# post02

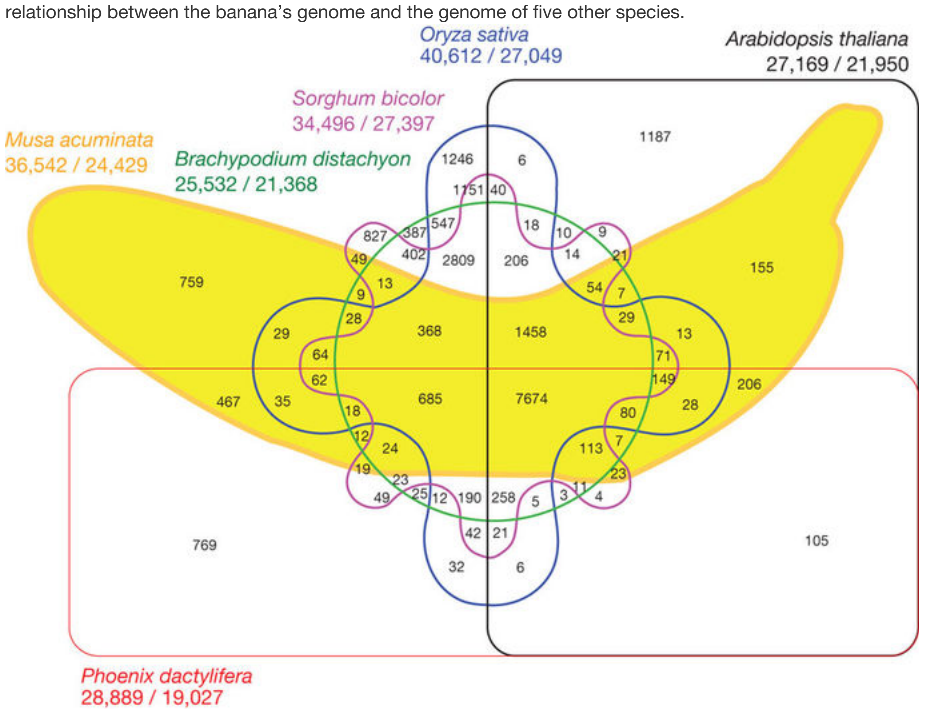
*Rui Sun*

*2017/12/2*

# UpSet: Visualization of Intersecting Sets

## Introduction

Venn diagram is the most common way to visualize intersections. However, when there are more than three or four sets, Venn diagrams are a horrible way to visualize intersections. The figure below shows an example of a six-set venn diagram published in Nature that shows the relationship between the banana's genome and the genome of five other species.



Even though the diagram is delicate, it is hard for people to get information from observing the diagram.

To address this, we introduce UpSet, a novel visualization technique for the quantitative analysis of sets, their intersections, and aggregates of intersections.

## Installation

2 ways to install "UpSet" package

```r
# from CRAN
# install.packages("UpSet.R")
# from Github
devtools::install_github("hms-dbmi/UpSetR")
```

```
## Skipping install of 'UpSetR' from a github remote, the SHA1 (67d3ea9b) has not changed since last install.
##   Use `force = TRUE` to force installation
```

