Towards a Map of Genetic Interactions in Cancer Cells

Benedikt Rauscher
German Cancer Research Center (DKFZ), Heidelberg, Germany
b.rauscher @ dkfz.de

2017-12-29

Abstract

Cancer genomes often harbor hundreds of molecular aberrations. Such genetic variants can be drivers or passengers of tumorigenesis and, as a side effect, create new vulnerabilities for potential therapeutic exploitation. To systematically identify genotype-dependent vulnerabilities and synthetic lethal interactions, forward genetic screens in different genetic backgrounds have been conducted. We devised MINGLE, a computational framework that integrates CRISPR/Cas9 screens originating from many different libraries and laboratories to build genetic interaction maps. It builds on analytical approaches that were established for genetic network discovery in model organisms. We applied this method to integrate and analyze data from 85 CRISPR/Cas9 screens in human cancer cell lines combining functional data with information on genetic variants to explore the relationships of more than 2.1 million gene-background relationships. In addition to known dependencies, our analysis identified new genotype-specific vulnerabilities of cancer cells. Experimental validation of predicted vulnerabilities associated with aberrant Wnt/β-catenin signaling identified GANAB and PRKCSH as new positive regulators of Wnt/β-catenin signaling. By clustering genes with similar genetic interaction profiles, we drew the largest genetic network in cancer cells to date. Our scalable approach highlights how diverse genetic screens can be integrated to systematically build informative maps of genetic interactions in cancer, which can grow dynamically as more data is included.

Contents

About

This document contains code and data that can reproduce figures 2-4 in the manuscript. Figure 5 can be reproduced using Cytoscape (Shannon et al. 2003) with the Cytoscape session file included in the supplemental materials of the paper.

1 Analysis of CRISPR-Cas9 screens

Before the normalization and analysis of CRISPR screens we load a number data. These include tissue annotations for the cancer cell lines, lists of core-essential and non-essential training genes (Hart et al. 2017) and the sgRNA-level fold changes for each screen. sgRNA-level fold changes are calculated from the raw sgRNA counts available from GenomeCRISPR (Rauscher et al. 2017) after normalization for sequencing depth by median-scaling. We finally load a list of annotated variants based on data from various sources (Forbes et al. 2017, Barretina et al. (2012), Klijn et al. (2014)) as described in the manuscript.

```
## load core essentials
data('ref_essential', package='CGIMhd17')
## load non-essentials
data('ref_nonessential', package='CGIMhd17')
## cell line annotation
data('cell_line_anno_v', package='CGIMhd17')
cell_line_anno <- cell_line_anno_v %>% dplyr::select(`CCLE name`, `Site Primary`) %>%
  separate(`CCLE name`, c('cellline', 'tissue'), extra='merge') %>% dplyr::select(-tissue)

## un-normalized sgRNA-level fold changes
data('fc', package='CGIMhd17')

## cancer variants
data('somatic_filtered', package='CGIMhd17')
## copy number gains
data('cna_gains', package='CGIMhd17')
```

1.1 Normalization and batch adjustment

We first aggregate technical replicates by averaging. We then calculate gene-level CRISPR scores by averaging sgRNA fold changes for each gene excluding genes that are targeted by only 2 or less reagents. We annotate the library used for each screen before performaning a gene level batch adjustment for artificial effects introduced by the library used.

```
## CRISPR scores - first average fold changes across technical replicates
cscores <- fc %>% filter(grepl('viability', screen)) %>%
  gather(rep, fc, -c(SEQID, GENE, screen)) %>% drop_na() %>%
  group_by(screen, GENE, SEQID) %>% summarise(fc=mean(fc)) %>% ungroup()

## only genes with at least 3 sgRNA measurements
```

```
counts <- cscores %>% count(screen, GENE) %>% ungroup()
# CRISPR scores are the average fold change of sgRNAs for a gene
cscores_filtered <- cscores %>% inner_join(counts) %>%
  filter(n>2) %>% dplyr::select(-n) %>%
  group_by(screen, GENE) %>% summarise(cs=mean(fc)) %>% ungroup()

## quantile normalization
norm_mat <- cscores_filtered %>% spread(screen, cs) %>% data.frame() %>%
  `rownames<-`(.$GENE) %>% dplyr::select(-GENE) %>% as.matrix()
cscores_filtered <- normalize.quantiles(as.matrix(norm_mat)) %>%
  `rownames<-`(rownames(norm_mat)) %>% `colnames<-`(colnames(norm_mat)) %>%
  data.frame() %>% mutate(GENE=rownames(.)) %>% gather(screen, cs, -GENE) %>%
  tbl_df %>% drop_na()

## annotate library
cscores_filtered <- cscores_filtered %>%
  separate(screen, c('fc', 'pubmed', 'cellline', 'genotype', 'condition'),
           sep='_', fill = 'right') %>%
  mutate(condition=ifelse(is.na(condition), genotype, condition)) %>%
  mutate(cellline = ifelse(condition == genotype, cellline, paste(cellline, genotype, sep='_'))) %>%
  dplyr::select(-c(genotype, fc)) %>%
  mutate(library=ifelse(pubmed %in% c('26627737', '27869803'), 'TKOv1',
                 ifelse(pubmed %in% c('27760321'), 'Yusa',
                 ifelse(pubmed %in% c('26472758', '28700943', '24336569', '28162770'), 'Sabatini',
                 ifelse(pubmed %in% c('27260156'), 'GeCKOv2',
                 ifelse(pubmed %in% ('27260157'), 'Novartis', NA))))))

gene_norm_factors <- function(df){
  libs <- levels(factor(df$library))
  if(length(libs) > 1){
    # lmsum <- lm(cs ~ 0 + library, data=df, offset=rep(median(df$cs), nrow(df))) %>% summary()
    med <- median(df$cs)
    lmsum <- rlm(I(cs-med) ~ 0 + library, data=df)
    pvals <- sapply(paste0('library', libs), function(x) f.robftest(lmsum, var=x) %>% .$p.value)
    res <- tibble(GENE=df$GENE[1],
                  lib=libs,
                  med=median(df$cs),
                  # p_batch=lmsum$coef[,4])
                  p_batch=pvals)
  } else{
    res <- tibble()
  }
  return(res)
}

gl_norm_facs <- cscores_filtered %>%
  group_by(cellline, GENE, library) %>% summarise(cs=mean(cs)) %>% ungroup() %>%
  group_by(GENE) %>%
  do(norm_facs=gene_norm_factors(.)) %>% ungroup() %>% dplyr::select(-GENE) %>%
  .$norm_facs %>% bind_rows() %>%
   mutate(padj_batch = p.adjust(p_batch, method='BH'))
## do not correct AML genes (that agree between yusa and sabatini)
aml <- gl_norm_facs %>% filter(padj_batch < 0.01) %>%
```

```
  group_by(GENE) %>% arrange(lib) %>% filter(n() == 2) %>%
  filter(lib[1] == 'Sabatini2', lib[2] == 'Yusa') %>% ungroup()
gl_norm_facs <- gl_norm_facs %>%
  mutate(padj_batch=ifelse(GENE %in% aml$GENE & lib %in% aml$lib, 1, padj_batch))

## apply batch correction according to the normalization factors
move_by <- cscores_filtered %>% left_join(gl_norm_facs %>% dplyr::rename(library=lib)) %>%
  group_by(GENE, library, cellline, padj_batch, med) %>% summarise(cs = mean(cs, na.rm=T)) %>%
  ungroup() %>% group_by(library, GENE) %>%
  mutate(move_by=ifelse(padj_batch[1]<0.05, sqrt((median(cs) - med[1])^2), 0),
         move_by=ifelse(median(cs) > med[1], move_by, (-1)*move_by)) %>%
  ungroup() %>% dplyr::select(GENE, library, move_by, med) %>% distinct()
bfs_norm <- cscores_filtered %>% left_join(move_by) %>%
  group_by(library, GENE) %>%
  mutate(BF= cs-move_by) %>%
  ungroup() %>%
  dplyr::rename(symbol=GENE) %>%
  dplyr::select(-c(cs, med)) %>%
  drop_na()

## remove some samples that were sequenced early
## where we cannot be sure that phenotypes
## are fully established
bfs_norm <- bfs_norm %>% filter(!grepl('after([1-9]|10)days', condition))

## add cell line tissue
bfs_norm <- bfs_norm %>% left_join(cell_line_anno) %>%
  mutate(`Site Primary` = ifelse(cellline == 'KBM7', 'haematopoietic_and_lymphoid_tissue',
                          ifelse(cellline == 'SF268', 'central_nervous_system', `Site Primary`))) %>%
  dplyr::rename(tissue=`Site Primary`)
```

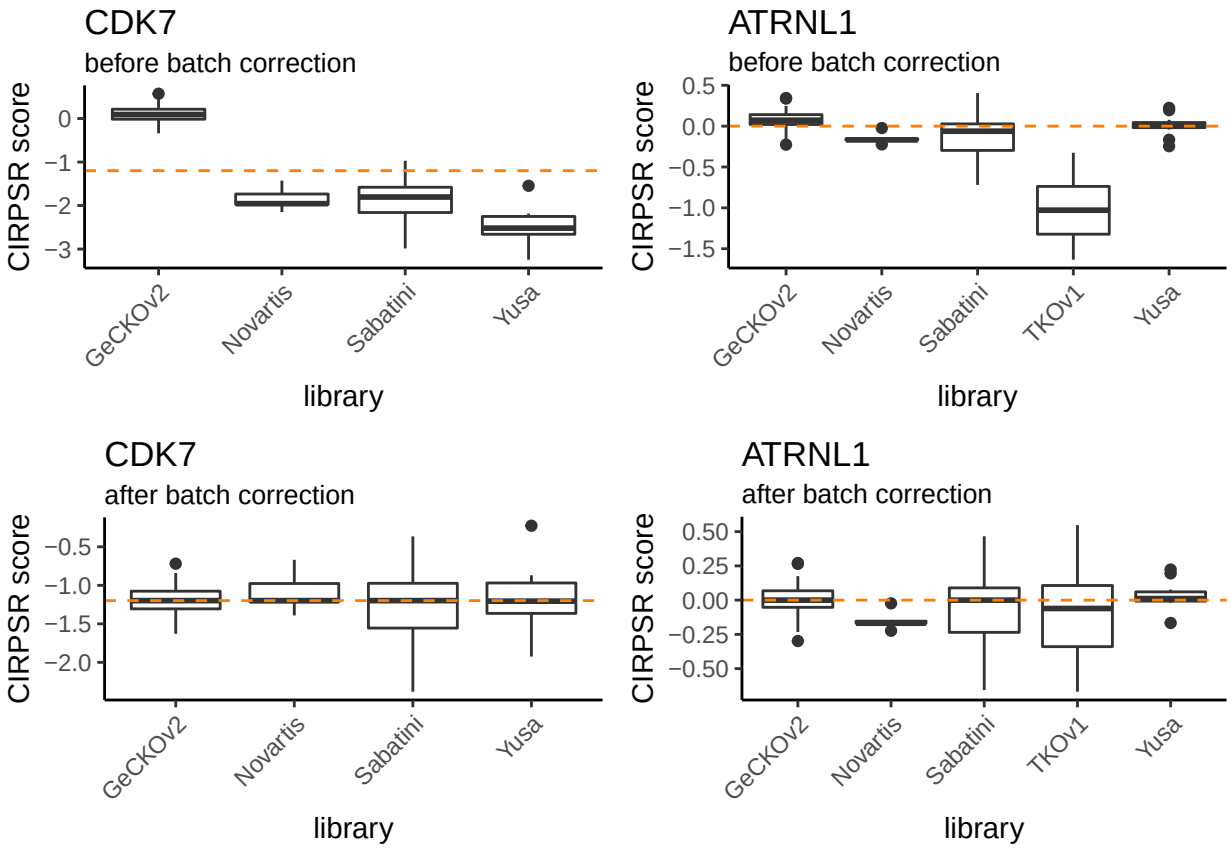
1.1.1 Normalization examples

```
multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL) {
  library(grid)

  # Make a list from the ... arguments and plotlist
  plots <- c(list(...), plotlist)

  numPlots = length(plots)

  # If layout is NULL, then use 'cols' to determine layout
  if (is.null(layout)) {
    # Make the panel
    # ncol: Number of columns of plots
    # nrow: Number of rows needed, calculated from # of cols
    layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
                     ncol = cols, nrow = ceiling(numPlots/cols))
  }

 if (numPlots==1) {
    print(plots[[1]])
```

```
  } else {
    # Set up the page
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout))))

    # Make each plot, in the correct location
    for (i in 1:numPlots) {
      # Get the i,j matrix positions of the regions that contain this subplot
      matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

      print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,
                                      layout.pos.col = matchidx$col))
    }
  }
}
```

We plot two examples of genes where normalization was applied for demonstration.

```
g <- c('CDK7', 'ATRNL1')

norm_plots <- lapply(g, function(gcurrent){
  med <- cscores_filtered %>% filter(GENE==gcurrent) %>%
    group_by(pubmed, cellline, library) %>% summarise(cs=mean(cs)) %>% ungroup() %>%
    .$cs %>% median(na.rm=T)
  p1 <- cscores_filtered %>% filter(GENE %in% gcurrent) %>%
    ggplot(aes(library, cs)) + geom_boxplot() + theme_classic() + ylab('CIRPSR score') +
    theme(axis.text.x=element_text(angle=45, hjust=1)) +
    geom_hline(yintercept=med, colour='#ff7f00', linetype='dashed') +
    ggtitle(gcurrent, subtitle='before batch correction')
  p2 <- bfs_norm %>% filter(symbol %in% gcurrent) %>%
    ggplot(aes(library, BF)) + geom_boxplot() + theme_classic() + ylab('CIRPSR score') +
    theme(axis.text.x=element_text(angle=45, hjust=1)) +
    geom_hline(yintercept=med, colour='#ff7f00', linetype='dashed') +
    ggtitle(gcurrent, subtitle='after batch correction')
  return(list(p1, p2))
})
multiplot(plotlist=unlist(norm_plots, recursive=F), cols=2)
```



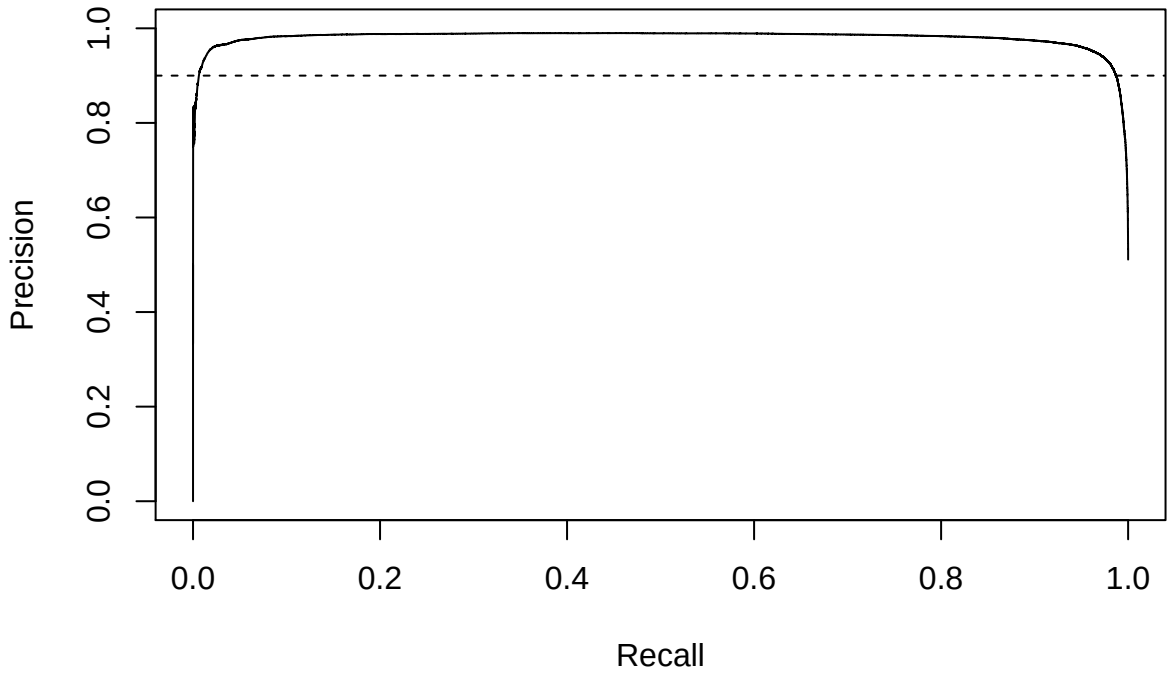
1.2 Quality Control

We perform an array of quality control analyses to show that the adjustment appropriately controls for batch effects and that true biological signal is preserved.

1.2.1 Precision-recall-characteristics

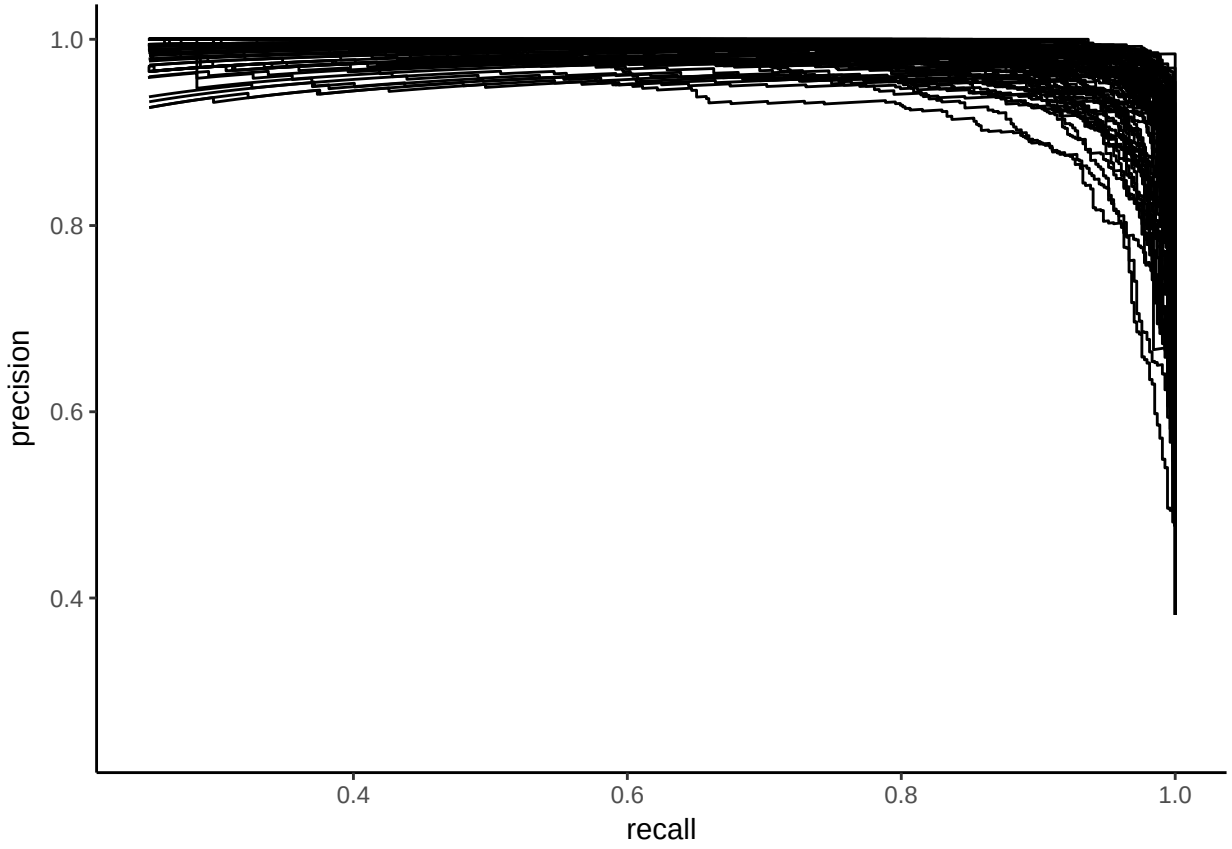
A precision recall curve can show if core-essential and non-essential reference genes can be separated based on the normalized data.

```
test_roc <- bfs_norm %>% mutate(essential=ifelse(symbol %in% ref_essential$symbol, 0,
                                        ifelse(symbol %in% ref_nonessential$symbol, 1, -1))) %>%
  filter(essential != -1)

## globally for all screens
perf <- performance(prediction(test_roc$BF, test_roc$essential), measure='prec', x.measure='rec')
plot(perf, ylim=c(0,1))
## 5% fdr where fdr = 1-precision
abline(0.9, 0, lty=2)
```




```
ind <- max(which(round(perf@y.values[[1]], digits=2)-0.9>=0))
cutoff <- perf@alpha.values[[1]][ind]

perc_recovered_all <- test_roc %>% mutate(rec=ifelse(BF < cutoff, 0, 1)) %>%
  filter(essential == 0) %>% summarise(percrec=round(length(which(essential == rec))/n(), 2))

## for each screen individually
percrec <- function(df){
  perf <- performance(prediction(df$BF, df$essential), measure='prec', x.measure='rec')
  ind <- max(which(round(perf@y.values[[1]], digits=2)-0.9>=0))
  cutoff <- perf@alpha.values[[1]][ind]
  perc <- df %>% mutate(cutoff=cutoff, rec=ifelse(BF < cutoff, 0, 1)) %>%
    filter(essential == 0) %>%
    summarise(percrec=round(length(which(essential == rec))/n(), 2)) %>%
    .$percrec
  return(list(perf=perf, percrec=perc))
}
perc_recovered_individual <- test_roc %>% group_by(pubmed, cellline, condition) %>%
  do(results=percrec(.)) %>% ungroup()

## generate data for roc curves
roc_data <- perc_recovered_individual %>% unite(screen, pubmed, cellline, condition, sep='_') %>%
  split(.$screen) %>%
  lapply(function(sc) return(tibble(precision=sc$results[1][[1]]$perf@y.values[[1]],
                                    recall=sc$results[1][[1]]$perf@x.values[[1]]) %>%
                               mutate(screen=sc$screen[1],
```

```
                                    percrec=sc$results[1][[1]]$percrec[1])) %>%
  bind_rows()

## plot curves
roc_data %>% ggplot(aes(recall, precision, group=screen)) + geom_line() +
  theme_classic() + xlim(c(0.25, 1)) + ylim(c(0.25, 1)) +
  theme(legend.position='none')
```

