Finals Code

Brian Jo Hsuan Lee | Jagjit Singh | Tanvir Khan

2022-05-10

Load packages

```
library(tidyverse)
library(caret)
library(earth)
library(rpart)
library(rpart.plot)
library(ranger)
library(visdat)
library(h2o)
```

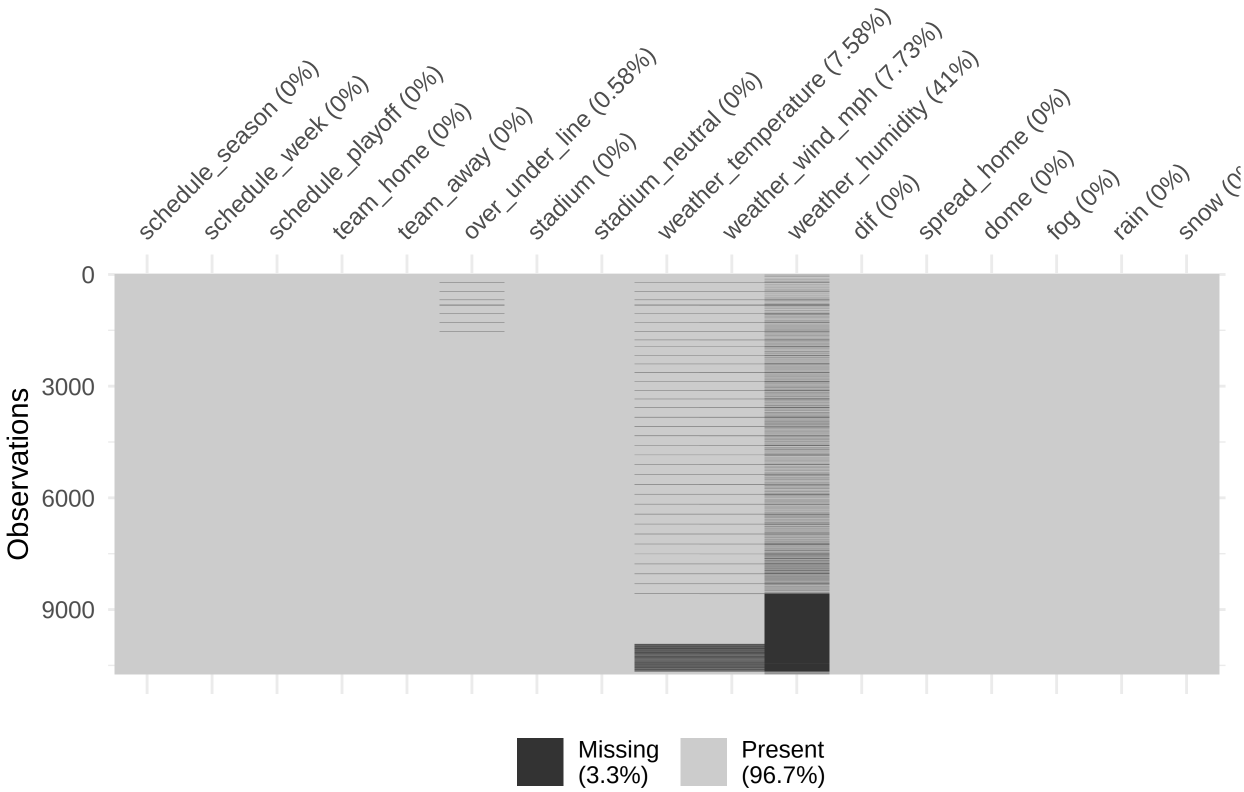
Clean data. Consolidate the historic team names into their corresponding current names; expand the weather detail column into boolean `dome`, `rain`, `fog`, and `snow` columns; update the spread values to those for the home team; rid seasons before 1979 due to incomplete betting line records; rid `schedule playoff` due to collinearity with the more informative `schedule week`; rid `spread_favored` and `weather details` as they are replaced by updated predictors; manually fill in the 2021 season SuperBowl final score as the dataset was created before said game.

```
data = read_csv("spreadspoke_scores.csv",
                    col_types = "iffffiiffddffiiic",
                    col_select = c("schedule_season":"weather_detail")) %>%
  filter(
    schedule_season %in% (1979:2021)
  ) %>%
  mutate(
    schedule_season = droplevels(schedule_season),
    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
    stadium = droplevels(stadium),
    score_home = ifelse(schedule_season == "2021" & schedule_week == "SuperBowl", 23, score_home),
