Predicting the NBA MVP

Using Machine Learning Models to Predict NBA MVP Voting

AUTHOR
Jose Saavedra
UCSB Spring 2025

Introduction

This project aims to build a machine learning model that can predict what player is most probable to win the MVP award in any given season using game performance metrics such as points per game, assists per game,

rebounds per game, etc. The data set I will be using is a data set containing Player MVP voting data from the 1982-2022 NBA season. This data set includes player information for each season such as player name and team, performance metrics, and award share earned by a player in that season's MVP voting.

Motivation

The NBA has had a rich history of great players who have forever changed the game and left a lasting legacy still felt by many in the league today. These players are oftent times recognized for their skill and talent through the NBA's MVP award. Players compete all season for the chance to hoist the trophy at the end of the season, and many players do show out and in those years the MVP race can be very close. As the season progresses different players rotate in and out of MVP talks and the one player who voters deem to be consistently the best throughout the season will ultimately lift up the MVP trophy in February.

Being as basketball is now more than ever an entertainment industry, the NBA MVP award has seen some controversy in the last couple of years. Fans use terms like "voter fatigue" to describe how a player who has won the award multiple in time in the previous seasons may be at risk of losing their deserved trophy due to voters wanting to create new story lines for the market. Other things also affect voters choices, things such as a player 's behavior, player story lines, team story lines, suspensions, controversies, are to name a few. We've all heard the saying "ball don't lie" and that is the ultimate motivation for this project. I want to cut out the story lines, the voter fatigue, and anything else that would influence a voters choice and look at the raw stats to see if we can predict who was statistically, the most valuable player of the season.

Roadmap

We will first begin identifying all features of our data set. This includes things like the structure of the data, variables in the data set which we can use for modeling, and checking for any potential missing values. We will also try to reduce the overall size of our data set to help our models train faster and more efficiently. Next we will perform EDA on the data set using graphs and plots such as correlation plots, box plots, and scatter plots to try and identify any significant relationships in the data that may help us in our model fitting. After this we will finalize our formula using model

selection and begin to fit our models to training data using the tidymodels package. In this project our goal is to predict the outcome variable award_share, which is continuous, the player with the highest award_share value for the respective season gets the MVP award. After fitting multiple models to the training data we will then assess each models performance and choose the best performing model to then be used on our testing data set to see how well our model generalizes on unseen data. Lastly we will assess our chosen model's performance using various metrics and draw conclusions from our findings and put our model to the test using the newly available 2025 NBA season player data set.

Data Set

Loading Necessary Packages and Data Set

```
# Required packages and nba data
library(tidymodels)
library(tidyverse)
library(paletteer)
library(dplyr)
library(corrplot)
library(knitr)
library(ranger)
library(naniar)
library(xgboost)
library(vip)
options(scipen = 999)

# clearing global enviroment

rm(list = ls())

nba <- read_csv('/Users/paris/SPRING 25/PSTAT 131/PROJECT/
```

The previously mentioned data set was provided by user Robert Sunderhaft via Kaggle

Now I don't really have that much space in this document and we have 55 variables to work with (see attached codebook) . Given that I have good knowledge in both the NBA and in statistics I chose to remove a good

amount of the variables in this data set. There are a lot of variables that are direct results from others, such as orb_per_g and drb_per_g (offensive and defensive rebounds per game) would both make up the trb_per_g (rebounds per game) variable. Player position and age are also not included as these are have no effect on MVP voting. The following list is of all the variables I have chosen to be relevant to this analysis. Later on I will do model selection to finalize this list of variables to be used in fitting the model.

- **season:** indicates the basketball season of play. 1982 would mean the 1981-1982 basketball season
- **player:** name of player
- **g:** games played
- **gs:** games started
- **mp_per_g:** minutes per game
- **fg_per_g:** field goals per game
- **fga_per_g:** field goals attempted per game
- **fg_pct:** field goal percentage per game
- **efg_pct:** effective shooting percentage per game
- **ft_per_g:** free throws per game
- **fta_per_g:** free throws attempted per game
- **ft_pct:** free throw percentage per game
- **trb_per_g:** total rebounds per game
- **ast_per_g:** assists per game
- **stl_per_g:** steals per game
- **blk_per_g:** blocks per game
- **tov_per_g:** turnovers per game
- **pts_per_g:** points per game

- **mp:** minutes played total
- **per:** player efficiency rating
- **ts_pct:** true shooting percentage
- **usg_pct:** usage percentage
- **ows:** offensive win shares
- **dws:** defensive win shares
- **ws:** win shares
- **ws_per_48:** win shares per 48 games
- **vorp:** value over replacement
- **award_share:** MVP voting win share percentage
- **mov:** average margin of victory during the season
- **mov_adj:** average Margin of Victory adjusted for difficulty of opponent during season
- **win_loss_pct:** player's win loss percentage for the season
- **bpm:** box plus minus

The list of variables to use is now 32, which is still a lot. I aim to identify which variables are the biggest indicators for potential MVP voting in the next section.

```
# selecting relevant variables

nba <- nba %>%
  select(season, player, g, gs, fg_per_g, fga_per_g, fg_pc

# making season a factor variable

nba$season <- as.factor(nba$season)

# checking to see that all variables are captured

length(nba)
```

```
[1] 32
```

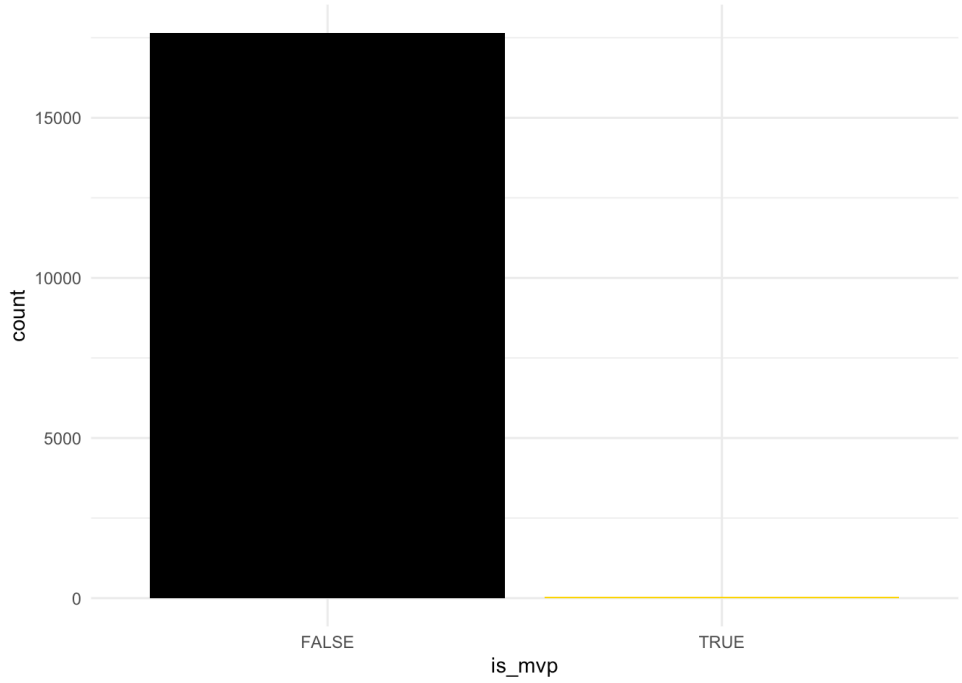
Exploratory Data Analysis

Before we begin predicting who will be the MVP, we need to do some EDA on the data. This will help us first by identifying any potential problems in our data like missing values, variables that need to be made factors before fitting a model, etc. EDA will also help us to identify trends in the data that may be useful in later model selection. Since we are predicting award share (which is a continuous variable), we are doing regression, but for later metrics i will be adding a variable to the data named “is_MVP” which will be a boolean value that is true when the player has the highest value for award_share in the respective season. Also to help us with visualizing the data later, I decided to add a column for players who received at least one vote.

```
# adding is_MVP variable to the nba data set

nba <- nba %>%
  group_by(season) %>%
  mutate(is_mvp = award_share == max(award_share, na.rm =
  mutate(mvp_vote = award_share > 0) %>%
  ungroup()

# value should be 41 represting the 41 players who earned

sum(nba$is_mvp)
```

```
[1] 41
```

```
# histogram of is_mvp variable

ggplot(nba, aes(x = is_mvp)) +
  geom_bar(fill = c("black", "gold")) +
  theme_minimal()
```



