Building an NBA All-Pro Team Prediction Model

Using Machine Learning Models to Predict What NBA Players Make the All-Pro Teams using Advanced NBA Statistics

Jimmy Dysart

2023-03-16

Introduction

The NBA's All-Pro (or All-NBA) teams are end-of-season awards given to the NBA's top 15 players. There are three tiers to this award: "First team", "Second team", and "Third team". For the purposes of this analysis, all of teams will mush into one category: Did you make one of the NBA's All-Pro teams or not. The All-Pro award is a regular season award that looks how well players played in the regular season. The voting is done by NBA broadcasters and sportswriters, also known as the NBA media.

The players with the highest point totals at their respective positions make the first team, with the next highest making the second team and so forth.

Interestingly, All-Pro teams are announced during the playoffs. This means that if I can get this model to work before this years' 2023 NBA playoffs and after the end of the regular season, I can use this years' 2023 NBA regular season data to predict the All-NBA teams(and possibly bet on it!).

I am interested in predicting if an NBA player will make an all-pro team.

The response(or outcome) variable is a factor that I will manually add in if they made "all-pro" team for that specific year. The remaining predictor variables will be the different RAPTOR advanced statistics. I will use minutes played and possessions as a cut off point to remove extremely influential outliers.

I believe that these questions will be best answered by a `classification` approach because I want to look at combination of predictor variables and how well they can predict whether or not a player will make the all-pro team. Classification is best here because the outcome variable is in the format of `Yes` or `No`. ' This will be a predictive model that will look to see if I can predict who this current seasons "all-pro" team will be with a full season of data.

Before we can get to building this model, we must look at the `Modern Raptor` data and make sure it is in the correct formatting that we want it in!

Exploratory Data Analysis

It is important to look at the data before building a prediction model because data often needs to be `cleaned`. In addition, it is important to look at the data and understand it thoroughly before building a model on it.

Let's look at the data!

Overview of Dataset

```
ModernRaptor %>% nrow()
```

```
## [1] 4685
```

```
ModernRaptor %>% ncol()
```

```
## [1] 21
```

```
ModernRaptor %>% as_tibble()
```

```
## # A tibble: 4,685 x 21
##    player_~1 playe~2 season  poss    mp raptor~3 rapto~4 rapto~5 rapto~6 rapto~7
##    <chr>     <chr>    <int> <int> <int>    <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
##  1 Alex Abr~ abrina~   2017  2387  1135  0.746    -0.373   0.373 -0.419  -3.86
##  2 Alex Abr~ abrina~   2018  2546  1244  0.318    -1.73   -1.41  -1.29   -0.0497
##  3 Alex Abr~ abrina~   2019  1279   588 -3.22      1.08   -2.14  -6.16    4.90
##  4 Precious~ achiup~   2021  1581   749 -4.12      1.36   -2.76  -4.05   -0.920
##  5 Precious~ achiup~   2022  3802  1892 -2.52      1.76   -0.758 -1.69    3.10
##  6 Quincy A~ acyqu01   2014  1716   847 -1.72      0.133  -1.58  -0.325  -1.66
##  7 Quincy A~ acyqu01   2015  2517  1287 -2.01     -1.27   -3.28  -3.86    2.80
##  8 Quincy A~ acyqu01   2016  1852   876 -0.00833   0.341   0.332 -2.80    0.130
##  9 Quincy A~ acyqu01   2017  1169   558 -0.129     0.444   0.315 -1.35    0.807
## 10 Quincy A~ acyqu01   2018  2856  1359 -2.62     -0.806  -3.43   0.0552 -0.0192
## # ... with 4,675 more rows, 11 more variables: raptor_onoff_total <dbl>,
## #   raptor_offense <dbl>, raptor_defense <dbl>, raptor_total <dbl>,
## #   war_total <dbl>, war_reg_season <dbl>, war_playoffs <dbl>,
## #   predator_offense <dbl>, predator_defense <dbl>, predator_total <dbl>,
## #   pace_impact <dbl>, and abbreviated variable names 1: player_name,
## #   2: player_id, 3: raptor_box_offense, 4: raptor_box_defense,
## #   5: raptor_box_total, 6: raptor_onoff_offense, 7: raptor_onoff_defense
```

This data set contains "contains `RAPTOR` data for every player broken out by season since `2014`, when NBA player-tracking data first became available."

The link to the `data set` is: https://github.com/fivethirtyeight/data/blob/5b8f3fe844b9559e398ae7178d20b58c4dc53fc7/nba-raptor/historical_RAPTOR_by_player.csv

The link to the `codebook` is: https://github.com/fivethirtyeight/data/blob/5b8f3fe844b9559e398ae7178d20b58c4dc53fc7/nba-raptor/README.md

Raptor data is a new advanced statistical metric to quantify the play of NBA players.

There are `4685 observations` with `21 predictors`.

There are two character predictors, one which represents the players name and the other which represents the players unique id:

- `player_name` -> Player Name

- `player_id` -> Basketball-Reference.com player ID

There are three integer values which are the season that the statistics are from, the # of possessions played, and the # of minutes played:

- `season` -> Year # of NBA Season

- `poss` -> Possessions Played

- `mp` -> Minutes Played

The rest of the variables are numeric decimal values which will be the different predictors in this data set.

There are four types of numeric decimal values in this data set:

- **Raptor**:

- `raptor_box_offense` -> Points above average per 100 possessions added by player on offense, based only on box score estimate

- `raptor_box_defense` -> Points above average per 100 possessions added by player on defense, based only on box score estimate

- `raptor_box_total` -> Points above average per 100 possessions added by player, based only on box score estimate

- `raptor_onoff_offense` -> Points above average per 100 possessions added by player on offense, based only on plus-minus data

- `raptor_onoff_defense` -> Points above average per 100 possessions added by player on defense, based only on plus-minus data

- `raptor_onoff_total` -> Points above average per 100 possessions added by player, based only on plus-minus data

- `raptor_offense` -> Points above average per 100 possessions added by player on offense, using both box and on-off components

- `raptor_defense` -> Points above average per 100 possessions added by player on defense, using both box and on-off components

- `raptor_total` -> Points above average per 100 possessions added by player on both offense and defense, using both box and on-off components

- **War**:

- `war_total` -> Wins Above Replacement between regular season and playoffs

- `war_reg_season` -> Wins Above Replacement for regular season

- `war_playoffs` -> Wins Above Replacement for playoffs

- **Predator**:

- `predator_offense` -> Predictive points above average per 100 possessions added by player on offense

