Cedhar Documentation

Release 0.0.6

JARO

Dec 02, 2020

Contents:

1 Digital Humanities

The use of digital technologies to pursue research questions in the humanities.

1.1 Data for humanities

1) Document markup languages
    - **ConTeXt** a TeX macro package that has a cleaner interface to control typography of the document while retaining LaTeX's structure-oriented approach
        - with separation of content and presentation, it can format XML text, ...
    - **EAD** (*Encoded Archival Description*)
    - ...
2) Citation
    - *Arts and Humanities Citation Index* (AHCI)
        - Machine-readable bibliographic record - MARC, RIS, BibTeX
3) Geospatial/geographical data
    - **GeoJSON** is a geospatial data interchange format based on *JavaScript Object Notation* (JSON).
    - **Leaflet** is an open-source JavaScript library for mobile-friendly, cross-browser, interactive maps.
    - A Web Map Service (**WMS**) is a standard protocol for serving georeferenced map images over the Internet that are generated by a map server using data from a GIS database.
        - See also Web Feature Service (**WFS**)

1.2 Data formats

Data can be stored in different formats.

JSON structure

JSON stands for JavaScript Object Notation and it is based on the JavaScript Programming Language Standard ECMA-262.

JSON is built on two structures, namely a collection of name/value pairs, and an ordered list of values. A JSON structure looks like:

```
Object {
        Identifier: Value
        Identifier: Array [
           Object {
                   Identifier: Value
             }
        ]
}
```

Where an `Identifier` is delimited by quotes, and a `Value` can be a `string`, a `number`, `"true"`, `"false"`, `"null"`, or an `Array` or another JSON `Object` as the above example.

JSON in R

Some R packages for reading JSON files in CRAN are

- `rjson` v0.1.0 released on Jul 30 2007
- `RJSONIO` v0.3-1 released on Oct 4 2010
- `jsonlite` v0.9.0 released on Dec 3 2013

eXtensible Markup Language

Todo: eXtensible markup language (XML) structure

Lightweight markup languages

Lightweight markup languages are for producing documentation on the Web.

Markdown

Markdown (MD), with suffixes `.md`, `.Rmd`, etc., is currently the markup language for GitHub, and hence very popular among developers using this platform. The popularity of this format for writing for the web is however challenging its consistency and robustness, and today there are several flavours of MD:

- Basics and syntax of the *"Gruber Markdown"* are in the creator's webpage
- *CommonMark* is an extension of the Gruber Markdown by users including representatives from GitHub, Stack Exchange, and Reddit, and therefore today "de facto" standard on the Web.
- *Github Flavored Markdown* or *GFM* is a superset of CommonMark with Github-specific extensions on syntax features.

- Other flavours of Markdown include *MultiMarkdown*, *Markdown Extra*, *CriticMarkup*, *Ghost Markdown*, and others...

reStructuredText

reStructuredText (RST) is written with the suffix `.rst` or `.txt` since is plaintext, which use simple and intuitive constructs to structure complex technical documentation. Here "complex" means things like indexing, glossaries, etc.

One significant innovation of Markdown was the use of headers and interpreted text. However, a step further of RST over MD is the use of *directives* and *specialized roles*. For example, these features allow reStructuredText rendering text and math formulae directly into LaTeX format.

The directive syntax in RST is

```
.. directive-type :: directive
block
```

and an illustration of a standard and specialized role is

```
*emphasis* as standard role
:title:*emphasis* with explicit role
```

where (most) of standard roles are common for interpreted text in MD and RST.

In order to produce a documentation, either in HTML or in LaTeX, reStructuredText needs a *builder*, which is a program that convert the RST source code into the desired format.

Popular builders are the `Python` package `docutils` with different options:

```
prompt> ./rst2html.py text.rst > text.html
prompt> ./rst2latex.py text.rst > text.tex
```

where RST sources are in a *source* folder and constructs go into a *build* folder.

Another alternative is `Sphinx` that constructs the API documentation with the two folders and perform the transformation afterwards.

```
prompt> ./sphinx-build [options] html source build
prompt> ./sphinx-build [options] latex source build
```

TeX and LaTeX

Todo: TBD

Another data format

Todo: TBD

2 User Acceptance Testing

User Acceptance Test or [UTA] is a type of testing performed by the Client to certify the system with respect to the requirements that was agreed upon. This report is to illustrate the application of [UTA] in developing, testing, and releasing a new or updated R package.

2.1 User Acceptance Test for R package release

In the DTAP process (development → test → acceptance → production), which is part of a software development process, the release of a new version of an R package is at the final step where the deployment needs to meet certain requirements. The "acceptance" portion in this process comes from both *humans* (developer and clients) and *machines* (a list of rules that the software must conform). Of course, human testing involves machines as well, and this distinction is made for referring to a systematic way of testing.

Human testing

The development of a software component on a local machine has folders containing a set of functions in a structure like this

```
FOLDER-BUILD
FOLDER-PUB
Batch file to build
```

where the testing of the software occurs in the three parts.

A set of R functions are written in *FOLDER-BUILD* with a version control, and then the functions are tested manually in the R console or other environment. It is important at this stage that the test involves random data, different data sets, and the extreme cases that the input data may have like zeroes, empty arrays, etc. Once there is an acceptance from the developer, *FOLDER-PUB* hosts a copy of the functions to publish and then the building with the *batch file*.

The batch file has the instructions in a sequential list to build the package that typically are R core functions to load files, write or recreate an object to a file, and the creation of a skeleton for a new source package.

Once the package is built then comes the documentation of the functions with files in a Latex format that conform the manual. The machine testing, which comes afterwards, will check among other things whether there is a correspondence or not between the script codes and the documentation.

Machine testing

In the MS Windows operating system, Rtools allows performing a machine testing of `R` packages. First the package is constructed as a tarball file with the command line and by typing `R CMD build`. Then the `R CMD check` performs different types of tests on this file based on pass/fail results where "fail" involves errors, warnings, and notes. The option `--as-cran` applies the most strict rules on the package and allows a successful submission for a publication on the CRAN repository whenever there is any fail.

An example of a successful testing is in the `.log` file given as appendix where the checking starts in the `DESCRIPTION` file with the basic information about the package. Then there is also a testing of things like whether the package can be loaded and unloaded, and consistencies both in the code and the documentation including meta data.

UAT for users in GitHub

As with machine testing, the acceptance tests among users of the `R` package should reveal as well a straightforward yes/no or pass/fail results. This means that the user should be able to download, install, uninstal, and run the software without any errors.

Another important component of the user acceptance is counting with a shared infrastructure that team members administer, use on a day-to-day basis and can reflect on and implement for others. GitHub is not only a git repository, but it is also a tool suitable for educational tasks such as periodical reflection and implementation for others.

GitHub storages the information and it can be used as well as backup for disaster recovery. In many cases GitHub is the place where the users report bugs, and where ideally users should become developers of the package as well.

UAT exercise

1) Install the beta version of the package from GitHub plus dependencies into the R environment
2) Load the package and run the scripts of the *README.md* file
3) Run the program with different datasets
   - if an error occurs then consult the manual to fix the input data
   - if the data introduced conforms the requirements from the manual and the error remains then report the bug

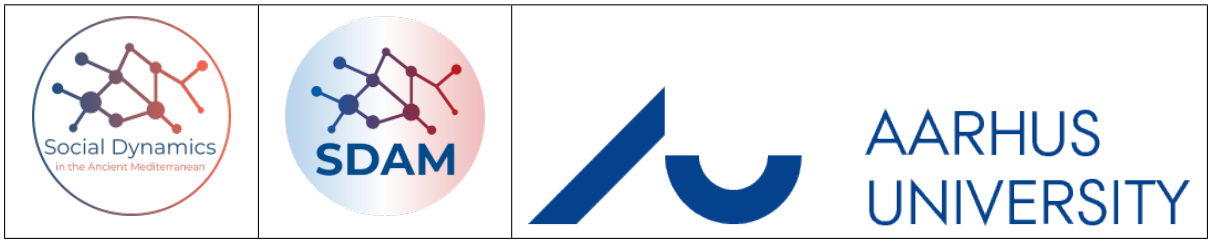
A successful user acceptance test has no errors, and the user is willing to use the package in his/her own work.

Todo: UAT for data science and ML engineering

---

3 The SDAM project

SDAM stands for **Social Dynamics and complexity in the Ancient Mediterranean**, which is a research group at the School of Culture and Society at Aarhus University (AU).



3.1 Objectives

The two mains outputs of this project are expected to be

- A comparative study of proxies for evolution of social complexity in the Ancient Mediterranean.
- Digital tools, workflows and processes that scale and that historians and archaeologists can use in their own research.
- See SDAM Website
- See SDAM on GitHub

4 R package `sdam`

4.1 Package Description

The R package "sdam" provides tools for performing analyses within Social Dynamics and complexity in the Ancient Mediterranean (SDAM) project.

- (See "sdam" manual)

Currently, it is possible with `sdam` to access data from the Epigraphic Database Heidelberg API with `get.edh()`, and the wrapper function `get.edhw()` as well. Most of the data is available in the dataset attached to the package, which is called `EDH`.

For applications of these functions and use of this dataset:

- (See *Epigraphic Database Heidelberg*)
- (See *Epigraphic Networks using R*)

Besides, the `request()` function allows performing different types of HTTP requests to a cloud repository like DEiC'S https://sciencedata.dk or another customized URL address.

- (See *DEiC's sciencedata.dk* for applications of this function.)

Similarity by simple matching among column vectors is achieved by the `simil()` function in order to make analyses of assemblages or artifacts. Note that this latter function still under early development.

- (See *Epigraphic Networks using R* for applications of this function.)

  See also:

  Package `multigraph`. Versions: [CRAN], [GitHub].

  Package `multiplex`. Versions: [CRAN], [GitHub].

