Predicting an NBA Player's Position

INTRO

The aim of this project is to build a machine learning model that can predict an NBA player's position based on their statistics. We will be using a data set from Kaggle that has every player's information who has played in the league from 2018 to 2021. All throughout the project, I utilize different machine learning techniques to achieve the best fitted model that I can then use to predict. Let's dive right in!

LINK TO DATA SOURCE:

https://www.kaggle.com/datasets/sumitredekar/nba-stats-2018-2021

Loading Packages and Data

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6      v purrr   0.3.4
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(tidymodels)
```

```
## -- Attaching packages -------------------------------------- tidymodels 1.0.0 --
## v broom        1.0.1     v rsample      1.1.1
## v dials        1.1.0     v tune         1.0.1
## v infer        1.0.4     v workflows    1.1.2
## v modeldata    1.1.0     v workflowsets 1.0.0
## v parsnip      1.0.3     v yardstick    1.1.0
## v recipes      1.0.4
## -- Conflicts ----------------------------------------- tidymodels_conflicts() --
## x scales::discard() masks purrr::discard()
## x dplyr::filter()   masks stats::filter()
## x recipes::fixed()  masks stringr::fixed()
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
## * Use tidymodels_prefer() to resolve common conflicts.
```

```
library(ggplot2)
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
library(timetk)
library(dplyr)
```

```r
library(ggthemes)
library(naniar)
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```r
library(kableExtra)
```

```
##
## Attaching package: 'kableExtra'
##
## The following object is masked from 'package:dplyr':
##
##     group_rows
```

```r
library(glmnet)
```

```
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
##
## Loaded glmnet 4.1-6
```

```r
library(themis)
library(discrim)
```

```
##
## Attaching package: 'discrim'
##
## The following object is masked from 'package:dials':
##
##     smoothness
```

```r
tidymodels_prefer()

set.seed(3435)
```

```r
NBA <- read_csv("NBA_stats_18-21.csv")
```

```
## Rows: 2728 Columns: 25
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (3): Player, Pos, Team
## dbl (22): Age, Games, Minutes Played, FG, FGA, 3P, 3PA, 2P, 2PA, FT, FTA, OR...
##
## 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.
```

```r
#NBA %>% View()
```

EXPLORATORY DATA ANALYSIS (EDA):

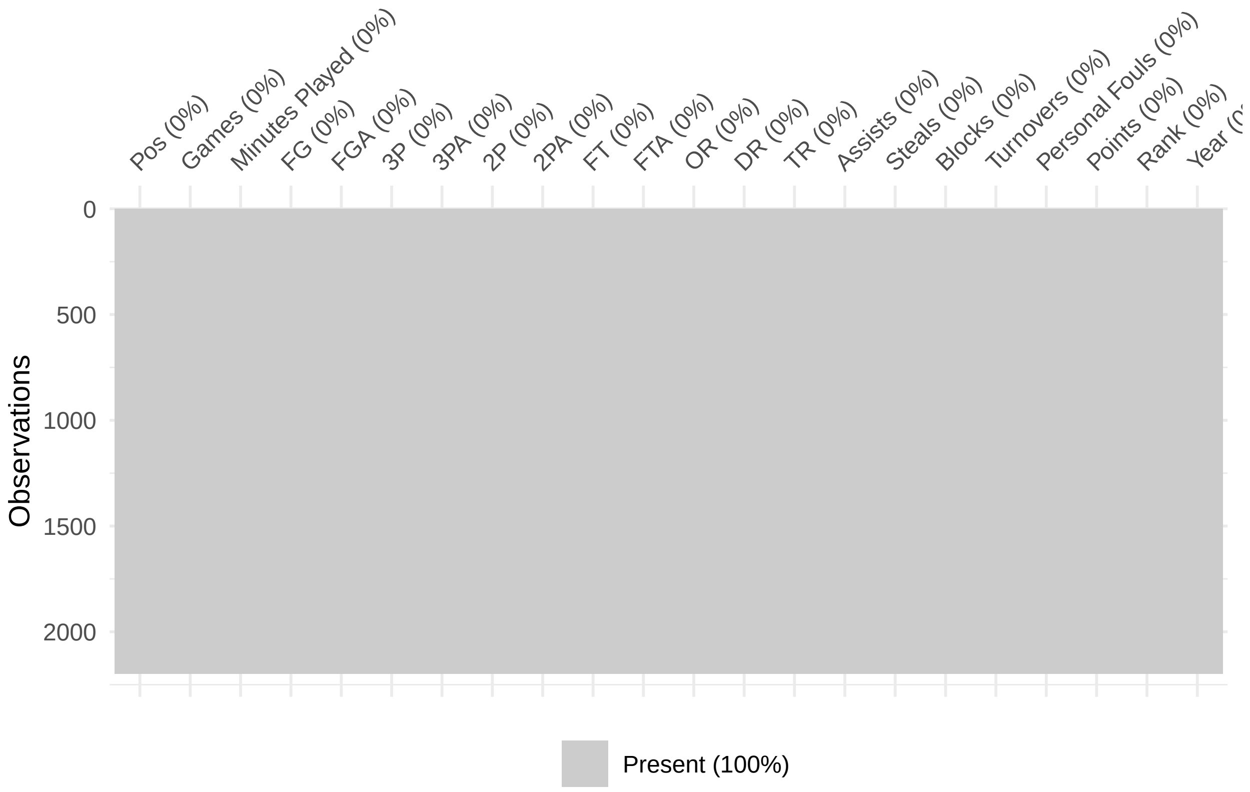
Cleaning the Data

```
#CLEANING DATA
sub_NBA <- subset(NBA, NBA$`Minutes Played` > 10)

#changing position values
sub_NBA$Pos[sub_NBA$Pos == 'SF-SG'] <- 'SF'
sub_NBA$Pos[sub_NBA$Pos == 'SF-C'] <- 'SF'
sub_NBA$Pos[sub_NBA$Pos == 'SF-PF'] <- 'SF'
sub_NBA$Pos[sub_NBA$Pos == 'PG-SG'] <- 'PG'
sub_NBA$Pos[sub_NBA$Pos == 'SG-PF'] <- 'SG'
sub_NBA$Pos[sub_NBA$Pos == 'SG-SF'] <- 'SG'
sub_NBA$Pos[sub_NBA$Pos == 'SG-PG'] <- 'SG'
sub_NBA$Pos[sub_NBA$Pos == 'PF-SF'] <- 'PF'
sub_NBA$Pos[sub_NBA$Pos == 'PF-C'] <- 'PF'
sub_NBA$Pos[sub_NBA$Pos == 'C-PF'] <- 'C'

#dataset just to locate names (needed for MODEL INPUT)
NBA_names <- sub_NBA[,-c(3,4)]
#NBA_names %>% View()

#getting rid of unnecessary columns
updated_NBA <- sub_NBA[,-c(1,3,4)]
#updated_NBA %>% View()

