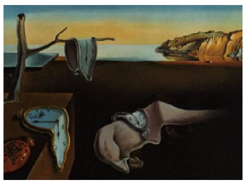


Journal of Statistical Software

October 2012, Volume 51, Issue 2. http://www.jstatsoft.org/

speedR: An R Package for Interactive Data Import, Filtering and Ready-to-Use Code Generation

Ilhami Visne
Austrian Institute of Technology

Ahmet Yildiz
Austrian Institute of Technology

Erkan Dilaveroglu
Austrian Institute of Technology

Klemens Vierlinger
Austrian Institute of Technology

Christa Nöhammer
Austrian Institute of Technology

Friedrich Leisch
University of Natural Resources and Life Sciences

Albert Kriegner
Austrian Institute of Technology

Abstract

Emerging technologies in the experimental sciences have opened the way for large-scale experiments. Such experiments generate ever growing amounts of data from which researchers need to extract relevant pieces for subsequent analysis. R offers a great environment for statistical analysis. However, due to the diversity of possible data sources and formats, data preprocessing and import can be time consuming especially with data that require user interaction such as editing, filtering or formatting. Writing a code for these tasks can be time-consuming, error prone and rather complex. We present **speedR**, an R-package for interactive data import, filtering and code generation in order to address these needs. Using **speedR**, researchers can import new data, make basic corrections, examine current R session objects, open them in the **speedR** environment for filtering (subsetting), put the filtered data back into R, and even create new R functions with applied import and filtering constraints to speed up their productivity.

Keywords: filter, import, interactive, GUI, R, Java, script.

1. Introduction

Technological advances in the experimental sciences, especially in life sciences, have enabled researchers from various scientific domains to do more comprehensive research using more complete datasets. A remarkable example of this is the huge amount of data from genome sequencing acquired since the human genome sequencing was first accomplished in 2000 (Lander *et al.* 2001). Another tangible example of such a big data source is the microarray technology. From one single microarray experiment a large number of data tables are generated. R (R Development Core Team 2012) offers a great environment for statistical analysis. However, data preprocessing and import has always been a tedious issue especially with files that require any user interaction in formatting or editing. Especially data importing and filtering (subsetting) can be confusing to non-expert R users. Moreover, even minor errors in the input data can affect result accuracy. Thus the importance of reliable correction facilities can easily be realized. Tools like **Excel**, **TableButler** (Schwager, Wirkner, Abdollahi, and Huber 2009) are specialized for the preprocessing of such data tables prior to statistical analysis. Nevertheless, R users will always prefer using R, since otherwise one would have to open a separate program and changes are not recorded which reduces reproducibility.

Importers are available for frequently used file types such as the ones for microarray data from the Bioconductor (Gentleman *et al.* 2004) project, however most domains are lacking well defined file formats and hence such specific importers.

To address these needs, we have developed **speedR**, an interactive R package for data import and advanced filtering (subsetting). **speedR** project is hosted at `https://R-Forge.R-project.org/projects/speedr` and the package is available from the Comprehensive R Archive Network at `http://CRAN.R-project.org/package=speedR`.

2. Using speedR

2.1. Installation

speedR needs two main components for running: Java Runtime Environment ≥ 1.6 and R version $\geq 2.10.x$. After running R, the following code should be executed to install **speedR** and all required libraries:

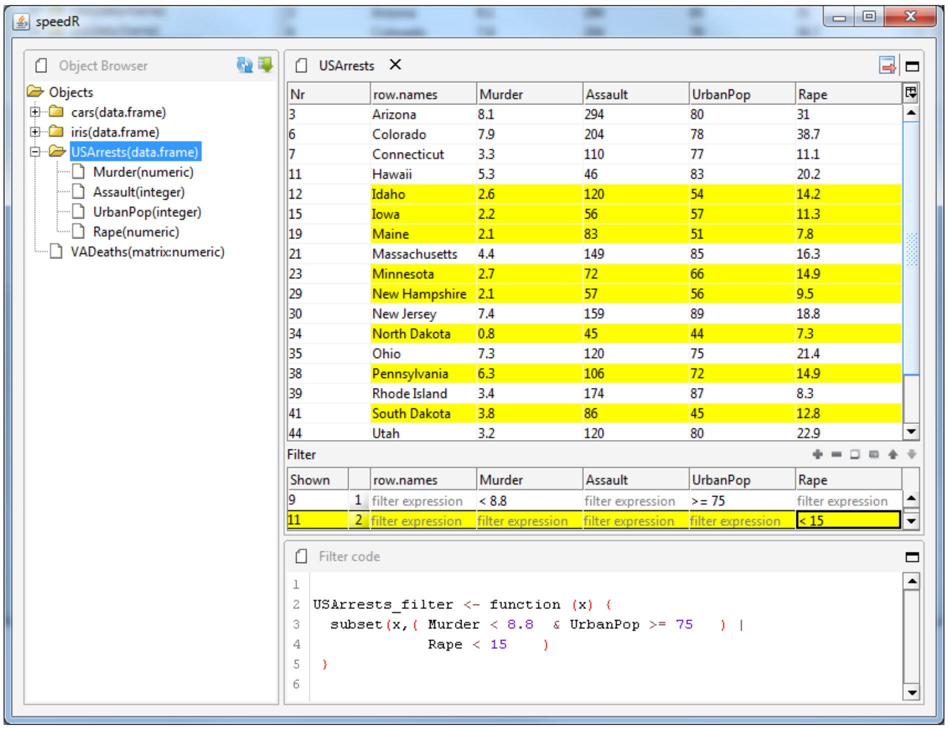
```
R> install.packages("speedR")
```

After successful installation, **speedR** can easily be started by the following code in R:

```
R> library("speedR")
R> speedR()
```

Mac OS X limitation

Certain Mac OS X systems (like Snow Leopard) have stopped to support abstract window toolkit (AWT Naughton 1996, p. 283) in a single-threaded applications. Since **speedR** depends on AWT, it is impossible to start **speedR** from R.app or from the terminal. The solution is to use R and **speedR** from within **JGR** (Helbig, Urbanek, and Fellows 2012).



