

# Package ‘monobinShiny’

October 13, 2022

**Title** Shiny User Interface for 'monobin' Package

**Version** 0.1.0

**Maintainer** Andrija Djurovic <djandrija@gmail.com>

## Description

This is an add-on package to the 'monobin' package that simplifies its use. It provides shiny-based user interface (UI) that is especially handy for less experienced 'R' users as well as for those who intend to perform quick scanning of numeric risk factors when building credit rating models. The additional functions implemented in 'monobinShiny' that do not exist in 'monobin' package are: descriptive statistics, special case and outliers imputation. The function descriptive statistics is exported and can be used in 'R' sessions independently from the user interface, while special case and outlier imputation functions are written to be used with shiny UI.

**License** GPL (>= 3)

**URL** <https://github.com/andrija-djurovic/monobinShiny>

**Encoding** UTF-8

**RoxygenNote** 7.1.1

**Depends** DT, monobin, shiny, shinydashboard, shinyjs

**Imports** dplyr

**NeedsCompilation** no

**Author** Andrija Djurovic [aut, cre]

**Repository** CRAN

**Date/Publication** 2021-11-22 07:10:02 UTC

## R topics documented:

algo.ui . . . . .	2
check.vars . . . . .	3
cum.ui . . . . .	3

desc.report . . . . .	4
desc.stat . . . . .	5
di.server . . . . .	6
di.ui . . . . .	7
dm.server . . . . .	7
dm.ui . . . . .	8
hide.dwnl.buttons . . . . .	8
iso.ui . . . . .	9
mb.server . . . . .	10
mb.ui . . . . .	10
mdt.ui . . . . .	11
mono.inputs.check . . . . .	11
monobin.fun . . . . .	12
monobin.run . . . . .	13
monobinShinyApp . . . . .	14
ndr.sts.ui . . . . .	14
num.inputs . . . . .	15
out.impute . . . . .	16
pct.ui . . . . .	17
sc.check . . . . .	17
sc.impute . . . . .	18
sync.m23 . . . . .	19
sync.m23.imp . . . . .	20
upd.dm . . . . .	20
upd.si.m23 . . . . .	21
woe.ui . . . . .	22

**Index** **23**

---

algo.ui	<i>Server side for monobin functions' inputs</i>
---------	--

---

**Description**

Server side for monobin functions' inputs

**Usage**

algo.ui(id)

**Arguments**

id                      Namespace id.

**Value**

No return value, server side call for user interface of the selected binning algorithm.

**Examples**

```
if (interactive()) {  
  algo.ui(id = "monobin")  
}
```

---

check.vars	<i>Check for categorical variables when importing the data</i>
------------	--

---

**Description**

Check for categorical variables when importing the data

**Usage**

```
check.vars(tbl)
```

**Arguments**

tbl                    Imported data frame.

**Value**

Returns a character vector which describes variables type of imported data frame.

**Examples**

```
if (interactive()) {  
  check.msg <- check.vars(tbl = rv$db)  
}
```

---

cum.ui	<i>cum.bin - monobin functions' inputs</i>
--------	--

---

**Description**

cum.bin - monobin functions' inputs

**Usage**

```
cum.ui(id)
```

**Arguments**

id                    Namespace id.

**Value**

No return value, called for user interface of the cum.bin - monobin functions' inputs.

**Examples**

```
if (interactive()) {
  output$algo.args <- renderUI({tagList(
    switch(algo.select, "cum.bin" = cum.ui(id = id),
      "iso.bin" = iso.ui(id = id),
      "ndr.bin" = ndr.sts.ui(id = id),
      "sts.bin" = ndr.sts.ui(id = id),
      "pct.bin" = pct.ui(id = id),
      "woe.bin" = woe.ui(id = id),
      "mdt.bin" = mdt.ui(id = id)))
  })
}
```

---

desc.report

*Descriptive statistics report*

---

**Description**

Descriptive statistics report

**Usage**

```
desc.report(target, rf, sc, sc.method, db)
```

**Arguments**

target	Selected target.
rf	Vector of a selected numeric risk factors.
sc	Numeric vector of special case values.
sc.method	Define how special cases will be treated, all together or in separate bins.
db	Data frame of target and numeric risk factors.

**Value**

Returns a data frame with descriptive statistics for the selected risk drivers.

