IBM Applied Data Science Capstone

Predicting MLB Team Wins per Season

Landon Albritton

1. Introduction/ Business Problem

Sporting events are deeply integrated in many people's lives. In the United States, baseball is one of the most popular sports. The highest level of play occurs in Major League Baseball (MLB), a professional league worth billions of dollars. From television contracts alone, MLB earns $1.5 billion each year from domestic viewers. It is important for a baseball team to win games because it will attract higher fan attendance, television viewership, and, perhaps most importantly, revenue. Predicting the outcome of baseball games is an even more lucrative field. CNBC estimates that over $30-$40 billion is wagered on games every year. Luckily, MLB games lend themselves well to machine learning because of the vast amount of data that is recorded for each game. A downside, though, is that baseball itself is quite noisy and thus difficult to predict with high accuracy. Any player having a "good" day can hit a home run, and this action alone can allow for a lesser-skilled team to defeat a better team. After all, if it were that easy to predict the winning team for each game, then why even play the games at all? Nevertheless, we will explore machine learning algorithms using data from recent MLB seasons and try to predict the MLB Team Wins per Season as accurately as possible.

2. Data

2.1 MLB Data

The data for this project came from the [Lahman database] (http://www.seanlahman.com/baseball-archive/statistics/), a respected record of historical baseball data, which I downloaded and placed in a local database. The following is the list of attributes and its description.

Table – I: Dataset Features

Feature	**Description**
yearID	Year
lgID	League
teamID	Team
franchID	Franchise (links to TeamsFranchise table)
divID	Team's division
Rank	Position in final standings
G	Games played
GHome	Games played at home
W	Wins
L	Losses
DivWin	Division Winner (Y or N)
WCWin	Wild Card Winner (Y or N)
LgWin	League Champion(Y or N)
WSWin	World Series Winner (Y or N)
R	Runs scored
AB	At bats
H	Hits by batters
2B	Doubles
3B	Triples
HR	Homeruns by batters
BB	Walks by batters
SO	Strikeouts by batters
SB	Stolen bases

CS	Caught stealing
HBP	Batters hit by pitch
SF	Sacrifice flies
RA	Opponents runs scored
ER	Earned runs allowed
ERA	Earned run average
CG	Complete games
SHO	Shutouts
SV	Saves
IPOuts	Outs Pitched (innings pitched x 3)
HA	Hits allowed
HRA	Homeruns allowed
BBA	Walks allowed
SOA	Strikeouts by pitchers
E	Errors
DP	Double Plays
FP	Fielding percentage
name	Team's full name
park	Name of team's home ballpark
attendance	Home attendance total
BPF	Three-year park factor for batters
PPF	Three-year park factor for pitchers
teamIDBR	Team ID used by Baseball Reference website
teamIDlahman45	Team ID used in Lahman database version 4.5
teamIDretro	Team ID used by Retrosheet

3. Methodology

3.1 Cleaning and Preparing Data

A) Irrelevant Features

Through literature survey, I dropped columns which are irrelevant to study. The columns used are: 'yearID', 'teamID', 'G', 'W', 'R', 'AB', 'H', '2B', '3B', 'HR', 'BB', 'SO', 'SB', 'CS', 'HBP', 'RA', 'ER', 'ERA', 'CG', 'SHO', 'SV', 'IPouts','HA', 'HRA', 'BBA', 'SOA', 'E', 'DP', 'FP'.

B) NULL Values

Null values influence the data quality, which in turn can cause issues with machine learning algorithms.

Three of the columns have a relatively small amount of null values. There are 120 null values in the SO (Strike Outs) column, 144 null values in SB (Stolen Bases) and 317 in the DP (Double Play) column. Two of the columns have a relatively large amount of them. There are 859 null values in the CS (Caught Stealing) column and 2325 in the HBP (Hit by Pitch) column. So we dropped CS and HBP column. And, also SO, DP, SB column’s null value are filled with median value.

3.2) Data Exploration and Visualization

The shape of data now is (2805, 27).

Below is the distribution of Wins.

