archivist: An R Package for Managing, Recording and Restoring Data Analysis Results

Przemysław Biecek
University of Warsaw

Marcin Kosiński
Warsaw University of Technology

Abstract

Everything that exists in R is an object Chambers (2016). This article examines what would be possible if we kept copies of all R objects that have ever been created. Not only objects but also their properties, meta-data, relations with other objects and information about context in which they were created.

We introduce **archivist**, an R package designed to improve the management of results of data analysis. Key functionalities of this package include: (i) management of local and remote repositories which contain R objects and their meta-data (objects' properties and relations between them); (ii) archiving R objects to repositories; (iii) sharing and retrieving objects (and it's pedigree) by their unique hooks; (iv) searching for objects with specific properties or relations to other objects; (v) verification of object's identity and context of it's creation.

The presented **archivist** package extends, in a combination with packages such as **knitr** and **Sweave**, the reproducible research paradigm by creating new ways to retrieve and validate previously calculated objects. These new features give a variety of opportunities such as: sharing R objects within reports or articles; adding hooks to R objects in table or figure captions; interactive exploration of object repositories; caching function calls with their results; retrieving object's pedigree (information about how the object was created); automated tracking of the performance of considered models, restoring R libraries to the state in which object was archived.

Keywords: recordable research, reproducible research, data analysis management, data governance, meta-data, results management.

A vigniette for the Biecek and Kosinski (2017) article.

1. Introduction

In most of the cases the outcome of the process of data analysis is a set of objects in the form of statistical models, charts or tables. Three requirements are often superimposed to ensure sufficient quality of such results: they should be reproducible, verifiable and accessible. Reproducibility means that there is a process that reproduces results. Verifiability means that it is possible to check whether the newly generated results are identical to previously obtained results, and it is possible to check the context of object's creation. Accessibility means that results can be easily accessed for future computer based processing. Reproducibility gets increasing attention in the academic literature across various disciplines, see for example Peng (2009) for bioinformatics or Koenker and Zeileis (2009) for the econometric research

or Drummond (2009) for more general discussion about differences between replicability and reproducibility.

The R ecosystem of packages is equipped with wonderful tools such as **knitr** (see Xie 2013, 2015) or **Sweave** (see Leisch 2002; Rossini and Leisch 2003) which allow to create reproducible reports or articles. They follow the *literate programming* principle, and the R code, its results and its explanations appear together in a single document. It is assumed that the same input and identical instructions executed on the same operating system with the same local settings and with identical versions of installed libraries will result in the same output. Under these assumptions **knitr** or **Sweave** reports are sufficient to recreate the previously obtained results.

But there are cases in which it is not convenient to recreate results from scratch, from raw input. Consider the following situations:

- the input data is large or with limited/restricted access (e.g., for genomic data the raw input may easily hit few TB);
- computations take a lot of time or require specialized hardware (e.g., calculations tuned for Graphics Processing Unit cards);
- calculations are based on a very specific version of software or require commercial versions of software or some functions are deprecated or removed over time. It can be an issue even for open software, e.g., due to rapid development of R, even widely used packages experience significant changes, like **ggplot2** or **lme4** in the year 2015;
- results are generated and processed periodically and you wish to restore and compare models across all reports.

In such situations it is desirable to retrieve the results that were calculated in the past rather than reproducing them from scratch. Objects that are backed up can be reused even if they cannot be reproduced or the reproducibility will be too complex or time consuming. Alternatively, it may be desired to check whether the reproduced results are the same as those obtained previously.

An interesting example of such a feature are StatLinks (see OECD 2015) commonly used in reports prepared by OECD (Organization for Economic Co-operation and Development). In addition to scripts that generate results, most tables and plots that are presented in the reports are equipped with their own DOIs (digital object identifier) and web hooks. Through these links readers may download selected tables and plots, in the Excel format. The `xls` and `xlsx` formats are not ideal as they are proprietary and difficult to read in an automated way. But for extensive studies it is convenient and faster to access final results in such formats instead of having scripts that reproduce them.

If the only result from the data analysis is a single plot, a model or a table, it is easy to save it in the `rda` format and make it accessible for the others. But increasing amounts of heterogeneous data results in growing complexity of the process of data analysis. The complexity comes either from data volume, data heterogeneity, numerous steps required for data preparation, results validation etc. Moreover, working with data is often a highly iterative process that generates large amount of partial or final results. For all the above reasons the management of versions of results becomes a task in itself. Neglecting this process results in *Reproducibility Debt* and may consequently lead to huge additional workload when it comes

to recreation of results. The *Reproducibility Debt* is a part of wider category called *Technical Debt* (see Sculley, Holt, Golovin, Davydov, Phillips, Ebner, Chaudhary, and Young 2014).

It should be noted that the concept of recording and exploring relations between objects is not new. Potential applications in auditable data analyses were discussed almost 30 years ago (see Becker and Chambers 1988). What we present in this article may be perceived as implementation of some of these concepts. It is now easier due to lower costs of data storage.

The **archivist** package helps in managing, sharing, storing, linking and searching for R objects in a platform agnostic way. Its core functionalities allow for many interesting applications - some of them are presented in the Section 2. The **archivist** package automatically retrieves the object's meta-data and creates a rich structure that allows for easy management of stored R objects. The meta-data covers object's properties such as: name, creation date, class, versions of attached packages, structure and relations between R objects (as for example, that an object A was used for creation of an object B). All examples presented here are related to R objects. In the Section 3.4 we discuss how this approach can be extended to other languages.

The rest of the article has the following structure. In the Section 2 (*Motivation*) we introduce key motivations and use-cases behind **archivist**. In the Section 3 (*Functionality*) we present all functions available in the package and point out some further directions how this functionality can be integrated with GitHub, **knitr**, or be extended on other languages / formats. In the Section 4 (*Conclusions*) we gather some final thoughts related to recordable and restorable research.

2. Motivation

In this section we present key concepts and some use-cases behind the **archivist** package. In the Section 3 we present all functions available in the **archivist** in a more formal way. First let us introduce some terminology.

- Artifact - an R object that is saved to the repository. Artifacts are identified by their MD5 hashes.
- Repository - a collection of artifacts stored as binary files outside of the R session. Repositories are either local (with a read-write access) or remote (with a read access only). The API for repositories allow for following actions: add, delete, read or search for an artifact with selected Tags. In the current version of the **archivist** local repositories are folders in the file system while remote repositories are Git or Mercurial based repositories. The same mechanism can be used to access repositories pointed as URL addresses or folders attached to R packages.
- MD5 hash - a unique identifier of an artifact. It's a 32-character-long string, result of cryptographical hash function MD5 (Message Digest algorithm 5). Here, we are using implementation of hash function available in the **digest** package (Dirk Eddelbuettel, Tuszynski, Bengtsson, Urbanek, Frasca, Lewis, Stokely, Muehleisen, Murdoch, Hester, and Wu. 2014, see). In the **archivist** package MD5 hashes are used as object's hooks.
- Tag - an attribute of an artifact. Tags are represented as character strings; they usually have the following structure: `key:value`. An artifact may have many tags, even with the same key. Some tags are automatically derived from artifacts, others may be

added manually. Tags may be referred as meta-data of artifacts as they describe either properties of artifacts (e.g., class, name, date of creation) or relations between artifacts (e.g., being a part of, being a result of).

The **archivist** package manages R objects outside the R session. It stores binary copies of R objects in `rda` files and provides easy access for seeking and restoring these objects based on timestamps, classes or other properties.

But, why anybody would like to store copies of R objects? Let's imagine the following use-cases:

- A data scientist creates a report or an article and would like to provide an access to results presented in the article. Typically, these results are presented as plots, tables or models. Apart from including these results in the report or article in a human-readable form, it may be beneficial to be able to restore a given result in a machine-readable form for further processing. Having a possibility to retrieve an R plot or table, one can perform some further transformation of it. The opportunity to retrieve a regression model enables additional residuals' validation or applying model to the new data. The **archivist** creates a hook to a copy of R object which restores the object in a remote R session. Such hooks are short one-line instructions and can be embedded in figures' or tables' captions.

  An example report that illustrates this use-case is available at `http://bit.ly/1nW9Cvz`. A part of it is presented in Figure 1. The report is created with the use of **knitr** package. It contains both R code and it's results in the form of tables and plots created with **ggplot2** package (see Wickham 2009). In addition, there are also hooks to selected results. These hooks allow to restore a given plot or table directly in the local R session. Hooks of such a form restore a `gg` object in an R session.

  ```
  archivist::aread("pbiecek/Eseje/arepo/ba7f58fafe7373420e3ddce039558140")
  ```

- A team of data scientists is working for some time on a forecasting model. During a certain period of time a large set of competing models is created. The team needs a tool that stores all models with additional metadata, such as model performance, information which data was used for model training and testing. The **archivist** creates a shared repository which can be used for storing models along with their meta-data and provides API for searching objects with specific meta-data. The example below reads all objects of the class `lm`, calculates a BIC score for them and sorts objects with respect to these scores.

```
library("archivist")

## Welcome to archivist (version:  2.2).
```

```
models <- asearch("pbiecek/graphGallery", patterns = "class:lm")
modelsBIC <- sapply(models, BIC)
sort(modelsBIC)
```

```
## 990861c7c27812ee959f10e5f76fe2c3 2a6e492cb6982f230e48cf46023e2e4f
##                         39.05577                         67.52735
## 0a82efeb8250a47718cea9d7f64e5ae7 378237103bb60c58600fe69bed6c7f11
##                        189.73593                        189.73593
## 7f11e03539d48d35f7e7fe7780527ba7 c1b1ef7bcddefb181f79176015bc3931
##                        189.73593                        189.73593
## 0e213ac68a45b6cd454d06b91f991bc7 e58d2f9d50b67ce4d397bf015ec1259c
##                        243.49450                        243.49450
## 18a98048f0584469483afb65294ce3ed
##                        396.16690
```

- Results are generated in a remote R process, like for example with a Shiny application.

