Wang L et al, 2017

Damiano Fantini

July 5, 2017

Introduction

This document describes the code used for generating some of the plots included in the paper *Lu et al (2017?)*. The paper includes a set of analyses of **TCGA** (*The Cancer Genome Atlas*) data that were retrieved from cBioportal (www.cbioportal.org/) via the *TCGAretriever* package. The core functions used for the analyses are available online. These require several R libraries from CRAN or bioconductor to be installed: *TCGAretriever*, *survival*, *gplots*, *ggplot2*, *OrganismDbi*, *GO.db*, *BSgenome.Hsapiens.UCSC.hg19*, *org.Hs.eg.db*, *Homo.sapiens*.

To get started, run the following line of code.

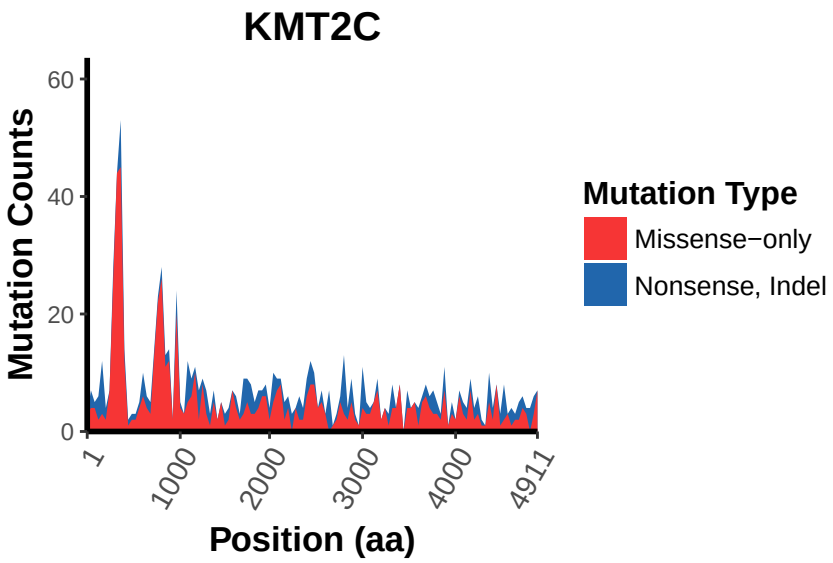
```
base::source("http://www.labwizards.com/rlib/tcgaTools.R")
```

Question 1: Detect and Visualize Hotspot Mutations

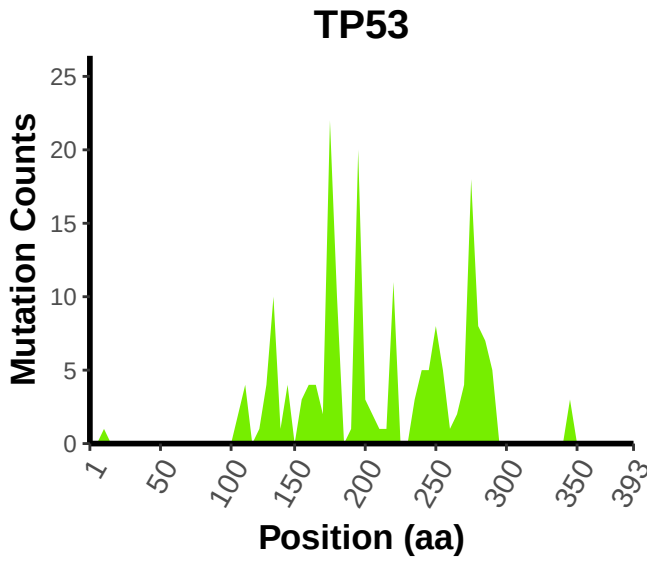
Here we will discuss about how to identify and visualize hotspot mutation sites in human cancer on a certain gene of interest. The analysis can be run on a specific type of cancer or on all TCGA provisional datasets, by setting the *study.id* argument in the *get_hotspot_mutations()* function. The identifier used to select a specific gene is the *OFFICIAL SYMBOL* of that gene. Two examples are provided.

```
# Gene: KMT2C; all TCGA datasets; automatic binning.
my.gene <- "KMT2C"
cur.mutations <- get_hotspot_mutations(gene.symbol = my.gene)
cur.mut.binned <- mutations_along_seq(mutation.counts = cur.mutations,
                                      bin = -1)
hotspot.plot <- plot_binned_mutcounts(mut.binned = cur.mut.binned)
```

```
hotspot.plot +
  theme(plot.title = element_text(hjust = 0.5))
```



```
# Gene: TP53; only brca_tcga; binning: 5 aa;
# only missense mutations; custom color.
new.gene = "TP53"
brca.tp53.mut <- get_hotspot_mutations(gene.symbol = new.gene,
                                       study.id = "brca_tcga")
new.mut.binned <- mutations_along_seq(mutation.counts = brca.tp53.mut,
                                      bin = 5)
hotspot.plot <- plot_binned_mutcounts(mut.binned = new.mut.binned,
                                      mut.type = "missense",
                                      colors = c(NA, "chartreuse2"))
```

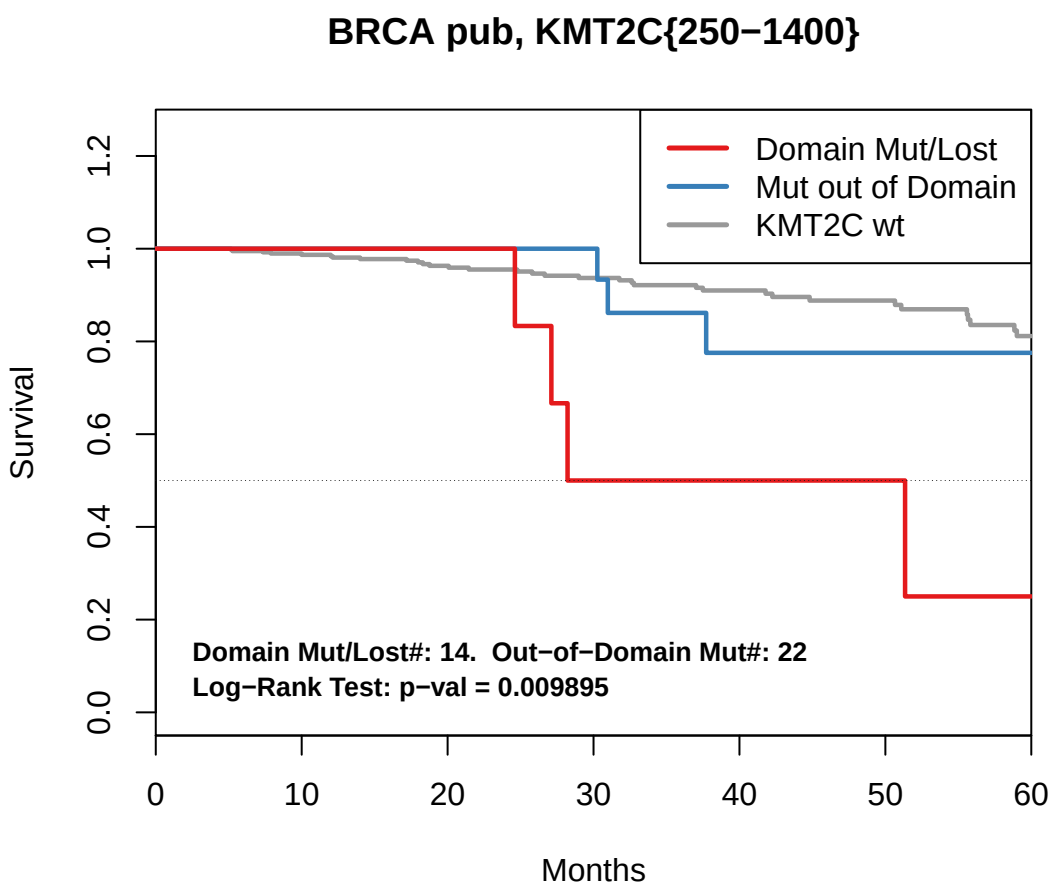


Question 2: Detect Mutations in a specific aa range and explore patient survival

The following code is aimed at exploring survival in patients carrying specific mutations. Patients will be segmented based on the mutation status of one or two genes. Also two types of patient segmentation can be performed by setting the *method* argument of the *call_mutcases_by_range()* function. Briefly, *method="missense.only"* enables searching for missense mutations located in the amino acid range of interest (as defined by the *window.min* and *window.max* arguments). On the contrary, *method="range.alive"* means that all mutations disrupting the domain of interest (for example, an upstream nonsense mutation) will be compared against mutations not affecting that domain (for example, an upstream missense mutation or a downstream nonsense mutation).
call_mutcases_by_range will segment patients in 3 groups: i) wild type patients, ii) patients with a mutation that does not affect the region of interest, and iii) patients with a mutation affecting the domain of interest. Survival analyses and the Log-Rank tests are performed using the functions in the *survival* library. Here we will cover three examples.

