openSIMD

Maike Waldmann & Roman Popat

last compiled: 2017-05-18

Contents

Chapter 1

Preface

The Scottish Index of Multiple Deprivation (SIMD) is the Scottish Government's official tool for finding the most deprived areas in Scotland. SIMD is used by government, councils, charities and communities as evidence to help target their work to those areas that need it most.

SIMD is best known for how it ranks each small area in Scotland by how deprived it is. But in addition to the rankings, all indicator datasets that go into SIMD are also published on a small area level. This wealth of granular data provides detailed information about the underlying issues in deprived areas.

openSIMD opens up SIMD further by making the code used to calculate SIMD available to anyone who wants to understand exactly how SIMD indicators are combined, or replicate the method for similar measures.

SIMD is updated around every three years, and up until the latest edition, SIMD16, SIMDs have been calculated using the statistical package SAS. Through the **openSIMD** project, we have translated the calculation steps into R. The results of both calculations differ slightly due to small differences in how each program runs a statistical procedure called factor analysis. SIMD16 as published on the SIMD website is the official version.

Future updates of SIMD will be calculated using R, and the **openSIMD** R code will produce SIMD ranks that are identical with the official SIMD ranks published by the Scottish Government.

The move from SAS to R through the **openSIMD** project was made possible by a collaboration between The Data Lab and The Scottish Government.

Chapter 2

Introduction

This is a step-by-step guide to the **openSIMD** R code, which creates the Scottish Index of Multiple Deprivation (SIMD), starting from the indicators.

SIMD is made up of over 30 indicators which are grouped into seven domains of deprivation. Each domain summarises one aspect of deprivation by combining some of the indicators and using the resulting domain scores to rank each area in Scotland. The seven domain rankings are then combined into an overall, multiple-deprivation SIMD ranking.

To run the **openSIMD** R code, you need two datasets, SIMD indicator data, and SIMD domain rankings. Both datasets are included here, but can also be downloaded from the SIMD website.

For four out of the seven domains, you will use the SIMD indicator data. For the remaining three domains, you will use the domain rankings instead. The reason for this is that in these three domains, the individual indicators are not made publicly available to ensure that any individual's privacy is protected. For the same reason, the published domain scores are rounded. The domain rankings on the other hand were calculated using unrounded data and are therefore more precise.

After running the **openSIMD** R code, you will have created two csv documents which contain the domain rankings and the overall SIMD ranking for each small area (called data zone) in Scotland.

We tested this code with SIMD16 data, and you will need to adapt it for other versions of SIMD.

The rankings you get when using the **openSIMD** code with the published SIMD16 data slightly differ from the published, offical SIMD16 rankings, see my note in the Preface. The overall SIMD rankings differ by up to 18 ranks. More detail about this can be found in the Quality Assurance section. SIMD16 as published on the SIMD website is the official version.

We will now walk you through the R code. In the Calculating Domains section, we will explain the code used to calculate the individual domain scores. In the section on Calculating SIMD, we will go through the steps necessary for combining the domains into the overall SIMD. In the Functions section, we will introduce the **openSIMD** utility functions used throughout the code for those calculation steps that came up repeatedly. And finally, we'll show how well the R code replicates the SAS code in the Quality Assurance section.

For any questions about the SIMD methodology have a look at the SIMD Technical Notes on the SIMD website, and feel free to contact the SIMD team at simd@gov.scot.

Chapter 3

Calculating Domains

In this section, I will explain the code used to calculate the individual domain scores.

3.1 Setting up

Here is a complete list of the files in `openSIMD_analysis` that we need for **openSIMD**:

- `scripts/calculations/domains.R` to calculate the domains
- `scripts/calculations/openSIMD.R` to calculate SIMD
- `scripts/utils/helpers.R`, some utility functions (see Functions for documentation)
- `data/SIMD16 indicator data.xlsx`
- `data/SIMD16 ranks and domain ranks.xlsx`.

If you use RStudio you can open `openSIMD_analysis.Rproj` to open the project.

Starting in `domains.R`, the first things to do are:

- load a few packages
- source the utility functions
- read in the data

You will need to make sure that the file paths in your code correspond to where your files are. If you are using RStudio and you have opened the `openSIMD_analysis.Rproj` then the directories of interest will be `scripts/utils` and `data`.

