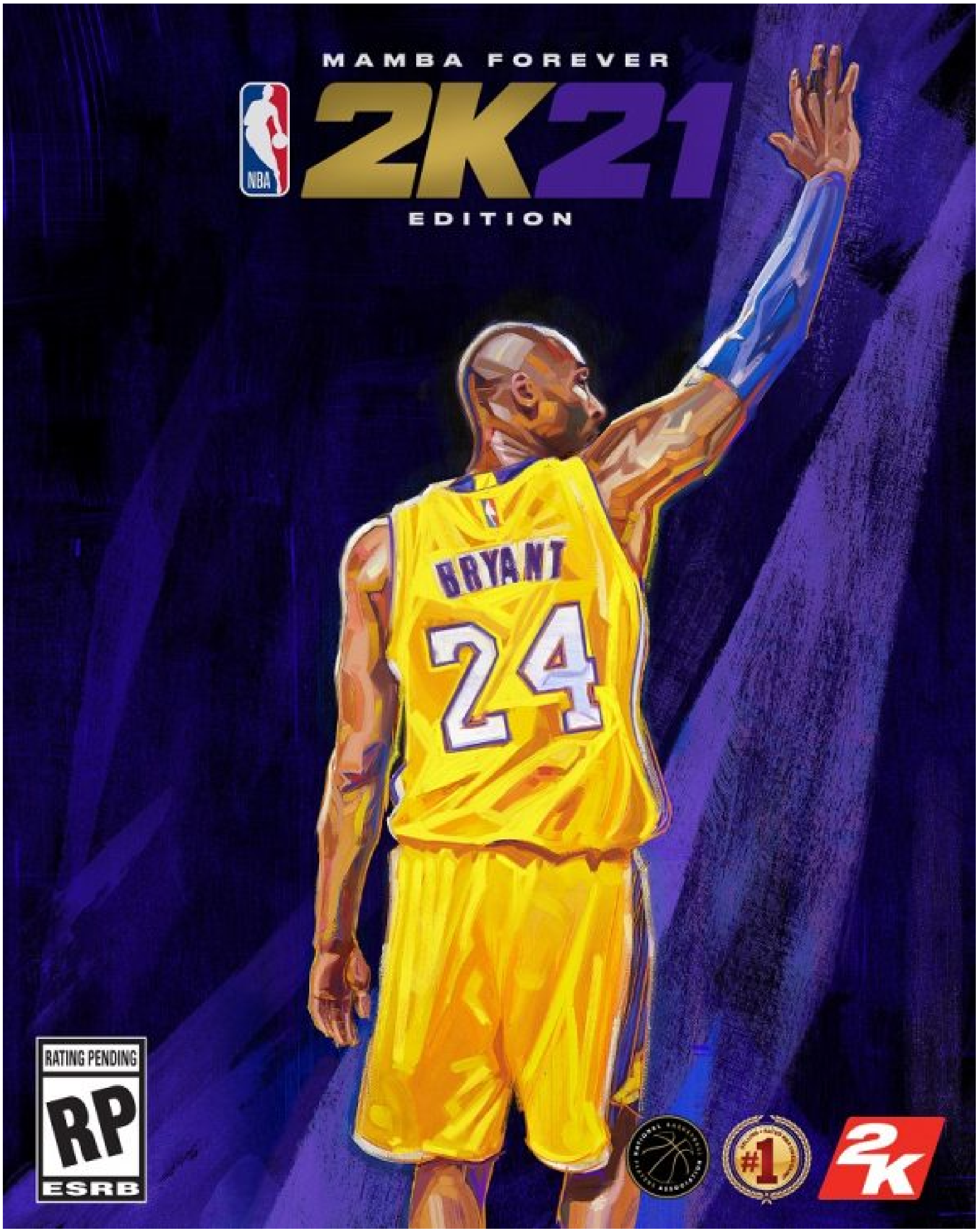
# Predicting NBA 2K21 Player Overall Ratings

PSTAT 231 Final Project

Rebecca Chang UCSB Spring 2023

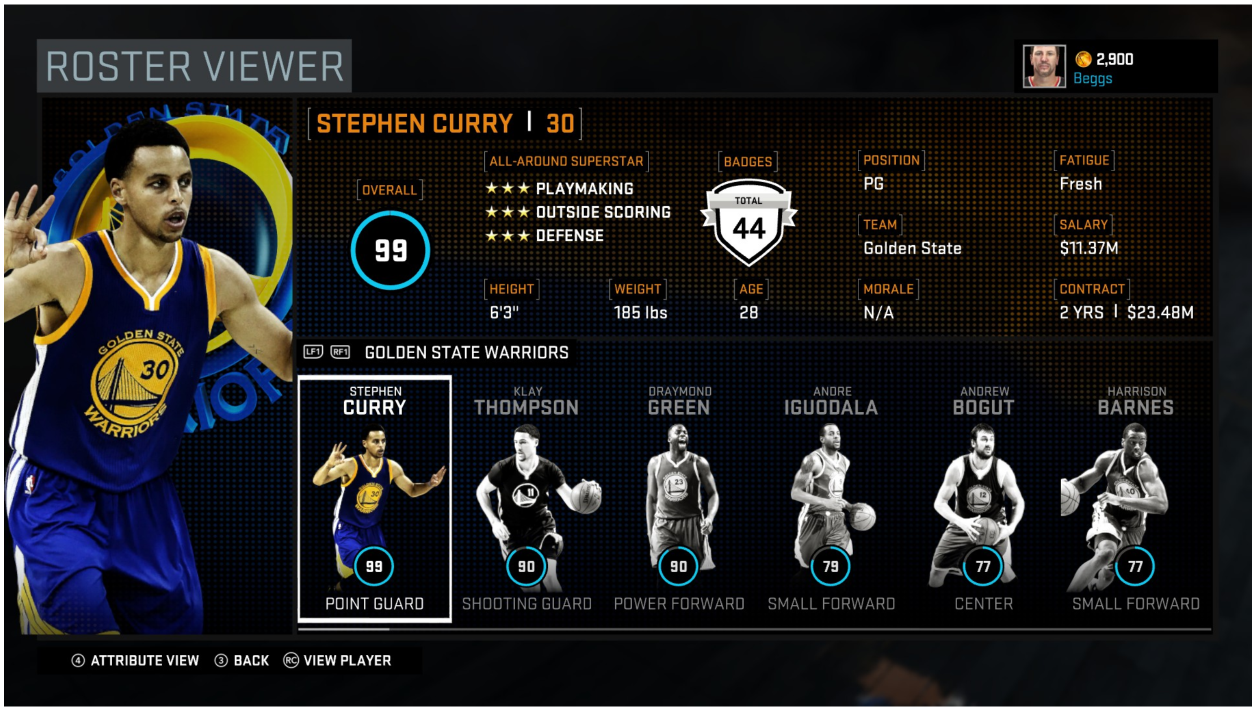
**Introduction**

The purpose of this project is to develop the best machine learning model that predicts the overall rating of NBA players in the video game NBA 2K21.


MAMBA FOREVER
NBA
2K21
EDITION
BRYANT
24
RATING PENDING
RP
ESRB
#1
2K

**What is NBA 2K?** NBA 2K is a popular basketball video game that emulates real players in the NBA (National Basketball Association) and is often hailed for its realistic graphics and gameplay. Each year, a new installment is released which updates any changes that were made to the roster of each team. It is also an opportunity to re-evaluate the rating of each player based on how well the real-life players performed in the previous season. Each installment is named after the following season and features new cover athletes. For example, NBA 2K21 was released in late 2020 after the end of the 2019-2020 season and features cover athletes Damian Lillard, Zion Williamson, and Kobe Bryant across the different editions.

**What are ratings and how are they calculated?** The rating system in NBA 2K assigns each game player/character a overall score ranging from 40 to the high 90's based on the performance of the real corresponding NBA player. This includes statistics on the players points per game, field goal percentage, plus/minus rating, and many more. Multiple players can be ranked with the same score and a higher score corresponds to a better player. Although there are numerous sub-categories such at 3-point rating for each player, this project will only focus on predicting the overall rating for simplicity.



**Why?** By accurately predicting individual player ratings, gamers are able to choose the players that will perform better in the gameplay. They will also better understand which statistics of their favorite players impact their rating and ensure that the rating is an impartial and honest representation of reality. Overall, it provides a better gaming experience and gamers can learn more about the rating system in the game.

**Project Outline** We will start by reading in our data and doing some data manipulation and cleaning to obtain only the relevant information we need. Then, we will perform some exploratory data analysis to better understand our variables and data, specifically how the other predictors relate to our prediction variable of ratings. We will then split our data into train/test sets, create a recipe, and set up a 10-fold cross validation. We will use Linear Regression, Ridge Regression, Lasso Regression, Elastic Net Linear Regression, K-Nearest Neighbors, Random Forest, and Boosted Trees to model the training data. The model that performs the best will be fit on the testing data so we can analyze its effectiveness. Let's begin!

**Exploratory Data Analysis**

By exploring the collected data and performing some preliminary analysis, we can gain a better understanding of our data and make any adjustments if necessary. This includes checking for any missing data, converting necessary qualitative variables into factors, removing unhelpful variables, and tidying up the data before we begin visualizing and analyzing key variables.

**Loading Packages and Exploring Data** We want to first load any R packages we will be needing later on and read in our data.

```
library(tidyverse)
library(dplyr)
library(tidymodels)
library(readr)
library(kknn)
library(ISLR)
library(discrim)
library(poissonreg)
library(glmnet)
library(corrr)
library(corrplot)
library(ggplot2)
library(ggthemes)
library(naniar)
library(janitor)
library(vip)
tidymodels_prefer()

# Assigning a name to the dataset
ratings <- read.csv("/Users/Rebecca/PSTAT 231 S'23/nba_ratings_2014-2020.csv")

# View the first 10 rows of data
head(ratings, 10)
```

```
##    X           PLAYER TEAM AGE  SEASON GP  W  L  MIN  PTS FGM  FGA  FG. X3PM
## 1  0      Aaron Gordon  ORL  24 2019-20 62 30 32 32.5 14.4 5.4 12.4 43.7  1.2
## 2  1     Aaron Holiday  IND  23 2019-20 66 42 24 24.5  9.5 3.5  8.5 41.4  1.3
## 3  2       Abdel Nader  OKC  26 2019-20 55 37 18 15.8  6.3 2.2  4.8 46.8  0.9
## 4  3       Adam Mokoka  CHI  21 2019-20 11  3  8 10.2  2.9 1.1  2.5 42.9  0.5
## 5  4 Admiral Schofield  WAS  23 2019-20 33  9 24 11.2  3.0 1.1  2.8 38.0  0.6
## 6  5        Al Horford  PHI  34 2019-20 67 39 28 30.2 11.9 4.8 10.6 45.0  1.5
## 7  6   Al-Farouq Aminu  ORL  29 2019-20 18  7 11 21.1  4.3 1.4  4.8 29.1  0.5
## 8  7        Alec Burks  PHI  28 2019-20 66 21 45 26.6 15.0 4.9 11.6 41.8  1.8
## 9  8    Alen Smailagic  GSW  19 2019-20 14  1 13  9.9  4.2 1.4  2.9 50.0  0.2
## 10 9       Alex Caruso  LAL  26 2019-20 64 48 16 18.4  5.5 1.9  4.5 41.2  0.6
##    X3PA X3P. FTM FTA  FT. OREB DREB REB AST TOV STL BLK  PF   FP DD2 TD3 X...
## 1   3.8 30.8 2.4 3.6 67.4  1.7  5.9 7.7 3.7 1.6 0.8 0.6 2.0 31.9  20   1 -1.1
## 2   3.3 39.4 1.1 1.3 85.1  0.3  2.0 2.4 3.4 1.3 0.8 0.2 1.8 19.3   3   0  1.7
## 3   2.3 37.5 0.9 1.2 77.3  0.3  1.6 1.8 0.7 0.8 0.4 0.4 1.4 11.1   0   0 -1.5
## 4   1.4 40.0 0.2 0.4 50.0  0.6  0.3 0.9 0.4 0.2 0.4 0.0 1.5  5.5   0   0  4.5
## 5   1.8 31.1 0.3 0.5 66.7  0.2  1.2 1.4 0.5 0.2 0.2 0.1 1.5  6.3   0   0 -1.7
## 6   4.2 35.0 0.9 1.2 76.3  1.5  5.3 6.8 4.0 1.2 0.8 0.9 2.1 30.0   6   0  1.9
## 7   2.0 25.0 1.1 1.6 65.5  1.3  3.5 4.8 1.2 0.9 1.0 0.4 1.5 15.3   1   0 -1.9
## 8   4.6 38.5 3.6 4.0 88.7  0.7  3.5 4.3 2.9 1.4 0.9 0.3 1.9 26.7   1   0 -3.4
## 9   0.9 23.1 1.1 1.4 84.2  0.7  1.2 1.9 0.9 0.8 0.2 0.3 1.0  8.6   0   0 -0.6
## 10  1.9 33.3 1.1 1.5 73.4  0.3  1.7 1.9 1.9 0.8 1.1 0.3 1.5 14.0   0   0  3.8
##    ratings
## 1       80
## 2       76
## 3       71
## 4       68
## 5       71
## 6       80
## 7       75
## 8       77
## 9       71
## 10      74
```

```
view(ratings)
```

