Prognostic Risk and Mutational Co-occurrance in AML

Contents


```
library(rlang)
library(tictoc)
library(eulerr)
library(caret)
library(gridExtra)
library(tidyverse)
```

Importing Data

AMLSG data import

```
#Load datasets

# clinicalData holds FLT3-ITD, NPM1, and CEBPA mutational data
load("./data/AMLSG_Clinical_Anon.RData") #loaded as clinicalData

rawFLT3 <- read.delim("./data/AMLSG_FLT3ITD.txt")
rawKaryotype <- read.delim("./data/AMLSG_Karyotypes.txt")
rawGenetic <- read.delim("./data/AMLSG_Genetic.txt")
rawClassification <- read.delim("./data/AMLSG_Classification.txt")
```

```
#Select only relevant fields, convert to factor
clinicalTrim <- clinicalData %>%
```

```r
  select(PDID, CEBPA, NPM1, FLT3_ITD) %>%
  mutate_all(funs(as.factor(.)))
summary(clinicalTrim)
```

```
##        PDID        CEBPA       NPM1      FLT3_ITD
##  PD10789a:   1   0   :1384   0:1104   0:1199
##  PD10790a:   1   1   :  56   1: 436   1: 341
##  PD10792a:   1   2   :  73
##  PD10793a:   1   NA's:  27
##  PD10794a:   1
##  PD10795a:   1
##  (Other) :1534
```

Mutation annotation (FLT3, NPM1, CEBPA)

CEBPA annotation CEBPA is annotated for both monoallelic and biallelic
Given the annotation is present, we will only use biallelic samples as CEBPA-mutated

Excluding CEBPA monoallelic removes 56 positive-samples

```r
#Converting FLT3/NPM1/CEBPA calls to logical variables
clinical_f <- clinicalTrim %>%
  mutate(FLT3_ITD =
           ifelse(clinicalTrim$FLT3_ITD == 1, TRUE,
                  ifelse(clinicalTrim$FLT3_ITD == 0, FALSE, NA)))
clinical_fn <- clinical_f %>%
  mutate(NPM1 =
           ifelse(clinical_f$NPM1 == 1, TRUE,
                  ifelse(clinical_f$NPM1 == 0, FALSE, NA)))

#Counting only CEBPA biallelic
clinical_fnc <- clinical_fn %>%
  mutate(CEBPA =
           ifelse(clinical_fn$CEBPA == 1, FALSE, #Monoallelic counts as non-mutated
                  ifelse(clinical_fn$CEBPA == 2, TRUE,
                         ifelse(clinical_fn$CEBPA == 0, FALSE, NA))))
```

Karyotype cleanup

Substantial portion of samples have no karyotype annotation

```r
summary(rawKaryotype)
```

```
##     Study           PDID                          karyotype
##  _07-04:766   PD10789a:   1   46,XX                     :332
##  98A   :632   PD10790a:   1   46,XY                     :312
##  98B   :176   PD10792a:   1   no metaphases             : 97
##               PD10793a:   1   na                        : 36
##               PD10794a:   1   46,XY,inv(16)(p13q22)   : 15
##               PD10795a:   1   46,XY,t(15;17)(q22;q21): 15
##               (Other) :1568   (Other)                   :767
```

```r
#Joining karyotype data with annotated mutational data
clinkaryo <- full_join(clinical_fnc, rawKaryotype, by = "PDID")
```

```r
#convert karyotype to character variable
clinkaryo$karyotype <- as.character(clinkaryo$karyotype)

#Replace incomplete/non-conforming karyotype entries to NA (later removed)
#String patterns determined through manual review of dataset
clin_cleankaryo <- clinkaryo %>%
  mutate(karyotype =
           ifelse(
             str_detect(clinkaryo$karyotype,
                        regex(
                          paste("no metaphases",
                                "^na$", #detects records where the entry is only "na"
                                "no analysis",
                                "no material",
                                "PCR",
                                "FISH",
                                 "metaphase",
                                 "tetraploid",
                                 "^ND", #detects records beginning with "ND"
                                 "n\\.d\\.", #detects "n.d."
                                 "outside",
                                 "incompl", sep = "|"),
                               ignore_case = TRUE)), #matches are not case sensitive
                  NA, clinkaryo$karyotype)
         )

#Additional cleaning, taking records with free-text descriptions and converting to NA
clin_clean_na_karyotype <- clin_cleankaryo %>%
  mutate(karyotype =
           ifelse(str_detect(clinkaryo$karyotype,
                        paste("Pentaploide Metaphasen",
                              "Komplexer Karyotyp",
                              "Keine analysierbaren", sep = "|"
                        )),
                  NA, clin_cleankaryo$karyotype)
         )
```

Mutation Annotation (TP53, RUNX1, ASXL1)

Adding all mutations regardless of "Consequence" or "Result" - Seems to match analysis performed in original AMLSG paper

Samples without a called mutation are considered negative for mutations at the given genes

```r
#Create list of samples mutated for TP53
P53 <- rawGenetic %>%
  filter(GENE == "TP53") %>% #If there is a mutation recorded at TP53
  select(SAMPLE_NAME, GENE) %>% #Remove other fields
  mutate(TP53 = TRUE) %>%  #Mark sample as mutated for TP53
  rename(PDID = SAMPLE_NAME) %>% #Uniform naming
  select(-GENE) %>%
  unique() #Keep only unique records of sample-TP53 mutation pairs
```

```r
#Create list of samples mutated for RUNX1
RUNX1 <- rawGenetic %>%
  filter(GENE == "RUNX1") %>%
  select(SAMPLE_NAME, GENE) %>%
  mutate(RUNX1 = TRUE) %>%
  rename(PDID = SAMPLE_NAME) %>%
  select(-GENE) %>%
  unique()

#Create list of samples mutated for ASXL1
ASXL1 <- rawGenetic %>%
  filter(GENE == "ASXL1") %>%
  select(SAMPLE_NAME, GENE) %>%
  mutate(ASXL1 = TRUE) %>%
  rename(PDID = SAMPLE_NAME) %>%
  select(-GENE) %>%
  unique()

#Join lists of mutations for three genes with earlier data
clin_newmutations <- clin_clean_na_karyotype %>%
  full_join(.,P53, by = "PDID") %>%
  full_join(.,RUNX1, by = "PDID") %>%
  full_join(., ASXL1, by = "PDID")

#Convert NA's into FALSE for these fields
#The absence of a mutation is interpreted to be a wildtype gene

mutated.fields <- c("TP53", "RUNX1", "ASXL1")
mutated.only <- clin_newmutations[mutated.fields]
mutated.only[is.na(mutated.only)] <- FALSE #Fields where gene is NA converted to FALSE (wildtype)
clin_newmutations[mutated.fields] <- mutated.only #Add data back into core dataframe

#Converting to common format
amlsg <- clin_newmutations %>%
  rename(CYTOGENETICS = karyotype) %>%
  select(PDID, CYTOGENETICS, FLT3_ITD, NPM1,
         CEBPA, TP53, RUNX1, ASXL1)

#Number of Incomplete records
amlsg %>%
  filter(is.na(CYTOGENETICS) | is.na(FLT3_ITD) |
           is.na(NPM1) | is.na(CEBPA)) %>%
  nrow()
```

```
## [1] 193
```

```r
#Number of records Incomplete for all fields
amlsg %>%
  filter(is.na(CYTOGENETICS) & is.na(FLT3_ITD) &
           is.na(NPM1) & is.na(CEBPA)) %>%
  nrow()
```

```
## [1] 30
```

```
#Number of Complete records
amlsg %>%
  filter(!is.na(CYTOGENETICS) & !is.na(FLT3_ITD) &
           !is.na(NPM1) & !is.na(CEBPA)) %>%
  nrow()
```

```
## [1] 1384
```

```
#Proceed with data only complete for all fields
#No need to filter on records incomplete for TP53/RUNX1/ASXL1
#since full coverage was assumed
amlsg_complete <- amlsg %>%
  filter(!is.na(CYTOGENETICS) & !is.na(FLT3_ITD) &
           !is.na(NPM1) & !is.na(CEBPA)) %>%
  mutate(source = "AMLSG") #Tag records with source of data
```