# checking to see if positions are evenly distributed
table(updated_NBA$Pos)
```

```
##
##   C  PF  PG  SF  SG
## 420 430 426 392 529
```

```
# checking if there are NA values
vis_miss(updated_NBA)
```

```
## Warning: `gather_()` was deprecated in tidyr 1.2.0.
## i Please use `gather()` instead.
## i The deprecated feature was likely used in the visdat package.
##   Please report the issue at <https://github.com/ropensci/visdat/issues>.
```



In today's NBA, players are no longer forced into a box consisting of the five traditional positions: Point Guard (PG), Shooting Guard (SG), Small Forward (SF), Power Forward (PF), & Center (C). Now, they can play multiple positions which leads to some players being classified as a combination of one or more positions. In an effort to simplify the project, I assigned players who were categorized as more than one position back to only being their original position. For example, a player who was classified as PF-C would return to just being a PF. Next, I also removed players who played less than an average of 10 minutes per game. The reason being that players who played less than this acceptable level of minutes would unfairly skew the data. Moreover, I removed unnecessary columns same as Name, Age, and Team as they would have no effect on a player's position. Using the table function, I was able to check if each position was evenly distributed which they were so no re-balancing was needed. Lastly with the vis_miss function, I was able to confirm that my data set had **no missing values**.

CODEBOOK:

Response Variable - Pos: Position (PG: Point Guard, SG: Shooting Guard, SF: Small Forward, PF: Power Forward, C: Center)

Predictor Variables (numeric variables outside of Games, Rank, & Year are season averages so they'll be positive real numbers)

- `Games`: Games played (values from 1-82)
- `Minutes Played`: Minutes Played
- `FG`: Field Goals (includes both 2-point field goals and 3-point field goals)
- `FGA`: Field Goal Attempts (includes both 2-point field goal attempts and 3-point field goal attempts)
- `3P`: 3-Point Field Goals
- `3PA`: 3-Point Field Goal Attempts
- `2P`: 2-Point Field Goals

- `2PA`: 2-Point Field Goal Attempts
- `FT`: Free Throws
- `FTA`: Free Throw Attempts
- `OR`: Offensive Rebounds
- `DR`: Defensive Rebounds
- `TR`: Total Rebounds; the formula is OR + DR
- `Assists`: Assists
- `Steals`: Steals
- `Blocks`: Blocks
- `Turnovers`: Turnovers
- `Personal Fouls`: Fouls
- `Points`: Points
- `Rank`: Row number from the original data set
- `Year`: Year that the season occurred (2018, 2019, 2020, or 2021)

Summary Statistics by Position

