Empirical design of genome-scale CRISPR/Cas9 libraries

Benedikt Rauscher

2019-10-31

Abstract

Until today large-scale CRISPR/Cas9 screens for the identification of essential genes have been conducted in several hundred mammalian cell lines to identify functional dependencies. However, optimal criteria for library composition and experimental design for high-quality CRISPR/Cas9 screens at minimal experimental size remain to be explored. In this study, we analyzed previously published CRISPR screens in human cells to identify single-guide (sg) RNAs with consistent high on-target and low off-target activity. We then performed screens with an empirically designed genome-scale CRISPR library in the near-haploid human HAP1 cell line, comparing hit calling in a Cas9-expressing bulk population as well as two single cell clones selected for high Cas9 activity. Our library achieved near-optimal performance at distinguishing core- and nonessential genes, identified context-dependent essential genes and avoided off-target effects of previous libraries. Depletion phenotypes were stronger in single cell clones enabling essential gene identification at a reduced library coverage of 4 sgRNAs per gene. Overall, the compendium of essential genes was highly similar between bulk population and single cell clones. Furthermore, our screens revealed previously undescribed dependencies of HAP1 cells on the Fanconi anemia pathway and the pluripotency factors SOX2 and POU5F1 (Oct4), which in part are shared dependencies with embryonic stem cells. In summary, we demonstrate that highly active CRISPR/Cas9 libraries can be designed using prior information on previously published sgRNA phenotypes. Our results further indicate that single cell clones with high Cas9 activity are powerful models to increase the dynamic range in gene essentiality screens at reduced library coverage.

Contents

```
library(HDCRISPR2019)
library(ROCR)
#> Loading required package: gplots
#>
#> Attaching package: 'gplots'
#> The following object is masked from 'package:stats':
#>
#>     lowess
library(clusterProfiler)
#>
#> Registered S3 method overwritten by 'enrichplot':
#>   method               from
#>   fortify.enrichResult DOSE
#> clusterProfiler v3.12.0  For help: https://guangchuangyu.github.io/software/clusterProfiler
#>
#> If you use clusterProfiler in published research, please cite:
#> Guangchuang Yu, Li-Gen Wang, Yanyan Han, Qing-Yu He. clusterProfiler: an R package for comparing biologica
library(limma)
library(org.Hs.eg.db)
#> Loading required package: AnnotationDbi
#> Loading required package: stats4
#> Loading required package: BiocGenerics
#> Loading required package: parallel
#>
#> Attaching package: 'BiocGenerics'
#> The following objects are masked from 'package:parallel':
#>
#>     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
#>     clusterExport, clusterMap, parApply, parCapply, parLapply,
#>     parLapplyLB, parRapply, parSapply, parSapplyLB
#> The following object is masked from 'package:limma':
#>
#>     plotMA
#> The following objects are masked from 'package:stats':
#>
#>     IQR, mad, sd, var, xtabs
#> The following objects are masked from 'package:base':
#>
#>     Filter, Find, Map, Position, Reduce, anyDuplicated, append,
#>     as.data.frame, basename, cbind, colnames, dirname, do.call,
#>     duplicated, eval, evalq, get, grep, grepl, intersect,
#>     is.unsorted, lapply, mapply, match, mget, order, paste, pmax,
#>     pmax.int, pmin, pmin.int, rank, rbind, rownames, sapply,
#>     setdiff, sort, table, tapply, union, unique, unsplit, which,
#>     which.max, which.min
#> Loading required package: Biobase
#> Welcome to Bioconductor
#>
#>     Vignettes contain introductory material; view with
#>     'browseVignettes()'. To cite Bioconductor, see
#>     'citation("Biobase")', and for packages 'citation("pkgname")'.
```

```
#> Loading required package: IRanges
#> Loading required package: S4Vectors
#>
#> Attaching package: 'S4Vectors'
#> The following object is masked from 'package:gplots':
#>
#>     space
#> The following object is masked from 'package:base':
#>
#>     expand.grid
#>
library(preprocessCore)
library(gscreend)
library(SummarizedExperiment)
#> Loading required package: GenomicRanges
#> Loading required package: GenomeInfoDb
#> Loading required package: DelayedArray
#> Loading required package: matrixStats
#>
#> Attaching package: 'matrixStats'
#> The following objects are masked from 'package:Biobase':
#>
#>     anyMissing, rowMedians
#> Loading required package: BiocParallel
#>
#> Attaching package: 'DelayedArray'
#> The following objects are masked from 'package:matrixStats':
#>
#>     colMaxs, colMins, colRanges, rowMaxs, rowMins, rowRanges
#> The following object is masked from 'package:clusterProfiler':
#>
#>     simplify
#> The following objects are masked from 'package:base':
#>
#>     aperm, apply, rowsum
library(reshape2)
library(eulerr)
#> Registered S3 method overwritten by 'eulerr':
#>   method    from
#>   plot.venn gplots
#>
#> Attaching package: 'eulerr'
#> The following object is masked from 'package:gplots':
#>
#>     venn
library(UpSetR)
library(gridExtra)
#>
#> Attaching package: 'gridExtra'
#> The following object is masked from 'package:Biobase':
#>
```

```
#>     combine
#> The following object is masked from 'package:BiocGenerics':
#>
#>     combine
library(pheatmap)
library(ggrastr)
library(cowplot)
#>
#> ******************************************************
#> Note: As of version 1.0.0, cowplot does not change the
#>   default ggplot2 theme anymore. To recover the previous
#>   behavior, execute:
#>   theme_set(theme_cowplot())
#> ******************************************************
library(ggrepel)
#> Loading required package: ggplot2
library(tidyverse)
#> -- Attaching packages ---------------------------------------------- tidyverse 1.2.1 --
#> v tibble  2.1.3     v purrr   0.3.3
#> v tidyr   1.0.0     v dplyr   0.8.3
#> v readr   1.3.1     v stringr 1.4.0
#> v tibble  2.1.3     v forcats 0.4.0
#> -- Conflicts ------------------------------------------------- tidyverse_conflicts() --
#> x ggplot2::Position() masks BiocGenerics::Position(), base::Position()
#> x dplyr::collapse()   masks IRanges::collapse()
#> x dplyr::combine()    masks gridExtra::combine(), Biobase::combine(), BiocGenerics::combine()
#> x dplyr::count()      masks matrixStats::count()
#> x dplyr::desc()       masks IRanges::desc()
#> x tidyr::expand()     masks S4Vectors::expand()
#> x dplyr::filter()     masks stats::filter()
#> x dplyr::first()      masks S4Vectors::first()
#> x dplyr::lag()        masks stats::lag()
#> x purrr::reduce()     masks GenomicRanges::reduce(), IRanges::reduce()
#> x dplyr::rename()     masks S4Vectors::rename()
#> x dplyr::select()     masks AnnotationDbi::select()
#> x purrr::simplify()   masks DelayedArray::simplify(), clusterProfiler::simplify()
#> x dplyr::slice()      masks IRanges::slice()
```

1 Design

1.1 Data

We require a number of datasets for downstream analyses that we first load into memory.

```
## core-essential and non-essential genes
data('cene', package='HDCRISPR2019')
```

1.2 Overview library composition

1.2.1 Determination of validated sgRNAs

1.2.1.1 BAGEL evalutation of dropout screens

We previously re-evaluated 439 dropout screen in GenomeCRISPR using BAGEL. We use the calculated Bayes Factors and corresponding log2 fold changes to determine hits at 5% false discovery rate (FDR). We then select the sgRNAs that contributed to each of the hits by showing a dropout phenotype. These sgRNAs will then be selected for the HD CRISPR library. If an sgRNA showed a phenotype in more than one screen, we prefer it over other sgRNAs with the same target gene.

1.2.1.1.1 BAGEL results hit calling

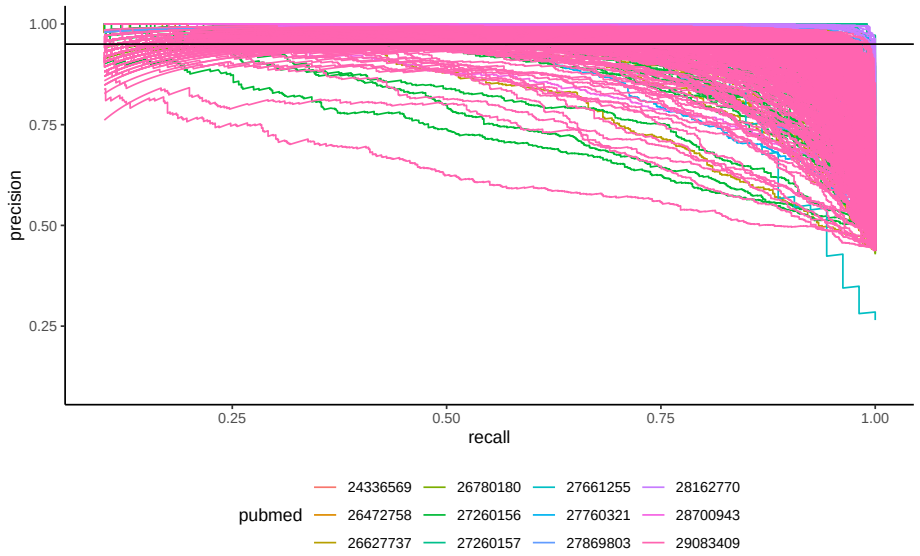
First we read the Bayes Factors as returned by BAGEL and use the ROCR package with the core-essential gene set 2 and the non-essential gene set (Hart 2017, G3) to determine hits at 5% FDR.

```
## load BAGEL results for GenomeCRISPR screens
data('bagel_results', package='HDCRISPR2019')

## generate ROC curves
roc_results <- lapply(bagel_results, function(screen){
  data <- screen %>% mutate(label=ifelse(GENE %in% ce$symbol, 1,
                                  ifelse(GENE %in% ne$symbol, 0, -1))) %>%
    filter(label != -1)
  pred <- prediction(data$BF, data$label)
  perf <- performance(pred, measure='prec', x.measure='rec')
  ## also calculate auc
  auc <- performance(pred, measure='auc')@y.values[[1]]
  cutoff <- perf@alpha.values[[1]][max(which(perf@y.values[[1]] > 0.95))]
  return(tibble(
    pubmed = data$pubmed[1],
    cellline = data$cellline[1],
    condition = data$condition[1],
    precision = perf@y.values[[1]],
    recall = perf@x.values[[1]],
    cutoff = cutoff,
    auc = auc
  ))
})
```

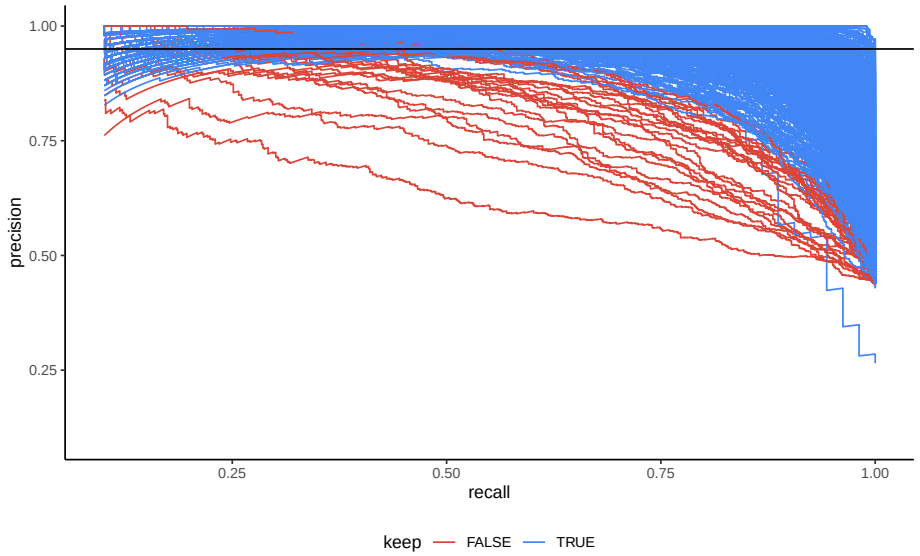
To use only high-quality screens for downstream processing, we check the quality of the screens by plotting the ROC curves. We evalute overall screen quality by the area under the precision-recall-curve. We retain screens for futher analysis if the area under the curve (AUC) is greater than 0.9.

```
## plot ROC curves to check whether all screens are OK
roc_results %>% bind_rows() %>% unite(screen, pubmed, cellline, condition, sep='_', remove=F) %>%
  ggplot(aes(x=recall, y=precision, group=screen, colour=pubmed)) +
  geom_line() + theme_classic() + geom_hline(yintercept=0.95) +
```

```
  theme(legend.position = 'bottom') +
  ylim(c(0.1, 1)) + xlim(c(0.1, 1))
```

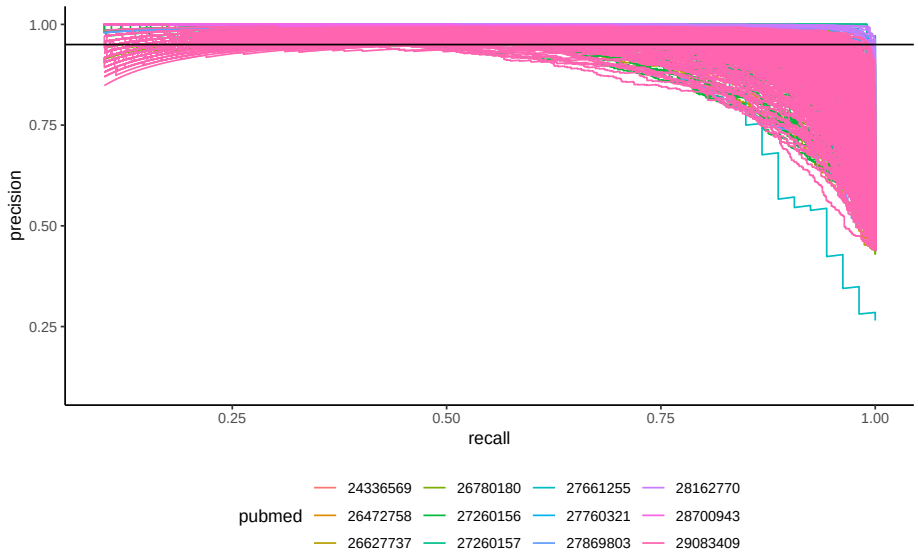



```
## highlight screens ROC < 0.9
roc_results %>% bind_rows() %>%
  unite(screen, pubmed, cellline, condition, sep='_', remove=F) %>%
  mutate(keep = ifelse(auc > 0.9, T, F)) %>%
  ggplot(aes(x=recall, y=precision, group=screen, colour=keep)) +
  geom_line() + theme_classic() + geom_hline(yintercept=0.95) +
  theme(legend.position = 'bottom') +
  ylim(c(0.1, 1)) + xlim(c(0.1, 1)) +
  scale_colour_manual(values = c('#db4437', '#4285f4'))
```



Some of the screens have low AUC values. We exclude these from the data before continuing with downstream analyses. We keep only screens with an AUC greater than 0.9. We plot the ROC curves again to visualize retained and excluded screens based on their precision-recall-curves.

```
## remove auc < 0.9
roc_results_fil <- roc_results %>% bind_rows() %>%
  filter(auc > 0.9, !is.na(cutoff))

## plot roc curves again
roc_results_fil %>%
  unite(screen, pubmed, cellline, condition, sep='_', remove=F) %>%
  ggplot(aes(x=recall, y=precision, group=screen, colour=pubmed)) +
  geom_line() + theme_classic() + geom_hline(yintercept=0.95) +
  theme(legend.position = 'bottom') +
  ylim(c(0.1, 1)) + xlim(c(0.1, 1))
```



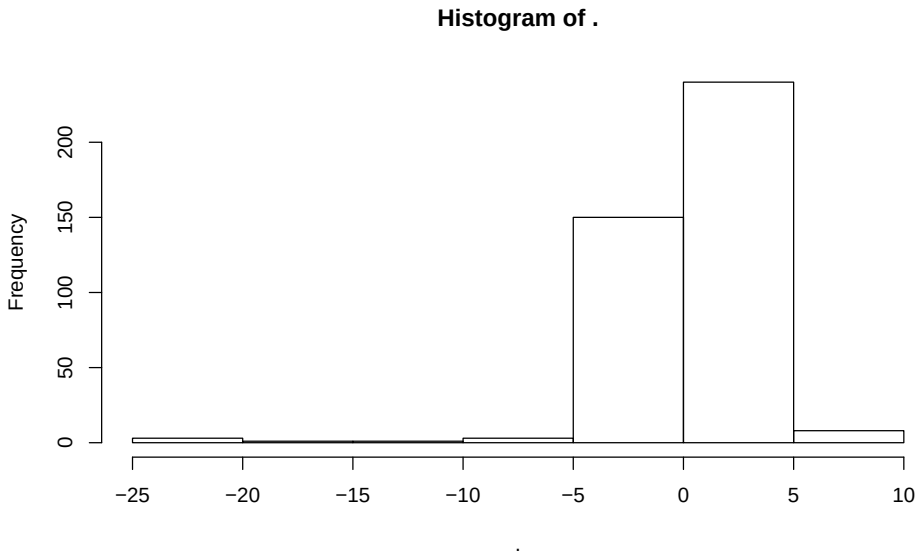
We next load the corresponding log2-fold changes to get sgRNA level dropout phenotypes for each screens.

```
## read fold changes object from disk
data('fchanges', package = 'HDCRISPR2019')
```

1.2.1.2 BAGEL results quality control

To assess if the BAGEL analysis worked as intended we generate a number of quality control visualizations. We first plot a histogram of all selected cut-offs for gene essentiality.

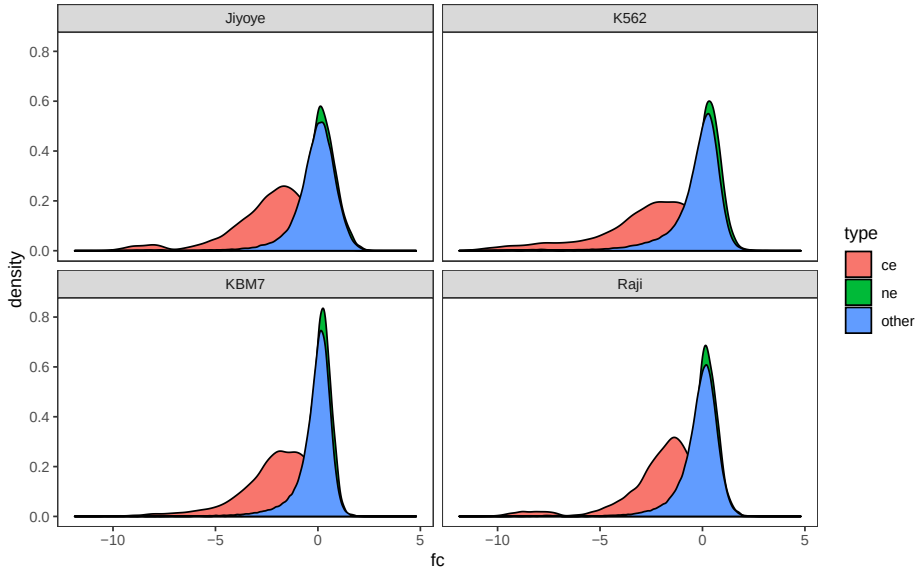
```
roc_results_fil %>%
  distinct(pubmed, cellline, condition, cutoff) %>%
  .$cutoff %>% hist()
```



.