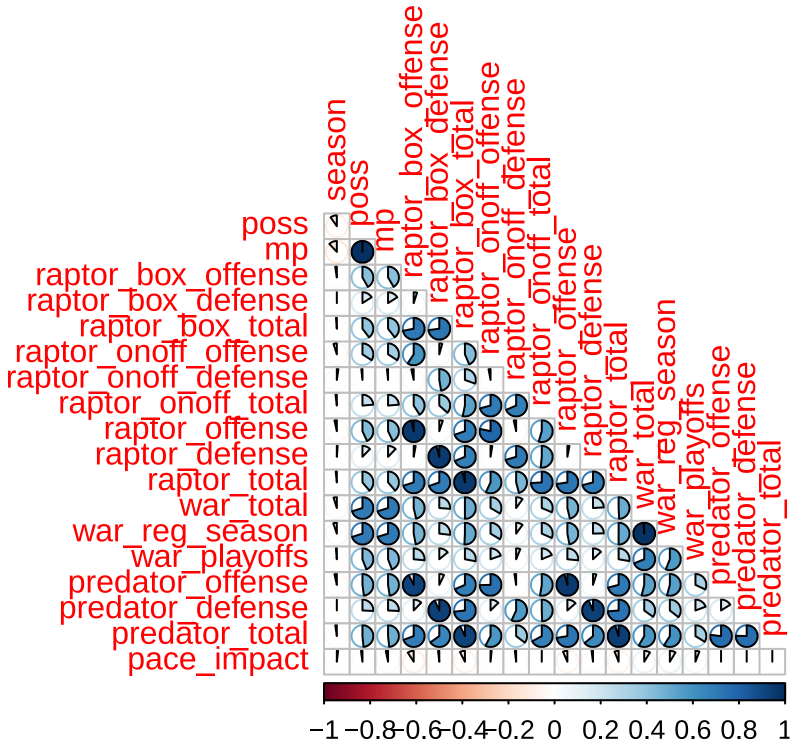
- `predator_defense` -> Predictive points above average per 100 possessions added by player on defense

- `predator_total` -> Predictive points above average per 100 possessions added by player on both offense and defense

- **Pace**:

- `pace_impact` -> Player impact on team possessions per 48 minutes

There is a missing column data, the outcome variable for the model. I am going to add a binary classification value for if the player made an `allpro` team that year. This will allow me to draw conclusions and make predictions based on how the different variables effect the response variable.



From the corrplot, we can tell that there is a strong correlation between these predictor variables:

- Poss & MP

- raptor_offense & raptor_box_offense

- raptor_total & raptor_box_total

- predator_total & raptor_box_total

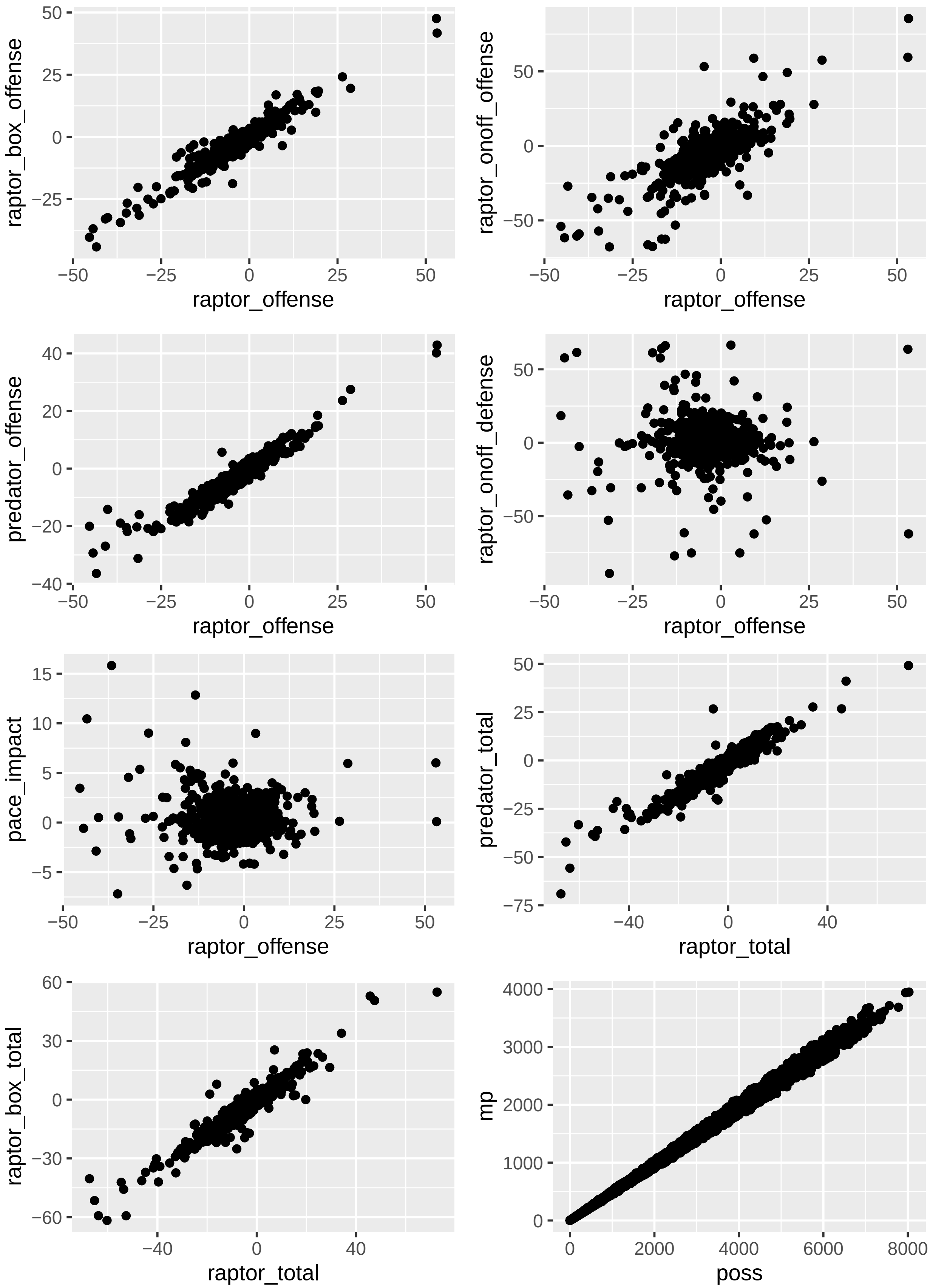
- war_reg_season & war_total

and many more!

Interestingly, there is almost no red on the corrplot. This means that none of the predictor variables have negative correlations with another predictor variable. (Very slight red correlation between `season` and other predictor variables)

A high correlation between two predictor variables is a sign of `collinearity`.