You may experience an error when reading the .xlsx file files with `read_excel`. If this happens you have two options. Either you can open the file in excel, save it and try again. The file should now read into R without an error. Alternatively you can save the file a .csv file (make sure you select the correct sheet) and then read it into R using the `read.csv` function.

```
library(readxl)
library(dplyr)
source("../openSIMD_analysis/scripts/utils/helpers.R")
d <- read_excel("../openSIMD_analysis/data/SIMD16 indicator data.xlsx", sheet = 3, na = "*")
```

The data here contains the published indicators, in this case from SIMD 2016.

```
str(d)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    6976 obs. of  36 variables:
##  $ Data_Zone                    : chr  "S01006506" "S01006507" "S01006508" "S01006509" ...
##  $ Intermediate_Zone            : chr  "Culter" "Culter" "Culter" "Culter" ...
```

```
##  $ Council_area                  : chr  "Aberdeen City" "Aberdeen City" "Aberdeen City" "Aberdeen Ci
##  $ Total_population              : num  904 830 694 573 676 ...
##  $ Working_age_population_revised: num  605 491 519 354 414 464 322 354 710 491 ...
##  $ Income_rate                   : num  0.07 0.07 0.05 0.05 0.1 0.03 0.02 0.03 0.01 0.01 ...
##  $ Income_count                  : num  60 60 30 30 70 25 10 20 15 10 ...
##  $ Employment_rate               : num  0.07 0.05 0.03 0.06 0.07 0.03 0.02 0.03 0.01 0.01 ...
##  $ Employment_count              : num  40 25 15 20 30 15 5 10 10 5 ...
##  $ CIF                           : num  60 40 45 65 75 50 35 55 25 35 ...
##  $ ALCOHOL                       : num  103.4 33 54.3 141.9 52.3 ...
##  $ DRUG                          : num  27.6 39.9 32.9 49.2 234.9 ...
##  $ SMR                           : num  67 86 43 88 149 68 77 137 86 61 ...
##  $ DEPRESS                       : num  0.123 0.148 0.127 0.154 0.197 ...
##  $ LBWT                          : num  0 0.0256 0.0556 0.037 0 ...
##  $ EMERG                         : num  80 97.7 70 87.9 105.1 ...
##  $ Attendance                    : num  0.863 0.825 0.877 0.942 0.805 ...
##  $ Attainment                    : num  5.93 5.57 5.8 5.62 5.33 ...
##  $ Noquals                       : num  52.8 95.9 38.6 80.1 77.2 ...
##  $ NEET                          : num  0.03 0.01 0 0.04 0.04 0.03 0.08 0.02 0.03 0.03 ...
##  $ HESA                          : num  0.1304 0.1014 0.1486 0.0816 0.0857 ...
##  $ drive_petrol                  : num  2.42 3.68 3.24 2.51 2.08 ...
##  $ drive_GP                      : num  2.92 4.02 3.6 2.47 2.17 ...
##  $ drive_PO                      : num  1.52 2.74 2.09 1.96 1.67 ...
##  $ drive_primary                 : num  2.03 3.13 2.66 1.44 1.51 ...
##  $ drive_retail                  : num  1.5 2.65 1.88 2.22 1.93 ...
##  $ drive_secondary               : num  10.8 11.5 11.5 10.8 10.6 ...
##  $ PT_GP                         : num  8.44 8.33 7.85 7.43 5.14 ...
##  $ PT_Post                       : num  5.99 7.26 5.83 8.31 6.63 ...
##  $ PT_retail                     : num  5.71 6.79 5.25 8.44 6.62 ...
##  $ crime_count                   : num  8.01 4 4 NA 12.01 ...
##  $ crime_rate                    : num  88.6 48.2 57.7 NA 177.7 ...
##  $ overcrowded_count             : num  87 85 31 42 50 27 27 15 10 29 ...
##  $ nocentralheat_count           : num  10 4 8 6 7 8 9 4 3 1 ...
##  $ overcrowded_rate              : num  0.1021 0.1017 0.0482 0.0724 0.0867 ...
##  $ nocentralheat_rate            : num  0.01174 0.00478 0.01244 0.01034 0.01213 ...
```

3.2 The recipe

For each domain, the process has several steps, many of which are repeated across domains. To make this recipe easier to follow, I have defined some **openSIMD** utility functions in `scripts/utils/helpers.R`, see Functions. The steps of the recipe and the associated functions are as follows:

- Select the indicators (columns) that are relevant to that domain, using the function `dplyr::select`
- Calculate normalised ranks for each indicator, using the function `normalScores`
- Replace missing values, using the function `replaceMissing`
- Derive the factor analysis weights for each indicator, using the function `getFAWeights`
- Combine the normalised ranks and weights to generate the indicator score, using the function `combineWeightsAndNorms`
- Rank the indicator score, using the function `base::rank`.

Not all steps are used for every domain.

3.3 dplyr

Throughout this project, I make use of the tools in the `dplyr` package for data manipulation including the `%>%` notation for forwards piping. I won't introduce these tools but if they are unfamiliar, I recommend reading this article. See the section on chaining for an explanation of the `%>%` pipe.

