Guidelines & toolkits

NiPN
National Information Platforms for Nutrition

Toolkit for assessing the quality of anthropometry data

Draft Version – December 2017

Mark Myatt

ABOUT THE NIPN INITIATIVE

National Information Platforms for Nutrition (NIPN) is an initiative of the European Commission supported by the United Kingdom Department for International Development and the Bill & Melinda Gates Foundation. The initiative aims to strengthen national capacity to manage and analyse information and data from all sectors which have an influence on nutrition and to disseminate and use information so as to better inform the strategic decisions countries are faced with to prevent undernutrition and its consequences. A Global Support Facility has been set up by the European Commission to coordinate the NIPN initiative, managed by the Agrinatura alliance and hosted by Agropolis International.

DISCLAIMER

AUTHOR

Mark Myatt, Brixton Health, Consultant Epidemiologist

COPYRIGHT STATEMENT

RECOMMENDED CITATION

Myatt, M. *Toolkit for assessing the quality of anthropometry data*. Montpellier, France: Agropolis International, Global Support Facility for the National Information Platforms for Nutrition initiative. 2017.

PUBLICATION DATE

December 2017

The full report can be downloaded here:
http://www.nipn-nutrition-platforms.org/IMG/pdf/nipn-data-quality-toolkit.pdf

Contents

1. Introduction

This document presents a set of practical analytical methods that can be applied to variables in datasets to assess their quality. Criteria to judge data quality are also presented. An additional function to calculate z-scores of anthropometric indices is described in an Appendix.

The focus of this toolkit is on data required to assess the anthropometric status of humans, such as measurements of weight, height or length, mid upper-arm circumference (MUAC), sex and age. Many of the methods could be applied to other types of data.

Data quality is assessed by:

Checking ranges and legal value to identify *univariate* outliers (Section 3: Checking ranges and legal values).

Assessing the sex ratio (Section 4: Sex ratio).

Assessing age distributions and age by sex distributions (Section 5: Age and sex distributions).

Assessing the extent of digit preference in recorded measurements (Section 6: Digit preference in anthropometric measurements).

Assessing the extent of age heaping in recorded ages (Section 7: Age heaping).

Using scatterplots and statistical methods to identify *bivariate* outliers (Section 8: Using scatterplots to identify outliers).

Using flags to identify outliers in anthropometric indices (Section 9: Identifying outliers using flags).

Examining the distribution and the statistics of the distribution of measurements and anthropometric indices (Section 10: Assessing the distribution of anthropometric variable, indices and indicators).

These activities and a proposed order in which they should be performed are shown in *Figure 1.1*.

An additional section provides material relating the mean and standard deviations of indicator values, prevalence, and the PROBIT estimator (Section 11: Mean, standard deviation, prevalence, and the PROBIT estimator).

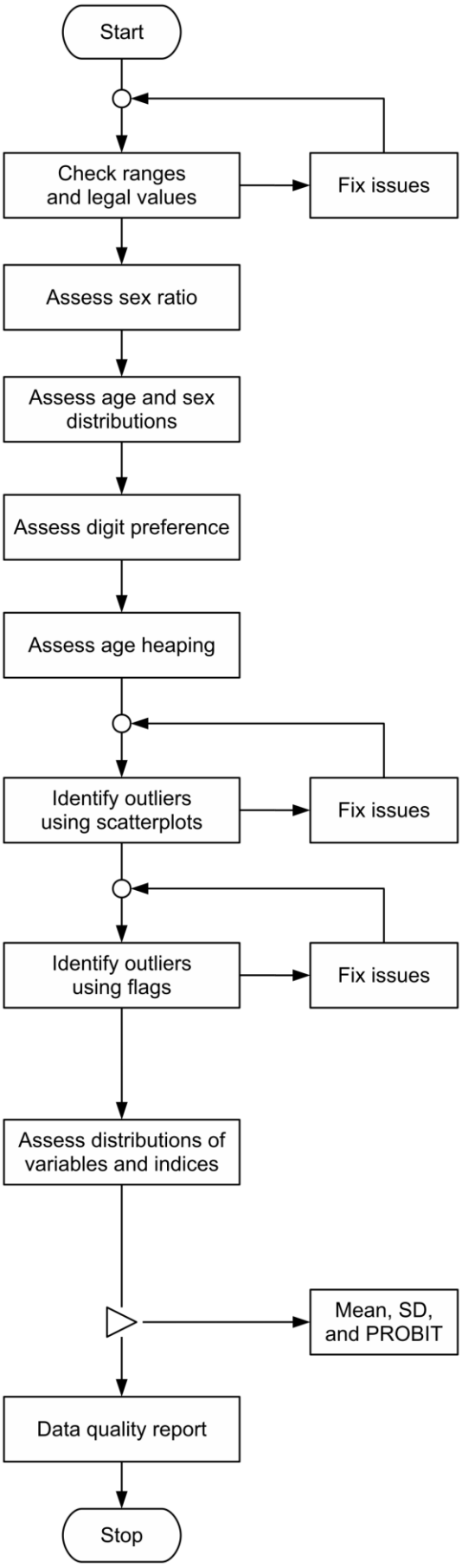
The quality of data can then be assessed (Section 12: Assessing data quality).

The material is intended to provide a practical or "hands on" introduction to assessing data quality and is presented as a series of computer-based exercises. Example datasets are provided.

Extensive use is made of the *R* language and environment for statistical computing. This is a free and powerful data analysis system. Methods have been described in sufficient detail to allow activities to be performed using other data analysis systems.

R provides a very extensive language for working with data. The material presented here has been written using only a small subset of the *R* language. Many of the data quality activities are supported by *R* functions that have been written specifically for this purpose. These simplify the assessment of the quality of data related to anthropometry and anthropometric indices. The basic *R* functions, the purpose-written functions, and the filenames of example datasets are also shown in *Figure IN01*.

Figure 1.1 Data quality flow chart, associated *R* functions, and example datasets



Key *R* functions	Example datasets

`read.table()`
`head()`
`summary()`
`boxplot()`
`outliersUV()`
`table()`
`write.table()`
	`rl.ex01.csv`

`prop.test()` **`sexRatioTest()`**	`ah.ex01.csv` `dp.ex02.csv`

`recode()`
`pyramid.plot()`
`cut()`
`ageChildren()`
`ageRatioTest()`
	`as.ex01.csv`
	`as.ex02.csv`
	`dp.ex02.csv`

`barplot()`
`abline()`
`chisq.test()`
`fullTable()`
`digitPreference()`
	`dp.ex01.csv`
	`dp.ex02.csv`
	`dp.ex03.csv`

`%%` **`ageHeaping()`**	`ah.ex01.csv` `dp.ex02.csv`

`plot()`
`cor()`
`identify()`
`outliersMD()`
`by()`
`lines()`
	`sp.ex01.csv`
	`sp.ex02.csv`

`ifelse()`
`is.na()`
`table()`
`prop.table()`
`mean()`
`hist()`
`national.SMART()`
`subset()`
	`flag.ex01.csv`
	`flag.ex02.csv`
	`flag.ex03.csv`

`sd()`
`histNormal()`
`qqNormalPlot()`
`shapiro.test()`
`skewKurt()`
`aov()`
`TukeyHSD()`
`mad()`
`greensIndex()`
	`dist.ex01.csv`
	`dist.ex02.csv`
	`flag.ex01.csv`

`pnorm()`
`function()`
`optimise()`	

Functions given in **`bold`** are purpose-written functions provided by the NiPN data quality toolkit

2. The NIPN data quality toolkit

The material in this guide uses some purpose-written functions in the *R* language. R is free software for statistical computing. It can be downloaded and installed from here:

https://www.r-project.org/

The functions for use in the NIPN data quality toolkit are provided in a file named **`nipnTK.r`**. You can load this set of functions into *R* using the **`source()`** function. You should do this each time you start *R*.

For example, if you place the **`nipnTK.r`** file in the directory:

```
~/Documents/Clients/NIPN/toolkit/
```

you would use:

```
source("~/Documents/Clients/NIPN/toolkit/nipnTK.r")
```

to load the file into *R*.

That is a UNIX path. If you are using Microsoft Windows™ and have placed the file **`nipnTK.r`** in the directory:

```
c:\dataquality
```

then you would use:

```
source("c:/dataquality/nipnTK.r")
```

to load the file into *R*.

Note that *R* uses the forward slash (**`/`**) rather than the backslash (**`\`**) as separators in pathnames. This is because *R* uses the backslash character as an "escape" character (i.e. a special character that alters the meaning of the subsequent character).

Be careful if you copy and paste the *R* commands from this document and then edit them to create new commands. It is best to do this in the user interface that you use with R, or in a plain text editor such as WordPad or Notepad++ on Windows; EMACS, VIM, or Scintilla on Linux; BBEdit on MacOS; or the editor provided by your preferred *R* environment (e.g. RStudio). This is because word processors such as Microsoft Word® tend to use "smart" asymmetrical quotes like this **`“text”`** rather than plain, symmetrical quotes such as this **`"text"`**. *R* does not recognise the asymmetrical quotes and will give an error message if they are used.

A set of example data files is also provided with this toolkit. It is easiest to set the working directory to the directory where these files are stored once you start *R*. You can do this using the *R* graphical user interface when you start *R* or by using the **`setwd()`** function.

A quick way of opening file in *R* without having to type a long pathname is to use the **`file.choose()`** function as in:

```
source(file.choose())
```

to load a collection of *R* functions or an *R* script.

A similar technique may be used for reading data files. For example:

```
svy <- read.table(file.choose(), header = TRUE, sep = ",")
```

will allow you to select and read a *comma separated value* (CSV) file and store it in a **`data.frame`** object named **`svy`**. You will do this each time you load a new data set.

Example datasets to accompany each exercise has been provided in CSV format files.

The CSV format has been used because it is easy to use with different data analysis systems. Most users of *R* tend to use data in simple text-based formats such as CSV files.

R can read and write data in a number of different formats, which are shown in Table 2.1. Libraries and functions for parsing XML data and for extracting data from websites and PDF files are also available.

Table 2.1 The format of files that R can read and write, with the additional package required if needed.

	R can ...			
Format	**Read**	**Write**	**Functions**	**Package***
All text based formats**	●	●	**`read.table()`**	None
Weka ARFF	●	●	**`read.arff()`**	**`foreign`*****
dBase (DBF)	●	●	**`read.dbf()`**	
STATA	●	●	**`read.dta()`**	
EpiInfo / EpiData (REC)	●	○	**`read.epiinfo()`**	
SPSS	●	○	**`read.spss()`**	
Systat	●	○	**`read.systat()`**	
SAS	●	○	**`read.ssd()`**	
ODBC data sources	●	●	Many. SQL queries supported.	**`RODBC`**
Excel (XLS, XLSX)	●	●		
	●	○	**`read.xls()`**	**`gdata`**
	●	●	Many	**`XLConnect`**
	●	●	**`read.xlsx()`**	**`xlsx`**

* Packages provide additional functions, example datasets, and associated help pages. A wide variety of packages are available from https://cran.r-project.org

** This includes comma separated value (CSV) files. The field separator character may be specified to enable datasets to be read that use a comma as the decimal separator (e.g. **`12,3`** to represent **`12.3`**) and are delimited with tabs or semi-colons. The **`read.fwf()`** and **`read.fortran()`** functions are used to read files in fixed width formats. These formats are commonly used in census and large-survey data.

*** The foreign library is installed by default and will not need to be downloaded.

2.1 Using *R*

R is a language product. Almost everything you will do in *R* will require you to issue commands at the keyboard or apply commands stored in a text file. You can cut and paste the commands from this toolkit directly into the interface and then press the *Enter* key to run them.

The default graphical user interface (GUI) for *R* is quite basic. The *R* Core Team concentrate on the development and maintenance of the *R* language and provide only a very simple GUI. Separate teams are responsible for developing graphical user interfaces.

A number of alternative graphical user interfaces for *R* are available.

Both *RStudio*:

https://www.rstudio.com

and *R-AnalyticFlow*:

http://r.analyticflow.com/en/

provide sophisticated, open-source and free graphical user interfaces for *R*.

RStudio provides an interface similar to an integrated development environment (IDE) for a programming language.

R-AnalyticFlow provides a scientific workflow system with graphical user interface components for common data management and data analysis activities.

R is a widely used and well-supported system. A large number of books and manuals are available in paper and digital formats. The *R* project provides books and manuals, a number of mailing lists with searchable archives that provide general and specialist support, and a refereed journal. See:

https://cran.r-project.org

There are also a number of independent online forums and blogs providing support to *R* users. These are easily accessed from an Internet search engine. Preceding each search term with **R** is usually sufficient to find what you want. For example, a search using **R test for normality** will return many useful links.

Courses in using *R* are available in many countries: try searching for **R training**.

The *R* help system is very comprehensive. To find help on a specific function type **?** followed by the name of the function you are interested in. For example, typing:

```
?read.table
```

will display help for the **read.table()** function.

To find help of a specific topic type **??** followed by the topic of interest. For example, typing:

```
??normality
```

will display a list of functions relating to the term “normality”.

2.2 Functions provided by the NIPN data quality toolkit

The remainder of this section provides details of the functions provided by the NIPN data quality toolkit.

Functions whose names start with **`plot.`** and **`print.`** are used whenever **`plot()`** and **`print()`** functions are used with objects returned by specified functions. For example, the function **`plot.digitPreference()`** causes **`plot()`** to produce a plot of final digit frequencies from objects returned by the **`digitPreference()`** function. These functions are used automatically as and when needed. The functions are presented in alphabetical order in Table 2.2

Table 2.2 A list of the functions available in the NIPN data quality toolkit, in alphabetical order.

Function	**`ageChildren()`**	
Purpose	Goodness of fit to an expected (model-based) age distribution	
Parameters	**`age`**	Vector of child ages
	`u5mr`	Under five years mortality rate as deaths / 10,000 / day (default = 1)
Returns	**`u5mr`**	Specified mortality rate
	`observed`	Table of counts in each (year-centred) age group
	`Expected`	Table of expected counts in each (year-centred) age group
	`X2`	Chi-squared test statistic
	`df`	Degrees of freedom for Chi-squared test statistic
	`p`	p-value for Chi-squared test statistic
Notes	Creates year centred age-groups (i.e. 6:17, 18:29, 30:41, 42:53, and 54:59 months). This function is of most use with data from SMART (and similar) surveys.	

Function	**`ageHeaping()`**	
Purpose	Age-heaping analysis	
Parameters	**`X`**	Vector of child ages
	`divisor`	Divisor usually 5, 6, 10, or 12 (default = 12)
Returns	**`X2`**	Chi-squared test statistic
	`df`	Degrees of freedom for Chi-squared test statistic
	`p`	p-value for Chi-squared test statistic
	`tab`	Table of remainders (i.e. for x / divisor)
	`pct`	Table of proportions (%) of remainders (for x / divisor)

Function	**`ageRatioTest()`**	
Purpose	Age ratio test	
Parameters	**`x`**	Vector of child ages
	`ratio`	Expected age ratio (default = 0.85)
Returns	**`expectedR`**	Expected sex ratio
	`expectedP`	Expected proportion aged 6:29 months
	`observedR`	Observed sex ratio
	`observedP`	Observed proportion aged 6:29 months
	`X2`	Chi-squared test statistic
	`df`	Degrees of freedom for Chi-squared test statistic
	`p`	p-value for Chi-squared test statistic
Notes	Uses 6:29 and 30:59 month age-groups. This function is of most use for data from SMART (and similar) surveys.	

Function	**`boxText()`**
Purpose	Plot text in a coloured bounding box to a graphics device
Parameters	See function definition in **`nipnTK.r`**
Returns	See function definition in **`nipnTK.r`**
Notes	This is a helper function used by **`ageHeaping()`** and **`digitPreference()`**

Table 2.2 contd. A list of the functions available in the NIPN data quality toolkit, in alphabetical order.

Function	**digitPreference()**	
Purpose	Digit preference score	
Parameters	**x**	Numeric vector
	digits	Number of decimal (digits = 1 (e.g.) treats 105 as 105.0
	values	A vector of possible values for the final digit (default = 0:9)
Returns	**dps**	Digit Preference Score (DPS)
	tab	Table of final digit counts
	pct	Table of proportions (%) of final digit counts
	dpsClass	SMART DPS classification
Notes	DPS definition from: Kari Kuulasmaa K, Hense HW, Tolonen H (for the WHO MONICA Project), *Quality Assessment of Data on Blood Pressure in the WHO MONICA Project*, WHO MONICAProject e-publications No. 9, WHO, Geneva, May 1998 Available from : http://www.thl.fi/publications/monica/bp/bpqa.htm	

Function	**fullTable()**	
Purpose	Fill out a one-dimensional table to include a specified range of values	
Parameters	**x**	A vector to tabulate
	values	A vector of values to be included in a table defaults to range of **x**)
Returns	A table object including zero cells	
Notes	This is a helper function used by **ageChildren()**, **ageHeaping()**, and **digitPreference()**. May also be used as a standalone function. The returned table object will only contain cells for values of **x** specified by **values**. The default for **values** will only work reliably with numeric variable containing whole numbers.	

Function	**greensIndex()**	
Purpose	Green's Index of Dispersion by bootstrap	
Parameters	**data**	Survey dataset (as an R data.frame)
	psu	Name of variable holding PSU (cluster) data as a character
	case	Name of variable holding case status (case = 1)
	replicates	Number of bootstrap replicates to use (default is 999)
Returns	**GI**	Point estimate of Green's Index of dispersion
	LCL	95% LCL for Green's Index of dispersion
	UCL	95% UCL for Green's Index of dispersion
	minGI	Minimum possible GI (maximum uniformity) for the data
	p	p-value (H0 := Random distribution of cases across PSUs)

Function	**histNormal()**	
Purpose	Histogram with normal curve superimposed	
Parameters	**x**	Numeric vector
Returns	**NULL**	The null (empty) object
Notes	Additional names parameters **xlab**, **ylab**. **main**, **breaks**, and **ylim** are passed to **hist()** with sensible default values if none are specified.	

Function	**national.SMART()**	
Purpose	Add SMART flags to a stratified sample survey (e.g. MICS, DHS, national SMART)	
Parameters	**x**	Survey dataset (as an *R* data.frame object) with indices present
	strata	Name of column in **x** that defines the strata
	indices	Names of columns in **x** containing indices
Returns	An *R* data.frame object with same structure as **x** with a **flagSMART** column added. This column is coded using sums of powers of two.	

Table 2.2 contd. A list of the functions available in the NIPN data quality toolkit, in alphabetical order.

Function	**outliersMD()**	
Purpose	Mahalanobis distance to detect bivariate outliers	
Parameters	**x**	Numeric vector
	y	Numeric vector
	alpha	Critical alpha value to detect an outlier (default = 0.001)
Returns	A logical vector (**TRUE** for an outlier at p < **alpha**)	

Function	**outliersUV()**	
Purpose	IQR to detect univariate outliers	
Parameters	**x**	Numeric vector
	Fence	IQR multiplier (default = **1.5**)
Returns	A logical vector (**TRUE** for an outlier)	

Function	**plot.ageChildren()**
Purpose	Provides **plot()** method for object with class **ageChildren**

Function	**plot.ageHeaping()**
Purpose	Provides **plot()** method for object with class **ageHeaping**

Function	**plot.digitPreference()**
Purpose	Provides **plot()** method for object with class **digitPreference**

Function	**print.ageChildren()**
Purpose	Provides **print()** method for object with class **ageChildren**

Function	**print.ageHeaping()**
Purpose	Provides **print()** method for object with class **ageHeaping**

Function	**print.ageRatioTest()**
Purpose	Provides **print()** method for object with class **ageRatioTest**

Function	**print.digitPreference()**
Purpose	Provides **print()** method for object with class **digitPreference**

Function	**print.greensIndex()**
Purpose	Provides **print()** method for object with class **greensIndex**

Function	**print.sexRatioTest()**
Purpose	Provides **print()** method for object with class **sexRatioTest**

Function	**print.skewKurt()**
Purpose	Provides **print()** method for object with class **skewKurt**

Function	**pyramid.plot()**	
Purpose	Age-by-sex pyramid plot	
Parameters	**x**	Vector of ages (usually grouped)
	g	Vector of groups (usually sex)
	main	Plot title
	xlab	x-axis label
	ylab	y-axis label (usually omitted)
	col	Colours for bars. Either a single colour (default is **col = "white"**) for all bars, two colours (e.g. **col = c("lightblue", "pink"**) for left hand side bars and right hand side bars respectively, or many colours allocated on a "checkerboard" basis to each bar.
	...	Additional parameters (e.g. **cex.names**) passed to **barplot()**
Returns	A table of **x** by **g** (not displayed)	
Notes	Useful defaults are provided for the **main**, **xlab**, **ylab**, and **col** parameters	

Table 2.2 contd. A list of the functions available in the NIPN data quality toolkit, in alphabetical order.

Function	**qqNormalPlot()**	
Purpose	Normal quantile-quantile plot	
Parameters	**x**	A numeric vector

Function	**recode()**	
Purpose	A recode function	
Parameters	**var**	Variable to recode
	recodes	Recode specifications in a character string separated by semicolons of the form **input=output** as in: **"1=1;2=1;3:6=2;else=NA"** If an input value satisfies more than one specification, then the first (reading from left to right) is applied. If no specification is satisfied, then the input value in **var** is carried over to the result unchanged **NA** is allowed on both input and output. The following recode specifications are supported: **Specification** / **Example** / **Notes** Single values: **9=NA** Set of values: **c(1,2,5)=1**; **seq(1,9,2)='odd'**; **1:10=1** — The left-hand-side is any *R* function call that returns a vector Range of values: **7:9=3**; **lo:115=1** — Special values **lo** and **hi** (for lowest and highest values) may be used Other values: **else=NA** Character values are quoted as in: **recodes = "c(1,2,5)='sanitary';else='unsanitary'"** The output may be the (scalar) result of a function call as in: **recodes = "999=median(var, na.rm = TRUE)"** The ouput may be the (scalar) value of a variable as in: **recodes = "999=scalarVariable"** If all of the output values are numeric, and if **afr** is FALSE, then a numeric result is returned; if **var** is a factor then (by default) so is the result.
	afr	Return a factor. Default is **TRUE** if **var** is a factor and is **FALSE** otherwise.
	anr	Coerce result to numeric (default = **TRUE**).
	levels	Order of the levels in the returned factor. The default is to use the sort order of the level names.
Returns	Recoded variable	
Notes	Users are strongly advised to carefully check the results of **recode()** calls with any outputs that are the results of a function call.	

Table 2.2 contd. A list of the functions available in the NIPN data quality toolkit, in alphabetical order.

Function	`sexRatioTest()`	
Purpose	Sex ratio test	
Parameters	`sex`	Numeric vector (sex)
	`codes`	Codes used to identify males and females **(in that order)**
	`pop`	Relative populations of males and females **(in that order)**
Returns	`pM`	Observed proportion male
	`eM`	Expected proportion male
	`X2`	Chi-squared test statistic
	`df`	Degrees of freedom for Chi-squared test
	`p`	p-value for for Chi-squared test

Function	`skewKurt()`	
Purpose	Skewness and kurtosis statistics and tests	
Parameters	`x`	Numeric vector
Returns	`s`	Skewness with direction
	`s.se`	Standard error of skewness
	`s.z`	Test statistic
	`s.p`	p-value for skewness
	`k`	Kurtosis with direction
	`k.se`	Standard error of kurtosis
	`k.z`	Test statistic
	`k.p`	p-value for kurtosis

3. Checking ranges and legal values

Checking that data are within an acceptable or plausible range is an important basic test to apply to quantitative data. Checking that data are recorded with appropriate legal values or codes is an important basic check to apply to categorical data.

3.1 Checking quantitative data

We will retrieve a survey dataset:

```
svy <- read.table("rl.ex01.csv", header = TRUE, sep = ",")
head(svy)
```

The file **`rl.ex01.csv`** is a comma-separated-value (CSV) file containing anthropometry data from a SMART survey in Angola.

We can use the **`summary()`** function to examine the range and other summary statistics of a quantitative variable:

```
summary(svy$muac)
```

This returns:

```
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
11.1   128.0   139.0   140.3   148.0   999.0
```

A graphical examination can also be made:

```
boxplot(svy$muac, horizontal = TRUE, xlab = "MUAC (mm)", frame.plot = FALSE)
```

See *Figure 3.1*. The "whiskers" on the boxplot extend to 1.5 times the interquartile range from the ends of the box (i.e. the lower and upper quartiles). This is known as the *inner fence*. Data points that are outside the inner fence are considered to be *mild outliers*. The NIPN data quality toolkit provides an *R* language function **`outliersUV()`** that uses the same method to identify outliers:

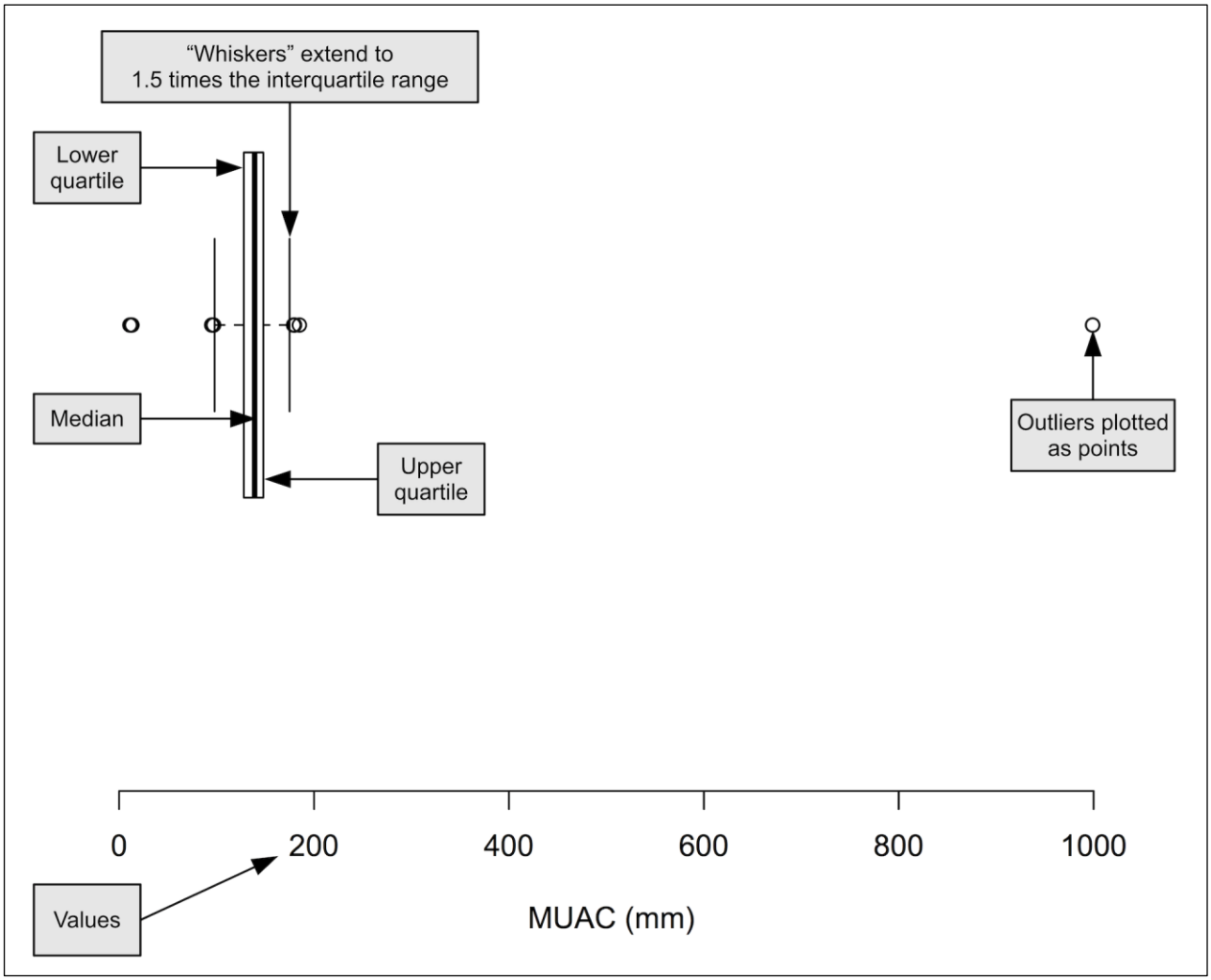
```
svy[outliersUV(svy$muac), ]
```

This returns:

```
Univariate outliers : Lower fence = 98, Upper fence = 178

    age sex weight height  muac oedema
 33  24   1    9.8   74.5 180.0      2
 93  12   2    6.7   67.0  96.0      1
126  16   2    9.0   74.6 999.0      2
135  18   2    8.5   74.5 999.0      2
194  24   M    7.0   75.0  95.0      2
227   8   M    6.2   66.0  11.1      2
253  35   2    7.6   75.6  97.0      2
381  24   1   10.8   82.8  12.4      2
501  36   2   15.5   93.4 185.0      2
594  21   2    9.8   76.5  13.2      2
714  59   2   18.9   98.5 180.0      2
752  48   2   15.6  102.2 999.0      2
756  59   1   19.4  101.1 180.0      2
873  59   1   20.6  109.4 179.0      2
```

Figure 3.1. A boxplot to identify outliers in a **`muac`** variable.



We can count the number of outliers or use:

```
table(outliersUV(svy$muac))
```

This returns:

```
FALSE  TRUE
  892    14
```

We can express this as a proportion:

```
prop.table(table(outliersUV(svy$muac)))
```

This returns:

```
     FALSE       TRUE
0.98454746 0.01545254
```

You may find it easier to use percentages:

```
prop.table(table(outliersUV(svy$muac))) * 100
```

This returns:

```
    FALSE      TRUE
98.454746  1.545254
```

Some of the **muac** values identified as potential outliers are possible values of **muac**, which are underlined:

```
    age sex weight height  muac oedema
 33  24   1    9.8   74.5 180.0      2
 93  12   2    6.7   67.0  96.0      1
126  16   2    9.0   74.6 999.0      2
135  18   2    8.5   74.5 999.0      2
194  24   M    7.0   75.0  95.0      2
227   8   M    6.2   66.0  11.1      2
253  35   2    7.6   75.6  97.0      2
381  24   1   10.8   82.8  12.4      2
501  36   2   15.5   93.4 185.0      2
594  21   2    9.8   76.5  13.2      2
714  59   2   18.9   98.5 180.0      2
752  48   2   15.6  102.2 999.0      2
756  59   1   19.4  101.1 180.0      2
873  59   1   20.6  109.4 179.0      2
```

The **outliersUV()** function provides a **fence** parameter which alters the threshold at which a data point is considered to be an outlier.

The default is **fence = 1.5**, which defines the *inner fence* (i.e **1.5** times the interquartile range below the lower quartile and above the upper quartile). This will identify *mild* and *severe* outliers:

The value **fence = 3** defines the *outer fence* (i.e **3** times the interquartile range below the lower quartile and above the upper quartile). This will identify *severe* outliers only:

```
svy[outliersUV(svy$muac, fence = 3), ]
```

This returns:

```
Univariate outliers : Lower fence = 68, Upper fence = 208

    age sex weight height  muac oedema
126  16   2    9.0   74.6 999.0      2
135  18   2    8.5   74.5 999.0      2
227   8   M    6.2   66.0  11.1      2
381  24   1   10.8   82.8  12.4      2
594  21   2    9.8   76.5  13.2      2
752  48   2   15.6  102.2 999.0      2
```

There is something wrong with all of these values of **muac**.

The intention was that the **muac** variable records mid-upper-arm-circumference (MUAC) in mm. There are some impossibly small (i.e. **11.1**, **12.4**, and **13.2**) and impossibly large values (i.e. **999.0**).

The three impossibly small values are probably due to data being recorded in cm rather than mm. It is probably safe to change these three values to **111**, **124**, and **132**. It is easiest to do this each record separately:

```
svy$muac[svy$muac == 11.1] <- 111
```

An alternative approach is to specify the row numbers instead of the values:

```
svy$muac[381] <- 124
svy$muac[594] <- 132
```

The three **999.0** values are missing values coded as **999.0**. It is safe to set these three values to missing using the special **NA** value:

```
svy$muac[svy$muac == 999.0] <- NA
```

Range checks should be repeated after editing the data to ensure that the problems have been fixed:

```
summary(svy$muac)
svy[outliersUV(svy$muac), ]
svy[outliersUV(svy$muac, fence = 3), ]
```

Figure 3.2 shows a new boxplot of the **muac** variable made using:

```
boxplot(svy$muac, horizontal = TRUE, xlab = "MUAC (mm)", frame.plot = FALSE)
```

after the fixes for incorrectly entered data and missing values were made. There should now be no severe outliers:

```
prop.table(table(outliersUV(svy$muac, fence = 3))) * 100
```

Returns:

```
FALSE
  100
```

It is usually better to identify and edit only the most extreme *univariate* outliers, as we have done here, and use a scatterplot and the statistical distance methods described in Section 8 of this toolkit to identify other potential outliers.

3.2 Editing data

We have edited records with outliers at the *R* command line.

It is a good idea to edit data at the command line or using a script containing the required commands.

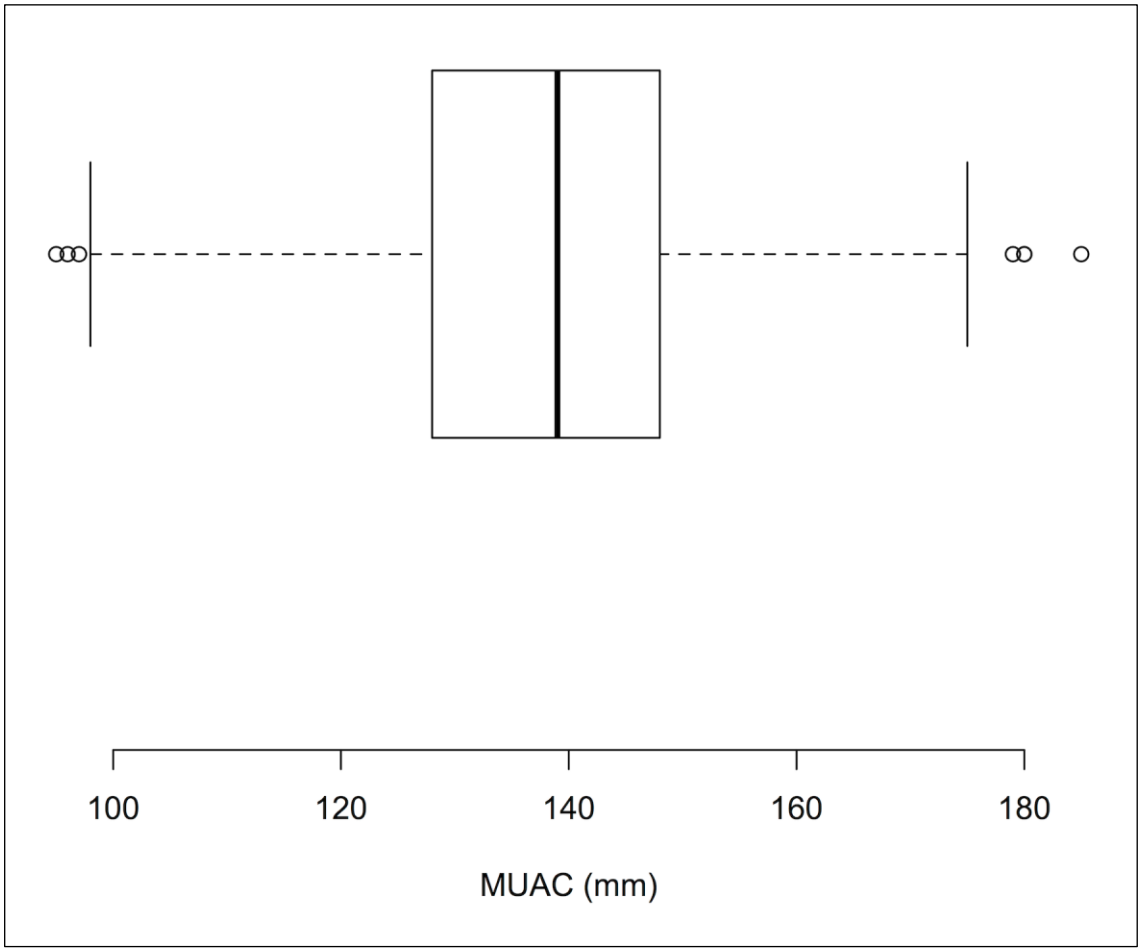
A script should be saved to provide a record of changes made to the data.

R also keeps a record of whatever you do at the command line in a "history file". The history file is a plain text file which is usually called **.Rhistory** and stored in your home directory.

Some regulatory authorities require you to keep a history file.

Some publications may require you to provide a "reproducible data analysis". This could be an edited and annotated copy of your history file.

Figure 3.2. A boxplot of the **muac** variable after editing the incorrectly entered data and recoding values of **999.0** to **NA** (missing)



The **edit()** function provides a basic tool for editing data interactively.

Editing data using the **edit()** function is typically a three stage process:

1. Create a new object containing only the data that requires editing.
2. Use the **edit()** function to edit data in the new object closing the data editor window when you are finished.
3. Replace the old records with the edited records

We will try this using a separate copy of the example data:

```
x <- read.table("rl.ex01.csv", header = TRUE, sep = ",")
records2update <- x[outliersUV(x$muac, fence = 3), ]
records2update <- edit(records2update)
x[row.names(records2update), ] <- records2update
```

We can check that the edits have been made using:

```
x[outliersUV(x$muac, fence = 3), ]
```

If you have fixed the problems in the data this should return:

```
age    sex    weight height muac   oedema
<0 rows> (or 0-length row.names)
```

The **edit()** function works differently on different operating systems and with different graphical user interfaces. If you are using *RStudio* or *RAnalyticFlow* on Mac OS X you will need to install *XQuartz* if you want to use the **edit()** function. *XQuartz* is available from:

https://www.xquartz.org/index.html

3.3 Checking categorical variables

We can use the **table()** function to examine the codes used for categorical variables. For example:

```
table(svy$sex)
```

returns:

```
  1   2   3   F   M
404 458   1  24  19
```

The intention was that the **sex** variable was coded using **1** for male and **2** for female, but in a small number of records the codes **M** for male and **F** for female have been used. A mixed coding scheme like this will complicate data management and data analysis. Data in the sex variable should be edited to ensure that consistent coding is used:

```
svy$sex[svy$sex == "M"] <- 1
svy$sex[svy$sex == "F"] <- 2
```

You may find that a few records contain meaningless codes. The code **3** in the example dataset has, very probably, no meaning and is likely to be a simple data entry error. This record should be checked and corrected, if possible. If the record cannot be corrected then the **sex** variable should be set to missing:

```
svy$sex[svy$sex == 3] <- NA
```

Legal value checks should be repeated after editing to ensure that problems have been fixed:

```
table(svy$sex)
```

now returns:

```
  1   2   3   F   M
423 482   0   0   0
```

The table contains cells for the values **M**, **F**, and **3** because *R* imported the variable as a categorical or "factor" variable:

```
str(svy)
```

returns:

```
'data.frame':906 obs. of  6 variables:
$ age   : int  12 6 6 8 12 8 18 9 12 12 ...
$ sex   : Factor w/ 5 levels "1","2","3","F",..: 2 1 2 1 1 1 1 1 2 1 ...
$ weight: num  6.7 6.4 6.5 7.2 6.1 7.7 6.4 7.8 7.5 6.5 ...
$ height: num  68.5 65 65.6 68.4 65.4 66.5 66.7 65.3 69.1 70.3 ...
$ muac  : num  148 125 125 144 114 146 119 140 138 121 ...
$ oedema: int  2 2 2 2 2 2 2 2 2 2 ...
```

We can fix this by redefining the levels of the sex variable:

```
levels(svy$sex) <- c("1", "2", NA, NA, NA)
table(svy$sex)
```

3.4 Saving changes

We have edited some data.

We usually want to save changes.

