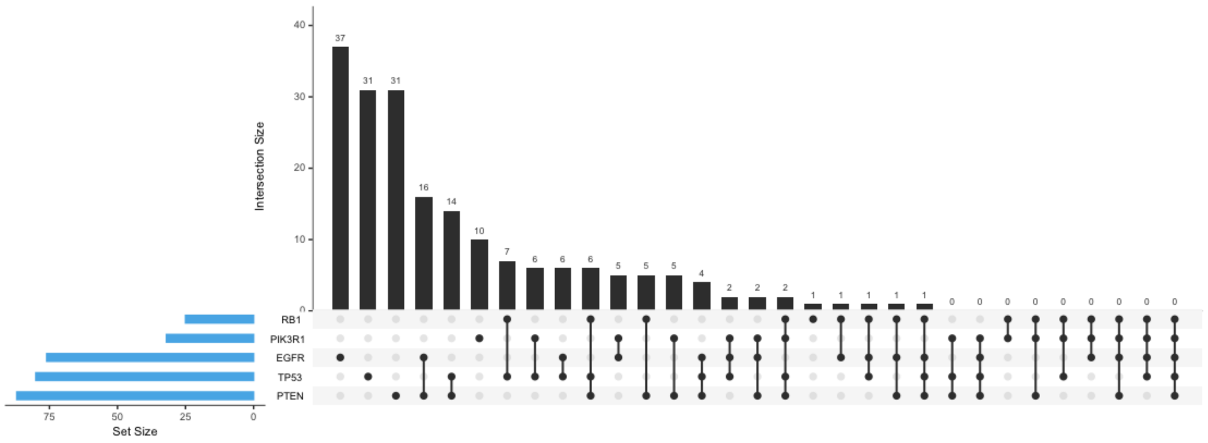
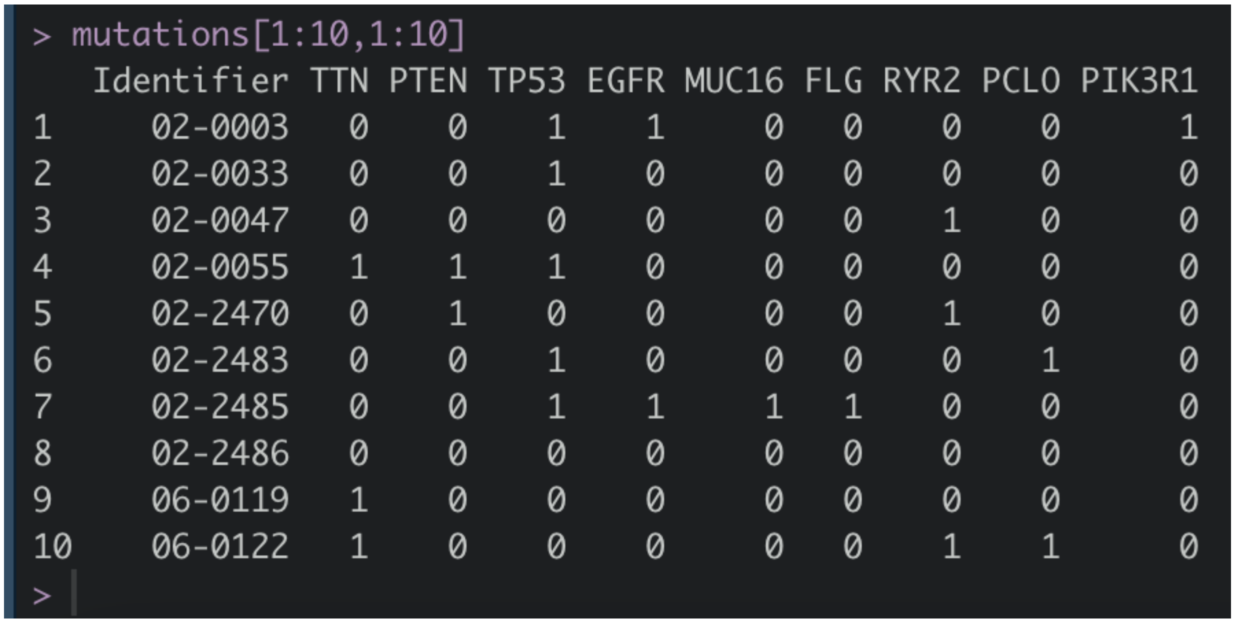
Upset Plots