We notice that the data lists each player alphabetically and provides information about their team, age, various game statistics, as well as rating for each season from 2014 to 2020.

**Tidying Our Data** The dataset we have currently is very large and includes information that is not necessary for this project. Since the data includes multiple years, we want to filter our data to see only the most recently available year. There are also many predictor variables, so we want to further filter our variables so only the most relevant variables are used in our prediction. For example, team name `TEAM`, observation number `X`, and `SEASON` don't seem to be helpful in predicting the rating, so we will take those variables out. Lastly, we will clean the column names so that they are all snake case and are standardized.

```
# Filtering the year and variables
ratings19_20 <- ratings %>%
  filter(SEASON == "2019-20") %>%
  select(-TEAM, -X, -SEASON) %>%
  clean_names()

head(ratings19_20)
```

```
##               player age gp  w  l  min  pts fgm  fga   fg x3pm x3pa  x3p ftm fta
## 1      Aaron Gordon  24 62 30 32 32.5 14.4 5.4 12.4 43.7  1.2  3.8 30.8 2.4 3.6
## 2     Aaron Holiday  23 66 42 24 24.5  9.5 3.5  8.5 41.4  1.3  3.3 39.4 1.1 1.3
## 3       Abdel Nader  26 55 37 18 15.8  6.3 2.2  4.8 46.8  0.9  2.3 37.5 0.9 1.2
## 4       Adam Mokoka  21 11  3  8 10.2  2.9 1.1  2.5 42.9  0.5  1.4 40.0 0.2 0.4
## 5 Admiral Schofield  23 33  9 24 11.2  3.0 1.1  2.8 38.0  0.6  1.8 31.1 0.3 0.5
## 6        Al Horford  34 67 39 28 30.2 11.9 4.8 10.6 45.0  1.5  4.2 35.0 0.9 1.2
##     ft oreb dreb reb ast tov stl blk  pf   fp dd2 td3    x ratings
## 1 67.4  1.7  5.9 7.7 3.7 1.6 0.8 0.6 2.0 31.9  20   1 -1.1      80
## 2 85.1  0.3  2.0 2.4 3.4 1.3 0.8 0.2 1.8 19.3   3   0  1.7      76
## 3 77.3  0.3  1.6 1.8 0.7 0.8 0.4 0.4 1.4 11.1   0   0 -1.5      71
## 4 50.0  0.6  0.3 0.9 0.4 0.2 0.4 0.0 1.5  5.5   0   0  4.5      68
## 5 66.7  0.2  1.2 1.4 0.5 0.2 0.2 0.1 1.5  6.3   0   0 -1.7      71
## 6 76.3  1.5  5.3 6.8 4.0 1.2 0.8 0.9 2.1 30.0   6   0  1.9      80
```

```
dim(ratings19_20)
```

```
## [1] 507  29
```

Our data now includes 507 observations and 29 key variables, including our response variable `ratings`. A brief description of the following variables are as follows:

`player`: Name of NBA player

`age`: Age of player

`gp`: Games played

`w`: Number of games won

`l`: Number of games lost

`min`: Minutes played per game

`pts`: Points per game

`fgm`: Field goals made per game

`fga`: Field goal attempts per game

`fg.`: Field goal percentage

`x3pm`: 3-pointers made per game

`x3pa`: 3-point attempts per game

`x3p`: 3-point percentage

`ftm`: Free throws made per game

`fta`: Free throw attempts per game

`ft`: Free throw percentage

`oreb`: Offensive rebounds per game

`dreb`: Defensive rebounds per game

`reb`: Rebounds per game

`ast`: Assists per game

`tov`: Turnovers per game

`stl`: Steals per game

`blk`: Blocks per game

`pf`: Personal fouls per game

`fp`: Fantasy points

`dd2`: Number of double-doubles

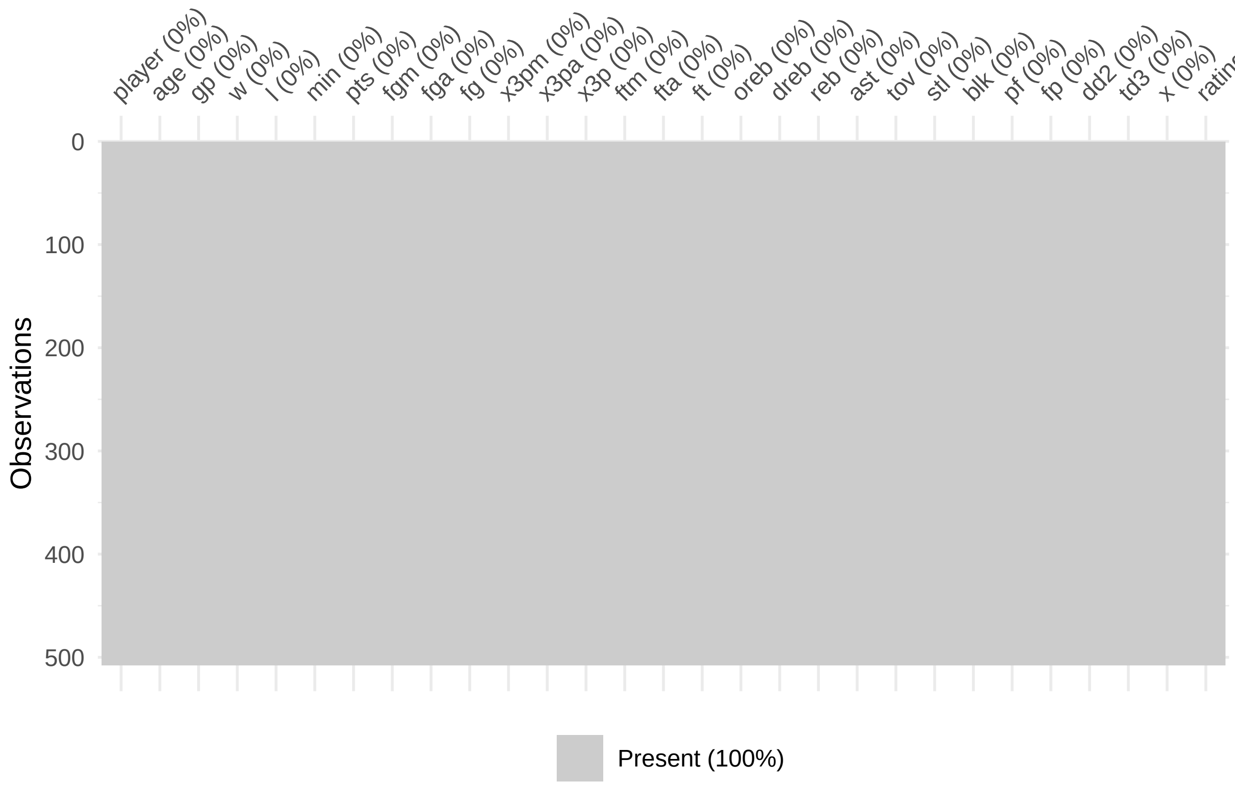
`td3`: Number of triple-doubles

`x`: +/- player point differential

`ratings`: Rating of the player on a 99-point scale

Now let's check if there's any missing data before we dig a little deeper into our data.

```
vis_miss(ratings19_20)
```



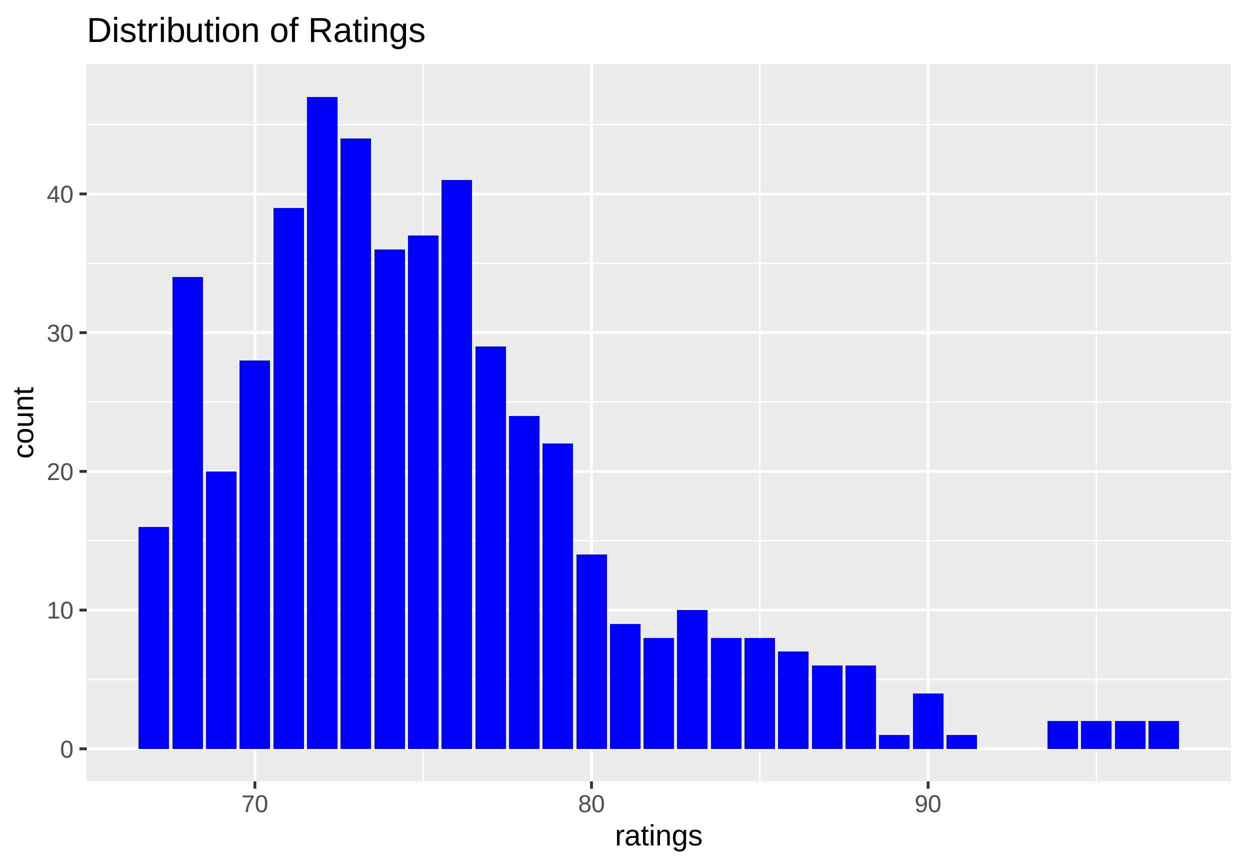
**Missing Data**

We see that there are no missing data, so we can proceed to the next step in our visual exploratory data analysis.

