# “Moneyball” Project

Table of Contents:

**In the movie *Moneyball*, GM & Assistant GM (Brad Pitt & Jonah Hill) are tasked with optimizing their roster with undervalued players on a limited budget. They did this by using Sabermetrics in analyzing and scouting players.
This project was inspired by the film.

<u>Goal:</u>
Develop a selection of models which accurately predict a player’s runs scored per game based on player characteristics such as games played, RBIs, Batting Average, Stolen Bases & other variables.
From there, determine what features are driving a player’s runs per game (RPG). Obtain key insights from the models which would be beneficial in player evaluation & scouting.

---

<u>Summary:</u>
Based on the models developed, the Elastic Net Model most accurately predicted a player’s runs per game (RPG) – although all models were similar in their evaluation scores. Please see the table at the bottom of the document for the predicted outcomes alongside the actual values.

<u>High-Level:</u>
To maximize runs per game, scouts should focus on younger players with higher OPS & slugging scores, who also have the propensity to steal bases.

Scouts should avoid players who have low RBI & hit values. Scouts should refrain from using batting average & intentional bases on balls as evaluation metrics.

Concerning feature interactions based on the terms above:

- A higher RBI score is associated with a higher OPS score, which positively drives a player’s runs per game.

- A player with a below average slugging percentage tends to have less stolen bases, which is associated with very low/negative impacts on runs per game

Target players with the following characteristics:
- Young
- Good OPS/OPS+ scores
- Good Slugging scores
- Tendency to steal bases
- Above average RBI scores

Avoid players with the following characteristics:
- Older
- Below average slugging percentage
- Below average RBIs & Hits scores

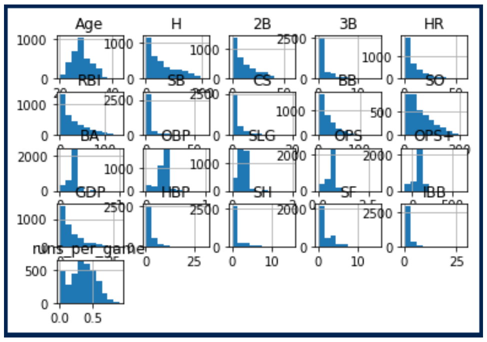
Walkthrough & in-depth summary:

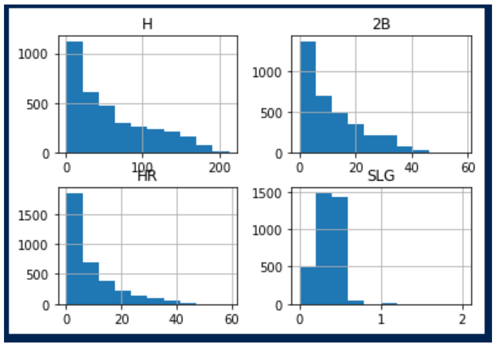
Data:
The data was gathered from baseball-reference.com. The data consisted of player batting statistics from the 2016-2022 season. Each observation was on the player level, for a given season.

The features included statistics such as, total hits, runs scored, games played, doubles, triples, home runs, batting averages, slugging percentages & more.

Clean up:
- Imported the data
- Dropped duplicates & Nas
- Dropped features in which the target (runs per game) was directly derived from.
- Also dropped features which would not be useful from the scouting perspective
    - Plate Appearances, Total Bases, At Bats
- Developed the target variable, runs per game
- Shuffled the data
- Split the data
    - Dropped outlier values from the training set
    - Dropped players who played less than 20 games from the training data

Preprocessing
- Analyzed the training data distributions
- Tested for a normally distributed training target (JB & Shapiro)

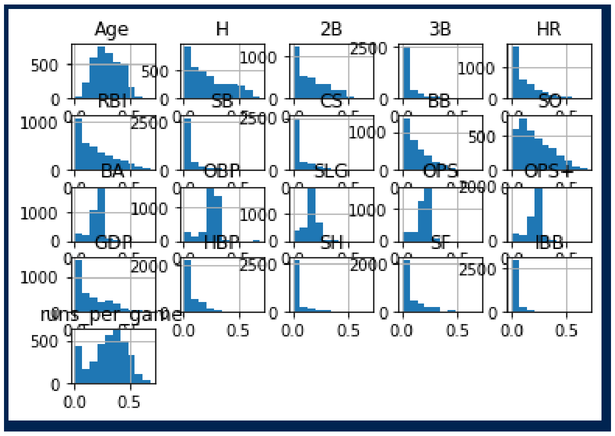




Upon testing, it seemed like the majority of the features followed a Poisson distribution. Or they were just heavily skewed. Initially thought it was a Poisson because of data structure – it is essentially count data (think totals hits, RBIs, stolen bases, etc). For a quick test, I calculated the means & variances for each variable & compared. For a Poisson distribution, the mean should be very close to the variance. This was not the case once tested. So, I concluded that the data was just highly skewed with a large variance.

To stabilize the variance seen in the data, the log transformation was used after the data was MinMax scaled.

Post Transformations

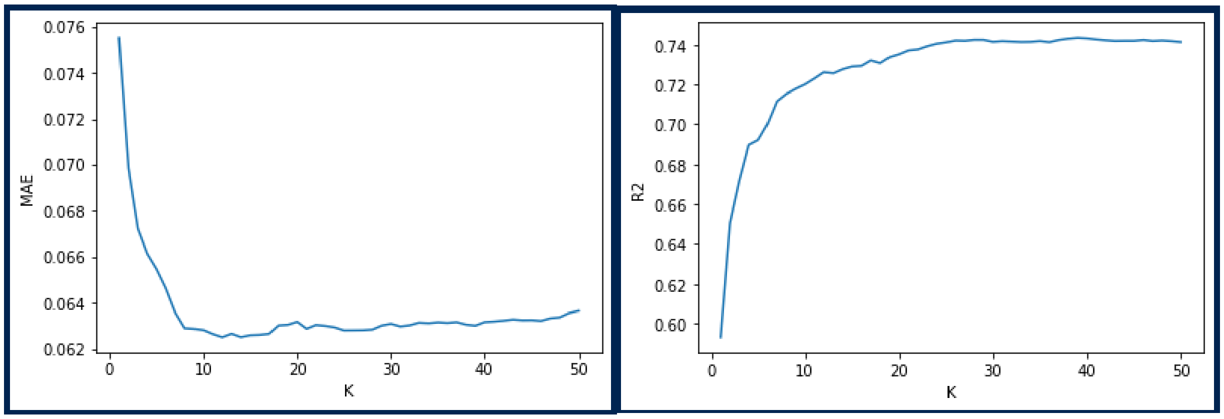


- We then defined the testing/training/validation features and labels & proceeded to the modeling.

