131 Final Project

Caleb Mazariegos

2022-06-10

Contents

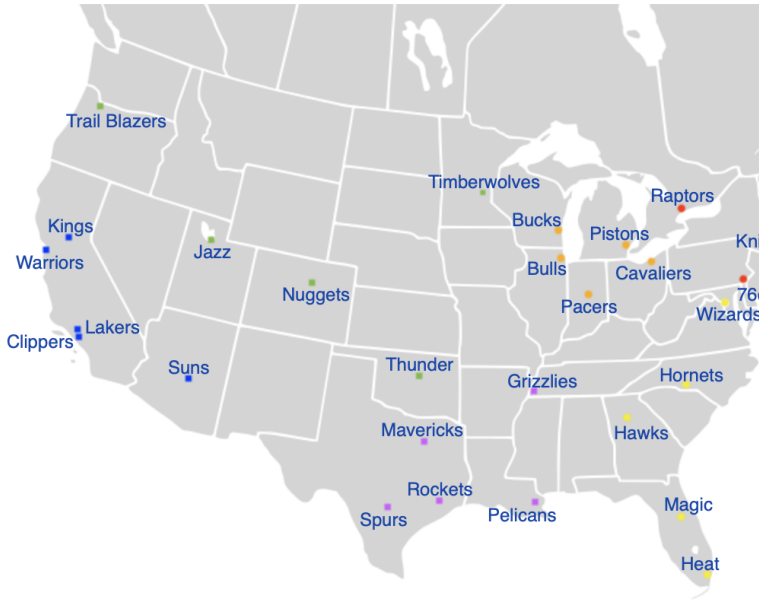
Introduction

The purpose of this project is to generate a model that will predict the average points per game an NBA player will make. I am using data from the 2020 - 2021 regular NBA season.

What is the NBA?

The NBA is an abbreviation for the National Basketball Association. It is a professional men's basketball league in North America. The league consists of 30 teams.

A regular basketball season runs from October to April, with playoffs extending into June.



A map displaying the NBA teams organized by conference:

source: https://en.wikipedia.org/wiki/National_Basketball_Association

Diagram of Basketball Court

An image of half of an NBA Basketball Court:

source: https://en.wikipedia.org/wiki/Basketball

A 2-pointer is a shot that is scored anywhere inside of the arc. A 3-pointer is a shot made anywhere outside of the arc. Foul shots from the free-throw line count for 1 point.

Basketball Positions

There are 5 players from each team on the court during a game. There will usually be a combination of the following positions:

- **Center (C)** - On offense, the center tries to score on close shots and rebounds. On Defense, the center blocks opponents' shots and rebounds. Usually the tallest and strongest player on the team.
- **Center Forward (CF)** - Players who play or have played both forward and center on a consistent basis

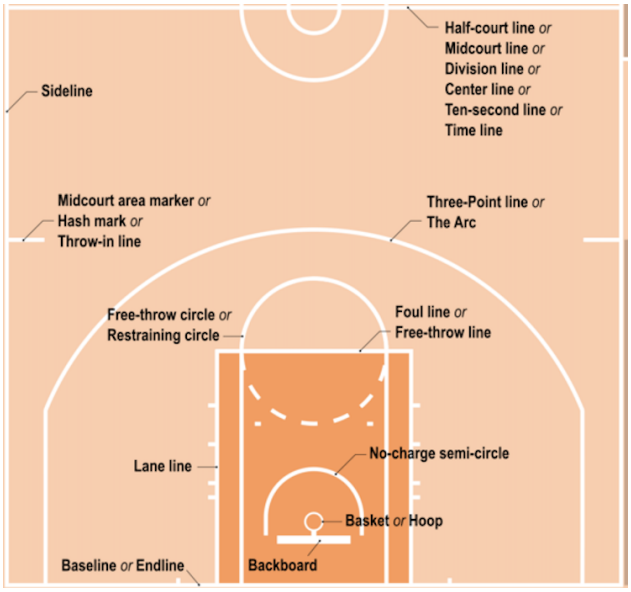


Figure 1: an image caption Source: Half of a basketball court

- **Forward (F)** - Forwards are able to play both inside the paint and outside of it on offense and should be aggressive and rebound-minded on defense.
- **Power Forward (F-C)** - The power forward does many of the things a center does, playing near the basket while rebounding and defending taller players. But power forwards also take longer shots than centers.
- **Shooting Guard (F-G)** - The shooting guard is usually the team's best shooter. The shooting guard can make shots from long distance and also is a good dribbler.
- **Point Guard (G)** - Usually the best dribbler and passer, the point guard defends the opponent's point guard and tries to steal the ball
- **Guard Forward (G-F)** - A combination between a forward and shooting guard

Rundown of Basketball Statistics

Here are a list of statistics that I will be referencing in my project

- **Games Played** `gp` - Total number of games a player has played for the season
- **Minutes Per Game** `mpg` - The average number of minutes a player has played per game
- **Minutes Percentage** `min_percentage` - Percentage of team minutes used by a player while he was on the floor
- **Usage Percentage** `usg` - The percentage of team plays used by a player when they are on
- **Turnover Rate** `to_percentage` - A metric that estimates the number of turnovers a player commits per 100 possessions
- **Free Throws Attempted** `fta` - The number of free throws that a player has attempted
- **Free Throw Percentage** `ft_percentage` - The percentage of free throw attempts that a player has made successfully

- **2 Point Field Goals Attempted** `x2pa` - The number of two pointers that a player has attempted
- **2 Point Field Goals** `x2p_percentage` - The percentage of points scored by a player that are from 2 point field goals
- **3 Point Field Goals Attempted** `x3pa` - The number of 3 point field goals that a player has attempted
- **3 Point Field Goals** `x3p_percentage` - The percentage of points scored by a player that are from 3 point field goals
- **Effective Field Goal Percentage** `e_fg` - Measures field goal percentage adjusting for made 3-point field goals being 1.5 times more valuable than made 2-point field goals.
- **True Shooting Percentage** `ts_percent` - A shooting percentage that factors in the value of three-point field goals and free throws in addition to conventional two-point field goals
- **Rebounds Per Game** `rpg` - The number of rebounds a player makes per game
- **Total Rebound Percentage** `trb_percent` - The estimated percentage of available rebounds grabbed by the player while the player is on the court
- **Assists Per Game** `apg`- The average number of assists a player makes per game
- **Assist Percentage** `ast_percentage` - The estimated percentage of teammate field goals a player assisted while the player is on the court
- **Steals Per Game** `spg` - The average number of steals a player makes per game
- **Blocks Per Game** `bpg` - The average number of blocks a player makes per game
- **Turnovers Per Game** `topg` - The average number of turnovers a player makes per game, it is when a team loses possession of the ball to the opposing team
- **Versatility Index**`vi_versatility`- A metric that measures a player's ability to produce in points, assists, and rebounds. The average player will score around a 5 on the index, while top players score above a 10
- **Offensive Rating** `ortg_offensive` - The number of points produced by a player per 100 total individual possessions
- **Defensive Rating** `drtg_defensive` - The number of points the player allowed per 100 possessions he individually faced while staying on the court

In addition to the statistics mentioned above, I will also be using the following variables in my report: * `age` : The age of the player

- `pos` - The position that of the player (F, G, C-F, F, F-C, F-G, G, G-F)

I will not be using the team, as I do not think it is relevant.

Data Cleaning

- Cleaning the names and loading the data set:

```
# loading and cleaning data set
basketball_codebook <- read.csv("/Users/calebmazariegos/Desktop/2020 - 2021 Basketball Data Set - Sheet1

basketball_codebook <- clean_names(basketball_codebook)

basketball_codebook <- na.omit(basketball_codebook)
```

- Removing unnecessary variables and displaying the first 3 observations:

```
options(width = 100)

basketball_codebook <- basketball_codebook %>%
  select(-team, -full_name)

head(basketball_codebook, n = 3)
```

```
##   pos   age gp  mpg min_percentage  usg to_percentage fta ft_percentage x2pa x2p_percentage x3pa
## 1   F 21.66 61 12.1           25.2 19.5          13.5 110         0.509  227          0.546    1
## 3   C 27.83 58 27.7           57.6 11.7          17.7 135         0.444  305          0.620    3
## 4 C-F 23.83 64 33.5           69.7 23.7          15.0 354         0.799  792          0.573    8
##   x3p_percentage  e_fg ts_percent  ppg rpg trb_percent apg ast_percentage  spg  bpg topg
## 1           0.00 0.544      0.550  5.0 3.4        16.1 0.5            6.1 0.33 0.46 0.70
## 3           0.00 0.614      0.596  7.6 8.9        17.4 1.9            9.1 0.93 0.66 1.36
## 4           0.25 0.571      0.626 18.7 9.0        15.3 5.4           26.9 1.17 1.03 2.64
##   vi_versatility ortg_offensive drtg_defensive
## 1            6.7          106.8           99.7
## 3            7.3          119.7          107.8
## 4           11.6          121.7          105.0
```

- Setting `pos` as a factor:

```
basketball_codebook$pos <- as.factor(basketball_codebook$pos)
```

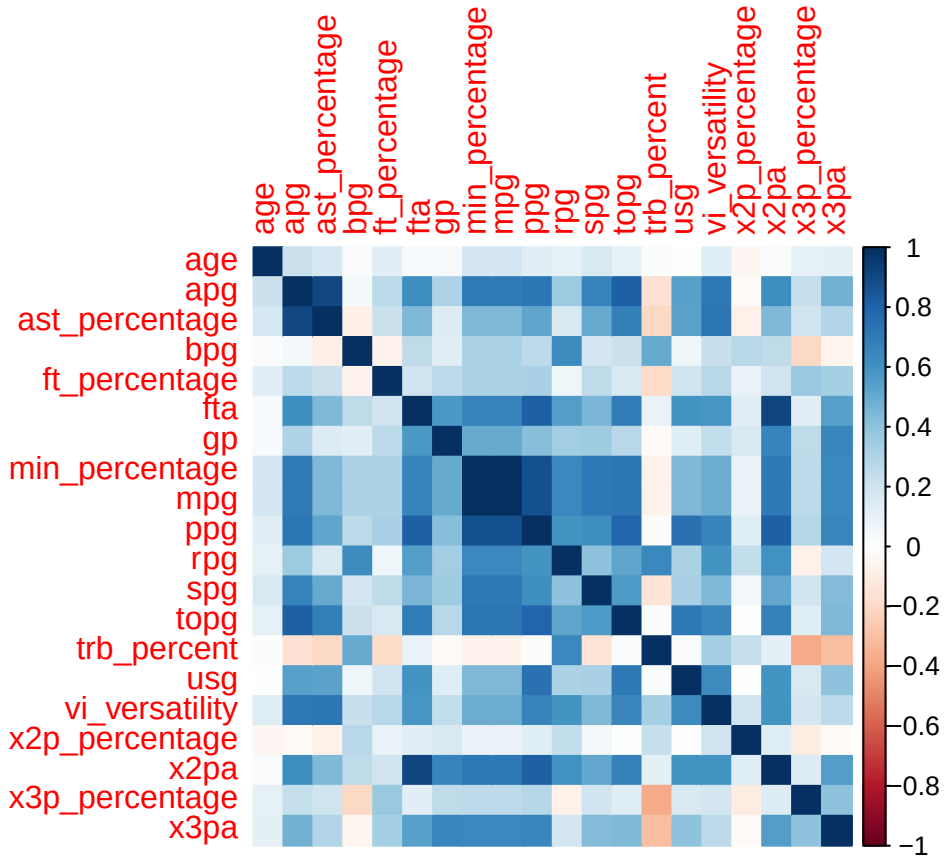
Exploratory Data Analysis

The exploratory data analysis is based on 626 observations. Each observation represents a single NBA player from the 2020 - 2021 season.

Correlation between variables

removing non-numeric variables and variables with NA correlation to clean up plot:

```
# removing non-numeric variables and variables with NA values
basketball_codebook_continuous <- basketball_codebook %>%
  select(-pos, -drtg_defensive, -e_fg, -ortg_offensive, -to_percentage, -ts_percent)
M = cor(basketball_codebook_continuous)
corrplot(M, method = "color", order = "alphabet")
```