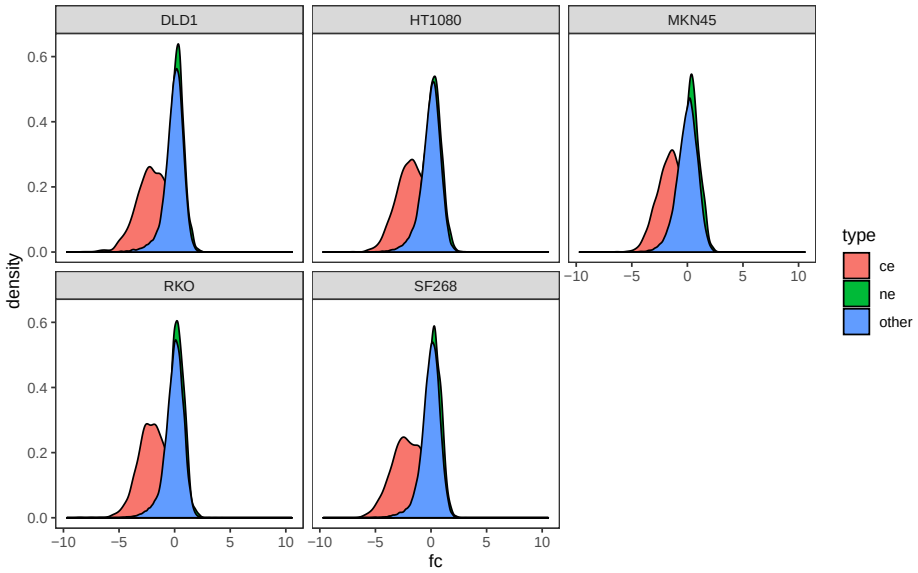
Most screens have a cutoff that matches what we would have expected. For some, however, the cutoff is quite high and for others it is considerably smaller than 0 (we would not expect any < 0 at all). It seems that these screens are mostly the ones performed using the library by Wang et al. We produce boxplots/density plots for the Wang 2015 screens (which are known to be of good quality).

```
fchanges %>% filter(pubmed == '26472758') %>%
  mutate(type = ifelse(GENE %in% ce$symbol, 'ce',
                ifelse(GENE %in% ne$symbol, 'ne', 'other'))) %>%
    ggplot(aes(fc, fill=type)) + geom_density() +
    facet_wrap(~cellline) +  theme_bw() +
    theme(panel.grid=element_blank())
```




```
fchanges %>% filter(pubmed == '27260157') %>%
  mutate(type = ifelse(GENE %in% ce$symbol, 'ce',
```

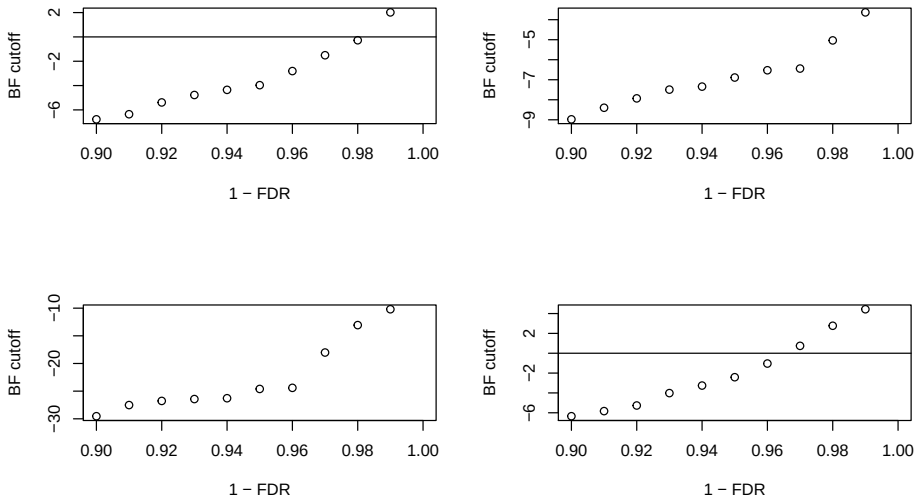
```
                  ifelse(GENE %in% ne$symbol, 'ne', 'other'))) %>%
    ggplot(aes(fc, fill=type)) + geom_density() +
    facet_wrap(~cellline) +  theme_bw() +
    theme(panel.grid=element_blank())
```



These look absolutely fine. It could be that these screens separate core- and non-essential genes so well that the cutoff becomes negative to generate the 5% FDR that we allow. This might not be too surprising as these screens were used to actually generate the core- and nonessential gene lists. We can confirm this by selecting cutoffs for different false discovery rates.

```
## 4 screens with negative cutoff
negscreens <- list(
  list(pubmed = '26472758', cellline = 'Jiyoye'),
  list(pubmed = '24336569', cellline = 'KBM7'),
  list(pubmed = '27260157', cellline = 'RKO'),
  list(pubmed = '26472758', cellline = 'KBM7')
)

par(mfrow=c(2,2))
for(screen in negscreens){
  wscreen <- bagel_results %>% bind_rows() %>%
    filter(pubmed == screen$pubmed, cellline == screen$cellline) %>%
    mutate(label=ifelse(GENE %in% ce$symbol, 1,
                  ifelse(GENE %in% ne$symbol, 0, -1))) %>%
    filter(label != -1)

  wpred <- prediction(wscreen$BF, wscreen$label)
  wperf <- performance(wpred, measure='prec', x.measure='rec')
  cos <- lapply(seq(1, 0.9, by= -0.01), function(x){
    ifelse(length(which(wperf@y.values[[1]] > x)) == 0, NA,
      wperf@alpha.values[[1]][max(which(wperf@y.values[[1]] > x))]
    )
```

```
  })
  plot(seq(1, 0.9, by= -0.01), unlist(cos),
       xlab='1 - FDR', ylab='BF cutoff')
  if(max(unlist(cos), na.rm=T) > 0){
    abline(h=0)
  }
}
```

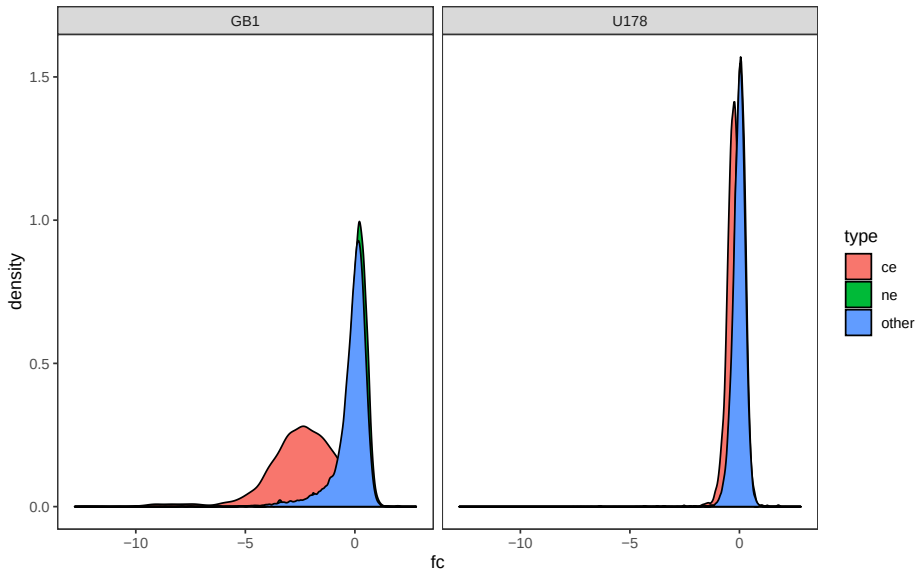


This confirms what we expected. Hence we will just set the cutoff for these screens to 0.

```
roc_results_fil <- roc_results_fil %>%
  mutate(cutoff = ifelse(cutoff < 0, 0, cutoff))
```

We futher compare a screen with a large AUC to a screen with a small AUC.

```
fchanges %>%
  filter(pubmed == '29083409', cellline %in% c('U178', 'GB1')) %>%
  mutate(type = ifelse(GENE %in% ce$symbol, 'ce',
                ifelse(GENE %in% ne$symbol, 'ne', 'other'))) %>%
  ggplot(aes(fc, fill=type)) + geom_density() +
  facet_wrap(~cellline) +  theme_bw() +
  theme(panel.grid=element_blank())
```



This matches what we would have expected. While the separation is very clear in the screen with the high AUC, it is much smaller than in the screen with the low AUC value.

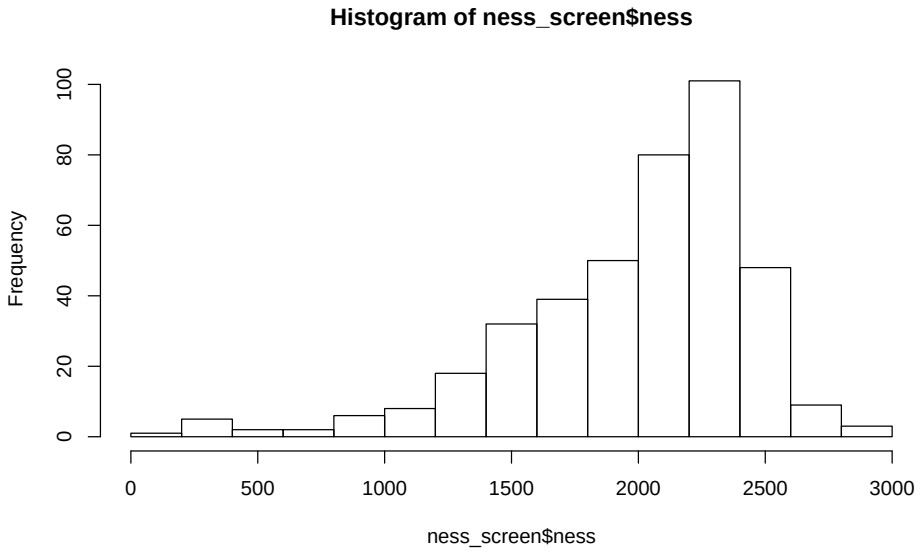
1.2.1.3 Merging data

We merge Bayes Factors and sgRNA fold changes into a shared data frame for downstream analysis.

```
## merge and apply cutoff.
merged <- fchanges %>%
  inner_join(roc_results_fil %>% distinct(pubmed, cellline,
                                          condition, cutoff)) %>%
  inner_join(bagel_results %>% bind_rows() %>%
               dplyr::select(pubmed, cellline,
                             condition, GENE, BF)) %>%
  mutate(hit = ifelse(BF > cutoff & BF > 0, T, F))
```

How many essentials do we get per screen? Is there anything unexpected going on?

```
## count essential genes per screen
ness_screen <- merged %>% distinct(pubmed, cellline, condition,
                                   cutoff, GENE, hit) %>%
  group_by(pubmed, cellline, condition, cutoff) %>%
  summarise(ness = sum(hit)) %>% ungroup()

hist(ness_screen$ness)
```



This looks fine. The screens with low amounts of essential genes are screens with smaller libraries where only a limited number of genes were screened. Screens with the Yusa library were done use shorter 19 bp sgRNAs. We extend these sequences to 20 bp, which is what we want to use in our library. For simplicity, we assume that the 1 bp difference does not have a major impact on sgRNA performance.

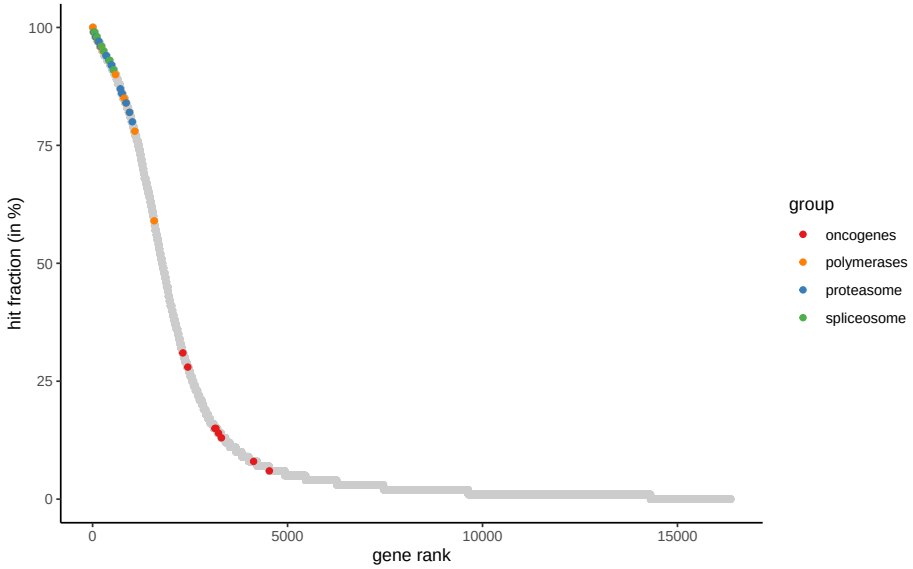
```
data('yusa_map', package = 'HDCRISPR2019')

## adapt yusa sequences to 23 bp
yusa_adapted <- merged %>% filter(nchar(SEQID) == 22) %>%
  left_join(yusa_map %>% dplyr::rename(SEQID=short)) %>%
  mutate(SEQID=long) %>%
  filter(!is.na(long)) %>% distinct() %>% dplyr::select(-long)

## add back to the merged data
merged <- merged %>% filter(nchar(SEQID) != 22) %>%
  bind_rows(yusa_adapted)

## the data includes one CRISPRi screen that we remove
merged <- merged %>% filter(pubmed != '27661255')
```

1.2.1.4 Frequent hitters

As an additional validation we generate a plot that shows the fraction of cell lines in which selected genes are essential.

```
## number of hits per gene
hits_per_gene <- merged %>% filter(hit) %>%
  distinct(pubmed, cellline, condition, GENE) %>% count(GENE)

## hit frequencey per gene
freq_hits <- hits_per_gene %>%
  inner_join(merged %>% distinct(pubmed, cellline, condition, GENE) %>%
               count(GENE) %>% dplyr::rename(n_screens=n)) %>%
```

```
  mutate(fraction=round(n/n_screens, 2)*100)

## some annotation
oncogene <- c('NRAS', 'KRAS', 'BRAF', 'PIK3CA',
              'CTNNB1', 'MYC', 'MYB', 'AKT1', 'EGFR')

proteasome <- ce %>% filter(grepl('^PSM', symbol)) %>% .$symbol
spliceosome <- ce %>% filter(grepl('^SF', symbol)) %>% .$symbol
polymerases <- ce %>% filter(grepl('^POLR', symbol)) %>% .$symbol

plot_data <- freq_hits %>% arrange(desc(fraction)) %>%
  mutate(rank=1:n(),
         group=ifelse(GENE %in% oncogene, 'oncogenes',
               ifelse(GENE %in% proteasome, 'proteasome',
               ifelse(GENE %in% spliceosome, 'spliceosome',
               ifelse(GENE %in% polymerases, 'polymerases',
                      'rest')))))

plot_data %>% ggplot(aes(rank, fraction)) +
  geom_point(data=filter(plot_data, group == 'rest'),
             colour='#cccccc') +
  geom_point(data=filter(plot_data, group != 'rest'),
             aes(colour=group)) +
  theme_classic() +
  scale_colour_manual(values=c('#e41a1c', '#ff7f00',
                               '#377eb8', '#4daf4a')) +
  xlab('gene rank') + ylab('hit fraction (in %)')
```



This looks as expected. Core-essential processes are essential in $>$ 75 % of all cell lines, while oncogenes are essential only in specific contexts.

1.2.2 Selection of validated sgRNAs

In the next step we select empirically 'validated' sgRNAs based on the data set generated in previous sections. We select all genes that are called 'hits' (or essential) and select the sgRNAs targeting these hits and that rank amongst the 20% most depleted sgRNAs in the whole screen. We choose 20% cutoff as we expect up to 20% of the genes to be essential in a screen. This is a relaxed threshold and the true fraction of essential genes in most screens is likely to be around 10%.

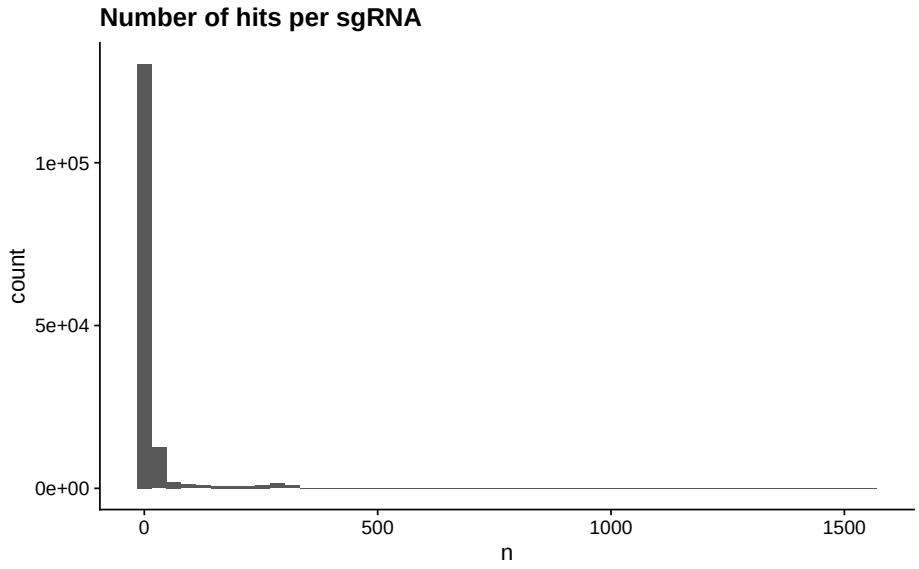
```
## sgRNAs with strong depletion phenotypes that
## target essential genes
valid_sgRNAs <- merged %>%
  group_by(pubmed, cellline, condition) %>%
  mutate(q20=quantile(fc, probs=0.2)) %>%
  ungroup() %>%
  filter(hit, fc < q20)

## we also remember the bad sgRNAs so they can be excluded later
invalid_sgRNAs <- merged %>%
  group_by(pubmed, cellline, condition) %>%
  mutate(q20=quantile(fc, probs=0.2)) %>%
  ungroup() %>%
  filter(hit, fc > q20)
```

Now we count for each unique sgRNA in the determined set of sgRNAs the number of screens it showed a phenotype in.

```
valid_sgRNAs <- valid_sgRNAs %>% count(GENE, SEQID) %>%
  arrange(GENE, desc(n))
```

We find 153016 unique sgRNA sequences targeting in total 16221 genes.

```
## histogram of hit count
valid_sgRNAs %>% ggplot(aes(n)) +
  geom_histogram(bins=50) +
  ggtitle('Number of hits per sgRNA')
```



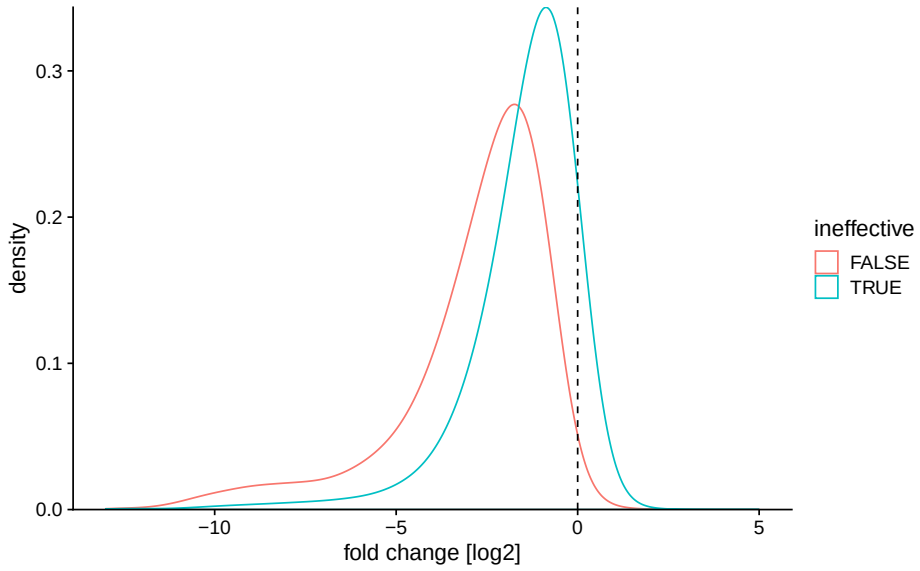
1.3 Annotated pool of sgRNAs

We load an annotated pool of sgRNAs that compiles all relevant information.