    score_away = ifelse(schedule_season == "2021" & schedule_week == "SuperBowl", 20, score_away),
    dif = score_away - score_home,
    weather_detail = replace(weather_detail, is.na(weather_detail), "Dry"),
    weather_detail = factor(weather_detail),
    team_home = fct_collapse(team_home,
                             "Tennessee Titans" = c("Tennessee Titans", "Tennessee Oilers", "Houston Oil
                             "Washington Football Team" = c("Washington Football Team", "Washington Reds
                             "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 Oil
                              "Washington Football Team" = c("Washington Football Team", "Washington Reds
                              "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(-score_home, -score_away, -team_favorite_id, -spread_favorite, -weather_detail)
```

```
vis_miss(data)
```

```
## Warning: `gather_()` was deprecated in tidyr 1.2.0.
## Please use `gather()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```

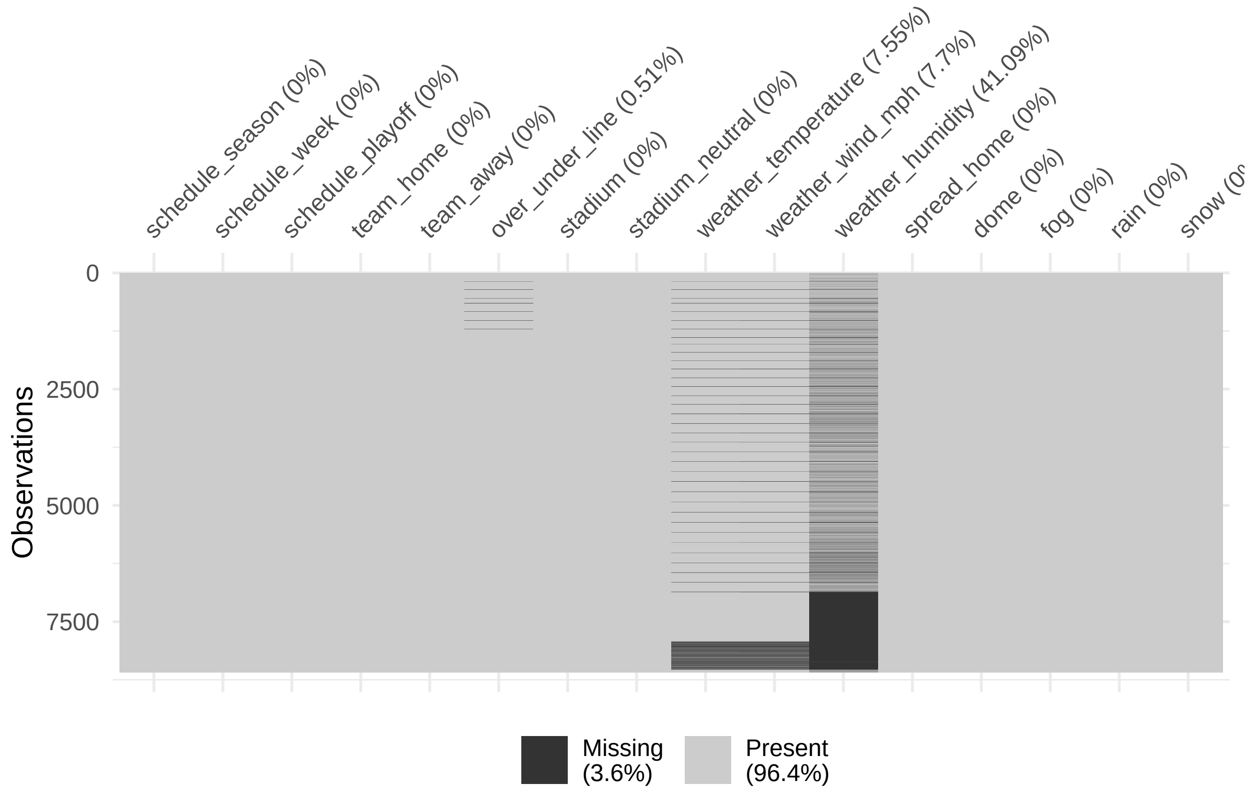


Transform Partition data into training and testing sets, and define the resampling method.

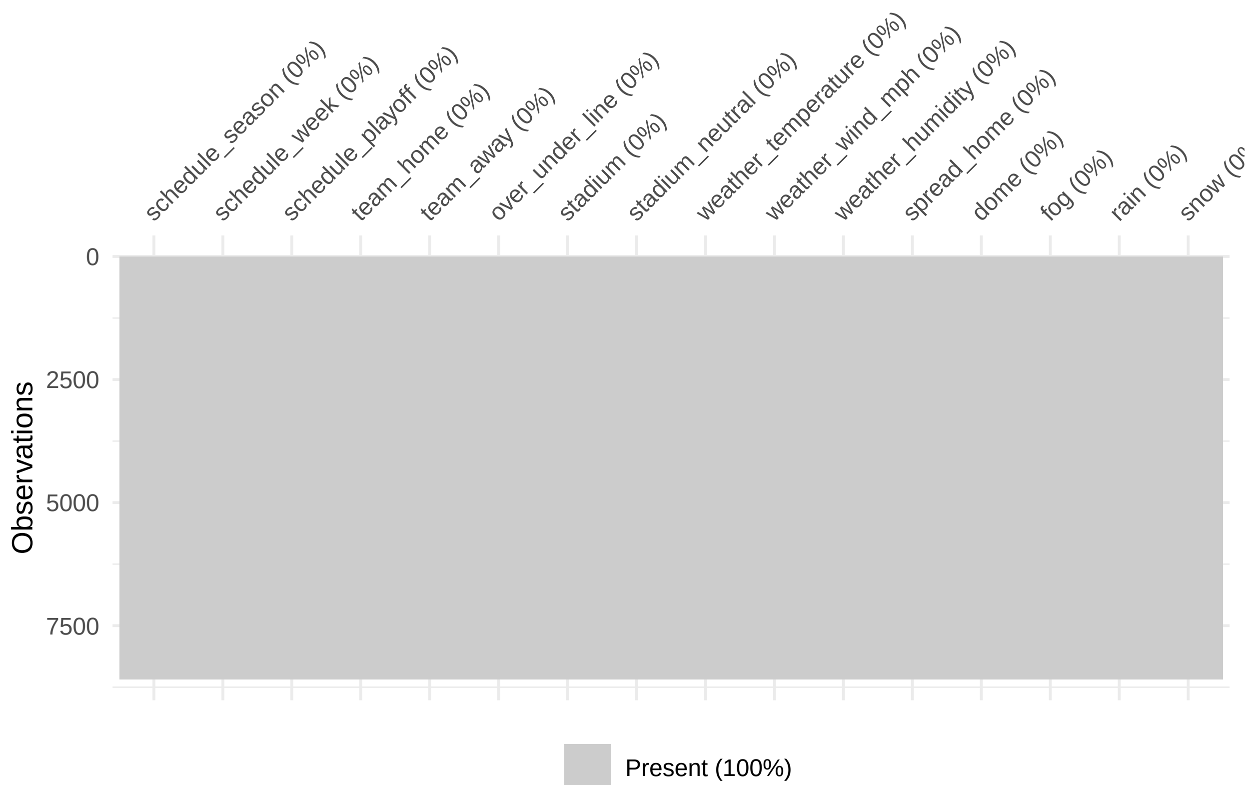
```
set.seed(2022)