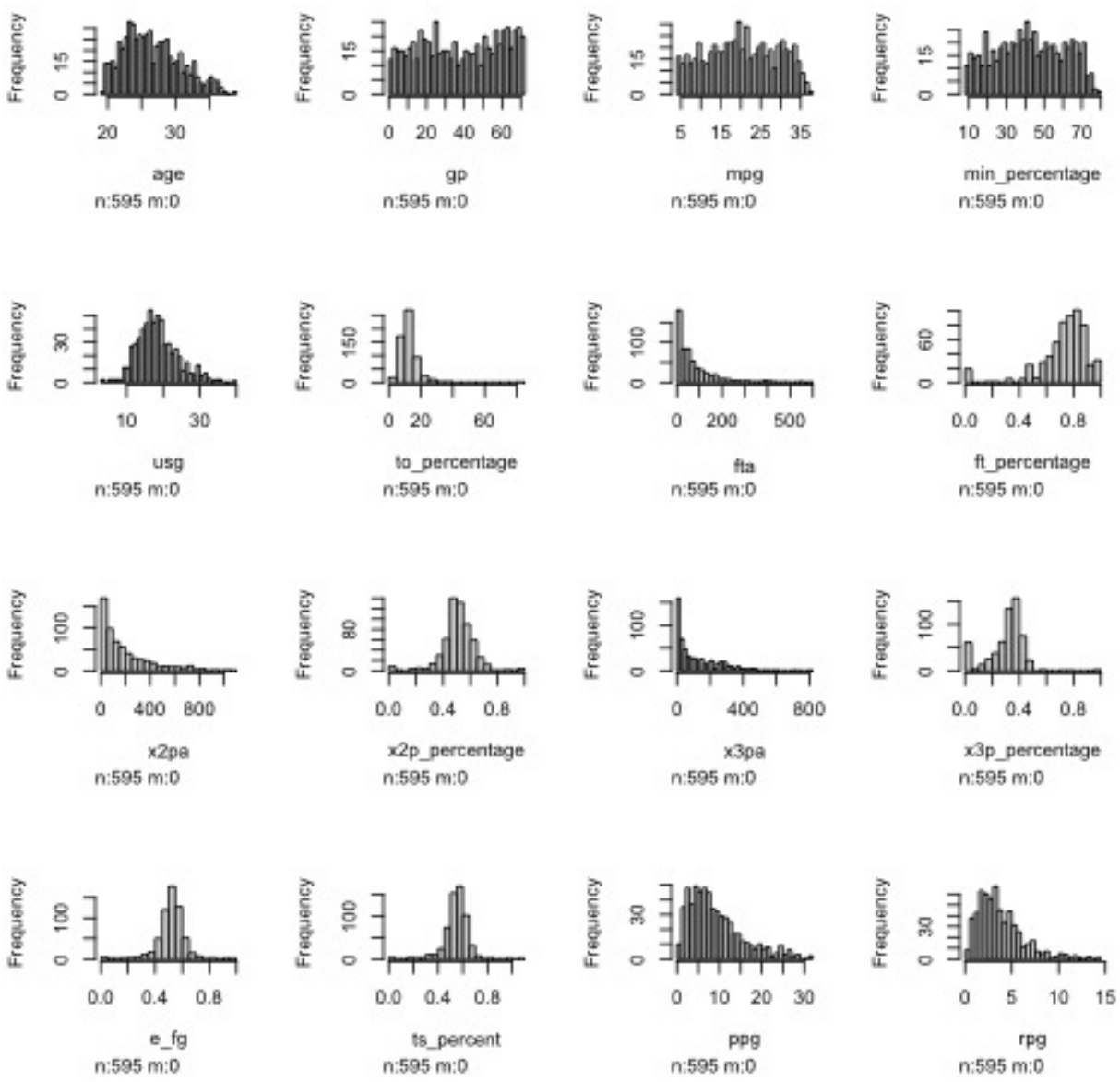


Distribution of continuous variables

I saved the histogram as a jpeg because the plot did not fit all of the histograms correctly

```
df1 <- basketball_codebook[,c('age','gp','mpg', 'min_percentage', 'usg', 'to_percentage', 'fta', 'ft_pe

jpeg(file = "saving_plot1.jpeg")
hist.data.frame(df1, main = "Histograms of all Continuous Variables")
dev.off()
```

```
## pdf
##   2
```

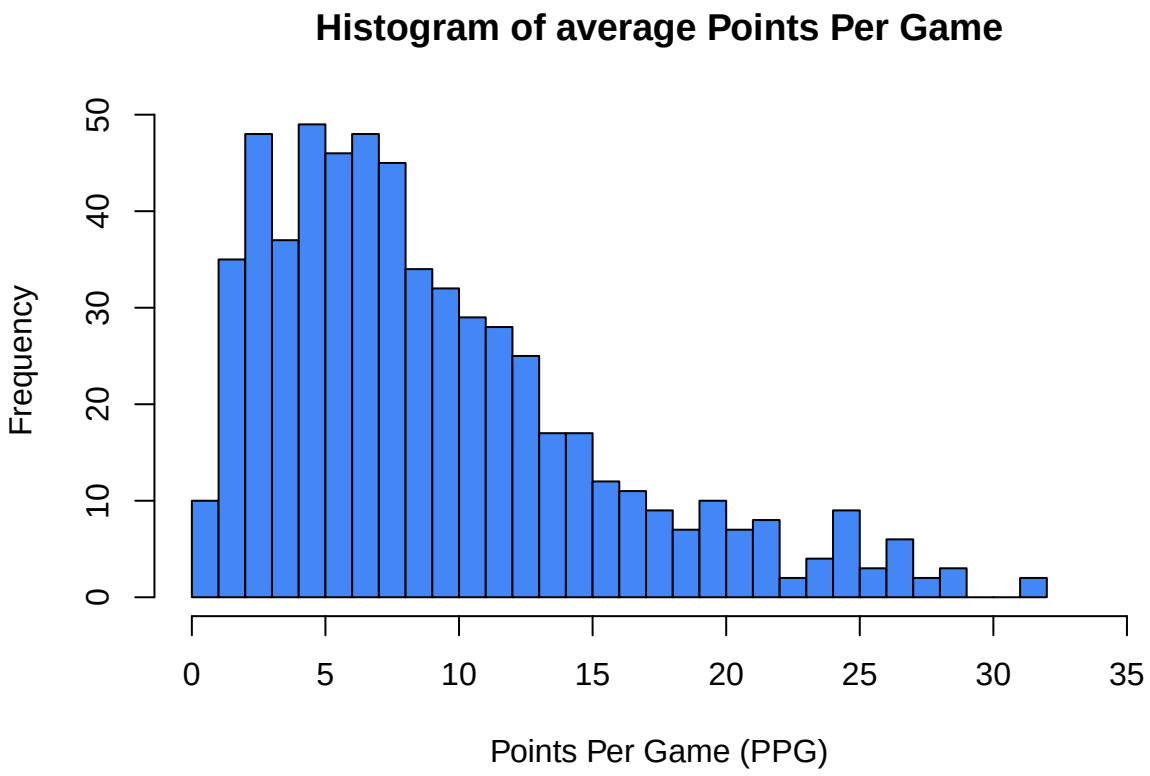


Now it is easy to see which variables are skewed left, skewed right, and which ones have normal and uniform distributions.

Histogram of average Points Per Game

Let's take a closer look at the distribution of the average points per game:

```
hist(basketball_codebook$ppg, main = "Histogram of average Points Per Game", xlab = "Points Per Game (PP
     xlim = c(0,35), breaks = 25)
```

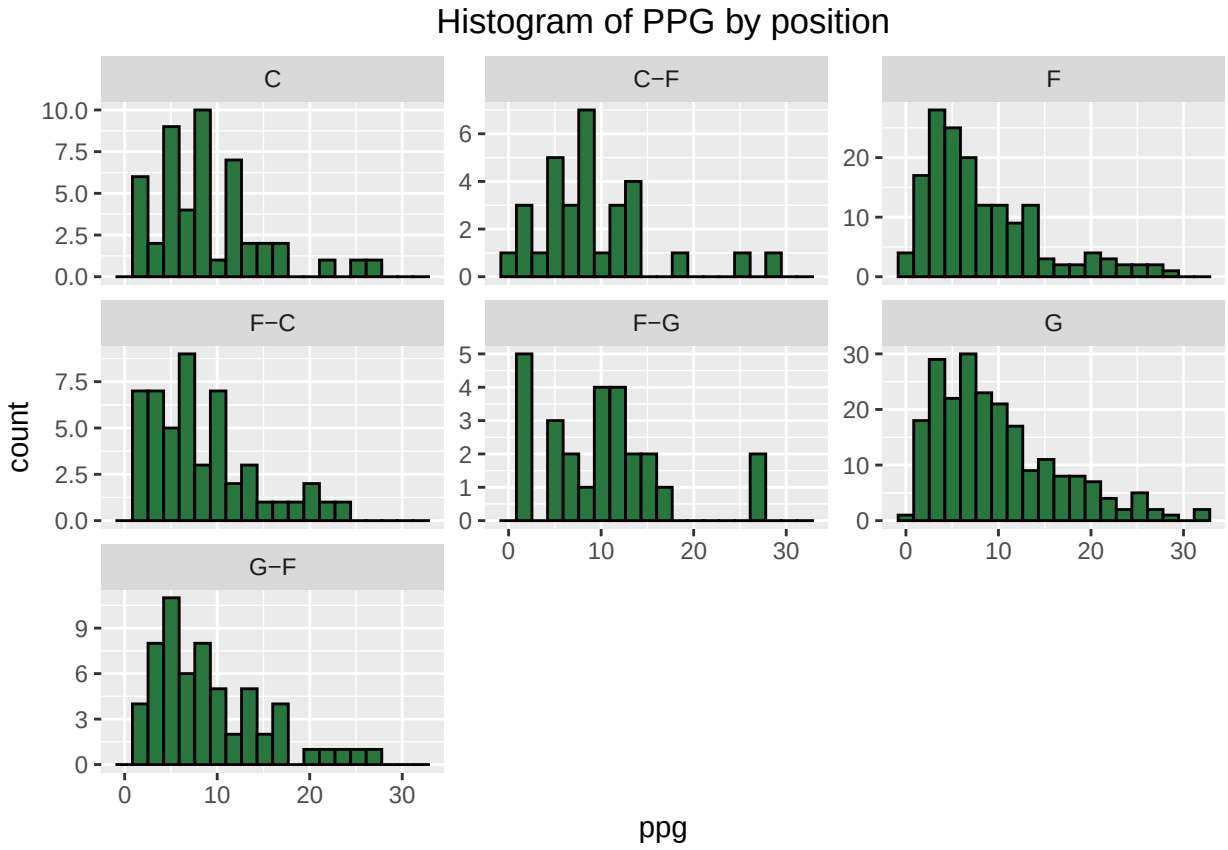


There is a rightward skew, this means that most players tend to score around 3 - 7 points per game.

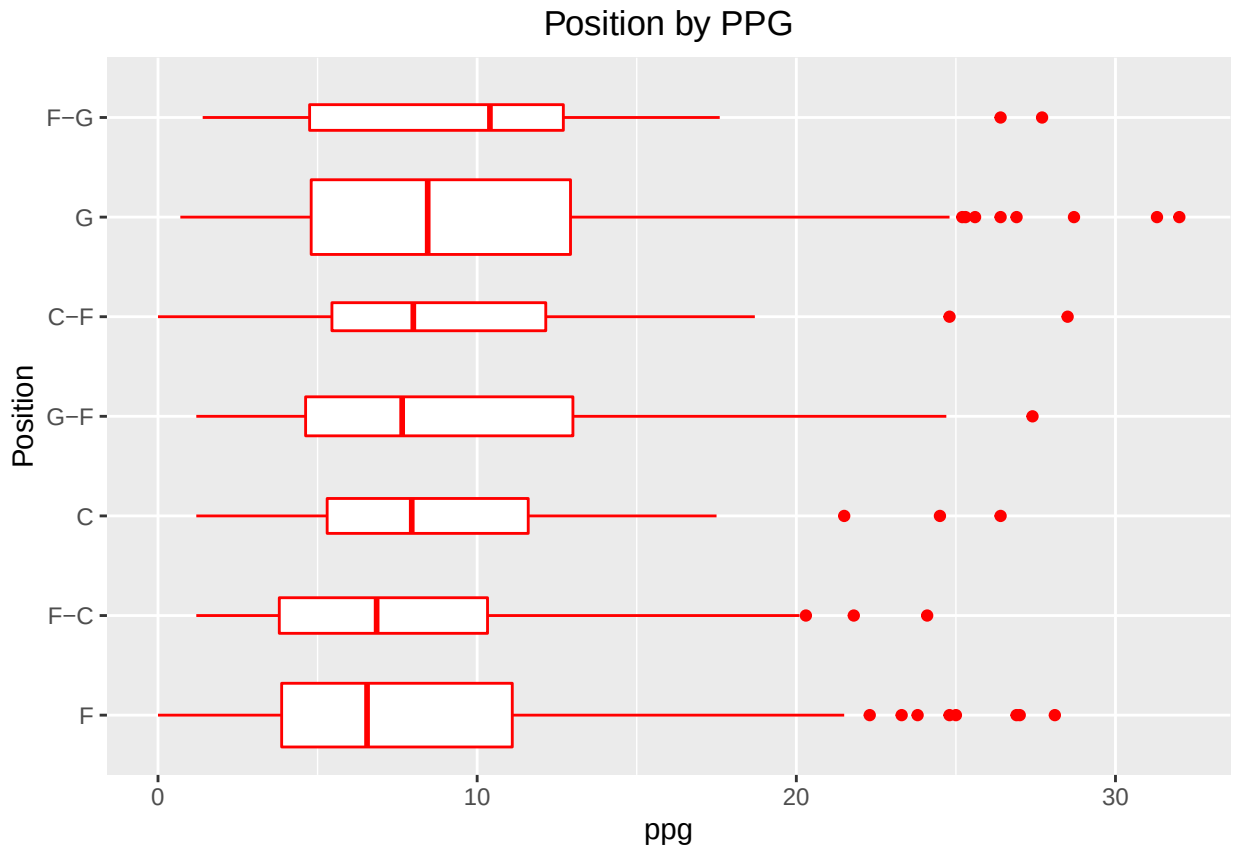
Position

I hypothesize that the position of the player affects how many points per game the player will average because some positions might be better suited to shooting goals. I will break this down further by looking into how the position of the player affects how many points per game they score:

```
ggplot(basketball_codebook, aes(ppg)) +
  geom_histogram(bins = 20, color = "black", fill = "#28753d") +
  facet_wrap(~pos, scales = "free_y") +
  labs(
    title = "Histogram of PPG by position"
  ) +  theme(plot.title = element_text(hjust = 0.5))
```