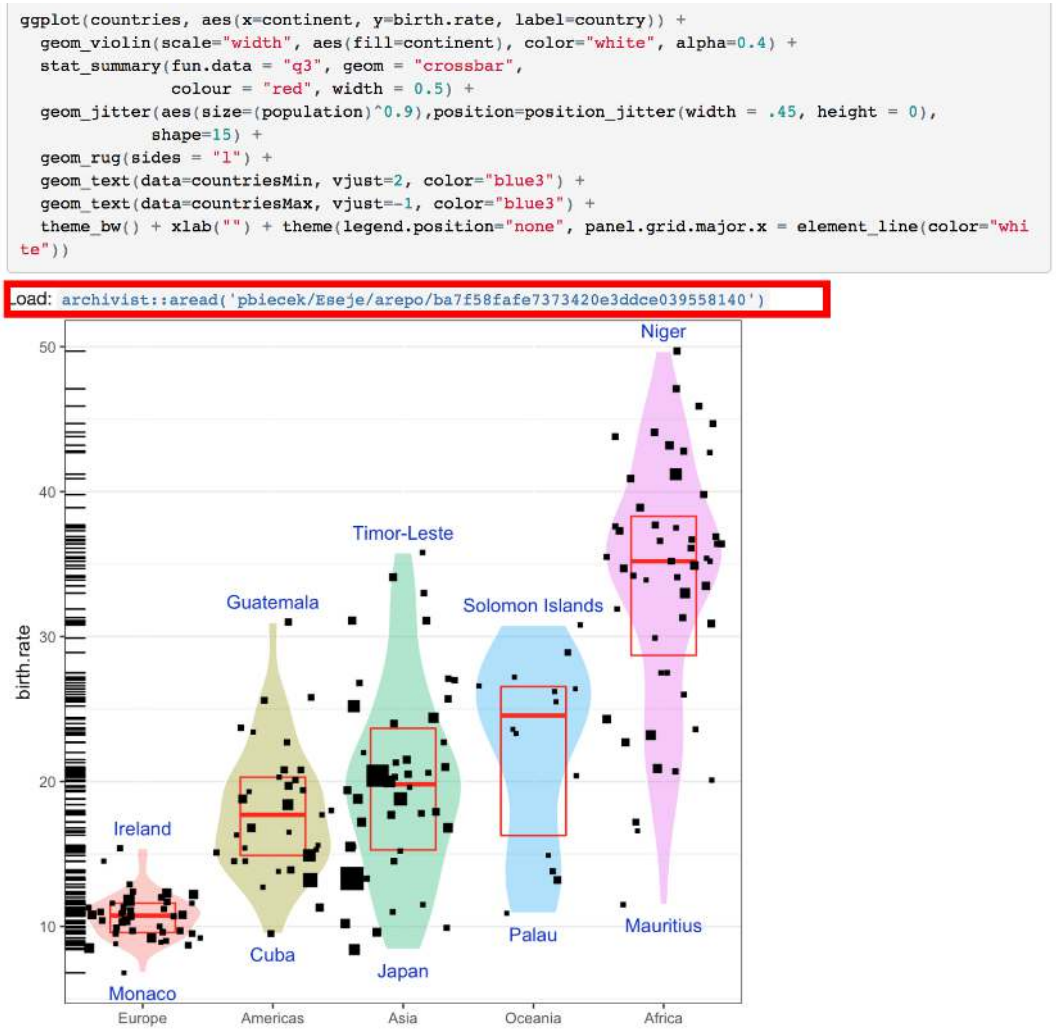


Figure 1: A part of a **knitr** report `http://bit.ly/1nW9Cvz` that uses the `addHooksToPrint` function that automatically adds **archivist** hooks to all objects of a given class. Objects can be accessed either by copying highlighted `aread` instructions to R or by clicking the link.

The **archivist** saves created R artifacts in an URL repository.

See for example Figure 2 that presents a screenshot from the Shiny application `https://cogito.shinyapps.io/archivistShiny`. All plots generated by this application are stored in an **archivist** repository and may be accessed with hooks presented below plots. Following line downloads a single plot directly to the local R session.

```
archivist::aread("https://cogito.shinyapps.io/archivistShiny/arepo/
ca680b829abd8f0a4bd2347dcf9fe534").
```

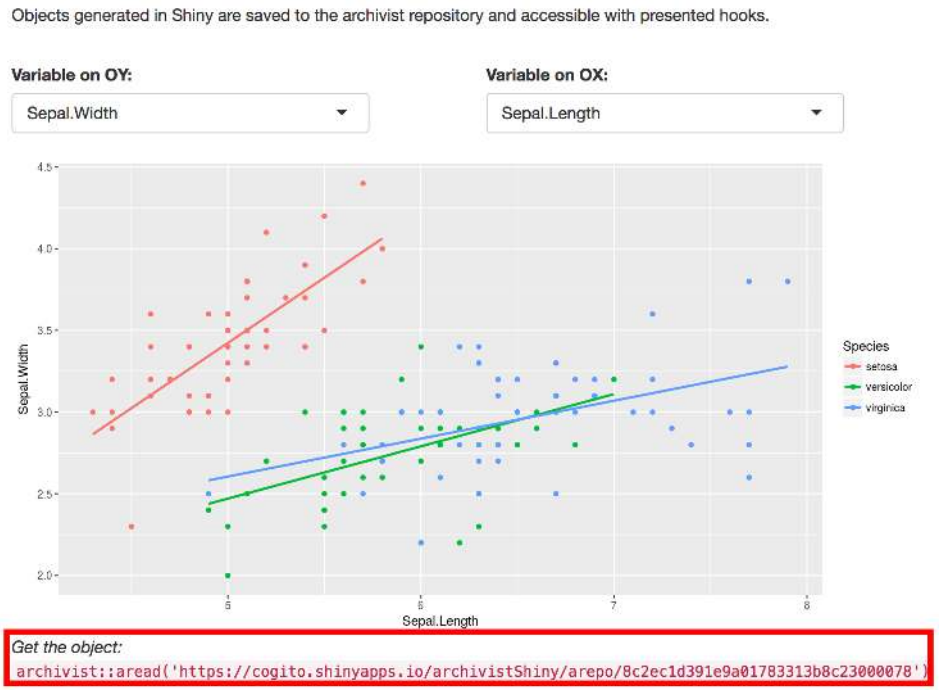


Figure 2: A screenshot from a Shiny application hosted under the link `https://cogito.shinyapps.io/archivistShiny`. The **archivist** hook is included below each plot.

3. Functionality

The key functionality of the **archivist** package is to manage copies of R objects, called artifacts, stored as binary files. Artifacts are stored in collections called repositories. Properties of artifacts and relations between artifacts are described by their tags.

Typical lifetime of the repository is presented in Figure 3. The local repository is created with the `createLocalRepo` function. It can be set as a default repository so that calls of the other **archivist** functions can be simplified. Once the repository is created, new R objects can be archived with the `saveToLocalRepo` function or can be removed with the `rmFromLocalRepo` function. Artifacts can be restored from the repository with `loadFromLocalRepo` function. One can also get all objects that match given criteria with the function named `searchInLocalRepo`. Both functions have wrappers called `aread` and

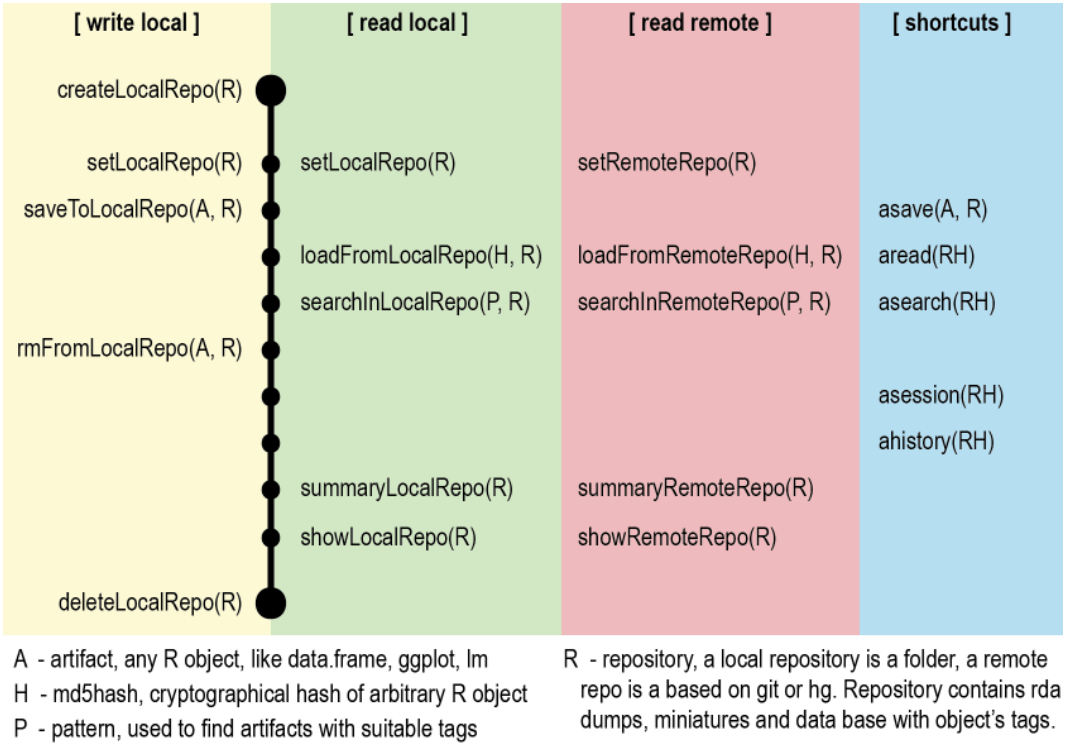


Figure 3: Overview of the most important functions related to a life-cycle of a repository and an artifact.

`asearch`, respectively, with the simplified and shorter interface. To summarise what kind of artifacts are in the repository one can use `summaryLocalRepo` or `showLocalRepo` functions. The repository can be removed with the `deleteLocalRepo` function.

Table 1 presents all functions available in the **archivist** package. These functions are divided into four core groups:

- Functions for repository management. In this group there are functions used to create a new empty repository, to create a repository as a copy of an existing local or GitHub repository, to backup an entire repository into a single *zip* file, to present summary statistics of objects stored in the repository and to delete existing repository.
- Functions for saving artifacts to a repository, loading artifacts from a repository and removing artifacts from a repository. Functions that show relations between artifacts, present artifacts' history or context in which they were created.
- Functions for searching for artifacts within a repository. Artifacts may be accessed through date of creation, a tag or a list of tags.
- Other features that do not fit previous categories.

In sections 3.1-3.4 each group of these functions is presented separately.

3.1. Repository management

	Local	**Remote**
Repository managment	createLocalRepo	
	setLocalRepo	setRemoteRepo
	deleteLocalRepo	
	showLocalRepo	showRemoteRepo
	summaryLocalRepo	summaryRemoteRepo
	zipLocalRepo	zipRemoteRepo
	copyLocalRepo	copyRemoteRepo
Artifacts management	saveToLocalRepo	
	rmFromLocalRepo	
	loadFromLocalRepo	loadFromRemoteRepo
	aread	aread
	asession	asession
	aformat	aformat
	ahistory	
	%a%	
Artifacts' exploration	searchInLocalRepo	searchInRemoteRepo
	asearch	asearch
	shinySearchInLocalRepo	
Extensions	restoreLibs	
	atrace	
	addHooksToPrint	
	createMDGallery	

Table 1: The list of functions available in the **archivist** package version 2.1.

A repository is a collection of artifacts and their meta-data. In this section you will find a list of functions for repository management (used to create a new empty repository, create a copy, present summary statistics or delete existing repository).

Technically, repository is a directory with the following structure (see Figure 4).

- A `backpack.db` file which contains an SQLite database. The database contains two tables with a structure presented in Figure 5. The table named *artifact* contains artifacts' MD5 hashes and basic information about the artifacts. The table called *tag* contains artifacts' tags. Since both artifacts and tags may be added into the database an unspecified number of times, each tag and artifact has one or more time points - one for each attempt to artifact's or tag's archiving to the repository.
- A subdirectory called `gallery` with artifacts' storage. Artifacts are stored as separate files. Names of files start with MD5 hashes of corresponding artifacts. Extensions correspond to formats in which artifacts are saved. The current implementation for R stores artifacts in the `rda` format, but it can be easily extended to handle other formats. Additionally, also an artifact's miniature is saved. For plots the default format for miniatures is raster file with `png` extension, for other objects it is a text file with `txt` extension (e.g., for data frames it contains first few rows).

