

# Integrating R Machine Learning Algorithms in Stata using rcall

## A Tutorial

E. F. Haghish

University of Oslo

# Overview

- Part 1: Introduction
  - Introduction to `rcall` package
  - Why do we need language interfacing?
  - Language interfacing for statistical analysis
  - How does `rcall` work?
  - Interactive vs non-interactive workflows
  - Installing `rcall`
  - `rcall` syntax and arguments
- Part 2: Decision trees
  - a very brief introduction to decision trees
  - Pros and cons
- Part 3: Using `rcall` interactively
  - Interactive workflow
  - Using decision trees algorithms with `rcall`
  - covering tips about using `rcall` interactively
- Part 4: using `rcall` for writing Stata packages
  - Examples of embedding R in Stata packages

# References

## Data

- Statlog (German Credit Data) Data Set
- [https://archive.ics.uci.edu/ml/datasets/statlog+\(german+credit+data\)](https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data))
- The dataset is included in the presentation repository
  - `./examples/credit.csv`
- data is about identifying risky bank loans, includes 1000 Obs.
- The variable of interest is default i.e. loan goes to default:
  - 1 = NO (70%) and 2 = YES (30%)

## Machine learning (ML) examples

- Solely focus on decision trees, using C5 algorithm
- Machine learning with R
- <https://www.packtpub.com/product/machine-learning-with-r-third-edition/9781788295864>

# Notes

## R is rcall's abbreviation

- `rcall` command can be abbreviated as `R`
- In this presentation `rcall` and `R` commands are used interchangeably

## Why teaching `rcall` with machine learning?

- ML models often require follow ups and refining
- The workflow is often *interactive*
  - the dataset is split into **training** and **test** datasets
  - the model(s) is developed on a training dataset
  - the model(s) are tested on the test dataset
  - often many models are compared, fine-tuned, and optimized
- They are idea for demonstrating the interactive workflow with `rcall`

# Part 1: Introduction to rcall package

# Why language interfacing matters?

- Definition
  - facilitate communication between programs written in different languages
  - facilitate sharing objects between programs
- Popularity
  - Saving resources
  - Avoiding *reinventing the wheel*
- Reproducibility
  - There is no statistical software that does everything
  - We might need a different program for a part of the analysis
  - Interfacing helps to keep the analysis in one place

# Language interfacing for statistical analysis

- Interfacing is **different from automation**
  - We can write a script to execute multiple instructions:
  - e.g. Calling MPlus, R, Stata from shell script
  - Stata supports executing shell script and automation
  - Interfacing typically allows **object communication**
- Interfacing is common in computer sciences
  - e.g. running a Java library within Python
  - e.g. running C++ within an R program to execute a loop
  - e.g. running MATA within Stata
- Embedding a different language inside a program requires a strict structure
- Statistical analysis is interactive
  - Interfacing for regular data analysis should be seamless
  - We need to be able to integrating different statistical programs for daily use

# How does rcall work?

- rcall is a general interfacing program to embed R within Stata
- It provides a seamless procedure for using R in Stata
  - Can be used for interactive data analysis within Stata
  - Can also be used for embedding R in Stata programs and packages
- It facilitate object communications between Stata and R
  - dataset
  - matrices
  - scalars
  - variables
  - macros
- When calling R, the results will be available to Stata as objects



