Predicting the Next Defensive Player of the Year in the NBA

Joshua Samuel
jsamuel41@tamu.edu

Alex LeGresley
alexlegresley@tamu.edu

Texas A&M University

Contents

1 Importance of Study

We seek to predict the winner of this year's NBA's Defensive Player of the Year Award, in addition to identifying other players who will be in contention for the award and how they will stack up. The Defensive Player of the Year (DPOY) award is one of the most coveted awards given for the performance of NBA players during the regular season, second only to the overall Most Valuable Player of the Year Award. With betting continuing to become more and more prevalent in sports, the amount of money won and lost regarding regular season awards continues to increase dramatically. Furthermore, player contracts often contain lucrative incentives for winning regular season awards, nevertheless the intangible boost to one's legacy and status in the game. Thus, statistical prediction of the DPOY award is of potential importance to a variety of parties.

2 Exploratory Data Analysis

2.1 Description of Dataset

The dataset used in this analysis was aggregated from various collections provided by Basketball Reference. The data was obtained in CSV and Excel format. It comprises of four primary datasets spanning the NBA seasons from 1982-1983 to 2022-2023. The first dataset encompasses standard statistics (such as blocks, steals, etc.) of all active players who participated in at least one game during this period. The second dataset consists of advanced metrics (including box plus/minus, win shares, etc.) of the same players. The third dataset, covering the same time frame, lists every player who received at least one vote for the Defensive Player of the Year award in each respective season. The fourth and final dataset adds the respective team standings position for every player in the dataset.

2.2 Data Cleaning

Duplicates were removed from each dataset, specifically instances where a player switched teams within a season. Instead of retaining separate observations for each team, the statistics for such players were aggregated for the entire season. The team rank associated with players who played for multiple teams during a season was taken to be the average of the teams ranks they played for weighted by the number of minutes played for each team. Ultimately, each row of the dataset was a unique combination of player and season. A few columns provided in the Basketball-Reference dataset did not have meaningful associated names and thus were dropped since they could not be confidently regarded as a known metric. This led to a total of 49 meaningful covariance and one response variable.

While the details of the statistical models will be covered in detail later, data setup was necessary to be congruent with their limitations. All but one of the covariates was quantitative, with the player position being multi-class categorical. The XGBoost implementation provides built-in support for handling categorical variables, however the Scikit-Learn implementation of linear models does not. Dummy variables for player position were created for use with the linear models. Furthermore, many of the quantitative covariates had missing values. The XGBoost implementation also provides built-in support for handling missing values, however the Scikit-Learn linear models implementation does not. Median imputation was used to fill the missing values for use with the linear models. We chose median imputation due to the median being more robust to skewness, as we anticipated some of the covariates would be skewed.

2.3 Summary Statistics

The distribution of the share of Defensive Player of the Year Award votes is highly right-skewed, with a vast majority of players having a vote share of zero. There is nearly a 30:1 ratio of players who did not receive a DPOY award vote to those who received at least one vote. This important characteristic of the data is accounted for in our methodology.

3 Methodology

To address the significant imbalanced nature of the voting for the DPOY award due to very few players receiving any votes, we will employ a two stage process.

3.1 Stage 1

The first stage model aims to classify whether a player is likely to receive any votes for the Defensive Player of the Year award. We will build the stage 1 model first with XGBoost, then compare the performance with a simpler logistic regression model. For this model, we will transform the response variable, which is the DPOY vote share for each player in each season, to a binary outcome: 1 if they received any share, that is their vote share is non-zero, and zero otherwise. The classes will be very unbalanced, with many zeros and a few ones. We will balance the classes in the statistical model training process by adjusting the class weights such that the observations of players who received a vote will be weighted higher. Specifically, the class weights will correspond to a 30:1 ratio as found during the exploratory data analysis process.

Model performance will be evaluated using the metric area under the precision recall curve (AUCPR). This metric is advantageous in scenarios with significant class imbalance since it will penalize false negatives. This is necessary because a model that only predicts zero would have a high accuracy but ultimately not be useful. We will use cross-validation when evaluating potential models, as it provides a statistically more reliable estimation of the performance than a single train/test split. Specifically, we will use stratified k-fold cross-validation. Stratification is necessary due to the class imbalance, as it helps ensure an approximately equal representation of classes in each group compared to standard k-fold cross-validation. Furthermore, the model will output probabilities of belonging to the positive class, that is receiving any vote share, and it is up to us to determine whether the probability is high enough to expect the player to receive any vote share.

We will tune this probability threshold using the F1 score metric, which is also a robust metric in cases of class imbalance. Ultimately, the stage 1 model will act as a filter, removing those who are unlikely to receive any votes. We can then take those who we expect to receive at least one vote and inspect on a more granular level to determine the specific amount of vote share.

