ass_2

Noah King - nki38 - 96851177

2024-09-07

Question 1 - Death

Preamble

1. Load in the dataset…

```r
data <- read.csv("heart.csv", header=TRUE, sep=",")
data <- data[complete.cases(data[, c("DEATH", "GLUCOSE", "SYSBP")]), ]
```

We wish to predict the value of DEATH, which is defined in our training data as either =1 or =0 given some number of explanatory variables…. We will consider GLUCOSE (amount of glucose in blood at last measurement) and SYSBP (systolic blood pressure at last measurement).

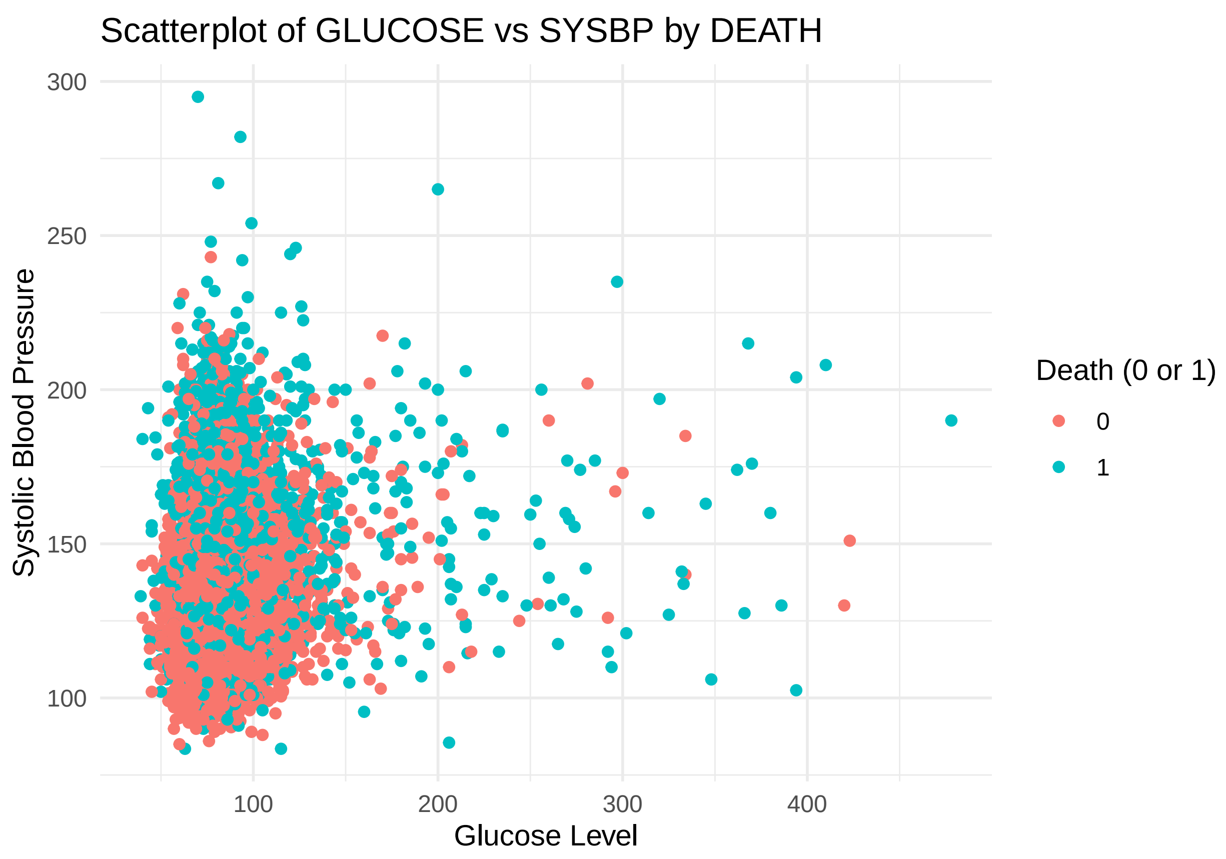
a Firstly split the dataset into train, test

```r
train_ind <- sample(1:nrow(data), size = nrow(data) * 0.8)
df_train <- data[train_ind, ]
df_test <- data[-train_ind, ]
```

b Intuitively, we might want to form a hypothesis that More Glucose or Higher blood pressure is correlated with Death = 1…. We will form the hypothesis that if both are high, theur comibination increases the risk of death.

Note: The data below is the entire set

```r
ggplot(data, aes(x = GLUCOSE, y = SYSBP, color = factor(DEATH))) +
  geom_point() +
  labs(title = "Scatterplot of GLUCOSE vs SYSBP by DEATH",
       x = "Glucose Level",
       y = "Systolic Blood Pressure",
       color = "Death (0 or 1)") +
  theme_minimal()
```



Indeed, per above, we see that increases in either Blood Pressure or Glucose does correlate (at least per looking) with death; it is therefore likely that htese contribute to risk of death.

c: We can fit a multiple logistic regression model thusly:

```r
model <- glm(DEATH ~ GLUCOSE + SYSBP, data = df_train, family = binomial)
summary(model)
```

```
##
## Call:
## glm(formula = DEATH ~ GLUCOSE + SYSBP, family = binomial, data = df_train)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -4.785990   0.174984 -27.351  < 2e-16 ***
## GLUCOSE      0.007172   0.001053   6.808 9.88e-12 ***
## SYSBP        0.024284   0.001126  21.571  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 10023  on 8148  degrees of freedom
## Residual deviance:  9416  on 8146  degrees of freedom
## AIC: 9422
##
```

```
## Number of Fisher Scoring iterations: 4
```

We define a classification threshold (i.e. the model will output a prob of class =1, class = 0, if the threshold is met it is the class)

```
CLASSIFICATION_THRESH <- 0.5
```

c.i Now that we've created the model, we fit the model.... Using the logic above we define the output class in line 2

```
predicted_prob <- predict(model, newdata = df_test, type = "response")
predicted_class <- ifelse(predicted_prob > CLASSIFICATION_THRESH, 1, 0)
```

Our first metric is the misclass. rate:

```
misclassification_rate <- mean(predicted_class != df_test$DEATH)
cat("For the test set, the misclassification rate is:", misclassification_rate, "\n")
```

```
## For the test set, the misclassification rate is: 0.281158
```

c.ii The confusion matrix for our tes set is the following:

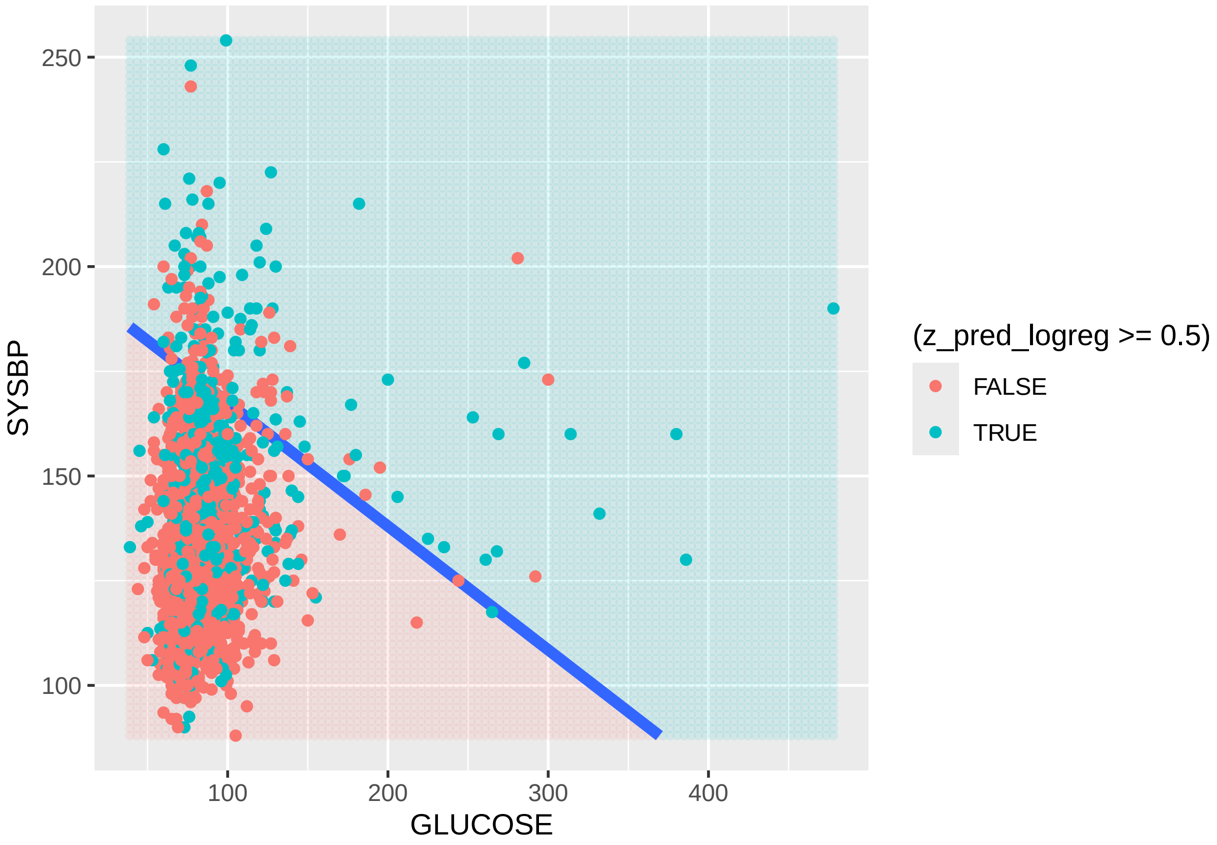
```
confusion_matrix <- table(Predicted = predicted_class, Actual = df_test$DEATH)
#formatting for readability
TN <- confusion_matrix[1,1]
FP <- confusion_matrix[2,1]
FN <- confusion_matrix[1,2]
TP <- confusion_matrix[2,2]
confusion_df <- data.frame(
  Outcome = c("True Positive", "False Positive", "True Negative", "False Negative"),
  Count = c(TP, FP, TN, FN)
)
kable(confusion_df, caption = "All Confusion Matrix Results")
```