# partition data into training and testing sets into randomized 4:1 splits
train_index = createDataPartition(y = data$dif, p = 0.8, list = FALSE)
train_pred =
  data %>%
  filter(row_number() %in% train_index) %>%
  select(-dif)
test_pred =
  data %>%
  filter(!row_number() %in% train_index) %>%
  select(-dif)
```

In the midterm, we replaced the NA values in weather temperature, humidity and wind speed with each of their grand averages. However, missing data imputation could be improved. Transformation on continuous predictors would also help improve predictions by scaling and standardizing their weights on the response. Here, we use the Yeo-Johnson transformation for non-positive numeric predictors and bag imputation to fill in the missing weather data.

```
vis_miss(train_pred)
```




```r
trans_imp = preProcess(train_pred, method = c("YeoJohnson", "zv", "bagImpute"))
ti_train_pred = predict(trans_imp, train_pred)
vis_miss(ti_train_pred)
```



```
ti_test_pred = predict(trans_imp, test_pred)
```

Re-generate our training and testing data for analysis. Reserve an unpartitioned dataset for exploratory analysis.

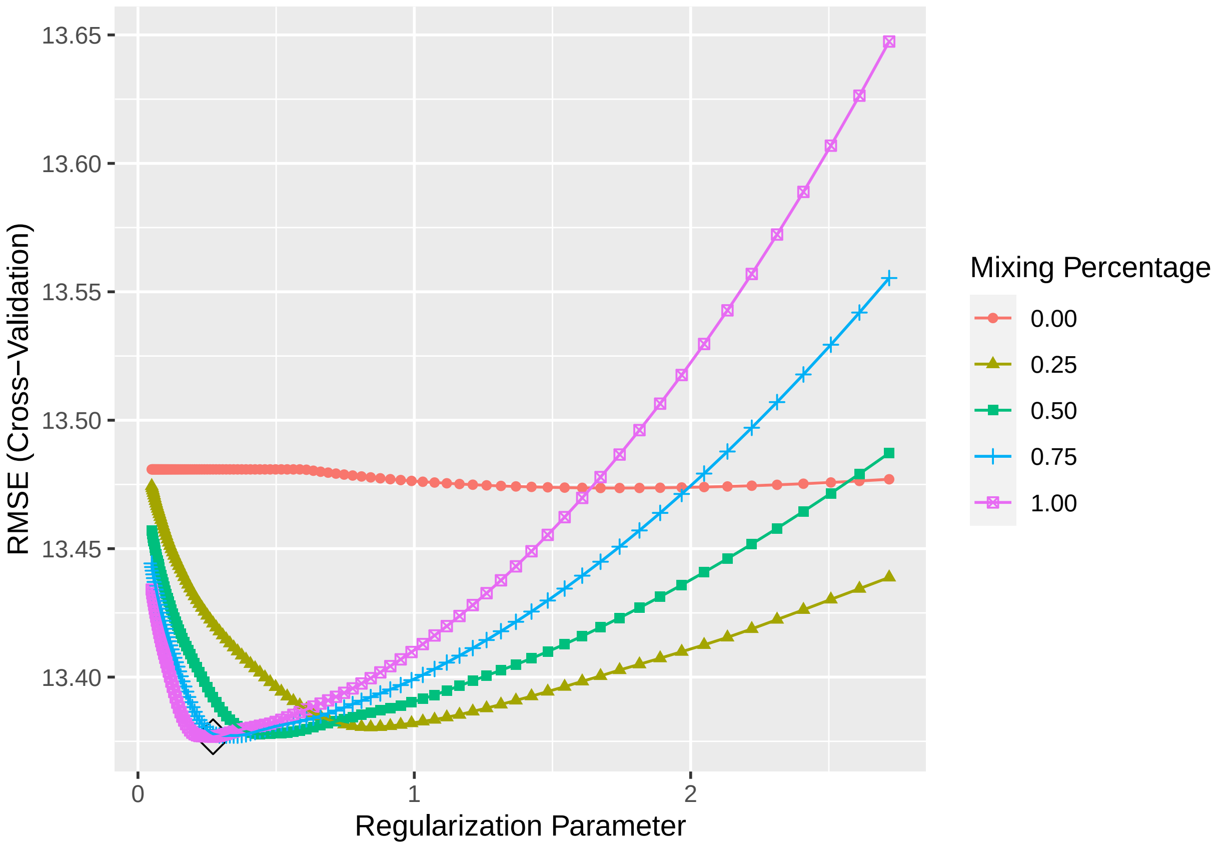
```
eda_data =
  bind_rows(
    data %>%
      filter(row_number() %in% train_index) %>%
      select(dif) %>%
      bind_cols(., ti_train_pred),
    data %>%
      filter(!row_number() %in% train_index) %>%
      select(dif) %>%
      bind_cols(., ti_test_pred))
train_data =
  data %>%
  filter(row_number() %in% train_index) %>%
  select(dif) %>%
  bind_cols(., ti_train_pred) %>%
  select(-schedule_season, -schedule_playoff)
test_data =
  data %>%
  filter(!row_number() %in% train_index) %>%
  select(dif) %>%
  bind_cols(., ti_test_pred) %>%
  select(-schedule_season, -schedule_playoff)
train_cont_data =
  train_data %>%
  select(dif, over_under_line, spread_home, weather_temperature, weather_wind_mph, weather_humidity)
```

Some model building methods may require matrices with indicator varibles for categorical predictors.

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

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

ctrl = trainControl(method = "cv")
```

