23 March 2023

Faculty of Undergraduate Studies
University of California, Davis
Davis, CA 95616

Professor Bo (Yu-Chien) Ning
Statistics Department
University of California, Davis
Davis, CA 95616

Dear Professor Ning,

We wish to submit an original research article entitled “Predicting NBA Player Net Ratings and Draft Positions.” We confirm that this work is original and has not been published elsewhere, nor is it currently under consideration for publication elsewhere.

In this paper, we report on a statistical investigation into the relationship between National Basketball Association (NBA) player performance metrics, their draft position, and net ratings. This is significant because the NBA is the world’s most highly paid professional sports league and it is quintessential for players, fans, and front offices to be familiar with the metrics that best predict great performance in the NBA.

We believe that this manuscript is appropriate for submission for the Statistics 141C: Big Data & High-Performance Statistical Computing course at the University of California, Davis because we have combined several statistical computing methods introduced by the course with our knowledge of data analysis methods and NBA basketball analytics. We have implemented the Cholesky method to optimize our code in the project.

We have created two strong models that predict Net Rating and Draft Position of a NBA player respectively. Our Net Rating Model has a usable Python function and our Draft Position model has a Shiny web application that can be accessed for easy use of our model. We hope to hear your feedback for our models and project.

Sincerely,

Raghuram Palaniappan
Jeff Lee
Koby Lieu
Jasper Liu

Predicting NBA Player Net Ratings and Draft Positions

Group 19 STA 141C
Raghuram Palaniappan
Jeff Lee
Koby Lieu
Jasper Liu
https://github.com/jefflee702/NBA

1 Introduction

The National Basketball Association (NBA) is the top paying sports league in the world [1]. The players on an active NBA roster receive the highest average salary for players on active rosters, and as such, team investments in their players are more monumental than ever for the short-term and long-term future of the basketball franchise. Whether players are acquired through free agency, trade, or draft, it is increasingly important for franchises to evaluate their players and prospects accurately so that teams can build the strongest roster possible to win the coveted NBA Championship. One such method of quantifying a player's contributions on the court is net rating, which is an all-encompassing number that includes both offensive and defensive production to separate the star players from the average players.

We investigated about 33 years worth of NBA draft and performance data to find statistical models that best predict top NBA performance through net rating and draft round. While not perfect complements to one another, we'd expect that players with higher projected draft ratings would have been selected earlier in the draft, as teams want the best players they can get with their highest picks.

Our goal is to model net rating and draft positioning, so NBA owners and industry members could predict these important factors.

2 Data and Methodology

2.1 Data

We utilized several datasets from Kaggle to answer our questions of interest. Our first dataset ranged from 2004 to 2020, containing individual NBA player biometrics (height and weight), statistics and performance in games on the season, including net rating. Our second data set ranged from 1996 to 2021, containing similar individual player biometrics, statistics, and draft round. We used each data set to investigate net rating and draft round respectively.

For our investigation, removal of undrafted players was necessary for the investigation pertaining to the player draft round. We also removed undrafted players from all other datasets for simplicity. By removing the 'undrafted' value, we were able to focus solely on those with a draft round and position. The draft year was limited to anything after 1989 since that was when the NBA reduced their draft rounds to two and draft positions to sixty. As a result, we removed data that had more than two draft rounds and moreo than sixty draft positions. In addition, we decided to remove several significant outliers from the dataset; many of these were the very top and very bottom percentage of performers in each category, which would skew the results of the investigation and were not representative of the sample.

2.2 Methodology

For the net rating prediction model, we employed various methodologies to develop and deploy a machine learning model. Initially, we implemented six different types of regression models, including Linear Regression, Random Forest, Support Vector, Gradient Boosting, Lasso, and Ridge. We then evaluated the performance of each model and identified the Random Forest and Gradient Boosting Regressions as the most effective models for our data.

To further optimize our model, we utilized hyperparameter tuning through GridSearchCV, a method that allowed us to define a grid of hyperparameters to search over and evaluate the best performing combination of hyperparameters for each model. This iterative approach helped us identify the optimal configuration for each model, resulting in improved model performance.

