Final Project

Hung Nguyen (8215071)

3/6/2023

INTRODUCTION

In this project, I would like to predict a soccer player's market value (in Euros) using the player's ratings and stats from the game FIFA 23. Furthermore, I would also like to see if there exists any significant connection between the response which is the market value and any specific individual base stat. In soccer, base stats include the main skills that a player is rated on and they are Pace, Shooting, Passing, Dribbling, Defending, and Physicality. Within each of these base stats, there exists more specific skills. For example, within the base stat Shooting, there are more specific skills such as Finishing, Heading, Volley, etc.

The response variable is a player's market value. A player's market value is determined by the transfer market and not by the club the player is contracted to nor the player himself. Many aspects go into consideration when determining a player's market value besides from the player's skills such as performance, image, age, potential to grow, selling strength, etc. The market value is highly taken into consideration in the process of selling or purchasing players between clubs as the price that a club sets out to sell a player is influenced by the market value. Similarly, the negotiation prices given by the club purchasing the player are also influenced by the market value.

As mentioned above, I am planning on using 13 predictors and they are: overall rating, potential rating, age, height, weight, wage, international reputation, pace, shooting, passing, dribbling, defending, and physicality. These variables are used by transfer market panel in the process of determining a player's market value so I thought they would be useful for this project.

The project is mainly for predictive purposes as I would like to focus on predicting the market value of a player. However, I am also interested in whether if certain stats of a player are more favorable to give a player a higher market value.

LOAD PACKAGES

```
library(janitor)
library(tidyverse)
library(tidymodels)
library(glmnet)
library(ggplot2)
library(vip)
library(rpart.plot)
library(ranger)
library(caret)
library(kernlab)
library(car)
library(corrplot)
library(MASS)
library(kknn)
```

READ & FORMAT DATA

The original dataset includes information of approximately 18,000 players, both well-known and not well-known. For this project, I would like to focus on predicting the market value of players whose overall ratings are higher than 70 since not only is there more information about them, their information is also more accurate and more frequently updated.

```
# Loading the dataset
fifa <- read.csv("Fifa 23 Players Data.csv")
fifa

# Subsetting the dataset to only include variables of interest
data <- fifa[c('Overall', 'Potential', 'Value.in.Euro.', 'Age', 'Height.in.cm.', 'Wei
ght.in.kg.', 'Wage.in.Euro.', 'International.Reputation', 'Pace.Total', 'Shooting.Tot
al', 'Passing.Total', 'Dribbling.Total', 'Defending.Total', 'Physicality.Total')]

# Filtering the dataset to exclude players with ratings lower than 70 and market valu
es equal 0
data <- data %>% filter(Overall >= 70)
data <- data %>% filter(Value.in.Euro. > 0)

# Renaming variables
names(data)[names(data) == "Overall"] <- "overall"
names(data)[names(data) == "Potential"] <- "potential"
names(data)[names(data) == "Value.in.Euro."] <- "value"
names(data)[names(data) == "Age"] <- "age"
names(data)[names(data) == "Height.in.cm."] <- "height"
names(data)[names(data) == "Weight.in.kg."] <- "weight"
names(data)[names(data) == "Wage.in.Euro."] <- "wage"
names(data)[names(data) == "International.Reputation"] <- "rep"
names(data)[names(data) == "Pace.Total"] <- "pace"
names(data)[names(data) == "Shooting.Total"] <- "shoot"
names(data)[names(data) == "Passing.Total"] <- "pass"
names(data)[names(data) == "Dribbling.Total"] <- "dribble"
names(data)[names(data) == "Defending.Total"] <- "defend"
names(data)[names(data) == "Physicality.Total"] <- "physicality"

data
```

SPLIT & FOLD DATA

I performed an 80% random split for the training set and saved the remaining 20% for the testing set, stratified on the response variable `value`. I also folded the training dataset 5 times to perform cross-validation. After performing the split, I checked the dimensions of the training set and testing set to verify the number of observations in each set. The training set has 4130 observations and the testing set has 1034 observations.

```
set.seed(28)

# Splitting data
data.split <- initial_split(data, prop = 0.80, strata = "value")
data.train <- training(data.split)
data.test <- testing(data.split)

# Checking dimension of the training and testing set
dim(data.train)
```

```
## [1] 4130   14
```

```
dim(data.test)
```

```
## [1] 1034   14
```

```
# Folding data
data.fold <- vfold_cv(data.train, v = 5)
```

LINEAR REGRESSION

Diagnostics

To perform linear regression, the dataset must satisfy certain assumptions. Thus, I performed a diagnostic test to check for:

1. Linearity of the data: The relationship between the predictor variables and the response variable is assumed to be linear. This is evaluated by the Residuals vs Fitted plot (want horizontal line to follow assumption).
2. Normality of the errors: The error terms are assumed to follow a normal distribution. This is evaluated by the Normal Q-Q plot (want y = x line to follow assumption).
3. Homoscedasticity: The error terms are assumed to have constant variance. This is evaluated by the Scale-Location plot (want horizontal line to follow assumption).
4. Non-collinearity: The predictor variables are assumed to not be closely related to one another (they should not increase or decrease together). This is evaluated by the variance inflation factor (want a value of 5.0 or lower).
5. Outliers: a data point whose response y does not follow the general trend of the rest of the data. This is evaluated by the studentized residuals.
6. High-leverage points: a data point whose predictor x has extreme values. This is evaluated by the leverage statistics (want a value lower than 2).

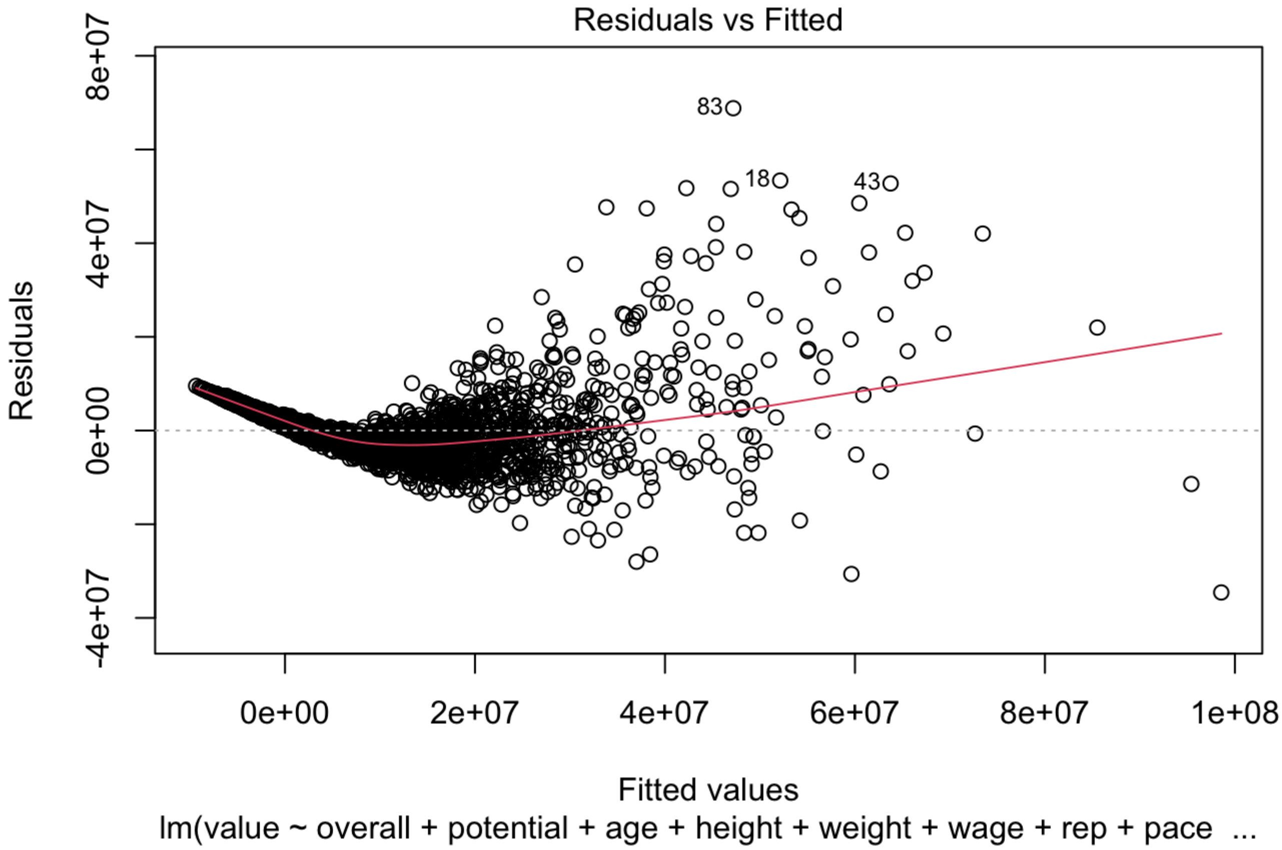
It is evident from these diagnostic tests that the dataset does not satisfy the assumptions required to perform linear regression. For instance, the residuals vs. fitted plot of the linear model cannot be described by a horizontal line which means that the relationship between the predictor variables and the response variable is not linear. Furthermore, as depicted by the Normal Q-Q plot, the points do not follow the y = x line which means that the error terms do not follow a normal distribution. The Scale-Location plot cannot be described by a horizontal line which means the error terms do not have constant variance. Collinearity also exists in the dataset as the variables `overall`, `potential`, and `dribble` have a VIF higher than 5.0.

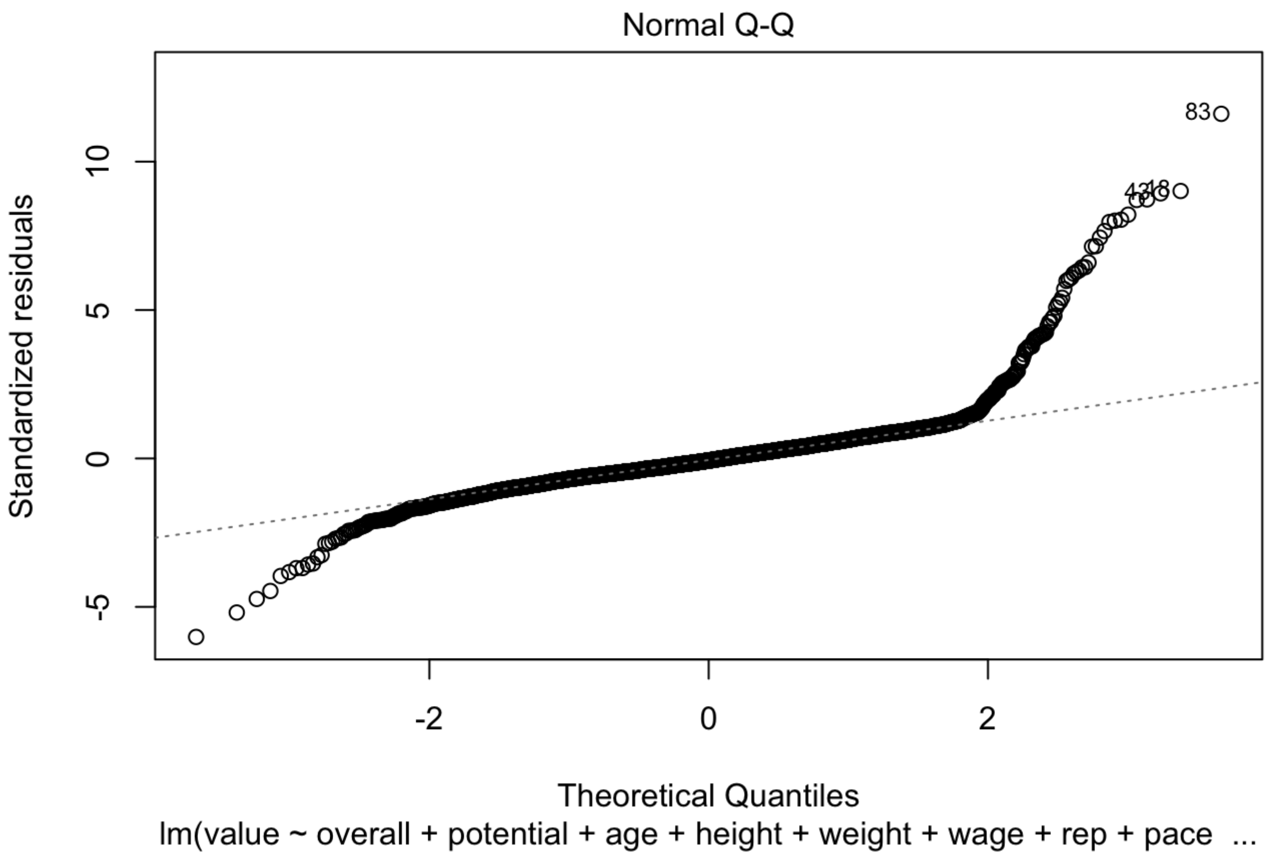
However, I would like to still fit a linear regression model to assess how the model will perform despite the dataset not satisfying the required conditions. From the results of the diagnostic tests, I predict that the linear models would not perform well in predicting a player's market value.

