Tutorial: Sampling, Weighting and Estimation
Part 2

Stefan Zins, Matthias Sand
and Jan-Philipp Kolb

GESIS - Leibniz Institute
for the Social Sciences

February 1, 2016

Leibniz Institute for the Social Sciences

Selecting a Subgroup by logical operators:

To select a subgroup where all elements equal or do not equal a specific value, you can use == and !=

```
a <- as.vector(c("Aa","AA","Aa","Bb","AA","A","BB","Ba"))
a=="Aa"

## [1]  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE

a[a!="AA"]

## [1] "Aa" "Aa" "Bb" "A"  "BB" "Ba"
```

```
b <- 1:length(a)
a == "AA" & b > 3

## [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE

ab <- which(a=="Aa" & b<=3)
ab

## [1] 1 3

a[ab]

## [1] "Aa" "Aa"
```

Does an element belong to a group?

```
a %in% c("AA","Ba")

## [1] FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE
```

Subsetting

```
sub1 <- bm[bm$Province==3,]
head(sub1[,1:7])

##           Commune   INS Province Arrondiss Men04 Women04  Tot04
## 182       Beernem 31003        3        31  7496    7055  14551
## 183 Blankenberge 31004        3        31  8591    9452  18043
## 184        Bruges 31005        3        31 56565   60283 116848
## 185         Damme 31006        3        31  5494    5482  10976
## 186       Jabbeke 31012        3        31  6879    6807  13686
## 187      Oostkamp 31022        3        31 10616   10837  21453

s <- which(bm$Commune %in% c("Brecht",
                             "Grimbergen","As","Dinant"))
sub2 <- bm[s,]
sub2[,1:7]

##         Commune   INS Province Arrondiss Men04 Women04 Tot04
## 7        Brecht 11009        1        11 12975   12976 25951
## 96   Grimbergen 23025        2        23 16002   17420 33422
## 464          As 71002        7        71  3701    3705  7406
## 556      Dinant 91034        9        91  6138    6668 12806
```

The `subset()` function can also be employed to generate subsets of a data frame

subset {base}

Subsetting Vectors, Matrices and Data Frames

Description

Return subsets of vectors, matrices or data frames which meet conditions.

Usage

```
subset(x, ...)

## Default S3 method:
subset(x, subset, ...)

## S3 method for class 'matrix'
subset(x, subset, select, drop = FALSE, ...)

## S3 method for class 'data.frame'
subset(x, subset, select, drop = FALSE, ...)
```

Arguments

`x`	object to be subsetted.
`subset`	logical expression indicating elements or rows to keep: missing values are taken as false.
`select`	expression, indicating columns to select from a data frame.
`drop`	passed on to `[` indexing operator.
`...`	further arguments to be passed to or from other methods.

```
sub3 <- subset(bm,Commune %in% c("Brecht",
                                  "Grimbergen","As","Dinant"))
sub3$Commune == sub2$Commune

## [1] TRUE TRUE TRUE TRUE
```

```
sub4 <- subset(bm,substr(as.character(Commune),1,1)=="B")
head(sub4[,1:7])

##         Commune   INS Province Arrondiss Men04 Women04 Tot04
## 3      Boechout 11004        1        11  6027    5927 11954
## 4          Boom 11005        1        11  7640    8066 15706
## 5      Borsbeek 11007        1        11  4948    5328 10276
## 6    Brasschaat 11008        1        11 18142   18916 37058
## 7       Brecht 11009        1        11 12975   12976 25951
## 31      Berlaar 12002        1        12  5145    5206 10351
```

- Loops are convenient when conducting one task several times

LOOPS

- Loops are convenient when conducting one task several times
- Very useful for e.g. simulations

Loops

- Loops are convenient when conducting one task several times
- Very useful for e.g. simulations
- But: CPU-intensive

Loops

- Loops are convenient when conducting one task several times
- Very useful for e.g. simulations
- But: CPU-intensive

$\Rightarrow$ avoid loops if possible (esp. for large datasets)