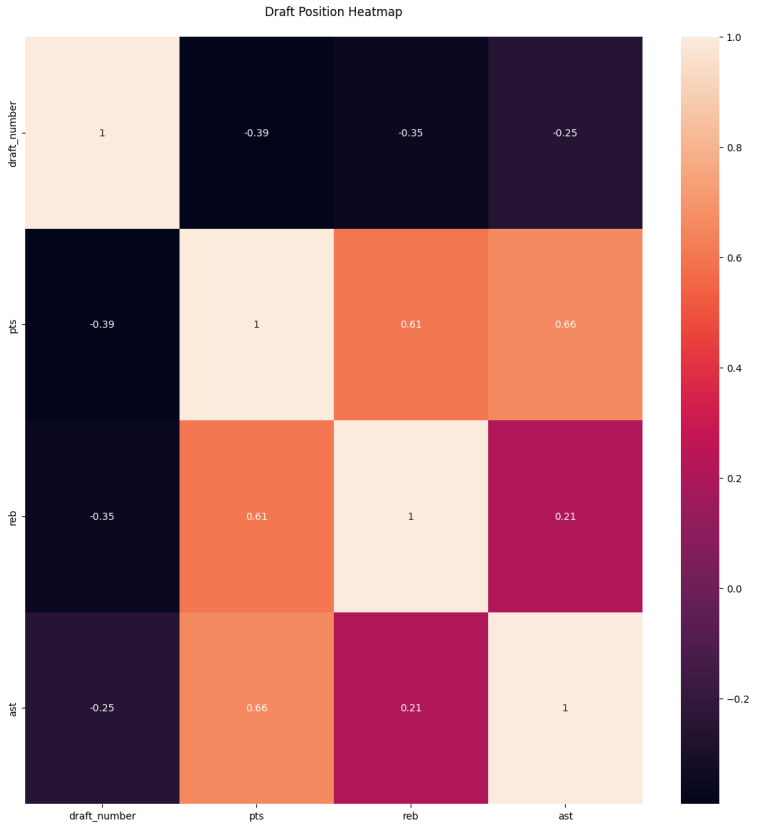
Finally, we utilized the Flask web framework to deploy our machine learning model online.

For our 2nd model, we created a draft position prediction model using multiple linear regression models. Our objective was to forecast a player's draft position by analyzing their performance metrics. Additionally, we attempted to predict player points, rebounds, and assists based on their draft position and performance metrics. To construct the model, we selected several significant variables from the dataset and used them to make predictions for each draft position from first overall to sixtieth overall. Ultimately, our model allowed us to estimate the average points, rebounds, and assists for each position in the draft.

3 Data Visualization

To get an understanding of our NBA data, we created a heatmap and pairplot.

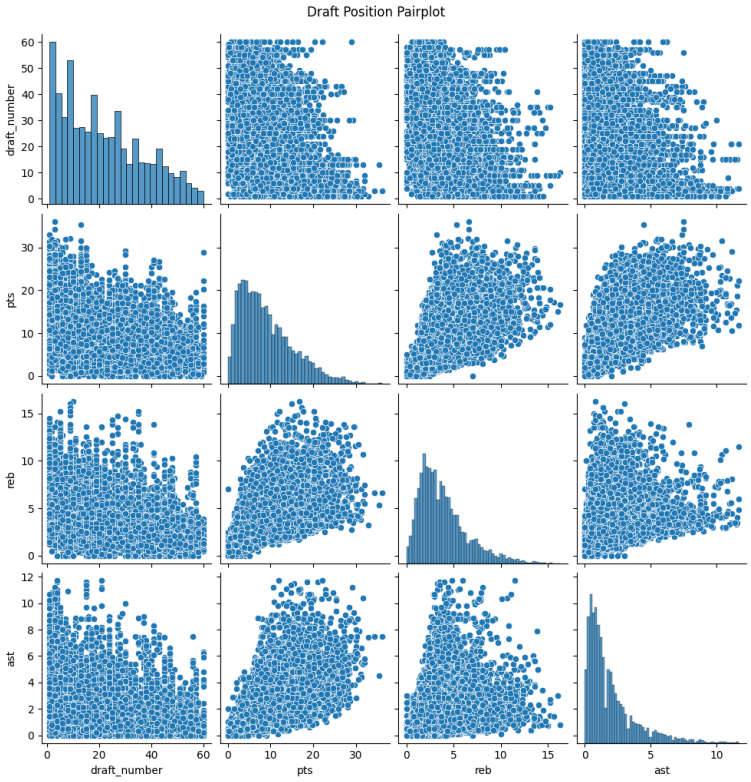
3.1 Draft Position Heatmap



Although the performance of players decreased as the draft position increased, we can see that the variables that determined the performance of players had a positive correlation with each other.

