Part 2 - Regression

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 error
```

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.

The code chunk below assembles the data for Part ii) of the project. We use this data set for all regression modeling tasks. The logit-transformed output is named `y`. The `dfii` dataframe as the original `response` and Binary output, `outcome`, removed. This way we can focus on the variables specific to the regression task.

```r
dfii <- df %>%
  mutate(y = boot::logit( (response - 0) / (100 - 0) ) ) %>%
  select(R, G, B,
         Lightness, Saturation, Hue,
         y)
```

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

```r
ready_dfii <- dfii
ready_dfii$R <- scale(ready_dfii$R, center = TRUE, scale = TRUE)
ready_dfii$R <- as.vector(ready_dfii$R)
ready_dfii$G <- scale(ready_dfii$G, center = TRUE, scale = TRUE)
ready_dfii$G <- as.vector(ready_dfii$G)
ready_dfii$B <- scale(ready_dfii$B, center = TRUE, scale = TRUE)
ready_dfii$B <- as.vector(ready_dfii$B)
ready_dfii$Hue <- scale(ready_dfii$Hue, center = TRUE, scale = TRUE)
ready_dfii$Hue <- as.vector(ready_dfii$Hue)
ready_dfii$y <- scale(ready_dfii$y, center = TRUE, scale = TRUE)
ready_dfii$y <- as.vector(ready_dfii$y)
```

We will use the standardized dataset `ready_dfii` in our regression models.

A) Linear Models

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

```r
mod_2A_1 <- lm(y ~ 1, data = ready_dfii)
```

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

```r
mod_2A_2 <- lm(y ~ Lightness + Saturation, data = ready_dfii)
```

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

```r
mod_2A_3 <- lm(y ~ R + G + B + Hue, data = ready_dfii)
```

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

```r
mod_2A_4 <- lm(y ~ R + G + B + Hue + Lightness + Saturation, data = ready_dfii)
```

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

```r
mod_2A_5 <- lm(y ~ (R + G + B + Hue) * (Lightness + Saturation), data = ready_dfii)
```

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

```r
mod_2A_6 <- lm(y ~ (R + G + B + Hue)^2 + Lightness + Saturation, data = ready_dfii)
```

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

```r
mod_2A_7 <- lm(y ~ (R + G + B + Hue)^2 * (Lightness + Saturation), data = ready_dfii)
```

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

```r
mod_2A_8 <- lm(y ~ splines::ns(R, 3) + splines::ns(G, 3) + splines::ns(B, 3) + splines::ns(Hue, 3) + Li
```

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_2A_9 <- lm(y ~ 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:

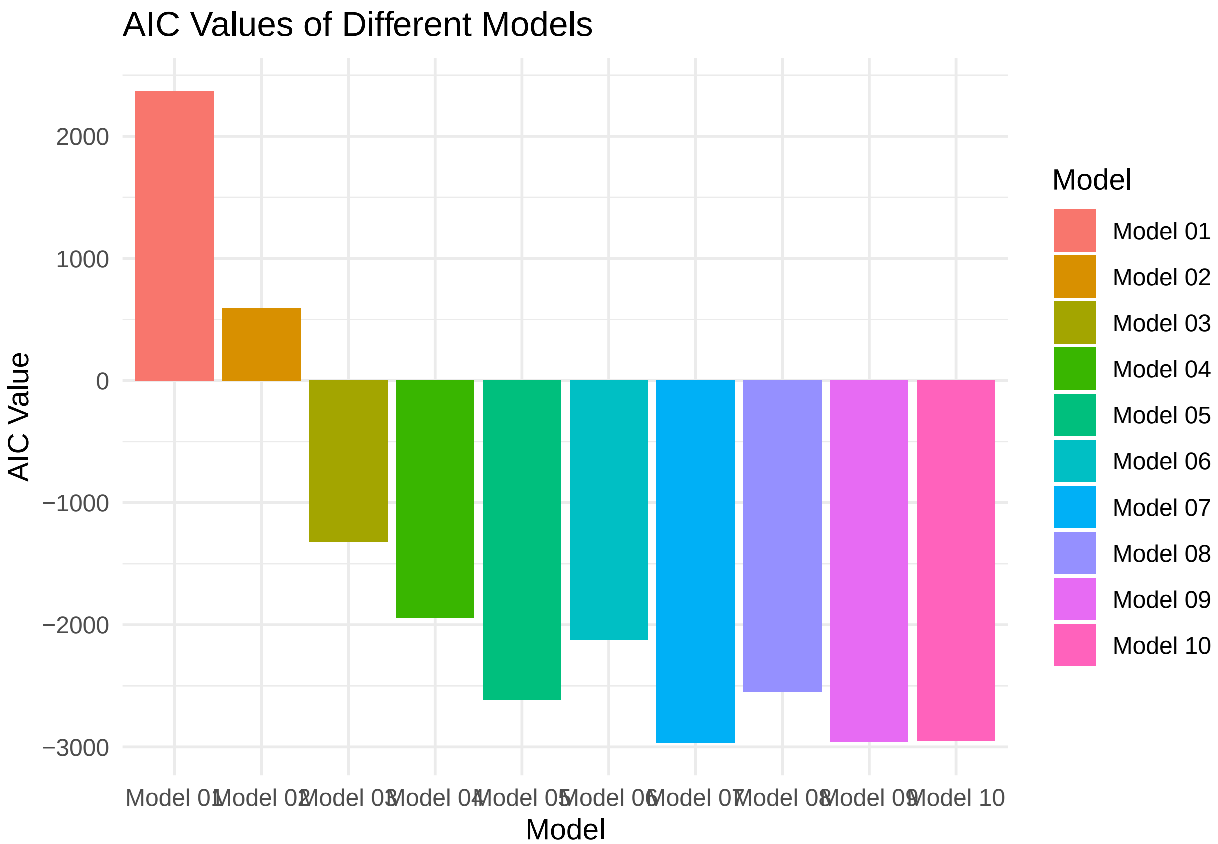
```
mod_2A_10 <- lm(y ~ splines::ns(R, 3) * (G + B + Hue) * (Lightness + Saturation), data=ready_dfii)
```

The following code chunk calculates AIC values of the models.