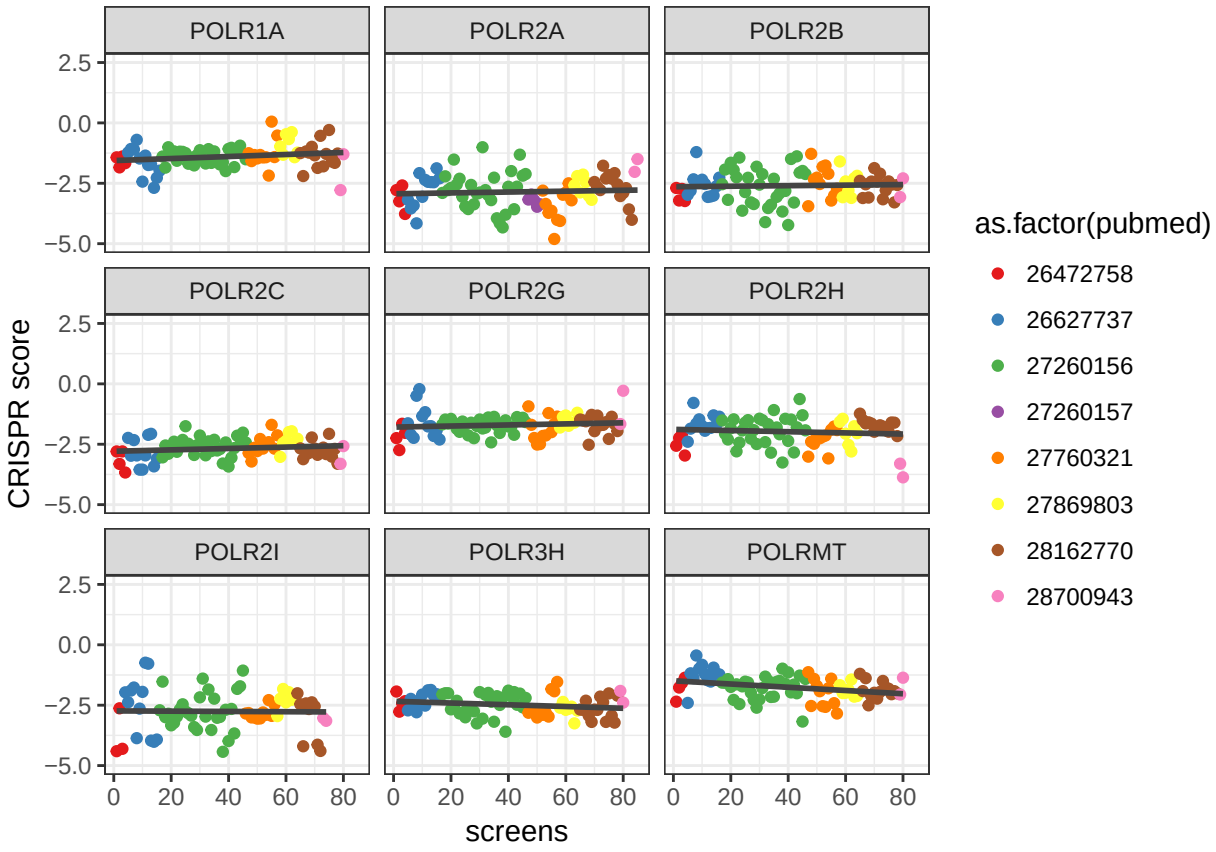


1.2.2 Batch-effect control: core-essential genes and chromosome Y genes

To show that phenotypes are comparable across experiments, we show data for various core-essential polymerases.

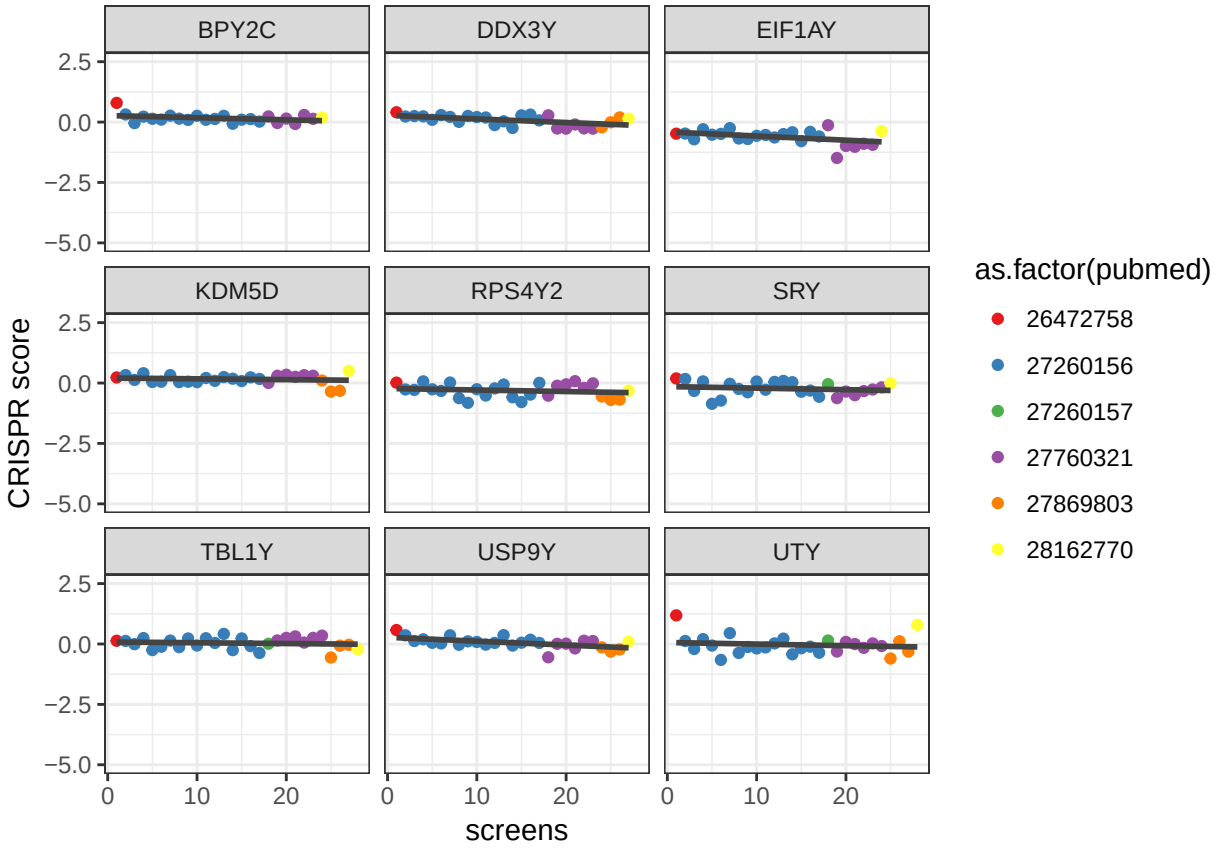
```
## plot batch effects plot
check_batch <- function(data, genes=NULL){
  set.seed(120)
  if(is.null(genes)){
    genes <- data %>% filter(symbol %in% ref_essential$symbol) %>% count(symbol) %>%
      arrange(desc(n)) %>% filter(n>70) %>% filter(grepl('^POLR', symbol)) %>% .$symbol %>% sample(9)
  }
  data %>% filter(symbol %in% genes) %>%
    group_by(symbol) %>% arrange(pubmed, cellline, condition) %>% mutate(n=1:n()) %>% ungroup() %>%
    ggplot(aes(x=n, y=BF)) + geom_point(aes(colour=as.factor(pubmed))) +
    geom_smooth(method='lm', se=F, colour='#444444') + ylab('CRISPR score') + xlab('screens') +
    theme_bw() + facet_wrap(~symbol) + ylim(c(-5, 2.5)) +
    scale_colour_brewer(palette='Set1')
```

```
}

check_batch(bfs_norm)
```



Looking at Y-chromosome gnees in female cell lines can mimick non-targeting controls to show that no artificial phenotypes get introduced by the approach.

```
## female cell lines
female <- cell_line_anno_v %>%
  separate(`CCLE name`, c('cellline', 'tissue'), sep='_', extra='merge') %>%
  filter(cellline %in% toupper(bfs_norm$cellline)) %>% filter(Gender == 'F') %>% .$cellline
## y chromsome genes
y_genes <- c('BPY2C', 'DDX3Y', 'EIF1AY', 'KDM5D',
             'RPS4Y2', 'SRY', 'TBL1Y', 'USP9Y', 'UTY')

## check batch effects
check_batch(bfs_norm %>% filter(cellline %in% female),
            genes=bfs_norm %>% filter(symbol %in% y_genes) %>% count(symbol) %>%
              arrange(desc(n)) %>% .$symbol %>% head(9))
```

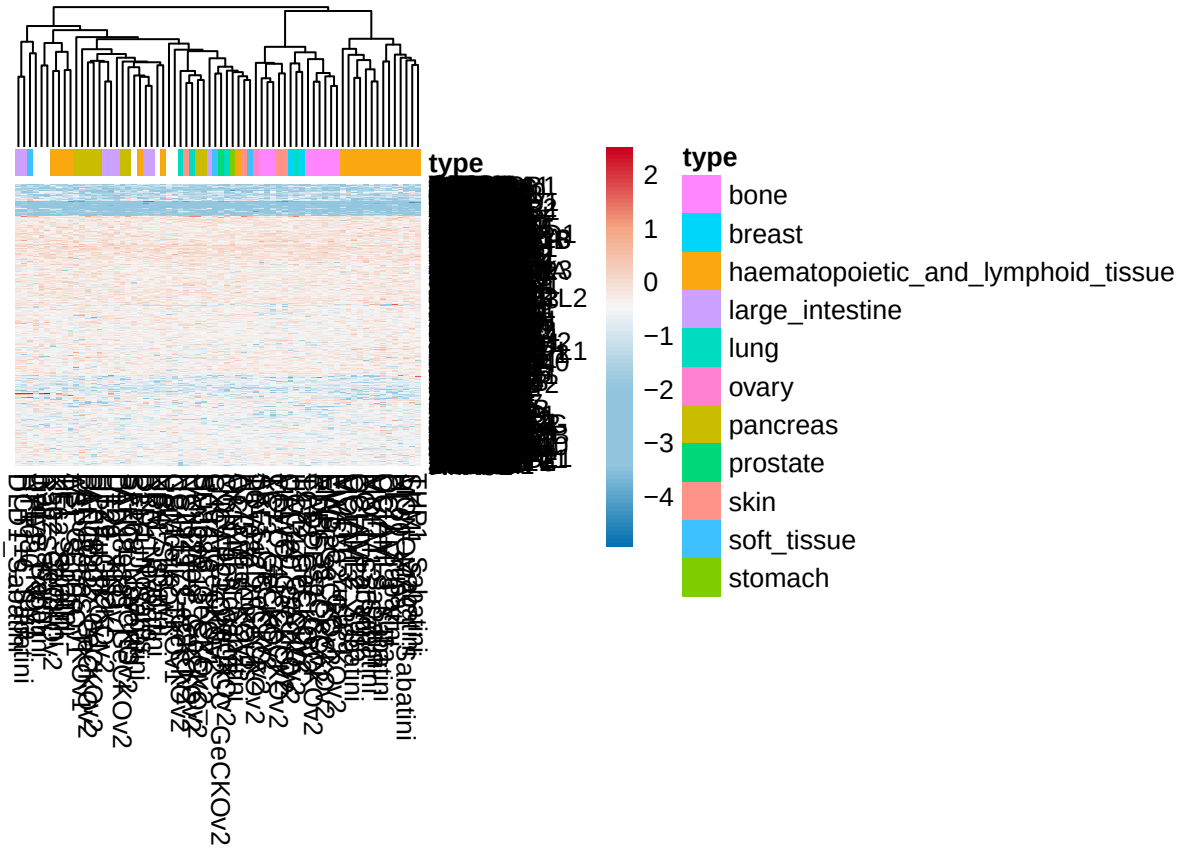


1.2.3 Heatmap of screens

A heatmap of the data can visualize relationships between the experiments and clustering can indicate if the data is dominated by batch effects or if biological signal is the main cause for variability.

```
## assemble heat map data
hm_data <- bfs_norm %>% group_by(cellline, library, pubmed, symbol) %>%
  summarise(BF=mean(BF)) %>% ungroup() %>%
  unite(screen, cellline, library) %>%
  dplyr::select(screen, symbol, BF) %>%
  filter(! symbol %in% aml$GENE) %>%
  spread(screen, BF) %>% data.frame() %>% `rownames<-`(.$symbol) %>%
  dplyr::select(-symbol) %>% drop_na()

## annotation of cancer type for heat map
cancer_type_anno <- cell_line_anno_v %>%
  separate(`CCLE name`, c('cellline', 'tissue'), sep='_', extra='merge') %>%
  dplyr::rename(type=`Hist Subtype1`) %>%
  mutate(type=ifelse(type == 'NS', `Site Primary`, `Site Primary`)) %>%
  group_by(cellline) %>% summarise(type=paste(type, collapse=',')) %>% ungroup()
cancer_type_anno <- tibble(screen=colnames(hm_data)) %>%
  separate(screen, c('cellline', 'rest'), extra='merge', remove=F) %>% dplyr::select(-rest) %>%
  left_join(cancer_type_anno) %>% data.frame() %>%
  `rownames<-`(.$screen) %>% dplyr::select(-c(screen, cellline))

##plot heat map
```

```
pheatmap(hm_data, treeheight_row=0,
         color=colorRampPalette(rev(c('#ca0020','#f4a582','#f7f7f7',
                                      '#92c5de', '#92c5de','#0571b0')))(100),
         clustering_method = 'ward.D2', annotation_col=cancer_type_anno)
```

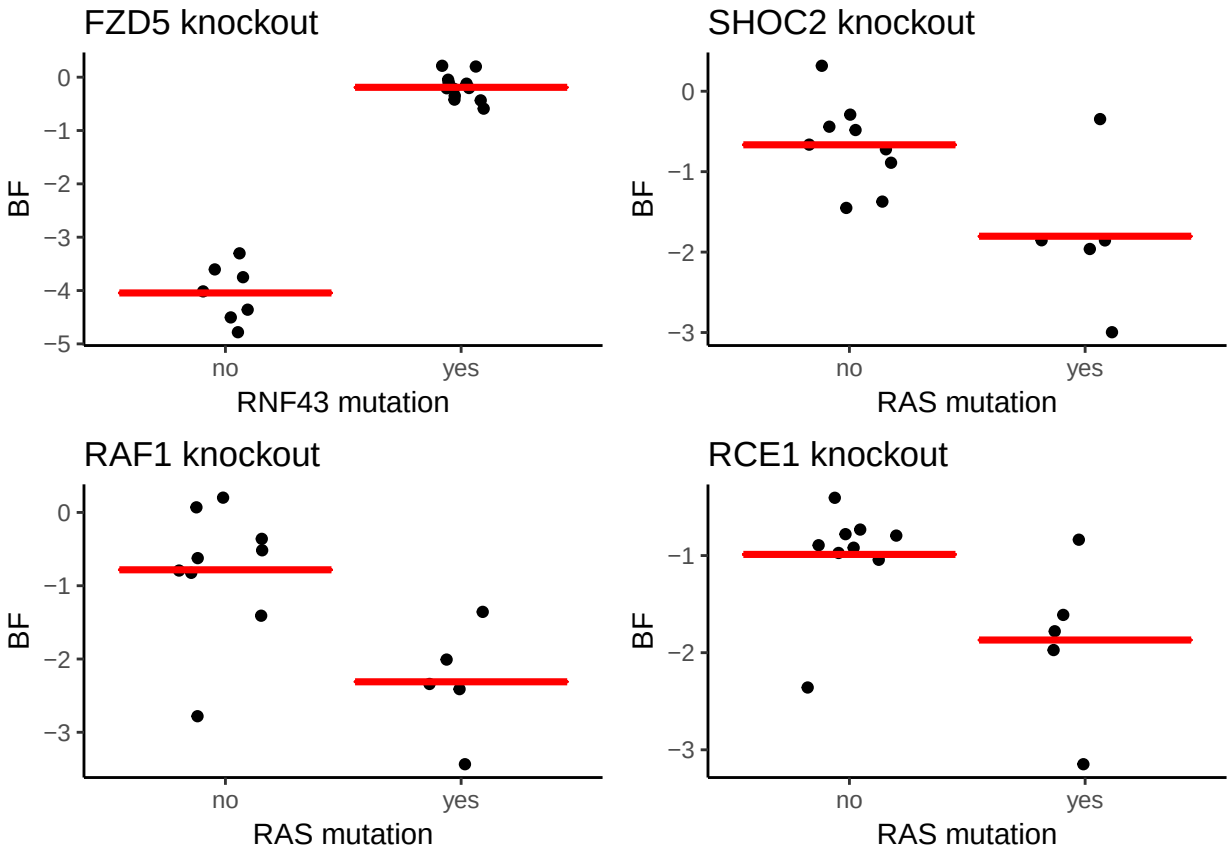


1.2.4 Published results do not get lost

Our data set contains data from published experiments. Several biological findings have been reported based on these experiments (Steinhart et al. 2017, T. Wang et al. (2017)). We next show that these findings can still be recovered based on the normalized Bayes Factors.

```
## FZD5 and RNF43
fzd <- bfs_norm %>% filter(pubmed %in% c('27869803', '26627737')) %>%
  filter(symbol == "FZD5") %>%
  mutate(rnf=ifelse(pubmed=='26627737', 'yes', 'no')) %>%
  ggplot(aes(rnf, BF)) + geom_jitter(width=0.1) + theme_classic() +
  stat_summary(fun.y='mean', fun.ymin='mean', fun.ymax='mean', geom='crossbar', colour='red') +
  xlab('RNF43 mutation') + ggtitle('FZD5 knockout')

## Wang17 Oncogenic ras
## first annotate ras mutation status
ras_cl <- c('SKM1', 'OCIAML3', 'P31FUJ', 'PL21', 'NB4')
wang17 <- bfs_norm %>% filter(pubmed == '28162770') %>%
  mutate(ras_mut = ifelse(cellline %in% ras_cl, 'yes', 'no'))
```

```
## RAF1 - RAS
ras1 <- wang17 %>% filter(symbol == 'RAF1') %>%
  ggplot(aes(ras_mut, BF)) + geom_jitter(width=0.2) + theme_classic() +
  stat_summary(fun.y='mean', fun.ymin='mean', fun.ymax='mean', geom='crossbar', colour='red') +
  xlab('RAS mutation') + ggtitle('RAF1 knockout')

## SHOC2 - RAS
ras2 <- wang17 %>% filter(symbol == 'SHOC2') %>%
  ggplot(aes(ras_mut, BF)) + geom_jitter(width=0.2) + theme_classic() +
  stat_summary(fun.y='mean', fun.ymin='mean', fun.ymax='mean', geom='crossbar', colour='red') +
  xlab('RAS mutation') + ggtitle('SHOC2 knockout')

## RCE1 - RAS
ras3 <- wang17 %>% filter(symbol == 'RCE1') %>%
  ggplot(aes(ras_mut, BF)) + geom_jitter(width=0.2) + theme_classic() +
  stat_summary(fun.y='mean', fun.ymin='mean', fun.ymax='mean', geom='crossbar', colour='red') +
  xlab('RAS mutation') + ggtitle('RCE1 knockout')

## ICMT - RAS
ras4 <- wang17 %>% filter(symbol == 'ICMT') %>%
  ggplot(aes(ras_mut, BF)) + geom_jitter(width=0.2) + theme_classic() +
  stat_summary(fun.y='mean', fun.ymin='mean', fun.ymax='mean', geom='crossbar', colour='red') +
  xlab('RAS mutation') + ggtitle('ICMT knockout')

## plot four plots for supplementary figure
multiplot(fzd, ras1, ras2, ras3, cols=2)
```



2 Combinatorial tests for synthetic genetic interactions

In the nexts steps, we test if there are differences in the response to a perturbation under the condition that perturbed cell lines have certain genetic variants. In the previous steps we normalized the CRISPR data and removed batch effects. Hence we can now select all genes from the genetic profile that are altered in at least three different cell lines screened in the CRISPR screens as query screens. We find all combinations between genes perturbed in CRISPR screens (target genes) and these query genes and generate data frames containing viability phenotypes for each combination with an additional column that annotates the mutation status corresponding to a knockout phenotype. These data are used subsequently for statistical modelling of genetic interactions for which we use a multilevel model. As the modelling takes too long to be performed in an R session we split the data into chunks and process them on a compute cluster. We load the results back into R for downstream analyses.

