

# Package ‘vipor’

May 8, 2026

**Type** Package

**Title** Plot Categorical Data Using Quasirandom Noise and Density Estimates

**Version** 0.4.7

**Date** 2023-12-15

**Author** Scott Sherrill-Mix, Erik Clarke

**Maintainer** Scott Sherrill-Mix <ssm@msu.edu>

**Description** Generate a violin point plot, a combination of a violin/histogram plot and a scatter plot by offsetting points within a category based on their density using quasirandom noise.

**License** GPL (>= 2)

**LazyData** True

**Depends** R (>= 3.5.0)

**Imports** stats, graphics

**Suggests** testthat, beeswarm, lattice, ggplot2, beanplot, vioplot, ggbeeswarm,

**RoxygenNote** 7.2.3

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2023-12-18 14:30:02 UTC

## Contents

aveWithArgs . . . . .	2
counties . . . . .	3
digits2number . . . . .	4
generatePermuteString . . . . .	4
integrations . . . . .	5
number2digits . . . . .	6
offsetX . . . . .	6
permute . . . . .	8

topBottomDistribute . . . . .	9
tukeyPermutates . . . . .	10
tukeyT . . . . .	10
tukeyTexture . . . . .	11
vanDerCorput . . . . .	12
vipor . . . . .	12
vpPlot . . . . .	13

<b>Index</b>	<b>15</b>
--------------	-----------

---

aveWithArgs	<i>the ave() function but with arguments passed to FUN</i>
-------------	--

---

### Description

A function is applied to subsets of x where each subset consist of those observations with the same groupings in y

### Usage

```
aveWithArgs(x, y, FUN = mean, ...)
```

### Arguments

x	a vector to apply FUN to
y	a vector or list of vectors of grouping variables all of the same length as x
FUN	function to apply for each factor level combination.
...	additional arguments to FUN

### Value

A numeric vector of the same length as x where an each element contains the output from FUN after FUN was applied on the corresponding subgroup for that element (repeated if necessary within a subgroup).

### See Also

[ave](#)

### Examples

```
aveWithArgs(1:10,rep(1:5,2))
aveWithArgs(c(1:9,NA),rep(1:5,2),max,na.rm=TRUE)
```

---

`counties`*Census data on US counties*

---

**Description**

A dataset containing data from the US census bureau

**Usage**

```
counties
```

**Format**

A data frame with 3143 rows and 8 variables:

**id** GEO.id from original data

**state** state in which the county is located

**county** name of the county

**population** population of the county

**housingUnits** housing units in the county

**totalArea** Area in square miles - Total area

**waterArea** Area in square miles - Water area

**landArea** Area in square miles - Land area

**Source**

[http://factfinder.census.gov/bkmk/table/1.0/en/DEC/10\\_SF1/GCTPH1.US05PR](http://factfinder.census.gov/bkmk/table/1.0/en/DEC/10_SF1/GCTPH1.US05PR) (link now dead), `system.file("data-raw", "makeCounties.R", package = "vipor")`

**References**

<https://web.archive.org/web/20150326040847/https://www.census.gov/prod/cen2010/cph-2-1.pdf>

---

digits2number	<i>Convert a vector of integers representing digits in an arbitrary base to an integer</i>
---------------	--

---

### Description

Takes a vector of integers representing digits in an arbitrary base e.g. binary or octal and converts it into an integer (or the integer divided by  $\text{base}^{\text{length}(\text{digits})}$  for the number of digits if fractional is TRUE). Note that the first digit in the input is the least significant.

### Usage

```
digits2number(digits, base = 2, fractional = FALSE)
```

### Arguments

digits	a vector of integers representing digits in an arbitrary base
base	the base for the numeral system (e.g. 2 for binary or 8 for octal)
fractional	divide the output by the max for this number of digits and base. Note that this is $\text{base}^{\text{length}(\text{digits})}$ not $\text{base}^{\text{length}(\text{digits})-1}$ .

### Value

an integer

### References

<https://en.wikipedia.org/wiki/Radix>

### Examples

```
digits2number(c(4,4,1),8)
digits2number(number2digits(100))
```

---

generatePermuteString	<i>Generate a permutation string meeting Tukey criteria</i>
-----------------------	---

---

### Description

Find a random string of concatenated permutations of 1:n fulfilling Tukey's criteria that there are no runs of 3 or more increases or decreases in a row. Tukey just uses the default  $n=5$ .