Table 1: All Confusion Matrix Results

Outcome	Count
True Positive	90
False Positive	74
True Negative	1375
False Negative	499

```
#The below line just lets us reuse this code later, ignore
LR_CONF_MAT <- confusion_matrix
```

```r
df_test$z_pred_logreg <- predict(model, type = "response", newdata = df_test)
grid_glucose <- seq(min(df_test$GLUCOSE), max(df_test$GLUCOSE), length.out = 100)
grid_sysbp <- seq(min(df_test$SYSBP), max(df_test$SYSBP), length.out = 100)


dfplot <- expand.grid(GLUCOSE = grid_glucose, SYSBP = grid_sysbp)


dfplot$z_pred_logreg <- predict(model, newdata = dfplot, type = "response")
ggplot() +
  geom_point(data = dfplot, aes(x = GLUCOSE, y = SYSBP, color = (z_pred_logreg >= 0.5)), alpha = 0.1, s
  geom_contour(data = dfplot, aes(x = GLUCOSE, y = SYSBP, z = z_pred_logreg), breaks = c(0.5), linewidt
  geom_point(data = df_test, aes(x = GLUCOSE, y = SYSBP, color = (DEATH >= 0.5)))
```



c.ii

We can see the classification boudnary on the data above… Our generated decision boundary is *okay* but not a great indication of the true bondary.

d

-

For public health purposes it is more important to catch positives, i.e. potential mortality risks, even if they end up not eventuating. In other words, false negatives are more dangerous than false positives.

In order to address this problem, we can change the threshold at which an patient is classified as being "risky": Instead of setting the decision boundary at probability p =50% , we classify a customer as "risky" (i.e., we predict DEATH) if the risk of them dying is higher than 10% . Modify your logistic regression to do this, and repeat the tasks of question c).

Compare the performance of logistic regression and discriminant analysis on this classification problem. *

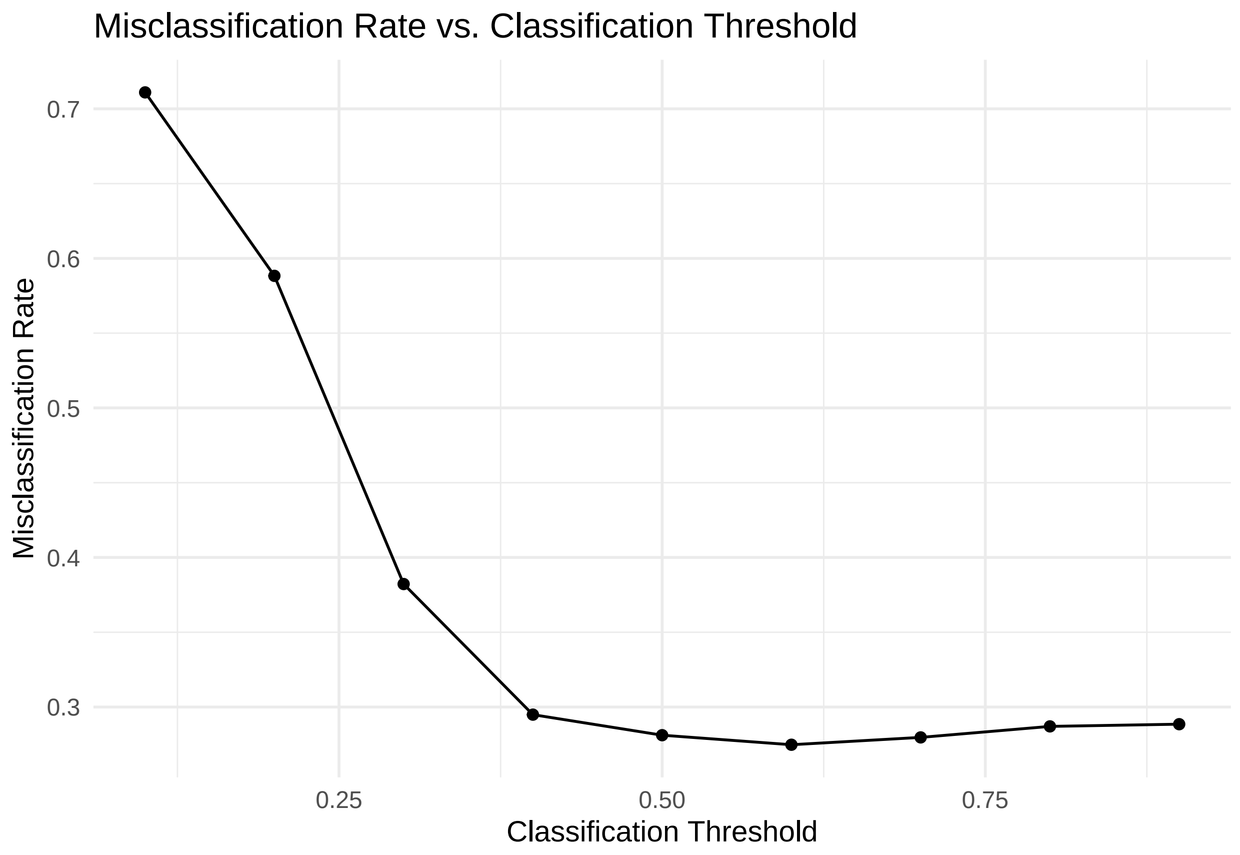
We will consider a number of Thresholds.

```r
# Define thresholds explicitly
thresholds <- seq(0.9, 0.1, by = -0.1)
results <- data.frame(Threshold = numeric(), Misclassification_Rate = numeric())
confusion_matrices <- list()
#data
for (threshold in thresholds) {
  predicted_prob <- predict(model, newdata = df_test, type = "response")
  predicted_class <- ifelse(predicted_prob > threshold, 1, 0)
  misclassification_rate <- mean(predicted_class != df_test$DEATH)
  results <- rbind(results, data.frame(Threshold = threshold, Misclassification_Rate = misclassificatio

  if (threshold %in% c(0.5,0.7, 0.1)) {
    # R ..
    confusion_matrix <- table(Predicted = predicted_class, Actual = df_test$DEATH)
    confusion_matrices[[as.character(threshold)]] <- confusion_matrix
  }
}





kable(results, caption = "Misclassification Rates for Different Thresholds")
```

Table 2: Misclassification Rates for Different Thresholds

Threshold	Misclassification_Rate
0.9	0.2885182
0.8	0.2870461
0.7	0.2796860
0.6	0.2747792
0.5	0.2811580
0.4	0.2948970
0.3	0.3822375
0.2	0.5883219
0.1	0.7109912

```r
ggplot(results, aes(x = Threshold, y = Misclassification_Rate)) +
  geom_line() +
  geom_point() +
  labs(title = "Misclassification Rate vs. Classification Threshold",
       x = "Classification Threshold",
       y = "Misclassification Rate") +
  theme_minimal()
```



```r
for (threshold in c(0.5,0.7, 0.1)) {
  cat("\nConfusion Matrix for Threshold:", threshold, "\n")
  print(confusion_matrices[[as.character(threshold)]])
}
```

```
##
## Confusion Matrix for Threshold: 0.5
##          Actual
## Predicted    0    1
##         0 1375  499
##         1   74   90
##
## Confusion Matrix for Threshold: 0.7
##          Actual
## Predicted    0    1
##         0 1444  565
##         1    5   24
##
## Confusion Matrix for Threshold: 0.1
##          Actual
## Predicted    0    1
##         1 1449  589
```

Graphing for values $<$ *0.5* is not prudent given the domain; however we do see that the difference in misclassification rate as our threshold increases is rather small and such a change does not well increase the model's ability to determine the class given some input.