4.2 Package installation

You can install the `sdam` package from these GitHub repositories using the `R` console or RStudio if you wish.

```
# install beta version
R> devtools::install_github("mplex/cedhar", subdir="pkg/sdam")
```

or

```
# install release candidate
R> devtools::install_github("sdam-au/sdam")
```

4.3 Assembly

Todo: Package assembly workflow

4.4 Making Documentation

Todo: Making Documentation workflow

4.5 News from `sdam` package

- **Version 0.3.0 (beta) released (21-11-2020).**
  - *New functions* `plot.dates()` and `prex()` (cf.)
  - `EDH` dataset is updated
  - Function `edhw()` combines `"people"` with other EDH variables
  - Argument *select* added to `edhw()`
  - Argument *maxlimit* added to `get.edh()`

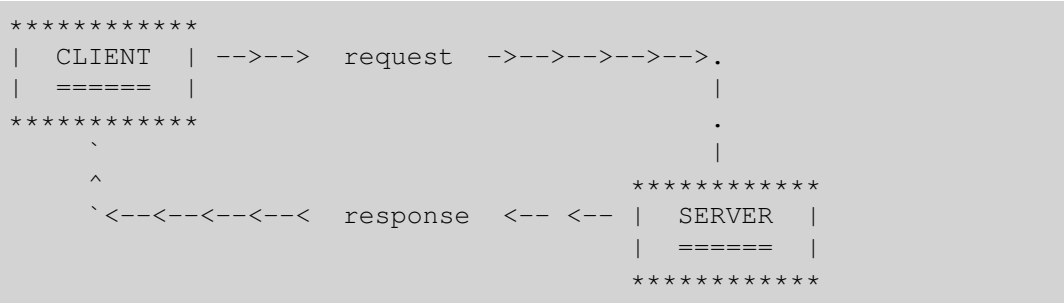
  - Argument *authenticate* in `request()` renamed to *anonymous*

- **Version 0.2.0 released (19-5-2020).**
  - *New function* `edhw()` to extract fragments of the `EDH` dataset
  - Documentation website launched
  - Arguments *force* and *rm.file* added to `request()`
- **Version 0.1.0 released (6-5-2020).**
  - *First functions* `get.edh()`, `get.edhw()`, `request()`, `simil()`, and `EDH` *dataset*

Todo: Connect with GitHub `sdam-au\sdam\docs`

5 DEiC's sciencedata.dk

Accessing DEiC's (Danish e-Infrastructure Cooperation) sciencedata.dk needs another tools, and this is basically achieved by performing a HTTP request as a **client** to one of DEiC's **server**.



The server's response depends on the method used in the HTTP request.

5.1 Accessing DEiC's sciencedata.dk using R

Function `request()` from the R package "sdam" is aimed to interact with DEiC's sciencedata.dk

Note that this function requires the [R] package `"httr"`.

Functions Usage

request()

```
# arguments supported (currently)
R> request(file
           ,URL="https://sciencedata.dk"
           ,method=c("GET","POST","PUT","DELETE")
           ,anonymous=FALSE
           ,path="/files"
           ,cred=NULL
           ,subdomain=NULL
           ,...)
```

Parameters

- *file* (object under 'method')
- *URL* (protocol and domain of the url)
- *method* (the http "verb" for the object)

  "GET" (list)

  "POST" (place)

  "PUT" (update)

  "DELETE" (cancel)
- *anonymous* (logical, unauthenticated user?)
- *path* (optional path or subdirectory to add to the url)

Additional parameters

- *cred* (authentication credentials, vector with username and password)
- *subdomain* (optional, add subdomain to the url)
- ... (extra parameters if required)

Arguments

Arguments of request() are retrieved with the formals() function.

```
R> formals(request)
#$file
#
#
#$URL
#[1] "https://sciencedata.dk"
#
```

(continues on next page)

(continued from previous page)

```
#$method
#c("GET", "POST", "PUT", "DELETE")
#
#$anonymous
#[1] FALSE
#
#$cred
#NULL
#
#$path
#[1] "/files"
#
#$subdomain
#NULL
```

Note: Aliases for `request()` are `sddk()` and `SDDK()`.

Output

The output is the server's response that depends on the method to be used in the request.

A `Response` message is returned when the method is `PUT` with the url and items `Date`, `Status`, `Content-Type`.

Details

There are two types of folders in DEiC's sciencedata.dk that are *personal* and *shared* folders and both requires authentication with credentials.

The *path* to the shared folders where the files are located must be specified with the `path` argument. However, for personal folders is the `file` argument that includes the path information.

That is, an [R] code will be like

```
# personal folders
R> request("path/file")

# shared folders
R> request("file", path="/path")
```

Many times, DEiC's sciencedata.dk places the data on a *subdomain*, and for some request methods like `PUT` it is needed to specify the subdomain as well.

5.2 Authentification

In case that accessing the server requires basic authentification, then package `"tcltk"` may be needed as well to input the credentials with a widget prompt. `request()` has the `cred` argument for performing a basic authentification.

In DEiC's sciencedata.dk, both personal and shared folders need some sort of authentication. With the basic authentication, the credentials are given *with the username and password used under your personal* 'sciencedata.dk' **settings**.

Hint:

It is possible to prevent the widget by recording this information in a vector object. If you want to avoid a dialog box then save your credentials.

```
# save authentication credentials
R> mycred <- c("YOUR-AUID@au.dk", "YOURPASSWORD")
```

However, in many cases such as with public folders in sciencedata.dk authentification is not needed and you can disable it by setting *anonymous* to `TRUE`.

5.3 Responses

Server responses carry a code called *HTTP status code* where `2xx` means **success**, and `4xx` means **client error**. There is also a status code like `5xx` for server error, and `3xx` for redirection (and where codes `1xx` are just informative).

Todo: Typical staus codes in the response are `404`, `201`, `307`...

When using the `request()` function, the HTTP status code is given under `Status` in the response message below the time stamp.

5.4 Examples

Some examples of HTTP requests are given next where reponse messages in some cases are given afterwards, and recall that `request()` requires the `httr` package.

```
# load required package
R> require("httr")   # https://cran.r-project.org/package=httr
```

Method GET

This method is for accessing the files with the data.

```
# for personal data (in case you have this file)
R> request("df.json", cred=mycred)

#[1] {"a":{"0":"a1","1":"a2"},"b":{"0":"b1","1":"b2"},"c":{"0":"c1","1":"c2"}}
```

```
# for shared folders (example Vojtech test folder), where both options work
R> request("df.json", path="/sharingin/648597@au.dk/TEST_shared_folder/", method="GET",

#[1] {"a":{"0":"a1","1":"a2"},"b":{"0":"b1","1":"b2"},"c":{"0":"c1","1":"c2"}}
```

> **Note:** If there is any error, then the HTTP status code with the `GET` method is `200` or OK but it is not returned.

Method PUT

The URL typically includes also a *subdomain* that for DEiC's sciencedata.dk is named `silo` followed by a number. For instance, my personal documents are located in `silo1.sciencedata.dk`, and other users that will follow are probably located at `silo2`, etc.

`PUT` in own folder

For method `PUT`, the subdomain is mandatory; otherwise the request is redirected.

```
# for personal data (in my case) I need to specify the subdomain; otherwise it gets redi
R> request(system.file("CITATION"), method="PUT", cred=mycred)

# Response [https://sciencedata.dk/files/CITATION]
#   Date: 2020-01-17 13:31
#   Status: 307
#   Content-Type: text/html; charset=UTF-8
#<EMPTY BODY>
```

The HTTP status code `307` means temporary redirect.

```
# my data is in subdomain "silo1"
R> request(system.file("CITATION"), method="PUT", cred=mycred, subdomain="silo1")

# Response [https://silo1.sciencedata.dk/files/CITATION]
#   Date: 2020-01-17 13:31
#   Status: 201
#   Content-Type: text/html; charset=UTF-8
#<EMPTY BODY>
```

The HTTP status code `201` means that the file was created in the server side.

`PUT` in a sharing folder

```
# (example Vojtech test folder)

R> request(system.file("CITATION"), path="sharingin/648597@au.dk/TEST_shared_folder",
+    method="PUT", cred=mycred)

# Response [https://sciencedata.dk/sharingin/648597@au.dk/TEST_shared_folder/CITATION]
#  Date: 2020-01-17 13:34
#  Status: 307
#  Content-Type: text/html; charset=UTF-8
#<EMPTY BODY>



R> request(system.file("CITATION"), path="sharingout/648597@au.dk/TEST_shared_folder",
+    method="PUT", cred=mycred)

#Response [https://sciencedata.dk/sharingout/648597%40au.dk/TEST_shared_folder//CITATION
#  Date: 2020-02-10 09:32
#  Status: 201
#  Content-Type: text/html; charset=UTF-8
#<EMPTY BODY>
```

Hence, the `PUT` method for a shared folder needs `'sharingout'` in the path; otherwise it gets redirected.

> **Note:** In some cases, the metacharacter `@` in the path is "escaped" as `%40`.

Method DELETE

In the case of accesing with a request using methods `GET` or `PUT`, the path in the url is followed by `sharingin/USERID/FOLDERNAME`, and for `DELETE` the *response* is given with `sharingout` in the path.

```
# for personal folder
R> request("df.json", method="DELETE", cred=mycred)

# In my case, this is in
#[1] "https://silo1.sciencedata.dk/files/df.json"
```

```
# for shared folders (example Vojtech test folder)
R> request("CITATION", path="/sharingin/648597@au.dk/TEST_shared_folder/", method="DELE

#[[1]]
#[1] "https://sciencedata.dk/sharingout/648597%40au.dk/TEST_shared_folder/CITATION"
```

Method POST

Finally, there is also the possibility to *place* files with the `POST` method along with extra information.

```
R> request(FILE, URL, method="POST")
```

Typically with a `path` argument and `subdomain` if required.

> **Note:** Method `POST` is not yet implemented in sciencedata.dk

6 Epigraphic Database Heidelberg

This post is about accessing the "Epigraphic Database Heidelberg" (EDH), which is one of the longest running database projects in digital Latin epigraphy. The [EDH] database started as early as year 1986, and in 1997 the Epigraphic Database Heidelberg website was launched at https:/edh-www.adw.uni-heidelberg.de where inscriptions, images, bibliographic and geographic records can be searched and browsed online.

6.1 Open Data Repository

Despite the possibility of accessing the [EDH] database through a Web browser, it is many times convenient to get the Open Data Repository by the [EDH] through its public Application Programming Interface (API).

For inscriptions, the generic search pattern Uniform Resource Identifier (URI) is:

```
https://edh-www.adw.uni-heidelberg.de/data/api/inscriptions/search?par_1=value&par_2=val
```

with parameters *par* $1, 2, ...n$.

6.2 Response

The response from a query is in a Java Script Object Notation or JSON format such as:

```
{
   "total" : 61,
   "limit" : "20",
   "items" : [ ... ]
}
```

- (see *"JSON structure" in Digital Humanities*)

In this case, `"items"` has an array as a value where the returned records are located. The `"total"` and `"limit"` values correspond to the *total* number of records of the query, and the *limit* number is the amount of records to appear in the browser after the query.