Assists and points seemed to have the highest correlation while assists and rebounds seemed to have the lowest.

3.2 Pairplot



The pairplot above shows that the points, rebounds, and assists trend downwards as the draft position increases. The frequency of these variables also tended to decrease as their value rose.

4 Results

We decided to set up 2 models from our data. For these models we developed a function and an interactive online app.

4.1 Net Rating Prediction Model

The first model is the Net Rating Prediction model which predicts the Net Rating of a player given certain inputs. These inputs are age, player height and weight, points, rebounds, assists, offensive and defensive rebound percentage, usage percentage, true shooting percentage, and assist percentage.

4.1.1 Regression Models

After preparing the data by scaling and generating visualizations, we proceeded to build six regression models: Linear Regression, Random Forest, Support Vector, Gradient Boosting, Lasso, and Ridge. However, our initial attempts yielded suboptimal results, prompting us to explore further optimization methods. Initially, we attempted to remove outliers by applying numpy Z-score values and removing values that exceeded three standard deviations. However, this worsened our model, likely due to the numerous outliers present in the NBA. Superstars such as Steph Curry, Lebron James, and Giannis are examples of outliers that cannot be compared to role players in the league, highlighting the importance of retaining all data points. We also attempted data transformation, but it failed to improve our model significantly. Ultimately, our most successful models were Random Forest Regression

and Gradient Boosting Regression. These tree-based models are well-suited for modeling non-linear relationships and handling robust outliers, aligning with our findings.

4.1.2 Hyperparameter Tuning

To further improve our models, we utilized Hyperparameter tuning through GridSearchCV, a method that allows us to define a grid of hyperparameters to search over and evaluate the best performing combination of hyperparameters for each model. After conducting GridSearchCV, our best-performing model was the Random Forest Regression with hyperparameter tuning, yielding an RSquared value of 0.73. Our methodology involved iteratively testing various combinations of hyperparameters to identify the optimal configuration that maximized our model's performance. This approach allowed us to fine-tune our models and generate more accurate results.

4.1.3 Python Model Function

We created a Python function to make predictions. This function allowed us to predict net rating using our optimized Random Forest model with Hyperparameter tuning.

```
# Random Forest Model Function
def predict_net_rating(age, player_height, player_weight, pts, reb, ast, oreb_pct, dreb_pct, usg_pct, ts_pct, ast_pct):
    # Load the data
    X = players_df[["age", "player_height", "player_weight", "pts", "reb", "ast",
                    "oreb_pct", "dreb_pct", "usg_pct", "ts_pct", "ast_pct"]]
    y = players_df["net_rating"]

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Scale and normalize the data
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Set up the parameter grid to search over
    param_grid = {'n_estimators': [100, 200, 500],
                  'max_depth': [None, 10, 20],
                  'max_features': ['auto', 'sqrt']}

    # Instantiate a random forest regression model
    rf_model = RandomForestRegressor(random_state=42)

    # Create a GridSearchCV object to find the best hyperparameters
    grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5)

    # Fit the GridSearchCV object to the training data
    grid_search.fit(X_train, y_train)

    # Make a prediction using the trained model
    input_data = [[age, player_height, player_weight, pts, reb, ast, oreb_pct, dreb_pct, usg_pct, ts_pct, ast_pct]]
    input_data_scaled = scaler.transform(input_data)
    net_rating = grid_search.predict(input_data_scaled)[0]

    return net_rating
```

4.2 Draft Position Prediction Model

Our 2nd model was the Draft Position Prediction model. The inputs for this model are age, player height and weight, points, rebounds, assists, offensive and defensive rebound percentage, usage percentage, true shooting percentage, assist percentage, and net rating.

