P8106 Midterm

Brian Jo Hsuan Lee

3/15/2022

Load packages

```
library(tidyverse)
library(corrplot)
library(caret)
library(splines)
```

Clean data. Consolidate the historic team names into their corresponding current names; replace NA values in weather temperature, humidity and wind with each of their means; expand the weather detail column into boolean `dome`, `rain`, `fog`, and `snow` columns; update the spread values to those for the home team; rid seasons before 1979 due to incomplete records; rid `schedule playoff` due to collinearity with the more informative `schedule week`; rid `stadium` due to speculated insignificant contribution; rid `spread_favored` and `weather details` as they are replaced by new predictors

```
data = read_csv("spreadspoke_score.csv",
                    col_types = "iffffiiffddffiiic",
                    col_select = c("schedule_season":"weather_detail")) %>%
  filter(
    schedule_season %in% (1979:2021)
  ) %>%
  mutate(
    schedule_season = droplevels(schedule_season),
    weather_detail = replace(weather_detail, is.na(weather_detail), "Dry"),
    weather_detail = factor(weather_detail),
    weather_temperature = replace(weather_temperature, is.na(weather_temperature), round(mean(weather_t
    weather_wind_mph = replace(weather_wind_mph, is.na(weather_wind_mph), round(mean(weather_wind_mph,
    weather_humidity = replace(weather_humidity, is.na(weather_humidity),round(mean(weather_humidity, na
    schedule_week = fct_collapse(schedule_week,
                                 "SuperBowl" = c("Superbowl","SuperBowl"),
                                 "WildCard" = c("Wildcard","WildCard")),
    schedule_week = fct_relevel(schedule_week, c(1:18, "WildCard", "Division", "Conference", "SuperBowl
    team_home = fct_collapse(team_home,
                             "Tennessee Titans" = c("Tennessee Titans", "Tennessee Oilers", "Houston Oi
                             "Washington Football Team" = c("Washington Football Team", "Washington Red
                             "Las Vegas Raiders" = c("Oakland Raiders", "Los Angeles Raiders", "Las Vega
                             "Indianapolis Colts" = c("Baltimore Colts", "Indianapolis Colts"),
                             "Los Angeles Chargers" = c("Los Angeles Chargers", "San Diego Chargers"),
                             "Arizona Cardinals" = c("St. Louis Cardinals", "Phoenix Cardinals", "Arizon
                             "Los Angeles Rams" = c("Los Angeles Rams", "St. Louis Rams"),
                             "New England Patriots" = c("New England Patriots", "Boston Patriots")),
    team_away = fct_collapse(team_away,
                             "Tennessee Titans" = c("Tennessee Titans", "Tennessee Oilers", "Houston Oi
                             "Washington Football Team" = c("Washington Football Team", "Washington Red
```

```
                              "Las Vegas Raiders" = c("Oakland Raiders", "Los Angeles Raiders", "Las Vega
                              "Indianapolis Colts" = c("Baltimore Colts", "Indianapolis Colts"),
                              "Los Angeles Chargers" = c("Los Angeles Chargers", "San Diego Chargers"),
                              "Arizona Cardinals" = c("St. Louis Cardinals", "Phoenix Cardinals", "Arizon