6.3 Accessing the EDH database using R

Accessing the [EDH] database [API] using R is possible with a convenient function that produces the generic search pattern [URI]. Hence, the function `get.edh()` from the `sdam` package allows having access to the data with the available parameters that are recorded as arguments. Then the returned [JSON] file is converted into a list data object with function `fromJSON()` from the `rjson` package.

Currently, function `get.edh()` allows getting data with the `search` parameter either from `"inscriptions"` (the default option) or else from `"geography"`. The other two search options from the [EDH] database [API], which are `"photos"` and `"bibliography"`, may be implemented in the future in this function.

- (see *R package "sdam"*)

Function usage

`get.edh()`

```
# arguments supported
R> get.edh(search = c("inscriptions", "geography")
           , url = "https://edh-www.adw.uni-heidelberg.de/data/api"
           , hd_nr, province, country, findspot_modern
           , findspot_ancient, year_not_before, year_not_after
           , tm_nr, transcription, type, bbox, findspot, pleiades_id
           , geonames_id, offset, limit, maxlimit=4000, addID, printQ)
```

> **Note:** "R>" at the beginning of the line means that the following code is written in R. Comments are preceded by "#".

Search parameters

The following parameter description is from the [EDH] database [API]

Inscriptions and Geography

- *province:*

  get list of valid values at province terms, in the [EDH] database [API], case insensitive

- *country:*

  get list of valid values at country terms in the [EDH] database [API], case insensitive

- *findspot_modern:*

add leading and/or trailing truncation by asterisk *, e.g. `findspot\_modern=köln\*`, case insensitive

- *findspot_ancient:*

  add leading and/or trailing truncation by asterisk *, e.g. `findspot\_ancient=aquae\*`, case insensitive

- *offset:*

  clause to specify which row to start from retrieving data, integer

- *limit:*

  clause to limit the number of results, integer (by default includes all records)

- *bbox:*

  bounding box with the format `bbox=minLong, minLat, maxLong, maxLat`.

  The query example:

```
https://edh-www.adw.uni-heidelberg.de/data/api/inscriptions/search?bbox=11,47,12,48
```

that in [R] is a vector character.

Hint: Just make sure to quote the arguments in `get.edh()` for the different parameters that are not integers. This means for example that the query for the last parameter with the two search options is written as

```
R> get.edh(search="inscriptions", bbox="11,47,12,48")
R> get.edh(search="geography", bbox="11,47,12,48")
```

Inscriptions only

- *hd_nr:*

  HD-No of inscription

- *year_not_before:*

  integer, BC years are negative integers

- *year_not_after:*

  integer, BC years are negative integers

- *tm_nr:*

  Trismegistos database number (?)

- *transcription:*

  automatic leading and trailing truncation, brackets are ignored

- *type:*

  of inscription, get list of values at terms type in the [EDH] database [API], case insensitive

Geography only

- *findspot:*

  level of village, street etc.; add leading and/or trailing truncation by asterisk *, e.g. `findspot\_modern=köln\*`, case insensitive

- *pleiades_id:*

  Pleiades identifier of a place; integer value

- *geonames_id:*

  Geonames identifier of a place; integer value

Extra parameters

- *maxlimit:*

  Maximum limit of the query; integer, default 4000

- *addID:*

  Add identification to the output?

- *printQ:*

  Print also the query?

The two functions we have seen so far, `get.edh()` and `get.edhw()`, are available in the *R package "sdam"*.

See also:

- *"sdam" package installation*.

6.4 Examples

The examples are made with the `sdam` R package.

Since the `get.edh()` function needs to transform JSON output using `rjson::fromJSON()`, you need to have this package installed as well.

Then, to run the examples you need to load the required libraries.

```
R> library("sdam")
R> require("rjson")   # https://cran.r-project.org/package=rjson
```

The query

```
R> get.edh(findspot_modern="madrid")
```

returns this truncated output:

```
#$ID
#[1] "041220"
#
#$commentary
#[1] " Verschollen. Mögliche Datierung: 99-100."
#
#$country
#[1] "Spain"
#
#$diplomatic_text
#[1] "[ ] / [ ] / [ ] / GER PO[ ]TIF / [ ] / [ ] / [ ] / ["
#
#...
#
#$findspot_modern
#[1] "Madrid"
#
#$id
#[1] "HD041220"
#
#$language
#[1] "Latin"
#
#...
#
```

With `"inscriptions"`, which is the default option of `get.edh()` and of the wrapper function `get.edhw()`, the `id` "component" of the output list has not a numeric format. However, many times is convenient to have a numerical identifier in each record, and function `get.edh()` adds an `ID` with a numerical format at the beginning of the list.

Having a numerical identifier is useful for plotting the results, for example, and an `ID` is added to the output by default. You can prevent such addition by disabling argument `addID` with `FALSE`.

```
R> get.edh(findspot_modern="madrid", addID=FALSE)
```

Further extensions to the [EDH] database [API] may be added in the future, and this will be handled with similar arguments in the `get.edh()` function ...

6.5 Accessing Epigraphic Database Heidelberg: Inscriptions

`get.edhw()`

```
# to perform several queries
R> get.edhw(hd_nr, ...)
```

To study temporary uncertainty, for example, we need to access to an epigraphic database like the Heidelberg. The wrapper function `get.edhw()` allows multiple queries by using the Heidelberg number `hd_nr`.

`get.edhw()` is a wrapper function to perform several queries from the Epigraphic Database Heidelberg API using identification numbers.

> **Note:** Currently, function `get.edhw()` works only for inscriptions.

```
# get data API from EDH with a wrapper function
R> EDH <- get.edhw(hd_nr=1:83821)   # (03-11-2020)

R> length(EDH)
#[1] 83821

# or load it from the package
R> data("EDH")
```

This wrapper function basically perform the following loop that will produce a list object with the existing entries for each inscription, and where entries have different length.

```
# grab the data from EDH API and record it in 'EDH'
R> EDH <- list()
# 82464 INSCRIPTIONS (20-11-2019)
R> for(i in seq_len(82464)) {
+    EDH[[length(EDH)+1L]] <- try(get.edh(hd_nr=i))
+    }
```

Beware that retrieving such a large number of records will take a very long time, and this can be done by parts and then collate the lists into the `EDH` object.

> **Note:** Character + in the code shows the scope of the loop.

Output

The output depends on each particular case.

```
R> is(EDH)
#[1] "list"   "vector"
```

The first record has 28 *attribute* names

```
# check variable names of first entry
R> attr(EDH[[1]], "names")
# [1] "ID"                    "commentary"            "country"
# [4] "depth"                 "diplomatic_text"       "edh_geography_uri"
# [7] "findspot_ancient"      "findspot_modern"       "height"
#[10] "id"                    "language"              "last_update"
#[13] "letter_size"           "literature"            "material"
#[16] "modern_region"         "not_after"             "not_before"
#[19] "people"                "province_label"        "responsible_individual"
#[22] "transcription"         "trismegistos_uri"      "type_of_inscription"
#[25] "type_of_monument"      "uri"                   "width"
#[28] "work_status"
```

While record 21 has 34 items.

```
R> attr(EDH[[21]], "names")
# [1] "ID"                          "commentary"
# [3] "country"                     "depth"
# [5] "diplomatic_text"             "edh_geography_uri"
# [7] "findspot"                    "findspot_ancient"
# [9] "findspot_modern"             "geography"
#[11] "height"                      "id"
#[13] "language"                    "last_update"
#[15] "letter_size"                 "literature"
#[17] "material"                    "military"
#[19] "modern_region"               "not_after"
#[21] "not_before"                  "people"
#[23] "present_location"            "province_label"
#[25] "responsible_individual"      "social_economic_legal_history"
#[27] "transcription"               "trismegistos_uri"
#[29] "type_of_inscription"         "type_of_monument"
#[31] "uri"                         "width"
#[33] "work_status"                 "year_of_find"
```

Attribute `people` is another list with other *attribute* names

```
R> length(EDH[[1]]$people)
#[1] 3

R> attr(EDH[[1]]$people[[1]], "names")
#[1] "name"      "gender"    "nomen"     "person_id" "cognomen"

...

R> attr(EDH[[1]]$people[[3]], "names")
[1] "cognomen"  "praenomen" "person_id" "gender"    "name"      "nomen"
```

- (see *attributes in EDH dataset*)

7 Extracting EDH Variables

7.1 EDH variables

Another wrapper function, this time for the extraction of variables from the EDH dataset is found in edhw().

edhw()

Function usage

```
# accepted parameter arguments
R> edhw(vars, x = NULL, as = c("list", "df"), type = c("long", "wide", "narrow"),
        split, select, addID, limit, id, na.rm)
```

Parameters

Formal arguments of edhw() are:

- *vars:*

  Chosen variables from the EDH dataset (vector)

- *x:*

  An optional list object name with fragments of the EDH dataset

- *as:*

  Format to return the output. Currently either as a "list" or a data frame "df" object.

- *type:*

  Format of the data frame output. Currently either a "long" or a "wide" table (option "narrow" not yet implemented).

- ***split:*** Divide the data into groups by *id*? (logical and optional)

- ***select:*** "people" variables to select (vector and optional, data frame *type* "long" only)

- *addID:*

  Add identification to the output? (optional and logical)

- *limit:*

  Limit the returned output. Ignored if *id* is specified (optional, integer or vector)

- *id:*

  Select only the hd_nr *id(s)* (optional, integer or character)

- *na.rm:*

  Remove entries with <NA> information? (logical and optional)

Todo: Implement the "narrow" *type* option to edhw().

7.2 Attributes in EDH dataset

The aim of the edhw() function is to extract attributes or variables of inscriptions. These inscriptions are output values typically produced by the get.edh() or get.edhw() functions.