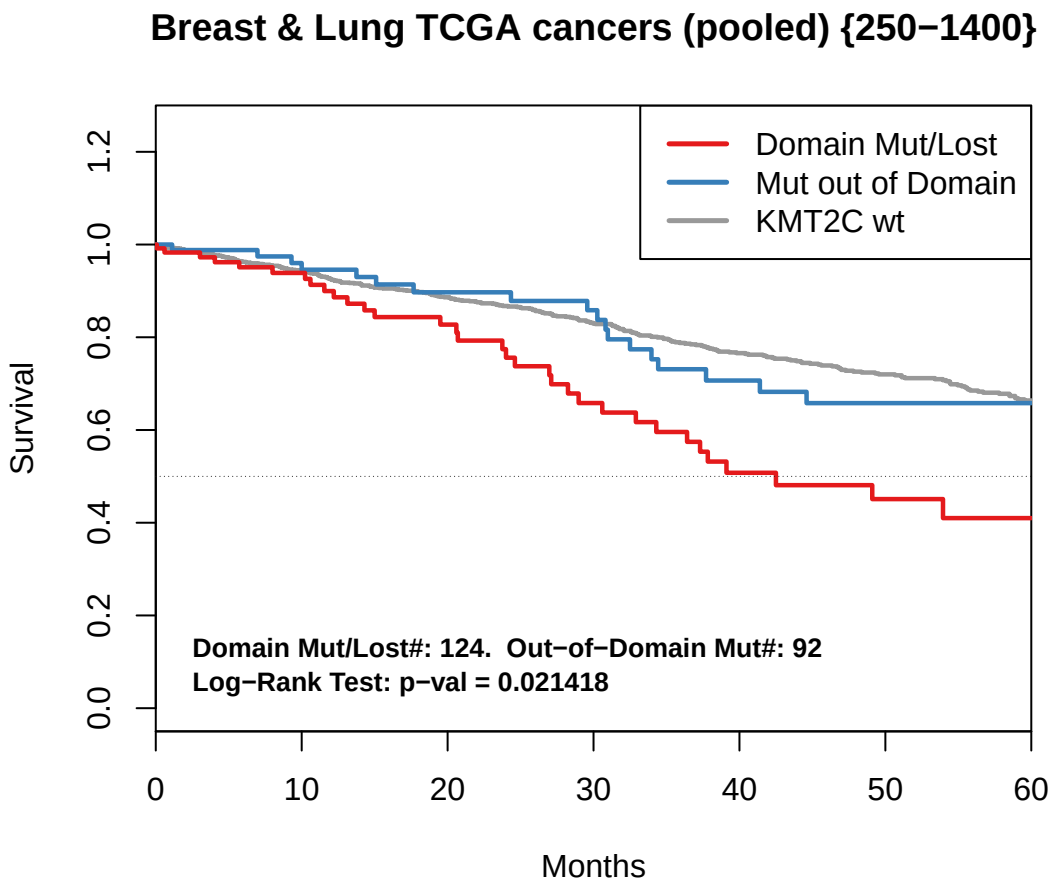
```
# Survival of patients with mutation in PHD1,2 Domains of MLL3
# Dataset, Breast cancer pub (TCGA)
mll3.surv <- call_mutcases_by_range(gene.symbol = "KMT2C",
                                    cancer.study = "brca_tcga_pub",
                                    data.type = "tcga.id",
                                    clinic.data = NULL,
                                    window.min = 250,
                                    window.max = 1400,
                                    method = "range.alive")
```

```
mll3.surv <- call_survival_by_range (mutcases.list = mll3.surv)
plot_km_custom(mll3.surv,
               xlim = c(0,60),
               ylim = c(0, 1.25),
               main = "BRCA pub, KMT2C{250-1400}")
```




```
# More complex analysis. Analyze survival in two datasets and pool results
# Define datasets of interest (tcga_id), data retrieved via TCGAretriever
tcga.ids = c("brca_tcga", "nsclc_tcga_broad_2016")
label <- "Breast & Lung TCGA cancers (pooled)"
method = "range.alive"
my.gene <- "KMT2C"
my.aarange <- c(250, 1400)
#
# "loop" through TCGA datasets
tcga.patient.survival <- lapply(tcga.ids, (function(csid){
  #
  # retrieve survival of patients with mutation in gene 1 (MLL3)
  gene01.surv <- call_mutcases_by_range(gene.symbol = my.gene,
                                        cancer.study = csid,
                                        data.type = "tcga.id",
                                        clinic.data = NULL,
                                        window.min = my.aarange[1],
                                        window.max = my.aarange[2],
                                        method = method)
  #
```

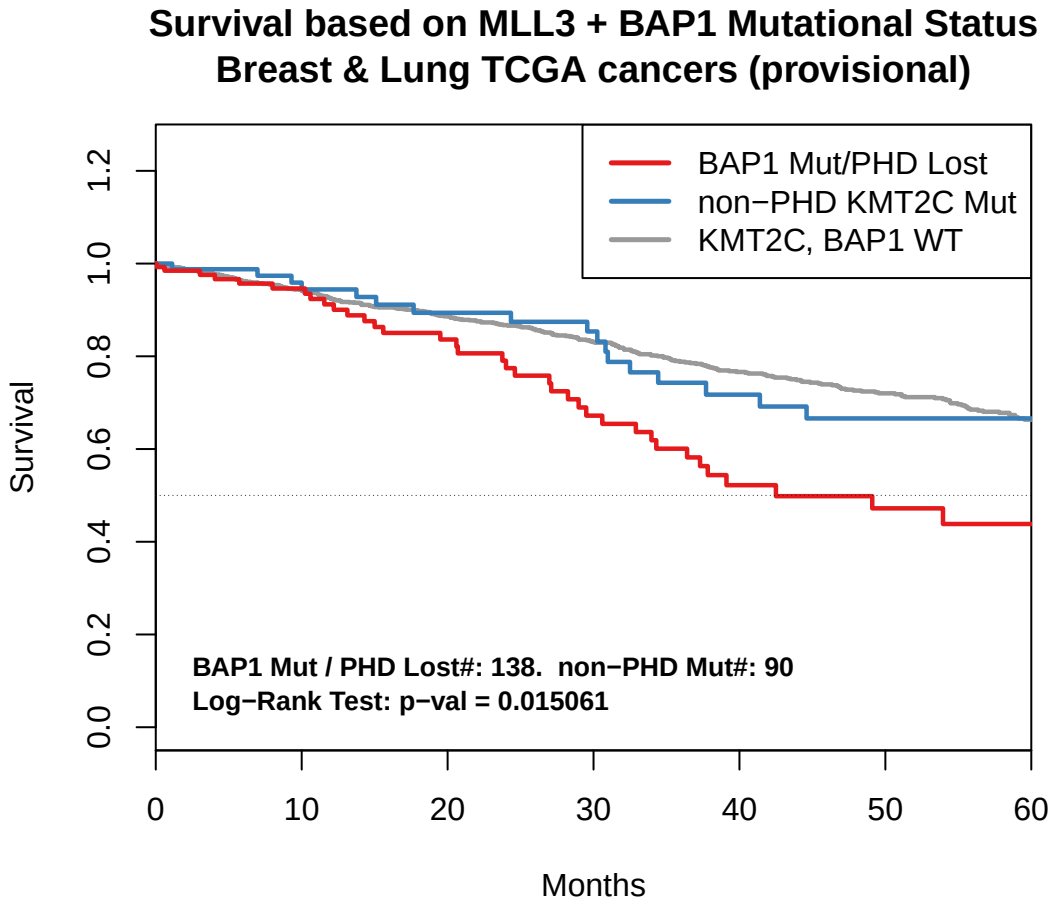
```
  # call survival
  call_survival_by_range (mutcases.list = gene01.surv)
}))
# Merge results in the list, then plot
tcga.patient.survival <- merge_km_data(km.data.list = tcga.patient.survival)
plot_km_custom(tcga.patient.survival,
               xlim = c(0,60),
               ylim = c(0, 1.25),
               main = paste(label, "{250-1400}"))
```




```
# Third and most complex analysis of this kind.
# Analyze survival based on mutation status of 2 genes.
# Also, analyze two or more datasets and pool results.
tcga.ids = c("brca_tcga", "nsclc_tcga_broad_2016")
label <- paste("Survival based on MLL3 + BAP1 Mutational Status",
               "Breast & Lung TCGA cancers (provisional)", sep = "\n")
method = "range.alive"
gene.01 <- "KMT2C"
aarange.01 <- c(250, 1400)
gene.02 <- "BAP1"
aarange.02 <- c(-1, -1) # no range defined, any mutation will count
#
#
# Loop through different datasets
surv.2gene <- lapply(tcga.ids, (function(csid){
  #
```

```
  # retrieve survival of patients with mutation in gene 1 (MLL3)
  gene01.surv <- call_mutcases_by_range(gene.symbol = gene.01,
                                        cancer.study = csid,
                                        data.type = "tcga.id",
                                        clinic.data = NULL,
                                        window.min = aarange.01[1],
                                        window.max = aarange.01[2],
                                        method = method)
  #
  # retrieve survival of patients with mutation in gene 2 (BAP1)
  gene02.surv <- call_mutcases_by_range(gene.symbol = gene.02,
                                        cancer.study = csid,
                                        data.type = "tcga.id",
                                        clinic.data = NULL,
                                        window.min = aarange.02[1],
                                        window.max = aarange.02[2],
                                        method = method)
  #
  # retrieve data relative to gene #2 (BAP1)
  new.IN <- gene02.surv$calls$case.in
  new.OUT <- gene02.surv$calls$case.out
  new.EXCL <- gene02.surv$calls$case.exclude
  #
  # Create a final copy (merged)
  merged.surv <- gene01.surv
  #
  # Add cases
  merged.surv$calls$case.exclude <- unique(c(new.EXCL,
                                             merged.surv$calls$case.exclude))
  merged.surv$calls$case.in <- unique(c(new.IN,
                                        merged.surv$calls$case.in))
  merged.surv$calls$case.out <- unique(c(new.OUT,
                                         merged.surv$calls$case.out))
  #
  # Remove duplicates
  merged.surv$calls$case.in <-
    merged.surv$calls$case.in[!merged.surv$calls$case.in %in%
                              merged.surv$calls$case.exclude]
  merged.surv$calls$case.out <-
    merged.surv$calls$case.out[!merged.surv$calls$case.out %in%
                               c(merged.surv$calls$case.exclude,
                                 merged.surv$calls$case.in)]
  merged.surv$calls$case.bckground <-
    merged.surv$calls$case.bckground[!merged.surv$calls$case.bckground %in%
                                       c(merged.surv$calls$case.exclude,
                                         merged.surv$calls$case.in,
                                         merged.surv$calls$case.out)]
  #
  # Done, call survival and return
  call_survival_by_range(mutcases.list = merged.surv)
}))
#
#
```

```
# Merge data in the list and then plot
surv.2gene <- merge_km_data(km.data.list = surv.2gene)
plot_km_custom(surv.2gene,
               xlim = c(0,60),
               ylim = c(0, 1.25),
               main = label,
               cust.mll3.lab = TRUE)
```