```
aic_model1 <- AIC(mod_2A_1)
aic_model2 <- AIC(mod_2A_2)
aic_model3 <- AIC(mod_2A_3)
aic_model4 <- AIC(mod_2A_4)
aic_model5 <- AIC(mod_2A_5)
aic_model6 <- AIC(mod_2A_6)
aic_model7 <- AIC(mod_2A_7)
aic_model8 <- AIC(mod_2A_8)
aic_model9 <- AIC(mod_2A_9)
aic_model10 <- AIC(mod_2A_10)

aic_values <- data.frame(
  Model = c("Model 01", "Model 02", "Model 03", "Model 04", "Model 05", "Model 06", "Model 07", "Model 0
  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")
```

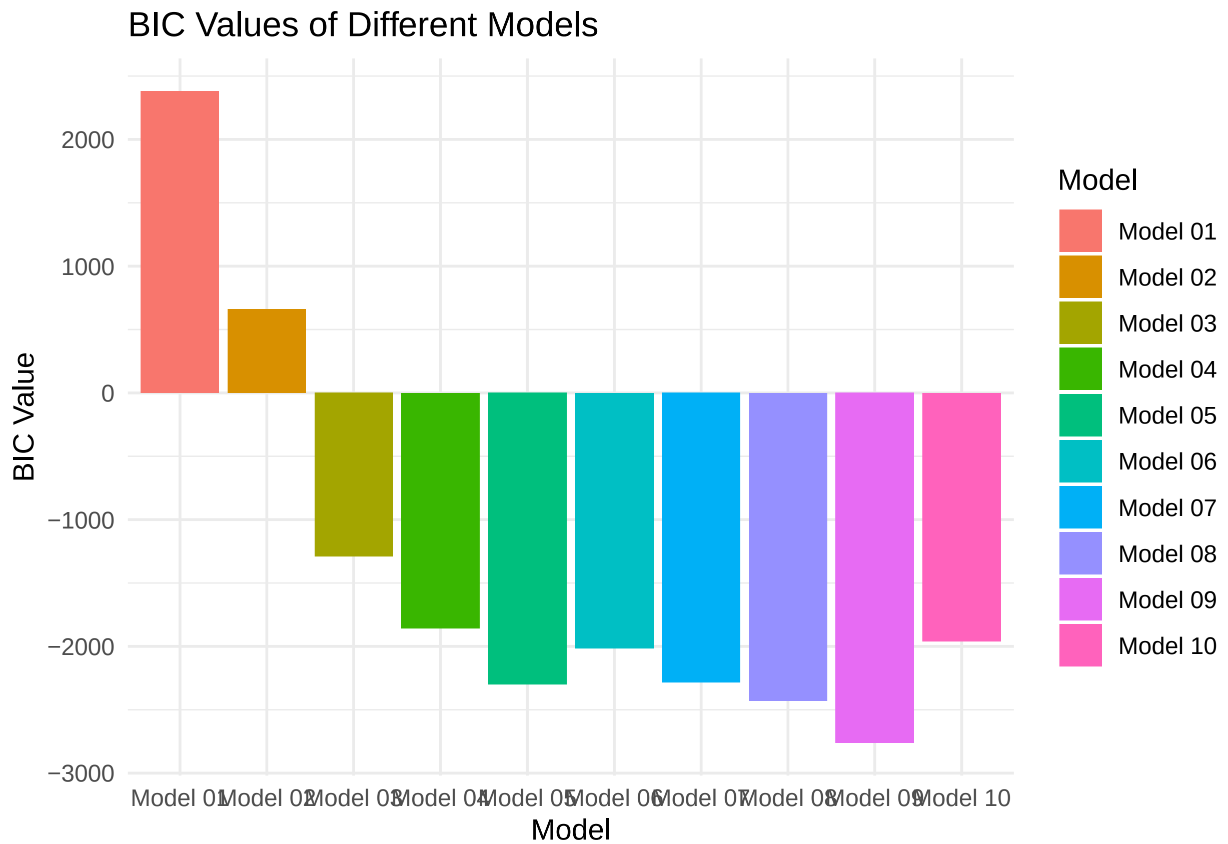


The following code chunk calculates BIC values of the models.

```
bic_model1 <- BIC(mod_2A_1)
bic_model2 <- BIC(mod_2A_2)
bic_model3 <- BIC(mod_2A_3)
bic_model4 <- BIC(mod_2A_4)
bic_model5 <- BIC(mod_2A_5)
bic_model6 <- BIC(mod_2A_6)
bic_model7 <- BIC(mod_2A_7)
bic_model8 <- BIC(mod_2A_8)
bic_model9 <- BIC(mod_2A_9)
bic_model10 <- BIC(mod_2A_10)

bic_values <- data.frame(
  Model = c("Model 01", "Model 02", "Model 03", "Model 04", "Model 05", "Model 06", "Model 07", "Model 0
  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 AIC metric to determine the best model. We will not choose `Model 10` in out list because its BIC metric performance does not look good enough.

