainly use *Pandas* as the default (see McKinney, 2010).

## List by country [ edit ]






Country	Freedom in the World 2019 <sup>[10]</sup>	2019 Index of Economic Freedom <sup>[11]</sup>	2019 Press Freedom Index <sup>[3]</sup>	2018 Democracy Index <sup>[13]</sup>
 Abkhazia	partly free	n/a	n/a	n/a
 Afghanistan	not free	mostly unfree	difficult situation	authoritarian regime
 Albania	partly free	moderately free	noticeable problems	hybrid regime
 Algeria	not free	repressed	difficult situation	authoritarian regime
 Andorra	free	n/a	satisfactory situation	n/a

Figure 2.1 Example of data frame with text values

Image captured from List of freedom Indices, in Wikipedia, n.d., Retrieved June 1, 2019.

## 2.2 Data Types in R and Python

### 2.2.1 Text Values

In Figure 2.1, you see five columns, where each should have a particular data type. The first column is not qualifying in any way the country, it is just naming or *identifying* the row. An identifier is unique (unless it is a composite identifier<sup>5</sup>), and you should set its data type as text. You are not planning to do any mathematical operation with identifiers. **R** will use the type *character* (or *chr*) for this kind of data; while **Python** will use the type *string* (or *str*) (or *object* if it is a column in a data frame as defined by *Pandas* in **Python**).

### 2.2.2 Categorical Values

You have categorical variables when the column represents a characteristic of the unit of analysis. In this situation, you are aware that one characteristic or attribute can be shared among many rows in your data.

<sup>5</sup> A city name can be an identifier, but sometimes you need city and county names, making a composite identifier when city names are not unique in a county. This is solved if the row has a unique code.

Index	Scale				
Freedom in the World <sup>[10]</sup>	free		partly free		not free
Index of Economic Freedom <sup>[11]</sup>	free	mostly free	moderately free	mostly unfree	repressed
Press Freedom Index <sup>[3]</sup>	good situation	satisfactory situation	noticeable problems	difficult situation	very serious situation
Democracy Index <sup>[13]</sup>	full democracy		flawed democracy	hybrid regime	authoritarian regime

Figure 2.2 Levels of ordinal variables

Image captured from List of freedom Indices, In Wikipedia, n.d., Retrieved June 1, 2019.

Categorical data can be of two sub-types:

- **Nominal.** These are attributes that do not express any kind of order other than alphabetical. They will not tell you which is the “best” or the “worst”. When nominal values can only take two values, they are known as dichotomous variables. The data from Figure 2.1 has no nominal data. **R** will use the type *factor* for this kind of value; while **Python** will use the type *category* if it is a column in a data frame as defined by *Pandas* in **Python**.
- **Ordinal.** These are attributes that do express, as the name implies, some kind of order. They indicate a certain degree of achievement or sequence. However, as these are not numbers, you cannot compute a numerical distance among the levels. In general, the amount of levels is limited to a few values. The last four columns in Figure 2.1 represent ordinal data, whose levels are labelled with some text. **R** will use the type *ordered* for this kind of data; while **Python** will use the type *Ordered category* if it is a column in a data frame as defined by *Pandas* in **Python**.

Figure 2.2 shows the levels of each column. In this situation you see that each level has a text label, but these columns are not text type. You can also use integer numbers to represent levels, but keeping in mind they are not actually numbers. In other words, if you use the numbers from one to five to represent the levels of the *Press Freedom Index*, you cannot say that level four is two times better than level two, as that statement implies numerical distance. You can only say that four is two levels above two.