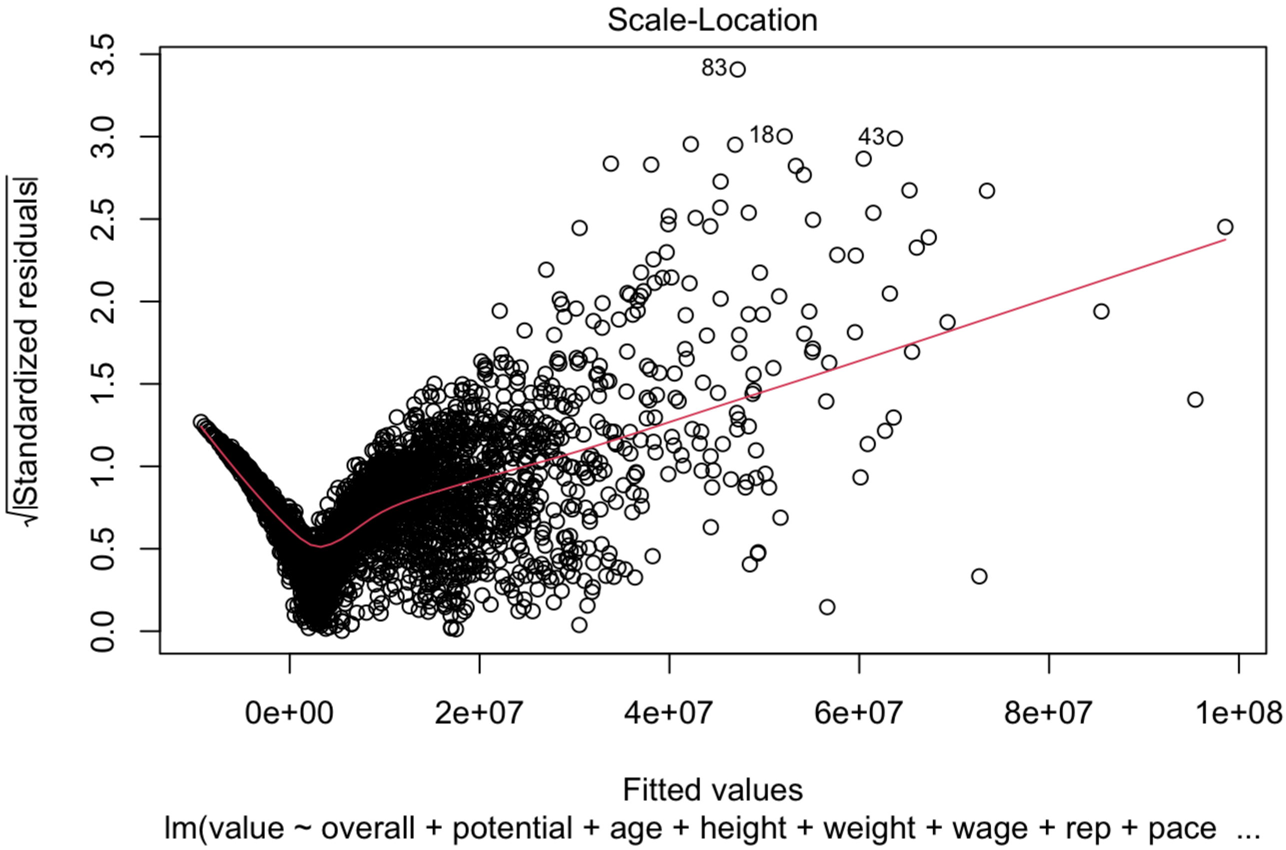
```
set.seed(28)

# Building initial linear regression model
l.mod <- lm(value ~ overall + potential + age + height + weight + wage + rep + pace +
shoot + pass + dribble + defend + physicality, data = data.train)
summary(l.mod)
```

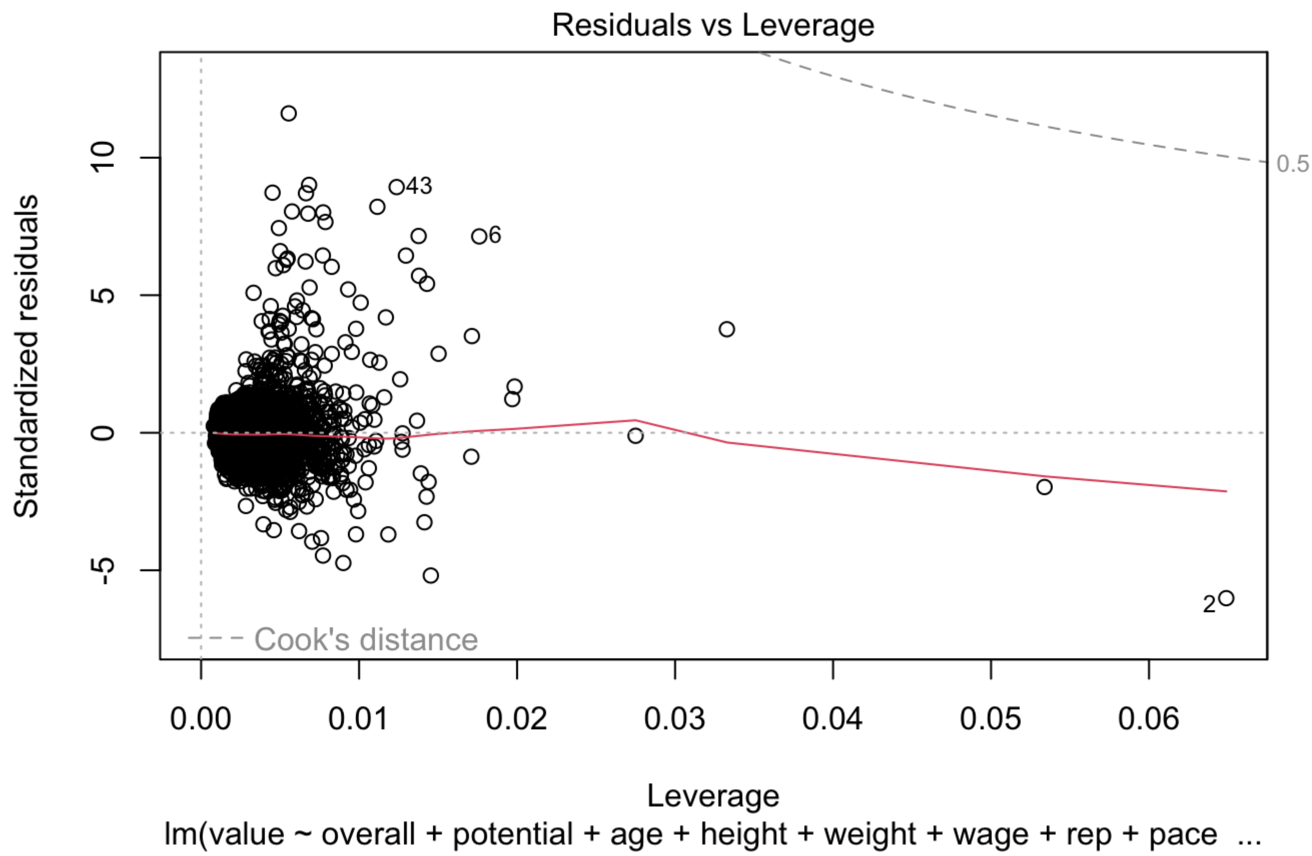
```
##
## Call:
## lm(formula = value ~ overall + potential + age + height + weight +
##     wage + rep + pace + shoot + pass + dribble + defend + physicality,
##     data = data.train)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -34562721  -2947851   -403996   2343204  68812765
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.092e+08  4.446e+06 -24.560  < 2e-16 ***
## overall      1.672e+06  6.480e+04  25.802  < 2e-16 ***
## potential    2.235e+05  4.826e+04   4.632 3.74e-06 ***
## age         -5.058e+05  3.979e+04 -12.710  < 2e-16 ***
## height      -6.399e+04  2.238e+04  -2.859 0.004274 **
## weight      -1.669e+04  2.191e+04  -0.762 0.446223
## wage         1.418e+02  4.537e+00  31.258  < 2e-16 ***
## rep          5.969e+05  2.125e+05   2.808 0.005002 **
## pace         1.990e+04  1.127e+04   1.766 0.077456 .
## shoot        1.130e+04  1.439e+04   0.785 0.432550
## pass         3.072e+04  2.263e+04   1.358 0.174679
## dribble     -9.829e+04  2.956e+04  -3.325 0.000891 ***
## defend      -3.830e+04  9.890e+03  -3.873 0.000109 ***
## physicality  5.564e+04  1.735e+04   3.207 0.001352 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5943000 on 4116 degrees of freedom
## Multiple R-squared:  0.7821, Adjusted R-squared:  0.7814
## F-statistic:  1137 on 13 and 4116 DF,  p-value: < 2.2e-16
```

```
# LINEARITY, NORMALITY, HOMOSCEDASTICITY
plot(l.mod)
```

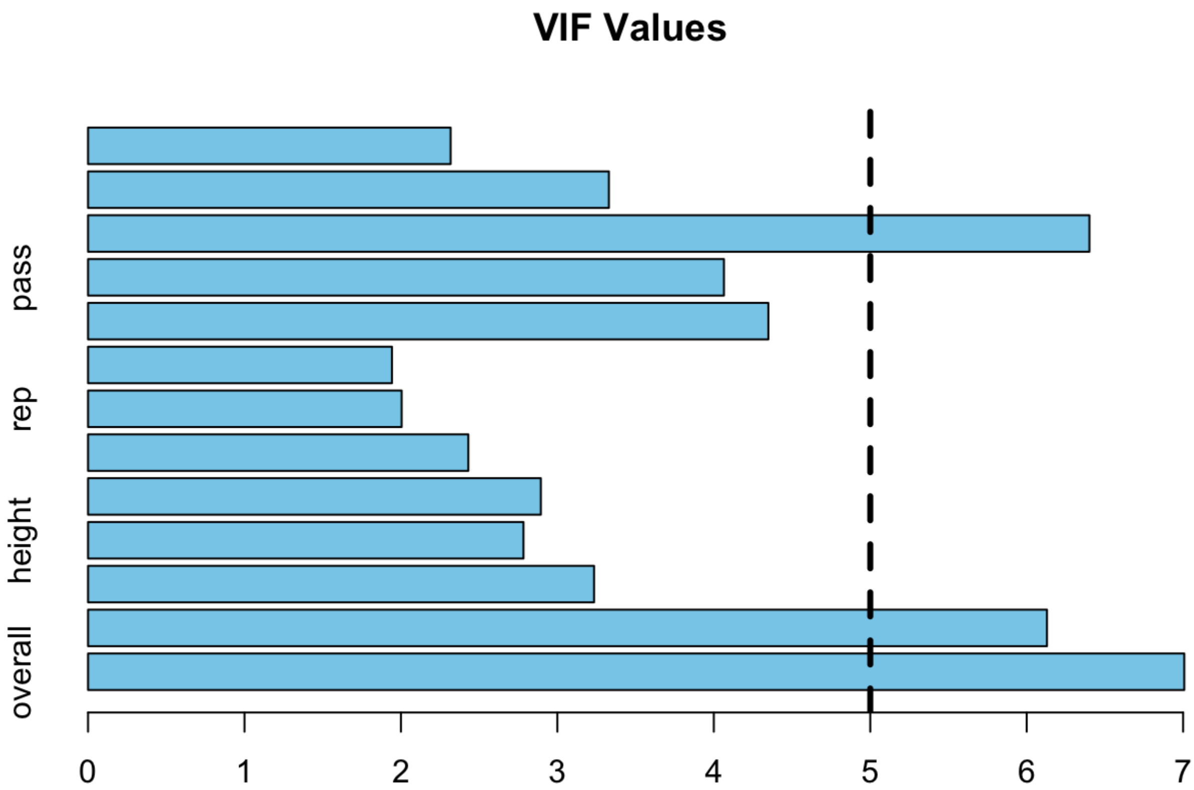

Residuals vs Fitted
Residuals
8e+07
4e+07
0e+00
-4e+07
83
18
43
0e+00
2e+07
4e+07
6e+07
8e+07
1e+08
Fitted values
lm(value ~ overall + potential + age + height + weight + wage + rep + pace ...


