Further Exploration

Miles Rowbottom

Loading in Data

Load in the dataset:

Create a Table with FACS and Proliferation Score for Each Genotype

Extract the key columns for this initial exploration:

```
whole_df <- data.frame (
  genotype = sce_genotyped$GT,
  facs_score = sce_genotyped$facs_z_score_edited,
  proliferation_score = sce_genotyped$prolif_z_score_edited
)
```

Begin Building a Genotype Key Table

```
unique_genotypes <- unique(whole_df$genotype)

# There are NA values across the board for WT, and genotypes at index 21 and 56
unique_genotypes <- unique_genotypes[-c(1, 21, 56)]
```

There are duplicate FACS and proliferation scores for two genotypes. Create duplicates of those:

```
facs_score <- c()
proliferation_score <- c()

# Extract the first FACS and proliferation scores for each unique genotype
for (i in 1:length(unique_genotypes)) {
  facs_score <- c(
    facs_score,
    whole_df$facs_score[whole_df$genotype == unique_genotypes[i]][1]
  )

  proliferation_score <- c(
    proliferation_score,
    whole_df$proliferation_score[whole_df$genotype == unique_genotypes[i]][1]
  )
}

# Manually append certain unique genotype alternate names
unique_genotypes <- c(
  unique_genotypes,
  paste0(unique_genotypes[11], "_(1)"),
  paste0(unique_genotypes[18], "_(1)")
)
```

```r
# Manually append certain unique genotype alternate FACS scores
facs_score <- c(
  facs_score,
  whole_df$facs_score[whole_df$genotype == unique_genotypes[11]][3],
  whole_df$facs_score[whole_df$genotype == unique_genotypes[18]][8]
)

# Manually append certain unique genotype alternate proliferation scores
proliferation_score <- c(
  proliferation_score,
  whole_df$proliferation_score[whole_df$genotype == unique_genotypes[11]][3],
  whole_df$proliferation_score[whole_df$genotype == unique_genotypes[18]][8]
)
```

We can now place these three vectors into a single data frame:

```r
genotype_df <- data.frame(unique_genotypes, facs_score, proliferation_score)
```

Clustering

```r
library(ggplot2)
library(ggthemes)
library(tidyverse)
```

Linear Regression

Our data is *not* linear. Biologically, we might expect a logarithmic or potentially sigmoidal curve - above a certain level of proteins, increase in cell proliferation slows. It is also expected that at low protein levels (low FACS scores), small increases in FACS have a large effect on proliferation, as it begins to meet the threshold for pathway activation.

However, a linear model does provide an effective visualisation of the intermediate values that lie above what the linear relationship *might* be. We plot the 95% confidence interval for this relationship, so that we can see the points whose position above the linear relationship is statistically significant.

```r
model_lm <- lm(proliferation_score ~ facs_score, data = genotype_df)

dense_x <- seq(-2.6, 21.6, length.out = 500)

dense_predictions <- predict(model_lm, newdata = data.frame(facs_score = dense_x),
                        interval = "confidence", level = 0.95)
```

```r
summary(model_lm)
```

```
##
## Call:
## lm(formula = proliferation_score ~ facs_score, data = genotype_df)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -4.355 -1.885 -1.106  1.030  9.444
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.49939    0.51113   2.933  0.00428 **
```

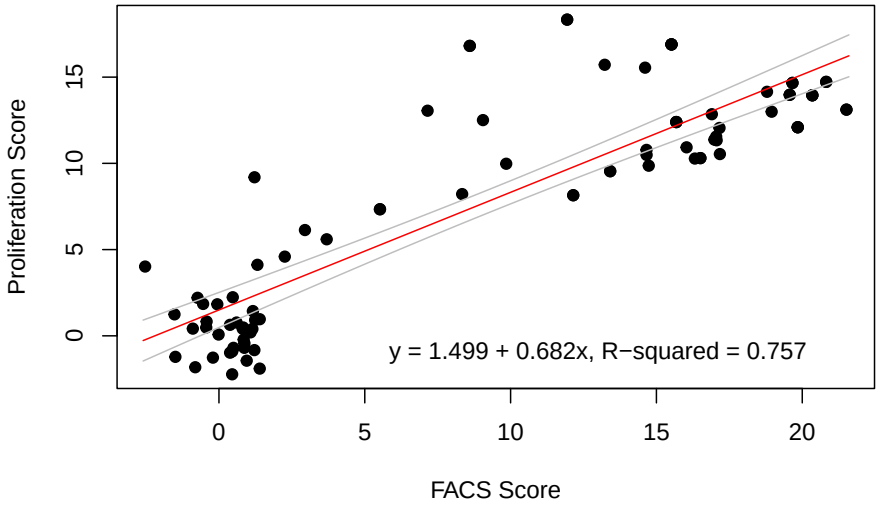
```
## facs_score   0.68222    0.04146  16.456  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.171 on 87 degrees of freedom
## Multiple R-squared:  0.7568, Adjusted R-squared:  0.754
## F-statistic: 270.8 on 1 and 87 DF,  p-value: < 2.2e-16
```

```
AIC(model_lm)
```

```
## [1] 461.9549
```