### 2.2.3 Numerical Values

Numbers are simpler to explain. These are values that represent order, as with the ordinals, but that also represent distance. We can divide numbers into two groups:

- Counts. As the name implies, these values arise from counting. Population is an example of count data, for instance. **R** will use the type *integer* for this kind of data; and **Python** will also use the type *integer* if it is a column in a data frame as defined by *Pandas* in **Python**.
- Measurements. When you measure something, you can expect decimal values. However, a measurement process has particular characteristics which can divide numbers into two different groups or scales:
  - Interval scale. These are values where the distance can not be expressed as a proportion, only as distance; that is, 40° Fahrenheit is not two times warmer than 20°, but it is in fact 20 units apart in that scale. Of course, you can even express that distance using decimals. Scales where zero does not represent the absence of the attribute are designated as interval values. In that situation, zero in Fahrenheit does not mean an absence of temperature, or zero in the Intelligence Quotient (IQ) does not mean the absence of intelligence. As the zero is not absolute, you will find that, as in temperature and IQ, there are alternative ways to measure that characteristic; and in each situation, the zero may have a different meaning. **R** will simply use the type *numeric* or *double* for this kind of data; while **Python** will use the type *float* if it is a column in a data frame as defined by *Pandas* in **Python**.
  - Ratio scale. You have Ratio values when the distance can also be interpreted as a proportion. A person with a height of 1.80 meters, is 1.5 times taller than a person who measures 1.20 meters, while also 60 centimeters taller. You can have alternative units to measure height, but zero has the same meaning in all of them: zero US Dollars is zero Euros, and zero Yens; and zero meters is zero feet. **R** uses the type *numeric* or *double*; while **Python** uses *float*.

### 2.2.4 Boolean Values

You have Boolean data types when values can be either true or false. You can use dichotomous values or even a text value for that, but programming languages support this specific data type. **R** will simply use the type *logical*

for this kind of data; while **Python** will use the type *boolean* if it is a column in a data frame as defined by *Pandas* in **Python**.

### 2.2.5 Date Values

Date values are a particular representation of time. The computer is actually counting time, but it has a means to convert that into a date, which may be used in some particular applications that involve, for example, time differences. Both **R** and **Python** have a *date* data type.

### 2.2.6 Missing Values

Missing values can always be present in any column, but this is not informed as a separate data type: if a column is a number or a category, the presence of a missing value will not change the data type. Missing values need to respect how a language represents them; if you use an “x” to denote a missing value in a numeric column, that character will turn or *coerce* the column into a text data type. **R** uses *NA* to signal a missing value, and **Python** uses *None*.<sup>6</sup>

Here, I want to show you how **R** and **Python** can open a file and recognize a default data type. Let me use some data I collected in 2017 from the Common Core of Data from the US Department of Education.<sup>7</sup> I kept detailed information on public schools from the state of Washington. Let me try **R** first:

```
> #link to data
> linkRepo='https://github.com/resourcesbookvisual/data/'
> linkEDU='raw/master/eduwa.csv'
> fullLink=paste0(linkRepo,linkEDU)
> #
> #getting the data:
> #avoiding that text values are read as categorical
> eduwa=read.csv(fullLink,stringsAsFactors = FALSE)
> #
> #what you have
> str(eduwa,width = 65,strict.width='cut')
```

```
'data.frame': 2427 obs. of 24 variables:
 $ NCES.School.ID : num 5.30e+11 5.30e+11 5.31e+11 5.30e..
 $ State.School.ID : chr "WA-31025-1656" "WA-06114-1646"..
 $ NCES.District.ID : int 5304860 5302700 5309100 5300030 ..
 $ State.District.ID : chr "WA-31025" "WA-06114" "WA-34033"..
 $ Low.Grade : chr "6" "KG" "9" "PK" ...
 $ High.Grade : chr "8" "12" "12" "6" ...
 $ School.Name : chr "10th Street School" "49th Stre"..
 $ District : chr "Marysville School District" "E"..
```

<sup>6</sup> Notice that both **R** and **Python** also use *Inf/-Inf* to represent infinite values, and *NaN* to say a value is not a number. Pay attention to those, as you may need to decide whether they are to be considered missing when a column is the result of a computation.

