Package provenance - Quick start guide

Martin Rittner

18/08/2014

1 1. Introduction

Package `provenance` provides functions for the visualisation of typical data utilised in sediment provenance analysis in the geosciences, as well as helper functions aiding data analysis. Plotting takes advantage of the framework provided by the `ggplot2` package, and the output plots are ggplot objects, which allows for further modification with `ggplot2`'s functions and easy saving of the graphics. This document is meant for geoscientists who want to get step-by-step guidelines for using `provenance`'s plotting functions, and who might not be experienced users of the R language.
The "workhorse functions" of the package are:

`plotKDE()` provides plotting of kernel density estimates (**KDE**s) of geochronological data for single or suites of samples

`plotMDS()` provides plotting of multidimensional scaling (**MDS**) maps for quick visual identification of trends and similar groups within a set of sample data

`plot3Way()` is designed to be very similar to `plotMDS()`, providing individual scaling (or **3-way MDS**) maps of data sets combining several different provenance proxies

Working with provenance data using the `provenance` package in R typically involves three main steps:

1. load data files, filter and reformat input data for use
2. plot visualisation of data
3. save output

Of these steps, 2 usually is an iterative "trial-and-error" process to determine best graphical representation of the data, with a few additional calculations necessary here and there. Step 3 is, due to `ggplot2`'s functionality, a trivial call to `ggsave()`. The first item, loading the data, is usually the most involved and requires the user to write their own scripts for the purpose, as this can not easily be standardised over the wide range of possible data file formats and the individual requirements of each user. This step can be further subdivided into:

1a. loading data files
1b. reformattiong data
1c. adding information for visualisation

All basic steps will be presented in workable examples in section 2 of this document. Section 3 illustrates the effects of different settings of the parameters given to the main plotting functions. Section 4 gives a basic generic script that should be easily adaptable to the user's needs. Section 5 contains additional tips & tricks on plotting and data import.

2 Basic workflow

First, we need to load the package:

```
library(provenance)
```

2.1 Loading data

For this document, we use example data files installed with `provenance`, but the principles would be the same for any other data file. Simple example data files are provided in the /inst subfolder of the package installation folder, these can serve as an example of how to prepare data. Alternatively, the data loading process can be adapted to the existing data files, and the data restructured in R, to obtain objects suitable for the plotting functions. The latter approach is preferrable, as it leaves the original data files untouched, and new and altered data is easily replotted by simply re-running the R script, instead of preparing intermediate data files by hand. See also section Tips & Tricks for notes on how to import lists of individual data files and MS Excel files.

2.1.1 Loading data files

In the /inst folder, there is a simple example file of detrital zircon U-Pb ages. The individual measured ages are arranged column-wise, one column per sample, with the sample names in the first line.

site1.1	site1.2	site1.3	site1.4	site2.1	site2.2	site2.3	site2.4	site2.5	bedrock	site3
398.6	1084.9	224.3	3517.8	433.5	2358.2	525.5	403.7	404.6	320.5	121
1980.0	503.2	415.9	828.1	403.0	517.2	473.8	349.7	516.6	1941.0	192
1980.0	290.0	1861.9	1681.0	217.3	2502.4	776.3	1078.1	2537.4	1891.7	289
1282.3	493.9	1311.2	332.4	240.5	445.1	1130.7	915.1	165.9	542.5	292
198.5	410.7	389.6	828.1	254.1	2140.3	2270.5	2534.5	1065.0	2420.9	339
346.1	1346.0	1938.7	1681.0	227.1	2358.2	1155.5	2390.4	2431.0	272.4	381
248.7	750.5	2221.3	332.4	236.9	1828.7	2106.9	423.4	424.3	279.1	396
1282.3	278.4	465.3	1798.6	236.9	2502.4	683.2	419.0	425.4	945.7	407

To load this data, you could use:

```
data<-read.csv(file.path(path.package("provenance"),"zircon_ages.csv"),
    stringsAsFactors=FALSE)
```

The data table is loaded with `read.csv()`, the result is a data frame. The default behaviour of R is to convert character vectors into factors (see R help); since `read.csv()` reads the (text) file with headers first, and more importantly in the case of reading Excel files (see Tips & Tricks) which are converted in csv files in an intermediate step in the background, numerical values can end up being interpreted as factors of strings that look like numbers, but cause nothing but headaches and confusion. For most cases discussed here, factors should be avoided.
The functions within `provenance` generally work with lists, where each element of the list would be a vector of ages for one sample, respectively. Since data frames are a suclass of list, no further conversion is needed. Before plotting, we can have a look at some properties of the data:

```
names(data)
```

```
##  [1] "site1.1" "site1.2" "site1.3" "site1.4" "site2.1" "site2.2" "site2.3"
##  [8] "site2.4" "site2.5" "bedrock" "site3"   "Area1"   "Area2"   "River"
## [15] "Area3"
```

We see there is data from three sampling sites, two of which are comprised of several subsamples, three potential source areas ("Area 1 - 3"), a river sample and a bedrock sample.

2.1.2 Reformatting and filtering data

Depending on the structure of the input data and the intended analysis, this step might be trivial, complex, or possibly not necessary at all.

As an example, let's assume we want to lump all the individual data from site 1 and 2 together. We'll save the result in a new list, in case we wanted to use the original one later on.

```
data2<-list()
sitenames<-unique(sapply(strsplit(names(data),"\\."),
    FUN=function(x){return(x[[1]][1])}))
for(s in sitenames){
    data2[[s]]<-unlist(data[grep(s,names(data))])
}
```

We applied the `strsplit()` function to extract the site names, taking advantage of the fact the data columns all had names in a "sitex.x" pattern. The standard functions applied here are not scope of this document, please see the R help for further details.

2.1.3 Adding information for visualisation

For e.g. colouring purposes, additional information can be added to the data. For example, we might classify our data into sampling sites, source areas, river and bed rock.

```
datatype<-rep("n/a",length(data))
names(datatype)<-names(data)
datatype[grep("site",names(datatype))]<-"sampling site"
datatype[grep("Area",names(datatype))]<-"source area"
datatype[grep("River",names(datatype))]<-"river"
datatype[grep("bedrock",names(datatype))]<-"bed rock"
```