D2: We compare the performance of this data set against QDA using the inbuilt lib (manually implemented in p2 as well):

```
ddata_t <- df_train
ddata_t$DEATH <- as.factor(ddata_t$DEATH)
qda_model <- qda(DEATH ~ GLUCOSE + SYSBP, data=ddata_t)
df_test$DEATH <- as.factor(df_test$DEATH)
qda_preds <- predict(qda_model, newdata=df_test)$class
actual_classes <- df_test$DEATH
lr_preds <- predict(model, newdata = df_test)
```

*We'll just reuse our matrix from earlier for log regression performance

```
qda_conf_matrix <- table(Predicted = qda_preds, Actual = actual_classes)
LR_CONF_MAT
```

```
##          Actual
## Predicted    0    1
##         0 1375  499
##         1   74   90
```

```
qda_conf_matrix
```

```
##          Actual
## Predicted    0    1
##         0 1350  478
##         1   99  111
```

And calculating the residuals:

```
getstats <- function(conf_matrix) {

  TN <- conf_matrix[1, 1]
  FP <- conf_matrix[2, 1]
  FN <- conf_matrix[1, 2]
  TP <- conf_matrix[2, 2]


  sensitivity <- TP / (TP + FN)
  specificity <- TN / (TN + FP)
  accuracy <- (TP + TN) / (TP + TN + FP + FN)

  results <- list(
    Sensitivity = sensitivity,
    Specificity = specificity,
    Accuracy = accuracy
  )

  return(results)
}

lr_stats <- getstats(LR_CONF_MAT)
qda_stats <-getstats(qda_conf_matrix)
cat("Log.Reg: Sensitivity", lr_stats$Sensitivity,"Log.Reg: Specificity", lr_stats$Specificity,"Log.Reg:
```

```
## Log.Reg: Sensitivity 0.1528014 Log.Reg: Specificity 0.9489303 Log.Reg: Accuracy 0.718842
```

```
cat("QDA: Sensitivity", qda_stats$Sensitivity,"QDA: Specificity", qda_stats$Specificity,"QDA: Accuracy"
```

```
## QDA: Sensitivity 0.188455 QDA: Specificity 0.931677 QDA: Accuracy 0.7168793
```

Noting we are just using the 0.5 thresehold as we didn't observe great improvements with changing that (to specificity, accuracy or sensitivty)

Both models, approaches have similar accuracy (although this doesnt account fo rthe balance of FN / FP). The higher specificfty of our Log.Reg indicates that LR is marginally better at correctly identifying TN; however; for our domain, identifying TP is paramoutn and as such the higher sensitivty of QDA is (identifying TP) is best; indicating that QDA may be the better of the models to select here (marginally).

D3 The dataset contains more columns than simply Glucose, SYSBP and DEATH. To identify risk factors (i.e. factosr most associated with death) we can assess some summary statistics by grouping on DEATH = 1 and DEATH = 0 (note, I didn't remove other binary vars like SEX)

The following simply calculates the % diff in our descriptive stat, we skip to the first non 200 value as those are artifacts of our negligence with not scrubbing binary cols.

```
df_train_clean <- na.omit(df_train)
```

```
library(dplyr)

# Summarize data by DEATH
summary_by_death <- df_train_clean %>%
  group_by(DEATH) %>%
  summarise(across(everything(), list(mean = ~mean(., na.rm = TRUE),
                                      sd = ~sd(., na.rm = TRUE),
                                      median = ~median(., na.rm = TRUE),
                                      IQR = ~IQR(., na.rm = TRUE))))





summary_by_death
```

```
## # A tibble: 2 x 153
##   DEATH RANDID_mean RANDID_sd RANDID_median RANDID_IQR SEX_mean SEX_sd
##   <int>       <dbl>     <dbl>         <dbl>      <dbl>    <dbl>  <dbl>
## 1     0    4957048.  2904423.       4876316    5116843     1.61  0.489
## 2     1    4993800.  2891239.       4969600    4734126     1.41  0.493
## # i 146 more variables: SEX_median <dbl>, SEX_IQR <dbl>, TOTCHOL_mean <dbl>,
## #   TOTCHOL_sd <dbl>, TOTCHOL_median <dbl>, TOTCHOL_IQR <dbl>, AGE_mean <dbl>,
## #   AGE_sd <dbl>, AGE_median <dbl>, AGE_IQR <dbl>, SYSBP_mean <dbl>,
## #   SYSBP_sd <dbl>, SYSBP_median <dbl>, SYSBP_IQR <dbl>, DIABP_mean <dbl>,
## #   DIABP_sd <dbl>, DIABP_median <dbl>, DIABP_IQR <dbl>, CURSMOKE_mean <dbl>,
## #   CURSMOKE_sd <dbl>, CURSMOKE_median <dbl>, CURSMOKE_IQR <dbl>,
## #   CIGPDAY_mean <dbl>, CIGPDAY_sd <dbl>, CIGPDAY_median <dbl>, ...
```

```r
zero_values <- summary_by_death[1, ]
one_values <- summary_by_death[2, ]
zero_values <- as.numeric(zero_values)
one_values <- as.numeric(one_values)
difference <- (abs(zero_values - one_values) / ((zero_values + one_values) / 2)) * 100
summary_by_death <- rbind(summary_by_death, difference)
third_row <- as.numeric(summary_by_death[3, ])
ordered_columns <- order(third_row, decreasing = TRUE)
summary_by_death <- summary_by_death[, ordered_columns]
summary_by_death
```

