(Very) basic steps to weight a survey sample

An explained example with R code - V0.3

Josep Espasa Reig

Table of Contents

library(RCurl)  
library(MASS)  
library(glmnet)  
library(caret)  
library(survey)  
library(readxl)  
library(stringr)  
library(forcats)  
library(foreign)  
library(magrittr)  
library(tidyverse)  
options(scipen = 9999)  
options(dplyr.width = Inf)  
  
  
set.seed(456162)

## Introduction

### Aim of this guide and further readings

This is a short introductory guide that shows the basic procedures to weight a survey. Please keep in mind that there are many different ways of doing this. This guide intends to be a practical document and a step-by-step walkthrough for weighting a survey. It provides [R code](https://www.r-project.org/) for all actions: from reading, manipulating and presenting data to modelling and callibration. This should allow readers to reproduce procerudes and results as well as to inspect objects at any given part of this guide. The source code in an 'R notebook' format can be found and publicly accessed from the [author's github page](https://github.com/JosepER/PPMI_how_to_weight_a_survey/01-_Basic_steps_on_how_to_weight_a_survey_sample.Rmd). It is important to note that this is not an 'R tutorial'. Thus, the guide does not provide a detailed comment on general functions used in this walk-through. Some intermediate 'R' skills might be requiered to follow all steps of the procedure. Readers with basic or no knowledge of 'R' can still benefit from this note as it explains the steps and principles behind weighting.

For questions, clarifications or suggestions feel free to contact him at *jespasareig at gmail.com*. It is relevant to say that this text intentionally avoids explaining complex or advanced methods. Instead, it aims at providing users with a standard way of weighting and a limited number of variations.

Next sections will first give a very broad glimpse at all [main survey weighting steps](#basic%20steps%20in%20weighting%20a%20survey).The second section deals with importing data into R, data manipulation and briefly [presenting the dataset used for this guide](#exploring%20and%20presenting%20the%20dataset). Readers interested in a specific step, familiar with the 7th round of the European Social Survey or that want to jump directly into weighting procedures can skip this part of the guide. The three sections are the main components of this guide and show how to compute [design weights](#design_weights), [non-response weights](#nonresponse_weights) and [calibration weights](#calibration). Two more sections will be added to this survey in the future. These correspond to the analysis of weight variability and computing weighted estimates.

For more information you can check the following introductory texts:

* Valliant et al. (2013) *Practical Tools for Designing and Weighting Survey Samples*. New York: Springer Science Business Media.
* Lohr,S.L. (2009) *Sampling: Design and Analysis*. 2nd Edition. Boston: Books/Cole.
* Blair, E. & Blair, J. (2015) *Applied Survey Sampling*. London: SAGE Publications Inc.

And the book accompaining the R 'survey' package:

* Lumley,T. (2010) *Complex Surveys: A Guide to Analysis Using R*. New Jersey: John Wiley & Sons Inc.

It might be also worth keeping an eye on the (still incipient) R package [*srvyr*](https://cran.r-project.org/web/packages/srvyr/index.html), developed and mantained by Greg Freedman Ellis.

Note: This guide focuses on surveys based on 'probability sample'. These are surveys where all units in our statistical population have a chance of being selected and the probability of selection is known to the researcher. A brief note on how to weight non-probability samples is included at the end of the guide.

### Basic steps in weighting a survey

Weights are applied to reduce survey bias. In plain words, weighting consists on making our sample of survey respondents (more) representative of our statistical population. By statistical population we mean all those units for which we want to compute estimates.

There are four basic steps in weighting. These are:

1. **Base/design weights**
2. **Non-response weights**
3. **Use of auxiliary data/calibration**
4. **Analysis of weight variability/trimming**

The first step consists on computing weights to take into account the differences of units in the probability of being sampled. 'Being sampled' means being selected from the survey frame (i.e. the list of all units) to be approached for a survey response. This step can be skipped if all units in the survey frame have the same probability of being sampled. This happens, for example: \* when all units in the survey frame are approached for the sample or; \* with certain sampling designs (such as 'simple random sampling without replacement' or 'stratified random sampling without replacement' with distribution of sampled units across stratums proportional to the number of units in each stratum). These are usually called 'self-weighted' surveys.

In the second step we need to adjust our responses by the differences in the probability of sampled units to reply to our survey. Our estimates would be biased if some profile of sampled units had higher propensity to reply than another and these profiles had differences in the dependent variables (i.e. our variables of interest). In this step, we need to estimate the probability of response using information available for both respondents and non-respondents. Non-response adjustment is not needed if all sampled units responded to the survey (i.e. in probability sampling surveys with 100% response rates).

The third step consists on adjusting our weights using available information about total population estimates. Note that this requieres data that is different from that needed in non-response adjustment (second step). Here we need auxiliary data which tells us information (i.e. estimates such as proportions, means, sums, counts) about the statistical population. The same variables should be available from our respondents but here we don't need information about non-respondents.

The last step is to check the variablity in our computed weights. High variation in weights can lead to some observations having too much importance in our sample. Even if weights reduce bias, they might largely inflate variance of estimates. Therefore, some survey practitioners worry about dealing with highly unequal weights.

## Read data and data management

We first need to import data into R. In this guide we will use UK data from the 7th round of the [European Social Survey](http://www.europeansocialsurvey.org/). The advantage of this data is that the European Social Survey (ESS) is a well documented and high quality probability survey. It allows us to understand how responses were collected and provides some useful information about non-respondents. At the same time, the 7th ESS was weighted by expert statisticians. The process of the two phases of weighting they applied is explained in their [website](http://www.europeansocialsurvey.org/methodology/ess_methodology/data_processing_archiving/weighting.html). This will allow us to compare our own weights and results with those already computed by their team of experts. Focusing on the UK sample will allow us to narrow down the analysis and fasten computation by reducing the amount of data used in each step.

For this guide we will use the following 7th ESS datafiles in SPSS ('.sav') format:

* [sample data (SDDF), edition 1.1](http://www.europeansocialsurvey.org/download.html?file=ESS7SDDFe01_1&y=2014), which contains the probability of being sampled for all respondents and non-respondents invited to the survey;
* [the data from Contact forms, edition 2.1](http://www.europeansocialsurvey.org/download.html?file=ESS7CFe02_1&y=2014), which provides information about the process of data collection (e.g. number of times the person was approached for a response, ID of interviewer in each approach, conditions of the house/area where the potential respondent lived.). We will call this data the *'paradata'* of the survey;
* [the integrated interviewer data file, edition2.1](http://www.europeansocialsurvey.org/download.html?file=ESS7INTe02_1&y=2014). These are the responses to the survey.

The following sections explain data import, selection, merging and recoding. Readers who are not interested in technical details about datasets can skip them and jump directly into [exploration and presention of the data](#exploring%20and%20presenting%20the%20dataset)

#### Import data

The following chunk of code loads the data sets from a *data* folder in the working directory. The sample data file is stored into the *sample.data* 'data\_frame' object. The contact forms information is stored into the *paradata* folder. Survey responses from the integrated interviewer file are saved in the *responses* object. We also store the weight variables included in the integrated interviewer file in the *original.weights* data\_frame.

sample.data <- read.spss("data/ESS7SDDFe1\_1.sav", to.data.frame = T) %>%  
 filter(cntry == "United Kingdom")  
  
paradata <- read.spss("data/ess7CFe02\_1.sav", to.data.frame = T) %>%  
 filter(cntry == "United Kingdom")   
  
responses <- read.spss("data/ESS7e02\_1.sav", to.data.frame = T) %>%  
 filter(cntry == "United Kingdom")   
  
original.weights <- responses %>% select(idno ,dweight, pspwght, pweight)

#### Select variables

Once the data has been read into R, we select the variables we are going to use in our analysis. Selecting variables is a good practice as the ESS files contain much more information that we need for this example. This will allow us to easily find and see the data that is important for us in this guide. Here we just write the names of the variables we intend to use and we will later explain the content of these in more substantial terms.

vars.sample.data <- c("idno", "psu", "prob")  
  
vars.paradata <- c("idno", "typesamp", "interva", "telnum",   
 "agea\_1", "gendera1", "type", "access",   
 "physa", "littera", "vandaa")  
  
resp.id <- c("idno")  
  
resp.y <- c("cgtsmke", "cgtsday",  
 "alcfreq", "alcwkdy", "alcwknd")  
  
resp.x <- c("vote", "prtvtbgb",  
 "prtclbgb", "prtdgcl",  
 "ctzcntr", "ctzshipc",  
 "brncntr","cntbrthc",  
 "gndr", "agea", "hhmmb","eisced", "region",  
 "pdwrk", "edctn", "uempla", "uempli", "rtrd",  
 "wrkctra", "hinctnta")

We will also keep the variable labels from the SPSS (.sav) file, although these are not so common in R.

selected.labels.sample.data <- attributes(sample.data)$variable.labels[which(names(sample.data) %in% vars.sample.data)]  
  
selected.labels.paradata <- attributes(paradata)$variable.labels[which(names(paradata) %in% vars.paradata)]  
  
selected.labels.responses <- attributes(responses)$variable.labels[which(names(responses) %in% c(resp.y, resp.x))]   
  
attributes(responses)$variable.labels %>%   
 cbind(names(responses),.) %>%   
 as\_data\_frame %>%   
 write\_csv("interim\_output/variable\_labels.csv")

Now we do the selection of variables from the three data sets using the names of the variables written a couple of code chunks ago.

sample.data %<>%   
 .[vars.sample.data]  
  
paradata %<>%  
 .[vars.paradata]  
  
responses %<>%  
 .[which(names(responses) %in% c(resp.id, resp.y, resp.x))]

#### Merging datafiles

After selecting the variables for the analysis, we merge the 'paradata' file containing all sampled units (respondents and non-respondents) with the 'survey responses' file, containing interview responses (only for respondents). The resulting data\_frame is the 'data' object. It contains the 'paradata' information for all sampled individuals and responses for those that were interviewed successfully.

