A Minimal Book Example

Yihui Xie

2019-10-10

# Examples

This is a *sample* book written in **Markdown**. You can use anything that Pandoc’s Markdown supports, e.g., a math equation .

The **bookdown** package can be installed from CRAN or Github:

install.packages("bookdown")  
# or the development version  
# devtools::install\_github("rstudio/bookdown")

Remember each Rmd file contains one and only one chapter, and a chapter is defined by the first-level heading #.

To compile this example to PDF, you need XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): <https://yihui.name/tinytex/>.

# Introduction

You can label chapter and section titles using {#label} after them, e.g., we can reference Chapter 2. If you do not manually label them, there will be automatic labels anyway, e.g., Chapter ??.

Figures and tables with captions will be placed in figure and table environments, respectively.

par(mar = c(4, 4, .1, .1))  
plot(pressure, type = 'b', pch = 19)



Figure 1: Here is a nice figure!

Reference a figure by its code chunk label with the fig: prefix, e.g., see Figure 2. Similarly, you can reference tables generated from knitr::kable(), e.g., see Table 2.

knitr::kable(  
 head(iris, 20), caption = 'Here is a nice table!',  
 booktabs = TRUE  
)

Table 1: Here is a nice table!

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
| 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 5.0 | 3.6 | 1.4 | 0.2 | setosa |
| 5.4 | 3.9 | 1.7 | 0.4 | setosa |
| 4.6 | 3.4 | 1.4 | 0.3 | setosa |
| 5.0 | 3.4 | 1.5 | 0.2 | setosa |
| 4.4 | 2.9 | 1.4 | 0.2 | setosa |
| 4.9 | 3.1 | 1.5 | 0.1 | setosa |
| 5.4 | 3.7 | 1.5 | 0.2 | setosa |
| 4.8 | 3.4 | 1.6 | 0.2 | setosa |
| 4.8 | 3.0 | 1.4 | 0.1 | setosa |
| 4.3 | 3.0 | 1.1 | 0.1 | setosa |
| 5.8 | 4.0 | 1.2 | 0.2 | setosa |
| 5.7 | 4.4 | 1.5 | 0.4 | setosa |
| 5.4 | 3.9 | 1.3 | 0.4 | setosa |
| 5.1 | 3.5 | 1.4 | 0.3 | setosa |
| 5.7 | 3.8 | 1.7 | 0.3 | setosa |
| 5.1 | 3.8 | 1.5 | 0.3 | setosa |

You can write citations, too. For example, we are using the **bookdown** package (Xie [2019](#ref-R-bookdown)) in this sample book, which was built on top of R Markdown and **knitr** (Xie [2015](#ref-xie2015)).

### Email replies in more detail

I have just copied a few simple tutorials online incase you need a few resources.

*Im currently running an EFA* **(sorry im not sure is this shorthand stands for?)** *and I’m totally unsure of the cross-loading of some of the components!*

This is often the case when the relationships are weak or the power of the study is too low.

*I have attached output (it is the final rotated component matrix) as i have removed specific questions.*

*I would like to move onto a multiple regression analysis or something to determine meaningful results from my questionnaire however I’m just super unsure of ways to move forward* **I have started this in the report below but just need to sit down with some of your data I think**.

*What are your thoughts on the cross loading?* **Depends on the power of the data mostly I think**

as a general, the components are almost fitting well with the theory i generated the survey around (the theory of planned behaviour)- into attitudes, norms and perceived behaviours but due to the multiple weighings, I’m unsure how to get a full understanding of seperation. **This is a very hard question generally. Happy to discuss**

## Overall

Here I have put together a draft of some simple mixed models that can be looked at after the proposed objectives/hypothesis are laid out. It can sometimes be good to think of these sorts of problems as process diagrams as I have given below as an example

### Principal component analysis

“In simple words, principal component analysis is a method of extracting important variables (in form of components) from a large set of variables available in a data set. It extracts low dimensional set of features from a high dimensional data set with a motive to capture as much information as possible. With fewer variables, visualization also becomes much more meaningful. PCA is more useful when dealing with 3 or higher dimensional data.” [ref: here](https://www.analyticsvidhya.com/blog/2016/03/practical-guide-principal-component-analysis-python/).