4.2.1 Cholesky

One of the questions that we had was whether the draft position of a player correlated with their performance. We had players' performances determined by their average points, rebounds, and assists. To determine the correlation between a player's draft position and performance, we used the LU, Cholesky, and QR matrix factorization techniques to solve for the unknown vector. While all three techniques were relatively quick since our matrix was not too large, Cholesky turned out to be the fastest technique. This is because Cholesky is the most efficient option of the three. Our matrix is also symmetric and positive definite, so we are able to utilize this method the best. This provided the vector: [-0.6204988, -1.1340337, -0.3726529] for our three variables. This came with a computational runtime of .03 seconds when the matrix factorization was replicated 100 times.

4.2.2 Multiple Linear Regression

In the three separate multiple linear regression models that predicted points, assists, and rebounds based on draft position, we obtained coefficients of -0.05231345, -0.01130981, and -0.02063268 respectively. We interpret each as follows: For each spot later in the draft players were selected, those players would score an average of 0.05231345 less points per game than players selected ahead of them. For each spot later in the draft players were selected, those players would record an average of 0.01130981 less assists per game than players selected ahead of them. For each spot later in the draft players were selected, those players would record an average of 0.02063268 less rebounds per game than players selected ahead of them. This follows intuition, as players selected higher in the draft are expected to (and often do) perform at a higher level than players selected behind them on average. It also follows that a player drafted first would score an average of 10.1 points per game, record an average of 2.2 assists per game, and record an average of 4.3 rebounds per game. Meanwhile, a player drafted sixtieth, last in the draft, would score an average of 7 points per game, record an average of 1.5 assists per game, and record an average of 3.1 rebounds per game.

4.2.3 Deploying the Model Online with Shiny

To implement our model online, we utilized the Shiny Package in R to create a user friendly app. This allows anyone to use our model to predict draft positions

`https://kobylieu.shinyapps.io/DraftPositionPredictor/`

5 Conclusion

In conclusion, we can say that our investigation into NBA draft and performance data has resulted in the development of two models. The first model, a Net Rating Prediction Model, predicts the net rating of a player given certain inputs such as age, height, weight, and various performance metrics. Our analysis of the data showed that Random Forest and Gradient Boosting Regressions were the most effective models for our data. The second model, a Draft Position Prediction Model, predicts a player's draft position by analyzing their performance metrics. Our analysis of the data showed that players with higher projected draft ratings were typically selected earlier in the draft. Through the use of Shiny, we were able to deploy our Draft Position model online, allowing for easy access and utilization. Our findings emphasize the importance of accurately evaluating player performance and prospects, allowing teams to build the strongest roster possible for future success.

References

[1] Henningsen, Evan. "What Is the Highest Paid Sport in the World?" World Sports Network, World Sports Network, 13 Jan. 2023, https://www.wsn.com/blog/highest-paid-sport/.

[2] "Predicting the Outcome of NBA Games with Machine Learning." Towards Data Science, Medium, 6 Aug. 2019, towardsdatascience.com/predicting-the-outcome-of-nba-games-with-machine-learning-a810bb768f20.

[3] "Shiny User Showcase." RStudio Shiny, RStudio, shiny.rstudio.com/tutorial/.