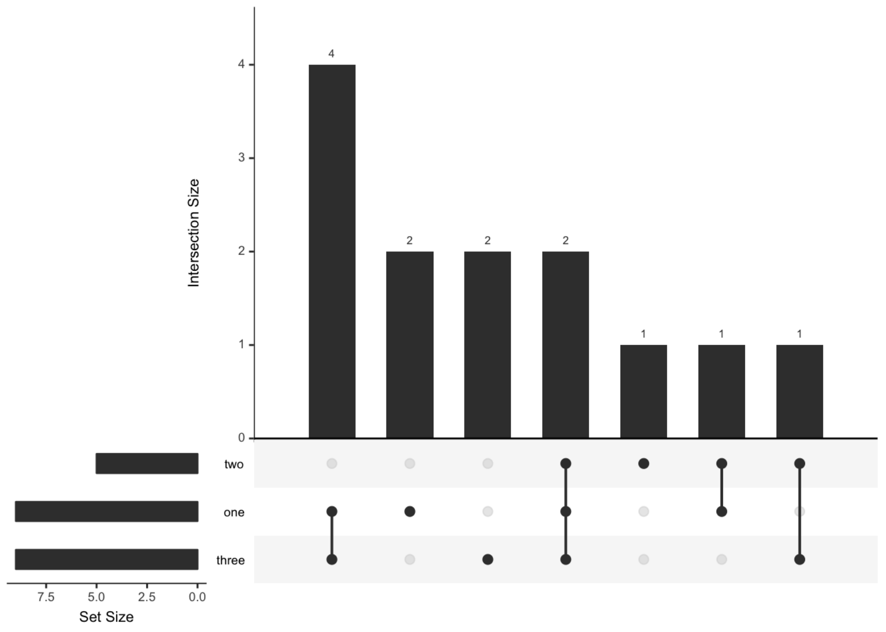
## data importing

UpSetR has two functions: `fromList` `fromExpression`. These functions can convert data to a data form that is applicable to UpsetR.

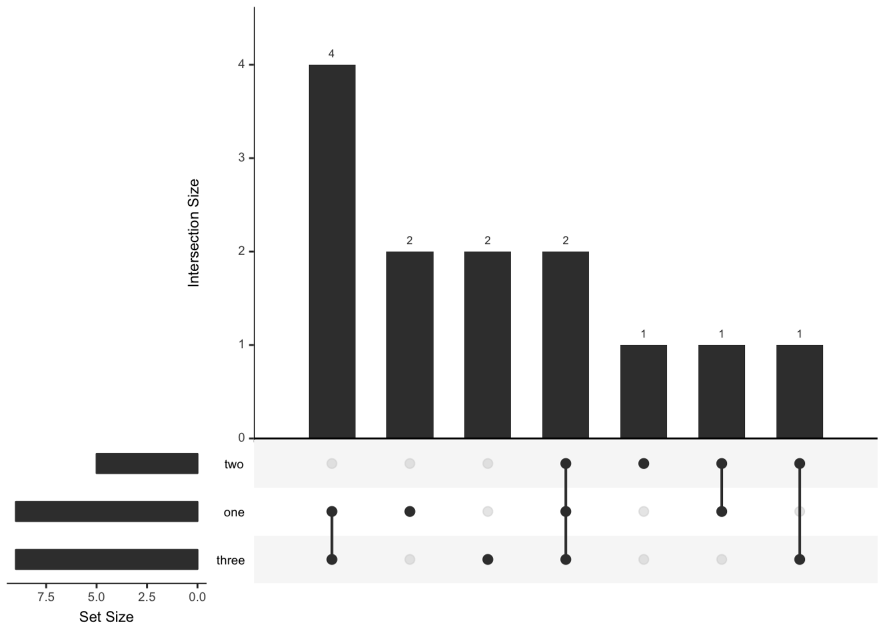
```r
# fromList
listinput <- list(one = c(1, 2, 3, 5, 7, 8, 11, 12, 13), two = c(1, 2, 4, 5,
    10), three = c(1, 5, 6, 7, 8, 9, 10, 12, 13))
#fromExpression
expressionInput <- c(one = 2, two = 1, three = 2, `one&two` = 1, `one&three` = 4,
    `two&three` = 1, `one&two&three` = 2)
```

Then we are able to draw diagram by UpSetR

```r
library(UpSetR)
upset(fromList(listinput), order.by = "freq")
```



```r
# the output is same
upset(fromExpression(expressionInput), order.by = "freq")
```



