# Part 3 - Classification

Ibrahim Yazici

## Load packages

We use the `tidyverse` suite of packages.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

## Read data

The code chunk below reads in the final project data.

```
df <- readr::read_csv("paint_project_train_data.csv", col_names = TRUE)
```

```
## Rows: 835 Columns: 8
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): Lightness, Saturation
## dbl (6): R, G, B, Hue, response, outcome
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

The `readr::read_csv()` function displays the data types and column names associated with the data. However, a glimpse is shown below that reveals the number of rows and also shows some of the representative values for the columns.

```
df %>% glimpse()
```

The data consist of continuous and categorical inputs. The `glimpse()` shown above reveals the data type for each variable which state to you whether the input is continuous or categorical. The RGB color model inputs, `R`, `G`, and `B` are continuous (dbl) inputs. The HSL color model inputs consist of 2 categorical inputs, `Lightness` and `Saturation`, and a continuous input, `Hue`. Two outputs are provided. The continuous output, `response`, and the Binary output, `outcome`. However, the data type of the Binary outcome is numeric because the Binary `outcome` is **encoded** as `outcome = 1` for the EVENT and `outcome = 0` for the NON-EVENT.

## Binary classification task

The Binary output variable, `outcome`, is a numeric variable.

```
df %>% pull(outcome) %>% class()
```

```
## [1] "numeric"
```

However, there are **only** two unique values for `outcome`.

```
df %>% count(outcome)
```

```
## # A tibble: 2 x 2
##   outcome     n
##     <dbl> <int>
## 1       0   644
## 2       1   191
```

As stated previously, `outcome = 1` denotes the **EVENT** while `outcome = 0` denotes the **NON-EVENT**. Thus, the `outcome` variable uses the 0/1 encoding! This encoding is appropriate for `glm()` and the functions we create in homework assignments, and lecture examples. However, `caret` and `tidymodels` prefer a different encoding. For those reasons, two different binary classification data sets are defined. The first will be used for Parts iiiA) and iiiB) while the second will be used for iiiD).

The data set associated with iiiA) and iiiB) is created below. It removes the `response` variable so that way we can focus on the inputs and binary outcome.

```
dfiiiA <- df %>%
  select(-response)
```

The data set associated with iiiD) changes the data type of the `outcome` variable. The `ifelse()` function is used to convert `outcome` to a character data type. The value of `outcome = 1` is converted to the string `'event'` and the value of `outcome = 0` is converted to `'non_event'`. The `outcome` data type is then converted to a factor (R's categorical variable data type) with `'event'` forced as the first level.

```
dfiiiD <- df %>%
  select(-response) %>%
  mutate(outcome = ifelse(outcome == 1, 'event', 'non_event'),
         outcome = factor(outcome, levels = c('event', 'non_event')))
```

By converting `outcome` to a factor, the unique values of the variables are "always known":

```
dfiiiD %>% pull(outcome) %>% levels()
```

```
## [1] "event"     "non_event"
```

However, the value counts are the same as the original encoding.

```
dfiiiD %>% count(outcome)
```

```
## # A tibble: 2 x 2
##   outcome       n
##   <fct>     <int>
## 1 event       191
## 2 non_event   644
```

We standardize the continuous inputs `R`, `G`, `B`, and `Hue` below.

```
ready_dfiiiA <- dfiiiA
ready_dfiiiA$R <- scale(ready_dfiiiA$R, center = TRUE, scale = TRUE)
ready_dfiiiA$R <- as.vector(ready_dfiiiA$R)
ready_dfiiiA$G <- scale(ready_dfiiiA$G, center = TRUE, scale = TRUE)
```

```
ready_dfiiiA$G <- as.vector(ready_dfiiiA$G)
ready_dfiiiA$B <- scale(ready_dfiiiA$B, center = TRUE, scale = TRUE)
ready_dfiiiA$B <- as.vector(ready_dfiiiA$B)
ready_dfiiiA$Hue <- scale(ready_dfiiiA$Hue, center = TRUE, scale = TRUE)
ready_dfiiiA$Hue <- as.vector(ready_dfiiiA$Hue)
```

We will use the standardized dataset `ready_dfiiiA` in our classification models in parts A) and B) below.

### A) Linear Models

1) Model1: Intercept Only Model (No Inputs):

```
mod_3A_1 <- glm(outcome ~ 1, data = ready_dfiiiA, family = "binomial")
```

2) Model2: Categorical Variables Only (Linear Additive):

```
mod_3A_2 <- glm(outcome ~ Lightness + Saturation, data = ready_dfiiiA, family = "binomial")
```

3) Model3: Continuous Variables Only (Linear Additive):

```
mod_3A_3 <- glm(outcome ~ R + G + B + Hue, data = ready_dfiiiA, family = "binomial")
```

4) Model4: All Categorical and Continuous Variables (Linear Additive):

```
mod_3A_4 <- glm(outcome ~ R + G + B + Hue + Lightness + Saturation, data = ready_dfiiiA, family = "bino
```

5) Model5: Interaction of Categorical Inputs with All Continuous Inputs (Main Effects):

```
mod_3A_5 <- glm(outcome ~ (R + G + B + Hue) * (Lightness + Saturation), data = ready_dfiiiA, family = "
```

6) Model6: Add Categorical Inputs to All Main Effect and All Pairwise Interactions of Continuous Inputs:

```
mod_3A_6 <- glm(outcome ~ (R + G + B + Hue)^2 + Lightness + Saturation, data = ready_dfiiiA, family = "
```

7) Model7: Interaction of Categorical Inputs with All Main Effects and All Pairwise Interactions of Continuous Inputs:

```
mod_3A_7 <- glm(outcome ~ (R + G + B + Hue)^2 * (Lightness + Saturation), data = ready_dfiiiA, family =
```

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

8) Model8: Add Categorical Inputs to 3 degree-of-freedom natural (DOF) spline from continuous variables:

```
mod_3A_8 <- glm(outcome ~ splines::ns(R, 3) + splines::ns(G, 3) + splines::ns(B, 3) + splines::ns(Hue, 
```

9) Model9: Add Categorical Inputs to Interactions from 3 DOF spline from input R and All Pairwise Interactions of Continuous Inputs G, B, Hue:

```
mod_3A_9 <- glm(outcome ~ splines::ns(R, 3) * (G + B + Hue)^2 + Lightness + Saturation, data=ready_dfii
```

10) Model10: Interact Categorical Inputs to Interactions from 3 DOF spline from input R Interactions of Continuous Inputs G, B, Hue:

```
mod_3A_10 <- glm(outcome ~ splines::ns(R, 3) * (G + B + Hue), data=ready_dfiiiA, family = "binomial")
```

Here we note that the following warnings occured when we fit Model 7: "## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred"

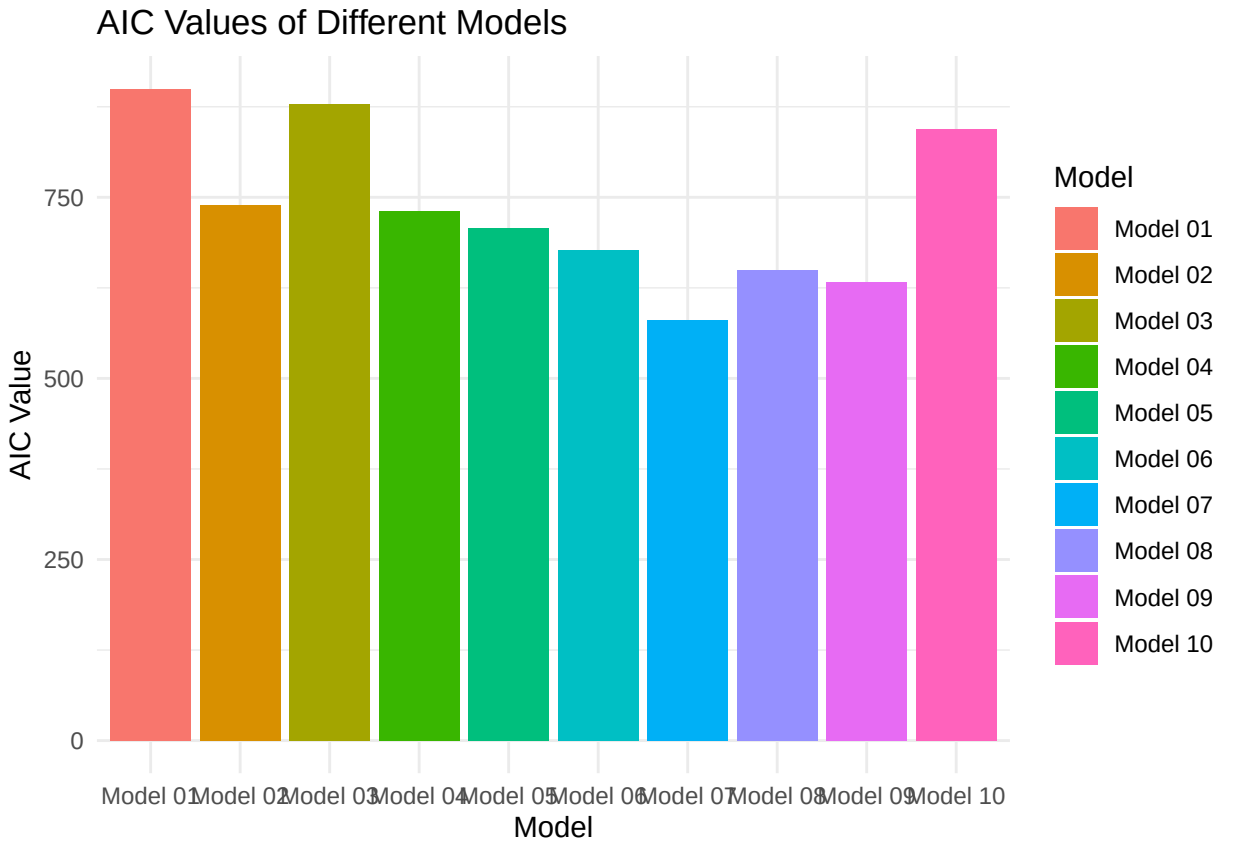
The following code chunk calculates AIC values of the models.

```
aic_model1 <- AIC(mod_3A_1)
aic_model2 <- AIC(mod_3A_2)
aic_model3 <- AIC(mod_3A_3)
```

```r
aic_model4 <- AIC(mod_3A_4)
aic_model5 <- AIC(mod_3A_5)
aic_model6 <- AIC(mod_3A_6)
aic_model7 <- AIC(mod_3A_7)
aic_model8 <- AIC(mod_3A_8)
aic_model9 <- AIC(mod_3A_9)
aic_model10 <- AIC(mod_3A_10)

aic_values <- data.frame(
  Model = c("Model 01", "Model 02", "Model 03", "Model 04", "Model 05", "Model 06", "Model 07", "Model 
  AIC = c(aic_model1, aic_model2, aic_model3, aic_model4, aic_model5, aic_model6, aic_model7, aic_model8
)