Code Appendix

For the method from STA 141C, we utilized the Cholesky method as our data is symmetric and positive definite. This helped optimize our code by bringing down the runtime to .03 seconds per 100 iterations.

```
---
# Python Net Rating Model Code
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
from sklearn.preprocessing import MinMaxScaler
import numpy as np
---
#Dataset #1
games_details = pd.read_csv ("./data/games_details.csv")
games = pd.read_csv ("./data/games.csv")
players = pd.read_csv ("./data/players.csv")
ranking = pd.read_csv ("./data/ranking.csv")
teams = pd.read_csv ("./data/teams.csv")
---
#Dataset #2
players_data = pd.read_csv ("./data/all_seasons.csv")
---
#Dataset #3
draft_data = pd.read_csv ("./data/NBA_Draft_1980_2017.tsv", sep = '\t')
---
# Remove Undrafted Player Data
players_data = players_data[['draft_round', 'player_height', 'player_weight', 'pts', 'ast', 'oreb
players_data = players_data.drop(players_data[players_data.draft_round == 'Undrafted'].index)#Rem
players_data['draft_round'] = pd.to_numeric(players_data['draft_round'])
---
# Heatmap
fig, ax = plt.subplots(figsize=(10, 10))
sns.heatmap(players_data.corr(), annot=True)
---
# Pairplot
sns.pairplot(players_data)
---
# Testing Data without Outliers (Did not end up using in models)
from scipy import stats

# select only numerical columns
numerical_cols = players_df.select_dtypes(include=np.number).columns.tolist()

# calculate z-scores on numerical columns
z_scores = stats.zscore(players_df[numerical_cols])

# set threshold and mask outliers
threshold = 3
mask = (z_scores.abs() > threshold).any(axis=1)

# filter the dataframe to exclude outliers
df = players_df[~mask]

df
---
# Linear Regression Predicting Net Rating
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Split the dataset into training and testing sets
X = players_df[["age", "player_height", "player_weight", "pts", "reb", "ast","oreb_pct","dreb_pct
y = players_df["net_rating"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Fit the linear regression model to the training data
reg_model = LinearRegression().fit(X_train, y_train)
```

```
# Use the model to make predictions on the testing data
y_pred = reg_model.predict(X_test)

# Calculate the model's performance metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")
---
# Random Forest Regression Predicting Net Rating
from sklearn.ensemble import RandomForestRegressor

# Fit a random forest regression model to the training data
rf_model = RandomForestRegressor(n_estimators=100, random_state=42).fit(X_train, y_train)

# Use the model to make predictions on the testing data
y_pred = rf_model.predict(X_test)

# Calculate the model's performance metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")
---
# Support Vector Regression Predicting Net Rating
from sklearn.svm import SVR

# Fit a support vector regression model to the training data
svr_model = SVR(kernel='rbf', C=100, gamma=0.1).fit(X_train, y_train)

# Use the model to make predictions on the testing data
y_pred = svr_model.predict(X_test)

# Calculate the model's performance metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")
---
# Gradient Boosting Regression Predicting Net Rating
from sklearn.ensemble import GradientBoostingRegressor

# Fit a gradient boosting regression model to the training data
gb_model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_sta

# Use the model to make predictions on the testing data
y_pred = gb_model.predict(X_test)

# Calculate the model's performance metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")
---
```

```
# Ridge Regressiom
from sklearn.linear_model import Ridge

# Load the data
X = players_df[["age", "player_height", "player_weight", "pts", "reb", "ast",
                "oreb_pct", "dreb_pct", "usg_pct", "ts_pct", "ast_pct"]]
y = players_df["net_rating"]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale and normalize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Instantiate the Ridge regression model with default hyperparameters
ridge = Ridge()

# Train the model on the training data
ridge.fit(X_train, y_train)

# Predict the target variable for the testing data
y_pred = ridge.predict(X_test)

# Evaluate the performance of the model using mean squared error and R-squared
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean squared error:", mse)
print("R-squared:", r2)
---
# Lasso With Grid Search CV
# Import the necessary libraries
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Load the data
X = players_df[["age", "player_height", "player_weight", "pts", "reb", "ast",
                "oreb_pct", "dreb_pct", "usg_pct", "ts_pct", "ast_pct"]]