Screenshot1

Screenshot1

**NOTE:** It is always performed on a symmetric correlation or covariance matrix. This means the matrix should be numeric and have standardized data.

And results often look like this by plotting the first and second dimentions like so…

showing the first two dimentions (PC1 and PC2) **NOTE** Any more than two diamentions are often hard to interperate and impossible if the power is low. To understand the number of diamentions needed the eigen values and vectors can be investigated.

### Normalization

The principal component analysis should be fitted to a normalized version of original predictors/variables collected. This is because, the original data may have different scales.

**For example:** Imagine a data set with variables’ measuring units as gallons, kilometers, light years etc. It is definite that the scale of variances in these variables will be large.

Performing PCA on un-normalized variables will lead to insanely large loadings for variables with high variance. In turn, this will lead to dependence of a principal component on the variable with high variance. This is undesirable.

### Mixed models

Mixed models essentially start with sample mean differences between groups and then add additional complexity in favour of poor fitting models.

You can write out models and variable relationships using statistical equations such as:

where the variables can be collected either through direct questions or transformed data from the orginal questions. What does it look like in a diagram explaining how the variables are proposed to be similar? Example below:

For example you might think that generally individuals that …

# Node and Edge Data Frames

You can label chapter and section titles using {#label} after them, e.g., we can reference Chapter 2. If you do not manually label them, there will be automatic labels anyway, e.g., Chapter ??.

Figures and tables with captions will be placed in figure and table environments, respectively.

par(mar = c(4, 4, .1, .1))  
plot(pressure, type = 'b', pch = 19)



Figure 2: Here is a nice figure!

Reference a figure by its code chunk label with the fig: prefix, e.g., see Figure 2. Similarly, you can reference tables generated from knitr::kable(), e.g., see Table 2.

knitr::kable(  
 head(iris, 20), caption = 'Here is a nice table!',  
 booktabs = TRUE  
)

Table 2: Here is a nice table!

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
| 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 5.0 | 3.6 | 1.4 | 0.2 | setosa |
| 5.4 | 3.9 | 1.7 | 0.4 | setosa |
| 4.6 | 3.4 | 1.4 | 0.3 | setosa |
| 5.0 | 3.4 | 1.5 | 0.2 | setosa |
| 4.4 | 2.9 | 1.4 | 0.2 | setosa |
| 4.9 | 3.1 | 1.5 | 0.1 | setosa |
| 5.4 | 3.7 | 1.5 | 0.2 | setosa |
| 4.8 | 3.4 | 1.6 | 0.2 | setosa |
| 4.8 | 3.0 | 1.4 | 0.1 | setosa |
| 4.3 | 3.0 | 1.1 | 0.1 | setosa |
| 5.8 | 4.0 | 1.2 | 0.2 | setosa |
| 5.7 | 4.4 | 1.5 | 0.4 | setosa |
| 5.4 | 3.9 | 1.3 | 0.4 | setosa |
| 5.1 | 3.5 | 1.4 | 0.3 | setosa |
| 5.7 | 3.8 | 1.7 | 0.3 | setosa |
| 5.1 | 3.8 | 1.5 | 0.3 | setosa |

You can write citations, too. For example, we are using the **bookdown** package (Xie [2019](#ref-R-bookdown)) in this sample book, which was built on top of R Markdown and **knitr** (Xie [2015](#ref-xie2015)).

### Node and Edge Data Frames

These functions are used to create and manipulate specialized data frames: node data frames (NDFs) and edge data frames (EDFs). The functions are useful because one can selectively add field data to these data frames and combine them as necessary before addition to a graph object.