ggplot(aic_values, aes(x = Model, y = AIC, fill = Model)) +
  geom_bar(stat = "identity") +
  theme_minimal() +
  labs(title = "AIC Values of Different Models", x = "Model", y = "AIC Value")
```



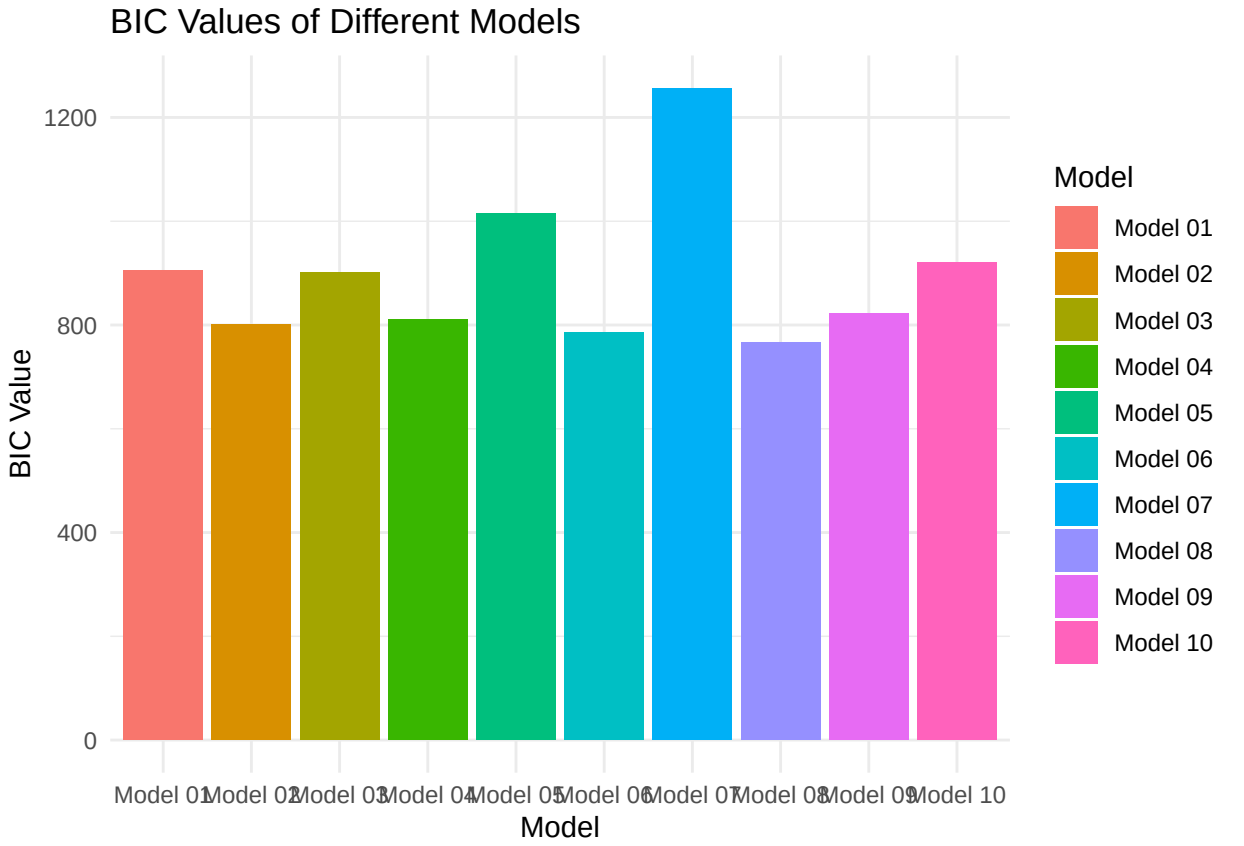

The following code chunk calculates BIC values of the models.

```r
bic_model1 <- BIC(mod_3A_1)
bic_model2 <- BIC(mod_3A_2)
bic_model3 <- BIC(mod_3A_3)
bic_model4 <- BIC(mod_3A_4)
bic_model5 <- BIC(mod_3A_5)
bic_model6 <- BIC(mod_3A_6)
bic_model7 <- BIC(mod_3A_7)
```

```r
bic_model8 <- BIC(mod_3A_8)
bic_model9 <- BIC(mod_3A_9)
bic_model10 <- BIC(mod_3A_10)

bic_values <- data.frame(
  Model = c("Model 01", "Model 02", "Model 03", "Model 04", "Model 05", "Model 06", "Model 07", "Model 
  BIC = c(bic_model1, bic_model2, bic_model3, bic_model4, bic_model5, bic_model6, bic_model7, bic_model8
)

ggplot(bic_values, aes(x = Model, y = BIC, fill = Model)) +
  geom_bar(stat = "identity") +
  theme_minimal() +
  labs(title = "BIC Values of Different Models", x = "Model", y = "BIC Value")
```




We will use BIC metric to determine the best model. We will not choose `Model 2` in out list because its AIC metric performance does not look good enough.

Here are top 3 models:
1) `Model 8`
2) `Model 6`
3) `Model 4`

Here is the Coefficient summary visualization for `Model 8`.

```r
coefplot::coefplot(mod_3A_8) +
theme_bw()
```

We can list significant inputs for `Model 8` below.

```
tidy_mod8 <- broom::tidy(mod_3A_8, eval = FALSE)
significant_inputs_mod8 <- tidy_mod8 %>% filter(p.value <0.05)
significant_inputs_mod8
```

```
## # A tibble: 14 x 5
##    term                  estimate std.error statistic  p.value
##    <chr>                    <dbl>     <dbl>     <dbl>    <dbl>
##  1 splines::ns(R, 3)1      -3.02     1.11      -2.72 6.60e- 3
##  2 splines::ns(R, 3)3      -3.63     1.41      -2.57 1.02e- 2
##  3 splines::ns(G, 3)1      -3.28     1.57      -2.09 3.69e- 2
##  4 splines::ns(G, 3)2     -16.1      3.55      -4.53 5.86e- 6
##  5 splines::ns(B, 3)1       4.48     1.28       3.49 4.76e- 4
##  6 splines::ns(B, 3)2       8.18     3.09       2.65 8.12e- 3
##  7 splines::ns(B, 3)3       4.48     1.42       3.15 1.65e- 3
##  8 splines::ns(Hue, 3)2     6.38     1.13       5.65 1.59e- 8
##  9 splines::ns(Hue, 3)3    -1.89     0.622     -3.04 2.33e- 3
## 10 Saturationgray           4.63     0.594      7.79 6.91e-15
## 11 Saturationneutral        2.44     0.514      4.76 1.94e- 6
## 12 Saturationpure           1.20     0.553      2.17 3.02e- 2
## 13 Saturationshaded         1.99     0.520      3.82 1.32e- 4
## 14 Saturationsubdued        1.64     0.528      3.10 1.94e- 3
```

Here is the Coefficient summary visualization for `Model 6`.

```
coefplot::coefplot(mod_3A_6) +
theme_bw()
```

We can list significant inputs for `Model 6` below.

```
tidy_mod6 <- broom::tidy(mod_3A_6)
significant_inputs_mod6 <- tidy_mod6 %>% filter(p.value <0.05)
significant_inputs_mod6
```

Here is the Coefficient summary visualization for `Model 4`.