2.1 Remove high copy number targets which lead to false phenotypes

Previous results have shown that targeting highly amplified loci can introduce false positive phenotypes in CRISPR screen (Munoz et al. 2016, Aguirre et al. (2016)). This effect is also visible in our data and hence we exclude these perturbations.

```
## copy number amplifications are addded as annotated in previously loaded file
bfs_norm <- bfs_norm %>% filter(cellline %in% toupper(somatic_filtered$cellline)) %>%
  left_join(cna_gains %>% distinct(symbol, cellline, type)) %>%
  filter(is.na(type), cellline != 'TF1') %>% dplyr::select(-type)
```

2.2 Create combinations for testing

Next we can generate the data for testing and write it as chunks to the hard disk.

```
## make all possible target/query combinations
combis_batch <- expand.grid(bfs_norm %>% .$symbol %>% unique,
                            unique(somatic_filtered$symbol)) %>% tbl_df %>%
  dplyr::rename(target=Var1, query=Var2) %>%
  mutate(target=as.character(target), query=as.character(query))

## loop query by query to generate test file
combis <- combis_batch %>% split(.$query) %>% lapply(function(x){
  print(paste('Processing', as.character(x$query[1]), '...'))

  ## merge all data into big data frame for testing.
  combis_bf_batch <- x %>%
    inner_join(bfs_norm %>% dplyr::rename(target=symbol)) %>%
    left_join(somatic_filtered %>% mutate(mutated=1) %>%
                dplyr::select(symbol, cellline, mutated) %>% dplyr::rename(query=symbol)) %>%
    mutate(type=ifelse(is.na(mutated), 'target', 'combi')) %>%
    dplyr::select(-c(mutated, condition)) %>% distinct()

  ## remove all cases where < 2 target/query cell lines are left
  valid_pairs <- combis_bf_batch %>% distinct(target, query, cellline, type) %>%
    group_by(target, query) %>% count(type) %>%
    summarise(nquery=n[1], ntarget=n[2]) %>% ungroup() %>%
    filter(nquery>2, ntarget>2) %>% dplyr::select(target, query) %>% mutate(valid=1)

  ## remove invalid pairs
  combis_bf_batch <- combis_bf_batch %>% inner_join(valid_pairs) %>% dplyr::select(-valid)
```

```
  ## write chunks to file for testing on a cluster
  write_tsv(combis_bf_batch,
            paste0('../data/sgi_test/chunks/data_23072017/test_data_',
                   gsub('\\.', '', as.character(x$query[1])), '.tsv.gz'))

  return(combis_bf_batch)
})

bfs_sgi <- combis %>% bind_rows()
```

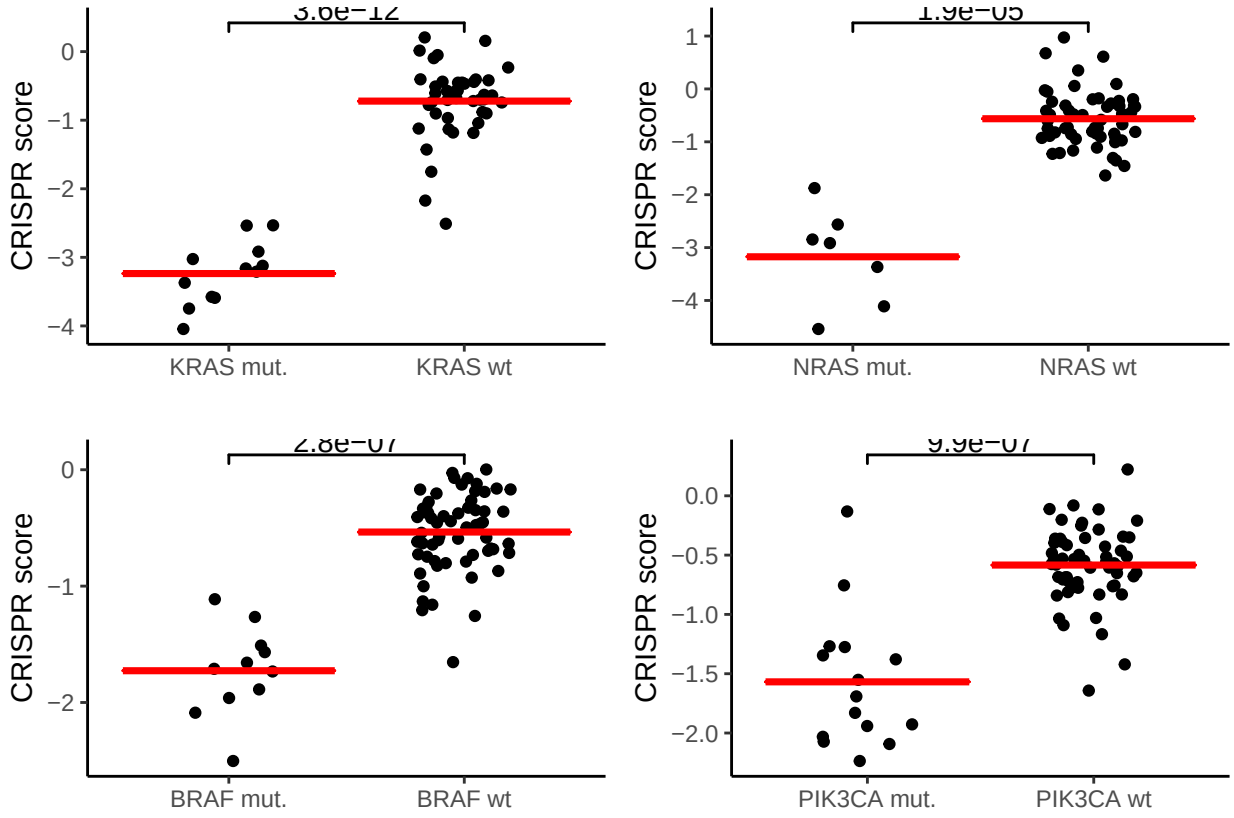
We define a function that can generate jittered dot plots of specific interactions that will be useful to examine relationships of interest in downstream analyes.

```
plot_interaction <- function(t, q){
  data <- bfs_norm %>% filter(symbol == t) %>%
    mutate(cellline = toupper(cellline)) %>%
    left_join(somatic_filtered %>% filter(symbol == q) %>% mutate(mutated=1) %>%
                dplyr::select(symbol, cellline, mutated) %>%
                dplyr::rename(query=symbol)) %>%
    mutate(type=ifelse(is.na(mutated), paste(q, 'wt'),
                       paste(q, 'mut.')), query=q) %>%
    dplyr::select(-c(mutated, condition)) %>% distinct()

  ## plot
  return(list(
    plot=data %>% ggplot(aes(x=type, y=BF)) + geom_jitter(width=0.2) +
      theme_classic() +
      stat_summary(fun.y='mean', fun.ymin='mean',
                   fun.ymax='mean', colour='red', geom='crossbar') +
      xlab('') + ylab('CRISPR score') +
      geom_signif(comparisons=list(c(paste(q, 'wt'),
                                     paste(q, 'mut.'))), map_signif_level=F),
    data=data
  ))
}
```

2.2.1 Examples of genetic interactions - control combinations

We plot known oncogene dependencies as controls, such as KRAS-KRAS and others.

```
p1 <- plot_interaction('KRAS', 'KRAS')$plot
p2 <- plot_interaction('BRAF', 'BRAF')$plot
p3 <- plot_interaction('NRAS', 'NRAS')$plot
p4 <- plot_interaction('PIK3CA', 'PIK3CA')$plot
p5 <- plot_interaction('TP53', 'TP53')$plot
p6 <- plot_interaction('MDM2', 'TP53')$plot
multiplot(p1, p2, p3, p4, cols=2)
```



2.3 Downstream analysis of SGI test results

We load the results generated by the compute cluster back into the R session for downstream processing.

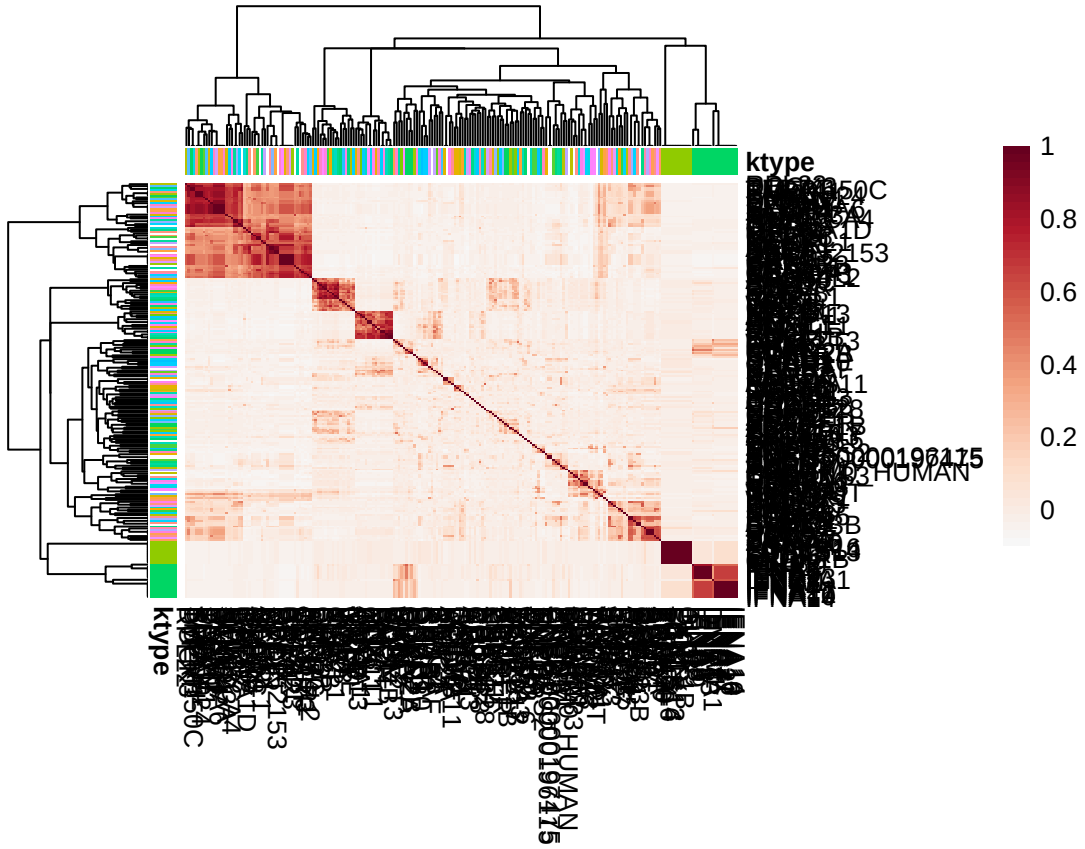
```
## load lme4 results from the data chunks calculated on the cluster
data('mut_vs_non_lme', package='CGIMhd17')
## karyotype annotation for different genes
data('ktype_map', package='CGIMhd17')
```

2.3.1 Correlations of query genes

Some query genes are highly correlated because they, for example, are located on the chromsome and get deleted together in the context of one loss of copy number event. This can confound the analysis. We visualize these correlations by heat map.

```
ktype_anno <- ktype_map %>% filter(symbol %in% mut_vs_non_lme$query) %>%
  distinct() %>% drop_na() %>% data.frame() %>% `rownames<-`(.$symbol) %>% dplyr::select(-symbol)

query_cor <- mut_vs_non_lme %>% dplyr::select(target, query, statistic) %>%
  spread(query, statistic) %>% data.frame() %>% `rownames<-`(.$target) %>%
  dplyr::select(-target) %>% cor(use='pairwise.complete.obs')

##plot correlation heatmap with karyotype anotation
cols <- rev(c('#67001f','#b2182b','#d6604d','#f4a582','#fddbc7','#f7f7f7'))
```

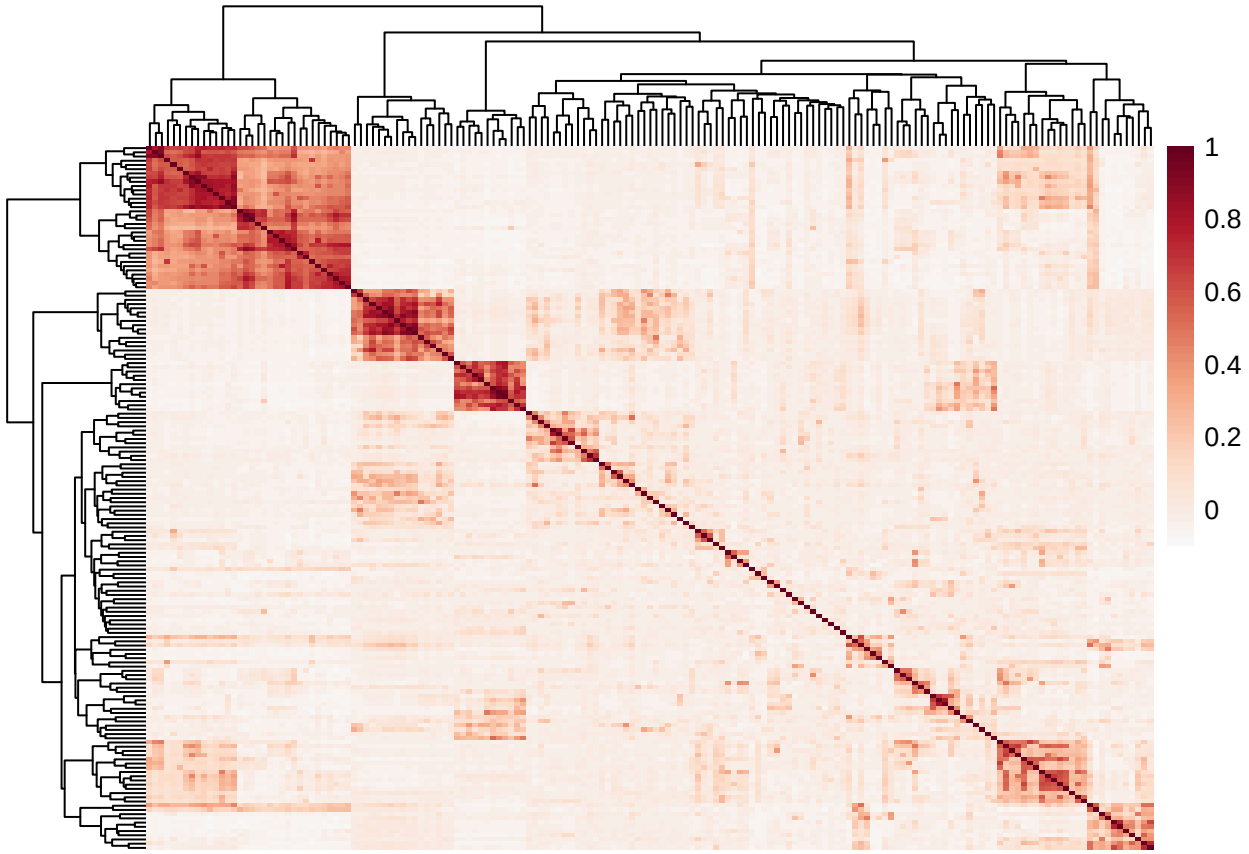
```
pheatmap(query_cor, annotation_row=ktype_anno, annotation_col=ktype_anno,
         color = colorRampPalette(cols)(50))
```



To avoid bias and false postivie interactions we aggregate identical query genes into meta genes that can be used for downstream calculations.

```
## coerce redundant queries into a metagene query
sgi_filtered <- mut_vs_non_lme %>%
  left_join(query_cor %>% data.frame() %>% mutate(query_one=rownames(.)) %>%
              gather(query_two, corr, -query_one) %>% filter(corr==1) %>%
              group_by(query_one) %>% mutate(redundant=paste(query_two, collapse='_')) %>%
              ungroup() %>% dplyr::select(query_one, redundant) %>%
              dplyr::rename(query=query_one))

## save query mapping
query_map <- sgi_filtered %>% distinct(query, redundant)

sgi_filtered <- sgi_filtered %>%
  mutate(query=redundant) %>% group_by(target, query) %>%
  summarise(statistic=mean(statistic), p.value=mean(p.value)) %>% ungroup() %>%
  arrange(p.value) %>% mutate(fdr=p.adjust(p.value, method='fdr'))

## plot heatmap of aggregated queries
sgi_filtered %>% dplyr::select(-c(fdr, p.value)) %>% spread(query, statistic) %>%
  data.frame() %>% `rownames<-`(.$target) %>% dplyr::select(-target) %>%
  cor(use='pairwise.complete.obs') %>%
  pheatmap(show_rownames = F, show_colnames=F, border_color=NA,
```

```
          color = colorRampPalette(cols)(50))
```



We need to also adapt the phenotype data accordingly. For all further analyses, however, we only need the combined phenotypes (ko + mutated), which are sufficient to compute π-scores (Laufer et al. 2013). Hence we can exclude the single phenotypes to save working memory.

```
## load adapted data from file as input data are too large to be included.
data('bfs_sgi', package='CGIMhd17')
```

2.3.2 Binary distance between pairwise queries comparing shared cell lines

We calculate binary distances between the queries in terms of the cell lines they are derrived from. This later allows us to exclude highly similar (but not fully identical) query genes to avoid bias in the genetic network analysis.

```
cl_bin_mat <- somatic_filtered %>% distinct(symbol, cellline) %>% mutate(has =1) %>%
  spread(cellline, has) %>% data.frame() %>% `rownames<-`(NULL) %>%
  mutate_all(funs(ifelse(is.na(.), 0, .))) %>% column_to_rownames('symbol')

pw_dist <- as.matrix(dist(cl_bin_mat, method='binary', diag=T, upper=T))
redundant_queries <- tibble(query = findCorrelation(1-pw_dist, cutoff = 0.7, names=T)) %>%
  inner_join(query_map) %>% .$redundant %>% unique
```