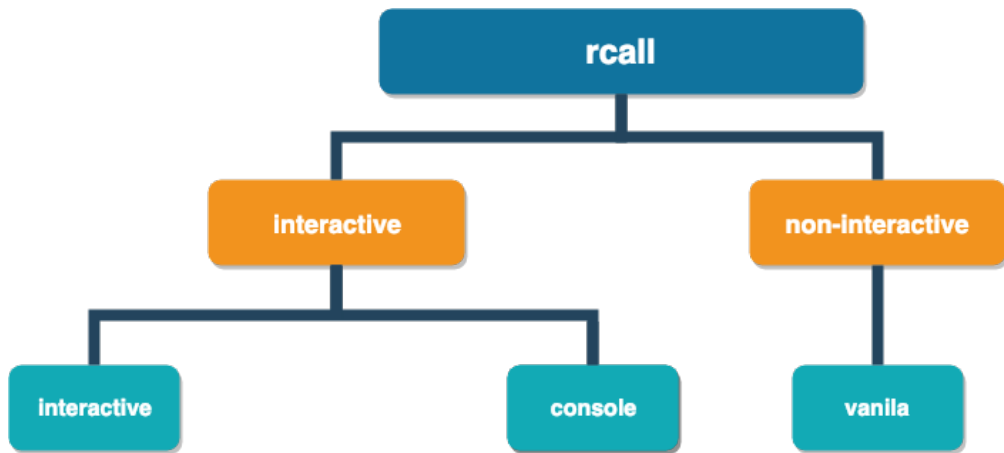


Figure 1: `rcall` modes

# Interactive vs non-interactive workflows

- In interactive sessions R will preserve all objects existing in the memory
  - consequent commands that are executed in the same environment
  - This is similar to working interactively in Rstudio
  - Desired when R is used interactively for data analysis
- Non-interactive sessions do not have any memory
  - Every command is executed in a new environment
  - After the execution, the environment is removed
  - Restrict the session to a specific computation
  - Desired when R is embedded in Stata programs

# Installing rcall

- rcall is hosted on GitHub.
- The only recommended installation method is using the `github` package
- First, install the `github` package:

```
net install github, from("https://haghish.github.io/github/")
```

- Then install the latest rcall stable release

```
github install haghish/rcall, stable
```

- You can alternatively install the latest development version

```
github install haghish/rcall
```

- rcall required R package will be installed automatically
- The dependencies can also be installed manually within R
- See the `dependency.do` file in the GitHub repo
- To update rcall, type: `github update rcall`

# Syntax

- ❶ `rcall [subcommand]`
- ❷ `rcall script "filename.R" [, args() vanilla ]`
- ❸ `rcall [mode] [:] [R-command]`
  - console
  - interactive
  - non-interactive (vanilla)

# Data communication

- `rcall` offers several functions for passing dataset, variables, matrices, and scalars to R
- datasets can also be loaded from R environment to Stata
- `rcall` returns matrices and scalars automatically from R to Stata

Function	Description
<code>st.scalar(name)</code>	passes a scalar to R
<code>st.matrix(name)</code>	passes a matrix to R
<code>st.var(varname)</code>	passes a numeric or string variable to R
<code>st.data(filename)</code>	passes Stata data to R. without filename, the currently loaded data is used.
<code>st.load(dataframe)</code>	loads data from R dataframe to Stata

Figure 2: `rcall` subcommands

# Console mode

- Is useful for casual or exploratory work
- type `rcall` to enter the simulated R environment
- type `end` to exit the simulated environment

```
Results Q
. rcall clear
(R memory cleared)

. rcall → enter the interactive console mode
_____ R (type end to exit) _____

. a = 99
. end → exit the console mode
_____

. return list → see the results returned by R to Stata

scalars:
      r(rc) = 0
      r(a) = 99
.
_____
_____
Command
```

# Console mode

```
. sysuse auto, clear
(1978 Automobile Data)

. rcall
_____ R (type end to exit) _____

. df <- st.data()
. head(df)
      make price mpg rep78 headroom trunk weight length turn displacement
1  AMC Concord  4099  22     3     2.5   11  2930   186   40         121
2    AMC Pacer  4749  17     3     3.0   11  3350   173   40         258
3    AMC Spirit 3799  22    NA     3.0   12  2640   168   35         121
4 Buick Century 4816  20     3     4.5   16  3250   196   40         196
5 Buick Electra 7827  15     4     4.0   20  4080   222   43         350
6 Buick LeSabre 5788  18     3     4.0   21  3670   218   43         231

gear_ratio foreign
1      3.58 Domestic
2      2.53 Domestic
3      3.08 Domestic
4      2.93 Domestic
5      2.41 Domestic
6      2.73 Domestic
. make <- as.matrix(df$price)
. end
_____

. return list

scalars:

      r(rc) =  0

matrices:

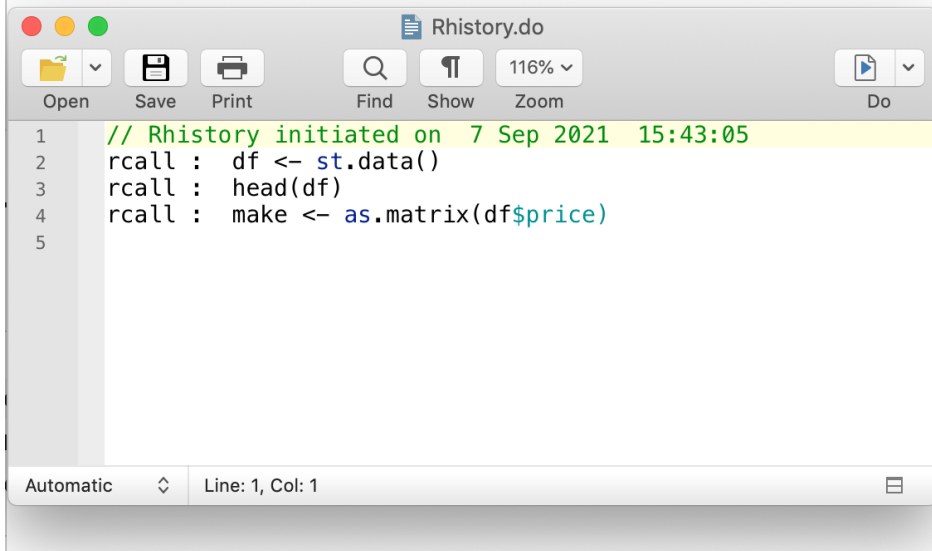
      r(make) :  74 x 1

.
```