                              "Los Angeles Rams" = c("Los Angeles Rams", "St. Louis Rams"),
                              "New England Patriots" = c("New England Patriots", "Boston Patriots")),
    team_away = fct_relevel(team_away, levels(team_home)),
    team_favorite_id = recode_factor(team_favorite_id,
                                     "MIA" = "Miami Dolphins",
                                     "TEN" = "Tennessee Titans",
                                     "LAC" = "Los Angeles Chargers",
                                     "GB" = "Green Bay Packers",
                                     "ATL" = "Atlanta Falcons",
                                     "BUF" = "Buffalo Bills",
                                     "DET" = "Detroit Lions",
                                     "PIT" = "Pittsburgh Steelers",
                                     "SF" = "San Francisco 49ers",
                                     "ARI" = "Arizona Cardinals",
                                     "WAS" = "Washington Football Team",
                                     "LAR" = "Los Angeles Rams",
                                     "CLE" = "Cleveland Browns",
                                     "DAL" = "Dallas Cowboys",
                                     "DEN" = "Denver Broncos",
                                     "MIN" = "Minnesota Vikings",
                                     "NYJ" = "New York Jets",
                                     "LVR" = "Las Vegas Raiders",
                                     "PHI" = "Philadelphia Eagles",
                                     "IND" = "Indianapolis Colts",
                                     "NE" = "New England Patriots",
                                     "KC" = "Kansas City Chiefs",
                                     "NYG" = "New York Giants",
                                     "CHI" = "Chicago Bears",
                                     "NO"= "New Orleans Saints",
                                     "CIN" = "Cincinnati Bengals",
                                     "SEA" = "Seattle Seahawks",
                                     "TB" = "Tampa Bay Buccaneers",
                                     "JAX" = "Jacksonville Jaguars",
                                     "CAR" = "Carolina Panthers",
                                     "BAL" = "Baltimore Ravens",
                                     "HOU" = "Houston Texans",
                                     .default = "None"),
    spread_home = ifelse(as.character(team_away) == as.character(team_favorite_id), abs(spread_favorite
    dome = ifelse(((as.character(weather_detail) == "DOME") | (as.character(weather_detail) == "DOME (Op
    fog = ifelse((as.character(weather_detail) == "Fog") | (as.character(weather_detail) == "Rain | Fog
    rain = ifelse((as.character(weather_detail) == "Rain") | (as.character(weather_detail) == "Rain | Fo
    snow = ifelse((as.character(weather_detail) == "Snow") | (as.character(weather_detail) == "Snow | Fo
  ) %>%
  select(-schedule_playoff, -team_favorite_id, -spread_favorite, -stadium, -weather_detail) %>%
  drop_na()
```

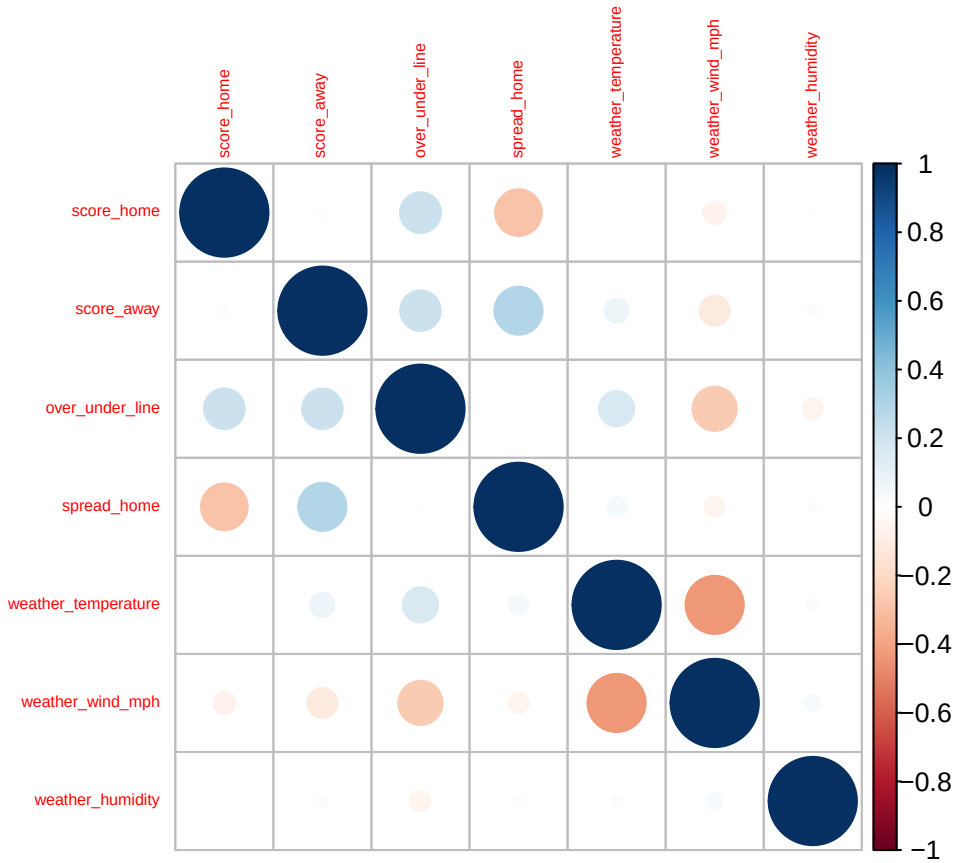
Partition data into training and testing sets, and define resampling method.