<sup>7</sup> <https://nces.ed.gov/ccd/>.

```

$ County      : chr "Snohomish" "Clark" "Thurston" "...
$ Street.Address : chr "7204 27th Ave NE" "14619B NE 4"...
$ City        : chr "Marysville" "Vancouver" "Tumwa"...
$ State       : chr "WA" "WA" "WA" "WA" ...
$ ZIP         : int 98271 98682 98512 98520 99205 98...
$ ZIP.4.digit  : int NA 6308 NA 5510 NA NA NA NA 9...
$ Phone       : chr "(360)965-0400" "(360)604-6700"...
$ Locale.Code  : int 22 12 13 33 12 13 21 12 41 41 ...
$ LocaleType   : chr "Suburb" "City" "City" "Town" ...
$ LocaleSub    : chr "Suburb: Midsize" "City: Midsiz"...
$ Charter      : chr "No" "No" "No" "No" ...
$ Title.I.School : chr "Yes" "No" "No" "Yes" ...
$ Title.I.School.Wide : chr "Yes" NA NA "Yes" ...
$ Student.Teacher.Ratio: num 23.4 8.4 21.5 15.9 6.5 15.3 NA 1...
$ Free.Lunch    : int 28 53 169 292 12 411 48 102 101 ..
$ Reduced.Lunch  : int 3 9 40 10 4 23 12 22 23 0 ...

```

The data were stored in the GitHub repo of the book. I split the link to the data into two texts, 'linkRepo' and 'linkFile', and then I concatenated both into 'fullLink' (paste0 will not put any character between the texts being concatenated). This step is not needed but I like to do it so that it does not use the whole page width (you will see that I follow this strategy throughout the book). Notice that `read.csv` can read the data from the GitHub link directly (notice I stopped the conversion of text into categorical data, the default behavior when reading in the data). Finally, you see I use the function `str` to get the data type information. I have included some parameters so this function does not populate the width of the page (in most cases, the default output can be very messy).

I will translate the previous **R** code into **Python**. The coding strategy will be the same, but there will be differences: **Python** can use a `+` to concatenate text, and the *Pandas* function `read_csv` will also call the link, but will not try to convert text into categories by default. Notice that **R** does not need to activate a particular package to deal with 'data frames'; this is a native structure for **R**. On the other hand, **Python** needs you to activate *Pandas*. In both cases 'eduwa' is a data frame, but **R** will use a function (`str`) to recover the data type, while **Python** will call the data types an attribute (`dtypes`) of the data frame.

```

# link to data
linkRepo = 'https://github.com/resourcesbookvisual/data/'
linkEDU = "raw/master/eduwa.csv"
fullLink= linkRepo + linkEDU

# activating Pandas and getting the data:
import pandas as pd
eduwa = pd.read_csv(fullLink)

#what you have
eduwa.dtypes

```

In both cases, you get the default data types, which may or may not correspond to what they are supposed to be.

```

NCES.School.ID          int64
State.School.ID         object
NCES.District.ID       int64
State.District.ID      object
Low.Grade              object
High.Grade             object
School.Name            object
District               object
County                 object
Street.Address         object
City                   object
State                  object
ZIP                    int64
ZIP.4-digit            float64
Phone                  object
Locale.Code            float64
LocaleType             object
LocaleSub              object
Charter                object
Title.I.School         object
Title.1.School.Wide    object
Student.Teacher.Ratio  float64
Free.Lunch             float64
Reduced.Lunch          float64
dtype: object

```

From what you learned in the previous subsections, you realize that both **R** and **Python** could differentiate textual from numeric values. However, numeric values are not of the same sub-type in some cases; this is not a problem at all, as integers and real numbers will go through the same processes. You can also see that the ones recognized as text are candidates to be *identifiers* or *key* columns. In this case, the first four are identifiers, as well as the 7th, 10th, and 15th columns ('school name', 'address', and 'phone', respectively). Those variables are not to be analyzed statistically, but may be used for annotating (7th and 15th column) or for geocoding<sup>8</sup> (10th column). Notice that for these data, 'State' is not an identifier but is not a variable either; it is a constant and will not be analyzed statistically. A small issue would be the manual identification of categorical, ordinal or non-ordinal, values that are in fact variables and not identifiers.