In a real situation where we collect the data ourselves we would also have a 'survey frame'. This 'survey frame' would ideally include include all units from the population and characteristics of these such as stratification variables. A survey frame would include sampled units (respondents and non-respondents) as well as non-sampled units.

data <- paradata %>%  
 left\_join(sample.data, by = "idno") %>%  
 left\_join(responses, by = "idno") %>%  
 arrange(interva)   
  
rm(paradata,  
 sample.data,  
 responses)

Here we add the variable labels to the datasets with we kept before.

attributes(data)$variable.labels <- c(selected.labels.paradata, selected.labels.sample.data[!names(selected.labels.sample.data) %in% "idno"],  
 selected.labels.responses)

#### Recoding

Here we will recode our two dependent variables: cigarette and alcohol consumption. All those respondents that don't smoke should have a 0 in the **cigarretes smoked per day** variable. To calculate the **alcohol consumption** of respondents, we first calculate the daily consumption of alcohol if they were to consume alcohol daily and then weight by their stated frequency of alcohol consumption.

data$cgtsday[data$cgtsmke %in% c("I have never smoked",  
 "I don't smoke now but I used to",  
 "I have only smoked a few times")] <- 0  
  
data$alcohol\_day <- NA   
data$alcohol\_day <- (data$alcwkdy \* 5 + data$alcwknd \*2)/7   
  
data$alcohol\_day[which(data$alcfreq == "Several times a week")] <- data$alcohol\_day / 2.5  
data$alcohol\_day[which(data$alcfreq == "Once a week")] <- data$alcohol\_day/7  
data$alcohol\_day[which(data$alcfreq == "2-3 times a month")] <- data$alcohol\_day/10  
data$alcohol\_day[which(data$alcfreq == "Once a month")] <- data$alcohol\_day/30  
data$alcohol\_day[which(data$alcfreq == "Less than once a month")] <- data$alcohol\_day/50  
data$alcohol\_day[which(data$alcfreq == "Never")] <- 0  
  
resp.y <- c(resp.y, "alcohol\_day")

### Exploring and presenting the dataset

The merged data set contains sampled **respondents and non-respondents**. It contains a total of 5600 units and 39 variables.

dim(data)

## [1] 5600 39

The data set contains information about 2265 respondents and 3335 non-respondents.

And this is a list of the variables it contains (with their labels). **idno** is the individual identification variable.

data.variables <- cbind(names(data),attributes(data)$variable.labels) %>%   
 as\_data\_frame()  
  
data.variables$V2 <- format(data.variables$V2 , justify = "left")  
  
data.variables %>%  
 print(n = 40)

## # A tibble: 39 x 2  
## V1 V2  
## <chr> <chr>  
## 1 idno Respondent's identification number   
## 2 typesamp Type of the sample   
## 3 interva Interview information for the sample unit   
## 4 telnum Telephone number   
## 5 agea\_1 Estimation of age of respondent or household member who refuses, by interviewer  
## 6 gendera1 Gender of respondent or household member who refuses, recorded by interviewer   
## 7 type Type of house respondent lives in   
## 8 access Entry phone or locked gate/door before reaching respondent's individual door   
## 9 physa Assessment overall physical condition building/house   
## 10 littera Amount of litter and rubbish in the immediate vicinity   
## 11 vandaa Amount of vandalism and graffiti in the immediate vicinity   
## 12 psu PSU   
## 13 prob PROB   
## 14 vote Voted last national election   
## 15 prtvtbgb Party voted for in last national election, United Kingdom   
## 16 prtclbgb Which party feel closer to, United Kingdom   
## 17 prtdgcl How close to party   
## 18 ctzcntr Citizen of country   
## 19 ctzshipc Citizenship   
## 20 brncntr Born in country   
## 21 cntbrthc Country of birth   
## 22 cgtsmke Cigarettes smoking behaviour   
## 23 cgtsday How many cigarettes smoke on typical day   
## 24 alcfreq How often drink alcohol   
## 25 alcwkdy Grams alcohol, last time drinking on a weekday, Monday to Thursday   
## 26 alcwknd Grams alcohol, last time drinking on a weekend day, Friday to Sunday   
## 27 hhmmb Number of people living regularly as member of household   
## 28 gndr Gender   
## 29 agea Age of respondent, calculated   
## 30 eisced Highest level of education, ES - ISCED   
## 31 pdwrk Doing last 7 days: paid work   
## 32 edctn Doing last 7 days: education   
## 33 uempla Doing last 7 days: unemployed, actively looking for job   
## 34 uempli Doing last 7 days: unemployed, not actively looking for job   
## 35 rtrd Doing last 7 days: retired   
## 36 wrkctra Employment contract unlimited or limited duration   
## 37 hinctnta Household's total net income, all sources   
## 38 region Region   
## 39 alcohol\_day Respondent's identification number

rm(data.variables)

The goal of this guide will be to to give UK population estimates for cigarette and alcohol consumption based on ESS respondents. These will be our *Y* variables or variables of interest. The idea is to give descriptives of the distribution of these two variables (such as quantiles and mean) and then a simple extrapolation and compute total cigarette and alcohol consumption for the whole UK.

These are our *Y* variables:

* **cgtsday** : Number of cigarettes smoked on a typical day.
* **alcohol\_day** : Grams of alcohol ingested on a daily basis. Computed in the [*Recoding*](#recoding) section from the amount of alcohol drank last time during weekdays and weekend.

data[resp.y] %>%  
 as\_data\_frame() %>%  
 print()

## # A tibble: 5,600 x 6  
## cgtsmke cgtsday alcfreq alcwkdy alcwknd alcohol\_day  
## \* <fctr> <dbl> <fctr> <dbl> <dbl> <dbl>  
## 1 I have never smoked 0 Once a week 0 8 0.32653061  
## 2 I smoke daily 10 Once a week 48 96 8.81632653  
## 3 I have never smoked 0 Several times a week 42 0 0.91428571  
## 4 I have never smoked 0 Never NA NA 0.00000000  
## 5 I smoke daily 20 2-3 times a month 55 0 0.03265306  
## 6 I don't smoke now but I used to 0 Several times a week 26 32 24.68571429  
## 7 I don't smoke now but I used to 0 Once a week 17 0 0.13061224  
## 8 I smoke daily 15 Never NA NA 0.00000000  
## 9 I have never smoked 0 Never NA NA 0.00000000  
## 10 I smoke daily 20 Several times a week 42 121 12.00000000  
## # ... with 5,590 more rows

#### Paradata variables

The 7th ESS survey contains variables which give information about the data collection process. First, we have some variables that come from the 'sample data (SDDF)' file. These contain info about the 'primary sampling unit' and the probability of each unit of being selected in the sample. These two variables are only available for respondents. In a real project we would most probably have to compute the probability of being sampled ourselves.

* **psu**: This variable includes information on the primary sampling unit (PSU). In the UK this refers to the 'postcode address file'.
* **prob**: Probability of being included in the sample (i.e. approached for survey).

data[vars.sample.data] %>%  
 as\_data\_frame() %>%  
 print()

## # A tibble: 5,600 x 3  
## idno psu prob  
## \* <dbl> <dbl> <dbl>  
## 1 100000003 9388 0.00020306164  
## 2 100000005 9241 0.00020306164  
## 3 100000008 9472 0.00020306164  
## 4 100000009 9450 0.00005076541  
## 5 100000010 9479 0.00010153082  
## 6 100000012 9460 0.00010153082  
## 7 100000015 9440 0.00010153082  
## 8 100000016 9494 0.00020306164  
## 9 100000017 9376 0.00010153082  
## 10 100000020 9273 0.00010153082  
## # ... with 5,590 more rows

The 7th ESS also contains variables for all sampled units (i.e. respondents and non-respondents). These give information about the events that occurred during the data collection process. We will use these variables as covariates during the computation of **Non-response weights** in [step two](#nonresponse%20weights).

* **typesamp**: Refers to the type of unit sampled. In the UK addresses were the final sampling units. In some other countries these were households and individual people.
* **interva**: Shows the final outcome of the contact. In the UK sample, only codes 'Complete ...' and 'No interview ...' were used for respondents and non-respondents respectively.
* **telnum**: The interviewed person gave his/her mobile phone to the interviewer.
* **agea\_1**: Interviewer estimation of age of respondent or household member who refuses to give the interview.
* **gendera1**: Interviewer estimation of gender of respondent or household member who refuses to give the interview.
* **type**: Type of house sampled unit lives in.
* **access**: Entry phone or locked gate/door before reaching respondent's individual door.
* **physa**: Interviewer assessment overall physical condition building/house.
* **littera**: Interviewer assessment of amount of litter and rubbish in the immediate vicinity.
* **vandaa**: Interviewer assessment of amount of vandalism and graffiti in the immediate vicinity.

data[vars.paradata] %>%  
 head(6)

## idno typesamp interva telnum  
## 1 100000003 Address Complete and valid interview related to CF Present  
## 2 100000005 Address Complete and valid interview related to CF Present  
## 3 100000008 Address Complete and valid interview related to CF Present  
## 4 100000009 Address Complete and valid interview related to CF Present  
## 5 100000010 Address Complete and valid interview related to CF Present  
## 6 100000012 Address Complete and valid interview related to CF Present  
## agea\_1 gendera1 type access  
## 1 <NA> <NA> Single unit: Terraced house No, neither of these  
## 2 <NA> <NA> Single unit: Semi-detached house No, neither of these  
## 3 <NA> <NA> Single unit: Terraced house Yes, locked gate/door  
## 4 <NA> <NA> Single unit: Semi-detached house No, neither of these  
## 5 <NA> <NA> Single unit: Terraced house No, neither of these  
## 6 <NA> <NA> Single unit: Terraced house No, neither of these  
## physa littera vandaa  
## 1 Satisfactory Small amount None or almost none  
## 2 Good None or almost none None or almost none  
## 3 Good None or almost none None or almost none  
## 4 Satisfactory Large amount None or almost none  
## 5 Satisfactory None or almost none None or almost none  
## 6 Very good None or almost none None or almost none