Question 3: Plot a custom Oncoprint for two genes of interest

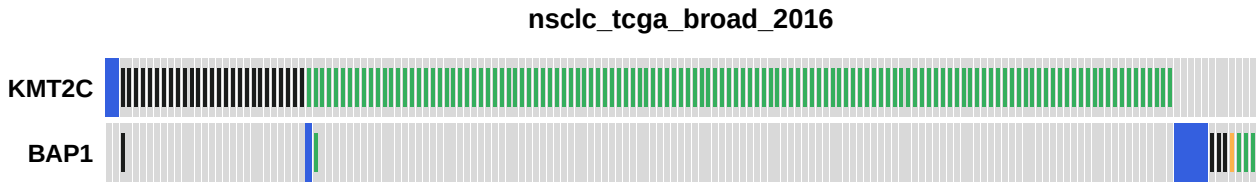
The following code is aimed at building a custom Oncoprints using data from TCGA. Patients carrying mutations that disrupt the domains of interest are counted. Next, a visualization is built using the functions in the *ggplot2* library. Two types of Oncoprints are generated in the following lines of code: i) standard Oncoprint without gene amplification and ii) custom Oncoprint reporting deletions, mutations that disrupt a domain of interest and mutations that leave the domain of interest intact. Contingency tables are returned as an element of the list returned by the *std_2gene_oncoprint()* and the *prep_custom_oncoprint()* functions. These can be used for performing Fisher tests when needed.

```
# non-small-cell lung cancer dataset; KMT2C and BAP1
std.oncoprint.nsclc <- std_2gene_oncoprint(csid = "nsclc_tcga_broad_2016",
                                           tcga.only = T,
                                           gene.01 = "KMT2C",
                                           gene.02 = "BAP1")
#
```

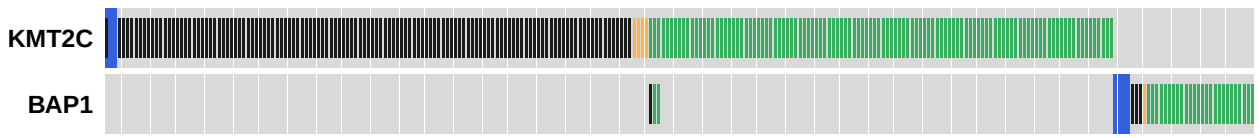
```
cat(std.oncoprint.nsclc$plot.legend)
```

```
## blue box = deep deletion;
## black square = nonsense mutation;
## yellow square = coding INDEL
## green square = missense mutation
```

```
print(std.oncoprint.nsclc$plot)
```




```
# Metabric dataset; KMT2C and BAP1
std.oncoprint.metabric <- std_2gene_oncoprint(csid = "brca_metabric",
                                              gene.01 = "KMT2C",
                                              gene.02 = "BAP1")
print(std.oncoprint.metabric$plot)
```




```
conting.tab <- std.oncoprint.metabric$count.table
conting.tab <- conting.tab / sum(conting.tab)
kmt2c.mutFreq <- sum(conting.tab[2,])
kmt2c.mutFreq
```

```
## [1] 0.1184788
```

```
bap1.mutFreq <- sum(conting.tab[,2])
bap1.mutFreq
```

```
## [1] 0.01803998
```

```
# Expected BAP1 + KMT2C mutations if totally independent
kmt2c.mutFreq <- kmt2c.mutFreq * bap1.mutFreq
kmt2c.mutFreq
```

```
## [1] 0.002137355
```

```
# Observed BAP1 + KMT2C mutation, lower than expected
conting.tab[2,2]
```

```
## [1] 0.001462701
```

```
conting.tab[2,2] < kmt2c.mutFreq
```

```
## [1] TRUE
```

```
# Fisher test
fisher.test(std.oncoprint.metabric$count.table,
            alternative = "less")
```

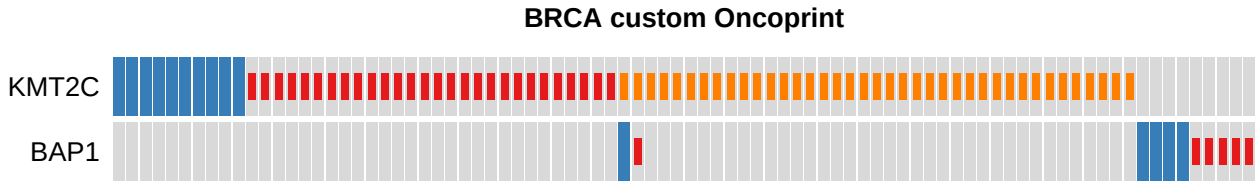
```
##
```

```
##  Fisher's Exact Test for Count Data
##
## data:  std.oncoprint.metabric$count.table
## p-value = 0.3448
## alternative hypothesis: true odds ratio is less than 1
## 95 percent confidence interval:
##  0.000000 1.825345
## sample estimates:
## odds ratio
##  0.6523224
```

```
# Alternative Oncoprint (based on disruption or not of a specific domain)
my.oncoprint <- prep_custom_oncoprint(csid = "brca_tcga",
                                      gene.01 = "KMT2C",
                                      aarange.01 = c(250, 1400),
                                      gene.02 = "BAP1")
oncoplot <- plot_custom_oncoprint(my.oncoprint)
cat(oncoplot$plot.legend)
```

```
## blue box = deep deletion;
## red square = mutation disrupting the domain of interest;
## orange square = mutation not affecting the domain of interest
```

```
print(oncoplot$plot +
        ggtitle("BRCA custom Oncoprint"))
```




```
conting.tab <- my.oncoprint$count.table
conting.tab <- conting.tab[, -2]
conting.tab
```

```
##            BAP1.WT BAP1.LOST
## KMT2C.WT       877         9
## KMT2C.MUT       37         2
## KMT2C.LOST      38         0
```

```
fisher.test(conting.tab,
            alternative = "less")
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  conting.tab
## p-value = 0.1331
## alternative hypothesis: less
```

Done, **success!** This document covered and described the R code used for TCGA data analyses included in Wang L et al, 2017. These lines of code can be used for reproducing the plots included in the publication or adapted for running similar analyses. For questions, email "damiano.fantini@gmail.com"