Normal Q-Q
Standardized residuals
10
5
0
-5
83
43
18
-2
0
2
Theoretical Quantiles
lm(value ~ overall + potential + age + height + weight + wage + rep + pace ...

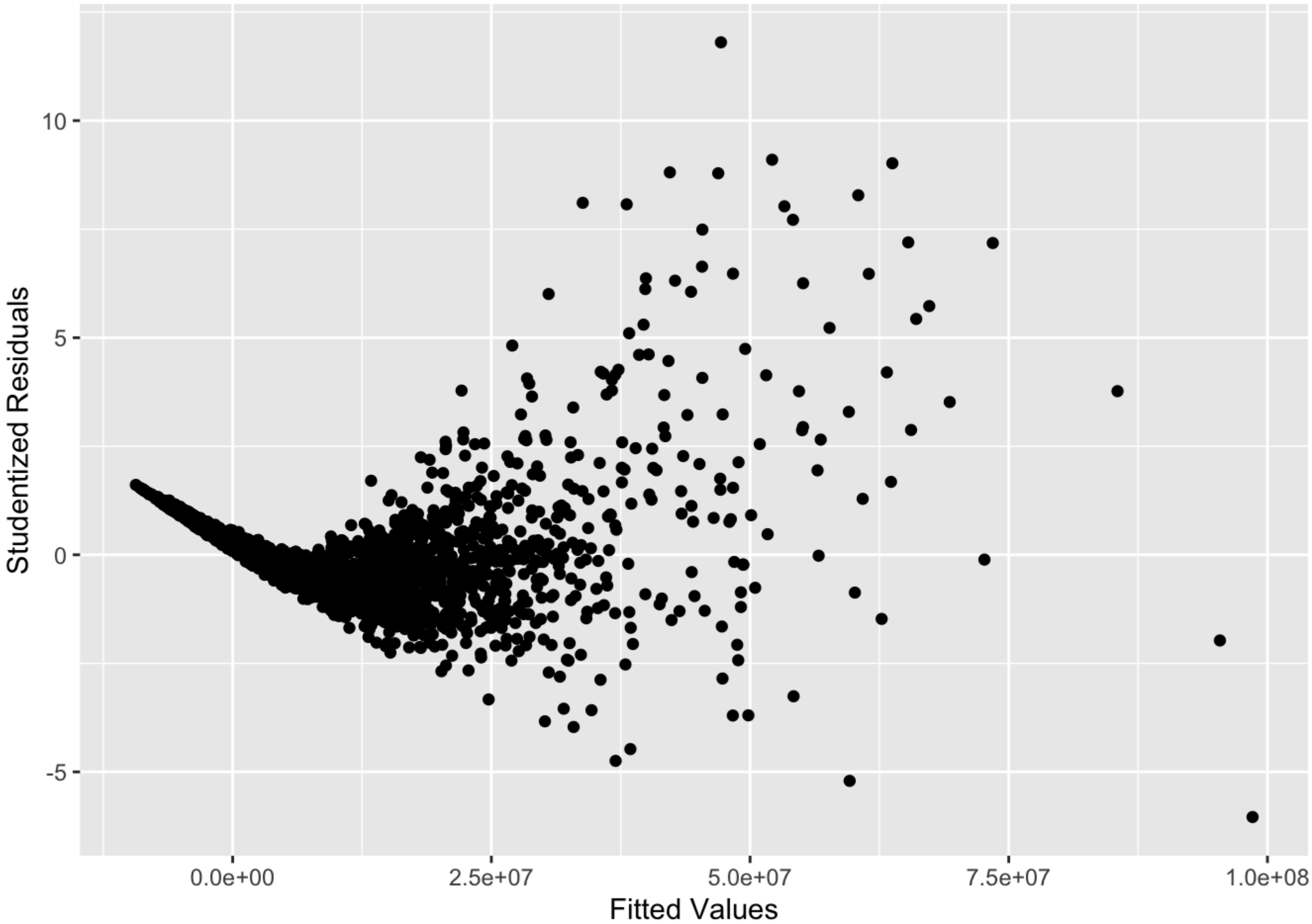

Scale-Location
√|Standardized residuals|
Fitted values
lm(value ~ overall + potential + age + height + weight + wage + rep + pace ...
83
18
43
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0e+00
2e+07
4e+07
6e+07
8e+07
1e+08


Residuals vs Leverage
Standardized residuals
Leverage
lm(value ~ overall + potential + age + height + weight + wage + rep + pace ...
Cook's distance
0.5
43
6
2
-5
0
5
10
0.00
0.01
0.02
0.03
0.04
0.05
0.06

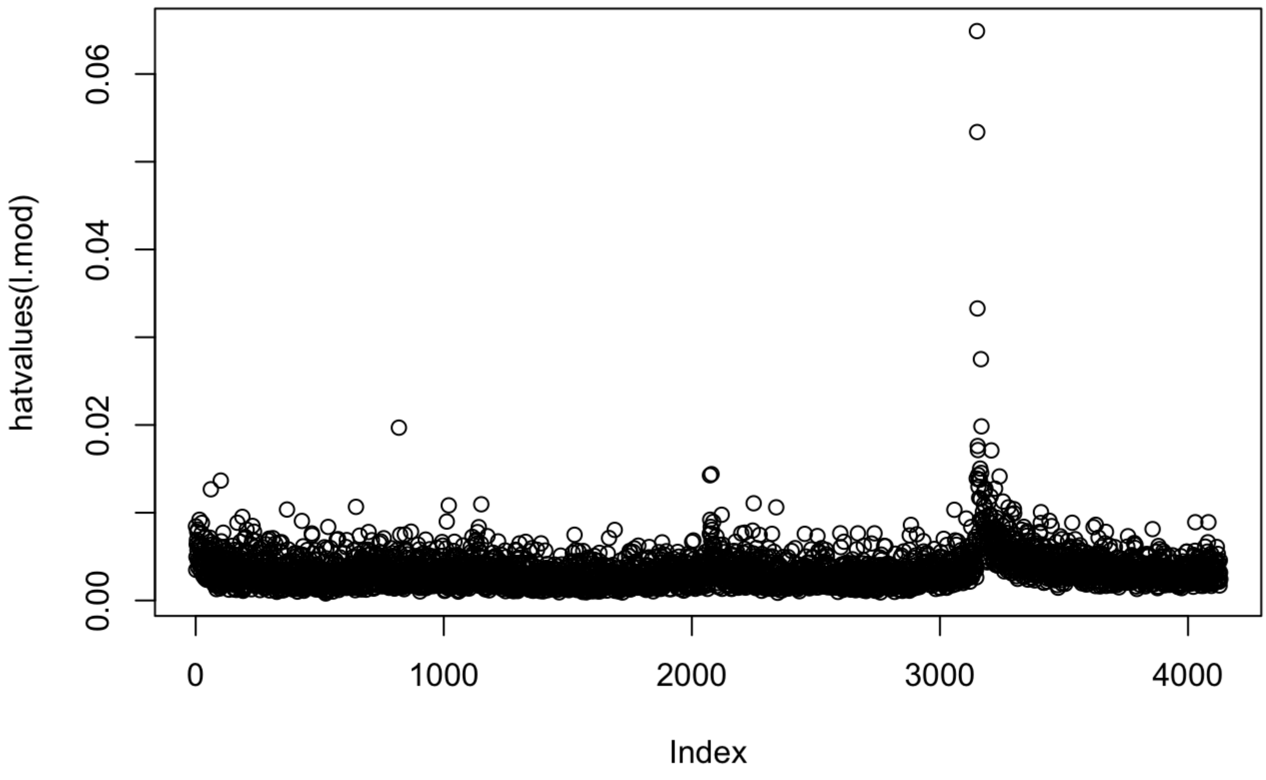
```
# COLLINEARITY
# create horizontal bar chart to display each VIF value
barplot(vif(l.mod), main = "VIF Values", horiz = TRUE, col = "skyblue")
# add vertical line at 5 as after 5 there is severe correlation
abline(v = 5, lwd = 3, lty = 2)
```




```
# OUTLIERS
# create a plot of studentized residuals against fitted values
ggplot(l.mod, aes(x=fitted(l.mod), y=studres(l.mod))) + geom_point() + xlab("Fitted V
alues") + ylab("Studentized Residuals")
```




```
# HIGH-LEVERAGE POINTS
plot(hatvalues(l.mod))
```



Fitting

In this step, I created a universal recipe for all of the models that I wish to build in this project. I used the step_normalize() function to center and scale all of the predictors. In addition, I created a model and workflow to perform linear regression.

```
set.seed(28)

# Recipe
rec <- recipe(value ~ overall + potential + age + height + weight + wage + rep + pace
+ shoot + pass + dribble + defend + physicality, data = data.train) %>% step_normaliz
e(all_predictors())
prep(rec) %>% bake(new_data = data.train) %>% head()
```

```
## # A tibble: 6 × 14
##   overall potent…¹   age height weight     wage    rep  pace shoot  pass dribble
##     <dbl>    <dbl> <dbl>  <dbl>  <dbl>    <dbl>  <dbl> <dbl> <dbl> <dbl>   <dbl>
## 1    1.32    0.544  3.00  2.06   1.80  -0.437    2.77  0.434 1.36  0.193   0.647
## 2    1.32    0.544  1.81  1.19   0.542  0.129    1.14  0.783 1.13  1.41    1.28
## 3    1.32    0.544  2.28  0.462  0.820  0.00314  2.77  0.521 1.28  0.921   1.28
## 4    1.32    0.544  2.05  1.19  -0.294  0.286    2.77  0.434 1.36  0.678   0.773
## 5    1.32    0.544  1.81  1.04   1.66  -0.469   -0.479 0.609 0.910 0.314   1.78
## 6    1.05    0.333  2.76  0.317  0.542 -0.437    2.77  0.521 1.21  2.13    0.900
## # … with 3 more variables: defend <dbl>, physicality <dbl>, value <int>, and
## #   abbreviated variable name ¹potential
```

```
# Model
lin.mod <- linear_reg() %>% set_engine("lm") %>% set_mode("regression")
# Workflow
lin.wf <- workflow() %>% add_model(lin.mod) %>% add_recipe(rec)

# Fit
lin.fit <- fit(lin.wf, data.train)
lin.fit %>% extract_fit_parsnip() %>% tidy()
```

```
## # A tibble: 14 × 5
##    term         estimate std.error statistic   p.value
##    <chr>           <dbl>     <dbl>     <dbl>     <dbl>
##  1 (Intercept)  8124948.    92483.    87.9   0
##  2 overall      6317602.   244845.    25.8   3.31e-136
##  3 potential    1060623.   228999.     4.63  3.74e-  6
##  4 age         -2114317.   166346.   -12.7   2.47e- 36
##  5 height       -441138.   154311.    -2.86  4.27e-  3
##  6 weight       -119873.   157356.    -0.762 4.46e-  1
##  7 wage         4507737.   144209.    31.3   1.19e-192
##  8 rep           367771.   130952.     2.81  5.00e-  3
##  9 pace          227674.   128915.     1.77  7.75e-  2
## 10 shoot         151394.   192880.     0.785 4.33e-  1
## 11 pass          253156.   186479.     1.36  1.75e-  1
## 12 dribble      -778154.   234011.    -3.33  8.91e-  4
## 13 defend       -653645.   168767.    -3.87  1.09e-  4
## 14 physicality   451586.   140813.     3.21  1.35e-  3
```