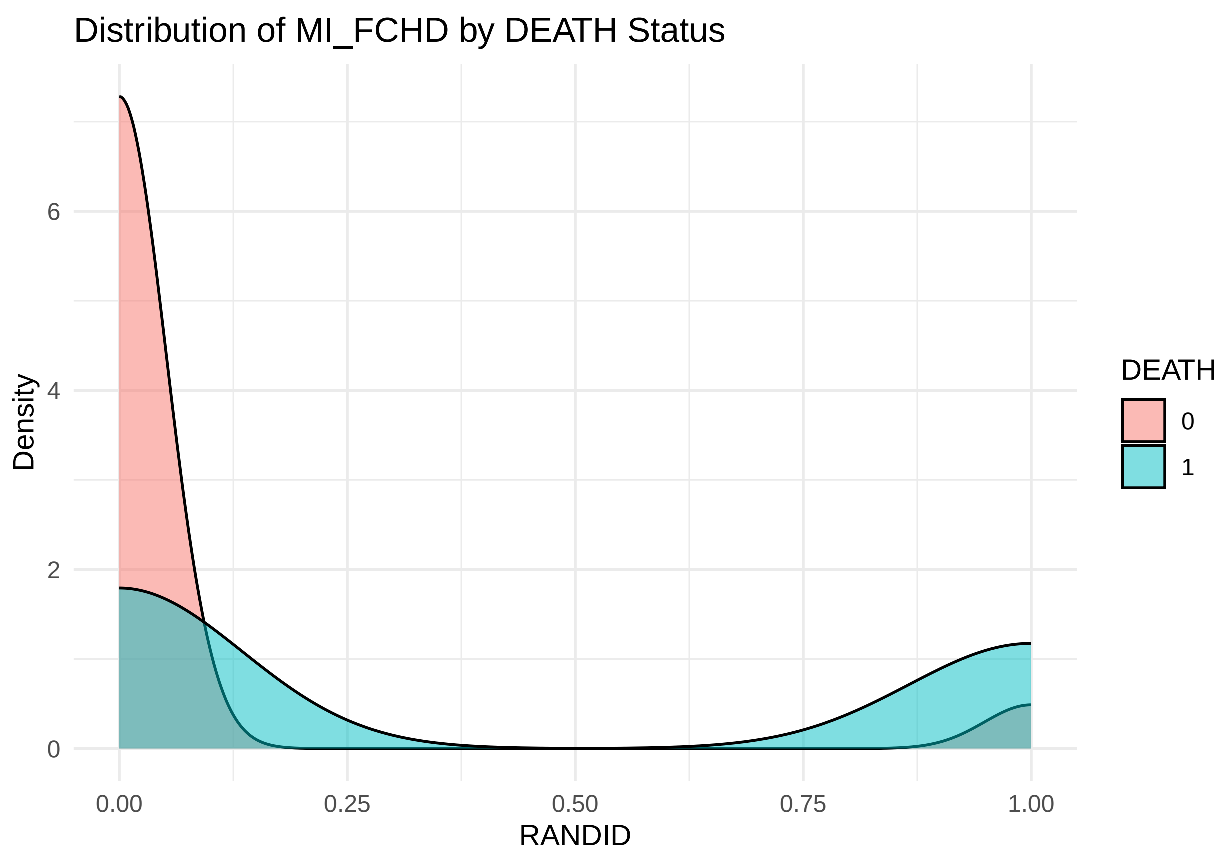
```
## # A tibble: 3 x 153
##   DEATH PREVHYP_IQR ANGINA_IQR MI_FCHD_IQR ANYCHD_median ANYCHD_IQR CVD_median
##   <dbl>       <dbl>      <dbl>       <dbl>         <dbl>      <dbl>      <dbl>
## 1     0           1          0           0             0          0          0
## 2     1           0          1           1             1          1          1
## 3   200         200        200         200           200        200        200
## # i 146 more variables: CVD_IQR <dbl>, HYPERTEN_IQR <dbl>, TIMEAP_IQR <dbl>,
## #   TIMEMI_IQR <dbl>, TIMEMIFC_IQR <dbl>, TIMECHD_IQR <dbl>,
## #   TIMESTRK_IQR <dbl>, TIMECVD_IQR <dbl>, TIMEDTH_sd <dbl>, TIMEDTH_IQR <dbl>,
## #   MI_FCHD_mean <dbl>, PREVSTRK_mean <dbl>, PREVMI_mean <dbl>,
## #   HOSPMI_mean <dbl>, CVD_mean <dbl>, PREVAP_mean <dbl>, STROKE_mean <dbl>,
## #   PREVCHD_mean <dbl>, DIABETES_mean <dbl>, ANYCHD_mean <dbl>,
## #   TIMEHYP_median <dbl>, PREVSTRK_sd <dbl>, PREVMI_sd <dbl>, ...
```

Firstly; we simply see what vales are the most different between the two classes (row 1 = death = 0, row 2 = death = 1)

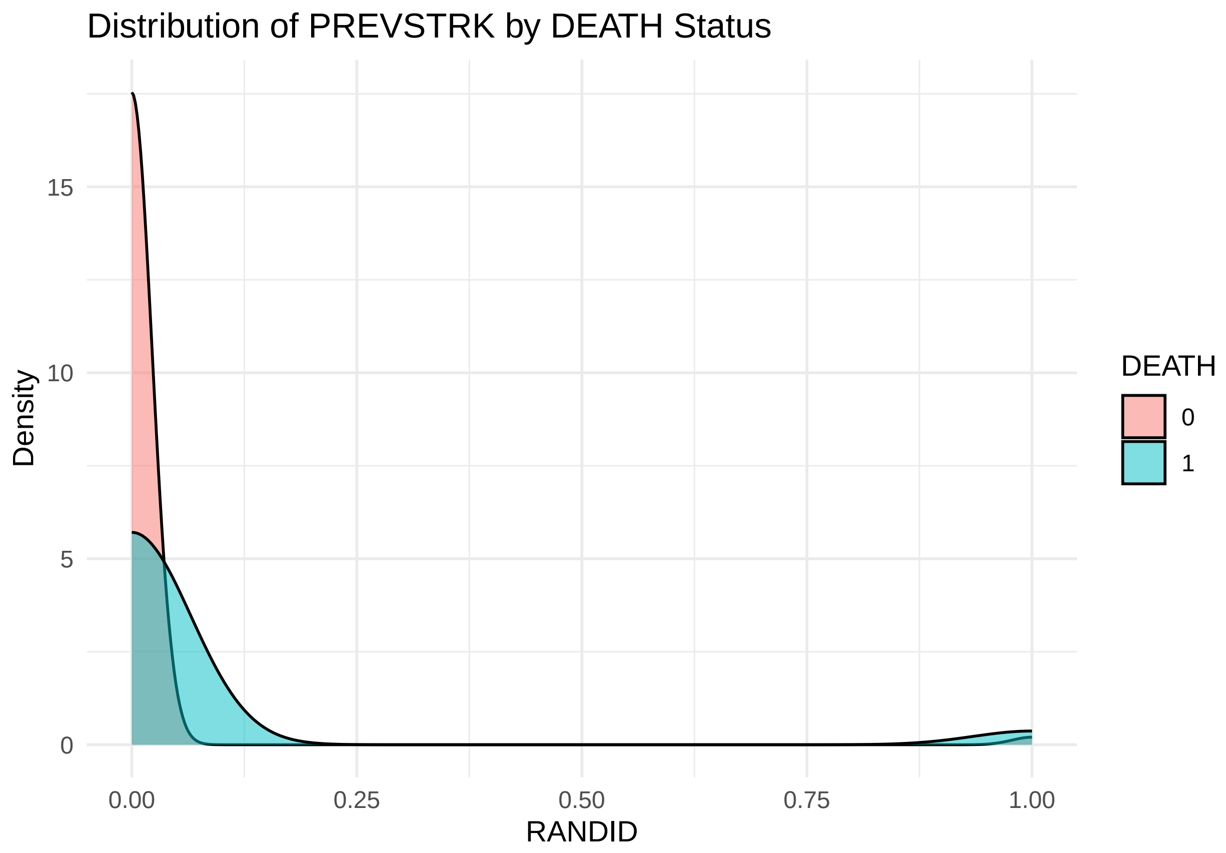
We identify the two values whose means are most differentiated by classification:

We see that these two values, particularly have signficant impact on the likelihood of death; that is Koalas with low MI_FHCD readings are much less at risk. We can discard the PREVSTRK death status.

```r
ggplot(df_train_clean, aes(x = MI_FCHD, fill = as.factor(DEATH))) +
  geom_density(alpha = 0.5) +  # Density plot with transparency
  labs(title = "Distribution of MI_FCHD by DEATH Status",
       x = "RANDID",
       y = "Density",
       fill = "DEATH") +
  theme_minimal()
```



```r
ggplot(df_train_clean, aes(x = PREVSTRK, fill = as.factor(DEATH))) +
  geom_density(alpha = 0.5) +  # Density plot with transparency
  labs(title = "Distribution of PREVSTRK by DEATH Status",
       x = "RANDID",
       y = "Density",
       fill = "DEATH") +
  theme_minimal()
```