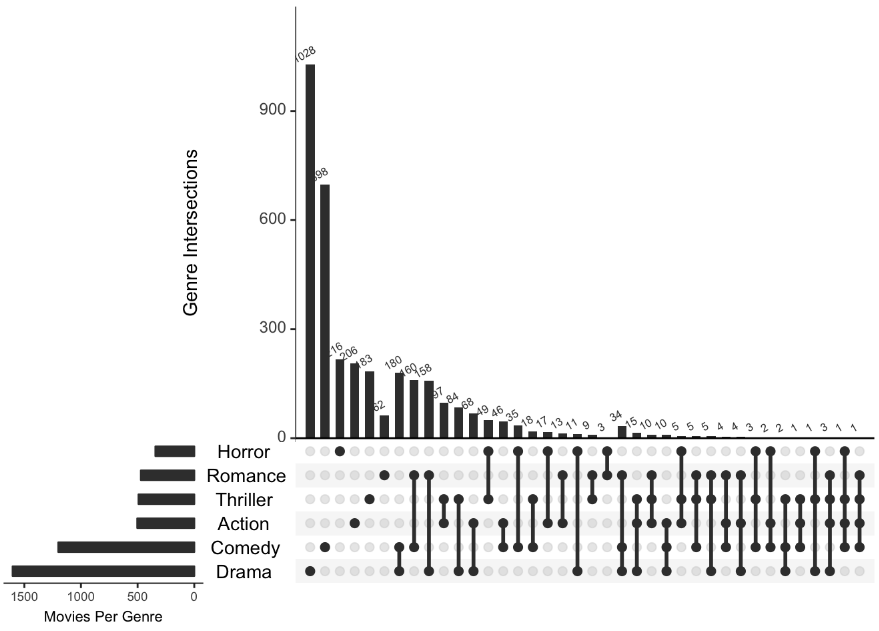
## Example

```r
# import data
movies <- read.csv(system.file("extdata", "movies.csv", package = "UpSetR"), header = TRUE, sep = ";")
# since the data is too big we only need first few columns
knitr::kable(head(movies[,1:10]))
```

| Name | ReleaseDate | Action | Adventure | Children | Comedy | Crime | Documentary | Drama | Fantasy |
|---|---|---|---|---|---|---|---|---|---|
| Toy Story (1995) | 1995 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| Jumanji (1995) | 1995 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| Grumpier Old Men (1995) | 1995 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| Waiting to Exhale (1995) | 1995 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| Father of the Bride Part II (1995) | 1995 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| Heat (1995) | 1995 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| The above data set shows Name, Realse | Data and Type | of movies | . | | | | | | |

UpsetR draws a collection of visual graphics using function `upset()`

```r
upset(movies, nsets = 6, number.angles = 30, point.size = 2, line.size = 1, mainbar.y.label = "Genre Intersections
", sets.x.label = "Movies Per Genre", text.scale = c(1.3, 1.3, 1, 1, 1.5, 1))
```

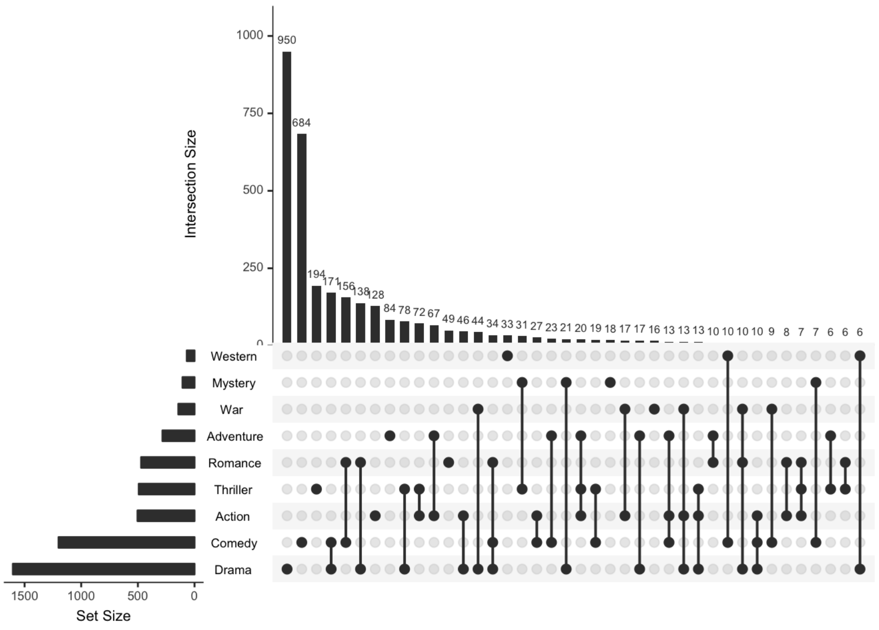


## Explaination of above arguments

- `nests` : Number of sets to look at
- `number.angles` : The angle of the numbers atop the intersection size bars
- `mainbar.y.label` : The y-axis label of the intersection size bar plot
- `sets.x.label` : The x-axis label of the set size bar plot
- `text.scale` : Numeric, value to scale the text sizes, applies to all axis labels, tick labels, and numbers above bar plot. Can be a universal scale, or a vector containing individual scales in the following format: c(intersection size title, intersection size tick labels, set size title, set size tick labels, set names, numbers above bars)