### Creating an NDF

With the create\_node\_df() function, one can create a node data frame (NDF) with nodes and their attributes. This object is really just an R data.frame object. In most cases, it’s recommended to use create\_node\_df() instead of data.frame() (or, as.data.frame()) to create an NDF. Using create\_node\_df() allows for validation of the input data (so that the integrity of the graph is not compromised) and the function provides some additional functionality that the base R functions for data frame creation do not have:

* single values are repeated for n number of nodes supplied
* selective setting of attributes (e.g., giving attr values for 3 of 10 nodes, allowing non-set nodes to use defaults or globally set attr values)
* supplying overlong vectors for attributes will result in trimming down to the number of nodes
* setting label = FALSE will conveniently result in a non-labeled node

The function only has one argument that requires values to be supplied: the nodes argument. Just supplying a set of unique ID values for nodes will create an NDF with nodes that have no additional attributes.

When incorporated into a graph, an NDF is parsed and column names that match keywords for node attributes indicate to **DiagrammeR** that the values in such columns should provide attribute values on a per-node basis. Columns with names that don’t match reserved attribute names are disregarded and, because of this, you can include columns with useful data for analysis. When creating a node data frame, one column named nodes will be created (and it will be the first column in the NDF). That’s where unique values for the node ID should reside. As for other attribute columns, the type and label columns will also be created. However, these do not necessarily need to be populated with values and thus they can be left blank, if desired. Other attributes are voluntarily added as named vectors for the R triple-dot (...) argument. Here are all of the node attribute names and the types of values to supply:

* color — provide an X11 or hexadecimal color (append 2 digits to hex for alpha)
* distortion — the node distortion for any shape = polygon
* fillcolor — provide an X11 or hexadecimal color (append 2 digits to hex for alpha)
* fixedsize — true or false
* fontcolor — provide an X11 or hexadecimal color (append 2 digits to hex for alpha)
* fontname — the name of the font
* fontsize — the size of the font for the node label
* height — the height of the node
* penwidth — the thickness of the stroke for the shape
* peripheries — the number of peripheries (essentially, additional shape outlines)
* shape — the node shape (e.g., ellipse, polygon, circle, etc.)
* sides — if shape = polygon, the number of sides can be provided here
* style — usually given the value filled if you’d like to fill a node with a color
* tooltip — provide text here for an unstyled browser tooltip
* width — the width of the node
* x — the x position of the node (requires graph attr layout = neato to use)
* y — the y position of the node (requires graph attr layout = neato to use)

In the following examples, node data frames are created. There are a few worthwhile things to notice here. The nodes can be supplied as a character vector (as in c("a", "b", "c", "d")), or, as a range of integers (1:4). It may be a matter of preference, but the numbering system seems to be the better choice (and other functions available in the package will take advantage of a graph with ID values that are available as monotonically increasing integer values). Secondly, a single logical value can be supplied to the label argument, where TRUE copies the node ID to the label attribute, and FALSE yields a blank, unset value for all nodes in the NDF. Finally, the provision of single values in a call that creates more than a single node will result in all nodes having that attribute (e.g., color = "aqua" sets all nodes in the NDF with the ‘aqua’ color).

# Create a node data frame  
nodes\_1 <-  
 create\_node\_df(  
 n = 4,   
 type = "lower",  
 label = c("a", "b", "c", "d"),  
 style = "filled",  
 color = "aqua",  
 shape = c("circle", "circle",  
 "rectangle", "rectangle"),  
 data = c(3.5, 2.6, 9.4, 2.7))  
  
# Inspect the `nodes\_1` NDF  
nodes\_1  
#> id type label style color shape data  
#> 1 1 lower a filled aqua circle 3.5  
#> 2 2 lower b filled aqua circle 2.6  
#> 3 3 lower c filled aqua rectangle 9.4  
#> 4 4 lower d filled aqua rectangle 2.7