It is simple to save a dataset in a comma-separated-value (CSV) text file using the **`write.table()`** function:

```
write.table(x = svy, file = "rl.ex01.clean.csv", sep = ",", quote = FALSE,
      row.names = FALSE, fileEncoding = "ASCII")
```

R can work with a variety of files formats but it is usually simplest to work with plain text files.

4. Sex ratio

The male to female sex ratio test checks whether the ratio of the number of males to the number of females in a survey sample is similar to an expected ratio. The expected male to female sex ratio can be calculated from census or similar data. If there is no expected value then it is usually assumed that there should be equal numbers of males and females in the survey sample. This is usually true for children and young adults but may not be true for older adults.

4.1 Sex ratios in children

We will retrieve a survey dataset:

```
svy <- read.table("dp.ex02.csv", header = TRUE, sep = ",")
head(svy)
```

The file **dp.ex02.csv** is a comma-separated-value (CSV) file containing anthropometric data from a SMART survey in Kabul, Afghanistan.

It is reported that there are about 2.658 million boys and 2.508 million girls aged between zero and four years in Afghanistan (2012 estimates).

The male to female sex ratio can be calculated by entering:

```
2.658 / 2.508
```

which gives:

```
1.059809
```

It is often easier to work with the proportion of the population that is male:

```
2.658 / (2.658 + 2.508)
```

which gives:

```
0.514518
```

We can compare this to the proportion of the sample that is male:

```
table(svy$sex)
```

which gives:

```
  1   2
438 435
```

This table is more useful when the cell counts are expressed as proportions:

```
prop.table(table(svy$sex))
```

which gives:

```
        1         2
0.5017182 0.4982818
```

A formal test can be made to compare the observed proportion with the proportion of the population:

```
prop.test(table(svy$sex), p = 0.514518)
```

This returns:

```
        1-sample proportions test with continuity correction

data:  table(svy$sex), null probability 0.514518
X-squared = 0.5225, df = 1, p-value = 0.4698
alternative hypothesis: true p is not equal to 0.514518
95 percent confidence interval:
 0.4680459 0.5353752
sample estimates:
        p
0.5017182
```

The male to female sex ratio (expressed as the proportion male) in the example data is not significantly different from the expected male to female sex ratio expressed as the proportion that is male.

The NIPN data quality toolkit provides an *R* language function called **`sexRatioTest()`** that performs a sex ratio test:

```
sexRatioTest(svy$sex, codes = c(1, 2), pop = c(2.658, 2.508))
```

which returns:

```
        Sex Ratio Test

Expected proportion male = 0.5145
Observed proportion male = 0.5017
X-squared = 0.5225, p = 0.4698
```

The codes used in the sex variable for male and female are specified using the `codes` parameter. If **`sex`** was coded using **`M`** and **`F`** then you would specify **`codes = c("M", "F")`**.

Population data are specified using the **`pop`** parameter (males then females). This can be specified as numbers or as a ratio. The test above could have been specified as:

```
sexRatioTest(svy$sex, codes = c(1, 2), pop = c(1.059809, 1))
```

If (e.g.) you want to specify a one to one sex ratio then you would use **`pop = c(1, 1)`**.

The typical sex ratio observed at birth is 1.06:1.00 (males to females). This could be used to assess if selective abortion or female infanticide is taking place, although a large sample size (i.e. about n = 6200) is required for such a test to have sufficient power.

4.2 Sex ratios by age-group

The sex ratio test may be performed on each age group separately. You can apply the sex ratio test to each age-group using the **by()** function:

```
svy$ycag <- recode(svy$age, "6:17=1; 18:29=2; 30:41=3; 42:53=4; 54:59=5")
by(svy$sex, svy$ycag, sexRatioTest, codes = c(1, 2), pop = c(2.658, 2.508))
```

Note that the variable **ycag** created above holds the year-centred age-group.

This approach assumes that the sex ratio is independent of age.

An approach that does not make this assumption is to use the numbers of male and female children in the same age-ranges in the population taken from census data.

A useful source of census data is the United States Census Bureau's International Data Base:

```
https://www.census.gov/population/international/data/idb/informationGateway.php
```

This source gives the following estimates for Afghanistan in 2016:

Age (years)	Number males	Number female	Proportion Male	Proportion female	Male to female sex ratio
0	594,602	573,956	0.5088	0.4912	1.04:1.00
1	550,593	533,579	0.5078	0.4922	1.03:1.00
2	526,827	510,479	0.5079	0.4921	1.03:1.00
3	509,048	493,185	0.5079	0.4921	1.03:1.00
4	493,521	478,137	0.5079	0.4921	1.03:1.00

We need to ensure we use the same age-ranges as the census:

```
svy$ageGroup <- recode(svy$age, "0:11=0; 12:23=1; 24:35=2; 36:47=3; 48:59=4")
```

and then test the sex ratio in each age group separately:

```
sexRatioTest(svy$sex[svy$ageGroup == 0], pop = c(594602, 573956))
sexRatioTest(svy$sex[svy$ageGroup == 1], pop = c(550593, 533579))
sexRatioTest(svy$sex[svy$ageGroup == 2], pop = c(526827, 510479))
sexRatioTest(svy$sex[svy$ageGroup == 3], pop = c(509048, 493185))
sexRatioTest(svy$sex[svy$ageGroup == 4], pop = c(493521, 478137))
```

All of these tests find no significant differences between the observed and expected sex ratios.

It should be noted that some (or all) of the tests might be based on small sample sizes:

```
table(svy$ageGroup)
```

and may, therefore, be able to detect only large differences.

4.3 Sex ratios in adults

With data from children we usually expect something like a one to one male to female sex ratio.

This will not usually be the case with adults, especially older adults.

We will retrieve a survey dataset:

```
svy <- read.table("ah.ex01.csv", header = TRUE, sep = ",")
head(svy)
```

The file **ah.ex01.csv** is a comma-separated-value (CSV) file containing anthropometry data from a Rapid Assessment Method for Older People (RAM-OP) survey in the Dadaab refugee camps in Garissa, Kenya. This is a survey of older people, defined as people aged sixty years and older.

With this type of survey it is usually possible to use camp administration data to find the expected male to female sex ratio. This information was not given in the RAM-OP survey report.

The camp population is predominantly Somali. It is reported that there are 188 thousand men and 220 thousand women aged sixty years and older in Somalia (2010 estimates). The sex ratio is:

```
188 / 220
```

which is:

```
0.8545455
```

The expected proportion of the population that is male is:

```
188 / (188 + 220)
```

which is:

```
0.4607843
```

The proportion of the sample that is male:

```
prop.table(table(svy$sex))
```

is:

```
0.381113
```

This looks to be much smaller than the expected proportion. The sex ratio test:

```
sexRatioTest(svy$sex, codes = c(1, 2), pop = c(188, 220))
```

reports:

```
      Sex Ratio Test

Expected proportion male = 0.4608
Observed proportion male = 0.3811
X-squared = 14.8305, p = 0.0001
```

The proportion of males in the sample is significantly smaller than we expected.

This result could be due to the extraordinary nature of the population (e.g. the camp population could really have very many more older women than older men). It could also due to a selection bias in the survey. In this example, men were more likely than women to be away from home during the day and a household sample taken during the day would have systematically excluded the more active members of the male population.

Note that the sex ratio test only applies to population surveys. If surveys focus on (e.g.) carers of small children then the observed male to female sex ratio is likely to be strongly biased towards women. In such cases it is not sensible to apply a sex ratio test.

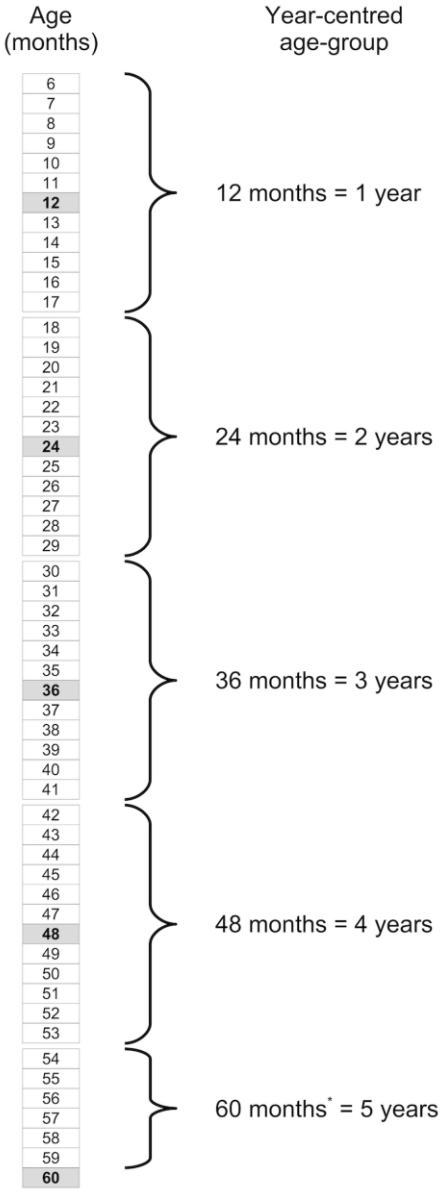
5. Age and sex distributions

Age heaping is the tendency to report children's ages to the nearest year or adult's ages to the nearest multiple of 5 or 10 years. Age heaping is very common. It is a major reason why data from nutritional anthropometry surveys are often analysed and reported using broad age-groups.

5.1 Age and sex distributions: children's data

The commonest age-groups used with children's data are 6 to 17 months, 18 to 29 months, 30 to 41 months, 42 to 53 months, and 54 to 59 months (see *Figure 5.1*). These are known as *year-centred age-groups*. Note that the last age-group covers only six months but is nominally centred at five years. Other age-groups may be used for specific analyses. The techniques presented here can be adapted to work with other age-groups.

Figure 5.1 The age-groups frequently used when working with data from children.



5.1.1 Tabulation and visualisation

We will retrieve a survey dataset:

```
svy <- read.table("dp.ex02.csv", header = TRUE, sep = ",")
head(svy)
```

The file **dp.ex02.csv** is a comma-separated-value (CSV) file containing anthropometric data from a SMART survey in Kabul, Afghanistan.

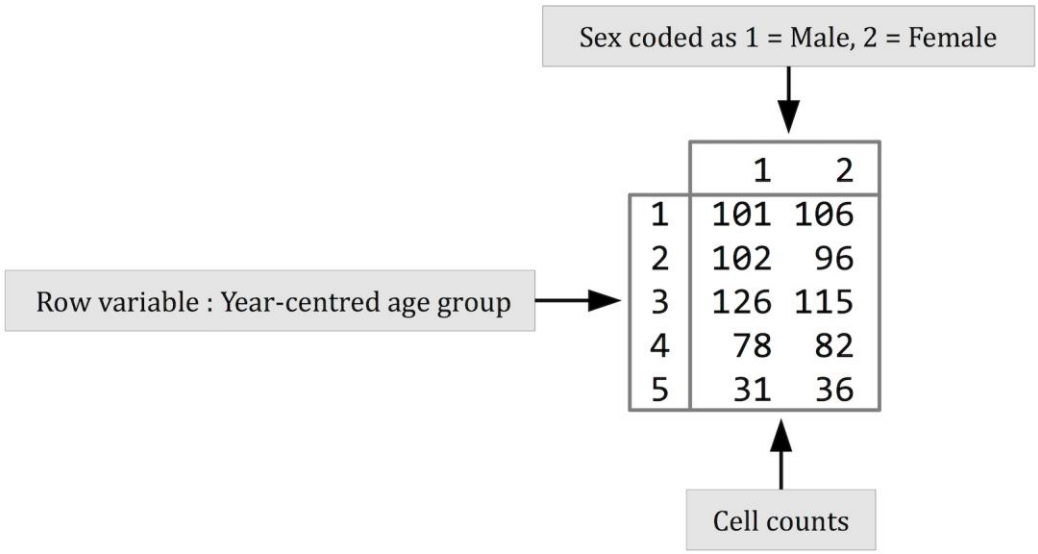
The NIPN data quality toolkit provides an *R* language function called **recode()** that makes it easy to recode and group any data. We will use the **recode()** function to group the data in the **age** variable (age in months) into year-centred age-groups.

```
svy$ycag <- recode(svy$age, "6:17=1; 18:29=2; 30:41=3; 42:53=4; 54:59=5")
head(svy)
```

A tabular analysis can be performed:

```
table(svy$ycag, svy$sex)
prop.table(table(svy$ycag, svy$sex)) * 100
```

The **table()** function performs a cross-tabulation. The first variable specified (**svy$ycag** in this example) is the row variable. The second variable specified (**svy$sex** in this example) is the column variable.



	1	2
1	101	106
2	102	96
3	126	115
4	78	82
5	31	36

It is useful to examine row percentages and column percentages in tables of age-group by sex.

We should look at row percentages:

```
prop.table(table(svy$ycag, svy$sex), margin = 1) * 100
```

This returns:

```
          1        2
1 48.79227 51.20773
2 51.51515 48.48485
3 52.28216 47.71784
4 48.75000 51.25000
5 46.26866 53.73134
```

Which shows approximately equal proportions of males and females in each year-centred age-group. We specified **`margin = 1`** with the **`prop.table()`** function because we wanted row percentages.

We should also look at column percentages:

```
prop.table(table(svy$ycag, svy$sex), margin = 2) * 100
```

This returns:

```
          1         2
1 23.059361 24.367816
2 23.287671 22.068966
3 28.767123 26.436782
4 17.808219 18.850575
5  7.077626  8.275862
```

We specified **`margin = 2`** with the **`prop.table()`** function because we wanted column percentages. We expect there to be approximately equal proportions of children in the age-groups centred at 1, 2, 3, and 4 years and a smaller proportion (i.e. about half that in the other age-groups) in the age-group centred at 5 years.

A graphical analysis using a *population pyramid* can be useful. The NIPN data quality toolkit provides an *R* language function called **`pyramid.plot()`** for plotting population pyramids:

```
pyramid.plot(svy$ycag, svy$sex)
```

We can make the plot more informative by specifying a title and axis labels:

```
pyramid.plot(svy$ycag, svy$sex, main = "Distribution of age by sex",
      xlab = "Frequency (Males | Females)", ylab = "Year-centred age-group")
```

and applying shading:

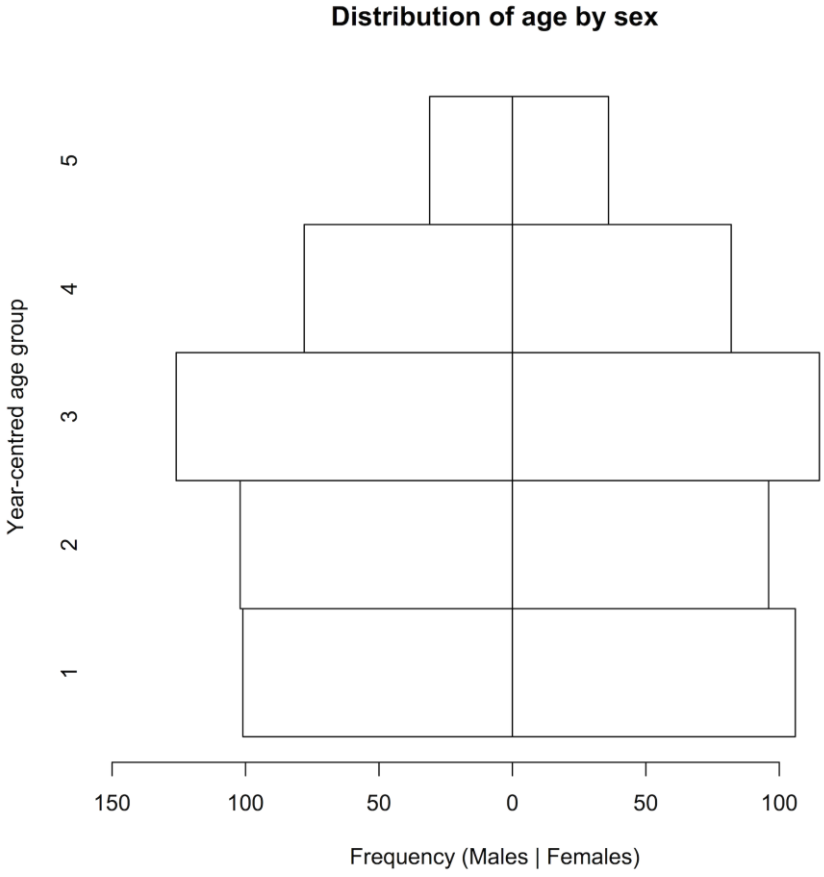
```
pyramid.plot(svy$ycag, svy$sex, main = "Distribution of age by sex",
      xlab = "Frequency (Males | Females)", ylab = "Year-centred age-group",
            col = c("grey80", "white"))
```

or colours:

```
pyramid.plot(svy$ycag, svy$sex, main = "Distribution of age by sex",
      xlab = "Frequency (Males | Females)", ylab = "Year-centred age-group",
            col = c("pink","lightblue"))
```

We expect there to be approximately equal numbers of children in the age-groups centred at 1, 2, 3, and 4 years and a smaller number (i.e. about half the number in the other age-groups) in the age-group centred at 5 years. There should also be approximately equal numbers of males and females. This is what we see in the population pyramid (see *Figure 5.2*).

Figure 5.2 Pyramid plot showing distribution of (grouped) age by sex.



The **pyramid.plot()** function uses the values of the grouped age variable as *y*-axis value labels.

We can assign descriptive text values using the **recode()** function. For example:

```
svy$ageLabel <- recode(svy$age, "6:29='< 30 months'; 30:hi='30 month or older'")
pyramid.plot(svy$ageLabel, svy$sex, main = "Distribution of age by sex",
      xlab = "Frequency (Males | Females)", ylab = "Age-group")
```

We can also use a *factor* type variable. This type of variable allows labels to be specified:

```
svy$ageLabel <- factor(svy$ycag,
      labels = c("6:17", "18:29", "30:41", "42:53", "54:59"))
pyramid.plot(svy$ageLabel, svy$sex, main = "Distribution of age by sex",
      xlab = "Frequency (Males | Females)", ylab = "Year-centred age-group")
```

The **cut()** function may also be used:

```
svy$ageCuts <- cut(svy$age, breaks = c(0, 17, 29, 41, 53, 59))
pyramid.plot(svy$ageCuts, svy$sex, main = "Age-group (months) ",
      xlab = "Frequency (Males | Females)", ylab = "Year-centred age-group",
            cex.names = 0.9)
```

The **cut()** function is a versatile grouping function. It is explained in more detail later in this section.

The **cex.names** parameter of the **pyramid.plot()** function allows us to change the size of the value labels on the y-axis. The value for **cex.names** is a magnification factor. Values above one make the labels larger than the default; values below one make the labels smaller than the default.

5.1.2 Simple testing

It is possible to perform a formal test on the distribution of age-groups by sex.

A simple test is:

```
chisq.test(table(svy$ycag, svy$sex))
```

This yields:

```
        Pearson's Chi-squared test

data:  table(svy$ycag, svy$sex)
X-squared = 1.2675, df = 4, p-value = 0.8669
```

In this example the p-value is not below 0.05 so we accept the null hypothesis that there is no significant association between age and sex. This is an important test as it tests whether the distribution of ages is similar for males and females. It does not, however, test whether the age structure in the sample meets expectations. This requires a test that compares the observed numbers with the expected numbers derived from an external source such as census data or from a demographic model.

5.1.3 A model of the expected age structure

A simple model-based method for calculating expected numbers is *exponential decay* in a population in which births and deaths balance each other and with a 1:1 male to female sex ratio. Under this model the proportion surviving in each group at each year can be calculated as:

$$p = e^{-zt}$$

in which *e* is the base of the natural logarithm (approximately 2.7183), *z* is the mortality rate associated with each time period, and *t* is time. Time (*t*) starts at zero for the purposes of computation. Age can be used as a measure of time since birth. We should use **0** for the first year-centred age-group, **1** for the second year-centred age-group, and so-on. This is the rationale for using **t <- 0:4** below.

With five year-centred age-groups and a mortality rate of 1 / 10,000 / day, the expected proportions surviving at each year can be calculated as:

```
z <- (1 / 10000) * 365.25
t <- 0:4
p <- exp(-z * t)
p
```

This yields the following survival probabilities:

```
1.0000000 0.9641340 0.9295544 0.8962149 0.8640713
```

We need to specify the duration (i.e. the number of years) represented by each age-group:

```
d <- c(1, 1, 1, 1, 0.5)
```

We can then calculate expected proportions of children in each age-group:

```
ep <- d * p / sum(d * p)
ep
```

This gives:

```
0.2368580 0.2283628 0.2201724 0.2122757 0.1023311
```

We can now calculate expected numbers:

```
expected <- ep * sum(table(svy$ycag))
names(expected) <- 1:5
expected
```

giving:

```
        1         2         3         4         5
206.77703 199.36076 192.21049 185.31667  89.33505
```

A formal test would compare the observed numbers with the expected numbers. The observed numbers can be found using:

```
observed <- table(svy$ycag)
observed
```

This gives:

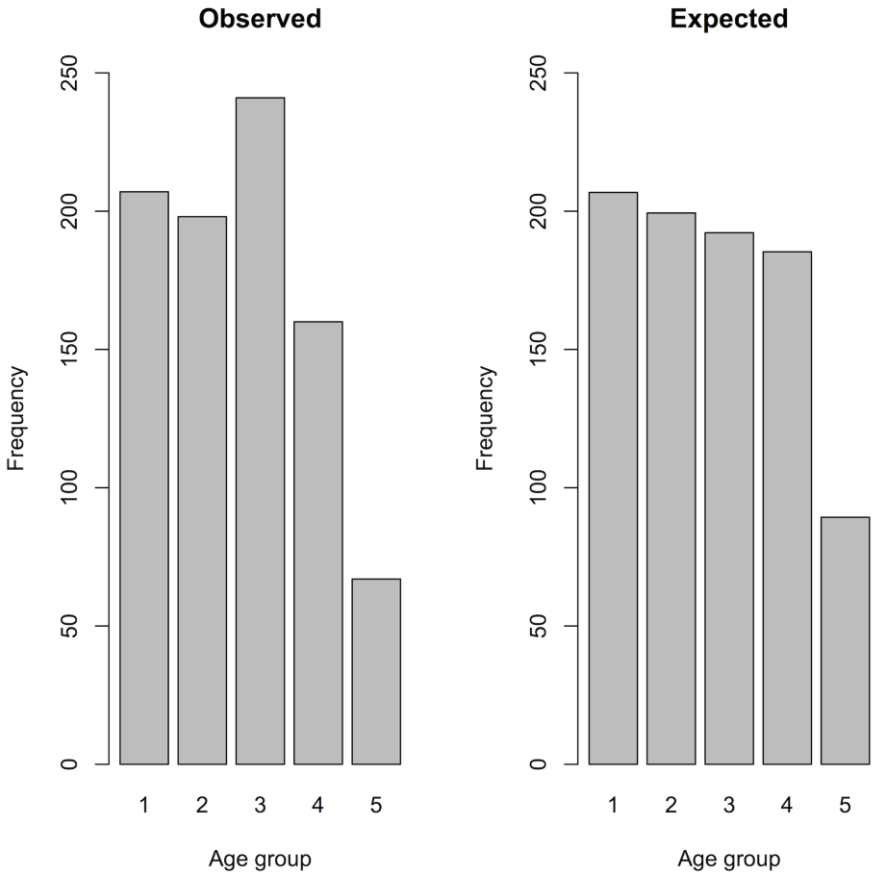
```
  1   2   3   4   5
207 198 241 160  67
```

It can be useful to examine observed and expected numbers graphically:

```
par(mfcol = c(1, 2))
barplot(observed, main = "Observed", xlab = "Age group", ylab = "Frequency",
      ylim = c(0, 250))
barplot(expected, main = "Expected", xlab = "Age group", ylab = "Frequency",
      ylim = c(0, 250))
```

See *Figure 5.3*.

Figure 5.3 Observed and expected numbers of children in each age-group under the assumptions of a uniform sex ratio, no population growth, exponential decay, and a mortality rate of one death per ten thousand children per day (1 / 10,000 / day).



We can calculate a Chi-squared test statistic:

$$\chi^2 = \sum \frac{(observed - expected)^2}{expected}$$

using:

```
X2 <- sum((observed - expected)^2 / expected)
X2
```

which yields a Chi-Squared test statistic of:

```
21.43662
```

We can find the p-value using:

```
pchisq(X2, df = 4, lower.tail = FALSE)
```

This gives:

```
0.000259395
```

In this example the age distribution is significantly different from the expected numbers calculated using a simple demographic model.

Note that we specify the degrees of freedom (**df**) for the Chi-Squared test as the number of age-groups minus one. As we have five age-groups we specify **df = 4**. The degrees of freedom (**df**) that we need to specify will depend on the number of age-groups that we use. It is always the number of age-groups minus one. If, for example, there are ten age-groups we would need to specify **df = 9**.

The NIPN data quality toolkit provides an *R* function called **ageChildren()** that performs the model-based Chi-Squared test:

```
ageChildren(svy$age, u5mr = 1)
```

which returns:

```
        Age Test (Children)

X-squared = 21.4366, df = 4, p = 0.0003
```

Note that we specified the under five years mortality rate as 1 / 10,000 / day using **u5mr = 1**. Another, more appropriate, rate may be specified.

The **ageChildren()** function calculates year-centred age-groups for children aged between six and fifty-nine months by default. This is a standard survey population and is used in SMART and many other surveys. The use of year-centred age-groups is also standard practice. The commands that are given above can, however, be adapted for use with different age-groups.

The output of the **ageChildren()** function can be saved for later use:

```
ac <- ageChildren(svy$age, u5mr = 1)
```

The saved output contains the Chi-squared test results and tables of observed and expected values.

These can be accessed using:

```
ac
ac$X2
ac$df
ac$p
ac$observed
ac$expected
```

The saved results may also be plotted:

```
plot(ac)
```

The **ageChildren()** function can be applied to each sex separately.

To males:

```
acM <- ageChildren(svy$age[svy$sex == 1], u5mr = 1)
acM
plot(acM)
```

and to females:

```
acF <- ageChildren(svy$age[svy$sex == 2], u5mr = 1)
acF
plot(acF)
```

An easier way of doing this is:

```
by(svy$age, svy$sex, ageChildren, u5mr = 1)
```

The test statistics should be interpreted with caution. A significant test result may, for example, be due to the use of an inappropriate model to generate the expected numbers.

A significant result in this particular test may be due to:

> **Specifying an inappropriate under five years mortality rate.** This is a particular problem because the specified under five years mortality rate is assumed to have applied for five years prior to data being collected.
>
> **The assumption of a 1:1 male to female sex ratio.** This is a particular problem in settings in which there is sex-selective abortion and sex-selective infanticide.

The model is crude. Mortality is related to age. Younger children have a greater mortality risk than older children but an average mortality rate for children under five years is used in the calculations. A more sophisticated model could be used but, in many settings, we will not have the data required to apply such a model.

It should also be noted that the sample sizes used in most survey can cause tests to yield statistically significant results for small differences between observed and expected numbers.

5.1.4 Use of census data

The use of simple demographic models is far from ideal. It is usually better to calculate the expected proportions from census data. A useful source of census data is the United States Census Bureau's International Data Base:

```
https://www.census.gov/population/international/data/idb/informationGateway.php
```

The population in single year age-groups for 0, 1, 2, 3, and 4 years for Afghanistan in 2015 was:

Age	Both Sexes	Males	Females
0	1,148,379	584,276	564,103
1	1,062,635	539,589	523,046
2	1,015,688	515,793	499,895
3	981,288	498,365	482,923
4	950,875	482,926	467,949

We can calculate expected values from these data:

```
pop <- c(1148379, 1062635, 1015688, 981288, 950875)
ep <- pop / sum(pop)
```

With a sample size of n = 900 the expected number in each age-group would be:

```
expected <- ep * 900
expected
```

These expected values can be used in a Chi-squared test as is illustrated above.

Census data may also be used to estimate the under five years' mortality rate (U5MR) which can then be used with the **`ageChildren()`** function.

The model of *exponential decay* in a population in which births and deaths balance each other with a 1:1 male to female sex ratio:

$$p = e^{-zt}$$

means that we can, given an age-distribution, estimate mortality by fitting the model:

$$log_e(n) = \alpha + \beta t$$

where n is the count of children in each age-group.

The absolute value of the β coefficient is the point estimate of the mortality rate (z). Using the 2015 population data for Afghanistan:

```
t <- 0:4
lm(log(pop) ~ t)
```

This gives:

```
Coefficients:
(Intercept)         t
13.93601     -0.04571
```

The value reported under **`t`** is the β coefficient (**`-0.04571`**). The absolute value of the β coefficient (i.e. the value without the sign) is **`0.04571`**. This is the point estimate of the mortality rate. Expressed as the number of deaths / 10,000 persons / day:

```
(0.04571 / 365.25) * 10000
```

this is:

```
1.251472
```

We can use this estimate with the **`ageChildren()`** function:

```
ageChildren(svy$age, u5mr = 1.251472)
```

5.1.5 The age ratio

A much simpler and less problematic age-related test of survey and data quality is the *age ratio* test.

The age ratio is defined as:

$$Age\ ratio = \frac{number\ of\ children\ aged\ between\ 6\ and\ 29\ months}{number\ of\ childen\ aged\ between\ 30\ and\ 59\ months}$$

We will use the **`recode()`** function from NIPN data quality toolkit to create the relevant age-groups:

```
svy$ageGroup <- recode(svy$age, "6:29=1; 30:59=2")
head(svy)
```

The observed age ratio is:

```
sum(svy$ageGroup == 1) / sum(svy$ageGroup == 2)
```

which gives:

```
0.8653846
```

It is often easier to work with proportions than with ratios, so we only need to calculate the proportion in the younger age-group:

```
sum(svy$ageGroup == 1) / sum(table(svy$ageGroup))
```

which gives:

```
0.4639175
```

We can calculate an expected value using census data or a simple demographic model. The simplest approach is to use a standard value. SMART surveys often use the ratio 0.85:1 (SMART, 2015, p11).

We only need to calculate the expected proportion in the younger group. For the ratio 0.85:1 this is:

```
p <- 0.85 / (0.85 + 1)
p
```

This gives:

```
0.4594595
```

The observed proportion (0.4639175) and expected proportion (0.4594595) are so similar that a formal test of statistical significance is not required in this case.

Formal testing can be done using a Chi-squared test:

```
prop.test(sum(svy$ageGroup == 1), sum(table(svy$ageGroup)), p = 0.4594595)
```

This returns:

```
        1-sample proportions test with continuity correction

data:  sum(svy$ag == 1) out of sum(table(svy$ag)), null probability 0.4594595
X-squared = 0.053062, df = 1, p-value = 0.8178
alternative hypothesis: true p is not equal to 0.4594595
95 percent confidence interval:
 0.4304994 0.4976573
sample estimates:
        p
0.4639175
```

The age ratio in the example data is not significantly different from the expected age ratio.

The NIPN data quality toolkit provides an *R* function called **`ageRatioTest()`** that performs the age ratio test:

```
ageRatioTest(svy$age, ratio = 0.85)
```

This returns:

```
                    Age Ratio Test

                     Expected age ratio = 0.8500
Expected proportion aged 6 - 29 months = 0.4595

                     Observed age ratio = 0.8654
Observed proportion aged 6 - 29 months = 0.4639

X-squared = 0.0531, p = 0.8178
```

The **`ratio`** parameter of the **`ageRatioTest()`** function allows you to specify an expected age ratio other than 0.85:1.

Note that the **`ageRatioTest()`** function applies the test to data from children aged between 6 and 59 months only, all other ages are ignored.

The age ratio test might be applied to data from both sexes (as above) and to each sex separately:

```
by(svy$age, svy$sex, ageRatioTest, ratio = 0.85)
```

The example data meets expectations regarding the age ratio for all children and for male and female children separately.

5.2 Age and sex distributions: adults and general population surveys

A key test of survey quality is whether the survey data represents the population in terms of the age and sex distribution. We can test this by comparison with census data.

We will retrieve some example data:

```
svy <- read.table("as.ex01.csv", header = TRUE, sep = ",")
head(svy)
```

These data are taken from household rosters collected as part of a household survey in Tanzania.

We will use census data taken from the Wolfram|Alpha knowledge engine:

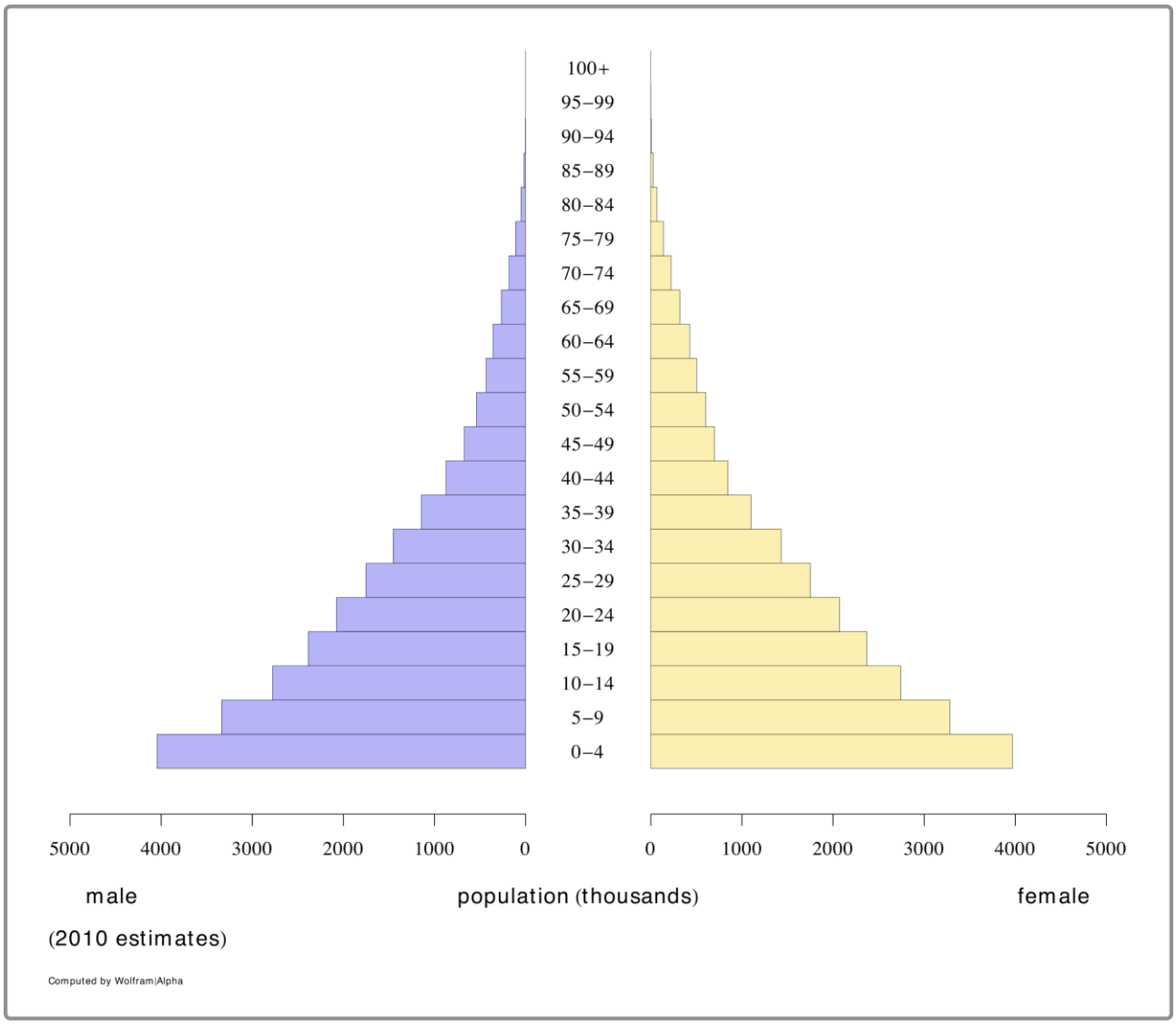
```
http://www.wolframalpha.com/input/?i=Tanzania+age+distribution
```

Another useful source of census data is the United States Census Bureau's International Data Base:

```
https://www.census.gov/population/international/data/idb/informationGateway.php
```

The pyramid plot produced by Wolfram|Alpha is shown in *Figure 5.4*.

Figure 5.4 Pyramid plot of the age and sex distribution in Tanzania computed by the `Wolfram|Alpha` knowledge engine (2010 estimates)



The table produced by `Wolfram|Alpha` was downloaded and stored in a CSV file:

```
ref <- read.table("as.ex02.csv", header = TRUE, sep = ",")
ref
```

The age-groups are expressed using the form specified in ISO 31-11, an international standard that applies to mathematical symbols. The form `[a,b)` expresses the interval $a \leq x < b$. For example, `[30,35)` is used to indicate the set `{30, 31, 32, 33, 34}` of ages in years. The form `[a,b)` is said to be *closed* on the left and *open* on the right.

The reference data (**`ref`**) uses five-year age-groups. We will create the same age-groups in the example dataset.

We should first check the range of ages in the example data:

```
range(svy$age)
```

which returns:

```
0 93
```

The *R* language provides a function that makes it easy to create ISO 31-11 groupings from raw data:

```
svy$ageGroup <-cut(svy$age, breaks = seq(from = 0, to = 95, by = 5),
      include.lowest = TRUE, right = FALSE)
```

Using **`include.lowest = TRUE`** tells the **`cut()`** function to include the lowest **`breaks`** value (zero in this case). Using **`right = FALSE`** tells the **`cut()`** function to use groupings that are closed on the left. This combination of parameters creates the same "closed on the left" and "open on the right" age-groups as are used in the reference (**`ref`**) data:

```
table(svy$ageGroup)
```

A tabular analysis of age-group by sex can be produced using:

```
table(svy$ageGroup, svy$sex)
```

A visual inspection is useful:

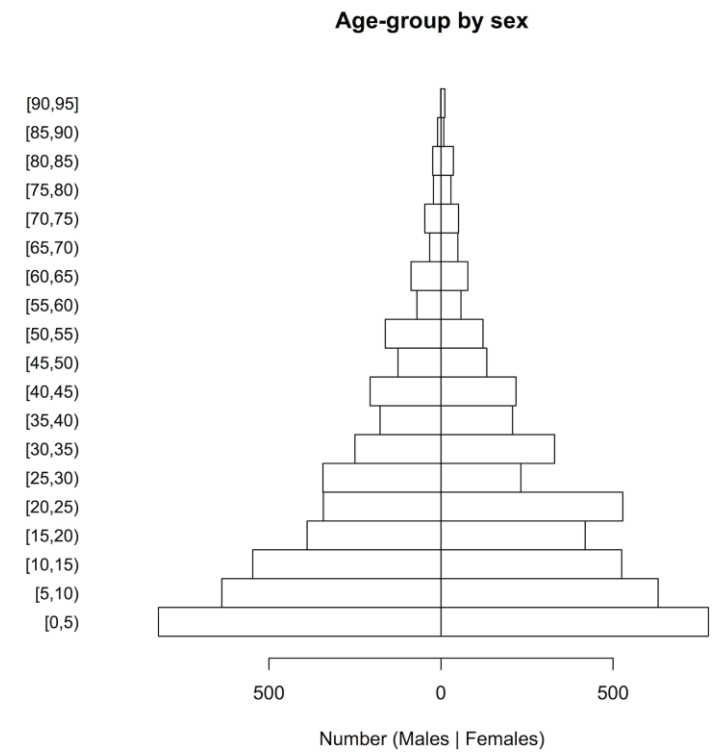
```
pyramid.plot(svy$ageGroup, svy$sex)
```

We can make this easier to read:

```
pyramid.plot(svy$ageGroup, svy$sex, main = "Age-group by sex",
      xlab = "Number (Males | Females)", ylab = "", las = 1, cex.names = 0.9)
```

See *Figure 5.5*.

Figure 5.5 Pyramid plot of the age and sex distribution in the Tanzania survey dataset.



