Final Year Project

Machine Learning and Baseball

Kelsey Osos

Student ID: 16201972

A thesis submitted in part fulfilment of the degree of

BSc. (Hons.) in Computer Science

Supervisor: Gianluca Pollastri

UCD School of Computer Science

University College Dublin

May 17, 2020

Table of Contents

Abstract

Github

Machine Learning and Baseball represents an attempt to provide a new approach to quantifying performance value in baseball players and use those measures to develop a performance prediction model for teams. Data analysis and machine learning are nothing new to the sport, and a multitude of methods have been employed to give players "ratings" which can be used to compare them to others and derive some meaningful measure of success. The typical approach is to attempt to develop a "one size fits all" statistic and compare it against real-world performance, and the *de facto* implementation of this statistic has changed as the game has been analyzed at greater depth. Along with player evaluation, a key aspect of baseball data analytics is predictive modelling: finding the most successful way possible to, given all data points possible, construct some probabilistic description of a game's outcome. This project will attempt only to construct a predictive model; however, having historical statistical measures will provide the interested party with a sufficiently representative aggregate baseline against which to measure numerical results. The culmination of this endeavor should be that it is possible to create a team as a network of players, along with outlying factors such as park, manager, time of day and any other measure found to have a meaningful impact, and pit that team against another to derive a model of the outcome for a given game.

Special Note

Due to the Coronavirus pandemic, I was forced to make a priority of packing up my life in Ireland and returning home to the United States as quickly as possible. This unexpected uprooting severely affected my ability to work on this project to the extent which I would have otherwise, and subsequently I was unable to achieving all of the milestones I had planned on.

Knowing that I would not achieve my advanced goals, I altered my process and attempted to find a way to build a complete (albeit stripped down) product rather than attempt to get as much of the previously stated functionality as possible and end up with a partially completed, unusable application. The updated functional aims are as follows:

Functionality

1. Integration of data from Retrosheet and Sean Lahman datasets into single store in SQLite or PostgreSQL.
2. Usage of Python, TensorFlow and Keras to perform evaluation of data using convolutional neural networks.
3. Development of ANN models to assign players (hitters and pitchers) with numeric rating values to derive a new way to evaluate statistics. These models will combine to form the team model mentioned next.
4. Creation of a "team" model, consisting of players, which will be paired with another team model to predict the outcome of a game using a CNN based on statistics of pitchers, position players and season trends of the teams.
    - While linear comparisons will suffice early on, the eventual aim is to produce a more expressive model that defines not only the ability, but the character of each team.
5. Inclusion of external factors into decision-making process, such as: park factor (home team, dimensions, elevation), season trends, temperature, manager adjustments etc.

I understand that I am moving the goalposts after the fact, as it were, which is not my right; I do hope, however, that the current situation is taken into account when evaluating these projects.

Additional sections added for the final report are: Data and Context, Core Contribution and Evaluation.

Chapter 1: Project Specification

Core Functionality

1. Integration of data from Retrosheet and Sean Lahman datasets into single store in SQLite or PostgreSQL.
2. Usage of Python, TensorFlow and Keras to perform evaluation of data using convolutional neural networks.
3. Creation of a "team" model, consisting of players, which will be paired with another team model to predict the outcome of a game using a CNN based on statistics of pitchers, position players and season trends of the teams.
    - While linear comparisons will suffice early on, the eventual aim is to produce a more expressive model that defines not only the ability, but the character of each team.
4. Extrapolation of this predictive model to run season simulations and predict end-of-year standings for each team.
5. Inclusion of external factors into decision-making process, such as: park factor (home team, dimensions, elevation), season trends, temperature, manager adjustments etc.

Advanced Functionality

1. MVC web application representation of resulting dataset via a RESTful API served through Django and using HTML/SCSS, JavaScript, jQuery and Bootstrap.
2. Deconstruction of predictive model to formulate one-stop player statistical measures - use same player-based neural network approach to define the individual importance of each statistic in formulating how "good" a player is.
3. Web application feature to "run simulations" on seasons when given a schedule and rosters.

Chapter 2: **Introduction**

Since baseball's inception as a major, professional competitive game in 1876, it has evolved into an ideal playing ground for statistical analysis. In 1971, the Society for American Baseball Research (SABR) was founded by Robert Davids as an organization with the intent of collecting, categorizing and disseminating all relevant baseball history and records. The group offers open access to publications to support and foster interest and knowledge in the discipline. [1] In 1977, Bill James revolutionized the baseball world with the first publication of *The Bill James Baseball Abstract*. This book changed the way people look at the game: James presented the data, gathered from his analysis of box scores, in a completely sanitized environment independent of any emotion, superstition or "gut-feeling". [2] This paved the way for a purely analytical approach to the game, and baseball became a haven for mathematicians and statisticians who wanted to study something "fun". Data analysts have seen the potential in this sport for decades, and the world of baseball statistics has undergone varied and fascinating eras which all strive to answer the same basic questions: what makes a player good, and what is the best way to measure his value and impact on his team?

The game of baseball provides an ideal stage for observation and analysis: each play is isolated, with a set beginning, middle and end, and each exists as a single point of data between (usually) two players - the pitcher and the hitter- where each attribute of interest can be keenly dissected and looked at for years to come. It follows a simple pattern: the setup, the throw, the swing or take, and the result. Each pitch is a row and each game a table, and every single facet of the game can be analyzed in the search for any conclusion the analyst could want to make. Baseball is no longer just a sport that fosters a perfect environment for data analysis: it has become a game *driven* by it.

Deriving a prediction model for a team as a conglomeration of the wealth of available player data is nothing new to the sport. An analyst must simply take the primary data that is recorded, find some method of evaluation of that data into a player statistic, aggregate the statistics of those players and plug the result into a predictor with another such collection. If one requires inclusion of environmental data such as game temperature and park factor, those factors are relatively simple to feed in. This is a natural culmination of the history of statistical analysis in baseball: a predictive model such as a decision tree is perfectly suited to game outcomes, and a simple Bayesian approach to constructing win probabilities is so ubiquitous that it appears in machine learning coursework examples. As the reader will observe in the following discussion of player evaluation, calculating advanced measures is essentially just the process of applying weights to, then aggregating, primary statistics. One can easily understand that this linear weights model is already a sort of baseline machine learning implementation: naturally, the next step would be to leave linearity behind and attempt to construct a more expressive network.

The problem then is volume of data: Yakovenko asserts that not enough exists to train deep networks for baseball, and he does so after describing his fully-connected network. [3] Perhaps, though, this is not the end of the conversation. While a baseball data set certainly would not have the same volume of entities as something like an image set, there should be a way to handle the data that is available so as to derive a successful neural network: the analyst need only approach the problem with forethought and care. This conundrum motivates the use of a convolutional neural network, a partially-connected model.

Chapter 3: Related Work and Ideas

The majority of baseball analysis attempts to evaluate the game at the player level, constructing some way to evaluate a player's past performance and, more importantly, predict his future impact to the team. Dozens of models have been proposed and accepted with varying success, and analysts are always attempting to find that vastly underestimated measure or perfect combination of statistics that hasn't yet been discovered.

In the course of these evaluations, a central concern has always presented itself foremost among the others: rather than focusing on how good a player is, the world of baseball analytics asks instead how much value that player adds to his team. This measure is more quantifiable; it can be expressed both as "how many wins will he earn us over the course of 162 games" and "how much, in dollars, is he worth?". With the advent of the moneyball movement, the focus has centered around finding value in undervalued players and constructing an affordable team that can win: this very approach relies on being able to spot something in a player that the twenty-nine other teams cannot. In a conversation about finding "hidden" correlations and contributions in a wide array of data, the potential application of a neural network seems evident.

Of course, any project in data analytics benefits from sufficient foreknowledge in the problem domain, and in the case of baseball this means a baseline understanding of each statistic and what it says about a player or team. From this, one may move then into the models and methods put forth by past experts to describe, as completely as possible, some evaluation of player value and team performance.

There are two main categories of player, and they must be analyzed differently both in terms of how they impact their team's performance and what it means to be "good". As such, this report will discuss pitchers and position players separately.

3.1 Evaluating Pitchers

A starting pitcher could be said to be the primary contributing factor on his team on the day that he pitches, which is typically once every five games (in other words, a team typically has five starting pitchers who rotate). As such, it may be important to dissect a pitcher's statistics with greater fastidiousness than we would a hitter: an important aspect of a game's outcome is the performance difference of the teams' pitchers. The pitcher's impact is clear: there are nine players in the lineup who attempt to score runs, but only one who has to get the ball over the plate without giving them up (defense notwithstanding). This view is shared by the analysts at FiveThirtyEight: when deriving their Elo system for team ratings, one of the key factors affecting the pregame rating is the impact of the starting pitcher (the Elo system will be described in "Evaluating Teams", while the pitcher's impact is discussed in "Measuring a Pitcher's Performance: Advanced Statistics") Clearly the final result, a pitcher's stat-line against a team with some estimated offensive ability, is the measure an analyst may find most interesting. However, the goal here is projection - not reflection. So one must look for measures of value in the data that can be extrapolated to future games in an attempt to predict how that pitcher will perform. For instance: if the model classifies Barry Zito as an offspeed pitcher who relies on a lot of flyouts to get through a start, and he's throwing against the hard-hitting Colorado Rockies in their stadium at high elevation on a warm

day, one would have valid reason to be worried that many of those fly-outs could instead become home runs and Zito will underperform his norm in that start. What other characteristics can be found in pitcher data that demonstrably affect a pitcher's success?

To understand how pitchers are analyzed, one must look at three tiers of statistics: primary, secondary and advanced.

3.1.1 Primary Statistics

Primary statistics are immediately observable results of single plays, and can be distributively aggregated. They can be split into two categories: result-dependent and result-independent. The primary statistics observed for the purposes of this project are:

Performance-Dependent Primary Statistics (PDP)

Innings Pitched (IP)	Hits Allowed (H)	Runs Allowed (R)	Earned Runs (ER)
Home Runs Allowed (HR)	Bases on Balls (BB)	Strikeouts (K, SO)	Batters Faced by Pitcher (BFP)

These statistics cannot be separated from the results, as they comprise the direct outcome of each pitch. Performance-dependent statistics must be handled more carefully than performance-independent, as they are inherently affected by exterior factors such as offensive abilities of the team faced, park altitude and temperature, and more. As better classifications of team "character" are developed, though, more valuable information may be extrapolated from these statistics as they are aggregated over time. In a neural network, the team will be considered as a single data point: thus, all factors will be implicitly considered.

Performance-Independent Primary Statistics (PIP)

Average Fastball Velocity (AFBV)	Pitching Handedness (PH)
Height (HT)	Weight (WT)

These measures are direct characteristics of a pitcher and should be consistent throughout the season (assuming no major changes in health) regardless of team faced.

3.1.2 Measuring a Pitcher's Performance: Secondary Statistics

Secondary statistics are algebraic functions that aim to provide direct and easily interpretable rates or averages based on the distributive, atomic primary statistics. They provide moderately granular representations of a pitcher's ability and are useful for evaluating a pitcher in a way that: is easily readable and understandable by a casual fan; fits nicely as an aggregate attribute in a typical statistical analysis or machine learning model; and can be altered (usually by adjustments for the league mean) to derive advanced statistics.

Earned Run Average (ERA)

$$ERA = 9 * \frac{ER}{IP}$$

The average number of earned runs relinquished by a pitcher per nine innings pitched (adjusted to nine innings as that is the typical length of a game).

ERA discounts runs allowed through defensive error or a passed ball, but doesn't take into account the quality of the pitcher's team's defense or the character of the stadium (some are more "hitter-friendly", and some are more "pitcher-friendly"). There are advanced statistical measures (discussed later) which attempt to make up for these shortcomings, but this provides a compelling example for why a neural network might do the job well - one can measure norms for defense and park factor, and adjust results accordingly.

Developed near the end of the 19th century, ERA was proposed to improve upon win-loss record as a measure of a pitcher's performance. In the early days of baseball statistics, the main concern was runs: a pitcher was measured by his ability to prevent them, and a hitter by his ability to earn them. While clearly an important factor, it was later proposed that those measures are too volatile and environment-dependent to use as the primary way to evaluate a pitcher. As sabermetrics came more into vogue, that baseball data analyst community move more towards factors that were a direct result of pitching performance: walks, hits and strikeouts.

Walks and Hits per Inning Pitched (WHIP)

$$WHIP = \frac{BB + H}{IP}$$

WHIP describes the total number of baserunners, regardless of whether via hit or walk, allowed by a pitcher. This offers a less abstract and more meaningful look at a pitcher's performance: a pitcher could give up multiple hits an inning but no recorded runs due to stellar defensive performances or plain luck, and end up with a low ERA. WHIP is an attempt to ignore all aspects of the game except the pitcher, the hitter and the result. While an improvement on ERA, WHIP is still not a perfect statistic: it counts home runs and walks equally, and does not factor in the batter being hit-by-pitch (which is fairly obviously a direct indicator of the quality of the pitch). Nevertheless, the impact of sabermetrics is clear in that the baseball world has begun to look at bases rather than runs: a pitcher is measured by bases given up and a hitter by bases earned, without even looking at final results of men on base (runs).

Strikeout Rate, Strikeout Percentage (K%)

$$K\% = \frac{K}{BFP}$$

Percentage of batters faced who recorded a strikeout. A simple but compelling statistic: while seemingly not as descriptive as WHIP or even ERA, K% is highly regarded by both scouts and sabermetricians: it can be argued that it represents one of the most important measures in evaluating future performance of a young pitcher. Nikolai Yakovenko, citing Bill James, treats this as the primary pitching statistic and expounds on that hypothesis in the following discussion. While it seems simple, K% may thus be a significant driving force in the network's implicit evaluation of a team's value.

3.1.3 A Deeper Look into Strikeout Rates

This concept is explored by Nikolai Yakovenko in his "Machine Learning for Baseball" article. [3] Yakovenko begins with two simple concepts:

1. He groups pitchers into eight categories, based purely on their primary and secondary pitch types, using k-means clustering with Manhattan distance
2. He asserts that strikeout percentage is the leading low-level indicator of a pitcher's success, and sets out to find a set of optimal "rules" associated with this statistic - but only those which are essentially unrelated to performance (in other words, we look at the types of pitches and player details without caring about the results of the pitches)

This focus on strikeouts is not unique; in fact, it is a concept mirrored both by team front offices and the godfather of sabermetrics himself, Bill James. James points out that a rookie pitcher's strikeout rate can be viewed as the most important determining factor affecting long-term success. In fact, James goes even further and boils it down to two "absolute facts... all good young pitchers with strikeout rates below 4.00 per game disappear quickly, [and] all pitchers who have long careers start out with strikeout rates in excess of the league average". [2] [p. 291]

Yakovenko's approach to finding association rules for strikeout rates is simple: feed in 30-ish features, apply M5-rules in Weka, and end up with a set of rules (linear and related to the input features) which have a strong correlation to success. [4] There are two primary indicative factors in predicting a pitcher's strikeout rate, and they present a high level of insight into how Yakovenko views the importance of statistics: average fastball velocity and "handedness". One important factor is already abundantly clear: Yakovenko is looking entirely at performance-independent primary statistics. As such, his tools are both easily interpretable and, presumably, easily projected: a pitcher can be expected to have roughly the same velocity regardless of who he is facing or where the game is, and it is unlikely that a pitcher in the majors will suddenly decide to switch his throwing arm.

Average fastball velocity is a wide-ranging statistic, considering that every pitcher throws some form of fastball regardless of their specialty or "pitcher category", and it's important to point out that this relationship between velocity and success is not linear. If a pitcher has a slower fastball (in the mid-high 80s or low 90s, for instance), he probably does something else well; he most likely focuses on good offspeed pitches and either gets swinging strikes or weak contact outs. If the analyst is looking at pitchers who average in the mid-high 90s, they can expect that those pitchers are all somewhat reliant on the fastball for, as stated, an indicator of strikeout success. As such, one can expect that a pitcher who relies on a 98 MPH fastball will probably be more successful than one who relies on a 95 MPH fastball. That doesn't give the same information regarding how they would compare to someone who averages 90 MPH but throws an effective offspeed pitch. The upshot is that fastball speed cannot be measured the same for all pitchers - this is where one sees the effectiveness of clustering pitchers into categories and applying rules differently based on where they fall.

The second relevant statistic for strikeout rates is a simple one: is the pitcher left-handed? Yakovenko found a correlation between handedness and K%, deducing that being left-handed earns a pitcher, on average, and extra 8.5 strikeouts per nine innings. [5] Additionally, it can be shown to have a greater impact on "hard-throwing" pitchers - those who throw faster, on average, than the threshold speed of 91/90.6 MPH for NL/AL pitchers, respectively. He concludes with a summation of the gathered data: "Most of the variance between pitcher strikeouts at the MLB level can be explained by two variables: fastball velocity and "is he a lefty"... and the pitchers with the same physical characteristics get consistently more strikeouts in the National League than the American League..." [3]

3.1.4 Measuring a Pitcher's Performance: Advanced Statistics

These measures are more complex, normalized and robust in their description of an individual pitcher's abilities and impact. They will be used as performance benchmarks but not as statistical inputs for the following reasons: they are designed to have easily interpretable outputs but highly specialized and, thus, essentially unreadable methods; they involve micromanagement, tweaking and low-level "judgment call" adjustments, all things this project is trying to avoid in its sanitized approach; they are highly regarded as one-stop statistics, and so may be more useful as validation for the model rather than as input. They derive from primary statistics measured against others in the league; this is, of course, what the neural network accomplishes on its own.

Fielding Independent Pitching (FIP)

$$FIP = \frac{13\,(HR) + 3\,(BB) - 2\,(K)}{IP} + C, \;\; C = \overline{ERA} - \frac{13\,(\overline{HR}) + 3\,(\overline{BB}) - 2\,(\overline{K})}{\overline{IP}}$$

FIP is just one of several defense-independent pitching statistics (DIPS), which remove entirely the factor of a pitcher's team's defense and instead calculate using only plays for which the pitcher was directly responsible. FIP, developed by Tom Tango [6], is the most highly-regarded and widely used of these statistics, so it is the only one this report will examine. Other measures, though possibly more accurate, are less ubiquitous and less well-documented: FIP is both widely available and easily calculable. It is also sufficiently robust: as Bryan Grosnick says: "With FIP, only those items that are considered fully within the purview of the pitcher are recorded, so there's less luck involved." [7]

The factor C is a slight adaptation on Tango's original implementation and, as described by Bryan Grosnick, is a method of adjusting the pitcher's base FIP relative to the league average FIP to "put FIP on the same league-wide scale as ERA". [7] It achieves this by adjusting for the league average (for ERA) and league totals (for everything else) for each measure. This factor is useful in normalizing our results when we want to test performance against established statistics, and its introduction into the formula provides a bridge between our secondary and advanced statistics. This is expanded on in the formula for xFIP.

Expected Fielding Independent Pitching (xFIP)

$$xFIP = \frac{13\,(xHR) + 3\,(BB) - 2\,(K)}{IP} + C, \;\; C = \overline{ERA} - \frac{13\,(\overline{HR}) + 3\,(\overline{BB}) - 2\,(\overline{K})}{\overline{IP}}$$

$$xHR = FB * \frac{\overline{HR}}{\overline{FB}}$$

The constant is the same as regular FIP, but the first term is different: rather than simply multiplying by the pitcher's HR against, we multiply by projected home runs (a factor of the league average HR/FB rate). [8] This is "an estimate of how many home runs they should have allowed given the number of fly balls they surrendered while assuming a league average home run to fly ball percentage". [9]

xFIP is "a regressed version of FIP that ... [can be] used to predict future pitching performance." [7] Where FIP attempts to remove factors that are out of a pitcher's control (such as defense, luck and park factors), xFIP expands on this to also describe a more normalized and less random evaluation of the pitcher's performance: there is a small difference in the variance of a pitch that leads to a 499-foot fly ball out versus the one that gives up a 501-foot home run when the wall is 500 feet out. Understanding this, a sort of data cleaning is performed to derive a more consistently

indicative model.

It is important to note that FIP is a common factor when measuring a pitcher's WAR and is utilized, for example, in the WAR calculation provided by Fangraphs [10] This is unnecessary complexity for this project's use-case, and as such the author will use xFIP as the highest level of performance analysis using defense independent pitching statistics.

Adjusted Earned Run Average (ERA+)

$$ERA+ = 100 * \left(2 - \frac{ERA}{\overline{ERA}} * \frac{1}{PF}\right)$$

(Note: PF = Park factor, an adjustment for average pitching performances in the given ballpark)

Adjusted Earned Run Average takes the normal ERA formula and alters it in two ways: it adjusts for the league average and park factors, and it puts the measure on a percentage scale where 100 is the "average" pitcher and each point above or below constitutes one percentage point above or below the league average for the observed pitcher. [8]

Game Score [9] (GS1, GS2)

$$GS1 = 50 + O + 2\,(\#IP - 4) + K - 2\,(H) - 4\,(ER) - 2\,(UER) - BB$$

$$GS2 = C + 2\,(O) + K - 2\,(BB) - 2\,(H) - 3\,(R) - 6\,(HR)$$

(Note: UER is a new initialism which just means "Unearned Runs", e.g. as a result of an error)

There are two versions of Game Score that will be utilized - the first was introduced by Bill James, the second by Tom Tango as an attempt to both improve the usefulness of the original and adapt it to the more modern, offense-heavy style of play.

The implementation of Bill James' original formula is straightforward and essentially just simple arithmetic. What it attempts to find is a measure of "how good was that start?", [9] and it does so by just aggregating the familiar statistics into some formula that fits on an easily interpretable 0-100 scale (barring outliers both below and above the scale, which are possible but rare and show an extremely abnormal start in either direction).

Tom Tango's alteration of the statistic offers us a priceless look into how a mathematician formulates advanced statistics by showing us the starting point, the finished product and, most importantly and thanks to a Fangraphs blog post, the process. [11]

For a variety of reasons explained in his blog, Tango lowers the starting constant from 50 to 40. The idea is to punish very short outings. He further adjusts this by allowing for it to be incremented or decremented in such a way as to make the league average always 50 - thus as pitching and offensive trends change, we can measure pitchers from different eras against one another. [12] This presents a much more important alteration to the formula than merely lowering the starting constant by ten.

A further adjustment made by Tango is increased importance of a walk: since a walk puts a player on base, Tango argues that it should be treated equivalently to a non-home run hit. This is in line with current trends in sabermetrics, where we are interested not in hits/walks/runs but rather in total bases (discussed more in the hitting section). Finally, Tango includes a specific penalty for a home run given up: he adds six penalty points on top of the two for a hit.

Tango observes that there is a close linear relationship between game score and wins: according to his data, "a pitcher that averages a game score of 65 will win 65% of the time." [9] It could be argued that this is moving against the goal established in xFIP, where analysts attempt to account for the fact that a home run and a flyball out are similar occurrences with notably different outcomes. However, the analyst is more concerned with outcomes than with reasoning. If game score truly represents win percentage in non-extreme cases, it surely must be considered as a valid statistical measure.

Jay Boice, writing for FiveThirtyEight [13], describes a similar use of game score to evaluate pitchers in their attempt to assign overall team rankings (a goal which is well inline with this project's). While the exact details are not disclosed, Boice gives a similar computation to the Tango version and adjusts for the era and park normalization. They also adjust for the offensive abilities of the opposing team as a season-long trend. FiveThirtyEight evaluates a pitcher's Rolling Game Score (rGS), "the model's best guess as to how the pitcher would perform in a typical start", and computes the pitcher's contribution to team performance as a factor of his rGS compared to the rest of the pitching staff: *i.e.*, an rGS above the team average means the pitcher is expected to positively impact his team, while a lower one implies that he will have a negative impact.

3.2 Evaluating Position Players

A team has a starting lineup of nine position players, which in the National League includes the pitcher but in the American League instead includes the designated hitter. As a given position player constitutes one-ninth of the team's total offensive power, it naturally follows than he could be said to offer less value to the team than a pitcher. The value he does add can be broken down into two categories: hitting and fielding. In other words, the team's offense versus the team's defense.

Fielding

Quantification of fielding ability is a difficult thing: many factors must be considered, and many of them are more subjective than what baseball statisticians are used to. For instance: how does one know if a ball should have been fielded by the center fielder or left fielder, if it falls directly in between their zones? Who gets the blame for that miss, or the credit for a catch? Most analytics associations have their own measures, but they are proprietary and complex and all that really matters is the results. To further the issues, any fielding evaluation requires some amount of geographical informational processing. There is one definite statistic that can be used to measure a fielder's value, though it has its own obvious problems: the **Error (E)**. An error is assigned to a fielder when he makes a mistake that prevents him from creating an out on the play. The glaring issue with errors is that they are subjective: the official scorekeeper must make a judgment call at the time of play. Generally the scorekeeper is "correct", and with relatively few concrete fielding measures the analyst must take what they can get; however, one must keep in mind this inherent flaw.

Hitting

The concern of a data analyst is the culmination off all pitches against a particular hitter in what we call either an at bat or a plate appearance.

At Bat (AB): a player's turn at the plate that results in a hit, a strikeout, reaching on a fielder's

choice (meaning the fielder threw the ball to get an out at a different base), an error (described in the Fielding section), or a bat-out (flyout or groundout). Notice that the AB stat does not include walks! This is a fundamental problem with the statistic: part of a hitter's prowess is his ability to "spot" bad pitches and not swing at them.

Plate Appearance (PA): a much more comprehensive statistic that essentially includes every time a player is at the plate. The only outcome not included in a PA is catcher's interference, which is a play that is unaffected by the hitter himself.

Most traditional statistics are given in terms of at bats, while advanced statistics and sabermetrics prefer plate appearances.

3.2.1 Primary Statistics (Attributes)

Hits (H)	Runs (R)	Doubles (2B)	Triples (3B)
Home Runs (HR)	Runs Batted In (RBI)	Bases on Balls (BB)	Strikeouts (K, SO)
Stolen Bases (SB)	Caught Stealing (CS)	Sacrifice Hits/Flies (SH/F)	Hit-By-Pitch (HBP)

3.2.2 Secondary Statistics

Like with pitching, the preferred overall evaluation of a position player's value has changed over time as new primary statistics are found to be more important.

Batting Average (AVG)

$$AVG = \frac{H}{AB}$$

The earliest method of deciding how "good" a hitter is; simply the number of at bats which result in any kind of hit. This statistic is obviously insufficient: it treats all hits equally and completely ignores walks, which are clearly an important aspect of hitting (as discussed in the at bat section). By the nature of defining the statistic in terms of at bats, an unacceptable amount of information is lost. AVG is a stat for the fans only.

On Base Percentage/Average (OBP/OBA)

$$OBP = \frac{H + BB + HBP}{PA}$$

Not a true percentage, OBP measures "how frequently a batter reaches base per plate appearance", or more eloquently, how effective a hitter is at avoiding an out. [8] OBP makes up half of one of the premier sabermetrics statistics, OPS (explained later), and in layman's terms gives an important bit of information: how many of this hitter's plate appearances resulted in earning a base. OBP only counts "earned" bases: it ignores bases from errors, fielder's choice etc.

Slugging Percentage (SLG)

$$SLG = \frac{H + 2\,(2B) + 3\,(3B) + 4\,(HR)}{AB}$$

SLG, an arguably imperfect statistic which certainly is not a percentage (its max value is 4.0), was developed "as a response to a noticeable flaw in batting average: Not all hits are created equal". [8] Where OBP measures percentage of appearances where a base was earned, SLG measures the total *number* of bases earned by a hitter.

The keen eye notices an immediate issue: based on at bats, SLG does not take into account bases earned by walk or hit-by-pitch. It is a measure of pure hitting power.

On Base Plus Slugging (OPS)

$$OPS = OBP + SLG$$

One of the cornerstones of sabermetrics, OPS is a simple arithmetic that attempts to give the full picture of a player's hitting skillset: how often does he get on base, and how many bases does he get? It takes into account how good a hitter is at laying off, how skilled he is at making contact, and how well he hits for power. It is clearly a much more sound and complete statistic than the batting average of the olden days, and has dominated baseball statistical analysis since its inception in 1982 in *The Hidden Game of Baseball* by John Thorn and and Pete Palmer - two godfathers of baseball statistics compilation. [14] [pp. 69-70]

The benefits of OPS are manifold and readily apparent: it is complete, interpretable, on an easily understandable scale, and consists of simple statistics without hidden normalization factors - thus, easy to reproduce for the casual statistician. Unfortunately, is is not the holy grail it may initially be seen as. One immediate concern is the differing denominators: OBP uses plate appearances since it records bases earned for any reason, but SLG uses at bats as it is concerned only with pure hitting. Additionally, the statistician is given two measures on separate scales with different interpretations of "good" and is asked merely to mash them together as a simple sum. As Grosnick stresses, these two statistics are not on comparable scales: OBP, which ranges from zero to one, is seen as "good" at around 0.350. SLG, on a scale from zero to four, has a much lower relative "good" threshold at around 0.430. It is evident that a player's OPS will predominantly come from his SLG, and thus a power hitter will be depicted as more valuable than a contact hitter. [15] Unfortunately, SLG is not more valuable than OBP: in fact, it has shown by Tom Tango to be worth less. In his attempt to derive a more usable compound statistic, Tango suggests that OBP is 170% as important as SLG [16]. Even then, other factors must be taken into account when deriving a player's value added from those numbers. Baseball statisticians like linear weight models, and the usage of OPS does not work well with that methodology. While it is special in that it is both easily interpretable and highly available, the OPS statistic should not represent an analyst's statistical zenith: it is not a measure for the world of advanced baseball data.

Runs Created (RC)

$$RC = \frac{(H + BB - CS + HBP - GIDP) * (TB + (0.26 * (BB - IBB + HBP)) + (0.52 * (SH + SF + SB)))}{AB + BB + HBP + SH + SF}$$

(Note: TB = Total bases earned)

A clearly manageable and lightweight statistic, RC was devised by Bill James as an early attempt to present a concrete measure of a player's value. James elucidates perfectly what an analyst's

concern should be when it comes to defining offensive performance: "With regard to an offensive player, the first key question is how many runs have resulted from what he has done... his job [is] not to hit doubles, nor to hit singles, nor to hit triples, nor to draw walks or even hit home runs, but rather to put runs on the scoreboard..." [2] [pp. 273-4] While RC has the obvious issues in that it does not account for playing time or league trends, it provides an excellent baseline to find a measure of how valuable a player is.

3.2.3 Advanced Statistics

Like with pitchers, these advanced statistics aim to normalize player value based on the era, park factor and league average at the time.

On Base Plus Slugging Plus (OPS+) [17]

$$OPS+ = \frac{OPS}{\overline{OPS} * ParkAdjustment} * 100$$

(Note: ParkAdjustment is the same concept as PF, but this project gives the official representation of the OPS+ formula and so ParkAdjustment is used.)

OPS+ normalizes a players OPS to the league average with park factor taken into consideration and puts it on a percentage scale, where 100 is the league average and a player with an OPS+ of 150 is 50% better than that. [8] OPS+ has not gained as much widespread usage as OPS, but attempts to alter it in some way to be more representative. As it accounts for league average, it is especially useful in measuring a player's seasonal trends. Though technically better than OPS, OPS+ is more proprietary and the $\overline{OPS}$ value is calculated differently depending on the provider. It also has a holdover problem from OPS: it weighs OBP and SLG incorrectly. [9]

Weighted On Base Average (wOBA)

$$wOBA_{2018} = \frac{0.69\,(BB - IBB) + 0.72\,(HBP) + 0.88\,(1B) + 1.247\,(2B) + 1.578\,(3B) + 2.031\,(HR)}{AB + BB - IBB + SF + HBP}$$

[18] [2018 weights]

Like OPS+, wOBA uses linear weights-adjusted statistics to measure the offensive output of a player normalized to the league average. Unlike OPS+, it tunes those weights to give true values of each result in comparison to the others (for example: it, unlike OPS/+, does not treat a double as twice as valuable as a single, because that is not the case). Also unlike OPS+, unfortunately, it does not adjust for park factors or era trends. [19] Regardless of this, wOBA offers an immensely useful offensive measure by building on the core strengths of OPS/+: "OPS is asking the right question, but [with wOBA] we can arrive at a more accurate number quite easily" [9]

To obtain a measure of a player's added value in runs (i.e. compared to the "average" player), the analyst may merely convert wOBA to **Weighted Runs Above Average (wRAA)** [20] with this formula:

$$wRAA = \frac{\left(wOBA - \overline{wOBA}\right)}{wOBA\ \ Scale} * PA$$

This project is only concerned with wRAA as a component of other formulas, discussed later, as there are more complete and effective normalized offensive statistics. wOBA is almost the perfect

offensive statistic, but one which can normalize the data for era and park would be the holy grail.

Weighted Runs Created Plus (wRC+)

$$wRC+ = \frac{\left(\frac{wRAA}{PA} + \frac{\overline{R}}{\overline{PA}}\right) + \left(\frac{\overline{R}}{\overline{PA}} - PF\left(\frac{\overline{R}}{\overline{PA}}\right)\right)}{\frac{\overline{wRC}}{\overline{PA \backslash PA_{Pitchers}}}} * 100 \quad [9]$$

wRC+ is an adjusted version of wRC, which this project will not consider. wRC is a cumulative statistic that builds over a season and, as discussed with previous cumulative statistics, is thus less useful for getting an idea of how many plate appearances a player had. It also, of course, does not translate well across eras or parks. wRC+ is a solution to both of these issues, and provides a league-, era-, and park-adjusted measure of offensive performance: "take a player's wOBA, add it with what a league-average player would be expected to generate per plate appearance, combine that with constants that have been ascribed to the different parks the player has performed in, and then divide that by the AL or NL's average wRC/PA." [21] Like other normalized statistics, 100 is the league average and each point is a percentage point away from it. The C value is an adjustment for the league average wRC of that year rated against number of plate appearances (excluding those by pitchers).

wOBA is a fantastic measure, but wRC+ demonstrates what is truly important to this project: since a century's worth of data is being processed and the league has of course undergone a multitude of changes, it is fundamental to the success of *Machine Learning and Baseball* that these outlying factors are taken into account. A player's statistics must necessarily be adjusted if a real observation of his value to a team can be measured, and simply using statistics such as wOBA and OPS will not address this issue. For this reason, wRC+ will be a foundational statistic for player offensive evaluation in the project (as it is in broader scope of baseball analytics).

3.2.4 Wins Above Replacement

Composite evaluation of position players in terms of offense and defense is as messy as the statistics at the analyst's disposal, but the generally agreed-upon measure is **Wins Above Replacement (WAR)**. This is a single number that "measures a player's value in all facets of the game by deciphering how many more wins he's worth than a replacement-level player at his same position" [8] It is certainly not a definite and concrete statistic: "Given the nature of the calculation and potential measurement errors, WAR should be used as a guide for separating groups of players and not as a precise estimate." [9] Comparing two players' WAR becomes less effective the closer together their values are. WAR is calculated differently depending on the source: the running theme, though, is that all implementations are long, complex, proprietary and mostly private. The general formula is:

$$WAR = \frac{BattingRuns + BaseRunningRuns + FieldingRuns + PositionalAdjustments + LeagueAdjustment + ReplacementRuns}{Runs\ per\ Win}$$

In short: it measures how many runs a player is worth in all aspects of the game, tunes that based on the league average and the expected output of that player's position, compares that amount to the expected output of some league-average replacement player, and adjusts that value based on how many runs are expected to result in a win. [8] [9]

As WAR is both built from subjective statistics and inconsistent between implementations - as well as being largely unavailable compared to other measures - it is a poor choice for serious statistical

analysis of a player's value added when working with far-reaching data. However, it cannot be ignored that WAR is a major factor teams look at when placing value on potential additions: regardless of method, the end result is effective. As a member of team administration, a talent scout or even just a fan of the game, "WAR estimates a player's total value and allows us to make comparisons among players with vastly different skill sets. Who is better, a slugging first baseman or a superlative defensive shortstop? WAR gives you a method for answering that question." [9]

In essence, wins above replacement is what *Machine Learning and Baseball* is attempting to replace. While adored by the baseball community, WAR is a clunky and imprecise statistic that leaves a lot to be desired: it throws implementation interpretability out the window and essentially just allows an analyst to feed data into a black box and get a result on the other end. Nevertheless, it has become the premier measurement of value for positional players. If player evaluation has become simply feeding attributes in, normalizing and getting a statistic out, without a clear idea of how that result is calculated, why not just make that black box a neural network and let the tuning occur without direct intervention?

3.3 Putting it all Together: Evaluating Teams and Predicting Games

With these statistics, an analyst now possesses a veritable toolbox from which to pick those they need and tune them to their heart's desire. The obvious prediction implementation would be to simply aggregate league- and era-adjusted values such as xFIP and wRC+ together into a cumulative statistic through simple algebra, yielding a measure of "how many runs this team can score, and how many they can prevent". This would of course produce a simple linear model based on the statistics, and may be sufficiently effective. This project will ignore all of those models but one, as they offer no new insight beyond what has been discussed. Instead, more advanced machine learning implementations will be analyzed. The one exception is PECOTA, a prediction model by Baseball Prospectus which is too important and relevant to skip.

PECOTA: Player Empirical Comparison and Optimization Test Algorithm

PECOTA is a fully computerized and proprietary prediction model for projected player value. With largely secret implementation details and no baseline to measure against, there was not much reason to discuss it in the player evaluation section. However, as a guideline of important factors to consider when normalizing player impact projections and extrapolating that data to team performance predictions, PECOTA provides valuable insight into what factors the "experts" think are most important.

The low-level details are an organization secret, but the basic elements of a PECOTA rating are: minor league normalization to project a player's performance if he hasn't played in the majors; baseline forecasts to estimate a player's true talent level using weighted averages and regression to the mean; and an adjustment based on comparable historical careers to predict how a player's output will change over time. [22] The first detail is not important to the project: the main points here are in how to normalize a player's performance to account for career trajectory, regression to the mean and adjustments from the rest of the league. Another note: as simple as this model is, it is "practically as accurate as one with far greater complexity". [23] As FiveThirtyEight points out, a complex model isn't necessary as long is at can find that special detail that sets it apart. [23] In PECOTA's case, this detail is career trajectory.

Once PETOCA analyzes statistics for each player, it feeds those "into team-depth-chart projections to generate win predictions for every team". [23] PECOTA performs with a root-mean-square error of 8.9 wins, 2.5 away from the best prediction possible at 6.4 (more on this later). [23] Though the precise implementation of this statistical measure is secret and proprietary, the results indicate that it should be considered in any comparisons as a significant baseline for the project's performance.

Run Differential and Wins [24]

Gary Talsma, quoting Bill James, begins his analysis with the notion that "the objective of every baseball team and player is to do things that contribute to winning games while avoiding actions that contribute to losing games". [24] Talsma, then, takes an elegant and visual approach to the problem: he simply gathers a team's statistics and plots them against wins in a search for linear relationships. His approach expresses a similar issue discussed in the WAR section: as he says, "to succeed in baseball, a team must play well both offensively and defensively. Sabermetric investigations have revealed that the key to predicting wins is indeed a combination of good offense and good defense." [24] Thus at the most basic level, since the main concern in sabermetrics is runs, the simple answer is to just plot run differential (how many runs a team scores versus how many they allow) against wins. As expected, this plot portrays a clearer linear relationship than other attributes tested such as hits and home runs. Talsma acknowledges that of course this relationship is obvious for a single game: a team that scores more runs than they give up will win that game. The interesting note is that this rule "should be true, generally speaking, for an entire season." [24]

Over the course of a season, a team's overall wins can subsequently be found as some "linear function of run differential". Talsma uses least-squares regression to plot this function against the scatterplot (unfortunately, while the text of this article is available, the graphs are not). Talsma derives from this model an equation showing wins as a function of run differential in a simple linear relationship: $W = .088(RDIFF) + 80.964$. Essentially, with about 81 wins as a baseline (a record of .500), the findings show that scoring one more run than an opponent results in winning 88% more of a game. The crux is that one can, for example, "set W equal to 90 and solve the regression equation for RDIFF to find that a team must score about 102 runs more than it allows to reach the ninety-win level." [24] Such a function, paired with expected divisional opponent wins, should inform a team how many wins they need to secure a playoff spot and what run differential is required to achieve those wins.

The unseen plots, as described by Talsma, do show some breakdowns in the data: in the timeframe he observed, twenty-two of the twenty-eight teams' records fell within five wins of the prediction. There will of course be some volatility in outcomes of any sport, and the best an analyst can do is attempt to assign probabilities for success in these "random" samples. Talsma, echoing James and a widely-held truism in baseball, states: "teams that win substantially fewer games than expected, on the basis of the numbers of runs that they score and allow, are teams that do quite poorly in close games, whereas teams that outperform expectations tend to win more than their expected share of close games." [24]

Talsma's analysis is more reflective than projective: while it is useful to calculate a mapping from offensive and defensive output to run differential and from that to wins, he does not tackle the issue of how to predict a team's offensive and defensive performance. The best approach for this model may be to simply project a team's players' wRC+: as this statistic normalizes for the league average, one may predict the league average as a trend of recent history and calculate expected wRC+ as xwRC+. Even then, defensive performance must be taken into account. The good news is that environmental factors such as temperature trends, ballpark dimensions and "manager effect" should be easy to project for normalization. Additionally, there are several fairly trustworthy ways to handle pitcher projections: Yakovenko provides just one approach in his discussion of strikeout rates and pitcher handedness, both measures than would ostensibly remain fairly consistent from

season to season. [4]

Bayes-ball: A Model for Performance Projection [25]

The article "A Two-Stage Bayesian Model for Predicting Winners in Major League Baseball" by Tae Young Yang *et al.*, hereafter referred to as Yang, proposes a different approach to performance projection. In Yang's words, this model considers "past performance of the two teams, the batting ability of the two teams and the starting pitchers" as a single measure, combined with a factor of home field advantage, as the basis for a two-stage Bayesian model. [25] For past performance, Yang uses the ratio between the two teams; for batting ability, aggregate batting average; and for pitching, aggregate ERA. Clearly there is issue with this approach, as the shortfalls of AVG and ERA as value measures have already been discussed, but perhaps the overall model could be tweaked to use wOBA/wRC+ and xFIP instead.

Another key aspect, and one which possibly should not be thrown away so readily, is the disclusion of team defense as a contributing factor. From an anecdotal standpoint, this is not entirely farfetched: barring maybe one or two outliers per team in the negative or positive, there really is not much variation in defensive abilities. One would expect a fielder to make catches in his zone and to have a decent arm (depending on the position). Nevertheless, maybe an aggregate error statistic could be incorporated in some way to at least give a perfunctory representation of defensive value.

Yang assigns each ratio factor a contribution parameter based on that measure's impact on the overall performance; he points out what was discussed earlier, that "[starting pitcher] is more important in determining the probability of a win than [past performance] and [offensive ability]." [25] The contribution parameters are constant across all teams, and the home field advantage factor is *a priori* independent of them.

With those parameters, Yang proposes his two-stage Bayesian Model. In stage one, those parameters form a beta distribution from which a team's win probability is taken as a random sample. For stage two, the outcome is taken as a random sample from the Bernoulli distribution of this win probability. From this probability distribution, the model uses a Markov Chain Monte Carlo algorithm to sample and simulate future outcomes; eventually, division winners are picked. [25]

Yang gives the "relative strength of the home team over the visiting team at time s" as:

$$\lambda_S = \alpha_S^{r_1} * \beta_S^{r_2} * \gamma_S^{r_3}$$

where:

$$\alpha = team\ winning\ percentage\ ratio \tag{3.1}$$

$$\beta = team\ batting\ average\ ratio \tag{3.2}$$

$$\gamma = ERA\ ratio\ of\ starting\ pitchers \tag{3.3}$$

$$r_i = contribution\ parameter\ of\ associated\ ratio \tag{3.4}$$

This gives a single metric for relative strength, λ_i; it is important to note that this value "is flexible as it readily accommodates other factors..." [25] Home field advantage δ is assigned a uniform distribution $(\delta_0 = 0, \delta_1 = 2)$.

The final model for prediction given by Yang is:

$$Prob\,(X_S) = p_S^{X_S}\,(1 - p_S)^{1-X}$$

where:

$$p_S = Prob\left(X_S = 1\right) \sim beta\left(\lambda_S \delta, 1\right))$$

Three models are presented, but this is determined to be the best as it strikes a balance between expressiveness and avoidance of overfitting. Exact results are given in an unviewable table, but Yang elucidates that "22 of the 30 predictions from July 30 are closer to the final winning percentages than the current winning percentages", "the predictive ability... is superior to simple extrapolation" and while "the model tends to exhibit a regression towards the mean effect", "the results... are in accord with prevailing wisdom that suggests that starting pitching is a key determinant at prediction." [25] This is the key takeaway from the paper: starting pitching is not to be discounted.

An Elo-Based Predictive Model

FiveThirtyEight are the kings of prediction: their team has developed predictive models for sports like American football, basketball, tennis and even predicts political trends in the United States. Luckily, as of 2015 they also predict outcomes of Major League Baseball. Even more luckily, they pull historical data from Retrosheet and expanded their model to measure and train on team trends since 1871. This formula is the brainchild of Nate Silver and the staff of FiveThirtyEight (namely Jay Boice), and is the premier predictive model in Major League Baseball.

The measure is calculated in terms of an oft-implemented method of ranking competitors and assigning them single performance measures: the Elo system. An "Elo" is a simple one-stop value given to, in this case, a baseball team, which describes some measure of how that team could be expected to compare to another. Nate Silver describes the baseball-oriented implementation of Elo in an article on Baseball Prospectus [26], but the essential idea is that it is a conglomeration of parameters that are tuned based on hyperparameters which aim to maximize predictive power trained from past data.

In his description, Silver raises several points that are quite applicable to the *Machine Learning and Baseball* project. Namely: how quickly should rankings change; how should margins of victory and homefield advantage be accounted for; and how do these ratings carry over between seasons? His actual solutions are not as applicable insofar as they are implementation-specific tweaks, but his ideas for seasonal carryover are important to observe. He splits the difference between "starting from scratch" and "completely carrying over ratings", and splits the difference by halving the distance between a team's season-ending rating and the league mean (in Elo's case, 1500). The important fact is that these rankings are "self-correcting. If your rating is low, then it's easier to gain points, and harder to lose them, and vice versa if your rating is high." [26] Upon completion of a game, some amount of the losing team's Elo points are assigned to the winning team at rate decided by the other factors. [27] An expected result means less shift, while a large upset leads to greater differences.

Jay Boice at FiveThirtyEight expands on this simple formulation, with a general game-by-game prediction model outlined as:

[28]

He assigns a specific adjustment value to each of these: home field advantage is worth twenty-four points, travel is calculated as $(miles)^{\frac{1}{3}} * -0.31$, and each day of rest is worth 2.3 points (maxed at 6.9). [28]

Starting pitcher is a much more important factor, and Boice calculates it as a slight adjustment on Tom Tango's Game Score v2.0 (discussed in the "Pitching" section):

$$gameScore = 47.4 + K + 1.5\,(O) - 2\,(BB) - 2\,(H) - 3\,(R) - 4\,(HR)$$

He then applies a proprietary transformation to normalize this score based on era and stadium, and adjust based on the opposing team's offensive strength. [28]

Game Score, even this altered and normalized version, is a reflective statistic: it tells you about how the pitcher *performed*. Boice uses this only as a tool in his broader projective adjustment, **Rolling Game Score (rGS)**. This is an averaged value recording across a pitcher's season and career. Boice also maintains a general team rGS, and derives a pitcher's Elo adjustment as "his rGS relative to his team's rGS; pitchers who are better than the team's rGS give the team a bonus when they start, and pitchers below the team's rGS give the team a penalty." [28]

With that, a pitcher's value added to his team is calculated as:

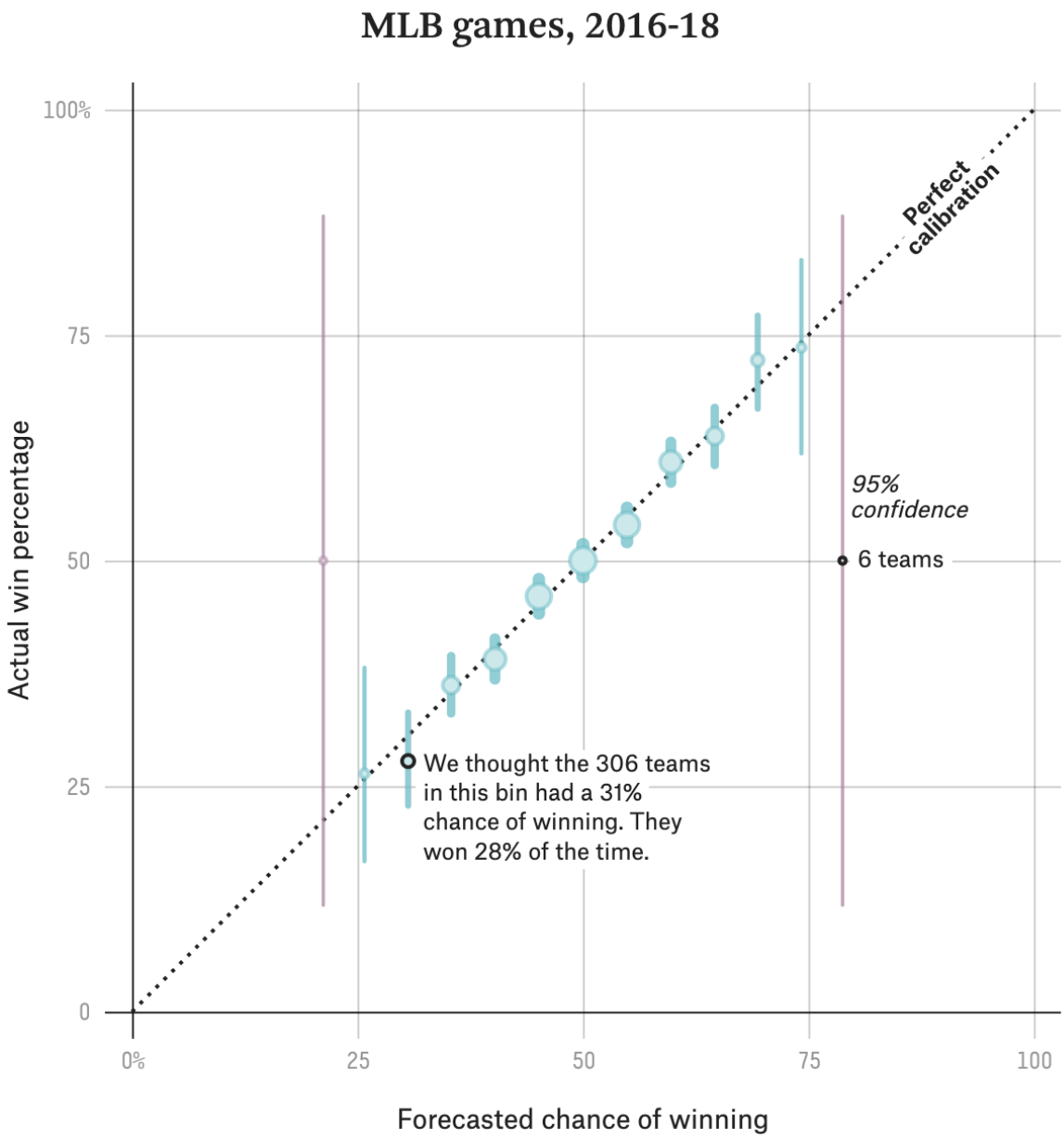
$$ratingAdj = 4.7 * (pitcher\ rGS - team\ rGS)$$

[28] This adjustment gives the predictive model "about a 1 percentage point improvement in the percentage of games correctly 'called'...". [28] Clearly, a pitcher's impact is important and the handling of that data is a significant factor in the projection of a team's performance.

Boice also handles seasonal transitions differently from Silver: he combines (computerized) pre-season win projections from three sources with the team's Elo from the previous season, reverted to the mean by one third. [28]

With this model in place, it is time to make predictions. Since what they are dealing with is a collection of probabilities, the season is played out thousands of times using Monte Carlo simulations; the ratings change in between each iteration to reflect the team's performance in those 162 games. At the end, each team has a final record out of 162 games and all that is left to do is watch the season play out and see how the model fared.

In June 2019, FiveThirtyEight released a retrospective discussion of the accuracy of their forecasts. Their probabilities tend to be closely clustered around 50%, and as they say: "We're not trying to pick winners, though; we're trying to model the games, which means including in our predictions all of the randomness inherent in baseball." [29] Obviously any real measure of success must be calculated over long periods with as much data as possible: Boice takes their 2016-2018 predictions, bins them with equal widths of 5% probability, and plots those estimated win percentages against actuals. The result [29]:



The data is promising, to a degree: their prediction model is "very well-calibrated" [29], but as stated before generally assigns win probabilities very close to 50% - almost random. Compared to a coin toss, though, their models perform better (just barely).