Models:

KNN:

- Ran a standard model with K as the only set parameter.
- Went through 50 iterations to find the optimal K
    - Tested while evaluating both MAE & $R^2$.



    - K = 12/13 & K = 22/23 provided the best results.
    - Decided to run through a grid search to find better parameters.

KNN Regression:

- For KNN regression, the output value is the average of the values of k nearest neighbors.

- KNN identifies training observations $N_0$ closest to the prediction point $X_0$.
- KNN estimates $f(X_0)$ using the average of all the responses which were identified closest to the prediction point
  - In other words, KNN estimates $f(X_0)$ using the average of $N_0$
  - $f(X_0) = 1/k \sum_{x_i \in N_0} y_i$ (sts.duke.edu)

Currently, sklearn selects "brute" if the algorithm parameter is set to the default "auto" and any of these parameters are met.
- Input data is sparse
- Metric = 'precomputed'
- D > 15
  - Dimensionality is too high for tree-based methods
- K >= n/2

In our case, D > 15 since we have around 20 features. So sklearn selects "brute"
- Brute computes the distances between <u>all pairs</u> of points in the dataset for N samples & D dimensions. As compared to tree-based methods, which are described in a later section.
- In our case, Manhattan distance was selected. (p=1)
  - Manhattan distance is calculated as the sum of the absolute difference between the measures in all dimensions (D) of two points. Also known as taxi-cab distance. For example, Euclidean distance measures the shortest distance whereas Manhattan measures the sum of the absolute differences of their Cartesian coordinates – in other words, zig-zag distance (for lack of a better term)
- Leaf size was set to 1 since brute force was selected (not referenced/used)
  - If a tree-based method was selected, the leaf size would indicate the point where the tree-based method would switch to brute force measurements. (Essentially getting the sample size small enough where brute force computation would be more efficient than continuing with tree methods)
    - Tree methods infer distances based on previous calculated distances, for example, if A & B are far away & B & C are close, we can infer that A & C are also far away without having to compute the distance
- Neighbors(k) was tuned to 16
  - This is the value of the defined "neighborhood" which is used to calculate/predict new points.
  - So, we take the 16 closest observations according to the Manhattan distance computed using the brute force method, then we take the uniform average of k values since the weight parameter was set to default.
    - Because we used weights as "uniform", the weights will be equally distributed among all the 16 neighbor values.
    - Each point in the local neighborhood of k=16 will contribute equally to the prediction point

So, after the grid search was completed along with 10-fold CV, these where the selected parameters which provided best predictions on the training data. The parameters were then set & used to predict on the validation data. The MSE & predictions can be found at the bottom of the summary.

Elastic Net:
- In order to understand Elastic Net, we first need to understand L1/L2 regularization (Lasso & Ridge Regressions).
  - Elastic Net is a combination of these two approaches
- The purpose of lasso & ridge is to help stabilize a vanilla linear regression & make it more resistant against outliers/overfitting/etc.
- Each of these models achieve this by introducing a regularization/penalty term
- For Ridge, we add a penalty term which is equal to the squared magnitude of the coefficient. We also add a lambda term to control the impact of the term.
    - $RSS + \lambda * \sum \beta^2$

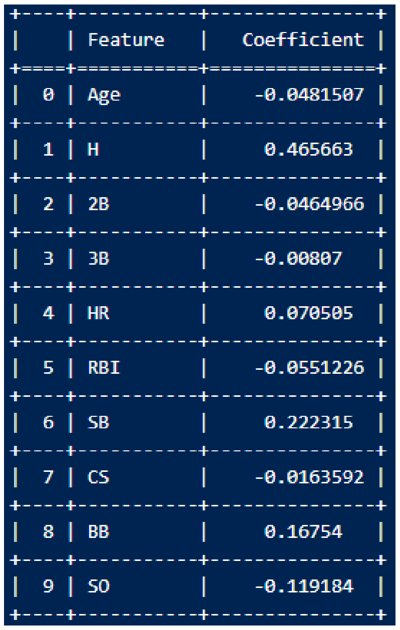
    - If lambda is 0 – the penalty term is zero, so we have a basic OLS model
    - If lambda > 0 – a constraint is added to the bias coefficient
    - As lambda gets larger, the penalty term increases – thus moving the value of the coefficient closer to 0 – i.e. become less impactful (driving down the overall size of the weight values during optimization)
    - For ridge, the coefficients can be shrunk towards zero, but none are completely eliminated
        - In other words, lambda is controlling the magnitude of the coefficient
        - If lambda is increasing, the L2 penalty is getting more strict & penalizing more – placing larger & larger constraints on the coefficients

- Lasso adds a penalty term which equals the absolute value of the magnitude of the coefficient ($\lambda*\sum|\beta|$). It is also referred to as shrinkage.
- Lasso applies its penalty term to the cost function
    - The cost function being what we are attempting to minimize in a linear regression – least squares or residual sum of squares
    - $RSS + \lambda*\sum|\beta|$
        - Residual Sum of Squares + lambda term * summation of the absolute values of the coefficients)
    - Lambda is called alpha in the sklearn code. And this is the tunable parameter
        - As lambda increases, the magnitude of the coefficients move towards zero, and can eventually be eliminated from the model
        - This regularization technique helps remove insignificant features from the model, thus providing a simpler model which is less likely to overfit & more robust against outliers.

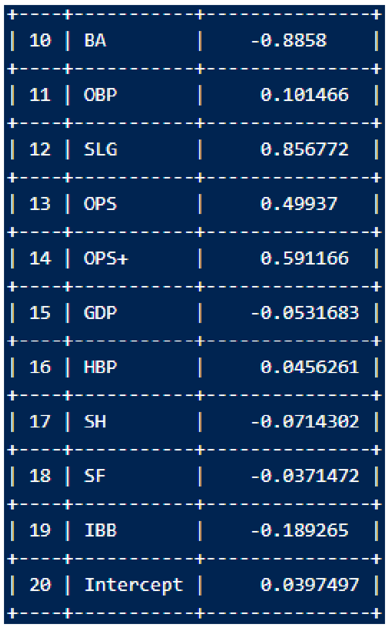
- Elastic Net – now that we have covered Lasso & Ridge (which are basically the same thing, just with different penalty terms), we can discuss how Elastic Net combines both models.
    - Elastic net literally just adds both previously discussed regularization penalties in the loss function
        - $RSS + (\lambda*\sum\beta^2 + \lambda*\sum|\beta|)$
        - Or written as $RSS + (\lambda*(1-L1\_ratio)*\sum\beta^2 + (\lambda*L1\_ratio)*\sum|\beta|)$
            - This way, we can tune the elastic net model only needing a lambda parameter & an L1_ratio parameter.
        - In our case, the L1_ratio was tuned to 0. So, if we plugged in this value to the elastic net cost function – we can see we end up with a ridge regression
            - $RSS + (\lambda*(1-0)*\sum\beta^2 + (\lambda*0)*\sum|\beta|)$
                - $RSS + \lambda*\sum\beta^2$
    - From here, the models functionality behaves in the same way as a linear regression (with a penalty term which combines L1/L2 regularization)
    - This allows for a more efficient regularization process