Note that we specified **`ylab = ""`** because it is clear that the category labels represent age-groups and to prevent the y-axis label from obscuring the category labels, which happens with:

```
pyramid.plot(svy$ageGroup, svy$sex, main = "Age-group by sex",
      xlab = "Number (Males | Females)", ylab = "Age-group", las = 1,
            cex.names = 0.9)
```

It is possible to alter the number of lines of text in margins of the plot, reduce the size of the age-group labels, and place the y-axis label on a specific line in the left margin of the plot in order to make a clearer plot:

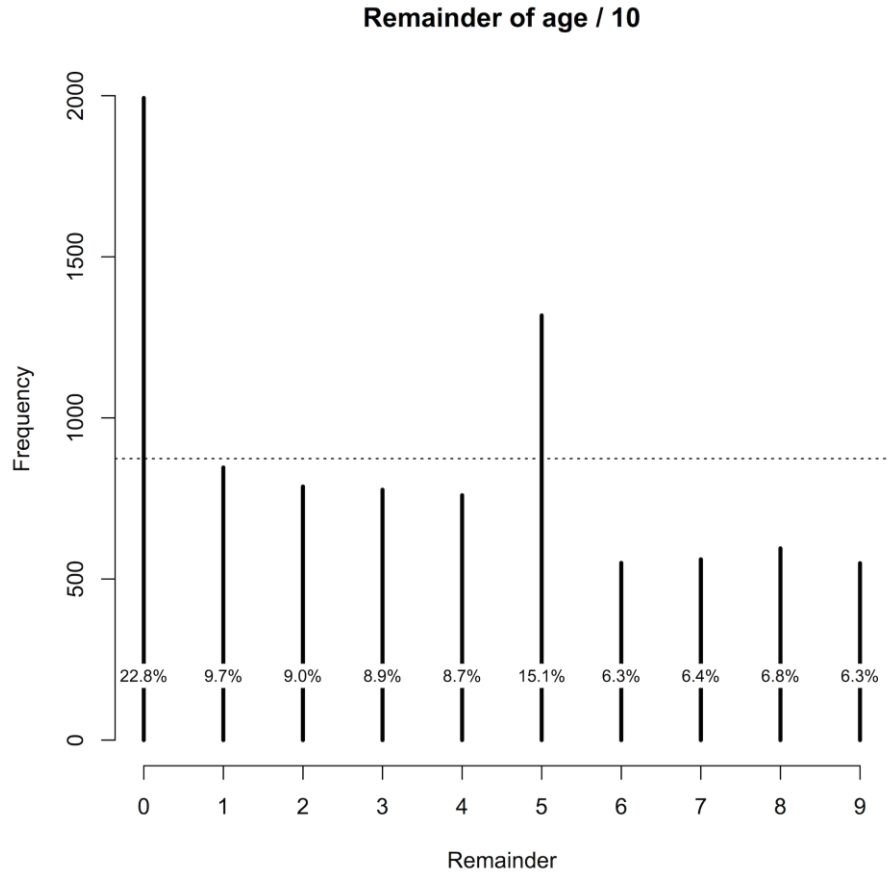
```
par(mar = c(5, 5, 4, 2))
pyramid.plot(svy$ageGroup, svy$sex, main = "Age-group by sex",
      xlab = "Number (Males | Females)", ylab = "", las = 1, cex.names = 0.8)
title(ylab = "Age-group", line = 4)
```

The easiest way of checking whether the survey data represents the general population in terms of the age and sex distribution is to compare the observed (*Figure 5.5*) and expected (*Figure 5.4*) distributions. The general shapes of the two distributions are similar. Some of the lumpiness in *Figure 5.5* is due to age heaping in the adult ages at decades and half-decades:

```
ah <- ageHeaping(svy$age, divisor = 10)
plot(ah, main = "Remainder of age / 10")
```

See *Figure 5.6.*

Figure 5.6 Age heaping in the Tanzania dataset. Age heaping at 20, 30, 40, 50, 60, 70, and 80 years can also be seen in *Figure 5.5* and *Figure 5.7*.



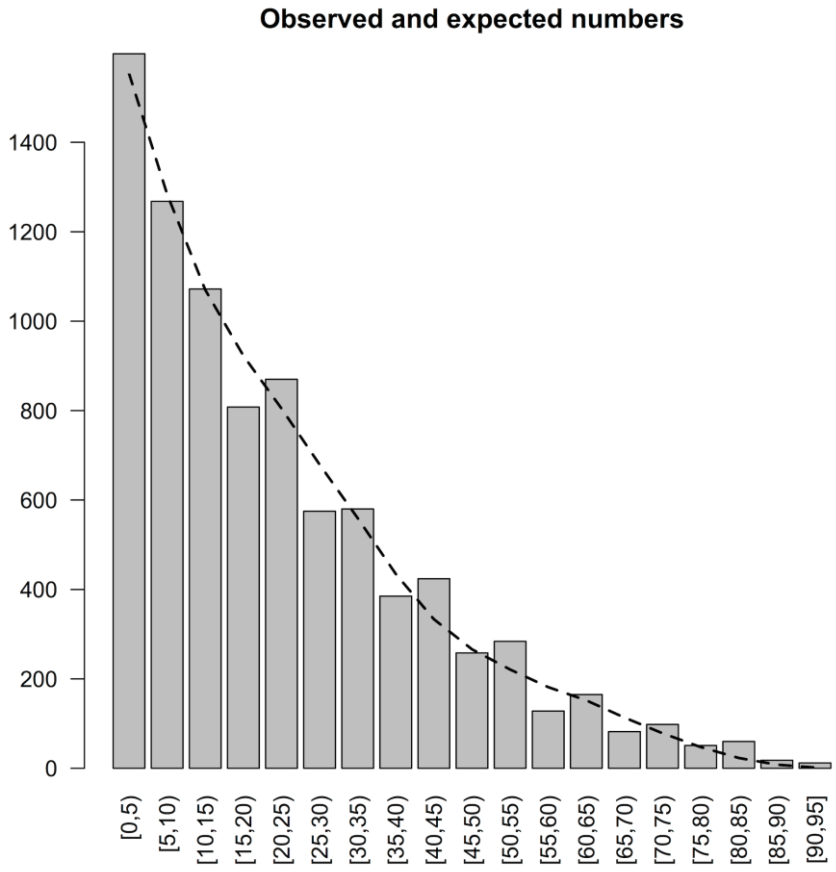
A more formal test of the age structure can be made by comparing observed and expected numbers.

We can do this graphically:

```
ref <- ref[1:19, ]
expectedProportions <- ref$All / sum(ref$All)
expectedNumbers <- expectedProportions * sum(table(svy$ageGroup))
mp <- barplot(table(svy$ageGroup), main = "Observed and expected numbers",
      ylim = c(0, max(expectedNumbers)), las = 2)
lines(mp, expectedNumbers, lty = 2, lwd = 2)
```

The observed and expected numbers are similar to each other. The lumpiness in the observed numbers is due to age heaping. See *Figure 5.7*.

Figure 5.7 Observed and expected numbers in the Tanzania survey dataset. Bars show observed numbers. The dashed line shows expected numbers.



Formal testing can be performed:

```
chisq.test(table(svy$ageGroup), p = expectedProportions)
```

This gives:

```
        Chi-squared test for given probabilities

data:  table(svy$ag)
X-squared = 248.41, df = 18, p-value < 2.2e-16

Warning message:
In chisq.test(table(svy$ag), p = expectedProportions) :
    Chi-squared approximation may be incorrect
```

The warning message is due to small expected numbers (i.e. $n < 5$) in the older age-groups. *R* provides a more robust "Monte Carlo" test:

```
chisq.test(table(svy$ageGroup), p = expectedProportions, simulate.p.value = TRUE)
```

This may take a few seconds to compute and yields:

```
        Chi-squared test for given probabilities ...

data:  table(svy$ag)
X-squared = 248.41, df = NA, p-value = 0.0004998
```

The test results need to be interpreted with caution. The sample size ($n = 8736$) is large in this example. This means that small differences, which may be due to age heaping, become statistically significant. This test cannot be considered to be good evidence that the age-structure of the sample differs from the expected age-structure of the population.

We also need to examine the sex ratio of the sample. A sex ratio test can be performed using the **`sexRatioTest()`** function from the NIPN data quality toolkit and the sex ratio observed in the census data:

```
censusM <- sum(ref$Males)
censusF <- sum(ref$Females)
sexRatioTest(svy$sex, codes = c(1, 2), pop = c(censusM, censusF))
```

This yields:

```
        Sex Ratio Test

Expected proportion male = 0.4988
Observed proportion male = 0.4914
X-squared = 1.8770, p = 0.1707
```

There is no evidence that the sex ratio in the sample differs much from the expected sex ratio in the population.

The techniques outlined in this section are illustrative. This is because many surveys, other than nutritional anthropometry surveys in young children, are not standardised. A survey may sample only women of child-bearing age, so the sample may be restricted to women aged between 15 and 45 years. In this case the age-structure can be examined using the techniques outlined above but it would make no sense to examine the sex ratio. Care should be taken when examining data from surveys that may have deliberately oversampled specific age-groups.

6. Digit preference in anthropometric measurements

Measurements in nutritional anthropometry surveys are usually taken and recorded to one decimal place. Examples are given in *Table 6.1*.

Table 6.1 Common measurements used in anthropometric surveys.

Variable	Unit	Precision	Example	Notes
Weight	kg	Nearest 0.1 kg	8.7 kg	Most surveys use scales with a 0.1 kg precision.
Height / length	cm	Nearest 0.1 cm	85.3 cm	Height boards tend to have a 0.1 cm precision.
MUAC	cm	Nearest 0.1 cm	13.7 cm	MUAC may be measured and recorded in centimetres or millimetres. Sometimes both may be used in the same survey. You will need to check this and recode data to use only a single format / precision.
	mm	Nearest 1 mm	137 mm	

Digit preference is the observation that the final number in a measurement occurs with a greater frequency that is expected by chance. This can occur because of rounding, which is the practice of increasing or decreasing the value in a measurement to the nearest whole or half unit, or because data are made up.

When taking and recording measurements in the field it is common for field staff to round the first value after the decimal point to zero or five. Measurements in whole numbers may also be rounded to the nearest decade (e.g. 137 mm may be rounded to 140 mm) or half-decade (e.g. 137 mm may be rounded to 135 mm). A small number of rounded measurements is unlikely to affect survey results. A large number of rounded measurements can affect survey results particularly if measurements have been systematically rounded in one direction. This is a form of bias.

Made up data often shows digit preference with (e.g.) ”2” and “6” appearing as final digits much more frequently than expected. This happens because, without using a computer, a large quantity of random data is very much harder to make up than merely random-looking data.

If there were little or no digit preference in anthropometric data then we would expect the final recorded digit of each measurement to occur with approximately equal frequency. We can check if digit preference is absent in data by testing whether this is the case.

6.1 Tabulation and visualisation

First we will work with some artificial data:

```
set.seed(0)
finalDigits <- sample(x = 0:9, size = 1000, replace = TRUE)
```

The use of **`set.seed()`** resets the pseudorandom number generator. This ensures that the results shown here are the same as you will get when you follow the example analyses.

You should always examine data before performing any formal tests.

A table can be useful:

```
table(finalDigits)
```

This returns:

```
finalDigits
  0   1   2   3   4   5   6   7   8   9
 96 104  91 113 115  85  90 107  89 110
```

We can look at proportions instead of counts:

```
prop.table(table(finalDigits))
```

This returns:

```
    0     1     2     3     4     5     6     7     8     9
0.096 0.104 0.091 0.113 0.115 0.085 0.090 0.107 0.089 0.110
```

If you prefer working with percentages then:

```
prop.table(table(finalDigits)) * 100
```

returns:

```
   0    1    2    3    4    5    6    7    8    9
 9.6 10.4  9.1 11.3 11.5  8.5  9.0 10.7  8.9 11.0
```

Examining data graphically is very useful:

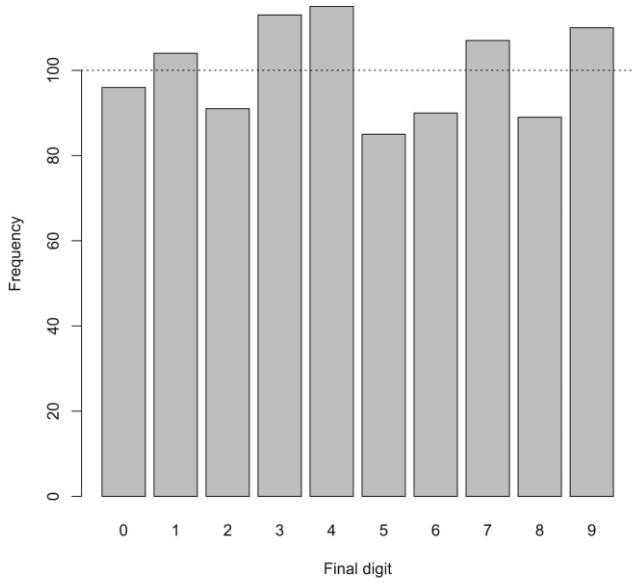
```
barplot(table(finalDigits), xlab = "Final digit", ylab = "Frequency")
```

We can add a line showing our expectation that each final digit should occur about 10% of the time:

```
abline(h = sum(table(finalDigits)) / 10, lty = 3)
```

The resulting plot is shown in *Figure 6.1*.

Figure 6.1. Example (generated) data with little or no digit preference. The dotted line shows expected values.



The tabular and graphical analyses are consistent with there being little or no digit preference in the generated data.

Both analyses agree with the expectation that each final digit should occur about 10% of the time.

All we are seeing is random variation.

We can use a formal test to confirm this:

```
chisq.test(table(finalDigits))
```

This returns:

```
X-squared = 11.02, df = 9, p-value = 0.2743
```

In this example the *p-value* is not below 0.05 so we accept the *null hypothesis* that there is no digit preference.

It is important to check that each digit between zero and nine is represented in tables and plots.

Missing digits can indicate strong digit preference.

The NIPN data quality toolkit provides the **`fullTable()`** function. This *R* language function produces a table that includes cells with zero counts.

As an example we will remove all the values with a final digit equal to **6** from our generated data:

```
finalDigits[finalDigits == 6] <- NA
```

and see the effect:

```
table(finalDigits)
prop.table(table(finalDigits)) * 100
barplot(table(finalDigits), xlab = "Final digit", ylab = "Frequency")
abline(h = sum(table(finalDigits)) / 10, lty = 3)
chisq.test(table(finalDigits))
```

This is a misleading analysis. It is very easy to miss that there are no final digits equal to **6** in the data. The plot is misleading because the final digit **6** is not represented and we assumed that there were ten rather than nine final digits when we calculated the expected frequencies. The Chi-squared test is not correct because it does not account for there being zero cases in which the final digit is equal to **6**.

The **`fullTable()`** function avoids these issues:

```
fullTable(finalDigits)
prop.table(fullTable(finalDigits)) * 100
barplot(fullTable(finalDigits), xlab = "Final digit", ylab = "Frequency")
abline(h = sum(fullTable(finalDigits)) / 10, lty = 3)
chisq.test(fullTable(finalDigits))
```

The Chi-squared test (incorrectly) calculated without the zero cell:

```
X-squared = 9.8, df = 8, p-value = 0.2793
```

indicates that there is no problem with the data.

The chi-square test (correctly) calculated with the zero cell:

```
X-squared = 112, df = 9, p-value < 2.2e-16
```

indicates that there is a problem with the data.

Note that we use **`sum(fullTable(finalDigits)) / 10`** (i.e. we divide by ten) because we know that there should be ten final digits (i.e. **`0`**, **`1`**, **`2`**, **`3`**, **`4`**, **`5`**, **`6`**, **`7`**, **`8`**, **`9`**).

There is an issue with using hypothesis test such as the chi-squared test. Test values are strongly influenced by the sample size and can yield *false-negative* results when used with small sample sizes and *false-positive* results when used with large sample sizes.

We can illustrate this by generating some new artificial data with marked digit preference:

```
finalDigits <- as.table(x = c(11, 7, 5, 4, 7, 11, 5, 4, 4, 2))
names(finalDigits) <- 0:9
```

This creates a table object containing counts of imaginary final digits.

Looking at these data:

```
finalDigits
prop.table(finalDigits) * 100
barplot(finalDigits, xlab = "Final digit", ylab = "Frequency")
abline(h = sum(finalDigits) / 10, lty = 3)
```

There is a marked digit preference for zero and five (*Figure 6.2*).

The Chi-squared test:

```
chisq.test(finalDigits)
```

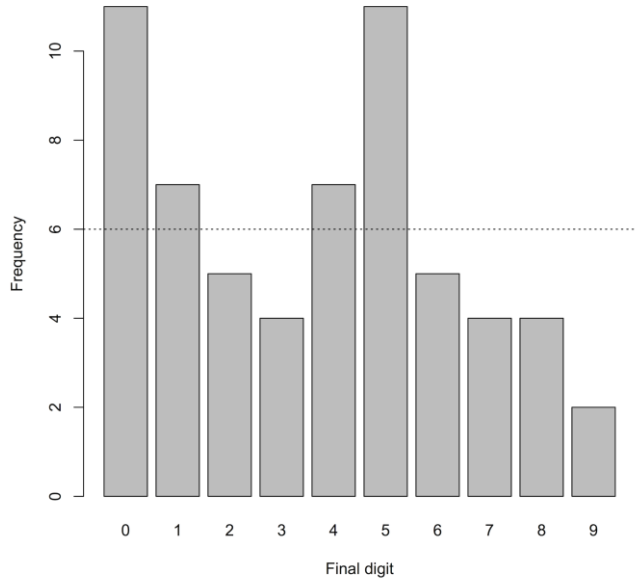
returns:

```
X-squared = 13.667, df = 9, p-value = 0.1347
```

In this example the Chi-squared test has failed to detect marked digit preference. This is a *false negative* test result. The failure of the Chi-squared test in this example is due to the small number of observations (i.e. $n = 60$) used in the analysis.

A tabular and graphical analysis was required to identify the digit preference problem in this example.

Figure 6.2 Example (generated) data with marked digit preference. The dotted line shows expected values.



We will usually be working with large sample sizes. This can bring the problem of *false positives*.

We will generate some artificial data:

```
set.seed(3)
finalDigits <- sample(x = 0:9, size = 1000, replace = TRUE)
```

These data will approximate the properties of a set of true uniformly random numbers.

Any digit preference that we might observe in these data is due solely to chance.

The generated data appear to exhibit some digit preference:

```
table(finalDigits)
prop.table(fullTable(finalDigits)) * 100
barplot(fullTable(finalDigits), xlab = "Final digit", ylab = "Frequency")
abline(h = sum(fullTable(finalDigits)) / 10, lty = 3)
```

but this digit preference is not especially marked. See *Figure 6.3*.

The Chi-squared test:

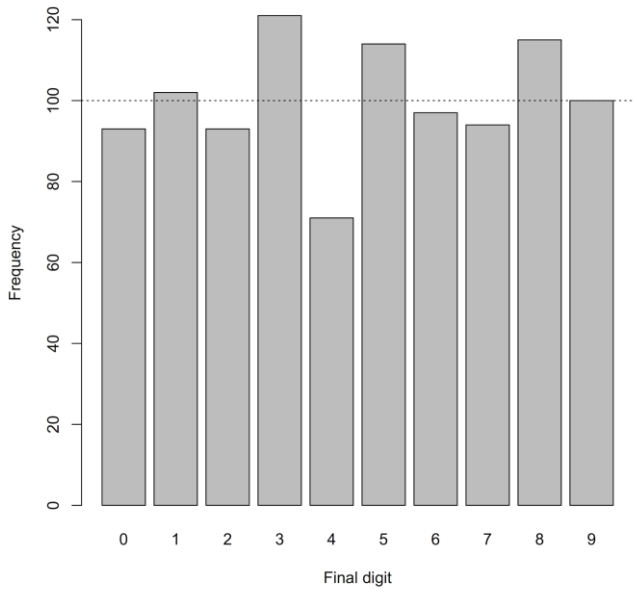
```
chisq.test(fullTable(finalDigits))
```

yields:

```
X-squared = 18.5, df = 9, p-value = 0.0298
```

which suggests significant digit preference.

Figure 6.3 Example (generated) data with little or no digit preference. The dotted line shows expected values.



This is a *false positive* result because the generated data is constrained to be uniformly random and any digit preference that we observed is due solely to chance.

The failure of the Chi-squared test in this example is due to the test mistaking random variation for digit preference which is, in part, due to the use of a large (i.e. $n = 1000$) number of observations.

It is also important to note that any test with a $p < 0.05$ significance threshold will generate a positive result in 1 in 20 tests of data that exhibit nothing but random variation. All tests with a $p < 0.05$ significance threshold have a 5% false positive rate.

6.2 Avoiding false positives using the digit preference score

The problem of false-positives can be addressed by using a summary measure that takes the effect of sample size into account. A widely used method is the *digit preference score* (DPS). The DPS was developed by the World Health Organization for the MONICA project:

http://www.thl.fi/publications/monica/bp/bpqa.htm

The DPS corrects the Chi-squared statistic (X^2) for the sample size (n) and the degrees of freedom (df) of the test:

$$DPS = 100 \times \sqrt{\frac{\chi^2}{n \times df}}$$

This has the effect of "desensitising" the Chi-squared test.

The DPS can be used with anthropometric data from all types of surveys and may also be applied to clinical data. A low DPS value indicates little or no digit preference; a high DPS value indicates considerable digit preference.

Guideline values for DPS are shown in *Table 6.2*.

Table 6.2 Guideline thresholds for the digit preference score (SMART, 2015)

DPS value	Interpretation
0 ≤ DPS < 8	Excellent
8 ≤ DPS < 12	Good
12 ≤ DPS < 20	Acceptable
DPS ≥ 20	Problematic

The NIPN data quality toolkit provides the *R* language function **`digitPreference()`** for calculating the DPS. Applying this function to the example data:

```
digitPreference(finalDigits, digits = 0)
```

yields:

```
       Digit Preference Score

data:  finalDigit
Digit Preference Score (DPS) = 4.53 (Excellent)
```

which is consistent with there being little or no digit preference in the example data.

The output of the **`digitPreference()`** function can be saved for later use:

```
dpsResults <- digitPreference(finalDigits, digits = 0)
```

The saved output contains the DPS value and frequency tables of the final digits (counts and percentages). These can be accessed using:

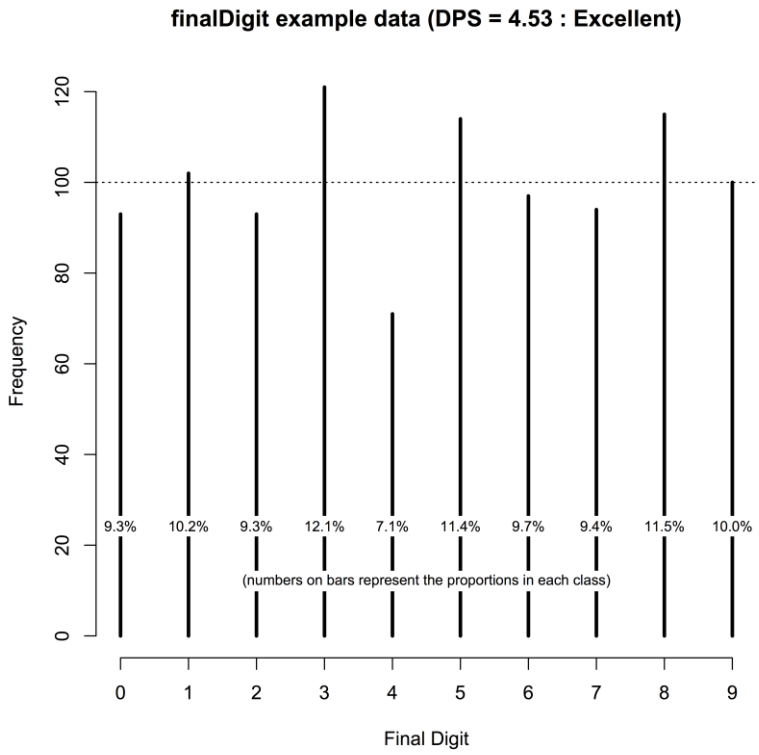
```
dpsResults$dps
dpsResults$tab
dpsResults$pct
dpsResults$dpsClass
```

The saved results may also be plotted:

```
plot(dpsResults, main = "finalDigit example data")
```

The resulting plot is shown in *Figure 6.4*.

Figure 6.4 An example DPS plot made using the **`digitPreference()`** function.



We will now practice using the **`digitPreference()`** function on survey data.

We will start by retrieving some survey data:

```
svy <- read.table("dp.ex01.csv", header = TRUE, sep = ",")
```

The file `dp.ex01.csv` is a comma-separated-value (CSV) file containing anthropometric data for children in a single state of a West African country in a Demographic and Health Survey (DHS).

The first few records in this dataset can be seen using:

```
head(svy)
```

This returns:

```
  psu age sex   wt   ht oedema
1 330  14   1  5.0 65.6      2
2 330  54   2 12.1 99.0      2
3 330  25   1  8.9 59.5      2
4 330  52   1 14.6 98.0      2
5 330  43   1 10.1 99.1      2
6 330   7   1  4.0 58.1      2
```

The two variables of interest are **`wt`** (weight) and **`ht`** (height).

We can examine digit preference in the variable for weight (**`wt`**) using:

```
digitPreference(svy$wt, digits = 1)
```

which returns:

```
        Digit Preference Score

data:  svy$wt
Digit Preference Score (DPS) = 11.86 (Good)
```

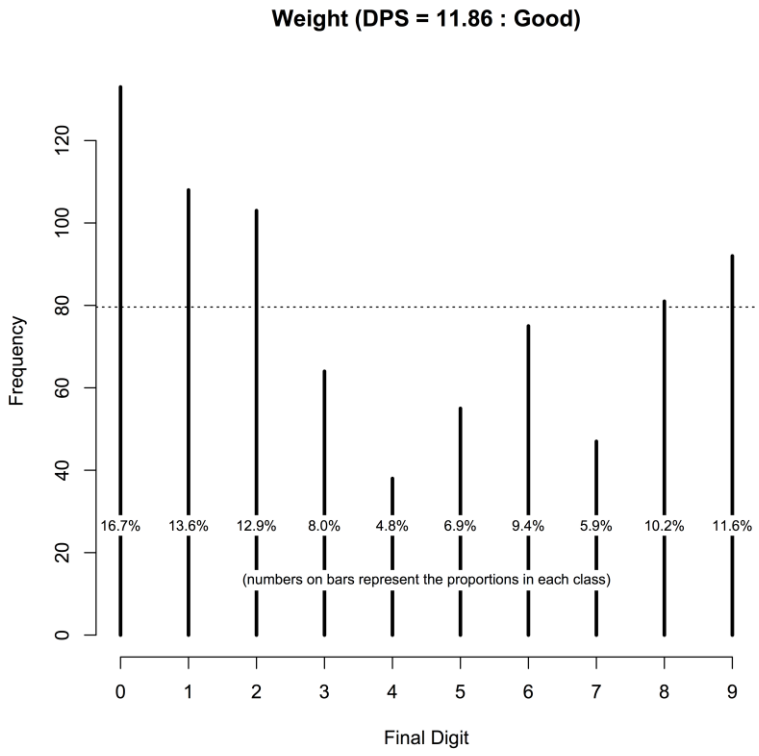
We can plot digit preference using:

```
plot(digitPreference(svy$wt, digits = 1), main = "Weight")
```

The resulting plot is shown in *Figure 6.5*.

The weight data shows some digit preference and would be classified as "Good" using the classifications shown in *Table 6.2* (above).

Figure 6.5 DPS for weight in part of a DHS survey



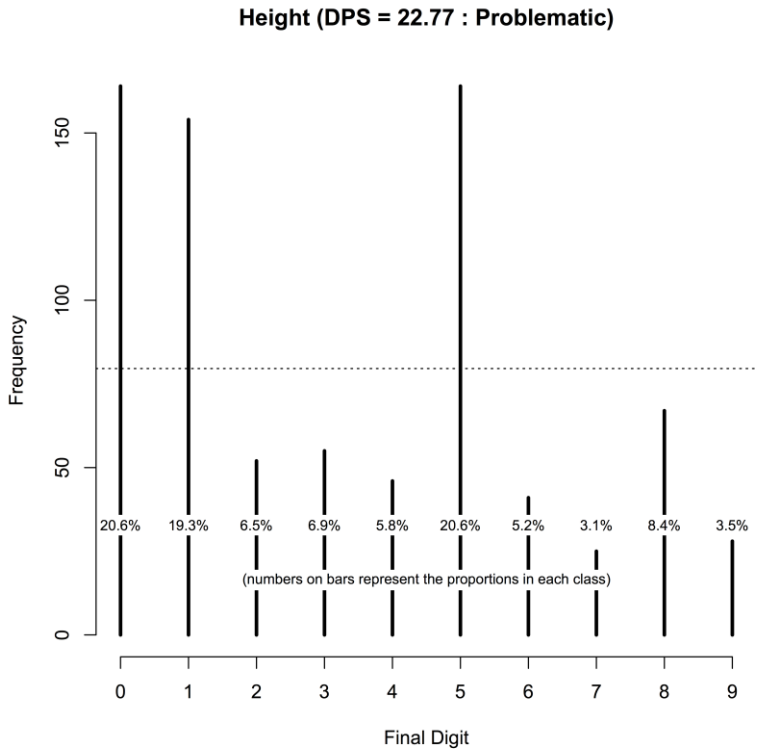
We can examine digit preference in the variable for height (**ht**) using:

```
digitPreference(svy$ht, digits = 1)
plot(digitPreference(svy$ht, digits = 1), main = "Height")
```

The DPS value (22.77) and the DPS plot (*Figure 6.6*) show considerable digit preference in the height (**ht**) variable. This would be classified as "Problematic" using the classifications shown in *Table 6.2* (above).

Note that we specified **digits = 1** when we used the **digitPreference()** function for the weight and height data in the example DHS data. This is because these variables are measured and recorded to one decimal place.

Figure 6.6 DPS for height in part of a DHS survey



If we were using the **`digitPreference()`** function with MUAC data that is measured and recorded as whole numbers (i.e. with no decimal places) then we should specify `digits = 0`. For example:

```
svy <- read.table("dp.ex02.csv", header = TRUE, sep = ",")
```

The file `dp.ex02.csv` is a comma-separated-value (CSV) file containing anthropometric data from a SMART survey in Kabul, Afghanistan.

The first few records in this dataset can be seen using:

```
head(svy)
```

which returns:

```
  psu age sex weight height muac oedema
1   1   6   1    7.3   65.0  146      2
2   1  42   2   12.5   89.5  156      2
3   1  23   1   10.6   78.1  149      2
4   1  18   1   12.8   81.5  160      2
5   1  52   1   12.1   87.3  152      2
6   1  36   2   16.9   93.0  190      2
```

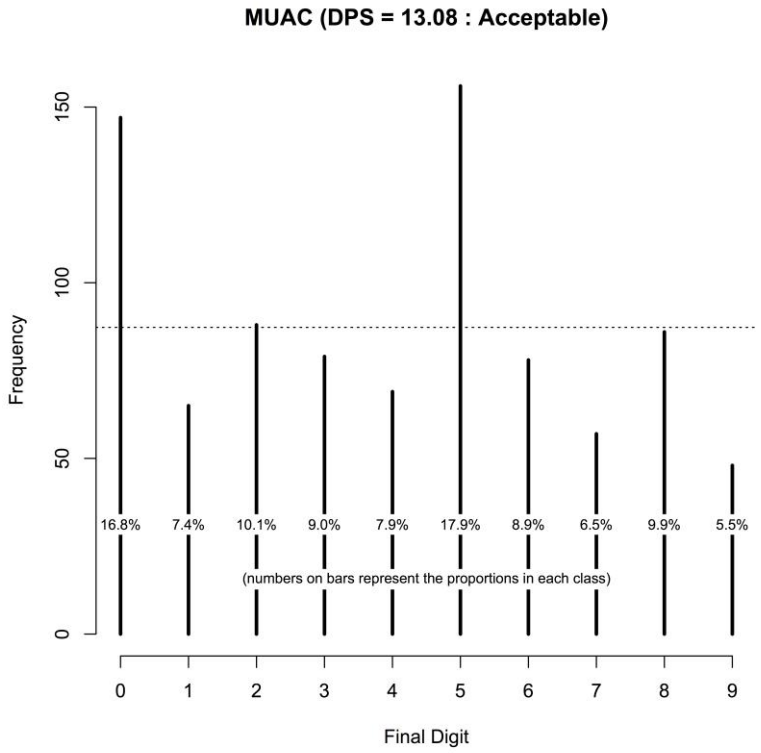
The variable of interest is **`muac`** (MUAC). This variable is measured and recorded in whole millimetres.

We can examine digit preference in the MUAC variable using:

```
digitPreference(svy$muac, digits = 0)
plot(digitPreference(svy$muac, digits = 0), main = "MUAC")
```

The DPS value (13.08) and the DPS plot (*Figure 6.7*) show considerable digit preference and would be classified as "Acceptable" using the classifications shown in *Table 6.2*.

Figure 6.7 DPS for MUAC in a SMART survey.



6.3 Some warnings

The material presented here has assumed that data are recorded with a fixed precision (e.g. one decimal place for weight and height, no decimal places for MUAC). It may be the case that data are recorded with mixed precision. For example, the weights of younger children may be measured using "baby scales" and recorded to the nearest 10 g (i.e. to two decimal places) and the weights of older children measured using "hanging scales" and recorded to the nearest 100 g (i.e. to one decimal place). These sorts of situations can be difficult to handle automatically since (e.g.) 3.1 and 3.10 are the same number and both will be stored in the same way. The easiest approach is to treat the data as two separate datasets when examining digit preference.

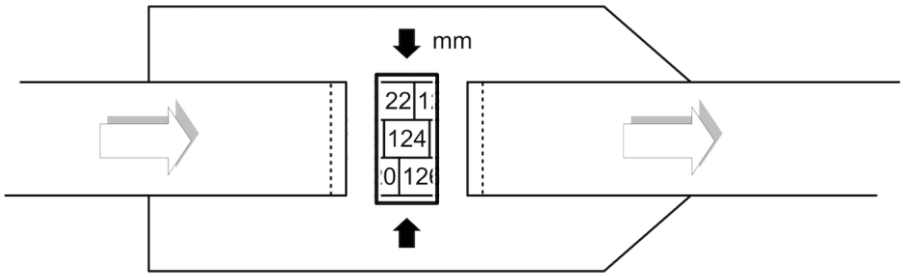
Care should be taken to ensure that you do not mistake the limitations of the measuring instrument for digit preference. For example, some designs of MUAC tape can only return measurements with an even number for the final digit. In this case you should never see MUAC measurements with **1**, **3**, **5**, **7**, or **9** as the final digit. This limitation of the instrument would look like digit preference. The **`digitPreference()`** function can handle this situation.

We will retrieve a dataset:

```
svy <- read.table("dp.ex03.csv", header = TRUE, sep = ",")
head(svy)
```

The file `dp.ex03.csv` is a comma-separated-value (CSV) file containing anthropometric data for a sample of children living in a refugee camp in a West African country.

MUAC was measured using a "numbers in boxes" design MUAC tape:



There can only be even numbers in the final digit when this type of MUAC tape is used.

We should check this:

```
table(svy$muac)
```

This returns:

```
108 114 118 120 122 124 126 128 130 132 134 136 138 140 142 144
  1   1   3   3   2   6   5   5  21   8  16  23  20  16  32  26

146 148 150 152 154 156 158 160 162 164 166 168 170 174 176 178
 24  22  16  25  16  14  19   8   7   7   9   3  11   2   2   1
```

There are only even numbers. Any odd number would be a recording error or a data-entry error.

We can examine digit preference in these data using the **`digitPreference()`** function:

```
digitPreference(svy$muac, digits = 0)
```

This returns:

```
       Digit Preference Score

data:  svy$muac
Digit Preference Score (DPS) = 33.34 (Problematic)
```

This is misleading because the **`digitPreference()`** function assumes that all possible final digits (i.e. **0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**) should be present. This is not the case in the example data.

We can examine this using:

```
digitPreference(svy$muac, digits = 0)$tab
```

which returns:

```
 0  1  2  3  4  5  6  7  8
75  0 74  0 74  0 77  0 74
```

We can use the **`values`** parameter of the **`digitPreference()`** function to specify the values that are allowed in the final digit:

```
digitPreference(svy$muac, digits = 0, values = c(0, 2, 4, 6, 8))
```

This returns:

```
       Digit Preference Score

data:  svy$muac
Digit Preference Score (DPS) = 0.78 (Excellent)
```

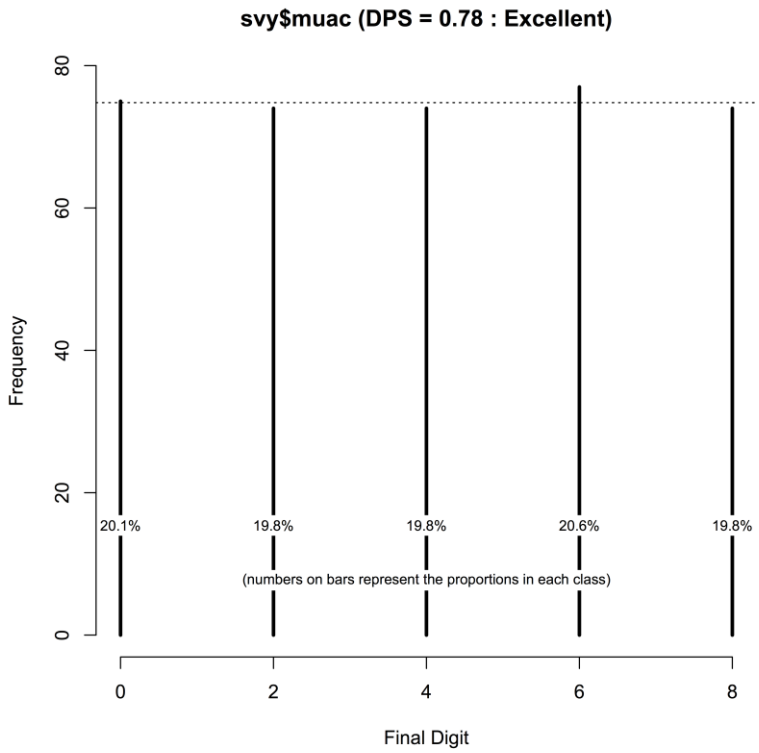
The DPS has changed from 33.34 ("Problematic") to 0.78 ("Excellent").

We can tabulate and plot the frequency of final digits in the **muac** variable:

```
dpsResults <- digitPreference (svy$muac, digits = 0, values = c(0, 2, 4, 6, 8))
dpsResults$tab
dpsResults$pct
plot(dpsResults)
```

See *Figure 6.8*.

Figure 6.8 DPS for MUAC in a survey using a "numbers in boxes" design MUAC tape



7. Age heaping

This section depends on some of the material presented in Section 6 relating to digit preference.

Age heaping is the tendency to report children's ages to the nearest year or adults' ages to the nearest multiple of five or ten years. Age heaping is very common. This is a major reason why data from nutritional anthropometry surveys is often analysed and reported using broad age groups.

7.1 Summarising, tabulating, and visualising age data

We will retrieve a survey dataset:

```
svy <- read.table("dp.ex02.csv", header = TRUE, sep = ",")
```

The file **dp.ex02.csv** is a comma-separated-value (CSV) file containing anthropometric data from a SMART survey in Kabul, Afghanistan.

The first few records in this dataset can be seen using:

```
head(svy)
```

This returns:

```
  psu age sex weight height muac oedema
1   1   6   1    7.3   65.0  146      2
2   1  42   2   12.5   89.5  156      2
3   1  23   1   10.6   78.1  149      2
4   1  18   1   12.8   81.5  160      2
5   1  52   1   12.1   87.3  152      2
6   1  36   2   16.9   93.0  190      2
```

The variable of interest is **age** (age in months):

```
summary(svy$age)
```

Tables can be difficult to use with ungrouped age data because there are usually many different values:

```
table(svy$age)
```

The **fullTable()** function from the NIPN data-quality toolkit is preferred, as this will include values with zero counts:

```
fullTable(svy$age, values = 6:59)
```

We used the **fullTable()** function here because it returns a table containing cells for every value specified by the **values** parameter. The returned table will also only contain cells for the values specified by the **values** parameter. The default for the **values** parameter is the range of the variable being tabulated. This means that the **values** parameter can be sometimes be omitted:

```
fullTable(svy$age)
```

Omitting the **values** parameter only works reliably for numeric variables containing whole numbers. If the variable being tabulated is a character variable or is a numeric variable containing one or more numbers with decimal places then you should specify the **values** parameter.