Compared to other sports, though, baseball just isn't easy to predict. A fantastic team can hope for about a 100-win season: that still means losing 38.3% of their games. This is why many analysts points out that baseball isn't a sprint: it's a marathon, and that fact has important implications for any predictive model application.

3.4 Measuring Success

What constitutes a successful predictive model? What is the upper limit of success? This can be found with simple statistics. Nested somewhere in a long discussion at Inside the Book [30], Tom Tango finds that, on average, one-third of a team's games will be decide by luck. So if one assumes

a middle-of-the-road team, this team can be "guaranteed" (for statistical purposes) that they will win one-third, lose one-third, and split one-third. That leaves a definite record of 54-54, with fifty-four games left to chance. In assuming that those games are split, a team will go 81-81. If, as Phil Barnaum posits, one can "know" that a .500-level team will split those chance games and go 81-81, then the outcome of any game is equivalent to a coin flip. [31] Thus, 162 results are chosen with a probability of 0.5. Taking the probability mass function of these random variables yields the formula $162 * 0.5 * 0.5 = 40.5$, giving a standard deviation of 6.4 games. This limit of about 50% predictability in any given game is supported by Boice's work at FiveThirtyEight discussed previously: an analyst may beat random chance when predicting outcomes, but nobody has been able to do it by much. And even if they do, the predictions are mostly done as probabilities where the winning team does not have much more of a win probability than the losing team. [29]

And thus, *Machine Learning and Baseball* is given its measure of success: the closer to a 6.4-game error it gets, the better. In theory, over a long stretch of predictions based on historical outcomes, that 6.4-game error should be insurmountable; however, if it can surpass an 8.9-game error, it can be said to perform better than PECOTA - the premier player-statistic-aggregate prediction model in baseball projection analytics. If it can maintain that accuracy with predictions beyond the 40%-60% win percentage threshold outline by Boice [29], it can compete with FiveThirtyEight's Elo-based prediction model. Given some level success relative to these milestones, Machine Learning and Baseball will be among the upper echelon of baseball prediction models.

Chapter 4: Data Considerations

A major motivating factor for data analysis projects centered around baseball is the wealth of informative and mostly-complete "official" data (Retrosheet points out that, in this case, "official" does not necessarily correlate with 100% accuracy [32] - it just denotes the data as that collected and distributed by the league office). All statistics were gathered by official score-keepers in real time and curated and validated by the league office, and have been digitized and made widely available by several conglomeration and curation organizations.

Retrosheet is the primary source of baseball statistics for data analytics, and this project will use their Game Logs dataset. [33] All games have been transcribed, with varying degrees of completeness, since 1871. This project will focus only on data from the last century for two reasons: records are less complete before the turn of the 20th century, leading to more inconsistencies and necessitating more complex preprocessing of the records; and there was a fundamental shift in how baseball was played in 1919 (this zeitgeist shift is covered in more detail elsewhere in the report) which correlates to a vastly different set of statistical benchmarks which are both out of date and not useful for projecting future trends and performance. In short: a team from 1890 was so different from a team in 2019 that it's not worth the effort preprocessing old and inconsistent data in an attempt to glean some amount of useful information from those past records. Disregarding those first forty-eight years, the data being used should be robust enough for any kind of machine learning, even a convolutional neural network. The analyst has at their disposal a mostly complete collection of 100 years of 162 (since 1920) games between sixteen and thirty teams (depending on the year) yielding modern-day totals of 2,430 games a year.

As such, one is left with the sum total of all relevant baseball-related data to query, analyze and manipulate as much as one would like. The game log files give, for each game played in a season, 161 attributes describing all team-, park-, and league-based statistical measures needed for any sort of analysis on the data. [34] These game logs are available as plaintext files with no headers, so the first step is to write a simple Python script to convert them to comma-separated value (csv) files combined with a header listing obtained from Retrosheet's associated list. [34] While performing these transformations in DataFrames, irrelevant features will also be eliminated: things like Protest Information (column 16), Additional Information (160), and all umpire data (80-89). At this point the project will contain a data store of relevant features with associated headers which can be written to SQLite and/or .csv files. There is one more consideration before writing, though, as this Retrosheet data is missing a key factor: individual performances of players.

To fill this gap, Retrosheet's data will be combined with a companion dataset called the Lahman Baseball Database (LBD). [35] This set complements the Retrosheet data by providing biographical information and career statistics for players and meta-information for parks and teams. Luckily, the LBD matches up with the Retrosheet data via a foreign key on players - the column retroID in the People table of LBD refers to the player IDs utilized by Retrosheet data, so that provides an entry point into matching up player biographical statistics with career data and the related team-based performance on a game-by-game basis. The LBD contains as well additional data that may be used in the more advanced feature set of this project: tables containing information on managers, parks, teams and fielding statistics. The post-season data is out of scope of this project, as season-long trends being evaluated have less projective validity in more microscopic post-season series.

Retrosheet and the LBD both began as projects to digitally compile, transcribe and curate past hand-written game log information taken down by official scorekeepers. Naturally, due to both

human error and archaic processes in early 20th-century scorekeeping, there are expected to be some inconsistencies and discrepancies in that data. Both organizations, through separate means, have attempted to clean and process all issues as much as possible; however, the process is ongoing. As these records are currently the most consistent and accurate ones available, they will be treated as the final word. This is not the view solely of this project's author: all serious and important data analysis in baseball that the author has found is done with data compiled from these two sources.

By Retrosheet's account in 2018, they had "computerized 84% of the games played" between 1902 and 2012, [36], though this completeness is of course skewed more towards the later years: luckily, many of the gaps are in the first seventeen years with which this project not concerned.

The LBD is offered under a Creative Commons Attribution-ShareAlike 3.0 Unported License, and thus may be copied, redistributed, transformed and adapted in any medium and for any purpose - including commercially. [35] Retrosheet allows that users "are free to make any desired use of the information, including (but not limited to) selling it, giving it away, or producing a commercial product based upon the data" as long as they are properly cited (see the project readme for this citation, as well as one for the LBD). [33]

Once the data has been aggregated and integrated into a singular store, it may be thought of as a static OLAP data warehouse: updates and deletes will be nonexistent and creation will only occur when new records become available at the end of a season. During development it will be stored as simply as possible, as SQLite tables. Once the web application portion is implemented, that data can be moved to a PostgreSQL database if necessary. Pre-querying will be necessary before feeding data into the network for such applications as finding all players who played for a particular team from an entry in the game log and aggregating rows from the LBD by playerID into career representations for a given player. All data integration will occur, as should be possible, at this point. What will be written to the final store is a collection of integrated, structured, robust and purely relevant data that will be optimized for efficient analytical querying.

One would typically expect serious neural network evaluation to require lengthy training and testing, thus possibly necessitating exportation of the computation to more robust processors on remote servers. As the baseball dataset is, ultimately, not as big as what is normally used in these instances (in terms of depth or width), this step presumably will not be necessary: all data analysis and storage should be possible locally.

Chapter 5: Research Summary and Conclusions

The goal of Machine Learning and Baseball is clear: discover the most relevant individual statistics; normalize them appropriately by era, league and park to ensure consistent training over the entire dataset; aggregate those measures into a cohesive and representational team measurement; train this formulation on the past one hundred years of baseball games to train and tune it; and eventually derive a prediction model to project team performances of future games. The last step will be to provide a simple user interface to represent the results of the project.

This report has examined a selection of the multitude of statistics and evaluations put forth to measure past performance and predict future success of players and teams. The golden standard has been found: if this model can find some heretofore undiscovered measure or combination of measures and use it to either approach or match the suggested mathematical limit of 6.4 game error on a consistent level, it will be deemed a major success. PECOTA reaches 8.9 error: attaining that level will of course also be considered a success. If this model achieves its goal with some formulation that aggregates individual player impact in a measurable way, perhaps it can also tell statisticians something new about measurement of player impact and value in the game of baseball. And perhaps more importantly, it will do this with an approach that has so far seen little success or even use in baseball statistics: the deep neural network.

Chapter 6: Data and Context

6.1 Data Sources

The primary source of data for this project (given the abandonment of the plan to run season simulations) is the Sean Lahman baseball database. The data used came in four tables: People, a metadata table with player names, ID numbers, height and weight information, handedness data and more; Batting, which gives batting statistics for all players per year; Fielding, which gives fielding statistics for all players per year; and finally, Pitching, which gives pitching statistics for all players per year.

The information in the Sean Lahman database is offered through a Creative Commons Attribution-ShareAlike 3.0 unported license. Under this license, users are free to use, share, and adapt the data in any way - even commercially. This data is collected and maintained by Major League Baseball and additionally curated by the Sean Lahman team; it is widely regarded as the best, most accurate and most complete representation of all baseball data.

Game logs were obtained from Retrosheet and are offered with open availability; users are "free to make any desired use of the information, including (but not limited to) selling it, giving it away, or producing a commercial product based upon the data". [33]. These game logs comprised a large amount of information pertaining to every game played every season, including: date and time; teams, players, managers and umpires; line scores and team statistics; park and league data.

6.2 Data Handling and Processing

While baseball certainly has significantly more associated data than other sports, as it is a heavily statistic-driven game, the amount still pales in comparison with many neural network-driven problems. This is a major consideration in both the absence of baseball-related neural networks projects and this project's attempt to derive a novel model solution where it seemingly should not be applicable, but this issue is discussed elsewhere. Discussion here will pertain to the time cost benefits of working with a relatively small dataset using Python, Pandas and TensorFlow.

Numbers of examples do not grow beyond the tens of thousands, and feature sets can be counted as tens rather than hundreds. It is sometimes necessary to derive new features based on other information (described below), and when this is the case it is somewhat trivial to map those calculations along the entirety of the dataset.

The People metadata required data preprocessing: handedness and league were one-hot encoded and shrunk to one column; also, height and weight were normalized using Min-Max normalization to maintain a scale of difference while putting the values more in-line with the rest of the statistics.

It was stated in the previous report that a major contributing factor to the decision to work with baseball data was the fact that it is well-maintained and relatively "clean" compared to many datasets. The thought was that the preprocessing steps would be relatively quick and most of the time would be left to testing and evaluating the model. To say nothing of the extraordinary

circumstances around the development of this project against the backdrop of a worldwide pandemic, the author fell into the all-too-common trap when working with data: seemingly without fail, preprocessing takes significantly longer than one would think. While most of the data was in the right form and not missing, there were a few spots where a significant amount of work needed to be done.

The feature values for the Lahman tables were mostly complete, with the majority of missing information coming in the earlier years of the data range (specifically, pre-1950). Given that baseball has changed a lot since then and the fact that it is somewhat simple to extrapolate approximations based on other, present data, this missing information did little to hamper the project's progress. A guide to the exact steps taken in cleaning and normalizing the data from Batting, Fielding and Pitching can be found in the Notebooks addendum, which shows a walkthrough of the process used.

6.3 Position Player Data

The majority of tensor inputs for batting comes in the form of primary statistics drawn from the Lahman Batting table. A number of advanced statistics, drawn from Fangraphs [37], have been added as deemed appropriate. Through research and deliberation outlined in the batting section of this report (section 3.2.3), these advanced, regressed statistics have been chosen as primary indicators of a position player's team contributions.

A large portion of missing data could easily be filled in as zeros, but there were some statistics which necessitated a great deal of thought to avoid swaying the model's outcomes. In some cases, for example with the "caught stealing" feature in Batting, it was necessary to derive a value based on both a player's likeliness to attempt to steal a base and his historical success in doing so. In this particular example, it was found that a player is generally twice as likely to be successful as he is to get caught. Since there is no statistic keeping track of total attempts to steal, it was decided that this feature would be populated as "number of stolen bases divided by two". Of course, this method leaves something to be desired: the important considering is that this "fudging" of the data only needed be performed for the first forty-one years in the dataset. After 1960, there is no missing information for the feature. As stated earlier, this project is more concerned with statistics after the 1950s than it is with statistics before them. If the missing information accounted for a significant portion of the total example set, this approach may not be sufficient: as it is, only about 8

This represents an example of how data preprocessing and cleaning was handled. Full walkthroughs of the processes taken can be found in the supplemental materials - preprocessing PDFs showing the exact steps taken in Jupyter notebooks.

Where additional statistics were required, the calculations for these were put into their own functions to be generated based on primary statistics and used throughout the application. It was necessary to construct advanced statistics programmatically, and where these were league-regressed the required information, such as league average values by year, was drawn from Fangraphs 'Guts'. [18] This information was essential for the derivation of weighted statistics such as weight on-base average (wOBA) and weighted runs created plus (wRC+). These calculations are processed in 'batting_stats.py', but the information is also available on Fangraphs [37]. There is in important distinction between the available data and the calculation methods employed by this application: regressed statistics like wOBA and wRC+ take 'park factor' into account, but the application's versions do not. Park factor is an indicator of the effect a park's character (dimensions, temperature, average performances) has on a player playing in that park. This project aims to, eventually,

feed in this park data as a separate tensor input when simulating games (a feature beyond the scope of the post-COVID project goals), and thus park factor is unneeded and unwanted in these calculations. As such, the approach taken is to simply calculate regressed statistics merely without the park factor weighting.

6.4 Pitcher Data

Pitching data was more readily available, which made programmatic construction of advanced statistics largely unnecessary. Almost all pitching statistics were taken from Fangraphs [38]; the core numbers used to derive a rating system were ERA (earned run average), FIP (fielding-independent pitching), IP (innings pitched), K $K\% = \frac{K}{BFP}$ Strikeout rate is a non-regressed secondary statistic, but its inclusion and perceived importance is detailed earlier in the report (section 3.1.3: A Deeper Look into Strikeout Rates).

The innings pitched feature was normalized to [0, 1]. It is important to take into context how much a player pitches, but before normalization IP was weighing too heavily on the overall Pitching statistic. Before inclusion of IP, the lower and higher ends of Pitching-rated players consisted overwhelmingly of those with fewer appearances. This makes sense: if a player makes only one appearances, strikes out every batter (or even the only batter he faces) and ends with a 0.0 ERA, he will of course end up as a top-rated pitcher. Thus, though many volumetric statistics may best be avoided (as longevity does not necessarily correlate completely with talent, but rather just says that a player is at least serviceable), the inclusion of one helps to normalize ratings across the entire dataset. It was decided by the author that innings pitched is the most informative volumetric statistic available: game appearances (G) doesn't say how long they were in the game; games started (GS) gives no indication of performance in that game; wins and losses (W/L) are more team-focused than pitching statistics. Innings pitched gives a more microscopic look at exactly how much a pitcher plays, and implicit in that information is some indication of how well he performs.

6.5 Games

Game logs were pulled from Retrosheet [33] for the years 1919 to 2019. Where games featured a player that did not qualify to be counted (*i.e.* a pitcher with only one or two total batters faced), he was treated as a zero-level player. Unneeded columns were stripped away as, for now, the model is only focusing on the people playing.

Each game log lists the nine position players and starting pitcher for each team. The primary preprocessing step in these tables was to replace those player IDs with their preprocessed batting stats. This resulted in each game represented as a table with eighteen rows (players) and thirty columns (player statistics). These tables were flattened to store as CSV files, with an additional column added to depict whether or not the home team won. For the CNN, these tables were reconstructed after being read in; for the ANN, they were processed as-is.

Given more time, the model would have been built to figure in other factors such as park conditions and dimensions, manager, era and more. Unfortunately these goals were not realized within the given timeframe.

Chapter 7: Core Contribution

The foundation of this project is the player model, and each player can either be "position player/-batter" or a "pitcher". In the National League, a pitcher takes his turn to hit in the lineup (typically in the ninth position), while in the American League he does not hit at all. For the first version of this project, a pitcher's hitting statistics are ignored for the purposes of his rating; this output is typically negligible and has no real effect on the measure of a pitcher's impact on his team's performance. However, this offensive output is tracked and used for the predictive game model.

7.1 General Setup

After each model run, a record was made of: minimum, average and final loss for both the training set and the test set; loss function and optimization method; early stop patience and monitored value; total number of epochs run (since early stop was used); and the TensorFlow-created model summary. The loss and validation loss were graphed over the course of the run and recorded in the documentation. For all ANNs in the project, inner layers used ReLU activation and the output layers used sigmoid. Other activation functions were attempted for inner layers, chiefly tanh and sigmoid, but performance significantly decreased in those runs and they were not deemed worthy to record.

Dropout was used across the board, primarily 50% but occasionally 20% in cases where that seemed to perform better (this was rare). Since regression models were used for player ratings, mean squared error was used as a loss metric. Others were attempted but performance was significantly worse.

7.2 Position Players

The position player model was developed as a typical artificial neural network. Early versions of the model were quite deep, with the initial attempt consisting of nine layers. It was quickly discovered that less layers were better, and best results were found in the later versions of the model with only three layers.

This project experimented with different batch sizes early on. With a relatively small dataset, it may be that batches are largely unnecessary; however, it was found that small alterations to this hyperparameter led to demonstrable results in the model accuracy. In early model versions, with an early stopping patience around six, there was less leeway for smaller batches. A size of 128 performed best overall in these cases. Later, when the patience was raised to twenty and the model improved overall, smaller batch sizes were revisited and the best results were found with batches of 64 examples. Looking at the graphs, it becomes apparent that the smaller batch sizes lead to less batch-by-batch consistency in loss; however, the data in 'batting_results.csv' makes it clear that the model performs better overall. A note for 'batting_results.csv': the first twelve rows record the tweaking of the model, while the rest are test runs with the twelfth model's setup

and hyperparameters.

Each layer of the model uses ridge regression, built into TensorFlow as an L2 regularizer. This regularizer adds the "squared magnitude of the coefficient as the penalty term to the loss function." [39] This regularization technique helps control overfitting in the model; this is an imperative concern, as the limited number of examples means the model must be wary of drawing too much from the training data.

The loss function used in the position player model is mean squared error. This model approximates a player "rating" value, meaning that it is a regression model rather than classification. The model's determined value for each example is measured against a combination of some of the predominant advanced statistics (WAR, wRC+ and wOBA) to find loss. These statistics are currently seen as the cutting edge of player impact evaluation in baseball. The optimization algorithm used is Adam: a variation of stochastic gradient descent which combines momentum and RMSProp. Adam should allow the model to find minima more efficiently while hopefully avoiding overshooting.

With the final player model version, which early-stopped at 200 epochs, a mean loss on the test set was reached at 0.00124253. The minimum test loss was 0.000846773. Compared to the mean and minimum training losses of 0.0018555 and 0.00136249, these values were deemed acceptable for the final player model.

7.3 Pitchers

Pitchers were handled separately in this project: batting stats for pitchers are less important when discussing their overall impact on a game (as previously stated), so the model was set up to instead focus on pure pitching statistics. Much of the setup work and model design was the same as with position players: the general model was an ANN with L2 regularization; the output was regression as a 'Pitching' rating was derived; loss was measured with mean squared error; and optimization of this loss was accomplished using the Adam algorithm.

The twelve tested model versions were recorded in the 'Pitching Models' document, and the results of those models were recorded in 'pitching_results.csv'. The final version of the model achieved a mean train loss of 0.0.000448769 and a mean test loss of 0.0.000225915. Several more runs were performed with the same configuration and the observed results were similar.

7.4 Games

7.4.1 General Setup

In addition to different model architectures, there were multiple runs used to test different batch sizes, number of epochs, early stopping patience, evaluation methods and optimization algorithms.

The evaluation measure used was binary cross-entropy - this makes sense, as we are merely classifying each example into one of two classes and binary cross-entropy tends to be the most effective all-around method to evaluate this process.

Different optimization algorithms were attempted: Adagrad and Adam experienced similar success,

with Adam obviously reaching the model's minimum much more quickly. No other optimization algorithms or evaluation methods experienced nearly the same level of success.

7.4.2 CNN Implementation

For each game, all players were looked up in the batters.csv table and their field in the game log table (retroID) was replaced with their statline up to that point in their career; each game then consisted of an X (the list of all players' statistics) of 54 features (18 players with 30 features each) and a y (the outcome: did the home team win?).

The games were saved in this manner, as rows of 541 features. When brought into memory for the predictive model, these rows were processed to form 18x30 game tables paired with a numeric result indicator. This is where the CNN design comes in: these 18x30 tables were treated as images to be classified.

It quickly became clear that the CNN benefited from larger batch sizes and slightly more layers. Early models overfit the training data quite a bit, as evidenced by a drop in loss paired with an increase in validation loss. Best results were achieved with a four-layer architecture (each layer being a pair of convolution and max pooling layers) with the first layer having a 1x1 kernel for both.

The hope in using a CNN was that through multiple processing steps built upon a smaller amount of data, a more robust predictive model would be built than what is attainable with a regular ANN.

7.4.3 ANN Implementation

The ANN simply took all games as rows of 540 features (along with the result feature (y) and processed them as rows. Different layer amounts and setups were used, and it quickly became apparent that an architecture with more layers tended to be slightly more gainful. The most successful versions had between three and four layers. The models benefited from L2 regularization, which ended up being utilized in all recorded architectures.

The model benefited from larger batch sizes, with 2048 seeming to be the most successful.

Chapter 8: Evaluation and Conclusions

8.0.1 Player Models

Both player models (position players and pitchers) at first glance performed much better than the predictive game model. Losses were extremely low ($< 0.1\%$) for both and the test loss seemed to closely mirror the training loss. Minimal layers were needed to achieve these levels and small batches could be used as the overall complexity of the data was not that high.

Of course performance of this level immediately raises red flags, as it would seem to have almost perfect performance. Of course there are several missteps where evaluation may have failed.

First, it is of course possible that the metric compared against is insufficient to rate a player and/or too easy to copy. As stated earlier, that Batting/Pitching statistics created in this project are merely aggregations of advanced statistics used in Sabermetrics. It could be that combining those statistics in the way in which they were combined provides an inferior method of evaluating a player. However, these metrics were compared to WAR (the overall industry standard in player evaluation) and found to be similar enough to pass muster.

It is also possible that the model is overfitting the training data in a way that the validation steps against test data were unable to pinpoint. If so, that must merely be accepted as a shortcoming of the model that could not be resolved in the scope of this project: especially if the issue was with improper data preprocessing, as the majority of the work was in that area.

Of course, it may be that the absurdly high level of success shows *some* amount of general success, and perhaps the player and pitcher models *did* in fact show some level of adroitness in handling player statistics. In this case, it may simply be taken as a small victory in the attempt to automate ratings and rankings of baseball players.

8.0.2 Predictive Game Models

The predictive game models that were built experienced significantly less success than the player models: this may have been a result of inferior preprocessing, or more nuance in how the data was handled and the model constructed. Results from the CNN model were compared to a standard ANN model. Unfortunately, as shown in the model runs in the supplemental materials, neither was particularly successful. Both achieved around 69% loss and an accuracy in the mid-50s. The CNN had more trouble actually learning, as we saw several flat-lines where both loss and validation loss failed to shrink appreciably. The CNN was also more likely to overfit the data. Steps were taken to avoid this overfitting (changed makeup of convolutional layers and different pooling types), but the runs shown were the most successful.

CNN

The CNN models were unable to perform better than around 68% loss on the test set. Early models suffered from overfitting, likely as a result of batches that were too small. It also seems that they had too many layers - only three or four, but certainly enough to fit the training data too closely. Once the number of layers was lessened and the batch size increased, the test loss

stopped climbing and levelled off, keeping pace with the training loss.

Unfortunately, these both flatlined at, at best, 68% loss with accuracy around 56%. It was shown that smaller batch sizes could work with appropriate tweaking of the model architecture. In short, the most beneficial setup was either less layers (two) or more layers with staggered dropout and the first layers consisting of 1x1 kernels for both the convolution and max pooling layers.

While there was small success in minimizing overfitting and stopping the exponential climb of test loss, that level of 68% loss could not be overcome in this project.

ANN

Once that 68% plateau was hit with the CNN, it was decided that this level of success should be tested against a variety of typical artificial neural network models. These models immediately showed the expected curve of exponential distribution for regular ANNs on loss, and test loss closely follows training loss in each attempted architecture. This is a positive sign for the fight against overfitting, performance was improved by using less layers (four or less) and keeping dropout around 50% (20% was attempted but led to significantly less stability in the model). Use of more layers (more than four) resulted in a model that showed close correlation between training and testing, but these setups failed to beat 69

In the end, the best model used three layers of halving neuron counts and 50% dropout on each layer, with a batch size of 2048. However, this model showed only a 5% improvement in test loss over the 89 epochs that it ran; obviously, this is not a very good performance.

Game Predictive Model: Conclusions

Neither model was able to perform better than 68% loss, and in fact both peaked at right around that amount. It would seem that the data could have been preprocessed better, and doing so *may* have led to better overall models. Of course preprocessing it the most important part of any deep learning project and, while it was previously stated that baseball data was chosen because it should be relatively clean compared to most other data sources, it is possible that this project did not go far enough to clean and normalize the game logs.

Within the context of the project, the original question was: can a model be built that is robust enough to handle the ultimate lack of expressiveness in baseball data? Based on the results and with the (broad) assumption that the dataset was sufficiently preprocessed, the answer would seem to be no. Of course it has been said before from various sources that it would be hard to beat the performance of simple but very finely-tuned normalized linear weights models such as those used by the likes of PECOTA [22] and FiveThirtyEight [27] [29] with any sort of a deep model. This project would appear to prove that supposition correct. This is not, however, the end of the journey to find a suitable deep model to predict baseball game outcomes: it merely provides groundwork and ideas to mark the beginning of such a pursuit.

8.0.3 Overall Conclusions

While significant work went into properly assembling and preprocessing baseball data, researching past methods and attempting to learn from them, and tweaking models to perform as well as possible given the datasets, the findings of this report show a shortcoming in the attempt to predict game outcomes and, possible, rate players. Given more time, steps would have been taken to re-process the data and begin to built separate models from the ground up to address these

shortcomings. This will constitute a future pursuit in *Machine Learning and Baseball*, but falls outside of the scope and timeframe of this version of the project.

There were some successes, though, resulting from the work done here. I learned significantly more about gathering and processing data using Python and Pandas; data visualization using Seaborn and MatPlotLib; building both artificial neural networks and convolutional neural networks using TensorFlow 2 and Keras; and overall I improved my skills in Python and general programming for data analytics. While the results of the predictive model were not what I had hoped, and were undoubtedly negatively affected by the necessity to return to my home country amid a pandemic in the middle of the semester, I am still happy with what I learned and the amount of research I did for this project. I see that aspect as an unbridled success.

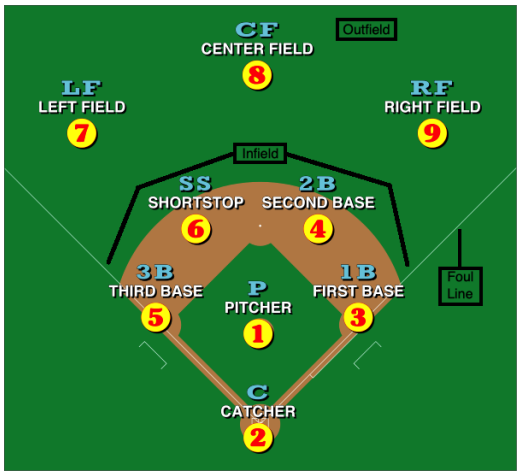
Appendix A: **Understanding Baseball**

Baseball is played over nine "innings". An inning consists of first the away team batting until they get three outs (the "top" of the inning), then the home team batting until they get three outs (the "bottom"). If the home team is ahead after the top of the ninth inning, the bottom of the ninth is not played.

There are four bases, and the hitter hits from the fourth - home plate. The goal for a hitter is to get to each base and then back to home - this scores a run. Each base has a defensive player at it - a first, second or third baseman. Beyond the bases is the outfield, and there are three defensive outfielders who "field" the ball if it gets to them. There is also a "shortstop" between second and third base - these positions, plus the pitcher and catcher, comprise the nine players on the field playing defense. These nine players come up in a defined order to hit when that team is on offense.

The pitcher throws the baseball to the hitter (the catcher crouches behind the hitter and catches the ball, also providing guidance to the pitcher on what to throw). If the hitter swings and fails to make contact, or does not swing when the ball is considered "in the hitting zone", this is a strike. If a hitter gets three strikes, he is out on a "strikeout". If the hitter hits the ball in front of him, he attempts to run to the first base. If a defensive player on the pitcher's team catches the ball before it hits the ground, the hitter is out. If a defensive player throws the ball to any baseman before the hitter reaches that base, the hitter is out. The hitter must get to a base, then can attempt to advance to the next base (or multiple bases) if the next hitter gets a hit. If a player gets a hit, he is awarded a single, double, triple or home run depending on how many bases he advances. A home run is awarded when the ball goes past the outfield and over the wall, inside the range of the foul line. If a ball is hit outside the range of the foul line, it is a foul ball and is considered a strike. However, a player can never get a third strike on a foul - he gets a chance to hit again.

There are no penalties or fouls, and no ties: if the game is tied after the ninth inning, play continues until one team wins.



[40] [Altered by author]

Appendix B: **Workbooks and Models**

Following is a series of PDFs showing the steps for compiling and preprocessing data, building tables and tensors, and constructing ANN and CNN models for player rating and game predictions.

Data Aggregation and

Preprocessing and Tensor Setup

Data Preprocessing

Data cleaning, normalization and processing and addition of advanced statistics pulled from external sources.

batting_pre

March 9, 2020

[120]:
```
import math
import numpy as np
import pandas as pd

# We're going to be reassigning some columns, so we'll turn off this warning -␣
 ↪we know what we're doing!
pd.options.mode.chained_assignment = None  # default='warn'
```

[121]:
```
# This will be exported to a separate module
ids = pd.read_csv('../data/lahman/mlb_data/People.csv')
ids = ids[['playerID', 'retroID']]
id_dict = ids.set_index('playerID').to_dict()['retroID']

def get_retroid(id):
    return id_dict[id] if id_dict is not None else id
```

[122]:
```
df = pd.read_csv('../data/lahman/mlb_data/Batting.csv').sort_values('playerID')
```

[123]:
```
df['playerID'] = df['playerID'].apply(get_retroid)
```

[124]:
```
df.rename(columns={'playerID': 'retroID'}, inplace=True)
```

[125]:
```
df[df['retroID'] == None]
```

[125]:
```
Empty DataFrame
Columns: [retroID, yearID, stint, teamID, lgID, G, AB, R, H, 2B, 3B, HR, RBI,
SB, CS, BB, SO, IBB, HBP, SH, SF, GIDP]
Index: []

[0 rows x 22 columns]
```

Cleaning the Data - Missing Values

Print percentages of missing data in each column of the batting table

[126]:
```
100 * df.isnull().sum() / len(df)
```

[126]:
```
retroID     0.000000
yearID      0.000000
stint       0.000000
teamID      0.000000
lgID        0.000000
G           0.000000
AB          0.000000
R           0.000000
H           0.000000
2B          0.000000
3B          0.000000
HR          0.000000
RBI         0.000000
SB          0.000000
CS          8.221708
BB          0.000000
SO          0.000000
IBB        21.711883
HBP         0.000000
SH          0.000000
SF         21.090864
GIDP        9.839985
dtype: float64
```

Since this data is by season, it's likely that we have entries for a player for one season with no data in these fields but there is data for other seasons. Since we're taking aggregate sums for each player, we have two options: set these null values to zero so they don't add to the sum, or set them to the average for that player. We'll have to test the theory to see which is more viable.

We're going to start with IBB rather than CS, since it's a more significant chunk of the dataset.

Handling missing IBB data

[127]:
```
df[(df['IBB'].isnull())]
```

[127]:

	retroID	yearID	stint	teamID	lgID	G	AB	R	H	2B	...	RBI	\
19269	aaroh101	1954	1	ML1	NL	122	468	58	131	27	...	69	
18684	abera101	1953	2	DET	AL	17	23	2	3	0	...	2	
16858	abera101	1950	1	CLE	AL	1	2	0	0	0	...	0	
19270	abera101	1954	1	DET	AL	32	39	3	5	0	...	3	
18683	abera101	1953	1	CLE	AL	6	0	0	0	0	...	0	
...	...	...	...	...	...	...	...	..	...	..	...	...	
15128	zubeb101	1946	1	NYA	AL	3	2	0	0	0	...	0	
14447	zubeb101	1945	1	NYA	AL	21	42	1	7	0	...	3	
13868	zubeb101	1944	1	NYA	AL	22	31	1	4	0	...	1	
19843	zuveg101	1954	1	CIN	NL	2	2	1	1	0	...	0	
19844	zuveg101	1954	2	DET	AL	35	64	1	8	1	...	3	

```
       SB   CS  BB  SO  IBB  HBP  SH   SF  GIDP
19269   2  2.0  28  39  NaN    3   6  4.0  13.0
18684   0  0.0   1   6  NaN    0   1  NaN   0.0
16858   0  0.0   1   1  NaN    0   0  NaN   0.0
19270   0  0.0   2  17  NaN    0   3  1.0   1.0
18683   0  0.0   2   0  NaN    0   0  NaN   0.0
...    ..  ...  ..  ..  ...  ...  ..  ...   ...
15128   0  0.0   0   1  NaN    0   0  NaN   0.0
14447   0  0.0   1  13  NaN    0   2  NaN   1.0
13868   0  0.0   0  10  NaN    0   4  NaN   1.0
19843   0  0.0   0   1  NaN    0   0  0.0   0.0
19844   0  1.0   1  14  NaN    0   9  0.0   2.0

[19159 rows x 22 columns]
```

[128]:
```
df[(df['retroID'] == 'abera101')]
```

[128]:
```
        retroID  yearID  stint teamID lgID   G  AB  R  H  2B  ...  RBI  SB  \
19846  abera101    1955      1    DET   AL  39  17  0  1   0  ...    0   0
18684  abera101    1953      2    DET   AL  17  23  2  3   0  ...    2   0
16858  abera101    1950      1    CLE   AL   1   2  0  0   0  ...    0   0
19270  abera101    1954      1    DET   AL  32  39  3  5   0  ...    3   0
18683  abera101    1953      1    CLE   AL   6   0  0  0   0  ...    0   0
21123  abera101    1957      2    KC1   AL   3   1  0  1   0  ...    0   0
20501  abera101    1956      1    DET   AL  42  10  0  3   0  ...    0   0
21122  abera101    1957      1    DET   AL  28   8  0  1   0  ...    1   0

        CS  BB  SO  IBB  HBP  SH   SF  GIDP
19846  0.0   0   9  0.0    0   2  0.0   1.0
18684  0.0   1   6  NaN    0   1  NaN   0.0
16858  0.0   1   1  NaN    0   0  NaN   0.0
19270  0.0   2  17  NaN    0   3  1.0   1.0
18683  0.0   2   0  NaN    0   0  NaN   0.0
21123  0.0   0   0  0.0    0   0  0.0   0.0
20501  0.0   1   4  0.0    0   2  0.0   0.0
21122  0.0   1   4  0.0    0   0  0.0   0.0

[8 rows x 22 columns]
```

[129]:
```
df[(df['retroID'] == 'zubeb101')]
```

[129]:
```
        retroID  yearID  stint teamID lgID   G  AB  R  H  2B  ...  RBI  SB  \
9445   zubeb101    1936      1    CLE   AL   2   5  1  1   0  ...    0   0
10501  zubeb101    1938      1    CLE   AL  15   7  0  0   0  ...    0   0
11080  zubeb101    1939      1    CLE   AL  16   5  0  1   0  ...    0   0
11621  zubeb101    1940      1    CLE   AL  17   3  0  1   0  ...    0   0
12203  zubeb101    1941      1    WS1   AL  36  26  0  0   0  ...    0   0
```

```
12742  zubeb101    1942      1    WS1   AL  37  39  5  6   3  ...    3   0
13299  zubeb101    1943      1    NYA   AL  20  38  1  7   1  ...    2   0
15711  zubeb101    1947      1    BOS   AL  20  13  0  2   0  ...    0   0
15129  zubeb101    1946      2    BOS   AL  15  18  1  2   0  ...    2   0
15128  zubeb101    1946      1    NYA   AL   3   2  0  0   0  ...    0   0
14447  zubeb101    1945      1    NYA   AL  21  42  1  7   0  ...    3   0
13868  zubeb101    1944      1    NYA   AL  22  31  1  4   0  ...    1   0

        CS  BB  SO  IBB  HBP  SH  SF  GIDP
9445   0.0   0   1  NaN    0   0 NaN   NaN
10501  0.0   0   1  NaN    0   1 NaN   NaN
11080  0.0   0   2  NaN    0   0 NaN   0.0
11621  0.0   0   0  NaN    0   0 NaN   0.0
12203  0.0   1   8  NaN    0   2 NaN   0.0
12742  0.0   1   7  NaN    0   3 NaN   2.0
13299  0.0   4  14  NaN    0   5 NaN   2.0
15711  0.0   2   3  NaN    0   2 NaN   2.0
15129  0.0   1   6  NaN    0   1 NaN   0.0
15128  0.0   0   1  NaN    0   0 NaN   0.0
14447  0.0   1  13  NaN    0   2 NaN   1.0
13868  0.0   0  10  NaN    0   4 NaN   1.0

[12 rows x 22 columns]
```

First let's look at IBB, intentional bases on balls. It seems like most of the missing data is from early in the dataset - it could be that IBB was not recorded then, and/or not considered a trackable play?

[130]:
```
df[(df['IBB'].isnull())]['yearID'].max()
```

[130]:
```
1954
```

[131]:
```
df[(df['IBB'].isnull())]['yearID'].min()
```

[131]:
```
1919
```

[132]:
```
df[(df['IBB'].isnull())]
```

[132]:
```
        retroID  yearID  stint teamID lgID    G   AB   R    H  2B  ...  RBI  \
19269  aaroh101    1954      1    ML1   NL  122  468  58  131  27  ...   69
18684  abera101    1953      2    DET   AL   17   23   2    3   0  ...    2
16858  abera101    1950      1    CLE   AL    1    2   0    0   0  ...    0
19270  abera101    1954      1    DET   AL   32   39   3    5   0  ...    3
18683  abera101    1953      1    CLE   AL    6    0   0    0   0  ...    0
...         ...     ...    ...    ...  ...  ...  ...  ..  ...  ..  ...  ...
15128  zubeb101    1946      1    NYA   AL    3    2   0    0   0  ...    0
14447  zubeb101    1945      1    NYA   AL   21   42   1    7   0  ...    3
```

```
13868  zubeb101    1944      1    NYA   AL   22   31   1    4   0  ...    1
19843  zuveg101    1954      1    CIN   NL    2    2   1    1   0  ...    0
19844  zuveg101    1954      2    DET   AL   35   64   1    8   1  ...    3

       SB   CS  BB  SO  IBB  HBP  SH   SF  GIDP
19269   2  2.0  28  39  NaN    3   6  4.0  13.0
18684   0  0.0   1   6  NaN    0   1  NaN   0.0
16858   0  0.0   1   1  NaN    0   0  NaN   0.0
19270   0  0.0   2  17  NaN    0   3  1.0   1.0
18683   0  0.0   2   0  NaN    0   0  NaN   0.0
...    ..  ...  ..  ..  ...  ...  ..  ...   ...
15128   0  0.0   0   1  NaN    0   0  NaN   0.0
14447   0  0.0   1  13  NaN    0   2  NaN   1.0
13868   0  0.0   0  10  NaN    0   4  NaN   1.0
19843   0  0.0   0   1  NaN    0   0  0.0   0.0
19844   0  1.0   1  14  NaN    0   9  0.0   2.0

[19159 rows x 22 columns]
```

We have 19159 total rows where there is no data for IBB, and we know none of those rows goes past the year 1954…

[133]:
```
df[(df['yearID'] < 1955)]
```

[133]:
```
        retroID  yearID  stint teamID lgID    G   AB   R    H  2B  ...  RBI  \
19269  aaroh101    1954      1    ML1   NL  122  468  58  131  27  ...   69
18684  abera101    1953      2    DET   AL   17   23   2    3   0  ...    2
16858  abera101    1950      1    CLE   AL    1    2   0    0   0  ...    0
19270  abera101    1954      1    DET   AL   32   39   3    5   0  ...    3
18683  abera101    1953      1    CLE   AL    6    0   0    0   0  ...    0
...         ...     ...    ...    ...  ...  ...  ...  ..  ...  ..  ...  ...
13868  zubeb101    1944      1    NYA   AL   22   31   1    4   0  ...    1
18682  zuveg101    1952      1    CLE   AL    2    0   1    0   0  ...    0
19843  zuveg101    1954      1    CIN   NL    2    2   1    1   0  ...    0
19844  zuveg101    1954      2    DET   AL   35   64   1    8   1  ...    3
18050  zuveg101    1951      1    CLE   AL   16    0   0    0   0  ...    0

       SB   CS  BB  SO  IBB  HBP  SH   SF  GIDP
19269   2  2.0  28  39  NaN    3   6  4.0  13.0
18684   0  0.0   1   6  NaN    0   1  NaN   0.0
16858   0  0.0   1   1  NaN    0   0  NaN   0.0
19270   0  0.0   2  17  NaN    0   3  1.0   1.0
18683   0  0.0   2   0  NaN    0   0  NaN   0.0
...    ..  ...  ..  ..  ...  ...  ..  ...   ...
13868   0  0.0   0  10  NaN    0   4  NaN   1.0
18682   0  0.0   0   0  0.0    0   0  0.0   0.0
19843   0  0.0   0   1  NaN    0   0  0.0   0.0
```

```
19844   0  1.0   1  14  NaN    0   9  0.0   2.0
18050   0  0.0   0   0  0.0    0   0  0.0   0.0

[19845 rows x 22 columns]
```

And we have 19845 total rows up to the year 1954. That means...

[134]:
```
19159 / 19845
```

[134]:
```
0.9654320987654321
```

Over 96% of the data before 1955 is missing IBB. I think this gives justification to just setting all of those NaNs to 0.

[135]:
```
df['IBB'].fillna(value=0, inplace=True)
```

[136]:
```
100 * df.isnull().sum() / len(df)
```

[136]:
```
retroID     0.000000
yearID      0.000000
stint       0.000000
teamID      0.000000
lgID        0.000000
G           0.000000
AB          0.000000
R           0.000000
H           0.000000
2B          0.000000
3B          0.000000
HR          0.000000
RBI         0.000000
SB          0.000000
CS          8.221708
BB          0.000000
SO          0.000000
IBB         0.000000
HBP         0.000000
SH          0.000000
SF         21.090864
GIDP        9.839985
dtype: float64
```

Our IBB issue is solved. Let's move on to SF (sacrifice flies). We'll check the years and rows again to see if we're justified in using the same method to eliminate nulls.

Handling missing SF data

[137]:
```
df[(df['SF'].isnull())]['yearID'].max()
```

[137]: 1953

[138]:
```
df[(df['SF'].isnull())]['yearID'].min()
```

[138]: 1919

[139]:
```
df[(df['SF'].isnull())].shape[0]
```

[139]: 18611

[140]:
```
df[(df['yearID'] < 1954)].shape[0]
```

[140]: 19269

[141]:
```
18611/19269
```

[141]: 0.965851886449738

Almost the same percentage, and one less year covered. I think we can fill those missing values with 0.

[142]:
```
df['SF'].fillna(value=0, inplace=True)
```

[143]:
```
100 * df.isnull().sum() / len(df)
```

[143]:
```
retroID    0.000000
yearID     0.000000
stint      0.000000
teamID     0.000000
lgID       0.000000
G          0.000000
AB         0.000000
R          0.000000
H          0.000000
2B         0.000000
3B         0.000000
HR         0.000000
RBI        0.000000
SB         0.000000
CS         8.221708
BB         0.000000
SO         0.000000
IBB        0.000000
HBP        0.000000
SH         0.000000
SF         0.000000
GIDP       9.839985
dtype: float64
```

Two more to go, let's move on to CS (caught stealing)

Handling missing CS data

[144]:
```
df[(df['CS'].isnull())]['yearID'].max()
```

[144]:
```
1950
```

[145]:
```
df[(df['CS'].isnull())]['yearID'].min()
```

[145]:
```
1919
```

[146]:
```
df[(df['CS'].isnull())]
```

[146]:

	retroID	yearID	stint	teamID	lgID	G	AB	R	H	2B	...	RBI	\
14448	aberw101	1946	1	NY1	NL	15	8	0	0	0	...	0	
16285	aberc101	1949	1	CHN	NL	4	7	0	0	0	...	0	
15712	aberc101	1948	1	CHN	NL	12	32	1	6	1	...	6	
15131	aberc101	1947	1	CHN	NL	47	140	24	39	6	...	20	
16286	abrac101	1949	1	BRO	NL	8	24	6	2	1	...	0	
...	...	...	...	...	...	...	...	..	..	..	...	...	
532	zitzb101	1919	1	PIT	NL	11	26	5	5	1	...	2	
4782	zitzb101	1927	1	CIN	NL	88	232	47	66	10	...	24	
5312	zitzb101	1928	1	CIN	NL	101	266	53	79	9	...	33	
533	zitzb101	1919	2	CIN	NL	2	1	0	0	0	...	0	
5842	zitzb101	1929	1	CIN	NL	47	84	18	19	3	...	6	

	SB	CS	BB	SO	IBB	HBP	SH	SF	GIDP
14448	0	NaN	0	4	0.0	0	0	0.0	0.0
16285	0	NaN	0	2	0.0	0	0	0.0	1.0
15712	0	NaN	5	10	0.0	0	0	0.0	0.0
15131	0	NaN	20	32	0.0	0	0	0.0	5.0
16286	1	NaN	7	6	0.0	0	0	0.0	1.0
...	..	..	..	..	...	...	..	...	...
532	2	NaN	0	6	0.0	0	1	0.0	NaN
4782	9	NaN	20	18	0.0	4	17	0.0	NaN
5312	13	NaN	13	22	0.0	3	14	0.0	NaN
533	0	NaN	0	0	0.0	0	0	0.0	NaN
5842	4	NaN	9	10	0.0	1	2	0.0	NaN

[7255 rows x 22 columns]

[147]:
```
df[(df['retroID'] == 'zitzb101')]
```

[147]:

	retroID	yearID	stint	teamID	lgID	G	AB	R	H	2B	...	RBI	SB	\
4241	zitzb101	1926	1	CIN	NL	53	94	21	23	2	...	3	3	
532	zitzb101	1919	1	PIT	NL	11	26	5	5	1	...	2	2	
3715	zitzb101	1925	1	CIN	NL	104	301	53	76	13	...	21	11	

```
4782  zitzb101    1927      1    CIN   NL   88  232  47  66  10  ...   24   9
5312  zitzb101    1928      1    CIN   NL  101  266  53  79   9  ...   33  13
533   zitzb101    1919      2    CIN   NL    2    1   0   0   0  ...    0   0
5842  zitzb101    1929      1    CIN   NL   47   84  18  19   3  ...    6   4

        CS  BB  SO  IBB  HBP  SH   SF  GIDP
4241   NaN   6   7  0.0    2   3  0.0   NaN
532    NaN   0   6  0.0    0   1  0.0   NaN
3715  11.0  35  22  0.0    6   2  0.0   NaN
4782   NaN  20  18  0.0    4  17  0.0   NaN
5312   NaN  13  22  0.0    3  14  0.0   NaN
533    NaN   0   0  0.0    0   0  0.0   NaN
5842   NaN   9  10  0.0    1   2  0.0   NaN

[7 rows x 22 columns]
```

[148]:
```
df[(df['CS'].isnull())].shape[0]
```

[148]: 7255

[149]:
```
df[(df['yearID'] < 1951)].shape[0]
```

[149]: 17435

[150]:
```
7255/17435
```

[150]: 0.4161170060223688

There isn't a great solution for this. If we drop all missing rows with NaN for CS, we're going to lose over 41% of the data prior to 1951. It doesn't encompass enough of the data to just fill in values like we did before, we can't drop rows, and we don't want to drop the column since it isn't missing any data after 1950. One idea, and this may be controversial, is to find the average ratio between SB (stolen bases) and CS and fill in with values based on that ratio.

First, we'll get all data without missing CS values

[151]:
```
df_temp = df[(df['CS'].notnull())]
# df_temp
```

[152]:
```
total_sb = df_temp['SB'].sum()
total_sb
```

[152]: 182622

[153]:
```
total_cs = df_temp['CS'].sum()
total_cs
```

[153]: 94186.0

[154]:
```
total_sb/total_cs
```

[154]:
```
1.9389505871360924
```

So on average, players are almost twice as likely to steal a base as they are to get caught. This is easy math that we're going to round to make it even easier. It's probably not the best method of solving this issue but at least we still have over 60 years of clean data!