Most of time, we need to specify data sets.

```r
upset(movies, sets = c("Action", "Adventure", "Comedy", "Drama", "Mystery",
    "Thriller", "Romance", "War", "Western"), mb.ratio = c(0.55, 0.45), order.by = "freq")
```

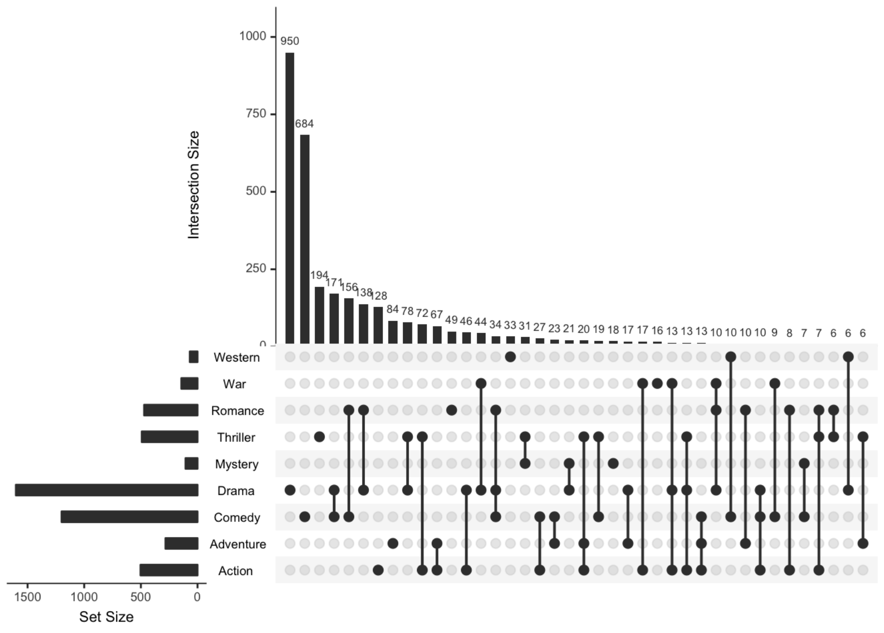


## Explaination of above arguments

- `mb.ratio` : Ratio between matrix plot and main bar plot (Keep in terms of hundreths)
- `order.by` : How the intersections in the matrix should be ordered by. Options include frequency (entered as "freq"), degree, or both in any order.

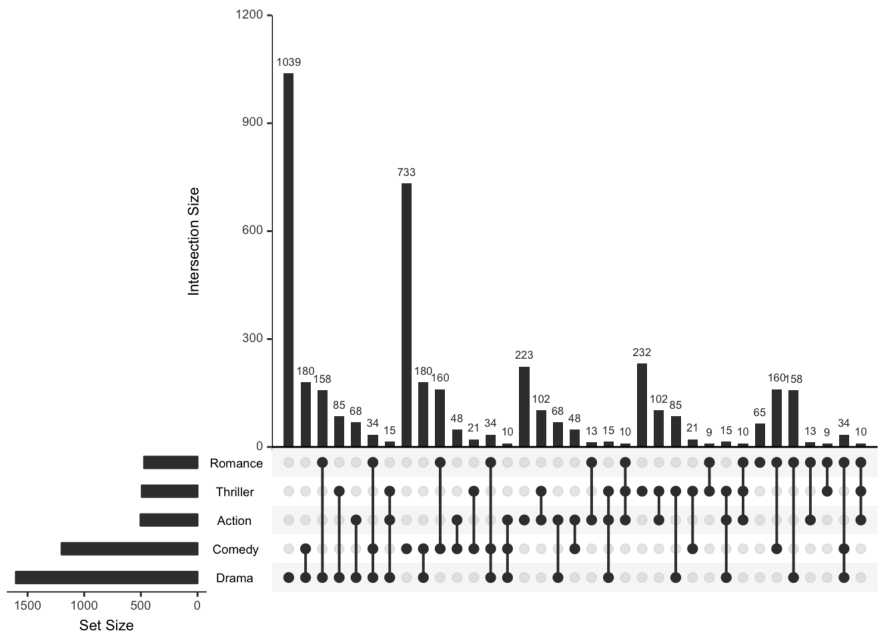
To keep the sets in the order entered using the `sets` parameter, set the keep.order parameter to TRUE.

```r
upset(movies, sets = c("Action", "Adventure", "Comedy", "Drama", "Mystery",
    "Thriller", "Romance", "War", "Western"), mb.ratio = c(0.55, 0.45), order.by = "freq",
    keep.order = TRUE)
```