Nr	row.names	Murder	Assault	UrbanPop	Rape
3	Arizona	8.1	294	80	31
6	Colorado	7.9	204	78	38.7
7	Connecticut	3.3	110	77	11.1
11	Hawaii	5.3	46	83	20.2
12	Idaho	2.6	120	54	14.2
15	Iowa	2.2	56	57	11.3
19	Maine	2.1	83	51	7.8
21	Massachusetts	4.4	149	85	16.3
23	Minnesota	2.7	72	66	14.9
29	New Hampshire	2.1	57	56	9.5
30	New Jersey	7.4	159	89	18.8
34	North Dakota	0.8	45	44	7.3
35	Ohio	7.3	120	75	21.4
38	Pennsylvania	6.3	106	72	14.9
39	Rhode Island	3.4	174	87	8.3
41	South Dakota	3.8	86	45	12.8
44	Utah	3.2	120	80	22.9

Shown		row.names	Murder	Assault	UrbanPop	Rape
9	1	filter expression	< 8.8	filter expression	>= 75	filter expression
11	2	filter expression	filter expression	filter expression	filter expression	< 15

```

USArrests_filter <- function (x) {
  subset(x,( Murder < 8.8  & UrbanPop >= 75   ) |
             Rape < 15    )
 }

```

Figure 1: **speedR** main window.

2.2. Importing files

speedR enables data import from a wide range of sources such as **Excel** (from version 97), **OpenOffice.org Calc** (only ODS, open document spreadsheet), CSV (comma-separated values) and character delimited (fixed-width format is not supported in this version) text files. Unknown data formats will be treated like text files. The `Importer` can be opened by clicking the `import` button located on the right top corner of the `Object Browser` window (see Figure 2).

Data import consists of a two step wizard "select data" (Figure 3) and "edit table" (Figure 4). In the first step (Figure 3), the user can specify the data source. This can be performed by either using the `file chooser` GUI (graphical user interface) element or by pasting or writing a URL in the corresponding `text field`. Alternatively, the user may paste raw data from the system clipboard into the corresponding `text area`.

Once the file source has been defined, the user can click the `Next` button in order to pass to the second step. The importer now tries to automatically pick up all relevant file parameters such as *separator*, *quote*, *column* and *row names* by analyzing the data. The resulting table is now shown in the second step. Depending on the detected file type, different options will be enabled to modify the settings such as *separator*, *quote*, *column* and *row names*. Thus, the user can manually define a colum as the *row names* holder and whether a row should be used as the *column names row*. Furthermore, the user can define what should be imported by modifying the data range in the `Data range` options field. Changes to these settings will immediately be applied to the table and refreshed in the table view.

Once the user reviewed and configured the settings, the user can set the data types (`numeric`,

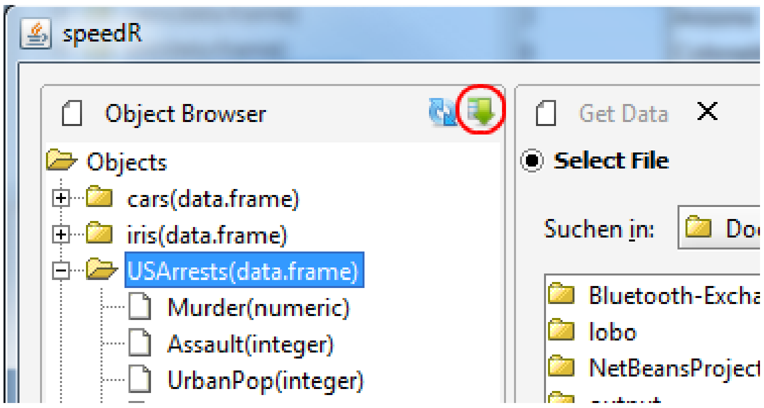


Figure 2: Button to start the import wizard.

`character`, `factor`, `Date` and `POSIXct`) for each column seperately from a `dropdown list` standing above every column. If the user sets a column as `numeric`, **speedR** checks all values in that column and if there exists a non-numeric value in a row, the user will be informed. Then user can correct that row and try againg. In case of `Date` and `POSIXct`, **speedR** tries to convert using the most frequently used `Date` and `POSIXct` formats. If the values in that column have a different format, the user will be prompted for the right format and then **speedR** will try again using this format. Additionally, the user can use the hierarchical filter to subset the data to be imported. Before completing the data import, the user can assign a variable

