

# Package ‘venn’

May 8, 2026

**Version** 1.13

**Title** Draw Venn Diagrams

**Depends** R (>= 3.5.0)

**Imports** admisc (>= 0.33)

**Suggests** QCA (>= 3.9), ggplot2, ggpolypath

**Description** A close to zero dependency package to draw and display Venn diagrams up to 7 sets, and any Boolean union of set intersections.

**License** GPL (>= 3)

**URL** <https://github.com/dusadrian/venn>

**BugReports** <https://github.com/dusadrian/venn/issues>

**NeedsCompilation** no

**Author** Adrian Dusa [aut, cre, cph] (ORCID:  
<<https://orcid.org/0000-0002-3525-9253>>)

**Maintainer** Adrian Dusa <dusa.adrian@unibuc.ro>

**Repository** CRAN

**Date/Publication** 2026-03-31 06:50:02 UTC

## Contents

|                                  |   |
|----------------------------------|---|
| About the venn package . . . . . | 2 |
| extractInfo . . . . .            | 2 |
| getCentroid . . . . .            | 3 |
| getZones . . . . .               | 4 |
| venn . . . . .                   | 6 |

|              |           |
|--------------|-----------|
| <b>Index</b> | <b>12</b> |
|--------------|-----------|

About the venn package

*Draw Venn Diagrams*

---

### **Description**

A close to zero dependency package to draw and display Venn diagrams, and any boolean union of set intersections.

### **Details**

Package: venn  
Type: Package  
Version: 1.13  
Date: 2026-03-31  
License: GPL (>= 2)

### **Author(s)**

**Authors:**

Adrian Dusa  
Department of Sociology  
University of Bucharest  
<dusa.adrian@unibuc.ro>

**Maintainer:**

Adrian Dusa

---

extractInfo

*Extract information about a list object.*

---

### **Description**

Extracts useful information such as the counts or the actual intersections between several sets.

### **Usage**

```
extractInfo(  
  x, what = c("counts", "intersections", "both"), use.names = FALSE  
)
```

**Arguments**

|           |   |
|-----------|---|
| x         | A list object containing set elements, or a list of binary values |
| what      | What to extract   |
| use.names | Logical, use the set names to indicate intersections              |

**Details**

When the argument `x` is a list, the counts are taken from the number of common values for each intersection, and when `x` is a dataframe, (comprised of exclusively binary values 0 and 1) the counts are taken from the number of similar rows.

**Value**

A dataframe, when extracting the counts or a list if extracting intersections.

**Examples**

```
set.seed(12345)
xlist <- list(A = 1:20, B = 10:30, C = sample(25:50, 15))
xdf <- as.data.frame(matrix(
  sample(0:1, 90, replace = TRUE),
  ncol = 3
))
colnames(xdf) <- LETTERS[1:3]

extractInfo(xlist) # counts by default

extractInfo(xlist, what = "intersections")

extractInfo(xlist, what = "both")

extractInfo(xdf)
```

---

getCentroid

*Calculate the centroid of a polygon.*

---

**Description**

This function takes a list of dataframes or a matrices containing `x` and `y` values, which define zones (polygons), and calculates their centroids.

**Usage**

```
getCentroid(data)
```

**Arguments**

|      |   |
|------|---|
| data | A matrix or a dataframe with two columns, for <code>x</code> and <code>y</code> coordinates |
|------|---|

**Details**

Most of the coordinates for the intersection labels in this package were calculated using the formula for a centroid of a non-self-intersecting closed polygon, approximated by 10 vertices.

**Value**

A list with x and y coordinates, for each zone in the input list.

**References**

Centroid. (n.d.). In Wikipedia. Retrieved January 06, 2016, from <https://en.wikipedia.org/wiki/Centroid>

**Examples**

```
venn("0110")

# centroid for the intersection "0110" in a 4 set diagram
centroid <- getCentroid(getZones("0110"))[[1]]

text(centroid[1], centroid[2], labels = "0110", cex = 0.85)

# centroids for the two zones in the "E not A" zones
venn(5)
area <- getZones("0--1") # list of length 2

polygon(area[[1]], col="lightblue")

polygon(area[[2]], col="lightblue")

text(do.call("rbind", getCentroid(area)),
      labels = c("zone 1", "zone 2"), cex = 0.85)
```

---

getZones

*Calculate the union(s) of set intersections.*

---

**Description**

This function uses a metaccommand to calculate the shape of a specific zone or a list of zones.

