**Create individual vectors**

For the purpose of clarity and ease of debugging, my approach will be to first set up each simulated variable as individual labelled vectors, and then bind them together into a data frame at the end. To adorn variable and value labels to a numeric vector, I will use set\_varl() and set\_vall() from {surveytoolbox} to do these tasks respectively.

I want to create a dataset with 1000 observations, so I will start with creating v\_id as an ID variable running from 1 to 1000, which can simply be generated with the seq() function.[3](https://martinctc.github.io/blog/vignette-simulating-a-minimal-spss-dataset-from-r/#fn3) I will then use set\_varl() from {surveytoolbox} to set a variable label for the v\_id vector. The second argument of set\_varl() takes in a character vector and assigns it as the variable label of the target variable – super straightforward.

## Record Identifier

v\_id <-

seq(1, 1000) %>%

set\_varl("Record Identifier")

The same goes for v\_gender, but this time I want to also (1) *apply an arbitrary probability to the distribution*, and (2) *give each value in the vector a value label (“Male”, “Female”, “Other”)*.

To do (1), I pass a numeric vector to the prob argument to represent the probabilities that 1, 2, and 3 will fall out for n = 1000.

To do (2), I run set\_vall() and pass the desired labels to the value\_labels argument. set\_vall() acccepts a named character vector to be assigned as value labels.

Finally, I run set\_varl() again to make sure that a variable label is present.

## Gender

v\_gender <-

sample(x = 1:3,

size = 1000, replace = TRUE,

prob = c(.48, .48, .04)) %>% # arbitrary probability

set\_vall(value\_labels = c("Male" = 1,

"Female" = 2,

"Other" = 3)) %>%

set\_varl("Q1. Gender")

Now that we’ve got our ID variable and a basic grouping variable (gender), let’s also create some mock metric variables.

I want to create a 5-point scale KPI variable (which could represent *customer satisfaction* or *likelihood to recommend*). One way to do this is to simply run sample() again, and do the same thing we did for v\_gender:

## KPI - #1 simple sampling

v\_kpi <-

sample(x = 1:5,

size = 1000,

replace = TRUE) %>%

set\_vall(value\_labels = c("Extremely dissatisfied" = 1,

"Somewhat dissatisfied" = 2,

"Neither" = 3,

"Satisfied" = 4,

"Extremely satisfied" = 5)) %>%

set\_varl("Q2. KPI")

Whilst the above approach is straightforward, the downside is that the numbers are likely to look completely random if we try to actually analyse the results – which is what sample() is supposed to do – but clearly isn’t ideal.

I want to simulate numbers that are more realistic, i.e. data which will form a discernible pattern when grouping and summarising by gender. What I’ll therefore do is to iterate through each number in v\_gender, and sample numbers based on the gender of the ‘respondent’.

The values that are passed below to the prob argument within sample() are completely arbitrary, but are designed to generate results where a bigger KPI value is more likely if v\_gender == 1, followed by v\_gender == 3, then v\_gender == 2.

Note that I’ve used map2\_dbl() here (from the {purrr} package, part of {tidyverse}), which “loops” through v\_gender and returns a numeric value for each iteration.

## KPI - #2 gender-dependent sampling

v\_kpi <-

v\_gender %>%

map\_dbl(function(x){

if(x == 1){

sample(1:5,

size = 1,

prob = c(10, 17, 17, 28, 28)) # Sum to 100

} else if(x == 2){

sample(1:5,

size = 1,

prob = c(11, 22, 28, 22, 17)) # Sum to 100

} else {

sample(1:5,

size = 1,

prob = c(13, 20, 20, 27, 20)) # Sum to 100

}

}) %>%

set\_vall(value\_labels = c("Extremely dissatisfied" = 1,

"Somewhat dissatisfied" = 2,

"Neither" = 3,

"Satisfied" = 4,

"Extremely satisfied" = 5)) %>%

set\_varl("Q2. KPI")

To add a level of complexity, let me also simulate a mock NPS variable. One way to do this is to punch in random numbers like how it is done above with v\_kpi, but this will involve a lot more random punching than is desirable for a 11-point scale NPS variable.

I will therefore instead write a custom function called skew\_inputs() that ‘expands’ three arbitrary input numbers into 11 numbers, which will then serve as the probability anchors for my sample() functions later on.

## Generate skew inputs for sample probability

##

## `value1`, `value2` and `value3`

## generate the skewed probabilities

##

skew\_inputs <- function(value1, value2, value3){

all\_n <-

c(rep(value1, 7), # 0 - 6

rep(value2, 2), # 7 - 8

rep(value3, 2)) # 9 - 10

return(sort(all\_n))

}

## Outcome KPI - NPS

v\_nps <-

v\_gender %>%

map\_dbl(function(x){

if(x == 1){

sample(0:10, size = 1, prob = skew\_inputs(1, 1, 8))

} else if(x == 2){

sample(0:10, size = 1, prob = skew\_inputs(2, 3, 5))

} else if(x == 3){

sample(0:10, size = 1, prob = skew\_inputs(1, 3, 6))

} else {

stop("Error - check x")

}

}) %>%

set\_varl("Q3. NPS")

Admittedly that the above procedure isn’t *minimal*, but note that this is a trade-off to introduce some arbitrary patterns to the data. A ‘quick and dirty’ alternative simulation would simply be to run sample(x = 0:10, size = 1000, replace = TRUE) for v\_nps.