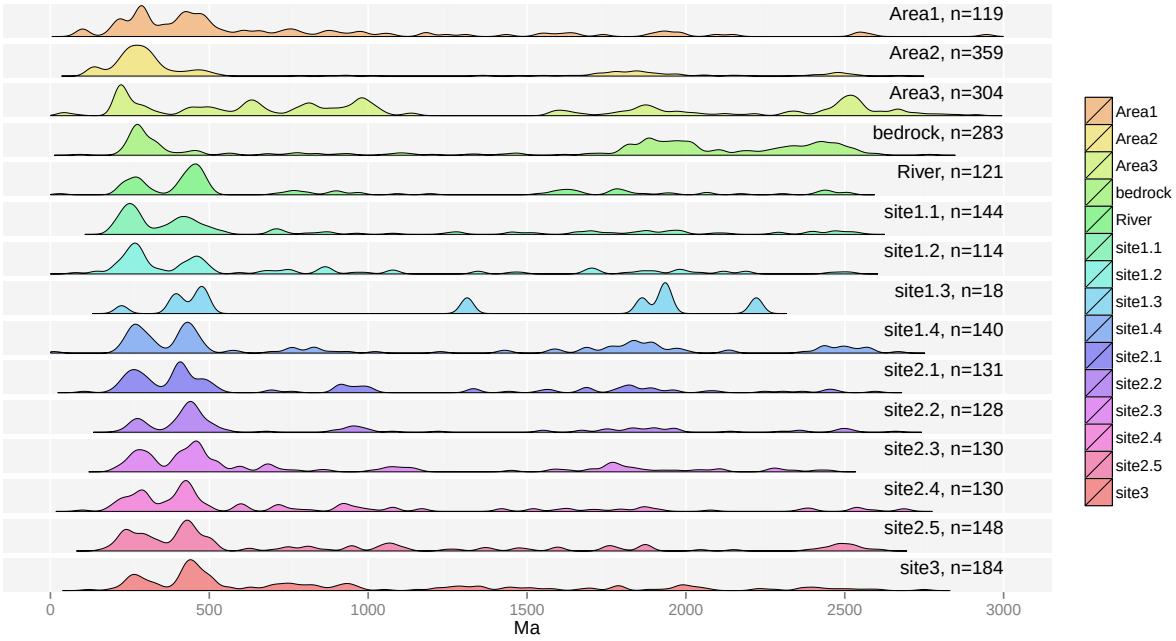
The type of each data set in `data` is now recorded in `datatype` (we'll use this later):

```
##         site1.1         site1.2         site1.3         site1.4
## "sampling site" "sampling site" "sampling site" "sampling site"
##         site2.1         site2.2         site2.3         site2.4
## "sampling site" "sampling site" "sampling site" "sampling site"
##         site2.5         bedrock           site3           Area1
## "sampling site"      "bed rock" "sampling site"   "source area"
##           Area2           River           Area3
##   "source area"         "river"   "source area"
```

2.2 Plotting data

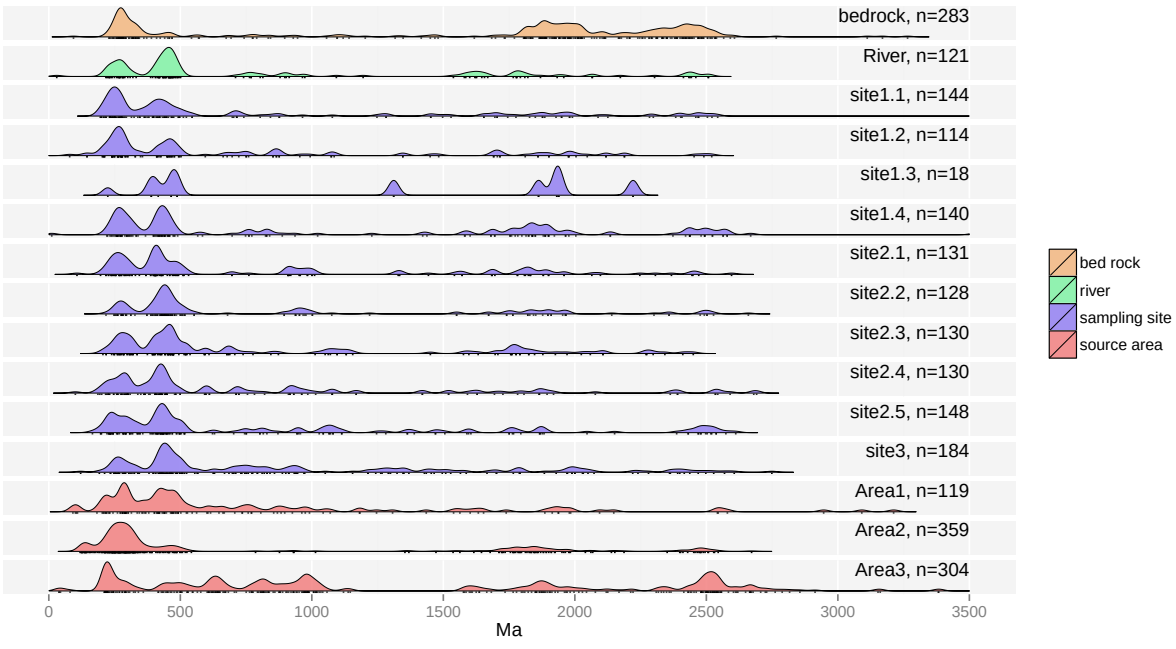
To get a quick look at the loaded data, `plotKDE()` is a good point to start.

```
plotKDE(data)
```



With the `datatype` variable we genrated, and some additional adjustments, we can make this a little clearer. See the examples and help files for details.

```
plotKDE(data,classes=datatype,markers="dash",limits=c(0,3500))
```



2.3 Saving the output

Since plots generated with `provenance` are also ggplot objects, `ggsave()` from package `ggplot2` can be utilised for saving in a wide range of file formats.

```
ggsave("~/test.pdf",width=10,height=8)
```

Adapt the output path, file format (indicated simply by the file extension) and sizes to your needs. You can also save a specific plot, if you stored it in a variable earlier:

```
p1<-plotKDE(data)

#
# ... a lot of clever code here, generating other plots (p2, p3,...)
#

ggsave(plot=p1,filename="~/test.png",dpi=300,width=10,height=8)
# save p2, p3,...
```

3 Examples and detailed description of parameters

In the following, the effects of the many different parameters for the plotting functions are illustrated by example plots. The examples assume `data2` loaded as described previously. Many of the parameters can be used together (see examples at the end of this section), not all possible combinations can be detailed here.

3.1 `plotKDE()`

The full function call to plotKDE is:

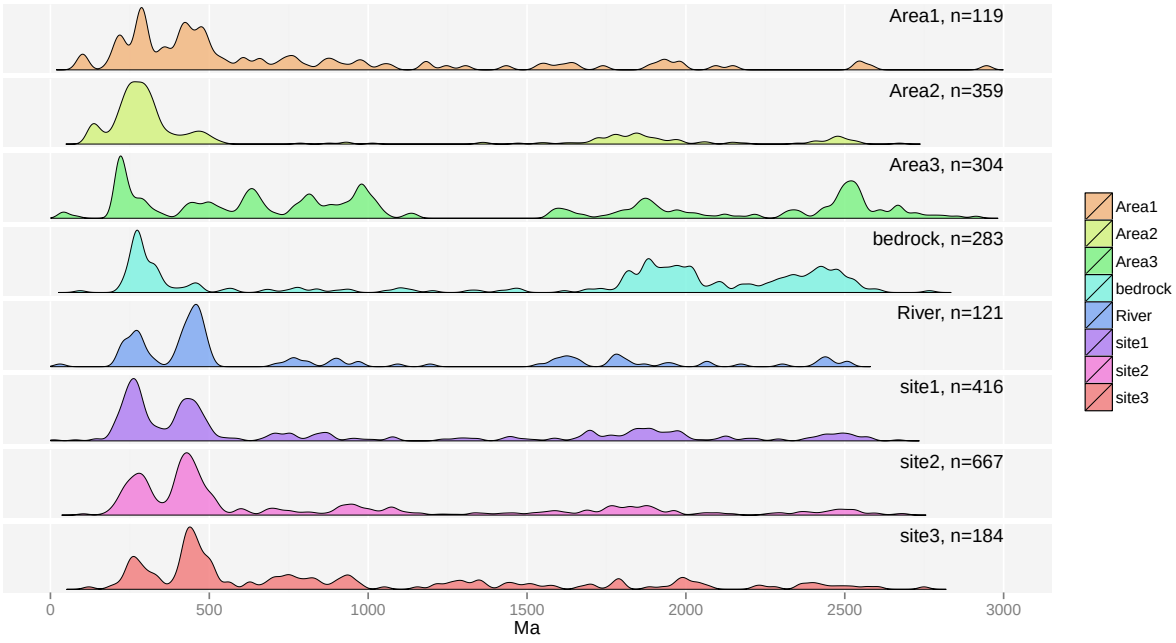
```
plotKDE(data, title, limits = c(0, 3000), breaks = NA, bandwidth = NULL,
    fcolour = NA, splitat = NA, plot = names(data), periods = FALSE,
    classes = NA,   hist = FALSE, markers = "none", order = TRUE, logx = FALSE,
    method = "botev",   ...)‘
```

See the help files for details.