Predicting

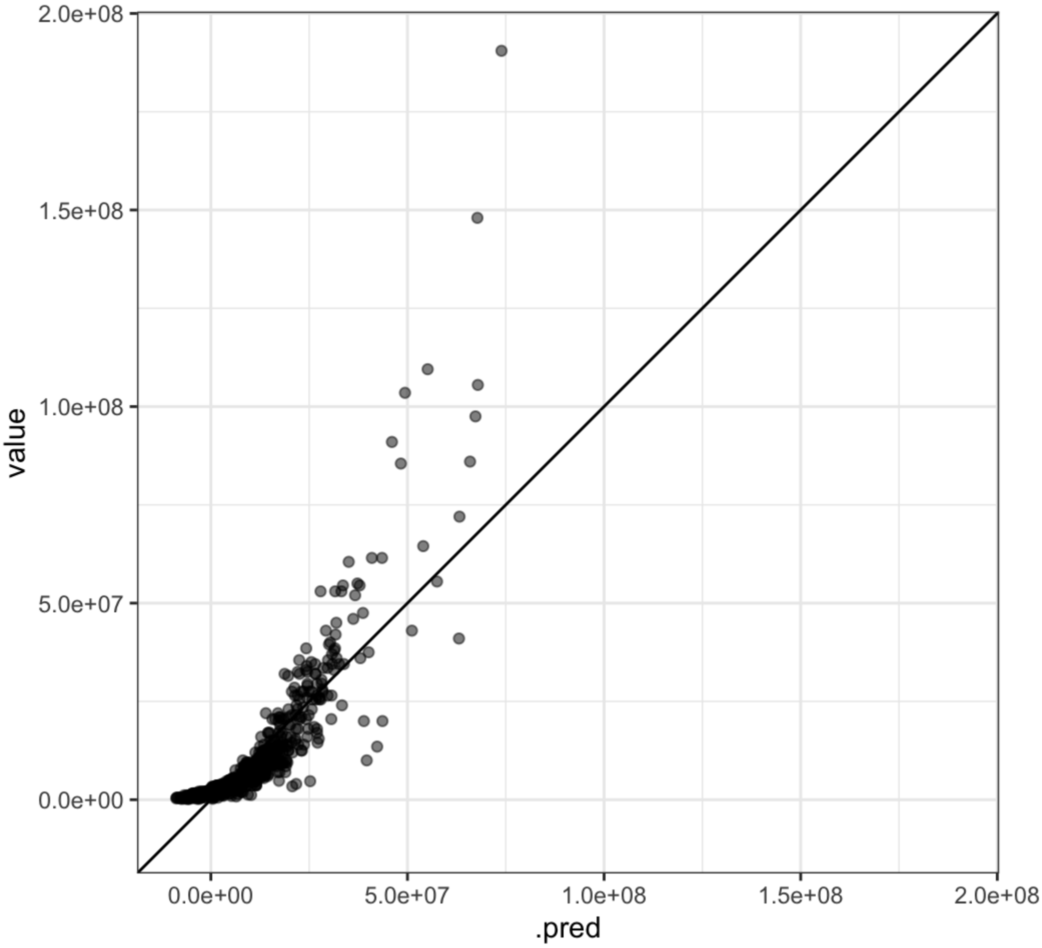
Using the metric set of R-Squared, RMSE, and MAE, I checked the performance of the model on the testing set. $R^2 = 0.7357267$, $RMSE = 7400786$, and $MAE = 3898254$. I then created a Predicted vs. Observed values plot to visualize the performance of the model.

R-Squared: The proportion of variance of the response variable that is explained by the predictor variables. RMSE (Root Mean Square Error): The standard deviation of the residuals; measures how spread out the residuals are. MAE (Mean Absolute Error): The average difference between the predicted values and true observations.

```
set.seed(28)

# create a table with 2 columns: predicted values and observed values from the testin
g set
lin.res <- predict(lin.fit, new_data = data.test %>% dplyr::select(-value))
lin.res <- bind_cols(lin.res, data.test %>% dplyr::select(value))
lin.res
```

```
## # A tibble: 1,034 × 2
##        .pred     value
##        <dbl>     <int>
##  1 73912896. 190500000
##  2 42287908.  13500000
##  3 63097412.  41000000
##  4 67899092. 105500000
##  5 65913377.  86000000
##  6 48321524.  85500000
##  7 63229056.  72000000
##  8 67780046. 148000000
##  9 49375102. 103500000
## 10 67294152.  97500000
## # … with 1,024 more rows
```

```
# create a plot of observed values from testing set against predicted values
lin.res %>% ggplot(aes(x = .pred, y = value)) + geom_point(alpha = 0.5) + geom_abline
(lty = 1) + theme_bw() + coord_obs_pred()
```




```
# checking the performance of the model using metric set of R-Squared, RMSE, and MAE
metrics <- metric_set(rsq, rmse, mae)
metrics(lin.res, truth = value, estimate = .pred)
```

```
## # A tibble: 3 × 3
##   .metric .estimator   .estimate
##   <chr>   <chr>            <dbl>
## 1 rsq     standard         0.736
## 2 rmse    standard   7400786.
## 3 mae     standard   3898254.
```

Analysis

My prediction was correct because with an R-Square of 73.57%, RMSE of 7400786, and MAE of 3898254, this model does not perform as well as desired on the testing set. This can be explained by the dataset's violation of the assumptions required to perform linear regression. Furthermore, the relationship between the response variable and the predictor variables itself is not linear which explains the ineffectiveness of the linear regression model to predict the market value of a soccer player.

RIDGE REGRESSION

Different from the usual linear regression, ridge regression shrinks the regression coefficients so that they are closer to 0. The amount of shrinkage can be tuned using λ. When $\lambda = 0$, there is no shrinkage and ridge regression performs like linear regression. As λ increases, the impact of shrinkage also grows and regression coefficients get closer to 0. Furthermore, ridge regression still performs well even with collinearity present in the predictor variables.

```
set.seed(28)

# Model
ridge.mod <- linear_reg(mixture = 0, penalty = tune()) %>% set_mode("regression") %>%
set_engine("glmnet")
# Workflow
ridge.wf <- workflow() %>% add_model(ridge.mod) %>% add_recipe(rec)
```

Hypertuning Parameters

Next, I performed cross-validation using 5 folds to choose the most optimal value for the tuning parameter λ. I wanted to select two values of λ, one that gives the highest R-Squared value and one that gives the lowest RMSE value to build two different ridge regression models. However, both R-Squared and RMSE are optimized at one value of $\lambda = 1e^{-10}$. Thus, I fit one ridge regression model using the optimal parameter value.

```
set.seed(28)

# Tuning lambda
ridge.pen.grid <- grid_regular(penalty(), levels = 10)
ridge.tune.res <- tune_grid(ridge.wf, resamples = data.fold, grid = ridge.pen.grid)
collect_metrics(ridge.tune.res)
```

```
## # A tibble: 20 × 7
##          penalty .metric .estimator        mean     n      std_err .config
##            <dbl> <chr>   <chr>            <dbl> <int>        <dbl> <chr>
##  1 0.0000000001  rmse    standard   5997622.        5 117155.      Preprocessor…
##  2 0.0000000001  rsq     standard         0.779     5      0.00965 Preprocessor…
##  3 0.00000000129 rmse    standard   5997622.        5 117155.      Preprocessor…
##  4 0.00000000129 rsq     standard         0.779     5      0.00965 Preprocessor…
##  5 0.0000000167  rmse    standard   5997622.        5 117155.      Preprocessor…
##  6 0.0000000167  rsq     standard         0.779     5      0.00965 Preprocessor…
##  7 0.000000215   rmse    standard   5997622.        5 117155.      Preprocessor…
##  8 0.000000215   rsq     standard         0.779     5      0.00965 Preprocessor…
##  9 0.00000278    rmse    standard   5997622.        5 117155.      Preprocessor…
## 10 0.00000278    rsq     standard         0.779     5      0.00965 Preprocessor…
## 11 0.0000359     rmse    standard   5997622.        5 117155.      Preprocessor…
## 12 0.0000359     rsq     standard         0.779     5      0.00965 Preprocessor…
## 13 0.000464      rmse    standard   5997622.        5 117155.      Preprocessor…
## 14 0.000464      rsq     standard         0.779     5      0.00965 Preprocessor…
## 15 0.00599       rmse    standard   5997622.        5 117155.      Preprocessor…
## 16 0.00599       rsq     standard         0.779     5      0.00965 Preprocessor…
## 17 0.0774        rmse    standard   5997622.        5 117155.      Preprocessor…
## 18 0.0774        rsq     standard         0.779     5      0.00965 Preprocessor…
## 19 1             rmse    standard   5997622.        5 117155.      Preprocessor…
## 20 1             rsq     standard         0.779     5      0.00965 Preprocessor…
```

```
# Selecting the best lambdas using two different metrics: R-Squared and Root Mean Squ
are Error
# best R-Squared lambda
ridge.best.pen.rsq <- select_best(ridge.tune.res, metric = "rsq")
ridge.best.pen.rsq
```

```
## # A tibble: 1 × 2
##        penalty .config
##          <dbl> <chr>
## 1 0.0000000001 Preprocessor1_Model01
```

```
# best RMSE lambda
ridge.best.pen.rmse <- select_best(ridge.tune.res, metric = "rmse")
ridge.best.pen.rmse
```

```
## # A tibble: 1 × 2
##        penalty .config
##          <dbl> <chr>
## 1 0.0000000001 Preprocessor1_Model01
```

Fitting and Predicting

Using the metric set of R-Squared, RMSE, and MAE, I checked the performance of the model on the testing set. $R^2 = 0.7325607$, $RMSE = 7497189$, and $MAE = 3839902$. I then created a Predicted vs. Observed values plot to visualize the performance of the model.

```
set.seed(28)

# Final workflow
ridge.final.wf <- finalize_workflow(ridge.wf, ridge.best.pen.rsq)

# Fitting using best lambda
ridge.fit <- fit(ridge.final.wf, data = data.train)
ridge.fit %>% extract_fit_parsnip() %>% tidy()
```

```
## # A tibble: 14 × 3
##    term         estimate      penalty
##    <chr>           <dbl>        <dbl>
##  1 (Intercept)  8124948. 0.0000000001
##  2 overall      4735301. 0.0000000001
##  3 potential    2112321. 0.0000000001
##  4 age         -1432980. 0.0000000001
##  5 height       -315753. 0.0000000001
##  6 weight        -82259. 0.0000000001
##  7 wage         4328703. 0.0000000001
##  8 rep           646306. 0.0000000001
##  9 pace          330442. 0.0000000001
## 10 shoot         180448. 0.0000000001
## 11 pass          183476. 0.0000000001
## 12 dribble      -185286. 0.0000000001
## 13 defend       -274589. 0.0000000001
## 14 physicality   534273. 0.0000000001
```

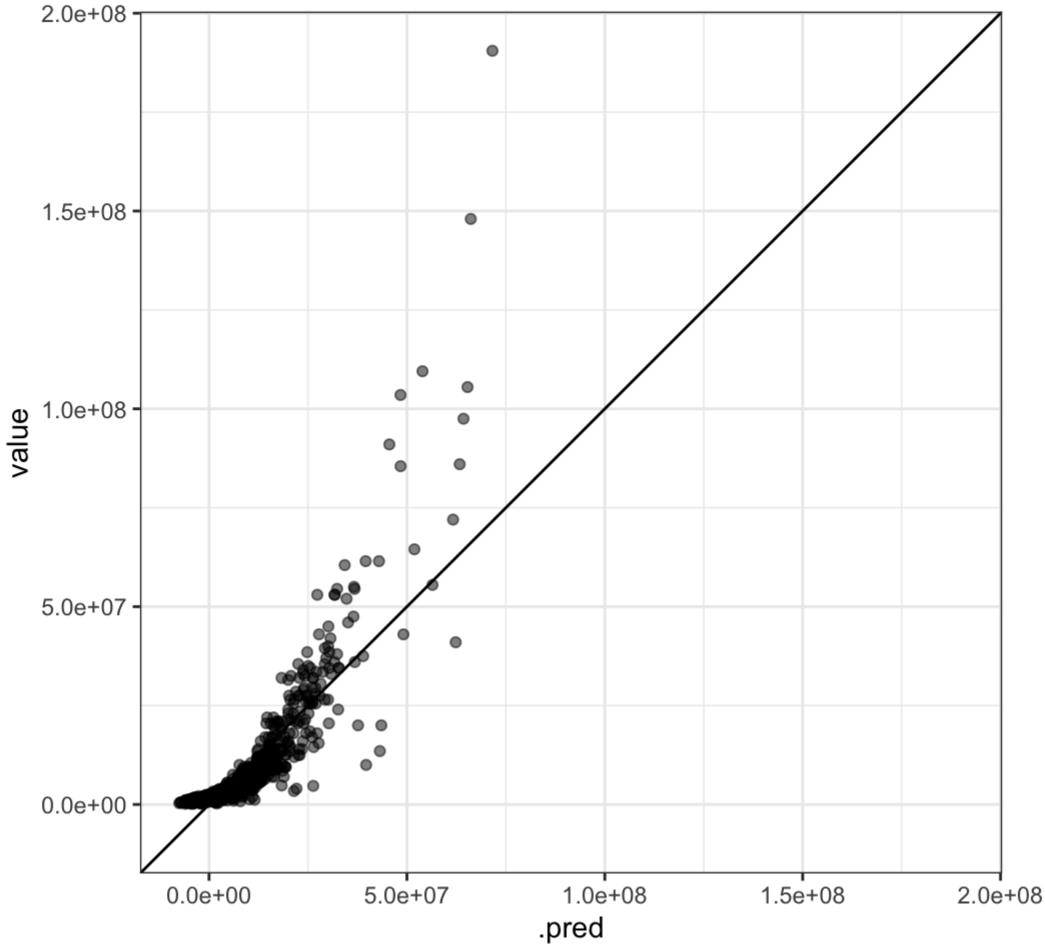
```
# create a table with 2 columns: predicted values and observed values from the testin
g set using best lambda
ridge.res <- predict(ridge.fit, new_data = data.test %>% dplyr::select(-value))
ridge.res <- bind_cols(ridge.res, data.test %>% dplyr::select(value))
ridge.res
```

```
## # A tibble: 1,034 × 2
##        .pred     value
##        <dbl>     <int>
##  1 71598825. 190500000
##  2 43162657.  13500000
##  3 62328284.  41000000
##  4 65319421. 105500000
##  5 63331750.  86000000
##  6 48399653.  85500000
##  7 61659201.  72000000
##  8 66139042. 148000000
##  9 48375132. 103500000
## 10 64289588.  97500000
## # … with 1,024 more rows
```

```
# checking the performance of the models using metric set of R-Squared, RMSE, and MAE
metrics(ridge.res, truth = value, estimate = .pred)
```

```
## # A tibble: 3 × 3
##   .metric .estimator   .estimate
##   <chr>   <chr>            <dbl>
## 1 rsq     standard         0.733
## 2 rmse    standard   7497189.
## 3 mae     standard   3839902.
```

```
# create a plot of observed values from testing set against predicted values
ridge.res %>% ggplot(aes(x = .pred, y = value)) + geom_point(alpha = 0.5) + geom_abli
ne(lty = 1) + theme_bw() + coord_obs_pred()
```