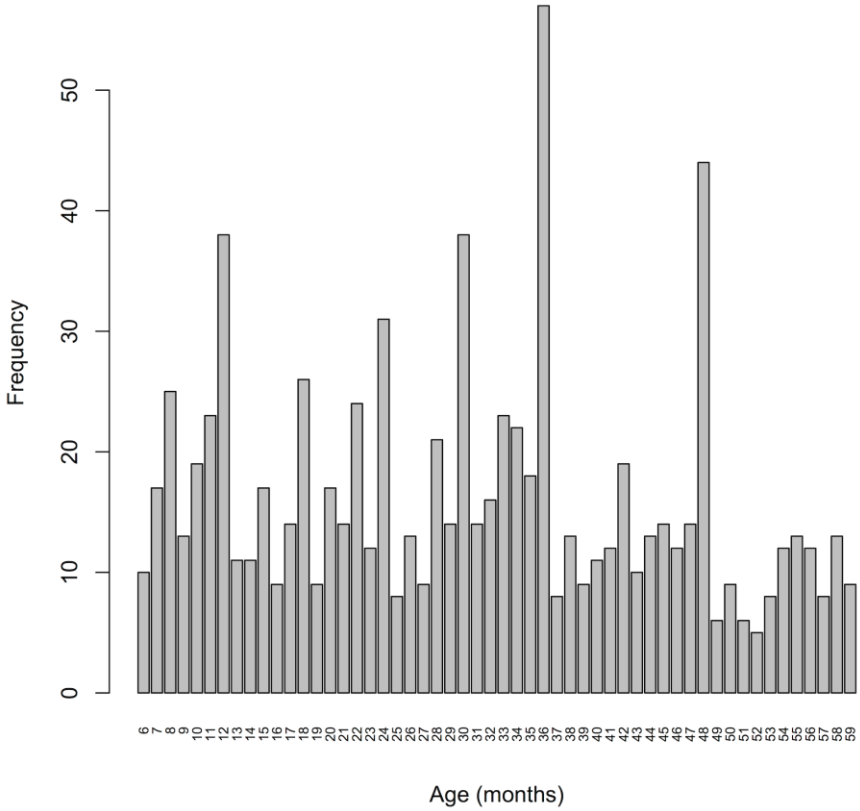
A graphical analysis is usually more informative than a tabular analysis:

```
barplot(fullTable(svy$age, values = 6:59),
      xlab = "Age (months)", ylab = "Frequency", las = 3, cex.names = 0.6)
```

We expect all ages to be present with roughly equal frequency or with frequency reducing slowly with age due to mortality. We can see that there is marked age-heaping at 12, 18, 24, 30, 36, and 48 months (see *Figure 7.1*). This is very common when age is reported by mothers. This is because of a tendency for mothers and other carers to round ages to whole years or half years.

Note that we used **`values = 6:59`** with the **`fullTable()`** function from the NIPN data quality toolkit. We did this because it is the range of values that should be present in the age variable.

Figure 7.1 Distribution of ages in a SMART survey with age-heaping at whole and half years.



7.2 Age heaping in children

Age heaping can seriously affect survey results for indices that include an age component (e.g. height-for-age and weight-for age). The effect is important when there is *systematic* rounding up or systematic rounding down. Systematic rounding can lead to bias. If rounding is systematically down then indices will be biased upwards and prevalence biased downwards. If rounding is systematically up then indices will be biased downwards and prevalence biased upwards.

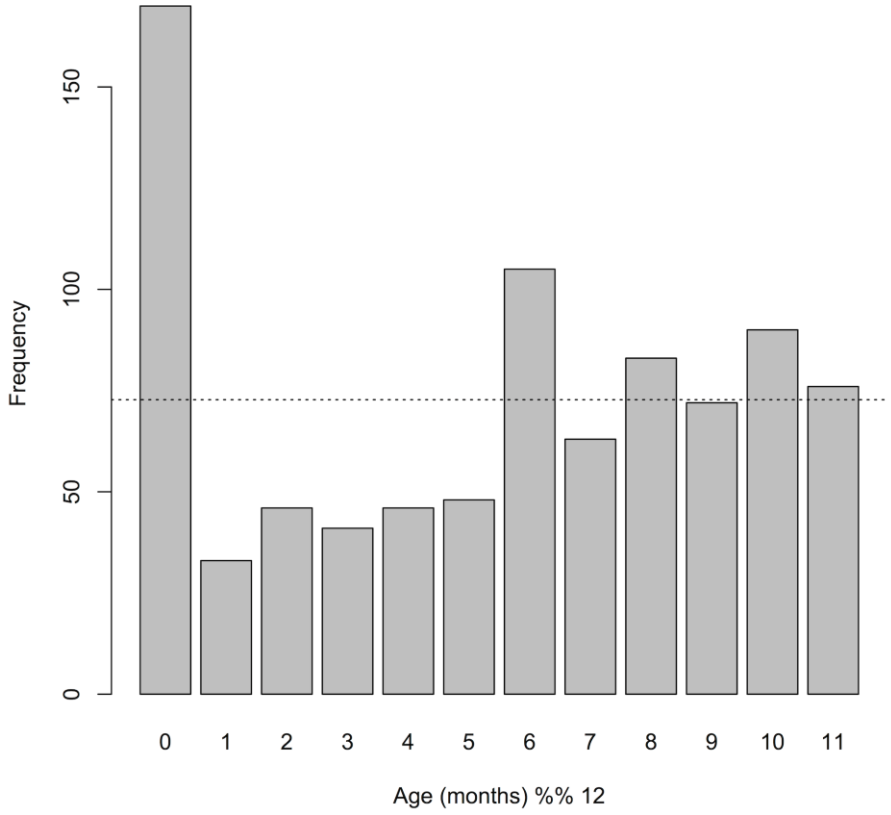
A useful way of looking at age heaping when age is recorded in months is to examine the remainders when the ages are divided by 12.

The *R* language provides a special operator (**%%**) to help with this:

```
rem <- svy$age %% 12
remTable <- fullTable(rem, values = 0:11)
remTable
prop.table(remTable) * 100
barplot(remTable, xlab = "Age (months) %% 12", ylab = "Frequency")
abline(h = sum(remTable / 12), lty = 3)
chisq.test(remTable)
```

See *Figure 7.2*.

Figure 7.2 Remainder of age divided by 12 in a SMART survey with age-heaping at whole and half years. The dotted line shows expected numbers.



The NIPN data quality toolkit provides an *R* language function called **ageHeaping()** that performs this age-heaping analysis. Applying this function to the example data:

```
ageHeaping(svy$age)
```

This returns:

```
        Age-heaping Analysis

data:  Remainder of svy$age / 12
X-squared = 214.9588, df = 11, p-value = 0.0000
```

The output of the **`ageHeaping()`** function can be saved for later use:

```
ah12 <- ageHeaping(svy$age)
```

The saved output contains the Chi-squared test and frequency tables of the final digits (counts and percentages). These can be accessed using:

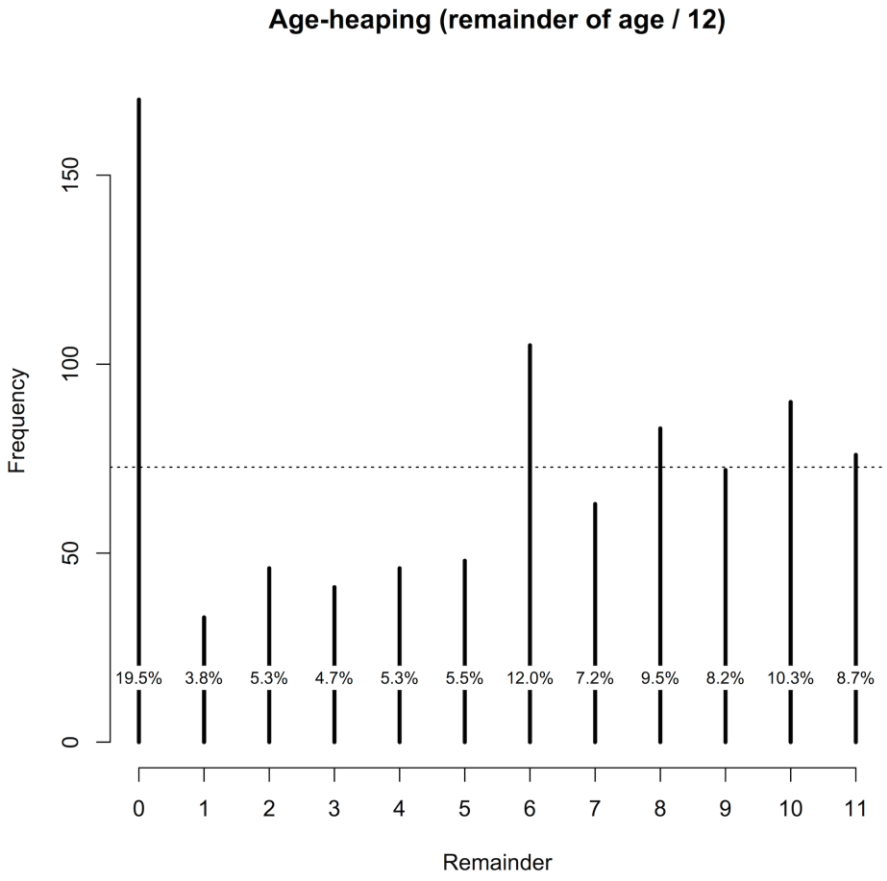
```
ah12
ah12$X2
ah12$df
ah12$p
ah12$tab
ah12$pct
```

The saved results may also be plotted:

```
plot(ah12, main = "Age-heaping (remainder of age / 12)")
```

The resulting plot is shown in *Figure 7.3*.

Figure 7.3 Age heaping at whole and half years in a SMART survey. The dotted line shows expected numbers.

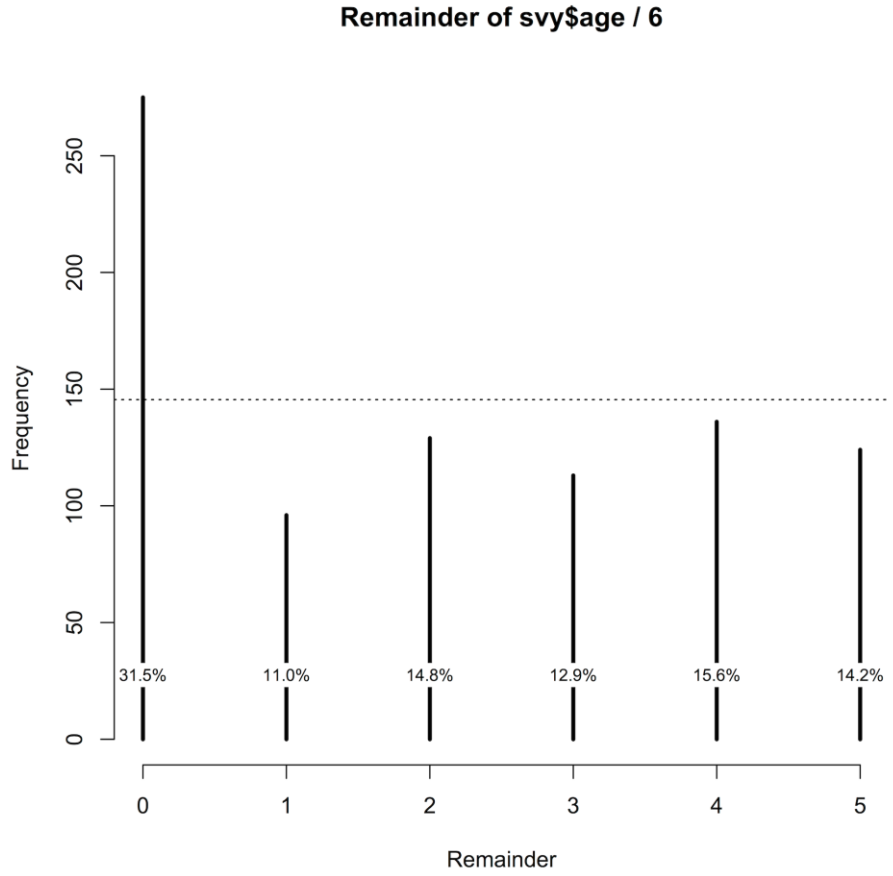


The **ageHeaping()** function assumes that you want to examine the remainder after dividing by twelve. This is useful when working with ages that are recorded in months. It may also be useful to use other divisors, such as examining the remainder after dividing by six:

```
ah6 <- ageHeaping(svy$age, divisor = 6)
print(ah6)
plot(ah6)
```

This shows the extent of age heaping at whole and half-years (see *Figure 7.4*).

Figure 7.4 Age heaping at whole and half years in a SMART survey. The dotted line shows expected numbers.



7.3 Age heaping in adults

Using ten and five as divisors can be useful when dealing with data for adults in which ages are recorded in whole years.

We will retrieve a survey dataset:

```
svy <- read.table("ah.ex01.csv", header = TRUE, sep = ",")
head(svy)
```

The file **ah.ex01.csv** is a comma-separated-value (CSV) file containing anthropometric data from a Rapid Assessment Method for Older People (RAM-OP) survey in the Dadaab refugee camp in Garissa, Kenya. This is a survey of people aged sixty years and older.

The variable of interest is **age** (age in years):

```
summary(svy$age)
```

Care should be exercised when specifying the **divisor** to use in the analysis of age heaping. Not all calendars use base ten. Amongst Han Chinese, for example, age heaping may occur with a twelve-year cycle corresponding to preferred animal years in the Chinese calendar. An analysis of age heaping that concentrates on specific digits (e.g. zero and five) or on decimal intervals will not be appropriate in all populations. It is advisable, therefore to use simple tabulation and visualisation techniques to heap data first and then decide on an appropriate **divisor**.

With the example data:

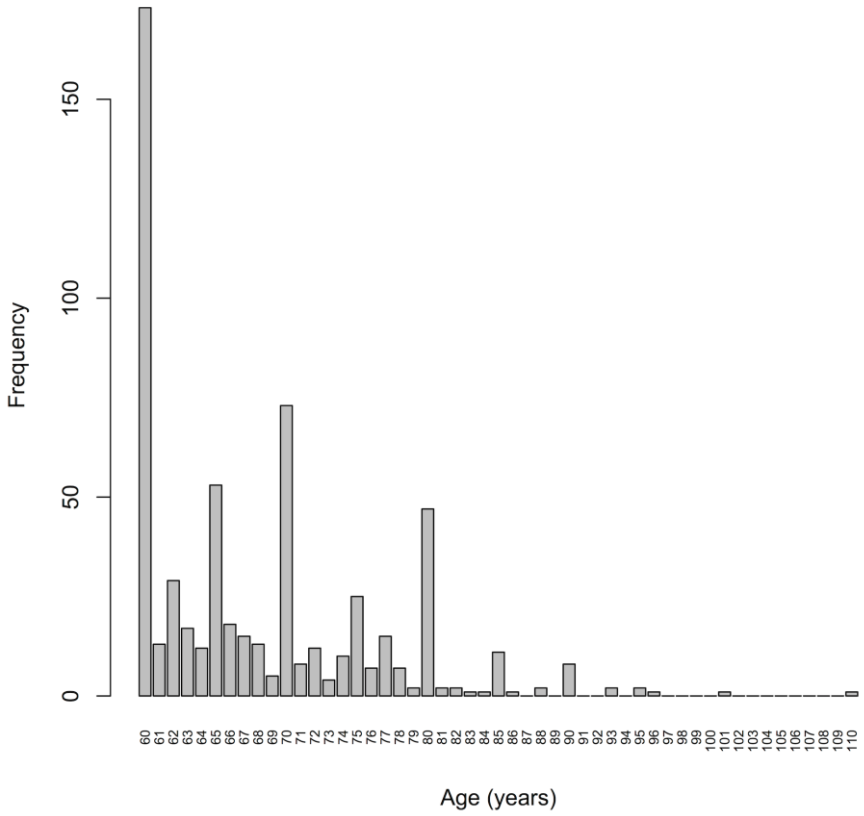
```
summary(svy$age)

fullTable(svy$age)

barplot(fullTable(svy$age),
      xlab = "Age (years)", ylab = "Frequency", las = 3, cex.names = 0.6)
```

shows age-heaping at decades and half-decades (see *Figure 7.5*).

Figure 7.5 Age heaping at decades and half-decades in a RAM-OP survey of adults aged 60 years and older.

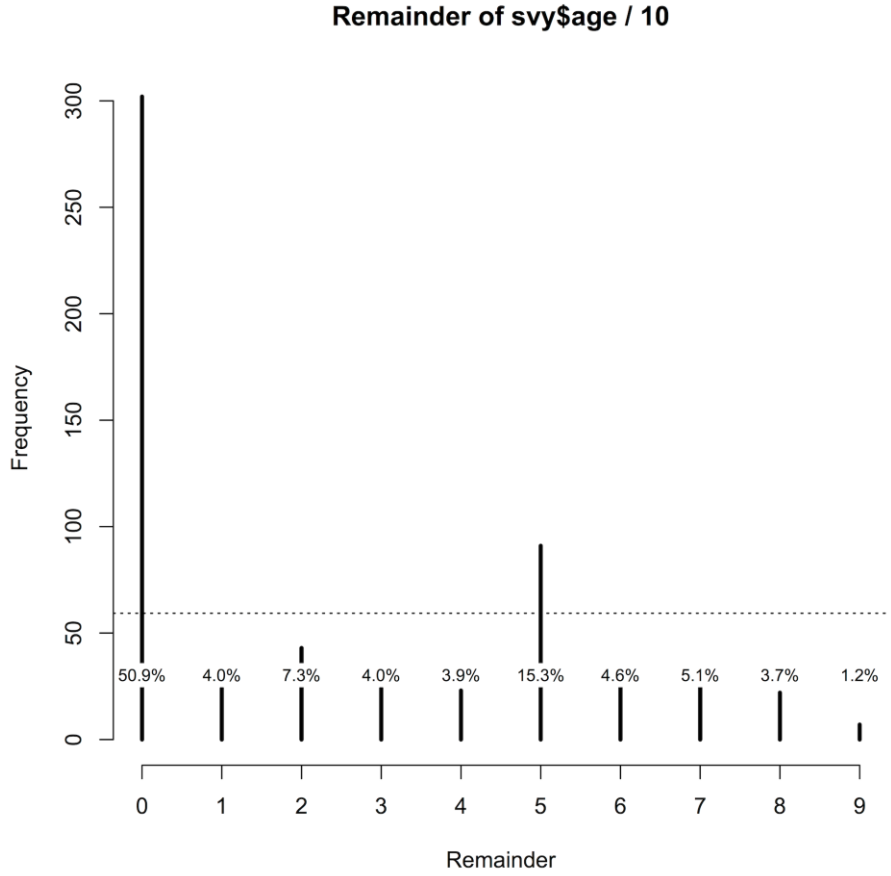


In this survey using a **`divisor`** of 10 would be appropriate:

```
ah10 <- ageHeaping(svy$age, divisor = 10)
print(ah10)
plot(ah10)
```

There is pronounced age heaping at decades and, to a lesser extent, half-decades in these data (see *Figure 7.6*).

Figure 7.6 Age-heaping at decades and half-decades in a RAM-OP survey. The dotted line shows expected numbers.

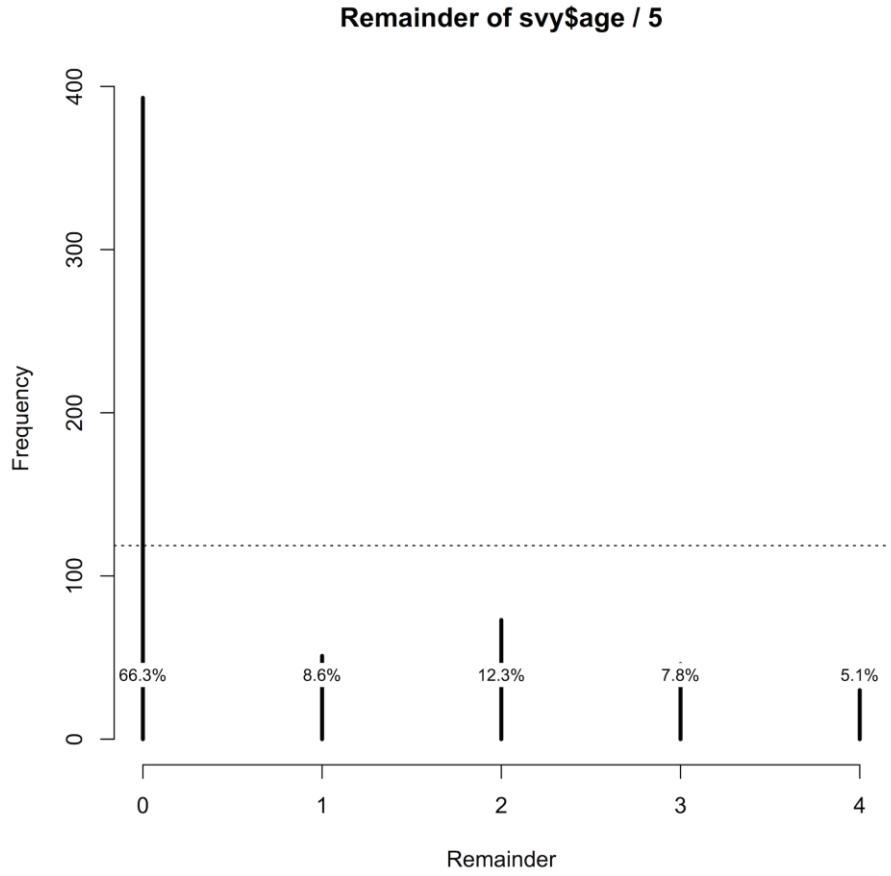


It may also be useful to use other divisors, such as examining the remainder after dividing by five:

```
ah5 <- ageHeaping(svy$age, divisor = 5)
print(ah5)
plot(ah5)
```

This shows the extent of age heaping at whole and half decades (see *Figure 7.7*).

Figure 7.7 Age heaping at whole and half decades in a RAM-OP survey. The dotted line shows expected numbers.



8. Using scatterplots to identify outliers

We can expect anthropometric variables in children to be strongly and positively associated with each other. This is because children tend to gain both weight and height as they grow. This allows us to use graphical and numerical methods to identify outliers (i.e. observations that are distant from most other observations) that may be due to errors.

It is important to note that anthropometric surveys often use a method of comparing observed values with reference values using a process known as "flagging" to identify and censor outliers. The methods outlined in this section are intended to complement rather than replace the "flagging" methods discussed elsewhere.

8.1. Identifying outliers by observation

We will retrieve a survey dataset:

```
svy <- read.table("sp.ex01.csv", header = TRUE, sep = ",")
head(svy)
```

The file **`sp.ex01.csv`** is a comma-separated-value (CSV) file containing anthropometric data from a SMART survey from the Democratic Republic of Congo.

We will look at the relationship between height and weight in this dataset:

```
plot(svy$height, svy$weight)
```

The resulting plot is shown in *Figure 8.1*. There is a clear positive linear relationship between height and weight (i.e. weight increases with increasing height along a straight line). We can assess the strength of this relationship using the Pearson correlation coefficient:

```
cor(svy$height, svy$weight, method = "pearson", use = "complete.obs")
```

which returns:

```
0.9204116
```

This is very close to one, which would indicate a perfect positive association. There are, however, a few points that lie outside of the bulk of the plotted points. These *outliers* may be due to errors in the data.

The presence of oedema can be associated with increased body weight. This is a particular issue with severe oedema. An outlier with a high value of weight for a given height could be due to oedema. We can check this:

```
plot(svy$height, svy$weight, pch = ifelse(svy$oedema == 1, 19, 1))
```

The **`pch = ifelse(svy$oedema == 1, 19, 1)`** tells the **`plot()`** function to plot filled circles for oedema cases and open circles for children without oedema. The resulting plot is shown in *Figure 8.2*. A single high weight for height outlier appears to be due to the presence of oedema.

The other filled circles that are located in the main mass of plotted points show that children with oedema may have a body weight within the normal range for their height. These children may not be classified as wasted, but they are suffering from a form of severe acute malnutrition (SAM) known as *kwashiorkor*.

Figure 8.1. The relationship between height and weight in the example dataset.

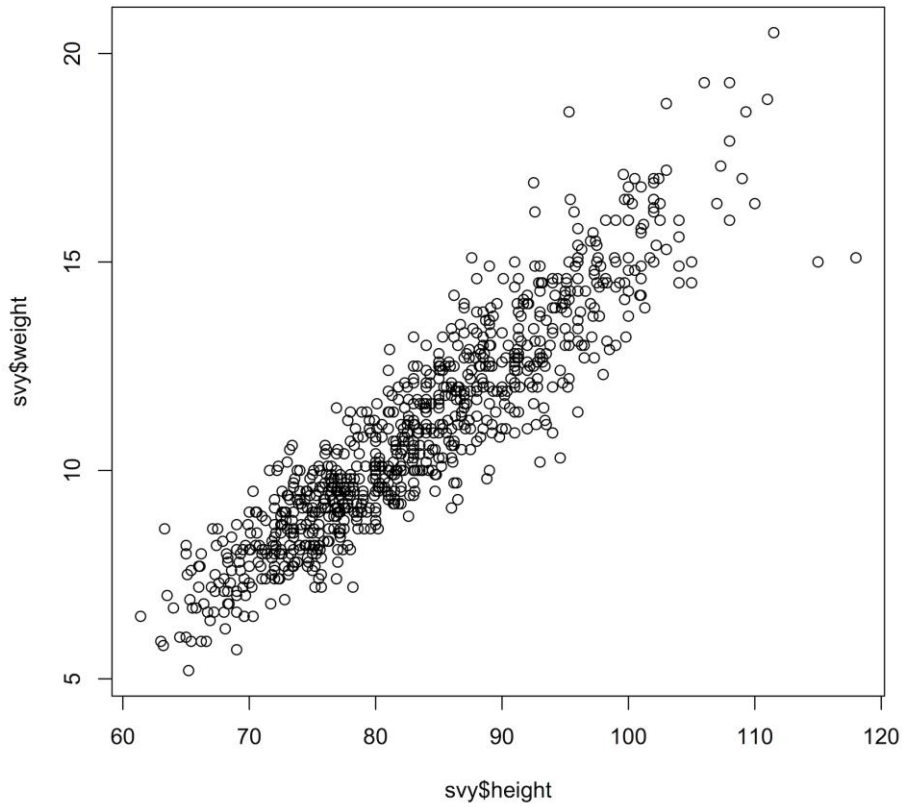
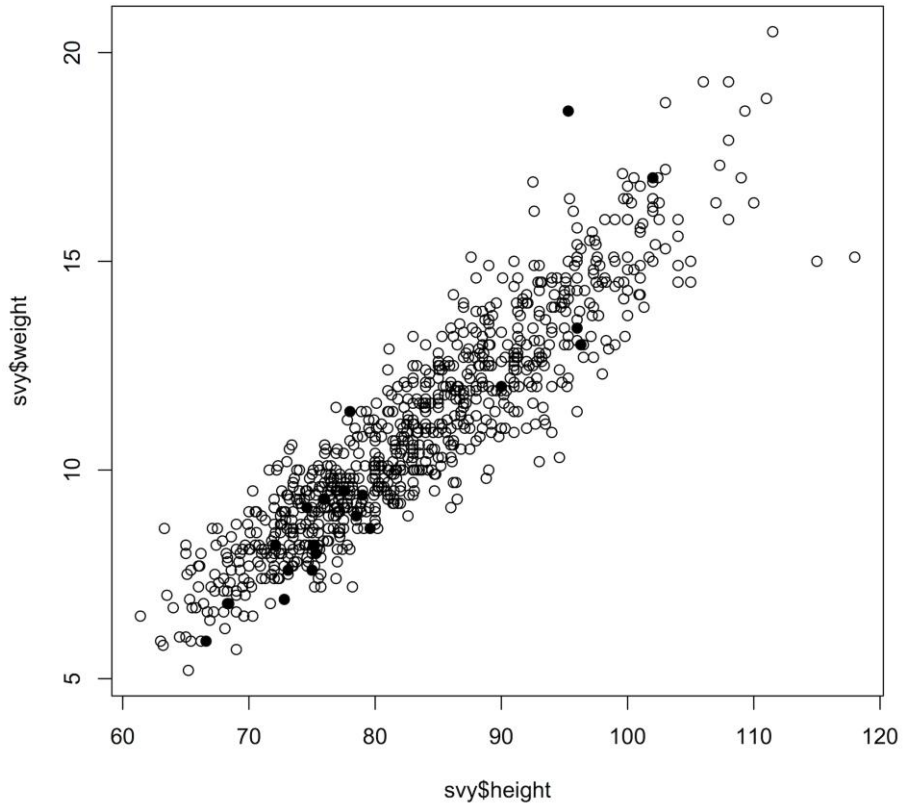


Figure 8.2. The relationship between height and weight in children with and without oedema in the example dataset. The filled circles show children with oedema.



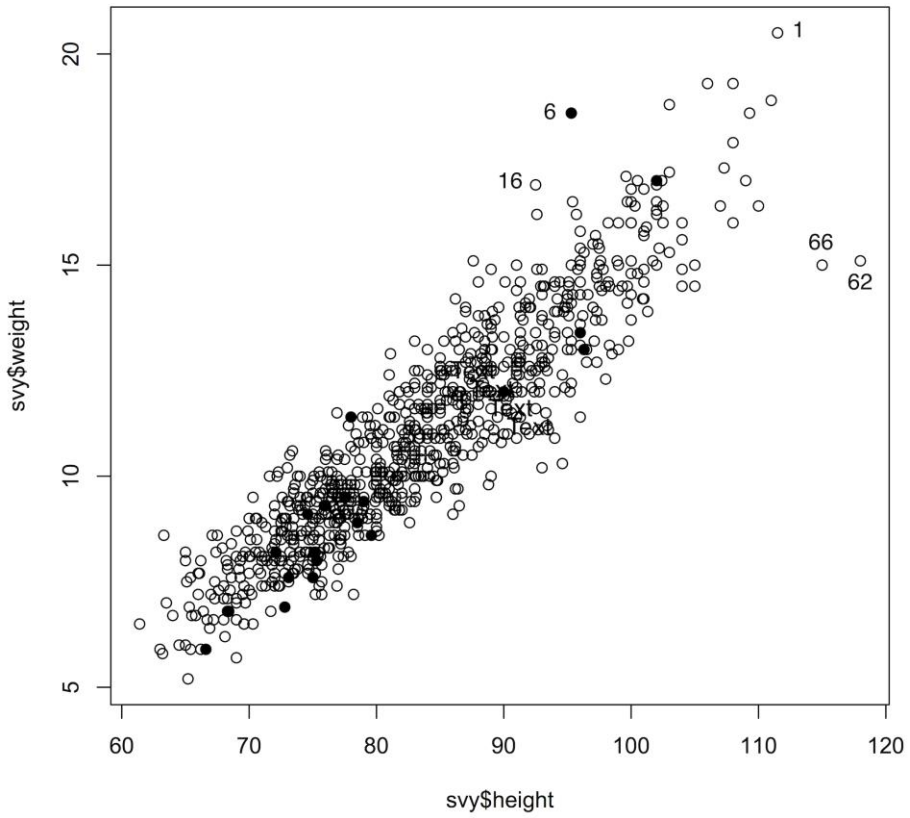
Outliers can be identified by eye. The **identify()** function can help with this:

```
plot(svy$height, svy$weight, pch = ifelse(svy$oedema == 1, 19, 1))
identify(svy$height, svy$weight)
```

Clicking on any point will cause the record (row) number associated with each point to be displayed on the plot (see *Figure 8.3*). Right-clicking on the plot or pressing the "escape" key will stop **identify()**.

The behaviour of the **identify()** function may be different when you use an alternative graphical user interface for *R* such as *RStudio* or *R-AnalyticFlow*.

Figure 8.3. Outliers identified by eye using the **plot()** and **identify()** functions. The numbered points were identified by eye as outliers (the number corresponds to the record (row) number in the example dataset). Filled circles show children with oedema.



The **identify()** function will, by default, display record (row) numbers for the points identified. This is usually what is needed. Alternative labels can be displayed. For example:

```
plot(svy$height, svy$weight, pch = ifelse(svy$oedema == 1, 19, 1))
identify(svy$height, svy$weight,
       labels = paste(svy$height, svy$weight, sep = ";"), cex = 0.75)
```

displays the height and weight values at selected points.

The ability to display custom labels is useful if there is a variable (column) in a dataset that contains unique record identifiers.

It is useful to be able to store the record (row) numbers of identified points:

```
plot(svy$height, svy$weight, pch = ifelse(svy$oedema == 1, 19, 1))
stored <- identify(svy$height, svy$weight)
```

If the same points shown in *Figure 8.3* are clicked to identify them then:

```
stored
```

will return:

```
1 6 16 62 66
```

We can examine the data for the identified points:

```
svy[stored, ]
```

This returns:

```
   age sex weight height muac oedema
1   54   1   20.5  111.5  180      2
6   48   2   18.6   95.3  171      1
16  30   1   16.9   92.5  188      2
62  55   1   15.1  118.0  156      2
66  56   1   15.0  115.0  148      2
```

The **oedema** data is coded **1** for present and **2** for absent.

Data can be checked and edited if needed. Note that record **6** is an oedema case and should probably not be changed.

If your dataset has many variables (columns) then you may specify only the variables (columns) of interest:

```
svy[stored, c("weight", "height", "oedema")]
```

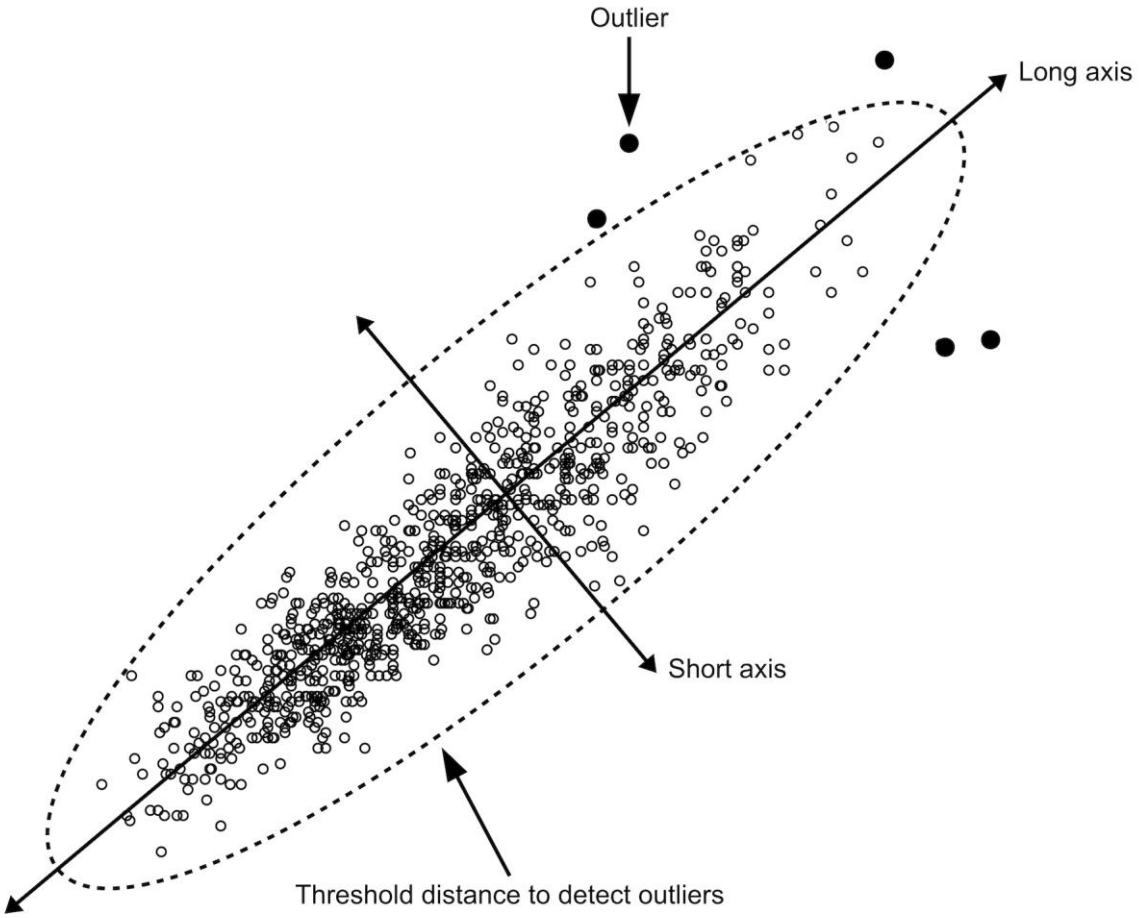
This returns:

```
   weight height oedema
 1   20.5  111.5      2
 6   18.6   95.3      1
16   16.9   92.5      2
62   15.1  118.0      2
66   15.0  115.0      2
```

8.2 Identifying outliers using statistical distance

A more formal method of identifying outliers is to use a measure of the *statistical distance*. A common measure that is applied to scatterplot data is the *Mahalanobis distance*. This treats the bivariate probability distribution as an ellipsoid. The Mahalanobis distance is the distance of a point from the centre of mass of the distribution divided by width of the ellipsoid in the direction of the point. This is shown in *Figure 8.4.*

Figure 8.4. A scatterplot of data illustrating the two axes around the centre mass of a distribution used to calculate the Mahalanobis distance.



In directions in which the ellipsoid has a short axis the test point must be close to the centre of mass of the distribution. In directions in which the ellipsoid has a long axis the test point may be more distant from the centre of mass of the distribution.

The NIPN data quality toolkit provides an *R* language function **`outliersMD()`** that uses the Mahalanobis distance to identify outliers in the same dataset:

```
svy[outliersMD(svy$height,svy$weight), ]
```

This returns the same set of records that was identified by eye:

```
   age sex weight height muac oedema
1   54   1   20.5  111.5  180      2
6   48   2   18.6   95.3  171      1
16  30   1   16.9   92.5  188      2
62  55   1   15.1  118.0  156      2
66  56   1   15.0  115.0  148      2
```

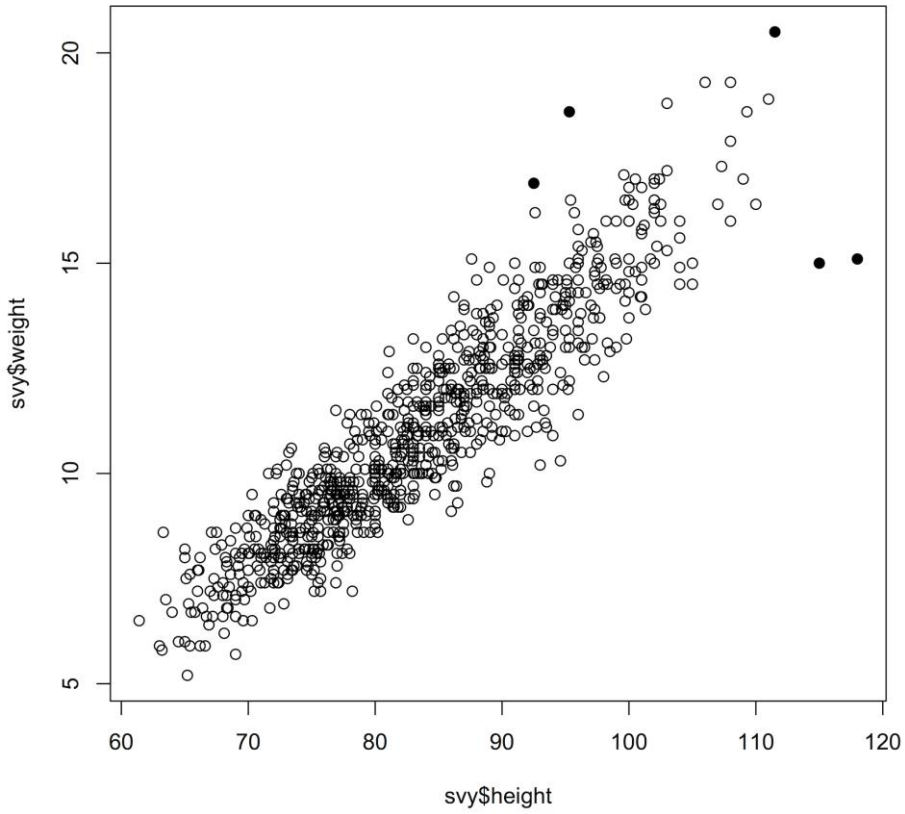
Data can be checked and edited if needed. Note that record **6** is an oedema case and should probably not be changed.