LOOPS

- Loops are convenient when conducting one task several times
- Very useful for e.g. simulations
- But: CPU-intensive

$\Rightarrow$ avoid loops if possible (esp. for large datasets)

```
A1 <- vector()
for(i in 1:10){
  A1[i] <- sample(1:10,1)
}
A1

##  [1] 10 10  3  9  7  6  8  2  7  8
```

gesis Leibniz Institute for the Social Sciences

- To save the results, it is useful to create a container prior to the loop

```
A2 <- matrix(nrow = 5,ncol = 2)
for(i in 1:nrow(A2)){
  A <- sample(1:50,30)
  A2[i,1] <- mean(A)
  A2[i,2] <- var(A)
}
A2

##       [,1]  [,2]
## [1,] 27.40 230.7
## [2,] 27.50 195.6
## [3,] 25.33 221.7
## [4,] 29.67 198.0
## [5,] 28.40 251.1
```

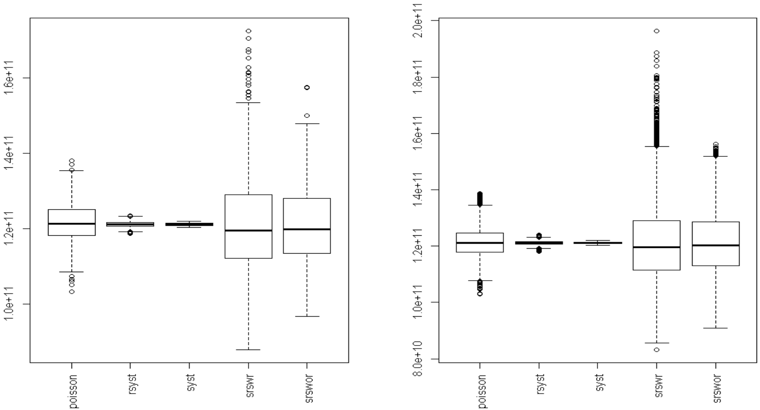
```
data(belgianmunicipalities)
pik <- inclusionprobabilities(belgianmunicipalities$Tot04,200)
# Computes the inclusion probabilities
N <- length(pik)
# population size
n <- sum(pik)
# sample size
sim <- 1000
ss <- array(0, c(sim, 5))
# sim2 <- 10000   #second simulation
# ss2 <- array(0, c(sim2, 5))
# number of simulations
y <- belgianmunicipalities$TaxableIncome
# variable of interest
ht <- numeric(5)
# Horvitz-Thompson estimator for the simulation
```

```
for (i in 1:sim) {
    cat("Step ", i, "\n")
    s <- UPpoisson(pik)
    ht[1] <- HTestimator(y[s == 1], pik[s == 1])
    s <- UPrandomsystematic(pik)
    ht[2] <- HTestimator(y[s == 1], pik[s == 1])
    s <- UPsystematic(pik)
    ht[3] <- HTestimator(y[s == 1], pik[s == 1])
    s <- sample(y, n,replace = T)
    ht[4] <- HTestimator(s, rep(n/N,n))
    s <- srswor(n, N)
    ht[5] <- HTestimator(y[s == 1], rep(n/N, n))
    ss[i, ] <- ss[i, ] + ht
  }
```

- `cat()` can be used to display the simulation step that is recently proceeded

Example based on: Tille,Y and Matai, A. (2010). TEACHING SURVEY SAMPLING WITH THE SAMPLING R PACKAGE. *ICOTS8 (2010) Invited Paper Refereed*

```r
# boxplots of the estimators
  par(mfrow=c(1,2))
  boxplot(data.frame(ss), las = 3)
  boxplot(data.frame(ss2), las = 3)
```



apply {base} R Documentation

Apply Functions Over Array Margins

Description

Returns a vector or array or list of values obtained by applying a function to margins of an array or matrix.