**Examples**

```
if (interactive()) {
  srv$desc.stat <- withProgress(message = "Running descriptive statistics report",
    value = 0, {
    desc.report(target = "qual",
      rf = rf,
      sc = sc,
```

```

sc.method = sc.method,
db = isolate(rv$db)
})
}

```

---

desc.stat

*Descriptive statistics*


---

### Description

desc.stat returns the descriptive statistics of numeric risk factor. Reported metrics covers mainly univariate and part of bivariate analysis which are usually standard steps in credit rating model development. Metrics are reported for special (if exists) and complete case groups separately. Report includes:

- risk.factor: Risk factor name.
- type: Special case or complete case group.
- bin: When special case method is together then bin is the same as type, otherwise all special cases are reported separately.
- cnt: Number of observations.
- pct: Percentage of observations.
- min: Minimum value.
- p1, p5, p25, p50, p75, p95, p99: Percentile values.
- avg: Mean value.
- avg.se: Standard error of mean.
- max: Maximum value.
- neg: Number of negative values.
- pos: Number of positive values.
- cnt.outliers: Number of outliers. Records above and below  $Q75 + 1.5 * IQR$ , where  $IQR = Q75 - Q25$ , where IQR is interquartile range.

### Usage

```
desc.stat(x, y, sc = c(NA, NaN, Inf), sc.method = "together")
```

### Arguments

x	Numeric risk factor.
y	Numeric target vector (binary or continuous).
sc	Numeric vector with special case elements. Default values are c(NA, NaN, Inf). Recommendation is to keep the default values always and add new ones if needed. Otherwise, if these values exist in x and are not defined in the sc vector, function will report the error.
sc.method	Define how special cases will be treated, all together or in separate bins. Possible values are "together", "separately".

**Value**

Data frame of descriptive statistics metrics, separately for complete and special case groups.

**Examples**

```
suppressMessages(library(monobinShiny))
data(gcd)
desc.stat(x = gcd$age, y = gcd$qual)
gcd$age[1:10] <- NA
gcd$age[50:75] <- Inf
desc.stat(x = gcd$age, y = gcd$qual, sc.method = "together")
desc.stat(x = gcd$age, y = gcd$qual, sc.method = "separately")
```

---

di.server

*Descriptive statistics and imputation module - server side*

---

**Description**

Descriptive statistics and imputation module - server side

**Usage**

```
di.server(id)
```

**Arguments**

id                    Namespace id.

**Value**

No return value, called for descriptive statistics and imputation module server side.

**Examples**

```
if (interactive()) {
  di.server(id = "desc.imputation")
}
```

---

`di.ui`*Descriptive statistics and imputation module - user interface*

---

**Description**

Descriptive statistics and imputation module - user interface

**Usage**

```
di.ui(id)
```

**Arguments**

`id`                    Namespace id.

**Value**

No return value, called for descriptive statistics and imputation module user interface.

**Examples**

```
if (interactive()) {  
  di.ui(id = "desc.imputation")  
}
```

---

`dm.server`*Data manager module - server side*

---

**Description**

Data manager module - server side

**Usage**

```
dm.server(id)
```

**Arguments**

`id`                    Namespace id.

**Value**

No return value, called for data manager module server side.

**Examples**

```
if (interactive()) {  
  dm.server(id = "data.manager")  
}
```

dm.ui

*Data manager module - user interface*

---

**Description**

Data manager module - user interface

**Usage**

```
dm.ui(id)
```

**Arguments**

id                    Namespace id.

**Value**

No return value, called for data manager module user interface.

**Examples**

```
if (interactive()) {  
  dm.ui(id = "data.manager")  
}
```

---

hide.dwnl.buttons*Hide download buttons from descriptive statistics module*

---

**Description**

Hide download buttons from descriptive statistics module

**Usage**

```
hide.dwnl.buttons(id)
```

**Arguments**

id                    Namespace id.

**Value**

No return value, called in order to hide download buttons (imp.div and out.div) from descriptive statistics module.



**Examples**

```
if (interactive()) {
  observeEvent(rv$dwnl.sync, {
    hide.dwnl.buttons(id = "desc.imputation")
  }, ignoreInit = TRUE)
}
```

---

iso.ui

*iso.bin - monobin functions' inputs*

---

**Description**

iso.bin - monobin functions' inputs

**Usage**

```
iso.ui(id)
```

**Arguments**

id                    Namespace id.

**Value**

No return value, called for user interface of the iso.bin - monobin functions' inputs.