3.2 Stage 2

The second stage model predicts the amount of share in the Defensive Player of the Year Award for players classified in the previous model as projected to receive a share. We will build the stage 2 model with XGBoost, then compare the performance with a simpler linear regression model. The player predicted to have the highest share in a given season is expected to win the award.

We will work with the response variable as a continuous quantitative variable and while utilizing the same covariates as the first model. Model performance will be evaluated using the metrics root mean squared error (RMSE) and R^2. RMSE is a robust standardized metric that indicates how well the model is able to predict the response variable, while R^2 indicates how well the covariates explain the variation in the response. Furthermore, we will use cross-validation when evaluating potential models. Specifically, k-fold cross-validation since the response is quantitative.

4 Stage 1 Model

XGBoost classifier models were fit. The dataset consists of 17900 observations (all players appearing in at least one game from the 1982-1983 to 2022-2023 seasons) with 49 covariates (48 quantitative, 1 categorical) and 1 binary response. The class weights were adjusted to account for the imbalance and stratified cross-validation employed with evaluation by AUCPR. The built-in implementation handling of categorical predictors and missing observations was also used.

4.1 Initial Results

We initially fit the model with default hyperparameters. This yielded an average AUCPR from cross-validation of 0.478 and an average F1 score from cross-validation of 0.469. These values suggest that the model is similar to a coin flip in terms of determining whether a player is likely to have received at least one DPOY vote, and thus there is much room for improvement in the predictive performance of the model.

4.2 Tuning

We turned to tuning the hyperparameters of the model as a source of potential improvement. We identified the XGBoost hyperparameters learning rate, maximum depth of trees, and number of estimators as being

most likely to result in significantly enhanced predictive performance. We chose a randomized search technique due to the balance between computational efficiency and ability to search a wide space of potential values for the hyperparameters. Specifically, randomized search helps find optimal hyperparameter values by randomly sampling with replacement from a given statistical distribution of hyperparameter values. This randomized search process was ran with 75 iterations of stratified 5-fold cross-validation. The hyperparameters leading to the model with the maximum AUCPR were considered optimal.

4.3 Post-Tuned Results

The average AUCPR from cross-validation for our best model was 0.609 with the best parameters being 0.153 for the learning rate, 11 for the max tree depth, and 162 for the number of estimators. We next tuned the probability decision threshold for considering a player to receive at least one DPOY vote. Stratified 5-fold cross-validation was used to test thresholds in increments of 0.05 with evaluation by F1 score. The average best threshold from cross-validation was 0.35 with a corresponding average F1 score from cross-validation of 0.581.

The feature importance chart Figure 1 shows the relative importance of each feature in the model. Features with higher importance have more influence on the model's predictions, while those with lower importance have less. It is not surprising to see defensive metrics such as Personal Fouls (PF), Defensive Box Plus Minus (DBPM), and Defensive Win Share (DWS) near the top in terms of influence. However, it is surprising to see Steal % (STL%) with the least influence, as this is a commonly cited defensive metric.