Appendix 1: Custom functions - Code

The analyses described in this Vignette rely on some published libraries as well as on a set of custom in-house functions that were not published before. The source code of these custom functions is available at the following URL: "http://www.labwizards.com/rlib/tcgaTools.R". The R code of the custom in-house functions is provided below.

```
fetch_and_prepare_from <- function(where = "cbio",
                                   data = "blca_tcga", #data_table
                                   gene.symbol = "TP53",
                                   clin.data = NULL) {
  #A
  list.to.return <- list()
  #
  if (where == "cbio" & is.character(data) & length(data) == 1){
    # We need to retrieve cases and profile definitions first
    seq.case.id <- get_case_lists(data)$case_list_id
    seq.case.id <- grep ("sequenc|mutation", seq.case.id,
                         value = TRUE, ignore.case = TRUE)[1]
    #
    tmp.mut.profi <- get_genetic_profiles(data)$genetic_profile_id
    tmp.mut.profi <- grep("sequenc|mutation", tmp.mut.profi,
                          value = TRUE, ignore.case = TRUE)[1]
    #
    # Now make sure that seq/mutation data are available
    if (is.na(tmp.mut.profi) & is.na(seq.case.id)) {
      stop("No data available")
    }
    #
    # Retrieve clinical data to attach
    clinic.data <- suppressWarnings(get_clinical_data(seq.case.id))
    #
    # First, let's retrieve all cases
    tot.cases <- expand_cases(data)
    case.idx <- which(sapply(1:length(tot.cases), (function(i){
      tot.cases[[i]]$case_list_id == seq.case.id
    })))
    tot.cases <- tot.cases[[case.idx]]$case_id
    #
    # Now let's retrieve mutations
    tmp.mut.dset <- data.frame(get_ext_mutation(case_id = seq.case.id,
                                                gprofile_id = tmp.mut.profi,
                                                glist = gene.symbol),
                               stringsAsFactors = FALSE)
    tmp.mut.dset <- tmp.mut.dset[,c("mutation_type",
                                    "amino_acid_change",
                                    "case_id")]
    tmp.mut.dset <- tmp.mut.dset[grep("^TCGA", tmp.mut.dset$case_id), ]
    #
    case.to.exclude <- c() #place holder for vector including cases to exclude
    #
  } else if (where == "raw_data") {
    if (!(is.data.frame(data) &
          sum(c("SYMBOL", "SAMPLE", "type", "Protein_Change") %in%
```

```
            colnames(data)) == 4 &
          ! is.null(clin.data) )) {
      stop("MAF file or clinic data are not suitable for analysis")
    }
    #
    # First, let's retrieve all cases, assuming there are no samples with 0 mutations
    tot.cases <- unique(data$SAMPLE)
    #
    # Now let's retrieve mutations of interest
    tmp.mut.dset <- data[data$SYMBOL == gene.symbol, ]
    tmp.mut.dset <- tmp.mut.dset[,c("type",
                                    "Protein_Change",
                                    "SAMPLE")]
    tmp.mut.dset <- tmp.mut.dset[- grep("Silent", tmp.mut.dset$type) , ]
    tmp.mut.dset <- tmp.mut.dset[tmp.mut.dset$Protein_Change != "", ]
    tmp.mut.dset$Protein_Change <-
      gsub("\\*{2}", "*", gsub("fs$", "*fs", tmp.mut.dset$Protein_Change))
    #
    case.to.exclude <-
      unique(tmp.mut.dset[grep("Splice", tmp.mut.dset$type) , ]$SAMPLE)
    tmp.mut.dset <- tmp.mut.dset[!tmp.mut.dset$SAMPLE %in% case.to.exclude, ]
    #
    colnames(tmp.mut.dset) <- c("mutation_type", "amino_acid_change", "case_id")
    #
    # Format clinic data
    clinic.data <- data.frame(clin.data, stringsAsFactors = FALSE)
  }
  list.to.return$mutations.dataset <- tmp.mut.dset
  list.to.return$clinic.data <- clinic.data
  list.to.return$tot.cases <- tot.cases
  list.to.return$case.to.exclude <- case.to.exclude
  list.to.return$call <- list()
  list.to.return$call$where = where
  list.to.return$call$gene.symbol = gene.symbol

  if (where == "cbio") {
    list.to.return$call$tcga.ids <- c(seq.case.id, tmp.mut.profi)
    list.to.return$call$csid <- data
  }
  return(list.to.return)
}
#
#
call_mutcases_by_range <- function(gene.symbol,
                                   cancer.study,
                                   clinic.data = NULL,
                                   window.min = -1,
                                   window.max = -1,
                                   data.type = "tcga.id", # c("tcga.id", "raw")
                                   method = "missense.only"
)
{
  #
```

```
# -------------------------- data prep ----------------------
#
if (data.type[1] == "tcga.id") {
  #
  # It is a cancestudy ID; download data from cBio
  cancer.data <- fetch_and_prepare_from(where = "cbio",
                                        data = cancer.study,
                                        gene.symbol = gene.symbol,
                                        clin.data = NULL)
  #
} else if (data.type[1] == "raw")
{
  cancer.data <- fetch_and_prepare_from(where = "raw_data",
                                        data = cancer.study,
                                        gene.symbol = gene.symbol,
                                        clin.data = clinic.data)
  #
} else {
  stop("Unexpected exception!")
}
#
# Retrieve data out of cancer.data object
#
tmp.mut.dset <- cancer.data$mutations.dataset
exluded.cases <- cancer.data$case.to.exclude
tot.cases <- cancer.data$tot.cases
clinic.data <- cancer.data$clinic.data
#
#
supported_methods <- c("missense.only", "range.alive")
if (! (is.character(method) & length(method) == 1 &
       method %in% supported_methods)) {
  stop (paste("Unsupported method.
              The supported methods are:",
              paste(supported_methods, collapse = ", ")))
}
#
# ------------ data are now ready for calling the mutations -------------------
#
#
if (method == "missense.only") {
  ex.idx <- grep("missense", tmp.mut.dset$mutation_type,
                 ignore.case = TRUE, invert = TRUE)
  exluded.cases <- c(exluded.cases, tmp.mut.dset$case_id[ex.idx])
  tmp.mut.dset <- tmp.mut.dset[! tmp.mut.dset$case_id %in% exluded.cases ,]
} else if (method == "range.alive") {
  #
  tmp.mut.dset$amino_acid_change <-
    gsub("(\\*).*$", "*", tmp.mut.dset$amino_acid_change)
  tmp.mut.dset$amino_acid_change <-
    gsub("^.*(splice).*$", "", tmp.mut.dset$amino_acid_change)
  exluded.cases <-
    unique(tmp.mut.dset[tmp.mut.dset$amino_acid_change == "",]$case_id)
```

```
  tmp.mut.dset <-
    tmp.mut.dset[tmp.mut.dset$amino_acid_change != "",]
  #
  tmp.mut.dset$range.status <- sapply(tmp.mut.dset$amino_acid_change,
                                      (function(aach){
    if (window.min < 0 | window.max < 0) {
      "affected"
    } else {
      if(regexpr("\\*", aach) > 0) {
        # it's a nonsense
        if(is.na(as.integer(gsub("[[:alpha:]]|\\*|\\-|\\_", "", aach)))) {
          "exclude"
        } else if(as.integer(gsub("[[:alpha:]]|\\*|\\-|\\_", "", aach)) <
                  (window.max + 1)) {
          "affected"
        } else {
          "OK"
        }
      } else {
        #it's missense
        tmp.pos <- as.integer(gsub("[[:alpha:]]|\\*|\\-|\\_", "", aach))
        if (is.na(tmp.pos)) {
          "exclude"
        } else if(tmp.pos > window.max | tmp.pos < window.min) {
          "OK"
        } else {
          "affected"
        }
      }
    }
  }))
}
#
# Now convert AA substitution to position
tmp.mut.dset$position <-
  suppressWarnings(as.integer(gsub("[[:alpha:]]|\\*|\\.", "",
                                   tmp.mut.dset$amino_acid_change)))
#
# Take note of NA positions, remove them if not present in mutated/missense cases
na.cases <- tmp.mut.dset$case_id[is.na(tmp.mut.dset$position)]
tmp.mut.dset <- tmp.mut.dset[!is.na(tmp.mut.dset$position),]
#
#
# Check this
if (window.min < 0)
  window.min <- min(tmp.mut.dset$position) - 1
if (window.max < 0)
  window.max <- max(tmp.mut.dset$position) + 1
#
#
# Which cases have mutations in frame?
tmp.mut.dset$inside <- tmp.mut.dset$position > window.min &
  tmp.mut.dset$position < window.max
```

```
unique.cases <- unique(tmp.mut.dset$case_id)
#
if (method == "missense.only") {
  case.report <- sapply(unique.cases, (function(ucs){
    slice <- tmp.mut.dset[tmp.mut.dset$case_id == ucs,]
    slice.ratio <- sum(slice$inside == TRUE) / nrow(slice)
    if (slice.ratio == 1) {
      "in"
    } else if (slice.ratio == 0){
      "out"
    } else {
      NA
    }
  }))
  exluded.cases <-
    unique(c(exluded.cases, names(case.report[is.na(case.report)])))
  case.report <- case.report[!is.na(case.report)]
  #
} else if (method == "range.alive") {
  #
  case.report <- sapply(unique.cases, (function(ucs){
    slice <- tmp.mut.dset[tmp.mut.dset$case_id == ucs,]
    slice.ratio <- sum(slice$range.status == "affected")
    if (slice.ratio > 0) {
      "in"
    } else if (slice.ratio == 0){
      "out"
    } else {
      NA
    }
  }))
  exluded.cases <-
    unique(c(exluded.cases, names(case.report[is.na(case.report)])))
  case.report <- case.report[!is.na(case.report)]
  #
}
#
#
case.in <- names(case.report[case.report == "in"])
case.in <- unique(case.in[!case.in %in% exluded.cases])
#
case.out <- names(case.report[case.report == "out"])
case.out <- unique(case.out[!case.out %in% exluded.cases])
#
case.exclude <- unique(exluded.cases)
case.bckground <-
  unique(tot.cases[! tot.cases %in% c(case.in, case.out, case.exclude)])
#
#
# Handle case lists
#
mutcases.by.range <- list()
mutcases.by.range$gene.symbol <- gene.symbol
```

```
  mutcases.by.range$study.id <- cancer.study
  mutcases.by.range$window <- list()
  mutcases.by.range$window$min <- window.min
  mutcases.by.range$window$max <- window.max
  mutcases.by.range$method <- method
  mutcases.by.range$clinic.data <- clinic.data
  #
  mutcases.by.range$calls <- list()
  mutcases.by.range$calls$data.type <- data.type
  mutcases.by.range$calls$case.in <- case.in
  mutcases.by.range$calls$case.out <- case.out
  mutcases.by.range$calls$case.bckground <- case.bckground
  mutcases.by.range$calls$case.exclude <- case.exclude
  #
  #
  return(mutcases.by.range)
}
#
#
call_survival_by_range <- function(mutcases.list)
{
  #retrieve stuff from the list
  data.type <- mutcases.list$calls$data.type
  case.in <- mutcases.list$calls$case.in
  case.out <- mutcases.list$calls$case.out
  case.bckground <- mutcases.list$calls$case.bckground
  #
  # Check and warn
  sapply(c("case.in", "case.out", "case.bckground"), (function(elem){
    if(is.null(mutcases.list$calls[[elem]]))
      message(paste(elem, "element is empty!!"))
  }))
  #
  #
  if (data.type == "raw") {
    tmp.cd <- mutcases.list$clinic.data
    i.keep <- rownames(tmp.cd) %in% c(case.in, case.out, case.bckground)
    tmp.cd <- tmp.cd[i.keep,]
    my.f <- sapply(rownames(tmp.cd), (function(elm){
      if (elm %in% case.in) {
        "mut.in"
      } else if (elm %in% case.out) {
        "mut.out"
      } else {
        "wt"
      }
    }))
    tmp.surv <- check_survival(tmp.cd, my.f)$data
    tmp.surv$CASE_ID <- rownames(tmp.surv)
    tmp.surv$OS_MONTHS <- round(as.numeric(tmp.surv$survival_time)/30 , 2)
    tmp.surv$OS_STATUS <-
      gsub("(^.*)dea(.*$)","DECEASED",tmp.surv$survival_status)
    tmp.surv$OS_STATUS <- gsub("(^.*)liv(.*$)","LIVING",tmp.surv$OS_STATUS)
```

```
    tmp.surv$group <- tmp.surv$grouping