The records in the EDH dataset have at least one the following items:

"commentary"	"geography"	"religion"
"fotos"	"height" "id"	"province_label"
"country"	"language"	"responsible_individual"
"depth"	"last_update"	"social_economic_legal_history"
"diplomatic_text"	"letter_size"	"transcription"
"edh_geography_uri"	"literature"	"trismegistos_uri"
"findspot"	"material"	"type_of_inscription"
"findspot_ancient"	"military"	"type_of_monument" "uri"
"findspot_modern"	"modern_region"	"width" "work_status"
	"not_after"	"year_of_find"
	"not_before"	
	"present_location"	

Another output variable is "people" that is a list of persons named in the inscriptions with at least the following items

"person_id" "nomen" "cognomen" "praenomen" "name" "gender" "status" "tribus"	"origo" "occupation" "age: years" "age: months" "age: days"

7.3 Example: Relative dating in EDH

We are going to apply the edhw() function to check the relative dating of Roman inscriptions, and for a simple relative dating analysis, we choose chronological data variables in vars:

```
# make a list for relative variables in 'EDH' (default)
R> edhw(vars=c("not_after", "not_before"))
```

In this case, the boundaries of the time span of existence TPQ and TAQ are variables "not_after" and "not_before", respectively.

- (see *Boundaries of existence* for the meaning of TAQ and TPQ.)

Since argument *x* is not specified in the function, the "EDH" dataset in the `sdam` package is taken if available with a *Warning* message.

Hint: The above use of function `edhw()` is wrapping the `base lapply` function as

```
# recursively apply a function over the list for  variables
R> lapply(EDH, `[`, c("not_after", "not_before") )
```

where a pair of backquotes (aka "backticks") is a way to refer in `R` to names or combinations of symbols that are otherwise reserved or illegal, or non-syntactic names. Hence, e.g. `apply(foo, `[`, c(...) )` is the same as `apply(foo, function (x) x[c(...)])`.

The structure of such chronological data items is a list object with an `id` for all entries that is the `EDH hd_nr`.

```
R> str(edhw(vars=c("not_after", "not_before")))
#List of 83821
# $ :List of 3
#  ..$ id        : chr "HD000001"
#  ..$ not_after : chr "0130"
#  ..$ not_before: chr "0071"
# $ :List of 3
#  ..$ id        : chr "HD000002"
#  ..$ not_after : chr "0200"
#  ..$ not_before: chr "0051"
# ...
```

Complete cases

By default, function `edhw()` do not remove missing data when present in all variables, but is possible to remove missing information by activating the *na.rm* argument.

```
# remove missing data
R> str(edhw(vars=c("not_after", "not_before"), na.rm=TRUE))
#List of 60224
# $ :List of 3
#  ..$ id        : chr "HD000001"
#  ..$ not_after : chr "0130"
#  ..$ not_before: chr "0071"
# $ :List of 3
#  ..$ id        : chr "HD000002"
#  ..$ not_after : chr "0200"
#  ..$ not_before: chr "0051"
# ...
```

And work with complete cases.

See also:

Missing data within uncertainty.

8 Time and dating

Time is a (quantitative) continuous variable, but we often think *time* as a discrete variable since is rounded when measured (e.g. a person is 40 years old and not between 40 and 41).

8.1 Chronological periods of the Mediterranean Sea

We use chronological periods for dating purposes. Sometimes chronological periods are divided by phases and occasionally they are referred as chronological phases.

For example, *chronological phases* for dating ancient cultures of the Mediterranean Sea are

Chronological phase	Absolute dates (approx.)
Middle to Late Neolithic (MN-LN)	6000-4500 BC
Final Neolithic to Early Bronze 1 (FN-EB1)	4500-2700 BC
Early Bronze 2 (EB2)	2700-2200 BC
Late Prepalatial (LPrepal)	2200-1900 BC
First Palace (FPal)	1900-1700 BC
Second Palace (SPal)	1700-1450 BC
Third Palace (TPal)	1450-1200 BC
Post-Palatial to Protogeometric (PPalPg)	1200-900 BC
Geometric (Geo)	900-700 BC
Archaic (Arch)	700-500 BC
Classical (Class)	500-325 BC
Hellenistic (Hell)	325 BC - AC 0
Early Roman (ERom)	AC 0-200
Middle Roman (MRom)	AC 200-350
Late Roman (LRom)	AC 350-650
Early Byzantine (EByz)	AC 650-900
Middle Byzantine (MByz)	AC 900-1200
Early Venetian (EVen)	AC 1200-1400
Middle Venetian (MVen)	AC 1400-1600
Late Venetian (LVen)	AC 1600-1800
Recent (Recent)	AC 1800-present

(adapted from *Bevan et al*, 2013 (doi: 10.1111/j.1475-4754.2012.00674.x))

AC is sometimes ommited, and a negative number represents BC.

8.2 Relative dating of Roman inscriptions

Since the range of time in the relative dating of Roman inscriptions without outliers is between 530 BC and 950 AC, the chronological periods involving epigraphic Roman material are

Period	Dates years
Archaic (Arch)	-700 to -500
Classical (Class)	-500 to -325
Hellenistic (Hell)	-325 to 0
Early Roman (ERom)	0 to 200
Middle Roman (MRom)	200 to 350
Late Roman (LRom)	350 to 650
Early Byzantine (EByz)	650 to 900
Middle Byzantine (MByz)	900 to 1200

The eight chronological phases are reduced to five categories with their respective years:

Period	Dates years
Archaic (Arch)	-700 to -500
Classical (Class)	-500 to -325
Hellenistic (Hell)	-325 to 0
Roman (Rom)	0 to 650
Byzantine (Byz)	650 to 1200

Eight- and five chronological phases are for the analysis of temporal uncertainty of the `EDH` dataset.

Dates in data frames

To produce data frames, we need to make explicit with the *as* argument; in the example below with chronological data in the function where `"id"` is added by default.

```
# produce a data frame with chronological data variables and remove missing data
R> EDHdates <- edhw(vars=c("not_after", "not_before"), as="df")
```

The first entries are

```
# look at the first ones
R> head(EDHdates)
#        id not_before not_after
#1 HD000001       0071       0130
#2 HD000002       0051       0200
#3 HD000003       0131       0170
#4 HD000004       0151       0200
#5 HD000005       0001       0200
#6 HD000006       0071       0150
```

and the last entries

```
# look at the last ones
R> tail(EDHdates)
#             id not_before not_after
#83816 HD081504       0071      0130
#83817 HD081505       0071      0130
#83818 HD081506       0071      0130
#83819 HD081507       0101      0200
#83820 HD081508       0151      0230
#83821 HD081509       0151      0250
```

Time spans of existence

Object `EDHdates` has information about relative dating with boundaries of existence in variables `not_before` and `not_after`.

In `EDHdates`, however, these variables are factors and they need to be converted into vectors with a numeric format in order to represent the boundaries of existence.

```
# time columns in EDHdates are factors
R> is.factor(EDHdates$not_after)
R> is.factor(EDHdates$not_before)
#[1] TRUE

# boundaries of existence not_before and not_after
R> nb <- as.numeric(as.vector(EDHdates$not_before))
R> na <- as.numeric(as.vector(EDHdates$not_after))
```

To compute oldest and latest years with `min` and `max` functions, we also need to remove `NA` data from these vectors.

```
# oldest and latest dates
R> years <- c(min(nb, na.rm=TRUE), max(na, na.rm=TRUE))
#[1] -530 1998
```

Dates in years are between 530 BC and 1998 AC.

Plotting time intervals

To be able to plot efficiently time spans of existence of records in the `EDH` dataset, we need to count with a numerical identifier.

```
# get IDs by removing alphabetic characters in id
R> ID <- as.numeric(sub("[[:alpha:]]+","",EDHdates$id))
```

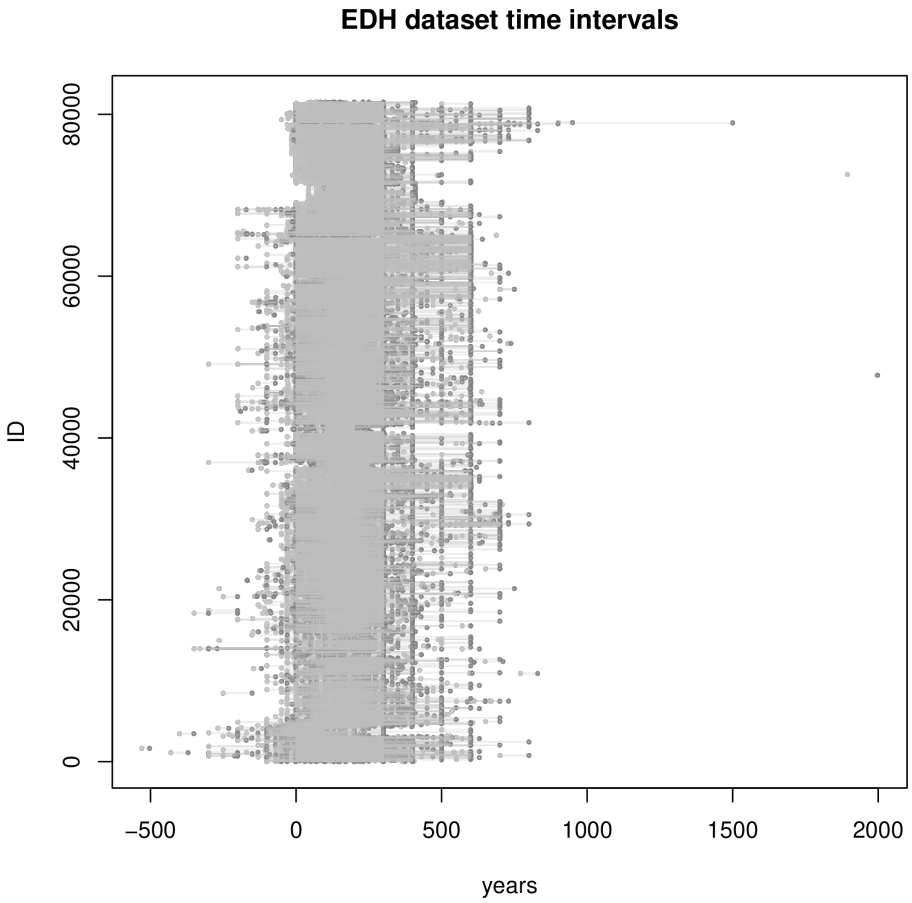
A plot of relative dating for Roman inscriptions with `ID` is made with the [R] `graphics` core package (or `base` package with R version 4.0.0).

```
# plot with graphics
R> plot(nb, ID, pch=20, col="#C0C0C0", xlab="Year", ylab="ID",
+       xlim=years, main="EDH dataset time intervals")
R> points(na, ID, pch=20, col="#808080")
R> segments(nb, ID, na, ID, col=grDevices::adjustcolor(8,alpha=.25))
```

or using the `plot.dates()` function from `sdam`

```
R> plot.dates(main="EDH dataset time intervals",
+              taq="not_before", tpq="not_after", cex=.5)
```