Usage

```
apply(X, MARGIN, FUN, ...)
```

Arguments

`X`	an array, including a matrix.
`MARGIN`	a vector giving the subscripts which the function will be applied over. E.g., for a matrix `1` indicates rows, `2` indicates columns, `c(1, 2)` indicates rows and columns. Where `X` has named dimnames, it can be a character vector selecting dimension names.
`FUN`	the function to be applied: see 'Details'. In the case of functions like `+`, `%*%`, etc., the function name must be backquoted or quoted.
`...`	optional arguments to `FUN`.

- If `margin=1`, the function will be applied to the rows of an array
- If `margin=2`, the function will be applied to the columns of an array

The apply function

- If `margin=1`, the function will be applied to the rows of an array
- If `margin=2`, the function will be applied to the columns of an array

```
Applydat <- matrix(1:25, nrow = 5, ncol = 5, byrow = F)
apply(Applydat,1,mean)

## [1] 11 12 13 14 15

apply(Applydat,2,mean)

## [1]  3  8 13 18 23
```

tapply {base} R Documentation

Apply a Function Over a Ragged Array

Description

Apply a function to each cell of a ragged array, that is to each (non-empty) group of values given by a unique combination of the levels of certain factors.

Usage

```
tapply(X, INDEX, FUN = NULL, ..., simplify = TRUE)
```

Arguments

`X`	an atomic object, typically a vector.
`INDEX`	list of one or more factors, each of same length as `X`. The elements are coerced to factors by `as.factor`.
`FUN`	the function to be applied, or `NULL`. In the case of functions like `+`, `%*%`, etc., the function name must be backquoted or quoted. If `FUN` is `NULL`, tapply returns a vector which can be used to subscript the multi-way array `tapply` normally produces.
`...`	optional arguments to `FUN`: the Note section.

```r
Tapplydat <- data.frame(Income = rnorm(6,1400,200),
                        Gender = sample(c("Male","Female"),
                                        6,replace = T))
Tapplydat

##   Income Gender
## 1   1703   Male
## 2   1452   Male
## 3   1418   Male
## 4   1376   Male
## 5   1161 Female
## 6   1522 Female

tapply(Tapplydat$Income, Tapplydat$Gender, mean)

## Female   Male
##   1342   1487
```

The finite population correction...

$$1 - f = \frac{N - n}{N} = 1 - \frac{n}{N}$$

...can also be turned into a `R` function

```
fpc <- function(N,n){(N-n)/N}
fpc(100,8)

## [1] 0.92
```

Generating a telephone sample with the approach of Gabler and Häder:

Constructing a synthetic frame:

```
fra <-data.frame(pre = sample(c(30,40,89,221,621),10000,
                              replace = T),bank = sample(100:99999,1000
fra[1:4,]

##   pre  bank
## 1 221  5728
## 2 221 32661
## 3 621 64698
## 4 621 35754

fra <- fra[order(fra[,1]),]
fra[1:4,]

##    pre  bank
## 11  30 87896
## 13  30  7164
## 15  30 63673
## 16  30 17772
```

```
tel.samp <- function(fra,n){
  len <- nrow(fra)*100
  s <- sort(sample(len,n))
  row <- ceiling(s/100)
  app <- s%%100
  ts <- fra[row,]
  num <- data.frame(prefix = paste("0",ts[,1],sep = ""),
                    number = paste(ts[,2],app,sep = ""))
  return(num)
}
```

- `fra` is the sampling frame
- `n` is the sample size
- with `return()`, one can determine the results that the function should display

```
my.first.ts <- tel.samp(fra,10)
head(my.first.ts)

##   prefix  number
## 1    030 4965129
## 2    030 2690436
## 3    040 8495419
## 4    040 7116060
## 5    040 3043965
## 6    089 3369754
```

- `sort()` sorts an atomic vector in an ascending or descending order and returns the values
- `order()` sorts an atomic vector or a data frame in an ascending or descending order and returns the row number