3 Quantification of interactions

3.1 Calculation of π-scores

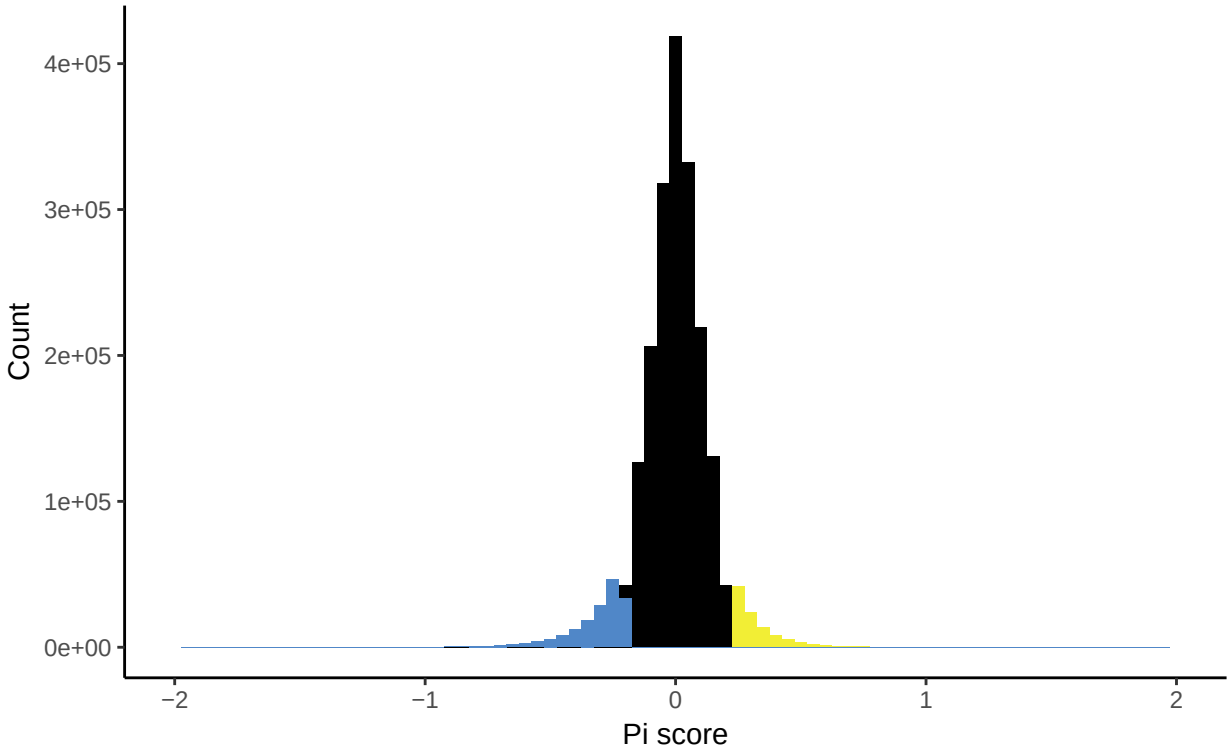
Genetic interactions that are stastically significant can often have a very low effect size that might limit their interpretability and biological relevance. Therefore it is important to be able to quantify the interaction strength. One measure that can be used to achieve this is the π-score, which was introduced in previous studies. The π-score can be calculated from the fitness effects of gene perturbations in mutated cell lines under the assumption that genes, on average, do not interact. A function to compute the π-score is implemented in the Bioconductor package HD2013SGI (Laufer et al. 2013). We use this function in the next step to compute π-scores for the CRISPR screening data. The function computing the π-scores also provides information about corresponding main effects.

```
## calculation of pi-scores for CRISPR data
## only combis that are supported by at least 3 different cell lines
pl_out <- bfs_sgi %>% dplyr::select(target,query,BF) %>%
  dplyr::rename(fitness=BF) %>%
  dplyr::group_by(target,query) %>% summarise(fitness=mean(fitness)) %>% ungroup() %>%
  spread(query,fitness) %>%
  data.frame() %>% `rownames<-`(.$target) %>% dplyr::select(-target) %>%
  as.matrix() %>% HD2013SGImaineffects()

all_effects <- pl_out$pi %>% melt %>%
  `colnames<-`(c('target', 'query', 'pi')) %>% tbl_df %>%
  inner_join(pl_out$targetMainEffect %>% melt %>% mutate(target=rownames(.)) %>%
               dplyr::rename(target_main=value)) %>%
  inner_join(cbind(pl_out$queryMainEffect, pl_out$pi %>% colnames()) %>% tbl_df %>%
               `colnames<-`(c('query_main', 'query')) %>%
               mutate(query_main=as.numeric(query_main)))

pi_scores <- bfs_sgi %>% dplyr::select(target, query, BF) %>%
  mutate(fitness=BF) %>% dplyr::select(-BF) %>%
  group_by(target, query) %>% summarise(fitness=mean(fitness)) %>% ungroup() %>%
  mutate(fitness=scale(fitness)[,1]) %>% dplyr::rename(measured=fitness) %>%
  inner_join(all_effects)
```

3.1.1 Visualization of the π-score distribution

To visualize the results of the π-score calculation, we first plot the overall distribution of the π-scores. We choose a relevance threshold of 0.2 indicating that an interaction is strong enough for consideration. Positive interactions (postive π-score) are illustrated in yellow colour and negative interactions (negative π-score), indicating synthetic lethality are drawn in blue. We further plot in a bar plot, based on this threshold, how many interactions genes tend to have.

```
## plot pi-score distribution
## negative interaction: #5087c8
## positive interaction: #f2ee35
## distribution of bayes factors (essentiality score)
ggplot(pi_scores, aes(pi)) +
  geom_histogram(data=subset(pi_scores,pi>=0.2), fill='#f2ee35', binwidth=0.05) +
  geom_histogram(data=subset(pi_scores,pi< 0.2 & pi> -0.2), fill='black', binwidth=0.05) +
  geom_histogram(data=subset(pi_scores,pi<= -0.2), fill='#5087c8', binwidth=0.05) +
  theme_classic() + ylab('Count') + xlab('Pi score') + xlim(-2, 2)
```



3.2 Combining p-values and pi-scores

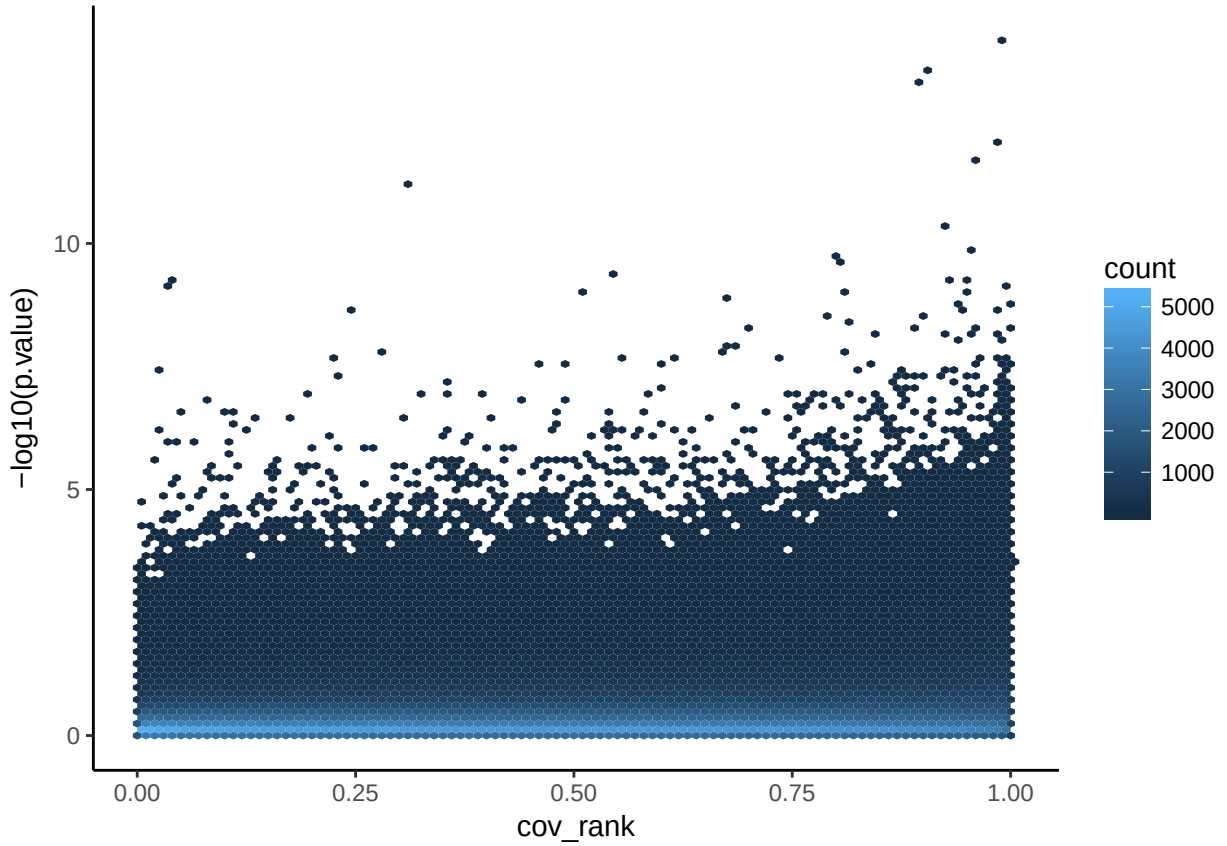
We combine the P-values determined in previous steps with the calculated π-scores into an interaction dataset.

```
## volcano plot based on effects used for the lme test
all_interactions <- pi_scores %>% inner_join(sgi_filtered) %>% inner_join(
  bfs_sgi %>% group_by(target, query) %>%
    summarise(var_bf=var(BF)) %>% ungroup() %>% distinct()
)
```

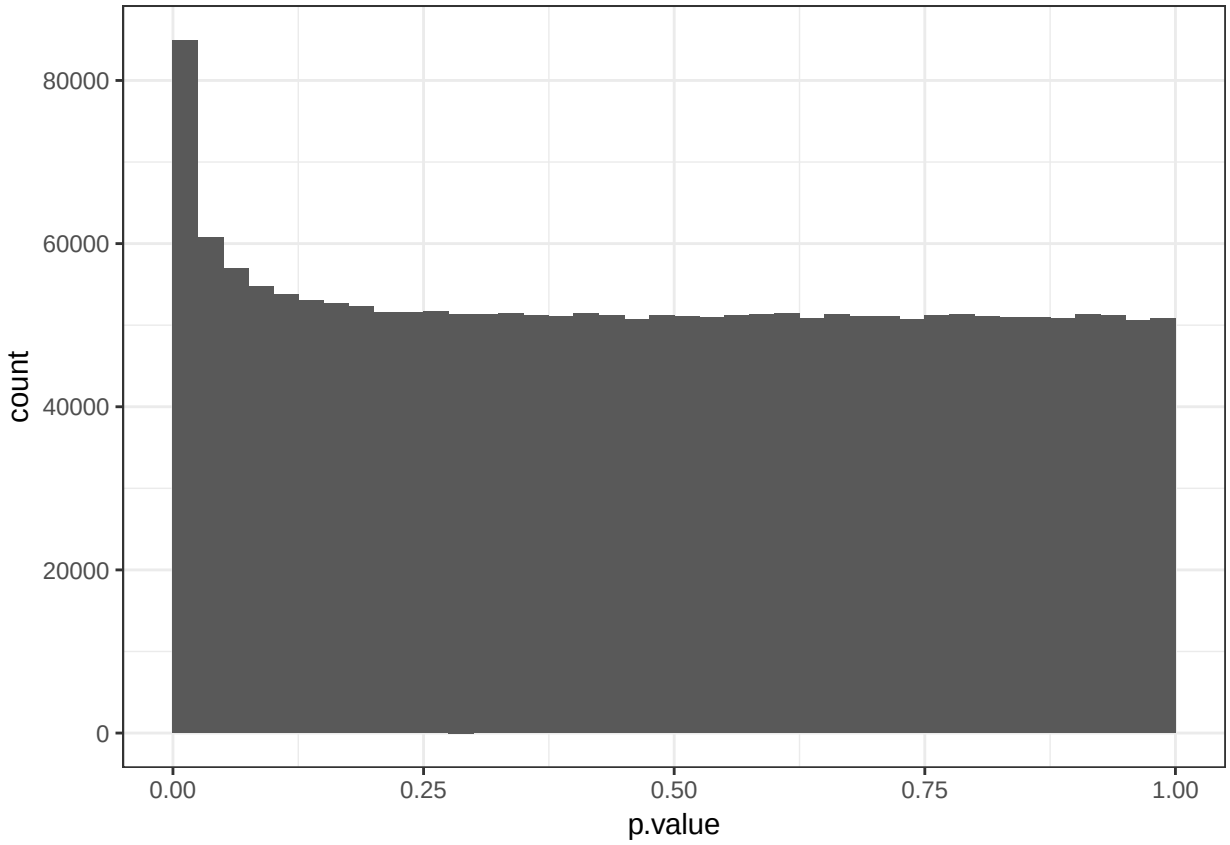
3.2.1 Multiple testing correction

We can next adjust the P-values for multiple testing using Independent Hypothesis Weighting (IHW) (Ignatiadis et al. 2016). We generate in addition a number of control plots that justify the use of the covariate. As a covariate we use the variance of CRISPR scores of the altered group of cell lines.

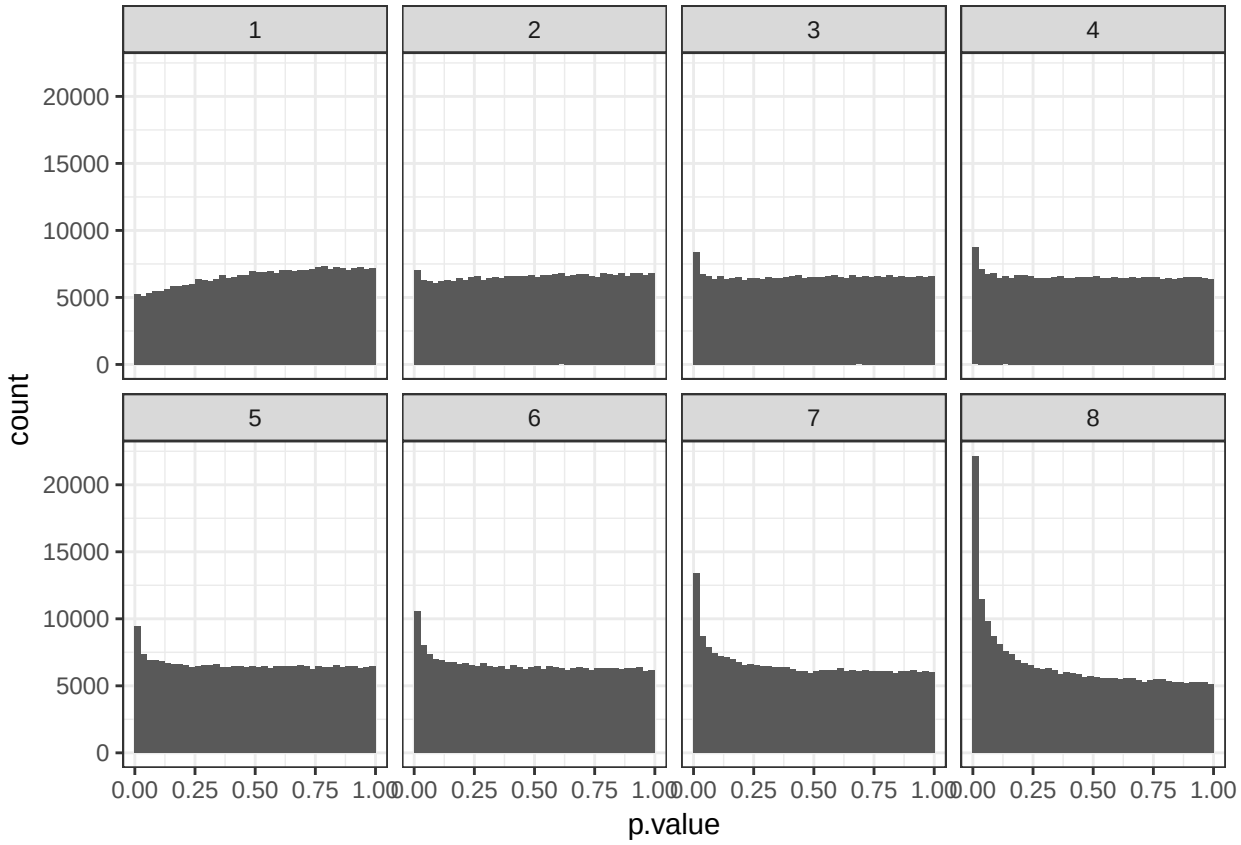
```
## is the pi-score an appropriate covariate?
## scatter plot
all_interactions %>% arrange(var_bf) %>% mutate(cov_rank=1:n()/n()) %>%
  ggplot(aes(cov_rank, -log10(p.value))) + geom_hex(bins=100) + theme_classic()
```



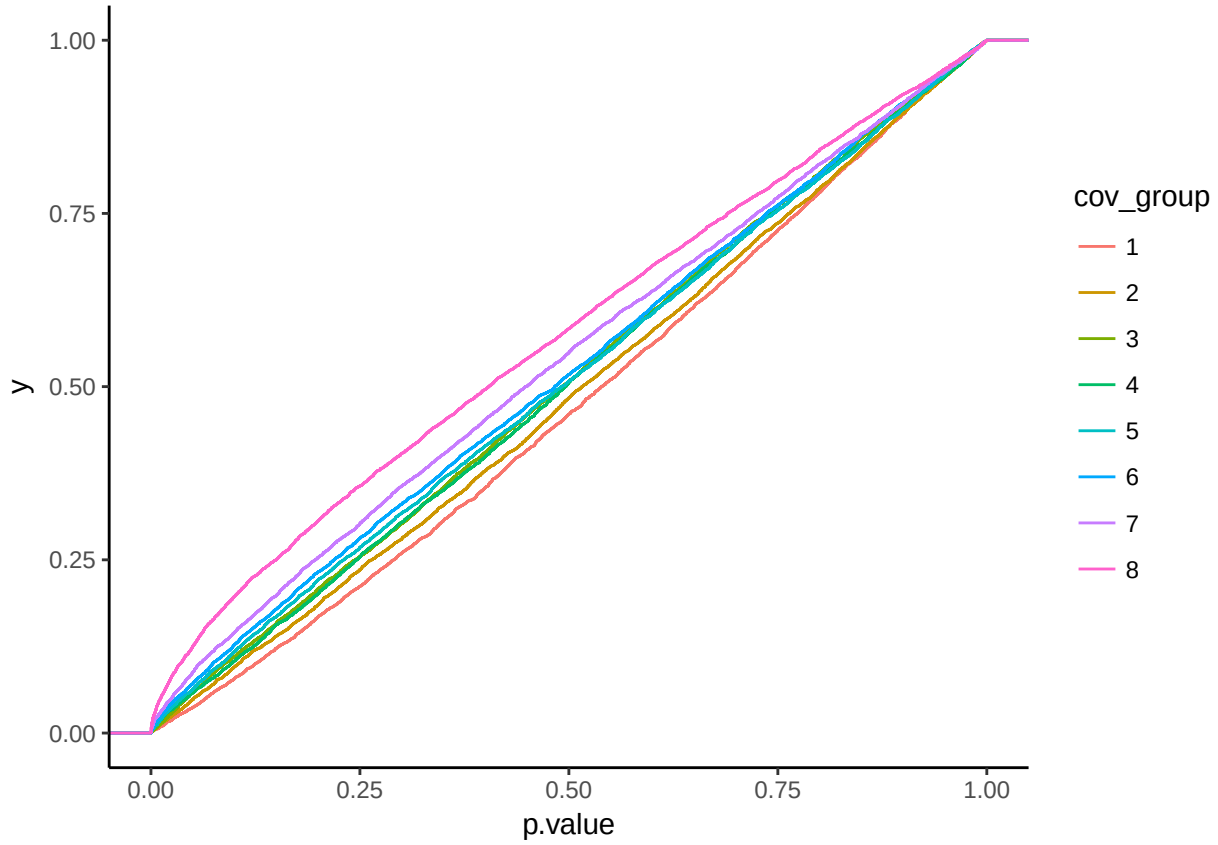

```
## stratified histograms
## all observations
all_interactions %>% ggplot(aes(p.value)) + geom_histogram(binwidth=0.025, boundary=0) + theme_bw()
```