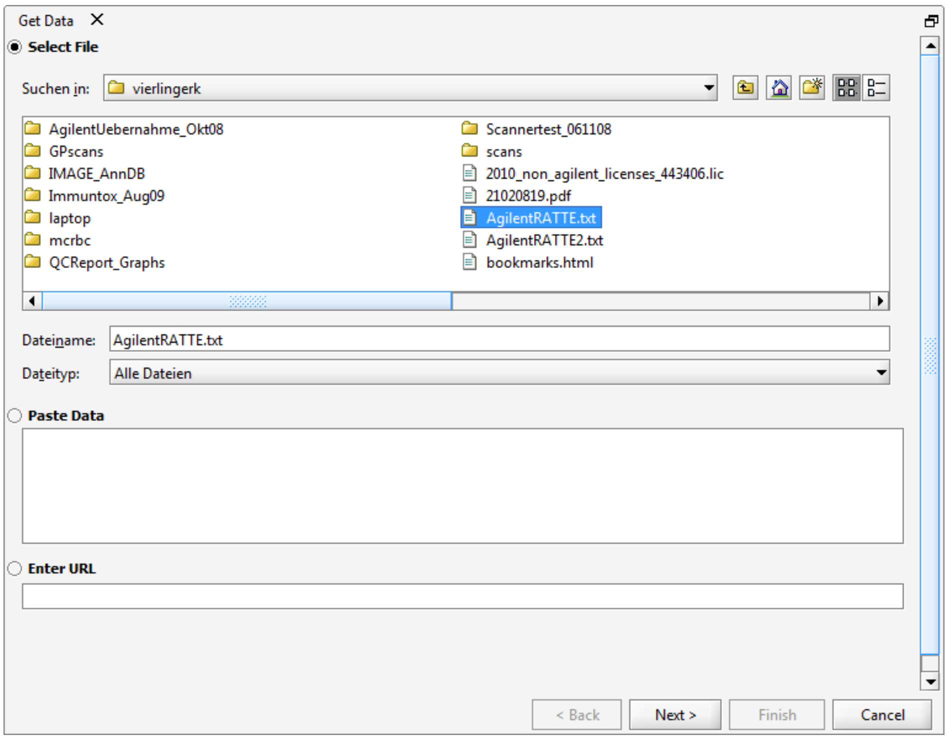
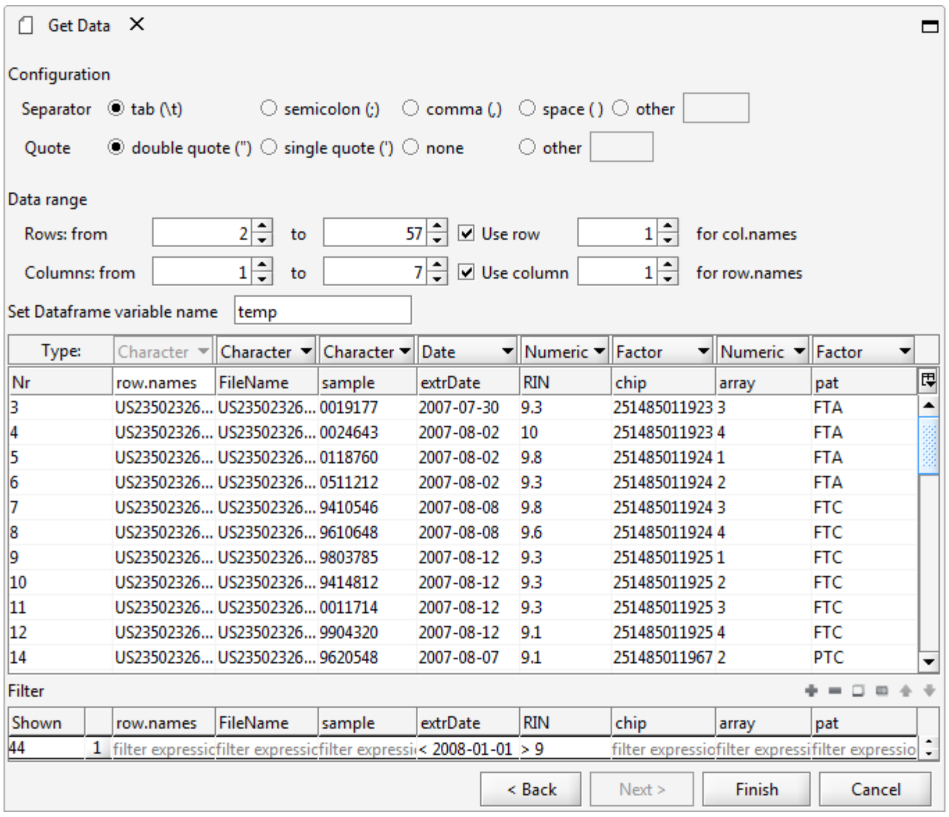


Figure 3: Select data wizard step with embedded file browser and text fields to paste file URLs or direct data.



Configuration

Separator [✓ tab (\t)] semicolon (;) comma (,) space ( ) other

Quote [✓ double quote (")] single quote (') none other

Data range

Rows: from 2 to 57 [✓ Use row] 1 for col.names

Columns: from 1 to 7 [✓ Use column] 1 for row.names

Set Dataframe variable name temp

Type:	Character	Character	Character	Date	Numeric	Factor	Numeric	Factor
Nr	row.names	FileName	sample	extrDate	RIN	chip	array	pat
3	US23502326...	US23502326...	0019177	2007-07-30	9.3	251485011923	3	FTA
4	US23502326...	US23502326...	0024643	2007-08-02	10	251485011923	4	FTA
5	US23502326...	US23502326...	0118760	2007-08-02	9.8	251485011924	1	FTA
6	US23502326...	US23502326...	0511212	2007-08-02	9.3	251485011924	2	FTA
7	US23502326...	US23502326...	9410546	2007-08-08	9.8	251485011924	3	FTC
8	US23502326...	US23502326...	9610648	2007-08-08	9.6	251485011924	4	FTC
9	US23502326...	US23502326...	9803785	2007-08-12	9.3	251485011925	1	FTC
10	US23502326...	US23502326...	9414812	2007-08-12	9.3	251485011925	2	FTC
11	US23502326...	US23502326...	0011714	2007-08-12	9.3	251485011925	3	FTC
12	US23502326...	US23502326...	9904320	2007-08-12	9.1	251485011925	4	FTC
14	US23502326...	US23502326...	9620548	2007-08-07	9.1	251485011967	2	PTC