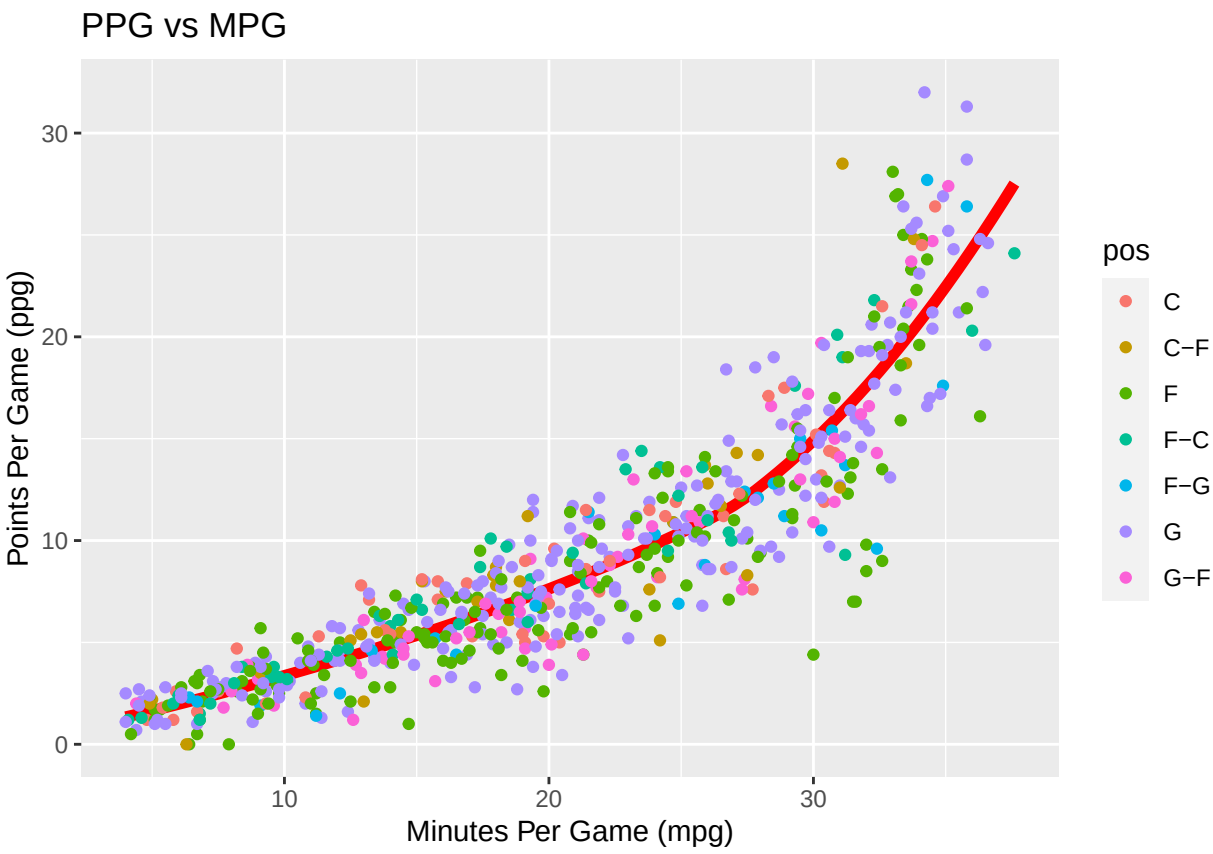

```
ggplot(basketball_codebook, aes(reorder(pos, ppg), ppg)) +
  geom_boxplot(varwidth = TRUE, color = "red", fill = "white") +
  coord_flip() +
  labs(
    title = "Position by PPG",
    x = "Position"
  ) + theme(plot.title = element_text(hjust = 0.5))
```



It looks like most positions have the same average, which is between 5 - 10 goals. Point Guards (G) and Forwards tend to have the most outliers (F). As expected, Shooting Guards (F-G) have the highest average.

Now, let's examine how the number of games played impacts the average number of minutes per game.

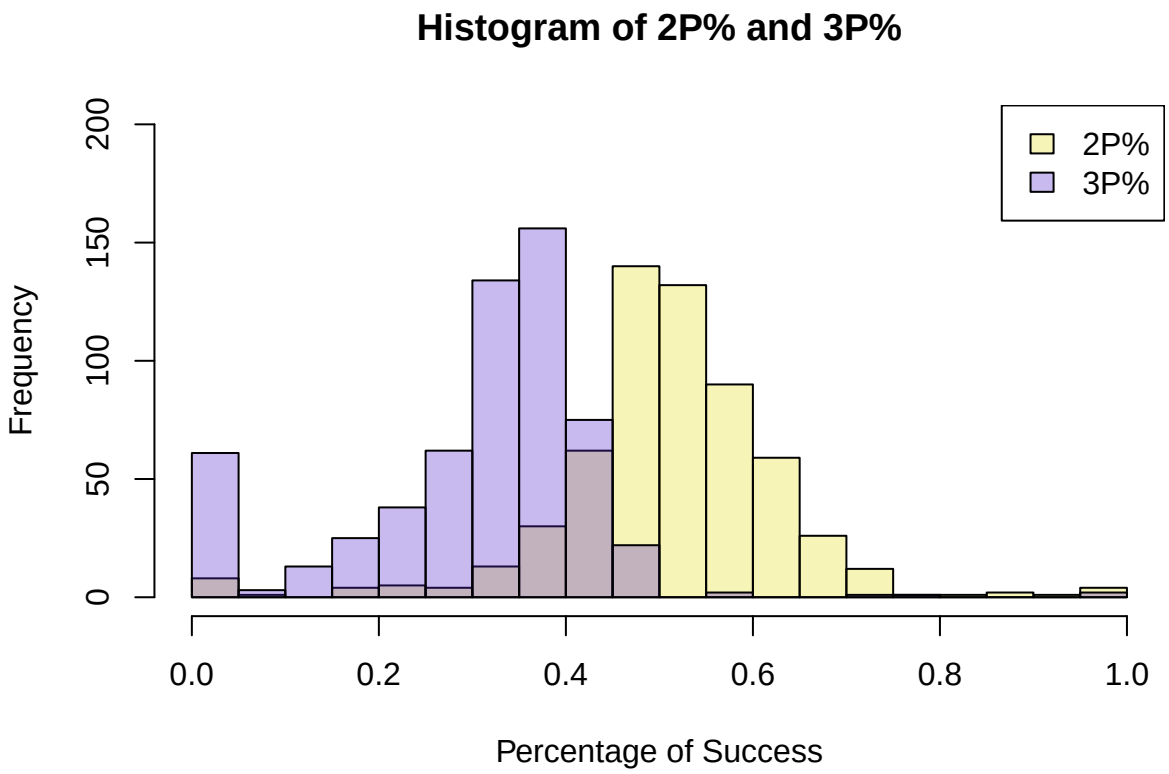
```
ggplot(basketball_codebook, aes(mpg, ppg, colour = pos)) + labs(x = "Minutes Per Game (mpg)", y = "Poin
  geom_smooth(se = FALSE, color = "red", size = 2) +geom_point()
```



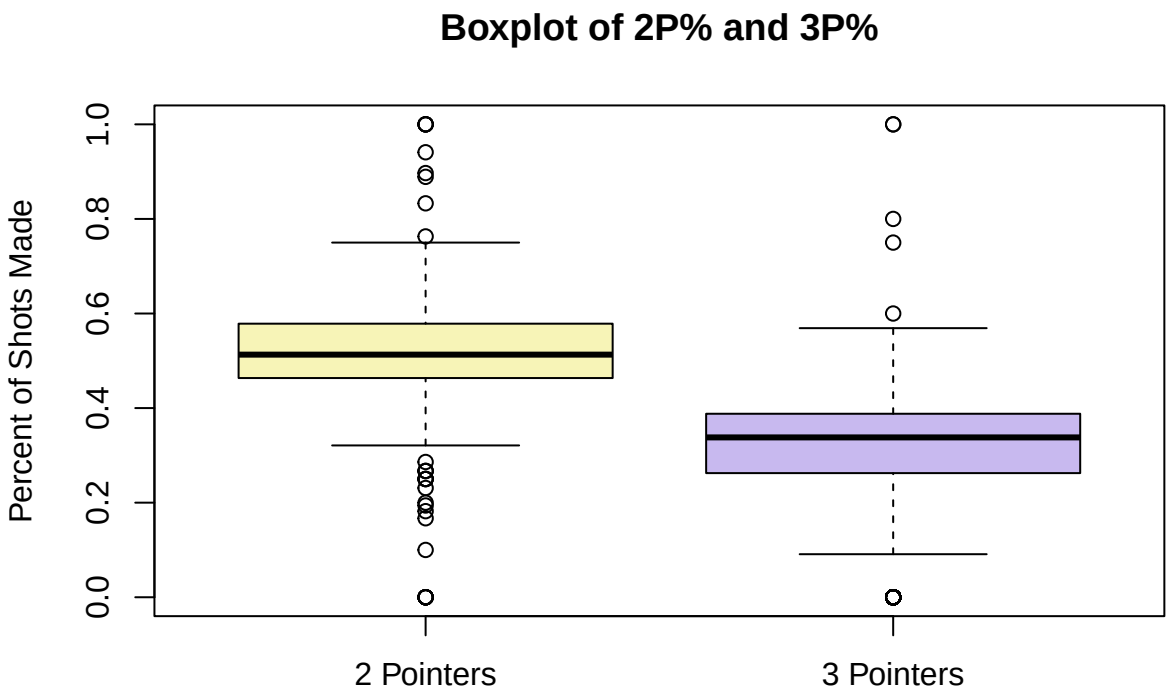
The relationship between Minutes Per Game `mpg` and `points per game` seems to be positive. This means that if the player has a higher average minutes per game, they will likely have more points per game. This makes sense since they will have more time on the court to have a chance to shoot.

3-pointers vs 2-pointers

```
library(scales)
c1 <- rgb(80, 35, 204,max = 255, alpha = 80, names = "lt.blue")
c2 <- rgb(230, 223, 28, max = 255, alpha = 80, names = "lt.pink")
hgA <- hist(basketball_codebook$x2p_percentage, breaks = 20,plot = FALSE)
hgB <- hist(basketball_codebook$x3p_percentage, breaks = 20,plot = FALSE)

plot(hgA, col = c2, ylim = c(0,200), main = "Histogram of 2P% and 3P%", xlab = "Percentage of Success",
plot(hgB, col = c1, add = TRUE)


legend("topright", c("2P%", "3P%"), fill=c(c2, c1))
```



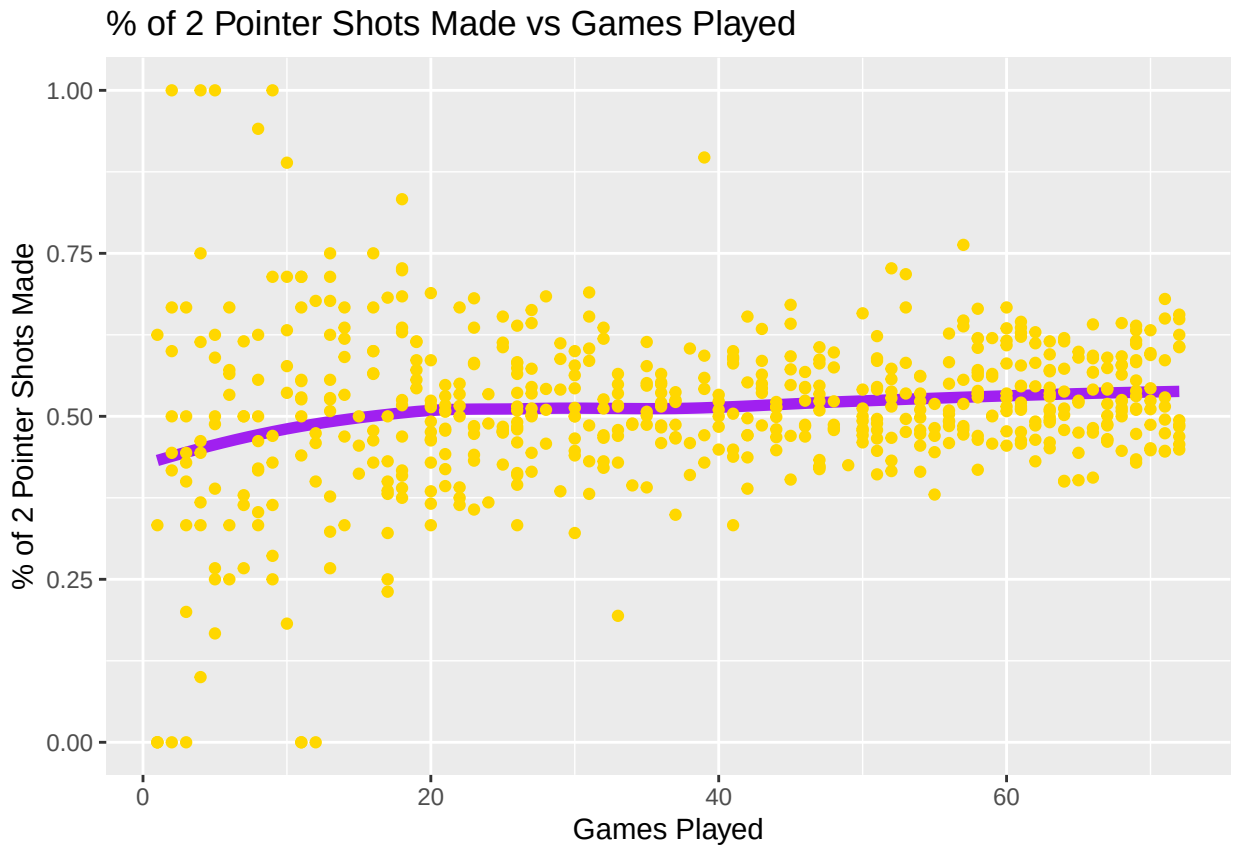

```
boxplot(basketball_codebook$x2p_percentage, basketball_codebook$x3p_percentage, col = c(c2, c1), names =
```



It appears that 2 point field goals seem to have a higher percentage of success. There also appears to be some outliers of high point accuracy which may be explained by only being in a small amount of games. I will explore this further by looking at shot accuracy and number of games played.

