Predicting NBA player impact and cost-efficiency

Introduction

The goal of this project is to predict future performance of NBA players and determine whether an NBA player will be "worth" his current salary. We want to see if we can use data to separate contracts based on value and predict whether a contract would be worthwhile based on previous performance.

Considering the drastic changes in both salary rules and gameplay in the NBA beginning in 2015 and fully taking action in the late 2010s, we'll only consider the data from 2017 onward. For more information regarding these changes, check out these articles on the three-point revolution and articles related to the recent salary cap increase like this one.

Criteria: Measuring Impact

First, we need to determine how to measure a player's impact towards a team winning. We'll only consider from advanced metrics over traditional box score metrics because it is now widely accepted in the NBA community that traditional box score stats do not reflect a player's actual impact towards winning due to a variety of factors.

For a quick anecdotal example, Michael Adams of the Denver Nuggets averaged 26.5/3.9/10.5 during the 90-91 season. Magic Johnson averaged 19.4/7/12.5 during the same season, which seems relatively even with Adams' season considering the higher rebounds and assists average. However, Magic Johnson was 2nd in MVP voting while Michael Adams was not even an all-star that season. Even if MVP voting in the NBA is quite a subjective procedure, Adams should have been in the MVP race if he had an impact towards winning anywhere close to Magic's. As such, we'll stick to advanced statistics already developed to measure a player's impact.

BPM is an advanced statistic in basketball that estimates a player's contribution while the player is on the court. In essence, it's a player's value-add to a team adjusted for possessions played.

VORP is a similar metric to BPM, but it takes the total minutes spent on the court. This aspect of VORP makes the metric inherently more advantaged towards already established players who play significant minutes.

Considering the ultimate goal of this exercise is determine the fair value of all players including those that might not have shined due to numerous circumstances, BPM would be the fairer metric to use.

Importing the data

Basektball reference has the cleanest data on player stats by far, but we need to build a scraper to get its data. We can use this Towards Data Science article by Oscar Sanchez as reference, but we'll be replacing BeautifulSoup with rvest since it's much easier to use rvest in an R environment.

Import player stats

```
# get the html of the bball ref pages with every player's advanced metrics for the seasons we're considering
# since we'll be using improvements from the previous seasons, get data beginning in 2015
years <- 2015:2021
for (year in years) {
  var_name <- paste0('html_', year)
  html <- read_html(paste0("https://www.basketball-reference.com/leagues/NBA_", year, "_advanced.html"))
  assign(var_name, html)
}
```

```
get_advanced_stats_from_html <- function(html) {
  columns <- html_text(html %>% html_nodes("th"))[2:29]
  columns <- tolower(columns)
  data <- data.frame(matrix(unlist(html_text(html %>% html_nodes("td"))), ncol = 28, byrow = TRUE))
  colnames(data) <- columns
  data <- data[, !duplicated(colnames(data))]
  return(data)
}
```

```
advanced_stats_2015 <- get_advanced_stats_from_html(html_2015)
advanced_stats_2016 <- get_advanced_stats_from_html(html_2016)
advanced_stats_2017 <- get_advanced_stats_from_html(html_2017)
advanced_stats_2018 <- get_advanced_stats_from_html(html_2018)
advanced_stats_2019 <- get_advanced_stats_from_html(html_2019)
advanced_stats_2020 <- get_advanced_stats_from_html(html_2020)
advanced_stats_2021 <- get_advanced_stats_from_html(html_2021)
```