[155]:
```
df[(df['CS'].isnull())]
```

[155]:
```
        retroID  yearID  stint teamID lgID    G   AB   R   H  2B  ...  RBI  \
14448  aberw101    1946      1    NY1   NL   15    8   0   0   0  ...    0
16285  aberc101    1949      1    CHN   NL    4    7   0   0   0  ...    0
15712  aberc101    1948      1    CHN   NL   12   32   1   6   1  ...    6
15131  aberc101    1947      1    CHN   NL   47  140  24  39   6  ...   20
16286  abrac101    1949      1    BRO   NL    8   24   6   2   1  ...    0
...         ...     ...    ...    ...  ...  ...  ...  ..  ..  ..  ...  ...
532    zitzb101    1919      1    PIT   NL   11   26   5   5   1  ...    2
4782   zitzb101    1927      1    CIN   NL   88  232  47  66  10  ...   24
5312   zitzb101    1928      1    CIN   NL  101  266  53  79   9  ...   33
533    zitzb101    1919      2    CIN   NL    2    1   0   0   0  ...    0
5842   zitzb101    1929      1    CIN   NL   47   84  18  19   3  ...    6

       SB  CS  BB  SO  IBB  HBP  SH   SF  GIDP
14448   0 NaN   0   4  0.0    0   0  0.0   0.0
16285   0 NaN   0   2  0.0    0   0  0.0   1.0
15712   0 NaN   5  10  0.0    0   0  0.0   0.0
15131   0 NaN  20  32  0.0    0   0  0.0   5.0
16286   1 NaN   7   6  0.0    0   0  0.0   1.0
...    ..  ..  ..  ..  ...  ...  ..  ...   ...
532     2 NaN   0   6  0.0    0   1  0.0   NaN
4782    9 NaN  20  18  0.0    4  17  0.0   NaN
5312   13 NaN  13  22  0.0    3  14  0.0   NaN
533     0 NaN   0   0  0.0    0   0  0.0   NaN
5842    4 NaN   9  10  0.0    1   2  0.0   NaN

[7255 rows x 22 columns]
```

[156]:
```
df[(df['CS']).isnull()].apply(lambda x: x['SB'] / 2, axis=1)
```

[156]:
```
14448    0.0
16285    0.0
15712    0.0
15131    0.0
16286    0.5
        ...
532      1.0
```

```
4782     4.5
5312     6.5
533      0.0
5842     2.0
Length: 7255, dtype: float64
```

[157]:
```
df[(df['CS']).isnull()].apply(lambda x: x['SB'] / 2, axis=1).value_counts()
```

[157]:
```
0.0     4494
0.5      794
1.0      438
1.5      303
2.0      257
2.5      165
3.0      139
3.5      131
4.0       91
4.5       81
5.0       51
5.5       50
6.5       38
6.0       36
7.5       28
7.0       22
8.0       21
9.0       19
8.5       16
9.5       11
11.5       8
10.0       8
10.5       8
11.0       7
13.0       6
14.0       5
12.0       5
14.5       3
18.5       3
12.5       2
17.5       2
13.5       2
16.5       2
16.0       2
20.0       1
15.0       1
15.5       1
24.0       1
18.0       1
```

```
17.0       1
21.5       1
dtype: int64
```

I don't love the max of 24, but overall these values look good and we definitely don't have many of the higher values. So we're going to apply this to our missing CS data

First I'm going to test it out on a copy

[158]:
```
df_temp = df[(df['CS']).isnull()]
```

[159]:
```
df_temp['CS'] = df_temp.apply(lambda x: x['SB'] / 2, axis=1)
```

[160]:
```
df_temp[(df_temp['retroID'] == 'zitzb101')]
```

[160]:

	retroID	yearID	stint	teamID	lgID	G	AB	R	H	2B	...	RBI	SB	\
4241	zitzb101	1926	1	CIN	NL	53	94	21	23	2	...	3	3	
532	zitzb101	1919	1	PIT	NL	11	26	5	5	1	...	2	2	
4782	zitzb101	1927	1	CIN	NL	88	232	47	66	10	...	24	9	
5312	zitzb101	1928	1	CIN	NL	101	266	53	79	9	...	33	13	
533	zitzb101	1919	2	CIN	NL	2	1	0	0	0	...	0	0	
5842	zitzb101	1929	1	CIN	NL	47	84	18	19	3	...	6	4	

	CS	BB	SO	IBB	HBP	SH	SF	GIDP
4241	1.5	6	7	0.0	2	3	0.0	NaN
532	1.0	0	6	0.0	0	1	0.0	NaN
4782	4.5	20	18	0.0	4	17	0.0	NaN
5312	6.5	13	22	0.0	3	14	0.0	NaN
533	0.0	0	0	0.0	0	0	0.0	NaN
5842	2.0	9	10	0.0	1	2	0.0	NaN

[6 rows x 22 columns]

We know from before that this guy had NaNs for his CS and now it's all filled in, so our plan worked. Let's do it for the actual data

I don't know how to reassign values to a subset of a DataFrame based on a predicate (or if it's possible), so we'll get a little hacky and apply a function with a conditional. Here's what I tried originally:

df[(df['CS']).isnull()]['CS'] = df.apply(lambda x: x['SB'] / 2, axis=1)

[161]:
```
def fill_cs(data):
    if math.isnan(data['CS']):
        return data['SB'] / 2
    else:
        return data['CS']
```

[162]:
```
df['CS'] = df.apply(lambda x: fill_cs(x), axis=1)
```

[163]:
```
df[(df['retroID'] == 'zitzb101')]
```

[163]:
```
       retroID  yearID  stint teamID lgID    G   AB   R   H  2B  ...  RBI  SB  \
4241  zitzb101    1926      1    CIN   NL   53   94  21  23   2  ...    3   3
532   zitzb101    1919      1    PIT   NL   11   26   5   5   1  ...    2   2
3715  zitzb101    1925      1    CIN   NL  104  301  53  76  13  ...   21  11
4782  zitzb101    1927      1    CIN   NL   88  232  47  66  10  ...   24   9
5312  zitzb101    1928      1    CIN   NL  101  266  53  79   9  ...   33  13
533   zitzb101    1919      2    CIN   NL    2    1   0   0   0  ...    0   0
5842  zitzb101    1929      1    CIN   NL   47   84  18  19   3  ...    6   4

          CS  BB  SO  IBB  HBP  SH   SF  GIDP
4241     1.5   6   7  0.0    2   3  0.0   NaN
532      1.0   0   6  0.0    0   1  0.0   NaN
3715    11.0  35  22  0.0    6   2  0.0   NaN
4782     4.5  20  18  0.0    4  17  0.0   NaN
5312     6.5  13  22  0.0    3  14  0.0   NaN
533      0.0   0   0  0.0    0   0  0.0   NaN
5842     2.0   9  10  0.0    1   2  0.0   NaN

[7 rows x 22 columns]
```

[164]:
```
100 * df.isnull().sum() / len(df)
```

[164]:
```
retroID    0.000000
yearID     0.000000
stint      0.000000
teamID     0.000000
lgID       0.000000
G          0.000000
AB         0.000000
R          0.000000
H          0.000000
2B         0.000000
3B         0.000000
HR         0.000000
RBI        0.000000
SB         0.000000
CS         0.000000
BB         0.000000
SO         0.000000
IBB        0.000000
HBP        0.000000
SH         0.000000
SF         0.000000
GIDP       9.839985
dtype: float64
```

Handling missing GIDP data

[165]:
```
df[(df['GIDP'].isnull())]
```

[165]:
```
        retroID  yearID  stint teamID lgID    G   AB   R   H  2B  ...  RBI  \
2082   abrag101    1923      1    CIN   NL    3    1   0   1   0  ...    0
534    acosj101    1920      1    WS1   AL   17   25   2   6   1  ...    1
1569   acosj101    1922      1    CHA   AL    5    5   0   1   0  ...    0
1049   acosj101    1921      1    WS1   AL   33   30   2   2   0  ...    0
6374   adaij102    1931      1    CHN   NL   18   76   9  21   3  ...    3
...         ...     ...    ...    ...  ...  ...  ...  ..  ..  ..  ...  ...
5312   zitzb101    1928      1    CIN   NL  101  266  53  79   9  ...   33
533    zitzb101    1919      2    CIN   NL    2    1   0   0   0  ...    0
5842   zitzb101    1929      1    CIN   NL   47   84  18  19   3  ...    6
9445   zubeb101    1936      1    CLE   AL    2    5   1   1   0  ...    0
10501  zubeb101    1938      1    CLE   AL   15    7   0   0   0  ...    0

       SB   CS  BB  SO  IBB  HBP  SH   SF  GIDP
2082    0  0.0   0   0  0.0    0   0  0.0   NaN
534     0  0.0   4   7  0.0    0   2  0.0   NaN
1569    0  0.0   1   1  0.0    0   0  0.0   NaN
1049    1  0.0   6  14  0.0    0   1  0.0   NaN
6374    1  0.5   1   8  0.0    0   2  0.0   NaN
...    ..  ...  ..  ..  ...  ...  ..  ...   ...
5312   13  6.5  13  22  0.0    3  14  0.0   NaN
533     0  0.0   0   0  0.0    0   0  0.0   NaN
5842    4  2.0   9  10  0.0    1   2  0.0   NaN
9445    0  0.0   0   1  0.0    0   0  0.0   NaN
10501   0  0.0   0   1  0.0    0   1  0.0   NaN

[8683 rows x 22 columns]
```

[166]:
```
df[(df['GIDP'].isnull())]['yearID'].max()
```

[166]:
```
1938
```

[167]:
```
df[(df['yearID'] < 1939)].shape[0]
```

[167]:
```
10502
```

[168]:
```
df[(df['GIDP'].isnull())].shape[0]
```

[168]:
```
8683
```

[169]:
```
8683/10502
```

[169]:
```
0.8267948962102457
```

Over 82% of records before 1939 are missing GIDP, but it doesn't extend beyond that. I think we can once again just fill the values in with 0

[170]:
```
df['GIDP'].fillna(value=0, inplace=True)
```

[171]:
```
100 * df.isnull().sum() / len(df)
```

[171]:
```
retroID    0.0
yearID     0.0
stint      0.0
teamID     0.0
lgID       0.0
G          0.0
AB         0.0
R          0.0
H          0.0
2B         0.0
3B         0.0
HR         0.0
RBI        0.0
SB         0.0
CS         0.0
BB         0.0
SO         0.0
IBB        0.0
HBP        0.0
SH         0.0
SF         0.0
GIDP       0.0
dtype: float64
```

We've handled all missing data in the batting database

Data Integration

Now we need to eliminate an columns that we don't want (if any) and convert the ones we keep to numerical values.

[172]:
```
df.head()
```

[172]:

	retroID	yearID	stint	teamID	lgID	G	AB	R	H	2B	...	RBI	SB	\
79400	aardd001	2013	1	NYN	NL	43	0	0	0	0	...	0	0	
82244	aardd001	2015	1	ATL	NL	33	1	0	0	0	...	0	0	
69712	aardd001	2006	1	CHN	NL	45	2	0	0	0	...	0	0	
73859	aardd001	2009	1	SEA	AL	73	0	0	0	0	...	0	0	
71089	aardd001	2007	1	CHA	AL	25	0	0	0	0	...	0	0	

	CS	BB	SO	IBB	HBP	SH	SF	GIDP
79400	0.0	0	0	0.0	0	0	0.0	0.0

```
82244  0.0   0   1  0.0    0   0  0.0   0.0
69712  0.0   0   0  0.0    0   1  0.0   0.0
73859  0.0   0   0  0.0    0   0  0.0   0.0
71089  0.0   0   0  0.0    0   0  0.0   0.0

[5 rows x 22 columns]
```

[173]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 88242 entries, 79400 to 86706
Data columns (total 22 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   retroID  88242 non-null  object
 1   yearID   88242 non-null  int64
 2   stint    88242 non-null  int64
 3   teamID   88242 non-null  object
 4   lgID     88242 non-null  object
 5   G        88242 non-null  int64
 6   AB       88242 non-null  int64
 7   R        88242 non-null  int64
 8   H        88242 non-null  int64
 9   2B       88242 non-null  int64
 10  3B       88242 non-null  int64
 11  HR       88242 non-null  int64
 12  RBI      88242 non-null  int64
 13  SB       88242 non-null  int64
 14  CS       88242 non-null  float64
 15  BB       88242 non-null  int64
 16  SO       88242 non-null  int64
 17  IBB      88242 non-null  float64
 18  HBP      88242 non-null  int64
 19  SH       88242 non-null  int64
 20  SF       88242 non-null  float64
 21  GIDP     88242 non-null  float64
dtypes: float64(4), int64(15), object(3)
memory usage: 15.5+ MB
```

We will handle the metadata columns later and only worry about numerical columns for now

[174]:
```
df['lgID'].value_counts()
```

[174]:
```
NL    44129
AL    44113
Name: lgID, dtype: int64
```

[175]:
```
pd.get_dummies(df['lgID'], drop_first=True)
```

[175]:
```
        NL
79400    1
82244    1
69712    1
73859    0
71089    0
...     ..
20499    0
18050    0
83729    0
85212    0
86706    0

[88242 rows x 1 columns]
```

This one will be easy - there are only two leagues in the dataset, so we can just transform that into a single boolean column. Of course that column will be NL, the superior league.

[176]:
```
df['NL'] = pd.get_dummies(df['lgID'], drop_first=True)
df.drop(columns=['lgID'], inplace=True)
```

[177]:
```
df
```

[177]:

	retroID	yearID	stint	teamID	G	AB	R	H	2B	3B	...	SB	CS	BB	\
79400	aardd001	2013	1	NYN	43	0	0	0	0	0	...	0	0.0	0	
82244	aardd001	2015	1	ATL	33	1	0	0	0	0	...	0	0.0	0	
69712	aardd001	2006	1	CHN	45	2	0	0	0	0	...	0	0.0	0	
73859	aardd001	2009	1	SEA	73	0	0	0	0	0	...	0	0.0	0	
71089	aardd001	2007	1	CHA	25	0	0	0	0	0	...	0	0.0	0	
...	...	...	...	...	..	..	..	..	..	..	...	..	...	..	
20499	zuveg101	1955	2	BAL	28	23	1	5	1	0	...	0	0.0	1	
18050	zuveg101	1951	1	CLE	16	0	0	0	0	0	...	0	0.0	0	
83729	zycht001	2015	1	SEA	13	0	0	0	0	0	...	0	0.0	0	
85212	zycht001	2016	1	SEA	12	0	0	0	0	0	...	0	0.0	0	
86706	zycht001	2017	1	SEA	45	0	0	0	0	0	...	0	0.0	0	

	SO	IBB	HBP	SH	SF	GIDP	NL
79400	0	0.0	0	0	0.0	0.0	1
82244	1	0.0	0	0	0.0	0.0	1
69712	0	0.0	0	1	0.0	0.0	1
73859	0	0.0	0	0	0.0	0.0	0
71089	0	0.0	0	0	0.0	0.0	0
...	..	...	...	..	...	...	..
20499	5	0.0	0	1	0.0	1.0	0
18050	0	0.0	0	0	0.0	0.0	0
83729	0	0.0	0	0	0.0	0.0	0
85212	0	0.0	0	0	0.0	0.0	0

```
86706   0  0.0    0   0  0.0   0.0   0

[88242 rows x 22 columns]
```

Now we need to figure out how to handle the teamID column.

[178]:
```
df['teamID'].nunique()
```

[178]:
```
45
```

Since we have more than 30 team IDs, to keep things consistent I'm just going to map them to franchise ID.

[179]:
```
# This will be exported to a separate module
teams = pd.read_csv('../data/lahman/mlb_data/Teams.csv')
teams = teams[['teamID', 'franchID']]
team_dict = teams.set_index('teamID').to_dict()['franchID']

def get_team(team):
    return team_dict[team] if id_dict is not None else team
```

[180]:
```
df['teamID'] = df['teamID'].apply(get_team)
```

[181]:
```
df['teamID'].nunique()
```

[181]:
```
30
```

We're now all set with team IDs as strings

[182]:
```
df.head()
```

[182]:
```
        retroID  yearID  stint teamID   G  AB  R  H  2B  3B  ...  SB   CS  BB  \
79400  aardd001    2013      1    NYM  43   0  0  0   0   0  ...   0  0.0   0
82244  aardd001    2015      1    ATL  33   1  0  0   0   0  ...   0  0.0   0
69712  aardd001    2006      1    CHC  45   2  0  0   0   0  ...   0  0.0   0
73859  aardd001    2009      1    SEA  73   0  0  0   0   0  ...   0  0.0   0
71089  aardd001    2007      1    CHW  25   0  0  0   0   0  ...   0  0.0   0

       SO  IBB  HBP  SH   SF  GIDP  NL
79400   0  0.0    0   0  0.0   0.0   1
82244   1  0.0    0   0  0.0   0.0   1
69712   0  0.0    0   1  0.0   0.0   1
73859   0  0.0    0   0  0.0   0.0   0
71089   0  0.0    0   0  0.0   0.0   0

[5 rows x 22 columns]
```

[183]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 88242 entries, 79400 to 86706
Data columns (total 22 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   retroID  88242 non-null  object
 1   yearID   88242 non-null  int64
 2   stint    88242 non-null  int64
 3   teamID   88242 non-null  object
 4   G        88242 non-null  int64
 5   AB       88242 non-null  int64
 6   R        88242 non-null  int64
 7   H        88242 non-null  int64
 8   2B       88242 non-null  int64
 9   3B       88242 non-null  int64
 10  HR       88242 non-null  int64
 11  RBI      88242 non-null  int64
 12  SB       88242 non-null  int64
 13  CS       88242 non-null  float64
 14  BB       88242 non-null  int64
 15  SO       88242 non-null  int64
 16  IBB      88242 non-null  float64
 17  HBP      88242 non-null  int64
 18  SH       88242 non-null  int64
 19  SF       88242 non-null  float64
 20  GIDP     88242 non-null  float64
 21  NL       88242 non-null  uint8
dtypes: float64(4), int64(15), object(2), uint8(1)
memory usage: 14.9+ MB
```

[184]:
```
df = df.sort_index()
```

[185]:
```
df.head()
```

[185]:

	retroID	yearID	stint	teamID	G	AB	R	H	2B	3B	...	SB	CS	BB	\
0	adamb104	1919	1	PIT	34	92	2	17	2	1	...	0	0.0	6	
1	adamb106	1919	1	PHI	78	232	14	54	7	2	...	4	2.0	6	
2	adamw101	1919	1	OAK	1	2	0	0	0	0	...	0	0.0	0	
3	agnes101	1919	1	MIN	42	98	6	23	7	0	...	1	0.5	10	
4	ainse101	1919	1	DET	114	364	42	99	17	12	...	9	4.5	45	

	SO	IBB	HBP	SH	SF	GIDP	NL
0	13	0.0	0	3	0.0	0.0	1
1	27	0.0	0	3	0.0	0.0	1
2	1	0.0	0	0	0.0	0.0	0
3	8	0.0	1	9	0.0	0.0	0
4	30	0.0	1	12	0.0	0.0	0

```
[5 rows x 22 columns]
```

We need some sort of dictionary to associate a player's retroID with an index. The following steps care of that. This is so we can later associate the correct retroID with our data.

[186]:
```
df.reset_index(inplace=True)
```

[187]:
```
metadata_column_labels = ['index', 'yearID', 'stint', 'teamID']
```

[188]:
```
metadata = df[metadata_column_labels].set_index(df['retroID']).reset_index()
```

[189]:
```
metadata.head()
```

[189]:

	retroID	index	yearID	stint	teamID
0	adamb104	0	1919	1	PIT
1	adamb106	1	1919	1	PHI
2	adamw101	2	1919	1	OAK
3	agnes101	3	1919	1	MIN
4	ainse101	4	1919	1	DET

The metadata table will eventually be expanded with information from Players.csv to hold all relevant player information that isn't used for the neural network.

[190]:
```
indexer = metadata.drop_duplicates('retroID').set_index('index').T.
↪to_dict('retroID')[0]
```

[191]:
```
df = df.drop(columns=metadata_column_labels)
```

[192]:
```
df.head()
```

[192]:

	retroID	G	AB	R	H	2B	3B	HR	RBI	SB	CS	BB	SO	IBB	HBP	SH	SF	GIDP	NL
0	adamb104	34	92	2	17	2	1	0	4	0	0.0	6	13	0.0	0	3	0.0	0.0	1
1	adamb106	78	232	14	54	7	2	1	17	4	2.0	6	27	0.0	0	3	0.0	0.0	1
2	adamw101	1	2	0	0	0	0	0	0	0	0.0	0	1	0.0	0	0	0.0	0.0	0
3	agnes101	42	98	6	23	7	0	0	10	1	0.5	10	8	0.0	1	9	0.0	0.0	0
4	ainse101	114	364	42	99	17	12	3	32	9	4.5	45	30	0.0	1	12	0.0	0.0	0

Now that the metadata is gone, we just have the ID and the numerical batting information. We can group by the ID and just sum every other column to get player career totals.

[193]:
```
df = df.groupby('retroID').sum().reset_index()
```

[194]:
```
df
```

[194]:

	retroID	G	AB	R	H	2B	3B	HR	RBI	SB	CS	BB	\
0	aardd001	331	4	0	0	0	0	0	0	0	0.0	0	
1	aaroh101	3298	12364	2174	3771	624	98	755	2297	240	73.0	1402	
2	aarot101	437	944	102	216	42	6	13	94	9	8.0	86	
3	aased001	448	5	0	0	0	0	0	0	0	0.0	0	
4	abada001	15	21	1	2	0	0	0	0	0	1.0	4	
...	...	...	...	...	...	...	...	..	...	...	...	...	...
15187	zupcb001	319	795	99	199	47	4	7	80	7	5.0	57	
15188	zupof101	16	18	3	3	1	0	0	0	0	0.0	2	
15189	zuveg101	266	142	5	21	2	1	0	7	0	1.0	9	
15190	zuvep001	209	491	41	109	17	2	2	20	2	0.0	34	
15191	zycht001	70	0	0	0	0	0	0	0	0	0.0	0	

	SO	IBB	HBP	SH	SF	GIDP	NL
0	2	0.0	0	1	0.0	0.0	4
1	1383	293.0	32	21	121.0	328.0	21
2	145	3.0	0	9	6.0	36.0	7
3	3	0.0	0	0	0.0	0.0	2
4	5	0.0	0	0	0.0	1.0	1
...	...	...	...	..	...	...	..
15187	137	3.0	6	20	8.0	15.0	0
15188	6	0.0	0	0	0.0	0.0	0
15189	39	0.0	0	16	0.0	3.0	1
15190	50	1.0	2	18	0.0	8.0	4
15191	0	0.0	0	0	0.0	0.0	0

[15192 rows x 19 columns]

Since we summed everything, we just need to change the NL column back. We can divide each value by itself to get either 1 or 0 like we had before.

[195]:
```
df['NL'] = np.where(df['NL'] > 0, 1, 0)
```

[196]:
```
tensor = df.drop(columns=['retroID'])
```

[197]:
```
tensor
```

[197]:

	G	AB	R	H	2B	3B	HR	RBI	SB	CS	BB	SO	\
0	331	4	0	0	0	0	0	0	0	0.0	0	2	
1	3298	12364	2174	3771	624	98	755	2297	240	73.0	1402	1383	
2	437	944	102	216	42	6	13	94	9	8.0	86	145	
3	448	5	0	0	0	0	0	0	0	0.0	0	3	
4	15	21	1	2	0	0	0	0	0	1.0	4	5	

```
...     ...    ...   ...   ...   ...  ..  ...   ...  ...   ...   ...   ...
15187   319    795    99   199    47   4    7    80    7   5.0    57   137
15188    16     18     3     3     1   0    0     0    0   0.0     2     6
15189   266    142     5    21     2   1    0     7    0   1.0     9    39
15190   209    491    41   109    17   2    2    20    2   0.0    34    50
15191    70      0     0     0     0   0    0     0    0   0.0     0     0

         IBB  HBP  SH     SF   GIDP  NL
0        0.0    0   1    0.0    0.0   1
1      293.0   32  21  121.0  328.0   1
2        3.0    0   9    6.0   36.0   1
3        0.0    0   0    0.0    0.0   1
4        0.0    0   0    0.0    1.0   1
...      ...  ...  ..    ...    ...  ..
15187    3.0    6  20    8.0   15.0   0
15188    0.0    0   0    0.0    0.0   0
15189    0.0    0  16    0.0    3.0   1
15190    1.0    2  18    0.0    8.0   1
15191    0.0    0   0    0.0    0.0   0

[15192 rows x 18 columns]
```

[198]:
```
tensor.to_csv('../output/tensor.csv')
metadata.to_csv('../output/metadata.csv')
```

We now have a tensor with only relevant information, an indexing dictionary to get the player for each row, and a (soon to be expanded) metadata table to get more information on each player.

fielding_pre

March 9, 2020

[47]:
```
import numpy as np
import pandas as pd
pd.options.mode.chained_assignment = None  # default='warn'
```

[48]:
```
df = pd.read_csv('../data/lahman/mlb_data/Fielding.csv').sort_values('playerID')
```

[49]:
```
# This will be exported to a separate module
ids = pd.read_csv('../data/lahman/mlb_data/People.csv')
ids = ids[['playerID', 'retroID']]
id_dict = ids.set_index('playerID').to_dict()['retroID']

def get_retroid(id):
    return id_dict[id] if id_dict is not None else id
```

[50]:
```
df['playerID'] = df['playerID'].apply(get_retroid)
df.rename(columns={'playerID': 'retroID'}, inplace=True)
```

Exploration

[51]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 112837 entries, 85308 to 106797
Data columns (total 18 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   retroID  112837 non-null  object 
 1   yearID   112837 non-null  int64  
 2   stint    112837 non-null  int64  
 3   teamID   112837 non-null  object 
 4   lgID     112837 non-null  object 
 5   POS      112837 non-null  object 
 6   G        112837 non-null  int64  
 7   GS       89431 non-null   float64
 8   InnOuts  89431 non-null   float64
 9   PO       112837 non-null  int64  
 10  A        112837 non-null  int64  
 11  E        112836 non-null  float64
```

```
 12  DP       112837 non-null  int64
 13  PB       8538 non-null    float64
 14  WP       1169 non-null    float64
 15  SB       6389 non-null    float64
 16  CS       6389 non-null    float64
 17  ZR       1169 non-null    float64
dtypes: float64(8), int64(6), object(4)
memory usage: 16.4+ MB
```

[52]:
```
df.shape
```

[52]:
```
(112837, 18)
```

[53]:
```
df.columns
```

[53]:
```
Index(['retroID', 'yearID', 'stint', 'teamID', 'lgID', 'POS', 'G', 'GS',
       'InnOuts', 'PO', 'A', 'E', 'DP', 'PB', 'WP', 'SB', 'CS', 'ZR'],
      dtype='object')
```

We want to get rid of columns which already exist in the Batting DataFrame (with which we will be merging this)

[54]:
```
columns_to_drop = ['stint', 'teamID', 'lgID', 'G']
```

[55]:
```
df.drop(columns=columns_to_drop, inplace=True)
```

[56]:
```
df.head()
```

[56]:

	retroID	yearID	POS	GS	InnOuts	PO	A	E	DP	PB	WP	SB	CS	ZR
85308	aardd001	2004	P	0.0	32.0	0	0	0.0	0	NaN	NaN	NaN	NaN	NaN
101187	aardd001	2013	P	0.0	119.0	1	5	0.0	0	NaN	NaN	NaN	NaN	NaN
99344	aardd001	2012	P	0.0	3.0	0	0	0.0	0	NaN	NaN	NaN	NaN	NaN
95793	aardd001	2010	P	0.0	149.0	2	3	1.0	0	NaN	NaN	NaN	NaN	NaN
104866	aardd001	2015	P	0.0	92.0	0	1	1.0	0	NaN	NaN	NaN	NaN	NaN

Cleaning and Preprocessing

We see a lot of NaNs in the last 5 columns. According to the Lahman readme, these are:

- PB - Passed Balls (by catchers)
- WP - Wild Pitches (by catchers)
- SB - Opponent Stolen Bases (by catchers)
- CS - Opponents Caught Stealing (by catchers)
- ZR - Zone Rating

It looks like the data demands that we treat catchers separately from other position players. This intuitively makes sense from what we know about baseball, and it saves us from getting rid of a lot of data. First, though, let's look at how much of that data is missing if we JUST look at catchers.

[57]:
```
df_catchers = df[df['POS'] == 'C']
```

[58]:
```
# Get missing data in the catchers category as a percentage
100 * df_catchers.isnull().sum() / len(df)
```

[58]:
```
retroID    0.000000
yearID     0.000000
POS        0.000000
GS         1.901858
InnOuts    1.901858
PO         0.000000
A          0.000000
E          0.000000
DP         0.000000
PB         0.000000
WP         6.530659
SB         1.904517
CS         1.904517
ZR         6.530659
dtype: float64
```

Most of the percentages are negligable, but we can take a look at WP and ZR and see if the missing data is from early years.

[59]:
```
early_catchers = df_catchers[df_catchers['yearID'] < 1955]
```

[60]:
```
100 * early_catchers.isnull().sum() / len(df)
```

[60]:
```
retroID    0.000000
yearID     0.000000
POS        0.000000
GS         1.901858
InnOuts    1.901858
PO         0.000000
A          0.000000
E          0.000000
DP         0.000000
PB         0.000000
WP         1.901858
SB         1.901858
CS         1.901858
ZR         1.901858
dtype: float64
```

Definitely not the case. Let's try to narrow down where the issue is.

[61]:
```
post1985_catchers = df_catchers[df_catchers['yearID'] > 1985]
```

[62]:
```
100 * post1985_catchers.isnull().sum() / len(df)
```

[62]:
```
retroID    0.000000
yearID     0.000000
POS        0.000000
GS         0.000000
InnOuts    0.000000
PO         0.000000
A          0.000000
E          0.000000
DP         0.000000
PB         0.000000
WP         3.265773
SB         0.000000
CS         0.000000
ZR         3.265773
dtype: float64
```

[63]:
```
df_1955_to_1986_catchers = df_catchers[(df_catchers['yearID'] >= 1955) &␣
↪(df_catchers['yearID'] <= 1985)]
```

[64]:
```
100 * df_1955_to_1986_catchers.isnull().sum() / len(df)
```

[64]:
```
retroID    0.000000
yearID     0.000000
POS        0.000000
GS         0.000000
InnOuts    0.000000
PO         0.000000
A          0.000000
E          0.000000
DP         0.000000
PB         0.000000
WP         1.363028
SB         0.002659
CS         0.002659
ZR         1.363028
dtype: float64
```

[65]:
```
pre_1930_catchers = df_catchers[df_catchers['yearID'] < 1930]
```

[66]:
```
100 * pre_1930_catchers.isnull().sum() / len(df)
```

[66]:
```
retroID    0.000000
yearID     0.000000
POS        0.000000
GS         0.591118
```

```
InnOuts    0.591118
PO         0.000000
A          0.000000
E          0.000000
DP         0.000000
PB         0.000000
WP         0.591118
SB         0.591118
CS         0.591118
ZR         0.591118
dtype: float64
```

We see that the issue is mainly in the very early years, and we are fine with dropping that information by just filling it in as we did in the Batters table.

So with that, we are fine with filling all NA values with 0.

[67]:
```
df_catchers['GS'].fillna(value=0, inplace=True)
df_catchers['InnOuts'].fillna(value=0, inplace=True)
df_catchers['WP'].fillna(value=0, inplace=True)
df_catchers['SB'].fillna(value=0, inplace=True)
df_catchers['CS'].fillna(value=0, inplace=True)
df_catchers['ZR'].fillna(value=0, inplace=True)
```

[68]:
```
df['GS'].fillna(value=0, inplace=True)
df['InnOuts'].fillna(value=0, inplace=True)
#We can just drop the catcher-related columns from the original dataframe, as we␣
 ↪will also drop all catcher rows
catcher_columns = ['PB', 'WP', 'SB', 'CS', 'ZR']
df.drop(columns=catcher_columns, inplace=True)
```

Now drop all catcher rows so we have two separate dataframes, and get rid of the yearID column which we're done with and will be useless after aggregation.

[69]:
```
df = df[df['POS'] != 'C']
```

[70]:
```
df.drop(columns=['yearID'], inplace=True)
df_catchers.drop(columns=['yearID'], inplace=True)
```

[71]:
```
df.shape
```

[71]:
```
(104299, 8)
```

[72]:
```
df_catchers.shape
```

[72]:
```
(8538, 13)
```

[73]:
```
100 * df.isnull().sum() / len(df)
```

[73]:
```
retroID    0.000000
POS        0.000000
GS         0.000000
InnOuts    0.000000
PO         0.000000
A          0.000000
E          0.000959
DP         0.000000
dtype: float64
```

Now we just see a little bit of information missing from Errors, so we can fill that with 0s no problem.

[74]:
```
df['E'].fillna(value=0, inplace=True)
```

[75]:
```
100 * df.isnull().sum() / len(df)
```

[75]:
```
retroID    0.0
POS        0.0
GS         0.0
InnOuts    0.0
PO         0.0
A          0.0
E          0.0
DP         0.0
dtype: float64
```

[76]:
```
100 * df_catchers.isnull().sum() / len(df)
```

[76]:
```
retroID    0.0
POS        0.0
GS         0.0
InnOuts    0.0
PO         0.0
A          0.0
E          0.0
DP         0.0
PB         0.0
WP         0.0
SB         0.0
CS         0.0
ZR         0.0
dtype: float64
```

Aggregation

Now we just need to aggregate all stats to get total career numbers for each player.

[77]:
```
df = df.groupby('retroID').sum().reset_index()
```

[78]:
```
df_catchers = df_catchers.groupby('retroID').sum().reset_index()
```

[79]:
```
df
```

[79]:

	retroID	GS	InnOuts	PO	A	E	DP
0	aardd001	0.0	1011.0	11	29	3.0	2
1	aaroh101	2977.0	78414.0	7436	429	144.0	218
2	aarot101	206.0	6472.0	1317	113	22.0	124
3	aased001	91.0	3328.0	67	135	13.0	10
4	abada001	4.0	138.0	37	1	1.0	3
...	...	...	...	...	...	...	...
14222	zumaj001	0.0	629.0	7	14	2.0	1
14223	zupcb001	198.0	5842.0	483	22	12.0	5
14224	zuveg101	31.0	1847.0	45	145	7.0	10
14225	zuvep001	136.0	3844.0	267	415	23.0	84
14226	zycht001	1.0	218.0	1	6	1.0	0

[14227 rows x 7 columns]

[80]:
```
df_catchers
```

[80]:

	retroID	GS	InnOuts	PO	A	E	DP	PB	WP	SB	CS	\
0	adamb105	1.0	27.0	6	0	0.0	0	0.0	0.0	1.0	0.0	
1	adamb106	0.0	0.0	249	90	12.0	15	7.0	0.0	0.0	0.0	
2	adamd101	3.0	78.0	9	2	0.0	0	1.0	0.0	0.0	0.0	
3	adled101	65.0	1840.0	453	26	4.0	2	8.0	19.0	37.0	16.0	
4	afent001	20.0	613.0	123	5	1.0	3	6.0	0.0	17.0	3.0	
...	...	...	...	...	...	...	..	...	...	...	...	
1524	zimmd101	27.0	744.0	150	18	6.0	1	5.0	12.0	10.0	10.0	
1525	zimmj101	298.0	8560.0	2131	150	21.0	26	19.0	84.0	110.0	80.0	
1526	zinta001	0.0	3.0	2	0	0.0	0	0.0	0.0	0.0	0.0	
1527	zunim001	535.0	14489.0	4356	264	21.0	22	39.0	0.0	248.0	98.0	
1528	zupof101	1.0	114.0	31	1	2.0	0	1.0	1.0	2.0	1.0	

	ZR
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0
...	...
1524	3.0
1525	4.0
1526	0.0
1527	0.0

```
1528  0.0

[1529 rows x 12 columns]
```

[ ]:

add_advanced_batting

April 29, 2020

[76]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

[30]:
```
df = pd.read_csv('../core/output/batters.csv')
df_adv = pd.read_csv('../core/output/advanced_batting.csv')
```

Adding Advanced Stats

We will use a combination of wOBA, wRC+ and WAR as our overall rating - our Y value.

[43]:
```
df_adv.sort_values('retroID')
```

[43]:

	retroID	wOBA	wRC+	WAR
9203	aardd001	0.000	-100.0	-0.1
3	aaroh101	0.403	153.0	136.3
13920	aarot101	0.282	76.0	-1.7
9158	aased001	0.000	-100.0	-0.1
11841	abada001	0.184	0.0	-0.4
...	...	...	...	...
13227	zupcb001	0.293	74.0	-0.9
10487	zupof101	0.225	37.0	-0.2
11591	zuveg101	0.179	0.0	-0.3
14134	zuvep001	0.254	52.0	-2.2
6759	zycht001	0.000	NaN	0.0

[14399 rows x 4 columns]

[42]:
```
df
```

[42]:

	retroID	weight	height	debutYear	finalYear	pos_1B	pos_2B	\
0	aardd001	0.569672	0.60	2004	2015	0	0	
1	aaroh101	0.426230	0.45	1954	1976	0	0	
2	aarot101	0.467213	0.60	1962	1971	1	0	
3	aased001	0.467213	0.60	1977	1990	0	0	
4	abada001	0.442623	0.50	2001	2006	1	0	
...	...	...	...	...	...	...	...	
15288	zupcb001	0.590164	0.65	1991	1994	0	0	
15289	zupof101	0.434426	0.40	1957	1961	0	0	

```
15290  zuveg101  0.487705    0.65       1951       1959       0       0
15291  zuvep001  0.397541    0.45       1982       1991       0       0
15292  zycht001  0.467213    0.60       2015       2017       0       0

       pos_3B  pos_C  pos_OF  ...   SB    CS    BB    SO  IBB  HBP  SH   SF  \
0           0      0       0  ...    0   0.0     0     2    0    0   1    0
1           0      0       1  ...  240  73.0  1402  1383  293   32  21  121
2           0      0       0  ...    9   8.0    86   145    3    0   9    6
3           0      0       0  ...    0   0.0     0     3    0    0   0    0
4           0      0       0  ...    0   1.0     4     5    0    0   0    0
...       ...    ...     ...  ...  ...   ...   ...   ...  ...  ...  ..  ...
15288       0      0       1  ...    7   5.0    57   137    3    6  20    8
15289       0      1       0  ...    0   0.0     2     6    0    0   0    0
15290       0      0       0  ...    0   1.0     9    39    0    0  16    0
15291       0      0       0  ...    2   0.0    34    50    1    2  18    0
15292       0      0       0  ...    0   0.0     0     0    0    0   0    0

       GIDP  NL
0         0   1
1       328   1
2        36   1
3         0   1
4         1   1
...     ...  ..
15288    15   0
15289     0   0
15290     3   1
15291     8   1
15292     0   0

[15293 rows x 36 columns]
```

[35]:
```
df.shape
```

[35]:
```
(15293, 36)
```

[36]:
```
df_adv.shape
```

[36]:
```
(14399, 4)
```

[44]:
```
df = df.merge(df_adv, how='left')
```

[46]:
```
df['wOBA'].fillna(0, inplace=True)
df['wRC+'].fillna(0, inplace=True)
df['WAR'].fillna(0, inplace=True)
```

[47]:
```
df
```

[47]:
```
         retroID    weight  height  debutYear  finalYear  pos_1B  pos_2B  \
0      aardd001  0.569672    0.60       2004       2015       0       0
1      aaroh101  0.426230    0.45       1954       1976       0       0
2      aarot101  0.467213    0.60       1962       1971       1       0
3      aased001  0.467213    0.60       1977       1990       0       0
4      abada001  0.442623    0.50       2001       2006       1       0
...         ...       ...     ...        ...        ...     ...     ...
15288  zupcb001  0.590164    0.65       1991       1994       0       0
15289  zupof101  0.434426    0.40       1957       1961       0       0
15290  zuveg101  0.487705    0.65       1951       1959       0       0
15291  zuvep001  0.397541    0.45       1982       1991       0       0
15292  zycht001  0.467213    0.60       2015       2017       0       0

       pos_3B  pos_C  pos_OF  ...    SO  IBB  HBP  SH   SF  GIDP  NL   wOBA  \
0           0      0       0  ...     2    0    0   1    0     0   1  0.000
1           0      0       1  ...  1383  293   32  21  121   328   1  0.403
2           0      0       0  ...   145    3    0   9    6    36   1  0.282
3           0      0       0  ...     3    0    0   0    0     0   1  0.000
4           0      0       0  ...     5    0    0   0    0     1   1  0.184
...       ...    ...     ...  ...   ...  ...  ...  ..  ...   ...  ..    ...
15288       0      0       1  ...   137    3    6  20    8    15   0  0.293
15289       0      1       0  ...     6    0    0   0    0     0   0  0.225
15290       0      0       0  ...    39    0    0  16    0     3   1  0.179
15291       0      0       0  ...    50    1    2  18    0     8   1  0.254
15292       0      0       0  ...     0    0    0   0    0     0   0  0.000

         wRC+    WAR
0     -100.0   -0.1
1      153.0  136.3
2       76.0   -1.7
3     -100.0   -0.1
4        0.0   -0.4
...      ...    ...
15288   74.0   -0.9
15289   37.0   -0.2
15290    0.0   -0.3
15291   52.0   -2.2
15292    0.0    0.0

[15293 rows x 39 columns]
```

For now, we're just going to take the mean of the three most accepted advanced statistics, giving them equal importance. This will lead to a model that favors offense over defense, as WAR is the only stat that takes defense into account, but that's fine.

[50]:
```
df['Batting'] = df[['wOBA', 'wRC+', 'WAR']].mean(axis=1).round(3)
```

[51]:
```
df['Batting']
```

[51]:
```
0       -33.367
1        96.568
2        24.861
3       -33.367
4        -0.072
           ...
15288    24.464
15289    12.342
15290    -0.040
15291    16.685
15292     0.000
Name: Rating, Length: 15293, dtype: float64
```

[81]:
```
df['Batting'].mean()
```

[81]:
```
11.660071993722617
```

[82]:
```
df['Batting'].min()
```

[82]:
```
-33.5
```

[83]:
```
df['Batting'].max()
```

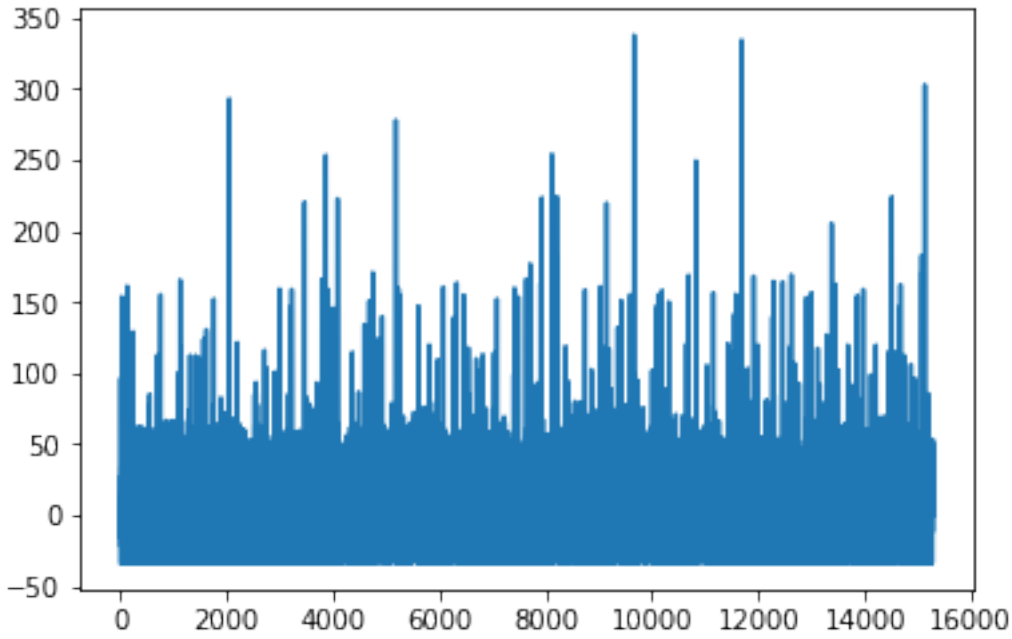
[83]:
```
337.916
```

Normalization

The Batting stat now has a very wide range which seems to trend more toward the lower end. We need to normalize the statistic so that our sigmoid output will be able to accurately predict it. For this reason, we'll use min-max normalization to get a range $[0, 1]$.

[77]:
```
plt.plot(df['Batting'])
```

[77]:
```
[<matplotlib.lines.Line2D at 0x12a2a5190>]
```

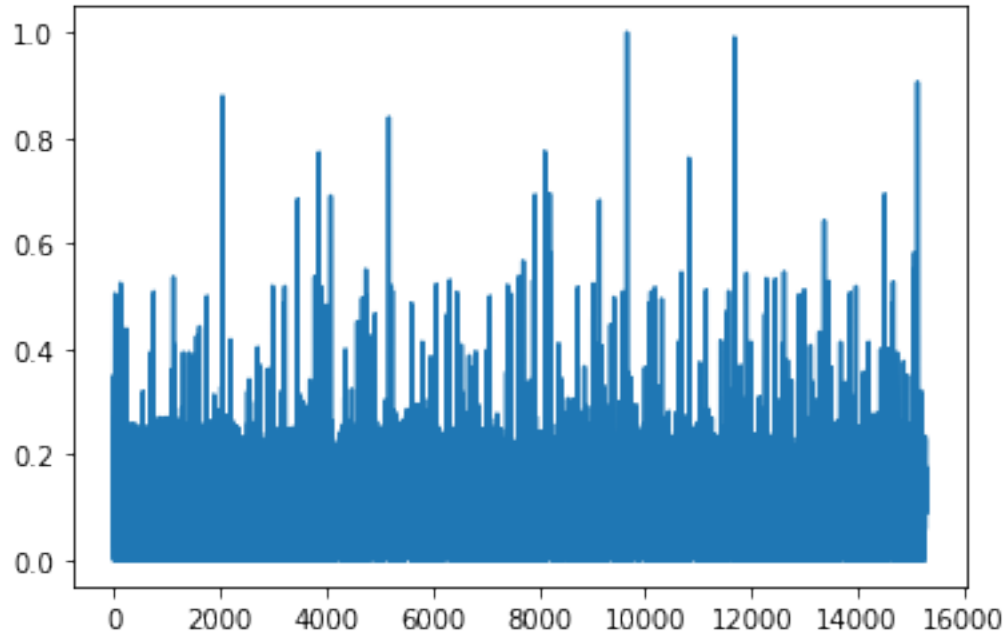


[84]:
```
from sklearn.preprocessing import MinMaxScaler
```

[85]:
```
scaler = MinMaxScaler()
```

[86]:
```
plt.plot(scaler.fit_transform(df[['Batting']]))
```

[86]: [<matplotlib.lines.Line2D at 0x121311650>]



[88]:
```
df['Batting'] = scaler.fit_transform(df[['Batting']])
```

[89]:
```
df
```

[89]:

	retroID	weight	height	debutYear	finalYear	pos_1B	pos_2B	\
0	aardd001	0.569672	0.60	2004	2015	0	0	
1	aaroh101	0.426230	0.45	1954	1976	0	0	
2	aarot101	0.467213	0.60	1962	1971	1	0	
3	aased001	0.467213	0.60	1977	1990	0	0	
4	abada001	0.442623	0.50	2001	2006	1	0	
...	...	...	...	...	...	...	...	
15288	zupcb001	0.590164	0.65	1991	1994	0	0	
15289	zupof101	0.434426	0.40	1957	1961	0	0	
15290	zuveg101	0.487705	0.65	1951	1959	0	0	
15291	zuvep001	0.397541	0.45	1982	1991	0	0	
15292	zycht001	0.467213	0.60	2015	2017	0	0	

	pos_3B	pos_C	pos_OF	...	IBB	HBP	SH	SF	GIDP	NL	wOBA	wRC+	\
0	0	0	0	...	0	0	1	0	0	1	0.000	-100.0	
1	0	0	1	...	293	32	21	121	328	1	0.403	153.0	
2	0	0	0	...	3	0	9	6	36	1	0.282	76.0	
3	0	0	0	...	0	0	0	0	0	1	0.000	-100.0	
4	0	0	0	...	0	0	0	0	1	1	0.184	0.0	
...	...	...	...	...	...	...	..	...	...	..	...	...	
15288	0	0	1	...	3	6	20	8	15	0	0.293	74.0	
15289	0	1	0	...	0	0	0	0	0	0	0.225	37.0	
15290	0	0	0	...	0	0	16	0	3	1	0.179	0.0	
15291	0	0	0	...	1	2	18	0	8	1	0.254	52.0	
15292	0	0	0	...	0	0	0	0	0	0	0.000	0.0	

	WAR	Rating
0	-0.1	0.000358
1	136.3	0.350195
2	-1.7	0.157131
3	-0.1	0.000358
4	-0.4	0.090002
...	...	...
15288	-0.9	0.156062
15289	-0.2	0.123425
15290	-0.3	0.090088
15291	-2.2	0.135118
15292	0.0	0.090195

[15293 rows x 40 columns]

We now have the Y value that our NN should attempt to predict. We'll keep wOBA, wRC+ and WAR as columns at this point so we can decide later if they need to come out.

[ ]:

pitching_pre

April 28, 2020

[211]:
```
import pandas as pd
import numpy as np
pd.options.mode.chained_assignment = None  # default='warn'
```

[212]:
```
df = pd.read_csv('../data/lahman/mlb_data/Pitching.csv')
```

[213]:
```
# This will be exported to a separate module
ids = pd.read_csv('../data/lahman/mlb_data/People.csv')
ids = ids[['playerID', 'retroID']]
id_dict = ids.set_index('playerID').to_dict()['retroID']

def get_retroid(id):
    return id_dict[id] if id_dict is not None else id
```

[214]:
```
df['playerID'] = df['playerID'].apply(get_retroid)
df.rename(columns={'playerID': 'retroID'}, inplace=True)
```

Exploration

[215]:
```
df.head()
```

[215]:

	retroID	yearID	stint	teamID	lgID	W	L	G	GS	CG	...	IBB	WP	HBP	\
0	adamb104	1919	1	PIT	NL	17	10	34	29	23	...	NaN	2	3	
1	adamw101	1919	1	PHA	AL	0	0	1	0	0	...	NaN	0	1	
2	alexg102	1919	1	CHN	NL	16	11	30	27	20	...	NaN	1	0	
3	altrn101	1919	1	WS1	AL	0	0	1	0	0	...	NaN	0	0	
4	amesr101	1919	1	SLN	NL	3	5	23	7	1	...	NaN	3	1	

	BK	BFP	GF	R	SH	SF	GIDP
0	0	1017.0	5	66	NaN	NaN	NaN
1	0	21.0	1	2	NaN	NaN	NaN
2	0	906.0	3	51	NaN	NaN	NaN
3	0	4.0	0	4	NaN	NaN	NaN
4	0	314.0	10	44	NaN	NaN	NaN

[5 rows x 30 columns]

[216]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40372 entries, 0 to 40371
Data columns (total 30 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   retroID  40372 non-null  object 
 1   yearID   40372 non-null  int64  
 2   stint    40372 non-null  int64  
 3   teamID   40372 non-null  object 
 4   lgID     40372 non-null  object 
 5   W        40372 non-null  int64  
 6   L        40372 non-null  int64  
 7   G        40372 non-null  int64  
 8   GS       40372 non-null  int64  
 9   CG       40372 non-null  int64  
 10  SHO      40372 non-null  int64  
 11  SV       40372 non-null  int64  
 12  IPouts   40372 non-null  int64  
 13  H        40372 non-null  int64  
 14  ER       40372 non-null  int64  
 15  HR       40372 non-null  int64  
 16  BB       40372 non-null  int64  
 17  SO       40372 non-null  int64  
 18  BAOpp    40360 non-null  float64
 19  ERA      40298 non-null  float64
 20  IBB      32121 non-null  float64
 21  WP       40372 non-null  int64  
 22  HBP      40372 non-null  int64  
 23  BK       40372 non-null  int64  
 24  BFP      40369 non-null  float64
 25  GF       40372 non-null  int64  
 26  R        40372 non-null  int64  
 27  SH       27512 non-null  float64
 28  SF       27512 non-null  float64
 29  GIDP     26381 non-null  float64
dtypes: float64(7), int64(20), object(3)
memory usage: 9.2+ MB
```

[217]:
```
df.columns
```

[217]:
```
Index(['retroID', 'yearID', 'stint', 'teamID', 'lgID', 'W', 'L', 'G', 'GS',
       'CG', 'SHO', 'SV', 'IPouts', 'H', 'ER', 'HR', 'BB', 'SO', 'BAOpp',
       'ERA', 'IBB', 'WP', 'HBP', 'BK', 'BFP', 'GF', 'R', 'SH', 'SF', 'GIDP'],
      dtype='object')
```

[218]:
```
columns_to_drop = ['stint', 'teamID', 'lgID']
```

[219]:
```
df.drop(columns=columns_to_drop, inplace=True)
```

[220]:
```
df.shape
```

[220]:
```
(40372, 27)
```

Cleaning and Preprocessing

[221]:
```
100 * df.isnull().sum() / len(df)
```

[221]:
```
retroID     0.000000
yearID      0.000000
W           0.000000
L           0.000000
G           0.000000
GS          0.000000
CG          0.000000
SHO         0.000000
SV          0.000000
IPouts      0.000000
H           0.000000
ER          0.000000
HR          0.000000
BB          0.000000
SO          0.000000
BAOpp       0.029724
ERA         0.183295
IBB        20.437432
WP          0.000000
HBP         0.000000
BK          0.000000
BFP         0.007431
GF          0.000000
R           0.000000
SH         31.853760
SF         31.853760
GIDP       34.655207
dtype: float64
```

[222]:
```
df_early = df[df['yearID'] <= 1930]
```

[223]:
```
100 * df_early.isnull().sum() / len(df)
```

[223]:
```
retroID    0.000000
yearID     0.000000
```

```
W           0.000000
L           0.000000
G           0.000000
GS          0.000000
CG          0.000000
SHO         0.000000
SV          0.000000
IPouts      0.000000
H           0.000000
ER          0.000000
HR          0.000000
BB          0.000000
SO          0.000000
BAOpp       0.002477
ERA         0.042108
IBB         6.442584
WP          0.000000
HBP         0.000000
BK          0.000000
BFP         0.007431
GF          0.000000
R           0.000000
SH          6.442584
SF          6.442584
GIDP        6.442584
dtype: float64
```

[224]:
```
df_modern = df[df['yearID'] >= 1980]
```

[225]:
```
100 * df_modern.isnull().sum() / len(df)
```

[225]:
```
retroID     0.000000
yearID      0.000000
W           0.000000
L           0.000000
G           0.000000
GS          0.000000
CG          0.000000
SHO         0.000000
SV          0.000000
IPouts      0.000000
H           0.000000
ER          0.000000
HR          0.000000
BB          0.000000
SO          0.000000
BAOpp       0.019816
```

```
ERA        0.056970
IBB        0.000000
WP         0.000000
HBP        0.000000
BK         0.000000
BFP        0.000000
GF         0.000000
R          0.000000
SH         0.000000
SF         0.000000
GIDP       0.000000
dtype: float64
```

Luckily the more modern data is barely missing any information.

[226]:
```
df_mid = df[(df['yearID'] > 1935) & (df['yearID'] < 1975)]
```

[227]:
```
100 * df_mid.isnull().sum() / len(df)
```

[227]:
```
retroID     0.000000
yearID      0.000000
W           0.000000
L           0.000000
G           0.000000
GS          0.000000
CG          0.000000
SHO         0.000000
SV          0.000000
IPouts      0.000000
H           0.000000
ER          0.000000
HR          0.000000
BB          0.000000
SO          0.000000
BAOpp       0.007431
ERA         0.066878
IBB        11.428713
WP          0.000000
HBP         0.000000
BK          0.000000
BFP         0.000000
GF          0.000000
R           0.000000
SH         22.845041
SF         22.845041
GIDP       25.646488
dtype: float64
```

We see that much of the lost data comes within this 40-year span. I think that given what the major missing information is - intentional bases on balls, sacrifice hits, sacrifice flies and grounded into double play - and the fact that these statistics are not often used as primary indicators of a pitcher's ability, coupled with the fact that it's mostly localized within less than half of our time frame, I can be forgiven for just filling these values as 0.

[228]:
```
df['IBB'].fillna(0, inplace=True)
df['SH'].fillna(0, inplace=True)
df['SF'].fillna(0, inplace=True)
df['GIDP'].fillna(0, inplace=True)
```

[229]:
```
100 * df.isnull().sum() / len(df)
```

[229]:
```
retroID    0.000000
yearID     0.000000
W          0.000000
L          0.000000
G          0.000000
GS         0.000000
CG         0.000000
SHO        0.000000
SV         0.000000
IPouts     0.000000
H          0.000000
ER         0.000000
HR         0.000000
BB         0.000000
SO         0.000000
BAOpp      0.029724
ERA        0.183295
IBB        0.000000
WP         0.000000
HBP        0.000000
BK         0.000000
BFP        0.007431
GF         0.000000
R          0.000000
SH         0.000000
SF         0.000000
GIDP       0.000000
dtype: float64
```

We're left with three fields that having missing data: opponents' batting average, earned run average and batters faced by pitcher. We'll have to do some data exploration on these because I don't want to just fill them with 0s.