A way to visualise overlap when your data is a bit too much for (readable) Venn diagrams

```
# example using UpsetR

require(UpsetR)

mutations <- read.csv( system.file("extdata", "mutations.csv", package =
"UpSetR"), header=T, sep = ",")
upset(mutations, sets = c("PTEN", "TP53", "EGFR", "PIK3R1", "RB1"),
sets.bar.color = "#56B4E9",
order.by = "freq", empty.intersections = "on")
```



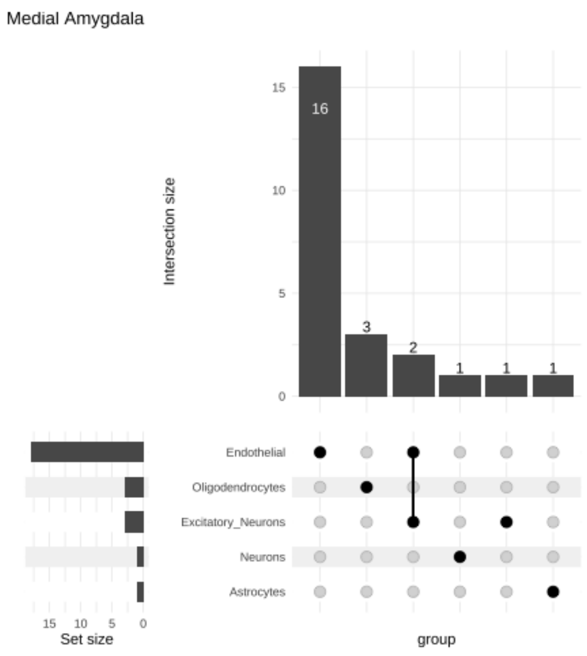
Will move on to using ComplexUpset, which basically combines UpSetR and ggplot so you can make highly customisable upset plots, and do things like plot them side by side, edit colours, etc. We need to format our data to look like the example data ('mutations') in the script above. for us, i think we want our sets to be cell types - in the example plot each set is a particular gene

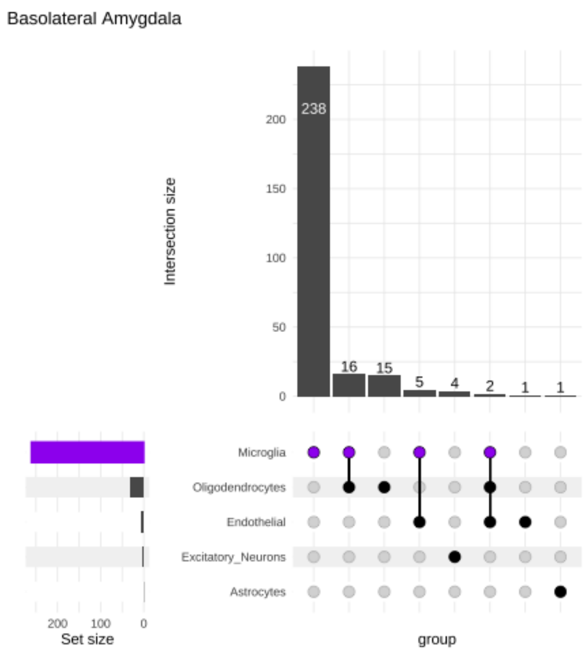

```
> mutations[1:10,1:10]
   Identifier TTN PTEN TP53 EGFR MUC16 FLG RYR2 PCLO PIK3R1
1     02-0003   0    0    1    1     0   0    0    0      1
2     02-0033   0    0    1    0     0   0    0    0      0
3     02-0047   0    0    0    0     0   0    1    0      0
4     02-0055   1    1    1    0     0   0    0    0      0
5     02-2470   0    1    0    0     0   0    1    0      0
6     02-2483   0    0    1    0     0   0    0    1      0
7     02-2485   0    0    1    1     1   1    0    0      0
8     02-2486   0    0    0    0     0   0    0    0      0
9     06-0119   1    0    0    0     0   0    0    0      0
10    06-0122   1    0    0    0     0   0    1    1      0
> 
```
So we need to format our data to show one column for each cell type, populated with zeroes and ones for every gene's DEG in that cell type

