

# Package: bfw (via r-universe)

October 11, 2024

**Title** Bayesian Framework for Computational Modeling

**Version** 0.4.2

**Date** 2022-02-22

**Maintainer** Øystein Olav Skaar <bayesianfw@gmail.com>

**Description** Derived from the work of Kruschke (2015, <ISBN:9780124058880>), the present package aims to provide a framework for conducting Bayesian analysis using Markov chain Monte Carlo (MCMC) sampling utilizing the Just Another Gibbs Sampler (JAGS', Plummer, 2003, <<https://mcmc-jags.sourceforge.io>>). The initial version includes several modules for conducting Bayesian equivalents of chi-squared tests, analysis of variance (ANOVA), multiple (hierarchical) regression, softmax regression, and for fitting data (e.g., structural equation modeling).

**License** MIT + file LICENSE

**URL** <https://github.com/oeysan/bfw/>

**BugReports** <https://github.com/oeysan/bfw/issues/>

**Depends** R (>= 3.5.0)

**Imports** circlize (>= 0.4.4), coda (>= 0.19-1), data.table (>= 1.12.2), dplyr (>= 0.7.7), ggplot2 (>= 2.2.1), graphics, grDevices, grid, magrittr (>= 1.5), MASS (>= 7.3-47), officer (>= 0.3.1), parallel, plyr (>= 1.8.4), png (>= 0.1-7), runjags (>= 2.0.4-2), rvg (>= 0.1.9), scales (>= 0.5.0), stats, utils

**Suggests** testthat (>= 3.0.0), knitr (>= 1.20), lavaan (>= 0.6-1), psych (>= 1.7.8), rmarkdown (>= 1.10)

**VignetteBuilder** knitr

**Encoding** UTF-8

**LazyData** true

**NeedsCompilation** no

**RoxygenNote** 7.1.2

**SystemRequirements** JAGS >=4.3.0 <<https://mcmc-jags.sourceforge.io>>,  
Java JDK >=1.4 <<https://www.java.com/en/download/manual.jsp>>

**Config/testthat/edition** 3

**Repository** <https://oeysan.r-universe.dev>

**RemoteUrl** <https://github.com/oeysan/bfw>

**RemoteRef** HEAD

**RemoteSha** 9d2b94c2b4fa42681bc3be7bd6063810e7612200

## Contents

|                 |    |
|-----------------|----|
| AddNames        | 3  |
| bfw             | 4  |
| CapWords        | 6  |
| Cats            | 7  |
| ChangeNames     | 7  |
| ComputeHDI      | 8  |
| ContrastNames   | 9  |
| DiagMCMC        | 9  |
| DistinctColors  | 10 |
| ETA             | 11 |
| FileName        | 11 |
| FindEnvironment | 12 |
| FlattenList     | 13 |
| GammaDist       | 14 |
| GetRange        | 14 |
| Interleave      | 15 |
| InverseHDI      | 16 |
| Layout          | 17 |
| MatrixCombn     | 17 |
| MergeMCMC       | 18 |
| MultiGrep       | 19 |
| Normalize       | 19 |
| PadVector       | 20 |
| ParseNumber     | 20 |
| ParsePlot       | 21 |
| PlotCirclize    | 22 |
| PlotData        | 23 |
| PlotMean        | 24 |
| PlotNominal     | 25 |
| PlotParam       | 26 |
| ReadFile        | 27 |
| RemoveEmpty     | 28 |
| RemoveGarbage   | 29 |
| RemoveSpaces    | 29 |
| RunContrasts    | 30 |
| RunMCMC         | 30 |

|                           |    |
|---------------------------|----|
| SingleString . . . . .    | 33 |
| StatsBernoulli . . . . .  | 33 |
| StatsCovariate . . . . .  | 35 |
| StatsFit . . . . .        | 37 |
| StatsKappa . . . . .      | 39 |
| StatsMean . . . . .       | 40 |
| StatsMetric . . . . .     | 41 |
| StatsNominal . . . . .    | 42 |
| StatsRegression . . . . . | 43 |
| StatsSoftmax . . . . .    | 44 |
| SumMCMC . . . . .         | 46 |
| SumToZero . . . . .       | 47 |
| TidyCode . . . . .        | 47 |
| Trim . . . . .            | 48 |
| TrimSplit . . . . .       | 48 |
| VectorSub . . . . .       | 49 |

## Index 51

---

|          |                  |
|----------|------------------|
| AddNames | <i>Add Names</i> |
|----------|------------------|

---

### Description

Add names to columns from naming list

### Usage

```
AddNames(
  par,
  job.names,
  job.group = NULL,
  keep.par = TRUE,
  names.only = FALSE,
  ...
)
```

### Arguments

|            |                                                                                                                                 |
|------------|---------------------------------------------------------------------------------------------------------------------------------|
| par        | defined parameter to analyze (e.g., "cor[1,2]")                                                                                 |
| job.names  | names of all parameters in analysis, Default: NULL                                                                              |
| job.group  | for some hierarchical models with several layers of parameter names (e.g., latent and observed parameters), Default: NULL       |
| keep.par   | logical, indicating whether or not to keep parameter name (e.g., "cor[1,2]"), Default: TRUE                                     |
| names.only | logical, indicating whether or not to return vector (TRUE) or string with separator (e.g., "cor[1,2]: A vs. B"), Default: FALSE |
| ...        | further arguments passed to or from other methods                                                                               |

## Examples

```
par <- "cor[1,2]"
job.names <- c("A","B")
AddNames(par, job.names, keep.par = TRUE)
# [1] "cor[1,2]: A vs. B"
AddNames(par, job.names, keep.par = FALSE)
# [1] "A vs. B"
AddNames(par, job.names, names.only = TRUE)
# [1] "A" "B"
```

---

bfw

*Settings*

---

## Description

main settings for bfw

## Usage

```
bfw(
  job.title = NULL,
  job.group = NULL,
  jags.model,
  jags.seed = NULL,
  jags.method = NULL,
  jags.chains = NULL,
  custom.function = NULL,
  custom.model = NULL,
  params = NULL,
  saved.steps = 10000,
  thinned.steps = 1,
  adapt.steps = NULL,
  burnin.steps = NULL,
  initial.list = list(),
  custom.name = NULL,
  project.name = "Project",
  project.dir = "Results/",
  project.data = NULL,
  time.stamp = TRUE,
  save.data = FALSE,
  data.set = "AllData",
  data.format = "csv",
  raw.data = FALSE,
  run.robust = FALSE,
  merge.MCMC = FALSE,
  run.diag = FALSE,
  sep = ",")
```

```

    silent = FALSE,
    ...
)

```

### Arguments

|                              |                                                                                                                           |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>job.title</code>       | title of analysis, Default: NULL                                                                                          |
| <code>job.group</code>       | for some hierarchical models with several layers of parameter names (e.g., latent and observed parameters), Default: NULL |
| <code>jags.model</code>      | specify which module to use                                                                                               |
| <code>jags.seed</code>       | specify seed to replicate a analysis, Default: NULL                                                                       |
| <code>jags.method</code>     | specify method for JAGS (e.g., parallel or simple), Default: NULL                                                         |
| <code>jags.chains</code>     | specify specify number of chains for JAGS, Default: NULL                                                                  |
| <code>custom.function</code> | custom function to use (e.g., defined function, external R file or string with function), Default: NULL                   |
| <code>custom.model</code>    | define a custom model to use (e.g., string or text file (.txt), Default: NULL                                             |
| <code>params</code>          | define parameters to observe, Default: NULL                                                                               |
| <code>saved.steps</code>     | define the number of iterations/steps/chains in the MCMC simulations, Default: 10000                                      |
| <code>thinned.steps</code>   | save every kth step of the original saved.steps, Default: 1                                                               |
| <code>adapt.steps</code>     | the number of adaptive iterations to use at the start of each simulation, Default: NULL                                   |
| <code>burnin.steps</code>    | the number of burnin iterations, NOT including the adaptive iterations to use for the simulation, Default: NULL           |
| <code>initial.list</code>    | initial values for analysis, Default: list()                                                                              |
| <code>custom.name</code>     | custom name of project, Default: NULL                                                                                     |
| <code>project.name</code>    | name of project, Default: 'Project'                                                                                       |
| <code>project.dir</code>     | define where to save data, Default: 'Results/'                                                                            |
| <code>project.data</code>    | define data to use for analysis (e.g., csv, rda, custom data.frame or matrix, or data included in package, Default: NULL  |
| <code>time.stamp</code>      | logical, indicating whether or not to append unix time stamp to file name, Default: TRUE                                  |
| <code>save.data</code>       | logical, indicating whether or not to save data, Default: FALSE                                                           |
| <code>data.set</code>        | define subset of data, Default: 'AllData'                                                                                 |
| <code>data.format</code>     | define what data format is being used, Default: 'csv'                                                                     |
| <code>raw.data</code>        | logical, indicating whether or not to use unprocessed data, Default: FALSE                                                |
| <code>run.robust</code>      | logical, indicating whether or not robust analysis, Default: FALSE                                                        |
| <code>merge.MCMC</code>      | logical, indicating whether or not to merge MCMC chains, Default: FALSE                                                   |
| <code>run.diag</code>        | logical, indicating whether or not to run diagnostics, Default: FALSE                                                     |
| <code>sep</code>             | symbol to separate data (e.g., comma-delimited), Default: ','                                                             |
| <code>silent</code>          | logical, indicating whether or not to run analysis without output, Default: FALSE                                         |
| <code>...</code>             | further arguments passed to or from other methods                                                                         |

**Details**

Settings act like the main framework for `bfw`, connecting function, model and JAGS.

