BCB 724 Homework 1

Matthew Sutcliffe

2023-11-14

```
library(readxl, quietly = TRUE)
library(survival, quietly = TRUE)
library(survminer, quietly = TRUE)
```

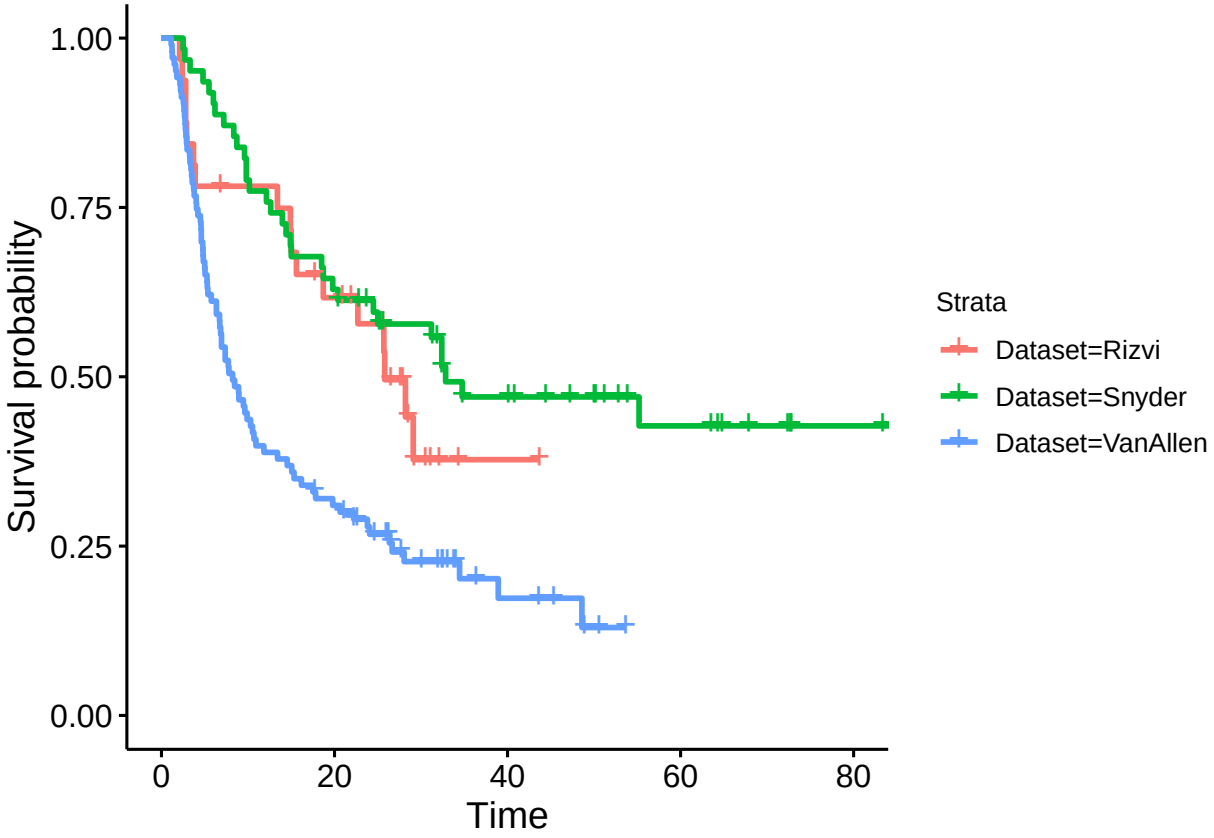
```
##
## Attaching package: 'survminer'
```

```
## The following object is masked from 'package:survival':
##
##     myeloma
```

```
set.seed(731)

x1 <- read_excel(path = "Homework 1/nature24473_MOESM4_neoantigens.xlsx", sheet = 1) |> as.data.frame()
x2 <- read_excel(path = "Homework 1/nature24473_MOESM4_neoantigens.xlsx", sheet = 2) |> as.data.frame()
x3 <- read_excel(path = "Homework 1/nature24473_MOESM4_neoantigens.xlsx", sheet = 3) |> as.data.frame()

y1 <- read_excel(path = "Homework 1/nature24473_MOESM5_survival.xlsx", sheet = 1) |> as.data.frame()
y2 <- read_excel(path = "Homework 1/nature24473_MOESM5_survival.xlsx", sheet = 2) |> as.data.frame()
y3 <- read_excel(path = "Homework 1/nature24473_MOESM5_survival.xlsx", sheet = 3) |> as.data.frame()
```