A repository may be accessed in two ways.

- Local - in this case repository is identified by its path in the local file system. The repository is in the read-write mode. If the file system is shared (shared file system on HPC cluster, a Dropbox directory, a mounted folder on Network File System, Secure Shell Filesystem, etc.) multiple users may read and write into the repository at the same time.

- Remote - Currently **archivist** supports GitHub and Bitbucket repositories, but it can be easily extended to support any git or mercurial repository, see Section 4. Repository is identified by it's type (github/bitbucket), a username and the repository's name. The repository is accessible in read-only mode. Multiple users can read from such repository at the same time. In order to write to a remote repository one should either synchronize a local directory with GitHub/Bitbucket account or use a **archivist.github** package, which is **archivist**'s first extension (see Kosinski and Biecek 2016).

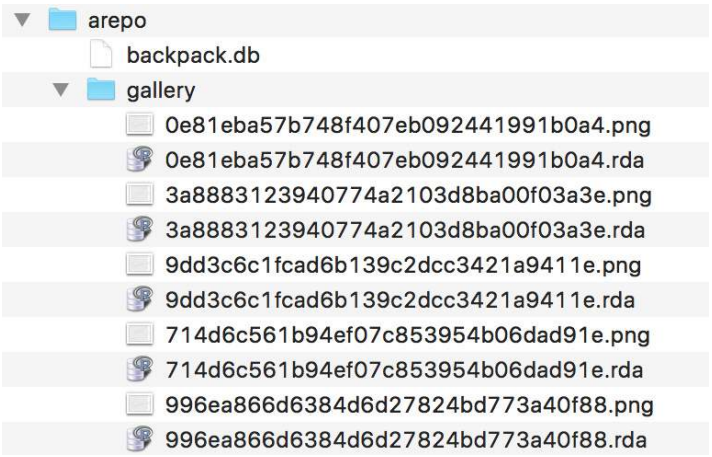


Figure 4: The structure of an example `arepo` repository. It contains database with objects' meta-data stored in an SQLite file `backpack.db` and a subfolder `gallery` with binary copies of R objects and their miniatures.

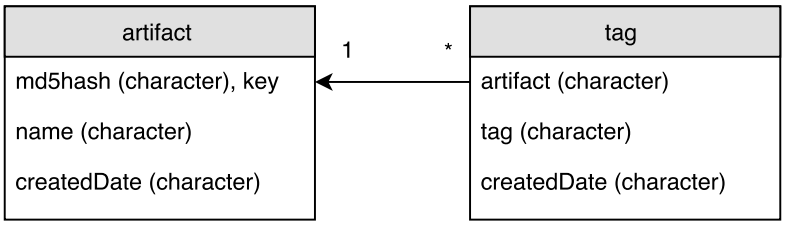


Figure 5: The Entity-Relationship Diagram that presents the structure of tables: *artifact* and *tag*, summarizing the relations between artifacts. The SQLite database with both tables is stored in `backback.db` file in the repository.

The logic behind this is as follows. Depending on the user's needs it is possible to create a single repository per project or per group of projects or keep all artifacts ever created in a single repository. Since (i) a local repository is accessible even without an Internet connection, (ii) the access is faster and (iii) there is both read and write access, it is easier to work with local repositories, which are just a directory identified by its path. If the user wants to share a repository with artifacts with a general public then he or she can publish the local repository on GitHub or Bitbucket or make it available as a subdirectory of an R package.

Creation of a new empty repository

The `createLocalRepo` function creates a new local repository. The `repoDir` argument points to a directory that will be used as a repository root. The directory will be created if it does not exist. The `default=TRUE` argument marks the newly created repository as a default one.

The directory may be specified either by global path or local path. The example below will create a repository named `arepo` in the current working directory.

```
repo <- "arepo"
createLocalRepo(repoDir = repo, default = TRUE)
```

Deletion of an existing repository

The `deleteLocalRepo` function deletes all artifacts, miniatures, the database with meta-data and the directory identified by the `repoDir` argument.

```
repo <- "arepo"
deleteLocalRepo(repoDir = repo)
```

Copying artifacts from other repositories

Functions `copyLocalRepo` and `copyRemoteRepo` copy selected artifacts from either local or remote (GitHub or Bitbucket) repository into a local repository. Artifacts to be copied are identified by their MD5 hashes.

In the example below the artifact identified by hash `7f3453331910e3f321ef97d87adb5bad` is copied along with its meta-data from remote GitHub repository `pbiecek/graphGallery` to the local repository `arepo`.

```
repo <- "arepo"
createLocalRepo(repoDir = repo, default = TRUE)
copyRemoteRepo(repoTo = repo,
    md5hashes = "7f3453331910e3f321ef97d87adb5bad",
    user = "pbiecek", repo = "graphGallery", repoType = "github")
```

Functions `zipLocalRepo` and `zipRemoteRepo` download all artifacts and create a single `zip` archive.

Showing repository's statistics

A repository is a collection of artifacts and their meta-data. Functions `summaryLocalRepo` and `summaryRemoteRepo` summarize basic statistics about artifacts in the repository. Functions `showLocalRepo` and `showRemoteRepo` list all MD5 hashes and artifact's meta-data.

Functions `show*Repo` take argument `method` which may be either `"tags"` (the result is a data frame with artifact's tags) or `"md5hashes"` (default, result is a data frame with artifact's MD5 hashes).

In the previous example we copied a single artifact from GitHub repository to the local one. The artifact is copied with its tags. In the example below we list all the tags within this single-artifact repository.

```
showLocalRepo(repoDir = repo, method = "tags")

##                            artifact
## 1 7f3453331910e3f321ef97d87adb5bad
## 2 7f3453331910e3f321ef97d87adb5bad
## 3 7f3453331910e3f321ef97d87adb5bad
## 4 7f3453331910e3f321ef97d87adb5bad
## 5 7f3453331910e3f321ef97d87adb5bad
## 6 7f3453331910e3f321ef97d87adb5bad
## 7 7f3453331910e3f321ef97d87adb5bad
## 8 7f3453331910e3f321ef97d87adb5bad
## 9 7f3453331910e3f321ef97d87adb5bad
##                                              tag         createdDate
## 1                                    format:rda 2016-12-31 15:50:59
## 2                                      name:pl1 2016-12-31 15:50:59
## 3                                      class:gg 2016-12-31 15:50:59
## 4                                  class:ggplot 2016-12-31 15:50:59
## 5                            labelx:Sepal.Length 2016-12-31 15:50:59
## 6                            labely:Petal.Length 2016-12-31 15:50:59
## 7                       date:2016-12-31 15:50:59 2016-12-31 15:50:59
## 8 session_info:0c325724f6118fdd80e6504204b72cfa 2016-12-31 15:50:59
## 9                                    format:png 2016-12-31 15:51:00
```

In the example below the function `summaryLocalRepo` is used to list summaries of artifacts in the repository called `graphGallery` which is attached to the **archivist** package. One can find information about dates on which artifacts were added, classes of artifacts and the total number of artifacts in the repository.

```
summaryLocalRepo(repoDir =
    system.file("graphGallery", package = "archivist"))

## Number of archived artifacts in Repository:  7
## Number of archived datasets in Repository:  3
## Number of various classes archived in Repository:
```

```
##             Number
## lm               3
## data.frame       2
## summary.lm       1
## gg               2
## ggplot           2
## Saves per day in Repository:
##             Saves
## 2016-02-07     5
## 2016-02-08    13
## 2016-03-04     3
## 2016-12-31     4
## 2017-11-21     2
```

Setting a default repository

In most of the cases we work with one repository per project. In such cases it is convenient to set a default local or remote repository. It can be done with `setLocalRepo` or `setRemoteRepo` functions. Look at the example below.

```
setRemoteRepo(user = "pbiecek", repo = "graphGallery", repoType = "github")
setLocalRepo(repoDir =
    system.file("graphGallery", package = "archivist"))
```

After setting a default repository, one can use the following functions

- `saveToLocalRepo`,
- `loadFromLocalRepo`, `loadFromRemoteRepo`,
- `rmFromLocalRepo`,
- `searchInLocalRepo`, `searchInRemoteRepo`,

without specification of `repoDir` or `user/repo/branch/subdir/repoType` arguments.
For example, the instruction below will add `iris` data frame to the default local repository.

```
data("iris")
saveToRepo(iris)

## [1] "ff575c261c949d073b2895b05d1097c3"
```

Another option for setting a default value for an argument is the function `aoptions()`. It sets the default value for any argument that is used by `archivist`. For example the instruction below sets the default value for `repoType` to `"github"`.

```
aoptions("repoType", "github")

## [1] "github"
```

3.2. Artifact management

An artifact is an R object with its meta-data. Artifacts are stored in repositories. Key functions for artifact's management are functions for saving, loading and removing artifacts from a repository.

Saving an R object into a repository

The `saveToLocalRepo` function saves any R object into the selected repository. It stores in the repository both the object and its tags. Some tags and some meta-data are extracted in an automated way. The `saveToLocalRepo` function recognizes the class of the artifact and extracts tags typical for that class. It is possible to add support for a new class of objects or change list of tags extracted for selected classes, just extend the generic function `extractTags()`. Table 2 lists classes that are recognized in the current version of the package and lists tags that are derived automatically from objects of a given class. For other classes the following attributes are extracted: name, creation time and MD5 hash.

Artifact's class	Tags
lm	date, name, class, coefname, rank, df.residual
survfit	date, name, class, strata, type, n, conf.type, conf.int
ggplot	date, name, class, labelx, labely
twins	date, name, class, ac
partition	date, name, class, objective, memb.exp, coeff, k.crisp, conv, clus.avg.widths, avg.width
qda	date, name, class, terms, N, lev, counts, prior, ldet
lda	date, name, class, N, lev, counts, prior, svd
htest	date, name, class, alternative, method, data.name, null.value, statistic, parameter, p.value, intervals, estimate
data.frame	date, name, class, varname
summary.lm	date, name, class, sigma, df, r.squared, adj.r.squared, fstatistic, fstatistic.df
glmnet	date, name, class, dim, nulldev, npasses, offset, nobs
default	date, name, class

Table 2: Tags that are automatically extracted from objects depending on the object's class. See `?Tags` for more details.

The `saveToLocalRepo` function takes at least two arguments: `artifact` - an R object which is about to be saved and `repoDir` which is a path to the local repository. The process of adding an R object to the repository triggers a chain of actions listed below. By setting some arguments of `saveToLocalRepo` to `FALSE` some of these actions may be skipped.

- The name of the object is derived and stored as the object's tag `name:xxx`. It may be useful when searching for an object. One can search for all objects that had a specific name with `asearch(pattern="name:iris")`.
- An MD5 hash is calculated for the object with the use of **digest** package. Then the object is saved as a binary file named `md5hash.rda` with the use of `save` function.
- If there is any dependent object, it is saved separately to the repository (e.g., for object of class `gg` or `lm` the `data` slot is extracted from the object and saved separately. Additionally a tag `relationWith:xxx` is added, where `xxx` is the MD5 hash of the dataset).
- The current session info, with the list of versions of attached packages, is saved to the repository. The session info is linked to the artifact. The link is a tag of the form `sessionInfo:xxx`, where `xxx` stands for MD5 hash of the object with session info.
- A set of tags is extracted automatically and these tags are saved to the repository. See Table 2 for the list of tags that are automatically derived. Tags extracted for a given class are defined by the generic `extractTags` function.
- Additional tags specified by a user (with the `userTags` argument) are saved to the repository as well.
- A miniature for the object is created – for plots it is a png file while for data frames or models it is a text description of the object.