## 2.3 Dataset Structure

It is important to be aware that there are different formats in which the data values are stored. The example data in Figure 2.1 are in a rectangular shape, but

<sup>8</sup> Use **R** or **Python** to recover a geographical coordinate.



that may not always be the case. Data formats do not affect the data type, but do affect the way you plan your visuals. And, more important, some functions are good for some formats but fail if the format is not the right one.

### 2.3.1 Tables

Tables are by far the most common format you will encounter. This will be very familiar to most of you as it is the basic structure a spreadsheet represents. In this situation, the values are in the intersection of a row and a column, and the column represents a particular data type. Rows may or not be the unit of analysis, as it depends whether the table has a *wide* or *long* format. All table types are well managed in **R** and **Python**.

#### Wide Format

Wide formats are the most familiar table structures. In this situation, one row gives us all the information of the unit of analysis.

#### Long Format

Long formats<sup>9</sup> are less common than wide formats, but are still very familiar for analysts used to panel data. In this format, there will be a row for every value measured or observed, then the unit of analysis may be distributed in more than one row; that is, it will be a sub-table. This may produce a more concise table.

Data in wide format can be turned into long format, and vice versa. Figure 2.3 shows you the analogy.

Notice that in Figure 2.3, the missing value that was present in the wide format, need not be present in the long one. As mentioned, in the long format each age has its own row, and most units of analysis are a sub table (except for the last case).

### 2.3.2 Non-Table Formats

Most of what analysts do makes use of tables; but non-table formats exist and are also becoming very important. In the next subsections, I will show these data structures.

<sup>9</sup> Wickham (2014a) prefers to call this data “stacked”, which will represent a *tidy* format.

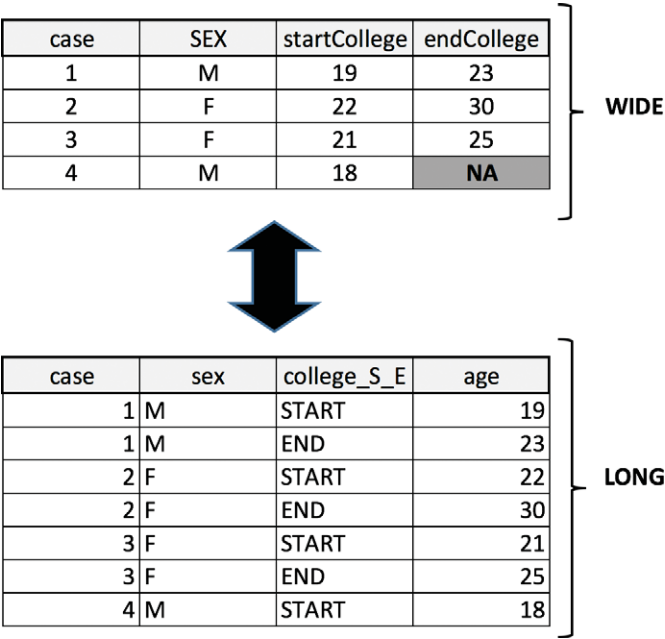


Figure 2.3 Wide and long formats

Networks

Imagine we want to show if five neighboring countries ever invaded one another. We can have a plot like the one in Figure 2.4 for this invented situation.

This imaginary plot can be stored in a spreadsheet like an **adjacency matrix**, as shown in Figure 2.5.

An adjacency matrix will have the same elements as row names and column names, and the intersection will signal if there is a link between them. In this example, the diagonal will not be populated, because I am not assuming that a country can invade itself (there may be other networks when a self-loop is possible). That same information can come in other structures. An **adjacency list**, as shown in Figure 2.6, is a structure where the first element to the left is the source, and every element to the right is a target. The **edge list** is a collection of pairs, where the first element indicates the *source* and the second the *target*. The edge list for our network is shown in Figure 2.7.