Let's do some initial cleanup, and see general trends in the 3 datasets.

```
# One sample doesn't have a Status, so let's remove it
x2 <- x2[!(x2$Sample %in% y2$Sample[which(!(y2$Status %in% c("0", "1")))]), ]
y2 <- y2[!(y2$Sample %in% y2$Sample[which(!(y2$Status %in% c("0", "1")))]), ]
y2$Status <- as.numeric(y2$Status)

x1$Dataset <- "VanAllen"
x2$Dataset <- "Snyder"
x3$Dataset <- "Rizvi"

y1$Dataset <- "VanAllen"
y2$Dataset <- "Snyder"
y3$Dataset <- "Rizvi"

x <- rbind(x1, x2, x3)
y <- rbind(y1, y2, y3)
rm(x1, x2, x3, y1, y2, y3)

y$Dataset <- factor(y$Dataset)

# There are samples for which we don't have any mutants... Let's remove those
y <- y[y$Sample %in% intersect(y$Sample, x$Sample), ]

fit <- survfit(formula = Surv(Months, Status) ~ Dataset, data = y)
ggsurvplot(fit, legend = "right")
```



Dataset seems important.

Output of interest

We're going to try to predict whether patients will live for $>$ 18 months.

```
month_threshold <- 18
```

Defining variables

Let's make a bunch of features based on the data matrices. Each new column in the y data frame will be a new feature.

```
# Number of neoantigens per patient
y$Num_Neoantigens <- table(x$Sample)[y$Sample]

# Stats based on relative IC50 of WT and MT
y$Mean_MT_Score <- sapply(X = y$Sample, FUN = function(iSample) {
    xi <- x[x$Sample == iSample, ]
    mean_score <- mean(xi$MT.Score)
    return(mean_score)
})
y$Max_MT_Score <- sapply(X = y$Sample, FUN = function(iSample) {
    xi <- x[x$Sample == iSample, ]
    max_score <- max(xi$MT.Score)
    return(max_score)
})
y$Min_MT_Score <- sapply(X = y$Sample, FUN = function(iSample) {
    xi <- x[x$Sample == iSample, ]
    min_score <- min(xi$MT.Score)
    return(min_score)
})
y$Mean_Score_Ratio <- sapply(X = y$Sample, FUN = function(iSample) {
    xi <- x[x$Sample == iSample, ]
    score_ratio <- xi$WT.Score / xi$MT.Score
    avg_score_ratio <- mean(score_ratio, na.rm = TRUE)
```

```
    return(avg_score_ratio)
})
y$Min_Score_Ratio <- sapply(X = y$Sample, FUN = function(iSample) {
    xi <- x[x$Sample == iSample, ]
    score_ratio <- xi$WT.Score / xi$MT.Score
    min_score_ratio <- min(score_ratio, na.rm = TRUE)
    return(min_score_ratio)
})
y$Max_Score_Ratio <- sapply(X = y$Sample, FUN = function(iSample) {
    xi <- x[x$Sample == iSample, ]
    score_ratio <- xi$WT.Score / xi$MT.Score
    max_score_ratio <- max(score_ratio, na.rm = TRUE)
    return(max_score_ratio)
})
y$MT_score_less <- table(factor(x = x$Sample, levels = y$Sample), x$WT.Score > x$MT.Score)[,2]
y$MT_score_greater <- table(factor(x = x$Sample, levels = y$Sample), x$WT.Score > x$MT.Score)[,1]

# Number of mutations per chromosome
chr <- sapply(X = x$MUTATION_ID, FUN = function(i) {
    strsplit(x = i, split = "_")[[1]][1]}) |> unname()
chr_matrix <- as.data.frame.matrix(table(factor(x = x$Sample,
                                                levels = y$Sample), chr))[y$Sample, ]
names(chr_matrix) <- paste0("chr_", names(chr_matrix))
y <- cbind(y, chr_matrix)

# Number of mutations per chromosome (as a fraction of total mutations)
chr_matrix_fraction <- apply(X = chr_matrix,
                             MARGIN = 1,
