MSI data science manual

A review of analytical approaches followed by Management Systems International

2023-09-21

Table of contents

Preface

This manual introduces a variety of analytical approaches MSI adopts in order to learn, teach, and meet its client deliverables.

1 Introduction

We used to do data science in spreadsheets, while a more select realm of demi-gods used expensive statistical analysis packages or wrote code. Now we have access to open source software that puts immense computing power in the hands of the people. The easy accessibility of such power is both a blessing and a curse. This manual seeks to bestow the blessings while avoiding the curse.

This manual is already out of date. Tomorrow, we will be able to tell our AI assistants what we want, and the AI instance will provide it to us through some opaque combination of computation and creation. Aspiring analysts are still encouraged to learn this manual as a way to welcome our new overlords.

2 Summary

First and foremost, MSI is client driven. We provide what is asked for, following client guidance. Within this framework, we use a variety of analytical approaches and tools that follow best practice while satisfying the guidance we are under.

This manual provides an idealized approach of a data analysis. MSI is agnostic about the tools used to conduct an analysis, but the majority of explanation and examples are provided in the R programming language.

In unit one, we introduce R, explain how it works, and provide guidance as to how to get set up and start analyzing.

In unit two, we go through the steps of a data analysis.

In unit three, we review different ways to report your analyses.

Throughout the journey, we will provide examples of MSI analyses that were used for internal learning, or to generate a client deliverable.

Let's start!

Part I

Entering the R ecosystem

3 How R works

Most of us are familiar with Excel or used a software like SPSS, SAS, or STATA in school. Some of us use these regularly at work.

A programming environment, such as R, offers some cool stuff.

- R is open-source and free. It has a huge support community that is constantly de-bugging and creating new functionalities.
- If you find an analysis or cool example, the code is almost always included. The R community is all about sharing.
- R was developed specifically for statistical programming.
- If you can imagine an analytic task, you can implement it in R.
- Analyses in R are transparent, easily shareable, and reproducible. Can you remember every step you did to create a data visualization in Excel so that someone else could add to it?
- R integration with Github allows a team to work together.
- R can read and write in virtually any data format.
- R can be used for any data science task: scraping websites, developing websites, making static or interactive charts, automating repetitive tasks, statistical computations, querying databases, and many others.
- R has a lot of inter-operability with other platforms.

To realize these benefits, however, requires an understanding of how R works. This chapter will walk through some foundations of using R and its data structures

3.1 Basic use

In R, you create objects and then use those objects for your analysis. Objects are made up of whatever you assign them to. They can be a direct value, a file, a graphic, etc. Here's an example:

```
a <- 5
```

We have assigned the object, `a`, the value of 5. The assignment operator `<-` is what tells R to assign the value of 5 to `a`.

Now we can use the object `a`. As in `a + a.` We use the `#` to annotate our code for human readers. R will not compute any text to the right of a `#`. Annotating code is very helpful for code review and for remembering what you were doing when you open up a script that you have not worked on for 6 months.

```
# Assign a the value of 5
a <- 5

# Add a + a (or 5 +5)
a+a
```

```
[1] 10
```

Notice that R understands to output the value of `a+a` without any additional instructions. Or, you could store the value of `a + a` as a new object.

```
a <- 5

# Assign the value of a + a to b
b <- a + a

#print value of b
b
```

```
[1] 10
```

3.2 Data Structures

The primary data structures in R are vectors, matrices, lists, and data frames. They all basically begin as a vector. The idea here is not to master what the data structures are, but to understand how R handles each one as it will affect more advanced coding operations. Knowledge of data structures is also helpful when debugging code because error messages will reference the different data structures.

Naturally, we will start with the most "atomic" of the data structures, the (atomic) vector.

3.2.1 Vectors

A vector is the most basic data structure in R. A vector can only contain a single data type. It can be any of logical, integer, double, character, complex or raw, but it cannot mix and match types.

Here's a vector

```
# Create vectors
vector <- 10
vector1 <- c(10, 14, 27, 99)
vector2 <- c("purple", "blue", "red")

# Print the value of each vector
vector
```

```
[1] 10
```

```
vector1
```

```
[1] 10 14 27 99
```

```
vector2
```

```
[1] "purple" "blue"   "red"
```

3.3 Matrices

A matrix is a vector with dimensions - it has rows and columns. As with a vector, the elements of a matrix must be of the same data type. Here are a few examples.

```
# Create a 2 x 2 matrix with the numbers 1 through 4
m <- matrix(1:4, nrow = 2, ncol = 2)

# Note that the matrix is filled column-wise. (e.g., it completes # the left column with 1 a
# and entering 3 and 4
m
```

```
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

3.4 Lists

A list is a vector that can have multiple data types. You can call `class()` on any object in R to display the type of object that it is.

```
# Make a list a
a <- list(10, "red", 74, "blue")

# What is the class, or type, of a?
class(a)
```

```
[1] "list"
```

3.5 Dataframes

You can think of a dataframe as your Excel Spreadheet. At MSI, this is the most common form of dataset. We read a .xlsx or .csv file into R, and we get a dataframe. At its core, a dataframe is a list of lists where each list (column) is the same length (i.e., it is a "rectangular list"). A data frame can contain many types of data because it is a collection of lists, and lists, as you remember, can consist of multiple data types.

```r
# Create a dataframe called df

df <- data.frame(a = c(10,20,30,40)
                 , b = c('book', 'pen', 'textbook', 'pencil_case')
                 , c = c(TRUE,FALSE,TRUE,FALSE)
                 , d = c(TRUE,FALSE,TRUE,FALSE))

# Print df
df
```

```
   a           b     c     d
1 10        book  TRUE  TRUE
2 20         pen FALSE FALSE
3 30    textbook  TRUE  TRUE
4 40 pencil_case FALSE FALSE
```

Now that we have a dataframe, we want to look at some of its details using `glimpse()`.

```r
# Create a dataframe called df

df <- data.frame(a = c(10,20,30,40)
                 , b = c('book', 'pen', 'textbook', 'pencil_case')
                 , c = c(TRUE,FALSE,TRUE,FALSE)
                 , d = c(TRUE,FALSE,TRUE,FALSE))

# Look at structure of df
dplyr::glimpse(df)
```

```
Rows: 4
Columns: 4
$ a <dbl> 10, 20, 30, 40
$ b <chr> "book", "pen", "textbook", "pencil_case"
$ c <lgl> TRUE, FALSE, TRUE, FALSE
$ d <lgl> TRUE, FALSE, TRUE, FALSE
```

3.6 Factors

Factors are numeric vectors that contain only pre-defined values (categories), and where each of these categories has a label.

```r
a <- sample(1:2, 100, replace=T)
table(a)
```

```
a
 1  2
51 49
```

```r
a_f <- factor(a, labels=c("Male","Female"))
table(a_f)
```

```
a_f
  Male Female
    51     49
```

Note that the labels are just labels, the underlying representation is still 1 and 2.

```r
str(a_f)
```

```
 Factor w/ 2 levels "Male","Female": 1 2 1 1 1 2 2 1 2 2 ...
```

Factors can sometimes cause trouble. More contemporary practice is to stick with an integer data type and add your own labels.