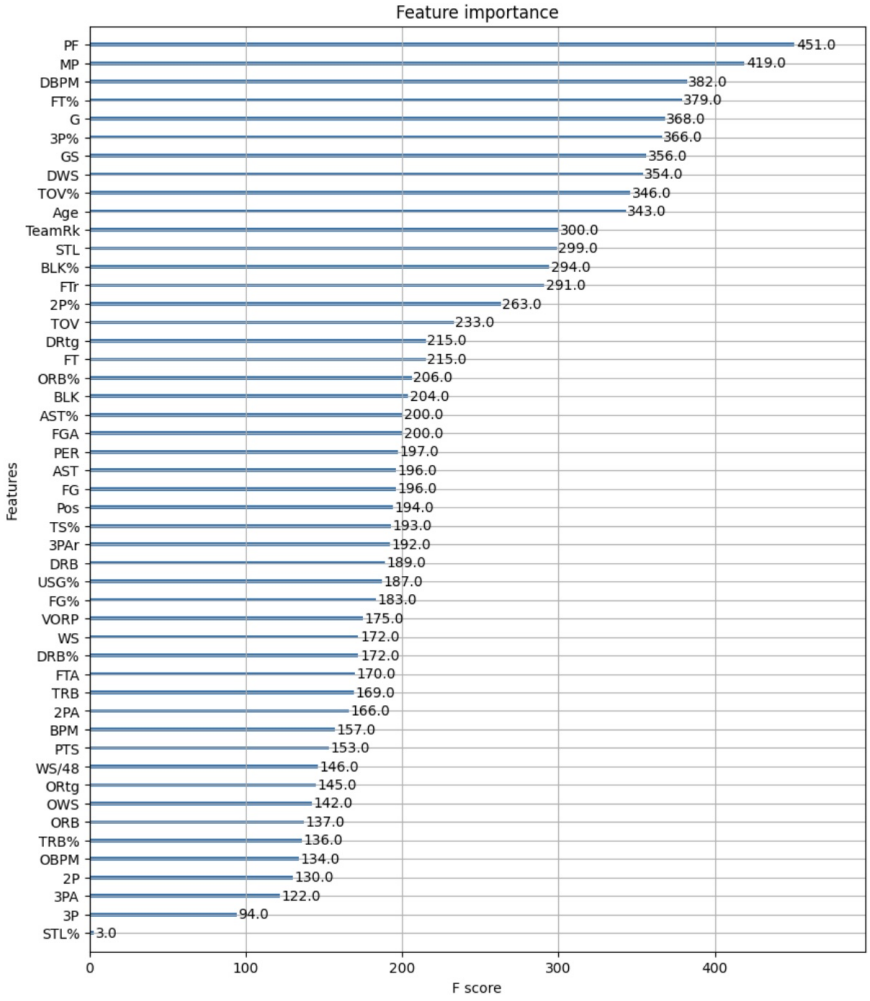

Figure 1: Stage 1 feature importance for tuned XGBoost classifier model

4.4 Logistic Model Comparison

To investigate whether a complex model such as XGBoost is necessary, we compared the performance to a simpler logistic regression model. Class weights were adjusted to account for the imbalance. Dummy variables were created for the categorical Position covariate and missing data for quantitative covariates was imputed using the median. Random search with 75 iterations of stratified 5-fold cross-validation was used to tune the C hyperparameter, which controls the aggressiveness of the regularization process. The optimal model was that which maximized AUCPR.

The best logistic regression model had an average AUCPR from cross-validation of 0.496. The probability decision threshold was optimized by employing stratified 5-fold cross-validation to test thresholds in increments of 0.05 with evaluation by F1 score. The maximum average F1 score from cross-validation was 0.524 achieved at a threshold of 0.90.

The AUCPR and F1 score for the logistic regression model was slightly lower than those of the XGBoost model. Thus, we conclude that the XGBoost model has slightly better predictive performance.

5 Stage 2 Model

XGBoost regressor models were fit. The dataset consists of 575 observations (players receiving any vote share from the 1982-1983 to 2022-2023 seasons) with the same 49 covariates (48 quantitative, 1 categorical) as the stage 1 model and 1 quantitative response. K-fold cross-validation and evaluation by RMSE was used to select the best model. The built-in implementation handling of categorical predictors and missing observations was also used.

5.1 Tuning

We tuned the same XGBoost hyperparameters as in stage 1, those being learning rate, max tree depth, and number of estimators. Randomized search was used with 75 iterations of 5-fold cross-validation with evaluation by -RMSE. The use of -RMSE rather than RMSE is an implementation detail consequence of the Sckikit-Learn Unified Scoring API. The interpretation is the same, that is a score closer to zero represents less prediction error and is desirable. The model that minimized -RMSE was considered optimal.

5.2 Post-Tuned Results

The average -RMSE from cross-Validation for our best model was -0.153 with the best parameters being 0.097 for the learning rate, 3 for the max tree depth, and 94 for the number of estimators. Furthermore, the average R^2 from cross-validation was 0.292. This means that our model explains only about 29.2% of the variability in DPOY vote share, and thus there is definitely room for improvement.

The feature importance chart Figure 2 shows the relative importance of each feature in the XGBoost model. Advanced defensive metrics such as Defensive Win Shares (DWS) and Defensive Box Plus Minus (DBPM) were considered the two most important features which is not surprising. However, Blocks (BLK) and Defensive Rebounding Percentage (DRB%) being low in terms of importance is surprising given that they are considered by many to be important defensive statistics. Field Goal Percentage (FG%) being ranked 7th in importance illustrates how the efficiency of a player on the offensive end still plays a factor in whether they win the award.