Import salary data

We'll be using the data from HoopsHype, which also needs to be scraped. However, there's extra levels of cleansing of the data required here due to the HTML format of the website as well as the way the text is stored. The functions below will handle that process.

```
get_nba_salary_data_2021_onwards <- function(url) {
  html <- read_html(url)
  raw_data <- html_text(html %>% html_nodes("td"))
  raw_data <- raw_data[9:length(raw_data)]
  raw_data <- data.frame(matrix(unlist(raw_data), ncol = 8, byrow = TRUE))
  columns <- c("rank", "name", "salary_2021", "salary_2022", "salary_2023", "salary_2024", "salary_2025", "salary
_2026")
  colnames(raw_data) <- columns
  raw_data[,1] <- str_remove_all(raw_data[,1], '\t|\n|\\$|,')
  raw_data[,2] <- str_remove_all(raw_data[,2], '\t|\n|\\$|,')
  raw_data[,3] <- str_remove_all(raw_data[,3], '\t|\n|\\$|,')
  raw_data[,4] <- str_remove_all(raw_data[,4], '\t|\n|\\$|,')
  raw_data[,5] <- str_remove_all(raw_data[,5], '\t|\n|\\$|,')
  raw_data[,6] <- str_remove_all(raw_data[,6], '\t|\n|\\$|,')
  raw_data[,7] <- str_remove_all(raw_data[,7], '\t|\n|\\$|,')
  raw_data[,8] <- str_remove_all(raw_data[,8], '\t|\n|\\$|,')
  raw_data[,3] <- as.numeric(raw_data[,3])
  raw_data[,4] <- as.numeric(raw_data[,4])
  raw_data[,5] <- as.numeric(raw_data[,5])
  raw_data[,6] <- as.numeric(raw_data[,6])
  raw_data[,7] <- as.numeric(raw_data[,7])
  raw_data[,8] <- as.numeric(raw_data[,8])
  clean_data <- raw_data[,2:8]
  return(clean_data)
}
```

```
get_nba_salary_data_pre_2021 <- function(url) {
  html <- read_html(url)
  raw_data <- html_text(html %>% html_nodes("td"))
  raw_data <- raw_data[5:length(raw_data)]
  raw_data <- data.frame(matrix(unlist(raw_data), ncol = 4, byrow = TRUE))
  columns <- c("rank", "name", "salary_raw", "salary_inflation_adjusted")
  colnames(raw_data) <- columns
  raw_data[,1] <- str_remove_all(raw_data[,1], '\t|\n|\\$|,')
  raw_data[,2] <- str_remove_all(raw_data[,2], '\t|\n|\\$|,')
  raw_data[,3] <- str_remove_all(raw_data[,3], '\t|\n|\\$|,')
  raw_data[,4] <- str_remove_all(raw_data[,4], '\t|\n|\\$|,')
  raw_data[,3] <- as.numeric(raw_data[,3])
  raw_data[,4] <- as.numeric(raw_data[,4])
  clean_data <- raw_data[,2:4]
  return(clean_data)
}
```

```
salary_data_2021_onwards <- get_nba_salary_data_2021_onwards("https://hoopshype.com/salaries/players/")
salary_data_2020 <- get_nba_salary_data_pre_2021("https://hoopshype.com/salaries/players/2019-2020/")
salary_data_2019 <- get_nba_salary_data_pre_2021("https://hoopshype.com/salaries/players/2018-2019/")
salary_data_2018 <- get_nba_salary_data_pre_2021("https://hoopshype.com/salaries/players/2017-2018/")
salary_data_2017 <- get_nba_salary_data_pre_2021("https://hoopshype.com/salaries/players/2016-2017/")
salary_data_2016 <- get_nba_salary_data_pre_2021("https://hoopshype.com/salaries/players/2015-2016/")
salary_data_2015 <- get_nba_salary_data_pre_2021("https://hoopshype.com/salaries/players/2014-2015/")
salary_data_2021 <- salary_data_2021_onwards %>% select(name, salary_2021)
```

```
#source: Wikipedia
salary_cap_2015 <- 63065000
salary_cap_2016 <- 70000000
salary_cap_2017 <- 94143000
salary_cap_2018 <- 99093000
salary_cap_2019 <- 101869000
salary_cap_2020 <- 109140000
salary_cap_2021 <- 117000000
```