Missing Values: BAOpp

[230]:
```
df_baopp_missing = df[df['BAOpp'].isnull()]
```

[231]:
```
df_baopp_missing.shape
```

[231]:
```
(12, 27)
```

[232]:
```
df_baopp_missing.sort_values('retroID')
```

[232]:

	retroID	yearID	W	L	G	GS	CG	SHO	SV	IPouts	...	IBB	WP	HBP	\
14000	apodb101	1973	0	0	1	0	0	0	0	0	...	0.0	0	0	
19114	arrof001	1986	0	0	1	0	0	0	0	0	...	0.0	0	0	
39581	brotr001	2018	0	0	1	0	0	0	0	0	...	0.0	0	0	
36447	dunnj001	2014	0	0	1	0	0	0	0	2	...	0.0	2	0	
38848	eschj001	2017	0	0	1	0	0	0	0	0	...	0.0	0	0	
3709	fordw103	1936	0	0	1	0	0	0	0	0	...	0.0	0	0	
297	glasn101	1920	0	0	1	0	0	0	0	7	...	0.0	0	1	
26791	halts001	2000	0	0	1	0	0	0	0	0	...	0.0	0	0	
27036	radis001	2000	0	0	1	0	0	0	0	0	...	0.0	0	0	
36189	tolls002	2013	0	0	1	0	0	0	0	0	...	0.0	0	0	
36208	villb002	2013	0	0	1	0	0	0	0	0	...	0.0	0	0	
13621	youn1101	1971	0	0	1	0	0	0	0	0	...	0.0	0	0	

	BK	BFP	GF	R	SH	SF	GIDP
14000	0	2.0	0	1	0.0	0.0	0.0
19114	0	3.0	0	0	0.0	0.0	0.0
39581	0	2.0	0	1	0.0	0.0	0.0
36447	0	2.0	0	0	0.0	1.0	0.0
38848	0	2.0	0	0	0.0	0.0	0.0
3709	0	3.0	0	2	0.0	0.0	0.0
297	0	12.0	0	4	0.0	0.0	0.0
26791	0	1.0	0	0	0.0	0.0	0.0
27036	0	1.0	0	0	0.0	0.0	0.0
36189	0	2.0	0	0	0.0	0.0	0.0
36208	0	1.0	1	0	0.0	0.0	0.0
13621	0	0.0	0	0	0.0	0.0	0.0

[12 rows x 27 columns]

Nobody appears in this table more than once. We'll hope that we can get career numbers for them and fill with the average. For anyone who only appears once I'll go with the league average, and I'll add a standard deviation since they probably we're exactly middle-of-the-road. Probably not the best way but it's only a few datapoints.

[233]:
```
baopp_checks = df[df['retroID'].isin(df_baopp_missing['retroID'])].
↪sort_values('retroID')
```

[234]:
```
baopp_checks['retroID'].value_counts()
```

[234]:
```
radis001    11
arrof001     9
brotr001     7
tolls002     5
apodb101     5
villb002     4
dunnj001     2
eschj001     2
halts001     2
younl101     1
glasn101     1
fordw103     1
Name: retroID, dtype: int64
```

We only have three data points with one year of appearances. We'll fill those with the league average.

[235]:
```
val_counts = baopp_checks['retroID'].value_counts()
```

[236]:
```
from itertools import compress
# Get list of retroIDs of players who only have one year appearance
```

[237]:
```
one_time_players = list(compress(val_counts.index, val_counts.eq(1)))
```

[238]:
```
df[df['retroID'].isin(one_time_players)]
```

[238]:

	retroID	yearID	W	L	G	GS	CG	SHO	SV	IPouts	...	IBB	WP	HBP	\
297	glasn101	1920	0	0	1	0	0	0	0	7	...	0.0	0	1	
3709	fordw103	1936	0	0	1	0	0	0	0	0	...	0.0	0	0	
13621	younl101	1971	0	0	1	0	0	0	0	0	...	0.0	0	0	

	BK	BFP	GF	R	SH	SF	GIDP
297	0	12.0	0	4	0.0	0.0	0.0
3709	0	3.0	0	2	0.0	0.0	0.0
13621	0	0.0	0	0	0.0	0.0	0.0

[3 rows x 27 columns]

[239]:
```
df['BAOpp'].mean()
```

[239]:
```
0.27445659068384537
```

[240]:
```
df['BAOpp'].std()
```

[240]:
```
0.07751058079199835
```

[241]:
```
filled_baopp = df['BAOpp'].mean() + df['BAOpp'].std()
```

[242]:
```
filled_baopp
```

[242]:
```
0.3519671714758437
```

[243]:
```
df.loc[df['retroID'].isin(one_time_players), ['BAOpp']] = filled_baopp
```

[244]:
```
df[df['retroID'].isin(one_time_players)]['BAOpp']
```

[244]:
```
297      0.351967
3709     0.351967
13621    0.351967
Name: BAOpp, dtype: float64
```

[245]:
```
df_baopp_missing = df[df['BAOpp'].isnull()].sort_values('retroID')
```

[246]:
```
df_baopp_missing
```

[246]:

	retroID	yearID	W	L	G	GS	CG	SHO	SV	IPouts	...	IBB	WP	HBP	\
14000	apodb101	1973	0	0	1	0	0	0	0	0	...	0.0	0	0	
19114	arrof001	1986	0	0	1	0	0	0	0	0	...	0.0	0	0	
39581	brotr001	2018	0	0	1	0	0	0	0	0	...	0.0	0	0	
36447	dunnj001	2014	0	0	1	0	0	0	0	2	...	0.0	2	0	
38848	eschj001	2017	0	0	1	0	0	0	0	0	...	0.0	0	0	
26791	halts001	2000	0	0	1	0	0	0	0	0	...	0.0	0	0	
27036	radis001	2000	0	0	1	0	0	0	0	0	...	0.0	0	0	
36189	tolls002	2013	0	0	1	0	0	0	0	0	...	0.0	0	0	
36208	villb002	2013	0	0	1	0	0	0	0	0	...	0.0	0	0	

	BK	BFP	GF	R	SH	SF	GIDP
14000	0	2.0	0	1	0.0	0.0	0.0
19114	0	3.0	0	0	0.0	0.0	0.0
39581	0	2.0	0	1	0.0	0.0	0.0
36447	0	2.0	0	0	0.0	1.0	0.0
38848	0	2.0	0	0	0.0	0.0	0.0
26791	0	1.0	0	0	0.0	0.0	0.0
27036	0	1.0	0	0	0.0	0.0	0.0
36189	0	2.0	0	0	0.0	0.0	0.0
36208	0	1.0	1	0	0.0	0.0	0.0

[9 rows x 27 columns]

Now we just have to worry about the players with at least two years of appearances.

[247]:
```
baopp_checks = df[df['retroID'].isin(df_baopp_missing['retroID'])].
↪sort_values('retroID')
```

[248]:
```
baopp_checks['retroID'].value_counts()
```

[248]:
```
radis001    11
arrof001     9
brotr001     7
tolls002     5
apodb101     5
villb002     4
dunnj001     2
eschj001     2
halts001     2
Name: retroID, dtype: int64
```

[249]:
```
one_time_players = list(val_counts.index)
```

[250]:
```
one_time_players
```

[250]:
```
['radis001',
 'arrof001',
 'brotr001',
 'tolls002',
 'apodb101',
 'villb002',
 'dunnj001',
 'eschj001',
 'halts001',
 'younl101',
 'glasn101',
 'fordw103']
```

[251]:
```
df[df['retroID'] == 'radis001']['BAOpp']
```

[251]:
```
21355    0.241
21865    0.206
22335    0.243
22871    0.268
23957    0.309
24557    0.264
25145    0.236
25740    0.272
26377    0.270
27036      NaN
27708    0.400
Name: BAOpp, dtype: float64
```

[252]:
```
df[df['retroID'] == 'radis001']['BAOpp'].mean().round(4)
```

[252]:
```
0.2709
```

This looks good, so we'll iterate through and assign each player's missing BAOpp as his career

mean for that state.

[253]:
```
df['BAOpp'] = df.groupby("retroID")['BAOpp'].transform(lambda baopp: baopp.
↪fillna(baopp.mean()))
```

[254]:
```
100 * df.isnull().sum() / len(df)
```

[254]:
```
retroID    0.000000
yearID     0.000000
W          0.000000
L          0.000000
G          0.000000
GS         0.000000
CG         0.000000
SHO        0.000000
SV         0.000000
IPouts     0.000000
H          0.000000
ER         0.000000
HR         0.000000
BB         0.000000
SO         0.000000
BAOpp      0.000000
ERA        0.183295
IBB        0.000000
WP         0.000000
HBP        0.000000
BK         0.000000
BFP        0.007431
GF         0.000000
R          0.000000
SH         0.000000
SF         0.000000
GIDP       0.000000
dtype: float64
```

[255]:
```
df.head()
```

[255]:

	retroID	yearID	W	L	G	GS	CG	SHO	SV	IPouts	...	IBB	WP	HBP	\
0	adamb104	1919	17	10	34	29	23	6	1	790	...	0.0	2	3	
1	adamw101	1919	0	0	1	0	0	0	0	14	...	0.0	0	1	
2	alexg102	1919	16	11	30	27	20	9	1	705	...	0.0	1	0	
3	altrn101	1919	0	0	1	0	0	0	0	0	...	0.0	0	0	
4	amesr101	1919	3	5	23	7	1	0	1	210	...	0.0	3	1	

	BK	BFP	GF	R	SH	SF	GIDP
0	0	1017.0	5	66	0.0	0.0	0.0
1	0	21.0	1	2	0.0	0.0	0.0

```
2   0   906.0   3  51  0.0  0.0   0.0
3   0     4.0   0   4  0.0  0.0   0.0
4   0   314.0  10  44  0.0  0.0   0.0

[5 rows x 27 columns]
```

Missing Values: ERA

[256]:
```
df_era_missing = df[df['ERA'].isnull()].sort_values('retroID')
```

[257]:
```
df_era_missing
```

[257]:
```
        retroID  yearID  W  L  G  GS  CG  SHO  SV  IPouts  ...  IBB  WP  HBP  \
3      altrn101    1919  0  0  1   0   0    0   0       0  ...  0.0   0    0
20491  alvaw001    1989  0  1  1   1   0    0   0       0  ...  0.0   0    0
14000  apodb101    1973  0  0  1   0   0    0   0       0  ...  0.0   0    0
19114  arrof001    1986  0  0  1   0   0    0   0       0  ...  0.0   0    0
663    bents101    1922  0  0  1   0   0    0   0       0  ...  0.0   0    0
...         ...     ... .. .. ..  .. ..  ...  ..     ...  ...  ...  ..  ...
40336  weisz001    2018  0  0  1   0   0    0   0       0  ...  0.0   0    0
6277   willa103    1946  0  0  1   0   0    0   0       0  ...  0.0   0    0
10476  willt102    1962  0  0  1   0   0    0   0       0  ...  0.0   0    0
18235  wortr101    1983  0  0  1   0   0    0   0       0  ...  0.0   0    0
13621  younl101    1971  0  0  1   0   0    0   0       0  ...  0.0   0    0

       BK  BFP  GF  R   SH   SF  GIDP
3       0  4.0   0  4  0.0  0.0   0.0
20491   0  5.0   0  3  0.0  0.0   0.0
14000   0  2.0   0  1  0.0  0.0   0.0
19114   0  3.0   0  0  0.0  0.0   0.0
663     0  2.0   0  0  0.0  0.0   0.0
...    ..  ...  .. ..  ...  ...   ...
40336   0  4.0   0  4  0.0  0.0   0.0
6277    0  2.0   0  0  0.0  0.0   0.0
10476   0  3.0   0  1  0.0  0.0   0.0
18235   0  4.0   0  1  0.0  0.0   0.0
13621   0  0.0   0  0  0.0  0.0   0.0

[74 rows x 27 columns]
```

[258]:
```
df_era_missing['retroID'].nunique()
```

[258]:
```
74
```

74 rows and 74 unique IDs means that each of these players is only missing the ERA stat for one year. We'll first see, like with BAOpp, if they played other years.

[259]:
```
era_checks = df[df['retroID'].isin(df_era_missing['retroID'])].
 ↪sort_values('retroID')
```

[260]:
```
val_counts = era_checks['retroID'].value_counts()
```

[261]:
```
one_time_players = list(compress(val_counts.index, val_counts.eq(1)))
```

[262]:
```
one_time_players
```

[262]:
```
['russr102',
 'moorb104',
 'palam101',
 'musis101',
 'garda103',
 'weisz001',
 'koenw101',
 'bents101',
 'fordw103',
 'schej101',
 'brucf101',
 'davav101',
 'hamad101',
 'walkm101',
 'sundg101',
 'younl101',
 'browj102']
```

[263]:
```
df['ERA'].mean()
```

[263]:
```
5.165393567919002
```

[264]:
```
df['ERA'].std()
```

[264]:
```
5.2791159271962815
```

Intuitively, 5.17 is a bit of a high ERA. Though the stat can grow infinitely in theory and low numbers are very difficult, I don't want to assign 10 to the missing values. It's just too much. I'll just do mean + std/2.

[265]:
```
filled_era = df['ERA'].mean() + (df['ERA'].std())/2
```

[266]:
```
df.loc[df['retroID'].isin(one_time_players), ['ERA']] = filled_era
df[df['retroID'].isin(one_time_players)]['ERA']
```

[266]:
```
101      7.804952
173      7.804952
663      7.804952
```

```
722      7.804952
931      7.804952
1447     7.804952
1755     7.804952
2154     7.804952
3709     7.804952
4485     7.804952
4513     7.804952
7678     7.804952
8773     7.804952
9903     7.804952
12532    7.804952
13621    7.804952
40336    7.804952
Name: ERA, dtype: float64
```

[267]:
```
df_era_missing = df[df['ERA'].isnull()].sort_values('retroID')
```

We'll continue to follow the same method as for BAOpp with the rest of the missing values.

[268]:
```
df_era_missing.shape
```

[268]:
```
(57, 27)
```

[269]:
```
era_checks = df[df['retroID'].isin(df_era_missing['retroID'])].
 ↪sort_values('retroID')
era_checks['retroID'].value_counts()
```

[269]:
```
hillr001    16
choar001    16
mclic101    15
alvaw001    15
farme101    14
medid101    14
kosld101    13
coopm101    13
radis001    11
burkb102    10
owchb001    10
milna101    10
arrof001     9
chent101     9
harvb001     9
perim001     9
deanp101     9
ray-j101     9
pennb001     7
```

```
navaj101     7
brotr001     7
jonen001     7
painp101     6
willt102     6
moorc101     6
tolls002     5
luebs101     5
reina102     5
scarm101     5
mccud001     5
apodb101     5
blake101     4
villb002     4
kreur101     4
wortr101     4
tankd001     3
pitls101     3
stufp101     3
sabee001     3
geard101     3
kochm001     3
vaugp101     3
roeto101     2
urdal001     2
willa103     2
green002     2
dibup101     2
uhl-b101     2
wardd101     2
kella101     2
eschj001     2
kammb101     2
jeant101     2
engej101     2
smitd105     2
halts001     2
altrn101     2
Name: retroID, dtype: int64
```

[270]:
```
df['ERA'] = df.groupby("retroID")['ERA'].transform(lambda era: era.fillna(era.
 ↪mean()))
```

[271]:
```
100 * df.isnull().sum() / len(df)
```

[271]:
```
retroID    0.000000
yearID     0.000000
```

```
W           0.000000
L           0.000000
G           0.000000
GS          0.000000
CG          0.000000
SHO         0.000000
SV          0.000000
IPouts      0.000000
H           0.000000
ER          0.000000
HR          0.000000
BB          0.000000
SO          0.000000
BAOpp       0.000000
ERA         0.000000
IBB         0.000000
WP          0.000000
HBP         0.000000
BK          0.000000
BFP         0.007431
GF          0.000000
R           0.000000
SH          0.000000
SF          0.000000
GIDP        0.000000
dtype: float64
```

Missing Values: BFP

[272]:
```
df_bfp_missing = df[df['BFP'].isnull()].sort_values('retroID')
```

[273]:
```
df_bfp_missing
```

[273]:
```
       retroID  yearID  W  L   G  GS  CG  SHO  SV  IPouts  ...  IBB  WP  HBP  \
709   fourj101    1922  0  0   1   0   0    0   0       3  ...  0.0   0    0
1171  jamel101    1924  0  0   1   0   0    0   0       3  ...  0.0   0    0
802   pierb103    1922  3  9  29  12   7    1   0     364  ...  0.0   4    6

      BK  BFP  GF   R   SH   SF  GIDP
709    0  NaN   1   0  0.0  0.0   0.0
1171   0  NaN   1   2  0.0  0.0   0.0
802    0  NaN  10  77  0.0  0.0   0.0

[3 rows x 27 columns]
```

These amounts are negligable. Rather than take averages or set to 0, I'm going to get a little clever. A pitcher intuitively faces hitters until he gets an out, and comes out of the game if he can't get one. There's a lot of work I could do to get a good approximation, but since I'm only filling 3 rows

out of over 40,000 I'm just going to set the missing values to (IPouts - G). This gives us the number of outs a pitcher earned minus 1 for each game he appeared in (presumably this 1 represents the final batter, whom the pitcher did not get out).

[274]:
```
df['BFP'].fillna(df['IPouts'] - df['G'], inplace=True)
```

[275]:
```
100 * df.isnull().sum() / len(df)
```

[275]:
```
retroID    0.0
yearID     0.0
W          0.0
L          0.0
G          0.0
GS         0.0
CG         0.0
SHO        0.0
SV         0.0
IPouts     0.0
H          0.0
ER         0.0
HR         0.0
BB         0.0
SO         0.0
BAOpp      0.0
ERA        0.0
IBB        0.0
WP         0.0
HBP        0.0
BK         0.0
BFP        0.0
GF         0.0
R          0.0
SH         0.0
SF         0.0
GIDP       0.0
dtype: float64
```

Data Aggregation

Now we can group by retroID, but we need to be more careful than we were with fielding, catching and batting. Some of these stats are averages and some are sum totals, so when we group by we need to handle them differently. We'll split them into two dataframes and do a join. The splitting step will require some intuitive knowledge about baseball statistics. But before we do all of this, we can now get rid of the yearID column.

[276]:
```
df.drop(columns=['yearID'], inplace=True)
```

[277]:
```
df.columns
```

[277]:
```
Index(['retroID', 'W', 'L', 'G', 'GS', 'CG', 'SHO', 'SV', 'IPouts', 'H', 'ER',
       'HR', 'BB', 'SO', 'BAOpp', 'ERA', 'IBB', 'WP', 'HBP', 'BK', 'BFP', 'GF',
       'R', 'SH', 'SF', 'GIDP'],
      dtype='object')
```

[278]:
```
average_stats = ['BAOpp', 'ERA']
```

It's only two columns that are averages, and we could probably do without ERA since it's a function of batters faced and runs allowed. We'll keep it since it's such a fundamental statistic in the sport and we have to split anyway for BAOpp, which is a very important one to keep track of.

[279]:
```
df_avgs = df[['retroID', 'BAOpp', 'ERA']]
```

[280]:
```
df_avgs.head()
```

[280]:

	retroID	BAOpp	ERA
0	adamb104	0.22	1.98
1	adamw101	0.38	3.86
2	alexg102	0.21	1.72
3	altrn101	1.00	0.00
4	amesr101	0.31	4.89

[281]:
```
df_sums = df.drop(columns=average_stats)
```

[282]:
```
df_sums.head()
```

[282]:

	retroID	W	L	G	GS	CG	SHO	SV	IPouts	H	...	IBB	WP	HBP	BK	\
0	adamb104	17	10	34	29	23	6	1	790	213	...	0.0	2	3	0	
1	adamw101	0	0	1	0	0	0	0	14	7	...	0.0	0	1	0	
2	alexg102	16	11	30	27	20	9	1	705	180	...	0.0	1	0	0	
3	altrn101	0	0	1	0	0	0	0	0	4	...	0.0	0	0	0	
4	amesr101	3	5	23	7	1	0	1	210	88	...	0.0	3	1	0	

	BFP	GF	R	SH	SF	GIDP
0	1017.0	5	66	0.0	0.0	0.0
1	21.0	1	2	0.0	0.0	0.0
2	906.0	3	51	0.0	0.0	0.0
3	4.0	0	4	0.0	0.0	0.0
4	314.0	10	44	0.0	0.0	0.0

[5 rows x 24 columns]

[283]:
```
df_avgs.shape
```

[283]:
```
(40372, 3)
```

[284]:
```
df_sums.shape
```

[284]:
```
(40372, 24)
```

[289]:
```
df_avgs = df_avgs.groupby('retroID').mean().round(4).reset_index()
```

[292]:
```
df_avgs.shape
```

[292]:
```
(7835, 3)
```

[288]:
```
df_sums = df_sums.groupby('retroID').sum().reset_index()
```

[293]:
```
df_sums.shape
```

[293]:
```
(7835, 24)
```

[291]:
```
pd.merge(df_avgs, df_sums, on='retroID')
```

[291]:

	retroID	BAOpp	ERA	W	L	G	GS	CG	SHO	SV	...	IBB	WP	\
0	aardd001	0.2574	5.1944	16	18	331	0	0	0	69	...	22.0	12	
1	aased001	0.2508	3.4931	66	60	448	91	22	5	82	...	45.0	22	
2	abadf001	0.2501	4.0733	8	27	363	6	0	0	2	...	10.0	9	
3	abbog001	0.2786	4.3317	62	83	248	206	37	5	0	...	28.0	18	
4	abboj001	0.2804	4.4964	87	108	263	254	31	6	0	...	30.0	53	
...	...	...	...	..	...	...	...	..	...	..	...	...	..	
7830	zolds101	0.2700	3.6890	43	53	250	93	30	5	8	...	0.0	8	
7831	zubeb101	0.2717	5.3617	43	42	224	65	23	3	6	...	0.0	28	
7832	zumaj001	0.2286	3.4420	13	12	171	0	0	0	5	...	11.0	16	
7833	zuveg101	0.2760	4.1280	32	36	265	31	9	2	40	...	29.0	10	
7834	zycht001	0.2183	2.8000	7	3	70	1	0	0	1	...	5.0	2	

	HBP	BK	BFP	GF	R	SH	SF	GIDP
0	16	1	1475.0	141	169	17.0	11.0	21.0
1	7	3	4730.0	235	503	50.0	34.0	106.0
2	12	2	1350.0	96	137	7.0	12.0	22.0
3	32	5	5508.0	13	707	60.0	39.0	111.0
4	32	11	7211.0	5	880	70.0	47.0	200.0
...	...	..	...	...	...	...	...	...
7830	3	4	3946.0	78	423	0.0	0.0	0.0
7831	4	1	3476.0	90	418	0.0	0.0	0.0
7832	4	0	911.0	35	80	6.0	10.0	10.0
7833	27	1	2746.0	139	296	0.0	0.0	0.0
7834	8	1	309.0	14	24	1.0	3.0	6.0

[7835 rows x 26 columns]

This gives us the appropriate amount of rows and columns, so the merge worked. We'll send this as our final output.

[294]:
```
df = pd.merge(df_avgs, df_sums, on='retroID')
```

[296]:
```
df.shape
```

[296]:
```
(7835, 26)
```

We're ready to export the resulting table by saving to a csv.

add_advanced_pitching_stats

April 29, 2020

[53]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

[54]:
```
df = pd.read_csv('../core/output/pitchers.csv')
df_adv = pd.read_csv('../core/output/advanced_pitching.csv')
```

Adding Advanced Stats

[55]:
```
df_adv.sort_values('retroID')
```

[55]:

	retroID	IP	K/9	BB/9	HR/9	BABIP	LOB%	ERA	FIP	WAR
2866	aardd001	0.062360	9.08	4.89	1.09	0.285	74.5	4.27	4.45	1.1
841	aased001	0.205233	5.20	3.71	0.72	0.282	73.4	3.80	3.85	11.7
3237	abadf001	0.061102	7.62	3.16	1.14	0.281	77.7	3.67	4.24	0.6
949	abbog001	0.237967	3.39	2.46	1.13	0.278	69.3	4.39	4.46	10.2
394	abboj001	0.309765	4.77	3.33	0.83	0.295	70.0	4.25	4.25	22.7
...	...	...	...	...	...	...	...	...	...	...
1030	zolds101	0.171925	2.00	2.91	0.52	0.267	70.7	3.54	3.80	9.3
1934	zubeb101	0.145445	4.39	5.36	0.40	0.283	69.0	4.28	3.96	3.3
2098	zumaj001	0.038711	9.01	4.89	0.77	0.267	78.7	3.00	3.94	2.7
2399	zuveg101	0.118817	3.12	2.84	0.78	0.270	73.2	3.54	3.93	1.9
2808	zycht001	0.013360	9.91	4.21	0.37	0.293	79.1	2.72	3.22	1.1

[8025 rows x 10 columns]

[56]:
```
df
```

[56]:

	retroID	BAOpp	ERA	CG	SHO	IPouts	H	ER	HR	BB	...	WP	\
0	aardd001	0.2574	5.1944	0	0	1011	296	160	41	183	...	12	
1	aased001	0.2508	3.4931	22	5	3328	1085	468	89	457	...	22	
2	abadf001	0.2447	4.0810	0	0	992	309	135	42	116	...	10	
3	abbog001	0.2786	4.3317	37	5	3858	1405	627	162	352	...	18	
4	abboj001	0.2804	4.4964	31	6	5022	1779	791	154	620	...	53	
...	...	...	...	..	...	...	...	...	...	...	...	..	
8020	zolds101	0.2700	3.6890	30	5	2788	956	366	54	301	...	8	
8021	zubeb101	0.2717	5.3617	23	3	2358	767	374	35	468	...	28	
8022	zumaj001	0.2286	3.4420	0	0	629	169	71	18	114	...	16	

```
8023  zuveg101  0.2760  4.1280   9    2    1927   660  253   56  203  ...  10
8024  zycht001  0.2183  2.8000   0    0     218    57   22    3   34  ...   2

      HBP  BK   BFP   GF    R  SH  SF  GIDP        K%
0      16   1  1475  141  169  17  11    21  0.230508
1       7   3  4730  235  503  50  34   106  0.135518
2      12   2  1399   97  143   7  12    25  0.200143
3      32   5  5508   13  707  60  39   111  0.087872
4      32  11  7211    5  880  70  47   200  0.123145
...   ...  ..   ...  ...  ...  ..  ..   ...       ...
8020    3   4  3946   78  423   0   0     0  0.052458
8021    4   1  3476   90  418   0   0     0  0.110184
8022    4   0   911   35   80   6  10    10  0.230516
8023   27   1  2746  139  296   0   0     0  0.081209
8024    8   1   309   14   24   1   3     6  0.258900

[8025 rows x 22 columns]
```

[57]:
```
df = df.drop(columns=['ERA'])
```

[58]:
```
df = df.merge(df_adv, on='retroID' ,how='left')
```

[59]:
```
100 * df.isnull().sum() / len(df)
```

[59]:
```
retroID    0.0
BAOpp      0.0
CG         0.0
SHO        0.0
IPouts     0.0
H          0.0
ER         0.0
HR         0.0
BB         0.0
SO         0.0
IBB        0.0
WP         0.0
HBP        0.0
BK         0.0
BFP        0.0
GF         0.0
R          0.0
SH         0.0
SF         0.0
GIDP       0.0
K%         0.0
IP         0.0
K/9        0.0
```

```
BB/9       0.0
HR/9       0.0
BABIP      0.0
LOB%       0.0
ERA        0.0
FIP        0.0
WAR        0.0
dtype: float64
```

[60]:
```
df['Pitching'] = df[['K%', 'ERA', 'FIP', 'WAR']].mean(axis=1).round(3)
```

[61]:
```
df['Pitching']
```

[61]:
```
0       2.513
1       4.871
2       2.178
3       4.784
4       7.831
        ...
8020    4.173
8021    2.913
8022    2.468
8023    2.363
8024    1.825
Name: Pitching, Length: 8025, dtype: float64
```

Finalizing the new Pitching statistic

[64]:
```
df['Pitching'].mean()
```

[64]:
```
3.627380436137072
```

[65]:
```
df['Pitching'].min()
```

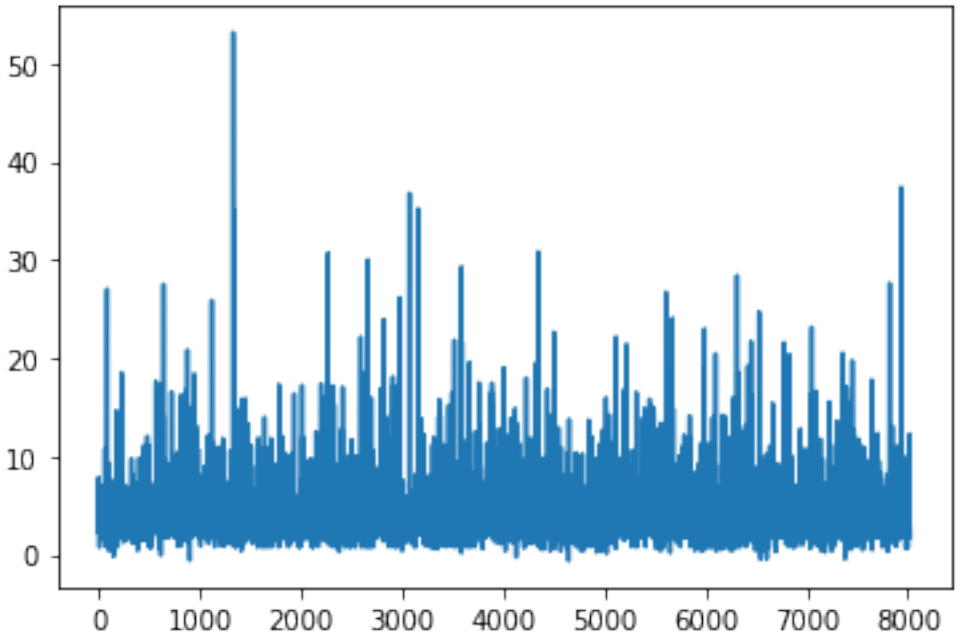
[65]:
```
-0.654
```

[66]:
```
df['Pitching'].max()
```

[66]:
```
53.138
```

[63]:
```
plt.plot(df['Pitching'])
```

[63]:
```
[<matplotlib.lines.Line2D at 0x127615250>]
```



[67]:
```
df[df['Pitching'] == df['Pitching'].max()]
```

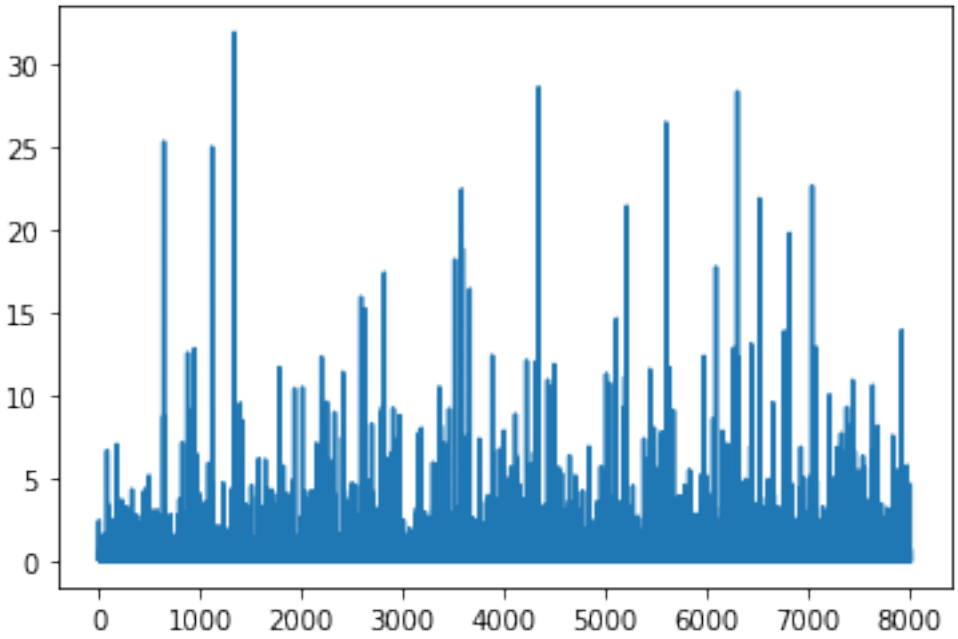
[67]:
```
       retroID  BAOpp  CG  SHO  IPouts  H  ER  HR  BB  SO  ...        IP  \
1333  cleaj101   0.83   0    0       1  5   7   0   3   1  ...  0.000019

       K/9  BB/9  HR/9  BABIP  LOB%    ERA    FIP  WAR  Pitching
1333  27.0  81.0   0.0    1.0  12.5  189.0  23.54 -0.1    53.138

[1 rows x 31 columns]
```

This immediately demonstrates an issue with our Pitching stat - a player with very few appearances, but who did well in those appearances, will be skewed too high. We need to also take innings pitched into account.

[91]:
```
df['Pitching'] = df[['K%', 'ERA', 'FIP', 'WAR']].mean(axis=1).round(3) * df['IP']
```

[92]:
```
plt.plot(df['Pitching'])
```

[92]:
```
[<matplotlib.lines.Line2D at 0x12793b850>]
```



[93]:
```
df[df['Pitching'] == df['Pitching'].max()]
```

[93]:
```
       retroID   BAOpp   CG  SHO  IPouts     H    ER   HR    BB    SO  ...  \
1341  clemr001  0.2308  118   46   14750  4185  1707  363  1580  4672  ...

             IP   K/9  BB/9  HR/9  BABIP  LOB%   ERA   FIP    WAR   Pitching
1341  0.909717  8.55  2.89  0.66  0.284  74.6  3.12  3.09  133.7  31.871935

[1 rows x 31 columns]
```

This looks like it worked, but let's explore deeper.

[94]:
```
df['Pitching'].mean()
```

[94]:
```
0.49352977097414313
```

[95]:
```
df['Pitching'].min()
```

[95]:
```
-5.0887375e-05
```

[96]:
```
df['Pitching'].max()
```

[96]:
```
31.871935094999998
```

[97]:
```
df.sort_values('Pitching').tail(10)
```

[97]:
```
        retroID   BAOpp   CG  SHO  IPouts     H    ER   HR    BB    SO  ...  \
5219  niekp001  0.2570  245   45   16213  5044  2012  482  1809  3342  ...
6529  seavt001  0.2285  231   61   14348  3971  1521  380  1390  3640  ...
3588  johnr005  0.2252  100   37   12406  3346  1513  411  1497  4875  ...
7049  suttd001  0.2376  178   58   15847  4692  1914  472  1343  3574  ...
1121  carls001  0.2546  254   55   15652  4672  1864  414  1833  4136  ...
645   blylb001  0.2487  242   60   14910  4632  1830  430  1322  3701  ...
5607  perrg101  0.2540  303   53   16051  4938  1846  399  1379  3534  ...
6311  ryann001  0.2088  222   61   16158  3923  1911  321  2795  5714  ...
4347  maddg002  0.2553  109   35   15025  4726  1756  353   999  3371  ...
1341  clemr001  0.2308  118   46   14750  4185  1707  363  1580  4672  ...

             IP    K/9  BB/9  HR/9  BABIP  LOB%   ERA   FIP    WAR   Pitching
5219  1.000000   5.57  3.01  0.80  0.270  73.6  3.35  3.62   78.5  21.404000
6529  0.884921   6.85  2.62  0.72  0.259  76.7  2.86  3.04   92.7  21.854894
3588  0.765178  10.61  3.26  0.89  0.291  74.7  3.29  3.19  110.4  22.412829
7049  0.977406   6.09  2.29  0.80  0.261  73.6  3.26  3.24   85.9  22.618152
1121  0.965397   7.13  3.16  0.71  0.279  74.1  3.22  3.15   96.9  24.969993
645   0.919672   6.70  2.39  0.78  0.282  74.1  3.31  3.19  103.3  25.286382
5607  0.990008   5.94  2.32  0.67  0.275  73.3  3.11  3.06  100.5  26.441134
6311  0.996651   9.55  4.67  0.54  0.265  73.1  3.19  2.97  107.2  28.307878
4347  0.926722   6.06  1.80  0.63  0.281  72.3  3.16  3.26  116.7  28.562499
1341  0.909717   8.55  2.89  0.66  0.284  74.6  3.12  3.09  133.7  31.871935

[10 rows x 31 columns]
```

[98]:
```
df[df['retroID'] == 'kersc001']
```

[98]:
```
        retroID   BAOpp  CG  SHO  IPouts     H   ER   HR   BB    SO  ...  \
3761  kersc001  0.2105  25   15    6824  1715  617  173  577  2464  ...

             IP   K/9  BB/9  HR/9  BABIP  LOB%   ERA   FIP   WAR  Pitching
3761  0.420829  9.75  2.28  0.68   0.27  79.4  2.44  2.74  64.5  7.359878

[1 rows x 31 columns]
```

This looks good, but we intuitively see a problem with the Pitching stat. ERA and FIP are part of the average, but a low ERA/FIP is better than a high one. We need to subtract them rather than add. With this change, I'm going to see how the stat looks without taking IP into account.

[132]:
```
df['-ERA'] = 0 - df['ERA']
df['-FIP'] = 0 - df['FIP']
```

[133]:
```
df
```

[133]:
```
        retroID   BAOpp  CG  SHO  IPouts     H   ER  HR   BB   SO  ...  BB/9  \
0     aardd001  0.2574   0    0    1011   296  160  41  183  340  ...  4.89
```

```
1     aased001  0.2508  22    5    3328  1085  468   89   457  641  ...  3.71
2     abadf001  0.2447   0    0     992   309  135   42   116  280  ...  3.16
3     abbog001  0.2786  37    5    3858  1405  627  162   352  484  ...  2.46
4     abboj001  0.2804  31    6    5022  1779  791  154   620  888  ...  3.33
...        ...     ...  ..  ...     ...   ...  ...  ...   ...  ...  ...   ...
8020  zolds101  0.2700  30    5    2788   956  366   54   301  207  ...  2.91
8021  zubeb101  0.2717  23    3    2358   767  374   35   468  383  ...  5.36
8022  zumaj001  0.2286   0    0     629   169   71   18   114  210  ...  4.89
8023  zuveg101  0.2760   9    2    1927   660  253   56   203  223  ...  2.84
8024  zycht001  0.2183   0    0     218    57   22    3    34   80  ...  4.21

      HR/9  BABIP  LOB%   ERA   FIP   WAR  Pitching  -ERA  -FIP
0     1.09  0.285  74.5  4.27  4.45   1.1    -1.847 -4.27 -4.45
1     0.72  0.282  73.4  3.80  3.85  11.7     1.046 -3.80 -3.85
2     1.14  0.281  77.7  3.67  4.24   0.6    -1.777 -3.67 -4.24
3     1.13  0.278  69.3  4.39  4.46  10.2     0.359 -4.39 -4.46
4     0.83  0.295  70.0  4.25  4.25  22.7     3.581 -4.25 -4.25
...    ...    ...   ...   ...   ...   ...       ...   ...   ...
8020  0.52  0.267  70.7  3.54  3.80   9.3     0.503 -3.54 -3.80
8021  0.40  0.283  69.0  4.28  3.96   3.3    -1.207 -4.28 -3.96
8022  0.77  0.267  78.7  3.00  3.94   2.7    -1.002 -3.00 -3.94
8023  0.78  0.270  73.2  3.54  3.93   1.9    -1.372 -3.54 -3.93
8024  0.37  0.293  79.1  2.72  3.22   1.1    -1.145 -2.72 -3.22

[8025 rows x 33 columns]
```

[134]:
```
df['Pitching'] = df[['K%', '-ERA', '-FIP', 'WAR']].mean(axis=1).round(3)
```

[135]:
```
df.sort_values('Pitching').tail(10)
```

[135]:
```
       retroID   BAOpp   CG  SHO  IPouts     H    ER   HR    BB    SO  ...  \
7049  suttd001  0.2376  178   58   15847  4692  1914  472  1343  3574  ...
2824  grovl101  0.2535  298   35   11822  3849  1339  162  1187  2266  ...
6529  seavt001  0.2285  231   61   14348  3971  1521  380  1390  3640  ...
1121  carls001  0.2546  254   55   15652  4672  1864  414  1833  4136  ...
5607  perrg101  0.2540  303   53   16051  4938  1846  399  1379  3534  ...
645   blylb001  0.2487  242   60   14910  4632  1830  430  1322  3701  ...
6311  ryann001  0.2088  222   61   16158  3923  1911  321  2795  5714  ...
3588  johnr005  0.2252  100   37   12406  3346  1513  411  1497  4875  ...
4347  maddg002  0.2553  109   35   15025  4726  1756  353   999  3371  ...
1341  clemr001  0.2308  118   46   14750  4185  1707  363  1580  4672  ...

      BB/9  HR/9  BABIP  LOB%   ERA   FIP    WAR  Pitching  -ERA  -FIP
7049  2.29  0.80  0.261  73.6  3.26  3.24   85.9    19.891 -3.26 -3.24
2824  2.71  0.37  0.284  71.8  3.06  3.36   88.8    20.629 -3.06 -3.36
6529  2.62  0.72  0.259  76.7  2.86  3.04   92.7    21.747 -2.86 -3.04
1121  3.16  0.71  0.279  74.1  3.22  3.15   96.9    22.680 -3.22 -3.15
```

```
5607  2.32  0.67  0.275  73.3  3.11  3.06  100.5    23.623 -3.11 -3.06
645   2.39  0.78  0.282  74.1  3.31  3.19  103.3    24.245 -3.31 -3.19
6311  4.67  0.54  0.265  73.1  3.19  2.97  107.2    25.323 -3.19 -2.97
3588  3.26  0.89  0.291  74.7  3.29  3.19  110.4    26.051 -3.29 -3.19
4347  1.80  0.63  0.281  72.3  3.16  3.26  116.7    27.611 -3.16 -3.26
1341  2.89  0.66  0.284  74.6  3.12  3.09  133.7    31.930 -3.12 -3.09

[10 rows x 33 columns]
```

[136]:
```
df[df['retroID'] == 'kersc001']
```

[136]:
```
       retroID   BAOpp  CG  SHO  IPouts     H   ER   HR   BB    SO  ...  BB/9  \
3761  kersc001  0.2105  25   15    6824  1715  617  173  577  2464  ...  2.28

      HR/9  BABIP  LOB%   ERA   FIP   WAR  Pitching  -ERA  -FIP
3761  0.68   0.27  79.4  2.44  2.74  64.5    14.899 -2.44 -2.74

[1 rows x 33 columns]
```

[137]:
```
df[df['retroID'] == 'johnr005']
```

[137]:
```
       retroID   BAOpp   CG  SHO  IPouts     H    ER   HR    BB    SO  ...  \
3588  johnr005  0.2252  100   37   12406  3346  1513  411  1497  4875  ...

      BB/9  HR/9  BABIP  LOB%   ERA   FIP    WAR  Pitching  -ERA  -FIP
3588  3.26  0.89  0.291  74.7  3.29  3.19  110.4    26.051 -3.29 -3.19

[1 rows x 33 columns]
```

[138]:
```
df[df['retroID'] == 'bumgm001']
```

[138]:
```
       retroID   BAOpp  CG  SHO  IPouts     H   ER   HR   BB    SO  ...  BB/9  \
942  bumgm001  0.2358  15    6    5538  1622  642  192  428  1794  ...  2.09

     HR/9  BABIP  LOB%   ERA   FIP   WAR  Pitching  -ERA  -FIP
942  0.94  0.284  76.4  3.13  3.32  31.3     6.272 -3.13 -3.32

[1 rows x 33 columns]
```

[139]:
```
df[df['retroID'] == 'mahop001']
```

[139]:
```
       retroID   BAOpp  CG  SHO  IPouts    H   ER   HR   BB   SO  ...  BB/9  \
4377  mahop001  0.2751   0    0    2127  738  431  116  392  452  ...  4.98

      HR/9  BABIP  LOB%   ERA   FIP  WAR  Pitching  -ERA  -FIP
4377  1.47  0.284  69.5  5.47  5.62 -3.0    -3.487 -5.47 -5.62
```

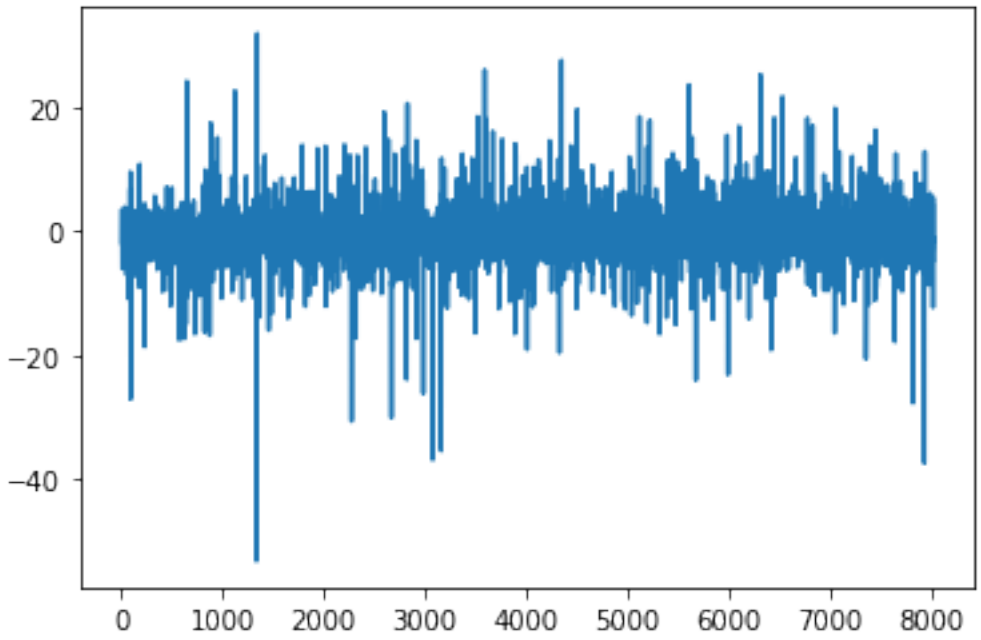
[1 rows x 33 columns]

[140]:
```
df[df['retroID'] == 'coleg001']
```

[140]:
```
       retroID   BAOpp  CG  SHO  IPouts     H   ER   HR   BB    SO  ...  BB/9  \
1383  coleg001  0.2381   2    1    3585  1034  427  115  315  1336  ...  2.37

      HR/9  BABIP  LOB%   ERA   FIP   WAR  Pitching  -ERA  -FIP
1383  0.87  0.303  75.6  3.22  3.06  28.8     5.699 -3.22 -3.06

[1 rows x 33 columns]
```

[141]:
```
plt.plot(df['Pitching'])
```

[141]:
```
[<matplotlib.lines.Line2D at 0x12a1537d0>]
```



[131]:
```
df.sort_values('Pitching').head(10)
```

[131]:
```
       retroID  BAOpp  CG  SHO  IPouts  H  ER  HR  BB  SO  ...   BB/9  HR/9  \
1333  cleaj101   0.83   0    0       1  5   7   0   3   1  ...   81.0   0.0
7932  wurmf101   0.50   0    0       1  1   4   0   5   1  ...  135.0   0.0
3074  heart001   0.75   0    0       1  3   4   0   4   0  ...  108.0   0.0
3159  herne001   0.50   0    0       1  1   3   1   2   0  ...   54.0  27.0
2272  fishf101   0.66   0    0       1  2   4   0   2   1  ...   54.0   0.0
2666  gomec002   0.00   0    0       1  0   3   0   4   0  ...  108.0   0.0
7819  wilst104   0.00   0    0       1  0   3   0   2   0  ...   54.0   0.0
```

```
88    alexm001   0.50   0    0       2  1   5   1   4   0  ...   54.0  13.5
2981  harll101   0.50   0    0       2  2   5   1   4   1  ...   54.0  13.5
5674  pickr001   0.60   0    0       2  3   6   0   4   2  ...   54.0   0.0

      BABIP  LOB%    ERA    FIP  WAR  Pitching   -ERA   -FIP
1333   1.00  12.5  189.0  23.54 -0.1   -53.132 -189.0 -23.54
7932   1.00  33.3  108.0  41.54 -0.1   -37.374 -108.0 -41.54
3074   0.75  28.6  108.0  39.21 -0.1   -36.828 -108.0 -39.21
3159   0.00   0.0   81.0  60.16 -0.3   -35.365  -81.0 -60.16
2272   1.00   0.0  108.0  14.60  0.0   -30.600 -108.0 -14.60
2666   0.00  25.0   81.0  39.16 -0.1   -30.065  -81.0 -39.16
7819   0.00   0.0   81.0  29.57 -0.1   -27.667  -81.0 -29.57
88     0.00   0.0   67.5  40.67 -0.1   -27.068  -67.5 -40.67
2981   0.50  21.7   67.5  37.08 -0.1   -26.139  -67.5 -37.08
5674   1.00  14.3   81.0  15.14  0.0   -23.979  -81.0 -15.14

[10 rows x 33 columns]
```

I'm happy with this. The new Pitching stat somewhat reflects its component parts but doesn't immediately align with WAR. We don't just want to recreate WAR so that's a good thing.

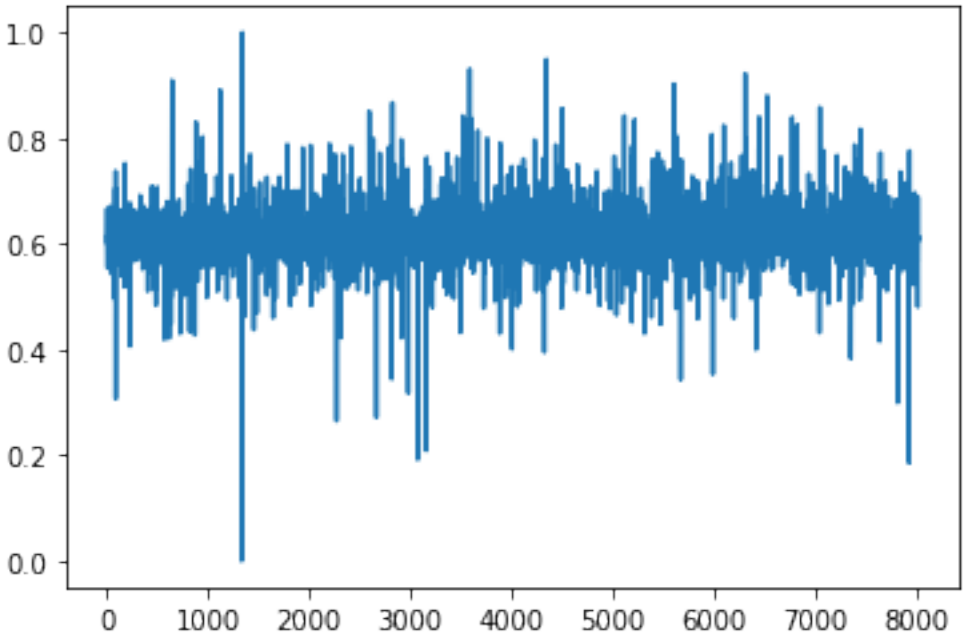
Normalization

[142]:
```
from sklearn.preprocessing import MinMaxScaler
```

[143]:
```
scaler = MinMaxScaler()
```

[144]:
```
plt.plot(scaler.fit_transform(df[['Pitching']]))
```

[144]:
```
[<matplotlib.lines.Line2D at 0x1337d3f50>]
```



[145]:
```
df['Pitching'] = scaler.fit_transform(df[['Pitching']])
```

[147]:
```
df.sort_values('Pitching').head(10)
```

[147]:

	retroID	BAOpp	CG	SHO	IPouts	H	ER	HR	BB	SO	...	BB/9	HR/9	\
1333	cleaj101	0.83	0	0	1	5	7	0	3	1	...	81.0	0.0	
7932	wurmf101	0.50	0	0	1	1	4	0	5	1	...	135.0	0.0	
3074	heart001	0.75	0	0	1	3	4	0	4	0	...	108.0	0.0	
3159	herne001	0.50	0	0	1	1	3	1	2	0	...	54.0	27.0	
2272	fishf101	0.66	0	0	1	2	4	0	2	1	...	54.0	0.0	
2666	gomec002	0.00	0	0	1	0	3	0	4	0	...	108.0	0.0	
7819	wilst104	0.00	0	0	1	0	3	0	2	0	...	54.0	0.0	
88	alexm001	0.50	0	0	2	1	5	1	4	0	...	54.0	13.5	
2981	harll101	0.50	0	0	2	2	5	1	4	1	...	54.0	13.5	
5674	pickr001	0.60	0	0	2	3	6	0	4	2	...	54.0	0.0	

	BABIP	LOB%	ERA	FIP	WAR	Pitching	-ERA	-FIP
1333	1.00	12.5	189.0	23.54	-0.1	0.000000	-189.0	-23.54
7932	1.00	33.3	108.0	41.54	-0.1	0.185253	-108.0	-41.54
3074	0.75	28.6	108.0	39.21	-0.1	0.191672	-108.0	-39.21
3159	0.00	0.0	81.0	60.16	-0.3	0.208871	-81.0	-60.16
2272	1.00	0.0	108.0	14.60	0.0	0.264889	-108.0	-14.60
2666	0.00	25.0	81.0	39.16	-0.1	0.271179	-81.0	-39.16
7819	0.00	0.0	81.0	29.57	-0.1	0.299370	-81.0	-29.57
88	0.00	0.0	67.5	40.67	-0.1	0.306412	-67.5	-40.67
2981	0.50	21.7	67.5	37.08	-0.1	0.317333	-67.5	-37.08

```
5674   1.00  14.3   81.0  15.14  0.0  0.342726  -81.0 -15.14

[10 rows x 33 columns]
```

[148]:
```
df.sort_values('Pitching').tail(10)
```

[148]:
```
       retroID   BAOpp   CG  SHO  IPouts     H    ER   HR    BB    SO  ...  \
7049  suttd001  0.2376  178   58   15847  4692  1914  472  1343  3574  ...
2824  grovl101  0.2535  298   35   11822  3849  1339  162  1187  2266  ...
6529  seavt001  0.2285  231   61   14348  3971  1521  380  1390  3640  ...
1121  carls001  0.2546  254   55   15652  4672  1864  414  1833  4136  ...
5607  perrg101  0.2540  303   53   16051  4938  1846  399  1379  3534  ...
645   blylb001  0.2487  242   60   14910  4632  1830  430  1322  3701  ...
6311  ryann001  0.2088  222   61   16158  3923  1911  321  2795  5714  ...
3588  johnr005  0.2252  100   37   12406  3346  1513  411  1497  4875  ...
4347  maddg002  0.2553  109   35   15025  4726  1756  353   999  3371  ...
1341  clemr001  0.2308  118   46   14750  4185  1707  363  1580  4672  ...

      BB/9  HR/9  BABIP  LOB%   ERA   FIP    WAR  Pitching  -ERA  -FIP
7049  2.29  0.80  0.261  73.6  3.26  3.24   85.9  0.858468 -3.26 -3.24
2824  2.71  0.37  0.284  71.8  3.06  3.36   88.8  0.867144 -3.06 -3.36
6529  2.62  0.72  0.259  76.7  2.86  3.04   92.7  0.880287 -2.86 -3.04
1121  3.16  0.71  0.279  74.1  3.22  3.15   96.9  0.891256 -3.22 -3.15
5607  2.32  0.67  0.275  73.3  3.11  3.06  100.5  0.902342 -3.11 -3.06
645   2.39  0.78  0.282  74.1  3.31  3.19  103.3  0.909654 -3.31 -3.19
6311  4.67  0.54  0.265  73.1  3.19  2.97  107.2  0.922327 -3.19 -2.97
3588  3.26  0.89  0.291  74.7  3.29  3.19  110.4  0.930886 -3.29 -3.19
4347  1.80  0.63  0.281  72.3  3.16  3.26  116.7  0.949225 -3.16 -3.26
1341  2.89  0.66  0.284  74.6  3.12  3.09  133.7  1.000000 -3.12 -3.09

[10 rows x 33 columns]
```

Finally, we should get rid of the -ERA and -FIP columns.

