archivist : : **record, restore and govern your R objects**

Everything that exists in R is an object. **archivist** is an R package that stores copies of all objects along with their metadata. It helps to manage and recreate objects with final or partial results from data analysis.

Use the archivist to record every result, to share these results with future you or with others, to search through repository of objects created in the past but needed now.

Key functionalities 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 their 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 its creation.

Record artifacts

R objects with partial or final results are called artifacts. They may be either tables, models, plots or any other structures.
Artifacts are stored in repositories. One repositories may be either local or remote.
- local repository is a folder with write/read access,
- remote repositories are usually available through http and have only read access.

Use the **createLocalRepo()** function to create an empty local repository.

```
library(archivist)
createLocalRepo("arepo")
```

Use the **saveToRepo()** function to store selected R objects to a local repository. Each object is stored along with rich metadata (class, date of archivisation) and unique MR5 hash. All related objects, like data, are also stored in the repository.

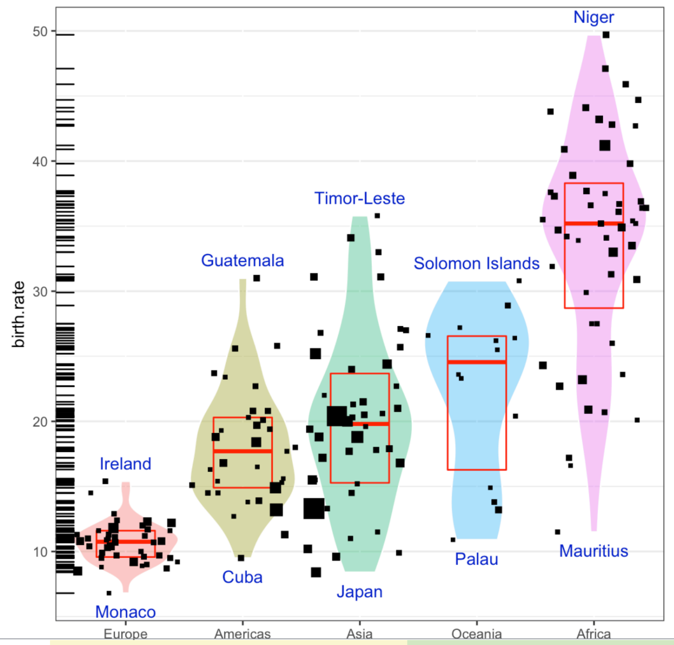
```
model <- lm(len ~ supp+dose,
            data = ToothGrowth)
saveToRepo(model, repoDir = "arepo")
[1] "7c6b38019d2e028b0fb38fc0ed44a175"
attr(,"data")
[1] “1563e36bc621d97b9a736c63f1ccf5cb"
```

Share artifacts

Use **loadFromLocalRepo()** or **loadFromRemote Repo()** or more compact **aread()** functions to retrieve artifacts from repositories.
It's a good idea to attach hooks to key artifacts in reports or articles. The command below restores a ggplot2 object from GitHub `pbiecek/Eseje`.

```
aread('pbiecek/Eseje/arepo/65e430c41')
```

Load: `archivist::aread('pbiecek/Eseje/arepo/65e430c4180e97a704249a56be4a7b88')`



Browse artifacts

Use **searchInLocalRepo()** or **searchInRemoteRepo**() to filter out artifacts that matches selected citeria. Any meta-data collected for the artifact may be used for searching.
The more compact function with smaller number of arguments is **asearch()**.

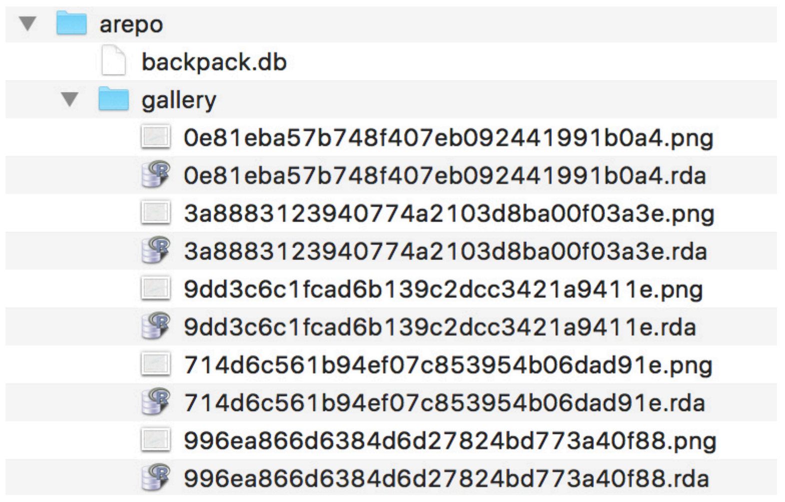
The code below downloads and calculates BIC scores for all artifacts with tag `class:lm` (linear models) from the remote GitHub repository `pbiecek/graphGallery`.

```
models <- asearch("pbiecek/graphGallery",
            patterns = "class:lm")
modelsBIC <- sapply(models, BIC)
sort(modelsBIC)
990861c7c27812ee959f10e5f76fe2c3
                        39.05577
2a6e492cb6982f230e48cf46023e2e4f
                        67.52735
0a82efeb8250a47718cea9d7f64e5ae7
                       189.73593
```

