Tree Based Methods Assignment

Luke Clement

In this assignment, you are going to explore fitting, evaluating, and comparing different tree-based methods to two different dataset, one is a classification task and one is a regression task. The classification data is a set of 78 mortgages with various pieces of information about them and you will be interested in classifying the mortgage rate. The regression data is about a professor's salary based on different characteristics about them.

```
# load both datasets here
mortgages <- read_csv("mortgages.csv")
prof_salary <- read_csv("prof_salary.csv")
```

Instructions

Don't put everything into one block.

For this assignment, I want you to complete the following:

- There is a small amount of preprocessing that needs to be done before you can start fitting models. The first column in each dataset is an index variable that should not be included. Remove them and comment on why an index column would be harmful to the modeling process.

```
# Refining
#################
mortgages <- mortgages[,-1] %>% clean_names() %>%
  mutate(rate = as.factor(rate)) %>% drop_na()
prof_salary <- prof_salary[,-1] %>% clean_names() %>% drop_na()
```

The index would be harmful because the index is randomly assigned to each data point and doesn't have anything to do with the actual data. Therefore it could damage the model by creating false correlations that don't actual exist.

- For both datasets, split the data into training and testing sets. Use an 85/15 split for mortgages and a 90/10 split for professor salaries

```
# Setup for models
#######################
set.seed(54321)
mortgage_split <- initial_split(mortgages, prop = 0.85)
m_training <- training(mortgage_split)
m_testing <- testing(mortgage_split)

prof_split <- initial_split(prof_salary, prop = 0.9)
p_training <- training(prof_split)
p_testing <- testing(prof_split)

p_samples <- vfold_cv(p_training)
m_samples <- vfold_cv(m_training)
p_recipe <- recipe(salary ~ ., p_training)
m_recipe <- recipe(rate ~ ., m_training)
```

- Fit a decision tree to both datasets using the training data and visualize the trees. You can prune the trees if you want, but it isn't required. Use `rpart()` to fit the models and `rpart.plot()` to visualize the trees. Use all of the columns except for the response as predictors.

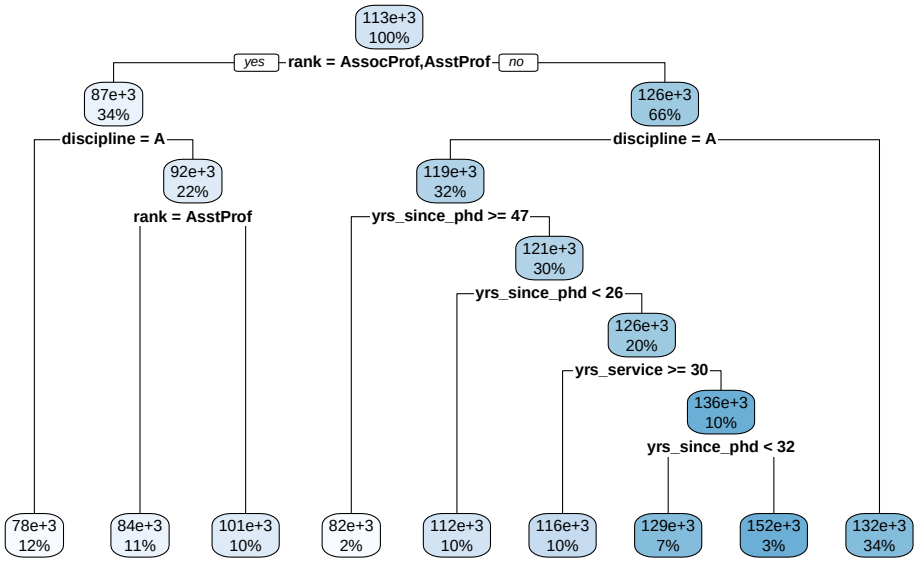
I decided to run all types of classification and Regression twice with different seeds so that I could compare different splits of data in relation to the type of tree based method. I have also run some of the models a couple times over the course of working on this homework. That is how I got my data. Note only the most recent seed data will be rendered in the document.

```
# decision tree with rpart
alpha_seq <- seq(0, 1, by = 0.01)
prof_split_decision <- function(data){
    split <- initial_split(data, prop = 0.9)
    train <- training(split)
    test <- testing(split)

prof_findrmse <- function(prunefactor) {
    prof_fit <- rpart(salary ~ ., data = train)
    prune_fit <- prune.rpart(prof_fit, prunefactor)
    prune_preds <- predict(prune_fit, newdata = test)
    prune_rmse <- sqrt(mean((test$salary - prune_preds)^2))
    prune_rmse
```