Now we see that in total only 41 players out of the over 17697 in the data set were MVP, some of them multiple times. Now 17697 is a lot of players. This data set includes players who may have played only 10 minutes in one season, or maybe even didn't get on the court once. We next do some data cleaning to reduce the total number of observations and remove observations that would otherwise be beneficial in our model fitting. I will provide accompanying justifications for removing the observations as well.

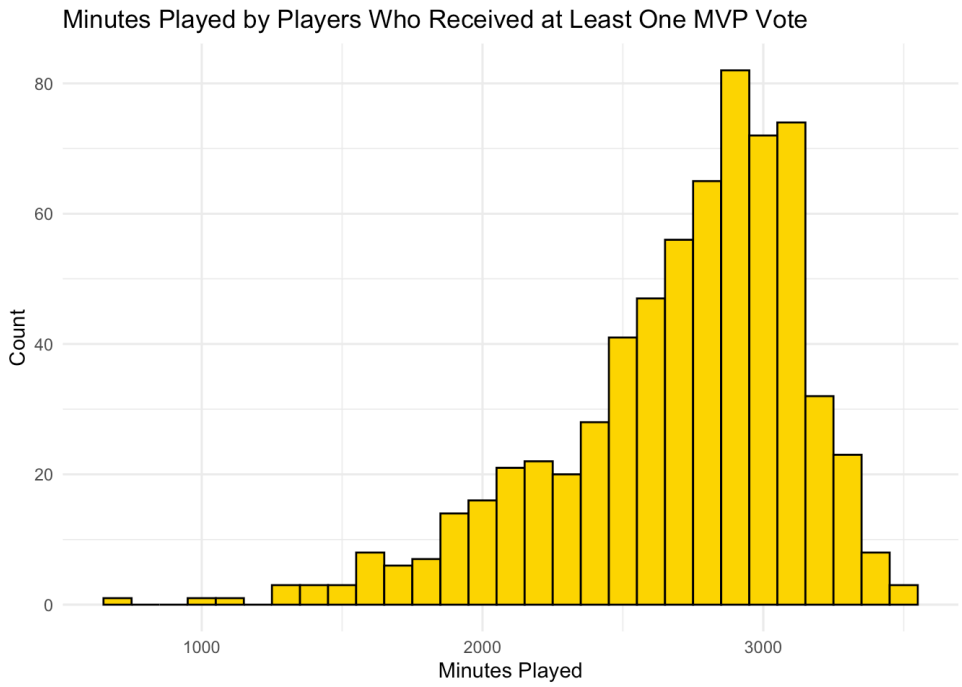
Since this is not a classification problem, we do not need to worry too much about possible class imbalances.

Data Cleaning

Now let's consider the fact that most players who would be considered for the MVP will be players who played a significant amount of minutes in their MVP caliber season. This means that most players who played little to no minutes will have almost no consideration for the award. We explore this more with this graph below.

```
nba %>%
  filter(mvp_vote == TRUE) %>%
ggplot(aes(x = mp)) +
  geom_histogram(binwidth = 100, fill = "gold", color = "b
  labs(title = "Minutes Played by Players Who Received at
```

```
       x = "Minutes Played",
       y = "Count") +
  theme_minimal()
```



These graphs shows the distribution for player who received at least one vote in their seasons MVP race, with frequency count on the y-axis, and the total minutes played on the x-axis.

We see that in general to be even considered you have to play more than 1500 minutes. We see some outliers in this data and should investigate them further.

```
nba %>%
  select(player, season, mp, award_share) %>%
  filter(award_share > 0) %>%
  arrange(mp, descending = TRUE) %>%
  head() %>%
  kable()
```

player	season	mp	award_share
Michael Jordan	1995	668	0.011
Magic Johnson	1996	958	0.007
Shaquille O'Neal	2007	1135	0.002

Derrick Rose	2021	1279	0.010
Steve Smith	1999	1314	0.001
Arvydas Sabonis	1999	1349	0.003

Now we see two very interesting observations, those of Micheal Jordan in 1995, and Magic Johnson in 1996. These two observations are actually similar as for both players, these were the seasons in which they had returned from a hiatus from basketball. Micheal Jordan had left to play baseball, and Magic Johnson has returned from a hiatus after being diagnosed with AIDs. Now these observations unfortunately will not help our data and could even potentially effect our models performance given their outlier status, my feeling is that these votes were given to them partly due to their name and their story line. For this reason I chose to remove these two from the model. As for the rest of the observations I will introduce a cutoff for minutes played. I choose this value to be 3 standard deviations away from the mean minutes played by players who received an MVP vote. Let's call Giannis Antetokounmpo for some help.

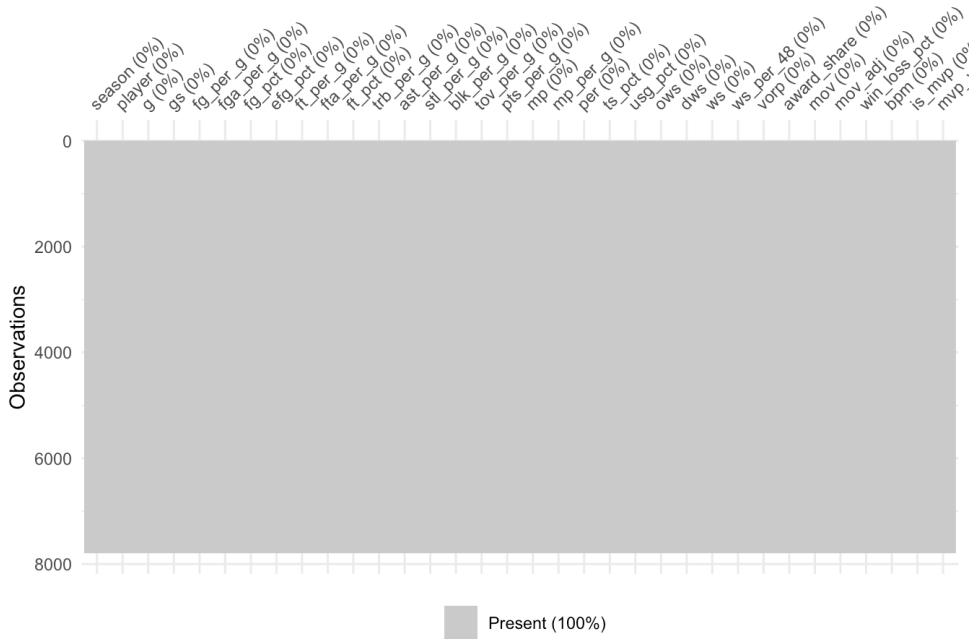

```
nba <- nba %>%
  filter(mp > mean(mp[award_share > 0]) - 3*sd(mp[award_sh

count(nba) %>%
  kable()
```

n
7794

Nice block Giannis! We now reduced our data by a significant amount. Now let's take care of the any potential missing values in our data.