#### Survey responses

Apart from the variables of interest (cigarette and alcohol consumption) our dataset has other variables obtained from survey responses. Obviously, these are only available for respondents. We will try to use some of these variables to calibrate the survey in [**Use of auxiliary data/calibration** step](#calibration). Some of these variables are:

* **vote**: Voted last national election (Yes/No)
* **prtvtbgb**: Party voted for in last national election
* **prtclbgb**: Which party feel closer to, United Kingdom
* **prtdgcl**: How close does the repondent feel to the party party from 'prtclbgb'
* **ctzcntr**: Has UK citizenship (Yes/No)
* **ctzshipc**: Citizenship of respondent
* **brncntr**:Respondent born in the UK
* **cntbrthc**: Respondent country of birth
* **gndr**: Gender of respondent
* **agea**: Calculated age of respondent
* **eisced**: Highest level of education of respondent
* **pdwrk**: In paid work
* **edctn**: In education
* **uempla**: In unemployment, actively looking for a job
* **uempli**: In unemployment, not actively looking for a job
* **rtrd**: Retired
* **wrkctra**: Employment contract unlimited or limited duration
* **hinctnta**: Household's total net income, all sources

data[c(resp.id, resp.x)] %>%  
 head(6)

## idno vote prtvtbgb prtclbgb prtdgcl ctzcntr ctzshipc  
## 1 100000003 Yes Liberal Democrat <NA> <NA> Yes <NA>  
## 2 100000005 Yes Liberal Democrat <NA> <NA> Yes <NA>  
## 3 100000008 Yes Labour Labour Not close Yes <NA>  
## 4 100000009 Yes Labour <NA> <NA> Yes <NA>  
## 5 100000010 No <NA> <NA> <NA> Yes <NA>  
## 6 100000012 No <NA> <NA> <NA> Yes <NA>  
## brncntr cntbrthc gndr agea hhmmb  
## 1 Yes <NA> Female 49 1  
## 2 Yes <NA> Female 45 2  
## 3 Yes <NA> Male 76 1  
## 4 No <NA> Male 50 5  
## 5 Yes <NA> Male 67 2  
## 6 Yes <NA> Female 31 2  
## eisced  
## 1 ES-ISCED V2, higher tertiary education, >= MA level  
## 2 ES-ISCED V1, lower tertiary education, BA level  
## 3 ES-ISCED I , less than lower secondary  
## 4 ES-ISCED II, lower secondary  
## 5 ES-ISCED I , less than lower secondary  
## 6 ES-ISCED IIIa, upper tier upper secondary  
## region pdwrk edctn uempla uempli  
## 1 West Midlands (England) Marked Not marked Not marked Not marked  
## 2 Scotland Marked Not marked Not marked Not marked  
## 3 East Midlands (England) Not marked Not marked Not marked Not marked  
## 4 South East (England) Not marked Not marked Not marked Not marked  
## 5 Yorkshire and the Humber Not marked Not marked Not marked Not marked  
## 6 South West (England) Marked Not marked Not marked Not marked  
## rtrd wrkctra hinctnta  
## 1 Not marked Unlimited M - 4th decile  
## 2 Not marked No contract R - 2nd decile  
## 3 Marked Unlimited R - 2nd decile  
## 4 Not marked Unlimited S - 6th decile  
## 5 Marked <NA> J - 1st decile  
## 6 Not marked Unlimited F - 5th decile

## Step 1: Design weights

The first step in weighing is taking into account the different probabilities of being sampled that respondents may have. The 7th ESS did not use a register of people in the UK (in other countries they did). They first selected postcode sectors from the Post Office’s small user postcode address file (PAF) merging smaller sectors. The probability of each PAF of being selected was proportional to the number of addresses it contained. Then, they selected 20 addresses inside of each sampled PAF and a dwelling for each address. For each dewlling they selected a household and then a person in each household. The full explanation of the sampling procedure is given in page 163 of The data documentation report ([Edition 3.1](http://www.europeansocialsurvey.org/docs/round7/survey/ESS7_data_documentation_report_e03_1.pdf)).

This sampling design is typical of survey frames where there is an available/public list of addresses but not a list of households or individuals. If we don't weight this survey, we would probably over-represent people in addresses that have smaller number of dwellings, dwellings that include smaller number of households and households that comprise smaller number of people (we will actually see this below).

Fortunately for us, the probablity of each respondent of being sampled was computed by national experts and included in the 7th ESS dataset. In many projects, however, we would have to compute sampling probailities ourselves[[1]](#footnote-42). A basic but important test that should be performed after computing the probabilities is **making sure that all probabilities are between 0 and 1**.

We will perform this test in the next chunk of code, which should give us an error if any of the probabilities is not in the interval .

probabilities <- data %>%  
 summarise(min.probability = min(prob, na.rm = T),  
 max.probability = max(prob, na.rm = T)) %>%  
 as.vector()  
  
print(probabilities)

## min.probability max.probability  
## 1 0.00001269135 0.0002030616

if(probabilities$min.probability < 0){stop("Minimum probability of being sampled is smaller than 0. Review sampling probabilities before computing base weights.")}else if(probabilities$max.probability > 1){stop("Maximum probability of being sampled is larger than 1. Review sampling probabilities before computing base weights.")}  
  
rm(probabilities)

We see that there are actually 16 unique sampling probabilities computed in the dataset.

unique(sort(data$prob))

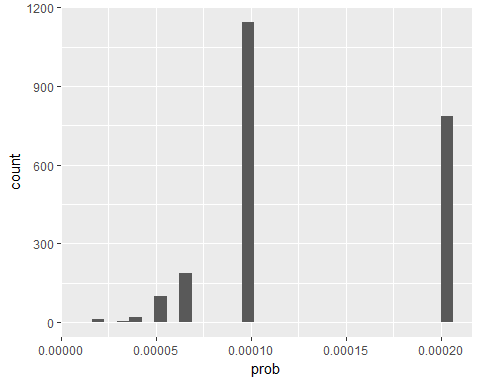
## [1] 0.00001269135 0.00001682761 0.00001692180 0.00001846015 0.00002030616  
## [6] 0.00002256240 0.00002538271 0.00002900881 0.00003384361 0.00004061233  
## [11] 0.00005076541 0.00006768721 0.00010153082 0.00013681033 0.00020306164

The vast majority of respondents had probability of 0.0001015 or 0.00020306 (i.e. one in 9849 and one in 4925). We see a minority of around 15% of observations with smaller probabilities. These probabilities might seem very small. This is because the whole population is very large and the survey only sampled a small part of it.

table(round(data$prob\*100,6))

##   
## 0.001269 0.001683 0.001692 0.001846 0.002031 0.002256 0.002538 0.002901   
## 1 1 1 2 1 6 1 1   
## 0.003384 0.004061 0.005077 0.006769 0.010153 0.013681 0.020306   
## 6 22 101 188 1144 2 787

ggplot(data, aes(x = prob)) +  
 geom\_histogram()



The sampling probability of respondents seems to be related to the type of dwelling of respondents and the number of people in their household. We would have expected this as some types of dwellings might tend to be linked to a single address. For other (smaller) types there might usually be many dwellings sharing the same address[[2]](#footnote-44). Something similar would happen for size of household. Individuals in large households would have smaller probabilities of being sampled than individuals who are the only person in the household.

These differences in sampling probabilities across dweling type and household size show that, without any kind of adjustment, our sample would over-represent people living in certain types of dweling (e.g. 'Multi-unit house, flat') and people living in small households. If respondents from living in different types of dwellings and household sizes had differences in our *'Y'* variables (e.g. smoked more or drank more alcohol) then our estimates from the survey sample would be biased.

data %>%  
 filter(!is.na(prob)) %>%  
 group\_by(type) %>%  
 summarise(n = n(),  
 mean.prob.percentage = mean(prob, na.rm = T)\*100) %>%  
 arrange(desc(mean.prob.percentage))

## # A tibble: 11 x 3  
## type n mean.prob.percentage  
## <fctr> <int> <dbl>  
## 1 NA 1 0.02030616  
## 2 Multi-unit: Sheltered/retirement housing 21 0.01869456  
## 3 Multi-unit house, flat 346 0.01617988  
## 4 Other 15 0.01601931  
## 5 House-trailer or boat 3 0.01353744  
## 6 Single unit: Terraced house 591 0.01338198  
## 7 Only housing unit in building with other purpose 7 0.01305396  
## 8 Single unit: Semi-detached house 670 0.01213712  
## 9 Single unit: Detached house 582 0.01173768  
## 10 Farm 24 0.01149272  
## 11 Multi-unit: Student apartments, rooms 4 0.00412469

data %<>%  
 mutate(hhmmb.factor = as.factor(hhmmb) %>% fct\_recode(`+5` = "6",  
 `+5` = "7",  
 `+5` = "8"))  
  
data %>%  
 filter(!is.na(prob)) %>%  
 filter(!is.na(hhmmb.factor)) %>%  
 group\_by(hhmmb.factor) %>%  
 summarise(n = n(),  
 mean.prob.percentage = mean(prob, na.rm = T)\*100) %>%  
 arrange(desc(mean.prob.percentage))

## # A tibble: 6 x 3  
## hhmmb.factor n mean.prob.percentage  
## <fctr> <int> <dbl>  
## 1 1 745 0.019260215  
## 2 2 764 0.010841915  
## 3 3 301 0.009803027  
## 4 4 303 0.009026823  
## 5 5 118 0.008328013  
## 6 +5 33 0.007486616

To solve these differences in sampling probabilities we have to compute **design weights** (sometimes also called **base weights**. The design weights are equal to the inverse of the probability of inclusion to the sample. Therefore, the design weight (*d0*) of a respondent (*i*) will be equal to: where is the probability of that unit of being included in the sampling.