Calculating salary

Because the salary cap of the NBA increases every year, raw contract values aren't equivalent to how much of a team's cap space players are taking up. As such, we need to adjust for the percentage of cap spaces a player took up.

```
salary_data_2015$salary_pct <- salary_data_2015$salary_raw / salary_cap_2015
salary_data_2016$salary_pct <- salary_data_2016$salary_raw / salary_cap_2016
salary_data_2017$salary_pct <- salary_data_2017$salary_raw / salary_cap_2017
salary_data_2018$salary_pct <- salary_data_2018$salary_raw / salary_cap_2018
salary_data_2019$salary_pct <- salary_data_2019$salary_raw / salary_cap_2019
salary_data_2020$salary_pct <- salary_data_2020$salary_raw / salary_cap_2020
salary_data_2021$salary_pct <- salary_data_2021$salary_2021 / salary_cap_2021
```

Additional data work

Using the rationale explained above, we need to first modify the data to only include the relevant metrics in BPM and the percentage of the salary cap each player takes up. After that, we can

```
append_salary_to_performance <- function(year) {
  query <-
  "
  SELECT
    advanced_stats_{year}.player,
    tm,
    obpm,
    dbpm,
    mp,
    bpm,
    salary_pct
  FROM advanced_stats_{year}
  LEFT JOIN salary_data_{year}
  ON advanced_stats_{year}.player = salary_data_{year}.name
  "
  query <- glue(query)
  joined_data <- sqldf(query)
  #remove duplicate entries for players who got traded during the season
  joined_data <- joined_data[!duplicated(joined_data$player) | joined_data$tm == 'TOT',]
  joined_data <- joined_data[!is.na(joined_data$salary_pct),]
  joined_data$year <- as.numeric(year)
  joined_data$obpm <- as.numeric(joined_data$obpm)
  joined_data$dbpm <- as.numeric(joined_data$dbpm)
  joined_data$bpm <- as.numeric(joined_data$bpm)
  joined_data$mp <- as.numeric(joined_data$mp)
  return(joined_data)
}
```