```
coefplot::coefplot(mod_3A_4) +
theme_bw()
```

We can list significant inputs for `Model 4` below.

```
tidy_mod4 <- broom::tidy(mod_3A_4)
significant_inputs_mod4 <- tidy_mod4 %>% filter(p.value <0.05)
significant_inputs_mod4
```

### B) Bayesian Linear Models

I will fit the best model (Model 8) and second best model (Model 6) we fit in part A). The reason I pick Model 6 is that I want to compare the two models here again and compare the results with part A).

Here is the design matrix and required information for `Model 8` in Bayesian case.

```
Xmat_8 <- model.matrix(~ splines::ns(R, 3) + splines::ns(G, 3) + splines::ns(B, 3) + splines::ns(Hue, 3

info_8 <- list(
  yobs = ready_dfiiiA$outcome,
  design_matrix = Xmat_8,
  mu_beta = 0,
```

```
  tau_beta = 4.5
)
```

Here is the design matrix and required information for `Model 6` in Bayesian case.

```
Xmat_6 <- model.matrix(~ (R + G + B + Hue)^2 + Lightness + Saturation, data = ready_dfiiiA)

info_6 <- list(
  yobs = ready_dfiiiA$outcome,
  design_matrix = Xmat_6,
  mu_beta = 0,
  tau_beta = 4.5
)
```

We define the log-posterior function by completing the code chunk below.

```
logistic_logpost <- function(unknowns, my_info) {
  # extract the design matrix and assign to X
  X <- my_info$design_matrix

  # calculate the linear predictor
  eta <- X %*% unknowns

  # calculate the event probability
  mu <- boot::inv.logit(eta)

  # evaluate the log-likelihood
  log_lik <- sum(dbinom(x = my_info$yobs, size = 1, prob = mu, log = TRUE))

  # evaluate the log-prior
  log_prior <- sum(dnorm(unknowns, mean = my_info$mu_beta, sd = my_info$tau_beta, log = TRUE))

  # sum together
  log_lik + log_prior
}
```

We define the `my_laplace()` function is defined for you in the code chunk below.

```
my_laplace <- function(start_guess, logpost_func, ...)
{
  # code adapted from the `LearnBayes`` function `laplace()`
  fit <- optim(start_guess,
               logpost_func,
               gr = NULL,
               ...,
               method = "BFGS",
               hessian = TRUE,
               control = list(fnscale = -1, maxit = 5001))

  mode <- fit$par
  post_var_matrix <- -solve(fit$hessian)
  p <- length(mode)
  int <- p/2 * log(2 * pi) + 0.5 * log(det(post_var_matrix)) + logpost_func(mode, ...)
  # package all of the results into a list
  list(mode = mode,
       var_matrix = post_var_matrix,
```

```
       log_evidence = int,
       converge = ifelse(fit$convergence == 0,
                         "YES",
                         "NO"),
       iter_counts = as.numeric(fit$counts[1]))
}
```

We execute the Laplace Approximation for the Model 8 formulation and the Model 6 formulation.

```
laplace_8 <- my_laplace(rep(0, ncol(Xmat_8)), logistic_logpost, info_8)
laplace_8$converge
```

```
## [1] "YES"
```

```
laplace_6 <- my_laplace(rep(0, ncol(Xmat_6)), logistic_logpost, info_6)
laplace_6$converge
```

```
## [1] "YES"
```

We use the Bayes Factor to compare the models. We can conclude that `Model 8` is the better of the models.

```
exp(laplace_8$log_evidence)/exp(laplace_6$log_evidence)
```

```
## [1] 1723939143
```

A function is defined in the code chunk below. This function creates a coefficient summary plot in the style of the `coefplot()` function, but uses the Bayesian results from the Laplace Approximation.

```
viz_post_coefs <- function(post_means, post_sds, xnames)
{
  tibble::tibble(
    mu = post_means,
    sd = post_sds,
    x = xnames
  ) %>%
    mutate(x = factor(x, levels = xnames)) %>%
    ggplot(mapping = aes(x = x)) +
    geom_hline(yintercept = 0, color = 'grey', linetype = 'dashed') +
    geom_point(mapping = aes(y = mu)) +
    geom_linerange(mapping = aes(ymin = mu - 2 * sd,
                                 ymax = mu + 2 * sd,
                                 group = x)) +
    labs(x = 'feature', y = 'coefficient value') +
    coord_flip() +
    theme_bw()
}
```

We create the posterior summary visualization figure for our best model, `Model 8`.

```
post_means_8 <- laplace_8$mode
post_sds_8 <- sqrt(diag(laplace_8$var_matrix))
feature_names_8 <- colnames(Xmat_8)

viz_post_coefs(post_means_8, post_sds_8, feature_names_8)
```

We can easily observe that coefficient plot for `Model 8` above and the corresponding plot in Part A) for `Model 8` have similar values and intervals for inputs.

### C) Linear models Predictions

We will make predictions with our 2 selected linear models, `Model 8` and `Model 6` in order to visualize the trends of the event probability with respect to the inputs. We will use non Bayesian models for the predictions.

As a reminder,

`Model 8` is: `mod_3A_8 <- glm(outcome ~ splines::ns(R, 3) + splines::ns(G, 3) + splines::ns(B, 3) + splines::ns(Hue, 3) + Lightness + Saturation, data=ready_dfiiiA, family = "binomial")`

and

`Model 6` is: `mod_3A_6 <- glm(outcome ~ (R + G + B + Hue)^2 + Lightness + Saturation, data = ready_dfiiiA, family = "binomial")`

Our primary input will be `R` and secondary input will be `Hue`. We decide the reference values to use for the remaining inputs.

The `plogis()` function in R is the logistic function, which is the inverse of the logit function. We will use it to calculate the confidence intervals.

The following code chunk gives predictions for `Model 8` with Confidence Intervals, when Lightness = "saturated" and Saturation = "subdued".