Here we compute the design weight from the probability given in the ESS database.

data %<>%  
 mutate(base.weight = 1/prob)  
  
data %>%  
 select(prob, base.weight) %>% head(10)

## prob base.weight  
## 1 0.00020306164 4924.613  
## 2 0.00020306164 4924.613  
## 3 0.00020306164 4924.613  
## 4 0.00005076541 19698.452  
## 5 0.00010153082 9849.226  
## 6 0.00010153082 9849.226  
## 7 0.00010153082 9849.226  
## 8 0.00020306164 4924.613  
## 9 0.00010153082 9849.226  
## 10 0.00010153082 9849.226

A simple interpretation of design weights it 'the number of units in our population that each unit in our sample represents'. There is a simple but important test that we should perform after computing design weights. **The sum of all design weights should be equal to the total number of units in our population**. The ESS dataset for UK only included sampling probabilities for respondents (i.e. sampled units that responded to the survey!) but they did not include sampling probabilities of non-respondents. We can guess that this is because sampling probability depends on information that is obtained from the interview (i.e. number of people in household, number of households in dwelling, number of dwellings in adress, etc.). Not knowing the sampling probability for some sampled units is not an optimal situation.

The sum of our computed weights in the ESS dataset with 2,265 respondents equals 21,338,524. Doing a very simple Extrapolation to include the 3,335 non-respondents would give us a sum of weights equal to 52,757,499. This last figure would be much closer to the total UK population over 15.

It is a common practice for many researchers to scale the weights so that their sum equals the sample size (instead of the population size). Scaled weights would equally adjust for differences in sampling probabilities.

Here we compute our scaled design weights and we compare them with the ones given in the ESS dataset. We see that our weights scaled (*base.weigth.scaled*) are almost equal to those computed in the ESS dataset (*dweigth*). The small differences are probably due to rounding error.

data %>%  
 filter(!is.na(base.weight)) %>%   
 select(idno, base.weight) %>%  
 mutate(base.weight.scaled = base.weight/sum(base.weight, na.rm = T)\*nrow(data[!is.na(data$prob),])) %>%  
 left\_join(original.weights %>% select(idno, dweight),  
 by = "idno") %>% head(10)

## idno base.weight base.weight.scaled dweight  
## 1 100000003 4924.613 0.5224974 0.526796  
## 2 100000005 4924.613 0.5224974 0.526796  
## 3 100000008 4924.613 0.5224974 0.526796  
## 4 100000009 19698.452 2.0899897 2.107184  
## 5 100000010 9849.226 1.0449949 1.053592  
## 6 100000012 9849.226 1.0449949 1.053592  
## 7 100000015 9849.226 1.0449949 1.053592  
## 8 100000016 4924.613 0.5224974 0.526796  
## 9 100000017 9849.226 1.0449949 1.053592  
## 10 100000020 9849.226 1.0449949 1.053592

data %<>%  
 mutate(base.weight.scaled = base.weight/sum(base.weight, na.rm = T)\*nrow(data[!is.na(data$prob),]))

As we mentioned before, design weights should sum up to the entire population from which the sample is drawn or to the total number of respondents if scaled as they did in the ESS. In this example both sums should equal 2,265.

data %>%  
 left\_join(original.weights %>% select(idno, dweight),  
 by = "idno") %>%  
 summarise(sum.all.base.weights.scaled = sum(base.weight.scaled, na.rm = T) %>% round(0),  
 sum.all.design.weights.ess = sum(dweight, na.rm = T) %>% round(0))

## sum.all.base.weights.scaled sum.all.design.weights.ess  
## 1 2264 2264

## Step 2: Non-response weights

### What are of non-response weights?

The second basic step in weighting a survey is accounting for differences in the propensity to respond. Imagine a situation in which a profile of sampled units (e.g. lower income people) had higher propensity to respond than another profile (e.g. people with higher incomes). Imagine as well that the characteristics of both profiles were associated to our *'Y'* variables, here alcohol and cigarretes consumption (e.g. people with lower income smoking and drinking more than people with higher income). As we then would have a larger proportion of lower income/smokers in our sampleour analyses would be biased.

If we compute the sample ourselves, we can also compute the probability of a unit of being sampled. Computing the probability of replying to the survey is, however, slightly more challenging. We can not direclty observe the probability of a unit of replying to the survey, therefore we need to estimate it. This is done using information which we know for both respondent and non-respondent units. Here it is useful to think about the probability of response as a latent variable (i.e. not directly observable).

There are two main ways of using this information. The first one would be creating cells from variable intersections (i.e. sampled units 15-24 & countryside; 15-24 & cities; 25-34 & countryside, etc.) and then calculate the probability of response in each cell. The second method is to estimate the probability of response by modelling it.

The first approach has the advantage of being more simple. However, computing a large number of cells from crossing variable could most probably lead to having empty cells or cells with a very small number of sampled units. The probability estimated for these cells with smaller number of units could be far from the real one. We could apply it if we thought that the probability of responding can actually be explained by the variable intersections we used.

The second approach (explained in ['Propensity adjustment' section](#propensity_adjustment) below) relies on the modelling skills of the researcher computing the weights. In order to estimate the proability of response we need to predict it. 'Predictive modelling' is slightly different than the usual 'inference modelling'. Here we are not interested in understanding the causes of differences in response propensities. We are just interested in predicting these. Therefore, we should take a 'data driven' approach that differs from those we usually see in social sciences. For an excellent introduction to 'predictive statistics' you can check the free book ['An Introduction to Statistical Learning' by James et al.](http://www-bcf.usc.edu/~gareth/ISL/).

This is not a guide on 'predictive modelling'. However, it might be worth it to very briefly explain the basic principle behind it. We should try to build a model which is able to estimate the probability using only 'relevant' information and excluding 'noise' from the model. Therefore, a predictive model should be fitting the observed data well enough but at the same time not too specific to it.

For this specific case of non-response weighting, we are especially interested in using propensity predictors which are related to both the response propensity and our dependent variables.

Here we will use the paradata information to model the probability of response. The variables describing the type of house and the immediate vicinity around houses have a relatively small number of missing values (~ 8%) these missings seem to be related (i.e. all variables missing for certain units). Those sampled units that have missing values on all these paradata variables are always non-respondents. It would be useful to know why this pattern exists. We would guess that these are sampled units which interviewers could not reach for some reason. For this analysis, we will ignore this fact and not use these units with missing values in ALL paradata variables. This should only result in a linear transformation of the estimated probabilities of response. Moreover, we can later adjust our model to 'undo' this transformation.

data.nonresponse.model <- data[vars.paradata] %>% select(idno, interva, type:vandaa)  
  
data.nonresponse.model$all.missing <- NA  
  
data.nonresponse.model$all.missing[is.na(data.nonresponse.model$type) &  
 is.na(data.nonresponse.model$access) &  
 is.na(data.nonresponse.model$physa) &   
 is.na(data.nonresponse.model$littera) &  
 is.na(data.nonresponse.model$vandaa)] <- 1  
  
data.nonresponse.model %<>%  
 filter(is.na(all.missing)) %>%  
 select(-all.missing)  
  
indep.vars.nonresponse <- c("type", "access", "physa", "littera", "vandaa")  
  
data.nonresponse.model[,c("type", "access")] %<>% map(function(x){as.character(x)})  
  
#data.nonresponse.model %>% map(function(x){sum(is.na(x))})  
  
# Missing category for missings in multinomial variables.  
   
for(i in c("type", "access")) {  
 data.nonresponse.model[[i]][is.na(data.nonresponse.model[[i]])] <- "Missing"  
   
}  
  
# Mean imputation for ordinal variables.  
  
for(i in c("physa", "littera", "vandaa")) {  
 data.nonresponse.model[[i]][is.na(data.nonresponse.model[[i]])] <- levels(data.nonresponse.model[[i]])[median(data.nonresponse.model[[i]] %>% as.numeric(), na.rm = T) ]   
   
}  
  
for(i in c("physa", "littera", "vandaa")) {  
 data.nonresponse.model[[i]] <- as.ordered(data.nonresponse.model[[i]])  
  
   
}  
  
data.nonresponse.model %<>%  
 mutate(response = as.numeric(as.numeric(interva) == 1))  
  
data %<>%  
 mutate(response = as.numeric(as.numeric(interva) == 1))

### Estimating response propensities using logistic regression

Valliant et al (2013) recommend estimating the response propensities and then grouping them in classes. The grouping step should avoid extreme weights. One way of estimating the response propensities is using logistic regression. This logistic regression should be unweighted. Later in this section we will try other ways of computing estimates of response propensities.

In order to do a logistic regression in R, we need to specify the dependent variable (response) and predictors (type, access, physa, littera and vandaa) in a formula. Then we input the formula and the dataset into the *'glm'* function with a *'family = binomial'* argument.

The following output shows the first six estimated response propensities of our dataset.

formula <- as.formula("response ~ type + access + physa + littera + vandaa")  
  
options(na.action = 'na.pass')  
  
x.matrix <- model.matrix(formula, data = data.nonresponse.model)[, -1]  
  
log.reg.m <- glm(formula,   
 data = data.nonresponse.model,  
 family = "binomial")  
  
coef.response.log <- coef(log.reg.m)  
  
predicted.log <- log.reg.m$fitted.values  
  
data.nonresponse.model$predicted.log <- predicted.log  
  
predicted.log %>% head()

## 1 2 3 4 5 6   
## 0.4101303 0.4586384 0.4675598 0.5551654 0.4117369 0.5143249

As explained before, we are now computing our estimates from a subset of sampled units. This does not take into account many non-respondents which had missing values in all predictors of our non-response model. This means that the previous propensities are slightly overestimated. This is because the response rate in our full dataset is 40.4% and in our subset used for non-response adjustment is 44.1%. This is not entirely related to non-response weight adjustment so we will just compute the new estimates with modified intercept in the next chunk and skip the explanation on how to perform the trasformation. For an excellent description on how it's done readers can check Stanford University's online course on [SP Statistical Learning](https://lagunita.stanford.edu/courses/HumanitiesSciences/StatLearning/Winter2016/info).