```
set.seed(2022)
```

```
# partition data into training and testing sets into randomized 4:1 splits
train_index = createDataPartition(y = data$score_home, p = 0.8, list = FALSE)
train_data = data[train_index, ]
train_cont_data =
  train_data %>%
  select(score_home, score_away, over_under_line, spread_home, weather_temperature, weather_wind_mph, w
test_data = data[-train_index, ]

# matrices of predictors
train_pred = model.matrix(score_home ~ ., train_data)[ ,-1]
train_cont_pred = model.matrix(score_home ~ ., train_cont_data)[ ,-1]
test_pred = model.matrix(score_home ~ ., test_data)[ ,-1]

# vectors of response
train_resp = train_data$score_home
test_resp = test_data$score_home

# use 5 repeats of 10-Fold CV as the resampling method
ctrl = trainControl(method = "repeatedcv", repeats = 5, number = 10)
```

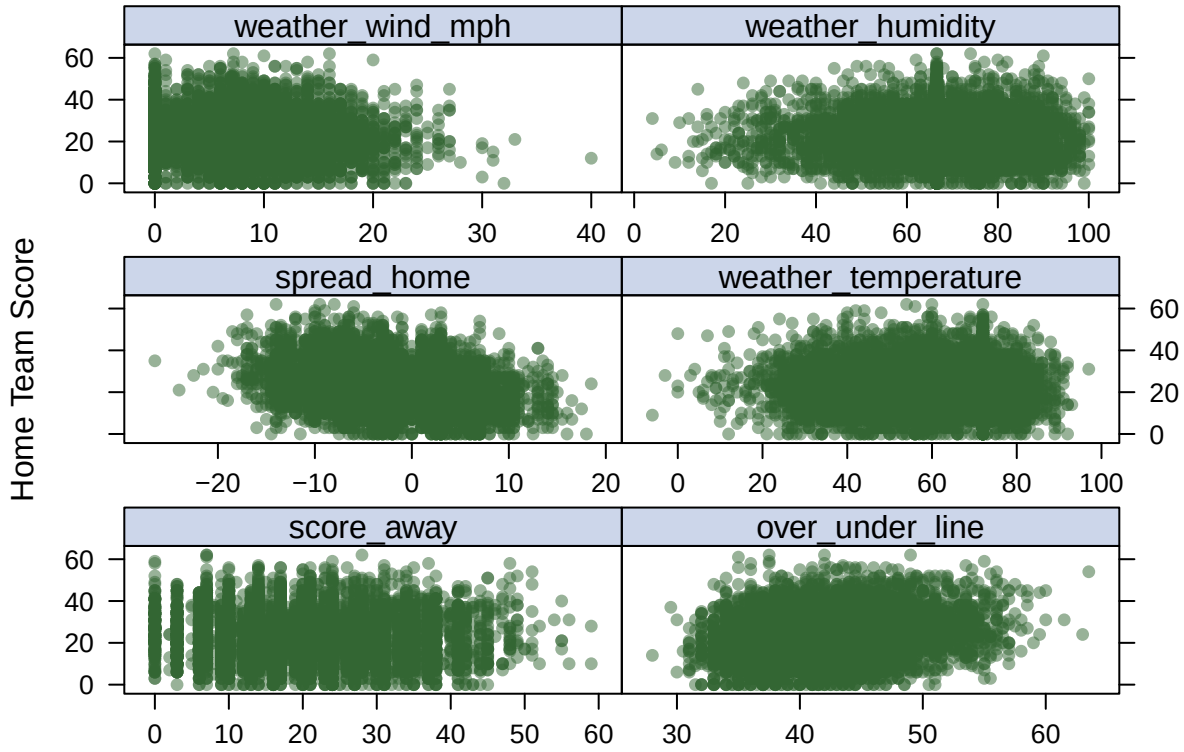
Visualize of the correlation among continuous variables, and the linearity among continuous predictors against the response.