**Value**

data from MCMC [RunMCMC](#)

**See Also**

[head](#), [modifyList](#), [capture.output](#)

---

CapWords

*Capitalize Words*

---

**Description**

capitalize the first letter in each words in a string

**Usage**

```
CapWords(s, strict = FALSE)
```

**Arguments**

|                     |                                                                                 |
|---------------------|---------------------------------------------------------------------------------|
| <code>s</code>      | string                                                                          |
| <code>strict</code> | logical, indicating whether or not string it set to title case , Default: FALSE |

**Value**

returns capitalized string

**Examples**

```
CapWords("example eXAMPLE", FALSE)
# [1] "Example EXAMPLE"
CapWords("example eXAMPLE", TRUE)
# [1] "Example Example"
```

---

|      |                          |
|------|--------------------------|
| Cats | <i>Dataset with Cats</i> |
|------|--------------------------|

---

**Description**

Shamelessly adapted from Field (2017).

**Usage**

Cats

**Format**

A data frame with 2000 rows and 4 variables:

Reward integer Food or Affection

Dance integer Yes or No

Alignment integer Good or Evil

Ratings double Cats rate their owners (average of multiple seven-point Likert-type scale (1 = Hate ... 7 = Love))

**Details**

Example data for BFW

---

|             |                     |
|-------------|---------------------|
| ChangeNames | <i>Change Names</i> |
|-------------|---------------------|

---

**Description**

Change names, colnames or rownames of single items or a list of items

**Usage**

```
ChangeNames(  
  x,  
  names,  
  single.items = FALSE,  
  row.names = FALSE,  
  param = NULL,  
  where = NULL,  
  environment = NULL  
)
```

**Arguments**

|              |                                                                                                  |
|--------------|--------------------------------------------------------------------------------------------------|
| x            | list, vector, matrix, dataframe or a list of such items                                          |
| names        | names to insert                                                                                  |
| single.items | logical, indicating whether or not to use names rather than colnames or rownames, Default: FALSE |
| row.names    | logical, indicating whether or not to use rownames rather than colnames, Default: FALSE          |
| param        | Variable name, Default: NULL                                                                     |
| where        | select parents, Default: NULL                                                                    |
| environment  | select reference environment, Default: NULL                                                      |

**Value**

returns Named items # ABC <- c("1","2","3") # "1" "2" "3" # ChangeNames(ABC, names = c("A","B","C"), single.items = TRUE) # A B C # "1" "2" "3"

---

 ComputeHDI

*Compute HDI*


---

**Description**

Compute highest density interval (HDI) from posterior output

**Usage**

```
ComputeHDI(data, credible.region)
```

**Arguments**

|                 |                                                                                                                          |
|-----------------|--------------------------------------------------------------------------------------------------------------------------|
| data            | data to compute HDI from                                                                                                 |
| credible.region | summarize uncertainty by defining a region of most credible values (e.g., 95 percent of the distribution), Default: 0.95 |

**Details**

values within the HDI have higher probability density than values outside the HDI, and the values inside the HDI have a total probability equal to the credible region (e.g., 95 percent).

**Value**

Return HDI



**Examples**

```

set.seed(1)
data <- rnorm(100, 0, 1)
credible.region <- 0.95
ComputeHDI(data, credible.region)
# HDIlo HDIhi
# -1.99 1.60

```

---

ContrastNames

*Contrast Names*


---

**Description**

utilize the AddNames function to create contrast names

**Usage**

```
ContrastNames(items, job.names, col.names)
```

**Arguments**

|           |                                                    |
|-----------|----------------------------------------------------|
| items     | items to create names for                          |
| job.names | names of all parameters in analysis, Default: NULL |
| col.names | columns in MCMC to create names from               |

---

DiagMCMC

*Diagnose MCMC*


---

**Description**

MCMC convergence diagnostics

**Usage**

```

DiagMCMC(
  data.MCMC,
  par.name,
  job.names,
  job.group,
  credible.region = 0.95,
  monochrome = TRUE,
  plot.colors = c("#495054", "#e3e8ea")
)

```

**Arguments**

|                              |                                                                                                                           |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>data.MCMC</code>       | MCMC chains to diagnose                                                                                                   |
| <code>par.name</code>        | parameter to analyze                                                                                                      |
| <code>job.names</code>       | names of all parameters in analysis, Default: NULL                                                                        |
| <code>job.group</code>       | for some hierarchical models with several layers of parameter names (e.g., latent and observed parameters), Default: NULL |
| <code>credible.region</code> | summarize uncertainty by defining a region of most credible values (e.g., 95 percent of the distribution), Default: 0.95  |
| <code>monochrome</code>      | logical, indicating whether or not to use monochrome colors, else use <a href="#">DistinctColors</a> , Default: TRUE      |
| <code>plot.colors</code>     | range of color to use, Default: <code>c("#495054", "#e3e8ea")</code>                                                      |

**Value**

list of diagnostic plots

**See Also**

[dev.new](#), [colorRampPalette](#), [recordPlot](#), [graphics.off](#), [dev.list](#), [dev.off](#) `par`, [layout](#), [plot.new](#), [matplot](#), [abline](#), [text](#), [traceplot](#), [gelman.plot](#), [effectiveSize](#) `sd`, [acf](#), [density](#)

---

DistinctColors

*Distinct Colors*

---

**Description**

create vector containing Hex color codes

**Usage**

```
DistinctColors(range, random = FALSE)
```

**Arguments**

|                     |                                                                             |
|---------------------|-----------------------------------------------------------------------------|
| <code>range</code>  | number of colors as sequence                                                |
| <code>random</code> | logical, indicating whether or not to provide random colors, Default: FALSE |

**Examples**

```
DistinctColors(1:3)
# [1] "#FFFF00" "#1CE6FF" "#FF34FF"
set.seed(1)
DistinctColors(1:3, TRUE)
# [1] "#575329" "#CB7E98" "#D86A78"
```

---

ETA

*ETA*

---

### Description

Print estimated time for arrival (ETA)

### Usage

```
ETA(start.time, i, total, results = NULL)
```

### Arguments

|                         |                                                            |
|-------------------------|------------------------------------------------------------|
| <code>start.time</code> | start time (preset variable with <code>Sys.time()</code> ) |
| <code>i</code>          | incremental steps towards total                            |
| <code>total</code>      | total number of steps                                      |
| <code>results</code>    | message to display, Default: NULL                          |

### See Also

[flush.console](#)

---

FileName

*File Name*

---

### Description

simple function to construct a file name for data

### Usage

```
FileName(  
  project = "Project",  
  subset = NULL,  
  type = NULL,  
  name = NULL,  
  unix = TRUE,  
  ...  
)
```

**Arguments**

|         |                                                                            |
|---------|----------------------------------------------------------------------------|
| project | name of project, Default: 'Project'                                        |
| subset  | define subset of data, Default: NULL                                       |
| type    | type of data, Default: NULL                                                |
| name    | save name, Default: NULL                                                   |
| unix    | logical, indicating whether or not to append unix timestamp, Default: TRUE |
| ...     | further arguments passed to or from other methods                          |

**Examples**

```

FileName()
# [1] "Project-Name-1528834963"

FileName(project = "Project" ,
         subset = "subset" ,
         type = "longitudinal" ,
         name = "cheese" ,
         unix = FALSE)
# [1] "Projectsubset-longitudinal-cheese"

```

---

|                 |                         |
|-----------------|-------------------------|
| FindEnvironment | <i>Find Environment</i> |
|-----------------|-------------------------|

---

**Description**

Find the environment of a selected variable.

**Usage**

```
FindEnvironment(x, where = NULL)
```

**Arguments**

|       |                                             |
|-------|---------------------------------------------|
| x     | any type of named object                    |
| where | select reference environment, Default: NULL |

**Value**

returns Found environment, Default: R\_GlobalEnv.

---

|             |                     |
|-------------|---------------------|
| FlattenList | <i>Flatten List</i> |
|-------------|---------------------|

---

**Description**

flatten a nested list into a single list

**Usage**

```
FlattenList(li, rm.duplicated = TRUE, unname.li = TRUE, rm.empty = TRUE)
```

**Arguments**

|                            |                                                                              |
|----------------------------|------------------------------------------------------------------------------|
| <code>li</code>            | list to flatten                                                              |
| <code>rm.duplicated</code> | logical, indicating whether or not to remove duplicated lists, Default: TRUE |
| <code>unname.li</code>     | logical, indicating whether or not to unname lists, Default: TRUE            |
| <code>rm.empty</code>      | logical, indicating whether or not to remove empty lists, Default: TRUE      |

**Examples**

```
li <- list(LETTERS[1:3],
          list(letters[1:3],
              list(LETTERS[4:6])),
          DEF = letters[4:6],
          LETTERS[1:3],
          list() # Empty list
)
print(li)
# [[1]]
# [1] "A" "B" "C"
#
# [[2]]
# [[2]][[1]]
# [1] "a" "b" "c"
#
# [[2]][[2]]
# [[2]][[2]][[1]]
# [1] "D" "E" "F"
#
#
# $DEF
# [1] "d" "e" "f"
#
# [[4]]
# [1] "A" "B" "C"
#
# [[5]]
# list()
```

```
FlattenList(li)
# [[1]]
# [1] "A" "B" "C"
#
# [[2]]
# [1] "a" "b" "c"
#
# [[3]]
# [1] "D" "E" "F"
#
# [[4]]
# [1] "d" "e" "f"
```

---

GammaDist

*Gamma Distribution*

---

### Description

compute gamma distribution (shape and rate) from mode and standard deviation

### Usage

```
GammaDist(mode, sd)
```

### Arguments

|      |                              |
|------|------------------------------|
| mode | mode from data               |
| sd   | standard deviation from data |

