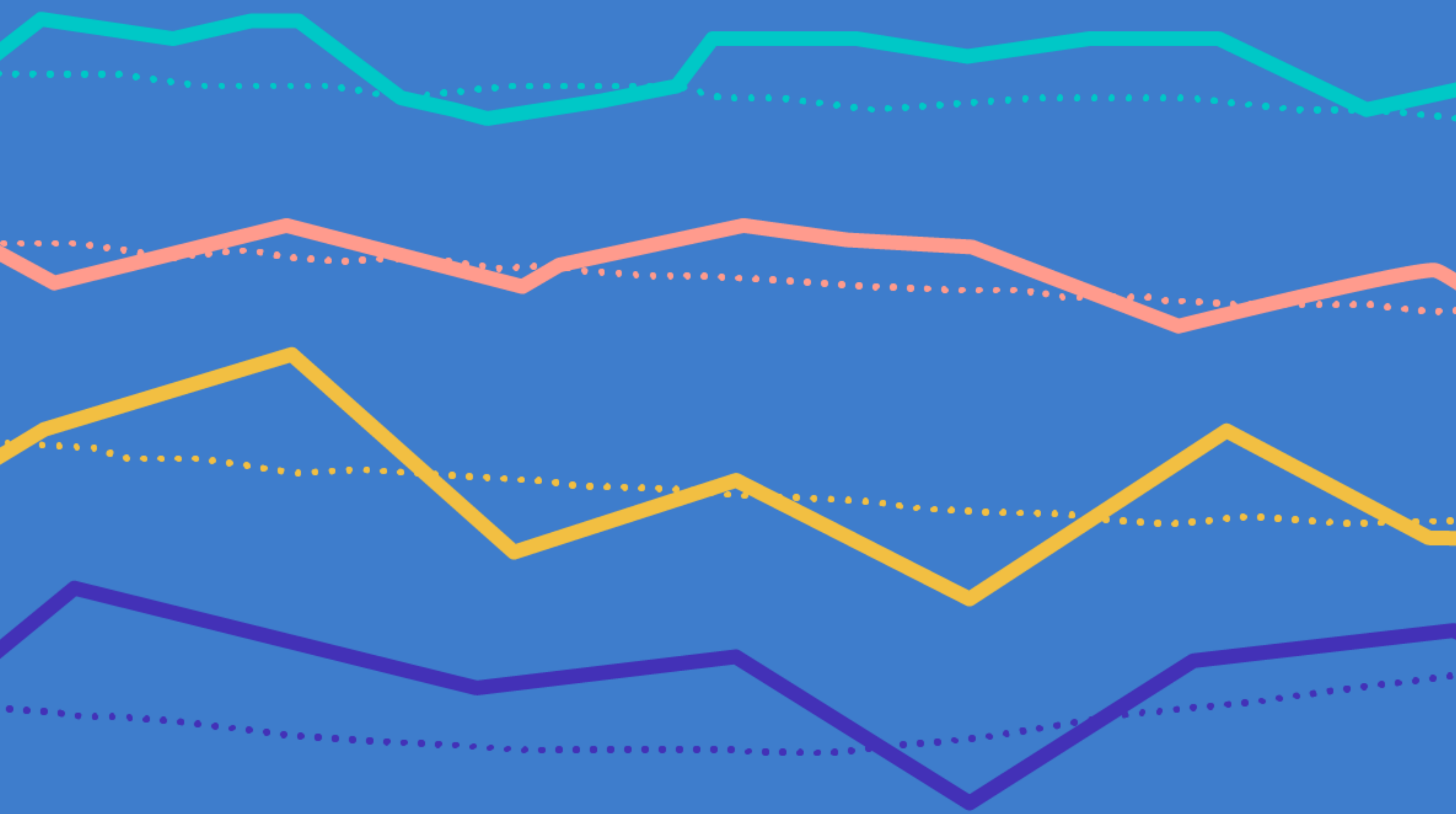
DISPLAYR

RESEARCHSOFTWARE
A DIVISION OF DISPLAYR

Cleaning Survey Data



Analysis and reporting software

displayr.com

Table of Contents

Overview and background

This book describes all the key steps involved in cleaning survey data.

It provides instructions for specialist survey analysis platforms Q and Displayr, as well as for general-purpose stats packages R and SPSS. While the latter are not designed for survey analysis, with effort, time, and the technical skills – it's possible to achieve good outcomes.

What is data cleaning?

Data Cleaning refers to checking and correcting problems in a data file. It is also known as *Data Cleansing and Data Scrubbing.*

The goal is to identify and correct data that is incorrect in some way (i.e., *Dirty Data)*. Poor decisions made at this stage can greatly impact the results of a study. In the worst case, key conclusions are driven entirely by errors that were made when cleaning the data.

This book describes and illustrates the basic processes involved in cleaning a survey data file.

Workflow

The basic workflow consists of the following four states, which are typically performed in an iterative fashion:

1. Obtain a high-quality data file
2. Inspect the data (error identification)
3. Change values
4. Delete respondents

Software

Using Displayr with this book

This book has been written in May 2019. By the time you read it there will be additional features not described in this book.

Using Q with this book

This book has been written using Q5.5.

Using R with this book

This book has been written using R 4.3. The instructions in this book assume a basic familiarity with R. They assume that you know how to find and install packages from CRAN and GitHub, and are using an IDE, such as RStudio.

The following packages are used in this book: `Hmisc`, `psych`, `validate`, `tidyverse`.

If using R, please also see the appendices:

- **Beginner's mistakes for people new to R**
- **The limitations of R for survey analysis**

Using SPSS with this book

This book has been written using the features available in SPSS Statistics 25 Professional Edition.

This book assumes that you have set up the R integration for use with SPSS. This integration comes standard with SPSS Statistics.

Working safely with SPSS Each of R, Q, and Displayr have inbuilt safeguards that mean you can usually undo any errors. SPSS does not. It is easy to accidentally "lose" all your work in SPSS. The trick to avoiding this in SPSS is to always keep multiple versions of files. That is:

- Never, ever, modify the original `.SAV` data file.
- At least once a day when working heavily, save a new copy of the data file.
- Create a single syntax file that includes the entire process of reading the original file, saving it with a new name, modifying the data, and saving it again.

The goal: a cleaned data set with rich metadata

Irrespective of where the data comes from and how messy it is to start with, the endpoint of the data preparation process is:

- One or more *tidy raw data sets*.
- Variables are of an appropriate type.
- Variables are grouped into variable sets.
- The values in the variables have been cleaned, with data integrity issues understood and ideally rectified or at least documented.
- As much information as possible in the data set itself as *metadata,* rather than being documented elsewhere.
- A unique ID variable.

These and related ideas are described in more detail in the **Appendix.**

Obtaining a high-quality data file

The first step in data cleaning is to obtain a high-quality data file.

This chapter describes the most common file formats available, and their strengths and weaknesses.

There are an infinite number of possible formats of data files. However, most data analysis software can only ingest data in a small number of formats, so the first step in tidying data is usually to find an appropriate format.

Not all formats are equivalent, and it is often possible to choose the wrong format for the source data. In this chapter, the formats are ranked from what are typically the worst through to the best:

- The worst formats are *unstructured files.*
- The second-worst formats, and most common formats, are *metadata-poor data files.*
- The best formats are *metadata-rich data files.*

The chapter ends with a discussion of:

- Obtaining metadata
- Creating data files

Unstructured files

As the term implies, *unstructured data* is data that is not organized in a predefined format.

Web pages

Often useful data exists in web pages. For example, sometimes the data may be a table in a webpage. Or, various unstructured types of information in web pages, such as comments and product ratings. The term for extracting data from such pages and converting it to an appropriate form is called *web-scraping.* Web-scraping can be done in each of R, SPSS, Q, and Displayr, using various tools, including the `rvest` package. This is not discussed further in this eBook as there are many web pages and several books describing how to do this.

Poorly-structured Excel files

A poorly structured Excel file is one that is not structured like a **Tidy raw data set**. Such data sets are commonly created when people collect data manually. For example, somebody may manually record experiment or survey results in a questionnaire.

Refer to **Creating your own data file** at the end of the chapter for general tips on creating your own data file.

Log files

A *log file* is a file that records things that occur in software or messages between different people or systems. Sometimes log files are a garbage dump of sort, just outputting data as it occurs without any easy-to-manipulate structure. Such files can be analyzed, but require extensive skills for manipulating text files, and is usually best done using specialist tools (e.g., Perl). If only done occasionally, it may be practically to use the text analysis tools in R. However, this is beyond the scope of this book.

Sometimes, fortunately, log files are created in a way that makes data analysis relatively straightforward. An example of this is *web server logs,* which track which pages have been requested on a website, and may also contain information about users IP addresses, time and/or date, and other related information.

Metadata-poor data files

Metadata-poor files contain data, but little information about what the data means. Such data cannot be interpreted without additional documentation describing its meaning. So it is invariably necessary to obtain a second file, or set of files, called a *data dictionary,* which contain all the **Variable metadata**.

It is almost always preferable to have a metadata-rich file format (discussed below) than a metadata-poor data file with a data dictionary, as usually data dictionaries contain omissions and, even when they don't, add time and error to the analysis process (as discussed below). The exception is that if data needs to be regularly updated, it is sometimes preferable to have automatically updated data (e.g., via an SQL query) without metadata (versus manually updated meta-data rich data).

Fixed column text files

The worst of the widely-used formats is the fixed column text file, where each character in each row represents a column of data. For instance, in the example below, the first character may represent age bands (e.g., 3 may represent 35 to 44), and the second column gender (e.g., 1 = Male, 2 = Female).

```
31
32
12
42
```

Typically, such data also has a *Tidy Data* structure, whereby each row represents an *observation* (e.g., a person), and each column a *variable*. However, this is not guaranteed. For example:

- Sometimes fixed column data contain no return characters (e.g., the data above may be represented as `31321242`).
- Sometimes the first few lines of the data file contains other information (e.g., the name of the study, the number of variables in the study, the number of observations).

Tab-delimited text files

The next step up from a fixed column text file is where tabs delimit the different columns. Typically, but not always, this file format also contains *column names* in the first row.

```
Age	Gender
3	1
3	2
1	2
4	2
```

Comma-delimited text files

An alternative to delimiting with tab characters is to instead use commas.

```
Age,Gender,Attitude
3,1,2
3,2,3
1,2,3
4,2,5
```

An alternative to representing values with numbers is to use more meaningful text. For example:

```
Age,Gender,Attitude
35 to 44,Male,Somewhat disagree
35 to 44,Female,Neither agree nor disagree
Under 18,Female,Neither agree nor disagree
60 or more,Female,Strongly agree
```

This is usually substantially less convenient than numbers. Most data analysis does, under the hood, convert such text back to numbers, and important information is lost along the way. Consider the third variable, `Attitude`. If the text is mapped back to numbers automatically (which is what most software will automatically do), it will likely assign values based on alphabetic order, leading to a value of 1 to `Neither agree nor disagree`, a value of 2 to `Somewhat disagree`, and a value of 3 to `Strongly agree`. Such a coding is not sensible and needs to be rectified prior to any analysis.

When the data is stored in numbers, it is usually relatively efficient to add in lots of labels for multiple columns at the same time. But, when it is stored as numbers it tends to be the case that you end up having to fix every variable one-by-one, which takes much longer.

Excel files

From a basic data storage perspective, Excel files are much like CSV files (discussed in the next section). In practice, Excel files suffer from a number of problems:

- They are limited to being 1,048, 576 rows and 16,384 columns.
- They contain multiple sheets, but typically most data analysis software will ignore all the data except in one sheet.
- People that use Excel tend to often store data in ways that gets ignored when the data is imported which creates bugs. For example, comments, tables of variable definitions, inconsistently formatted comments, chart, pictures.

If your data is in Excel, it is often useful to save it as a CSV file, and then import it back into Excel and check that everything still looks OK.

CSV Files

The problem with a comma-delimited file is that sometimes data contains columns with alternative meanings. For example, a text field describing why somebody has churned to another company may contain the text `I hate you because you are too expensive, difficult to deal with, and suck!` If the data is assumed to be comma delimited, then this will be split into three columns, and the data becomes uninterpreted.

This format, which has the full name of **C**omma **S**eparated **V**alues, is very similar to comma-delimited text files, except there are some additional rules. For example, where text contains commas, it is surrounded in quotes. For example, lines of data end with a return character.[1]

```
Age,Gender,Reason for defection
3,1,
3,2,Forgot to pay
1,2,"I hate you because you are too expensive, difficult to deal with, and
suck!."
4,2,
```

As discussed in the previous section, it is preferable to use numbers rather than text when storing categorical data.

Typically, CSV files have a file extension of `.CSV`.

SQL databases

SQL databases typically return data in either a tab delimited or CSV format. The benefit that SQL has over data files is that:

- It is a live link. That is, a CSV file is a *file* that needs to be generated and then imported. When data is extracted from an SQL database it arrives immediately.

---

[1] This character is invisible to the human eye in most standard software.

- You can use SQL (**S**tructed **Q**uery **L**anguage) to pre-process the data. For example, many of the ways that data can be tidied can be done in a SQL query.

SQL queries can return multiple *results sets* (i.e., tables), but typically when importing data, you want to request only a single set at a time, as this is what most software expects.

XML

E**x**tsensible **M**arkup **L**anguage files are files that are intended to be readable by both humans and machines. Unlike the previous data file formats, XML *can* be rich in metadata. However, as a pretty basic rule, where a data file has an extension of .XML, they tend to be unusable. Typically, it is just a CSV file where somebody has chosen the **Save as XML** option or, worse yet, it is a non-standard format that cannot be interpreted by any widely used data analysis program.

The XML below shows our earlier data. Note that:

1. The data is no longer in the format of *columns.* So, while Age and Gender are *variables* (i.e., properties of each record), they are no longer *columns* in the data.
2. The XML still does not contain any explanation of what the 3 for Age represents.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<data-set xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <record>
            <Age>3</Age>
            <Gender>1</Gender>
      </record>
      <record>
            <Age>3</Age>
            <Gender>2</Gender>
      </record>
      <record>
            <Age>1</Age>
            <Gender>2</Gender>
      </record>
      <record>
            <Age>4</Age>
            <Gender>2</Gender>
      </record>
</data-set>
```

If data is in an XML format, then it is important that it comes in a standard XML format. Some of the most common ones are described below.

JSON text files

Java**S**cript **O**bject **N**otation (JSON) are similar to XML files in their structure, except they store the data in a format that takes up less space. As with XML files, they are only useful when specifically written for the purpose of being read by the data analysis software that you are using.

HTML

Hyper **T**ext **M**arkup **L**anguage describes how to present information, whereas XML shows the structure of the data. Commonly HTML does contain data, whether the data is in a freeform text format (e.g., comments), or highly structured information (such as tables). The tidying of such data, which is a part of *web scraping,* is a specialist topic and is outside the scope of this book.

Metadata-rich data files

Metadata-rich formats are formats that contain more than just the values and names of each variables. It is difficult to understate the benefit of having detailed **Variable metadata** in a data file. It:

- makes it faster for people to do their work, as much less time is spent looking up or cross-referencing information,
- reduces errors that occur when people can't be bothered or fail to look up key information, and
- saves considerable time during reporting when the reporting software automatically uses the metadata in the creation of tables and visualizations.

In survey analysis, where data files contain large number of variables, metadata-rich formats are the norm among people who analyze such data for a living.

The following meta-data rich file formats are ordered in terms of the amount of variable metadata that they contain, from least to most.

SPSS Syntax Files

SPSS Syntax Files are files that contain instructions for creating a data file and are written in the SPSS Syntax Language. In theory, these files contain all the information that is in an SPSS Data File. In practice this format should be avoided as:

- very few programs can read it. SPSS Statistics can read it. Q can read it sometimes.[2] PSPP can read it sometimes.
- it is quite common that files written in this format have bugs and cannot be read.

SPSS Data Files

The SPSS data file (`.sav`) is the most widely used data file format that contains extensive metadata. It is the lingua franca of much of social science survey-based data analysis (e.g., psychology, marketing, sociology, politics).

The actual file format is a *binary format,* which is to say it is all written in 0s and 1s and cannot readily be read by the human eye.

Nevertheless, you can think about it as being essentially like the earlier XML format, except that it also can contain a lot more information about the meaning of the data. It will usually contain:

- The *Variable Label* for each variable. For example, `Attitude` may have the label `How strongly do you agree with the statement 'Data Science is Cool'?`
- Each variable has a set of *Value Labels.* For example, a 1 for `Gender` may mean `Male` and a 2 may mean `Female`.
- One variable may be flagged as a *weight* and another as a *filter.*
- Related variables may be grouped into *Multiple Response Sets.*
- Certain values may be flagged as *Missing Values.*
- The *scale type* of variables will be stored as one of *Nominal, Ordinal,* or *Scale.*
- Information about the date format may be stored.

Triple-S Files

This format is a standard and open data file format[3] for survey data. Typically, this format consists of two files. One file that contains the data, usually as a fixed-column or CSV format, and a second file containing the metadata with extension `.xml` or `.sss` . Additional files are included if the data contains multiple hierarchically-related files (e.g., one file containing variables describing characteristics of households, and a second data file containing variables describing characteristics of people).

The newest format is version 3.0, but most software only reads version 2.0.

The main benefits of the Triple-S format over the SPSS Data File is that it contains more metadata about different types of questions, and usually has a clearer distinction between when questions were not asked versus when they were not seen.

---

[2] See Creating SPSS Syntax Files for Use in Q.

[3] http://www.triple-s.org/

SPSS Dimensions/Data Collection files

This format is created by the modern SPSS Data Collection products and can contain all of the metadata described in **Variable metadata** and **Variable set metadata**.

These files have file extension (`.mdd`) and either:

- A reference to a database that needs to be connected to. This can be done using Q, Displayr, or the more modern SPSS products.
- A .ddf file, which is a database format.
- Some other file format, such as `.sav`, `.csv`, `.sss`. In these cases, often this other file needs to be used and the `.mdd` file is redundant.

Q Packs

This is the file format created by Q and Displayr. It contains:

- All the **Variable metadata**.
- **Variable set metadata**
- Complete instructions for repeating analyses created in the software (i.e., *reproducible reporting*).

The format is a zip file, containing one or more data files (of any of the formats described above), and a `.Q` file that contains all additional metadata and reproducible reporting instructions.

Creating your own data file

In some situations, it is necessary to create your own data file. For example:

- If you have written a software system that collects data, and you want to analyze that data.
- If you have collected data in a manual way (e.g., paper questionnaires) and wish to answer that data.

This section of the eBook describes some general principles to keep in mind when creating a data file.

Type of file format

Ideally, the data will be created in a metadata-rich format. If the resources are not available to do that, some recommendations are:

- Create a CSV file if you are automatically generating the file and it does not contain a large number of variables.
- An Excel file if the file is being generated by manual data entry or cut and pasting.
- An SPSS `.SAV` file if you have collected survey data. Do this by using re-entering the data into a survey monkey or some similar data collection program, and then exporting the data from this. This is a time-consuming process, but it will lead to much fewer data entry errors and better metadata.
- One of the metadata-rich formats if you have written your own survey software (Triple-S is the most straightforward to implement).

Tips for how to structure the data file

1. **The data file should have a rectangular structure**, with rows representing units of analysis and columns for variables. Do not put any **Data set metadata** or **Variable set metadata** into the file, unless you are using one of the metadata-rich formats, as all that this will do is prevent people from reading the file using standard tools.

2. **Row 1 should contain variable names**, where the first character should be a letter. If you are importing into R or SPSS, you should use a single word and avoid any funny symbols. If using Q or Displayr for your analysis, you can use whatever wording you like.

3. **Each subsequent row should contain the data for one unit of analysis**.

4. **Storing data as labels or values.** If the data will be analyzed in R or SPSS Statistics, use values rather than labels to represent unique values. For example, represent males with a 1 and not with the word `male`. By contrast, if using Q or Displayr, instead use the labels rather than the numbers.

   The reason for the difference between the programs is that in SPSS and R the programs will analyze each variable separately when interpreting the labels, so if you have a variable set you can end up with, say, Strongly Agree represented as a 5 in one variable and a 4 in another, whereas Q and Displayr attempt to discern variable set membership when importing data and try to create consistent *value attributes*.

5. **Non-response and other types of missing data.** Respondents who were not asked a particular question (i.e., were intentionally or unintentionally skipped), should have a NA. It is never appropriate to record all missing values in a data file as having a value of 0 (this is very important, as for many binary variables the No response is often coded as a 0, making it impossible to determine which respondents said No and which were not asked the question).

6. Where there are multiple different types of missing data (e.g., where some questions were not asked of some respondents while others were asked but not answered), they should be coded with different values (e.g., NA where not asked to respondents and -99 if asked but not answered). Sometimes it is appropriate to treat missing values for some of the questions as

being equivalent to a "No" response (e.g., giving them a value of 0). For example, if people are asked which brands they have consumed, but are only shown brands that they are aware of. In this instance, the question should be included in the data file twice, once with the NA values and once with the "No" responses instead.

7. **The value for Don't know needs to be different to the value for non-response**. More generally, if there are multiple reasons for missing data, there should ideally be a different value for each.

8. **Mutually exclusive categories should be in a single variable**. Where categories are mutually exclusive, they should usually be represented as a single variable. For example, rather than have one variable indicating who is a male and a second variable indicating who is a female, there should be a single gender variable.

9. **Overlapping categories should be represented as binary variables, unless there is a huge number of categories.** For example, a database stores "which of the following products a person owns" - saving account, checking account, credit card, home loan – each category should be represented as its own binary variable, with a 1 indicating if the person has the product, a 0 indicating if they do not, an NA indicating if this is not known.

   When representing the data using labels rather than numbers, it is better to use a consistent label across variables in a variable set. For example, using `Yes` and `No` to indicate that somebody has a home loan is preferable to using the labels of `Has Home Loan` and `Does Not Have Home Loan`. This is important because it means that when the data is being analyzed, the similar variables can all be analyzed at the same time.

   The exception to this is where there are a huge number of categories and the number of binary variables becomes impractical (e.g., hundreds or thousands), in which case a *max-multi*[4] format can be used, where the first variable shows the first category of the first of analysis, etc.

10. **Use common variable labels and names to show variable set membership.** If there are four variables that indicate which of four products a person owns, it is useful if the names have a common structure with a commonality at the beginning of the name (e.g., q4a, q4b, q4c, and q4d). Similar, the labels should have a commons structure, as this makes it easy for both people and computes to recognize variable sets. For example:

```
Products owned: Savings account
Products owned: Checking account
Products owned: Loan account
Products owned: Credit account
```

4 https://wiki.q-researchsoftware.com/wiki/Multiple_Response_Data_Formats

11. **Remove randomizations and rotations**. For example, if in a survey, one person was asked "Which of these do you prefer? McDonald's, Burger King", and another was asked "Which of these do you prefer? Burger King, McDonald's". A single variable should contain their choice and another variable the order. That is, it is usually extremely painful if instead the first question appears as one variable with a second variable that has an alternative ordering of responses.

12. **Only export text data as strings.** For example, if you were recording in a variable the values pertaining to the number of people purchasing product, the data file should show, say, 1, 4, 3, 2 (ie; numeric data) rather than "1", "4", "3", and "2" (ie: text representations of numbers).

13. **Use a consistent format for dates.** The global standard is `YYYY-MM-DD` (e.g., `2018-11-29`). If you instead go with MM-DD-YYYY or DD-MM-YYY you are potentially creating a world of pain, as in many situations these formats will confuse a machine (e.g., is 01-08-2018 the eighth of January or the first of August)?

14. **Put one type of data in each column**. For example, if the result is 45KG, it is better to have one column containing the value 45 and another the unit of KG. Similarly, it's almost always a bad idea to have a column that contains a whole lot of comma-delimited data.

More detailed instructions for exporting surveys as CSV files can be found at https://wiki.q-researchsoftware.com/wiki/Excel_and_CSV_Data_File_Specifications and as SPSS data files at https://docs.displayr.com/wiki/SPSS_Data_File_Specifications.

Additional tips if creating the file in Excel

If creating a file in Excel for use in data analysis, it is advisable to do *all* of the following in addition to the points mentioned in the previous section:

1. Create the file in the first worksheet (tab) of the Excel Workbook.
2. Make sure the data starts at cell A1, with no blank cells or titles to the left or above the data.
3. Delete any rows and columns below and to the right of the data. It is common that these will contain spaces and things that cause difficulty when reading data, so delete them even if you cannot see anything.
4. Be consistent. For example, if you are representing male as a `1`, do it consistently. Do not do it sometimes as `1`, sometimes as `m`, sometimes as `males`, and other times as `Male`. This is easiest to do using Excel's *data validation* tools (see http://bit.ly/excel_dataval).
5. Don't use comments.
6. Don't use highlights, cell, or font formatting to convey meaning (there is no straightforward way of converting this to analyzable data).

Inspecting the data (error identification)

In practice, data cleaning is an iterative process. You find one problem. You fix it. You find another problem. You fix it.

However, for reasons of clarity this book addresses this iterative process across six chapters.

This chapter describes how to inspect data to find errors in need of cleaning.

The next chapter describes what cleaning involves.

The final four chapters show which buttons to push to perform the inspection and cleaning in each of Displayr, Q, R, and SPSS

The key stages in data inspection typically involve checking:

- Sample size
- Screening criteria
- Data quality for each question and variable
- Routing and filtering instructions
- Missing data patterns
- Respondent quality metrics
- Looking for duplicates
- Unit tests

Checking the sample size

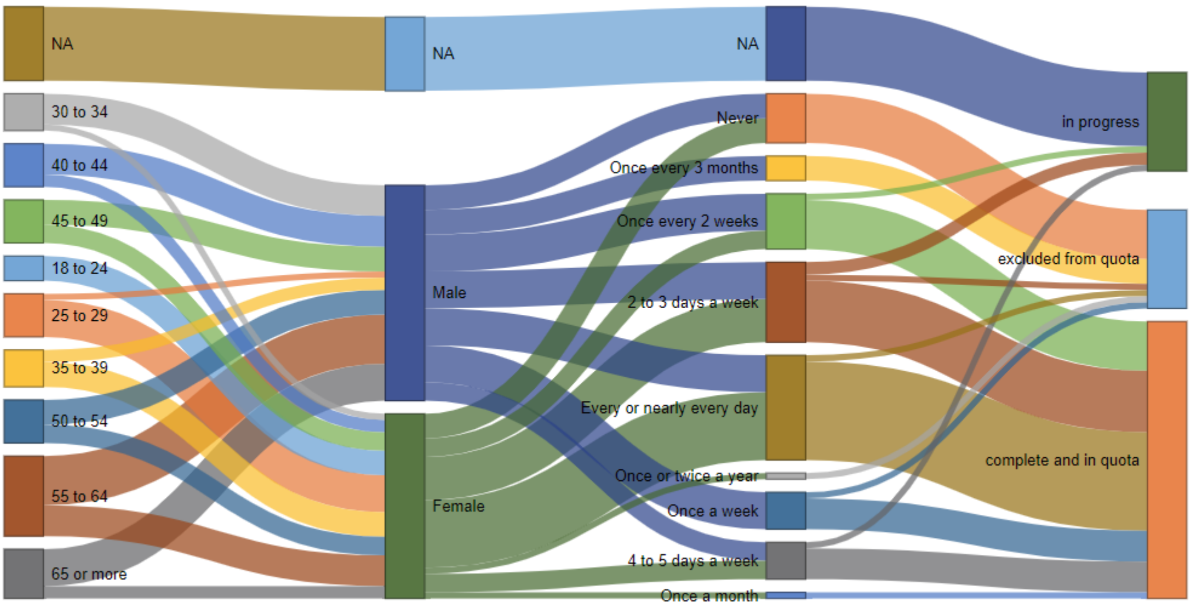
Step one in checking a data set is always to review the number of observations in the data set (i.e., the sample size). If the result is different to what you expect, it can indicate a problem in the data collection or data file exporting processes. Most commonly, when the sample size is bigger than anticipated it will be caused by the data file containing respondents who did not complete the study.

Checking screening criteria

Many surveys include *screening questions* at the beginning to qualify respondents for participation. For example, if doing a study of the cola market, you may have questions designed to check that respondents are aged 18 or more and drink cola at least once a week.

It is usually good practice to check that the screening questions are working as intended. This can be done with a data file that contains only complete respondents, but it is more rigorous to do this using a file that also includes respondents that were excluded due to failing the criteria.

Sankey diagrams are a particularly fast way of checking screening criteria (the alternative is typically to write code). The example below allows us to quickly see that if somebody has been flagged as '*complete and in quota'*, that this means that they were aged 18 or more, have a known gender, and consumed cola once a month or more regularly.



A more rudimentary approach is to use *crosstabs*. For example, the table below shows that we have 2 *Part-time workers* that have no occupation.

Occupation

Duplicate Work status

n	Student	Home maker	Retired	Not working	Part-time worker	Fulltime worker	Don't know/refused	NET
-9	0	0	0	0	2	0	0	2
0	4	0	1	0	0	0	0	5
manager/administrator	1	0	0	0	2	30	0	33
professional	0	0	0	0	3	82	0	85
Associate professional	2	0	0	0	3	43	0	48
Tradesperson and related	0	0	0	0	1	32	0	33
clerical, sales and services	3	0	0	0	19	63	0	85
production/transport worker	0	0	0	0	0	23	0	23
Labourer and related worker	0	0	0	0	1	21	0	22
don't know/refused	0	0	0	0	0	0	0	0
NET	10	0	1	0	31	294	0	336

Table Rules

; and Questions Data Notes

nple Weight: None base n = 336; total n = 725; 389 missing

When looking at a Crosstab it is important to inspect the sample size information. In the case of the Q table above, this is shown at the bottom-right corner. This tells us that the table is only from 336 of the 725 people in the table. The reason for this is that most software excludes people from contingency tables where the people have missing data on either variable. Consequently, when checking data using contingency tables it is generally a good idea to *recode* the data so that missing values appear on the table.

The table below uses the data from all 725 people. It gives us a better understanding of the missing data issues. We can see that, as we might expect, most of the people with *Missing data* for

Occupation (the rows) are *Students, Homemakers, Retired, and Part-time workers,* as we might expect. But there are also some inconsistencies in the data, with the *-9* and *0* categories, and the issues identified in the earlier table.

n	Missing data	Student	Home maker	Retired	Not working	Part-time worker	Fulltime worker	Don't know/refused	NET
Missing data	25	201	62	28	20	53	0	0	389
-9	0	0	0	0	0	2	0	0	2
0	0	4	0	1	0	0	0	0	5
manager/administrator	0	1	0	0	0	2	30	0	33
professional	0	0	0	0	0	3	82	0	85
Associate professional	0	2	0	0	0	3	43	0	48
Tradesperson and related	0	0	0	0	0	1	32	0	33
clerical, sales and services	0	3	0	0	0	19	63	0	85
production/transport worker	0	0	0	0	0	0	23	0	23
Labourer and related worker	0	0	0	0	0	1	21	0	22
don't know/refused	0	0	0	0	0	0	0	0	0
NET	25	211	62	29	20	84	294	0	725

Table Rules

and Questions Data Notes

ple Weight: None base n = 725

Screening criteria can also be checked using *unit tests*, described later in the chapter.

Checking data quality for each question and variable

Most of the heavy lifting in data inspection involves creating tables that summarize the data, inspecting them to look for anything odd, where "odd" most commonly means:

- Poor or incomplete metadata
- Incorrect sample sizes
- Outliers and other unusual values
- Too-small categories
- Incorrect variable types
- Incorrect variable sets

Poor or incomplete metadata

If we cannot determine, with certainty, the meaning of data then we have poor *metadata.* For example, the table to the right is from the phone study and it shows that when asked whether people have a mobile phone, there are two categories, 3 and 5 that are missing metadata (i.e., their labels). Or, perhaps there are two typographical errors and they should be Yes and No (which are stored as a 1 and 2 in the data file).

	%	Count
Yes	99%	715
No	1%	6
3	0%	1
5	0%	3
NET	100%	725

Does respondent have a mobile phone?
SUMMARY
sample size = 725

Another example is shown below. This contains data from the question asking who pays the phone bill. We can see three different problems in this example. First, quantitative data is appearing as Text. Second, there are at least two columns of data (as indicated by the comma). Third, there is no information to describe the meaning of the data. For example, what does a 1 indicate?

Text	Who pays the bill?
55	1
56	1
57	1,4
58	1
59	1
60	1
61	1
62	5
63	1
64	1
65	4
66	1,4
67	2
68	1
69	1
70	1,2

Unusual sample sizes

The *base* for any calculations is the sample size of non-missing values. A rudimentary data check is to make sure that the base for any variable is as expected.

In this table, we can see a host of different issues relating to the sample size. In particular:

- Looking at the bottom of the table we can see that the sample size varies. This could be because there are some kinds of skips or filters in the questionnaire but can also foreshadow a data integrity problem.
- Looking at the bottom row of the table we can see a host of problems. First, note that there is a 99% for the NET. Generally, NETs should be 100%, so the score of 99% indicates that the reported sample size of 725 may include some people who in fact have no data.
- The 0% NET for the last row of second column and the associated very small sample size of 7 also indicates a missing value problem of some kind.

% Sample Size	Unaided	Aided	NET
AAPT/Cellular One	8% 725	16% 668	16% 668
New Tel	2% 725	7% 708	7% 708
One-tel	24% 725	19% 552	19% 552
Optus	88% 725	15% 84	15% 84
Orange (Hutchison)	43% 725	18% 415	18% 415
Telstra (Mobile Net)	83% 725	25% 122	25% 122
Virgin Mobile	24% 725	16% 549	16% 549
Vodafone	78% 725	20% 162	20% 162
Other 1	6% 725	0% 681	0% 681
Other 2	1% 725	0% 721	0% 721
Don't know	0% 725	0% 722	0% 722
NET	99% 725	0% 7	0% 7

Q SUMMARY
sample size = from 7 to 725; total sample size = 725; 718 missing

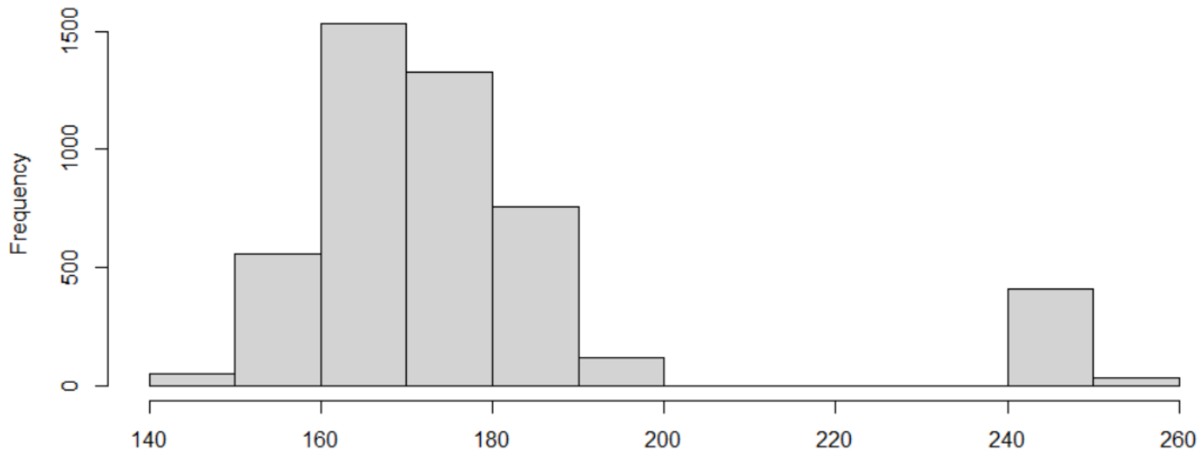
We will continue to explore and fix the issues in this table over the next few chapters.

Outliers and other unusual values

As we saw before, the first question in the phone study asked people if they owned a phone. In addition to Yes and No responses, the data contains values of 3 and 5. These values are unusual (i.e., unexpected).

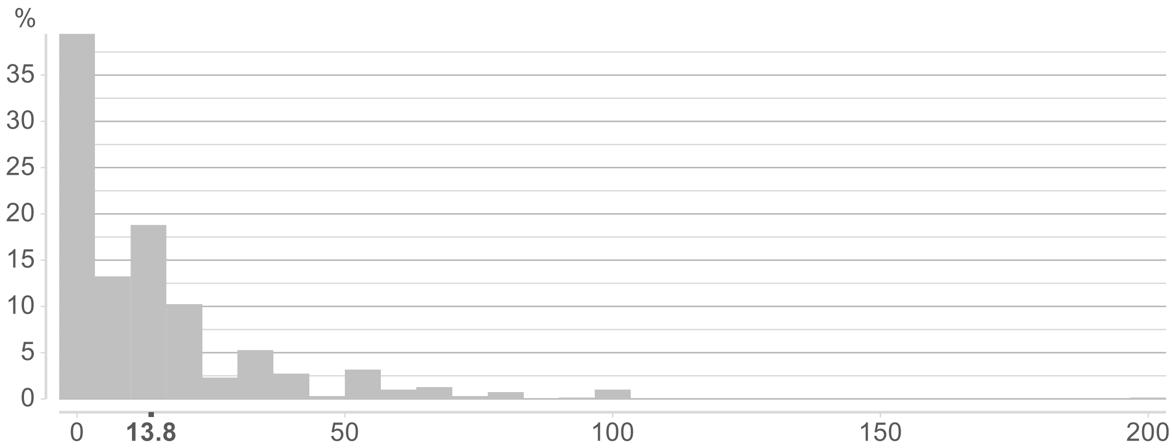
With numeric data, histograms are often more useful for identifying unusual values. The histogram below shows the distribution for the *NHIS* healthy study of heights[5], measured in CM (183CM is 6'). We can see we have a surprisingly large number of people more than 8 foot tall.

[5] The data set is from http://people.ucsc.edu/~cdobkin/NHIS%202007%20data.csv.

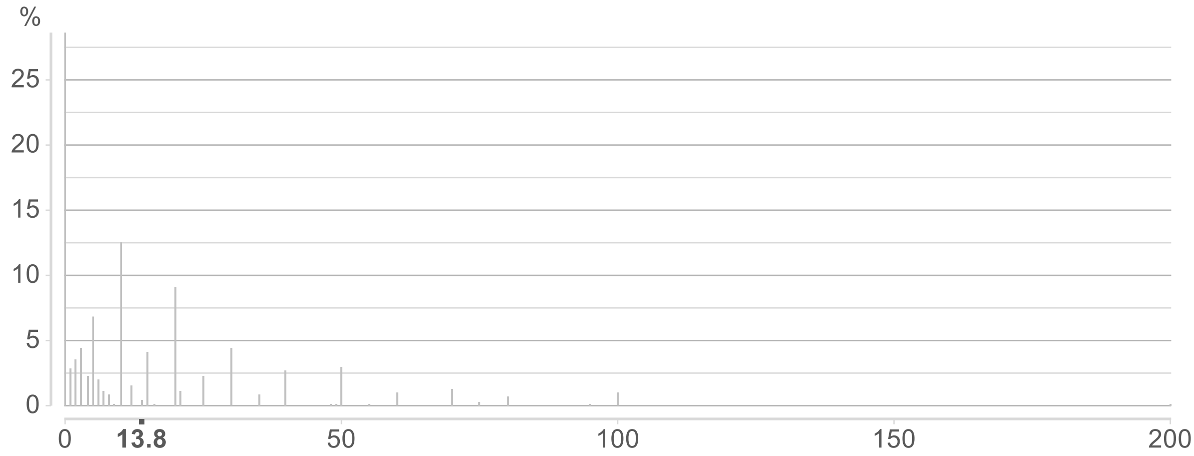


When we have unusual values, an important part of the data inspection process is making a conclusion about the cause of the missing values, as without such conclusions there is no way of correcting the problem. In the case of the *Phone* study, a review of the questionnaire shows that 3 and 5 responses were illegal, suggesting these are typographical errors and need to be treated as missing values. For the NHIS health study, documentation reveals that missing values are recorded as 99 which, when converted from inches to CM, becomes 240.

The Histogram below shows data form the *Phone* data set, showing the number of text messages sent by the respondents. By default, most histograms tend to show a relatively small number of columns, such as in the example below. Such histograms are not particularly useful for manually checking data.



When the goal is to check data, it is usually advisable to increase the number of columns in the histograms (or, to use the jargon, increase the number of *bins*). In the chart below this allows us to see we have a data integrity problem known as *shelving,* whereby people have tended to provide data in round numbers. Of the spikes at 0, 10, 20, 30, 40, 50, 60, 70, 80, and 100, only the one at 0 is plausible.



Shelving is a particularly difficult problem to address. One implication is that if the values are merged into bands, multiples of 10 should not be used. For example, the value of *50 or more* contains almost twice as many observations as *More than 50*.

Too small categories

The table below shows the age data from the *Phone* study. Many of these categories are too small to be used in analysis, suggesting that the categories should be merged.

% n Base n	%	n	Base n
15 and under	0%	1	719
16-19 yrs	10%	73	719
20-24 yrs	32%	230	719
25-29 yrs	15%	106	719
30-34 yrs	3%	25	719
35-44 yrs	5%	35	719
45-54 yrs	28%	200	719
55-64 yrs	5%	36	719
65 and over	2%	13	719
NET	100%	719	719

Incorrect variable types

When data is imported into a data analysis package it is automatically categorized as having a specific *type* of some kind. This type information is used by each of the software to work out how to compute and present outputs, so it is necessary to review and rectify any date issues prior to attempting to clean the data.

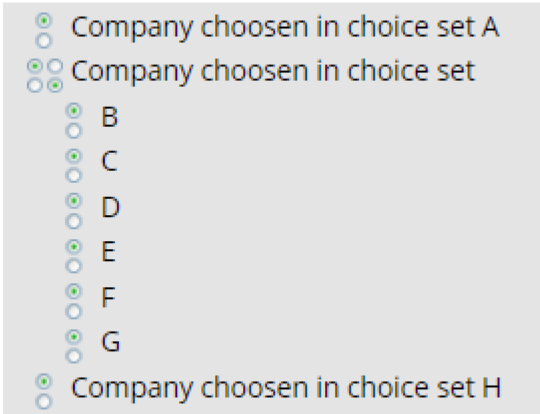
In most programs, the biggest bugbears with variable type issues are:

- Numeric data stored as text
- Dates stored as text or categorical variables
- Categorical data stored as text
- Text data stored as categories

Incorrect variable sets

The concept of a *variable type* relates to a single variable. A *variable set* is a group of variables of the same *variable type* that have a structural relationship of some kind.

Although the concept of a variable set is applicable across much of data analysis, the only field in which it is essential is the analysis of surveys, where the concept of variable sets is closely aligned to the types of questions that are asked in surveys. In surveys, there are various types of multiple response questions (e.g., “tick all that apply”) and grid questions (e.g., “which of these drinks have you consumed at the following locations”).



As an example, the screenshot to the right shows eight variables from the phone study (as they are imported into Displayr). The first and the last variables are appearing on their own, while B through G have automatically been grouped into a variable set (we know that from the icons next to the label).

The purpose of variable sets is to save time in analysis and reporting, as it allows a user to perform a single action and have it applied to all the variables (e.g., merging categories). Because the variable set information in the above is not accurate (as A ang H presumably belong in the one set with B to G) it needs to be corrected.

There are two basic reasons that programs fail to correctly determine variable sets:

- In the case of R and SPSS, neither program supports variable sets by default, although for multiple response questions they do have some support in SPSS as *multiple response sets.*
- In Q and Displayr, the variable sets are typically not correct if there are data quality issues. For example, if the metadata or range of values in categorical variables is inconsistent then they are not (automatically) grouped together.

Checking routing and filtering instructions

Routing and filtering instructions in a questionnaire determine who sees what questions (routing) and which options they are shown (filtering). For example, only asking people their occupation if you know from an earlier question they are employed, and, only asking people to show their attitudes towards brands that they have heard of.

The most basic way of checking routing instructions is to review the raw data. The table below shows the Raw Data for the first five variables in the *Phone* case study. We can see, for example, that we only have Occupation data for people have a Work status of Fulltime worker.

	IID - Interviewer Identification	Does respondent have a mobile phone?	Work status	Occupation	Age	Top of mind awareness
1	853.0	Yes	Fulltime worker	Associate professional	45-54 yrs	Optus
2	854.0	Yes	Fulltime worker	clerical, sales and services	20-24 yrs	Optus
3	855.0	Yes	Retired		45-54 yrs	Optus
4	851.0	Yes	Fulltime worker	manager/ administrator	25-29 yrs	Optus
5	852.0	Yes	Student		20-24 yrs	Optus
6	883.0	Yes	Fulltime worker	Associate professional	45-54 yrs	Telstra (Mobile Net)
7	884.0	Yes	Fulltime worker	clerical, sales and services	20-24 yrs	Vodafone
8	885.0	Yes	Retired		45-54 yrs	Optus
9	881.0	Yes	Student		16-19 yrs	Optus
10	882.0	Yes	Fulltime worker	manager/ administrator	20-24 yrs	Telstra (Mobile Net)
11	713.0	Yes	Part-time worker		20-24 yrs	Optus
12	714.0	Yes	Fulltime worker	professional	55-64 yrs	Optus

Showing 1 to 12 of 725 rows.

Start Previous Next End | Find row

Eye balling like this is a useful way of providing a rudimentary check, but it is typically not enough, as it is too easy to miss exceptions. It can be useful to:

- Use the same processes as used when **Checking screening criteria**.
- Filter or sort raw data
- Filter summary tables.
- Create unit tests (discussed later in this chapter).

Filter or sorting raw data

The table below has been filtered to show only people that did not say *Yes* in Q1 (*Does respondent have a mobile phone*?). It confirms that the data for these respondents is, to say the least, patchy.

	IID - Interviewer Identification	Does respondent have a mobile phone?	Work status	Occupation	Age	Top of mind awareness
40	795.0	5			20-24 yrs	Vodafone
123	1673.0	5			45-54 yrs	
124	1674.0	No			45-54 yrs	
142	1542.0	No				
159	1634.0	No				
594	64.0	No				
603	1673.0	5			45-54 yrs	Optus
604	1674.0	No			45-54 yrs	Optus
643	503.0	No				
665		3			45-54 yrs	Optus

Filter summary tables

An alternative to filtering the raw data, is to instead filter summary tables. For example, if Q2 should only be asked to people that said No in Q1, then you create a summary table of Q2 filtered to only show people that said No in Q1.

Checking for missing data patterns

After having reviewed the sample size, screening criteria, routing and filtering, and the quality of each variable and question, much of the cleaning should have been completed. A particularly useful way of checking the data is to create a *heatmap* or *line chart* showing missing values for each observation in a data set. Such a visualization needs to be easy to zoom and to see the observation numbers, so that the specific problems can be identified, investigated, and remedied.