```
corrplot(cor(train_cont_data), method = "circle", type="full", tl.cex = 0.5)
```




```
data_theme1 = trellis.par.get()
data_theme1$plot.symbol$col = rgb(.2, .4, .2, .5)
```

```
data_theme1$plot.symbol$pch = 16
data_theme1$plot.line$col = rgb(.8, .1, .1, 1)
data_theme1$plot.line$lwd = 2
data_theme1$strip.background$col = rgb(.0, .2, .6, .2)
trellis.par.set(data_theme1)

featurePlot(train_cont_pred, train_resp,
            plot = "scatter", labels = c("", "Home Team Score"),
            type = c("p"), layout = c(2, 3))
```



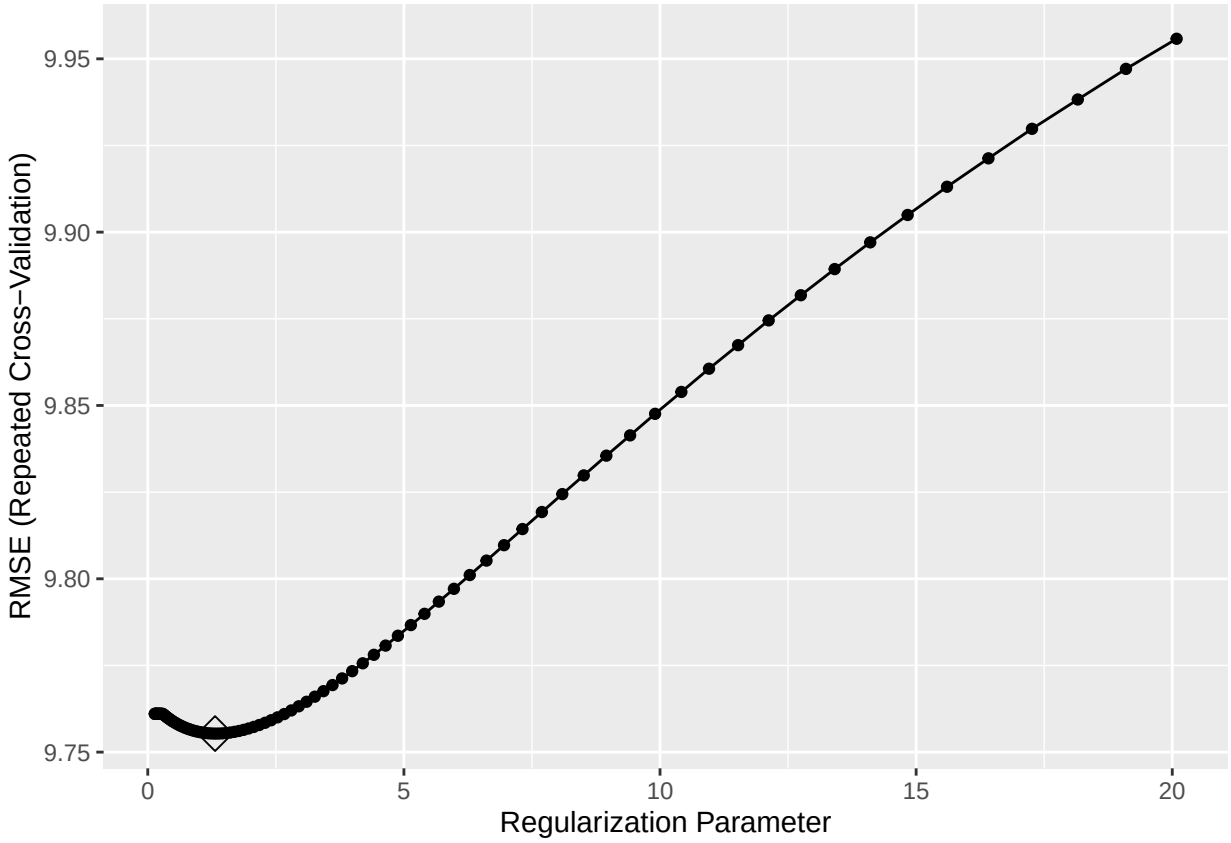
Fit a k-nearest neighbor model. Might have to rely on the Cluster.

```
set.seed(2022)
k_grid = expand.grid(k = seq(from = 1, to = 5, by = 1))

knn_fit = train(train_pred, train_resp,
                method = "knn",
                trControl = ctrl,
                tuneGrid = k_grid)
ggplot(knn_fit)

knn_fit$bestTune[1,1]
```

Fit linear models

```
set.seed(2022)

# Fit a multiple linear model
lm_fit = train(train_pred, train_resp,
               method = "lm",
               trControl = ctrl)
```

```
# Make prediction on test data
lm_pred = predict(lm_fit, newdata = test_pred)

# Fit a ridge model (L2 regularization, alpha = 0)
ridge_fit = train(train_pred, train_resp,
                  method = "glmnet",
                  tuneGrid = expand.grid(alpha = 0,
                                         lambda = exp(seq(-2, 3, length=100))),
                  preProc = c("center", "scale"),
                  trControl = ctrl)
ggplot(ridge_fit, highlight = TRUE) # could xTrans = log from plot() be implemented somehow?