```r
#finding avg of each position
avg_pos <- updated_NBA %>% group_by(Pos) %>%
  summarise(mean_points = mean(Points),
            mean_rebounds = mean(TR),
            mean_off_rebounds = mean(OR),
            mean_assists = mean(Assists),
            mean_steals = mean(Steals),
            mean_blocks = mean(Blocks),
            mean_fg = mean(FG),
            mean_fga = mean(FGA),
            mean_3p = mean(`3P`),
            mean_3pa = mean(`3PA`),
            mean_2p = mean(`2P`),
            mean_2pa = mean(`2PA`),
            mean_ft = mean(FT),
            mean_fta = mean(FTA),
            mean_pfouls = mean(`Personal Fouls`),
            mean_turnovers = mean(Turnovers),
            .groups = 'drop') %>%
  as.data.frame()
avg_pos
```

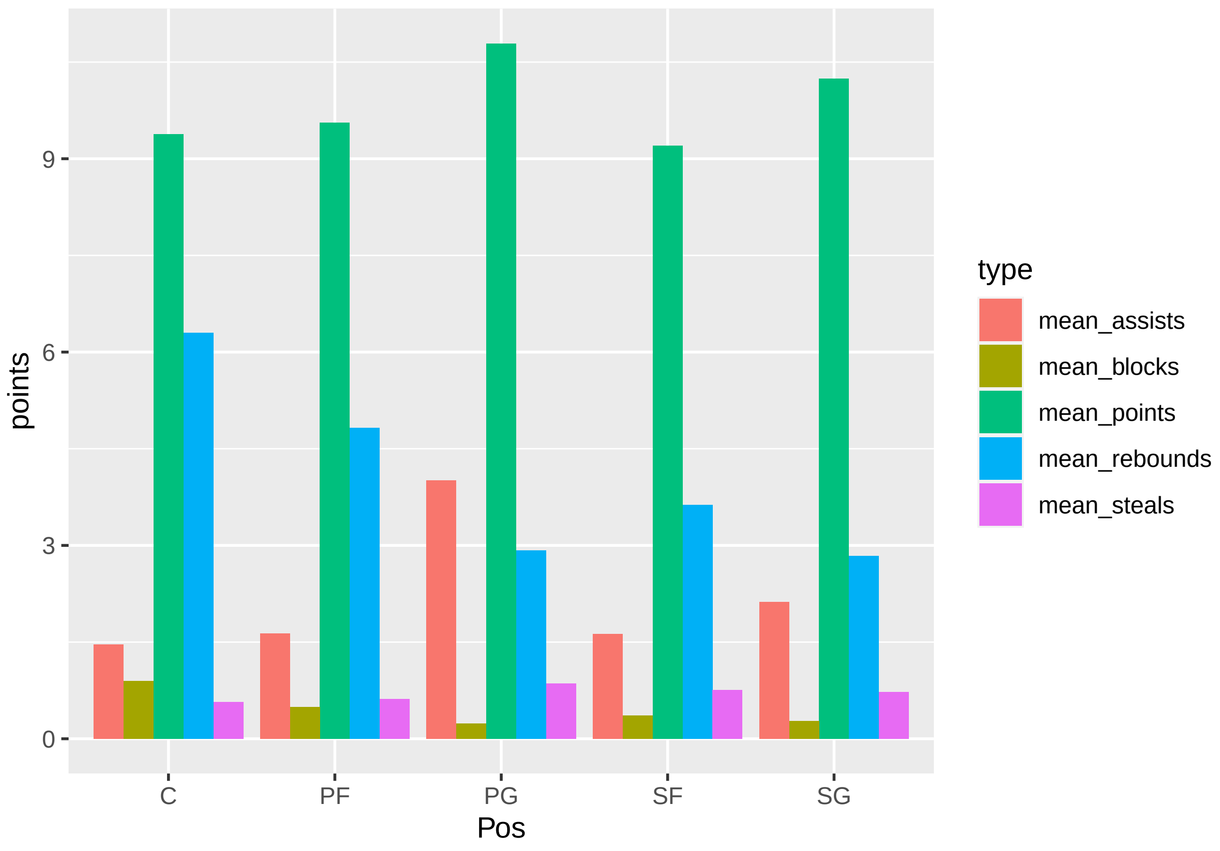
```
##   Pos mean_points mean_rebounds mean_off_rebounds mean_assists mean_steals
## 1   C    9.383810      6.296905         1.8538095     1.466667   0.5726190
## 2  PF    9.557442      4.822326         1.0795349     1.635814   0.6172093
## 3  PG   10.787559      2.923474         0.5192488     4.007746   0.8575117
## 4  SF    9.200765      3.626786         0.7262755     1.622449   0.7535714
## 5  SG   10.242722      2.834783         0.4867675     2.126465   0.7294896
##   mean_blocks  mean_fg mean_fga   mean_3p mean_3pa  mean_2p mean_2pa  mean_ft
## 1   0.8938095 3.682857 6.903810 0.4095238 1.202619 3.273810 5.702857 1.607381
## 2   0.4969767 3.529535 7.667674 1.0334884 2.926279 2.501163 4.745814 1.473023
## 3   0.2401408 3.899765 9.005399 1.1798122 3.349296 2.720657 5.657042 1.817371
## 4   0.3647959 3.331122 7.551276 1.1561224 3.273214 2.175765 4.279337 1.375000
## 5   0.2724008 3.689225 8.646881 1.4234405 3.970510 2.267864 4.680718 1.446314
##   mean_fta mean_pfouls mean_turnovers
## 1 2.268571    2.291667       1.184286
## 2 1.947674    1.972558       1.122326
## 3 2.265493    1.737089       1.666901
## 4 1.773214    1.823469       1.015051
## 5 1.814178    1.736484       1.193195
```

In this next step, I found the average statistics of each position by using the mean() function. From the table created, we can also see how certain statistics are more geared towards certain positions: point guards-assists and centers-tr (total rebounds). This makes sense as point guards are the ones who take the ball up the court and are tasked with distributing the ball. Similarly, centers are often the tallest players on the court and are assigned to guard the basket when playing on defense and to (traditionally) stay under the opposing basket to pick up missed shots when playing on offense. Furthermore, this was extremely helpful because it allowed me to have an idea of what a typical player's statistics regarding these 5 different positions would be like. In doing so, I was also able to get a good idea of what to expect from the model, especially in terms of how the model would prioritize certain statistics that help categorize a player's position. That would beg the question: would the model ultimately be able to better predict positions that have certain statistics more related to it (PG & C) over others that don't (SG, SF, PF)?

Moving on, I started to visualize the data by constructing different plots.

PLOTTING:

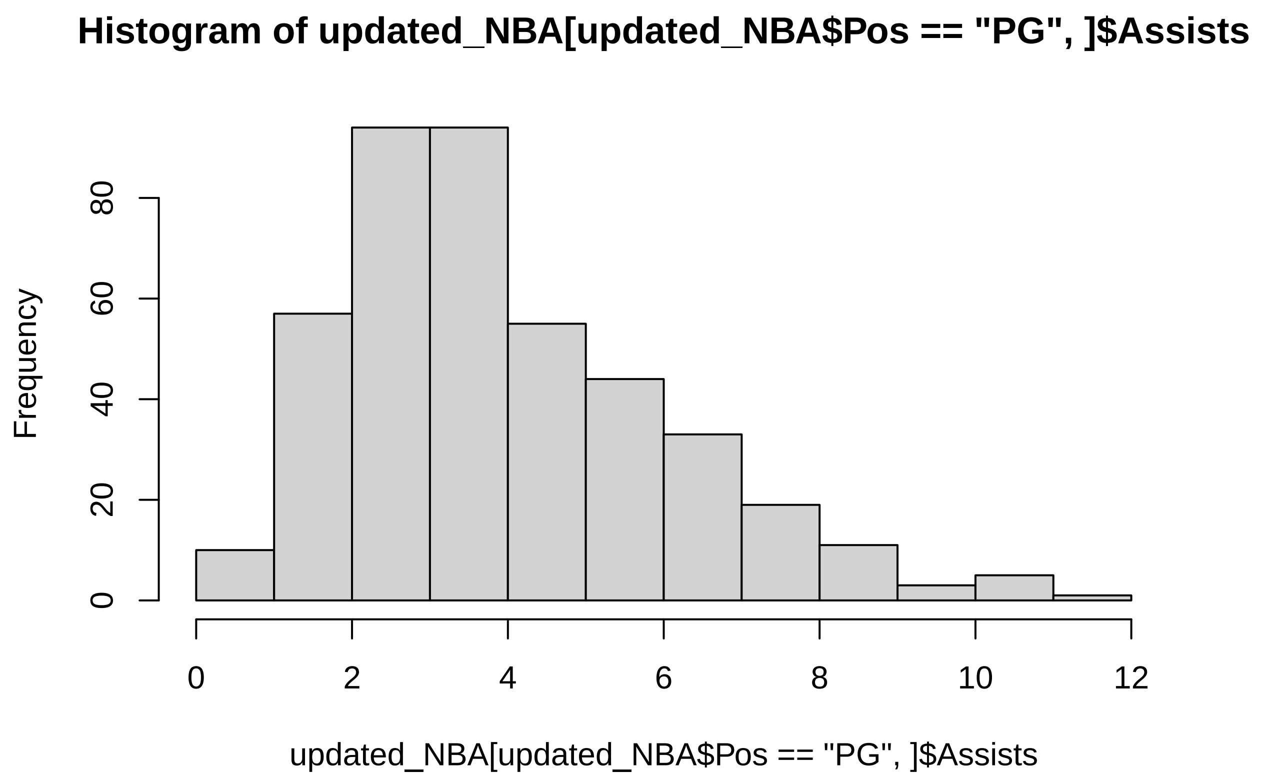
```r
avg_pos %>%
  select(-c(mean_3p, mean_3pa, mean_2p, mean_2pa, mean_fg, mean_fga,
            mean_ft, mean_fta, mean_off_rebounds, mean_pfouls, mean_turnovers)) %>%
  pivot_longer(cols = starts_with("mean"), names_to = "type",
               values_to = "points") %>%
  ggplot(aes(x = Pos, y = points, fill = type)) + geom_bar(stat = "identity", position = "dodge")
```



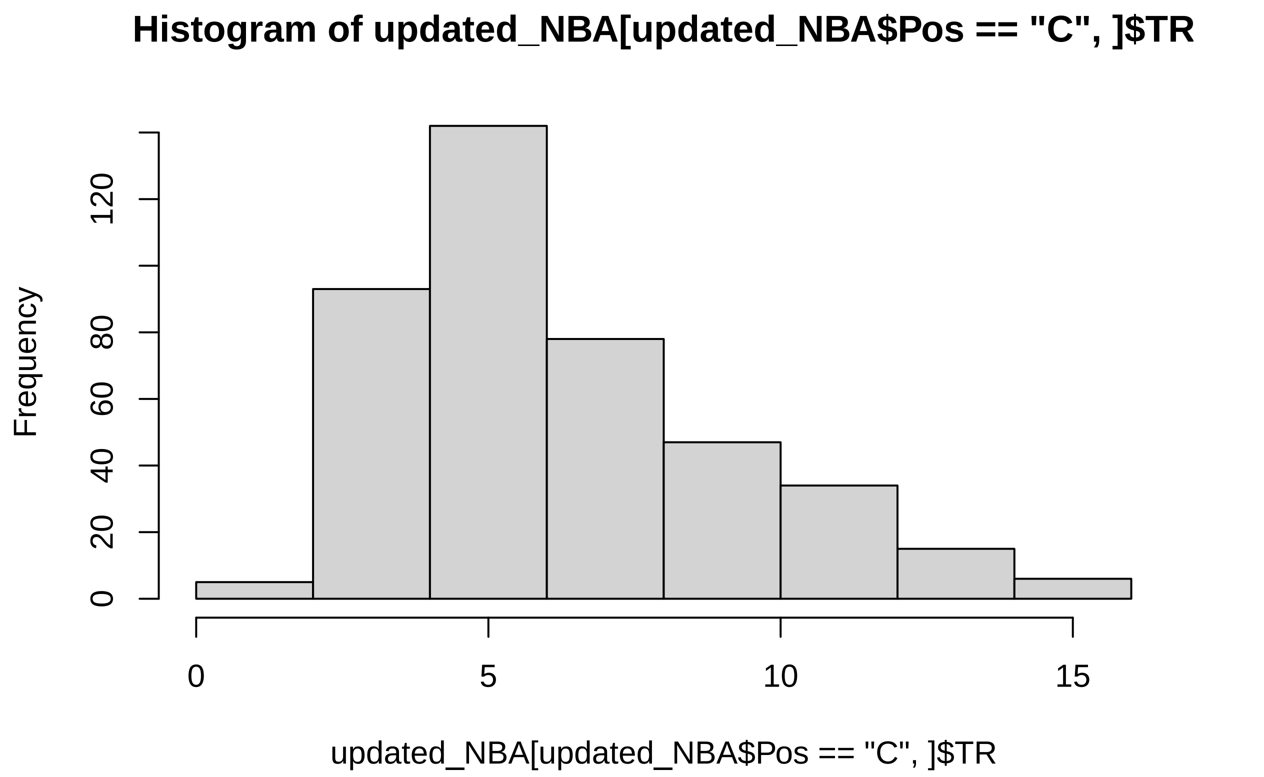
In this bar chart, it shows the five main statistics in basketball - Assists, Blocks, Points, Rebounds, & Steals - and how they are distributed among the five positions. Just like before, we are able to confirm how some statistics are higher for certain positions, while points does not vary that much. On top of assists and rebounds, blocks is another statistic that is more geared towards centers. Likewise to the table, this feature could be deemed as important by the upcoming models.

Histograms

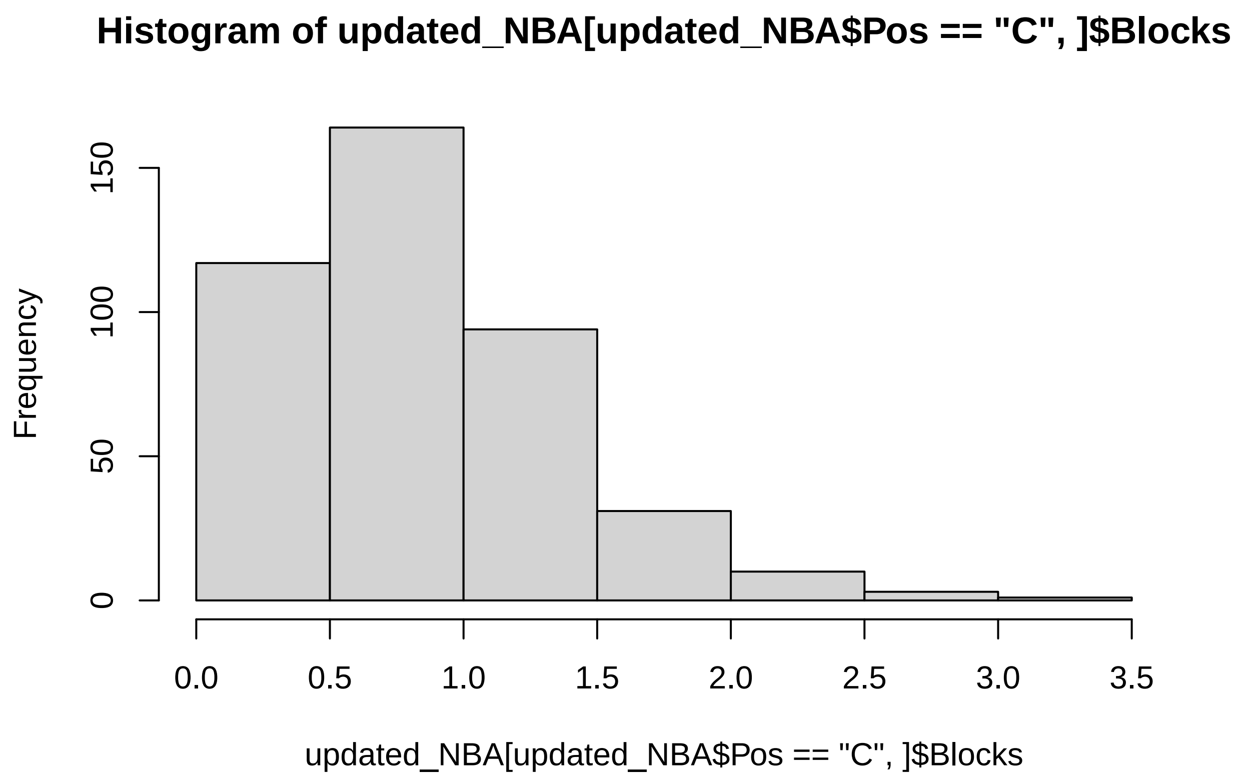
```
hist(updated_NBA[updated_NBA$Pos == "PG", ]$Assists)
```