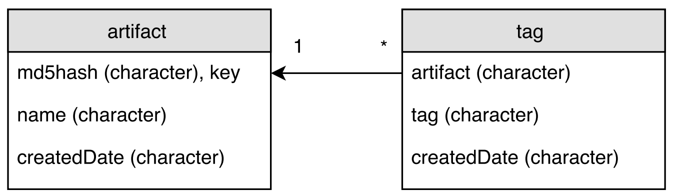
Use **asession()** to retrieve a session info object related to the selected artifact. Use **ahistory()** to retrieve list of commands used to create the artifact.

Architecture

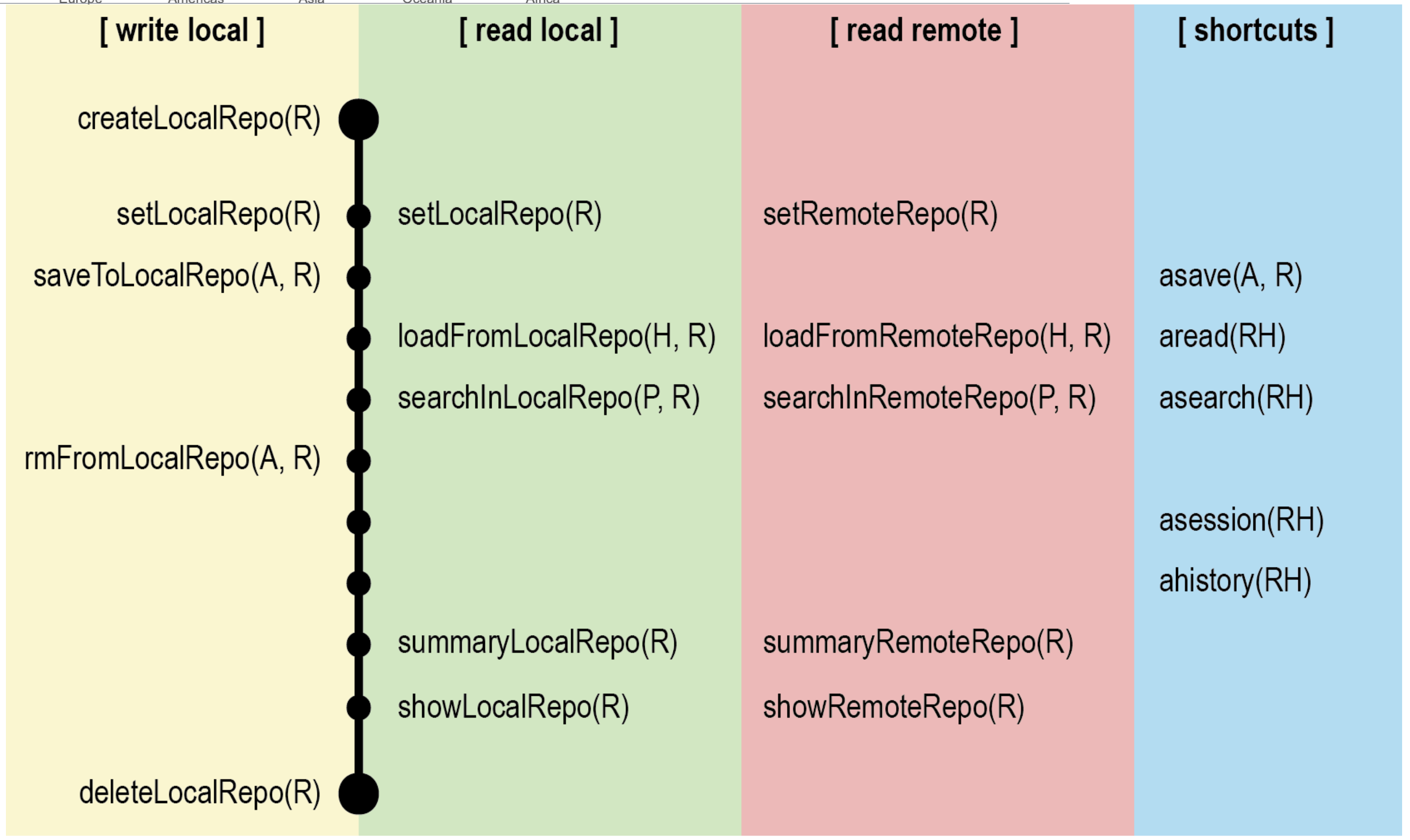
An archivist repository is a folder with following components:
- a **backpack.db** file with SQLite database that stores artifacts metadata, tags and relations between artifacts. One can also use other databases like PostgreSQL.
- a subfolder **gallery** with binary copies of artifacts. Each archivist is stored as a raw rda file with name that corresponds to it's MD5 hash
- small graphical or text miniatures that summaries each artifact. These are stored as txt or png files.



```
▼ arepo
    backpack.db
  ▼ gallery
      0e81eba57b748f407eb092441991b0a4.png
      0e81eba57b748f407eb092441991b0a4.rda
      3a8883123940774a2103d8ba00f03a3e.png
      3a8883123940774a2103d8ba00f03a3e.rda
      9dd3c6c1fcad6b139c2dcc3421a9411e.png
      9dd3c6c1fcad6b139c2dcc3421a9411e.rda
      714d6c561b94ef07c853954b06dad91e.png
      714d6c561b94ef07c853954b06dad91e.rda
      996ea866d6384d6d27824bd773a40f88.png
      996ea866d6384d6d27824bd773a40f88.rda
```