[149]:
```
df = df.drop(columns=['-ERA', '-FIP'])
```

[150]:
```
df
```

[150]:
```
       retroID   BAOpp  CG  SHO  IPouts     H    ER   HR   BB   SO  ...  \
0     aardd001  0.2574   0    0    1011   296   160   41  183  340  ...
1     aased001  0.2508  22    5    3328  1085   468   89  457  641  ...
2     abadf001  0.2447   0    0     992   309   135   42  116  280  ...
3     abbog001  0.2786  37    5    3858  1405   627  162  352  484  ...
4     abboj001  0.2804  31    6    5022  1779   791  154  620  888  ...
...        ...     ...  ..  ...     ...   ...   ...  ...  ...  ...  ...
8020  zolds101  0.2700  30    5    2788   956   366   54  301  207  ...
8021  zubeb101  0.2717  23    3    2358   767   374   35  468  383  ...
```

```
8022  zumaj001  0.2286   0    0     629   169   71   18  114  210  ...
8023  zuveg101  0.2760   9    2    1927   660  253   56  203  223  ...
8024  zycht001  0.2183   0    0     218    57   22    3   34   80  ...

            IP   K/9  BB/9  HR/9  BABIP  LOB%   ERA   FIP   WAR  Pitching
0     0.062360  9.08  4.89  1.09  0.285  74.5  4.27  4.45   1.1  0.602913
1     0.205233  5.20  3.71  0.72  0.282  73.4  3.80  3.85  11.7  0.636924
2     0.061102  7.62  3.16  1.14  0.281  77.7  3.67  4.24   0.6  0.603736
3     0.237967  3.39  2.46  1.13  0.278  69.3  4.39  4.46  10.2  0.628847
4     0.309765  4.77  3.33  0.83  0.295  70.0  4.25  4.25  22.7  0.666725
...        ...   ...   ...   ...    ...   ...   ...   ...   ...       ...
8020  0.171925  2.00  2.91  0.52  0.267  70.7  3.54  3.80   9.3  0.630540
8021  0.145445  4.39  5.36  0.40  0.283  69.0  4.28  3.96   3.3  0.610437
8022  0.038711  9.01  4.89  0.77  0.267  78.7  3.00  3.94   2.7  0.612847
8023  0.118817  3.12  2.84  0.78  0.270  73.2  3.54  3.93   1.9  0.608497
8024  0.013360  9.91  4.21  0.37  0.293  79.1  2.72  3.22   1.1  0.611166

[8025 rows x 31 columns]
```

[ ]:

teams_pre

March 9, 2020

[4]:
```
import math
import numpy as np
import pandas as pd
```

[5]:
```
df = pd.read_csv('../data/lahman/mlb_data/Teams.csv')
```

[6]:
```
df.columns
```

[6]:
```
Index(['yearID', 'lgID', 'teamID', 'franchID', 'divID', 'Rank', 'G', 'Ghome',
       'W', 'L', 'DivWin', 'WCWin', 'LgWin', 'WSWin', 'R', 'AB', 'H', '2B',
       '3B', 'HR', 'BB', 'SO', 'SB', 'CS', 'HBP', 'SF', 'RA', 'ER', 'ERA',
       'CG', 'SHO', 'SV', 'IPouts', 'HA', 'HRA', 'BBA', 'SOA', 'E', 'DP', 'FP',
       'name', 'park', 'attendance', 'BPF', 'PPF', 'teamIDBR',
       'teamIDlahman45', 'teamIDretro'],
      dtype='object')
```

[7]:
```
df.head()
```

[7]:

	yearID	lgID	teamID	franchID	divID	Rank	G	Ghome	W	L	...	DP	\
0	1919	AL	BOS	BOS	NaN	6	138	66	66	71	...	118	
1	1919	NL	BRO	LAD	NaN	5	141	70	69	71	...	84	
2	1919	NL	BSN	ATL	NaN	6	140	68	57	82	...	111	
3	1919	AL	CHA	CHW	NaN	1	140	70	88	52	...	116	
4	1919	NL	CHN	CHC	NaN	3	140	71	75	65	...	87	

	FP	name	park	attendance	BPF	PPF	teamIDBR	\
0	0.975	Boston Red Sox	Fenway Park I	417291	94	94	BOS	
1	0.963	Brooklyn Robins	Ebbets Field	360721	103	103	BRO	
2	0.966	Boston Braves	Braves Field	167401	95	98	BSN	
3	0.969	Chicago White Sox	Comiskey Park	627186	100	99	CHW	
4	0.969	Chicago Cubs	Wrigley Field	424430	100	99	CHC	

	teamIDlahman45	teamIDretro
0	BOS	BOS
1	BRO	BRO
2	BSN	BSN
3	CHA	CHA

```
4              CHN          CHN

[5 rows x 48 columns]
```

[8]:
```
df = df.drop(columns=['teamIDlahman45', 'teamIDBR'])
```

The first step is to ensure we're only using one ID per team. It would be best to just use Retrosheet's values, so our first step is to see where teamID differs from teamIDretro. Once we come up with a way to fix these differences, we'll want to write it as a script that we can use elsewhere - for example, in the batting table where we're using the regular teamID values.

[9]:
```
df[(df['teamID'] != df['teamIDretro'])][['yearID', 'teamID', 'teamIDretro',␣
↪'name']]
```

[9]:

	yearID	teamID	teamIDretro	name
551	1953	ML1	MLN	Milwaukee Braves
568	1954	ML1	MLN	Milwaukee Braves
585	1955	ML1	MLN	Milwaukee Braves
601	1956	ML1	MLN	Milwaukee Braves
617	1957	ML1	MLN	Milwaukee Braves
633	1958	ML1	MLN	Milwaukee Braves
649	1959	ML1	MLN	Milwaukee Braves
665	1960	ML1	MLN	Milwaukee Braves
683	1961	ML1	MLN	Milwaukee Braves
702	1962	ML1	MLN	Milwaukee Braves
722	1963	ML1	MLN	Milwaukee Braves
742	1964	ML1	MLN	Milwaukee Braves
762	1965	ML1	MLN	Milwaukee Braves
867	1970	ML4	MIL	Milwaukee Brewers
891	1971	ML4	MIL	Milwaukee Brewers
915	1972	ML4	MIL	Milwaukee Brewers
939	1973	ML4	MIL	Milwaukee Brewers
963	1974	ML4	MIL	Milwaukee Brewers
987	1975	ML4	MIL	Milwaukee Brewers
1011	1976	ML4	MIL	Milwaukee Brewers
1035	1977	ML4	MIL	Milwaukee Brewers
1061	1978	ML4	MIL	Milwaukee Brewers
1087	1979	ML4	MIL	Milwaukee Brewers
1113	1980	ML4	MIL	Milwaukee Brewers
1139	1981	ML4	MIL	Milwaukee Brewers
1165	1982	ML4	MIL	Milwaukee Brewers
1191	1983	ML4	MIL	Milwaukee Brewers
1217	1984	ML4	MIL	Milwaukee Brewers
1243	1985	ML4	MIL	Milwaukee Brewers
1269	1986	ML4	MIL	Milwaukee Brewers
1295	1987	ML4	MIL	Milwaukee Brewers
1321	1988	ML4	MIL	Milwaukee Brewers

```
1347    1989    ML4         MIL              Milwaukee Brewers
1373    1990    ML4         MIL              Milwaukee Brewers
1399    1991    ML4         MIL              Milwaukee Brewers
1425    1992    ML4         MIL              Milwaukee Brewers
1453    1993    ML4         MIL              Milwaukee Brewers
1481    1994    ML4         MIL              Milwaukee Brewers
1509    1995    ML4         MIL              Milwaukee Brewers
1537    1996    ML4         MIL              Milwaukee Brewers
1565    1997    ML4         MIL              Milwaukee Brewers
1801    2005    LAA         ANA  Los Angeles Angels of Anaheim
1831    2006    LAA         ANA  Los Angeles Angels of Anaheim
1861    2007    LAA         ANA  Los Angeles Angels of Anaheim
1891    2008    LAA         ANA  Los Angeles Angels of Anaheim
1921    2009    LAA         ANA  Los Angeles Angels of Anaheim
1951    2010    LAA         ANA  Los Angeles Angels of Anaheim
1981    2011    LAA         ANA  Los Angeles Angels of Anaheim
2010    2012    LAA         ANA  Los Angeles Angels of Anaheim
2040    2013    LAA         ANA  Los Angeles Angels of Anaheim
2070    2014    LAA         ANA  Los Angeles Angels of Anaheim
2100    2015    LAA         ANA  Los Angeles Angels of Anaheim
2130    2016    LAA         ANA  Los Angeles Angels of Anaheim
2160    2017    LAA         ANA  Los Angeles Angels of Anaheim
2190    2018    LAA         ANA  Los Angeles Angels of Anaheim
```

So clearly we have three teams where the IDs differ. We need to ask a few questions though:

Do they differ on those teams every time? We can't just take that for granted.

[10]:
```
df[df['franchID'] == 'ANA']['teamID'].value_counts()
```

[10]:
```
CAL    32
LAA    18
ANA     8
Name: teamID, dtype: int64
```

[11]:
```
df[(df['teamID'] != df['teamIDretro'])][['teamID', 'teamIDretro', 'name']].
 ↪shape[0]
```

[11]:
```
55
```

[12]:
```
df[(df['teamID'] == 'ML1')].shape[0] + df[(df['teamID'] == 'ML4')].shape[0] +␣
 ↪df[(df['teamID'] == 'LAA')].shape[0]
```

[12]:
```
59
```

Unfortunately we have a disparity of 4, so we need to find out where that is.

[13]:
```
df[(df['teamID'] == 'ML1') & (df['teamID'] == df['teamIDretro'])]
```

[13]:
```
Empty DataFrame
Columns: [yearID, lgID, teamID, franchID, divID, Rank, G, Ghome, W, L, DivWin,
WCWin, LgWin, WSWin, R, AB, H, 2B, 3B, HR, BB, SO, SB, CS, HBP, SF, RA, ER, ERA,
CG, SHO, SV, IPouts, HA, HRA, BBA, SOA, E, DP, FP, name, park, attendance, BPF,
PPF, teamIDretro]
Index: []

[0 rows x 46 columns]
```

[14]:
```
df[(df['teamID'] == 'ML4') & (df['teamID'] == df['teamIDretro'])]
```

[14]:
```
Empty DataFrame
Columns: [yearID, lgID, teamID, franchID, divID, Rank, G, Ghome, W, L, DivWin,
WCWin, LgWin, WSWin, R, AB, H, 2B, 3B, HR, BB, SO, SB, CS, HBP, SF, RA, ER, ERA,
CG, SHO, SV, IPouts, HA, HRA, BBA, SOA, E, DP, FP, name, park, attendance, BPF,
PPF, teamIDretro]
Index: []

[0 rows x 46 columns]
```

[15]:
```
df[(df['teamID'] == 'LAA') & (df['teamID'] == df['teamIDretro'])]
```

[15]:
```
     yearID lgID teamID franchID divID  Rank    G  Ghome   W   L  ...  SOA  \
680    1961   AL    LAA      ANA   NaN     8  162     82  70  91  ...  973
699    1962   AL    LAA      ANA   NaN     3  162     81  86  76  ...  858
719    1963   AL    LAA      ANA   NaN     9  161     81  70  91  ...  889
739    1964   AL    LAA      ANA   NaN     5  162     81  82  80  ...  965

       E   DP     FP                name                park  attendance  BPF  \
680  192  154  0.969  Los Angeles Angels  Wrigley Field (LA)      603510  111
699  175  153  0.972  Los Angeles Angels      Dodger Stadium     1144063   97
719  163  155  0.974  Los Angeles Angels      Dodger Stadium      821015   94
739  138  168  0.978  Los Angeles Angels      Dodger Stadium      760439   90

     PPF  teamIDretro
680  112          LAA
699   97          LAA
719   94          LAA
739   90          LAA

[4 rows x 46 columns]
```

[16]:
```
df[(df['teamID'] == 'LAA') & (df['teamID'] != df['teamIDretro'])]
```

[16]:
```
      yearID lgID teamID franchID divID  Rank    G  Ghome    W   L  ...   SOA  \
1801    2005   AL    LAA      ANA     W     1  162     81   95  67  ...  1126
1831    2006   AL    LAA      ANA     W     2  162     81   89  73  ...  1164
```

```
1861    2007   AL    LAA      ANA     W     1  162     81   94  68  ...  1156
1891    2008   AL    LAA      ANA     W     1  162     81  100  62  ...  1106
1921    2009   AL    LAA      ANA     W     1  162     81   97  65  ...  1062
1951    2010   AL    LAA      ANA     W     3  162     81   80  82  ...  1130
1981    2011   AL    LAA      ANA     W     2  162     81   86  76  ...  1058
2010    2012   AL    LAA      ANA     W     3  162     81   89  73  ...  1157
2040    2013   AL    LAA      ANA     W     3  162     81   78  84  ...  1200
2070    2014   AL    LAA      ANA     W     1  162     81   98  64  ...  1342
2100    2015   AL    LAA      ANA     W     3  162     81   85  77  ...  1221
2130    2016   AL    LAA      ANA     W     4  162     81   74  88  ...  1136
2160    2017   AL    LAA      ANA     W     2  162     81   80  82  ...  1312
2190    2018   AL    LAA      ANA     W     4  162     81   80  82  ...  1386

        E   DP     FP                          name  \
1801   87  139  0.986  Los Angeles Angels of Anaheim
1831  124  154  0.979  Los Angeles Angels of Anaheim
1861  101  154  0.983  Los Angeles Angels of Anaheim
1891   91  159  0.985  Los Angeles Angels of Anaheim
1921   85  174  0.986  Los Angeles Angels of Anaheim
1951  113  116  0.981  Los Angeles Angels of Anaheim
1981   93  157  0.985  Los Angeles Angels of Anaheim
2010   98  141  0.984  Los Angeles Angels of Anaheim
2040  112  135  0.981  Los Angeles Angels of Anaheim
2070   83  127  0.986  Los Angeles Angels of Anaheim
2100   93  108  0.984  Los Angeles Angels of Anaheim
2130   97  148  0.983  Los Angeles Angels of Anaheim
2160   80  135  0.986  Los Angeles Angels of Anaheim
2190   76  173  0.987  Los Angeles Angels of Anaheim

                          park  attendance  BPF  PPF  teamIDretro
1801             Angel Stadium     3404686   98   97          ANA
1831             Angel Stadium     3406790  100  100          ANA
1861             Angel Stadium     3365632  101  100          ANA
1891             Angel Stadium     3336747  103  102          ANA
1921             Angel Stadium     3240386   99   98          ANA
1951             Angel Stadium     3250816   98   98          ANA
1981             Angel Stadium     3166321   93   93          ANA
2010  Angel Stadium of Anaheim     3061770   92   92          ANA
2040  Angel Stadium of Anaheim     3019505   94   94          ANA
2070  Angel Stadium of Anaheim     3095935   96   95          ANA
2100  Angel Stadium of Anaheim     3012765   94   95          ANA
2130  Angel Stadium of Anaheim     3016142   95   95          ANA
2160  Angel Stadium of Anaheim     3019585   96   96          ANA
2190  Angel Stadium of Anaheim     3020216   97   97          ANA

[14 rows x 46 columns]
```

[17]:
```
df['franchID'].unique()
```

[17]:
```
array(['BOS', 'LAD', 'ATL', 'CHW', 'CHC', 'CIN', 'CLE', 'DET', 'SFG',
       'NYY', 'OAK', 'PHI', 'PIT', 'BAL', 'STL', 'MIN', 'ANA', 'TEX',
       'HOU', 'NYM', 'KCR', 'WSN', 'SDP', 'MIL', 'SEA', 'TOR', 'COL',
       'FLA', 'ARI', 'TBD'], dtype=object)
```

[18]:
```
df[(df['franchID'].isnull())]
```

[18]:
```
Empty DataFrame
Columns: [yearID, lgID, teamID, franchID, divID, Rank, G, Ghome, W, L, DivWin,
WCWin, LgWin, WSWin, R, AB, H, 2B, 3B, HR, BB, SO, SB, CS, HBP, SF, RA, ER, ERA,
CG, SHO, SV, IPouts, HA, HRA, BBA, SOA, E, DP, FP, name, park, attendance, BPF,
PPF, teamIDretro]
Index: []

[0 rows x 46 columns]
```

[19]:
```
df['franchID'].nunique()
```

[19]:
```
30
```

It looks like it will be easiest to just use the franchise ID - they stay consistent throughout and there are only ever 30 max. We'll need a way to map to these values from an external script so we can use it in other files.

Building Tables and Tensors

Aggregating preprocessed data and compiling it into tables that are ready to be read into the models as tensors

build_tables

March 28, 2020

[88]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Building the Tables

We've done the major preprocessing in other scripts, and now it's time to get our final tables together for fielders, catchers and pitchers with all appropriate stats.

[89]:
```
df_bat = pd.read_csv('../core/output/batting_pre.csv')
df_field = pd.read_csv('../core/output/fielding_pre.csv')
df_catch = pd.read_csv('../core/output/catching_pre.csv')
df_pitch = pd.read_csv('../core/output/pitching_pre.csv')
df_meta = pd.read_csv('../core/output/metadata.csv')
```

[90]:
```
df_meta.head()
```

[90]:

	retroID	POS	birthYear	bats	throws	weight	height	debutYear	finalYear
0	aardd001	P	1981.0	R	R	215.0	75.0	2004	2015
1	aaroh101	OF	1934.0	R	R	180.0	72.0	1954	1976
2	aarot101	1B	1939.0	R	R	190.0	75.0	1962	1971
3	aased001	P	1954.0	R	R	190.0	75.0	1977	1990
4	abada001	1B	1972.0	L	L	184.0	73.0	2001	2006

Making Metadata Usable

We are interested in all of these fields, so we want to convert POS, bats and throws to numbers and use dummy variables. Note that we won't be using birthYear as-is, but rather subtracting it from current year to get a player's age for a season. This won't matter for the player career stats tensor so we can drop it here.

[91]:
```
df_meta.drop(columns=['birthYear'], inplace=True)
```

[92]:
```
df_meta_pos = pd.get_dummies(df_meta['POS'], prefix='pos')
df_meta_bats = pd.get_dummies(df_meta['bats'], drop_first=True, prefix='bats')
df_meta_throws = pd.get_dummies(df_meta['throws'], prefix='throws')
```

[93]:
```
dropped_meta_cols = ['POS', 'bats', 'throws']
df_meta.drop(columns=dropped_meta_cols, inplace=True)
```

```
df_meta.head()
```

[93]:

	retroID	weight	height	debutYear	finalYear
0	aardd001	215.0	75.0	2004	2015
1	aaroh101	180.0	72.0	1954	1976
2	aarot101	190.0	75.0	1962	1971
3	aased001	190.0	75.0	1977	1990
4	abada001	184.0	73.0	2001	2006

[94]:
```
df_meta = df_meta.join([df_meta_pos, df_meta_bats, df_meta_throws])
df_meta
```

[94]:

	retroID	weight	height	debutYear	finalYear	pos_1B	pos_2B	pos_3B	pos_C	pos_OF	pos_P	pos_SS	bats_L	bats_R	throws_L	throws_R	throws_S
0	aardd001	215.0	75.0	2004	2015	0	0	0	0	0	1	0	0	1	0	1	0
1	aaroh101	180.0	72.0	1954	1976	0	0	0	0	1	0	0	0	1	0	1	0
2	aarot101	190.0	75.0	1962	1971	1	0	0	0	0	0	0	0	1	0	1	0
3	aased001	190.0	75.0	1977	1990	0	0	0	0	0	1	0	0	1	0	1	0
4	abada001	184.0	73.0	2001	2006	1	0	0	0	0	0	0	1	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
15026	zupcb001	220.0	76.0	1991	1994	0	0	0	0	1	0	0	0	1	0	1	0
15027	zupof101	182.0	71.0	1957	1961	0	0	0	1	0	0	0	1	0	1	0	0
15028	zuveg101	195.0	76.0	1951	1959	0	0	0	0	0	1	0	0	1	0	1	0
15029	zuvep001	173.0	72.0	1982	1991	0	0	0	0	0	0	1	0	1	0	1	0
15030	zycht001	190.0	75.0	2015	2017	0	0	0	0	0	1	0	0	1	0	1	0

```
15028         0
15029         0
15030         0

[15031 rows x 17 columns]
```

I didn't drop_first on the 'throws_' columns because I want to get rid of 'throws_S' instead of 'throws_L'

[95]:
```
df_meta.drop(columns=['throws_S'], inplace=True)
```

We want to use weight and height but we can normalize them

[96]:
```
from sklearn.preprocessing import MinMaxScaler
```

[97]:
```
scaler = MinMaxScaler()
```

[98]:
```
df_meta[['weight', 'height']] = scaler.fit_transform(df_meta[['weight',␣
↪'height']])
df_meta
```

[98]:

	retroID	weight	height	debutYear	finalYear	pos_1B	pos_2B	pos_3B	pos_C	pos_OF	pos_P	pos_SS	bats_L	bats_R	throws_L
0	aardd001	0.569672	0.60	2004	2015	0	0	0	0	0	1	0	0	1	0
1	aaroh101	0.426230	0.45	1954	1976	0	0	0	0	1	0	0	0	1	0
2	aarot101	0.467213	0.60	1962	1971	1	0	0	0	0	0	0	0	1	0
3	aased001	0.467213	0.60	1977	1990	0	0	0	0	0	1	0	0	1	0
4	abada001	0.442623	0.50	2001	2006	1	0	0	0	0	0	0	1	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
15026	zupcb001	0.590164	0.65	1991	1994	0	0	0	0	1	0	0	0	1	0
15027	zupof101	0.434426	0.40	1957	1961	0	0	0	1	0	0	0	1	0	0
15028	zuveg101	0.487705	0.65	1951	1959	0	0	0	0	0	1	0	0	1	0
15029	zuvep001	0.397541	0.45	1982	1991	0	0	0	0	0	0	1	0	1	0
15030	zycht001	0.467213	0.60	2015	2017	0	0	0	0	0	1	0	0	1	0

```
       throws_R
0             1
1             1
2             1
3             1
4             0
...         ...
15026         1
15027         1
15028         1
15029         1
15030         1

[15031 rows x 16 columns]
```

The metadata is now ready to go into the final tensor.

Combining Batting Data

[99]:
```
df_bat
```

[99]:
```
        retroID     G     AB     R     H   2B  3B   HR   RBI   SB    CS    BB  \
0      aardd001   331      4     0     0    0   0    0     0    0   0.0     0
1      aaroh101  3298  12364  2174  3771  624  98  755  2297  240  73.0  1402
2      aarot101   437    944   102   216   42   6   13    94    9   8.0    86
3      aased001   448      5     0     0    0   0    0     0    0   0.0     0
4      abada001    15     21     1     2    0   0    0     0    0   1.0     4
...         ...   ...    ...   ...   ...  ...  ..  ...   ...  ...   ...   ...
15187  zupcb001   319    795    99   199   47   4    7    80    7   5.0    57
15188  zupof101    16     18     3     3    1   0    0     0    0   0.0     2
15189  zuveg101   266    142     5    21    2   1    0     7    0   1.0     9
15190  zuvep001   209    491    41   109   17   2    2    20    2   0.0    34
15191  zycht001    70      0     0     0    0   0    0     0    0   0.0     0

         SO  IBB  HBP  SH   SF  GIDP  NL
0         2    0    0   1    0     0   1
1      1383  293   32  21  121   328   1
2       145    3    0   9    6    36   1
3         3    0    0   0    0     0   1
4         5    0    0   0    0     1   1
...     ...  ...  ...  ..  ...   ...  ..
15187   137    3    6  20    8    15   0
15188     6    0    0   0    0     0   0
15189    39    0    0  16    0     3   1
15190    50    1    2  18    0     8   1
15191     0    0    0   0    0     0   0

[15192 rows x 19 columns]
```

[100]:
```
df = pd.merge(df_meta, df_bat, how='inner', on=['retroID'])
```

[187]:
```
df.shape
```

[187]:
```
(15031, 34)
```

[101]:
```
df.head(10)
```

[101]:

	retroID	weight	height	debutYear	finalYear	pos_1B	pos_2B	pos_3B	\
0	aardd001	0.569672	0.60	2004	2015	0	0	0	
1	aaroh101	0.426230	0.45	1954	1976	0	0	0	
2	aarot101	0.467213	0.60	1962	1971	1	0	0	
3	aased001	0.467213	0.60	1977	1990	0	0	0	
4	abada001	0.442623	0.50	2001	2006	1	0	0	
5	abadf001	0.590164	0.50	2010	2017	0	0	0	
6	abbog001	0.508197	0.75	1973	1984	0	0	0	
7	abboj001	0.508197	0.60	1989	1999	0	0	0	
8	abboj002	0.467213	0.55	1997	2001	0	0	0	
9	abbok001	0.508197	0.65	1991	1996	0	0	0	

	pos_C	pos_OF	...	SB	CS	BB	SO	IBB	HBP	SH	SF	GIDP	NL
0	0	0	...	0	0.0	0	2	0	0	1	0	0	1
1	0	1	...	240	73.0	1402	1383	293	32	21	121	328	1
2	0	0	...	9	8.0	86	145	3	0	9	6	36	1
3	0	0	...	0	0.0	0	3	0	0	0	0	0	1
4	0	0	...	0	1.0	4	5	0	0	0	0	1	1
5	0	0	...	0	0.0	0	5	0	0	0	0	1	1
6	0	0	...	0	0.0	0	0	0	0	0	0	0	0
7	0	0	...	0	0.0	0	10	0	0	3	0	0	1
8	0	1	...	6	5.0	38	91	2	3	5	7	12	1
9	0	0	...	0	0.0	1	19	0	0	6	0	0	1

[10 rows x 34 columns]

We noticed that df_bat and df_meta don't have the same number of rows, so we want to find out what's going on there.

[102]:
```
df_bat[~df_bat['retroID'].isin(df_meta['retroID'])]
```

[102]:

	retroID	G	AB	R	H	2B	3B	HR	RBI	SB	CS	BB	SO	IBB	HBP	SH	\
121	albeb101	6	18	1	5	1	0	0	0	0	0.0	0	2	0	0	0	
358	aragj101	1	0	0	0	0	0	0	0	0	0.0	0	0	0	0	0	
445	atkil101	1	1	1	0	0	0	0	0	0	0.0	0	0	0	0	0	
611	banij001	1	1	0	1	0	0	0	0	0	0.0	0	0	0	0	0	
633	barbr101	1	1	0	0	0	0	0	0	0	0.0	0	0	0	0	0	
...	...	..	..	..	..	..	..	..	..	...	..	...	..	..	...	...	..	
14543	westj101	1	1	0	0	0	0	0	0	0	0.0	0	1	0	0	0	

```
14730  willh101  10   9  0  2   0   0   0    0   0  0.0   1   4    0    0   0
14811  wilsi101   1   1  0  0   0   0   0    0   0  0.0   0   0    0    0   0
15009  wrigr002   1   3  0  0   0   0   0    0   0  0.0   0   1    0    0   0
15050  yeabb101   3   0  0  0   0   0   0    0   0  0.0   1   0    0    0   0

       SF  GIDP  NL
121     0     0   0
358     0     0   1
445     0     0   0
611     0     0   1
633     0     0   0
...    ..   ...  ..
14543   0     0   1
14730   0     0   1
14811   0     0   0
15009   0     1   0
15050   0     0   1

[161 rows x 19 columns]
```

[103]:
```
df_bat[~df_bat['retroID'].isin(df_meta['retroID'])]['G'].max()
```

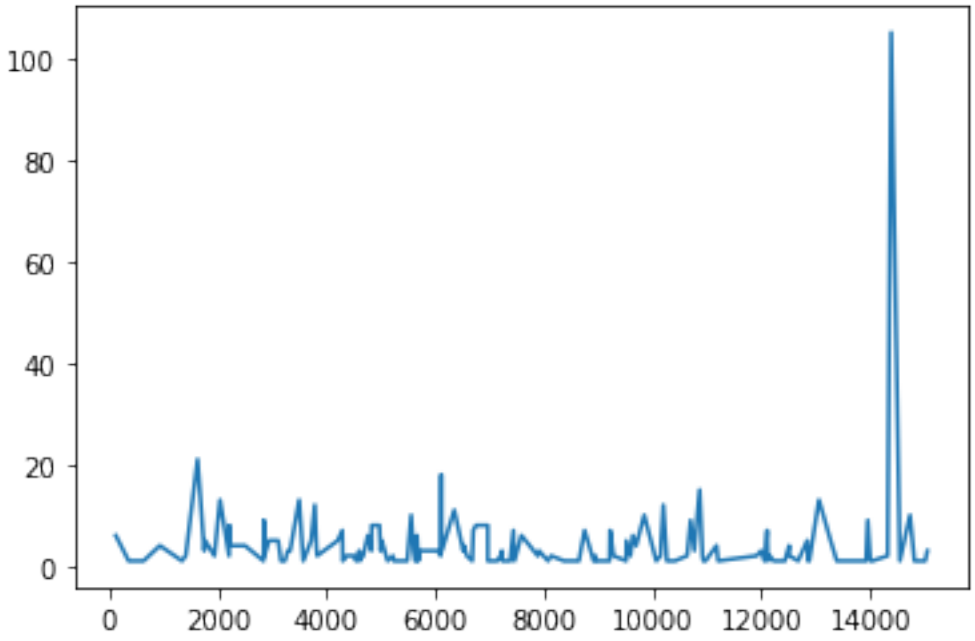
[103]:
```
105
```

[104]:
```
df_bat[~df_bat['retroID'].isin(df_meta['retroID'])]['G'].mean()
```

[104]:
```
4.105590062111801
```

[105]:
```
plt.plot(df_bat[~df_bat['retroID'].isin(df_meta['retroID'])]['G'])
```

[105]:
```
[<matplotlib.lines.Line2D at 0x11ffb5bd0>]
```



For the most part, we're talking about players who have played under 20 total games. We can easily drop these data points and not really affect the overall result.

Combining the Tensors

Catchers

[106]:
```
df_catch.shape[0] + df_field.shape[0] + df_pitch.shape[0]
```

[106]: 23591

[107]:
```
df_catch
```

[107]:

	retroID	GS	InnOuts	PO	A	E	DP	PB	WP	SB	CS	ZR
0	adamb105	1	27	6	0	0	0	0	0	1	0	0
1	adamb106	0	0	249	90	12	15	7	0	0	0	0
2	adamd101	3	78	9	2	0	0	1	0	0	0	0
3	adled101	65	1840	453	26	4	2	8	19	37	16	0
4	afent001	20	613	123	5	1	3	6	0	17	3	0
...	...	...	...	...	...	..	..	..	..	...	..	..
1524	zimmd101	27	744	150	18	6	1	5	12	10	10	3
1525	zimmj101	298	8560	2131	150	21	26	19	84	110	80	4
1526	zinta001	0	3	2	0	0	0	0	0	0	0	0
1527	zunim001	535	14489	4356	264	21	22	39	0	248	98	0
1528	zupof101	1	114	31	1	2	0	1	1	2	1	0

[1529 rows x 12 columns]

[108]:
```
np.intersect1d(df_catch.columns, df.columns)
```

[108]:
```
array(['CS', 'SB', 'retroID'], dtype=object)
```

The 'caught stealing' and 'stolen bases' stats appear both offensively and defensively (CS/SB against) for catchers. We need to keep them separate when merging the metadata and we can do so by just adding a prefix to the defensive stats.

[109]:
```
df_catch.rename(columns={'CS': 'CS_A', 'SB': 'SB_A'}, inplace=True)
```

[110]:
```
catchers = pd.merge(df_catch, df, how='inner', on=['retroID'])
```

[111]:
```
catchers
```

[111]:

	retroID	GS	InnOuts	PO	A	E	DP	PB	WP	SB_A	...	SB	CS	\
0	adamb105	1	27	6	0	0	0	0	0	1	...	0	0.0	
1	adamb106	0	0	249	90	12	15	7	0	0	...	4	2.0	
2	adamd101	3	78	9	2	0	0	1	0	0	...	0	0.0	
3	adled101	65	1840	453	26	4	2	8	19	37	...	0	0.0	
4	afent001	20	613	123	5	1	3	6	0	17	...	0	0.0	
...	...	...	...	...	...	..	..	..	..	...	...	..	...	
1524	zimmd101	27	744	150	18	6	1	5	12	10	...	45	25.0	
1525	zimmj101	298	8560	2131	150	21	26	19	84	110	...	1	2.0	
1526	zinta001	0	3	2	0	0	0	0	0	0	...	0	0.0	
1527	zunim001	535	14489	4356	264	21	22	39	0	248	...	2	4.0	
1528	zupof101	1	114	31	1	2	0	1	1	2	...	0	0.0	

	BB	SO	IBB	HBP	SH	SF	GIDP	NL
0	0	5	0	0	0	0	0	0
1	6	27	0	0	3	0	0	1
2	1	3	0	0	0	0	1	0
3	18	80	5	2	2	1	9	1
4	5	32	0	0	1	1	1	1
...	...	...	...	...	..	..	...	..
1524	246	678	27	13	37	14	99	1
1525	78	154	12	11	31	4	38	1
1526	5	34	0	0	0	1	0	1
1527	138	714	1	45	8	11	38	0
1528	2	6	0	0	0	0	0	0

[1529 rows x 45 columns]

[112]:
```
catchers.columns
```

[112]:
```
Index(['retroID', 'GS', 'InnOuts', 'PO', 'A', 'E', 'DP', 'PB', 'WP', 'SB_A',
       'CS_A', 'ZR', 'weight', 'height', 'debutYear', 'finalYear', 'pos_1B',
       'pos_2B', 'pos_3B', 'pos_C', 'pos_OF', 'pos_P', 'pos_SS', 'bats_L',
```

```
       'bats_R', 'throws_L', 'throws_R', 'G', 'AB', 'R', 'H', '2B', '3B', 'HR',
       'RBI', 'SB', 'CS', 'BB', 'SO', 'IBB', 'HBP', 'SH', 'SF', 'GIDP', 'NL'],
      dtype='object')
```

There's no reason to waste columns on position for the catchers.

[113]:
```
catchers.drop(columns=['pos_1B', 'pos_2B', 'pos_3B', 'pos_C',
       'pos_OF', 'pos_P', 'pos_SS'], inplace=True)
```

[114]:
```
catchers.columns
```

[114]:
```
Index(['retroID', 'GS', 'InnOuts', 'PO', 'A', 'E', 'DP', 'PB', 'WP', 'SB_A',
       'CS_A', 'ZR', 'weight', 'height', 'debutYear', 'finalYear', 'bats_L',
       'bats_R', 'throws_L', 'throws_R', 'G', 'AB', 'R', 'H', '2B', '3B', 'HR',
       'RBI', 'SB', 'CS', 'BB', 'SO', 'IBB', 'HBP', 'SH', 'SF', 'GIDP', 'NL'],
      dtype='object')
```

Pitchers

[147]:
```
np.intersect1d(df_pitch.columns, df.columns)
```

[147]:
```
array(['G', 'retroID'], dtype=object)
```

We have quite a few common columns for pitching data and metadata. We'll do what we did for catching and just add 'A' to the end (for 'against'). Since there are quite a few, we'll define a conversion dictionary ahead of time. Before we do that, we see that we can drop the 'G' (games) column as it should be the same between the tables. We can also drop the position information.

[148]:
```
batting_data_for_pitchers = df.drop(columns=['G', 'pos_1B', 'pos_2B',
                     'pos_3B', 'pos_C', 'pos_OF', 'pos_P', 'pos_SS'])
```

[149]:
```
pitching_data_conversion_dict = {
    'BB': 'BB_A',
    'GIDP': 'GIDP_A',
    'H': 'H_A',
    'HBP': 'HBP_A',
    'HR': 'HR_A',
    'IBB': 'IBB_A',
    'R': 'R_A',
    'SF': 'SF_A',
    'SH': 'SH_A',
    'SO': 'SO_A'
}
```

[150]:
```
df_pitch.rename(columns=pitching_data_conversion_dict, inplace=True)
```

[151]:
```
pitchers = pd.merge(df_pitch, batting_data_for_pitchers, how='inner',␣
 ↪on=['retroID'])
```

[152]:
```
pitchers
```

[152]:

	retroID	BAOpp	ERA	W	L	G	GS	CG	SHO	SV	...	SB	CS	\
0	aardd001	0.2574	5.1944	16	18	331	0	0	0	69	...	0	0.0	
1	aased001	0.2508	3.4931	66	60	448	91	22	5	82	...	0	0.0	
2	abadf001	0.2501	4.0733	8	27	363	6	0	0	2	...	0	0.0	
3	abbog001	0.2786	4.3317	62	83	248	206	37	5	0	...	0	0.0	
4	abboj001	0.2804	4.4964	87	108	263	254	31	6	0	...	0	0.0	
...	...	...	...	..	...	...	...	..	...	..	...	..	...	
7830	zolds101	0.2700	3.6890	43	53	250	93	30	5	8	...	1	0.0	
7831	zubeb101	0.2717	5.3617	43	42	224	65	23	3	6	...	0	0.0	
7832	zumaj001	0.2286	3.4420	13	12	171	0	0	0	5	...	0	0.0	
7833	zuveg101	0.2760	4.1280	32	36	265	31	9	2	40	...	0	1.0	
7834	zycht001	0.2183	2.8000	7	3	70	1	0	0	1	...	0	0.0	

	BB	SO	IBB	HBP	SH	SF	GIDP	NL
0	0	2	0	0	1	0	0	1
1	0	3	0	0	0	0	0	1
2	0	5	0	0	0	0	1	1
3	0	0	0	0	0	0	0	0
4	0	10	0	0	3	0	0	1
...	..	..	...	...	..	..	...	..
7830	10	52	0	1	9	0	4	0
7831	10	66	0	0	20	0	8	0
7832	0	0	0	0	0	0	0	0
7833	9	39	0	0	16	0	3	1
7834	0	0	0	0	0	0	0	0

[7835 rows x 51 columns]

Fielders

[32]:
```
np.intersect1d(df_field.columns, df.columns)
```

[32]:
```
array(['retroID'], dtype=object)
```

We don't need to worry about common columns between the general fielding stats and metadata.

[136]:
```
fielders = pd.merge(df_field, df, how='inner', on=['retroID'])
```

[137]:
```
fielders = fielders[~fielders['retroID'].isin(pitchers['retroID'])]
```

[138]:
```
fielders
```

[138]:

	retroID	GS	InnOuts	PO	A	E	DP	weight	height	\
1	aaroh101	2977	78414	7436	429	144	218	0.426230	0.45	
2	aarot101	206	6472	1317	113	22	124	0.467213	0.60	
4	abada001	4	138	37	1	1	3	0.442623	0.50	

```
8       abboj002   140     3688   299     2    8    0  0.467213    0.55
10      abbok002   504    13474   938  1262   79  275  0.426230    0.40
...          ...   ...      ...   ...   ...  ...  ...       ...     ...
14218   zoske001     8      404    16    42    2    8  0.405738    0.45
14220   zubej001    26      702   167    12    2   11  0.467213    0.50
14221   zulej001    36     1019   296    15    5   20  0.631148    0.75
14223   zupcb001   198     5842   483    22   12    5  0.590164    0.65
14225   zuvep001   136     3844   267   415   23   84  0.397541    0.45

        debutYear  ...   SB    CS    BB    SO  IBB  HBP  SH   SF  GIDP  NL
1            1954  ...  240  73.0  1402  1383  293   32  21  121   328   1
2            1962  ...    9   8.0    86   145    3    0   9    6    36   1
4            2001  ...    0   1.0     4     5    0    0   0    0     1   1
8            1997  ...    6   5.0    38    91    2    3   5    7    12   1
10           1993  ...   22  11.0   133   571   11   17  21   12    37   1
...           ...  ...  ...   ...   ...   ...  ...  ...  ..  ...   ...  ..
14218        1991  ...    0   0.0     1    13    0    0   1    1     1   1
14220        1996  ...    1   0.0    12    20    1    1   1    1     4   1
14221        2000  ...    0   2.0    10    51    1    6   0    1     5   1
14223        1991  ...    7   5.0    57   137    3    6  20    8    15   0
14225        1982  ...    2   0.0    34    50    1    2  18    0     8   1

[6392 rows x 40 columns]
```

[140]:
```
catchers
```

[140]:
```
        retroID   GS  InnOuts    PO    A   E  DP  PB  WP  SB_A  ...  SB    CS  \
0     adamb105    1       27     6    0   0   0   0   0     1  ...   0   0.0
1     adamb106    0        0   249   90  12  15   7   0     0  ...   4   2.0
2     adamd101    3       78     9    2   0   0   1   0     0  ...   0   0.0
3     adled101   65     1840   453   26   4   2   8  19    37  ...   0   0.0
4     afent001   20      613   123    5   1   3   6   0    17  ...   0   0.0
...        ...  ...      ...   ...  ...  ..  ..  ..  ..   ...  ...  ..   ...
1524  zimmd101   27      744   150   18   6   1   5  12    10  ...  45  25.0
1525  zimmj101  298     8560  2131  150  21  26  19  84   110  ...   1   2.0
1526  zinta001    0        3     2    0   0   0   0   0     0  ...   0   0.0
1527  zunim001  535    14489  4356  264  21  22  39   0   248  ...   2   4.0
1528  zupof101    1      114    31    1   2   0   1   1     2  ...   0   0.0

        BB   SO  IBB  HBP  SH  SF  GIDP  NL
0        0    5    0    0   0   0     0   0
1        6   27    0    0   3   0     0   1
2        1    3    0    0   0   0     1   0
3       18   80    5    2   2   1     9   1
4        5   32    0    0   1   1     1   1
...    ...  ...  ...  ...  ..  ..   ...  ..
1524   246  678   27   13  37  14    99   1
```

```
1525   78  154   12   11  31   4    38   1
1526    5   34    0    0   0   1     0   1
1527  138  714    1   45   8  11    38   0
1528    2    6    0    0   0   0     0   0

[1529 rows x 38 columns]
```

[169]:
```
fielders[fielders['retroID'].isin(catchers['retroID'])]
```

[169]:
```
        retroID   GS  InnOuts     PO     A    E    DP    weight  height  \
54     adamb105    2       54     20     1    0     1  0.508197    0.55
55     adamb106    0        0      0     0    0     0  0.446721    0.50
108    ainse101    0        0     11     0    0     0  0.426230    0.40
144    alexg101   22      500     83     6    3     4  0.487705    0.55
151    alfaj002    1       31      8     2    0     1  0.610656    0.55
...         ...  ...      ...    ...   ...  ...   ...       ...     ...
14111  yorkr101    0        0  11425  1030  136  1077  0.545082    0.50
14139  younj001  843    23486   1679   756  115   113  0.426230    0.45
14183  zaung001    0       33      3     2    0     1  0.385246    0.35
14199  zimmd101  813    21993   1491  2204  150   417  0.364754    0.30
14210  zinta001    5      198     65     5    1     4  0.508197    0.55

       debutYear  ...  SB    CS   BB   SO  IBB  HBP  SH  SF  GIDP  NL
54          1977  ...   0   0.0    0    5    0    0   0   0     0   0
55          1910  ...   4   2.0    6   27    0    0   3   0     0   1
108         1910  ...  20  11.5  125  122    0    3  44   0     0   1
144         1975  ...   8  12.0  154  381   12    5   4  19    34   1
151         2016  ...   3   0.0   22  179    8   18   0   1     4   1
...          ...  ...  ..   ...  ...  ...  ...  ...  ..  ..   ...  ..
14111       1934  ...  38  26.0  792  867    0   12  25   0   155   0
14139       1976  ...  60  55.0  332  589   30   36  18  33    80   1
14183       1995  ...  23  19.0  479  544   30   29  14  31    87   1
14199       1954  ...  45  25.0  246  678   27   13  37  14    99   1
14210       2002  ...   0   0.0    5   34    0    0   0   1     0   1

[655 rows x 40 columns]
```

We have some catchers that are also in the fielders table.

[168]:
```
fielders[(fielders['retroID'].isin(catchers['retroID']) & fielders['pos_C'] ==␣
↪1)]
```

[168]:
```
        retroID   GS  InnOuts    PO   A   E   DP    weight  height  debutYear  \
108    ainse101    0        0    11   0   0    0  0.426230    0.40       1910
144    alexg101   22      500    83   6   3    4  0.487705    0.55       1975
151    alfaj002    1       31     8   2   0    1  0.610656    0.55       2016
156    allaa001    0       33    15   0   0    2  0.590164    0.70       1986
```

```
169    alleg001    1       54     4   4   0    2  0.446721    0.40       1979
...         ...  ...      ...   ...  ..  ..  ...       ...     ...        ...
13914  wingi101    0        0     7   2   1    0  0.344262    0.35       1911
13956  wockj001  249     6120  1429  90  17  133  0.467213    0.45       1974
14066  wronr001    0        6     2   0   0    0  0.446721    0.50       1988
14074  wyneb001    0        6     0   0   0    0  0.467213    0.50       1976
14183  zaung001    0       33     3   2   0    1  0.385246    0.35       1995

       ...  SB    CS   BB   SO  IBB  HBP  SH  SF  GIDP  NL
108    ...  20  11.5  125  122    0    3  44   0     0   1
144    ...   8  12.0  154  381   12    5   4  19    34   1
151    ...   3   0.0   22  179    8   18   0   1     4   1
156    ...  23  18.0   87  223    4    9  35  16    27   1
169    ...   3   7.0  130  192    3    5  15  11    35   0
...    ...  ..   ...  ...  ...  ...  ...  ..  ..   ...  ..
13914  ...  17  16.0  121   84    0    4  36   0     0   1
13956  ...   5  11.0  277  278   14    7   5  12    52   1
14066  ...   1   0.0    5   41    2    1   2   2     3   1
14074  ...  10  13.0  626  428   41   17  58  36   119   0
14183  ...  23  19.0  479  544   30   29  14  31    87   1

[430 rows x 40 columns]
```

[170]:
```
to_inspect = fielders[(fielders['retroID'].isin(catchers['retroID']) &␣
 ↪fielders['pos_C'] == 1)]['retroID']
```

[171]:
```
catchers[catchers['retroID'].isin(to_inspect)]
```

[171]:
```
       retroID    GS  InnOuts    PO    A   E  DP  PB  WP  SB_A  ...  SB    CS  \
6     ainse101     0        0  1528  361  72  31  34   0     0  ...  20  11.5
7     alexg101   205     5481  1008  100  35   8  24   0   212  ...   8  12.0
8     alfaj002   130     3430  1135   71  13   9  14   0    72  ...   3   0.0
10    allaa001   453    11965  2395  208  52  24  41   0   292  ...  23  18.0
11    alleg001   342     8959  1762  146  32  24  27   0   233  ...   3   7.0
...        ...   ...      ...   ...  ...  ..  ..  ..  ..   ...  ...  ..   ...
1498  wingi101     0        0  1716  536  79  53  36   0     0  ...  17  16.0
1502  wockj001   216     5957  1212  119  39  17  27   0   188  ...   5  11.0
1508  wronr001    47     1360   296   32   8   3   4   0    25  ...   1   0.0
1509  wyneb001  1164    31563  6281  583  75  88  61   0   708  ...  10  13.0
1521  zaung001   910    24700  6134  418  88  59  51   0   651  ...  23  19.0

       BB   SO  IBB  HBP  SH  SF  GIDP  NL
6     125  122    0    3  44   0     0   1
7     154  381   12    5   4  19    34   1
8      22  179    8   18   0   1     4   1
10     87  223    4    9  35  16    27   1
11    130  192    3    5  15  11    35   0
```

```
...   ...  ...  ...  ...  ..  ..   ...  ..
1498  121   84    0    4  36   0     0   1
1502  277  278   14    7   5  12    52   1
1508    5   41    2    1   2   2     3   1
1509  626  428   41   17  58  36   119   0
1521  479  544   30   29  14  31    87   1

[430 rows x 38 columns]
```

It looks like the information in the catchers table is a better indicator of the player's career.

[175]:
```
fielders[fielders['retroID'] == 'alexg101'][['debutYear', 'finalYear']]
```

[175]:
```
     debutYear  finalYear
144       1975       1981
```

[174]:
```
catchers[catchers['retroID'] == 'alexg101'][['debutYear', 'finalYear']]
```

[174]:
```
    debutYear  finalYear
7        1975       1981
```

Thee years line up. So for any catcher who appears in the fielders table with his position as catcher, we're going to drop him from the fielders table and only use the catchers information. We'll keep catchers in the fielders table if they're in a different position.

[176]:
```
fielders[fielders['pos_C'] == 1]
```

[176]:
```
        retroID   GS  InnOuts    PO   A   E   DP    weight  height  debutYear  \
108    ainse101    0        0    11   0   0    0  0.426230    0.40       1910
144    alexg101   22      500    83   6   3    4  0.487705    0.55       1975
151    alfaj002    1       31     8   2   0    1  0.610656    0.55       2016
156    allaa001    0       33    15   0   0    2  0.590164    0.70       1986
169    alleg001    1       54     4   4   0    2  0.446721    0.40       1979
...         ...  ...      ...   ...  ..  ..  ...       ...     ...        ...
13914  wingi101    0        0     7   2   1    0  0.344262    0.35       1911
13956  wockj001  249     6120  1429  90  17  133  0.467213    0.45       1974
14066  wronr001    0        6     2   0   0    0  0.446721    0.50       1988
14074  wyneb001    0        6     0   0   0    0  0.467213    0.50       1976
14183  zaung001    0       33     3   2   0    1  0.385246    0.35       1995

       ...  SB    CS   BB   SO  IBB  HBP  SH  SF  GIDP  NL
108    ...  20  11.5  125  122    0    3  44   0     0   1
144    ...   8  12.0  154  381   12    5   4  19    34   1
151    ...   3   0.0   22  179    8   18   0   1     4   1
156    ...  23  18.0   87  223    4    9  35  16    27   1
169    ...   3   7.0  130  192    3    5  15  11    35   0
...    ...  ..   ...  ...  ...  ...  ...  ..  ..   ...  ..
13914  ...  17  16.0  121   84    0    4  36   0     0   1
```

```
13956  ...   5  11.0  277  278   14    7   5  12    52   1
14066  ...   1   0.0    5   41    2    1   2   2     3   1
14074  ...  10  13.0  626  428   41   17  58  36   119   0
14183  ...  23  19.0  479  544   30   29  14  31    87   1

[430 rows x 40 columns]
```

All fielders with a position of 'C' are in the catchers table, so we don't have to worry about leaving any by filtering with the following predicate.