This isn't really a strong indicator however; in fact we can fit a general L.R model as below. We don't find *any* particularly strong indicator vars (no P Values <0.05), in fact ALL are highly variable; atomically no individual explanatory variable indicates a high risk factor. Combinatorally this may not be the case. We could use some stepwise feature selection to remove features who are of no use / importance explanatorially.... The entire model summary is output below...

```
model <- glm(DEATH ~ ., data = df_train_clean, family = binomial())
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(model)
```

```
##
## Call:
## glm(formula = DEATH ~ ., family = binomial(), data = df_train_clean)
##
## Coefficients: (1 not defined because of singularities)
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.235e+04  2.458e+05   0.050    0.960
## RANDID      -8.150e-07  2.108e-04  -0.004    0.997
## SEX         -2.174e+00  1.165e+03  -0.002    0.999
## TOTCHOL     -1.462e-02  4.312e+01   0.000    1.000
## AGE          2.729e-01  9.338e+01   0.003    0.998
## SYSBP        1.912e-01  3.230e+01   0.006    0.995
## DIABP       -1.822e-01  4.267e+01  -0.004    0.997
## CURSMOKE     2.529e+00  1.782e+03   0.001    0.999
## CIGPDAY      5.132e-02  7.131e+01   0.001    0.999
## BMI         -6.367e-01  1.570e+02  -0.004    0.997
## DIABETES    -1.277e+01  5.252e+03  -0.002    0.998
## BPMEDS       1.333e+01  9.623e+02   0.014    0.989
## HEARTRTE     1.686e-01  3.955e+01   0.004    0.997
## GLUCOSE      1.134e-01  2.115e+01   0.005    0.996
## educ        -1.082e+00  5.383e+02  -0.002    0.998
## PREVCHD      3.200e+01  7.075e+03   0.005    0.996
## PREVAP       3.818e+00  4.896e+03   0.001    0.999
## PREVMI      -3.264e+01  7.437e+03  -0.004    0.996
## PREVSTRK    -2.672e+01  3.276e+05   0.000    1.000
## PREVHYP     -2.463e+01  3.600e+03  -0.007    0.995
## TIME         4.654e-02  7.805e+00   0.006    0.995
## PERIOD              NA         NA      NA       NA
## HDLC        -5.247e-02  6.728e+01  -0.001    0.999
## LDLC        -1.124e-01  4.344e+01  -0.003    0.998
## ANGINA      -1.032e+01  7.434e+03  -0.001    0.999
## HOSPMI       2.156e+02  1.012e+05   0.002    0.998
## MI_FCHD     -2.107e+02  1.012e+05  -0.002    0.998
## ANYCHD       2.344e+01  7.857e+03   0.003    0.998
## STROKE      -5.639e+00  2.391e+04   0.000    1.000
## CVD         -9.063e+00  7.902e+03  -0.001    0.999
## HYPERTEN    -1.182e+01  2.490e+03  -0.005    0.996
## TIMEAP      -1.129e-03  1.584e+00  -0.001    0.999
## TIMEMI       4.582e-02  2.634e+01   0.002    0.999
## TIMEMIFC    -5.244e-02  2.641e+01  -0.002    0.998
## TIMECHD      7.547e-03  2.116e+00   0.004    0.997
## TIMESTRK    -2.086e-03  3.717e+00  -0.001    1.000
## TIMECVD      5.607e-04  1.501e+00   0.000    1.000
## TIMEDTH     -1.431e+00  2.759e+01  -0.052    0.959
## TIMEHYP     -2.886e-03  4.522e-01  -0.006    0.995
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1.8429e+03  on 1778  degrees of freedom
## Residual deviance: 1.1842e-05  on 1741  degrees of freedom
## AIC: 76
##
## Number of Fisher Scoring iterations: 25
```

Predicting Colour names w/ Discriminant Analysis for RG values of RGB (i.e. g = 0 for all instances;)

```r
#rm(list = ls())
```

###Preamble

Data is given already as simply the R,B and a value of the set below describing it's colour (sans the G component; predicting the classification is done with QDA as follows…) Firstly; load the data; observe the features.

Note: I was unsuccessful in matching the names of colours (dynamically) to their areas in the visualisation we see later. I apologise for the trouble that that causes to read / interpret!

```
# Load the data
colordata <- read.csv("colors_train.csv", header = TRUE, sep = ",")

colordata <- as.data.frame(colordata)
num_unique_values <- length(unique(colordata$color))
cat("The dataset contains", num_unique_values, "classes.\n")
```

A

```
## The dataset contains 5 classes.
```

```
unique(colordata$color)
```

```
## [1] "pink"   "purple" "red"    "brown"  "blue"
```

```
unique_colors <- unique(colordata$color)
unique_colors
```

```
## [1] "pink"   "purple" "red"    "brown"  "blue"
```

```
train_size <- floor(0.8 * nrow(colordata))
train_ind <- sample(seq_len(nrow(colordata)), size = train_size)
df_train <- colordata[train_ind, ]
df_test <- colordata[-train_ind, ]
```

B Fit a QDA algorithm to the dataset for calssification…

Because we will generate some number of datapoints per class; we define the function as follows

```
#Assuming R doesn't need to count len of df each time....
# It's a tiny bit hard coded with our mean calcs as being explicitly for the R,B cols and not dynamic,
# But R is not a fun language to work with !
manual_qda <- function(classname, df, column) {
  number_of_occurances <- df %>% filter(df[[column]] == classname) %>% nrow()

  pi <- number_of_occurances / nrow(df)

  mean <- df %>%
    filter(df[[column]] == classname) %>%
    dplyr::select(-all_of(column)) %>%
    colMeans()
  sigma <- df %>% filter(df[[column]] == classname) %>% dplyr::select(r, b) %>% cov
  #because R
  return(list(pi = pi, mean = mean, sigma = sigma))
}
```

```
delta_no <- function(X, mean, sigma, pi) {
  return(-t(X - mean) %*% solve(sigma) %*% (X - mean) / 2 - log(det(sigma)) / 2 + log(pi))
}
```

With the definitions above; we will (still manually) call some values and store them into explicit vars per colour channel. A more appropriate implementation is some dict equivalent; but for this purpose (and for the sake of avoiding R's cumbersome language) we simply define these explicitly

```
qda_results <- list()
for (color in unique_colors) {
  qda_results[[color]] <- manual_qda(color, df_train, "color")
}
```

```
qda_results[1:2]
```

```
## $pink
## $pink$pi
## [1] 0.221875
##
## $pink$mean
##        r        b
## 215.7183 147.4225
##
## $pink$sigma
##          r         b
## r 530.1195  560.4207
## b 560.4207 2746.7903
##
##
## $purple
## $purple$pi
## [1] 0.353125
##
## $purple$mean
##        r        b
## 125.1416 153.0000
##
## $purple$sigma
##          r        b
## r 2139.373 2108.643
## b 2108.643 4169.268
```

```
calculate_deltas_old <- function(qda_results, X) {
  deltas <- list()

  for (color in names(qda_results)) {
    mean <- qda_results[[color]]$mean
    sigma <- qda_results[[color]]$sigma
    pi <- qda_results[[color]]$pi

    delta_value <- delta_no(X, mean, sigma, pi)

    deltas[[color]] <- delta_value
  }

  return(deltas)
}
```

```
calculate_deltas <- function(qda_results, X) {
  deltas <- sapply(names(qda_results), function(color) {
    mean <- qda_results[[color]]$mean
    sigma <- qda_results[[color]]$sigma
    pi <- qda_results[[color]]$pi

    delta_no(X, mean, sigma, pi)
  })

  max_label <- names(deltas)[which.max(deltas)]

  return(max_label)  # Just return the label with the highest delta value
}
QDA_test_res <- df_test
QDA_test_res <- QDA_test_res %>%
  rowwise() %>%
  mutate(prediction = calculate_deltas(qda_results, c(r, b))) %>%
  ungroup()
```

```
QDA_test_res
```