Below the next chunk of code you can see the first six transformed estimated response propensities of our dataset.

resp.pop <- mean(data$response)  
   
resp.subgroup <- mean(data.nonresponse.model$response)  
  
log.reg.m$coefficients[1] <- coef(log.reg.m)[[1]] + log(resp.pop/(1-resp.pop) - log(resp.subgroup/(1-resp.subgroup)) )  
  
rm(resp.pop, resp.subgroup)  
  
predicted.log.tr <- predict(log.reg.m, data.nonresponse.model, type = "response")  
  
data.nonresponse.model$predicted.log.tr <- predicted.log.tr  
  
predicted.log.tr %>% head()

## 1 2 3 4 5 6   
## 0.3886949 0.4365441 0.4453892 0.5329992 0.3902731 0.4919858

rm(coef.response.log, formula, i)

### Creating non-response adjustment classes

Here we will follow the procedure for creating classes explained in pag. 329 of Valliant et al.(2013). From their text it is not so clear how many classes should be created.

Looking at the distribution of estimated probabilities of response, we observe a large majority of values between 0.4 and 0.5. However, there are several outliers at both ends of the distribution.

As there is not so much dispersion in values in the middle of the distribution, creating classess accoring to quintiles might not be the best way to account for differences in estimated response propensities. However, other methods might create classess which are too specific to outliers. This is kind of a bias-variance trade off. If we fit broad classes which encompass very different estimated probabilities within them, we will be adjusting less and so keeping more bias in our estiamtes. If we create tight classes capturing these outliers, then we will have large differences in weights and so more variance in our estimates.

predicted.log.tr %>% quantile(probs = seq(0,1,0.05)) %>% round(4)

## 0% 5% 10% 15% 20% 25% 30% 35% 40% 45%   
## 0.0124 0.2878 0.3457 0.3638 0.3887 0.3903 0.4022 0.4038 0.4211 0.4227   
## 50% 55% 60% 65% 70% 75% 80% 85% 90% 95%   
## 0.4227 0.4365 0.4365 0.4365 0.4370 0.4370 0.4626 0.5061 0.5065 0.5065   
## 100%   
## 0.7526

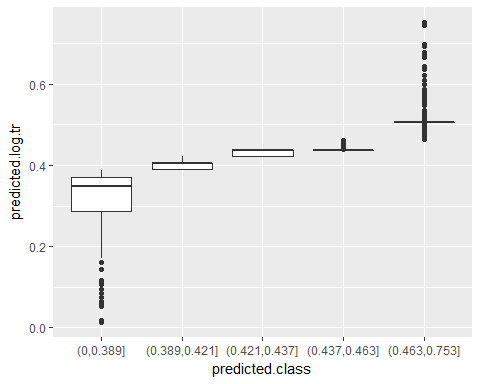
These are the 5 classes created using 20th, 40th, 60th and 80th quintiles.

data.nonresponse.model$predicted.class <- cut(x = data.nonresponse.model$predicted.log.tr, breaks = c(0,quantile(predicted.log.tr, probs = seq(0,1,0.2))[-1] ))  
  
data.nonresponse.model$predicted.class %>% levels()

## [1] "(0,0.389]" "(0.389,0.421]" "(0.421,0.437]" "(0.437,0.463]"  
## [5] "(0.463,0.753]"

Below there is a summary of the estimated propensities included in each of them (in boxplots). We can see that the first and last groups have more dispersion in propensities. The middle three groups have very little dispersion and are similar between them. In this case they are so similar that we could even consider merging them into one single group.

ggplot(data.nonresponse.model, aes(x = predicted.class, y = predicted.log.tr)) +  
 geom\_boxplot()



To compute the non-response weights, we can use the mean estimated probability of response in each class.

data.nonresponse.model %>%  
 group\_by(predicted.class) %>%  
 summarise(mean.prob = mean(predicted.log.tr) %>% round(4))

## # A tibble: 5 x 2  
## predicted.class mean.prob  
## <fctr> <dbl>  
## 1 (0,0.389] 0.3283  
## 2 (0.389,0.421] 0.4016  
## 3 (0.421,0.437] 0.4310  
## 4 (0.437,0.463] 0.4387  
## 5 (0.463,0.753] 0.5145

And then we can compute the non-response weight as the inverse of the mean probabilities in each class.

data.nonresponse.model %<>% left\_join(  
data.nonresponse.model %>%  
 group\_by(predicted.class) %>%  
 summarise(mean.prob = mean(predicted.log.tr) ), by = "predicted.class")  
  
data.nonresponse.model %<>%  
 mutate(nonresp.weight = round(1/mean.prob, 4) )  
  
data.nonresponse.model$nonresp.weight %>% unique %>% sort

## [1] 1.9435 2.2793 2.3201 2.4900 3.0463

### Testing non-response adjustment classes

After creating the classes, a good practice would be to check for covariate balance. This procedure is explained in pag. 330 of Valliant et al.(2013). Units in the same class should have similar values in covariates. At the same time, we would ideally find differencess between classes. In other words, here we would check if classes are made of homogeneous units (within-class check) and if they really distinguish different profiles of these (between-class check). For the within-group check, we are especially interested in checking if profiles of respondents and non-respondents within each class are similar.

The best way of doing this analysis might be fitting two different regressions for each covariate[[3]](#footnote-53). As an example we will do the balance analysis for the ordinal covariate *physa*. In a real analysis this should be repeated for all covariates.

For the between-class check, we can fit a model only with class predictors. This should show if classes explain differences in covariate variables. As we have an ordinal covariate (*physa*) with four categories (*'Very bad'*, *'Bad'*, *'Satisfactory'*, *'Good'*, *'Very good'*), we will use an ordered logistic regression. An alternative would be to treat the covariate as a continous varaible and use an OLS regression.

The variable *physa* compares the 'Overall physical condition building/house'. The variable is negativelly coded so larger values mean worse physical condition. The coefficients of the ordered logistic regression show that the larger the estimated propensity, the smaller the probability of being in bad physical conditions. Therefore, people in houses with worse physical conditions would have smaller response propensity and so be underrepresented in our sample. From the coefficients below we see that the classes we have created for non-response adjustment somehow explain differences in our analysed covariate. However, ideally we would have a more clear effect of classes on our covariate. We see that the second adjustment class created (that for estimated propensities between 0.41 and 0.443) actually has larger probabilities of having worse physical conditions than the base category (0 to 0.41). At the same time, the third and fourth categories (0.443 to 0.459) and (0.459 to 0.485) have very similar coefficients, which might indicate that they do not really distinguish different classes of sampled units.

formula. <- as.formula("physa ~ predicted.class")  
  
test.between.model <- polr(formula = formula., data = data.nonresponse.model, method = "logistic")  
  
ctable.between <- coef(summary(test.between.model))  
  
p <- pnorm(abs(ctable.between[, "t value"]), lower.tail = FALSE) %>% round(4)  
  
ctable.between <- cbind(ctable.between, "p value" = p)  
  
ctable.between

## Value Std. Error t value p value  
## predicted.class(0.389,0.421] 0.4616699 0.08909791 5.181602 0  
## predicted.class(0.421,0.437] -3.0018358 0.10543331 -28.471419 0  
## predicted.class(0.437,0.463] -2.8379051 0.12184972 -23.290207 0  
## predicted.class(0.463,0.753] -7.1107000 0.14993912 -47.423915 0  
## Very good|Good -5.1316957 0.11232331 -45.686826 0  
## Good|Satisfactory -0.6246051 0.06405379 -9.751260 0  
## Satisfactory|Bad 2.5006755 0.08826855 28.330310 0  
## Bad|Very bad 4.6467655 0.20292616 22.898799 0

For the within-class check, we can extend our model to include interactions between class and response. This will check if, within a non-response class, there are differences between respondents and non-respondents in our covariate *physa* (ideally they wouldn't be and we would not find them).

In this second test (see coefficients below) we see that one of the interactions has a significant coefficient at 5% confidence level. Ideally, all interaction terms would be non-significant, meaning that we do not observe within-group differences between respondents and non-respondents in our covariates. A way of dealing with this situation with one significant coefficient would be to explore other ways of spliting units into classes. However, as explained before, these other categorisations would have drawbacks in terms of inflated variance. Moreover, if we have a large number of classes and covariates, we would expect to find significant coefficients just by chance. Therefore, as long as these unbalances do not occur for most classes and covariates, reporting these unbalances should be enough. In this example, we see that the differences in the physical condition of their houses we observe within each computed class respondents and non-respondents are so small that might be due to sampling error[[4]](#footnote-54).