[184]:
```
fielders = fielders[~(fielders['retroID'].isin(catchers['retroID']) &␣
 ↪fielders['pos_C'] == 1)]
```

[185]:
```
fielders.shape
```

[185]:
```
(5962, 40)
```

batting_build_tensor

April 30, 2020

Building the Batters Tensor

I've separated this from the model itself so that we could visualize and work through the data, but in the actual script this will just be a short few lines at the beginning of the model file.

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Loading the Data

[2]:
```
df = pd.read_csv('../core/output/batters.csv')
```

[3]:
```
indexer = df.reset_index()[['index', 'retroID']].to_dict()['retroID']
```

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15293 entries, 0 to 15292
Data columns (total 38 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   retroID    15293 non-null  object
 1   weight     15285 non-null  float64
 2   height     15287 non-null  float64
 3   debutYear  15293 non-null  int64
 4   finalYear  15293 non-null  int64
 5   pos_1B     15293 non-null  int64
 6   pos_2B     15293 non-null  int64
 7   pos_3B     15293 non-null  int64
 8   pos_C      15293 non-null  int64
 9   pos_OF     15293 non-null  int64
 10  pos_P      15293 non-null  int64
 11  pos_SS     15293 non-null  int64
 12  bats_L     15293 non-null  int64
 13  throws_L   15293 non-null  int64
 14  G          15293 non-null  int64
 15  AB         15293 non-null  int64
```

```
 16  PA          15293 non-null  int64
 17  R           15293 non-null  int64
 18  H           15293 non-null  int64
 19  1B          15293 non-null  int64
 20  2B          15293 non-null  int64
 21  3B          15293 non-null  int64
 22  HR          15293 non-null  int64
 23  RBI         15293 non-null  int64
 24  SB          15293 non-null  int64
 25  CS          15293 non-null  float64
 26  BB          15293 non-null  int64
 27  SO          15293 non-null  int64
 28  IBB         15293 non-null  int64
 29  HBP         15293 non-null  int64
 30  SH          15293 non-null  int64
 31  SF          15293 non-null  int64
 32  GIDP        15293 non-null  int64
 33  NL          15293 non-null  int64
 34  wOBA        15293 non-null  float64
 35  wRC+        15293 non-null  int64
 36  WAR         15293 non-null  float64
 37  Batting     15293 non-null  float64
dtypes: float64(6), int64(31), object(1)
memory usage: 4.4+ MB
```

[5]:
```
y = df['Batting'].values
```

[6]:
```
y
```

[6]:
```
array([0.00035809, 0.350195  , 0.157131  , ..., 0.0900877 , 0.135118  ,
       0.0901954 ])
```

[7]:
```
to_drop = ['retroID', 'debutYear', 'finalYear', 'Batting']
```

[8]:
```
df.drop(columns=to_drop, inplace=True)
```

[9]:
```
df
```

[9]:

	weight	height	pos_1B	pos_2B	pos_3B	pos_C	pos_OF	pos_P	pos_SS	\
0	0.569672	0.60	0	0	0	0	0	1	0	
1	0.426230	0.45	0	0	0	0	1	0	0	
2	0.467213	0.60	1	0	0	0	0	0	0	
3	0.467213	0.60	0	0	0	0	0	1	0	
4	0.442623	0.50	1	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	
15288	0.590164	0.65	0	0	0	0	1	0	0	
15289	0.434426	0.40	0	0	0	1	0	0	0	

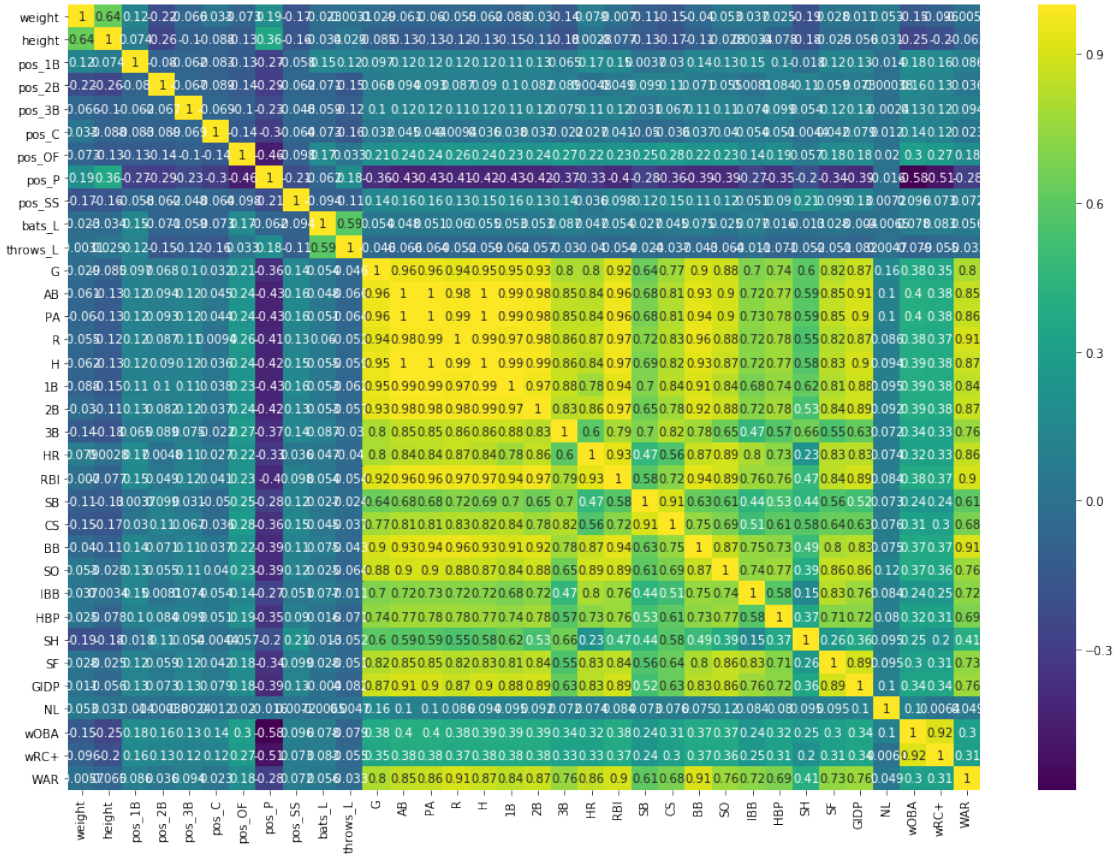
```
15290  0.487705    0.65       0       0       0      0       0      1       0
15291  0.397541    0.45       0       0       0      0       0      0       1
15292  0.467213    0.60       0       0       0      0       0      1       0

       bats_L  ...    SO  IBB  HBP  SH   SF  GIDP  NL   wOBA  wRC+    WAR
0           0  ...     2    0    0   1    0     0   1  0.000  -100   -0.1
1           0  ...  1383  293   32  21  121   328   1  0.403   153  136.3
2           0  ...   145    3    0   9    6    36   1  0.282    76   -1.7
3           0  ...     3    0    0   0    0     0   1  0.000  -100   -0.1
4           1  ...     5    0    0   0    0     1   1  0.184     0   -0.4
...       ...  ...   ...  ...  ...  ..  ...   ...  ..    ...   ...    ...
15288       0  ...   137    3    6  20    8    15   0  0.293    74   -0.9
15289       1  ...     6    0    0   0    0     0   0  0.225    37   -0.2
15290       0  ...    39    0    0  16    0     3   1  0.179     0   -0.3
15291       0  ...    50    1    2  18    0     8   1  0.254    52   -2.2
15292       0  ...     0    0    0   0    0     0   0  0.000     0    0.0

[15293 rows x 34 columns]
```

We now have a sort of proto-tensor, but maybe we can do some data manipulation to make the resulting model more efficient.

Observing Data Information

[10]:
```
plt.figure(figsize=(17,12))
ax = sns.heatmap(df.corr(), annot=True, cmap='viridis')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

[10]:
```
(34.0, 0.0)
```

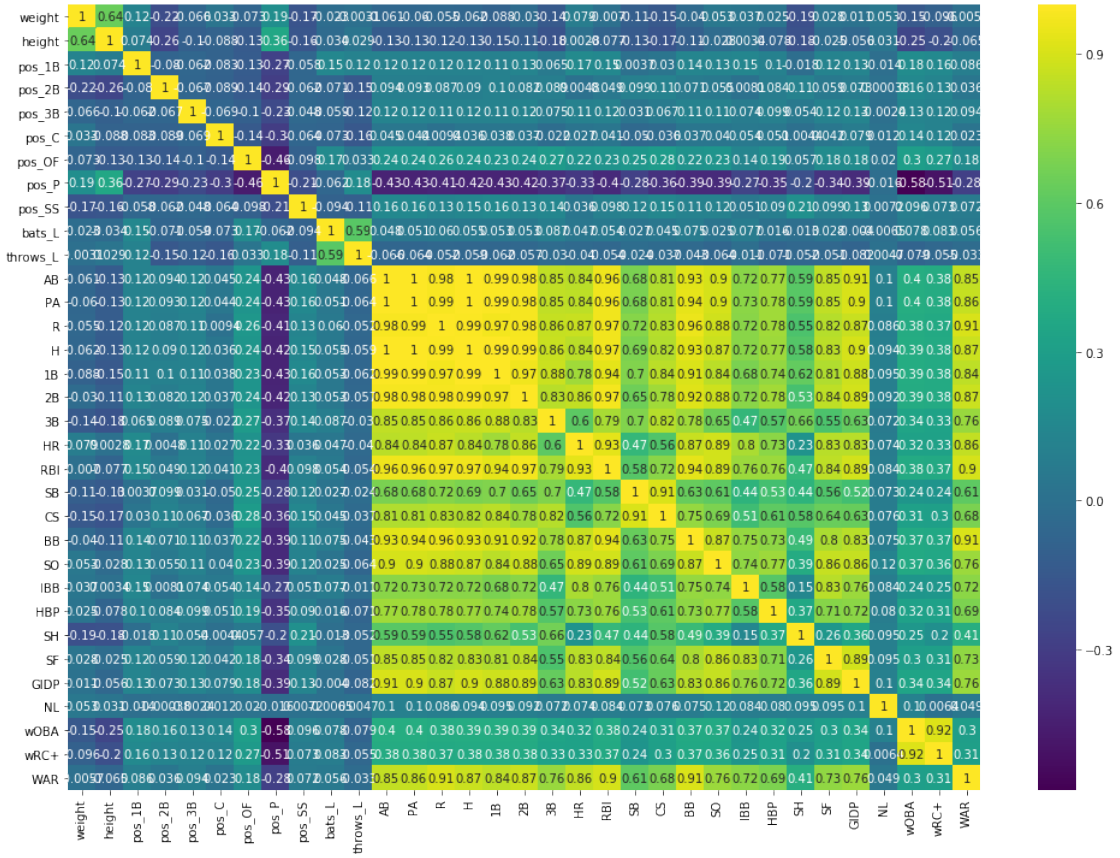


We see a high correlation between G (games) and AB/PA (at-bats/plate appearances). It makes sense that we can drop the G column.

[11]:
```
df.drop(columns=['G'], inplace=True)
```

[12]:
```
plt.figure(figsize=(17,12))
ax = sns.heatmap(df.corr(), annot=True, cmap='viridis')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

[12]: (33.0, 0.0)

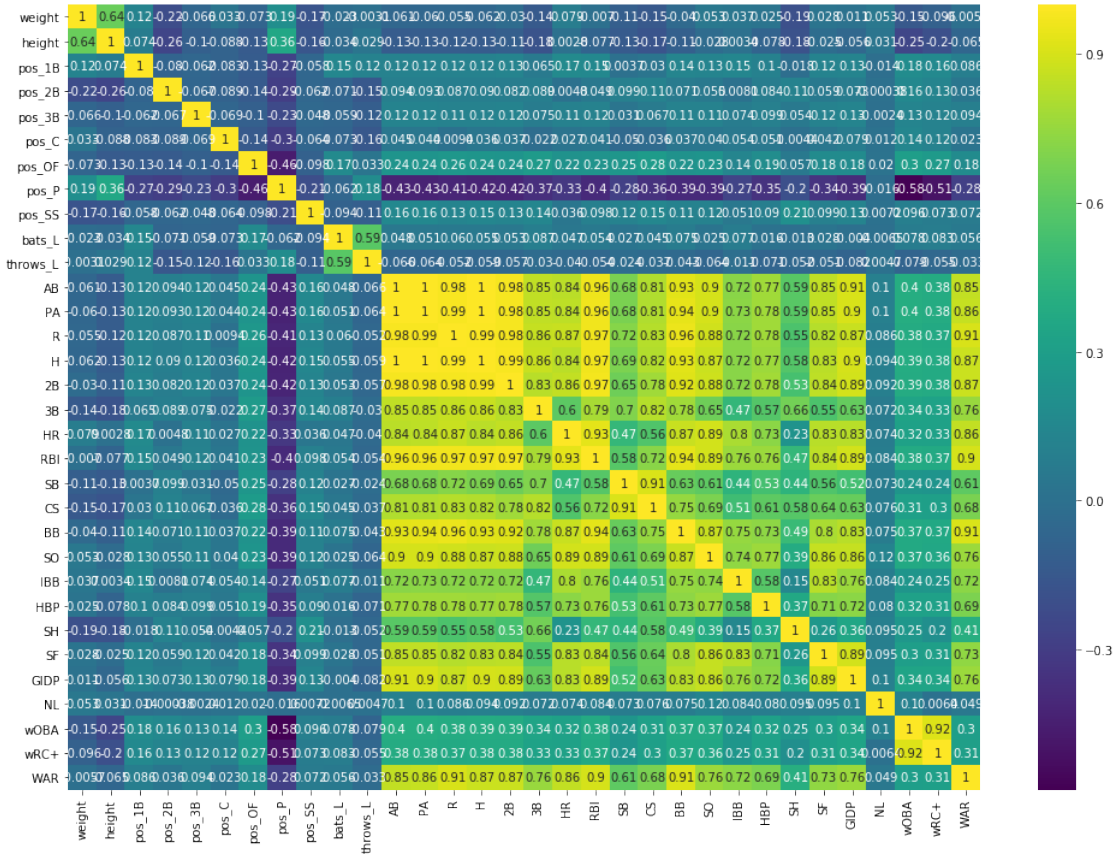


There's obviously a high correlation in H (hits) and 1B/2B/3B/HR (singles, doubles, triples and home runs). We added 1B as a column to help with statistics but it's unnecessary now - the relationship between hits and types of hits will be preserved in the model.

[13]:
```
df.drop(columns=['1B'], inplace=True)
```

[14]:
```
plt.figure(figsize=(17,12))
ax = sns.heatmap(df.corr(), annot=True, cmap='viridis')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

[14]: (32.0, 0.0)

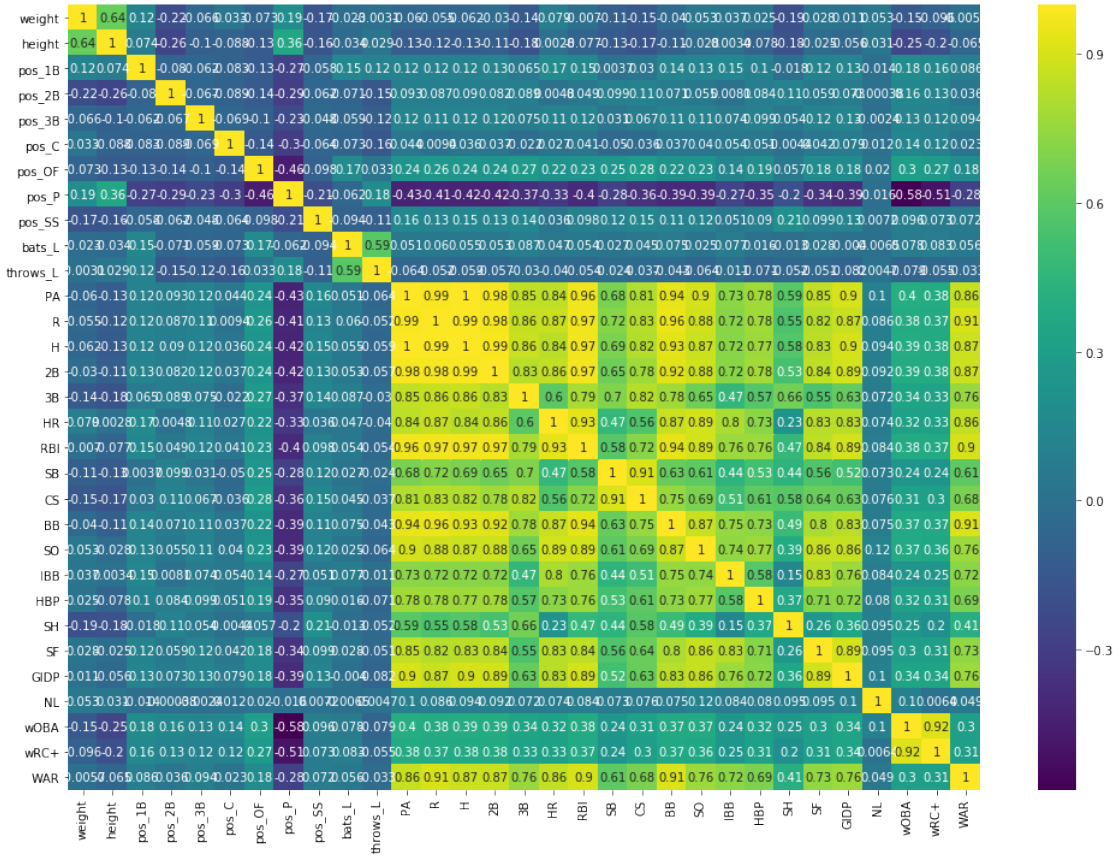


We have total correlation between AB (at-bats) and PA (plate-appearances). This makes sense, because PA is just AB with some other situations added in. PA is more robust and is better related to overall output (since it includes sacrifices, hits-by-pitch and walks) so we'll keep PA and get rid of AB.

[15]:
```
df.drop(columns=['AB'], inplace=True)
```

[16]:
```
plt.figure(figsize=(17,12))
ax = sns.heatmap(df.corr(), annot=True, cmap='viridis')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

[16]: (31.0, 0.0)



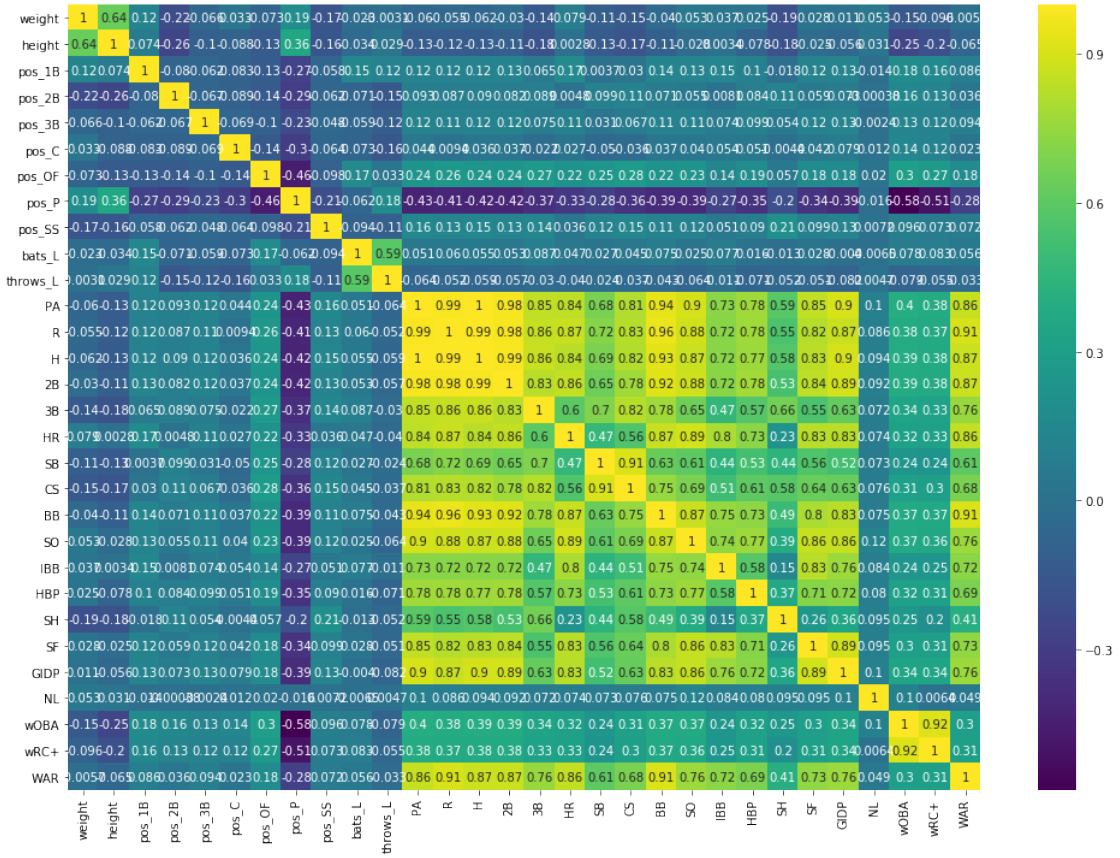
With the high correlation along the PA line, it may seem that we don't need them either. My reasoning for keeping them is a baseball-related one: we have most of the stats that make up a plate appearance, but we're missing the 'flied out' stat. This is an important one, as flyouts are a huge part of producing outs. Because of this, I'm going to keep PA as a stat.

One thing that I think we could drop is the RBI (runs batted in) stat. We don't see it appearing in many advanced stats, primarily wOBA and OPS+ with which we are concerned, and we intuitively see that it's encompasses factors beyond the pure output of the hitter. It could be said that the RBI stat tracks a hitter's ability to hit "under pressure", but that's the kind of soft feature we're not going to consider. I think we get more important information from hits, doubles, triples, home runs and even runs than we do from RBIs. We also see extremely high correlation between RBI and R/H/2B/HR, which further supports the decision to remove RBI.

[21]:
```
df.drop(columns=['RBI'], inplace=True)
```

[22]:
```
plt.figure(figsize=(17,12))
ax = sns.heatmap(df.corr(), annot=True, cmap='viridis')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

[22]: (30.0, 0.0)



We still see a hot spot around PA/R/H/2B, but these are all important stats that we're not going to remove.

One thing I think we SHOULD consider is that we have both wOBA and wRC+. There are a few issues with this:

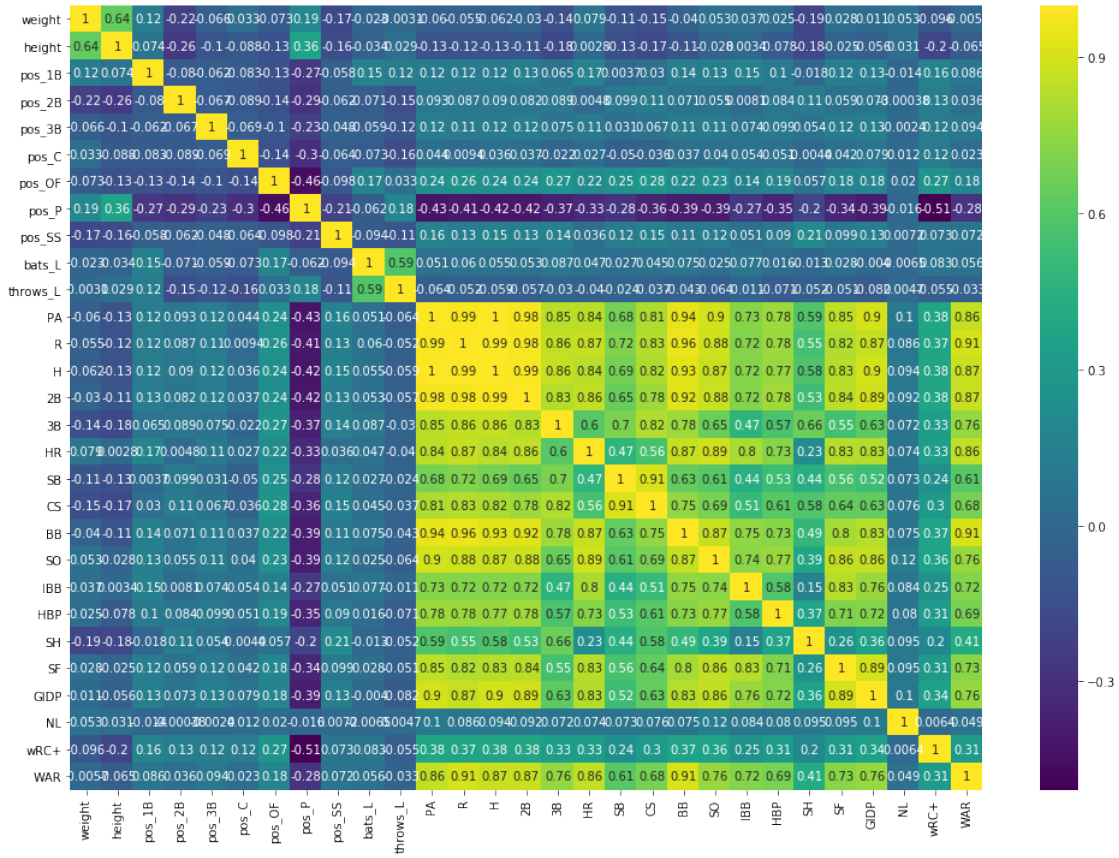
- They are both stats that essentially only consider offensive output, so do we really need two?
- wRC+ incorporates wRAA, which is built from wOBA, so the information is a bit repeated.
- We want to minimize the model's use of secondary/advanced stats which are extrapolated from the primary stats we have. However, wRC+ and WAR both normalize to league trends and thus offer a nice aggregation of data that might not be intrinsically found by the model.

So I think we can go ahead and get rid of wOBA.

[23]:
```
df.drop(columns=['wOBA'], inplace=True)
```

[24]:
```
plt.figure(figsize=(17,12))
ax = sns.heatmap(df.corr(), annot=True, cmap='viridis')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

[24]: (29.0, 0.0)



I said before that I don't want to drop R/H/2B/3B, so ignoring that area I think we now have a good looking model that's ready to run.

[25]:
```
df
```

[25]:

	weight	height	pos_1B	pos_2B	pos_3B	pos_C	pos_OF	pos_P	pos_SS	\
0	0.569672	0.60	0	0	0	0	0	1	0	
1	0.426230	0.45	0	0	0	0	1	0	0	
2	0.467213	0.60	1	0	0	0	0	0	0	
3	0.467213	0.60	0	0	0	0	0	1	0	
4	0.442623	0.50	1	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	
15288	0.590164	0.65	0	0	0	0	1	0	0	
15289	0.434426	0.40	0	0	0	1	0	0	0	
15290	0.487705	0.65	0	0	0	0	0	1	0	
15291	0.397541	0.45	0	0	0	0	0	0	1	
15292	0.467213	0.60	0	0	0	0	0	1	0	

```
       bats_L  ...    BB    SO  IBB  HBP  SH   SF  GIDP  NL  wRC+    WAR
0           0  ...     0     2    0    0   1    0     0   1  -100   -0.1
1           0  ...  1402  1383  293   32  21  121   328   1   153  136.3
2           0  ...    86   145    3    0   9    6    36   1    76   -1.7
3           0  ...     0     3    0    0   0    0     0   1  -100   -0.1
4           1  ...     4     5    0    0   0    0     1   1     0   -0.4
...       ...  ...   ...   ...  ...  ...  ..  ...   ...  ..   ...    ...
15288       0  ...    57   137    3    6  20    8    15   0    74   -0.9
15289       1  ...     2     6    0    0   0    0     0   0    37   -0.2
15290       0  ...     9    39    0    0  16    0     3   1     0   -0.3
15291       0  ...    34    50    1    2  18    0     8   1    52   -2.2
15292       0  ...     0     0    0    0   0    0     0   0     0    0.0

[15293 rows x 29 columns]
```

[28]:
```
# We'll add the 'Batting' column back in to save our tensor
df.insert(loc=len(df.columns), column='Batting', value=y)
```

[29]:
```
df
```

[29]:
```
         weight  height  pos_1B  pos_2B  pos_3B  pos_C  pos_OF  pos_P  pos_SS  \
0      0.569672    0.60       0       0       0      0       0      1       0
1      0.426230    0.45       0       0       0      0       1      0       0
2      0.467213    0.60       1       0       0      0       0      0       0
3      0.467213    0.60       0       0       0      0       0      1       0
4      0.442623    0.50       1       0       0      0       0      0       0
...         ...     ...     ...     ...     ...    ...     ...    ...     ...
15288  0.590164    0.65       0       0       0      0       1      0       0
15289  0.434426    0.40       0       0       0      1       0      0       0
15290  0.487705    0.65       0       0       0      0       0      1       0
15291  0.397541    0.45       0       0       0      0       0      0       1
15292  0.467213    0.60       0       0       0      0       0      1       0

       bats_L  ...    SO  IBB  HBP  SH   SF  GIDP  NL  wRC+    WAR   Batting
0           0  ...     2    0    0   1    0     0   1  -100   -0.1  0.000358
1           0  ...  1383  293   32  21  121   328   1   153  136.3  0.350195
2           0  ...   145    3    0   9    6    36   1    76   -1.7  0.157131
3           0  ...     3    0    0   0    0     0   1  -100   -0.1  0.000358
4           1  ...     5    0    0   0    0     1   1     0   -0.4  0.090001
...       ...  ...   ...  ...  ...  ..  ...   ...  ..   ...    ...       ...
15288       0  ...   137    3    6  20    8    15   0    74   -0.9  0.156062
15289       1  ...     6    0    0   0    0     0   0    37   -0.2  0.123425
15290       0  ...    39    0    0  16    0     3   1     0   -0.3  0.090088
15291       0  ...    50    1    2  18    0     8   1    52   -2.2  0.135118
15292       0  ...     0    0    0   0    0     0   0     0    0.0  0.090195

[15293 rows x 30 columns]
```

pitching_build_tensor

April 30, 2020

Building the Batters Tensor

I've separated this from the model itself so that we could visualize and work through the data, but in the actual script this will just be a short few lines at the beginning of the model file.

[14]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Loading the Data

[33]:
```
df = pd.read_csv('../core/output/pitchers.csv')
```

[34]:
```
indexer = df.reset_index()[['index', 'retroID']].to_dict()['retroID']
```

[35]:
```
y = df['Pitching'].values
```

[36]:
```
y
```

[36]:
```
array([0.602913, 0.636924, 0.603736, ..., 0.612847, 0.608497, 0.611166])
```

[37]:
```
df.columns
```

[37]:
```
Index(['retroID', 'BAOpp', 'CG', 'SHO', 'IPouts', 'H', 'ER', 'HR', 'BB', 'SO',
       'IBB', 'WP', 'HBP', 'BK', 'BFP', 'GF', 'R', 'SH', 'SF', 'GIDP', 'K%',
       'IP', 'K/9', 'BB/9', 'HR/9', 'BABIP', 'LOB%', 'ERA', 'FIP', 'WAR',
       'Pitching'],
      dtype='object')
```

[38]:
```
to_drop = ['retroID', 'Pitching']
```

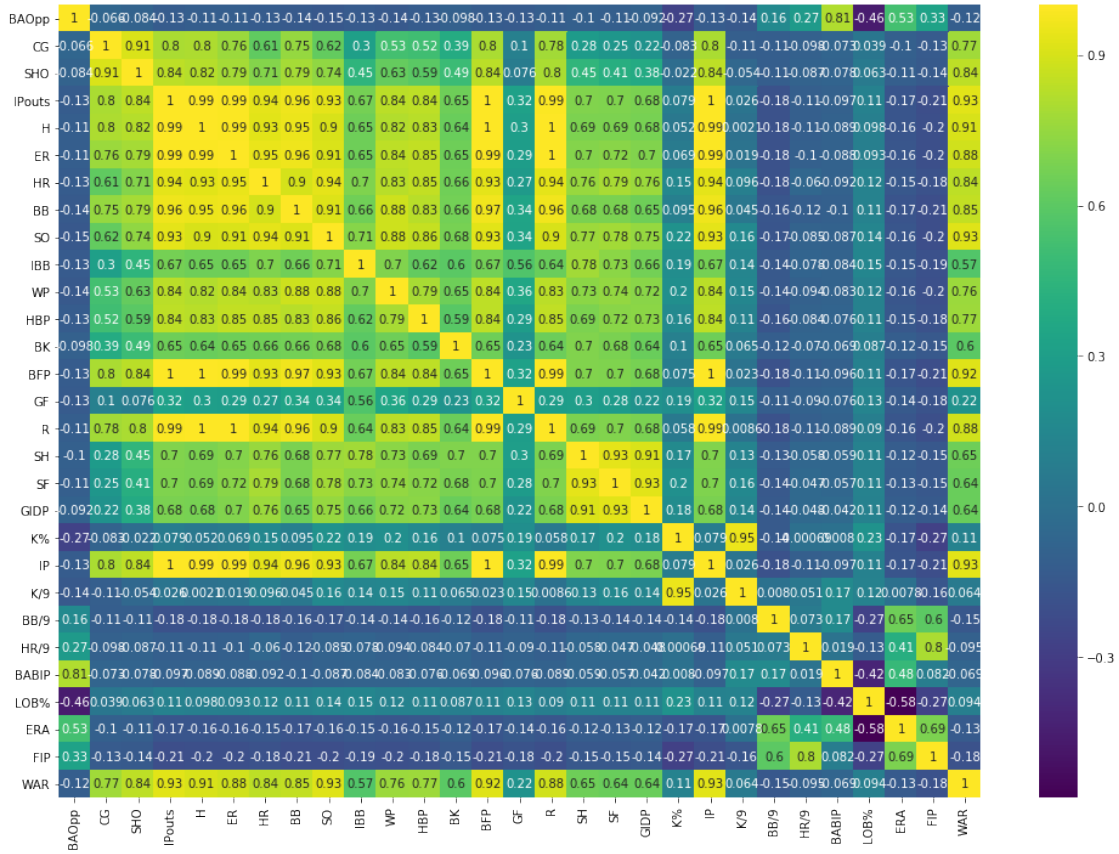
[39]:
```
df = df.drop(columns=to_drop)
```

Observing Data Information

[40]:
```
plt.figure(figsize=(17,12))
ax = sns.heatmap(df.corr(), annot=True, cmap='viridis')
bottom, top = ax.get_ylim()
```

```
ax.set_ylim(bottom + 0.5, top - 0.5)
```

[40]: (29.0, 0.0)

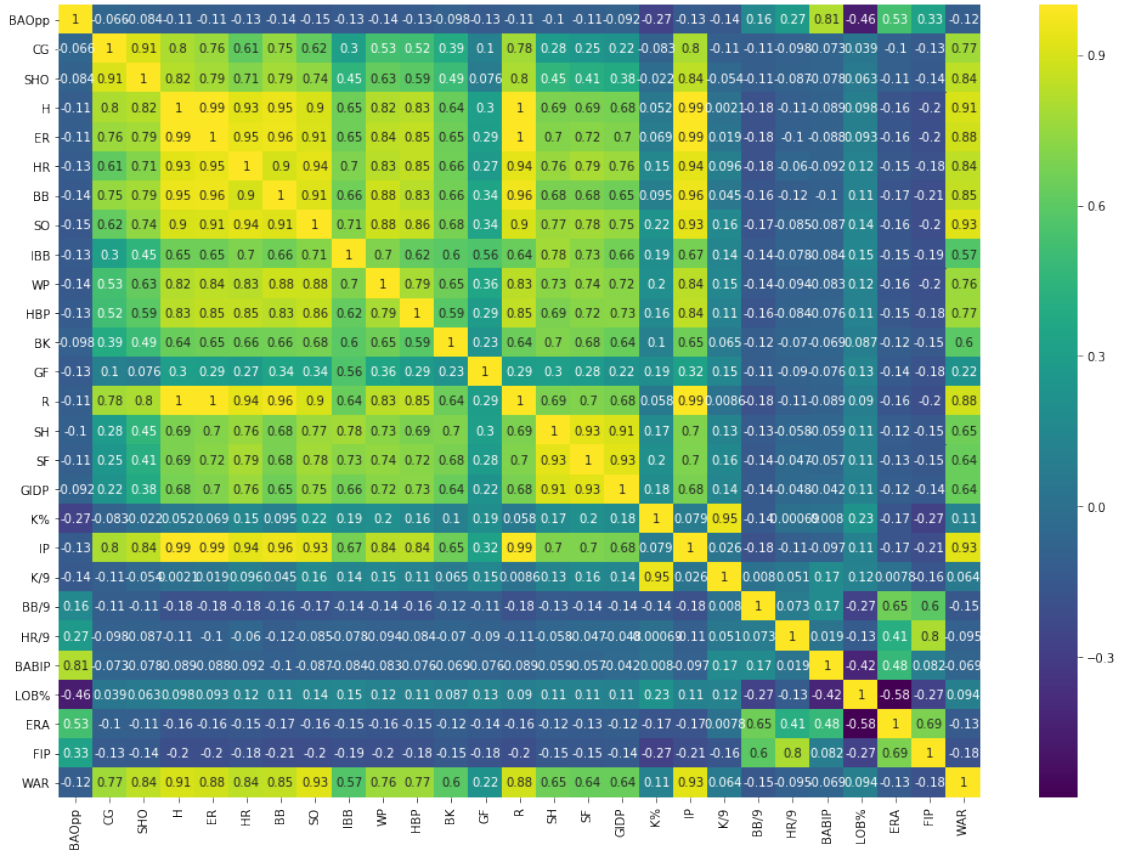


We see a lot of correlations with both IPouts and BFP, so we'll drop those.

[41]:
```
df = df.drop(columns=['IPouts', 'BFP'])
```

[42]:
```
plt.figure(figsize=(17,12))
ax = sns.heatmap(df.corr(), annot=True, cmap='viridis')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

[42]: (27.0, 0.0)

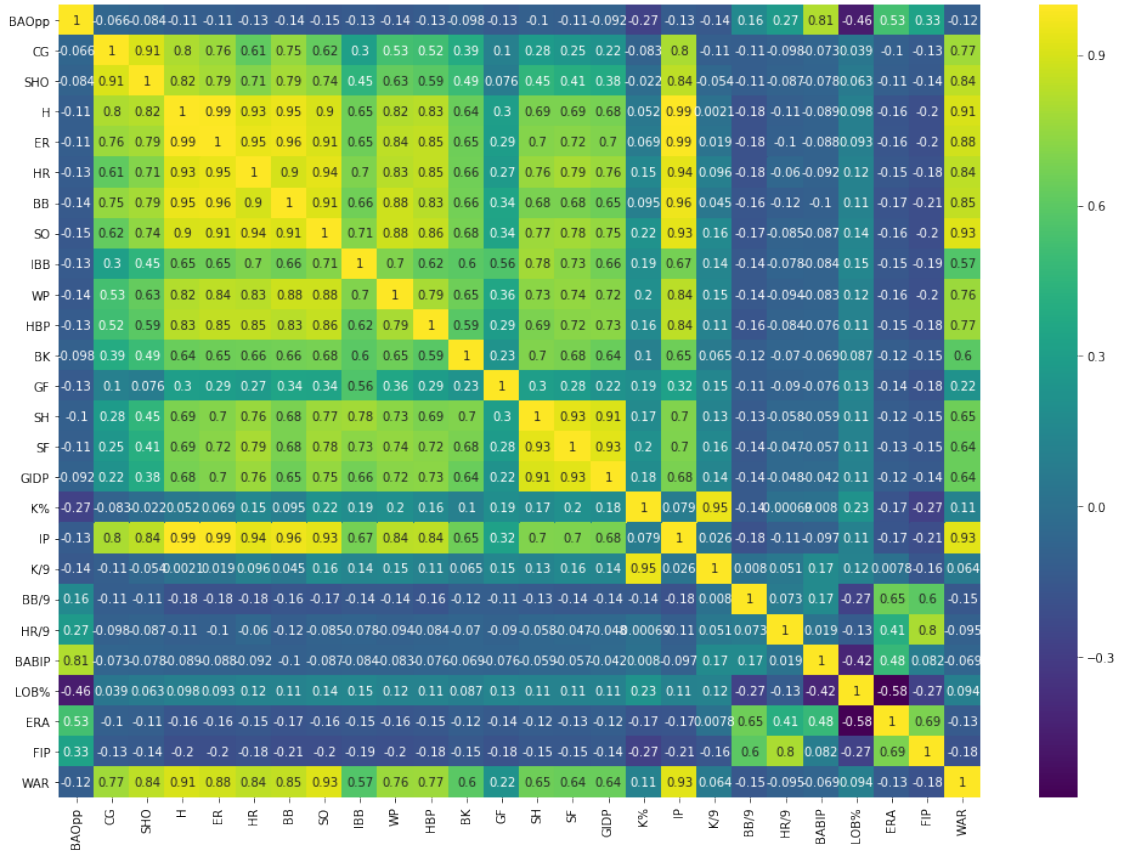


I think R (runs) is sufficiently covered by ER (earned runs) and HR (home runs), so I'll drop it too.

[43]:
```
df = df.drop(columns=['R'])
```

[44]:
```
plt.figure(figsize=(17,12))
ax = sns.heatmap(df.corr(), annot=True, cmap='viridis')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

[44]: (26.0, 0.0)



I'm happy with this version of the tensor.

[45]:
```
# We'll add the 'Pitching' column back in to save our tensor
df.insert(loc=len(df.columns), column='Pitching', value=y)
```

[46]:
```
df
```

[46]:

	BAOpp	CG	SHO	H	ER	HR	BB	SO	IBB	WP	...	IP	K/9	\
0	0.2574	0	0	296	160	41	183	340	22	12	...	0.062360	9.08	
1	0.2508	22	5	1085	468	89	457	641	45	22	...	0.205233	5.20	
2	0.2447	0	0	309	135	42	116	280	10	10	...	0.061102	7.62	
3	0.2786	37	5	1405	627	162	352	484	28	18	...	0.237967	3.39	
4	0.2804	31	6	1779	791	154	620	888	30	53	...	0.309765	4.77	
...	...	..	...	...	...	...	...	...	...	..	...	...	...	
8020	0.2700	30	5	956	366	54	301	207	0	8	...	0.171925	2.00	
8021	0.2717	23	3	767	374	35	468	383	0	28	...	0.145445	4.39	
8022	0.2286	0	0	169	71	18	114	210	11	16	...	0.038711	9.01	
8023	0.2760	9	2	660	253	56	203	223	29	10	...	0.118817	3.12	
8024	0.2183	0	0	57	22	3	34	80	5	2	...	0.013360	9.91	

```
      BB/9  HR/9  BABIP  LOB%   ERA   FIP   WAR  Pitching
0     4.89  1.09  0.285  74.5  4.27  4.45   1.1  0.602913
1     3.71  0.72  0.282  73.4  3.80  3.85  11.7  0.636924
2     3.16  1.14  0.281  77.7  3.67  4.24   0.6  0.603736
3     2.46  1.13  0.278  69.3  4.39  4.46  10.2  0.628847
4     3.33  0.83  0.295  70.0  4.25  4.25  22.7  0.666725
...    ...   ...    ...   ...   ...   ...   ...       ...
8020  2.91  0.52  0.267  70.7  3.54  3.80   9.3  0.630540
8021  5.36  0.40  0.283  69.0  4.28  3.96   3.3  0.610437
8022  4.89  0.77  0.267  78.7  3.00  3.94   2.7  0.612847
8023  2.84  0.78  0.270  73.2  3.54  3.93   1.9  0.608497
8024  4.21  0.37  0.293  79.1  2.72  3.22   1.1  0.611166

[8025 rows x 27 columns]
```

[ ]:

Game Log Processing

Taking Retrosheet game log CSV files and converting them into appropriate tensors for the predictive models

convert_gamelogs

May 7, 2020

Processing Game Logs

The information used here was obtained free of charge from and is copyrighted by Retrosheet. Interested parties may contact Retrosheet at "www.retrosheet.org".

We're going to use Retrosheet game logs as input data to our predictive model. The first thing we need to do is process them to fit our needs.

[135]:
```
import pandas as pd
import os
```

[156]:
```
df = pd.read_csv('../core/data/lahman/mlb_data/Teams.csv')
df = df[['teamID', 'franchID']]
team_dict = df.set_index('teamID').to_dict()['franchID']
team_dict['MLN'] = 'ATL'


def get_team(team):
    return team_dict[team] if team_dict[team] is not None else team
```

These are the columns of the Retrosheet game logs. This metadata was obtained here: https://www.retrosheet.org/gamelogs/glfields.txt

[137]:
```
columns = [
    'date',
    'game_number',
    'day_of_week',
    'visit_team',
    'visit_league',
    'visit_game_number',
    'home_team',
    'home_league',
    'home_game_number',
    'visit_score',
    'home_score',
    'game_length_outs',
    'day_night',
    'completion_info',
    'forfeit_info',
```

```
    'protest_info',
    'park_id',
    'attendance',
    'time_minutes',
    'visit_line_score',
    'home_line_score',
    'visit_ab',
    'visit_h',
    'visit_2b',
    'visit_3b',
    'visit_hr',
    'visit_rbi',
    'visit_sh',
    'visit_sf',
    'visit_hbp',
    'visit_bb',
    'visit_ibb',
    'visit_k',
    'visit_sb',
    'visit_cs',
    'visit_gidp',
    'visit_ci',
    'visit_lob',
    'visit_pitchers_used',
    'visit_individual_er',
    'visit_team_er',
    'visit_wp',
    'visit_bk',
    'visit_po',
    'visit_assists',
    'visit_e',
    'visit_passed_balls',
    'visit_double_plays',
    'visit_triple_plays',
    'home_ab',
    'home_h',
    'home_2b',
    'home_3b',
    'home_hr',
    'home_rbi',
    'home_sh',
    'home_sf',
    'home_hbp',
    'home_bb',
    'home_ibb',
    'home_k',
    'home_sb',
```

```
    'home_cs',
    'home_gidp',
    'home_ci',
    'home_lob',
    'home_pitchers_used',
    'home_individual_er',
    'home_team_er',
    'home_wp',
    'home_bk',
    'home_po',
    'home_assists',
    'home_e',
    'home_passed_balls',
    'home_double_plays',
    'home_triple_plays',
    'hp_ump_id',
    'hp_ump_name',
    '1b_ump_id',
    '1b_ump_name',
    '2b_ump_id',
    '2b_ump_name',
    '3b_ump_id',
    '3b_ump_name',
    'lf_ump_id',
    'lf_ump_name',
    'rf_ump_id',
    'rf_ump_name',
    'visit_manager_id',
    'visit_manager_name',
    'home_manager_id',
    'home_manager_name',
    'winning_pitcher_id',
    'winning_pitcher_name',
    'losing_pitcher_id',
    'losing_pitcher_name',
    'saving_pitcher_id',
    'saving_pitcher_name',
    'winning_rbi_batter_id',
    'winning_rbi_batter_name',
    'visit_sp_id',
    'visit_sp_name',
    'home_sp_id',
    'home_sp_name',
    'visit_player_1_id',
    'visit_player_1_name',
    'visit_player_1_pos',
    'visit_player_2_id',
```

```
    'visit_player_2_name',
    'visit_player_2_pos',
    'visit_player_3_id',
    'visit_player_3_name',
    'visit_player_3_pos',
    'visit_player_4_id',
    'visit_player_4_name',
    'visit_player_4_pos',
    'visit_player_5_id',
    'visit_player_5_name',
    'visit_player_5_pos',
    'visit_player_6_id',
    'visit_player_6_name',
    'visit_player_6_pos',
    'visit_player_7_id',
    'visit_player_7_name',
    'visit_player_7_pos',
    'visit_player_8_id',
    'visit_player_8_name',
    'visit_player_8_pos',
    'visit_player_9_id',
    'visit_player_9_name',
    'visit_player_9_pos',
    'home_player_1_id',
    'home_player_1_name',
    'home_player_1_pos',
    'home_player_2_id',
    'home_player_2_name',
    'home_player_2_pos',
    'home_player_3_id',
    'home_player_3_name',
    'home_player_3_pos',
    'home_player_4_id',
    'home_player_4_name',
    'home_player_4_pos',
    'home_player_5_id',
    'home_player_5_name',
    'home_player_5_pos',
    'home_player_6_id',
    'home_player_6_name',
    'home_player_6_pos',
    'home_player_7_id',
    'home_player_7_name',
    'home_player_7_pos',
    'home_player_8_id',
    'home_player_8_name',
    'home_player_8_pos',
```

```
    'home_player_9_id',
    'home_player_9_name',
    'home_player_9_pos',
    'additional_info',
    'acquisition_info'
]
```

The script is broken up here, then I later explore what I need to do to process the data. At the end I combine that all into one loop.

[12]:
```
for year in range(1919, 2020):
    file_path = '../core/data/retrosheet/gamelogs/GL{}'.format(year)
    df = pd.read_csv(file_path + '.TXT', delimiter = ',', header = 0, names =␣
 ↪columns)
    if os.path.exists(file_path + '.TXT'):
        os.remove(file_path + '.TXT')
    df.to_csv(file_path + '.csv')
```

[109]:
```
df = pd.read_csv('../core/data/retrosheet/gamelogs/GL2015.csv')
```

We don't want every column, so we'll specify exactly which ones to use

[110]:
```
df = df[[
        'date',
        'visit_team',
        'home_team',
        'visit_score',
        'home_score',
        'game_length_outs',
        'day_night',
        'park_id',
        'visit_manager_id',
        'home_manager_id',
        'visit_sp_id',
        'home_sp_id',
        'visit_player_1_id',
        'visit_player_2_id',
        'visit_player_3_id',
        'visit_player_4_id',
        'visit_player_5_id',
        'visit_player_6_id',
        'visit_player_7_id',
        'visit_player_8_id',
        'visit_player_9_id',
        'home_player_1_id',
        'home_player_2_id',
        'home_player_3_id',
```

```
        'home_player_4_id',
        'home_player_5_id',
        'home_player_6_id',
        'home_player_7_id',
        'home_player_8_id',
        'home_player_9_id'
    ]]
```

[111]:
```
df
```

[111]:

	date	visit_team	home_team	visit_score	home_score	\
0	20150406	MIN	DET	0	4	
1	20150406	CLE	HOU	0	2	
2	20150406	CHA	KCA	1	10	
3	20150406	TOR	NYA	6	1	
4	20150406	TEX	OAK	0	8	
...	...	...	...	...	...	
2423	20151004	CHN	MIL	3	1	
2424	20151004	WAS	NYN	0	1	
2425	20151004	MIA	PHI	2	7	
2426	20151004	CIN	PIT	0	4	
2427	20151004	COL	SFN	7	3	

	game_length_outs	day_night	park_id	visit_manager_id	home_manager_id	\
0	51	D	DET05	molip001	ausmb001	
1	51	N	HOU03	frant001	hinca001	
2	51	D	KAN06	ventr001	yoste001	
3	54	D	NYC21	gibbj001	giraj001	
4	51	N	OAK01	banij001	melvb001	
...	...	...	...	...	...	
2423	54	D	MIL06	maddj801	counc001	
2424	51	D	NYC20	willm003	collt801	
2425	51	D	PHI13	jennd801	mackp101	
2426	51	D	PIT08	pricb801	hurdc001	
2427	54	D	SFO03	weisw001	bochb002	

	...	visit_player_9_id	home_player_1_id	home_player_2_id	\
0	...	schaj002	davir003	kinsi001	
1	...	ramij003	altuj001	sprig001	
2	...	johnm006	escoa003	mousm001	
3	...	travd001	ellsj001	gardb001	
4	...	odorr001	gentc001	fulds001	
...	...	...	...	...	
2423	...	hared001	genns001	petes002	
2424	...	roart001	granc001	wrigd002	
2425	...	conla001	galvf001	altha001	
2426	...	smitj004	polag001	harrj002	