Elastic Net is our preferred model from the midterm project. We will compare our new models with this.

```
set.seed(2022)
# Fit an elastic net model (L1 & L2 regularization, alpha = [0,1])
enet_fit = train(dif ~ .,
                 data = train_data,
                 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)
```




```
enet_fit$bestTune
```

```
##     alpha    lambda
## 443     1 0.2717072
```

```
# training RMSE
enet_prediction_train = predict(enet_fit, newdata = train_data)
rmse_enet_train = RMSE(enet_prediction_train, train_resp); rmse_enet_train
```

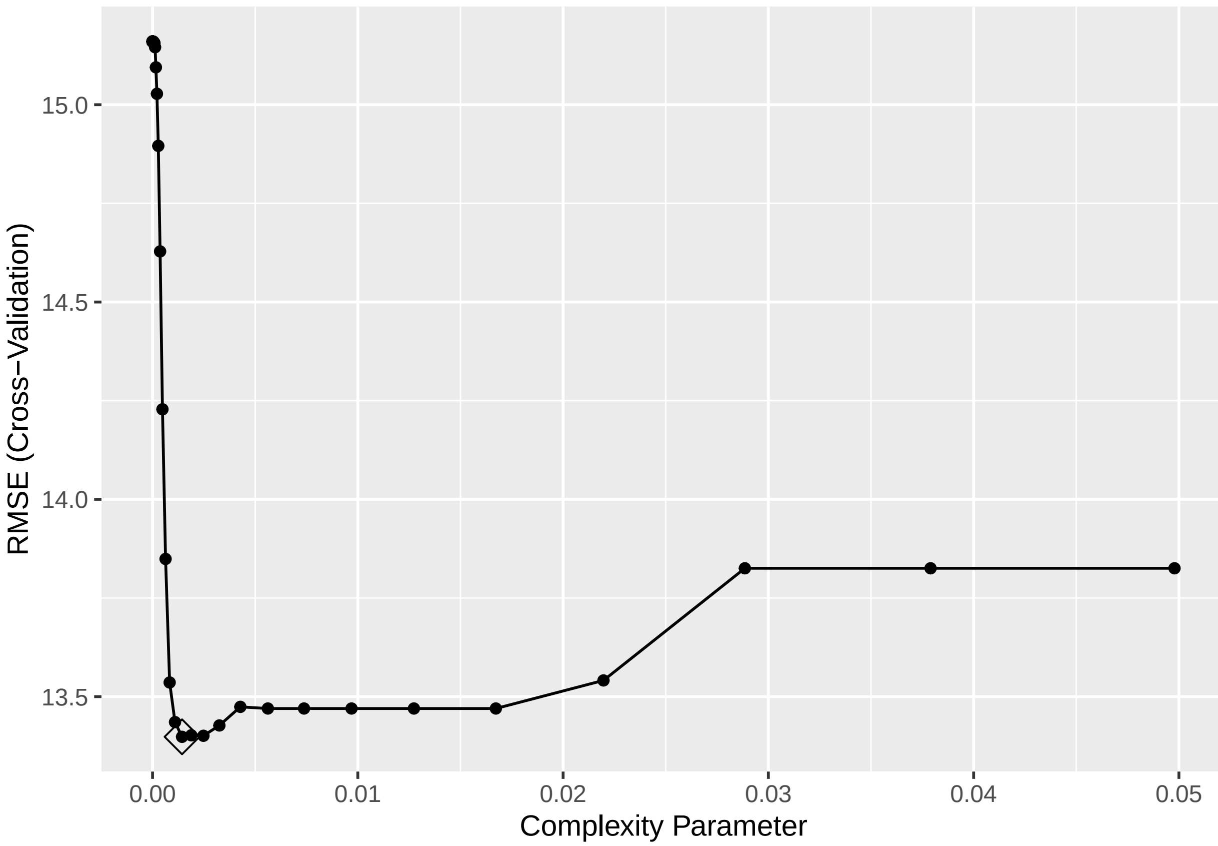
```
## [1] 13.33931
```

```
# testing RMSE
enet_prediction_test = predict(enet_fit, newdata = test_data)
rmse_enet_test <- RMSE(enet_prediction_test, test_resp); rmse_enet_test
```