Let's look more into that:


raptor_box_offense
50
25
0
−25
−50
−25
0
25
50
raptor_offense
raptor_onoff_offense
50
0
−50
−50
−25
0
25
50
raptor_offense
predator_offense
40
20
0
−20
−40
−50
−25
0
25
50
raptor_offense
raptor_onoff_defense
50
0
−50
−50
−25
0
25
50
raptor_offense
pace_impact
15
10
5
0
−5
−50
−25
0
25
50
raptor_offense
predator_total
50
25
0
−25
−50
−75
−40
0
40
raptor_total
raptor_box_total
60
30
0
−30
−60
−40
0
40
raptor_total
mp
4000
3000
2000
1000
0
0
2000
4000
6000
8000
poss

There is a high correlation between `raptor_total` , `raptor_box_total`, and `predator_total`. This is also the case for the offense versions and defense versions of these variables. This will be a problem later down the line when build the models because of collinearity. Collinearity is a strong correlation between predictor variables. To adjust for collinearity, let's remove all of the `predator..` and `raptor_box..` data as predictors because it is repetitive and has a high correlation with the raptor data. We will do this step later when we build our recipe and models.

Later in the report I will do another corrplot when we add the outcome variable `allpro` into the data frame. This will give us a surface-level idea of the correlation between predictor variables and the outcome variable.

Tidying Data

To properly build a model which accurately predicts who will make the all-pro team, it is important to filter out data that could be influential points on the model. This means filtering out data of guys who had no chance of making all-NBA teams based purely on total minutes played.

The lowest minutes played for an NBA player to make the all-NBA team was LeBron James in 2021, with 1728 minutes.

I felt like a `cutoff of 1500 minutes` was proper to ensure that no players who had minutes significantly below that were considered, as they wouldn't even be eligible for voting consideration for an all-NBA team.

```
ModernRaptorOver1500Min %>% nrow() # Find number of rows after data cleaning
```

```
## [1] 1729
```

It is important to note that NBA all-pro teams are decided based on ONLY regular season play. This means that variables like `war_total` and `war_playoffs`, which tell us playoff statistics, are not important in predicting whether or not a player will make an NBA all-pro team. Let's remove those two predictor variables.

```
ModernRaptor_noWARPlayoffs <- ModernRaptorOver1500Min %>% select(-c(war_total, war_playoffs))

ModernRaptor_noWARPlayoffs %>% ncol()
```

```
## [1] 19
```

While we have a bunch of great predictor variables for this model, we don't have an outcome variable. Our model will be a classification model because our outcome variable will be boolean. Boolean means that it will be a column consisting of just `0` or `1`, where `0` means the player DID NOT make any all-pro team that year, and `1` means the player DID make an all-pro team that year. To create this outcome variable we need to use the function add_column.

Then we need to manually input `1` for the rows where the year and player name match the names and years of players who made the all-pro team.To manually input the `1` value, we will create a new data frame with just the keys `player_name`, `season`, and the boolean variable `allpro`. We will make a csv file for this data frame and export it to Excel where I can manually input the `1` value for each player that made the allpro teams from 2014-2022.

Here is the website I used to figure out what players made the all-pro teams for what seasons:

https://www.nba.com/news/history-all-nba-teams

```r
ModernRaptorwithOutcomeVar<-ModernRaptor_noWARPlayoffs %>% add_column(allpro = 0)

SavedCSV<- ModernRaptorwithOutcomeVar %>% select(player_name,season,allpro)

SavedCSV1<-SavedCSV[order(-SavedCSV$season),]
  #SavedCSV %>% order(season, decreasing = TRUE) # merge

write_csv(SavedCSV1, file="/Users/jimmydysart/Documents/PSTAT\ 131/Final")
```

After doing work in Excel, I will now import the CSV file back into R.

```r
library(readxl)
Book1 <- read_excel("Book11.xlsx")
```

This file has the 1 values where a player's name and season # is the same as the year they made an all-pro team.

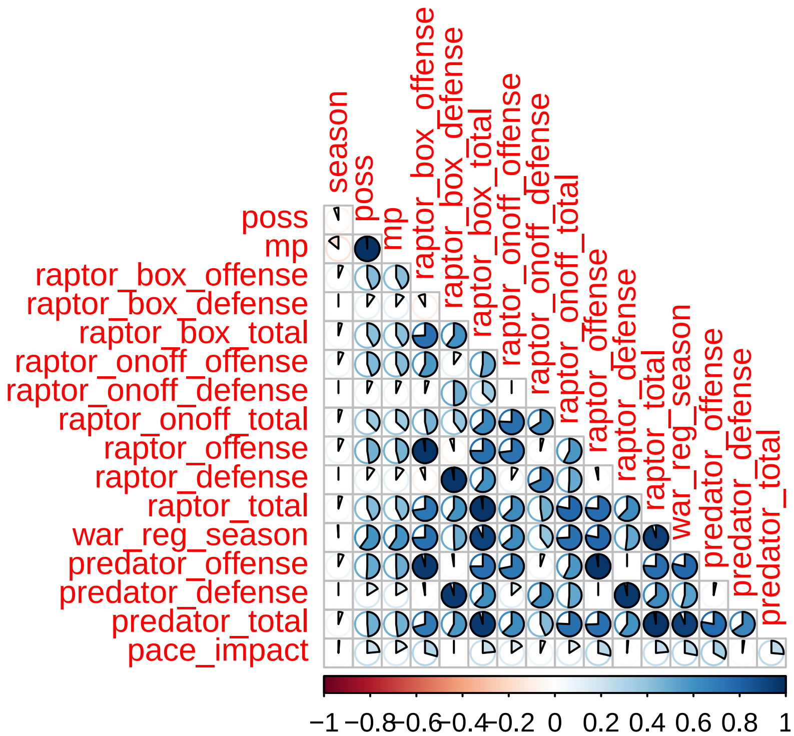
Now, I need to merge the csv file with the existing data frame to add the boolean column to the data set.

```r
str(OfficialModernRaptorData)
```

```
## 'data.frame':    1729 obs. of  20 variables:
##  $ player_name         : chr  "Aaron Brooks" "Aaron Brooks" "Aaron Gordon" "Aaron Gordon" ...
##  $ season              : int  2014 2015 2016 2017 2018 2019 2020 2021 2022 2020 ...
##  $ player_id           : chr  "brookaa01" "brookaa01" "gordoaa01" "gordoaa01" ...
##  $ poss                : int  3238 3956 3823 4687 3994 5774 4190 3489 5225 3531 ...
##  $ mp                  : int  1557 2017 1863 2298 1909 2797 2017 1683 2536 1689 ...
##  $ raptor_box_offense  : num  0.974 1.086 1.216 -0.237 -0.323 ...
##  $ raptor_box_defense  : num  -2.1277 -3.4856 0.0288 -0.0824 1.6271 ...
##  $ raptor_box_total    : num  -1.154 -2.399 1.245 -0.319 1.304 ...
##  $ raptor_onoff_offense: num  -1.612 1.054 -0.741 6.142 1.714 ...
##  $ raptor_onoff_defense: num  0.1454 -0.4272 0.0387 -1.1385 0.7485 ...
##  $ raptor_onoff_total  : num  -1.467 0.627 -0.702 5.003 2.463 ...
##  $ raptor_offense      : num  0.476 1.145 0.894 1.139 0.134 ...
##  $ raptor_defense      : num  -1.7759 -3.0764 0.0458 -0.2784 1.5721 ...
##  $ raptor_total        : num  -1.3 -1.931 0.94 0.861 1.706 ...
##  $ war_reg_season      : num  1.16 1.45 3.51 4.23 4.35 ...
##  $ predator_offense    : num  0.592 1.256 0.756 0.664 0.343 ...
##  $ predator_defense    : num  -1.317 -2.203 -0.213 -0.695 1.235 ...
##  $ predator_total      : num  -0.725 -0.9469 0.5436 -0.0314 1.5781 ...
##  $ pace_impact         : num  0.4026 -0.4393 -0.0626 -0.1146 0.3273 ...
##  $ allpro              : num  0 0 0 0 0 0 0 0 0 0 ...
```