Basic data exploration

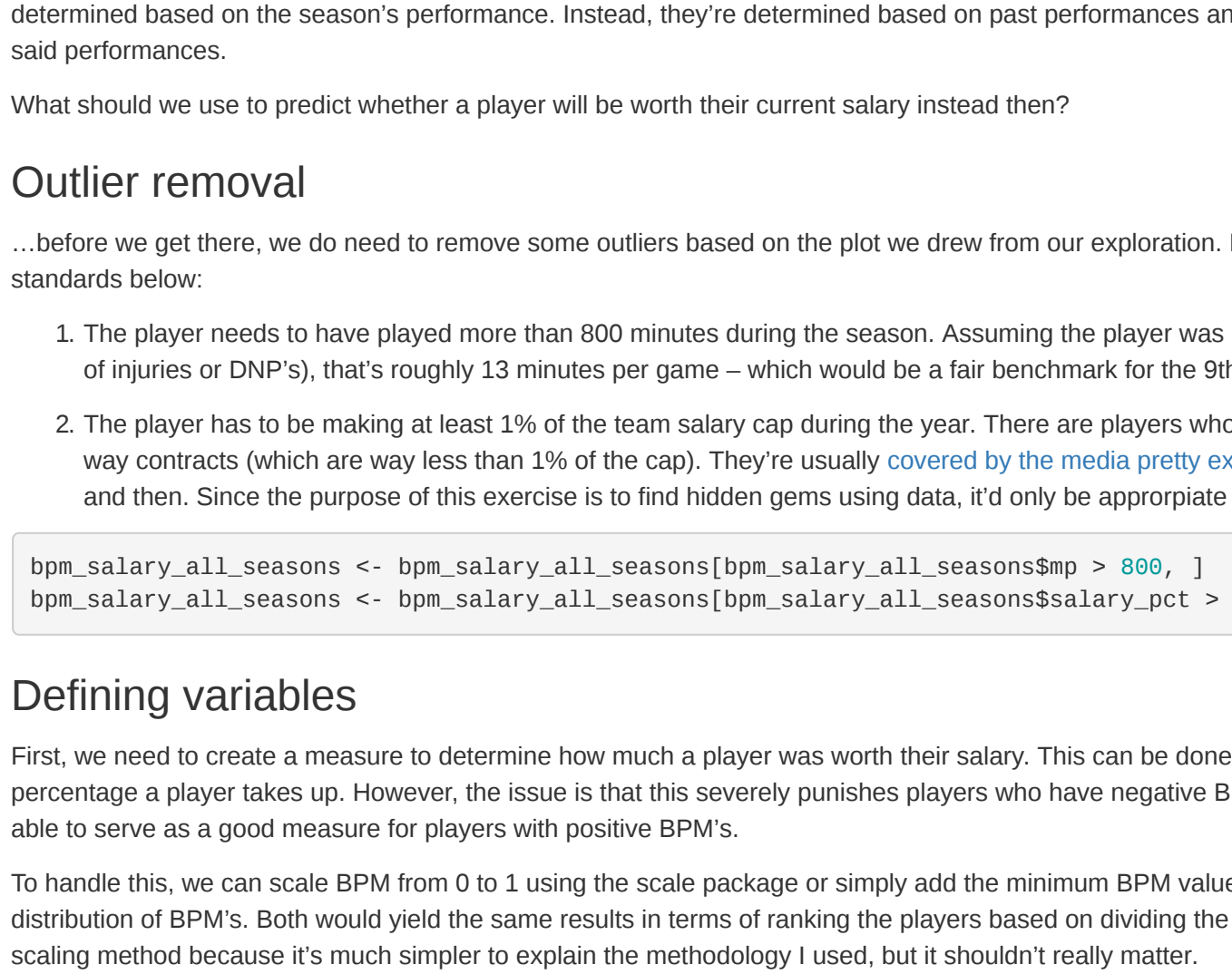
Let's now explore the relationship between BPM and the percentage of salary cap each year.

```
#combine data from all seasons we're using
bpm_salary_2015 <- append_salary_to_performance("2015")
bpm_salary_2016 <- append_salary_to_performance("2016")
bpm_salary_2017 <- append_salary_to_performance("2017")
bpm_salary_2018 <- append_salary_to_performance("2018")
bpm_salary_2019 <- append_salary_to_performance("2019")
bpm_salary_2020 <- append_salary_to_performance("2020")
bpm_salary_2021 <- append_salary_to_performance("2021")
bpm_salary_all_seasons <- rbind(bpm_salary_2017, bpm_salary_2018, bpm_salary_2019, bpm_salary_2020, bpm_salary_20
21)
```

```
#simple linear regression
bpm_to_salary <- lm(bpm~salary_pct, data = bpm_salary_all_seasons)
summary(bpm_to_salary)
```

```
##
## Call:
## lm(formula = bpm ~ salary_pct, data = bpm_salary_all_seasons)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -52.315  -1.598   0.302   2.021  54.795
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -3.1210     0.1204  -25.91   <2e-16 ***
## salary_pct   24.0940     1.2114   19.89   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.485 on 2290 degrees of freedom
## Multiple R-squared:  0.1473, Adjusted R-squared:  0.1469
## F-statistic: 395.6 on 1 and 2290 DF,  p-value: < 2.2e-16
```

```
#plot with linear regression line
ggplot(bpm_salary_all_seasons, aes(x = salary_pct, y = bpm)) + geom_point() +
  geom_smooth(method='lm', formula= y~x)
```



We can see that there is clearly a significant relationship between the salary percentage a player takes up and the impact to winning each players makes (measured by BPM). That being said, this regression model is effectively useless in practice for one simple reason: NBA salaries are not determined based on the season's performance. Instead, they're determined based on past performances and speculation of the future based on said performances.

What should we use to predict whether a player will be worth their current salary instead then?

Outlier removal

…before we get there, we do need to remove some outliers based on the plot we drew from our exploration. In order to do so, I've set the standards below:

1. The player needs to have played more than 800 minutes during the season. Assuming the player was able to play in 60 games (regardless of injuries or DNP's), that's roughly 13 minutes per game – which would be a fair benchmark for the 9th ~ 10th man for any given team.
2. The player has to be making at least 1% of the team salary cap during the year. There are players who occasionally shine from 10-day or 2-way contracts (which are way less than 1% of the cap). They're usually covered by the media pretty extensively when it happens every now and then. Since the purpose of this exercise is to find hidden gems using data, it'd only be approrpiate to remove them as potential outliers.

```
bpm_salary_all_seasons <- bpm_salary_all_seasons[bpm_salary_all_seasons$mp > 800, ]
bpm_salary_all_seasons <- bpm_salary_all_seasons[bpm_salary_all_seasons$salary_pct > 0.01, ]
```

Defining variables

First, we need to create a measure to determine how much a player was worth their salary. This can be done through dividing BPM by the salary percentage a player takes up. However, the issue is that this severely punishes players who have negative BPM's on lower salaries while being able to serve as a good measure for players with positive BPM's.