We have a multiple R-squared value of `0.7568`, and an equation of $y = 1.499 + 0.682x$. The AIC for our model is `461.95`.

```
plot(genotype_df$facs_score, genotype_df$proliferation_score, pch = 19,
     xlab = "FACS Score", ylab = "Proliferation Score")
lines(dense_x, dense_predictions[, "fit"], col = "red")
lines(dense_x, dense_predictions[, "lwr"], col = "grey")
lines(dense_x, dense_predictions[, "upr"], col = "grey")
text(x = 13, y = -1, labels = "y = 1.499 + 0.682x, R-squared = 0.757")
```



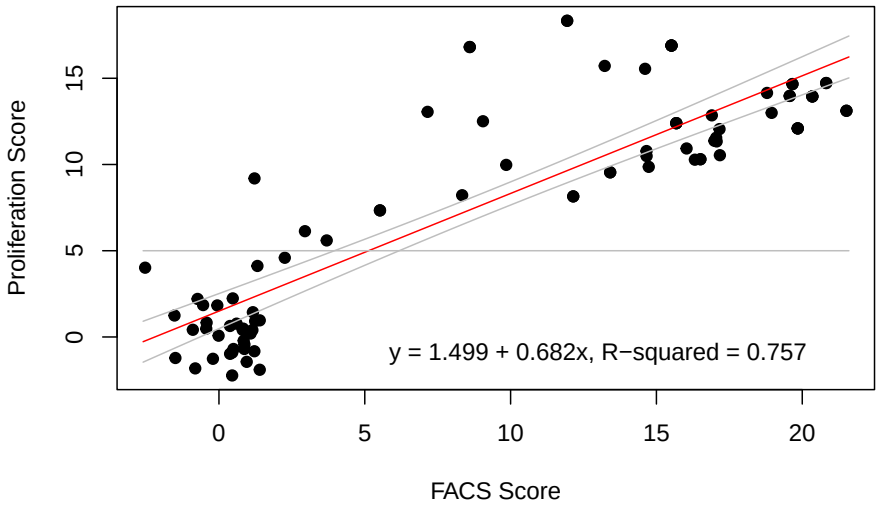
This splits our data quite nicely. Where it isn't perfect is towards the bottom left where it is arguably capturing points that belong to the low proliferation-low FACS cluster.

We can perhaps also set a minimum proliferation score threshold to separate these points. I have no way of contextually knowing a good value to select here, but `5` appears to be an appropriate cut-off:

```
minimum_proliferation <- 5

plot(genotype_df$facs_score, genotype_df$proliferation_score, pch = 19,
     xlab = "FACS Score", ylab = "Proliferation Score")
lines(dense_x, dense_predictions[, "fit"], col = "red")
lines(dense_x, dense_predictions[, "lwr"], col = "grey")
lines(dense_x, dense_predictions[, "upr"], col = "grey")
text(x = 13, y = -1, labels = "y = 1.499 + 0.682x, R-squared = 0.757")
```

```r
lines(dense_x, rep(minimum_proliferation, times = 500), col = "grey")
```