formula. <- as.formula("physa ~ response + predicted.class + response \* predicted.class")  
  
test.within.model <- polr(formula = formula., data = data.nonresponse.model, method = "logistic")  
  
ctable.within <- coef(summary(test.within.model))  
  
p <- pnorm(abs(ctable.within[, "t value"]), lower.tail = FALSE) %>% round(4)  
  
ctable.within <- cbind(ctable.within, "p value" = p)  
  
ctable.within

## Value Std. Error t value  
## response 0.16629093 0.12952169 1.28388484  
## predicted.class(0.389,0.421] 0.59606463 0.11503347 5.18166279  
## predicted.class(0.421,0.437] -2.93757529 0.12899719 -22.77239763  
## predicted.class(0.437,0.463] -2.70924911 0.15584853 -17.38386062  
## predicted.class(0.463,0.753] -7.13945161 0.19426167 -36.75172547  
## response:predicted.class(0.389,0.421] -0.34119548 0.18198801 -1.87482398  
## response:predicted.class(0.421,0.437] -0.18968461 0.18651462 -1.01699595  
## response:predicted.class(0.437,0.463] -0.32814861 0.22822726 -1.43781515  
## response:predicted.class(0.463,0.753] -0.02271575 0.23907967 -0.09501331  
## Very good|Good -5.07916071 0.12147163 -41.81355457  
## Good|Satisfactory -0.56711343 0.07902913 -7.17600521  
## Satisfactory|Bad 2.56185632 0.10036301 25.52590104  
## Bad|Very bad 4.70784119 0.20850649 22.57887082  
## p value  
## response 0.0996  
## predicted.class(0.389,0.421] 0.0000  
## predicted.class(0.421,0.437] 0.0000  
## predicted.class(0.437,0.463] 0.0000  
## predicted.class(0.463,0.753] 0.0000  
## response:predicted.class(0.389,0.421] 0.0304  
## response:predicted.class(0.421,0.437] 0.1546  
## response:predicted.class(0.437,0.463] 0.0752  
## response:predicted.class(0.463,0.753] 0.4622  
## Very good|Good 0.0000  
## Good|Satisfactory 0.0000  
## Satisfactory|Bad 0.0000  
## Bad|Very bad 0.0000

rm(formula., test.within.model, p, indep.vars.nonresponse, i, test.between.model,  
 ctable.between, ctable.within)

### Propensity adjustment (alternative to adjustment classes)

An alternative to computing class adjustments is to directly use the inverse of the estimated probabilities of response. Adjustment classes act are a way of 'smoothing' predictions, avoiding extreme values and overfit. Therefore, they are based on some kind of 'mistrust' of statistical models. Even if they are a standard procedure in survey methodology and backed by solid literature, they might look a bit naive to researchers coming from other areas (e.g. econometrics, genomics, etc.). Adjusting directly with propensity scores relies more on statistical/modelling skills of researchers. Here we will show a couple of alternative methods. As previously explained, these need to avoid being too specific to our data. Instead, they should try to give information on the distribution that generated this data.

#### Estimating response propensities using Cross-validated LASSO Regression

The first method we will try is a penalised regression, more specifically a LASSO regression. Penalised regressions are very similar to a logistic regression but have the particularity of having 'shrinked' coefficients. This reduces the influence of predictors to avoid overfit. It does this by producing a model which would be better to predict 'unseen' data and is less specific to our data. In other words, it tries to avoid noise from our data set into the model. This is done by tunning the penalty parameters with cross-validation (a resampling method).

The first six estimated propensities are the output of the next code chunk.

formula. <- as.formula("response ~ type + access + physa + littera + vandaa")  
  
options(na.action = 'na.pass')  
  
x.matrix <- model.matrix(formula., data = data.nonresponse.model)[, -1]  
  
glm.cv.model <- cv.glmnet(x.matrix, data.nonresponse.model$response, alpha = 1, family="binomial")  
  
predicted.lasso <- predict(glm.cv.model, x.matrix, s = "lambda.min", type = "response")[,1]  
  
data.nonresponse.model$predicted.lasso <- predicted.lasso  
  
head(predicted.lasso)

## 1 2 3 4 5 6   
## 0.4087345 0.4611603 0.4523603 0.4758645 0.4198371 0.5135011

Comparing the predictions obtained with these two methods (logistic regression and cross-validated lasso) we observe that they all give relatively similar values. They are all centered around the same mean, which is the proportion of respondents in our subset. Lasso regression is the more 'rigid' with a smaller standard deviation in its predictions. Logistic regression is almost always slightly more distant of the mean than the lasso regression. It is important to note that our logistic and lasso regressions included only linear parameters (i.e. no squared coefficients nor interactions). Therefore, these were both rather 'rigid' methods. Below we will see that our prediction of response probabilities is relatively poor. This is most probably because our covariates are far from excellent predictors of response propensities[[5]](#footnote-57). Therefore, it might not be worth the time to try more 'flexible' models.

mean(data.nonresponse.model$response)

## [1] 0.4414344

list.comparison <- list()  
  
list.comparison$head.predicted.vals <- cbind(predicted.log, predicted.lasso) %>% head(10)  
  
list.comparison$means <- c(mean.log.reg = mean(predicted.log),   
 mean.lasso = mean(predicted.lasso))   
  
list.comparison$sd <- c(sd.log.reg = sd(predicted.log),   
 sd.lasso = sd(predicted.lasso))   
  
list.comparison

## $head.predicted.vals  
## predicted.log predicted.lasso  
## 1 0.4101303 0.4087345  
## 2 0.4586384 0.4611603  
## 3 0.4675598 0.4523603  
## 4 0.5551654 0.4758645  
## 5 0.4117369 0.4198371  
## 6 0.5143249 0.5135011  
## 7 0.4259045 0.4268628  
## 8 0.4446954 0.4540173  
## 9 0.3908374 0.4105130  
## 10 0.4254382 0.4268628  
##   
## $means  
## mean.log.reg mean.lasso   
## 0.4414344 0.4414344   
##   
## $sd  
## sd.log.reg sd.lasso   
## 0.07217839 0.05818359

rm(list.comparison)

Comparing the fit of both models (below the next chunk), we see that the logistic regression fits a bit better to the whole sample, with 57.6% of success in classifying sample units as respondents vs non-respondents. However, the difference is almost negligible. Moreover, a 58% success in classifying is a rather poor model (by chance we would already expect to correctly classify 50%).

data.nonresponse.model %<>%  
 mutate(predicted.category.log.reg = (predicted.log > 0.5) %>% as.numeric,  
 predicted.category.lasso = (predicted.lasso > 0.5) %>% as.numeric)  
  
train.correct.logreg <- table(data.nonresponse.model$response, data.nonresponse.model$predicted.category.log.reg) %>% diag() %>% sum()/nrow(data.nonresponse.model)  
  
train.correct.lasso <- table(data.nonresponse.model$response, data.nonresponse.model$predicted.category.lasso) %>% diag() %>% sum()/nrow(data.nonresponse.model)  
  
c(train.correct.logreg = train.correct.logreg, train.correct.lasso = train.correct.lasso)

## train.correct.logreg train.correct.lasso   
## 0.5755213 0.5737673

The two chunks below compute the cross-validated ratio of correctly classified units. We would expect these to be lower than the successfuly classified units for the whole sample. This is because cross-validation is a resampling method. We use it to get an idea of how our estimates (e.g. ratio of correctly classified units) would fit the whole statistical population which includes 'unseen' observations (i.e. not only the sample at hand).

Cross-validated ratio of correctly classified units would vary each time we run the procedure. This will not happen here because we set a seed for random procedures at the beginning or the script. We observe that, for both methods, the cross-validated ratio of correctly classified units is slightly lower than the estimate for our sample. Also, both methods have a very similar predictive capacity.

folds <- createFolds(data.nonresponse.model$response, k = 10, list = TRUE, returnTrain = FALSE)  
  
formula. <- as.formula("response ~ type + access + physa + littera + vandaa")  
  
 success.rate.logreg <- vector()  
  
for(i in 1:length(folds)){  
   
 t <- i  
   
 train.folds <- c(1:10)[-t]  
   
 temp.train.data <- data.nonresponse.model[folds[train.folds] %>% unlist(),]  
   
 temp.test.data <- data.nonresponse.model[folds[t] %>% unlist(),]  
   
 temp.log.reg.m <- glm(formula., data = temp.train.data, family = "binomial")  
   
 temp.predicted <- predict(temp.log.reg.m, temp.test.data, type = "response")  
   
 temp.predicted <- ifelse(temp.predicted > 0.5, 1, 0)  
   
 success.rate.logreg <- c(success.rate.logreg, table(temp.predicted, temp.test.data$response) %>% diag() %>% sum()/nrow(temp.test.data) )  
   
   
}  
  
 cv.success.log.reg <- success.rate.logreg %>% sum()/length(success.rate.logreg)  
  
 rm(train.folds, temp.train.data, temp.test.data, temp.log.reg.m, temp.predicted)

success.rate.glmnet <- vector()  
  
for(i in 1:length(folds)){  
   
 t <- i  
   
 train.folds <- c(1:10)[-t]  
   
 temp.train.data <- x.matrix[folds[train.folds] %>% unlist(),]  
   
 temp.train.y <- data.nonresponse.model[folds[train.folds] %>% unlist(),"response"]  
  
 temp.test.data <- x.matrix[folds[t] %>% unlist(),]  
   
 temp.test.y <- data.nonresponse.model[folds[t] %>% unlist(),"response"]  
  
 temp.glmnet.m <- glmnet(x = temp.train.data, y = temp.train.y, family = "binomial")  
   
 temp.predicted <- predict(temp.glmnet.m, temp.test.data, type = "response", s = glm.cv.model$lambda.min)  
   
 temp.predicted <- ifelse(temp.predicted > 0.5, 1, 0)  
   
 success.rate.glmnet <- c(success.rate.glmnet, table(temp.predicted, temp.test.y) %>% diag() %>% sum()/nrow(temp.test.data) )  
   
}  
  
 cv.success.glmnet <- success.rate.glmnet %>% sum()/length(success.rate.glmnet)  
  
 rm(t, train.folds, temp.train.data, temp.train.y, temp.test.data,  
 tempt.test.y, temp.glmnet.m, temp.predicted)   
   
 c(cv.success.log.reg = cv.success.log.reg, cv.success.glmnet = cv.success.glmnet)

## cv.success.log.reg cv.success.glmnet   
## 0.5718210 0.5690923

If we wanted to do an adjustment directly using the estimated propensity scores and without computing adjustment classes, we would just use the inverse of the estimated propensities as the non-response weights. It would be a good idea to use the Lasso predictions as these are less sparce than those from the logistic regression. Therefore, we might have less variance inflation with (virtually) the same expected reduction in bias.

If we had no information about population estimates, we would end the weighting procedure here. The 'final weight' would be the multiplicaiton of both base scaled weight and the scaled non-response weight. Here we will call this new weights 'final weights' although we still have to perform adjustments to them and so will not really be 'final'.