The following example creates a plot of the class `gg` and saves the object into the repository. Plots created with the use of **ggplot2** package are objects and can be serialized in the same way as any other R objects (see Wickham 2009). A hash of the recorded object is returned. In the example below it is `11127cc6ce69a89d11d0e30865a33c13`. By default, the related data object is also saved. In this case the dependent object is a dataset `iris` which is saved with the hash `ff575c261c949d073b2895b05d1097c3`.

```
library("ggplot2")
repo <- "arepo"
pl <- qplot(Sepal.Length, Petal.Length, data = iris)
saveToLocalRepo(pl, repoDir = repo)

## [1] "bca8c0e007cf3579a8bbac6fe17a0206"
## attr(,"data")
## [1] "ff575c261c949d073b2895b05d1097c3"
```

The function `saveToLocalRepo` extracts additional tags such as the name of the original object (here: `name:pl`), its class (`class:gg`), labels on OX and OY axes (`labelx:Sepal.Length`) and MD5 hash of the data object. These tags are listed if we use `showLocalRepo` function on the repository.

```
showLocalRepo(repoDir = repo, "tags")
```

```
##                            artifact
## 1  7f3453331910e3f321ef97d87adb5bad
## 2  7f3453331910e3f321ef97d87adb5bad
## 3  7f3453331910e3f321ef97d87adb5bad
## 4  7f3453331910e3f321ef97d87adb5bad
## 5  7f3453331910e3f321ef97d87adb5bad
## 6  7f3453331910e3f321ef97d87adb5bad
## 7  7f3453331910e3f321ef97d87adb5bad
## 8  7f3453331910e3f321ef97d87adb5bad
## 9  7f3453331910e3f321ef97d87adb5bad
## 10 bca8c0e007cf3579a8bbac6fe17a0206
## 11 bca8c0e007cf3579a8bbac6fe17a0206
## 12 bca8c0e007cf3579a8bbac6fe17a0206
## 13 bca8c0e007cf3579a8bbac6fe17a0206
## 14 bca8c0e007cf3579a8bbac6fe17a0206
## 15 bca8c0e007cf3579a8bbac6fe17a0206
## 16 bca8c0e007cf3579a8bbac6fe17a0206
## 17 bd29400655e0562b4435fc6b2046033d
## 18 bca8c0e007cf3579a8bbac6fe17a0206
## 19 ff575c261c949d073b2895b05d1097c3
## 20 ff575c261c949d073b2895b05d1097c3
## 21 ff575c261c949d073b2895b05d1097c3
## 22 bca8c0e007cf3579a8bbac6fe17a0206
##                                                 tag         createdDate
## 1                                    format:rda 2016-12-31 15:50:59
## 2                                      name:pl1 2016-12-31 15:50:59
## 3                                      class:gg 2016-12-31 15:50:59
## 4                                  class:ggplot 2016-12-31 15:50:59
## 5                            labelx:Sepal.Length 2016-12-31 15:50:59
## 6                            labely:Petal.Length 2016-12-31 15:50:59
## 7                       date:2016-12-31 15:50:59 2016-12-31 15:50:59
## 8  session_info:0c325724f6118fdd80e6504204b72cfa 2016-12-31 15:50:59
## 9                                    format:png 2016-12-31 15:51:00
## 10                                   format:rda 2017-11-25 23:34:38
## 11                                     name:pl 2017-11-25 23:34:38
## 12                                     class:gg 2017-11-25 23:34:38
## 13                                 class:ggplot 2017-11-25 23:34:38
## 14                           labelx:Sepal.Length 2017-11-25 23:34:38
## 15                           labely:Petal.Length 2017-11-25 23:34:38
## 16                      date:2017-11-25 23:34:38 2017-11-25 23:34:38
## 17                                   format:rda 2017-11-25 23:34:38
## 18 session_info:bd29400655e0562b4435fc6b2046033d 2017-11-25 23:34:38
## 19                                   format:rda 2017-11-25 23:34:38
## 20                                   format:txt 2017-11-25 23:34:38
```

```
## 21 relationWith:bca8c0e007cf3579a8bbac6fe17a0206 2017-11-25 23:34:38
## 22                                   format:png 2017-11-25 23:34:39
```

By default, for each artifact also it's context, i.e., session info, is saved. It can be accessed with the function `asession()`. See the example below. Such additional information may be very useful if we cannot replicate previous results and we are in the need of recovering the exact versions of important packages, which can be done with `restoreLibs` function.

```
asession("7f3453331910e3f321ef97d87adb5bad")

## Session info -------------------------------------

##  setting  value
##  version  R version 3.3.2 (2016-10-31)
##  system   x86_64, darwin13.4.0
##  ui       RStudio (1.0.44)
##  language (EN)
##  collate  en_US.UTF-8
##  tz       Europe/Warsaw
##  date     2016-12-31

## Packages -----------------------------------------

##  package       * version    date
##  archivist     * 2.1.1      2016-11-18
##  assertthat      0.1        2013-12-06
##  backports       1.0.4      2016-10-24
##  bitops          1.0-6      2013-08-17
##  colorspace      1.3-1      2016-11-18
##  DBI             0.5-1      2016-09-10
##  devtools      * 1.12.0     2016-06-24
##  digest          0.6.10     2016-08-02
##  dplyr         * 0.5.0      2016-06-24
##  evaluate        0.10       2016-10-11
##  fields          8.4-1      2016-05-05
##  foreign         0.8-67     2016-09-13
##  geosphere       1.5-5      2016-06-15
##  ggmap         * 2.7        2016-12-28
##  ggplot2       * 2.2.0.9000 2016-12-28
##  ggthemes      * 3.3.0      2016-11-24
##  gridExtra     * 2.2.1      2016-02-29
##  gtable          0.2.0      2016-02-26
##  highr           0.6        2016-05-09
##  htmltools     * 0.3.5      2016-03-21
##  httpuv          1.3.3      2015-08-04
##  httr          * 1.2.1      2016-07-03
```

```
##  jpeg            0.1-8      2014-01-23
##  jsonlite        1.1        2016-09-14
##  knitr         * 1.15       2016-11-09
##  labeling        0.3        2014-08-23
##  lattice       * 0.20-34    2016-09-06
##  latticeExtra  * 0.6-28     2016-02-09
##  lazyeval        0.2.0      2016-06-12
##  lubridate     * 1.6.0      2016-09-13
##  magrittr        1.5        2014-11-22
##  mapproj         1.2-4      2015-08-03
##  maps            3.1.1      2016-07-27
##  maptools        0.8-40     2016-11-15
##  memoise         1.0.0      2016-01-29
##  mime            0.5        2016-07-07
##  munsell         0.4.3      2016-02-13
##  plyr            1.8.4      2016-06-08
##  png             0.1-7      2013-12-03
##  proto           1.0.0      2016-10-29
##  R6              2.2.0      2016-10-05
##  RColorBrewer  * 1.1-2      2014-12-07
##  Rcpp            0.12.8     2016-11-17
##  RCurl           1.95-4.8   2016-03-01
##  readr         * 1.0.0      2016-08-03
##  reshape2        1.4.2      2016-10-22
##  RgoogleMaps     1.4.1      2016-09-18
##  rjson           0.2.15     2014-11-03
##  RJSONIO         1.3-0      2014-07-28
##  rmarkdown       1.2        2016-11-21
##  rprojroot       1.1        2016-10-29
##  rsconnect       0.6        2016-11-21
##  RSQLite         1.0.0      2014-10-25
##  rstudioapi      0.6        2016-06-27
##  rworldmap     * 1.3-6      2016-02-03
##  scales        * 0.4.1      2016-11-09
##  shiny         * 0.14.2     2016-11-01
##  SmarterPoland * 1.7        2016-03-28
##  sp            * 1.2-3      2016-04-14
##  spam            1.4-0      2016-08-30
##  stringi         1.1.2      2016-10-01
##  stringr         1.1.0      2016-08-19
##  tibble          1.2        2016-08-26
##  tidyr         * 0.6.0      2016-08-12
##  withr           1.0.2      2016-06-20
##  xtable          1.8-2      2016-02-05
##  yaml            2.1.14     2016-11-12
##  source
```

```
##  local (pbiecek/archivist@NA)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.2)
##  cran (@0.5-1)
##  CRAN (R 3.3.0)
##  cran (@0.6.10)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.2)
##  CRAN (R 3.3.0)
##  Github (dkahle/ggmap@c6b7579)
##  Github (hadley/ggplot2@f6f9f9d)
##  cran (@3.3.0)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.0)
##  cran (@1.2.1)
##  CRAN (R 3.3.0)
##  cran (@1.1)
##  CRAN (R 3.3.2)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.2)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.0)
##  cran (@1.6.0)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.2)
##  CRAN (R 3.3.0)
##  cran (@0.5)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.0)
##  cran (@2.2.0)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.2)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.0)
##  cran (@1.4.2)
```

```
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.2)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.2)
##  CRAN (R 3.1.2)
##  cran (@0.6)
##  CRAN (R 3.3.0)
##  cran (@0.4.1)
##  cran (@0.14.2)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.0)
##  cran (@1.1.2)
##  CRAN (R 3.3.0)
##  cran (@1.2)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.0)
##  CRAN (R 3.3.0)
##  cran (@2.1.14)
```

Serialization of an object creation event into repository

The **archivist** provides a new operator %a% that works as the extended pipe operator %>% from the **magrittr** package (see Bache and Wickham 2014, for more details). In addition, it saves the resulting object to the default **archivist** repository together with the function call and its parameters. The default repository should be set first, see the `setLocalRepo` function for instructions how to do this. With this functionality it is possible to trace function calls and extract pedigree for some artifacts.

```
library("archivist")
createLocalRepo("arepo", default = TRUE)

## Directory arepo does exist and contain the backpack.db file.  Use force=TRUE
to reinitialize.

library("dplyr")
iris %a%
    dplyr::filter(Sepal.Length < 6) %a%
    lm(Petal.Length~Species, data=.) %a%
    summary() -> tmp
```

How to recreate an object's history? The function `ahistory` extracts the chain of calls that leads to the selected object. As an argument one can specify either an object's value or its

MD5 hash. The value of `ahistory` function is a `data.frame` with two columns – first contains function calls while second contains MD5 hashes of partial results.