To handle this, we can scale BPM from 0 to 1 using the scale package or simply add the minimum BPM value to all BPM values to shift the distribution of BPM's. Both would yield the same results in terms of ranking the players based on dividing the modified BPM by . I prefer to use the scaling method because it's much simpler to explain the methodology I used, but it shouldn't really matter.

```
#add new features
bpm_salary_all_seasons$bpm_scaled <- rescale(bpm_salary_all_seasons$bpm)
bpm_salary_all_seasons$bpm_scaled_to_salary <- bpm_salary_all_seasons$bpm_scaled / bpm_salary_all_seasons$salary_
pct
```

We have the variable that represents a player's relative contribution to his salary. Now, we need to develop the features to predict said variable from. For the purpose of this exercise, I figured two metrics would be simple and appropriate to develop.

The first is the BPM from the season prior. Players generally don't jump or dip in performance significantly barring injuries, which means using BPM from the prior season should be a good predictor for their BPM the next year.

The second is the growth rate of BPM from two seasons prior to the season prior. We want to use this metric to show whether the player is declining or improving. Theoretically, high growth rate from the previous season should imply that the player still has potential for growth.

The third is the salary percentage from the season prior. We can use this as the salary expectation of the player for the next season. Also, in some classification methods, I'm hoping that high BPM paired with low salary percentage can be identified as a player who might be looking for a payday.

```
#need to append the data from previous seasons to calculate the previous three year average
advanced_stats_2015$year <- 2015
advanced_stats_2016$year <- 2016
fix_format <- function(df) {
  df <- df[!duplicated(df$player) | df$tm == 'TOT',]
  df$obpm <- as.numeric(df$obpm)
  df$dbpm <- as.numeric(df$dbpm)
  df$bpm <- as.numeric(df$bpm)
  df$mp <- as.numeric(df$mp)
  df <- df %>% select(year, player, tm, obpm, dbpm, bpm, mp)
  return(df)
}
advanced_stats_2015 <- fix_format(advanced_stats_2015)
advanced_stats_2016 <- fix_format(advanced_stats_2016)
bpm_salary_all_seasons <- bind_rows(bpm_salary_all_seasons, advanced_stats_2015, advanced_stats_2016)
```

```
query_prev_season <-
  "
  SELECT
    player,
    year,
    bpm_scaled,
    LAG(bpm_scaled , 1) OVER (PARTITION BY player ORDER BY year) AS bpm_prev_season,
    1.00 * (LAG(bpm_scaled , 1) OVER (PARTITION BY player ORDER BY year) - LAG(bpm_scaled , 2) OVER (PARTITION BY
player ORDER BY year)) / LAG(bpm_scaled , 2) OVER (PARTITION BY player ORDER BY year) AS bpm_prev_season_growth,
    LAG(salary_pct , 1) OVER (PARTITION BY player ORDER BY year) AS salary_pct_prev_season
  FROM bpm_salary_all_seasons
  "
player_prev_season_data <- sqldf(query_prev_season)
```

```
prediction_data_query <-
  "
  SELECT
    bpm_salary_all_seasons.player,
    bpm_salary_all_seasons.year,
    bpm_salary_all_seasons.bpm_scaled_to_salary,
    bpm_salary_all_seasons.bpm_scaled,
    player_prev_season_data.bpm_prev_season_growth,
    player_prev_season_data.bpm_prev_season,
    player_prev_season_data.salary_pct_prev_season
  FROM bpm_salary_all_seasons
  INNER JOIN player_prev_season_data
  ON bpm_salary_all_seasons.player = player_prev_season_data.player AND bpm_salary_all_seasons.year = player_prev
_season_data.year
  WHERE 1=1
    AND player_prev_season_data.bpm_prev_season_growth IS NOT NULL
    AND player_prev_season_data.bpm_prev_season IS NOT NULL
    AND player_prev_season_data.salary_pct_prev_season IS NOT NULL
  "

prediction_data <- sqldf(prediction_data_query)
prediction_data$bpm_scaled_to_salary_prev_season <- prediction_data$bpm_prev_season / prediction_data$salary_pct_
prev_season
```