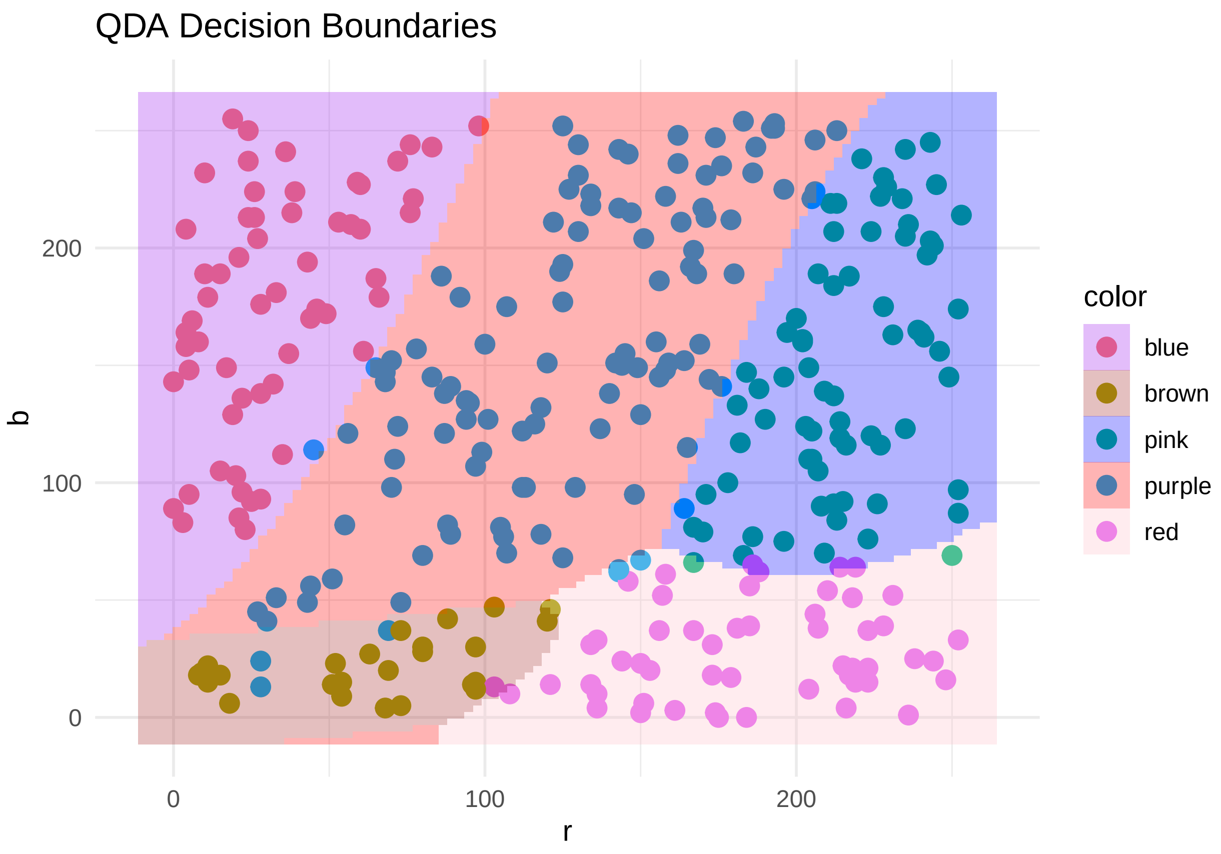
```
## # A tibble: 80 x 4
##        r     b color  prediction
##    <int> <int> <chr>  <chr>
##  1   159   227 purple purple
##  2   234   157 pink   pink
##  3    76   115 purple purple
##  4    80   164 purple purple
##  5    40   173 blue   blue
##  6   117    56 brown  purple
##  7   129   149 purple purple
##  8   197    29 red    red
##  9    13    58 blue   blue
## 10   201   116 pink   pink
## # i 70 more rows
```

We now have a useful, reproducable way to query all members of the class. . . .

```
X <- c(100, 100)  # Replace with R and B as needed
delta_results <- calculate_deltas(qda_results, X)
print(delta_results)
```

```
## [1] "purple"
```

```r
# Create grid for visualization
r_range <- seq(min(df_train$r) - 10, max(df_train$r) + 10, length.out = 100)
b_range <- seq(min(df_train$b) - 10, max(df_train$b) + 10, length.out = 100)
grid <- expand.grid(r = r_range, b = b_range)

# Predict class labels for the grid points
grid$color <- sapply(seq_len(nrow(grid)), function(i) {
  calculate_deltas(qda_results, as.numeric(grid[i, ]))
})

# Plot decision boundaries
ggplot() +
  geom_point(data = df_train, aes(x = r, y = b, color = color), size = 3) +
  geom_raster(data = grid, aes(x = r, y = b, fill = color), alpha = 0.3) +
  scale_fill_manual(values = unique(df_train$color)) +
  labs(title = "QDA Decision Boundaries", x = "r", y = "b") +
  theme_minimal()
```



Noting the above: Some difficulty was encountered in translating the label names for the named list (names of colours) into the colour regions on the graph.

c:

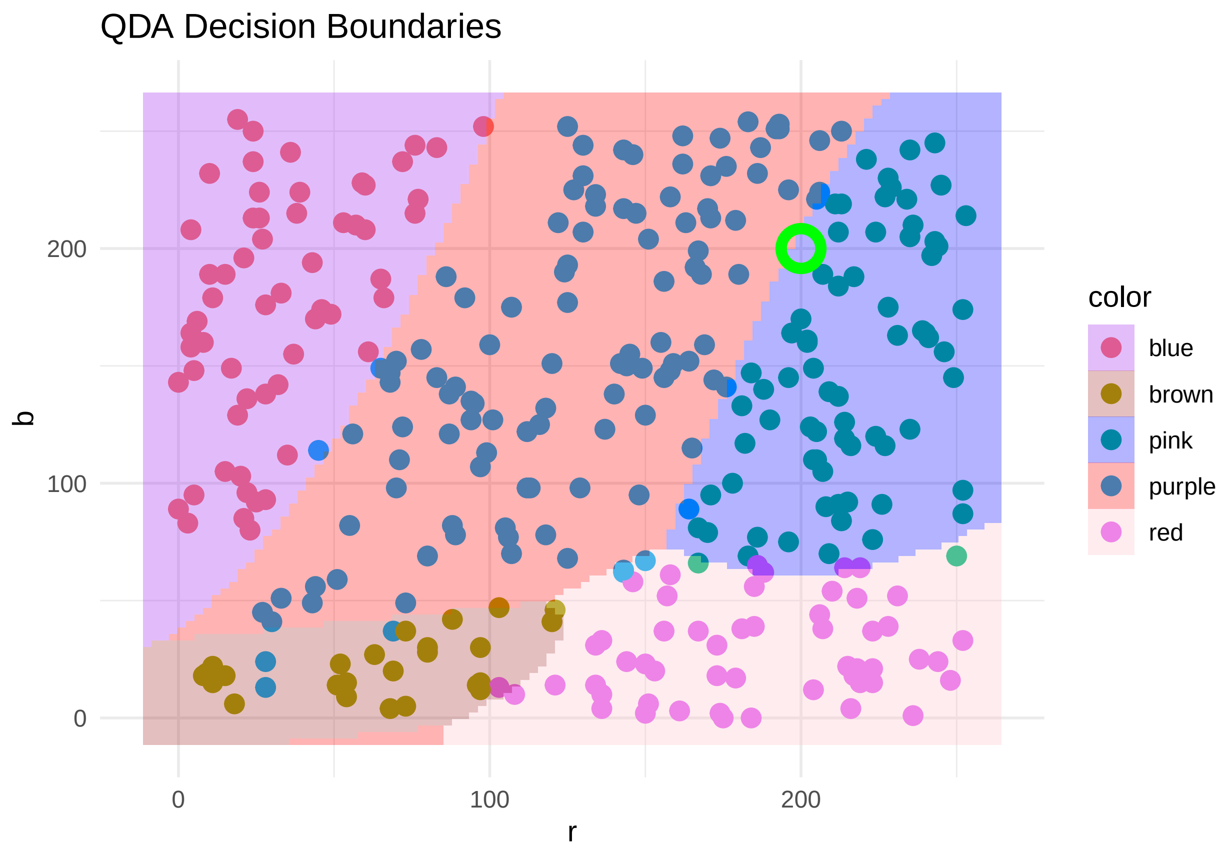
We wish to query a specific point. . . .

Specificially:

R = 200 G = 0 B = 200, which we see on the graph in green:

```r
test_R = 200
test_B = 200
```

```r
ggplot() +
  geom_point(data = df_train, aes(x = r, y = b, color = color), size = 3) +
  geom_raster(data = grid, aes(x = r, y = b, fill = color), alpha = 0.3) +
  scale_fill_manual(values = unique(df_train$color)) +
    geom_point(aes(x = test_R, y = test_B), color = "green", size = 5, shape = 1, stroke = 3) +  # High
  labs(title = "QDA Decision Boundaries", x = "r", y = "b") +
  theme_minimal()
```



Testing our model on the point 200,0,200 and indeed it matches 'pink' as per the display above.