Figure 4: example of working in console mode

# Logging R code

. rcall history



```
1 // Rhistry initiated on 7 Sep 2021 15:43:05
2 rcall : df <- st.data()
3 rcall : head(df)
4 rcall : make <- as.matrix(df$price)
5
```

Figure 5: rcall history



# Subcommands

- Commands for setting up and monitoring R within Stata

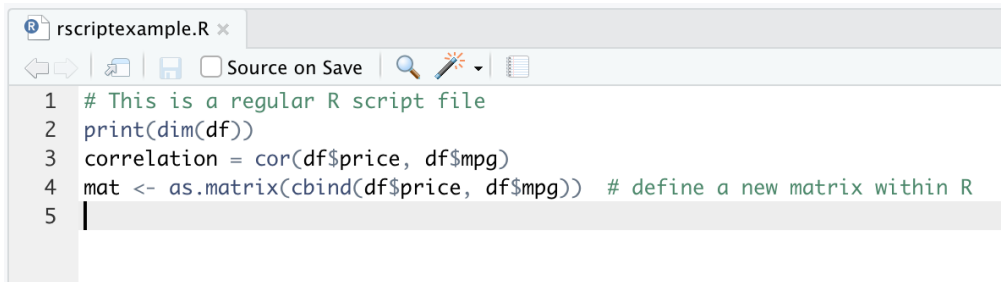
Subcommand	Description
setpath	permanently defines the path to executable R on the machine
clear	erases the R memory and history in the interactive mode
script	executes an R script file and returns the results to Stata
warnings	shows the warnings returned from R
describe	returns the R version and paths to R, RProfile, and Rhistory
history	opens <b>Rhistory.do</b> in do-file editor which stores the history of the interactive R session
site	opens <b>rprofile.site</b> in do-file editor which is used for customizing R when is called from Stata

Added in rcall 3.0

# Script subcommand

- rcall can also source an R script file:
  - `rcall script "filename.R" [, args() vanilla ]`
- the `args()` can be used to give instructions or define objects in R, prior to sourcing
- e.g. pass dataset, matrices, variables, scalars, and macros to R
- the `script` subcommand is the simplest way for running R within Stata programs
  - In this case, the `vanila` option is recommended

# Example 1



```
rscriptexample.R x
Source on Save
1 # This is a regular R script file
2 print(dim(df))
3 correlation = cor(df$price, df$mpg)
4 mat <- as.matrix(cbind(df$price, df$mpg)) # define a new matrix within R
5 |
```

Figure 6: rscriptexample.R file

```
clear
sysuse auto
rcall script: ./examples/rscriptexample.R , args(df<-st.data()) vanilla
```

# Example 1

```
. rcall script: ./examples/rscriptexample.R , args(df<-st.data()) vanilla
[1] 74 12

. return list

scalars:
      r(rc) = 0
r(correlation) = -.4685967

matrices:
      r(mat) : 74 x 2

.
```