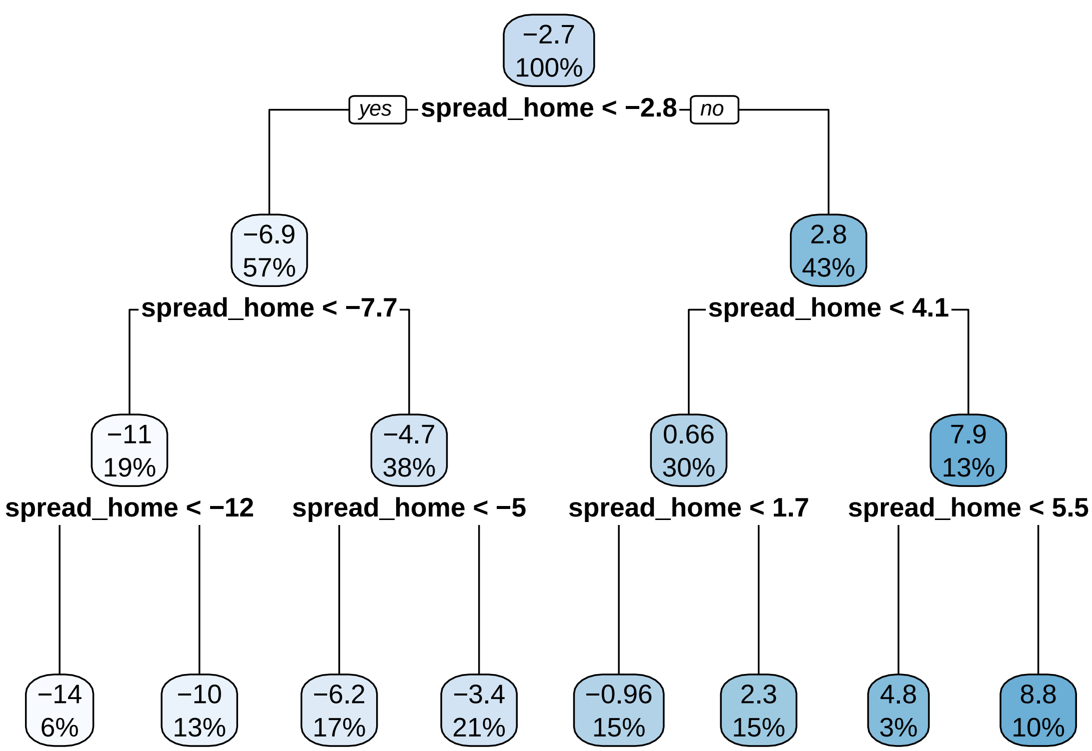
```
## [1] 13.43166
```

DECISION TREE - Using CARET

```
set.seed(2022)
rpart_fit = train(dif ~ .,
                  data = train_data,
                  method = "rpart",
                  tuneGrid = data.frame(cp = exp(seq(-30,-3, length = 100))),
                  trControl = ctrl)
ggplot(rpart_fit, highlight = TRUE)
```




```
rpart.plot(rpart_fit$finalModel)
```



```
rpart_fit$bestTune
```

```
##              cp
## 87 0.001436631
```

```
# training RMSE
dtree_prediction_train = predict(rpart_fit, newdata = train_data)
rmse_dtree_train = RMSE(dtree_prediction_train, train_resp); rmse_dtree_train
```