y = players_df["net_rating"]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale and normalize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Set up a grid of hyperparameters to search over
param_grid = {'alpha': [0.1, 1, 10, 100]}

# Instantiate a Lasso regression model
lasso = Lasso()

# Perform cross-validation to find the best hyperparameters
```

```
grid_search = GridSearchCV(lasso, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Train the model on the training data with the best hyperparameters
lasso = grid_search.best_estimator_
lasso.fit(X_train, y_train)

# Predict the target variable for the testing data
y_pred = lasso.predict(X_test)

# Evaluate the performance of the model using mean squared error and R-squared
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean squared error:", mse)
print("R-squared:", r2)
___
# Best Model: Random Forest Regression with GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

# Set up the parameter grid to search over
param_grid = {'n_estimators': [100, 200, 500],
              'max_depth': [None, 10, 20],
              'max_features': ['auto', 'sqrt']}

# Instantiate a random forest regression model
rf_model = RandomForestRegressor(random_state=42)

# Create a GridSearchCV object to find the best hyperparameters
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5)

# Fit the GridSearchCV object to the training data
grid_search.fit(X_train, y_train)

# Best GridSearchCV Model
y_pred = grid_search.predict(X_test)

# Calculate the model's performance metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")
__
# Optimized Random Forest Model Function
def predict_net_rating(age, player_height, player_weight, pts, reb, ast, oreb_pct, dreb_pct, usg_
    # Load the data
    X = players_df[["age", "player_height", "player_weight", "pts", "reb", "ast",
                    "oreb_pct", "dreb_pct", "usg_pct", "ts_pct", "ast_pct"]]
    y = players_df["net_rating"]

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Scale and normalize the data
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
```

```
    # Set up the parameter grid to search over
    param_grid = {'n_estimators': [100, 200, 500],
                  'max_depth': [None, 10, 20],
                  'max_features': ['auto', 'sqrt']}

    # Instantiate a random forest regression model
    rf_model = RandomForestRegressor(random_state=42)

    # Create a GridSearchCV object to find the best hyperparameters
    grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5)

    # Fit the GridSearchCV object to the training data
    grid_search.fit(X_train, y_train)

    # Make a prediction using the trained model
    input_data = [[age, player_height, player_weight, pts, reb, ast, oreb_pct, dreb_pct, usg_pct,
    input_data_scaled = scaler.transform(input_data)
    net_rating = grid_search.predict(input_data_scaled)[0]

    return net_rating
```

```
# Cholesky R Code

players_data <- read.csv("./data/all_seasons.csv")

# Remove undrafted
players_data <- players_data[!(players_data$draft_year == "Undrafted"),]
players_data <- players_data[!(players_data$draft_round == "Undrafted"),]
players_data <- players_data[!(players_data$draft_number == "Undrafted"),]

# Make numeric
players_data$draft_year <- as.numeric(players_data$draft_year)
players_data$draft_round <- as.numeric(players_data$draft_round)
players_data$draft_number <- as.numeric(players_data$draft_number)

# Drop wrong data
players_data <- players_data[players_data$draft_year >= 1990,]
players_data <- players_data[players_data$draft_round <= 2,]
players_data <- players_data[players_data$draft_number <= 60,]
players_data <- players_data[players_data$draft_number >= 1,]

#Set data for draft number
nba_data <- players_data[, c(11, 13:15)]
Xy <- cbind(1, as.matrix(nba_data[, c(2:4,1)])) #X and y
colnames(Xy)[1] <- "intercept"

#Cholesky
#form the Grammian matrix
yXXy <- crossprod(Xy)
#Cholesky decomposition of Grammian matrix
cholR <- chol(yXXy)
#regression coefficients
betahat <- backsolve(cholR[1:4, 1:4], cholR[1:4, 5])

#Cholesky Function
cholfunc <- function(){
  yXXy <- crossprod(Xy)
  cholR <- chol(yXXy)
  betahat <- backsolve(cholR[1:4, 1:4], cholR[1:4, 5])
```

```
}

#Cholesky Time
ptm <- system.time({
  results <- replicate(100, cholfunc())
})

print(betahat)
print(ptm)
---
# Draft Prediction Model Code in R

dataStats <- read.csv("data/all_seasons.csv")
dataStats <- dataStats[-1]
dataStats <- dataStats[-2]
dataStats <- dataStats[, -c(5:6)]
dataStats <- dataStats[-18]

# filter out undrafted players
nba_data <- dataStats[dataStats$draft_number != "Undrafted",]

players_data <- read.csv("data/all_seasons.csv")
# Remove undrafted
players_data <- players_data[!(players_data$draft_year == "Undrafted"),]
players_data <- players_data[!(players_data$draft_round == "Undrafted"),]