                             FUN = function(i) i / sum(i)) |> t() |> as.data.frame.matrix()
names(chr_matrix_fraction) <- paste0("chr_fraction_", names(chr_matrix_fraction))
y <- cbind(y, chr_matrix_fraction)

# DNA base mutation WT
mut_from <- sapply(X = x$MUTATION_ID,
                   FUN = function(i) strsplit(x = i, split = "_")[[1]][3]) |> unname()
mut_from_matrix <- as.data.frame.matrix(table(factor(x = x$Sample,
                                                     levels = y$Sample), mut_from))[y$Sample, ]
names(mut_from_matrix) <- paste0("mut_from_", names(mut_from_matrix))
y <- cbind(y, mut_from_matrix)

# DNA base mutation MT
mut_to <- sapply(X = x$MUTATION_ID,
                 FUN = function(i) strsplit(x = i, split = "_")[[1]][3]) |> unname()
mut_to_matrix <- as.data.frame.matrix(table(factor(x = x$Sample,
                                                   levels = y$Sample), mut_to))[y$Sample, ]
names(mut_to_matrix) <- paste0("mut_to_", names(mut_to_matrix))
y <- cbind(y, mut_to_matrix)

# DNA base mutation WT_MT
mut <- sapply(X = x$MUTATION_ID,
              FUN = function(i) paste(strsplit(x = i, split = "_")[[1]][c(3,4)],
                                      collapse = "_")) |> unname()
mut_matrix <- as.data.frame.matrix(table(factor(x = x$Sample,
                                                levels = y$Sample), mut))[y$Sample, ]
names(mut_matrix) <- paste0("mut_", names(mut_matrix))
y <- cbind(y, mut_matrix)

# DNA base mutations WT_MT (as a fraction of total mutations)
mut_matrix_fraction <- apply(X = mut_matrix,
                             MARGIN = 1,
                             FUN = function(i) i / sum(i)) |> t() |> as.data.frame.matrix()
names(mut_matrix_fraction) <- paste0("mut_fraction_", names(mut_matrix_fraction))
y <- cbind(y, mut_matrix_fraction)

# DNA base mutations chr_WT_MT
```

```r
chr_mut <- sapply(X = x$MUTATION_ID,
                  FUN = function(i) paste(strsplit(x = i,
                                                   split = "_")[[1]][c(1,3,4)],
                                          collapse = "_")) |> unname()
chr_mut_matrix <- as.data.frame.matrix(table(factor(x = x$Sample,
                                                    levels = y$Sample), chr_mut))[y$Sample, ]
names(chr_mut_matrix) <- paste0("chr_mut_", names(chr_mut_matrix))
y <- cbind(y, chr_mut_matrix)

# DNA base mutations chr_WT_MT (as a fraction of total mutations on that chr)
chr_mut_matrix_fraction <- apply(X = chr_mut_matrix, MARGIN = 1, FUN = function(i) i / sum(i)) |> t() |> as.d
names(chr_mut_matrix_fraction) <- paste0("chr_mut_fraction_", names(chr_mut_matrix_fraction))
y <- cbind(y, chr_mut_matrix_fraction)

# Neoantigens per HLA
HLA_matrix <- table(factor(x = x$Sample, levels = y$Sample), x$MT.Allele)
HLA_matrix <- as.data.frame.matrix(HLA_matrix)[y$Sample, ]
HLA_matrix <- HLA_matrix[, colSums(HLA_matrix > 0) > 1]
y <- cbind(y, HLA_matrix)

# 9-mer count (removing unique 9-mers)
MT_matrix <- table(factor(x = x$Sample, levels = y$Sample), x$MT.Peptide)
MT_matrix <- as.data.frame.matrix(MT_matrix)[y$Sample, ]
MT_matrix <- MT_matrix[, colSums(MT_matrix > 0) > 1]