**Examples**

```
if (interactive()) {
  output$algo.args <- renderUI({tagList(
    switch(algo.select, "cum.bin" = cum.ui(id = id),
      "iso.bin" = iso.ui(id = id),
      "ndr.bin" = ndr.sts.ui(id = id),
      "sts.bin" = ndr.sts.ui(id = id),
      "pct.bin" = pct.ui(id = id),
      "woe.bin" = woe.ui(id = id),
      "mdt.bin" = mdt.ui(id = id)))
  })
}
```

---

mb.server	<i>Monobin module - server side</i>
-----------	-------------------------------------

---

**Description**

Monobin module - server side

**Usage**

```
mb.server(id)
```

**Arguments**

id	Namespace id.
----	---------------

**Value**

No return value, called for monobin module server side.

**Examples**

```
if (interactive()) {  
  mb.server(id = "monobin")  
}
```

---

mb.ui	<i>Monobin module - user interface</i>
-------	--

---

**Description**

Monobin module - user interface

**Usage**

```
mb.ui(id)
```

**Arguments**

id	Namespace id.
----	---------------

**Value**

No return value, called for monobin module user interface.

**Examples**

```
if (interactive()) {  
  mb.ui(id = "monobin")  
}
```

---

mdt.ui	<i>mdt.bin - monobin functions' inputs</i>
--------	--

---

**Description**

mdt.bin - monobin functions' inputs

**Usage**

```
mdt.ui(id)
```

**Arguments**

id                    Namespace id.

**Value**

No return value, called for user interface of the iso.bin - monobin functions' inputs.

**Examples**

```
if (interactive()) {  
  output$algo.args <- renderUI({tagList(switch(algo.select, "cum.bin" = cum.ui(id = id),  
    "iso.bin" = iso.ui(id = id),  
    "ndr.bin" = ndr.sts.ui(id = id),  
    "sts.bin" = ndr.sts.ui(id = id),  
    "pct.bin" = pct.ui(id = id),  
    "woe.bin" = woe.ui(id = id),  
    "mdt.bin" = mdt.ui(id = id)))  
})  
}
```

---

mono.inputs.check	<i>Check for numeric arguments - monobin module</i>
-------------------	---

---

**Description**

Check for numeric arguments - monobin module

**Usage**

```
mono.inputs.check(x, args.e)
```

**Arguments**

x                    Binning algorithm from monobin package.  
args.e                Argument elements of the selected monobin function.

**Value**

Returns a list of two vectors: logical if validation is successful and character vector with validation message.

**Examples**

```
if (interactive()) {  
  num.inp <- mono.inputs.check(x = bin.algo, args.e = args.e)  
}
```

---

monobin.fun

*Evaluation expression of the selected monobin function and its arguments*

---

**Description**

Evaluation expression of the selected monobin function and its arguments

**Usage**

```
monobin.fun(x)
```

**Arguments**

x                    Binning algorithm from monobin package.

**Value**

Returns an evaluation expression of the selected monobin algorithm.

**Examples**

```
if (interactive()) {  
  expr.eval <- monobin.fun(x = algo)  
}  
monobin.fun(x = "ndr.bin")
```

---

`monobin.run`*Run monobin algorithm for the selected inputs*

---

**Description**

Run monobin algorithm for the selected inputs

**Usage**

```
monobin.run(algo, target.n, rf, sc, args.e, db)
```

**Arguments**

<code>algo</code>	Binning algorithm from monobin package.
<code>target.n</code>	Selected target.
<code>rf</code>	Vector of a selected numeric risk factors.
<code>sc</code>	Numeric vector of special case values.
<code>args.e</code>	Argument elements of the selected monobin function.
<code>db</code>	Data frame of target and numeric risk factors.

**Value**

Returns a list of two data frame. The first data frame contains the results of implemented binning algorithm, while the second one contains transformed risk factors.

**Examples**

```
if (interactive()) {  
  tbls <- withProgress(message = "Running the binning algorithm",  
    value = 0, {  
    suppressWarnings(  
      monobin.run(algo = bin.algo,  
        target.n = isolate(input$trg.select),  
        rf = isolate(input$rf.select),  
        sc = scr.check.res[[1]],  
        args.e = args.e,  
        db = isolate(rv$db))  
    })  
}
```

---

monobinShinyApp	<i>Starts shiny application for the monobin package</i>
-----------------	---

---

**Description**

Starts shiny application for the monobin package

**Usage**

```
monobinShinyApp()
```

**Value**

Starts shiny application for the monobin package.