The resulting visualization is always extremely messy, but don't be too concerned about that - this is a case where the messiness is a product of the sheer amount of information rather than a problem with formatting.

Missing values by case

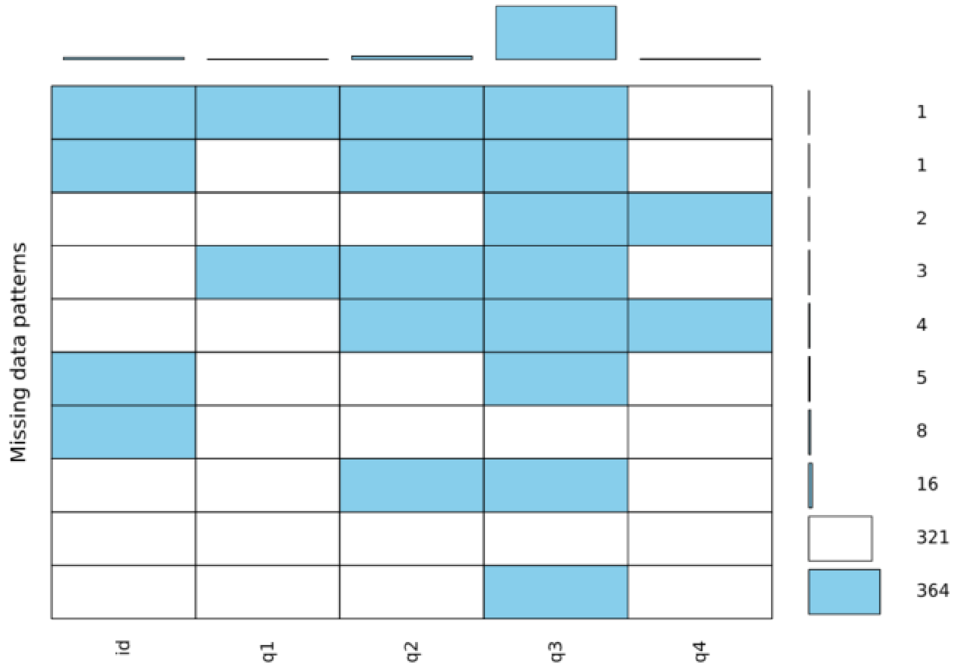
Patterns that can occur include:

- **Vertical columns**. These represent variables affected by routing and filtering and, provided you have followed the more basic data cleaning steps described above, these can be ignored.
- **Short horizontal lines**. These represent multiple variables in a row with missing values. If you look hard, you will see hundreds of such problems in this example. Some are highlighted on the left.
- **Long horizontal lines**. These can be due to routing and filtering but can also indicate the presence of observations (cases) with severe missing data problems. In the visualization above we can see quite a bit of this.
- **Horizontal lines that continue to the end (right) of the data set.** These are symptomatic of incomplete data.
- **Clustering of horizontal lines.** There tend to be three causes of such clustering. Fist, coincidence is always a possibility. Second, they can indicate some fundamental problem in the data collection process. For example, the clustering highlighted on the right side of the visualization above is caused by one interviewer doing a particularly bad job. Third, these can be caused by intentional aspects of the research design (e.g., perhaps the study had a quota for some of its questions, and people were not asked those questions once the quota was exceeded).

When creating visualizations of missing values, it is:

- usually a good idea to only select variables that show survey responses (i.e., don't include administrative variables such as sample source and completion status).
- important to first fix any issues identified in the earlier data cleaning stages. For example, in the earlier inspections we have seen that 3 and 5 appear as values in q1, and these should be recoded as missing. Then a visualization, like the one above, can help determine if these missing values relate to other missing values.

An alternative visualization is made by grouping together observations with the same missing value patterns. The bottom row of the visualization below shows us that the most common missing data pattern is that we have 364 observations that have no data on Q3 but do have data on the other questions. We can also see that have 321 observations with no missing data.



Respondent quality metrics

Each of the metrics described above identifies problems with specific variables or variable sets. One of the remedies for such problems is to modify the data (e.g., merging categories, fixing metadata, recoding values as missing values). A more extreme remedy is to delete observations where the data is regarded is being of insufficient quality. Specific criteria for doing this include:

- Too much missing data.
- Missing data on key questions.
- Not meeting the screening criteria for eligibility to participate in the study
- Inconsistent data. Some caution needs to be exercised if doing this, as there is always a bit of error in questionnaire responses. For example, the difference between Strongly Agree and Agree can be quite hard to judge for a respondent, so many will change their answer if asked the same question twice. The most common criteria for deleting respondents' data is in conjoint and MaxDiff studies, where it is possible to check to see if respondents are just randomly answering (using the RLH statistic).
- Failing lie tests. For example, a question may include a fake brand, and then if respondents choose this it suggests their data is of insufficient quality.
- *Flatlining,* which is when a respondent consistently chooses the same response in a grid question (e.g., chooses all the middle options).

- *Speeding,* which is when the time taken to complete the questionnaire is regarded as being implausible.

Checking for duplicates

The appearance of duplicate data in a data set is a particularly serious problem. If the same person is counted two or more times, the results of a study can become highly misleading.

Sometimes duplicates can be detected by just examining the ID variable for duplicates. In other instances, sets of variables need to be jointly analyzed (e.g., first name + family name + phone number) or responses to open-ended questions.

When dealing with duplicates it is important to understand why they have come into existence prior to trying to resolve the issue. For example, sometimes duplicate entries come into existence in surveys because people make multiple attempts and their first attempt should be treated as representative.

Unit tests

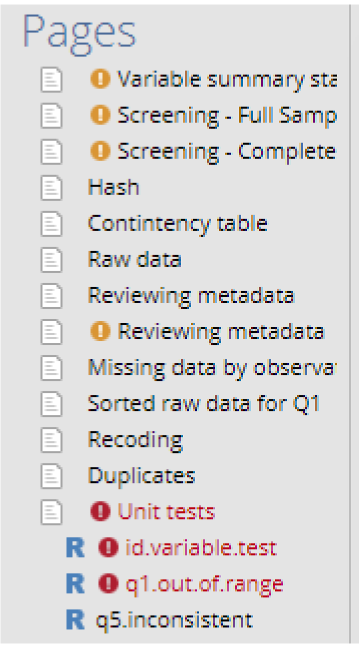
A *unit test* is a term of art in computer programming. The basic idea is that you write a bit of computer code which will show an error if a result is not as expected. For example, it may show an error if somebody has data for occupation but has no job or has a job but no data for occupation, or, if the data contains duplicates.

Unit tests are most useful for longitudinal/tracking projects, as they can entirely automate the process of identifying error, by automatically checking key things whenever a data set is updated.

Common things to test in surveys using unit tests include:

- *Out of Range Errors*, which are situations where values appear in a data file that are bigger or smaller than the expected range of values (e.g., the values of 3 and 5 for q1).
- *Flow Errors,* where the values in one variable are inconsistent with what we would expect based on other data (these are also known as *skip errors, filtering errors,* and *piping errors*). For example, people without employment that have occupations and vice versa.
- *Variable Non-response,* where observations that should contain data do not contain data (e.g., id). This is also known as *Item Non-response.*

- *Checking if data has been coded.*
- *Variable Consistency Errors,* such as having a variable that indicates the number of people in a household which has a smaller value than another indicating the number of children in a household.
- *Lie Tests,* where data has been collected in such a way as to permit identification of people providing deliberately incorrect data. Common ways of doing this include asking questions where you know the answer (e.g., anybody who says *No* when asked “Do you ever lie?” is typically defined as a liar), and comparison based on alternative ways of asking data (e.g., comparing age derived from a birth year with claimed age, where the data has been collected at different points in time).
- *Sum Constraint Errors,* where we specify equality or inequalities regarding the sum of variables. For example:
    - Variables measuring percentages adding up to 100%.
    - Variables measuring time spent doing activities adding up to less than 24 hours in a day.
- *Checking that old results don’t change.*



As an example, the screenshot to the right lists all the pages in a Displayr document. The page name of the page containing unit tests is shown in red (*Unit tests*). This indicates that some of the unit tests therein have failed (i.e., there is a problem with the data). The specific tests have failed are shown in red (*id.variable.test* and *q1.out.of.range*). The user can then click on the page to get more diagnostic information.

Data cleaning

Once you have identified problems (see the previous chapter), you need to clean the data. The basic operations to be performed are:

- Editing values
- Recoding values
- Re-basing data
- Merging categories / capping
- Fixing metadata
- Creating filter variables
- Deleting respondents

Two true stories illustrate the type of mistakes that occur when data cleaning is done poorly. One study identified a large segment of consumers who never used their banks branches. An audit of the research reveals that the segment was entirely an artefact of data cleaning. Many respondents had not actually been asked about their frequency of visiting the bank, and each of these respondents was assigned a value of ‘0’ in the data file, which was the same value as given to people who genuinely had not gone!

In a second more humorous example, a beer consumption survey in Australia identified that beer consumption in the rural city of Darwin was low. This was surprising as beer sales in Darwin are high. An audit revealed the problem. A non-drinker was responsible for entering data entry in a study looking at alcohol consumption, and whenever a respondent had said they had consumed a case of 24 bottles of beer, he assumed they must have made a mistake, and changed the value from 1 case to 1 bottle.

Editing values

Editing values involves replacing the value that has been recorded in the data file with a different value.

	%	Count
Yes	99%	715
No	1%	6
3	0%	1
5	0%	3
NET	100%	725

Does respondent have a mobile phone?
SUMMARY
sample size = 725

Consider again the table to the right. What do the 3 and the 5 mean? The actual question asked only had two options: Yes and No. Ideally, we can work out what the correct values should be. One way to do this is to inspect other related data. The table below shows the raw data for several questions in the survey about phones - sorted so that we can see the problematic data. This table shows us that for respondents 665, 40, and 603 the correct response was *Yes* (based on the other columns which suggest they have a mobile phone). On the other hand, for respondent 123, we need to regard it as a *No* (as regardless of whether it was Yes or No, the data that is consistent with a Yes has not been collected).

	Does respondent have a mobile phone?	Currently on contract?	Do they pre-pay calls?	Time: with current phone	Time: since first phone	Time: until next phone	Still with company with previous contract?	Compa for: pr ious co ract - if contra
665	3	Yes		1 to 6 mths	11 yrs or more	13 to 18 mths		Vodafone
40	5	Yes		13 to 18 mths	2 to 3 yrs	7 to 12 mths		Vodafone
123	5							
603	5	Yes		19 to 24 mths	19 to 24 mths	99		Vodafone

Recoding values and rebasing data

While editing values involves replacing the data of an individual respondent, recoding values involves changing all occurrences of a specific value to some other value.

Consider the table to the right 5 people have said DON'T KNOW. It is reasonable for a person to say they don't know something. But, when the number who have said it is small, as in this case, it is not interesting data. Consequently, it is traditionally regarded as being "dirt" and the fix is to replace it with a *missing value.*

	%	Count
Strongly agree	31%↑	225
Agree a little	31%↑	220
Neither	18%	128
Disagree a little	12%↓	89
Strongly disagree	7%↓	50
DON'T KNOW	1%↓	5
NET	100%↑	717

Surprised by bill size SUMMARY
sample size = 717; total sample size = 725; 8 missing; 95% confidence level

A missing value is a special value that is typically understood by analysis software as being an instruction to automatically filter the table and recompute the values with this data excluded. The table here shows the summary of the same data as above, but the DON'T KNOW value has been set as a missing value (so it disappears from the table). Note that the actual results have changed (e.g., *Disagree* a little is now 13% rather than 12%). The act of recoding data as missing to change the computed results is known as *rebasing* (rebasing can also be done using filters).

	%	Count
Strongly agree	32%↑	225
Agree a little	31%↑	220
Neither	18%	128
Disagree a little	13%↓	89
Strongly disagree	7%↓	50
NET	100%↑	712

Surprised by bill size SUMMARY
sample size = 712; total sample size = 725; 13 missing; 95% confidence level

Other common ways of recoding values when cleaning data are:

- *Capping*. Capping involves replacing values above with the bottom value. For example, if you have a questionnaire asking about the value of houses, you replace all values above $2M with $2M. This prevents very high values from playing too large a role in influencing subsequent analyses.
- *Aligning values with labels.* For example, if a question asks '*How likely are you to recommend this on a scale of 0 to 10*?' typically the answer of 0 is represented as a 1 in the data file, the answer of 2 as a 1, and so on. This is because by default most data collection software assigns a value of 1 to the first category. It is typically a lot better to recode the values to align with the labels, making computation of means accurate.

Merging categories

Another way of cleaning data is to merge together very small categories (e.g., small brands). In both R and SPSS merging is just another example of recoding. In Q and Displayr, however, merging is a distinct idea to recoding. In Displayr and Q, merging does not affect the underlying values of data, and, recoding the values of data does not affect how categories are merged. This means that it is possible to compute the average of a variable and merge it into top two boxes, without either operation affecting the other.

Fixing metadata

Fixing metadata typically involves adding and correcting labels so that all values, variable labels, and question names and other data are appropriately described.

Creating filter variables

As a precursor to deleting respondents it is typically useful to create filter variables which identify different integrity issues. There are three different reasons for doing this:

- The most efficient way to delete respondents is usually to do so via filters (discussed below).
- The process of creating filters effectively documents the data deletion process, ensuring that you can both understand what has been done and also that you can reproduce it in the future (e.g., if updating with a revised data file).
- Filter variables are particularly useful when creating unit tests.

Deleting respondents

Where some aspect of a respondent's data is such that all the respondent's data is non-useful, we typically delete the row of the data file containing their data (i.e., the observation).

The next four chapters describing the mechanics of cleaning data in each of Displayr, Q, R, and SPSS.

Data cleaning in Displayr

This chapter describes the process of performing data cleaning in Displayr.

Checking the sample size

The number of observations and variables is shown on the right-side of the screen whenever a data set is imported or selected.

Number of cases:	725
Variable count:	283
File:	Phone.sav

Checking screening criteria

Sankey diagrams

Visualization > Sankey Diagram and input the variables (over on the right in the *Object Inspector*). For example, select variables used the screeners and a variable showing the status of the respondent (i.e., whether they completed the survey or not).

Crosstabs

Crosstabs are created as follows:

- Drag a variable set onto the page from **Data Sets**.
- Drag a second variable set over the table created by the first, releasing it in the **Columns** box.

Large numbers of crosstabs can be created using **Anything > Report**.

Checking data quality for each question and variable

Two automated tools save a lot of time when checking data quality:

1. **Anything > Data > Miscellaneous > Check for Errors in Data File Construction** scans through the whole data set and performs a whole lot of rudimentary checks, including:

- Variable type and variable set specification
- Blank labels
- There is an appropriate ID variable
- Missing data issues

2. **Anything > Report > Tables for Data Checking** scans through the whole data set and creates tables and histograms that may reveal data integrity issues. It identifies all tables that:
    - Contain cells with sample sizes of less than 30.
    - Contain Don't Know responses.
    - Have blank labels.
    - Have options.
    - Have options chosen by 5 or fewer people, or, less than 1% respondents.
    - Numeric data containing outliers, where a variable contains values that are more than (or less than) 3 standard deviations above (or below) the mean.
    - Variables where the sample size is not the same for all variables.

Histograms to check numeric variables

Visualization > Histogram and input the variables (over on the right in the *Object Inspector*). Set column widths by clicking on the Chart tab (on the right-side of the *Object Inspector*), unchecking **APPEARANCE > Automatic column widths (bins)**, and increasing the **Maximum columns (bins)** to, say, 200.

Missing data patterns

Viewing, filtering, and sorting raw data

In Displayr, there are two ways of viewing the raw data. One is to create table containing subsets of variables, which is done by selecting the variables you want to view under Data Sets, and then selecting **Anything > Table > Raw Data > Variable(s)**. To filter by specific values, press the filter button (funnel), to the right of the column headings. To sort, click the buttons to the left of each heading.

The other approach is to use the **Data Editor**, which you can open by selecting one or more variables under Data Sets, right-clicking, and selecting **View in Data Editor**. To sort, click the buttons to the right of each column heading. It is possible to apply filters at the top of the **Data Editor**, but they have the effect of coloring the rows to identify cases that match the selected filter. You can use the magnifying glass to search for specific terms in the data, and you can choose whether to display the raw data as **Labels** or **Values**.

Missing values by case visualization

Anything > Data > Missing Data > Plot by Case and select the relevant variables (over on the right-hand side in *Object Inspector*). You can click and drag with your mouse pointer to zoom in to view specific cases and variables. Additional information may be available by hovering over the visualization.

Missing data patterns visualization Displayr

Anything > Data > Missing Data > Plot of Patterns and select the desired variables (over on the right-hand side in the *Object Inspector*). It is usually a good idea to only select a relatively small number of variables.

Checking the data for flatlining

Anything > Report > Straight-Lining/Flat-Lining. The feature will run on the Data Set selected and produces output at the end of the **Pages** tree.

Checking data for Duplicates

To create a report showing duplicates, press the **Calculation** button, draw a box on the page with your mouse, and paste the code below into the **R CODE**, replacing `` `IID - Interviewer Identification` `` with the name or label of your ID variable.