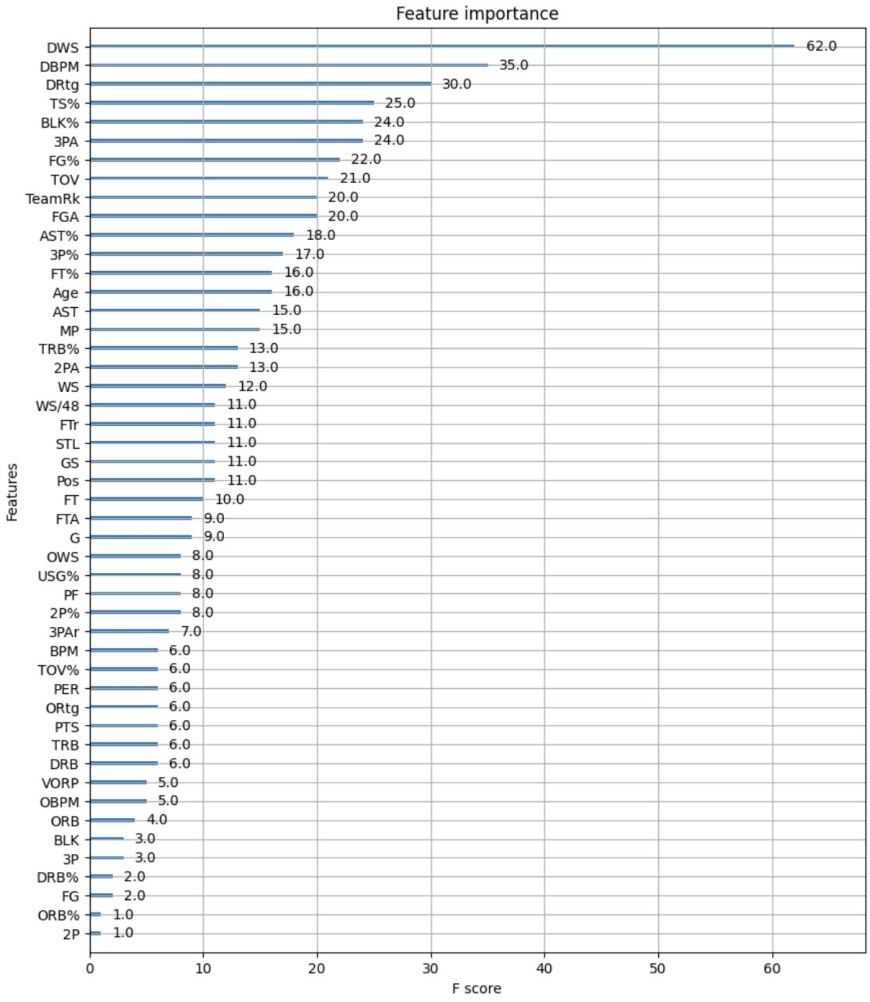

Figure 2: Stage 2 feature importance for XGBoost regressor model

5.3 Linear Model Comparison

To investigate whether a complex model such as XGBoost is again necessary, we compared the performance to a simpler linear regression model. Dummy variables were created for the categorical Position covariate and missing data for quantitative covariates was imputed using the median. Reverse Feature Elimination (RFE) was used to identify the optimal number of predictors to include in the model. This was run with 75 iterations of 5-fold cross-validation and the best model identified based on minimizing -RMSE.

The best linear regression model had an average -RMSE from cross-validation of -0.167 and an average R^2 from cross-validation of 0.200. Thus, the linear regression model has slightly higher prediction error and explains slightly less of the variability in vote share compared to the XGBoost regressor model.

6 Current Season Results

In line with our goal, we tested the stage 1 and stage 2 XGBoost models and the stage 1 and stage 2 linear models with the current NBA season data to predict this year's Defensive Player of the Year. The current season has concluded so all data is available, however the award has not yet been given out. The predictions using the stage 1 and stage 2 XGBoost models are shown in Table 1, the stage 1 and stage 2 linear models in Table 2, and the current Vegas betting odds from the DraftKings Sportsbook as of 4/17/2024 in Table 3.

Our XGBoost process performed quite well as the top 4 DPOY candidates according to Vegas are accounted for in the top 8 for our preferred process. Our XGBoost's top 2 candidates are the same as Vegas however

the order is different. Our linear model process had a few of the same projected candidates as our XGBoost process. However, there are two very significant outliers at the top of the linear process's ranking, as background knowledge of the NBA season reinforces that Markquis Nowell and Hamidou Diallo are in reality nowhere close to being in consideration for the award. The two players currently play in the development league (G League) of their respective teams and will occasionally be called up to play for the actual NBA team.