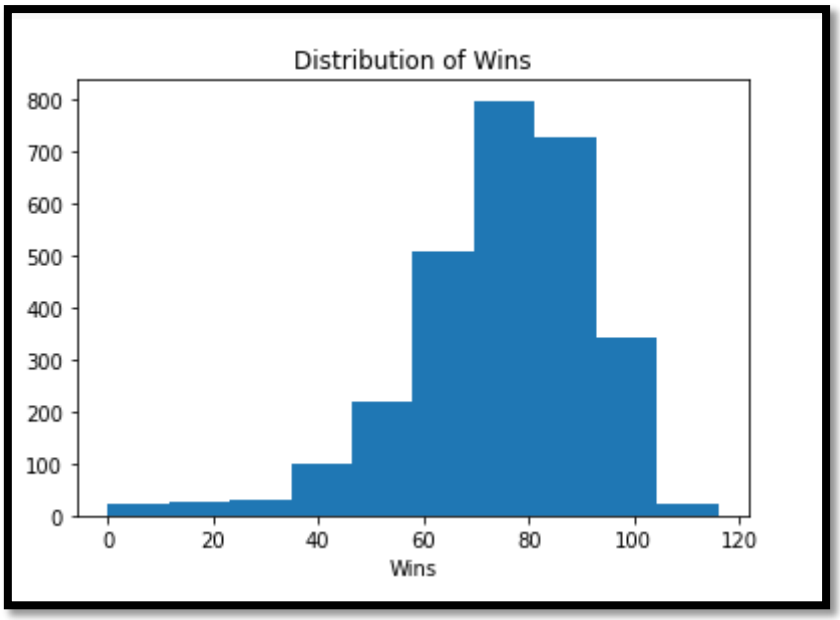

Figure-1 Distribution of Wins

From Figure-1, we notice that the wins range from 0 to 120 and mean lies at around 74.740.

Now let's make a scatter graph with the year on the x-axis and wins on the y-axis and highlight the win_bins column with colors.

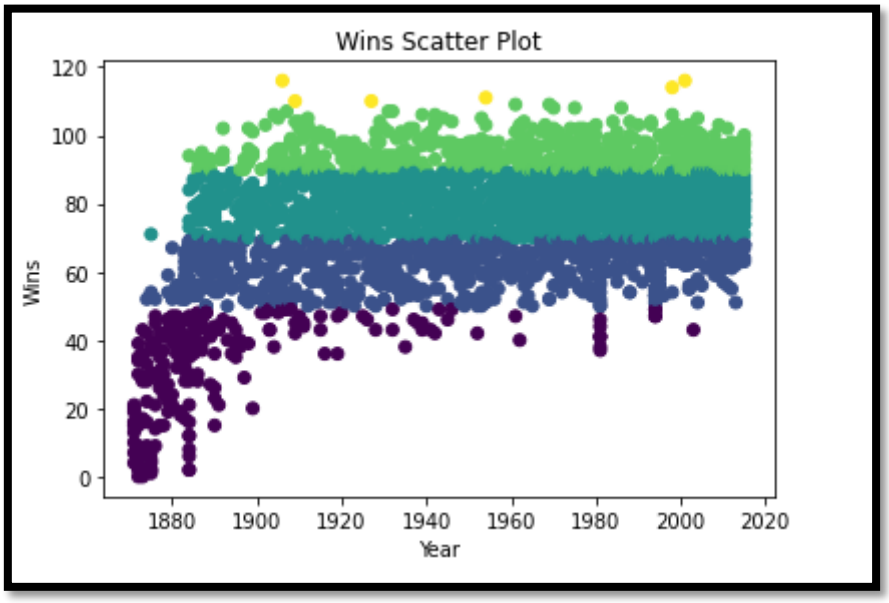

Figure-2 Scatterplot of Wins w.r.t Year

As you can see in the above scatter plot, there are very few seasons from before 1900 and the game was much different back then. Because of that, it makes sense to eliminate those rows from the data set.

When dealing with continuous data and creating linear models, integer values such as a year can cause issues. It is unlikely that the number 1950 will have the same relationship to the rest of the data that the model will infer. We avoid these issues by creating new variables that label the data based on the yearID value.