That produces:

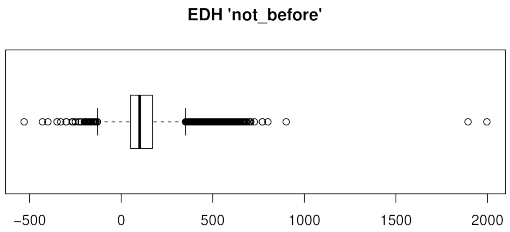


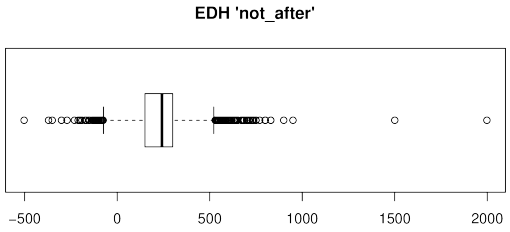
Treating Outliers

By looking at the box-and-whisker plots below, we can clearly see a couple of outliers at least in each category.

```
# par(mfrow =  c(2, 1))
R> boxplot(nb, horizontal=TRUE, main="EDH 'not_before'")
R> boxplot(na, horizontal=TRUE, main="EDH 'not_after'")
```

That produces:





The two most extreme outliers in each category are given in `$out` produced by the `boxplot()` function.

```
# first outlier is the maximum value of the dates
R> outliers<- c(tail(sort(boxplot(nb, plot=FALSE)$out),2),
+                tail(sort(boxplot(na, plot=FALSE)$out),2))
#[1] 1894 1997 1500 1998
```

We remove these outliers in *not_before* and *not_after*, and we update the `EDHdates` object.

```
R> c(nb[which(nb %in% outliers)], na[which(na %in% outliers)])
#[1] 1997 1894 1998 1500

# update by removing outliers in both categories
R> EDHdates <- EDHdates[-c(which(nb %in% outliers),which(na %in% outliers)), ]
```

Since rows are removed from `EDHdates`, we need to update object identifiers to compute the new the range of time.

```
# update values
R> ID <- as.numeric(sub("[[:alpha:]]+","",EDHdates$id))
R> nb <- as.numeric(as.vector(EDHdates$not_before))
R> na <- as.numeric(as.vector(EDHdates$not_after))

# new dates
R> years <- c(min(nb, na.rm=TRUE), max(na, na.rm=TRUE))
#[1] -530  950
```

Years are now between 530 BC and 950 AC. That is, from *Archaic* to *Middle Byzantine* chronological periods.

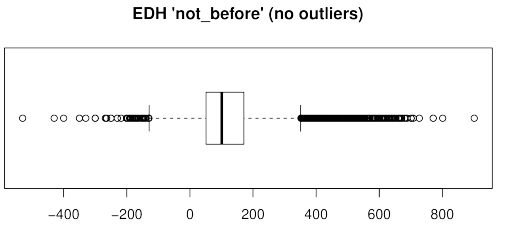
- (See: *Time and dating*)

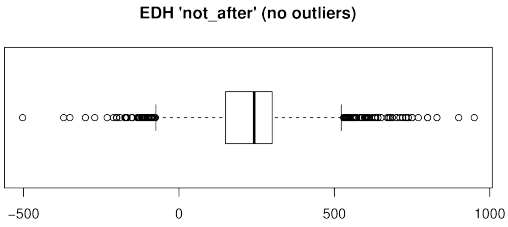
Plotting without outliers

Now we take a look at the box-and-whisker plots without outliers

```
# par(mfrow =  c(2, 1))
R> boxplot(nb, horizontal=TRUE, main="EDH 'not_before' (no outliers)")
R> boxplot(na, horizontal=TRUE, main="EDH 'not_after' (no outliers)")
```

that produces:

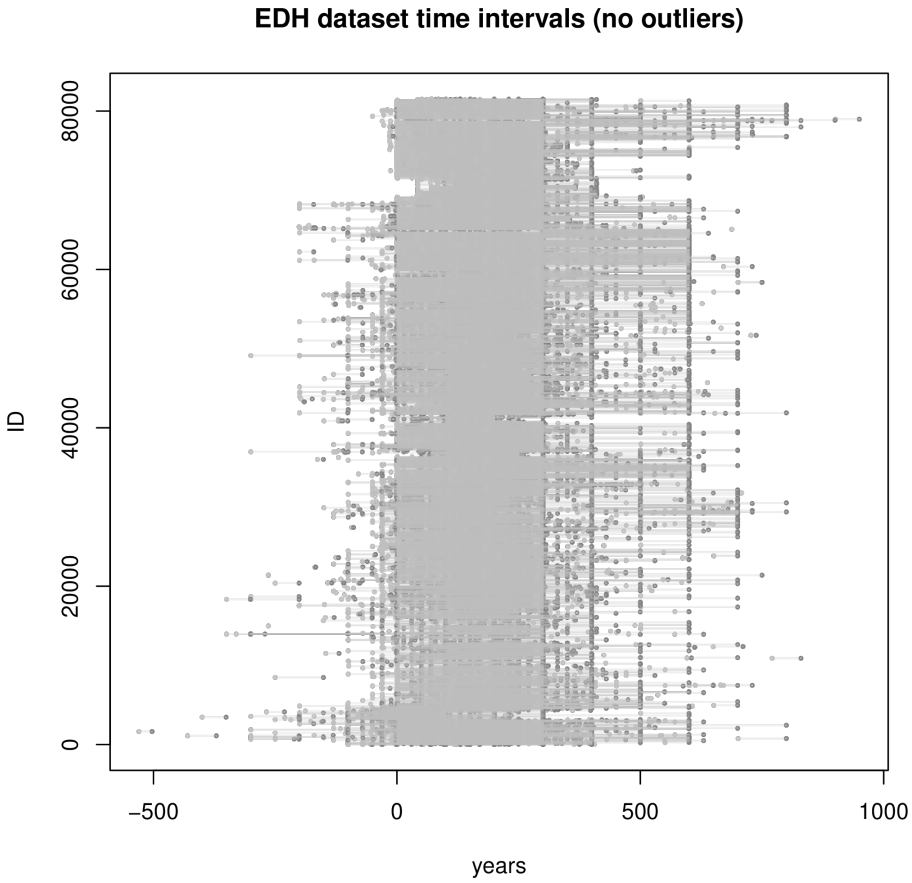




The next step is to plot relative dating without outliers and with updated range of time and with `plot.dates()` function.

```
R> plot.dates(main="EDH dataset time intervals (no outliers)",
+       taq="not_before", tpq="not_after", out=2, cex=.5)
```

that produces a plot with no outliers:



where we can see that most of the inscriptions in the `EDH` dataset are from Early to Middle Roman.

8.3 Plotting time intervals

To plot time intervals, we use function `plot.dates()`.

`plot.dates()`

Function usage

```
# use generic function
R> plot.dates(file=NULL, x=NULL, taq, tpq, out,
              main=NULL, xlab=NULL, ylab=NULL, xlim=NULL,
              pch, cex, col, lwd, lty, alpha, ... )
```

Parameters

Formal arguments of `plot.dates()` are:

- *file:*

  path to file for a PDF format (optional)

- *x:*

  data frame object of variables and observations. If `NULL` then `EDH` dataset is taken.

- *taq:*

  terminus ante quem

- *tpq:*

  terminus post quem

- *out:*

  number of outliers to omit (integer or vector where first entry id for latest date)

Optional arguments from the `graphics` package for the plot are

- *main:*

  main tile

- *xlab:*

  `x` label

- *ylab:*

  `y` label

- *xlim:*

  `x` limit

And for the representation of time interval and boundaries of existence in the plot

- *pch:*

  symbol for *taq* and *tpq*

- *cex:*

  size of *pch*

- *col:*

  colors of *pch* and time interval segment.

for time interval segments:

- *lwd:*

  width.

- *lty:*

  shape.

- *alpha:*

  alpha color transparency.

- ...

  additional parameters if needed

  See also:

  Computing the probability of existence

9 Temporal Uncertainty

Temporal is related to time, which is a (quantitative) continuous variable, while *uncertainty* refers to situations involving unknown information.

- (see also *Time and dating*)

Monte-Carlo simulation tests the significance of the observed fluctuations in the context of uncertainty in the calibration curve and [archaeological] sampling.

9.1 Probability and uncertainty

We can apply a probabilistic approach when dealing with uncertainty. In this case, uncertainty is considered a measure quantifying the likelihood that events will occur.

Different approaches for assigning probabilities include:

- the proportion of a particular outcome to all possible outcomes (classical)
- the long-run relative frequency of the probability for an outcome to occur
- certain degree of belief (subjective)

Probability space

A *probability space* defines an occurrence or event as a subset of that space.

For example, if the probability space is the interval $[0, 1]$, with the *uniform distribution* then the interval $[\frac{1}{3}, 1]$ is an event that represents a randomly chosen number between 0 and 1 turning out to be at least $\frac{1}{3}$.

9.2 Probability distributions

A *probability distribution* describes the associated probability of possible outcomes for a random variable X.

For a single variable, the data is allocated in univariate distributions, and we use multivariate distributions to infer on multiple parameters.

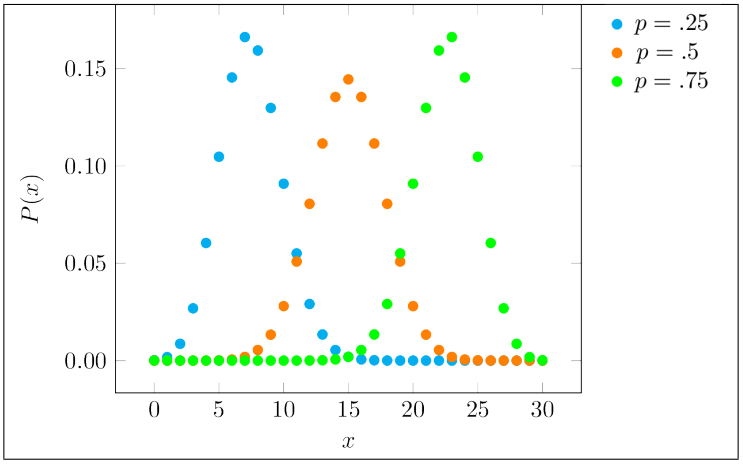
Binomial distribution

A binomial distribution is a continuous distribution that describes a binomial experiment where there are two possible outcomes.