Predicting BPM

```
bpm_prediction <- lm(bpm_scaled~bpm_prev_season_growth + bpm_prev_season, data = prediction_data)
summary(bpm_prediction)
```

```
##
## Call:
## lm(formula = bpm_scaled ~ bpm_prev_season_growth + bpm_prev_season,
##     data = prediction_data)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.311420 -0.057637  0.000933  0.054533  0.292865
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)             0.06532    0.01200   5.444 8.28e-08 ***
## bpm_prev_season_growth -0.06105    0.01266  -4.821 1.92e-06 ***
## bpm_prev_season         0.83712    0.02720  30.776  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.08613 on 486 degrees of freedom
## Multiple R-squared:  0.6609, Adjusted R-squared:  0.6595
## F-statistic: 473.6 on 2 and 486 DF,  p-value: < 2.2e-16
```

This model did a pretty good job of predicting BPM given the relatively high R-squared value.

Predicting cost-efficiency: linear regression

```
bpm_to_salary_prediction_include_growth <- lm(bpm_scaled_to_salary ~ bpm_prev_season_growth + bpm_scaled_to_salar
y_prev_season, data = prediction_data)
summary(bpm_to_salary_prediction_include_growth)
```

```
##
## Call:
## lm(formula = bpm_scaled_to_salary ~ bpm_prev_season_growth +
##     bpm_scaled_to_salary_prev_season, data = prediction_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.636  -1.992  -1.169   1.027  18.768
##
## Coefficients:
##                                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)                        2.82372    0.29293   9.640   <2e-16 ***
## bpm_prev_season_growth            -0.21757    0.68609  -0.317    0.751
## bpm_scaled_to_salary_prev_season  0.47162    0.03076  15.333   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.536 on 486 degrees of freedom
## Multiple R-squared:  0.3419, Adjusted R-squared:  0.3392
## F-statistic: 126.2 on 2 and 486 DF,  p-value: < 2.2e-16
```

```
bpm_to_salary_prediction_exclude_growth <- lm(bpm_scaled_to_salary ~ bpm_scaled_to_salary_prev_season, data = pre
diction_data)
summary(bpm_to_salary_prediction_exclude_growth)
```

```
##
## Call:
## lm(formula = bpm_scaled_to_salary ~ bpm_scaled_to_salary_prev_season,
##     data = prediction_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.676  -1.997  -1.157   1.050  18.800
##
## Coefficients:
##                                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)                        2.82884    0.29221   9.681   <2e-16 ***
## bpm_scaled_to_salary_prev_season  0.46887    0.02949  15.901   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.531 on 487 degrees of freedom
## Multiple R-squared:  0.3417, Adjusted R-squared:  0.3404
## F-statistic: 252.8 on 1 and 487 DF,  p-value: < 2.2e-16
```

Well, the regression really didn't work. Growth was not statistically significant in predicting cost-efficiency for some reason, and the R-squared value is pretty low to the point where I'd say cost-efficiency from the previous season does not explain cost-efficiency for the predicted season well at all.

Let's try KNN to see if we can build a model that could be more helpful than this regression model.