Major League Baseball (MLB) progressed, different eras emerged where the amount of runs per game increased or decreased significantly. The dead ball era of the early 1900s is an example of a low scoring era and the steroid era at the turn of the 21st century is an example of a high scoring era.

Let's make a graph below that indicates how much scoring there was for each year.

We'll start by creating dictionaries runs_per_year and games_per_year. Populate the runs_per_year dictionary with years as keys and how many runs were scored that year as the value. Populate the games_per_year dictionary with years as keys and how many games were played that year as the value. Below is the code snippet.

```
# Create runs per year and games per year dictionaries
runs_per_year = {}
games_per_year = {}

for i, row in df.iterrows():
    year = row['yearID']
    runs = row['R']
    games = row['G']
    if year in runs_per_year:
        runs_per_year[year] = runs_per_year[year] + runs
        games_per_year[year] = games_per_year[year] + games
    else:
        runs_per_year[year] = runs
        games_per_year[year] = games

print(runs_per_year)
print(games_per_year)
```

Finally, we create the plot from the mlb_runs_per_game dictionary by putting the years on the x-axis and runs per game on the y-axis.

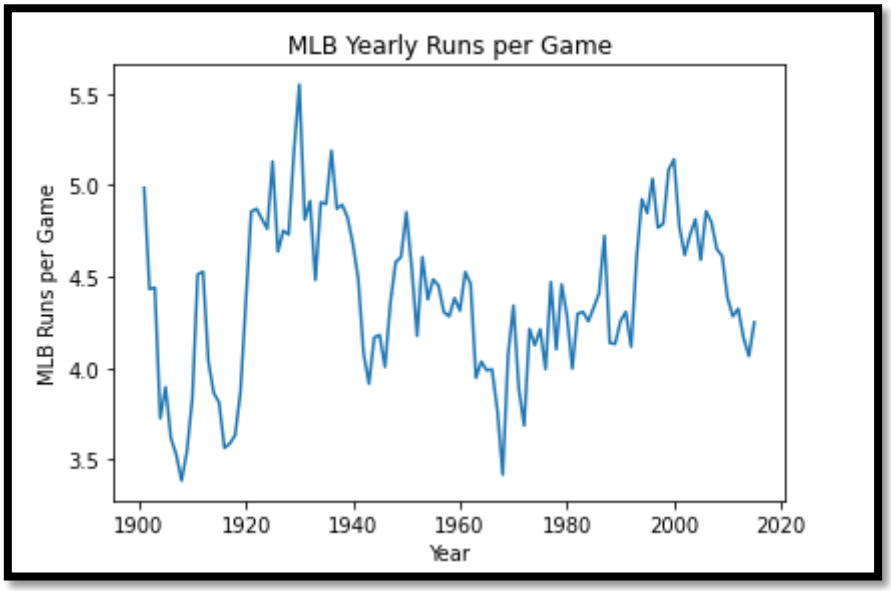

Figure-3 MLB Runs per Game vs. Year

Adding New Features

Now we can create new variables that indicate a specific era that each row of data falls in based on the yearID. We'll convert the years into decades by creating dummy columns for each decade. Then we can drop the columns that we don't need anymore. The bottom line in the game of baseball is how many runs we score and how many runs we allow. We can significantly increase the accuracy of your model by creating columns which are ratios of other columns of data. Runs per game and runs allowed per game will be great features to add to our data set. From Figure-5, we can say that Runs per Game is positively correlated with Wins. And, Runs Allowed per Game is negatively correlated with Wins.