Table 1: XGBoost predicted DPOY vote share

Place	Player	Vote Share
1	Victor Wembanyama	0.345
2	Rudy Gobert	0.283
3	Chet Holmgren	0.185
4	Kristaps Porzingis	0.182
5	Anthony Davis	0.144
6	Giannis Antetokounmpo	0.137
7	Shai Gilgeous-Alexander	0.135
8	Bam Adebayo	0.062
9	Nic Claxton	0.008

Table 2: Linear model predicted DPOY vote share

Place	Player	Vote Share
1	Markquis Nowell	0.587
2	Hamidou Diallo	0.351
3	Rudy Gobert	0.321
4	Chet Holmgren	0.152
5	Victor Wembanyama	0.147
6	Anthony Davis	0.143
7	Shai Gilgeous-Alexander	0.142
8	Isaiah Hartenstein	0.131
9	Josh Hart	0.130
10	Nikola Jokic	0.124
11	Derrick White	0.116
12	Herbert Jones	0.110
13	Shaquille Harrison	0.108
14	Giannis Antetokounmpo	0.089
15	Brook Lopez	0.080
16	Kentavious Caldwell-Pope	0.073
17	Jrue Holliday	0.069

Table 3: DraftKings Sportsbook DPOY odds as of 4/17/2024

Player	Odds
Rudy Gobert	-3,000
Victor Wembanyama	+1,000
Bam Adebayo	+20,000
Anthony Davis	+20,000
Jarret Allen	+20,000
Derrick White	+20,000

7 Final Conclusions & Future Modifications

Our two-stage XGBoost model process predicts with reasonable accuracy which players will be near the top of the DPOY award voting based off data from a given regular season. We have also been able to identify which

metrics contribute most (and least) significantly in predicting how much of a share in DPOY votes a player will receive. We have seen how our preferred tuned XGBoost models outperform our more elementary linear models. In the future, we would consider taking into account the years of NBA experience a player has, since typically the DPOY award is not given to rookies. While we did have an "age" predictor, an additional "years of experience" predictor would be helpful given players enter the league at various ages.

A limitation of our model is that the final predictions are not restricted such that the vote shares sum to one. This is an issue if the specific vote share itself if highly desired to be accurate. In our case, we cared more about the relative rankings of the players in terms of the order they will finish in the voting. In the future, we would like to refine the model to address this limitation and thus produce more realistic vote shares.

8 References

[1] Sports Reference LLC. Basketball-Reference.com - Basketball Statistics and History. https://www.basketball-reference.com/.

[2] Salah, AJ. "*2024 NBA Defensive Player of the Year Odds - Gobert Huge Favorite.*" Covers, 17 Apr. 2024. https://www.covers.com/nba/nba-defensive-player-of-the-year-odds.

9 Code & Data

The data files and a Python notebook containing the code used to produce all of the analyses with explanatory code comments can be obtained from https://github.com/alexlegresley/STAT482-Capstone. The code is also listed below in the Code Appendix.