Filter

Shown		row.names	FileName	sample	extrDate	RIN	chip	array	pat
44	1	filter expressic	filter expressic	filter expressi	< 2008-01-01	> 9	filter expressio	filter expressi	filter expressio

Figure 4: Shows the options for a text file.

```
temp <- speedR.importany(file = "C:/Users/visnei/Documents/targets.txt",
 separator = "\t",quote = "\"", rowstart = 2,colend = 7,hasRowNames = TRUE,
 rowNamesColumnIndex = 1,hasColumnNames = TRUE,
 colClasses=c("character","character","Date","numeric","factor","numeric","factor"))

temp_filter <- function (x) {
  subset(x,( extrDate < as.Date("2008-01-01") & RIN > 9) )
}

temp <- temp_filter(temp)
```

Figure 5: The generated code for the settings in Figure 4

name (default `temp`) under which the data will directly be imported to the R workspace.

At the end of these steps, **speedR** generates the ready-to-use importing code with the settings made by the user and a filter function according to the filter expressions. Figure 5 shows the generated code for the settings in Figure 4.

Comparision with other tools and packages

There are other GUI-based tools and packages that support data import to R, such as **RKward** (Rödiger, Friedrichsmeier, Kapat, and Michalke 2012), **JGR** (Helbig *et al.* 2012), **Rcmdr**

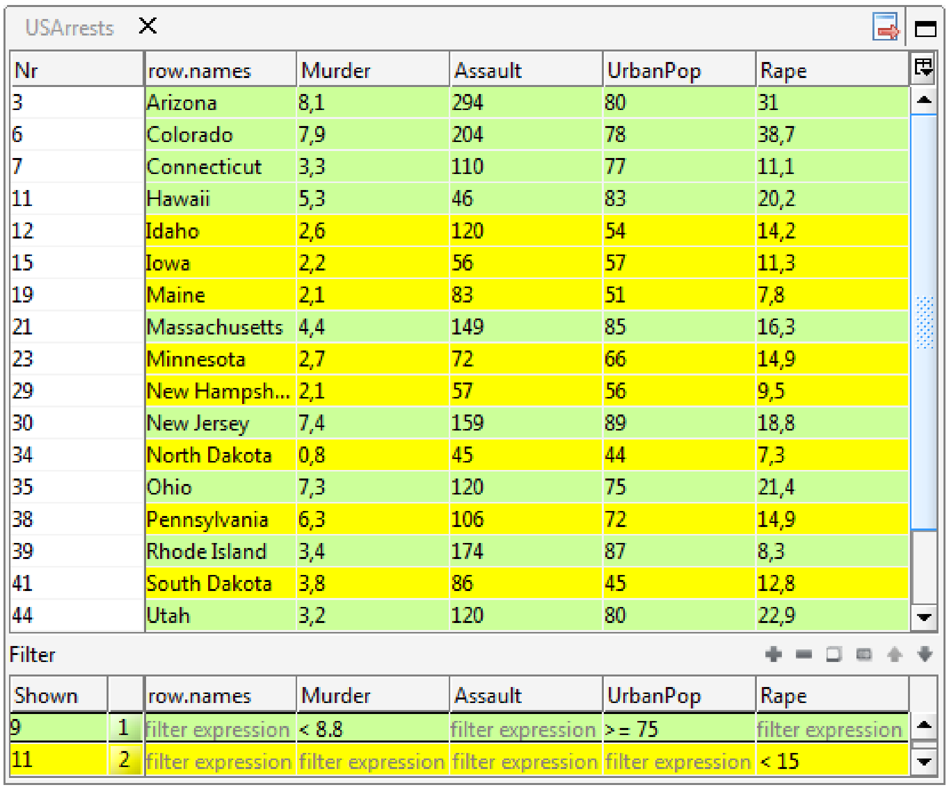
(Fox 2005). All of these tools support similar file formats such as SPSS, Stata, text/CSV providing separate GUIs for each file type. **RKward** is a GUI frontend for R with a click and point user interfaces to standard statistical functions. **RKward** has an import wizard that supports SPSS, Stata and text/CSV files but not **Excel** or **OpenOffice.org Calc**. Import of text/CSV files require previous knowledge of file properties such as *separator*, *quote*, etc. Like **speedR**, **RKward** generates R code. Visual feedback is provided only at the end of the import procedure.

JGR is a universal and unified graphical user interface for R that supports import of various file types such as SPSS, Stata, SAS export, text/CSV, etc. Again **Excel** (XLS, **Excel** spreadsheet, and XLSX, **Excel** spreadsheet XML) and **OpenOffice.org Calc** are not supported. For the import of text/CSV files **JGR** detects *separator* and *quote* automatically and displays a portion of the result in a preview. Unlike **RKward** and **JGR**, **Rcmdr** is an R package that provides GUIs to basic-statistic tests and not a frontend to R. Import of text/CSV files require previous knowledge of file properties (*separator*, *quote*, etc.) and no visual feedback is provided. None of these tools support data editing, filtering, sub setting in their import wizards.

