UpSetColor usage

Francesc Puig-Castellvi

16/8/2022

The UpSetColor package

The *UpSetColor* package generates UpSet plots that are easily customisable from a list of sets or a binary matrix or data frame.

Some of the possible costumizations are the following: * Choose the set comparison criteria (intersection, distinct or union). * Limit the number of comparisons displayed according to several criteria. * Color independently the 3 panels of the upset plots. * Exclude some of the sets according to their size. * Sort the elements of the 3 panels. * Highlight set combinations. * Display on the console the progress of the set comparison, which can be useful when comparing a large number of sets.

In addition to all this, if further customization is desired, the main function of the package allows you to save each of the ggplot objects comprised in the UpSet plot. As if this were not enough, the user can also obtain the data frames used in the construction of each of the plots.

The idea of this package was born to join functionalities present in the UpSet-generator functions from the *UpSetR* and *ComplexHeatmap* packages, as well as to add some additional functionalities and allow the end-user to fully control the editing of the UpSet plots.

Installing UpSetColor

The package can be installed with devtools:

```
if (!require("devtools")) install.packages("devtools")
```

```
## Loading required package: devtools
```

```
## Loading required package: usethis
```

```
library("devtools")
install_github("f-puig/UpSetColor")
```

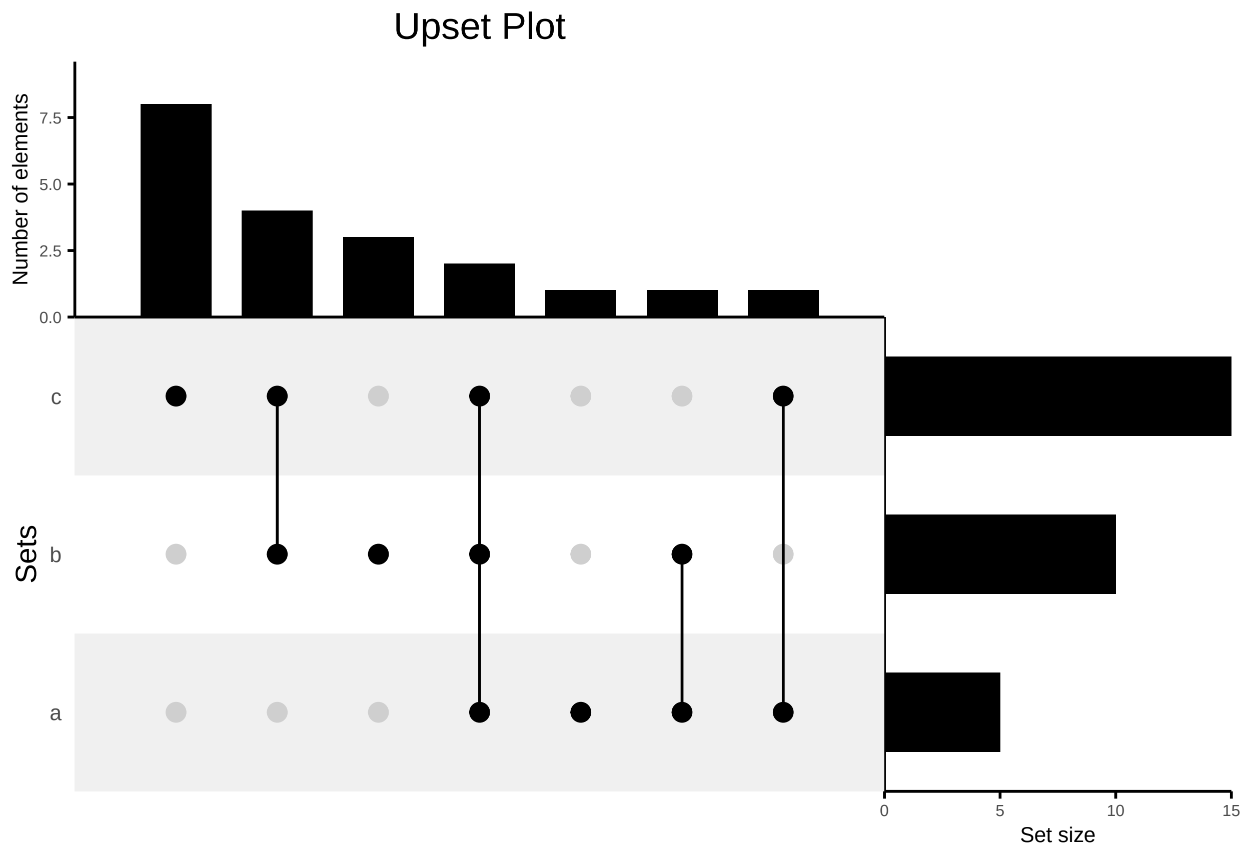
```
## Skipping install of 'UpSetColor' from a github remote, the SHA1 (93495fb9) has not changed since las
##   Use `force = TRUE` to force installation
```

```
# Load R.ComDim
library("UpSetColor")
```

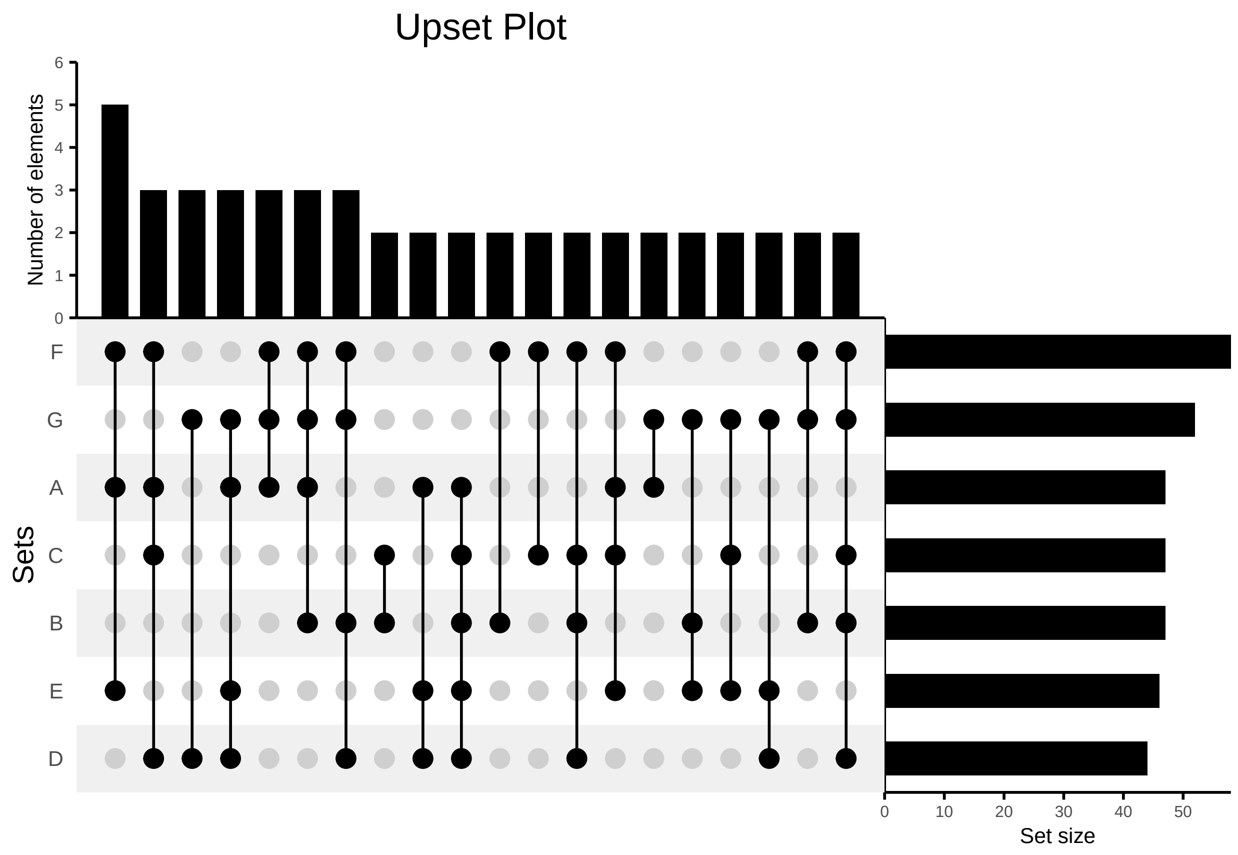
Starting with UpSetColor

The only function in the package is *UpSetColor*. This functions accept three types of data: A list of sets (set of the list of sets is a vector), a binary matrix or a binary data frame. For the last two, the elements are in the rows and the sets in the columns.

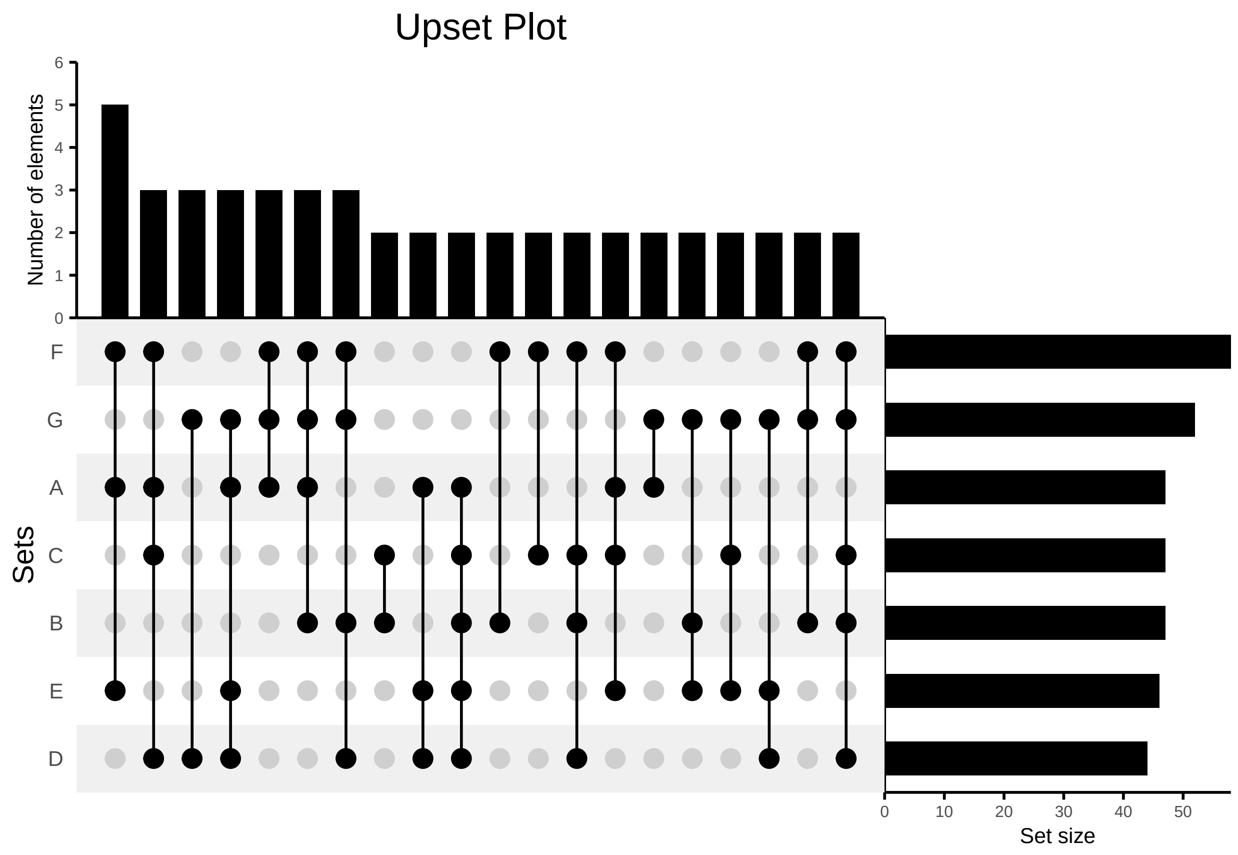
```
set.seed(123)

# Using a list
lt = list(a = sample(letters, 5),
          b = sample(letters, 10),
          c = sample(letters, 15))
UpSetColor(data = lt)
```



```
# Or a matrix
mm <- matrix(runif(700, min=0, max = 1),nrow =100,ncol = 7)
mm[mm <= 0.5] <- 0
mm[mm > 0.5] <- 1
colnames(mm) <- LETTERS[1:7]
UpSetColor(data = mm)
```



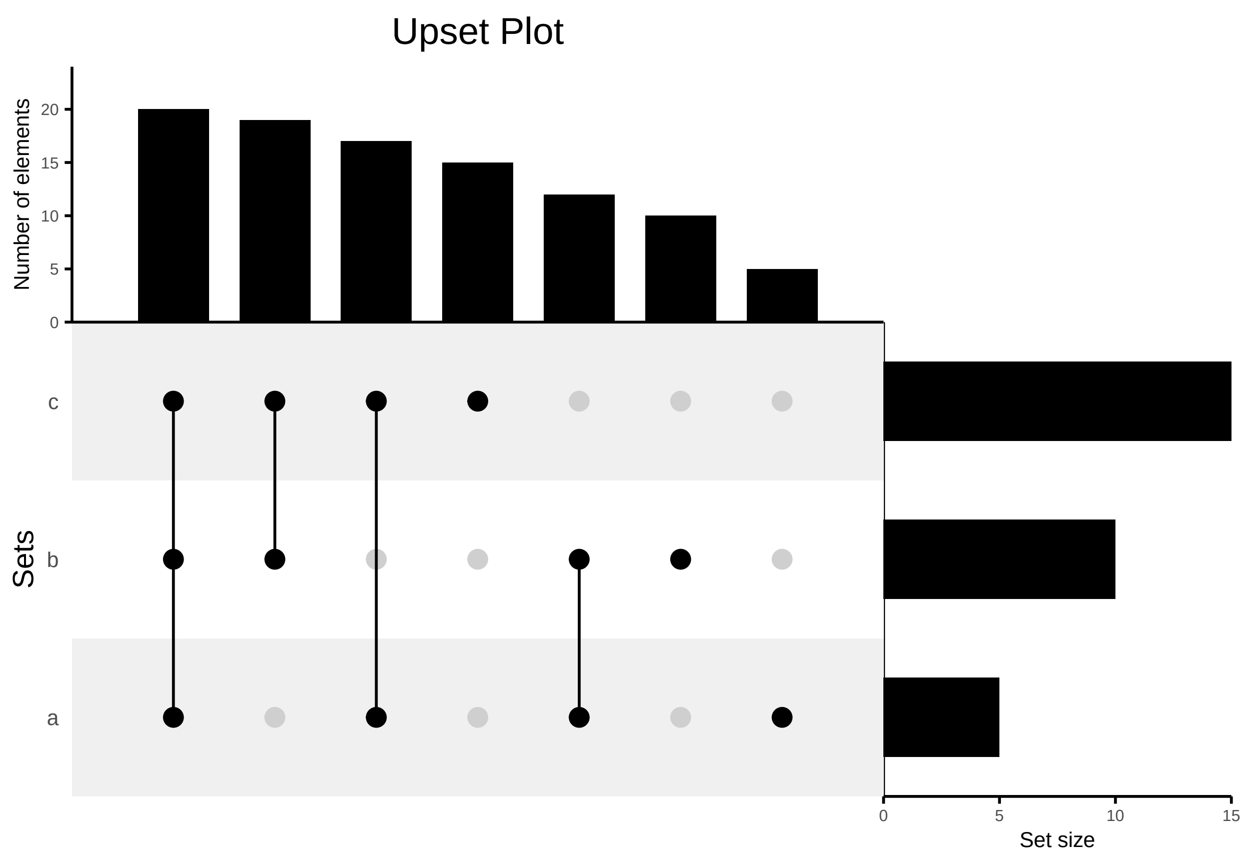
```r
# Or a data.frame
mmdf <- as.data.frame(mm)
UpSetColor(data = mmdf)
```