10 Code Appendix

```
# ----------------------------------------------------------------------
# Imports

import pathlib
import os
import pandas as pd
import numpy as np
import functools
import operator

import scipy.stats as stats
from matplotlib import pyplot as plt

import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV, StratifiedKFold, KFold,
↪  cross_val_score
from sklearn.metrics import f1_score
from sklearn.linear_model import LogisticRegressionCV, LinearRegression
from sklearn.feature_selection import RFECV


# ----------------------------------------------------------------------
# Utility functions

def computeWeightedAverage(row):
    combos = basicStats.loc[(basicStats["Season"] == row["Season"]) & (basicStats["Tm"] !=
    ↪  "TOT") & (basicStats["Player"] == row["Player"])]
```

```
    rkKeys = zip(combos["Season"], combos["Tm"])
    weights = combos["MP"] / combos["MP"].sum()
    return np.average([rkMapping[k] for k in rkKeys], weights = weights)


def optimizeThreshold(model, X, y):
    best = []
    for trainIndex, testIndex in StratifiedKFold(n_splits = 5).split(X, y):
        mod = model.fit(X.iloc[trainIndex], y.iloc[trainIndex])
        res = []
        for th in np.arange(0.05, 1, 0.05):
            yPred = mod.predict_proba(X.iloc[testIndex])[:, 1] > th
            res.append((th, f1_score(y.iloc[testIndex], yPred)))
        best.append(max(res, key = operator.itemgetter(1)))

    return np.average([th for th, _ in best]).round(2), np.average([f1 for _, f1 in best])


def withDummyPos(X):
    posFeatures = pd.get_dummies(X.Pos, drop_first = True, prefix = "pos")
    return pd.concat([X, posFeatures], axis = 1).drop("Pos", axis = 1).apply(lambda x:
    ↪  x.fillna(x.median()) if pd.api.types.is_numeric_dtype(x) else x)


def createModelData(basic_stats, adv_stats, standings):
    basic = pd.read_csv(basic_stats)
    adv = pd.read_csv(adv_stats)
    stand = pd.read_csv(standings, header = 1)

    basic.loc[:, "Player"] = basic["Player"].map(lambda x: x.rstrip("*"))
    adv.loc[:, "Player"] = adv["Player"].map(lambda x: x.rstrip("*"))

    tmMapping = {
        "Atlanta Hawks": "ATL",
        "Boston Celtics": "BOS",
        "Brooklyn Nets": "BRK",
        "Charlotte Hornets": "CHO",
        "Chicago Bulls": "CHI",
        "Cleveland Cavaliers": "CLE",
        "Dallas Mavericks": "DAL",
        "Denver Nuggets": "DEN",
        "Detroit Pistons": "DET",
        "Golden State Warriors": "GSW",
        "Houston Rockets": "HOU",
        "Indiana Pacers": "IND",
        "Los Angeles Clippers": "LAC",
        "Los Angeles Lakers": "LAL",
        "Memphis Grizzlies": "MEM",
        "Miami Heat": "MIA",
        "Milwaukee Bucks": "MIL",
        "Minnesota Timberwolves": "MIN",
        "New Orleans Pelicans": "NOP",
        "New York Knicks": "NYK",
        "Oklahoma City Thunder": "OKC",
```

```
        "Orlando Magic": "ORL",
        "Philadelphia 76ers": "PHI",
        "Phoenix Suns": "PHO",
        "Portland Trail Blazers": "POR",
        "Sacramento Kings": "SAC",
        "San Antonio Spurs": "SAS",
        "Toronto Raptors": "TOR",
        "Utah Jazz": "UTA",
        "Washington Wizards": "WAS"
    }
    stand["Tm"] = stand["Team"].map(tmMapping)
    rkMapping = dict(zip(stand["Tm"], stand["Rk"]))

    def computeWeightedAverage(row):
        combos = basic.loc[(basic["Player"] == row["Player"]) & (basic["Tm"] != "TOT")]
        weights = combos["MP"] / combos["MP"].sum()
        return np.average([rkMapping[k] for k in combos["Tm"]], weights = weights)

    basic["TeamRk"] = [rkMapping[rowKey] if (rowKey := row["Tm"]) in rkMapping.keys() else
    ↪  computeWeightedAverage(row) for _, row in basic.iterrows()]

    basic.drop_duplicates(subset = "Player", keep = "first", inplace = True)
    adv.drop_duplicates(subset = "Player", keep = "first", inplace = True)

    basic.drop(["Rk", "Tm", "Unnamed: 29", "Player-additional"], axis = 1, inplace = True)
    adv.drop(["Rk", "Pos", "Age", "Tm", "G", "MP", "Unnamed: 19", "Unnamed: 24",
    ↪  "Player-additional"], axis = 1, inplace = True)

    combined = pd.merge(basic, adv, how = "left", on = "Player")
    combined["Pos"] = combined["Pos"].astype("category")

    return combined


def predictVoteShare(data, stage1, threshold, stage2):
    probSomeShare = stage1.predict_proba(data.drop("Player", axis = 1))[:, 1]

    noShare = data.loc[~(probSomeShare > threshold)].copy()
    noShare["Vote_Share"] = 0

    someShare = data.loc[probSomeShare > threshold].copy()
    someShare["Vote_Share"] = stage2.predict(someShare.drop("Player", axis = 1))