Our algorithm defines this as pink.

```r
sample <- calculate_deltas(qda_results, c(test_R, test_B))
sample
```

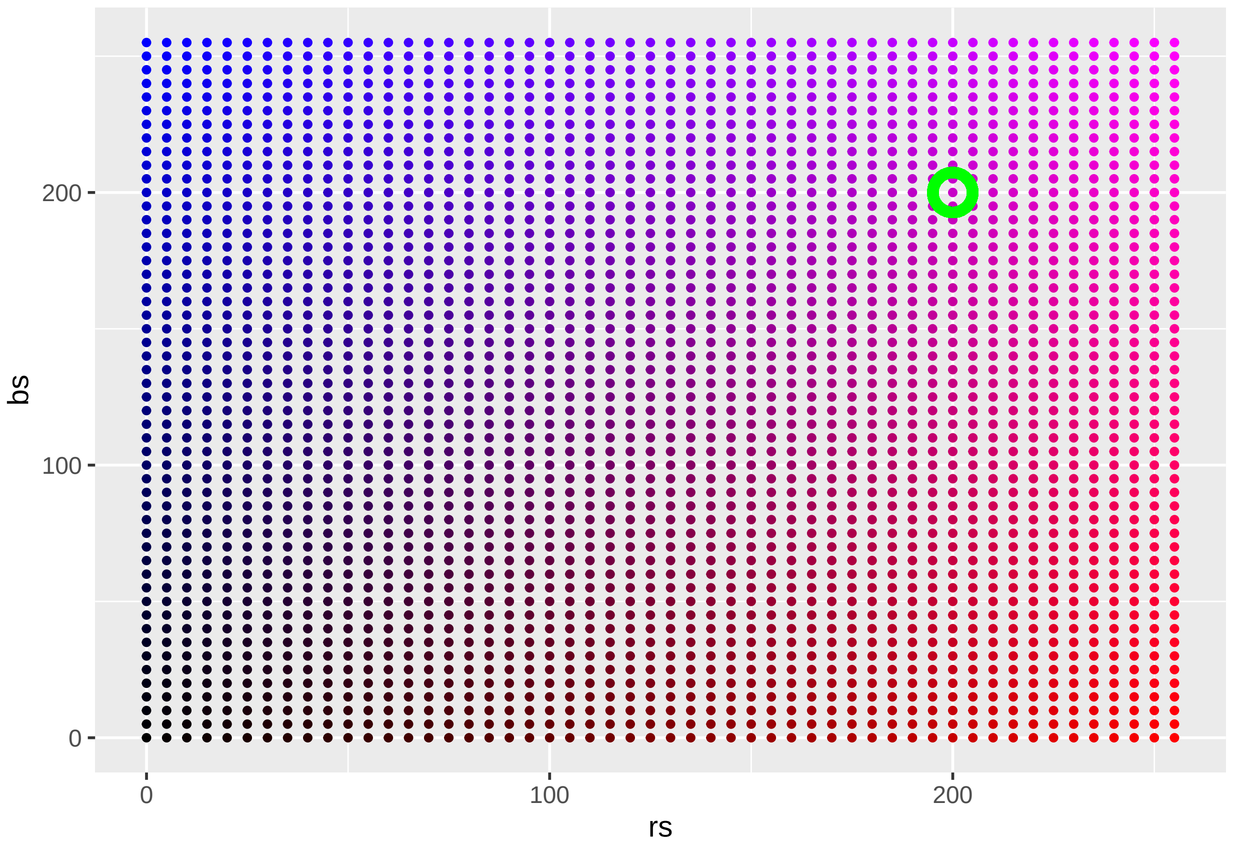
```
## [1] "pink"
```

for a quick comparison:

```r
rs <- seq(0,256,5)
bs <- seq(0,256,5)
df_plot_colors <- data.frame(rs = rs, bs=bs) %>% tidyr::expand(rs, bs)
```

```r
ggplot(data=df_plot_colors) +
  geom_point(aes(x=rs, y=bs, color=rgb(rs/256,0,bs/256)), size=1)+# R's rgb code works with numbers bet
  geom_point(aes(x = test_R, y = test_B), color = "green", size = 5, shape = 1, stroke = 3) +  # Highli
  scale_color_identity() +
  theme(legend.position = "none")
```

```
## Warning in geom_point(aes(x = test_R, y = test_B), color = "green", size = 5, : All aesthetics have l
## i Please consider using ‘annotate()‘ or provide this layer with data containing
##   a single row.
```



And indeed; looking to the original data graphed, we can see that ‘pink’ is a fair prediction

d: Knn Classification:

An alternative approach to predicting values for unseen data is K-Nearest-Neighbours classification. Intuitively; we are looking for some number of neighbours to the query point based on some distance metric.

KNN-Classification in it’s regular case, simply will return the most often seen class of the k selections. We call this majority voting. Intuitioniistically we must consider cases where there are ties in the votes. One option is simply increasing K until ties are mitigated. Another is to randomly choose a class of the tied pool. We will use weighted voting. i.e. the ‘closer’ the datapoint, the more important it is. This is trivially implemented by averaging the distances between each of the class points.

We use R’s inbuilt function for this; which implements knn majority voting,

Build a KNN model on df_train…

```
k <- 5
knn_test_res <- df_test
# Convert color to a factor for knn
df_train$color <- as.factor(df_train$color)
df_test$color <- as.factor(df_test$color)

train_features <- df_train %>% dplyr::select(r, b)
train_labels <- df_train$color
test_features <- df_test %>% dplyr::select(r, b)


knn_predictions <- knn(train = train_features,
                       test = test_features,
                       cl = train_labels,
                       k = k)

knn_test_res <- knn_test_res %>%
  mutate(prediction = knn_predictions)
```

```
misclassified_QDA <- QDA_test_res %>%
  filter(prediction != color) %>%
  nrow()
misclassified_KNN <- knn_test_res %>%
  filter(prediction != color) %>%
  nrow()
total <- nrow(df_test)
misclassification_rate_KNN <- misclassified_KNN / total
misclassification_rate_QDA <- misclassified_QDA / total
misclassification_rate_KNN
```

```
## [1] 0.0375
```

```
misclassification_rate_QDA
```

```
## [1] 0.0375
```

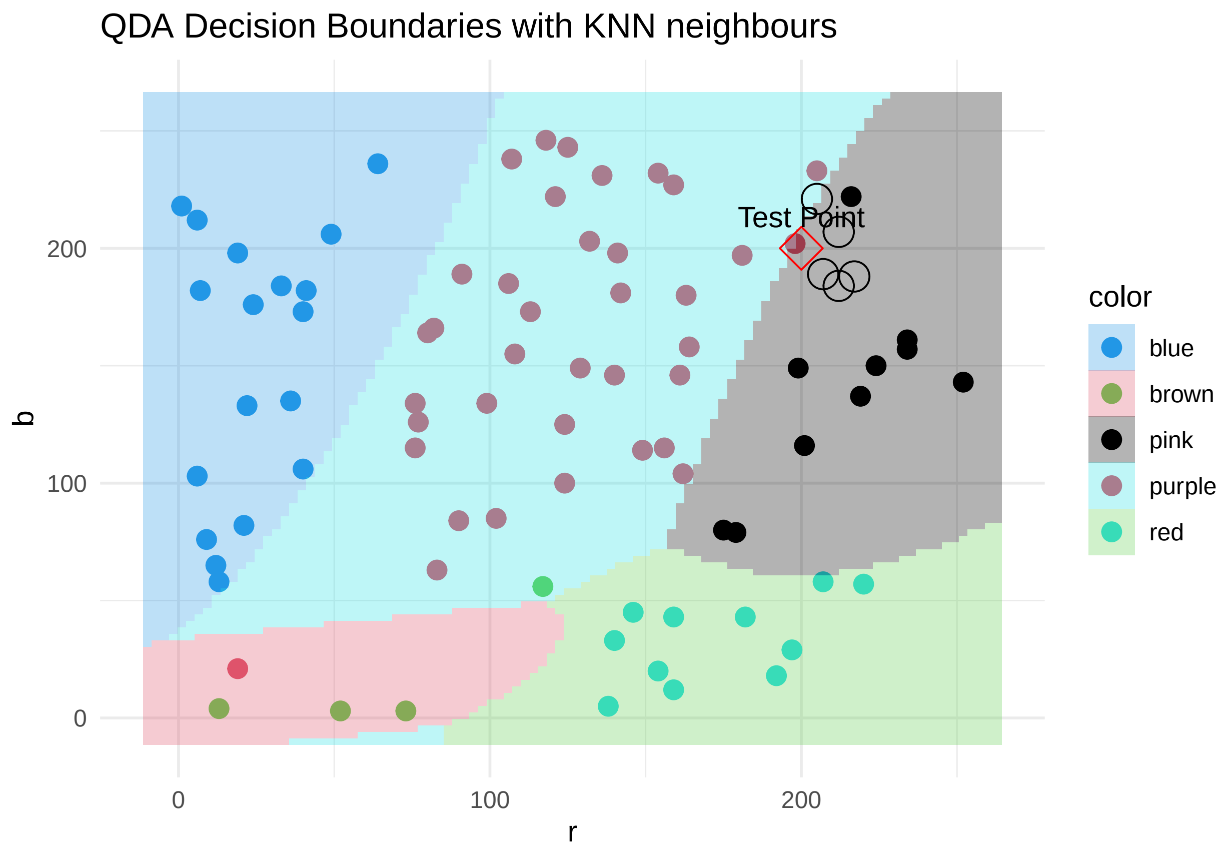
Convieniently; our data is in 2space; sampling from KNN is convient to think about; in fact it's very clear how just adding more points to the dataset would improve our knn model somewaht irrespective of their distribution…

