2023 Best Ball Data Bowl: The Su-PPPOR Bowl

Zach Bradlow (X: @zachbradlow)

2023-08-01

Introduction

For my 2023 Best Ball Data Bowl, I focused on one question: when I am on the clock in a Best Ball draft: who should I select with my next overall selection? While this may seem like a straightforward problem, it necessitates considering a myriad of factors, such as existing roster construction through draft capital spent, positional value, stacking ability, draft pick number, draft order, and ADP value. To further illustrate this point, here is a meme below to articulate the complexity of the problem


DRAFT THE BEST PLAYER
OPTIMIZE FOR PICK POINTS BY POSITION
MAKE A POSITIONAL DRAFT CURVE
CREATE A NEW METRIC FOR EXPECTED PICK POINTS OVER REPLACEMENT GIVEN EXISTING ROSTER CONSTRUCTION, POSITIONAL VALUE, STACKING, PICK #, ORDER, AND ADP
imgflip.com

To gain valuable insights into optimizing individual player selection, I developed a metric known as predicted pick points over replacement (PPPOR). This metric predicts the amount of points that a player will contribute to a Best Ball roster over the regular season over a player that is drafted two rounds (24 picks) later. By maximizing PPPOR, Best Ball drafters will able to dynamically optimize their team to increase their regular season advance rate and overall projected roster points.

Assumptions

In conducting this analysis, I needed to make a series of assumptions in order to create a player-agnostic dynamic strategy optimization method:

- The ADP Market is Efficient: I assume that the Average Draft Position (ADP) accurately reflects the collective knowledge and expectations of the participants, and thus represents what I am competing against in the drafts.

- Rookies and Team Changes are Priced In: I take into account that the market has already factored in the potential impact of rookies and team changes on player performances and draft positions.

- Regular Season Optimization is Easier than Playoff Optimization: I consider that focusing on optimizing my team for the regular season enables a larger sample size, which could potentially increase my competitive edge gained. However, I acknowledge that the payout structure of Best Ball is skewed towards playoff optimization in order to maximize EV - although, I am assuming that winning in the playoffs is random (and that constructing a player-neutral portfolio maximizing EPPOR will not negatively affect the chances of winning relative to a baseline win probability)

Data Cleaning:

My project specifically focuses on Best Ball Mania III data from Underdog and NFL roster data from the nflverse in the 2022 NFL season. This data cleaning process of removing erroneous teams, adding rostered players, and removing duplicate named players (ex: Michael Carter v. Michael Carter) ensures that the dataset is ready for further analysis

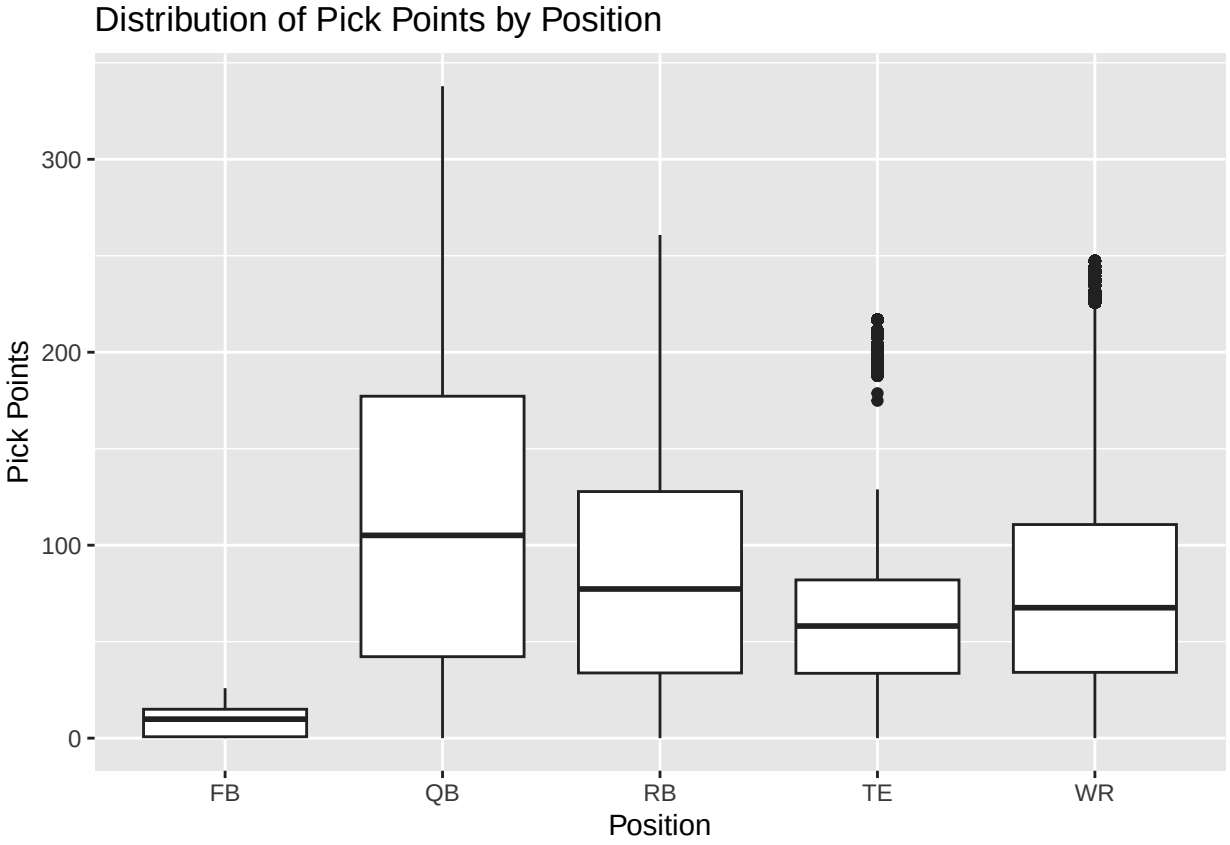
Feature Engineering

The feature engineering process in this code aims to enhance the predictive power of the dataset for modeling individual pick points in a Best Ball data analysis. Starting with the initial dataset, several new features are derived to capture essential insights into player draft strategies and positional values. The first step involves calculating the adjusted draft pick over expected (ADPOE) for each player, representing the difference between their overall_pick_number and projection_adp where positive is positive surplus value for a user. By grouping the data by tournament_entry_id and ordering it based on overall_pick_number, cumulative counts of players in each position (QB, RB, WR, TE) are computed, helping to track the team composition throughout the draft. Cumulative ADPOE is tracked by position in addition to the calculation of TotalADPOE, providing an overall measure of a team's draft ADP relative to expectation.

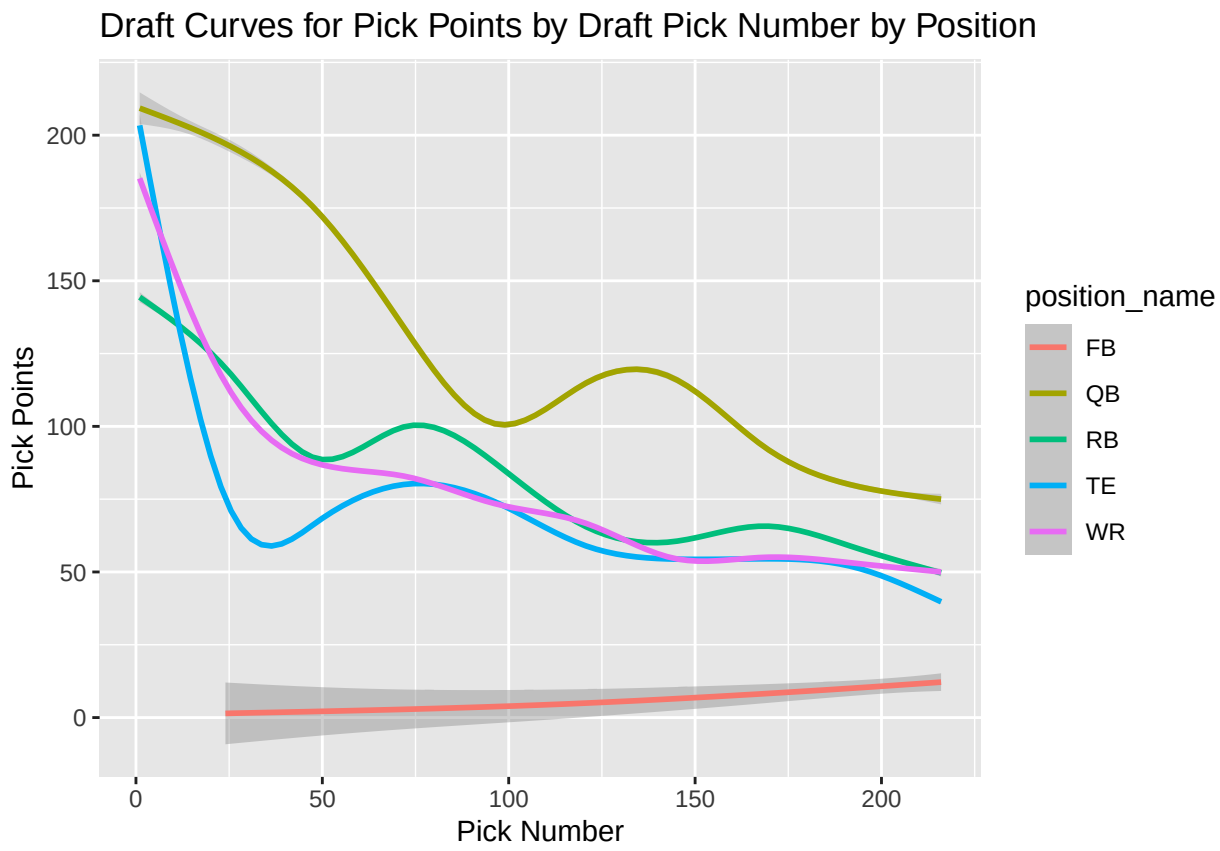
Exploratory Data Analysis

This code performs EDA containing information about pick_points, overall_pick_number, and position_name. The first graph creates side-by-side boxplots to visualize the distribution of pick_points by player position. The second graph fits linear regression curves to each position's pick_points based