```




```
# Get the optimal penalty term lambda
ridge_fit$bestTune
```

```
##    alpha   lambda
## 46     0 1.313542
```

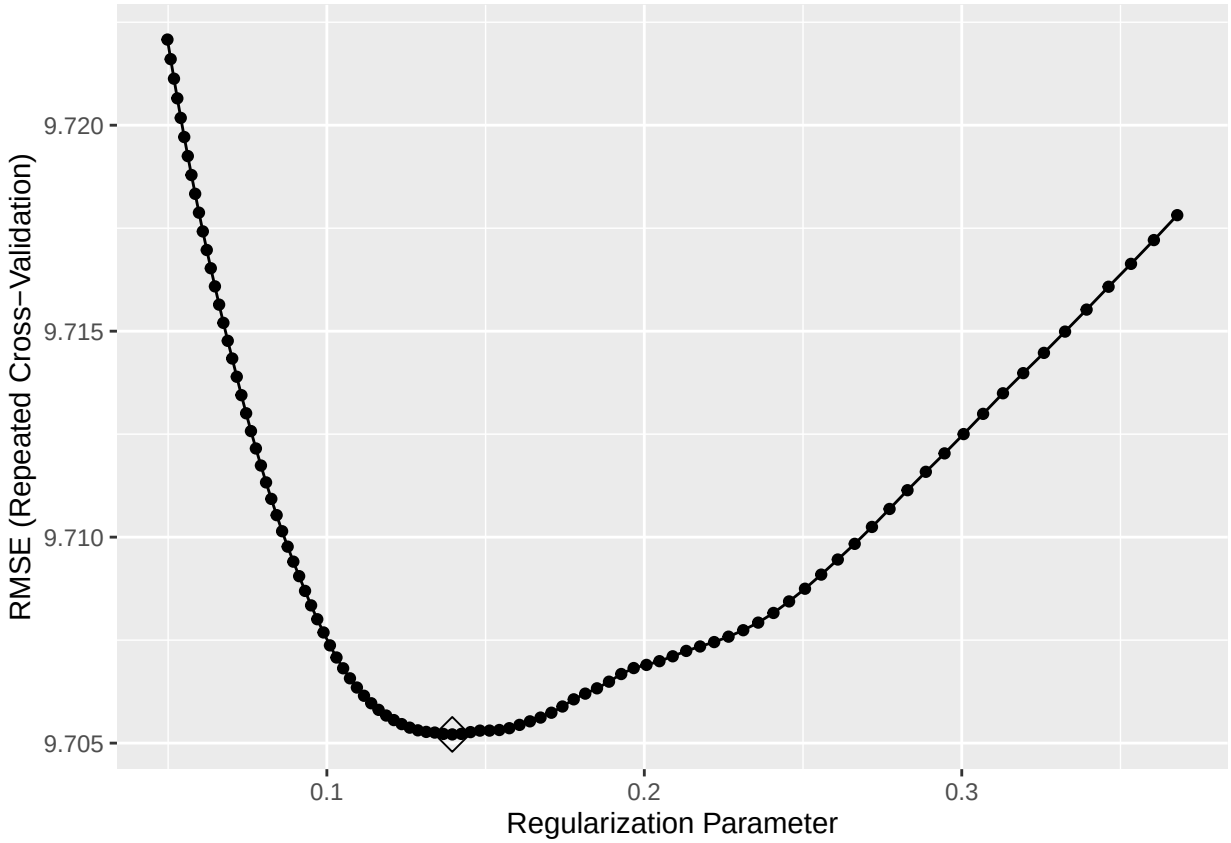
```
# Make prediction on test data and evaluate model using test MSE
ridge_pred = predict(ridge_fit, newdata = test_pred)
ridge_tmse = mean((test_resp - ridge_pred)^2); ridge_tmse
```

```
## [1] 97.05659
```

```
# Count the number of non-zeroed predictors are left in the trained model
ridge_coef = coef(ridge_fit$finalModel, ridge_fit$bestTune$lambda)
ridge_coef_count = ridge_coef@p[2]; ridge_coef_count
```

```
## [1] 137
```

```
# Fit a lasso model (L1 regularization, alpha = 1)
lasso_fit = train(train_pred, train_resp,