```r
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to beco
```

```r
library(sjmisc)
```

```
Attaching package: 'sjmisc'

The following object is masked from 'package:purrr':

    is_empty

The following object is masked from 'package:tidyr':

    replace_na

The following object is masked from 'package:tibble':

    add_case
```

```
library(sjlabelled)
```

```

Attaching package: 'sjlabelled'

The following object is masked from 'package:forcats':

    as_factor

The following object is masked from 'package:dplyr':

    as_label

The following object is masked from 'package:ggplot2':

    as_label
```

```
a_l <- a %>%
  set_labels(labels=c("Male","Female"))
str(a_l)
```

```
 int [1:100] 1 2 1 1 1 2 2 1 2 2 ...
 - attr(*, "labels")= Named num [1:2] 1 2
  ..- attr(*, "names")= chr [1:2] "Male" "Female"
```

```
frq(a)
```

```
x <integer>
# total N=100 valid N=100 mean=1.49 sd=0.50

Value |  N | Raw % | Valid % | Cum. %
--------------------------------------
    1 | 51 |    51 |      51 |     51
    2 | 49 |    49 |      49 |    100
 <NA> |  0 |     0 |    <NA> |   <NA>
```

```
frq(a_f)
```

```
x <categorical>
# total N=100 valid N=100 mean=1.49 sd=0.50

Value  |  N | Raw % | Valid % | Cum. %
-------------------------------------
Male   | 51 |    51 |      51 |     51
Female | 49 |    49 |      49 |    100
<NA>   |  0 |     0 |    <NA> |   <NA>
```

```
frq(a_l)
```

```
x <integer>
# total N=100 valid N=100 mean=1.49 sd=0.50

Value |  Label |  N | Raw % | Valid % | Cum. %
-----------------------------------------------
    1 |   Male | 51 |    51 |      51 |     51
    2 | Female | 49 |    49 |      49 |    100
 <NA> |   <NA> |  0 |     0 |    <NA> |   <NA>
```

The underlying integers behind factor labels have no ordering. To establish an ordering, make an ordered factor.

```
ord <- sample(1:5, 100, replace=T)
table(ord)
```

```
ord
 1  2  3  4  5
15 25 10 26 24
```

```
ord.labs <- c("Not at all","A little","Somewhat","Much","Completely")
ord.fac <- ordered(ord, labels=ord.labs)
table(ord.fac)
```

```
ord.fac
Not at all   A little   Somewhat       Much Completely
        15         25         10         26         24
```

But again, you have to be careful not to accidentally jumble the underlying integers with the ordered labels.

To keep things more explicit, I would still stick with an integer variable with labels, rather than an ordered factor.

```
ord.l <- ord %>%
  set_labels(labels=ord.labs)
table(ord.l)
```

```
ord.l
 1  2  3  4  5
15 25 10 26 24
```

```
frq(ord.l)
```

```
x <integer>
# total N=100 valid N=100 mean=3.19 sd=1.43

Value |      Label |  N | Raw % | Valid % | Cum. %
-----------------------------------------------------
    1 | Not at all | 15 |    15 |      15 |     15
    2 |   A little | 25 |    25 |      25 |     40
    3 |   Somewhat | 10 |    10 |      10 |     50
    4 |       Much | 26 |    26 |      26 |     76
    5 | Completely | 24 |    24 |      24 |    100
 <NA> |       <NA> |  0 |     0 |    <NA> |   <NA>
```

When you're ready to dive into this sometimes-frustrating subject, start here:

- forcats package in the tidyverse
- working with labelled data
- wrangling categorical data in R

3.7 Sub-setting

You can cut up your objects into other objects. The base R way to do this is to use brackets.

3.7.1 sub-setting vectors

```
a <- rpois(5, 8)
a
```

```
[1]  6  9  9 11  7
```

```
a[1:3]
```

```
[1] 6 9 9
```

```
a[c(1,4,2)]
```

```
[1]  6 11  9
```

```
a[-2]
```

```
[1]  6  9 11  7
```

```
a[a>9]
```

```
[1] 11
```

3.7.2 sub-setting data frames

Since data frames are two dimensional, you usually (but not always) need to specify both dimensions.

```
str(df)
```

```
'data.frame':   4 obs. of  4 variables:
 $ a: num  10 20 30 40
 $ b: chr  "book" "pen" "textbook" "pencil_case"
 $ c: logi  TRUE FALSE TRUE FALSE
 $ d: logi  TRUE FALSE TRUE FALSE
```

```
df_f <- df[1:5,2]
str(df_f)
```

```
 chr [1:5] "book" "pen" "textbook" "pencil_case" NA
```

The comma within the bracket specifies rows and columns.

More contemporary practice uses what is referred to as the tidyverse.

4 How R programs

Part II

Workflow

5 Happy Git with R

6 R and Data Science resources

Part III

Data Analysis

7 Designing an analytical activity

8 Sampling

9 Introduction

10 Simple random sampling

10.1 With replacement

10.2 Without replacement

11 Complex sampling

11.1 Stratified sampling

11.2 Cluster sampling

11.3 Using intraclass-correlation to estimate sample sizes

There is simple random sampling and there is complex sampling. Let's explain how we get to each.

```
n_rng <- (3.4 * 1.96^2 * .25) / (c(.02, .03, .05)^2 * .95)

srs <- c((1.96^2*.25)/.05^2,
         (1.96^2*.25)/.03^2,
          (1.96^2*.25)/.02^2)


out <- data.frame(margin=c("5%","3%", "2%"),
                  srs=srs,
                  complex=round(rev(n_rng), 0),
                  scope=c("Overall sampling area", "One disaggregate", "Two disaggregates, ne
flextable(out) %>%
  set_header_labels(
                    margin="Margin of error",
                    srs="Simple random sample",
                    complex="Complex sample",
                    scope="Geographic scope") %>%
  autofit()
```

Margin of error	Simple random sample	Complex sample	Geographic scope
5%	384	1,375	Overall sampling area
3%	1,067	3,819	One disaggregate

Margin of error	Simple random sample	Complex sample	Geographic scope
2%	2,401	8,593	Two disaggregates, nested

How do we derive each type of sample size?

11.3.1 Simple random sampling

Sample size calculation to achieve a given margin of error starts with assumptions of simple random sampling (SRS). A first statistics course establishes a 95% confidence interval around a sample mean, in which 95 out of 100 instances of a given experiment would contain the fixed population parameter.

$$CI = \bar{x} \pm z\frac{s}{\sqrt{n}}$$

Where z is the z-score for a given confidence level (most commonly 1.96 for a 95% confidence level), s is the sample standard deviation, and n is the sample size. Recognize the expression $\frac{s}{\sqrt{n}}$ as the standard deviation of the sample mean, or standard error.

The above expression generates the confidence interval from a sample of data. If we're interested in planning our experiment to reach a desired benchmark margin of error, than we can drop the sample mean and set the margin of error E to the error calculation.

$$E = z\frac{s}{\sqrt{n}}$$

Knowing that we will set E to our desired benchmark, we rearrange terms to solve for the unknown n.

$$\sqrt{n} = \frac{z * s}{E}$$

$$n = \frac{z^2 s^2}{E^2}$$

Note that we can further simplify this expression if we use a proportion as the outcome of interest. Recall that the variance of a proportion p is $p * (1 - p)$, which we can plug in for the value of s^2. This gives us something more tractable:

$$n = \frac{z^2 p(1-p)}{E^2}$$

Furthermore, note that variance is maximized at $p = .5$, such that $p(1-p) = .5 * .5 = .25$. Maximizing the value of the numerator also maximizes the sample size calculation for any value of p. Therefore, we can assume $p = .5$ so as to calculate the upper bound of the needed sample size. Finally, we'll plug in $z = 1.96$ to set a 95% confidence interval and $E = .05$ as our desired benchmark for margin of error. This lets us solve for n:

$$n = \frac{1.96^2 * .25}{.05^2}$$

Which comes out to 384. This is a common sample size that is quoted for any simple random sample with a margin of error of 5%.

11.3.2 Cluster sampling

A common problem is that the sample size based on simple random sampling is often quoted for the needed sample size for household surveys. But, household surveys typically involve a complex design that includes organizing the sampling frame by strata, and then sampling clusters at multiple stages (stratified, multi-stage, cluster sampling design).

Sampling clusters reduces the geographic scope of data collection, relative to a simple random sample. However, this reduced scope comes at the cost of a degree of similarity within the sampled clusters that reduces the amount of effective information contained within the clusters. A simple random sample assumes statistical independence of each element in the sampling frame, but cluster sampling violates the assumption of independence.

To adjust for the clustered nature of the data collection, we introduce the design effect *deff*. The design effect is the variance inflation factor that occurs as we move from simple random sampling to cluster sampling:

$$Deff_p(\hat{\theta}) = \frac{var(\hat{\theta}_{cluster})}{var(\hat{\theta}_{srswo})}$$

We can enter this adjustment directly into the sample size calculation:

$$n = \frac{deff * z^2 p(1-p)}{E^2}$$

As a final step, we can adjust for non-response by taking the actual response rate in the denominator, which will adjust the sample size upward. This gives us:

$$n = \frac{deff * z^2 p(1-p)}{E^2 r}$$

where r is the response rate, which we'll set at 95%.

We now need to estimate the design effect. We don't know the complex variance in the denominator of the expression for $Deff$, but we can can estimate $Deff$ through use of the intra-class correlation between clusters, ρ:

$$D_{eff} = 1 + (\bar{b} - 1)\rho$$

where $\bar{b}$ is the average cluster sample size.

We don't know the design effect or intra-class correlation for Lebanon, but we can look to other surveys. The Arab Barometer Wave 7 surveys in 2021-2022 included Lebanon, from which we can estimate the intra-class correlation and design effect.