players_data <- players_data[!(players_data$draft_number == "Undrafted"),]

# Make numeric
players_data$draft_year <- as.numeric(players_data$draft_year)
players_data$draft_round <- as.numeric(players_data$draft_round)
players_data$draft_number <- as.numeric(players_data$draft_number)

# Drop wrong data
players_data <- players_data[players_data$draft_year >= 1990,]
players_data <- players_data[players_data$draft_round <= 2,]
players_data <- players_data[players_data$draft_number <= 60,]
players_data <- players_data[players_data$draft_number >= 1,]

#Set data for draft number
nba_data <- players_data[, c(4, 5, 6, 11, 12, 13:21)]
#X = nba_data[,2:4]
#y = nba_data[,1]

# convert draft number and player height to numeric variables
nba_data$draft_number <- as.numeric(nba_data$draft_number)
nba_data$player_weight <- as.numeric(nba_data$player_weight)
nba_data$age <- as.numeric(nba_data$age)

# fit a linear regression model for points
lmPoints <- lm(pts ~ draft_number + player_height + age + player_weight + gp + net_rating + oreb_

# predict points for draft picks 1-60
draft_picks <- data.frame(draft_number = 1:60, player_height = mean(nba_data$player_height), age
pred_pts <- predict(lmPoints, newdata = draft_picks)

# fit a linear regression model for rebounds
lmRebounds <- lm(reb ~ draft_number + player_height + age + player_weight + gp + net_rating + ore

# predict rebounds for draft picks 1-60
```

```
pred_reb <- predict(lmRebounds, newdata = draft_picks)

# fit a linear regression model for assists
lmAssists <- lm(ast ~ draft_number + player_height + age + player_weight + gp + net_rating + oreb

# predict assists for draft picks 1-60
pred_ast <- predict(lmAssists, newdata = draft_picks)


print(pred_pts)
print(pred_ast)
print(pred_reb)

# create a data frame with predicted values
pred_df <- data.frame(Draft_Pick = 1:60, Points = pred_pts, Assists = pred_ast, Rebounds = pred_r

# plot predicted values for each category by draft pick
library(ggplot2)
ggplot(pred_df, aes(x = Draft_Pick)) +
  geom_point(aes(y = Points, color = "Points")) +
  geom_point(aes(y = Assists, color = "Assists")) +
  geom_point(aes(y = Rebounds, color = "Rebounds")) +
  scale_color_manual(values = c("red", "green", "blue")) +
  xlab("Draft Position") + ylab("Predicted Statistics") +
  ggtitle("Predicted NBA Statistics based on Draft Position") +
  labs(color = "Category")


# create a matrix of the predicted values
predMat <- cbind(pred_pts, pred_reb, pred_ast)

# normalize the matrix for each column
predMat2 <- apply(predMat, 2, function(x) (x - min(x)) / (max(x) - min(x)))

# create a heatmap using the normalized matrix
heatmap(predMat2, Rowv = NA, Colv = NA, col = colorRampPalette(c("#FFFFFF", "#FFFF00", "#FFA500",

# fit a linear regression model for draft position
lmDraftNumber <- lm(draft_number ~ pts + ast + reb + player_height + age +
                      player_weight + gp + net_rating + oreb_pct + dreb_pct +
                      usg_pct + ts_pct + ast_pct, data = nba_data)

# predict a player's draft position based on all available data
draftnum <- function(df){
  output = predict(lmDraftNumber, newdata = df)
  return(output)
  }

# predict a player's points based on draft number
player_pred_pts <- function(draftnum = 1){
  output = round(pred_pts[draftnum], 1)
  return(output)
}

# predict a player's assists based on draft number
player_pred_ast <- function(draftnum = 1){
  output = round(pred_ast[draftnum], 1)
  return(output)
}
```

```
# predict a player's rebounds based on draft number
player_pred_reb <- function(draftnum = 1){
  output = round(pred_reb[draftnum], 1)
  return(output)
}
```

```
# Shiny App Code: Draft Position Predictor App

#
# This is a Shiny web application. You can run the application by clicking
# the 'Run App' button above.
#
# Find out more about building applications with Shiny here:
#
#    http://shiny.rstudio.com/
#

source("functions/DraftRoundPredictorFunction.R")
library(shiny)

# Define UI for application that draws a histogram
ui <- fluidPage(

    # Application title
    titlePanel("NBA Player Draft Position Predictor"),

    # Enter average points
    sliderInput("avg_pts", label = h3("Enter average points"), min = 1,
                max = 60, value = 1),

    # Enter average rebounds
    sliderInput("avg_reb", label = h3("Enter average rebounds"), min = 1,
                max = 30, value = 1),

    # Enter average assists
    sliderInput("avg_ast", label = h3("Enter average assists"), min = 1,
                max = 30, value = 1),

    # Enter player height (inches)
    sliderInput("height", label = h3("Enter player height (inches)"),
                min = 48, max = 96, value = 48),

    # Enter player age (years)