> Disclaimer: as of the time of writing (2014-08-19), the `periods` parameter does not yet have any effect. Histograms currently suffer from a bug, causing all plots of a multi-KDE plot to show the same histogram. The `title` parameter currently only allows to set the same title for all plots.

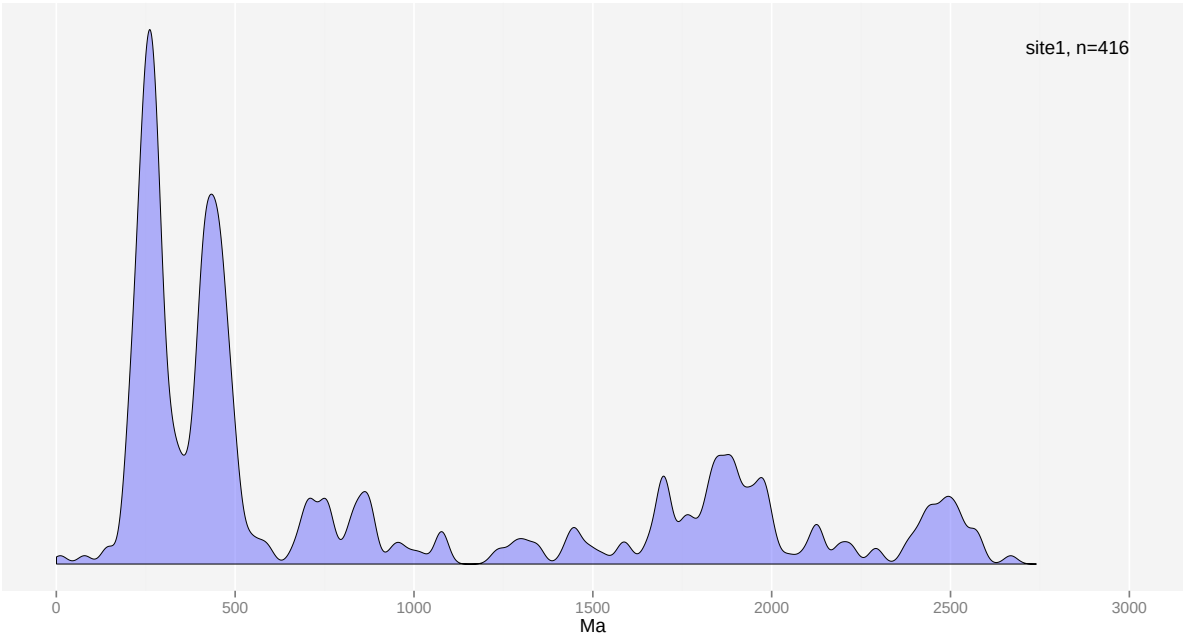
The simplest call plots KDEs for all elements of `data`. Each plot is assigned an individual colour, the default age range is 0 - 3000 Ma, bandwidth is chosen automatically.

```
plotKDE(data2)
```



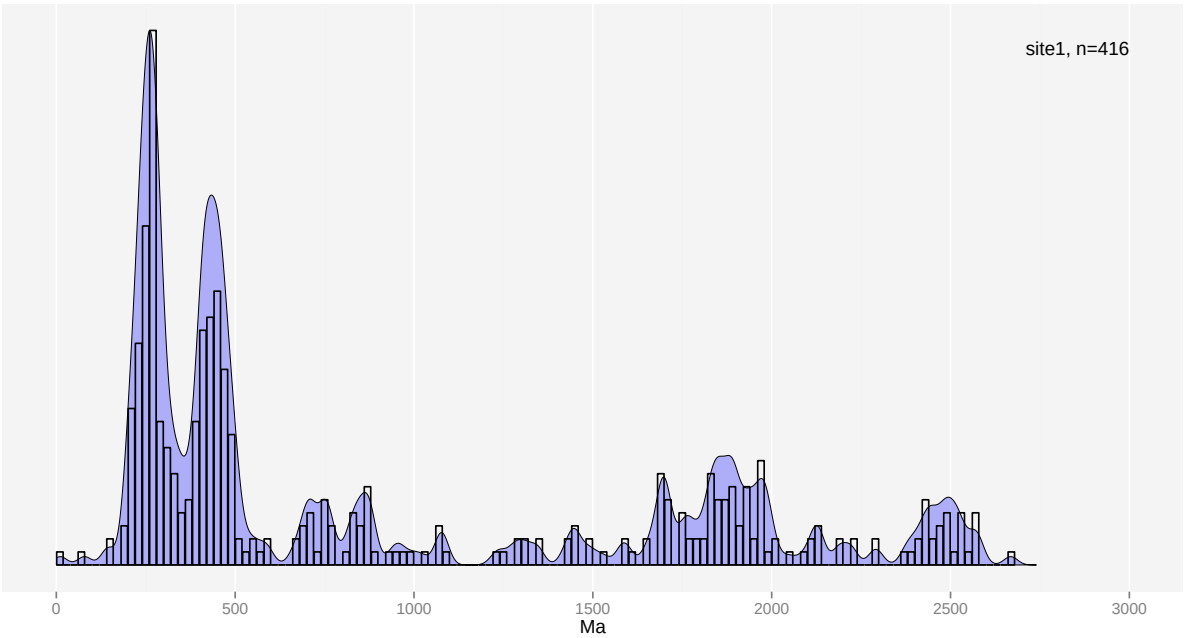
The `plot` parameter allows to select specific data sets (samples) by their names. The below example has the same effect as subsetting the data provided to the function like in e.g. `plotKDE(data[["site1"]])` or `plotKDE(data$site1)`. Note automatic removal of the legend, as it is obsolete here.

```
plotKDE(data2,plot=c("site1"))
```