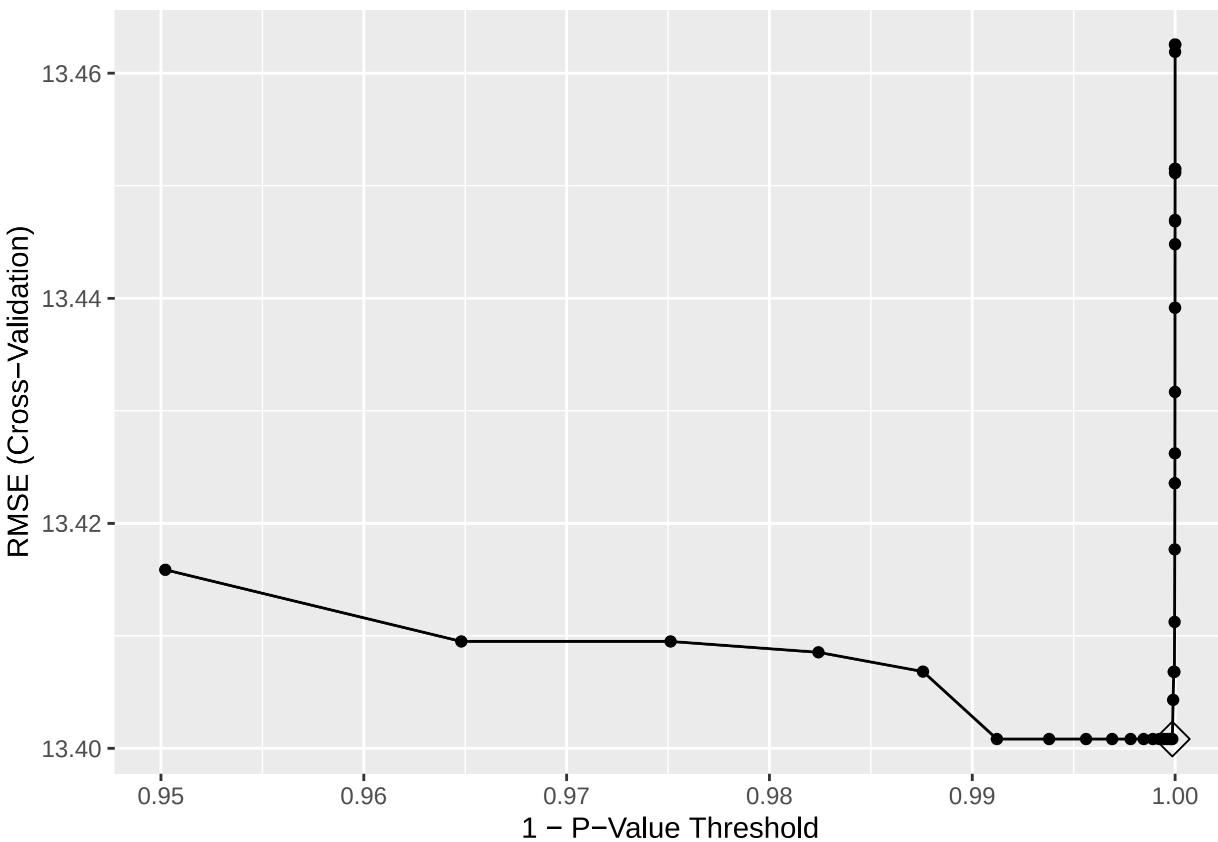
```
## [1] 13.35145
```

```
# testing RMSE
dtree_prediction_test = predict(rpart_fit, newdata = test_data)
rmse_dtree_test <- RMSE(dtree_prediction_test, test_resp); rmse_dtree_test
```