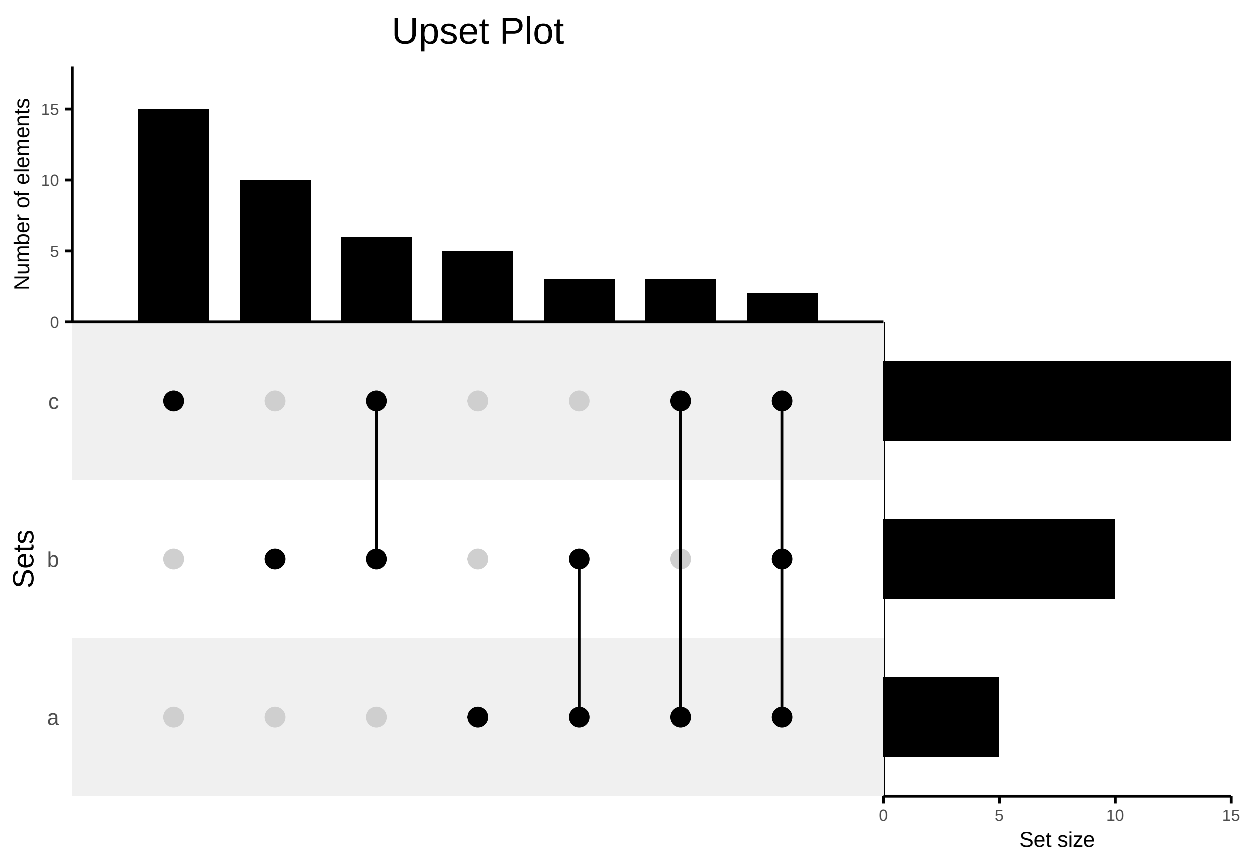
The mode

Using the same idea from the *ComplexHeatmap* package, sets can be compared with 3 different modes: “distinct” (the default mode), “intersect”, and “union”. In the *distinct* mode, if two sets are connected (1) in the matrix plot and a third one is not, it means that the combination bar in the upper plot shows what is common between the two sets and not uncommon from the third one. This mode is the traditional mode used in UpSet plots. In the *intersect* mode, the number of counted elements in the case of two connected sets does not depend on the unconnected ones. In the *union* mode, when two sets are connected, the number of counted elements was obtained from counting the single instances of the elements presents in at least one of both sets.

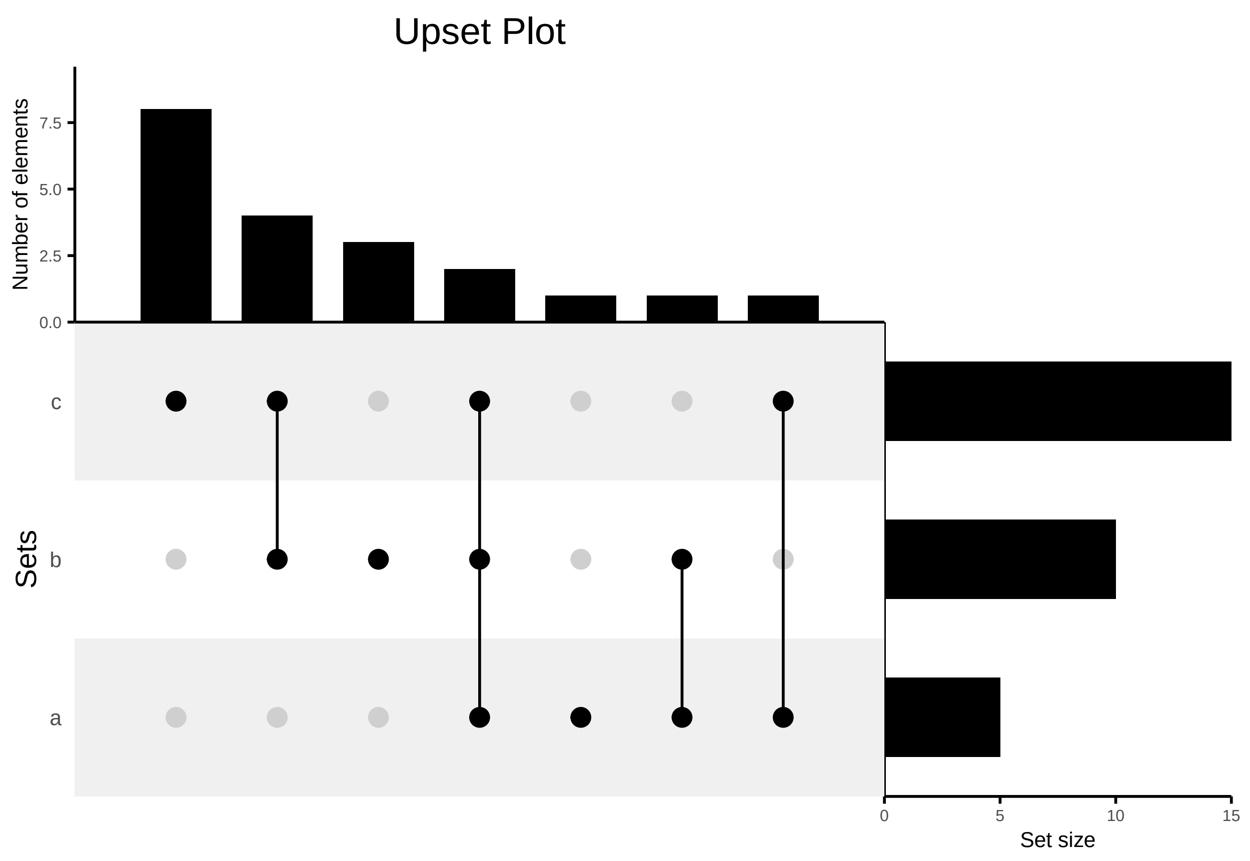
```
# Union mode
UpSetColor(data = lt, mode = 'union')
```



```
# Interpretation: There are 20 different letters across the 3 sets.
# 15 of these letters are in 'c'.

# Intersect mode
UpSetColor(data = lt, mode = 'intersect')
```



```
# Interpretation: There are 6 letters common between 'b' and 'c': "v" "k" "e" "n" "s" "i"
# This situation corresponds to: intersect(lt$b, lt$c)

# Distinct mode
UpSetColor(data = lt, mode = 'distinct')
```

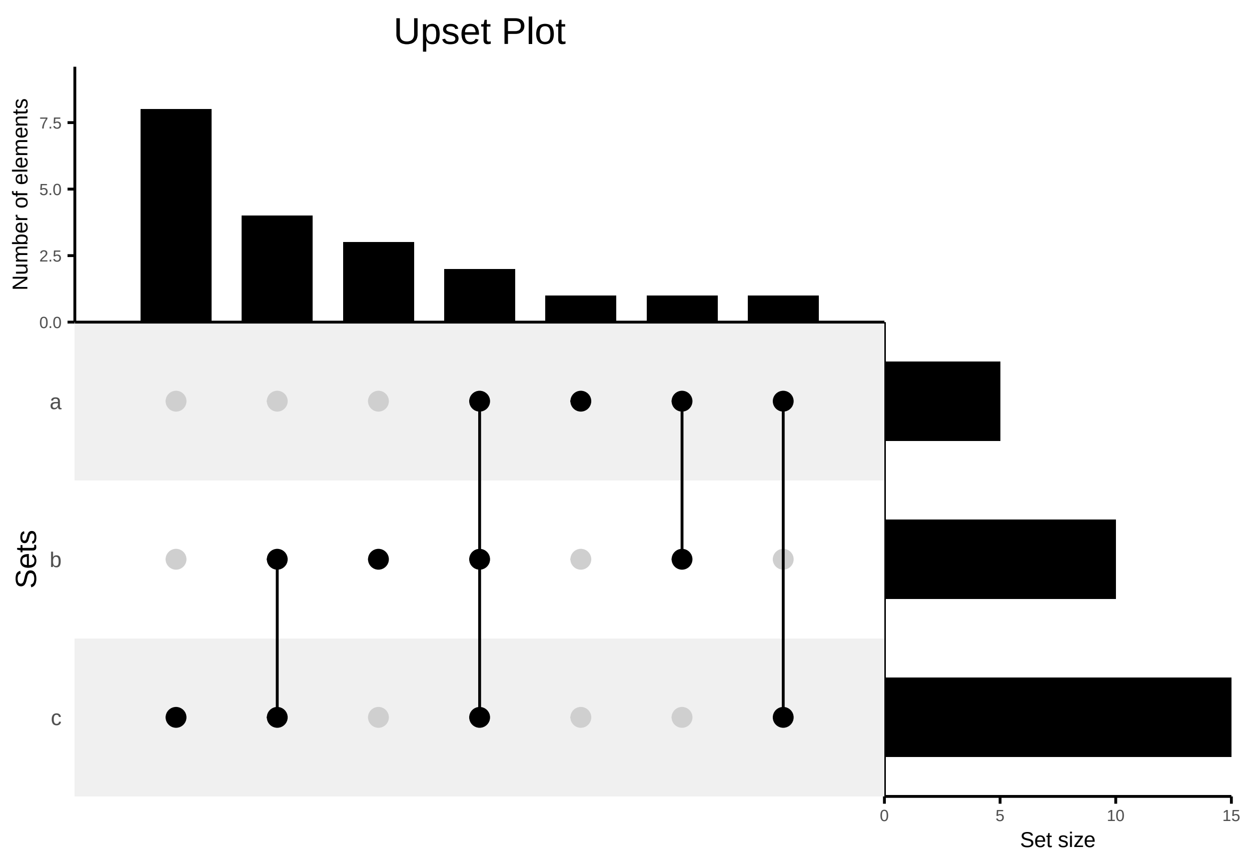


```
# Interpretation: There are 4 letters common between 'b' and 'c' and  absent from 'a'.
# These are: "v" "k" "e" "i"
# This situation corresponds to: setdiff(intersect(lt$b,lt$c), lt$a)
```

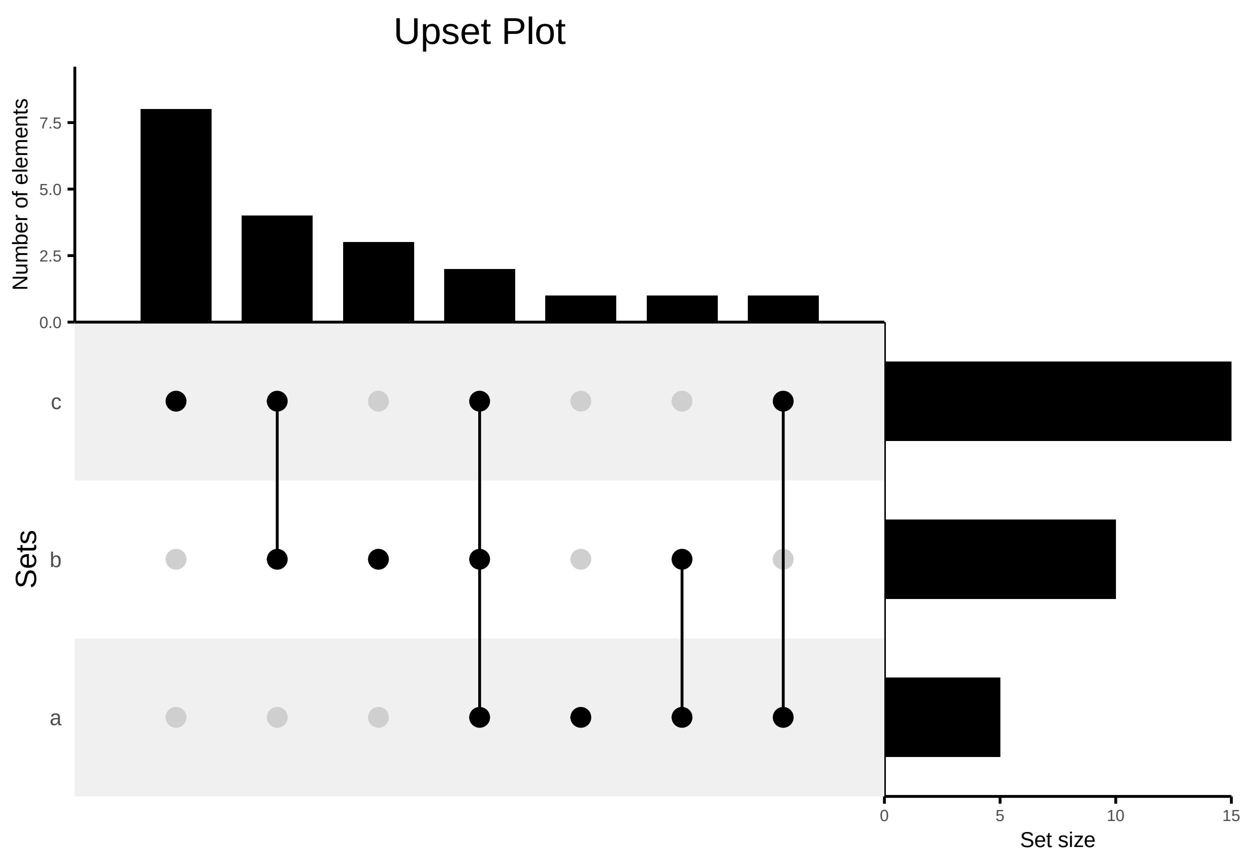
Sorting the data

The Sets and combinations displayed in the UpSet plot can be sorted with the arguments set_order and comb_order, correspondingly.

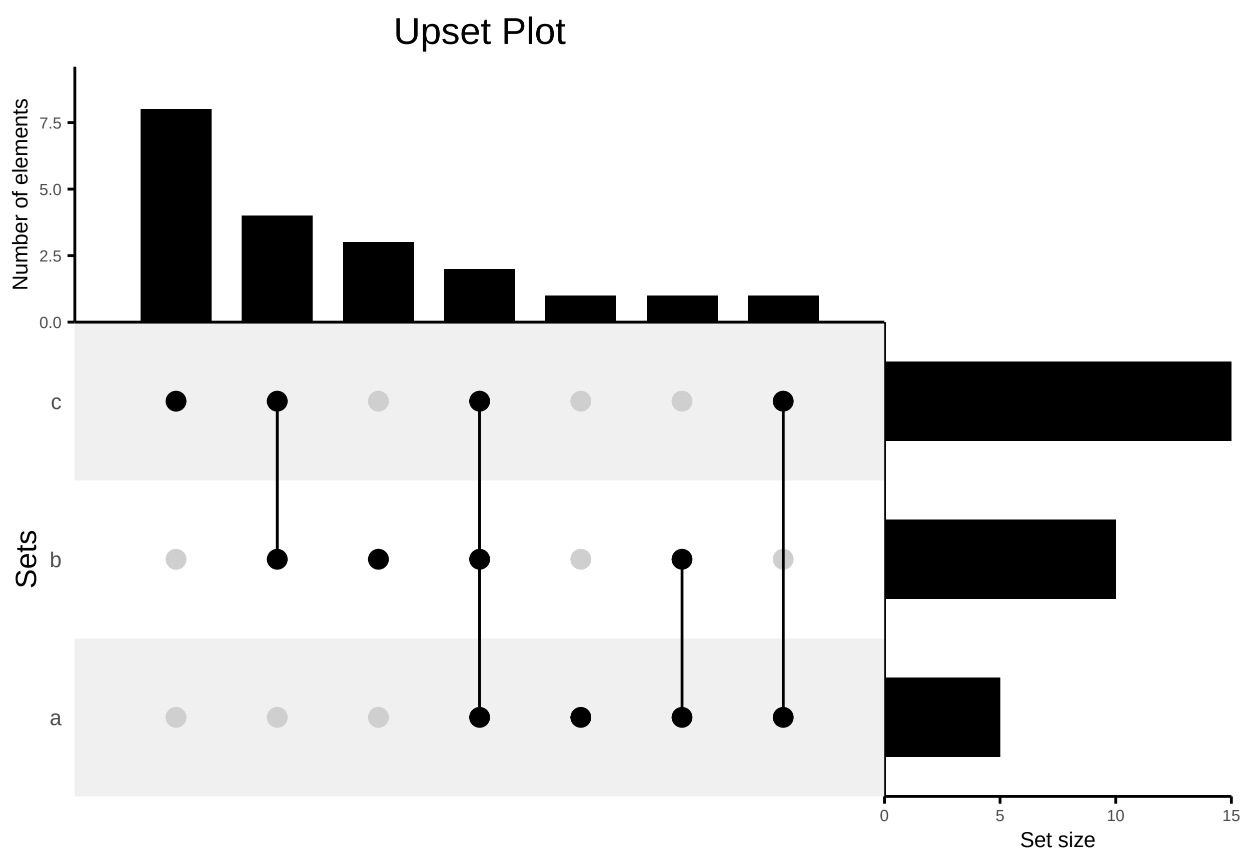
```
## Set order
# From bottom to top: c, b, a (15 > 10 > 5 elements)
UpSetColor(data = lt, set_order = 'decreasing')
```



