Final Project

The following block reads in the data sets containing player statistics and salaries for the 2019-2020 and 2020-2021 seasons. It also changes a few variable names to ensure they match for the join.

```
player_stats_2019_20 <- read.csv("2019-20_player_stats.csv")
player_salaries_2019_20 <- read.csv("2019-20_player_salaries.csv")
#player_stats_2019_20
colnames(player_salaries_2019_20)[3] <- "Player"
#player_salaries_2019_20

player_stats_2020_21 <- read.csv("2020-21_player_stats.csv")
player_salaries_2020_21 <- read.csv("2020-21_player_salaries.csv")
#player_stats_2020_21
colnames(player_salaries_2020_21)[3] <- "salary"
#player_salaries_2020_21
```

The following code joins the player statistics and player salary data sets into one data frame by different years. It also removes the cases where players were traded in the middle of the season to prevent any duplicate salary information from being included.

```
merged_2019_20 <- merge(player_salaries_2019_20, player_stats_2019_20, by = 'Player')
merged_2019_20 <- merged_2019_20 %>%
  distinct(Player, .keep_all = TRUE)

merged_2020_21 <- merge(player_salaries_2020_21, player_stats_2020_21, by = 'Player')
merged_2020_21 <- merged_2020_21 %>%
  distinct(Player, .keep_all = TRUE)
```

The following code divides our tables into 3 different sets. We will train our multiple-regression model on the 2019-2020 season. The train data frame will additionally be divided into a dev data frame to compare with our test data frame to make sure our multiple-regression model remains consistent over different data frames. We also end up using the 2020-2021 season as our test data frame.

```
data_split <- sort(sample(nrow(merged_2019_20), nrow(merged_2019_20)*0.8))
train <- merged_2019_20[data_split,]
dev <- merged_2019_20[-data_split,]
test <- merged_2020_21
```

Here we create our multiple-regression model.

```
model <- lm(salary ~ 1 + Age + G + GS + BLK + TOV + PF + PTS, train)
summary(model)
```

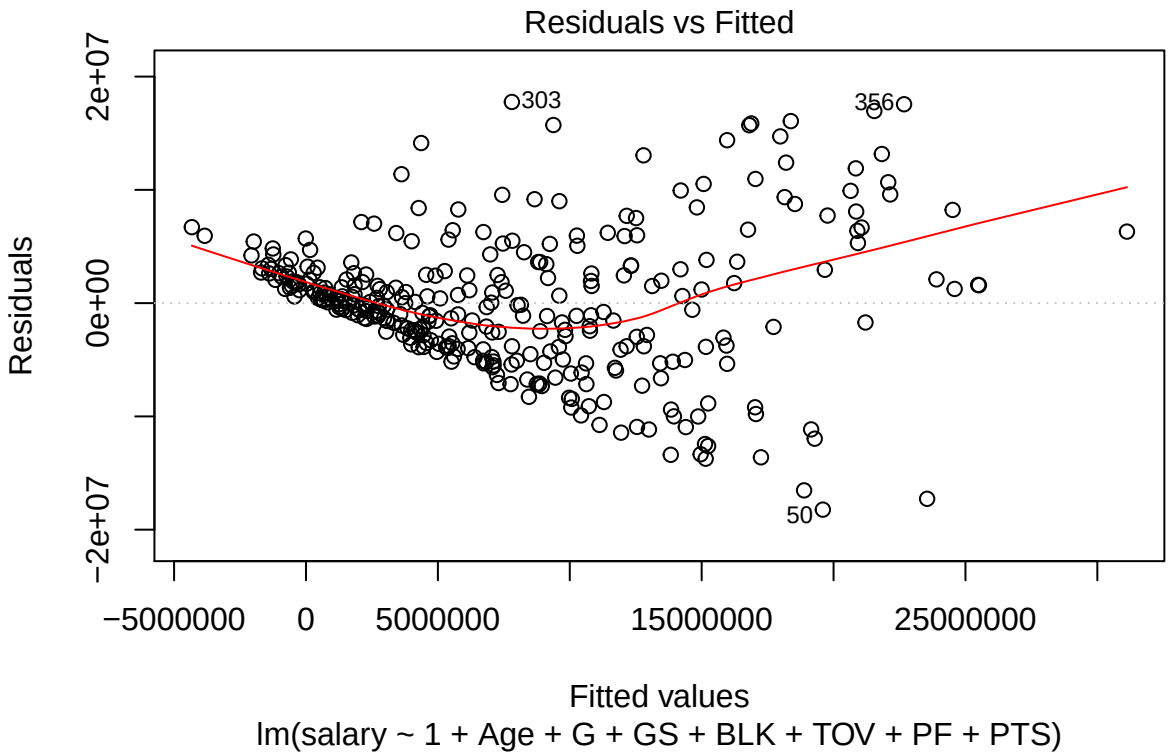
```
##
## Call:
```

```
## lm(formula = salary ~ 1 + Age + G + GS + BLK + TOV + PF + PTS,
##     data = train)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -18230431  -3644300     45352   2701047  17759293
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -16278516    2324014  -7.004 1.46e-11 ***
## Age            770489      79813   9.654  < 2e-16 ***
## G              -71726      19806  -3.621 0.000340 ***
## GS              81126      21627   3.751 0.000209 ***
## BLK           2400295    1002816   2.394 0.017260 *
## TOV           1623942     841481   1.930 0.054505 .
## PF           -1701002     637639  -2.668 0.008026 **
## PTS            606029     119264   5.081 6.37e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6117000 on 321 degrees of freedom
## Multiple R-squared:  0.5431, Adjusted R-squared:  0.5331
## F-statistic: 54.51 on 7 and 321 DF,  p-value: < 2.2e-16
```