on overall_pick_number using a linear function. It then predicts the pick_points for different overall_pick_number values using these regression curves. Finally, the code plots the regression curves for each position, distinguishing them by color (blue for QB, red for RB, green for WR, and purple for TE). Note that these are two simplistic ways that one could draft by maximizing their pick points in choosing many QBs - however, it does not account for the roster construction of having multiple quarterbacks




```
## `geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
```



Modeling

As a reminder, the main goal of this project is to develop an EPPOR (Expected Pick Points Over Replacement) model for optimizing draft strategy in Best Ball. To achieve this, the first model is to predict pick points in order to create an estimated draft curve based on those predictions (that accounts for other situational variables). The expected pick points compared to pick number (the estimated draft curve) more accurately represents the estimated draft capital (as a measure of points) of each player.

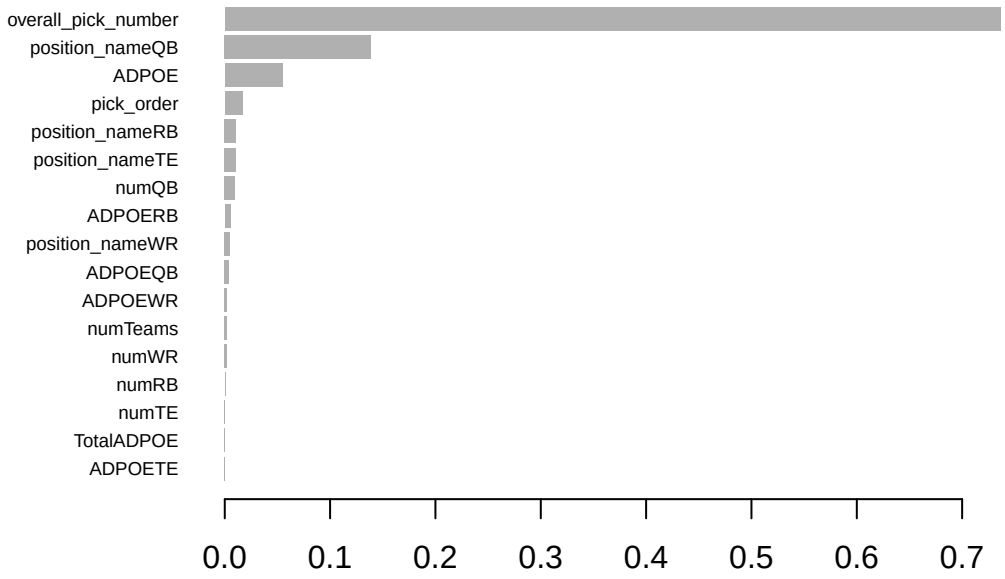
```
# Select relevant columns for training data
training_data <- best_ball_data_full %>%
    select("pick_order", "overall_pick_number", "pick_points",
           "team_pick_number", "ADPOE", "numQB", "numRB","numWR", "numTE",
           "ADPOERB", "ADPOEWR", "ADPOEQB", "ADPOETE", "ADPOEFB",
           "position_name", "TotalADPOE", "numTeams")

# Perform one-hot encoding on the training data (predictors)
dummy <- dummyVars(" ~ .", data = training_data %>% select(-pick_points))
X <- predict(dummy, newdata = training_data %>% select(-pick_points))

# Extract the target variable "pick_points" and assign it to the variable Y
Y <- training_data$pick_points

# Split data into training and validation sets (70% for training, 30% for validation)
set.seed(31700)  # For reproducibility
```

```r
split <- sample(1:nrow(training_data), 0.7 * nrow(training_data))
train_data <- training_data[split, ]
validation_data <- training_data[-split, ]

# Create DMatrix for training, validation, and all data
dtrain <- xgb.DMatrix(data = as.matrix(X[split, ]), label = Y[split])
dval <- xgb.DMatrix(data = as.matrix(X[-split, ]), label = Y[-split])
dall <- xgb.DMatrix(data = as.matrix(X), label = Y)

# Define XGBoost parameters
params <- list(
  booster = "gbtree",           # Tree-based models
  objective = "reg:squarederror",  # Default objective for regression tasks
  eval_metric = "rmse",          # Default evaluation metric for regression tasks: root mean squared er
  eta = 0.3,                    # Learning rate (Typical values: [0.01, 0.2])
  max_depth = 6,                # Maximum depth of a tree (Typical values: [3, 10])
  min_child_weight = 1,         # Minimum sum of instance weight needed in a child (Typical values: [1,
  subsample = 1,                # Subsample ratio of the training instances (Typical values: [0.5, 1])
  colsample_bytree = 1,         # Subsample ratio of columns when constructing each tree (Typical value
  gamma = 0,                    # Minimum loss reduction required to make a further partition (Typical
  lambda = 1,                   # L2 regularization term on weights (Typical values: [0, 1])
  alpha = 0,                    # L1 regularization term on weights (Typical values: [0, 1])
  num_parallel_tree = 1,        # Number of parallel trees constructed (for boosting)
  colsample_bynode = 1,         # Subsample ratio of columns when constructing each split (for tree con
  colsample_bylevel = 1,        # Subsample ratio of columns for each level (for tree construction)
  scale_pos_weight = 1,         # Balancing of positive and negative weights (for imbalanced datasets)
  base_score = 0.5,             # The initial prediction score of all instances
  seed = 0                      # Random seed for reproducibility
)

# Train the XGBoost model with early stopping
watchlist <- list(train = dtrain, validation = dval)
xgb_model <- xgb.train(params = params, data = dtrain, nrounds = 50, early_stopping_rounds = 3, watchli
```