Custom Clustering

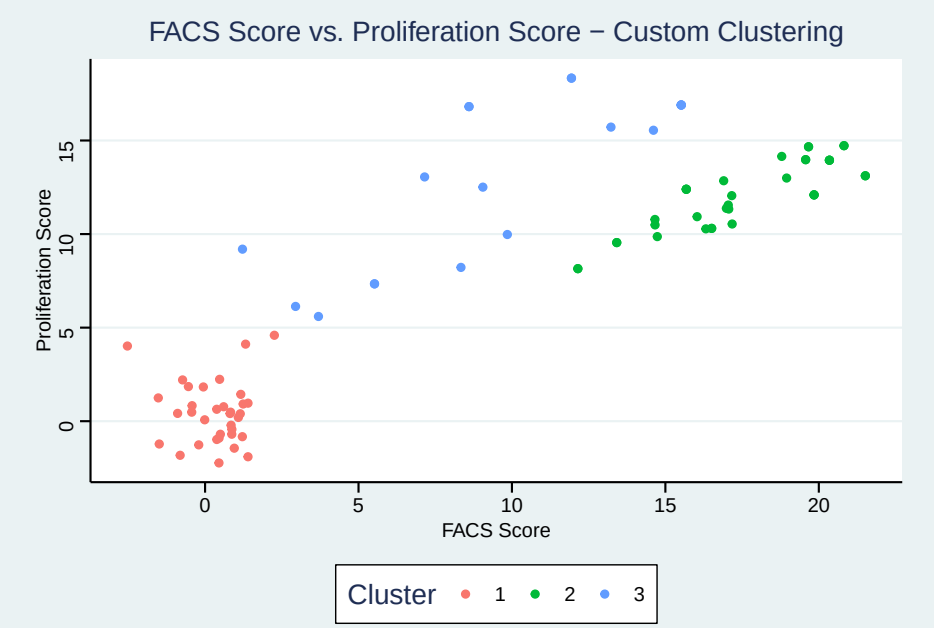
Let's now take our ideas from the linear regression and allocate clusters appropriately.

```r
predictions <- predict(model_lm,
                       newdata = data.frame(facs_score = genotype_df$facs_score),
                       interval = "confidence", level = 0.95)

custom_clusters <- c()
for (i in 1:length(genotype_df$proliferation_score)) {
  # If proliferation above the upper 95% bound and minimum threshold: cluster 3
  if (genotype_df$proliferation_score[i] > predictions[, "upr"][i] &&
      genotype_df$proliferation_score[i] > minimum_proliferation) {
    custom_clusters <- c(custom_clusters, 3)
  # Else, if below the minimum threshold: cluster 1
  } else if (genotype_df$proliferation_score[i] < minimum_proliferation) {
    custom_clusters <- c(custom_clusters, 1)
  # Else, if above the minimum threshold: cluster 2
  } else {
    custom_clusters <- c(custom_clusters, 2)
  }
}

genotype_df <- cbind(genotype_df, custom_clusters)
```

```r
plot <- ggplot(data = genotype_df, aes(x = facs_score, y = proliferation_score,
                                       color = as.factor(custom_clusters)))
plot <- plot + theme_stata()
plot <- plot + labs(x = "FACS Score", y = "Proliferation Score", color = "Cluster",
                    title = "FACS Score vs. Proliferation Score - Custom Clustering")
plot + geom_point()
```


FACS Score vs. Proliferation Score − Custom Clustering
Proliferation Score
0
5
10
15
0
5
10
15
20
FACS Score
Cluster
1
2
3

Exporting to Python via Feather

Note that all of the following code is not ran by the markdown (to prevent the operations from occurring).

```
library(arrow)
```

Start by exporting the FACS and proliferation scores key table:

```
write_feather(genotype_df, "genotype_df.feather")
```