In the example above, a chain of three operations converts input `iris` data into the `tmp` object. The **dplyr** package (see Wickham and Francois 2015) has to be loaded first since the function `filter` is used in this example. Following lines present the chain of consecutive transformations that are recorded in the repository.

```
ahistory(tmp)

##    iris                                    [ff575c261c949d073b2895b05d1097c3]
## -> filter(Sepal.Length < 6)                [d3696e13d15223c7d0bbccb33cc20a11]
## -> lm(Petal.Length ~ Species, data = .)    [990861c7c27812ee959f10e5f76fe2c3]
## -> summary()                               [050e41ec3bc40b3004bc6bdd356acae7]

ahistory(md5hash = "050e41ec3bc40b3004bc6bdd356acae7")

##    iris                                    [ff575c261c949d073b2895b05d1097c3]
## -> filter(Sepal.Length < 6)                [d3696e13d15223c7d0bbccb33cc20a11]
## -> lm(Petal.Length ~ Species, data = .)    [990861c7c27812ee959f10e5f76fe2c3]
## -> summary()                               [050e41ec3bc40b3004bc6bdd356acae7]
```

In order to restore an object's pedigree all partial results must be saved in a repository. So this option will work only for objects created by a chain of calls that use the `%a%` operator.

Loading an object from a repository

To read an object from repository we may consider the following four scenarios.

- We know the object's MD5 hash and the object is in a local directory.
- We know the object's MD5 hash and the object is in a remote repository, i.e., on GitHub or BitBucket.
- We do not know the hash but we know some properties of the object so we need to find it first by its tags. The object is in a local repository.
- As above, but the object is in a remote repository.

If we know the MD5 hash of the requested artifact, we can directly load the object from the repository and in this section we are going to show how this can be done. If we do not know the MD5 hash, then we need to use one of `search*` functions presented in Section 3.3.

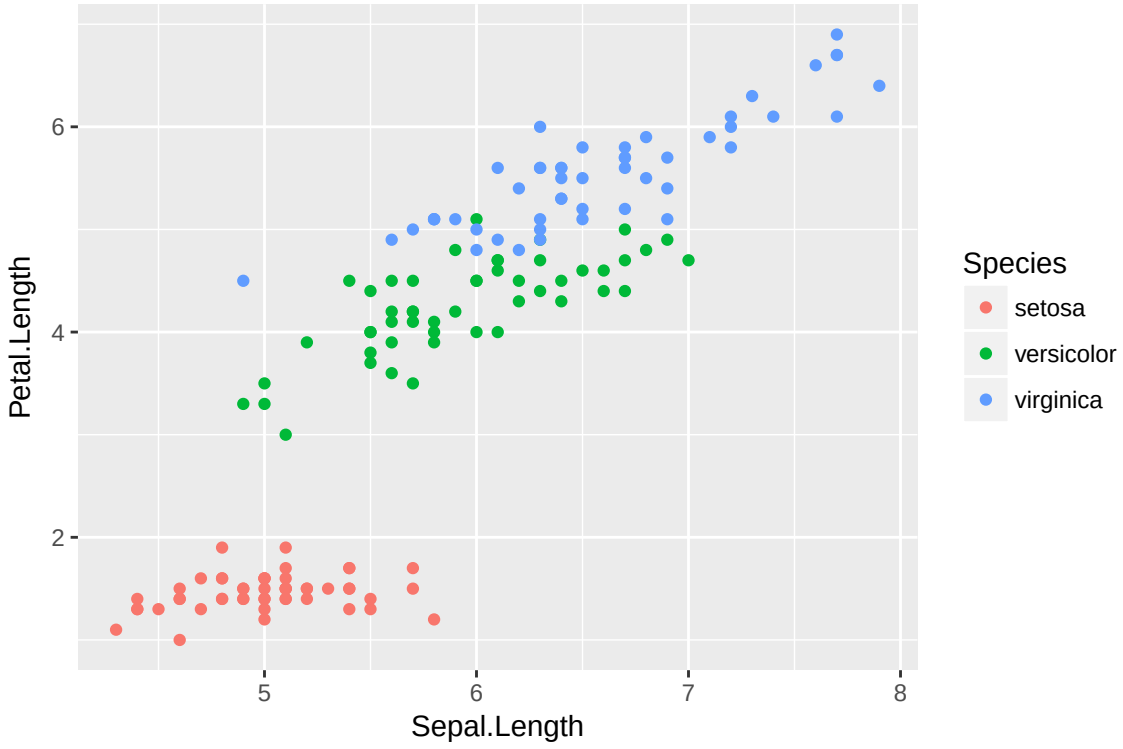
Functions `loadFromLocalRepo` and `loadFromRemoteRepo` read artifacts from either local or remote repositories. The local repository is defined by a path to it's root; remote repository is defined by it's type (currently `"github"` (default) or `"bitbucket"`), the username, repository's name and a subdirectory within the repository. In both functions the argument `value` specifies whether the function should return the object by value (`value=TRUE`) or it should load the object into the namespace with its original name (`value=FALSE`).

For the purpose of this example we have created a repository `graphGallery`, with two objects: a plot and a regression model. The repository is available both on GitHub (see `https://github.com/pbiecek/graphGallery`) and within the **archivist** package (see the `graphGallery` directory). Two archived objects have `7f3453331910e3f321ef97d87adb5bad` and `2a6e492cb6982f230e48cf46023e2e4f` hashes respectively.

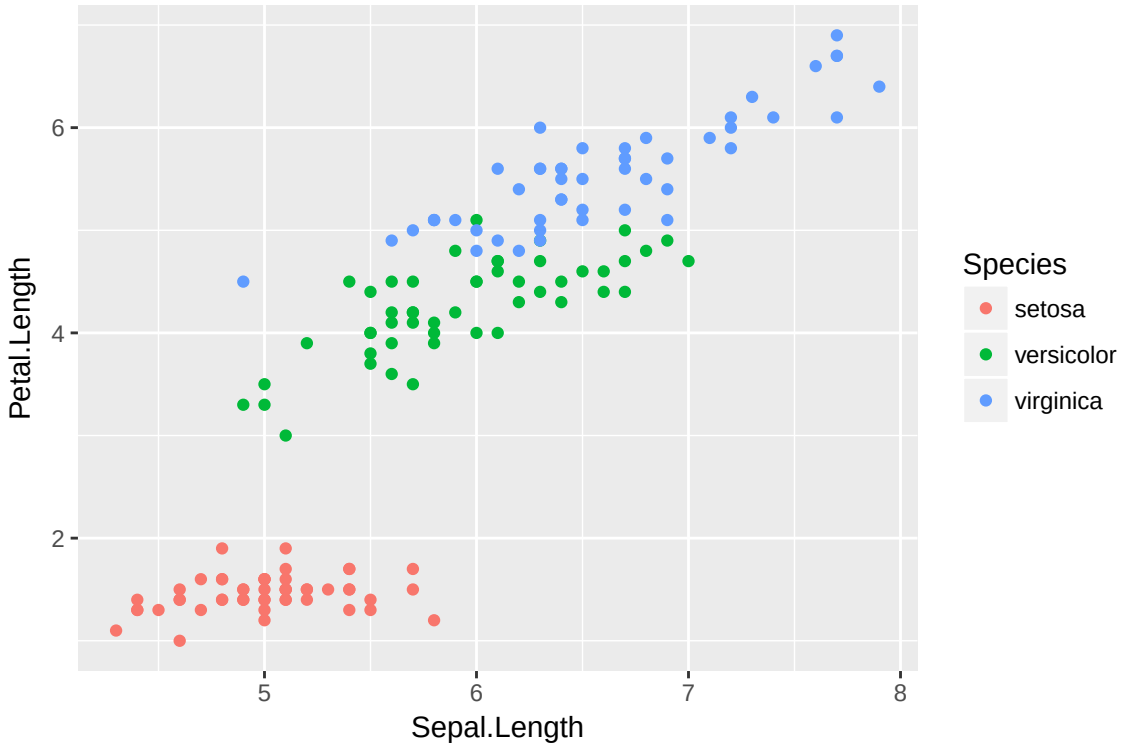
The full MD5 hash of an artifact is a 32-characters-long string but it is enough to set only the first few characters. In the example below it is enough to use `"7f34533"` prefix to load an artifact with the `"7f3453331910e3f321ef97d87adb5bad"` hash. There is only one artifact with prefix `"7f34533"` in its MD5 hash. If there is more, all that match the prefix are returned. Note that one should not use this feature unless is sure that new objects with colliding hashes will not be added. For small repositories conflicts are unlikely even for first five characters, but be careful when using this feature.

Both following instructions retrieve an R object from GitHub, load it into R session and make it accessible for further processing. In this case it is a `ggplot2` object so after being loaded the `print` function is triggered and a plot is generated (see Figure 6). Note that by default the GitHub is assumed, but this may be changed with the parameter `repoType`.

```
loadFromRemoteRepo("7f3453331910e3f321ef97d87adb5bad",
    repo = "graphGallery", user = "pbiecek", value = TRUE)
```



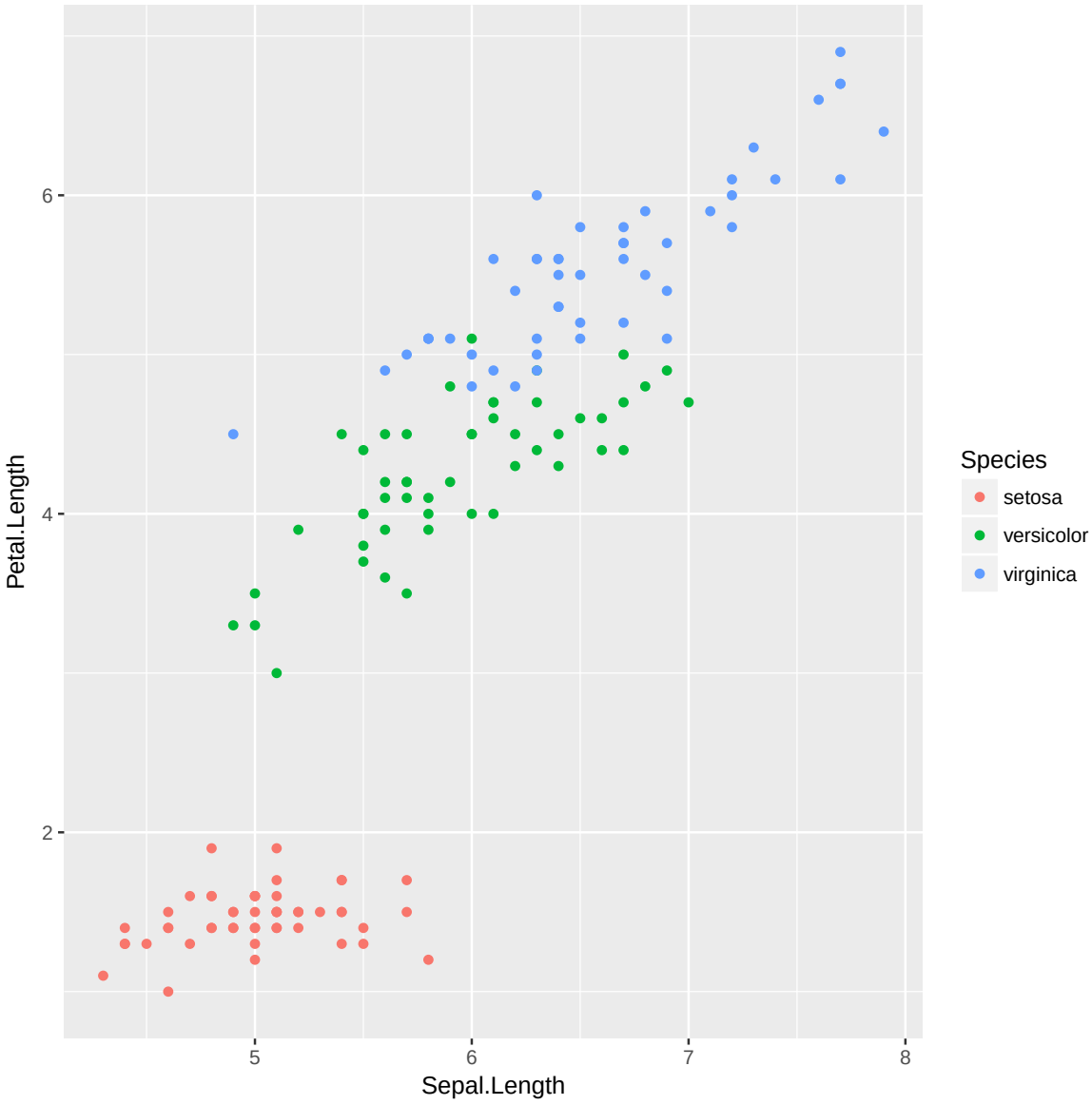

```
loadFromLocalRepo("7f34533",
    system.file("graphGallery", package = "archivist"), value = TRUE)
```



The `aread` function is a wrapper over `loadFromRemoteRepo` with more compact form. Shorter instructions and shorter code snippets might be placed in a figure or table caption. The single line below reads an object with the `7f34533...` hash from `graphGallery` GitHub repository that is owned by the `pbiecek` user.

```
archivist::aread("pbiecek/graphGallery/7f3453331910e3f321ef97d87adb5bad")
```

The following instructions retrieve the same R object but this time from the `graphGallery` repository attached to the **archivist** package. Note that the default repository is set first with the `setLocalRepo` function.