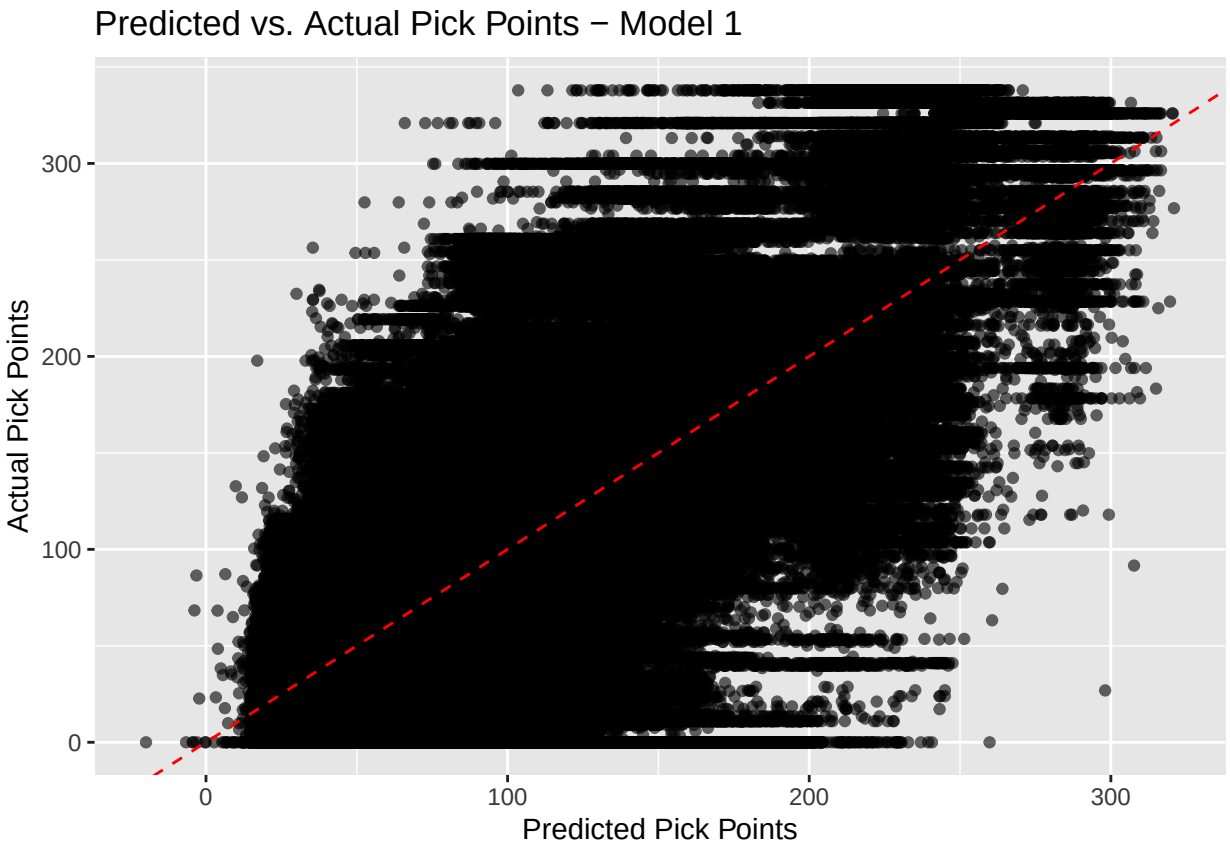
```
## [1]  train-rmse:81.237888    validation-rmse:81.164197
## Multiple eval metrics are present. Will use validation_rmse for early stopping.
## Will train until validation_rmse hasn't improved in 3 rounds.
##
## [2]  train-rmse:65.031745    validation-rmse:64.956643
## [3]  train-rmse:55.292694    validation-rmse:55.218534
## [4]  train-rmse:49.608327    validation-rmse:49.533597
## [5]  train-rmse:46.494586    validation-rmse:46.419654
## [6]  train-rmse:44.859925    validation-rmse:44.792011
## [7]  train-rmse:43.908911    validation-rmse:43.844271
## [8]  train-rmse:43.387538    validation-rmse:43.326064
## [9]  train-rmse:43.089791    validation-rmse:43.031004
## [10] train-rmse:42.909602    validation-rmse:42.853011
## [11] train-rmse:42.826080    validation-rmse:42.768306
## [12] train-rmse:42.751417    validation-rmse:42.695380
## [13] train-rmse:42.594336    validation-rmse:42.539871
## [14] train-rmse:42.493359    validation-rmse:42.439294
## [15] train-rmse:42.387268    validation-rmse:42.334105
## [16] train-rmse:42.289906    validation-rmse:42.238683
```

```
## [17] train-rmse:42.267920    validation-rmse:42.216580
## [18] train-rmse:42.247704    validation-rmse:42.197245
## [19] train-rmse:42.224602    validation-rmse:42.174797
## [20] train-rmse:42.152852    validation-rmse:42.104060
## [21] train-rmse:42.109240    validation-rmse:42.061602
## [22] train-rmse:42.096188    validation-rmse:42.049254
## [23] train-rmse:42.084708    validation-rmse:42.038544
## [24] train-rmse:42.078198    validation-rmse:42.032887
## [25] train-rmse:41.993042    validation-rmse:41.948113
## [26] train-rmse:41.966835    validation-rmse:41.921483
## [27] train-rmse:41.933748    validation-rmse:41.889411
## [28] train-rmse:41.869675    validation-rmse:41.825282
## [29] train-rmse:41.823778    validation-rmse:41.780509
## [30] train-rmse:41.796420    validation-rmse:41.753408
## [31] train-rmse:41.791565    validation-rmse:41.749353
## [32] train-rmse:41.775141    validation-rmse:41.733173
## [33] train-rmse:41.756314    validation-rmse:41.714477
## [34] train-rmse:41.716089    validation-rmse:41.674753
## [35] train-rmse:41.674628    validation-rmse:41.636137
## [36] train-rmse:41.641076    validation-rmse:41.603008
## [37] train-rmse:41.623735    validation-rmse:41.585941
## [38] train-rmse:41.573223    validation-rmse:41.536641
## [39] train-rmse:41.566294    validation-rmse:41.530885
## [40] train-rmse:41.553177    validation-rmse:41.518686
## [41] train-rmse:41.547650    validation-rmse:41.514146
## [42] train-rmse:41.542256    validation-rmse:41.509597
## [43] train-rmse:41.538618    validation-rmse:41.506698
## [44] train-rmse:41.533150    validation-rmse:41.502093
## [45] train-rmse:41.496961    validation-rmse:41.465963
## [46] train-rmse:41.479217    validation-rmse:41.450438
## [47] train-rmse:41.469910    validation-rmse:41.441261
## [48] train-rmse:41.430854    validation-rmse:41.402853
## [49] train-rmse:41.402039    validation-rmse:41.374587
## [50] train-rmse:41.376013    validation-rmse:41.349052
```

```r
# Get importance scores for the features
importance_scores <- xgb.importance(model = xgb_model)

# Plot importance scores nicely
xgb.plot.importance(importance_matrix = importance_scores)
```




```r
# Make predictions on validation data and calculate residuals
validation_data$predicted_pick_points <- predict(xgb_model, dval)
validation_data$residuals <- validation_data$predicted_pick_points - validation_data$pick_points

# Plot predictions versus actuals
ggplot(validation_data, aes(x = predicted_pick_points, y = pick_points)) +
  geom_point(alpha = 0.6) +
  geom_abline(intercept = 0, slope = 1, color = "red", linetype = "dashed") +
  labs(title = "Predicted vs. Actual Pick Points - Model 1", x = "Predicted Pick Points", y = "Actual P
```




```
best_ball_data_full$predicted_pick_points = predict(xgb_model, dall)
cat("For Model 1, the R-squared between actual and predicted points is", cor(best_ball_data_full$predic
```

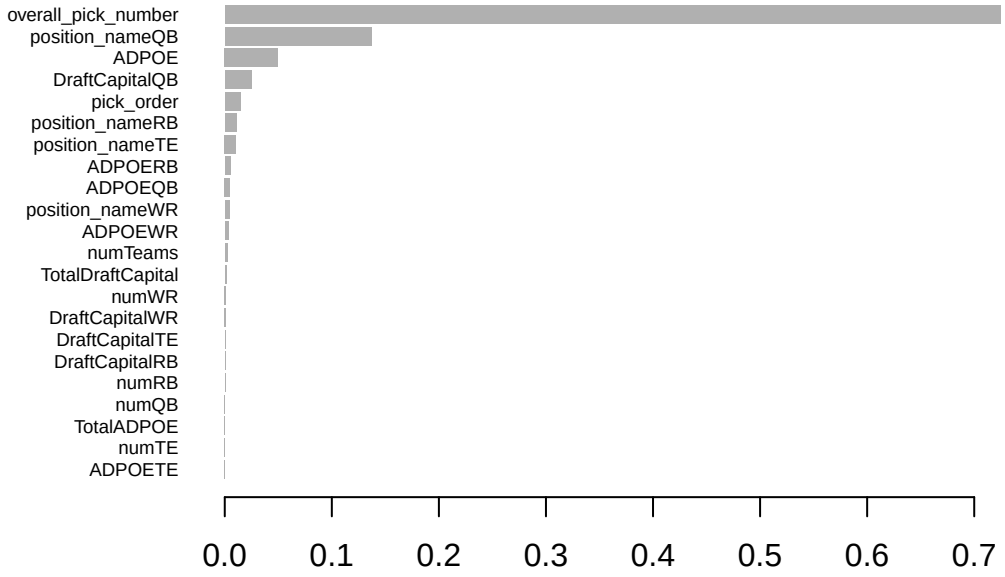
```
## For Model 1, the R-squared between actual and predicted points is 0.6081462
```