Instead of the default method of grouping by degree, grouping by sets can be acheived using `group.by`. To set a cutoff for the number of intersections per group of sets use `cutoff`.

```
upset(movies, nintersects = 70, group.by = "sets", cutoff = 7)
```



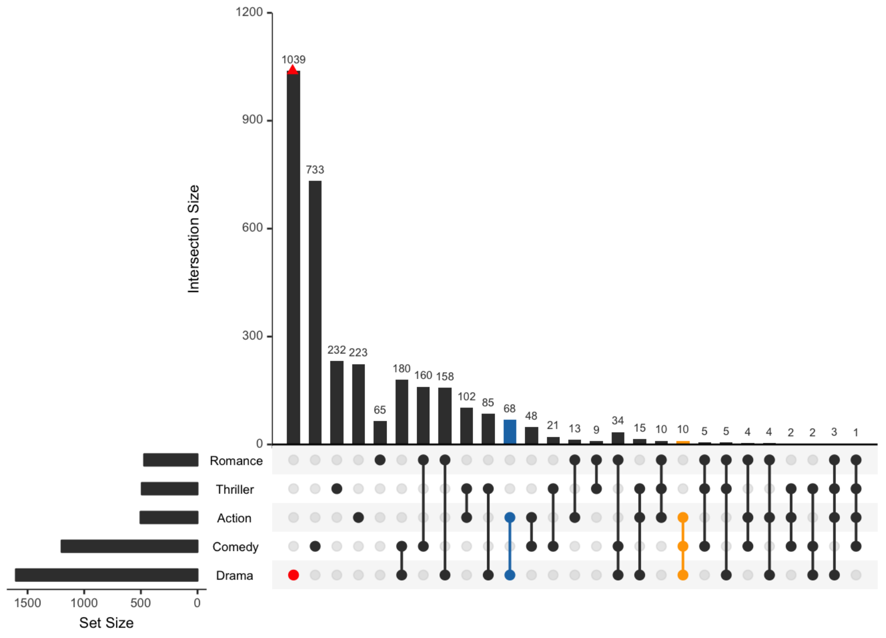
## queries parameter

Each list contained in the queries parameter takes 4 fields: `query`, `param`, `color`, `active`.

- `query`: specifies which query is going to be run
- `param`: is a list of paramters for the query to work on
- `color`: is the color that will represent the query on the plot. If no color is provided, a color will be selected from the UpSetR default color palette.
- `active`: determines how the query will be represented on the plot. If active is TRUE, the intersection size bar will be overlayed by a bar representing the query. If active is FALSE, a jitter point will be placed on the intersection size bar.