Based on the variables provided on the players, without using their names, we are able to achieve an R-squared value of **0.5622**. The most important variables are "Age", "PTS", and "G". This makes sense as younger players lack experience that would make them more valuable to the team and older players are more likely to have a harder time keeping up with the speed of the game. As for points, it is pretty self explanatory as players who score more points are more valuable to their team. Additionally, games played is important because a team does not want to pay a player that doesn't play.

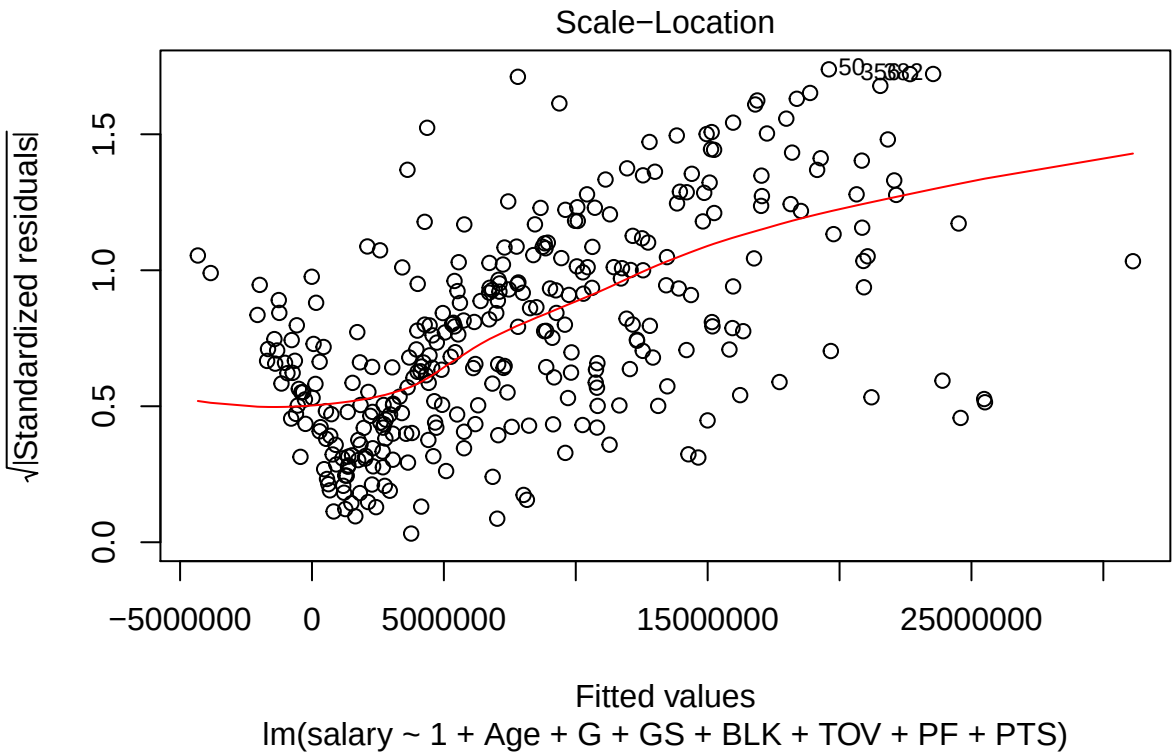
As the primary goal of this model is to predict the salary variable, we can use the R-squared statistic to analyze how good the model. It is clear that a R-squared of **0.5622** is not very good. In addition, if we look at the model diagnostics, it becomes pretty clear that our current model is not very good. The first one we will look at is the Residual vs Fitted plot.

```
plot(model, which = 1)
```