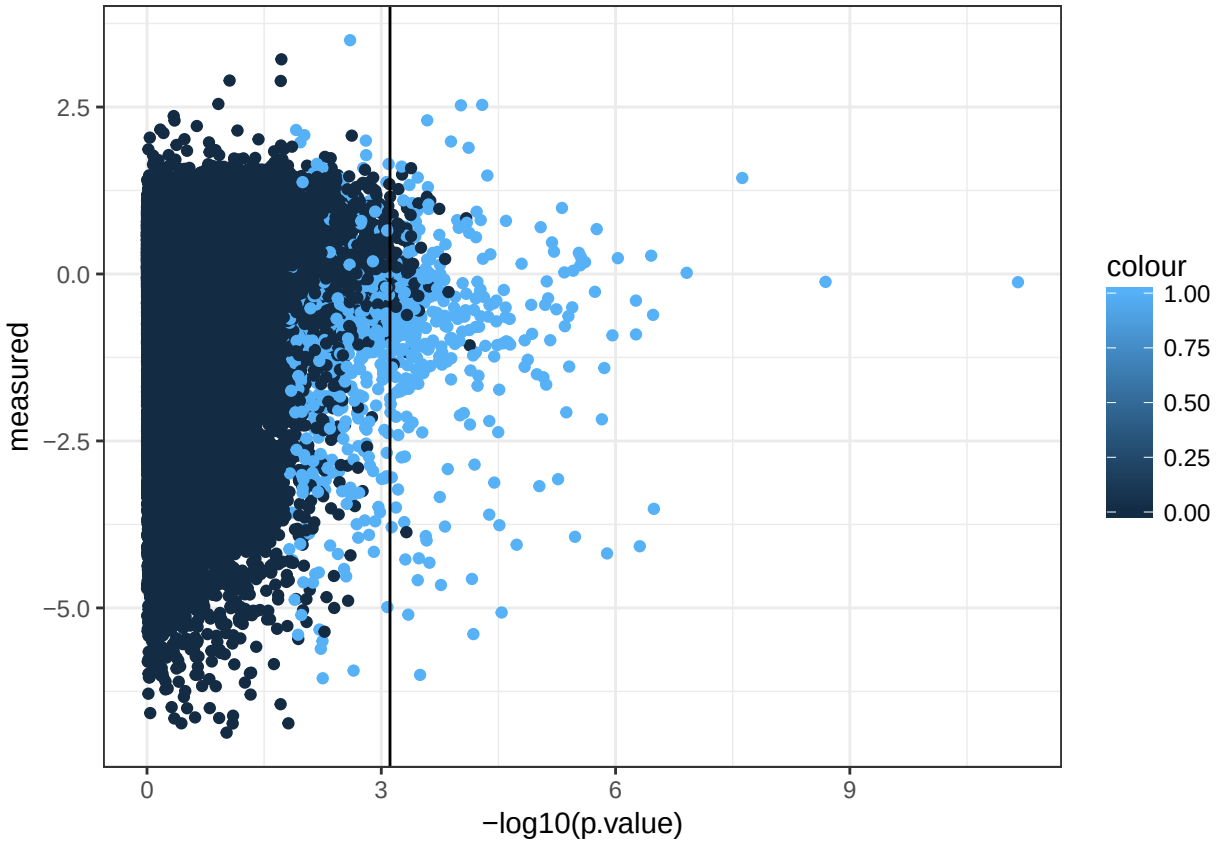

```
## groups split by covariate
all_interactions$cov_group <- groups_by_filter(abs(all_interactions$var_bf), 8)
ggplot(all_interactions, aes(x=p.value)) +
  geom_histogram(binwidth = 0.025, boundary = 0) +
  facet_wrap( ~ cov_group, nrow = 2) + theme_bw()
```



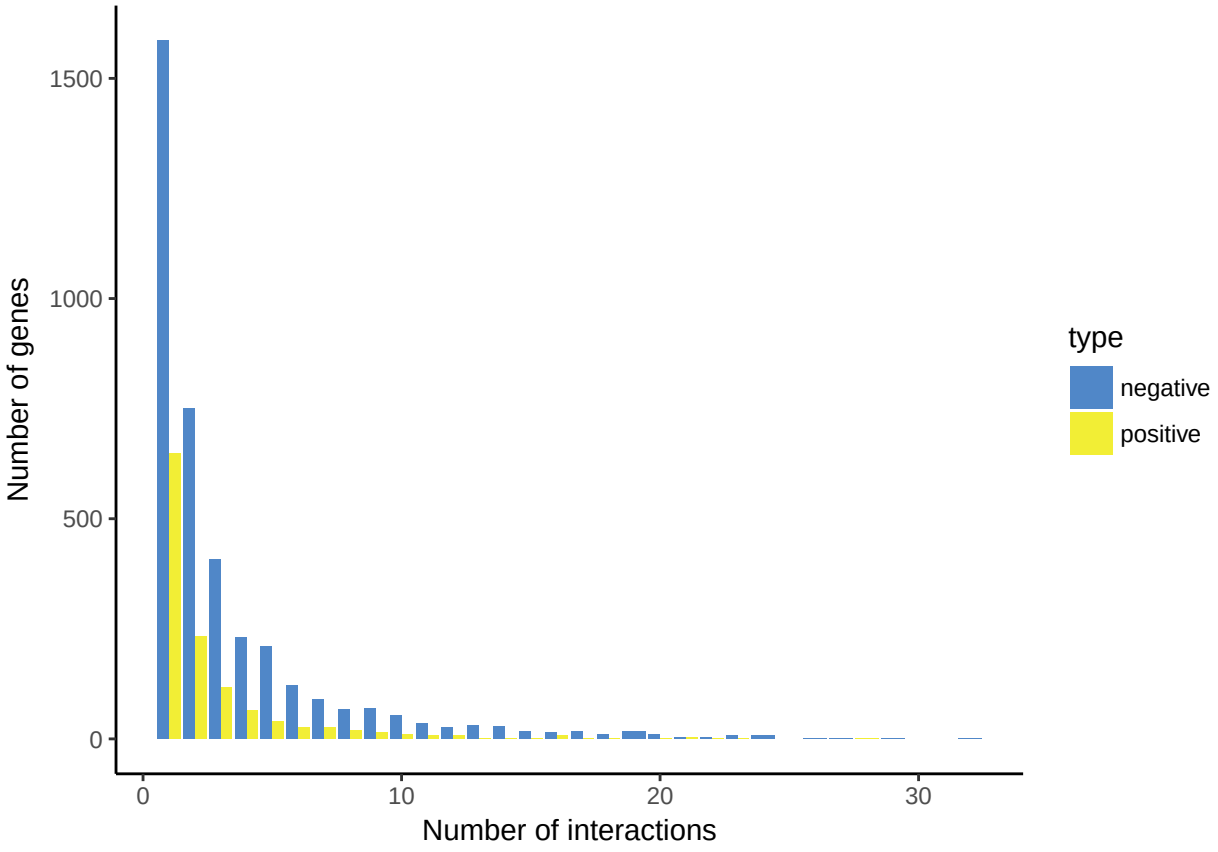

```
## ecdf plot
ggplot(sample_n(all_interactions, 50000), aes(x = p.value, col = cov_group)) +
  stat_ecdf(geom = "step") + theme_classic()
```



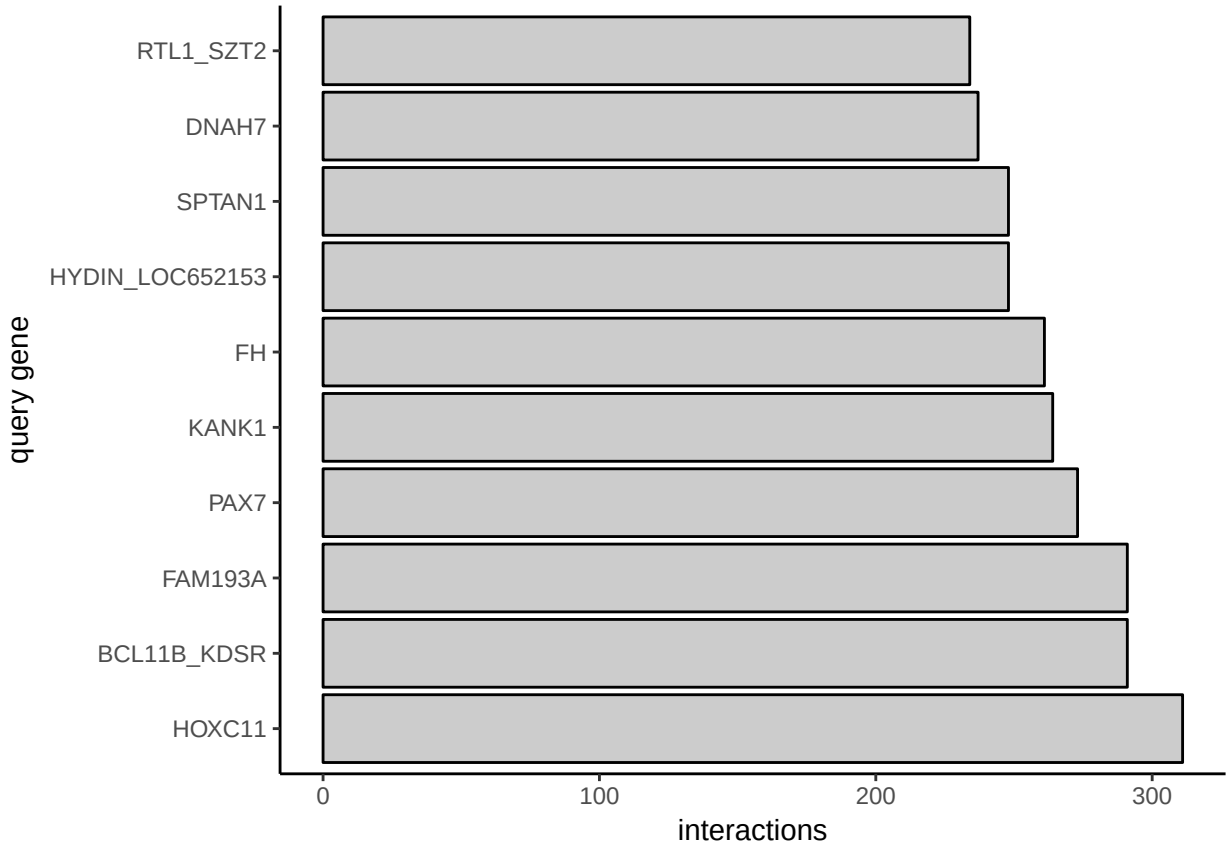

```
ihw_obj <- ihw(p.value ~ var_bf, data=all_interactions, alpha=0.2)
all_interactions <- all_interactions %>% mutate(padj=adj_pvalues(ihw_obj))

## plot comparison BH vs IHW
all_interactions %>% mutate(colour=ifelse(padj<0.2, 1, 0)) %>% sample_n(100000) %>%
  ggplot(aes(-log10(p.value), measured)) + geom_point(aes(colour=colour)) +
  geom_vline(aes(xintercept=filter(., fdr<0.2) %>%
                   arrange(desc(fdr)) %>% .$p.value %>% .[1] %>% log10 %>% `*`(-1))) +
  theme_bw()
```




```
## plot positive and negative interactions as bar chart
all_interactions %>% mutate(type=ifelse(pi>0.2 & padj < 0.2, 'positive',
                                        ifelse(pi< -0.2 & padj < 0.2, 'negative', 'none'))) %>%
  filter(type != 'none') %>% count(target, type) %>%
  group_by(n) %>% count(type) %>% ungroup() %>%
  ggplot(aes(x=n, y=nn, fill=type)) +
  geom_bar(stat='identity', position='dodge') +
  theme_classic() +
  scale_fill_manual(values=c('#5087c8', '#f2ee35')) +
  xlab('Number of interactions') + ylab('Number of genes')
```




```
## plot 10 genes with most interactions at padj < 0.2
all_interactions %>% filter(padj < 0.2) %>% count(query) %>%
  arrange(desc(n)) %>% dplyr::slice(1:10) %>%
  ggplot(aes(y=n, x=factor(query, levels=arrange(., desc(n)) %>% .$query))) +
  geom_bar(fill='#cccccc', colour='black', stat='identity') +
  coord_flip() + theme_classic() +
  xlab('query gene') + ylab('interactions')
```



3.3 Network chart of gene set enrichment

We write a number of interactions with well-characterized query genes to a file using a cutoff of FDR < 0.2. We visualize these interactions in Cytoscape (Shannon et al. 2003) and perform gene set enrichment analyses using ConsensusPathDB (Kamburov et al. 2013) as shown in Figure 3.

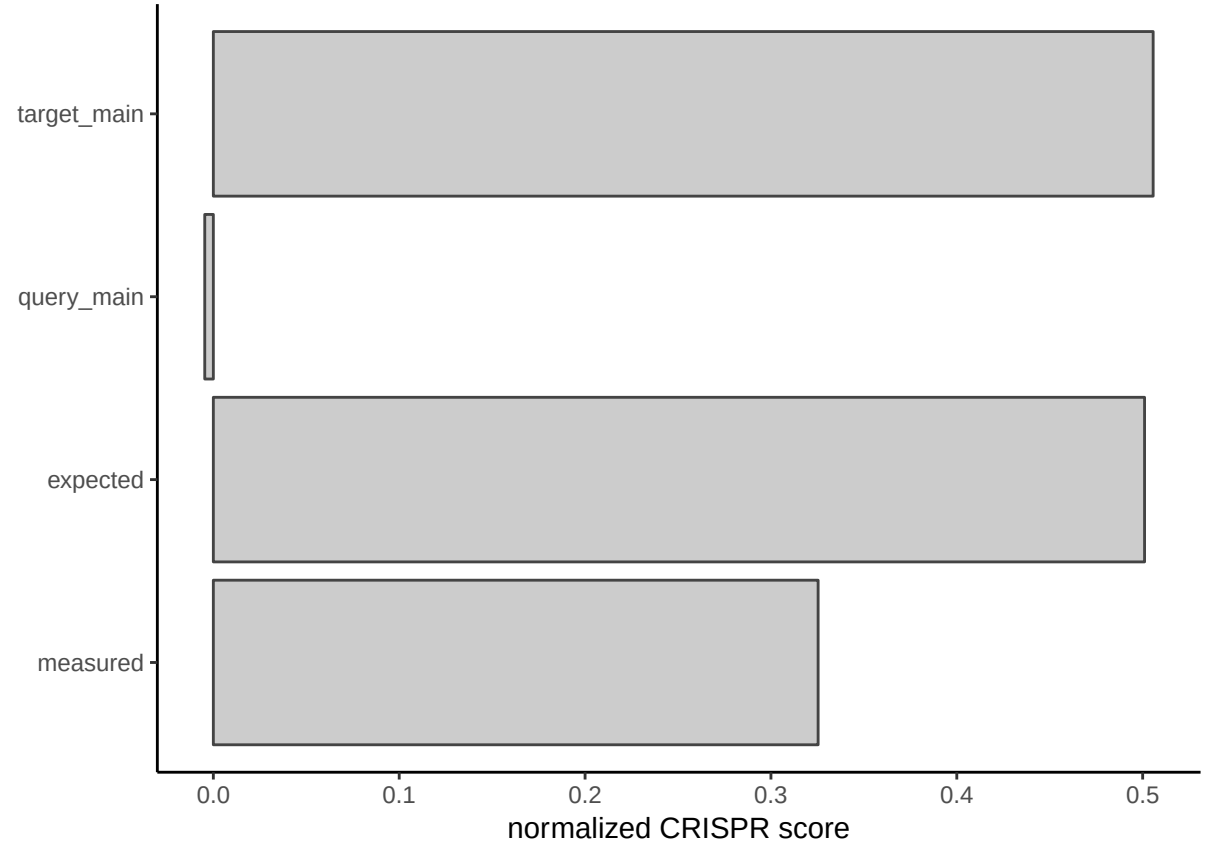
```
## some selected queries to look at
queries <- c('TCF7L2', 'TP53', 'KRAS', 'NRAS',
             'PIK3CA', 'HNF1A', 'MYB',
             'BRAF', 'SMAD4', 'RNF43', 'BCL2')

selected <- all_interactions %>% filter(query %in% queries, padj < 0.2) %>%
  group_by(target) %>% mutate(count=n()) %>% ungroup() %>%
  mutate(count=ifelse(count == 1, 0.1, 1))

## write to tsv, visualize in Cytoscape
write_tsv(selected %>% dplyr::select(target, query, count), 'selected_queries_interactions.tsv')
```

3.4 Epistasis bar plot for selected interactions

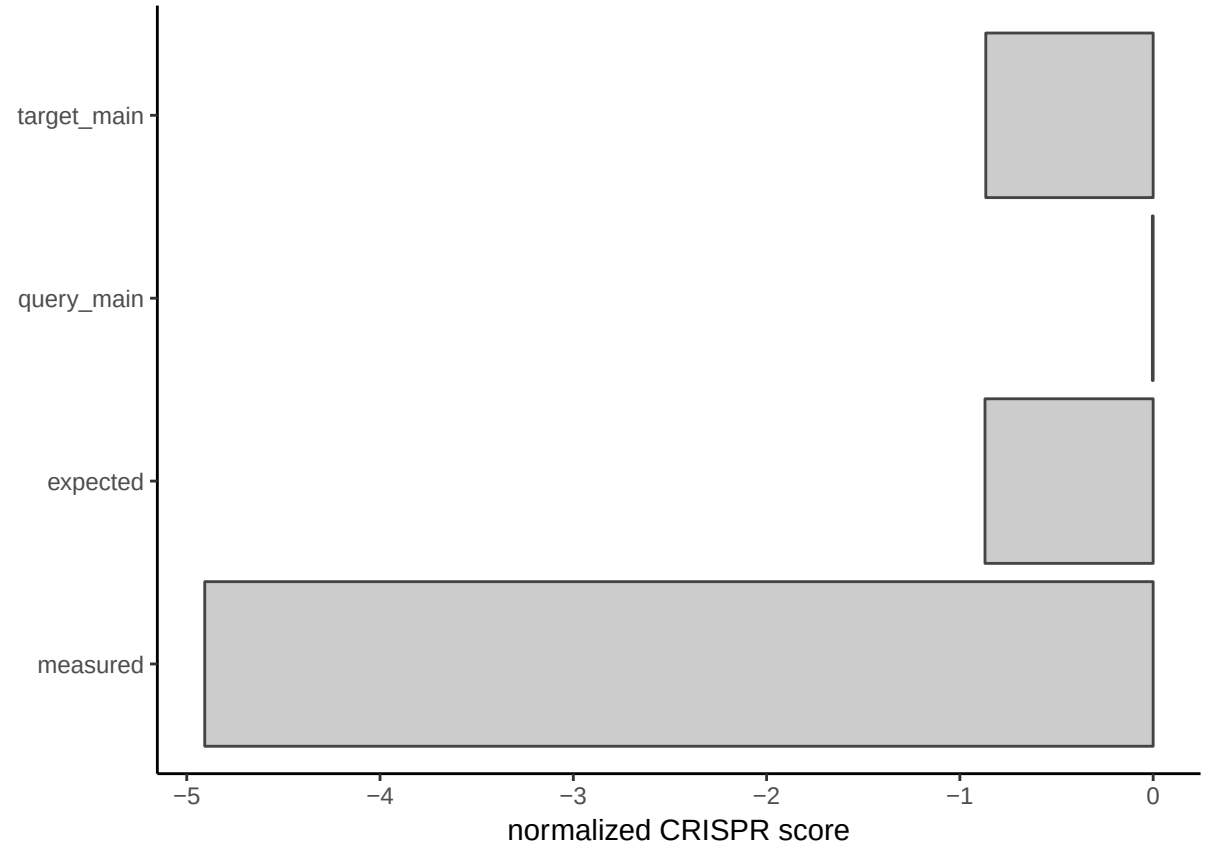
We visualize a number of control interactions as bar plots showing main effects, expected phenotype and measured phenotype.

```
## example of TP53-TP53 interaction
all_interactions %>% filter(target == 'TP53', query == 'TP53') %>%
```