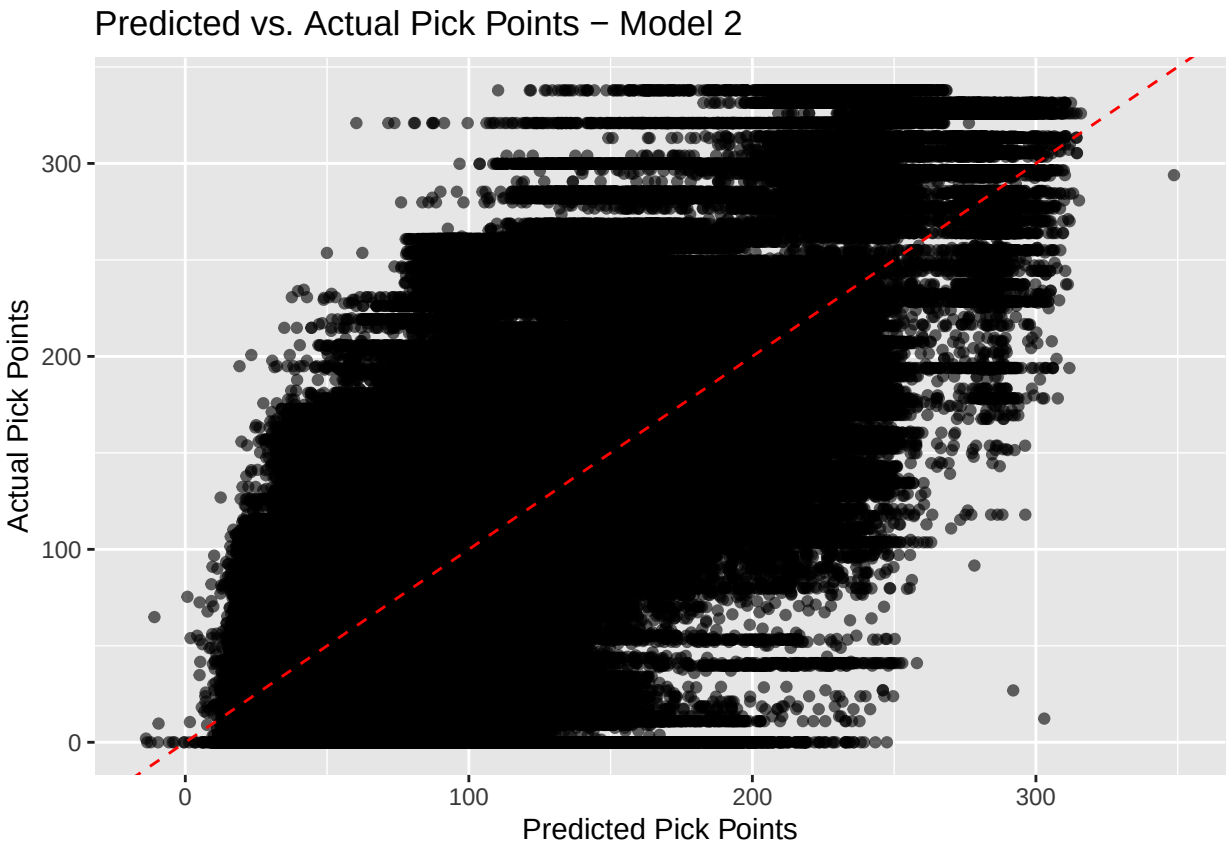
After 50 iterations of training the XGBoost model, the training RMSE is ~41 and the validation RMSE is ~41 with an R-squared of 0.61 between actual and predicted points for individuals. Note that performance stayed constant even as nrounds went up in the many thousands when tested. This shows that this model is not subject to overfitting. The most important features in this model are the pick number, whether or not a QB is being drafted, and the ADP value over expected. These predictions are subsequently used as a feature in the second model that is additionally predicting pick points at the player-level. The difference between this model and the last model is that it accounts for the draft capital used at various positions.

This code builds an XGBoost regression model (Model 2) to predict pick points in a Best Ball draft. The training data is split into a training set (70%) and a validation set (30%). The XGBoost model is trained with early stopping to optimize performance. The feature importance scores are plotted for model interpretation. Then, predictions are made on the validation set. A plot is generated to compare the predicted pick points with the actual pick points. Finally, the R-squared value is calculated to assess the model's predictive performance, indicating how well the model's predictions align with the actual pick points.

```
## [1]  train-rmse:81.157677    validation-rmse:81.084012
## Multiple eval metrics are present. Will use validation_rmse for early stopping.
## Will train until validation_rmse hasn't improved in 3 rounds.
##
## [2]  train-rmse:64.878210    validation-rmse:64.806648
```

```
## [3]  train-rmse:55.033654    validation-rmse:54.962707
## [4]  train-rmse:49.318241    validation-rmse:49.250825
## [5]  train-rmse:46.201436    validation-rmse:46.139233
## [6]  train-rmse:44.504590    validation-rmse:44.445994
## [7]  train-rmse:43.524753    validation-rmse:43.469944
## [8]  train-rmse:43.027277    validation-rmse:42.974273
## [9]  train-rmse:42.683105    validation-rmse:42.630343
## [10] train-rmse:42.521093    validation-rmse:42.472389
## [11] train-rmse:42.421255    validation-rmse:42.375435
## [12] train-rmse:42.324099    validation-rmse:42.280895
## [13] train-rmse:42.190346    validation-rmse:42.149469
## [14] train-rmse:42.147471    validation-rmse:42.107679
## [15] train-rmse:41.970220    validation-rmse:41.932156
## [16] train-rmse:41.922555    validation-rmse:41.886645
## [17] train-rmse:41.875257    validation-rmse:41.840896
## [18] train-rmse:41.780715    validation-rmse:41.746188
## [19] train-rmse:41.724192    validation-rmse:41.690539
## [20] train-rmse:41.683042    validation-rmse:41.649980
## [21] train-rmse:41.633415    validation-rmse:41.601569
## [22] train-rmse:41.609727    validation-rmse:41.579010
## [23] train-rmse:41.566954    validation-rmse:41.537496
## [24] train-rmse:41.553033    validation-rmse:41.524504
## [25] train-rmse:41.542094    validation-rmse:41.513740
## [26] train-rmse:41.531594    validation-rmse:41.504346
## [27] train-rmse:41.514823    validation-rmse:41.488819
## [28] train-rmse:41.468146    validation-rmse:41.443151
## [29] train-rmse:41.462177    validation-rmse:41.438239
## [30] train-rmse:41.458025    validation-rmse:41.436541
## [31] train-rmse:41.422060    validation-rmse:41.401955
## [32] train-rmse:41.380724    validation-rmse:41.360627
## [33] train-rmse:41.356335    validation-rmse:41.337187
## [34] train-rmse:41.336074    validation-rmse:41.317067
## [35] train-rmse:41.290091    validation-rmse:41.272100
## [36] train-rmse:41.282892    validation-rmse:41.266497
## [37] train-rmse:41.232416    validation-rmse:41.216597
## [38] train-rmse:41.200405    validation-rmse:41.184701
## [39] train-rmse:41.127529    validation-rmse:41.114758
## [40] train-rmse:41.111972    validation-rmse:41.098586
## [41] train-rmse:41.099274    validation-rmse:41.086453
## [42] train-rmse:41.004350    validation-rmse:40.992780
## [43] train-rmse:40.970136    validation-rmse:40.961975
## [44] train-rmse:40.960340    validation-rmse:40.951742
## [45] train-rmse:40.948617    validation-rmse:40.940594
## [46] train-rmse:40.927505    validation-rmse:40.920246
## [47] train-rmse:40.912313    validation-rmse:40.905861
## [48] train-rmse:40.906003    validation-rmse:40.898971
## [49] train-rmse:40.889665    validation-rmse:40.883671
## [50] train-rmse:40.876428    validation-rmse:40.871128
```


overall_pick_number
position_nameQB
ADPOE
DraftCapitalQB
pick_order
position_nameRB
position_nameTE
ADPOERB
ADPOEQB
position_nameWR
ADPOEWR
numTeams
TotalDraftCapital
numWR
DraftCapitalWR
DraftCapitalTE
DraftCapitalRB
numRB
numQB
TotalADPOE
numTE
ADPOETE
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7




```
## For Model 2, the R-squared between actual and predicted points is 0.6174208
```

After 50 iterations of training the XGBoost model, the training RMSE and validation RMSE are lower than the first model with the additional feature (note that performance stayed constant even as nrounds went up in the many thousands). The most important feature in this model is predominantly the predicted pick points which comprises the other variables. Now that these predictions have been made, we can now aggregate the individual pick point estimates to find who the best expected teams are.