### Built-in Intersection Query

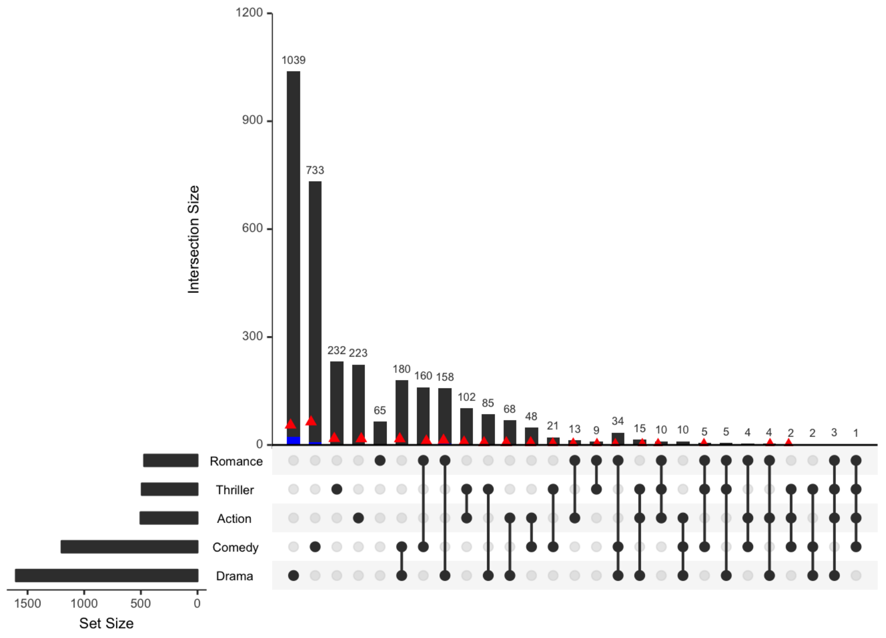
```
upset(movies, queries = list(list(query=intersects, params=list("Drama", "Comedy", "Action"), color="orange", acti
ve=T),
                             list(query=intersects, params=list("Drama"), color="red", active=F),
                             list(query=intersects, params=list("Action", "Drama"), active=T)))
```



## `elements` query

This example shows how to use the built in element query, elements, to visualize how certain elements are distributed amongst the intersections.

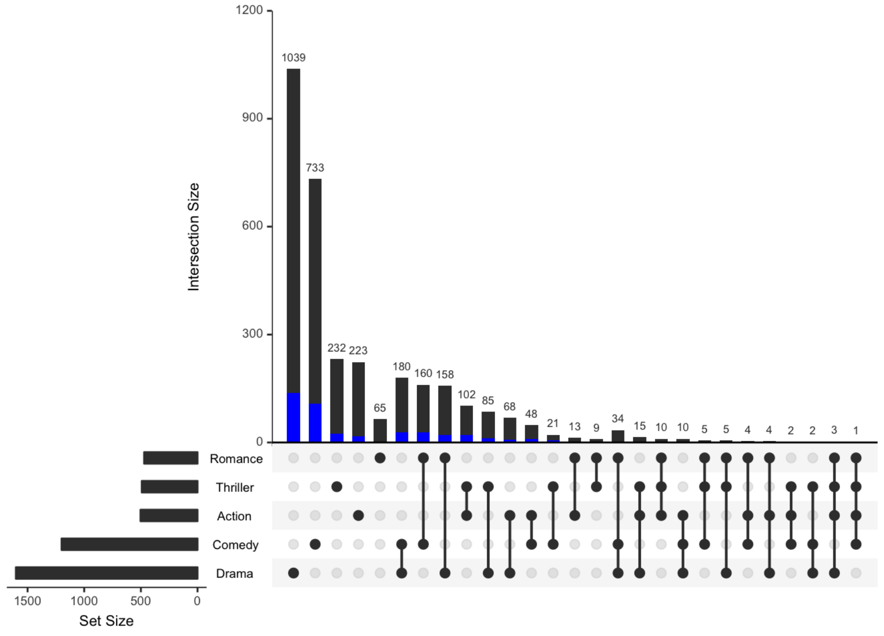
```
upset(movies, queries = list(list(query=elements, params=list("AvgRating", 3.5, 4.1), color="blue", active=T),
                             list(query=elements, params=list("ReleaseDate", 1980, 1990, 2000), color="red", activ
e=F)))
```



## Custom Queries

Creating a custom query to operate on the rows of the data.

```
Myfunc <- function(row, release, rating) {
    data <- (row["ReleaseDate"] %in% release) & (row["AvgRating"] > rating)
}
upset(movies, queries = list(list(query = Myfunc, params = list(c(1970, 1980,
    1990, 1999, 2000), 2.5), color = "blue", active = T)))
```



## Parameter `attribute.plots`

The `attribute.plots` parameter is broken down into 3 fields: `gridrows`, `plots`, and `ncols`

`gridrows`: specifies how much to expand the plot window to add room for attribute plots. The UpSetR plot is plotted on a 100 by 100 grid. So for example, if we set `gridrows` to 50, the new grid layout would be 150 by 100, setting aside 1/3 of the plot for the attribute plots.