                  method = "glmnet",
                  tuneGrid = expand.grid(alpha = 1,
                                         lambda = exp(seq(-3, -1, length=100))),
                  preProc = c("center", "scale"),
                  trControl = ctrl)
ggplot(lasso_fit, highlight = TRUE)
```




```
# Get the optimal penalty term lambda
lasso_fit$bestTune
```

```
##    alpha    lambda
## 52     1 0.1394991
```

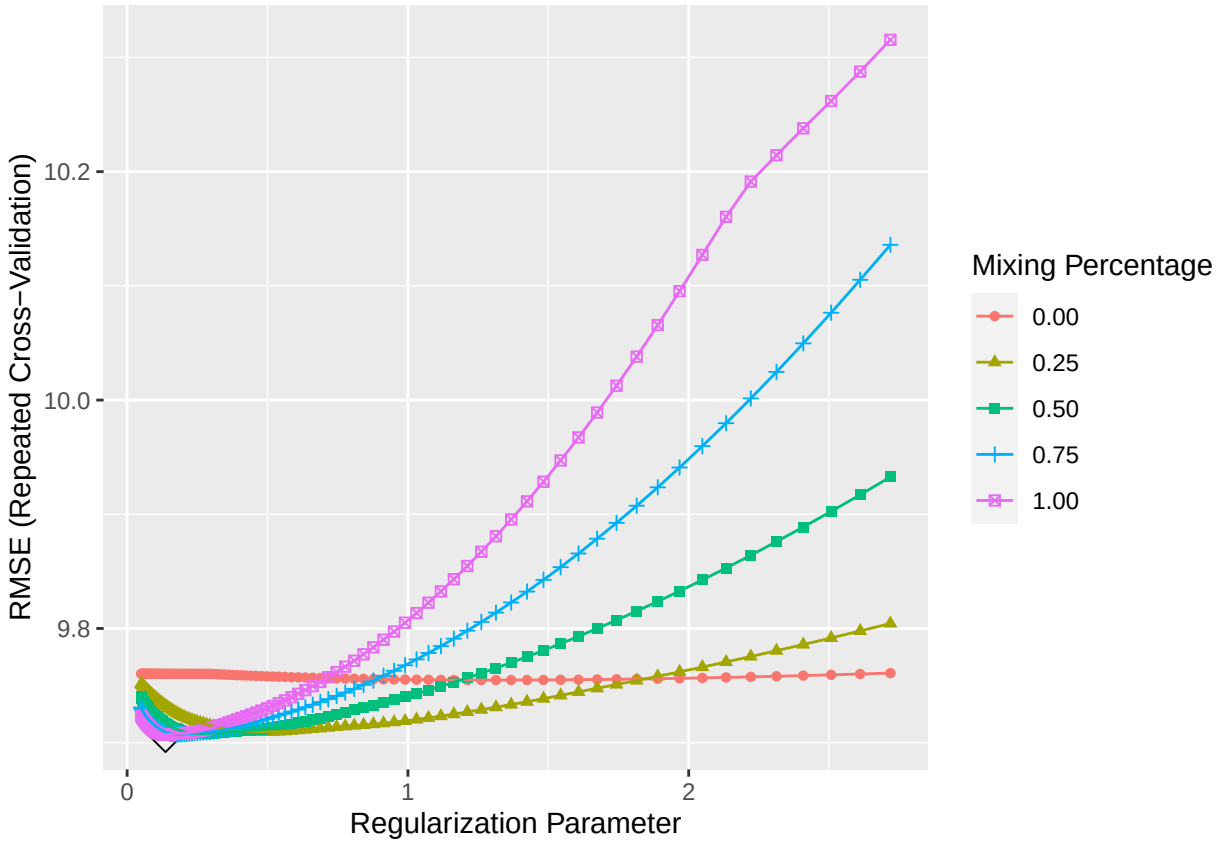
```
# Make prediction on test data and evaluate model using test MSE
lasso_pred = predict(lasso_fit, newdata = test_pred)
lasso_tmse = mean((test_resp - lasso_pred)^2); lasso_tmse