```
id.variable = `IID - Interviewer Identification`
dupes = duplicated(id.variable)
dupe.values = id.variable[id.variable %in%
unique(id.variable[dupes])]
sort(table(dupe.values, useNA = "always"), decreasing = TRUE)
```

This will create a table where each row number shows the ID variable that is duplicated, and the number to the right shows how many times it appeared.

	[,1]
NA	15.0
61	6.0
571	6.0
572	6.0
573	6.0
574	6.0
575	6.0
106	5.0
122	5.0
124	5.0
5417	5.0

Showing 1 to 11 of 25 rows.

Start Previous Next End

To create a variable that marks which values are duplicates, **Anything > Data > Variables > New > Custom Code > R – Numeric** and use the following code (replacing `` `IID - Interviewer Identification` `` with your ID variable).

```
duplicated(`IID - Interviewer Identification`)
```

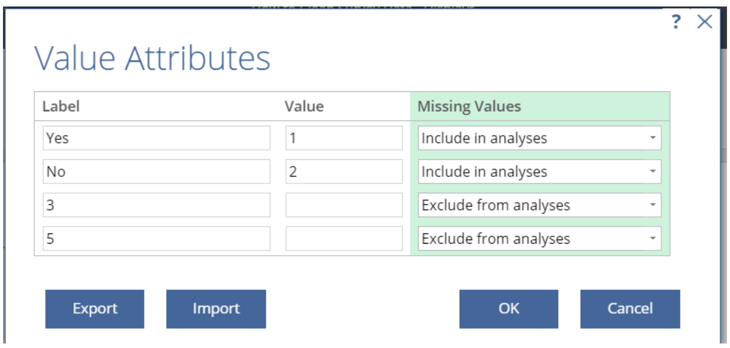
To use multiple variables to defined duplicates, modify the first line to be:

```
id.variable = paste(id, age, gender, occupation)
```

Cleaning the data

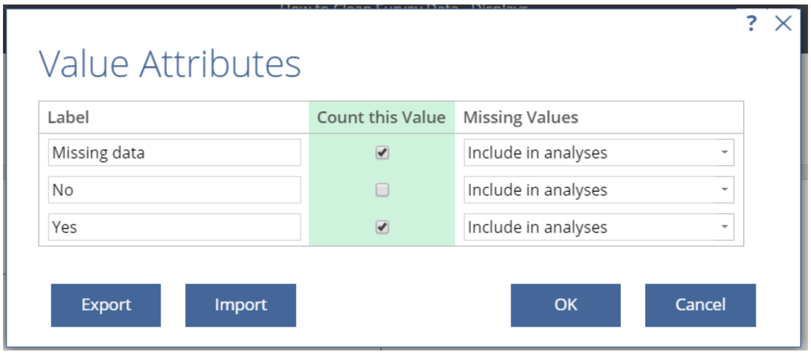
Recoding and rebasing

Recoding and rebasing data in Displayr is done by selecting a variable or variable set in the **Data Sets tree** (bottom-left of the screen) and going to **DATA VALUES > Values** or **Missing Data** on the right-side of the screen. This opens the *Value Attributes* dialog box, where the user can change values and specify **Missing Values**. Missing values can also be changed programmatically by creating new variables and using code.

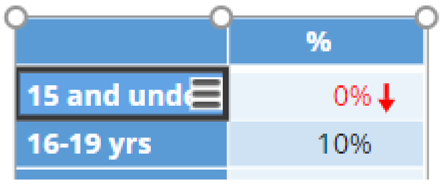


Count this Value

With **Binary – Multi** and **Binary – Grid** variable sets, the **Value** column is replaced by a **Count this Value** column. This is useful both as a way of creating top two box scores, and as a way of dealing with filtered options in a questionnaire. For example, if a question asks people which brands they consume, but has missing values for people known to consume the brand (e.g., known to be current customers, so not asked the question), we can thus recode this by changing the **Missing Values** setting to **Include in analyses** and ticking **Count this Value**.



Merging

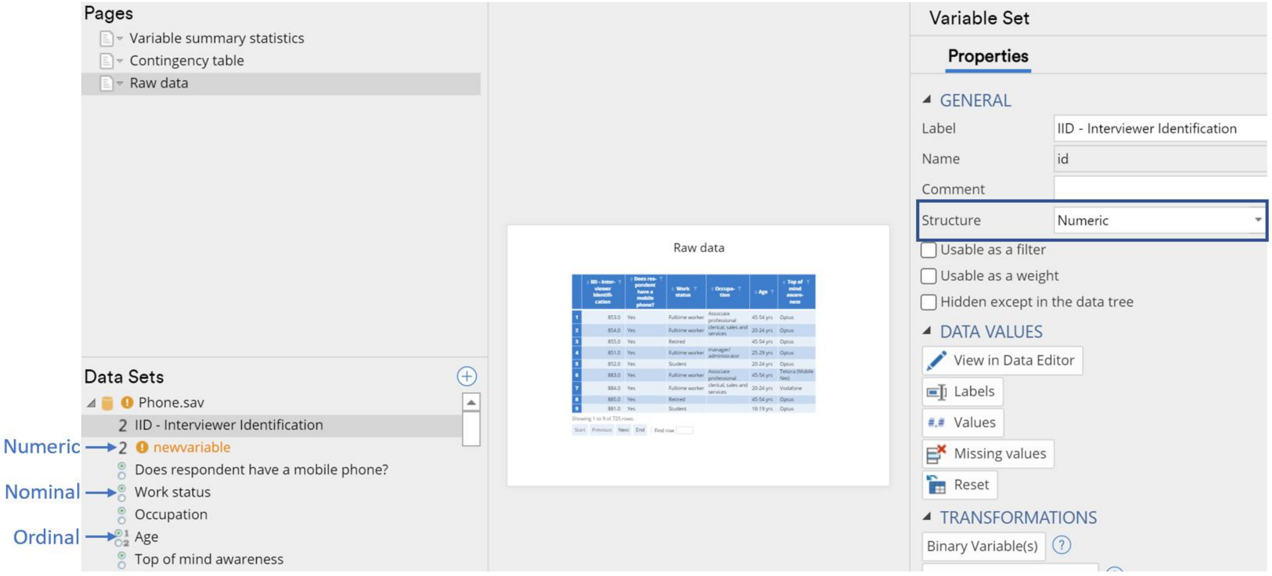


Categories are merged by creating a table and then clicking on the row or column headings until three horizontal lines appear. These are then dragged onto other row or column headings to cause them to merge. You can also select two or more row or column headings in the table and use options at the top of the screen, like **Combine > As One Category** or **Combine > As New Category**.

Displayr will reapply decisions about how to merge categories and change their values whenever the underlying data is used again. That is, a change made on one table will also update all other tables using the same variable set. To avoid this, first duplicate the variable set by selecting it under Data Sets and then using the **Duplicate** button, and then modify the new copy.

Changing variable set structure

Displayr automatically groups together variables into variable sets when the data is imported. Displayr indicates the structure of variables by icons in the data tree. If a variable is selected, its **Structure** is shown in the *Object Inspector* (on the right).



Creating and modifying variable sets

In particularly messy data sets, such as the phone case study, the classification of variables into variable sets can be inaccurate, and Variable Sets are manually created by:

- If variables have been incorrectly grouped, select the Variable Set in the **Data** tree (bottom-left of the screen), right-click, and press **Split**
- Select the variables you wish to combine, right-click, and select **Combine**
- Select an appropriate *Structure* for the *Variable Set* (this option is on the right-side of the screen). See https://docs.displayr.com/wiki/Structure for an overview of the different types of *Structures*.

Fixing metadata

The easiest win in terms of fixing metadata is often **Obtaining a high-quality data file**.

When data is imported into Displayr, common HTML tags are automatically removed from labels.

Displayr also has a powerful find and replace function. Click into the search bar at the top of the screen, and then click **This Document** to the right to bring up the find and replace section on the left. Two things that make it particularly powerful for cleaning metadata are:

- You can customize how the search uses, including:
  - Controlling where the search occurs
  - Using wildcards (expand out **Advanced)**
  - Ignoring punctuation
  - Capitalization
- A panel appears showing you all the search results so you can check that nothing has been found or changed inadvertently.

Creating filter variables

See https://www.displayr.com/5-ways-create-a-filter/ for a description of the main ways of creating filters in Displayr.

Creating a filter variable based on ID

Sometimes you know the ID values of respondents that you wish to delete. If so, you create a filter variable as follows:

- **Anything > Data > Variables > New > Custom Code > R - Numeric**
- Paste in the code `ID %in% c(123:125, 311)` replacing `ID` with the name of your ID variable and updating the ID's to be deleted. In the code example here, IDs 311 and from 123 to 125 are included in the filter.
- Check **Usable as a filter** (above the **R CODE**)

Creating a filter variable based on case number

Sometimes you may want to delete by observation number (also known as case number or respondent numbers). It is generally a bad idea to do this, as such deletion rules don't work when you revise data files, as often the observation numbers will be inconsistent across data sets.

The solution to this is to instead identify some other unique variable and use it is as a substitute. For example, identify the values of the ID variable or some text variables, and instead use these when creating the filter.

If you do want to delete data using case number, it is important to do this with great care, as the position of observations in the data set will change depending on what else is done. For example, if the 72^{nd} and only 72^{nd} respondent has already been deleted, then `ID[72]` will return the value of the 73rd rather than 72nd respondent.

Deleting observations

Once you have created a filter:

- Click on the data set (in the **Data Sets tree**, bottom-left of the screen)
- Over on the right, set **INPUTS > Unique identifier** to an ID variable. This ensures that if you revise the data file, the same respondents will be deleted again. If your final doesn't have an ID variable try and get one which has it; otherwise, choose **[Use case number]**.
- Right-click a variable under **Data Sets** and select **View in Data Editor** to bring up the **Data Editor**
- Use the **Filter** menu at the top of the **Data Editor** to apply the filter which corresponds to the observations you wish to delete. These rows will be highlighted in green.
- Right-click one of the row numbers on the left side of the **Data Editor** and select **Delete Row(s) Matching Filter**.

Note that you can also delete a selection of rows by clicking **Delete Row(s)**.

A strategy for safely deleting observations

A strength of Displayr is that you can always recover any deleted observations. However, there are some practical issues that can cause problems if not addressed:

- Displayr deletes the observations currently selected by the filter. It does not re-apply the deletion rule automatically. For example, if you deleted respondents for flatlining, and then reimported a revised data file with new respondents in it, any new respondents who flatlined would **not** be automatically deleted.
- When you delete by case number, the case numbers update based on previous deletion, as mentioned in the previous section
- When Displayr shows the row number on the sides of tables, it shows the original row numbers (prior to any deletion). This means that if you have already deleted respondents you need to exercise care when deleting any further respondents based on observation number.
- It can be easy to lose track of which respondents were deleted and why, which makes it hard to describe what's been done, hard to check, and hard to repeat the process in the future.

Fortunately, you can create two safeguards that jointly avoid these issues.

Safeguard 1: Creating a consolidated filter variable for deleting cases

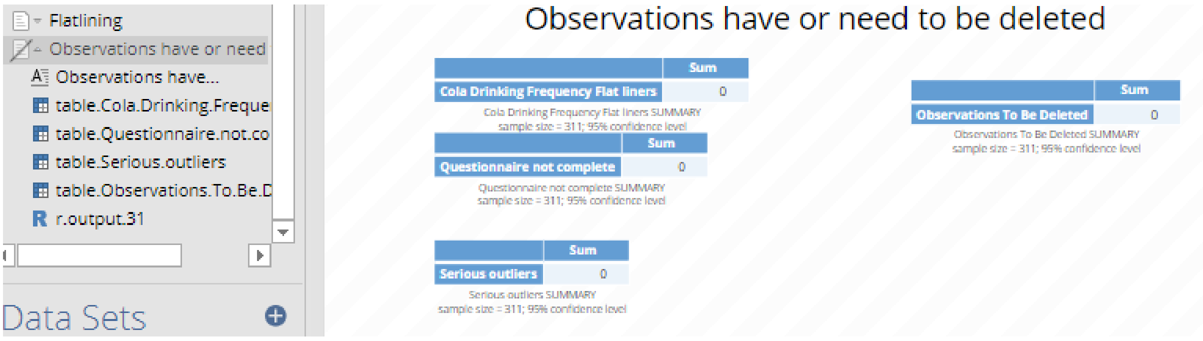
It is good practice to create a single variable for each distinct reason to delete the respondents, with a `TRUE` for each problematic observation. Then to create a single consolidated filter variable for people with missing data on any of these variables. For example, the code below creates such a composite variable, where `|` means *or*. By using such a single consolidated variable (**Anything > Data > Variables > New > Custom Code > R - Numeric**), all you need to do when importing a revised data file is to delete using this variable.

```
`Cola Drinking Frequency Flat liners` |
`Questionnaire not complete` |
`Serious outliers`
```

Safeguard 2: The hidden filter page with error

The basic idea here is to create a page that documents all of your data integrity issues and alerts you when you need to delete some observations. An example of such a page is below. The three tables on the left show filter variables that identify the number of respondents (**STATISTICS > Sum**) to be deleted for different reasons. The table at the top right shows that 16 observations in total need to be deleted. The red box is an error message telling us that we need to delete observations. As the page has an error it will also appear in the **Pages tree** so is unmissable, which ensure that you have forget to delete observations.

Once you have such a page, then when you delete the observations, the error disappears, as shown below. And, if you update a revised data file that requires further deletion of cases, the error will re-appear.



Such a page is created as follows:

1. Create a separate variable for each source of missing data, with a `TRUE` (or 1) and a `FALSE` (0) indicating whether the case should be deleted or not.
2. Drag each of the variables describing a reason for deleting data onto a page so that summary tables are created for each of them. Choose **Inputs > STATISTICS > Cells > Sum** to show the totals
3. Also create a table showing the number of people in the consolidated variable.
4. Create a unit test. This is discussed in the next section.

Unit tests

There are a variety of ways of setting up unit tests in Displayr. The most straightforward is to:

1. Create a new blank page at the top of the document and give it the name of `Unit tests`
2. Hide the page with the **Hide** button at the top of the screen
3. Create each unit test by clicking **Calculation** and drawing a box on your page**,** and then entering a snippet of code that performs the unit test.

When a unit test is failed, this will cause the page name in the **Pages tree** to go red. You can then either click on it, or, click the expander to the left of the page name, to see the status of each of the individual errors.

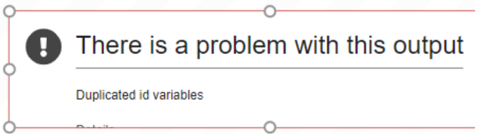
Checking for duplicate IDs

Where `my.id` is the name of your id variable, the following code will test for the existence of any duplicates.

```
if(any(duplicated(my.id)))
    stop("Duplicated id variables")
id.variable.test = "No duplicate IDs"
```

The basic structure, which is applicable to all unit tests in Displayr is:

- The first line checks to see if there is a problem. In this case, `duplicated` returns a vector of `TRUE` for every duplicate and `FALSE` otherwise. So, if a particular ID value appears three times, the first of these will have `FALSE` and the two subsequent values, the duplicates, will have a `TRUE`. Then, `any` evaluates as `TRUE` if there are any `TRUE` values.
- `stop` creates an error if a `TRUE` is returned above, printing on the screen the text in quotes, as shown to the right.



- The final line names the unit test as `id.variable.test` which is what appears at the left of the page in the **Pages tree**. And, the text is what is shown on the page if the unit test is not failed.

Missing values in a variable

The following unit test checks to see if there are any missing values in q1.

```
if(any(is.na(q1)))
    stop("Missing values in q1")
missing.values.q1 = "No missing values in q1"
```

Out of range values

The following code checks for any values other than `Yes` or `No` in q1, where:

- `unique` returns a vector of the unique values in `q1`
- `unique(q1) %in% c("Yes", "No")` returns vector containing of the same length as the vector returned by unique, where a `TRUE` is shown if the unique value is either `Yes` or `No` and a `FALSE` is shown otherwise
- The `!` before unique means *not* and has the effect of reversing the `TRUE` and `FALSE` values

```
if(any(!unique(q1) %in% c("Yes", "No")))
    stop("Out of range in Q1")
q1.out.of.range = "Out of range values in Q1"
```

This next example looks for out of range values for numeric variables:

```
max.q25 = max(q25, na.rm = TRUE)
min.q25 = min(q25, na.rm = TRUE)
if(max.q25 > 50)
    stop("The highest value in q25 is ", max.q25)
if(min.q25 < 0)
    stop("The lowest value in q25 is ", max.q25)
q25.values.between.0.and.50 = "Q25 values are in the expected range"
```

Checking response option filtering

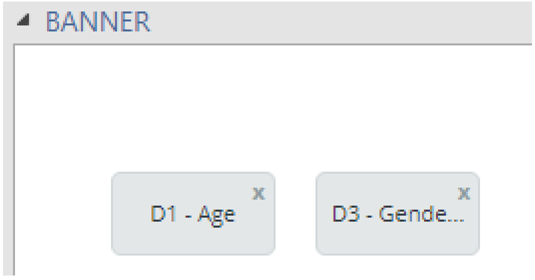
The code below checks for correct implementation of filtering of options in two multiple response questions. The first of the questions, `Unaided awareness`, has structure of **Binary – Multi** and should contain a 1 if a respondent has indicated they were aware of a brand and a 0 otherwise. The second variable set, `Aided Awareness`, should show a missing value if the person has a 1 in `Unaided awareness`.