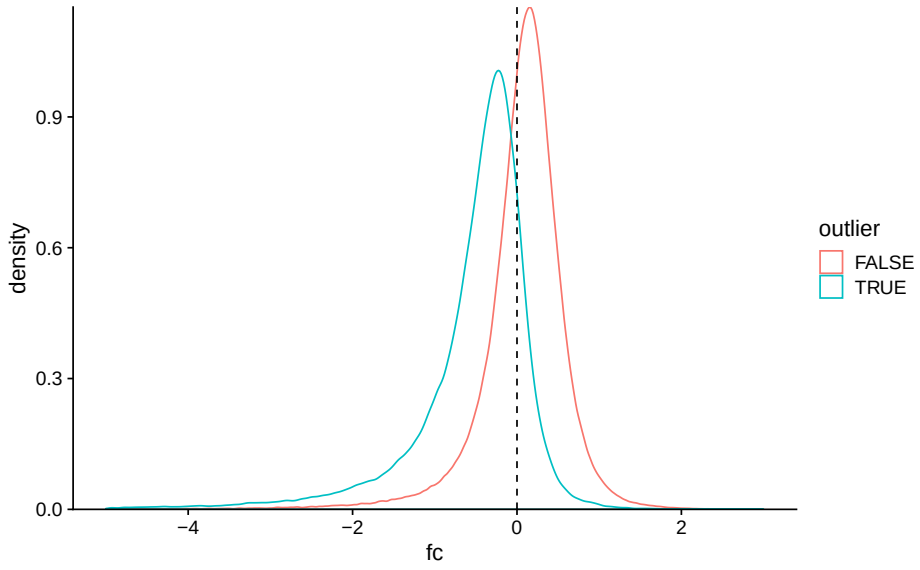
```
data('sgRNA_pool_refined', package = 'HDCRISPR2019')
```

We visualize core-essential targeting sgRNAs, comparing validated in ineffective guides. We do a similar analysis for non-essential gene-targeting sgRNAs comparing toxic and non-toxic guides.

```
## list of 'invalid' sequences
inv_seqs <- invalid_sgRNAs %>%
  mutate(nopam = substr(SEQID, 1, 20)) %>%
  pull(nopam)

## density plot of 'valid' and 'invalid' sgRNAs (core-essential)
merged %>% filter(GENE %in% ce$symbol) %>%
  mutate(nopam = substr(SEQID, 1, 20)) %>%
  mutate(ineffective = ifelse(nopam %in% inv_seqs, T, F)) %>%
  ggplot(aes(fc, colour = ineffective)) +
  geom_density(bw = 0.5) +
  scale_y_continuous(expand = c(0,0)) +
  xlab('fold change [log2]') + xlim(c(-13, 5)) +
  geom_vline(xintercept = 0, linetype = 'dashed')
```




```
## similar plot for non-essential genes
noness_outliers <- pool %>% filter(genes_hit %in% ne$symbol) %>%
  mutate(outlier = ifelse(effect_z < -1.25, T, F)) %>%
  filter(outlier) %>% pull(sequence)

merged %>% filter(GENE %in% ne$symbol) %>%
  mutate(outlier = ifelse(SEQID %in% noness_outliers, T, F)) %>%
  ggplot(aes(fc, colour = outlier)) + geom_density() +
  geom_vline(xintercept = 0, linetype = 'dashed') +
  scale_y_continuous(expand = c(0, 0)) + xlim(c(-5, 3))
```



1.3.1 Removing BbsI restriction sites and setting minimal number of screens

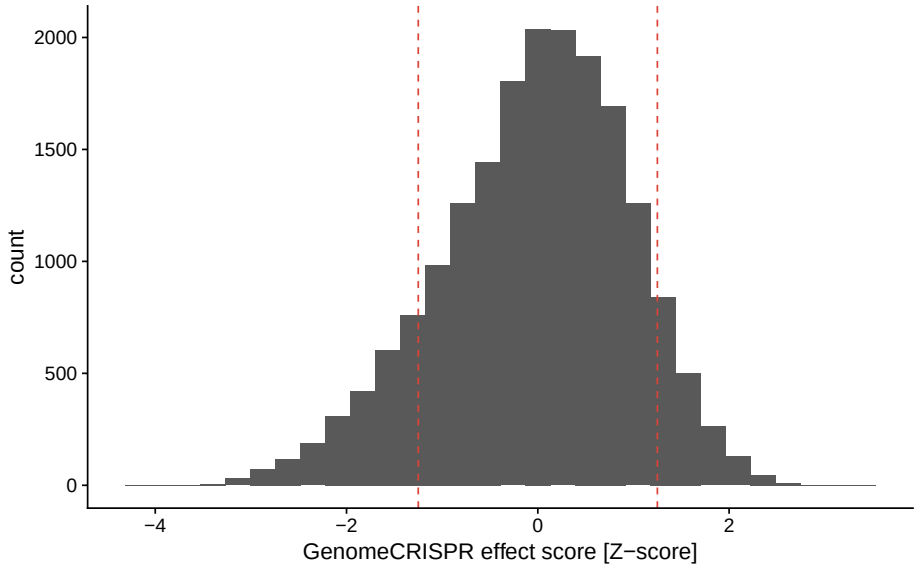
Some additional restrictions: require minimum number of screens for consideration and remove BbsI restriction sites.

```
is_bbs1 <- function(sequence){
  bbs1 <- c(
    which(grepl('GAAGAC', sequence)),
    which(grepl(system('echo GAAGAC | tr "ACGT" "TGCA" | rev', intern=T), sequence))
  )
  res <- logical(length=length(sequence))
  res[bbs1] <- TRUE
  return(res)
}

## consider only performance based on at least 5 screens
pool <- pool %>% mutate(screen_hit_ratio = ifelse(n_screens >= 5, screen_hit_ratio, 0))
## remove BbsI restriction sites
pool <- pool %>% filter(!is_bbs1(nopam))
```

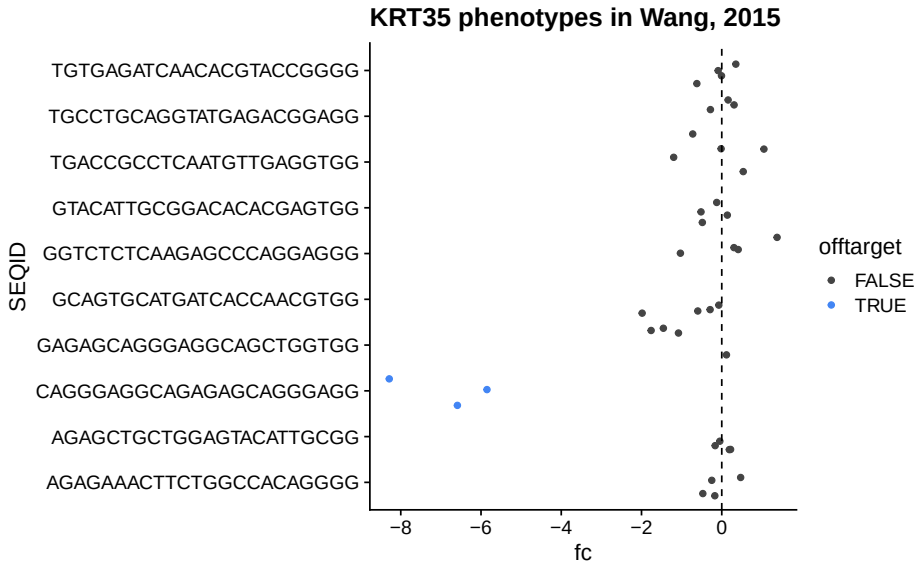
1.3.2 Examples of valid and invalid sgRNAs for core-essential genes.

For demonstration we generate plots for valid and invalid sequences determined for an essential gene (NOP2) and a nonessential gene (KRT35), highlighting sgRNAs with weak phenotypes or potential off-target effects.

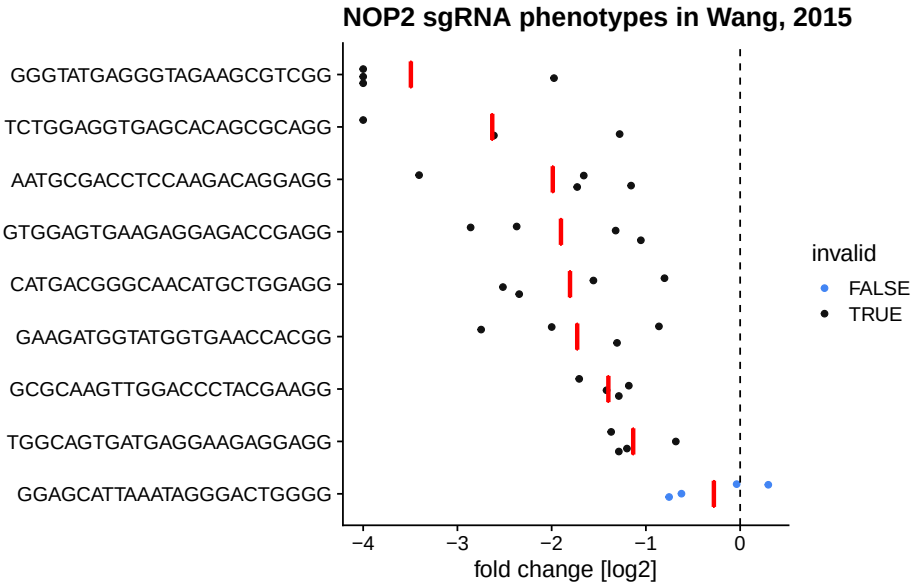
```
## a distribution of effect Z-scores
pool %>% filter(genes_hit %in% ne$symbol) %>%
  ggplot(aes(effect_z)) + geom_histogram() +
  geom_vline(xintercept = c(-1.25, 1.25),
             linetype = 'dashed', colour = '#db4437') +
  xlab('GenomeCRISPR effect score [Z-score]')
```



```
## KRT35 example
krt35_offtarget <- pool %>%
  filter(genes_hit %in% ne$symbol, effect_z < -3) %>%
  filter(genes_hit == 'KRT35') %>% pull(sequence)

fchanges %>% filter(GENE == 'KRT35', pubmed == '26472758') %>%
  mutate(offtarget = ifelse(SEQID == krt35_offtarget, T, F)) %>%
  ggplot(aes(fc, SEQID, colour = offtarget)) + geom_jitter() +
  geom_vline(xintercept = 0, linetype = 'dashed') +
  scale_colour_manual(values = c('#444444', '#4285f4')) +
  ggtitle('KRT35 phenotypes in Wang, 2015')
```




```
## visualize as jitter plot
fchanges %>% filter(pubmed == '26472758', GENE == 'NOP2') %>%
  mutate(invalid = ifelse(SEQID %in% 'GGAGCATTAAATAGGGACTGGGG', F, T),
         fc = ifelse(fc < -4, -4, fc)) %>%
  group_by(SEQID) %>% mutate(mean_fc = mean(fc)) %>% ungroup() %>%
  arrange(desc(mean_fc)) %>%
  mutate(SEQID = factor(SEQID, levels = unique(SEQID))) %>%
  ggplot(aes(SEQID, fc, colour = invalid)) +
  geom_jitter(width = 0.2) +
  stat_summary(fun.y = 'mean',
               fun.ymin = 'mean',
               fun.ymax = 'mean',
               geom='crossbar',
               color = 'red', width = 0.5) +
  geom_hline(yintercept = 0, linetype = 'dashed') +
  ylab('fold change [log2]') + xlab('') +
  scale_colour_manual(values = c('#4285f4', '#111111')) +
  coord_flip() +
  theme_cowplot() +
  ggtitle('NOP2 sgRNA phenotypes in Wang, 2015')
```



1.4 Summary statistics of final library

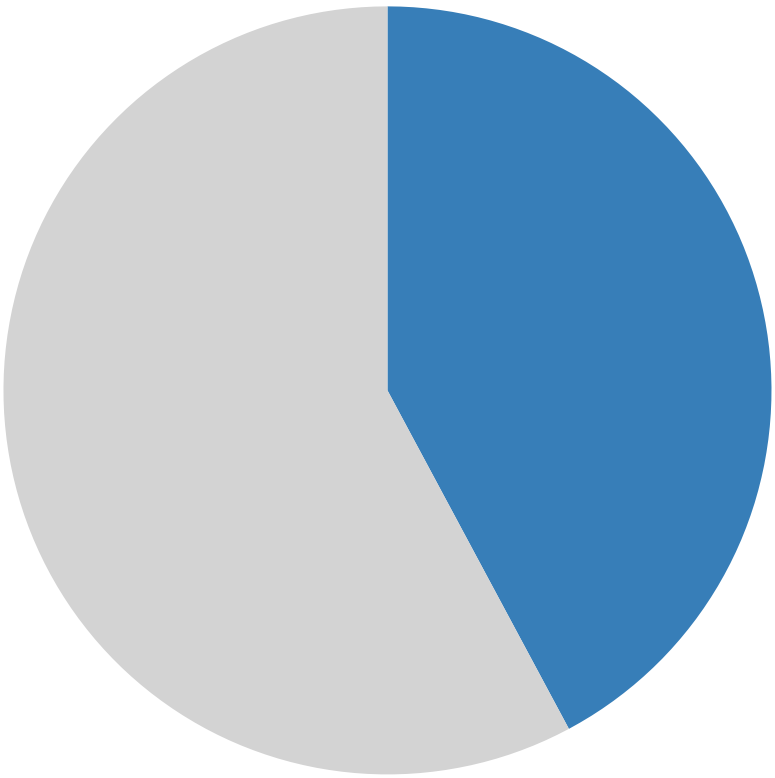
We parse the final library sequence lists from disk.

```
data('hdcrispr_seqs', package = 'HDCRISPR2019')
```

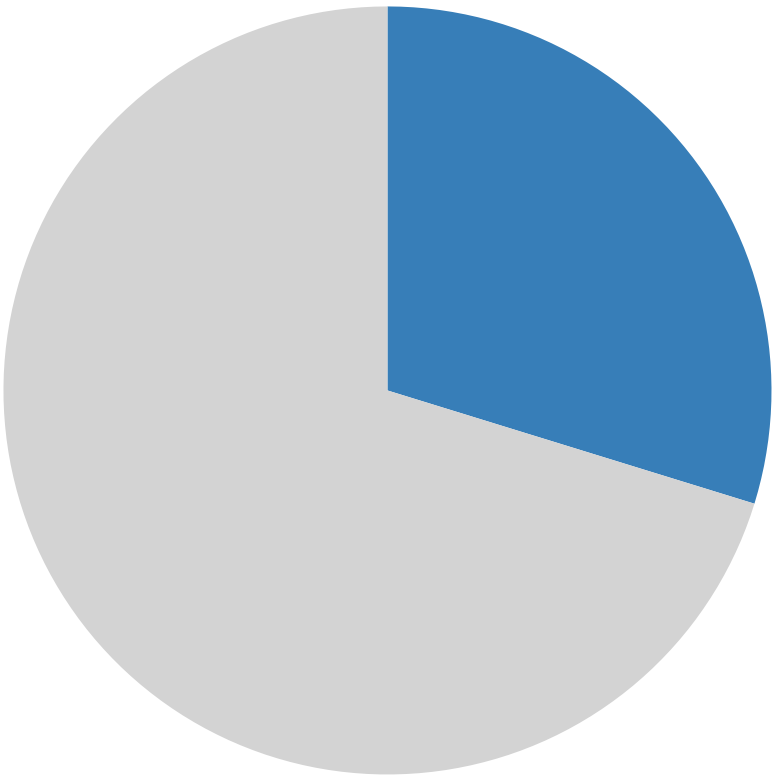
We calculate the percentage of validated sgRNAs for each sub-library.

```
valid_seqs <- valid_sgRNAs %>% filter(n>=1) %>%
  mutate(nopam = substr(SEQID, 1, 20)) %>% pull(nopam)

## annoate maximal screen-hit-ratio per gene
pool <- pool %>% group_by(genes_hit) %>% mutate(max_shr = max(screen_hit_ratio)) %>% ungroup()

## calculate percentages, make pie chart.
n_a <- libA_seqs %>% dplyr::select(nopam) %>%
  left_join(pool %>% dplyr::select(nopam, screen_hit_ratio, max_shr)) %>%
  mutate(valid = ifelse(is.na(screen_hit_ratio), F,
                 ifelse(screen_hit_ratio > 0 & max_shr > 0.05, T, F))) %>%
  count(valid)
n_b <- libB_seqs %>% dplyr::select(nopam) %>%
  left_join(pool %>% dplyr::select(nopam, screen_hit_ratio, max_shr)) %>%
  mutate(valid = ifelse(is.na(screen_hit_ratio), F,
                 ifelse(screen_hit_ratio > 0 & max_shr > 0.05, T, F))) %>%
  count(valid)

## fractions
n_a %>% summarise(frac = n[2]/sum(n))
n_b %>% summarise(frac = n[2]/sum(n))
```

```
## pie charts
n_a %>% ggplot(aes(x='', y=n, fill=valid)) +
  geom_bar(stat='identity') + coord_polar('y', start = 0) +
  scale_fill_manual(values = c('#D3D3D3', '#377EB8')) +
  theme_nothing()
```

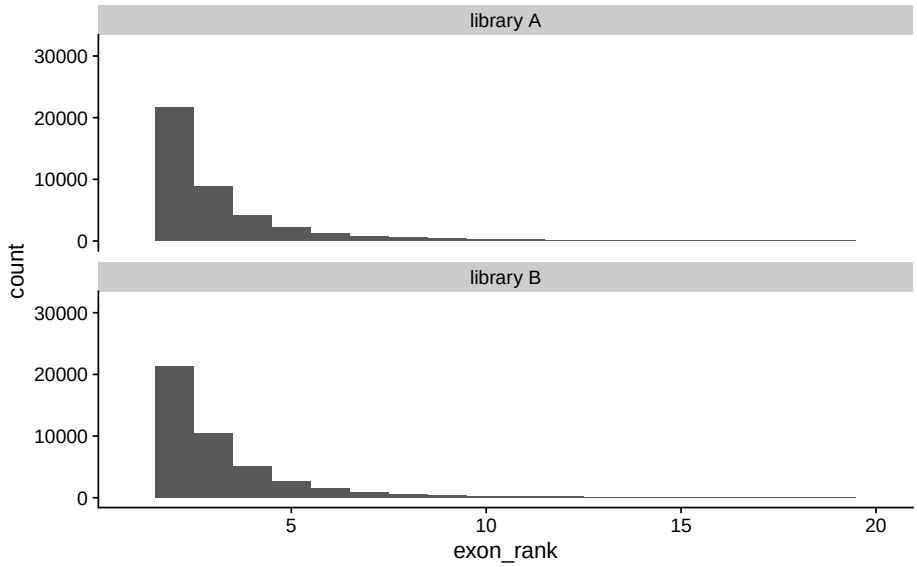


```
n_b %>% ggplot(aes(x='', y=n, fill=valid)) +
  geom_bar(stat='identity') + coord_polar('y', start = 0) +
  scale_fill_manual(values = c('#D3D3D3', '#377EB8')) +
  theme_nothing()
```

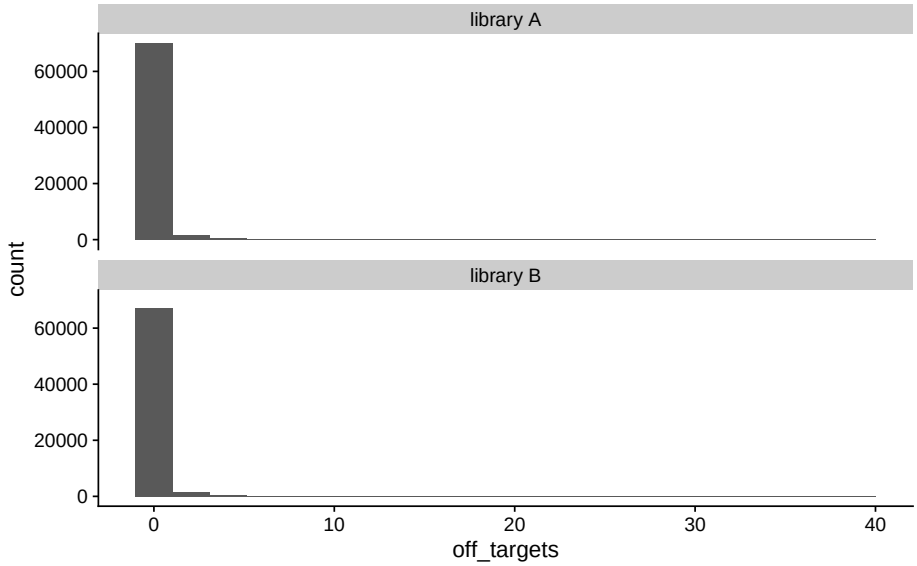


We plot the distribution of exon ranks/predicted off-targets per sgRNA.

```
## library guides with annotations
lib_df <- libA_seqs %>% dplyr::select(nopam) %>% mutate(lib = 'library A') %>%
  bind_rows(libB_seqs %>% dplyr::select(nopam) %>% mutate(lib = 'library B')) %>%
  inner_join(pool)

## exon ranks
lib_df %>%
  ggplot(aes(exon_rank)) + geom_histogram(bins=20) +
  xlim(c(1,20)) + facet_wrap(~lib, ncol=1)
```




```
## off-targets
lib_df %>% ggplot(aes(off_targets)) +
  geom_histogram(bins=20) + facet_wrap(~lib, ncol=1)
```