Before going to the next step we will include the computed non-response weights using adjustment classes to the main *'data'* dataframe object. Then we will drop all non-respondents as we are not going to use them any more in the next steps of our analysis. After that, we will scale the non-response weights to the sample of respondents and multiply the (scaled) design weights and the (scaled) non-response weights.

data %<>%  
 left\_join(data.nonresponse.model %>% select(idno, nonresp.weight),  
 by = "idno")  
  
data %<>%  
 filter(response == 1, !is.na(base.weight.scaled)) %>%  
 mutate(nonresp.weight.scaled = nonresp.weight/mean(nonresp.weight),  
 final.weight = base.weight.scaled \* nonresp.weight.scaled)  
  
rm(x.matrix, cv.success.glmnet, cv.success.log.reg, folds, formula.,  
 glm.cv.model, i, log.reg.m, predicted.lasso,predicted.log.tr, success.rate.glmnet, success.rate.logreg, train.correct.lasso, train.correct.logreg)

## Step 3: Use of auxiliary data for weight calibration

In certain cases we might have information about our statistical population (e.g. census counts or proportions from official statistics). We can then use these to 'correct' our weights. This adjustment is called calibration and consists on finding a new set of weights that are as near as possible the input (*'final'*) weights but reproduce the population information exactly. Valliant et al (2013) explain that using the previous *'final'* weights as input for the calibration step allows us to 'borrow' good estiamtion properties from those.

An important difference between this step and the one on non-response adjustment is that non-response adjustment requieres having information for both sampled respondents and non-respondents. Calibration only requieres information for respondents and population in general.

Here we will calibrate weights using a 'raking' procedure (explained in Valliant el al 2013 page 358). Unlike other calibration methods, 'raking' does not requiere information on cross-classification categories but just marginal population counts. In other words, we do not need the information from crossing several variables (although we can use it if available). As explained by Lumley (2010, page 139), the process is very much iterative. It involves post-stratifying on each set of variables in turn, and repeating the process until the weights stop changing.

The 7th ESS used cross-classifications of age group and gender plus region (separately) to calibrate UK data. Here we will try to reproduce their calibration. For more information on ESS post-stratification weights see their document: [Documentation of ESS Post-Stratification Weights](http://www.europeansocialsurvey.org/docs/methodology/ESS_post_stratification_weights_documentation.pdf)

First we will compute the interaction between gender and age with the categories used for calibration in the ESS.

data %<>%  
 mutate(agea\_rec = cut(agea %>% as.numeric(), breaks = c(14, 34, 54, 99))) %>%  
 unite(col = gender\_age\_rec, gndr, agea\_rec, remove = F) %>%  
 mutate(gender\_age\_rec = replace(x = gender\_age\_rec,   
 list = gender\_age\_rec %in% c("Female\_NA", "Male\_NA"),  
 values = NA ))  
  
data %<>%  
 filter(!is.na(gender\_age\_rec), !is.na(region)) # see footnote in text below.

The total number of weighted units in each of the categories we will use for calibration can be seen in tables below[[6]](#footnote-60).

data %>%  
 group\_by(gender\_age\_rec) %>%  
 summarise(total\_n\_sample = sum(final.weight))

## # A tibble: 6 x 2  
## gender\_age\_rec total\_n\_sample  
## <chr> <dbl>  
## 1 Female\_(14,34] 278.3145  
## 2 Female\_(34,54] 441.9940  
## 3 Female\_(54,99] 465.4844  
## 4 Male\_(14,34] 256.5388  
## 5 Male\_(34,54] 357.8829  
## 6 Male\_(54,99] 426.7719

data %>%  
 group\_by(region) %>%  
 summarise(total\_n\_sample = sum(final.weight))

## # A tibble: 12 x 2  
## region total\_n\_sample  
## <fctr> <dbl>  
## 1 North East (England) 108.7733  
## 2 North West (England) 255.8543  
## 3 Yorkshire and the Humber 165.7948  
## 4 East Midlands (England) 175.9855  
## 5 West Midlands (England) 177.8547  
## 6 East of England 182.0182  
## 7 London 239.2660  
## 8 South East (England) 320.6673  
## 9 South West (England) 187.9519  
## 10 Wales 127.3771  
## 11 Scotland 222.2071  
## 12 Northern Ireland 63.2364

Now we import Eurostat data[[7]](#footnote-61) and recode it into ESS adjustment/post-stratification categories.

The data used in this guide are hosted in the [author's github page](https://github.com/JosepER/PPMI_how_to_weight_a_survey/tree/master/data/Eurostat) and the raw version can be checked in the following links:

* [Age and gender](https://raw.githubusercontent.com/JosepER/PPMI_how_to_weight_a_survey/master/data/Eurostat/Agebygender_demo_pjangroup.csv).
* [Age by NUTS 2 regions](https://raw.githubusercontent.com/JosepER/PPMI_how_to_weight_a_survey/master/data/Eurostat/Agebygender_demo_pjangroup.csv).

age.gender.eurostat <- read.csv(text=getURL("https://raw.githubusercontent.com/JosepER/PPMI\_how\_to\_weight\_a\_survey/master/data/Eurostat/Agebygender\_demo\_pjangroup.csv"), header=T)  
  
age.gender.eurostat %<>%  
 spread(key = Age, value = Value)  
  
age.gender.eurostat %<>%  
 mutate(`15to34` = `From 15 to 19 years` + `From 20 to 24 years` + `From 25 to 29 years` +  
 `From 30 to 34 years`,  
 `35to54` = `From 35 to 39 years` + `From 40 to 44 years` + `From 45 to 49 years` +   
 `From 50 to 54 years`,  
 `55to99` = `From 55 to 59 years` + `From 60 to 64 years` + `From 65 to 69 years` +  
 `From 70 to 74 years` + `75 years or over`) %>%  
 select(SEX, `15to34`:`55to99`)

region.eurostat <- read.csv(text=getURL("https://raw.githubusercontent.com/JosepER/PPMI\_how\_to\_weight\_a\_survey/master/data/Eurostat/Nuts2byage.csv"), header=T, stringsAsFactors = F)  
   
region.eurostat %<>%  
 gather(key = age, value = population, -Country)   
  
region.eurostat %<>%  
 group\_by(Country) %>%  
 summarise(pop\_sum = sum(population) )

We will now scale the Eurostat data to our sample size. The idea is to obtain the weights that make our sample proportions look like those in Eurostat. For these, we will calculate how many respondents in our sample should pertain to each category if we had Eurostat proportions. We will use our sample size based only on (weighted) completed responses in our post-stratification adjustment variables (age, gender and region).

First we compute the total (weighted) observations in our sample of respondents.

data.calibration <- data  
  
weighted.pop <- sum(data.calibration$final.weight)  
  
weighted.pop

## [1] 2226.987

In the next chunk of code we will scale Eurostat population by age an gender data to the size of our sample. We will do this by dividing Eurostat absolute population numbers by the sum of population in all categories and multiplying it by the previously computed sum of weighted respondents. This will provide us a dataframe (*'age.gender.eurostat'*) with the number of respondents our survey sample should have in each gender and age crossed category if it was to resemble the proportions found in reality (i.e. official statistics).

age.gender.eurostat %<>%  
 gather(key = age, value = population, -SEX) %>%  
 unite(col = gender\_age\_rec, SEX, age)   
  
total.population <- age.gender.eurostat$population %>%   
 sum()  
  
age.gender.eurostat %<>%  
 mutate(Freq = round(population/total.population \* weighted.pop, 0) ) %>%  
 select(-population)  
  
age.gender.eurostat$gender\_age\_rec <- c("Female\_(14,34]", "Male\_(14,34]",  
 "Female\_(34,54]", "Male\_(34,54]",  
 "Female\_(54,99]", "Male\_(54,99]")  
  
age.gender.eurostat

## gender\_age\_rec Freq  
## 1 Female\_(14,34] 352  
## 2 Male\_(14,34] 358  
## 3 Female\_(34,54] 372  
## 4 Male\_(34,54] 364  
## 5 Female\_(54,99] 417  
## 6 Male\_(54,99] 364

rm(total.population)

Next we will do the same for Eurostat's population data on NUTS 2 regions.

total.population <- region.eurostat$pop\_sum %>%  
 sum()  
  
region.eurostat %<>%  
 mutate(Freq = round(pop\_sum/total.population \* weighted.pop, 0) ) %>%  
 select(-pop\_sum)  
  
names(region.eurostat)[[1]] <- "region"   
  
region.eurostat$region[region.eurostat$region == "East Midlands (UK)"] <- "East Midlands (England)"  
region.eurostat$region[region.eurostat$region == "North East (UK)"] <- "North East (England)"  
region.eurostat$region[region.eurostat$region == "North West (UK)"] <- "North West (England)"  
region.eurostat$region[region.eurostat$region == "Northern Ireland (UK)"] <- "Northern Ireland"  
region.eurostat$region[region.eurostat$region == "South East (UK)"] <- "South East (England)"  
region.eurostat$region[region.eurostat$region == "South West (UK)"] <- "South West (England)"  
region.eurostat$region[region.eurostat$region == "West Midlands (UK)"] <- "West Midlands (England)"  
region.eurostat$region[region.eurostat$region == "Yorkshire and The Humber"] <- "Yorkshire and the Humber"

Here we will briefly test that we have the same categories in calibration variables of both survey and Eurostat datasets.

if( identical(region.eurostat$region %>% unique %>% sort, data.calibration$region %>% as.character() %>% unique %>% sort) != T) {  
 stop("Levels in region variable have to be the the same in the calibration and dataset used for population frequencies")  
}  
  
if( identical(age.gender.eurostat$gender\_age\_rec %>% unique %>% sort, data.calibration$gender\_age\_rec %>% as.character() %>% unique %>% sort) != T) {  
 stop("Levels in age by gender categories variable have to be the the same in the calibration and dataset used for population frequencies")  
}

Now we will use the R *'survey'* package (Lumley,T., 2010) to calibrate weights using the raking procedure. We will do this twice. First time we will compute the raked weighs using our *'final.weight'* as an input. These contain information from both the base weights and our adjustment for non-response. In the second computation we will repeat the ESS design and use (only) the design/base weights as an input (variable *'base.weight'*). This will allow us to compare our output weights with those computed by the experts behind the weighting procedure of the 7th ESS.