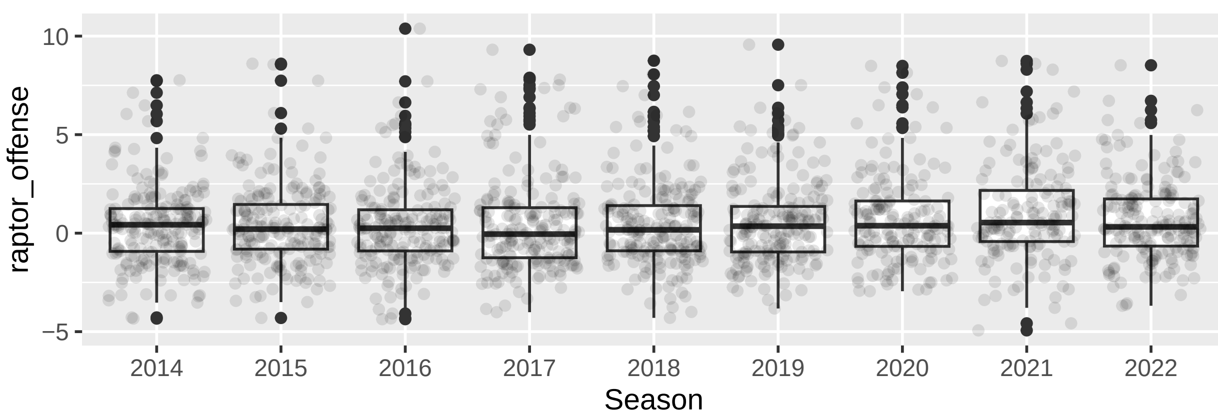
```r
OfficialModernRaptorData$allpro <- as.factor(OfficialModernRaptorData$allpro) # convert allpro into a f
```

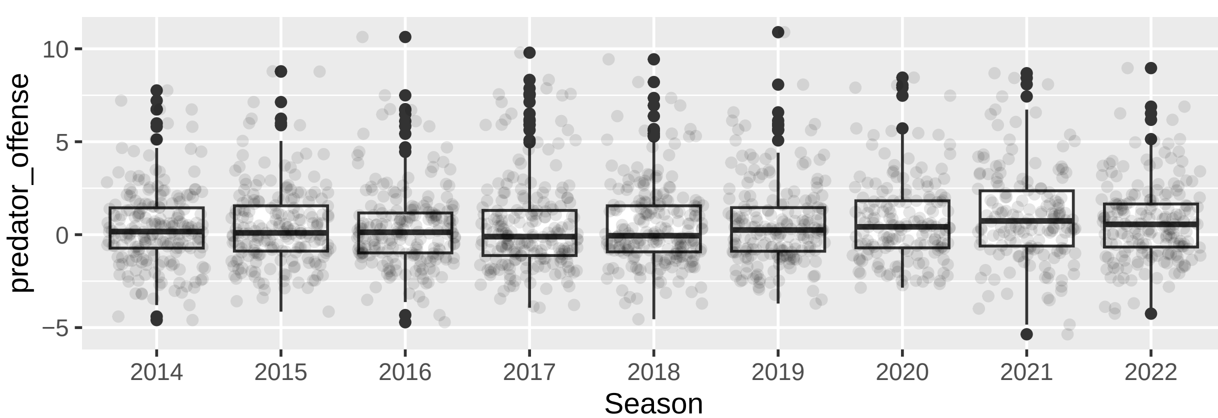
Let's go! I have oficially merged and cleaned the data! Now lets quickly look at another corrplot of this new data.

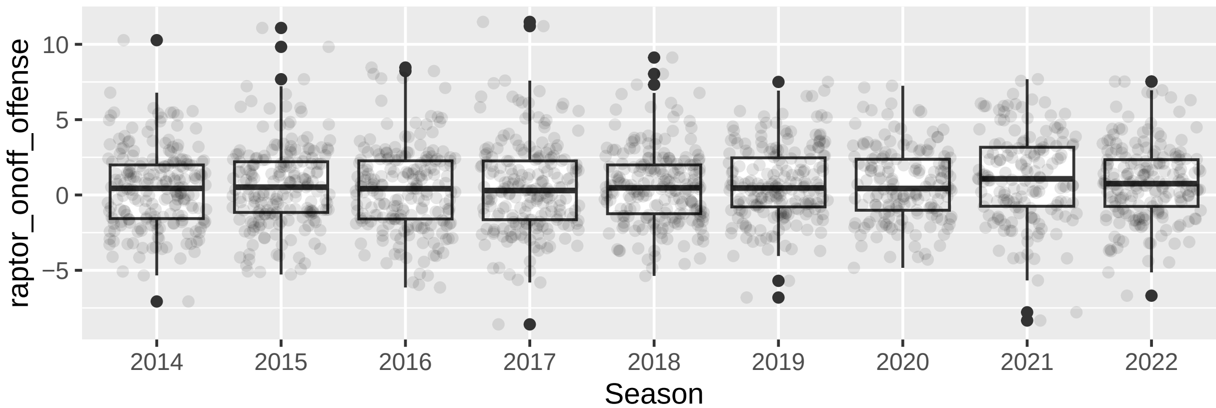


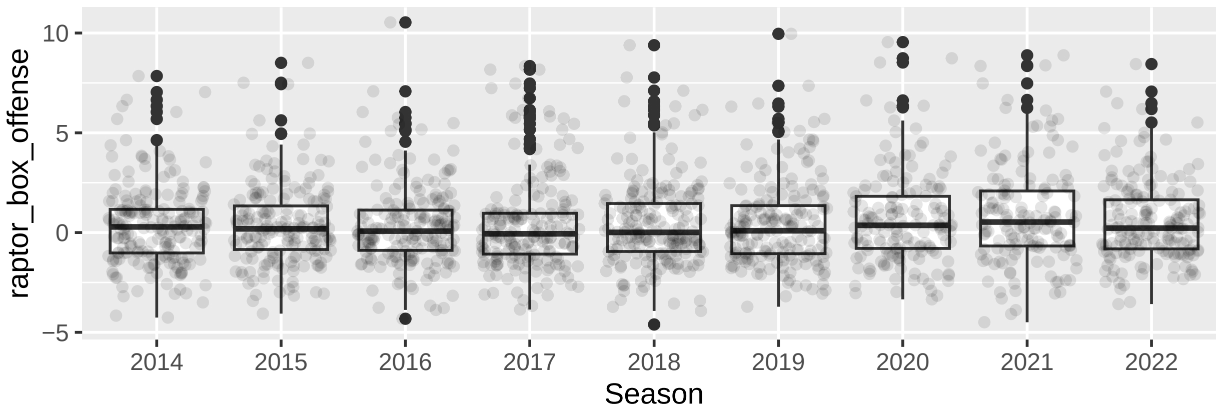
With this new plot,I am worried about multi-collinearity.