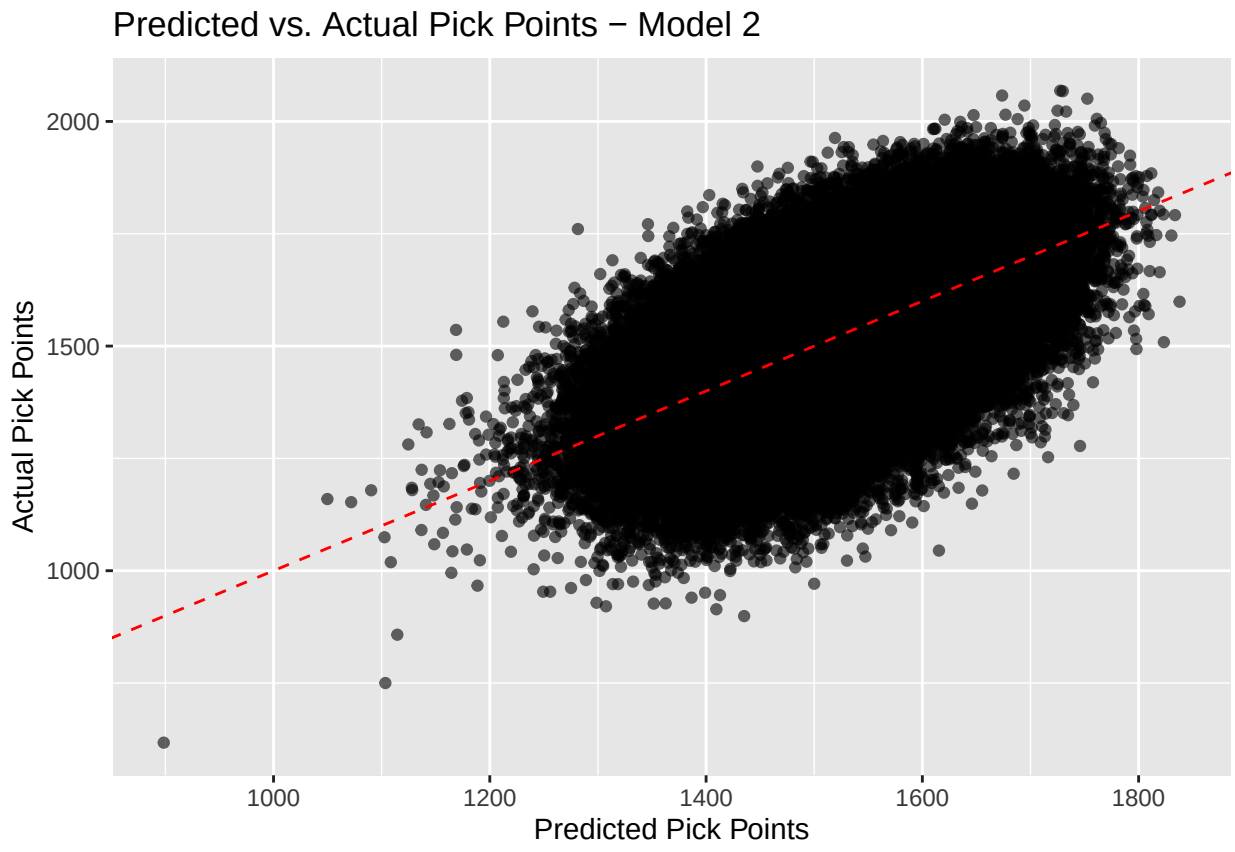
```
# best_ball_data_full$predictions <- predict(xgb_model, dtrain)
to_join = best_ball_data_full %>%
  left_join(training_data) %>%
  filter(team_pick_number == 18) %>%
  select(tournament_entry_id,  "numQB", "numRB","numWR", "numTE", "ADPOERB",
         "ADPOEWR", "ADPOEQB", "ADPOETE", "ADPOEFB", "TotalADPOE", "pick_order",
         "playoff_team","numTeams","DraftCapitalQB","DraftCapitalRB",
         "DraftCapitalWR","DraftCapitalTE","DraftCapitalFB","TotalDraftCapital")
```

```
## Joining with `by = join_by(pick_order, overall_pick_number, team_pick_number,
## pick_points, ADPOE, numQB, numRB, numWR, numTE, numTeams, ADPOERB, ADPOEWR,
## ADPOEQB, ADPOETE, ADPOEFB, position_name, TotalADPOE)`

## Warning in left_join(., training_data): Detected an unexpected many-to-many relationship between `x`
## i Row 1 of `x` matches multiple rows in `y`.
## i Row 9 of `y` matches multiple rows in `x`.
## i If a many-to-many relationship is expected, set `relationship =
##   "many-to-many"` to silence this warning.
```

```
training_teams <- best_ball_data_full %>%
  group_by(tournament_entry_id) %>%
  summarise(roster_points = max(roster_points),
            predicted_points = sum(predicted_pick_points)) %>%
  left_join(to_join)
```

```
## Joining with `by = join_by(tournament_entry_id)`
```

```
# Plot predictions versus actuals
ggplot(training_teams, aes(x = predicted_points, y = roster_points)) +
  geom_point(alpha = 0.6) +
  geom_abline(intercept = 0, slope = 1, color = "red", linetype = "dashed") +
  labs(title = "Predicted vs. Actual Pick Points - Model 2",
       x = "Predicted Pick Points", y = "Actual Pick Points")
```




```
cat("For the team-level of Model 2, the R-squared between actual and predicted points is", cor(training
```

```
## For the team-level of Model 2, the R-squared between actual and predicted points is 0.3007153
```

```
kable(head(training_teams %>% arrange(desc(predicted_points))))
```

tournament_entry_id	roster_points	predicted_points	numQB	numRB	numWR	numTE	A
6287f35d-8de7-49b2-b150-3c79c34d7090	1598.74	1837.878	2	6	8	2	
432e67dd-fc23-4eb1-b1ac-2acf62635f67	1791.52	1833.629	2	5	9	2	
0c20056b-e8c8-48bd-bd18-a4983eaaa92f	1746.10	1830.313	2	6	8	2	
fd9370c4-34b1-4e5d-a11d-25d8335797fd	1508.72	1823.367	4	6	5	3	
406718ae-9108-4343-8078-e9a2c799ed37	1792.96	1822.909	2	6	8	2	
4c273fb4-d329-4985-afd0-34a19b8ee8ea	1801.30	1819.542	2	6	7	3	

Additionally, it is interesting to note that the r-squared is significantly lower for the combined team model than it is for the individual (approximately half). This is relatively surprising given you would think that more players would provide a more accurate estimate/balance out any outliers.

In order to analyze the regular season advance rate of rosters, I had to adjust the playoff team variable to make it the top two highest scoring teams per league throughout weeks 1-14 as defined by roster points.

```
# Extract unique tournament_entry_id and draft_id pairs from the original dataset
unique_teams <- best_ball_data_full %>%
  select(tournament_entry_id, draft_id) %>%
  unique()

# Merge the training data with the unique tournament_entry_id and draft_id pairs
training_teams <- training_teams %>%
  left_join(unique_teams) %>%

  # Group the data by draft_id for further processing
  group_by(draft_id) %>%

  # Arrange the data within each group in descending order of roster_points
  arrange(desc(roster_points)) %>%

  # Create a new column 'playoff_team' that marks top 2 teams in each draft_id as 1 (playoff team) and
  mutate(playoff_team = if_else(row_number() <= 2, 1, 0)) %>%

  # Ungroup the data
  ungroup()
```

```
## Joining with `by = join_by(tournament_entry_id)`
```

This code builds a playoff classification model using XGBoost. It starts by performing one-hot encoding on the training data to convert categorical variables into binary features. The XGBoost model is then trained with binary logistic regression objective to predict playoff teams. Early stopping is applied to optimize the model's performance. The model's predictions are made on the validation set, and a confusion matrix is generated to evaluate the model's classification performance. The confusion matrix provides insights into the number of true positives, true negatives, false positives, and false negatives, which help assess the model's accuracy in classifying playoff teams.

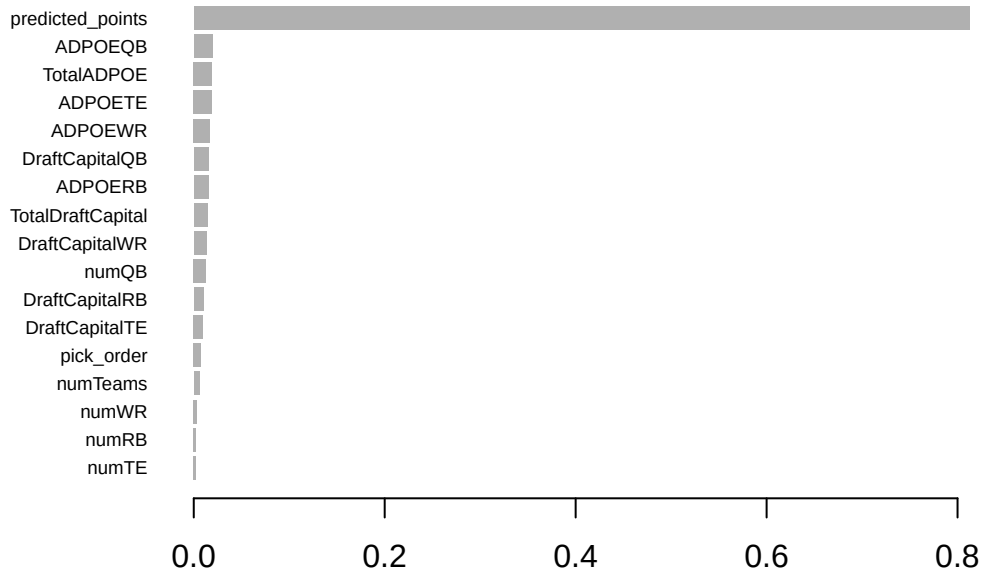
```
# Perform one-hot encoding on the training data (predictors)
dummy <- dummyVars(" ~ .", data = training_teams %>% select(-c("roster_points", "playoff_team", "tournar
X <- predict(dummy, newdata = training_teams %>% select(-c("roster_points", "playoff_team", "tournament_

# Extract the target variable "playoff_team" and assign it to the variable Y
Y <- training_teams$playoff_team

# Split data into training and validation sets (70% for training, 30% for validation)
```

```r
set.seed(123)  # For reproducibility
split <- sample(1:nrow(training_teams), 0.7 * nrow(training_teams))
train_data <- training_teams[split, ]
validation_data <- training_teams[-split, ]

# Create dummy variables for training data
dummy <- dummyVars(" ~ .", data = train_data %>% select(-c("roster_points", "playoff_team", "tournament_
X_train <- predict(dummy, newdata = train_data %>% select(-c("roster_points", "playoff_team", "tournamen

# Extract the target variable "playoff_team" for training set and assign it to the variable Y_train
Y_train <- train_data$playoff_team

# Create dummy variables for validation data
X_val <- predict(dummy, newdata = validation_data %>% select(-c("roster_points", "playoff_team", "tourna

# Extract the target variable "playoff_team" for the validation set and assign it to the variable Y_val
Y_val <- validation_data$playoff_team

# Convert data to DMatrix format (required for xgboost)
dtrain <- xgb.DMatrix(data = as.matrix(X_train), label = Y_train)
dval <- xgb.DMatrix(data = as.matrix(X_val), label = Y_val)

params <- list(
  objective = "binary:logistic",
  booster = "gbtree",
  eta = 0.3,
  max_depth = 10,
  min_child_weight = 10,
  subsample = 1,
  colsample_bytree = 1,
  gamma = 0,
  lambda = 1,
  alpha = 1,
  eval_metric = "error"
)

# Train the XGBoost model with early stopping
xgb_model <- xgb.train(params = params, data = dtrain, nrounds = 100,
                       early_stopping_rounds = 3,
                       watchlist = list(train = dtrain, validation = dval))
```

```
## [1]  train-error:0.157453    validation-error:0.162178
## Multiple eval metrics are present. Will use validation_error for early stopping.
## Will train until validation_error hasn't improved in 3 rounds.
##
## [2]  train-error:0.155815    validation-error:0.162043
## [3]  train-error:0.155088    validation-error:0.161928
## [4]  train-error:0.154806    validation-error:0.161470
## [5]  train-error:0.154521    validation-error:0.161459
## [6]  train-error:0.154561    validation-error:0.161543
## [7]  train-error:0.154557    validation-error:0.161209
## [8]  train-error:0.154320    validation-error:0.161438
## [9]  train-error:0.154257    validation-error:0.161501
```

```
## [10] train-error:0.153999    validation-error:0.161699
## Stopping. Best iteration:
## [7]  train-error:0.154557    validation-error:0.161209
```

```
importance_scores <- xgb.importance(model = xgb_model)

# Optionally, plot the importance scores
xgb.plot.importance(importance_matrix = importance_scores)
```




```
y_pred_class <- ifelse(predict(xgb_model, as.matrix(X_val)) >= 0.5, 1, 0)

# Create a confusion matrix
confusionMatrix(data = factor(y_pred_class), reference = factor(Y_val))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 78490 13922
##          1  1557  2049
##
##                Accuracy : 0.8388
##                  95% CI : (0.8364, 0.8411)
##     No Information Rate : 0.8337
##     P-Value [Acc > NIR] : 9.502e-06
```

```
##
##                   Kappa : 0.1577
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9805
##             Specificity : 0.1283
##          Pos Pred Value : 0.8493
##          Neg Pred Value : 0.5682
##              Prevalence : 0.8337
##          Detection Rate : 0.8175
##    Detection Prevalence : 0.9624
##       Balanced Accuracy : 0.5544
##
##        'Positive' Class : 0
##
```