```
primary_seq <- seq(min(ready_dfiiiA$R), max(ready_dfiiiA$R), length.out = 100)

prediction_data <- expand.grid(
  R = primary_seq,
  G = mean(ready_dfiiiA$G),
  B = mean(ready_dfiiiA$B),
  Hue = seq(min(ready_dfiiiA$Hue), max(ready_dfiiiA$Hue), length.out = 6),
  Lightness = "saturated",
  Saturation = "subdued"
)

preds <- predict(mod_3A_8, newdata = prediction_data, type = "link", se.fit = TRUE)

link_lwr <- preds$fit - 2 * preds$se.fit
link_upr <- preds$fit + 2 * preds$se.fit

prediction_data$prob <- plogis(preds$fit)
prediction_data$prob_lwr <- plogis(link_lwr)
prediction_data$prob_upr <- plogis(link_upr)

ggplot(prediction_data, aes(x = R, y = prob)) +
  geom_line() +
  geom_ribbon(aes(ymin = prob_lwr, ymax = prob_upr), fill = "blue", alpha = 0.2) +
  facet_wrap(~Hue, scales = "free_x") +
  labs(title = "Model 8: Predictive Probability Trend with Confidence Intervals",
       x = "R", y = "Predicted Probability") +
  theme_minimal()
```

The following code chunk gives predictions for `Model 6` with Confidence Intervals, when Lightness = "saturated" and Saturation = "subdued".

```
primary_seq <- seq(min(ready_dfiiiA$R), max(ready_dfiiiA$R), length.out = 100)

prediction_data <- expand.grid(
```

```
  R = primary_seq,
  G = mean(ready_dfiiiA$G),
  B = mean(ready_dfiiiA$B),
  Hue = seq(min(ready_dfiiiA$Hue), max(ready_dfiiiA$Hue), length.out = 6),
  Lightness = "saturated",
  Saturation = "subdued"
)

preds <- predict(mod_3A_6, newdata = prediction_data, type = "link", se.fit = TRUE)

link_lwr <- preds$fit - 2 * preds$se.fit
link_upr <- preds$fit + 2 * preds$se.fit

prediction_data$prob <- plogis(preds$fit)
prediction_data$prob_lwr <- plogis(link_lwr)
prediction_data$prob_upr <- plogis(link_upr)

ggplot(prediction_data, aes(x = R, y = prob)) +
  geom_line() +
  geom_ribbon(aes(ymin = prob_lwr, ymax = prob_upr), fill = "blue", alpha = 0.2) +
  facet_wrap(~Hue, scales = "free_x") +
  labs(title = "Model 6: Predictive Probability Trend with Confidence Intervals",
       x = "R", y = "Predicted Probability") +
  theme_minimal()
```

The following code chunk gives predictions for `Model 8` with Confidence Intervals, when Lightness = "dark" and Saturation = "muted".

```
primary_seq <- seq(min(ready_dfiiiA$R), max(ready_dfiiiA$R), length.out = 100)

prediction_data <- expand.grid(
  R = primary_seq,
  G = mean(ready_dfiiiA$G),
  B = mean(ready_dfiiiA$B),
  Hue = seq(min(ready_dfiiiA$Hue), max(ready_dfiiiA$Hue), length.out = 6),
  Lightness = "dark",
  Saturation = "muted"
)

preds <- predict(mod_3A_8, newdata = prediction_data, type = "link", se.fit = TRUE)

link_lwr <- preds$fit - 2 * preds$se.fit
link_upr <- preds$fit + 2 * preds$se.fit

prediction_data$prob <- plogis(preds$fit)
prediction_data$prob_lwr <- plogis(link_lwr)
prediction_data$prob_upr <- plogis(link_upr)

ggplot(prediction_data, aes(x = R, y = prob)) +
  geom_line() +
  geom_ribbon(aes(ymin = prob_lwr, ymax = prob_upr), fill = "blue", alpha = 0.2) +
  facet_wrap(~Hue, scales = "free_x") +
  labs(title = "Model 8: Predictive Probability Trend with Confidence Intervals",
       x = "R", y = "Predicted Probability") +
```

```
  theme_minimal()
```

The following code chunk gives predictions for `Model 6` with Confidence Intervals, when Lightness = "dark" and Saturation = "muted".

```
primary_seq <- seq(min(ready_dfiiiA$R), max(ready_dfiiiA$R), length.out = 100)

prediction_data <- expand.grid(
  R = primary_seq,
  G = mean(ready_dfiiiA$G),
  B = mean(ready_dfiiiA$B),
  Hue = seq(min(ready_dfiiiA$Hue), max(ready_dfiiiA$Hue), length.out = 6),
  Lightness = "dark",
  Saturation = "muted"
)

preds <- predict(mod_3A_6, newdata = prediction_data, type = "link", se.fit = TRUE)

link_lwr <- preds$fit - 2 * preds$se.fit
link_upr <- preds$fit + 2 * preds$se.fit

prediction_data$prob <- plogis(preds$fit)
prediction_data$prob_lwr <- plogis(link_lwr)
prediction_data$prob_upr <- plogis(link_upr)

ggplot(prediction_data, aes(x = R, y = prob)) +
  geom_line() +
  geom_ribbon(aes(ymin = prob_lwr, ymax = prob_upr), fill = "blue", alpha = 0.2) +
  facet_wrap(~Hue, scales = "free_x") +
  labs(title = "Model 6: Predictive Probability Trend with Confidence Intervals",
       x = "R", y = "Predicted Probability") +
  theme_minimal()
```

In the plots above we can observe that the mean predictive trends are consistent (except some fixed values of Hue input) between the 2 selected linear models, `Model 8` and `Model 6`. However, the confidence intervals are wider in the better model, `Model 8`.

### D) Train/tune with resampling

We will train, assess, tune, and compare more complex methods via resampling. We will use `caret` to handle the preprocessing, training, testing, and evaluation.