2.3. Filtering (Subsetting)

Data filtering (subsetting) is important issues at any stage of an analysis. During the analysis flow the researcher always needs to check intermediate results, extract relevant subsets of information for more detailed analysis or edit and correct minor errors. The **speedR** filtering engine addresses these needs with advanced filtering. The **speedR** filter engine (Figure 6) accepts multiple filter levels which are hierarchically applied to the data, starting with the first filter level. A filter level is the combination of all filter criteria of the same line in the filter editor. Each filter level is assigned to an editable default color; data matching that filter level criteria are labeled with that filter level color. The filter criteria are applied immediately. Hit counts for each filter level are displayed left to the corresponding filter level in the `Shown` column (Figure 6). The filtered table can be added to the R workspace or exported to the file system by clicking the `export` button in the upper right corner (Figure 6), the user will be prompted for a variable or file name, respectively. **speedR** is equipped with a simple expression syntax which makes it easy to write powerful filter conditions. Depending on the data type, proper filter expressions are offered in a popup menu when clicking a cell in the `Filter Editor`.

- `<` *value*: Less than the *value* (`number`, `Date` or `POSIXct`).
- `>` *value*: Greater than the *value*.
- `<=` *value*: Less than or equals to the *value*.
- `>=` *value*: Greater than or equals the *value*.
- *value* `to` *value*: Defines an interval. Example: `5 to 10`.
- `is.na`: Selects a row if the cell value is `NA`.
- `contains` *"string"*: Selects cells having strings which contains given character set.



Nr	row.names	Murder	Assault	UrbanPop	Rape
3	Arizona	8,1	294	80	31
6	Colorado	7,9	204	78	38,7
7	Connecticut	3,3	110	77	11,1
11	Hawaii	5,3	46	83	20,2
12	Idaho	2,6	120	54	14,2
15	Iowa	2,2	56	57	11,3
19	Maine	2,1	83	51	7,8
21	Massachusetts	4,4	149	85	16,3
23	Minnesota	2,7	72	66	14,9
29	New Hampsh...	2,1	57	56	9,5
30	New Jersey	7,4	159	89	18,8
34	North Dakota	0,8	45	44	7,3
35	Ohio	7,3	120	75	21,4
38	Pennsylvania	6,3	106	72	14,9
39	Rhode Island	3,4	174	87	8,3
41	South Dakota	3,8	86	45	12,8
44	Utah	3,2	120	80	22,9

Filter

Shown		row.names	Murder	Assault	UrbanPop	Rape
9	1	filter expression	< 8.8	filter expression	>= 75	filter expression
11	2	filter expression	filter expression	filter expression	filter expression	< 15

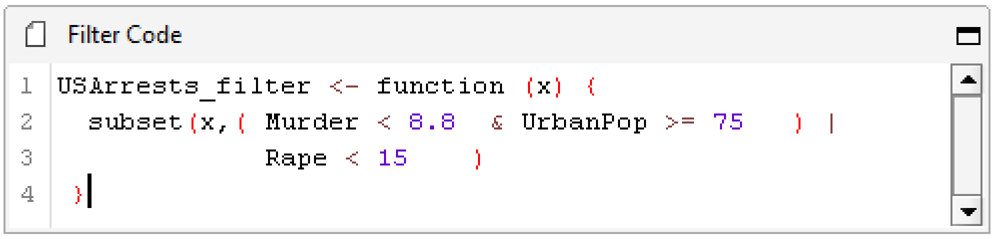
Figure 6: Table panel: Advanced filter engine in **speedR**.

- `equals` *"string"*: The cells are only than selected if cell value is *equals* to the given string. Note that this is case-sensitive.
- `not` *expression*: `not` predicate can be used together with any type of expressions to inverse the sub-expression result.
- *expression* `and` *expression*: Expressions may be bound with `and` keyword.
- *expression* `or` *expression*: Expressions may be bound with *or* keyword.
- *( )*: Parenthesis can be used to groups the expressions, e.g., `is.na or (<5 and >2)`.

While writing the filter expressions, **speedR** translates these filter expressions into an equivalent R function (Figure 7), which is displayed in the `Table Panel` (Figure 6). This function uses only function from base package and doesn't depend on **speedR**.