All of the variables that have `offense` in their name are very correlated with eachother. This is the same with `defense`. Let's look at a plot of all of the offense statistics over the seasons.









Looks like `raptor_offense` is roughly the same throughout the different seasons.

Looks like `predator_offense` is roughly the same throughout the different seasons. If anything, a slight increase as the seasons go on.

Looks like`raptor_onoff_offense` is roughly the same throughout the different seasons. If anything, a slight increase as the seasons go on.

I can tell now that most of the predictor variables are very correlated with eachother. This means we need to remove some of the predictor variables from the recipe in order not to deal with multicollinearity.

I have decided to go with variables:

- `mp`
- `raptor_onoff_offense`
- `raptor_onoff_defense`
- `raptor_onoff_total`
- `raptor_offense`
- `raptor_defense`
- `raptor_total`
- `war_reg_season`
- `pace_impact`

Now, we are ready to take a deeper dive and start building a model!

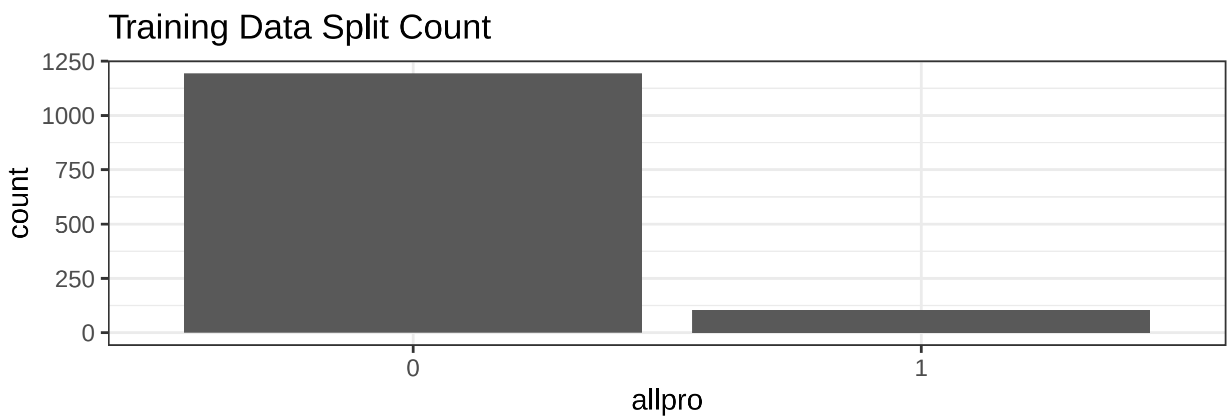
Data Splitting and Cross-Validation

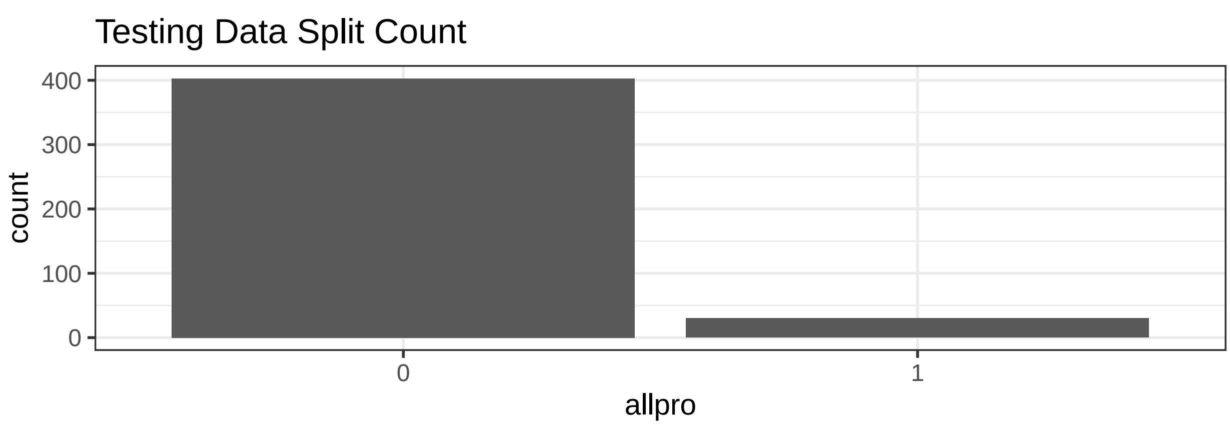
To build a machine learning model, we need to have a model, a data set for the model to `train` on, and a data set for the model to `test` on.

To begin, let's build a training and testing data set. We can do this by splitting up the `OfficialModernRaptor` Data set into two.In addition, lets make a `V-fold cross-validation` to randomly split the training data into `10 equally sized folds`. This will help us test the model even better on the data and help with building some of the models.

```
set.seed(3435) # randomize seed
Modernraptor_split<- initial_split(OfficialModernRaptorData, prop = .75, strata = allpro) #75 percent o


Modernraptor_train <- training(Modernraptor_split)
Modernraptor_test <- testing(Modernraptor_split)

Modernraptor_fold <- vfold_cv(Modernraptor_train, v = 10, strata = allpro) # this will be useful for th
```

I wanted to do a little data exploration on the `Modernraptor_train` and `Modernraptor_test` data subsets.






```
## # A tibble: 2 x 2
##   allpro   prop
##   <fct>   <dbl>
## 1 0      0.920
## 2 1      0.0802

## # A tibble: 2 x 2
##   allpro   prop
##   <fct>   <dbl>
## 1 0      0.931
## 2 1      0.0693
```

The first tibble is for the training data, the second tibble is for the testing data.

```
## [1] 1296   20
```

```
## [1] 433  20
```

The results from the data exploration is that `92% of the training data is not an allpro` that season, and `93% of the testing data is not an allpro` that season. There is a total of `1296 data points` in the `training` data and `433 data points` in the `testing` data.

We have successfully split up our data into a training set and a testing set.