Meta-data about artifacts is stored in the database in two tables: *artifact* and *tag*. Various tags are attached to an artifact. These tags store information about class, variable names, session info, relation among artifacts and other predefined or user defined tags. They are useful to trace and recover artifacts.



[write local]	[read local]	[read remote]	[shortcuts]
createLocalRepo(R)			
setLocalRepo(R)	setLocalRepo(R)	setRemoteRepo(R)	
saveToLocalRepo(A, R)			asave(A, R)
	loadFromLocalRepo(H, R)	loadFromRemoteRepo(H, R)	aread(RH)
	searchInLocalRepo(P, R)	searchInRemoteRepo(P, R)	asearch(RH)
rmFromLocalRepo(A, R)			
			asession(RH)
			ahistory(RH)
	summaryLocalRepo(R)	summaryRemoteRepo(R)	
	showLocalRepo(R)	showRemoteRepo(R)	
deleteLocalRepo(R)			

A - artifact, any R object, like data.frame, ggplot, lm
H - md5hash, cryptographical hash of arbitrary R object
P - pattern, used to find artifacts with suitable tags

R - repository, a local repository is a folder, a remote repo is a based on git or hg. Repository contains rda dumps, miniatures and data base with object's tags.

breakDown plots : : **visual explanations for lm/glm models**

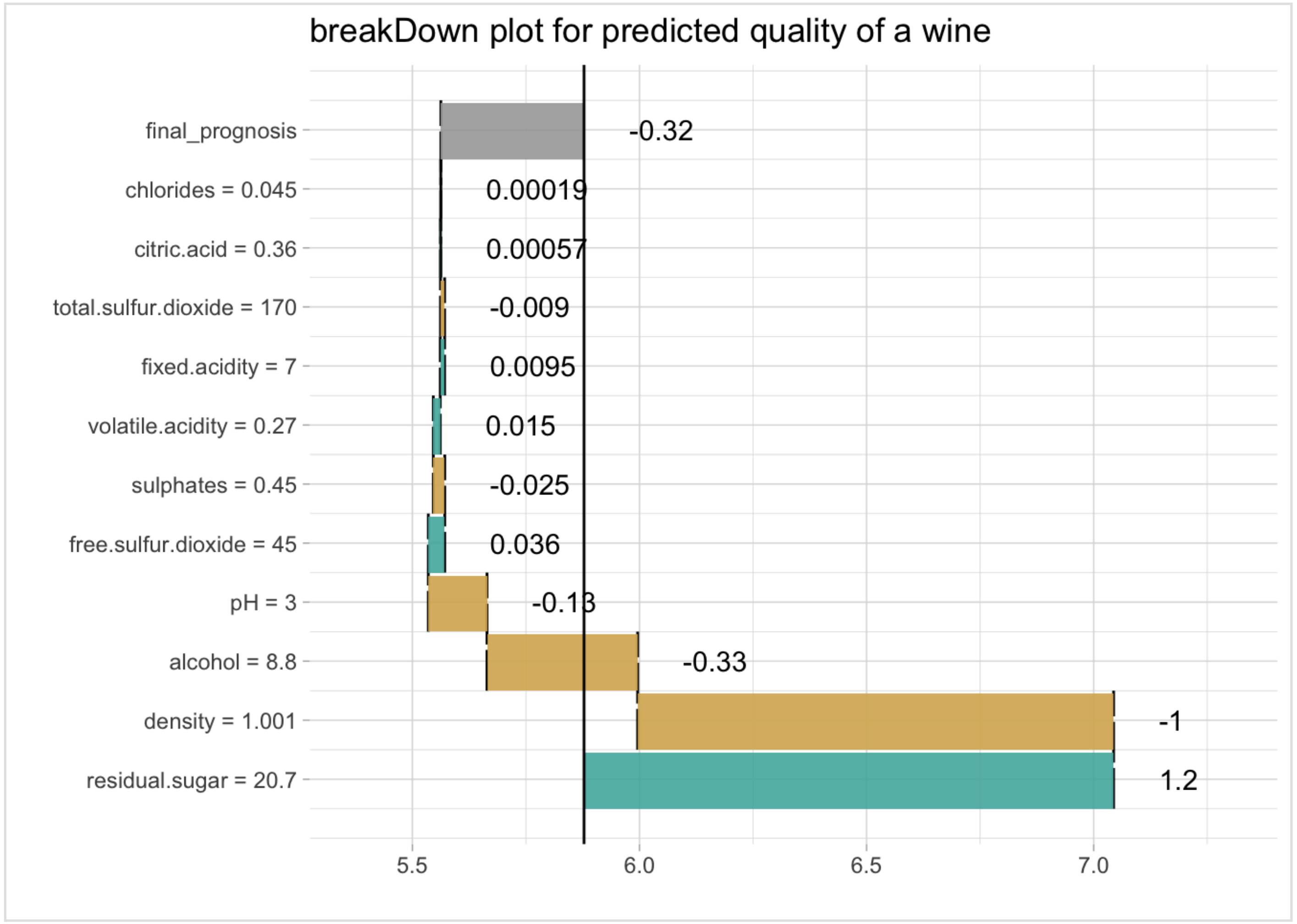
Linear model

Linear models are widely used in predictive modeling.
They have simple structure, which makes them easy to deploy or implement.
But models with many variables are hard to understand.

The **breakDown** plot explains the relation between variables and model prediction for a new observation.

Explanations are generated in three steps:
1) create model with **lm**() function
2) break down model predictions with the **broken()** function
3) plot the graphical summary with the generic **plot**() function.

```
library(breakDown)
library(ggplot2)
model <- lm(quality ~ ., data = wineQuality)
br <- broken(model, wineQuality[1,],
          baseline = "Intercept")
br
#>                              contribution
#> residual.sugar = 20.7             1.20000
#> density = 1.001                  -1.00000
#> alcohol = 8.8                    -0.33000
#> pH = 3                           -0.13000
#> free.sulfur.dioxide = 45          0.03600
#> sulphates = 0.45                 -0.02500
#> volatile.acidity = 0.27           0.01500
#> fixed.acidity = 7                 0.00950
#> total.sulfur.dioxide = 170       -0.00900
#> citric.acid = 0.36                0.00057
#> chlorides = 0.045                 0.00019
#> final_prognosis                  -0.32000
#> baseline:  5.877909
plot(br)
```