We can use the **outliersMD()** to identify and display outliers on a scatterplot:

```
plot(svy$height, svy$weight,
       pch = ifelse(outliersMD(svy$height, svy$weight), 19, 1))
```

See *Figure 8.5*.

Figure 8.5. Outliers identified automatically using the Mahalanobis distance method implemented by the **outliersMD()** function in the NIPN data quality toolkit. Filled circles show the outliers identified by the Mahalanobis distance.



The **outliersMD()** function has an **alpha** parameter. The default value for the **alpha** parameter is **alpha = 0.001**. This value is used automatically unless another value is specified.

When we use **alpha = 0.001** we are looking for records with values so extreme that we would expect to find them with a probability of 0.001 when there are no problems with the data.

We can calculate the number of outliers that we expect to see by chance with **alpha = 0.001** using:

```
round(nrow(svy) * 0.001)
```

This returns:

```
1
```

We found five potential outliers. The difference between the number that we expected and the number that we observed (i.e. one expected vs. five observed) suggests that some of the identified outliers are true outliers or due to data errors.

Another way of looking at the **alpha** parameter is that it alters the sensitivity of the **outlierMD()** function by altering the threshold distance that is used to define outliers. This can be useful when using the **outlierMD()** function with some, but not all, curvilinear relationships (see below).

Larger values of **alpha** will tend to detect more potential outliers. For example:

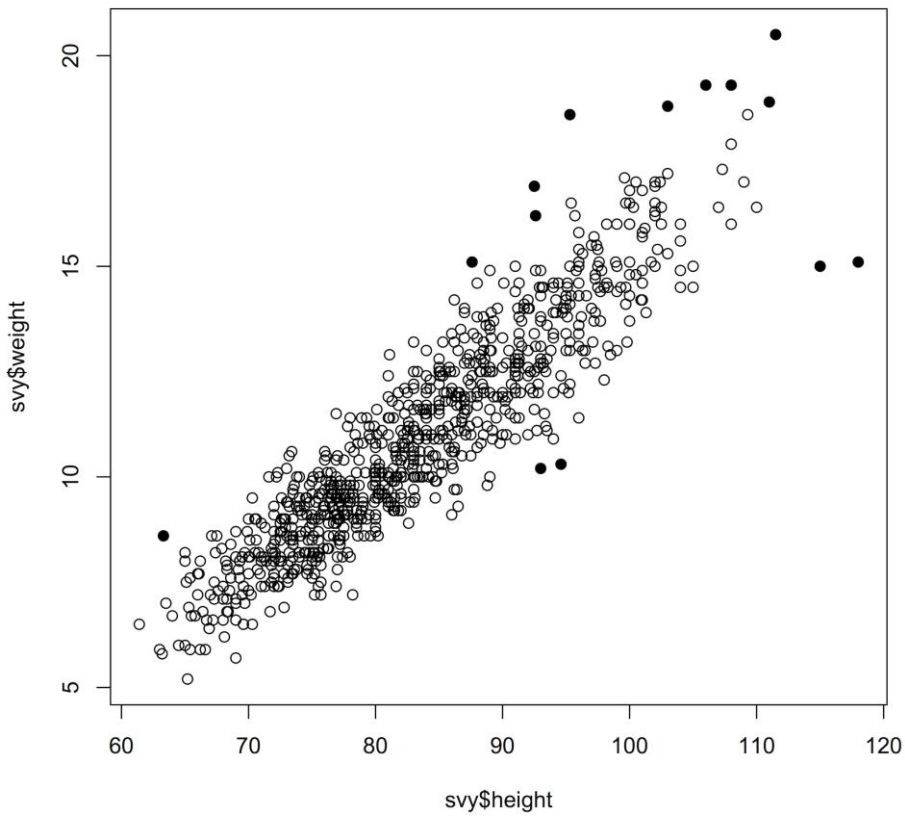
```
plot(svy$height, svy$weight,
      pch = ifelse(outliersMD(svy$height, svy$weight, alpha = 0.01), 19, 1))
```

and:

```
svy[outliersMD(svy$height,svy$weight, alpha = 0.01), ]
```

See *Figure 8.6*.

Figure 8.6. Outliers identified automatically by the **outliersMD()** function with **alpha = 0.01**. Filled circles show the outliers identified by the Mahalanobis distance.



In almost all cases the default **alpha = 0.001** will be appropriate.

The techniques outlined above can be used to examine the relationships between other pairs of anthropometric variables (e.g. **weight** and **muac**) and to identify outliers. All sensible pairings of variables should be examined.

8.3 Anthropometric measurements and age

We also expect anthropometric variables to be associated with age. This relationship is particularly strong in children. It will be less strong in adults and may be weak or even reversed in older people.

We can explore the relationship between an anthropometric variable and age using the techniques described above. For example:

```
plot(svy$age, svy$height, pch = ifelse(outliersMD(svy$age, svy$height), 19, 1))
svy[outliersMD(svy$age, svy$height), ]
```

See *Figure 8.7*.

There are some problems with this approach. Age is often reported and recorded so that the data show considerable age heaping (see Section 7). Age is unlikely to be approximately normally distributed, which is an assumption of the Mahalanobis distance method. The relationship between anthropometric variables and age usually follows a "growth curve" rather than a straight line.

The combination of age heaping, non-normality, and a curvilinear relationship may reduce the effectiveness of the Mahalanobis distance method for detecting outliers. It may be useful, in such cases, to increase the value of the **alpha** parameter. For example:

```
plot(svy$age, svy$height,
       pch = ifelse(outliersMD(svy$age, svy$height, alpha = 0.025), 19, 1))
```

See *Figure 8.8*.

Outliers can be listed using the same value for **alpha**:

```
svy[outliersMD(svy$age, svy$height, alpha = 0.025), ]
```

The Mahalanobis distance method is usually robust enough to deal with age data provided an appropriate value of **alpha** is used.

Figure 8.7. The relationship between age and height in the example dataset showing outliers identified by the **`outliersMD()`** function. Filled circles show the outliers identified by the Mahalanobis distance.

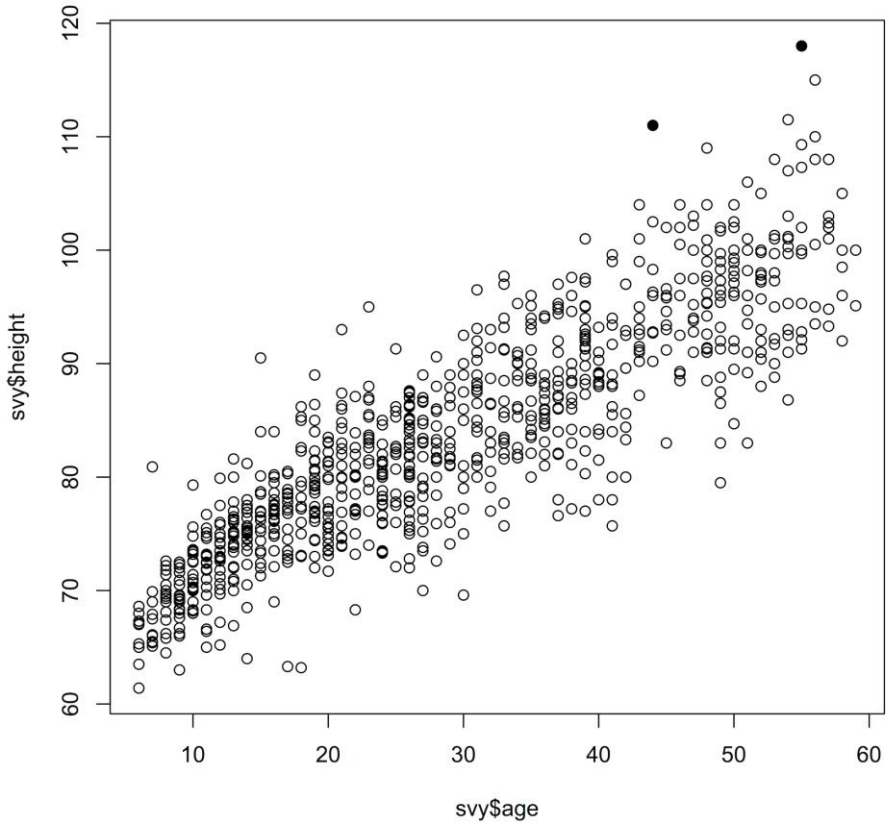
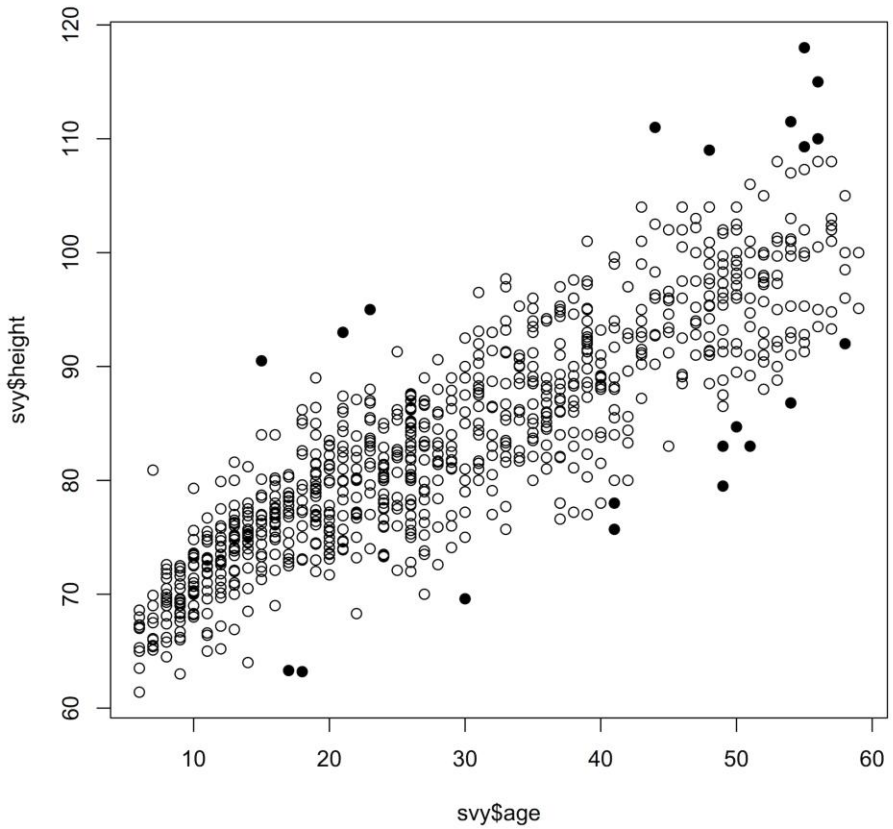


Figure 8.8. The relationship between age and height in the example dataset showing outliers identified by the **`outliersMD()`** function with **`alpha = 0.025.`** Filled circles show the outliers identified by the Mahalanobis distance.



8.4 Difficult relationships for the Mahalanobis distance method

The Mahalanobis distance method works well with pairs of variables as long as the relationship between the two variables is *monotonic* (i.e. one variables always increases or always decreases in value as the other variable increases in value). This is usually the case with anthropometric data.

We will explore the use of the Mahalanobis distance method with data that is not monotonic using generated data:

```
x <- c(4,  8, 16, 17, 22, 27, 38, 40, 47, 48, 53, 55, 63, 71, 76, 85, 92, 96)
y <- c(6, 22, 34, 42, 51, 59, 64, 69, 70, 20, 70, 63, 63, 55, 46, 33, 19,  6)
plot(x, y)
```

There is a clear relationship between **x** and **y** but it is **not** a monotonic relationship (i.e. it is not always increasing or decreasing). There is a single obvious outlier. See *Figure 8.8*.

The Mahalanobis distance method will **not** work well with this data.

This:

```
plot(x, y, pch = ifelse(outliersMD(x, y), 19, 1))
```

fails to detect the outlier. Relaxing the **alpha** parameter:

```
plot(x, y, pch = ifelse(outliersMD(x, y, alpha = 0.025), 19, 1))
```

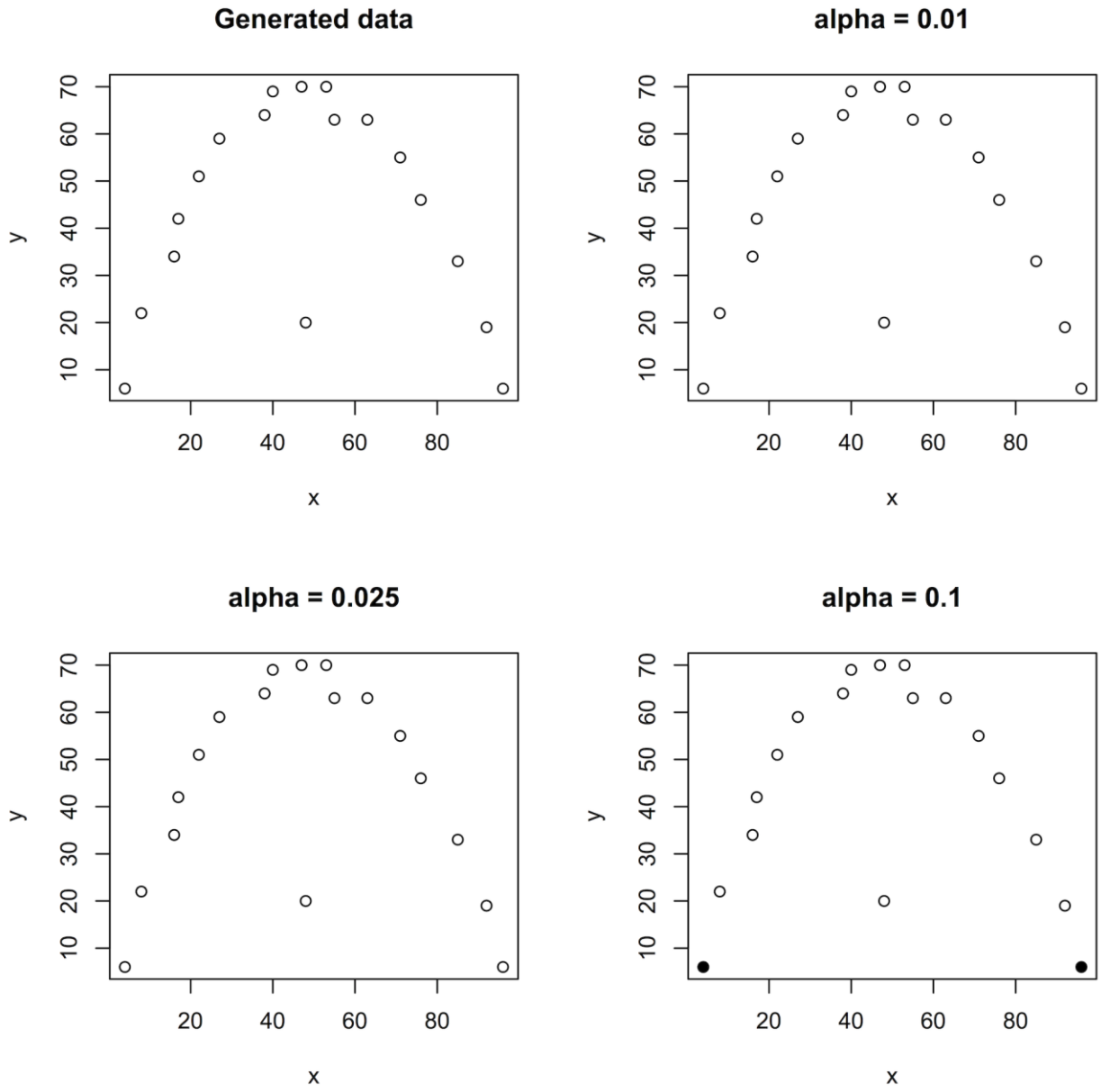
does not help. Relaxing the **alpha** parameter further:

```
plot(x, y, pch = ifelse(outliersMD(x, y, alpha = 0.1), 19, 1))
```

results in false positive results but fails to identify the clear outlier.

All of these plots are shown in *Figure 8.9*.

Figure 8.9. The Mahalanobis distance method fails with non-monotonic relationships. Filled circles show the outliers identified by the Mahalanobis distance at the specified **`alpha`**. Note that the Mahalanobis distance method fails to identify the clear outlier and falsely identifies two points as outliers.



Although the Mahalanobis distance cannot be used directly to identify outliers in non-monotonic relationships, it can be applied to residuals from fitted non-linear models. This technique is unlikely to be required with anthropometric data and is not covered in this toolkit.

It is very unlikely that you will see non-monotonic relationships with anthropometric data. You are likely to see "growth curves" that look like this:

```
set.seed(0)
x <- 0:100
y <- 1 - exp(-x / 50) + rnorm(101, 0, 0.05)
plot(x, y)
lines(x, 1 - exp(-x / 50), lty = 2)
```

See *Figure 8.10*. This is a monotonic relationship. The Mahalanobis distance method should work well with this data. If we add a clear outlier:

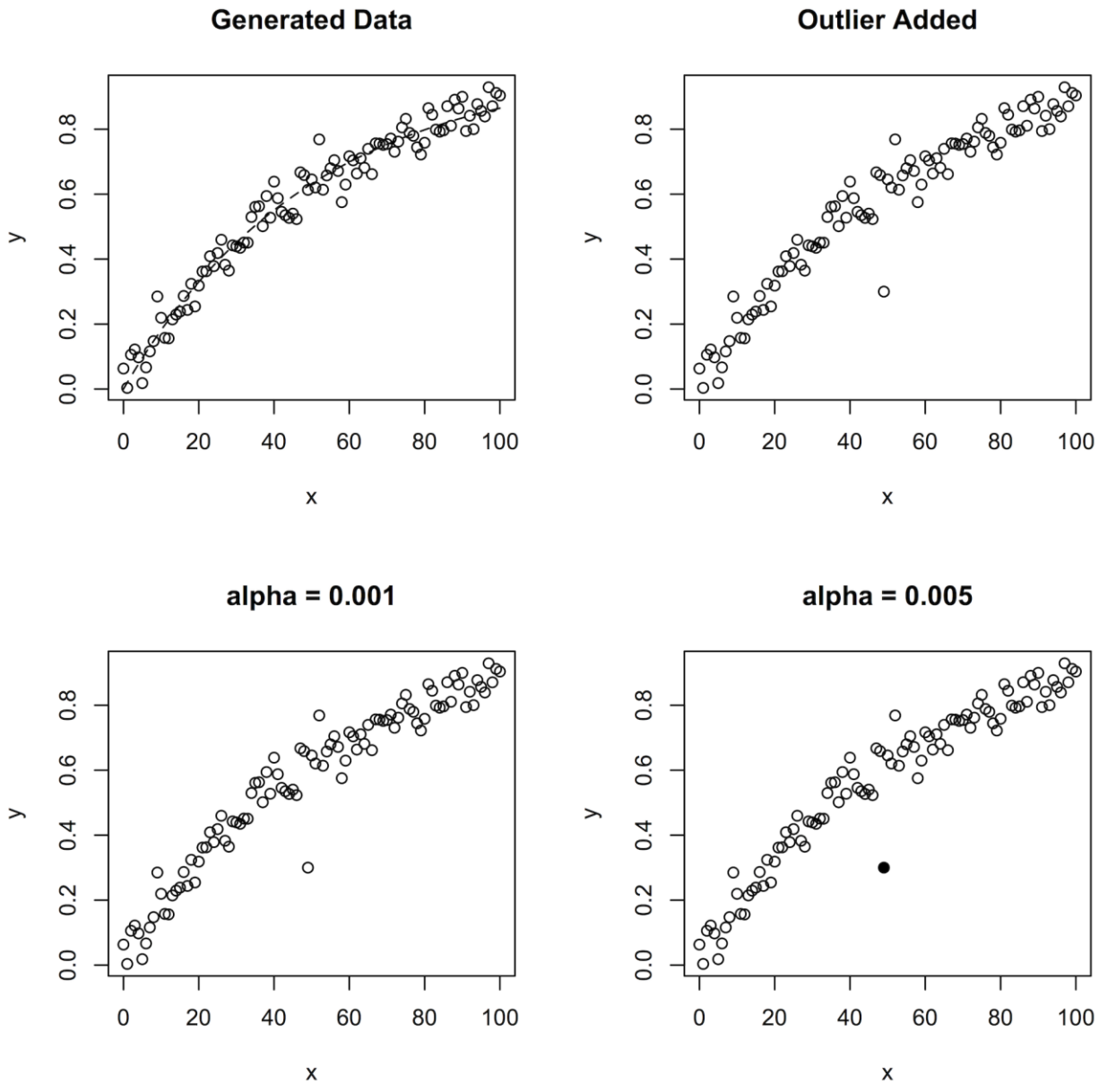
```
y[50] <- 0.3
plot(x, y)
```

this can be detected using the Mahalanobis distance method using a slightly relaxed **`alpha`** value:

```
plot(x, y, pch = ifelse(outliersMD(x, y, alpha = 0.005), 19, 1))
```

All of these plots are shown in *Figure 8.10*.

Figure 8.9. The Mahalanobis distance method works with monotonic relationships. The filled circle shows the outlier identified by the Mahalanobis distance using **`alpha = 0.005`**. Note that the Mahalanobis distance method correctly identifies the clear outlier.



8.5 Working with data from children older than 5 years

We will now look at using scatterplots and the Mahalanobis distance methods with data from older children.

We will retrieve a survey dataset:

```
svy <- read.table("sp.ex02.csv", header = TRUE, sep = ",")
head(svy)
```

The file **sp.ex02.csv** is a comma-separated-value (CSV) file containing anthropometric data from a survey of school-age (i.e. between 5 and 15 years) children in Pakistan.

We can summarise the dataset using:

```
summary(svy)
```

This returns:

```
     region          school        ageMonths          sex            weight     
 Min.   :1.000   Min.   : 1.00   Min.   : 60.0   Min.   :1.000   Min.   :10.30  
 1st Qu.:3.000   1st Qu.: 8.00   1st Qu.: 83.0   1st Qu.:1.000   1st Qu.:17.20  
 Median :4.000   Median :15.00   Median : 98.0   Median :1.000   Median :21.30  
 Mean   :4.491   Mean   :15.51   Mean   :104.8   Mean   :1.397   Mean   :22.62  
 3rd Qu.:7.000   3rd Qu.:23.00   3rd Qu.:124.0   3rd Qu.:2.000   3rd Qu.:27.00  
 Max.   :8.000   Max.   :30.00   Max.   :178.0   Max.   :2.000   Max.   :51.90  

     height           haz              waz              baz         
 Min.   : 86.2   Min.   :-5.730   Min.   :-5.350   Min.   :-4.7000  
 1st Qu.:108.7   1st Qu.:-2.640   1st Qu.:-2.380   1st Qu.:-1.2900  
 Median :120.9   Median :-1.790   Median :-1.615   Median :-0.7600  
 Mean   :121.2   Mean   :-1.705   Mean   :-1.581   Mean   :-0.7758  
 3rd Qu.:132.6   3rd Qu.:-0.790   3rd Qu.:-0.805   3rd Qu.:-0.2100  
 Max.   :164.2   Max.   : 3.550   Max.   : 3.010   Max.   : 1.9900  
                                  NA's   :267      NA's   :8        
```

The **baz** variable contains the BMI-for-age z-score calculated from the **ageMonths**, **sex**, **weight**, and **height** variables using the WHO growth reference. A key thing to notice in the summary is the large number of missing values in the **waz** variable. This is because the weight-for-age z-score is not calculated for children aged older than 120 months. You can check this using:

```
by(svy$ageMonths, is.na(svy$waz), summary)
```

This gives:

```
is.na(svy$waz): FALSE
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  60.00   76.00   88.00   88.24   99.00  120.00 
------------------------------------------------ 
is.na(svy$waz): TRUE
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  121.0   125.5   141.0   140.8   151.0   178.0 
```

There appears to be nothing odd about the large number of missing values in the **waz** variable.

We should investigate the missing values in the **baz** variable:

```
svy[is.na(svy$baz), ]
```

This returns:

```
    region school ageMonths sex weight height   haz   waz baz
83       1      3       143   2   14.0  125.9 -3.64    NA  NA
158      2      6        96   1   12.3  118.4 -1.57 -5.26  NA
275      3     10        77   1   10.3  113.9 -0.88 -5.35  NA
415      4     15        75   1   33.0  108.3 -1.90  3.01  NA
508      5     19        85   2   11.1  111.5 -1.78 -4.84  NA
529      6     20        78   1   12.1  111.9 -1.37 -4.45  NA
761      8     28        62   1   13.3  115.4  0.99 -2.70  NA
806      8     29       100   1   13.2  121.2 -1.36 -5.01  NA
```

The data required to calculate the BMI-for-age z-score are present. Given the extreme values in the **waz** variable it is likely that the BMI-for-age z-scores in these records were calculated, found to be outside the upper and lower flagging criteria, and the value for **baz** were set to missing. We should check this and recalculate the BMI-for-age z-scores.

We can use scatterplots to examine the relationship between **ageMonths**, **weight**, and **height**:

```
plot(svy$ageMonths, svy$weight)
plot(svy$ageMonths, svy$height)
plot(svy$height, svy$weight)
```

See *Figure 8.11*.

These relationships are not as simple as in younger children:

Variability in **weight** appears to increase with increasing **ageMonths**.

The relationship between **height** and **ageMonths** may not be entirely linear.

The relationship between **weight** and **height** is clearly non-linear.

All of these relationships are monotonic (see *Figure 8.10*) so we should still be able to use the Mahalanobis distance method to identify outliers:

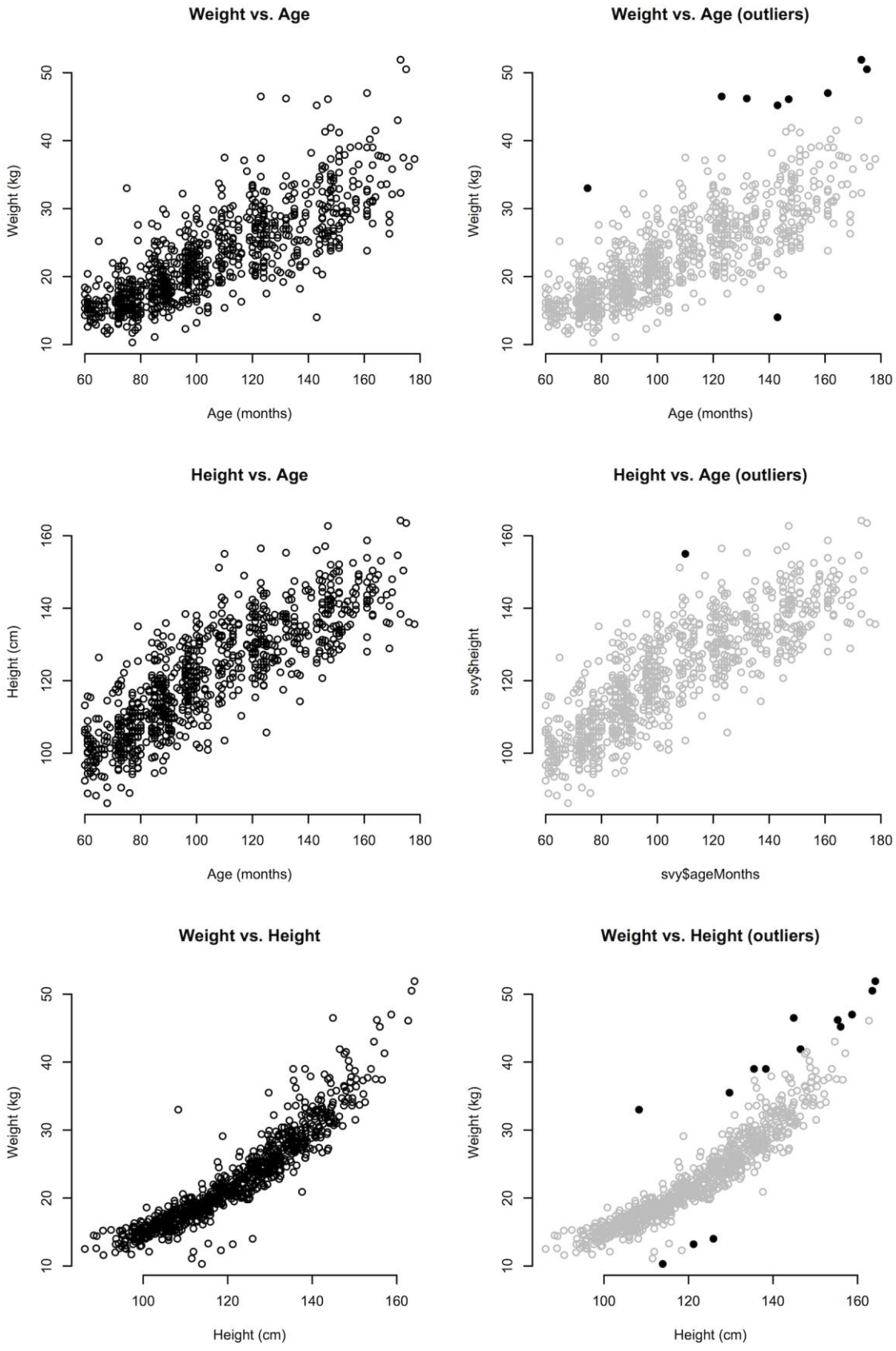
```
plot(svy$ageMonths, svy$weight,
      pch = ifelse(outliersMD(svy$ageMonths, svy$weight), 19, 1))
plot(svy$ageMonths, svy$height,
      pch = ifelse(outliersMD(svy$ageMonths, svy$height), 19, 1))
plot(svy$height, svy$weight,
      pch = ifelse(outliersMD(svy$height, svy$weight), 19, 1))
```

See *Figure 8.11*. You may want to experiment with different values of the **alpha** parameter of the **outliersMD()** function as described above. Records containing values identified as outliers can be listed:

```
svy[outliersMD(svy$ageMonths, svy$weight), ]
svy[outliersMD(svy$ageMonths, svy$height), ]
svy[outliersMD(svy$weight, svy$height), ]
```

These records can be checked, edited (if required), and anthropometric indices recalculated.

Figure 8.11. The relationship between age, height, and weight in the example dataset of school-age children. The filled circles show the outliers identified by the Mahalanobis distance.



9. Identifying outliers using flags

Flagging is a way of identifying records for which there is a strong likelihood that values of anthropometric measurements or the age of the child are incorrect. Records can then be checked and corrected, or censored (i.e. excluded), from subsequent analyses.

Flagging is a process of checking whether values of anthropometric indices are outside a given range and recording the result in one or more new variables. The result may be a set of logical (i.e. 1/0 or true/false) flag variables (i.e. one flag variable per anthropometric index) or a single variable holding a code number that classifies the nature of the detected problem(s).

Flagging is usually applied to height-for-age z-scores (HAZ), weight-for-age z-scores (WAZ), weight-for-height z-scores (WHZ), and BMI-for-age z-scores (BAZ) calculated from data collected during nutritional anthropometry surveys. The flagging process can be easily applied to other variables.

Two flagging criteria for anthropometric indices are in common use. These are the WHO flagging criteria and the SMART flagging criteria. Both methods flag records in which one or more anthropometric indices are more than a certain distance either side of a *reference value*. The two methods are summarised in *Table 9.1*.

Table 9.1. WHO and SMART flagging criteria (SMART, 2012; WHO, 2009; WHO, 2011) applied to the anthropometric indices of children, with the old criteria applied to the NCHS references.

	WHO		**SMART**		**NCHS**	
	Reference mean (zero)		**Survey mean (observed)**		**Reference mean (zero)**	
Anthropometric index	**Lower limit**	**Upper limit**	**Lower limit**	**Upper limit**	**Lower limit**	**Upper limit**
Weight-for-length	-5	+5	-3	+3	-4	+6
Weight-for-height	-5	+5	-3	+3	-4	+6
Length-for-age	-6	+6	-3	+3	-6	+6
Height-for-age	-6	+6	-3	+3	-6	+6
Weight-for-age	-6	+5	-3	+3	-6	+6
BMI-for-age	-5	+5	NA*	NA*	-5	+5
MUAC-for-age	-5	+5	NA*	NA*	-5	+5
Triceps-for-age	-5	+5	NA*	NA*	-5	+5
Subscapular skinfold-for-age	-5	+5	NA*	NA*	-5	+5
Head circumference-for-age	-5	+5	NA*	NA*	-5	+5

* NA = not available

Applying flagging criteria is a matter of checking that individual values of these indices are within the lower and upper limits shown in *Table 9.1*. Values that are outside of these limits are flagged in a new variable.

The WHO ranges for the other anthropometric indices (MUAC-for-age, triceps skinfold-for-age, subscapular skinfold-for-age and head circumference-for-age) are -5 to +5 z-scores. These indices are calculated by WHO Anthro software but not by ENA for SMART (see Appendix 2).

The WHO criteria are simple *biologically plausible* ranges around the reference mean of zero. If, for example, a value for WHZ is below −5 or above +5 then the record is flagged to indicate a likely problem with WHZ. This will usually be caused by an erroneous value of weight or height.

Note that values outside these flagging limits may be observed in children admitted into therapeutic feeding programmes.

SMART criteria are more complicated. They require the mean value of the index to be calculated from the survey data. This is then used as the reference value and then 3 z-scores are added or subtracted to create a range. For example, if a value of WHZ is below:

mean WHZ – 3

or above:

mean WHZ + 3

then the record is flagged to indicate a likely problem with WHZ.

For example, a mean WHZ of –1.15 gives lower and upper SMART flagging limits of:

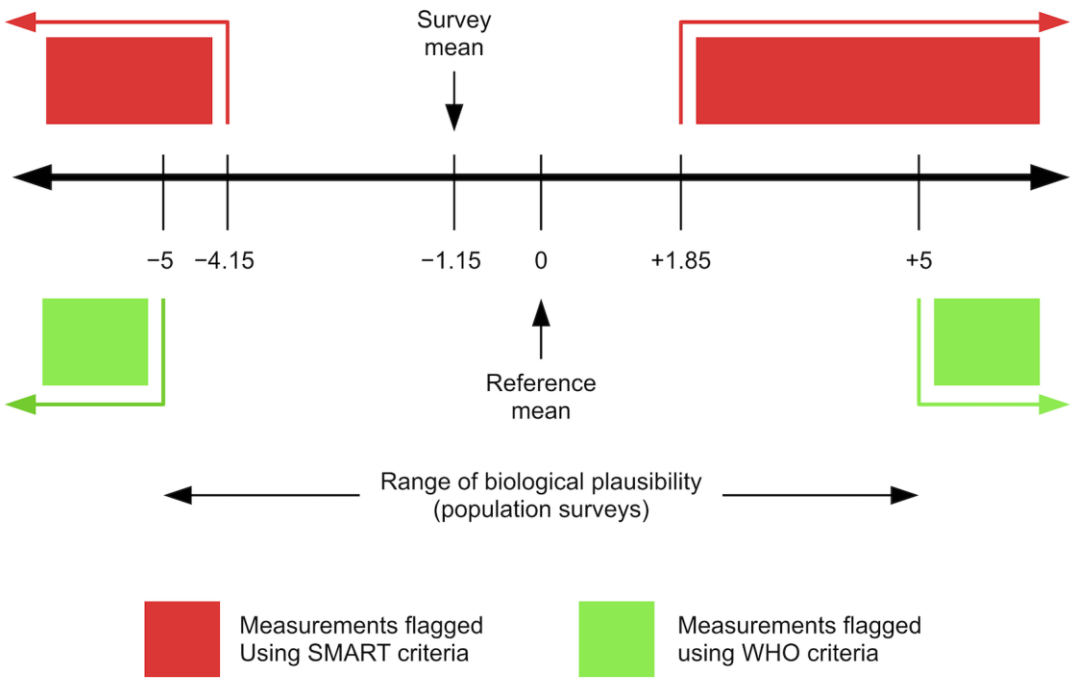
–1.15 – 3 = –4.15

and:

–1.15 + 3 = +1.85

respectively. These limits may incorrectly flag biologically plausible values. See *Figure 9.1*.

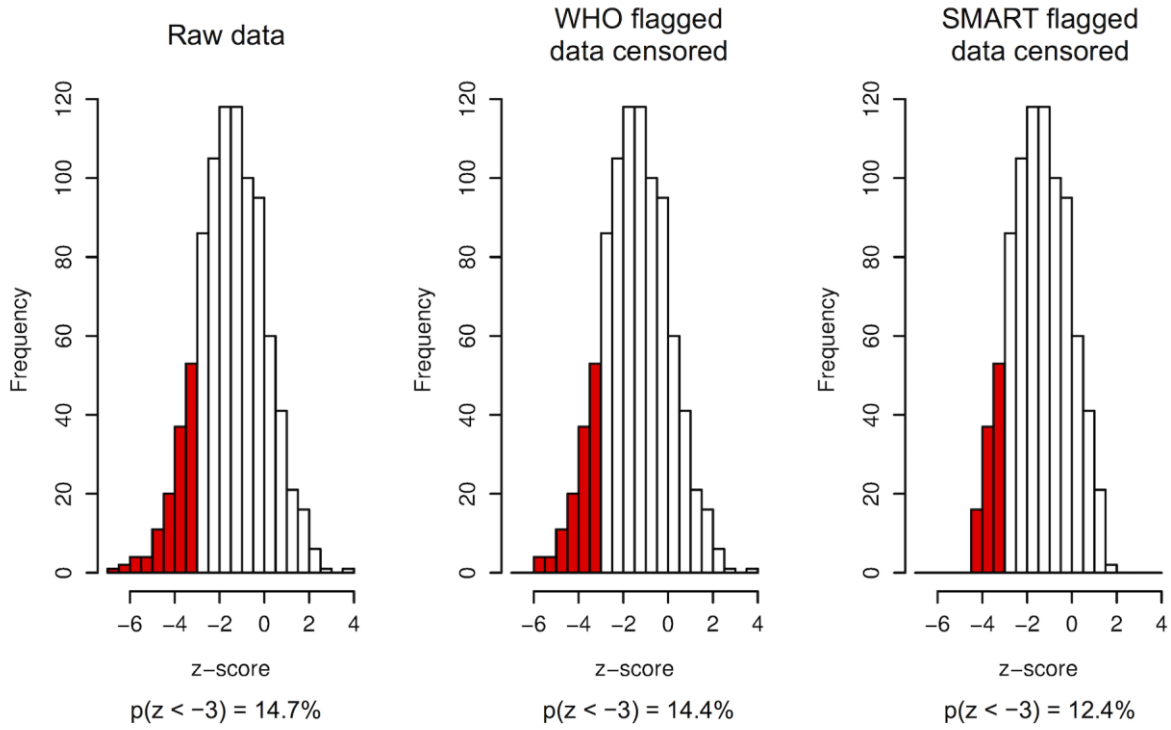
Figure 9.1. Example of WHO and SMART flagging criteria for weight-for-height z-scores (WHZ). WHO flagging criteria are wider for height-for-age z-scores (HAZ) and weight-for-age z-scores (WAZ) (see Table 9.1)



The WHO and SMART flagging criteria will flag different but overlapping sets of measurements. This means that survey results can be affected by the flagging criteria used. This is because the prevalence of an indicator describes the proportion of values in the one of the "tails" of the distribution of an index (see *Figure 9.2*).