3.4 An example: Education

As an example of the process, here we will calculate the domain rank for the education domain. In this first chunk of code, I will create the normalised ranks for the education domain as follows:

```
normalised_education <- d %>% # start with the raw data
  select(Attendance, Attainment, Noquals, NEET, HESA) %>% # select relevant columns
  mutate(Attendance = normalScores(Attendance, forwards = FALSE)) %>% # replace each column with its no
  mutate(Attainment = normalScores(Attainment, forwards = FALSE)) %>%
  mutate(Noquals    = normalScores(Noquals, forwards = TRUE)) %>%
  mutate(NEET       = normalScores(NEET, forwards = TRUE)) %>%
  mutate(HESA       = normalScores(HESA, forwards = FALSE)) %>%
  mutate_all(funs(replaceMissing)) # replace missing values
```

```
## Warning in qnorm(rn, mean = 0, sd = 1, lower.tail = forwards): NaNs
## produced

## Warning in qnorm(rn, mean = 0, sd = 1, lower.tail = forwards): NaNs
## produced

## Warning in qnorm(rn, mean = 0, sd = 1, lower.tail = forwards): NaNs
## produced

## Warning in qnorm(rn, mean = 0, sd = 1, lower.tail = forwards): NaNs
## produced
```

Note: you may see a warning here because `normalScores` can generate missing values from missing values. This is fine, these will be replaced with 0, when you call `replaceMissing`.

The only decisions to make are (a) which columns to select using `select` and (b) which orientation to rank (and then normalise) each indicator. The orientation is determined by the `forwards` argument to `normalScores`, see `normalScores` and Variations for further information.

When combining the indicators, we need to apply a different weight to each. The weights are derived through factor analysis of the normalised indicator scores, and the proportional loadings on factor 1 serve as the weightings. We extract the loadings using the `getFAWeights` function as follows:

```
education_weights <- getFAWeights(normalised_education)
```

Now that we have the normalised indicator scores and weights, we can combine them with the utility function `combineWeightsAndNorms`. Each normalised indicator variable is multiplied by its weight derived from factor analysis, as follows:

```
education_score <- combineWeightsAndNorms(education_weights, normalised_education)
```

Finally we rank these weighted scores to generate the domain rank (1 = most deprived).

```
education_rank <- rank(-education_score)
```

3.5 Variations

The remaining domains are calculated in a similar way with some variations. Rather than explaining each domain, I will explain the possible variations.

For the housing domain, the sum of the overcrowding rate and non-central heating rate is ranked.

For the crime, income and employment domains, we simply use the published ranks. The reason for this is that the published indicator data for these domains is rounded, whereas the published domain ranks are based on unrounded data and therefore more precise.

In the education example above, when applying the `normalScores` function, we needed to pay attention to the `forwards` argument to orient the variables (decide whether a high value was good or bad). In the other domains this is not the case and each indicator can take the default value `forwards = TRUE` (high value = deprived). This means we can use the `dplyr::mutate_all()` function instead of mutating each variable independently. In addition, if indicator column names have something in common, we can select them with `dplyr::select(contains("some_common_text"))`.

The access domain is unique in that it has two sub-domains (drive time and public transport time) which are processed separately in the normal way (normalise -> weight -> rank). Then, each sub-domain is exponentially transformed (covered in the next section on calculating SIMD) and the resulting scores are summed in a 2:1 ratio before the final ranking for the domain.

3.6 Re-assigning ranks

We have included some functionality to manually re-assign ranks to allow for certain exceptions. This is done via the `reassignRank` function.

Chapter 4

Calculating SIMD

For the final calculation of the SIMD ranks we follow a few simple steps, see the script `scripts/calculations/openSIMD.R`:

- load packages and data
- inverse domain rankings
- exponentially transform the inversed domain rankings
- combine transformed domain scores
- rank final SIMD scores

First, we load the packages and data from the previous domain rank calculations.

```
library(dplyr)
source("../openSIMD_analysis/scripts/utils/helpers.R")
domains <- read.csv("../openSIMD_analysis/results/domain_ranks.csv")
```

Then we invert the domain rankings (so that 1 = least deprived) and exponentially transform them, using `expoTransform`. This step spreads out the scores at the deprived end of each domain, so that where an area is deprived in one domain, this deprivation won't be cancelled out by a lack of deprivation in another domain.

```
invRank <- function(v) rank(-v)

exponential_domains <- domains %>%
  mutate_at(vars(-data_zone), funs(invRank)) %>%
  mutate_at(vars(-data_zone), funs(expoTransform))
```

We then combine the domain ranks using fixed weightings.

```
with(exponential_domains, {
  simd_score <<-
    .28 * income +
    .28 * employment +
    .14 * health +
    .14 * education +
    .09 * access +
    .05 * crime +
    .02 * housing
})
```

Finally, we rank the final SIMD score to get the SIMD rankings (1 = most deprived).

```
simd_rank <- rank(-simd_score)
```

SIMD and domain ranks are saved in `results/domain_ranks.csv` and `results/openSIMD_ranks.csv`.

Chapter 5

Functions

Below is the documentation of the functions developed for this project, which are used throughout the **openSIMD** procedure.