Predicting cost-efficiency: KNN

I think Fred VanVleet in 2020 was an amazing value player with a raw BPM of 2.7 on a salary of just $8 million. That's similar contributions per possession as Ben Simmons in 2021, who's in Top 3 for Defensive Player of the Year votes and was named to the all-star team. Simmons is paid roughly 30 million. Naturally, we can use FVV as the threshold for a very cost efficient player.

```
#determine whether a player is more cost-efficient than 2019 FVV
criteria_for_cost_efficient <- prediction_data[prediction_data$player == "Fred VanVleet" & prediction_data$year =
= 2020,]$bpm_scaled_to_salary
determine_player_cost_efficiency <- function(x) {
  super_cost_efficient <- "Super cost efficient"
  not_super_cost_efficient <- "Not super cost efficient"
  if (x < criteria_for_cost_efficient) {
    return(not_super_cost_efficient)
  } else {
    return(super_cost_efficient)
  }
}
prediction_data$cost_efficient <- sapply(prediction_data$bpm_scaled_to_salary, determine_player_cost_efficiency)
```

```
#seperate test and training datasets
set.seed(123)
train_size <- floor(0.75 * nrow(prediction_data))
train_indices <- sample(seq_len(nrow(prediction_data)), size = train_size)
train_data_all <- prediction_data[train_indices,]
test_data_all <- prediction_data[-train_indices,]
```

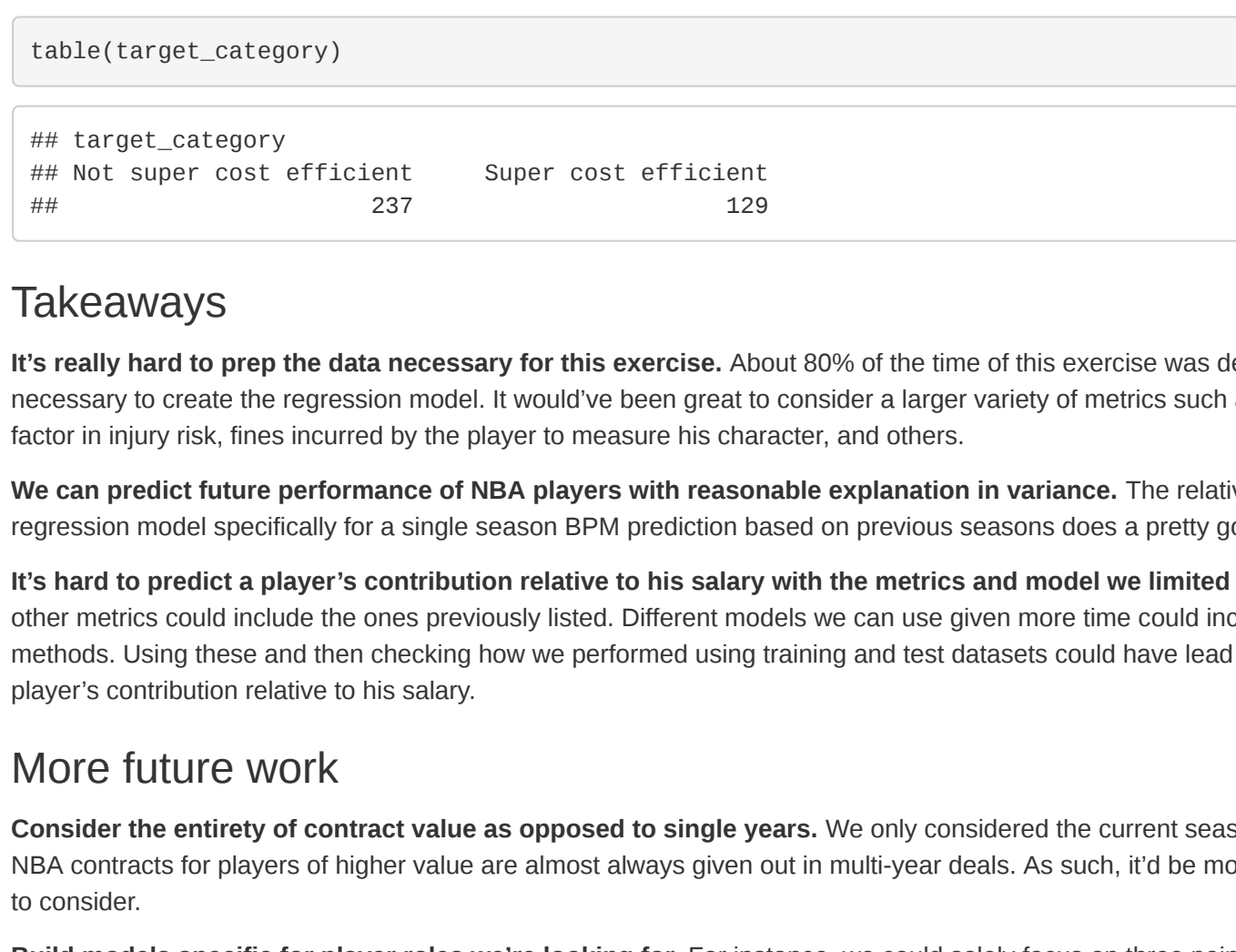
```
#get necessary features to run KNN classification
target_category <- train_data_all$cost_efficient
test_category <- test_data_all$cost_efficient
train_data <- train_data_all %>% select(bpm_prev_season, bpm_prev_season_growth, salary_pct_prev_season)
test_data <- test_data_all %>% select(bpm_prev_season, bpm_prev_season_growth, salary_pct_prev_season)
```

```
#function to calculate accuracy
accuracy <- function(x) {
  sum(diag(x) / sum(rowSums(x))) * 100
}
```

```
#find k that maximizes accuracy
max_k <- -1
max_accuracy <- -1
range_k <- 2:100
accuracy_per_k <- c()
for (k in 2:100) {
  test_pred <- knn(train_data, test_data, target_category, k = 50)
  test_tab <- table(test_pred, test_category)
  pred_accuracy <- accuracy(test_tab)
  if (pred_accuracy > max_accuracy) {
    max_k <- k
    max_accuracy <- pred_accuracy
  }
  accuracy_per_k <- c(accuracy_per_k, pred_accuracy)
}
string_to_print <- paste0("The k with the highest accuracy was ", max_k, " with accuracy of ", round(max_accurac
y, 4))
print(string_to_print)
```

```
## [1] "The k with the highest accuracy was 69 with accuracy of 77.2358"
```

```
accuracy_plot_df <- cbind(data.frame(range_k), data.frame(accuracy_per_k))
ggplot(data = accuracy_plot_df, aes(x = range_k, y = accuracy_per_k)) + geom_point() + geom_line() + ylim(0, 100)
```