Here are top 3 models:
1) `Model 9`
2) `Model 7`
3) `Model 5`

Here is the Coefficient summary visualization for `Model 9`.

```
coefplot::coefplot(mod_2A_9) +
theme_bw()
```

We can list significant inputs for `Model 9` below.

```
tidy_mod9 <- broom::tidy(mod_2A_9)
significant_inputs_mod9 <- tidy_mod9 %>% filter(p.value <0.05)
significant_inputs_mod9
```

Here is the Coefficient summary visualization for `Model 7`.

```
coefplot::coefplot(mod_2A_7) +
theme_bw()
```

We can list significant inputs for `Model 7` below.

```
tidy_mod7 <- broom::tidy(mod_2A_7)
significant_inputs_mod7 <- tidy_mod7 %>% filter(p.value <0.05)
significant_inputs_mod7
```

Here is the Coefficient summary visualization for `Model 5`.

```r
coefplot::coefplot(mod_2A_5) +
theme_bw()
```

We can list significant inputs for `Model 5` below.

```r
tidy_mod5 <- broom::tidy(mod_2A_5)
significant_inputs_mod5 <- tidy_mod5 %>% filter(p.value <0.05)
significant_inputs_mod5
```

B) Bayesian Linear Models

I will fit the best model (Model 9) and second best model (Model 7) we fit in part A). The reason I pick Model 7 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 9` in Bayesian case.

```r
X09 <- model.matrix (y ~ splines::ns(R, 3) * (G + B + Hue)^2 + Lightness + Saturation, data = ready_dfi

info_09_strong <- list(
  yobs = ready_dfii %>% pull(y) %>% as.matrix(),
  design_matrix = X09,
  mu_beta = 0,
  tau_beta = 1,
  sigma_rate = 1
)
```

Here is the design matrix and required information for `Model 7` in Bayesian case.

```r
X07 <- model.matrix (y ~ (R + G + B + Hue)^2 * (Lightness + Saturation), data = ready_dfii)

info_07_strong <- list(
  yobs = ready_dfii %>% pull(y) %>% as.matrix(),
  design_matrix = X07,
  mu_beta = 0,
  tau_beta = 1,
  sigma_rate = 1
)
```

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

```r
lm_logpost <- function(unknowns, my_info)
{
  # specify the number of unknown beta parameters
  length_beta <- length(unknowns)-1

  # extract the beta parameters from the `unknowns` vector
  beta_v <- unknowns[1:length_beta]

  # extract the unbounded noise parameter, varphi
  lik_varphi <- unknowns[length(unknowns)]

  # back-transform from varphi to sigma
  lik_sigma <- exp(lik_varphi)

  # extract design matrix
  X <- my_info$design_matrix
```

```
  # calculate the linear predictor
  mu <- X%*%beta_v

  # evaluate the log-likelihood
  log_lik <- sum(dnorm(x = my_info$yobs,
                       mean = mu,
                       sd = lik_sigma,
                       log = TRUE))

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

  log_prior_sigma <- dexp(x = lik_sigma,
                          rate = my_info$sigma_rate,
                          log = TRUE)

  # add the mean trend prior and noise prior together
  log_prior <- log_prior_beta+log_prior_sigma

  # account for the transformation
  log_derive_adjust <- lik_varphi

  # sum together
  log_lik+log_prior+log_derive_adjust
}
```

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 = 1001))

  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 9 formulation and the Model 7 formulation.

```
  num_beta_params09 <- ncol(X09)

  init_beta09 <- rnorm(num_beta_params09)

  init_varphi09 <- log(rexp(1))

  laplace_09_strong <- my_laplace(c(init_beta09, init_varphi09), lm_logpost, info_09_strong)

  laplace_09_strong$converge
```

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

```
  num_beta_params07 <- ncol(X07)

  init_beta07 <- rnorm(num_beta_params07)

  init_varphi07 <- log(rexp(1))

  laplace_07_strong <- my_laplace(c(init_beta07, init_varphi07), lm_logpost, info_07_strong)

  laplace_07_strong$converge
```

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

We use the Bayes Factor to compare the models. We can conclude that `Model 9` is the better of the models.

```
exp(laplace_09_strong$log_evidence)/exp(laplace_07_strong$log_evidence)
```

```
## [1] Inf
```

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 `Model 9`.

```r
post_means_09_strong <- laplace_09_strong$mode
post_sds_09_strong <- sqrt(diag(laplace_09_strong$var_matrix))
feature_names_09 <- colnames(X09)

viz_post_coefs(post_means_09_strong[-length(post_means_09_strong)], post_sds_09_strong[-length(post_sds_
```

Now for our best model `Model 9` we study the posterior UNCERTAINTY on the likelihood noise.

```r
info_09_weak <- list(
  yobs = ready_dfii %>% pull(y) %>% as.matrix(),
  design_matrix = X09,
  mu_beta = 0,
  tau_beta = 50,
  sigma_rate = 1
)

laplace_09_weak <- my_laplace(c(init_beta09, init_varphi09), lm_logpost, info_09_weak)
laplace_09_weak$converge
```

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

```r
info_09_very_strong <- list(
  yobs = ready_dfii %>% pull(y) %>% as.matrix(),
  design_matrix = X09,
  mu_beta = 0,
  tau_beta = 1/50,
  sigma_rate = 1
)
laplace_09_very_strong <- my_laplace(c(init_beta09, init_varphi09), lm_logpost, info_09_very_strong)
laplace_09_very_strong$converge
```

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

We create the posterior summary visualization figure for `Model 9` when `tau_beta = 50`.

```r
post_means_09_weak <- laplace_09_weak$mode
post_sds_09_weak <- sqrt(diag(laplace_09_weak$var_matrix))
feature_names_09 <- colnames(X09)

viz_post_coefs(post_means_09_weak[-length(post_means_09_weak)], post_sds_09_weak[-length(post_sds_09_wea
```

We create the posterior summary visualization figure for `Model 9` when `tau_beta = 1/50`.

```r
post_means_09_very_strong <- laplace_09_very_strong$mode
post_sds_09_very_strong <- sqrt(diag(laplace_09_very_strong$var_matrix))
feature_names_09 <- colnames(X09)

viz_post_coefs(post_means_09_very_strong[-length(post_means_09_very_strong)], post_sds_09_very_strong[-l
```

When we compare the coefficient plots above, particularly the ones for `tau_beta = 1` and `tau_beta = 50`, we observe that the coefficients are similar. This suggests that the posterior is precise.