Figure 7: Example of running an R script within Stata

## Part 2: Decision trees

# Decision trees

- very popular ML classifiers, due to their simplicity
- Can be applied on most data types
- a tree-like structure to model the relationships among potential outcomes
- intuitive and human-readable, offering high transparency in decision making
- the model structure:
  - begins at a wide trunk (**root node**)
  - splits into narrower branches (**decision nodes**)
  - every split is a decision, creating **branches**
  - with the final decision, the model reaches **leaf nodes** or **terminal nodes**
- Here I focus on the **C5.0** decision tree algorithm

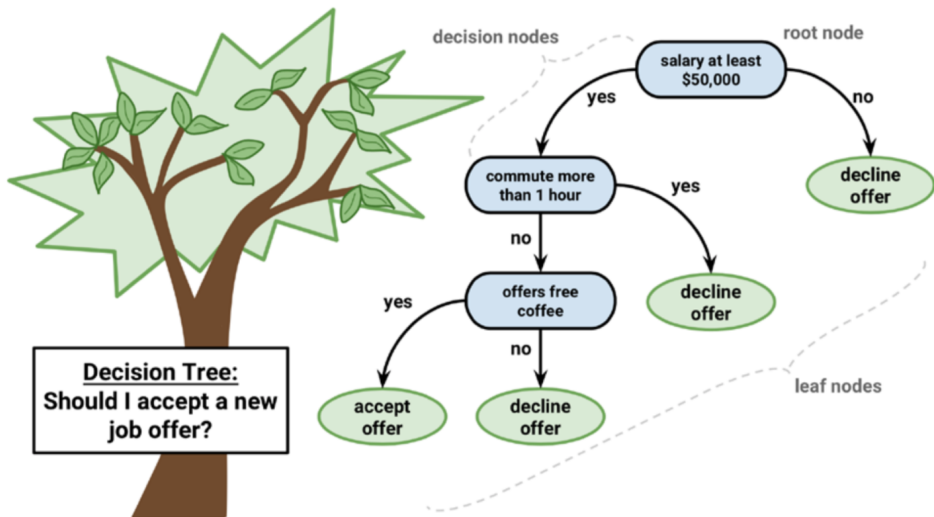


Figure 8: Decision tree example

# Pros and cons

## Pros

- general purpose classifier
- well on most problems
- offer automatic learning process
- can work with different types
- can handle missing data
- can work on different sample sizes
- highly interpretable

## Cons

- become over-complex with complicated data structures
- biased toward splits on variables with higher levels
- prone to overfitting and underfitting



## Part 3: Using `rcall` interactively

# Using `rcall` from Stata do-file editor

- R code can be executed within Stata do-file editor
  - `rcall [mode] [:] [R-command]`
- in `rcall` 3.0, the `mode` argument can be
  - **vanilla** for non-interactive session
  - for interactive session, no `mode` is required
  - the `sync` mode is removed in `rcall` version 3
- When using `rcall` interactively, **ALWAYS** start a new R session
  - type: `rcall clear` to start a fresh session

# Example 1

## Loading credit.csv dataset from R to Stata

- I can load the dataset with rcall as follows

```
// load the data in a new R session and pass it to Stata
rcall clear
rcall: df<-read.csv("./examples/credit.csv", stringsAsFactor = TRUE)
rcall: st.load(df)
```

```
. table default
```

```
-----
default |      Freq.
-----+-----
      1 |         700
      2 |         300
-----
```

- I could load the CSV data with Stata, without converting strings to factors:
  - `import delimited "./examples/credit.csv"`

# Working with the C5 R package

```
1 // load the dataset for the analysis
2 use credit, clear
3
4
5 R clear // cleaning the R environment
6 R: credit <- st.data() // pass the data to R
7 R: class(credit\default) //check that it is a factor
8
9 // prepare the train and test datasets
10 R: set.seed(123); ///
11   train_sample <- sample(1000, 900); ///
12   train <- credit[train_sample, ]; ///
13   test <- credit[-train_sample, ]
14
15
16 // load the library and create the model
17 R: library(C50); ///
18   model <- C5.0(train[-17], train\default)
19
20 R: model|
```