5.1 normalScores

This function calculates the normal scores for each indicator. The normal score is defined as follows:

$$y_i = \phi^{-1}\left(\frac{r_i}{n+1}\right)$$

where: ϕ^{-1} is the inverse cumulative normal (probit) function, r_i is the rank of the i'th observation and n is the number of non-missing observations for the ranking variable. This is the inverse cumulative normal probability density of proportional ranks. The resulting variable should appear normally distributed regardless of the input data. We translated this approach using the SAS documentation as a guide resulting in the following R function.

```r
normalScores <- function(
  v,                  # a numeric vector as the input variable
  ties = "average",   # passed to ties.method argument in rank()
  forwards = TRUE     # smallest numerical value on left? default is TRUE
) {

  r <- rank(v, ties.method = ties)
  n <- length(na.omit(v))

  rn <- r / (n + 1)

  y <- qnorm(rn, mean = 0, sd = 1, lower.tail = forwards)

  return(y)

}
```

The function takes a numeric vector as its input `v`. It first ranks this input `r` and then calculates the proportional rank `rn`. The final step is to apply the cumulative normal probability using the `qnorm` function. The return value is a numeric vector of the same length as the input `v`.

5.2 replaceMissing

This function replaces missing values, once normalised indicator scores have been calculated. The function finds missing values (as well as `Inf` and `-Inf`) in a vector and replaces them with `0`. This is used in **openSIMD** where a data zone has zero population, or has a missing value for an indicator, and we want it to sit in the centre of the distribution and so we assign a value of 0.

```
replaceMissing <- function(v) replace(v, is.na(v) | v == Inf | v == -Inf, 0)
```

The function takes a numeric vector `v` and returns a vector of equal length with missing values replaced with 0.

5.3 getFAWeights

This function performs a factor analysis using the `psych::fa` function. It then extracts the loadings on the first resulting factor, converts them to proportions of the sum of loadings, the weights, and then returns them as individual elements of a list. This is designed to be equivalent to the SAS procedure in previous SIMD calculations, however the results are only comparable up to two decimal places, likely due to differences in the implementation of factor analysis in the two packages (see section on Quality Assurance).

```
getFAWeights <- function(dat, ...) {

  fact <- psych::fa(dat, nfactors = 1, fm = "ml", rotate = "none", ...)

  f1_scores <- as.data.frame(fact$weights) %>% select(ML1)

  f1_weights <- f1_scores / sum(f1_scores)

  # This is just to make each weight an individual element of a list
  # For compatibility with purrr::map2() in the next step, combineWeightsAndNorms()
  return(lapply(seq_along(f1_weights$ML1), function(i) f1_weights$ML1[i]))

  }
```

The function takes a data.frame `dat` which contains all of the variables for factor analysis. The return value is a list with individual elements corresponding to the weights of variables in the same order as the columns in the input data. A list is returned here for compatibility with the next function `combineWeightsAndNorms`; however, this can be easily converted to a vector with `unlist`.

5.4 combineWeightsAndNorms

This function takes the normalised indicator scores and the weights derived from factor analysis, multiplies them out and then takes the sum of these weighted indicator scores to get the final score for that domain. Weights and indicators need to be in the same order.

```
combineWeightsAndNorms <- function(weights, norms) {

  combined <- purrr::map2(weights, norms, ~ .x * .y)

  combined %>% data.frame %>% rowSums

}
```

The function takes a list of weights (generated by `getFAWeights`) and a data.frame of normalised scores. The function returns a numeric vector containing the combined domain score.

5.5 expoTransform

This function exponentially transforms the (inverted) domain ranks.

```
expoTransform <- function(ranks) {

  prop_ranks <- ranks / max(ranks)

  expo <- -23 * log(1 - prop_ranks * (1 - exp( -100 / 23)))

  return(expo)
}
```

The function takes a numeric vector of the (inverted) ranks and returns a numeric vector of equal length containing the transformed values.

5.6 reassignRank

This function manually reassigns ranks to individual data zones. It can be used, for example, when the domain ranking needs to reflect a population of zero in a data zone.

```
reassignRank <- function(data, domain, data_zone, end = "max", offset = 0) {

  if(end == "max") {
    data[data$data_zone == data_zone, domain] <-
      max(data[, domain], na.rm = TRUE) - (offset + 0.1)
  }else
    if(end == "min") {
      data[data$data_zone == data_zone, domain] <-
        min(data[, domain], na.rm = TRUE) + (offset + 0.1)
    }

  data[, domain] <- rank(data[, domain])

  return(data)
}
```

The function takes a data.frame containing a column named 'data_zone' and a ranked variable. The function changes the rank of the data zone in question (with an optional offset), and it then re-ranks the whole variable. The function returns the corrected data.frame.

Chapter 6

Quality Assurance

In this section, we will show how accurate our translation of the SIMD procedure from SAS to R is. For this, we'll compare the outputs from SAS and R at different points in the calculation.

You will see that the R code correctly replicates the SAS code, but that there are small numeric differences between SAS and R outputs due to the way SAS and R run the factor analysis procedure, and rounding. This means that SIMD ranks produced through the **openSIMD** code slightly differ from the official SIMD16 ranks published on the Scottish Government SIMD website.