Using a loop to draw stratified random samples

```
str.bm <- split(bm,bm$Province)
nh <- c(2,3,7,3,2,6,7,2,9)
res <- list()
for(i in 1:length(str.bm)){
  ID <- str.bm[[i]]$INS
  res[[i]] <- sample(ID,nh[i],replace=F)
}
s <- unlist(res)
result<-bm[bm$INS %in% s,]
table(result$Province)

##
## 1 2 3 4 5 6 7 8 9
## 2 3 7 3 2 6 7 2 9
```

You can also use the `strata()` command from the `sampling` package

⇒ the function returns the unit's identifier, stratum and its inclusion probability

```
s <- strata(bm,"Province",nh,method = "srswor")
result1 <- getdata(bm,s)
head(result1[,c(1:3,ncol(result)-1,ncol(result))])

##                    Commune   INS Arrondiss medianincome Province
## 48                  Dessel 13006        13        20212        1
## 51               Herentals 13011        13        19141        1
## 89   Woluwe-Saint-Pierre 21019        21        22051        2
## 151                 Linter 24133        24        21053        2
## 161          Grez-Doiceau 25037        25        21029        2
## 182                Beernem 31003        31        20268        3
```

⇒ `getdata()` merges the sample-IDs with your original dataset and returns your sample as a data frame

Proportional Allocation:

$$\gamma_h = \frac{N_h}{N}$$

$$n_h = n * \gamma_h$$

```
n <- 30
gamma <- prop.table(table(bm$Province))
nh <- n*gamma
t(nh)

##
##           1     2     3     4     5     6     7     8     9
##   [1,] 3.565 5.654 3.260 3.311 3.514 4.278 2.241 2.241 1.935

s <- strata(bm,"Province",nh,"srswor")
result.p <- getdata(bm,s)
nrow(result.p)

## [1] 26
```

Optimal Allocation:

$$n_h = n * \frac{\gamma_h \sigma_h}{\sum_{g=1}^{L} \gamma_g \sigma_g} = n * \frac{N_h \sigma_h}{\sum_{g=1}^{L} N_g \sigma_g} \qquad h = 1, \ldots, L$$

gesis Leibniz Institute for the Social Sciences

Optimal Allocation:

$$n_h = n * \frac{\gamma_h \sigma_h}{\sum_{g=1}^{L} \gamma_g \sigma_g} = n * \frac{N_h \sigma_h}{\sum_{g=1}^{L} N_g \sigma_g} \qquad h = 1, \dots, L$$

Step 1: getting $Var(\overline{y}_{StrRS,opt})$

$$Var(\overline{y}_{StrRS,opt}) = \frac{1}{n} \sum_{h=1}^{L} (\gamma_h * \sigma_h)^2 = \frac{1}{n} \sum_{h=1}^{L} (\frac{N_h}{N} \sigma_h)^2 = \frac{1}{N^2} \sum_{h=1}^{L} \frac{N_h^2}{n_h} \sigma_h^2$$

```r
GetStratVar <- function(Y, sind, nh) {
  Nh <- tapply(sind,sind,length)
  N <- length(sind)
  sum(Nh^2*tapply(Y,sind,function(x)
    var(x)*(length(x)-1)/length(x))/nh)/N^2
}
GetStratVar(bm$Tot04,bm$Province,
            rep(5,length(unique(bm$Province))))

## [1] 18941644
```