```
q5 = NULL
n.var = ncol(`Unaided awareness`) - 1 # Excluding NET
for (i in 1:n.var){
    unaided = `Unaided awareness`[, i]
    aided = `Aided awareness`[, i]
    consistent = unaided == 0 & !is.na(aided) | unaided == 1 & is.na(aided)
    if (!all(consistent)) {
        first.inconsistent = (1:length(consistent))[!consistent][1]
        stop("Inconsistencies in Q5_", i, "; first inconsisent observation ",
first.inconsistent)
    }
}
q5.inconsistent = "Q5 data is consistent"
```

(Note: in phone.sav 'Unaided awareness' and 'Aided awareness' are by default set into a grid called 'Q'. You will need to split and/or combine the variables into their respective sets to make it work).

Changes in historical results in trackers

Sometimes data problems can cause historical results to inadvertently change (e.g., due to data cleaning processes being inconsistent). Unit tests can be created to avoid this as follows:



1. Create a new page
2. Drag the new page under the unit tests page
3. Select a key variable (e.g., age) and press **Anything > Data > Variables > New > Banner**
4. Drag across any other key variables (e.g., gender) to the right of the first variable, as shown to the right
5. Drag the BANNER onto the page
6. Drag across the wave or date variable onto the table and release it on the **Columns** field of the current table.
7. Click on the table and select **STATISTICS > Cells > Count** and remove the percentages. You should now have a table like the one to the right
8. Click on the **Properties** tab (right of the screen) and change the **Name** field's contents to `latest.results`

Count		Jan-Mar 17	Apr-Jun 17	Jul-Sep 17
D1 - Age	18 to 24	26	23	25
	25 to 29	23	25	26
	30 to 34	19	20	19
	35 to 39	23	23	27
	40 to 44	21	26	21
	45 to 49	15	17	16
	50 to 54	28	24	18
	55 to 64	31	29	34
	65+	14	13	14
	NET	200	200	200
D3 - Gender	Male	103	100	96
	Female	97	100	104
	NET	200	200	200

BANNER by Date of Interview
sample size = 600; 95% confidence level; Significance: Compare to rest of data

9. Click **Calculation** and draw a box on the page**,** and then paste in the code `snapshot = latest.results` and press **CALCULATE**. This will create a new table with the same data as in the crosstab.
10. Uncheck the **Automatic** updating at the top-right of the screen. This will ensure that when future data is updated this table remains unchanged. When you need to update further waves, remember to calculate again on this prior to the update
11. Click **Calculation** and draw a box on the page**,** and then type in the following code, which will create a table containing all 0s, showing the current difference between the tables

```
n.waves = NCOL(snapshot)
difference.between.results = latest.results[, 1:n.waves] - snapshot
```

12. Click **Calculation** and draw a box on the page**,** and then paste in the following code for the unit test, which will give an error if the data changes when the results are updated.

```
if (any(difference.between.results != 0))
    stop("Historical results have changed")
consistency.over.time = paste("Results have not changed for first",
NCOL(difference.between.results), "waves")
```

Data cleaning in Q

> This chapter describes the process of performing data cleaning in Q.

Checking the sample size

In Q, the number of variables is always shown at the bottom-right of the **Outputs** tab, as **total n**.

base n = 710; total n = 725; 15 missing

The number of variables is found by going to the **Variables and Questions** tab and scrolling to the bottom.

Checking screening criteria

Sankey diagrams

Create > Charts > Visualization > Sankey Diagram and select the variables that are used in the screeners and a variable showing the status of the respondent (i.e., whether they completed the survey or not)

Crosstabs

Select the first variable or question in the **Blue Dropdown menu** and the second one in the **Brown Dropdown menu**.

Checking data quality for each question and variable

Two automated tools save a lot of time when checking data quality:

1. **Automate > Browse Online Library > Preliminary Project Setup > Check for Errors in Data File Construction** scans through the whole data set and performs a whole lot of rudimentary checks, including:
   - Variable type and variable set specification
   - Blank labels
   - There is an appropriate ID variable
   - Missing data issues
2. **Automate > Browse Online Library > Preliminary Project Setup > Tables for Data Checking** scans through the whole data set and creates tables and histograms that may reveal data integrity issues. The tables are highlighted in yellow, things it investigates includes. It identifies all tables that:
   - Contain cells with sample sizes of less than 30.
   - Contain Don't Know responses.
   - Have blank labels
   - Have options
   - Have options chosen by 5 or fewer people, or, less than 1% respondents
   - Numeric data containing outliers, where a variable contains values that are more than (less than) 3 standard deviations above (below) the mean
   - Variables where the sample size is not the same for all variables

Variable types and variable sets are most easily reviewed on the **Variables and Questions** tab. This tab also permits viewing of range, mean, and sample size for all variables in a study, by pressing the button at the top of the screen. The resulting output is shown below.

	Name	Label	Min	Mean	Max	n	Variable Type
1	id	IID - Interviewer Identification	21.0	1,409,498.0	200,029,169.0	710	Numeric
2	q1	Does respondent have a mobile phone?	1.0	1.0	5.0	725	Categorical
3	q2	Work status	1.0	3.8	6.0	700	Categorical
4	q3	Occupation	-9.0	3.5	7.0	336	Categorical
5	q4	Age	1.0	4.7	9.0	719	Ordered Categ
6	q5	Top of mind awareness	1.0	5.8	99.0	712	Categorical

Alternatively, **Create > Tables > Descriptive Statistics** and select the variables of interest; this approach produces more detail, but is not practical if wanting to view data for all the variables in the file.

Histograms to check numeric variables

Select the variable of interest in the **Blue Dropdown menu**, and select **Show Data as** (from top-middle of the screen) and **Histogram.** The column widths are set using **Chart > HISTOGRAM BINS** (on the right-side of the screen).

Missing data patterns

Viewing, filtering, and sorting raw data

In Q, there are two ways of viewing the raw data. One is to create table containing subsets of variables: **Create > Tables > Raw Data > Variables** and select the variables you want. To filter by specific values, press the filter button (funnel), to the right of the column headings (see the image above). To sort, click the buttons to the left of each heading.

The other approach is to use the **Data** tab, which shows all the variables. Sorting is done by right-clicking on columns and choose the desired sort, or, via **View > Sort**. It is possible to apply filters to the **Data** tab, but they have the effect of coloring the rows, so usually it is more useful to instead sort the table by the variables that you want to filter by.

Missing values by case visualization

Automate > Browse Online Library > Missing Data > Plot by Case and select the relevant variables. Additional information is available by hovering over the visualization. You can click and drag with your mouse pointer to zoom in to view specific cases and

Missing data patterns visualization

Automate > Browse Online Library > Missing Data > Plot of Patterns and select the desired variables. It is usually a good idea to only select a relatively small number of variables.

Checking the data for flatlining

Automate > Browse Online Library > Preliminary Project Setup > Identify Questions with Straight-Lining/Flat-Lining. The feature will run on the Data Set selected and produces output at the end of the Pages tree.

Checking data for Duplicates

To create a report showing duplicates, press **Create > R Output**, paste in the code below, replacing `` `IID - Interviewer Identification` `` with the name or label of your ID variable, and press **Calculate**.

```
id.variable = `IID - Interviewer Identification`
dupes = duplicated(id.variable)
dupe.values = id.variable[id.variable %in%
unique(id.variable[dupes])]
sort(table(dupe.values, useNA = "always"), decreasing
= TRUE)
```

	[,1]
NA	15.0
61	6.0
571	6.0
572	6.0
573	6.0
574	6.0
575	6.0
106	5.0
122	5.0
124	5.0
5417	5.0

Showing 1 to 11 of 25 rows.

Start Previous Next End

This will create a table where each row number shows the ID variable that is duplicated, and the number to the right shows how many times it appeared.

To create a variable that marks which values are duplicates, **Create > Variables and Questions > Variable > R Variable > Numeric Variable** and use the following code (replacing `` `IID - Interviewer Identification` `` with your ID variable).

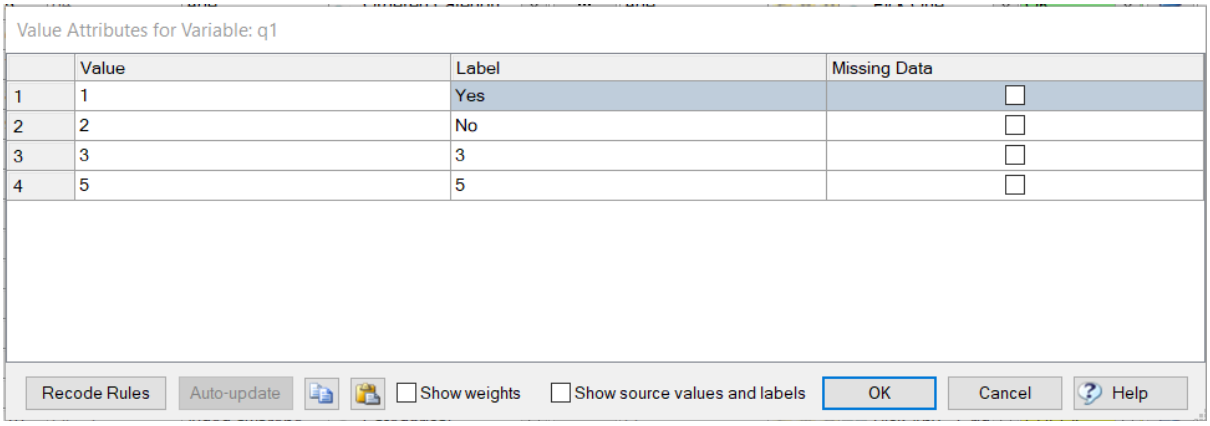
```
duplicated(`IID - Interviewer Identification`)
```

To use multiple variables to defined duplicates, modify the first line to be:
`id.variable = paste(id, age, gender, occupation)`

Cleaning the data

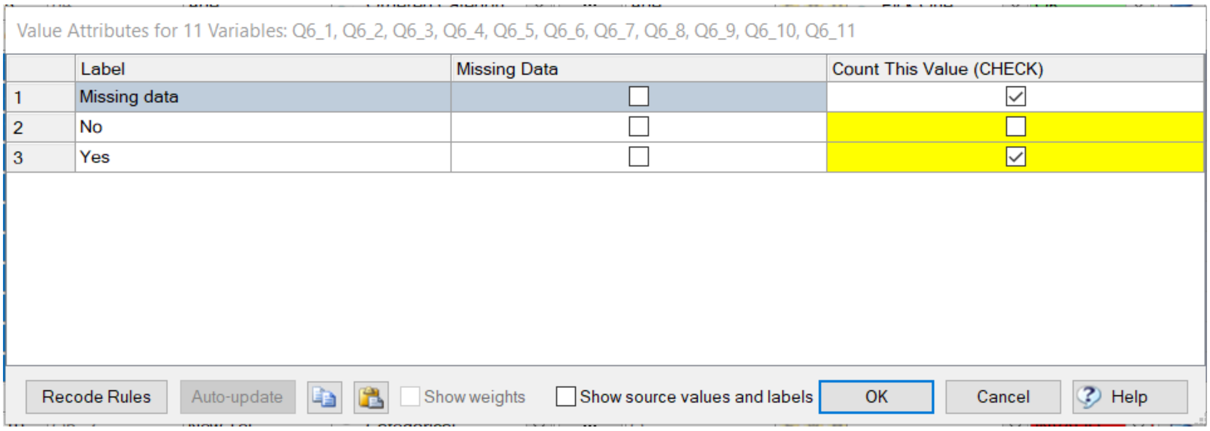
Recoding and rebasing

Recoding and rebasing data in Q is done by selecting one or more variables in the **Data Sets tree** (bottom-left of the screen), and pressing the … button in the middle of the screen. This opens the *Value Attributes* dialog box, where the user can change values and specify **Missing Values**. Ranges of missing values can also be changed by clicking the **Recode Rules** button, and, programmatically by creating new variables and using code.



Count this Value

With **Pick Any** and **Pick Any – Grid** questions, the **Value** column is replaced by a **Count this Value** column. This is useful both as a way of creating top two box scores, and as a way of dealing with filtered options in a questionnaire. For example, if a question asks people which brands they consume, but has missing values for people known to consume the brand (e.g., known to be current customers, so not asked the question), we can thus recode this this appropriately by ensuring that the Missing data row is not checked as **Missing data** but is checked as **Count this Value**.



Merging

Categories are merged by creating a table and then clicking on the row or column headings and dragging and dropping. Additional options are available by right-clicking and choosing from the context menu.

Q will reapply decisions about how to merge categories and change their values whenever the underlying data is used again. That is, a change made on one table will also update all other tables using the same question. To avoid this, first duplicate the question (right-click and select **Duplicate Question**, and then change and use this new version of the data.

Changing Question Type

Q automatically groups together variables into variable sets, which it refers to as *questions,* when the data is imported. In particularly messy data sets, such as the phone data set, the classification of variables into variable sets can be inaccurate, and questions are manually created by going to the **Variables and Questions**, being set as follows

- If variables have been incorrectly grouped, select the question, right-click on any of its row numbers and select **Split Variables from Question.**
- To combine variables into a question, select them, right-click and select **Set Question.**
- To change a question's **Question** Type, change the **Question Type** dropdown on the right of the screen. See https://wiki.q-researchsoftware.com/wiki/Question_Types for an overview of the different types.

Fixing metadata

The easiest win in terms of fixing metadata is often **Obtaining a high-quality data file**.

When data is imported into Q, common HTML tags are automatically removed from labels.

Q also has a powerful search and replace function. To search, first type into the **Search features and data** box at the top of the screen, and then select **Advanced Find/Replace**. Two things that make it particularly powerful for cleaning metadata are:

- You can customize how the search uses, including:
  - Controlling where the search occurs
  - Using wildcards
  - Ignoring punctuation
  - Capitalization
- A panel appears showing you all the search results so you can check that nothing has been found or changed inadvertently.

Creating filter variables

See https://wiki.q-researchsoftware.com/wiki/Category:Filters for a description of the main ways of creating filters in Q.

Creating a filter variable based on ID

Sometimes you know the ID values of respondents that you wish to delete. If so, you create a filter variable as follows:

- **Create > Variables and Questions > Variable > R Variable > Numeric Variable**
- Paste in the code `ID %in% c(123:125, 311)` replacing `ID` with the name of your ID variable and updating the ID's to be deleted. In the code example here, IDs 311 and from 123 to 125 are included in the filter and run it and give it a name.
- Check the yellow F button to the right of the newly-created variable.

Creating a filter variable based on case number

Sometimes you may want to delete by observation number (also known as case number or respondent numbers). It is generally a bad idea to do this, as such deletion rules don't work when you revise data files, as often the observation numbers will be inconsistent across data sets.

The solution to this is to instead identify some other unique variable and use it is as a substitute. For example, identify the values of the ID variable or some text variables, and instead use these when creating the filter.

If you do want to delete data using case number, it is important to do this with great care, as the position of observations in the data set will change depending on what else is done. For example, if the 72nd and only 72nd respondent has already been deleted, then `ID[72]` will return the value of the 73rd rather than 72nd respondent.

Deleting observations

Once you have created a filter, go to the **Data tab,** select the filter, and then right-click on a row to choose from the options for deleting observations (cases).

Unit tests

There are a variety of ways of setting up unit tests in Q. The most straightforward is to

1. Create a folder called `Unit tests`
2. Create each unit test in the folder by clicking **Create > R Output** and entering a snippet of code that performs the unit test.

When a unit test is failed, this will cause the folder and the specific R Output with the error to go read.

Please refer to the previous chapter on Displayr for detailed examples of creating unit tests (the code is identical in Displayr to in Q).

Data cleaning in R

> This chapter describes the process of performing data cleaning in R.

Checking the sample size

In R, the sample size is checked using the NROW function and the number of variables by NCOL. Where `phone` is the name of your data frame:

```
> NROW(phone)
[1] 725
> NCOL(phone)
[1] 283
```

Checking screening criteria

Sankey diagrams

Use the following code and select the variables that are used in the screeners and a variable showing the status of the respondent (i.e., whether they completed the survey or not).