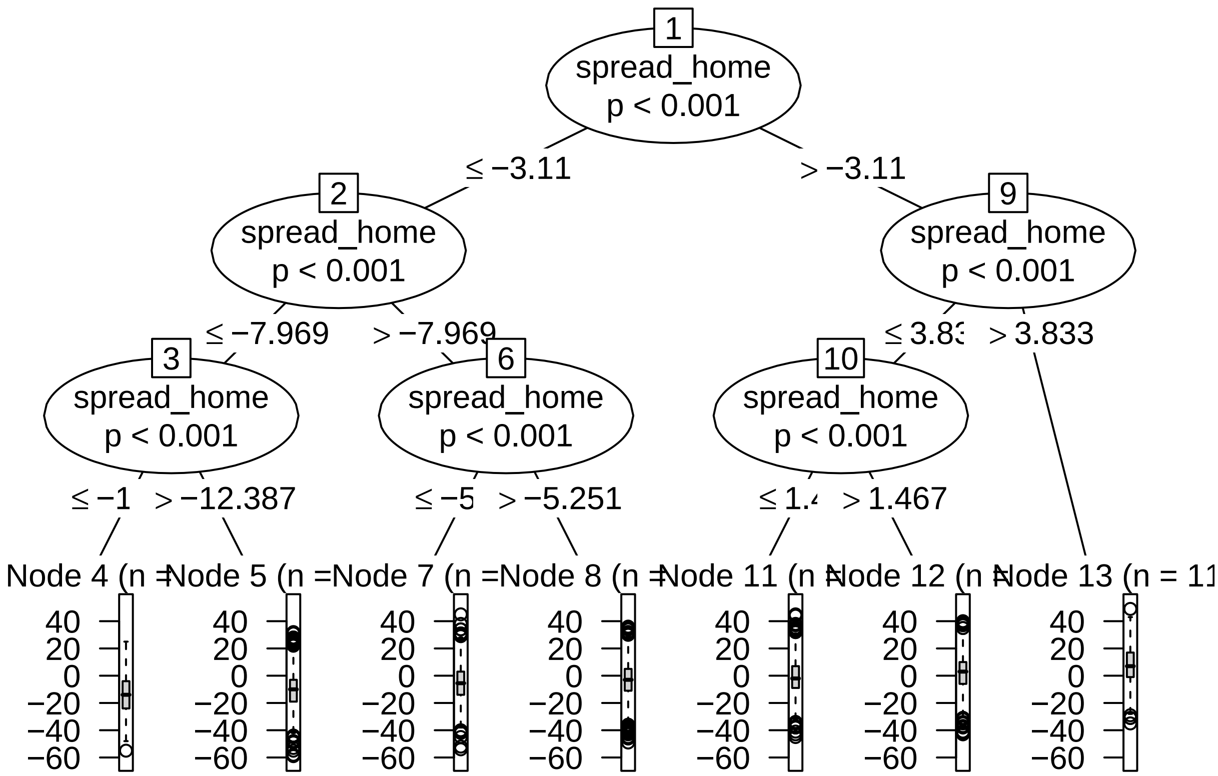
```
## [1] 13.4683
```

CONDITIONAL INFERENCE TREE - Using CARET

```
set.seed(2022)
ctree_fit = train(dif ~ . ,
                  train_data,
                  method = "ctree",
                  tuneGrid = data.frame(mincriterion = 1-exp(seq(-20, -3, length = 50))),
                  trControl = ctrl)
ggplot(ctree_fit, highlight = TRUE)
```




```
plot(ctree_fit$finalModel)
```




```
ctree_fit$bestTune
```

```
##    mincriterion
## 18    0.9998633
```

```
# training RMSE
ctree_prediction_train = predict(ctree_fit, newdata = train_data)
rmse_ctree_train = RMSE(ctree_prediction_train, train_resp); rmse_ctree_train
```

```
## [1] 13.36529
```

```
# testing RMSE
ctree_prediction_test = predict(ctree_fit, newdata = test_data)
rmse_ctree_test = RMSE(ctree_prediction_test, test_resp); rmse_ctree_test
```

```
## [1] 13.46514
```

Random Forest - Using Caret

```
rf_grid = expand.grid(mtry = c(1:14),
                      splitrule = "variance",
                      min.node.size = 1:6)
set.seed(2022)
rf_fit = train(dif ~ . ,
               train_data,
               method = "ranger",
               tuneGrid = rf_grid,
               trControl = ctrl)
ggplot(rf_fit, highlight = TRUE)

rf_fit$bestTune
```

```r
# training RMSE: 8.354
rf_prediction_train = predict(rf_fit, newdata = train_data)
rmse_rf_train = RMSE(rf_prediction_train, train_resp); rmse_rf_train

# testing RMSE: 13.519
rf_prediction_test = predict(rf_fit, newdata = test_data)
rmse_rf_test = RMSE(rf_prediction_test, test_resp); rmse_rf_test
```

GBM - Using CARET

```r
gbm_grid = expand.grid(n.trees = c(2000,3000, 4000, 5000),
                       interaction.depth = 1:5,
                       shrinkage = c(0.001,0.003,0.005),
                       n.minobsinnode = 10)
set.seed(2022)
gbm_fit = train(dif ~ . ,
                train_data,
                method = "gbm",
                tuneGrid = gbm.grid,
                trControl = ctrl,
                verbose = FALSE)
ggplot(gbm_fit, highlight = TRUE)

gbm_fit$bestTune

# training RMSE: 13.101
gbm_prediction_train = predict(gbm_fit, newdata = train_data)
rmse_gbm_train = RMSE(gbm_prediction_train, train_resp); rmse_gbm_train

# testing RMSE: 13.735
gbm_prediction_test = predict(gbm_fit, newdata = test_data)
rmse_gbm_test = RMSE(gbm_prediction_test, test_resp); rmse_gbm_test
```

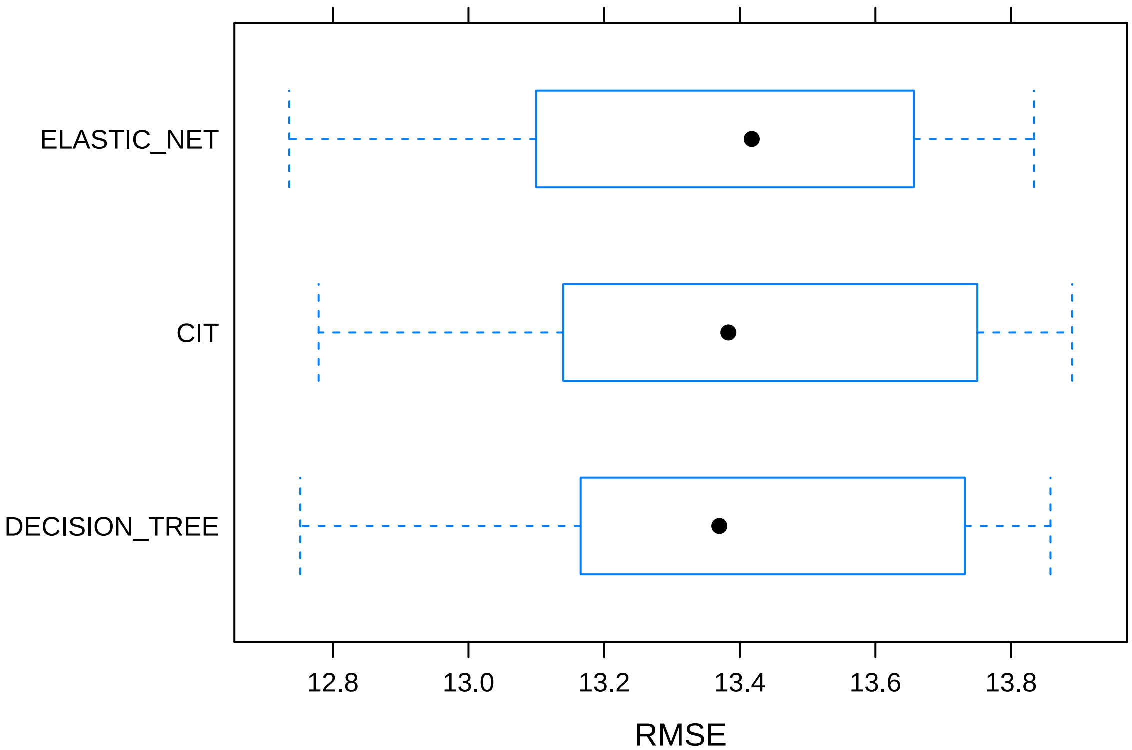
Elastic Net, Decision tree, Conditional Inference Tree validated RMSE comparison

```r
resamp = resamples(list("ELASTIC_NET" = enet_fit,
                        "DECISION_TREE" = rpart_fit,
                        "CIT" = ctree_fit))
summary(resamp)
```

```
##
## Call:
## summary.resamples(object = resamp)
##
## Models: ELASTIC_NET, DECISION_TREE, CIT
## Number of resamples: 10
##
## MAE
##                    Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## ELASTIC_NET    9.977236 10.25643 10.43413 10.44009 10.60643 10.92765    0
## DECISION_TREE  9.999808 10.32098 10.48435 10.49321 10.70548 10.92080    0
## CIT           10.005226 10.35001 10.47121 10.49196 10.72462 10.88887    0
##
## RMSE
##                   Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
```

```
## ELASTIC_NET   12.73563 13.10450 13.41766 13.37677 13.64850 13.83385    0
## DECISION_TREE 12.75197 13.16794 13.36984 13.39819 13.71048 13.85815    0
## CIT           12.77902 13.14649 13.38318 13.40083 13.72197 13.89023    0
##
## Rsquared
##                    Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## ELASTIC_NET   0.1305396 0.1439525 0.1687585 0.1664141 0.1845589 0.2166575    0
## DECISION_TREE 0.1221658 0.1420837 0.1620009 0.1634406 0.1806421 0.2111783    0
## CIT           0.1184573 0.1428264 0.1669554 0.1629872 0.1781092 0.2069257    0
```

```
bwplot(resamp, metric = "RMSE")
```