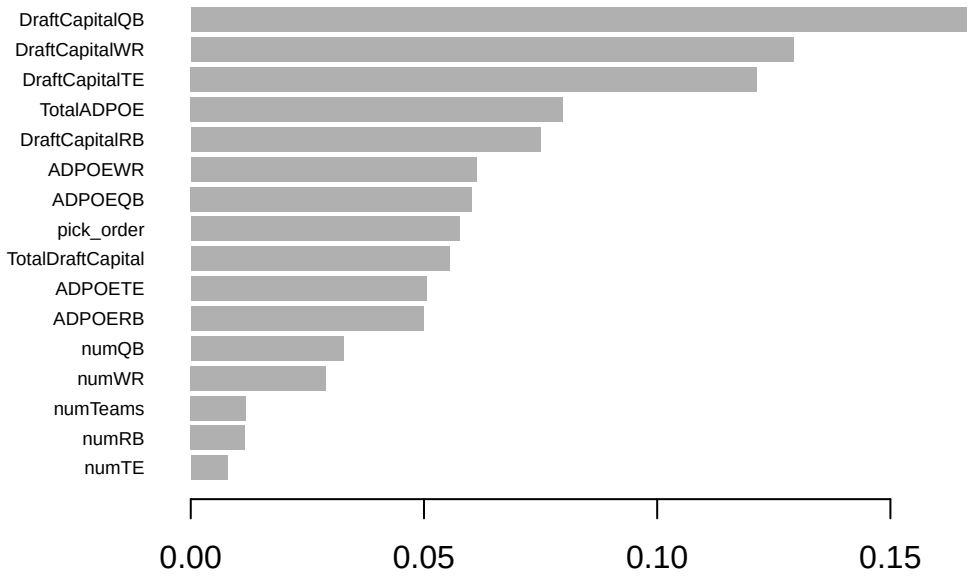
In Model Four, the objective is to classify teams into "playoff_team" or "non-playoff_team" categories using XGBoost classification. Comparing Model Four to Model Three, both are classification models targeting "playoff_team" prediction. However, Model Four differs in terms of feature engineering, as it does not include our output variable from Model 2 "predicted_points" that was present in Model Three. This could potentially decrease the model's performance. The overall accuracy of the confusion matrix in Model Four is slightly worse than Model Three. The confidence interval for Model Four's accuracy includes the null hypothesis, indicating that the model may not perform significantly better than random guessing.

```
# Perform one-hot encoding on the training data
dummy <- dummyVars(" ~ .", data = training_teams %>%
                     select(-c("roster_points", "playoff_team", "tournament_entry_id","draft_id","predi
X <- predict(dummy, newdata = training_teams %>%
               select(-c("roster_points", "playoff_team", "tournament_entry_id",
                         "draft_id","predicted_points")))

# Extract the target variable "playoff_team" and assign it to the variable Y
Y <- training_teams$playoff_team

# Split data into training and validation sets (70% for training, 30% for validation)
set.seed(123)  # For reproducibility
split <- sample(1:nrow(training_teams), 0.7 * nrow(training_teams))
train_data <- training_teams[split, ]
validation_data <- training_teams[-split, ]

# Create dummy variables for training data
dummy <- dummyVars(" ~ .", data = train_data %>% select(-c("roster_points", "playoff_team", "tournament_
X_train <- predict(dummy, newdata = train_data %>% select(-c("roster_points", "playoff_team", "tournamen

# Extract the target variable "playoff_team" for training set and assign it to the variable Y_train
Y_train <- train_data$playoff_team

# Create dummy variables for validation data
X_val <- predict(dummy, newdata = validation_data %>% select(-c("roster_points", "playoff_team", "tourna

# Extract the target variable "playoff_team" for the validation set and assign it to the variable Y_val
Y_val <- validation_data$playoff_team
```

```r
# Convert data to DMatrix format (required for xgboost)
dtrain <- xgb.DMatrix(data = as.matrix(X_train), label = Y_train)
dval <- xgb.DMatrix(data = as.matrix(X_val), label = Y_val)

params <- list(
  objective = "binary:logistic",
  booster = "gbtree",
  eta = 0.3,
  max_depth = 10,
  min_child_weight = 10,
  subsample = 1,
  colsample_bytree = 1,
  gamma = 0,
  lambda = 1,
  alpha = 1,
  eval_metric = "error"
)

# Train the XGBoost model with early stopping
xgb_model <- xgb.train(params = params, data = dtrain, nrounds = 100,
                       early_stopping_rounds = 3,
                       watchlist = list(train = dtrain, validation = dval))
```

```
## [1]  train-error:0.164872    validation-error:0.168427
## Multiple eval metrics are present. Will use validation_error for early stopping.
## Will train until validation_error hasn't improved in 3 rounds.
##
## [2]  train-error:0.164274    validation-error:0.167219
## [3]  train-error:0.164157    validation-error:0.166927
## [4]  train-error:0.163952    validation-error:0.166969
## [5]  train-error:0.164086    validation-error:0.166927
## [6]  train-error:0.163863    validation-error:0.166740
## [7]  train-error:0.163640    validation-error:0.166740
## [8]  train-error:0.163479    validation-error:0.166740
## [9]  train-error:0.163403    validation-error:0.166885
## Stopping. Best iteration:
## [6]  train-error:0.163863    validation-error:0.166740
```

```r
importance_scores <- xgb.importance(model = xgb_model)

# Optionally, plot the importance scores
xgb.plot.importance(importance_matrix = importance_scores)
```




```
y_pred_class <- ifelse(predict(xgb_model, as.matrix(X_val)) >= 0.5, 1, 0)

# Create a confusion matrix
confusionMatrix(data = factor(y_pred_class), reference = factor(Y_val))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 79795 15758
##          1   252   213
##
##                Accuracy : 0.8333
##                  95% CI : (0.8309, 0.8356)
##     No Information Rate : 0.8337
##     P-Value [Acc > NIR] : 0.6343
##
##                   Kappa : 0.0167
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.99685
##             Specificity : 0.01334
##          Pos Pred Value : 0.83509
##          Neg Pred Value : 0.45806
```

```
##              Prevalence : 0.83367
##          Detection Rate : 0.83104
##    Detection Prevalence : 0.99516
##       Balanced Accuracy : 0.50509
##
##        'Positive' Class : 0
##
```