Analysis

With an R-Square of 73.26%, RMSE of 7497189, and MAE of 3839902, this model does not perform as well as desired on the testing set. This can be explained by the dataset's violation of the assumptions required to perform linear regression. Furthermore, the relationship between the response variable and the predictor variables itself is not linear which explains the ineffectiveness of the linear regression model to predict the market value of a player.

LASSO REGRESSION

Similar to ridge regression, lasso regression also shrinks the regression coefficients towards 0. The amount of shrinkage can be tuned by the parameter λ. When $\lambda = 0$, there is no shrinkage and lasso regression performs like linear regression. As λ increases, the impact of shrinkage also grows and regression coefficients get closer to 0. The difference between ridge and lasso regression is that the latter can force some coefficients to be exactly 0 to perform variable selection. Furthermore, lasso regression still performs well even with collinearity present in the predictor variables.

```
set.seed(28)

# Model
lasso.mod <- linear_reg(penalty = tune(), mixture = 1) %>% set_mode("regression") %>%
set_engine("glmnet")
# Workflow
lasso.wf <- workflow() %>% add_model(lasso.mod) %>% add_recipe(rec)
```

Hypertuning Paramters

Next, I performed cross-validation using 5 folds to choose the most optimal value for the tuning parameter λ. I wanted to select two values of λ, one that gives the highest R-Squared value and one that gives the lowest RMSE value to build two different lasso regression models. However, both R-Squared and RMSE are optimized at one value of $\lambda = 1e^{-10}$. Thus, I fit one lasso regression model using the optimal parameter value.

```
set.seed(28)

# Tuning lambda
lasso.pen.grid <- grid_regular(penalty(), levels = 10)
lasso.tune.res <- tune_grid(lasso.wf, resamples = data.fold, grid = lasso.pen.grid)
collect_metrics(lasso.tune.res)
```

```
## # A tibble: 20 × 7
##          penalty .metric .estimator        mean     n      std_err .config
##            <dbl> <chr>   <chr>            <dbl> <int>        <dbl> <chr>
##  1 0.0000000001  rmse    standard   5956499.        5 113205.       Preprocessor…
##  2 0.0000000001  rsq     standard         0.781     5       0.00976 Preprocessor…
##  3 0.00000000129 rmse    standard   5956499.        5 113205.       Preprocessor…
##  4 0.00000000129 rsq     standard         0.781     5       0.00976 Preprocessor…
##  5 0.0000000167  rmse    standard   5956499.        5 113205.       Preprocessor…
##  6 0.0000000167  rsq     standard         0.781     5       0.00976 Preprocessor…
##  7 0.000000215   rmse    standard   5956499.        5 113205.       Preprocessor…
##  8 0.000000215   rsq     standard         0.781     5       0.00976 Preprocessor…
##  9 0.00000278    rmse    standard   5956499.        5 113205.       Preprocessor…
## 10 0.00000278    rsq     standard         0.781     5       0.00976 Preprocessor…
## 11 0.0000359     rmse    standard   5956499.        5 113205.       Preprocessor…
## 12 0.0000359     rsq     standard         0.781     5       0.00976 Preprocessor…
## 13 0.000464      rmse    standard   5956499.        5 113205.       Preprocessor…
## 14 0.000464      rsq     standard         0.781     5       0.00976 Preprocessor…
## 15 0.00599       rmse    standard   5956499.        5 113205.       Preprocessor…
## 16 0.00599       rsq     standard         0.781     5       0.00976 Preprocessor…
## 17 0.0774        rmse    standard   5956499.        5 113205.       Preprocessor…
## 18 0.0774        rsq     standard         0.781     5       0.00976 Preprocessor…
## 19 1             rmse    standard   5956499.        5 113205.       Preprocessor…
## 20 1             rsq     standard         0.781     5       0.00976 Preprocessor…
```

```
# Selecting the best lambdas using two different metrics: R-Squared and Root Mean Squ
are Error
# best R-squared lambda
lasso.best.pen.rsq <- select_best(lasso.tune.res, metric = "rsq")
lasso.best.pen.rsq
```

```
## # A tibble: 1 × 2
##        penalty .config
##          <dbl> <chr>
## 1 0.0000000001 Preprocessor1_Model01
```

```
#best RMSE lambda
lasso.best.pen.rmse <- select_best(lasso.tune.res, metric = "rmse")
lasso.best.pen.rmse
```

```
## # A tibble: 1 × 2
##        penalty .config
##          <dbl> <chr>
## 1 0.0000000001 Preprocessor1_Model01
```

Fitting and Predicting

Using the metric set of R-Squared, RMSE, and MAE, I checked the performance of the model on the testing set. $R^2 = 0.7357347$, $RMSE = 7403193$, and $MAE = 3889372$. I then created a Predicted vs. Observed values plot to visualize the performance of the model.

```
set.seed(28)

# Final workflow
lasso.final.wf <- finalize_workflow(lasso.wf, lasso.best.pen.rsq)

# Fitting using best lambda
lasso.fit <- fit(lasso.final.wf, data = data.train)
lasso.fit %>% extract_fit_parsnip() %>% tidy()
```

```
## # A tibble: 14 × 3
##    term         estimate      penalty
##    <chr>           <dbl>        <dbl>
##  1 (Intercept)  8124948. 0.0000000001
##  2 overall      6287240. 0.0000000001
##  3 potential    1068211. 0.0000000001
##  4 age         -2081963. 0.0000000001
##  5 height       -415925. 0.0000000001
##  6 weight        -94333. 0.0000000001
##  7 wage         4507623. 0.0000000001
##  8 rep           345969. 0.0000000001
##  9 pace          196540. 0.0000000001
## 10 shoot         132700. 0.0000000001
## 11 pass          157509. 0.0000000001
## 12 dribble      -610010. 0.0000000001
## 13 defend       -588853. 0.0000000001
## 14 physicality   414094. 0.0000000001
```

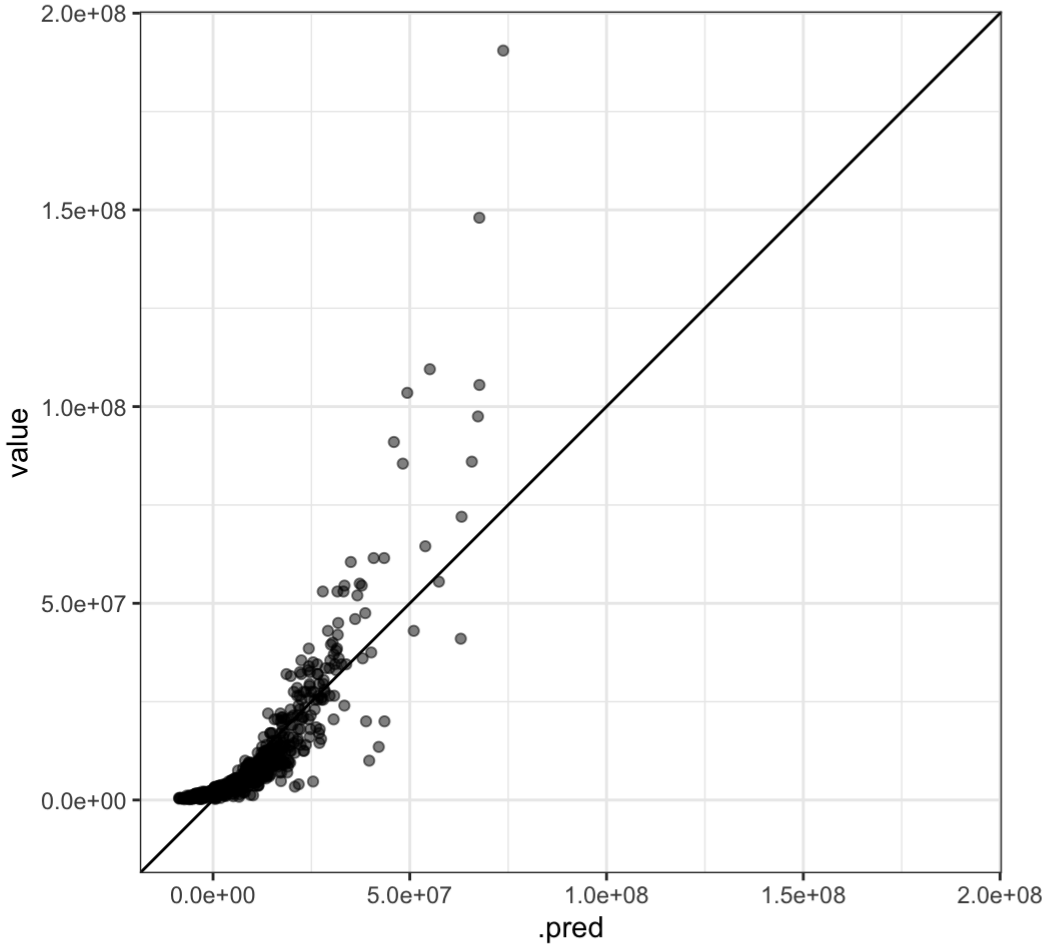
```
# create a table with 2 columns: predicted values and observed values from the testin
g set using best lambda
lasso.res <- predict(lasso.fit, new_data = data.test %>% dplyr::select(-value))
lasso.res <- bind_cols(lasso.res, data.test %>% dplyr::select(value))
lasso.res
```

```
## # A tibble: 1,034 × 2
##         .pred     value
##         <dbl>     <int>
##   1 73742951. 190500000
##   2 42117802.  13500000
##   3 62965299.  41000000
##   4 67696131. 105500000
##   5 65766626.  86000000
##   6 48231101.  85500000
##   7 63167734.  72000000
##   8 67676171. 148000000
##   9 49366439. 103500000
##  10 67329045.  97500000
## # … with 1,024 more rows
```

```
# checking the performance of the models using metric set of R-Squared, RMSE, and MAE
metrics(lasso.res, truth = value, estimate = .pred)
```

```
## # A tibble: 3 × 3
##   .metric .estimator   .estimate
##   <chr>   <chr>            <dbl>
## 1 rsq     standard         0.736
## 2 rmse    standard   7403193.
## 3 mae     standard   3889372.
```

```
# create a plot of observed values from testing set against predicted values
lasso.res %>% ggplot(aes(x = .pred, y = value)) + geom_point(alpha = 0.5) + geom_abli
ne(lty = 1) + theme_bw() + coord_obs_pred()
```



Analysis

With an R-Square of 73.57%, RMSE of 7403193, and MAE of 3889372, this model does not perform as well as desired on the testing set. Similar to the linear regression and ridge regression models, this can be explained by the dataset's violation of the assumptions required to perform linear regression. Furthermore, the relationship between the response variable and the predictor variables itself is not linear which explains the ineffectiveness of the linear regression model to predict the market value of a player.