TCGA public AML data

Karyotype cleanup

```
#Import TCGA data
raw.karyotype <- read.delim("./data/laml_tcga_pub/data_clinical.txt",
                            skip = 5, header = TRUE)

trim.karyotype <- raw.karyotype %>% select(PATIENT_ID, CYTOGENETICS,
                                          INFERRED_GENOMIC_REARRANGEMENT,
                                          RISK_CYTO, RISK_MOLECULAR,
                                          HISTOLOGICAL_SUBTYPE,
                                          CYTOGENETIC_CODE_OTHER)

#Convert non-conforming karyotype entries to NA (later removed)
trim.karyotype$CYTOGENETICS <- as.character(trim.karyotype$CYTOGENETICS)
clean.karyotype <- trim.karyotype %>%
    mutate(CYTOGENETICS = (
           ifelse(str_detect(trim.karyotype$CYTOGENETICS,
                             regex(
                               paste("no metaphases",
                                     "^na$",
                                     "no analysis",
                                     "no material",
                                     "PCR",
                                     "FISH",
                                      "metaphase",
                                      "tetraploid",
                                      "^ND",
                                      "n\\.d\\.",
                                      "outside",
                                      "incompl", sep = "|"),
                                    ignore_case = TRUE)),
                  NA, trim.karyotype$CYTOGENETICS)
    ))

summary(clean.karyotype)
```

```
##         PATIENT_ID  CYTOGENETICS
##  TCGA-AB-2802:  1   Length:200
##  TCGA-AB-2803:  1   Class :character
##  TCGA-AB-2804:  1   Mode  :character
##  TCGA-AB-2805:  1
##  TCGA-AB-2806:  1
##  TCGA-AB-2807:  1
##  (Other)     :194
##        INFERRED_GENOMIC_REARRANGEMENT        RISK_CYTO
##                        :120           Good        : 37
##  t(15;17)(q22;q21)     : 13           Intermediate:115
##  t(16;16)(p13.11;q22.1):  8           N.D.        :  5
##  t(21;8)(q22.3;q22)    :  5           Poor        : 43
##  t(11;19)(q23;p13.1)   :  3
##  del17q11.2            :  2
##  (Other)               : 49
##      RISK_MOLECULAR                                  HISTOLOGICAL_SUBTYPE
##  Good        : 39    Normal Karyotype                          :86
##  Intermediate:106    Complex Cytogenetics                      :24
##  N.D.        :  4    PML-RARA                                  :20
##  Poor        : 51    Intermediate Risk Cytogenetic Abnormality:19
##                      CBFB-MYH11                                :12
##                      Poor Risk Cytogenetic Abnormality        :10
##                      (Other)                                   :29
##                               CYTOGENETIC_CODE_OTHER
##  Normal Karyotype                          :92
##  Complex Cytogenetics                      :24
##  Intermediate Risk Cytogenetic Abnormality:22
##  PML-RARA                                  :18
##  CBFB-MYH11                                :12
##  Poor Risk Cytogenetic Abnormality         :10
##  (Other)                                   :22
```

Mutation annotation

```
raw.mutations <- read.delim("./data/laml_tcga_pub/data_mutations_extended.txt",
                            header = TRUE, skip = 1)
```

NPM1 mutations

We're going to count all NPM1 somatically mutated patients, even though one patient has a point mutation (as opposed to the more common frameshift insertions). Patient: TCGA-AB-2915

TCGA in their paper counted this patient as NPM1 mutated (based on recalculation)

FLT3 ITD

TCGA didn't distinguish between FLT3-ITD and FLT3 mutations (kinase domain). We have code to grab only insertion/indel mutations in FLT3, and we manually confirmed these all occur in/around exon 14 (ITD) and not exon 20 (Kinase domain).

CEBPA mutations

New ELN guidelines specify that CEBPA mutations need to be biallelic to be relevant. However, the TCGA dataset does not clearly identify biallelic mutations as they were not relevant at the time. Accordingly, we will analyze this data set according to the 2008 ELN guidelines, which only specify CEBPA mutations generally, and count all CEBPA mutated samples.

```
short.mutations <- raw.mutations %>% select(PATIENT_ID = Matched_Norm_Sample_Barcode,
                                            Hugo_Symbol, VARIANT_CLASS)

#Create list of samples mutated at each gene
NPM.mutations <- short.mutations %>% filter(Hugo_Symbol == "NPM1") %>%
  select(PATIENT_ID) %>%
  mutate(NPM1 = TRUE) %>%
  unique()

FLT3.ITD <- short.mutations %>%
  filter(Hugo_Symbol == "FLT3" & VARIANT_CLASS == "insertion" |
           VARIANT_CLASS == "indel") %>%
  select(PATIENT_ID) %>%
  mutate(FLT3_ITD = TRUE) %>%
  unique()

CEBPA.mutations <- short.mutations %>%
  filter(Hugo_Symbol == "CEBPA") %>%
  select(PATIENT_ID) %>%
  mutate(CEBPA = TRUE) %>%
  unique()

TP53.mutations <- short.mutations %>%
  filter(Hugo_Symbol == "TP53") %>%
  select(PATIENT_ID) %>%
  mutate(TP53 = TRUE) %>%
  unique()

RUNX1.mutations <- short.mutations %>%
  filter(Hugo_Symbol == "RUNX1") %>%
  select(PATIENT_ID) %>%
  mutate(RUNX1 = TRUE) %>%
  unique()

ASXL1.mutations <- short.mutations %>%
  filter(Hugo_Symbol == "ASXL1") %>%
  select(PATIENT_ID) %>%
  mutate(ASXL1 = TRUE) %>%
  unique()
```

```
#Join karyotype data with lists of samples mutated at each individual gene
tcga.karyo.mutations <- clean.karyotype %>%
  full_join(FLT3.ITD, by = "PATIENT_ID") %>%
  full_join(NPM.mutations, by = "PATIENT_ID") %>%
  full_join(CEBPA.mutations, by = "PATIENT_ID") %>%
  full_join(TP53.mutations, by = "PATIENT_ID") %>%
  full_join(RUNX1.mutations, by = "PATIENT_ID") %>%
  full_join(ASXL1.mutations, by = "PATIENT_ID") %>%
```

```r
  filter(!is.na(CYTOGENETICS)) %>%
  mutate(source = "TCGA")

#Clean NA's into FALSE for mutated fields
#Samples not reported as mutant for a given gene are called as wildtype/unmutated
mutated.fields <- c("NPM1", "FLT3_ITD", "CEBPA",
                    "TP53", "RUNX1", "ASXL1")
mutated.only <- tcga.karyo.mutations[mutated.fields]
mutated.only[is.na(mutated.only)] <- FALSE #Replacing NA values with FALSE
tcga.karyo.mutations[mutated.fields] <- mutated.only

#Making uniform naming and formatting
tcga_complete <- tcga.karyo.mutations %>%
  rename(PDID = PATIENT_ID) %>%
  select(PDID, CYTOGENETICS, FLT3_ITD, NPM1, CEBPA,
         TP53, RUNX1, ASXL1, source)
```

Beat AML dataset

```r
#Load data
raw.baml <- read.csv("./data/BeatAMLdataset.csv")

#Uniform names
baml_complete <- raw.baml %>%
  rename(PDID = lab_id) %>%
  select(-patient_id, -X) %>%
  mutate(source = "BAML")
```

Dataset combination

```r
allsets <- amlsg_complete %>%
  bind_rows(tcga_complete) %>%
  bind_rows(baml_complete)
```

Karyotype parsing

```r
#Karyotype parsing script returns a dataframe of detected abnormalities
source("./Karyotype_parser.R")

allsets.karyo <- allsets$CYTOGENETICS %>%
  karyotype_parse() %>%
  cbind(allsets,.) #Abnormalities are merged into existing dataset
```

Prognostic risk calling

```
#Prognostic risk caller script, returns a column with the assigned prognostic risk
source("./AML_ELNrisk_caller.R")

allsets.karyo$eln_risk <- allsets.karyo %>%
  eln_risk_caller()

write_csv(allsets.karyo, "./Output/AllDatasetsCombined.csv") #Most comprehensive output of parsed data
```

Creating a simple karyotype field

```
#Creating a field for "simple karyotype"
#assigning karotypes to favorable/intermediate/adverse/normal karyotype
simple.risk.withkaryo <- allsets.karyo %>%
  mutate(simplekaryo = as.factor(
         ifelse(PML_RARA | RUNX1_RUNX1T1 | CBFB_MYH11 , "Favorable",
          ifelse(DEK_NUP214 | MLL_rearranged | BCR_ABL |
                   RPN_EVI1_Inv3 | Monosomy_Deletion_5 |
                   Monosomy_7 | Abnormal_17 |
                   complex_karyotype | double_minutes |
                   monosomal_karyotype, "Adverse",
          ifelse(MLLT3_KMT2A | other_abnormalities, "Intermediate",
          ifelse(normal_karyotype, "Normal", "ERROR"
                 )))))) %>%
  select(PDID, CYTOGENETICS, simplekaryo, FLT3_ITD,
         NPM1, CEBPA, TP53, RUNX1, ASXL1, eln_risk, source) %>%
  mutate(pra = TP53 | RUNX1 | ASXL1)

#Smaller dataframe
simple.risk <- simple.risk.withkaryo %>%
  select(-PDID, -CYTOGENETICS)

#Counts for total number of samples and samples from each data source
#Used later for calculating percentage rates
nAll <- simple.risk %>% nrow()
nTCGA <- simple.risk %>% filter(source == "TCGA") %>% nrow()
nBAML <- simple.risk %>% filter(source == "BAML") %>% nrow()
nAMLSG <- simple.risk %>% filter(source == "AMLSG") %>% nrow()
```

Figure 1 - Number of samples from each data source

```
simple.risk %>% nrow() #Total number of samples
```

```
## [1] 1682
```

```
simple.risk %>% count(source) #Samples by data source
```

```
## # A tibble: 3 x 2
##   source     n
```

```
##    <chr> <int>
## 1  AMLSG  1384
## 2   BAML   105
## 3   TCGA   193
```