## Visual EDA

**Ratings** We will first take a closer look at our response variable `ratings` by plotting a histogram.

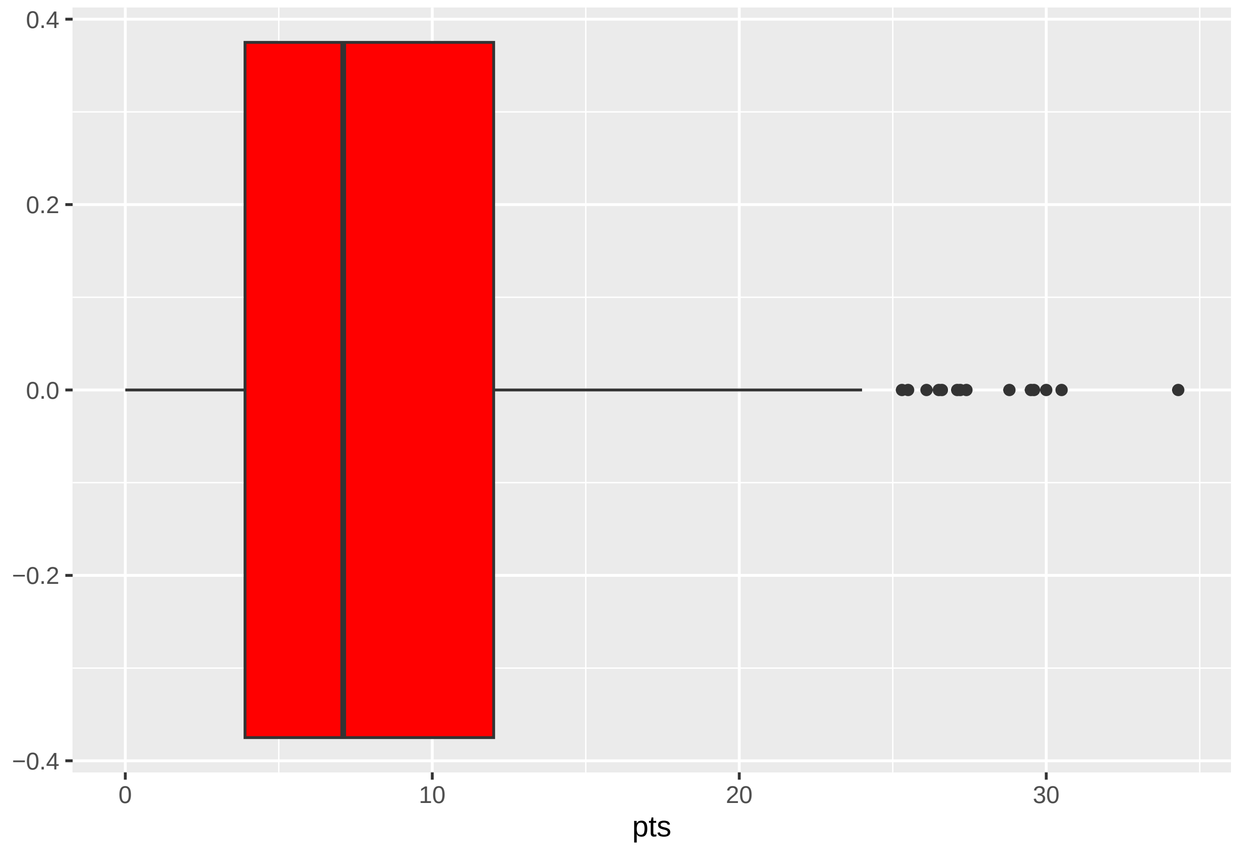
```
ggplot(ratings19_20, aes(ratings)) +
  geom_bar(fill='blue') +
  labs(
    title = "Distribution of Ratings"
  )
```



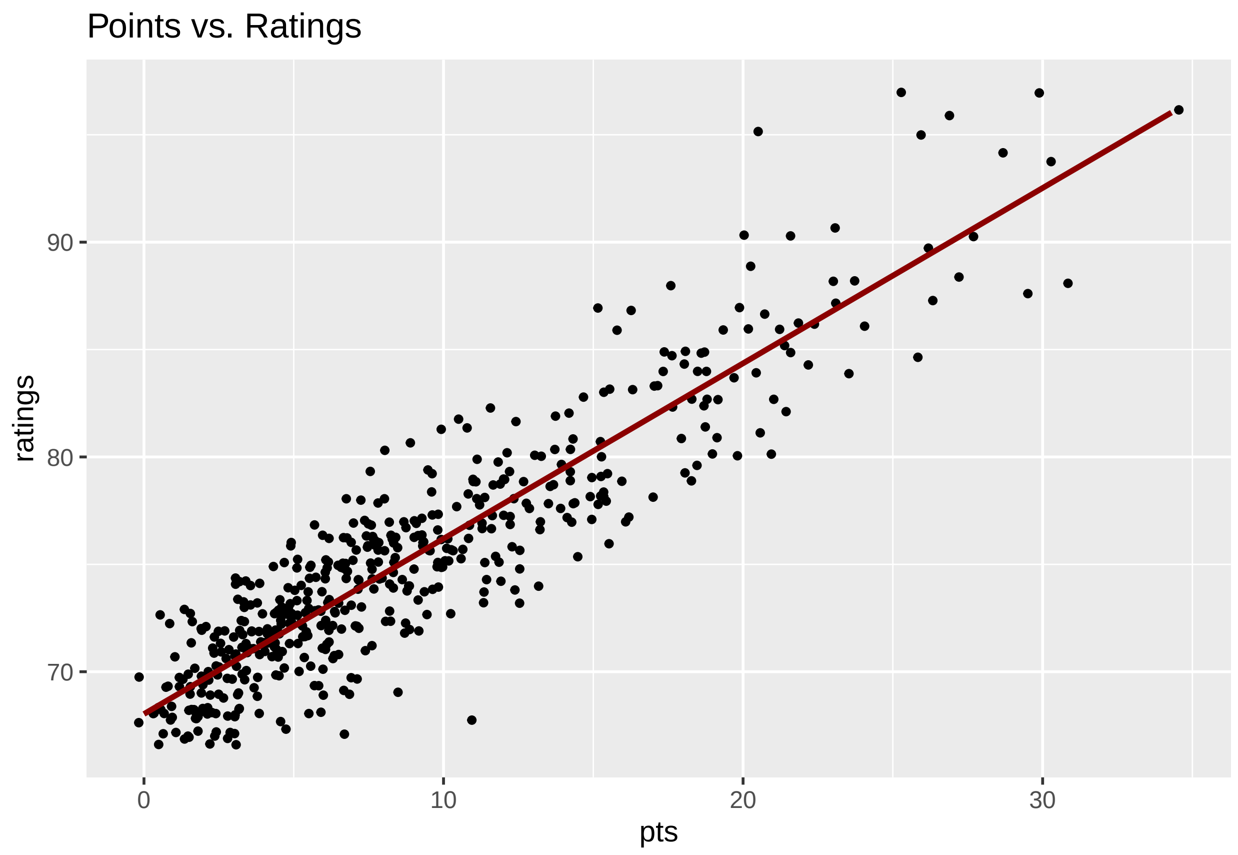
We notice that in this particular season, the ratings range from 67 to 97. There is also a peak at 72, which means the most common rating for the season is 72 with a mean that seems to be around 75. Only a handful of players have a rating higher than 90.

**PTS** Next, we will look at the variable `pts`, which is one of the most common statistics when judging a player's performance.

```
ggplot(ratings19_20, aes(pts)) +
  geom_boxplot(fill = "red")
```



```r
ratings19_20 %>%
  ggplot(aes(x=pts, y=ratings)) +
  geom_jitter(width = 0.5, size = 1) +
  geom_smooth(method = "lm", se =F, col="darkred") +
  labs(title = "Points vs. Ratings")
```

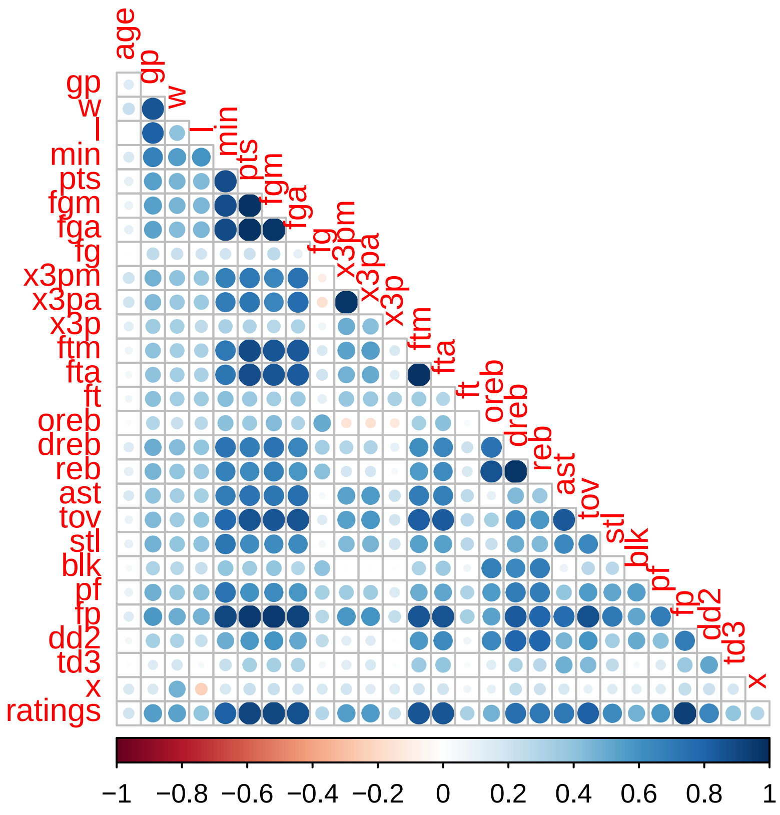


Looking at the boxplot, we notice that the average points per game across all the players are between 0 and the mid-30s, with the most common being around 8 points per game. There are several outliers for players who achieved points per game higher than 25.

After plotting the graph to take a closer look at the relationship between points and ratings, we see an apparent positive linear association. This means that players who had more points per game tended to have a higher overall rating, as expected.