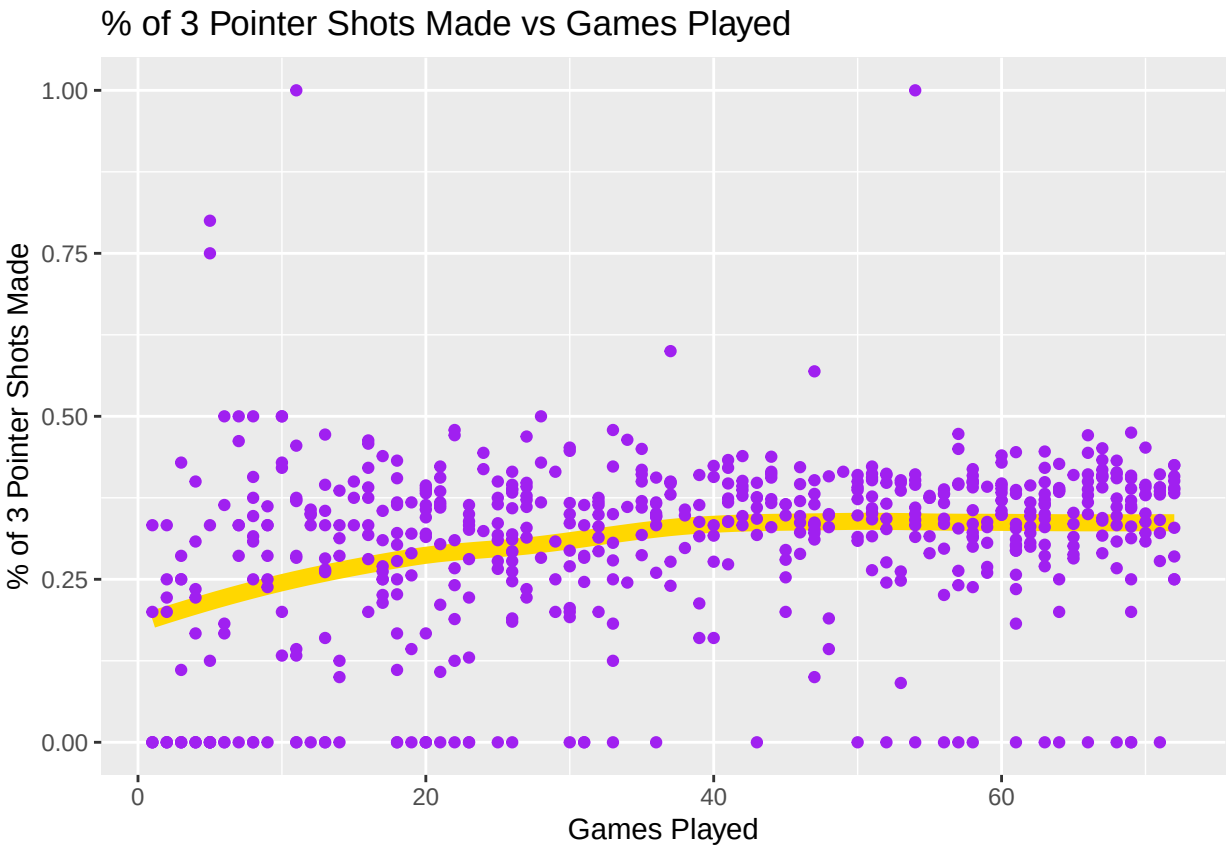
```
ggplot(basketball_codebook, aes(gp, x2p_percentage)) +geom_smooth(se = FALSE, color = "purple", size = 1
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




```
ggplot(basketball_codebook, aes(gp, x3p_percentage)) + geom_smooth(se = FALSE, color = "gold", size = 3
  labs(title = "% of 3 Pointer Shots Made vs Games Played", x = "Games Played", y= "% of 3 Pointer Shots
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



When it comes to 2 Point field goals made, it looks like most of the outliers can be attributed to the low number of games played. After about 20 games played, the percentage of shots made seems to taper between 0.3 - 0.6.

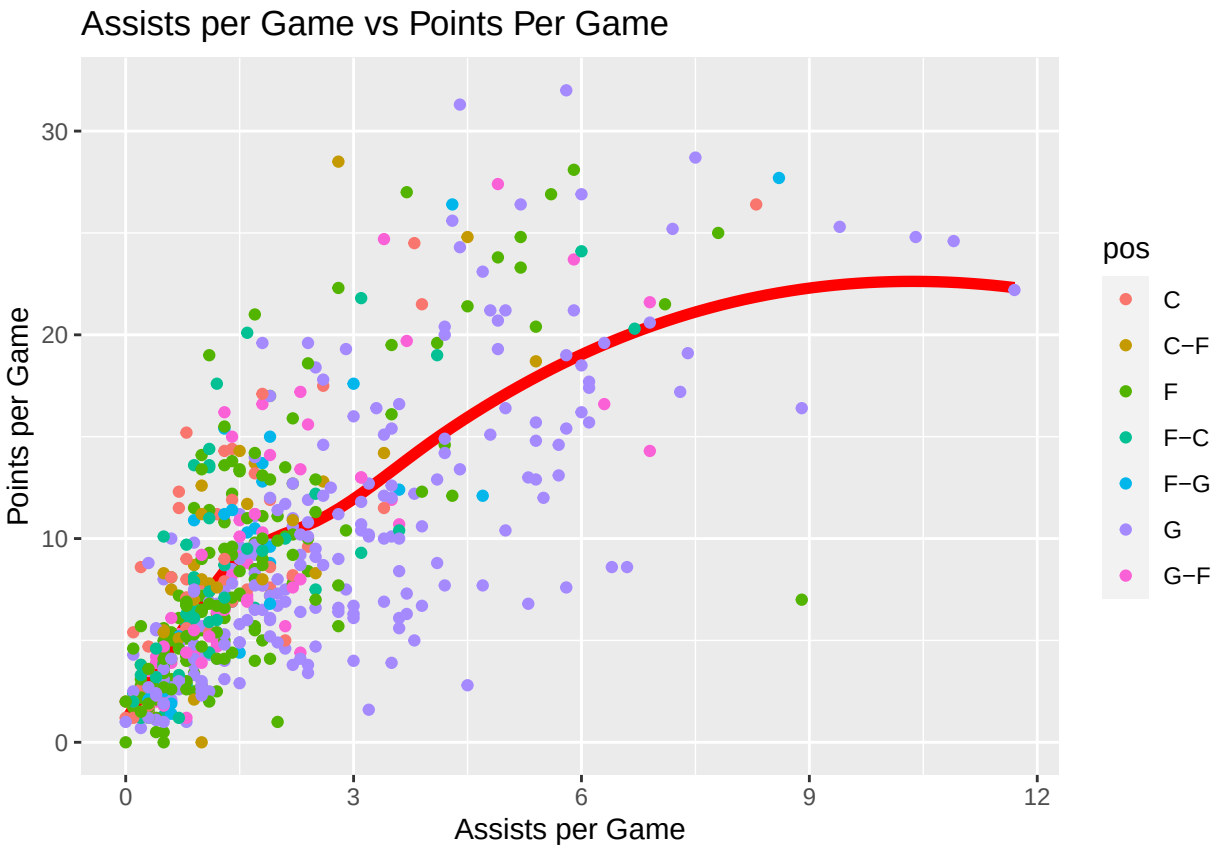
Since 3 point field goals are more difficult to score, it makes sense that there are players with 0% 3 pointers scored, even with 40+ games played. This distribution is not as clean as the 2 Point field goals.

Relationship between Assists, Blocks, Steals, Rebounds and Points per Game

ggplot(basketball_codebook, aes(apg, spg))+ geom_smooth(se = FALSE, color = "red", size = 2) +geom_point()

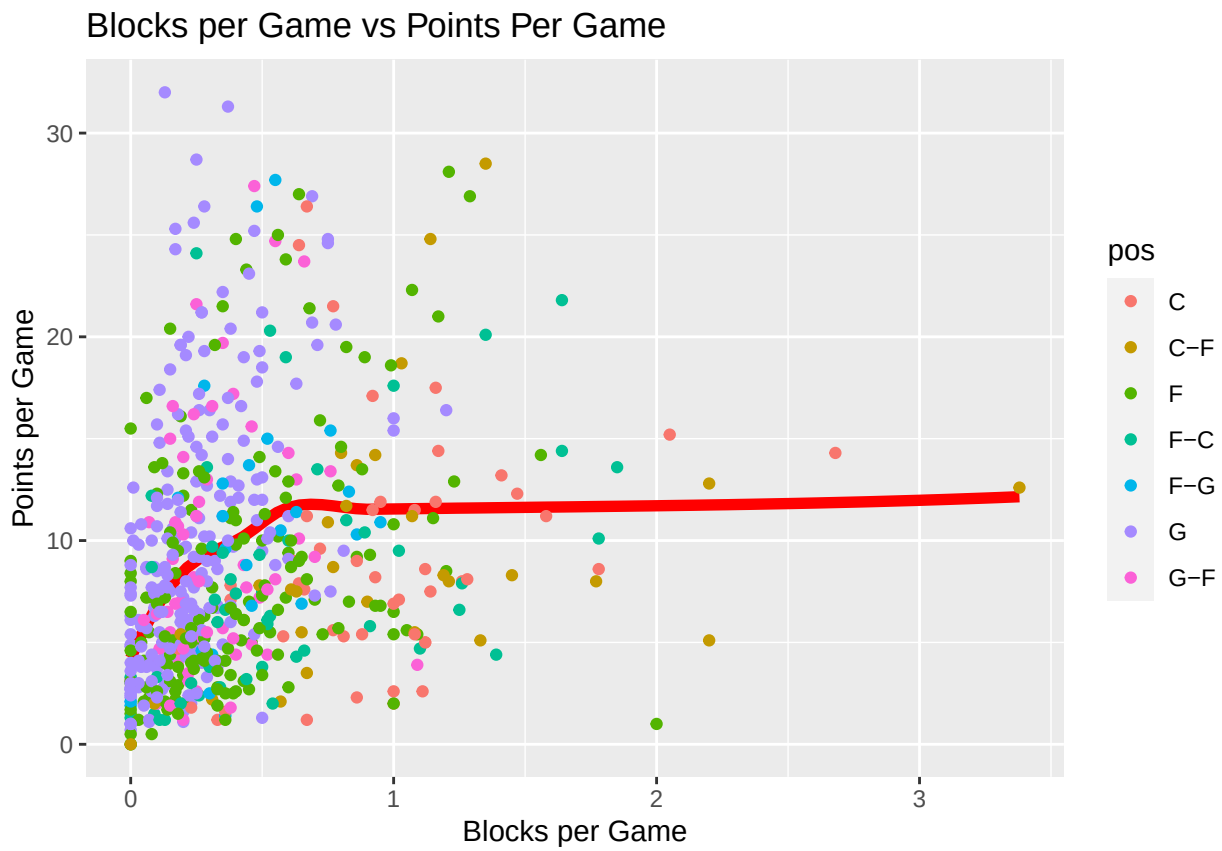
```
ggplot(basketball_codebook, aes(apg, ppg, color = pos))+
  geom_smooth(se = FALSE, color = "red", size = 2) + labs(title = "Assists per Game vs Points Per Game"
                                                          x = "Assists per Game", y = "Points per Game"
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



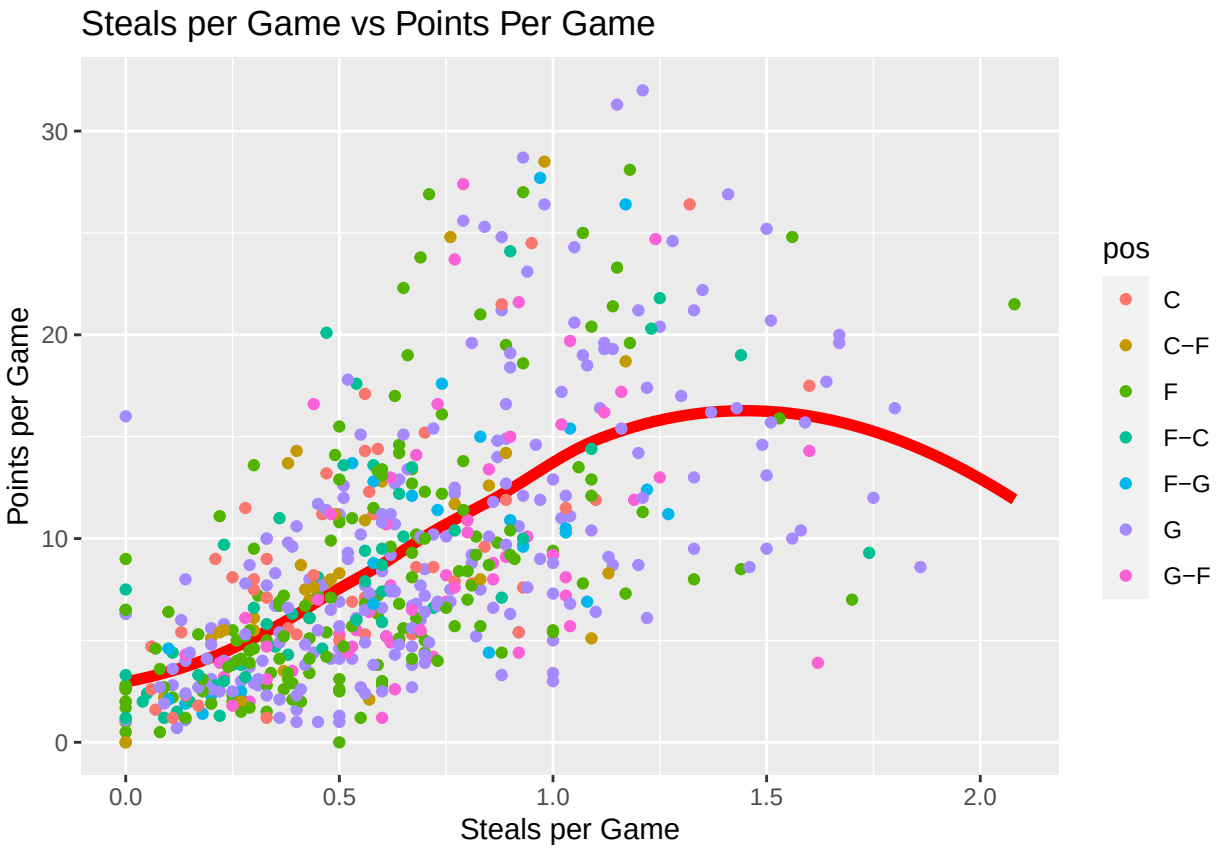

```
ggplot(basketball_codebook, aes(bpg, ppg, color = pos))+
  geom_smooth(se = FALSE, color = "red", size = 2) + labs(title = "Blocks per Game vs Points Per Game",
                                                          x = "Blocks per Game", y = "Points per Game")+
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