How is the HD CRISPR library composed of designs in previous libraries?

```
## load sequences that are also found with CLD
data('cld_seqs', package = 'HDCRISPR2019')

## generate upset plot
lib_df %>% split(.$lib) %>% walk(~{
  list(cld = cld_seqs,
     novartis = filter(.x, grepl('Novartis', in_libraries)) %>% .$nopam,
     brunello = filter(.x, grepl('Brunello', in_libraries)) %>% .$nopam,
     tkov3 = filter(.x, grepl('TKOv3', in_libraries)) %>% .$nopam,
```

```
      sabatini2014 = filter(.x, grepl('Sabatini2014', in_libraries)) %>% .$nopam,
      yusa2016 = filter(.x, grepl('Yusa2016', in_libraries)) %>% .$nopam,
      avana = filter(.x, grepl('Avana', in_libraries)) %>% .$nopam,
      tkov1 = filter(.x, grepl('TKOv1', in_libraries)) %>% .$nopam,
      gecko2 = filter(.x, grepl('GeCKOv2', in_libraries)) %>% .$nopam,
      sabatini = filter(.x, grepl('Sabatini', in_libraries)) %>% .$nopam
  ) %>% fromList() %>% upset(nsets=10)
})
```

2 Screen analysis

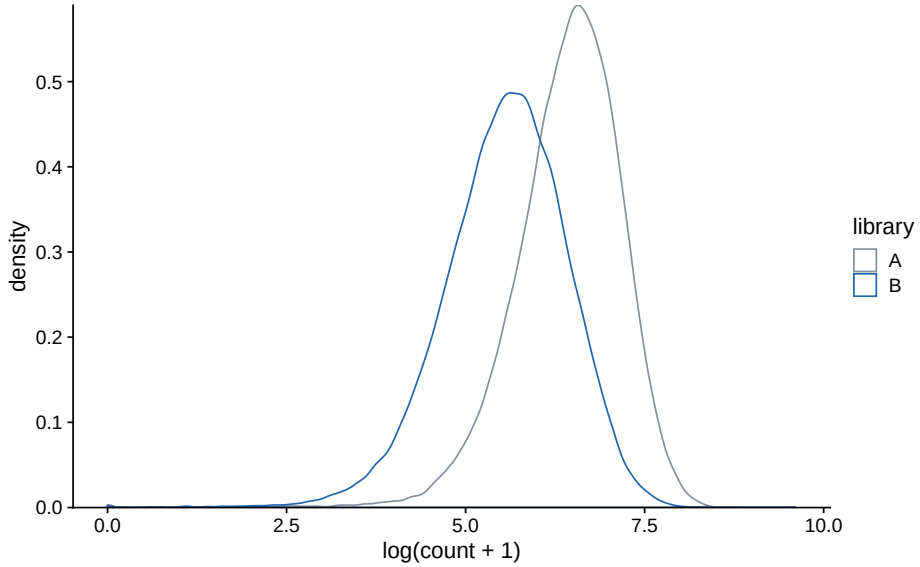
2.1 Data loading

In total we have performed 6 genome-wide dropout screens. We screened a bulk and two single cell clones with the sub-library A and B, respetively. We next load the raw count data for these experiments. The raw counts were determined from the sequencing data using Mageck-count.

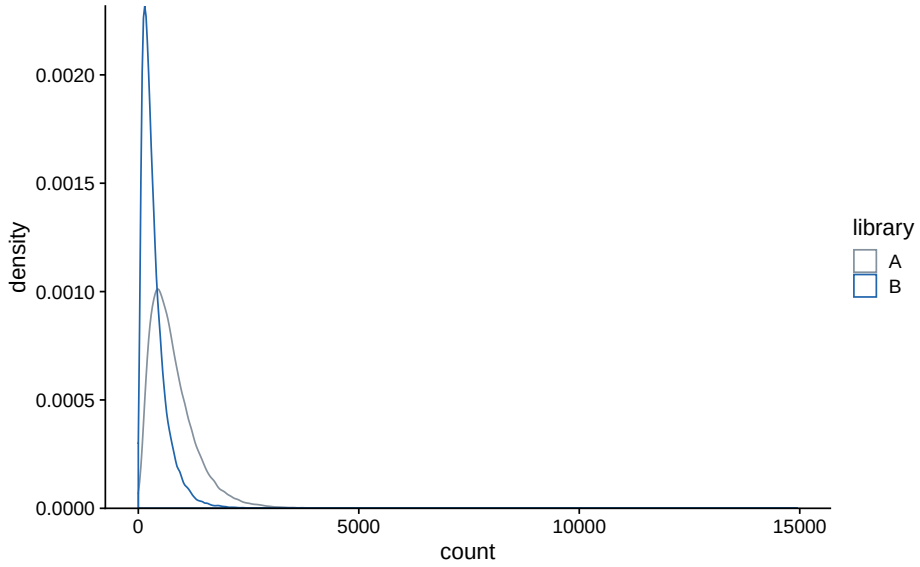
```
data('screen_counts', package = 'HDCRISPR2019')

## combine datasets and format as long df
counts_df <- counts_a %>% dplyr::select(Plasmid = plasmid_LibA, everything()) %>%
  gather(sample, count, -c(sgRNA, Gene)) %>%
  mutate(library = 'A') %>%
  bind_rows(counts_b %>% dplyr::select(-Pool_T0) %>%
              gather(sample, count, -c(sgRNA, Gene)) %>%
              mutate(library = 'B'))
```

We plot the skew ratio and check missing sgRNAs in both plasmid libraries.

```
counts_df %>% filter(sample == 'Plasmid') %>%
  ggplot(aes(log(count+1))) +
  geom_density(aes(color = library)) +
  scale_y_continuous(expand=c(0,0)) +
  scale_colour_manual(values = c('#85929E', '#2166AC'))
```




```
counts_df %>% filter(sample == 'Plasmid') %>%
  ggplot(aes(count)) +
  geom_density(aes(color = library)) +
  scale_y_continuous(expand=c(0,0)) +
  scale_colour_manual(values = c('#85929E', '#2166AC'))
```




```
## calculate skew ratio
counts_df %>% filter(sample == 'Plasmid') %>%
  group_by(library) %>%
  summarise(q10 = quantile(count, probs=0.1),
            q90 = quantile(count, probs=0.9),
            skew_ratio = q90/q10) %>% ungroup()

## how many guides are missing? which?
```

```
counts_df %>% filter(sample == 'Plasmid') %>%
  filter(count == 0)  %>% count(library)
counts_df %>% filter(sample == 'Plasmid') %>%
  filter(count == 0)
```

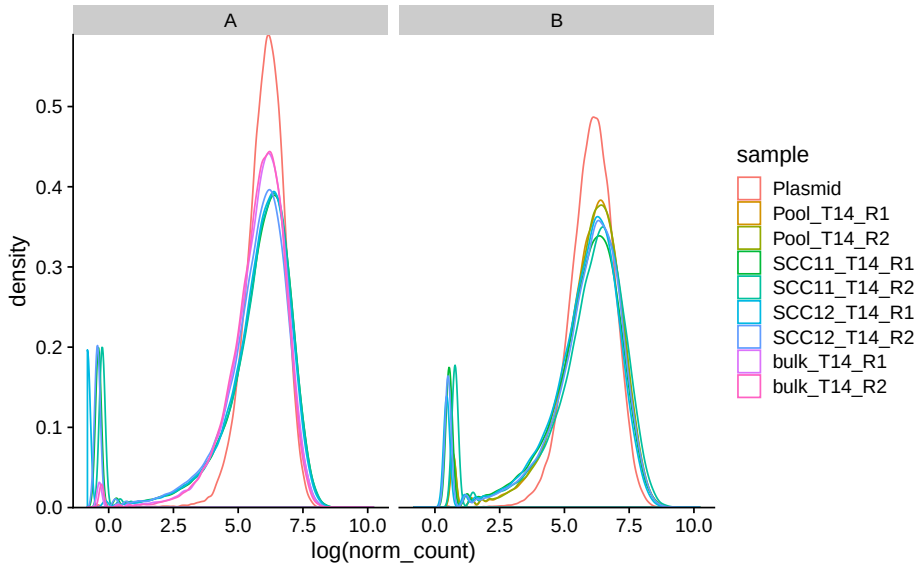
2.2 Normalization

First, we normalize each sample by dividing through the median count of all targeting controls. Before we do that we add a global pseudo count to avoid dividing by 0.

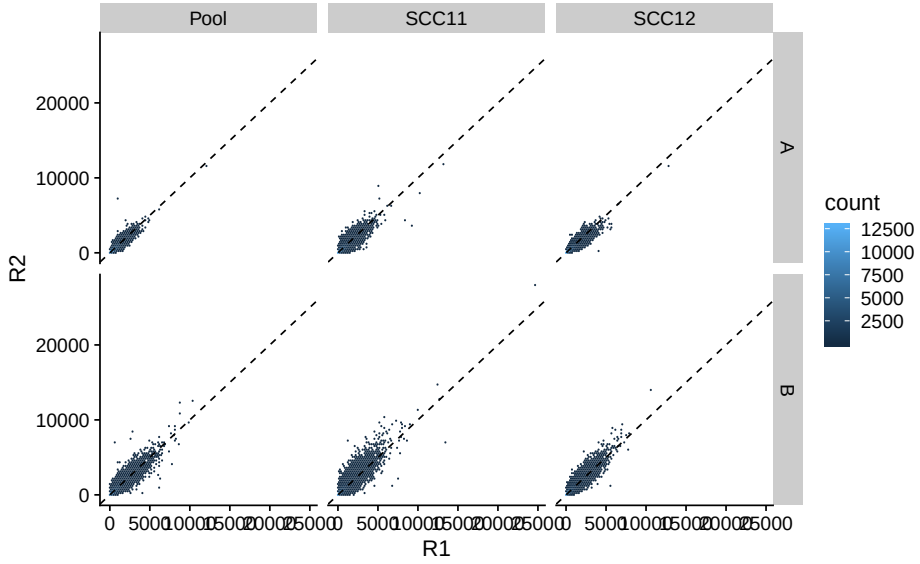
```
## list of non-targeting CTRLs
nontarg <- c('NONTARG', 'LacZ', 'EGFP', 'luciferase')
ctrls <- counts_df %>% distinct(sgRNA, Gene) %>%
  filter(grepl('CONTROL', sgRNA)) %>%
  mutate(ctrl_type = ifelse(Gene %in% nontarg, 'nontarg', 'targ'))

## median of targeting controls per library
targ_med <- counts_df %>%
  filter(sgRNA %in% (filter(ctrls, ctrl_type == 'targ') %>% pull(sgRNA))) %>%
  group_by(library, sample) %>%
  summarise(med_count = median(count + 1)) %>% ungroup()

## perform normalization
norm_df <- counts_df %>% inner_join(targ_med) %>%
  mutate(norm_count = ((count+1)/med_count)*median(count+1))
```

We plot log-scaled distributions of normalized counts for each sample. We also plot correlations of normalized counts between replicates.

```
## norm-count [log] distributions per sample
norm_df %>% ggplot(aes(log(norm_count))) +
  geom_density(aes(colour = sample)) +
  facet_wrap(~library) +
  scale_y_continuous(expand=c(0, 0))
```




```
## reproducibility at norm-count level
df_samples <- norm_df %>% filter(sample != 'Plasmid') %>%
  separate(sample, c('type', 'time', 'rep'), sep='_', remove=F) %>%
  dplyr::select(sgRNA, library, type, rep, norm_count) %>%
  mutate(type = ifelse(type == 'bulk', 'Pool', type))

df_samples %>% spread(rep, norm_count) %>%
  ggplot(aes(R1, R2)) + geom_hex(bins=100) +
  geom_abline(linetype = 'dashed') +
  facet_grid(library ~ type)
```




```
## correlation coefficients
df_samples %>% spread(rep, norm_count) %>%
  group_by(library, type) %>%
```

```
  summarise(PCC = cor(R1, R2, method='pearson'),
            SCC = cor(R1, R2, method='spearman')) %>% ungroup()
```

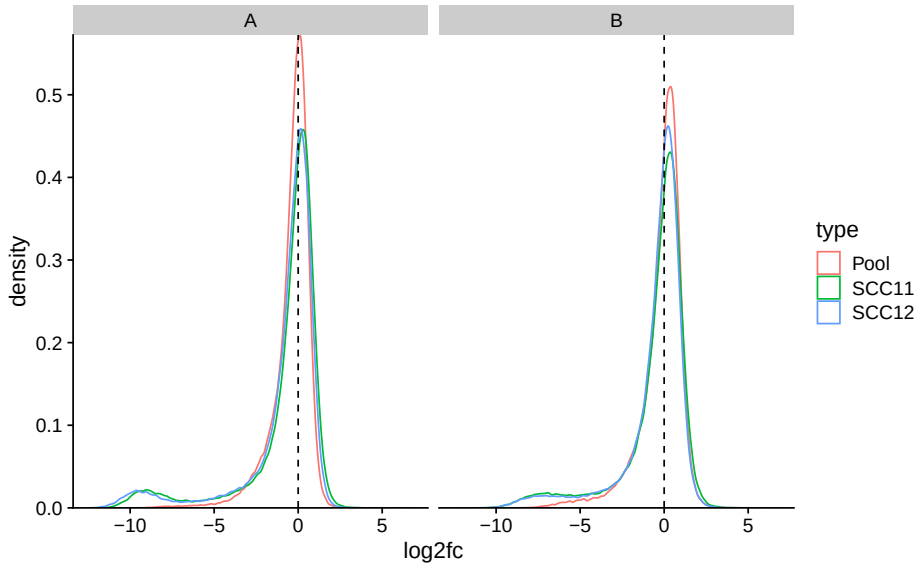
2.3 Fold changes

We next calculate log2-scaled fold changes for each sample by dividing log2-scaled normalized counts for each sample by the log2-scaled normalized counts of the correpsonding plasmid libraries.

```
## get plasmid counts
plasmid_counts <- norm_df %>% filter(sample == 'Plasmid') %>%
  distinct(library, sgRNA, norm_count) %>%
  dplyr::select(plasmid_count = norm_count, everything())

## calculate log-fold-changes
fold_changes <- df_samples %>% inner_join(plasmid_counts) %>%
  mutate(log2fc = log2(norm_count) - log2(plasmid_count)) %>%
  ungroup()
```

We can now plot fold-change distributions for each of the samples.

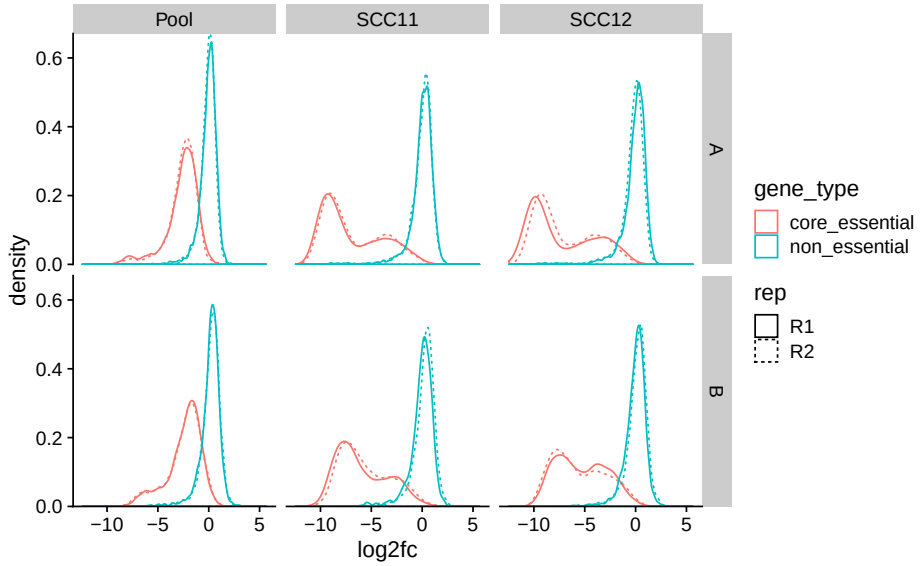
```
fold_changes %>% ggplot(aes(log2fc)) +
  geom_density(aes(colour = type)) + facet_wrap(~ library) +
  scale_y_continuous(expand=c(0,0)) +
  geom_vline(xintercept = 0, linetype = 'dashed')
```



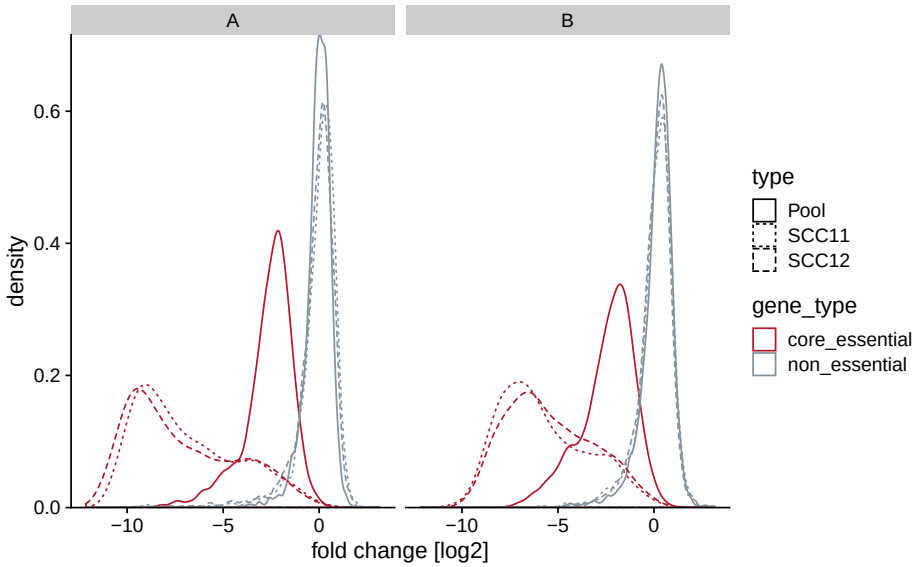
In addition we can compare dropout (quantified as log2-scaled fold changes) of core-essential genes compared to non-essential genes in each sample. We do this both for each replicate separately and for merged replicates (by averaging).

```
## data frame with target ENSG and gene symbol for each sgRNA
sgrna_targets <- counts_df %>% distinct(sgRNA, Gene) %>%
```

```
  left_join(distinct(lib_df, genes_hit, ensg) %>%
              dplyr::select(Gene=ensg, symbol=genes_hit))

## annotate fold changes with target gene info
fold_changes <- fold_changes %>% left_join(sgrna_targets)

## select core-/non-essential genes
cene <- fold_changes %>% filter(symbol %in% c(ce$symbol, ne$symbol)) %>%
  mutate(gene_type = ifelse(symbol %in% ce$symbol, 'core_essential', 'non_essential'))

## plot fold changes of CEG vs. NEG
cene %>% ggplot(aes(log2fc, colour = gene_type, linetype = rep)) +
  geom_density() + facet_grid(library ~ type) +
  scale_y_continuous(expand = c(0,0))
```