The probability mass function (p.m.f.) corresponding to the binomial distribution represents the probability of x successes in a binomial experiment

$$P(x) = \binom{n}{x} p^x (1-p)^{n-x}$$

where $P(x)$ is the distribution function for the probability of success for $x = 0, 1, 2, \ldots, n$.



A plot of the binomial distribution for $n = 30$ and with different p values.

Note: The Bernoulli distribution is a special case of the binomial distribution when $n = 1$.

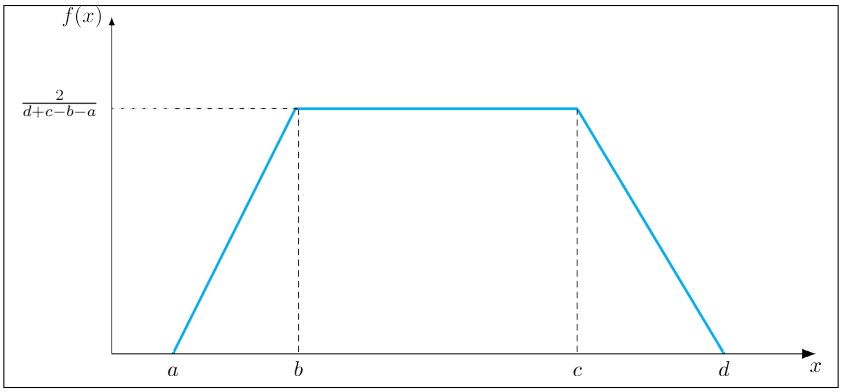
Trapezoidal distribution

Trapezoidal distributions seems appropriate for modeling the duration and the form of a phenomenon with a growth-stage, relative stability, and decline. These three parameters are not necessarily similar and an occurrence can have, for example, a long development and abrupt decay.

The probability density function of the trapezoidal distribution for $a \leqslant b \leqslant c \leqslant d$ is

$$f(x \mid a, b, c, d) = \begin{cases} U\left(\frac{x-a}{b-a}\right) & a \leqslant x < b \\ U & b \leqslant x < c \\ U\left(\frac{d-x}{d-c}\right) & c \leqslant x < d \\ 0 \quad \text{elsewhere} \end{cases}$$

where $U = 2(d + c - b - a)^{-1}$ corresponds to the uniform distribution.



A trapezoidal distribution where decline is longer than growth.

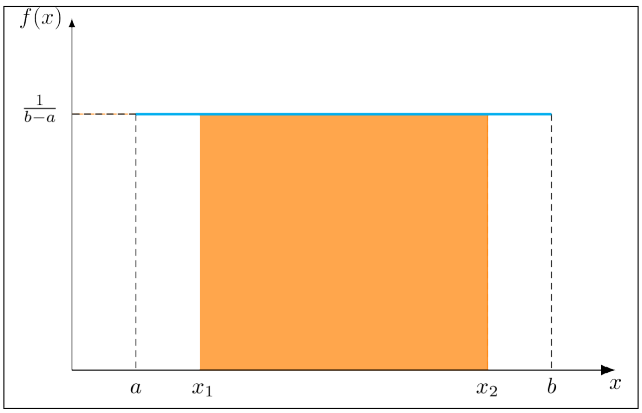
Uniform distribution

The uniform distribution is a special case of a trapezoidal distribution with a constant probability.

The probability density function of the continuous uniform distribution for $a \leqslant x \leqslant b$

$f(x) = \frac{1}{b-a}$

and $f(x) = 0$ iff $x < a$ or $x > b$



Uniform distribution plot for $P(x_1 < X < x_2)$.

Note: There is also a discrete version of the uniform distribution with a mass probability function.

Normal distribution

The normal distribution –or "bell curve"– is the probability distribution for a normal random variable $X \sim N(\mu, \sigma^2)$. The normal distribution is also called *Gaussian* distribution due to C.F. Gauss who described this distribution in mathematical terms.

The normal distribution is the most important distribution in statistics, and play a crucial role in statistical inference. This is partly because it approximates well the distributions of many types of variables.

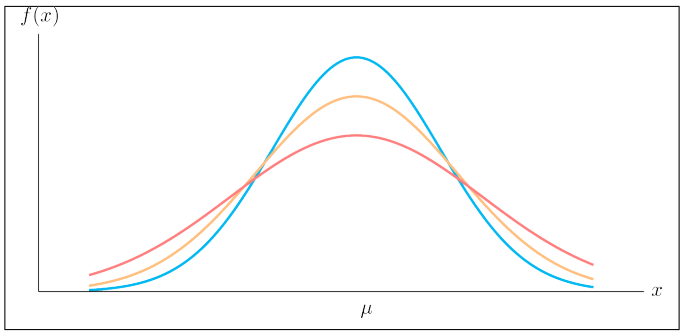
The exact form of the distribution depends on the values of the mean μ and the standard deviation σ parameters.

The probability density function of a normal random variable $-\infty < x < \infty$ is:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

where $e \approx 2.7183$ (Euler's number), and $\pi \approx 3.1416$ (Pi number).

> **Note:** A special case of the normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 1$ is the *standard normal distribution* Z. Any arbitrary normal distribution can be converted to Z.



Normal distributions with different variances σ^2 and same mean μ.

9.3 Notation

To study temporal uncertainty, the following notation is adopted:

$\Omega =$ range of time

$\tau =$ time span of existence

$\Delta\tau =$ duration of τ

$t_i =$ a given portion of time

$\Delta t_i =$ duration of t_i

$\varphi =$ temporal resolution

$e =$ event or occurrence

$P, p =$ probability

Boundaries of existence

Limits of dating for events are the earliest and the latest time the event may have happened. These upper and lower bounds are termed

- *terminus ante quem* (TAQ) or "limit before which"
- *terminus post quem* (TPQ) or "limit after which"

9.4 Aoristic analysis

Aoristic analysis is based on the creation of a series of these artificial divisions of the range of time with a fixed value, called time blocks, and the definition of their probability of existence.

Temporal resolution

Temporal resolution φ refers to the duration of time blocks, and with aoristic analysis φ is fixed.

Aoristic sum

A proxy for evaluating change in the total counts of events across time is the *aoristic sum*, which is the sum of probabilities for each time block.

The aoristic sum is computed accross the sum of probabilities for events in a single portion of time t_i.

```
    t1  t2  t3
     ^   ^   ^
a    |   |   |
b    |   |   |
c    |   |   |
     ^   ^   ^
     aoristic
       sums
```

Todo: Aoristic sum explained in R...

9.5 Probability of existence t_i

The probability of existence $p[t_i]$ of an event e in a range of time Ω is the probabilty distribution

$p[t_i] = \Delta t_i / \Delta \tau$

where τ is the time span of existence, Δ is for the duration of a given portion of time t_i or of τ.

To compute $p[t_i]$, Crema (2012, p. 447) takes the following example (years are BC):

```
350 ---- 300 ---- 250
     t1        t2

     342 ---- 288
          tau
```

Hence, $\Delta\tau$ in this case is 54.

To retrieve the minimum probability $P[\varphi_\alpha]$ for the boundaries of the first and last time-blocks, we define $\varphi_\alpha = 1$ where

$\phi_\alpha / \Delta_\tau = 1/54 = P[\phi_\alpha] = .018$ (approx)

Hence, the probability of existence for t_1

$P[t_1] = P[\phi_\alpha] \cdot 42$

(i.e. $342 - 300$)

$P[t_1] = .018 \cdot 42 = .78$

The probability of existence for for t_2

$P[t_2] = P[\phi_\alpha] \cdot 12$

(i.e. $300 - 288$)

$P[t_2] = .018 \cdot 12 = .22$

Etc.

Assumption (among others): Any equally long portion of time within the time span τ has the same probability of existence.

9.6 Probabilities of existence for time blocks

The probability of existence for each time block (Appendix A, Crema, 2012)

$P_b = \sum_{i=0}^{\pi-1} \sum_{j=1}^{d/\varphi} \theta(i+j,\ b)/\pi$

where π is the number of all possible permutations, and b is a numerical index of temporal blocks within τ.

On the other hand, the number of all possible permutations π equals $(\Delta\tau - d)/\varphi + 1$ where d is the duration of e.

Note: d is rounded to the value of the temporal resolution, and b ranges from 1 to $\Delta t_i/\varphi$, which are the edges of the time span.

Temporal resolution for time blocks

The duration of time blocks that is φ in the denominator for computing π is known as the *temporal resolution*, while the numerator expression for the probability of existence for each time block is defined as

$$\theta(i+j,\ b) = \begin{cases} 1 & \text{if } i+j=b \\ 0 & \text{if } i+j \neq b \end{cases}$$

Rate of change

Rate of change refers to transition probabilities like increase/stability/decrease as with the *trapezoidal* distribution.

Rate of chage = $\frac{(e\ t_{i+1} - e\ t_i)}{\varphi}$

An example of a transition probabilities are increase $= 0.4$; stability $= 0.5$ and decrease $= 0.1$

9.7 Imputation and Missing data

In statistics, *imputation* is the process of replacing missing data with plausible estimates.

Missing data

Every data point has some likelihood of being missing, and Rubin (1976) classified missing data problems into three categories: missing completely at random (MCAR), missing at random (MAR) and missing not at random (MNAR).

For some parameter ψ, where Y_{obs} is the observed sample data and Y_{mis} the unobserved sample data, the overall probability of being missing x depends,

- MCAR: only on some parameters ψ

  $$P(x = 0 \mid Y_{obs}, Y_{mis}, \psi) = P(x = 0 \mid \psi)$$

- MAR: on observed information, including any design factors

  $$P(x = 0 \mid Y_{obs}, Y_{mis}, \psi) = P(x = 0 \mid Y_{obs}, \psi)$$

- MNAR: also depends on unobserved information, including Y_{mis} itself

  $$P(x = 0 \mid Y_{obs}, Y_{mis}, \psi)$$

  See also:

  Complete cases

9.8 Computing the probability of existence

We use function `prex()` to compute the probability of existence.

`prex()`

Function usage

```
# probability of existence
R> prex(x, vars, bins=NULL, cp=c("bin5","bin8"), aoristic=TRUE, weight=1, DF,
        plot=FALSE, main=NULL,...)
```

Parameters

- *x:*

  list or data frame object of variables and observations.