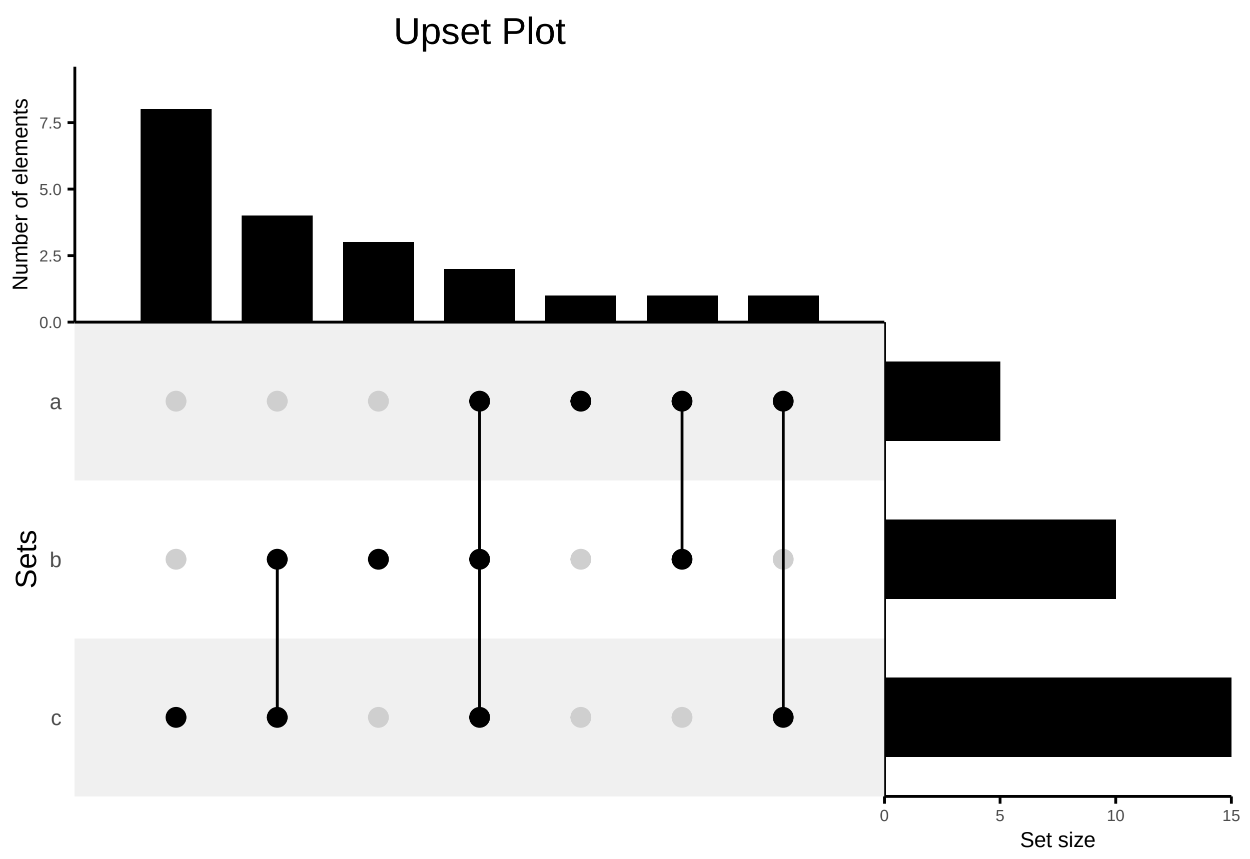
```
# From bottom to top: a, b, c (5 > 10 > 15 elements)
UpSetColor(data = lt, set_order = 'increasing')
```



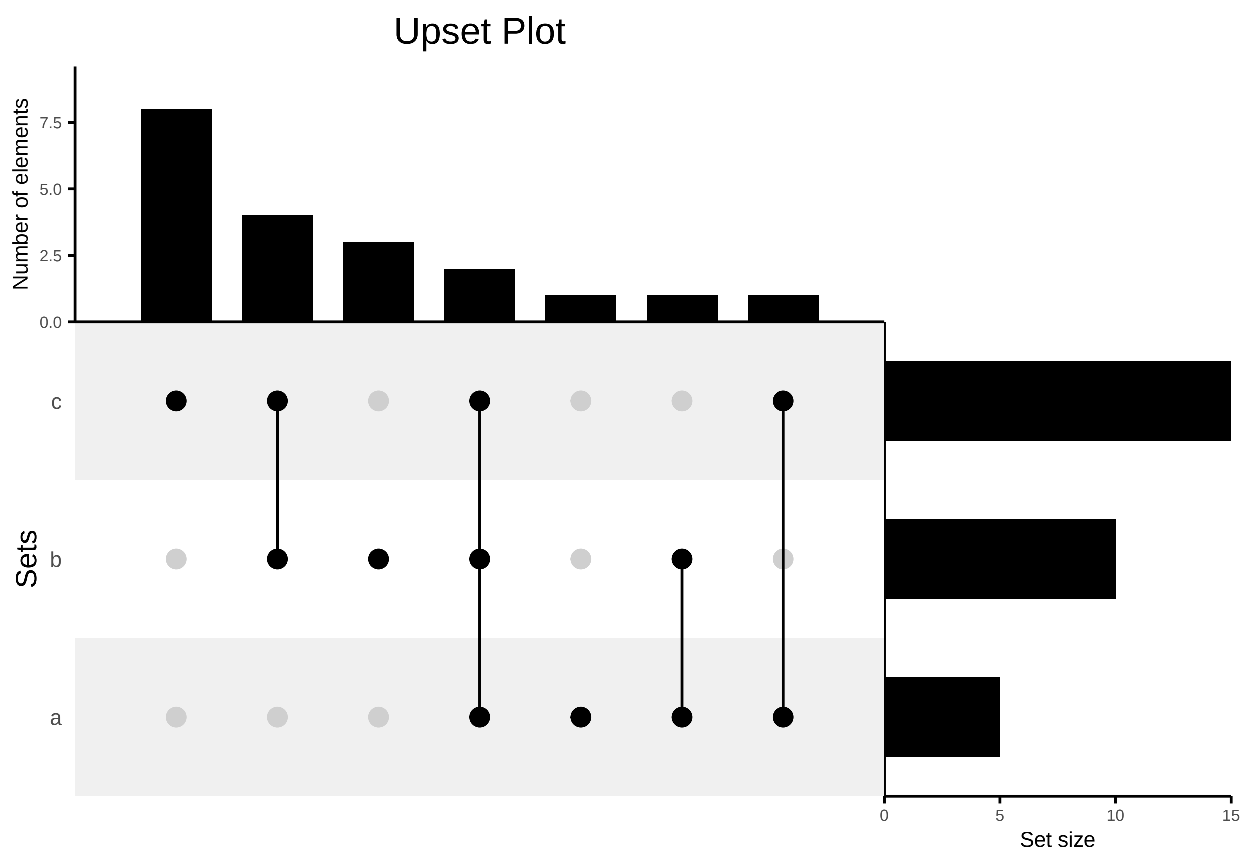
```
# From bottom to top: a (1st in lt), b (2nd), c (3rd) elements)
UpSetColor(data = lt, set_order = 'as.given')
```



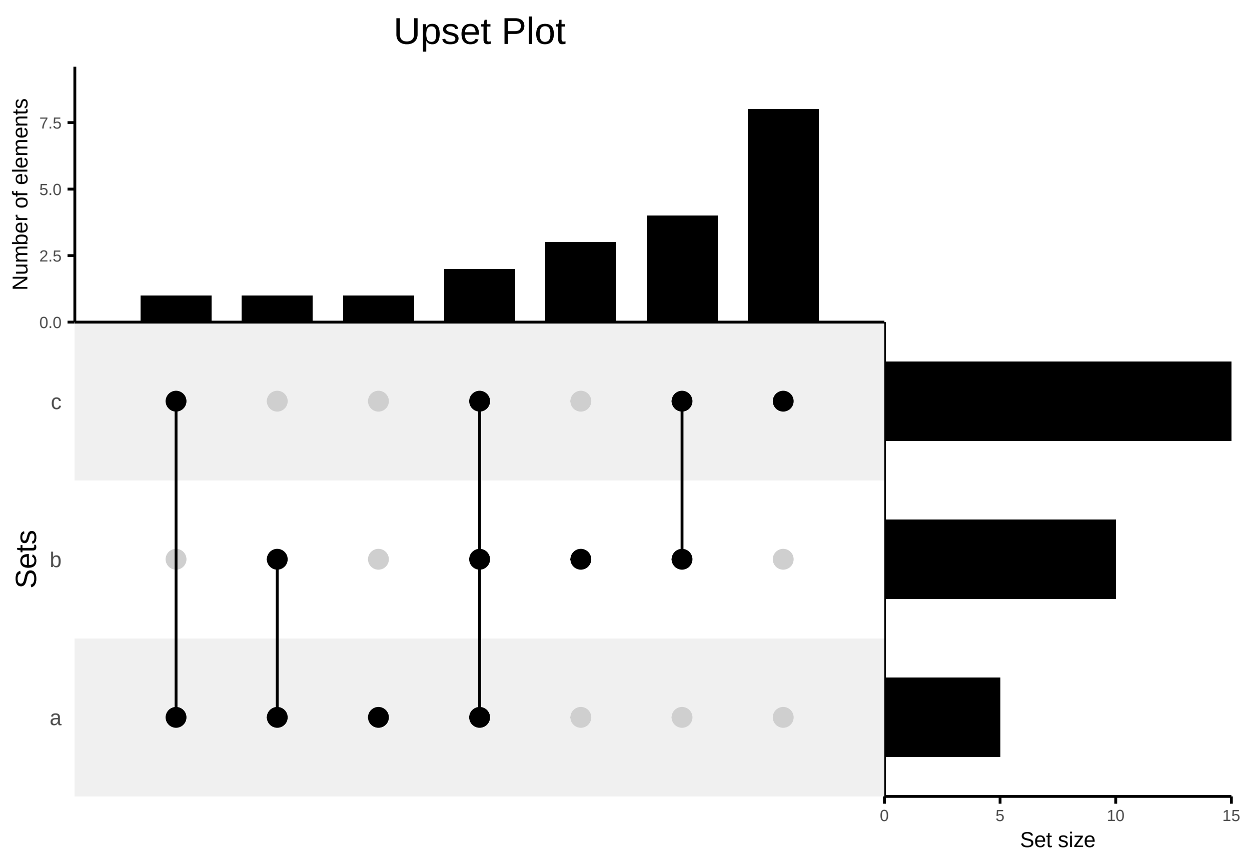
```
# From bottom to top: c (3rd in lt), b (2nd), c (1st) elements)
UpSetColor(data = lt, set_order = 'as.given.reverse')
```



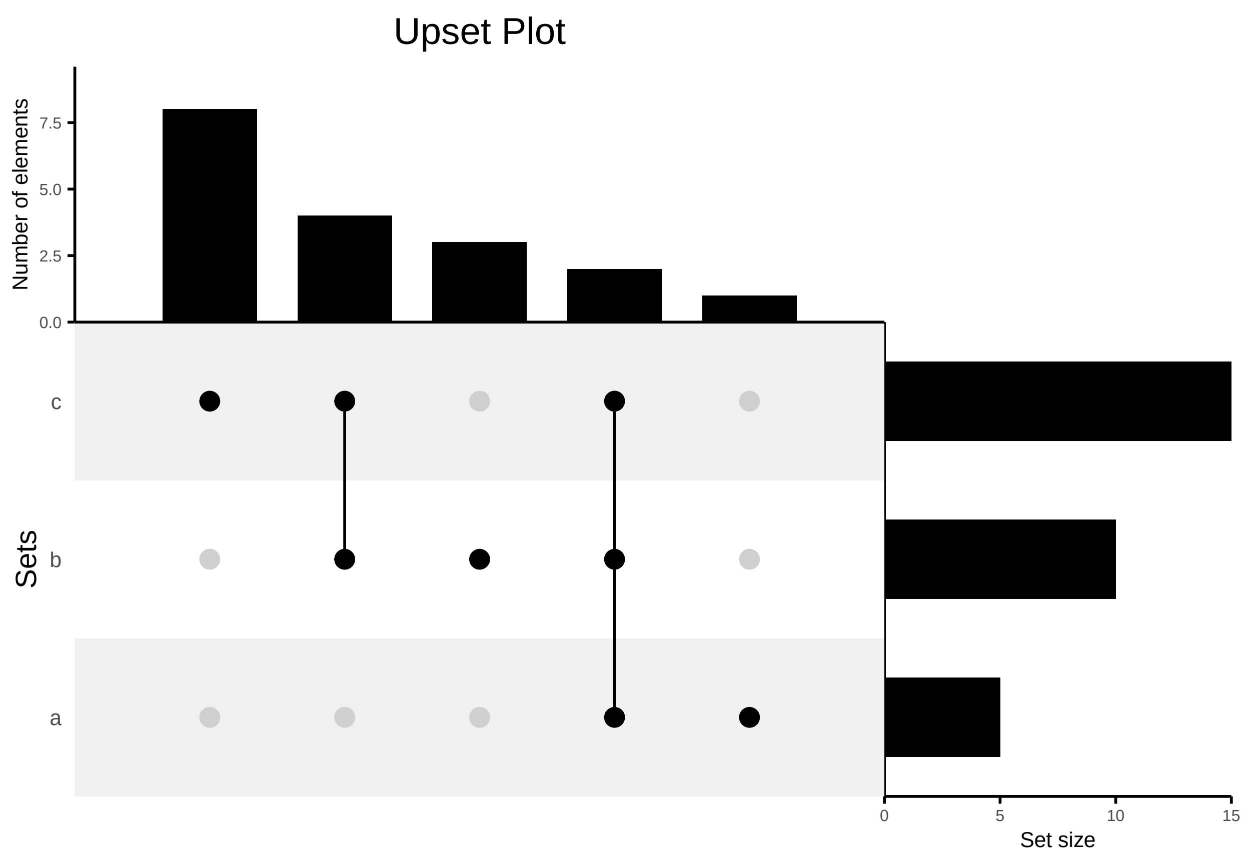
```
## Comb order
UpSetColor(data = lt, comb_order = 1:7)
```