```
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

We must specify a resampling scheme and a primary performance metric. Let's use 5-fold cross-validation with 3-repeats. Our primary performance metric will be RMSE.

```
my_ctrl <- trainControl(method = "repeatedcv", number = 5, repeats = 3)

my_metric <- "Accuracy"
```

Below we train and tune 12 models:

1) All Categorical and Continuous Variables (Linear Additive):

```
set.seed(2023)

fit_glm_1 <- train(outcome ~ R + G + B + Hue + Lightness + Saturation,
                   data = dfiiiD,
                   method = "glm",
                   metric = my_metric,
                   preProcess = c("center", "scale"),
                   trControl = my_ctrl)

fit_glm_1
```

```
## Generalized Linear Model
##
## 835 samples
##   6 predictor
##   2 classes: 'event', 'non_event'
##
## Pre-processing: centered (16), scaled (16)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 668, 668, 668, 668, 668, 668, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8187678  0.3720999
```

2) Add Categorical Inputs to All Main Effect and All Pairwise Interactions of Continuous Inputs (This is `Model 6` in Part A):

```
set.seed(2023)

fit_glm_2 <- train(outcome ~ (R + G + B + Hue)^2 + Lightness + Saturation,
                   data = dfiiiD,
                   method = "glm",
                   metric = my_metric,
                   preProcess = c("center", "scale"),
                   trControl = my_ctrl)

fit_glm_2
```

```
## Generalized Linear Model
##
## 835 samples
##   6 predictor
##   2 classes: 'event', 'non_event'
##
## Pre-processing: centered (22), scaled (22)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 668, 668, 668, 668, 668, 668, ...
```

```
## Resampling results:
##
##   Accuracy  Kappa
##   0.812359  0.3882556
```

3) Add Categorical Inputs to 3 degree-of-freedom natural (DOF) spline from continuous variables (This is `Model 8` in Part A):

```
set.seed(2023)

fit_glm_3 <- train(outcome ~ splines::ns(R, 3) + splines::ns(G, 3) + splines::ns(B, 3) + splines::ns(Hue
                   data = dfiiiD,
                   method = "glm",
                   metric = my_metric,
                   preProcess = c("center", "scale"),
                   trControl = my_ctrl)

fit_glm_3
```

```
## Generalized Linear Model
##
## 835 samples
##   6 predictor
##   2 classes: 'event', 'non_event'
##
## Pre-processing: centered (24), scaled (24)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 668, 668, 668, 668, 668, 668, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.851113  0.5294859
```

4) Add Categorical Inputs to Interactions from 3 DOF spline from input R and All Pairwise Interactions of Continuous Inputs G, B, Hue (This is `Model 9` in Part A):

```
set.seed(2023)

fit_glm_4 <- train(outcome ~ splines::ns(R, 3) * (G + B + Hue)^2 + Lightness + Saturation,
                   data = dfiiiD,
                   method = "glm",
                   metric = my_metric,
                   preProcess = c("center", "scale"),
                   trControl = my_ctrl)
```

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

```
fit_glm_4
```

```
## Generalized Linear Model
##
## 835 samples
##   6 predictor
##   2 classes: 'event', 'non_event'
##
## Pre-processing: centered (39), scaled (39)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
```

```
## Summary of sample sizes: 668, 668, 668, 668, 668, 668, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8231447  0.4704794
```

5) Elastic Net - Add Categorical Inputs to All Main Effect and All Pairwise Interactions of Continuous Inputs (This is `Model 6` in Part A):

```
set.seed(2023)

fit_enet_1 <- train(outcome ~ (R + G + B + Hue)^2 + Lightness + Saturation,
                    data = dfiiiD,
                    method = "glmnet",
                    metric = my_metric,
                    preProcess = c("center", "scale"),
                    trControl = my_ctrl)

fit_enet_1
```

```
## glmnet
##
## 835 samples
##   6 predictor
##   2 classes: 'event', 'non_event'
##
## Pre-processing: centered (22), scaled (22)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 668, 668, 668, 668, 668, 668, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda        Accuracy   Kappa
##   0.10   0.0003523518  0.8207663  0.3912532
##   0.10   0.0035235183  0.8219710  0.3781104
##   0.10   0.0352351830  0.8231758  0.3742558
##   0.55   0.0003523518  0.8191671  0.3895342
##   0.55   0.0035235183  0.8223726  0.3790735
##   0.55   0.0352351830  0.8227766  0.3713490
##   1.00   0.0003523518  0.8159710  0.3881397
##   1.00   0.0035235183  0.8227742  0.3800590
##   1.00   0.0352351830  0.8227766  0.3713490
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.1 and lambda = 0.03523518.
```

6) Elastic Net - Add Categorical Inputs to 3 degree-of-freedom natural (DOF) spline from continuous variables (This is `Model 8` in Part A):

```
set.seed(2023)

fit_enet_2 <- train(outcome ~ splines::ns(R, 3) + splines::ns(G, 3) + splines::ns(B, 3) + splines::ns(Hu
                    data = dfiiiD,
                    method = "glmnet",
                    metric = my_metric,
                    preProcess = c("center", "scale"),
                    trControl = my_ctrl)
```

```
fit_enet_2
```

```
## glmnet
##
## 835 samples
##   6 predictor
##   2 classes: 'event', 'non_event'
##
## Pre-processing: centered (24), scaled (24)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 668, 668, 668, 668, 668, 668, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda        Accuracy   Kappa
##   0.10   0.0003523518  0.8495114  0.5214013
##   0.10   0.0035235183  0.8435257  0.4849775
##   0.10   0.0352351830  0.8239838  0.3759259
##   0.55   0.0003523518  0.8495114  0.5214013
##   0.55   0.0035235183  0.8395313  0.4687123
##   0.55   0.0352351830  0.8247774  0.3715415
##   1.00   0.0003523518  0.8499106  0.5223381
##   1.00   0.0035235183  0.8343392  0.4463356
##   1.00   0.0352351830  0.8243734  0.3753131
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 1 and lambda = 0.0003523518.
```