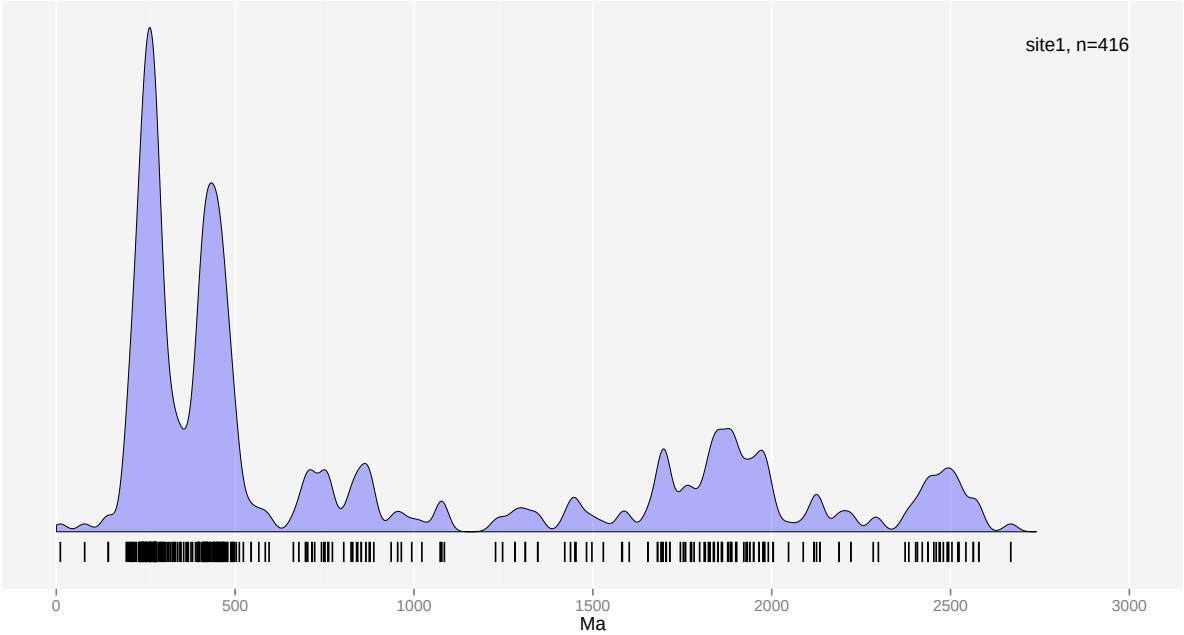
Add histograms.

```
plotKDE(data2,plot=c("site1"),hist=TRUE)
```



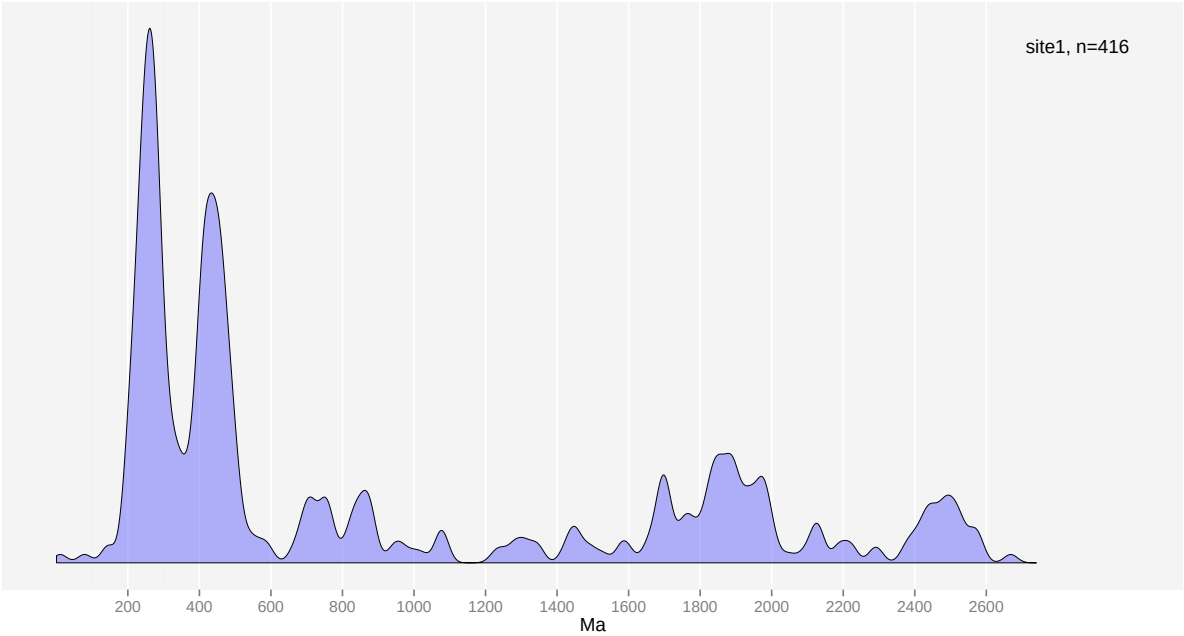
Add data markers. Possible values are `"dash"` for little tick marks and `"circle"` for semi-transparent circles.

```
plotKDE(data2,plot=c("site1"),markers="dash")
```



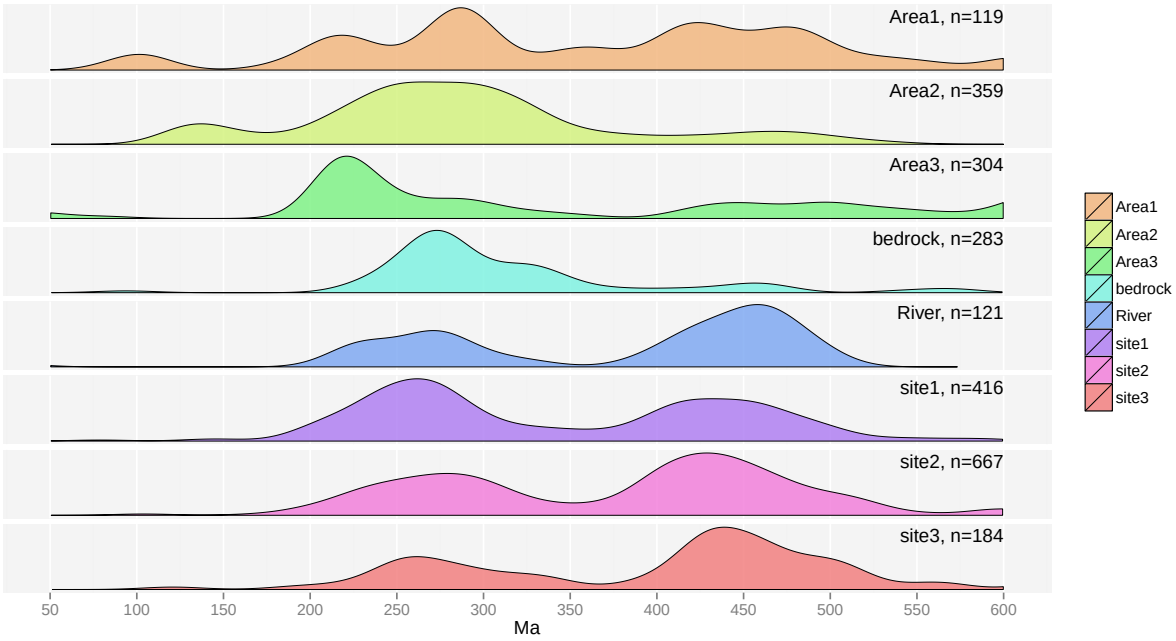
Custom breaks. The `breaks` parameter overrides `limits`, i.e. if any breaks lie outside the limits, the limits will be expanded accordingly.

```
plotKDE(data2,plot=c("site1"),breaks=seq(200,2600,200))
```