Figure 2 - Creating summary numbers for prognostic tree breakouts

```
#Basically expanding a combination matrix using for loops
#However to get the data needed to make parent nodes some variables must be kept as NA (not considered)
groups.to.return <- vector()
long.k <- vector()
long.pra <- vector() #Catch-all for TP53, RUNX1, ASXL1 mutations (share the same node in figure)
long.n <- vector()
long.f <- vector()
long.c <- vector()
long.p <- vector()
long.r <- vector()
long.a <- vector()
loopcounter <- 0

##Creates specifications for each requrired node in a series of vectors

for(k in c("Adverse", "Intermediate", "Favorable", "Normal", NA)) {
  for(pra in c(0,1,NA)) {
    for(n in c(0,1,NA)) {
      # End iteration after first NA, which allows us to calculate parent/core nodes
      if(is.na(pra) & !is.na(n)) {next}
      # Make sure we aren't iterating down the tree for PRA mutatnts
      if(pra == 1 & !is.na(n)) {next}

      for(f in c(0,1,NA)) {
        if(is.na(n) & !is.na(f)) {next}

        for(c in c(0,1,NA)) {
          if(is.na(f) & !is.na(c)) {next}
          loopcounter <- loopcounter + 1
          long.k[loopcounter] <- k
          long.pra[loopcounter] <- pra
          long.n[loopcounter] <- n
          long.f[loopcounter] <- f
          long.c[loopcounter] <- c
        }
      }
    }
  }
}

##Creates groups for TP53 RUNX1 ASXL1 venn diagram for figure

loopcounter <- 0
for(p in c(0,1,NA)) {
  for(r in c(0,1,NA)) {
```

```r
    for(a in c(0,1,NA)){
      loopcounter <- loopcounter + 1
      long.p[loopcounter] <- p
      long.r[loopcounter] <- r
      long.a[loopcounter] <- a
    }
  }
}

small.trees <- data.frame("karyotype" = long.k, "pra_mut" = long.pra,
                          "npm1_mut" = long.n,"flt3_itd" = long.f,
                          "cebpa_mut" = long.c)

# Equivalent to expand.grid(p = c(0,1,NA), r = c(0,1,NA), a = c(0,1,NA))
pra.trees <- data.frame("P53_mut" = long.p, "RUNX1_mut" = long.r,
                        "ASXL1" = long.a)
```

```r
#count the number of samples matching each criteria from the total dataset,
#or each individual dataset
all.sources <- vector()
tcga.count <- vector()
baml.count <- vector()
amlsg.count <- vector()

#Iterate through all nodes of figure tree
#e.g. Adverse karyotype, PRA-negative, NPM1-negative, FLT3-ITD-positive
for(r in 1:nrow(small.trees)) {
combo.matching <- simple.risk %>%
  #Filter sample list down to samples matching the tree node specifications
    filter(simplekaryo == small.trees[r,1] | is.na(small.trees[r,1])) %>%
    filter(pra == small.trees[r,2] | is.na(small.trees[r,2])) %>%
    filter(NPM1 == small.trees[r,3] | is.na(small.trees[r,3])) %>%
    filter(FLT3_ITD == small.trees[r,4] | is.na(small.trees[r,4])) %>%
    filter(CEBPA == small.trees[r,5] | is.na(small.trees[r,5]))

  #Count number of matching samples and store
  all.sources[r] <- combo.matching %>%
    nrow()

  #Number of samples from each data source
  tcga.count[r] <- combo.matching %>%
    filter(source == "TCGA") %>% nrow()

  baml.count[r] <- combo.matching %>%
    filter(source == "BAML") %>% nrow()

  amlsg.count[r] <- combo.matching %>%
    filter(source == "AMLSG") %>% nrow()
}

#Combine data into a single dataframe
scored.trees <- cbind(small.trees, all.sources,
                      tcga.count, baml.count, amlsg.count) %>%
  #Create field for percentage rates
```

```r
  mutate(allsources.pct = all.sources / nAll) %>%
  mutate(tcga.pct = tcga.count / nTCGA) %>%
  mutate(baml.pct = baml.count / nBAML) %>%
  mutate(amlsg.pct = amlsg.count / nAMLSG)

## Repeating process for breakout of TP53 RUNX1 ASXL1 status by data source

all.sources <- vector()
tcga.count <- vector()
baml.count <- vector()
amlsg.count <- vector()
for(r in 1:nrow(pra.trees)){
  combo.matching <- simple.risk %>%
    filter(TP53 == pra.trees[r,1] | is.na(pra.trees[r,1])) %>%
    filter(RUNX1 == pra.trees[r,2] | is.na(pra.trees[r,2])) %>%
    filter(ASXL1 == pra.trees[r,3] | is.na(pra.trees[r,3]))

  all.sources[r] <- combo.matching %>%
    nrow()

  tcga.count[r] <- combo.matching %>%
    filter(source == "TCGA") %>% nrow()

  baml.count[r] <- combo.matching %>%
    filter(source == "BAML") %>% nrow()

  amlsg.count[r] <- combo.matching %>%
    filter(source == "AMLSG") %>% nrow()
}

scored.pra.trees <- cbind(pra.trees, all.sources,
                          tcga.count, baml.count, amlsg.count) %>%
  mutate(allsources.pct = all.sources / nAll) %>%
  mutate(tcga.pct = tcga.count / nTCGA) %>%
  mutate(baml.pct = baml.count / nBAML) %>%
  mutate(amlsg.pct = amlsg.count / nAMLSG)
```

Figure 2: Proportion of samples in each node of tree diagram Also Figure S1: Proportion of samples in each node divided by data source

```r
write_csv(scored.trees, "./Output/AllSources_allcombinations_fortrees.csv")
write_csv(scored.pra.trees, "./Output/AllSources_allpra_fortrees.csv")
```

Figure 2: Exceptions to the tree visualization, cases where TP53, RUNX1, ASXL1 mutations are trumped by other marks

Used in writing figure legends or additional explanation

```r
#Cases where samples are determined to be favorable risk even with the presence
#of a TP53, RUNX1, or ASXL1 mutation
simple.risk.withkaryo %>%
  filter(eln_risk == "Favorable") %>%
  filter(pra == TRUE) %>%
  mutate_if(is.logical,as.numeric) %>%
  arrange(eln_risk, simplekaryo, FLT3_ITD, NPM1,
```

```r
          CEBPA, TP53, RUNX1, ASXL1, source)
```

```
##         PDID             CYTOGENETICS simplekaryo FLT3_ITD NPM1 CEBPA TP53
## 1   PD7686a  46,XX,t(8;21)(q22;q22)   Favorable        0    0     0    0
## 2   PD7846a 46,XX,t(15;17)(q22;q21)   Favorable        0    0     0    0
## 3   PD8144a    46,XY,inv(16)(p13q22)   Favorable        0    0     0    0
## 4   PD7990a                   46,XY      Normal        0    1     0    0
## 5   PD8367a                   46,XX      Normal        0    1     0    0
## 6   PD8493a                   46,XY      Normal        0    1     0    0
## 7  15-00990               46,XX[20]      Normal        0    1     0    0
## 8  PD10891a                   46,XY      Normal        0    1     0    0
## 9  PD11101a               46,XY[20]      Normal        0    1     0    0
## 10  PD8040a                   46,XX      Normal        0    1     0    0
##    RUNX1 ASXL1  eln_risk source pra
## 1      0     1 Favorable  AMLSG   1
## 2      0     1 Favorable  AMLSG   1
## 3      0     1 Favorable  AMLSG   1
## 4      0     1 Favorable  AMLSG   1
## 5      0     1 Favorable  AMLSG   1
## 6      0     1 Favorable  AMLSG   1
## 7      0     1 Favorable   BAML   1
## 8      1     0 Favorable  AMLSG   1
## 9      1     0 Favorable  AMLSG   1
## 10     1     0 Favorable  AMLSG   1
```

```r
#TP53 mutations do not co-occur with favorable karyotypes
simple.risk.withkaryo %>%
  filter(TP53 == TRUE) %>%
  filter(simplekaryo != "Adverse") %>%
  mutate_if(is.logical,as.numeric) %>%
  arrange(eln_risk, simplekaryo, FLT3_ITD, NPM1,
          CEBPA, TP53, RUNX1, ASXL1, source)
```

```
##            PDID                                            CYTOGENETICS
## 1      PD11151a                                                47,XX,+11
## 2       PD7867a 46,XY,t(9;11)(p22;q23)[10]/47,XY,+8,t(9;11)(p22;q23)[5]
## 3       PD9270a                              46,XX,t(8;20;21)(q22;q13;q22)
## 4  TCGA-AB-2938                                      45,X,-Y[3]/46,XY [17]
## 5       PD7711a                                                    46,XY
## 6       PD8189a                                                    46,XX
## 7       PD8310a                                                    46,XY
## 8       PD8418a                                                    46,XY
## 9       PD8457a                                                    46,XY
## 10     PD10919a                                                    46,XY
## 11     14-00092                                                46,XX[20]
## 12      PD8083a                                                    46,XX
##     simplekaryo FLT3_ITD NPM1 CEBPA TP53 RUNX1 ASXL1 eln_risk source pra
## 1  Intermediate        0    0     0    1     0     0  Adverse  AMLSG   1
## 2  Intermediate        0    0     0    1     0     0  Adverse  AMLSG   1
## 3  Intermediate        0    0     0    1     0     0  Adverse  AMLSG   1
## 4  Intermediate        0    0     0    1     0     0  Adverse   TCGA   1
## 5        Normal        0    0     0    1     0     0  Adverse  AMLSG   1
## 6        Normal        0    0     0    1     0     0  Adverse  AMLSG   1
## 7        Normal        0    0     0    1     0     0  Adverse  AMLSG   1
## 8        Normal        0    0     0    1     0     0  Adverse  AMLSG   1
```

```
## 9        Normal        0     0     0     1     0     1  Adverse  AMLSG   1
## 10       Normal        0     0     1     1     0     0  Adverse  AMLSG   1
## 11       Normal        0     1     0     1     0     0  Adverse   BAML   1
## 12       Normal        1     1     0     1     0     0  Adverse  AMLSG   1
```