```
2427  ...          bergc001         pagaa001         tomlk001

     home_player_3_id home_player_4_id home_player_5_id home_player_6_id  \
0            cabrm001         martv001         martj006         cespy001
1            valbl001         gatte001         cartc002         castj006
2            cainl001         hosme001         morak001         gorda001
3            beltc001         teixm001         mccab002         headc001
4            zobrb001         butlb003         davii001         lawrb002
...               ...              ...              ...              ...
2423         linda001         davik003         santd002         pereh001
2424         murpd006         cespy001         dudal001         darnt001
2425         franm004         ruf-d001         franj004         blana001
2426         mccua001         walkn001         marts002         alvap001
2427         duffm002         poseb001         parkj002         willm008

     home_player_7_id home_player_8_id home_player_9_id
0            castn001         avila001         iglej001
1            lowrj001         rasmc001         marij002
2            riosa002         peres002         infao001
3            rodra001         drews001         gregd001
4            vogts001         semim001         sogae001
...               ...              ...              ...
2423         seguj002         maldm001         lopej004
2424         confm001         tejar001         degrj001
2425         krate001         ruppc001         buchd001
2426         cervf001         mercj002         happj001
2427         noonn001         willj005         cainm001

[2428 rows x 33 columns]
```

[112]:
```
df['date'] = df['date'].astype(str)
```

'date' isn't very useful, so we'll export it to three separate columns.

[113]:
```
df['year'] = df['date'].str[0:4].astype(int)
df['month'] = df['date'].str[4:6].astype(int)
df['day'] = df['date'].str[6:8].astype(int)
```

[114]:
```
df
```

[114]:
```
          date visit_team home_team  visit_score  home_score  \
0     20150406        MIN       DET            0           4
1     20150406        CLE       HOU            0           2
2     20150406        CHA       KCA            1          10
3     20150406        TOR       NYA            6           1
4     20150406        TEX       OAK            0           8
...        ...        ...       ...          ...         ...
```

```
2423  20151004        CHN       MIL            3           1
2424  20151004        WAS       NYN            0           1
2425  20151004        MIA       PHI            2           7
2426  20151004        CIN       PIT            0           4
2427  20151004        COL       SFN            7           3

      game_length_outs day_night park_id visit_manager_id home_manager_id  \
0                   51         D   DET05         molip001        ausmb001
1                   51         N   HOU03         frant001        hinca001
2                   51         D   KAN06         ventr001        yoste001
3                   54         D   NYC21         gibbj001        giraj001
4                   51         N   OAK01         banij001        melvb001
...                ...       ...     ...              ...             ...
2423                54         D   MIL06         maddj801        counc001
2424                51         D   NYC20         willm003        collt801
2425                51         D   PHI13         jennd801        mackp101
2426                51         D   PIT08         pricb801        hurdc001
2427                54         D   SFO03         weisw001        bochb002

      ... home_player_3_id home_player_4_id home_player_5_id home_player_6_id  \
0     ...         cabrm001         martv001         martj006         cespy001
1     ...         valbl001         gatte001         cartc002         castj006
2     ...         cainl001         hosme001         morak001         gorda001
3     ...         beltc001         teixm001         mccab002         headc001
4     ...         zobrb001         butlb003         davii001         lawrb002
...   ...              ...              ...              ...              ...
2423  ...         linda001         davik003         santd002         pereh001
2424  ...         murpd006         cespy001         dudal001         darnt001
2425  ...         franm004         ruf-d001         franj004         blana001
2426  ...         mccua001         walkn001         marts002         alvap001
2427  ...         duffm002         poseb001         parkj002         willm008

     home_player_7_id home_player_8_id home_player_9_id  year month day
0            castn001         avila001         iglej001  2015     4   6
1            lowrj001         rasmc001         marij002  2015     4   6
2            riosa002         peres002         infao001  2015     4   6
3            rodra001         drews001         gregd001  2015     4   6
4            vogts001         semim001         sogae001  2015     4   6
...               ...              ...              ...   ...   ...  ..
2423         seguj002         maldm001         lopej004  2015    10   4
2424         confm001         tejar001         degrj001  2015    10   4
2425         krate001         ruppc001         buchd001  2015    10   4
2426         cervf001         mercj002         happj001  2015    10   4
2427         noonn001         willj005         cainm001  2015    10   4

[2428 rows x 36 columns]
```

We aren't going to use every column in the final model, but we want to make sure that the ones we will are in the proper format.

[115]:
```
night_game = pd.get_dummies(df['day_night'], drop_first=True)
```

[116]:
```
night_game
```

[116]:
```
      N
0     0
1     1
2     0
3     0
4     1
...  ..
2423  0
2424  0
2425  0
2426  0
2427  0

[2428 rows x 1 columns]
```

[117]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2428 entries, 0 to 2427
Data columns (total 36 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   date                2428 non-null   object
 1   visit_team          2428 non-null   object
 2   home_team           2428 non-null   object
 3   visit_score         2428 non-null   int64
 4   home_score          2428 non-null   int64
 5   game_length_outs    2428 non-null   int64
 6   day_night           2428 non-null   object
 7   park_id             2428 non-null   object
 8   visit_manager_id    2428 non-null   object
 9   home_manager_id     2428 non-null   object
 10  winning_pitcher_id  2428 non-null   object
 11  losing_pitcher_id   2428 non-null   object
 12  saving_pitcher_id   1291 non-null   object
 13  visit_sp_id         2428 non-null   object
 14  home_sp_id          2428 non-null   object
 15  visit_player_1_id   2428 non-null   object
 16  visit_player_2_id   2428 non-null   object
 17  visit_player_3_id   2428 non-null   object
 18  visit_player_4_id   2428 non-null   object
```

```
 19  visit_player_5_id   2428 non-null   object
 20  visit_player_6_id   2428 non-null   object
 21  visit_player_7_id   2428 non-null   object
 22  visit_player_8_id   2428 non-null   object
 23  visit_player_9_id   2428 non-null   object
 24  home_player_1_id    2428 non-null   object
 25  home_player_2_id    2428 non-null   object
 26  home_player_3_id    2428 non-null   object
 27  home_player_4_id    2428 non-null   object
 28  home_player_5_id    2428 non-null   object
 29  home_player_6_id    2428 non-null   object
 30  home_player_7_id    2428 non-null   object
 31  home_player_8_id    2428 non-null   object
 32  home_player_9_id    2428 non-null   object
 33  year                2428 non-null   int64
 34  month               2428 non-null   int64
 35  day                 2428 non-null   int64
dtypes: int64(6), object(30)
memory usage: 683.0+ KB
```

[118]:
```
df.insert(loc=6, column='night_game', value=night_game)
```

[119]:
```
df
```

[119]:

	date	visit_team	home_team	visit_score	home_score	\
0	20150406	MIN	DET	0	4	
1	20150406	CLE	HOU	0	2	
2	20150406	CHA	KCA	1	10	
3	20150406	TOR	NYA	6	1	
4	20150406	TEX	OAK	0	8	
...	...	...	...	...	...	
2423	20151004	CHN	MIL	3	1	
2424	20151004	WAS	NYN	0	1	
2425	20151004	MIA	PHI	2	7	
2426	20151004	CIN	PIT	0	4	
2427	20151004	COL	SFN	7	3	

	game_length_outs	night_game	day_night	park_id	visit_manager_id	...	\
0	51	0	D	DET05	molip001	...	
1	51	1	N	HOU03	frant001	...	
2	51	0	D	KAN06	ventr001	...	
3	54	0	D	NYC21	gibbj001	...	
4	51	1	N	OAK01	banij001	...	
...	...	...	...	...	...	...	...
2423	54	0	D	MIL06	maddj801	...	
2424	51	0	D	NYC20	willm003	...	
2425	51	0	D	PHI13	jennd801	...	

```
2426               51           0         D   PIT08         pricb801  ...
2427               54           0         D   SFO03         weisw001  ...

     home_player_3_id home_player_4_id home_player_5_id home_player_6_id  \
0           cabrm001         martv001         martj006         cespy001
1           valbl001         gatte001         cartc002         castj006
2           cainl001         hosme001         morak001         gorda001
3           beltc001         teixm001         mccab002         headc001
4           zobrb001         butlb003         davii001         lawrb002
...              ...              ...              ...              ...
2423        linda001         davik003         santd002         pereh001
2424        murpd006         cespy001         dudal001         darnt001
2425        franm004         ruf-d001         franj004         blana001
2426        mccua001         walkn001         marts002         alvap001
2427        duffm002         poseb001         parkj002         willm008

     home_player_7_id home_player_8_id home_player_9_id  year month day
0           castn001         avila001         iglej001  2015     4   6
1           lowrj001         rasmc001         marij002  2015     4   6
2           riosa002         peres002         infao001  2015     4   6
3           rodra001         drews001         gregd001  2015     4   6
4           vogts001         semim001         sogae001  2015     4   6
...              ...              ...              ...   ...   ...  ..
2423        seguj002         maldm001         lopej004  2015    10   4
2424        confm001         tejar001         degrj001  2015    10   4
2425        krate001         ruppc001         buchd001  2015    10   4
2426        cervf001         mercj002         happj001  2015    10   4
2427        noonn001         willj005         cainm001  2015    10   4

[2428 rows x 37 columns]
```

[120]:
```
to_drop = ['date', 'day_night']
```

[121]:
```
df = df.drop(columns=to_drop)
```

[122]:
```
df
```

[122]:
```
     visit_team home_team  visit_score  home_score  game_length_outs  \
0           MIN       DET            0           4                51
1           CLE       HOU            0           2                51
2           CHA       KCA            1          10                51
3           TOR       NYA            6           1                54
4           TEX       OAK            0           8                51
...         ...       ...          ...         ...               ...
2423        CHN       MIL            3           1                54
2424        WAS       NYN            0           1                51
2425        MIA       PHI            2           7                51
```

```
2426        CIN       PIT            0           4               51
2427        COL       SFN            7           3               54

      night_game park_id visit_manager_id home_manager_id winning_pitcher_id  \
0              0   DET05         molip001        ausmb001           pricd001
1              1   HOU03         frant001        hinca001           keucd001
2              0   KAN06         ventr001        yoste001           venty001
3              0   NYC21         gibbj001        giraj001           hutcd001
4              1   OAK01         banij001        melvb001           grays001
...          ...     ...              ...             ...                ...
2423           0   MIL06         maddj801        counc001           hared001
2424           0   NYC20         willm003        collt801           clipt001
2425           0   PHI13         jennd801        mackp101           garcl005
2426           0   PIT08         pricb801        hurdc001           happj001
2427           0   SFO03         weisw001        bochb002           brotr001

      ... home_player_3_id home_player_4_id home_player_5_id home_player_6_id  \
0     ...         cabrm001         martv001         martj006         cespy001
1     ...         valbl001         gatte001         cartc002         castj006
2     ...         cainl001         hosme001         morak001         gorda001
3     ...         beltc001         teixm001         mccab002         headc001
4     ...         zobrb001         butlb003         davii001         lawrb002
...   ...              ...              ...              ...              ...
2423  ...         linda001         davik003         santd002         pereh001
2424  ...         murpd006         cespy001         dudal001         darnt001
2425  ...         franm004         ruf-d001         franj004         blana001
2426  ...         mccua001         walkn001         marts002         alvap001
2427  ...         duffm002         poseb001         parkj002         willm008

     home_player_7_id home_player_8_id home_player_9_id  year month day
0            castn001         avila001         iglej001  2015     4   6
1            lowrj001         rasmc001         marij002  2015     4   6
2            riosa002         peres002         infao001  2015     4   6
3            rodra001         drews001         gregd001  2015     4   6
4            vogts001         semim001         sogae001  2015     4   6
...               ...              ...              ...   ...   ...  ..
2423         seguj002         maldm001         lopej004  2015    10   4
2424         confm001         tejar001         degrj001  2015    10   4
2425         krate001         ruppc001         buchd001  2015    10   4
2426         cervf001         mercj002         happj001  2015    10   4
2427         noonn001         willj005         cainm001  2015    10   4

[2428 rows x 35 columns]
```

[123]:
```
df['visit_team'] = df['visit_team'].apply(get_team)
```

[124]:
```
df['home_team'] = df['home_team'].apply(get_team)
```

Early games only took place during the day, so we need to handle the effects of using one-hot encoding when dropping first with those.

[147]:
```
file_path = '../core/data/retrosheet/gamelogs/GL{}'.format(1919)
df = pd.read_csv(file_path + '.TXT', delimiter = ',', header = 0, names =␣
 ↪columns)
```

[148]:
```
df['day_night'].nunique()
```

[148]:
```
1
```

[149]:
```
pd.get_dummies(df['day_night'])
```

[149]:
```
      D
0     1
1     1
2     1
3     1
4     1
...  ..
1112  1
1113  1
1114  1
1115  1
1116  1

[1117 rows x 1 columns]
```

Final Script

[172]:
```
for year in range(1919, 2020):
    file_path = '../core/data/retrosheet/gamelogs/GL{}'.format(year)
    df = pd.read_csv(file_path + '.TXT', delimiter = ',', header = 0, names =␣
 ↪columns)
    df = df[[
        'date',
        'visit_team',
        'home_team',
        'visit_score',
        'home_score',
        'game_length_outs',
        'day_night',
        'park_id',
        'visit_manager_id',
        'home_manager_id',
        'winning_pitcher_id',
        'losing_pitcher_id',
        'saving_pitcher_id',
```

```
        'visit_sp_id',
        'home_sp_id',
        'visit_player_1_id',
        'visit_player_2_id',
        'visit_player_3_id',
        'visit_player_4_id',
        'visit_player_5_id',
        'visit_player_6_id',
        'visit_player_7_id',
        'visit_player_8_id',
        'visit_player_9_id',
        'home_player_1_id',
        'home_player_2_id',
        'home_player_3_id',
        'home_player_4_id',
        'home_player_5_id',
        'home_player_6_id',
        'home_player_7_id',
        'home_player_8_id',
        'home_player_9_id',
    ]]
    df['date'] = df['date'].astype(str)
    df['year'] = df['date'].str[0:4].astype(int)
    df['month'] = df['date'].str[4:6].astype(int)
    df['day'] = df['date'].str[6:8].astype(int)
    night_game = pd.get_dummies(df['day_night'], drop_first=(df['day_night'].
↪nunique() > 1))
    df.insert(loc=6, column='night_game', value=night_game)
    df = df.drop(columns=['date', 'day_night'])
    df['visit_team'] = df['visit_team'].apply(get_team)
    df['home_team'] = df['home_team'].apply(get_team)
    if os.path.exists(file_path + '.TXT'):
        os.remove(file_path + '.TXT')
    df.to_csv(file_path + '.csv', index=False)
```

[170]:
```
df = pd.read_csv('../core/data/retrosheet/gamelogs/GL2015.csv')
```

[173]:
```
df
```

[173]:

	visit_team	home_team	visit_score	home_score	game_length_outs	\
0	MIN	DET	0	4	51	
1	CLE	HOU	0	2	51	
2	CHW	KCR	1	10	51	
3	TOR	NYY	6	1	54	
4	TEX	OAK	0	8	51	
...	...	...	...	...	...	
2423	CHC	MIL	3	1	54	

```
2424        WSN       NYM            0           1               51
2425        FLA       PHI            2           7               51
2426        CIN       PIT            0           4               51
2427        COL       SFG            7           3               54

      night_game park_id visit_manager_id home_manager_id winning_pitcher_id  \
0              0   DET05         molip001        ausmb001           pricd001
1              1   HOU03         frant001        hinca001           keucd001
2              0   KAN06         ventr001        yoste001           venty001
3              0   NYC21         gibbj001        giraj001           hutcd001
4              1   OAK01         banij001        melvb001           grays001
...          ...     ...              ...             ...                ...
2423           0   MIL06         maddj801        counc001           hared001
2424           0   NYC20         willm003        collt801           clipt001
2425           0   PHI13         jennd801        mackp101           garcl005
2426           0   PIT08         pricb801        hurdc001           happj001
2427           0   SFO03         weisw001        bochb002           brotr001

      ... home_player_3_id home_player_4_id home_player_5_id home_player_6_id  \
0     ...         cabrm001         martv001         martj006         cespy001
1     ...         valbl001         gatte001         cartc002         castj006
2     ...         cainl001         hosme001         morak001         gorda001
3     ...         beltc001         teixm001         mccab002         headc001
4     ...         zobrb001         butlb003         davii001         lawrb002
...   ...              ...              ...              ...              ...
2423  ...         linda001         davik003         santd002         pereh001
2424  ...         murpd006         cespy001         dudal001         darnt001
2425  ...         franm004         ruf-d001         franj004         blana001
2426  ...         mccua001         walkn001         marts002         alvap001
2427  ...         duffm002         poseb001         parkj002         willm008

     home_player_7_id home_player_8_id home_player_9_id  year month day
0            castn001         avila001         iglej001  2015     4   6
1            lowrj001         rasmc001         marij002  2015     4   6
2            riosa002         peres002         infao001  2015     4   6
3            rodra001         drews001         gregd001  2015     4   6
4            vogts001         semim001         sogae001  2015     4   6
...               ...              ...              ...   ...   ...  ..
2423         seguj002         maldm001         lopej004  2015    10   4
2424         confm001         tejar001         degrj001  2015    10   4
2425         krate001         ruppc001         buchd001  2015    10   4
2426         cervf001         mercj002         happj001  2015    10   4
2427         noonn001         willj005         cainm001  2015    10   4

[2428 rows x 35 columns]
```

When we do the actual script, we don't want the column names hardcoded into it. So I've pasted

those to .csv files but I need to process them a bit.

[252]:
```
gla = pd.read_csv('../core/data/retrosheet/rs_gl_cols_all.csv', header=None)
gl = pd.read_csv('../core/data/retrosheet/rs_gl_cols.csv', header=None)
```

[254]:
```
gl
```

[254]:

	0
0	'date',
1	'visit_team',
2	'home_team',
3	'visit_score',
4	'home_score',
5	'game_length_outs',
6	'day_night',
7	'park_id',
8	'visit_manager_id',
9	'home_manager_id',
10	'winning_pitcher_id',
11	'losing_pitcher_id',
12	'saving_pitcher_id',
13	'visit_sp_id',
14	'home_sp_id',
15	'visit_player_1_id',
16	'visit_player_2_id',
17	'visit_player_3_id',
18	'visit_player_4_id',
19	'visit_player_5_id',
20	'visit_player_6_id',
21	'visit_player_7_id',
22	'visit_player_8_id',
23	'visit_player_9_id',
24	'home_player_1_id',
25	'home_player_2_id',
26	'home_player_3_id',
27	'home_player_4_id',
28	'home_player_5_id',
29	'home_player_6_id',
30	'home_player_7_id',
31	'home_player_8_id',
32	'home_player_9_id'

We need to get rid of whitespace, commas and quotation marks.

[189]:
```
gla.iloc[156][0].replace(',', '')
```

[189]:
```
"    'home_player_9_id'"
```

[190]:
```
from functools import reduce
```

[241]:
```
def trim_cell(cell):
    replacements = {' ': '', "'": '', ',': ''}
    string = cell[0]
    return reduce(lambda a, kv: a.replace(*kv), replacements.items(), string)
```

[200]:
```
print(trim_cell(gla.iloc[156]))
```

```
home_player_9_id
```

[255]:
```
gla = gla.apply(trim_cell, axis=1)
```

[256]:
```
type(gla)
```

[256]:
```
pandas.core.series.Series
```

[257]:
```
gl = gl.apply(trim_cell, axis=1)
```

[258]:
```
gla.to_csv('../core/data/retrosheet/rs_gl_cols_all.csv', header=None)
gl.to_csv('../core/data/retrosheet/rs_gl_cols.csv', header=None)
```

[259]:
```
gla = pd.read_csv('../core/data/retrosheet/rs_gl_cols_all.csv', header=None)
gl = pd.read_csv('../core/data/retrosheet/rs_gl_cols.csv', header=None)
```

[260]:
```
gl
```

[260]:

	0	1
0	0	date
1	1	visit_team
2	2	home_team
3	3	visit_score
4	4	home_score
5	5	game_length_outs
6	6	day_night
7	7	park_id
8	8	visit_manager_id
9	9	home_manager_id
10	10	winning_pitcher_id
11	11	losing_pitcher_id
12	12	saving_pitcher_id
13	13	visit_sp_id
14	14	home_sp_id
15	15	visit_player_1_id
16	16	visit_player_2_id
17	17	visit_player_3_id
18	18	visit_player_4_id

```
19  19   visit_player_5_id
20  20   visit_player_6_id
21  21   visit_player_7_id
22  22   visit_player_8_id
23  23   visit_player_9_id
24  24    home_player_1_id
25  25    home_player_2_id
26  26    home_player_3_id
27  27    home_player_4_id
28  28    home_player_5_id
29  29    home_player_6_id
30  30    home_player_7_id
31  31    home_player_8_id
32  32    home_player_9_id
```

[261]:
```
gla[1].tolist()
```

[261]:
```
['date',
 'game_number',
 'day_of_week',
 'visit_team',
 'visit_league',
 'visit_game_number',
 'home_team',
 'home_league',
 'home_game_number',
 'visit_score',
 'home_score',
 'game_length_outs',
 'day_night',
 'completion_info',
 'forfeit_info',
 'protest_info',
 'park_id',
 'attendance',
 'time_minutes',
 'visit_line_score',
 'home_line_score',
 'visit_ab',
 'visit_h',
 'visit_2b',
 'visit_3b',
 'visit_hr',
 'visit_rbi',
 'visit_sh',
 'visit_sf',
 'visit_hbp',
```

```
'visit_bb',
'visit_ibb',
'visit_k',
'visit_sb',
'visit_cs',
'visit_gidp',
'visit_ci',
'visit_lob',
'visit_pitchers_used',
'visit_individual_er',
'visit_team_er',
'visit_wp',
'visit_bk',
'visit_po',
'visit_assists',
'visit_e',
'visit_passed_balls',
'visit_double_plays',
'visit_triple_plays',
'home_ab',
'home_h',
'home_2b',
'home_3b',
'home_hr',
'home_rbi',
'home_sh',
'home_sf',
'home_hbp',
'home_bb',
'home_ibb',
'home_k',
'home_sb',
'home_cs',
'home_gidp',
'home_ci',
'home_lob',
'home_pitchers_used',
'home_individual_er',
'home_team_er',
'home_wp',
'home_bk',
'home_po',
'home_assists',
'home_e',
'home_passed_balls',
'home_double_plays',
'home_triple_plays',
```

```
'hp_ump_id',
'hp_ump_name',
'1b_ump_id',
'1b_ump_name',
'2b_ump_id',
'2b_ump_name',
'3b_ump_id',
'3b_ump_name',
'lf_ump_id',
'lf_ump_name',
'rf_ump_id',
'rf_ump_name',
'visit_manager_id',
'visit_manager_name',
'home_manager_id',
'home_manager_name',
'winning_pitcher_id',
'winning_pitcher_name',
'losing_pitcher_id',
'losing_pitcher_name',
'saving_pitcher_id',
'saving_pitcher_name',
'winning_rbi_batter_id',
'winning_rbi_batter_name',
'visit_sp_id',
'visit_sp_name',
'home_sp_id',
'home_sp_name',
'visit_player_1_id',
'visit_player_1_name',
'visit_player_1_pos',
'visit_player_2_id',
'visit_player_2_name',
'visit_player_2_pos',
'visit_player_3_id',
'visit_player_3_name',
'visit_player_3_pos',
'visit_player_4_id',
'visit_player_4_name',
'visit_player_4_pos',
'visit_player_5_id',
'visit_player_5_name',
'visit_player_5_pos',
'visit_player_6_id',
'visit_player_6_name',
'visit_player_6_pos',
'visit_player_7_id',
```

```
 'visit_player_7_name',
 'visit_player_7_pos',
 'visit_player_8_id',
 'visit_player_8_name',
 'visit_player_8_pos',
 'visit_player_9_id',
 'visit_player_9_name',
 'visit_player_9_pos',
 'home_player_1_id',
 'home_player_1_name',
 'home_player_1_pos',
 'home_player_2_id',
 'home_player_2_name',
 'home_player_2_pos',
 'home_player_3_id',
 'home_player_3_name',
 'home_player_3_pos',
 'home_player_4_id',
 'home_player_4_name',
 'home_player_4_pos',
 'home_player_5_id',
 'home_player_5_name',
 'home_player_5_pos',
 'home_player_6_id',
 'home_player_6_name',
 'home_player_6_pos',
 'home_player_7_id',
 'home_player_7_name',
 'home_player_7_pos',
 'home_player_8_id',
 'home_player_8_name',
 'home_player_8_pos',
 'home_player_9_id',
 'home_player_9_name',
 'home_player_9_pos',
 'additional_info',
 'acquisition_info']
```

[247]:
```
gl
```

[247]:

	0
0	visit_team
1	home_team
2	visit_score
3	home_score
4	game_length_outs
5	day_night

```
6              park_id
7     visit_manager_id
8      home_manager_id
9   winning_pitcher_id
10   losing_pitcher_id
11   saving_pitcher_id
12         visit_sp_id
13          home_sp_id
14   visit_player_1_id
15   visit_player_2_id
16   visit_player_3_id
17   visit_player_4_id
18   visit_player_5_id
19   visit_player_6_id
20   visit_player_7_id
21   visit_player_8_id
22   visit_player_9_id
23    home_player_1_id
24    home_player_2_id
25    home_player_3_id
26    home_player_4_id
27    home_player_5_id
28    home_player_6_id
29    home_player_7_id
30    home_player_8_id
31    home_player_9_id
```

[ ]:

create_gamelog_tensors

May 10, 2020

[13]:
```
import os
import pandas as pd
import numpy as np
from tensorflow.keras.models import load_model
from joblib import load
pd.options.mode.chained_assignment = None  # default='warn'
```

[14]:
```
bat = load_model('../core/models/model_batting.h5')
pitch = load_model('../core/models/model_pitching.h5')
bat_scaler = load('../core/models/batting_scaler.save')
pitch_scaler = load('../core/models/pitching_scaler.save')
```

[15]:
```
gl = pd.read_csv('../core/data/retrosheet/gamelogs/GL2015.csv')
```

[16]:
```
gl
```

[16]:

	visit_team	home_team	visit_score	home_score	game_length_outs	\
0	MIN	DET	0	4	51	
1	CLE	HOU	0	2	51	
2	CHW	KCR	1	10	51	
3	TOR	NYY	6	1	54	
4	TEX	OAK	0	8	51	
...	...	...	...	...	...	
2423	CHC	MIL	3	1	54	
2424	WSN	NYM	0	1	51	
2425	FLA	PHI	2	7	51	
2426	CIN	PIT	0	4	51	
2427	COL	SFG	7	3	54	

	night_game	park_id	visit_manager_id	home_manager_id	visit_sp_id	...	\
0	0	DET05	molip001	ausmb001	hughp001	...	
1	1	HOU03	frant001	hinca001	klubc001	...	
2	0	KAN06	ventr001	yoste001	samaj001	...	
3	0	NYC21	gibbj001	giraj001	hutcd001	...	
4	1	OAK01	banij001	melvb001	gally001	...	
...	...	...	...	...	...	...	
2423	0	MIL06	maddj801	counc001	hared001	...	

```
2424           0   NYC20         willm003        collt801    roart001  ...
2425           0   PHI13         jennd801        mackp101    conla001  ...
2426           0   PIT08         pricb801        hurdc001    smitj004  ...
2427           0   SFO03         weisw001        bochb002    bergc001  ...

     home_player_4_id home_player_5_id home_player_6_id home_player_7_id  \
0            martv001         martj006         cespy001         castn001
1            gatte001         cartc002         castj006         lowrj001
2            hosme001         morak001         gorda001         riosa002
3            teixm001         mccab002         headc001         rodra001
4            butlb003         davii001         lawrb002         vogts001
...               ...              ...              ...              ...
2423         davik003         santd002         pereh001         seguj002
2424         cespy001         dudal001         darnt001         confm001
2425         ruf-d001         franj004         blana001         krate001
2426         walkn001         marts002         alvap001         cervf001
2427         poseb001         parkj002         willm008         noonn001

     home_player_8_id home_player_9_id  year month day home_win
0            avila001         iglej001  2015     4   6        1
1            rasmc001         marij002  2015     4   6        1
2            peres002         infao001  2015     4   6        1
3            drews001         gregd001  2015     4   6        0
4            semim001         sogae001  2015     4   6        1
...               ...              ...   ...   ...  ..      ...
2423         maldm001         lopej004  2015    10   4        0
2424         tejar001         degrj001  2015    10   4        1
2425         ruppc001         buchd001  2015    10   4        1
2426         mercj002         happj001  2015    10   4        1
2427         willj005         cainm001  2015    10   4        0

[2428 rows x 33 columns]
```

[17]:
```
columns = {
    'batting': [],
    'pitching': []
}
```

[18]:
```
batters = pd.read_csv('../core/output/batters.csv')
batter_years = pd.read_csv('../core/output/batting.csv')
batters_not_counted = list(batter_years[~batter_years['retroID']
                                        .isin(batters['retroID'])]['retroID'].
 ↪values)
pitchers = pd.read_csv('../core/output/pitchers.csv')
pitcher_years = pd.read_csv('../core/output/pitching.csv')
bat_scaler = load('../core/models/batting_scaler.save')
pitch_scaler = load('../core/models/pitching_scaler.save')
```

```
scalers = {
    'batting': bat_scaler,
    'pitching': pitch_scaler
}
career_features = {
    'batting': [
        'G', 'AB', 'PA', 'R', 'H', '1B', '2B', '3B',
        'HR', 'RBI', 'SB', 'CS', 'BB', 'SO', 'IBB',
        'HBP', 'SH', 'SF', 'GIDP'
    ],
    'pitching': [
        'CG', 'SHO', 'H', 'ER', 'HR', 'BB', 'SO',
        'BAOpp', 'ERA', 'IBB', 'WP', 'HBP', 'BK',
        'BFP', 'GF', 'R', 'SH', 'SF', 'GIDP'
    ]
}
unwanted_features = {
    'batting': ['retroID', 'G', 'AB', '1B', 'RBI', 'wOBA', 'Batting'],
    'pitching': ['IPouts', 'BFP', 'R', 'Pitching']
}
players = {
    'batting': {
        'players': batters,
        'years': batter_years
    },
    'pitching': {
        'players': pitchers,
        'years': pitcher_years
    }
}
```

[19]:
```
def to_tensor_input(scaler, player, label):
    scalers[label] = scaler
    return scaler.transform(player.values.reshape(-1, player.shape[0]))[0]


def convert_single_player(retro_id, year, player_type_label):
    scaler = scalers[player_type_label]
    if retro_id in batters_not_counted:
        return np.zeros(shape=(1, 30))
    player_table = players[player_type_label]['players']
    player_so_far_table = players[player_type_label]['years']
    player = player_table[player_table['retroID'] == retro_id]
    player_so_far = player_so_far_table[(player_so_far_table['retroID'] ==␣
 ↪retro_id)
                                        & (player_so_far_table['yearID'] <=␣
 ↪year)]
```

```
    if not player.size | player_so_far.size:
        print('Handled: {}'.format(retro_id))
        return np.zeros(shape=(1, 30))
    player_so_far = player_so_far.groupby('retroID').sum()
    features = career_features[player_type_label]
    try:
        for column in player[features]:
            player.iloc[0][column] = player_so_far.iloc[0][column]
    except:
        print(retro_id)
    player_columns_to_drop = unwanted_features[player_type_label]
    player = player.drop(columns=player_columns_to_drop)
    if not len(list(columns[player_type_label])):
        columns[player_type_label] = player.columns
    return to_tensor_input(scaler, player.T, player_type_label)


def get_batter_as_tensor_input(batter, year):
    scaler = scalers['batting']
    player = batters[batters['retroID'] == batter]
    player_so_far = batter_years[(batter_years['retroID'] == batter)
                                 & (batter_years['yearID'] <= year)]
    player_so_far = player_so_far.groupby('retroID').sum()
    features = ['G', 'AB', 'PA', 'R', 'H', '1B', '2B', '3B',
                'HR', 'RBI', 'SB', 'CS', 'BB', 'SO', 'IBB',
                'HBP', 'SH', 'SF', 'GIDP']
    for column in player[features]:
        player.iloc[0][column] = player_so_far.iloc[0][column]
    player_columns_to_drop = ['retroID', 'wOBA', 'Batting']
    player = player.drop(columns=player_columns_to_drop)
    return to_tensor_input(scaler, player, 'batting')
```

[20]:
```
convert_single_player('bettm001', 2015, 'batting')
```

[20]:
```
array([0.42623   , 0.3       , 0.        , 0.        , 0.        ,
       0.        , 1.        , 0.        , 0.        , 0.        ,
       0.        , 0.16129032, 0.2288002 , 0.2671024 , 0.22673872,
       0.30697051, 0.13612565, 0.1824147 , 0.08961593, 0.07462687,
       0.14503518, 0.17866769, 0.03633721, 0.06666667, 0.01486989,
       0.25      , 0.10379747, 0.        , 0.21133094, 0.26473988])
```

[21]:
```
gl.iloc[43]
```

[21]:
```
visit_team               SFG
home_team                SDP
visit_score                1
home_score                 0
```

```
game_length_outs           72
night_game                  0
park_id                 SAN02
visit_manager_id     bochb002
home_manager_id      blacb001
visit_sp_id          hudst001
home_sp_id           kenni001
visit_player_1_id    aokin001
visit_player_2_id    panij002
visit_player_3_id    pagaa001
visit_player_4_id    poseb001
visit_player_5_id    crawb001
visit_player_6_id    mcgec001
visit_player_7_id    blang001
visit_player_8_id    ariaj001
visit_player_9_id    hudst001
home_player_1_id     myerw001
home_player_2_id     norrd001
home_player_3_id     kempm001
home_player_4_id     uptoj001
home_player_5_id     middw001
home_player_6_id     alony001
home_player_7_id     gyorj001
home_player_8_id     amara001
home_player_9_id     kenni001
year                    2015
month                      4
day                        9
home_win                   0
Name: 43, dtype: object
```

[22]:
```
v1 = gl.iloc[0]['visit_player_1_id']
```

[23]:
```
v1
```

[23]:
```
'santd001'
```

[24]:
```
visit_id = []
home_id = []
```

[25]:
```
for i in range(1, 10):
    visit_id.append(gl.iloc[43]['visit_player_{}_id'.format(i)])
    home_id.append(gl.iloc[43]['home_player_{}_id'.format(i)])
```

[26]:
```
visit_id
```

[26]:
```
['aokin001',
 'panij002',
 'pagaa001',
 'poseb001',
 'crawb001',
 'mcgec001',
 'blang001',
 'ariaj001',
 'hudst001']
```

[27]:
```
gl.iloc[0]['year']
```

[27]:
```
2015
```

[28]:
```
visit = []
home = []
year = gl.iloc[43]['year']
for index in range(0, 9):
    vrid = visit_id[index]
#     vpos = 'pitching' if vrid == gl.iloc[0]['visit_sp_id'] else 'batting'
    vplayer = convert_single_player(vrid, year, 'batting')
    visit.append(vplayer)
    hrid = home_id[index]
#     hpos = 'pitching' if hrid == gl.iloc[0]['home_sp_id'] else 'batting'
    hplayer = convert_single_player(hrid, year, 'batting')
    home.append(hplayer)
```

[29]:
```
visit[0].shape
```

[29]:
```
(30,)
```

[30]:
```
home[0].shape
```

[30]:
```
(30,)
```

[31]:
```
gl.columns
```

[31]:
```
Index(['visit_team', 'home_team', 'visit_score', 'home_score',
       'game_length_outs', 'night_game', 'park_id', 'visit_manager_id',
       'home_manager_id', 'visit_sp_id', 'home_sp_id', 'visit_player_1_id',
       'visit_player_2_id', 'visit_player_3_id', 'visit_player_4_id',
       'visit_player_5_id', 'visit_player_6_id', 'visit_player_7_id',
       'visit_player_8_id', 'visit_player_9_id', 'home_player_1_id',
       'home_player_2_id', 'home_player_3_id', 'home_player_4_id',
       'home_player_5_id', 'home_player_6_id', 'home_player_7_id',
       'home_player_8_id', 'home_player_9_id', 'year', 'month', 'day',
       'home_win'],
```

```
      dtype='object')
```

[32]:
```
batters = visit + home
```

[33]:
```
dfb = pd.DataFrame(batters, columns=columns['batting'])
```

[34]:
```
dfb
```

[34]:

	weight	height	pos_1B	pos_2B	pos_3B	pos_C	pos_OF	pos_P	pos_SS	\
0	0.426230	0.30	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
1	0.508197	0.50	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
2	0.508197	0.55	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
3	0.549180	0.50	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
4	0.618852	0.55	0.0	0.0	0.0	0.0	0.0	0.0	1.0	
5	0.590164	0.50	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
6	0.454918	0.35	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
7	0.446721	0.50	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
8	0.405738	0.50	0.0	0.0	0.0	0.0	0.0	1.0	0.0	
9	0.528689	0.60	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
10	0.651639	0.45	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
11	0.610656	0.65	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
12	0.569672	0.50	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
13	0.590164	0.60	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
14	0.631148	0.50	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
15	0.569672	0.35	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
16	0.344262	0.15	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
17	0.528689	0.45	0.0	0.0	0.0	0.0	0.0	1.0	0.0	

	bats_L	...	BB	SO	IBB	HBP	SH	SF	\
0	1.0	...	0.091478	0.099345	0.004360	0.168421	0.111524	0.117188	
1	1.0	...	0.085614	0.097420	0.020349	0.073684	0.048327	0.203125	
2	0.0	...	0.124707	0.245668	0.026163	0.014035	0.078067	0.265625	
3	0.0	...	0.189210	0.244128	0.090116	0.147368	0.003717	0.398438	
4	1.0	...	0.155590	0.360801	0.091570	0.129825	0.022305	0.367188	
5	0.0	...	0.097342	0.199076	0.020349	0.028070	0.000000	0.242188	
6	1.0	...	0.141908	0.254524	0.030523	0.052632	0.096654	0.117188	
7	0.0	...	0.014464	0.057759	0.010174	0.028070	0.044610	0.070312	
8	0.0	...	0.010164	0.073161	0.000000	0.007018	0.249071	0.015625	
9	0.0	...	0.122361	0.322680	0.020349	0.045614	0.003717	0.156250	
10	0.0	...	0.076231	0.208317	0.014535	0.066667	0.007435	0.093750	
11	0.0	...	0.196638	0.616095	0.098837	0.091228	0.003717	0.562500	
12	0.0	...	0.284988	0.692337	0.071221	0.235088	0.011152	0.429688	
13	0.0	...	0.025020	0.125144	0.005814	0.031579	0.003717	0.078125	
14	1.0	...	0.143081	0.249519	0.042151	0.056140	0.003717	0.218750	
15	0.0	...	0.091087	0.243358	0.007267	0.059649	0.000000	0.171875	
16	1.0	...	0.042611	0.113593	0.018895	0.014035	0.085502	0.117188	
17	0.0	...	0.012901	0.059299	0.000000	0.003509	0.156134	0.015625	

```
        GIDP   NL      wRC+       WAR
0   0.124051  1.0  0.184353  0.105202
1   0.129114  1.0  0.177158  0.103468
2   0.136709  1.0  0.181655  0.157803
3   0.374684  1.0  0.205036  0.354335
4   0.235443  1.0  0.173561  0.172254
5   0.291139  1.0  0.171763  0.072832
6   0.088608  1.0  0.173561  0.101734
7   0.060759  1.0  0.158273  0.051445
8   0.030380  1.0  0.095324  0.058382
9   0.179747  1.0  0.186151  0.104624
10  0.106329  1.0  0.171763  0.105202
11  0.453165  1.0  0.198741  0.206358
12  0.313924  1.0  0.197842  0.262428
13  0.081013  1.0  0.158273  0.055491
14  0.237975  1.0  0.181655  0.080925
15  0.184810  1.0  0.179856  0.100000
16  0.081013  1.0  0.147482  0.036416
17  0.007595  1.0  0.097122  0.055491

[18 rows x 30 columns]
```

[35]:
```
btensor = [dfb, gl.iloc[43]['home_win']]
```

[36]:
```
# btensor
```

[37]:
```
gl.shape
```

[37]:
```
(2428, 33)
```

[38]:
```
gl.shape[0]
```

[38]:
```
2428
```

[39]:
```
players['batting']['players']['retroID'].str.contains('aardd001').sum() == 1
```

[39]:
```
True
```

Modular script to handle all gamelogs

[40]:
```
cols = list(columns['batting'].values) + ['Result']
for year in range(1919, 2020):
    df = pd.DataFrame()
    print('{}'.format(year))
#     gl = pd.read_csv('../core/data/retrosheet/gamelogs/GL{}.csv'.format(year))
#     for index in range(0, gl.shape[0]):
```

```
#         visit_id = []
#         home_id = []
#         for i in range(1, 10):
#             visit_id.append(gl.iloc[index]['visit_player_{}_id'.format(i)])
#             home_id.append(gl.iloc[index]['home_player_{}_id'.format(i)])
#         visit = []
#         home = []
#         for i in range(0, 9):
#             vrid = visit_id[i]
#             vplayer = convert_single_player(vrid, year, 'batting')
#             visit.append(vplayer)
#             hrid = home_id[i]
#             hplayer = convert_single_player(hrid, year, 'batting')
#             home.append(hplayer)
#         batters = list(np.append(np.array(visit + home).flatten(), gl.
 ↪iloc[index]['home_win']))
#         try:
#             bat_df = pd.DataFrame(batters)
#         except:
#             print('{0}\n{1}'.format(vrid))
#         df = df.append(bat_df.T)

#     if not os.path.exists('../core/tensors/games/'):
#         os.mkdir('../core/tensors/games/')
#     df.to_csv('../core/tensors/games/{0}.csv'.format(str(year)), index=False,␣
 ↪header=None)
```

1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986

```
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
```

[ ]:

Models

model_batting

April 29, 2020

[1]:
```
import tensorflow as tf
import pandas as pd
import numpy as np
```

[2]:
```
df = pd.read_csv('../core/output/batters.csv')
indexer = df.reset_index()[['index', 'retroID']].to_dict()['retroID']
y = df['Batting'].values
to_drop = ['debutYear', 'finalYear', 'G', '1B', 'AB', 'RBI', 'wOBA']
df.drop(columns=to_drop, inplace=True)
```

[3]:
```
df
```

[3]:

	retroID	weight	height	pos_1B	pos_2B	pos_3B	pos_C	pos_OF	\
0	aardd001	0.569672	0.60	0	0	0	0	0	
1	aaroh101	0.426230	0.45	0	0	0	0	1	
2	aarot101	0.467213	0.60	1	0	0	0	0	
3	aased001	0.467213	0.60	0	0	0	0	0	
4	abada001	0.442623	0.50	1	0	0	0	0	
...	...	...	...	...	...	...	...	...	
15288	zupcb001	0.590164	0.65	0	0	0	0	1	
15289	zupof101	0.434426	0.40	0	0	0	1	0	
15290	zuveg101	0.487705	0.65	0	0	0	0	0	
15291	zuvep001	0.397541	0.45	0	0	0	0	0	
15292	zycht001	0.467213	0.60	0	0	0	0	0	

	pos_P	pos_SS	...	SO	IBB	HBP	SH	SF	GIDP	NL	wRC+	WAR	\
0	1	0	...	2	0	0	1	0	0	1	-100	-0.1	
1	0	0	...	1383	293	32	21	121	328	1	153	136.3	
2	0	0	...	145	3	0	9	6	36	1	76	-1.7	
3	1	0	...	3	0	0	0	0	0	1	-100	-0.1	
4	0	0	...	5	0	0	0	0	1	1	0	-0.4	
...	...	...	...	...	...	...	..	...	...	..	...	...	
15288	0	0	...	137	3	6	20	8	15	0	74	-0.9	
15289	0	0	...	6	0	0	0	0	0	0	37	-0.2	
15290	1	0	...	39	0	0	16	0	3	1	0	-0.3	
15291	0	1	...	50	1	2	18	0	8	1	52	-2.2	
15292	1	0	...	0	0	0	0	0	0	0	0	0.0	

```
        Batting
0      0.000358
1      0.350195
2      0.157131
3      0.000358
4      0.090001
...         ...
15288  0.156062
15289  0.123425
15290  0.090088
15291  0.135118
15292  0.090195

[15293 rows x 31 columns]
```

Building the Model

[4]:
```
from sklearn.model_selection import train_test_split
```

[5]:
```
X = df.drop(columns=['Batting']).values
y = df[['retroID', 'Batting']].values
```

When we do our train-test split, since it's random in how it splits up the data, we need to keep track of the appropriate keys (retro IDs) for each data point.

[6]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=101)
X_train_keys = np.asarray([x[0] for x in X_train])
X_train = np.asarray([x[1:] for x in X_train])
X_test_keys = np.asarray([x[0] for x in X_test])
X_test = np.asarray([x[1:] for x in X_test])
y_train_keys = np.asarray([y[0] for y in y_train])
y_train = np.asarray([y[1] for y in y_train])
y_test_keys = np.asarray([y[0] for y in y_test])
y_test = np.asarray([y[1] for y in y_test])
```

[7]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras import regularizers
```

[8]:
```
X_train.shape
```

[8]:
```
(12234, 29)
```

[9]:
```
from sklearn.preprocessing import MinMaxScaler
```

[10]:
```
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[11]:
```
def to_tensor_input(player):
    return scaler.transform(player.values.reshape(-1,29))[0]
```

[12]:
```
tensor = df.drop(columns=['retroID', 'Batting'])
player_tensor_inputs = tensor.apply(lambda player: to_tensor_input(player),␣
 ↪axis=1)
```

[13]:
```
player_tensor_inputs
```

[13]:
```
0         [0.5696720000000001, 0.6, 0.0, 0.0, 0.0, 0.0, ...
1         [0.42623, 0.45, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, ...
2         [0.467213, 0.6, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
3         [0.467213, 0.6, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, ...
4         [0.44262299999999993, 0.5, 1.0, 0.0, 0.0, 0.0,...
                                ...
15288     [0.590164, 0.65, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0,...
15289     [0.43442600000000003, 0.4, 0.0, 0.0, 0.0, 1.0,...
15290     [0.487705, 0.65, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0,...
15291     [0.397541, 0.45, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,...
15292     [0.467213, 0.6, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, ...
Length: 15293, dtype: object
```

[14]:
```
tensor = pd.DataFrame(player_tensor_inputs.values.tolist())
```

[15]:
```
tensor.to_csv('../core/tensors/t_batting.csv', index=False, float_format='%g')
```

[27]:
```
epochs = 400
batch_size = 64
loss_param = 'mse'
optimizer_param = 'adam'
stop_monitor = 'val_loss'
stop_patience = 20
```

[28]:
```
model = Sequential()

model.add(Dense(116, activation='relu', kernel_regularizer=regularizers.l2(0.
 ↪0001)))
model.add(Dropout(0.5))

model.add(Dense(232, activation='relu', kernel_regularizer=regularizers.l2(0.
 ↪0001)))
model.add(Dropout(0.5))
```

```
model.add(Dense(116, activation='relu', kernel_regularizer=regularizers.l2(0.
↪0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```

[19]:
```
from tensorflow.keras.callbacks import EarlyStopping
```

[20]:
```
early_stop = EarlyStopping(monitor=stop_monitor, patience=stop_patience)
```