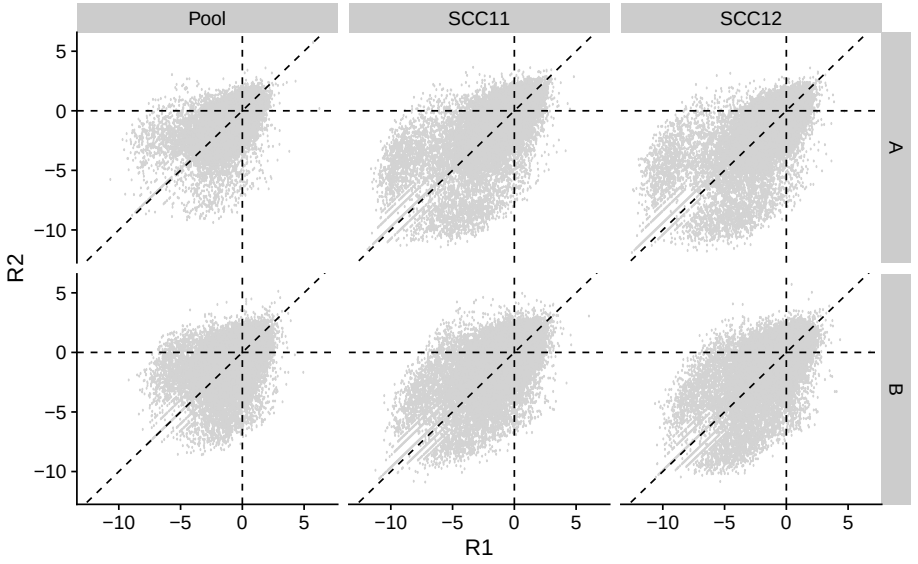



```
## same, but merging replicates
cene %>% group_by(library, type, gene_type, sgRNA) %>%
  summarise(log2fc = mean(log2fc)) %>% ungroup() %>%
  ggplot(aes(log2fc, colour = gene_type)) +
  geom_density(aes(linetype = type)) + facet_grid(~library) +
  scale_y_continuous(expand = c(0,0)) +
  scale_colour_manual(values = c('#B2182B', '#85929E')) +
  xlab('fold change [log2]')
```



Finally we can also look at the reproducibility between replicates at the log2-fold change level.

```
## scatter plots
fold_changes %>%
  dplyr::select(library, sgRNA, type, log2fc, rep) %>%
  spread(rep, log2fc) %>% ggplot(aes(R1, R2)) +
  geom_point_rast(colour = '#D3D3D3') +
  facet_grid(library ~ type) +
  geom_abline(linetype='dashed') +
  geom_vline(xintercept = 0, linetype = 'dashed') +
  geom_hline(yintercept = 0, linetype = 'dashed')
```




```
## correlation coefficients
fold_changes %>%
  dplyr::select(library, sgRNA, type, log2fc, rep) %>%
```

```
  spread(rep, log2fc) %>%
  group_by(library, type) %>%
  summarise(PCC = cor(R1, R2, method='pearson'),
            SCC = cor(R1, R2, method='spearman')) %>%
  ungroup()
```

2.4 Hit calling with BAGEL

We use BAGEL to calculate Bayes Factors for each screen to determine essential genes. We run the tool separately for each sample type and each library. We also run the tool for each sample type based on combined libraries (such that we get 8 sgRNAs per sample). As input files we must first generate tables of log2-fold changes that we can feed into BAGEL.

```
## file for library A
fold_changes %>% filter(library == 'A') %>%
  unite(sample, type, rep) %>%
  dplyr::select(sgRNA, symbol, sample, log2fc) %>%
  spread(sample, log2fc) %>% filter(!is.na(symbol)) %>%
  write_tsv('HDCRISPR_A_fold_changes.txt')

## file for library B
fold_changes %>% filter(library == 'B') %>%
  unite(sample, type, rep) %>% dplyr::select(sgRNA, symbol, sample, log2fc) %>%
  spread(sample, log2fc) %>% filter(!is.na(symbol)) %>%
  write_tsv('HDCRISPR_B_fold_changes.txt')

## one for combined libraries
fold_changes %>% filter(!grepl('CONTROL', sgRNA), !is.na(symbol)) %>%
  unite(sample, type, rep) %>%
  dplyr::select(sgRNA, symbol, sample, log2fc) %>%
  spread(sample, log2fc) %>%
  write_tsv('HDCRISPR_combined_fold_changes.txt')
```

Now we can now execute BAGEL outisde of R to calculate Bayes Factors for each sample from the fold changes that we wrote. We use the CEG2 and NEG gene sets as training sets. After running BAGEL we can read the results back into R.

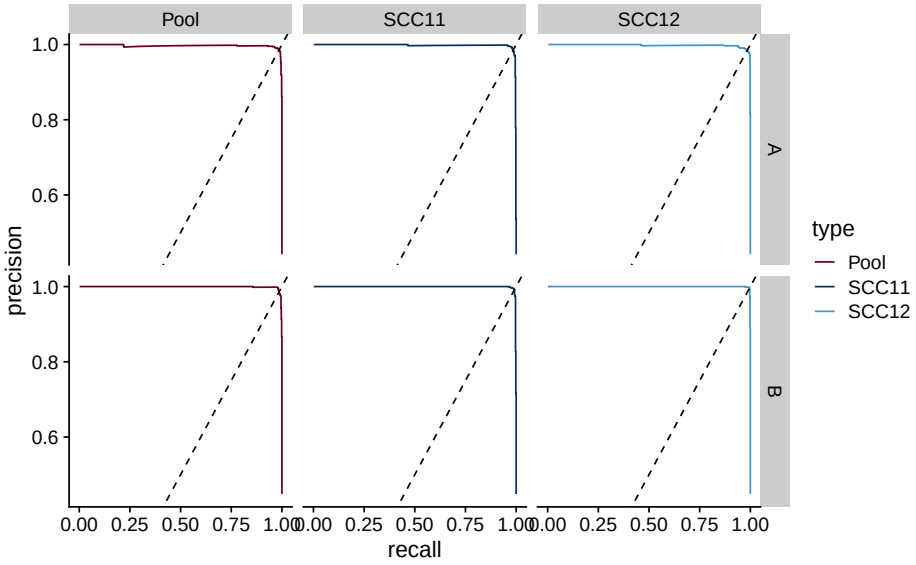
```
## load BAGEL bayes factors
data('bfs', package = 'HDCRISPR2019')
```

2.4.1 Precision-recall analysis

We generate PR curves for each screen.

```
## calculate curve data
pr_curves <- bfs %>%
  mutate(gene_type = ifelse(GENE %in% ce$symbol, 'core-essential',
                     ifelse(GENE %in% ne$symbol, 'non-essential', 'other'))) %>%
  filter(gene_type != 'other') %>%
```

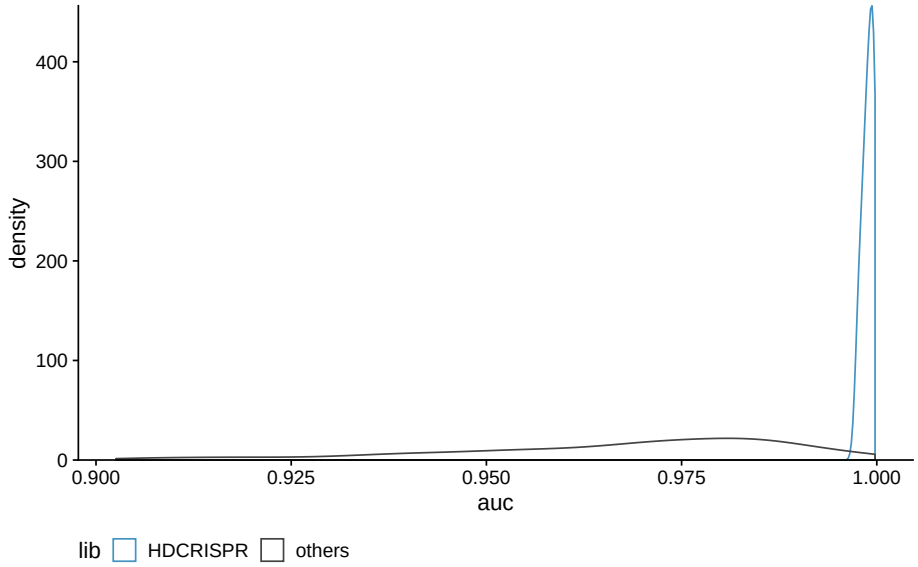
```
  group_by(type, library) %>%
  group_modify(~{
    pred <- prediction(.x$BF,
                       labels = ifelse(.x$gene_type == 'core-essential', 1, 0))
    perf <- performance(pred, measure = 'prec', x.measure = 'rec')
    auc <- performance(pred, measure = 'auc')@y.values[[1]]
    fdr5 <- perf@alpha.values[[1]][max(which(perf@y.values[[1]] > 0.97))]
    tibble(
      prec = perf@y.values[[1]],
      rec = perf@x.values[[1]],
      co = fdr5,
      auc = auc
    )
  }) %>% ungroup()

## draw curves
pr_curves %>% filter(library != 'combined') %>%
  ggplot(aes(rec, prec)) + geom_line(aes(colour = type)) +
  facet_grid(library~type) +
  geom_abline(linetype = 'dashed') +
  xlab('recall') + ylab('precision') +
  scale_colour_manual(values = c('#67001F', '#053061', '#4393C3'))
```



We compare AUC values to screens in previously published libraries.

```
## AUCs for screens with the HD CRISPR lib.
hdcrispr_auc <- distinct(pr_curves, type, library, auc)

## AUCs in other screens selected for design.
genomecrispr_auc <- roc_results %>% bind_rows() %>%
  distinct(pubmed, cellline, condition, auc) %>%
  filter(auc > 0.9)
```

```
## compare aucs between both groups
hdcrispr_auc %>% mutate(lib = 'HDCRISPR') %>%
  bind_rows(genomecrispr_auc %>%
              mutate(lib = 'others')) %>%
  ggplot(aes(auc, colour = lib)) +
  geom_density() +
  scale_y_continuous(expand = c(0,0)) +
  scale_colour_manual(values = c('#4393C3', '#444444')) +
  theme(legend.position = 'bottom')
```



We determine whether a gene is considered 'essential' or not based on a BF $>$ 6 threshold.

```
bfs <- bfs %>% mutate(essential = ifelse(BF > 6, T, F))
```

2.4.2 Comparison with previous Hap1 screens

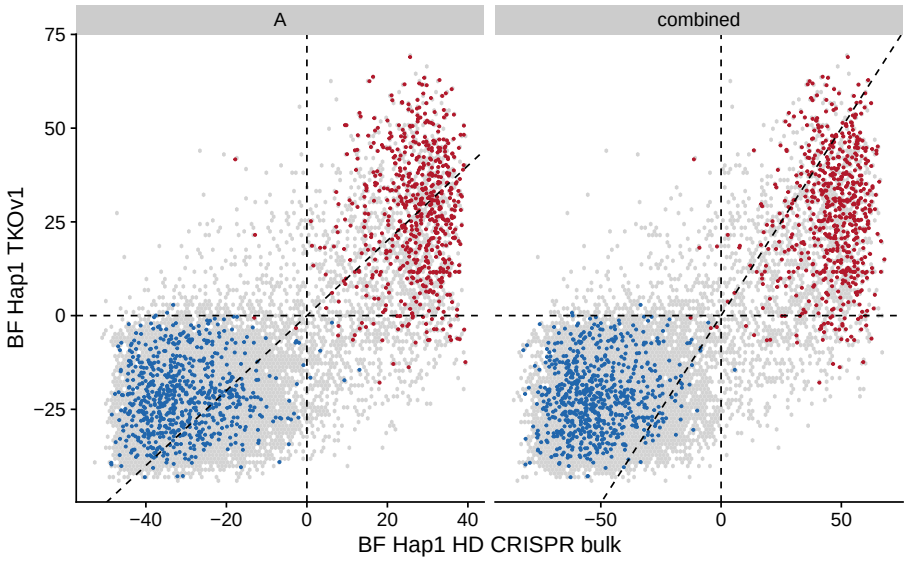
Genome-scale Hap1 gene dropout screens have been conducted previously with the TKOv3 library (Hart et al., 2017) and with the GeneTrap system. We can compare our results to those obtained in these screens. For the TKOv3 screen, Bayes Factors are available so that's what we will look at as gene-level pheontype scores.

```
## previous screens in hap1 cells (gene trap and tkov1)
data('previous_hap1_screens', package = 'HDCRISPR2019')
```

We create scatter plots to see whether Bayes Factors for our screens correlate with the previously published HAP1 screens.

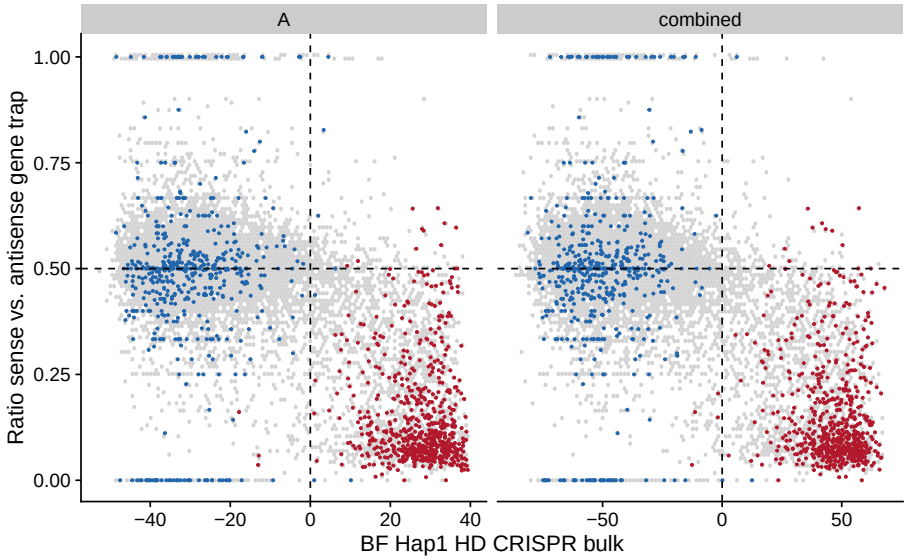
```
## datasets
comp_tko <- bfs %>% filter(type == 'Pool', library != 'B') %>%
  dplyr::select(GENE, BF, library) %>%
  inner_join(tko1_bf_hap1)
```

```
comp_genetrap <- bfs %>% filter(type == 'Pool', library != 'B') %>%
  dplyr::select(GENE, BF, library) %>%
  inner_join(genetrap_ess %>% dplyr::select(-q.val))

## scatter plots
comp_tko %>% ggplot(aes(BF, BF_HAP1)) +
  geom_hex(bins=100, fill = '#d3d3d3') +
  geom_point(data = subset(comp_tko, GENE %in% ce$symbol),
             aes(BF, BF_HAP1), colour = '#B2182B', size=0.5) +
  geom_point(data = subset(comp_tko, GENE %in% ne$symbol),
             aes(BF, BF_HAP1), colour = '#2166AC', size=0.5) +
  facet_wrap(~ library, scales = 'free_x') + geom_abline(linetype = 'dashed') +
  geom_hline(yintercept = 0, linetype = 'dashed') +
  geom_vline(xintercept = 0, linetype = 'dashed') +
  xlab('BF Hap1 HD CRISPR bulk') + ylab('BF Hap1 TKOv1')
```




```
## correlation coefficients
comp_tko %>% drop_na() %>%
  group_by(library) %>%
  summarise(PCC = cor(BF, BF_HAP1),
            SCC = cor(BF, BF_HAP1, method='spearman')) %>%
  ungroup()

## same for bloomen
comp_genetrap %>% ggplot(aes(BF, ratio)) +
  geom_hex(bins=100, fill = '#d3d3d3') +
  geom_point(data = subset(comp_genetrap, GENE %in% ce$symbol),
             aes(BF, ratio), colour = '#B2182B', size=0.5) +
  geom_point(data = subset(comp_genetrap, GENE %in% ne$symbol),
             aes(BF, ratio), colour = '#2166AC', size=0.5) +
  facet_wrap(~ library, scales='free_x') +
  geom_hline(yintercept = 0.5, linetype = 'dashed') +
```

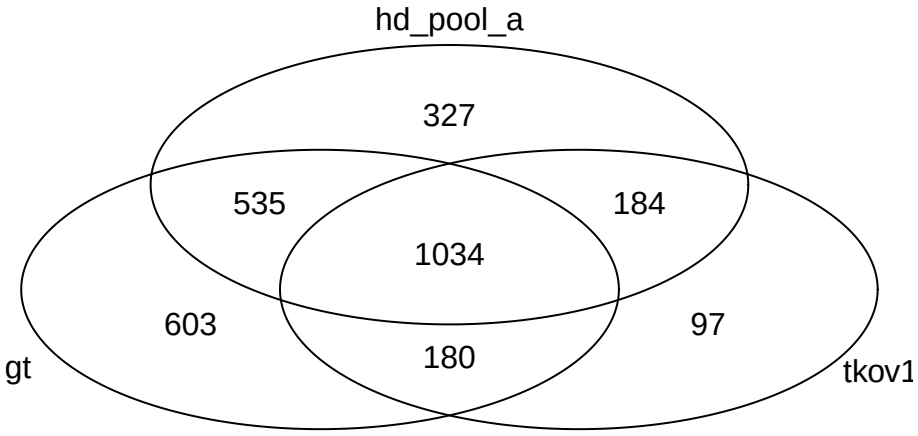
```
  geom_vline(xintercept = 0, linetype = 'dashed') +
  xlab('BF Hap1 HD CRISPR bulk') + ylab('Ratio sense vs. antisense gene trap')
```



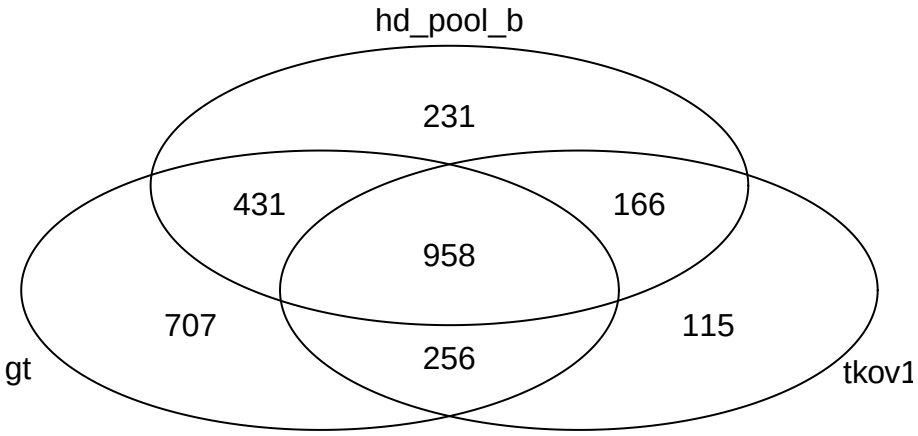

```
comp_genetrap %>% drop_na() %>%
  group_by(library) %>%
  summarise(PCC = cor(BF, ratio),
            SCC = cor(BF, ratio, method='spearman')) %>%
  ungroup()
```

We can also make a Venn diagram. We make two lists, representing libraries A and B, respectively. Accordingly, we make two Venn diagrams comparing screens in these libraries with existing experiments.

```
## HAP1 TKOv1 cutoff selected as described in Hart et al, 2017
venn_a <- list(
  gt = genetrap_ess %>% filter(q.val < 0.05) %>% pull(GENE),
  tkov1 = tko1_bf_hap1 %>% filter(BF_HAP1 > 6) %>% pull(GENE),
  hd_pool_a = bfs %>%
    filter(library == 'A', essential, type == 'Pool') %>% pull(GENE)
)

## HAP1 TKOv1 cutoff selected as described in Hart et al, 2017
venn_b <- list(
  gt = genetrap_ess %>% filter(q.val < 0.05) %>% pull(GENE),
  tkov1 = tko1_bf_hap1 %>% filter(BF_HAP1 > 6) %>% pull(GENE),
  hd_pool_b = bfs %>%
    filter(library == 'B', essential, type == 'Pool') %>% pull(GENE)
)

## plot venn diagrams
gplots::venn(venn_a)
```