**Examples**

```
if (interactive()) {  
  suppressMessages(library(monobinShiny))  
  monobinShinyApp()  
}
```

---

ndr.sts.ui	<i>ndr.bin / sts.bin - monobin functions' inputs</i>
------------	--

---

**Description**

ndr.bin / sts.bin - monobin functions' inputs

**Usage**

```
ndr.sts.ui(id)
```

**Arguments**

id	Namespace id.
----	---------------

**Value**

No return value, called for user interface of the ndr.bin / sts.bin - monobin functions' inputs.

**Examples**

```
if (interactive()) {
  output$algo.args <- renderUI({tagList(
    switch(algo.select, "cum.bin" = cum.ui(id = id),
      "iso.bin" = iso.ui(id = id),
      "ndr.bin" = ndr.sts.ui(id = id),
      "sts.bin" = ndr.sts.ui(id = id),
      "pct.bin" = pct.ui(id = id),
      "woe.bin" = woe.ui(id = id),
      "mdt.bin" = mdt.ui(id = id)))
  })
}
```

---

num.inputs

*Numeric arguments - monobin module*

---

**Description**

Numeric arguments - monobin module

**Usage**

```
num.inputs(x)
```

**Arguments**

x                    Binning algorithm from monobin package.

**Value**

Returns a list of two vectors: index and UI element label of numeric arguments of the selected monobin function.

**Examples**

```
if (interactive()) {
  inp.indx <- num.inputs(x = x)
}
num.inputs(x = "cum.bin")
```

---

out.impute	<i>Outliers imputation</i>
------------	----------------------------

---

## Description

Outliers imputation

## Usage

```
out.impute(tbl, rf, ub, lb, sc)
```

## Arguments

tbl	Data frame with risk factors ready for imputation.
rf	Vector of risk factors to be imputed.
ub	Upper bound percentiles.
lb	Lower bound percentiles.
sc	Numeric vector of special case values.

## Value

Returns a list of three elements. The first element is a data frame with imputed values, the second element is a vector of newly created risk factors (with imputed values) and the third one is a data frame with information about possible imputation errors.

## Examples

```
if (interactive()) {  
  imp.res <- suppressWarnings(  
    out.impute(tbl = rv$db,  
              rf = input$rf.out,  
              ub = upper.pct,  
              lb = lower.pct,  
              sc = sca.check.res[[1]])  
  )  
}
```





**Value**

Returns a list of three vectors: special case input(s) converted to numeric type, number of special case input(s) that cannot be converted to numeric type (including NA, NaN and Inf) and special case input(s) that cannot be converted to numeric type.

**Examples**

```
if (interactive()) {
  sca.check.res <- sc.check(x = input$sc.all)
  scr.check.res <- sc.check(x = input$sc.replace)
}
sc.check(x = "NA, NaN, Inf")
sc.check(x = "NA, abc")
sc.check(x = "NaN, abc")
sc.check(x = "Inf, abc")
sc.check(x = "9999999999, abc")
sc.check(x = "NA, NaN, Inf, 9999999999")
```

---

sc.impute

*Special case imputation*


---

**Description**

Special case imputation

**Usage**

```
sc.impute(tbl, rf, sc, sc.replace, imp.method)
```

**Arguments**

tbl	Data frame with risk factors ready for imputation.
rf	Vector of risk factors to be imputed.
sc	Numeric vector of special case values.
sc.replace	Numeric vector of special case values that are selected for imputation.
imp.method	Imputation method (mean or median).

**Value**

Returns a list of three elements. The first element is a data frame with imputed values, the second element is a vector of newly created risk factors (with imputed values) and the third one is a data frame with information about possible imputation errors.

**Examples**

```

if (interactive()) {
  imp.res <- suppressWarnings(
    sc.impute(tbl = rv$db,
              rf = rf,
              sc = sca.check.res[[1]],
              sc.replace = scr.check.res[[1]],
              imp.method = imp.method)
  )
}

```

---

sync.m23

*Sync between descriptive statistics and monobin module after data import*


---

**Description**

Sync between descriptive statistics and monobin module after data import

**Usage**

```
sync.m23(id, num.rf, module)
```

**Arguments**

id	Namespace id.
num.rf	Vector of updated numeric risk factors.
module	Descriptive statistic or monobin module.