**Correlation Plot** We will now create a correlation plot to see how each of the numerical variables relate to each other.

```
ratings19_20  %>%
  select(is.numeric) %>%
  cor() %>%
  corrplot(type = "lower", diag = FALSE, method = "circle")
```



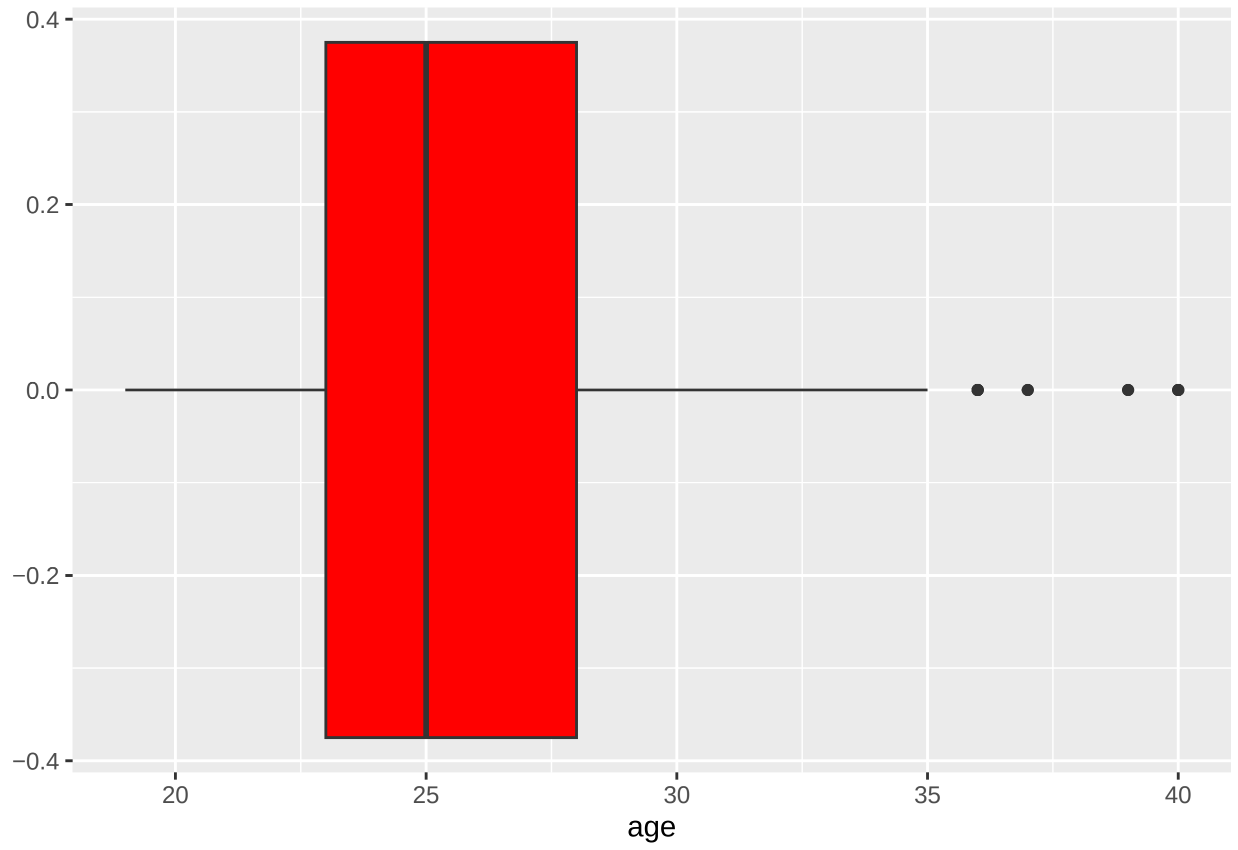
Based on the plot, we see that variables that have a strong positive correlation appear in darker blue and have bigger circles, while variables that have a negative correlation appear in red. Since we are predicting `ratings`, we will focus on variables that seem to have a strong correlation with that variable.

One relationship that stands out is the strong positive correlation between `ratings` and the variables `pts`, `fgm`, `ftm`, and `fp`. This makes sense because it is expected for a player who performs better by scoring more points, making more field goals and free throws, and racking up more fantasy points to have a higher rating. This is also verified in the histogram of `pts` and `ratings` we plotted before.

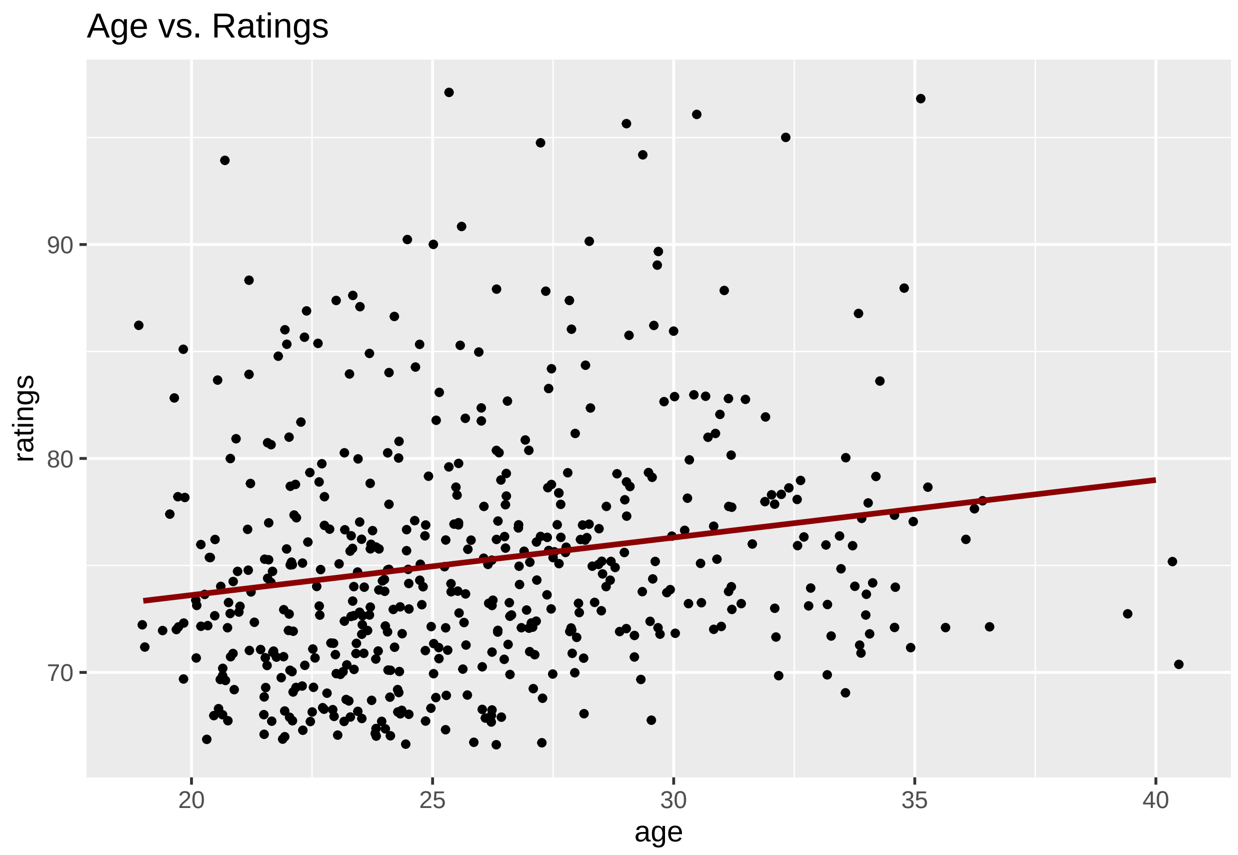
Another apparent relationship is the weak positive correlation between `age` and `ratings`, which implies that age has little affect on the rating of a player.

**Age** Let's take a closer look at the age variable to confirm our findings in the correlation plot.

```
ggplot(ratings19_20, aes(age)) +
  geom_boxplot(fill = "red")
```



```
ratings19_20 %>%
  ggplot(aes(x=age, y=ratings)) +
  geom_jitter(width = 0.5, size = 1) +
  geom_smooth(method = "lm", se =F, col="darkred") +
  labs(title = "Age vs. Ratings")
```



Looking at the boxplot, we notice that the age across all the players are between 19 and 40, with the average being 25 years old. There are a few outliers for players who are in their late 30s or are 40.

The graph showing the relationship between age and ratings also gives insight into how the two variables are related. Although there is a positive and linear trend, it is not very strong which means players can have a high rating in a wide range of ages. This makes sense because younger players could be more athletic and therefore perform better, while older players could have more experience and can also perform well.

### Setting Up Models

Now we are ready to start fitting different models to our data and see if any of them can accurately predict the rating of a player based on the predictors we have.

**Test/Train Split** We will split our data into a training and testing set by setting 80% of the data as training data and the rest as testing data. We will also stratify our predictor variable `ratings` to ensure the resamples have equivalent proportions as the original data set. This split allows us to train our model on a majority of the data first, then apply it to the testing data to determine its accuracy and to avoid over-fitting.

```
# Set seed for reproducible results
set.seed(1234)

ratings_split <- initial_split(ratings19_20, prop = 0.80,
                               strata = ratings)
ratings_train <- training(ratings_split)
```

```
ratings_test <- testing(ratings_split)

dim(ratings_train)
```

```
## [1] 403  29
```

```
dim(ratings_test)
```

```
## [1] 104  29
```

We can check the dimensions of the data sets to make sure there is a good amount of data in each set for the model to train and test on.

**Creating Recipe** Throughout our analysis, we will be essentially using the same procedures on different models, so it is a good idea to create 1 universal recipe using the training data for all our models, and we can make slight adjustments if necessary.

We will be using all the predictors except for `player` because we only need that for identification rather than making a prediction on `ratings`. We will also center and scale the predictors to normalize our variables.

```
ratings_recipe <- recipe(ratings ~ ., data = ratings_train) %>%
  step_rm(player) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors())

prep(ratings_recipe) %>%
  bake(new_data = ratings_train) %>%
  head()
```

```
## # A tibble: 6 x 28
##        age      gp      w      l    min    pts    fgm    fga     fg    x3pm
##      <dbl>   <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>   <dbl>
## 1  0.00180  0.582   1.13  -0.221 -0.402 -0.385 -0.445 -0.472  0.189 -0.0583
## 2 -1.20    -1.33   -1.25  -1.00  -0.998 -0.898 -0.910 -0.936 -0.168 -0.506
## 3 -0.722   -0.376  -0.828  0.249 -0.892 -0.883 -0.910 -0.875 -0.617 -0.394
## 4 -1.69    -1.20   -1.39  -0.612 -1.03  -0.702 -0.783 -0.855  0.482 -0.841
## 5  0.243   -0.0714 -0.688  0.641 -0.508 -0.641 -0.614 -0.572 -0.187 -0.617
## 6 -0.480   -1.07   -0.758 -1.08  -1.35  -1.03  -1.08  -1.10  -0.306 -0.841
## # i 18 more variables: x3pa <dbl>, x3p <dbl>, ftm <dbl>, fta <dbl>, ft <dbl>,
## #   oreb <dbl>, dreb <dbl>, reb <dbl>, ast <dbl>, tov <dbl>, stl <dbl>,
## #   blk <dbl>, pf <dbl>, fp <dbl>, dd2 <dbl>, td3 <dbl>, x <dbl>, ratings <int>
```

**K-Fold Cross Validation** We will conduct k-fold stratified cross validation with k=10 folds. That way, each observation in the training set will be automatically assigned to one of the 10 folds. Then each fold will become a testing set once while the other k-1 folds will be the training set. Whichever model is being used is then fit to each training set and tested on the corresponding testing set. The average accuracy and root mean squared error (RMSE) can then be used as metrics on the testing set of each of the folds.