### Usage

```
generatePermuteString(nReps = 20, n = 5)
```

**Arguments**

nReps            number of permutations to concatenate  
n                permutations from 1 to n

**Value**

a vector of nReps\*n integers giving concatenated permutations

**Examples**

```
tukeyPermuter()  
tukeyPermuter(6,3)
```

---

integrations	<i>Data on HIV integration sites from several studies</i>
--------------	---

---

**Description**

A dataset containing data from a meta-analysis looking for differences between active and inactive HIV integrations. Each row represents a provirus integrated somewhere in a human chromosome with whether viral expression was detected, the distance to the nearest gene and the number of reads from H4K12ac ChIP-Seq mapped to within 50,000 bases of the integration.

**Usage**

```
integrations
```

**Format**

A data frame with 12436 rows and 4 variables:

**study** the cell population infected by HIV

**latent** whether the provirus was active (expressed) or inactive (latent)

**nearestGene** distance to nearest gene (transcription unit) (0 if in a gene)

**H4K12ac** number of reads aligned within +/- 50,000 bases in a H4K12ac ChIP-Seq

**Source**

<https://retrovirology.biomedcentral.com/articles/10.1186/1742-4690-10-90>, system.file("data-raw", "makeIntegrations.R", package = "vipor")

**References**

<https://retrovirology.biomedcentral.com/articles/10.1186/1742-4690-10-90>

---

number2digits	<i>Convert an integer to an arbitrary base</i>
---------------	--

---

**Description**

Takes an integer and converts it into an arbitrary base e.g. binary or octal. Note that the first digit in the output is the least significant.

**Usage**

```
number2digits(n, base = 2)
```

**Arguments**

n	the integer to be converted
base	the base for the numeral system (e.g. 2 for binary or 8 for octal)

**Value**

a vector of length  $\text{ceiling}(\log(n+1, \text{base}))$  representing each digit for that numeral system

**References**

<https://en.wikipedia.org/wiki/Radix>

**Examples**

```
number2digits(100)
number2digits(100,8)
```

---

offsetX	<i>Offset data using quasirandom noise to avoid overplotting</i>
---------	--

---

**Description**

Arranges data points using quasirandom noise (van der Corput sequence), pseudorandom noise or alternatively positioning extreme values within a band to the left and right to form beeswarm/one-dimensional scatter/strip chart style plots. That is a plot resembling a cross between a violin plot (showing the density distribution) and a scatter plot (showing the individual points). This function returns a vector of the offsets to be used in plotting.

**Usage**

```
offsetX(y, x = rep(1, length(y)), width = 0.4, varwidth = FALSE, ...)

offsetSingleGroup(
  y,
  maxLength = NULL,
  method = c("quasirandom", "pseudorandom", "smiley", "maxout", "frowney", "minout",
    "tukey", "tukeyDense"),
  nbins = NULL,
  adjust = 1
)
```

**Arguments**

y	vector of data points
x	a grouping factor for y (optional)
width	the maximum spacing away from center for each group of points. Since points are spaced to left and right, the maximum width of the cluster will be approximately width*2 (0 = no offset, default = 0.4)
varwidth	adjust the width of each group based on the number of points in the group
...	additional arguments to offsetSingleGroup
maxLength	multiply the offset by sqrt(length(y)/maxLength) if not NULL. The sqrt is to match boxplot (allows comparison of order of magnitude different ns, scale with standard error)
method	method used to distribute the points: <b>quasirandom:</b> points are distributed within a kernel density estimate of the distribution with offset determined by quasirandom Van der Corput noise <b>pseudorandom:</b> points are distributed within a kernel density estimate of the distribution with offset determined by pseudorandom noise a la jitter <b>maxout:</b> points are distributed within a kernel density with points in a band distributed with highest value points on the outside and lowest in the middle <b>minout:</b> points are distributed within a kernel density with points in a band distributed with highest value points in the middle and lowest on the outside <b>tukey:</b> points are distributed as described in Tukey and Tukey "Strips displaying empirical distributions: I. textured dot strips" <b>tukeyDense:</b> points are distributed as described in Tukey and Tukey but are constrained with the kernel density estimate
nbins	the number of points used to calculate density (defaults to 1000 for quasirandom and pseudorandom and 100 for others)
adjust	adjust the bandwidth used to calculate the kernel density (smaller values mean tighter fit, larger values looser fit, default is 1)