Missing Values

```
vis_miss(nba)
```



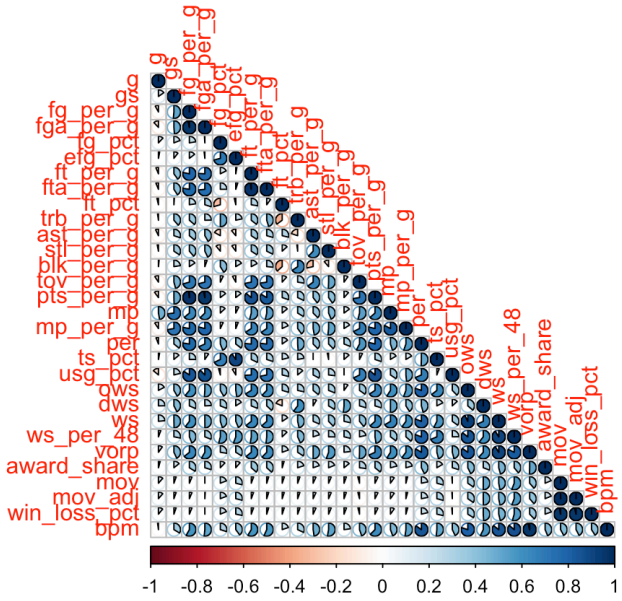
This is great news! We have no missing values and do not have to worry about filling anything in. Let's move on to visual EDA

Variable Correlation Plot

We use a correlation matrix to try and identify any variables that have strong correlation with our outcome variable. We can also use this to identify variables that may have strong correlation with each other which I do expect to happen as many of the variables are direct results from others.

```
nba_numeric <- nba %>%
  select_if(is.numeric)
```

```
cor(nba_numeric) %>%
  corrplot(type = "lower", method = "pie")
```



Now this is hard to read but don't worry, since we are mostly interested in the award_share variable we can make a table for the it's correlation with all other numeric variables.

```
corr <- cor(nba_numeric)

corr[,"award_share"] %>%
  kable()
```

	x
g	0.0430856
gs	0.1435014
fg_per_g	0.3388083
fga_per_g	0.2971008
fg_pct	0.1218942
efg_pct	0.1122703
ft_per_g	0.3632859
fta_per_g	0.3668066

ft_pct	0.0480116
trb_per_g	0.1984908
ast_per_g	0.1978637
stl_per_g	0.1869070
blk_per_g	0.1412835
tov_per_g	0.2694247
pts_per_g	0.3582367
mp	0.2102810
mp_per_g	0.2077056
per	0.4257460
ts_pct	0.1650090
usg_pct	0.2957327
ows	0.4262794
dws	0.3073428
ws	0.4565955
ws_per_48	0.3902744
vorp	0.5037225
award_share	1.0000000
mov	0.1860670
mov_adj	0.1861582
win_loss_pct	0.1945276
bpm	0.4375412

In general the more a player has started games the higher their potential MVP voting share is. This is to say that most MVP caliber players are part of the starting lineup on their respective team. Next we see that among all in-game performance metrics, field goals per game was the most correlated with award_share, the rest of them (points, assists, rebounds, etc.) are correlated but not too highly, still I choose to keep them in the model. Variables that are the most correlated with award_share are those which are measurements of a players total value to a team. Notably the variables for win shares and player efficiency. Win shares are estimates of how many wins a player is responsible for on their team. Player efficiency is another metric that represents a rating of a players overall performance, calculated using both positive and negative contributions to their team. Player per

attempts to show a players contribution to their team's success, similar to win shares. Among all variables vorp shares the highest correlation with award_share. VORP stands for Value Over Replacement Player, which is a metric that estimates how much a player contributes to their team, compared to the contribution a hypothetical replacement player would provide.

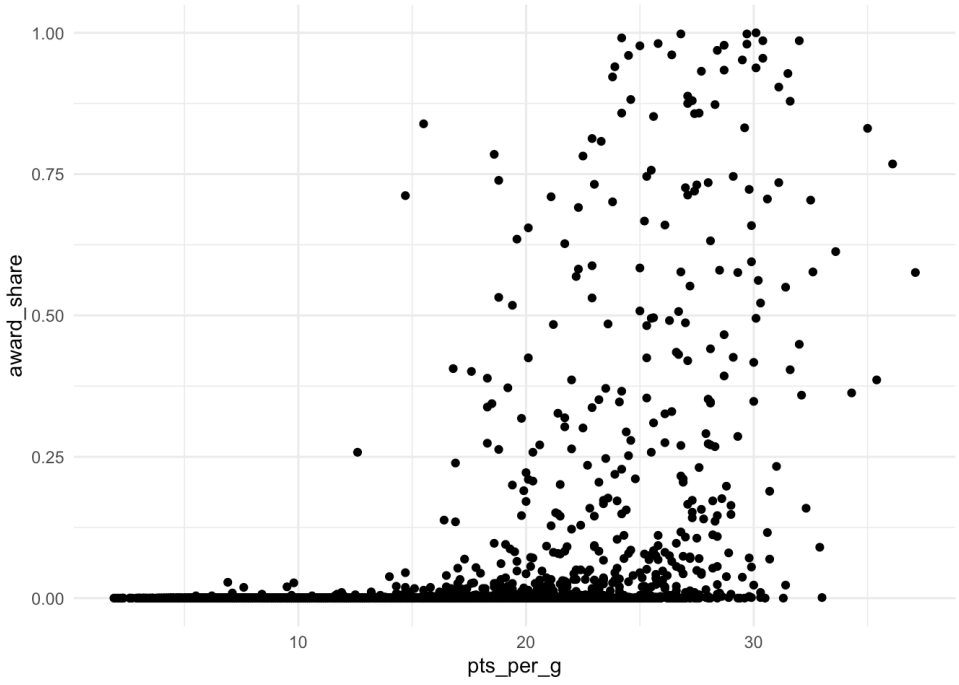
It is important to note that almost all variables do have correlations with award_share that would be considered weak or moderate (values around 0.10-0.50).