- *vars:*

  boundaries of existence in `x` (vector for TAQ and TPQ)

- *bins:*

  length of the break (integer)

- *cp:*

  chronological phase. `"bin5"` or `"bin8"` for five- or eight-periods

- *aoristic:*

  return aoristic sum? (logical)

- *weight:*

  weight to observations

- *DF:*

  return also data frame with observations? Ignored for plot (logical and optional)

- *plot:*

  plot the results? (logical and optional)

- *main:*

  plot's main title (optional)

- *. . . :*

  additional optional parameters

For example, the (default) aoristic sum of `EDH` inscriptions in the Roman province of `Iudaea` computed with `prex()`

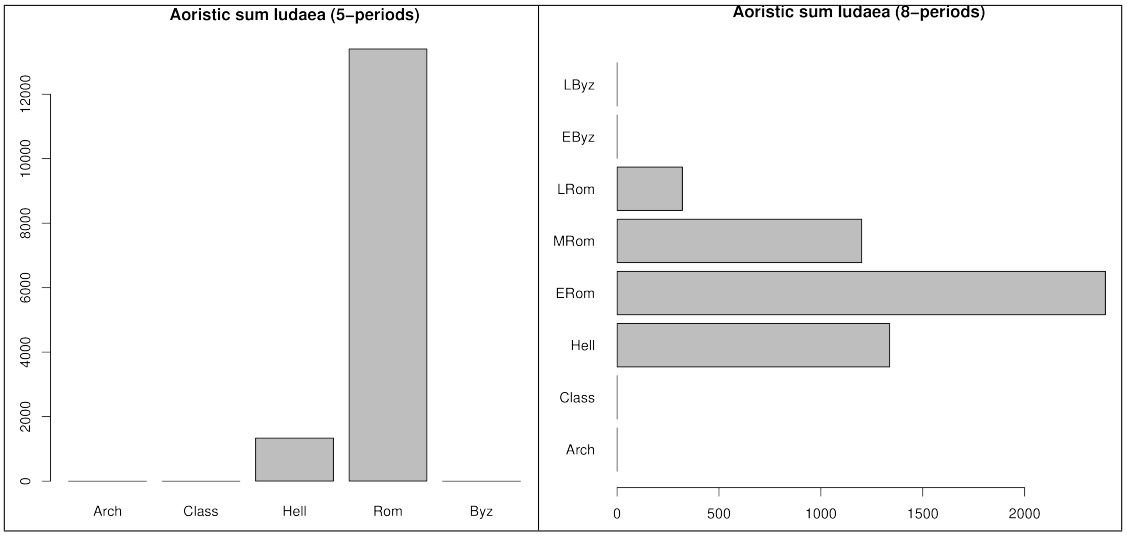
```
# get inscriptions from Iudaea in EDH data base
R> iud <- get.edh(search="inscriptions", province="Iud")

# 5-chronological periods
R> prex(x=iud, vars=c("not_before", "not_after"), cp="bin5")
#  Arch     Class      Hell       Rom       Byz
# 0.000     0.000  1337.904 13405.017     0.000

# 8-chronological periods
R> prex(x=iud, vars=c("not_before", "not_after"), cp="bin8")
#  Arch     Class      Hell      ERom      MRom      LRom      EByz      LByz
#0.0000    0.0000 1337.9040 2396.4529 1200.5623  320.5379    0.0000    0.0000
```

Plotting aoristic sum

To visualize the aoristic sum from `Iudaea` as bar plots and function `prex()`

```
#  five chronological phases
R> barplot(prex(x=iud, vars=c("not_before", "not_after"), cp="bin5"),
+          main="Aoristic sum Iudaea (5-periods)" )

# eight chronological phases
R> barplot(prex(x=iud, vars=c("not_before", "not_after"), cp="bin8"),
+          horiz=TRUE, las=1, main="Aoristic sum Iudaea (8-periods)" )
```

that produce



See also:

Relative dating of Roman inscriptions

10 Markov chain Monte Carlo

10.1 Monte Carlo

Monte Carlo (term coined by S. Ulam and N. Metropolis) refers to the method of generating random numbers from a certain distribution.

For dealing with *temporal uncertainty*, for example, a Monte Carlo method consists on creating artificial "potential" time series within the boundary of existence, and then assessing the likelihood of the variable's character state over time.

10.2 Markov chain

A *Markov chain* (after A. Markov) is a sequence of numbers where each number is dependent on the previous number in the sequence.

To simulate a Markov chain whose equilibrium distribution is a posterior density $p(x)$ defined on the set of possible configurations of a system, we use Markov chain Monte Carlo (MCMC) methods.

Some MCMC algorithms:

- Metropolis-Hastings
- Gibbs sampling
- Random walk Metropolis

Alternative to MCMC:

- Importance/rejection sampling (used to evaluate the probability of rare events)

MCMC is used to fit a model and to draw samples from the joint posterior distribution of the model parameters.

- (see *Bayesian statistics*)

10.3 Metropolis-Hastings

The *Metropolis-Hastings* algorithm serves to generate a sample from the posterior distribution of θ, which is the parameter of the data point's distribution.

Algorithm

1. Initialise x_0
2. For $i = 0$ to $N - 1$

    Sample $u \sim U(0, 1)$ uniform distribution

    Sample $x^\star \sim q(x^\star \mid x_t)$ candidate point on a proposal distribution

    If $u < U() = \min\{1, \frac{p(x^\star)\ q(x_t|x^\star)}{p(x_t)\ q(x^\star|x_t)}\}$

        $x_{(t+1)} = x^\star$ accept candidate

    Else

        $x_{(t+1)} = x_t$ increment time and sample $x^\star$ until $t_{\max}$ is reached.

10.4 Diagnostics

Diagnostics is to assess the MCMC performance.

The trace plot (or time-series plot) is to judge how quickly the MCMC procedure converges in distribution.

When the variability is not the same over all iterations, the trace plot presents a "random walk" pattern.

Burn-in

First generate a series of random numbers from a normal distribution with a mean value and some arbitrary variance.

Todo: Throwing away early samples is called burn-in...

MCMC Simulation

Todo: After the burn-in, are the MCM iterations...

Thinning

Todo: Thinning interval for reducing autocorrelation of random points in the sampling...

Convergence

Convergence in MCMC is obtained when these outcomes become minimal with the increase of n:

- the change in the variance
- the standard error

10.5 Summed probability distribution

Todo: A proxy for population levels is found in summed probability distributions (SPD) of dates, which is the summation of the posterior probability distributions of calibrated dates.

10.6 Bayesian statistics

Markov chain Monte Carlo is a tool for sampling from distributions with high dimensionality, which is a need typically found in some Bayesian statistics contexts.

Bayesian analysis implies Bayesian inference, model fitting, and performing inferential and predictive summaries with predictive checks.

As mentioned in *temporal uncertainty*, the subjective approach in assigning probabilities includes a certain degree of belief or knowledge to the probability, and this is to arrive at a posterior distribution to updated the state of knowledge about the parameters.

Bayes theorem

Key concepts in Bayesian statistics are related in Bayes theorem, which is a rule that provides the conditional probability of θ (model or hypothesis) occurring given that x (data or observation) already happened.

$P(\theta \mid x) = \frac{P(\theta)\ P(x|\theta)}{P(x)}$

That is, the posterior probability of θ given x where

- $P(\theta \mid x)$ is the **posterior** probability of θ
- $P(\theta)$ is the **prior** probability of θ

- $P(x \mid \theta)$ is the **likelihood** of θ
- $P(x)$ is the **unconditional** probability of x

Hence, the posterior is an updated degree of belief or knowledge based on the prior with known probabilities. This is our posterior *target* distribution sometimes written as $P_{target}(\theta)$.

On the other hand, the likelihood is the probability that the hypothesis confers upon the observation, and the probability of the observation irrespective of any hypothesis is the unconditional probability of the observation.

11 Epigraphic Networks using R

This post is about Epigraphic Networks based on measures of similarity of artefact assemblages and geographic proximity.

11.1 Measuring similarity of artefact assemblages and geographic proximity

To measure similarity of artefact assemblages and geographic proximity, [R] package `sdam` provides the function `simil()`, which allows assesing similarity by comparing columns representing –in this case– different attributes for epigraphic inscriptions.

Function usage

`simil``()`

```
# arguments supported (currently)
R> simil(x, att, null, uniq, diag.incl)
```

Which returns a square and valued matrix with similarity meassures based on simple match among variables.

Parameters

Formal arguments of `simil()` are:

- *x:*

  a data frame with an *id* column

- *att:*

  (vector) column(s) in `x` representing attributes

Optional parameters

- *null:*

  include in computation `NA` or `NULLs`?

- *uniq:*

  remove duplicates from `x`?

- *diag.incl:*

  include entries in the matrix diagonal?

At this point, the ID column represents the labels of the nodes. In case that an ID column does not exists, then the first column is taken as *id* provided that there are not duplicated entry names in `x`.

11.2 EDH dataset

We illustrate the use of the `simil()` function with ancient inscriptions from the Epigraphic Database Heidelberg, and we first follow the entry *Epigraphic Database Heidelberg* to see how accessing the `EDH` dataset using `sdam` [R] package.

```
# devtools::install_github("mplex/cedhar", subdir="pkg/sdam")
# devtools::install_github("sdam-au/sdam")
R> library("sdam")

# load the EDH data from this package
R> data("EDH")
```

Epigraphic network data

Creating epigraphic network data list with variables measures of similarity of artefact assemblages and geographic proximity. For example, a list object named `epinet` with the `ID` of the inscription plus seven other characteristics from the `EDH` dataset is produced by `edhw()`.

```
# choose variables of interest and record it as a data frame
R> epinet <- edhw(c("type_of_monument", "language", "material", "country", "findspot_anc
                    "not_after", "not_before"), as="df")
```

Take a look at this data:

```
# first eight entries in the data frame
R> head(epinet, 8)
#         id country findspot_ancient language                                material no
#1 HD000001   Italy       Cumae, bei    Latin                 Marmor, geädert / farbig
#2 HD000002   Italy             Roma    Latin       marble: rocks - metamorphic rocks
#3 HD000003   Spain             <NA>    Latin       marble: rocks - metamorphic rocks
#4 HD000004   Spain    Ipolcobulcula    Latin    limestone: rocks - clastic sediments
#5 HD000005   Italy             Roma    Latin                                    <NA>
#6 HD000006   Spain      Sabora, bei    Latin    limestone: rocks - clastic sediments
```

(continues on next page)

(continued from previous page)

```
#7 HD000007   Italy            Roma    Latin travertine: rocks - chemische Sedimente
#8 HD000008   Italy           Roma?    Latin       marble: rocks - metamorphic rocks
```

For instance, entry `8` indicates that this ancient findspot is uncertain.