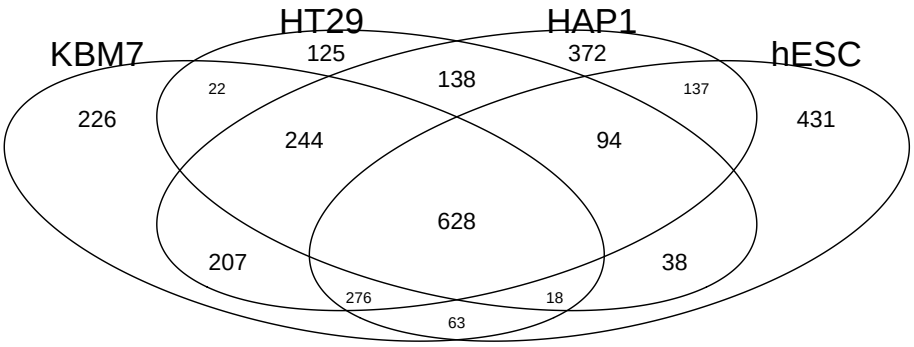

```
gplots::venn(venn_b)
```



We can also compare Hap1 cells to other cell lines. Specifically we compare Hap1 cells to KBM7 cells, previously published screens in human embryonic stem cells (Yilmaz et al., 2018 in Nat. Cell. Biol.) and then maybe some random cancer cell lines.

```
## read gene-level crispr phenotyeps in hesc cells
data('hesc_crispr', package = 'HDCRISPR2019')

## venn diagram comparing different lines
venn_comp <- list(
  ## kbm7 essential genes
  KBM7 = bagel_results %>% bind_rows() %>%
    filter(cellline == 'KBM7',
           pubmed != '24336569',
           condition == 'viability') %>%
    filter(BF > 6) %>% pull(GENE),
  ## HCT116
  HT29 = bagel_results %>% bind_rows() %>%
    filter(cellline == 'HT29',
           condition == 'viabilityafter25days') %>%
    filter(BF > 6) %>% pull(GENE),
  ## HAP1
  HAP1 = bfs %>%
```

```
    filter(library == 'combined', type == 'Pool') %>%
    filter(BF > 6)  %>% pull(GENE),
  ## hESC
  hESC = hesc_crispr %>% filter(cscore < 0, FDR < 0.05) %>% pull(symbol)
)

## draw venn
gplots::venn(venn_comp)
```

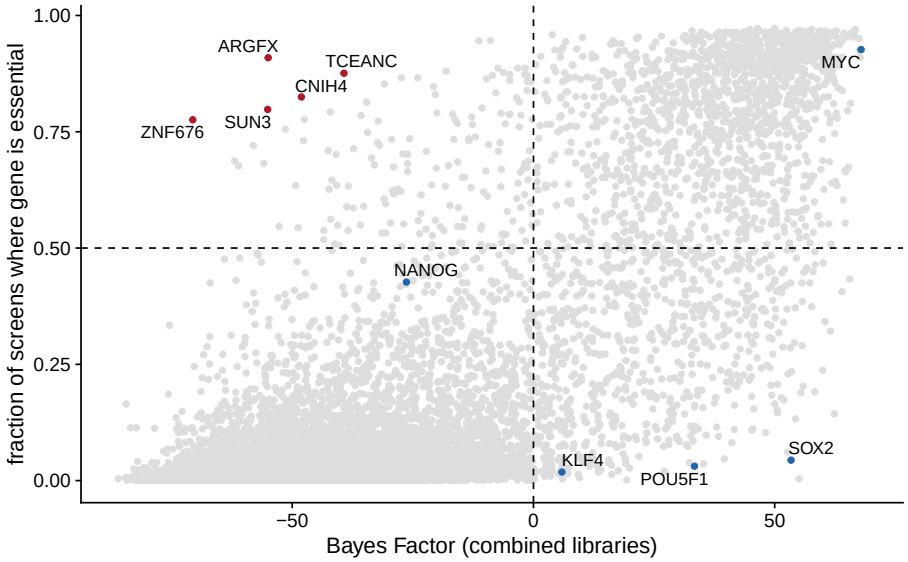


2.5 Context-dependent essential genes

We compare every hit in our screen with the fraction of previous screens that the gene showed a phenotype in. To this end we load pre-computed values indicating for each gene the fraction of cell lines that it is essential in.

```
## add promiscuity score
data('pscores', package = 'HDCRISPR2019')

## highlight yamanaka
yamanaka <- c('NANOG', 'MYC', 'POU5F1', 'KLF4', 'SOX2')
off_targets <- c('ARGFX', 'CNIH4', 'ZNF676', 'SUN3', 'TCEANC')

plot_df <- bfs %>% left_join(pscores) %>%
  filter(library == 'combined', type == 'Pool')

ggplot() +
  geom_point(data = plot_df, aes(BF, pscore),
             colour = '#dddddd') +
  geom_point(data = filter(plot_df, GENE %in% yamanaka),
             aes(BF, pscore), colour = '#2166AC') +
  geom_point(data = filter(plot_df, GENE %in% off_targets),
             aes(BF, pscore), colour = '#B2182B') +
  geom_text_repel(data = filter(plot_df,
                                GENE %in% c(yamanaka, off_targets)),
                  aes(BF, pscore, label = GENE)) +
  geom_vline(xintercept = 0, linetype = 'dashed') +
  geom_hline(yintercept = 0.5, linetype = 'dashed') +
  xlab('Bayes Factor (combined libraries)') +
```

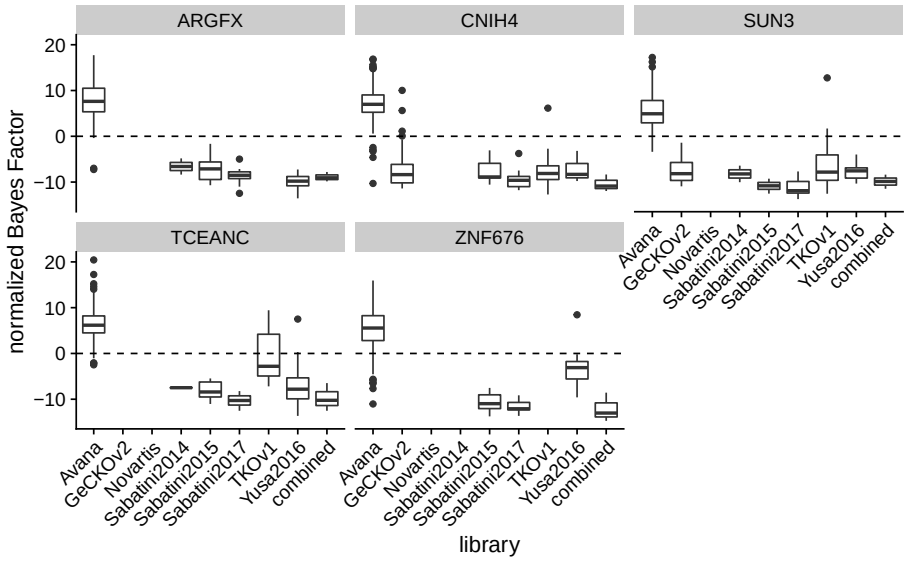
```
  ylab('fraction of screens where gene is essential')
```



The genes that are essential in most previous screens in cancer cell lines but are not found with the HD CRISPR library are mostly unknown / uncharacterized genes. These might be off-target effects in the Avana library that has been used for most screens. We make a plot comparing the Bayes Factors for these genes across different libraries that were used to design the HD CRISPR library.

```
## annotate library
lib_anno <- tibble(
  pubmed = c('26472758', '26627737', '24336569', '27260157', '26780180',
             '27760321', '27260156', '27869803', '28145866', '28162770',
             '29083409', '28700943'),
  library= c('Sabatini2015', 'TKOv1', 'Sabatini2014', 'Novartis', NA,
             'Yusa2016', 'GeCKOv2', 'TKOv1', 'GeCKOv2', 'Sabatini2017',
             'Avana', 'Sabatini2015')
)

## normalize bayes factors
bf_mat <- bind_rows(bagel_results) %>% inner_join(lib_anno) %>%
  unite(screen, cellline, condition, library) %>%
  dplyr::select(GENE, BF, screen) %>%
  bind_rows(bfs %>% filter(library == 'combined') %>%
              mutate(screen = paste('HAP1', type, library, sep='_')) %>%
              dplyr::select(GENE, BF, screen)) %>%
  reshape2::acast(GENE ~ screen, value.var = 'BF')

## quantile normalization
bf_mat_norm <- normalize.quantiles(bf_mat)
colnames(bf_mat_norm) <- colnames(bf_mat)
rownames(bf_mat_norm) <- rownames(bf_mat)

## draw box plot to see whether phenotypes are library specific
```

```
  as_tibble(bf_mat_norm, rownames='GENE') %>%
    filter(GENE %in% off_targets) %>% gather(screen, BF, -GENE) %>%
    filter(!grepl('KRAS', screen)) %>%
    separate(screen, c('cellline', 'condition', 'library')) %>%
    filter(library != 'NA') %>%
    ggplot(aes(library, BF)) + geom_boxplot() +
    facet_wrap(~GENE) +
    geom_hline(yintercept = 0, linetype = 'dashed') +
    theme(axis.text.x = element_text(angle=45, hjust=1)) +
    ylab('normalized Bayes Factor')
```



2.5.1 HAP1 specific essential genes

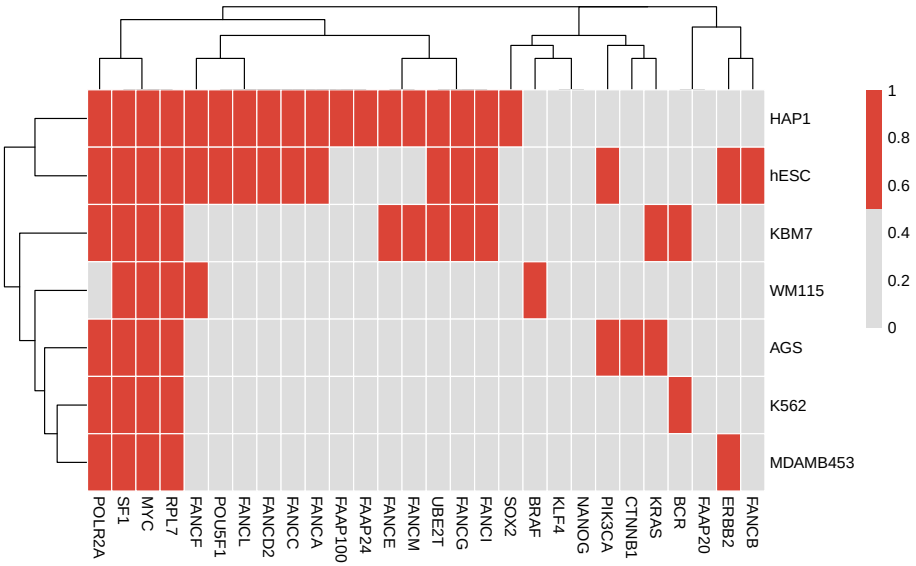
In the above scatter plot we saw context-specific Hap1 essential genes in the bottom right corner, and broadly essential genes that do not lead to a phenotype in HAP1 cells in the top left corner. We can pick a cutoff to select genes in the bottom right corner (HAP1 specific essential genes) and perform a gene set enrichment analysis to check whether there are overrepresented pathways.

```
  ## list of hap1 essentials ordered by specificity
  hap1_specific <- plot_df %>% filter(BF > 6, pscore < 0.4)

  ## convert gene symbol to entrez id
  data('symbol_to_entrez', package = 'HDCRISPR2019')

  ## add entrez to hap1 essentials
  hap1_specific <- hap1_specific %>%
    inner_join(symbol_to_entrez %>% dplyr::select(GENE=symbol, entrez)) %>%
    arrange(BF)

  ## perform gene set overrepresentation analysis on hap1-specific essentials
  goa_out <- limma::kegga(hap1_specific$entrez,
```

```
                    universe = symbol_to_entrez %>%
                      filter(symbol %in% plot_df$GENE) %>%
                      pull(entrez) %>% unique(),
                    species = 'Hs')
top_go <- limma::topKEGG(goa_out)
```

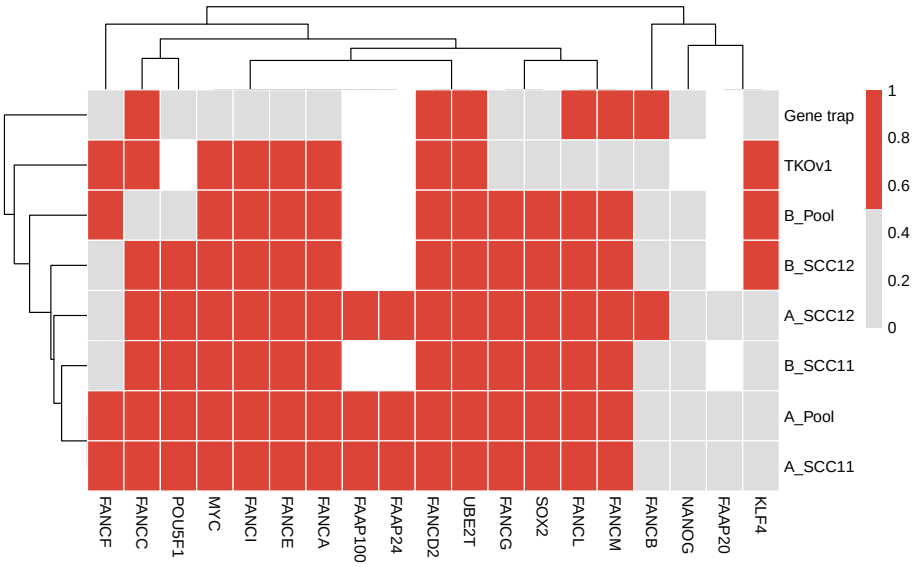
This reveals a strong enrichment for regulators of the Fanconi Anemia pathway. We make a heatmap that shows certain genes of interest and their essentiality across a number of cell lines.

```
## hap1-specific essential genes
hap1_specific <- bfs %>%
  filter(library == 'combined', type == 'Pool') %>%
  dplyr::select(GENE, BF) %>% mutate(cellline = 'HAP1')

## essential genes for a number of other cell lines
other_cl <- bagel_results %>% bind_rows() %>%
  filter(cellline %in% c('KBM7', 'K562', 'WM115', 'AGS', 'MDAMB453')) %>%
  filter(pubmed %in% c('26472758', '29083409')) %>%
  dplyr::select(GENE, BF, cellline)

## gene sets
yamanaka <- c('MYC', 'POU5F1', 'KLF4', 'SOX2', 'NANOG')
oncogenes <- c('KRAS', 'BRAF', 'PIK3CA', 'ERBB2', 'BCR', 'CTNNB1')
core_ess <- c('POLR2A', 'RPL7', 'SF1')
fanc <- c('FANCA', 'FANCB', 'FANCC', 'FANCE', 'FANCF', 'FANCG',
          'FANCL', 'FANCM', 'FAAP20', 'FAAP100', 'FAAP24',
          'UBE2T', 'FANCD2', 'FANCI')

## draw essentiality heat map
hap1_specific %>% bind_rows(other_cl) %>%
  mutate(ess = ifelse(BF > 6, 1, 0)) %>%
  bind_rows(hesc_crispr %>%
              mutate(ess = ifelse(cscore < 0 & FDR < 0.05, 1, 0),
                     cellline = 'hESC') %>%
              dplyr::select(GENE=symbol, ess, cellline)) %>%
  filter(GENE %in% c(yamanaka, oncogenes, core_ess, fanc)) %>%
  reshape2::acast(cellline ~ GENE, value.var = 'ess') %>%
  pheatmap::pheatmap(color = c('#dddddd', '#db4437'),
                     border_color = '#ffffff')
```



Do we see the Yamanaka and Fanconi Anemia dropouts in other Hap1 screens, too?

```
genetrap_ess %>%
  mutate(ess = ifelse(q.val < 0.05, 1, 0),
         screen = 'Gene trap') %>%
  dplyr::select(symbol = GENE, ess, screen) %>%
  bind_rows(
    tko1_bf_hap1 %>%
      mutate(ess = ifelse(BF_HAP1 > 6, 1, 0),
             screen = 'TKOv1') %>%
      dplyr::select(symbol = GENE, ess, screen)
  ) %>%
  bind_rows(
    bfs %>% filter(library != 'combined') %>%
      unite(screen, library, type) %>%
      dplyr::select(ess = essential, screen, symbol = GENE) %>%
      mutate(ess = ifelse(ess, 1, 0))
  ) %>%
  filter(symbol %in% c(yamanaka, fanc)) %>%
  acast(screen ~ symbol, value.var = 'ess') %>%
  pheatmap(color = c('#dddddd', '#db4437'),
           border_color = '#ffffff',
           na_col = '#ffffff')
```

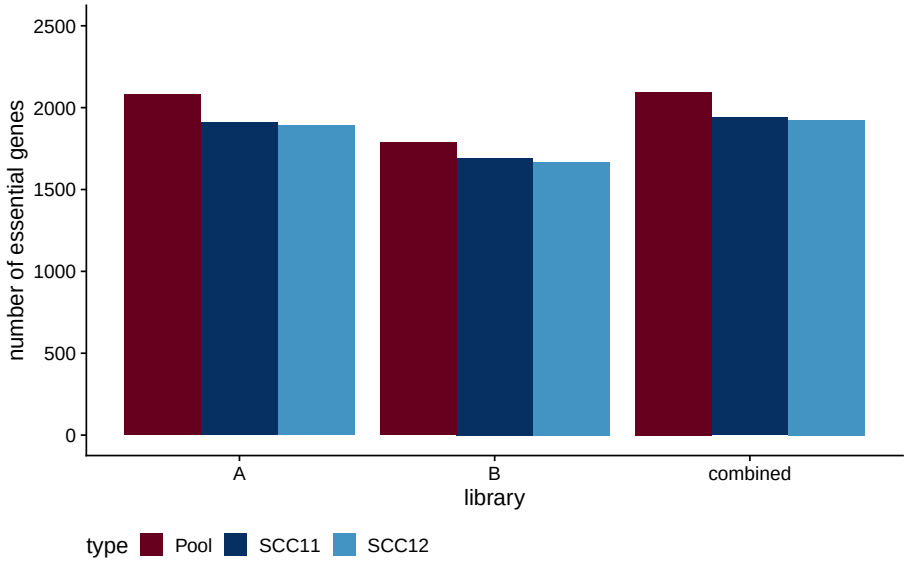


2.6 Hit calling with individual sub-libraries compared to the combined library

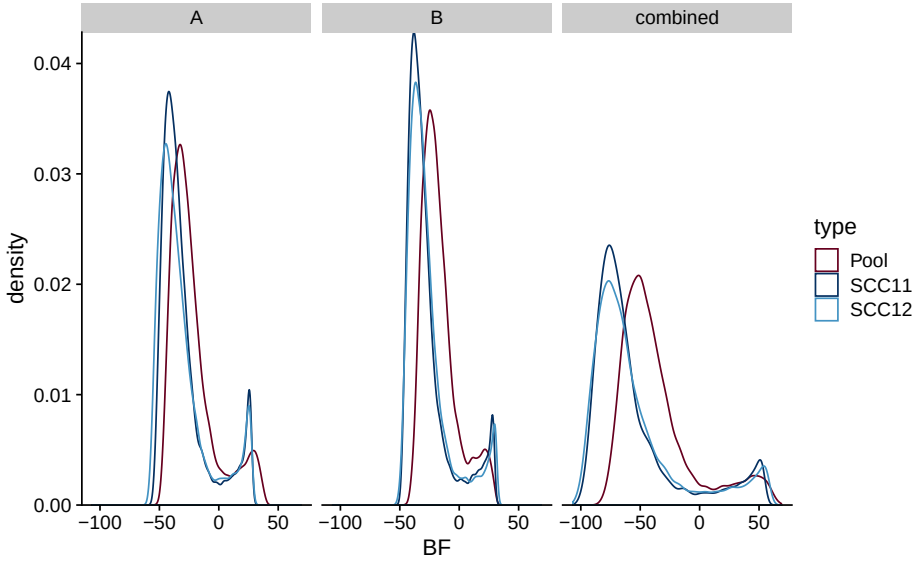
2.6.1 BAGEL

We want to observe whether we achieve increased performance, when we combine both libraries for hit calling. We first have a look at the BAGEL results.

```
## use BF > 6 as cutoff like in Hart et al.
bfs %>% mutate(essential = ifelse(BF > 6, T, F)) %>%
  dplyr::count(library, type, essential) %>% filter(essential) %>%
  ggplot(aes(library, n, fill = type)) +
  geom_bar(stat='identity', position='dodge') +
  ylab('number of essential genes') +
  ylim(c(0,2500)) +
  scale_fill_manual(values = c('#67001F', '#053061', '#4393C3')) +
  theme(legend.position = 'bottom')
```




```
## plot bf distributions for each screen
bfs %>% ggplot(aes(BF, colour = type)) +
  geom_density() + facet_wrap(~library) +
  scale_y_continuous(expand = c(0,0)) +
  scale_color_manual(values = c('#67001F', '#053061', '#4393C3'))
```

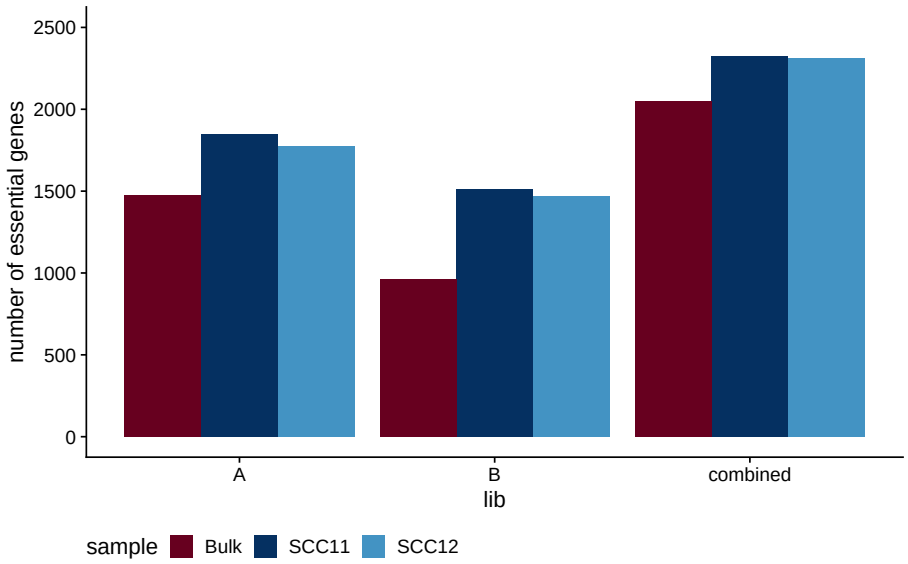


2.6.2 Mageck RRA

Another commonly used pipeline that is frequently used to analyze CRISPR screens and does not require prior information in the form of core- and non-essential genes is Mageck RRA. We run this algorithm on both individual and combined libraries to see how many hits can be identified.