```
rho_out <- samplesize4surveys::ICC(leb$usg, leb$PSU)

rho <- rho_out$ICC

rho # .34
```

```
[1] 0.34
```

Based on the Wave 7 Arab Barometer survey, the cluster sample size averaged 5.1 and ρ =.34. From this we can estimate the design effect from this survey as $1 + ((5.1 - 1) * .34)$.

```
psu_n <- leb %>%
  group_by(PSU) %>%
  tally() %>%
  summarise(n=mean(n)) %>%
  unlist() %>%
  round(1)

psu_n
```

```
  n
5.1
```

```
deff_barom <- 1 + ((psu_n-1)*.34) %>%
  unlist() %>%
  round(1)

deff_barom
```

```
  n
2.4
```

Cluster sampling in the Arab Barometer survey inflated the variance to 2.4 times the variance from a simple random sample. This inflation factor is kept manageable only by the fact that the sample size for each cluster is lower than normal.

We can use this design effect from the Arab Barometer survey to inform our sample size projections for the prospective perception survey. Cluster sample sizes usually range from 8-16.

```
deff_samp <- data.frame(psu_n=c(6,8,12,16,20)) %>%
  mutate(deff= round(1 + (psu_n - 1) * .34, 1))

flextable(deff_samp)
```

psu_n	deff
6	2.7
8	3.4
12	4.7
16	6.1
20	7.5

We'll consider a cluster sample size of 8, with an estimated design effect of 3.4.

This adjustment then enters the sample size expression, using a desired margin of error of three percent.

$$n = \frac{3.4 * 1.96^2(.5 * .5)}{.03^2 * .95}$$

Which gives a sample size of 3,819.

```
n_proj <- (3.4 * 1.96^2 * .25) / (.03^2 * .95)
n_proj
```

```
[1] 3819
```

Using the design effect from the Wave 7 Arab Barometer survey for Lebanon, we need a sample of 3,820 in order to achieve a desired precision of a 3% margin of error.

Let's provide a range of sample sizes, for margins of error of two, three, and five percent.

```
n_rng <- (3.4 * 1.96^2 * .25) / (c(.02, .03, .05)^2 * .95)

srs <- c((1.96^2*.25)/.05^2,
         (1.96^2*.25)/.03^2,
          (1.96^2*.25)/.02^2)


out <- data.frame(margin=c("5%","3%", "2%"),
                  srs=srs,
                  complex=round(rev(n_rng), 0),
                  scope=c("Overall sampling area", "One disaggregate", "Two disaggregates, ne
flextable(out) %>%
  set_header_labels(
                    margin="Margin of error",
                    srs="Simple random sample",
                    complex="Complex sample",
                    scope="Geographic scope") %>%
  autofit()
```

Margin of error	Simple random sample	Complex sample	Geographic scope
5%	384	1,375	Overall sampling area
3%	1,067	3,819	One disaggregate
2%	2,401	8,593	Two disaggregates, nested

MSI generally recommends a margin of error for a household survey of no less than five percent, which would require a sample size of 1,375 after incorporating the information about the clustering effect from the Wave 7 Arab Barometer Survey. To reach a desired margin of error of three percent, we need a sample of 3,819. To reach a desired margin of error of two percent, we need a sample of 8,593.

11.3.3 Conclusion

The use of the intra-class correlation and design effect from the Arab Barometer survey suggests that villages in Lebanon tend to be more homogeneous than what is typically found in other

countries. This higher level of homogeneity within villages requires the sampling of additional villages in order to reach a desired level of precision. MSI advises a precision of no less than five percent. This would require a sample of 1,375 and would be representative of the overall sampling area, but it may not be possible to meaningfully examined across any demographics of interest. A more desirable level of precision would be a margin of error of three percent, which would require a sample of 3,819. This would enable precise estimates at the overall level, as well as exploration of findings across a single disaggregate such as urban/rural locality, sex, age group, or education. If USAID would like to have the ability to meaningfully describe differences across more than one demographic (for example, sex within urban or rural locality), then MSI recommends an overall margin of error of two percent, which would require a sample size of 8, 593.

12 Data cleaning and preparation

12.1 Instrument Design and Scripting

12.1.1 Instrument Design and Scripting Workflows

In the process of designing and scripting instruments for data collection, it is crucial to follow a systematic workflow that ensures accuracy, completeness, and ease of updating. Before going through this section it is important to consider several overarching points:

- Client preferences will likely dictate in what form instruments are shared.
- Spending the time to develop a thorough data dictionary from the onset is worth it!
- Involve both HO and FO activity managers in instrument pre-testing whenever possible.

Below, we outline two workflows: the "Typical" Instrument Design and Scripting Workflow, and the "Proposed" Instrument Design and Scripting Workflow. The proposed workflow aims to address and improve upon the shortcomings observed in the typical workflow.

"Typical" Instrument Design and Scripting Workflow The typical workflow consists of five main stages:

IP/Client Document Review and Consultations:

- This initial stage involves reviewing relevant documents provided by the implementing partners (IPs) or clients and conducting consultations to understand the requirements and context.

Instrument Development in Word:

- Based on the gathered information, instruments are developed in Word documents. This step is critical but often leads to instruments that are difficult to update and manage, especially regarding data names and skip logic.

Client/IP Feedback:

- The draft instruments are then shared with the client or IP for feedback. This stage is essential for ensuring the instrument meets the client's needs and expectations.

Update Instruments in Word:

- Following feedback, the instruments are updated in Word. This iterative process can be time-consuming and prone to errors, particularly in managing complex logic and data structures. Script Instruments from Word into Data Collection Software:
- Finally, the instruments are scripted into the data collection software. This step often reveals issues not apparent in the Word documents, such as missing data names and ineffective skip logic, making the scripting process cumbersome. Typically, the client version of the instruments lacks unique data names, effective skip logic, and is challenging to update, leading to inefficiencies and potential errors in the data collection process.

Proposed Instrument Design and Scripting Workflow To overcome the limitations of the typical workflow, the proposed workflow introduces a more structured and efficient approach.

IP/Client Document Review and Consultations:

- As in the typical workflow, this stage involves a thorough review of documents and consultations to gather requirements.

Data Dictionary Development:

- Instead of directly developing instruments in Word, this stage focuses on creating a comprehensive data dictionary. The data dictionary includes unique data names, skip logic, and translations, providing a clear and structured framework for the instruments.

Client/IP Feedback:

- The data dictionary is shared with the client or IP for feedback, ensuring that all necessary elements are captured and agreed upon before developing the instruments. Update Data Dictionary:

Based on the feedback, the data dictionary is updated. This step ensures that any changes or additions are systematically incorporated, maintaining the integrity and structure of the data. Script Instruments into Data Collection Software:

- With a well-defined data dictionary, scripting the instruments into the data collection software becomes more straightforward and less error-prone. The unique data names and effective skip logic defined in the data dictionary ensure a smooth transition from design to implementation.

By adopting the proposed workflow, the instrument design and scripting process becomes more efficient and reliable. The use of a data dictionary ensures that all elements are clearly defined and managed, reducing the likelihood of errors and making the instruments easier to update and maintain. This approach not only improves the quality of the instruments but also enhances the overall efficiency of the data collection process.

12.2 Common Scripting Issues

13 Reproducibility

14 Data exploration

15 Introduction

In a well-defined study, exploratory analysis may be largely unnecessary. Consider a scenario where the client has identified questions / hypotheses of interest and the additional variables that may predict or mediate the identified outcomes. An activity partner such as MSI was provided the time and resources to consult with stakeholders, scope out the target environment to identify and map the data generating process, and develop an inferential design by which the collected data and analytical routines would address the questions posed by the client. In this scenario, exploratory analysis may be entirely eliminated, and the time that may be spent on exploration is shifted to sensitivity analysis of the pre-identified analytical routines.

On the other hand, consider a scenario where data has been collected for one purpose, and later realized to have possible use with other, not yet identified, purposes. In this case, the initial analysis is entirely exploratory.

16 Data visualization

16.1 Data visualization tactics

16.1.1 geomtextpath

There are common situations where MSI is tasked with ongoing data collection and evaluation of a client activity. For example, the MENA MELS activity (2020-2024) was tasked with ongoing monitoring and evaluation of the Middle East Partnership for Peace Activity (MEPPA). MEPPA comprised grants to several local partners organized around the common motif of building bonds between different demographic groups. The primary outcome of interest was whether or not a participant in the grant activity reported a perceived increase in understanding the political, social, and economic situation and viewpoints of another group. Data were collected on a rolling basis across several implementing partners, across baseline and endline. That data are summarized as follows:

```
df1 <- read_csv("data/short demo series/meppa response items.csv",
                show_col_types=F)
df1 %>%
  flextable() %>%
  autofit()
```

item	response	lab	endline	n	perc
Political views	1	Basic understanding	0	22	0.286
Political views	2	Fair understanding	0	38	0.494
Political views	3	High understanding	0	17	0.221
Political views	1	Basic understanding	1	18	0.173
Political views	2	Fair understanding	1	50	0.481
Political views	3	High understanding	1	36	0.346
Social views	1	Basic understanding	0	30	0.390
Social views	2	Fair understanding	0	25	0.325

item	response	lab	endline	n	perc
Social views	3	High understanding	0	22	0.286
Social views	1	Basic understanding	1	15	0.147
Social views	2	Fair understanding	1	52	0.510
Social views	3	High understanding	1	35	0.343
Economic views	1	Basic understanding	0	32	0.416
Economic views	2	Fair understanding	0	30	0.390
Economic views	3	High understanding	0	15	0.195
Economic views	1	Basic understanding	1	20	0.196
Economic views	2	Fair understanding	1	55	0.539
Economic views	3	High understanding	1	27	0.265

For purposes of client reporting, there are three ordinal responses across baseline and endline, for each of three types of viewpoint. There is insufficient data to conduct inferential tests at this level of granularity, but this data may be visualized descriptively.

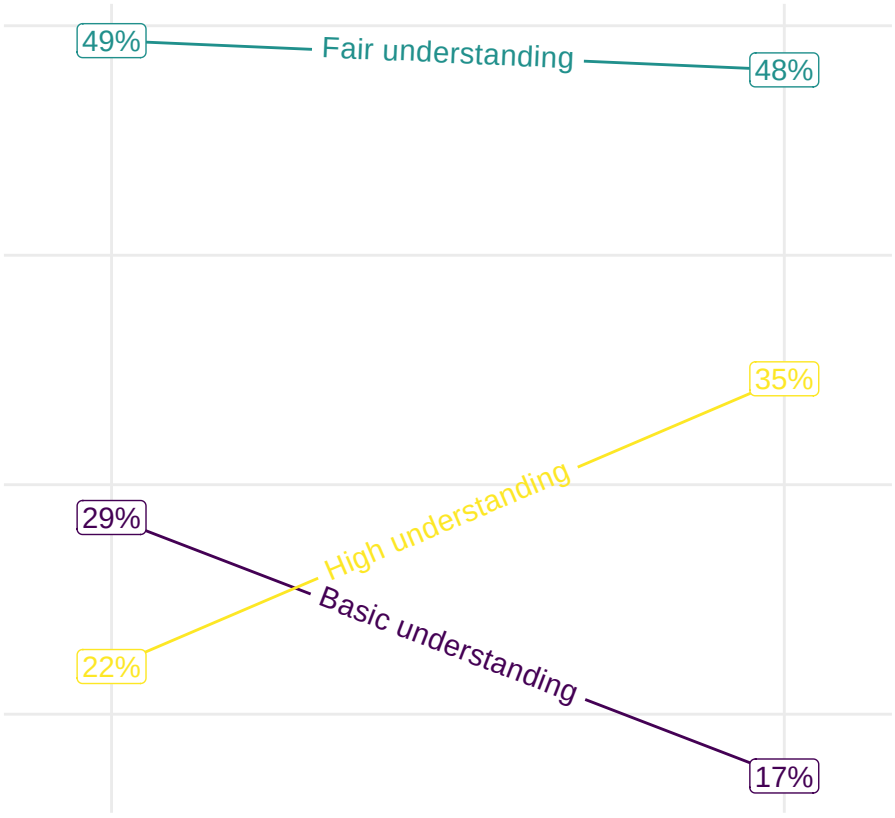
The geomtextpath package offers the functionality to directly label line-based plots with text that is able to follow a curved path. Simply replace 'geom_line' with 'geom_textpath' and assign the variable to use as the label. The following figures illustrate.

```
pol <- ggplot(filter(df1, item=="Political views"), aes(endline, perc, color=as.factor(respo
  geom_textpath(aes(label=lab),
                size=4) +
  geom_label(aes(label=paste(round(perc*100,0), "%", sep="")),
             size=4,
             label.padding = unit(.14, "lines")) +
  scale_color_viridis_d(option="D") +
  scale_x_continuous(limits=c(-.1, 1.1),
                     breaks=c(0,1),
                     labels=c("Baseline","Endline")) +
  scale_y_continuous(labels=percent_format(accuracy=1)) +
  theme(legend.position="none",
        axis.text.y=element_blank()) +
  labs(x="",
       y="",
       title="Political situation")
```

```
pol
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
width unknown for character 0x20
```

```
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
font width unknown for character 0x20
```



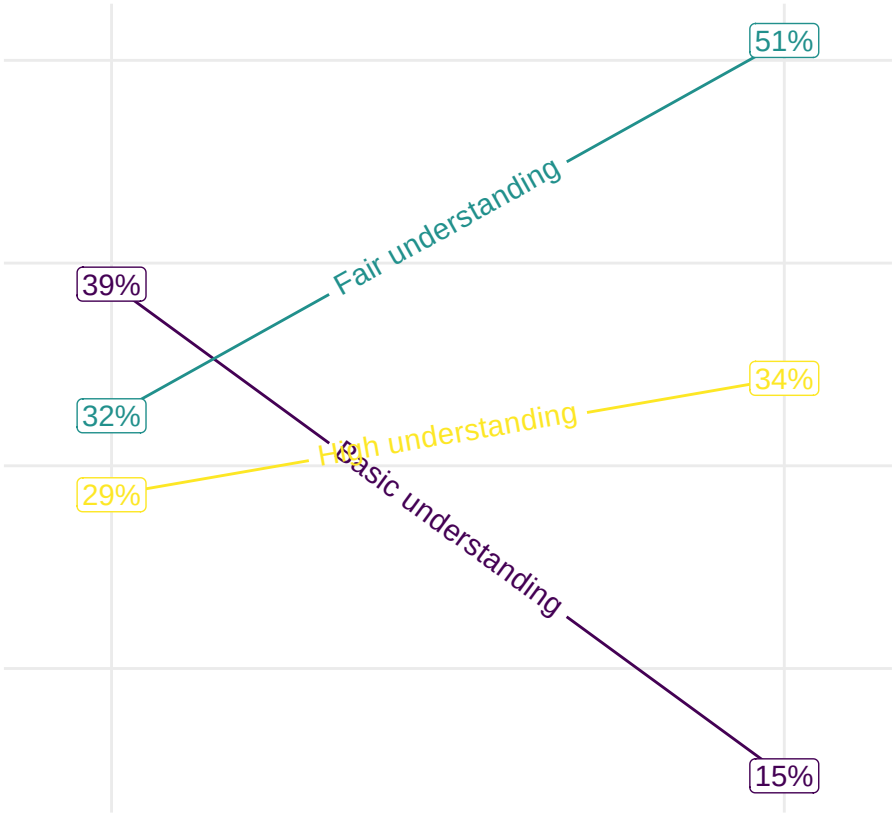