In Model Five, the focus is on using XGBoost regression to predict "pick_points" in a draft scenario. Model Five plays a crucial role in draft optimization, as it provides a means to predict the pick points for each player during the draft. By using XGBoost regression, teams can make informed decisions on player selections, taking into account the draft capital spent, positional value, stacking ability, draft pick number, draft order, and ADP value as initially laid out in the beginning. Model Five serves as a promising foundation for enhancing the draft selection process and can be refined further to achieve an even more accurate and effective draft optimizer for Best Ball leagues.

```r
# Select relevant columns for training data
# Perform one-hot encoding on the training data (predictors)
final_data <- training_data %>%
  mutate
dummy <- dummyVars(" ~ .", data = training_data %>% select(-pick_points))
X <- predict(dummy, newdata = training_data %>% select(-pick_points))

# Extract the target variable "pick_points" and assign it to the variable Y
Y <- training_data$pick_points

# Split data into training and validation sets (70% for training, 30% for validation)
set.seed(31700)  # For reproducibility
split <- sample(1:nrow(training_data), 0.7 * nrow(training_data))
train_data <- training_data[split, ]
validation_data <- training_data[-split, ]

# Create DMatrix for training, validation, and all data
dtrain <- xgb.DMatrix(data = as.matrix(X[split, ]), label = Y[split])
dval <- xgb.DMatrix(data = as.matrix(X[-split, ]), label = Y[-split])
dall <- xgb.DMatrix(data = as.matrix(X), label = Y)

# Define XGBoost parameters
params <- list(
  booster = "gbtree",           # Tree-based models
  objective = "reg:squarederror",  # Default objective for regression tasks
  eval_metric = "rmse",          # Default evaluation metric for regression tasks: root mean squared er
  eta = 0.3,                    # Learning rate (Typical values: [0.01, 0.2])
  max_depth = 6,                # Maximum depth of a tree (Typical values: [3, 10])
  min_child_weight = 1,         # Minimum sum of instance weight needed in a child (Typical values: [1,
  subsample = 1,                # Subsample ratio of the training instances (Typical values: [0.5, 1])
  colsample_bytree = 1,         # Subsample ratio of columns when constructing each tree (Typical value
  gamma = 0,                    # Minimum loss reduction required to make a further partition (Typical
  lambda = 1,                   # L2 regularization term on weights (Typical values: [0, 1])
  alpha = 1,                    # L1 regularization term on weights (Typical values: [0, 1])
  num_parallel_tree = 1,        # Number of parallel trees constructed (for boosting)
  colsample_bynode = 1,         # Subsample ratio of columns when constructing each split (for tree con
  colsample_bylevel = 1,        # Subsample ratio of columns for each level (for tree construction)
  scale_pos_weight = 1,         # Balancing of positive and negative weights (for imbalanced datasets)
```

```r
  base_score = 0.5,             # The initial prediction score of all instances
  seed = 0                      # Random seed for reproducibility
)

# Train the XGBoost model with early stopping
watchlist <- list(train = dtrain, validation = dval)
xgb_model <- xgb.train(params = params, data = dtrain, nrounds = 100,
                       early_stopping_rounds = 3, watchlist = watchlist, verbose = 1)
```

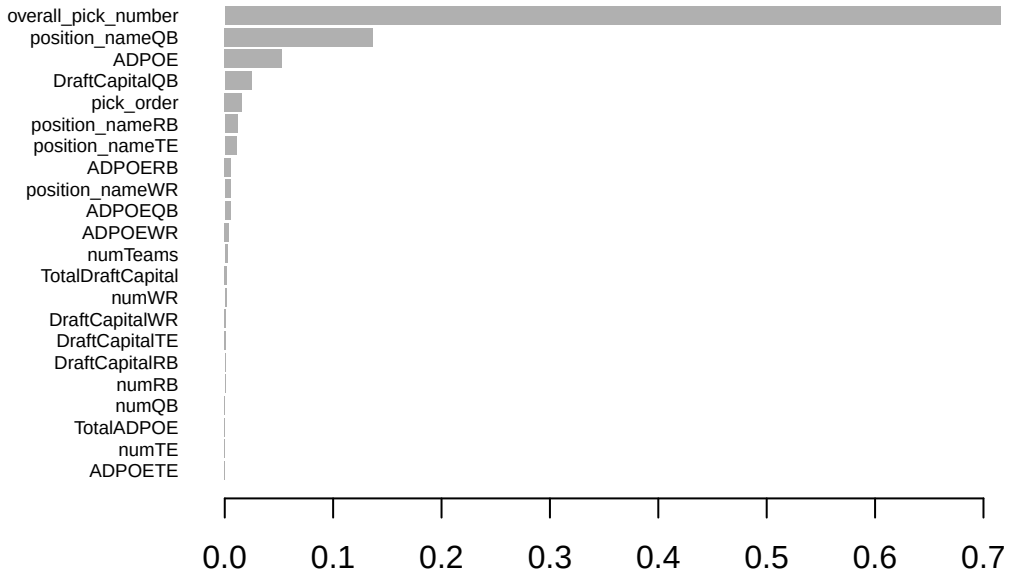
```
## Warning in xgb.train(params = params, data = dtrain, nrounds = 100,
## early_stopping_rounds = 3, : xgb.train: ‘seed‘ is ignored in R package.  Use
## ‘set.seed()‘ instead.
```

```
## [1]  train-rmse:81.157682    validation-rmse:81.084017
## Multiple eval metrics are present. Will use validation_rmse for early stopping.
## Will train until validation_rmse hasn't improved in 3 rounds.
##
## [2]  train-rmse:64.878218    validation-rmse:64.806656
## [3]  train-rmse:55.033662    validation-rmse:54.962715
## [4]  train-rmse:49.318247    validation-rmse:49.250832
## [5]  train-rmse:46.201442    validation-rmse:46.139239
## [6]  train-rmse:44.504595    validation-rmse:44.445999
## [7]  train-rmse:43.524758    validation-rmse:43.469948
## [8]  train-rmse:43.027281    validation-rmse:42.974277
## [9]  train-rmse:42.683109    validation-rmse:42.630347
## [10] train-rmse:42.521098    validation-rmse:42.472393
## [11] train-rmse:42.421259    validation-rmse:42.375439
## [12] train-rmse:42.324104    validation-rmse:42.280900
## [13] train-rmse:42.190351    validation-rmse:42.149473
## [14] train-rmse:42.147478    validation-rmse:42.107684
## [15] train-rmse:41.970228    validation-rmse:41.932162
## [16] train-rmse:41.922564    validation-rmse:41.886651
## [17] train-rmse:41.875266    validation-rmse:41.840902
## [18] train-rmse:41.780725    validation-rmse:41.746196
## [19] train-rmse:41.724203    validation-rmse:41.690547
## [20] train-rmse:41.683063    validation-rmse:41.649925
## [21] train-rmse:41.633437    validation-rmse:41.601515
## [22] train-rmse:41.609748    validation-rmse:41.578956
## [23] train-rmse:41.566978    validation-rmse:41.537444
## [24] train-rmse:41.553061    validation-rmse:41.524457
## [25] train-rmse:41.542123    validation-rmse:41.513693
## [26] train-rmse:41.531623    validation-rmse:41.504299
## [27] train-rmse:41.516127    validation-rmse:41.490259
## [28] train-rmse:41.468887    validation-rmse:41.444069
## [29] train-rmse:41.462640    validation-rmse:41.439963
## [30] train-rmse:41.456768    validation-rmse:41.435079
## [31] train-rmse:41.432665    validation-rmse:41.413250
## [32] train-rmse:41.406072    validation-rmse:41.386936
## [33] train-rmse:41.357572    validation-rmse:41.338072
## [34] train-rmse:41.327583    validation-rmse:41.307220
## [35] train-rmse:41.304205    validation-rmse:41.284206
## [36] train-rmse:41.269338    validation-rmse:41.249525
## [37] train-rmse:41.209720    validation-rmse:41.191560
```

```
## [38] train-rmse:41.170637    validation-rmse:41.152505
## [39] train-rmse:41.158790    validation-rmse:41.140845
## [40] train-rmse:41.137066    validation-rmse:41.120276
## [41] train-rmse:41.128440    validation-rmse:41.112851
## [42] train-rmse:41.090360    validation-rmse:41.075745
## [43] train-rmse:41.050330    validation-rmse:41.038352
## [44] train-rmse:41.041212    validation-rmse:41.029261
## [45] train-rmse:41.025010    validation-rmse:41.015710
## [46] train-rmse:41.007478    validation-rmse:40.999202
## [47] train-rmse:40.998402    validation-rmse:40.991245
## [48] train-rmse:40.953174    validation-rmse:40.946620
## [49] train-rmse:40.941450    validation-rmse:40.935145
## [50] train-rmse:40.932324    validation-rmse:40.927092
## [51] train-rmse:40.929355    validation-rmse:40.924842
## [52] train-rmse:40.923432    validation-rmse:40.919765
## [53] train-rmse:40.915884    validation-rmse:40.913714
## [54] train-rmse:40.899243    validation-rmse:40.898939
## [55] train-rmse:40.891894    validation-rmse:40.892224
## [56] train-rmse:40.865123    validation-rmse:40.866794
## [57] train-rmse:40.852225    validation-rmse:40.855811
## [58] train-rmse:40.843605    validation-rmse:40.847272
## [59] train-rmse:40.830336    validation-rmse:40.835442
## [60] train-rmse:40.805592    validation-rmse:40.810139
## [61] train-rmse:40.801412    validation-rmse:40.806760
## [62] train-rmse:40.785414    validation-rmse:40.792791
## [63] train-rmse:40.760020    validation-rmse:40.767159
## [64] train-rmse:40.744496    validation-rmse:40.752091
## [65] train-rmse:40.733823    validation-rmse:40.742226
## [66] train-rmse:40.702057    validation-rmse:40.713379
## [67] train-rmse:40.628343    validation-rmse:40.643007
## [68] train-rmse:40.619217    validation-rmse:40.634238
## [69] train-rmse:40.612553    validation-rmse:40.627974
## [70] train-rmse:40.595095    validation-rmse:40.611551
## [71] train-rmse:40.580936    validation-rmse:40.599108
## [72] train-rmse:40.560304    validation-rmse:40.579514
## [73] train-rmse:40.552498    validation-rmse:40.572713
## [74] train-rmse:40.546336    validation-rmse:40.567483
## [75] train-rmse:40.525549    validation-rmse:40.547255
## [76] train-rmse:40.515342    validation-rmse:40.539070
## [77] train-rmse:40.513611    validation-rmse:40.538475
## [78] train-rmse:40.492981    validation-rmse:40.518551
## [79] train-rmse:40.484088    validation-rmse:40.509917
## [80] train-rmse:40.449511    validation-rmse:40.477402
## [81] train-rmse:40.442097    validation-rmse:40.469882
## [82] train-rmse:40.435858    validation-rmse:40.463851
## [83] train-rmse:40.427260    validation-rmse:40.456145
## [84] train-rmse:40.426543    validation-rmse:40.455914
## [85] train-rmse:40.414174    validation-rmse:40.444341
## [86] train-rmse:40.409510    validation-rmse:40.440555
## [87] train-rmse:40.406493    validation-rmse:40.439063
## [88] train-rmse:40.393229    validation-rmse:40.426718
## [89] train-rmse:40.384788    validation-rmse:40.419022
## [90] train-rmse:40.374803    validation-rmse:40.409611
## [91] train-rmse:40.366728    validation-rmse:40.401985
```

```
## [92] train-rmse:40.356665    validation-rmse:40.392399
## [93] train-rmse:40.324758    validation-rmse:40.362163
## [94] train-rmse:40.295296    validation-rmse:40.333713
## [95] train-rmse:40.280761    validation-rmse:40.321093
## [96] train-rmse:40.222022    validation-rmse:40.262635
## [97] train-rmse:40.215563    validation-rmse:40.257009
## [98] train-rmse:40.197937    validation-rmse:40.239839
## [99] train-rmse:40.189989    validation-rmse:40.231998
## [100]    train-rmse:40.178371    validation-rmse:40.220793
```

```r
# Get importance scores for the features
importance_scores <- xgb.importance(model = xgb_model)

# Plot importance scores nicely
xgb.plot.importance(importance_matrix = importance_scores)
```