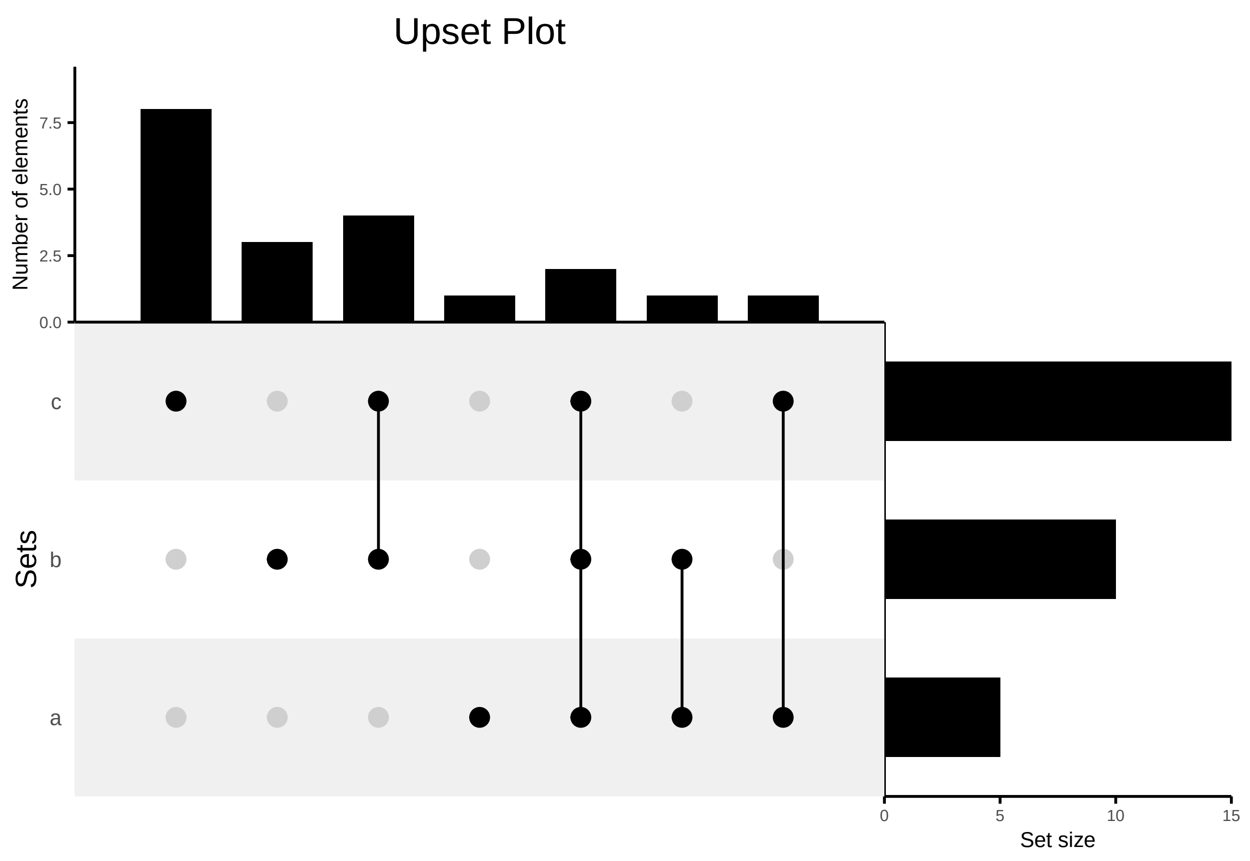
```r
UpSetColor(data = lt, comb_order = 7:1) # Sort combinations from the largest to the smallest.
```



```
UpSetColor(data = lt, top_comb = 5) # Show the 5 largest combinations
```



```
UpSetColor(data = lt, comb_order = c(1,3,2,5,4,6,7)) # Sort combinations manually
```

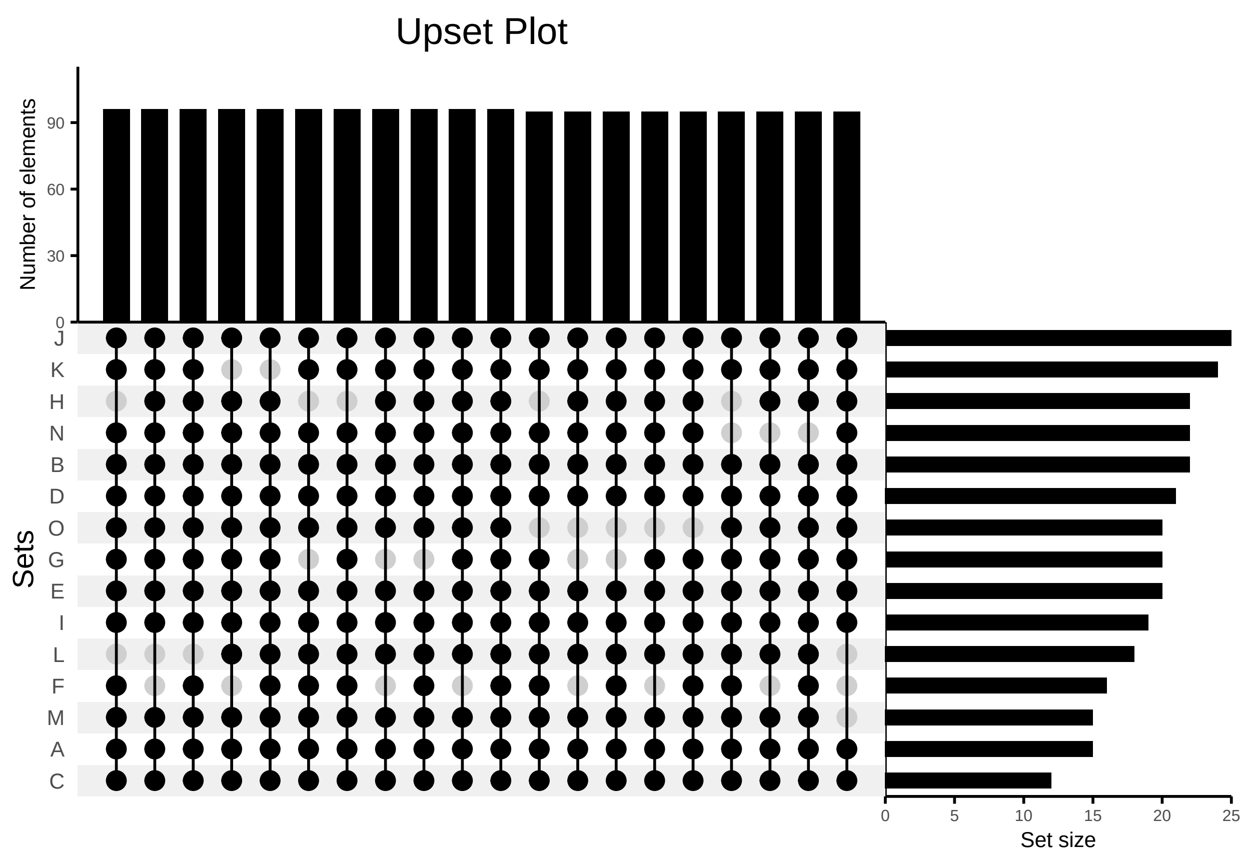


Sorting the data

Since the number of possible combinations is 2^(number of sets)-1, the time needed for comparing lots of sets (e.g. 20 sets) can be significant sometimes. In some cases, it may be advisable to visualize the progress of the analysis. This can be done by using *VERBOSE = TRUE*.

```
# With 15 sets
mm2 <- matrix(runif(1500, min=0, max = 1),nrow =100,ncol = 15)
mm2[mm2 <= 0.8] <- 0
mm2[mm2 > 0.8] <- 1
colnames(mm2) <- LETTERS[1:15]
UpSetColor(data = mm2, mode = 'union', verbose = TRUE)
```

```
##    |                                                                       |
```

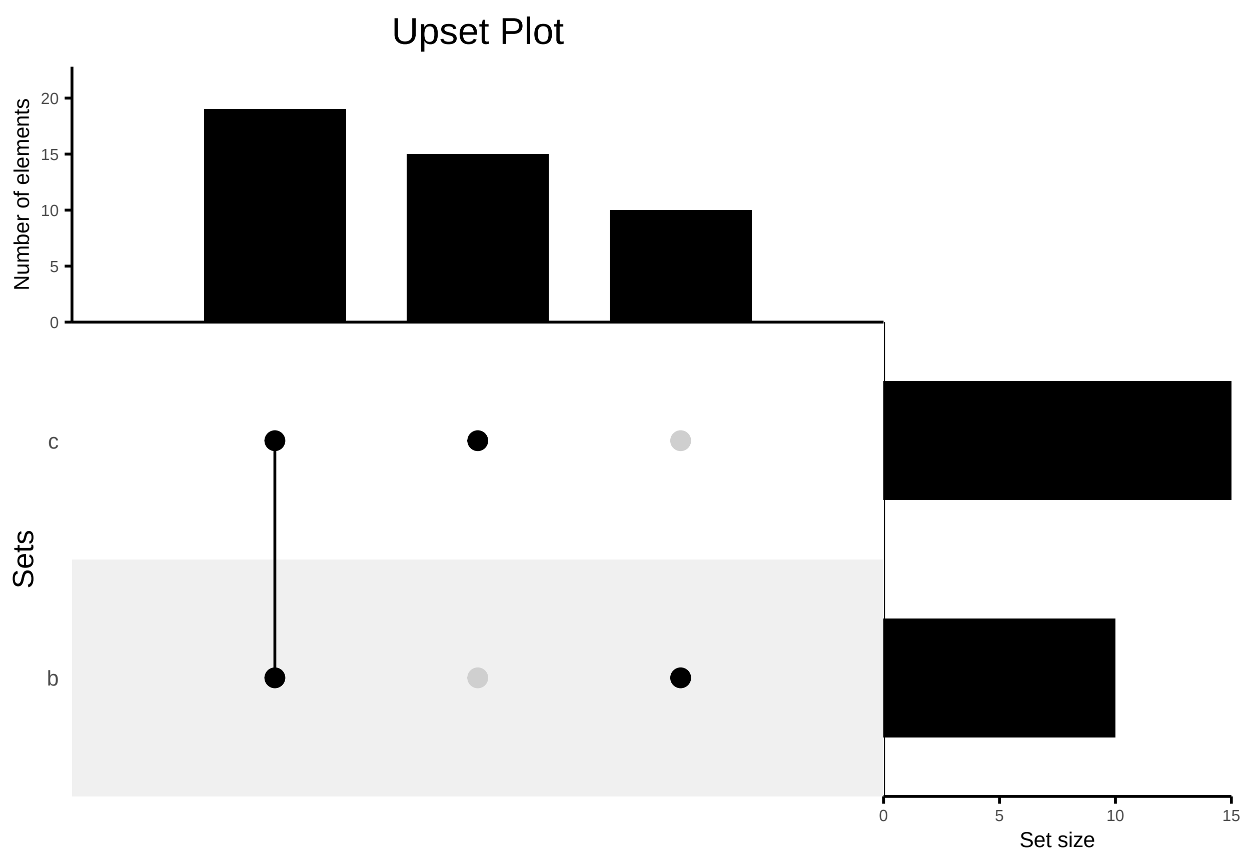


```
## Time difference of 37.36214 secs
```

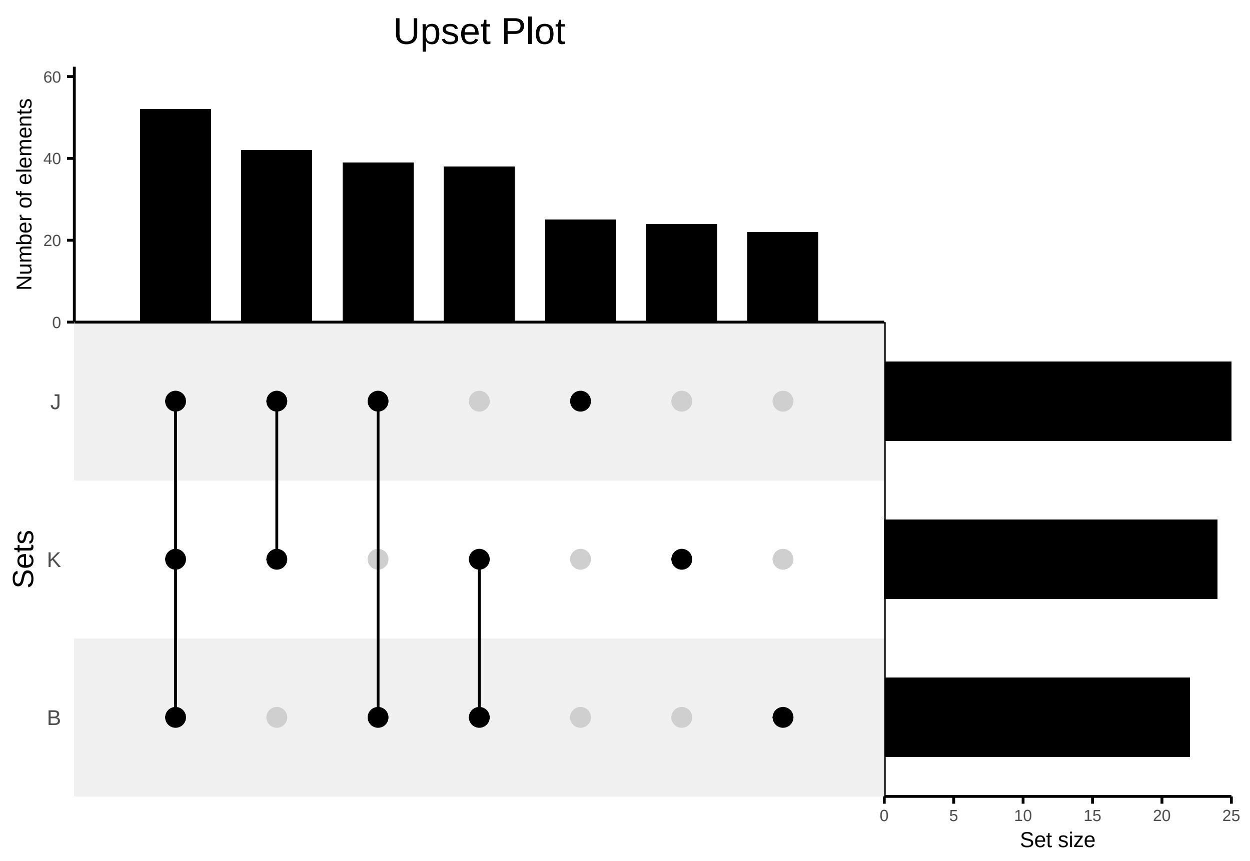
Filtering the data

We can limit the number of displayed sets with different parameters. Both *min_set_size* and *top_n_sets* evaluate the set sizes and can have implications on speeding up the computational time when comparing large lists of sets. For *min_comb_degree* and *max_comb_degree*, both filters operate after all the set combinations are calculated.

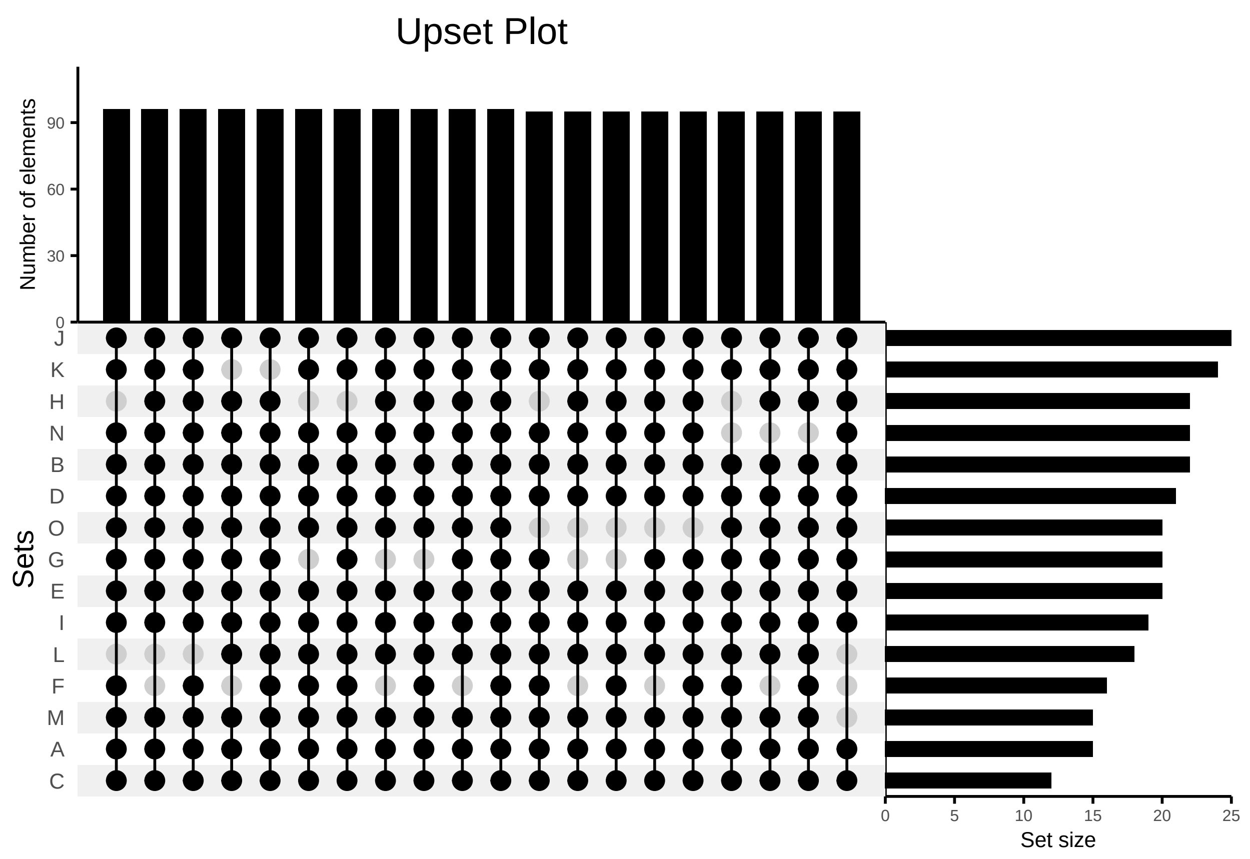
```
# Exclude sets with a small size
UpSetColor(data = lt, mode = 'union', min_set_size = 7)
```