Regular expressions

To remove question marks, for example, we use a regular expression

```
R> epinet2 <- as.data.frame(sapply(epinet, function (x)  as.list(gsub('\\?', '', x)) ))
```

> **Todo:** Make another function for cleaning? After all, `sapply()` converts `<NA>` into `NA`.

And then we take a look at `epinet2` again, and we assume the questioned entries.

```
# first eight entries in the new data frame
R> head(epinet2, 8)
#        id country findspot_ancient language                                material no
#1 HD000001   Italy       Cumae, bei    Latin                Marmor, geädert / farbig
#2 HD000002   Italy             Roma    Latin       marble: rocks - metamorphic rocks
#3 HD000003   Spain               NA    Latin       marble: rocks - metamorphic rocks
#4 HD000004   Spain    Ipolcobulcula    Latin    limestone: rocks - clastic sediments
#5 HD000005   Italy             Roma    Latin                                      NA
#6 HD000006   Spain      Sabora, bei    Latin    limestone: rocks - clastic sediments
#7 HD000007   Italy             Roma    Latin travertine: rocks - chemische Sedimente
#8 HD000008   Italy             Roma    Latin       marble: rocks - metamorphic rocks
```

The countries in `epinet2` are:

```
# need first to unlist the component object
R> unique(unlist(epinet2$country))
# [1] "Italy"                  "Spain"                  "United Kingdom"         "Portug
# [5] "France"                 "Libyan Arab Jamahiriya" "Germany"                "Hungar
# [9] "Austria"                "Bulgaria"               "Bosnia and Herzegovina" "Monter
#[13] "Netherlands"            "Tunisia"                "Romania"                "Algeri
#[17] "Jordan"                 "NULL"                   "Croatia"                "Switze
#[21] "Belgium"                "Albania"                "Serbia"                 "Egypt"
#[25] "Syrian Arab Republic"   "Morocco"                "Turkey"                 "Leban
#[29] "Kosovo"                 "Macedonia"              "Slovakia"               "Greece
#[33] "Slovenia"               "Iraq"                   "Israel"                 "unknow
#[37] "Vatican City State"     "Ukraine"                "Cyprus"                 "Yemen"
#[41] "Sudan"                  "Luxembourg"             "Czech Republic"         "Malta"
#[45] "Poland"                 "Armenia"                "Monaco"                 "Azerba
#[49] "Sweden"                 "Denmark"                "Moldova"                "Saudi
#[53] "Uzbekistan"             "Liechtenstein"          "Georgia"
```

Subsetting the data

For example, we use the `base` [R] function `subset()` to substract epigraphic material in "Greek-Latin" from Egypt.

```
# a subset of a subset
R> subset(subset(epinet2, country=="Egypt"), language=="Greek-Latin")
#            id country findspot_ancient    language    material not_after not_before
#2003  HD002009   Egypt           Philae Greek-Latin          NA        NA      -0116 ar
#23091 HD023319   Egypt              NA Greek-Latin Holz, Wachs        NA       0145
#23138 HD023368   Egypt            Syene Greek-Latin          NA        NA      -0029
#27345 HD024001   Egypt              NA Greek-Latin Holz, Wachs        NA       0145
#27351 HD024009   Egypt            Syene Greek-Latin          NA        NA      -0029
#32500 HD030457   Egypt       Alexandria Greek-Latin          NA      0011       0010
#34436 HD032412   Egypt          Schedia Greek-Latin          NA        NA         NA
#51198 HD049264   Egypt         Berenice Greek-Latin          NA        NA       0006
#54194 HD052276   Egypt       Alexandria Greek-Latin          NA      0225       0155
#58318 HD056791   Egypt         Berenice Greek-Latin          NA      0200       0001
#70110 HD068637   Egypt      Leontopolis Greek-Latin          NA     -0030      -0037
```

Ranked frequency

A ranked frequency of different *kinds of inscriptions* including missing information is computed as follows:

```
R> as.data.frame(sort(table(unlist(epinet2$type_of_inscription), useNA="ifany"), decreas
#                              Var1  Freq
#1                          epitaph 27898
#2                             <NA> 21883
#3                votive inscription 14213
#4           owner/artist inscription  4876
#5              honorific inscription  4314
#6    building/dedicatory inscription  3378
#7                  mile-/leaguestone  1713
#8         identification inscription  1353
#9                  military diploma   507
#10                      acclamation   495
#11                             list   356
#12                          defixio   309
#13                            label   266
#14          public legal inscription   254
#15              boundary inscription   252
#16                          elogium   154
#17                 seat inscription    86
#18                           prayer    56
#19                           letter    39
#20        private legal inscription    34
#21          assignation inscription    13
#22                         calendar    12
#23                       adnuntiatio     3
```

That is, a decreasing sorted table given as data frame of the `type_of_inscription` component of `epinet2`. Since `epinet2` is a list object, it is required to *unlist* the data object to produce a table with the frequencies.

11.3 Example: Similarity among Egyptian epigraphs

We can compute similarity among Egyptian epigraphs with function `simil()`. For this, we look at the attribute types stored in different columns.

```
R> as.data.frame(colnames(epinet2))
#    colnames(epinet2)
#1                   id
#2              country
#3    findspot_ancient
#4             language
#5             material
#6           not_after
#7          not_before
#8 type_of_inscription
```

For instance, in case we want to choose `"type_of_inscription"`, `"material"`, and `"findspot_ancient"`, these correspond to columns 8, 5, and 3.

Similarities among Egyptian epigraphs *by simple matching* with the above attribute variables are recorded in a matrix object named `epEgs` where the `ID` in `epinet2` corresponds to the dimensions labels.

```
# similarity function on the subset for the three variables
R> epEgs <- simil(subset(epinet2, country=="Egypt"), c(8,5,6))

# number of rows in this square matrix
R> nrow(epEgs)
#[1] 175
```

And then we look at some cell entries

```
# similarity between the first six inscriptions in 'epEgs'
R> epEgs[1:6, 1:6]
#         HD000744 HD002009 HD003137 HD006817 HD006820 HD008184
#HD000744        0        1        1        0        0        0
#HD002009        1        0        0        0        0        0
#HD003137        1        0        0        0        0        0
#HD006817        0        0        0        0        1        0
#HD006820        0        0        0        1        0        0
#HD008184        0        0        0        0        0        0
```

where we observe six records of a single similarity.

Plot similarities

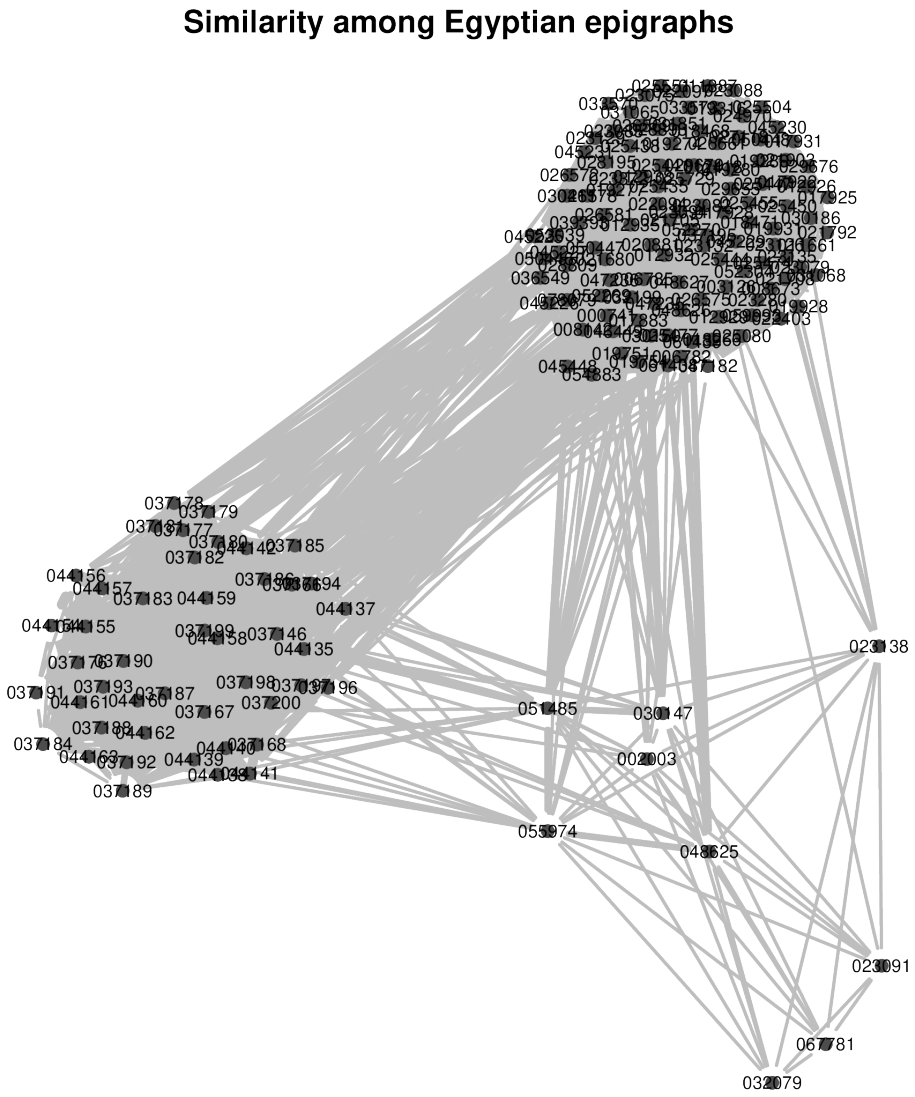
To produce a graph for the similarity among Egyptian epigraphs, we employ the [R] package `multigraph` that depends on `multiplex`.

```
# define scope for the graph
R> scp <- list(directed=FALSE, valued=TRUE, ecol=8, pos=0)

# load "multigraph" where "multiplex" gets invoked
```

(continues on next page)

(continued from previous page)

```
R> library(multigraph)

# plot similarity graph of 'epEgs' for the chosen variables
R> multigraph(epEgs, scope=scp, layout="force", maxiter=70, main="Similarity among Egypt
```



12 Modules

13 Indices and tables

- genindex
- search

13.1 Print

- Download this document.