    rownames(tmp.surv) <- NULL
    tmp.surv <- tmp.surv[,c("CASE_ID", "OS_MONTHS", "OS_STATUS",   "group")]
    mutcases.list$km.clin.data <- tmp.surv
  } else {
    #
    clin.data <-   mutcases.list$clinic.data
    if (sum(c("CASE_ID", "OS_MONTHS", "OS_STATUS") %in%
            colnames(clin.data)) == 3) {
      clin.data <- clin.data[,c("CASE_ID", "OS_MONTHS", "OS_STATUS")]
      clin.data <-
        clin.data[clin.data$CASE_ID %in% c(case.in, case.out, case.bckground),]
      clin.data$group <- "wt"
      clin.data[clin.data$CASE_ID %in% case.in, "group"] <- "mut.in"
      clin.data[clin.data$CASE_ID %in% case.out, "group"] <- "mut.out"
      mutcases.list$km.clin.data <- clin.data
    } else {
      mutcases.list$error <- TRUE
    }
  }
  return(mutcases.list)
}
#
#
merge_km_data <- function(km.data.list) {
  #cur.gene.symbol <- km.data.list[[1]]$gene.symbol
  my.symbols <- do.call(c, lapply(km.data.list, (function(lst){
    lst$gene.symbol
  })))
  top.gene <- names(sort(table(my.symbols), decreasing = TRUE)[1])
  if (sum(my.symbols != top.gene) > 0) {
    message(paste(sum(my.symbols != top.gene),
                  "datasets were excluded because different gene.symbols were tested..."))
    km.data.list <- km.data.list[my.symbols == top.gene]
  }
  #
  my.methods <- do.call(c, lapply(km.data.list, (function(lst){
    lst$method
  })))
  top.method <- names(sort(table(my.methods), decreasing = TRUE)[1])
  if (sum(my.methods != top.method) > 0) {
    message(paste(sum(my.methods != top.method),
                  "datasets were excluded because different analys methods were used..."))
    km.data.list <- km.data.list[my.methods == top.method]
  }
  #
  final.km.result <- list()
  final.km.result$gene.symbol <- top.gene
  final.km.result$method <- top.method
  final.km.result$km.clin.data <- do.call(rbind, lapply(km.data.list, (function(lst){
    lst$km.clin.data
  })))
  final.km.result$chk.gene <- my.symbols == top.gene
```

```
  final.km.result$chk.meth <- my.methods == top.method
  #
  return(final.km.result)
}
#
#
plot_km_custom <- function(survival.mutation.data,
                           colors = c("#377eb8", "#e41a1c",
                                      "#4daf4a", "#984ea3",
                                      "#ff7f00", "#ffff33"),
                           xlim = NULL,
                           ylim = c(0,1.1),
                           main = "",
                           xlab = "Months",
                           tucows.plot = TRUE, #TWo-group COmparison + Wild type Survival)
                           cust.mll3.lab = FALSE
)
{
  #Perform survival analysis
  km.data <- survival.mutation.data$km.clin.data
  gene.symbol <- survival.mutation.data$gene.symbol
  km.data$OS_MONTHS <- suppressWarnings(as.numeric(as.character(km.data$OS_MONTHS)))
  km.data <- km.data[!is.na(km.data$OS_MONTHS),]
  #
  if (tucows.plot == TRUE) {
    #pre-processing of data...
    survdata.for.anal <- km.data[km.data$group != "wt",]
    km.data$group[km.data$group == "wt"] <- "001"
    km.data$group[km.data$group == "mut.out"] <- "002"
    km.data$group[km.data$group == "mut.in"] <- "003"
    #...and color adjustment...
    colors <- c("gray60", colors)
  } else {
    survdata.for.anal <- km.data
  }
  #
  surv.fit <-survfit(survival::Surv(km.data$OS_MONTHS,
                                    km.data$OS_STATUS == "DECEASED") ~
                       km.data$group)
  #
  #
  # And then, plot
  if (main == "") {
    main <- gsub("[[:blank:]]{2}", " ",
                 paste("Survival by", gene.symbol ,"mutation status"))
  }
  if (!is.null(xlim)) {
    x.max <- max(xlim)
  } else {
    x.max <- NULL
  }
  #
  my.plot <- plot(surv.fit,
```

```
                col = colors,
                lwd = 2.25,
                xlim = xlim,
                ylim = ylim,
                main = main,
                ylab = "Survival",
                xlab = xlab,
                xmax = x.max)
if (is.null(xlim)){
  xlim = c(0, 1.1*max(my.plot$x))
}
segments(xlim[1], 0.5, xlim[2], 0.5, col = "gray10", lty = 3, lwd = 0.5)
#
if (survival.mutation.data$method == "missense.only"){
  custom.labs <- rev(c(paste(gene.symbol, "wt"), "Out of Domain Mut", "Domain Mut"))
} else {
  #
  custom.labs <- rev(c(paste(gene.symbol, "wt"),
                       "Mut out of Domain", "Domain Mut/Lost"))
  if (cust.mll3.lab) {
    custom.labs <- rev(c("KMT2C, BAP1 WT", "non-PHD KMT2C Mut", "BAP1 Mut/PHD Lost"))
  }
}
#
if (tucows.plot == TRUE){
  legend("topright",
         legend = custom.labs,
         lty = 1,
         lwd = 2.5,
         col = rev(colors[1:length(unique(km.data$group))]))
} else {
  legend("topright",
         legend = paste(gene.symbol, sort(unique(km.data$group))),
         lty = 1,
         lwd = 2.5,
         col = colors)
}
#
#
# If there are many groups, run stat test
if (length(unique(survdata.for.anal$group)) > 1) {
  log.rank.t <- survdiff(survival::Surv(survdata.for.anal$OS_MONTHS,
                                        survdata.for.anal$OS_STATUS == "DECEASED") ~
                           survdata.for.anal$group)
  p.vl <- 1 - pchisq(log.rank.t$chisq, length(log.rank.t$n) - 1)
  p.vl <- format(round(p.vl, 6), nsmall = 5)
  if (!is.null(ylim)) {
    my.y <- min(ylim) + ((max(ylim) - min(ylim)) * 0.04)
  } else {
    my.y <- 0.04
  }
  if (!is.null(xlim)) {
    my.x <- min(xlim) + ((max(xlim) - min(xlim)) * 0.02)
```

```
    } else {
      my.x <- max(my.plot$x) * 0.02
    }
    text(my.x,
         my.y,
         paste("Log-Rank Test: p-val =", p.vl),
         pos = 4, font = 2, cex = 0.86)
    #
    my.nums <- cbind(gsub("(^.+)=", "", names(log.rank.t$n)),
                     as.character(log.rank.t$n))
    my.num.text <- paste(sapply(1:nrow(my.nums), (function(jj){
      paste(my.nums[jj,], collapse = "#: ")
    })), collapse = ".  ")
    if (tucows.plot == TRUE) {
      if (!cust.mll3.lab) {
        my.num.text <- gsub("mut.in", "Domain Mut/Lost", my.num.text)
        my.num.text <- gsub("mut.out", "Out-of-Domain Mut", my.num.text)
      } else {
        my.num.text <- gsub("mut.in", "BAP1 Mut / PHD Lost", my.num.text)
        my.num.text <- gsub("mut.out", "non-PHD Mut", my.num.text)
      }
    }
    #
    text(my.x,
         my.y*2.5,
         my.num.text,
         pos = 4, font = 2, cex = 0.86)
    #
  }
}
#
get_hotspot_mutations <- function(gene.symbol,
                                  study.id = "all",
                                  precise = TRUE,
                                  freq = FALSE) {
  if(study.id[1] == "all") {
    study.id <- get_cancer_studies()[,1]
    study.id <- grep("tcga", study.id, value = TRUE)
    study.id <- grep("pub", study.id, value = TRUE, invert = TRUE)
  }
  #
  my.res.list <- lapply(study.id, (function(st.id){
    #
    cases.id <- get_case_lists(st.id)[,1]
    cases.id <- grep("sequenced|mutations", cases.id, value = TRUE)
    #
    profi.id <- get_genetic_profiles(st.id)[,1]
    profi.id <- grep("sequenced|mutations", profi.id, value = TRUE)
    #
    my.mut.gamma <- TCGAretriever::get_ext_mutation(case_id = cases.id,
                                                    gprofile_id = profi.id,
                                                    glist = gene.symbol)
```

```
  if (is.data.frame(my.mut.gamma) &
      "amino_acid_change" %in% colnames(my.mut.gamma) &
      nrow(my.mut.gamma) > 0 ) {
    aa.change <- gsub("^.*(splice).*$", "", my.mut.gamma$amino_acid_change)
    aa.change <- gsub("[[:alpha:]]|\\.|\\_.*$|", "", aa.change)
    #
    # Here we create a all-muts alternative
    aa.all.change <- gsub("\\*.*$", "", aa.change)
    #
  } else {
    aa.change <- (-1)
    aa.all.change <- (-1)
  }
  ret.list <- list()
  ret.list$mis.mut <- suppressWarnings(as.integer(aa.change))
  ret.list$all.mut <- suppressWarnings(as.integer(aa.all.change))
  ret.list
}))
#
# Making things a little more complex. Retrieve data from a list of lists
# First, missense mutations
mis.mut <- lapply(my.res.list, (function(iii){
  iii$mis.mut
}))
mut.positions <- do.call(base::c, mis.mut)
mut.positions <- mut.positions[! is.na(mut.positions)]
mut.positions <- sort(mut.positions[mut.positions > 0])
# Then, all mutations
all.mut <- lapply(my.res.list, (function(iii){
  iii$all.mut
}))
all.mut.positions <- do.call(base::c, all.mut)
all.mut.positions <- all.mut.positions[! is.na(all.mut.positions)]
all.mut.positions <- sort(all.mut.positions[all.mut.positions > 0])
#
#
if (precise == FALSE) {
  my.x <- 1:max(all.mut.positions)
} else {
  my.cds <- OrganismDbi::cds(Homo.sapiens::Homo.sapiens)
  my.cds.tx.id <-
    suppressMessages(OrganismDbi::select(Homo.sapiens::Homo.sapiens,
                                         keys = gene.symbol,
                                         keytype = "SYMBOL",
                                         columns = c("CDSID", "ENTREZID", "TXID")))
  my.cds.tx.id <- my.cds.tx.id[!is.na(my.cds.tx.id$CDSID), ]
  all.tx <- split(my.cds.tx.id, as.factor(my.cds.tx.id$TXID))
  iso.lens <- do.call(base::c, lapply(all.tx, (function(tx.lst){
    cds.seq <-
      suppressMessages(Biostrings::getSeq(BSgenome.Hsapiens.UCSC.hg19,
                                          my.cds[tx.lst$CDSID]))
    #paste(cds.seq, sep = "", collapse = "")
    pasted.seq <- paste(cds.seq, sep = "", collapse = "")
```

```
      pasted.seq <- gsub("(TAG$)|(TAA$)|(TGA$)", "", pasted.seq)
      #pasted.seq
      as.integer(nchar(pasted.seq)/3)
    })))
    max.len <- max(iso.lens)
    my.x <- 1:max.len
  }
  mis.counts <- sapply(my.x, (function(x){
    sum(mut.positions == x)
  }))
  all.counts <- sapply(my.x, (function(x){
    sum(all.mut.positions == x)
  }))
  #
  if(freq == TRUE) {
    mis.counts <- mis.counts / sum(all.counts)
    all.counts <- all.counts / sum(all.counts)
  }
  #
  final.result <- list()
  final.result$gene.symbol <- gene.symbol
  final.result$study.id <- study.id
  final.result$freq <- freq == TRUE
  final.result$mis.mut.counts <- mis.counts
  final.result$all.mut.counts <- all.counts
  return(final.result)
}
#
mutations_along_seq <- function(mutation.counts, bin = -1) {
  mis.mut.counts <- mutation.counts$mis.mut.counts
  all.mut.counts <- mutation.counts$all.mut.counts
  #
  if (bin == -1) {
    def.bin <- length(all.mut.counts) / 120
    bin <- as.integer(def.bin)
    if (bin < 2 & bin > 1.499) {
      bin <- 2
    } else if (bin < 1)
      bin <- 1
  }
  #
  my.xxx <- seq (0, length(all.mut.counts), by = bin)
  if(my.xxx[length(my.xxx)] < length(all.mut.counts)) {
    my.xxx[length(my.xxx)] <- length(all.mut.counts)
  }
  full_mut_count <- sapply(2:length(my.xxx), (function(jj){
    c(sum(all.mut.counts[(my.xxx[jj-1] + 1):(my.xxx[jj])] ),
      sum(mis.mut.counts[(my.xxx[jj-1] + 1):(my.xxx[jj])] ))
  }))
  result <- list()
  #
  result$binned.all.counts <- as.numeric(full_mut_count[1,])
  result$binned.mis.counts <- as.numeric(full_mut_count[2,])
```

```
  result$original.all.counts <- all.mut.counts
  result$original.mis.counts <- mis.mut.counts
  result$bin.size <- bin
  result$gene.symbol  <- mutation.counts$gene.symbol
  result$study.id  <- mutation.counts$study.id
  result$freq  <- mutation.counts$freq
  #
  #
  return(result)
}
plot_binned_mutcounts <- function(mut.binned,
                                  mut.type = "both",