### Examples

```
GammaDist(1,0.5)
# $shape
# [1] 5.828427
# $rate
# [1] 4.828427
```

---

GetRange

*Get Range*

---

### Description

simple function to extract columns from data frame

### Usage

```
GetRange(var, range = 1:8, df)
```

**Arguments**

var                variable of interest (e.g., V)  
 range             range of variables with same stem name (e.g., V1, V2, ..., V8) , Default: 1:8  
 df                 data to extract from

**Examples**

```
data <- as.data.frame(matrix(1:80,ncol=8))
GetRange("V", c(1,4), data)
#   V1 V4
# 1   1 31
# 2   2 32
# 3   3 33
# 4   4 34
# 5   5 35
# 6   6 36
# 7   7 37
# 8   8 38
# 9   9 39
# 10 10 40
```

---

 Interleave

*Interleave*


---

**Description**

mix vectors by alternating between them

**Usage**

```
Interleave(a, b)
```

**Arguments**

a                 first vector  
 b                 second vector

**Value**

mixed vector

**Examples**

```
a <- 1:3
b <- LETTERS[1:3]
Interleave(a,b)
# [1] "1" "A" "2" "B" "3" "C"
```

---

`InverseHDI`*Compute Inverse HDI*

---

**Description**

Compute inverse cumulative density function of the distribution

**Usage**

```
InverseHDI(  
  beta,  
  shape1,  
  shape2,  
  credible.region = 0.95,  
  tolerance = 0.00000001  
)
```

**Arguments**

|                              |                                                                                                                                                               |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>beta</code>            | density, distribution function, quantile function and random generation for the Beta distribution with parameters <code>shape1</code> and <code>shape2</code> |
| <code>shape1</code>          | non-negative parameter of the Beta distribution.                                                                                                              |
| <code>shape2</code>          | non-negative parameter of the Beta distribution.                                                                                                              |
| <code>credible.region</code> | summarize uncertainty by defining a region of most credible values (e.g., 95 percent of the distribution), Default: 0.95                                      |
| <code>tolerance</code>       | the desired accuracy, Default: 1e-8                                                                                                                           |

**Details**

values within the HDI have higher probability density than values outside the HDI, and the values inside the HDI have a total probability equal to the credible region (e.g., 95 percent).

**Value**

Return HDI

**See Also**

[Beta](#), [optimize](#)

**Examples**

```
InverseHDI( qbeta , 554 , 149 )  
# HDIlo HDIhi  
# 0.758 0.818
```



---

Layout

*Layout*

---

**Description**

collection of layout sizes

**Usage**

```
Layout(x = "a4", layout.inverse = FALSE)
```

**Arguments**

`x` type of layout, Default: 'a4'  
`layout.inverse` logical, indicating whether or not to inverse layout (e.g., landscape) , Default: FALSE

**Value**

width and height of select medium

**Examples**

```
Layout()  
# [1] 8.3 11.7
```

---

MatrixCombn

*Matrix Combinations*

---

**Description**

Create matrices from combinations of columns

**Usage**

```
MatrixCombn(  
  matrix,  
  first.stem,  
  last.stem = NULL,  
  q.levels,  
  rm.last = TRUE,  
  row.means = TRUE  
)
```

**Arguments**

|                         |                                                                                                                 |
|-------------------------|-----------------------------------------------------------------------------------------------------------------|
| <code>matrix</code>     | matrix to combine                                                                                               |
| <code>first.stem</code> | first name of columns to use (e.g., "m" for mean)                                                               |
| <code>last.stem</code>  | optional last name of columns to use (e.g., "p" for proportions) , Default: NONE                                |
| <code>q.levels</code>   | number of levels per column                                                                                     |
| <code>rm.last</code>    | logical, indicating whether or not to remove last combination (i.e., m1m2m3m4) , Default: TRUE                  |
| <code>row.means</code>  | logical, indicating whether or not to compute row means from combined columns, else use row sums, Default: TRUE |

---

MergeMCMC

---

*Merge MCMC*


---

**Description**

Merge two or more MCMC simulations

**Usage**

```
MergeMCMC(pat, project.dir = "Results/", data.sets)
```

**Arguments**

|                          |                                                |
|--------------------------|------------------------------------------------|
| <code>pat</code>         | pattern to select MCMC chain from              |
| <code>project.dir</code> | define where to save data, Default: 'Results/' |
| <code>data.sets</code>   | data sets to combine                           |

**Value**

Merged MCMC chains

**See Also**

[head combine.mcmc](#)

---

MultiGrep

*Multi Grep*

---

### Description

Use multiple patterns from vector to find element in another vector, with option to remove certain patterns

### Usage

```
MultiGrep(find, from, remove = NULL, value = TRUE)
```

### Arguments

|        |                                               |
|--------|-----------------------------------------------|
| find   | vector to find                                |
| from   | vector to find from                           |
| remove | variables to remove, Default: NULL            |
| value  | logical, if TRUE returns value, Default: TRUE |

---

Normalize

*Normalize*

---

### Description

simple function to normalize data

### Usage

```
Normalize(x)
```

### Arguments

|   |                             |
|---|-----------------------------|
| x | numeric vector to normalize |
|---|-----------------------------|

### Examples

```
Normalize(1:10)
# [1] 0.0182 0.0364 0.0545 0.0727 0.0909
# 0.1091 0.1273 0.1455 0.1636 0.1818
```

PadVector

*Pad Vector*

---

**Description**

Pad a numeric vector according to the highest value

**Usage**

```
PadVector(v)
```

**Arguments**

v                    numeric vector to pad

**Examples**

```
PadVector(1:10)
# [1] "01" "02" "03" "04" "05" "06" "07" "08" "09" "10"
```

---

ParseNumber

*Parse Numbers*

---

**Description**

simple function to extract numbers from string/vector

**Usage**

```
ParseNumber(x, digits = FALSE)
```

**Arguments**

x                    string or vector  
digits                logical, indicating whether or not to extract decimals, Default: FALSE

**See Also**

[na.omit](#)

**Examples**

```
ParseNumber("String1WithNumbers2")
# [1] 1 2
```

ParsePlot

*Parse Plot***Description**

Display and/or save plots

**Usage**

```
ParsePlot(
  plot.data,
  project.dir = "Results/",
  project.name = FileName(name = "Print"),
  graphic.type = "pdf",
  plot.size = "15,10",
  scaling = 100,
  plot.aspect = NULL,
  save.data = FALSE,
  vector.graphic = FALSE,
  point.size = 12,
  font.type = "serif",
  one.file = TRUE,
  ppi = 300,
  units = "in",
  layout = "a4",
  layout.inverse = FALSE,
  return.files = FALSE,
  ...
)
```

**Arguments**

|                             |                                                                                                       |
|-----------------------------|-------------------------------------------------------------------------------------------------------|
| <code>plot.data</code>      | a list of plots                                                                                       |
| <code>project.dir</code>    | define where to save data, Default: 'Results/'                                                        |
| <code>project.name</code>   | define name of project, Default: 'FileName(name="Print")'                                             |
| <code>graphic.type</code>   | type of graphics to use (e.g., pdf, png, ps), Default: 'pdf'                                          |
| <code>plot.size</code>      | size of plot, Default: '15,10'                                                                        |
| <code>scaling</code>        | scale size of plot, Default: 100                                                                      |
| <code>plot.aspect</code>    | aspect of plot, Default: NULL                                                                         |
| <code>save.data</code>      | logical, indicating whether or not to save data, Default: FALSE                                       |
| <code>vector.graphic</code> | logical, indicating whether or not visualizations should be vector or raster graphics, Default: FALSE |
| <code>point.size</code>     | point size used for visualizations, Default: 12                                                       |
| <code>font.type</code>      | font type used for visualizations, Default: 'serif'                                                   |

|                             |                                                                                                           |
|-----------------------------|-----------------------------------------------------------------------------------------------------------|
| <code>one.file</code>       | logical, indicating whether or not visualizations should be placed in one or several files, Default: TRUE |
| <code>ppi</code>            | define pixel per inch used for visualizations, Default: 300                                               |
| <code>units</code>          | define unit of length used for visualizations, Default: 'in'                                              |
| <code>layout</code>         | define a layout size for visualizations, Default: 'a4'                                                    |
| <code>layout.inverse</code> | logical, indicating whether or not to inverse layout (e.g., landscape) , Default: FALSE                   |
| <code>return.files</code>   | logical, indicating whether or not to return saved file names                                             |
| <code>...</code>            | further arguments passed to or from other methods                                                         |

**See Also**

[dev](#), [png](#), [ps.options](#), [recordPlot](#) [head](#) [readPNG](#) [par](#), [plot](#), [rasterImage](#) [read\\_pptx](#), [add\\_slide](#), [ph\\_with dml](#)

**Examples**

```
# Create three plots
plot.data <- lapply(1:3, function (i) {
  # Open new device
  grDevices::dev.new()
  # Print plot
  plot(1:i)
  # Record plot
  p <- grDevices::recordPlot()
  # Turn off graphics device drive
  grDevices::dev.off()
  return (p)
} )

# Print plots
ParsePlot(plot.data)
```

---

PlotCirclize

*Circlize Plot*


---

**Description**

Create a circlize plot

**Usage**

```
PlotCirclize(
  data,
  category.spacing = 1.2,
  category.inset = c(-0.4, 0),
  monochrome = TRUE,
```

```

    plot.colors = c("#CCCCCC", "#DEDEDE"),
    font.type = "serif"
)

```

### Arguments

|                               |                                                                                                                       |
|-------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| <code>data</code>             | data for circlize plot                                                                                                |
| <code>category.spacing</code> | spacing between category items , Default: 1.25                                                                        |
| <code>category.inset</code>   | inset of category items form plot , Default: c(-0.5, 0)                                                               |
| <code>monochrome</code>       | logical, indicating whether or not to use monochrome colors, else use <a href="#">Distinct-Colors</a> , Default: TRUE |
| <code>plot.colors</code>      | range of color to use, Default: c("#CCCCCC", "#DEDEDE")                                                               |
| <code>font.type</code>        | font type used for visualizations, Default: 'serif'                                                                   |