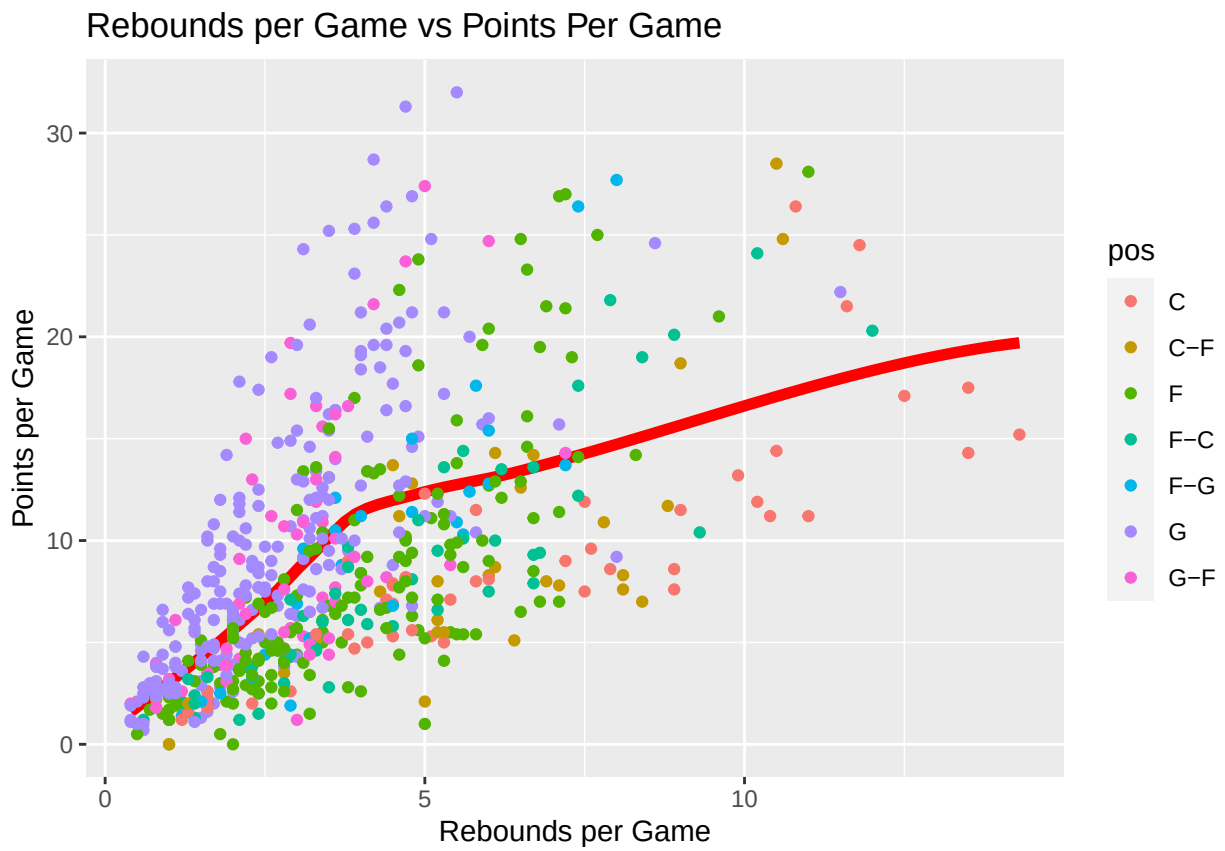

```
ggplot(basketball_codebook, aes(spg, ppg, color = pos))+
  geom_smooth(se = FALSE, color = "red", size = 2) + labs(title = "Steals per Game vs Points Per Game",
                                                          x = "Steals per Game", y = "Points per Game")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




```
ggplot(basketball_codebook, aes(rpg, ppg, color = pos))+
  geom_smooth(se = FALSE, color = "red", size = 2) + labs(title = "Rebounds per Game vs Points Per Game
                                                          x = "Rebounds per Game", y = "Points per Game
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



For Assists, Steals, and Rebounds, there seems to be a positive relationship with how many points per game the player scores. There does not seem to be a very obvious correlation with Blocks, but it does appear that Point Guards tend to have less blocks per game, but the most points per game.

Data Split

The data was split in a 70% training and 30% testing split.

```
set.seed(123)
basketball_split <- basketball_codebook %>%
  initial_split(prop = 0.7, strata = "ppg")

basketball_train <- training(basketball_split)
basketball_test <- testing(basketball_split)
```

The training data set has about 440 observations and the testing data set has about 185.

Model Building

Creating a recipe

We want to create a recipe to represent the model we are fitting

```
basketball_recipe <- recipe(ppg ~ pos + age + gp + mpg + min_percentage + usg + to_percentage + fta + f
                            + x2pa + x2p_percentage + x3pa + x3p_percentage + e_fg + ts_percent + rpg +
                              apg + ast_percentage + spg + bpg + topg + vi_versatility + ortg_offensive
  step_dummy(all_nominal_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_normalize(all_predictors()) %>%
  step_novel(all_nominal_predictors()) %>%
  step_nzv(all_nominal_predictors())

basketball_recipe <- na.omit(basketball_recipe)
```

Folds

Stratified cross-validation with 10 folds and 5 repeats:

```
basketball_folds <- vfold_cv(basketball_train, strata = ppg, v = 10, repeats = 5)
na.omit(basketball_folds)
```

```
## #  10-fold cross-validation repeated 5 times using stratification
## # A tibble: 50 x 3
##    splits           id      id2
##    <list>           <chr>   <chr>
##  1 <split [371/44]> Repeat1 Fold01
##  2 <split [371/44]> Repeat1 Fold02
##  3 <split [372/43]> Repeat1 Fold03
##  4 <split [374/41]> Repeat1 Fold04
##  5 <split [374/41]> Repeat1 Fold05
##  6 <split [374/41]> Repeat1 Fold06
##  7 <split [374/41]> Repeat1 Fold07
##  8 <split [375/40]> Repeat1 Fold08
##  9 <split [375/40]> Repeat1 Fold09
## 10 <split [375/40]> Repeat1 Fold10
## # ... with 40 more rows
```

```
head(basketball_folds)
```

```
## # A tibble: 6 x 3
##   splits           id      id2
##   <list>           <chr>   <chr>
## 1 <split [371/44]> Repeat1 Fold01
## 2 <split [371/44]> Repeat1 Fold02
## 3 <split [372/43]> Repeat1 Fold03
## 4 <split [374/41]> Repeat1 Fold04
## 5 <split [374/41]> Repeat1 Fold05
## 6 <split [374/41]> Repeat1 Fold06
```

Models

I will conduct my models in the following way: - Specifying the model - Setting up the workflow - Tuning - Selecting the best values - Fitting the model to the testing set - Making predictions and creating visualizations

Ridge Regression

Ridge regression is one of the types of regularization modeling. As λ increases, bias increases and variance decreases. Ridge regression minimizes the sum of squared residuals as well as $\lambda\times$ the slope^2

Specifying the model type and engine: setting `mixture = 0` to specify ridge regression.

```
ridge_spec <- linear_reg(penalty = tune(), mixture = 0) %>%
  set_mode("regression") %>%
  set_engine("glmnet")
```

In this step, I am setting up the workflow and adding the `basketball_recipe`:

```
ridge_workflow <- workflow() %>%
  add_recipe(basketball_recipe) %>%
  add_model(ridge_spec)
```

Creating a grid with the values of penalty that we are trying, and tuning:

```
set.seed(24)

penalty_grid <- grid_regular(penalty(range = c(1, 11)), levels = 50)
penalty_grid
```

```
## # A tibble: 50 x 1
##    penalty
##      <dbl>
##  1    10
##  2    16.0
##  3    25.6
##  4    40.9
##  5    65.5
##  6   105.
##  7   168.
##  8   268.
##  9   429.
## 10   687.
## # ... with 40 more rows
```

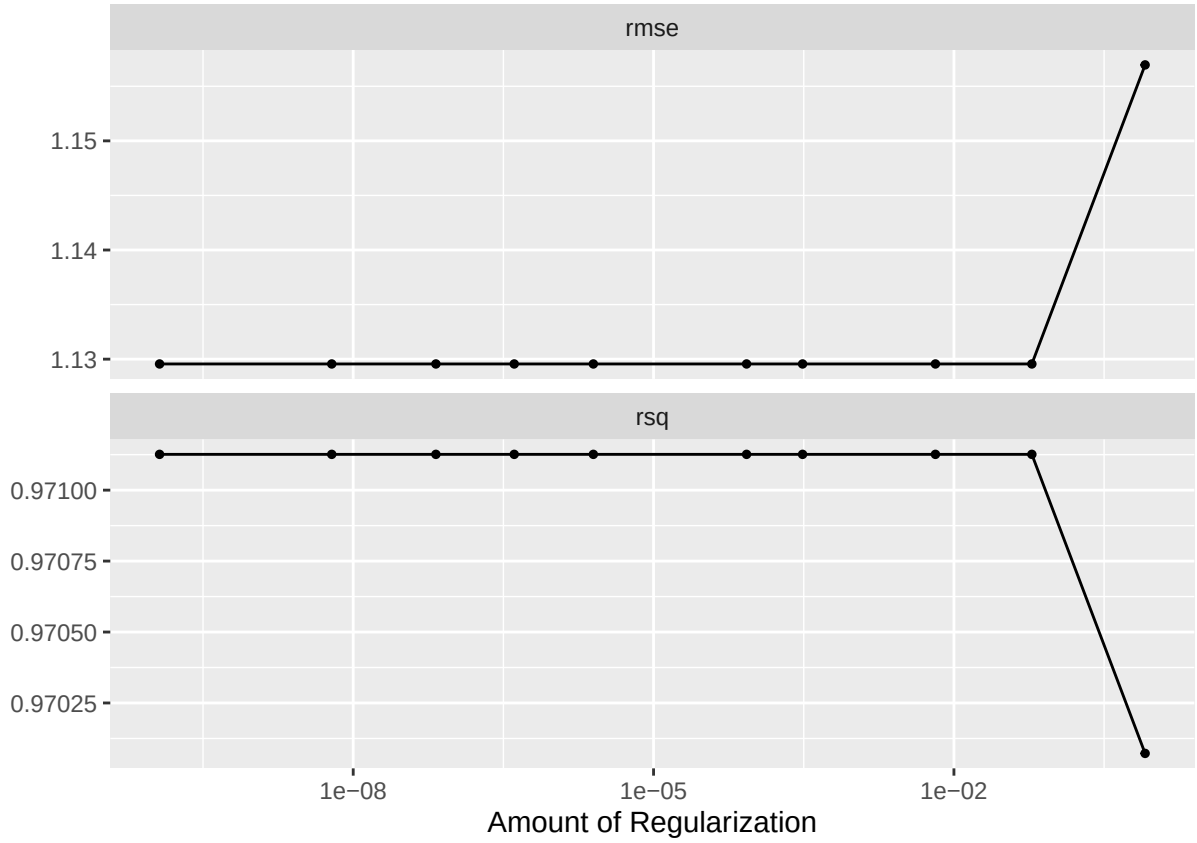
```
tune_res <- tune_grid(
  ridge_workflow,
  resamples = basketball_folds)
tune_res
```

```
## # Tuning results
## # 10-fold cross-validation repeated 5 times using stratification
## # A tibble: 50 x 5
##    splits           id      id2    .metrics          .notes
##    <list>           <chr>   <chr>  <list>            <list>
##  1 <split [371/44]> Repeat1 Fold01 <tibble [20 x 5]> <tibble [0 x 3]>
##  2 <split [371/44]> Repeat1 Fold02 <tibble [20 x 5]> <tibble [0 x 3]>
##  3 <split [372/43]> Repeat1 Fold03 <tibble [20 x 5]> <tibble [0 x 3]>
```

```
##  4 <split [374/41]> Repeat1 Fold04 <tibble [20 x 5]> <tibble [0 x 3]>
##  5 <split [374/41]> Repeat1 Fold05 <tibble [20 x 5]> <tibble [0 x 3]>
##  6 <split [374/41]> Repeat1 Fold06 <tibble [20 x 5]> <tibble [0 x 3]>
##  7 <split [374/41]> Repeat1 Fold07 <tibble [20 x 5]> <tibble [0 x 3]>
##  8 <split [375/40]> Repeat1 Fold08 <tibble [20 x 5]> <tibble [0 x 3]>
##  9 <split [375/40]> Repeat1 Fold09 <tibble [20 x 5]> <tibble [0 x 3]>
## 10 <split [375/40]> Repeat1 Fold10 <tibble [20 x 5]> <tibble [0 x 3]>
## # ... with 40 more rows
```

Creating a visualization of the output of `tune_grid()`:

```
autoplot(tune_res)
```



Seeing the raw metrics:

```
collect_metrics(tune_res)
```

```
## # A tibble: 20 x 7
##     penalty .metric .estimator  mean     n std_err .config
##       <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1 1.16e-10 rmse    standard   1.13     50 0.0248  Preprocessor1_Model01
##  2 1.16e-10 rsq     standard   0.971    50 0.00135 Preprocessor1_Model01
##  3 6.10e- 9 rmse    standard   1.13     50 0.0248  Preprocessor1_Model02
##  4 6.10e- 9 rsq     standard   0.971    50 0.00135 Preprocessor1_Model02
##  5 6.70e- 8 rmse    standard   1.13     50 0.0248  Preprocessor1_Model03
```

```
##  6 6.70e- 8 rsq     standard   0.971    50 0.00135 Preprocessor1_Model03
##  7 4.06e- 7 rmse    standard   1.13     50 0.0248  Preprocessor1_Model04
##  8 4.06e- 7 rsq     standard   0.971    50 0.00135 Preprocessor1_Model04
##  9 2.50e- 6 rmse    standard   1.13     50 0.0248  Preprocessor1_Model05
## 10 2.50e- 6 rsq     standard   0.971    50 0.00135 Preprocessor1_Model05
## 11 8.49e- 5 rmse    standard   1.13     50 0.0248  Preprocessor1_Model06
## 12 8.49e- 5 rsq     standard   0.971    50 0.00135 Preprocessor1_Model06
## 13 3.08e- 4 rmse    standard   1.13     50 0.0248  Preprocessor1_Model07
## 14 3.08e- 4 rsq     standard   0.971    50 0.00135 Preprocessor1_Model07
## 15 6.54e- 3 rmse    standard   1.13     50 0.0248  Preprocessor1_Model08
## 16 6.54e- 3 rsq     standard   0.971    50 0.00135 Preprocessor1_Model08
## 17 6.00e- 2 rmse    standard   1.13     50 0.0248  Preprocessor1_Model09
## 18 6.00e- 2 rsq     standard   0.971    50 0.00135 Preprocessor1_Model09
## 19 8.12e- 1 rmse    standard   1.16     50 0.0258  Preprocessor1_Model10
## 20 8.12e- 1 rsq     standard   0.970    50 0.00140 Preprocessor1_Model10
```