```
hist(updated_NBA[updated_NBA$Pos == "C", ]$TR)
```



```
hist(updated_NBA[updated_NBA$Pos == "C", ]$Blocks)
```



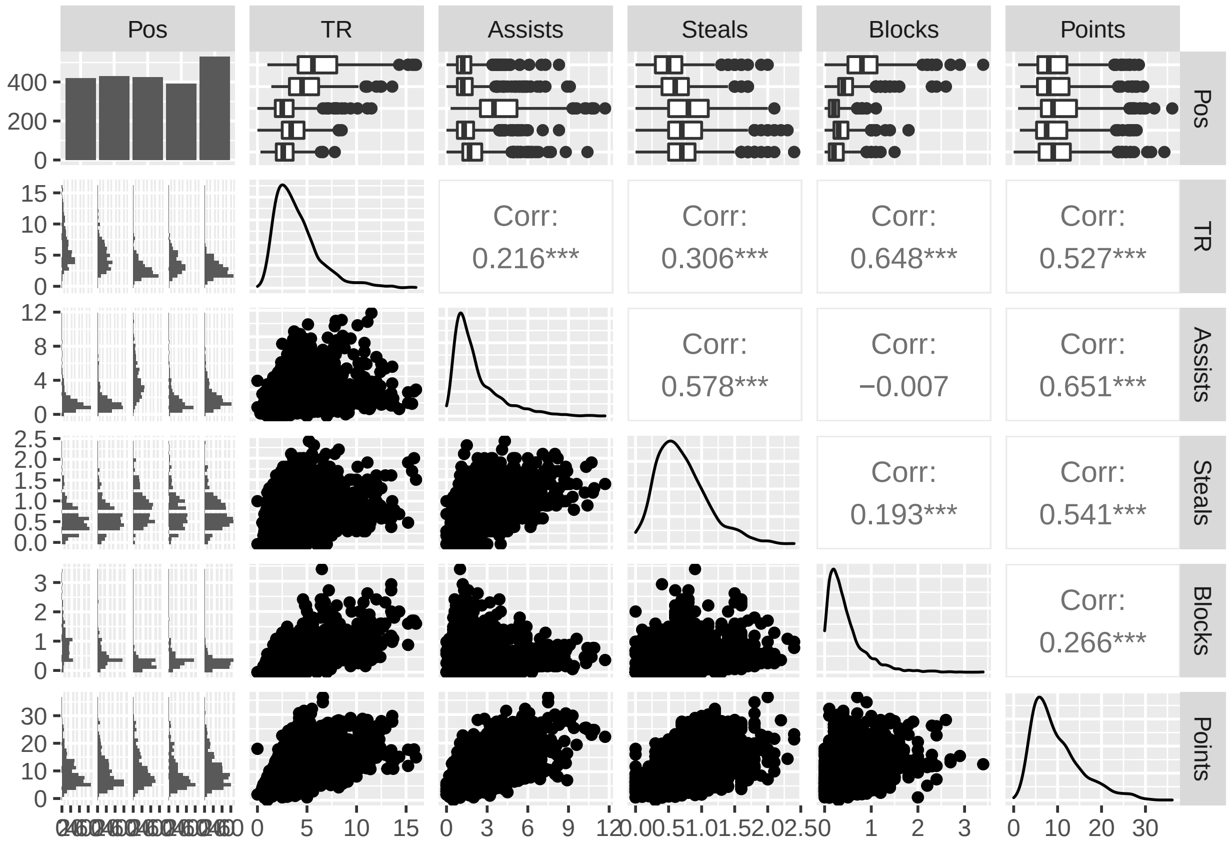
Using histograms, I wanted to investigate further the relationships between certain statistics and their respective positions. All three of them were more centered towards small values with the skew going towards a larger value. Looking at the histogram for assists, we can see how point guards mainly average 2-4 assists with the histogram skewing towards having a higher number like 12. For total rebounds, centers mainly average 5 rebounds with the histogram also skewing towards having a higher number such as 15+. For blocks, centers mainly average 0.5-1 blocks with no surprise, the histogram skewing towards having a higher number like 3.5.

I found these histograms surprising yet realistic because I imagined these numbers being way higher. It made sense to me though because I realized I'm so used to stat checking the best players on NBA rosters; it would make sense for stars to have high numbers while role/bench players would have more moderate numbers which the data ultimately proved.

Seaborn Pair Plot