C) Linear models Predictions

We will make predictions with our 2 selected linear models, `Model 9` and `Model 7` in order to visualize the trends of the LOGIT transformed response with respect to the inputs.

As a reminder,

`Model 9` is: `mod_2A_9 <- lm(y ~ splines::ns(R, 3) * (G + B + Hue)^2 + Lightness + Saturation, data=ready_dfii)`

and

`Model 7` is: `mod_2A_7 <- lm(y ~ (R + G + B + Hue)^2 * (Lightness + Saturation), data = ready_dfii)`

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

The following code chunk gives predictions for `Model 9` with Confidence and Prediction Intervals, when Lightness = "saturated" and Saturation = "subdued".

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

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

preds <- predict(mod_2A_9, newdata = prediction_data, interval = "confidence")
preds_pred_interval <- predict(mod_2A_9, newdata = prediction_data, interval = "prediction")

prediction_data$fit <- preds[, "fit"]
prediction_data$conf_low <- preds[, "lwr"]
prediction_data$conf_high <- preds[, "upr"]
prediction_data$pred_low <- preds_pred_interval[, "lwr"]
prediction_data$pred_high <- preds_pred_interval[, "upr"]

ggplot(prediction_data, aes(x = R, y = fit)) +
  geom_line() +
  geom_ribbon(aes(ymin = conf_low, ymax = conf_high), fill = "blue", alpha = 0.4) +
  geom_ribbon(aes(ymin = pred_low, ymax = pred_high), fill = "red", alpha = 0.2) +
  facet_wrap(~Hue, scales = "free_x") +
  labs(title = "Model 9: Predictive Mean Trend with Confidence and Prediction Intervals",
       x = "R", y = "Predicted Logit(Response)") +
  theme_minimal()
```

The following code chunk gives predictions for `Model 7` with Confidence and Prediction Intervals, when Lightness = "saturated" and Saturation = "subdued".

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

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

```
preds <- predict(mod_2A_7, newdata = prediction_data, interval = "confidence")
preds_pred_interval <- predict(mod_2A_7, newdata = prediction_data, interval = "prediction")

prediction_data$fit <- preds[, "fit"]
prediction_data$conf_low <- preds[, "lwr"]
prediction_data$conf_high <- preds[, "upr"]
prediction_data$pred_low <- preds_pred_interval[, "lwr"]
prediction_data$pred_high <- preds_pred_interval[, "upr"]

ggplot(prediction_data, aes(x = R, y = fit)) +
  geom_line() +
  geom_ribbon(aes(ymin = conf_low, ymax = conf_high), fill = "blue", alpha = 0.4) +
  geom_ribbon(aes(ymin = pred_low, ymax = pred_high), fill = "red", alpha = 0.2) +
  facet_wrap(~Hue, scales = "free_x") +
  labs(title = "Model 7: Predictive Mean Trend with Confidence and Prediction Intervals",
       x = "R", y = "Predicted Logit(Response)") +
  theme_minimal()
```

Let's change cathegorical inputs and visualize the predictions.

The following code chunk gives predictions for `Model 9` with Confidence and Prediction Intervals, when Lightness = "dark" and Saturation = "muted".

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

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

preds <- predict(mod_2A_9, newdata = prediction_data, interval = "confidence")
preds_pred_interval <- predict(mod_2A_9, newdata = prediction_data, interval = "prediction")

prediction_data$fit <- preds[, "fit"]
prediction_data$conf_low <- preds[, "lwr"]
prediction_data$conf_high <- preds[, "upr"]
prediction_data$pred_low <- preds_pred_interval[, "lwr"]
prediction_data$pred_high <- preds_pred_interval[, "upr"]

ggplot(prediction_data, aes(x = R, y = fit)) +
  geom_line() +
  geom_ribbon(aes(ymin = conf_low, ymax = conf_high), fill = "blue", alpha = 0.4) +
  geom_ribbon(aes(ymin = pred_low, ymax = pred_high), fill = "red", alpha = 0.2) +
  facet_wrap(~Hue, scales = "free_x") +
  labs(title = "Model 9: Predictive Mean Trend with Confidence and Prediction Intervals",
       x = "R", y = "Predicted Logit(Response)") +
  theme_minimal()
```

The following code chunk gives predictions for `Model 7` with Confidence and Prediction Intervals, when Lightness = "dark" and Saturation = "muted".

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

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

preds <- predict(mod_2A_7, newdata = prediction_data, interval = "confidence")
preds_pred_interval <- predict(mod_2A_7, newdata = prediction_data, interval = "prediction")

prediction_data$fit <- preds[, "fit"]
prediction_data$conf_low <- preds[, "lwr"]
prediction_data$conf_high <- preds[, "upr"]
prediction_data$pred_low <- preds_pred_interval[, "lwr"]
prediction_data$pred_high <- preds_pred_interval[, "upr"]

ggplot(prediction_data, aes(x = R, y = fit)) +
  geom_line() +
  geom_ribbon(aes(ymin = conf_low, ymax = conf_high), fill = "blue", alpha = 0.4) +
  geom_ribbon(aes(ymin = pred_low, ymax = pred_high), fill = "red", alpha = 0.2) +
  facet_wrap(~Hue, scales = "free_x") +
  labs(title = "Model 7: Predictive Mean Trend with Confidence and Prediction Intervals",
       x = "R", y = "Predicted Logit(Response)") +
  theme_minimal()
```

In the plots anove we can observe that the predictive trends are consistent between the 2 selected linear models, `Model 9` and `Model 7`. Also the confidence and prediction intervals are narrower in the better model, `Model 9`, as expected.

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.