Figure 9: Preparing the data for C5

# Working with the C5 R package

```
. R: model
```

**Call:**

```
C5.0.default(x = train[-17], y = train$default)
```

**Classification Tree**

**Number of samples: 900**

**Number of predictors: 20**

**Tree size: 42**

**Non-standard options: attempt to group attributes**

**.**

Figure 10: Preparing the data for C5

# Summarizing the training model

```
. R: summary(model)
```

- shows the structure of the decision trees
- shows how much different variables are contributed to the model
- confusion matrix of the training dataset

**Evaluation on training data (900 cases):**

```
      Decision Tree
-----
Size      Errors

  42  136(15.1%)  <<

(a)  (b)  <-classified as
----  ----
 612   23   (a): class no
 113  152   (b): class yes
```

# Predicting the test dataset

```
47  
48 R: prediction <- predict(model, test)  
49 R: prediction  
50
```

Figure 12: Predicting the test dataset

```
. R: prediction  
[1] no no no yes no yes no yes no no no no no yes no no no no  
[19] no no no no no no no no no no no no yes no no no yes no  
[37] yes no no yes no yes no no no no yes no no no no no no  
[55] no no no no yes no yes no no no no no no yes yes no no yes  
[73] no no no no no yes no yes no no no no yes yes yes no no no  
[91] no no no yes no no yes no no yes  
Levels: no yes
```

Figure 13: Predicted variable

# Producing the confusion matrix

```
26  
27 R: prediction <- predict(model, test)  
28 R: prediction  
29 R: table(test$default, prediction, ///  
30         dnn = c("actual default", "predicted default"))  
31
```

Figure 14: Predicting the test dataset

	predicted default	
actual default	no	yes
no	55	10
yes	22	13



# Using rcall from do-file editor is similar to console mode

```
Results Q

. R
----- R (type end to exit) -----

. prediction
[1] no no no yes no yes no yes no no no no no yes no no no no
[19] no no no no no no no no no no no no yes no no no yes no
[37] yes no no yes no yes no no no no yes no no no no no no
[55] no no no no yes no yes no no no no no no yes yes no no yes
[73] no no no no no yes no yes no no no no yes yes yes no no no
[91] no no no yes no no yes no no yes

Levels: no yes

. ls()
[1] "adj.names" "credit" "model" "prediction" "rc"
[6] "stata.output" "test" "train" "train_sample"
. end

. return list

scalars:

      r(rc) = 0

macros:

      r(train_sample) : "415 463 179 526 195 938 818 118 299 229 244 14 374 665 602 603 768 709 91 953 348 649 355 840.."
```

Figure 15: Accessing R session from rcall console

# Return a particular matrix or table to Stata

```
. R: confusion <- table(test$default, prediction,    ///  
>   dnn = c("actual default", "predicted default"))  
  
. R: class(confusion)  
[1] "table"  
  
. R: confusion <- as.matrix(unclass(confusion), dnn = NULL)  
  
. // the matrix will be returned to Stata  
. R: confusion  
      predicted default  
actual default no yes  
      no  55  10  
      yes  22  13  
  
. return list  
  
scalars:  
      r(rc) =  0  
  
macros:  
      r(train_sample) : "415 463 179 526 195 938 818 118 299 229 244 14 374 665 602 603 768 709 91 953 348 649 355 840.."  
  
matrices:  
      r(confusion) : 2 x 2  
  
.  
end of do-file
```

Figure 16: Convert the class of object to Matrix to return it to Stata

```
. matrix list r(confusion)
```

```
r(confusion)[2,2]
```

	no	yes
no	<b>55</b>	<b>10</b>
yes	<b>22</b>	<b>13</b>

Figure 17: Access the returned matrix in Stata

# Break a complex R object to a number of simple objects




- Complex objects can be thought of lists
  - might include datasets
  - matrices
  - scalars
  - arrays . . .
- You can unclass or slice a complex object into simple objects
- Simple objects must be recognized by Stata
  - numeric matrices
  - scalars
  - datasets
- Such objects will be returned by `rcall` automatically
- If you want to avoid returning an object, just remove it in the R code!