```

```
## [1] 96.4017
```

```
# Count the number of non-zeroed predictors are left in the trained model
lasso_coef = coef(lasso_fit$finalModel, lasso_fit$bestTune$lambda)
lasso_coef_count = lasso_coef@p[2]; lasso_coef_count
```

```
## [1] 39
```

```
# Fit an elastic net model (L1 & L2 regularization, alpha = [0,1])
enet_fit = train(train_pred, train_resp,
                 method = "glmnet",
                 tuneGrid = expand.grid(alpha = seq(0, 1, length = 5),
                                        lambda = exp(seq(1, -3, length = 100))),
                 trControl = ctrl)
ggplot(enet_fit, highlight = TRUE)
```



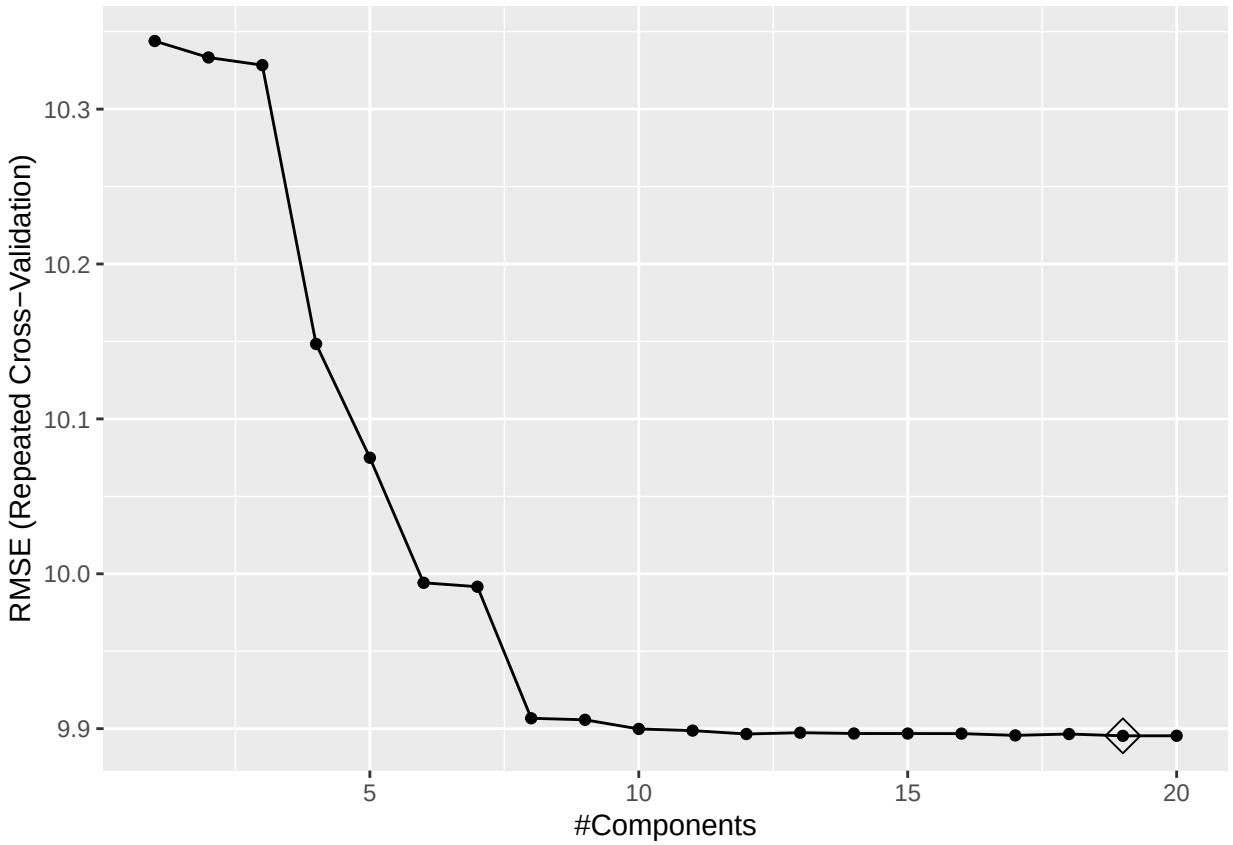