Selecting the best values:

```
best_penalty <- select_best(tune_res, metric = "rsq")
best_penalty
```

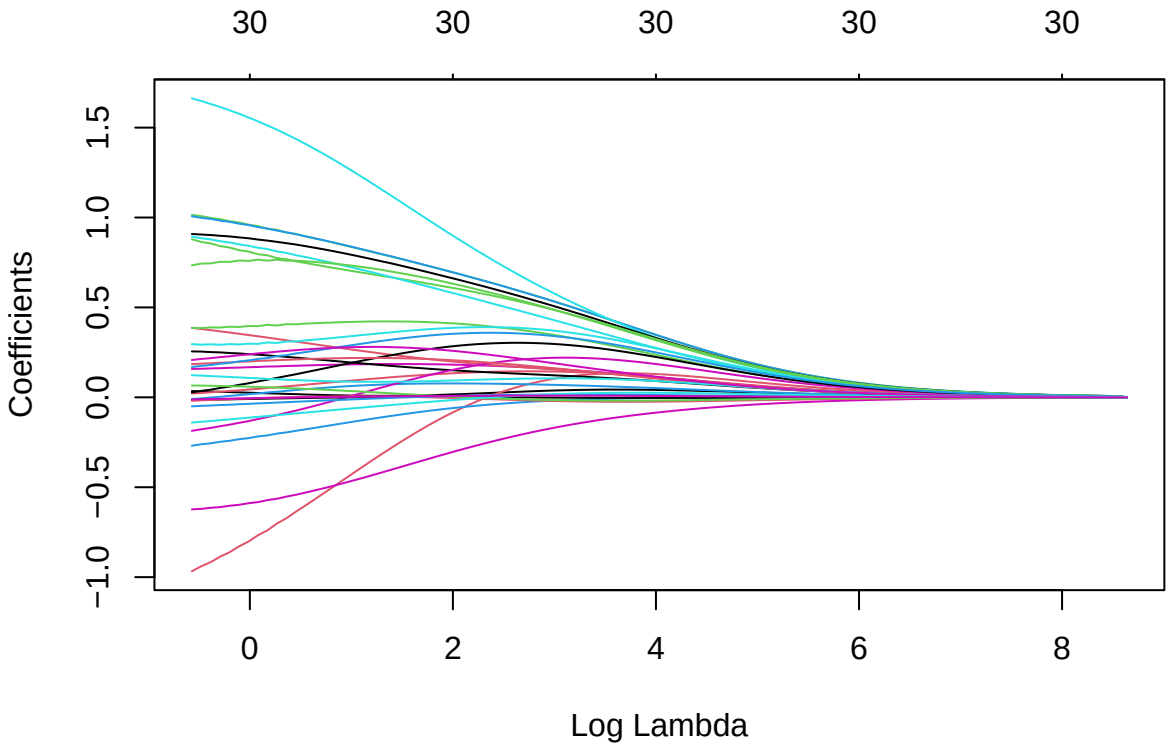
```
## # A tibble: 1 x 2
##    penalty .config
##      <dbl> <chr>
## 1 1.16e-10 Preprocessor1_Model01
```

Fitting the best model using the whole training data set:

```
ridge_final <- finalize_workflow(ridge_workflow, best_penalty)
ridge_final_fit <- fit(ridge_final, data = basketball_train)
```

Visualizing how the magnitude of the coefficients are being regularized towards zero as the penalty increases:

```
ridge_final_fit %>%
  extract_fit_engine() %>%
  plot(xvar = "lambda")
```

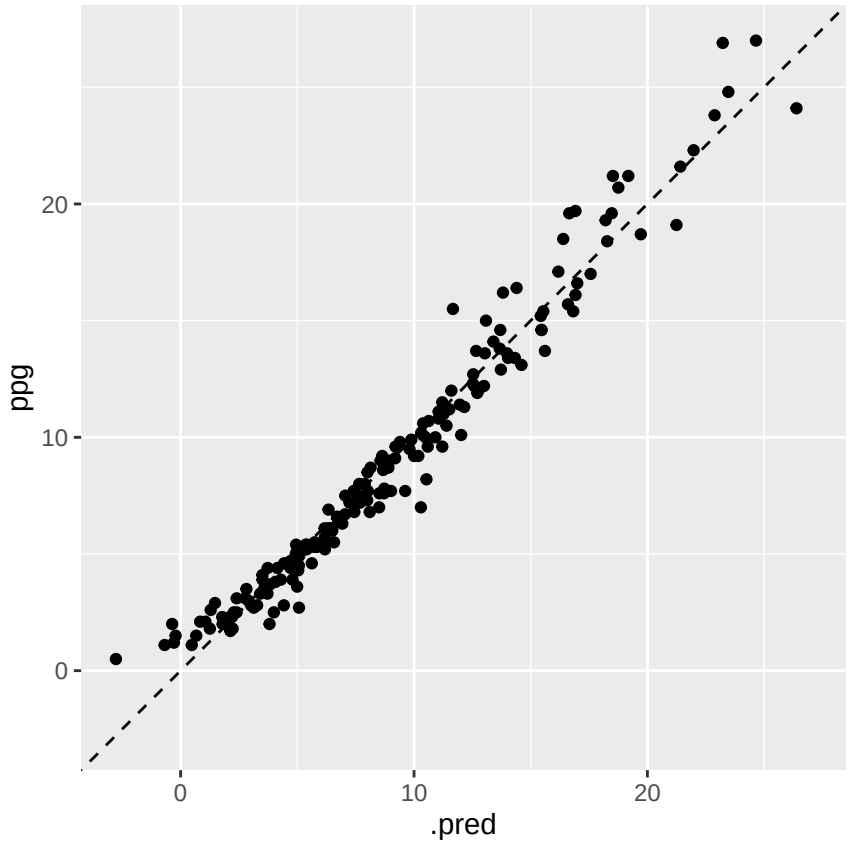


Predict on testing set, and showing a visualization:

```
basketball_train_res1 <- predict(ridge_final_fit, new_data = basketball_test %>% dplyr::select(-ppg))
basketball_train_res1 <- bind_cols(basketball_train_res1, basketball_test %>% dplyr::select(ppg))


ridge_graph <- basketball_train_res1 %>%
  ggplot(aes(x=.pred, y =ppg)) + geom_point(alpha=1) + geom_abline(lty = 2) + coord_obs_pred()

ridge_accuracy <- augment(ridge_final_fit, new_data = basketball_test) %>%
  rsq(truth = ppg, estimate = .pred)


ridge_accuracy
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rsq     standard       0.964
```

```
ridge_graph
```



Here we can see what the model predicted vs the actual values of `ppg`:

```
head(basketball_train_res1)
```

```
## # A tibble: 6 x 2
##    .pred   ppg
##    <dbl> <dbl>
## 1  8.71    7.6
## 2 19.7    18.7
## 3 12.7    13.7
## 4 11.3    11
## 5  6.57    5.5
## 6 -0.216   1.5
```

Decision Tree Model

A decision tree uses a tree model of decisions and possible outcomes. Visualizing the model is easy to follow. For this model, I am fitting regression trees.

Creating a general decision tree specification using `rpart` as the engine

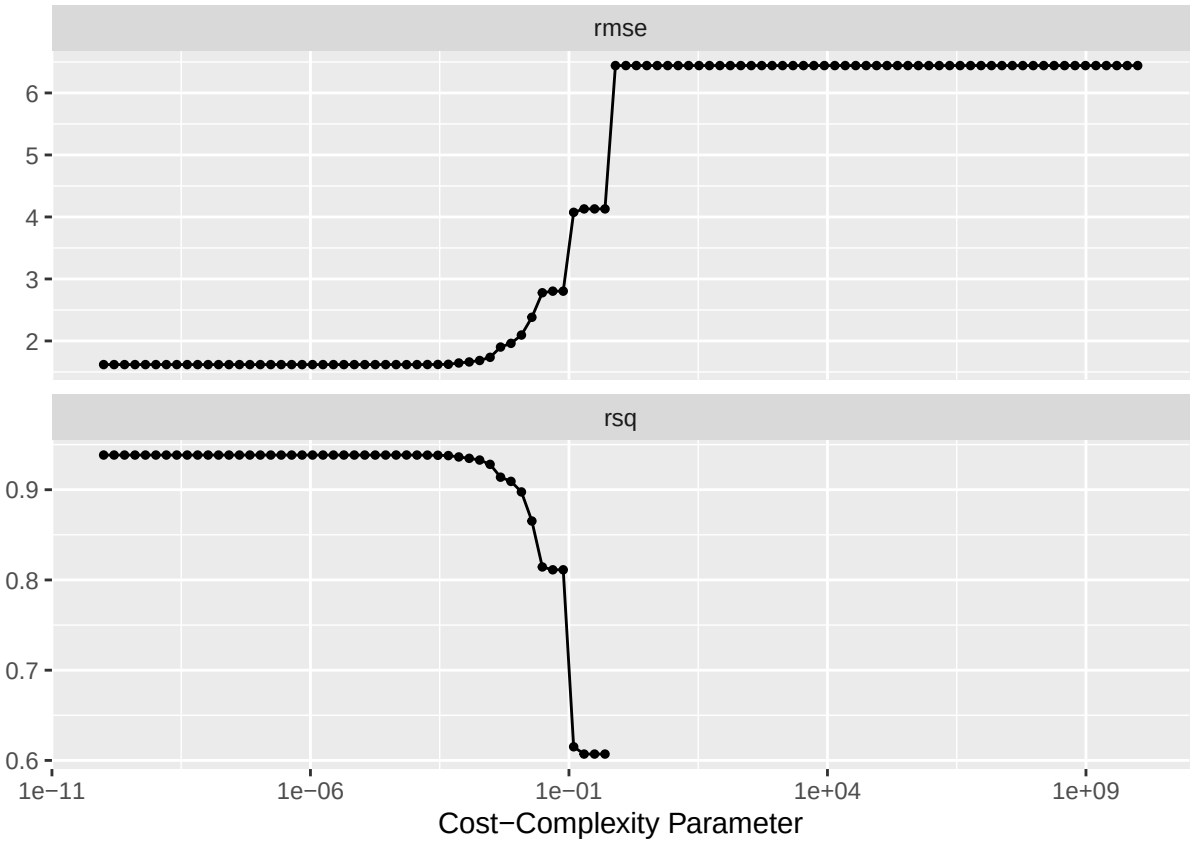
```
tree_spec <- decision_tree() %>%
  set_engine("rpart")
```

Regression decision tree engine:

```
reg_tree_spec <- tree_spec %>%
  set_mode("regression")
```

Tuning `cost_complexity` to find the best performing decision tree

```
reg_tree_wf <- workflow() %>%
  add_model(reg_tree_spec %>% set_args(cost_complexity = tune())) %>%
  add_recipe(basketball_recipe)

set.seed(124)
param_grid <- grid_regular(cost_complexity(range = c(-10,10)), levels = 100)

tune_res_tree <- tune_grid(
  reg_tree_wf,
  resamples = basketball_folds,
  grid = param_grid
)

autoplot(tune_res_tree)
```



Collecting the metrics:

```
tree_roc_auc <- collect_metrics(tune_res_tree) %>%
  arrange(-mean)

head(tree_roc_auc)
```

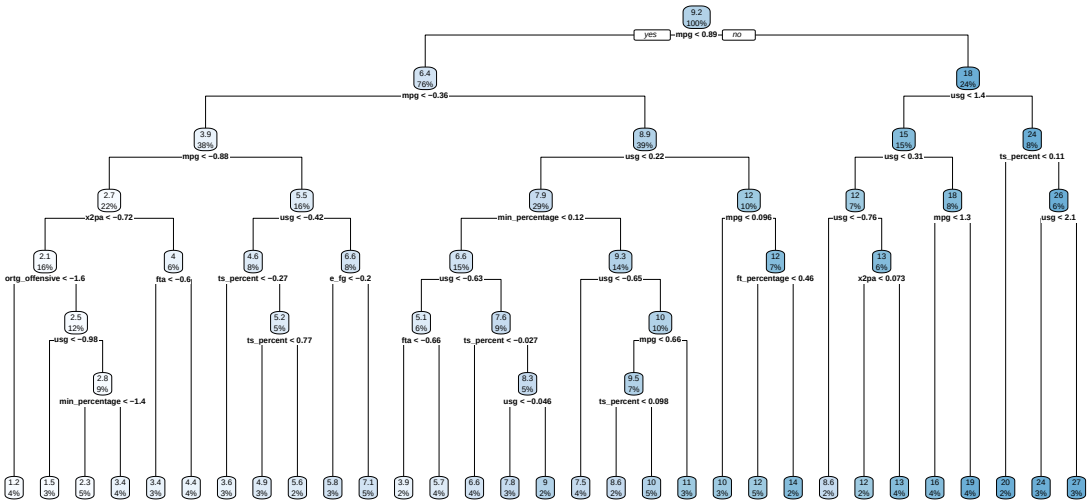
```
## # A tibble: 6 x 7
##   cost_complexity .metric .estimator  mean     n std_err .config
##             <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1           0.792 rmse    standard    6.44    50   0.108 Preprocessor1_Model050
## 2           1.26  rmse    standard    6.44    50   0.108 Preprocessor1_Model051
## 3           2.01  rmse    standard    6.44    50   0.108 Preprocessor1_Model052
## 4           3.20  rmse    standard    6.44    50   0.108 Preprocessor1_Model053
## 5           5.09  rmse    standard    6.44    50   0.108 Preprocessor1_Model054
## 6           8.11  rmse    standard    6.44    50   0.108 Preprocessor1_Model055
```