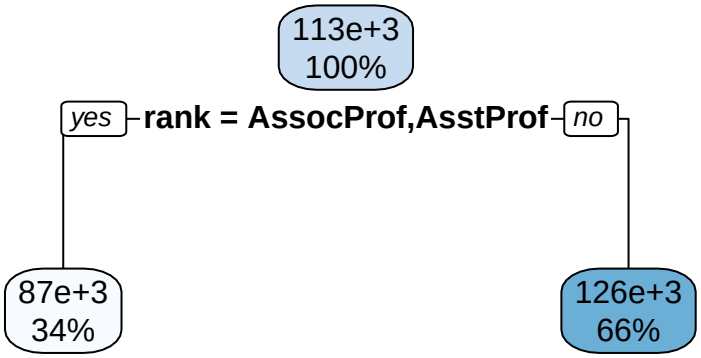
```
}

  map_dbl(alpha_seq, prof_findrmse)

}

prof_split_decision(prof_salary)
```

```
 [1] 24950.16 24950.16 24841.02 25154.82 26270.74 26270.74 26270.74 26270.74
 [9] 26270.74 26270.74 26270.74 26270.74 26270.74 26270.74 26270.74 26270.74
[17] 26270.74 26270.74 26270.74 26270.74 26270.74 26270.74 26270.74 26270.74
[25] 26270.74 26270.74 26270.74 26270.74 26270.74 26270.74 26270.74 26270.74
[33] 26270.74 26270.74 26270.74 26270.74 26270.74 26270.74 26270.74 32442.37
[41] 32442.37 32442.37 32442.37 32442.37 32442.37 32442.37 32442.37 32442.37
[49] 32442.37 32442.37 32442.37 32442.37 32442.37 32442.37 32442.37 32442.37
[57] 32442.37 32442.37 32442.37 32442.37 32442.37 32442.37 32442.37 32442.37
[65] 32442.37 32442.37 32442.37 32442.37 32442.37 32442.37 32442.37 32442.37
[73] 32442.37 32442.37 32442.37 32442.37 32442.37 32442.37 32442.37 32442.37
[81] 32442.37 32442.37 32442.37 32442.37 32442.37 32442.37 32442.37 32442.37
[89] 32442.37 32442.37 32442.37 32442.37 32442.37 32442.37 32442.37 32442.37
[97] 32442.37 32442.37 32442.37 32442.37 32442.37
```

```
# fit
p_full_fit <- rpart(salary ~ ., data = p_training)
p_prune_fit <- prune.rpart(p_full_fit, 0.10)

# Preds
p_full_preds <- predict(p_full_fit, newdata = p_testing)
p_prune_preds <- predict(p_prune_fit, newdata = p_testing)

# visualizing the decision tree
rpart.plot(p_full_fit)
```




```
rpart.plot(p_prune_fit)
```




```
p_pred_decision <- tibble(salary = p_testing$salary,
                          pred = p_prune_preds,
                          full = p_full_preds)

# data
p_pred_decision %>% metrics(salary, full)
```

```
# A tibble: 3 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
```

```
1 rmse    standard   24326.
2 rsq     standard       0.469
3 mae     standard   16663.
```

```
p_pred_decision %>% metrics(salary, pred)
```

```
# A tibble: 3 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard   25876.
2 rsq     standard       0.411
3 mae     standard   19236.
```

```
# decision tree with rpart
alpha_seq <- seq(0, 1, by = 0.01)
mortgage_split_decision <- function(data){
    split_data <- initial_split(data, prop = 0.85)
    train <- training(split_data)
    test <- testing(split_data)

findAccuracy <- function(prunefactor) {
    full_fit <- rpart(rate ~ ., data = train)
    prune_fit <- prune.rpart(full_fit, prunefactor)
    prune_preds <- predict(prune_fit, newdata = test)[,1]
    prune_preds2 <- case_when(
        prune_preds < .5 ~ "fixed",
        prune_preds >= .5 ~ "adjustable"
    )
    accuracy <- mean(test$rate == prune_preds2)
    accuracy
}

  map_dbl(alpha_seq, findAccuracy)

}

mortgage_split_decision(mortgages)
```

```
[1] 0.7500000 0.7500000 0.7500000 0.7500000 0.7500000 0.7500000 0.7500000
```

```
 [8] 0.7500000 0.7500000 0.7500000 0.7500000 0.7500000 0.7500000 0.7500000
[15] 0.7500000 0.7500000 0.7500000 0.7500000 0.7500000 0.7500000 0.7500000
[22] 0.7500000 0.7500000 0.7500000 0.7500000 0.7500000 0.7500000 0.7500000
[29] 0.7500000 0.7500000 0.7500000 0.7500000 0.7500000 0.7500000 0.7500000
[36] 0.7500000 0.7500000 0.7500000 0.7500000 0.7500000 0.7500000 0.7500000
[43] 0.7500000 0.7500000 0.7500000 0.7500000 0.7500000 0.7500000 0.7500000
[50] 0.7500000 0.7500000 0.7500000 0.5833333 0.5833333 0.5833333 0.5833333
[57] 0.5833333 0.5833333 0.5833333 0.5833333 0.5833333 0.5833333 0.5833333
[64] 0.5833333 0.5833333 0.5833333 0.5833333 0.5833333 0.5833333 0.5833333
[71] 0.5833333 0.5833333 0.5833333 0.5833333 0.5833333 0.5833333 0.5833333
[78] 0.5833333 0.5833333 0.5833333 0.5833333 0.5833333 0.5833333 0.5833333
[85] 0.5833333 0.5833333 0.5833333 0.5833333 0.5833333 0.5833333 0.5833333
[92] 0.5833333 0.5833333 0.5833333 0.5833333 0.5833333 0.5833333 0.5833333
[99] 0.5833333 0.5833333 0.5833333
```