### See Also

[dev](#), [recordPlot](#) [legend](#) [circos.par](#), [chordDiagram](#), [circos.trackPlotRegion](#), [circos.clear](#)

---

PlotData

*Plot Data*

---

### Description

Plot data as violin plot visualizing density, box plots to display HDI, whiskers to display standard deviation

### Usage

```
PlotData(data, data.type = "Mean", ...)
```

### Arguments

|                        |                                                         |
|------------------------|---------------------------------------------------------|
| <code>data</code>      | data to plot data from                                  |
| <code>data.type</code> | define what kind of data is being used, Default: 'Mean' |
| <code>...</code>       | further arguments passed to or from other methods       |

---

 PlotMean

*Plot Mean*


---

**Description**

Create a (repeated) mean plot

**Usage**

```
PlotMean(
  data,
  monochrome = TRUE,
  plot.colors = c("#495054", "#e3e8ea"),
  font.type = "serif",
  run.repeated = FALSE,
  run.split = FALSE,
  y.split = FALSE,
  ribbon.plot = TRUE,
  y.text = "Score",
  x.text = NULL,
  remove.x = FALSE
)
```

**Arguments**

|                           |                                                                                                                       |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------|
| <code>data</code>         | MCMC data to plot                                                                                                     |
| <code>monochrome</code>   | logical, indicating whether or not to use monochrome colors, else use <a href="#">Distinct-Colors</a> , Default: TRUE |
| <code>plot.colors</code>  | range of color to use, Default: <code>c("#495054", "#e3e8ea")</code>                                                  |
| <code>font.type</code>    | font type used for visualizations, Default: 'serif'                                                                   |
| <code>run.repeated</code> | logical, indicating whether or not to use repeated measures plot, Default: FALSE                                      |
| <code>run.split</code>    | logical, indicating whether or not to use split violin plot and compare distribution between groups, Default: FALSE   |
| <code>y.split</code>      | logical, indicating whether or not to split within (TRUE) or between groups, Default: FALSE                           |
| <code>ribbon.plot</code>  | logical, indicating whether or not to use ribbon plot for HDI, Default: TRUE                                          |
| <code>y.text</code>       | label on y axis, Default: 'Score'                                                                                     |
| <code>x.text</code>       | label on x axis, Default: NULL                                                                                        |
| <code>remove.x</code>     | logical, indicating whether or not to show x.axis information, Default: FALSE                                         |

**See Also**

[ggproto](#), [ggplot2-ggproto](#), [aes](#), [margin](#), [geom\\_boxplot](#), [geom\\_crossbar](#), [geom\\_path](#), [geom\\_ribbon](#), [geom\\_violin](#), [ggplot](#), [scale\\_manual](#), [scale\\_x\\_discrete](#), [theme](#), [layer](#), [labs](#) [arrange](#), [rbind.fill](#) [zero\\_range](#) [grid.grob](#), [grobName](#), [unit](#) [approxfun](#) [colorRamp](#)



PlotNominal

*Plot Nominal***Description**

Create a nominal plot

**Usage**

```
PlotNominal(
  data,
  monochrome = TRUE,
  plot.colors = c("#CCCCCC", "#DEDEDE"),
  font.type = "serif",
  bar.dodge = 0.6,
  bar.alpha = 0.7,
  bar.width = 0.4,
  bar.extras.dodge = 0,
  bar.border = "black",
  bar.label = FALSE,
  bar.error = TRUE,
  use.cutoff = FALSE,
  diff.cutoff = 1,
  q.items = NULL
)
```

**Arguments**

|                               |                                                                                                                       |
|-------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| <code>data</code>             | MCMC data to plot                                                                                                     |
| <code>monochrome</code>       | logical, indicating whether or not to use monochrome colors, else use <a href="#">Distinct-Colors</a> , Default: TRUE |
| <code>plot.colors</code>      | range of color to use, Default: <code>c("#CCCCCC", "#DEDEDE")</code>                                                  |
| <code>font.type</code>        | font type used for visualizations, Default: 'serif'                                                                   |
| <code>bar.dodge</code>        | distance between within bar plots, Default: 0.6                                                                       |
| <code>bar.alpha</code>        | transparency for bar plot, Default: 0.7                                                                               |
| <code>bar.width</code>        | width of bar plot, Default: 0.4                                                                                       |
| <code>bar.extras.dodge</code> | dodge of error bar and label, Default: 0                                                                              |
| <code>bar.border</code>       | color of the bar border, Default: 'black'                                                                             |
| <code>bar.label</code>        | logical, indicating whether or not to show bar labels, Default: TRUE                                                  |
| <code>bar.error</code>        | logical, indicating whether or not to show error bars, Default: TRUE                                                  |
| <code>use.cutoff</code>       | logical, indicating whether or not to use a cutoff for keeping plots, Default: FALSE                                  |
| <code>diff.cutoff</code>      | if using a cutoff, determine the percentage that expected and observed values should differ, Default: 1               |
| <code>q.items</code>          | which variables should be used in the plot. Defaults to all , Default: NULL                                           |

**See Also**

[aes](#), [margin](#), [geom\\_crossbar](#), [ggplot](#), [scale\\_manual](#), [theme](#)

---

 PlotParam

*Plot Param*


---

**Description**

Create a density plot with parameter values

**Usage**

```
PlotParam(
  data,
  param,
  ROPE = FALSE,
  monochrome = TRUE,
  plot.colors = c("#495054", "#e3e8ea"),
  font.type = "serif",
  font.size = 4.5,
  rope.line = -0.2,
  rope.tick = -0.1,
  rope.label = -0.35,
  line.size = 0.5,
  dens.zero.col = "black",
  dens.mean.col = "white",
  dens.median.col = "white",
  dens.mode.col = "black",
  dens.ropes.col = "black",
  scale = FALSE,
  y.limits = NULL,
  y.breaks = NULL,
  x.limits = NULL,
  x.breaks = NULL,
  plot.title = NULL
)
```

**Arguments**

|                          |                                                                                                                       |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------|
| <code>data</code>        | MCMC data to plot                                                                                                     |
| <code>param</code>       | parameter of interest                                                                                                 |
| <code>ROPE</code>        | plot ROPE values, Default: FALSE                                                                                      |
| <code>monochrome</code>  | logical, indicating whether or not to use monochrome colors, else use <a href="#">Distinct-Colors</a> , Default: TRUE |
| <code>plot.colors</code> | range of color to use, Default: <code>c("#495054", "#e3e8ea")</code>                                                  |

|                 |                                                          |
|-----------------|----------------------------------------------------------|
| font.type       | font type used for visualizations, Default: 'serif'      |
| font.size       | font size, Default: 4.5                                  |
| rope.line       | size of ROPE lien, Default: -0.2                         |
| rope.tick       | distance to ROPE tick, Default: -0.1                     |
| rope.label      | distance to ROPE label, Default: -0.35                   |
| line.size       | overall line size, Default: 0.5                          |
| dens.zero.col   | colour of line indicating zero, Default: 'black'         |
| dens.mean.col   | colour of line indicating mean value, Default: 'white'   |
| dens.median.col | colour of line indicating median value, Default: 'white' |
| dens.mode.col   | colour of line indicating mode value, Default: 'black'   |
| dens.rope.col   | colour of line indicating ROPE value, Default: 'black'   |
| scale           | scale x and y axis, Default: FALSE                       |
| y.limits        | vector of y limits, Default: NULL                        |
| y.breaks        | vector of y breaks, Default: NULL                        |
| x.limits        | = vector of x limits, Default: NULL                      |
| x.breaks        | = vector of x breaks, Default: NULL                      |
| plot.title      | = title of plot, Default: NULL                           |

**Value**

Density plot of parameter values

**See Also**

[mutate](#), [group\\_by](#), [join](#), [select](#), [slice](#), [filter](#) [approxfun](#) [aes](#), [margin](#), [geom\\_density](#), [geom\\_polygon](#), [geom\\_segment](#), [geom](#)

---

ReadFile

*Read File*

---

**Description**

opens connection to a file

**Usage**

```
ReadFile(
  file = NULL,
  path = "models/",
  package = "bfw",
  type = "string",
  sep = ",",
  data.format = "txt",
  custom = FALSE
)
```

**Arguments**

|             |                                                                         |
|-------------|-------------------------------------------------------------------------|
| file        | name of file, Default: NULL                                             |
| path        | path to file, Default: 'models/'                                        |
| package     | choose package to open from, Default: 'bfw'                             |
| type        | Type of file (i.e., text or data), Default: 'string'                    |
| sep         | symbol to separate data (e.g., comma-delimited), Default: ','           |
| data.format | define what data format is being used, Default: 'csv'                   |
| custom      | logical, indicating whether or not to use custom file, , Default: FALSE |

**See Also**

[read.csv](#)

**Examples**

```
# Print JAGS model for bernoulli trials
cat(ReadFile("stats_bernoulli"))
# model {
#   for (i in 1:n){
#     x[i] ~ dbern(theta)
#   }
#   theta ~ dunif(0,1)
# }
```

---

RemoveEmpty

*Remove Empty*

---

**Description**

Remove empty elements in vector

**Usage**

```
RemoveEmpty(x)
```

**Arguments**

x                      vector to eliminate NA and blanks

**Examples**

```
RemoveEmpty( c("",NA,"","Remains") )
# [1] "Remains"
```

---

|               |                       |
|---------------|-----------------------|
| RemoveGarbage | <i>Remove Garbage</i> |
|---------------|-----------------------|

---

**Description**

Remove variable(s) and remove garbage from memory

**Usage**

RemoveGarbage(v)