```
seaborn <- subset(updated_NBA, select = -c(Games, `Minutes Played`, DR, Rank, Year,
                                           `3P`, `3PA`, `2P`, `2PA`, FG, FGA, FT,
                                           FTA, OR, `Personal Fouls`, Turnovers))
ggpairs(seaborn)
```

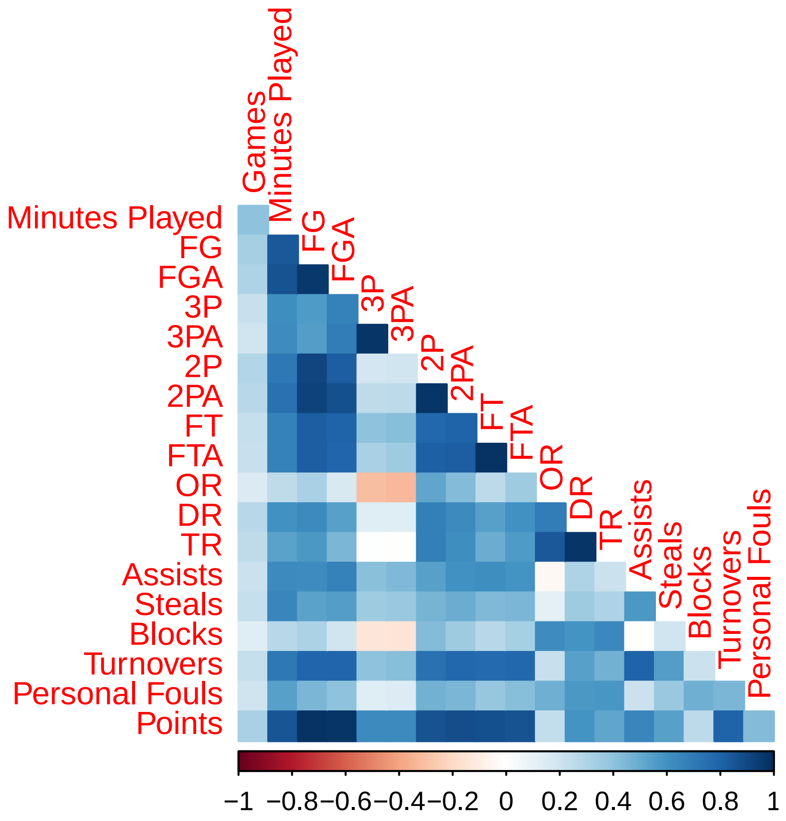
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Moving on to a close-enough R-equivalent of a Seaborn Pair Plot, this was especially useful in gauging the relationship between the different variables. I noticed that most of the scatter plots generally depicted a positive correlation - in other words, as one variable increased, so did the other. This would make sense because most good players will be well-rounded; they won't only be good in one statistical category but rather others as well. However, I do realize that in some scenarios, there is a sort of negative correlation. This would be the case for the relationships between TR (total rebound) & Assists, as well as Blocks and Assists. There is no surprise there because as previously discussed, the statistical category of assists is more geared towards point guards while total rebounds and blocks lean more towards centers or even power forwards (often times the tallest players on a team). Seeing as how these statistics have a propensity towards two positions that play completely different roles, the negative correlation definitely makes total sense.

Variable Correlation Plot

```
#corr matrix
updated_NBA %>%
  select(where(is.numeric), -c(Year, Rank)) %>%
  cor() %>%
  corrplot(type = 'lower', diag = FALSE,
           method = 'color')
```



Similarly to the Seaborn Pair Plots, the correlation matrix allows us to see the relationships between the different variables - however this time, we don't focus on just the response variable and the 5 main statistics in basketball, but every variable at once. Notice how there's a heavy correlation between attempts and shots made (FGA & FG, 2PA & 2P, etc) which is understandable; if you have more attempts, it would make sense for you to make more shots. Likewise, points has a strong correlation with most of the points related statistics (FG/FGA, 3P/3PA, 2P/2PA, & FT/FTA).

An interesting thing I notice is that there is a pretty strong correlation between FT (free throws) and 2PA. As they are relatively near the same zone in the basketball court, this is unsurprising. Points and its related statistics also have a decent correlation with turnovers; if someone is able to score more, it would imply that they have the ball more often than not. In that case, it would make sense for them to have more turnovers as they would have higher chances of losing the ball.

Contrasting these correlations, there is a weak relationship between OR (offensive rebounds) and 3P/3PA, in addition to Blocks and 3P/3PA. Just like the difference between assists and TR/Blocks, the explanation is the exact same in this scenario. As we know, OR and Blocks are more prevalent for the taller positions on the court: PF and C. On the other hand, 3P/3PA would be more closely related to point guards and shooting guards traditionally. This also makes sense just thinking about basketball logically; 3P/3PA would be outside of the 3-point line in a basketball court, while OR and Blocks would be more towards the basket.

SETTING UP MODELS:

With the EDA portion done, we now move on to setting up our models. The first thing we have to do is first split the data into a training set and a testing set. I split up the training and testing tests with a 80:20 ratio with the training set being the one with more data. With the previously set seed, the results should also be consistent even after multiple iterations. Stratifying the data on the response variable, `Pos`, we are one step closer towards building our models!

Training/Testing Split

```r
#split data into training and testing
#usually ratios of 70:30 to 80:20 (more training data than testing data)
nba_split <- initial_split(updated_NBA, prop = 0.80,
                              strata = Pos)
nba_train <- training(nba_split)
nba_test <- testing(nba_split)

#nba_train %>% View()
```

Making the Recipe

Now that we have our training and testing sets, the next step is to make our recipe. A recipe is exactly as it sounds - it is a collection of steps that is applied to our training set which we can then use for data analysis. Excluding variables with high correlation, all the other predictor variables are used in our recipe. The recipe in full can be seen below!

```r
#recipe
new <- subset(nba_train, select=-c(FG, `3P`, `2P`, FT, DR))
nba_recipe <- recipe(Pos ~ ., data = new) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_predictors())
nba_recipe %>% prep() %>% bake(nba_train)
```

```
## # A tibble: 1,756 x 17
##      Games Minutes P~1    FGA `3PA`   `2PA`     FTA     OR     TR Assists Steals
##      <dbl>       <dbl>  <dbl> <dbl>   <dbl>   <dbl>  <dbl>  <dbl>   <dbl>  <dbl>
##  1  1.27         1.44   0.327 -1.39  1.34    1.09    5.50   2.10   -0.564  1.27
##  2  0.969       -0.340 -0.720 -1.35 -0.0675  0.353   1.03   0.614  -0.391 -0.522
##  3  1.10        -0.312 -0.580 -1.30  0.0886 -0.0172  1.42   0.572  -0.853 -0.779
##  4 -1.25        -1.68  -1.58  -1.39 -1.19   -0.820  -0.552 -0.620  -1.14  -1.55
##  5  1.49        -0.547 -0.580 -1.25  0.0574 -0.635   0.896  0.572  -0.622 -1.04
##  6  0.448       -1.11  -1.12  -1.35 -0.567  -0.758   0.238 -0.194  -0.218 -0.265
##  7 -0.204       -1.17  -1.16  -1.39 -0.630  -0.326   1.03   0.104  -0.795 -0.779
##  8  1.53        -0.561 -0.859 -1.39 -0.224  -0.0789  1.16   0.699  -0.795 -1.04
##  9  1.19         0.722  0.257 -1.39  1.28    0.909   3.13   2.87   -0.737  0.248
## 10 -0.0299       0.377 -0.906 -1.39 -0.286  -0.0172  2.61   2.15   -0.564 -1.04
## # ... with 1,746 more rows, 7 more variables: Blocks <dbl>, Turnovers <dbl>,
## #   `Personal Fouls` <dbl>, Points <dbl>, Rank <dbl>, Year <dbl>, Pos <fct>,
## #   and abbreviated variable name 1: `Minutes Played`
```

```r
#cross validation
nba_folds <- vfold_cv(data = nba_train, v = 5)
```

To help with the issue of imbalanced data, we use cross validation using 5 folds.

4 MODELS:

With everything in place now, we can start fitting our data to different models. I decided to go with logistic regression, K-Nearest Neighbors (KNN), elastic net logistic regression, and linear discriminant analysis for my models. Since our variable of interest is categorical (`Pos`), we use classification. The **full model building process** is as follows:

1. Set up the model by specifying what type it is, then set up the engine and mode (classification in our case)

2. Set up the workflow for the corresponding model, add the new model, & add the recipe

For Logistic Regression, Linear Discriminant Analysis (LDA), & Quadratic Discriminant Analysis (QDA), skip steps 3-5!

3. Set up the tuning grid with the desired parameters that are to be tuned, as well as specifying the total number of levels for tuning for each parameter
4. Tune the models with desired parameters
5. After tuning, it is now time to choose the best model! Once decided, finalize the workflow with the specific tuning parameters
6. Fit the model with the workflow to the training set!