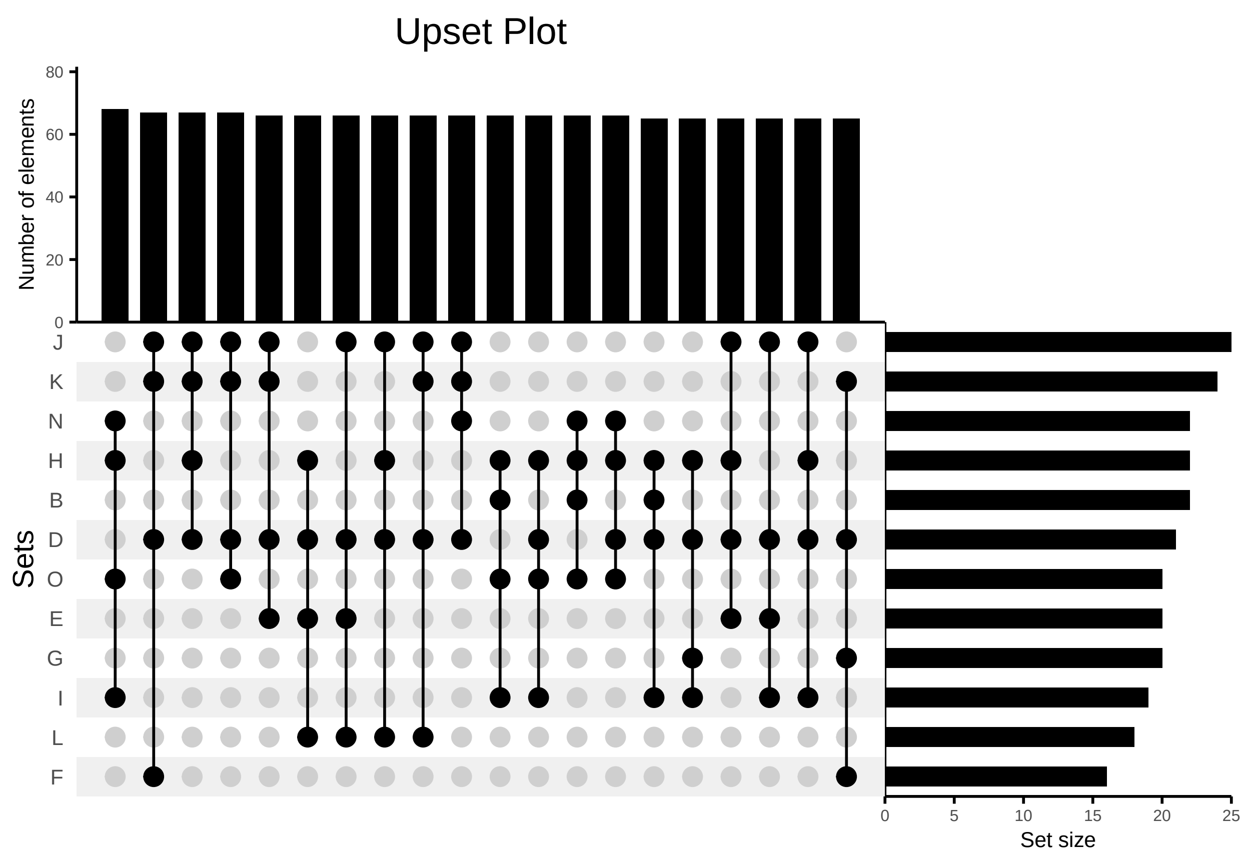
```r
# Estimate combinations with the 3 largest sets
UpSetColor(data = mm2, mode = 'union', top_n_sets = 3)
```



```
# Look at the combinations with involving at least 12 sets
UpSetColor(data = mm2, mode = 'union', min_comb_degree = 12)
```



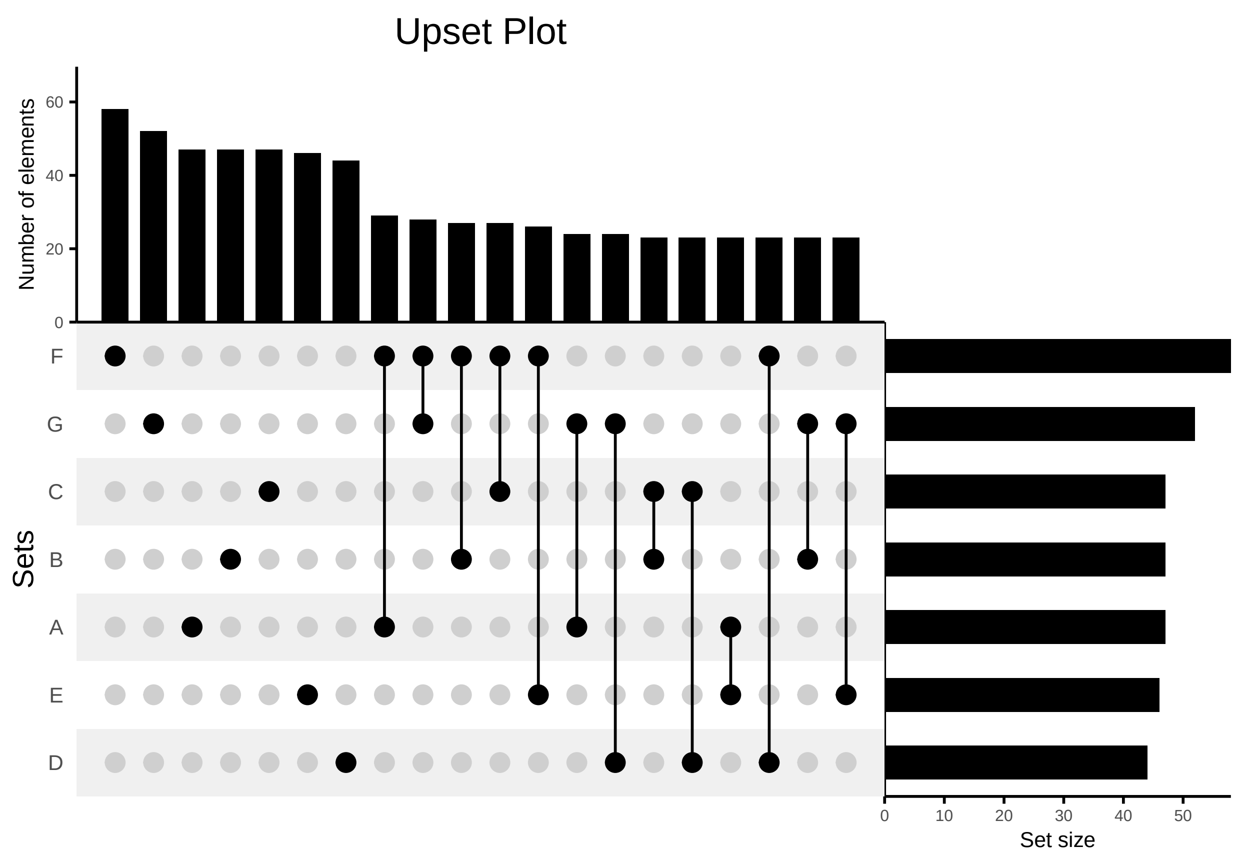
```
# Look at the combinations with involving at most 4 sets
UpSetColor(data = mm2, mode = 'union', max_comb_degree = 4)
```



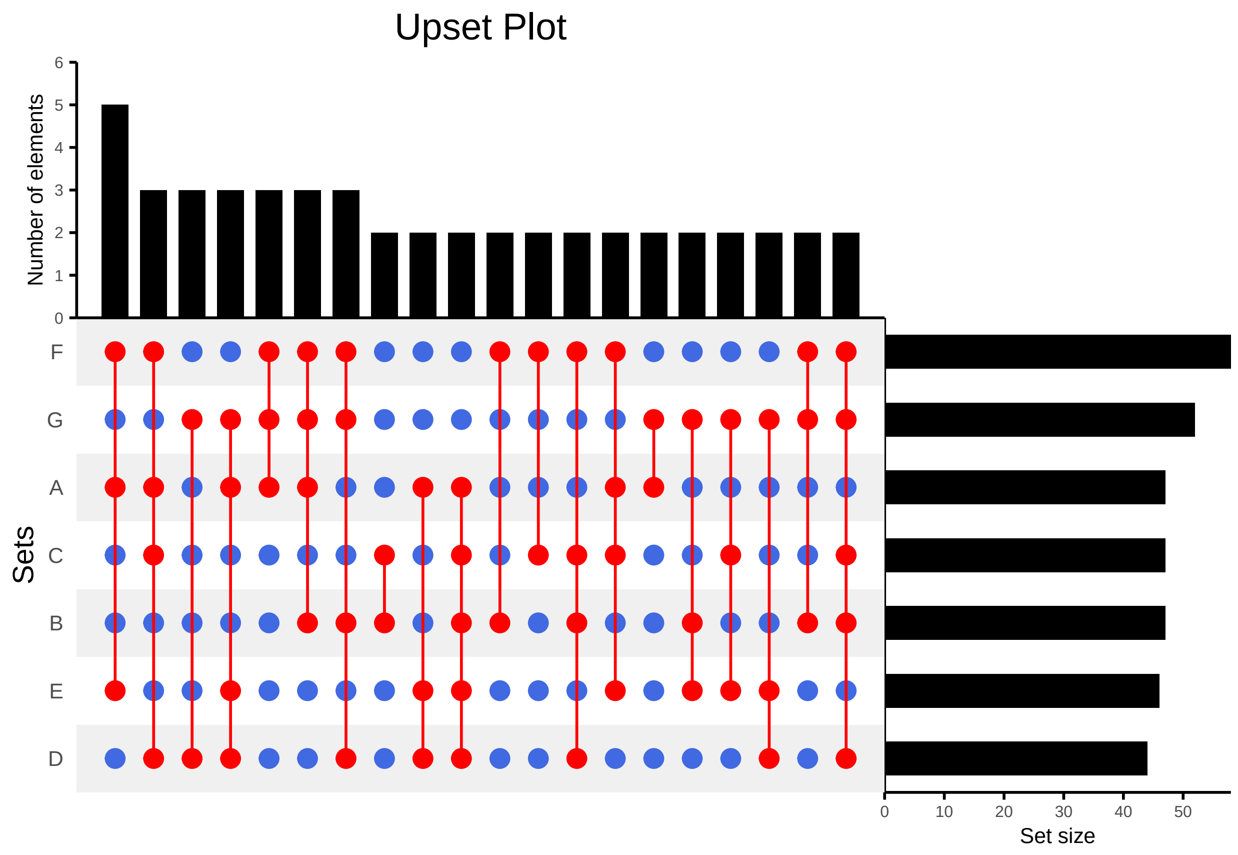
Coloring the plots

The arguments to use are *color.1* for the connected dots, *color.0* for the unconnected dots, *color.bar.comb* for the bars showing the set combinations, and *color.bar.set* for the bars showing the set sizes.

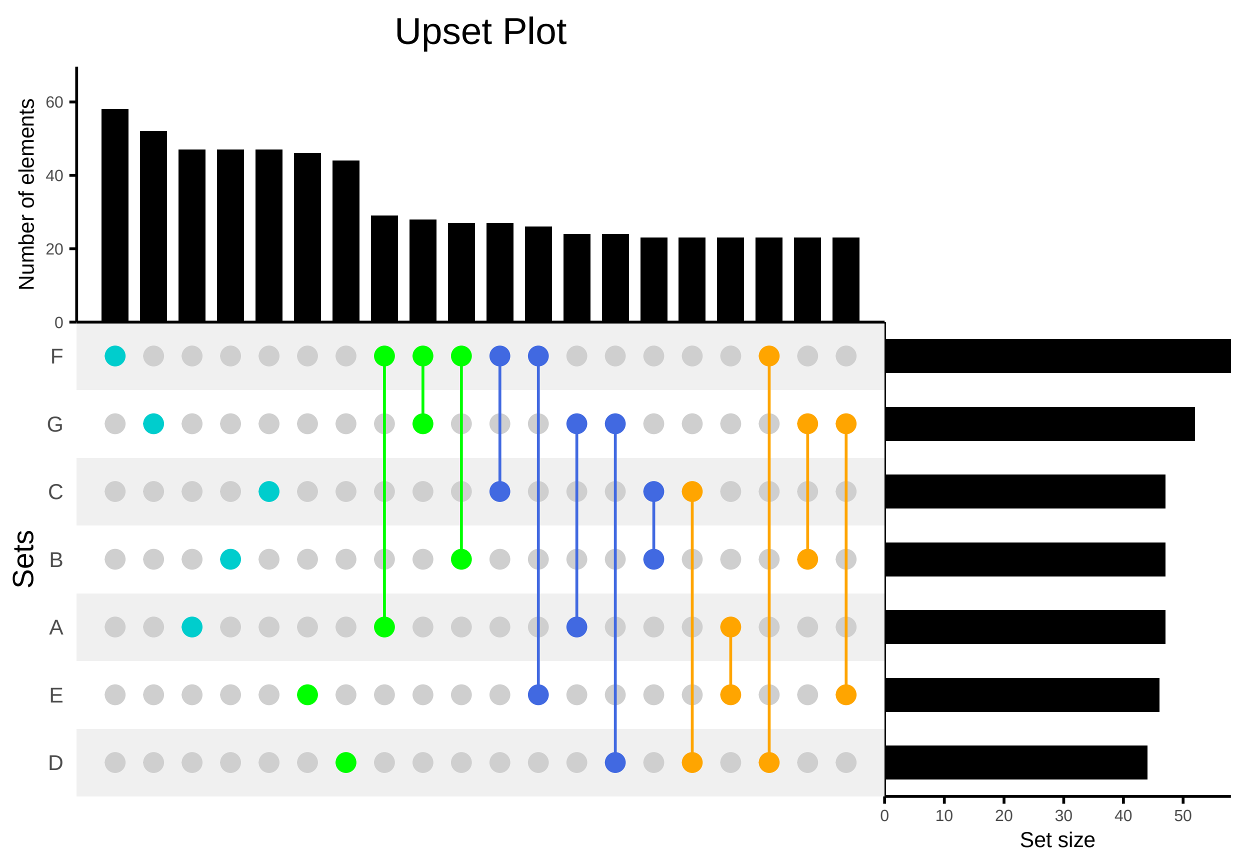
```
mm3 <- mm[,1:7]

# UpSet plot without color
UpSetColor(data = mm3, mode = 'intersect')
```



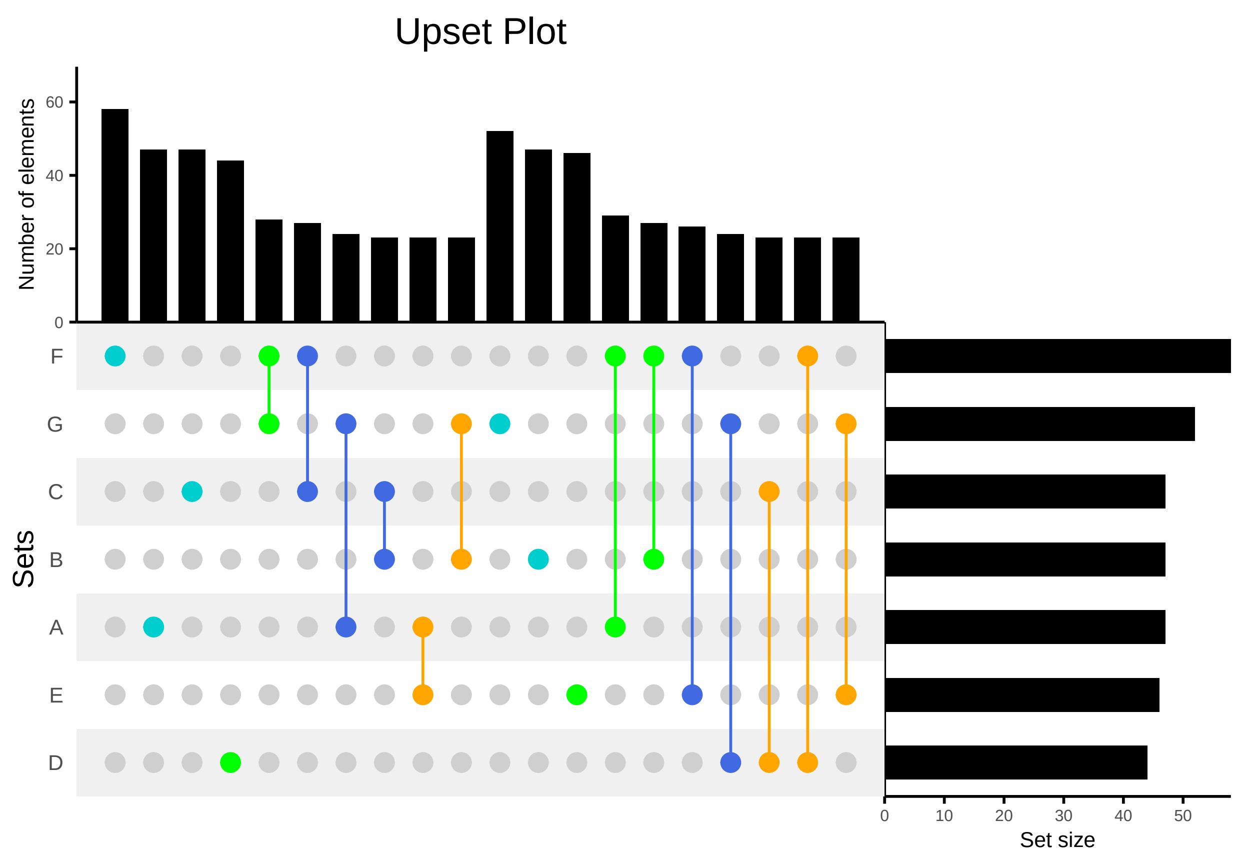
```
## Coloring the dots
# Simple edition of the dot colors
UpSetColor(data = mm3, color.1 = 'red', color.0 = 'royalblue')
```