gesis
Leibniz Institute
for the Social Sciences

Step 2: calculating n_h

```
GetOptAlloc <- function(Y, sind, n){
  L <- length(unique(sind))
  nh <- rep(2,L)
  Nh <- tapply(sind,sind,length)
  v <- numeric(L)
  M <- diag(rep(1,L))
  while (sum(nh) < n) {
    for (i in 1:L) {
      if (nh[i] == Nh[i]) {
        v[i] <- Inf
      } else {
        v[i] <- GetStratVar(Y, sind, nh + M[,i])
      }
    }
    nh <- nh + M[,which.min(v)]
  }
  nh
}
```

```
nh <- GetOptAlloc(bm$Tot04,bm$Province,50)
t(nh)

##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
## [1,]   13    9    4    6    6    6    2    2    2

nh2 <- GetOptAlloc(bm$averageincome,bm$Province,50)
t(nh2)

##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
## [1,]    6   14    4    5    6    6    2    4    3
```

```
nh <- GetOptAlloc(bm$Tot04,bm$Province,50)
t(nh)

##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
## [1,]   13    9    4    6    6    6    2    2    2

nh2 <- GetOptAlloc(bm$averageincome,bm$Province,50)
t(nh2)

##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
## [1,]    6   14    4    5    6    6    2    4    3
```

$\Rightarrow$ Allocation differs depending on the variable of interest

Simple Method:

⇒ Sample cluster proportional to size; sample all units within a cluster

```
l <- 4
gamma <- prop.table(table(bm$Province))
clus <- sample(unique(bm$Province),l, prob = gamma, replace = F)
res.clus <- bm[bm$Province %in% clus,]
nrow(res.clus)

## [1] 303
```

Simple Method:

⇒ Sample cluster proportional to size; sample all units within a cluster

```
l <- 4
gamma <- prop.table(table(bm$Province))
clus <- sample(unique(bm$Province),l, prob = gamma, replace = F)
res.clus <- bm[bm$Province %in% clus,]
nrow(res.clus)

## [1] 303
```

⇒ Sample size varies

Fixed Sample Size:

⇒ Sample cluster proportional to size; sample the same number of units within each cluster

```
l <- 4
gamma <- prop.table(table(bm$Province))
clus <- sample(unique(bm$Province),l, prob = gamma, replace = F)
fixed.res.clus <-list()
for(i in 1:l){
  nh <- 30
  bm.cl <- bm[bm$Province == clus[i],]
  fixed.res.clus[[i]] <- sample(bm.cl$INS,nh, replace = F)
}
ID <- unlist(fixed.res.clus)
fixed.clus <- bm[bm$INS %in% ID,]
nrow(fixed.clus)

## [1] 120
```

Cluster Sampling

The `sample` package also offers a cluster function

```
l <- 4
sam.clus <- cluster(bm,"Province",4,method = "srswor")
res.clus.samp <- getdata(bm,sam.clus)
nrow(res.clus.samp)

## [1] 216
```

$\Rightarrow$ Samples all units within a cluster

Mean: Simple Random Sampling (SRS)

$$\bar{y}_{SRS} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

```
SRS.mean <- function(Y,S){return(mean(Y[S]))}
```

Variance: Simple Random Sampling (SRS)

$$V(\bar{y}_{SRS}) = \frac{\sigma^2}{n}; \qquad V(\bar{y}_{SRSWOR}) = \frac{N-n}{N-1}\frac{\sigma^2}{n} = (1-\frac{n}{N})\frac{S^2}{n}$$

$$\hat{V}(\bar{y}_{SRSWOR}) = (1-\frac{n}{N})\frac{s^2}{n}$$

```
SRS.evar <- function(Y,S){return(var(Y[S])/length(S))}
SRSWOR.evar <- function(Y,S)
  {return(fpc(nrow(Y),length(S))*var(Y[S])/length(S))}
```

Leibniz Institute for the Social Sciences

Mean: Stratified Random Sampling (StrRS)

$$\hat{\bar{y}}_{StrRS} = \sum_{h=1}^{L} \gamma_h \frac{1}{n_h} \sum_{i=1}^{n_h} y_i$$

```
Strat.mean <- function(Y,sind,S){
  Nh <- tapply(Y,sind,length)
  Str.mean <- sum(Nh*tapply(Y[S], sind[S], mean) / sum(Nh))
  return(Str.mean)
}
S <- as.numeric(row.names(result))
Strat.mean(bm$averageincome,bm$Province,S)

## [1] 24000
```

- `Y` is the variable of interest
- `sind` is the identifier of the strata (`length(Y)`)
- `S` are the row names of the sample