The k-value vs. accuracy graph is suspiciously stable at around 75%, but I'm assuming this is due to a small training dataset at only 366 entries. For the record, the "not super cost efficient" players comprise of 64.7% of the training data (as seen below), so the KNN model does predict better than simply guessing "not super cost efficient" all the time.

```
table(target_category)
```

```
## target_category
## Not super cost efficient     Super cost efficient
##                      237                      129
```

Takeaways

It's really hard to prep the data necessary for this exercise. About 80% of the time of this exercise was dedicated to prepping the data necessary to create the regression model. It would've been great to consider a larger variety of metrics such as missed games without a DNP to factor in injury risk, fines incurred by the player to measure his character, and others.

We can predict future performance of NBA players with reasonable explanation in variance. The relatively higher R^2 value indicates that our regression model specifically for a single season BPM prediction based on previous seasons does a pretty good job.

It's hard to predict a player's contribution relative to his salary with the metrics and model we limited ourselves to. We notice that The other metrics could include the ones previously listed. Different models we can use given more time could include logistic regression or clustering methods. Using these and then checking how we performed using training and test datasets could have lead to more effective ways to predict a player's contribution relative to his salary.

More future work

Consider the entirety of contract value as opposed to single years. We only considered the current season's salary for this exercise. In reality, NBA contracts for players of higher value are almost always given out in multi-year deals. As such, it'd be more but admittedly much more difficult to consider.

Build models specific for player roles we're looking for. For instance, we could solely focus on three point shooting and defense to find 3&D role players that fit our team.

Set more benchmarks and see if the KNN model still holds in accuracy. For example, we can have three sets of cost-efficiency like "good," "meh," and "bad." We can test if our KNN model still stands up if it's provided with more categories.