```
#logistic regression
log_mod <- multinom_reg(penalty = 0) %>%
    set_mode("classification") %>%
    set_engine("glmnet")

log_wf <- workflow() %>%
    add_model(log_mod) %>%
    add_recipe(nba_recipe)
```

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

knn_wf <- workflow() %>%
    add_model(knn_mod) %>%
    add_recipe(nba_recipe)

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

```
#elastic net logistic regression
elastic_log_mod <- multinom_reg(penalty = tune(), mixture = tune(), ) %>%
    set_mode("classification") %>%
    set_engine("glmnet")

elastic_log_wf <- workflow() %>%
    add_model(elastic_log_mod) %>%
    add_recipe(nba_recipe)

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

```
#Linear Discriminant Analysis
lda_mod <- discrim_linear() %>%
    set_mode("classification") %>%
    set_engine("MASS")

lda_wf <- workflow() %>%
    add_model(lda_mod) %>%
    add_recipe(nba_recipe)
```

Having completed setting up our models, we can now tune our models.

.metric	.estimator	mean	n	std_err	.config
roc_auc	hand_till	0.8906321	5	0.0048432	Preprocessor1_Model1

neighbors	.metric	.estimator	mean	n	std_err	.config
1	roc_auc	hand_till	0.7461770	5	0.0076379	Preprocessor1_Model01
2	roc_auc	hand_till	0.8015697	5	0.0071495	Preprocessor1_Model02
3	roc_auc	hand_till	0.8241486	5	0.0065815	Preprocessor1_Model03
4	roc_auc	hand_till	0.8378665	5	0.0058286	Preprocessor1_Model04
5	roc_auc	hand_till	0.8459822	5	0.0045651	Preprocessor1_Model05
6	roc_auc	hand_till	0.8507586	5	0.0046447	Preprocessor1_Model06
7	roc_auc	hand_till	0.8549652	5	0.0040601	Preprocessor1_Model07
8	roc_auc	hand_till	0.8571620	5	0.0040067	Preprocessor1_Model08
9	roc_auc	hand_till	0.8593955	5	0.0037174	Preprocessor1_Model09
10	roc_auc	hand_till	0.8600300	5	0.0035546	Preprocessor1_Model10

```r
# Fitting all models to folded data:
log_res <- fit_resamples(object = log_wf, resamples = nba_folds)

knn_res <- tune_grid(object = knn_wf, resamples = nba_folds,
    grid = knn_grid)

elastic_res <- tune_grid(object = elastic_log_wf, resamples = nba_folds, grid = elastic_log_grid)

lda_res <- fit_resamples(object = lda_wf, resamples = nba_folds)
```

```r
#using collect_metrics():
collect_metrics(log_res) %>%
    filter(.metric == "roc_auc") %>%
    kable() %>%
    kable_styling(full_width = FALSE) %>%
    scroll_box(width = "100%", height = "200px")
```

```r
collect_metrics(knn_res) %>%
    filter(.metric == "roc_auc") %>%
    kable() %>%
    kable_styling(full_width = FALSE) %>%
    scroll_box(width = "100%", height = "200px")
```

```r
collect_metrics(elastic_res) %>%
    filter(.metric == "roc_auc") %>%
    kable() %>%
    kable_styling(full_width = FALSE) %>%
    scroll_box(width = "100%", height = "200px")
```

```r
#check!!
collect_metrics(lda_res) %>%
    filter(.metric == "roc_auc") %>%
    kable() %>%
    kable_styling(full_width = FALSE) %>%
    scroll_box(width = "100%", height = "200px")
```

Now, we can determine which model fits best.

penalty	mixture	.metric	.estimator	mean	n	std_err	.config
0.0000000	0.0000000	roc_auc	hand_till	0.8794531	5	0.0045026	Preprocessor1_Model001
0.0000000	0.0000000	roc_auc	hand_till	0.8794531	5	0.0045026	Preprocessor1_Model002
0.0000000	0.0000000	roc_auc	hand_till	0.8794531	5	0.0045026	Preprocessor1_Model003
0.0000002	0.0000000	roc_auc	hand_till	0.8794531	5	0.0045026	Preprocessor1_Model004
0.0000028	0.0000000	roc_auc	hand_till	0.8794531	5	0.0045026	Preprocessor1_Model005
0.0000359	0.0000000	roc_auc	hand_till	0.8794531	5	0.0045026	Preprocessor1_Model006
0.0004642	0.0000000	roc_auc	hand_till	0.8794531	5	0.0045026	Preprocessor1_Model007
0.0059948	0.0000000	roc_auc	hand_till	0.8794531	5	0.0045026	Preprocessor1_Model008
0.0774264	0.0000000	roc_auc	hand_till	0.8701790	5	0.0041613	Preprocessor1_Model009
1.0000000	0.0000000	roc_auc	hand_till	0.8215672	5	0.0049298	Preprocessor1_Model010
0.0000000	0.1111111	roc_auc	hand_till	0.8906415	5	0.0048043	Preprocessor1_Model011
0.0000000	0.1111111	roc_auc	hand_till	0.8906415	5	0.0048043	Preprocessor1_Model012
0.0000000	0.1111111	roc_auc	hand_till	0.8906415	5	0.0048043	Preprocessor1_Model013
0.0000002	0.1111111	roc_auc	hand_till	0.8906415	5	0.0048043	Preprocessor1_Model014
0.0000028	0.1111111	roc_auc	hand_till	0.8906415	5	0.0048043	Preprocessor1_Model015
0.0000359	0.1111111	roc_auc	hand_till	0.8906415	5	0.0048043	Preprocessor1_Model016
0.0004642	0.1111111	roc_auc	hand_till	0.8903008	5	0.0048130	Preprocessor1_Model017
0.0059948	0.1111111	roc_auc	hand_till	0.8861803	5	0.0045668	Preprocessor1_Model018
0.0774264	0.1111111	roc_auc	hand_till	0.8697501	5	0.0040128	Preprocessor1_Model019
1.0000000	0.1111111	roc_auc	hand_till	0.7348183	5	0.0059402	Preprocessor1_Model020
0.0000000	0.2222222	roc_auc	hand_till	0.8906721	5	0.0048172	Preprocessor1_Model021
0.0000000	0.2222222	roc_auc	hand_till	0.8906721	5	0.0048172	Preprocessor1_Model022
0.0000000	0.2222222	roc_auc	hand_till	0.8906721	5	0.0048172	Preprocessor1_Model023
0.0000002	0.2222222	roc_auc	hand_till	0.8906721	5	0.0048172	Preprocessor1_Model024
0.0000028	0.2222222	roc_auc	hand_till	0.8906721	5	0.0048172	Preprocessor1_Model025
0.0000359	0.2222222	roc_auc	hand_till	0.8906721	5	0.0048172	Preprocessor1_Model026
0.0004642	0.2222222	roc_auc	hand_till	0.8903470	5	0.0048361	Preprocessor1_Model027
0.0059948	0.2222222	roc_auc	hand_till	0.8862743	5	0.0046310	Preprocessor1_Model028
0.0774264	0.2222222	roc_auc	hand_till	0.8673571	5	0.0037671	Preprocessor1_Model029
1.0000000	0.2222222	roc_auc	hand_till	0.6497027	5	0.0044068	Preprocessor1_Model030
0.0000000	0.3333333	roc_auc	hand_till	0.8906925	5	0.0048214	Preprocessor1_Model031
0.0000000	0.3333333	roc_auc	hand_till	0.8906925	5	0.0048214	Preprocessor1_Model032
0.0000000	0.3333333	roc_auc	hand_till	0.8906925	5	0.0048214	Preprocessor1_Model033
0.0000002	0.3333333	roc_auc	hand_till	0.8906925	5	0.0048214	Preprocessor1_Model034
0.0000028	0.3333333	roc_auc	hand_till	0.8906925	5	0.0048214	Preprocessor1_Model035
0.0000359	0.3333333	roc_auc	hand_till	0.8906925	5	0.0048214	Preprocessor1_Model036
0.0004642	0.3333333	roc_auc	hand_till	0.8903443	5	0.0048408	Preprocessor1_Model037
0.0059948	0.3333333	roc_auc	hand_till	0.8864307	5	0.0046941	Preprocessor1_Model038
0.0774264	0.3333333	roc_auc	hand_till	0.8605252	5	0.0031793	Preprocessor1_Model039
1.0000000	0.3333333	roc_auc	hand_till	0.5000000	5	0.0000000	Preprocessor1_Model040
0.0000000	0.4444444	roc_auc	hand_till	0.8907001	5	0.0048075	Preprocessor1_Model041
0.0000000	0.4444444	roc_auc	hand_till	0.8907001	5	0.0048075	Preprocessor1_Model042
0.0000000	0.4444444	roc_auc	hand_till	0.8907001	5	0.0048075	Preprocessor1_Model043
0.0000002	0.4444444	roc_auc	hand_till	0.8907001	5	0.0048075	Preprocessor1_Model044
0.0000028	0.4444444	roc_auc	hand_till	0.8907001	5	0.0048075	Preprocessor1_Model045
0.0000359	0.4444444	roc_auc	hand_till	0.8907001	5	0.0048075	Preprocessor1_Model046
0.0004642	0.4444444	roc_auc	hand_till	0.8903694	5	0.0047976	Preprocessor1_Model047
0.0059948	0.4444444	roc_auc	hand_till	0.8865859	5	0.0046946	Preprocessor1_Model048
0.0774264	0.4444444	roc_auc	hand_till	0.8487826	5	0.0036433	Preprocessor1_Model049
1.0000000	0.4444444	roc_auc	hand_till	0.5000000	5	0.0000000	Preprocessor1_Model050
0.0000000	0.5555556	roc_auc	hand_till	0.8906574	5	0.0047777	Preprocessor1_Model051
0.0000000	0.5555556	roc_auc	hand_till	0.8906574	5	0.0047777	Preprocessor1_Model052
0.0000000	0.5555556	roc_auc	hand_till	0.8906574	5	0.0047777	Preprocessor1_Model053
0.0000002	0.5555556	roc_auc	hand_till	0.8906574	5	0.0047777	Preprocessor1_Model054
0.0000028	0.5555556	roc_auc	hand_till	0.8906574	5	0.0047777	Preprocessor1_Model055
0.0000359	0.5555556	roc_auc	hand_till	0.8906574	5	0.0047777	Preprocessor1_Model056
0.0004642	0.5555556	roc_auc	hand_till	0.8903479	5	0.0048000	Preprocessor1_Model057