# Break a complex R object to a number of simple objects

```
. R
----- R (type end to exit) -----
. class(model)
[1] "C5.0"
. str(model)
List of 16
 $ names      : chr "| Generated using R version 4.1.0 (2021-05-18)|\n| on Thu Sep 09 10:02:16 2021\noutcome.\n\noutcome: n
> o,yes.\nch"| __truncated__
 $ cost       : chr ""
 $ costMatrix : NULL
 $ caseWeights: logi FALSE
 $ control    :List of 11
  ..$ subset  : logi TRUE
  ..$ bands   : num 0
  ..$ winnow   : logi FALSE
  ..$ noGlobalPruning: logi FALSE
  ..$ CF       : num 0.25
  ..$ minCases : num 2
  ..$ fuzzyThreshold : logi FALSE
  ..$ sample   : num 0
  ..$ earlyStopping : logi TRUE
  ..$ label    : chr "outcome"
  ..$ seed     : int 1098
 $ trials     : Named num [1:2] 1 1
  -- attr(*, "names")= chr [1:2] "Requested" "Actual"
 $ rbm        : logi FALSE
 $ boostResults: NULL
 $ size       : int 42
 $ dims       : int [1:2] 900 20
 $ call       : language C5.0.default(x = train[-17], y = train$default)
 $ levels     : chr [1:2] "no" "yes"
 $ output     : chr "\nC5.0 [Release 2.07 GPL Edition] \tThu Sep 9 10:02:16 2021\n-----\n\nnCla
> ss specifi"| __truncated__
 $ tree       : chr "id=\"See5/C5.0 2.07 GPL Edition 2021-09-09\"\nentries=\"1\"\nntype=\"3\" class=\"no\" freq=\"635,265\"
> att=\"che"| __truncated__
 $ predictors : chr [1:20] "checking_balance" "months_loan_duration" "credit_history" "purpose" ...
 $ rules      : chr ""
 - attr(*, "class")= chr "C5.0"
.
```

Figure 18: Viewing the model object

# Better, faster, and cleaner workflow?

```
1 # credit object must be defined before execution
2 # -----
3  credit$default <- as.factor(credit$default)
4
5 # set seed and define the train and test datasets
6 set.seed(123)
7 train_sample <- sample(1000, 900)
8  train <- credit[train_sample, ]
9  test <- credit[-train_sample, ]
10
11 library(C50)
12 model <- C5.0(train[-17], train$default)
13
14 prediction <- predict(model, test)
15 table(test$default, prediction,
16       dnn = c("actual default", "predicted default"))
17
```

```
1 // load the dataset for the analysis
2 use credit, clear
3 R clear // cleaning the R environment
4 R script "./examples/decisiontree.R", args(credit<-st.data())
5 return list
6
```

## Part 4: using `rcall` for writing Stata packages

# Stata programming with `rcall`

- `rcall` allows embedding R code inside Stata programs
- It also provides a strict procedures for executing the computation
  - checking for required dependencies (R version, versions of R packages)
  - making sure the analysis is reproducible, by starting a new R environment
  - providing tools for checking required Stata dependencies and `rcall` version
- Programming with `rcall` is best practiced if Stata syntax is adopted



# Example Stata programs utilizing `rcall`

- examples in the manuscript: Seamless interactive language interfacing between R and Stata
- example packages written by the community
  - type `github search rcall, all in(all)`

# github search rcall, all in(all)

Results			
<code>rcall</code>	<code>haghish</code>	<code>Install</code> 1968k	Seamless interactive R in Stata. rcall allows communicating data sets, matrices, variables, and scalars between Stata and R conveniently homepage <a href="http://www.haghish.com/packa~p">http://www.haghish.com/packa~p</a> updated on 2021-09-07 Fork:22 Star:64 Lang:Stata (dependency)
<code>did</code>	<code>NickCH-K</code>	<code>Install</code> 452k	A Stata package that acts as a wrapper for Callaway and Santannas R did package updated on 2021-08-19 Fork:11 Star:26 Lang:Stata
<code>stata-rcallst-t</code>	<code>luispfons~a</code>	<code>Install</code> 27k	Call Rs stringdist package from Stata using rcall updated on 2020-01-15 Fork:2 Star:1 Lang:Stata (dependency)
<code>importsav</code>	<code>jh-min</code>	<code>Install</code> 51k	Program to convert SPSS file to Stata (requires R) updated on 2020-07-27 Fork:0 Star:1 Lang:Stata
<code>stata-rcallco-e</code>	<code>luispfons~a</code>	<code>Install</code> 49k	Call Rs countrycode package from Stata using rcall updated on 2021-03-10 Fork:0 Star:0 Lang:Stata (dependency)
<code>cquadr</code>	<code>fravale</code>	<code>Install</code> 253k	Run the cquad R package in Stata by rcall updated on 2020-12-02 Fork:0 Star:0 Lang:Stata
<code>rcall</code>	<code>kapustinmax</code>	<code>Install</code> 43k	No description, website, or topics provided. updated on 2020-04-04 Fork:0 Star:0 Lang:Stata (dependency)
<code>ehcvm-tri-aut-e</code>	<code>arthur-shaw</code>		Trie automatiquement les entretiens de LEHCVm en trois tas : à rejeter, à regarder de plus près, à approuver updated on 2019-06-13 Fork:1 Star:1 Lang:Stata
<code>Should-We-Tal~r</code>	<code>edwardgol~g</code>		No description, website, or topics provided. updated on 2019-02-06 Fork:0 Star:0 Lang:Stata