**Arguments**

v                    variables to remove

---

|              |                      |
|--------------|----------------------|
| RemoveSpaces | <i>Remove Spaces</i> |
|--------------|----------------------|

---

**Description**

simple function to remove whitespace

**Usage**

RemoveSpaces(x)

**Arguments**

x                    string

**Examples**

```
RemoveSpaces(" No More S p a c e s")  
# [1] "NoMoreSpaces"
```

---

|              |                      |
|--------------|----------------------|
| RunContrasts | <i>Run Contrasts</i> |
|--------------|----------------------|

---

**Description**

Compute contrasts from mean and standard deviation (Cohen's d) or frequencies (odds ratio)

**Usage**

```
RunContrasts(contrast.type, q.levels, use.contrast, contrasts, data, job.names)
```

**Arguments**

|                            |                                                                                                                                                                            |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>contrast.type</code> | type of contrast: "m" indicate means and standard deviations, "o" indicate frequency                                                                                       |
| <code>q.levels</code>      | Number of levels of each variable/column                                                                                                                                   |
| <code>use.contrast</code>  | choose from "between", "within" and "mixed". Between compare groups at different conditions. Within compare a group at different conditions. Mixed compute all comparisons |
| <code>contrasts</code>     | specified contrasts columns                                                                                                                                                |
| <code>data</code>          | data to compute contrasts from                                                                                                                                             |
| <code>job.names</code>     | names of all parameters in analysis, Default: NULL                                                                                                                         |

**See Also**

[combn](#)

---

|         |                 |
|---------|-----------------|
| RunMCMC | <i>Run MCMC</i> |
|---------|-----------------|

---

**Description**

Conduct MCMC simulations using JAGS

**Usage**

```
RunMCMC(
  jags.model,
  params = NULL,
  name.list,
  data.list,
  initial.list = list(),
  run.contrasts = FALSE,
  use.contrast = "between",
```

```

contrasts = NULL,
custom.contrast = NULL,
run.ppp = FALSE,
k.ppp = 10,
n.data,
credible.region = 0.95,
save.data = FALSE,
ROPE = NULL,
merge.MCMC = FALSE,
run.diag = FALSE,
param.diag = NULL,
sep = ",",
monochrome = TRUE,
plot.colors = c("#495054", "#e3e8ea"),
graphic.type = "pdf",
plot.size = "15,10",
scaling = 100,
plot.aspect = NULL,
vector.graphic = FALSE,
point.size = 12,
font.type = "serif",
one.file = TRUE,
ppi = 300,
units = "in",
layout = "a4",
layout.inverse = FALSE,
...
)

```

### Arguments

|                 |                                                                                                                                                                                                 |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| jags.model      | specify which module to use                                                                                                                                                                     |
| params          | define parameters to observe, Default: NULL                                                                                                                                                     |
| name.list       | list of names                                                                                                                                                                                   |
| data.list       | list of data                                                                                                                                                                                    |
| initial.list    | initial values for analysis, Default: list()                                                                                                                                                    |
| run.contrasts   | logical, indicating whether or not to run contrasts, Default: FALSE                                                                                                                             |
| use.contrast    | choose from "between", "within" and "mixed". Between compare groups at different conditions. Within compare a group at different conditions. Mixed compute all comparisons, Default: "between", |
| contrasts       | define contrasts to use for analysis (defaults to all) , Default: NULL                                                                                                                          |
| custom.contrast | define contrasts for custom models , Default: NULL                                                                                                                                              |
| run.ppp         | logical, indicating whether or not to conduct ppp analysis, Default: FALSE                                                                                                                      |
| k.ppp           | run ppp for every kth length of MCMC chains, Default: 10                                                                                                                                        |
| n.data          | sample size for each parameter                                                                                                                                                                  |

|                              |                                                                                                                          |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| <code>credible.region</code> | summarize uncertainty by defining a region of most credible values (e.g., 95 percent of the distribution), Default: 0.95 |
| <code>save.data</code>       | logical, indicating whether or not to save data, Default: FALSE                                                          |
| <code>ROPE</code>            | define range for region of practical equivalence (e.g., <code>c(-0.05 , 0.05)</code> ), Default: NULL                    |
| <code>merge.MCMC</code>      | logical, indicating whether or not to merge MCMC chains, Default: FALSE                                                  |
| <code>run.diag</code>        | logical, indicating whether or not to run diagnostics, Default: FALSE                                                    |
| <code>param.diag</code>      | define parameters to use for diagnostics, default equals all parameters, Default: NULL                                   |
| <code>sep</code>             | symbol to separate data (e.g., comma-delimited), Default: <code>' '</code>                                               |
| <code>monochrome</code>      | logical, indicating whether or not to use monochrome colors, else use <a href="#">Distinct-Colors</a> , Default: TRUE    |
| <code>plot.colors</code>     | range of color to use, Default: <code>c("#495054", "#e3e8ea")</code>                                                     |
| <code>graphic.type</code>    | type of graphics to use (e.g., pdf, png, ps), Default: <code>'pdf'</code>                                                |
| <code>plot.size</code>       | size of plot, Default: <code>'15,10'</code>                                                                              |
| <code>scaling</code>         | scale size of plot, Default: 100                                                                                         |
| <code>plot.aspect</code>     | aspect of plot, Default: NULL                                                                                            |
| <code>vector.graphic</code>  | logical, indicating whether or not visualizations should be vector or raster graphics, Default: FALSE                    |
| <code>point.size</code>      | point size used for visualizations, Default: 12                                                                          |
| <code>font.type</code>       | font type used for visualizations, Default: <code>'serif'</code>                                                         |
| <code>one.file</code>        | logical, indicating whether or not visualizations should be placed in one or several files, Default: TRUE                |
| <code>ppi</code>             | define pixel per inch used for visualizations, Default: 300                                                              |
| <code>units</code>           | define unit of length used for visualizations, Default: <code>'in'</code>                                                |
| <code>layout</code>          | define a layout size for visualizations, Default: <code>'a4'</code>                                                      |
| <code>layout.inverse</code>  | logical, indicating whether or not to inverse layout (e.g., landscape) , Default: FALSE                                  |
| <code>...</code>             | further arguments passed to or from other methods                                                                        |

**Value**

list containing MCMC chains , MCMC chains as matrix , summary of MCMC, list of name used, list of data, the jags model, running time of analysis and names of saved files

**See Also**

[runjags.options](#), [run.jags.detectCores](#) as `mcmc.list`, `varnames` `rbind.fill` `cor`, `cov`, `sd` `mvrnorm` `write.table`



---

|              |                      |
|--------------|----------------------|
| SingleString | <i>Single String</i> |
|--------------|----------------------|

---

**Description**

determine whether input is a single string

**Usage**

```
SingleString(x)
```

**Arguments**

|   |        |
|---|--------|
| x | string |
|---|--------|

**Value**

true or false

**Examples**

```
A <- "This is a single string"
SingleString(A)
# [1] TRUE
is.character(A)
# [1] TRUE
B <- c("This is a vector" , "containing two strings")
SingleString(B)
# [1] FALSE
is.character(B)
# [1] TRUE
```

---

|                |                         |
|----------------|-------------------------|
| StatsBernoulli | <i>Bernoulli Trials</i> |
|----------------|-------------------------|

---

**Description**

Conduct bernoulli trials

**Usage**

```
StatsBernoulli(
  x = NULL,
  x.names = NULL,
  DF,
  params = NULL,
  initial.list = list(),
  ...
)
```

**Arguments**

**x** predictor variable(s), Default: NULL  
**x.names** optional names for predictor variable(s), Default: NULL  
**DF** data for analysis  
**params** define parameters to observe, Default: NULL  
**initial.list** initial values for analysis, Default: list()  
**...** further arguments passed to or from other methods

**See Also**

[complete.cases](#)

**Examples**

```

## Create coin toss data: heads = 50 and tails = 50
#fair.coin<- as.matrix(c(rep("Heads",50),rep("Tails",50)))
#colnames(fair.coin) <- "X"
#fair.coin <- bfw(project.data = fair.coin,
#                 x = "X",
#                 saved.steps = 50000,
#                 jags.model = "bernoulli",
#                 jags.seed = 100,
#                 ROPE = c(0.4,0.6),
#                 silent = TRUE)
#fair.coin.freq <- binom.test( 50000 * 0.5, 50000)

## Create coin toss data: heads = 20 and tails = 80
#biased.coin <- as.matrix(c(rep("Heads",20),rep("Tails",80)))
#colnames(biased.coin) <- "X"
#biased.coin <- bfw(project.data = biased.coin,
#                   x = "X",
#                   saved.steps = 50000,
#                   jags.model = "bernoulli",
#                   jags.seed = 101,
#                   initial.list = list(theta = 0.7),
#                   ROPE = c(0.4,0.6),
#                   silent = TRUE)
#biased.coin.freq <- binom.test( 50000 * 0.8, 50000)

## Print Bayesian and frequentist results of fair coin
#fair.coin$summary.MCMC[,c(3:6,9:12)]

## Mode      ESS      HDIlo    HDIhi    ROPElo    ROPEhi    ROPEin    n
## 0.505 50480.000    0.405    0.597    2.070    2.044    95.886   100.00

#sprintf("Frequentist: %.3f [%%.3f , %%.3f], p = %%.3f" ,
#        fair.coin.freq$estimate ,
#        fair.coin.freq$conf.int[1] ,
#        fair.coin.freq$conf.int[2] ,
#        fair.coin.freq$p.value)

```

```
## [1] "Frequentist: 0.500 [0.496 , 0.504], p = 1.000"

## Print Bayesian and frequentist results of biased coin
#biased.coin$summary.MCMC[,c(3:6,9:12)]

## Mode      ESS      HDIlo      HDIhi      ROPElo      ROPEhi      ROPEin      n
## 0.803 50000.000      0.715      0.870      0.000      99.996      0.004      100.000

#sprintf("Frequentist: %.3f [%.3f , %.3f], p = %.3f" ,
#        biased.coin.freq$estimate ,
#        biased.coin.freq$conf.int[1] ,
#        biased.coin.freq$conf.int[2] ,
#        biased.coin.freq$p.value)

## [1] "Frequentist: 0.800 [0.796 , 0.803], p = 0.000"
```