K-NEAREST NEIGHBOR

K-Nearest Neighbor (or K-NN) is a nonparametric model that approximates the association between the predictor variables and the response variable by averaging the observations in the same area of proximity. The size of the area of proximity, or `neighbors`, can be tuned using cross-validation. As `neighbors` increases, more observations are included in the process of averaging and predicting the outcome.

```
set.seed(28)

# Model
knn.mod <- nearest_neighbor(neighbors = tune()) %>% set_engine("kknn") %>% set_mode
("regression")
# Workflow
knn.wf <- workflow() %>% add_model(knn.mod) %>% add_recipe(rec)
```

Hypertuning Paramters

Next, I performed cross-validation using 5 folds to choose the most optimal value for the tuning parameter `neighbors`. I wanted to select two values of `neighbors`, one that gives the highest R-Squared value and one that gives the lowest RMSE value to build two different k-nn models. However, both R-Squared and

RMSE are optimized at one value of neighbors = 10. Thus, I fit one k-nn model using the optimal parameter value.

```
set.seed(28)

# Tuning neighbors
knn.param.grid <- grid_regular(neighbors(), levels = 10)
knn.tune.res <- tune_grid(knn.wf, resamples = data.fold, grid = knn.param.grid)
collect_metrics(knn.tune.res)
```

```
## # A tibble: 20 × 7
##    neighbors .metric .estimator        mean     n      std_err .config
##        <int> <chr>   <chr>            <dbl> <int>        <dbl> <chr>
##  1         1 rmse    standard   5421148.        5  55832.      Preprocessor1_Mo…
##  2         1 rsq     standard         0.822     5      0.00442 Preprocessor1_Mo…
##  3         2 rmse    standard   4987920.        5 145616.      Preprocessor1_Mo…
##  4         2 rsq     standard         0.848     5      0.00881 Preprocessor1_Mo…
##  5         3 rmse    standard   4668584.        5 195015.      Preprocessor1_Mo…
##  6         3 rsq     standard         0.869     5      0.00999 Preprocessor1_Mo…
##  7         4 rmse    standard   4475161.        5 186670.      Preprocessor1_Mo…
##  8         4 rsq     standard         0.883     5      0.00873 Preprocessor1_Mo…
##  9         5 rmse    standard   4358012.        5 169691.      Preprocessor1_Mo…
## 10         5 rsq     standard         0.891     5      0.00740 Preprocessor1_Mo…
## 11         6 rmse    standard   4278245.        5 154846.      Preprocessor1_Mo…
## 12         6 rsq     standard         0.898     5      0.00641 Preprocessor1_Mo…
## 13         7 rmse    standard   4230212.        5 143848.      Preprocessor1_Mo…
## 14         7 rsq     standard         0.902     5      0.00577 Preprocessor1_Mo…
## 15         8 rmse    standard   4205094.        5 136312.      Preprocessor1_Mo…
## 16         8 rsq     standard         0.905     5      0.00533 Preprocessor1_Mo…
## 17         9 rmse    standard   4191979.        5 131390.      Preprocessor1_Mo…
## 18         9 rsq     standard         0.907     5      0.00503 Preprocessor1_Mo…
## 19        10 rmse    standard   4184221.        5 129307.      Preprocessor1_Mo…
## 20        10 rsq     standard         0.909     5      0.00487 Preprocessor1_Mo…
```

```
# Selecting the best neighbors using two different metrics: R-Squared and Root Mean S
quare Error
# best R-squared neighbors
knn.best.pen.rsq <- select_best(knn.tune.res, metric = "rsq")
knn.best.pen.rsq
```

```
## # A tibble: 1 × 2
##   neighbors .config
##       <int> <chr>
## 1        10 Preprocessor1_Model10
```

```
# best RMSE neighbors
knn.best.pen.rmse <- select_best(knn.tune.res, metric = "rmse")
knn.best.pen.rmse
```

```
## # A tibble: 1 × 2
##   neighbors .config
##       <int> <chr>
## 1        10 Preprocessor1_Model10
```

Fitting and Predicting

Using the metric set of R-Squared, RMSE, and MAE, I checked the performance of the model on the testing set. $R^2 = 0.8700372$, $RMSE = 5767945$, and $MAE = 1742924$. I then created a Predicted vs. Observed values plot to visualize the performance of the model.

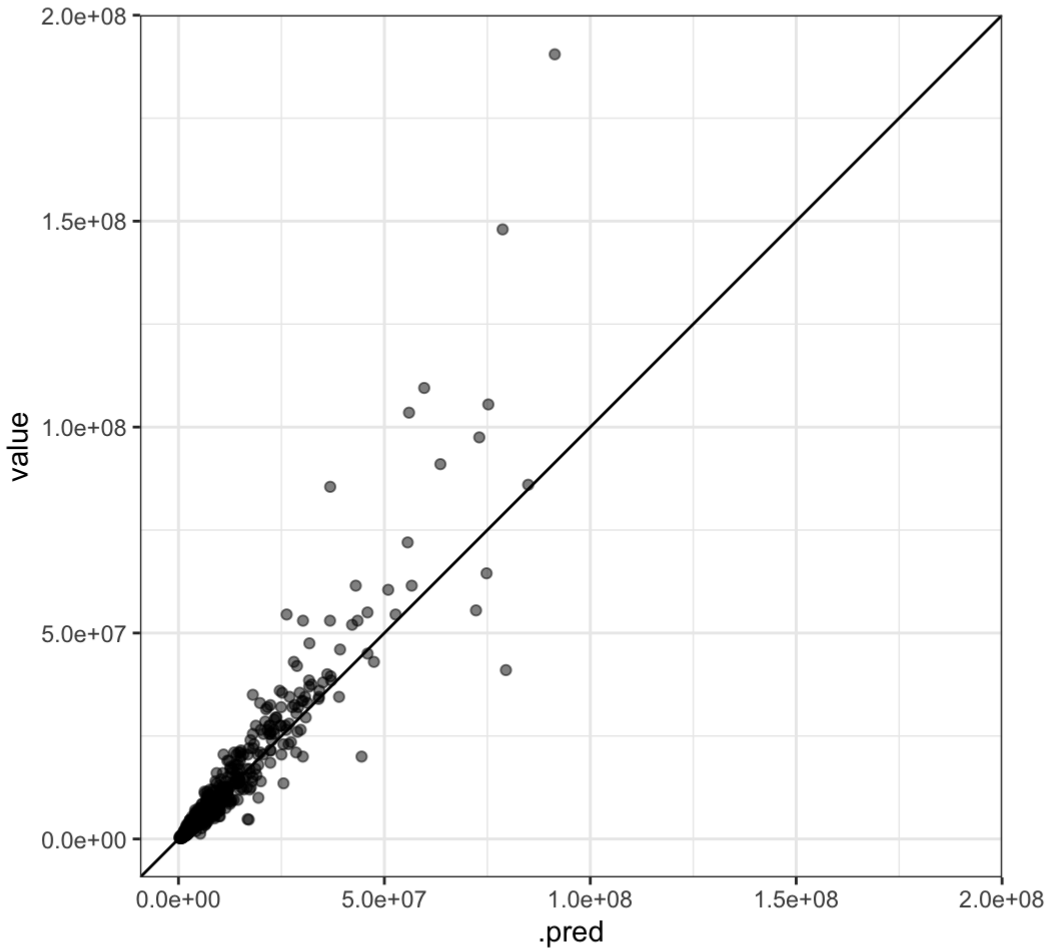
```
set.seed(28)

# Final workflow
knn.final.wf <- finalize_workflow(knn.wf, knn.best.pen.rsq)

# Fitting using best neighbors
knn.fit <- fit(knn.final.wf, data = data.train)

# create a table with 2 columns: predicted values and observed values from the testin
g set using best neighbors
knn.res <- predict(knn.fit, new_data = data.test %>% dplyr::select(-value))
knn.res <- bind_cols(knn.res, data.test %>% dplyr::select(value))
knn.res
```

```
## # A tibble: 1,034 × 2
##        .pred     value
##        <dbl>     <int>
##  1 91373320. 190500000
##  2 25499422.  13500000
##  3 79515797.  41000000
##  4 75240434. 105500000
##  5 84922153.  86000000
##  6 36844870.  85500000
##  7 55653187.  72000000
##  8 78714559. 148000000
##  9 55987619. 103500000
## 10 73047958.  97500000
## # … with 1,024 more rows
```

```
# checking the performance of the models using metric set of R-Squared, RMSE, and MAE
metrics(knn.res, truth = value, estimate = .pred)
```

```
## # A tibble: 3 × 3
##   .metric .estimator   .estimate
##   <chr>   <chr>            <dbl>
## 1 rsq     standard         0.870
## 2 rmse    standard   5767945.
## 3 mae     standard   1742924.
```

```
# create a plot of observed values from testing set against predicted values
knn.res %>% ggplot(aes(x = .pred, y = value)) + geom_point(alpha = 0.5) + geom_abline
(lty = 1) + theme_bw() + coord_obs_pred()
```



Analysis

With an R-Square of 87.00%, RMSE of 5767945, and MAE of 1742924, this model performs fairly well on the testing set. There is significant improvement in comparison to the linear models. This is because the K-NN model is not a linear model and is more effective for datasets in which the relationship between the response variable and the predictor variables is not linear.

SINGLE DECISION TREE

```
set.seed(28)

# model
tree.mod <- decision_tree(cost_complexity = tune()) %>% set_mode("regression") %>% se
t_engine("rpart")
# workflow
tree.wf <- workflow() %>% add_model(tree.mod) %>% add_recipe(rec)
```

Hypertuning Parameters

Next, I performed cross-validation using 5 folds to choose the most optimal value for parameter `complexity`. I wanted to select two values of `complexity`, one that gives the highest R-Squared value and one that gives the lowest RMSE value to build two different single decision trees. However, both R-Squared and RMSE are optimized at one value of complexity = 1e-10. Thus, I fit one single decision tree model using the optimal parameter value.

```
set.seed(28)

# tuning parameter cost_complexity
tree.param.grid <- grid_regular(cost_complexity(), levels = 10)
tree.tune.res <- tune_grid(tree.wf, resamples = data.fold, grid = tree.param.grid)
collect_metrics(tree.tune.res)
```

```
## # A tibble: 20 × 7
##    cost_complexity .metric .estimator        mean     n     std_err .config
##              <dbl> <chr>   <chr>            <dbl> <int>       <dbl> <chr>
##  1    0.0000000001 rmse    standard   2878895.        5 334712.     Preprocesso…
##  2    0.0000000001 rsq     standard         0.947     5      0.0117 Preprocesso…
##  3    0.000000001  rmse    standard   2878895.        5 334712.     Preprocesso…
##  4    0.000000001  rsq     standard         0.947     5      0.0117 Preprocesso…
##  5    0.00000001   rmse    standard   2878896.        5 334712.     Preprocesso…
##  6    0.00000001   rsq     standard         0.947     5      0.0117 Preprocesso…
##  7    0.0000001    rmse    standard   2878904.        5 334719.     Preprocesso…
##  8    0.0000001    rsq     standard         0.947     5      0.0117 Preprocesso…
##  9    0.000001     rmse    standard   2879275.        5 334642.     Preprocesso…
## 10    0.000001     rsq     standard         0.947     5      0.0117 Preprocesso…
## 11    0.00001      rmse    standard   2884106.        5 334436.     Preprocesso…
## 12    0.00001      rsq     standard         0.946     5      0.0117 Preprocesso…
## 13    0.0001       rmse    standard   2928209.        5 329860.     Preprocesso…
## 14    0.0001       rsq     standard         0.945     5      0.0117 Preprocesso…
## 15    0.001        rmse    standard   3143928.        5 299462.     Preprocesso…
## 16    0.001        rsq     standard         0.937     5      0.0115 Preprocesso…
## 17    0.01         rmse    standard   4445996.        5 184671.     Preprocesso…
## 18    0.01         rsq     standard         0.878     5      0.0114 Preprocesso…
## 19    0.1          rmse    standard   5920315.        5 113304.     Preprocesso…
## 20    0.1          rsq     standard         0.784     5      0.0101 Preprocesso…
```

```
# Selecting the best parameter values using two different metrics: R-Squared and Root
Mean Square Error
# best R-squared parameter value
tree.best.complex.rsq <- select_best(tree.tune.res, metric = "rsq")
tree.best.complex.rsq
```

```
## # A tibble: 1 × 2
##   cost_complexity .config
##             <dbl> <chr>
## 1    0.0000000001 Preprocessor1_Model01
```

```
# best RMSE parameter value
tree.best.complex.rmse <- select_best(tree.tune.res, metric = "rmse")
tree.best.complex.rmse
```

```
## # A tibble: 1 × 2
##   cost_complexity .config
##             <dbl> <chr>
## 1    0.0000000001 Preprocessor1_Model01
```

Fitting and Predicting

Using the metric set of R-Squared, RMSE, and MAE, I checked the performance of the model on the testing set. $R^2 = 0.8692827$, $RMSE = 5161271$, and $MAE = 1136096$. I then created a Predicted vs. Observed values plot to visualize the performance of the model.