Weighted venn diagram for TP53 RUNX1 ASXL1 overlap with each other

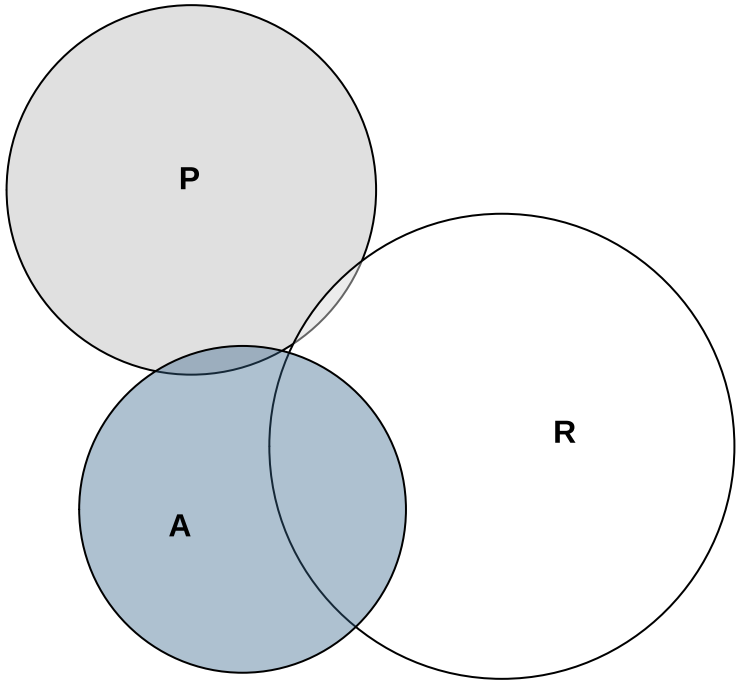
```
pre.venn.df <- scored.pra.trees %>%
  filter(!is.na(P53_mut) & !is.na(RUNX1_mut) & !is.na(ASXL1))

vennReady <- c("P" = pre.venn.df %>% filter(P53_mut == 1 &
                                              RUNX1_mut == 0 & ASXL1 == 0) %>%
                 .$all.sources,
               "R" = pre.venn.df %>% filter(P53_mut == 0 &
                                              RUNX1_mut == 1 & ASXL1 == 0) %>%
                 .$all.sources,
               "A" = pre.venn.df %>% filter(P53_mut == 0 &
                                              RUNX1_mut == 0 & ASXL1 == 1) %>%
                 .$all.sources,
               "P&R" = pre.venn.df %>% filter(P53_mut == 1 &
                                                RUNX1_mut == 1 & ASXL1 == 0) %>%
                 .$all.sources,
               "P&A" = pre.venn.df %>% filter(P53_mut == 1 &
                                                RUNX1_mut == 0 & ASXL1 == 1) %>%
                 .$all.sources,
               "R&A" = pre.venn.df %>% filter(P53_mut == 0 &
                                                RUNX1_mut == 1 & ASXL1 == 1) %>%
                 .$all.sources,
               "P&R&A" = pre.venn.df %>% filter(P53_mut == 1 &
                                                  RUNX1_mut == 1 & ASXL1 == 1) %>%
                 .$all.sources
)

venndiagram <- euler(vennReady)
svg(filename = "./Output/totalvenn.svg")
plot(venndiagram)
dev.off()
```

```
## pdf
##   2
```

```
plot(venndiagram)
```




```
#Residuals indicate whether the venn diagram inaccurately represents proportions
venndiagram
```

```
##       original fitted residuals regionError
## P           98     98         0           0
## R          135    135         0           0
## A           53     53         0           0
## P&R          1      1         0           0
## P&A          2      2         0           0
## R&A         24     24         0           0
## P&R&A        0      0         0           0
##
## diagError: 0
## stress:    0
```

Table and Venn diagram for TP53, RUNX1, or ASXL1 overlap with other adverse marks

```
#Recreating simple karyotype field
pra.cooccurance <- allsets.karyo %>%
  mutate(simplekaryo = as.factor(
           ifelse(PML_RARA | RUNX1_RUNX1T1 | CBFB_MYH11 , "Favorable",
            ifelse(DEK_NUP214 | MLL_rearranged | BCR_ABL | RPN_EVI1_Inv3 |
                     Monosomy_Deletion_5 | Monosomy_7 | Abnormal_17 |
                     complex_karyotype | double_minutes | monosomal_karyotype,
```

```
                   "Adverse",
            ifelse(MLLT3_KMT2A | other_abnormalities, "Intermediate",
            ifelse(normal_karyotype, "Normal", "ERROR"
                   )))))) %>%
  #Field to indicate adverse karyotype not including complex karyotype
    mutate(other_adverse =
           ifelse(RPN_EVI1_Inv3 | Monosomy_Deletion_5 | Monosomy_7 | Abnormal_17 |
                    double_minutes | BCR_ABL | DEK_NUP214 | MLL_rearranged |
                    monosomal_karyotype, 1, 0)) %>%
  #Determines if a sample has non-TP53 adverse marks, including presence of RUNX1 or ASXL1
  mutate(other_adverse_nonP53 =
           ifelse(other_adverse | RUNX1 | ASXL1, 1, 0)) %>%
  mutate(other_adverse_nonRUNX =
           ifelse(other_adverse | TP53 | ASXL1, 1, 0)) %>%
  mutate(other_adverse_nonASXL =
           ifelse(other_abnormalities | TP53 | RUNX1, 1, 0))
```

Figure 2: Table of TP53 RUNX1 ASXL1 mutation by karyotype group

```
#For TP53-mutant samples, count by karyotype category and calculate a percentage rate
p53_co <- pra.cooccurance %>%
  filter(TP53 == 1) %>%
  count(simplekaryo) %>%
  rename(n_TP53 = n) %>%
  mutate(pct_P53 = n_TP53 / sum(n_TP53))

RUNX1_co <- pra.cooccurance %>%
  filter(RUNX1 == 1) %>%
  count(simplekaryo) %>%
  rename(n_RUNX1 = n) %>%
  mutate(pct_RUNX1 = n_RUNX1 / sum(n_RUNX1))

ASXL1_co <- pra.cooccurance %>%
  filter(ASXL1 == 1) %>%
  count(simplekaryo) %>%
  rename(n_ASXL1 = n) %>%
  mutate(pct_ASXL1 = n_ASXL1 / sum(n_ASXL1))

#Join data for TP53, RUNX1, and ASXL1
pra.by.karyo <- p53_co %>%
  full_join(RUNX1_co, by = "simplekaryo") %>%
  full_join(ASXL1_co, by = "simplekaryo")

#Table for figure 2
write_csv(pra.by.karyo, "./Output/PRA_ByKaryotype_Table.csv")
pra.by.karyo
```

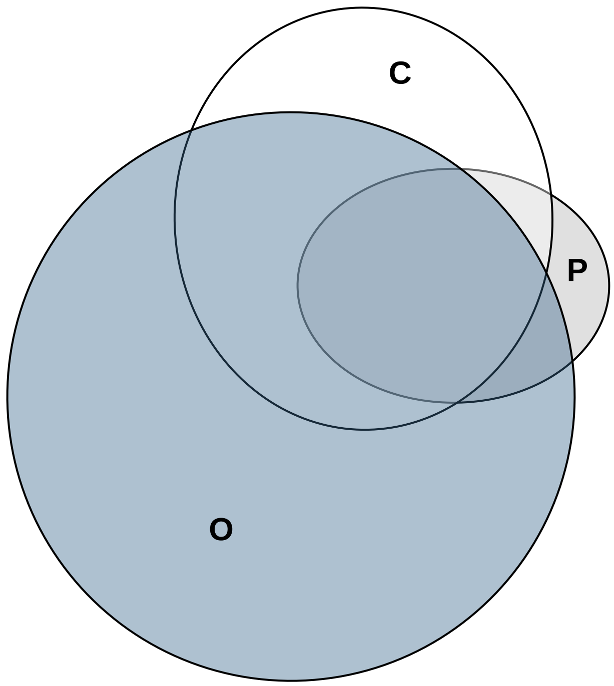
```
## # A tibble: 4 x 7
##    simplekaryo n_TP53    pct_P53 n_RUNX1 pct_RUNX1 n_ASXL1  pct_ASXL1
##         <fctr>  <int>      <dbl>   <int>     <dbl>   <int>      <dbl>
## 1      Adverse     89 0.88118812      37   0.23125      13 0.16455696
## 2 Intermediate      4 0.03960396      43   0.26875      26 0.32911392
## 3       Normal      8 0.07920792      80   0.50000      37 0.46835443
## 4    Favorable     NA         NA      NA        NA       3 0.03797468
```