The use of MD5 hashes as objects identifiers has some advantages. In some use cases we may be restricted to use only models approved by some authority. For example, due to some hypothetical regulatory issues in production it might be advisable to use only a specific version of a model (such as credit scoring model or some forecasting model).

In the **archivist** package all objects have their cryptographical hash calculated with the MD5 algorithm. One can use the `digest` function to validate the object's MD5 hash at any moment. One can also call an object the from repository by its MD5 hash. Having a list of MD5 hashes of *allowed* objects one can validate their identity.

In the example below the downloaded regression model is digested to confirm its identity.

```
setLocalRepo(system.file("graphGallery", package = "archivist"))
model <- aread("2a6e492cb6982f230e48cf46023e2e4f")
digest::digest(model)
```

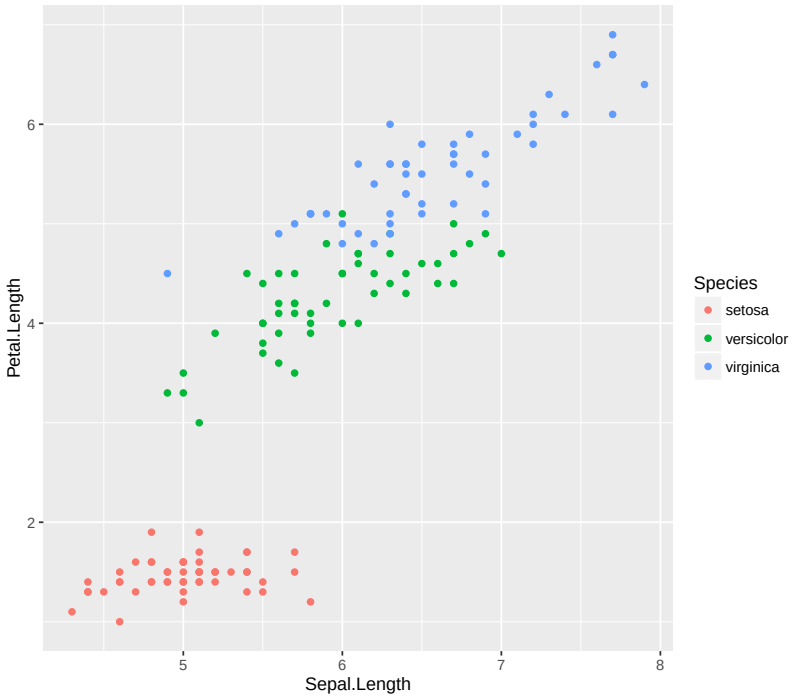


Figure 6: Object with the hash `7f3453331910e3f321ef97d87adb5bad` available in the repository called `graphGallery` of the GitHub user `pbiecek` and in the archivist package. It may be retrieved with the `aread` function.

```
## [1] "2a6e492cb6982f230e48cf46023e2e4f"
```

Removal of an object from a repository

To remove an artifact from a repository one can use the `rmFromLocalRepo` function.

In the example below the artifact `92ada1e052d4d963e5787bfc9c4b506c` and all its tags are removed from the repository called `repo`.

```
rmFromLocalRepo("7f3453331910e3f321ef97d87adb5bad", repoDir = repo)
```

A list of artifact's hashes that should be removed may be obtained with the `search*` function. The example below searches for all artifacts older than 30 days and removes them from the `repo` repository.

```
obj2rm <- searchInLocalRepo(list(dateFrom = "2010-01-01",
    dateTo = Sys.Date()-30), repoDir = repo)
rmFromLocalRepo(obj2rm, repoDir = repo, many = TRUE)
```

It is also possible to remove many artifacts with one call. Broader examples of this function are explained in the package manual page accessed from R with `?rmFromLocalRepo`.

3.3. Search for an artifact and explore the repository

One of the advantages of the **archivist** package is the automated derivation of artifact's tags and meta-data. It is useful when one wants to find previously calculated results in a large collection of R objects. Relations between artifacts are useful when we want to process the structure dependencies between artifacts. Below we present a list of functions for searching for artifacts on the basis of their properties.

Search in a local or remote repository

If we do not know the MD5 hashes of artifacts that are of our interest, we can find them with the use of `search*` functions.

Searching within a local repository and a remote repository is very similar. Functions `searchInLocalRepo` or `searchInRemoteRepo` differ only in the way in which the repository is specified.

In both functions the `pattern` argument may be either a tag (name, class, varname or other) or a date period in which given artifact was created. Hashes of all artifacts that meet all criteria (i.e., were created within a given time interval or have a given tag attached) are returned.

For example, the following command retrieves MD5 hashes of all objects of the class `gg` from the `pbiecek/graphGallery` repository.

```
searchInLocalRepo(pattern = "class:gg",
    repoDir = system.file("graphGallery", package = "archivist"))

## [1] "7f3453331910e3f321ef97d87adb5bad"
## [2] "369227e67f9164dcbe934dadf2b53cc2"
```

To get a list of artifacts created within a given date range one can use following instruction.

```
searchInLocalRepo(pattern = list(dateFrom = "2016-01-01",
    dateTo = "2016-02-07"),
    repoDir = system.file("graphGallery", package = "archivist"))

## [1] "d9313a0de3e2980201a8971e3384ff26"
## [2] "ff575c261c949d073b2895b05d1097c3"
## [3] "2a6e492cb6982f230e48cf46023e2e4f"
## [4] "93ecfdf1436932e2860c6dbdf2abc2ad"
## [5] "afb2550d0f886f0cf3b050f04c5cd4f8"
```

The `searchInLocalRepo` and `searchInRemoteRepo` functions allow to use more than one searching criteria. Additional argument `intersect` specifies if the resulting objects have to met all or any of the search criteria.

```
searchInLocalRepo(pattern=c("class:gg", "labelx:Sepal.Length"),
    repoDir = system.file("graphGallery", package = "archivist"))
```

```
## [1] "369227e67f9164dcbe934dadf2b53cc2"
## [2] "7f3453331910e3f321ef97d87adb5bad"
```

These two functions return MD5 hashes of artifacts. In order to load these artifacts from repository one needs to use either `loadFrom*Repo` or `aread` functions. Since both operations are usually performed together (search for MD5 hashes of artifacts by their tag / load artifacts with given MD5 hashes), one can use the `asearch` function which retrieves MD5 hashes and returns a list with values of artifacts that meet all selected criteria.

Retrieval of a list of R objects with given tags

When working in a team or for a longer period of time, one produces a lot of partial results and it becomes harder and harder to trace what kind of analyses were conducted in the past and where are the results.

The **archivist** extracts meta-data from R objects in the very same moment they are archived in a repository. For many researchers objects are so valuable, due to their pedigree and meta-data, that they can be regarded as artifacts. Having such additional meta-data it is easier to search for previously generated partial results, e.g., by specifying what kind of model with which variables we are looking for.

For example, the code below retrieves all objects of the `lm` class with the `Sepal.Length` variable from within a list of dependent variables. In this repository only two artifacts (here `lm` models) match both conditions.

The following instruction searches within the default local repository.

```
setLocalRepo(system.file("graphGallery", package = "archivist"))
models <- asearch(patterns = c("class:lm", "coefname:Sepal.Length"))
```

Below is the code that searches within the GitHub repository.

```
models <- asearch("pbiecek/graphGallery",
    patterns = c("class:lm", "coefname:Sepal.Length"))
lapply(models, coef)

## $`18a98048f0584469483afb65294ce3ed`
##  (Intercept) Sepal.Length
##    -7.101443     1.858433
##
## $`2a6e492cb6982f230e48cf46023e2e4f`
##       (Intercept)      Sepal.Length Speciesversicolor
##        -1.7023422         0.6321099         2.2101378
##  Speciesvirginica
##         3.0900021
```

The following instruction retrieves all artifacts of the `gg` class (created with the package **ggplot2**) with label `Sepal.Length` on the X axis. Two objects are returned as a result. They are plotted together by the `grid.arrange` function from **gridExtra** package (see Auguie 2015).

```
plots <- asearch(patterns = c("class:gg", "labelx:Sepal.Length"))
length(plots)

## [1] 2
```

```
library("gridExtra")
do.call(grid.arrange, plots)
```

Result of these instructions is presented in Figure 7.

Interactive search in a local repository

For local repositories, it is also possible to explore the repository interactively with the `shinySearchInLocalRepo` function. This function launches a Shiny application (see Chang, Cheng, Allaire, Xie, and McPherson 2015) which is dynamically created and which allows for interactive specification of tags and sorting criteria. See Figure 8 with an example screenshot of this application.