```r
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.

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

my_metric <- "RMSE"
```

Below we train and tune 10 models:

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

```
set.seed(2023)

fit_lm_1 <- train(y ~ R + G + B + Hue + Lightness + Saturation,
                  data = dfii,
                  method = "lm",
                  metric = my_metric,
                  preProcess = c("center", "scale"),
                  trControl = my_ctrl)

fit_lm_1
```

```
## Linear Regression
##
## 835 samples
##   6 predictor
##
## Pre-processing: centered (16), scaled (16)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 668, 668, 667, 668, 669, 668, ...
## Resampling results:
##
##   RMSE        Rsquared   MAE
##   0.08975913  0.9942679  0.06750588
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

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

```
set.seed(2023)

fit_lm_2 <- train(y ~ (R + G + B + Hue)^2 + Lightness + Saturation,
                  data = dfii,
                  method = "lm",
                  metric = my_metric,
                  preProcess = c("center", "scale"),
                  trControl = my_ctrl)

fit_lm_2
```

```
## Linear Regression
##
## 835 samples
##   6 predictor
##
## Pre-processing: centered (22), scaled (22)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 668, 668, 667, 668, 669, 668, ...
## Resampling results:
##
##   RMSE        Rsquared   MAE
##   0.08103047  0.9953265  0.0611509
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

3) Add Categorical Inputs to Interactions from 7 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_lm_3 <- train(y ~ splines::ns(R, 3) * (G + B + Hue)^2 + Lightness + Saturation,
                  data = dfii,
                  method = "lm",
                  metric = my_metric,
                  preProcess = c("center", "scale"),
                  trControl = my_ctrl)

fit_lm_3
```

```
## Linear Regression
##
## 835 samples
##   6 predictor
##
## Pre-processing: centered (39), scaled (39)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 668, 668, 667, 668, 669, 668, ...
## Resampling results:
##
##   RMSE        Rsquared   MAE
##   0.05075805  0.9981852  0.03671361
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

4) Interaction of Categorical Inputs with All Main Effects and All Pairwise Interactions of Continuous Inputs (This is `Model 7` in Part A):

```
set.seed(2023)

fit_lm_4 <- train(y ~ (R + G + B + Hue)^2 * (Lightness + Saturation),
                  data = dfii,
                  method = "lm",
                  metric = my_metric,
                  preProcess = c("center", "scale"),
                  trControl = my_ctrl)

fit_lm_4
```

```
## Linear Regression
##
## 835 samples
##   6 predictor
##
## Pre-processing: centered (142), scaled (142)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 668, 668, 667, 668, 669, 668, ...
## Resampling results:
##
##   RMSE        Rsquared   MAE
##   0.05666299  0.9977373  0.03915899
##
```

```
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

5) Elastic Net - Add Categorical Inputs to All Main Effect and All Pairwise Interactions of Continuous Inputs:

```r
set.seed(2023)

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

fit_enet_1
```

```
## glmnet
##
## 835 samples
##   6 predictor
##
## Pre-processing: centered (22), scaled (22)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 668, 668, 667, 668, 669, 668, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda       RMSE        Rsquared   MAE
##   0.10   0.002324946  0.08165127  0.9952667  0.06167801
##   0.10   0.023249462  0.08759725  0.9946190  0.06466421
##   0.10   0.232494618  0.14977489  0.9875129  0.11020423
##   0.55   0.002324946  0.08377511  0.9950160  0.06261841
##   0.55   0.023249462  0.09619464  0.9936402  0.06999642
##   0.55   0.232494618  0.20102591  0.9893612  0.15009780
##   1.00   0.002324946  0.08441828  0.9949410  0.06281064
##   1.00   0.023249462  0.10183414  0.9930497  0.07346151
##   1.00   0.232494618  0.26708859  0.9893038  0.20912853
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 0.1 and lambda = 0.002324946.
```

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

```r
set.seed(2023)

fit_enet_2 <- train(y ~ splines::ns(R, 3) * (G + B + Hue)^2 + Lightness + Saturation,
                    data = dfii,
                    method = "glmnet",
                    metric = my_metric,
                    preProcess = c("center", "scale"),
                    trControl = my_ctrl)

fit_enet_2
```

```
## glmnet
##
```

```
## 835 samples
##   6 predictor
##
## Pre-processing: centered (39), scaled (39)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 668, 668, 667, 668, 669, 668, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda       RMSE        Rsquared   MAE
##   0.10   0.002324946  0.06828869  0.9966806  0.05143427
##   0.10   0.023249462  0.08368706  0.9950879  0.06109753
##   0.10   0.232494618  0.13649601  0.9896168  0.09549405
##   0.55   0.002324946  0.07300753  0.9962175  0.05416558
##   0.55   0.023249462  0.08703765  0.9948753  0.06446190
##   0.55   0.232494618  0.22268986  0.9862944  0.16729674
##   1.00   0.002324946  0.07503549  0.9960085  0.05528771
##   1.00   0.023249462  0.09940154  0.9934686  0.07173152
##   1.00   0.232494618  0.30486550  0.9736650  0.24891846
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 0.1 and lambda = 0.002324946.
```