The R *'survey'* package works in a rather particular way[[8]](#footnote-64). It first requieres to specify the design of the survey with the *'svydesign'* function. This creates an object of an adhoc class 'survey.design' that is passed to further procedures such as weights raking.

The *'svydesign'* function requieres an *'ids'* argument with cluster ids. For the specific case of the 7th ESS in the UK, this would be the postal codes, which are the Primary Sampling Units (PSU). The procedures of the *'survey'* package would then take into account these related responses when computing variances. However, here we will ignore this fact and pretend that all responses where independent of each other (we can do this by passing *~ 0* to the *'ids'* argument). This should not affect the raking procedure. Next, we neet to specify the input weights in the *'weights'* argument. Here we will pass the computed *final weights* and the *design/base* weights respectively to create both survey designs. Last, we need to specify the data with the survey respondses in the *'data'* argument.

In this next chunk of code we will create these two objects which will correspond to the two explained computations of raked weights.

our.svydesign <- svydesign(ids = ~ 0, weights = ~final.weight, data = data.calibration)  
  
ess.svydesign <- svydesign(ids = ~ 0, weights = ~base.weight, data = data.calibration)

The raking procedure can be done with the *'rake'* function form the *'survey'* package. This function requieres to pass the previously computed survey design object as its first object (*'design'*). The second argument (*'sample.margins'*) is a list of formulas describing which variables are going to be used for calibration. Here we will pass a list with two formulas. The first one using the *'region'* variable and the second one the *'gender\_age\_rec'* variable which corresponds to the interaction/crosstabulation of gender and age categories. The third argument (*'population.margins'*) are the population counts for our calibration variables. Here we will pass our dataframes with the number of people that should be in each region/gender and age category if our sample followed population proportions.

our.raked <- rake(our.svydesign, sample.margins = list(~region, ~gender\_age\_rec),   
 population = list(region.eurostat, age.gender.eurostat))  
  
ess.raked <- rake(ess.svydesign, sample.margins = list(~region, ~gender\_age\_rec),   
 population = list(region.eurostat, age.gender.eurostat))

Then we collect the weights from the *'our.raked'* and *'ess.raked'* objects we just computed and we add them to our dataset for this section (the *'data.calibration'* dataframe).

raked.weight <- our.raked$postStrata[[1]][[1]] %>% attributes() %>% .[["weights"]]  
  
ess.raked.weight <- ess.raked$postStrata[[1]][[1]] %>% attributes() %>% .[["weights"]]  
  
data.calibration$ess.raked.weight <- ess.raked.weight  
  
data.calibration$raked.weight <- raked.weight  
  
rm(raked.weight, ess.raked.weight)

Next we compare the frequencies of weighted observations in our sample with those we would obtain if our dataset had the same shares of age, gender and region categories as official data (i.e. the data inputed to our calibration procedure). If our calibration is successful, the frequencies should be almost the same. Here we will do this test using our calibration procedure which included design and non-response weights as an input (i.e. calibration object *'raked.weight'*). The following chunk of code shows the results for age and gender categories.

left\_join(  
data.calibration %>%  
 group\_by(gender\_age\_rec) %>%  
 summarise(calibrated.sample.pop = sum(raked.weight) %>% round(1)),  
age.gender.eurostat)

## # A tibble: 6 x 3  
## gender\_age\_rec calibrated.sample.pop Freq  
## <chr> <dbl> <dbl>  
## 1 Female\_(14,34] 351.7 352  
## 2 Female\_(34,54] 371.7 372  
## 3 Female\_(54,99] 416.6 417  
## 4 Male\_(14,34] 357.7 358  
## 5 Male\_(34,54] 363.7 364  
## 6 Male\_(54,99] 363.7 364

Here we show the compared frequencies for regions. Both comparisons allow us to see that calibration performed as expected and weighted frequencies reflect population proportions from official statistics.

left\_join(  
data.calibration %>%  
 group\_by(region) %>%  
 summarise(calibrated.sample.pop = sum(raked.weight) %>% round(1)),  
region.eurostat)

## # A tibble: 12 x 3  
## region calibrated.sample.pop Freq  
## <chr> <dbl> <dbl>  
## 1 North East (England) 85 85  
## 2 North West (England) 235 235  
## 3 Yorkshire and the Humber 189 189  
## 4 East Midlands (England) 169 169  
## 5 West Midlands (England) 194 194  
## 6 East of England 220 220  
## 7 London 286 286  
## 8 South East (England) 312 312  
## 9 South West (England) 183 183  
## 10 Wales 114 114  
## 11 Scotland 176 176  
## 12 Northern Ireland 62 62

Now we can compare our computed final calibrated weights with those computed by the experts in charge of the weighting procedure of the 7th ESS. We will first join the calibrated weights with our original dataset of respondents.

data %<>%  
 left\_join(data.calibration %>% select(idno, raked.weight, ess.raked.weight), by = "idno")  
  
rm(data.calibration, age.gender.eurostat, region.eurostat)

Here we compare our raked weight using the same methodology as they used in the 7th ESS (*'ess.raked.weight'*) and the weight included in the 7th ESS dataset (*'pspwght'*). Looking at the first 15 observations we see that they are, in most cases, very close. This means that we successfully reproduced the weights computed in the 7th ESS!

For our estimations, we could also use the calibrated/raked weights we computed ourselves using the additional input of non-response weights (variable *'raked.weight'*). Using this input, our raking procedure would also 'borrow' good estimation properties from the non-response adjustment[[9]](#footnote-65).

left\_join(  
data %>% select(idno, raked.weight, ess.raked.weight),  
original.weights %>% select(idno, pspwght)  
) %>% head(15)

## idno raked.weight ess.raked.weight pspwght  
## 1 100000003 0.6033962 0.4567495 0.4674183  
## 2 100000005 0.3324131 0.3403316 0.3895551  
## 3 100000008 0.4118283 0.4077717 0.4023832  
## 4 100000009 1.7442021 2.1873476 2.1463098  
## 5 100000010 1.0769384 1.0082670 1.0230092  
## 6 100000012 1.0538387 1.2975135 1.3370290  
## 7 100000015 1.5882982 1.6647140 1.6514970  
## 8 100000016 0.5202722 0.5131737 0.4886227  
## 9 100000017 2.0233932 1.5814927 1.5864217  
## 10 100000020 0.9267053 0.8367614 0.8877654  
## 11 100000022 0.8641030 1.0493359 1.0878961  
## 12 100000023 2.3115622 2.3722391 2.3796326  
## 13 100000025 0.5406270 0.4424130 0.4768728  
## 14 100000030 0.9304049 1.0735789 1.0272274  
## 15 100000031 1.2475568 0.9753477 0.9355472

Here we try to compute a mesure of distance of our computed raked weights to those included in the original 7th ESS datafile. We can do this by calculating the sum of squared differences. As we would expect, we find that those computed using the ESS methodology tend to fall much closer to the ESS original weights than the weights which also include input from non-response estimations. It is important to stress again that this does not mean that closer weights are better than those including properties from non-response estimations.

left\_join(  
data %>% select(idno, raked.weight, ess.raked.weight),  
original.weights %>% select(idno, pspwght)  
) %>%  
 mutate(diff.myraked = pspwght-raked.weight,  
 diff.essraked = pspwght-ess.raked.weight) %>%  
 summarise(ssdiff.myraked = sum(diff.myraked ^ 2),  
 ssdiff.essraked = sum(diff.essraked ^ 2))

## ssdiff.myraked ssdiff.essraked  
## 1 107.686 49.53199

## Step 4: Analysis of weight variability

In progress!

## Computing estimates

In progress!

## Note for non-probability samples

1. In 'real' projects where we do the sampling ourselves we would have the sampling probability of both respondents and non-respondents. This example shows us that it is enough to know the probablity of inclsion of respondents. [↑](#footnote-ref-42)
2. As we explained before, this is relevant because sampling allocation inside Primary Sampling Units (postcode sectors) was proportional to the number of adresses in each of these. Therefore, if an address contains a large number of dwellings, each dwelling will have a smaller probability of being sampled than a dwelling that is the only one in an address. [↑](#footnote-ref-44)
3. This procedure slightly differs from that recommended in Valliant et al.(2013), where they suggest to fit only one regression per covariate. Interpreting the lower order coefficients from a regression with interactions between class predictors and a response indicator might not be interpreted as the authors from Valliant et al.(2013) argue. [↑](#footnote-ref-53)
4. With the exception of differences in class (0.389,0.421], which seem too large to be caused just by sampling error. [↑](#footnote-ref-54)
5. Although it could also be the case that we are missing to explore some important interaction between predictors [↑](#footnote-ref-57)
6. There is a small problem with 20 respondents for which we do not have information about their age. In order to keep thing simple this example we will ignore this issue. [↑](#footnote-ref-60)
7. Eurostat data corresponds to tables and which give information on population by age and gender (*'demo\_pjangroup'*) and age and NUTS2 region (*'demo\_r\_d2jan'*) on the 1st of January of 2014. Last updated on the 29th of April 2017. [↑](#footnote-ref-61)
8. It has its own requierements in terms of coding 'grammar'. This is may be because it was programmed in 2003, almost 15 years ago! Hopefully other packages such as *'srvyr'*, which at the time of writing this guide is still in version 0.2.1, will be able to take the content of the *'survey'* package and adapt it to current scripting [↑](#footnote-ref-64)
9. This idea of calibration weights 'borrowing' properties from input weights comes from Valliant et al. (2013, pag 231) [↑](#footnote-ref-65)