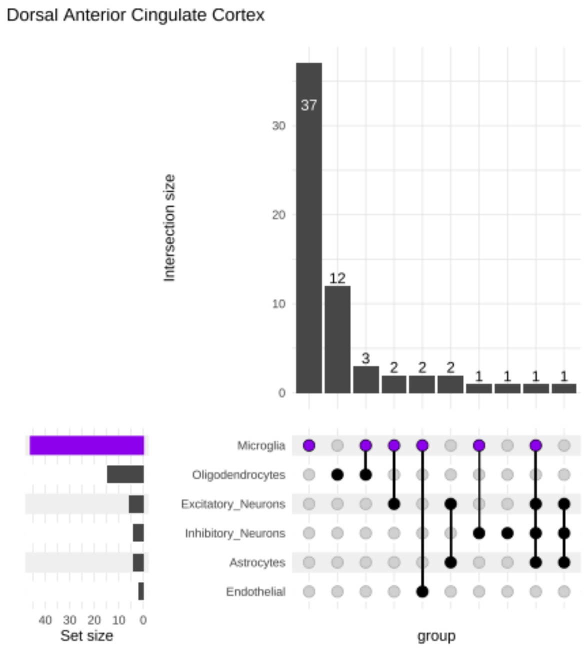
For more details on ComplexUpset see also:
https://krassowski.github.io/complex-upset/articles/Examples_R.html