The Recipe

The next step in setting up the model is creating a recipe. A recipe helps us get the data ready to be modeled. The same recipe can be used for each model because each model will be using at the same data set. A recipe is nice because it can be used for any of the `4 models` that we build. I'll talk about the 4 types of models that we will be using later.

```
original_recipe <- recipe(allpro ~ mp + raptor_onoff_offense+raptor_onoff_defense+raptor_onoff_total +ra

# this step accounts for an imbalance in the ratio of yes' to no's in the data


original_recipe
```

```
## Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor          9
##
## Operations:
##
## Dummy variables from all_nominal_predictors()
## Up-sampling based on allpro
## Centering and scaling for all_predictors()
```

There are `9 predictor variables` for the outcome variable `allpro` that we will be looking at. I took out `player_name`, `season` and `player_id` as predictor variables because they are irrelevant in quantitatively predicting the `allpro` outcome variable. As mentioned above, I took out `predator`, `raptor_box`, and `poss` data because of collinearity issues.

I added a couple steps into the recipe. The first step is making all categorical predictor variables into factors, so they are easy to quantify in the model. The second and third steps are to normalize the variables so the magnitude of one of the variables doesn't have an overwhelming impact on the model results.

The first of the 4 models is a logistic regression model! Let's build it.

Model Fitting

For model fitting, we will be using the recipe we just made to test four different machine learning models. The first three will be `Logistic Regression`, `LDA`, and `KNN`. The fourth one will be a regularized regression model which will be `Elastic Net Logistic Regression`. For all of these models, we will be looking at what `accuracy` output we get from the model on the training set of the data.

Logistic Regression

For each model, we need to specify which type of model we are using and set up a workflow.

```
log_reg <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification") # specify which type of model we are using
```

```
log_wkflow <- workflow() %>% add_model(log_reg) %>% add_recipe(original_recipe)
```

I am fitting the workflow I made for the logistic regression with the training data set.

```
log_fit <- fit(log_wkflow,Modernraptor_train)
log_fit %>% tidy() # view results
```

```
## # A tibble: 10 x 5
##    term                 estimate std.error statistic  p.value
##    <chr>                   <dbl>     <dbl>     <dbl>    <dbl>
##  1 (Intercept)          -1.90e+0   1.32e-1   -14.3   1.48e-46
##  2 mp                    6.18e-1   1.77e-1     3.49  4.81e- 4
##  3 raptor_onoff_offense -3.55e+9   7.94e+8    -4.47  7.71e- 6
##  4 raptor_onoff_defense -2.74e+9   6.13e+8    -4.47  7.71e- 6
##  5 raptor_onoff_total    4.67e+9   1.04e+9     4.47  7.71e- 6
##  6 raptor_offense        9.17e+8   7.29e+8     1.26  2.08e- 1
##  7 raptor_defense        6.35e+8   5.05e+8     1.26  2.08e- 1
##  8 raptor_total         -1.14e+9   9.05e+8    -1.26  2.08e- 1
##  9 war_reg_season        2.61e+0   5.01e-1     5.22  1.77e- 7
## 10 pace_impact           7.27e-2   1.04e-1     0.699 4.85e- 1
```

Linear Discriminant Analysis (LDA)

The second model that we will be fitting is linear discriminant analysis (LDA) model! LDA assumes normal distribution of predictors given the class of the `allpro` variable. LDA is a parametic model with a low model flexiblity. LDA assumes that the outcome variable has a different mean but the same covariance. This is pretty much creating a linear boundry line between the yes to `allpro` variable and no to `allpro` variable.

```
lda_mod <- discrim_linear()%>% set_engine("MASS") %>% set_mode("classification")

lda_wkflow <- workflow() %>% add_model(lda_mod) %>% add_recipe(original_recipe)

lda_fit <- fit(lda_wkflow, Modernraptor_train)
```

Hyperparameter Tuning Models

Hyperparameter tuning means that the training of the model is inputting in different a bunch of different values for the selected hyperparameter. Each model has it's own unique hyperparameters. Each model is trying to find the best values for the hyperparamter. The `KNN model uses 1 hyperparameter` and the `Elastic Net Logistic Regression uses 2 hyperparameters`.

These models will use the v-fold cross validation feature that we made earlier in the project.

Let's build those models.

K-Nearest Neighbors(KNN)

K nearest neighbors is a model where to predict the classification a new data point, it looks a the `k` nearest data points in the training data and sees what `true` classification value those data values have. It then classifies the new data point based on what the majority of the `k` nearest data points majorities are.

For example, let's say we set `k=5`, then we have a new data point and it's 5 nearest data points are `3 made NBA all-pro team` and `2 did not make NBA all-pro team`. Then, it would predict and classify that data point as a `made all-pro team`.

```
knn_mod <- nearest_neighbor(neighbors = tune()) %>%
  set_mode("classification") %>%
  set_engine("kknn")
```

Tune is used to tell the the cross validation process that mixture and penalty will be tested at different values for the model.

```
knn_wkflow <- workflow() %>% add_model(knn_mod) %>% add_recipe(original_recipe) # building workflow for
```

```
knn_grid <- grid_regular(neighbors(range = c(1,10)), levels = 10)
# this is the grid that the different tuning values will be tested in
```

```
knn_fit <- tune_grid(knn_wkflow, resamples = Modernraptor_fold, grid = knn_grid) # this is the fitting
```

Elastic Net Logistic Regression

Elastic Net Logistic Regression uses both mixture and penalty and different values to find the perfect fit between the Lasso Regression model and the Ridge Regression model.

```
en <- logistic_reg(mixture = tune(), penalty = tune()) %>% set_mode("classification") %>% set_engine("g
```