We will stratify on the prediction variable `ratings` to make sure the data in each fold are balanced.

```r
# Creating 10 folds
ratings_folds <- vfold_cv(ratings_train, v = 10, strata = ratings)
```

### Model Building

Now it's finally time to build our models! We will be using the metric of RMSE to evaluate the performance of our regression models. This metric measures the the distance between the model's predicted values and the true values of our prediction variable. Better models will result in a lower RMSE since the predicted values will have a smaller difference compared to the actual values. Of the 7 models we will be fitting, only the best-performing model will be selected for further analysis.

**Fitting Models** Each model will follow a similar process. Below is the step-by-step procedure I will be conducting for each of the models.

1. Set up the model by specifying model type, setting up its engine, and setting its mode (regression for this project) if necessary.
2. Set up the workflow, add the new model, and create the recipe.
3. Create the tuning grid to specify ranges for parameters we want to tune and how many levels of each (not applicable for linear regression model).
4. Tune the model and specify the workflow, k-fold cross validation folds, and the tuning grid for chosen parameters to tune.
5. Save the tuned models to an RDA file so we only need to run the file once and can save time in the future.
6. Load back the saved files.
7. Collect metrics of each model, arrange RMSE values in ascending order and choose the best model based on this metric.

```r


lm_model <- linear_reg() %>%
  set_engine("lm")

lm_wflow <- workflow() %>%
  add_recipe(ratings_recipe) %>%
  add_model(lm_model)

lm_fit <- fit_resamples(lm_wflow, resamples = ratings_folds)

lm_rmse <- collect_metrics(lm_fit) %>% filter(.metric == "rmse")
lm_rmse
```

### Linear Regression

```
## # A tibble: 1 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard    1.76    10   0.108 Preprocessor1_Model1
```

```
# Tuning penalty and setting mixture to 0 to specify ridge
ridge_model <- linear_reg(mixture = 0,
                          penalty = tune()) %>%
  set_mode("regression") %>%
  set_engine("glmnet")

ridge_wflow <- workflow() %>%
  add_recipe(ratings_recipe) %>%
  add_model(ridge_model)

penalty_grid <- grid_regular(penalty(range = c(-5,5)), levels = 50)
```

```
ridge_tune <- tune_grid(
  ridge_wflow,
  resamples = ratings_folds,
  grid = penalty_grid
)

save(ridge_tune, file = "ridge_tune.rda")
```

```
load("ridge_tune.rda")

collect_metrics(ridge_tune) %>% filter(.metric == "rmse")
```

**Ridge Regression**

```
## # A tibble: 50 x 7
##      penalty .metric .estimator  mean     n std_err .config
##        <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1 0.00001   rmse    standard    1.75    10   0.107 Preprocessor1_Model01
##  2 0.0000160 rmse    standard    1.75    10   0.107 Preprocessor1_Model02
##  3 0.0000256 rmse    standard    1.75    10   0.107 Preprocessor1_Model03
##  4 0.0000409 rmse    standard    1.75    10   0.107 Preprocessor1_Model04
##  5 0.0000655 rmse    standard    1.75    10   0.107 Preprocessor1_Model05
##  6 0.000105  rmse    standard    1.75    10   0.107 Preprocessor1_Model06
##  7 0.000168  rmse    standard    1.75    10   0.107 Preprocessor1_Model07
##  8 0.000268  rmse    standard    1.75    10   0.107 Preprocessor1_Model08
##  9 0.000429  rmse    standard    1.75    10   0.107 Preprocessor1_Model09
## 10 0.000687  rmse    standard    1.75    10   0.107 Preprocessor1_Model10
## # i 40 more rows
```

```
best_ridge <- show_best(ridge_tune, n = 1)
best_ridge
```

```
## # A tibble: 1 x 7
##   penalty .metric .estimator  mean     n std_err .config
##     <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 0.00001 rmse    standard    1.75    10   0.107 Preprocessor1_Model01
```

```r
# Tuning penalty and setting mixture to 1 to specify lasso
lasso_model <- linear_reg(penalty = tune(),
                          mixture = 1) %>%
  set_mode("regression") %>%
  set_engine("glmnet")

lasso_wflow <- workflow() %>%
  add_recipe(ratings_recipe) %>%
  add_model(lasso_model)

penalty_grid <- grid_regular(penalty(range = c(-5,5)), levels = 50)
```

```r
lasso_tune <- tune_grid(
  lasso_wflow,
  resamples = ratings_folds,
  grid = penalty_grid
)

save(lasso_tune, file = "lasso_tune.rda")
```

```r
load("lasso_tune.rda")

collect_metrics(lasso_tune) %>% filter(.metric == "rmse")
```

### Lasso Regression

```
## # A tibble: 50 x 7
##      penalty .metric .estimator  mean     n std_err .config
##        <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1 0.00001   rmse    standard    1.75    10   0.106 Preprocessor1_Model01
##  2 0.0000160 rmse    standard    1.75    10   0.106 Preprocessor1_Model02
##  3 0.0000256 rmse    standard    1.75    10   0.106 Preprocessor1_Model03
##  4 0.0000409 rmse    standard    1.75    10   0.106 Preprocessor1_Model04
##  5 0.0000655 rmse    standard    1.75    10   0.106 Preprocessor1_Model05
##  6 0.000105  rmse    standard    1.75    10   0.106 Preprocessor1_Model06
##  7 0.000168  rmse    standard    1.75    10   0.106 Preprocessor1_Model07
##  8 0.000268  rmse    standard    1.75    10   0.106 Preprocessor1_Model08
##  9 0.000429  rmse    standard    1.75    10   0.106 Preprocessor1_Model09
## 10 0.000687  rmse    standard    1.75    10   0.106 Preprocessor1_Model10
## # i 40 more rows
```

```r
best_lasso <- show_best(lasso_tune, n = 1)
best_lasso
```

```
## # A tibble: 1 x 7
##   penalty .metric .estimator  mean     n std_err .config
##     <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1  0.0295 rmse    standard    1.70    10  0.0979 Preprocessor1_Model18
```

```r
elastic_net <- linear_reg(penalty = tune(),
                          mixture = tune()) %>%
  set_mode("regression") %>%
  set_engine("glmnet")

elastic_wflow <- workflow() %>%
  add_recipe(ratings_recipe) %>%
  add_model(elastic_net)

elastic_grid <- grid_regular(penalty(range = c(-5, 5)),
                             mixture(range = c(0, 1)),
                             levels = 10)
```

```r
elastic_tune <- tune_grid(
  elastic_wflow,
  resamples = ratings_folds,
  grid = elastic_grid
)

save(elastic_tune, file = "elastic_tune.rda")
```

```r
load("elastic_tune.rda")

collect_metrics(elastic_tune) %>% filter(.metric == "rmse")
```

### Elastic Net Linear Regression

```
## # A tibble: 100 x 8
##          penalty mixture .metric .estimator  mean     n std_err .config
##            <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1      0.00001        0 rmse    standard    1.75    10   0.107 Preprocessor1_M~
##  2      0.000129       0 rmse    standard    1.75    10   0.107 Preprocessor1_M~
##  3      0.00167        0 rmse    standard    1.75    10   0.107 Preprocessor1_M~
##  4      0.0215         0 rmse    standard    1.75    10   0.107 Preprocessor1_M~
##  5      0.278          0 rmse    standard    1.75    10   0.107 Preprocessor1_M~
##  6      3.59           0 rmse    standard    1.87    10   0.116 Preprocessor1_M~
##  7     46.4            0 rmse    standard    2.93    10   0.152 Preprocessor1_M~
##  8    599.             0 rmse    standard    5.33    10   0.247 Preprocessor1_M~
##  9   7743.             0 rmse    standard    5.92    10   0.268 Preprocessor1_M~
## 10 100000              0 rmse    standard    5.92    10   0.268 Preprocessor1_M~
## # i 90 more rows
```

```r
best_en <- show_best(elastic_tune, n = 1)
best_en
```

```
## # A tibble: 1 x 8
##   penalty mixture .metric .estimator  mean     n std_err .config
##     <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1  0.0215       1 rmse    standard    1.70    10  0.0991 Preprocessor1_Model094
```

```
knn_model <- nearest_neighbor(neighbors = tune()) %>%
  set_mode("regression") %>%
  set_engine("kknn")


knn_wflow <- workflow() %>%
  add_model(knn_model) %>%
  add_recipe(ratings_recipe)

knn_grid <- grid_regular(neighbors(range=c(1,15)), levels = 10)
```

```
knn_tune <- tune_grid(
  knn_wflow,
  resamples = ratings_folds,
  grid = knn_grid
)

save(knn_tune, file = "knn_tune.rda")
```

```
load("knn_tune.rda")

collect_metrics(knn_tune) %>% filter(.metric == "rmse")
```

### K-Nearest Neighbors

```
## # A tibble: 10 x 7
##    neighbors .metric .estimator  mean     n std_err .config
##        <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1         1 rmse    standard    2.50    10  0.0507 Preprocessor1_Model01
##  2         2 rmse    standard    2.34    10  0.0597 Preprocessor1_Model02
##  3         4 rmse    standard    2.15    10  0.0803 Preprocessor1_Model03
##  4         5 rmse    standard    2.11    10  0.0852 Preprocessor1_Model04
##  5         7 rmse    standard    2.06    10  0.0957 Preprocessor1_Model05
##  6         8 rmse    standard    2.05    10  0.0998 Preprocessor1_Model06
##  7        10 rmse    standard    2.04    10  0.107  Preprocessor1_Model07
##  8        11 rmse    standard    2.04    10  0.110  Preprocessor1_Model08
##  9        13 rmse    standard    2.04    10  0.115  Preprocessor1_Model09
## 10        15 rmse    standard    2.04    10  0.117  Preprocessor1_Model10
```

```
best_knn <- show_best(knn_tune, n = 1)
best_knn
```

```
## # A tibble: 1 x 7
##   neighbors .metric .estimator  mean     n std_err .config
##       <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1        11 rmse    standard    2.04    10   0.110 Preprocessor1_Model08
```

```
rf_model <- rand_forest(mtry = tune(),
                        trees = tune(),
                        min_n = tune()) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("regression")

rf_wflow <- workflow() %>%
  add_recipe(ratings_recipe) %>%
  add_model(rf_model)

rf_parameter_grid <- grid_regular(mtry(range = c(1, 15)), trees(range = c(200,1000)), min_n(range = c(5
```

```
rf_tune<- tune_grid(
  rf_wflow,
  resamples = ratings_folds,
  grid = rf_parameter_grid
)

save(rf_tune, file = "rf_tune.rda")
```

```
load("rf_tune.rda")

collect_metrics(rf_tune) %>% filter(.metric == "rmse")
```