**Usage**

```
getZones(area, snames, ellipse = FALSE)
```

**Arguments**

|         |   |
|---------|---|
| area    | A character expression written in sum of products form.             |
| snames  | A string containing the sets' names, separated by commas.           |
| ellipse | Logical, get the zones from the shape of an ellipse, where possible |

## Details

A SOP ("sum of products") is also known as a DNF ("disjunctive normal form"), or in other words a "union of intersections", for example  $A \cdot D + B \cdot c$ .

The same expression can be written in curly brackets notation:  $A\{1\} \cdot D\{1\} + B\{1\} \cdot C\{0\}$ .

The expression  $B\{1\} \cdot C\{0\}$  can also be written in a pseudo-language, as "-10-" (assuming there are only four sets).

A "zone" is a union of set intersections. There are exactly  $2^k$  intersections in a Venn diagram, where  $k$  is the number of sets. To highlight an entire set, we need a union of all possible intersections which form that set.

The argument `ellipse` retrieves the data from the shape of an ellipse, and it only works with 4 and 5 sets.

## Value

A list of self-enclosed polygons, for each independent zone.

## Examples

```
venn(3)

area <- getZones("A", snames = "A, B, C")
# a list of length 1

polygon(area[[1]], col="lightblue")

# The very same result is obtained with:
zone <- getZones("1--")

# for 5 sets, the content of the 5th set but not in the first set is a
# list of two zones

venn(5)

zones <- getZones("0---1")
# this time a list of length 2

# (re)coloring the first zone (union)
polygon(zones[[1]], col="lightblue")

# and the second zone (union)
polygon(zones[[2]], col="lightblue")
```

---

venn *Draw and display a Venn diagram*

---

### Description

This function uses a variety of input data to draw and display a Venn diagram with up to 7 sets.

### Usage

```
venn(x, snames = "", ilabels = NULL, ellipse = FALSE, zcolor = "bw",
     opacity = 0.3, plotsize = 15, ilcs = 0.6, sncs = 0.85, borders = TRUE,
     box = TRUE, par = TRUE, ggplot = FALSE, ...)
```

### Arguments

|          |   |
|----------|---|
| x        | A single number (of sets), or a metacommand formula (see details), or a list containing set values, or a dataset containing boolean values. |
| snames   | An optional parameter containing the names for each set.  |
| ilabels  | Complex argument, see Details.  |
| ellipse  | Logical, force the shape to an ellipse, where possible  |
| zcolor   | A vector of colors for the custom zones, or predefined colors if "style"  |
| opacity  | Degree of opacity for the color(s) specified with zcolor (less opacity, more transparency).   |
| plotsize | Plot size, in centimeters.  |
| ilcs     | Character expansion (in base plots) or size (in ggplots) for the intersection labels  |
| sncs     | Character expansion (in base plots) or size (in ggplots) for the set names  |
| borders  | Logical: draw all intersection borders  |
| box      | Logical: draw the outside square  |
| par      | Logical: use the default, custom par settings   |
| ggplot   | Logical: plot the Venn diagram using ggplot   |
| ...      | Additional parameters, mainly for the outer borders of the sets   |

### Details

The argument x can be either:

- a single number (of sets), between 1 and 7
- a metacommand (character) to draw custom intersection zones
- a list, containing values for the different sets: each component is a set, and only up to 7 components are processed.
- a dataset of boolean values.

A "zone" is a union of set intersections. There are exactly  $2^k$  intersections in a Venn diagram, where  $k$  is the number of sets. To highlight an entire set, we need a union of all possible intersections which form that set.

For example, in a 3 sets diagram, the (overall) first set is composed by four intersections:

100 for what is in the first set but outside sets 2 and outside set 3

101 for the intersection between sets 1 and 3, outside set 2

110 for the intersection between sets 1 and 2, outside set 3

111 for the intersection between all three sets.

A meta-language can be used to define these intersections, using the values of 1 for what is inside the set, 0 for what is outside the set, and - when its either inside or outside of the set.