```
soc <- ggplot(filter(df1, item=="Social views"), aes(endline, perc, color=as.factor(response)
  geom_textpath(aes(label=lab),
                size=4) +
  geom_label(aes(label=paste(round(perc*100,0), "%", sep="")),
             size=4,
             label.padding = unit(.14, "lines")) +
  scale_color_viridis_d(option="D") +
  scale_x_continuous(limits=c(-.1, 1.1),
                     breaks=c(0,1),
```

```
                      labels=c("Baseline","Endline")) +
  scale_y_continuous(labels=percent_format(accuracy=1)) +
  theme(legend.position="none",
        axis.text.y=element_blank()) +
  labs(x="",
       y="",
       title="Social situation")

soc
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
width unknown for character 0x20
```

```
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
font width unknown for character 0x20
```

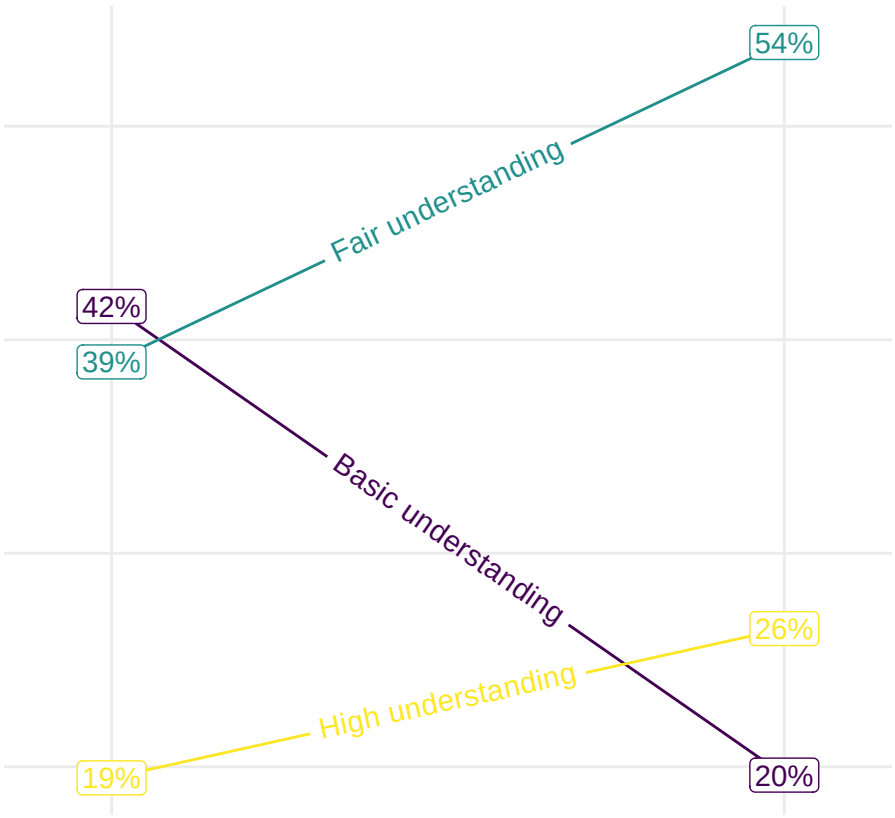


```
ec <- ggplot(filter(df1, item=="Economic views"), aes(endline, perc, color=as.factor(response
  geom_textpath(aes(label=lab),
                size=4) +
  geom_label(aes(label=paste(round(perc*100,0), "%", sep="")),
             size=4,
             label.padding = unit(.14, "lines")) +
  scale_color_viridis_d(option="D") +
  scale_x_continuous(limits=c(-.1, 1.1),
                     breaks=c(0,1),
                     labels=c("Baseline","Endline")) +
  scale_y_continuous(labels=percent_format(accuracy=1)) +
  theme(legend.position="none",
        axis.text.y=element_blank()) +
  labs(x="",
       y="",
       title="Economic situation")

ec
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
width unknown for character 0x20
```

```
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
font width unknown for character 0x20
```



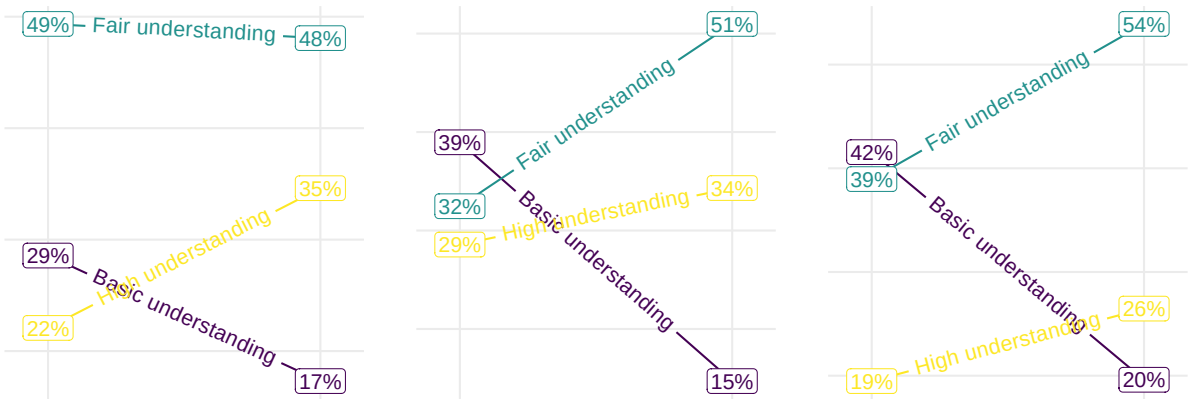
Given that the three types of understanding of others' situation are highly correlated, it makes sense to present these measures compactly as aspects of a deeper underlying construct. The patchwork library allows multiple ggplots to be assembled together as a single plot. The following figure illustrates.

```
pol + soc + ec +
  plot_annotation(title="How well do you understand the situation of others?")
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
width unknown for character 0x20
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
width unknown for character 0x20
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
width unknown for character 0x20
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
width unknown for character 0x20
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
width unknown for character 0x20
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
```

```
width unknown for character 0x20
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
width unknown for character 0x20
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
width unknown for character 0x20
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
width unknown for character 0x20
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
width unknown for character 0x20
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
width unknown for character 0x20

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
font width unknown for character 0x20
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
font width unknown for character 0x20
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
font width unknown for character 0x20
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
font width unknown for character 0x20
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
font width unknown for character 0x20
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
font width unknown for character 0x20
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
font width unknown for character 0x20
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
font width unknown for character 0x20
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
font width unknown for character 0x20
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
font width unknown for character 0x20
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
font width unknown for character 0x20
```



A final use of presenting the data more compactly is to collapse the ordinal responses to binary, and collect the three measures as lines in a single plot. The following data captures each type of understanding as either fair or high understanding as one category, and basic understanding as the other category.