# Create another node data frame  
nodes\_2 <-  
 create\_node\_df(  
 n = 4,  
 type = "upper",  
 label = TRUE,  
 style = "filled",  
 color = "red",  
 shape = "triangle",  
 data = c(0.5, 3.9, 3.7, 8.2))  
  
# Inspect the `nodes\_2` NDF  
nodes\_2  
#> id type label style color shape data  
#> 1 1 upper 1 filled red triangle 0.5  
#> 2 2 upper 2 filled red triangle 3.9  
#> 3 3 upper 3 filled red triangle 3.7  
#> 4 4 upper 4 filled red triangle 8.2

### Creating an EDF

Using the create\_edge\_df() function, an edge data frame (EDF) (comprising edges and their attributes) is created. As with the create\_node\_df() function, the resulting object is an R data.frame object. While the usual means for creating data frames could be used to create an EDF, the create\_edge\_df() function provides some conveniences and validation of the final object (e.g., checking for uniqueness of the supplied node ID values). Also, there is a special attribute for a edge called a rel, which is short for relationship. This is an optional attribute, so, it can be left blank. However, it’s advantageous to have these types of group labels set for all edges, especially if the resulting graph is to be fashioned as a property graph. (The node type values must also be set for each node to model your graph as a property graph.)

Edges define the connections between nodes in a graph. So, in a sense, an edge data frame is a complementary component to the node data frame within a graph. Therefore, the EDF contains node ID information in two columns named from and to. So, when making an edge data frame, there are two equal-length vectors that need to be supplied to the create\_edge\_df() function: one for the outgoing node edge (from), and, another for the incoming node edge (to). Each of the two columns will contain node ID values. As for the node data frame, attributes can be provided for each of the edges. The following edge attributes can be used:

* arrowhead — the arrow style at the head end (e.g, normal, dot)
* arrowsize — the scaling factor for the arrowhead and arrowtail
* arrowtail — the arrow style at the tail end (e.g, normal, dot)
* color — the stroke color; an X11 color or a hex code (add 2 digits for alpha)
* dir — the direction; either forward, back, both, or none
* fontcolor — choose an X11 color or provide a hex code (append 2 digits for alpha)
* fontname — the name of the font
* fontsize — the size of the font for the node label
* headport — a cardinal direction for where the arrowhead meets the node
* label — label text for the line between nodes
* minlen — minimum rank distance between head and tail
* penwidth — the thickness of the stroke for the arrow
* tailport — a cardinal direction for where the tail is emitted from the node
* tooltip — provide text here for an edge tooltip

Here are a few examples of how edge data frames can be created:

# Create an edge data frame  
edges\_1 <-  
 create\_edge\_df(  
 from = c(1, 1, 2, 3),  
 to = c(2, 4, 4, 1),  
 rel = "requires",  
 color = "green",  
 data = c(2.7, 8.9, 2.6, 0.6))  
  
edges\_1  
#> id from to rel color data  
#> 1 1 1 2 requires green 2.7  
#> 2 2 1 4 requires green 8.9  
#> 3 3 2 4 requires green 2.6  
#> 4 4 3 1 requires green 0.6

# Create another edge data frame  
edges\_2 <-  
 create\_edge\_df(  
 from = c(5, 7, 8, 8),  
 to = c(8, 8, 6, 5),  
 rel = "receives",  
 arrowhead = "dot",  
 color = "red")  
  
edges\_2  
#> id from to rel arrowhead color  
#> 1 1 5 8 receives dot red  
#> 2 2 7 8 receives dot red  
#> 3 3 8 6 receives dot red  
#> 4 4 8 5 receives dot red

### Combining NDFs

Several node data frames (NDFs) can be combined into one using the combine\_ndfs() function. You can combine two NDFs in a single call of combine\_ndfs(), or, combine even more in a single pass.