The command "1-" is translated as "display only the first, entire set" is equivalent with the union of the four intersections "100 + 101 + 110 + 111".

The parameter `snames` should have the same length as the number of sets specified by the parameter `x`.

When the parameter `x` is used as a metacommand, the number of sets is calculated as the number of characters in each intersection of the metacommand. One such character command is "100 + 101 + 110 + 111" or "1-", and all intersections have exactly three characters.

It is also possible to use a regular, disjunctive normal form, like "A", which is equivalent with "AbC + AbC + AbC + ABC". When `x` is an expression written in DNF, if a valid R statement then quoting is not even necessary.

The argument `snames` establishes names for the different sets, or in its absence it is taken from LETTERS. When `x` is a list or a dataframe, `snames` is taken from their names. The length of the `snames` indicates the total number of sets.

A numerical vector can be supplied with the argument `ilabels`, when the argument `x` is a single number of sets. The vector should match the increasing order of the binary representation for the set intersections.

This argument can also be logical, and if activated with TRUE it constructs the intersection labels from their particular combinations of 0s and 1s.

Finally, it can also be specified as `ilabels = "counts"`, for counting the frequency of appearance of each intersection. When the argument `x` is a list, the counts are taken from the number of common values for each intersection, and when `x` is a dataframe, (comprised of exclusively binary values 0 and 1) the counts are taken from the number of similar rows. If a particular intersection does not have any common values (or no rows), the count "0" is left blank and not displayed in the diagram.

The argument `ellipse` differentiates between two types of diagrams for 4 and 5 sets. The idea is to allow for as much space as possible for each intersection (also as equal as possible) and that is impossible if preserving the shape of an ellipse. The default is to create large space for the intersections, but users who prefer an ellipse might want to set this argument to TRUE.

Colors to fill the desired zones (or entire sets) can be supplied via the argument `zcolor` (the default is "bw" black and white, which means no colors at all). Users can either chose the predefined color style, using `zcolor = "style"`, or supply a vector of custom colors for each zone. If only one custom color is supplied, it will be recycled for all zones.

When using `zcolor = "style"`, any other additional arguments for the borders are ignored.

A different set of predefined colors is used, when argument `x` is a QCA type object (a truth table, either from a class `tt` or from a class `qca`). If custom colors are provided via `zcolor`, it should have

a length of 3 colors: the first for the absence of the outcome (0), the second for the presence of the outcome (1), and the third for the contradictions (C). Remainders have no color, by default.

The argument `ilcs` works only if the intersection labels (`ilabels`) have information, and it sets the size of the labels via a `cex` argument. In the absence of a specific value from the user, it's default is set to 0.6 for all Venn diagrams with up to five sets, and it automatically decreases to 0.5 for six sets and 0.45 for seven sets.

Via . . . , users can specify additional parameters, mainly for the outer borders of the sets, as specified by `par()`, and since version 1.9 it is also used to pass additional aesthetics parameters for the `ggplot2` graphics. All of them are fed either to the base function `lines()` which is responsible with the borders, or to the function `geom_path()` from package **ggplot2**.

For up to 3 sets, the shapes can be circular. For more than 3 sets, the shape cannot be circular: for 4 and 5 sets they can be ellipses, while for more than 5 sets the shapes cannot be continuous (they might be monotone, but not continuous). The 7 sets diagram is called "Adelaide" (Ruskey, 2005).

The most challenging diagram is the one with 6 sets, where for many years it was thought a Venn diagram didn't even exist. All diagrams are symmetric, except for the one with 6 sets, where some of the sets have different shapes. The diagram in this package is an adaptation from Mamakani, K., Myrvold W. and F. Ruskey (2011).

The argument `border` can be used only for custom intersections and/or unions, it has no effect when `x` is a list, or a data frame, or a truth table object.

The argument `par` is used to define a custom set of parameters when producing the plot, to ensure a square shape of about 15 cm and eliminate the outer regions. If deactivated, users can define their own size and shape of the plot using the system function `par()`. By default, the plot is always produced using a size of 1000 points for both horizontal and vertical, unless the argument `ggplot` is activated, when the argument `par` will have no effect.