There is one slight technicality: the so-called NPS question is strictly speaking a *likelihood to recommend* question which ranges from 0 to 10, and the **Net Promoter Score** itself is calculated on a recoded version of that question where *Detractors* (scoring 0 to 6) have to be coded as -100, *Passives* (scoring 7 to 8) as 0, and *Promoters* (scoring 9 to 10) as +100. The **Net Promoter Score** is simply calculated as a mean of those recoded values.

Fortunately, the {surveytoolbox} package comes shipped with a as\_nps() function that does this recoding for you, and also automatically applies the value labels. let’s call this new variable v\_nps2:

## Outcome KPI - Recoded NPS (NPS2)

v\_nps2 <- as\_nps(v\_nps) %>% set\_varl("Q3X. Recoded NPS")

**Combine vectors**

Now that all the individual variables are set up, I can simply combine them all into a tibble in one swift movement[4](https://martinctc.github.io/blog/vignette-simulating-a-minimal-spss-dataset-from-r/#fn4):

#### Combine individual vectors ####

combined\_df <-

tibble(id = v\_id,

gender = v\_gender,

kpi = v\_kpi,

nps = v\_nps,

nps2 = v\_nps2)

**Results!**

image from Giphy

Let’s run a few checks on our dataset to confirm that everything has worked out okay.

The classic {dplyr} glimpse():

combined\_df %>% glimpse()

## Observations: 1,000

## Variables: 5

## $ id 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 1...

## $ gender 2, 2, 1, 2, 2, 1, 1, 2, 1, 2, 1, 1, 2, 2, 1, 1, 2, 2, 2,...

## $ kpi 3, 2, 5, 2, 5, 5, 2, 2, 5, 3, 3, 4, 1, 5, 4, 4, 4, 1, 2,...

## $ nps 10, 5, 10, 8, 10, 9, 7, 5, 2, 5, 1, 4, 5, 9, 9, 10, 9, 3, 10...

## $ nps2 100, -100, 100, 0, 100, 100, 0, -100, -100, -100, -100, ...

Then head() to see the first five rows:

combined\_df %>% head()

## # A tibble: 6 x 5

## id gender kpi nps nps2

##

## 1 1 2 [Female] 3 [Neither] 10 100 [Promoter]

## 2 2 2 [Female] 2 [Somewhat dissatisfied] 5 -100 [Detractor]

## 3 3 1 [Male] 5 [Extremely satisfied] 10 100 [Promoter]

## 4 4 2 [Female] 2 [Somewhat dissatisfied] 8 0 [Passive]

## 5 5 2 [Female] 5 [Extremely satisfied] 10 100 [Promoter]

## 6 6 1 [Male] 5 [Extremely satisfied] 9 100 [Promoter]

So it appears that the value labels have been properly attached, and the range of values are what we’d expect. Now what about the “fake patterns”?

Looking at the topline result of the data, we seem to have succeeded in fabricating some sensible patterns in the data. It appears that this company X will need to work harder at winning over its female customers, who have rated them lower on two KPI metrics:

combined\_df %>%

group\_by(gender) %>%

summarise(n = n\_distinct(id),

kpi = mean(kpi),

nps2 = mean(nps2))

## # A tibble: 3 x 4

## gender n kpi nps2

##

## 1 1 [Male] 490 3.49 31.0

## 2 2 [Female] 464 3.07 -8.62

## 3 3 [Other] 46 3.15 17.4

**Check the labels **

Finally I’d like to share a couple of functions that enable you to explore the labels in a labelled dataset. surveytoolbox::varl\_tb() accepts a labelled data frame, and returns a two-column data frame with the variable name and its corresponding variable label:

combined\_df %>% varl\_tb()

## # A tibble: 5 x 2

## var var\_label

##

## 1 id Record Identifier

## 2 gender Q1. Gender

## 3 kpi Q2. KPI

## 4 nps Q3. NPS

## 5 nps2 Q3X. Recoded NPS

surveytoolbox::data\_dict() takes this further, and shows also the value labels as a third column. This is what effectively what’s typically referred to as a **code frame** in a market research context:

combined\_df %>%

select(-id) %>%

data\_dict()

## var label\_var

## 1 gender Q1. Gender

## 2 kpi Q2. KPI

## 3 nps Q3. NPS

## 4 nps2 Q3X. Recoded NPS

## label\_val

## 1 Male; Female; Other

## 2 Extremely dissatisfied; Somewhat dissatisfied; Neither; Satisfied; Extremely satisfied

## 3

## 4 Detractor; Passive; Promoter; Missing value

## value

## 1 1; 2; 3

## 2 1; 2; 3; 4; 5

## 3 0; 1; 2; 3; 4; 5; 6; 7; 8; 9; 10

## 4 -100; 0; 100

I would also highly recommend the view\_df() function from {sjPlot}, which exports a similar overview of variables and labels in a nicely formatted HTML table. For huge labelled datasets, this offers a fantastic light-weight way to browse through your variables and labels.

combined\_df %>% sjPlot::view\_df()

Once we’ve checked all the labels and we’re happy with everything, we can then export our dataset with haven::write\_sav()! If everything’s worked properly, all the labels should appear properly if you choose to open your example dataset in SPSS, or Q:

combined\_df %>% haven::write\_sav("Simulated Dataset.sav")