- Tuned parameters
    - Alpha (lambda) = 0.0001
        - Small value, so there is a smaller penalty applied – meaning less regularization
    - L1_ration = 0
        - So, we have a ridge regression

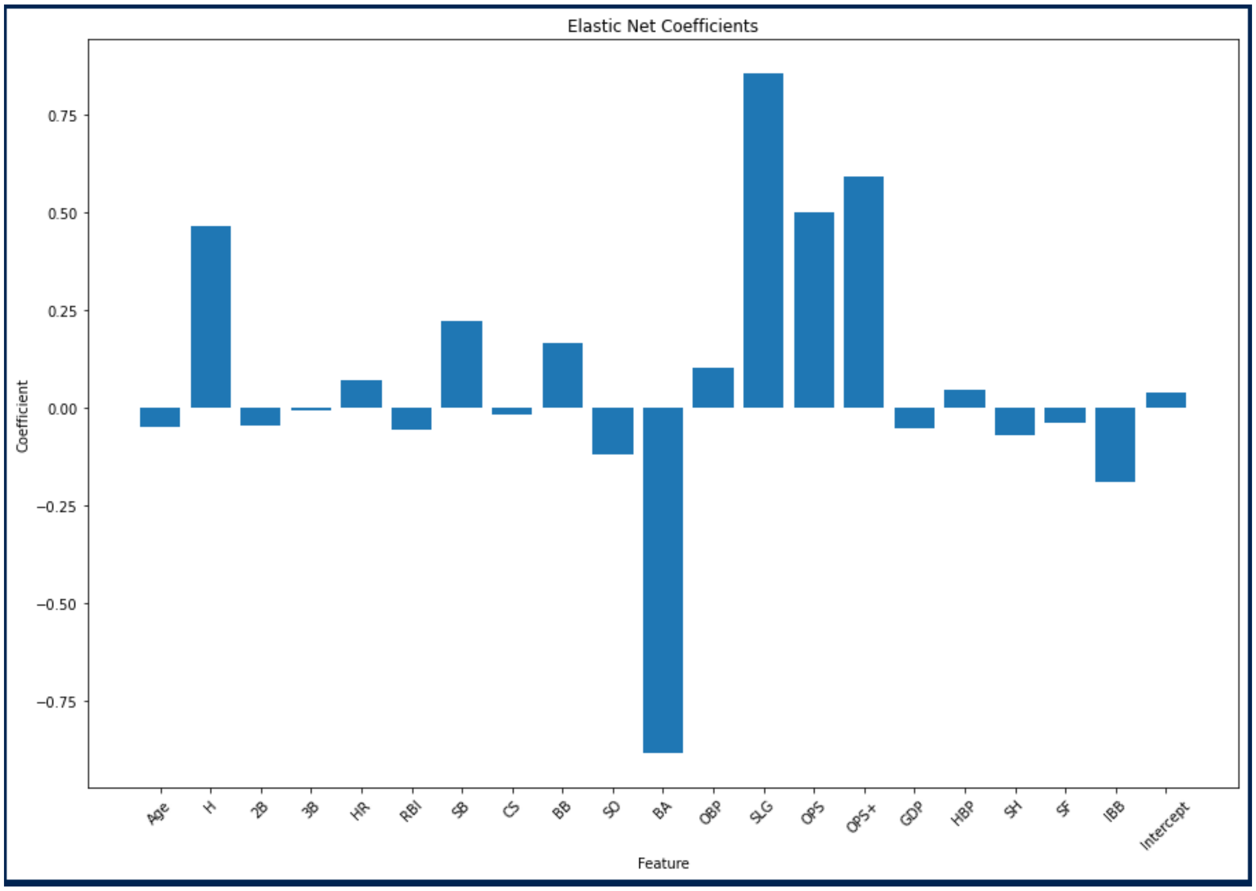
Elastic Net Results



```
+----+-----------+---------------+
|    | Feature   |   Coefficient |
+====+===========+===============+
|  0 | Age       |    -0.0481507 |
+----+-----------+---------------+
|  1 | H         |     0.465663  |
+----+-----------+---------------+
|  2 | 2B        |    -0.0464966 |
+----+-----------+---------------+
|  3 | 3B        |    -0.00807   |
+----+-----------+---------------+
|  4 | HR        |     0.070505  |
+----+-----------+---------------+
|  5 | RBI       |    -0.0551226 |
+----+-----------+---------------+
|  6 | SB        |     0.222315  |
+----+-----------+---------------+
|  7 | CS        |    -0.0163592 |
+----+-----------+---------------+
|  8 | BB        |     0.16754   |
+----+-----------+---------------+
|  9 | SO        |    -0.119184  |
+----+-----------+---------------+
```



```
+----+-----------+---------------+
| 10 | BA        |    -0.8858    |
+----+-----------+---------------+
| 11 | OBP       |     0.101466  |
+----+-----------+---------------+
| 12 | SLG       |     0.856772  |
+----+-----------+---------------+
| 13 | OPS       |     0.49937   |
+----+-----------+---------------+
| 14 | OPS+      |     0.591166  |
+----+-----------+---------------+
| 15 | GDP       |    -0.0531683 |
+----+-----------+---------------+
| 16 | HBP       |     0.0456261 |
+----+-----------+---------------+
| 17 | SH        |    -0.0714302 |
+----+-----------+---------------+
| 18 | SF        |    -0.0371472 |
+----+-----------+---------------+
| 19 | IBB       |    -0.189265  |
+----+-----------+---------------+
| 20 | Intercept |     0.0397497 |
+----+-----------+---------------+
```



We can see that slugging percentage had the most intense positive impact on a player's average runs per game. The next most positive were the OPS statistics, followed by total hits. It is expected that these features would have positive impacts on the runs per game as described below.