    return pd.concat([someShare[["Player", "Vote_Share"]], noShare[["Player",
    ↪  "Vote_Share"]]], ignore_index = True).sort_values("Vote_Share", ascending =
    ↪  False).reset_index(drop = True)


def matchDummiesPos(X, XTrain):
    presentDummies = pd.get_dummies(X, prefix = "pos")
    res = pd.DataFrame()
    for d in [d for d in pd.get_dummies(XTrain, prefix = "pos").columns if
    ↪  d.startswith("pos_")]:
        if d in presentDummies.columns:
```

```
            res[d] = presentDummies[d]
        else:
            res[d] = False
    res.drop(columns = res.columns[0], axis = 1, inplace = True)
    return pd.concat([X, res], axis = 1).drop("Pos", axis = 1).apply(lambda x:
    ↪  x.fillna(x.median()) if pd.api.types.is_numeric_dtype(x) else x)


# ------------------------------------------------------------------
# Data cleaning

basicStats = pd.concat(
    (pd.read_csv(f).assign(Season = os.path.basename(f).removesuffix(".csv")) for f in
    ↪  (pathlib.Path()/"Data"/"Basic_Stats").glob("*.csv")),
    ignore_index = True
)
advStats = pd.concat(
    (pd.read_csv(f).assign(Season = os.path.basename(f).removesuffix(".csv")) for f in
    ↪  (pathlib.Path()/"Data"/"Advanced_Stats").glob("*.csv")),
    ignore_index = True
)
dpoyVoting = pd.concat(
    (pd.read_excel(f, header = 1).assign(Season =
    ↪  os.path.basename(f).removesuffix(".xlsx")) for f in
    ↪  (pathlib.Path()/"Data"/"DPOY_Voting").glob("*.xlsx")),
    ignore_index = True
)
expStand = pd.concat(
    (pd.read_csv(f, header = 1).assign(Season = os.path.basename(f).removesuffix(".csv"))
    ↪  for f in (pathlib.Path()/"Data"/"Standings").glob("*.csv")),
    ignore_index = True
)

teamMappings = {
    "Atlanta Hawks": "ATL",
    "Boston Celtics": "BOS",
    "Brooklyn Nets": "BRK",
    "Charlotte Bobcats": "CHA",
    "Charlotte Hornets": "CHO",
    "Chicago Bulls": "CHI",
    "Cleveland Cavaliers": "CLE",
    "Dallas Mavericks": "DAL",
    "Denver Nuggets": "DEN",
    "Detroit Pistons": "DET",
    "Golden State Warriors": "GSW",
    "Houston Rockets": "HOU",
    "Indiana Pacers": "IND",
    "Kansas City Kings": "KCK",
    "Los Angeles Clippers": "LAC",
    "Los Angeles Lakers": "LAL",
    "Memphis Grizzlies": "MEM",
    "Miami Heat": "MIA",
    "Milwaukee Bucks": "MIL",
    "Minnesota Timberwolves": "MIN",
```

```
    "New Jersey Nets": "NJN",
    "New Orleans Hornets" : "NOH",
    "New Orleans Pelicans": "NOP",
    "New Orleans/Oklahoma City Hornets": "NOK",
    "New York Knicks": "NYK",
    "Oklahoma City Thunder": "OKC",
    "Orlando Magic": "ORL",
    "Philadelphia 76ers": "PHI",
    "Phoenix Suns": "PHO",
    "Portland Trail Blazers": "POR",
    "Sacramento Kings": "SAC",
    "San Antonio Spurs": "SAS",
    "San Diego Clippers": "SDC",
    "Seattle SuperSonics": "SEA",
    "Toronto Raptors": "TOR",
    "Utah Jazz": "UTA",
    "Vancouver Grizzlies": "VAN",
    "Washington Bullets": "WSB",
    "Washington Wizards": "WAS"
}
expStand["Tm"] = expStand["Team"].map(teamMappings)
rkMapping = dict(zip(zip(expStand["Season"], expStand["Tm"]), expStand["Rk"]))

basicStats.loc[:, "Tm"] = basicStats["Tm"].apply(lambda x: "CHO" if x == "CHH" else x)
advStats.loc[:, "Tm"] = advStats["Tm"].apply(lambda x: "CHO" if x == "CHH" else x)

basicStats["TeamRk"] = [rkMapping[rowKey] if (rowKey := (row["Season"], row["Tm"])) in
↪  rkMapping.keys() else computeWeightedAverage(row) for _, row in basicStats.iterrows()]

basicStats.drop_duplicates(subset = ["Season", "Player"], keep = "first", inplace = True)
advStats.drop_duplicates(subset = ["Season", "Player"], keep = "first", inplace = True)

basicStats.loc[:, "Player"] = basicStats["Player"].map(lambda x: x.rstrip("*"))
advStats.loc[:, "Player"] = advStats["Player"].map(lambda x: x.rstrip("*"))

basicStats = basicStats.drop(["Rk", "Tm", "Unnamed: 29", "Player-additional"], axis = 1)
advStats = advStats.drop(["Rk", "Pos", "Age", "Tm", "G", "MP", "Unnamed: 19", "Unnamed:
↪  24", "Player-additional"], axis = 1)
dpoyVoting = dpoyVoting[["Season", "Player", "Share"]]

combinedData = functools.reduce(
    lambda left, right: pd.merge(left, right, how = "left", on = ["Season", "Player"]),
    [basicStats, advStats, dpoyVoting]
)
combinedData["Share"] = combinedData["Share"].fillna(0)


# ---------------------------------------------------------------------
# Stage 1

model1Data = combinedData.drop(columns = ["Season", "Player"])
model1Data["Pos"] = model1Data["Pos"].astype("category")
model1Data["ShareBinary"] = [1 if x != 0 else 0 for x in model1Data["Share"]]