## References

Ruskey, F. and M. Weston. 2005. *Venn diagrams*. Electronic Journal of Combinatorics, Dynamic Survey DS5.

Mamakani, K., Myrvold W. and F. Ruskey. 2011. *Generating all Simple Convexly-drawable Polar Symmetric 6-Venn Diagrams*. International Workshop on Combinatorial Algorithms, Victoria. LNCS, 7056, 275-286.

## Examples

```
# A simple Venn diagram with 3 sets
venn(3)

# with a vector of counts: 1 for "000", 2 for "001" etc.
venn(3, counts = 1:8)

# display the first whole set
venn("1--")

# same with
venn("A", snames = "A, B, C")

# an equivalent command, from the union of all intersections
```

```
venn("100 + 110 + 101 + 111")

# same with
venn("A~B~C + AB~C + A~BC + ABC")

# adding the labels for the intersections
venn("1--", ilabels = TRUE)

# using different parameters for the borders
venn(4, lty = 5, col = "navyblue")

# using ellipses
venn(4, lty = 5, col = "navyblue", ellipse = TRUE)

# a 5 sets Venn diagram
venn(5)

# a 5 sets Venn diagram using ellipses
venn(5, ellipse = TRUE)

# a 5 sets Venn diagram with intersection labels
venn(5, ilabels = TRUE)

# and a predefined color style
venn(5, ilabels = TRUE, zcolor = "style")

# a union of two sets
venn("1---- + ----1")

# same with
venn("A + E", snames = "A, B, C, D, E")

# with different colors
venn("1---- , ----1", zcolor = "red, blue")

# same with
venn("A, E", snames = "A, B, C, D, E", zcolor = "red, blue")

# same colors for the borders
venn("1---- , ----1", zcolor = "red, blue", col = "red, blue")

# 6 sets diagram
venn(6)

# 7 sets "Adelaide"
venn(7)

# artistic version
venn(c("1000000", "0100000", "0010000", "0001000",
       "0000100", "0000010", "0000001", "1111111"))

# without all borders
```

```

venn(c("1000000", "0100000", "0010000", "0001000",
      "0000100", "0000010", "0000001", "1111111"),
     borders = FALSE)

# using sum of products notation
venn("A + B~C", snames = "A, B, C, D")

# when x is a list
set.seed(12345)
x <- list(First = 1:20, Second = 10:30, Third = sample(25:50, 15))
venn(x, ilabels = "counts")

# when x is a dataframe
set.seed(12345)
x <- as.data.frame(matrix(sample(0:1, 150, replace = TRUE), ncol = 5))
venn(x, ilabels = "counts")

# producing a ggplot2 graphics
venn(x, ilabels = "counts", ggplot = TRUE)

# increasing the border size
venn(x, ilabels = "counts", ggplot = TRUE, size = 1.5)

# with dashed lines
venn(x, ilabels = "counts", ggplot = TRUE, linetype = "dashed")

## Not run:
# produce Venn diagrams for QCA objects
library(QCA)

data(CVF)
obj <- truthTable(CVF, "PROTEST", incl.cut = 0.85)

venn(obj)

# to set opacity based on inclusion scores
# (less inclusion, more transparent)

venn(obj, opacity = obj$tt$incl)

# custom labels for intersections

pCVF <- minimize(obj, include = "?")
venn(pCVF$solution[[1]], zcol = "#ffdd77, #bb2020, #1188cc")
cases <- paste(c("HungariansRom", "CatholicsNIreland", "AlbaniansFYROM",
                "RussiansEstonia"), collapse = "\n")
coords <- unlist(getCentroid(getZones(pCVF$solution[[1]][2])))
text(coords[1], coords[2], labels = cases, cex = 0.85)

```

```
## End(Not run)
```

# Index

## \* **functions**

extractInfo, [2](#)

getCentroid, [3](#)

getZones, [4](#)

venn, [6](#)

## \* **package**

About the venn package, [2](#)

About the venn package, [2](#)

extractInfo, [2](#)

geom\_path, [8](#)

getCentroid, [3](#)

getZones, [4](#)

lines, [8](#)

par, [8](#)

venn, [6](#)

venn-package (About the venn package), [2](#)