At $k = 5$ our misclassification rate precisely matches between QDA and KNN. (increasing K to a point where the sampling is across colour areas)

Remembering our test point which classified as *pink* with qda… We can manually compute the distances on our test locations to get an idea of the *window* that knn 'creates

```
#same as before
test_point <- data.frame(r = test_R, b = test_B)
knn_prediction <- knn(train = train_features,
                      test = test_point,
                      cl = train_labels,
                      k = k)
```

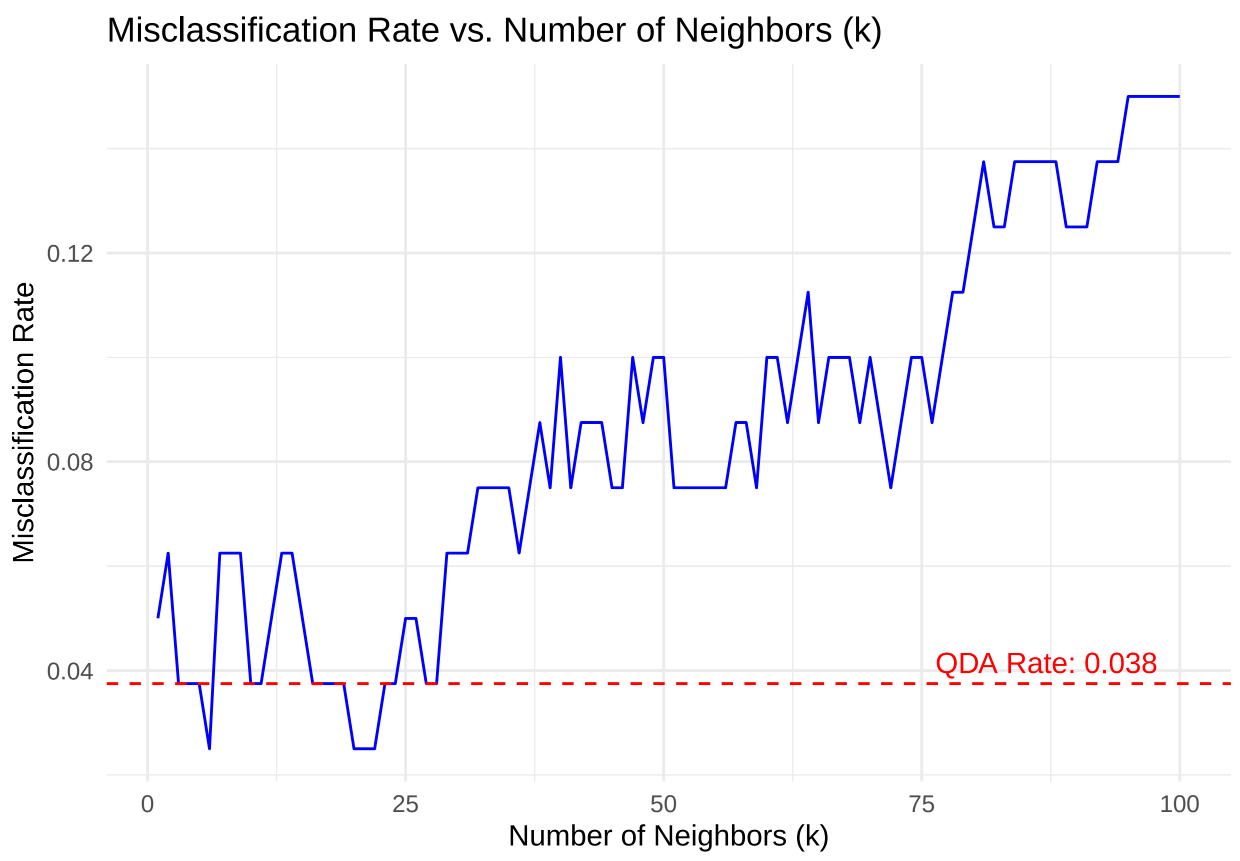
```r
distances <- as.matrix(dist(rbind(train_features, test_point)))
n_train <- nrow(train_features)
test_distances <- distances[(n_train + 1), 1:n_train]

sorted_indices <- order(test_distances)
nearest_indices <- sorted_indices[1:k]

neighbors_values <- train_features[nearest_indices, ]
neighbors_labels <- train_labels[nearest_indices]

neighbors_df <- data.frame(r = neighbors_values$r,
                           b = neighbors_values$b,
                           color = neighbors_labels)

neighbor_color <- "black"

# Plot using ggplot2
ggplot() +
  geom_point(data = df_test, aes(x = r, y = b, color = color), size = 3) +
  geom_raster(data = grid, aes(x = r, y = b, fill = color), alpha = 0.3) +
  scale_fill_manual(values = unique(df_train$color)) +
  annotate("point", x = test_R, y = test_B, color = "red", size = 5, shape = 5) +
  annotate("text", x = test_R, y = test_B, label = "Test Point", vjust = -1, color = "Black") +
  geom_point(data = neighbors_df, aes(x = r, y = b), color = neighbor_color, shape = 1, size = 5, show.
  scale_color_manual(values = unique(df_test$color)) +
  labs(title = "QDA Decision Boundaries with KNN neighbours", x = "r", y = "b") +
  theme_minimal()
```



Above we see the *test* point; which evaluates to pink for qda and for knn. Note that the KNN locations of our data (df test is graphed, but these originate in df_train..) are shown as open circles, and covnieneintly, on this graph, we can simply use the majority fill of the colour in that open circle as an identifier (for human readability).

We can

```r
results <- list()
k_values <- 1:100

for (k in k_values) {

  knn_predictions <- knn(train = train_features,
                         test = test_features,
                         cl = train_labels,
                         k = k)

  knn_test_res <- df_test %>%
    mutate(prediction = knn_predictions)

  misclassification_rate <- mean(knn_test_res$color != knn_test_res$prediction)
  results[[as.character(k)]] <- misclassification_rate
}

# Convert results to a dataframe for ggplot
results_df <- data.frame(
  k = as.integer(names(results)),
  misclassification_rate = unlist(results)
)



ggplot(results_df, aes(x = k, y = misclassification_rate)) +
  geom_line(color = "blue") +
    geom_hline(yintercept = misclassification_rate_QDA, linetype = "dashed", color = "red") +
  labs(title = "Misclassification Rate vs. Number of Neighbors (k)",
       x = "Number of Neighbors (k)",
       y = "Misclassification Rate") +
  annotate("text", x = max(results_df$k), y = misclassification_rate_QDA,
           label = paste("QDA Rate:", round(misclassification_rate_QDA, 3)),
           hjust = 1.1, vjust = -0.5, color = "red") +
  theme_minimal()
```



It’s notable that with all but the 14 and 17 values; qda obtains a better misclassification rate (that is, where any class other than the correct was chosen).

At times, in this document, we have mentioned ‘colour zones’; which are reflected in the decision boundaries in our qda analysis, but are also prevalent in the source data. As K increases, the likelihood of sampling in multiple ‘zones’ also increases; and cases of voting are more common. We would expect that k-nn would be particularly good (see accurate) on average, closer to the ‘center’ of a cluster of these colours.

QDA is the better choice here, given our data. In terms of computability neither offers a particularly invasive overhead; asymptotically; neither is of concern (we assume R isn’t simply sorting the entire dataset on each point)

QDA assumes that the data for each class follows a Gaussian distribution with a different covariance matrix for each class, which means it performs well when the classes have distinct and well-defined covariance structures. This makes QDA the most suitable for situations where we can reasonably expect the data to be normally distributed within each class and where the classes have different spread or orientation.

KNN on the other hand doesn't make any of these assumptions, and as we've seen just considers neighbours.

We can make asumptions about the data from intuition behind our perception of colour (as per the dots above) and it holds that QDA is the best choice here.

Were the data more 'randomly' distributed, KNN would be a better choice; as would be the case in some higher dimension.