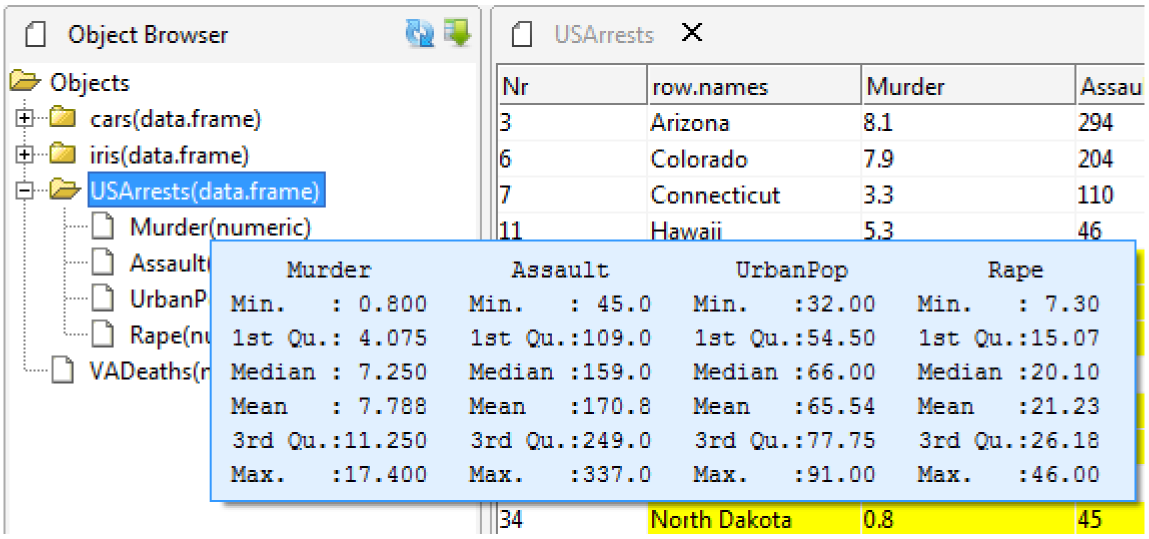
2.4. Interaction with R

In addition to data import and filtering, **speedR** allows easy access to data in the R workspace by providing an interface to open data from R and to return modified data to R. R objects can easily be opened via the `Object Browser` (see Figure 8) by simply double clicking them and the summary can be displayed by triggering the context menu action (system dependent, on Windows right mouse click). The content of the object browser can be refreshed by clicking the `Refresh` button.



```
USArrests_filter <- function (x) {
 subset(x,( Murder < 8.8  & UrbanPop >= 75   ) |
              Rape < 15     )
}
```

Figure 7: R filter function for given **speedR** easy filter constraints.



Nr	row.names	Murder	Assau
3	Arizona	8.1	294
6	Colorado	7.9	204
7	Connecticut	3.3	110
11	Hawaii	5.3	46
34	North Dakota	0.8	45

```
     Murder           Assault         UrbanPop          Rape
Min.    : 0.800   Min.    : 45.0   Min.    :32.00   Min.    : 7.30
1st Qu.: 4.075   1st Qu.:109.0   1st Qu.:54.50   1st Qu.:15.07
Median : 7.250   Median :159.0   Median :66.00   Median :20.10
Mean    : 7.788   Mean    :170.8   Mean    :65.54   Mean    :21.23
3rd Qu.:11.250   3rd Qu.:249.0   3rd Qu.:77.75   3rd Qu.:26.18
Max.    :17.400   Max.    :337.0   Max.    :91.00   Max.    :46.00
```

Figure 8: **speedR** object browser.

3. Implementation

In this section some internals of the package implemation will be discussed.

3.1. Overall architecture

The **speedR** R package is implemented in Java and R (see Figure 9). Java is used for GUI implementation, importing of new data, displaying the contents of objects from R workspace, filtering and R code generation. As shown in Figure 9, **rJava** (Urbanek 2011) bridges R and

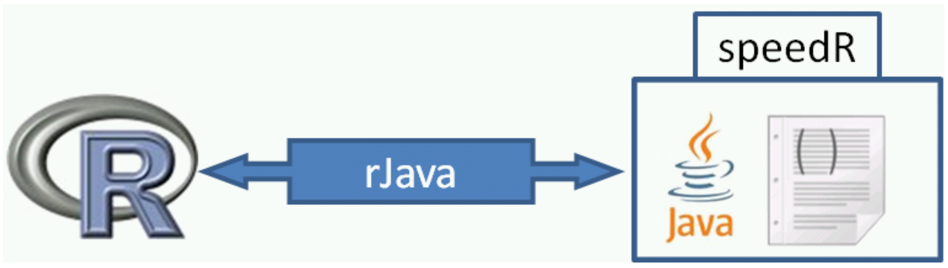


Figure 9: **speedR** architecture.

Java for seamless interaction. From the web page: ***rJava** provides a low-level bridge between R and Java via Java native interface (Liang 1999). It allows creating objects, calling methods and accessing fields of Java objects from R* (Urbanek 2011). **speedR**'s R functions are called via **rJava** i.e., to load the content and to refresh the `Object Browser` (see Figure 8).

3.2. Basic filter expression language

The **speedR**'s filter expression is a basic domain-specific language (DSL, Fowler 2010). *DSLs are generally very high-level languages tailored to specific tasks* (Parr 2007, p. 21). Any valid filter expression (Section 2.3) is converted into an equivalent R code. In the background, **speedR** uses ANTLR (Parr 2007) to translate filter expressions. *ANTLR is a parser generator to implement language interpreters, compilers, and other translators* (Parr 2007, p. 21). ANTLR takes a grammar file (formal language description) as an input and generates a program (parser) that determines whether the filter expressions conform to the syntax specified by the grammar. The Parser still does not emit any R code. To emit R code **speedR** uses the **StringTemplate** (Parr 2010, Chapter 12) engine. **StringTemplate** is a Java template engine for generating source code. Using a template engine allows us to separate the code generation from the parser, so parser can be changed without touching template and vice versa. A template (`RCodeGen.stg` in the project source directory) is applied after a successful parsing and this emits the R code. Below some possible filter expressions are displayed:

- `< 5 or = 8`.
- `contains "set" and not equal "setosa"`.
- `(>= 3 and < 9) or ( > 20 and not = 32 )`.

R code generation

Figure 10 shows the flow of a final R filter code being generated from a filter expression. The filter expression is passed to the parser, which is generated with ANTLR, and upon success it creates an intermediate form called abstract syntax tree (AST, Parr 2007, p.24) that is a condensed version of the filter expression i.e., the parenthesis are removed. In the last step, AST is applied to R code template that emits the R Code.