---

StatsCovariate

*Covariate*


---

## Description

Covariate estimations (including correlation and Cronbach's alpha)

## Usage

```
StatsCovariate(
  y = NULL,
  y.names = NULL,
  x = NULL,
  x.names = NULL,
  DF,
  params = NULL,
  job.group = NULL,
  initial.list = list(),
  jags.model,
  ...
)
```

## Arguments

|                      |                                                         |
|----------------------|---------------------------------------------------------|
| <code>y</code>       | criterion variable(s), Default: NULL                    |
| <code>y.names</code> | optional names for criterion variable(s), Default: NULL |
| <code>x</code>       | predictor variable(s), Default: NULL                    |
| <code>x.names</code> | optional names for predictor variable(s), Default: NULL |
| <code>DF</code>      | data to analyze                                         |
| <code>params</code>  | define parameters to observe, Default: NULL             |

|                           |                                                                                                                           |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>job.group</code>    | for some hierarchical models with several layers of parameter names (e.g., latent and observed parameters), Default: NULL |
| <code>initial.list</code> | initial values for analysis, Default: list()                                                                              |
| <code>jags.model</code>   | specify which module to use                                                                                               |
| <code>...</code>          | further arguments passed to or from other methods                                                                         |

**Value**

covariate, correlation and (optional) Cronbach's alpha

**See Also**

[complete.cases](#)

**Examples**

```
## Create normal distributed data with mean = 0 and standard deviation = 1
### r = 0.5
#data <- MASS::mvrnorm(n=100,
#                       mu=c(0, 0),
#                       Sigma=matrix(c(1, 0.5, 0.5, 1), 2),
#                       empirical=TRUE)
## Add names
#colnames(data) <- c("X","Y")
## Create noise with mean = 10 / -10 and sd = 1
### r = -1.0
#noise <- MASS::mvrnorm(n=2,
#                       mu=c(10, -10),
#                       Sigma=matrix(c(1, -1, -1, 1), 2),
#                       empirical=TRUE)
## Combine noise and data
#biased.data <- rbind(data,noise)
#
#
## Run analysis on normal distributed data
#mcmc <- bfw(project.data = data,
#            y = "X,Y",
#            saved.steps = 50000,
#            jags.model = "covariate",
#            jags.seed = 100,
#            silent = TRUE)
## Run robust analysis on normal distributed data
#mcmc.robust <- bfw(project.data = data,
#                  y = "X,Y",
#                  saved.steps = 50000,
#                  jags.model = "covariate",
#                  run.robust = TRUE,
#                  jags.seed = 101,
#                  silent = TRUE)
## Run analysis on data with outliers
#biased.mcmc <- bfw(project.data = biased.data,
```

```

#           y = "X,Y",
#           saved.steps = 50000,
#           jags.model = "covariate",
#           jags.seed = 102,
#           silent = TRUE)
## Run robust analysis on data with outliers
#biased.mcmc.robust <- bfw(project.data = biased.data,
#           y = "X,Y",
#           saved.steps = 50000,
#           jags.model = "covariate",
#           run.robust = TRUE,
#           jags.seed = 103,
#           silent = TRUE)
## Print frequentist results
#stats::cor(data)[2]
## [1] 0.5
#stats::cor(noise)[2]
## [1] -1
#stats::cor(biased.data)[2]
## [1] -0.498
## Print Bayesian results
#mcmc$summary.MCMC
##           Mean Median Mode   ESS HDIlo HDIhi   n
## cor[1,1]: X vs. X 1.000 1.000 0.999    0 1.000 1.000 100
## cor[2,1]: Y vs. X 0.488 0.491 0.496 19411 0.337 0.633 100
## cor[1,2]: X vs. Y 0.488 0.491 0.496 19411 0.337 0.633 100
## cor[2,2]: Y vs. Y 1.000 1.000 0.999    0 1.000 1.000 100
#mcmc.robust$summary.MCMC
##           Mean Median Mode   ESS HDIlo HDIhi   n
## cor[1,1]: X vs. X 1.00 1.000 0.999    0 1.000 1.000 100
## cor[2,1]: Y vs. X 0.47 0.474 0.491 18626 0.311 0.626 100
## cor[1,2]: X vs. Y 0.47 0.474 0.491 18626 0.311 0.626 100
## cor[2,2]: Y vs. Y 1.00 1.000 0.999    0 1.000 1.000 100
#biased.mcmc$summary.MCMC
##           Mean Median Mode   ESS HDIlo HDIhi   n
## cor[1,1]: X vs. X 1.000 1.000 0.999    0 1.000 1.000 102
## cor[2,1]: Y vs. X -0.486 -0.489 -0.505 19340 -0.627 -0.335 102
## cor[1,2]: X vs. Y -0.486 -0.489 -0.505 19340 -0.627 -0.335 102
## cor[2,2]: Y vs. Y 1.000 1.000 0.999    0 1.000 1.000 102
#biased.mcmc.robust$summary.MCMC
##           Mean Median Mode   ESS HDIlo HDIhi   n
## cor[1,1]: X vs. X 1.000 1.000 0.999    0 1.000 1.000 102
## cor[2,1]: Y vs. X 0.338 0.343 0.356 23450 0.125 0.538 102
## cor[1,2]: X vs. Y 0.338 0.343 0.356 23450 0.125 0.538 102

```

**Description**

Apply latent or observed models to fit data (e.g., SEM, CFA, mediation)

**Usage**

```

StatsFit(
  latent = NULL,
  latent.names = NULL,
  observed = NULL,
  observed.names = NULL,
  additional = NULL,
  additional.names = NULL,
  DF,
  params = NULL,
  job.group = NULL,
  initial.list = list(),
  model.name,
  jags.model,
  custom.model = NULL,
  run.ppp = FALSE,
  run.robust = FALSE,
  ...
)

```

**Arguments**

|                  |                                                                                                                           |
|------------------|---------------------------------------------------------------------------------------------------------------------------|
| latent           | latent variables, Default: NULL                                                                                           |
| latent.names     | optional names for for latent variables, Default: NULL                                                                    |
| observed         | observed variable(s), Default: NULL                                                                                       |
| observed.names   | optional names for for observed variable(s), Default: NULL                                                                |
| additional       | supplemental parameters for fitted data (e.g., indirect pathways and total effect), Default: NULL                         |
| additional.names | optional names for supplemental parameters, Default: NULL                                                                 |
| DF               | data to analyze                                                                                                           |
| params           | define parameters to observe, Default: NULL                                                                               |
| job.group        | for some hierarchical models with several layers of parameter names (e.g., latent and observed parameters), Default: NULL |
| initial.list     | initial values for analysis, Default: list()                                                                              |
| model.name       | name of model used                                                                                                        |
| jags.model       | specify which module to use                                                                                               |
| custom.model     | define a custom model to use (e.g., string or text file (.txt), Default: NULL                                             |
| run.ppp          | logical, indicating whether or not to conduct ppp analysis, Default: FALSE                                                |
| run.robust       | logical, indicating whether or not robust analysis, Default: FALSE                                                        |
| ...              | further arguments passed to or from other methods                                                                         |

**See Also**

[complete.cases](#)

---

StatsKappa

*Cohen's Kappa*

---

## Description

Bayesian alternative to Cohen's kappa

## Usage

```
StatsKappa(  
  x = NULL,  
  x.names = NULL,  
  DF,  
  params = NULL,  
  initial.list = list(),  
  ...  
)
```

## Arguments

|              |                                                         |
|--------------|---------------------------------------------------------|
| x            | predictor variable(s), Default: NULL                    |
| x.names      | optional names for predictor variable(s), Default: NULL |
| DF           | data to analyze                                         |
| params       | define parameters to observe, Default: NULL             |
| initial.list | initial values for analysis, Default: list()            |
| ...          | further arguments passed to or from other methods       |

## See Also

[complete.cases](#)

## Examples

```
## Simulate rater data  
#Rater1 <- c(rep(0,20),rep(1,80))  
#set.seed(100)  
#Rater2 <- c(rbinom(20,1,0.1), rbinom(80,1,0.9))  
#data <- data.frame(Rater1,Rater2)  
  
#mcmc <- bfw(project.data = data,  
#           x = "Rater1,Rater2",  
#           saved.steps = 50000,  
#           jags.model = "kappa",  
#           jags.seed = 100,  
#           silent = TRUE)  
  
## Print frequentist and Bayesian kappa
```

```

#library(psych)
#psych::cohen.kappa(data)$confid[1,]
## lower estimate upper
## 0.6137906 0.7593583 0.9049260
##' mcmc$summary.MCMC
##      Mean      Median      Mode      ESS  HDIlo  HDIhi  n
## Kappa[1]: 0.739176 0.7472905 0.7634503 50657 0.578132 0.886647 100

```

---

StatsMean

*Mean Data*


---

## Description

Compute means and standard deviations.