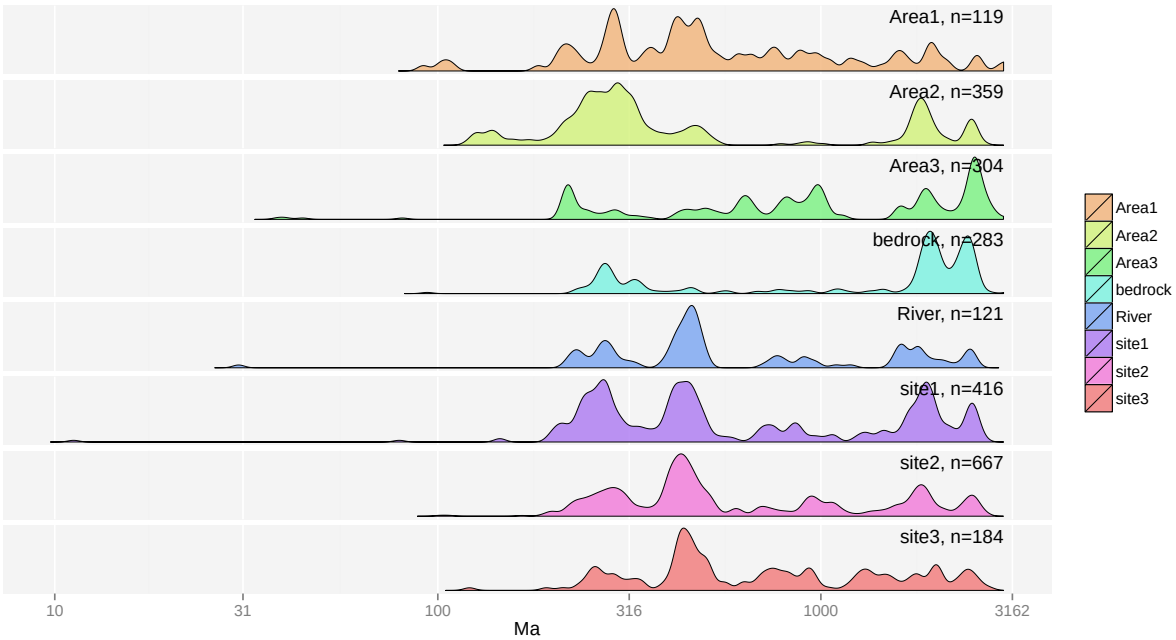
Change x- (time-) limits. Set limits to -1 to automatically use the range of values contained in `data`.

```
plotKDE(data2, limits=c(50,600))
```



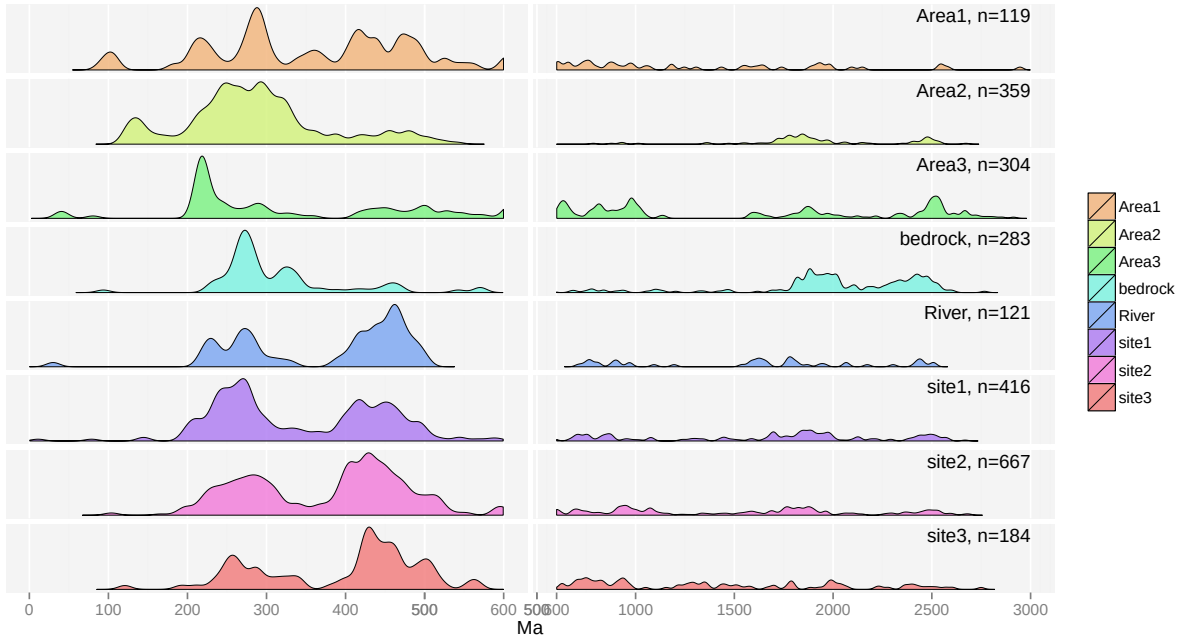
Logarithmic x-scale.

```
plotKDE(data2,logx=TRUE)
```



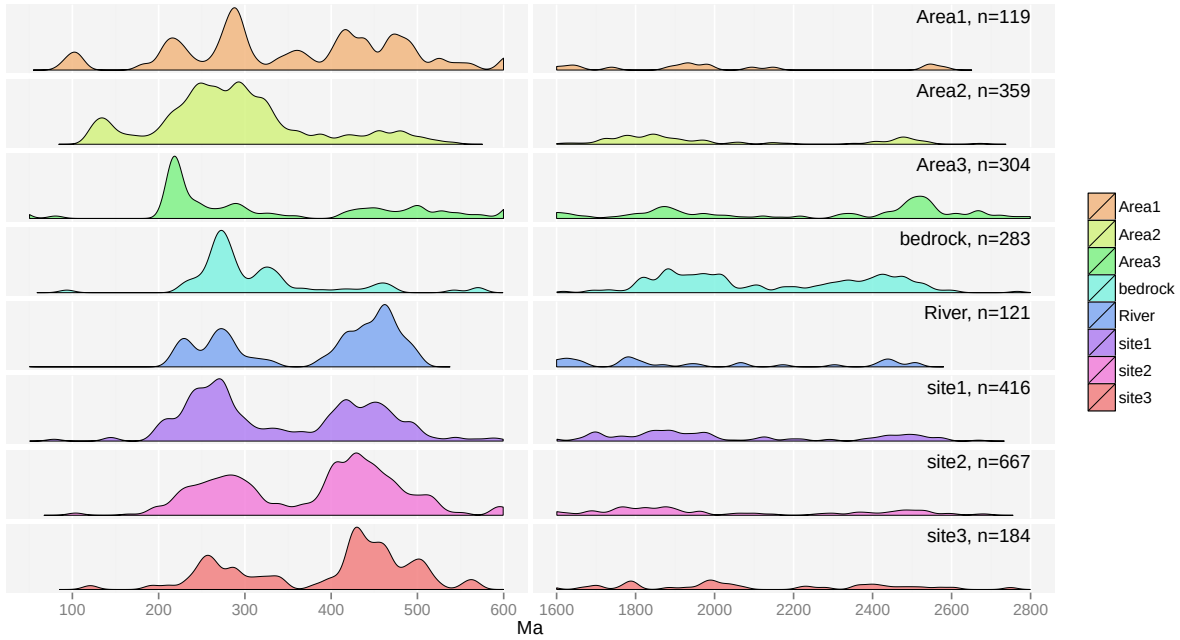
Split plot at a certain age, the two sub-ranges will occupy equal space (useful to emphasise the younger ages).

```
plotKDE(data2,splitat=600)
```