7) Elastic Net - Add Categorical Inputs to Interactions from 3 DOF spline from input R and All Pairwise Interactions of Continuous Inputs G, B, Hue (This is `Model 9` in Part A):

```
set.seed(2023)

fit_enet_3 <- train(outcome ~ splines::ns(R, 3) * (G + B + Hue)^2 + Lightness + Saturation,
                    data = dfiiiD,
                    method = "glmnet",
                    metric = my_metric,
                    preProcess = c("center", "scale"),
                    trControl = my_ctrl)

fit_enet_3
```

```
## glmnet
##
## 835 samples
##   6 predictor
##   2 classes: 'event', 'non_event'
##
## Pre-processing: centered (39), scaled (39)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 668, 668, 668, 668, 668, 668, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda        Accuracy   Kappa
##   0.10   0.0003523518  0.8215599  0.4155415
##   0.10   0.0035235183  0.8191646  0.3843785
```

```
##   0.10   0.0352351830  0.8227766  0.3713490
##   0.55   0.0003523518  0.8223559  0.4198842
##   0.55   0.0035235183  0.8191718  0.3789741
##   0.55   0.0352351830  0.8227766  0.3713490
##   1.00   0.0003523518  0.8219543  0.4263235
##   1.00   0.0035235183  0.8187726  0.3737363
##   1.00   0.0352351830  0.8227766  0.3713490
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.1 and lambda = 0.03523518.
```

8) Neural network

```
set.seed(2023)

fit_nnet <- train(outcome ~ .,
                  data = dfiiiD,
                  method = "nnet",
                  metric = my_metric,
                  preProcess = c("center", "scale"),
                  trControl = my_ctrl,
                  trace = FALSE)

fit_nnet
```

```
## Neural Network
##
## 835 samples
##   6 predictor
##   2 classes: 'event', 'non_event'
##
## Pre-processing: centered (16), scaled (16)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 668, 668, 668, 668, 668, 668, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##   1     0e+00  0.7916053  0.2356122
##   1     1e-04  0.8047981  0.2958143
##   1     1e-01  0.8147758  0.3716632
##   3     0e+00  0.8107743  0.4363324
##   3     1e-04  0.7976150  0.4026981
##   3     1e-01  0.8371552  0.5004745
##   5     0e+00  0.7832051  0.3648865
##   5     1e-04  0.7920354  0.3954537
##   5     1e-01  0.8295250  0.4876941
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 3 and decay = 0.1.
```

9) Random forest

```
set.seed(2023)

fit_rf <- train(outcome ~ .,
                data = dfiiiD,
```

```
                method = "rf",
                metric = my_metric,
                trControl = my_ctrl,
                importance = TRUE)

fit_rf
```

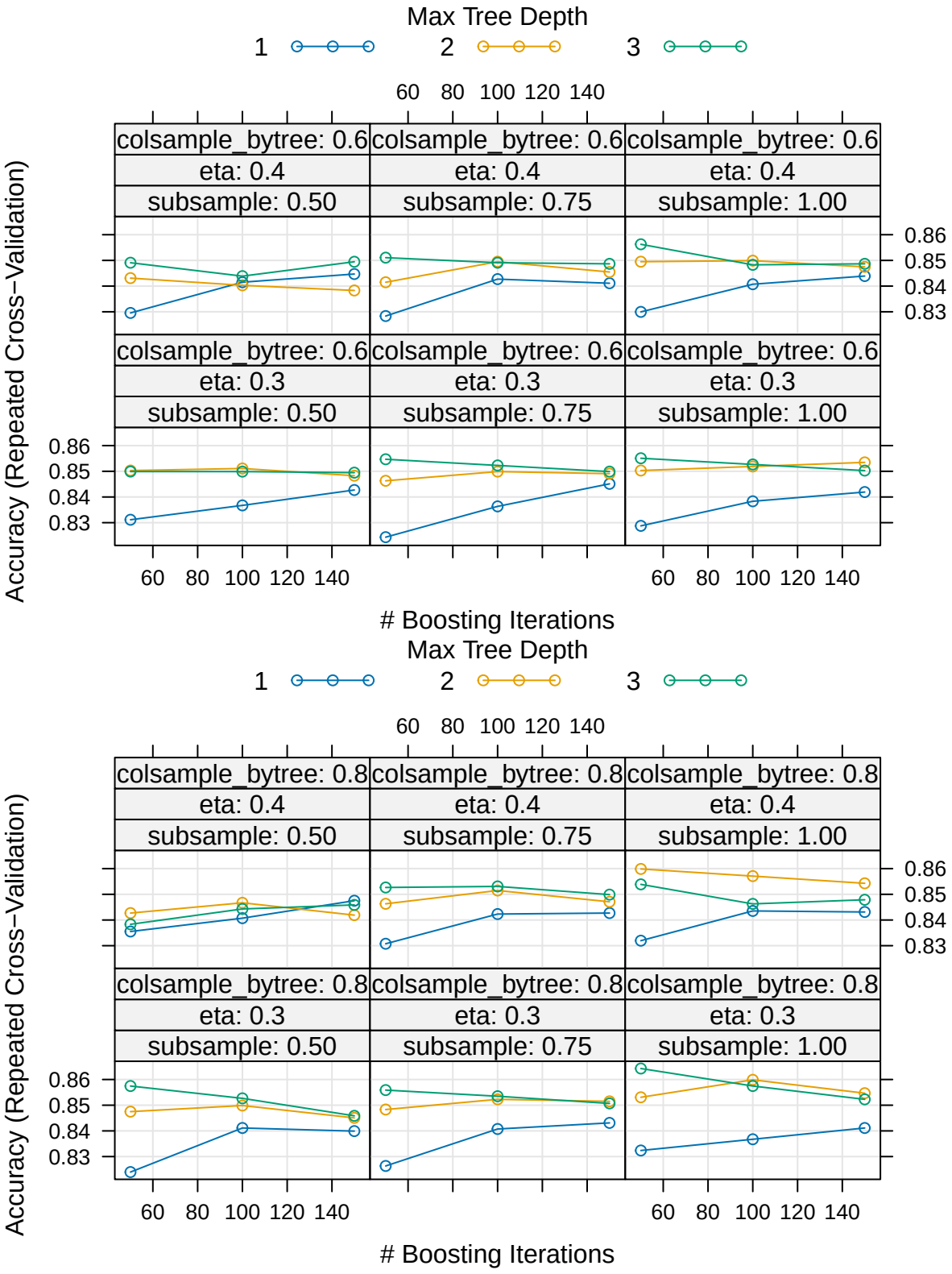
```
## Random Forest
##
## 835 samples
##   6 predictor
##   2 classes: 'event', 'non_event'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 668, 668, 668, 668, 668, 668, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.8243662  0.3679627
##    9    0.8563051  0.5608365
##   16    0.8570962  0.5693412
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 16.
```