Best Ball Mania IV Draft Example

With a hypothetical Best Ball draft, I decided to try out this approach in specifically predicting my PPP for all of the players currently on Underdog given my roster composition at the time of the pick, what selection it was, who was on board, my stacking thus far, etc.

```
# Read in Underdog 2023 ADP with the pre-loaded variables to save time done in Excel
underdog_R1 = read.csv("underdog_round_1.csv")

# Add a new column for "position_nameFB" and initialize it to 0
underdog_R1['position_nameFB'] = 0

# Prepare a replacement dataset to simulate adding new players to the draft
underdog_R1_replacement = underdog_R1 %>%
  mutate(overall_pick_number = overall_pick_number + 24,
         team_pick_number = team_pick_number + 2,
         ADPOE = ADPOE - 24)

# One-hot encode the original and replacement datasets
dummy <- dummyVars(" ~ .", data = underdog_R1 %>%
                     select(-c("selected","team","player_name","projection_adp")))
dummy_replacement <- dummyVars(" ~ .", data = underdog_R1_replacement %>% select(-c("selected","team","
underdog_X <- predict(dummy, newdata = underdog_R1 %>%
                        select(-c("selected","team","player_name","projection_adp")))
underdog_X_replacement <- predict(dummy_replacement, newdata = underdog_R1_replacement %>%
                        select(-c("selected","team","player_name","projection_adp")))

# Define the desired order of columns for the final dataset
desired_order <- c("pick_order", "overall_pick_number", "team_pick_number",
                   "ADPOE", "numQB", "numRB", "numWR", "numTE", "ADPOERB",
                   "ADPOEWR", "ADPOEQB", "ADPOETE", "ADPOEFB", "position_nameFB",
                   "position_nameQB", "position_nameRB", "position_nameTE", "position_nameWR",
                   "TotalADPOE", "numTeams", "DraftCapitalQB", "DraftCapitalRB",
                   "DraftCapitalWR", "DraftCapitalTE", "DraftCapitalFB", "TotalDraftCapital")

# Reorder the columns in underdog_X and underdog_X_replacement according to the desired order
underdog_X <- underdog_X[, desired_order]
underdog_X_replacement <- underdog_X_replacement[, desired_order]

# Make predictions using the XGBoost model for both the original and replacement datasets
predictions = predict(xgb_model, newdata = as.matrix(underdog_X))
predictions_replacement = predict(xgb_model, newdata = as.matrix(underdog_X_replacement))

# Calculate the PPPOR (Predicted Pick Points over Replacement) for each player in the draft
underdog_R1$PPPOR = predictions - predictions_replacement

# Filter the players who are not yet selected, arrange them by descending PPPOR,
# select player_name and PPPOR columns, and display the top 5 potential draft targets
underdog_R1 %>%
  filter(selected == 0) %>%
  arrange(desc(PPPOR)) %>%
  select(player_name, PPPOR) %>%
  head(5)
```

```
##       player_name    PPPOR
## 1    Travis Kelce 138.0274
## 2   Austin Ekeler 110.0290
## 3 Michael Pittman 108.0803
## 4      Mike Evans 108.0803
```

```
## 5    Stefon Diggs 105.1179
```

Using the trained XGBoost model, this code predicts the pick points for both the current player and its replacement player (2 rounds later), calculates the PPPOR metric, and identifies the top 5 potential draft targets based on their PPPOR values. This approach allows for analyzing player values and potentially uncovering under-the-radar players who may provide high PPPOR during the draft selection process. In this case, the model recommends drafting Travis Kelce at #6 overall, though optimizing for PPP would tell you to take a QB. This can then be done for subsequent rounds to provide optimized decision making, ideally in a slow Best Ball format.

Conclusion

In conclusion, the Underdog Best Ball Bowl project has been a remarkable journey that culminated in the creation and exploration of a novel metric called PPPOR (Predicted Pick Points over Replacement). This metric proved to be a valuable addition to the realm of fantasy football and, specifically, within the Best Ball format. PPPOR allowed for a more accurate assessment of player potential within a draft while remaining player-agnostic, considering both their market value and other covariates. The classification models, which aimed to predict playoff team status, showcased the importance of considering positional differences and draft capital in team composition. The continuous XGBoost regression models demonstrated the significance of ADP value in predicting pick points. Additionally, the calibration plots highlighted the model's ability to make reliable probability estimates.

The investigation of feature importance underscored the critical role of projections, draft capital, and position-specific ADPOE values in player selection. The concept of ADPOE itself, as well as its further enhancement through positional adjustments, proved to be a powerful tool for evaluating player performance and value in Best Ball leagues.

PPPOR's potential applications extend far beyond the Underdog Best Ball Bowl project. In the broader context of fantasy football, PPPOR can serve as a powerful metric to aid fantasy managers in player selection during drafts. It provides an objective evaluation of players' predicted points relative to their expected draft positions, offering a comprehensive view of player value.

Limitations and Future Research

One area where the project could be enhanced is hyperparameter tuning. Due to time and computational constraints, we used random search to find optimal hyperparameters for our models. While this approach provided satisfactory results, more exhaustive tuning methods, such as grid search or Bayesian optimization, could potentially lead to even better model performance.Another area of potential improvement is nuanced stacking feature engineering. While we successfully employed a simple stacking technique by taking the number of players on duplicate teams, it does not account for the nuances of a QB-WR-WR stack v. a RB-RB-TE stack.

In the future, this project would greatly benefit from a UI/website to make it easier for users to upload their latest draft/ADP/who has been selected. Additionally, I would have liked to investigate the impact of playoffs on expected value rather than purely advance rates. Moreover, there could be a way to allow player grades/influence make its way into a model rather than being purely player-agnostic. Other areas that could be interesting to look into would be bye week optimization, portfolio theory with ownership rates/risk management, and late-season optimization around trying to field the worst team possible that will reach the finals so that you have a large unowned player base.

Thank You to Underdog, Fantasy Data Pros, OpenAI + Google, and Mike Leone for putting on this competition and helping me get up to speed through reading/watching materials.