.metric	.estimator	mean	n	std_err	.config
roc_auc	hand_till	0.8758595	5	0.0071782	Preprocessor1_Model1

.metric	.estimator	mean	n	std_err	.config
roc_auc	hand_till	0.8906321	5	0.0048432	Preprocessor1_Model1

```
# determining which model fits best:
show_best(log_res, metric = "roc_auc") %>%
    kable() %>%
    kable_styling(full_width = FALSE) %>%
    scroll_box(width = "100%", height = "200px")
```

```
show_best(knn_res, metric = "roc_auc") %>%
    kable() %>%
    kable_styling(full_width = FALSE) %>%
    scroll_box(width = "100%", height = "200px")
```

```
show_best(elastic_res, metric = "roc_auc") %>%
    kable() %>%
    kable_styling(full_width = FALSE) %>%
    scroll_box(width = "100%", height = "200px")
```

```
#check!!
show_best(lda_res, metric = "roc_auc") %>%
    kable() %>%
    kable_styling(full_width = FALSE) %>%
    scroll_box(width = "100%", height = "200px")
```

```
log_best <- show_best(log_res, metric = "roc_auc")

knn_best <- select_by_one_std_err(knn_res, desc(neighbors), metric = "roc_auc")

knn_best %>%
    kable() %>%
    kable_styling(full_width = FALSE) %>%
    scroll_box(width = "100%", height = "200px")
```

```
elastic_log_best <- select_by_one_std_err(elastic_res, penalty, mixture, metric = "roc_auc")

elastic_log_best %>%
    kable() %>%
    kable_styling(full_width = FALSE) %>%
    scroll_box(width = "100%", height = "200px")
```

```
lda_best <- show_best(lda_res, metric = "roc_auc")
```

neighbors	.metric	.estimator	mean	n	std_err	.config
10	roc_auc	hand_till	0.8600300	5	0.0035546	Preprocessor1_Model10
9	roc_auc	hand_till	0.8593955	5	0.0037174	Preprocessor1_Model09
8	roc_auc	hand_till	0.8571620	5	0.0040067	Preprocessor1_Model08
7	roc_auc	hand_till	0.8549652	5	0.0040601	Preprocessor1_Model07
6	roc_auc	hand_till	0.8507586	5	0.0046447	Preprocessor1_Model06

penalty	mixture	.metric	.estimator	mean	n	std_err	.config
0.0e+00	0.8888889	roc_auc	hand_till	0.8907029	5	0.0048406	Preprocessor1_Model081
0.0e+00	0.8888889	roc_auc	hand_till	0.8907029	5	0.0048406	Preprocessor1_Model082
0.0e+00	0.8888889	roc_auc	hand_till	0.8907029	5	0.0048406	Preprocessor1_Model083
2.0e-07	0.8888889	roc_auc	hand_till	0.8907029	5	0.0048406	Preprocessor1_Model084
2.8e-06	0.8888889	roc_auc	hand_till	0.8907029	5	0.0048406	Preprocessor1_Model085

.metric	.estimator	mean	n	std_err	.config
roc_auc	hand_till	0.8758595	5	0.0071782	Preprocessor1_Model1

```r
tibble(Model = c("K-Nearest Neighbors", "Logistic Regression", "Elastic Net Logistic Regression",
                 "Linear Discriminant Analysis"),
    ROC_AUC = c(knn_best$mean, log_best$mean, elastic_log_best$mean, lda_best$mean)) %>%
    arrange(desc(ROC_AUC)) %>%
    kable() %>%
    kable_styling(full_width = FALSE) %>%
    scroll_box(width = "100%", height = "200px")
```

Since our desired metric of choice was `ROC_AUC`, the closer the value to 0.90, the better it is! To go more into depth as well, `ROC_AUC` is a method of computing the area under the curve (AUC) for the receiver operating characteristic (ROC) curve. It allows us to compare different ROC curves to determine which model worked best. Looking above to the table of our 4 models with their respective ROC_AUC values, the Elastic Net Logistic Regression model ended up being the one that worked best! We can now fit it with its workflow to the training set.