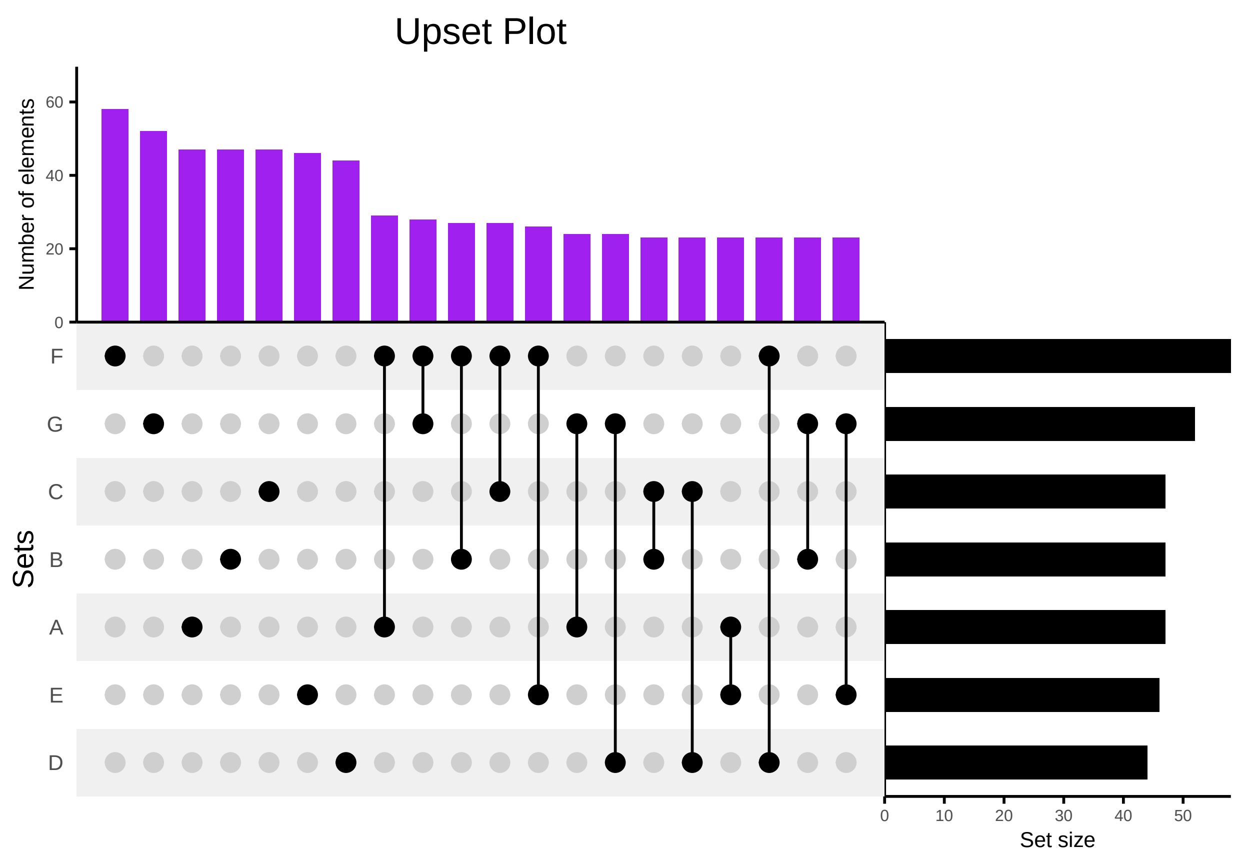
```r
# Dots are colored from left (so, from the largest combination of elements) to right
UpSetColor(data = mm3,
           mode = 'intersect',
           color.1 = rep(c('cyan3','green','royalblue','orange'), each = 5))
```



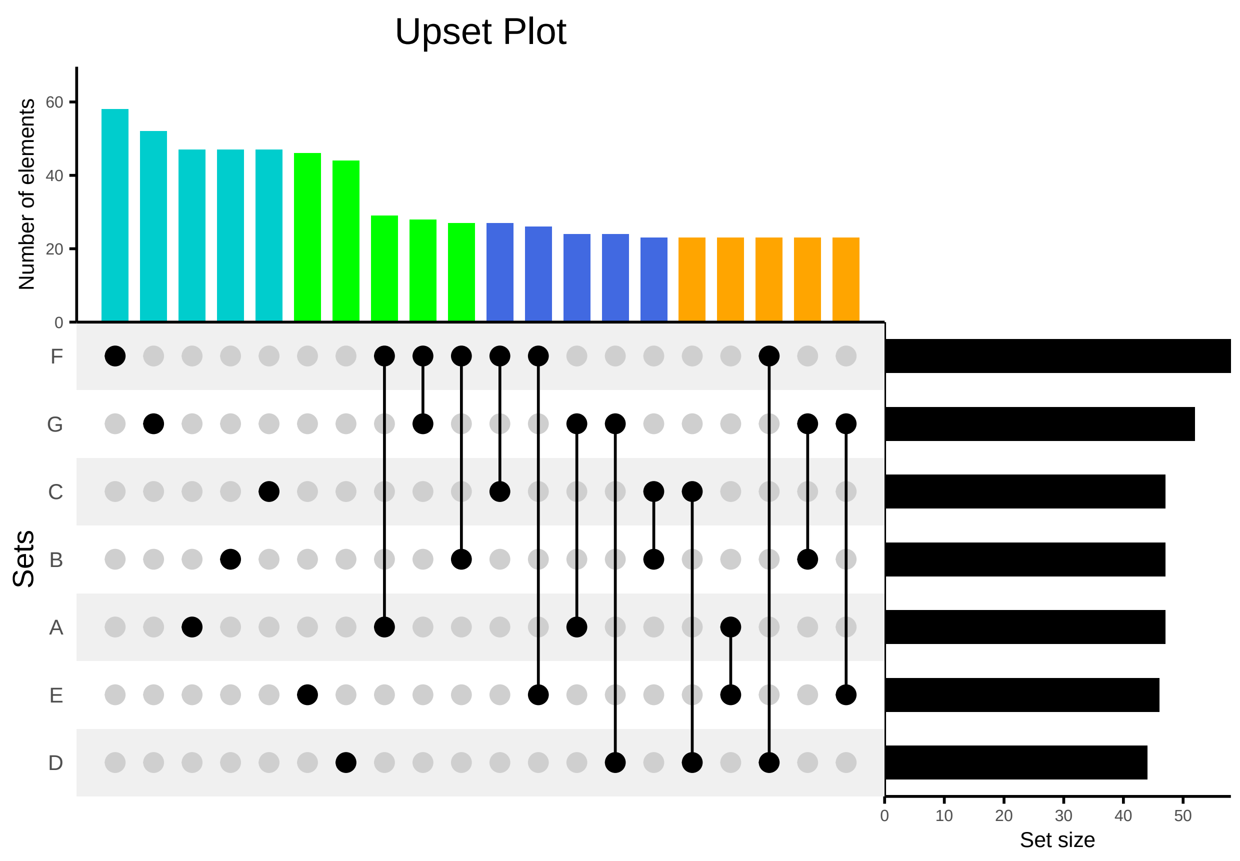
```r
# If comb_order is used, the same set combinations as before will be colored.
UpSetColor(data = mm3,
           comb_order = c(seq(1,20,2),seq(2,20,2)),
           mode = 'intersect',
           color.1 = rep(c('cyan3','green','royalblue','orange'), each = 5))
```



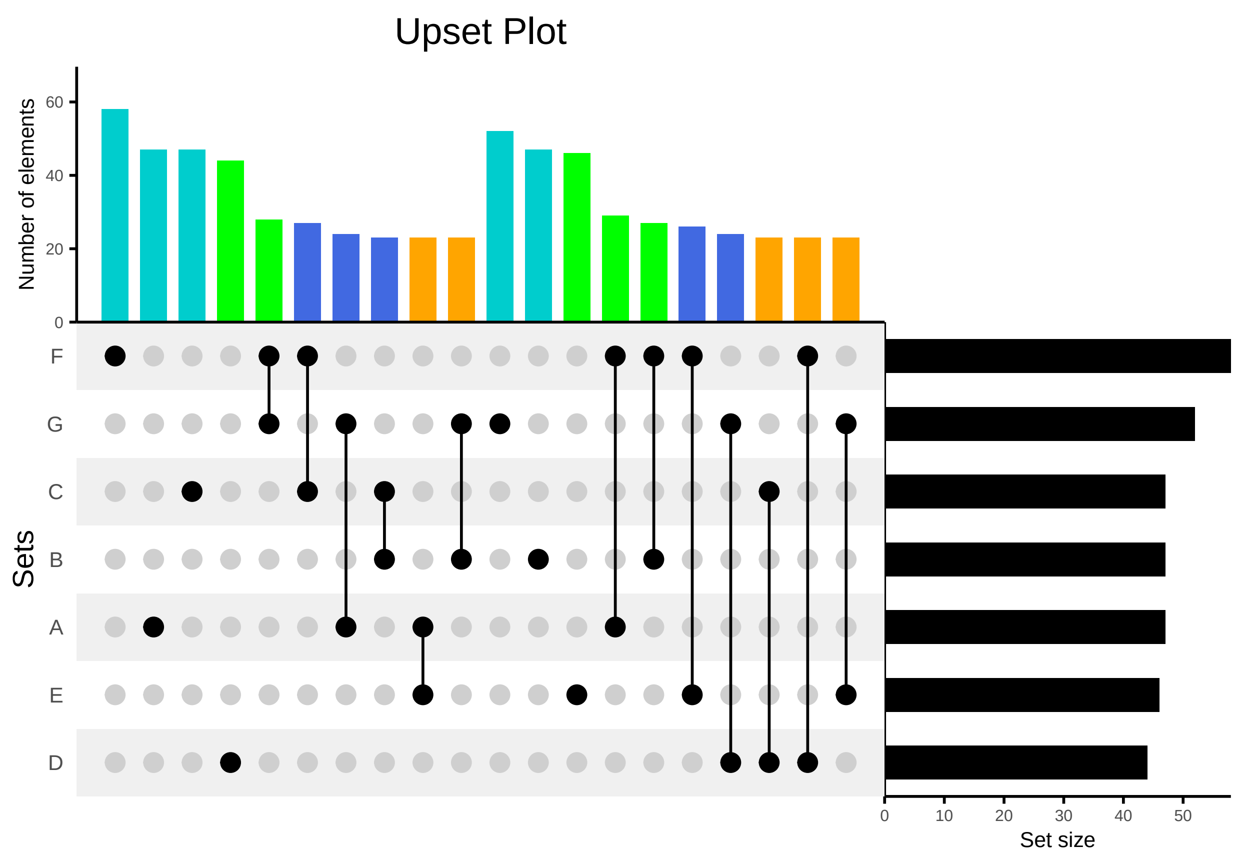
```
## Coloring the combination bars
UpSetColor(data = mm3,
           mode = 'intersect',
           color.bar.comb = 'purple')
```



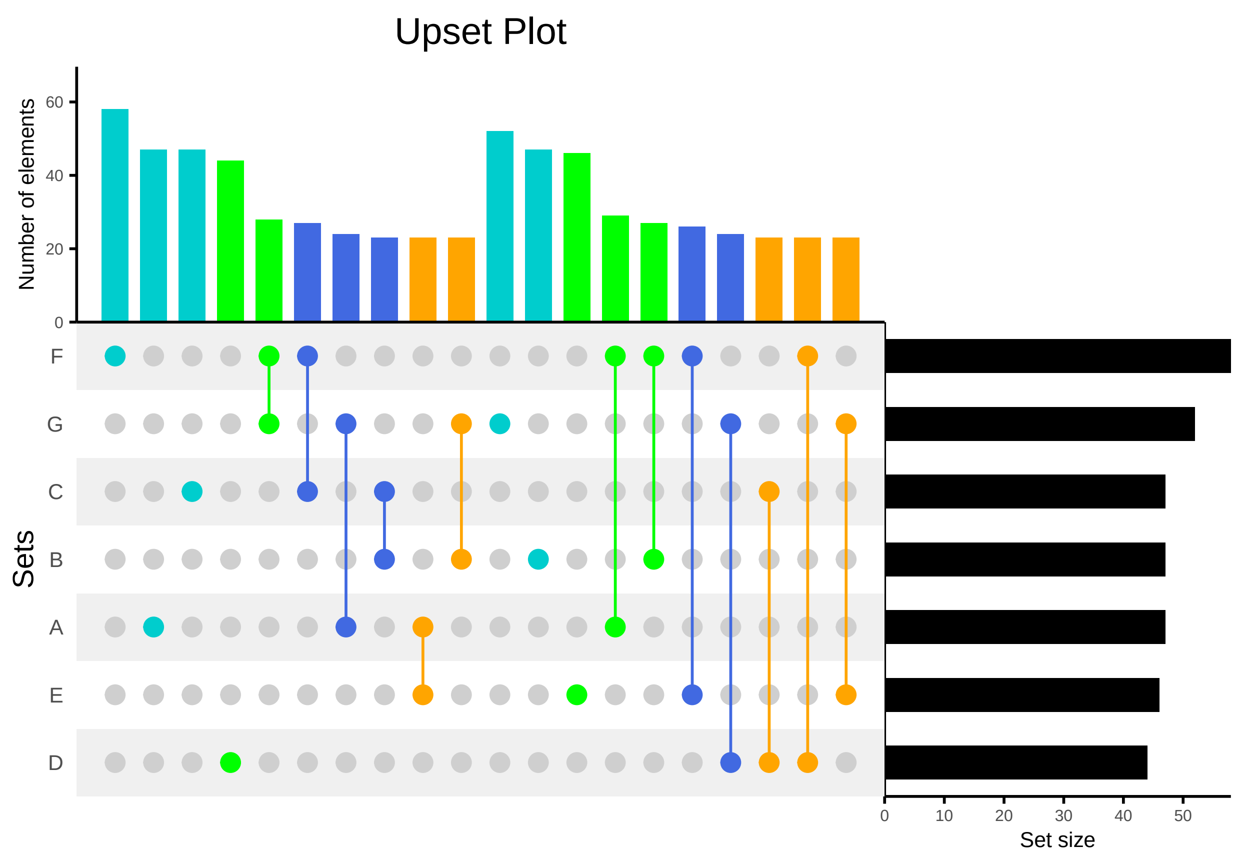
```
UpSetColor(data = mm3,
           mode = 'intersect',
           color.bar.comb = rep(c('cyan3','green','royalblue','orange'), each = 5))
```