10) Gradient boosted tree

```
set.seed(2023)

fit_xgb <- train(outcome ~ .,
                     data = dfiiiD,
                     method = "xgbTree",
                     metric = my_metric,
                     trControl = my_ctrl,
                     verbosity = 0,
                     nthread = 1)

plot(fit_xgb)
```




11) Support Vector Machines (SVM)

```
set.seed(2023)
```

```
fit_svm <- train(outcome ~ .,
                 data = dfiiiD,
                 method = "svmRadial",
                 metric = my_metric,
                 preProcess = c("center", "scale"),
                 trControl = my_ctrl)

fit_svm
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 835 samples
##   6 predictor
##   2 classes: 'event', 'non_event'
##
## Pre-processing: centered (16), scaled (16)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 668, 668, 668, 668, 668, 668, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.25  0.8227766  0.371349
##   0.50  0.8227766  0.371349
##   1.00  0.8227766  0.371349
##
## Tuning parameter 'sigma' was held constant at a value of 0.04018823
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.04018823 and C = 0.25.
```

12) Partial least squares (PLS)

```
pls_grid <- expand.grid(ncomp = 1:5)

set.seed(2023)

fit_pls <- train(outcome ~ .,
                 data = dfiiiD,
                 method = "pls",
                 metric = my_metric,
                 tuneGrid = pls_grid,
                 preProcess = c("center", "scale"),
                 trControl = my_ctrl)

fit_pls
```

```
## Partial Least Squares
##
## 835 samples
##   6 predictor
##   2 classes: 'event', 'non_event'
##
## Pre-processing: centered (16), scaled (16)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 668, 668, 668, 668, 668, 668, ...
## Resampling results across tuning parameters:
```

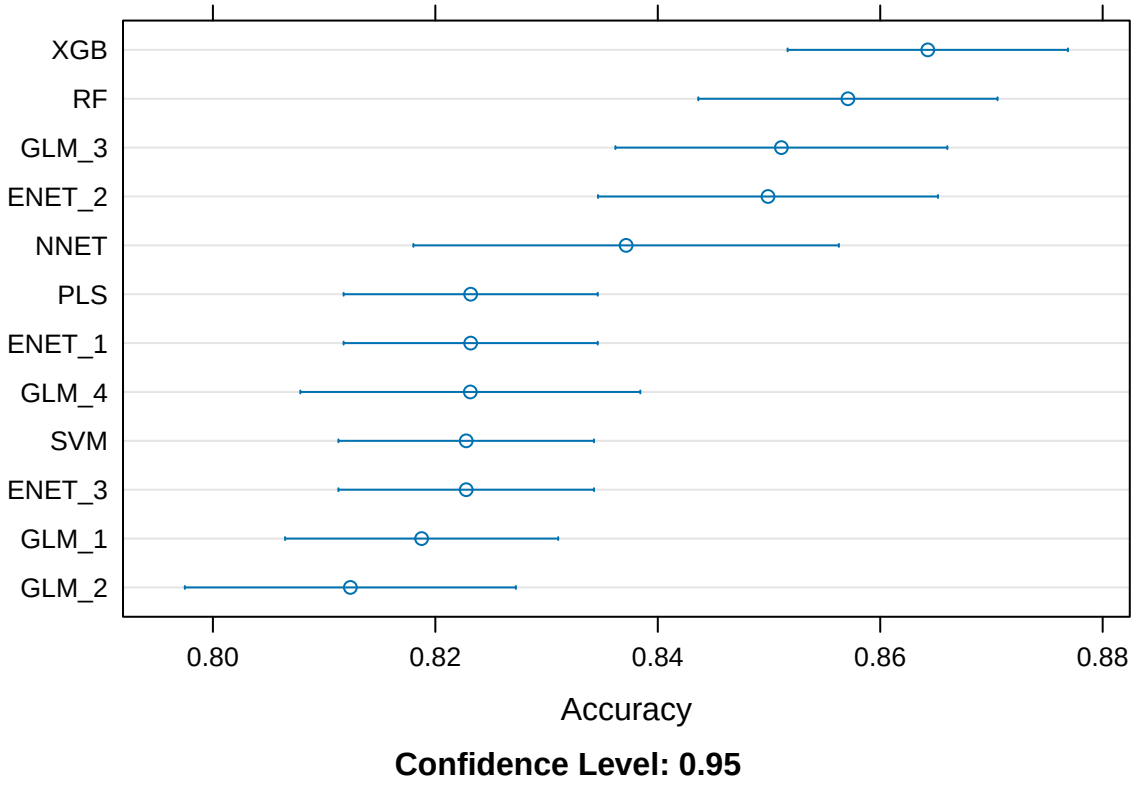
```
##
##   ncomp  Accuracy   Kappa
##   1      0.8223774  0.3724292
##   2      0.8227766  0.3713490
##   3      0.8219782  0.3716152
##   4      0.8211750  0.3697553
##   5      0.8231758  0.3742558
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was ncomp = 5.
```

Let's compare the models. We compile the resampling results together.

```
my_results <- resamples(list(GLM_1 = fit_glm_1,
                             GLM_2 = fit_glm_2,
                             GLM_3 = fit_glm_3,
                             GLM_4 = fit_glm_4,
                             ENET_1 = fit_enet_1,
                             ENET_2 = fit_enet_2,
                             ENET_3 = fit_enet_3,
                             NNET = fit_nnet,
                             RF = fit_rf,
                             XGB = fit_xgb,
                             SVM = fit_svm,
                             PLS = fit_pls))
```

Compare models based on Accuracy.

```
dotplot(my_results, metric = "Accuracy")
```




Based on the results above, the best 3 models are as follows:

1) `fit_xgb` - Gradient boosted tree
2) `fit_rf` - Random forest
3) `fit_glm_3` - Add Categorical Inputs to 3 degree-of-freedom natural (DOF) spline from continuous variables (This is `Model 8` in Part A)

We save the best classification model below:

```
fit_xgb %>% readr::write_rds("best_classification_model.rds")
```