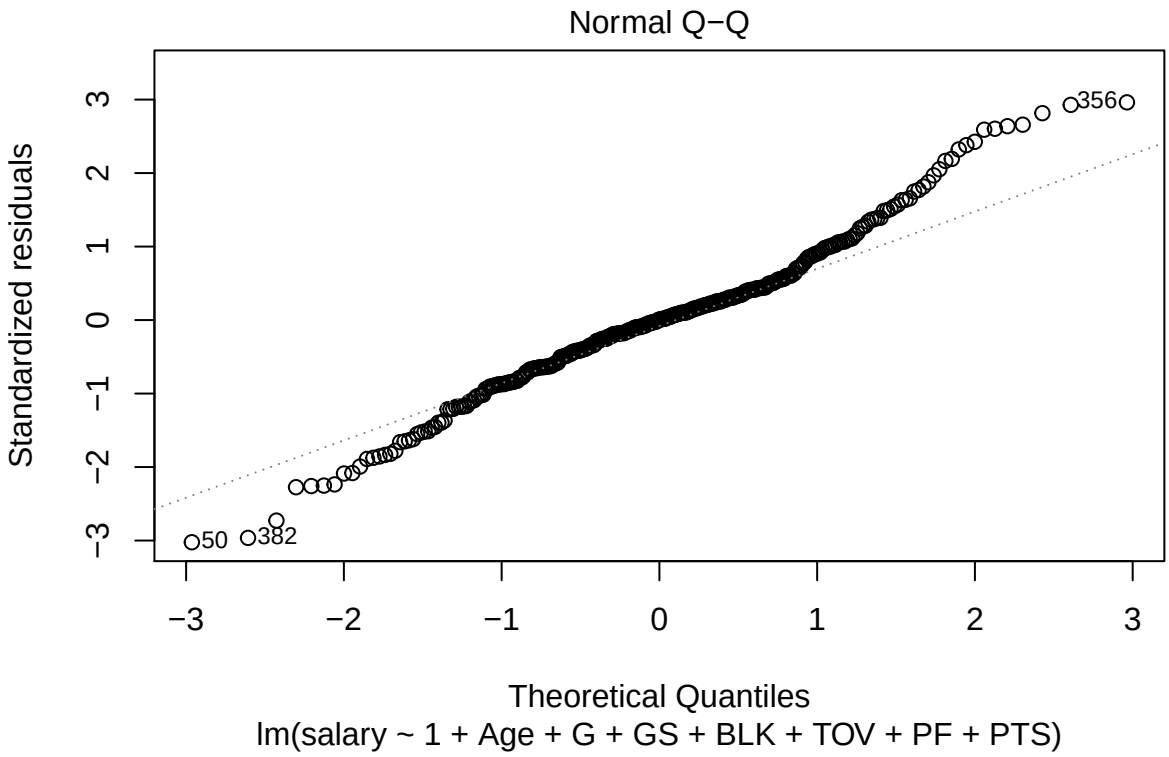
You can see that there is a "funneling" effect on the graph. The distribution of the residuals is concentrated around 0 for small fitted values, but they get more and more spread out as the fitted values increase. This phenomena is called increasing variance. And, the Scale-Location graph also confirms this below.

```
plot(model, which = 3)
```



There is a clear upward trend in the red line which reaffirms our previous statement that the model has non-constant variance. One more thing that we should look at is the Normal QQ plot which lets us assess whether the residuals are normally distributed.

```
plot(model, which = 2)
```



We observe heavier tails in this graph which tells us that tge model has larger values that we would expect under the standard modeling assumptions. But, it looks like normality assumption is still valid.

It is now clear that our model is not very good, so we have to resort to other methods. Our orignial plan was to use ridge regression to achieve a better model but an assumption of ridge regression is constant variance so we cannot use it. Lasso regression also has the same assumptions so we cannot use that one either. We had to resort to some online reserach and we found a method to acheive a lienar model with non-constant variances. We found out that the technical term for non-constant variance is heteroskedasticity. We found a good resource to solve this problem and I have provided the link below: http://bkenkel.com/psci8357/notes/05-ncv.html

The resource tells us that in order to solve heteroskedasticity of an unknown form we can use something called White's heteroskedasticity-consistent estimator which is what we did below

```
library(car)
```

```
## Loading required package: carData
```

```
##
## Attaching package: 'car'
```

```
## The following object is masked from 'package:dplyr':
##
##     recode
```

```
## The following object is masked from 'package:purrr':
```

```
##
##     some
```

```
library(lmtest)
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
vcv0 <- hccm(model, type = "hc0")
vcv0
```

```
##               (Intercept)           Age           G          GS           BLK
## (Intercept) 6314322708699 -231611351785 -1017777286  9027192865  469362807435
## Age         -231611351785    9793501114  -286716354   -56170678  -12450353153
## G             -1017777286    -286716354   331690263   -35082669    1180291429
## GS             9027192865     -56170678   -35082669   492524761     542049122
## BLK          469362807435  -12450353153  1180291429   542049122  874920777867
## TOV         -177456867893    3973653705  1904582532 -4717089967  141178595665
## PF          -521346928982    8425719078 -1913607988 -3316222656 -295283562182
## PTS           20367564803    -847218035  -665274808  -476344012  -27315148245
##                       TOV            PF          PTS
## (Intercept) -177456867893 -521346928982  20367564803
## Age            3973653705    8425719078   -847218035
## G              1904582532   -1913607988   -665274808
## GS            -4717089967   -3316222656   -476344012
## BLK          141178595665 -295283562182 -27315148245
## TOV         1010326126776 -199782818984 -84105865643
## PF          -199782818984  419241420119   4326065818
## PTS          -84105865643    4326065818  16744690845
```

```
library("broom")
modelv2 <- tidy(coeftest(model, vcov = vcv0))
```

This model know has accounted for the heteroskedacticity and we can now perform the predictions in the model.

Now we will predict on our dev data frame and test data frame.

```
dev_predictions <- predict(model, newdata = dev)
test_predictions <- predict(model, newdata = test)
```