Positive Impacts

- Slugging
    - A measurement of batting productivity which accounts for the added value of extra-base hits (doubles, triples, homeruns). Does not include walks, etc.
    - So, this tells us that the more productive a batter is, the more runs per game they should accumulate.
- OPS/OPS*
    - On base plus slugging & On base plus slugging adjusted for ballpark
    - Another measure of batter productivity
    - Along with the slugging percentage, these features also incorporate a players on-base percentage. Which is another measure of productivity but is not more so reliant just on batting.
    - These coefficient values also indicate that more runs per game come with higher batter productivity
- Stolen bases
    - More stolen bases are associated to more runs per game.
    - This could be attributed to player aggressiveness or player instinct

Negative Impacts

- Age
    - Age had a negative impact, meaning that older players produce less runs per game
    - This makes sense as year to year wear & tear take a toll on a player's abilities & speed usually decreases with age
- Intentional Bases on Balls
    - This also had a negative impact, which could be attributed to the opposing team strategies
    - If a player is intentionally walked, or is forced to intentionally move up a base, this means that for some reason the opponent did not want to pitch to the current player at bat & would rather give up a base than allow the batter a chance to hit the ball.
    - This is often a strategy with players who were so dangerous at the plate, the team would rather pitch to the next batter, who is probably not as good – thus giving the pitcher a better chance at getting the final out.
    - For instance, let's say there are 2 outs & a man on first. It is 2003 & Albert Pujols is up to bat. Instead of risking a big hit, the pitcher may opt to just intentionally walk Pujols & get the next guy out (picking battles)
        - If this happens, both Pujols & the man on first would record an Intentional Base on Balls, but neither scored since there were two outs & the man on deck recorded the 3rd out.
- Batting Average
    - This is the only result so far which did not make any sense.
    - Further investigation would be needed to accurately depict what is going on here.
    - Batting average is a measure of hits per at bat
        - So it is intuitive to consider that a player who had a higher BA (More hits each time they went to the plate) would have higher runs per game. This would be because they get on base, thus having a chance to score.
    - But according to this model, not only was this not the case, the opposite was true, a higher batting average resulted in lower runs per game.
        - As stated above, further investigation would be required to make an accurate statement regarding this outcome

Based on the coefficients above, we can state that if the goal is to scout & sign players to maximize runs per game: A scout should focus on younger players with high slugging & OPS statistics who tend to successfully steal bases. They should also refrain from players who have a high number of intentional bases on ball. They should also refrain from using batting average as an evaluation metric.

XGBoost Model:

Extreme Gradient Boosting is an ensemble algorithm which is constructed from tree-based models. Trees are added to the ensemble & fit with the goal of minimizing prediction errors(residuals).

The objective function in an XGBoost model is the loss function and a regularization term.
Training loss + regularization = obj(0) = L(0) + Ω(0)
Where L(0) is often just the MSE, L(0) = $\sum(y_i-y^{hat}{}_i)^2$
But before we get to the objective function, our model must predict some output – which comes from building tree(s) within the ensemble.

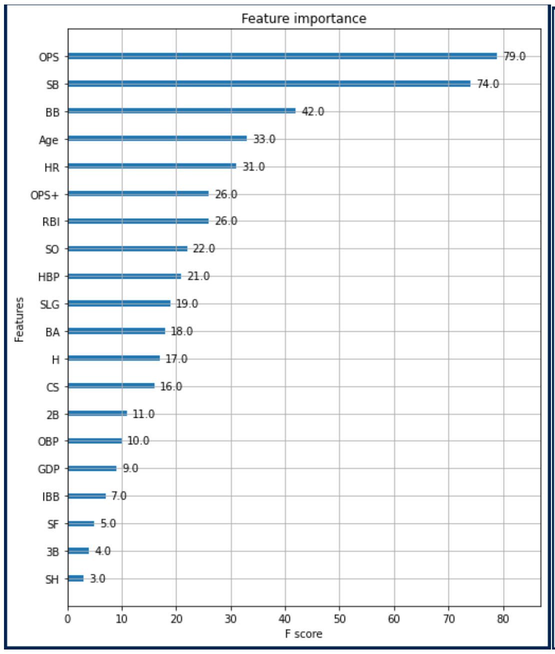
The basic steps in creating an XGBoost trees are as follows:

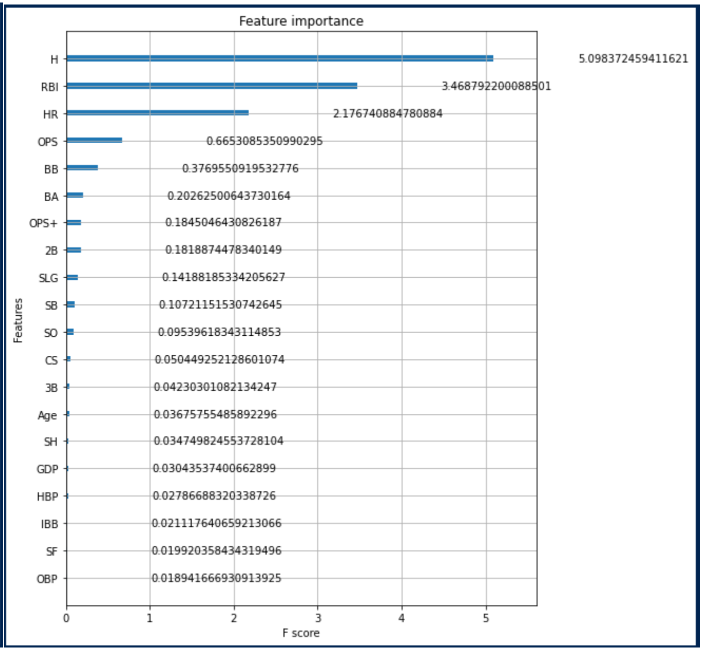
- Calculate the similarity score
  - Similarity Score/weight = $\sum(y_i-y^{hat}{}_i)^2$ / Number of residuals + λ
    - This is done for the residual (of the base learner) at the end of a split
- Calculate the gain based on the similarity scores & determine the best points to split
  - Gain = Left similarity score + right similarity score – root similarity score
    - Where the left/right similarity scores are based on the residuals resulting from a split
    - And the root similarity score is based on the base learner's residuals
- Prune the tree based on the tunable parameter gamma
  - Gamma is a pruning threshold which determines which parts of the tree should be tuned based on their information gain.
- The output for a training observation is then computed by determining the model output along with the base learner & the learning parameter.
  - A nodes output is defined by the average of the base residuals associated with the split.
  - So, we end up with the prediction being represented by…
    - Base Learner output + α*tree_output = prediction
      - Where α is the tunable learning rate
      - And tree_output = residual average based on the splitting points once the steps above are complete & the observation's feature values
      - There is no limit to the number of trees, and the above steps are only considering one tree
      - But we can have many trees/models in an ensemble, so it is more appropriate to represent the overall model output as
        - $\beta+\alpha_1[t_1]+\alpha_2[t_2]\ldots+\alpha_n[t_n]$
          - Where β is the base learner output
          - $\alpha_n$ is the learning rate with the associated tree
          - $t_1$ is the specific tree output based on the average residuals at the end of a node/final split
- This gives us our models output which we can then pass to the objective function and optimize accordingly
- The optimization process consists of evaluating a prediction based on the loss function
  - L(0), or $L(y_i,p_i) = \frac{1}{2}(y_i-p_i)^2$
- Once we apply this function, we cam determine if predictions are improving or not
  - XGboost does this by minimizing the related equation
    - $\sum L(0)+1/2(\lambda 0^2)$, where 0 is the model output value
    - $p_i$ takes the value of the base learner initial predictions & 0 is the output value (pi is from the L(0) term)
    - The loss functions containing the output values are then optimized with the goal of minimizing the function
    - This provides us with the "solution" to the output value which provides the lowest loss