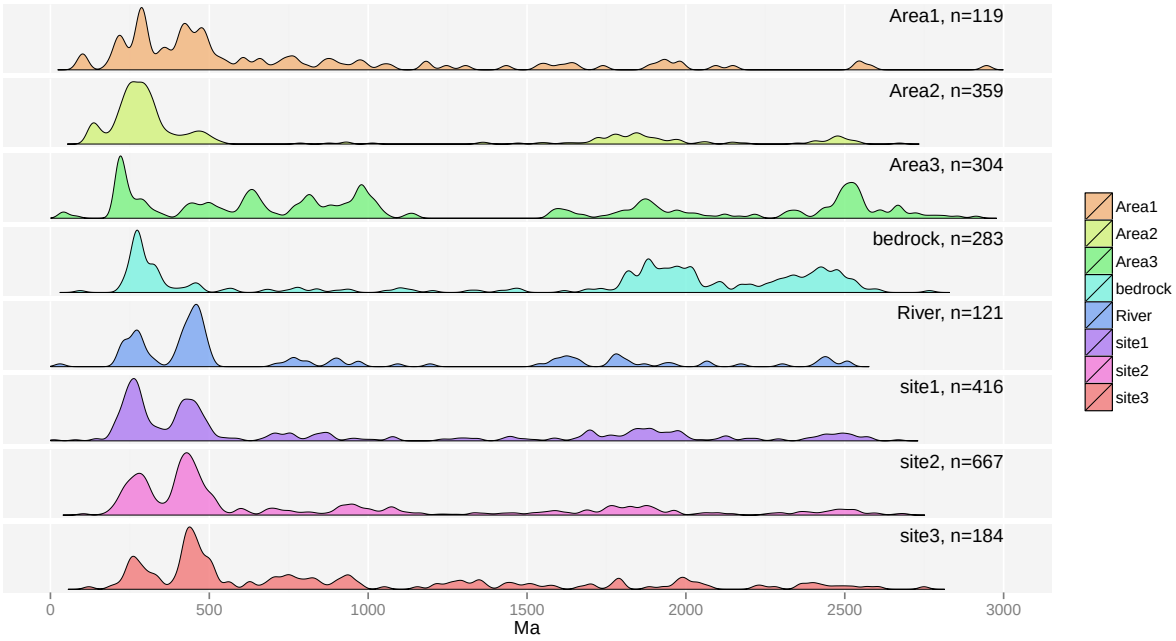
Splitting can be combined with custom ranges for the two half plots. A `limits` parameter of length 4 will override `splitat`.

```
plotKDE(data2,limits=c(50,600,1600,2800))
```



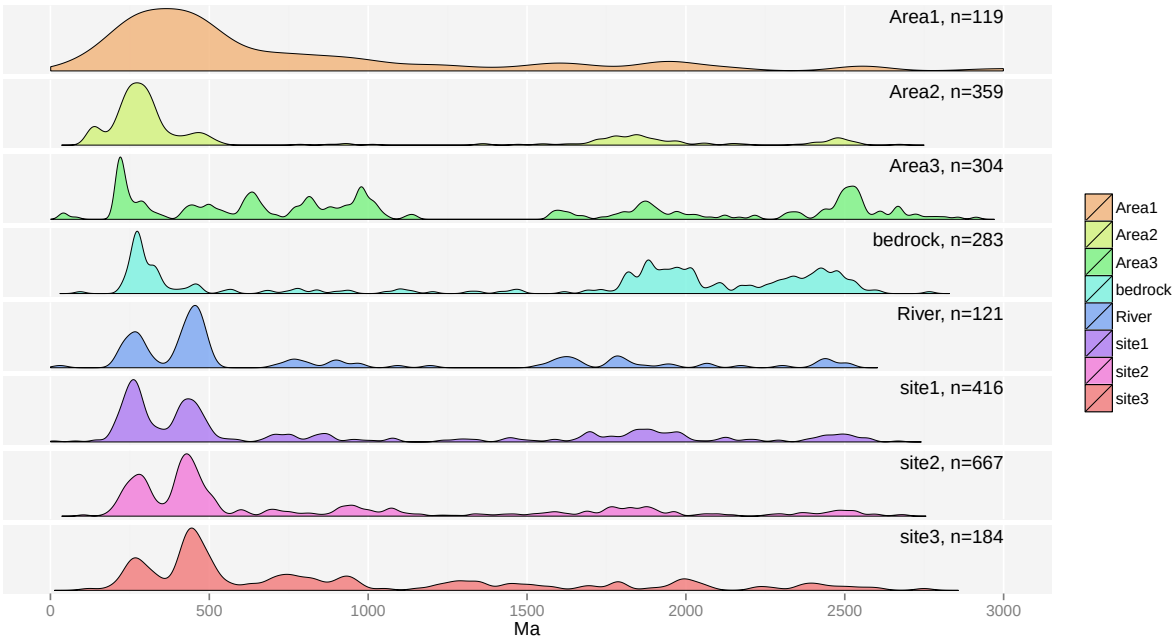
Usually, an "optimal bandwidth" is calculated automatically, and the average of all optimal bandwidths for all data series is set equally for all KDEs. It can also be set manually.

```
plotKDE(data2, bandwidth=16)
```



To plot each KDE with it's own optimal bandwidth, set `bandwidth=-1` (note that this sometimes leads to unexpected results, i.e. severe oversmoothing in data sets with very few data or very regular data distribution, see "Area1").

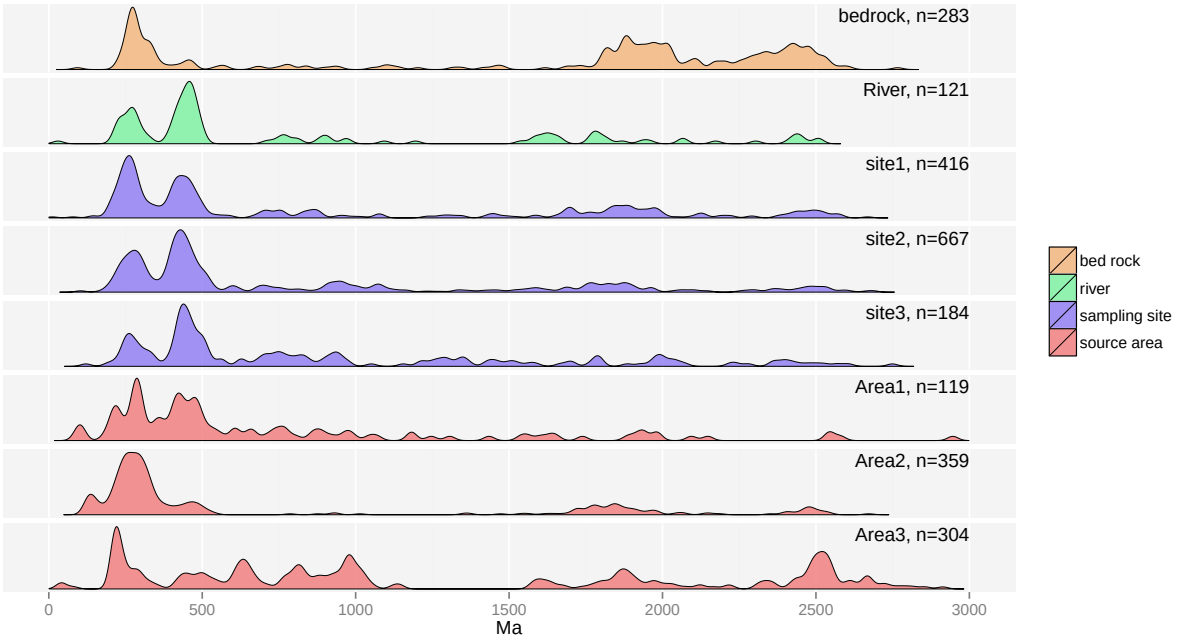
```
plotKDE(data2,bandwidth=-1)
```



The `classes` parameter allows the data to be classified based on the individual values encountered in