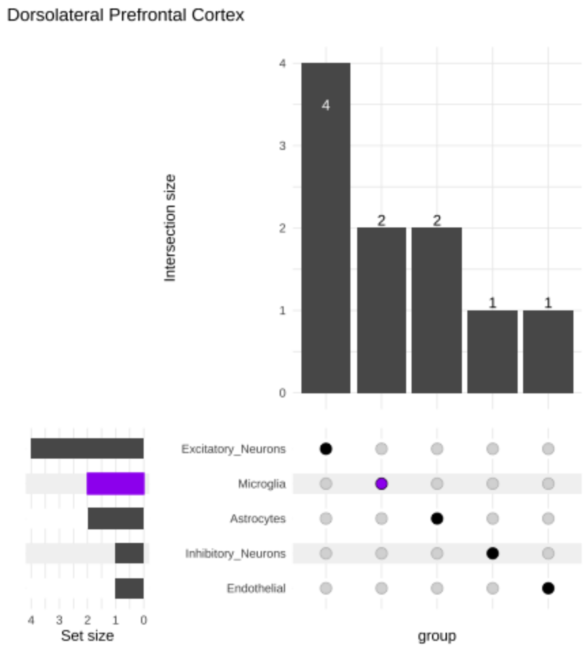
```
# each time i run this i have a clean slate except for the datasets for
Upset plot input i make per tissue
rm(list = ls()[-grep("keep",ls())])
# load packages
require(data.table)
require(ggplot2)
library(ComplexUpset)
require(gdata)

# load data - note here i use one big dataset with all results for all
tissue and cell types and cut down to tissue of interest each run
setwd("~/Dropbox/DEG_results/cell_type/")
dat<-fread("chronic_pain_cell_type_DEG_results_all_fdr")
dat<-dat[which(dat$P_bonf<0.05),] # significant results (bonferroni)

# this is just a function that makes it easier to chop up character
strings
source("~/Dropbox/nth_element.R")
```

```
tissue<-"DLPFC" # CHANGE THIS EACH RUN
dat_tissue<-dat[grep(tissue, dat$cell_type),]

##############################################################
cell1<-unique(dat_tissue$cell_type)

rm(i)
dat_tissue$N<-rep(NA,length(dat_tissue$gene))
for(i in 1:length(dat_tissue$gene)){
dat_tissue$N[i]<-length(which(dat_tissue$gene==dat_tissue$gene[i]))
}

# quick look at what genes are present in multiple cell types
unique(dat_tissue$gene[which(dat_tissue$N>1)])

##############

# making and populating a data frame that looks like the UpSetR example
one ('mutations')

# make an empty data frame with number rows equal to cell types with
significant DEGs in the tissue
# and number of columns equal to the number of unique DEGs in the tissue
dat_tissue_upset<-data.frame(matrix(ncol =
length(unique(dat_tissue$gene)),nrow = length(cell1)))
# name the columns after the genes
colnames(dat_tissue_upset)<-unique(dat_tissue$gene)
# make a cell type column
dat_tissue_upset$cell<-cell1
c1<-ncol(dat_tissue_upset)-1

rm(i)
for(i in 1:c1){
cells2<-dat_tissue$cell_type[grep(colnames(dat_tissue_upset)
[i],dat_tissue$gene)]

# for this gene, if its a DEG in a cell type, that cell type row gets a
'1'
dat_tissue_upset[which(dat_tissue_upset$cell %in%
cells2==T),which(colnames(dat_tissue_upset)==colnames(dat_tissue_upset)
[i])]<-1
# if not, that cell type row gets a '0'
dat_tissue_upset[which(dat_tissue_upset$cell %in%
cells2==F),which(colnames(dat_tissue_upset)==colnames(dat_tissue_upset)
[i])]<-0
rm(cells2)

}
dat_tissue_upset
c2<-as.numeric(ncol(dat_tissue_upset))

# transposing, so we have columns of cell types and rows of genes
d2<-data.frame(t(dat_tissue_upset))

# change column names to just be cell type instead of like
tissue_cellType
colnames(d2)<-nth_element(unlist(strsplit(as.character(d2[c2,]),
paste0(tissue,"_"))),2,2)

#remove 'cell' row at the bottom left over from the transpose step

d2<-d2[-c2,]
r1<-rownames(d2)
d2$gene<-r1
d2

rownames(d2)<-NULL

# make numeric - dont worry about the error here its just with regards
to the 'gene' column (which we dont need for plots)
d2 <- mutate_all(d2, function(x) as.numeric(x))

# making note of cell types
s1<-colnames(d2)[-grep("gene",colnames(d2))]

# final d2 = table with columns (1 per cell type) and where each row is
a significant gene, table populated with 0s and 1s to show which cell(s)
the gene is a signif DEG

# renaming using gdata functions
mv(from = "d2", to = paste0("keep1_",tissue))
mv(from = "s1", to = paste0("keep2_",tissue))

######## plotting together once you have data formatted for all 4
tissues

# removing stuff that isnt data for plotting
rm(list=ls()[-grep("keep",ls())])
require(ComplexUpset)


# upset plots where the set 'microglia' is highlighted in purple, and i
add a title (need to wrap the plot in order for this title to appear at
the top & not above the intersections matrix)

u1<-upset(keep1_MedialAmyg,intersect = keep2_MedialAmyg, wrap = T) +
ggtitle("Medial Amygdala")
u2<-upset(keep1_BasoAmyg,intersect = keep2_BasoAmyg,wrap = T,
queries=list(upset_query(set =
"Microglia",fill='purple',color="purple"))) + ggtitle("Basolateral
Amygdala")

u3<-upset(keep1_dACC,intersect = keep2_dACC, wrap = T,
queries=list(upset_query(set =
"Microglia",fill='purple',color="purple"))) + ggtitle("Dorsal Anterior
Cingulate Cortex")+ ylim(0,250)
u4<-upset(keep1_DLPFC,intersect = keep2_DLPFC,wrap = T,
queries=list(upset_query(set =
"Microglia",fill='purple',color="purple"))) + ggtitle("Dorsolateral
Prefrontal Cortex")+ ylim(0,250)

# plotting 2 upset plots at a time (2 x 2 is too big to look at properly
with R)
(u1 | u2) + plot_layout(guides='collect')
(u3 | u4) + plot_layout(guides='collect')

# saving plots to PDFs

# to make a 2-page PDF (2 upset plots per page)
pdf(file = "/path/My Plot.pdf", # The directory you want to save the
file in
width = 9,
height = 5)

(u1 | u2) + plot_layout(guides='collect')
(u3 | u4) + plot_layout(guides='collect')

dev.off()

# to make separate PDFs (2 plots per PDF, each PDF is 1 page long)
pdf(file = "upset1.pdf", # The directory you want to save the file in
width = 9,
height = 5)

(u1 | u2) + plot_layout(guides='collect')


dev.off()

pdf(file = "upset2.pdf", # The directory you want to save the file in
width = 9,
height = 5)
(u3 | u4) + plot_layout(guides='collect')

dev.off()

# can also save as PNGs, function is similar but the width and height
settings are in pixels and not inches like for PDF
```









nth_element function

*technically base R seq or seq_along (i think) would do the job here but this function is more flexible if you need to grab things that arent the first element in a set

```
#see https://stackoverflow.com/questions/5237557/extract-every-nth-
element-of-a-vector
nth_element <- function(vector, starting_position, n) {
vector[seq(starting_position, length(vector), n)]
}
```