Visual EDA

Points Per Game

Now i think most of us would think that the player getting the most buckets is the most likely to win the MVP. After all the main objective is to outscore your opponent, let's visualize the relationship between points per game and award share using a scatter plot.

```
ggplot(nba, aes(x = pts_per_g, y = award_share)) +
  geom_point() +
  theme_minimal()
```

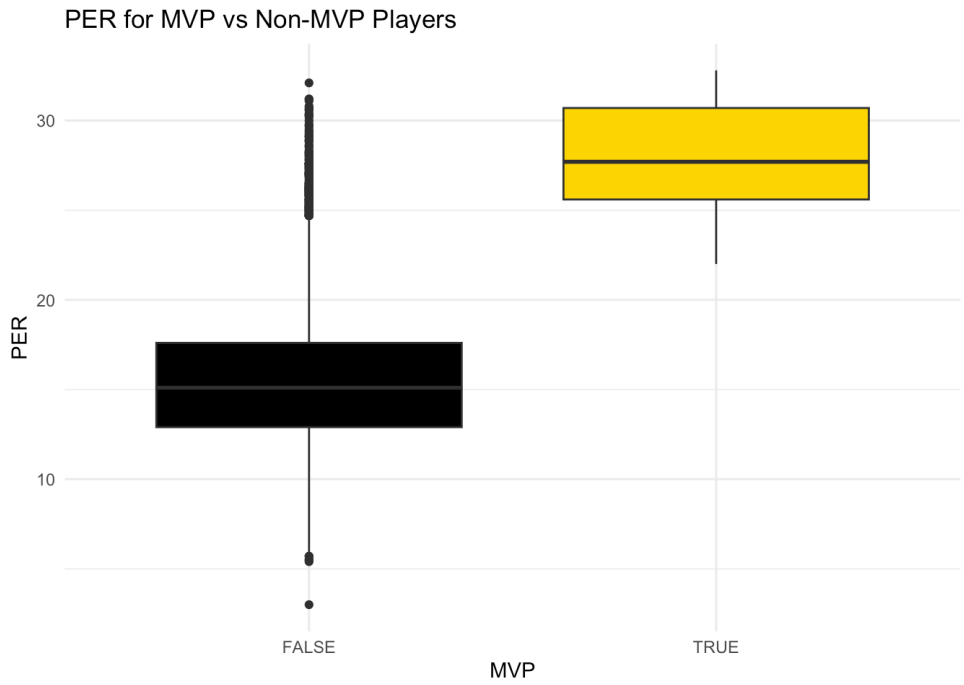


We see that the scatter plot is pretty spread out, meaning that maybe

scoring the most points isn't the only metric that matters in MVP voting. Additionally I believe this can be explained as there might be some observations where a player scored a lot of points in a few games and their average was very high but they did not play enough games to be considered for MVP. This graph shows us that points are important, but it is better to look at a player's total in-game contribution.

PER (Player Efficiency Rating)

```
ggplot(nba, aes(x = is_mvp, y = per, fill = mvp_vote)) +
  geom_boxplot(fill = c("black", "gold")) +
  labs(title = "PER for MVP vs Non-MVP Players", x = "MVP"
  theme_minimal()
```

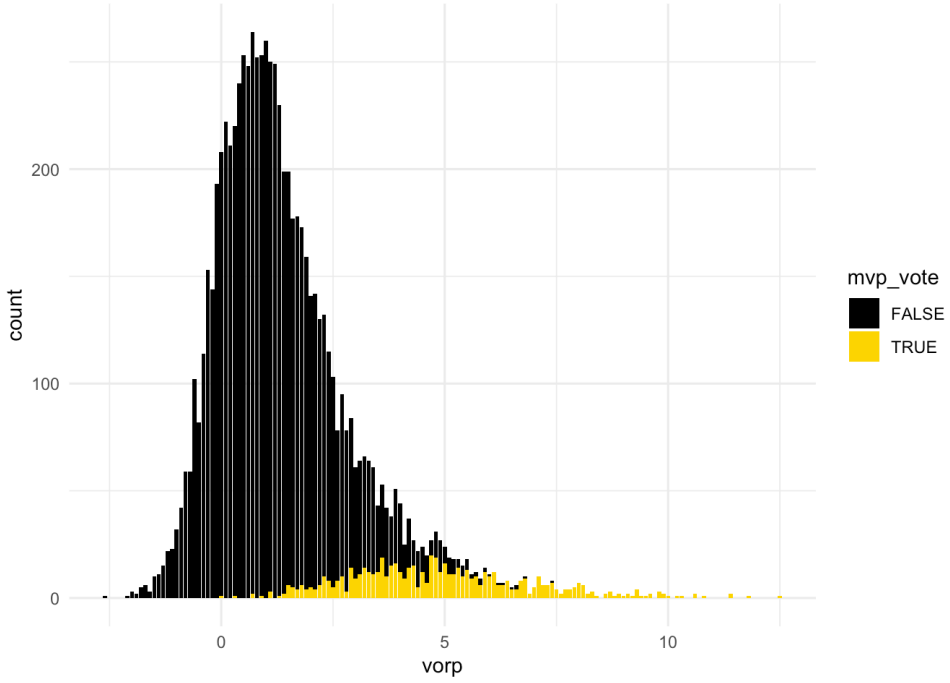


We see that on average MVPs are ones with very high player efficiency scores, though there is a good amount of observations in the non MVP group that reach the same per scores as the MVP group. This makes sense as we know that there is only one winner and there are also many other players who played well in any given season.

VORP

Value over replacement player, as previously mentioned, is a metric that measures how much a player contributes, compared to a hypothetical role player.

```
ggplot(nba, aes(vorp)) +
  geom_bar(aes(fill = mvp_vote)) +
  scale_fill_manual(values = c("black", "gold")) +
  theme_minimal()
```

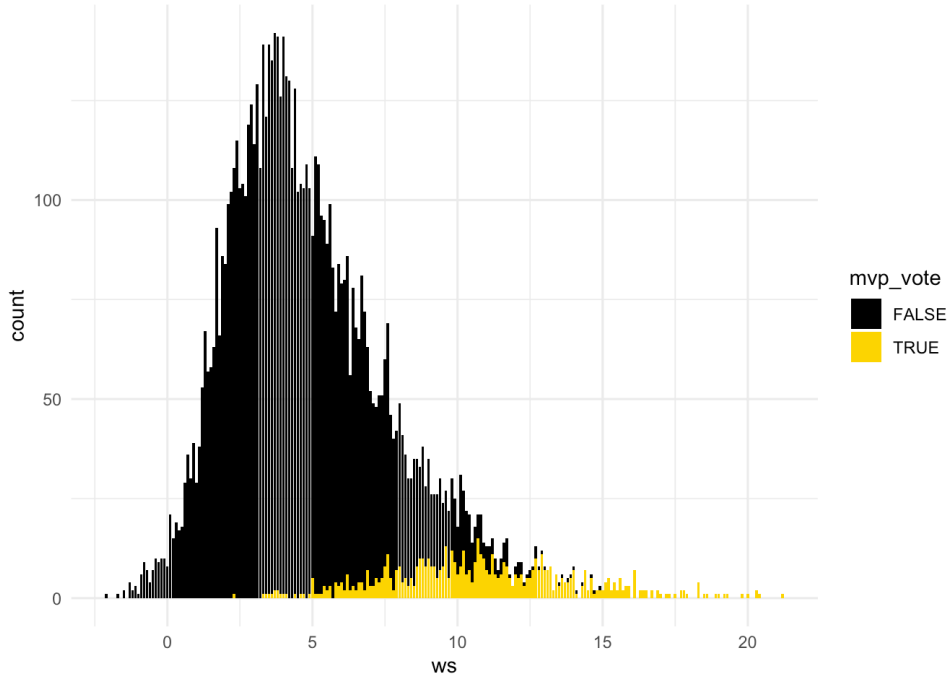


We see that in general, the higher VORP value, the more votes we see for a player in that years MVP voting.

Win Shares

Win Shares is an advanced basketball statistic that attempts to measure a player's total contribution to their team's success, assigning credit for team wins based on individual performance metrics. In the histogram above, we visualize the distribution of Win Shares (`ws`) for players, with bars colored by whether the player received MVP votes (`TRUE` in yellow, `FALSE` in black).