There may be occasion to combine several NDFs into a single node data frame. The combine\_ndfs() function works much like the base R rbind() function except that it accepts NDFs with columns differing in number, names, and ordering. Obtaining several node data frames may result from collecting data from various sources, at different times (where the collected data is different), or, simply from multiple calls of create\_node\_df() for practical or readability purposes, to name a few examples. Speaking of examples:

# Create an NDF  
nodes\_1 <-  
 create\_node\_df(  
 n = 4,  
 label = 1:4,  
 type = "lower",  
 data = c(8.2, 5.2, 1.2, 14.9))  
  
# Create another NDF  
nodes\_2 <-  
 create\_node\_df(  
 n = 4,  
 label = 5:8,  
 type = "upper",  
 data = c(0.3, 6.3, 10.7, 1.2))  
  
# Combine the NDFs  
all\_nodes <- combine\_ndfs(nodes\_1, nodes\_2)  
  
all\_nodes  
#> id type label data  
#> 1 1 lower 1 8.2  
#> 2 2 lower 2 5.2  
#> 3 3 lower 3 1.2  
#> 4 4 lower 4 14.9  
#> 5 5 upper 5 0.3  
#> 6 6 upper 6 6.3  
#> 7 7 upper 7 10.7  
#> 8 8 upper 8 1.2

### Combining EDFs

There may be cases where one might take data from one or more data frames and create multiple edge data frames. One can combine multiple edge data frames (EDFs) with the combine\_edfs() function. Any number of EDFs can be safely combined with one call of this function.

Again, it is advantageous to use this combining function for the purpose of coalescing EDFs. Once a single, combined EDF is generated, it is much easier to incorporate that data into a **DiagrammeR** graph object.

# Create an edge data frame  
edges\_1 <-  
 create\_edge\_df(  
 from = c(1, 1, 2, 3),  
 to = c(2, 4, 4, 1),  
 rel = "requires",  
 color = "green",  
 data = c(2.7, 8.9, 2.6, 0.6))  
  
# Create another edge data frame  
edges\_2 <-  
 create\_edge\_df(  
 from = c(5, 7, 8, 8),  
 to = c(8, 8, 6, 5),  
 rel = "receives",  
 arrowhead = "dot",  
 color = "red")  
  
# Combine edge data frames with 'combine\_edfs'  
all\_edges <- combine\_edfs(edges\_1, edges\_2)  
  
all\_edges  
#> id from to rel color data arrowhead  
#> 1 1 1 2 requires green 2.7 <NA>  
#> 2 2 1 4 requires green 8.9 <NA>  
#> 3 3 2 4 requires green 2.6 <NA>  
#> 4 4 3 1 requires green 0.6 <NA>  
#> 5 5 5 8 receives red NA dot  
#> 6 6 7 8 receives red NA dot  
#> 7 7 8 6 receives red NA dot  
#> 8 8 8 5 receives red NA dot

# Likert Scales

## The problem

To build a sampling design that accounts for the issues and nature of data collection of foriegn bank movements using person-to-person == surveys.

## Method

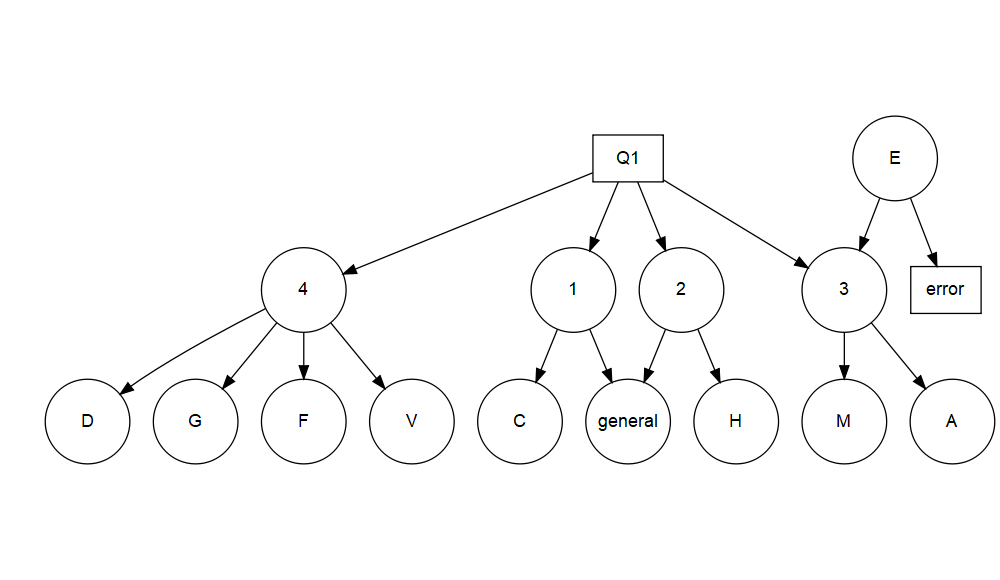
Here we want to address the key objectives of our reseach by collecting data in an unbias and representative way. To understand the true underlying trends in the data a pilot study with a large enough sample is explored below.