y <- cbind(y, MT_matrix)

# AA position change (1 - 9)
Position_Change <- apply(X = x, MARGIN = 1, FUN = function(i) {
    position_change <- mapply(function(x, y) which(x != y)[1], strsplit(i[4], ""), strsplit(i[5], ""))
    return(position_change)
})
Position_matrix <- table(factor(x = x$Sample, levels = y$Sample), Position_Change)
Position_matrix <- as.data.frame.matrix(Position_matrix)[y$Sample, ]
colnames(Position_matrix) <- paste0("Position_",1:9,"_Change_Count")
y <- cbind(y, Position_matrix)

# AA position change (1 - 9) (as a fraction of total mutations)
Position_Change_Fraction <- apply(X = Position_matrix, MARGIN = 2, FUN = function(i) i / y$Num_Neoantigens)
colnames(Position_Change_Fraction) <- paste0("Position_",1:9,"_Change_Fraction")
y <- cbind(y, Position_Change_Fraction)

# AA position change (1 - 9) (as a fraction of total mutations on that chr)
y <- cbind(y, as.data.frame.matrix(table(factor(x = x$Sample, levels = y$Sample), paste0("chr", chr, "_pos",

# AA substitution WT_MT
From_To <- apply(X = x, MARGIN = 1, FUN = function(i) {
    position_change <- mapply(function(x, y) which(x != y)[1], strsplit(i[4], ""), strsplit(i[5], ""))
    from_ <- strsplit(i[4], "")[[1]][position_change]
    to_ <- strsplit(i[5], "")[[1]][position_change]
    from_to <- paste0(from_, "_", to_)

    return(from_to)
})
From_To_Matrix <- table(factor(x = x$Sample, levels = y$Sample), From_To)
From_To_Matrix <- as.data.frame.matrix(From_To_Matrix)[y$Sample, ]
From_To_Matrix <- From_To_Matrix[, colSums(From_To_Matrix) > 1]
y <- cbind(y, From_To_Matrix)

# AA substitution WT_MT (as a fraction of total mutations)
From_To_Matrix_Fraction <- apply(X = From_To_Matrix, MARGIN = 2, FUN = function(i) i / y$Num_Neoantigens)
colnames(From_To_Matrix_Fraction) <- paste0(colnames(From_To_Matrix_Fraction), "_Fraction")
y <- cbind(y, From_To_Matrix_Fraction)

# AA substitution WT_MT (as a fraction of total mutations on that chr)
y <- cbind(y, as.data.frame.matrix(table(factor(x = x$Sample, levels = y$Sample), paste0("chr", chr, "_from_t
```

```r
# AA positioning count (1-mers)
amino_acids <- paste0(x$WT.Peptide, collapse = "") |> strsplit(split = "") |> unlist() |> unique() |> sort()
aa_positions <- paste(rep(amino_acids, each = 9), 1:9, sep = "_")
AA <- matrix(data = unlist(strsplit(x$MT.Peptide, split = "")), nrow = nrow(x), byrow = TRUE)

AA1 <- lapply(X = amino_acids, FUN = function(i) ifelse(AA == i, 1, 0))
AA1 <- Reduce(cbind, AA1)
colnames(AA1) <- aa_positions
AA1 <- apply(X = AA1, MARGIN = 2, FUN = function(i) {
    as.data.frame.matrix(table(factor(x = x$Sample, levels = y$Sample), i))[, 2]
})
y <- cbind(y, AA1)

# AA positioning count (2-mers)
AA2 <- apply(X = expand.grid(amino_acids, amino_acids), MARGIN = 1, FUN = function(i) paste0(i, collapse = "
for (i in AA2) {
    for (j in 1:(9-1)) {
        ijvec <- apply(X = AA, MARGIN = 1, FUN = function(k) {
            paste(k[j:(j+1)], collapse = "") == i
        }) |> ifelse(1,0)
        if (all(ijvec == 0)) {
            next
        }
        y[[paste(i, j, sep="_")]] <- as.data.frame.matrix(table(factor(x = x$Sample, levels = y$Sample), ijve
    }
}
```