Future updates of SIMD will be calculated using R, and the **openSIMD** R code will produce SIMD ranks that are identical with the official SIMD ranks published by the Scottish Government.

6.1 Normalisation

Here, we look at the normalisation step and compare the normalised scores of an education indicator, the percentage of people with no qualifications, calculated in SAS and in R.

Loading the indicator raw data and the (SAS-)normalised indicator scores:

```
noquals <- read_excel("../openSIMD_analysis/data/SAS NOQUALSDATA.xls")
str(noquals)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    6976 obs. of  3 variables:
##  $ DZ      : chr  "S01006506" "S01006507" "S01006508" "S01006509" ...
##  $ Noquals : num  52.8 95.9 38.6 80.1 77.2 ...
##  $ nnoquals: num  -0.7981 0.0512 -1.2185 -0.2042 -0.2607 ...
```

Normalising the indicator data, but now in R:

```
noquals$r_nnoquals <- normalScores(noquals$Noquals)
```

Comparing the two sets of normalised scores, asking to which degree of precision we have reproduced the result obtained in SAS.

```
checkEquivalence <- function(x, y, sig_figs) {
  same <- lapply(sig_figs, function(n) identical(signif(x, n), signif(y, n))) %>% unlist
  data.frame(significant_figures = sig_figs, is_identical = same)
}

checkEquivalence(noquals$nnoquals, noquals$r_nnoquals, 1:16)
```

```
##    significant_figures is_identical
## 1                    1         TRUE
## 2                    2         TRUE
## 3                    3         TRUE
## 4                    4         TRUE
## 5                    5         TRUE
## 6                    6         TRUE
## 7                    7         TRUE
## 8                    8         TRUE
## 9                    9         TRUE
## 10                  10         TRUE
## 11                  11         TRUE
## 12                  12         TRUE
## 13                  13        FALSE
## 14                  14        FALSE
## 15                  15        FALSE
## 16                  16        FALSE
```

The normalised scores for this indicator are identical to eleven decimal places. We believe the remaining differences are due to small differences in how SAS and R work.

6.2 Factor analysis

Here, we look at the factor analysis step.

To test whether SAS and R produce equivalent weights, we load the published indicators and the weights derived from SAS.

```
indicators <- read_excel("../openSIMD_analysis/data/SIMD16 indicator data.xlsx", sheet = 3, na = "*")
sas_weights <- read_excel("../openSIMD_analysis/data/SAS WEIGHTS.xlsx")
```

As an example, we'll calculate the health domain weights.

```
r_weights <- indicators %>%
  select(CIF, SMR, LBWT, DRUG, ALCOHOL, DEPRESS, EMERG) %>%
  mutate_all(funs(normalScores)) %>%
  mutate_all(funs(replaceMissing)) %>%
  getFAWeights %>%
  unlist
```

```
## Warning in qnorm(rn, mean = 0, sd = 1, lower.tail = forwards): NaNs
## produced

## Warning in qnorm(rn, mean = 0, sd = 1, lower.tail = forwards): NaNs
## produced
```

```
sas_weights <- sas_weights %>% select(wt_cif:wt_emerg) %>% unlist
names(sas_weights) <- NULL
```

Note: You should see a warning here when `normalScores` propagates a few missing values.

Now that we have the two sets of health domain weights, we can use the same method as above to compare them.

```
checkEquivalence(sas_weights, r_weights, 1:5)
```

```
##   significant_figures is_identical
```

```
## 1                   1         TRUE
## 2                   2         TRUE
## 3                   3        FALSE
## 4                   4        FALSE
## 5                   5        FALSE
```

The factor analysis weights are equivalent between the two platforms to two significant figures. Again, we think that this is due to the specific implementation of the factor analysis algorithm in the two platforms.

6.3 Domain ranks

Here, we look at how well domain ranks derived through R correlate with the ranks derived through SAS. First, we need a few more packages, and we will load in the SAS results and R results.