Here we have a difficult question to address due to the bias in the data capture method. This is really interesting and challenging from a statistical prospective. This is also a problem that the qualitative approach fails to address (waiting to see what others say about this..?)

Here is a step-by-step to approach to this research in a quantativite way:

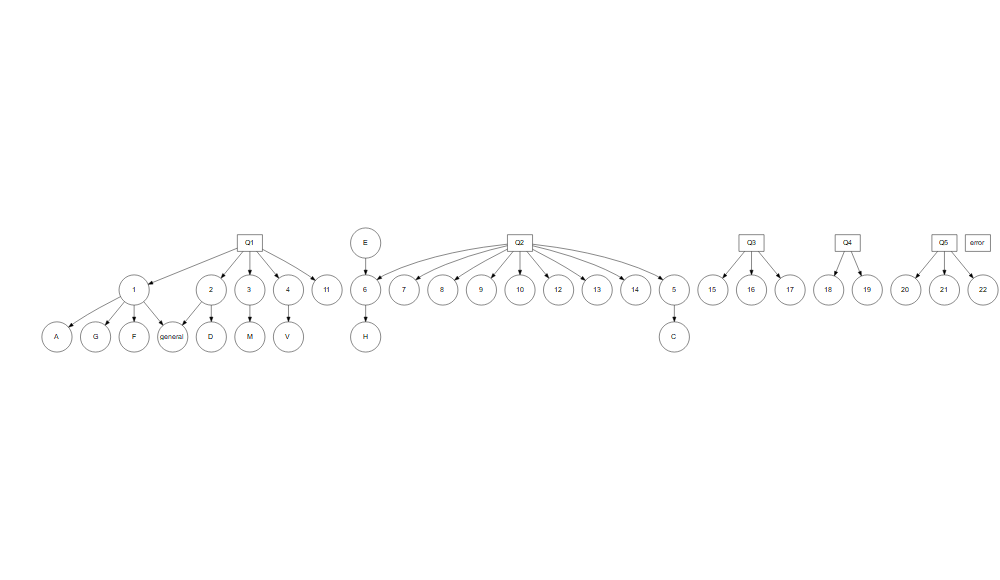
1. The questions are grouped into sets of questions that all address a general objective or question the research has. We can first see if these patterns are true for the responses. These are as follows:

grViz("  
digraph boxes\_and\_circles {  
  
 # a 'graph' statement  
 graph [overlap = true, fontsize = 10]  
  
 # several 'node' statements  
 node [shape = box,  
 fontname = Helvetica]  
 Q1; error  
  
 node [shape = circle,  
 fixedsize = true,  
 width = 0.9] // sets as circles  
 1; 2; 3; 4;  
  
 # several 'edge' statements  
 Q1->1 Q1->2 Q1->3 Q1->4   
 1->general 2->general 3->A 4->D 4->G 4->F  
 E->3 4->V 1->C 2->H 3->M E ->error  
}  
")