## Usage

```

StatsMean(
  y = NULL,
  y.names = NULL,
  x = NULL,
  x.names = NULL,
  DF,
  params = NULL,
  initial.list = list(),
  ...
)

```

## Arguments

|                           |                                                           |
|---------------------------|-----------------------------------------------------------|
| <code>y</code>            | criterion variable(s), Default: NULL                      |
| <code>y.names</code>      | optional names for criterion variable(s), Default: NULL   |
| <code>x</code>            | categorical variable(s), Default: NULL                    |
| <code>x.names</code>      | optional names for categorical variable(s), Default: NULL |
| <code>DF</code>           | User defined data frame, Default: NULL                    |
| <code>params</code>       | define parameters to observe, Default: NULL               |
| <code>initial.list</code> | Initial values for simulations, Default: list()           |
| <code>...</code>          | further arguments passed to or from other methods         |

## Value

mean and standard deviation



StatsMetric

*Predict Metric***Description**

Bayesian alternative to ANOVA

**Usage**

```
StatsMetric(
  y = NULL,
  y.names = NULL,
  x = NULL,
  x.names = NULL,
  DF,
  params = NULL,
  job.group = NULL,
  initial.list = list(),
  model.name,
  jags.model,
  custom.model = NULL,
  run.robust = FALSE,
  ...
)
```

**Arguments**

|                           |                                                                                                                           |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>y</code>            | criterion variable(s), Default: NULL                                                                                      |
| <code>y.names</code>      | optional names for criterion variable(s), Default: NULL                                                                   |
| <code>x</code>            | categorical variable(s), Default: NULL                                                                                    |
| <code>x.names</code>      | optional names for categorical variable(s), Default: NULL                                                                 |
| <code>DF</code>           | data to analyze                                                                                                           |
| <code>params</code>       | define parameters to observe, Default: NULL                                                                               |
| <code>job.group</code>    | for some hierarchical models with several layers of parameter names (e.g., latent and observed parameters), Default: NULL |
| <code>initial.list</code> | initial values for analysis, Default: list()                                                                              |
| <code>model.name</code>   | name of model used                                                                                                        |
| <code>jags.model</code>   | specify which module to use                                                                                               |
| <code>custom.model</code> | define a custom model to use (e.g., string or text file (.txt), Default: NULL                                             |
| <code>run.robust</code>   | logical, indicating whether or not robust analysis, Default: FALSE                                                        |
| <code>...</code>          | further arguments passed to or from other methods                                                                         |

**See Also**

[complete.cases](#), [sd](#), [aggregate](#), [median head](#)

StatsNominal

*Predict Nominal***Description**

Bayesian alternative to chi-square test

**Usage**

```
StatsNominal(
  x = NULL,
  x.names = NULL,
  DF,
  params = NULL,
  job.group = NULL,
  initial.list = list(),
  model.name,
  jags.model,
  custom.model = NULL,
  ...
)
```

**Arguments**

|              |                                                                                                                           |
|--------------|---------------------------------------------------------------------------------------------------------------------------|
| x            | categorical variable(s), Default: NULL                                                                                    |
| x.names      | optional names for categorical variable(s), Default: NULL                                                                 |
| DF           | data to analyze                                                                                                           |
| params       | define parameters to observe, Default: NULL                                                                               |
| job.group    | for some hierarchical models with several layers of parameter names (e.g., latent and observed parameters), Default: NULL |
| initial.list | initial values for analysis, Default: list()                                                                              |
| model.name   | name of model used                                                                                                        |
| jags.model   | specify which module to use                                                                                               |
| custom.model | define a custom model to use (e.g., string or text file (.txt), Default: NULL                                             |
| ...          | further arguments passed to or from other methods                                                                         |

**Examples**

```
## Use cats data
# mcmc <- bfw(project.data = bfw::Cats,
#           x = "Reward,Dance,Alignment",
#           saved.steps = 50000,
#           jags.model = "nominal",
#           run.contrasts = TRUE,
#           jags.seed = 100)
```

```
## Print only odds-ratio and effect sizes
#   mcmc$summary.MCMC[ grep("Odds ratio|Effect",
#                             rownames(mcmc$summary.MCMC)) , c(3:7) ]
##
##                                     Mode  ESS   HDIlo   HDIhi   n
## Effect size: Affection/Food vs. Evil/Good      0.12844 45222  0.00115  0.25510 2000
## Effect size: Affection/Food vs. No/Yes         1.05346 44304  0.90825  1.18519 2000
## Effect size: Affection/Food vs. No/Yes @ Evil  2.58578 30734  2.35471  2.85450 1299
## Effect size: Affection/Food vs. No/Yes @ Good -0.51934 35316 -0.73443 -0.30726  701
## Effect size: Food/Affection vs. Evil/Good     -0.12844 45222 -0.25510 -0.00115 2000
## Effect size: Food/Affection vs. No/Yes        -1.05346 44304 -1.18519 -0.90825 2000
## Effect size: Food/Affection vs. No/Yes @ Evil -2.58578 30734 -2.85450 -2.35471 1299
## Effect size: Food/Affection vs. No/Yes @ Good  0.51934 35316  0.30726  0.73443  701
## Effect size: No/Yes vs. Evil/Good             1.43361 43603  1.30715  1.55020 2000
## Effect size: Yes/No vs. Evil/Good            -1.43361 43603 -1.55020 -1.30715 2000
## Odds ratio: Affection/Food vs. Evil/Good      1.25432 45225  0.99311  1.57765 2000
## Odds ratio: Affection/Food vs. No/Yes         6.49442 44215  5.10392  8.46668 2000
## Odds ratio: Affection/Food vs. No/Yes @ Evil 104.20109 30523 66.55346 169.12331 1299
## Odds ratio: Affection/Food vs. No/Yes @ Good  0.36685 35417  0.25478  0.55982  701
## Odds ratio: Food/Affection vs. Evil/Good      0.77604 45245  0.62328  0.98904 2000
## Odds ratio: Food/Affection vs. No/Yes         0.14586 44452  0.11426  0.18982 2000
## Odds ratio: Food/Affection vs. No/Yes @ Evil  0.00848 31117  0.00527  0.01336 1299
## Odds ratio: Food/Affection vs. No/Yes @ Good  2.44193 35397  1.65204  3.63743  701
## Odds ratio: No/Yes vs. Evil/Good             13.12995 43500 10.58859 16.49207 2000
## Odds ratio: Yes/No vs. Evil/Good             0.07393 43739  0.05909  0.09221 2000
#
## The results indicate that evil cats are 13.13 times more likely than good cats to decline dancing
## Furthermore, when offered affection, evil cats are 104.20 times more likely to decline dancing,
## relative to evil cats that are offered food.
```

---

StatsRegression

*Regression*


---

## Description

Simple, multiple and hierarchical regression

## Usage

```
StatsRegression(
  y = NULL,
  y.names = NULL,
  x = NULL,
  x.names = NULL,
  x.steps = NULL,
  x.blocks = NULL,
  DF,
  params = NULL,
  job.group = NULL,
  initial.list = list(),
```

```
    ...
  )
```

### Arguments

|                           |                                                                                                                           |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>y</code>            | criterion variable(s), Default: NULL                                                                                      |
| <code>y.names</code>      | optional names for criterion variable(s), Default: NULL                                                                   |
| <code>x</code>            | predictor variable(s), Default: NULL                                                                                      |
| <code>x.names</code>      | optional names for predictor variable(s), Default: NULL                                                                   |
| <code>x.steps</code>      | define number of steps in hierarchical regression , Default: NULL                                                         |
| <code>x.blocks</code>     | define which predictors are included in each step (e.g., for three steps "1,2,3") , Default: NULL                         |
| <code>DF</code>           | data to analyze                                                                                                           |
| <code>params</code>       | define parameters to observe, Default: NULL                                                                               |
| <code>job.group</code>    | for some hierarchical models with several layers of parameter names (e.g., latent and observed parameters), Default: NULL |
| <code>initial.list</code> | initial values for analysis, Default: list()                                                                              |
| <code>...</code>          | further arguments passed to or from other methods                                                                         |

### See Also

[complete.cases](#)

---

StatsSoftmax

*Softmax Regression*

---

### Description

Perform softmax regression (i.e., multinomial logistic regression)

### Usage

```
StatsSoftmax(
  y = NULL,
  y.names = NULL,
  x = NULL,
  x.names = NULL,
  DF,
  params = NULL,
  job.group = NULL,
  initial.list = NULL,
  run.robust = FALSE,
  ...
)
```

**Arguments**

|              |                                                                                                                           |
|--------------|---------------------------------------------------------------------------------------------------------------------------|
| y            | criterion variable(s), Default: NULL                                                                                      |
| y.names      | optional names for criterion variable(s), Default: NULL                                                                   |
| x            | predictor variable(s), Default: NULL                                                                                      |
| x.names      | optional names for predictor variable(s), Default: NULL                                                                   |
| DF           | data to analyze                                                                                                           |
| params       | define parameters to observe, Default: NULL                                                                               |
| job.group    | for some hierarchical models with several layers of parameter names (e.g., latent and observed parameters), Default: NULL |
| initial.list | initial values for analysis, Default: list()                                                                              |
| run.robust   | logical, indicating whether or not robust analysis, Default: FALSE                                                        |
| ...          | further arguments passed to or from other methods                                                                         |

**See Also**

[complete.cases](#)