Figure 2: Venn diagrams of TP53 RUNX1 ASXL1 mutation vs other adverse marks

```
#Creating groups for TP53 venn diagram
dfVenn <- c("P" = pra.cooccurance %>%
              filter(TP53 & !complex_karyotype &  !other_adverse_nonP53) %>%
              nrow(),
            "C" = pra.cooccurance %>%
              filter(!TP53 & complex_karyotype &  !other_adverse_nonP53) %>%
              nrow(),
            "O" = pra.cooccurance %>%
              filter(!TP53 & !complex_karyotype &  other_adverse_nonP53) %>%
              nrow(),
            "P&C" = pra.cooccurance %>%
              filter(TP53 & complex_karyotype &  !other_adverse_nonP53) %>%
              nrow(),
            "P&O" = pra.cooccurance %>%
              filter(TP53 & !complex_karyotype &  other_adverse_nonP53) %>%
              nrow(),
            "C&O" = pra.cooccurance %>%
              filter(!TP53 & complex_karyotype &  other_adverse_nonP53) %>%
              nrow(),
            "P&C&O" = pra.cooccurance %>%
              filter(TP53 & complex_karyotype &  other_adverse_nonP53) %>%
              nrow()
            )
P53venn <- euler(dfVenn, shape = "ellipse")
svg(filename = "./Output/P53venn_ellip.svg")
plot(P53venn)
dev.off()
```

```
## pdf
##   2
```

```
plot(P53venn)
```




```r
#Residuals indicate whether the venn diagram inaccurately represents proportions
P53venn
```

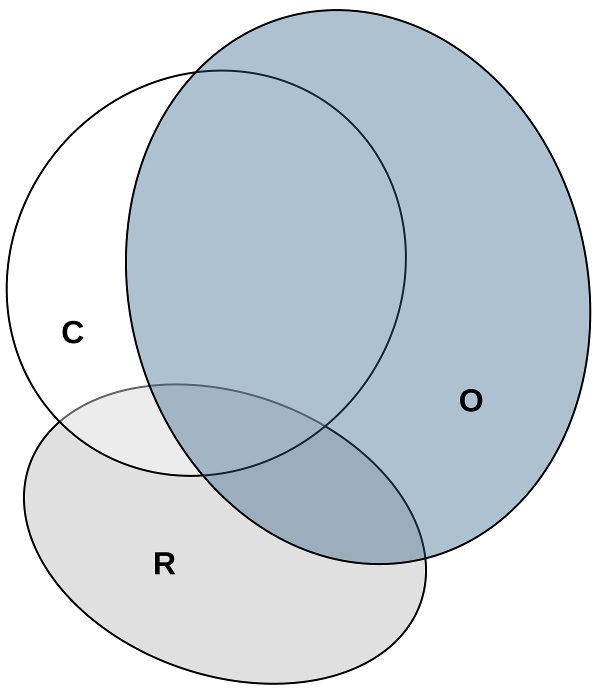
```
##       original  fitted residuals regionError
## P           11  11.002    -0.002           0
## C           58  58.011    -0.011           0
## O          281 281.055    -0.055           0
## P&C          6   6.001    -0.001           0
## P&O          9   9.002    -0.002           0
## C&O         82  82.016    -0.016           0
## P&C&O       75  75.015    -0.015           0
##
## diagError: 0
## stress:    0
```

```r
#Creating groups for RUNX1 venn diagram
dfVenn <- c("R" = pra.cooccurance %>%
              filter(RUNX1 & !complex_karyotype & !other_adverse_nonRUNX) %>%
              nrow(),
            "C" = pra.cooccurance %>%
              filter(!RUNX1 & complex_karyotype & !other_adverse_nonRUNX) %>%
              nrow(),
            "O" = pra.cooccurance %>%
              filter(!RUNX1 & !complex_karyotype & other_adverse_nonRUNX) %>%
              nrow(),
            "R&C" = pra.cooccurance %>%
              filter(RUNX1 & complex_karyotype & !other_adverse_nonRUNX) %>%
```

```r
              nrow(),
            "R&O" = pra.cooccurance %>%
              filter(RUNX1 & !complex_karyotype & other_adverse_nonRUNX) %>%
              nrow(),
            "C&O" = pra.cooccurance %>%
              filter(!RUNX1 & complex_karyotype & other_adverse_nonRUNX) %>%
              nrow(),
            "R&C&O" = pra.cooccurance %>%
              filter(RUNX1 & complex_karyotype & other_adverse_nonRUNX) %>%
              nrow()
            )
RUNX1venn <- euler(dfVenn, shape = "ellipse")
svg(filename = "./Output/RUNX1venn_ellip.svg")
plot(RUNX1venn)
dev.off()
```

```
## pdf
##   2
```

```r
plot(RUNX1venn)
```




```r
#Residuals indicate whether the venn diagram inaccurately represents proportions
RUNX1venn
```

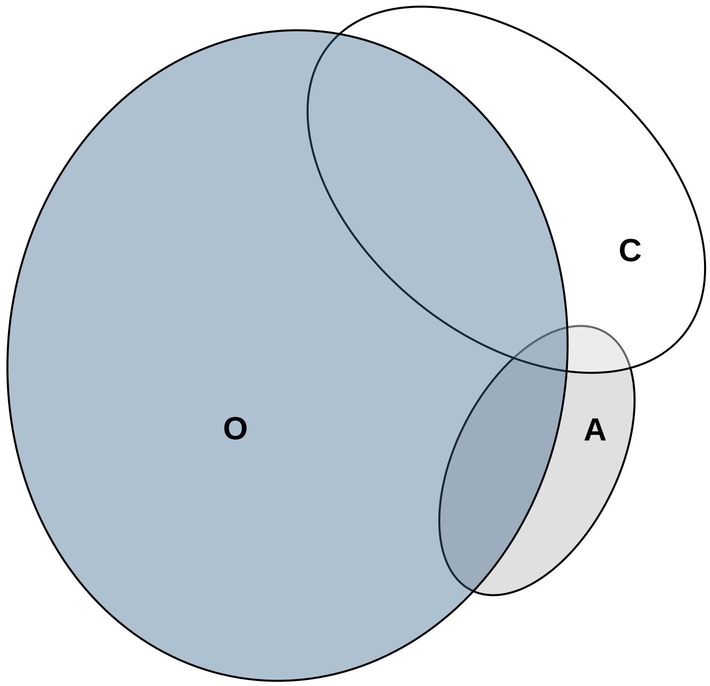
```
##       original  fitted residuals regionError
## R          101 101.738    -0.738           0
## C           58  58.424    -0.424           0
```

```
## O          170 171.243    -1.243           0
## R&C         12  12.088    -0.088           0
## R&O         30  30.219    -0.219           0
## C&O        134 134.980    -0.980           0
## R&C&O       17  17.124    -0.124           0
##
## diagError: 0
## stress:    0
```

```r
#Creating groups for ASXL1 venn diagram
dfVenn <- c("A" = pra.cooccurance %>%
              filter(ASXL1 & !complex_karyotype & !other_adverse_nonASXL) %>%
              nrow(),
            "C" = pra.cooccurance %>%
              filter(!ASXL1 & complex_karyotype & !other_adverse_nonASXL) %>%
              nrow(),
            "O" = pra.cooccurance %>%
              filter(!ASXL1 & !complex_karyotype & other_adverse_nonASXL) %>%
              nrow(),
            "A&C" = pra.cooccurance %>%
              filter(ASXL1 & complex_karyotype & !other_adverse_nonASXL) %>%
              nrow(),
            "A&O" = pra.cooccurance %>%
              filter(ASXL1 & !complex_karyotype & other_adverse_nonASXL) %>%
              nrow(),
            "C&O" = pra.cooccurance %>%
              filter(!ASXL1 & complex_karyotype & other_adverse_nonASXL) %>%
              nrow(),
            "A&C&O" = pra.cooccurance %>%
              filter(ASXL1 & complex_karyotype & other_adverse_nonASXL) %>%
              nrow()
            )
ASXL1venn <- euler(dfVenn, shape = "ellipse")
svg(filename = "./Output/ASXL1venn_ellip.svg")
plot(ASXL1venn)
dev.off()
```

```
## pdf
##   2
```

```r
plot(ASXL1venn)
```




```
#Residuals indicate whether the venn diagram inaccurately represents proportions
ASXL1venn
```

```
##        original  fitted residuals regionError
## A           33  32.489     0.511           0
## C          106 104.359     1.641           0
## O          464 456.816     7.184           0
## A&C          5   4.923     0.077           0
## A&O         38  37.412     0.588           0
## C&O        107 105.343     1.657           0
## A&C&O        3   2.953     0.047           0
##
## diagError: 0
## stress:    0
```

Miscellaneous details used in the manuscript

Manuscript: Each mutation rates as percentage of each karyotype group
For each karyotypic group, shows the percentage of samples that are positive for each given mutation

```
simple.risk %>%
  group_by(simplekaryo) %>%
  mutate(pct_FLT3_ITD = mean(FLT3_ITD)) %>%
  mutate(pct_NPM1 = mean(NPM1)) %>%
  mutate(pct_CEBPA = mean(CEBPA)) %>%
  mutate(pct_TP53 = mean(TP53)) %>%
```

```
  mutate(pct_RUNX1 = mean(RUNX1)) %>%
  mutate(pct_ASXL1 = mean(ASXL1)) %>%
  ungroup() %>%
  select(simplekaryo, contains("pct")) %>%
  unique() %>%
  mutate_if(is.numeric, funs(round(.,digits = 3))) %>%
  arrange(simplekaryo)
```

```
## # A tibble: 4 x 7
##    simplekaryo pct_FLT3_ITD pct_NPM1 pct_CEBPA pct_TP53 pct_RUNX1 pct_ASXL1
##         <fctr>        <dbl>    <dbl>     <dbl>    <dbl>     <dbl>     <dbl>
## 1      Adverse        0.099    0.036     0.018    0.268     0.111     0.039
## 2    Favorable        0.172    0.004     0.009    0.000     0.000     0.013
## 3 Intermediate        0.171    0.168     0.059    0.013     0.141     0.086
## 4       Normal        0.301    0.509     0.076    0.010     0.098     0.045
```