```
# Get tuning parameters alpha and lambda values. Alpha = 1, which matches lasso?
enet_fit$bestTune
```

```
##     alpha    lambda
## 426     1 0.1367092
```

```
# Make prediction on test data and evaluate model using test MSE
enet_pred = predict(enet_fit, newdata = test_pred)
enet_tmse = mean((test_resp - enet_pred)^2); enet_tmse
```

```
## [1] 96.40697
```

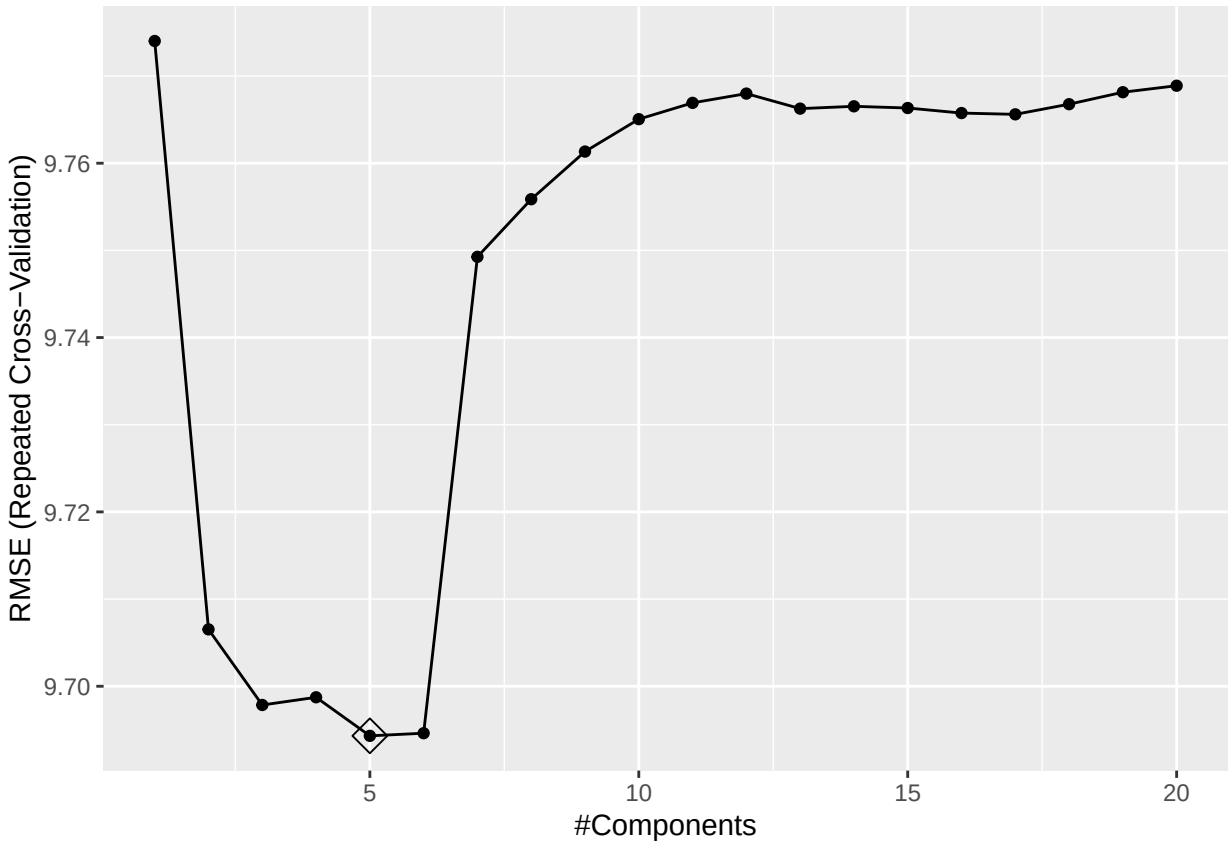
```
# Fit a principle component model
pcr_fit = train(train_pred, train_resp,
                  method = "pcr",
                  tuneGrid = data.frame(ncomp = 1:20),
                  trControl = ctrl,
                  scale = TRUE)
ggplot(pcr_fit, highlight = TRUE)
```




```
# Make prediction on test data and evaluate model using test MSE
pcr_pred = predict(pcr_fit, newdata = test_pred)
pcr_tmse = mean((test_resp - pcr_pred)^2); pcr_tmse
```

```
## [1] 99.587
```

```
# Fit a partial least squares model
pls_fit = train(train_pred, train_resp,
                method = "pls",
                tuneGrid = data.frame(ncomp = 1:20), ## ncomp range shortened for better graphing
                trControl = ctrl)
ggplot(pls_fit, highlight = TRUE)
```




```
# Make prediction on test data and evaluate model using test MSE
pls_pred = predict(pls_fit, newdata = test_pred)
pls_tmse = mean((test_resp - pls_pred)^2); pls_tmse
```

```
## [1] 96.00847
```

```
# Get the number of model components
pls_fit$bestTune
```

```
##   ncomp
## 5     5
```

Fit nonilinear models. Might have to rely on the Cluster.

```
mars_grid = expand.grid(degree = 1:3, nprune = 2:20)
mars_fit = train(train_pred, train_resp,
                 method = "earth",
                 tuneGrid = mars_grid,
                 trControl = ctrl)
```