`plots`: takes a list of paramters. These paramters include `plot`, `x`, `y` (if applicable), and queries.

`plot`: is a function that returns a ggplot

`x`: is the x aesthetic to be used in the ggplot (entered as string)

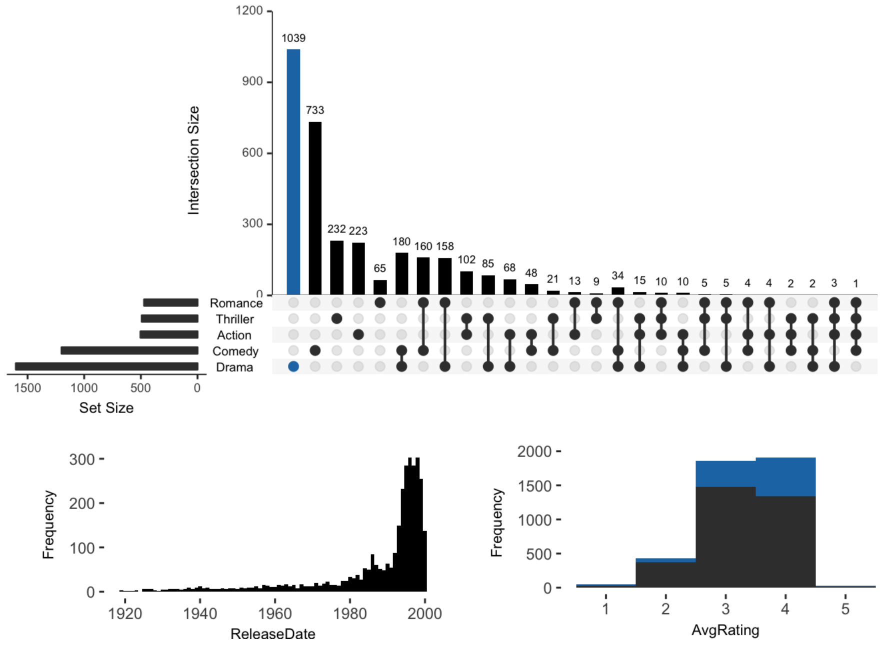
`y`: is the y aesthetic to be used in the ggplot (entered as string)

`queries`: indicates whether or not to overlay the plot with the queries present. If queries is TRUE, the attribute plot will be overlayed with data from the queries. If `queries` is `FALSE`, no query results will be plotted on the attribute plot.

`ncols`: specifies how the plots should be arranged in the `gridrows` space. If two attribute plots are entered and `ncols` is 1,then the plots will display one above the other. Alternatively, if two attribute plots are entered and ncols is 2, the attribute plots will be displayed side by side.

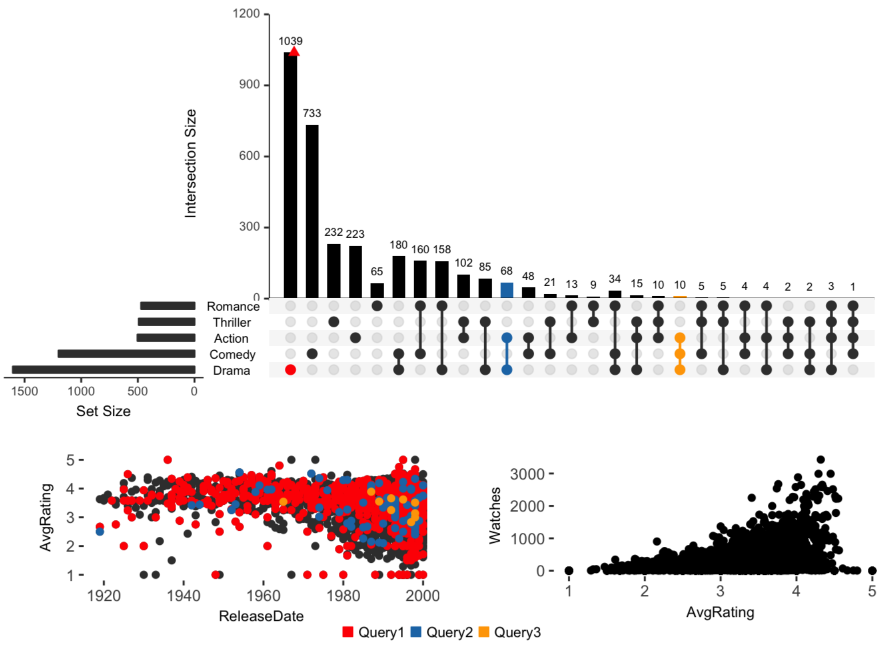
### barplot

```
upset(movies, main.bar.color = "black", queries = list(list(query = intersects,
    params = list("Drama"), active = T)), attribute.plots = list(gridrows = 50,
    plots = list(list(plot = histogram, x = "ReleaseDate", queries = F), list(plot = histogram,
        x = "AvgRating", queries = T)), ncols = 2))
```