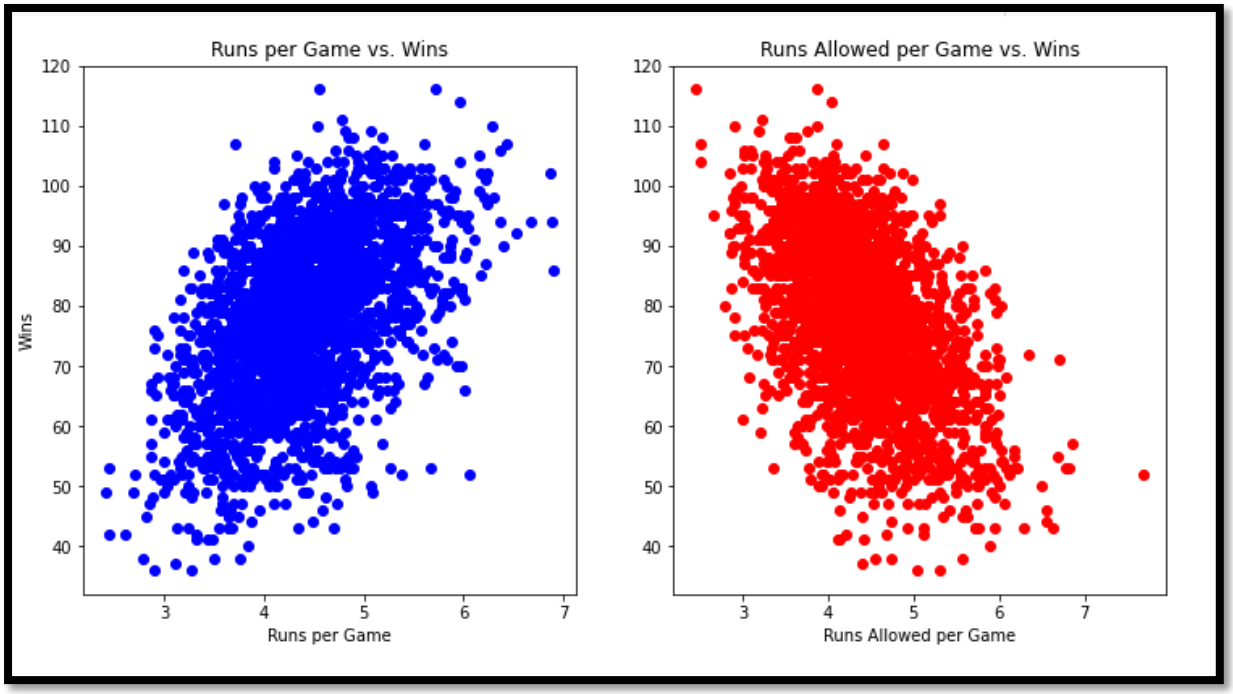

Figure-4 Scatterplot

Before getting into any machine learning models, it can be useful to see how each of the variables is correlated with the target variable.

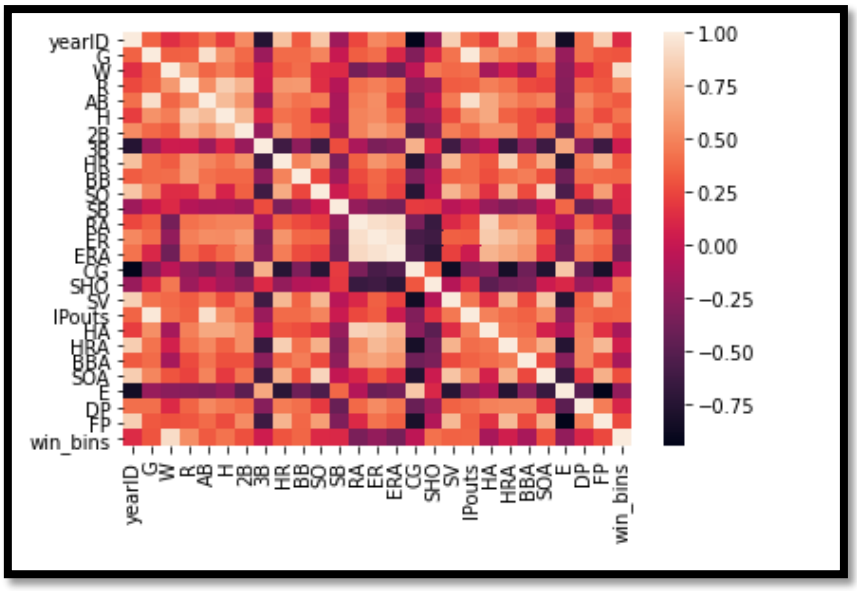

Figure-5 Correlation Heat Map

From Figure-5, we deduce that Column R and win_bins are highly correlated with column W with 0.59 and 0.91 correlation value respectively.