We first have to run MAGeCK on libraries A, B and the combined set. We do this outside of R using the MAGeCK command line tool with default parameters. We can then load the results back into R for visualization. We create similar bar graphs as we did for the BAGEL results above, using a 5% FDR cutoff to determine essential genes.

```
## load combined mageck results
data('mageck_res', package = 'HDCRISPR2019')

## similar bar graphs as we made for the BAGEL analysis
mageck_res %>% mutate(essential = ifelse(FDR < 0.05 & lfc < 0, T, F)) %>%
  filter(essential) %>% dplyr::count(lib, sample) %>%
  ggplot(aes(lib, n, fill=sample)) +
  geom_bar(stat='identity', position = 'dodge') +
  ylim(c(0, 2500)) +
  theme(legend.position = 'bottom') +
  scale_fill_manual(values = c('#67001F', '#053061', '#4393C3')) +
  ylab('number of essential genes')
```



2.6.3 gscreend (Imkeller et al.)

gscreend can model the assymetry in CRISPR dropout screens, which might detect more essential genes than MAGeCK at low library coverage. We run gscreend on all HD CRISPR library screens. The code below takes some time to run. Therefore we also include precomputed results with this R package.

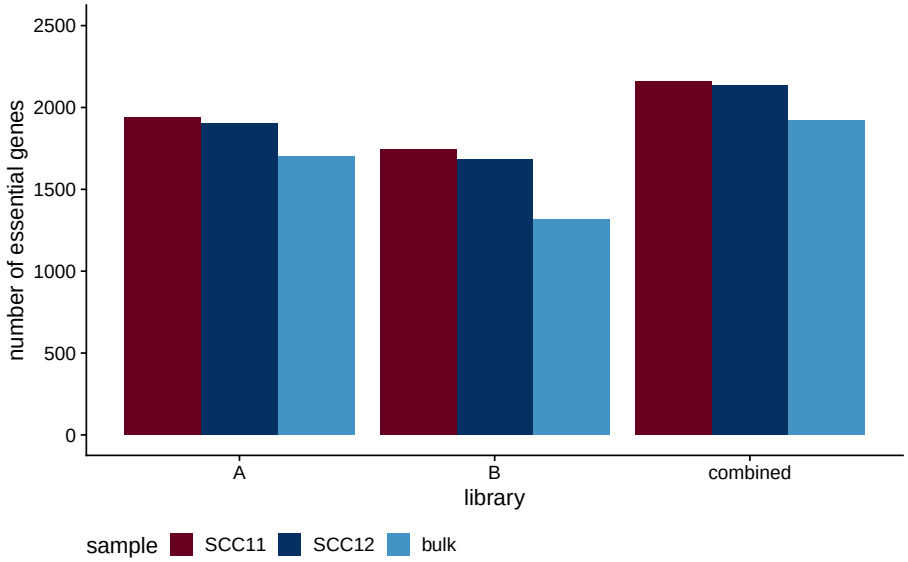
```
## add experiment information to counts df
counts_df_gsc <- norm_df %>% filter(!grepl('CONTROL', sgRNA)) %>%
  mutate(sample = gsub('Pool', 'bulk', sample)) %>%
  separate(sample, c('cellline', 'time', 'rep'),
           sep='_', fill = 'right', remove=F)

## add 'combined' library
counts_df_gsc <- counts_df_gsc %>%
```

```
  dplyr::select(sgRNA, Gene, sample, library, cellline, time, rep, count) %>%
  bind_rows(counts_df_gsc %>%
              mutate(library = 'combined',
                     norm_count = round(norm_count, digits = 0)) %>%
              dplyr::select(sgRNA, Gene, sample, library,
                            cellline, time, rep, count = norm_count))
## run gscreened on each library
gscreend_out <- counts_df_gsc %>% distinct(cellline, library) %>%
  filter(cellline != 'Plasmid') %>%
  mutate(gscreend = map2(cellline, library, ~{
    ## extract data needed for analysis
    hdcrispr_df <- counts_df_gsc %>%
      filter(cellline %in% c('Plasmid', .x), library == .y) %>%
      dplyr::select(-c(time, rep, cellline)) %>%
      spread(sample, count) %>%
      dplyr::select(sgRNA, Gene, library, Plasmid, everything())

    ## we first need a count matrix, three columns, plasmid and Tx
    counts_matrix <- as.matrix(cbind(hdcrispr_df[,4:6]))

    ## we then make an annotation object for rows
    rowData <- data.frame(sgRNA_id = hdcrispr_df$sgRNA,
                          gene = hdcrispr_df$Gene)

    ## same for the columns
    colData <- data.frame(samplename = c("library", "R1", "R2"),
                          # timepoint naming convention:
                          # T0 -> reference,
                          # T1 -> after proliferation
                          timepoint = c("T0", "T1", "T1"))

    ## use these to make a summarized experiment object
    se <- SummarizedExperiment(assays=list(counts=counts_matrix),
                               rowData=rowData, colData=colData)

    ## use this object to run gscreend
    pse <- createPoolScreenExp(se)
    pse_an <- RunGscreend(pse)

    ## return top table
    as_tibble(ResultsTable(pse_an)) %>%
      mutate(library = .y,
             sample = .x)
  }))
```

We visualize the results as a bar plot.

```
## visualize results as bar plot
gscreend_out %>% dplyr::select(-library) %>%
  unnest(gscreend) %>%
  mutate(essential = ifelse(fdr < 0.05 & lfc < 0, T, F)) %>%
```

```
  filter(essential) %>% dplyr::count(sample, library) %>%
  ggplot(aes(library, n, fill=sample)) +
  geom_bar(stat='identity', position = 'dodge') +
  scale_fill_manual(values = c('#67001F', '#053061', '#4393C3')) +
  ylim(c(0, 2500)) + ylab('number of essential genes') +
  theme(legend.position = 'bottom')
```



2.6.4 Comparison of hits determined by MAGeCK and BAGEL

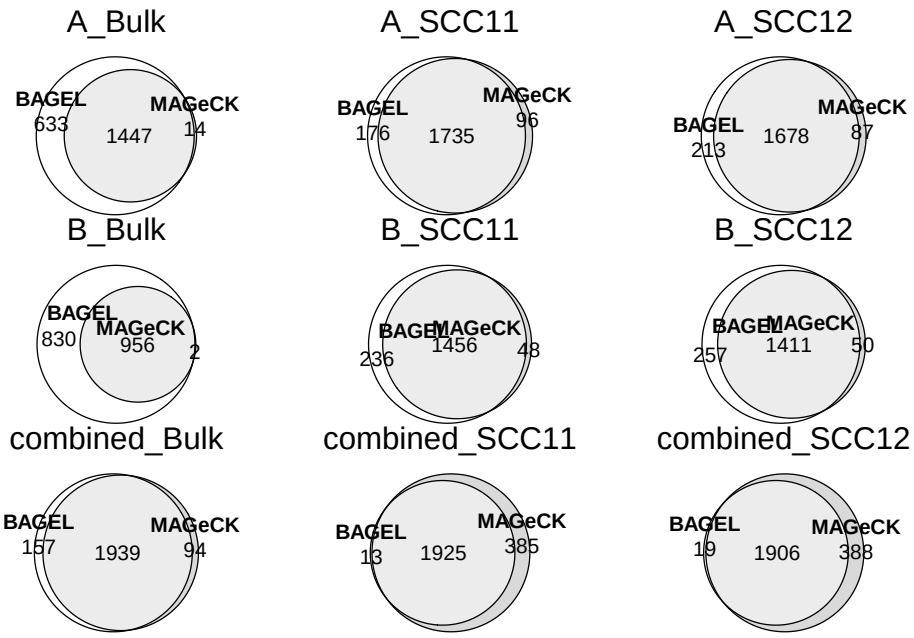
To see whether BAGEL and MAGeCK agree in terms of the essential genes that they determine, we generate Venn diagrams

```
## hits called by mageck
mageck_hits <- mageck_res %>%
  mutate(essential = ifelse(FDR < 0.05 & lfc < 0, T, F)) %>%
  filter(essential) %>%
  dplyr::select(library = lib, sample, symbol) %>%
  distinct()

## hits called by bagel
bagel_hits <- bfs %>%
  mutate(essential = ifelse(BF > 6, T, F),
         type = ifelse(type == 'Pool', 'Bulk', type)) %>%
  filter(essential) %>%
  dplyr::select(library, sample = type, symbol = GENE) %>%
  distinct()

## overlap between clones with mageck
mageck_venn <- mageck_hits %>% split(.$library) %>%
  map(~ .x %>% split(.$sample) %>%
        map(function(x) x$symbol))
```

```
walk(names(mageck_venn),
     ~ plot(euler(mageck_venn[[.x]]),
            quantities = T,
            main = paste('MAGeCK library', .x)))

## overlap between clones with bagel
bagel_venn <- bagel_hits %>% split(.$library) %>%
  map(~ .x %>% split(.$sample) %>%
        map(function(x) x$symbol))

walk(names(bagel_venn),
     ~ plot(euler(mageck_venn[[.x]]),
            quantities = T,
            main = paste('BAGEL library', .x)))
```

We further make pairwise comparisons between the tools for each library and cell type individually.

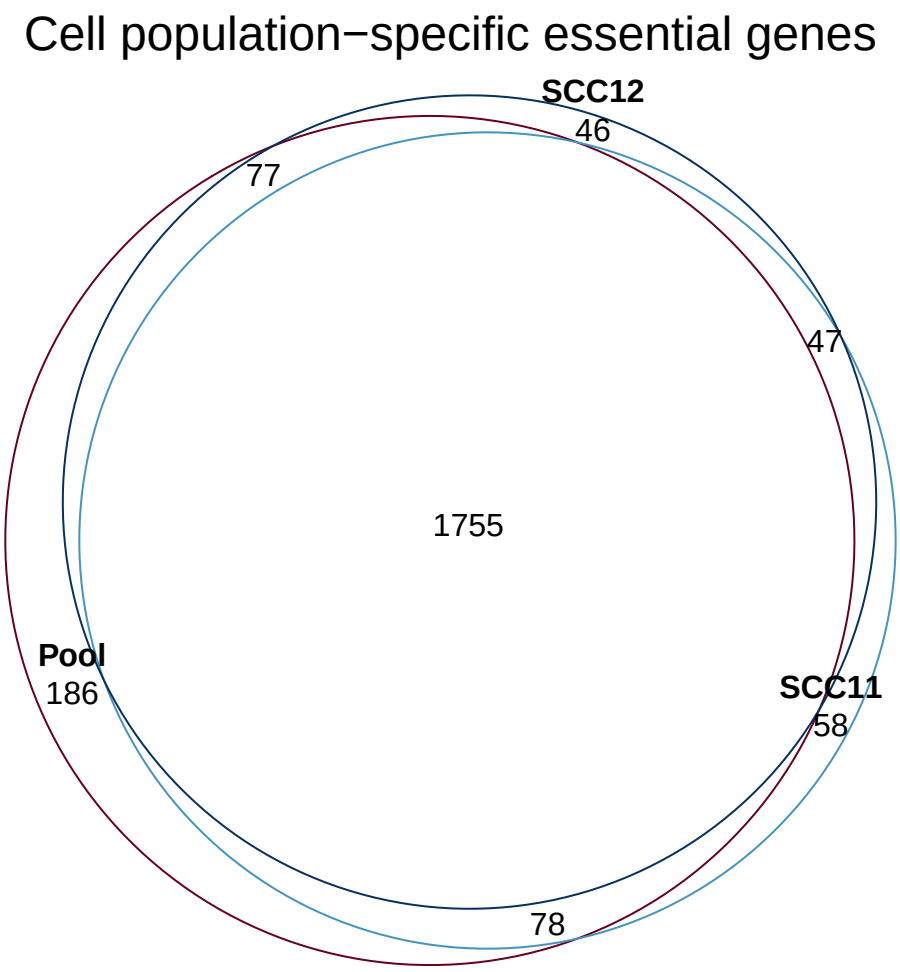
```
## compare n hits bagel vs mageck for different condition
venn_bagel_vs_mageck <- bagel_hits %>%
  mutate(tool = 'BAGEL') %>%
  bind_rows(mageck_hits %>% mutate(tool = 'MAGeCK')) %>%
  unite(screen, library, sample) %>%
  split(.$screen) %>%
  map(~ .x %>% split(.$tool) %>% map(function(x) x$symbol))

venn_diagrams <- map(names(venn_bagel_vs_mageck),
     ~ plot(euler(venn_bagel_vs_mageck[[.x]]),
            quantities = T,
            main = .x))

## print plots to canvase
n <- length(venn_diagrams)
nCol <- floor(sqrt(n))
do.call("grid.arrange", c(venn_diagrams, ncol=nCol))
```

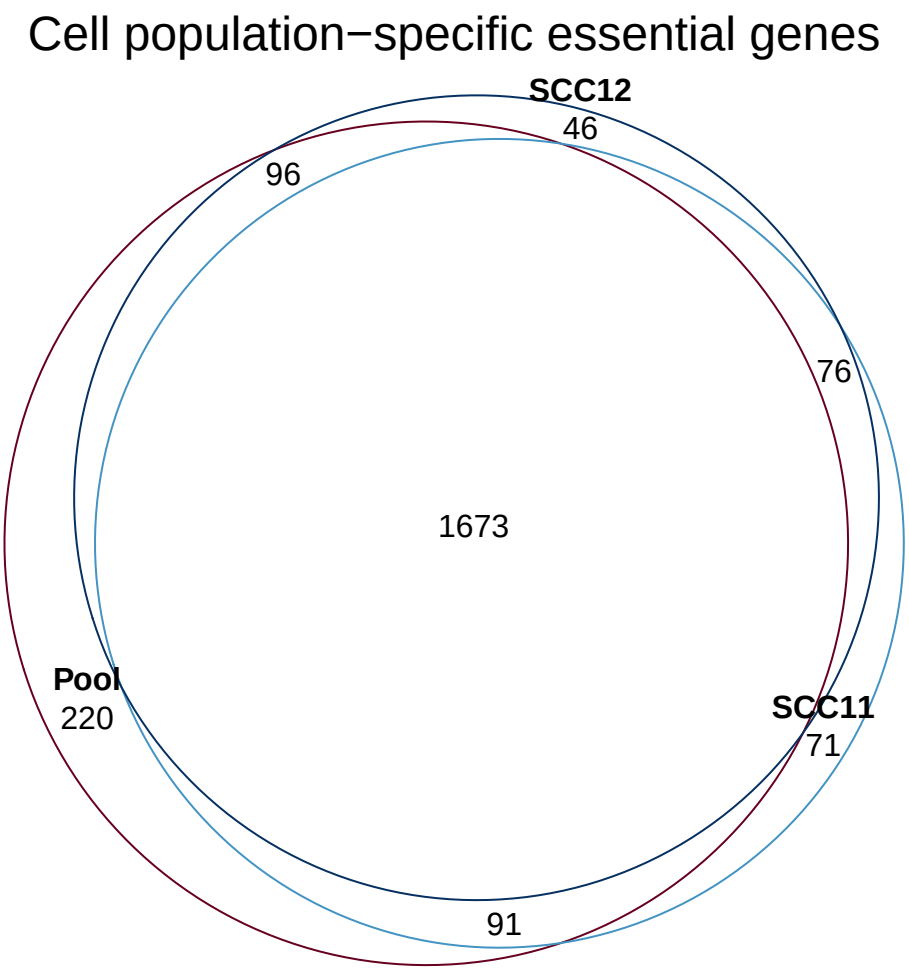


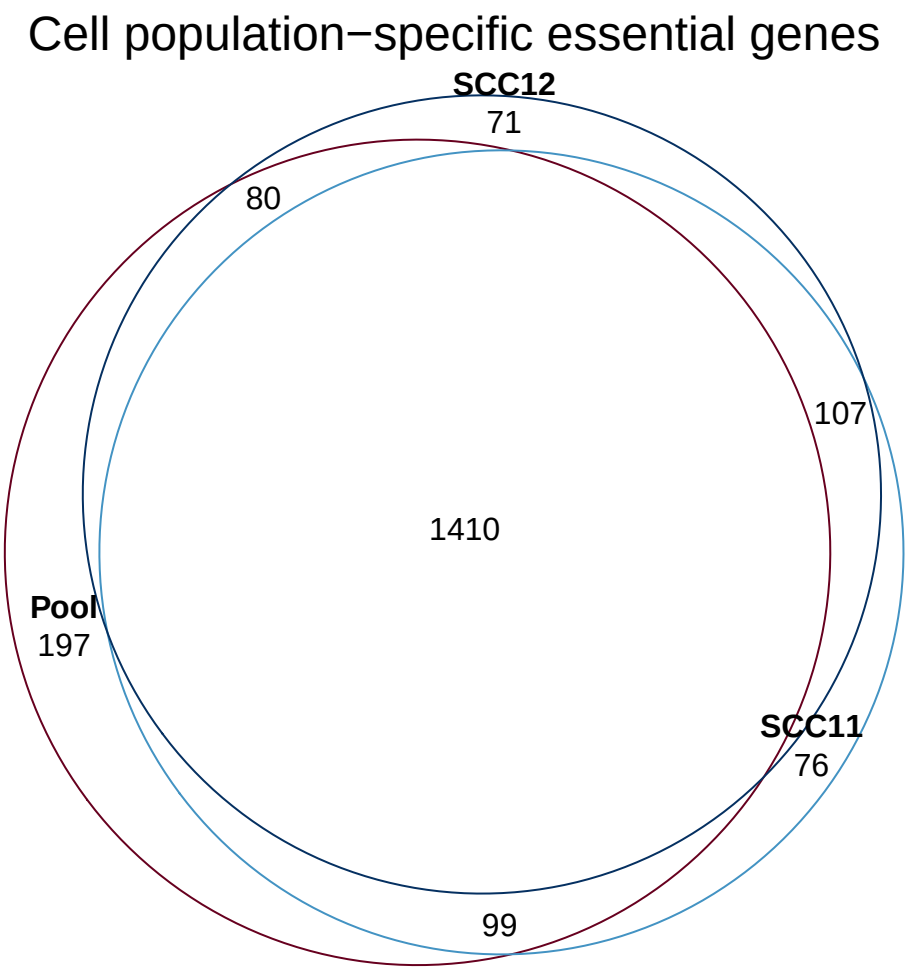
2.7 Differential essentiality between SCC and Bulk