The SMART flagging criteria will usually flag more records than the WHO flagging criteria. This will act to reduce the estimated prevalence (see *Figure 9.2*). This will be occur more often when the prevalence of severe forms of undernutrition is high.

Figure 9.2. Prevalence is in the "tails" of the distribution. The estimated prevalence is shown for cases defined using −3 z-scores below the reference median (i.e. zero). The red bars show the cases remaining after "outliers" to the left have been censored. The area covered by the red bars represents the estimated prevalence after flagged values have been censored. The estimated prevalence is reported below each plot as p(z < −3).



There are some problems with using the SMART flagging criteria:

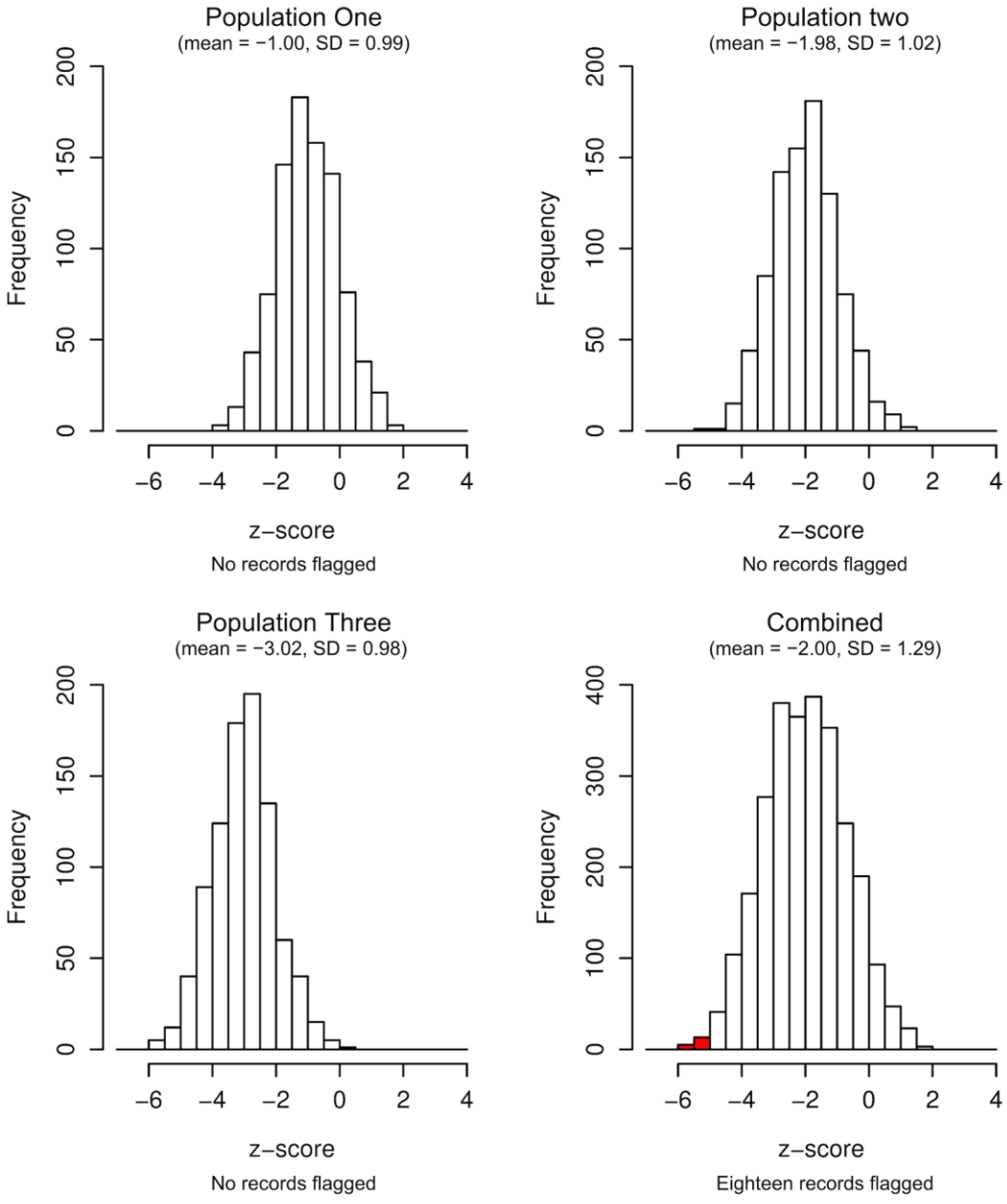
1. Flagging is about detecting outlier values. The SMART flagging criteria use distance from the sample mean, but the value of the mean can be strongly influenced by the presence of outliers. This could be overcome by, for example, using the median or a trimmed mean instead of the reference mean. If you do this you will **not** be using the SMART flagging criteria.

2. SMART flagging criteria are supposed to define outliers using *statistically plausible* limits. The underlying principle is that, for a normally distributed variable, we expect 99.87% of all values to lie within three sample standard deviations of the sample mean. If we exclude records with values more than three standard deviations from the mean then we would incorrectly flag very few records (i.e. 0.13% of the total) as problematic. The SMART method assumes that the distribution of each anthropometric index in a population is always perfectly normal and that the standard deviation is always exactly one. This assumption is almost always violated. If it is

violated then the use of the SMART flagging criteria may lead to records being flagged inappropriately. There are ways (e.g. transforming data toward normality, using the sample standard deviation) to avoid this problem but using them would also **not** be using the SMART flagging criteria.

3. Wide-area surveys such as MICS and DHS will usually collect data from many populations. Each population may have different distributions of anthropometric indices and different prevalence of anthropometric indicators. In this case the mean of the entire survey sample will **not** be a suitable reference mean and the assumed standard deviation (i.e. SD = 1) will usually be too narrow to set limits that define statistical outliers. This will lead to records being flagged incorrectly. This is illustrated in *Figure 9.3*. Stratum or district specific means should be used instead of whole sample means, but this may not solve the problem entirely.

Figure 9.3. Combining populations with different means and similar standard deviations increases the overall standard deviation and may cause SMART flagging criteria to flag records inappropriately. Flagged records are shaded red.



If SMART flagging criteria have already been applied to data and the flagged records have been removed from the dataset, then a subsequent application of the SMART flagging criteria will tend to flag additional records. SMART flagging criteria should, therefore, only be applied to raw data. Do not apply SMART flagging criteria to data from which flagged records have been removed.

It is important to note that only one set of flagging criteria, either WHO or SMART, should be used at any one time.

The WHO and SMART flagging criteria are designed to be applied to samples of children measured in surveys. They should **not** be applied to samples of severely malnourished or sick children.

Software such as *ENA from SMART*, *EpiInfo*, *WHO Anthro, WHO AnthroPlus*, and scripts or macros for *R*, *SAS*, *SPSS*, and *STATA* provided by the WHO (see Appendix 1 for more details) are frequently used to calculate anthropometric indices from anthropometric data and then apply flagging criteria to the data. It is quite common to receive data to which flagging criteria have already been applied and contain one or more flag variables. You may use these flags if you are sure which flagging criteria have been applied. If you are unsure which flagging criteria have been applied then you should apply your flagging criteria of choice using one of these software packages or the procedures outlined in this section. You may also need to recalculate anthropometric indices using WHO reference values if they were calculated using NCHS, CDC, or local growth references.

9.2 Applying WHO flagging criteria to survey data

For a first exercise, we will apply the WHO flagging criteria to survey data.

We will retrieve a survey dataset:

```
svy <- read.table("flag.ex01.csv", header = TRUE, sep = ",")
head(svy)
```

The file **`flag.ex01.csv`** is a comma-separated-value (CSV) file containing anthropometric data from a recent SMART survey in Sudan.

Applying WHO flagging criteria is straightforward. We first create a column that will contain the flag code and set this to zero (i.e. no flags) for all records:

```
svy$flag <- 0
head(svy)
```

Then we apply the flagging criteria for each index. Here we apply the WHO flagging criteria to the HAZ index:

```
svy$flag <- ifelse(!is.na(svy$haz) & (svy$haz < -6 | svy$haz > 6),
       svy$flag + 1, svy$flag)
```

This can be translated as "if HAZ is not missing and HAZ is below -6 or HAZ is above +6 then add 1 to the flag variable else leave the flag variable unchanged".

Be careful when using the **<** comparison operator with negative numbers. Always insert a space between the **<** and **–** characters. *R* interprets **<-** as an assignment operator and may produce unexpected and unwanted results without issuing a warning or error message.

Here we apply the WHO flagging criteria to the WHZ index:

```
svy$flag <- ifelse(!is.na(svy$whz) & (svy$whz < -5 | svy$whz > 5),
       svy$flag + 2, svy$flag)
```

Here we apply the WHO flagging criteria to the WAZ index:

```
svy$flag <- ifelse(!is.na(svy$waz) & (svy$waz < -6 | svy$waz > 5),
       svy$flag + 4, svy$flag)
```

Note that each time we apply a flagging criteria we increase the value of the same flagging variable (`svy$flag`) by the next power of two when a problem is detected:

We started with zero

Then we added 2^0 (i.e. 1) if HAZ was out of range.

Then we added 2^1 (i.e. 2) if WHZ was out of range.

Then we added 2^2 (i.e. 4) if WAZ was out of range.

If we had another index then we would use 2^3 (i.e. 8) to flag a problem in that index.

The advantage of using this coding scheme is that it compactly codes all possible combinations of problems in a single variable (see *Table 9.2*).

There are a number of flagged records in the example dataset. This:

```
table(svy$flag)
```

returns:

```
  0   1   2   3   5   6
751   9  12   9   2   3
```

This table shows the relative frequency of detected problems. See *Table 9.2* to find the meaning of each of the codes and the suggested action.

Table 9.2. Flagging codes for anthropometric indices calculated for children aged 0 - 59 months based on powers of two, and their meanings.

	Problem detected with* . . .			
Code	**HAZ**	**WHZ**	**WAZ**	**Suggested action(s)**
0	○	○	○	None
1	●	○	○	Check height and age
2	○	●	○	Check weight and height
3	●	●	○	Check height
4	○	○	●	Check weight and age
5	●	○	●	Check age
6	○	●	●	Check weight
7	●	●	●	Check age, height, and weight

* The indices are height-for age z-score (HAZ), weight-for-height z-score (WHZ) and weight-for-age z-score (WAZ).

The number of flagged records can be found using:

```
table(svy$flag != 0)["TRUE"]
```

which returns:

```
TRUE
  35
```

The proportion of records that are flagged can be found using:

```
prop.table(table(svy$flag != 0))["TRUE"]
```

This returns:

```
      TRUE
0.04452926
```

About 4.45% of records are flagged.

Note that missing values are not flagged. It can be useful to check missing values to see if there are missing component measurements or if a component measurement is out of range for the calculation of index values (e.g. WAZ is only calculated for children aged ten years or younger). This issue can be explored by selection and listing. For example:

```
svy[is.na(svy$whz), c("weight", "height", "whz")]
```

This returns:

```
  weight height whz
8    8.1     NA  NA
```

There is one missing value for **whz** in record **8**.This is due to a missing value for **height** (shown as **NA**). and **haz** will also be missing. It may be possible to fix this issue if the missing data are available on paper forms.

Flagging has a dual role:

1. It is a data-checking tool. If you have access to data collection forms you will be often able to check records and fix data-entry errors.

2. It is a measure of data-quality. Flagged records can indicate problems with measurement, recording, data-entry, and data-checking. The proportion of flagged records in a dataset should, ideally, be below 5%. SMART guidelines consider proportions above 7.5% to be problematic (SMART, 2015). We found that 4.45% of records in the example dataset were flagged. The data are of acceptable quality.

We can use:

```
svy[svy$flag != 0, ]
```

to display the flagged records.

This:

```
svy[svy$flag != 0, c("psu","child","flag")]
```

produces a more compact list.

In the example dataset records are identified using a combination of the **psu** and **child** variables.

The listed records can be checked and edited (see *Table 9.3*). Anthropometric indices can then be recalculated and the flagging process repeated until all records that can be fixed have been fixed.

Records that cannot be fixed can be censored during analysis. Records are usually censored on an index-by-index basis. For example, an analysis based on WHZ would censor records in which the flag variable is 2, 3, 6, or 7.

Table 9.3 shows censoring rules for each index:

Table 9.3. Censoring rules for each anthropometric index

Analysis uses* . . .	Censor if flag code is . . .
HAZ	1, 3, 5, or 7
WHZ	2, 3, 6, or 7
WAZ	4, 5, 6, or 7

* The indices are height-for age z-score (HAZ), weight-for-height z-score (WHZ) and weight-for-age z-score (WAZ).

You should be very careful when applying censoring rules. An analysis of prevalence using WHZ, for example, will usually include children with oedema because a commonly used case-definition for acute malnutrition is:

WHZ < -2 or bilateral pitting oedema

If you want to use case-definitions that include oedema then you should be careful not to exclude children with oedema when censoring flagged records. For an analysis using WAZ you might want to exclude oedema cases.

9.3 Applying SMART flagging criteria to survey data

In the next exercise we will apply SMART flagging criteria to the same survey dataset.

We will retrieve the survey dataset:

```
svy <- read.table("flag.ex01.csv", header = TRUE, sep = ",")
head(svy)
```

and create a column that will contain the flag code and set this to zero (i.e. no flags) for all records:

```
svy$flag <- 0
```

Applying SMART flagging criteria requires us to first calculate a mean index value:

```
meanHAZ <- mean(svy$haz, na.rm = TRUE)
```

```
meanHAZ
```

and then to use this mean value to define flagging ranges:

```
svy$flag <- ifelse(!is.na(svy$haz) &
       (svy$haz < (meanHAZ - 3) | svy$haz > (meanHAZ + 3)),
              svy$flag + 1, svy$flag)
```

We do this for each index:

```
meanWHZ <- mean(svy$whz, na.rm = TRUE)

svy$flag <- ifelse(!is.na(svy$whz) &
       (svy$whz < (meanWHZ - 3) | svy$whz > (meanWHZ + 3)),
              svy$flag + 2, svy$flag)

meanWAZ <- mean(svy$waz, na.rm = TRUE)

svy$flag <- ifelse(!is.na(svy$waz) &
       (svy$waz < (meanWAZ - 3) | svy$waz > (meanWAZ + 3)),
              svy$flag + 4, svy$flag)
```

There are a number of flagged records in the example dataset.

This:

```
table(svy$flag)
```

returns:

```
  0   1   2   3   4   5   6   7
660  59  11  16   1  19  16   4
```

This table shows the relative frequency of detected problems. See *Table 9.2* to find the meaning of each of the codes. The number of flagged records can be found using:

```
table(svy$flag != 0)["TRUE"]
```

which returns:

```
TRUE
 126
```

The proportion of records that are flagged can be found using:

```
prop.table(table(svy$flag != 0))["TRUE"]
```

which returns:

```
     TRUE
0.1603053
```

About 16% of records are flagged. This is a very high proportion of records flagged.

Note how the SMART flagging criteria identify considerably more records (126 records flagged) than the WHO flagging criteria (35 records flagged). In this example the SMART flagging criteria flagged 91 *biologically plausible* records.

We can list flagged records using:

```
svy[svy$flag != 0, ]
```

The listed records can be checked and edited (see *Table 9.2*). Anthropometric indices can then be recalculated and the flagging process repeated until all records that can be fixed have been fixed.

When listing records or displaying very large tables you may see a message like this:

```
[ reached getOption("max.print") -- omitted 43 rows ]
```

The **`max.print`** option sets a limit on the length of information that can be displayed by a single command. You can alter this behaviour using:

```
options(max.print  = 99999)
```

9.4 Flagging data from older children

The process of flagging anthropometric indices in older children is very similar to that used with younger children.

We will retrieve a survey dataset:

```
svy <- read.table("flag.ex02.csv", header = TRUE, sep = ",")
head(svy)
```

The file **`flag.ex02.csv`** is a comma-separated-value (CSV) file containing anthropometric data from a survey of children aged 11 year or older and attending school in Ethiopia.

The variables of interest are height-for-age z-score (**`haz`**) and BMI-for-age z-score (**`baz`**). We will apply the WHO flagging criteria (see *Table 9.1*) to these variables:

```
svy$flag <- 0
svy$flag <- ifelse(!is.na(svy$haz) & (svy$haz < -6 | svy$haz > 6),
       svy$flag + 1, svy$flag)
svy$flag <- ifelse(!is.na(svy$baz) & (svy$baz < -5 | svy$baz > 5),
       svy$flag + 2, svy$flag)
```

Note that we do not usually apply SMART flagging criteria to older children (i.e. > 59 months).

The coding of the **`flag`** variable is shown in *Table 9.4*.

Table 9.4. Flagging codes for anthropometric indices calculated for children aged 5 to 19 years based on powers of two, and their meanings.

	Problem with*...		
Code	**HAZ**	**BAZ**	**Suggested action(s)**
0	○	○	None
1	●	○	Check height and age
2	○	●	Check weight, height and age
3	●	●	Check weight, height and age

* The indices are height-for age z-score (HAZ) and BMI-for-age z-score (BAZ).

This:

```
table(svy$flag)
```

returns:

```
  0   1   2
960   2  11
```

This table shows the relative frequency of detected problems. See *Table 9.4* to find the meaning of each of the codes. The number of flagged records can be found using:

```
table(svy$flag != 0)["TRUE"]
```

which returns:

```
TRUE
  13
```

The proportion of records that are flagged can be found using:

```
prop.table(table(svy$flag != 0))["TRUE"]
```

which returns:

```
      TRUE
0.01336074
```

About 1.3% of records are flagged. This is an acceptably low proportion of records flagged.

We can list flagged records using:

```
svy[svy$flag != 0, ]
```

The listed records can be checked and edited (see *Table 9.4*). Anthropometric indices can then be recalculated and the flagging process repeated until all records that can be fixed have been fixed.

9.5 Applying SMART flagging criteria to national survey data

Now we will look at flagging records in large national survey using the SMART flagging criteria.

We will retrieve a dataset.

```
svy <- read.table("flag.ex03.csv", header = TRUE, sep = ",")
head(svy)
```

The file **`flag.ex03.csv`** is a comma-separated-value (CSV) file containing anthropometric data from a national survey in Nigeria.

The data were collected using methods similar to MICS and DHS surveys. The only difference is that the survey concentrated on collecting anthropometric data for children aged between 6 and 59 months.

Data are stratified by **region** and by **state** within **region**:

```
table(svy$region)
table(svy$state)
```

In this exercise we will concentrate on WHZ.

SMART flagging criteria use the sample mean as the reference value (see *Table 9.1*).

The overall mean WHZ:

```
mean(svy$whz, na.rm = TRUE)
```

is:

```
-0.4555963
```

We can examine mean WHZ by region:

```
by(svy$whz, svy$region, mean, na.rm = TRUE)
```

This returns:

```
svy$region: NC
[1] -0.2581365
--------------
svy$region: NE
[1] -0.6218837
--------------
svy$region: NW
[1] -0.5656509
--------------
svy$region: SE
[1] -0.364248
--------------
svy$region: SS
[1] -0.2675344
--------------
svy$region: SW
[1] -0.4835342
```

The mean WHZ varies between regions.

We can examine the mean WHZ by **region** and **state**.

First we will create a new variable that combines **region** and **state**:

```
svy$regionState <- paste(svy$region, svy$state, sep = ":")
head(svy)
table(svy$regionState)
```

We can now examine the mean WHZ for each combination of **region** and **state** in the survey dataset:

```
by(svy$whz, svy$regionState, mean, na.rm = TRUE)
```

The long output can be made more compact, easier to read, and easier to work with:

```
means <- by(svy$whz, svy$regionState, mean, na.rm = TRUE)
means <- means[1:length(means)]
means
```

The saved **means** object can be examined:

```
summary(means)
hist(means)
```

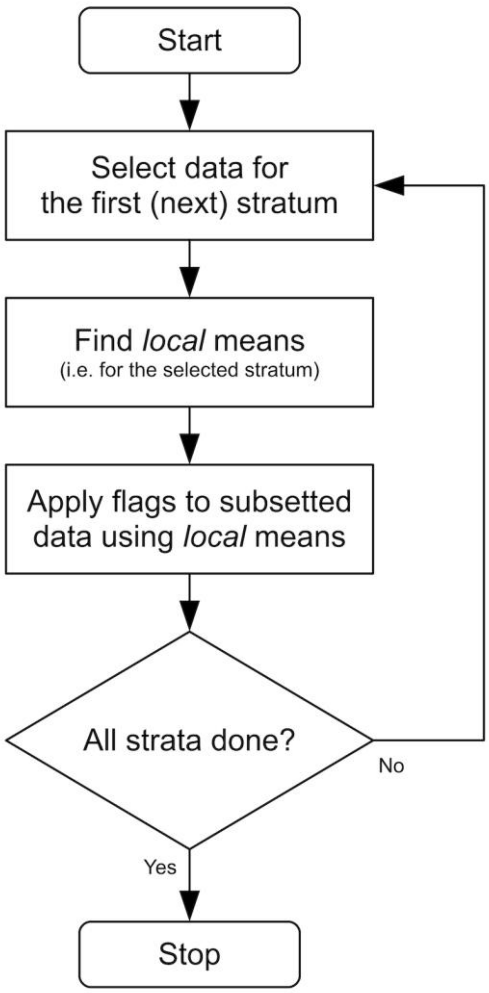
The mean WHZ varies between regions and between states within each region.

This is **not** a problem if we want to use WHO flagging criteria. We can simply apply the WHO flagging criteria to the whole dataset.

If we want to use SMART flagging criteria with these data then we will need to decide which mean to use as the reference mean. The most useful reference mean will usually be the one for the smallest spatial strata. This is **state** in the example data. This is not ideal as states are very large areas and may contain different ethnic groups and livelihood zones. It is the best that we can do.

The process of applying SMART flagging criteria to a national or wide area survey is shown in *Flowchart 9.1*.

Flowchart 9.1. Applying SMART flagging criteria in national or wide area survey



When dealing with national surveys we will usually want to use the WHO flagging criteria.

In some cases we may need to use the SMART flagging criteria.

The NIPN data quality toolkit provides an *R* language function called **`national.SMART()`** that simplifies the process of applying SMART flagging criteria to wide area surveys.

The **`national.SMART()`** function has three parameters shown in *Table 9.5*.

Table 9.5. The parameters used by the **`national.SMART()`** function in the NIPN anthropometry toolkit.

Parameter	Contents	Value for the example survey dataset	Notes
`x`	Survey dataset	`svy`	An R data.frame object
`strata`	The column in the survey dataset for the smallest spatial strata.	`"state"`	Quotes are needed.
`indices`	Variable names for the indices to be included in the flagging process.	`c("haz", "whz", "waz")`	Quotes are needed. The default value is as in the example. Change this to reflect indices and their column names. Coding increases by powers of two.

The **`national.SMART()`** function returns a copy of the survey dataset (**`x`**) with SMART flags added in a column names **`flagSMART`.**

We will use the **`national.SMART()`** function with the example data:

```
svyFlagged <- national.SMART(x = svy, strata = "state")
```

Examining the results:

```
table(svyFlagged$flagSMART)
table(svyFlagged$flagSMART != 0)
prop.table(table(svyFlagged$flagSMART != 0))
prop.table(table(svyFlagged$flagSMART != 0))["TRUE"]
svyFlagged[svyFlagged$flagSMART != 0, ]
```

A more compact list of flagged records can be produced using:

```
subset(svyFlagged, subset = (svyFlagged$flagSMART != 0),
       select = c(psu, regionState, age:height, flagSMART))
```

We can use the `write.table()` function to save the data with flags to a file for later analysis in *R* or in a statistics package:

```
write.table(svyFlagged, file = "svyFlagged.csv", sep = ",",
       quote = FALSE,  row.names = FALSE)
```

10. Assessing the distribution of anthropometric variables, indices, and indicators

In this section we will examine the distribution of anthropometric variables (e.g. weight, height, and MUAC), anthropometric indices (e.g. WHZ, HAZ, and WHZ), and anthropometric indicators (e.g. wasted, stunted, and underweight).

Topics such as the distribution of age, age by sex, age-heaping, and digit preference are covered in other sections of this toolkit.

10.1 Numerical summaries

We will retrieve a survey dataset:

```
svy <- read.table("dist.ex01.csv", header = TRUE, sep = ",")
head(svy)
```

The file **dist.ex01.csv** is a comma-separated-value (CSV) file containing anthropometric data from a SMART survey in Kabul, Afghanistan.

The **summary()** function in *R* provides a six-figure summary (i.e. minimum, first quartile, median, means, third quartile, and maximum) of a numeric variable. For example:

```
summary(svy$weight)
```

returns:

```
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
4.90    9.00   11.00   11.13   13.10   20.70
```

The six-figure summary does not report the standard deviation.

The **sd()** function in *R* calculates the standard deviation. For example:

```
sd(svy$weight)
```

returns:

```
2.802065
```

The **sd()** function may return **NA**. This will happen if there are missing values in the specified variable.

If this happens you can instruct the function to ignore missing values:

```
sd(svy$weight, na.rm = TRUE)
```

returns the same value:

```
2.802065
```

Using the **na.rm** parameter in this way (i.e. specifying **na.rm = TRUE**) works with many descriptive functions in *R* (see *Table 10.1*).

Table 10.1 : Some descriptive functions in *R.*

Function*	Returns
`mean()`	Mean of the specified variable
`sd()`	Standard deviation of the specified variable
`var()`	Variance of the specified variable
`median()`	Median of the specified variable
`min()`	Minimum of the specified variable
`max()`	Maximum of the specified variable
`range()`	Range of the specified variable
`IQR()`	Interquartile range of the specified variable
`quantile()`	Quantiles of the specified variable
`summary()`	Six-figure summary of the specified variable
`mad()`	Adjusted median absolute deviation
`length()`	Number of observations in the specified variable
`nrow()`	Number of rows in a **`data.frame`** (dataset)

* If a function returns **`NA`** this may be because the variable contains missing values. Use **`na.rm = TRUE`** to instruct the function to ignore missing values.

10.2 Graphical and numerical summaries

Numerical summaries are useful for checking that data are within an expected range.

Graphical methods are often more informative than numerical summaries.

A key graphical method for examining the distribution of a variable is the histogram.

For example:

```
hist(svy$weight)
```

displays a histogram of the **`weight`** variable in the example dataset (see *Figure 10.1*).

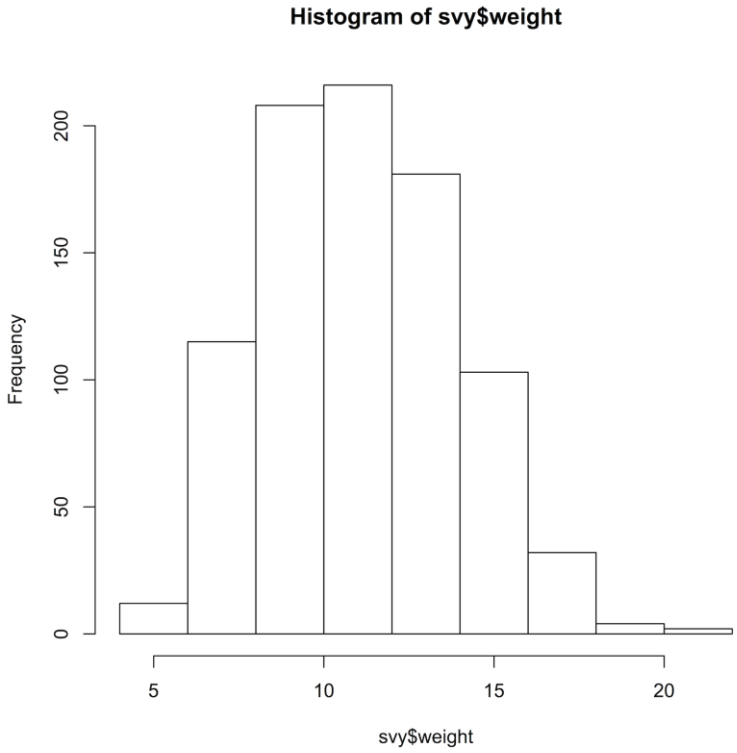
We need to be careful when examining the distribution of measurements, as they may vary by sex.

For example:

```
hist(svy$height)
```

will display the heights of both males and females. That is, it will display two separate distributions as if they were a single distribution.

Figure 10.1 A histogram showing the distribution of the **`weight`** variable in the example dataset



In this case it is sensible to look at the data for males and the data for females using separate histograms:

```
hist(svy$height[svy$sex == 1])
hist(svy$height[svy$sex == 2])
```

or using a box-plot:

```
boxplot(svy$height ~ svy$sex, names = c("M", "F"),
      xlab = "Sex", ylab = "Height (cm)", main = "Height by sex")
```

Numerical summaries can also be used:

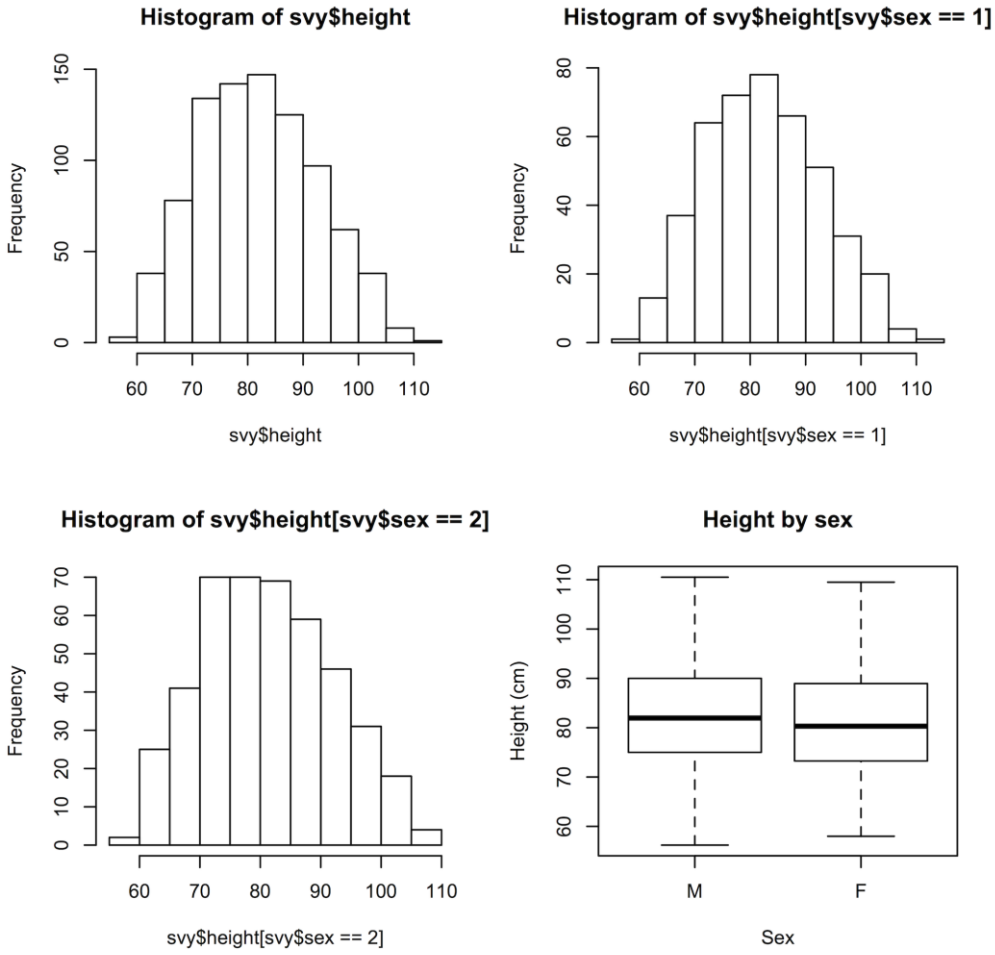
```
by(svy$height, svy$sex, summary)
```

This returns:

```
svy$sex: 1
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  56.20   75.00   81.95   82.49   90.00  110.50
-----------------------------------------------
svy$sex: 2
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  58.00   73.25   80.30   81.30   88.95  109.50
```

All of the above plots are shown in *Figure 10.2*.

Figure 10.2 Histograms and a box-plot of the **`height`** variable in the example dataset



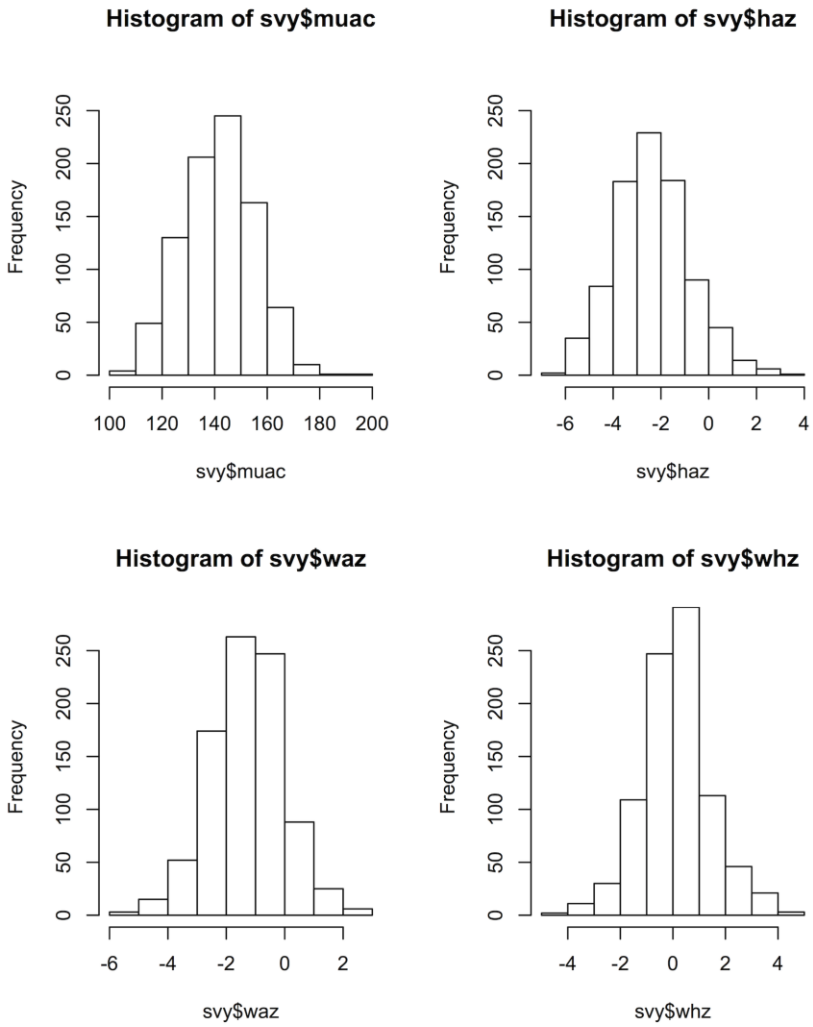
10.3 Normal distributions

With anthropometric variables and indices we usually expect a symmetrical (or nearly symmetrical) "bell-shaped" distribution. The variables and indices of interest are usually:

```
hist(svy$muac)
hist(svy$haz)
hist(svy$waz)
hist(svy$whz)
```

These plots are shown in *Figure 10.3*.

Figure 10.3 : Histograms showing the distribution of anthropometric indices in the example dataset



The number and size of the "intervals" (**`breaks`**) used when plotting a histogram is calculated to produce a useful plot. The intervals used are based on the range of the data.

You can specify a different set of **`breaks`** for the **`hist()`** function to use. For example:

```
hist(svy$haz, breaks = "scott")
```

calculates intervals using the standard deviation and the sample size. This:

```
hist(svy$haz, breaks = "FD")
```

calculates intervals using the inter-quartile range. This:

```
hist(svy$haz, breaks = 40)
```

will use about **`40`** intervals. This:

```
hist(svy$haz, breaks =
       seq(from = floor(min(svy$haz)), to = ceiling(max(svy$haz)), by = 0.5))
```

uses intervals that are **`0.5`** z-scores wide over the full range of **`haz`**.

All of these plots show nearly symmetrical “bell-shaped” distributions.

The *ideal* symmetrical "bell-shaped" distribution is the *normal distribution*.

There are a number of ways of assessing whether a variable is *normally distributed*.

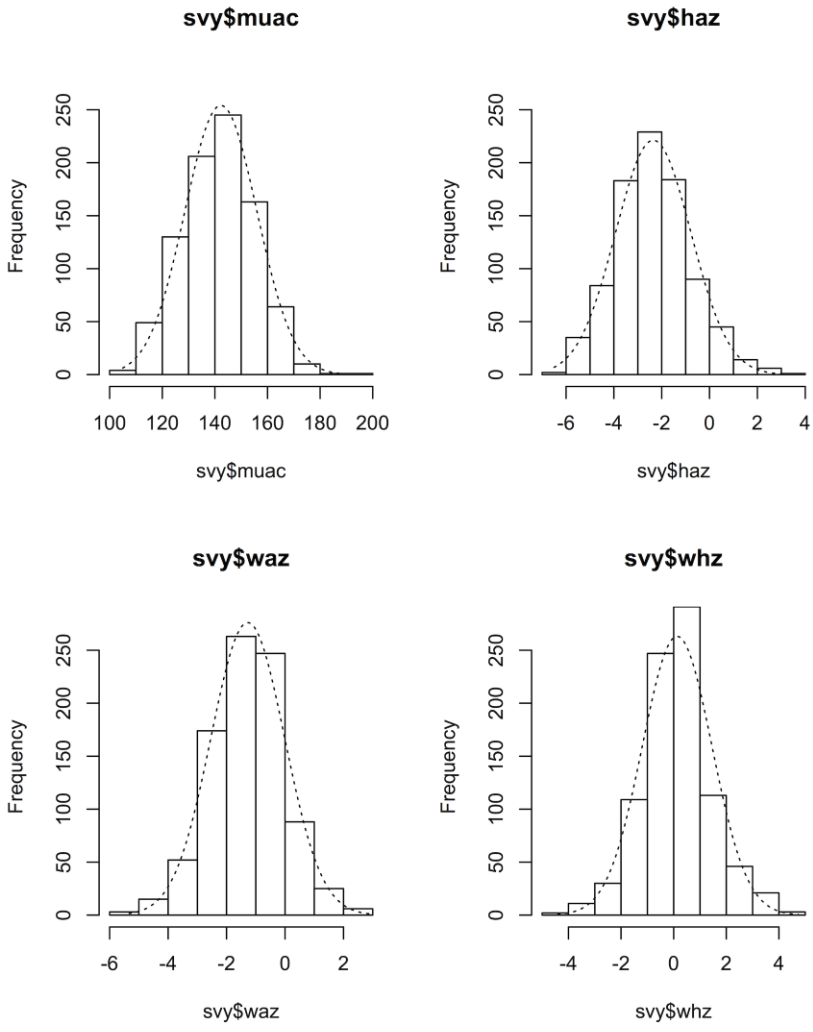
The first way of assessing whether a variable is normally distributed is a simple “by-eye” assessment as we have already done using histograms.

The NIPN data quality toolkit provides an *R* language function called **`histNormal()`** that can help with “by-eye” assessments by superimposing a normal curve on a histogram of the variable of interest:

```
histNormal(svy$muac)
histNormal(svy$haz)
histNormal(svy$waz)
histNormal(svy$whz)
```

These plots are shown in *Figure 10.4*. All variables appear to be approximately normally distributed.