**Random Forest**

```
## # A tibble: 512 x 9
##     mtry trees min_n .metric .estimator  mean     n std_err .config
##    <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1     1   200     5 rmse    standard    2.08    10  0.132  Preprocessor1_Model~
##  2     3   200     5 rmse    standard    1.91    10  0.0903 Preprocessor1_Model~
##  3     5   200     5 rmse    standard    1.86    10  0.0867 Preprocessor1_Model~
##  4     7   200     5 rmse    standard    1.86    10  0.0824 Preprocessor1_Model~
##  5     9   200     5 rmse    standard    1.82    10  0.0688 Preprocessor1_Model~
##  6    11   200     5 rmse    standard    1.82    10  0.0713 Preprocessor1_Model~
##  7    13   200     5 rmse    standard    1.80    10  0.0747 Preprocessor1_Model~
##  8    15   200     5 rmse    standard    1.81    10  0.0701 Preprocessor1_Model~
##  9     1   314     5 rmse    standard    2.07    10  0.135  Preprocessor1_Model~
## 10     3   314     5 rmse    standard    1.89    10  0.0963 Preprocessor1_Model~
## # i 502 more rows
```

```
best_rf <- show_best(rf_tune, n = 1)
best_rf
```

```
## # A tibble: 1 x 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    15   771     5 rmse    standard    1.79    10  0.0706 Preprocessor1_Model0~
```

```
boosted_model <- boost_tree(trees = tune(),
                            learn_rate = tune(),
                            min_n = tune()) %>%
  set_mode("regression") %>%
  set_engine("xgboost")

boosted_wflow <- workflow() %>%
  add_recipe(ratings_recipe) %>%
  add_model(boosted_model)

boosted_grid <- grid_regular(trees(range = c(5, 200)), learn_rate(range = c(0.01,0.1), trans = identity
```

```
boosted_tune <- tune_grid(
  boosted_wflow,
  resamples = ratings_folds,
  grid = boosted_grid
)

save(boosted_tune, file = "boosted_tune.rda")
```

```
load("boosted_tune.rda")

collect_metrics(boosted_tune) %>% filter(.metric == "rmse")
```

### Boosted Trees

```
## # A tibble: 125 x 9
##    trees min_n learn_rate .metric .estimator  mean     n std_err .config
##    <int> <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1     5    40       0.01 rmse    standard    71.4    10   0.196 Preprocessor1_~
##  2    53    40       0.01 rmse    standard    44.3    10   0.182 Preprocessor1_~
##  3   102    40       0.01 rmse    standard    27.2    10   0.168 Preprocessor1_~
##  4   151    40       0.01 rmse    standard    16.8    10   0.160 Preprocessor1_~
##  5   200    40       0.01 rmse    standard    10.5    10   0.159 Preprocessor1_~
##  6     5    45       0.01 rmse    standard    71.4    10   0.196 Preprocessor1_~
##  7    53    45       0.01 rmse    standard    44.3    10   0.182 Preprocessor1_~
##  8   102    45       0.01 rmse    standard    27.2    10   0.167 Preprocessor1_~
##  9   151    45       0.01 rmse    standard    16.8    10   0.160 Preprocessor1_~
## 10   200    45       0.01 rmse    standard    10.5    10   0.161 Preprocessor1_~
## # i 115 more rows
```

```
best_boosted <- show_best(boosted_tune, n = 1)
best_boosted
```

```
## # A tibble: 1 x 9
##   trees min_n learn_rate .metric .estimator  mean     n std_err .config
##   <int> <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1   151    40      0.055 rmse    standard    1.94    10   0.107 Preprocessor1_M~
```

**Model Results**

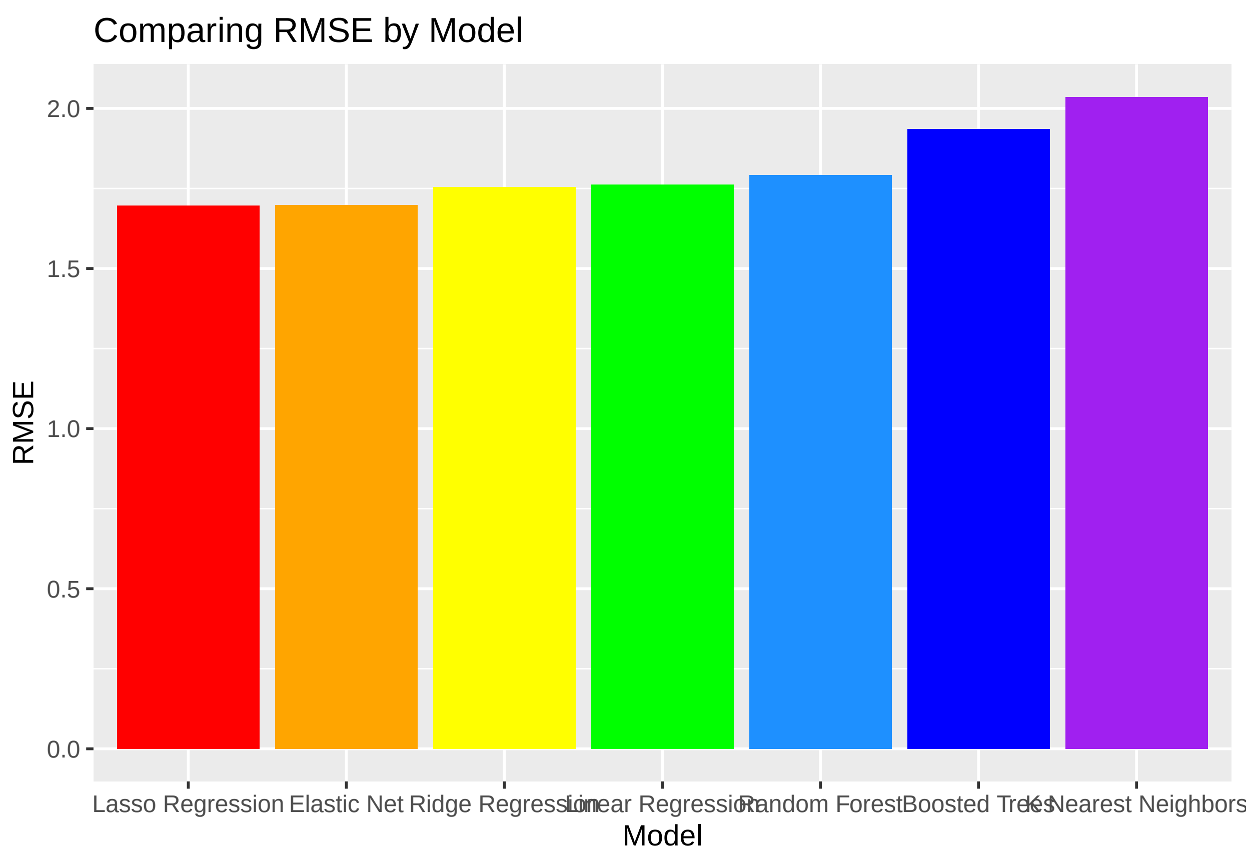
Now it's time to compare the results of each of the best models and see which is the best of the best!

```
final_compare <- tibble(Model = c("Linear Regression", "Ridge Regression", "Lasso Regression", "Elastic
                        RMSE = c(lm_rmse$mean, best_ridge$mean, best_lasso$mean, best_en$mean, best_knn

# Arrange by lowest RMSE
final_compare <- final_compare %>%
  arrange(RMSE)

final_compare
```

```
## # A tibble: 7 x 2
##   Model                RMSE
##   <chr>               <dbl>
## 1 Lasso Regression     1.70
## 2 Elastic Net          1.70
## 3 Ridge Regression     1.75
## 4 Linear Regression    1.76
## 5 Random Forest        1.79
## 6 Boosted Trees        1.94
## 7 K-Nearest Neighbors  2.04
```

We can also visualize the best models by plotting their RMSEs in a barplot.

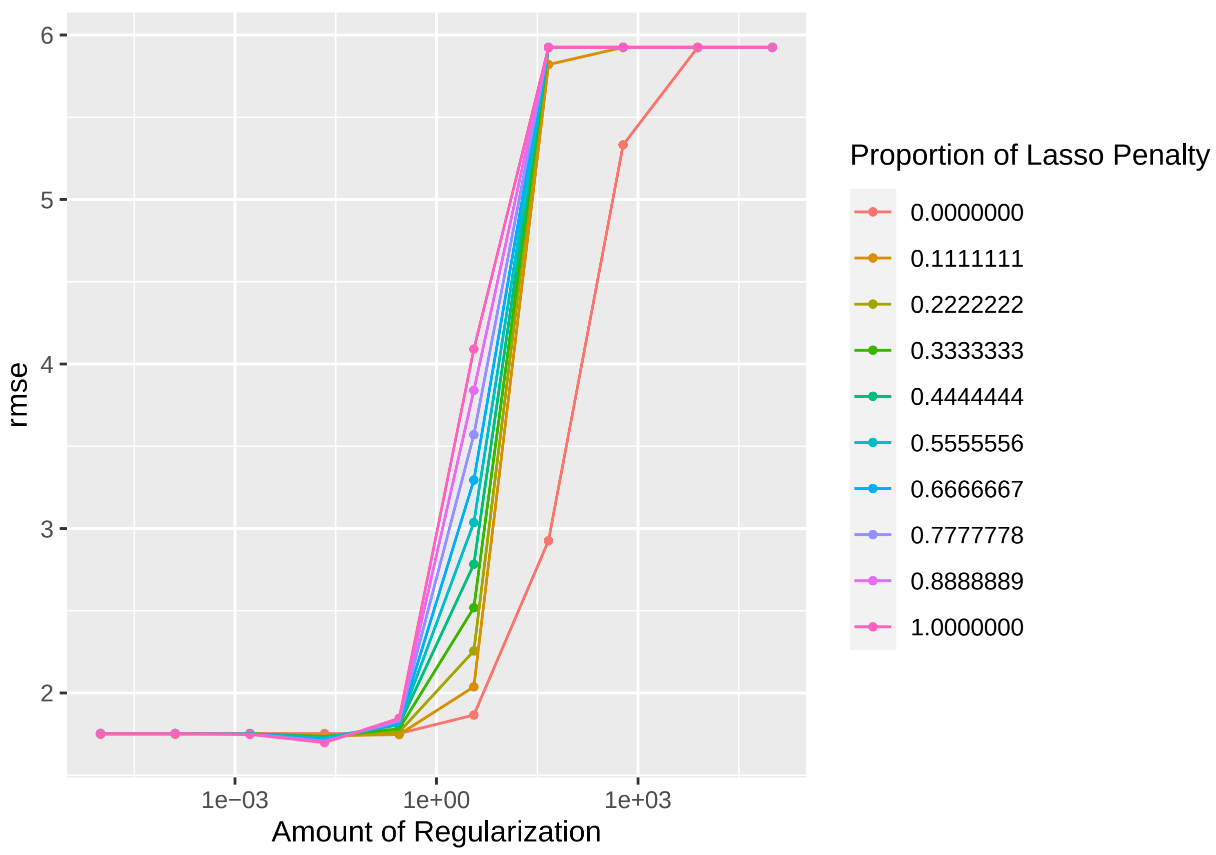
```
# Create a data frame of the model RMSE's so we can plot
all_models <- data.frame(Model = c("Linear Regression", "Ridge Regression", "Lasso Regression", "Elasti
                        RMSE = c(lm_rmse$mean, best_ridge$mean, best_lasso$mean, best_en$mean, best_knn

# Create a barplot of the RMSE values in ascending order
ggplot(all_models, aes(x=reorder(Model, RMSE), y=RMSE)) +
  geom_bar(stat = "identity", aes(fill = Model)) +
  scale_fill_manual(values = c("blue", "orange", "purple", "red", "green", "dodgerblue", "yellow")) +
  theme(legend.position = "none") +
  labs(title = "Comparing RMSE by Model", x = "Model")
```



From these results, we can conclude that the model that performed the best was the lasso regression, closely followed by elastic net linear regression. The models that did third and fourth best were ridge regression and linear regression. One key observation is that the simpler regression models performed better, which indicates that the data is likely pretty linear.

**Model Autoplots** Autoplots allow us to visualize patterns in the tuned parameters on the performance of a model based on the RMSE metric.