In summary, the XGBoost regression model creates an ensemble of models, then produces predictions based on each trees output, the base learner & other tunable parameters. Each trees output is based on the residuals and their associated similarity weights & gain. The overall model predictions are then evaluated based on a loss function – then optimized using an objective function which minimizes the loss, based on the initial predictions and output values. The optimization process then provides a solution to the model, which minimizes the objective function, and we are left with the most appropriate/efficient xgboost model.

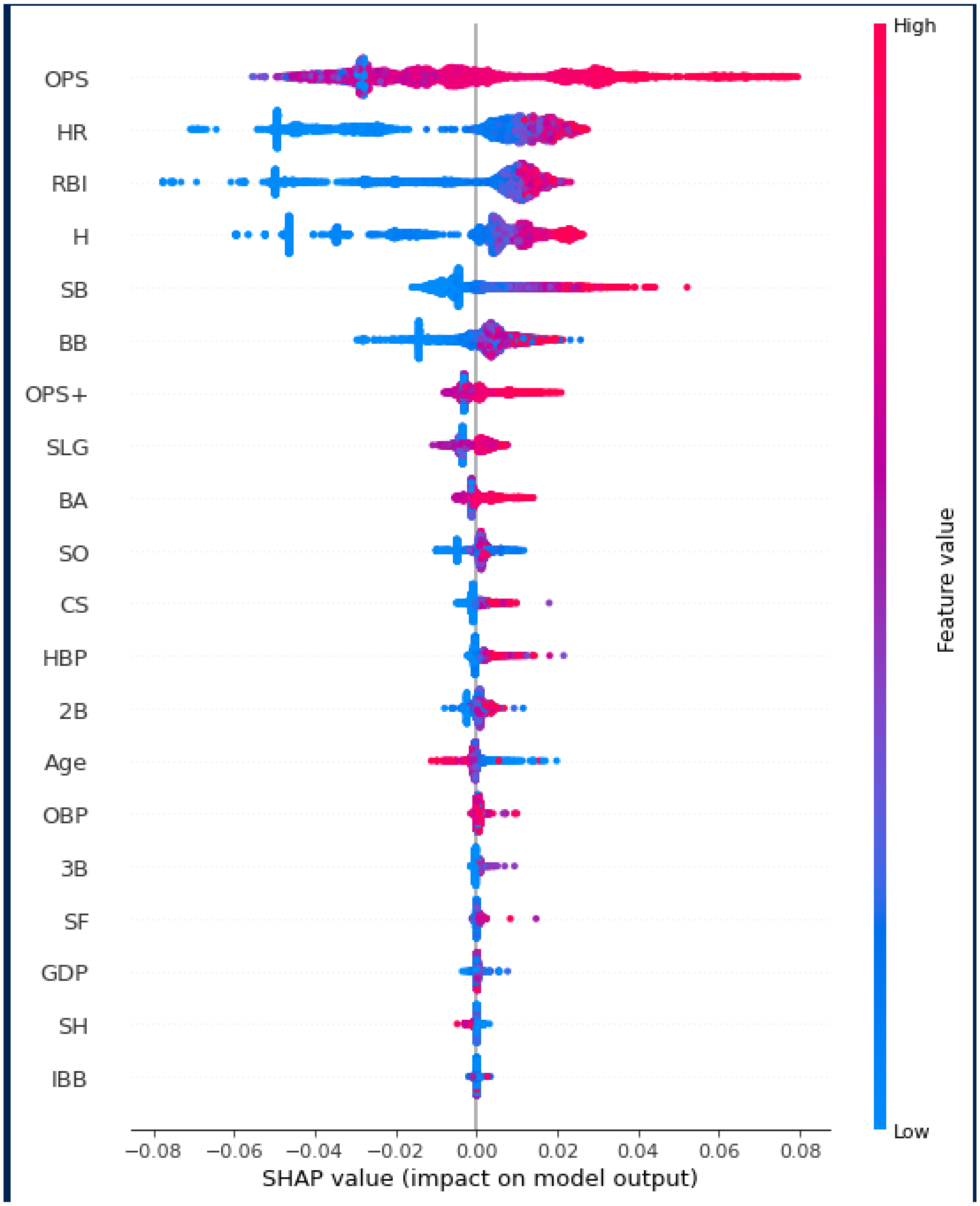
For our tuning we ended up with:
- Max Depth – Max depth of a given tree = 6
- Learning Rate – the α parameter above = 0.109
- Lambda – the λ parameter above = 0.9
- Gamma – the pruning parameter mentioned above = 0.0
- Col Sample by Tree = 0.6 – Subsampling ratio for the columns for each tree.

XGBoost Results





On the left we have weight-based feature importance. On the right we have gain-based feature importance. Weight based is the number of times a feature appears in a tree & gain is the average gain of splits which use the specific feature. It has been said that the weight-based feature importance tends to favor numerical & high cardinality features. We can see this taking place with features such as Triples & Intentional Bases on Balls, who tend to have a low range of values – are given lower importance. For that reason – gain-based feature importance will be primarily used in determining important features.