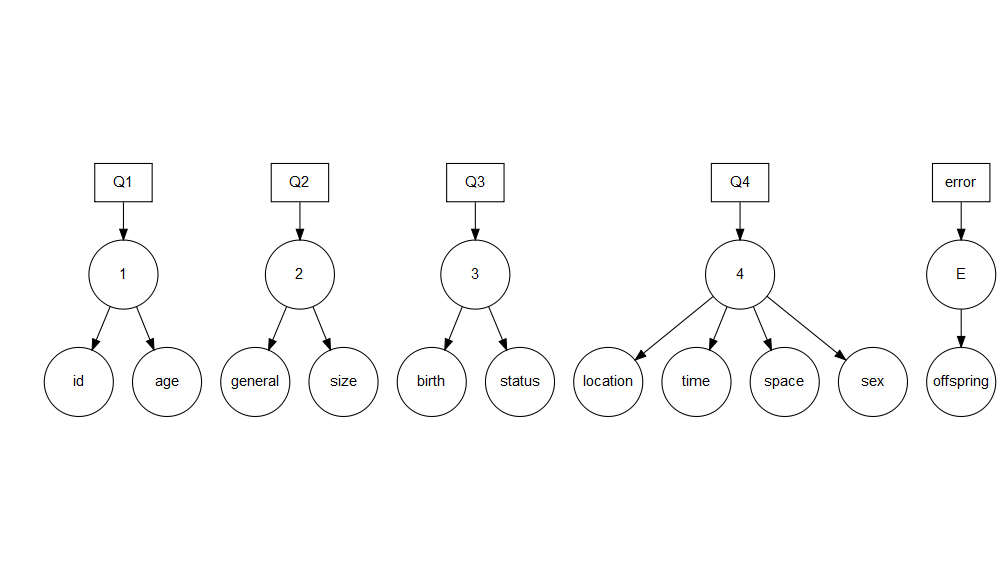


There are several issues with this because the data collected is often non-normal and needs to be transformed. The first step however is to simply visualise the data in a way that we will also be fitting the statistical models.

var <- c("scale","name","Q1.1","Q1.2","Q1.3","Q1.4","Q2.1","Q2.2","Q2.3","Q2.4","Q2.5","Q2.6","Q2.7","Q2.8","Q2.9","Q2.10","Q3.1","Q3.2","Q3.3","Q4.1","Q4.2","Q5.1","Q5.2","Q5.3")  
  
# dat <- read.csv("./data/overall\_dummy\_data.csv")  
# glimpse(dat)  
  
grViz("  
digraph boxes\_and\_circles {  
  
 # a 'graph' statement  
 graph [overlap = true, fontsize = 10]  
  
 # several 'node' statements  
 node [shape = box,  
 fontname = Helvetica]  
 Q1; Q2; Q3; Q4; Q5; error  
  
 node [shape = circle,  
 fixedsize = true,  
 width = 0.9] // sets as circles  
 1; 2; 3; 4; 5; 6; 7; 8  
  
 # several 'edge' statements  
 Q1->1 Q1->2 Q1->3 Q1->4   
 Q2->5 Q2->6 Q2->7 Q2->8 Q2->9 Q2->10 Q1->11 Q2->12 Q2->13 Q2->14  
 Q3->15 Q3->16 Q3->17  
 Q4->18 Q4->19  
 Q5->20 Q5->21 Q5->22  
 1->general 2->general 1->A 2->D 1->G 1->F  
 E->6 4->V 5->C 6->H 3->M  
}  
")



There is some simple dataset transformations that can be done to make it easier to work with the data in the plotting functions.



## Flow diagram

Here I think we should add the relationship to each question and the proposed hypothesis…

Xie, Yihui. 2015. *Dynamic Documents with R and Knitr*. 2nd ed. Boca Raton, Florida: Chapman; Hall/CRC. <http://yihui.name/knitr/>.

———. 2019. *Bookdown: Authoring Books and Technical Documents with R Markdown*. <https://CRAN.R-project.org/package=bookdown>.