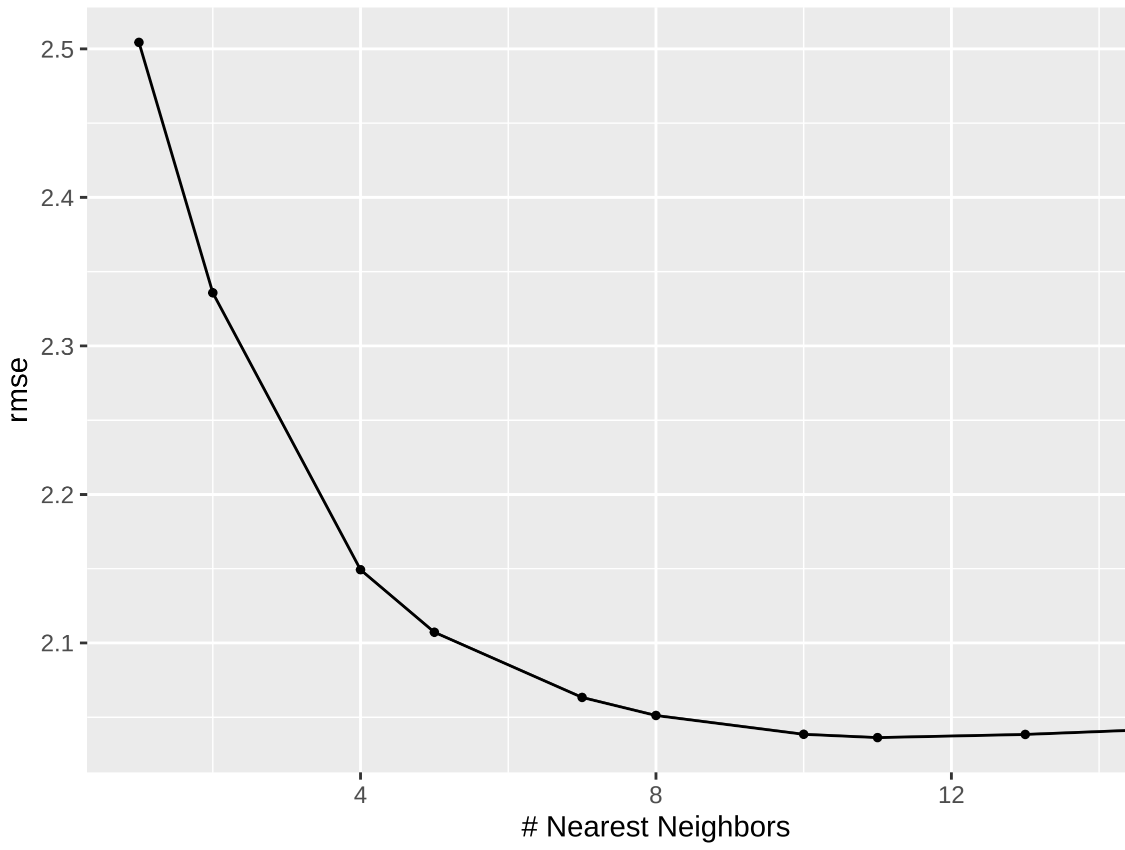
```
autoplot(elastic_tune, metric = "rmse")
```



**Elastic Net**

For the elastic net model, we used 10 different levels to tune `penalty` and `mixture`. The y-axis for the RMSE metric is quite small, which indicates that the resulting performance does not vary drastically across the models. The x-axis represents the penalty hyperparameter (set from 0 to 1) and the different-colored lines represent the values of mixture (set from 0 to 1). From the plot, it appears that a lower `penalty` value produces a lower RMSE, which means a better performance. This is because when the penalty becomes too large, the coefficients are reduced to small values which makes it difficult for the model to predict well. Overall, all the models including the model with zero percentage of lasso penalty, or the ridge regression model in the red line, had increasing RMSE values as `penalty` increased. However, the best model was the lasso regression denoted by the pink line with `mixture` $= 1$ and `penalty` $= 0.03$.

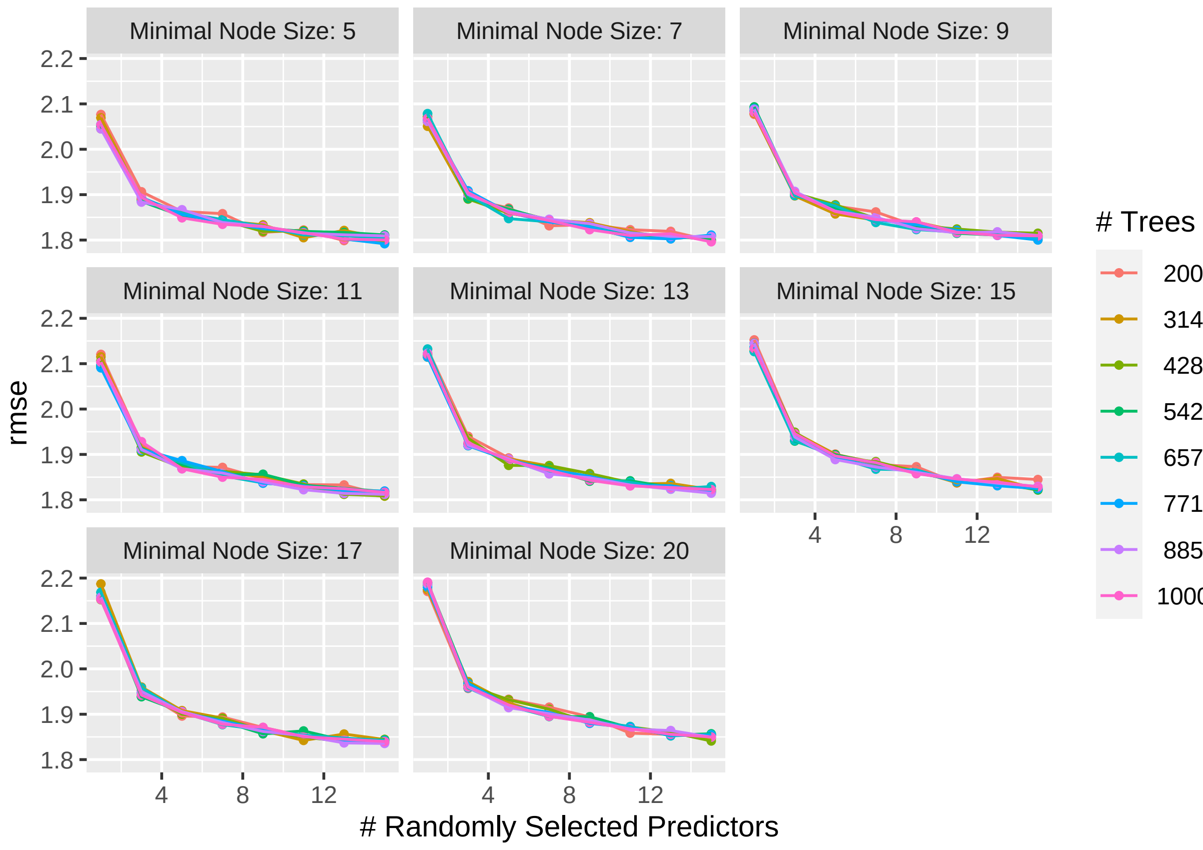
```
autoplot(knn_tune, metric = "rmse")
```



**K-Nearest Neighbors**

For the KNN model, we tuned the `neighbors` parameter for the range 1 to 15. We can see from the plot that generally, as the number of neighbors increases, the RMSE decreases. Our best number of neighbors value seems to be 11 since it minimizes the RMSE.

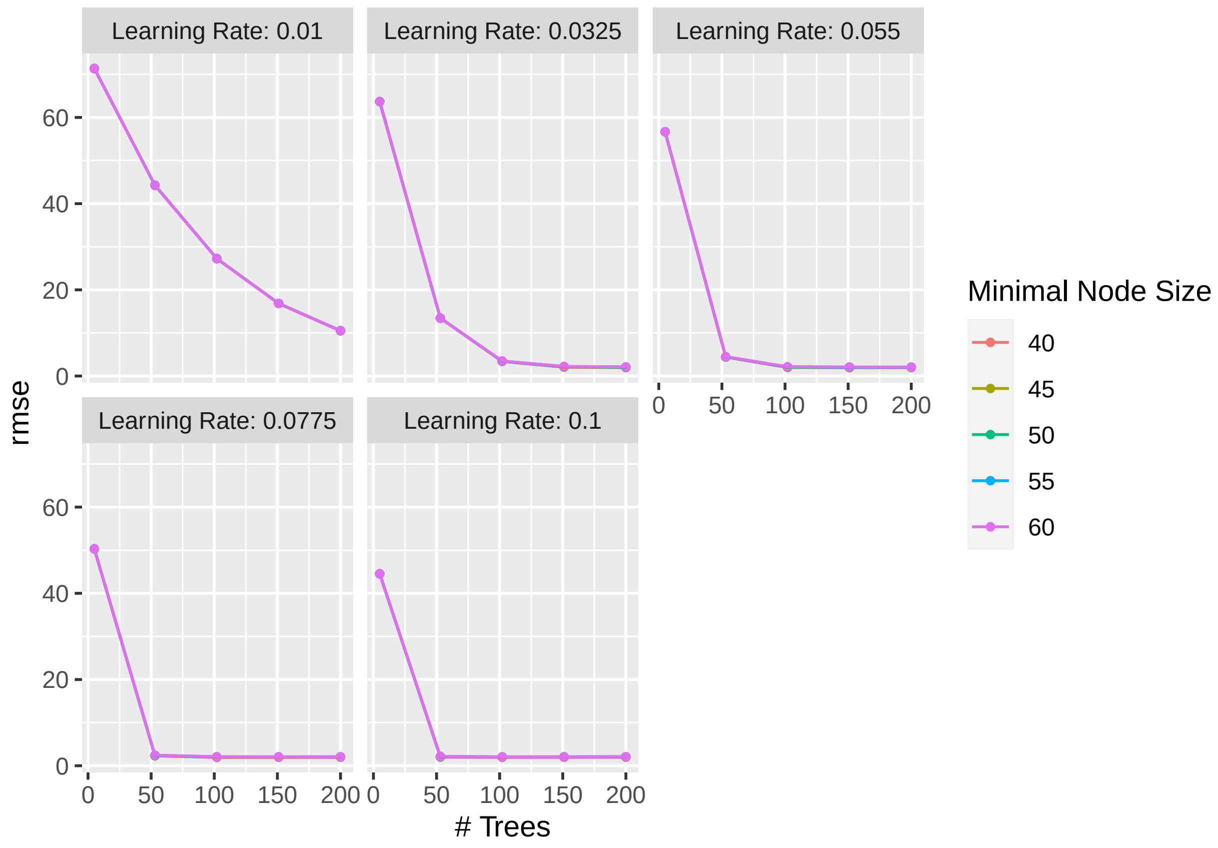
```
autoplot(rf_tune, metric = "rmse")
```



**Random Forest**

For the random forest model, the parameters we tuned were minimal node size, number of predictors, and number of trees. I chose to set the number of predictors for this model to go up to 15, which is about half of my total predictors in order to avoid a bagging model where trees may become dependent on one another. From the plot, it seems that `trees`, the number of trees for the model, does not make a huge impact on the performance. However, the minimal node size, or `min_n`, appears to have slightly lower RMSE values when it is smaller. The number of predictors also seems to have a pretty big impact on the performance as the plots show that a larger `mtry`, or more predictors, results in a lower RMSE value. Our best random forest model had parameters `mtry` = 15, `trees` = 428, and `min_n` = 9.

```
autoplot(boosted_tune, metric = "rmse")
```



### Boosted Trees

For the boosted trees model, we tuned learning rate, number of trees, and minimal node size with 5 different levels. The model seems to do better when the value of `learn_rate` is higher, or when the model is learning faster. Generally, an increase in `trees` also yielded lower RMSE values and better results. The minimal node size, `min_n`, seemed to not have an impact on the performance of the models. Our best boosted trees model with the lowest RMSE value was the model with `trees` = 151, `min_n` = 40, and `learn_rate` = 0.055.