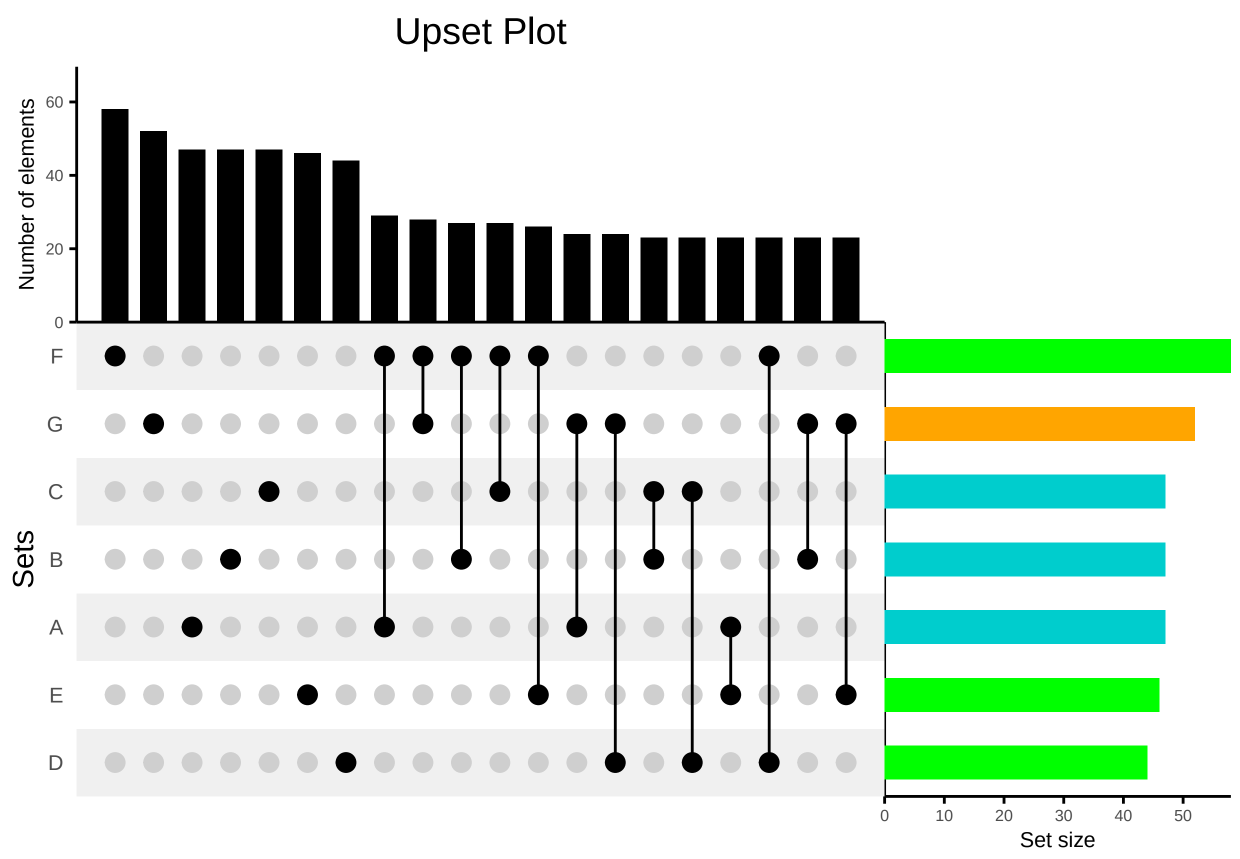
```r
# It also responds to comb_order
UpSetColor(data = mm3,
           comb_order = c(seq(1,20,2),seq(2,20,2)),
           mode = 'intersect',
           color.bar.comb = rep(c('cyan3','green','royalblue','orange'), each = 5))
```



```
## Coloring bars and dots simultaneously
UpSetColor(data = mm3,
           comb_order = c(seq(1,20,2),seq(2,20,2)),
           mode = 'intersect',
           color.bar.comb = rep(c('cyan3','green','royalblue','orange'), each = 5),
           color.1 = rep(c('cyan3','green','royalblue','orange'), each = 5))
```

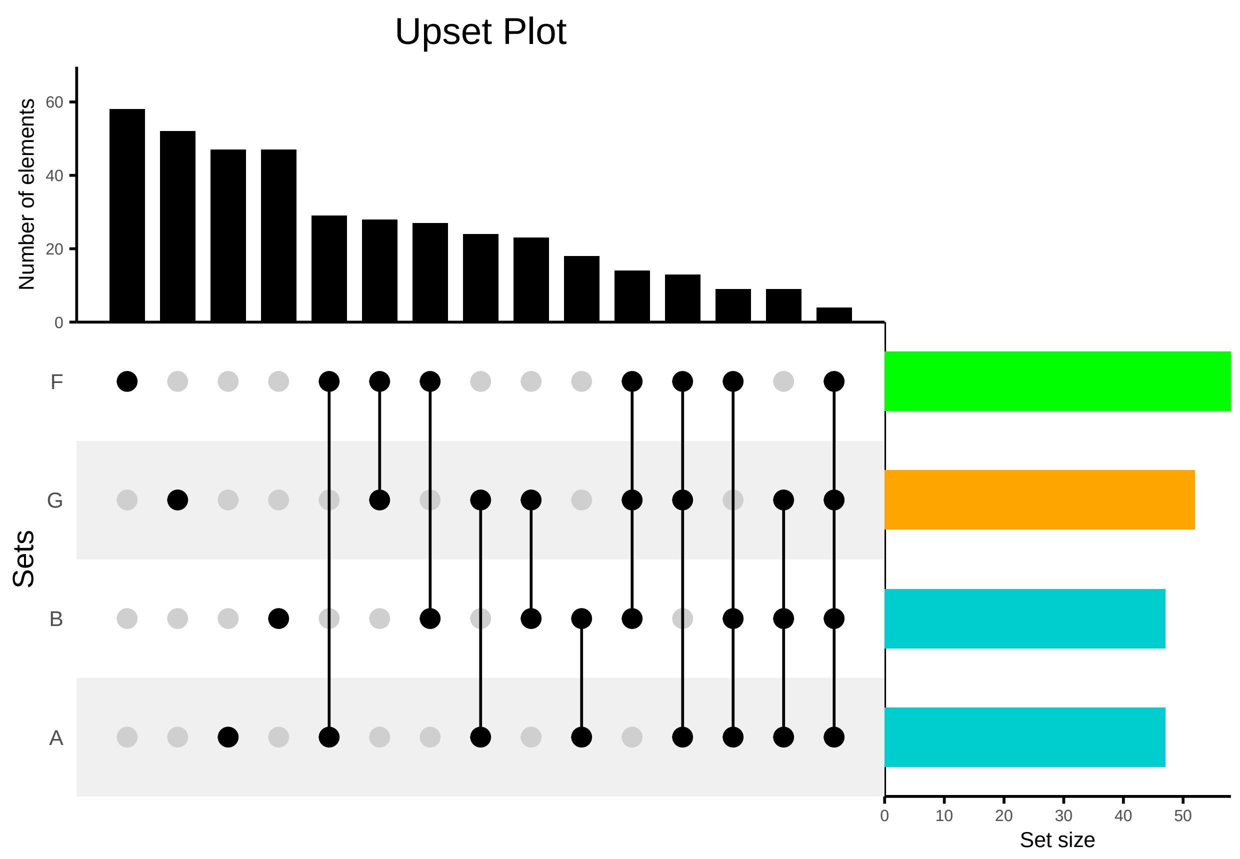


```
## Color sets
# To color the sets, the order is obtained from the initial sample list:
# A-C: cyan, D-F: green, G: orange
UpSetColor(data = mm3,
           mode = 'intersect',
           color.bar.sets = c('cyan3','cyan3','cyan3','green','green','green','orange'))
```



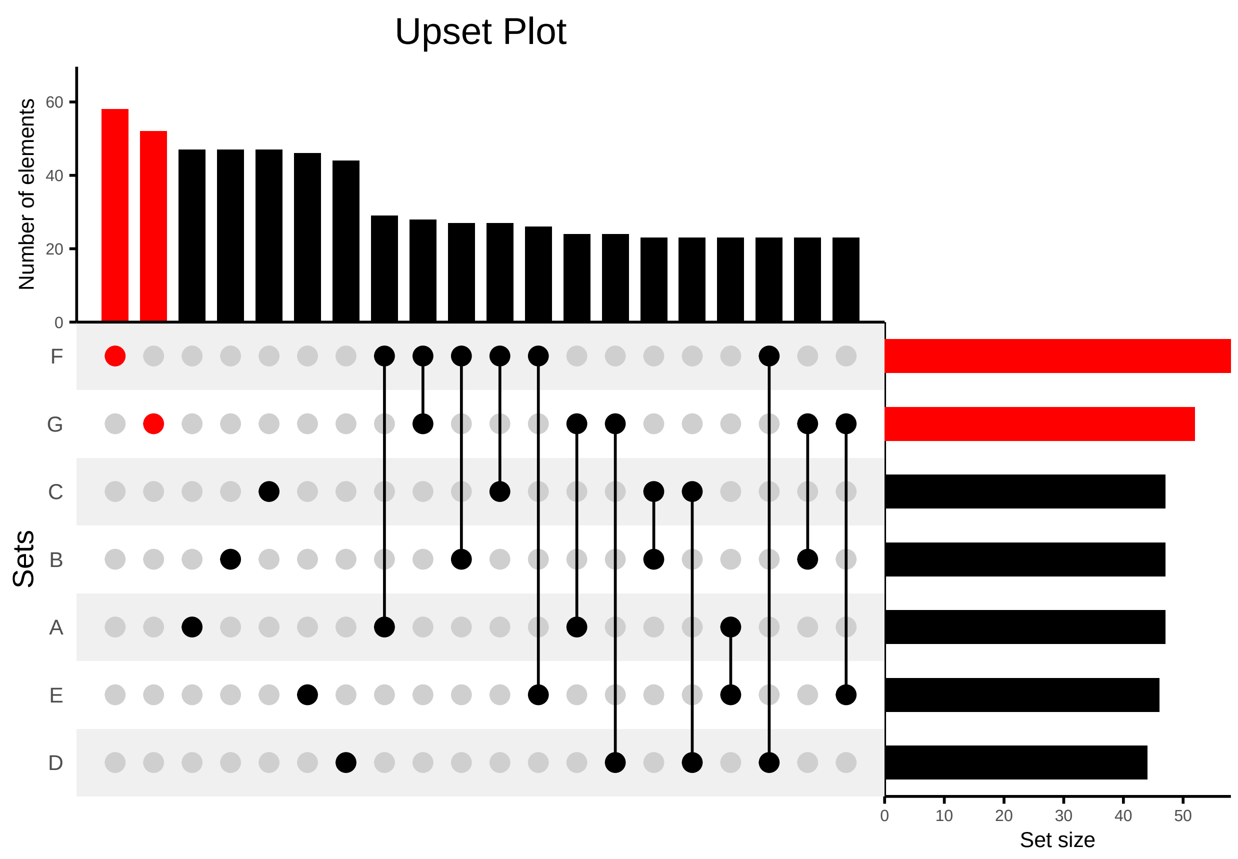
```
# The 4 sets displayed will have the same color as in the last plot.
UpSetColor(data = mm3,
           mode = 'intersect', top_n_sets = 4,
           color.bar.sets = c('cyan3','cyan3','cyan3','green','green','green','orange'))
```

```
## Warning in UpSetColor(data = mm3, mode = "intersect", top_n_sets = 4,
## color.bar.sets = c("cyan3", : There are less set combinations than 'top_comb'.
## All set combinations are shown
```

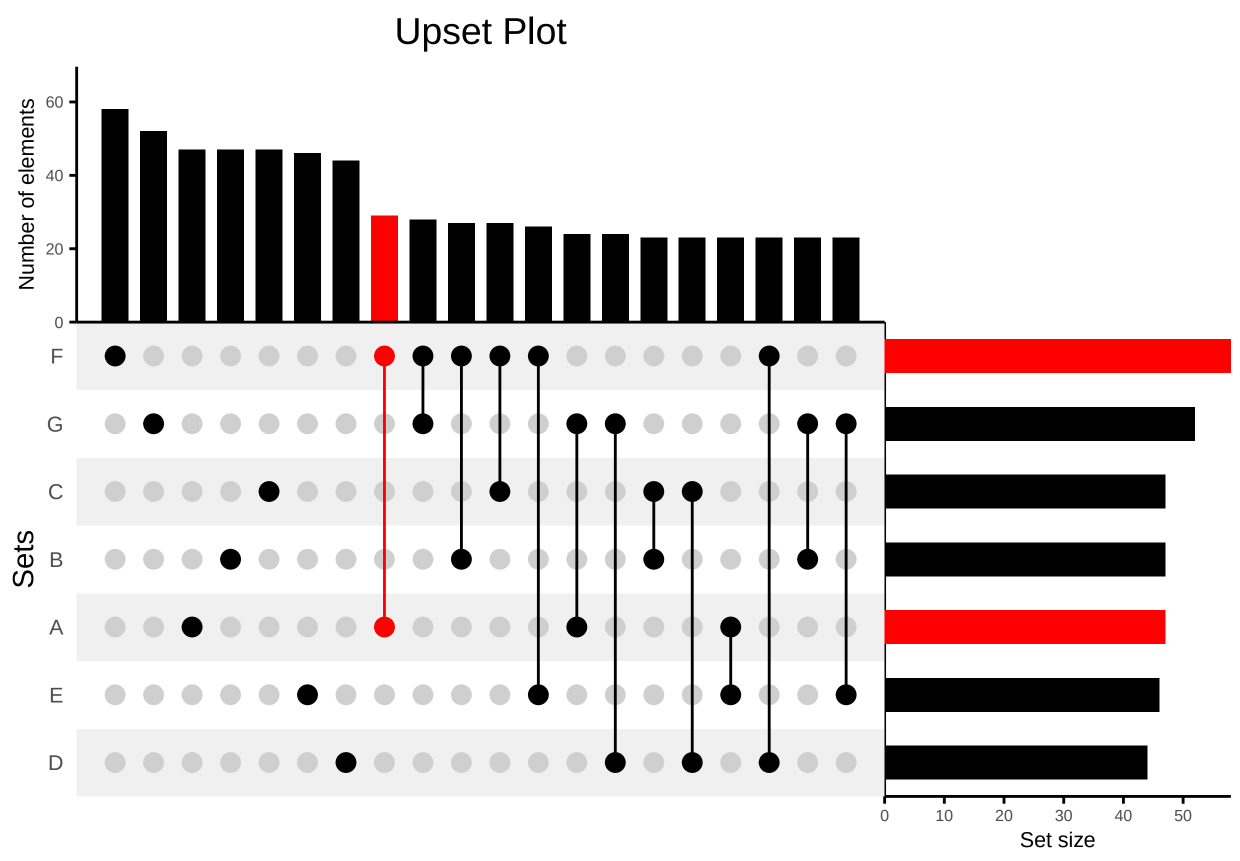


In addition, some connections can be highlighted with the *color.highlight* and *comb_highlight arguments*. *color.highlight* indicates the color used for highlighting, while *comb_highlight* is used to point the combination of sets to be highlighted.

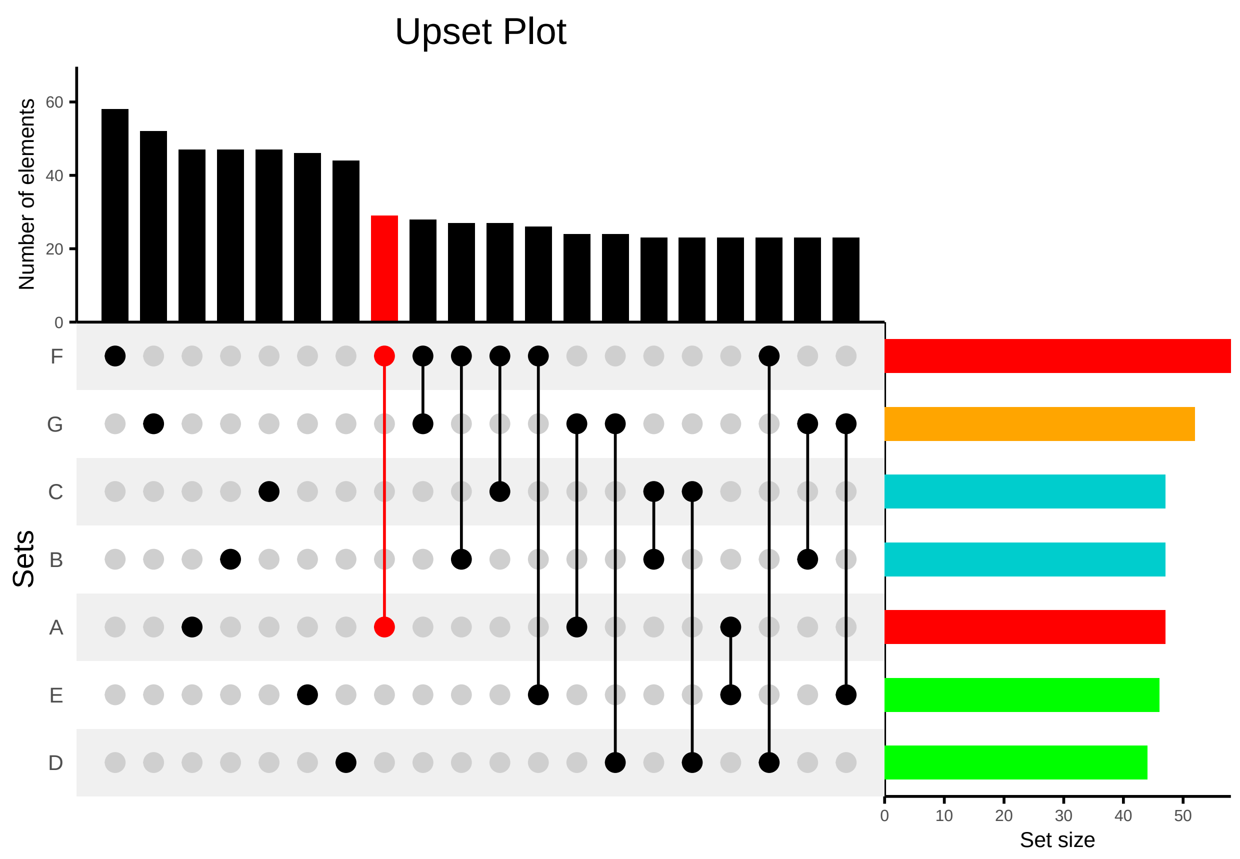
```
# Highlight (in the example, the 2 largest intersections)
UpSetColor(data = mm3,
           mode = 'intersect',
           color.highlight = 'red', comb_highlight = 1:2)
```