    sliderInput("age", label = h3("Enter player age (years)"),
                min = 18, max = 50, value = 18),

    # Enter player weight (pounds)
    sliderInput("weight", label = h3("Enter player weight (pounds)"),
                min = 80, max = 300, value = 80),

    # Enter player games played
    sliderInput("gp", label = h3("Enter player games played"),
                min = 0, max = 82, value = 0),

    # Enter player net rating
    sliderInput("net_rating", label = h3("Enter player net rating"),
                min = -20, max = 20, value = -20),
```

```
    # Enter player offensive rebound percentage
    sliderInput("oreb_pct", label = h3("Enter player offensive rebound percentage"),
                min = 0, max = 0.20, value = 0),

    # Enter player defensive rebound percentage
    sliderInput("dreb_pct", label = h3("Enter player defensive rebound percentage"),
                min = 0, max = 0.30, value = 0),

    # Enter player usage percentage
    sliderInput("usg_pct", label = h3("Enter player usage percentage"),
                min = 0, max = 0.40, value = 0),

    # Enter player true shooting percentage
    sliderInput("ts_pct", label = h3("Enter player true shooting percentage"),
                min = 0.30, max = 0.90, value = 0),

    # Enter player assist percentage
    sliderInput("ast_pct", label = h3("Enter player assist percentage"),
                min = 0, max = 0.60, value = 0),

    # Submit button for app
    submitButton(text = "Apply", icon = NULL, width = NULL),

    mainPanel(
        textOutput("avg_pts"), textOutput("avg_reb"), textOutput("avg_ast"),
        textOutput("height"), textOutput("age"), textOutput("weight"),
        textOutput("gp"), textOutput("net_rating"), textOutput("oreb_pct"),
        textOutput("dreb_pct"), textOutput("usg_pct"),
        textOutput("ts_pct"), textOutput("ast_pct"),
        textOutput("draft_position_prediction")
    )
)

# Define server logic required to draw a histogram
server <- function(input, output) {

    # display predicted draft round prediction
    output$draft_position_prediction <- renderText({
        paste("A player with the above NBA statistics is predicted by our model to be drafted No.
              round(draftnum(data.frame(pts = input$avg_pts,
                                        reb = input$avg_reb,
                                        ast = input$avg_ast,
                                        player_height = input$height,
                                        age = input$age,
                                        player_weight = input$weight,
                                        gp = input$gp,
                                        net_rating = input$net_rating,
                                        oreb_pct = input$oreb_pct,
                                        dreb_pct = input$dreb_pct,
                                        usg_pct = input$usg_pct,
                                        ts_pct = input$ts_pct,
                                        ast_pct = input$ast_pct, 0)
                                        )), " overall in the NBA draft."

        )
        })

    }
```

```
# Run the application
shinyApp(ui = ui, server = server)

Shiny app code: points/assists/rebounds predictor app

#
# This is a Shiny web application. You can run the application by clicking
# the 'Run App' button above.
#
# Find out more about building applications with Shiny here:
#
#    http://shiny.rstudio.com/
#

source("functions/DraftRoundPredictorFunction.R")
library(shiny)

# Define UI for application that draws a histogram
ui <- fluidPage(

    # Application title
    titlePanel("NBA Player Points, Assists, and Rebounds Predictor"),

    # Enter player draft position
    # numericInput("draft_pos", label = h3("Enter average points"), value = 1),

    sliderInput("draft_pos", label = h3("Enter player draft position"), min = 1,
                max = 60, value = 1),

    # Submit button for app
    submitButton(text = "Apply", icon = NULL, width = NULL),

    mainPanel(
        textOutput("draft_pos"),textOutput("pts_reb_ast_prediction")
    )
)

# Define server logic required to draw a histogram
server <- function(input, output) {

    # display predicted draft round prediction
    output$pts_reb_ast_prediction <- renderText({
        paste("A player drafted No. ", input$draft_pos, " is predicted by our model to score an a
               player_pred_pts(input$draft_pos), " points per game, record an average of ",
              player_pred_ast(input$draft_pos), "assists per game, and record an average of ",
              player_pred_reb(input$draft_pos), " rebounds per game."

        )
        })

    }

# Run the application
shinyApp(ui = ui, server = server)
---
```