Interpretations:
OPS

- Higher values indicated a more intense positive impact on a player’s runs per game
    - OPS = On Base plus Slugging = the sum of a player’s on base percentage & slugging percentage, where…
        - Slugging = a measurement of batting productivity. (Total bases/at bats)
            - Similar to batting average, but gives more weight to extra-bases hits
        - On base percentage = how frequent a batter reaches base under any circumstance.
- This tells us that more productive players based off their OPS are more likely to score runs

Stolen Bases

- Higher stolen bases indicated a more intense positive impact on runs per game
- Players who successfully steal more bases are more subject to have a higher scored runs per game total

Age

- A lower player age is associated with a higher positive impact on runs per game
    - This can be attributed to younger players having more speed & athleticism compared to older players

RBIs/Hits/Bases on Balls/Home Runs

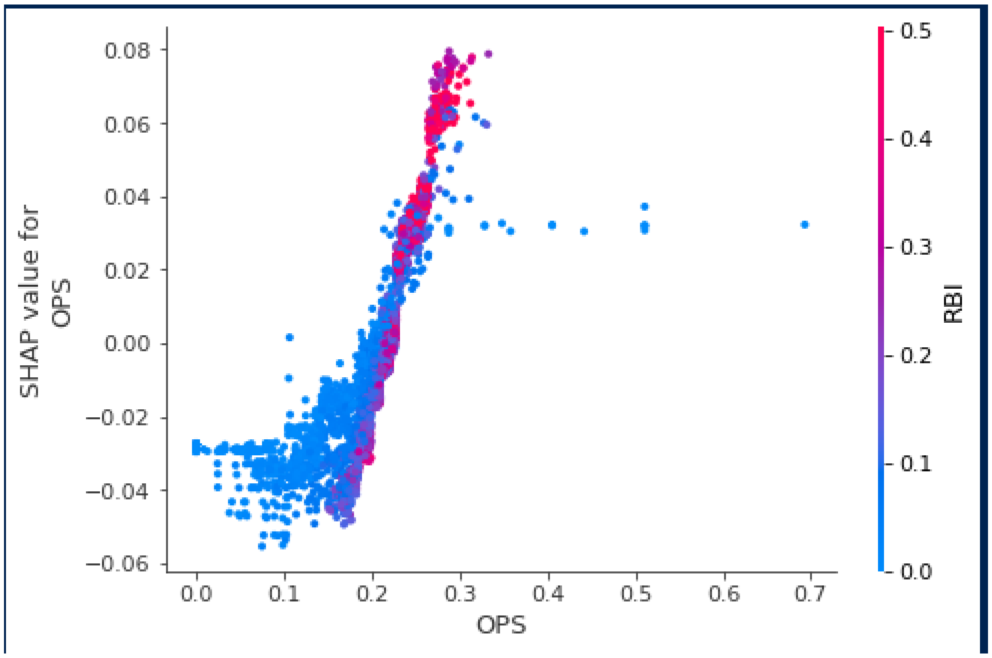
- Lower values had negative impacts on a player’s runs per game
- Lower values for these features RBI/Hits/HRs indicated that the player was having trouble getting hits – which would make it difficult to score runs.
    - So lower values for these features resulted in negative impacts on the runs per game

Key Points: If the goal is to get players & maximize runs per game

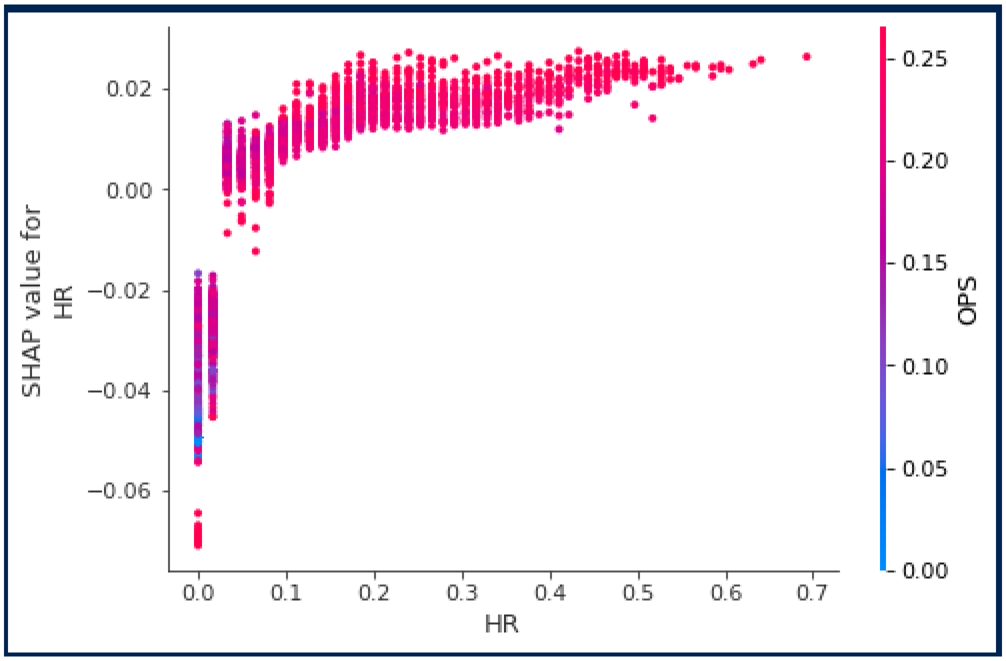
1. Get players with a high OPS & stay way from players with a low OPS
    a. This is derived from On-Base Percentage & Slugging Percentage
        i. We want the most productive players
2. Get players who can steal bases, it is not detrimental if they cannot steal. But players who can steal bases are very valuable
3. Lean towards keeping a younger roster
4. Stay away from players who have a low RBI/hits percentage & little to no bases-on-balls.