7) Elastic Net - Interaction of Categorical Inputs with All Main Effects and All Pairwise Interactions of Continuous Inputs (This is `Model 7` in Part A):

```r
set.seed(2023)

fit_enet_3 <- train(y ~ (R + G + B + Hue)^2 * (Lightness + Saturation),
                    data = dfii,
                    method = "glmnet",
                    metric = my_metric,
                    preProcess = c("center", "scale"),
                    trControl = my_ctrl)

fit_enet_3
```

```
## glmnet
##
## 835 samples
##   6 predictor
##
## Pre-processing: centered (142), scaled (142)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 668, 668, 667, 668, 669, 668, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda       RMSE        Rsquared   MAE
##   0.10   0.002324946  0.06953759  0.9965491  0.05204470
##   0.10   0.023249462  0.08396740  0.9950565  0.06222224
##   0.10   0.232494618  0.14900272  0.9876100  0.10861246
##   0.55   0.002324946  0.07808563  0.9956672  0.05835783
##   0.55   0.023249462  0.09480402  0.9938288  0.06890852
##   0.55   0.232494618  0.20102591  0.9893612  0.15009780
##   1.00   0.002324946  0.08119206  0.9953157  0.06031654
##   1.00   0.023249462  0.10070605  0.9932142  0.07269976
```

```
##   1.00   0.232494618  0.26708859  0.9893038  0.20912853
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 0.1 and lambda = 0.002324946.
```

8) Neural network

```r
set.seed(2023)

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

fit_nnet
```

```
## Neural Network
##
## 835 samples
##   6 predictor
##
## Pre-processing: centered (16), scaled (16)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 668, 668, 667, 668, 669, 668, ...
## Resampling results across tuning parameters:
##
##   size  decay  RMSE        Rsquared   MAE
##   1     0e+00  0.19320972  0.9597068  0.13355535
##   1     1e-04  0.16984072  0.9770668  0.12304679
##   1     1e-01  0.11490967  0.9907348  0.08739311
##   3     0e+00  0.08448377  0.9938737  0.05679022
##   3     1e-04  0.07850228  0.9950783  0.05481319
##   3     1e-01  0.07132761  0.9963056  0.05314323
##   5     0e+00  0.05574282  0.9977915  0.04112612
##   5     1e-04  0.05714402  0.9976555  0.04174699
##   5     1e-01  0.06575734  0.9969208  0.04839112
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were size = 5 and decay = 0.
```

9) Random forest

```r
set.seed(2023)

fit_rf <- train(y ~ .,
                data = dfii,
                method = "rf",
                metric = my_metric,
                trControl = my_ctrl,
                importance = TRUE)

fit_rf
```

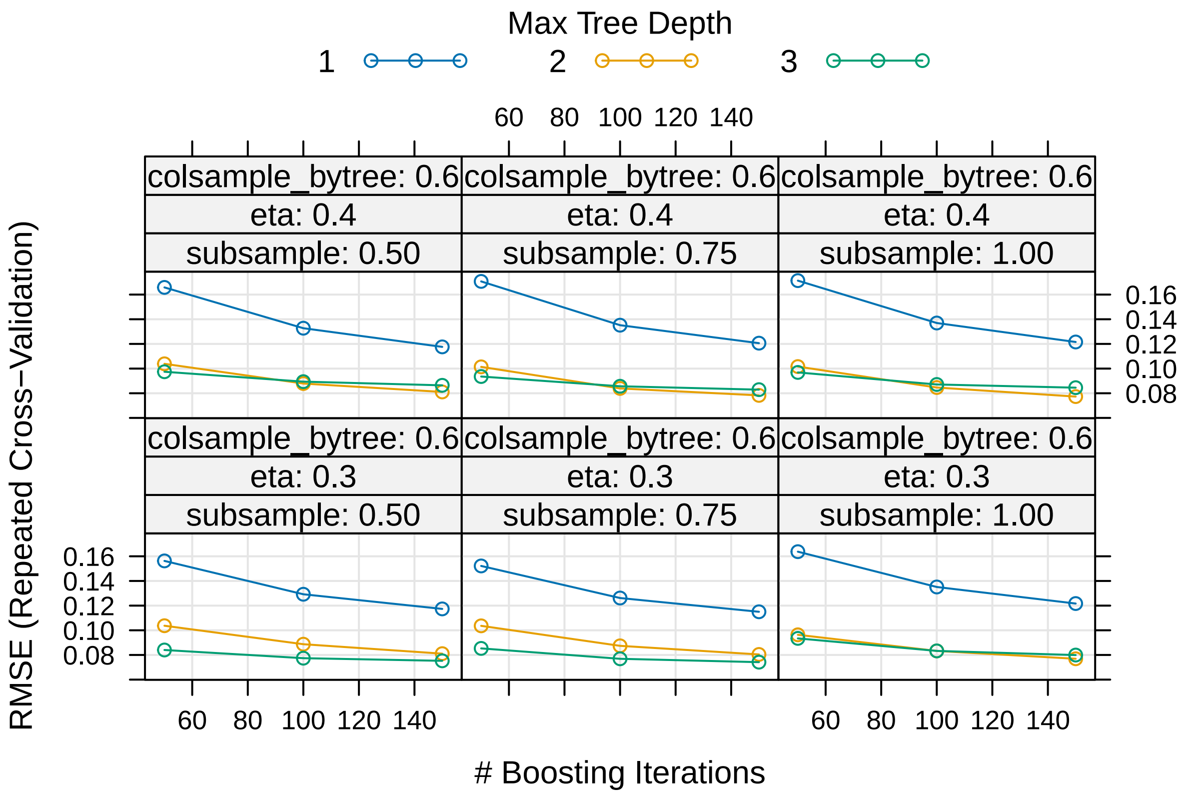
```
## Random Forest
##
## 835 samples
##   6 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 668, 668, 667, 668, 669, 668, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE        Rsquared   MAE
##    2    0.22462284  0.9765446  0.16499607
##    9    0.07004397  0.9966316  0.05158597
##   16    0.08726034  0.9945832  0.06318438
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 9.
```