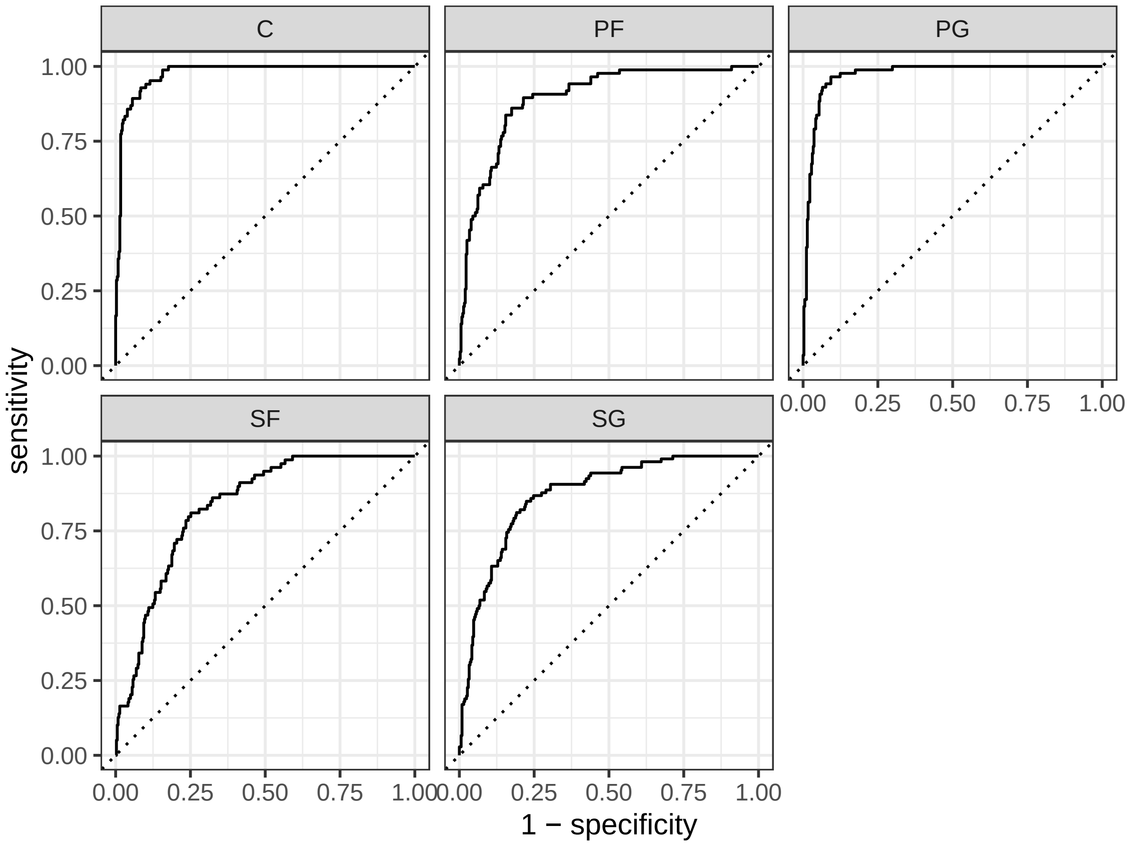
```r
#final fitting:
nba_final_wf <- finalize_workflow(elastic_log_wf, elastic_log_best)

nba_final_fit <- fit(nba_final_wf, nba_train)

augment(nba_final_fit, new_data = nba_test) %>%
    mutate(Pos = factor(Pos)) %>%
    roc_auc(truth = Pos, estimate = .pred_C:.pred_SG) %>%
    kable() %>%
    kable_styling(full_width = FALSE) %>%
    scroll_box(width = "100%", height = "200px")
```

```r
#show curves of each of the five positions
augment(nba_final_fit, new_data = nba_test) %>%
    mutate(Pos = factor(Pos)) %>%
    roc_curve(truth = Pos, estimate = .pred_C:.pred_SG) %>%
    autoplot()
```

neighbors	.metric	.estimator	mean	n	std_err	.config	.best	.bound
10	roc_auc	hand_till	0.86003	5	0.0035546	Preprocessor1_Model10	0.86003	0.8564754

penalty	mixture	.metric	.estimator	mean	n	std_err	.config	.best	.bo
0	0.1111111	roc_auc	hand_till	0.8906415	5	0.0048043	Preprocessor1_Model011	0.8907029	0.8858

Model	ROC_AUC
Elastic Net Logistic Regression	0.8906415
Logistic Regression	0.8906321
Linear Discriminant Analysis	0.8758595
K-Nearest Neighbors	0.8600300



In addition to finalizing our model, we were also able to visualize the ROC curves for each of the five positions! To determine how good a curve is, the closer it is to the top-left corner, the better it is. Notice how that is the case for PG's and C's. In other words, we can expect the model to more accurately predict those positions than PF, SF, and SG. This would answer our question that arose from the EDA portion; the model was ultimately able to better predict positions that have certain statistics more related to it (Point Guards - Assists, Centers - TRB & Blocks).

Finally! We have our model of choice and now it's time to start predicting! I decided to go with 5 All-Star players from each of the 5 respective basketball positions. For PG, I went with Damian Lillard's 2020 season. For SG, Devin Booker's 2020 season. For SF, LeBron James' 2019 season. For PF, Giannis Antetokounmpo's 2021 season and for C, Joel Embiid's 2021 season.

```
#pg: dame, age = 29, ROW: 1408
pg_dame <- subset(NBA_names, NBA_names$Player == 'Damian Lillard' & NBA_names$Year == 2020)
```

.metric	.estimator	.estimate
roc_auc	hand_till	0.9090664

```r
pg_dame <- subset(pg_dame, select=-c(FG, `3P`, `2P`, FT, DR))
predict(nba_final_fit, pg_dame)
```

```
## # A tibble: 1 x 1
##   .pred_class
##   <fct>
## 1 PG
```

```r
#sg: devin book, age = 23, ROW: 1154
sg_dbook <- subset(NBA_names, NBA_names$Player == 'Devin Booker' & NBA_names$Year == 2020)
sg_dbook <- subset(sg_dbook, select=-c(FG, `3P`, `2P`, FT, DR))
predict(nba_final_fit, sg_dbook)
```

```
## # A tibble: 1 x 1
##   .pred_class
##   <fct>
## 1 PG
```

```r
#sf: lebron, age = 35,  ROW: 1363
sf_lebron <- subset(NBA_names, NBA_names$Player == 'LeBron James' & NBA_names$Year == 2019)
sf_lebron <- subset(sf_lebron, select=-c(FG, `3P`, `2P`, FT, DR))
predict(nba_final_fit, sf_lebron)
```

```
## # A tibble: 1 x 1
##   .pred_class
##   <fct>
## 1 PG
```

```r
#pf: giannis, age = 26, ROW: 1643
pf_giannis <- subset(NBA_names, NBA_names$Player == 'Giannis Antetokounmpo' & NBA_names$Year == 2021)
pf_giannis <- subset(pf_giannis, select=-c(FG, `3P`, `2P`, FT, DR))
predict(nba_final_fit, pf_giannis)
```

```
## # A tibble: 1 x 1
##   .pred_class
##   <fct>
## 1 C
```

```r
#c: joel, age = 26, ROW: 1786
c_joel <- subset(NBA_names, NBA_names$Player == 'Joel Embiid' & NBA_names$Year == 2021)
c_joel <- subset(c_joel, select=-c(FG, `3P`, `2P`, FT, DR))
predict(nba_final_fit, c_joel)
```

```
## # A tibble: 1 x 1
##   .pred_class
##   <fct>
## 1 C
```

```r
all_data <- rbind(pg_dame, sg_dbook, sf_lebron, pf_giannis, c_joel) %>%
  tibble()
augment(nba_final_fit, all_data) %>%
  dplyr::select(Player, Pos, .pred_class)
```

```
## # A tibble: 5 x 3
##   Player                Pos   .pred_class
##   <chr>                 <chr> <fct>
## 1 Damian Lillard        PG    PG
## 2 Devin Booker          SG    PG
## 3 LeBron James          SF    PG
## 4 Giannis Antetokounmpo PF    C
## 5 Joel Embiid           C     C
```

CONCLUSION

Outside of PG and C, the other 3 positions were incorrectly predicted. This would make sense though after seeing their ROC curves and noticing that it's not as good compared to PG and C; compounded with the fact that they have specific statistics corresponding to their position, it's no surprise that they were the two positions that the model correctly predicted.

Overall, this project was extremely fun to do and super insightful into the machine learning process. I did not expect all of my models to do so well but that was the case since all of their ROC_AUC values were in the high 0.80s with the best model, Elastic Net Logistic Regression, having a value of 0.8906415. The next step would probably be to try incorporating other models such as Random Forest to try to get a better fit. Going forward, I hope to incorporate all the knowledge that I learned from this project and class into my future career. Cheers!