Figure 10.4 : Histograms of anthropometric indices with Normal curves superimposed



Changing the **`breaks`** parameter may make a histogram easier to "read". For example:

```
histNormal(svy$haz, breaks = 15)
```

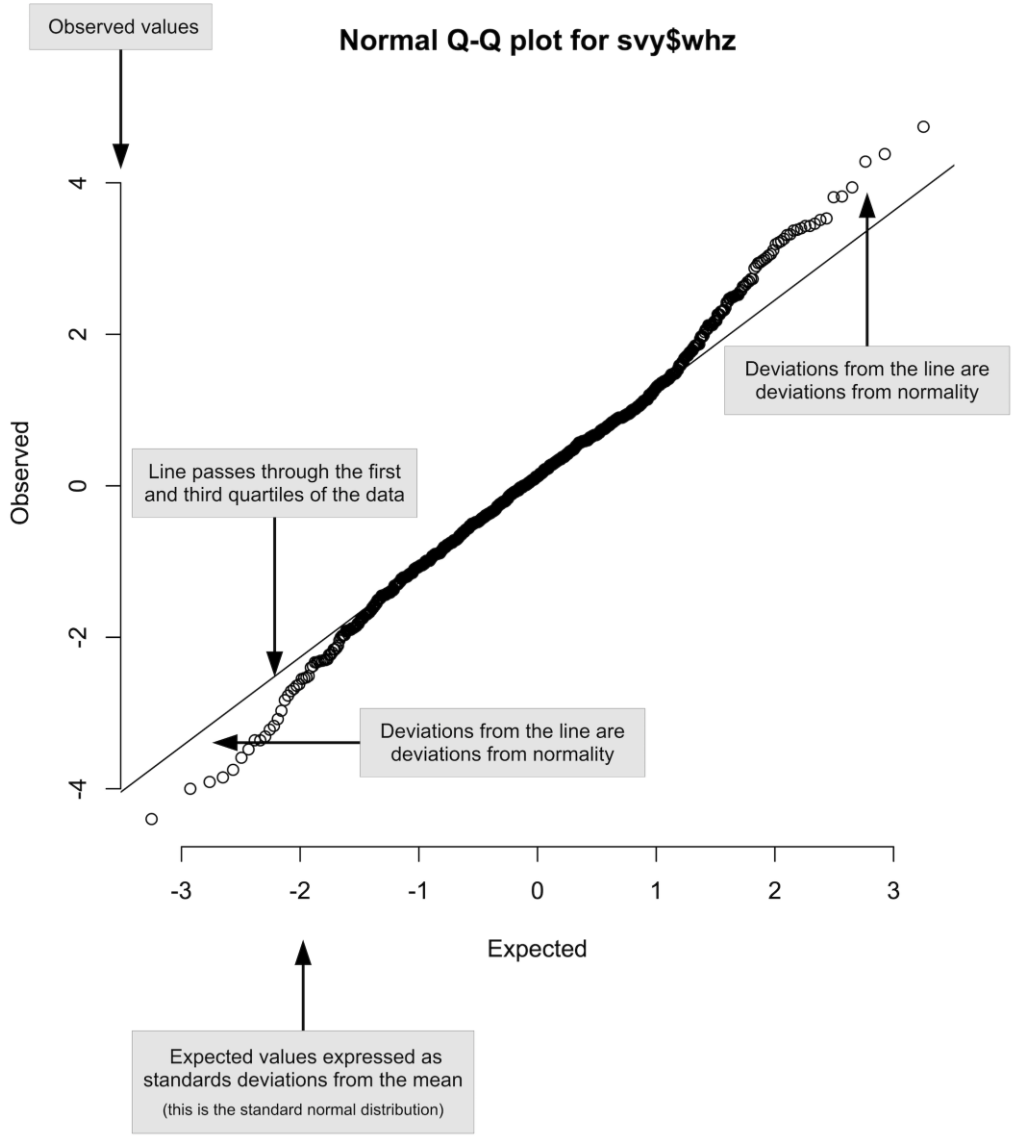
Another graphical method for assessing whether a variable is normally distributed is the *normal quantile-quantile plot*. These are easy to produce using *R*.

The NIPN data quality toolkit provided a helper function called **`qqNormalPlot()`** that produces a slightly enhanced normal quantile-quantile plot:

```
qqNormalPlot(svy$whz)
```

This plot is shown (with annotations) in *Figure 10.5*. In this example the tails of the distribution contain more cases than would be expected in a perfectly normally distributed variable.

Figure 10.5 : Annotated normal quantile-quantile plot of the **`whz`** variable in the example dataset

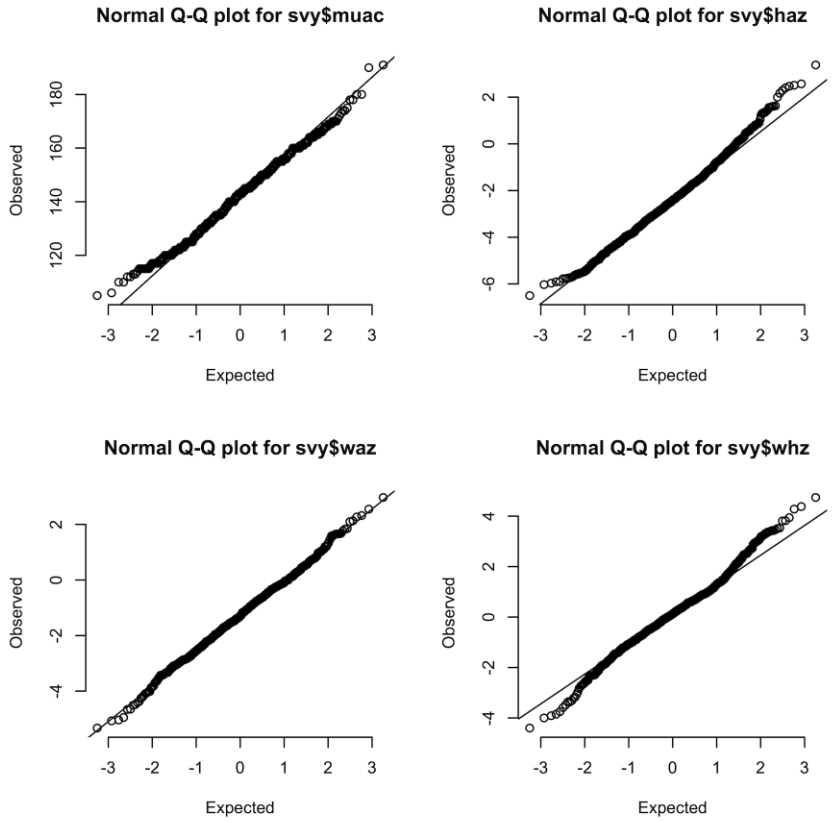


We should examine all of the relevant variables:

```
qqNormalPlot(svy$muac)
qqNormalPlot(svy$haz)
qqNormalPlot(svy$waz)
qqNormalPlot(svy$whz)
```

These plots are shown in *Figure 10.6*. There is evidence of small deviations from normality in **muac**, **haz**, and **whz**.

Figure 10.6 Normal quantile-quantile plots of anthropometric indices in the example dataset



A final way of assessing normality is to use a formal statistical significance test. The preferred test is the *Shapiro-Wilk test of normality*:

```
shapiro.test(svy$muac)
shapiro.test(svy$haz)
shapiro.test(svy$waz)
shapiro.test(svy$whz)
```

These tests indicate that **muac**, **haz**, and **whz** are significantly non-normal. Examination of the histograms and the normal quantile-quantile plots show that the deviation from normality in these indices are not particular marked. All indices have symmetrical, or nearly symmetrical, "bell-shaped" distributions.

We need to be careful when using significance tests such as the *Shapiro-Wilk test of normality* because the results can be strongly influenced by the sample size.

Small sample sizes can lead to tests missing large effects and large sample sizes can lead to tests identifying small effects as highly significant.

The analysis above found some highly significant but small deviations from normality that would probably not have been detected by a significance test if a smaller sample size had been used.

We can simulate a considerably smaller sample size by taking, for example, every fourth **`muac`** value:

```
length(svy$muac)
oneQuarter <- svy$muac[seq(from = 1, to = length(svy$muac), by = 4)]
length(oneQuarter)
```

Inspecting this smaller sample graphically:

```
histNormal(oneQuarter)
qqNormalPlot(oneQuarter)
```

yields results similar to those found when the complete sample was used, but the formal test:

```
shapiro.test(oneQuarter)
```

is no longer significant at $p < 0.05$.

If a distribution appears to be normal (i.e. has a symmetrical, or nearly symmetrical, "bell-shaped" distribution) then it is usually safe to assume normality and to use statistical procedures that assume normality. Formal tests for normality can be misleading when sample sizes of more than a few hundred cases are used. Graphical methods are not very useful when sample sizes are small. Formal test are not very useful when sample sizes are large. The sample sizes of most anthropometry surveys will be large enough to cause formal tests for normality to identify small deviations from normality as highly significant.

10.4 Skew and kurtosis

Skew is a measure of the asymmetry of a distribution about its mean. Skew can be zero, positive, or negative. Zero skew is found when the distribution is perfectly symmetrical. Positive skew is found when there is a long right tail to the distribution and the mass of the distribution is concentrated to the left. Negative skew is found when there is a long left tail to the distribution and the mass of the distribution is concentrated to the right. We can usually see skew in histograms. We can also calculate a skewness statistic and test if this is significantly different from zero.

Kurtosis is a measure of how much a distribution is concentrated about the mean. Kurtosis can be zero, positive, or negative. Zero kurtosis is found when a variable is normally distributed. Positive kurtosis is found when the mass of the distribution is concentrated about the mean and there are very few values far from the mean. Negative kurtosis is found when the mass of the distribution is concentrated in the tails of the distribution. We can usually see kurtosis in histograms. We can also calculate a kurtosis statistic and test if this is significantly different from zero.

The NIPN data quality toolkit provides an *R* language function called **skewKurt()** that calculates skewness and kurtosis statistics and tests whether they differ significantly from zero. Here we apply the **skewKurt()** function to the **muac** variable in the example dataset:

```
skewKurt(svy$muac)
```

This returns:

```
      Skewness and kurtosis

Skewness = +0.0525  SE = 0.0828  z = 0.6348   p = 0.5256
Kurtosis = -0.2412  SE = 0.1653  z = 1.4586   p = 0.1447
```

There is positive skew and negative kurtosis. Neither is significantly different from zero.

Applying the **skewKurt()** function to the **haz** variable in the example dataset:

```
skewKurt(svy$haz)
```

returns:

```
      Skewness and kurtosis

Skewness = +0.3074  SE = 0.0828  z = 3.7149   p = 0.0002
Kurtosis = +0.2074  SE = 0.1653  z = 1.2545   p = 0.2097
```

There is a positive skew and a positive kurtosis. The skew is significantly different from zero. The skew can be seen in the histogram:

```
histNormal(svy$haz, breaks = "scott")
```

Applying the **skewKurt()** function to the **waz** variable in the example dataset:

```
skewKurt(svy$waz)
```

returns:

```
      Skewness and kurtosis

Skewness = -0.0128  SE = 0.0828  z = 0.1541   p = 0.8775
Kurtosis = +0.1805  SE = 0.1653  z = 1.0919   p = 0.2749
```

There is negative skew and positive kurtosis. Neither is significantly different from zero.

Applying the **skewKurt()** function to the **whz** variable in the example dataset:

```
skewKurt(svy$whz)
```

returns:

```
      Skewness and kurtosis

Skewness = +0.0823  SE = 0.0828  z = 0.9946   p = 0.3199
Kurtosis = +0.7528  SE = 0.1653  z = 4.5530   p = 0.0000
```

There is a positive skew and a positive kurtosis. The kurtosis is significantly different from zero. The kurtosis can be seen on the histogram:

```
histNormal(svy$whz, breaks = "scott")
```

as the tall central columns that exceed the expected values shown by the overlaid normal distribution.

Skew and kurtosis are both used in SMART plausibility checks. *Table 10.2* shows how skew and kurtosis statistics are applied by SMART.

Table 10.2 : The range of absolute values of skewness and kurtosis statistics and they are applied by SMART (2015).

Absolute value* of skewness / kurtosis	**SMART interpretation**
< 0.2	Excellent
≥ 0.2 and < 0.4	Good
≥ 0.6 and < 0.6	Acceptable
≥ 0.6	Problematic

* This is the value of a number ignoring its negative or positive sign.
The absolute value of -0.2412 is 0.2412. The absolute value of +0.2412 is also 0.2412.

The **whz** variable in the example dataset is considered "problematic" according to this scheme because kurtosis is above 0.6.

Care should be exercised when using statistical significance tests to classify data as "problematic". The use of thresholds and ranges for skew and kurtosis statistics is usually a better approach than relying on tests of statistical significance. Significance tests can be strongly affected by sample size. Small sample sizes can lead to tests missing large effects and large sample sizes can lead to tests identifying small effects as highly significant. If a distribution appears to be normal (i.e. has a symmetrical, or nearly symmetrical, "bell-shaped" distribution) then it is usually safe to assume normality and to use statistical procedures that assume normality.

It is important to remember that the normal distribution is a mathematical abstraction. There is nothing to compel the real world to conform to the normal distribution. The normal distribution has become reified:

> *Everyone is sure of this [the normal distribution] … experimentalists believe that it is a mathematical theorem, and the mathematicians that it is an experimentally determined fact.*
>
> Henri Poincaré, *Calcul des Probabilités* (1912)

The data we see may be representative of reality even when it fails a test for normality.

Tests for normality are useful when selecting statistical methods that rely on normality. They are less useful for determining data quality. If data follows a symmetrical, or nearly symmetrical, "bell-shaped" distribution then it will usually safe to use.

10.5 Deviation from normality

Some anthropometric survey methods (e.g. SMART) use deviations from perfect normality as an indicator of poor data quality. But deviations from normality are not necessarily due to poor quality data; they can be due to sampling a mixed population. This is easy to demonstrate with some simulated data.

We will assume that we have a population consisting of two groups:

Group 1 : 75% of the population, mean = -0.48, sd = 0.87

Group 2 : 25% of the population, mean = -1.04, sd = 1.10

and that we take a sample size = 1000 from the whole population.

We can simulate this:

```
set.seed(0)
g1 <- rnorm(n = 750, mean = -0.48, sd = 0.87)
g2 <- rnorm(n = 250, mean = -1.04, sd = 1.11)
g1g2 <- c(g1, g2)
```

The distributions in the two subgroups (**g1** and **g2**) are both normally distributed:

```
histNormal(g1)
qqNormalPlot(g1)
shapiro.test(g1)
skewKurt(g1)
histNormal(g2)
qqNormalPlot(g2)
shapiro.test(g2)
skewKurt(g2)
```

but the distribution in the entire sample (**g1g2**) is not normal:

```
histNormal(g1g2)
qqNormalPlot(g1g2)
shapiro.test(g1g2)
skewKurt(g1g2)
```

The *Shapiro-Wilk test of normality* returns:

```
        Shapiro-Wilk normality test

data:  g1g2
W = 0.99671, p-value = 0.03514
```

There is statistically significant negative skew:

```
        Skewness and kurtosis

Skewness = -0.1767  SE = 0.0773  z = 2.2851   p = 0.0223
Kurtosis = +0.2894  SE = 0.1545  z = 1.8728   p = 0.0611
```

There is, however, nothing wrong with the sample or with the data

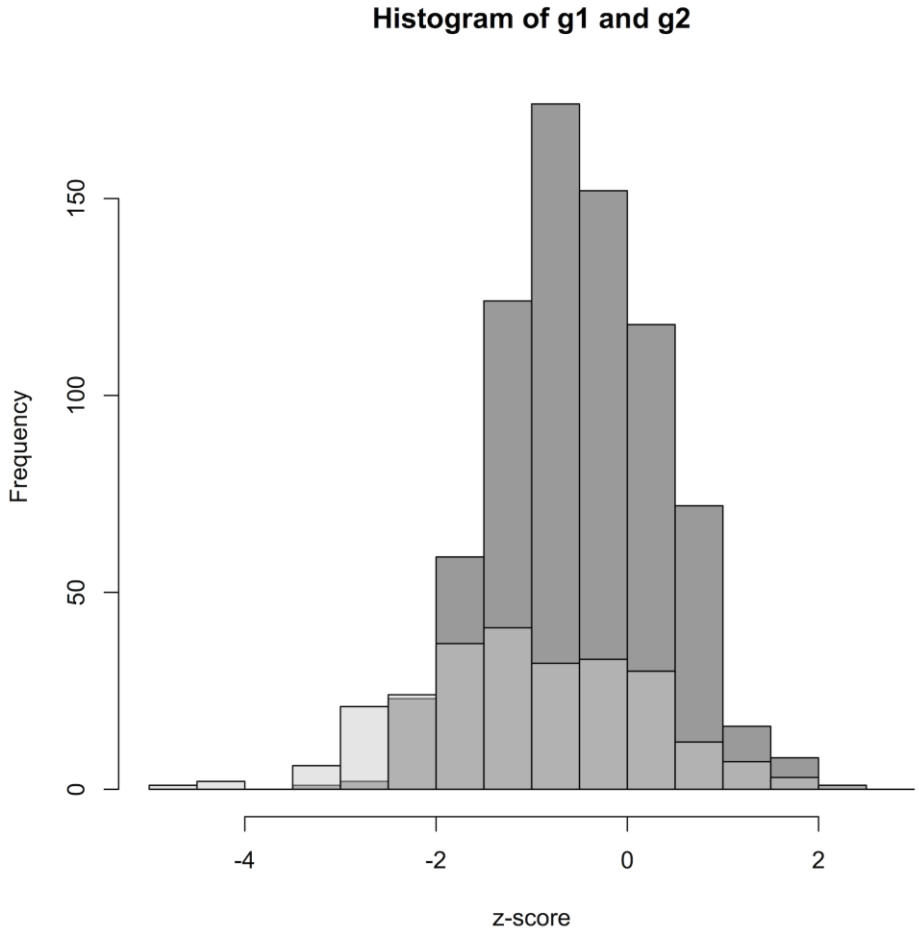
The distribution in the entire sample (**g1g2**) is called a "mixture of Gaussians" (the term "Gaussian" refers to the normal distribution in this context).

We can see this mixture of Gaussians with:

```
hist(g1, col=rgb(0.2, 0.2, 0.2, 0.5),
       breaks = seq(-5, 3, 0.5), xlab = "", main = "")
hist(g2, col=rgb(0.8, 0.8, 0.8, 0.5), breaks = seq(-5, 3, 0.5), add = TRUE)
title(main = "Histogram of g1 and g2", xlab = "z-score")
```

See *Figure 10.7*. In this case the mixture was already known. There are a number of methods for revealing the underlying mixture when the components of the mixture are unknown. These techniques are not covered in this toolkit.

Figure 10.7 : Example mixture of two normal distributions yielding a non-normal distribution. Dark grey is used to represent **g1**, light grey is used to represent **g2**, and middle grey is used to represent the overlap of **g1** and **g2**.



We will continue with an example in which the components of the mixture are suspected.

We will retrieve a survey dataset:

```
svy <- read.table("dist.ex02.csv", header = TRUE, sep = ",")
head(svy)
```

Twelve communities from a district of Somalia were selected for nutritional anthropometry assessment using a stratified random selection process.

Four communities were selected at random from each of the three rural livelihood zones (i.e. agro-pastoral, pastoral, and riverine agrarian) in the rural areas of the district:

```
table(svy$zone)
```

The **zone** column is coded:

1 = Agro-pastoral

2 = Pastoral

3 = Riverine agrarian

Data were collected by three teams assisted by local guides provided by community leaders. Sampling proceeded on a door-to-door (census) basis and data were collected for all eligible children present in the community on the day of the survey. Eligible children were defined as being between sixty five centimetres in length and one hundred and ten centimetres in height.

We will examine the distribution of weight-for-height z-scores:

```
histNormal(svy$whz)
```

The distribution of **whz** looks reasonably normal with a slight positive skew:

```
qqNormalPlot(svy$whz)
skewKurt(svy$whz)
```

The *Shapiro-Wilk test of normality*:

```
shapiro.test(svy$whz)
```

returns:

```
        Shapiro-Wilk normality test
data:  svy$whz
W = 0.99621, p-value = 0.03094
```

This suggests that there is some deviation from normality.

We may suspect that there is a mixture of Gaussians (i.e. from the different livelihood zones). We can examine this by looking at the distribution of **whz** in each of the three populations:

```
by(svy$whz, svy$zone, summary)
```

This returns:

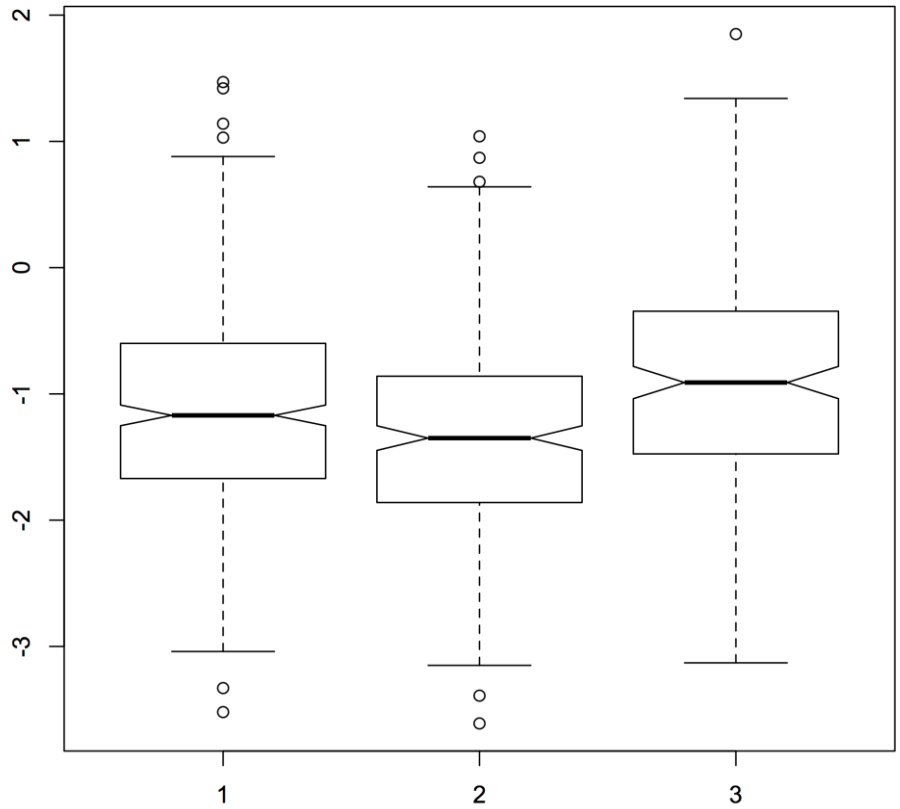
```
svy$zone: 1
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 -3.520  -1.667  -1.170  -1.142  -0.605   1.470
------------------------------------------------
svy$zone: 2
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 -3.610  -1.860  -1.350  -1.319  -0.860   1.040
------------------------------------------------
svy$zone: 3
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-3.1300 -1.4750 -0.9100 -0.8532 -0.3450  1.8500
```

An equivalent graphical analysis is:

```
boxplot(svy$whz ~ svy$zone, notch = TRUE)
```

This plot is shown in *Figure 10.8*. The notches around the medians for each box in the box plot represent approximate 95% confidence intervals for the medians. If the notches do not overlap then there is strong evidence that the medians of the distributions differ from each other. This looks to be the case in Figure *10.8*.

Figure 10.8 Distribution of weight-for-height z-scores in children measured in three rural livelihood zones



Livelihood zone is coded **1** = Agro-pastoral, **2** = Pastoral, **3** = Riverine agrarian.

The boxes in each plot extend between the upper and lower quartiles with the thick line in the box marking the position of the median. The whiskers extend to 1.5 times the interquartile distance above and below the upper and lower quartiles, and the isolated points mark data points outside of the range of values covered by the whiskers. The notches around the medians for each box represent approximate 95% confidence intervals for the medians. If the notches do not overlap then there is strong evidence that the medians of the distributions differ from each other.

We can examine the three **whz** distributions for normality:

```
by(svy$whz, svy$zone, skewKurt)
by(svy$whz, svy$zone, shapiro.test)
```

Each of the three **whz** distributions appear to be normally distributed.

Since we have three normal distributions we can test for differences using *one-way analysis of variance* (ANOVA):

```
aovTest <- aov(svy$whz ~ as.factor(svy$zone))
summary(aovTest)
```

This returns:

```
                        Df Sum Sq Mean Sq F value Pr(>F)
as.factor(svy$zone)   2   24.3  12.164   18.09  2e-08 ***
Residuals           880  591.7   0.672
```

There are statistically significant differences between the means of the three distributions.

We can further examine the differences using the **TukeyHSD()** function:

```
TukeyHSD(aovTest)
```

This returns:

```
           diff        lwr         upr     p adj
2-1 -0.1763167 -0.3274578 -0.02517566 0.0172882
3-1  0.2891166  0.1226730  0.45556035 0.0001465
3-2  0.4654334  0.2833653  0.64750143 0.0000000
```

The first item on each row of the output refers to the comparison being made. The row labelled **2-1** , for example, tells us that the results presented on that row relate to the mean **whz** in livelihood zones **1** and **2** (i.e. the agro-pastoral and pastoral livelihood zones). The difference between the two means is labelled **diff** and lower (**lwr**) and upper (**upr**) 95% confidence limits for the difference between the two means (**diff**) are given. The observed significance (**p adj**) is adjusted for multiple comparisons.

All three **whz** distributions are differences from each other. We can see this graphically using:

```
plot(TukeyHSD(aovTest))
```

In this example survey we found a mixture of Gaussians. This was suspected prior to data being collected and was the reason for the stratified sample design. You may be able to perform a similar analysis with SMART type surveys if, for example, you are able to assign primary sampling units (PSUs) to livelihood zones. Such an analysis may have limited power if SMART flagging criteria have been applied and flagged records have been removed. Analyses by wealth group, for example, will require the relevant data to be collected and recorded in the survey dataset and may be possible with MICS and DHS data.

We expect to see small deviations from normality in most survey datasets. This will often be the case when a survey takes a sample of subjects over a wide area covering, for example, several agro-ecological zones, socio-economic groups, or ethnic groups. This will almost always be the case, particularly with large surveys such as DHS, MICS, and national SMART surveys.

Another reason for non-normality is that one (or more) of the survey teams has a systematic bias in making a measurement. Identifying the “offending” survey team by examining and testing for

normality separately in all combinations of data from *n – 1* survey teams can be attempted. If (e.g.) there were three teams then we would need to separately test data from:

Team 1 and Team 2 (Team 3 excluded)

Team 1 and Team 3 (Team 2 excluded)

Team 2 and Team 3 (Team 1 excluded)

to see if the deviation from normality disappears when a particular team's data are excluded. There is, however, a problem with this type of analysis. In cluster-sampled surveys, teams often sample adjacent primary sampling units (clusters). When this occurs the "exclude one team" analysis cannot distinguish between differences due to spatial heterogeneity (i.e. patchiness) and differences due to a team having a systematic measurement bias.

10.6 The standard deviation and alternatives

The standard deviation is sometimes considered to be useful measure of data quality when applied to z-scores.

We will use the **`sd()`** function to calculate the standard deviation of **`whz`** in the Kabul data:

```
svy <- read.table("dist.ex01.csv", header = TRUE, sep = ",")
sd(svy$whz)
```

This returns:

```
1.323469
```

This may produce misleading values if applied to raw data. This procedure should only be applied to cleaned data from which erroneous data and flagged records have been censored.

SMART data quality guidelines (SMART, 2015) state that the acceptable range for the standard deviation of the weight-for-height z-scores (**`whz`**) is 0.8 to 1.2 when SMART flagging criteria have been applied and flagged records have been censored. Standard deviations outside this range are considered to indicate poor survey quality. Note that SMART does not define thresholds for anthropometric indices other than for weight-for-height z-scores. It is important to note that a standard deviation above 1.2 may be due to sampling from a mixed population rather than due to poor data quality.

The **`flag`** column in the example dataset contains a flagging code in which the codes **2**, **3**, **6**, or **7** indicate potential problems with weight and / or height. We should calculate the standard deviation of the **`whz`** variable using only the data in which records with these flagging codes have been censored and there is no oedema recorded:

```
sd(svy$whz[!(svy$flag %in% c(2, 3, 6, 7) | svy$oedema == 1)])
```

The **`!`** character specifies a logical "not". The standard deviation is, therefore, calculated using records in which the **`flag`** variable does **not** contain **2**, **3**, **6**, or **7** and oedema is **not** recorded as being present.

The standard deviation for **`whz`** when flagged records and oedema cases are censored is:

```
1.141944
```

This is within the SMART acceptable range of 0.8 to 1.2.

The problem with using the standard deviation with raw data is that it is a non-robust statistic. This means that it can be strongly influenced by outliers. For example:

```
sd(c(4.55, 5.93, 2.68, 5.61, 3.53, 4.78, 3.60, 5.82, 4.41, 5.42))
```

returns:

```
1.097533
```

Adding a single outlier (e.g. data entered as **7.84** rather than as **4.78**):

```
sd(c(4.55, 5.93, 2.68, 5.61, 3.53, 7.84, 3.60, 5.82, 4.41, 5.42))
```

This returns:

```
1.496963
```

In this example a single outlier has strongly influenced the standard deviation.

There are a number of robust estimators for the standard deviation. *R* provides the **mad()** function to calculate an adjusted *median absolute deviation* (MAD).

The median absolute deviation (MAD) is defined as the median of the absolute deviations from the median. It is the median of the absolute values of the differences between the individual data points and the median of the data:

$$MAD = median(|x_i - median(x)|)$$

The calculated MAD is adjusted to make it consistent with the standard deviation:

$$\hat{\sigma} = k \cdot MAD$$

where *k* is a constant scaling factor, which depends upon the distribution. For the normal distribution:

$$k = 1.4826$$

The **mad()** function in *R* returns the adjusted MAD:

$$\hat{\sigma} = 1.4826 \cdot MAD$$

This is a robust estimate of the standard deviation.

This estimator is preferred when a sample is taken from a mixed population (this is almost always the case) and when the distribution has "fat" or "heavy" tails, as is the case with the **whz** variable in the example dataset.

Using the **mad()** function with the raw WHZ data:

```
mad(svy$whz)
```

This returns:

```
1.156428
```

We would usually want to calculate the adjusted MAD of the **whz** variable using only the data in which records with flagging codes relevant to **whz** and cases of oedema are censored:

```
mad(svy$whz[!(svy$flag %in% c(2, 3, 6, 7) | svy$oedema == 1)])
```

This returns:

```
1.097124
```

The use of the standard deviation and robust equivalents such as the adjusted MAD with simple thresholds is problematic. Data that is a mixture of Gaussians distributions will tend to have large standard deviations even when there is no systematic error and nothing is wrong with the sample. Checks on the standard deviation in large surveys should, therefore, be performed on the smallest spatial strata above the PSU or cluster level. This reduces but does not eliminate the problem of sampling from mixed populations.

We will retrieve a dataset and examine within-strata MADs:

```
svy <- read.table("flag.ex03.csv", header = TRUE, sep = ",")
head(svy)
```

The file `flag.ex03.csv` is a comma-separated-value (CSV) file containing anthropometric data from a national SMART survey in Nigeria.

The data stored in the file `flag.ex03.csv` were collected using methods similar to MICS and DHS surveys. The only difference is that the survey collected anthropometric data on children aged between 6 and 59 months. In this exercise we will concentrate on WHZ.

Data are stratified by **region** and by **state** within **region**. We will create a new variable that combines **region** and **state**:

```
svy$regionState <- paste(svy$region, svy$state, sep = ":")
head(svy)
table(svy$regionState)
```

We can examine the adjusted MAD for **whz** for each combination of **region** and **state** in the survey dataset using:

```
by(svy$whz, svy$regionState, mad, na.rm = TRUE)
```

The long output can be made more compact, easier to read, and easier to work with:

```
mads <- by(svy$whz, svy$regionState, mad, na.rm = TRUE)
mads <- round(mads[1:length(mads)], 2)
mads
```

The saved **mads** object can be summarised:

```
summary(mads)
```

This returns:

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.8400  0.9300  0.9800  0.9892  1.0300  1.2000
```

A table can also be useful:

```
table(mads)
```

In this example the adjusted MAD of the **whz** variable is within the limits 0.8 to 1.2 for all combinations of **region** and **state**.

Note that we combined **region** and **state**. We did this to avoid potential problems with duplicate **state** names (i.e. the same **state** name used in more than one **region**).

In the previous exercise we used the raw data (i.e. without flagging). It is better to use only the data in which records with flagging codes relevant to **whz** and cases of oedema are censored.

This is a national SMART survey so we will use SMART flagging criteria. We will use the **national.SMART()** function to add SMART flags to the survey dataset:

```
svyFlagged <- national.SMART(x = svy, strata = "regionState")
```

We need to exclude records with flagging codes relevant to **whz**:

```
svyFlagged <- svyFlagged[!(svyFlagged$flagSMART %in% c(2, 3, 6, 7)), ]
```

Note that oedema is not recorded in the dataset so we cannot exclude oedema cases.

We can now calculate the MAD for **whz** in each stratum:

```
mads <- by(svyFlagged$whz, svyFlagged$regionState, mad, na.rm = TRUE)
mads <- round(mads[1:length(mads)], 2)
mads
```

The saved **mads** object can be summarised:

```
summary(mads)
```

This returns:

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.8500  0.9000  0.9600  0.9665  1.0100  1.1700
```

In this analysis the adjusted MAD of the **whz** variable is within the limits 0.8 to 1.2 for all combinations of **region** and **state**.

10.7 Measures of dispersion

Measures of dispersion summarise how cases (e.g. children classified as wasted, stunted, or underweight) are distributed across a survey's primary sampling units (e.g. clusters).

We will retrieve a survey dataset:

```
svy <- read.table("flag.ex01.csv", header = TRUE, sep = ",")
head(svy)
```

The file **flag.ex01.csv** is a comma-separated-value (CSV) file containing anthropometric data from a recent SMART survey in Sudan.
We will apply WHO flagging criteria to the data:

```
svy$flag <- 0
svy$flag <- ifelse(!is.na(svy$haz) & (svy$haz < -6 | svy$haz > 6),
      svy$flag + 1, svy$flag)
svy$flag <- ifelse(!is.na(svy$whz) & (svy$whz < -5 | svy$whz > 5),
      svy$flag + 2, svy$flag)
svy$flag <- ifelse(!is.na(svy$waz) & (svy$waz < -6 | svy$waz > 5),
      svy$flag + 4, svy$flag)
```

We should exclude flagged records:

```
svy <- svy[svy$flag == 0, ]
```

We will apply a case-definition for being stunted:

```
svy$stunted <- ifelse(svy$haz < -2, 1, 2)
```

We can examine the distribution of stunted cases across the primary sampling units in this survey:

```
table(svy$psu, svy$stunted)
```

We only need the counts of cases in each primary sampling unit:

```
table(svy$psu, svy$stunted)[,1]
barplot(table(svy$psu, svy$stunted)[,1], xlab = "PSU", ylab = "Cases",
      cex.names = 0.5)
```

It will be useful to keep this for later use:

```
casesPerPSU <- table(svy$psu, svy$stunted)[,1]
casesPerPSU
```

We are interested in how cases are distributed across the primary sampling units.

There are three general patterns. These are *random*, *clumped*, and *uniform*.

We can identify the pattern to which the example data most likely belongs using an *index of dispersion*.

The simplest index of dispersion, and the one used by SMART (2015), is the *variance to mean ratio*:

$$Variance\ to\ mean\ ratio = \frac{s^2}{\bar{x}}$$

The interpretation of the variance to mean ratio is straightforward:

Variance to mean ratio ≈ 1	Random
Variance to mean ratio > 1	Clumped (i.e. more clumped than random)
Variance to mean ratio < 1	Uniform (i.e. more uniform than random)

The value of the variance to mean ratio can range between zero (maximum uniformity) and the total number of cases in the data (maximum clumping). Maximum uniformity is found when the same number of cases are found in every primary sampling unit. Maximum clumping is found when all cases are found in one primary sampling unit.

With the example data:

```
varianceCasesPerPSU <- var(casesPerPSU)
meanCasesPerPSU <- sum(casesPerPSU) / length(casesPerPSU)
V2M <- varianceCasesPerPSU / meanCasesPerPSU
V2M
```

The observed variance to mean ratio (**0.6393127**) suggests that the distribution of cases across primary sampling units is not completely uniform, but neither is it random.

A formal (*Chi-squared*) test can be performed. The Chi-squared test statistic can be calculated using:

```
sum((casesPerPSU - meanCasesPerPSU)^2) / meanCasesPerPSU
```

This returns:

```
18.54007
```

The critical values for this test statistic can be found using:

```
qchisq(p = c(0.025, 0.975), df = length(casesPerPSU) - 1)
```

This returns:

```
16.04707 45.72229
```

If the Chi-squared test statistic was below **16.04707** then we would conclude that the pattern of cases across primary sampling units in the example data is uniform. This is not the case in the example data.

If the Chi-squared test statistic was above **45.72229** then we would conclude that the pattern of cases across primary sampling units in the example data is clumped. This is not the case in the example data.

Since the Chi-squared test statistic falls between **16.04707** and **45.72229** we conclude that the pattern of cases across primary sampling units in the example data is random.

There are problems with the variance to mean ratio. Some clearly non-random patterns can produce variance to mean ratios of one. The variance to mean ratio is also strongly influenced by the total number of cases present in the data when clumping is present.

A better measure is *Green's Index of Dispersion*:

$$Green's\ Index = \frac{(s^2/\bar{x}) - 1}{n - 1}$$

Green's Index corrects the variance to mean ratio for the total number of cases present in the data.

The value of Green's Index can range between $-1/(n-1)$ for maximum uniformity (specific to the dataset) and one for maximum clumping. The interpretation of Green's Index is straightforward:

Green's Index ≈ 0	Random
Green's Index > 0	Clumped (i.e. more clumped than random)
Green's Index < 0	Uniform (i.e. more uniform than random)

The sampling distribution of Green's Index is not well described. The NIPN data quality toolkit provides the **`greenIndex()`** function that overcomes this problem. This *R* language function uses the bootstrap technique to estimate Green's Index and test whether the distribution of cases across primary sampling units is random.

The **`greenIndex()`** function requires you to specify the name of the survey dataset, the name of the variable specifying the primary sampling unit, and the name of the variable specifying case status. With the example data:

```
greensIndex(data = svy, psu = "psu", case = "stunted")
```

this returns:

```
      Green's Index of Dispersion

Green's Index (GI) of Dispersion  = -0.0013, 95% CI = (-0.0021, -0.0004)
Maximum uniformity for this data  = -0.0035
                        p-value   =  0.0040
```

The point estimate of Green's Index (**`-0.0013`**) is below zero and the p-value of the test for a random distribution of cases across primary sampling units (**`0.0040`**) is below 0.05. The distribution of cases across primary sampling units in the example data is significantly more uniform than it is random. We can see this graphically using:

```
table(svy$psu, svy$stunted)[,1]

barplot(table(svy$psu, svy$stunted)[,1], xlab = "PSU", ylab = "Cases",
      cex.names = 0.5)

abline(h = sum(casesPerPSU) / length(casesPerPSU), lty = 2)
```

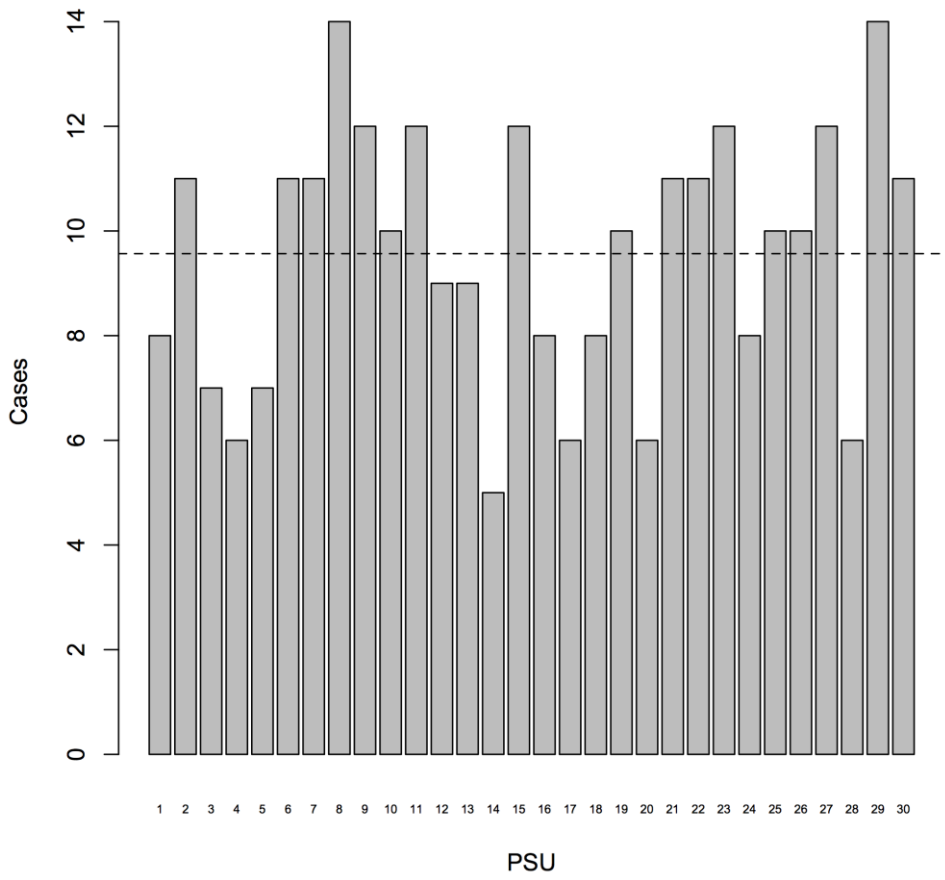
The dashed line on the plot marks the mean number of cases found in each primary sampling unit. A uniform distribution would show all bars ending close to this line (see *Figure 10.9*).