### Results of Best Model

```
best_lasso
```

```
## # A tibble: 1 x 7
##   penalty .metric .estimator  mean     n std_err .config
##     <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1  0.0295 rmse    standard    1.70    10  0.0979 Preprocessor1_Model18
```

Our overall best model was the lasso regression model with tuned parameter `penalty` = 0.029 and a RMSE of 1.696.

**Fitting to Training Data** Using this best model, we will fit it to the entire training data. Once we have fit and trained the model on the training data, we can do the same for the testing data.

```
final_wf <- finalize_workflow(lasso_wflow, best_lasso)
final_fit <- fit(final_wf, ratings_train)
```

**Testing Best Model**

Now it is time to try our best model on the testing set, which has not been trained on before.

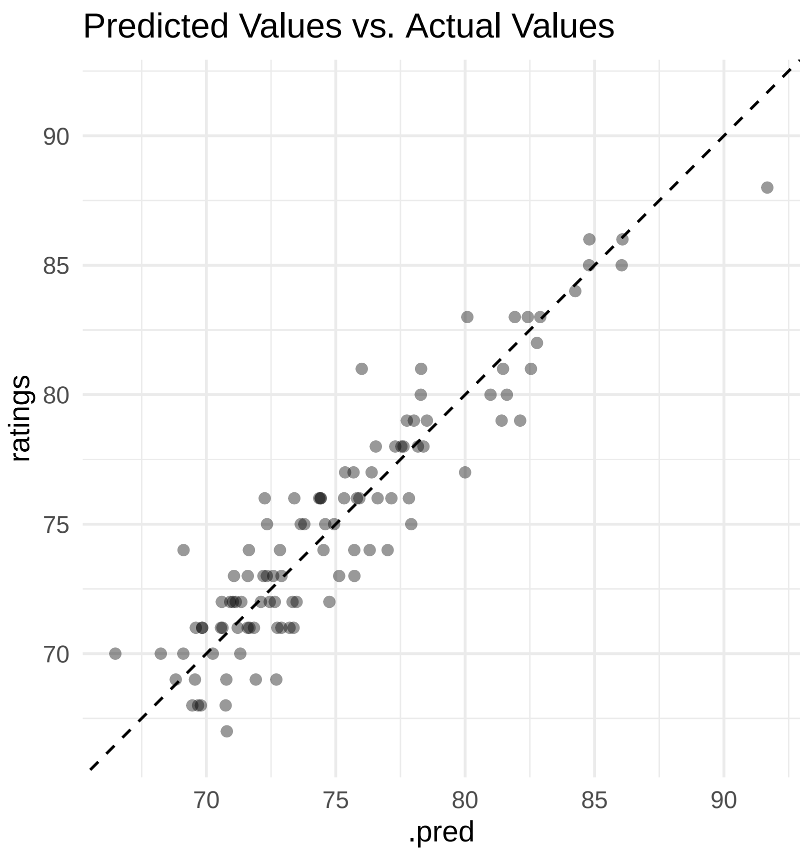
```r
train_rmse <- best_lasso$mean

test_rmse <- augment(final_fit, new_data = ratings_test) %>%
  rmse(truth = ratings, estimate = .pred)

rmse <- c(train_rmse, test_rmse$.estimate)
type <- c("Training", "Testing")
result <- tibble(RMSE = rmse, "Data Set" = type)
result
```

```
## # A tibble: 2 x 2
##    RMSE `Data Set`
##   <dbl> <chr>
## 1  1.70 Training
## 2  1.80 Testing
```

Our model performed slightly worse on the testing set than the training set with an RMSE of 1.804 compared to our training RMSE of 1.696. However, this is still a pretty decent outcome and our model did not perform too bad on the testing set.

Now let's plot predicted values with actual values to better visualize how our model did on the testing data.

```r
augment(final_fit, new_data = ratings_test) %>%
  ggplot(aes(x = .pred, y = ratings)) +
  geom_point(alpha = 0.4) +
  geom_abline(lty = 2) +
  theme_minimal() +
  coord_obs_pred() +
  labs(title = "Predicted Values vs. Actual Values")
```



The predicted observations denoted by each point does a pretty good job of following the straight line, which is what the plot would look like if all the points were predicted accurately. Overall, our model was able to predict ratings of players pretty closely to their true ratings with no impossible outcomes like negative ratings predictions.

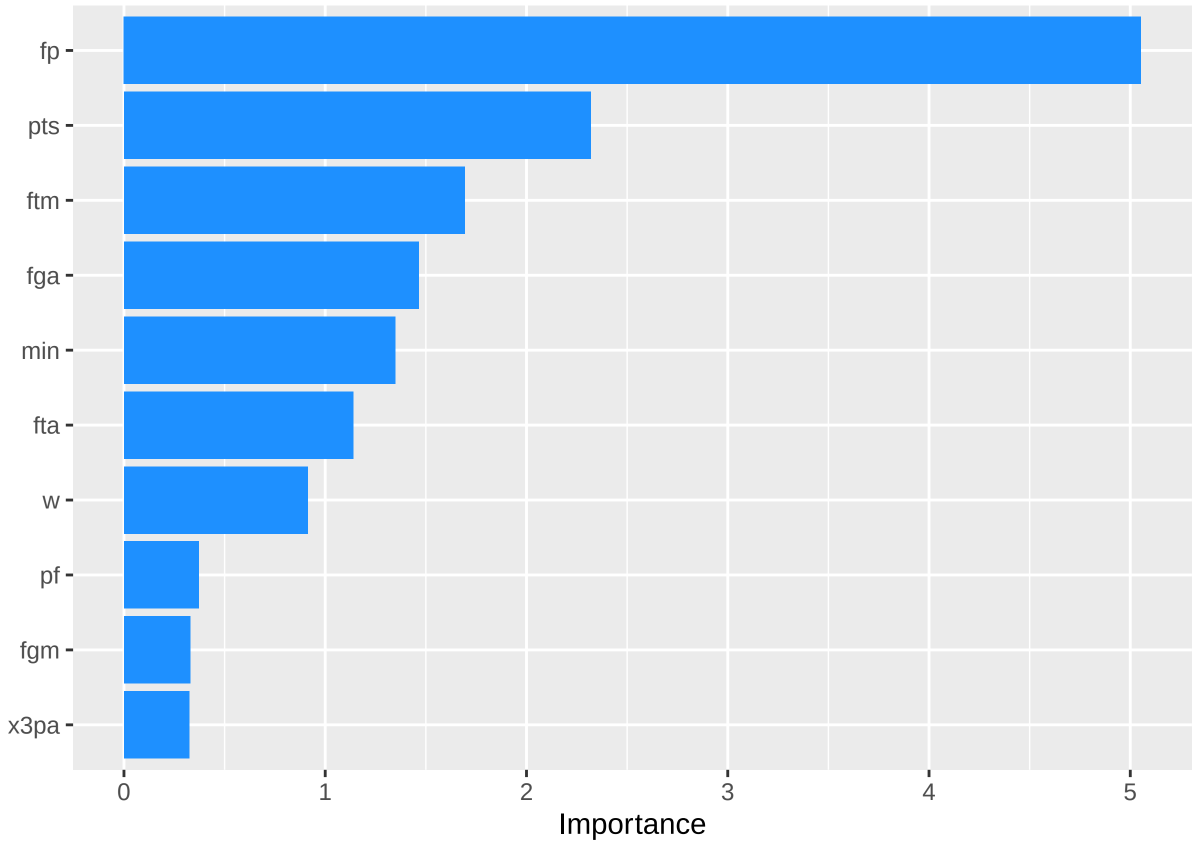
```
augment(final_fit, new_data = ratings_test) %>% select(player, ratings, .pred)
```

```
## # A tibble: 104 x 3
##    player           ratings .pred
##    <chr>              <int> <dbl>
##  1 Aaron Gordon          80  81.0
##  2 Aaron Holiday         76  75.8
##  3 Alec Burks            77  80.0
##  4 Amile Jefferson       69  68.8
##  5 Ante Zizic            73  71.1
##  6 Austin Rivers         75  74.9
##  7 Bam Adebayo           86  86.1
##  8 Bismack Biyombo       76  75.3
##  9 Blake Griffin         83  80.1
## 10 Bobby Portis          77  75.4
## # i 94 more rows
```

We can also take a look at the actual versus the predicted ratings of the players in the test set. The model does a pretty good job of making these predictions and is pretty accurate with only a few points off for some players.

**Variable Importance** The variable importance plot (VIP) allows us to visualize which variables are most relevant in predicting our outcome.

```
# Using the training fit to create the VIP because the model was not actually fit to the testing data
final_fit %>%
  extract_fit_engine() %>%
  vip(aesthetics = list(fill = "dodgerblue"))
```



We can immediately recognize that the variables `fp`, `pts`, `ftm`, `fga`, and `min` are among the top 5 most important variables when determining the prediction for `ratings`. It makes sense that `fp`, which represents fantasy points, is by far the most important variable because its calculation is actually based on other player statistics like points and field goal percentage. Additionally, offensive statistics like points, free throws made, and field goal attempts also make sense in increasing a player's rating as they generally mean the player is better. At first, `min`, or minutes played may seem like a surprising statistic that affects ratings, but it does make logical sense as better players tend to get to play more minutes.

### Conclusion

After testing out multiple models and conducting analysis, we can conclude that the best model to predict the rating of a player is the lasso regression model. The elastic net linear regression model also came very close and in fact, the best model that was chosen had a mixture equal to 1, which means it was also a lasso regression model. We obtained this best model by comparing the RMSE metric among all the models, and this model had the lowest RMSE. This is not that surprising because lasso regression is able to perform feature selection which means less relevant variables had their coefficients shrunk to 0 which allowed for better prediction performance.

The models I tested all had pretty close RMSE values. The only model that did not do as well as the rest was the K-Nearest Neighbors model. This makes sense since performance tends to decrease for KNN models when there are too many predictors. In a high dimensional data space, the data points are not close enough to each other for KNN to predict the outcome accurately.

I was quite surprised that my model did as well as it did, and I thought it was really interesting being able to compare specific players' actual ratings with their predicted ratings from my model. Some areas of improvement or next steps could be to introduce even more predictors such as a specific player's past ratings from previous years to test if the prediction performance is enhanced.

Overall, this project of predicting NBA player ratings using real statistics of players really allowed me to explore a topic I am interested in and build upon my machine learning skills and experience, especially in the sports analytics field.

## Sources

This data was taken from the Kaggle data set, "NBA 2K Ratings with Real NBA Stats, which was scraped by William Yu from https://hoopshype.com/ as well as the official https://stats.nba.com/players/ website.

Additional facts and definitions mentioned in this project were found on the following websites:

https://www.kaggle.com/code/willyiamyu/nba-2k-analysis

https://en.wikipedia.org/wiki/NBA_2K