Manuscript: Percentage of CEBPA mutations found in NPM1-wt FLT3-wt samples Subsetted by NPM1 and FLT3-ITD status, shows the percentage of total CEBPA-mutant samples in each group (i.e. 85.2% of all CEBPA mutant samples are wildtype for NPM1 and FLT3-ITD)

```
simple.risk %>%
  filter(CEBPA == TRUE) %>%
  count(NPM1, FLT3_ITD) %>%
  as.matrix() %>% as.data.frame() %>%
  mutate(pct = round(n / sum(n), digits = 3))
```

```
##   NPM1 FLT3_ITD  n   pct
## 1    0        0 75 0.852
## 2    0        1  9 0.102
## 3    1        0  3 0.034
## 4    1        1  1 0.011
```

Figure 4: Sequencing of tests and dealing with missing data

```
# Grid for 128 possible combinations of presence/absence for 7 diagnostic tests
# k - karyotype
# f - FLT3-ITD
# n - NPM1
# c - CEBPA
# r - RUNX1
# p - TP53
# a - ASXL1

possibleTestCombinations <-
  expand.grid(
    k = c(0,1),
    f = c(0,1),
    n = c(0,1),
    c = c(0,1),
    r = c(0,1),
    p = c(0,1),
    a = c(0,1)
  )
```

```
loopcounter <- 0
testNameList <- vector(length = 128)
allriskvariations <- allsets.karyo

#Iterate through all possible test combinations
for(row in 1:nrow(possibleTestCombinations)){
  loopcounter <- loopcounter + 1

  #clear testname variable
  testname <- ""

  #Iterate through each individual test in the combination
  for(col in 1:ncol(possibleTestCombinations)) {
    #If that test is True, add that test to the testname string
    if(possibleTestCombinations[row,col]) {
      testname <- paste0(testname, colnames(possibleTestCombinations)[col])
    }
  }

  #If all tests were set to zero, set name as "NoInfo"
  if(testname == "") {testname <- "NoInfo"}

  #Save testname to vector
  testNameList[loopcounter] <- testname

  tempdf <- allsets.karyo

  #If test combination is negative for k (karyotype)
  #then set all abnormalities to FALSE, set normal to TRUE.
  #This changes the data to a simulation of what would be assumed if
  #no karyotype data was availible
  if(!possibleTestCombinations$k[row]){
    tempdf <- tempdf %>%
        mutate(abnormalities = 0) %>%
        mutate(PML_RARA = 0) %>%
        mutate(RUNX1_RUNX1T1 = 0) %>%
        mutate(CBFB_MYH11 = 0) %>%
        mutate(MLLT3_KMT2A = 0) %>%
        mutate(DEK_NUP214 = 0) %>%
        mutate(MLL_rearranged = 0) %>%
        mutate(BCR_ABL = 0) %>%
        mutate(RPN_EVI1_Inv3 = 0) %>%
        mutate(Monosomy_Deletion_5 = 0) %>%
        mutate(Monosomy_7 = 0) %>%
        mutate(Abnormal_17 = 0) %>%
        mutate(complex_karyotype = 0) %>%
        mutate(double_minutes = 0) %>%
        mutate(monosomal_karyotype = 0) %>%
        mutate(other_abnormalities = 0) %>%
        mutate(normal_karyotype = 1)
  }

  #If specific test is negative, set that mutation field to FALSE
```

```
  if(!possibleTestCombinations$f[row]){
    tempdf <- tempdf %>%
      mutate(FLT3_ITD = FALSE)
  }
  if(!possibleTestCombinations$n[row]){
    tempdf <- tempdf %>%
      mutate(NPM1 = FALSE)
  }
  if(!possibleTestCombinations$c[row]){
    tempdf <- tempdf %>%
      mutate(CEBPA = FALSE)
  }
  if(!possibleTestCombinations$p[row]){
    tempdf <- tempdf %>%
      mutate(TP53 = FALSE)
  }
  if(!possibleTestCombinations$r[row]){
    tempdf <- tempdf %>%
      mutate(RUNX1 = FALSE)
  }
  if(!possibleTestCombinations$a[row]){
    tempdf <- tempdf %>%
      mutate(ASXL1 = FALSE)
  }

  #Calculate ELN risk given the artificially censored data
  temprisk <- eln_risk_caller(tempdf)

  #Save output risk calls under a column for the test series
  riskonly <- data.frame(result = temprisk)
  colnames(riskonly) <- testname

  #Join with earlier data
  allriskvariations <- cbind(allriskvariations, riskonly)
}

#Grab only columns with true risk calls ("eln_risk") and simulated risk calls
allriskonly <- allriskvariations %>%
  select(eln_risk:kfncrpa)
```

Remove one testing / partial information accuracy

Determine which type of errors are present under cases of limited information

```
#Create a dictionary to join into data frame
riskcalldictionary <- data.frame(concatenated_risk =
                                   c("Favorable Favorable",
                                     "Favorable Intermediate",
                                     "Favorable Adverse",
                                     "Intermediate Favorable",
                                     "Intermediate Intermediate",
                                     "Intermediate Adverse",
                                     "Adverse Favorable",
```

```r
                                     "Adverse Intermediate",
                                     "Adverse Adverse"),
                                 call_type = c("True_Favorable",
                                               "Favorable_called_Intermediate",
                                               "Favorable_called_Adverse",
                                               "Intermediate_called_Favorable",
                                               "True_Intermediate",
                                               "Intermediate_called_Adverse",
                                               "Adverse_called_Favorable",
                                               "Adverse_called_Intermediate",
                                               "True_Adverse"))
temp_riskcomparison <- allriskonly

#Establish output dataframe
callsFromLimitedInfo <- data.frame(tests = factor(),
                                   True_Favorable = int(),
                                   Favorable_called_Intermediate = int(),
                                   Favorable_called_Adverse = int(),
                                   Intermediate_called_Favorable = int(),
                                   True_Intermediate = int(),
                                   Intermediate_called_Adverse = int(),
                                   Adverse_called_Favorable = int(),
                                   Adverse_called_Intermediate = int(),
                                   True_Adverse = int())

#Iterate through risk calls from each test combination
for(col in 1:ncol(allriskonly)) {
  #paste the true risk call: allriskonly[,1]
  #with the artificial risk call
  callresults <- data.frame("concatenated_risk" =
               paste(allriskonly[,1], allriskonly[,col], sep = " ")) %>%
    #Join with dictionary above
    left_join(riskcalldictionary, by = "concatenated_risk") %>%
    count(call_type) %>% #Count number of each error type
    spread(key = call_type, value = n) %>%  #Pivot data
    mutate_all(funs(round(./1682, digits = 3))) #Calculate percentage rate

    #Create row of new data
  test_row <- cbind(data.frame(tests = colnames(allriskonly[col])), callresults)

  #Join with existing data
  callsFromLimitedInfo <- bind_rows(callsFromLimitedInfo, test_row)
}

#Create a CSV to allow filtering/exploration in Excel
#Filter based on presence/absence of each test, determine accuracy and types of errors
callsForCSVexport <- callsFromLimitedInfo %>%
  filter(tests != "eln_risk") %>%
  mutate(tests =
           ifelse(tests == "NoInfo", "", tests)) %>%
  mutate(k = str_detect(tests, "k")) %>%
  mutate(f = str_detect(tests, "f")) %>%
  mutate(n = str_detect(tests, "n")) %>%
```

```
  mutate(c = str_detect(tests, "c")) %>%
  mutate(p = str_detect(tests, "p")) %>%
  mutate(r = str_detect(tests, "r")) %>%
  mutate(a = str_detect(tests, "a")) %>%
  mutate_if(is.logical,as.numeric) %>%
  select(k,f,n,c,p,r,a,everything()) %>%
  select(-tests) %>%
  rowwise() %>%
  mutate(Number_of_tests = sum(k,f,n,c,p,r,a)) %>%
  arrange(desc(Number_of_tests))

#Replace NA values with zeroes
callsForCSVexport[is.na(callsForCSVexport)] <- 0

write_csv(callsForCSVexport, "./Output/CallTypesAndErrors_MissingTestCombos.csv")
```

```
#Determine accuracy or balanced accuracy for each test combination
#Also calculate accuracy for identifying samples relative to a single category
#e.g. Is a sample Intermediate or Not-Intermediate?

#Establish vectors
confusiondf <- data.frame()
loopcounter <- 0
totalACC <- vector()
favACC <- vector()
intACC <- vector()
advACC <- vector()
testname <- vector()

for(i in 2:ncol(allriskonly)) {
  loopcounter <- loopcounter + 1

  cfm <- confusionMatrix(allriskonly[,i], allriskonly$eln_risk)
  #pull out specific accuracy measures from confusionMatrix
  totalACC[loopcounter] <- cfm$overall[[1]]
  favACC[loopcounter] <- cfm$byClass[1,11]
  intACC[loopcounter] <- cfm$byClass[3,11]
  advACC[loopcounter] <- cfm$byClass[2,11]
  testname[loopcounter] <- colnames(allriskonly[,c(1,i)])[[2]]
}

testacc <- data.frame(tests = testname, totalACC = totalACC,
                      f_bacc = favACC, i_bacc = intACC, a_bacc = advACC)

#Create a field for number of tests included in a combination
ordered.acc <- testacc %>%
  mutate(testnum =
           ifelse(tests == "NoInfo", 0,
                  str_count(tests, "[:alpha:]"))) %>%
  arrange(testnum, desc(totalACC))
```