Selecting the best-performing model according to `rmse` and fitting the final model on the whole training data set:

```
best_complexity <- select_best(tune_res_tree, metric = "rmse")

reg_tree_final <- finalize_workflow(reg_tree_wf, best_complexity)

reg_tree_final_fit <- fit(reg_tree_final, data = basketball_train)
```

Visualizing the model:

```
reg_tree_final_fit %>%
  extract_fit_engine() %>%
  rpart.plot()
```

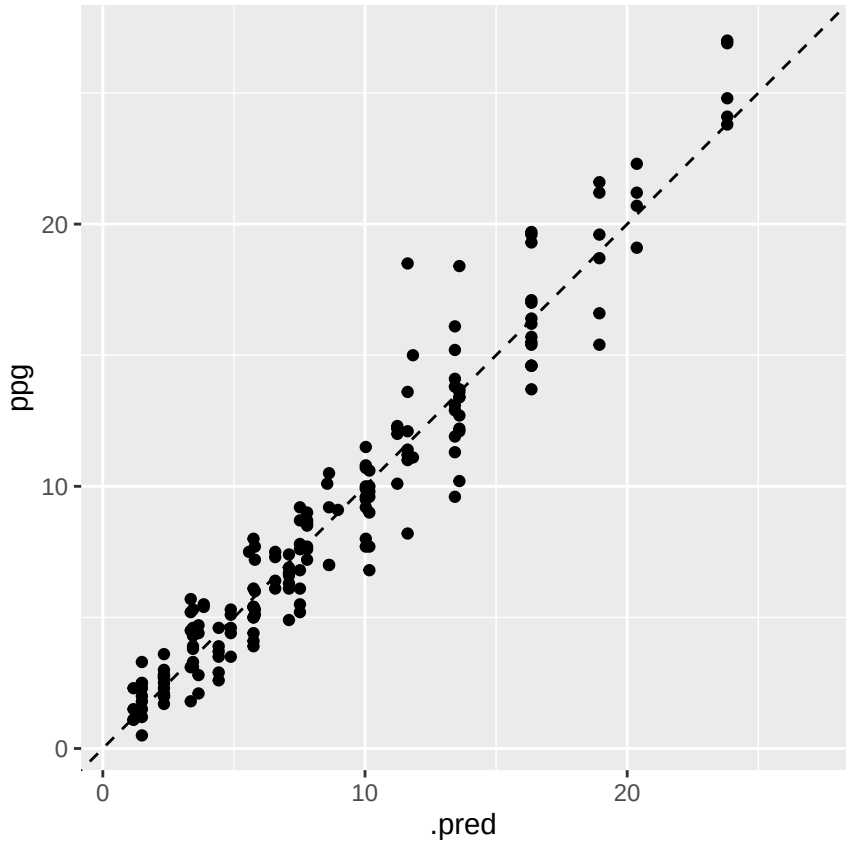


Predict on testing set, showing a visualization, and checking the performance on the testing data set:

```
tree_prediction <- predict(reg_tree_final_fit, new_data = basketball_test %>% dplyr:: select(-ppg))
tree_prediction <- bind_cols(tree_prediction, basketball_test %>% dplyr::select(ppg))


tree_graph <- tree_prediction %>%
  ggplot(aes(x = .pred, y = ppg)) + geom_point(alpha = 1) + geom_abline(lty = 2) + coord_obs_pred()

tree_accuracy <- augment(reg_tree_final_fit, new_data = basketball_test) %>%
  rsq(truth = ppg, estimate = .pred)

tree_accuracy
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rsq     standard       0.935
```

```
tree_graph
```



Here we can see what the model predicted vs the actual values of `ppg`:

```
head(tree_prediction)
```

```
## # A tibble: 6 x 2
##   .pred   ppg
##   <dbl> <dbl>
## 1  7.52   7.6
## 2 18.9   18.7
## 3 13.6   13.7
## 4 11.6   11
## 5  7.52   5.5
## 6  1.17   1.5
```

Lasso Regression

I am using the glmnet package to perform lasso linear regression. Lasso regression is another type of regularization modeling. Lasso minimizes the sum of squared residuals. Unline Ridge regression, Lasso has variable selection.

I am using the `glmnet` package to perform lasso linear regression. For lasso regression, I have set `mixture = 1`

```
lasso_spec <- linear_reg(penalty = tune(), mixture = 1) %>%
  set_mode("regression") %>%
  set_engine("glmnet")
```

Setting up the workflow:

```
lasso_workflow <- workflow() %>%
  add_recipe(basketball_recipe) %>%
  add_model(lasso_spec)
```

Setting up the penalty grid:

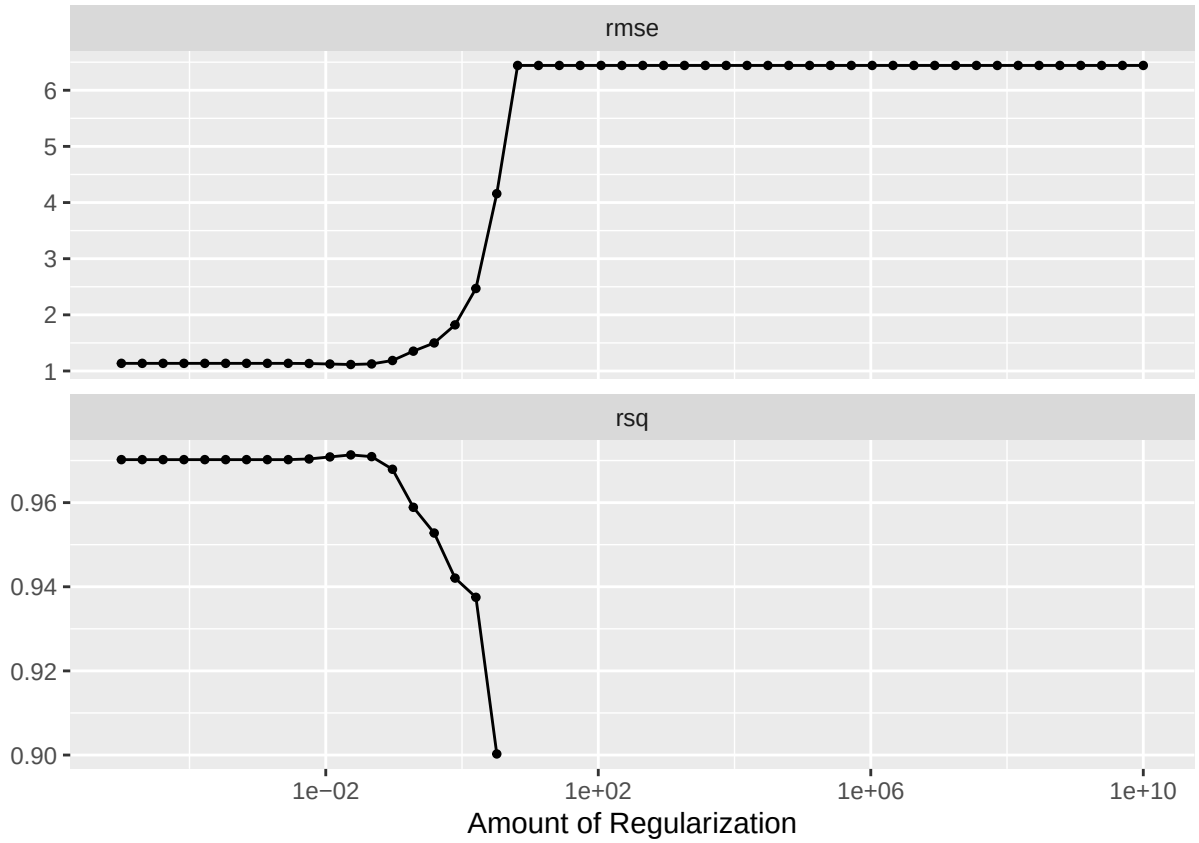
```
lasso_penalty_grid <- grid_regular(penalty(range = c(-5,10)), levels = 50)
```

Tuning and visualizing:

```
set.seed(300)
tune_res_lasso <- tune_grid(
  lasso_workflow,
  resamples = basketball_folds,
  grid = lasso_penalty_grid
)
head(tune_res_lasso)
```

```
## # A tibble: 6 x 5
##   splits           id      id2    .metrics           .notes
##   <list>           <chr>   <chr>  <list>             <list>
## 1 <split [371/44]> Repeat1 Fold01 <tibble [100 x 5]> <tibble [1 x 3]>
## 2 <split [371/44]> Repeat1 Fold02 <tibble [100 x 5]> <tibble [1 x 3]>
## 3 <split [372/43]> Repeat1 Fold03 <tibble [100 x 5]> <tibble [1 x 3]>
## 4 <split [374/41]> Repeat1 Fold04 <tibble [100 x 5]> <tibble [1 x 3]>
## 5 <split [374/41]> Repeat1 Fold05 <tibble [100 x 5]> <tibble [1 x 3]>
## 6 <split [374/41]> Repeat1 Fold06 <tibble [100 x 5]> <tibble [1 x 3]>
```

```
autoplot(tune_res_lasso)
```



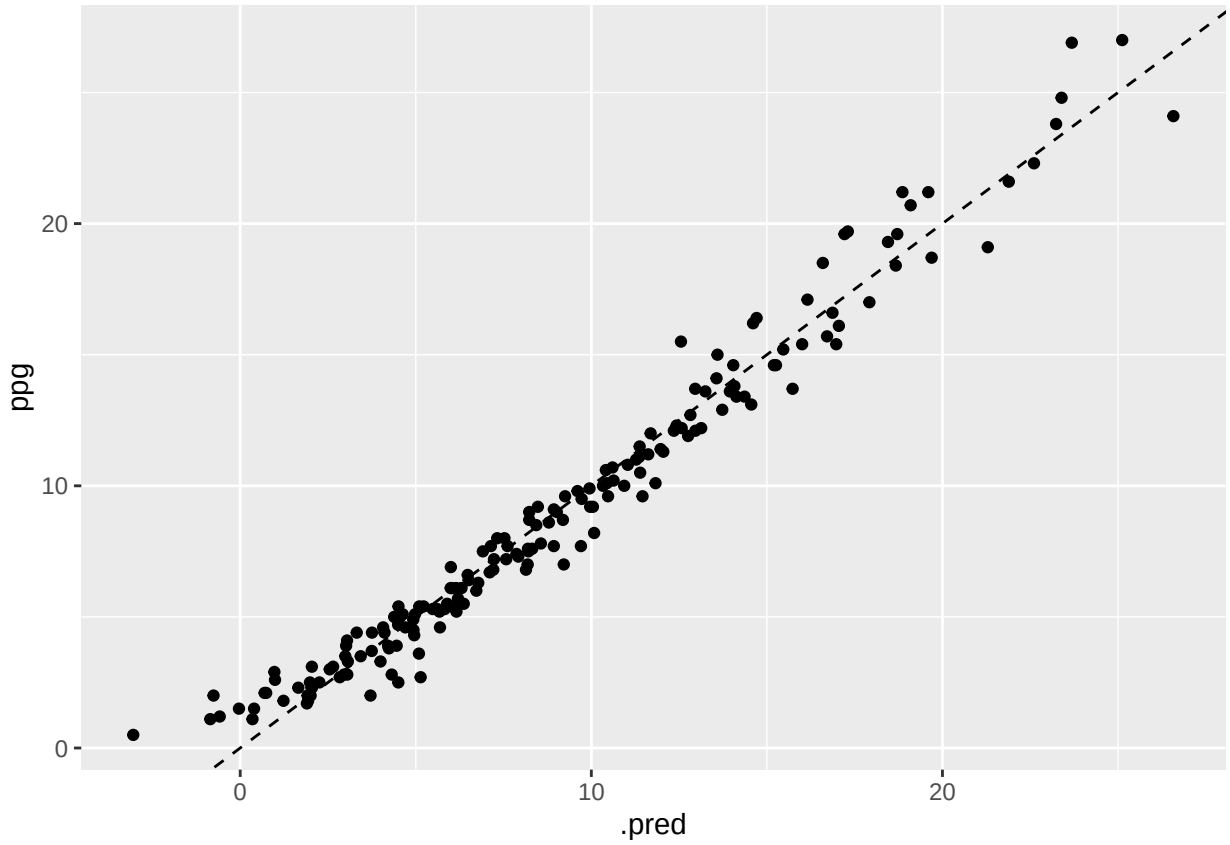
Collecting the metrics and selecting the best value of penalty and refitting using the test set:

```
lasso_metrics <- collect_metrics(tune_res_lasso)
head(lasso_metrics)
```

```
## # A tibble: 6 x 7
##     penalty .metric .estimator  mean     n std_err .config
##       <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 0.00001   rmse    standard   1.14     50 0.0258  Preprocessor1_Model01
## 2 0.00001   rsq     standard   0.970    50 0.00140 Preprocessor1_Model01
## 3 0.0000202 rmse    standard   1.14     50 0.0258  Preprocessor1_Model02
## 4 0.0000202 rsq     standard   0.970    50 0.00140 Preprocessor1_Model02
## 5 0.0000409 rmse    standard   1.14     50 0.0258  Preprocessor1_Model03
## 6 0.0000409 rsq     standard   0.970    50 0.00140 Preprocessor1_Model03
```

```
best_penalty_lasso <- select_best(tune_res_lasso, metric="rsq")

lasso_final <- finalize_workflow(lasso_workflow, best_penalty_lasso)

lasso_final_fit <- fit(lasso_final, data = basketball_train)

lasso_prediction <- predict(lasso_final_fit, new_data = basketball_test %>%
                            dplyr::select(-ppg))
lasso_prediction <- bind_cols(lasso_prediction, basketball_test %>% dplyr::select(ppg))
```

```
lasso_graph <- lasso_prediction %>%
  ggplot(aes(x=.pred, y=ppg)) + geom_point(alpha=1) + geom_abline(lty=2)
lasso_graph
```




```
lasso_accuracy <- augment(lasso_final_fit, new_data = basketball_test) %>%
  rsq(truth=ppg, estimate = .pred)
```