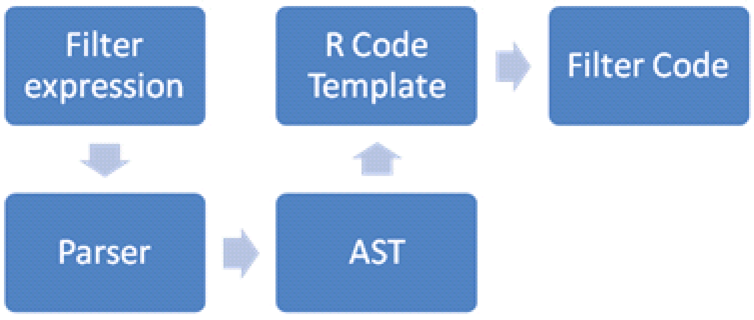
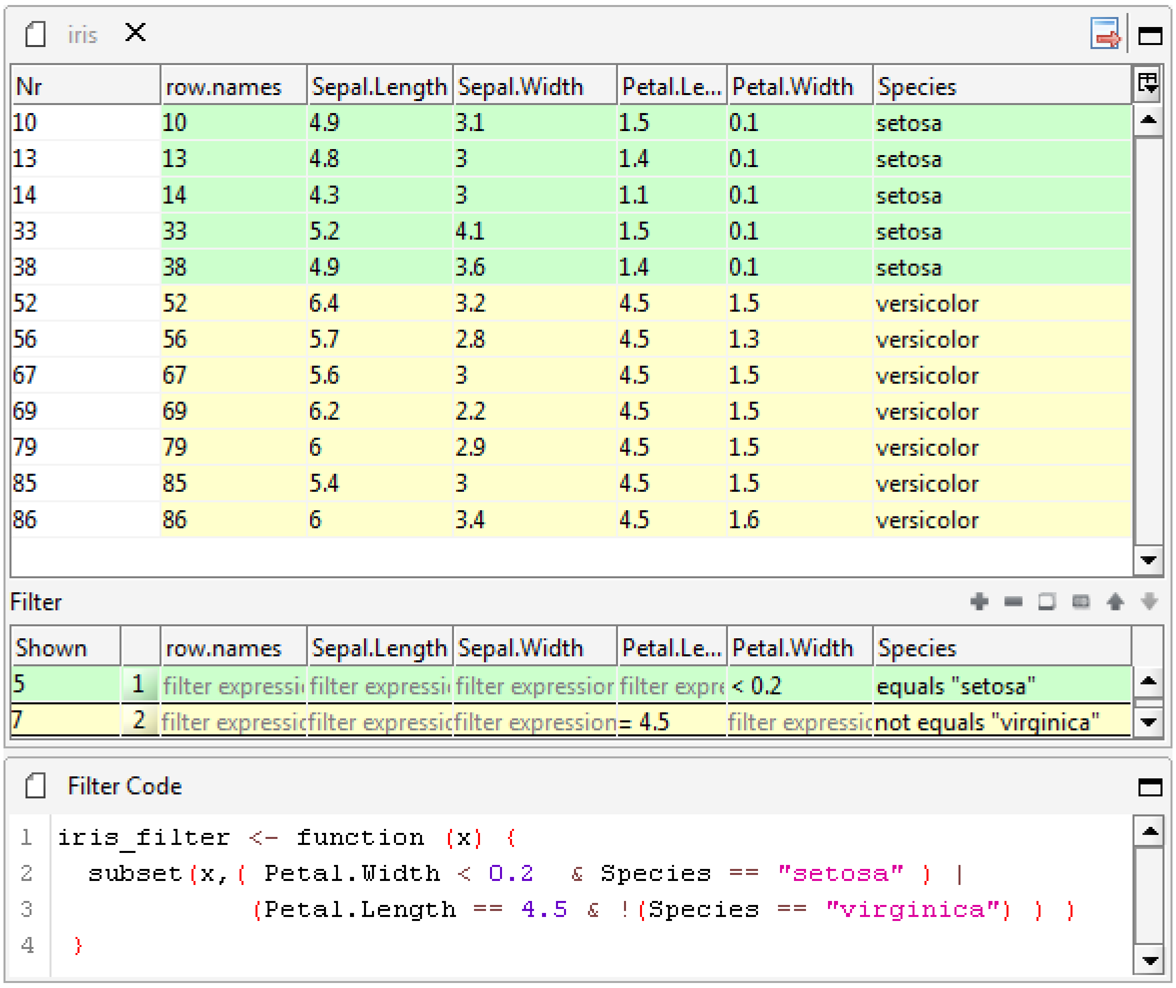


Figure 10: R code generation flow.



Nr	row.names	Sepal.Length	Sepal.Width	Petal.Le...	Petal.Width	Species
10	10	4.9	3.1	1.5	0.1	setosa
13	13	4.8	3	1.4	0.1	setosa
14	14	4.3	3	1.1	0.1	setosa
33	33	5.2	4.1	1.5	0.1	setosa
38	38	4.9	3.6	1.4	0.1	setosa
52	52	6.4	3.2	4.5	1.5	versicolor
56	56	5.7	2.8	4.5	1.3	versicolor
67	67	5.6	3	4.5	1.5	versicolor
69	69	6.2	2.2	4.5	1.5	versicolor
79	79	6	2.9	4.5	1.5	versicolor
85	85	5.4	3	4.5	1.5	versicolor
86	86	6	3.4	4.5	1.6	versicolor

Shown		row.names	Sepal.Length	Sepal.Width	Petal.Le...	Petal.Width	Species
5	1	filter expressi	filter expressi	filter expressior	filter expre	< 0.2	equals "setosa"
7	2	filter expressic	filter expressic	filter expression	= 4.5	filter expressic	not equals "virginica"

```
iris_filter <- function (x) {
  subset(x,( Petal.Width < 0.2  & Species == "setosa" ) |
             (Petal.Length == 4.5 & !(Species == "virginica") ) )
 }
```

Figure 11: The filtered iris data set and the generated R function.