```

```
X = model1Data.drop(["Share", "ShareBinary"], axis = 1)
y = model1Data["ShareBinary"]

classWeightRatio = y.value_counts()[0] / y.value_counts()[1]


# XGBoost classifier

mod = xgb.XGBClassifier(
    scale_pos_weight = classWeightRatio,
    eval_metric = "aucpr",
    enable_categorical = True
)
paramsDist = {
    "learning_rate": stats.uniform(0.01, 0.20),
    "max_depth": stats.randint(3, 12),
    "n_estimators": stats.randint(50, 350)
}
rsMod = RandomizedSearchCV(
    estimator = mod,
    param_distributions = paramsDist,
    scoring = "average_precision",
    n_iter = 75,
    cv = StratifiedKFold(n_splits = 5),
    refit = True
)
rsMod.fit(X, y)
print(f"Best parameters: {rsMod.best_params_}")
print(f"Average cross validation score (AUCPR) from best model: {rsMod.best_score_}")
tunedMod = rsMod.best_estimator_

_, ax = plt.subplots(figsize = (10, 12))
xgb.plot_importance(tunedMod, ax = ax)

bestThXGB, bestF1XGB = optimizeThreshold(tunedMod, X, y)
print(f"Average best threshold: {bestThXGB}")
print(f"Average best F1 score: {bestF1XGB}")


# Logistic regression

modLR = LogisticRegressionCV(
    class_weight = "balanced",
    scoring = "average_precision",
    solver = "newton-cholesky",
    cv = StratifiedKFold(n_splits = 5),
    refit = True
)
modLR.fit(withDummyPos(X), y)
print(f"Average cross validation score (AUCPR): {np.average(modLR.scores_[1])}")

bestThLR, bestF1LR = optimizeThreshold(modLR, withDummyPos(X), y)
print(f"Average best threshold: {bestThLR}")
print(f"Average best F1 score: {bestF1LR}")
```

```

# ------------------------------------------------------------------
# Stage 2

model2Data = model1Data.drop(["ShareBinary"], axis = 1)
model2Data = model2Data.loc[model2Data["Share"].gt(0)]
model2Data.reset_index(drop = True, inplace = True)

X = model2Data.drop(["Share"], axis = 1)
y = model2Data["Share"]


# XGBoost regressor

mod2 = xgb.XGBRegressor(
    eval_metric = "rmse",
    enable_categorical = True
)
paramsDist = {
    "learning_rate": stats.uniform(0.01, 0.20),
    "max_depth": stats.randint(3, 12),
    "n_estimators": stats.randint(25, 300)
}
rsMod2 = RandomizedSearchCV(
    estimator = mod2,
    param_distributions = paramsDist,
    scoring = "neg_root_mean_squared_error",
    n_iter = 75,
    cv = KFold(n_splits = 5),
    refit = True
)
rsMod2.fit(X, y)
print(f"Best parameters: {rsMod2.best_params_}")
print(f"Best cross validation score (neg RMSE): {rsMod2.best_score_}")
tunedMod2 = rsMod2.best_estimator_
print(
    f"Average cross validation score (r2): "
    f"{np.average(cross_val_score(tunedMod2, X, y, scoring = "r2", cv = KFold(n_splits =
    ↪  5)))}"
)

_, ax = plt.subplots(figsize = (10, 12))
xgb.plot_importance(tunedMod2, ax = ax)


# Linear regression

modLR2 = RFECV(
    estimator = LinearRegression(),
    min_features_to_select = 1,
    step = 1,
    scoring = "neg_root_mean_squared_error",
    cv = KFold(n_splits = 5)
```

```
)
modLR2.fit(withDummyPos(X), y)
print(
    f"Average cross validation score (negative RMSE): "
    f"{np.average(modLR2.cv_results_["mean_test_score"])}"
)
print(
    f"Average cross validation score (r2): "
    f"{np.average(cross_val_score(modLR2, withDummyPos(X), y, scoring = "r2", cv =
    ↪  KFold(n_splits = 5)))}"
)


# ------------------------------------------------------------------------
# Current year prediction

currentYear = createModelData(
    basic_stats =
    ↪  (pathlib.Path()/"Data"/"Current_Year"/"2023-2024_Basic_Stats.csv").resolve(),
    adv_stats = (pathlib.Path()/"Data"/"Current_Year"/"2023-2024_Adv_Stats.csv").resolve(),
    standings = (pathlib.Path()/"Data"/"Current_Year"/"2023-2024_Standings.csv").resolve()
)


# XGBoost models

resultsXGB = predictVoteShare(data = currentYear, stage1 = tunedMod, threshold = bestThXGB,
↪  stage2 = tunedMod2)
resultsXGB.loc[resultsXGB["Vote_Share"].gt(0)]


# Linear models

resultsLR = predictVoteShare(
    data = matchDummiesPos(currentYear, model1Data),
    stage1 = modLR,
    threshold = bestThLR,
    stage2 = modLR2
)
resultsLR.loc[resultsLR["Vote_Share"].gt(0)]
```