                                  colors = c("#2166ac", "#f63535"),
                                  main = "",
                                  ylim = NULL) {
  # prepare data
  mtbn.all.bin <- mut.binned$binned.all.counts
  mtbn.mis.bin <- mut.binned$binned.mis.counts
  #
  mtbn.all.ori <- mut.binned$original.all.counts
  mtbn.mis.ori <- mut.binned$original.mis.counts
  #
  mtbn.size <- mut.binned$bin.size
  gene.symbol <- mut.binned$gene.symbol
  freq.bar <- mut.binned$freq
  # generate data frame
  # we generate two matrices and then we rbind them
  my.mat.1 <- cbind(c(1,1:length(mtbn.all.bin),length(mtbn.all.bin)),
                    as.integer(c(1,seq(1, length(mtbn.all.ori),
                                       along.with = mtbn.all.bin),
                                 length(mtbn.all.ori))),
                    c(0,mtbn.all.bin,0),
                    rep(1, length(mtbn.all.bin) +2))
  my.mat.2 <- cbind(c(1,1:length(mtbn.mis.bin),length(mtbn.mis.bin)),
                    as.integer(c(1,seq(1, length(mtbn.mis.ori),
                                       along.with = mtbn.mis.bin),
                                 length(mtbn.mis.ori))),
                    c(0,mtbn.mis.bin,0),
                    rep(2, length(mtbn.mis.bin) +2))
  #
  if(mut.type == "both"){
    my.df <- data.frame(rbind(my.mat.1, my.mat.2))
  } else if (mut.type == "missense") {
    my.df <- data.frame(my.mat.2)
    my.color <- colors[2]
  } else {
    my.df <- data.frame(my.mat.1)
    my.color <- colors[1]
  }
  colnames(my.df) <- c("x.pos", "aa.pos", "counts", "group")
  #
  # Plot
  plt <- ggplot(my.df, aes(x = factor(aa.pos), y = counts,
```

```
                          fill = as.factor(group)))
plt <- plt + geom_polygon(aes(group=group))
plt <- plt + theme_classic()
#
if (mut.binned$freq) {
  yax.lab <- "Mutation Freq."
} else {
  yax.lab <- "Mutation Counts"
}
#
if (main == "")
  main = gene.symbol
plt <- plt + labs(list(title = main, x = "Position (aa)", y = yax.lab))
#
tick.vect <- c(5,10,25,50,100,200,250,300,500,1000,2000,
               5000,10000,20000,25000,50000,100000,200000)
tick.chk.res <- abs(tick.vect - (max(my.df$aa.pos) /6)  )
my.tix <- which(tick.chk.res  == min(tick.chk.res))[1]
tick.at <- tick.vect[my.tix]
#
my.brk.labs <- seq(0, max(my.df$aa.pos), by = tick.at)
my.brk.labs[1] <- 1
if (my.brk.labs[length(my.brk.labs)] < (max(my.df$aa.pos) + 10))
  my.brk.labs <- c(my.brk.labs, max(my.df$aa.pos))
my.brks <- sapply(my.brk.labs[2: (length(my.brk.labs) - 1)], (function(ii){
  slice <- my.df[my.df$group == unique(my.df$group)[1],]
  ii.test <- abs(slice$aa.pos[3:(nrow(slice)-2)] - ii)
  my.ii <- which(ii.test == min(ii.test))[1]
  my.ii + 2
}))
my.brks <- c(min(my.df$aa.pos), my.df$aa.pos[my.brks], max(my.df$aa.pos))
#
plt <-
  plt + scale_x_discrete(breaks = my.brks, labels = my.brk.labs, expand = c(0,0))
if (!is.null(ylim)) {
  plt <- plt + scale_y_continuous(limits = ylim,  expand = c(0,0))
} else {
  plt <- plt +
    scale_y_continuous(limits = c(0, max(my.df$counts) * 1.2),  expand = c(0,0))
}
#
if (length(unique(my.df$group)) == 1) {
  plt <- plt + theme(legend.position="none")
  plt <- plt + scale_fill_manual(values = my.color)
} else {
  plt <- plt + scale_fill_manual(values = colors, breaks = rev(c(1,2)),
                                 labels = rev(c("Nonsense, Indel", "Missense-only")),
                                 name="Mutation Type")
  plt <- plt + theme(legend.title = element_text(colour="black",
                                                 size=12, face="bold"))
  plt <- plt + theme(legend.text = element_text(colour="black", size=10))
}
#
```

```
  plt <- plt + theme(axis.text.x = element_text(angle = 60, hjust = 1, size = 11),
                     axis.title.x = element_text(face="bold", size=13),
                     axis.title.y = element_text(face="bold", size=13),
                     plot.title = element_text(face="bold", size=15),
                     axis.line.y = element_line(color = "black", size = 1),
                     axis.line.x = element_line(color = "black", size = 1))
  return(plt)
}
#
barplot.gene <- function(gene.freq.mutat, min.color = "#deebf7",
                         max.color = "#08306b", ylim = NULL, gene.symbol = NULL){
  test.my.gene <- gene.freq.mutat
  my.colors <-
    colorRampPalette(c(min.color, max.color))(max(as.numeric(test.my.gene$mut.count))+2)
  pl.dim <- barplot(test.my.gene$ratio, col = my.colors[test.my.gene$mut.count],
                    ylab = "Freq. of Tumors with Mutation", ylim = ylim,
                    main = paste(gene.symbol, "mutations in TCGA datasets"),
                    names.arg = rownames(test.my.gene), las = 2)
  # key legend
  min.x <- min(pl.dim[,1])
  max.x <- max(pl.dim[,1])
  if (!is.null(ylim)){
    max.y <- max(ylim) * 0.99
  } else {
    max.y <- max(test.my.gene$ratio)
  }
  #
  this.x1 <- min.x + (0.65*(max.x - min.x))
  this.x2 <- min.x + (0.9*(max.x - min.x))
  this.y1 <- max.y * 0.92
  this.y2 <- max.y * 0.875
  #
  all.xx <- seq(this.x1, this.x2, length.out = length(my.colors) + 1)
  for (idx in 2:length(all.xx)) {
    polygon(c(all.xx[idx-1],all.xx[idx],all.xx[idx],all.xx[idx-1]),
            c(this.y1,this.y1, this.y2, this.y2),
            col = my.colors[idx-1], border = NA)
  }
  text(this.x2, 1.02*this.y2, max(test.my.gene$mut.count), pos = 4)
  text(this.x1, 1.02*this.y2, min(test.my.gene$mut.count), pos = 2)
  #
  text(0.5*(this.x1+this.x2), this.y1, paste("Total Gene Mutations in Dataset"),
       adj = c(0.5, -0.5), font = 2, cex = 0.75)
}
#
prep_custom_oncoprint <- function(csid = "brca_tcga",
                                  gene.01,
                                  gene.02,
                                  aarange.01 = NULL,
                                  aarange.02 = NULL,
                                  method = "range.alive") {
  #
  # brief param validation
```

```
if (is.null(aarange.01)) {
  aarange.01 <- c(-1, -1)
}
if (is.null(aarange.02)) {
  aarange.02 <- c(-1, -1)
}
#
# Loop through different datasets
#
# retrieve survival of patients with mutation in gene 1 (MLL3)
gene01.surv <- call_mutcases_by_range(gene.symbol = gene.01,
                                      cancer.study = csid,
                                      data.type = "tcga.id",
                                      clinic.data = NULL,
                                      window.min = aarange.01[1],
                                      window.max = aarange.01[2],
                                      method = method)
#
# retrieve survival of patients with mutation in gene 2 (BAP1)
gene02.surv <- call_mutcases_by_range(gene.symbol = gene.02,
                                      cancer.study = csid,
                                      data.type = "tcga.id",
                                      clinic.data = NULL,
                                      window.min = aarange.02[1],
                                      window.max = aarange.02[2],
                                      method = method)
#
#
# collect cases
#
EXCL <- unique (c(gene01.surv$calls$case.exclude,
                  gene02.surv$calls$case.exclude))
#
GN1.in  <- gene01.surv$calls$case.in
GN1.out <- gene01.surv$calls$case.out
GN1.bck <- gene01.surv$calls$case.bckground
#
GN2.in  <- gene02.surv$calls$case.in
GN2.out <- gene02.surv$calls$case.out
GN2.bck <- gene02.surv$calls$case.bckground
#
GN1.in  <- GN1.in[!GN1.in %in% EXCL]
GN1.out <- GN1.out[!GN1.out %in% EXCL]
GN1.bck <- GN1.bck[!GN1.bck %in% EXCL]
#
GN2.in <- GN2.in[!GN2.in %in% EXCL]
GN2.out <- GN2.out[!GN2.out %in% EXCL]
GN2.bck <- GN2.bck[!GN2.bck %in% EXCL]
#
# Now, retrieve CNA data, looking for
cna.prof <- grep("_gistic$", get_genetic_profiles(csid = csid)[,1],
                 value = TRUE, ignore.case = TRUE)
if (length(cna.prof) == 0) {
```

```
  cna.prof <- grep("_cna$", get_genetic_profiles(csid = csid)[,1],
                   value = TRUE, ignore.case = TRUE)
}
cna.case <- grep("cnaseq$", get_case_lists(csid = csid)[,1],
                 value = TRUE, ignore.case = TRUE)
cna.data <- suppressWarnings(get_profile_data(case_id = cna.case,
                                              gprofile_id = cna.prof,
                                              glist = c(gene.01, gene.02)))
#
# define included samples (CNASEQ)
# I only care about deletions (-2)
all.cnaseq.id <- grep("^TCGA", names(cna.data), value = TRUE)
gene.01.del <- cna.data[cna.data$COMMON == gene.01, -c(1:2)]
gene.01.del <- names(gene.01.del)[as.numeric(gene.01.del[1,]) < (-1.5)]
gene.02.del <- cna.data[cna.data$COMMON == gene.02, -c(1:2)]
gene.02.del <- names(gene.02.del)[as.numeric(gene.02.del[1,]) < (-1.5)]
#
# trim away
GN1.in <- GN1.in[GN1.in %in% all.cnaseq.id]
GN1.out <- GN1.out[GN1.out %in% all.cnaseq.id]
GN1.bck <- GN1.bck[GN1.bck %in% all.cnaseq.id]
GN1.bck <- GN1.bck[!(GN1.bck %in% gene.01.del)]
#
GN2.in <- GN2.in[GN2.in %in% all.cnaseq.id]
GN2.out <- GN2.out[GN2.out %in% all.cnaseq.id]
GN2.bck <- GN2.bck[GN2.bck %in% all.cnaseq.id]
GN2.bck <- GN2.bck[!(GN2.bck %in% gene.02.del)]
#
#
# now, let's create a nice data.frame
# Manual scoring (5, 11, 23, 47, 95, 191, 383, 767, 1535)
my.df <- do.call(rbind,
                 lapply(grep("^TCGA", unique(all.cnaseq.id), value = TRUE),
                        (function(id){
                          #
                          # initialize score
                          score <- 0
                          #
                          # look for gene 1 (MLL3) statuses
                          #
                          if (id %in% gene.01.del){
                            tmp.del <- "DEL"
                            score <- score + 383
                          } else {
                            tmp.del <- "OK"
                          }
                          if (id %in% GN1.in) {
                            tmp.stat <- "LOST"
                            score <- score + 191
                          } else if ( id %in% GN1.out) {
                            tmp.stat <- "MUT"
                            score <- score + 95
                          } else {
```

```
                            tmp.stat <- "WT"
                          }
                          #
                          tmp.GN1 <- c(id=id,
                                       gene=gene.01,
                                       del=tmp.del,
                                       mut.stat=tmp.stat)
                          #
                          # look for gene 2 (BAP1) statuses
                          if (id %in% gene.02.del){
                            tmp.del <- "DEL"
                            score <- score + 47
                          } else {
                            tmp.del <- "OK"
                          }
                          if (id %in% GN2.in) {
                            tmp.stat <- "LOST"
                            score <- score + 23
                          } else if ( id %in% GN2.out) {
                            tmp.stat <- "MUT"
                            score <- score + 11
                          } else {
                            tmp.stat <- "WT"
                          }
                          #
                          tmp.GN2 <- c(id=id,
                                       gene=gene.02,