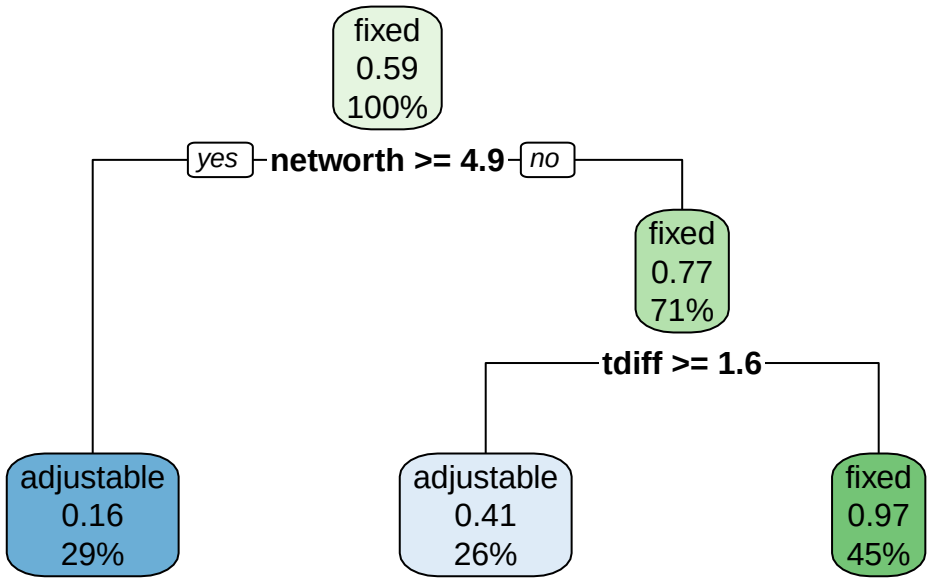
```r
m_full_fit <- rpart(rate ~ ., data = m_training)
m_prune_fit <- prune.rpart(m_full_fit, 0.15)

m_full_preds <- predict(m_full_fit, newdata = m_testing)[,1]
m_prune_preds <- predict(m_prune_fit, newdata = m_testing)

m_full_preds2 <- case_when(
    m_full_preds < .5 ~ "fixed",
    m_full_preds >= .5 ~ "adjustable"
)
m_prune_preds2 <- case_when(
    m_full_preds < .5 ~ "fixed",
    m_full_preds >= .5 ~ "adjustable"
)

(m_full_accuracy <- mean(m_testing$rate == m_full_preds2))
```

```
[1] 0.75
```