**Value**

a vector with of x-offsets of the same length as y

**Examples**

```
## Generate fake data
dat <- list(rnorm(50), rnorm(500), c(rnorm(100), rnorm(100,5)), rcauchy(100))
names(dat) <- c("Normal", "Dense Normal", "Bimodal", "Extremes")

## Plot each distribution with a variety of parameters
par(mfrow=c(4,1), mar=c(2,4, 0.5, 0.5))
sapply(names(dat),function(label) {
  y<-dat[[label]]

  offsets <- list(
    'Default'=offsetX(y),
    'Smoother'=offsetX(y, adjust=2),
    'Tighter'=offsetX(y, adjust=0.1),
    'Thinner'=offsetX(y, width=0.1)
  )
  ids <- rep(1:length(offsets), sapply(offsets,length))

  plot(unlist(offsets) + ids, rep(y, length(offsets)),
       ylab=label, xlab='', xaxt='n', pch=21, las=1)
  axis(1, 1:4, c("Default", "Adjust=2", "Adjust=0.1", "Width=10%"))
})
```

---

permute

*Return all permutations of a vector*


---

**Description**

Recursively generates all permutations of a vector. The result will be `factorial(length(vals))` long so be careful with any longer vectors (e.g. longer than 10).

**Usage**

```
permute(vals)
```

**Arguments**

`vals` a vector of elements to be permuted

**Value**

A list of vectors containing all permutation of the values

**See Also**

[sample](#)

**Examples**

```
permute(letters[1:3])
permute(1:5)
```

---

topBottomDistribute    *Produce offsets such that points are sorted with most extreme values to right and left*

---

**Description**

Produce offsets to generate smile-like or frown-like distributions of points. That is sorting the points so that the most extreme values alternate between the left and right e.g. (max,3rd max,...,4th max, 2nd max). The function returns either a proportion between 0 and 1 (useful for plotting) or an order

**Usage**

```
topBottomDistribute(x, frowney = FALSE, prop = TRUE)
```

**Arguments**

x	the elements to be sorted
frowney	if TRUE then sort minimums to the outside, otherwise sort maximums to the outside
prop	if FALSE then return an ordering of the data with extremes on the outside. If TRUE then return a sequence between 0 and 1 sorted by the ordering

**Value**

a vector of the same length as x with values ranging between 0 and 1 if prop is TRUE or an ordering of 1 to length(x)

**Examples**

```
topBottomDistribute(1:10)
topBottomDistribute(1:10,TRUE)
```

---

tukeyPermutates	<i>Find permutations meeting Tukey criteria</i>
-----------------	---

---

**Description**

Find all permutations of 1:n fulfilling Tukey's criteria that there are no runs of 3 or more increases or decreases in a row. Tukey just uses the default n=5 and limit=2.

**Usage**

```
tukeyPermutates(n = 5, limit = 2)
```

**Arguments**

n	permutations from 1 to n
limit	the maximum number of increases or decreases in a row

**Value**

a list of vectors containing valid permutations

**Examples**

```
tukeyPermutates()
tukeyPermutates(6, 3)
```

---

tukeyT	<i>Combine multiple permutation strings into one</i>
--------	--

---

**Description**

Combine base+1 permutation strings to generate offsets

**Usage**

```
tukeyT(nReps = 10, base = 5)
```

**Arguments**

nReps	number of permutations to paste together
base	generate permutations of integers 1:base

**Value**

A nReps\*base length vector giving offset positions based on Tukey's algorithm

**Examples**

```
tukeyT()
tukeyT()
tukeyT(5,4)
```

---

tukeyTexture	<i>Generate random positions based on Tukey texture algorithm</i>
--------------	---

---

**Description**

Generate partly random, partly constrained lateral displacements based on Tukey texture algorithm from Tukey and Tukey 1990

**Usage**

```
tukeyTexture(
  x,
  jitter = TRUE,
  thin = FALSE,
  hollow = FALSE,
  delta = diff(stats::quantile(x, c(0.25, 0.75))) * 0.03
)
```