**Value**

No return value, called in order to sync between descriptive statistics and monobin modules' UI elements after data import.

**Examples**

```

if (interactive()) {
  observeEvent(rv$sync, {
    sync.m23(id = "desc.imputation",
            num.rf = rv$num.rf,
            module = "desc")
    sync.m23(id = "monobin",
            num.rf = rv$num.rf,
            module = "monobin")
    rv$rf.imp <- NULL
    rv$rf.out <- NULL
  }, ignoreInit = TRUE)
}

```

---

sync.m23.imp	<i>Sync between descriptive statistics and monobin module after imputation process</i>
--------------	--

---

**Description**

Sync between descriptive statistics and monobin module after imputation process

**Usage**

```
sync.m23.imp(id, num.rf, module)
```

**Arguments**

id	Namespace id.
num.rf	Vector of updated numeric risk factors.
module	Descriptive statistic or monobin module.

**Value**

No return value, called in order to sync between descriptive statistics and monobin modules' UI elements after imputation process.

**Examples**

```
if (interactive()) {
  observeEvent(rv$sync2, {
    rf.update.2 <- c(rv$num.rf[!rv$num.rf%in%rv$target.select.2], rv$rf.imp, rv$rf.out)
    sync.m23.imp(id = "desc.imputation",
      num.rf = rf.update.2,
      module = "desc")
  }, ignoreInit = TRUE)
}
```

---

upd.dm	<i>Update data manager UI output</i>
--------	--------------------------------------

---

**Description**

Update data manager UI output

**Usage**

```
upd.dm(id, dummy)
```

**Arguments**

id	Namespace id.
dummy	A logical value indicating whether gcd data (from monobin package) or specific csv file are imported.

**Value**

No return value, called in order to update data manager UI output after data import.

**Examples**

```
if (interactive()) {
  observeEvent(rv$dm.uptd, {
    upd.dm(id = "data.manager", dummy = rv$import.dummy)
  }, ignoreInit = TRUE)
}
```

---

 upd.si.m23

---

*Sync between descriptive statistics and monobin module*


---

**Description**

Sync between descriptive statistics and monobin module

**Usage**

```
upd.si.m23(upd.rf, num.rf, session)
```

**Arguments**

upd.rf	Vector of risk factor field ids that need to be updated.
num.rf	Vector of updated numeric risk factors.
session	Session object.

**Value**

No return value, called in order to sync between descriptive statistics and monobin modules' UI elements after imputation procedures.

**Examples**

```
if (interactive()) {
  upd.si.m23(upd.rf = upd.rf,
            num.rf = num.rf,
            session = session)
}
```

---

woe.ui                      *woe.bin - monobin functions' inputs*

---

**Description**

woe.bin - monobin functions' inputs

**Usage**

```
woe.ui(id)
```

**Arguments**

id                      Namespace id.

**Value**

No return value, called for user interface of the woe.bin - monobin functions' inputs.

**Examples**

```
if (interactive()) {  
output$algo.args <- renderUI({tagList(switch(algo.select, "cum.bin" = cum.ui(id = id),  
  "iso.bin" = iso.ui(id = id),  
  "ndr.bin" = ndr.sts.ui(id = id),  
  "sts.bin" = ndr.sts.ui(id = id),  
  "pct.bin" = pct.ui(id = id),  
  "woe.bin" = woe.ui(id = id),  
  "mdt.bin" = mdt.ui(id = id)))  
})  
}
```

# Index

algo.ui, [2](#)

check.vars, [3](#)  
cum.ui, [3](#)

desc.report, [4](#)  
desc.stat, [5](#)  
di.server, [6](#)  
di.ui, [7](#)  
dm.server, [7](#)  
dm.ui, [8](#)

hide.dwnl.buttons, [8](#)

iso.ui, [9](#)

mb.server, [10](#)  
mb.ui, [10](#)  
mdt.ui, [11](#)  
mono.inputs.check, [11](#)  
monobin.fun, [12](#)  
monobin.run, [13](#)  
monobinShinyApp, [14](#)

ndr.sts.ui, [14](#)  
num.inputs, [15](#)

out.impute, [16](#)

pct.ui, [17](#)

sc.check, [17](#)  
sc.impute, [18](#)  
sync.m23, [19](#)  
sync.m23.imp, [20](#)

upd.dm, [20](#)  
upd.si.m23, [21](#)

woe.ui, [22](#)