```
dat <- read_csv("data/short demo series/meppa item ladder.csv",
                show_col_types=F)

dat_flx <- dat %>%
  flextable() %>%
  autofit()

dat_flx
```

endline	n	perc	item
0	55	0.714	Political situation
1	86	0.827	Political situation
0	47	0.610	Social situation
1	87	0.853	Social situation
0	45	0.584	Economic situation
1	82	0.804	Economic situation

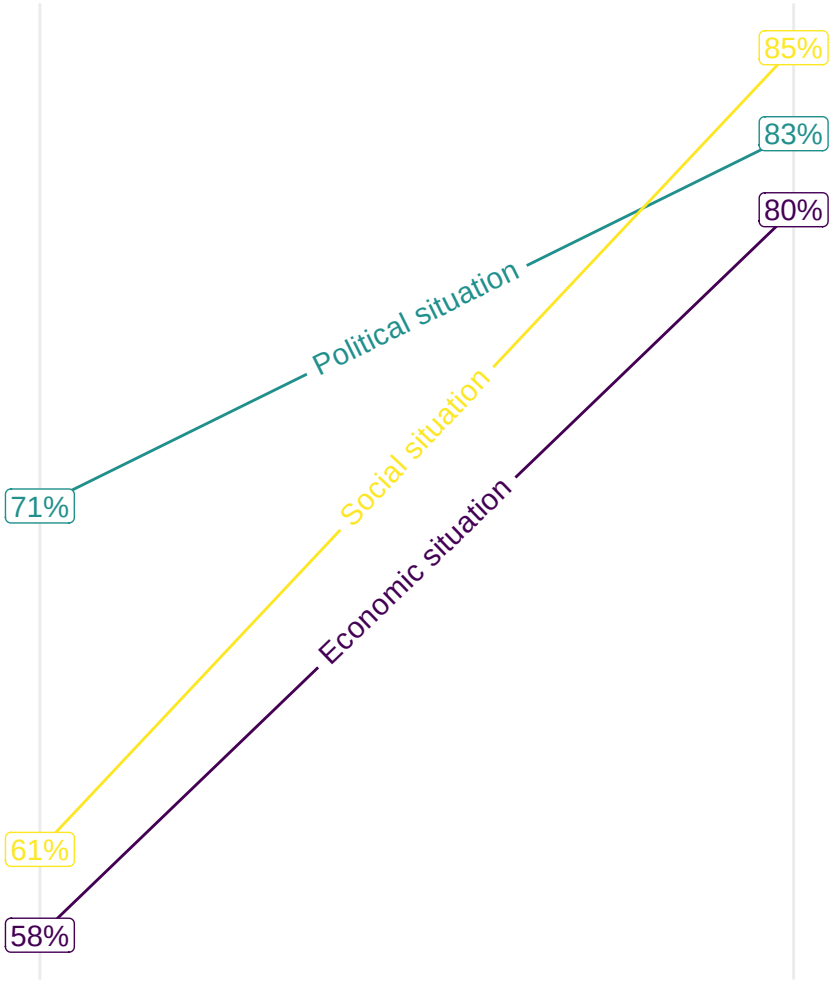
With this simplified data summary, the trendline for each type of understanding may now be collected in a single plot.

```r
ggplot(dat, aes(endline, perc, color=as.factor(item))) +
  geom_textpath(aes(label=item),
                size=4) +
  geom_label(aes(label=paste(round(perc*100,0), "%", sep="")),
             size=4,
             label.padding = unit(.14, "lines")) +
  scale_color_viridis_d(option="D") +
  scale_x_continuous(limits=c(-.1, 1.1),
                     breaks=c(0,1),
                     labels=c("Baseline","Endline")) +
  scale_y_continuous(labels=percent_format(accuracy=1),
                     breaks=c(.5,1),
                     sec.axis=dup_axis(breaks=c(.804,.827,.853),
                                       labels=c("+22","+12","+24"))) +
  theme(legend.position="none",
        axis.text.y.left=element_blank()) +
  labs(x="",
       y="",
       caption="Proportion reporting fair or high\nunderstanding of others' situation")
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
width unknown for character 0x20
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
width unknown for character 0x20
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
width unknown for character 0x20
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
width unknown for character 0x20
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
width unknown for character 0x20
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
width unknown for character 0x20
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
width unknown for character 0x20
```

```
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
font width unknown for character 0x20
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
font width unknown for character 0x20
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
font width unknown for character 0x20
```

```
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
font width unknown for character 0x20
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
font width unknown for character 0x20
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
font width unknown for character 0x20
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
font width unknown for character 0x20
```



Note further the use of secondary axis breaks to illustrate the change score for each trendline from baseline to endline.

The R computing language allows for several ways to customize the use of labels in statistical or descriptive graphics. This short demo has illustrated MSI's use of the geomtextpath package to place labels directly along the line or curve of a plot. This illustration used only straight

lines between two points in time. For additional use cases of the geomtextpath package, see the package vignette.

17 Mapping

Much of our data is collected from surveys and has geographic coordinates associated with a point of collection, a house, or a city, etc. It can be useful to generate a map to get a sense of where the data is coming from. Mapping is a part of MSI's exploratory data analysis process and is also used in developing a sampling plan and in producing high quality data vizualizations for clients. This section provides a brief introduction into mapping in R.

The most commonly used packages to handle spatial data are `sf` for vectors, `terra` for vectors and rasters, and `raster` for rasters. To visualize the data, two frequently used packages are tmap and ggplot2 packages.

To get started, we need to load our packages.

```
#run this line first if you have never used these packages before
#install.packages(c("tidyverse", "sf", "tmap", "readr", "here"))

library(tidyverse) #install the core tidyverse packages including ggplot2
library(sf) #provides tools to work with vector data
library(tmap) #for visualizing spatial data
library(readr) #functions for reading external datasets
library(here) #to better locate files not in working directory
library(geodata) #to download administrative boundaries
```

17.1 Read in the data

```
#It is a csv file so I use the read_csv function and provide the file path
cities <- read_csv(here::here("../methods corner/Map demo/data/Madagascar_Cities.csv")
                   , show_col_types = FALSE)

#Observe the first few rows of data
DT::datatable(head(cities))
```

```
Google Chrome was not found. Try setting the `CHROMOTE_CHROME` environment variable to the ex
```

Show 10 entries Search:

	Name	Population	Region	Province	Capital	Latitude	Longitude	Number of Activities
1	Ambatondrazaka	47649	Alaotra-Mangoro	Toamasina	Adm2	-17.8237084	48.4267188	11
2	Ambositra	41078	Amoron'i Mania	Fianarantsoa	Adm2	-20.5328149	47.2450521	39
3	Ambovombe-Androy	114230	Androy	Toliara	Adm2	-25.1781	46.0892	32
4	Antananarivo	1275207	Analamanga	Antananarivo	Adm0	-18.9100122	47.5255809	54
5	Antsirabe	245592	Vakinankaratra	Antananarivo	Adm2	-19.871403	47.034667	46
6	Antsiranana	131165	Diana	Antsiranana	Adm1	-12.2841	49.2943	29

Showing 1 to 6 of 6 entries Previous 1 Next

Now, for the administrative boundaries. Each of these are being read in using `gadm()` from the `geodata` package and then converted to an sf object with the `st_as_sf()` function. In the example, we download only the country boundary, but if we wanted regions or departments, we would simply change the level argument inside the `gadm()` function call to a 1 or a 2.

17.2 Country boundary

```
#This is only the country boundary.
mdg <- geodata::gadm(country = "MDG"
                     , level = 0
                     , path = tempdir()) |>
  st_as_sf()
```

17.3 Convert the cities to an sf object

Remember that the cities object is a standard .csv with longitude and latitude columns, but it is not yet recognized as an sf object that has geographic properties. Here is how to convert it to an sf object with a single geometry column and a crs.