SHAP Dependency Plots

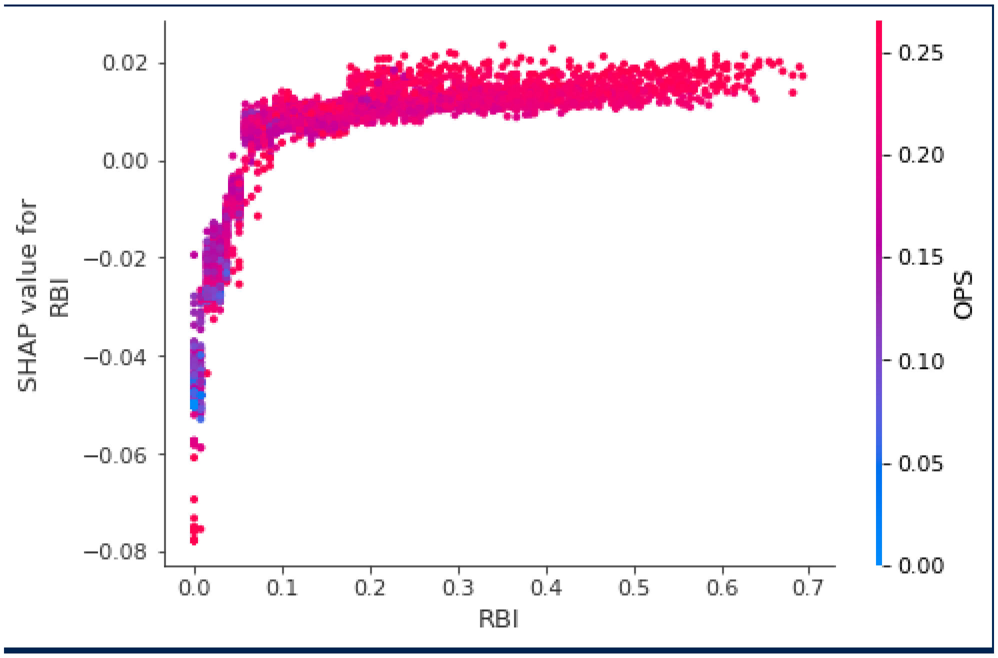
- Shows the effect of a single feature & its interaction with another feature.
- The SHAP value is on the Y axis, giving us the directional magnitude of the selected feature – and along the X axis, we have the features actual value
- The color is represented by the interaction features value.



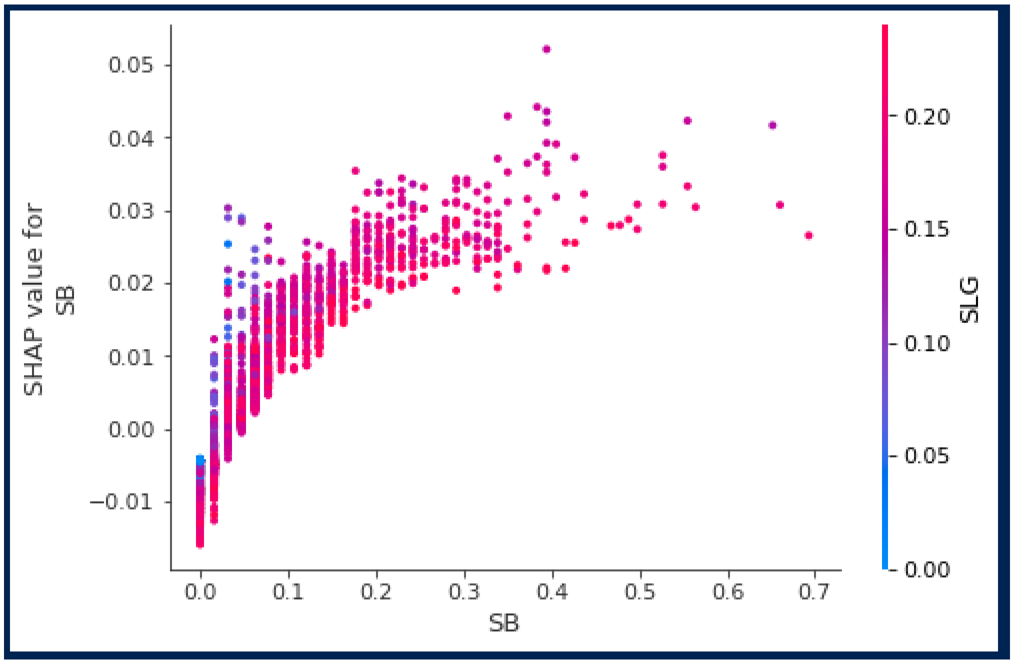
A higher RBI score is related to a higher OPS score, which has a more positive impact on runs per game. A lower RBI score is more related to a lower OPS score, which tended to have a negative impact on runs per game.



A lower OPS score is shown to be related to a lower HR score – which has a negative impact on runs per game. And a higher OPS score is related to a higher HR score – which has a positive impact on runs per game.



A higher OPS value signals a higher RBI value which is has a positive impact on runs per game. A lower OPS value is related to a lower RBI value, which has a negative impact on runs per game.



A lower slugging percentage is more related to a smaller stolen bases value  & smaller stolen bases value is less impactful on runs per game – but this impact is still almost always positive.

Model predictions on the next page.

```
Predictions on the validation data:

KKN MSE:
0.008717045649245848
Elastic Net MSE:
0.007991069461076689
XGBoost MSE:
0.008542048417424463
```

| obs | actual | knn_prediction | elnet_prediction | xgboost_prediction |
|---|---|---|---|---|
| 0 | 0.05 | 0 | 0.051066 | 0.023415 |
| 1 | 0 | 0.045078 | 0.24554 | 0.007371 |
| 2 | 0.351351 | 0.293348 | 0.32204 | 0.34585 |
| 3 | 0.553846 | 0.557441 | 0.650147 | 0.623086 |
| 4 | 0 | 0 | 0.023589 | 0.00178 |
| 5 | 0.058824 | 0.053522 | 0.024248 | 0.024875 |
| 6 | 0.142857 | 0.026553 | 0.154732 | -0.006112 |
| 7 | 0 | 0.002357 | 0.021221 | 0.001586 |
| 8 | 0.173913 | 0.347539 | 0.318478 | 0.390661 |
| 9 | 0.166667 | 0.177649 | 0.127719 | 0.197447 |
| 10 | 0.36 | 0.31589 | 0.298589 | 0.284672 |
| 11 | 0.54902 | 0.582431 | 0.621008 | 0.583792 |
| 12 | 0.428571 | 0.053848 | 0.129393 | 0.087484 |
| 13 | 0 | 0.031929 | 0.171657 | 0.009063 |
| 14 | 0.342593 | 0.324489 | 0.37864 | 0.344955 |
| 15 | 0.25 | 0.208219 | 0.218076 | 0.233225 |
| 16 | 0.776224 | 0.661911 | 0.726997 | 0.792139 |
| 17 | 0.368932 | 0.409176 | 0.469241 | 0.473229 |
| 18 | 0.153846 | 0.220671 | 0.174524 | 0.238494 |
| 19 | 0.03125 | 0.06166 | 0.062422 | 0.061073 |
| 20 | 0.472222 | 0.345215 | 0.349185 | 0.35351 |
| 21 | 0 | 0.000912 | 0.028788 | 0.009377 |
| 22 | 0.342857 | 0.189012 | 0.205793 | 0.216882 |
| 23 | 0.227273 | 0.208887 | 0.230101 | 0.217491 |
| 24 | 0 | 0.002357 | 0.020723 | 0.001586 |
| 25 | 0.166667 | 0.056328 | 0.247562 | 0.05086 |
| 26 | 0.207317 | 0.421568 | 0.394099 | 0.458868 |
| 27 | 0.5 | 0.581593 | 0.519148 | 0.616711 |
| 28 | 0.111111 | 0.057693 | 0.079002 | 0.076902 |
| 29 | 0.043478 | 0.027232 | 0.134215 | 0.032554 |
| 30 | 0.362903 | 0.322586 | 0.336318 | 0.389962 |
| 31 | 0 | 0.002357 | 0.021221 | 0.001586 |
| 32 | 0 | 0.030744 | 0.05202 | 0.017941 |
| 33 | 0.25 | 0.229303 | 0.267784 | 0.230451 |
| 34 | 0.572327 | 0.644441 | 0.813295 | 0.745788 |
| 35 | 0.515528 | 0.513261 | 0.509463 | 0.503778 |
| 36 | 0.209677 | 0.209097 | 0.172469 | 0.196271 |
| 37 | 0 | 0.043758 | 0.173799 | 0.039432 |
| 38 | 0.333333 | 0.255126 | 0.313949 | 0.289363 |