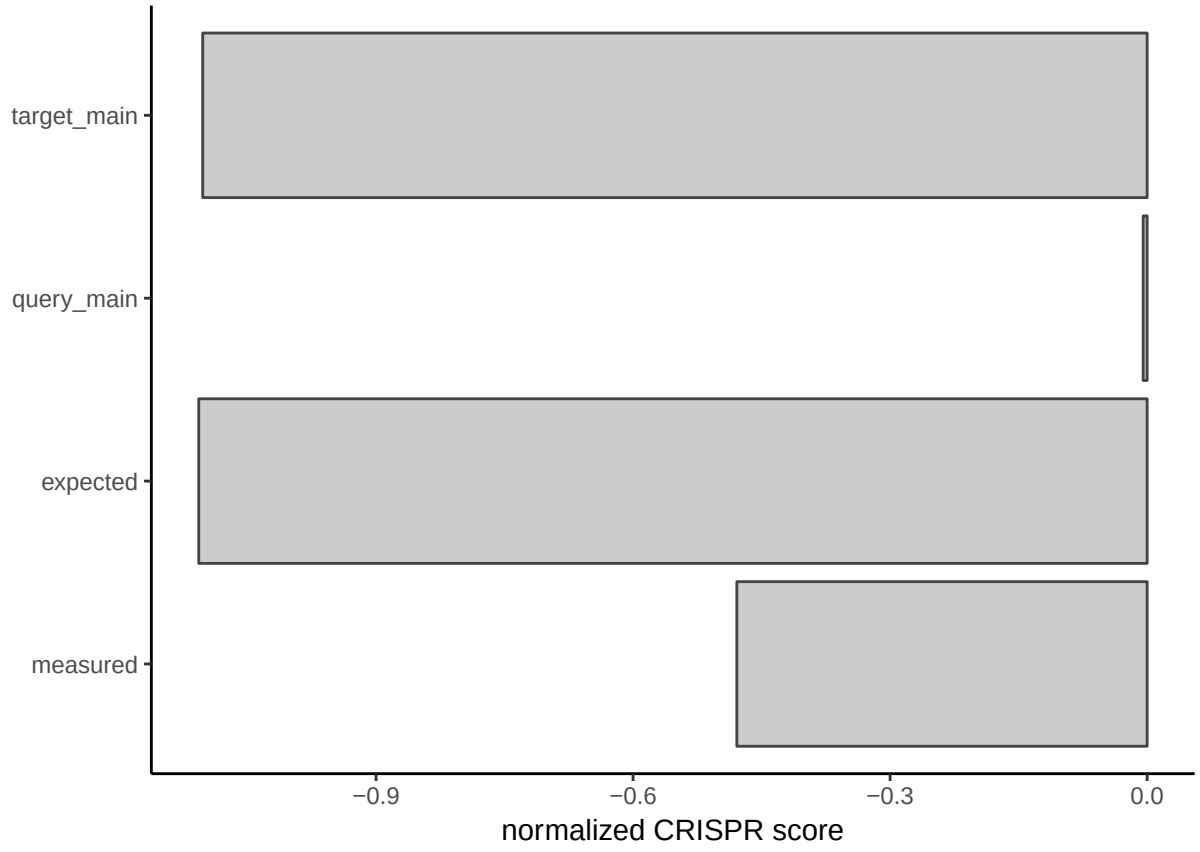
```
  mutate(expected = target_main + query_main) %>%
  dplyr::select(target, query, target_main, query_main, expected, measured) %>%
  gather(type, value, target_main:measured) %>%
  ggplot(aes(factor(type,
                    levels=rev(c('target_main', 'query_main', 'expected', 'measured'))), value)) +
  geom_bar(stat='identity', colour = '#444444', fill='#cccccc') +
  theme_classic() + coord_flip() +
  ylab('normalized CRISPR score') + xlab('')
```




```
## example of KRAS-KRAS interaction
all_interactions %>% filter(target == 'KRAS', query == 'KRAS') %>%
  mutate(expected = target_main + query_main) %>%
  dplyr::select(target, query, target_main, query_main, expected, measured) %>%
  gather(type, value, target_main:measured) %>%
  ggplot(aes(factor(type,
                    levels=rev(c('target_main', 'query_main', 'expected', 'measured'))), value)) +
  geom_bar(stat='identity', colour = '#444444', fill='#cccccc') +
  theme_classic() + coord_flip() +
  ylab('normalized CRISPR score') + xlab('')
```



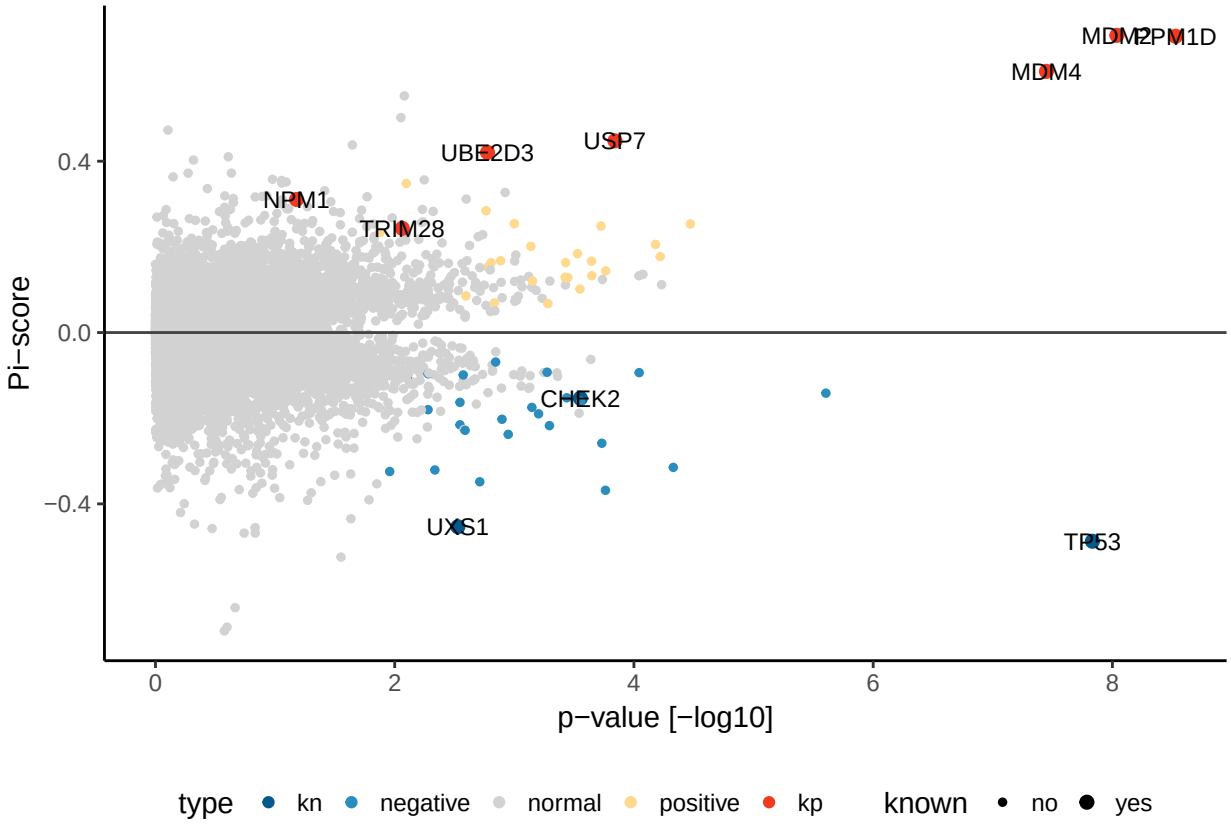

```
## example of MDM2-TP53 interaction
all_interactions %>% filter(target == 'MDM2', query == 'TP53') %>%
  mutate(expected = target_main + query_main) %>%
  dplyr::select(target, query, target_main, query_main, expected, measured) %>%
  gather(type, value, target_main:measured) %>%
  ggplot(aes(factor(type,
                    levels=rev(c('target_main', 'query_main', 'expected', 'measured'))), value)) +
  geom_bar(stat='identity', colour = '#444444', fill='#cccccc') +
  theme_classic() + coord_flip() +
  ylab('normalized CRISPR score') + xlab('')
```



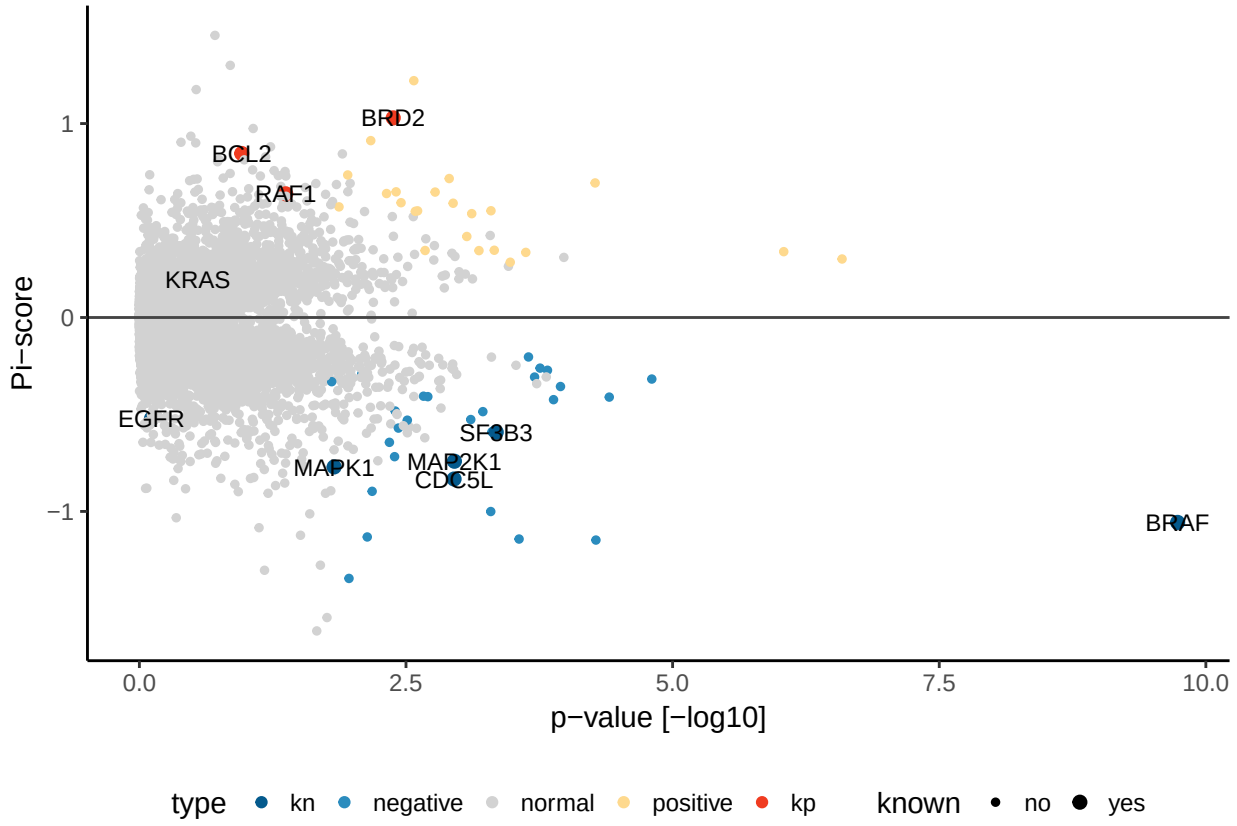
3.5 Volcano plots for specific queries

We further draw volcano plots showing the results of a number of selected query genes such as TP53, BRAF and the Wnt-mutation meta-gene, which consists of aggregated APC and RNF43 mutated cell lines.

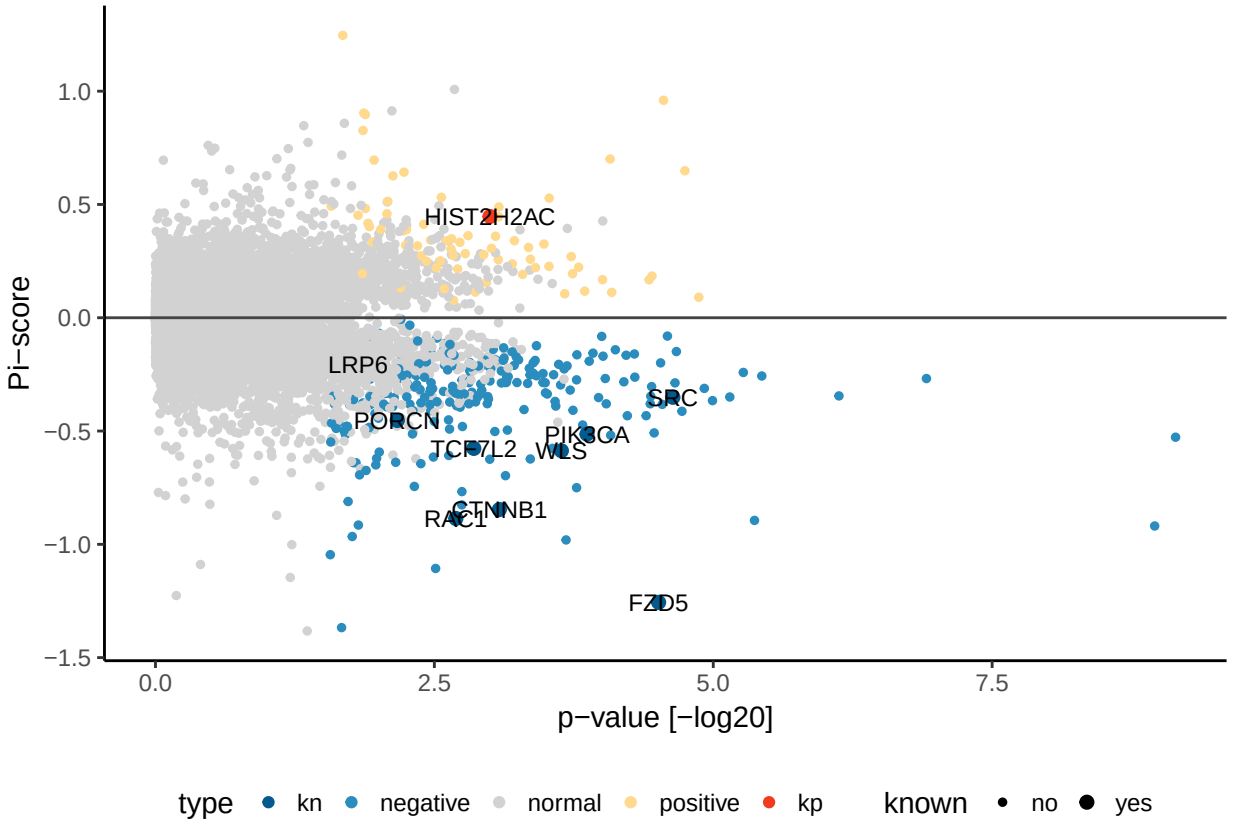
```
## volcano plots for certain queries
## TP53
tp53 <- c('MDM2', 'MDM4', 'PPM1D', 'NPM1', 'RECQL4', 'UBE2D3', 'TP53', 'USP7', 'CHEK2', 'TRIM28', 'UXS1')
tp53_inter <- all_interactions %>% filter(query=='TP53') %>% dplyr::select(-c(fdr))
tp53_inter %>% mutate(known=ifelse(target %in% tp53, 'yes', 'no'),
         label=ifelse(known=='yes', target, ''),
         type=ifelse(padj < 0.2 & pi > 0, 'positive',
                     ifelse(padj < 0.2 & pi < 0, 'negative', 'normal')),
         type=ifelse(known=='yes' & pi > 0, 'kp', ifelse(known=='yes' & pi < 0, 'kn', type)),
         type=factor(type, levels=c('kn', 'negative', 'normal', 'positive', 'kp'))) %>%
  ggplot(aes(x=-log10(p.value), y=pi, label=label)) +
  geom_point(aes(colour=type, size=known)) +
  geom_hline(yintercept=0, colour='#444444') + theme_classic() +
  scale_size_manual(values=c(1,2)) + geom_text(size=3) +
  scale_colour_manual(values=c('#045a8d', '#2b8cbe', '#d2d2d2', '#fed98e', '#f03b20')) +
  theme(legend.position='bottom') + xlab('p-value [-log10]') + ylab('Pi-score')
```




```
## BRAF interaction
braf <- c('BRAF', 'MAP2K1', 'RAF1', 'MAPK1', 'KRAS', 'EGFR', 'CDC5L', 'SF3B3', 'BRD2', 'BCL2')
braf_inter <- all_interactions %>% filter(query=='BRAF') %>% dplyr::select(-c(fdr))
braf_inter %>%
  mutate(known=ifelse(target %in% braf, 'yes', 'no'),
         label=ifelse(known=='yes', target, ''),
         type=ifelse(padj < 0.2 & pi > 0, 'positive',
                     ifelse(padj < 0.2 & pi < 0, 'negative', 'normal')),
         type=ifelse(known=='yes' & pi > 0, 'kp', ifelse(known=='yes' & pi < 0, 'kn', type)),
         type=factor(type, levels=c('kn', 'negative', 'normal', 'positive', 'kp'))) %>%
  ggplot(aes(x=-log10(p.value), y=pi, label=label)) +
  geom_point(aes(colour=type, size=known)) +
  geom_hline(yintercept=0, colour='#444444') + theme_classic() +
  scale_size_manual(values=c(1,2)) + geom_text(size=3) +
  scale_colour_manual(values=c('#045a8d', '#2b8cbe', '#d2d2d2', '#fed98e', '#f03b20')) +
  theme(legend.position='bottom') + xlab('p-value [-log10]') + ylab('Pi-score')
```




```
## wnt mutated for wnt genes
wntmut <- c('WLS', 'FZD5', 'CTNNB1', 'TCF7L2', 'PORCN', 'LRP6', 'RAC1', 'SRC', 'PIK3CA', 'HIST2H2AC')
wntmut_inter <- all_interactions %>% filter(query=='WNTMUT') %>% dplyr::select(-c(fdr))
wntmut_inter %>%
  mutate(known=ifelse(target %in% wntmut, 'yes', 'no'),
         label=ifelse(known=='yes', target, ''),
         type=ifelse(padj < 0.25 & pi > 0, 'positive',
                     ifelse(padj < 0.25 & pi < 0, 'negative', 'normal')),
         type=ifelse(known=='yes' & pi > 0, 'kp', ifelse(known=='yes' & pi < 0, 'kn', type)),
         type=factor(type, levels=c('kn', 'negative', 'normal', 'positive', 'kp'))) %>%
  ggplot(aes(x=-log10(p.value), y=pi, label=label)) +
  geom_point(aes(colour=type, size=known)) +
  geom_hline(yintercept=0, colour='#444444') + theme_classic() +
  scale_size_manual(values=c(1,2)) + geom_text(size=3) +
  scale_colour_manual(values=c('#045a8d', '#2b8cbe', '#d2d2d2', '#fed98e', '#f03b20')) +
  theme(legend.position='bottom') + xlab('p-value [-log20]') + ylab('Pi-score')
```



3.6 Enrichemnt of top interacting queries

Top 40 query genes were selected and enrichment analysis for GO molecular function was performed using ENRICHR (Kuleshov et al. 2016). We load the exported Enrichr data back into R to make a bar plot.

```
data('enrichr_out', package='CGIMhd17')
enrichr_out %>% arrange(desc(`Combined Score`)) %>% dplyr::slice(1:10) %>%
  ggplot(aes(factor(Term, levels=arrange(., desc(`Combined Score`)) %>% .$Term), `Combined Score`)) +
  geom_bar(stat='identity', fill='#bbbbbb', colour='black') + theme_classic() +
  theme(axis.text.x = element_text(angle=45, hjust=1)) + xlab('')
```

4 Experimental follow-up of candidate Wnt regulators

4.1 Network plot motivating follow up of candidate genes

From the total set of interactions we select the genes chosen for follow-up as well as key regulator genes of the Wnt pathway and show that they are connected to RNF43 by interactions with high π-scores. First we select all relvant interactions, including the follow-up genes and some known Wnt-regulators as controls.

```
## select interactions motivating the follow-up
links_fu <- all_interactions %>%
  filter(target %in% c('C2orf15', 'GANAB', 'PRKCSH', 'UGP2',
                       'CTNNB1', 'TCF7L2', 'WLS', 'PORCN', 'FZD5'),
         query %in% c('RNF43')) %>%
  filter(abs(pi) > 0.2)
```