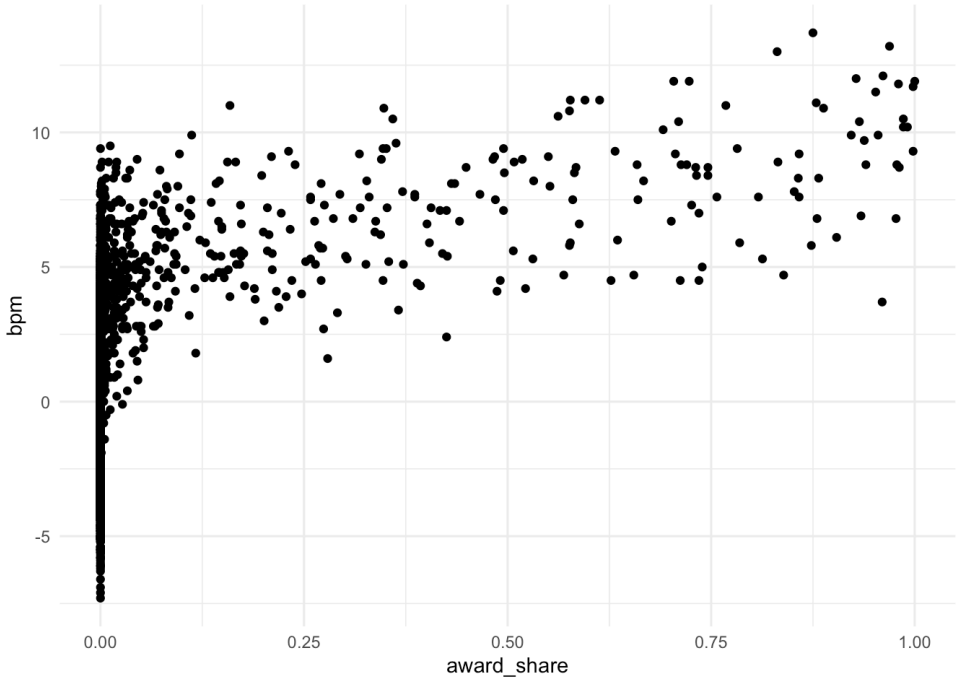
```
ggplot(nba, aes(ws)) +
  geom_bar(aes(fill = mvp_vote)) +
  scale_fill_manual(values = c("black", "gold")) +
  theme_minimal()
```



From the plot, we can see that while most players cluster around moderate Win Share values (between 2 and 10), those who received MVP votes tend to have significantly higher values. In particular, MVP vote-getters are concentrated toward the right tail of the distribution, suggesting that higher Win Shares are strongly associated with a greater likelihood of MVP consideration. This indicates Win Shares is likely a meaningful predictor in modeling MVP voting outcomes.

Box Plus Minus

```
ggplot(nba, aes(x = award_share, y = bpm)) +
  geom_point() +
  theme_minimal()
```



We see a pretty clear relationship between box plus minus and sward share. That is, we observe that among players who received mvp votes, they all had high values of box plus minus, with very few having negative values and a majority of observations that have high award shares being over a box plus minus score of 5. This means that box plus minus will be very helpful in predicting our outcome variable award_share.

Building the Models

Now its time to build our models, but first we have to do some preliminary work such as splitting our data and setting up our recipe and workflows.

Model Selection

Now we can choose our final list of variables to be included in the model. Season will not matter and we will simply assume that any specific era does not effect a players chances to win. This may be an issue as different eras often show different trends in in-game stats, anything like claims that more points are scored in the current era as compared to previous eras. Other variables I will not include are ones that measure percentages of baskets. This includes metrics like field goal percentage. We do this since these are metrics are not highly correlated with our outcome and are also correlated with other variables that may capture their contribution better

(for example field goal percentage and field goals per game). We will keep all core stats (points, assists, rebounds, turnovers, etc.) and true shooting percentage and drop efg_pct and fg_pct, since ts_pct is the most informative shooting metric. Usage percentage is another variable we will keep as it is unique and not correlated with any other variables. For general player stats we will keep per, ws_per_48 (this is more informative than the regular ws variable), and vorp and remove their specialized versions (ows and dws). Lastly we will remove mov and instead keep mov_adj as it is more informative and keep win_loss_pct and award_share (this one is the most important one). Now that we finalized our variable selection we can get into setting up our recipe

Alright chill Bron, we still have a long way to go.

Splitting the Data

Now we first need to split our data into a training and testing set. We do this to later measure our models performance. Essentially we train it on the training data set and then use the model on the testing data set to see how well our model generalizes on unseen data. We stratify on the outcome variable award_share, we do this to ensure that both the training and testing sets have a similar distribution of award outcomes. This helps prevent any imbalance such as one set having mostly high or low award shares that could lead to misleading model performance or biased results.

We also used a technique called cross validation, specifically k-fold cross validation, to get a more accurate estimate of model performance. In this method, the data is split into k smaller parts (“folds”). The model is trained

on k-1 folds and tested on the remaining one. This process is repeated k times so each part gets used as a test set once. The final performance is averaged across all k runs, giving us a more stable estimate than just a single train/test split.

```
set.seed(2013) # for replicability

nba_split <- initial_split(nba, prop = 0.80, strata = awar


nba_train <- training(nba_split)


nba_test <- testing(nba_split)

nba_folds <- vfold_cv(nba_train, v = 5, strata = award_sha
```

Recipe Building

In order to train our models we first need to make our recipe for our workflows. Since we are trying to predict x, we use all other variables to do this. We do this using tidymodels which includes a recipe function which we can use to set up our equation.

Our equation will be set up to where we predict the outcome variable award_share calculated as some function using the rest of the variables we chose. We normalize all predictors to have mean 0 and standard deviation 1. All variables are numeric so we do not have to worry about any transformations.

```
# setting up our recipe

 nba_recipe <- nba %>%
    recipe(award_share ~ gs + mp_per_g + fg_per_g + fga_pe
             pts_per_g + trb_per_g + ast_per_g + stl_per_g
           step_normalize(all_predictors())

 # prepping and baking our recipe

 prep(nba_recipe) %>%
   bake(new_data = nba_train)
```

```
# A tibble: 6,235 × 21
```

```
         gs mp_per_g fg_per_g fga_per_g ft_per_g
fta_per_g pts_per_g trb_per_g
      <dbl>    <dbl>    <dbl>     <dbl>    <dbl>
<dbl>     <dbl>     <dbl>
 1  0.940      1.00     2.45     1.55     1.03      1.31
1.95      1.34
 2 -0.00437    0.440   -0.232   -0.253    0.642     0.686
-0.105    -1.31
 3 -1.93      -1.75    -0.777   -0.976   -0.470    -0.202
-0.834    -0.0256
 4  1.05       1.36     1.46     1.41     0.315     0.164
1.03      1.07
 5 -1.78      -1.37    -0.529   -0.590   -0.470    -0.306
-0.651    -0.0256
 6  0.789      0.611    0.215    0.0124  -0.535    -0.620
-0.123     1.34
 7 -1.33      -0.894    0.215   -0.0599   0.0536
0.0591    0.0228   -0.784
 8 -1.82      -1.59    -1.08    -0.879   -0.666    -0.724
-0.998    -1.20
 9 -0.231      0.491    1.46     1.43     0.250     0.164
1.03     -1.12
10  0.109     -0.774   -1.12    -1.10    -0.600    -0.672
-1.13     -0.405
# ℹ 6,225 more rows
# ℹ 13 more variables: ast_per_g <dbl>, stl_per_g <dbl>,
blk_per_g <dbl>,
#   tov_per_g <dbl>, per <dbl>, ws_per_48 <dbl>, vorp
<dbl>, ts_pct <dbl>,
#   usg_pct <dbl>, mov_adj <dbl>, win_loss_pct <dbl>, bpm
<dbl>,
#   award_share <dbl>
```

Methods

Since we are doing a predicting a continuous variable we will be using regression methods to predict our outcome.