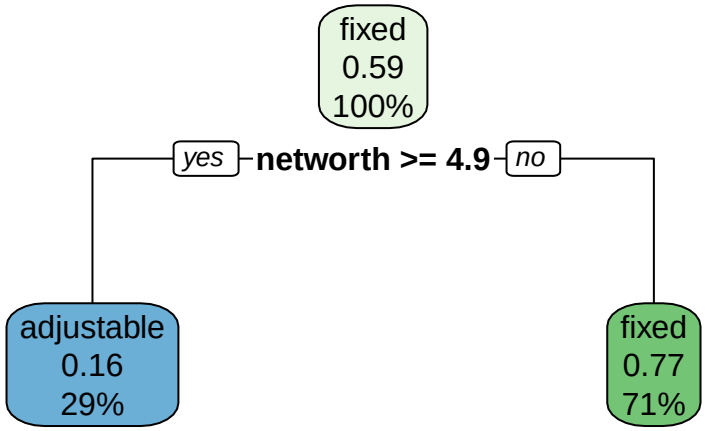
```r
(m_prune_accuracy <- mean(m_testing$rate == m_prune_preds2))
```

```
[1] 0.75
```

```
#visualizing the decision tree
rpart.plot(m_full_fit)
```




```
rpart.plot(m_prune_fit)
```




```
m_pred_decision <- tibble(rate = m_testing$rate,
                          pred = as.factor(m_prune_preds2),
                          full = as.factor(m_full_preds2))
```

```
m_pred_decision %>% metrics(rate, full)
```

```
# A tibble: 2 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary          0.75
2 kap      binary          0.5
```

```
m_pred_decision %>% metrics(rate, pred)
```

```
# A tibble: 2 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary          0.75
2 kap      binary          0.5
```

Comment: The two code chunks above were written in base R and I was able to produce a full tree and a pruned tree for regression and classification. The regression models varied between 20829 and 24326 RMSE for the full models and both were slightly higher with the pruned models. So pruning did not really help make these decision trees any better. I decided on .10 (regression) and .15 (classification) for the pruning factor after running the models a couple times and on occasion those prune factors would produce slightly better results than the full models. Also note, sometimes the pruned graph produced the same tree as the full graph.

- Make predictions on the testing data and calculate the test accuracy for each decision tree.

I think the classification for home rates is going to vary a lot because there is not a large sample size. On some trees you will probably see close to 100 % accuracy, while other trees will probably be around 50-60 % accuracy. This was confirmed by informal experiments I did over the course of the project.

In regard to the accuracy between models, I don't really know which will be the best. I feel like the decision tree being the most basic will probably have the lowest accuracy and highest RMSE, while Bagging and Forest will be about the same but better than the decision tree. I think Boost will be the best. My predictions roughly corresponds to the amount of depth each type of method will have. The more calculations a process does seems to be a reasonably good prediction for how a model will perform, although as we've seen in the past, that's not always the case.