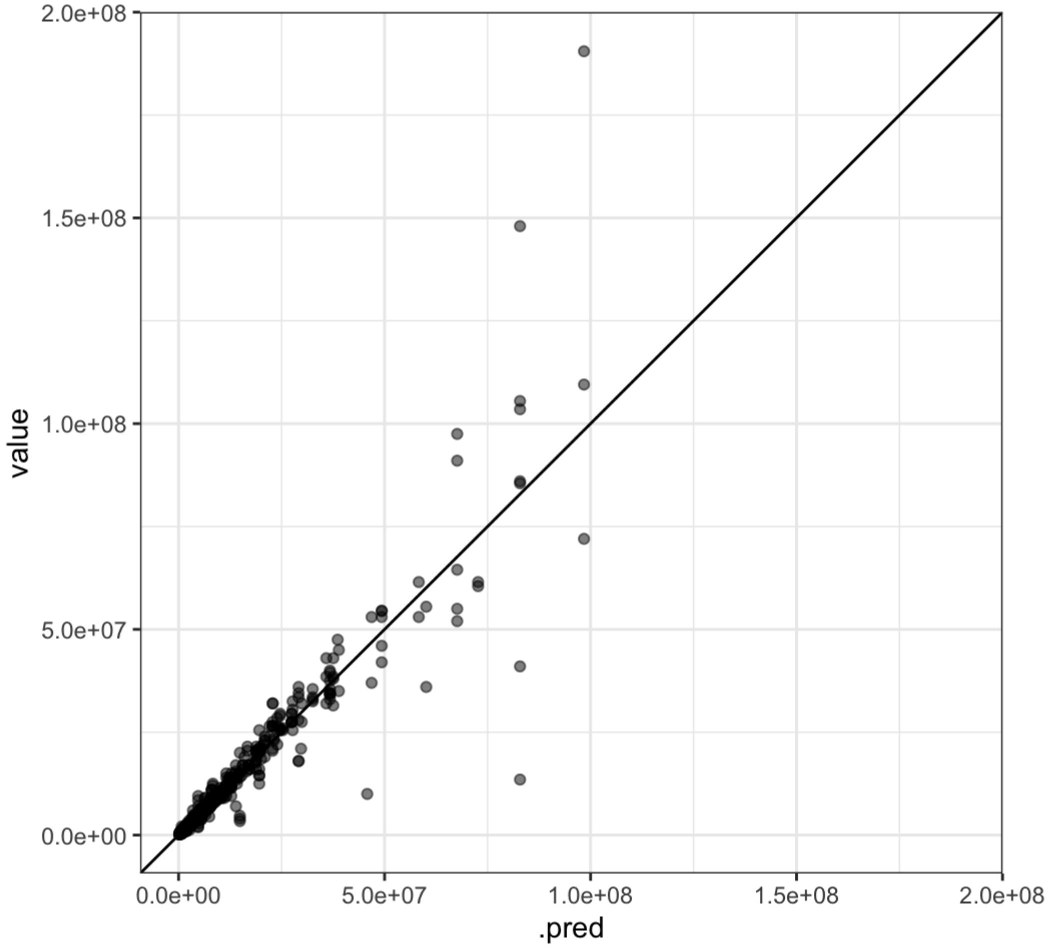
```
set.seed(28)

# final workflow
tree.final.wf <- finalize_workflow(tree.wf, tree.best.complex.rsq)

# fit a model using best complexity
tree.fit <- fit(tree.final.wf, data = data.train)

# create a table with 2 columns: predicted values and observed values from the testin
g set
tree.res <- predict(tree.fit, new_data = data.test %>% dplyr::select(-value))
tree.res <- bind_cols(tree.res, data.test %>% dplyr::select(value))
tree.res
```

```
## # A tibble: 1,034 × 2
##       .pred     value
##       <dbl>     <int>
##  1 98406250 190500000
##  2 82875000  13500000
##  3 82875000  41000000
##  4 82875000 105500000
##  5 82875000  86000000
##  6 82875000  85500000
##  7 98406250  72000000
##  8 82875000 148000000
##  9 82875000 103500000
## 10 67625000  97500000
## # … with 1,024 more rows
```

```
# checking the performance of the model using metric set of R-Squared, RMSE, and MAE
metrics(tree.res, truth = value, estimate = .pred)
```

```
## # A tibble: 3 × 3
##   .metric .estimator   .estimate
##   <chr>   <chr>            <dbl>
## 1 rsq     standard         0.869
## 2 rmse    standard   5161271.
## 3 mae     standard   1136096.
```

```
# create a plot of observed values from testing set against predicted values
tree.res %>% ggplot(aes(x = .pred, y = value)) + geom_point(alpha = 0.5) + geom_ablin
e(lty = 1) + theme_bw() + coord_obs_pred()
```



Analysis

With an R-Squared of 86.93%, RMSE of 5161271, and MAE of 1136096, the model performs fairly well in predicting the testing data. There is significant improvement in comparison to the linear models. Furthermore, the model performs as well as the K-NN model. Similar to the K-NN model, the single decision tree is not bound by the assumption that the relationship between the response and predictor variables must be linear, making it more flexible in predicting the market value of a player.

RANDOM FOREST

```
set.seed(28)

# model
rf.mod <- rand_forest(mtry = tune(), trees = tune()) %>% set_mode("regression") %>% s
et_engine("randomForest", importance = TRUE)
# workflow
rf.wf <- workflow() %>% add_model(rf.mod) %>% add_recipe(rec)
```

Hypertuning Parameters

Next, I performed cross-validation using 5 folds to choose the most optimal value for the parameters `mtry` and `trees` . I set the range of `mtry` to be between 1 and 13 because it has to be within the range of the number of predictor variables. I wanted to select two values of for each parameter, one that gives the highest R-Squared value and one that gives the lowest RMSE value to build two different random forest models. However, both R-Squared and RMSE are optimized at mtry = 13 and trees = 125. Thus, I fit one random forest model using the optimal parameter value.

```
set.seed(28)

# tuning parameter mtry and trees
rf.param.grid <- grid_regular(mtry(range = c(1,13)), trees(range = c(50, 200)), level
s = 5)
rf.tune.res <- tune_grid(rf.wf, resamples = data.fold, grid = rf.param.grid)
collect_metrics(rf.tune.res)
```

```
## # A tibble: 50 × 8
##     mtry trees .metric .estimator        mean     n      std_err .config
##    <int> <int> <chr>   <chr>            <dbl> <int>        <dbl> <chr>
##  1     1    50 rmse    standard   4124870.        5  83326.      Preprocessor1_…
##  2     1    50 rsq     standard         0.920     5      0.00523 Preprocessor1_…
##  3     4    50 rmse    standard   2620759.        5 152303.      Preprocessor1_…
##  4     4    50 rsq     standard         0.960     5      0.00573 Preprocessor1_…
##  5     7    50 rmse    standard   2168701.        5 183498.      Preprocessor1_…
##  6     7    50 rsq     standard         0.971     5      0.00517 Preprocessor1_…
##  7    10    50 rmse    standard   2114199.        5 194679.      Preprocessor1_…
##  8    10    50 rsq     standard         0.972     5      0.00548 Preprocessor1_…
##  9    13    50 rmse    standard   2064622.        5 214780.      Preprocessor1_…
## 10    13    50 rsq     standard         0.973     5      0.00594 Preprocessor1_…
## # … with 40 more rows
```

```
# Selecting the best parameter value using two different metrics: R-Squared and Root
Mean Square Error
# best R-squared parameter values
rf.best.rsq <- select_best(rf.tune.res, metric = "rsq")
rf.best.rsq
```

```
## # A tibble: 1 × 3
##    mtry trees .config
##   <int> <int> <chr>
## 1    13   125 Preprocessor1_Model15
```

```
# best R-squared parameter values
rf.best.rmse <- select_best(rf.tune.res, metric = "rmse")
rf.best.rmse
```

```
## # A tibble: 1 × 3
##    mtry trees .config
##   <int> <int> <chr>
## 1    13   125 Preprocessor1_Model15
```

Fitting and Predicting

Using the metric set of R-Squared, RMSE, and MAE, I checked the performance of the model on the testing set. $R^2 = 0.9120025$, $RMSE = 4336237$, and $MAE = 813451.1$. I then created a Predicted vs. Observed values plot to visualize the performance of the model.

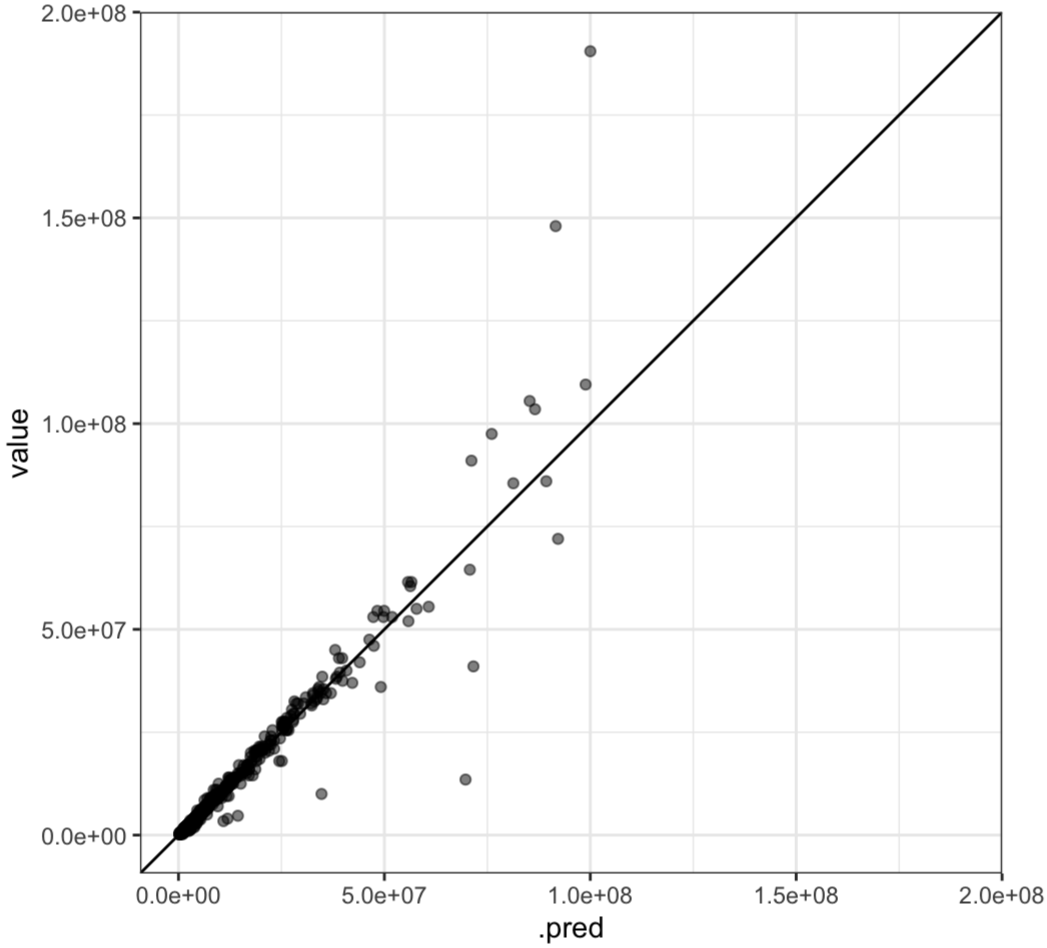
```
set.seed(28)

# Final workflow
rf.final.wf <- finalize_workflow(rf.wf, rf.best.rsq)

# fit a model using best parameter values
rf.fit <- fit(rf.final.wf, data = data.train)

# create a table with 2 columns: predicted values and observed values from the testin
g set using best parameter values
rf.res <- predict(rf.fit, new_data = data.test %>% dplyr::select(-value))
rf.res <- bind_cols(rf.res, data.test %>% dplyr::select(value))
rf.res
```

```
## # A tibble: 1,034 × 2
##         .pred     value
##         <dbl>     <int>
##  1 100013733. 190500000
##  2  69707800   13500000
##  3  71620467.  41000000
##  4  85279733. 105500000
##  5  89309200   86000000
##  6  81306867.  85500000
##  7  92170733.  72000000
##  8  91591267. 148000000
##  9  86577133. 103500000
## 10  76058467.  97500000
## # … with 1,024 more rows
```

```
# checking the performance of the models using metric set of R-Squared, RMSE, and MAE
metrics(rf.res, truth = value, estimate = .pred)
```

```
## # A tibble: 3 × 3
##   .metric .estimator   .estimate
##   <chr>   <chr>            <dbl>
## 1 rsq     standard         0.912
## 2 rmse    standard   4336237.
## 3 mae     standard    813451.
```

```
# create a plot of observed values from testing set against predicted values
rf.res %>% ggplot(aes(x = .pred, y = value)) + geom_point(alpha = 0.5) + geom_abline
(lty = 1) + theme_bw() + coord_obs_pred()
```