Leibniz Institute for the Social Sciences

Variance: Stratified Random Sampling (StrRS)

$$V(\overline{y}_{StrRS}) = \sum_{h=1}^{L} \gamma_h^2 \frac{\sigma_h^2}{n_h}; \qquad V(\overline{y}_{StrRSWOR}) = \sum_{h=1}^{L} \gamma_h^2 \frac{\sigma_h^2}{n_h} * \frac{N_h - n_h}{N_h - 1}$$

$$\hat{V}(\overline{y}_{StrRS}) = \sum_{h=1}^{L} \gamma_h^2 \frac{s_h^2}{n_h}; \qquad \hat{V}(\overline{y}_{StrRSWOR}) = \sum_{h=1}^{L} \gamma_h^2 \frac{s_h^2}{n_h} * \frac{N_h - n_h}{N_h}$$

```
Strat.evar<- function(Y, sind, S) {
  Nh <- tapply(sind,sind,length)
  nh <- tapply(sind[S], sind[S], length)
  ssh <- tapply(Y[S], sind[S], var)
  res <- sum((Nh/sum(Nh))^2*ssh/nh*(Nh-nh)/Nh)
  return(res)
}
S <- as.numeric(row.names(result))
Strat.evar(bm$averageincome,bm$Province,S)

## [1] 256082
```

Mean: One-Stage Cluster Sampling (SRCS)

$$\hat{\bar{y}}_{SRCS} = \frac{L}{l}\sum_{h=1}^{l}\gamma_h^a * \bar{y}_h^a = \frac{L}{l}\sum_{h=1}^{l} * \frac{N_h^a}{\sum_{g=1}^{L} N_g^a} * \frac{1}{N_h^a}\sum_{i=1}^{N_h^a} Y_{ih}$$

```
SRCS.mean <-function(Y,sind,S){
  L <- length(unique(sind))
  l <- length(unique(sind[S]))
  N <- length(Y)
  N_h_a <- tapply(Y[S],sind[S],length)
  mu_h_a <- tapply(Y[S],sind[S],mean)
  return(L/l*sum(N_h_a/N*mu_h_a))
}
Sc <- as.numeric(row.names(res.clus))
SRCS.mean(bm$averageincome,bm$Province,Sc)

## [1] 30483
```

gesis Leibniz Institute for the Social Sciences

Mean: One-Stage Cluster Sampling (SRCS)

$$V(\overline{y}_{SRCS}) = \frac{L^2}{N^2} * \frac{\sigma_e^2}{l} * \frac{L-l}{L-1}$$

$$\hat{V}(\overline{y}_{SRCS}) = \frac{L^2}{N^2} * \frac{s_e^2}{l} * \frac{L-l}{L};$$

$$s_e^2 = \frac{1}{l-1} \sum_{r=1}^{l} (N_r^a \overline{y}_r^a - \frac{N * \hat{\overline{y}}_{SRCS}}{L})^2$$

Calculating s_e^2

```
se.sq <- function(Y,sind,S){
  L <- length(unique(sind))
  l <- length(unique(sind[S]))
  N <- length(Y)
  mu.SRCS <- SRCS.mean(Y,sind,S)
  c <- N*mu.SRCS/L
  mu_h_a <- tapply(Y[S],sind[S],mean)
  N_h_a <- tapply(Y[S],sind[S],length)
 return ( 1/(l-1)*sum((N_h_a*mu_h_a-c)^2))
}
```

Calculating $\hat{V}(\bar{y}_{SRCS})$

```
SRCS.evar <- function(Y,sind,S){
  L <- length(unique(sind))
  l <- length(unique(sind[S]))
  N <- length(Y)
  part1 <- L^2/N^2
  part2 <- se.sq(Y,sind,S)/l
  part3 <- (L-l)/L
  return(part1*part2*part3)
}
Sc <- as.numeric(row.names(res.clus))
SRCS.evar(bm$averageincome,bm$Province,Sc)

## [1] 28973878
```