SMART uses the variance to mean ratio as a test of data quality. Green's Index is a more robust choice because it can be used to compare samples that vary in overall sample size and the number of sampling units used.

The idea behind using a measure of dispersion to judge data quality is a belief that the distribution of cases of malnutrition across primary sampling units should always be random. If this is not the case then the data are considered to be suspect. The problem with this approach is that deviations from random can reflect the true distribution of cases in the survey area. This may occur when the survey area comprises, for example, more than one livelihood zone. It is also less likely to be the case for conditions, such as wasting and oedema, that are associated with infectious disease and so may be more clumped than randomly distributed across primary sampling units. This may become a particular problem when proximity sampling is used to collect the within-cluster samples.

Measures of dispersion are problematic when used as measures of data quality and should be interpreted with caution. The exception to this rule is finding maximum, or almost maximum, uniformity or maximum, or almost maximum, clumping. A finding of maximum uniformity is likely only when data have been fabricated. A finding of maximum clumping may indicate poor data collection and / or poor data management.

Figure 10.9 : Distribution of cases of stunting across primary sampling units in the example dataset. The dashed line on the plot marks the mean number of cases found in each primary sampling unit. A uniform distribution would show all bars ending close to this line.



11. Mean, standard deviation, prevalence, and the PROBIT estimator

This section supports the data quality toolkit by providing tools to calculate the prevalence from the mean and standard deviation of a normally distributed variable using the PROBIT estimator, and to calculate the standard deviation when only the mean and prevalence are provided in a survey report.

11.1 The relationship between mean z-score and the prevalence of indicators

For many anthropometric indicators the raw measurements and derived z-scores are often normally distributed. When the distribution is not normal it can usually be transformed towards normality.

We will start with some simulated data:

```
set.seed(0)
zScore <- rnorm(1000, mean = -0.55, sd = 1.10)
```

This generates 1,000 values that are normally distributed around a population mean (i.e. **-0.55**) with a population standard deviation (i.e. **1.10**) and stores them in the **zScore** object. We can treat **zScore** as sample of size *n* = 1,000 taken from a much larger population that has a true mean of **-0.55** and a true standard deviation of **1.10**.

We can examine and summarise the generated data:

```
zScore
mean(zScore)
sd(zScore)
hist(zScore)
```

The observed mean (**-0.5674125**) is different from the population mean (**-0.55**) and the observed standard deviation (**1.097805**) is different from the population standard deviation (**1.10**). This is because **zScore** is a sample from a population with a mean of **-0.55** and a standard deviation of **1.10**.

We can apply a case-definition (1 for cases, 2 for not-case) using:

```
case <- ifelse(zScore < -2, 1, 2)
```

and estimate the prevalence:

```
prop.table(table(case))
```

This gives:

```
case
    1     2
0.094 0.906
```

The estimated prevalence is 9.40%. This is the *classical* or *frequentist* method for estimating prevalence: apply a case-definition and examine the frequency of cases as a proportion of the total sample.

An alternative approach is to use the *PROBIT* method. The *PROBIT* method uses the cumulative normal probability density function to estimate the prevalence for a given case-defining threshold using the sample mean and sample standard deviation:

```
pnorm(-2, mean = mean(zScore), sd = sd(zScore))
```

This returns:

```
0.09595394
```

The estimated prevalence is 9.60%.

The exact prevalence can also be calculated in the same way using the true mean and true standard deviation of the population:

```
pnorm(-2, mean = -0.55, sd = 1.10)
```

This returns:

```
0.0937214
```

The true prevalence is 9.37%.

The classical and the PROBIT approaches yield similar estimates of prevalence and both are close to the true prevalence (i.e. the prevalence in the population)

A key advantage of the *PROBIT* approach is that estimates are more precise (i.e. standard errors are smaller and confidence intervals narrower) than those obtained using the classical approach with a given sample size. This is because the PROBIT approach uses more information from the sample than the classical approach, which loses information when a wide range of values is reduced to a set of binary values (i.e. case / not case).

The key disadvantage of the *PROBIT* approach is that it relies on the variable being normally distributed. If this is not the case, then *PROBIT* estimates may be inaccurate.

Data that we suspect is not normally distributed can usually be transformed towards normality before using the *PROBIT* approach to estimate prevalence.

The main point to note here is that the mean and standard deviation of normally distributed indicators can be used to estimate the prevalence.

Another advantage of the *PROBIT* method is that it works when errors in data caused by rounding and digit preference cause the classical method to produce inaccurate estimates. Here we simulate some MUAC data:

```
set.seed(0)
muac <- round(rnorm(1000, mean = 140, sd = 11))
mean(muac)
sd(muac)
hist(muac)
```

The true prevalence for MUAC < 125 mm is:

```
pnorm(125, mean = 140, sd = 11)
```

This returns:

```
0.08634102
```

The true prevalence is 8.63%.

The classical prevalence estimate is:

```
case <- ifelse(muac < 125, 1, 2)
prop.table(table(case))
```

This returns:

```
case
    1     2
0.086 0.914
```

The estimated prevalence is 8.60%.

The PROBIT prevalence estimate is:

```
pnorm(125, mean = mean(muac) , sd = sd(muac))
```

This returns:

```
0.08892795
```

The estimate prevalence is 8.89%.

The classical and the PROBIT approaches yield similar prevalence estimates and both are close to the true prevalence.

We will now subject the data to rounding to simulate very strong digit preference:

```
muac <- round(muac / 5) * 5
table(muac)
digitPreference(muac, digits = 0)
```

The resulting digit preference score (i.e. DPS = 66.68) would render this data very suspect. We can see the digit preference using:

```
plot(digitPreference(muac, digits = 0))
```

or using the **`hist()`** function and specifying a large number for the **`breaks`** parameter:

```
hist(muac, breaks = 1000)
```

Estimating the prevalence from the rounded data using the classical approach:

```
case <- ifelse(muac < 125, 1, 2)
```

```
prop.table(table(case))
```

gives:

```
case
    1     2
0.051 0.949
```

This (i.e. 5.10%) is very different from the true prevalence (i.e. 8.63%).

Estimating prevalence from the rounded data using the *PROBIT* approach:

```
pnorm(125, mean = mean(muac) , sd = sd(muac))
```

yields:

```
0.08871178
```

This (i.e. 8.87%) is still close to the true prevalence.

These results are summarised in *Table 11..1*.

Table 11.1. Classical and PROBIT estimates of prevalence made using clean or rounded data

	Estimated prevalence							
	Clean data				**Rounded data**			
	Classical approach		**PROBIT approach**		**Classical approach**		**PROBIT approach**	
True prevalence	**Estimate**	**Error***	**Estimate**	**Error***	**Estimate**	**Error***	**Estimate**	**Error***
8.63%	8.60%	-0.03%	8.89%	+0.26%	5.10%	-3.53%	8.87%	+0.24%

* Error is calculated as: *estimated prevalence – true prevalence.*
Positive errors indicate overestimates. Negative errors indicate underestimates.

The PROBIT approach is not greatly affected by rounding and strong digit preference, giving only a small positive error when used with the rounded data. The classical approach gave a large negative error when used with the rounded data.

11.2 Mean, prevalence, and the standard deviation

The relationship between the prevalence and the distribution of a variable (defined by its mean and standard deviation) allows us to estimate the standard deviation from the mean and the prevalence. This can be useful if we want to find the standard deviation when only the mean and prevalence are reported, such as in some MICS and DHS survey reports.

We can use the cumulative probability density function of the normal distribution to help find the standard deviation for a given mean and prevalence. Here we will work with:

Observed mean = -0.67

Observed prevalence = 11.48%

The task is to find the value of the standard deviation (*SD*) in the equation:

$$observed\ prevalence - pnorm(value = -2, mean = -0.67, SD)$$

which gives an answer very close to zero.

We can try doing this using *R* and a directed trial and error approach. We start by guessing the value of the *SD*. A good first guess for *SD* is 1.00:

```
0.1148 - pnorm(-2, -0.67, 1.00)
```

This gives:

```
0.02304086
```

Note that we specify prevalence as a proportion.

The difference from the target value of zero is positive so we need to increase our guess for the *SD*:

```
0.1148 - pnorm(-2, -0.67, 1.20)
```

This gives:

```
-0.01905894
```

The difference from the target value of zero is negative so we need to decrease our guess for the *SD*:

```
0.1148 - pnorm(-2, -0.67, 1.10)
```

this gives:

```
0.001486039
```

The difference from the target value of zero is positive and small so we should to slightly increase our guess for the *SD*:

```
0.1148 - pnorm(-2, -0.67, 1.11)
```

This gives:

```
-0.0006199333
```

The difference from the target value of zero is now very small.

We would probably stop our search with estimated *SD* = **1.11** since it could only be improved by adding more decimal places, which we would be unlikely to report.

The search for the best estimate of *SD* can be automated. To do this we must first write an *objective function* that specifies the function that we want to be minimised:

```
objFun <- function(SD) abs(0.1148 - pnorm(-2, -0.67, SD))
```

The **abs()** function strips any sign from the results. We do this because we want to find the value of **SD** that yields the minimum possible difference from zero ignoring the direction (i.e. the sign) of the difference.

We can now use the **optimise()** function to find the value of **SD** that yields the difference closest to zero:

```
optimise(objFun, interval = c(0.5, 2.0))
```

The **interval** parameter tells the **optimise()** function to search between **0.5** and **2.0** for the value of **SD** that minimises the value returned by the objective function.

The **optimise()** function returns:

```
$minimum
[1] 1.107052

$objective
[1] 1.400078e-07
```

The item labelled **$minimum** is the value of **SD** that minimises the objective function and is our best estimate of the standard deviation. The item labelled **$objective** is the difference (very close to zero) between the observed prevalence and the prevalence expected when the mean is **-0.67** and the SD is **1.107052**.

The six decimal places given for **SD** are excessive given that our inputs are estimated to only two decimal places and the underlying distribution of the indicator of interest is unlikely to be perfectly normal. It is better to round the results to two decimal places:

```
estimatedSD <- optimise(objFun, interval = c(0.5, 2.0))
estimatedSD
round(estimatedSD$minimum, 2)
```

This returns:

```
1.11
```

We can use a very similar process if we have raw indicator values such as the mean and standard deviation of MUAC measurements. With:

Observed mean = 139.82

Observed prevalence = 8.6%

We would use:

```
objFun <- function(SD) abs(0.086 - pnorm(125, 139.82, SD))
optimise(objFun, interval = c(5, 20))
```

Note that we define the objective function using the observed prevalence specified as a proportion (**0.086**), the observed mean (**139.82**), and the appropriate case-defining threshold (i.e. **125** mm).

The **optimise()** function returns:

```
$minimum
[1] 10.85074

$objective
[1] 8.186181e-08
```

11.3 Diagnosing `optimise()`

The **`interval`** parameter of the **`optimise()`** function specifies the range of values to be searched when seeking the *minimum*. We need to specify an appropriate range of values for the **`interval`** parameter. The **`optimise()`** function can yield misleading results if we specify a range of values for the **`interval`** parameter that does not include the true minimum. It is easy to detect and fix this problem: if the reported **`$minimum`** is very close to either of the **`interval`** values then you should suspect a problem and try again with a wider range. For example:

```
optimise(objFun, interval = c(5, 8))
```

yields:

```
$minimum
[1] 7.999924

$objective
[1] 0.05402423
```

The reported **`$minimum`** is **`7.999924`**. This is very close to the upper value (i.e. **`8`**) of the specified **`interval`**. We suspect, therefore, that our specified **`interval`** is too narrow and so try again with a wider **`interval`**:

```
optimise(objFun, interval = c(5, 20))
```

which yields:

```
$minimum
[1] 10.85074

$objective
[1] 8.186181e-08
```

The reported **`$minimum`** is **`10.85074`**. This is not close to either the lower or upper value of the specified **`interval`**. We can, therefore, accept the reported **`$minimum`** as a good estimate of the standard deviation.

12. Assessing data quality

This section presents an approach to assessing data quality. A set of tests is used to assess whether data are of good quality. Each test has a *criterion* and a *standard*

The *criterion* describes what is being tested and always refers to an attribute of the data being tested. Examples of criteria are "Outliers", "Illegal values", and "Age-heaping". Criteria may also refer to a specific variable as in "Digit preference (height)".

The *standard* refers to how good quality data is defined. The standard may be either *wholly quantitative* or *semi-quantitative*:

> An example of a *wholly quantitative* standard is the maximum proportion of outliers accepted in a variable.
>
> An example of a *semi-quantitative standard* is an analysis of data using histograms, normal-normal quantile plots, tests of normality, and summary statistics such as skewness and kurtosis.

Data are assessed for their quality by applying each and every appropriate test to the data according to the order suggested in Figure 1.1, using the tools provided. The criteria to judge that data are of good quality are presented in *Table 12.1* with the classification used in SMART plausibility checks for the same or similar criterion. In general for the criteria that are the same the NIPN data quality toolkit judges data to be of good quality if it is classified as good or excellent quality in SMART plausibility checks.

The main differences from SMART plausibility checks are that the NIPN toolkit provides a classification of illegal values, outliers and age-heaping; it examines the deviation, skewness and kurtosis of all z-scores, not just weight-for-height; and it applies a different index of the dispersion of cases between clusters, using Green's index rather than the variance to mean ratio. The SMART plausibility checks also offer a grade for each individual score and a weighted grade for the total score, which may be found elsewhere (SMART, 2015).

Failing a test does not mean that the data that failed the test are of no use. It means that we should treat the failing data with a degree of caution. The usual effect of this will be to place limits on how we use the data. If we find, for example, marked age-heaping in children's ages at whole and half-years then we would want to limit the analysis to year-centred age-groups or broader age-groups based upon year-centred age-groups. The use of age-groups such as 12 to 23 months would result in misleading analyses. This is because many children aged between 9 and 11 months are likely to be wrongly included and many children between 21 and 23 months are likely to be wrongly excluded. An analysis that is nominally for children age between 12 and 23 months might really be analysis of children aged between 9 and 20 months.

Failing a test may also mean that common analyses will need to be replaced by more robust analyses. Classical statistical tests might be replaced by their *non-parametric* equivalents (e.g. a *student's t-test* might be replace by a *Wilcoxon rank-sum test*). Classical summaries might be replaced by robust summaries (e.g. means by medians, standard deviations by median absolute deviations). Estimators may also need to be changed (e.g. PROBIT estimators may need to replace classical estimators for estimating prevalences of anthropometric deficits).

Failing a test is a warning to proceed with caution. It does not condemn the data as useless. We expect most datasets to fail one or more tests.

A simple overall or summary measure of data quality is the proportions of tests that were not good quality. Like many summary measures this approach will hide details and will usually not be very useful

Table 12.1. Criteria of data quality proposed for the NIPN toolkit and the criteria applied by ENA software plausibility checks. Graphical methods to assess data quality should also be used.

	NIPN data quality toolkit					ENA software plausibility checks (SMART, 2015)					
Criteria of data quality	Description and variables applied to	Section in toolkit	Toolkit functions	Unit	Good quality	Description and variables applied to	Unit	Classification of data quality: Excellent	Good	Acceptable	Problematic
Illegal values	Perform on each categorical variable	3		%	< 1%	No classification					
Sex ratio	Significant difference from expected sex ratio	4	`sexRatioTest()`	p-value	p ≥ 0.05	Significant difference from expected ratio	p-value	>0.1	>0.05	>0.001	≤0.001
Age structure	Significant difference from expected structure	5	`pyramid.plot()` `ageChildren()`	p-value	p ≥ 0.05	No classification					
Age ratio	Significant difference in no. of children 6-29 vs 30-59 months	5	`ageRatioTest()`	p-value	p ≥ 0.05	Significant difference in no. of children 6-29 vs 30-59 months	p-value	>0.1	>0.05	>0.001	≤0.001
Digit preference score (DPS)	Measurements of: weight, length or height, MUAC, and others	6	`fullTable()` `digitPreference()`	Number	DPS <12	Measurements of: weight, length or height, MUAC only	Number	0 – 7	8 – 12	13 – 20	> 20
Age heaping	Significant difference from expected numbers	7	`ageHeaping()`	p-value	p ≥ 0.05	No classification					
Outliers	Perform on each quantitative variable	8	`outliersMD()` `outliersUV()`	%	< 1%	No classification					
Flagged z-scores	WHZ, HAZ, WAZ, BAZ* and others out of range using **WHO criteria**	9		%	< 5%	WHZ, HAZ, WAZ out of range using **SMART criteria**	%	0 – 2.5	>2.5 – 5.0	>5.0 – 7.5	>7.5
Deviation of z-scores	WHZ, HAZ, WAZ, BAZ and other z-scores	10	`histNormal()` `qqNormalPlot()` `mad()`	Adj. median absolute deviation	≥0.8 & ≤1.2	WHZ	Standard deviation	<1.10 & >0.90	<1.15 & >0.85	<1.20 & >0.80	≥1.20 & ≤0.80
Skewness of z-scores	WHZ, HAZ, WAZ, BAZ and other z-scores	10	`skewKurt()`	Number	<± 0.4	WHZ	Number	<± 0.2	<± 0.4	<± 0.6	>± 0.6
Kurtosis of z-scores	WHZ, HAZ, WAZ, BAZ and other z-scores	10	`skewKurt()`	Number	<± 0.4	WHZ	Number	<± 0.2	<± 0.4	<± 0.6	>± 0.6
Index of dispersion of cases	Green's index	10	`greenIndex()`	Value	p ≥ 0.05	Variance to mean ratio of WHZ	p-value	>0.1	>0.05	>0.001	≤0.001

* WHZ = z-score of weight-for-height; HAZ = z-score of height-for-age; WAZ = z-score of weight-for-age; BAZ = z-score of BMI-for-age

Appendix Z

Z.1 Calculating anthropometric z-scores using the `addWGSR()` function

The NiPN anthropometry data quality toolkit provides an *R* language function **`addWGSR()`** that calculates a range of anthropometric z-scores and adds them to survey data:

- Weight-for-length (wfl) z-scores for children with lengths between 45 and 110 cm
- Weight-for-height (wfh) z-scores for children with heights between 65 and 120 cm
- Length-for-age (lfa) z-scores for children aged less than 24 months
- Height-for-age (hfa) z-scores for children aged between 24 and 228 months
- Weight-for-age (wfa) z-scores for children aged between zero and 120 months
- Body mass index-for-age z-scores (bfa) for children aged between zero and 228 months
- MUAC-for-age (mfa) z-scores for children aged between 3 and 60 months
- Triceps skinfold-for-age (tsa) z-scores for children aged between 3 and 60 months
- Sub-scapular skinfold-for-age (ssa) z-scores for children aged between 3 and 60 months
- Head circumference-for-age (hca) z-scores for children aged between zero and 60 months

Note that length is measured supine while height is measured standing.

The z-scores are calculated using the WHO Child Growth Standards (2006a, 2006b) for children aged between zero and 60 months or the WHO Growth References (2007) for school-aged children and adolescents from 5 to 19 years.

The **`addWGSR()`** function and supporting data are in the file **`addWGSR.r`**.

The data set to practice using the function is in the file **`z.ex01.csv`**.

You should use **`source()`** to load the **`addWGSR()`** function.

For example, if you place the **`addWGSR.r`** file in the directory:

```
~/Documents/Clients/NIPN/toolkit/
```

you would use:

```
source("~/Documents/Clients/NIPN/toolkit/addWGSR.r")
```

to load the file into *R*.

That is a UNIX path. If you are using Microsoft Windows™ and have placed the file **`addWGSR.r`** in the directory:

```
c:\dataquality\
```

then you would use:

```
source("c:/dataquality/addWGSR.r")
```

to load the file into *R*.

Note that *R* uses the forward slash (**/**) rather than the backslash (****) as separators in pathnames. This is because *R* uses the backslash character as an "escape" character (i.e. a special character that alters the meaning of the subsequent character).

The file **addWGSR.r** contains the WHO standards and reference database and may take several seconds to load.

Z.2 How to use the addWGSR() function.

We will retrieve a survey dataset in the `c:\dataquality\` directory:

```
svy <- read.table("c:/dataquality/z.ex01.csv", header = TRUE, sep = ",")
head(svy)
```

This returns:

```
  psu age sex weight height muac oedema
1   1  10   1    5.7   64.2  125      2
2   1  10   2    5.8   64.4  121      2
3   1   9   2    6.5   62.2  139      2
4   1  11   9    6.5   64.9  129      2
5   1  24   2    6.5   72.9  120      2
6   1  12   2    6.6   69.4  126      2
```

The file **z.ex01.csv** is a comma-separated-value (CSV) file containing anthropometric data from a Rapid Assessment Method (RAM) survey from Burundi.

Anthropometric indices (e.g. weight-for-height z-scores) have not been calculated and added to the data.

We will use the **addWGSR()** function to add weight-for-height (wfh) z-scores to the example data:

```
svy <- addWGSR(data = svy, sex = "sex", firstPart = "weight",
               secondPart = "height", index = "wfh")
head(svy)
```

A new column named **wfhz** has been added to the dataset:

```
  psu age sex weight height muac oedema  wfhz
1   1  10   1    5.7   64.2  125      2 -2.73
2   1  10   2    5.8   64.4  121      2 -2.04
3   1   9   2    6.5   62.2  139      2  0.13
4   1  11   9    6.5   64.9  129      2    NA
5   1  24   2    6.5   72.9  120      2 -3.44
6   1  12   2    6.6   69.4  126      2 -2.26
```

The **wfhz** column contains the weight-for-height (wfh) z-scores calculated from the variable **sex**, **weight**, and **height** in the **svy** dataset. The calculated z-scores are rounded to two decimals places unless the **digits** option is used to specify a different precision (see *Table Z1*).

The **addWGSR()** function takes up to nine parameters to calculate each index separately, depending on the index required. These are described in *Table Z1*. The parameter **thirdPart** is needed only to calculate BMI-for-age. The **output**, **digits** and **standing** parameters are optional. All other parameters (**data**, **sex**, **firstPart**, **secondPart** and **index**) must be specified.

Table Z1. The nine parameters of the **addWGSR()** function.

Parameter	Description	Detail
data	Name of the survey dataset.	This should be an *R* data frame
sex	Name of the column in **data** that holds data on the sex of each respondent.	This must be present and coded as: **1** = male **2** = female
firstPart	Name of the column in **data** that holds the first component of the required index in the required units.	The first component of the index is: Weight (in **kg**) for wfl, wfh, wfa, bfa Height or length (in **cm**) for lfa, hfa, bfa MUAC (in **cm**) for mfa Sub-scapular skinfold (in **mm**) for ssa Triceps skin fold (in **mm**) for tsa Head circumference (in **cm**) for hca The column (variable) name must be specified as a character string (e.g. **"weight"** not **weight**).
secondPart	Name of the column in **data** that holds the second component of the required index in the required units.	The second component of the index is: Age (in **days**) for all "–for-age" indices except bfa Height or length (in **cm**) for bfa, wfh or wfl The column (variable) name must be specified as a character string (e.g. **"age"** not **age**).
thirdPart	Name of the column in **data** that holds age (in days) when calculating bfa z-scores.	Age (in **days**) for bfa. This parameter need only be specified when calculating bfa z-scores. The column (variable) name must be specified as a character string (e.g. **"age"** not **age**).
index	The short name of the index required.	This should be one of: **"wfl"** Weight for length **"wfh"** Weight for height **"lfa"** Length for age **"hfa"** Height for age **"wfa"** Weight for age **"bfa"** BMI for age **"mfa"** MUAC for age **"tsa"** Triceps skinfold for age **"ssa"** Sub-scapular skinfold for age **"hca"** Head circumference for age The **index** parameter must be specified as a character string (e.g. **"wfa"**) note **wfa**).
standing	The name of column (variable) specifying how "stature" was measured.	This optional parameter should be coded as: **1** = Standing **2** = Supine **3** = Unknown All other values will be recoded to **3** = Unknown. The column (variable) name must be specified as a character string (e.g. **"measured"** not **measured**). If no column (variable) name is specified height and age rules will be applied (see *Table Z2*).
output	The name of the column containing the specified index to be added to the survey dataset.	If you do not specify a value for **output** then the added column will take the short name of the specified **index** with a **z** appended (e.g. **wfaz**). A different **output** parameter must be specified as a character string (e.g. **"xyz1"** not **xyz1)**.
digits	The number of decimal places for **output**.	This is an optional parameter (e.g. **digits = 4**). The default is to round the calculated z-scored to two decimal places.

The **standing** parameter in *Table Z1* specifies how "stature" (i.e. length or height) was measured. If this is not specified, and in some special circumstances, height and age rules will be applied when calculating z-scores. These rules are described in *Table Z2*.

Table Z2. Height and age rules applied by the **addWGSR()** function.

index	standing	age	height*	Action
"hfa" or "lfa"	Standing	< 731 days		index ="lfa" height = height + 0.7 cm
	Supine			index = "lfa"
	Unknown			index = "lfa"
	Standing	≥ 731 days		index = "hfa"
	Supine			index = "hfa" height = height – 0.7 cm
	Unknown			Index = "hfa"
"wfh" or "wfl"	Standing		< 65 cm	index = "wfl" height = height + 0.7 cm
			≥ 65 cm	index = WFH
	Supine		≤ 110 cm	index = "wfl"
			> 110 cm	index = "wfh" height = height – 0.7 cm
	Unknown		< 87 cm	index = "wfl"
			≥ 87 cm	index = "wfh"
"bfa"	Standing	< 731 says		height = height + 0.7 cm
	Supine	≥ 731 days		height = height - 0.7 cm

* Height in this table refers to the variable holding "stature" as either height or length

The **addWGSR()** function will not produce error messages unless there is something very wrong with the data or the specified parameters. If an error is encountered in a record then the value **NA** is returned. Error conditions are listed in *Table Z3*.

Table Z3. Error conditions tested by the addWGSR() function

Error condition	Action
Missing or nonsense value in **standing** parameter	Set **standing** to **3** (unknown) and apply appropriate height or age rules (see *Table Z2*).
Unknown **index** specified	Return **NA** for z-score.
Missing **sex**	
Missing **firstPart**	
Missing **secondPart**	
sex is not male (**1**) or female (**2**)	
firstPart is not numeric*	
secondPart is not numeric*	
Missing **thirdPart** when **index = "bfa"**	
thirdPart is not numeric* when **index = "bfa"**	
secondPart is out of range for specified **index**	

* This refers to the column / variable type not to individual values.

We can see this error behaviour using the example data:

```
table(is.na(svy$wfhz))
```

This returns:

```
FALSE  TRUE
  220     1
```

We can display the problem record:

```
svy[is.na(svy$wfhz), ]
```

This returns:

```
  psu age sex weight height muac oedema wfhz
4   1  11   9    6.5   64.9  129      2   NA
```

The problem is due to the value **9** in the **sex** column, which should be coded **1** (for male) and **2** (for female). Z-scores are only calculated for records with sex specified as either **1** (male) or **2** (female). All other values, including **NA**, will return **NA**.

The **addWGSR()** function requires that data are recorded using the required units or required codes (see *Table Z1*).

The **addWGSR()** function will return incorrect values if the data are **not** recorded using the required units. For example, this attempt to add weight-for-age z-scores to the example data:

```
svy <- addWGSR(data = svy, sex = "sex", firstPart = "weight",
               secondPart = "age", index = "wfa")
```

will give incorrect results:

```
summary(svy$wfaz)
```

The odd range of values:

```
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
 3.450   7.692   9.840   9.684  11.430  15.900       1
```

is due to **age** being recorded in months rather than days.

It is simple to convert all ages from months to days:

```
svy$age <- svy$age * (365.25 / 12)
head(svy)
```

before calculating and adding weight-for-age z-scores:

```
svy <- addWGSR(data = svy, sex = "sex", firstPart = "weight",
               secondPart = "age", index = "wfa")
head(svy)
summary(svy$wfaz)
```

The **muac** column in the example dataset is recorded in millimetres (mm). We need to convert this to centimetres (cm) before using the **addWGS()** function to calculate MUAC-for-age z-scores:

```
svy$muac <- svy$muac / 10
head(svy)
svy <- addWGSR(svy, sex = "sex", firstPart = "muac",
               secondPart = "age", index = "mfa")
head(svy)
```

As a last example we will use the **addWGSR()** function to add body mass index-for-age (bfa) z-scores to the data to create a new variable called **bmiAgeZ** with a precision of 4 decimal places as:

```
svy <- addWGSR(data = svy, sex = "sex", firstPart = "weight",
       secondPart = "height", thirdPart = "age", index = "bfa",
       output = "bmiAgeZ", digits = 4)
head(svy)
```

Be careful if you copy and paste the *R* commands from this document and then edit them to create new commands. It is best to do this in the user interface that you use with R, or in a plain text editor (see **Troubleshooting**, below). This is because word processors such as Microsoft Word® tend to use "smart" asymmetrical quotes like this **"text"** rather than plain, symmetrical quotes such as in **"text"**. *R* does not recognise the asymmetrical quotes and will give an error message if they are used.

Z.3 Creating indicators from indices

The classical analysis of prevalence requires the application of a case-definition to a z-score to create an indicator, or to a raw measurement (e.g. MUAC) to create an indicator.

Cases definitions are usually applied after data have been checked and cleaned and after nutritional indices have been calculated and added. Care should be taken to apply flagging criteria to anthropometric indices. The NIPN anthropometry data quality toolkit describes methods to check all data before calculating any indices and can then add flags to outliers.

Table Z4 shows the ranges in values of z-scores used to calculate the main anthropometric indices. The ranges given are for each index for all cases (e.g. in total) and separately for moderate or severe cases.

Table Z4. Case definitions of anthropometric indicators of undernutrition derived from the anthropometric indices for children calculated by the **`addWGSR()`** function.

	Anthropometric indicator			
Anthropometric index	**Total if … z-score <-2**	**Moderate if … z-score -3 ≤ Z < -2**	**Severe if … z-score <-3**	**Age range (months)**
Weight-for-length	Wasted	Moderately wasted	Severely wasted	0 - 24
Weight-for-height	Wasted	Moderately wasted	Severely wasted	>24 - 60
Length-for-age	Stunted	Moderately stunted	Severely stunted	0 - 24
Height-for-age	Stunted	Moderately stunted	Severely stunted	>24 - 228
Weight-for-age	Underweight	Moderately underweight	Severely underweight	0 - 120
Body mass index-for-age	Thin	Moderately thin	Severely thin	0 - 228
MUAC-for-age	No term	No term	No term	3 - 60
Triceps skinfold-for-age	No term	No term	No term	3 - 60
Sub-scapular skinfold-for-age	No term	No term	No term	3 - 60
Head circumference-for-age	No term	No term	Microcephalic	0 - 60

We will use the **addWGSR()** function to add height-for-age (hfa) z-scores to the example data using age calculated in days.

```
svy <- addWGSR(data = svy, sex = "sex", firstPart = "height",
               secondPart = "age", index = "hfa")
head(svy)
```

A new column named **hfaz** has been added to the dataset:

```
  psu      age sex weight height muac oedema  ...   hfaz
1   1 304.3750   1    5.7   64.2 12.5      2  ...  -3.97
2   1 304.3750   2    5.8   64.4 12.1      2  ...  -2.86
3   1 273.9375   2    6.5   62.2 13.9      2  ...  -3.29
4   1 334.8125   9    6.5   64.9 12.9      2  ...     NA
5   1 730.5000   2    6.5   72.9 12.0      2  ...  -4.19
6   1 365.2500   2    6.6   69.4 12.6      2  ...   1.79
```

The **hfaz** column contains the height-for-age (hfa) z-scores calculated from the **sex**, **height**, and **age** in the **svy** dataset.

First we should check for outliers using the thresholds described in Section 9 of the NIPN Toolkit, *Identifying outliers using flags*, and add a flag variable:

```
svy$flag <- ifelse(!is.na(svy$hfaz) & (svy$hfaz < -6 | svy$hfaz > 6), 1, 0)
head(svy)
table(svy$flag)
```

There are no flagged records.

We can now apply a case definition to identify all children who are stunted in which 1=stunted and 2=not stunted:

```
svy$tst <- ifelse(svy$hfaz < -2, 1, 2)
head(svy)
```

Case definitions for children who are moderately stunted and severely stunted can be applied in a similar manner:

```
svy$mst <- ifelse(svy$hfaz >= -3 & svy$hfaz < -2, 1, 2)
svy$sst <- ifelse(svy$hfaz < -3, 1, 2)
head(svy)
```

In some cases it is necessary to account for additional variables when applying case-definitions. For example, the case definition of wasted is only one part of the case definition for "acute malnutrition" The definition of total or "global"[1] acute malnutrition (GAM) and severe acute malnutrition (SAM) include bilateral pitting oedema in their case definitions. In this case we would use the variable `wfhz` that we created earlier with the variable **`oedema`**:

```
svy$gam <- ifelse(svy$wfhz < -2 | svy$oedema == 1, 1, 2)
svy$mam <- ifelse(svy$wfhz >= -3 & svy$wfhz < -2 & svy$oedema != 1, 1, 2)
svy$sam <- ifelse(svy$wfhz < -3, 1, 2)
head(svy)
```

Indices with a weight component may be upwardly biased if cases of oedema are included in the calculation. In the case of weight-for-age z-scores, for example, we would use something like:

```
svy <- addWGSR(data = svy, sex = "sex", firstPart = "weight",
             secondPart = "age", index = "wfa")
svy$tuw <- ifelse(svy$wfaz < -2 & svy$oedema != 1, 1, 2)
svy$muw <- ifelse(svy$wfaz >= -3 & svy$wfaz < -2 & svy$oedema != 1, 1, 2)
svy$suw <- ifelse(svy$wfaz < -3 & svy$oedema != 1, 1, 2)
head(svy)
```

Alternatively we may apply the case definitions and the exclude children with oedema:

```
svy$tuw <- ifelse(svy$wfaz < -2, 1, 2)
svy$muw <- ifelse(svy$wfaz >= -3 & svy$wfaz < -2, 1, 2)
svy$suw <- ifelse(svy$wfaz < -3, 1, 2)
svy$tuw <- ifelse(svy$oedema == 1, NA, svy$tuw)
svy$muw <- ifelse(svy$oedema == 1, NA, svy$tuw)
svy$muw <- ifelse(svy$oedema == 1, NA, svy$tuw)
head(svy)
```

Z.4 Saving the dataset

You will usually want to save a dataset once you have calculated and added z-scores and applied case-definitions.

It is simple to save the dataset with the new variables in a comma-separated-value (CSV) text file using the **`write.table()`** function:

```
write.table(x = svy, file = "z.ex01.zscores.csv", sep = ",", quote = FALSE,
       row.names = FALSE, fileEncoding = "ASCII")
```

Before you use the added indices or case-definitions you should check for outliers and implausible values using the methods outlined in Section 9 of the NIPN anthropometry data quality toolkit called ***Identifying outliers using flags***.

[1] Global is a direct use of the French word 'globale' which means overall or total

Z.5 Troubleshooting addWGSR()

Do not edit commands in a text editor such as Microsoft Word ® as it can introduce characters and symbols that may not be recognised by R. Use a plain text editor such as WordPad orNotepad++ on Windows; EMACS, VIM, or Scintilla on Linux; BBEdit on MacOS; or the editor provided by your preferred *R* environment (e.g. RStudio).

Here are some things to check if you experience difficulties using the **addWGSR()** function:

Make sure you have loaded the **addWGSR()** function. An error message such as:

```
Error in addWGSR(data = svy, sex = "sex", firstPart = "weight", :
  could not find function "addWGSR"
```

means that the **addWGSR()** function has not been loaded.

Make sure all variables are recorded using the required units (see *Table Z1*). Transform variables to the required units if necessary.

Make sure you have specified the essential parameters correctly. An error message such as:

```
Error in addWGSR(sex = "sex", firstPart = "muac", :
  argument "data" is missing, with no default
```

means that you have failed to specified a required parameter (**data** in this example).

An error messages such as:

```
Error in (function(x, i, exact) if (is.matrix(i)) :
  object 'muac' not found
```

is usually due to a column being specified without enclosing its name in double quote (") characters (specifying **muac** instead of **"muac"** in this example).

An error message such as:

```
Error in if (is.na(sex) | is.na(firstPart) | is.na(secondPart)) { :
  argument is of length zero
In addition: Warning message:
In is.na(firstPart) :
  is.na() applied to non-(list or vector) of type 'NULL'
```

is usually due to a misspelled column name.

An error message such as:

```
Error in index %in% c("bfa", "hca", "hfa", "lfa", "mfa" :
  object 'mfa' not found
```

is usually due to specifying **index** without enclosing double quotes.

Check that columns are of the correct type. The **addWGSR()** assumes that all columns contain numeric data and can produce incorrect results if this not the case.

If **all** z-scores are **NA** then you should check that the required data is present in the dataset, check that you have specified the column names correctly, check that **index** is specified correctly, and check the units used in the data.

Make sure that you have used "plain" quotes not “smart” quotes around parameter names.

If you cannot find a solution to a problem with **addWGSR()** then you should contact the Global Support Facility for the NIPN initiative for assistance:

gsf_nipn@agropolis.fr

Z.6 References

WHO (2006a). *WHO Child Growth Standards. Length/height-for-age, weight-for-age, weight-for-length, weight-for-height and body mass index-for-age. Methods and development*, World Health Organization, Geneva.

WHO (2006b). *WHO Child Growth Standards. Head circumference-for-age, arm circumference-for-age, triceps skinfold-for-age, and subscapular skinfold-for-age. Methods and development*, World Health Organization , Geneva.

de Onis M, Onyango AW, Borghi E, Siyam A, Nishida C, Siekmann J (2007). *Development of a WHO growth reference for school-aged children and adolescents,* Bulletin of the World Health Organization, **85**: 660-7

NiPN
National Information Platforms for Nutrition

Summary

This document has been commissioned by the Global Support Facility for the National Information Platforms for Nutrition initiative. It presents a set of practical analytical methods that can be applied to variables in datasets to assess their quality. An index of data quality that both describes and scores the quality of the data is also presented. The focus of this toolkit is on data required to assess the anthropometric status of humans, such as measurements of weight, height or length, mid upper-arm circumference (MUAC), sex and age. However, many of presented methods could be applied to other types of data. Additional toolkits may be prepared to examine other variables or other types of variables. The material is intended to provide a practical or "hands on" introduction to assessing data quality and is presented as a series of computer-based exercises.

National Information Platforms for Nutrition is an initiative of the European Commission's Directorate General for Cooperation and Development, also supported by the United Kingdom Department for International Development and the Bill & Melinda Gates Foundation.

European Commission

UKaid from the British people

BILL & MELINDA GATES foundation

GSF-NIPN
Agropolis International
1000 avenue Agropolis
34394 Montpellier cedex 5
France

www.nipn-nutrition-platforms.org
gsf_nipn@agropolis.fr