- Now for the big part: Using tidymodels syntax, I want you to train, tune, test, and compare the 4 different tree-based methods (decision tree, bagged tree, random forest, boosted tree) using accuracy as your metric.

```
## Regression Prof Decision Tree
########################################

# set model
p_model <- decision_tree(mode = "regression",
                         cost_complexity = tune(),
                         tree_depth = tune()) %>%
    set_engine("rpart")

# workflow
p_wf <- workflow() %>%
    add_recipe(p_recipe) %>%
    add_model(p_model)

# create grid
p_tree_grid <- grid_regular(cost_complexity(),
                            tree_depth(),
                            levels = 5)

# does the tuning (takes some time to run)
p_tree_res <- p_wf %>%
    tune_grid(
        resamples = p_samples,
        grid = p_tree_grid
    )

# find the best collection of cost_complexity and tree_depth
p_best_tree <- p_tree_res %>%
    select_best("rmse")

# finalize the workflow
p_final_wf <- p_wf %>%
    finalize_workflow(p_best_tree)

# metrics of the tuned model
p_final_wf %>%
    last_fit(prof_split) %>%
    collect_metrics()
```

```
# A tibble: 2 x 4
  .metric .estimator .estimate .config
  <chr>   <chr>          <dbl> <chr>
1 rmse    standard   24888.    Preprocessor1_Model1
2 rsq     standard       0.465 Preprocessor1_Model1
```

```
# predictions of tuned model
pd_pred2 <- p_final_wf %>%
    last_fit(prof_split) %>%
    collect_predictions()

# metrics
pd_pred2 %>% metrics(salary, .pred)
```

```
# A tibble: 3 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard   24888.
2 rsq     standard       0.465
3 mae     standard   16574.
```

```
## Classification Mortgage Decision Tree
########################################

# create model, recipe, and workflow
m_model <- decision_tree(mode = "classification",
                            cost_complexity = tune(),
                            tree_depth = tune()) %>%
    set_engine("rpart")

# workflow
m_wf <- workflow() %>%
    add_recipe(m_recipe) %>%
    add_model(m_model)

# create grid
m_tree_grid <- grid_regular(cost_complexity(),
                            tree_depth(),
                            levels = 5)
```

```r
# does the tuning (takes some time to run)
m_tree_res <- m_wf %>%
    tune_grid(
        resamples = m_samples,
        grid = m_tree_grid
    )

# find the best collection of cost_complexity and tree_depth
m_best_tree <- m_tree_res %>%
    select_best("accuracy")

# finalize the workflow
m_final_wf <- m_wf %>%
    finalize_workflow(m_best_tree)

# metrics of the tuned model
m_final_wf %>%
    last_fit(mortgage_split) %>%
    collect_metrics()
```

```
# A tibble: 2 x 4
  .metric  .estimator .estimate .config
  <chr>    <chr>          <dbl> <chr>
1 accuracy binary         0.75  Preprocessor1_Model1
2 roc_auc  binary         0.843 Preprocessor1_Model1
```

```r
# predictions of tuned model
md_pred2 <- m_final_wf %>%
    last_fit(mortgage_split) %>%
    collect_predictions()

md_pred2  %>% metrics(rate, .pred_class)
```

```
# A tibble: 2 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary          0.75
2 kap      binary          0.5
```

I decided to tune cost complexity and tree_depth since that's what we did it in class. The two models above are decisions trees like we did before and so they should have about the same degree of accuracy as the two models that we did in base R. This is mostly true except that classification for the tidy verse was slightly worse getting about one more wrong than base R. It was also slightly worse with regression having an RMSE about 1000 above the base R model. These aren't too bad and it could have just been the division of training vs testing. So it's close to what I would have assumed but I feel they should have been a little bit closer since the tidyverse models should be tuned to be little better than the base R models. Overall, accuracy was slightly worse for the tidyverse models.

```
# Bagging Prof
########################################
pb_model <- bag_tree(mode = "regression",
                             cost_complexity = tune(),
                             tree_depth = tune()) %>%
      set_engine("rpart", times = 25)

pb_wf <- workflow() %>%
    add_recipe(p_recipe) %>%
    add_model(pb_model)

# create grid
pb_tree_grid <- grid_regular(cost_complexity(),
                          tree_depth(),
                          levels = 5)

# does the tuning (takes some time to run)
pb_tree_res <- pb_wf %>%
    tune_grid(
        resamples = p_samples,
        grid = pb_tree_grid
    )

# find the best collection of cost_complexity and tree_depth
pb_best_tree <- pb_tree_res %>%
    select_best("rmse")

# finalize the workflow
pb_final_wf <- pb_wf %>%
    finalize_workflow(pb_best_tree)

# metrics of the tuned model
```

```
pb_final_wf %>%
    last_fit(prof_split) %>%
    collect_metrics()
```

```
# A tibble: 2 x 4
  .metric .estimator .estimate .config
  <chr>   <chr>          <dbl> <chr>
1 rmse    standard   23243.    Preprocessor1_Model1
2 rsq     standard       0.541 Preprocessor1_Model1
```

```
# predictions of tuned model
pb_pred <- pb_final_wf %>%
    last_fit(prof_split) %>%
    collect_predictions()

# metrics
pb_pred %>% metrics(salary, .pred)
```

```
# A tibble: 3 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard   23273.
2 rsq     standard       0.531
3 mae     standard   15609.
```

```
# Bagging mortgage
########################################
mb_model <- bag_tree(mode = "classification",
                             cost_complexity = tune(),
                             tree_depth = tune()) %>%
    set_engine("rpart", times = 25)

mb_wf <- workflow() %>%
    add_recipe(m_recipe) %>%
    add_model(mb_model)

# create grid
mb_tree_grid <- grid_regular(cost_complexity(),
                           tree_depth(),
```

```
                          levels = 5)

# does the tuning (takes some time to run)
mb_tree_res <- mb_wf %>%
    tune_grid(
        resamples = m_samples,
        grid = mb_tree_grid
    )

# find the best collection of cost_complexity and tree_depth
mb_best_tree <- mb_tree_res %>%
    select_best("accuracy")

# finalize the workflow
mb_final_wf <- mb_wf %>%
    finalize_workflow(mb_best_tree)

# metrics of the tuned model
mb_final_wf %>%
    last_fit(mortgage_split) %>%
    collect_metrics()
```

```
# A tibble: 2 x 4
  .metric  .estimator .estimate .config
  <chr>    <chr>          <dbl> <chr>
1 accuracy binary         0.917 Preprocessor1_Model1
2 roc_auc  binary         0.914 Preprocessor1_Model1
```

```
# predictions of tuned model
mb_pred <- mb_final_wf %>%
    last_fit(mortgage_split) %>%
    collect_predictions()

mb_pred %>% metrics(rate, .pred_class)
```

```
# A tibble: 2 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.833
2 kap      binary         0.657
```