Optimal sequential ordering of tests for maximal accuracy

```
#Breaks strings into individual letters, sorts alphabetically, puts back into a string
string_sort <- function(x) {
  y <- paste(sort(unlist(str_split(x, ""))), collapse = "")
  return(y)
}
```

```
#Creates a row where tests are a single alphabetical string
alpha.acc <- ordered.acc %>%
  rowwise() %>%
  mutate(alphatests = string_sort(tests)) %>%
  ungroup() %>%
  as.data.frame()
```

```
#Creates a list of every possible permutation of tests as a single, ordered string
#5,040 total permutations
fullseries <- c("k","f","n","c","p","r","a")
loopcount <- 0
sequence <- vector()
for(l_one in fullseries) {
  twoseries <- fullseries[fullseries!=l_one]
  for(l_two in twoseries) {
    threeseries <- twoseries[twoseries!=l_two]
    for(l_three in threeseries) {
      fourseries <- threeseries[threeseries!=l_three]
      for(l_four in fourseries) {
        fiveseries <- fourseries[fourseries!=l_four]
        for(l_five in fiveseries) {
          sixseries <- fiveseries[fiveseries!=l_five]
          for(l_six in sixseries) {
            l_seven <- sixseries[sixseries!=l_six]
            loopcount <- loopcount + 1
            sequence[loopcount] <- paste(l_one, l_two,
                                         l_three, l_four,
                                         l_five, l_six,
                                         l_seven, collapse = "", sep="")
          }
        }
      }
    }
  }
}
```

The chunk below tests all 5,040 possible permutations of the seven diagnostic tests, returning the ELN risk categorization accuracy for each test in order.

The chunk is repeated three more times, to determine the same information when only considering accuracy for classifying in regards to a single prognostic group (i.e. correctly called Favorable or non-Favorable)

```
#Total accuracy

#Establish vectors
t_one <- vector()
t_two <- vector()
t_three <- vector()
```

```r
t_four <- vector()
t_five <- vector()
t_six <- vector()
t_seven <- vector()
p1 <- vector()
p2 <- vector()
p3 <- vector()
p4 <- vector()
p5 <- vector()
p6 <- vector()
p7 <- vector()
loopcount <- 0

#Iterate through 5,040 permutations established above
for(i in sequence){
  loopcount <- loopcount + 1

  #Extract tests in order (test one, test two)
  t_one <- string_sort(substr(i,1,1)) #e.g. k
  t_two <- string_sort(substr(i,1,2)) #e.g. kp
  t_three <- string_sort(substr(i,1,3)) #e.g. kpf
  t_four <- string_sort(substr(i,1,4)) #e.g. kpfc
  t_five <- string_sort(substr(i,1,5)) #e.g. kpfcn
  t_six <- string_sort(substr(i,1,6)) #e.g. kpfcnr
  t_seven <- string_sort(substr(i,1,7)) #e.g. kpfcnra

  #extract accuracy from matching record
  p1[loopcount] <- alpha.acc %>%
    filter(alphatests == t_one) %>%
    .$totalACC
  p2[loopcount] <- alpha.acc %>%
    filter(alphatests == t_two) %>%
    .$totalACC
  p3[loopcount] <- alpha.acc %>%
    filter(alphatests == t_three) %>%
    .$totalACC
  p4[loopcount] <- alpha.acc %>%
    filter(alphatests == t_four) %>%
    .$totalACC
  p5[loopcount] <- alpha.acc %>%
    filter(alphatests == t_five) %>%
    .$totalACC
  p6[loopcount] <- alpha.acc %>%
    filter(alphatests == t_six) %>%
    .$totalACC
  p7[loopcount] <- alpha.acc %>%
    filter(alphatests == t_seven) %>%
    .$totalACC
}

#assemble results
everysequence <- data.frame(ordered_tests = sequence, t1 = p1,
                            t2 = p2, t3 = p3, t4 = p4, t5 = p5,
```

```
                            t6 = p6, t7 = p7)

head(everysequence)
```

```
##   ordered_tests        t1        t2        t3       t4        t5        t6
## 1       kfncpra 0.7241379 0.7241379 0.8703924 0.901308 0.9084423 0.9797860
## 2       kfncpar 0.7241379 0.7241379 0.8703924 0.901308 0.9084423 0.9417360
## 3       kfncrpa 0.7241379 0.7241379 0.8703924 0.901308 0.9726516 0.9797860
## 4       kfncrap 0.7241379 0.7241379 0.8703924 0.901308 0.9726516 0.9934602
## 5       kfncapr 0.7241379 0.7241379 0.8703924 0.901308 0.9351962 0.9417360
## 6       kfncarp 0.7241379 0.7241379 0.8703924 0.901308 0.9351962 0.9934602
##   t7
## 1  1
## 2  1
## 3  1
## 4  1
## 5  1
## 6  1
```

```
#Pivot results to long format (used for graphs below)
longsequence <- everysequence %>%
  gather(key = TestInSeq, value = TotalAcc, -ordered_tests) %>%
  mutate(TestInSeq = as.numeric(
           substr(TestInSeq,2,2)))
```

```
## favorable accuracy
t_one <- vector()
t_two <- vector()
t_three <- vector()
t_four <- vector()
t_five <- vector()
t_six <- vector()
t_seven <- vector()
p1 <- vector()
p2 <- vector()
p3 <- vector()
p4 <- vector()
p5 <- vector()
p6 <- vector()
p7 <- vector()
loopcount <- 0

for(i in sequence){
  loopcount <- loopcount + 1
  t_one <- string_sort(substr(i,1,1))
  t_two <- string_sort(substr(i,1,2))
  t_three <- string_sort(substr(i,1,3))
  t_four <- string_sort(substr(i,1,4))
  t_five <- string_sort(substr(i,1,5))
  t_six <- string_sort(substr(i,1,6))
  t_seven <- string_sort(substr(i,1,7))

  p1[loopcount] <- alpha.acc %>%
    filter(alphatests == t_one) %>%
```

```
    .$f_bacc
  p2[loopcount] <- alpha.acc %>%
    filter(alphatests == t_two) %>%
    .$f_bacc
  p3[loopcount] <- alpha.acc %>%
    filter(alphatests == t_three) %>%
    .$f_bacc
  p4[loopcount] <- alpha.acc %>%
    filter(alphatests == t_four) %>%
    .$f_bacc
  p5[loopcount] <- alpha.acc %>%
    filter(alphatests == t_five) %>%
    .$f_bacc
  p6[loopcount] <- alpha.acc %>%
    filter(alphatests == t_six) %>%
    .$f_bacc
  p7[loopcount] <- alpha.acc %>%
    filter(alphatests == t_seven) %>%
    .$f_bacc
}

facc_sequence <- data.frame(ordered_tests = sequence, f1 = p1,
                            f2 = p2, f3 = p3, f4 = p4, f5 = p5,
                            f6 = p6, f7 = p7)

facc_long <- facc_sequence %>%
  gather(key = TestInSeq, value = f_bacc, -ordered_tests) %>%
  mutate(TestInSeq = as.numeric(
         substr(TestInSeq,2,2)))
```

```
## intermediate ACC
t_one <- vector()
t_two <- vector()
t_three <- vector()
t_four <- vector()
t_five <- vector()
t_six <- vector()
t_seven <- vector()
p1 <- vector()
p2 <- vector()
p3 <- vector()
p4 <- vector()
p5 <- vector()
p6 <- vector()
p7 <- vector()
loopcount <- 0

for(i in sequence){
  loopcount <- loopcount + 1
  t_one <- string_sort(substr(i,1,1))
  t_two <- string_sort(substr(i,1,2))
  t_three <- string_sort(substr(i,1,3))
  t_four <- string_sort(substr(i,1,4))
  t_five <- string_sort(substr(i,1,5))
```

```
  t_six <- string_sort(substr(i,1,6))
  t_seven <- string_sort(substr(i,1,7))

  p1[loopcount] <- alpha.acc %>%
    filter(alphatests == t_one) %>%
    .$i_bacc
  p2[loopcount] <- alpha.acc %>%
    filter(alphatests == t_two) %>%
    .$i_bacc
  p3[loopcount] <- alpha.acc %>%
    filter(alphatests == t_three) %>%
    .$i_bacc
  p4[loopcount] <- alpha.acc %>%
    filter(alphatests == t_four) %>%
    .$i_bacc
  p5[loopcount] <- alpha.acc %>%
    filter(alphatests == t_five) %>%
    .$i_bacc
  p6[loopcount] <- alpha.acc %>%
    filter(alphatests == t_six) %>%
    .$i_bacc
  p7[loopcount] <- alpha.acc %>%
    filter(alphatests == t_seven) %>%
    .$i_bacc
}

iacc_sequence <- data.frame(ordered_tests = sequence, i1 = p1,
                            i2 = p2, i3 = p3, i4 = p4, i5 = p5,
                            i6 = p6, i7 = p7)

iacc_long <- iacc_sequence %>%
  gather(key = TestInSeq, value = i_bacc, -ordered_tests) %>%
  mutate(TestInSeq = as.numeric(
           substr(TestInSeq,2,2)))
```

```
# adverse Acc

t_one <- vector()
t_two <- vector()
t_three <- vector()
t_four <- vector()
t_five <- vector()
t_six <- vector()
t_seven <- vector()
p1 <- vector()
p2 <- vector()
p3 <- vector()
p4 <- vector()
p5 <- vector()
p6 <- vector()
p7 <- vector()
loopcount <- 0

for(i in sequence){
```

```
  loopcount <- loopcount + 1
  t_one <- string_sort(substr(i,1,1))
  t_two <- string_sort(substr(i,1,2))
  t_three <- string_sort(substr(i,1,3))
  t_four <- string_sort(substr(i,1,4))
  t_five <- string_sort(substr(i,1,5))
  t_six <- string_sort(substr(i,1,6))
  t_seven <- string_sort(substr(i,1,7))

  p1[loopcount] <- alpha.acc %>%
    filter(alphatests == t_one) %>%
    .$a_bacc
  p2[loopcount] <- alpha.acc %>%
    filter(alphatests == t_two) %>%
    .$a_bacc
  p3[loopcount] <- alpha.acc %>%
    filter(alphatests == t_three) %>%
    .$a_bacc
  p4[loopcount] <- alpha.acc %>%
    filter(alphatests == t_four) %>%
    .$a_bacc
  p5[loopcount] <- alpha.acc %>%
    filter(alphatests == t_five) %>%
    .$a_bacc
  p6[loopcount] <- alpha.acc %>%
    filter(alphatests == t_six) %>%
    .$a_bacc
  p7[loopcount] <- alpha.acc %>%
    filter(alphatests == t_seven) %>%
    .$a_bacc
}

aacc_sequence <- data.frame(ordered_tests = sequence, a1 = p1,
                            a2 = p2, a3 = p3, a4 = p4, a5 = p5,
                            a6 = p6, a7 = p7)

aacc_long <- aacc_sequence %>%
  gather(key = TestInSeq, value = a_bacc, -ordered_tests) %>%
  mutate(TestInSeq = as.numeric(
           substr(TestInSeq,2,2)))
```