In the text box area one can specify tags that filter out objects presented on the right panel. Only miniatures of objects that meet all these criteria are presented. Additionally, the instruction `sort:key` sorts the artifacts along the key. For example, use `"sort:createdDate"` to sort miniatures along the date of creation of the object.

```
arepo <- system.file("graphGallery", package = "archivist")
shinySearchInLocalRepo(arepo)
```

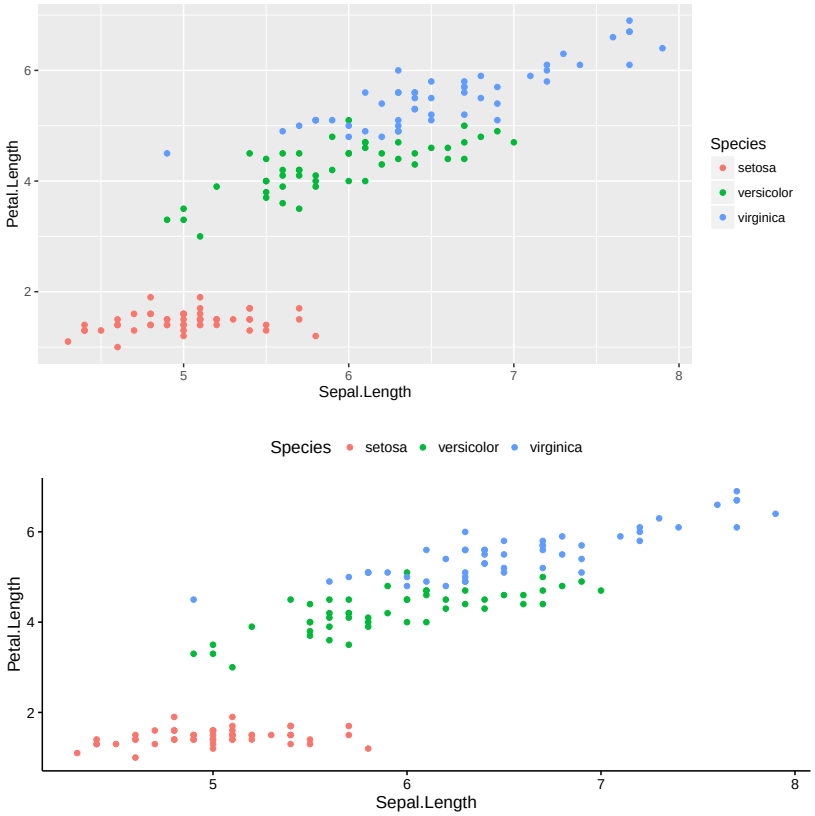


Figure 7: There are two objects of the class `gg` with annotation `Sepal.Length` on the X axis in the GitHub `pbiecek/graphGallery` repository. All objects in a repository that meet a set of conditions may be retrieved with the `asearch` function. Instructions how to extend the list of tags are in Section 3.2.

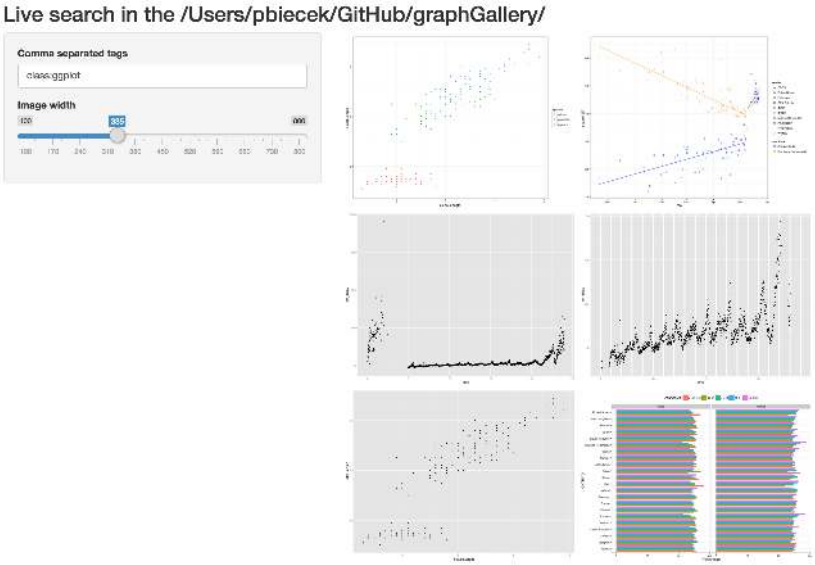


Figure 8: Model screen of a Shiny application produced by `shinySearchInLocalRepo` function. The application helps in searching for artifacts with given tags within a selected repository.

3.4. Extensions

The **archivist** package is designed as a multi-purpose manager of objects. In this section we present some specific extensions.

Archiving all results of a specific function

The `trace()` function from the **base** package allows to insert a specific instruction to the body of a selected function. It can be used for example to call `saveToLocalRepo()` function at the end of a selected function.

In the example below we modify the `lm()` function so that after it's each execution the created `lm` model is automatically added to the default local repository `allModels`.

```
library("archivist")
createLocalRepo("allModels", default = TRUE)
atrace("lm", "z")

## Tracing function "lm" in package "stats"

## [1] "lm"

lm(Sepal.Length ~ Sepal.Width, data=iris)

## Tracing lm(Sepal.Length ~ Sepal.Width, data = iris) on exit
##
## Call:
## lm(formula = Sepal.Length ~ Sepal.Width, data = iris)
##
## Coefficients:
## (Intercept)  Sepal.Width
##      6.5262      -0.2234

sapply(asearch("class:lm"), BIC)

## 42fcf77af2c40f70c445cbba513aeabd
##                         381.0236
```

*Integration with the **knitr** package*

The **knitr** package is a tool that transforms a mixture of R code and descriptions in natural language into a md, html or pdf report. Moreover the produced report contains results generated by the included R code. On one hand reader knows that presented results are generated by presented code. On the second hand the author does not waste time on coping the results, since they are automatically included in the output. Results included in a report are usually plots or tables. In such form they cannot be loaded from the pdf/html file directly to R. The **archivist** package records objects and makes them easier to access through local, GitHub or BitBucket repositories.

The function `addHooksToPrint` combines these two tools. A call to this function should be included on the beginning of a **knitr** report. It creates a new generic `print` functions for classes specified by the `class` argument. These functions save objects to the repository and add corresponding hooks to the report after every attempt to `print` the object. Hooks are short instructions on how the recorded objects can be accessed.

An example is presented in the report http://bit.ly/1nW9Cvz. Part of this report is presented in Figure 1. On the beginning there is a snippet presented below. It automatically adds hooks to the html report for all objects of classes `ggplot` or `data.frame`.

```
addHooksToPrint(class=c("ggplot", "data.frame"),
                  repoDir = "arepo",
                  repo = "Eseje", user = "pbiecek", subdir = "arepo")
```

As a result, just before each plot, there are automatically created hooks to corresponding objects e.g., `archivist::aread("pbiecek/Eseje/arepo/24ea7c04b861083d4bf56eee1c5a17b7")`. These hooks serve also as links to the corresponding R objects.

The biggest advantage of this integration is that a single call to `addHooksToPrint` is needed to enrich the **knitr** report in **archivist** hooks for all interesting objects.

Gallery of artifacts in the repository

Information about artifacts is stored in an SQLite database in the `backpack.db` file. The `createMDGallery` function creates a single markdown file with gallery of all artifacts in the repository.

Such gallery, if saved as file named `readme.md`, will automatically list all artifacts with miniatures and tags in the GitHub web portal user interface. See an example gallery at http://bit.ly/1Q62Tpz. This gallery was created with the following instruction. A part of the result is presented in Figure 9.

```
createMDGallery("arepo/readme.md",
           repo = "Eseje", user = "pbiecek", subdir = "arepo",
           addMiniature = TRUE, addTags = TRUE)
```

Support for other repositories, other languages and other formats

The current implementation of **archivist** supports local, GitHub and BitBucket repositories. The package is implemented in R and saves artifacts in the `rda` format.

In order to support other repositories one can extend the function `getRemoteHook`. It is used internally by other **archivist** functions to generate URL addresses to files in remote repositories. In order to support other repositories it's enough to extend this function.

All metadata related to artifacts is sorted in an SQLite database in `backpack.db` file. This database can be accessed from other languages. Objects are stored as files and can be added in different formats. Each artifact has an additional tag `format:xxx` that specifies in which format the artifact is saved, one artifact can be saved in more than one format. Currently artifacts are stored as `rda` files. In order to save objects in other formats, like `json` or `csv`,

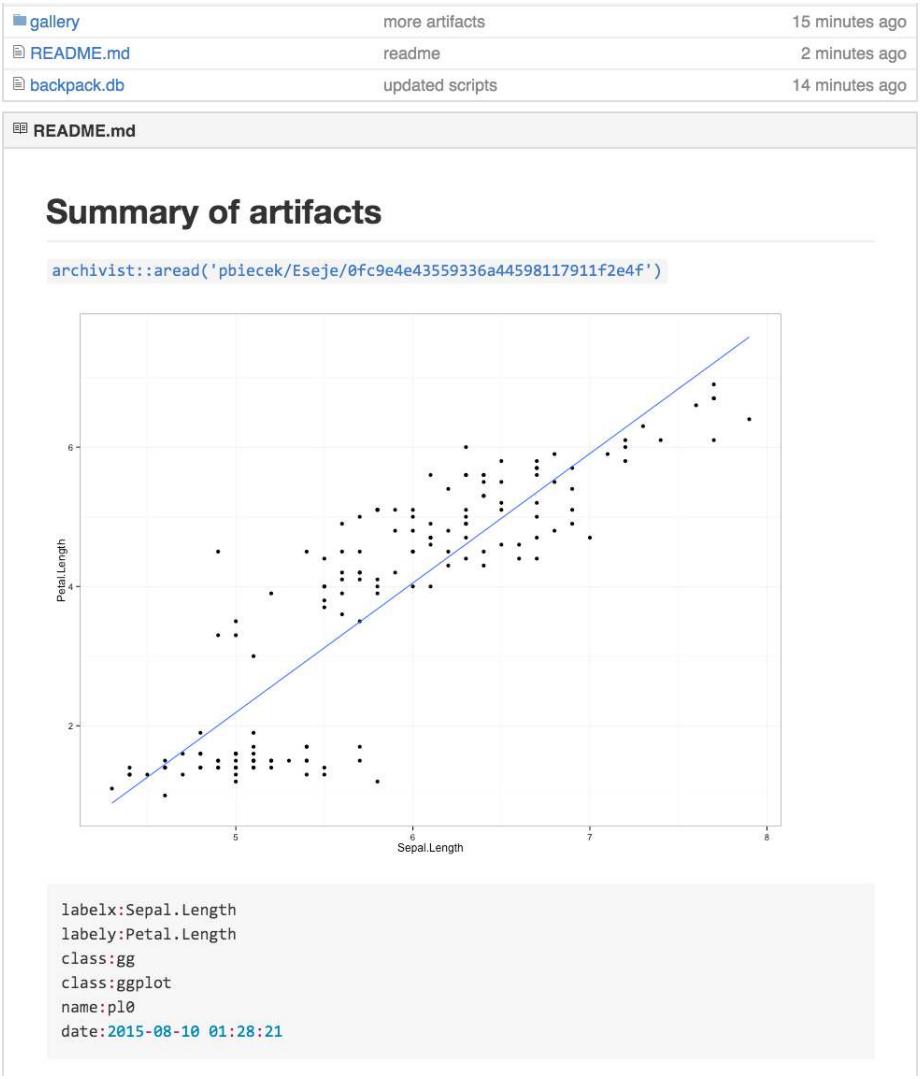


Figure 9: A part of the gallery `http://bit.ly/1Q62Tpz` created with function `createMDGallery`. The gallery presents hooks, miniatures and list of tags for each artifact in the repository.

it is enough to extend the `saveToLocalRepo` function. In order to load objects from other formats it is enough to overload `loadFromLocalRepo` and `loadFromRemoteRepo` functions.

Restoring older versions of packages

In some cases, in order to use an artifact it is not enough to restore it. A good example of this problem are objects of the `gg` class created with **ggplot2** package. The structure of `gg` objects is different in package **ggplot2** in the version 1.0, different in the version 2.0 and different in the version 2.1. It means that even if we have restored an object that was created with package in version 2.0 we will not be able to use the `plot` function for this object if one uses **ggplot2** package in the version 2.1 nor 1.0.