Save intermediate results because this takes a little while.

```r
save.image("hw1_makefeatures.RData")
# load("hw1_makefeatures.RData")
```

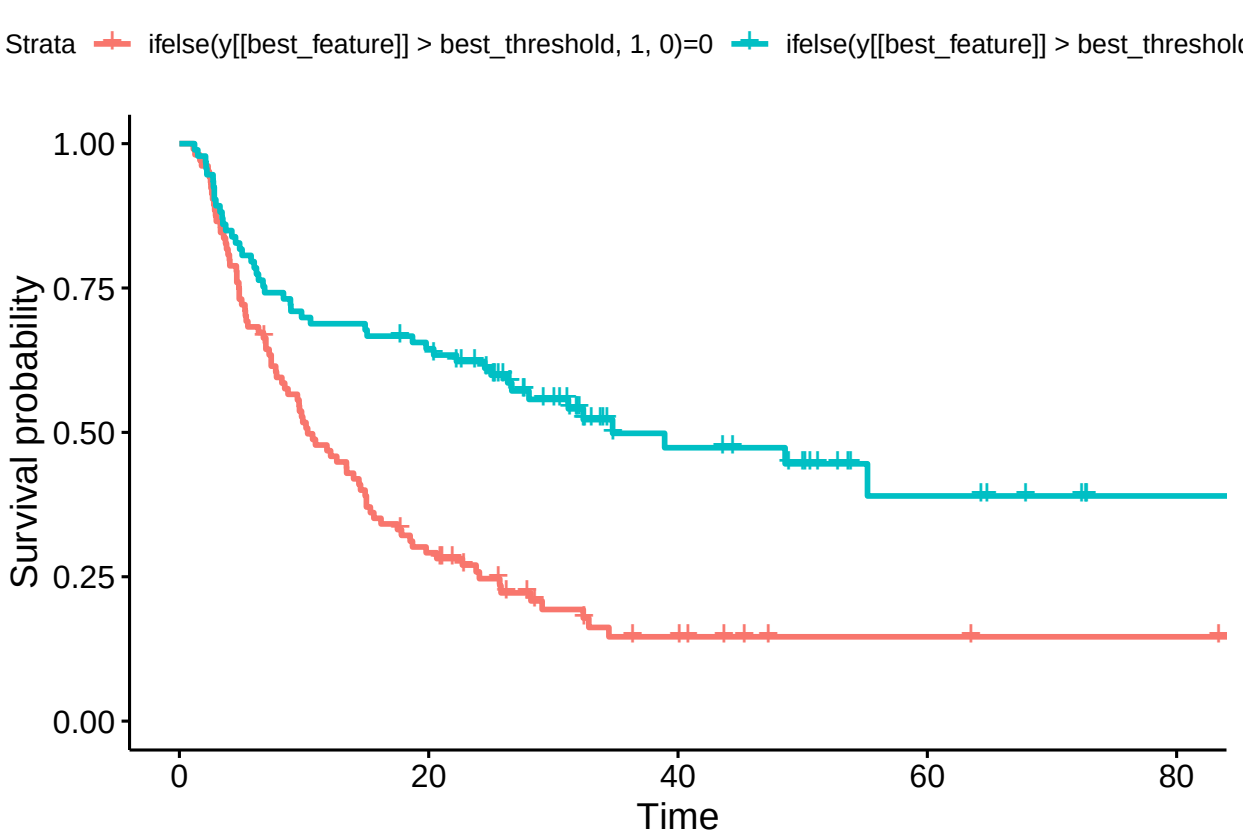
Feature selection

```r
best_feature <- ""
best_threshold <- NA
best_pvalue <- 1

all_features <- c()
all_thresholds <- c()
all_pvalues <- c()
for (iFeature in names(y)[-(1:5)]) {
    for (iThreshold in sort(unique(y[[iFeature]]))) {
        dist_feature <- table(y[[iFeature]] > iThreshold)
        # I only want to keep thresholds that reasonably split the dataset (not 99% patients + 1% patients)
        if ((length(dist_feature) != 2) | (any(dist_feature < 0.25 * nrow(y)))) {
            next
        }

        p <- survdiff(formula = Surv(Months, Status) ~ ifelse(y[[iFeature]] > iThreshold, 1, 0), data = y)$pv
        if (p < best_pvalue) {
            best_pvalue <- p
            best_feature <- iFeature
            best_threshold <- iThreshold
            message(paste("p =", best_pvalue, "with feature", best_feature, "and threshold >", best_threshold
        }
        all_features <- c(all_features, iFeature)
        all_thresholds <- c(all_thresholds, iThreshold)
        all_pvalues <- c(all_pvalues, p)
    }
}
```

```