For bagging, I decided to tune cost complexity and tree_depth as to try and keep the tuning parameters similar to the decision tree. Bagging was close to what I predicted. For classification, it was getting about 83 % accurate compared to the decision tree which was about 75 % accurate. In the cases I tested, it seemed to get about one additional prediction correct compared the decision tree. Similarly, it was getting slightly smaller RMSE. About a 1000 less than the decision tree methods. So overall, bagging based on my data appears to be slightly more accurate than the decision tree.

```
## Random Forests Prof
########################################

# model
pf_model <- rand_forest(mode = "regression",
                        min_n = tune(),
                        trees = tune()) %>%
    set_engine("ranger")

# workflow
pf_wf <- workflow() %>%
    add_model(pf_model) %>%
    add_recipe(p_recipe)

# create grid
pf_grid <- grid_regular(min_n(),
                        trees(),
                        levels = 5)

# does the tuning (takes some time to run)
pf_tree_res <- pf_wf %>%
    tune_grid(
        resamples = p_samples,
        grid = pf_grid
    )

# find the best collection of cost_complexity and tree_depth
pf_best_tree <- pf_tree_res %>%
    select_best("rmse")

# finalize the workflow
pf_final_wf <- pf_wf %>%
    finalize_workflow(pf_best_tree)
```

```r
# metrics of the tuned model
pf_final_wf %>%
    last_fit(prof_split) %>%
    collect_metrics()
```

```
# A tibble: 2 x 4
  .metric .estimator .estimate .config
  <chr>   <chr>          <dbl> <chr>
1 rmse    standard   22935.    Preprocessor1_Model1
2 rsq     standard       0.555 Preprocessor1_Model1
```

```r
# predictions of tuned model
pf_pred <- pf_final_wf %>%
    last_fit(prof_split) %>%
    collect_predictions()

# metrics
pf_pred %>% metrics(salary, .pred)
```

```
# A tibble: 3 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard   22981.
2 rsq     standard       0.551
3 mae     standard   15968.
```

```r
## Random Forests mortgage
########################################

# model
mf_model <- rand_forest(mode = "classification",
                        min_n = tune(),
                        trees = tune()) %>%
    set_engine("ranger")

# workflow
mf_wf <- workflow() %>%
    add_model(mf_model) %>%
    add_recipe(m_recipe)
```

```
# create grid
mf_tree_grid <- grid_regular(min_n(),
                             trees(),
                             levels = 5)

# does the tuning (takes some time to run)
mf_tree_res <- mf_wf %>%
    tune_grid(
        resamples = m_samples,
        grid = mf_tree_grid
    )

# find the best collection of cost_complexity and tree_depth
mf_best_tree <- mf_tree_res %>%
    select_best("accuracy")

# finalize the workflow
mf_final_wf <- mf_wf %>%
    finalize_workflow(mf_best_tree)

# metrics of the tuned model
mf_final_wf %>%
    last_fit(mortgage_split) %>%
    collect_metrics()
```

```
# A tibble: 2 x 4
  .metric  .estimator .estimate .config
  <chr>    <chr>          <dbl> <chr>
1 accuracy binary         0.75  Preprocessor1_Model1
2 roc_auc  binary         0.914 Preprocessor1_Model1
```

```
# predictions of tuned model
mf_pred <- mf_final_wf %>%
    last_fit(mortgage_split) %>%
    collect_predictions()

mf_pred %>% metrics(rate, .pred_class)
```

```
# A tibble: 2 x 3
```

```
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.75
2 kap      binary         0.471
```