                                       del=tmp.del,
                                       mut.stat=tmp.stat)
                          #
                          data.frame(rbind(tmp.GN1, tmp.GN2),
                                     score = score,
                                     stringsAsFactors = FALSE,
                                     row.names = NULL)
                        })))
#
# Prepare contingency table
tcga.id.all <- grep("^TCGA", unique(my.df$id), value = TRUE)
cont.tab <- data.frame(matrix(0, nrow = length(tcga.id.all), ncol = 6),
                       row.names = tcga.id.all)
colnames(cont.tab) <- c("G1.DEL", "G1.IN", "G1.OUT",
                        "G2.DEL", "G2.IN", "G2.OUT")
cont.tab[grep("^TCGA",gene.01.del, value = TRUE), "G1.DEL"] <- 1
cont.tab[grep("^TCGA",GN1.in,value = TRUE), "G1.IN"] <- 1
cont.tab[grep("^TCGA",GN1.out, value = TRUE), "G1.OUT"] <- 1
cont.tab[grep("^TCGA",gene.02.del, value = TRUE), "G2.DEL"] <- 1
cont.tab[grep("^TCGA",GN2.in,value = TRUE), "G2.IN"] <- 1
cont.tab[grep("^TCGA",GN2.out, value = TRUE), "G2.OUT"] <- 1
#
# 3x3 groups
names.row <- paste(gene.01, c("WT", "MUT", "LOST"), sep = ".")
names.col <- paste(gene.02, c("WT", "MUT", "LOST"), sep = ".")
#
```

```
  tab.9g <- matrix(0, ncol = 3, nrow = 3, dimnames = list(names.row, names.col))
  tab.9g[1,1] <- sum(apply(cont.tab, 1, sum) == 0)
  tab.9g[2,1] <- sum(cont.tab[,3] == 1 & apply(cont.tab[,4:6], 1, sum) == 0)
  tab.9g[3,1] <- sum(apply(cont.tab[,1:2], 1, sum) > 0 &
                       apply(cont.tab[,4:6], 1, sum) == 0)
  #
  tab.9g[1,2] <- sum(apply(cont.tab[,1:3], 1, sum) == 0 & cont.tab[,6]  == 1)
  tab.9g[2,2] <- sum(cont.tab[,3] == 1 & cont.tab[,6] == 1)
  tab.9g[3,2] <- sum(apply(cont.tab[,1:2], 1, sum) > 0 & cont.tab[,6] == 1)
  #
  tab.9g[1,3] <- sum(apply(cont.tab[,1:3], 1, sum) == 0 &
                       apply(cont.tab[,4:5], 1, sum) > 0)
  tab.9g[2,3] <- sum(cont.tab[,3] == 1 & apply(cont.tab[,4:5], 1, sum) > 0)
  tab.9g[3,3] <- sum(apply(cont.tab[,1:2], 1, sum) > 0 &
                       apply(cont.tab[,4:5], 1, sum) > 0)
  #
  # assemble and return
  final.out <- list()
  final.out[["data"]] <- my.df
  final.out[["count.table"]] <- tab.9g
  final.out[["params"]] <- list()
  final.out$params[["csid"]] <- csid
  final.out$params[["gene.01"]] <- gene.01
  final.out$params[["gene.02"]] <- gene.02
  final.out$params[["aarange.01"]] <- aarange.01
  final.out$params[["aarange.02"]] <- aarange.02
  final.out$params[["method"]] <- method
  #
  return(final.out)
}
#
plot_custom_oncoprint <- function(oncoprint.df, trim.wt = TRUE) {
  #
  my.df <- oncoprint.df$data
  my.genes <- c(oncoprint.df$params$gene.01, oncoprint.df$params$gene.02)
  square_w <- .6
  square_h <- .4
  #
  if (trim.wt) {
    my.df <- my.df[my.df$score > 0,]
  }
  #
  #
  pl <- ggplot(my.df, aes(x=id, y=gene)) +
    geom_tile(fill="white", colour="white", size=1.1) +
    scale_y_discrete(breaks=my.genes,limits = rev(my.genes)) +
    scale_x_discrete(limits = unique(my.df$id[order(my.df$score,
                                                    decreasing = TRUE)])) +
    geom_tile(data=my.df, aes(x=id, y=gene),
              inherit.aes=FALSE, width=.9, height=.9,
              fill="gray85", colour=NA, size=2) +
    geom_tile(data=my.df[my.df$del == "DEL",],
              aes(x=id, y=gene), inherit.aes=FALSE,
```

```
              width=.9, height=.9, fill="blue",
              colour=NA, size=2) +
    geom_tile(data=my.df[my.df$mut.stat == "LOST",],
              aes(x=id, y=gene), inherit.aes=FALSE,
              width=square_w, height=square_h, fill="red2") +
    geom_tile(data=my.df[my.df$mut.stat == "MUT",],
              aes(x=id, y=gene), inherit.aes=FALSE,
              width=square_w, height=square_h, fill="orange") +
    theme_minimal() + xlab("") + ylab("") +
    theme(panel.background = element_blank(), line = element_blank(),
          axis.text.x = element_blank(),
          axis.text.y = element_text(size = 12, colour = "black"),
          plot.title = element_text(face = "bold", size = 12, hjust = 0.5))
  #
  final.out <- list()
  final.out[["plot"]] <- pl
  final.out[["plot.legend"]] <-
    paste("blue box = deletion;",
          "red square = mutation disrupting the domain of interest;",
          "orange square = mutation not affecting the domain of interest",
          sep = "\n")
  return(final.out)
}
#
std_2gene_oncoprint <- function(csid = "nsclc_tcga_broad_2016",
                                gene.01,
                                gene.02,
                                tcga.only = FALSE,
                                trim.wt = TRUE) {
  #
  my.genes <- c(gene.01, gene.02)
  tmp.cases <- grep("cnaseq$", get_case_lists(csid)[,1], value = TRUE)
  tmp.clin <- suppressWarnings(get_clinical_data(case_id = tmp.cases))
  mut.profi <- grep("mutations$",
                    get_genetic_profiles(csid = csid)[,1], value = TRUE)
  cna.profi <- grep("_gistic$",
                    get_genetic_profiles(csid = csid)[,1], value = TRUE)
  if(length(cna.profi) == 0)
    cna.profi <- grep("_cna$",
                      get_genetic_profiles(csid = csid)[,1], value = TRUE)
  #
  mut.Data <- suppressWarnings({
    get_profile_data(case_id = tmp.cases,
                     gprofile_id = mut.profi,
                     glist = my.genes)
  })
  rownames(mut.Data) <- mut.Data$COMMON
  mut.Data <- mut.Data[,-c(1,2)]
  #
  cna.Data <- suppressWarnings({
    get_profile_data(case_id = tmp.cases,
                     gprofile_id = cna.profi,
                     glist = my.genes)
```

```
})
rownames(cna.Data) <- cna.Data$COMMON
cna.Data <- cna.Data[,-c(1,2)]
#
# Remove splicing variants
mut.Data[1,] <- gsub("X[[:digit:]]{1,4}_splice", "", mut.Data[1,])
mut.Data[2,] <- gsub("X[[:digit:]]{1,4}_splice", "", mut.Data[2,])
#
# define base that is common
full.ids <- names(mut.Data)[names(mut.Data) %in% names(cna.Data)]
full.tab <- do.call(rbind, lapply(full.ids, (function(id){
  #
  # init
  score = 0
  GN1.del <- "OK"
  GN2.del <- "OK"
  #
  if (as.numeric(cna.Data[my.genes[1], id]) < (-1)){
    GN1.del <- "DEL"
    score = score + 767
  }
  if (as.numeric(cna.Data[my.genes[2], id]) < (-1)){
    GN2.del <- "DEL"
    score = score + 47
  }
  #
  #search for mutations - gene 1, then gene 2
  if (mut.Data[my.genes[1], id] %in% c("", "NaN")) {
    GN1.lost <- "WT"
  } else if (regexpr("\\*", mut.Data[my.genes[1], id]) > 0) {
    GN1.lost <- "NSN"
    score <- score + 383
  } else if (regexpr("([[:alpha:]]{1}[[:digit:]]{1,4}del)|(_)",
                     mut.Data[my.genes[1], id]) > 0) {
    GN1.lost <- "IND"
    score <- score + 191
  } else {
    GN1.lost <- "MIS"
    score <- score + 95
  }
  #
  if (mut.Data[my.genes[2], id] %in% c("", "NaN")) {
    GN2.lost <- "WT"
  } else if (regexpr("\\*", mut.Data[my.genes[2], id]) > 0) {
    GN2.lost <- "NSN"
    score <- score + 23
  } else if (regexpr("([[:alpha:]]{1}[[:digit:]]{1,4}del)|(_)",
                     mut.Data[my.genes[2], id]) > 0) {
    GN2.lost <- "IND"
    score <- score + 11
  } else {
    GN2.lost <- "MIS"
    score <- score + 5
```

```
  }
  # return to upper level
  data.frame(rbind(c(id=id, gene=my.genes[1], del=GN1.del, mut=GN1.lost),
                   c(id=id, gene=my.genes[2], del=GN2.del, mut=GN2.lost)),
             score=score, stringsAsFactors = FALSE, row.names = NULL)

})))
#
# Apply some corrections (if needed)
if (tcga.only) {
  full.tab <- full.tab[grep("^TCGA", full.tab$id), ]
}
my.df <- data.frame(full.tab[order(full.tab$score, decreasing = TRUE), ],
                    row.names = NULL)
if (trim.wt) {
  my.df <- my.df[my.df$score>0,]
}
#
# Prepare to plot!
# Once again, I do not care of amplifications
#
square_h = 0.6
square_w = 0.5
gpp <- ggplot(my.df, aes(x=id, y=gene)) +
  geom_tile(fill="white", colour="white", size=1.1) +
  scale_y_discrete(breaks=my.genes, limits = rev(my.genes)) +
  scale_x_discrete(limits = unique(my.df$id[order(my.df$score, decreasing = TRUE)])) +
  geom_tile(data=my.df, aes(x=id, y=gene),
            inherit.aes=FALSE, width=.8, height=.9, fill="gray85", colour=NA, size=2) +
  geom_tile(data=my.df[my.df$del == "DEL",],
            aes(x=id, y=gene), inherit.aes=FALSE, width=.9,
            height=.9, fill="#345fdf", colour=NA, size=2) +

  geom_tile(data=my.df[my.df$mut == "NSN",],
            aes(x=id, y=gene), inherit.aes=FALSE,
            width=square_w, height=square_h, fill="#1a1c1b") +
  geom_tile(data=my.df[my.df$mut == "IND",],
            aes(x=id, y=gene), inherit.aes=FALSE,
            width=square_w, height=square_h, fill="#feb24c") +
  geom_tile(data=my.df[my.df$mut == "MIS",], aes(x=id, y=gene),
            inherit.aes=FALSE, width=square_w, height=square_h, fill="#35ae5d") +
  theme_minimal() + xlab("") + ylab("") +
  theme(panel.background = element_blank(), line = element_blank(),
        axis.text.x = element_blank(),
        axis.text.y = element_text(size = 11, face = "bold", colour = "black"),
        plot.title = element_text(face = "bold", size = 12, hjust = 0.5)) + ggtitle(csid)
#
# Prepare contingency table
id.all <- unique(full.tab$id)
cont.tab <- data.frame(matrix(0, nrow = length(id.all), ncol = 8),
                       row.names = id.all)
colnames(cont.tab) <- c("G1.DEL", "G1.NSN", "G1.IND", "G1.MIS",
                        "G2.DEL", "G2.NSN", "G2.IND", "G2.MIS")
```

```
  #
  TMP.tab <- full.tab[full.tab$gene == my.genes[1],]
  rownames(TMP.tab) <- TMP.tab$id
  TMP.tab <- TMP.tab[id.all,]
  cont.tab[TMP.tab$del == "DEL", "G1.DEL"] <- 1
  cont.tab[TMP.tab$mut == "NSN", "G1.NSN"] <- 1
  cont.tab[TMP.tab$mut == "IND", "G1.IND"] <- 1
  cont.tab[TMP.tab$mut == "MIS", "G1.MIS"] <- 1
  #
  TMP.tab <- full.tab[full.tab$gene == my.genes[2],]
  rownames(TMP.tab) <- TMP.tab$id
  TMP.tab <- TMP.tab[id.all,]
  cont.tab[TMP.tab$del == "DEL", "G2.DEL"] <- 1
  cont.tab[TMP.tab$mut == "NSN", "G2.NSN"] <- 1
  cont.tab[TMP.tab$mut == "IND", "G2.IND"] <- 1
  cont.tab[TMP.tab$mut == "MIS", "G2.MIS"] <- 1
  #
  # 2x2 groups
  names.row <- paste(gene.01, c("WT", "MUT"), sep = ".")
  names.col <- paste(gene.02, c("WT", "MUT"), sep = ".")
  #
  tab.4g <- matrix(0, ncol = 2, nrow = 2, dimnames = list(names.row, names.col))
  tab.4g[1,1] <- sum(apply(cont.tab, 1, sum) == 0)
  tab.4g[2,1] <- sum(apply(cont.tab[,1:4], 1, sum) > 0 &
                       apply(cont.tab[,5:8], 1, sum) == 0)
  tab.4g[1,2] <- sum(apply(cont.tab[,1:4], 1, sum) == 0 &
                       apply(cont.tab[,5:8], 1, sum) > 0)
  tab.4g[2,2] <- sum(apply(cont.tab[,1:4], 1, sum) > 0 &
                       apply(cont.tab[,5:8], 1, sum) > 0)
  #
  # assemble and return
  final.out <- list()
  final.out[["plot"]] <- gpp
  final.out[["plot.legend"]] <-
    paste("blue box = deep deletion;",
          "black square = nonsense mutation;",
          "yellow square = coding INDEL",
          "green square = missense mutation",
          sep = "\n")
  final.out[["data"]] <- my.df
  final.out[["count.full"]] <- cont.tab
  final.out[["count.table"]] <- tab.4g
  final.out[["params"]] <- list()
  final.out$params[["csid"]] <- csid
  final.out$params[["gene.01"]] <- gene.01
  final.out$params[["gene.02"]] <- gene.02
  final.out$params[["tcga.only"]] <- tcga.only
  final.out$params[["trim.wt"]] <- trim.wt
  #
  return(final.out)
}
```