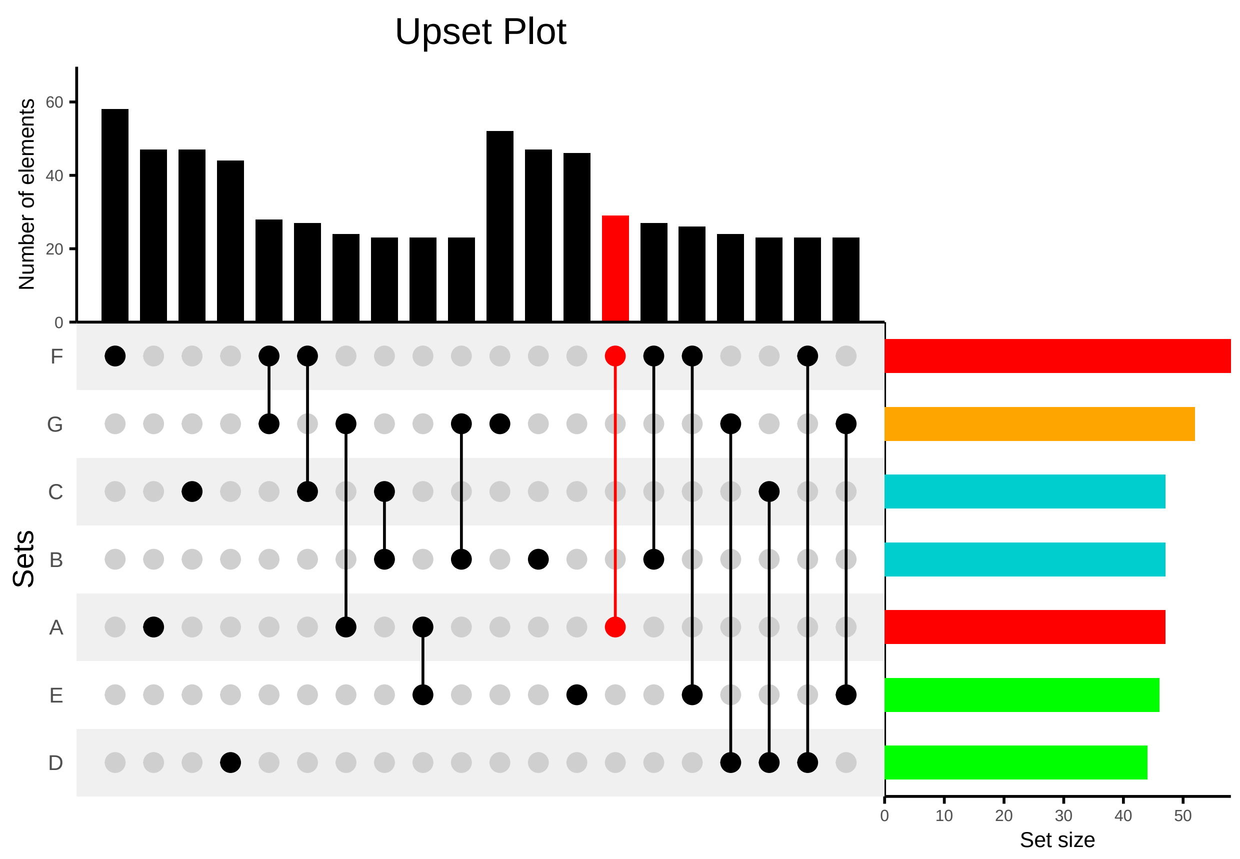
```r
# Highlight (in the example, the 8th largest intersection)
UpSetColor(data = mm3,
           mode = 'intersect',
           color.highlight = 'red', comb_highlight = 8)
```



```
# Notice that highlighted sets will be colored regardless of the information passed in the color.bar.se
UpSetColor(data = mm3,
           mode = 'intersect',
           color.highlight = 'red', comb_highlight = 8,
           color.bar.sets = c('cyan3','cyan3','cyan3','green','green','green','orange'))
```



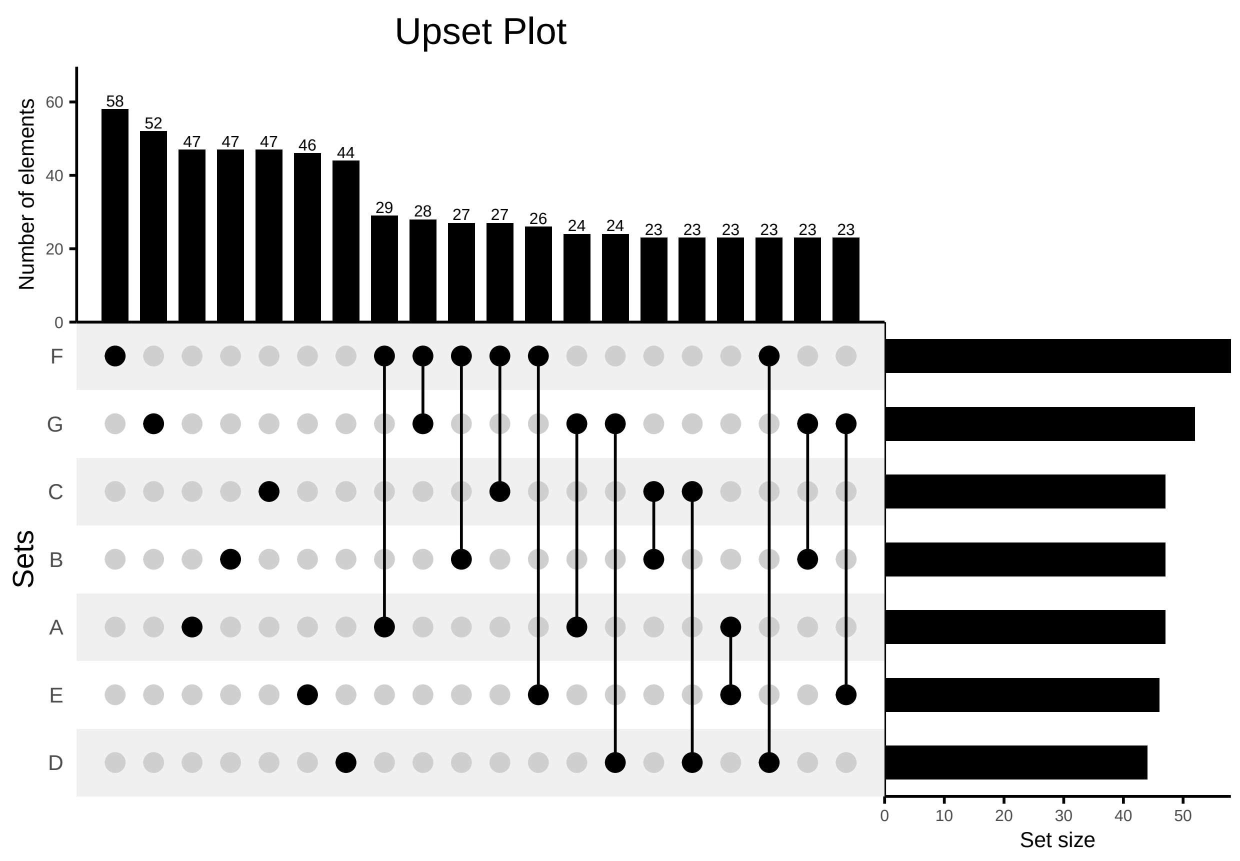
```
# Notice that highlighted sets will be colored regardless of the information passed in the color.bar.se
UpSetColor(data = mm3,
           mode = 'intersect',
           comb_order = c(seq(1,20,2),seq(2,20,2)),
           color.highlight = 'red', comb_highlight = 8,
           color.bar.sets = c('cyan3','cyan3','cyan3','green','green','green','orange'))
```



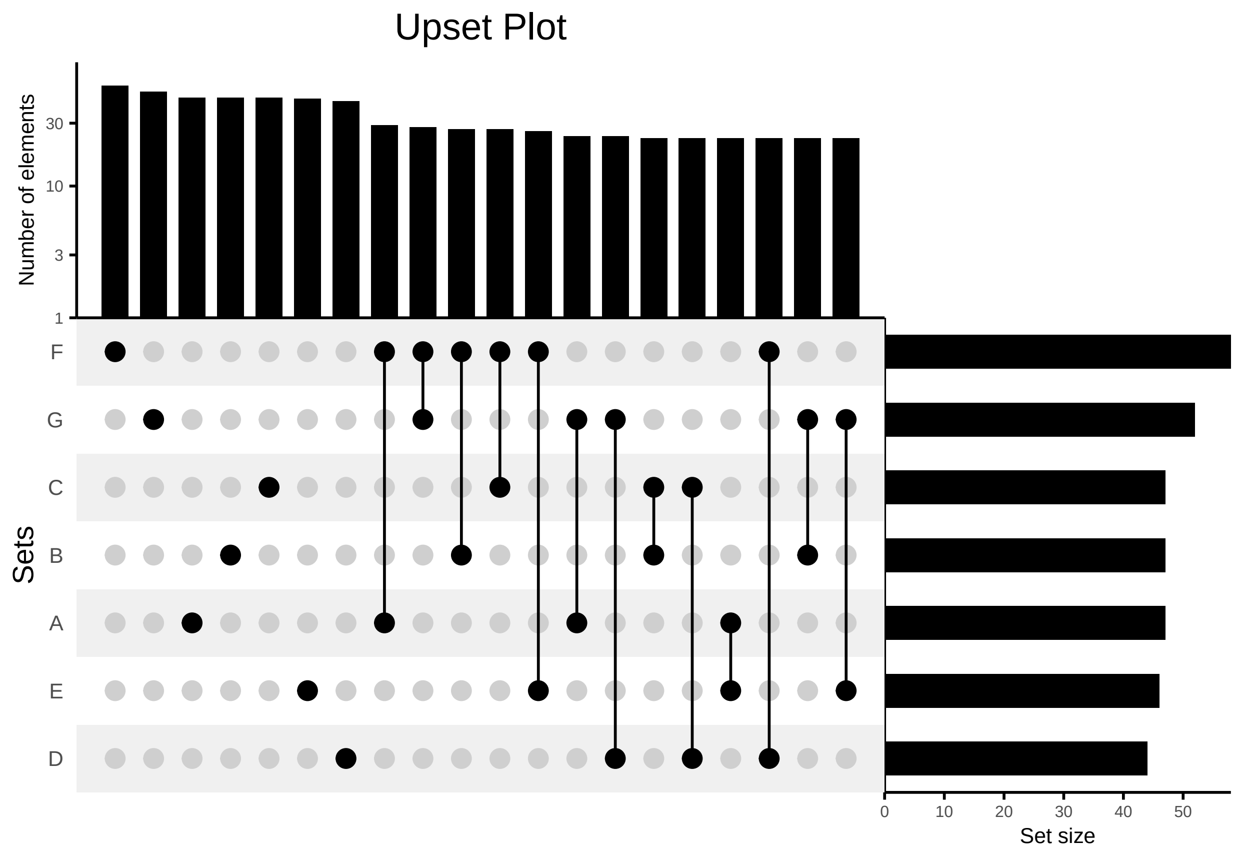
Other functionalities

It is possible to show labels with the number of elements in every combinations with the *numbers.size* argument. This argument controls the label font size. The combination plot can also be plotted in a log10-transformed y-axis.

```
# With labels
UpSetColor(data = mm3, numbers.size = 2, mode = 'intersect')
```



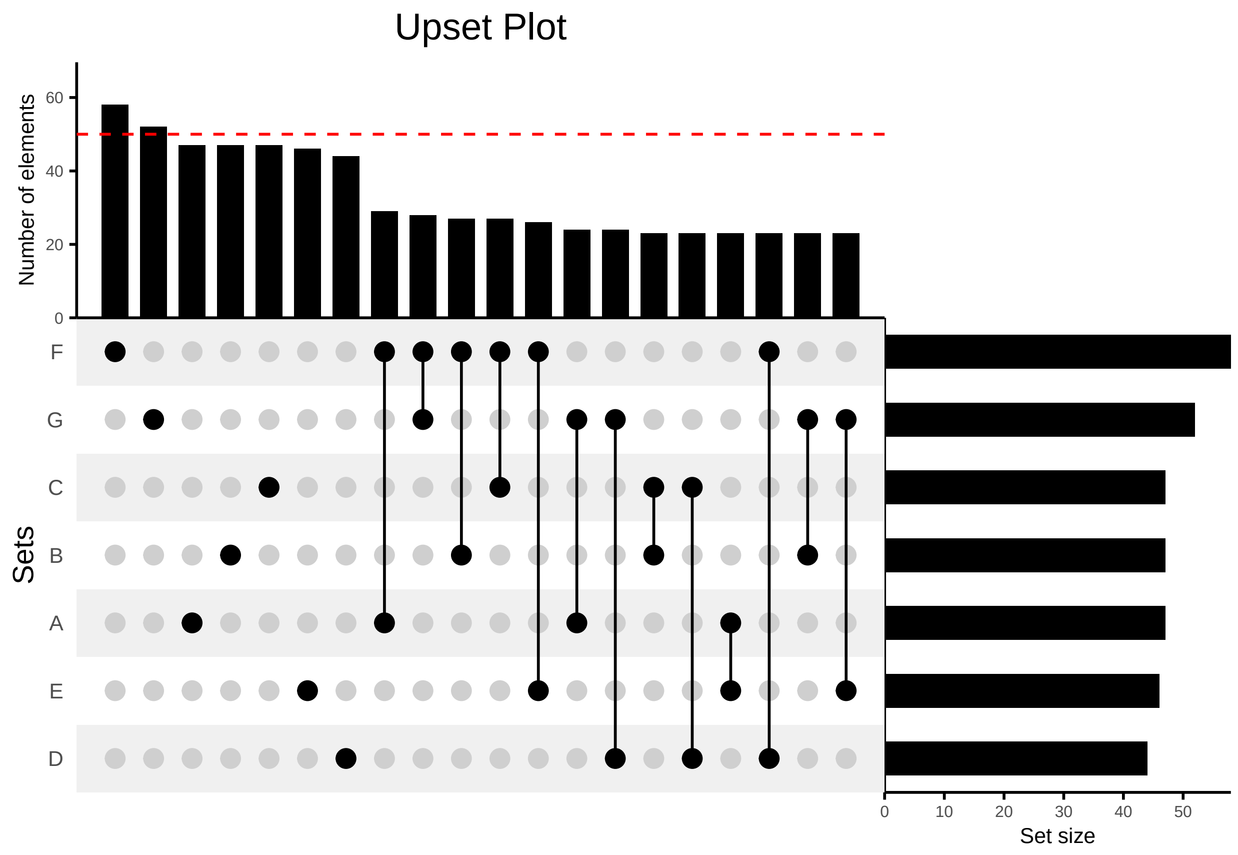
```
# log10 axis
UpSetColor(data = mm3, scale.Set.y.log10 = TRUE, mode = 'intersect')
```



Exporting data

The ggplot objects can be exported with *result = 'list.plot'*, allowing further customisations not available in the current package version. The data frames used for building each of the plots are accessible with *result = 'list.data.'*.

```
list.data <- UpSetColor(data = mm3, mode = 'intersect', result = 'list.data')
list.plot <- UpSetColor(data = mm3, mode = 'intersect', result = 'list.plot')

# Adding an horizontal red line
list.plot[['Comb']] <- list.plot[['Comb']] +
  ggplot2::geom_hline(yintercept=50, linetype="dashed", color = "red")

# Building the UpSet plot
egg::ggarrange(list.plot[['Comb']], list.plot[['Blank']],
               list.plot[['Matrix']], list.plot[['Set']],
               ncol = 2, nrow = 2,
               heights = c(0.35, 0.65), widths = c(0.7,0.3))
```


Upset Plot
Number of elements
0
20
40
60
F
G
C
B
A
E
D
Sets
0
10
20
30
40
50
Set size