I will implement 4 models to be explained in the following sections

In general all models will follow the same working steps. We first define the model and it's type, in this case we will be choosing the mode to be regression. Next we set up a workflow and add our model and recipe we defined earlier. Some models will have parameters which we can tune in

order to find the best possible values, we do this by defining a tuning grid with reasonable ranges for the parameters, we can later check which set of parameters performed the best. We then tune the models and specify the workflow, k-fold cross validation folds, and the tuning grid for our chosen parameters to tune. Lastly we collect RMSE values and see which model performed the best

We will choose the best model by looking at the metric rmse, which is a metric that measures the average magnitude of the error between predicted and actual values in a model, calculated as the square root of the average of squared differences. The lower the value, the better the model performs, we usually aim for values close to 0.

Simple Linear Regression:

A basic model that fits a straight line to the data by modeling the relationship between one or more features and the outcome as a linear function.

- **No major tuning parameters**, but assumes a linear relationship and can struggle with complex patterns.

```
set.seed(2013)
# linear regression

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

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

lm_results <- fit_resamples(lm_wflow, resamples = nba_fold

collect_metrics(lm_results) %>%
  kable
```

.metric	.estimator	mean	n	std_err	.config
rmse	standard	0.0698402	5	0.0030238	Preprocessor1_Model1

We see that simple linear regression gives us an average rmse of 0.0698, very close to 0.07, this is good on it's own, but let's compare it to the other

models average rmse first

K-Nearest Neighbors:

Predicts the outcome for a data point by looking at the *k* closest points in the training set. The prediction is the average (for regression) or most common class (for classification) among those neighbors.

- **neighbors:** Controls how many nearby points influence the prediction. A small neighbors can lead to overfitting, while a large neighbors may smooth out important patterns.

```
set.seed(2013)
# k-nearest neighbors

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

kknn_wflow <- workflow() %>%
              add_model(kknn_model) %>%
              add_recipe(nba_recipe)

neighbors_grid <- grid_regular(neighbors(range = c(1, 20)))

tune_res_kknn <- tune_grid(
  object = kknn_wflow,
  resamples = nba_folds,
  grid = neighbors_grid,
  metrics = metric_set(rmse)
)

show_best(tune_res_kknn, metric = "rmse") %>%
  kable()
```

neighbors	.metric	.estimator	mean	n	std_err	.config
20	rmse	standard	0.0507618	5	0.0026685	Preprocessor1_Model20
19	rmse	standard	0.0507921	5	0.0026703	Preprocessor1_Model19
18	rmse	standard	0.0508401	5	0.0026715	Preprocessor1_Model18
17	rmse	standard	0.0509014	5	0.0026685	Preprocessor1_Model17
16	rmse	standard	0.0509578	5	0.0026519	Preprocessor1_Model16

We see that in general, the k-nearest neighbors model is giving us a mean rmse values around 0.05, which is good and better than our simple linear regression model.

Elastic Net

A linear model that combines Lasso and Ridge regression to improve prediction and handle multicollinearity by penalizing large coefficients.

- **Mixture:** Controls the mix between Lasso (`alpha = 1`) and Ridge (`alpha = 0`).
- **Penalty:** Controls how strongly the model penalizes large coefficients to avoid overfitting.

```
set.seed(2013)

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

elasNet_wflow <- workflow() %>%
                 add_model(elastic_net) %>%
                 add_recipe(nba_recipe)

pandm_grid <- grid_regular(penalty(), mixture(range = c(0,

tune_res_elas <- tune_grid(
  object = elasNet_wflow,
  resamples = nba_folds,
  grid = pandm_grid,
  metrics = metric_set(rmse)
)

show_best(tune_res_elas, metric = "rmse") %>%
  kable()
```

penalty	mixture	.metric	.estimator	mean	n	std_err	.config
0	0.2413793	rmse	standard	0.0698295	5	0.0030291	Preprocessor1_M
0	0.2413793	rmse	standard	0.0698295	5	0.0030291	Preprocessor1_M
0	0.2413793	rmse	standard	0.0698295	5	0.0030291	Preprocessor1_M
0	0.2413793	rmse	standard	0.0698295	5	0.0030291	Preprocessor1_M

0	0.2413793	rmse	standard	0.0698295	5	0.0030291	Preprocessor1_M

We see that in general, Elastic Net also gives us average rmse values near 0.07, which again is not bad, just compared to the result from k-nearest neighbors, this wouldn't be our first choice when it comes to final model selection

Random Forest

An ensemble model that builds many decision trees and averages their predictions to reduce overfitting and improve accuracy.

- **trees (number of trees):** More trees can improve performance but take longer to train.
- **min_n:** Sets the minimum number of samples required at a leaf node, controlling complexity.
- **mtry:** Limits the number of features considered when splitting, adding randomness and improving generalization. Since we have 20 predictors, the valid range for this is 1-20

```
set.seed(2013)
# random forest model

rf_model <- rand_forest(mtry = tune(), trees = tune(), mi
            set_engine("ranger", importance = "impurity")
            set_mode("regression")

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

rf_params <- parameters(
  mtry(range = c(1, 20) ),  # finalizes mtry based on # o
  trees(range = c(50, 200)),  # range for number of trees
  min_n(range = c(5, 20))       # minimum samples in a no
)

# Set levels to 5 for the grid

rf_grid <- grid_regular(rf_params, levels = 5)

# tuning
```

```
tune_rf_res <- tune_grid(
  object = rf_wflow,
  resamples = nba_folds,
  grid = rf_grid,
  metrics = metric_set(rmse)
)


show_best(tune_rf_res, metric = "rmse") %>%
  kable()
```

mtry	trees	min_n	.metric	.estimator	mean	n	std_err	.config
5	87	8	rmse	standard	0.0478313	5	0.0025892	Preprocessor1_
5	162	8	rmse	standard	0.0478909	5	0.0028464	Preprocessor1_
5	200	8	rmse	standard	0.0481218	5	0.0027026	Preprocessor1_
5	87	5	rmse	standard	0.0481529	5	0.0024308	Preprocessor1_
5	200	5	rmse	standard	0.0482630	5	0.0028668	Preprocessor1_

We see that random forest has the lowest mean rmse, score out of all models, around 0.0478. This means that we can choose this model as it performed best on the cross validation data.