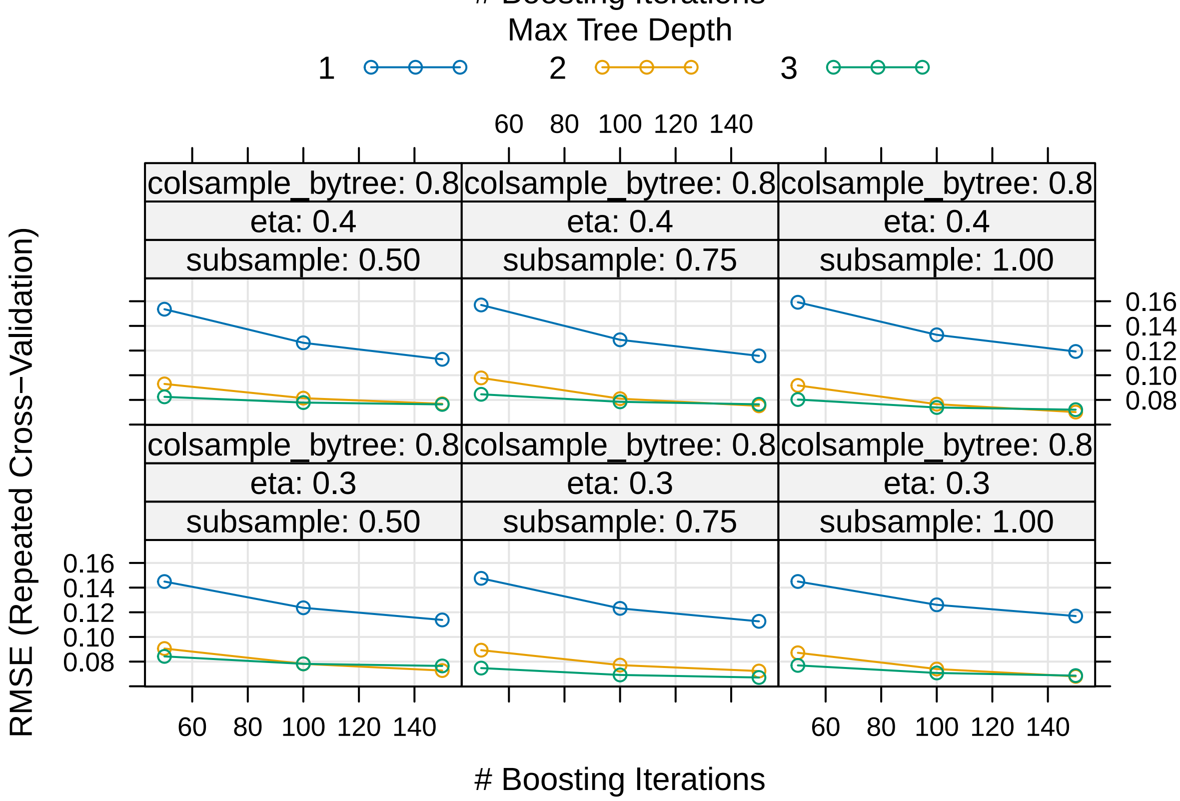
10) Gradient boosted tree

```r
set.seed(2023)

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

plot(fit_xgb)
```





11) Support Vector Machines (SVM)

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

```r
fit_svm <- train(y ~ .,
                 data = dfii,
                 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
##
## Pre-processing: centered (16), scaled (16)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 668, 668, 667, 668, 669, 668, ...
## Resampling results across tuning parameters:
##
##   C     RMSE       Rsquared   MAE
##   0.25  0.1391192  0.9870649  0.10440756
##   0.50  0.1193192  0.9901315  0.09022252
##   1.00  0.1043656  0.9924022  0.07961644
##
## Tuning parameter 'sigma' was held constant at a value of 0.04018823
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.04018823 and C = 1.
```

12) Partial least squares (PLS)

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

set.seed(2023)