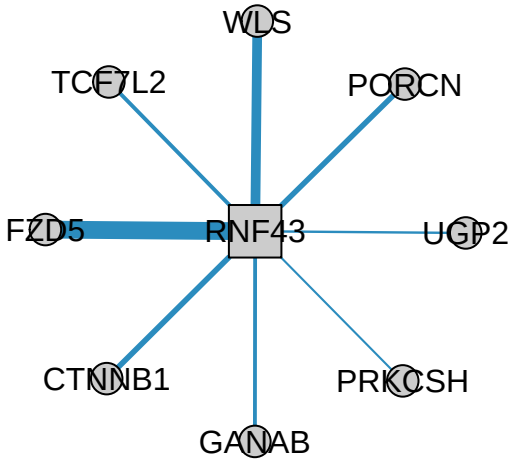
Next we can generate the graph.

```
## generate graph object
g <- graph_from_data_frame(links_fu, directed=F)

## set node colours
V(g)$color <- '#cccccc'

## node shape to indicate query genes
V(g)$shape <- ifelse(names(V(g)) %in% c('RNF43'), 'square', 'circle')
```

```
## make query nodes larger
V(g)$size <- ifelse(names(V(g)) %in% c('RNF43'), 25, 15)

## edge with according to pi score, colour by interaction type
E(g)$width <- abs(E(g)$pi) * 5
E(g)$color <- ifelse(E(g)$pi < 0, '#2B8CBE', '#FED98E')
```

Finally we plot the graph object to the canvas.

```
plot(g, vertex.label.degree = -pi/2,
     vertex.label.family = 'Helvetica',
     vertex.label.color = 'black')
```

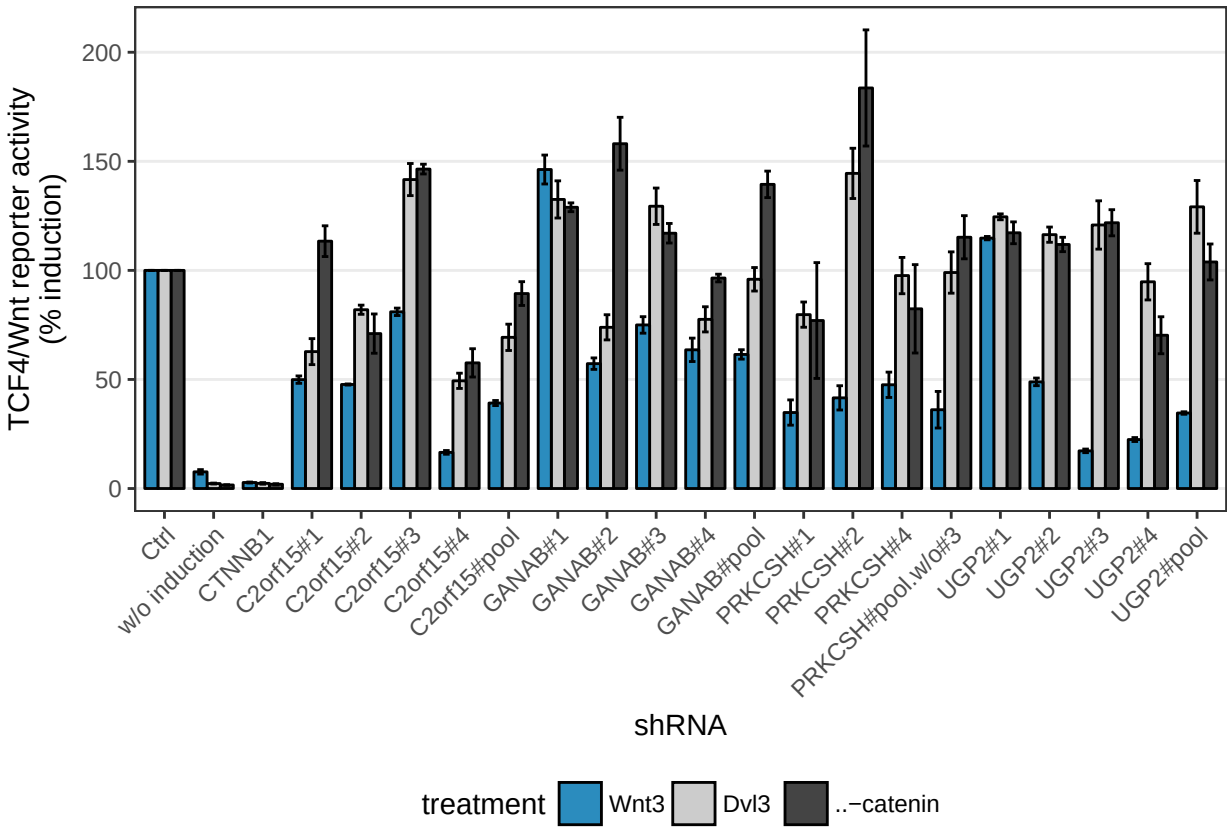


4.2 Wnt activity essays

We load the experimental data and visualize them as a bar plot.

```
data('wnt_data', package='CGIMhd17')

wnt_data %>% mutate(symbol = ifelse(grepl('w/o', shrna) & (!grepl('PRKCSH', shrna)),
                                    'w/o induction', symbol),
                    shrna = ifelse(symbol == 'w/o induction', 'w/o induction', shrna),
                    shrna = relevel(as.factor(shrna), 'w/o induction'),
                    treatment = relevel(as.factor(treatment), 'Wnt3')) %>%
  filter(!(duplicated(induction) & induction != 100),
         !(shrna == 'CTNNB1' & symbol != 'TLR3'),
```

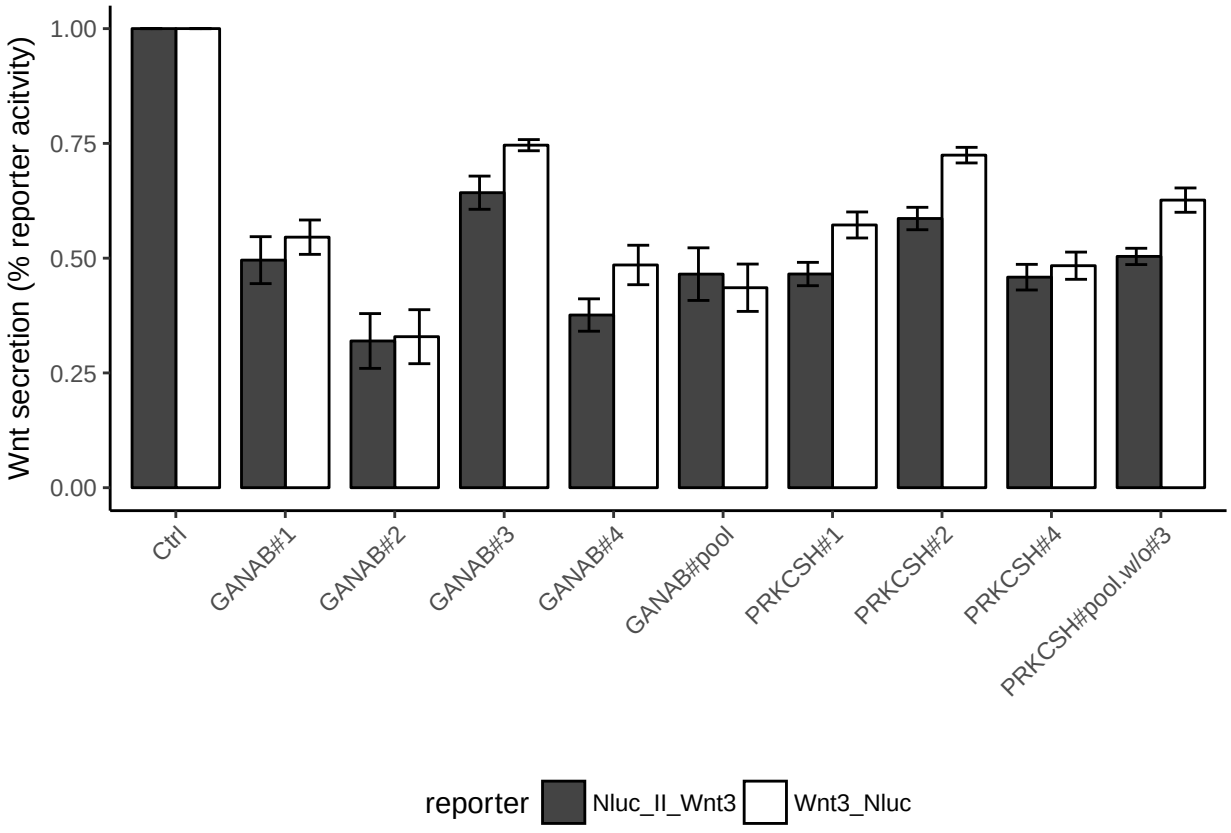
```
       !(shrna == 'Ctrl' & symbol != 'TLR3')) %>%
## means and error bars
group_by(treatment, shrna, symbol) %>%
summarise(mean_ind = mean(induction, na.rm=T),
          ymax = mean_ind + (sd(induction, na.rm=T)/n()),
          ymin = mean_ind - (sd(induction, na.rm=T)/n())) %>% ungroup() %>%
filter(!grepl('TLR3', shrna)) %>%
ggplot(aes(relevel(as.factor(shrna), 'CTNNB1') %>% relevel('w/o induction') %>% relevel('Ctrl'), mean_in
## bar plot
geom_bar(stat='identity', position='dodge', colour='black', width=0.8) +
## error bars
geom_errorbar(aes(ymin=ymin, ymax=ymax),
              position = position_dodge(0.8), width=0.4) +
theme_bw() + xlab('shRNA') + ylab('TCF4/Wnt reporter activity\n(% induction)') +
## colours
scale_fill_manual(values=c('#2B8CBE', '#cccccc', '#444444')) +
## legend and text styles
theme(legend.position = 'bottom', axis.text.x = element_text(angle = 45, hjust=1),
      panel.grid.minor=element_blank(), panel.grid.major.x = element_blank())
```



4.3 Wnt secretion assays

Similar to the Wnt activity essays we again visualize the experimental data as a bar plot.

```
data('w3_sec', package='CGIMhd17')

w3_sec %>% group_by(reporter, shrna) %>%
  summarise(mean_sec = mean(secretion),
            ymax = mean_sec + (sd(secretion)/n()),
            ymin = mean_sec - (sd(secretion)/n())) %>%
  ggplot(aes(shrna, mean_sec, fill=reporter)) +
    geom_bar(stat='identity', position='dodge', colour='black', width=0.8) +
    geom_errorbar(aes(ymin=ymin, ymax=ymax), position = position_dodge(0.9), width=0.4) +
    theme_classic() + theme(legend.position = 'bottom') +
    scale_fill_manual(values=c('#444444', '#ffffff')) +
    theme(axis.text.x = element_text(angle=45, hjust=1)) +
    xlab('') + ylab('Wnt secretion (% reporter acitvity)')
```



5 Benchmarking of genetic correlation networks using protein complex data

In order to judge the performance of estimating true biological relationships using correlations between interaction profiles we examine if there is power to correctly identify co-members of protein complexes. We first load protein complex data contained in the CORUM database (Ruepp et al. 2010). We then run a benchmarking analysis using several parameter combinations.

5.1 Loading the protein complex data

```
data('corum_hs', package='CGIMhd17')
```

5.2 Generating a correlation profile

Based on our interaction data we will generate an interaction table binarizing pairwise interactions. We will then correlate the resulting profiles to achieve gene similarity measures.

```
## returns a filtered interaction matrix
filter_interactions <- function(df, pi_co, pval_co=1){
   interaction_mat <- df %>%
    filter(abs(pi) > pi_co, p.value < pval_co) %>%
    dplyr::select(target, query, pi) %>% spread(query, pi) %>%
    data.frame() %>% `rownames<-`(.$target) %>% dplyr::select(-target)
  return(interaction_mat)
}
```

To generate pairwise correlations of interaction vectors between all pairs of target genes we use the Hmisc package as it provides, in addtion to the correlation values themselves, a matrix of p-values and the number of gene pairs each correlation is based upon. As our interaction matrix is somewhat sparse this can help removing artefacts. We will for purposes of validation only correlate genes that are members of the CORUM protein complex data set.

```
extract_corum <- function(int_mat, n_min=10){
  ## select corum genes
  int_mat_corum <- int_mat[rownames(int_mat) %in% unique(corum_hs$data),]
  ## generate correlation
  cor_results <- rcorr(as.matrix(t(int_mat_corum)), type='spearman')

  ## get correlations and p.values
  cor_pval_mat <- cor_results$P
  cor_pval_mat[cor_results$n <= n_min] <- NA
  cor_corr_mat <- cor_results$r
  cor_corr_mat[cor_results$n <= n_min] <- NA
  ## set upper triangle of p values to NA
  cor_pval_mat[upper.tri(cor_pval_mat)] <- NA
  cor_corr_mat[upper.tri(cor_pval_mat)] <- NA

  cor_pval_df <- cor_pval_mat %>% melt() %>% tbl_df %>%
    filter(!is.na(value), !is.nan(value)) %>% `colnames<-`(c('source', 'target', 'pval')) %>%
    inner_join(cor_corr_mat %>% melt() %>% tbl_df %>%
    filter(!is.na(value), !is.nan(value)) %>% `colnames<-`(c('source', 'target', 'corr'))) %>%
    mutate(padj=p.adjust(pval, method='bonferroni'))

  ## number of genes left
  gene_associations <- cor_pval_df %>% filter(padj < 0.05)
  unique(c(as.character(gene_associations$source),
           as.character(gene_associations$target))) %>% length %>% print()

  return(cor_pval_df)
}
```

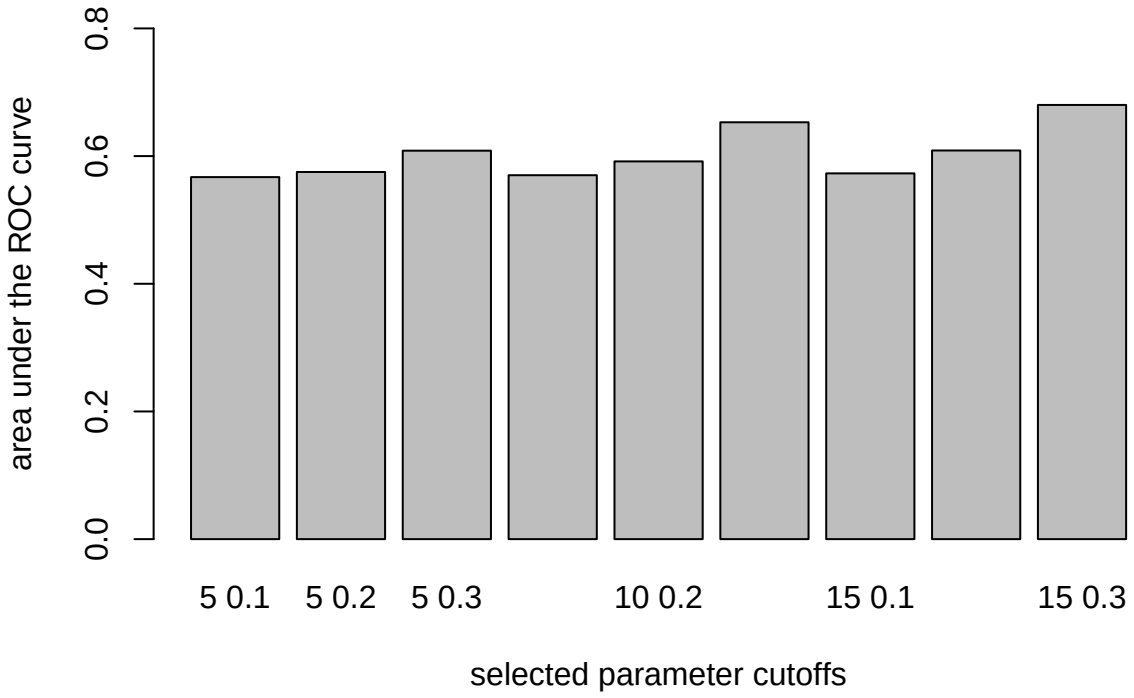
5.3 Perform ROC analysis

We merge together the matrix of p values with protein complex data and use this information as labels for a ROC analysis (Sing et al. 2005), where 0 means that the complex/protein interaction is in CORUM and 0 means that it isn't.

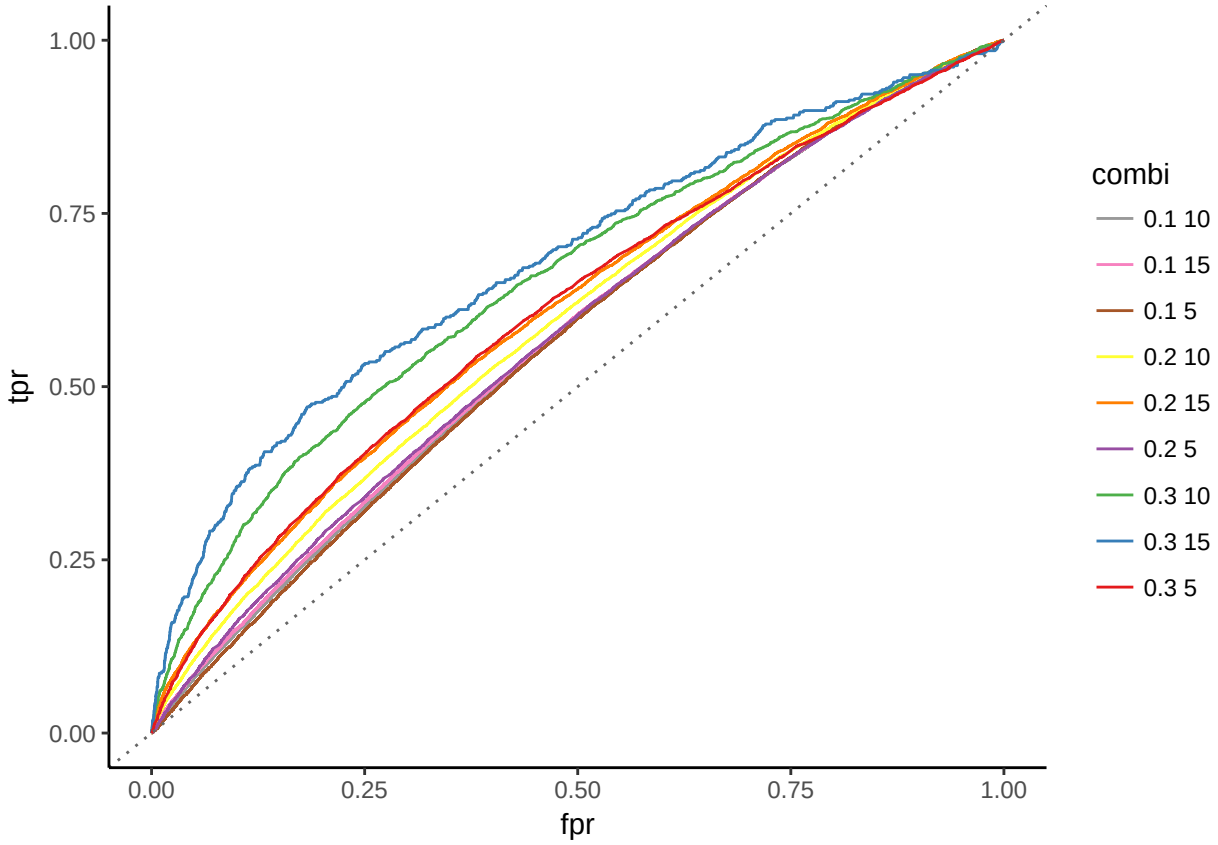
```
## generate data frame of pairs from corum data
corum_pairs <- corum_hs %>% dplyr::select(-id) %>% nest(data) %>%
  mutate(pairs= map(data, function(x) expand.grid(x$data, x$data) %>% tbl_df %>%
                      `colnames<-`(c('source', 'target')))) %>%
  dplyr::select(-data) %>% unnest(pairs) %>%
  dplyr::select(-complex_name) %>% filter(source != target)
```

Now we have a data frame that we can use to perform a ROC analysis.

```
roc_results <- lapply(c(5, 10, 15), function(n_min){
  lapply(seq(0.1, 0.3, by=0.1), function(pi_co){
    cor_pval_df <- filter(all_interactions, !query %in% redundant_queries) %>%
      filter_interactions(pi_co, 1) %>% extract_corum(n_min)
    ## merge to correlation data
    labeled_cors <- cor_pval_df %>% left_join(corum_pairs %>% mutate(pc=1)) %>%
      mutate(pc=ifelse(is.na(pc), 0, pc)) %>% distinct()

    ## with real labels
    pred <- prediction(labeled_cors$corr, labels=labeled_cors$pc)

    perf <- performance(pred, measure='tpr', x.measure='fpr')
    perf_auc <- performance(pred, measure='auc')

    ## fisher test for enrichment using padj 0.2
    ft <- fisher.test(labeled_cors %>% mutate(class = ifelse(corr > 0 & padj < 0.2, 1, 0)) %>%
                        dplyr::select(pc, class) %>% table())$p.value

    ## return ROC results
    return(list(perf_tpr = perf, auc = perf_auc,
                n_pairs = nrow(labeled_cors), pval = ft))
  })
}) %>% unlist(recursive=F)

barplot(roc_results %>% sapply(function(x)x$auc@y.values[[1]]),
        names.arg = paste(c(5,5,5,10,10,10,15,15,15), seq(0.1, 0.3, by=0.1)),
        xlab='selected parameter cutoffs', ylab='area under the ROC curve',
        ylim=c(0,0.8))
```




```
## generate data for roc curve plotting
labels_p <-  c('0.1 5', '0.2 5', '0.3 5', '0.1 10', '0.2 10',
               '0.3 10', '0.1 15', '0.2 15', '0.3 15')
curve_data <- lapply(1:9, function(i){
  tibble(fpr=roc_results[[i]]$perf_tpr@x.values[[1]],
         tpr=roc_results[[i]]$perf_tpr@y.values[[1]],
         combi=labels_p[i]) %>% return()
}) %>% bind_rows()

## plot
curve_data %>%
  ggplot(aes(fpr, tpr, group=combi)) +
  geom_line(aes(colour=combi)) +
  geom_abline(slope=1, linetype='dotted', colour='#666666') +
  theme_classic() +
  scale_colour_manual(values = rev(c('#e41a1c','#377eb8','#4daf4a','#984ea3','#ff7f00',
                                     '#ffff33','#a65628','#f781bf','#999999')))
```



We see that the interaction data hold some power to predict protein complex co-membership.