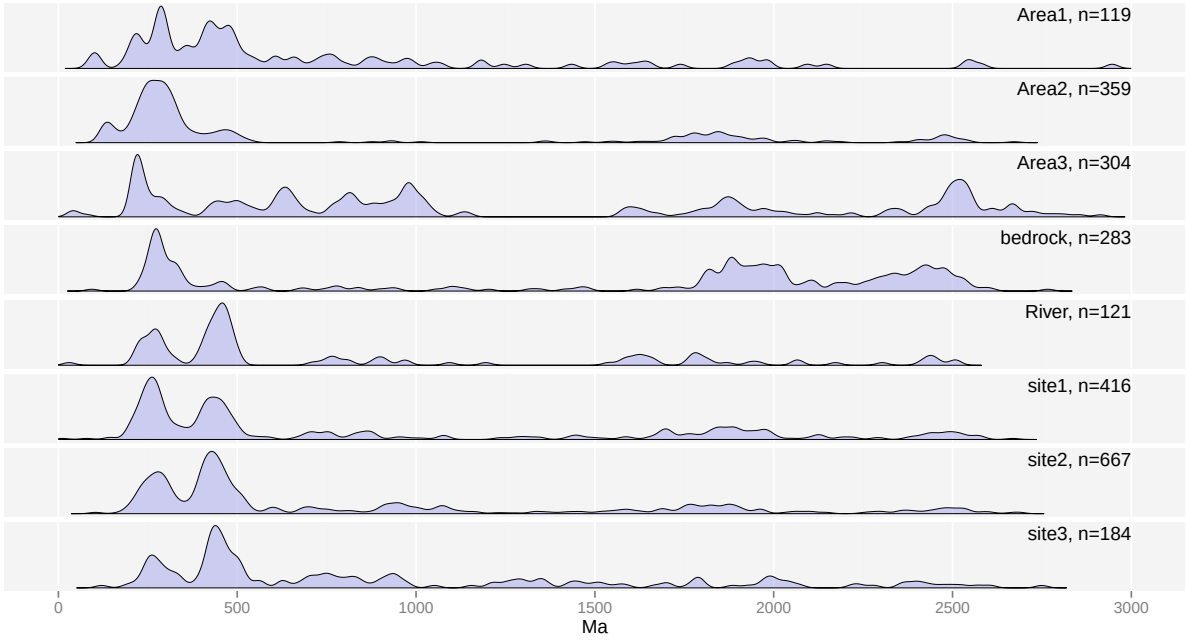
the supplied vector. The classes vector must be of the same length as data, after selection by `plot`. Set `classes=-1` to plot all KDEs in the same colour.

```
datatype2<-rep("n/a",length(data2))
names(datatype2)<-names(data2)
datatype2[grep("site",names(datatype2))]<-"sampling site"
datatype2[grep("Area",names(datatype2))]<-"source area"
datatype2[grep("River",names(datatype2))]<-"river"
datatype2[grep("bedrock",names(datatype2))]<-"bed rock"
plotKDE(data2,classes=datatype2)
```



The `fcolour` parameter allows to specify the fill colour(s) for all KDEs or the provided classes:

```
plotKDE(data2,fcolour="#AAAAEE88")
```



`order`

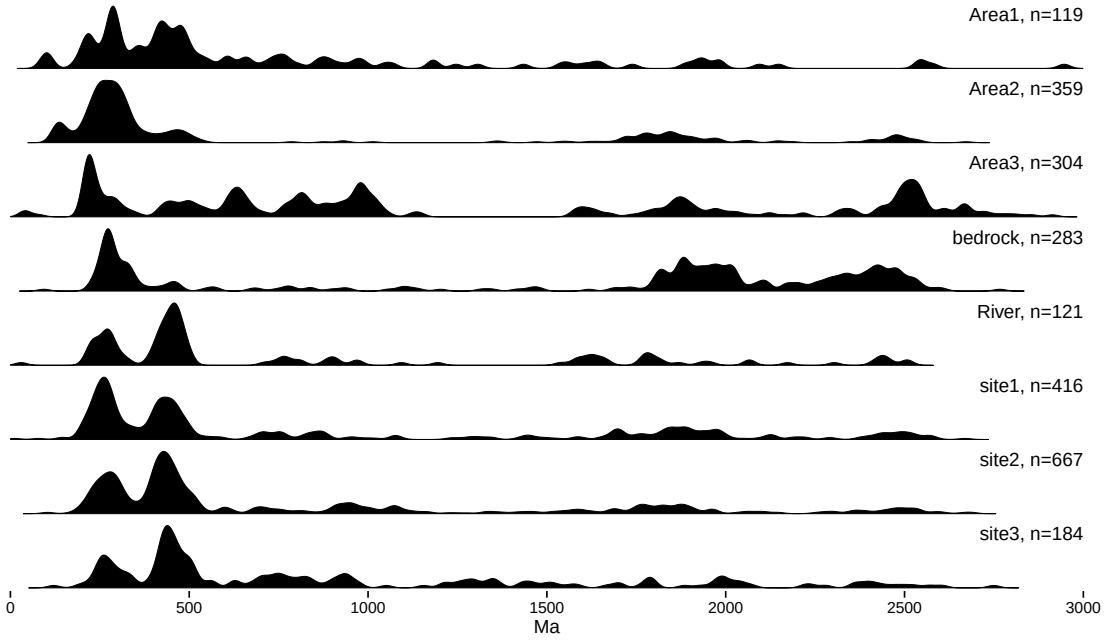
`method`

3.1.1 Examples of combining parameters:

Nicest combined colour plot select only sampling sites plotKDE(data2,plot=c("site1","site2","site3"),limits=c(150,600),bandwid
custom breaks on alog scale

Publication-quality pure black & white plot

```
plotKDE(data2,fcolour="black")+theme(panel.background=element_blank(),
    panel.grid=element_blank(),axis.ticks=element_line(colour="black"),
    axis.text=element_text(size=rel(0.8), colour="black"))
```



3.2 plotMDS()

```
plotMDS(mds, diss, col="", sym="", nearest=TRUE, labels=TRUE, symbols=TRUE,
    fcolour=NA, stretch=FALSE)
```

3.3 plot3Way()

3.4 other useful functions

3.4.1 plotShepard()

```
plotShepard(mds, diss, xlab="dissimilarity", ylab="distance", title="")
```