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

fit_pls
```

```
## Partial Least Squares
##
## 835 samples
##   6 predictor
##
## Pre-processing: centered (16), scaled (16)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 668, 668, 667, 668, 669, 668, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE       Rsquared   MAE
```

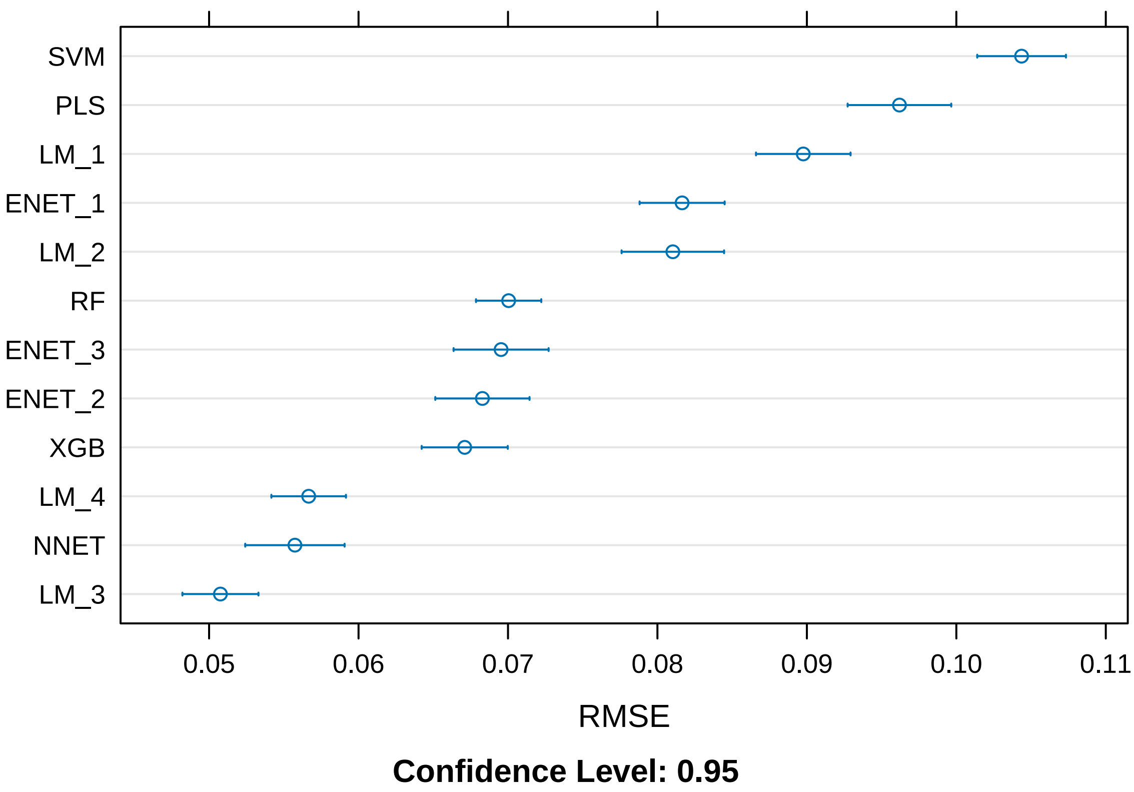
```
##    1       0.2610250  0.9518415  0.19936709
##    2       0.1735374  0.9786849  0.13336085
##    3       0.1379873  0.9864512  0.10376504
##    4       0.1040805  0.9923136  0.07882965
##    5       0.0961935  0.9934236  0.07122056
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was ncomp = 5.
```

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

```r
my_results <- resamples(list(LM_1 = fit_lm_1,
                             LM_2 = fit_lm_2,
                             LM_3 = fit_lm_3,
                             LM_4 = fit_lm_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))
```

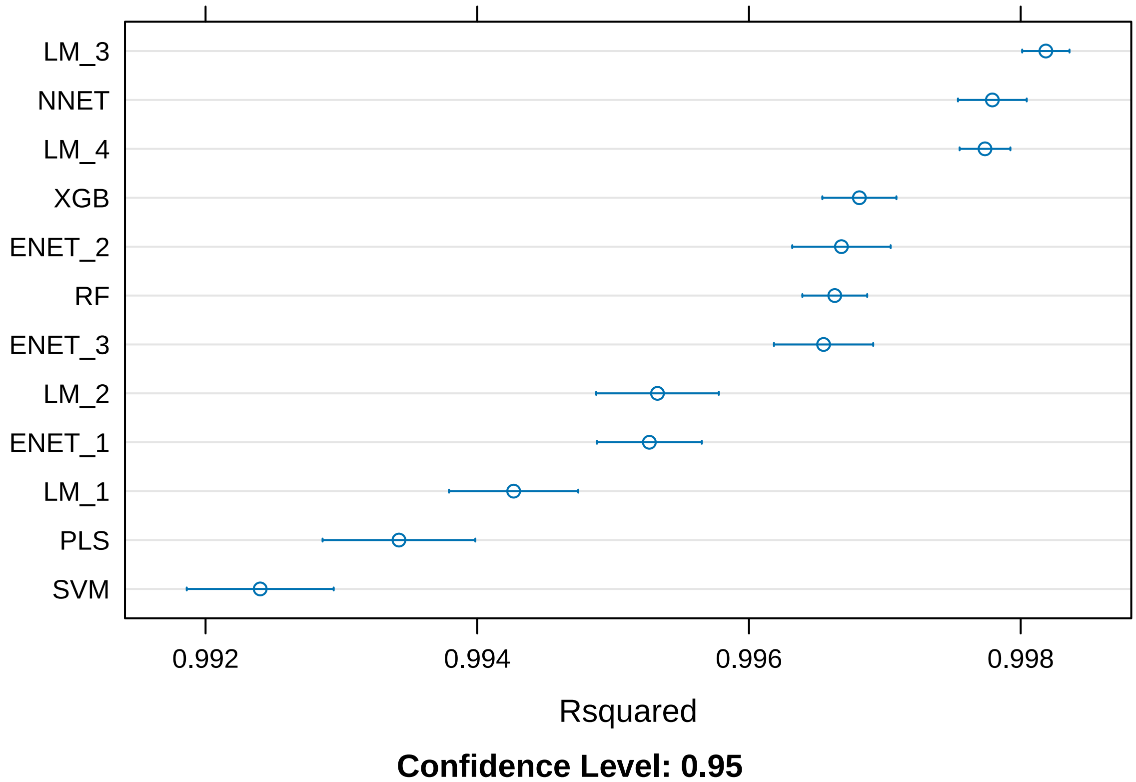
Compare models based on RMSE.

```r
dotplot(my_results, metric = "RMSE")
```



Compare the models based on R-squared.

```r
dotplot(my_results, metric = "Rsquared")
```



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

1) `fit_lm_3` - 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).
2) `fit_nnet` - Neural network
3) `fit_lm_4` - Interaction of Categorical Inputs with All Main Effects and All Pairwise Interactions of Continuous Inputs (This is Model 7 in Part A).

We save the best regression model below:

```r
fit_lm_3 %>% readr::write_rds("best_regression_model.rds")
```