Assemble data from chunks above

```
four_sequence <- left_join(everysequence, facc_sequence, by = "ordered_tests") %>%
  left_join(iacc_sequence, by = "ordered_tests") %>%
  left_join(aacc_sequence, by = "ordered_tests")

four_long <- left_join(longsequence, facc_long, by = c("ordered_tests", "TestInSeq")) %>%
  left_join(iacc_long, by = c("ordered_tests", "TestInSeq")) %>%
  left_join(aacc_long, by = c("ordered_tests", "TestInSeq"))

#save data in wide and long format as CSVs
write_csv(four_sequence, "./Output/AllSequencesWide.csv")
write_csv(four_long, "./Output/AllSequencesLong.csv")
```

Figure 4 Charts: Optimal sequencing of tests

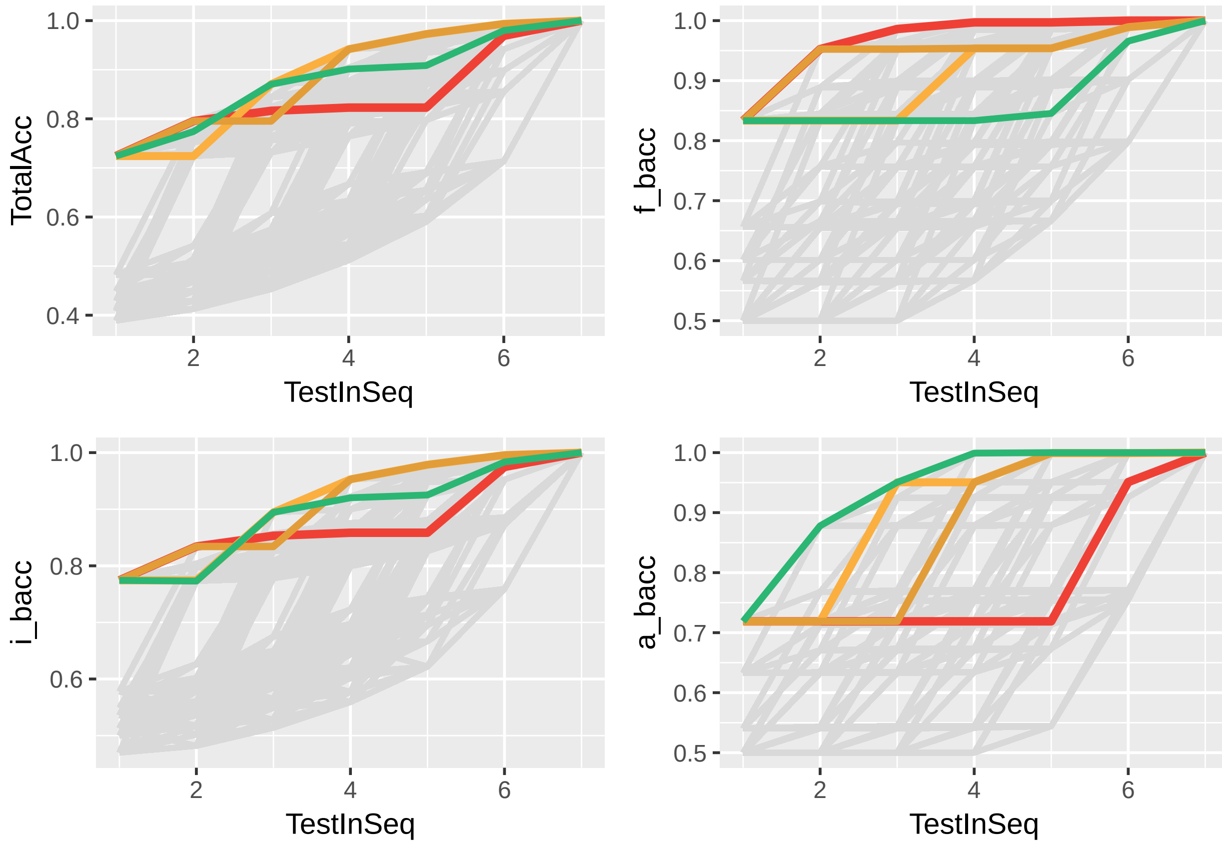
```r
c1 <- ggplot(four_long, aes(x=TestInSeq, y=TotalAcc, group=ordered_tests)) +
  geom_line(size = 1, color = "grey85") +
  geom_line(data = filter(four_long, ordered_tests == "krapfnc"),
            color = "#ef4036", size = 1.5) +
  geom_line(data = filter(four_long, ordered_tests == "kfnrcap"),
            color = "#fbaf3f", size = 1.4) +
  geom_line(data = filter(four_long, ordered_tests == "krfncap"),
            color = "#E29D39", size = 1.3) +
  geom_line(data = filter(four_long, ordered_tests == "knfcpra"),
            color = "#2bb673", size = 1.2)

c2 <- ggplot(four_long, aes(x=TestInSeq, y=f_bacc, group=ordered_tests)) +
  geom_line(size = 1, color = "grey85") +
  geom_line(data = filter(four_long, ordered_tests == "krapfnc"),
            color = "#ef4036", size = 1.5) +
  geom_line(data = filter(four_long, ordered_tests == "kfnrcap"),
            color = "#fbaf3f", size = 1.4) +
  geom_line(data = filter(four_long, ordered_tests == "krfncap"),
            color = "#E29D39", size = 1.3) +
  geom_line(data = filter(four_long, ordered_tests == "knfcpra"),
            color = "#2bb673", size = 1.2)

c3 <- ggplot(four_long, aes(x=TestInSeq, y=i_bacc, group=ordered_tests)) +
  geom_line(size = 1, color = "grey85") +
  geom_line(data = filter(four_long, ordered_tests == "krapfnc"),
            color = "#ef4036", size = 1.5) +
  geom_line(data = filter(four_long, ordered_tests == "kfnrcap"),
            color = "#fbaf3f", size = 1.4) +
  geom_line(data = filter(four_long, ordered_tests == "krfncap"),
            color = "#E29D39", size = 1.3) +
  geom_line(data = filter(four_long, ordered_tests == "knfcpra"),
            color = "#2bb673", size = 1.2)

c4 <- ggplot(four_long, aes(x=TestInSeq, y=a_bacc, group=ordered_tests)) +
  geom_line(size = 1, color = "grey85") +
  geom_line(data = filter(four_long, ordered_tests == "krapfnc"),
            color = "#ef4036", size = 1.5) +
  geom_line(data = filter(four_long, ordered_tests == "kfnrcap"),
            color = "#fbaf3f", size = 1.4) +
  geom_line(data = filter(four_long, ordered_tests == "krfncap"),
            color = "#E29D39", size = 1.3) +
  geom_line(data = filter(four_long, ordered_tests == "knfcpra"),
            color = "#2bb673", size = 1.2)

grid.arrange(c1,c2,c3,c4)
```




```r
coloredgraphs <- arrangeGrob(c1,c2,c3,c4)



ggsave("./Output/SequencingLinesColored.png", coloredgraphs, height = 20, width = 24)
ggsave("./Output/SequencingLinesColored.svg", coloredgraphs, height = 20, width = 24)
```