5.4 Some examples

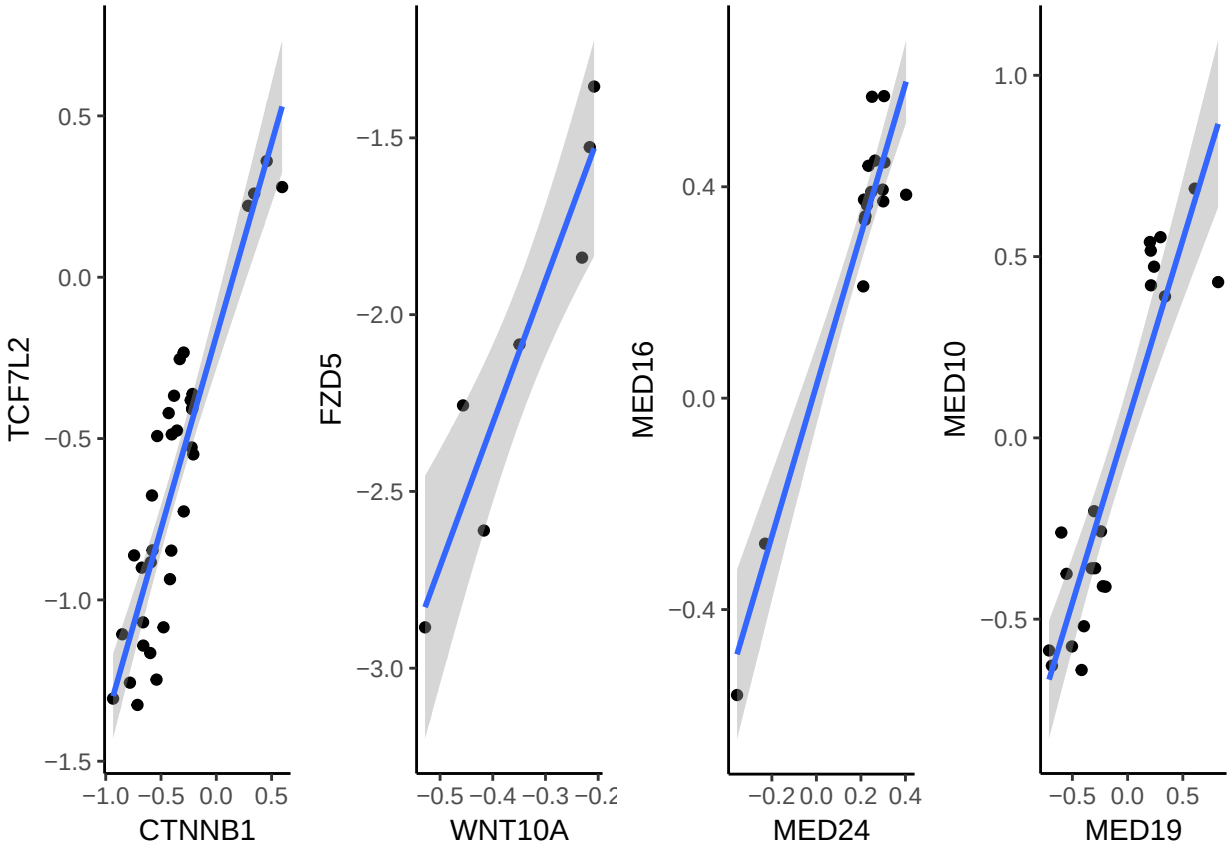
We show some examples where interaction profiles of protein complex members correlate.

5.4.1 Mediator complex

```
plot_correlation <- function(gene1, gene2, co){
  d <- all_interactions %>% filter(target %in% c(gene1, gene2),
                                   !query %in% redundant_queries) %>%
    filter(abs(pi) > co) %>% dplyr::select(target, query, pi) %>%
    spread(target, pi) %>% drop_na() %>%
    `colnames<-`(c('query', 'g2', 'g1'))
  p <- ggplot(d, aes(x=g1, y=g2)) + geom_point() + theme_classic() +
    geom_smooth(method='lm') +
    xlab(gene1) + ylab(gene2)
  cspear <- cor(d$g2, d$g1, method='spearman')
  return(list(p = p, cor = cspear))
}

med1 <- plot_correlation('MED24', 'MED16', 0.2)
med2 <- plot_correlation('MED19', 'MED10', 0.2)
```

5.4.2 Wnt/beta-catenin pathway

```
wnt1 <- plot_correlation('CTNNB1', 'TCF7L2', 0.2)
wnt2 <- plot_correlation('WNT10A', 'FZD5', 0.2)
multiplot(wnt1$p, wnt2$p, med1$p, med2$p, cols=4)
```



6 Towards a genetic interaction map of cancer cells

Based on our previously determined thresholds we generate an interaction table of π-scores. We then correlate the resulting profiles to achieve gene similarity measures.

```
interaction_mat <- all_interactions %>% filter(abs(pi) > 0.2) %>%
  # remove redundant query genes
  filter(! query %in% redundant_queries) %>%
  dplyr::select(target, query, pi) %>% spread(query, pi) %>%
  data.frame() %>% `rownames<-`(.$target) %>% dplyr::select(-target)
```

To generate pairwise correlations of interaction vectors between all pairs of target genes we use the Hmisc package as it provides, in addtion to the correlation values themselves, a matrix of p-values and the number of gene pairs each correlation is based upon. As our interaction matrix is somewhat sparse this can help removing artefacts. We select all correlations with a Bonferroni-corrected p-value of less than 0.5 and use these for visualization in Cytoscape (Shannon et al. 2003).

```
## generate correlation
cor_results <- rcorr(as.matrix(t(interaction_mat)), type='spearman')
```

```
## get correlations and p.values
cor_pval_mat <- cor_results$P
cor_pval_mat[cor_results$n <= 15] <- NA
cor_corr_mat <- cor_results$r
cor_corr_mat[cor_results$n <= 15] <- NA
## set upper triangle of p values to NA
cor_pval_mat[upper.tri(cor_pval_mat)] <- NA
cor_corr_mat[upper.tri(cor_pval_mat)] <- NA

cor_pval_df <- cor_pval_mat %>% melt() %>% tbl_df %>%
  filter(!is.na(value), !is.nan(value)) %>% `colnames<-`(c('source', 'target', 'pval')) %>%
  inner_join(cor_corr_mat %>% melt() %>% tbl_df %>%
  filter(!is.na(value), !is.nan(value)) %>% `colnames<-`(c('source', 'target', 'corr'))) %>%
  mutate(padj=p.adjust(pval, method='bonferroni'),
         fdr = p.adjust(pval, method='BH'))

## number of genes left
gene_associations <- cor_pval_df %>% filter(corr > 0, padj < 0.5)
unique(c(as.character(gene_associations$source),
         as.character(gene_associations$target))) %>% length

write_tsv(gene_associations, 'gene_associations.tsv')
```

In order to test if the enrichments are better than what we would get by chance we permute the interaction data set by shuffling the target vector.

```
set.seed(1231)
gene_associations %>% mutate(target=sample(target, length(target))) %>%
  write_tsv('gene_associations_1231.tsv')
set.seed(1232)
gene_associations %>% mutate(target=sample(target, length(target))) %>%
  write_tsv('gene_associations_1232.tsv')
set.seed(1233)
gene_associations %>% mutate(target=sample(target, length(target))) %>%
  write_tsv('gene_associations_1233.tsv')
```

7 Session info

```
sessionInfo()
## R version 3.4.1 (2017-06-30)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Sierra 10.12.6
##
## Matrix products: default
## BLAS: /System/Library/Frameworks/Accelerate.framework/Versions/A/Frameworks/vecLib.framework/Versions/A
## LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
```

```
##  [1] grid      stats4    parallel  stats     graphics  grDevices utils
##  [8] datasets  methods   base
##
## other attached packages:
##  [1] hexbin_1.27.1         bindrcpp_0.2          CGIMhd17_0.1.0
##  [4] Hmisc_4.0-3           Formula_1.2-2         survival_2.41-3
##  [7] igraph_1.1.2          IHW_1.4.0             reshape2_1.4.3
## [10] HD2013SGI_1.16.0      EBImage_4.18.3        LSD_3.0
## [13] vcd_1.4-4             limma_3.32.10         splots_1.42.0
## [16] geneplotter_1.54.0    annotate_1.54.0       XML_3.98-1.9
## [19] AnnotationDbi_1.38.2  IRanges_2.10.5        S4Vectors_0.14.7
## [22] Biobase_2.36.2        BiocGenerics_0.22.1   RColorBrewer_1.1-2
## [25] caret_6.0-78          lattice_0.20-35       ggsignif_0.4.0
## [28] sfsmisc_1.1-1         preprocessCore_1.38.1 MASS_7.3-47
## [31] pheatmap_1.0.8        broom_0.4.3           ROCR_1.0-7
## [34] gplots_3.0.1          forcats_0.2.0         stringr_1.2.0
## [37] dplyr_0.7.4           purrr_0.2.4           readr_1.1.1
## [40] tidyr_0.7.2           tibble_1.3.4          ggplot2_2.2.1.9000
## [43] tidyverse_1.2.1       BiocStyle_2.4.1
##
## loaded via a namespace (and not attached):
##   [1] readxl_1.0.0        backports_1.1.2     plyr_1.8.4
##   [4] lazyeval_0.2.1      splines_3.4.1       lpsymphony_1.4.1
##   [7] digest_0.6.13       foreach_1.4.4       htmltools_0.3.6
##  [10] tiff_0.1-5          gdata_2.18.0        checkmate_1.8.5
##  [13] magrittr_1.5        memoise_1.1.0       cluster_2.0.6
##  [16] recipes_0.1.1       modelr_0.1.1        gower_0.1.2
##  [19] dimRed_0.1.0        jpeg_0.1-8          colorspace_1.3-2
##  [22] blob_1.1.0          rvest_0.3.2         haven_1.1.0
##  [25] crayon_1.3.4        RCurl_1.95-4.8      jsonlite_1.5
##  [28] bindr_0.1           zoo_1.8-0           iterators_1.0.9
##  [31] glue_1.2.0          DRR_0.0.2           gtable_0.2.0
##  [34] ipred_0.9-6         kernlab_0.9-25      ddalpha_1.3.1
##  [37] DEoptimR_1.0-8      abind_1.4-5         scales_0.5.0.9000
##  [40] DBI_0.7             Rcpp_0.12.14        htmlTable_1.11.0
##  [43] xtable_1.8-2        foreign_0.8-69      bit_1.1-12
##  [46] lava_1.5.1          prodlim_1.6.1       htmlwidgets_0.9
##  [49] httr_1.3.1          acepack_1.4.1       pkgconfig_2.0.1
##  [52] nnet_7.3-12         locfit_1.5-9.1      labeling_0.3
##  [55] tidyselect_0.2.3    rlang_0.1.4         munsell_0.4.3
##  [58] cellranger_1.1.0    tools_3.4.1         cli_1.0.0
##  [61] RSQLite_2.0         devtools_1.13.4     fdrtool_1.2.15
##  [64] evaluate_0.10.1     fftwtools_0.9-8     yaml_2.1.16
##  [67] ModelMetrics_1.1.0  knitr_1.17          bit64_0.9-7
##  [70] robustbase_0.92-8   caTools_1.17.1      nlme_3.1-131
##  [73] slam_0.1-40         RcppRoll_0.2.2      xml2_1.1.1
##  [76] compiler_3.4.1      rstudioapi_0.7      png_0.1-7
##  [79] stringi_1.1.6       Matrix_1.2-12       psych_1.7.8
##  [82] lmtest_0.9-35       data.table_1.10.4-3 bitops_1.0-6
##  [85] latticeExtra_0.6-28 R6_2.2.2            gridExtra_2.3
##  [88] KernSmooth_2.23-15  codetools_0.2-15    gtools_3.5.0
##  [91] assertthat_0.2.0    CVST_0.2-1          rprojroot_1.2
##  [94] withr_2.1.0.9000    mnormt_1.5-5        hms_0.4.0
```

```
##  [97] rpart_4.1-11        timeDate_3042.101   class_7.3-14
## [100] rmarkdown_1.8       base64enc_0.1-3     lubridate_1.7.1
```

Bibliography

Aguirre, Andrew J, Robin M Meyers, Barbara A Weir, Francisca Vazquez, Cheng-Zhong Zhang, Uri Ben-David, April Cook, et al. 2016. "Genomic Copy Number Dictates a Gene-Independent Cell Response to CRISPR/Cas9 Targeting." *Cancer Discovery* 6 (8): 914–29. doi:10.1158/2159-8290.CD-16-0154.

Barretina, Jordi, Giordano Caponigro, Nicolas Stransky, Kavitha Venkatesan, Adam A Margolin, Sungjoon Kim, Christopher J Wilson, et al. 2012. "The Cancer Cell Line Encyclopedia enables predictive modelling of anticancer drug sensitivity." *Nature* 483 (7391): 603–7. doi:10.1038/nature11003.

Forbes, Simon A, David Beare, Harry Boutselakis, Sally Bamford, Nidhi Bindal, John Tate, Charlotte G Cole, et al. 2017. "COSMIC: somatic cancer genetics at high-resolution." *Nucleic Acids Research* 45 (D1): D777–D783. doi:10.1093/nar/gkw1121.

Hart, Traver, Amy Hin Yan Tong, Katie Chan, Jolanda Van Leeuwen, Ashwin Seetharaman, Michael Aregger, Megha Chandrashekhar, et al. 2017. "Evaluation and Design of Genome-Wide CRISPR/SpCas9 Knockout Screens." *G3 (Bethesda, Md.)* 7 (8). G3: Genes, Genomes, Genetics: 2719–27. doi:10.1534/g3.117.041277.

Ignatiadis, Nikolaos, Bernd Klaus, Judith B Zaugg, and Wolfgang Huber. 2016. "Data-driven hypothesis weighting increases detection power in genome-scale multiple testing." *Nature Methods* 13 (7). Europe PMC Funders: 577–80. doi:10.1038/nmeth.3885.

Kamburov, Atanas, Ulrich Stelzl, Hans Lehrach, and Ralf Herwig. 2013. "The ConsensusPathDB interaction database: 2013 update." *Nucleic Acids Research* 41 (D1): D793–D800. doi:10.1093/nar/gks1055.

Klijn, Christiaan, Steffen Durinck, Eric W Stawiski, Peter M Haverty, Zhaoshi Jiang, Hanbin Liu, Jeremiah Degenhardt, et al. 2014. "A comprehensive transcriptional portrait of human cancer cell lines." *Nature Biotechnology* 33 (3). Nature Research: 306–12. doi:10.1038/nbt.3080.

Kuleshov, Maxim V., Matthew R. Jones, Andrew D. Rouillard, Nicolas F. Fernandez, Qiaonan Duan, Zichen Wang, Simon Koplev, et al. 2016. "Enrichr: a comprehensive gene set enrichment analysis web server 2016 update." *Nucleic Acids Research* 44 (W1): W90–W97. doi:10.1093/nar/gkw377.

Laufer, Christina, Bernd Fischer, Maximilian Billmann, Wolfgang Huber, and Michael Boutros. 2013. "Mapping genetic interactions in human cancer cells with RNAi and multiparametric phenotyping." *Nature Methods* 10 (5): 427–31. doi:10.1038/nmeth.2436.

Munoz, Diana M, Pamela J Cassiani, Li Li, Eric Billy, Joshua M Korn, Michael D Jones, Javad Golji, et al. 2016. "CRISPR Screens Provide a Comprehensive Assessment of Cancer Vulnerabilities but Generate False-Positive Hits for Highly Amplified Genomic Regions." *Cancer Discovery* 6 (8): 900–913. doi:10.1158/2159-8290.CD-16-0178.

Rauscher, Benedikt, Florian Heigwer, Marco Breinig, Jan Winter, and Michael Boutros. 2017. "Genome-CRISPR - a database for high-throughput CRISPR/Cas9 screens." *Nucleic Acids Research* 45 (D1): D679–D686. doi:10.1093/nar/gkw997.

Ruepp, Andreas, Brigitte Waegele, Martin Lechner, Barbara Brauner, Irmtraud Dunger-Kaltenbach, Gisela Fobo, Goar Frishman, Corinna Montrone, and H.-Werner Mewes. 2010. "CORUM: the comprehensive resource of mammalian protein complexes—2009." *Nucleic Acids Research* 38 (suppl_1). Oxford University Press: D497–D501. doi:10.1093/nar/gkp914.

Shannon, Paul, Andrew Markiel, Owen Ozier, Nitin S Baliga, Jonathan T Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. 2003. "Cytoscape: a software environment for integrated models of biomolecular interaction networks." *Genome Research* 13 (11): 2498–2504. doi:10.1101/gr.1239303.

Sing, Tobias, Oliver Sander, Niko Beerenwinkel, and Thomas Lengauer. 2005. "ROCR: visualizing classifier performance

in R." *Bioinformatics (Oxford, England)* 21 (20): 3940–1. doi:10.1093/bioinformatics/bti623.

Steinhart, Zachary, Zvezdan Pavlovic, Megha Chandrashekhar, Traver Hart, Xiaowei Wang, Xiaoyu Zhang, Mélanie Robitaille, et al. 2017. "Genome-wide CRISPR screens reveal a Wnt-FZD5 signaling circuit as a druggable vulnerability of RNF43-mutant pancreatic tumors." *Nature Medicine* 23 (1): 60–68. doi:10.1038/nm.4219.

Wang, Tim, Haiyan Yu, Nicholas W Hughes, Bingxu Liu, Arek Kendirli, Klara Klein, Walter W Chen, Eric S Lander, and David M Sabatini. 2017. "Gene Essentiality Profiling Reveals Gene Networks and Synthetic Lethal Interactions with Oncogenic Ras." *Cell* 168 (5): 890–903.e15. doi:10.1016/j.cell.2017.01.013.