**Arguments**

x	the points to be jittered. really only used to calculate length
jitter	if TRUE add random jitter to each point
thin	if TRUE then push points to the center in thin regions
hollow	if TRUE then expand points outward to avoid “hollowness”
delta	a “reasonably small value” used in edge straightening and thinning

**Value**

a vector of length length(x) giving displacements for each corresponding point in x

**Examples**

```
x<-rnorm(200)
plot(tukeyTexture(x),x)
x<-1:100
plot(tukeyTexture(x),x)
plot(tukeyTexture(log10(counties$landArea),TRUE,TRUE),log10(counties$landArea),cex=.25)
```

---

vanDerCorput	<i>Generate van der Corput sequences</i>
--------------	--

---

**Description**

Generates the first (or an arbitrary offset) n elements of the van der Corput low-discrepancy sequence for a given base

**Usage**

```
vanDerCorput(n, base = 2, start = 1)
```

**Arguments**

n	the first n elements of the van der Corput sequence
base	the base to use for calculating the van der Corput sequence
start	start at this position in the sequence

**Value**

a vector of length n with values ranging between 0 and 1

**References**

[https://en.wikipedia.org/wiki/Van\\_der\\_Corput\\_sequence](https://en.wikipedia.org/wiki/Van_der_Corput_sequence)

**Examples**

```
vanDerCorput(100)
```

---

vipor	<i>Functions to generate violin scatter plots</i>
-------	---

---

**Description**

Arranges data points using quasirandom noise (van der Corput sequence) to create a plot resembling a cross between a violin plot (showing the density distribution) and a scatter plot (showing the individual points). The development version of this package is on <https://github.com/sherrillmix/vipor>

**Details**

The main functions are:

**offsetX**: calculate offsets in X position for plotting (groups of) one dimensional data

**vpPlot**: a simple wrapper around plot and offsetX to generate plots of grouped data

**Author(s)**

Scott Sherrill-Mix, <shescott@upenn.edu>

**See Also**

<https://github.com/sherrillmix/vipor>

**Examples**

```
dat<-list(rnorm(100),rnorm(50,1,2))
ids<-rep(1:length(dat),sapply(dat,length))
offset<-offsetX(unlist(dat),ids)
plot(unlist(dat),ids+offset)
```

---

vpPlot

*Plot data using offsets by quasirandom noise to generate a violin point plot*

---

**Description**

Arranges data points using quasirandom noise (van der Corput sequence), pseudorandom noise or alternatively positioning extreme values within a band to the left and right to form beeswarm/one-dimensional scatter/strip chart style plots. That is a plot resembling a cross between a violin plot (showing the density distribution) and a scatter plot (showing the individual points) and so here we'll call it a violin point plot.

**Usage**

```
vpPlot(x = rep("Data", length(y)), y, xaxt = "y", offsetXArgs = NULL, ...)
```

**Arguments**

x	a grouping factor for y (optional)
y	vector of data points
xaxt	if 'n' then no x axis is plotted
offsetXArgs	a list with arguments for offsetX
...	additional arguments to plot

**Value**

invisibly return the adjusted x positions of the points

**See Also**

[offsetX](#)

**Examples**

```
dat<-list(  
  'Mean=0'=rnorm(200),  
  'Mean=1'=rnorm(50,1),  
  'Bimodal'=c(rnorm(40,-2),rnorm(60,2)),  
  'Gamma'=rgamma(50,1)  
)  
labs<-factor(rep(names(dat),sapply(dat,length)),levels=names(dat))  
vpPlot(labs,unlist(dat))
```

# Index

- \* **datasets**
  - counties, [3](#)
  - integrations, [5](#)
- ave, [2](#)
- aveWithArgs, [2](#)
- counties, [3](#)
- digits2number, [4](#)
- generatePermuteString, [4](#)
- integrations, [5](#)
- number2digits, [6](#)
- offsetSingleGroup (offsetX), [6](#)
- offsetX, [6](#), [12](#), [13](#)
- permute, [8](#)
- sample, [8](#)
- topBottomDistribute, [9](#)
- tukeyPermutates, [10](#)
- tukeyT, [10](#)
- tukeyTexture, [11](#)
- vanDerCorput, [12](#)
- vipor, [12](#)
- vpPlot, [12](#), [13](#)