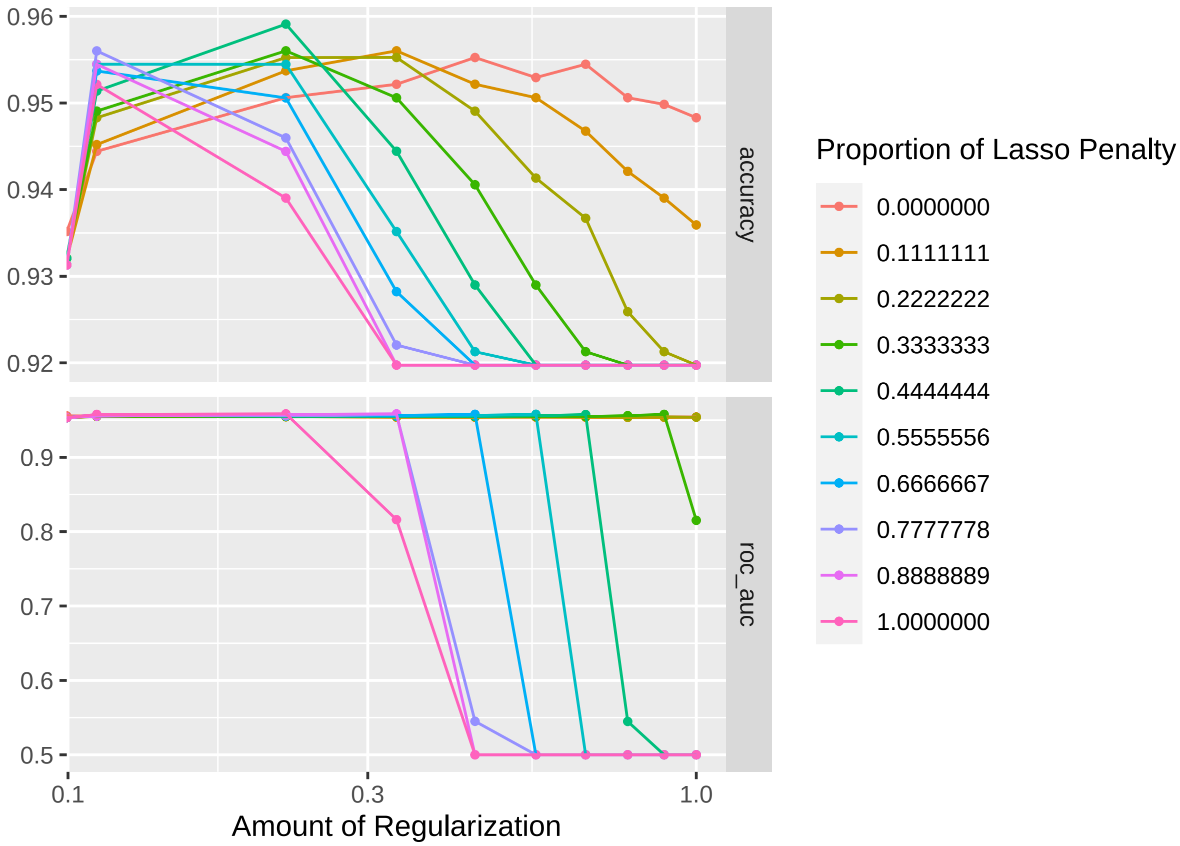
Tune is used to tell the the cross validation process that mixture and penalty will be tested at different values for the model.

```
#workflow for elastic net log regression

en_wrkflow <- workflow() %>% add_recipe(original_recipe) %>% add_model(en)
```

```
en_grid <- grid_regular(penalty(range = c(0,1), trans = identity_trans()), mixture(range = c(0,1)), leve

#grid that will host different tuning values for penalty and mixture
```

This is just building the grid that will be used by the model to test the two different hyperparameters (mixture and penalty).

```
tune_en <- tune_grid(en_wrkflow, resamples = Modernraptor_fold, grid = en_grid)
```



This `autoplot` shows us the model `accuracy` and `roc_auc` for each of the different elastic net tunes.

For our purposes, `roc_auc` is just another metric to visualize how well the model is performing at predicting the classification of the individual data points.

From the autoplot, we can tell that the model has the highest accuracy when the `lasso penalty is 0.333` and a `regularization of  0.25`.

Now let's analyze deeper into the performance of our models.

Model Selection and Performance

Logistic Regression

```
#check accuracy of the logistic regression model

log_accuracy <- augment(log_fit, new_data = Modernraptor_train) %>%
  accuracy(truth = allpro, estimate = .pred_class)

log_accuracy #0.931
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.931
```

LDA

```
lda_accuracy <- augment(lda_fit, new_data = Modernraptor_train) %>%
  accuracy(truth = allpro, estimate = .pred_class)

lda_accuracy #0.944
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.944
```

KNN

```
show_best(knn_fit, "accuracy")
```

```
## # A tibble: 5 x 7
##   neighbors .metric  .estimator  mean     n std_err .config
##       <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1         1 accuracy binary     0.921    10 0.00583 Preprocessor1_Model01
## 2         2 accuracy binary     0.921    10 0.00583 Preprocessor1_Model02
## 3         3 accuracy binary     0.921    10 0.00583 Preprocessor1_Model03
## 4         4 accuracy binary     0.921    10 0.00583 Preprocessor1_Model04
## 5         8 accuracy binary     0.915    10 0.00443 Preprocessor1_Model08
```

```
bestknn #neighbors = 1
```

```
## # A tibble: 1 x 2
##   neighbors .config
##       <int> <chr>
## 1         1 Preprocessor1_Model01
```

The best knn model has a neighbors value of 1, which means that it is only looking at the next closest data point in the set to assign itself to as a classification value.

The accuracy of the knn model is 0.921.

Elastic Net Logistic Regression

```
a<- collect_metrics(tune_en)

besten <- select_best(tune_en, metric = "accuracy")
show_best(tune_en, "accuracy")
```

```
## # A tibble: 5 x 8
##   penalty mixture .metric  .estimator  mean     n std_err .config
##     <dbl>   <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1   0.222   0.444 accuracy binary     0.959    10 0.00460 Preprocessor1_Model043
## 2   0.333   0.111 accuracy binary     0.956    10 0.00608 Preprocessor1_Model014
## 3   0.222   0.333 accuracy binary     0.956    10 0.00660 Preprocessor1_Model033
## 4   0.111   0.778 accuracy binary     0.956    10 0.00764 Preprocessor1_Model072
## 5   0.333   0.222 accuracy binary     0.955    10 0.00511 Preprocessor1_Model024
```

```
besten # penalty = 0.222 ; mixture = 0.444 ; accuracy = 0.959
```

```
## # A tibble: 1 x 3
##   penalty mixture .config
##     <dbl>   <dbl> <chr>
## 1   0.222   0.444 Preprocessor1_Model043
```

The best elastic net logistic regression model is with a penalty of 0.222 and a mixture of 0.444.

The accuracy of this model is 0.922.

Overall, the two models that I want to try on the testing data were the **Logistic Regression** and the **K-nearest neighbor** models.

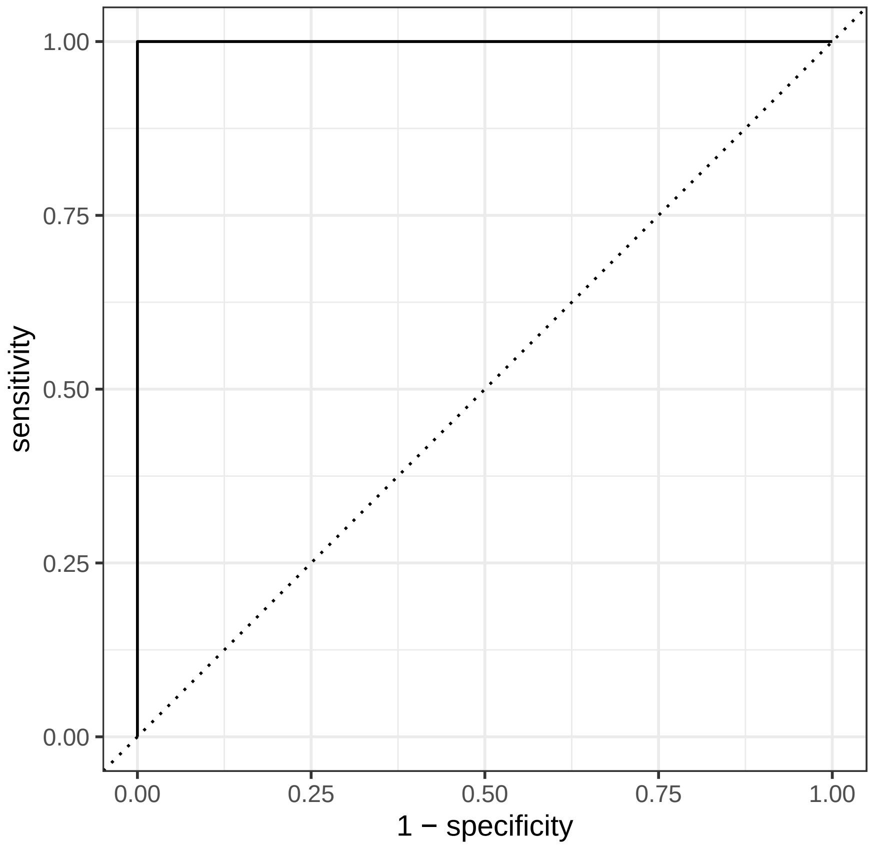
Now let's fit the data on our testing set and see how they perform!