Results for Best Model

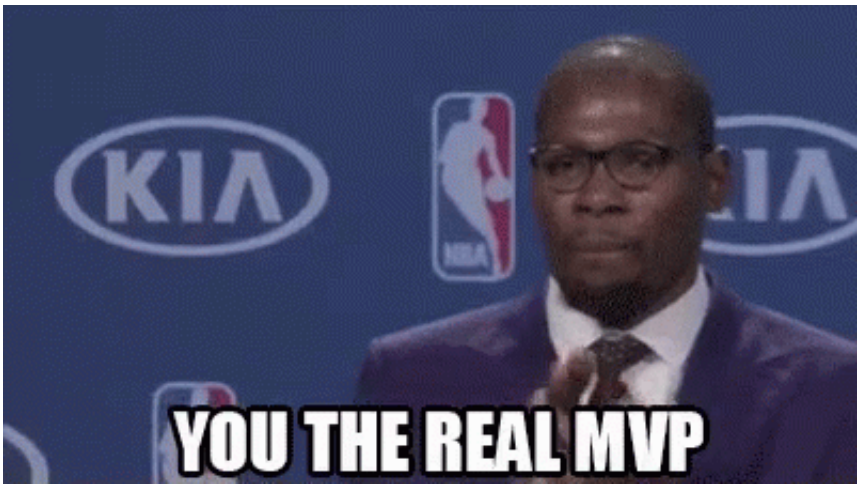
So since random forest performed the best lets now check what parameters it chose to for the best random forest model

```
tune_rf_res %>%
  collect_metrics() %>%
  arrange(mean) %>%
  head(1) %>%
  kable()
```

mtry	trees	min_n	.metric	.estimator	mean	n	std_err	.config
5	87	8	rmse	standard	0.0478313	5	0.0025892	Preprocessor1_

So now we see the value for our best random forest model and the parameters it chose. Random Forest #32 with 5 predictors, 87 trees, and a minimal node size of 8 did the best with a mean rmse value of 0.04783. #32

the real MVP



Fitting best model to training data

Now that we have our best performing model, lets fit it to the training set

```
best_rf <- select_best(tune_rf_res, metric = 'rmse')
rf_final_wflow <- finalize_workflow(rf_wflow, best_rf)
rf_final_fit <- fit(rf_final_wflow, data = nba_train)
```

Now that we fit it to the training data, and effectively trained our model using it, let's fit it to the testing data and check our results

Testing the Random Forest Model

Let's test out our random forest model on the nba_test data set

```
rf_predictions <- predict(rf_final_fit, new_data = nba_tes

nba_test_pred <- bind_cols(rf_predictions, nba_test)

nba_test_pred %>%
  metrics(truth = award_share, estimate = .pred) %>%
  filter(.metric == "rmse") %>%
  kable()
```

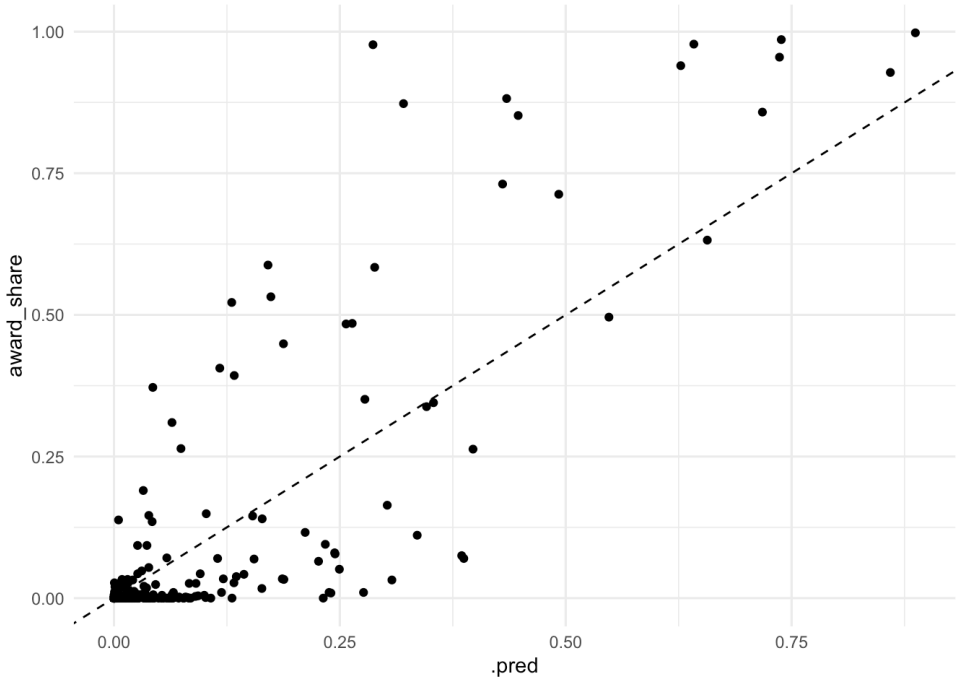
.metric	.estimator	.estimate
rmse	standard	0.0497831

Using our model on the testing data led to an rmse score of 0.05, which is

very good!

Lets plot predicted vs actual to see how well it did truly

```
nba_test_pred %>%
  ggplot(aes(x = .pred, y = award_share)) +
  geom_point() +
  geom_abline(lty = 2) +
  theme_minimal()
```

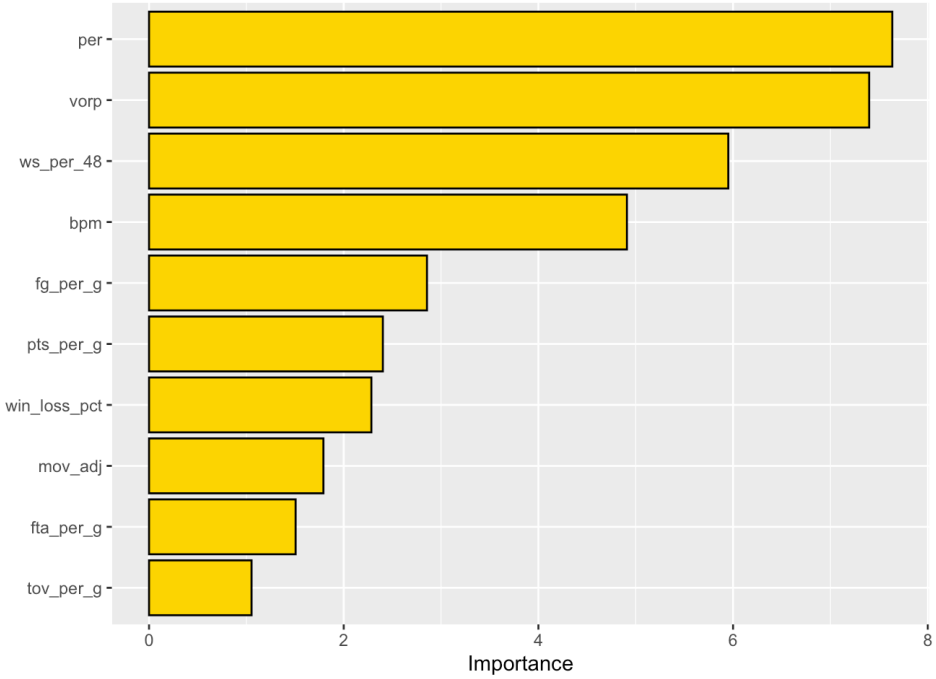


Well that doesn't look too good. We observe that in general, this model performs well for award share values close to 0. Once we start getting to higher award_share values, the model seems to begin to struggle a bit. It performs well in the middle ranged of award_share. This makes sense as the data did contain a lot of 0 values for award share, players who received no vote, so the model might be more inclined to predict closer to 0. This model can definitely be improved but for our purposes, this should do just fine.