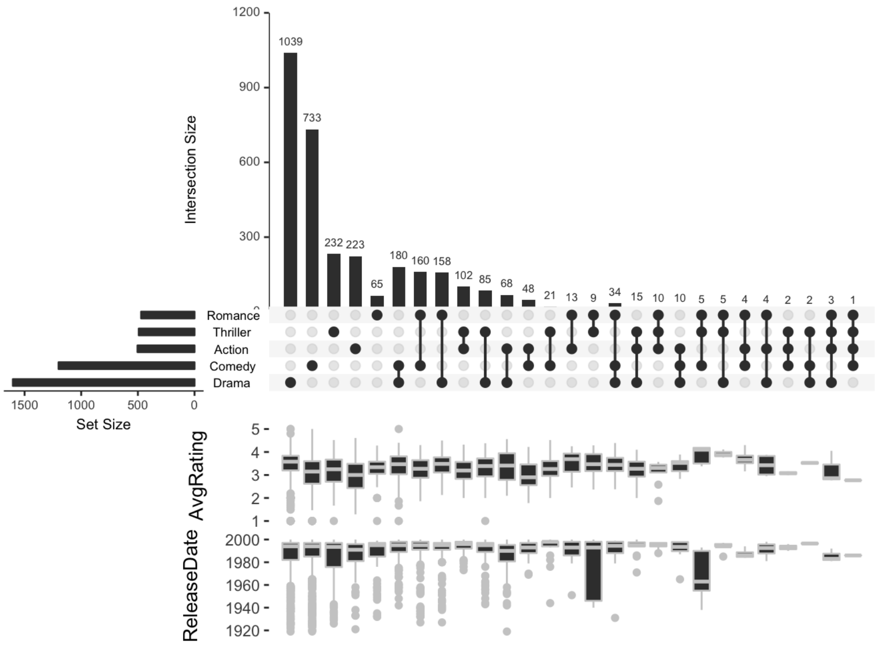
### scatterplot

```
upset(movies, main.bar.color = "black", queries = list(list(query = intersects,
    params = list("Drama"), color = "red", active = F), list(query = intersects,
    params = list("Action", "Drama"), active = T), list(query = intersects,
    params = list("Drama", "Comedy", "Action"), color = "orange", active = T)),
    attribute.plots = list(gridrows = 45, plots = list(list(plot = scatter_plot,
        x = "ReleaseDate", y = "AvgRating", queries = T), list(plot = scatter_plot,
        x = "AvgRating", y = "Watches", queries = F)), ncols = 2), query.legend = "bottom")
```



## boxplot

```
upset(movies, boxplot.summary = c("AvgRating", "ReleaseDate"))
```



## Conclusion

UpSetR has many of the features of our interactive UpSet plots, specifically it comes with various ways to sort and filter intersections and can plot attributes about the elements in the various sets.

In this post, I have demostrated:

- basic information about package `UpSet`
- a example that shows basic manipulations of `UpSet`
- queries parameter of `UpSet`
- data visualization of `UpSet`

## References

- https://www.rdocumentation.org/packages/UpSetR/versions/1.3.3/topics/upset
- https://cran.r-project.org/web/packages/UpSetR/vignettes/basic.usage.html
- https://stackoverflow.com/questions/47456663/using-queries-in-upsetr
- http://caleydo.org/tools/upset/

- https://cran.r-project.org/web/packages/UpSetR/README.html
- https://github.com/hms-dbmi/UpSetR
- https://cran.r-project.org/web/packages/UpSetR/vignettes/attribute.plots.html