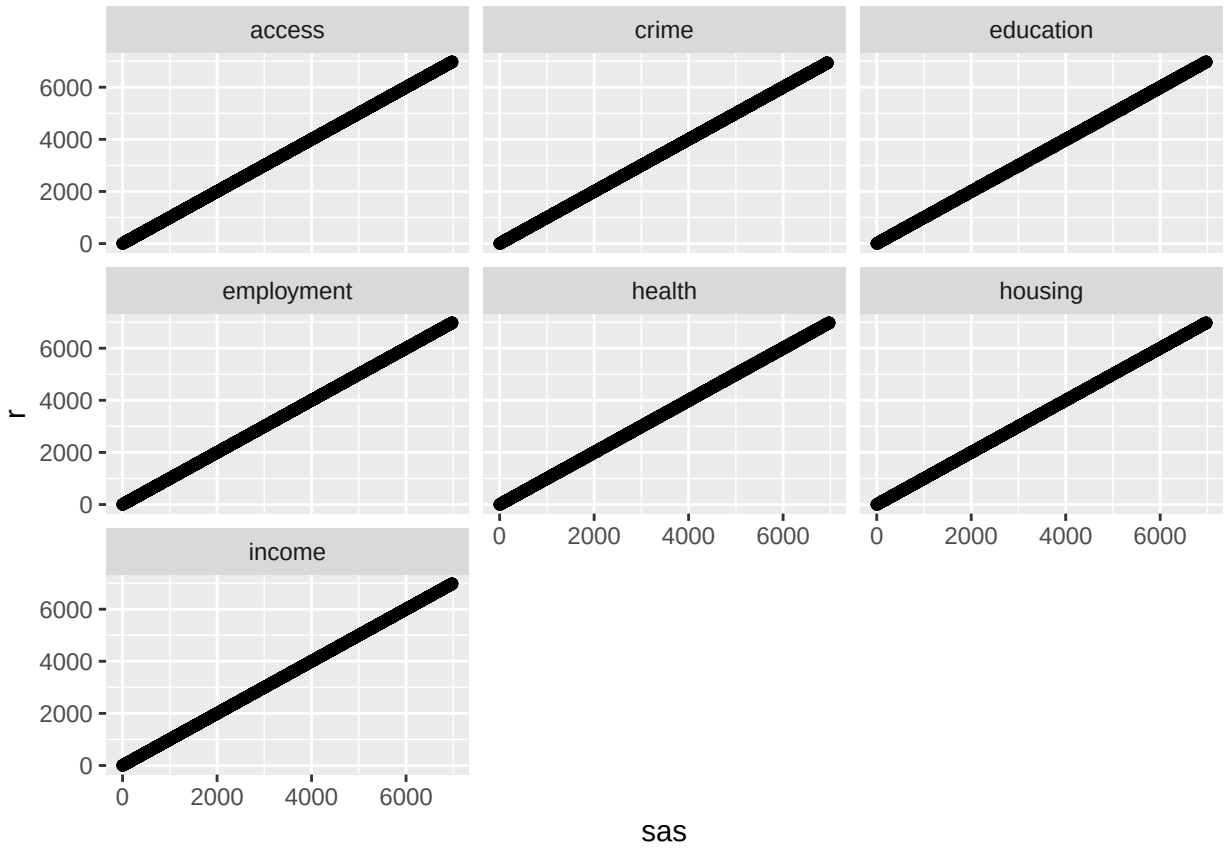
```
library(tidyr)
library(ggplot2)
library(purrr)
sas_results <- read_excel("../openSIMD_analysis/data/SAS SIMD and domain ranks.xlsx", 1)
r_results <- read.csv("../openSIMD_analysis/results/domain_ranks.csv")
```

Now we need to select the necessary columns and rename them. Then, we need to join the two datasets together for plotting.

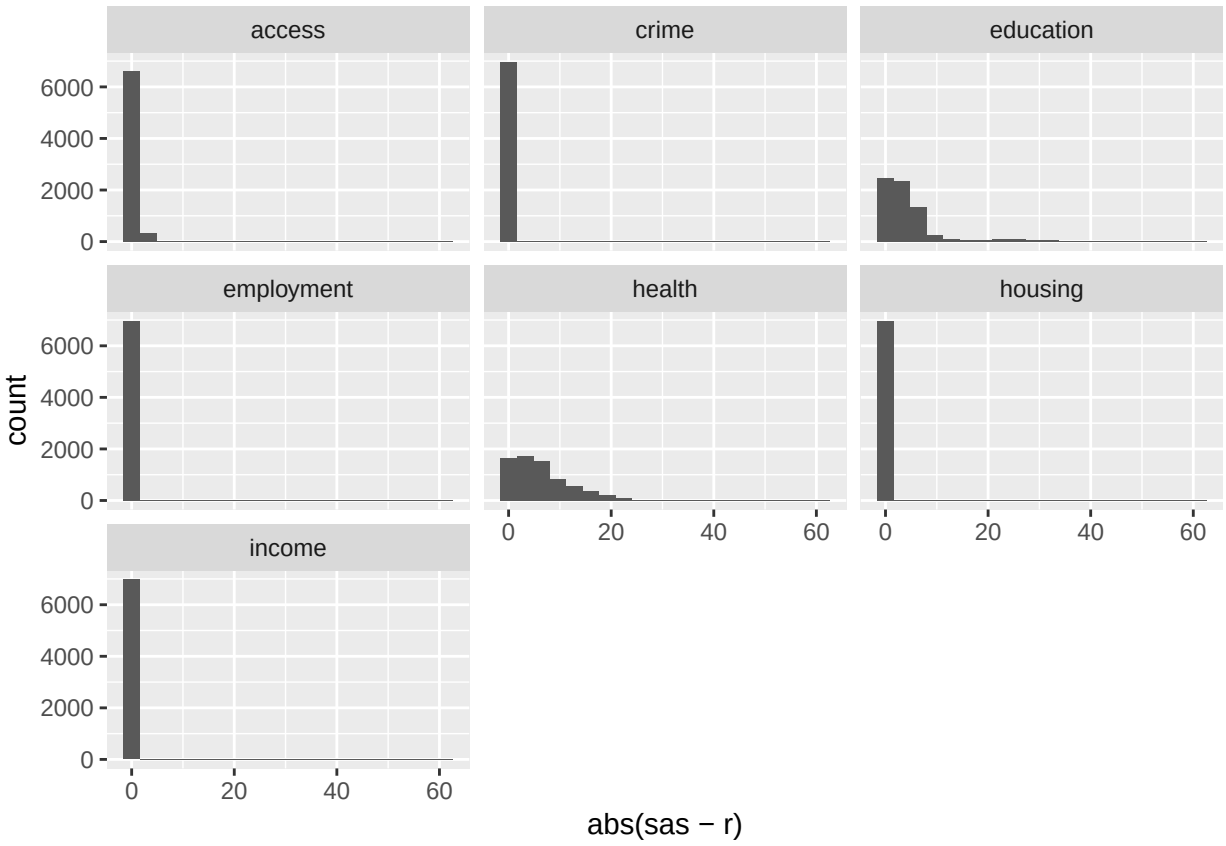
```
sas_domains <- sas_results %>%
  select(-IZ, -LA, -pop, -wapop, -SIMD)
names(sas_domains) <- c("data_zone", "income", "employment", "health",
                        "education", "access", "crime", "housing")

sas_domains$source <- "sas"
r_results$source <- "r"

sas_domains <- gather(sas_domains, domain, rank, -data_zone, -source)
r_results <- gather(r_results, domain, rank, -data_zone, -source)
results <- rbind(sas_domains, r_results) %>% spread(source, rank)
```