## p = 0.413337329735883 with feature Mean_MT_Score and threshold > 159.473214285714

## p = 0.404858396959418 with feature Mean_MT_Score and threshold > 160.690217391304

## p = 0.402582740739794 with feature Mean_MT_Score and threshold > 162.389830508475

## p = 0.352707613966562 with feature Mean_MT_Score and threshold > 164.368421052632

## p = 0.327075983108805 with feature Mean_MT_Score and threshold > 165.070422535211

## p = 0.262883025818874 with feature Mean_MT_Score and threshold > 165.14263322884

## p = 0.0123204872905733 with feature Max_MT_Score and threshold > 487

## p = 0.0111863672582609 with feature MT_score_less and threshold > 41

## p = 0.010007415071385 with feature MT_score_less and threshold > 44

## p = 0.0037479547723687 with feature MT_score_less and threshold > 45

## p = 0.00222778488631242 with feature MT_score_less and threshold > 48

## p = 0.00207300472316837 with feature MT_score_less and threshold > 49

## p = 0.000663633894015202 with feature MT_score_less and threshold > 50

## p = 0.000486153451198484 with feature MT_score_less and threshold > 56

## p = 0.000163211564056697 with feature MT_score_less and threshold > 59

## p = 0.000110233019398049 with feature MT_score_greater and threshold > 24

## p = 0.000102038999108569 with feature chr_8 and threshold > 3

## p = 9.81770098509217e-05 with feature mut_from_G and threshold > 33

## p = 1.17830542624967e-05 with feature Position_2_Change_Count and threshold > 8

## p = 1.5094666570299e-06 with feature chr8_pos4 and threshold > 0
```