4. Model Training

K-means is a simple clustering algorithm that partitions the data based on the number of k centroids we indicate. Each data point is assigned to a cluster based on which centroid has the lowest Euclidian distance from the data point.
One aspect of K-means clustering that we must determine before using the model is how many clusters we want. We can get a better idea of your ideal number of clusters by using sklearn's silhouette_score() function. This function returns the mean silhouette coefficient over all samples. We want a higher silhouette score, and the score decreases as more clusters are added. Using Silhouette score, we found 6 as optimal number of clusters.
So we retrain the Kmeans algorithm with n_clusters=6 and below is the result.

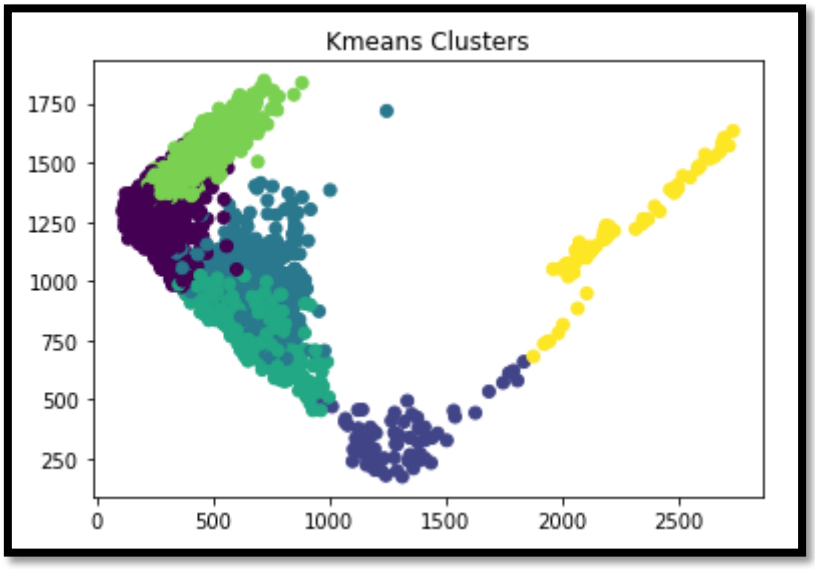

Figure-6 K-Means Cluster

Now, we add the labels from our clusters into the data set as a new column. Also, we add the string “labels” to the attributes list.
Mean Absolute Error (MAE) is the metric we’ll use to determine how accurate our model is. It measures how close the predictions are to the eventual outcomes. Specifically, for this data, that means that this error metric will provide us with the average absolute value that our prediction missed its mark. This means that if, on average, our predictions miss the target amount by 5 wins, and our error metric will be 5.
The first model we train is a linear regression model.
We found out that the average number of wins was about 79 wins. On average, the model is off by only 2.687 wins.
Next, we try a Ridge regression model. The RidgeCV model allows us to set the alpha parameter, which is a complexity parameter that controls the amount of shrinkage. The model will use cross-validation to determine which of the alpha parameters out of (0.01, 0.1, 1.0, 10.0) is ideal.
This model performed slightly better, and is off by 2.673 wins, on average.

5. Discussion

For this project, the Ridge Regression Model give accurate results as such its off by only 2.687 wins.
We applied K-means algorithm to get optimal number of clusters but we notice that the cluster is not precise enough. Adding more hybrid features would be useful.

6. Conclusion

In this project, we used the data from Lahman database to predict the MLB Team wins per season. From literature review, we first removed the irrelevant columns and further more removed the null values. We train linear regression and Ridge Regression model. Ridge Regression produce an offset of 2.673 which outperforms the linear regression which produce the offset of 2.687.

7. References

1. Firsick, Z. 2013. “Predicting Major League Baseball Playoff Outcomes Through Multiple Linear Regression”. PhD thesis, University of South Dakota.
2. Kelleher, J. D., B. M. Namee, and A. D’Arcy. 2015. *Fundamentals of machine learning for predictive data analytics: Algorithms, worked examples, and case studies*. MIT Press, Cambridge, MA.