And then moving on to export counts data:

```
counts_matrix <- read.csv("logcounts_scSNVseq_JAK1_cc_JAK_STAT.csv", row.names = 1, check.names = FALSE)

counts_df <- as.data.frame(t(counts_matrix))

# Extract full list of each cell's genotype
GT <- sce_genotyped$GT

# The cells with either of the two genotypes with different scores need to be
# separated and relabelled
Gene1_values <- whole_df$facs_score[whole_df$genotype == unique_genotypes[11]]
Gene2_values <- whole_df$facs_score[whole_df$genotype == unique_genotypes[18]]
Gene1_first_value <- Gene1_values[1]
Gene2_first_value <- Gene2_values[1]

Gene1_count <- 0
Gene2_count <- 0

for (i in 1:length(GT)) {
  if (GT[i] == "64850911-1/1/1-C-T") {
    Gene1_count <- Gene1_count + 1
    if (Gene1_values[Gene1_count] == Gene1_first_value) {
      print("Gene1: Not changed.")
      next
    } else {
      GT[i] <- "64850911-1/1/1-C-T_(1)"
      print("Gene1: Changed.")
    }
  }

  if (GT[i] == "64839784-1/1/1-C-T;64839785-1/1/1-C-T;64839786-1/1/1-C-T") {
    Gene2_count <- Gene2_count + 1
    if (Gene2_values[Gene2_count] == Gene2_first_value) {
      print("Gene2: Not changed.")
      next
    } else {
      GT[i] <- "64839784-1/1/1-C-T;64839785-1/1/1-C-T;64839786-1/1/1-C-T_(1)"
      print("Gene1: Changed.")
    }
  }
}

# Attach each cell's genotype to the counts dataframe
counts_df <- data.frame(GT = GT, counts_df)
```

```
# Remove cells with "64965487-1/1/1-C-T;64965488-1/1/1-C-T", "64965502-0/0/1-C-T"
# and "WT"
nrow(counts_df)
counts_df <- counts_df[counts_df$GT != "WT", ]
nrow(counts_df)
counts_df <- counts_df[counts_df$GT != "64965487-1/1/1-C-T;64965488-1/1/1-C-T", ]
nrow(counts_df)
counts_df <- counts_df[counts_df$GT != "64965502-0/0/1-C-T", ]
nrow(counts_df)

# Export via feather file
write_feather(counts_df, "counts_df.feather")
```

Exporting the counts data frame for JAK-STAT pathway and Cell Cycle genes

The following list of genes is extracted from: Human Gene Set: KEGG_JAK_STAT_SIGNALING_PATHWAY and Human Gene Set: KEGG_CELL_CYCLE. We have **274** genes that are associated with these pathways.

```
jak_stat_genes <- c("AKT1","AKT2","AKT3","BCL2L1","CBL","CBLB","CBLC","CCND1","CCND2",
                    "CCND3","CISH","CLCF1","CNTF","CNTFR","CREBBP","CRLF2","CSF2",
                    "CSF2RA","CSF2RB","CSF3","CSF3R","CSH1","CTF1","EP300","EPO",
                    "EPOR","GH1","GH2","GHR","GRB2","IFNA1","IFNA10","IFNA13","IFNA14",
                    "IFNA16","IFNA17","IFNA2","IFNA21","IFNA4","IFNA5","IFNA6","IFNA7",
                    "IFNA8","IFNAR1","IFNAR2","IFNB1","IFNE","IFNG","IFNGR1","IFNGR2",
                    "IFNK","IFNL1","IFNL2","IFNL3","IFNLR1","IFNW1","IL10","IL10RA",
                    "IL10RB","IL11","IL11RA","IL12A","IL12B","IL12RB1","IL12RB2",
                    "IL13","IL13RA1","IL13RA2","IL15","IL15RA","IL19","IL2","IL20",
                    "IL20RA","IL20RB","IL21","IL21R","IL22","IL22RA1","IL22RA2",
                    "IL23A","IL23R","IL24","IL26","IL2RA","IL2RB","IL2RG","IL3",
                    "IL3RA","IL4","IL4R","IL5","IL5RA","IL6","IL6R","IL6ST","IL7",
                    "IL7R","IL9","IL9R","IRF9","JAK1","JAK2","JAK3","LEP","LEPR","LIF",
                    "LIFR","MPL","MYC","OSM","OSMR","PIAS1","PIAS2","PIAS3","PIAS4",
                    "PIK3CA","PIK3CB","PIK3CD","PIK3CG","PIK3R1","PIK3R2","PIK3R3",
                    "PIK3R5","PIM1","PRL","PRLR","PTPN11","PTPN6","SOCS1","SOCS2",
                    "SOCS3","SOCS4","SOCS5","SOCS7","SOS1","SOS2","SPRED1","SPRED2",
                    "SPRY1","SPRY2","SPRY3","SPRY4","STAM","STAM2","STAT1","STAT2",
                    "STAT3","STAT4","STAT5A","STAT5B","STAT6","TPO","TSLP","TYK2")

cell_cycle_genes <- c("ABL1","ANAPC1","ANAPC10","ANAPC11","ANAPC13","ANAPC2","ANAPC4",
                      "ANAPC5","ANAPC7","ATM","ATR","BUB1","BUB1B","BUB3","CCNA1",
                      "CCNA2","CCNB1","CCNB2","CCNB3","CCND1","CCND2","CCND3","CCNE1",
                      "CCNE2","CCNH","CDC14A","CDC14B","CDC16","CDC20","CDC23",
                      "CDC25A","CDC25B","CDC25C","CDC26","CDC27","CDC45","CDC6",
                      "CDC7","CDK1","CDK2","CDK4","CDK6","CDK7","CDKN1A","CDKN1B",
                      "CDKN1C","CDKN2A","CDKN2B","CDKN2C","CDKN2D","CHEK1","CHEK2",
                      "CREBBP","CUL1","DBF4","E2F1","E2F2","E2F3","E2F4","E2F5",
                      "EP300","ESPL1","FZR1","GADD45A","GADD45B","GADD45G","GSK3B",
                      "HDAC1","HDAC2","MAD1L1","MAD2L1","MAD2L2","MCM2","MCM3","MCM4",
                      "MCM5","MCM6","MCM7","MDM2","MYC","ORC1","ORC2","ORC3","ORC4",
                      "ORC5","ORC6","PCNA","PKMYT1","PLK1","PRKDC","PTTG1","PTTG2",
                      "RAD21","RB1","RBL1","RBL2","RBX1","SFN","SKP1","SKP1P2","SKP2",
                      "SMAD2","SMAD3","SMAD4","SMC1A","SMC1B","SMC3","STAG1","STAG2",
                      "TFDP1","TFDP2","TGFB1","TGFB2","TGFB3","TP53","TTK","WEE1",
                      "WEE2","YWHAB","YWHAE","YWHAG","YWHAH","YWHAQ","YWHAZ","ZBTB17")
```

```r
# Combine them, removing duplicates
genes_vector <- union(jak_stat_genes, cell_cycle_genes)
```