Figure 19: Stata packages based on rcall

## Example 1: Loading data from CSV file

```
1 // Don't use this program at home! This is an example for loading data
2 // from R to Stata
3 // -----
4
5 program define rdata
6     version 14
7     syntax using/ , [stringsasfactor]
8     confirm file "`using'"
9
10    // should strings be converted as factors?
11    if "`stringsasfactor'" != "" local strfactor ", stringsAsFactors=TRUE"
12
13    // load the data in a new R session and pass it to Stata
14    rcall vanilla: df<-read.csv("`macval(using)'"`strfactor'); st.load(df)
15 end
16
```

Figure 20: example program for loading CSV files in Stata

```
//import delimited "credit.csv", clear
rdata using "credit.csv", stringsasfactor
```

# Is R installed? What R version is needed?

```
1 // Don't use this program at home! This is an example for loading data
2 //   from R to Stata
3 // -----
4
5 program define rdata
6   version 14
7   syntax using/ , [stringsasfactor]
8   confirm file "`using'"
9
10  //is R recognized by rcall?
11  //what is the minimum required R version
12  rcall_check , rversion(4.1)
13
14  // should strings be converted as factors?
15  if "`stringsasfactor'" != "" local strfactor " , stringsAsFactors=TRUE"
16
17  // load the data in a new R session and pass it to Stata
18  rcall vanilla: df<-read.csv("`macval(using)'"`strfactor'); st.load(df)
19 end
20
```

Figure 21: example program for loading CSV files in Stata

# What if error occurs?

```
1 // Don't use this program at home! This is an example for loading data
2 // from R to Stata
3 // -----
4
5 program define rdata
6     version 14
7     syntax using/ , [stringsasfactor]
8     confirm file "`using'"
9
10    //is R recognized by rcall?
11    //what is the minimum required R version
12    rcall_check , rversion(4.1)
13
14    // should strings be converted as factors?
15    if "`stringsasfactor'" != "" local strfactor " , stringsAsFactors=TRUE"
16
17    // load the data in a new R session and pass it to Stata
18    rcall vanilla: df<-read.csv("`macval(using)'"`strfactor'); st.load(df)
19
20    // in case of error, the program will stop and R's or Stata's error is returned
21    // otherwise, r(rc) = 0 is returned
22    if `r(rc)' == 0 {
23        di as txt "(write something here...)"
24    }
25
26 end
27
```

Figure 22: example program for loading CSV files in Stata

# Package versions & returning objects to Stata?

- you can specify a particular package version or a minimum version for dependencies
- check the version of the dependencies using `packageVersion("pkgname")` function

```
. R: packageVersion("C50")  
[1] '0.1.5'
```

- The `rcall_check` command can examine the required package dependencies as well
- `rcall` automatically returns objects from R to Stata function
  - include the `return add` command to pass the objects to the mother environment

```

1 program define c5, rclass
2   version 14
3   syntax [anything]
4
5   //make sure R is accessible to rcall
6   //make sure R is at least version 4.1.0
7   //make sure rcall is at least version 3.0.3
8   //make sure C50 is at least version 0.1.5
9   rcall_check C50>=0.1.5 , r(4.1.0) rcall(3.0.3)
10  rcall vanilla: hw = "Hello World"
11  return add // will pass the objects returned by rcall to the mother environment
12 end
13

```

## . return list

scalars:

`r(rc) = 0`

macros:

`r(hw) : "Hello World"`