Analysis

With an R-Squared of 91.20%, RMSE of 4336237, and MAE of 813451.1, this model performs well in predicting the testing data. There is significant improvement in comparison to the linear models and adequate improvement in comparison to the K-NN model and the single decision tree model. The improvement from a single decision tree can be attributed to the fact that a random forest is a collection of decision trees with aggregated results.

BOOSTED TREE

```
# model
boost.mod <- boost_tree(mtry = tune(), trees = tune(), learn_rate = tune()) %>% set_m
ode("regression") %>% set_engine("xgboost")
# workflow
boost.wf <- workflow() %>% add_model(boost.mod) %>% add_recipe(rec)
```

Hypertuning Parameters

Next, I performed cross-validation using 5 folds to choose the most optimal values for parameters `trees` , `mtry` , and `learn_rate` . I wanted to select two values for each parameter, one that gives the highest R-Squared value and one that gives the lowest RMSE value to build two different boosted tree models. However, both R-Squared and RMSE are optimized at mtry = 10, trees = 1000, and learn_rate = 0.1. Thus, I fit one boosted tree model using the optimal parameter values.

```
set.seed(28)

# tuning parameters trees, tree_depth, and learn_rate
boost.param.grid <- grid_regular(mtry(range = c(1,13)), trees(), learn_rate(), levels
= 5)
boost.tune.res <- tune_grid(boost.wf, resamples = data.fold, grid = boost.param.grid)
collect_metrics(boost.tune.res)
```

```
## # A tibble: 250 × 9
##     mtry trees   learn_rate .metric .estimator        mean     n std_err .config
##    <int> <int>        <dbl> <chr>   <chr>            <dbl> <int>   <dbl> <chr>
##  1     1     1 0.0000000001 rmse    standard       1.51e+7     5 1.06e+5 Prepro…
##  2     1     1 0.0000000001 rsq     standard       6.25e-1     5 4.87e-2 Prepro…
##  3     1   500 0.0000000001 rmse    standard       1.51e+7     5 1.06e+5 Prepro…
##  4     1   500 0.0000000001 rsq     standard       8.94e-1     5 5.28e-3 Prepro…
##  5     1  1000 0.0000000001 rmse    standard       1.51e+7     5 1.06e+5 Prepro…
##  6     1  1000 0.0000000001 rsq     standard       8.95e-1     5 6.04e-3 Prepro…
##  7     1  1500 0.0000000001 rmse    standard       1.51e+7     5 1.06e+5 Prepro…
##  8     1  1500 0.0000000001 rsq     standard       8.95e-1     5 6.73e-3 Prepro…
##  9     1  2000 0.0000000001 rmse    standard       1.51e+7     5 1.06e+5 Prepro…
## 10     1  2000 0.0000000001 rsq     standard       8.94e-1     5 6.67e-3 Prepro…
## # … with 240 more rows
```

```
# Selecting the best parameter value using two different metrics: R-Squared and Root
Mean Square Error
# best R-squared parameter values
boost.best.rsq <- select_best(boost.tune.res, metric = "rsq")
boost.best.rsq
```

```
## # A tibble: 1 × 4
##    mtry trees learn_rate .config
##   <int> <int>      <dbl> <chr>
## 1    10  1000        0.1 Preprocessor1_Model098
```

```
# best RMSE parameter values
boost.best.rmse <- select_best(boost.tune.res, metric = "rmse")
boost.best.rmse
```

```
## # A tibble: 1 × 4
##    mtry trees learn_rate .config
##   <int> <int>      <dbl> <chr>
## 1    10  1000        0.1 Preprocessor1_Model098
```

Fitting and Predicting

Using the metric set of R-Squared, RMSE, and MAE, I checked the performance of the model on the testing set. $R^2 = 0.9385811$, $RMSE = 3739774$, and $MAE = 641803.4$. I then created a Predicted vs. Observed values plot to visualize the performance of the model.

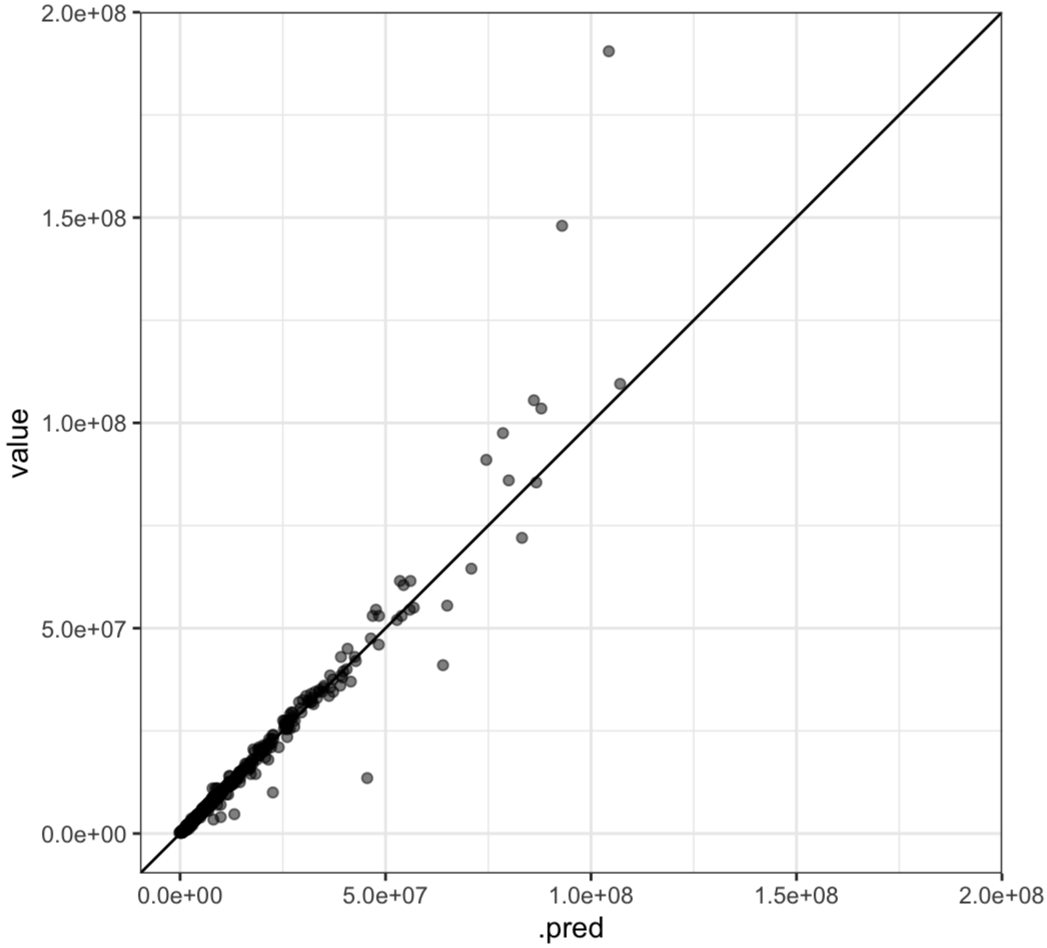
```
set.seed(28)

# final workflow
boost.final.wf <- finalize_workflow(boost.wf, boost.best.rsq)

# fitting a model using the best parameter values
boost.fit <- fit(boost.final.wf, data = data.train)

# create a table with 2 columns: predicted values and observed values from the testin
g set
boost.res <- predict(boost.fit, new_data = data.test %>% dplyr::select(-value))
boost.res <- bind_cols(boost.res, data.test %>% dplyr::select(value))
boost.res
```

```
## # A tibble: 1,034 × 2
##        .pred     value
##        <dbl>     <int>
##  1 104332264 190500000
##  2  45535544  13500000
##  3  63977936  41000000
##  4  86100328 105500000
##  5  79979336  86000000
##  6  86676416  85500000
##  7  83198320  72000000
##  8  92951928 148000000
##  9  87904104 103500000
## 10  78562912  97500000
## # … with 1,024 more rows
```

```
# checking the performance of the model using a metric set of R-squared, RMSE, and MA
E
metrics(boost.res, truth = value, estimate = .pred)
```

```
## # A tibble: 3 × 3
##   .metric .estimator   .estimate
##   <chr>   <chr>            <dbl>
## 1 rsq     standard         0.939
## 2 rmse    standard   3739774.
## 3 mae     standard    641803.
```

```
# create a plot of observed values from testing set against predicted values
boost.res %>% ggplot(aes(x = .pred, y = value)) + geom_point(alpha = 0.5) + geom_abli
ne(lty = 1) + theme_bw() + coord_obs_pred()
```



Analysis

With an R-Squared of 93.86%, RMSE of 3739774, and MAE of 641803.4, this model performs very well in predicting the testing data. There is significant improvement in comparison to the linear models, adequate improvement in comparison to the single decision tree model and the k-nn model, and slight improvement in comparison to the random forest model. As visualized by the plot of observed values against predicted values, the data points closely follow the $y = x$ line which means the model performs well in predicting the testing data.

BEST MODEL ANALYSIS

The best model is the boosted tree model with mtry = 10, trees = 1000, and learn_rate = 0.1. When used to predict the testing set, this model performs extremely well. The R-Squared value is 93.86%, the RMSE value is 3739774, and the MAE value is 641803.4 and they are significantly better than those of the other models besides the random forest model whose R-Squared is 91.20%, RMSE is 4336237, and MAE is 813451.1.

In comparison to the linear regression model, there is a 27.57% increase in R-Squared, 49.47% decrease in RMSE, and 75.18% decrease in MAE.

In comparison to the k-nearest neighbors model, there is a 7.88% increase in R-Squared, 35.16% decrease in RMSE, and 63.18% decrease in MAE.

In comparison to the single decision tree model, there is a 7.97% increase in R-Squared, 27.54% decrease in RMSE, and 43.51% decrease in MAE.

As seen in the statistics above, the boosted tree model performs extremely well in comparison to other models used in this project to predict a soccer player's market value given the chosen variables.

CONCLUSION

Using data provided by FIFA 23, I built 7 models to predict a soccer player's market value based on 13 predictor variables including his overall rating, potential rating, age, height, weight, wage, international reputation, pace rating, shooting rating, passing rating, dribbling rating, defending rating, and physicality rating. Before fitting the linear models, I performed diagnostics since there are certain assumptions required to fit these linear models. Evidently, the dataset fails to satisfy these assumptions. As seen above, the relationship between the market value of a player and the given predictors is not linear. Thus, as expected, the linear models (linear regression, ridge regression, and lasso regression) did not perform well in predicting the response variable using the testing set.

However, non-linear models performed fairly well on the testing set because they are not bound by the assumptions of the linear models, making them more flexible. The R-Squared values of K-Nearest Neighbor model and the single decision tree model both exceeded 85%. However, these two models still fell short of my expectations as I aimed to exceed 90% for the R-Squared value.

As expected, the random forest model and the boosted tree model were the best in predicting the response variable using the testing set as both exceeded 90% for their R-Squared values.