Testing Set Fitting

I am going to fit the `Knn` and `log-reg` models to the test data that we split earlier from the original data splitting.

Logisitic Regression Testing Fit

```
log_testing<- augment(log_fit, new_data = Modernraptor_test) %>% accuracy(truth = allpro, estimate = .pr

augment(log_fit, new_data = Modernraptor_test) %>%
  roc_curve(.pred_class, .pred_0) %>%
  autoplot()
```




```
augment(log_fit, new_data = Modernraptor_test) %>%
  mutate(allpro = factor(allpro)) %>%
  roc_auc(truth = allpro, estimate = .pred_0)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.942
```

```
log_testing #0.924
```

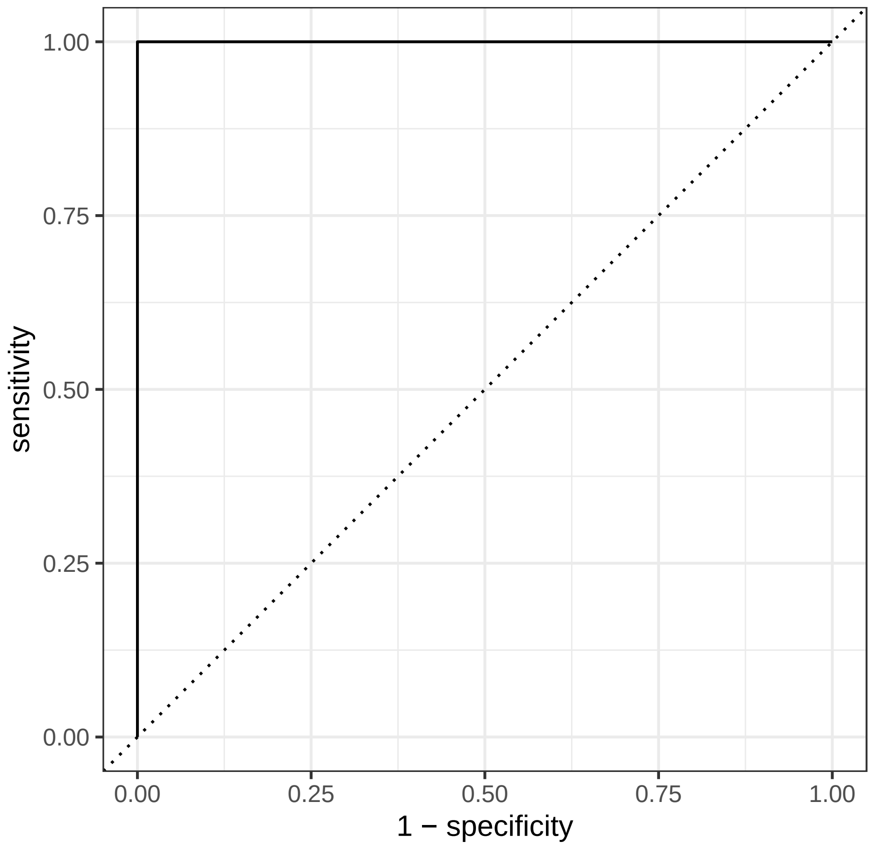
```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.924
```

I got an roc_auc of 0.942 and an accuracy of 0.924.

KNN Testing Fit

Lets apply the testing data set to the KNN model we built on the training set of data.

```r
knn_final <- finalize_workflow(knn_wkflow,
                                      bestknn)

knn_final<- fit(knn_final,
                        data = Modernraptor_test)
augment(knn_final, new_data = Modernraptor_test) %>%
  roc_curve(.pred_class, .pred_0) %>%
  autoplot()
```




```r
augment(knn_final, new_data = Modernraptor_test) %>%
  mutate(allpro = factor(allpro)) %>%
  roc_auc(truth = allpro, estimate = .pred_0) #0.993
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.993
```

```r
augment(knn_final, new_data = Modernraptor_test) %>%
  accuracy(truth = allpro, estimate = factor(.pred_0)) # 0.986
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary        0.0139
```

The `knn` model had an `accuracy of 0.986` and a `roc_auc of 0.993`. This is a really high value of accuracy and tells me that my model and predictor variables are really,really good at predicting whether or not a player will make the all pro team.

Conclusion

The research question that I had coming into this project was to "see if I can predict who this current seasons"all-pro" team will be with a full season of data."

After building 4 different machine learning models (`Logistic Regression`, `LDA`, `KNN`, and `Elastic Net Logistic Regression`), I tested two of the models(`Logistic Regression` and `KNN`) on the testing data.

What I found is that for the `Logistic Regression` model, I could predict with `92.4% accuracy` whether or not a specific player would make the all pro team given their advanced statistics. This is a really strong accuracy percentage and suggests that my model is really strong. The `roc_auc` score for the `Logistic Regression` model was `0.942`. This is also really strong and suggests again that the model is a great fit for predicting.

A worry I have with this model is the imbalance of outcome variable data. There is a significant larger percentage of the data that did not make the all pro team compared to those who did. This could lead to a skewed model and make the output values misleading.

For the `KNN` model, I got an `accuracy of 0.986`. This is unbelievable good. I can predict with 98.6% accuracy whether or not a player made the all pro team given the players advanced statistics. My roc_auc curve value was nearly perfect at 0.993. This is also unbelievably good and suggests that the model is nearly-perfect in predicting the outcome variable.

I think that an interesting future analysis from here is to test this data on the 2023 NBA data as well as using different models on the data like a Support Vector Classifier. With the SVC, I think that I wouldn't get an optimal seperating hyperplane, because there are most definitely `outliers` in this data set like players who should have been in the all-NBA team but were "snubbed".

Overall, both models were phenomenal in predicting whether or not a player will make the all pro team for that given season. I can't wait to test this model out on this current NBA regular season!