Here we can see what the model predicted vs the actual values of ppg:

```
head(lasso_prediction)
```

```
## # A tibble: 6 x 2
##     .pred   ppg
##     <dbl> <dbl>
## 1  8.32     7.6
## 2 19.7     18.7
## 3 13.0     13.7
## 4 11.3     11
## 5  6.37     5.5
## 6 -0.0358   1.5
```

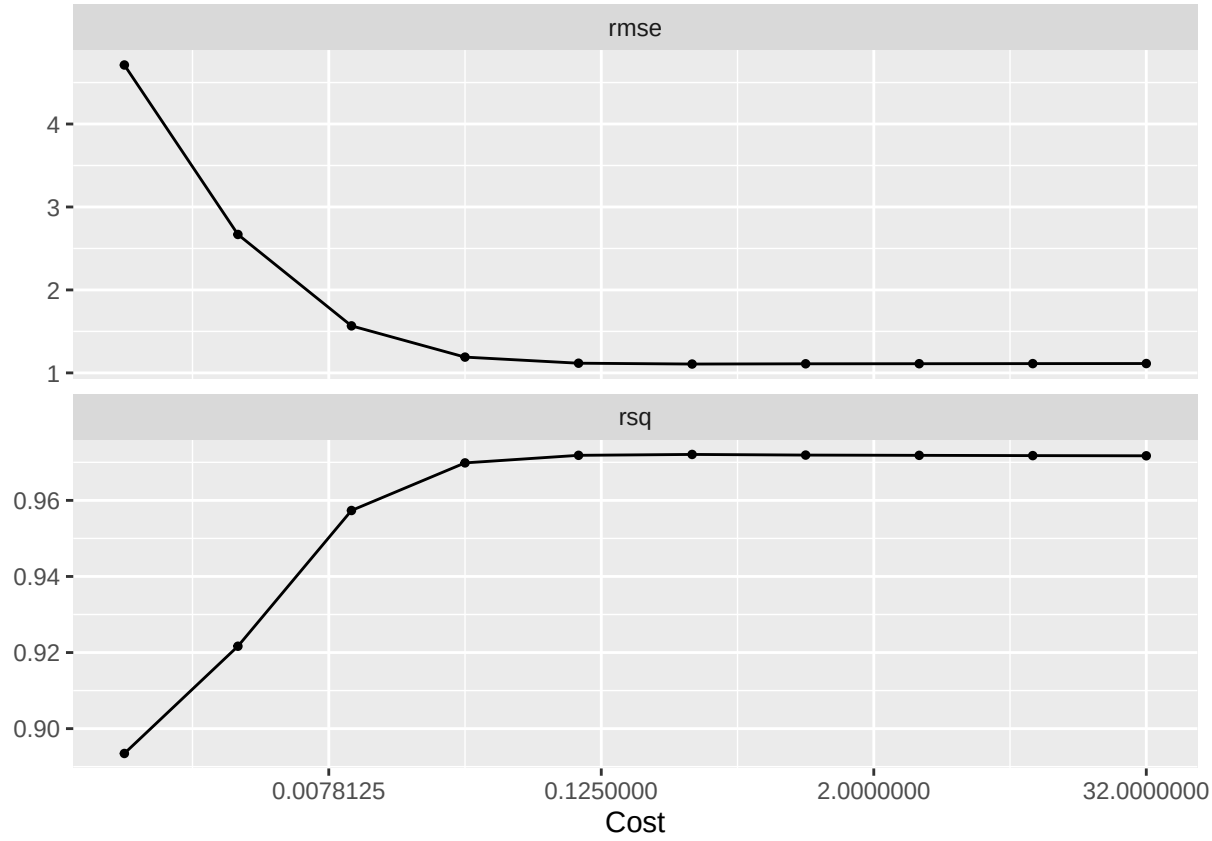
Support Vector Machine

Support Vector Regression is a supervised learning algorithm.

```
svm_linear_spec <- svm_poly(degree = 1) %>%
  set_mode("regression") %>%
  set_engine("kernlab", scaled = FALSE)
```

Setting up the workflow, tuning and visualizing:

```
svm_linear_wf <- workflow() %>%
  add_model(svm_linear_spec %>% set_args(cost = tune())) %>%
  add_recipe(basketball_recipe)

set.seed(4529)

svm_grid <- grid_regular(cost(), levels = 10)

svm_tune_res <- tune_grid(
  svm_linear_wf,
  resamples = basketball_folds,
  grid = svm_grid
)
autoplot(svm_tune_res)
```



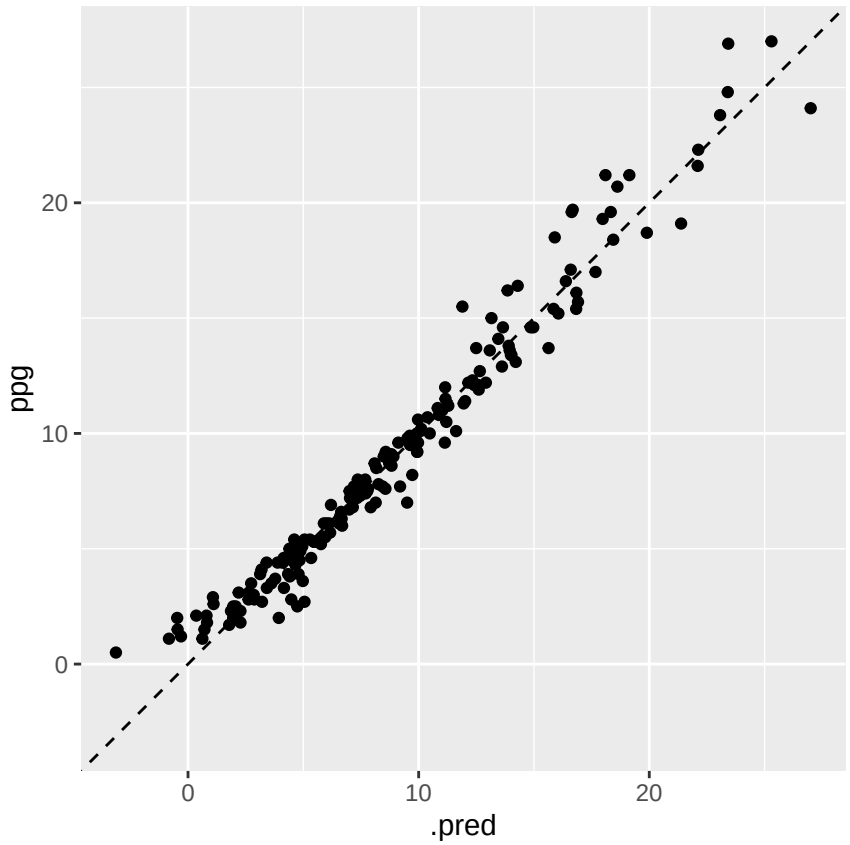
Collecting the metrics:

```
svm_RMSE <- collect_metrics(svm_tune_res) %>%
  dplyr::select(.metric, mean, std_err) %>%
  head()
svm_RMSE
```

```
## # A tibble: 6 x 3
##   .metric  mean std_err
##   <chr>   <dbl>   <dbl>
## 1 rmse    4.71  0.0937
## 2 rsq     0.893 0.00387
## 3 rmse    2.67  0.0634
## 4 rsq     0.922 0.00318
## 5 rmse    1.57  0.0398
## 6 rsq     0.957 0.00204
```

Selecting the best value of `rsq` and fitting the model on the entire testing set:

```
best_cost <- select_best(svm_tune_res, metric = "rsq")

svm_linear_final <- finalize_workflow(svm_linear_wf, best_cost)

svm_linear_fit <- fit(svm_linear_final, data = basketball_train)

svm_prediction <- predict(svm_linear_fit, new_data = basketball_test %>% dplyr::select(-ppg))

svm_prediction <- bind_cols(svm_prediction, basketball_test %>% dplyr::select(ppg))

svm_graph <- svm_prediction %>%
  ggplot(aes(x=.pred, y=ppg)) + geom_point(alpha=1) + geom_abline(lty = 2)  + coord_obs_pred()


svm_accuracy <- augment(svm_linear_fit, new_data = basketball_test) %>%
  rsq(truth = ppg, estimate = .pred)


svm_graph
```




```
svm_accuracy
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rsq     standard       0.965
```

Here we can see what the model predicted vs the actual values of `ppg`:

```
head(svm_prediction)
```

```
## # A tibble: 6 x 2
##    .pred   ppg
##    <dbl> <dbl>
## 1  8.56    7.6
## 2 19.9    18.7
## 3 12.5    13.7
## 4 11.0    11
## 5  5.95    5.5
## 6 -0.451   1.5
```

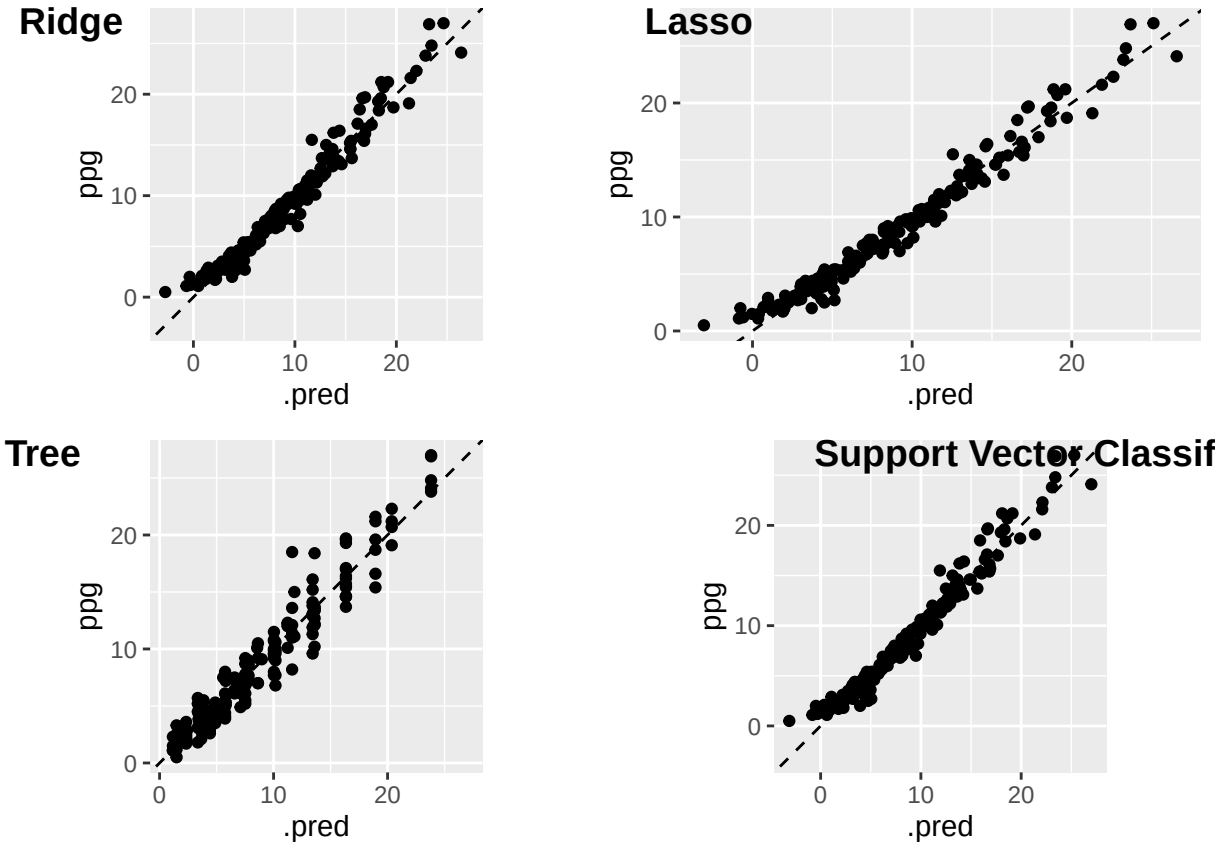
It looks like the model did a good job for the most part.

Conclusion

Graph

Creating a comparison of all models and how closely their predictions fit. The dotted line represents the actual value of ppg and each dot represents what the model predicted. Therefore, the closer the point is to the line, the more accurate the model predicted ppg:

```
comparison_figure <- ggarrange(ridge_graph, lasso_graph, tree_graph, svm_graph, labels = c("Ridge", "Las
comparison_figure
```



Comparing accuracies:

I am comparing the accuracy of the models by looking at their rsq value. R-Squared is a statistical measure of fit that indicates how much variation of a dependent variable is explained by the independent variable(s) .

```
rsq_comparisons <- bind_rows(ridge_accuracy, lasso_accuracy, tree_accuracy, svm_accuracy) %>%
  tibble() %>% mutate(model = c("Ridge", "Lasso", "Tree", "SVM")) %>%
  dplyr::select(model, .estimate) %>%
  arrange(.estimate)
rsq_comparisons
```

```
## # A tibble: 4 x 2
```

```
##   model .estimate
##   <chr>     <dbl>
## 1 Tree      0.935
## 2 Ridge     0.964
## 3 SVM       0.965
## 4 Lasso     0.968
```

Based on RSQ, the Lasso Regression model is the most accurate, and the tree model is the least accurate. Despite this, I would say that all of my models performed well because they all have accuracies that are over 0.90.

Summary

To predict the outcome of how many points per game a player in the NBA scores, I compared a Ridge Regression model, a Lasso Regression model, a Tree model and a Support Vector Machine model. Based on my exploratory data analysis, I can conclude that the most important factor on whether a player will score more points per game is minutes per game. Based on the comparisons I made, I can conclude that the Lasso Regression model is the best at predicting how many points on average a player will score per game. The rest of the models also performed very well, with all of them having high accuracies.