We can also check which variables were most important in predicting the outcome and maybe to solidify our belief that advanced metrics are more telling of a player's MVP contention, and because it's fun to look at.

```
rf_final_fit %>%
  extract_fit_engine() %>%
```

```
vip(aesthetics = list(fill = "gold", color = "black"))
```



It seems the importance generally follows the results we saw in the correlation matrix, vorp was the most important, with player efficiency rating close behind. What surprised me the most is just how little core player statistics influenced the models predictions.

Predicting the 2025 NBA MVP

Now that we found the best model and assessed it to make sure it generalizes well on unseen data, lets put it to use. The 2025 season has just recently wrapped up and Shai-Gilgeous Alexander was named the 2025 season MVP. However this year's award came with some controversy. 3-time winner Nikola Jokic also had an amazing season, he even even had a higher player efficiency rating than SGA. Some argued that Jokic had the better season but in the end the voters decided that SGA was the one who would receive the trophy. Now many people feel like SGA's award came as a result of voter fatigue, as Jokic had won 3 times in the last 5 years. Let's put our model to the test and see if we can correctly predict the MVP of this NBA season, SGA,

The data used was pulled from basketball-reference.com and includes 2025 player statistics, including in-game metrics as well as advanced

player stats. As for mov_adj and win_loss_pct, these are team stats and were pulled from online resources and pushed into a tibble to later be joined with the other data sets.

```
# game metrics and advanced stats

nba_2025 <- read_csv("/Users/paris/SPRING 25/PSTAT 131/PR(
```

```
Rows: 575 Columns: 32
── Column specification ────────────────────────────────────────
Delimiter: ","
chr  (4): player, team, pos, Player-additional
dbl (27): rank, age, g, gs, mp_per_g, fg_per_g,
fga_per_g, fg_pct, fg3_per_g...
lgl  (1): award_share

ℹ Use `spec()` to retrieve the full column specification
for this data.
ℹ Specify the column types or set `show_col_types =
FALSE` to quiet this message.
```

```
advanced_2025 <- read_csv("/Users/paris/SPRING 25/PSTAT 13
```

```
Rows: 569 Columns: 30
── Column specification ────────────────────────────────────────
Delimiter: ","
chr  (4): player, Team, Pos, Player-additional
dbl (25): Rk, age, G, GS, MP, per, ts_pct, 3PAr, FTr,
ORB%, DRB%, TRB%, AST%...
lgl  (1): award_share

ℹ Use `spec()` to retrieve the full column specification
for this data.
ℹ Specify the column types or set `show_col_types =
FALSE` to quiet this message.
```

```
# team stats (mov_adj and win_loss_pct)

nba_2025_stats <- tribble(
  ~team, ~mov_adj, ~win_loss_pct,
  "OKC",     12.2, 0.808,
```

```
  "CLE",      9.7, 0.758,
  "BOS",      8.7, 0.720,
  "IND",      2.5, 0.630,
  "HOU",      4.2, 0.619,
  "NYK",      3.4, 0.610,
  "MIN",      4.6, 0.598,
  "LAC",      4.3, 0.596,
  "DEN",      2.6, 0.594,
  "LAL",      0.7, 0.586,
  "GSW",      2.5, 0.568,
  "MIL",      2.1, 0.568,
  "MEM",      3.7, 0.557,
  "DET",      1.7, 0.523,
  "ORL",     -0.6, 0.489,
  "SAC",      0.3, 0.488,
  "DAL",     -1.2, 0.476,
  "ATL",     -1.5, 0.476,
  "CHI",     -1.8, 0.470,
  "MIA",     -0.5, 0.443,
  "POR",     -3.0, 0.439,
  "PHX",     -3.0, 0.439,
  "SAS",     -2.8, 0.415,
  "TOR",     -4.3, 0.366,
  "BKN",     -7.1, 0.317,
  "PHI",     -6.2, 0.293,
  "NOP",     -9.4, 0.256,
  "CHA",     -9.3, 0.230,
  "WAS",    -12.4, 0.220,
  "UTA",     -9.3, 0.207
)


# merge all datasets (game + advanced + team stats)

joined_table <- nba_2025 %>%
  left_join(advanced_2025, by = "player") %>%
  left_join(nba_2025_stats, by = "team")

# using same columns as in recipe

joined_table <- joined_table %>%
  select(player,
    gs, mp_per_g, fg_per_g, fga_per_g, ft_per_g, fta_per_g
    pts_per_g, trb_per_g, ast_per_g, stl_per_g, blk_per_g,
    per, ws_per_48, vorp, ts_pct, usg_pct, mov_adj, win_lo
```

```
  )

# clear any mising values

joined_table <- na.omit(joined_table)


# predictions, predictions table sorted by .pred

augment(rf_final_fit, new_data = joined_table) %>%
  select(player, .pred) %>%
  arrange(by_group = desc(.pred)) %>%
  head() %>%
  kable()
```

player	.pred
Shai Gilgeous-Alexander	0.8876556
Nikola Joki?	0.7774955
Giannis Antetokounmpo	0.4397301
Alondes Williams	0.3550833
James Wiseman	0.1996443
Jayson Tatum	0.1406513

And your NBA 2025 Random Forest MVP is, Shai-Gilgeous Alexander!!

Hold on though, I wouldnt take this model to Vegas just yet. While this

result is reflective of the result in real life, it is important to note that unfortunately the resulting table does show some problems in our model, notably how Alondes Williams and James Wiseman are being considered serious MVP candidates. Without going into too much detail on the players, these are both guys who rarely see playing time and in reality are not MVP contenders, however they must have really high scores for some variables (James Wiseman has a free throw percentage of 100% this season), that may have led our model to believe that these might have been serious contenders.

Conclusion

Now after fitting all of our models and seeing our results, we came away with random forest being the best for the job. This makes sense as random forest does well in handling data that may not meet assumptions for other regression models, that is to say it is more flexible to data. However as we see the model still has it's problems, notably in handling players with inflated stats that may not be representative of their true skill and value on the court.

Our worst models were simple linear regression and elastic net. This also makes sense especially for simple linear regression, as it primarily works best on data that have strong linear relationships, and as we saw, NBA metrics may not have well defined linear relationships with award share.

Now improvement is always good, it is how rookies eventually become MVP. We notice that our model tends to predict closer to 0. This came as a result of the overwhelming amount of 0 values for award share. Stratifying may have helped a little bit but still our model favored predictions closer to 0 a good amount of the time, as seen in the predicted vs actual plot for the random forest model. We may be able to address this by reducing our data size even more, maybe to only include players who received a vote.

We saw that advanced player statistics were among the most important when it came to predicting award shares, and that in-game metrics such as points and rebounds, played little effect. This makes sense as these metrics were designed for cases like this, to determine who is the best player mathematically using very complex formulas for calculations.

And while our model did in the end predict the MVP correctly, it is still important to note its shortcomings and how in the end, the machine may

not be able to truly capture everything that goes into an MVP race. There is still a lot more that goes into voting, metrics beyond just basketball, maybe if we had access to these kinds of statistics we could make our model even better.

Yet this process was very fun for me as I got to apply my new machine learning knowledge to something I enjoy. This project helped me learn more about the NBA and sports analytics in general. Thanks for taking the time to read about my project, you the real MVP

Sources

- NBA data set was downloaded from Kaggle titled "**1982-2022 "NBA Player Statistics with MVP Votes**'. All data was web scraped from basketball-reference.com by user Robert Sunderhaft.
- nba_2025 was scraped from basketball-reference.com
- mov_adj and win_loss_pct were pulled from basketball-reference.com