Are there strong clonality effects where genes are more essential in one clone compared to other clones or the bulk population? We first make a Venn diagram of all essential genes detected in bulk and single cell clone populations.

```
map(c('combined', 'A', 'B'), function(lib_type){
  ## overlap between clones with bagel
  scc_hits <- bfs %>% filter(library == lib_type) %>%
    mutate(essential = ifelse(BF > 6, T, F)) %>%
    filter(essential) %>% split(.$type) %>%
    map(~ .x$GENE)

  plot(euler(scc_hits), quantities = T,
       main = 'Cell population-specific essential genes',
       col = c('#67001F', '#4393C3', '#053061'),
       fill = NA)
})
```

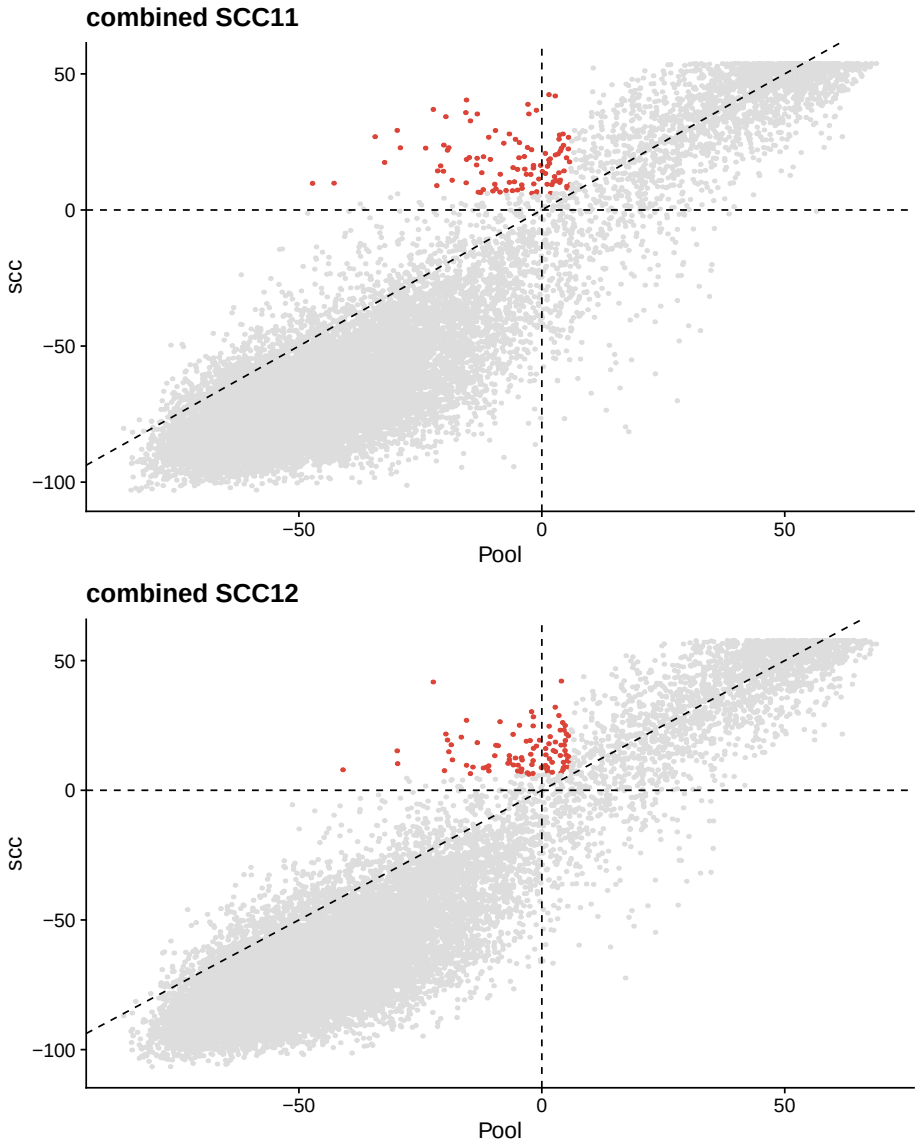

Cell population−specific essential genes
SCC12
46
77
47
1755
Pool
186
SCC11
58
78

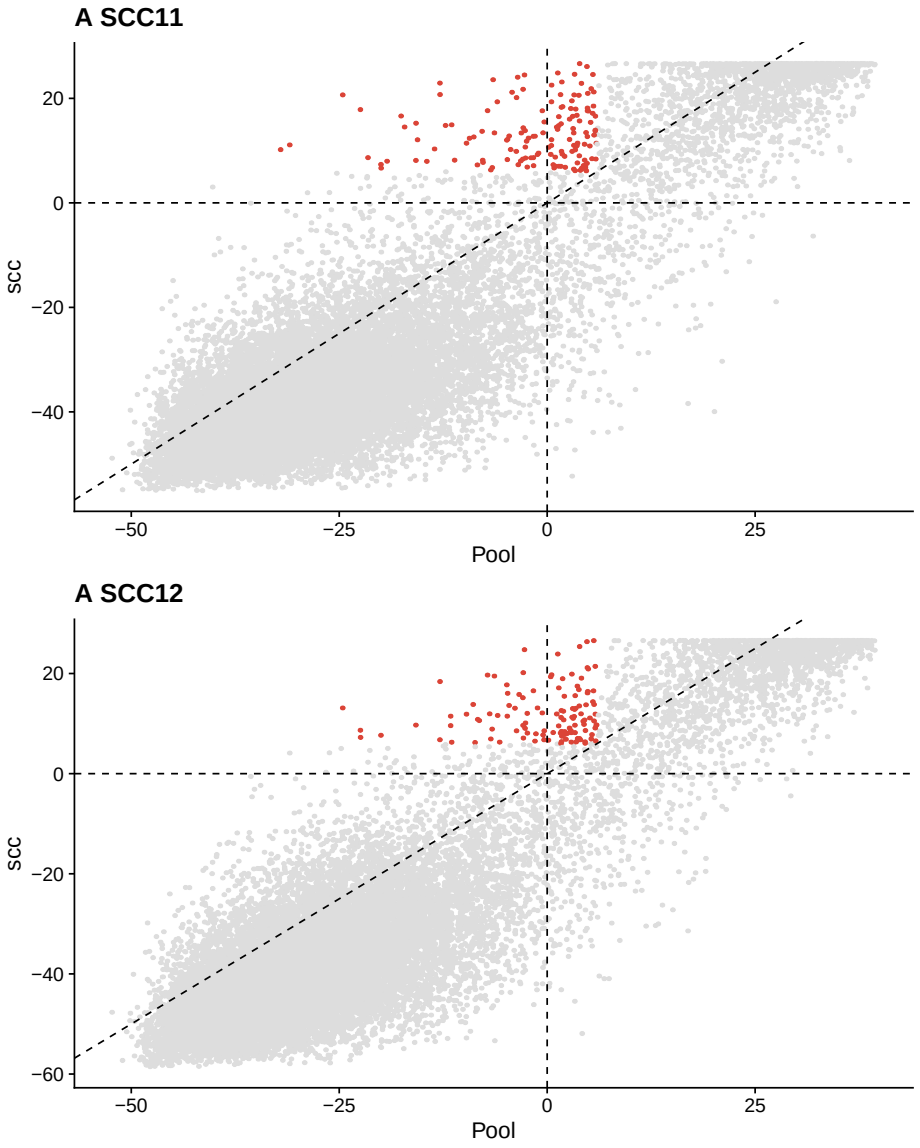

Cell population−specific essential genes
SCC12
46
96
76
1673
Pool
220
SCC11
71
91

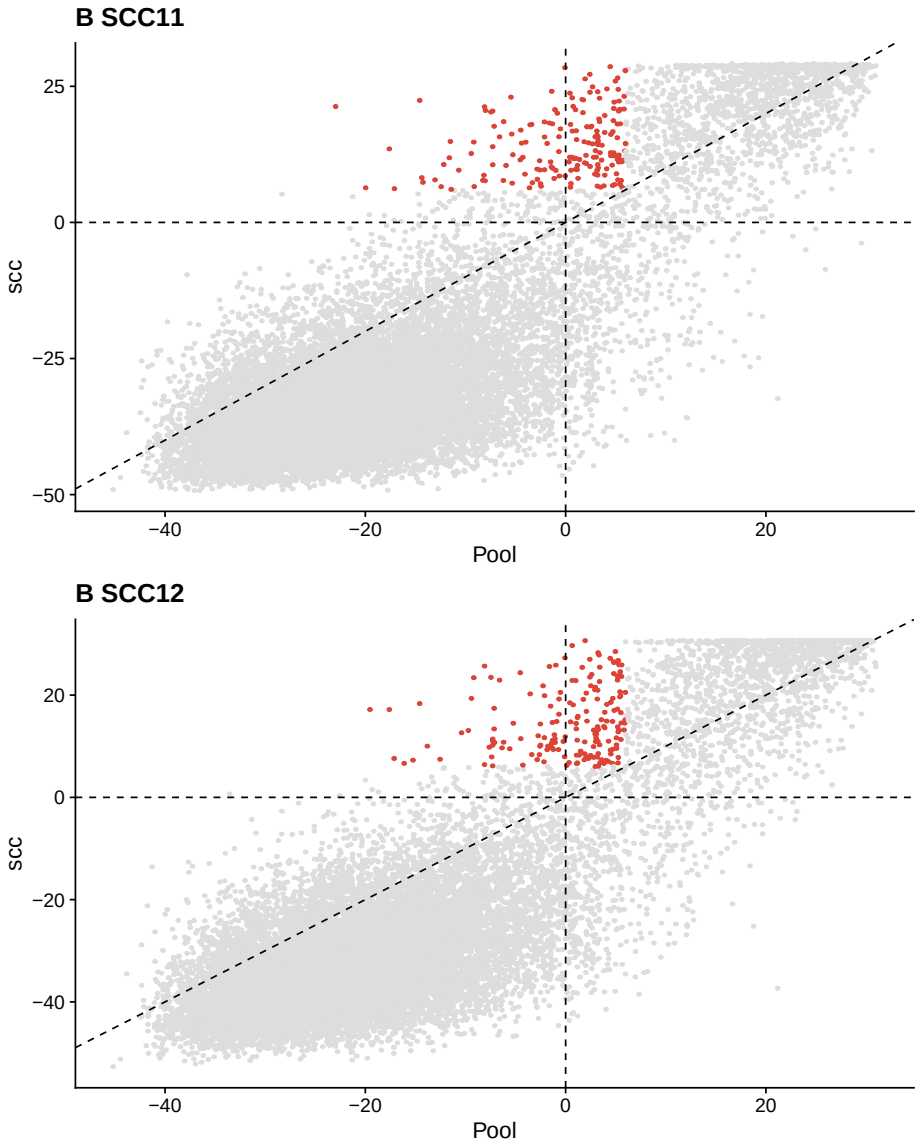


Overall, the single-cell clone-specific essential genes are fewer than expected. We generate scatter plots to compare SCC11 and SCC12 Bayes Factors to the bulk population for both sub-libarries and the combined library.

```
## scatter plots
map(c('combined', 'A', 'B'), function(lib_type){
  map(c('SCC11', 'SCC12'), function(clone){
    bfs %>% filter(library == lib_type, type %in% c('Pool', clone)) %>%
      dplyr::select(GENE, BF, type) %>%
      spread(type, BF) %>% dplyr::rename(scc = !!clone) %>%
      mutate(scc_spec = ifelse(Pool < 6 & scc > 6, T, F)) %>%
      ggplot(aes(Pool, scc, colour = scc_spec)) +
      ggrastr::geom_point_rast(size=1) +
      geom_abline(linetype='dashed') +
      geom_hline(yintercept = 0, linetype = 'dashed') +
      geom_vline(xintercept = 0, linetype = 'dashed') +
      scale_color_manual(values = c('#dddddd', '#db4437')) +
      theme(legend.position = 'none') +
      ggtitle(paste(lib_type, clone))
  })
})
```


combined SCC11
50
0
−50
−100
scc
−50
0
50
Pool
combined SCC12
50
0
−50
−100
scc
−50
0
50
Pool


A SCC11
20
0
−20
−40
scc
−50
−25
0
25
Pool
A SCC12
20
0
−20
−40
−60
scc
−50
−25
0
25
Pool



We perform a gene set enrichment analysis to determine whether there are specific pathways or biological processes that are essential only in bulk or single cell cloens.

```
## a list of all avana library genes
hdcrispr_genes <- unique(bfs$GENE)
## get entrez gene IDs
hdcrispr_genes <- bitr(hdcrispr_genes,
                       fromType = 'SYMBOL',
                       toType = 'ENTREZID',
                       OrgDb = org.Hs.eg.db)

## convert cluster genes to entrez id
go_results <- map(c('combined', 'A', 'B'), function(lib_type){
  df <- bfs %>% filter(library == lib_type) %>%
    dplyr::select(GENE, BF, type) %>%
```

```
    spread(type, BF)

  ## SCC11-specific genes
  genes_scc11 <- df %>% filter(SCC11 > 6, SCC12 < 6, Pool < 6) %>% pull(GENE)
  genes_scc11 <- hdcrispr_genes %>%
    filter(SYMBOL %in% genes_scc11) %>% pull(ENTREZID)
  ## SCC12-specific genes
  genes_scc12 <- df %>% filter(SCC11 < 6, SCC12 > 6, Pool < 6) %>% pull(GENE)
  genes_scc12 <- hdcrispr_genes %>%
    filter(SYMBOL %in% genes_scc12) %>% pull(ENTREZID)
  ## bulk-specific genes
  genes_bulk <- df %>% filter(SCC11 < 6, SCC12 < 6, Pool > 6) %>% pull(GENE)
  genes_bulk <- hdcrispr_genes %>%
    filter(SYMBOL %in% genes_bulk) %>% pull(ENTREZID)

  ## run go over-representation analysis
  go_enr <- map(list(genes_scc11, genes_scc12, genes_bulk), ~{
    goana(.x,
          universe = hdcrispr_genes$ENTREZID,
          FDR = 0.01, species = 'Hs') %>%
      as_tibble(rownames = 'go_id') %>% filter(Ont == 'BP') %>%
      arrange(P.DE) %>% mutate(FDR = p.adjust(P.DE, method = 'BH')) %>%
      filter(FDR < 0.05)
  })
})
```

3 Session info

```
sessionInfo()
#> R version 3.6.1 (2019-07-05)
#> Platform: x86_64-apple-darwin15.6.0 (64-bit)
#> Running under: macOS Mojave 10.14.6
#>
#> Matrix products: default
#> BLAS:   /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRblas.0.dylib
#> LAPACK: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRlapack.dylib
#>
#> locale:
#> [1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
#>
#> attached base packages:
#> [1] parallel  stats4    stats     graphics  grDevices utils     datasets
#> [8] methods   base
#>
#> other attached packages:
#>  [1] forcats_0.4.0               stringr_1.4.0
#>  [3] dplyr_0.8.3                 purrr_0.3.3
#>  [5] readr_1.3.1                 tidyr_1.0.0
```

```
#>  [7] tibble_2.1.3               tidyverse_1.2.1
#>  [9] ggrepel_0.8.1              ggplot2_3.2.1
#> [11] cowplot_1.0.0              ggrastr_0.1.7
#> [13] pheatmap_1.0.12            gridExtra_2.3
#> [15] UpSetR_1.4.0               eulerr_6.0.0
#> [17] reshape2_1.4.3             SummarizedExperiment_1.14.1
#> [19] DelayedArray_0.10.0        BiocParallel_1.18.1
#> [21] matrixStats_0.55.0         GenomicRanges_1.36.1
#> [23] GenomeInfoDb_1.20.0        gscreend_0.99.6
#> [25] preprocessCore_1.46.0      org.Hs.eg.db_3.8.2
#> [27] AnnotationDbi_1.46.1       IRanges_2.18.3
#> [29] S4Vectors_0.22.1           Biobase_2.44.0
#> [31] BiocGenerics_0.30.0        limma_3.40.6
#> [33] clusterProfiler_3.12.0     ROCR_1.0-7
#> [35] gplots_3.0.1.1             HDCRISPR2019_0.1.0
#> [37] BiocStyle_2.12.0
#>
#> loaded via a namespace (and not attached):
#>   [1] readxl_1.3.1           backports_1.1.5        fastmatch_1.1-0
#>   [4] plyr_1.8.4             igraph_1.2.4.1         lazyeval_0.2.2
#>   [7] polylabelr_0.1.0       splines_3.6.1          urltools_1.7.3
#>  [10] digest_0.6.22          htmltools_0.4.0        GOSemSim_2.10.0
#>  [13] viridis_0.5.1          GO.db_3.8.2            fansi_0.4.0
#>  [16] gdata_2.18.0           magrittr_1.5           memoise_1.1.0
#>  [19] graphlayouts_0.5.0     modelr_0.1.5           enrichplot_1.4.0
#>  [22] prettyunits_1.0.2      colorspace_1.4-1       rvest_0.3.4
#>  [25] blob_1.2.0             haven_2.1.1            xfun_0.10
#>  [28] hexbin_1.27.3          crayon_1.3.4           RCurl_1.95-4.12
#>  [31] jsonlite_1.6           zeallot_0.1.0          glue_1.3.1
#>  [34] polyclip_1.10-0        gtable_0.3.0           zlibbioc_1.30.0
#>  [37] XVector_0.24.0         fGarch_3042.83.1       scales_1.0.0
#>  [40] DOSE_3.10.2            DBI_1.0.0              Rcpp_1.0.2
#>  [43] viridisLite_0.3.0      progress_1.2.2         gridGraphics_0.4-1
#>  [46] bit_1.1-14             europepmc_0.3          timeSeries_3042.102
#>  [49] httr_1.4.1             fgsea_1.10.1           RColorBrewer_1.1-2
#>  [52] ellipsis_0.3.0         spatial_7.3-11         pkgconfig_2.0.3
#>  [55] farver_1.1.0           utf8_1.1.4             labeling_0.3
#>  [58] ggplotify_0.0.4        tidyselect_0.2.5       rlang_0.4.0
#>  [61] cellranger_1.1.0       munsell_0.5.0          tools_3.6.1
#>  [64] cli_1.1.0              generics_0.0.2         RSQLite_2.1.2
#>  [67] broom_0.5.2            ggridges_0.5.1         evaluate_0.14
#>  [70] yaml_2.2.0             knitr_1.25             bit64_0.9-7
#>  [73] tidygraph_1.1.2        caTools_1.17.1.2       ggraph_2.0.0
#>  [76] nlme_3.1-141           DO.db_2.9              xml2_1.2.2
#>  [79] rstudioapi_0.10        compiler_3.6.1         tweenr_1.0.1
#>  [82] stringi_1.4.3          lattice_0.20-38        fBasics_3042.89
#>  [85] Matrix_1.2-17          nloptr_1.2.1           vctrs_0.2.0
#>  [88] pillar_1.4.2           lifecycle_0.1.0        BiocManager_1.30.9
#>  [91] triebeard_0.3.0        data.table_1.12.6      bitops_1.0-6
#>  [94] qvalue_2.16.0          R6_2.4.0               bookdown_0.14
#>  [97] KernSmooth_2.23-16     MASS_7.3-51.4          gtools_3.8.1
```

```
#> [100] assertthat_0.2.1       withr_2.1.2           GenomeInfoDbData_1.2.1
#> [103] hms_0.5.1              grid_3.6.1            timeDate_3043.102
#> [106] rmarkdown_1.16         rvcheck_0.1.5         Cairo_1.5-10
#> [109] ggforce_0.3.1          lubridate_1.7.4
```