```
SankeyDiagram(phone[, c("q1", "q2", "q3", "q4", "q5", "Q5_1")])
```

Crosstabs

R primarily supports crosstabs between two variables (it doesn't natively support variable sets, such as multiple response questions). They are created using the `xtabs` function. When data cleaning, it is useful to use `addNA = TRUE` to show the missing values. For example:

```
xtabs(~q3 + q2, data = phone, addNA = TRUE)
```

Recoding, rebasing, Count this Value, and merging

In R, a missing value is typically denoted by `NA`, although in some more exotic circumstances it may appear as `NaN`, `NULL`, `Inf`, or `-Inf` (note that `0/Inf = 0/-Inf = 0`, whereas `0/NA = NA`).

Recoding in R is done by writing code. For example, if q5 is a variable in a data frame called `phone`, and we want to set all the "Don't Know" values to missing, we would write:

```
phone$q5[phone$q5 == "Don't Know"] <- NA
```

R does not have the concepts of Count this Value and merging. While the same outcome can be achieved via recoding, be careful when create multiple copies of the variables.

A more powerful, albeit more complicated, approach to recoding when you must set multiple values at the same time is to use the `case_when` function in the `dplyr` package.

Checking the data quality for each question and variable

Although R does not have any standard tools for automatically checking the quality of survey data, it has a variety of functions for general-purpose data checking.

When applied to a data frame, `str` provides an overview of the class of variables as well as other key information.

```
> str(phone)
'data.frame':	725 obs. of  283 variables:
 $ id      : num  853 854 855 851 852 883 884 885 881 882 ...
 $ q1      : Factor w/ 4 levels "Yes","No","3",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ q2      : Factor w/ 7 levels "Student","Home maker",..: 6 6 3 6 1 6 6 3 1 6 ...
 $ q3      : Factor w/ 10 levels "-9","0","manager/administrator",..: 5 7 NA 3 NA 5 7 NA NA 3 ...
 $ q4      : Factor w/ 10 levels "15 and under",..: 7 3 7 4 3 7 3 7 2 3 ...
 $ q5      : Factor w/ 12 levels "AAPT/Cellular One",..: 4 4 4 4 4 6 8 4 4 6 ...
 $ Q5_1    : Factor w/ 2 levels "No","Yes": 1 1 1 2 1 1 1 1 1 1 ...
 $ Q5_2    : Factor w/ 2 levels "No","Yes": 1 1 1 2 1 1 1 1 1 1 ...
 $ Q5_3    : Factor w/ 2 levels "No","Yes": 1 1 2 2 1 1 2 1 1 1 ...
 $ Q5_4    : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 2 2 2 2 ...
 $ Q5_5    : Factor w/ 2 levels "No","Yes": 1 2 1 2 2 1 2 1 1 1 ...
```

The `psych` package's `describe` function is useful for checking base sizes and the range of variables.

```
> psych::describe(phone)
           vars   n        mean              sd median trimmed    mad min       max
range  skew kurtosis      se
id         1 710 1409497.99 16738585.65  788.0  816.39 600.45  21 200029169 200029148 11.77  136.61 628187.93
```

```
q1*      2 725    1.02    0.22  1.0   1.00  0.00  1     4     3 11.21  135.27   0.01
q2*      3 700    3.84    2.21  5.0   3.92  1.48  1     6     5 -0.28   -1.75   0.08
q3*      4 336    5.55    1.83  5.0   5.52  2.97  1     9     8  0.10   -0.92   0.10
q4*      5 719    4.73    2.05  4.0   4.68  1.48  1     9     8  0.31   -1.41   0.08
q5*      6 712    5.71    1.65  6.0   5.65  2.97  1    12    11  0.28   -1.02   0.06
Q5_1*    7 725    1.08    0.27  1.0   1.00  0.00  1     2     1  3.12    7.77   0.01
Q5_2*    8 725    1.02    0.15  1.0   1.00  0.00  1     2     1  6.29   37.56   0.01
```

The `Hmisc` package's `describe` function is useful for creating summary tables that look for unusual values, small categories, and poor metadata.

```
> Hmisc::describe(phone[, 1:5])
phone[, 1:5]

 5  Variables      725  Observations
-----------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------
id
       n  missing distinct     Info     Mean      Gmd      .05      .10      .25      .50      .75      .90      .95
     710       15      647        1  1409498  2802013    106.0    171.9    413.2    788.0   1232.8   1572.1   1671.5

Value      0e+00 2e+08
Frequency    705     5
Proportion 0.993 0.007
-----------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------
q1
       n  missing distinct
     725        0        4

Value        Yes    No     3     5
Frequency    715     6     1     3
Proportion 0.986 0.008 0.001 0.004
-----------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------
q2
       n  missing distinct
     700       25        6

Value                Student        Home maker           Retired       Not working Part-time worker  Fulltime worker
Frequency                211                62                29                20                84              294
Proportion             0.301             0.089             0.041             0.029             0.120            0.420
-----------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------
q3
       n  missing distinct
     336      389        9

Value                                     -9                            0        manager/administrator                 profession
al        Associate professional      Tradesperson and related
Frequency                                  2                            5                           33
85                             48                            33
Proportion                             0.006                        0.015                        0.098                        0.2
53                          0.143                        0.098

Value      clerical, sales and services  production/transport worker  Labourer and related worker
Frequency                            85                           23                           22
Proportion                        0.253                        0.068                        0.065
```

```
------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------
q4
       n  missing distinct
     719        6        9

Value       15 and under    16-19 yrs    20-24 yrs    25-29 yrs    30-34 yrs    35-44 yrs    45-54 yrs    55-64 yrs  65 and o
ver
Frequency              1           73          230          106           25           35          200           36
13
Proportion         0.001        0.102        0.320        0.147        0.035        0.049        0.278        0.050        0.
018
--------------------------------------------------
```

Histograms are created using `plot(hist(phone$q25))`.

In the case of variable sets, this can often be addressed with a few lines of code. For example, a summary table of multiple response data (i.e., *tick all that apply*), can be created using:

```
> nms <- names(phone)
> tb = apply(phone[, match("Q5_1", nms):match("Q5_11", nms)], 2, FUN = table)
> rbind(tb, Total = colSums(tb))
      Q5_1 Q5_2 Q5_3 Q5_4 Q5_5 Q5_6 Q5_7 Q5_8 Q5_9 Q5_10 Q5_11
No     668  708  552   84  415  122  549  162  681   721   722
Yes     57   17  173  641  310  603  176  563   44     4     3
Total  725  725  725  725  725  725  725  725  725   725   725
```

Histograms to check numeric variables

Histograms are created in R using the `hist` function. The number of breaks is controlled via `breaks`. For example:

```
hist(phone$q25, breaks = 200, freq = FALSE)
```

Missing data patterns

Viewing, filtering, and sorting raw data

Viewing, filtering, and sorting, is achieved by subscripting. For example:

```
> phone[phone$q1 != "Yes", c("id", "q1", "q2", "q3", "q4", "q5")]
      id q1   q2   q3        q4       q5
40   795  5 <NA> <NA> 20-24 yrs Vodafone
123 1673  5 <NA> <NA> 45-54 yrs     <NA>
124 1674 No <NA> <NA> 45-54 yrs     <NA>
```

```
142 1542 No <NA> <NA>       <NA>      <NA>
159 1634 No <NA> <NA>       <NA>      <NA>
594   64 No <NA> <NA>       <NA>      <NA>
603 1673  5 <NA> <NA> 45-54 yrs     Optus
604 1674 No <NA> <NA> 45-54 yrs     Optus
643  503 No <NA> <NA>       <NA>      <NA>
665   NA  3 <NA> <NA> 45-54 yrs     Optus
```

Missing values by case visualization

```
library(Amelia)
missmap(phone[, c("id", "q1", "q2", "q3", "q4", "q5")])
```

Missing data patterns visualization

```
library(VIM)
aggr(phone[,  c("id", "q1", "q2", "q3", "q4", "q5")])
```

Fixing metadata

The easiest win in terms of fixing metadata is often **Obtaining a high-quality data file**.

There are no inbuilt functions in R designed specifically for fixing metadata. The main tools that need to be used to efficiently fix the metadata involve writing `for` loops and using regular expressions via the `gsub` function.

Checking data for duplicates

The following code creates a table, where each row number shows the ID variable that is duplicated, and the number to the right shows how many times it appeared. Replace `id` with whatever variable you want to use.

```
id.variable = `IID - Interviewer Identification`
dupes = duplicated(id.variable)
dupe.values = id.variable[id.variable %in% unique(id.variable[dupes])]
sort(table(dupe.values, useNA = "always"), decreasing = TRUE)
```

If we want to use multiple variables to defined duplicates, we just modify the first line as shown below:

```
id.variable = paste(id, age, gender, occupation)
```

To create a variable that marks which values are duplicates use:

```
duplicated(`IID - Interviewer Identification`)
```

To create a new data frame that removes the duplicates (leaving just their first appearance), use the following code, replacing df with the name of your data frame and id with the variable or variables you want to identify duplicates for (comma separated).

```
dplyr::distinct(df, id, .keep_all = TRUE)
```

Creating filter variables and deleting cases

Any variable in R can be used as a filter variable by subscripting and, in many functions, via the `subset` parameter.

For example, to create a new data frame with no duplicate OD variables, you can use:

```
new.phone <- phone[!duplicated(id), ]
```

Unit tests

There are a variety of ways of setting up unit tests in R. The simplest is to use the `stop` function.

Checking for duplicate IDs

Where `my.id` is the name of your id variable, the following code will test for the existence of any duplicates.

```
if(any(duplicated(id)))
    stop("Duplicated id variables") else "No duplicate IDs"
```

Missing values in a variable

The following unit test checks to see if there are any missing values in q1.

```
if(any(is.na(q1)))
    stop("Missing values in q1") else "No missing values in q1"
```

Out of range values

The following code checks for any values other than `Yes` or `No` in q1, where:

- `unique` returns a vector of the unique values in `q1`
- `unique(q1) %in% c("Yes", "No")` returns vector containing of the same length as the vector returned by unique, where a `TRUE` is shown if the unique value is either `Yes` or `No` and a `FALSE` is shown otherwise
- The `!` before unique means *not* and has the effect of reversing the `TRUE` and `FALSE` values

```
if(any(!unique(q1) %in% c("Yes", "No")))
    stop("Out of range in Q1") else "Out of range values in Q1"
```

This next example looks for out of range values for numeric variables:

```
max.q25 = max(q25, na.rm = TRUE)
min.q25 = min(q25, na.rm = TRUE)
if(max.q25 > 50)
    stop("The highest value in q25 is ", max.q25)
if(min.q25 < 0)
    stop("The lowest value in q25 is ", max.q25) else "Q25 values are in
the expected range"
```

Data cleaning in SPSS

> This chapter describes the process of performing data cleaning in SPSS Statistics.

Checking the sample size

Scroll to the bottom of the **Data View** to see the number of cases, and the bottom of the **Variable View** to see the number of variables. Do not rely on the number of cases shown at the bottom of the screen (this often shows 100, regardless of the data file size).

Checking screening criteria

Sankey diagram

SPSS does not have a Sankey diagram functionality. The options are to either use the R Integration and R code or create crosstabs.

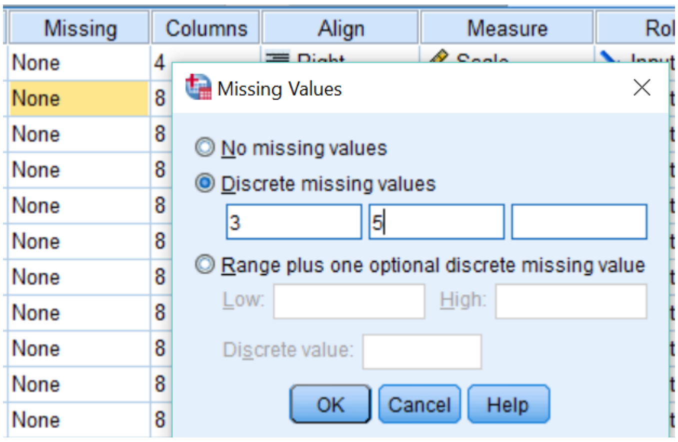
Crosstabs

Analyze > Summary Statistics > Crosstab

Recoding, rebasing, Count this Value, and merging

Recoding is performed in SPSS using **Transform > Recode into Same Variables** and **Recode into Different Variables**.

Additionally, any value can be set as missing (thereby re-basing any analyses using it) by clicking the **Missing** cell in the **Variable View**. This opens the **Missing Values** dialog box, where the user can specify which values or ranges of values should be treated as missing. This is shown below.



Count this Value is set using the *multiple response sets features.* However, these are not supported by most of the crosstabbing features in SPSS, so it is more common to use Custom Tables.

Merging is not supported in SPSS, but the same outcome can be achieved via recoding. Take care to have multiple versions of variables for each different analysis (e.g., one version for computing means and a separate version for computing percentages).

Checking the data quality for each question and variable

Unlike with Q and Displayr, SPSS has no tools for automatically checking data, so it is necessary to either write scripts to automate the process (typically using Python) or to create tables of all the data and inspect them.

- To create summary frequencies, use **Analyze > Summary Statistics > Summary**
- To compute means and tables of missing values, use **Insert > More (Analysis) > Tables > Descriptive Statistics** and select the **Variables** of interest (by dragging from the **Data** tree.

Type and **Measure** information is stored in the **Variable View**.

	Name	Type	Width	Decimals	Label	Values	Missing	Columns	Align	Measure	Role
1	id	Numeric	4	0	IID - Interviewer ...	None	None	4	Right	Scale	Input
2	q1	Numeric	8	0	Does responde...	{1, Yes}...	None	8	Right	Nominal	Input
3	q2	Numeric	8	0	Work status	{1, Student}...	None	8	Right	Nominal	Input
4	q3	Numeric	8	0	Occupation	{1, manager...	None	8	Right	Nominal	Input
5	q4	Numeric	4	0	Age	{1, 15 and u...	None	8	Right	Ordinal	Input
6	q5	Numeric	4	0	Top of mind aw...	{1, AAPT/C...	None	8	Right	Nominal	Input
7	Q5_1	Numeric	1	0	Unaided aware...	{0, No}...	None	8	Right	Nominal	Input

Histograms to check numeric variables

1. **Graphs > Legacy Dialogs > Histogram**
2. Select the variable of interest in the **Variables** box and press **OK**.
3. Click on the chart and then on the *x-axis.*
4. In the **Properties** form, click on the **Binning** tab, specify **X Axis** as **Custom**.
5. Specify the **Number of intervals** as 99 (the maximum) and press **OK.** (I get a bug when I do this, but hopefully that is just on my machine!)

Missing data patterns

Viewing, filtering, and sorting raw data

The raw data can be viewed in the **Data View** in SPSS. Sorting is done by right-clicking on columns and selecting sort. Filtering is problematic in SPSS. Either it does not change the **Data View** or it permanently deletes the data. For this reason, filtering should generally be avoided in SPSS when viewing raw data. Most people tend to find it easier to instead copy and paste the data into Excel and export it in Excel.

Missing values by case visualization

Using the R integration:

```
library(Amelia)
missmap(phone[, c("id", "q1", "q2", "q3", "q4", "q5")])
```

Missing data patterns visualization SPSS

If you have access to the SPSS Missing Values module, it produces a table which contains the same information, as follows:

Tabulated Patterns

Number of Cases	Missing Patterns[a] id	q1	q2	q3	q4	Complete if ...[b]
321						321
364				X		685
5	X			X		698
8	X					329
1	X		X	X		715
16			X	X		701
4			X	X	X	707
2				X	X	687
3		X	X	X		704
1	X	X	X	X		719

- **Analyze > Missing Value Analysis**
- Select the variables of interest as **Quantitative Variables** or **Categorical Variables**.
- Change **Maximum Categories** to 999999.
- Press **Patterns**.
- Check the option **Tabulated cases, grouped by missing value patterns**.
- Change **Omit patterns with less than** to 0 (unless you have a good reason to expect missing values).
- Uncheck **Sort variables by missing value pattern**
- Press **Continue**.
- Press **OK**.

Fixing metadata

Metadata is fixed in SPSS by writing syntax (in particular, the VALUES and VARIABLE LABEL commands.

Checking data for duplicates

To identify duplicates using a single variable, use **Data > Identify Duplicate Cases**. You can also delete causes using the **Select If condition is satisfied**.

Creating filters and deleting observations

Filters are created and observations deleted using **Data > Select** if. Please see https://youtu.be/ero4VR7h1HU.

Unlike with Displayr, Q, and R, there is no way to re-obtain deleted observations once they have been deleted, so good practice is to regularly change the name of the data file (e.g., `phone.sav` -> `phone1.sav` -> `phone2.sav`).

Unit tests

Although it is possible to write unit tests in SPSS Syntax, it is not recommended because the way that the output viewer works makes it easy to miss errors. Consequently, it is recommended to perform unit tests using the R integration (please see the previous chapter for more information).

Appendix

The data used in this book

This book uses a case study of the mobile phone market to illustrate most of the key concepts. This case study is a particularly messy data file.

The data set is an SPSS data file (see **SPSS Data Files**), which is the main file format in use in commercial survey research, called `Phone.sav`.

- It is available for download at `https://wiki.q-researchsoftware.com/images/3/30/Phone.sav`
- The questionnaire is here: `https://wiki.q-researchsoftware.com/images/3/33/Phone_Questionnaire.pdf`

Beginner's mistakes for people new to R

The following are some common mistakes that people make when learning to use R, whether using the R program itself, or from within SPSS, Q, or Displayr.

- Often you will type `=` when you should type `==`. The first assigns a value. The second checks if two things are equal.
- Case is important: `dog` and `Dog` are different objects. They are as different as `dog` and `Giraffe`, as far as R is concerned.
- If something is surrounded by "quotes" it is interpreted as a string (or, to use the proper jargon it has a class of type *character*). If something is not in quotes it is an *object.* Thus, `"dog"` and `dog` are entirely different things.
- We can usually use single and double quotations interchangeably, so long as we use them in pairs. Thus, `"dog" == 'dog'` whereas `"dog' == 'dog"` is interpreted as a single string containing `dog' == 'dog`. Using a combination of single and double quotations is particularly useful when writing text strings using code.
- Many of the characters that you use in normal programs cause an error in R. In the next point:

    - Spaces cut and paste from some programs do not always work. So, if cutting and pasting code from a book or a website, sometimes you will get a weird error that is fixed by re-typing it.
    - Only ugly quotation and apostrophes work in R. Thus, “ does not work, whereas " does work.
    - It is necessary to “escape” special characters. For example, if you wanted to create a string creating a special quotation mark, `"""` would not work, but `"\""` would work, where the \ tells R that we are using the quotation mark as a character rather than as a way of signaling the end of a string. Or, we could type '"' (a pair of double quotes with a single in the middle), which R will interpret as being the same thing as `"\""`.
- Sometimes functions fail silently or close-to-silently. For example, if you inadvertently use the wrong variable names in the `validate` package, you will not get an error. The consequences of mistakes are obvious if you know where to look but can be hard to spot if you are just cutting and pasting code that you do not understand well.

The limitations of R for survey analysis

Survey analysis is one general area of data science where R is limited. Compared to other traditional stats programs, such as SPSS and Stata, little commercial survey analysis is conducted in R. This is because R is missing concepts that are important in survey research. In particular:

- R does not have a concept of a *variable label*. For example, in each of SPSS, Displayr, and Q, a variable called Q12 may have a label associated with it, such as "What is your age?".
- R's concept of a `factor` can cause problems with survey analysis. Factors are assigned consecutive integers as their values, which is often less than ideal. In particular:
    - Often it is useful to assign a non-sequential integer to represent categories. For example, in SPSS, Q, and Displayr, you can, say, assign a value of 0 to represent the category of "Never visited", 1 to "Once a week", etc., whereas this cannot be done in R, without a lot of hacking.
    - R does not support permit a `matrix` to be a `factor`, which makes manipulation of variable sets containing factors difficult.
    - When importing multiple related factors, the same factors can have different values assigned to them for equivalent levels if any of them have categories that were not selected by respondents. For example, *Strongly Agree* may be a 4 in some variables and a *5* in others.
- The limitations of the factor concept in R means that it is often preferable to represent categorical variables in R as being text variables, which creates its own set of problems (e.g., ordering, assignment of values to categories).

- R does not have a concept of *variable sets* (i.e., groups of related variables). Such concepts are useful for representing multiple response questions and for creating tables from such data. In the absence of such tools with R, it is necessary to instead create hundreds or even thousands of tables using individual variables and splice them together.
- R does not have the ability to correctly read SPSS .SAV files, which are the closest there is to an industry standard in survey research. Due to the issues identified in the previous points, these files are generally mangled when they are imported into R.
- R produces very basic looking tables, and cannot, without a lot of work and the use of other programs (e.g., LaTeX), produce the types of tables and statistical tests in widespread use in market research (e.g., a table with multiple categorical variables in the columns and letters denoting statistical significance between subsets of the columns).

Except for the formatting of tables, an experienced and skilled R programmer can readily solve these issues, and there are multiple packages that address parts of these issues. However, doing so takes considerable time. For example, while it is straightforward to put variable label information as an attribute in a variable, you also need to rewrite all functions that manipulate variables (e.g., `+`, `as.numeric`, etc.) to ensure that this information is not lost whenever you manipulate data. All of this is to say, conducting the rigorous analysis of surveys in R ends up requiring a lot of coding to reinvent tools that already come in products designed for survey analysis, such as SPSS, Q, and R.

Data, data files, data file formats, databases, and data sets

Data

Data is a generic term which has a meaning that is defined by context only. Sometimes it refers to databases, sometimes to data file, sometimes to variables, sometimes to cases, etc.

Data file

A *data file* is a file that contains data and is located on a storage medium of some kind (e.g., a hard disk). Data files typically have *extensions* that allow people to determine what is in the file. For example, files containing comma-separate variables, have the extension of `.csv`.

Data file format

There are lots of possible ways that data can be formatted in files (i.e., *file formats*).

Database

A database is software that contains data and has been organized in a manner that permits access users to add, manipulate, and extract data.

When data is extracted from a database it is usually extracted as a data file.

Data set

A *data set* is data that is in a data analysis program (e.g., Q, R, SPSS, or Displayr), which can be queried by the user. In some programs the data sets are databases. In some programs the data sets are data files. In some programs the data sets exist in memory. Many programs support multiple types of data sets. From the perspective of the user, this distinction is usually inconsequential.

Data sets

Often analysis requires multiple data sets. For example, one data set may represent the characteristics of individual transactions, and a separate data set may represent the characteristics of people that made those transactions (where one person may have made multiple transactions – these are called *hierarchical data sets)*. Some software packages can only work with a single data set at a time, whereas others can integrate more than one into the analysis.

Observations

Almost all data science and statistical software assume that data sets are organized, or can be viewed as being organized, in a two-dimensional table, where the rows represent the *observations* (also known as cases) and columns represent *variables*. For example, the *data table* below shows 20 observations from a study of the telecommunications market, where observation (row) is a separate company that uses telecommunications and each column (variable).

ID	Industry	Shop	Understand	Key	Interest	Value	Profit ($)	# Employees
1	Retail Trade	A	A	A	A	D	9777.47	12
2	Retail Trade	A	A	A	D	A	3595.79	12
3	Cult. and Rec. Services	A	A	A	A	D	2660.15	20
4	Retail Trade	A	A	D	A	A	2303.08	30
5	Manufacturing	A	D	A	D	D	644.57	6

6	Mining	D	A	A	A	D	3517.85	99
7	Agr., Forest. & Fishing	A	D	A	D	D	6905.25	8
8	Retail Trade	D	D	A	A	D	9916.39	60
9	Health & Community Services	A	A	A	A	A	1855.43	56
10	Property & Business Services	A	A	A	A	D	765.10	4
11	Communication Services	D	A	D	D	A	838.13	1
12	Manufacturing	A	A	A	A	A	2303.08	30
13	Manufacturing	D	D	D	D	D	2151.92	7
14	Manufacturing	A	A	A	A	D	1263.65	1
15	Agr., Forest. & Fishing	D	D	D	A	A	394.87	2
16	Education	D	D	D	D	A	72196.56	50
17	Personal and Other Services	D	D	A	D	D	90.71	2
18	Personal and Other Services	D	A	D	A	D	171.77	2
19	Retail Trade	D	D	D	D	A	189.14	2
20	Education	A	A	A	A	D	255.38	10

Variables

Variables represent the characteristics of the observations. At a simple level, a variable can be thought of as being the columns in table of raw data (see the previous section).

A variable consists of:

- A set of *values*, where there is one value for each unit of analysis (e.g., for the **# Employees** variable: 12, 12, 20, 30, 6, 99, 8, 60, 56, 4, 1, 30, 7, 1, 2, 50, 2, 2, 2, 10).
- *Metadata* which describes the meaning of the variable.

Metadata

Metadata is all the data that describes the data in a data set. It is useful to distinguish between the *data set metadata*, *variable metadata*, and *variable set metadata.*

Data set metadata

The data set metadata is all the information required to understand the data set. For example, it will typically include information such as:

- Where the data set came from.
- How the data was created (e.g., survey administration instructions).
- Specific instructions for creating the data set (e.g., the questionnaire, programming instructions, SQL queries).
- Which variable(s) should be used as *unique identifiers* (keys) when joining the data set to other information. This is required for:
    - Data checking purposes. For example, when trying to work out why a value in the data appears to be "wrong".
    - Updating purposes. For example, once the data has been cleaned, if a revised data file is obtained you will typically want to apply the same cleaning decisions as before to any data that is unchanged. If you do not have a unique identifier, there is no way to do this.
    - *Joining purposes.* For example, to join two related data sets.
- When the data set was created.

Variable metadata

A variable's *metadata* consists of its:

- *Name*. Most programs have rules regarding what can be in a variable name, such as that it starts with a letter, and contains no spaces or unusual symbols. Typically, the variable names are short to aid typing and reading of code (e.g., `q8a1`).
- *Type*. This is information about how the variable should be stored and processed, and usually includes one or more of the following concepts:
    - The *data type* used to store the values, such as whether the data is a string, logical (Boolean), integer, floating-point number, or some other data type (e.g., in R, a matrix, vector, etc.).
    - The *scale* properties of the data, which governs how it should be analyzed. Whether it is nominal (unordered), ordinal (ordered), or numeric.

- *Label***.** This is the description of the meaning of the variable. For example: `How strongly do you agree with the statement 'Data Science is Cool'?`
- *Comments***.** This is additional information that helps people interpret the variable. For example, it may contain the variable's *change log* (i.e., changes in how the variable has been cleaned or modified over time) or known issues.
- *Formatting.* Instructions regarding how the data is to appear when viewed, such as the number of decimals, the width of the columns, and currency symbols.
- *Expression*. The code or formula used to construct the variable (e.g., `q2 + Q3 * 4`).
- *Value attributes* (also commonly known as the *code frame*). These consists of:
    - Any *value labels* associated with specific values (e.g., a 1 may denote `Male` and 2 `Female`).
    - *Missing value codes,* which indicate that specific values should be treated as missing. Modern data analysis programs have specific values for missing values and non-finite values (e.g., `NA`, `NaN`, `NA`, `Inf`, `-Inf`), whereas older products may designate values that are deemed unlikely to occur as missing values (e.g., -99, 99, 997, 998, 999). Sometimes different missing value codes exist for different reasons. For example, 997 may denote that no attempt was made to collect a specific piece of data, 998 that somebody refused to provide the data, and 999 that somebody was unsure of the correct value).
- *Data Reduction.* This is the set of rules that govern how specific values are to be combined when performing some analyses. For example, the values of 18, 19, 20, 21, 22, 23, and 24, may all be grouped into a new combined category with a label of 18 to 24, the values from 18 to 49 may also be included in a second overlapping category, and so on. As the term "data reduction" is not widely used, there is no standard terminology for this idea.

Some of the metadata for the earlier data table is shown in the table below.

ID	Each organization has one value on this variable and no other organizations have the same value.
Industry	The industry classification of the firm.
Shop	Agree (A) or disagree (D) that "It is important to shop around"
Understand	Agree (A) or disagree (D) that "I understand my company's communication needs"
Key	Agree (A) or disagree (D) that "Communications technology is key to our business"
Interested	Agree (A) or disagree (D) that "I am interested in communications technology"
Value	Agree (A) or disagree (D) that "Value for money is more important to us than getting the best technology"
Profit ($)	An estimate of the gross profit provided by each firm to the industry (excluding fixed costs). Constructed from a series of survey questions about the types of products held, usage levels and bill payments.

# Employees	Number of employees of the business

Variable set metadata

A *variable set* is a group of one or more structurally related variables (i.e., variables that typically need to be analyzed jointly). Common examples include:

- *Multiple responses sets.* For example, answers to the survey question "Which of these colas have you purchased? Coke, Pepsi, Dr Pepper, Pibb, Pepsi Max, None of These?"
- *Compositional/constant-sum* data. For example, three variables measuring the proportion of phone calls that a person makes to other people on the network in their country, to other people in the network in a foreign country, to people in their country on another network, and to people in another network in another country.
- *Scales.* For example, "On a scale of 1 to 5, how satisfied are you with your bank, internet company, local McDonalds".
- *Sequences and rankings.* For example, "Looking at the products below, please rank them in order of preference, with the one you like most at the top, the second one in second place, etc.".
- *Experimental data.* For example, one variable showing what was selected and another showing what the experimental manipulation was.
- *Value and unit.* For example, one variable containing people's weights, and second recording the unit it is measured in.

Variable set metadata consists of:

- The list of variables in the variable set.
- The *type* of the variable set (e.g., is it a ranking, is it constant-sum data).
- Any other metadata common to the set.

Tidy raw data set

A *tidy raw data set* is a data set that has the following characteristics:

1. The data is structured as a table, with rows and columns. Sometimes this is referred to as a *flat* format.
2. Each row corresponds to an observation.
3. Each column represents a variable (i.e., some property of the unit of analysis).

4. Enough metadata exists to permit analysis of the data.

Tidy data

The term *tidy data* has come into vogue in the R community over the past decade.[6] If you are not familiar with the term, just skip this section, as it is only intended for people that are confused about the difference between tidy data and a tidy raw data set.

A *tidy raw data set* is a special case of *tidy data.* The key differences are that a *tidy raw data set:*

- Is designed with multiple possible analyses in mind, whereas a tidy data set is typically designed with a single analysis in mind. For example, a tidy raw data set is created for an entire survey, whereas a tidy data set may be created for a specific analysis of one of the survey's questions.
- Is typically less aggregated, whereas the tidy data set may have been aggregated for a specific purpose. For example, a tidy data set may contain sales by product by month, whereas the tidy raw data may contain each purchase by each person over time.
- Typically contains many variables. By contrast, tidy data typically contains a subset of the data, where the focus is on a specific analysis or chart.

A few consequences of this are that:

- *Tidy raw data sets* are the data that is queried to make *tidy data*. That is, tidy raw data sets tend to be the focus earlier in the analysis process.
- Tidy raw data sets are always meaningfully filterable.
- Tidy raw data sets may contain weight variables (e.g., for survey analysis).
- Tidy raw data sets typically contain a unique identifier or key variable of some kind (henceforth referred to as the *ID variable*).

---

[6] Hadley Wickham (2014), "Tidy data" in *The Journal of Statistical Software*, 59.

Tidied, cleaned, and organized tidy raw data sets

The end-point of data tidying and cleaning is:

1. a tidy raw data set,
2. where all the values in the data set are accurate, and
3. everything has been structured to make data analysis easy.