| | | | | |
|---|---|---|---|---|
| 39 | 0.095238 | 0.108192 | 0.134638 | 0.089284 |
| 40 | 0.8 | 0.187938 | 0.471201 | 0.165587 |
| 41 | 0.5 | 0.495812 | 0.570434 | 0.555853 |
| 42 | 0 | 0.002779 | 0.026781 | 0.004857 |
| 43 | 0.133333 | 0.134451 | 0.286553 | 0.10668 |
| 44 | 0.166667 | 0.152581 | 0.271097 | 0.182857 |
| 45 | 0.121212 | 0.092275 | 0.071774 | 0.126432 |
| 46 | 0.398437 | 0.500187 | 0.503651 | 0.450107 |
| 47 | 0.267606 | 0.266032 | 0.271757 | 0.284305 |
| 48 | 0.318182 | 0.340635 | 0.322371 | 0.331254 |
| 49 | 0 | 0.000912 | 0.027787 | 0.009377 |
| 50 | 0.083333 | 0.199109 | 0.227226 | 0.182998 |
| 51 | 0.7 | 0.531218 | 0.646209 | 0.661408 |
| 52 | 0.288889 | 0.391021 | 0.337959 | 0.41495 |
| 53 | 0 | 0 | 0.021904 | 0.003415 |
| 54 | 0.314516 | 0.337694 | 0.284494 | 0.291502 |
| 55 | 0.381818 | 0.339895 | 0.348577 | 0.392377 |
| 56 | 0 | 0 | 0.024088 | 0.00178 |
| 57 | 0.481481 | 0.453495 | 0.42462 | 0.548683 |
| 58 | 0.4 | 0.426206 | 0.423642 | 0.499948 |
| 59 | 0.322581 | 0.131642 | 0.180619 | 0.19131 |
| 60 | 0 | 0 | 0.022631 | 0.001601 |
| 61 | 0.632653 | 0.441537 | 0.426809 | 0.526523 |
| 62 | 0.229358 | 0.365441 | 0.275825 | 0.309423 |
| 63 | 0.510638 | 0.336436 | 0.264087 | 0.307751 |
| 64 | 0.58209 | 0.439712 | 0.412036 | 0.469772 |
| 65 | 0 | 0.048925 | 0.036129 | 0.022083 |
| 66 | 0.363636 | 0.071282 | 0.363695 | 0.136376 |
| 67 | 0.035714 | 0.062579 | 0.124186 | 0.110509 |
| 68 | 0.503401 | 0.52738 | 0.525871 | 0.519185 |
| 69 | 0.230769 | 0.166218 | 0.229211 | 0.127573 |
| 70 | 0 | 0.002357 | 0.021721 | 0.001586 |
| 71 | 0.601307 | 0.591406 | 0.583552 | 0.604543 |
| 72 | 0.583333 | 0.400089 | 0.316957 | 0.47229 |
| 73 | 0.090909 | 0.066656 | 0.232233 | 0.057146 |
| 74 | 0.670807 | 0.670059 | 0.756384 | 0.714451 |
| 75 | 0.452381 | 0.475398 | 0.392512 | 0.420047 |
| 76 | 0.478723 | 0.374745 | 0.386866 | 0.400029 |
| 77 | 0.032258 | 0.066458 | 0.02179 | 0.062698 |
| 78 | 0.041667 | 0.007422 | 0.031839 | 0.025731 |
| 79 | 0.043478 | 0.0587 | 0.078989 | 0.074421 |
| 80 | 0 | 0.03812 | 0.209125 | 0.009673 |
| 81 | 0.222222 | 0.046756 | 0.173185 | 0.008564 |
| 82 | 0 | 0.040252 | 0.235735 | 0.028865 |
| 83 | 0.469231 | 0.562558 | 0.593673 | 0.611505 |
| 84 | 0.220339 | 0.270031 | 0.281101 | 0.308125 |
| 85 | 0.189873 | 0.323517 | 0.299867 | 0.306766 |

| | | | | |
|---|---|---|---|---|
| 86 | 0.096774 | 0.11377 | 0.1003 | 0.137765 |
| 87 | 0.2 | 0.246355 | 0.246618 | 0.21692 |
| 88 | 0.557047 | 0.590962 | 0.635001 | 0.621497 |
| 89 | 0 | 0.001543 | 0.013177 | 0.008762 |
| 90 | 0 | 0.054739 | 0.125793 | 0.081299 |
| 91 | 0.322314 | 0.427229 | 0.423222 | 0.382819 |
| 92 | 0 | 0 | 0.01936 | 0.001586 |
| 93 | 0.272727 | 0.114535 | 0.154278 | 0.122539 |
| 94 | 0.375 | 0.112254 | 0.396176 | 0.086694 |

References:

Lots & lots… of online resources.

scikit-learn.org
baseball-reference.com
xgboost.readthedocs.io
youtube.com
stackoverflow.com
stackexchange.com
statology.org
geeksforgeeks.org
medium.com
mljar.com
iq.opengenus.org
dataaspirant.com
wikipedia.org
machinelearningcompass.com
corporatefinanceinstitute.com
mygreatlearning.com