To use the object we need to downgrade **ggplot2** package to the version 2.0. This is possible

with the `restoreLibs` function. For a given hash of an artifact the `restoreLibs` function restores it's `session_info` and reinstalls required packages with versions attached during the artifact's archiving. Packages can be reinstalled in the new directory, not to affect the default R libraries.

For example, the `600bda83cb840947976bd1ce3a11879d` object was created with **ggplot2** version 2.0. The `asession()` function checks versions of packages that were then attached.

```
asession("pbiecek/graphGallery/arepo/600bda83cb840947976bd1ce3a11879d")

## Session info -------------------------------------

##  setting  value
##  version  R version 3.2.2 (2015-08-14)
##  system   x86_64, darwin13.4.0
##  ui       RStudio (0.99.441)
##  language (EN)
##  collate  en_US.UTF-8
##  tz       Europe/Warsaw
##  date     2016-02-09

## Packages -----------------------------------------

##  package      * version  date       source
##  acepack        1.3-3.3  2013-05-03 CRAN (R 3.1.0)
##  archivist    * 1.9.7.2  2016-02-08 CRAN (R 3.2.2)
##  assertthat     0.1      2013-12-06 CRAN (R 3.1.0)
##  bitops         1.0-6    2013-08-17 CRAN (R 3.1.0)
##  car            2.1-1    2015-12-14 CRAN (R 3.2.3)
##  cluster        2.0.3    2015-07-21 CRAN (R 3.2.2)
##  colorspace     1.2-6    2015-03-11 CRAN (R 3.1.3)
##  DBI            0.3.1    2014-09-24 CRAN (R 3.1.1)
##  devtools       1.9.1    2015-09-11 CRAN (R 3.2.0)
##  digest         0.6.9    2016-01-08 CRAN (R 3.2.3)
##  dplyr        * 0.4.3    2015-09-01 CRAN (R 3.2.0)
##  foreign        0.8-65   2015-07-02 CRAN (R 3.2.2)
##  Formula        1.2-1    2015-04-07 CRAN (R 3.1.3)
##  ggplot2        2.0.0    2015-12-16 Github (hadley/ggplot2@11679cd)
##  gridExtra    * 2.0.0    2015-07-14 CRAN (R 3.2.0)
##  gtable         0.1.2    2012-12-05 CRAN (R 3.1.0)
##  Hmisc          3.17-0   2015-09-21 CRAN (R 3.2.0)
##  httr           1.0.0    2015-06-25 CRAN (R 3.2.0)
##  intsvy         1.8      2015-11-30 CRAN (R 3.2.2)
##  labeling       0.3      2014-08-23 CRAN (R 3.1.1)
##  lattice        0.20-33  2015-07-14 CRAN (R 3.2.2)
##  latticeExtra   0.6-26   2013-08-15 CRAN (R 3.1.0)
##  lazyeval       0.1.10   2015-01-02 CRAN (R 3.1.2)
##  lme4           1.1-10   2015-10-06 CRAN (R 3.2.2)
```

```
##  lubridate      1.5.0    2015-12-03 CRAN (R 3.2.3)
##  magrittr       1.5      2014-11-22 CRAN (R 3.1.2)
##  MASS           7.3-43   2015-07-16 CRAN (R 3.2.2)
##  Matrix         1.2-2    2015-07-08 CRAN (R 3.2.2)
##  MatrixModels   0.4-1    2015-08-22 CRAN (R 3.2.0)
##  memisc         0.97     2015-03-08 CRAN (R 3.1.3)
##  memoise        0.2.1    2014-04-22 CRAN (R 3.1.0)
##  mgcv           1.8-7    2015-07-23 CRAN (R 3.2.2)
##  minqa          1.2.4    2014-10-09 CRAN (R 3.1.1)
##  munsell        0.4.2    2013-07-11 CRAN (R 3.1.0)
##  nlme           3.1-121  2015-06-29 CRAN (R 3.2.2)
##  nloptr         1.0.4    2014-08-04 CRAN (R 3.1.1)
##  nnet           7.3-10   2015-06-29 CRAN (R 3.2.2)
##  pbkrtest       0.4-4    2015-12-12 CRAN (R 3.2.3)
##  plyr           1.8.3    2015-06-12 CRAN (R 3.2.0)
##  proto          0.3-10   2012-12-22 CRAN (R 3.1.0)
##  quantreg       5.19     2015-08-31 CRAN (R 3.2.0)
##  R6             2.1.2    2016-01-26 CRAN (R 3.2.3)
##  RColorBrewer   1.1-2    2014-12-07 CRAN (R 3.1.2)
##  Rcpp           0.12.3   2016-01-10 CRAN (R 3.2.3)
##  RCurl          1.95-4.7 2015-06-30 CRAN (R 3.2.0)
##  reshape        0.8.5    2014-04-23 CRAN (R 3.1.0)
##  rpart          4.1-10   2015-06-29 CRAN (R 3.2.2)
##  RSQLite        1.0.0    2014-10-25 CRAN (R 3.1.2)
##  scales         0.3.0    2015-08-25 CRAN (R 3.2.0)
##  SparseM        1.7      2015-08-15 CRAN (R 3.2.0)
##  stringi        1.0-1    2015-10-22 CRAN (R 3.2.0)
##  stringr        1.0.0    2015-04-30 CRAN (R 3.2.0)
##  survival       2.38-3   2015-07-02 CRAN (R 3.2.2)
```

Here the **ggplot2** was in the version 2.0 and was installed from GitHub. The `restoreLibs()` function reinstalls all libraries from proper repositories (here GitHub) to proper versions (here commit `11679cd`).

```
restoreLibs("pbiecek/graphGallery/arepo/600bda83cb840947976bd1ce3a11879d")
```

After that one can load and plot the `ggplot` object since the structure of `gg` object is compatible with installed libraries.

```
aread("pbiecek/graphGallery/arepo/600bda83cb840947976bd1ce3a11879d")
```

4. Conclusions

The goal of a data analysis is not only to answer a research question based on data but also to collect findings that support that answer. These findings usually take the form of a table, plot

or regression/classification model and are usually presented in articles or reports. Such objects are mostly well presented graphically, but they are hard to recreate back in a computer.

In this paper we have presented the R package called **archivist**, which implements the logic of recordable research. The **archivist** stores R objects in repositories. The data scientist may share obtained results with other users, create hooks to models and then embed these hooks in articles, reports or web applications. One may also search within a repository and look for artifacts with given properties or relations with other artifacts. One may also validate the object's identity or derive its pedigree.

Repositories may be shared among team members or between different computers or systems. Statistical models or plots may be stored in a single repository which simplifies the object management.

In this article we have also presented some use-cases for the **archivist** package, such as: hooks for R objects that can be embedded in reports or articles, interactive searching within repository or retrieving object's pedigree.

5. Acknowledgments

Thanks go to Ross Ihaka, Łukasz Bartnik, Cezary Chudzian and two anonymous reviewers for valuable discussions and comments on the idea of recordable research and early versions of this paper. We would like to thank Witold Chodor for his great contributions to the development of this package. The package **archivist** was initiated as an open project in the company *iQor Polska sp. z o.o.*.

References

Auguie B (2015). ***gridExtra**: Miscellaneous Functions for "Grid" Graphics*. R package version 2.0.0, URL http://CRAN.R-project.org/package=gridExtra.

Bache SM, Wickham H (2014). ***magrittr**: A Forward-Pipe Operator for R*. R package version 1.5, URL http://CRAN.R-project.org/package=magrittr.

Becker RA, Chambers JM (1988). "Auditing of Data Analyses." *SIAM Journal on Scientific and Statistical Computing*, **9**(4), 747–760. URL http://epubs.siam.org/doi/ref/10.1137/0909049.

Biecek P, Kosinski M (2017). ***archivist**: An R Package for Managing, Recording and Restoring Data Analysis Results*.

Chambers JM (2016). *Extending R*. Chapman and Hall/CRC. ISBN 978-1498775717.

Chang W, Cheng J, Allaire J, Xie Y, McPherson J (2015). ***shiny**: Web Application Framework for R*. R package version 0.12.1, URL http://CRAN.R-project.org/package=shiny.

Dirk Eddelbuettel with contributions by AL, Tuszynski J, Bengtsson H, Urbanek S, Frasca M, Lewis B, Stokely M, Muehleisen H, Murdoch D, Hester J, Wu W (2014). ***digest**: Create Cryptographic Hash Digests of R Objects*. R package version 0.6.8, URL http://CRAN.R-project.org/package=digest.

Drummond C (2009). "Replicability is not Reproducibility: Nor is it Good Science." *ICML*. URL http://cogprints.org/7691/7/ICMLws09.pdf.

Koenker R, Zeileis A (2009). "On Reproducible Econometric Research." *J. Appl. Econ.*, **24**, 833–847. URL http://biostats.bepress.com/uwbiostat/paper194.

Kosinski M, Biecek P (2016). ***archivist.github**: Tools for Archiving, Managing and Sharing R Objects via GitHub*. R package version 0.2.1, URL https://CRAN.R-project.org/package=archivist.github.

Leisch F (2002). "Dynamic Generation of Statistical Reports Using Literate Data Analysis." *Compstat 2002 - Proceedings in Computational Statistics*, pp. 575–580.

OECD (2015). *StatLink*. The Organisation for Economic Co-operation and Development. URL http://www.oecd.org/statistics/statlink.

Peng R (2009). "Reproducible Research and Biostatistics." *Biostatistics*, **10**(3), 405–408. URL http://biostatistics.oxfordjournals.org/content/10/3/405.short.

Rossini A, Leisch F (2003). "Literate Statistical Practice." *Working Paper 194*. URL http://biostats.bepress.com/uwbiostat/paper194.

Sculley D, Holt G, Golovin D, Davydov E, Phillips T, Ebner D, Chaudhary V, Young M (2014). "Machine Learning: The High Interest Credit Card of Technical Debt." In *SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop)*. URL http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43146.pdf.

Wickham H (2009). ***ggplot2**: Elegant Graphics for Data Analysis*. Springer-Verlag New York. ISBN 978-0-387-98140-6. URL http://had.co.nz/ggplot2/book.

Wickham H, Francois R (2015). ***dplyr**: A Grammar of Data Manipulation*. R package version 0.4.2, URL http://CRAN.R-project.org/package=dplyr.

Xie Y (2013). *Dynamic Documents with R and **knitr***. Chapman and Hall/CRC. ISBN 978-1482203530.

Xie Y (2015). ***knitr**: A General-Purpose Package for Dynamic Report Generation in R*. CRAN. R package version 1.10.5, URL http://cran.r-project.org/package=knitr.

Affiliation:

Przemysław Biecek
Faculty of Mathematics, Informatics, and Mechanics
University of Warsaw
Banacha 2, 02-097 Warsaw, Poland
E-mail: Przemyslaw.Biecek@gmail.com

Marcin Kosiński
Faculty of Mathematics and Information Science

Warsaw University of Technology
Koszykowa 75, 00-662 Warsaw, Poland
E-mail: M.P.Kosinski@gmail.com