Appendix 2: sessionInfo

```
## R version 3.3.1 (2016-06-21)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04 LTS
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] Homo.sapiens_1.3.1
##  [2] TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
##  [3] BSgenome.Hsapiens.UCSC.hg19_1.4.0
##  [4] BSgenome_1.40.1
##  [5] rtracklayer_1.32.2
##  [6] Biostrings_2.40.2
##  [7] XVector_0.12.1
##  [8] org.Hs.eg.db_3.3.0
##  [9] OrganismDbi_1.14.1
## [10] GenomicFeatures_1.24.5
## [11] GenomicRanges_1.24.3
## [12] GenomeInfoDb_1.8.7
## [13] GO.db_3.3.0
## [14] AnnotationDbi_1.34.4
## [15] IRanges_2.6.1
## [16] S4Vectors_0.10.3
## [17] Biobase_2.32.0
## [18] BiocGenerics_0.18.0
## [19] ggplot2_2.2.1
## [20] gplots_3.0.1
## [21] survival_2.41-2
## [22] TCGAretriever_1.3
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.10               lattice_0.20-34
##  [3] Rsamtools_1.24.0           gtools_3.5.0
##  [5] assertthat_0.1             rprojroot_1.2
##  [7] digest_0.6.12              R6_2.2.0
##  [9] plyr_1.8.4                 backports_1.0.5
## [11] RSQLite_1.1-2              evaluate_0.10
## [13] httr_1.2.1                 BiocInstaller_1.22.3
## [15] zlibbioc_1.18.0            curl_2.3
## [17] lazyeval_0.2.0             gdata_2.17.0
## [19] Matrix_1.2-8               rmarkdown_1.4
## [21] labeling_0.3               splines_3.3.1
```

```
## [23] BiocParallel_1.6.6         stringr_1.2.0
## [25] RCurl_1.95-4.8             biomaRt_2.28.0
## [27] munsell_0.4.3              htmltools_0.3.5
## [29] SummarizedExperiment_1.2.3 tibble_1.2
## [31] XML_3.98-1.5               GenomicAlignments_1.8.4
## [33] bitops_1.0-6               grid_3.3.1
## [35] RBGL_1.48.1                gtable_0.2.0
## [37] DBI_0.6                    magrittr_1.5
## [39] scales_0.4.1               graph_1.50.0
## [41] KernSmooth_2.23-15         stringi_1.1.3
## [43] tools_3.3.1                yaml_2.1.14
## [45] colorspace_1.3-2           caTools_1.17.1
## [47] memoise_1.0.0              knitr_1.15.1
```