Performance on the entire dataset

```
best_fit <- survfit(formula = Surv(Months, Status) ~ ifelse(y[[best_feature]] > best_threshold, 1, 0), data =
ggsurvplot(best_fit)
```



Make small data frame with only the best feature (thresholded)

```
y_best <- y[, c("Sample", "Months", "Status", "Dataset")]
y_best[[paste0(best_feature,"_>_",best_threshold)]] <- y[[best_feature]] > 0
```

Training and Testing split

We're going to have 70% training and 30% testing, with approximately equal representation from each of the 3 datasets.

```
set.seed(0)
# Training and Test split -----
dataset_weight <- table(y_best$Dataset) / nrow(y)
samples_train <- sample(x = y_best$Sample, size = round(0.7 * nrow(y_best)), prob = dataset_weight[y_best$Dat
samples_test <- setdiff(y_best$Sample, samples_train)

y_train <- y_best[y_best$Sample %in% samples_train, ]
y_test <- y_best[y_best$Sample %in% samples_test, ]
```

Fit the model on the training set

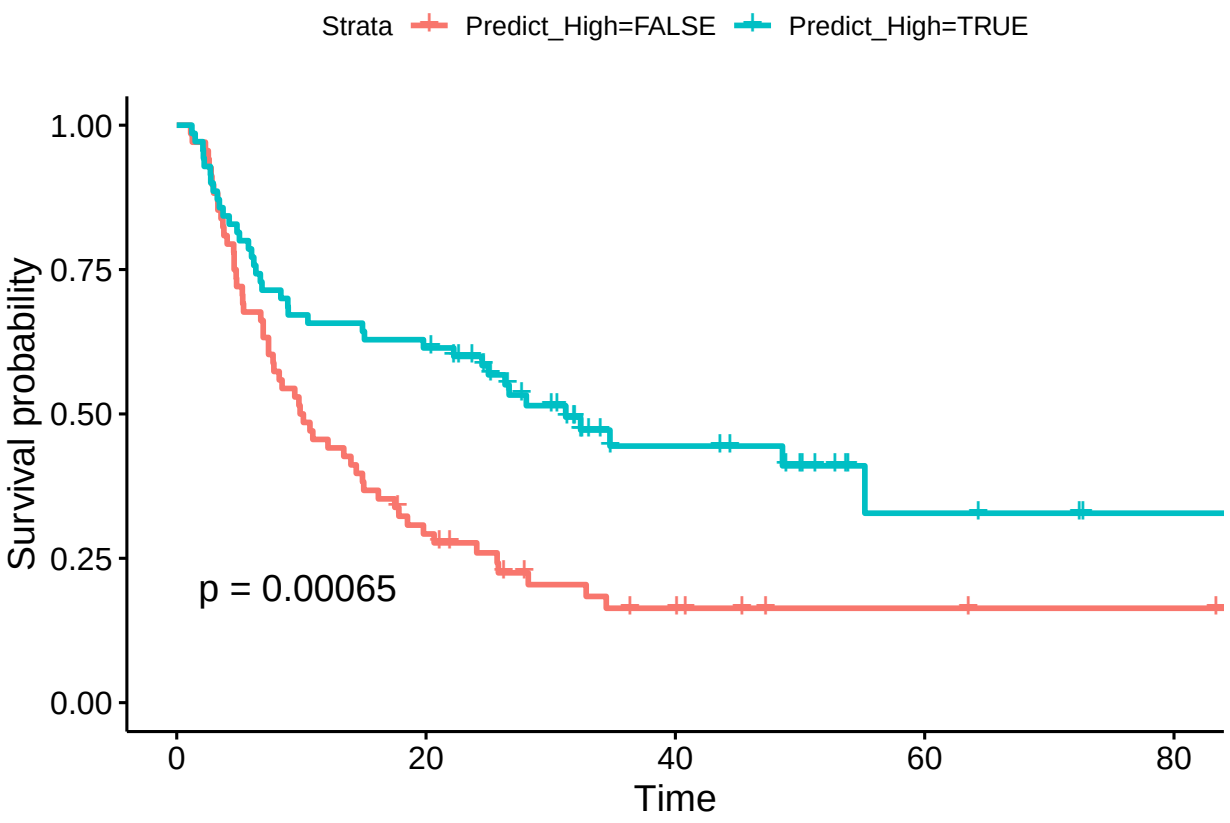
```
fit <- glm(formula = (Months > month_threshold) ~ 0 + `chr8_pos4_>_0`, family = "binomial", data = y_train)
summary(fit)
```

```
##
## Call:
## glm(formula = (Months > month_threshold) ~ 0 + `chr8_pos4_>_0`,
##     family = "binomial", data = y_train)
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## `chr8_pos4_>_0`FALSE  -0.8056     0.2625  -3.069  0.00215 **
## `chr8_pos4_>_0`TRUE    0.5261     0.2474   2.127  0.03344 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 191.31  on 138  degrees of freedom
## Residual deviance: 176.43  on 136  degrees of freedom
## AIC: 180.43
##
## Number of Fisher Scoring iterations: 4
```