However, our counts array uses Ensembl Gene IDs (i.e. `"JAK1"` → `"ENSG00000162434"`). So we need to convert this list to Ensembl IDs.

We can use g:Profiler to do this. However, we need to convert our list to a string, with each gene separated by whitespace.

```r
genes_string <- ""

for (i in 1:length(genes_vector)) {
  genes_string <- paste0(genes_string, genes_vector[i])
  if (i != length(genes_vector)) {
    genes_string <- paste0(genes_string, " ")
  }
}
```

By using this string as our Query for g:Profiler, we are returned the following `.csv` file, which can be used to access a list of Ensembl IDs. Note we have now gone from 274 distinct genes to **280** unique Ensembl IDs - this is because some genes have multiple encodings. This is normal behaviour.

```r
pathway_genes_conversion <- read.csv("pathway_genes_conversion.csv")

pathway_ENSG_IDs <- pathway_genes_conversion$converted_alias
```

We now wanted to reduce this list down to only those Ensembl IDs that appear amongst our features.

```r
# Get the full list of features from our data frame
all_ENSG_IDs <- colnames(counts_df)

interest_ENSG_IDs <- c()

for (i in 1:length(all_ENSG_IDs)) {
  if (all_ENSG_IDs[i] %in% pathway_ENSG_IDs) {
    interest_ENSG_IDs <- c(interest_ENSG_IDs, all_ENSG_IDs[i])
  }
}
```

```r
interest_counts_df <- counts_df[, interest_ENSG_IDs, drop = FALSE]

# Attach each cell's genotype to the counts dataframe
excluded_GT <- GT[GT != "WT"]
excluded_GT <- excluded_GT[excluded_GT != "64965487-1/1/1-C-T;64965488-1/1/1-C-T"]
excluded_GT <- excluded_GT[excluded_GT != "64965502-0/0/1-C-T"]
interest_counts_df <- data.frame(GT = excluded_GT, interest_counts_df)

# Export via feather file
write_feather(interest_counts_df, "interest_counts_df.feather")
```

As we have our data stored in feather files, we can now transfer them over to Python via the Pandas library.