Elastic Net, Decision tree, Conditional Inference tree, Random Forest, Gradient Boosted Model validated RMSE comparison

```
resamp = resamples(list("ELASTIC_NET" = enet_fit,
                        "DECISION_TREE" = rpart_fit,
                        "CIT" = ctree_fit,
                        "RF" = rf_fit,
                        "GBM" = gbm_fit))
summary(resamp)
bwplot(resamp, metric = "RMSE")
```

The super learner is an ensemble method using a variety of models as its base models. I want to build one using RF and GBM, so will need the optimal tuning parameters.

```
h2o.init()
```

2 model ensemble

```
# Data preparation
train_h2o = as.h2o(train_data)
test_h2o = as.h2o(test_data)
```

```r
y = "dif"
x = setdiff(names(train_h2o), y)
nfolds = 10

# Train & cross-validate a GBM:
h2o_gbm = h2o.gbm(x = x,
                  y = y,
                  training_frame = train_h2o,
                  distribution = "gaussian",
                  ntrees = 4000,
                  max_depth = 5,
                  min_rows = 10,
                  learn_rate = 0.001,
                  nfolds = nfolds,
                  keep_cross_validation_predictions = TRUE,
                  seed = 2022)

# Train & cross-validate a RF:
h2o_rf = h2o.randomForest(x = x,
                          y = y,
                          training_frame = train_h2o,
                          mtries = 14,
                          min_rows = 5,
                          nfolds = nfolds,
                          keep_cross_validation_predictions = TRUE,
                          seed = 2022)

# Train a stacked ensemble using the GBM and RF above:
ensemble = h2o.stackedEnsemble(x = x,
                               y = y,
                               training_frame = train_h2o,
                               keep_cross_validation_predictions = TRUE,
                               base_models = list(h2o_gbm, h2o_rf))

# Evaluate ensemble performance on training set
train_perf = h2o.performance(ensemble, newdata = train_h2o)
ensemble_train_rmse = h2o.rmse(train_perf); ensemble_train_rmse

# Evaluate ensemble performance on test set
perf = h2o.performance(ensemble, newdata = test_h2o)
ensemble_test_rmse = h2o.rmse(perf); ensemble_test_rmse

# Compare the ensemble to GBM and RF (baseline learners) performance on the training set
perf_gbm_train = h2o.performance(h2o_gbm, newdata = train_h2o)
perf_rf_train = h2o.performance(h2o_rf, newdata = train_h2o)
baselearner_best_train_rmse = min(h2o.rmse(perf_gbm_train),
                                  h2o.rmse(perf_rf_train)); baselearner_best_train_rmse

# Compare the ensemble to GBM and RF (baseline learners) performance on the test set
perf_gbm_test = h2o.performance(h2o_gbm, newdata = test_h2o)
perf_rf_test = h2o.performance(h2o_rf, newdata = test_h2o)
baselearner_best_test_rmse = min(h2o.rmse(perf_gbm_test),
                                 h2o.rmse(perf_rf_test)); baselearner_best_test_rmse
```

```
print(sprintf("Best Base-learner Test RMSE: %s", baselearner_best_auc_test))
print(sprintf("Ensemble Test RMSE: %s", ensemble_test_rmse))
```

```
h2o.shutdown()
```