```
cities_sf <- cities |>
  st_as_sf(coords = c("Longitude", "Latitude")
           , crs = 4326)

#observe the first few rows of data
DT::datatable(head(cities_sf))
```

Show 10 entries Search:

	Name	Population	Region	Province	Capital	Number of Activities	geometry
1	Ambatondrazaka	47649	Alaotra-Mangoro	Toamasina	Adm2	11	[object Object]
2	Ambositra	41078	Amoron'i Mania	Fianarantsoa	Adm2	39	[object Object]
3	Ambovombe-Androy	114230	Androy	Toliara	Adm2	32	[object Object]
4	Antananarivo	1275207	Analamanga	Antananarivo	Adm0	54	[object Object]
5	Antsirabe	245592	Vakinankaratra	Antananarivo	Adm2	46	[object Object]
6	Antsiranana	131165	Diana	Antsiranana	Adm1	29	[object Object]

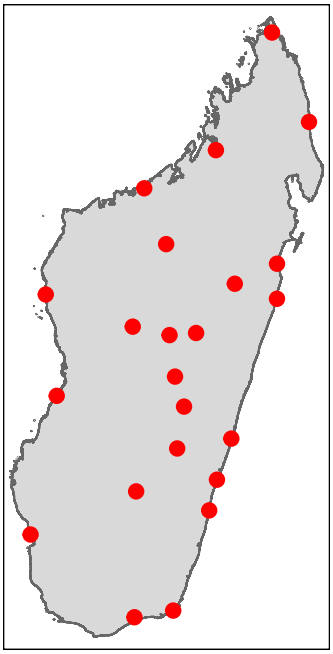
Showing 1 to 6 of 6 entries Previous 1 Next

17.4 Make the map

The following code chunks and tabs walk through the process of making and improving a map in both tmap and ggplot2. In the example, cities are what we are plotting, but we could be plotting any variable of a dataset.

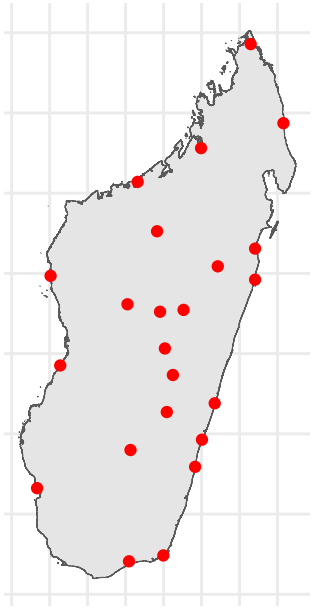
17.5 tmap

```
tmap_mode("plot") +
  tm_shape(mdg) +
  tm_polygons() + #for only the borders, use tm_borders()
  tm_shape(cities_sf) +
  tm_dots(size = .25, col = "red")
```



17.6 ggplot2

```
ggplot2::ggplot(mdg) +
  geom_sf() +
  geom_sf(data = cities_sf, color = "red")
```



17.7 Make the map better

17.8 tmap

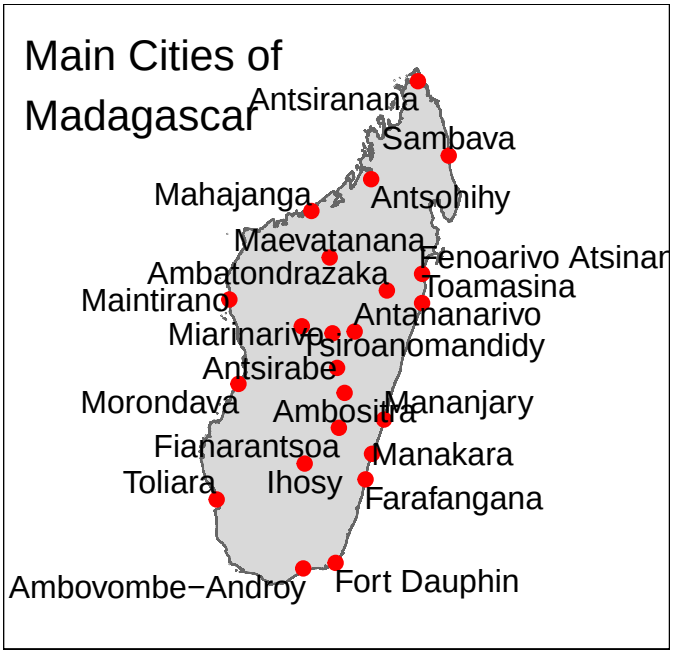
```
#the city names are long so we have to
# make a bigger window to fit them. This isn't part of the normal process
#make an object with the current bounding box
bbox_new <- st_bbox(mdg)

#calculate the x and y ranges of the bbox
xrange <- bbox_new$xmax - bbox_new$xmin # range of x values
yrange <- bbox_new$ymax - bbox_new$ymin # range of y values

#provide the new values for the 4 corners of the bbox
  bbox_new[1] <- bbox_new[1] - (0.7 * xrange) # xmin - left
  bbox_new[3] <- bbox_new[3] + (0.75 * xrange) # xmax - right
  bbox_new[2] <- bbox_new[2] - (0.1 * yrange) # ymin - bottom
  bbox_new[4] <- bbox_new[4] + (0.1 * yrange) # ymax - top

#convert the bbox to a sf collection (sfc)
bbox_new <- bbox_new |>  # take the bounding box ...
  st_as_sfc() # ... and make it a sf polygon
```

```r
#now plot the map
tmap_mode("plot") +
  tm_shape(mdg, bbox = bbox_new) +
  tm_polygons() +
  tm_shape(cities_sf) +
  tm_dots(size = .25, col = "red") +
  tm_text(text = "Name", auto.placement = T) +
  tm_layout(title = "Main Cities of\nMadagascar")
```



17.9 ggplot2

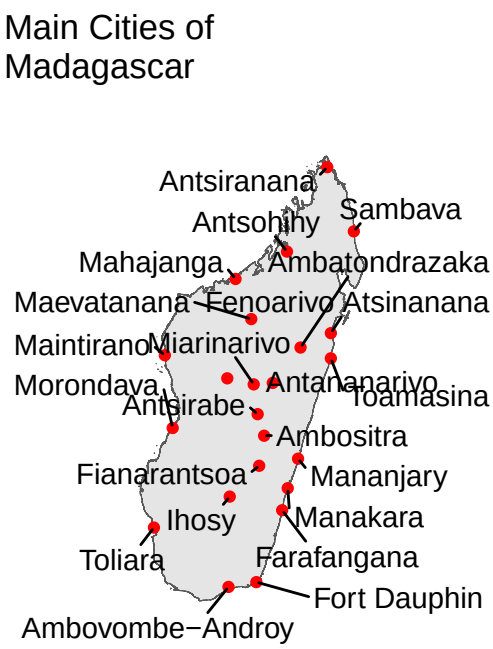
```r
#the city names are long so we have to
# make a bigger window to fit them. This isn't part of the normal process
#make an object with the current bounding box
bbox_new <- st_bbox(mdg)

#calculate the x and y ranges of the bbox
xrange <- bbox_new$xmax - bbox_new$xmin # range of x values
yrange <- bbox_new$ymax - bbox_new$ymin # range of y values
```

```r
#provide the new values for the 4 corners of the bbox
  bbox_new[1] <- bbox_new[1] - (0.5 * xrange) # xmin - left
  bbox_new[3] <- bbox_new[3] + (0.5 * xrange) # xmax - right
  bbox_new[2] <- bbox_new[2] - (0.1 * yrange) # ymin - bottom
  bbox_new[4] <- bbox_new[4] + (0.1 * yrange) # ymax - top