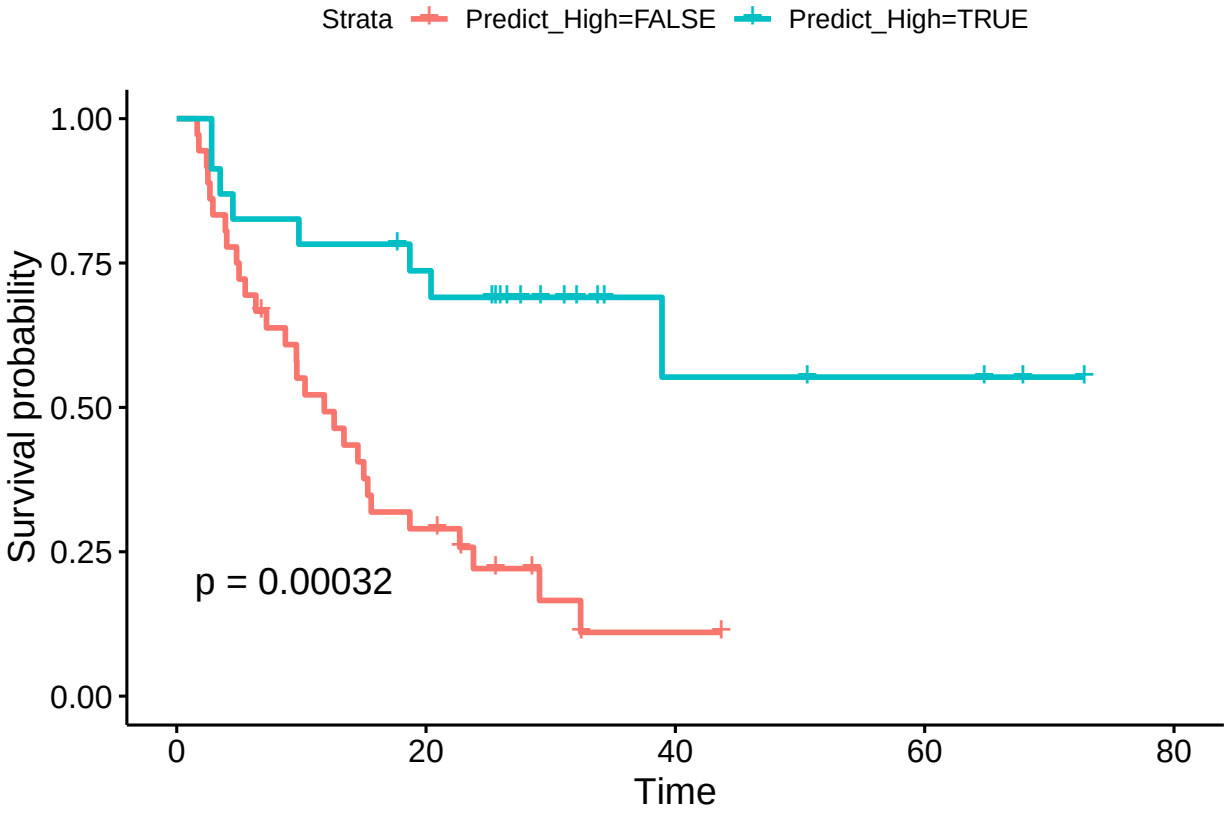
Performance on training set

```
y_train$Predict_High <- predict.glm(object = fit, newdata = y_train, type = "response") > 0.5
ggsurvplot(fit = survfit(Surv(Months, Status) ~ Predict_High, data = y_train), pval = TRUE)
```



Performance on test set

```
y_test$Predict_High <- predict.glm(object = fit, newdata = y_test, type = "response") > 0.5
ggsurvplot(fit = survfit(Surv(Months, Status) ~ Predict_High, data = y_test), pval = TRUE)
```



We see that the predictor performs very well on the test data!

What I actually did

First, I generated a lot of features. Then, I iteratively went through the features and noted those with significant associations with survival. I then used the best feature (number of position 4 [of 9] substitutions on chromosome 8) to fit a logistic regression model using the training data. I then applied this model to the test data and found a significant split.