Let's assume as an example the famous iris flower data set (Anderson 1935). The iris data set consists of measurements of four features *sepal width* , *sepal length*, *petal width*, *petal length* for 50 flowers from 3 iris species *setosa*, *versicolor* and *virgicolor*. In our imaginary case, we want to get a result, in which all *setosa* have *petal width* less than `0.2` or all species except *virginica* with an exact *petal length* value equals to `4.5`. Figure 11 shows the filtered iris data set. Two filter levels have been set according to our imaginary case. The subpanel below shows the generated R function. To this function, any matrix-like data can be passed that have the same column names as in the function body.

4. Summary

R is the leading open source statistics software with a vastly growing community. Data importing and filtering (subsetting) are the first steps in every analysis. Due to the diversity of the data sources and non-standardized formats, passing these steps can be a time-consuming issue espescially for the non-expert R users. **speedR** is an interactive GUI-based tool, for easy data import, data filtering and R code generation. **speedR** was implemented in Java and R and is provided as an R package. Data import in R requires the selection of the right method and arguments for each file types; often formatting or naming have to be performed. **speedR** recognizes a large number of file formats like **Excel** (XLS and XLSX), **OpenOffice.org Calc** (only ODS), CSV and text files automatically. Unknown formats will be treated like text files. For text files, **speedR** tries to guess all parameters such as *separator*, *quote* and *column names row*. The user will always see the resulting table, thus, in case they were misdetected, the settings on the GUI can be changed. Changes will be evaluated and the table immediately refreshed. If the user is interested only in a subset of the table, he can either manually define the range that needs to be imported, or use the easy to operate built-in filter to restrict the import data range. All settings will be translated in a corresponding ready-to-use R function. **speedR** includes an `Object Browser` where all data objects can be opened in the `Table Viewer` for quick preview or advanced data selection and subsetting. The built-in filter accepts multiple filter levels that are hierarchically applied to the data. A filter level is defined by the sum of all filter arguments of the same line in the filter editor; filter expressions are applied dynamically. Each filter level is assigned to a default color, so that the user gets a visual feedback on the data structure. Easy to use filter expression allows complex filtering with a easily readable syntax. Filter expressions are offered from a popup menu which is opened when clicking the corresponding cell in the filter editor. Finally, all filter settings are reflected in a corresponding ready-to-use R function.

References

Anderson E (1935). "The Irises of the Gaspe Peninsula." *Bulletin of the American Iris Society*, **59**, 2–5.

Fowler M (2010). *Domain-Specific Languages*. Addison-Wesley.

Fox J (2005). "The R Commander: A Basic-Statistics Graphical User Interface to R." *Journal of Statistical Software*, **14**(9), 1–42. URL `http://www.jstatsoft.org/v14/i09/`.

Gentleman RC, Carey VJ, Bates DM, Bolstad B, Dettling M, Dudoit S, Ellis B, Gautier L, Ge Y, Gentry J, Hornik K, Hothorn T, Huber W, Iacus S, Irizarry R, Leisch F, Li C, Maechler M, Rossini AJ, Sawitzki G, Smith C, Smyth G, Tierney L, Yang JYH, Zhang J (2004). "**Bioconductor**: Open Software Development for Computational Biology and Bioinformatics." *Genome Biology*, **5**, R80. URL `http://genomebiology.com/2004/5/10/R80`.

Helbig M, Urbanek S, Fellows I (2012). ***JGR**: Java Gui for R*. R package version 1.7-11, URL `http://CRAN.R-project.org/package=JGR`.

Lander ES, Linton LM, Birren B, Nusbaum C, Zody MC, Baldwin J, Devon K, Dewar K, Doyle M, FitzHugh W, et al (2001). "Initial Sequencing and Analysis of the Human Genome." *Nature*, **409**(6822), 860–921.

Liang S (1999). *The Java Native Interface: Programmer's Guide and Specification*. Addison-Wesley. URL `http://java.sun.com/docs/books/jni`.

Naughton P (1996). *The Java Handbook: The Authoritative Guide To The Java Revolution*. Osborne/McGraw-Hill, Berkeley, CA, USA.

Parr T (2007). *The Definitive ANTLR Reference: Building Domain-Specific Languages.* Pragmatic Bookshelf, Raleigh. URL http://www.pragmaticprogrammer.com/titles/tpantlr/.

Parr T (2010). *Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages.* Pragmatic Bookshelf. URL http://pragprog.com/book/tpdsl/language-implementation-patterns.

R Development Core Team (2012). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Rödiger S, Friedrichsmeier T, Kapat P, Michalke M (2012). "**RKWard**: A Comprehensive Graphical User Interface and Integrated Development Environment for Statistical Analysis with R." *Journal of Statistical Software*, **49**(9), 1–34. URL http://www.jstatsoft.org/v49/i09/.

Schwager C, Wirkner U, Abdollahi A, Huber P (2009). "**TableButler** – A Windows Based Tool for Processing Large Data Tables Generated with High-Throughput Methods." *BMC Bioinformatics*, **10**(1), 235.

Urbanek S (2011). ***rJava**: Low-Level R to Java Interface.* R package version 0.9-3, URL http://CRAN.R-project.org/package=rJava.

Affiliation:

Albert Kriegner
AIT Austrian Institute of Technology GmbH
Muthgasse 11/2
1190 Wien, Austria
E-mail: albert.kriegner@ait.ac.at
URL: http://www.ait.ac.at/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 51, Issue 2 *Submitted:* 2010-12-30
October 2012 *Accepted:* 2012-09-22