#convert the bbox to a sf collection (sfc)
bbox_new <- bbox_new |>  # take the bounding box
  st_as_sfc() # ... and make it a sf polygon


ggplot2::ggplot() +
  geom_sf(data = mdg) +
  geom_sf(data = cities_sf, color = "red") +
  ggrepel::geom_text_repel(data = cities_sf
                , aes(label = Name
                      , geometry = geometry)
                , stat = "sf_coordinates"
                , min.segment.length = 0) +
  coord_sf(xlim = st_coordinates(bbox_new)[c(1,2),1], # min & max of x values
           ylim = st_coordinates(bbox_new)[c(2,3),2]) + # min & max of y values +
  labs(title = "Main Cities of\nMadagascar") +
  theme_void()
```



17.10 Final touches

Now that we have a map with cities plotted (we achieved our goal!), we will add a few finishing touches and set the size of the city points to the `population` variable in the original dataset.

Additionally, tmap provides a simple interface to go from a static map to an interative map simply by changing `tmap_mode("plot")` to `tmap_mode("view")`.

17.11 tmap

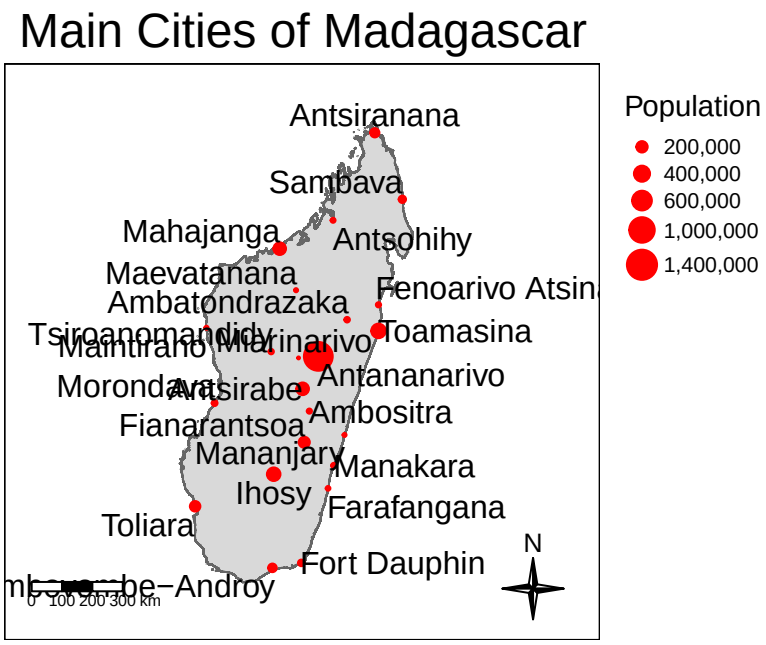
```
#the city names are long so we have to
# make a bigger window to fit them. This isn't part of the normal process
#make an object with the current bounding box
bbox_new <- st_bbox(mdg)

#calculate the x and y ranges of the bbox
xrange <- bbox_new$xmax - bbox_new$xmin # range of x values
yrange <- bbox_new$ymax - bbox_new$ymin # range of y values

#provide the new values for the 4 corners of the bbox
```

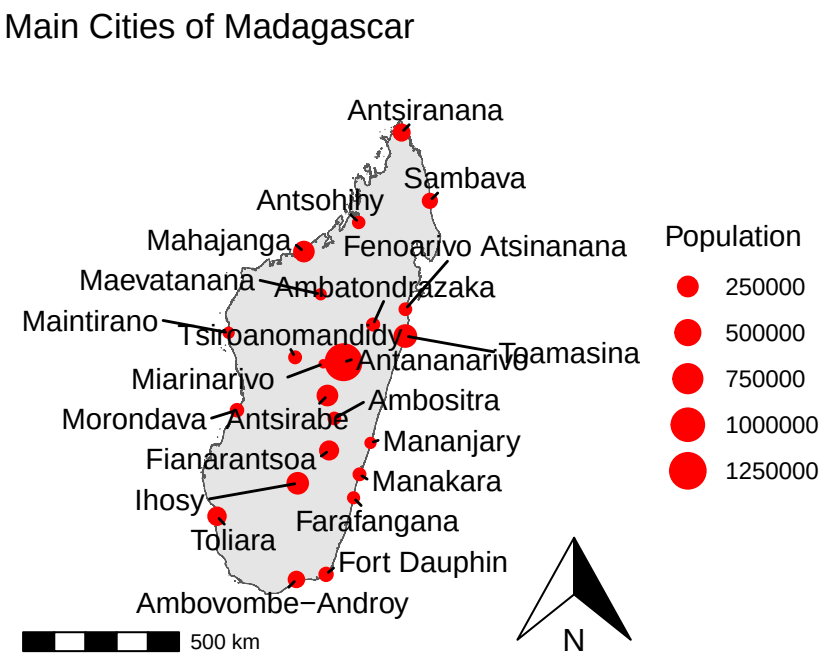
```
  bbox_new[1] <- bbox_new[1] - (0.7 * xrange) # xmin - left
  bbox_new[3] <- bbox_new[3] + (0.75 * xrange) # xmax - right
  bbox_new[2] <- bbox_new[2] - (0.1 * yrange) # ymin - bottom
  bbox_new[4] <- bbox_new[4] + (0.1 * yrange) # ymax - top

#convert the bbox to a sf collection (sfc)
bbox_new <- bbox_new |>  # take the bounding box ...
  st_as_sfc() # ... and make it a sf polygon
tmap_mode("plot") +
  tm_shape(mdg, bbox = bbox_new) +
  tm_polygons() +
  tm_shape(cities_sf) +
  tm_dots(size = "Population", col = "red"
          , legend.size.is.portrait = TRUE) +
  tm_text(text = "Name", auto.placement = T
          , along.lines = T) +
  tm_scale_bar(position = c("left", "bottom"), width = 0.15) +
  tm_compass(type = "4star"
             , position = c("right", "bottom")
             , size = 2) +
  tm_layout(main.title = "Main Cities of Madagascar"                    , legend.outside
```



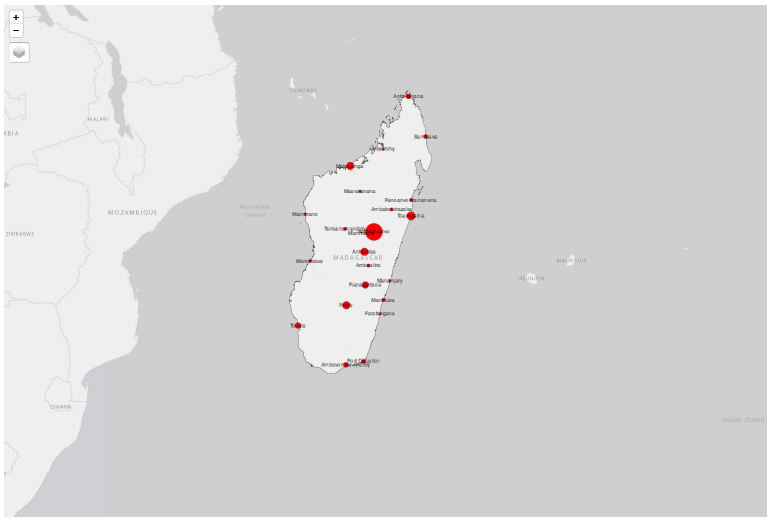
17.12 ggplot2

```
ggplot2::ggplot() +
  geom_sf(data = mdg) +
  geom_sf(data = cities_sf, aes(size = Population)
          , color = "red") +
  ggrepel::geom_text_repel(data = cities_sf
                , aes(label = Name
                      , geometry = geometry)
                , stat = "sf_coordinates"
                , min.segment.length = 0) +
  coord_sf(xlim = st_coordinates(bbox_new)[c(1,2),1], # min & max of x values
           ylim = st_coordinates(bbox_new)[c(2,3),2]) + # min & max of y values +
  ggspatial::annotation_scale(location = "bl") +
  ggspatial::annotation_north_arrow(location = "br"
                                    , which_north = "true"
                                    , size = 1)+
  labs(title = "Main Cities of Madagascar") +
  theme_void()
```



17.13 tmap interactive

```
tmap_mode("view") +
  tm_shape(mdg) +
  tm_borders() +
  tm_shape(cities_sf) +
  tm_dots(size = "Population", col = "red"
          , legend.size.is.portrait = TRUE) +
  tm_text(text = "Name", auto.placement = T
          , along.lines = T) +
  tm_scale_bar(position = c("left", "bottom"), width = 0.15) +
  tm_compass(type = "4star"
             , position = c("right", "bottom")
             , size = 2) +
  tm_layout(main.title = "Main Cities of Madagascar"                          , legend.outside
```



17.14 Additional Resources

For those interested in mapping in R (or QGIS) there are many free resources available online. A great starting point for R is the online text book, Geocomputation with R. If you would rather learn more in Python, Geocomputation with Python is a great resource.

18 Data modeling

19 Generating client deliverables

Part IV

Reporting

20 Reports

21 Presentations

22 Dashboards

Part V

Annexes