3.4.2 dissimilarity()

```
dissimilarity(data, metric="K-S")
```

4 Putting it all together

To help new users getting started, we present a short general script summing up the basic steps to load, format and plot a data file as described before. The script follows the presented outline, the individual parts can be altered and adapted to the user's need. Please refer to the details in the previous sections and the R help files for details.

```
########################################
## package 'provenance' example script ##
########################################

# Path to the data file. Change according to your needs, replacing the whole
# file.path() call with the respective value for your system, e.g.:
# for now, we use example data installed with the package
datapath<-file.path(path.package("provenance"), "inst", "zircon_ages.csv")

# datapath<-"C:\\Path\\To\\Some\\data file.csv" #for Windows
# please note that double backslashes (\\) might be needed

# datapath<-"/home/you/path/to/some/data_file.csv"  #Linux, Mac
# please note the single (forward) slashes

# path to save output graphics - adapt as needed
outpath<-"~"

# we need to load the provenance package before we can use it:
require(provenance)

##### Loading data #####
# reading data, store it in a data.frame():
data<-read.csv(datapath, stringsAsFactors=FALSE)

##### Reformatting data #####

##### Adding information for visualisation #####
datatype<-rep("n/a", length(data))
names(datatype)<-names(data)
datatype[grep("site", names(datatype))]<-"sampling site"
datatype[grep("Area", names(datatype))]<-"source area"
datatype[grep("River", names(datatype))]<-"river"
datatype[grep("bedrock", names(datatype))]<-"bed rock"

##### Plotting data #####
# we save the plots in variables instead of plotting them immediately
p1<-plotKDE(data, classes=datatype, markers="dash", limits=c(0,3500))

##### Saving output #####
# see ggsave()'s help for details
ggsave(plot=p1, filename=file.path(outpath, "zrn_KDE.png"),
    dpi=300, width=10, height=8)
ggsave(plot=p2, filename=file.path(outpath, "zrn_MDS.png"),
    dpi=300, width=10, height=8)
ggsave(plot=p3, filename=file.path(outpath, "zrn_Shepard.png"),
    dpi=300, width=2.5, height=2)
```

```
ggsave(plot=p4, filename=file.path(outpath, "HM_MDS.png"),
    dpi=300, width=10, height=8)
ggsave(plot=p5, filename=file.path(outpath, "HM_Shepard.png"),
    dpi=300, width=2.5, height=2)
ggsave(plot=p6, filename=file.path(outpath, "3Way_zrn_HM.png"),
    dpi=300, width=10, height=8)
ggsave(plot=p7, filename=file.path(outpath, "3Way_spaces.png"),
    dpi=300, width=10, height=8)
ggsave(plot=p8, filename=file.path(outpath, "3Way_components.png"),
    dpi=300, width=10, height=8)
```

5 Tips and Tricks

Following are some tips and tricks for commonly encountered issues and advanced requirements.

5.1 Plotting

5.1.1 Adding a plot title

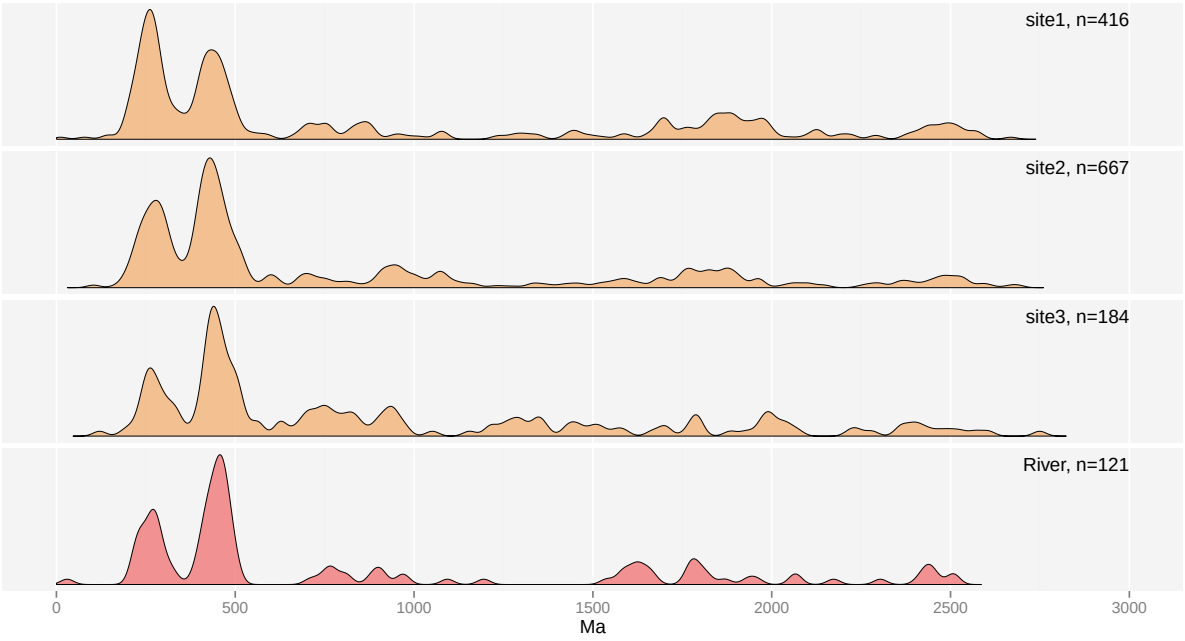
5.1.2 Combining plots

5.1.3 Manual colour scales

5.1.4 Plotting without a legend

In rare cases, one might like to manually remove the legend. This can be easily be achieved with `ggplot2`'s functions.

```
plotKDE(data2,plot=c("site1","site2","site3","River"),classes=c(1,1,1,2))+
    theme(legend.position="none")
```



5.2 File Formats

5.2.1 Loading data from individual data files

The following code snippet assumes analysis data for each sample in a separate csv file with a filename ending in "_preferred.csv", and in each data file, the zircon U-Pb ages of interest in a column called"preferred age" (the space in the header will be automatically converted into a dot upon import, so the column in the resulting data frame is called "preferred.age"). The below code deals with a very simple case, obviously loading individual data files can become arbitrarily copmlicated. The overall goal is to obtain a list object

containing one vector of the individual ages per sample; the steps to arrive at this data object will vary for each lab and user.

```
##### Loading data #####

## path to the data - change according to where you saved it
basepath<-"/home/user/path/to/data"

## find all ..._preferred.csv files in this folder, load them into a data object
## called 'data':
# list all .._preferred.csv files
fls<-list.files(path=basepath, pattern=".*_preferred.csv", full.names=TRUE)
# prepare data object
data<-list()
# load the individual data files
for(f in fls){
    # extract sample names from filename
    spl<-sub("_preferred.csv", "", basename(f))
    # load data
    cdat<-read.table(file=f, header=TRUE, sep=",", stringsAsFactors=FALSE, skip=6)
    # skip lines without a 'preferred age'
    cdat<-cdat[!is.na(cdat$preferred.age),]
    # copy the remaining ages into the 'data' object
    data[[spl]]<-cdat$preferred.age
}
```

5.2.2 Loading data from MS Excel files

Below is a code snippet used to load data from an MS Excel file, assuming the first sheet in the file contains the ages of interest one column per sample, with column headers (same data structure as the "zircon_data.csv" file provided with this package). This requires the `gdata` package to be installed, which in turn needs a working perl installation. On Windows, install Strawberry Perl before installing `gdata`. Linux and MacOS come with perl installed.

```
##### Loading data #####

#need gdata package to read Excel files
require(gdata)
#input file
inpath="...wherever_your_file_is/filename.xls"

## read xls file - this assumes data to be in the first sheet, with headers
## (sample names) in the first line
data<-read.xls(inpath, stringsAsFactors=FALSE)
#the columns aren't all the same length - cut out empty values
data<-as.list(alldata)
for(i in 1:length(data)){data[[i]]<-data[[i]][!is.na(data[[i]])]}
```

The second example is somewhat a combination of the two previous snippets, in that it assumes the samples in individual worksheets within one Excel file. The structure of each sheet is assumed to be the same as the individual "_preferred.csv" files mentioned above.

```
##### Loading data #####

#need gdata package to read Excel files
require(gdata)
#input file
inpath="...wherever_your_file_is/filename.xls"

#names of worksheets in xls file - assumed to be sample names
sheets<-sheetNames(inpath)
# prepare data object
data<-list()
# load the individual data files
for(smpl in sheets){
    # load data
    cdat<-read.xls(xls=inpath, sheet=smpl, header=TRUE, sep=",",
        stringsAsFactors=FALSE, skip=6)
    # skip lines without a 'preferred age'
    cdat<-cdat[!is.na(cdat$preferred.age),]
    # copy the remaining ages into the 'data' object
    data[[smpl]]<-cdat$preferred.age
}
```