**Examples**

```
## Conduct softmax regression on Cats data
### Reward is 0 = Food and 1 = Dance
### Sample 100 datapoints from Cats data
#mcmc <- bfw(project.data = bfw::Cats,
#           y = "Alignment",
#           x = "Ratings,Reward",
#           saved.steps = 50000,
#           jags.model = "softmax",
#           jags.seed = 100)
## Conduct binominal generalized linear model
#model <- glm(Alignment ~ Ratings + Reward, data=bfw::Cats, family = binomial(link="logit"))
## Print output from softmax
#mcmc$summary.MCMC
#
##
##beta[1,1]: Evil vs. Ratings  0.000  0.00 -0.000607  0  0.000  0.000  2000
##beta[1,2]: Evil vs. Reward  0.000  0.00 -0.000607  0  0.000  0.000  2000
##beta[2,1]: Good vs. Ratings  1.289  1.29  1.283403 19614  1.187  1.387  2000
##beta[2,2]: Good vs. Reward  1.276  1.27  1.279209 20807  0.961  1.597  2000
##beta0[1]: Intercept: Evil    0.000  0.00 -0.000607  0  0.000  0.000  2000
##beta0[2]: Intercept: Good   -7.690 -7.68 -7.659198 17693 -8.472 -6.918 2000
##zbeta[1,1]: Evil vs. Ratings  0.000  0.00 -0.000607  0  0.000  0.000  2000
##zbeta[1,2]: Evil vs. Reward  0.000  0.00 -0.000607  0  0.000  0.000  2000
##zbeta[2,1]: Good vs. Ratings  2.476  2.47  2.464586 19614  2.280  2.664  2000
##zbeta[2,2]: Good vs. Reward  0.501  0.50  0.501960 20807  0.377  0.626  2000
##zbeta0[1]: Intercept: Evil    0.000  0.00 -0.000607  0  0.000  0.000  2000
##zbeta0[2]: Intercept: Good   -1.031 -1.03 -1.024178 22812 -1.185 -0.870 2000
#
## Print (truncated) output from GML
```

| ##                | Estimate | Std. Error | z value | Pr(> z )    |
|-------------------|----------|------------|---------|-------------|
| ##(Intercept)     | -6.39328 | 0.27255    | -23.457 | < 2e-16 *** |
| ##Ratings         | 1.28480  | 0.05136    | 25.014  | < 2e-16 *** |
| ##RewardAffection | 1.26975  | 0.16381    | 7.751   | 9.1e-15 *** |

SumMCMC

*Summarize MCMC***Description**

The function provide a summary of each parameter of interest (mean, median, mode, effective sample size (ESS), HDI and n)

**Usage**

```
SumMCMC(
  par,
  par.names,
  job.names = NULL,
  job.group = NULL,
  credible.region = 0.95,
  ROPE = NULL,
  n.data,
  ...
)
```

**Arguments**

|                 |                                                                                                                           |
|-----------------|---------------------------------------------------------------------------------------------------------------------------|
| par             | defined parameter                                                                                                         |
| par.names       | parameter names                                                                                                           |
| job.names       | names of all parameters in analysis, Default: NULL                                                                        |
| job.group       | for some hierarchical models with several layers of parameter names (e.g., latent and observed parameters), Default: NULL |
| credible.region | summarize uncertainty by defining a region of most credible values (e.g., 95 percent of the distribution), Default: 0.95  |
| ROPE            | define range for region of practical equivalence (e.g., c(-0.05 , 0.05), Default: NULL                                    |
| n.data          | sample size for each parameter                                                                                            |
| ...             | further arguments passed to or from other methods                                                                         |

**See Also**

[effectiveSize](#)

---

SumToZero

*Sum to Zero*


---

**Description**

Compute sum to zero values across all levels of a data matrix

**Usage**

```
SumToZero(q.levels, data, contrasts)
```

**Arguments**

|           |                                          |
|-----------|------------------------------------------|
| q.levels  | number of levels of each variable/column |
| data      | data matrix to combine from              |
| contrasts | specified contrasts columns              |

**Examples**

```
data <- matrix(c(1,2),ncol=2)
colnames(data) <- c("m1[1]","m1[2]")
SumToZero( 2 , data , contrasts = NULL )
#           b0[1] b1[1] b1[2]
# m1[1]  1.5 -0.5  0.5
```

---

TidyCode

*Tidy Code*


---

**Description**

Small function that clears up messy code

**Usage**

```
TidyCode(tidy.code, jags = TRUE)
```

**Arguments**

|           |                                                        |
|-----------|--------------------------------------------------------|
| tidy.code | Messy code that needs cleaning                         |
| jags      | logical, if TRUE run code as JAGS model, Default: TRUE |

**Value**

(Somewhat) tidy code

**Examples**

```

messy <- "code <- function( x ) {
print (x ) }"
cat(messy)
code <- function( x ) {
print (x ) }
cat ( TidyCode(messy, jags = FALSE) )
code <- function(x) {
  print(x)
}

```

---

Trim

*Trim*


---

**Description**

remove excess whitespace from string

**Usage**

```
Trim(s, multi = TRUE)
```

**Arguments**

|       |                                                                                                  |
|-------|--------------------------------------------------------------------------------------------------|
| s     | string                                                                                           |
| multi | logical, indicating whether or not to remove excess whitespace between characters, Default: TRUE |

**Examples**

```

Trim("      Trimmed      string")
# [1] "Trimmed string"
Trim("      Trimmed      string", FALSE)
# [1] "Trimmed      string"

```

---

TrimSplit

*Trim Split*


---

**Description**

Extends strsplit by trimming and unlisting string



**Usage**

```
TrimSplit(
  x,
  sep = ",",
  fixed = FALSE,
  perl = FALSE,
  useBytes = FALSE,
  rm.empty = TRUE
)
```

**Arguments**

|          |                                                                                                                 |
|----------|-----------------------------------------------------------------------------------------------------------------|
| x        | string                                                                                                          |
| sep      | symbol to separate data (e.g., comma-delimited), Default: ','                                                   |
| fixed    | logical, if TRUE match split exactly, otherwise use regular expressions. Has priority over perl, Default: FALSE |
| perl     | logical, indicating whether or not to use Perl-compatible regexps, Default: FALSE                               |
| useBytes | logical, if TRUE the matching is done byte-by-byte rather than character-by-character, Default: FALSE           |
| rm.empty | logical. indicating whether or not to remove empty elements, Default: TRUE                                      |

**Details**

[strsplit](#)

**Examples**

```
TrimSplit("Data 1, Data2, Data3")
# [1] "Data 1" "Data2" "Data3"
```

---

VectorSub

*Pattern Matching and Replacement From Vectors*

---

**Description**

extending gsub by matching pattern and replacement from two vectors

**Usage**

```
VectorSub(pattern, replacement, string)
```

**Arguments**

|             |                                                    |
|-------------|----------------------------------------------------|
| pattern     | vector containing words to match                   |
| replacement | vector containing words to replace existing words. |
| string      | string to replace from                             |

**Value**

modified string with replaced values

**Examples**

```
pattern <- c("A","B","C")
replacement <- 1:3
string <- "A went to B went to C"
VectorSub(pattern,replacement,string)
# [1] "1 went to 2 went to 3"
```

# Index

## \* datasets

Cats, 7

abline, 10

acf, 10

add\_slide, 22

AddNames, 3

aes, 24, 26, 27

aggregate, 41

approxfun, 24, 27

arrange, 24

as.mcmc.list, 32

Beta, 16

bfw, 4

capture.output, 6

CapWords, 6

Cats, 7

ChangeNames, 7

chordDiagram, 23

circos.clear, 23

circos.par, 23

circos.trackPlotRegion, 23

colorRamp, 24

colorRampPalette, 10

combine.mcmc, 18

combn, 30

complete.cases, 34, 36, 38, 39, 41, 44, 45

ComputeHDI, 8

ContrastNames, 9

cor, 32

cov, 32

density, 10

detectCores, 32

dev, 22, 23

dev.list, 10

dev.new, 10

dev.off, 10

DiagMCMC, 9

DistinctColors, 10, 10, 23–26, 32

dml, 22

effectiveSize, 10, 46

ETA, 11

FileName, 11

filter, 27

FindEnvironment, 12

FlattenList, 13

flush.console, 11

GammaDist, 14

gelman.plot, 10

geom\_boxplot, 24

geom\_crossbar, 24, 26

geom\_density, 27

geom\_label, 27

geom\_path, 24

geom\_polygon, 27

geom\_ribbon, 24

geom\_segment, 27

geom\_violin, 24

GetRange, 14

ggplot, 24, 26, 27

ggplot\_build, 27

ggproto, 24

graphics.off, 10

grid.grob, 24

grobName, 24

group\_by, 27

head, 6, 18, 22, 41

Interleave, 15

InverseHDI, 16

join, 27

labs, 24, 27

layer, 24  
Layout, 17  
layout, 10  
legend, 23  
  
margin, 24, 26, 27  
matplot, 10  
MatrixCombn, 17  
median, 41  
MergeMCMC, 18  
modifyList, 6  
mtext, 10  
MultiGrep, 19  
mutate, 27  
mvrnorm, 32  
  
na.omit, 20  
Normalize, 19  
  
optimize, 16  
  
PadVector, 20  
par, 10, 22  
ParseNumber, 20  
ParsePlot, 21  
ph\_with, 22  
plot, 22  
plot.new, 10  
PlotCirclize, 22  
PlotData, 23  
PlotMean, 24  
PlotNominal, 25  
PlotParam, 26  
png, 22  
points, 10  
ps.options, 22  
  
rasterImage, 22  
rbind.fill, 24, 32  
read.csv, 28  
read\_pptx, 22  
ReadFile, 27  
readPNG, 22  
recordPlot, 10, 22, 23  
RemoveEmpty, 28  
RemoveGarbage, 29  
RemoveSpaces, 29  
run.jags, 32  
RunContrasts, 30  
  
runjags.options, 32  
RunMCMC, 6, 30  
  
scale\_continuous, 27  
scale\_manual, 24, 26  
scale\_x\_discrete, 24  
sd, 10, 32, 41  
select, 27  
SingleString, 33  
slice, 27  
StatsBernoulli, 33  
StatsCovariate, 35  
StatsFit, 37  
StatsKappa, 39  
StatsMean, 40  
StatsMetric, 41  
StatsNominal, 42  
StatsRegression, 43  
StatsSoftmax, 44  
strsplit, 49  
SumMCMC, 46  
SumToZero, 47  
  
text, 10  
theme, 24, 26, 27  
TidyCode, 47  
traceplot, 10  
Trim, 48  
TrimSplit, 48  
  
unit, 24  
  
varnames, 32  
VectorSub, 49  
  
write.table, 32  
  
zero\_range, 24