Finally, we plot the R- and SAS-derived domain ranks against each other and look at the correlations. We also look at the distribution of differences in domain ranks.

```
ggplot(results, aes(x = sas, y = r)) +
  geom_point() +
  facet_wrap(~ domain)
```




```
ggplot(results, aes(x = abs(sas - r))) +
  geom_histogram(bins = 20) +
  facet_wrap(~ domain)
```



Here, we calculate the correlation coefficient for each comparison, and the median difference in rank.

```
results %>%
  group_by(domain) %>%
  nest %>%
  mutate(rho = map_dbl(data, ~ cor.test(.$sas, .$r, method = "spearman", exact = FALSE)$estimate)) %>%
  mutate(median_diff = map_dbl(data, ~ median(.$r - .$sas))) %>%
  select(domain, rho, median_diff)
```

```
## # A tibble: 7 × 3
##       domain       rho median_diff
##        <chr>     <dbl>       <dbl>
## 1     access 0.9999999           0
## 2      crime 1.0000000           0
## 3  education 0.9999923           1
## 4 employment 1.0000000           0
## 5     health 0.9999911           0
## 6    housing 1.0000000           0
## 7     income 1.0000000           0
```

The results tell us that the ranks are highly correlated but not exactly equal. The largest rank difference is 61, found in the education domain. These differences are due to the numerical discrepancies in the factor analysis step.

6.4 SIMD ranks

Here, we compare the final SIMD rankings between the two platforms. Again, we need to read in the data and join it up.

```
sas_simd <- sas_results %>% select(DZ, SIMD)
r_simd <- read.csv("../openSIMD_analysis/results/openSIMD_ranks.csv")
names(sas_simd) <- c("data_zone", "sas")
names(r_simd) <- c("data_zone", "r")
simd_results <- left_join(sas_simd, r_simd)
```