For random forests, I decided to tune min_n and trees because I was unable to tune the other attributes like I was hoping, but these two still worked. The data appears to have turned out as expected. Sometimes random forest was slightly worse compared to bagging for classification and RMSE, while other times it was slightly better. Like bagging, the random forest models did a bit better compared to decision trees. So relative to decision trees, I would say the random forest model was overall more accurate while also being relatively similar to bagging.

```
## Boosting Trees
########################################

prof_salary_boost <- prof_salary %>%
  mutate(rank = as.numeric(as.factor(rank)),
                                          discipline = as.numeric(as.factor(discipline)),
                                          sex = as.numeric(as.factor(sex)))
p_split_boost <- initial_split(prof_salary_boost,
                               prop = 0.9)
p_training_boost <- training(p_split_boost)
p_testing_boost <- testing(p_split_boost)

p_boost_samples <- vfold_cv(p_training_boost)

# model
p_boost_model <- boost_tree(mode = "regression",
                            min_n = tune(),
                            tree_depth = tune(),
                            trees = 100) %>%
    set_engine("xgboost")

# data
p_boost_recipe <- recipe(salary ~ ., p_training_boost)

# workflow
p_boost_wf <- workflow() %>%
    add_model(p_boost_model) %>%
    add_recipe(p_boost_recipe)
```

```
# create grid
p_boost_grid <- grid_regular(min_n(),
                         tree_depth(),
                         levels = 5)

# does the tuning (takes some time to run)
p_boost_tree_res <- p_boost_wf %>%
    tune_grid(
        resamples = p_boost_samples,
        grid = p_boost_grid
    )

# find the best collection of cost_complexity and tree_depth
p_boost_best_tree <- p_boost_tree_res %>%
    select_best("rmse")

# finalize the workflow
p_boost_final_wf <- p_boost_wf %>%
    finalize_workflow(p_boost_best_tree)

# metrics of the tuned model
p_boost_final_wf %>%
    last_fit(p_split_boost) %>%
    collect_metrics()
```

```
# A tibble: 2 x 4
  .metric .estimator .estimate .config
  <chr>   <chr>          <dbl> <chr>
1 rmse    standard   17523.    Preprocessor1_Model1
2 rsq     standard       0.589 Preprocessor1_Model1
```

```
# predictions of tuned model
p_pred_boost <- p_boost_final_wf %>%
    last_fit(p_split_boost) %>%
    collect_predictions()

# metrics
p_pred_boost  %>% metrics(salary, .pred)
```

```
# A tibble: 3 x 3
```

```
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard   17523.
2 rsq     standard       0.589
3 mae     standard   13212.
```

```
## Boosting Trees
########################################
mortgage_rate_boost <- mortgages %>%
  mutate(married = as.numeric(as.factor(married)),
         first = as.numeric(as.factor(first)),
         selfemp = as.numeric(as.factor(selfemp)),
         coborrower = as.numeric(as.factor(coborrower)))
m_split_boost <- initial_split(mortgage_rate_boost,
                               prop = 0.85)
m_training_boost <- training(m_split_boost)
m_testing_boost <- testing(m_split_boost)

m_boost_samples <- vfold_cv(m_training_boost)

# model
m_boost_model <- boost_tree(mode = "classification",
                            min_n = tune(),
                            tree_depth = tune(),
                            trees = 100) %>%
    set_engine("xgboost")

m_boost_recipe <- recipe(rate ~ ., m_training_boost)

# workflow
m_boost_wf <- workflow() %>%
    add_model(m_boost_model) %>%
    add_recipe(m_boost_recipe)

# create grid
m_boost_grid <- grid_regular(min_n(),
                             tree_depth(),
                             levels = 5)

# does the tuning (takes some time to run)
m_boost_tree_res <- m_boost_wf %>%
    tune_grid(
```

```
        resamples = m_boost_samples,
        grid = m_boost_grid
    )

# find the best collection of cost_complexity and tree_depth
m_boost_best_tree <- m_boost_tree_res %>%
    select_best("accuracy")

# finalize the workflow
m_boost_final_wf <- m_boost_wf %>%
    finalize_workflow(m_boost_best_tree)

# metrics of the tuned model
m_boost_final_wf %>%
    last_fit(m_split_boost) %>%
    collect_metrics()
```

```
# A tibble: 2 x 4
  .metric  .estimator .estimate .config
  <chr>    <chr>          <dbl> <chr>
1 accuracy binary         0.833 Preprocessor1_Model1
2 roc_auc  binary         0.917 Preprocessor1_Model1
```

```
# predictions of tuned model
m_pred_boost <- m_boost_final_wf %>%
    last_fit(m_split_boost) %>%
    collect_predictions()

# metrics
m_pred_boost  %>% metrics(rate, .pred_class)
```

```
# A tibble: 2 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.833
2 kap      binary         0.667
```