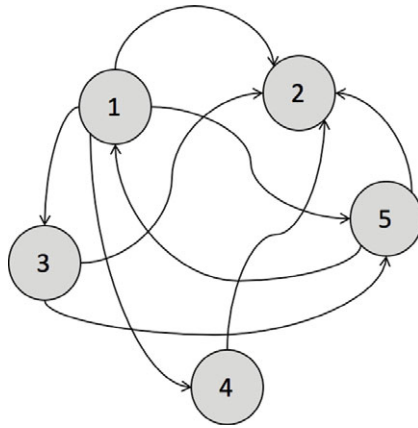


Figure 2.4 Network of invasions

Imaginary network representing a country that ever invaded another one.

	country_1	country_2	country_3	country_4	country_5
country_1		1	1	1	0
country_2	0		0	0	0
country_3	0	1		0	1
country_4	0	1	0		0
country_5	1	1	0	0	

Figure 2.5 Adjacency matrix of network of invasions

```
country_1 country_2 country_3 country_4 country_5
country_3 country_2 country_5
country_4 country_2
country_5 country_1 country_2
```

Figure 2.6 Adjacency list of network of invasions

```
country_1 country_2
country_1 country_3
country_1 country_4
country_1 country_5
country_3 country_2
country_3 country_5
country_4 country_2
country_5 country_1
country_5 country_2
```

Figure 2.7 Edge list of network of invasions

These structures can be stored in tables, so **Python** and **R** can be used without problem. Of course, dictionaries are always an alternative to tables. This structure comes next.

**Dictionaries**

You may be interested in a simple way of storing information about people. As it is a simple task, you want to populate a table with the data you collect from those people. However, even in this simple situations you may be limited by the table.

If you happen to have an online form, you need to plan the form structure that people will see and fill out. Asking for language spoken, for instance, will need a cell filling out; but this is uncomfortable for a person that speaks natively more than one language, and who uses the table. It may also be important for you to know that the person speaks more than one language. If you use a table, you will need to have several columns for languages spoken, but it will be difficult to anticipate how many columns you need (and many empty cells, that will be interpreted as missing values, could appear). In this situation, a dictionary is a great alternative. Our imaginary example is represented in Figure 2.8.

The dictionary is a structure that allows for a particular key (i.e. ‘language’, ‘country’, etc.) to have a collection of values assigned to it. This is what the lower part represents in Figure 2.8. According to Seeger (2009), dictionaries are the base data structure of the NoSQL model, which is precisely working to

NAME	COUNTRY	LANGUAGE	ADDITIONAL LANGUAGE 1	ADDITIONAL LANGUAGE 2	ADDITIONAL LANGUAGE 3
John	USA	English			
Pablo	Spain	Spanish	Basque		
Roger	Switzerland	German	French	Italian	Romansh

TABLE



DICTIONARY

Name:	John				
Country:	USA				
Language:	English				
Name:	Pablo				
Country:	Spain				
Language:	Spanish	Basque			
Name:	Roger				
Country:	Switzerland				
Language:	German	French	Italian	Romansh	

Figure 2.8 Converting data frame to dictionary

promote strategies that overcome the limitations of tables. If you run into open data portals you will for sure deal with this format, as the results are obtained in JSON format. Most information coming from social media applications like Twitter use dictionaries. Every *tweet* is represented by a complex dictionary.

Finally, complex formats such as maps, generally represented using *shape-files*, have a competitor in dictionaries, as they can be implemented more efficiently using these: the GeoJSON and TopoJSON formats.

A dictionary is not a native structure in **R**, but it is in **Python**. **R** generally uses lists to represent a dictionary. Collecting and formatting all these different data types and dataset formats has been widely discussed in Magallanes Reyes (2017).