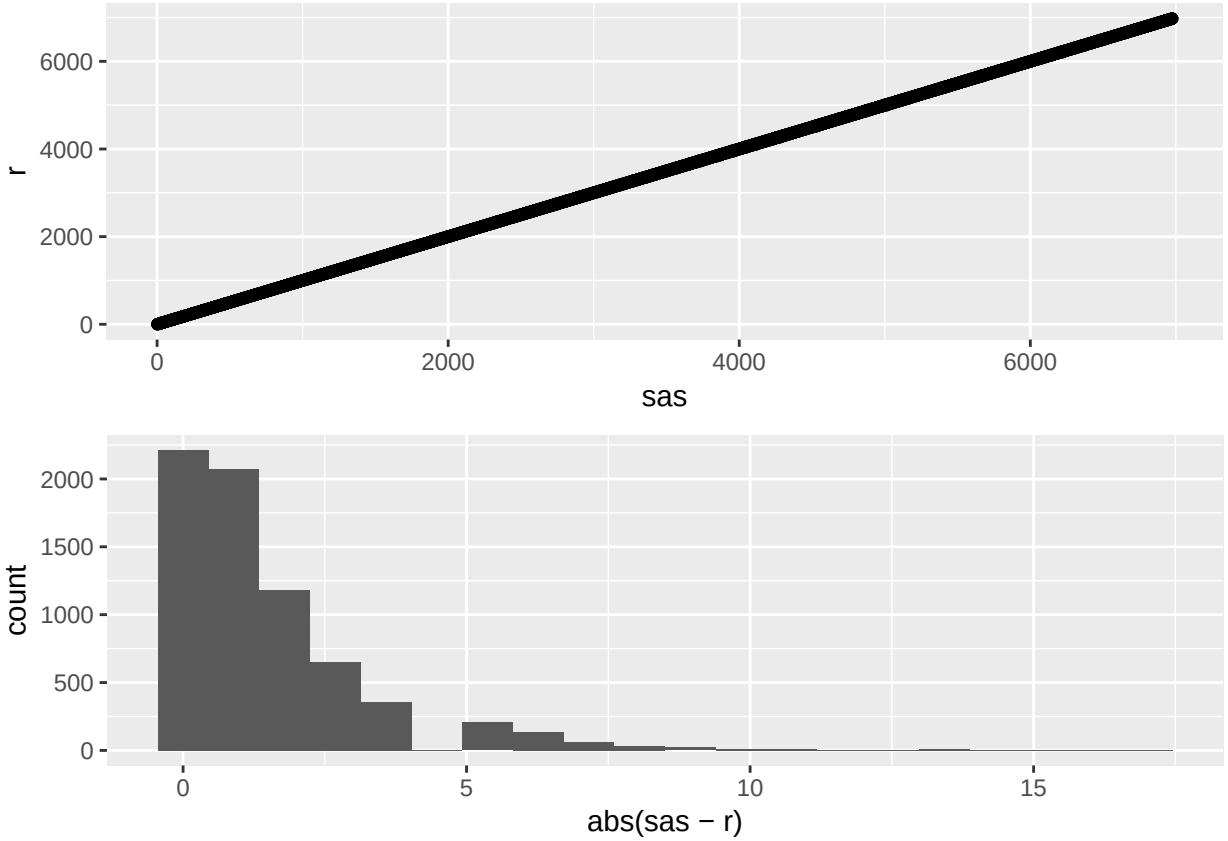
```
## Warning in left_join_impl(x, y, by$x, by$y, suffix$x, suffix$y): joining
## factor and character vector, coercing into character vector
```

Then we examine the correlation in SIMD rankings between R and SAS, and the distribution of differences in SIMD rank.

```
simd_results %>%
  mutate(rho = cor.test(.$sas, .$r, method = "spearman", exact = FALSE)$estimate) %>%
  mutate(median_diff = median(.$r - .$sas)) %>%
  mutate(domain = 'SIMD') %>%
  select(domain, rho, median_diff) %>%
  slice(1:1)
```

```
## # A tibble: 1 × 3
##   domain       rho median_diff
##    <chr>     <dbl>       <dbl>
## 1   SIMD 0.9999993           0
```

```
p1 <- ggplot(simd_results, aes(x = sas, y = r)) +
  geom_point()

p2 <- ggplot(simd_results, aes(x = abs(sas - r))) +
  geom_histogram(bins = 20)
gridExtra::grid.arrange(p1, p2)
```



While there is a tight correlation in the final SIMD rankings, there are some differences. Mostly, these differences lie between 0 and 10 with the largest difference as high as 18.

6.5 Conclusion

The differences between rankings derived using R versus using SAS are mainly due to the factor analysis step, which calculates the indicator weights in the health, education, and access domains. As a result, the weights differ in value after the second decimal place. As all discrepancies can be explained, and the correlation is very high, we conclude that the **openSIMD** R code replicates the original SAS code correctly.

Chapter 7

Appendix

This book was compiled in the following R session:

```
sessionInfo()
```

```
## R version 3.4.0 (2017-04-21)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Sierra 10.12.3
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_GB.UTF-8/en_GB.UTF-8/en_GB.UTF-8/C/en_GB.UTF-8/en_GB.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] purrr_0.2.2   ggplot2_2.2.1 tidyr_0.6.1   dplyr_0.5.0   readxl_1.0.0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.10     cellranger_1.1.0 compiler_3.4.0   plyr_1.8.4
##  [5] tools_3.4.0      digest_0.6.12    evaluate_0.10    tibble_1.3.0
##  [9] nlme_3.1-131     gtable_0.2.0     lattice_0.20-35  psych_1.7.3.21
## [13] DBI_0.6-1        rstudioapi_0.6   yaml_2.1.14      parallel_3.4.0
## [17] gridExtra_2.2.1  stringr_1.2.0    knitr_1.15.1     rprojroot_1.2
## [21] grid_3.4.0       R6_2.2.0         foreign_0.8-67   rmarkdown_1.5
## [25] bookdown_0.3     magrittr_1.5     backports_1.0.5  scales_0.4.1
## [29] htmltools_0.3.6  assertthat_0.2.0 mnormt_1.5-5     colorspace_1.3-2
## [33] labeling_0.3     stringi_1.1.5    lazyeval_0.2.0   munsell_0.4.3
```