For boosted trees I decided to tune min_n and tree_depth. As was expected, boosted trees consistently performed better than all three other types of tree based methods. In fact, it

was the only one to have an RMSE below 20000 in the experiments I ran. It also was just as accurate or more accurate than the others for classification. It would usually get an additional prediction correct compared to bagging and random forests while getting 2 more correct than decision trees. This provides evidence to my hypothesis that it was consistently the most accurate model of the four.

Below is all the various accuracy measures concentrated in one place. They are not necessarily the exact values I found in my experiments.

```
# Comparisions

# Mortgage
(m_pred_decision %>% metrics(rate, full))[1,] # Full Base R Decision Tree
```

```
# A tibble: 1 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary          0.75
```

```
(m_pred_decision %>% metrics(rate, pred))[1,] # Prune Base R Decision Tree
```

```
# A tibble: 1 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary          0.75
```

```
(md_pred2 %>% metrics(rate, .pred_class))[1,] # Tidy Decision Tree
```

```
# A tibble: 1 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary          0.75
```

```
(mb_pred %>% metrics(rate, .pred_class))[1,] # Bagging
```

```
# A tibble: 1 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.833
```

```
(mf_pred %>% metrics(rate, .pred_class))[1,] # Forest
```

```
# A tibble: 1 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary          0.75
```

```
(m_pred_boost %>% metrics(rate, .pred_class))[1,] # Boost
```

```
# A tibble: 1 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.833
```

```
# Professor
(p_pred_decision %>% metrics(salary, full))[1,] # Full Base R Decision Tree
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard      24326.
```

```
(p_pred_decision %>% metrics(salary, pred))[1,] # Prune Base R Decision Tree
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard      25876.
```

```
(pd_pred2 %>% metrics(salary, .pred))[1,] # Tidy Decision Tree
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard      24888.
```

```
(pb_pred %>% metrics(salary, .pred))[1,] # Bagging
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard      23273.
```

```
(pf_pred %>% metrics(salary, .pred))[1,] # Forest
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard      22981.
```

```
(p_pred_boost %>% metrics(salary, .pred))[1,] # Boost
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard      17523.
```

- Give your final thoughts on how the methods seem to handle the different datasets.

Overall, this homework seemed to go well and my hypothesizes about the different tree based methods seemed to have some supporting evidence. Tuning appeared to improve some of the models. I was a bit surprised the tidyverse version of decision trees was slightly worse especially after being tuned a bit more than the base R one. As with any experiment, variability in dividing the data played a roll. It is important to note that sometimes even the base R model would be 100% accurate meaning that it had the potential to be just as accurate with the

other models. Sometimes it would be 50% which really isn't any better than guessing. With that being said, I would say that bagging and random forests were generally more accurate followed by boosted trees being the most accurate. I would also like to point out that the models were fairly close in regression and classification. This means if a model was good at classification it was also good at regression and vice-versa. They both were fairly close not straying too far from one another.