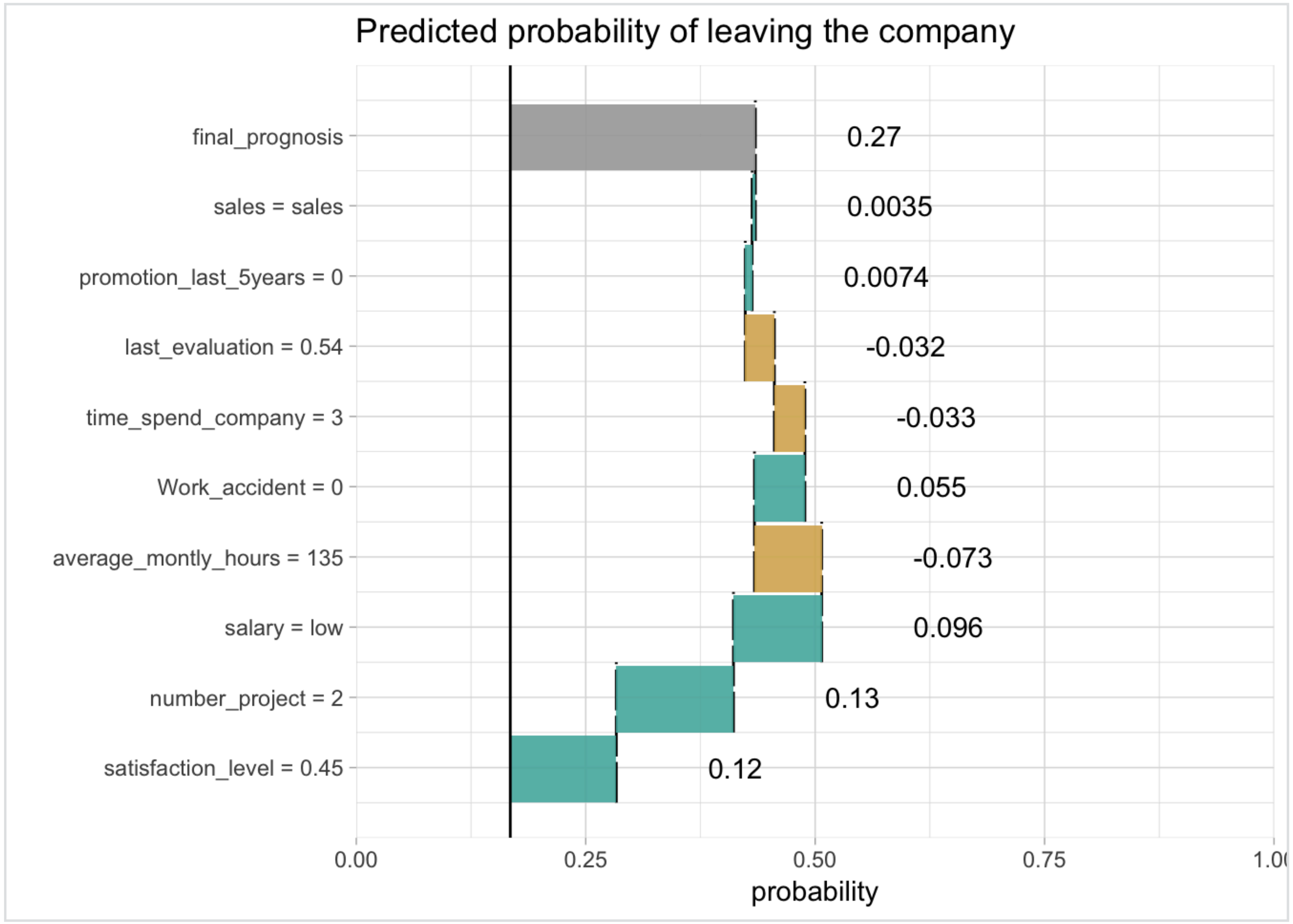
Logistic regression

breakDown plots may be also used to explain predictions from the logistic regression model.

On the OX axis one may present linear predictions (default) or use probit/logit transformation to present contributions of variables from the model. Use the trans= argument to define the transformation.

The baseline is presented by the vertical black line in the plot. One may set the baseline to 0 or to population average (use the *baseline = "intercept"* argument).

```
library(breakDown)
library(ggplot2)
model <- glm(left~., data = HR_data,
                 family = "binomial")
explain_1 <- broken(model, HR_data[11,],
                 baseline = "intercept")
explain_1
#>                              contribution
#> satisfaction_level = 0.45           0.670
#> number_project = 2                  0.570
#> salary = low                        0.390
#> average_montly_hours = 135         -0.290
#> Work_accident = 0                   0.220
#> time_spend_company = 3             -0.130
#> last_evaluation = 0.54             -0.130
#> promotion_last_5years = 0           0.030
#> sales = sales                       0.014
#> final_prognosis                     1.300
#> baseline:  -1.601457
plot(explain_1, trans = function(x)
               exp(x)/(1+exp(x)))
```


Predicted probability of leaving the company
final_prognosis
sales = sales
promotion_last_5years = 0
last_evaluation = 0.54
time_spend_company = 3
Work_accident = 0
average_montly_hours = 135
salary = low
number_project = 2
satisfaction_level = 0.45
probability