[29]:
```
results = model.fit(x=X_train, y=y_train,
                        epochs=epochs,
                        batch_size=batch_size,
                        validation_data=(X_test, y_test),
                        callbacks=[early_stop]
                    )
```

```
Train on 12234 samples, validate on 3059 samples
Epoch 1/400
12234/12234 [==============================] - 1s 88us/sample - loss: 0.0339 -
val_loss: 0.0177
Epoch 2/400
12234/12234 [==============================] - 1s 42us/sample - loss: 0.0144 -
val_loss: 0.0089
Epoch 3/400
12234/12234 [==============================] - 0s 40us/sample - loss: 0.0082 -
val_loss: 0.0050
Epoch 4/400
12234/12234 [==============================] - 0s 40us/sample - loss: 0.0053 -
val_loss: 0.0033
Epoch 5/400
12234/12234 [==============================] - 0s 39us/sample - loss: 0.0039 -
val_loss: 0.0026
Epoch 6/400
12234/12234 [==============================] - 0s 41us/sample - loss: 0.0032 -
val_loss: 0.0020
Epoch 7/400
12234/12234 [==============================] - 0s 39us/sample - loss: 0.0029 -
val_loss: 0.0018
Epoch 8/400
12234/12234 [==============================] - 0s 40us/sample - loss: 0.0026 -
val_loss: 0.0016
Epoch 9/400
12234/12234 [==============================] - 0s 40us/sample - loss: 0.0025 -
val_loss: 0.0017
```

```
Epoch 10/400
12234/12234 [==============================] - 0s 40us/sample - loss: 0.0024 -
val_loss: 0.0015
Epoch 11/400
12234/12234 [==============================] - 0s 41us/sample - loss: 0.0023 -
val_loss: 0.0015
Epoch 12/400
12234/12234 [==============================] - 0s 41us/sample - loss: 0.0022 -
val_loss: 0.0014
Epoch 13/400
12234/12234 [==============================] - 0s 40us/sample - loss: 0.0022 -
val_loss: 0.0014
Epoch 14/400
12234/12234 [==============================] - 1s 41us/sample - loss: 0.0022 -
val_loss: 0.0013
Epoch 15/400
12234/12234 [==============================] - 0s 40us/sample - loss: 0.0022 -
val_loss: 0.0013
Epoch 16/400
12234/12234 [==============================] - 1s 42us/sample - loss: 0.0021 -
val_loss: 0.0015
Epoch 17/400
12234/12234 [==============================] - 1s 42us/sample - loss: 0.0021 -
val_loss: 0.0014
Epoch 18/400
12234/12234 [==============================] - 1s 51us/sample - loss: 0.0020 -
val_loss: 0.0013
Epoch 19/400
12234/12234 [==============================] - 1s 55us/sample - loss: 0.0020 -
val_loss: 0.0013
Epoch 20/400
12234/12234 [==============================] - 1s 54us/sample - loss: 0.0019 -
val_loss: 0.0013
Epoch 21/400
12234/12234 [==============================] - 1s 49us/sample - loss: 0.0019 -
val_loss: 0.0013
Epoch 22/400
12234/12234 [==============================] - 1s 47us/sample - loss: 0.0019 -
val_loss: 0.0012
Epoch 23/400
12234/12234 [==============================] - 1s 47us/sample - loss: 0.0019 -
val_loss: 0.0013
Epoch 24/400
12234/12234 [==============================] - 1s 43us/sample - loss: 0.0019 -
val_loss: 0.0012
Epoch 25/400
12234/12234 [==============================] - 1s 42us/sample - loss: 0.0018 -
val_loss: 0.0012
```

```
Epoch 26/400
12234/12234 [==============================] - 0s 40us/sample - loss: 0.0018 -
val_loss: 0.0012
Epoch 27/400
12234/12234 [==============================] - 0s 41us/sample - loss: 0.0018 -
val_loss: 0.0013
Epoch 28/400
12234/12234 [==============================] - 0s 41us/sample - loss: 0.0018 -
val_loss: 0.0013
Epoch 29/400
12234/12234 [==============================] - 0s 41us/sample - loss: 0.0018 -
val_loss: 0.0012
Epoch 30/400
12234/12234 [==============================] - 0s 40us/sample - loss: 0.0018 -
val_loss: 0.0012
Epoch 31/400
12234/12234 [==============================] - 0s 41us/sample - loss: 0.0017 -
val_loss: 0.0012
Epoch 32/400
12234/12234 [==============================] - 0s 39us/sample - loss: 0.0017 -
val_loss: 0.0011
Epoch 33/400
12234/12234 [==============================] - 1s 43us/sample - loss: 0.0017 -
val_loss: 0.0012
Epoch 34/400
12234/12234 [==============================] - 0s 41us/sample - loss: 0.0017 -
val_loss: 0.0012
Epoch 35/400
12234/12234 [==============================] - 0s 40us/sample - loss: 0.0017 -
val_loss: 0.0012
Epoch 36/400
12234/12234 [==============================] - 1s 42us/sample - loss: 0.0017 -
val_loss: 0.0012
Epoch 37/400
12234/12234 [==============================] - 1s 49us/sample - loss: 0.0016 -
val_loss: 0.0013
Epoch 38/400
12234/12234 [==============================] - 1s 58us/sample - loss: 0.0016 -
val_loss: 0.0012
Epoch 39/400
12234/12234 [==============================] - 1s 56us/sample - loss: 0.0017 -
val_loss: 0.0013
Epoch 40/400
12234/12234 [==============================] - 1s 56us/sample - loss: 0.0017 -
val_loss: 0.0011
Epoch 41/400
12234/12234 [==============================] - 1s 54us/sample - loss: 0.0016 -
val_loss: 0.0012
```

```
Epoch 42/400
12234/12234 [==============================] - 1s 41us/sample - loss: 0.0017 -
val_loss: 0.0018
Epoch 43/400
12234/12234 [==============================] - 1s 52us/sample - loss: 0.0016 -
val_loss: 0.0012
Epoch 44/400
12234/12234 [==============================] - 1s 44us/sample - loss: 0.0016 -
val_loss: 0.0011
Epoch 45/400
12234/12234 [==============================] - 1s 46us/sample - loss: 0.0016 -
val_loss: 0.0011
Epoch 46/400
12234/12234 [==============================] - 1s 45us/sample - loss: 0.0016 -
val_loss: 0.0012
Epoch 47/400
12234/12234 [==============================] - 1s 42us/sample - loss: 0.0016 -
val_loss: 0.0011
Epoch 48/400
12234/12234 [==============================] - 1s 42us/sample - loss: 0.0017 -
val_loss: 0.0011
Epoch 49/400
12234/12234 [==============================] - 1s 43us/sample - loss: 0.0016 -
val_loss: 0.0011
Epoch 50/400
12234/12234 [==============================] - 1s 41us/sample - loss: 0.0016 -
val_loss: 0.0011
Epoch 51/400
12234/12234 [==============================] - 0s 40us/sample - loss: 0.0016 -
val_loss: 0.0013
Epoch 52/400
12234/12234 [==============================] - 1s 41us/sample - loss: 0.0016 -
val_loss: 0.0014
Epoch 53/400
12234/12234 [==============================] - 1s 41us/sample - loss: 0.0016 -
val_loss: 0.0011
Epoch 54/400
12234/12234 [==============================] - 1s 44us/sample - loss: 0.0016 -
val_loss: 0.0011
Epoch 55/400
12234/12234 [==============================] - 1s 53us/sample - loss: 0.0016 -
val_loss: 0.0012
Epoch 56/400
12234/12234 [==============================] - 1s 51us/sample - loss: 0.0016 -
val_loss: 0.0012
Epoch 57/400
12234/12234 [==============================] - 1s 53us/sample - loss: 0.0016 -
val_loss: 0.0011
```

```
Epoch 58/400
12234/12234 [==============================] - 1s 52us/sample - loss: 0.0015 -
val_loss: 0.0013
Epoch 59/400
12234/12234 [==============================] - 1s 54us/sample - loss: 0.0016 -
val_loss: 0.0012
Epoch 60/400
12234/12234 [==============================] - 1s 42us/sample - loss: 0.0015 -
val_loss: 0.0011
Epoch 61/400
12234/12234 [==============================] - 1s 48us/sample - loss: 0.0015 -
val_loss: 0.0010
Epoch 62/400
12234/12234 [==============================] - 0s 40us/sample - loss: 0.0015 -
val_loss: 0.0010
Epoch 63/400
12234/12234 [==============================] - 1s 41us/sample - loss: 0.0015 -
val_loss: 0.0011
Epoch 64/400
12234/12234 [==============================] - 1s 42us/sample - loss: 0.0015 -
val_loss: 0.0013
Epoch 65/400
12234/12234 [==============================] - 0s 41us/sample - loss: 0.0015 -
val_loss: 0.0010
Epoch 66/400
12234/12234 [==============================] - 1s 41us/sample - loss: 0.0015 -
val_loss: 0.0010
Epoch 67/400
12234/12234 [==============================] - 1s 41us/sample - loss: 0.0015 -
val_loss: 0.0012
Epoch 68/400
12234/12234 [==============================] - 1s 42us/sample - loss: 0.0016 -
val_loss: 0.0014
Epoch 69/400
12234/12234 [==============================] - 1s 43us/sample - loss: 0.0016 -
val_loss: 0.0010
Epoch 70/400
12234/12234 [==============================] - 0s 40us/sample - loss: 0.0015 -
val_loss: 0.0011
Epoch 71/400
12234/12234 [==============================] - 1s 41us/sample - loss: 0.0015 -
val_loss: 0.0014
Epoch 72/400
12234/12234 [==============================] - 1s 41us/sample - loss: 0.0016 -
val_loss: 0.0010
Epoch 73/400
12234/12234 [==============================] - 1s 55us/sample - loss: 0.0015 -
val_loss: 9.8471e-04
```

```
Epoch 74/400
12234/12234 [==============================] - 1s 55us/sample - loss: 0.0015 -
val_loss: 0.0011
Epoch 75/400
12234/12234 [==============================] - 1s 57us/sample - loss: 0.0015 -
val_loss: 9.6721e-04
Epoch 76/400
12234/12234 [==============================] - 1s 49us/sample - loss: 0.0015 -
val_loss: 0.0010
Epoch 77/400
12234/12234 [==============================] - 1s 54us/sample - loss: 0.0015 -
val_loss: 9.9823e-04
Epoch 78/400
12234/12234 [==============================] - 1s 49us/sample - loss: 0.0015 -
val_loss: 0.0010
Epoch 79/400
12234/12234 [==============================] - 1s 46us/sample - loss: 0.0016 -
val_loss: 0.0011
Epoch 80/400
12234/12234 [==============================] - 1s 42us/sample - loss: 0.0015 -
val_loss: 9.9099e-04
Epoch 81/400
12234/12234 [==============================] - 1s 44us/sample - loss: 0.0015 -
val_loss: 0.0010
Epoch 82/400
12234/12234 [==============================] - 1s 43us/sample - loss: 0.0015 -
val_loss: 0.0011
Epoch 83/400
12234/12234 [==============================] - 1s 42us/sample - loss: 0.0015 -
val_loss: 9.9834e-04
Epoch 84/400
12234/12234 [==============================] - 1s 64us/sample - loss: 0.0015 -
val_loss: 9.6191e-04
Epoch 85/400
12234/12234 [==============================] - 1s 42us/sample - loss: 0.0015 -
val_loss: 0.0010
Epoch 86/400
12234/12234 [==============================] - 1s 44us/sample - loss: 0.0015 -
val_loss: 0.0011
Epoch 87/400
12234/12234 [==============================] - 1s 41us/sample - loss: 0.0015 -
val_loss: 9.7191e-04
Epoch 88/400
12234/12234 [==============================] - 1s 44us/sample - loss: 0.0015 -
val_loss: 9.7572e-04
Epoch 89/400
12234/12234 [==============================] - 0s 41us/sample - loss: 0.0015 -
val_loss: 0.0012
```

```
Epoch 90/400
12234/12234 [==============================] - 1s 43us/sample - loss: 0.0015 -
val_loss: 0.0012
Epoch 91/400
12234/12234 [==============================] - 1s 55us/sample - loss: 0.0015 -
val_loss: 9.5506e-04
Epoch 92/400
12234/12234 [==============================] - 1s 57us/sample - loss: 0.0015 -
val_loss: 9.4303e-04
Epoch 93/400
12234/12234 [==============================] - 1s 56us/sample - loss: 0.0015 -
val_loss: 9.8456e-04
Epoch 94/400
12234/12234 [==============================] - 1s 56us/sample - loss: 0.0015 -
val_loss: 9.5695e-04
Epoch 95/400
12234/12234 [==============================] - 1s 55us/sample - loss: 0.0015 -
val_loss: 0.0010
Epoch 96/400
12234/12234 [==============================] - 1s 43us/sample - loss: 0.0015 -
val_loss: 9.7383e-04
Epoch 97/400
12234/12234 [==============================] - 1s 45us/sample - loss: 0.0015 -
val_loss: 0.0010
Epoch 98/400
12234/12234 [==============================] - 1s 43us/sample - loss: 0.0015 -
val_loss: 9.4741e-04
Epoch 99/400
12234/12234 [==============================] - 1s 42us/sample - loss: 0.0015 -
val_loss: 9.6006e-04
Epoch 100/400
12234/12234 [==============================] - 1s 43us/sample - loss: 0.0015 -
val_loss: 0.0010
Epoch 101/400
12234/12234 [==============================] - 1s 41us/sample - loss: 0.0015 -
val_loss: 9.5989e-04
Epoch 102/400
12234/12234 [==============================] - 1s 50us/sample - loss: 0.0015 -
val_loss: 9.6959e-04
Epoch 103/400
12234/12234 [==============================] - 1s 44us/sample - loss: 0.0015 -
val_loss: 9.6324e-04
Epoch 104/400
12234/12234 [==============================] - 0s 40us/sample - loss: 0.0016 -
val_loss: 9.4944e-04
Epoch 105/400
12234/12234 [==============================] - 1s 41us/sample - loss: 0.0014 -
val_loss: 9.5811e-04
```

```
Epoch 106/400
12234/12234 [==============================] - 1s 42us/sample - loss: 0.0014 -
val_loss: 0.0010
Epoch 107/400
12234/12234 [==============================] - 1s 43us/sample - loss: 0.0015 -
val_loss: 9.1667e-04
Epoch 108/400
12234/12234 [==============================] - 1s 47us/sample - loss: 0.0014 -
val_loss: 0.0011
Epoch 109/400
12234/12234 [==============================] - 1s 54us/sample - loss: 0.0015 -
val_loss: 9.7293e-04
Epoch 110/400
12234/12234 [==============================] - 1s 51us/sample - loss: 0.0014 -
val_loss: 9.5498e-04
Epoch 111/400
12234/12234 [==============================] - 1s 52us/sample - loss: 0.0014 -
val_loss: 9.5635e-04
Epoch 112/400
12234/12234 [==============================] - 1s 58us/sample - loss: 0.0015 -
val_loss: 9.4925e-04
Epoch 113/400
12234/12234 [==============================] - 1s 55us/sample - loss: 0.0015 -
val_loss: 0.0011
Epoch 114/400
12234/12234 [==============================] - 1s 48us/sample - loss: 0.0014 -
val_loss: 9.3379e-04
Epoch 115/400
12234/12234 [==============================] - 1s 48us/sample - loss: 0.0015 -
val_loss: 0.0014
Epoch 116/400
12234/12234 [==============================] - 1s 44us/sample - loss: 0.0015 -
val_loss: 9.9035e-04
Epoch 117/400
12234/12234 [==============================] - 1s 41us/sample - loss: 0.0014 -
val_loss: 0.0010
Epoch 118/400
12234/12234 [==============================] - 1s 43us/sample - loss: 0.0015 -
val_loss: 9.4646e-04
Epoch 119/400
12234/12234 [==============================] - 1s 42us/sample - loss: 0.0015 -
val_loss: 9.7571e-04
Epoch 120/400
12234/12234 [==============================] - 1s 42us/sample - loss: 0.0014 -
val_loss: 9.1578e-04
Epoch 121/400
12234/12234 [==============================] - 1s 44us/sample - loss: 0.0015 -
val_loss: 0.0012
```

```
Epoch 122/400
12234/12234 [==============================] - 1s 41us/sample - loss: 0.0014 -
val_loss: 9.1460e-04
Epoch 123/400
12234/12234 [==============================] - 1s 45us/sample - loss: 0.0015 -
val_loss: 0.0010
Epoch 124/400
12234/12234 [==============================] - 1s 41us/sample - loss: 0.0015 -
val_loss: 9.5794e-04
Epoch 125/400
12234/12234 [==============================] - 1s 46us/sample - loss: 0.0014 -
val_loss: 0.0010
Epoch 126/400
12234/12234 [==============================] - 1s 44us/sample - loss: 0.0014 -
val_loss: 9.2401e-04
Epoch 127/400
12234/12234 [==============================] - 1s 54us/sample - loss: 0.0014 -
val_loss: 9.5520e-04
Epoch 128/400
12234/12234 [==============================] - 1s 57us/sample - loss: 0.0014 -
val_loss: 9.2518e-04
Epoch 129/400
12234/12234 [==============================] - 1s 58us/sample - loss: 0.0014 -
val_loss: 9.1161e-04
Epoch 130/400
12234/12234 [==============================] - 1s 58us/sample - loss: 0.0014 -
val_loss: 9.1013e-04
Epoch 131/400
12234/12234 [==============================] - 1s 58us/sample - loss: 0.0015 -
val_loss: 9.9162e-04
Epoch 132/400
12234/12234 [==============================] - 1s 50us/sample - loss: 0.0015 -
val_loss: 9.1913e-04
Epoch 133/400
12234/12234 [==============================] - 1s 50us/sample - loss: 0.0015 -
val_loss: 9.8137e-04
Epoch 134/400
12234/12234 [==============================] - 1s 41us/sample - loss: 0.0014 -
val_loss: 9.1023e-04
Epoch 135/400
12234/12234 [==============================] - 1s 43us/sample - loss: 0.0015 -
val_loss: 9.3859e-04
Epoch 136/400
12234/12234 [==============================] - 0s 41us/sample - loss: 0.0014 -
val_loss: 8.9324e-04
Epoch 137/400
12234/12234 [==============================] - 1s 42us/sample - loss: 0.0014 -
val_loss: 9.4240e-04
```

```
Epoch 138/400
12234/12234 [==============================] - 1s 45us/sample - loss: 0.0014 -
val_loss: 9.0494e-04
Epoch 139/400
12234/12234 [==============================] - 1s 42us/sample - loss: 0.0014 -
val_loss: 0.0011
Epoch 140/400
12234/12234 [==============================] - 1s 43us/sample - loss: 0.0014 -
val_loss: 8.8544e-04
Epoch 141/400
12234/12234 [==============================] - 1s 42us/sample - loss: 0.0014 -
val_loss: 0.0011
Epoch 142/400
12234/12234 [==============================] - 1s 43us/sample - loss: 0.0014 -
val_loss: 8.8677e-04
Epoch 143/400
12234/12234 [==============================] - 1s 44us/sample - loss: 0.0014 -
val_loss: 0.0015
Epoch 144/400
12234/12234 [==============================] - 1s 44us/sample - loss: 0.0015 -
val_loss: 8.8369e-04
Epoch 145/400
12234/12234 [==============================] - 1s 54us/sample - loss: 0.0014 -
val_loss: 0.0015
Epoch 146/400
12234/12234 [==============================] - 1s 59us/sample - loss: 0.0015 -
val_loss: 9.7839e-04
Epoch 147/400
12234/12234 [==============================] - 1s 59us/sample - loss: 0.0015 -
val_loss: 0.0010
Epoch 148/400
12234/12234 [==============================] - 1s 64us/sample - loss: 0.0014 -
val_loss: 8.8733e-04
Epoch 149/400
12234/12234 [==============================] - 1s 62us/sample - loss: 0.0014 -
val_loss: 0.0010
Epoch 150/400
12234/12234 [==============================] - 1s 54us/sample - loss: 0.0014 -
val_loss: 9.7863e-04
Epoch 151/400
12234/12234 [==============================] - 1s 51us/sample - loss: 0.0014 -
val_loss: 9.1807e-04
Epoch 152/400
12234/12234 [==============================] - 1s 46us/sample - loss: 0.0014 -
val_loss: 8.7598e-04
Epoch 153/400
12234/12234 [==============================] - 1s 46us/sample - loss: 0.0014 -
val_loss: 9.9196e-04
```

```
Epoch 154/400
12234/12234 [==============================] - 1s 44us/sample - loss: 0.0014 -
val_loss: 0.0016
Epoch 155/400
12234/12234 [==============================] - 1s 45us/sample - loss: 0.0015 -
val_loss: 9.1860e-04
Epoch 156/400
12234/12234 [==============================] - 1s 44us/sample - loss: 0.0014 -
val_loss: 0.0011
Epoch 157/400
12234/12234 [==============================] - 1s 43us/sample - loss: 0.0014 -
val_loss: 9.0657e-04
Epoch 158/400
12234/12234 [==============================] - 1s 44us/sample - loss: 0.0014 -
val_loss: 9.2692e-04
Epoch 159/400
12234/12234 [==============================] - 1s 45us/sample - loss: 0.0014 -
val_loss: 9.2203e-04
Epoch 160/400
12234/12234 [==============================] - 1s 44us/sample - loss: 0.0014 -
val_loss: 9.2283e-04
Epoch 161/400
12234/12234 [==============================] - 1s 43us/sample - loss: 0.0014 -
val_loss: 9.1643e-04
Epoch 162/400
12234/12234 [==============================] - 1s 43us/sample - loss: 0.0014 -
val_loss: 9.0883e-04
Epoch 163/400
12234/12234 [==============================] - 1s 49us/sample - loss: 0.0014 -
val_loss: 0.0012
Epoch 164/400
12234/12234 [==============================] - 1s 55us/sample - loss: 0.0014 -
val_loss: 9.5491e-04
Epoch 165/400
12234/12234 [==============================] - 1s 63us/sample - loss: 0.0014 -
val_loss: 9.2393e-04
Epoch 166/400
12234/12234 [==============================] - 1s 57us/sample - loss: 0.0014 -
val_loss: 9.1901e-04
Epoch 167/400
12234/12234 [==============================] - 1s 59us/sample - loss: 0.0014 -
val_loss: 0.0014
Epoch 168/400
12234/12234 [==============================] - 1s 57us/sample - loss: 0.0015 -
val_loss: 8.8174e-04
Epoch 169/400
12234/12234 [==============================] - 1s 45us/sample - loss: 0.0014 -
val_loss: 8.5809e-04
```

```
Epoch 170/400
12234/12234 [==============================] - 1s 46us/sample - loss: 0.0014 -
val_loss: 9.0156e-04
Epoch 171/400
12234/12234 [==============================] - 1s 45us/sample - loss: 0.0014 -
val_loss: 0.0012
Epoch 172/400
12234/12234 [==============================] - 1s 45us/sample - loss: 0.0015 -
val_loss: 8.7756e-04
Epoch 173/400
12234/12234 [==============================] - 1s 54us/sample - loss: 0.0014 -
val_loss: 8.7620e-04
Epoch 174/400
12234/12234 [==============================] - 1s 55us/sample - loss: 0.0014 -
val_loss: 8.6786e-04
Epoch 175/400
12234/12234 [==============================] - 1s 61us/sample - loss: 0.0014 -
val_loss: 8.6015e-04
Epoch 176/400
12234/12234 [==============================] - 1s 46us/sample - loss: 0.0014 -
val_loss: 0.0010
Epoch 177/400
12234/12234 [==============================] - 1s 48us/sample - loss: 0.0014 -
val_loss: 9.9136e-04
Epoch 178/400
12234/12234 [==============================] - 1s 48us/sample - loss: 0.0015 -
val_loss: 9.1635e-04
Epoch 179/400
12234/12234 [==============================] - 1s 66us/sample - loss: 0.0014 -
val_loss: 8.6218e-04
Epoch 180/400
12234/12234 [==============================] - 1s 78us/sample - loss: 0.0014 -
val_loss: 8.4677e-04
Epoch 181/400
12234/12234 [==============================] - 1s 74us/sample - loss: 0.0014 -
val_loss: 9.0584e-04
Epoch 182/400
12234/12234 [==============================] - 1s 78us/sample - loss: 0.0014 -
val_loss: 9.9842e-04
Epoch 183/400
12234/12234 [==============================] - 1s 84us/sample - loss: 0.0014 -
val_loss: 0.0015
Epoch 184/400
12234/12234 [==============================] - 1s 68us/sample - loss: 0.0015 -
val_loss: 9.1945e-04
Epoch 185/400
12234/12234 [==============================] - 1s 58us/sample - loss: 0.0014 -
val_loss: 9.9103e-04
```

```
Epoch 186/400
12234/12234 [==============================] - 1s 57us/sample - loss: 0.0014 -
val_loss: 0.0015
Epoch 187/400
12234/12234 [==============================] - 1s 54us/sample - loss: 0.0014 -
val_loss: 8.5842e-04
Epoch 188/400
12234/12234 [==============================] - 1s 46us/sample - loss: 0.0014 -
val_loss: 8.9506e-04
Epoch 189/400
12234/12234 [==============================] - 1s 48us/sample - loss: 0.0014 -
val_loss: 8.5331e-04
Epoch 190/400
12234/12234 [==============================] - 1s 42us/sample - loss: 0.0014 -
val_loss: 8.8959e-04
Epoch 191/400
12234/12234 [==============================] - 1s 42us/sample - loss: 0.0014 -
val_loss: 8.8742e-04
Epoch 192/400
12234/12234 [==============================] - 1s 44us/sample - loss: 0.0014 -
val_loss: 9.0049e-04
Epoch 193/400
12234/12234 [==============================] - 1s 49us/sample - loss: 0.0014 -
val_loss: 9.6137e-04
Epoch 194/400
12234/12234 [==============================] - 1s 62us/sample - loss: 0.0014 -
val_loss: 9.6924e-04
Epoch 195/400
12234/12234 [==============================] - 1s 77us/sample - loss: 0.0014 -
val_loss: 8.5014e-04
Epoch 196/400
12234/12234 [==============================] - 1s 68us/sample - loss: 0.0014 -
val_loss: 0.0012
Epoch 197/400
12234/12234 [==============================] - 1s 61us/sample - loss: 0.0015 -
val_loss: 9.4546e-04
Epoch 198/400
12234/12234 [==============================] - 1s 57us/sample - loss: 0.0014 -
val_loss: 8.4731e-04
Epoch 199/400
12234/12234 [==============================] - 1s 49us/sample - loss: 0.0014 -
val_loss: 9.2903e-04
Epoch 200/400
12234/12234 [==============================] - 1s 48us/sample - loss: 0.0014 -
val_loss: 8.8370e-04
```

[30]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              multiple                  3480
_________________________________________________________________
dropout_3 (Dropout)          multiple                  0
_________________________________________________________________
dense_5 (Dense)              multiple                  27144
_________________________________________________________________
dropout_4 (Dropout)          multiple                  0
_________________________________________________________________
dense_6 (Dense)              multiple                  27028
_________________________________________________________________
dropout_5 (Dropout)          multiple                  0
_________________________________________________________________
dense_7 (Dense)              multiple                  117
=================================================================
Total params: 57,769
Trainable params: 57,769
Non-trainable params: 0
_________________________________________________________________
```

[31]:
```
import os
```

[32]:
```
losses = model.history.history
losses['loss'] = np.asarray(losses['loss'])
losses['val_loss'] = np.asarray(losses['val_loss'])
final_number_of_epochs = len(losses['loss'])
min_loss = losses['loss'].min()
mean_loss = losses['loss'].mean()
final_loss = losses['loss'][-1]
min_val_loss = losses['val_loss'].min()
mean_val_loss = losses['val_loss'].mean()
final_val_loss = losses['val_loss'][-1]

def get_model_summary():
    output = []
    model.summary(print_fn=lambda line: output.append(line))
    return str(output).strip('[]')

summary = get_model_summary()

record = {
    'Epochs': final_number_of_epochs,
    'Batch_Size': batch_size,
    'Loss_Func': loss_param,
```

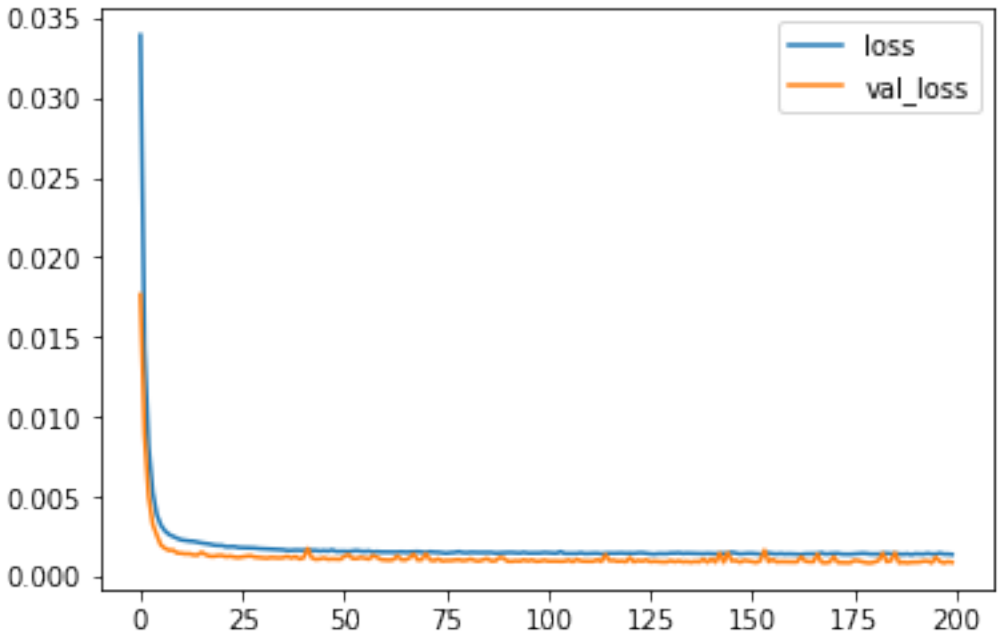
```
    'Optimizer': optimizer_param,
    'Early_Stop_Monitor': stop_monitor,
    'Early_Stop_Patience': stop_patience,
    'Min_Loss': min_loss,
    'Mean_Loss': mean_loss,
    'Final_Loss': final_loss,
    'Min_Val_Loss': min_val_loss,
    'Mean_Val_Loss': mean_val_loss,
    'Final_Val_Loss': final_val_loss,
    'Model': summary
}

new_data = pd.DataFrame(record, index=[0])

if os.path.exists('../core/records/batting_results.csv'):
    df_records = pd.read_csv('../core/records/batting_results.csv')
    df_records = df_records.append(new_data)
else:
    df_records = pd.DataFrame(new_data)

df_records.to_csv('../core/records/batting_results.csv', index=False,␣
 ↪float_format='%g')
```

Model Evaluation

[33]:
```
losses = pd.DataFrame(model.history.history)
```

[34]:
```
losses.plot()
```

[34]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x145a367d0>
```



[61]:
```
predictions = model.predict(X_test)
predictions = [pred for sublist in predictions for pred in sublist]
```

[29]:
```
test_player_ratings = dict(zip(X_test_keys, predictions))
```

[30]:
```
player_key = df['retroID']
```

[31]:
```
player_key
```

[31]:
```
0        aardd001
1        aaroh101
2        aarot101
3        aased001
4        abada001
           ...
15288    zupcb001
15289    zupof101
15290    zuveg101
15291    zuvep001
15292    zycht001
Name: retroID, Length: 15293, dtype: object
```

[32]:
```
results = model.predict(tensor.to_numpy())
```

[33]:
```
len(results)
```

[33]: 15293

[34]:
```
results.mean()
```

[34]: 0.11595928

[35]:
```
df['Batting'].shape
```

[35]: (15293,)

[36]:
```
results.shape
```

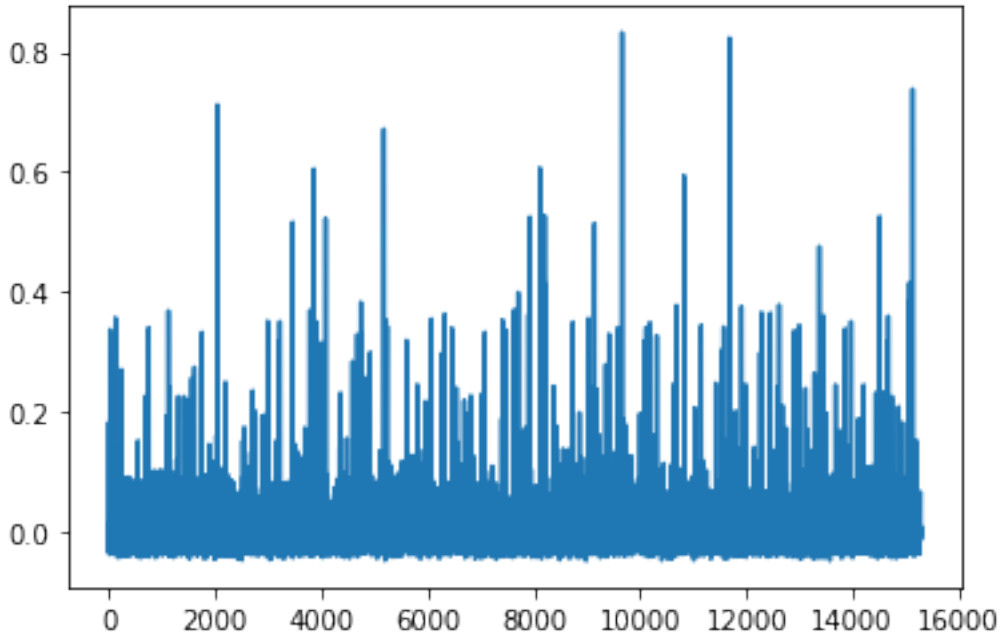
[36]: (15293, 1)

[37]:
```
results = [pred for sublist in results for pred in sublist]
```

[38]:
```
diff = df['Batting'] - results
```

[39]:
```
diff.plot()
```

[39]: <matplotlib.axes._subplots.AxesSubplot at 0x145c29890>



[40]:
```
diff.mean()
```

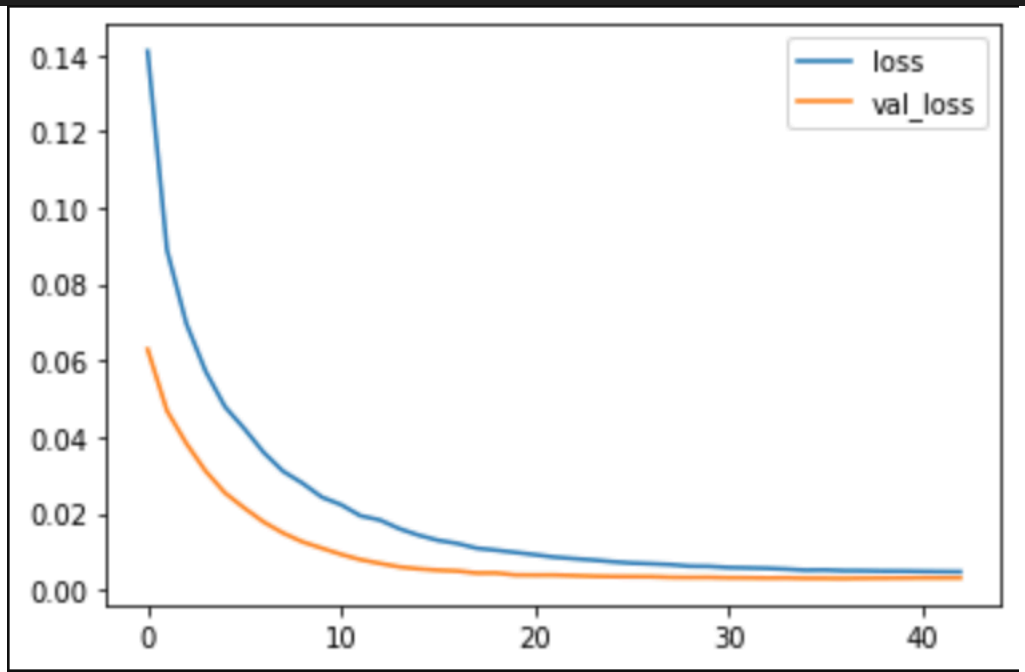
[40]: 0.005629653826018003

[ ]:

Model 1

Mean Loss:	0.0198435
Mean Validation Loss:	0.00966795
Batch Size:	128

```
model = Sequential()

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(116, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(232, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(116, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(14, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(7, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



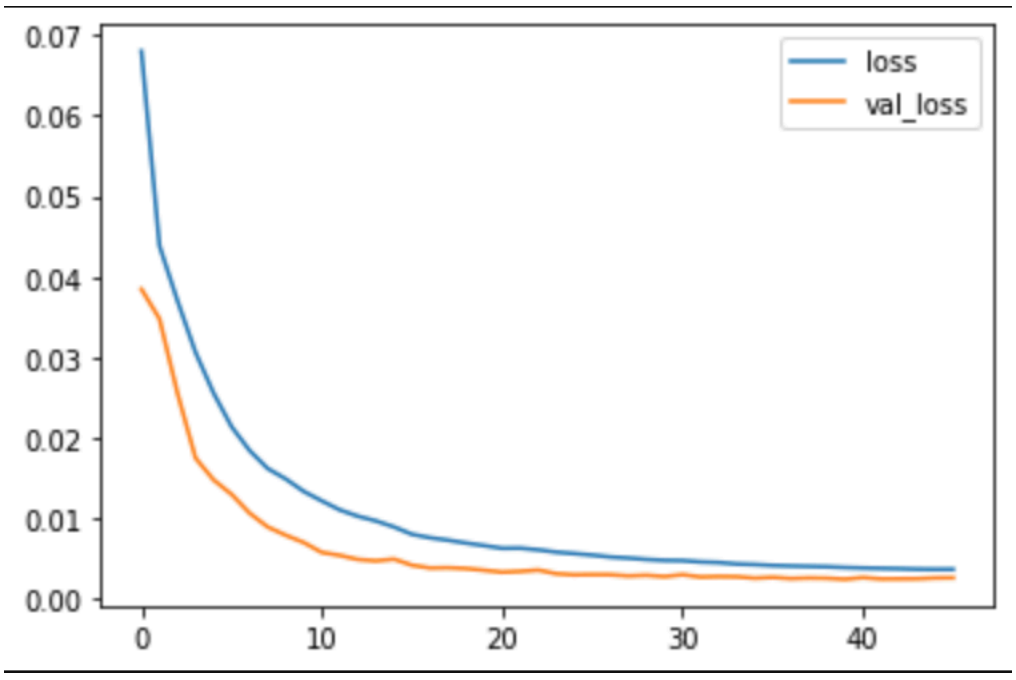
Model 2

Mean Loss: 0.0107984

Mean Validation Loss: 0.00640694

Batch Size: 128

```
model = Sequential()

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(116, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(14, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(7, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



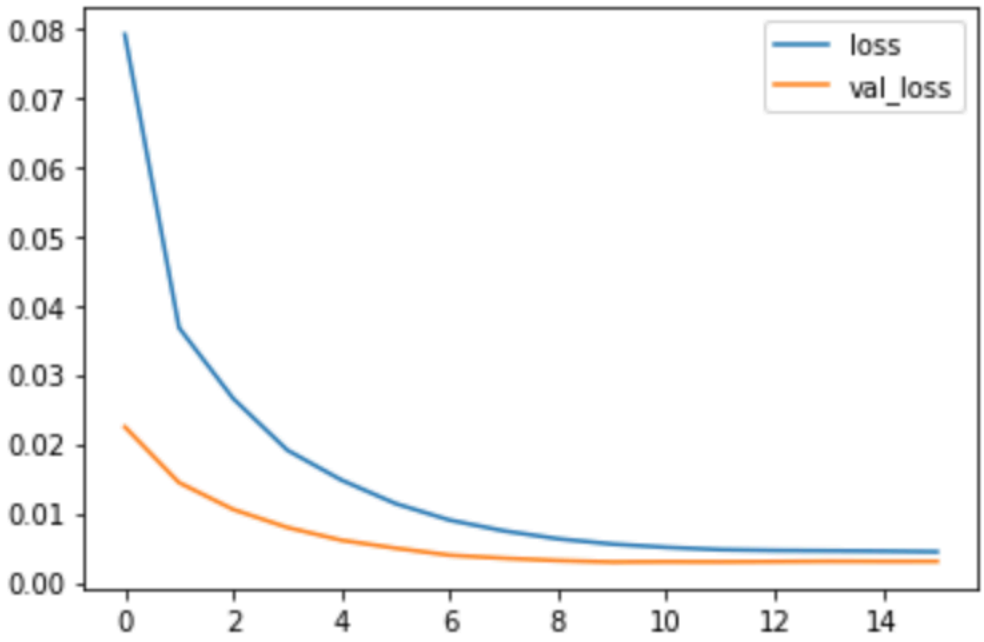
Model 3

Mean Loss: 0.0152167

Mean Validation Loss: 0.00610194

Batch Size: 32

```
model = Sequential()

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(116, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(14, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(7, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



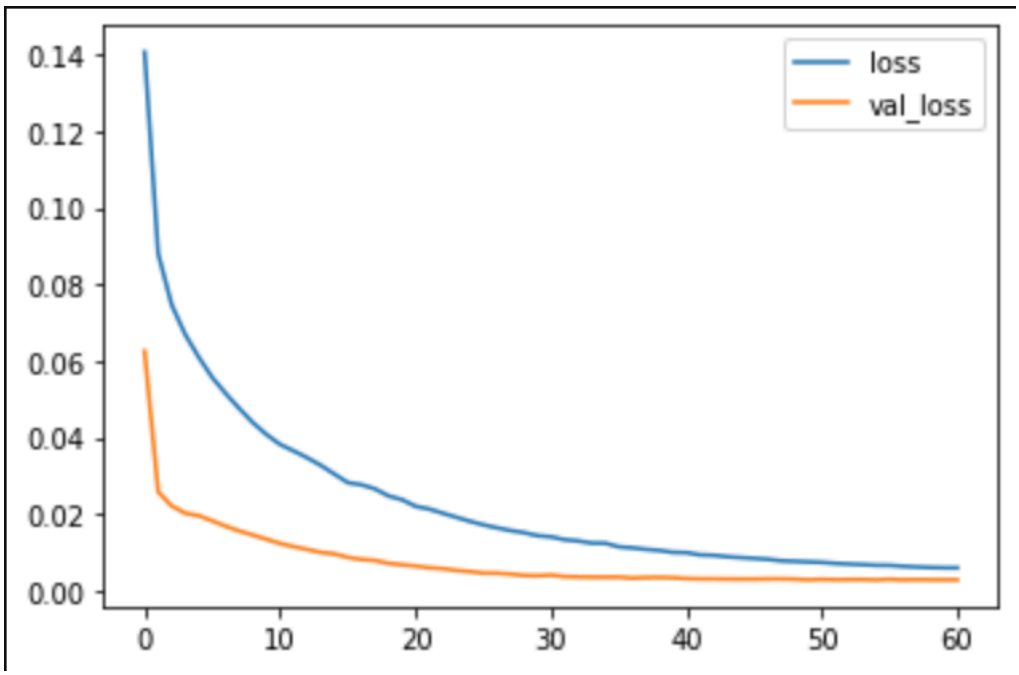
Model 4

Mean Loss: 0.0234176

Mean Validation Loss: 0.00764555

Batch Size: 256

```
model = Sequential()

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(116, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(14, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(7, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



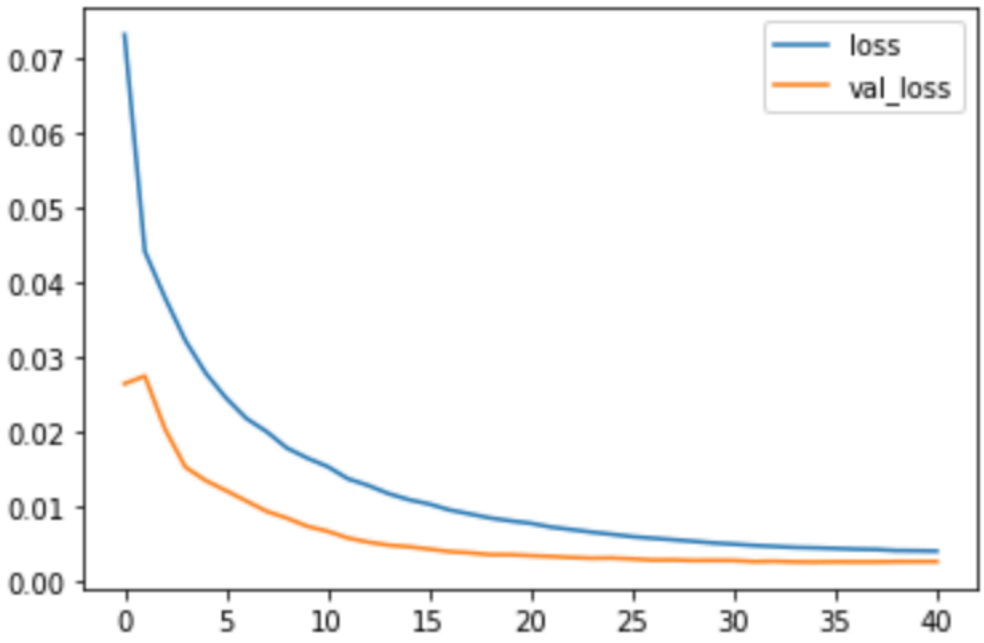
Model 5

Mean Loss: 0.0130172

Mean Validation Loss: 0.00618403

Batch Size: 128

```
model = Sequential()

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(14, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(7, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



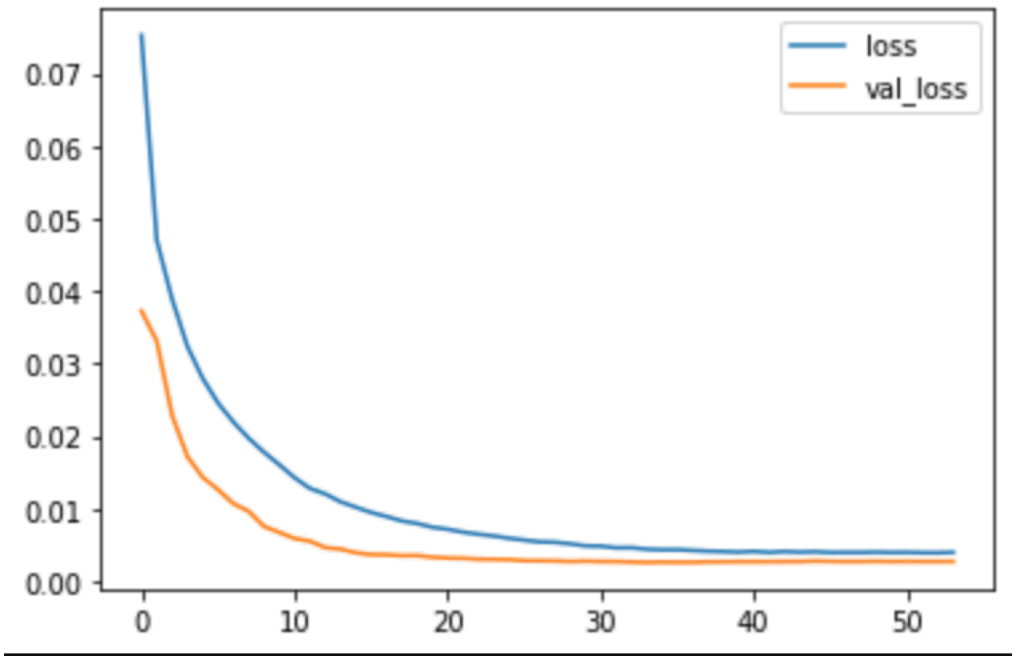
Model 6

Mean Loss: 0.0107793

Mean Validation Loss: 0.00569587

Batch Size: 128

```
model = Sequential()

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(116, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(14, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



Model 7

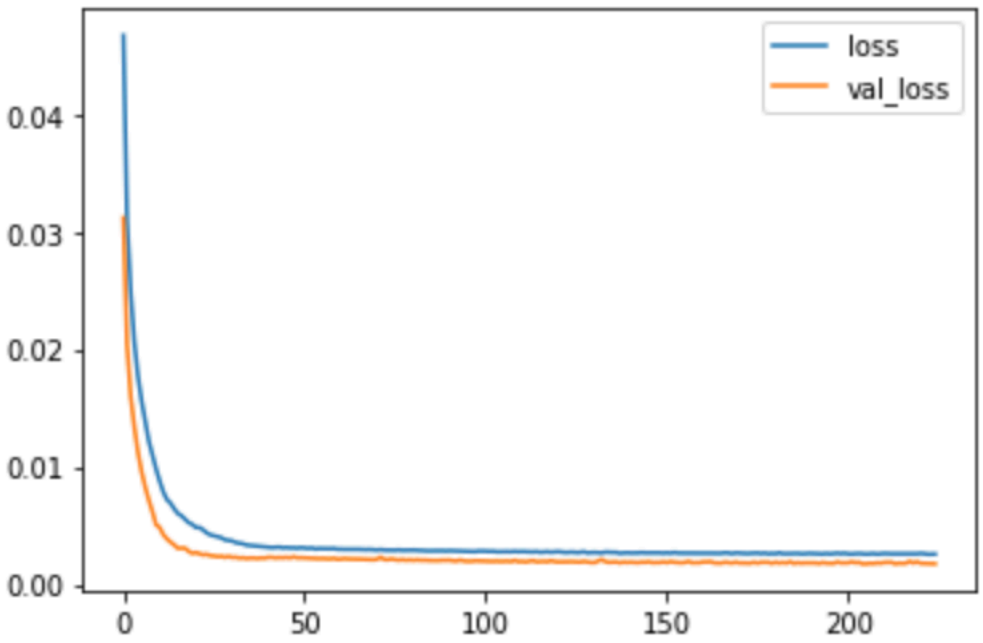
Mean Loss: 0.00380475

Mean Validation Loss: 0.00254377

Batch Size: 128

Early Stop Patience raised to 20

```
model = Sequential()

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(116, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



Model 8

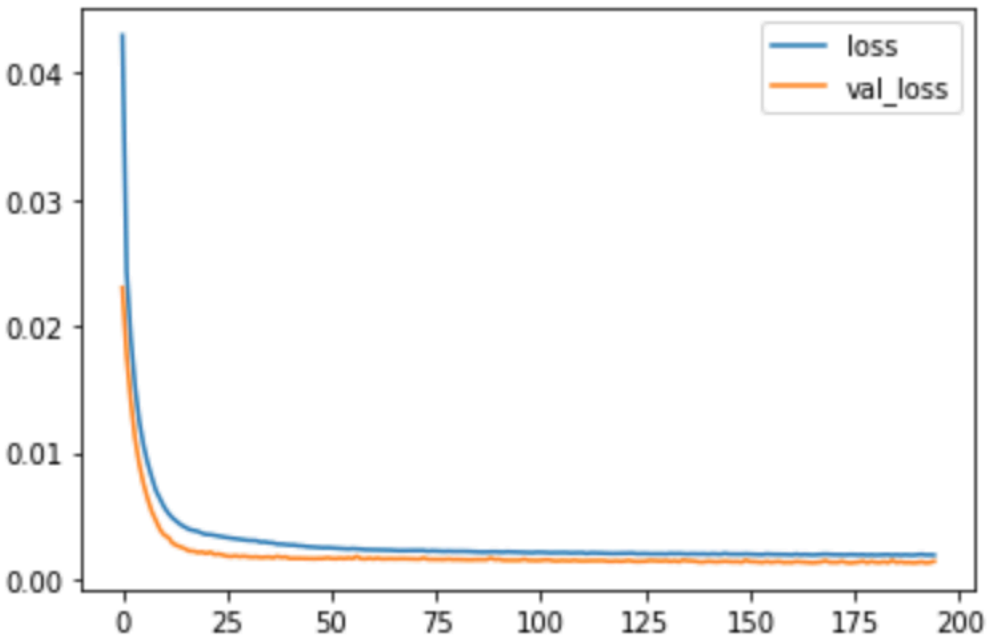
Mean Loss: 0.00300474

Mean Validation Loss: 0.00200529

Batch Size: 128

Early Stop Patience raised to 20

```
model = Sequential()

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(116, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



Model 9

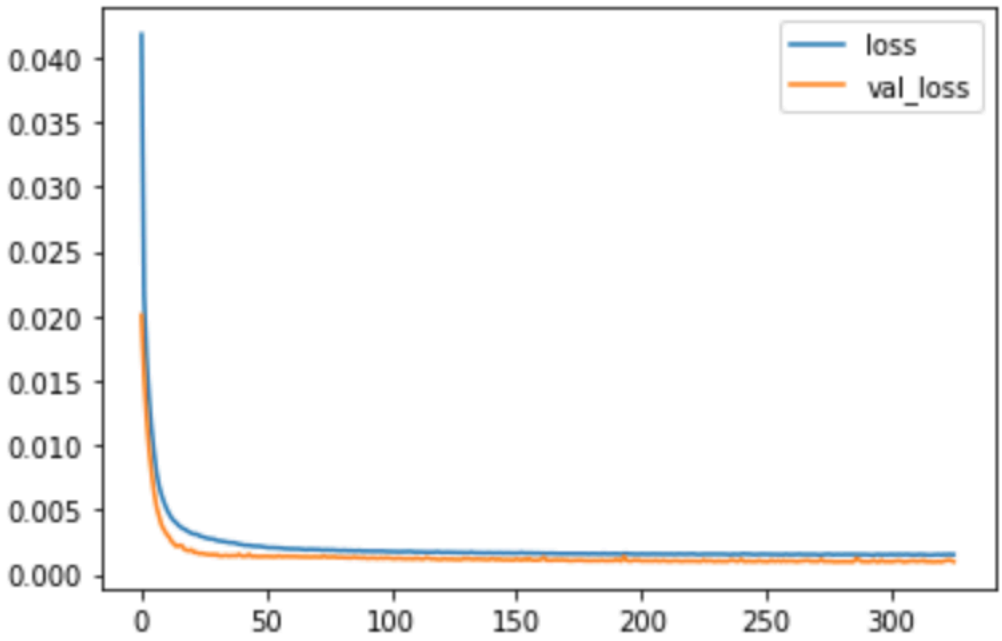
Mean Loss: 0.00218654

Mean Validation Loss: 0.00143522

Batch Size: 128

Early Stop Patience raised to 20

```
model = Sequential()

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(116, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



Model 10

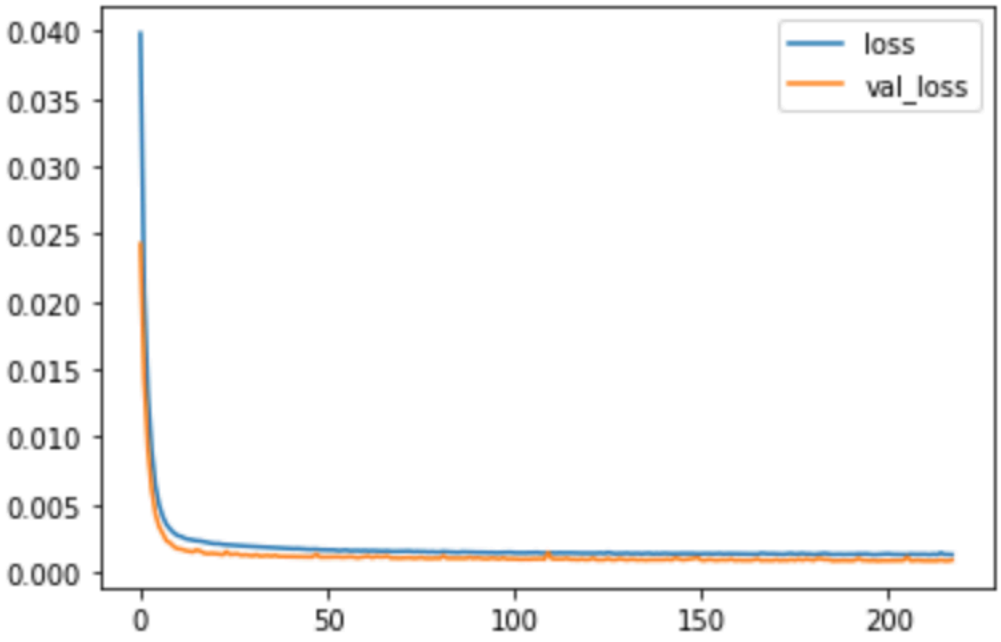
Mean Loss: 0.00193549

Mean Validation Loss: 0.00129605

Batch Size: 128

Early Stop Patience raised to 20

```
model = Sequential()

model.add(Dense(116, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(232, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(116, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



Model 11

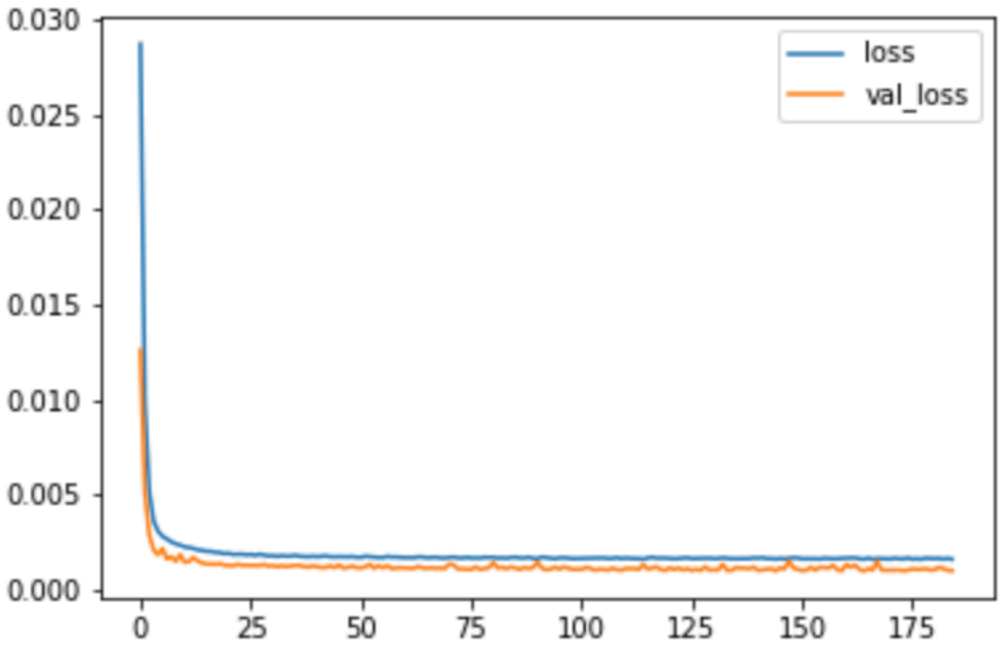
Mean Loss: 0.00192912

Mean Validation Loss: 0.00125362

Batch Size: 32

The same as model 10 with a smaller batch size

```
model = Sequential()

model.add(Dense(116, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(232, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(116, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



Model 12

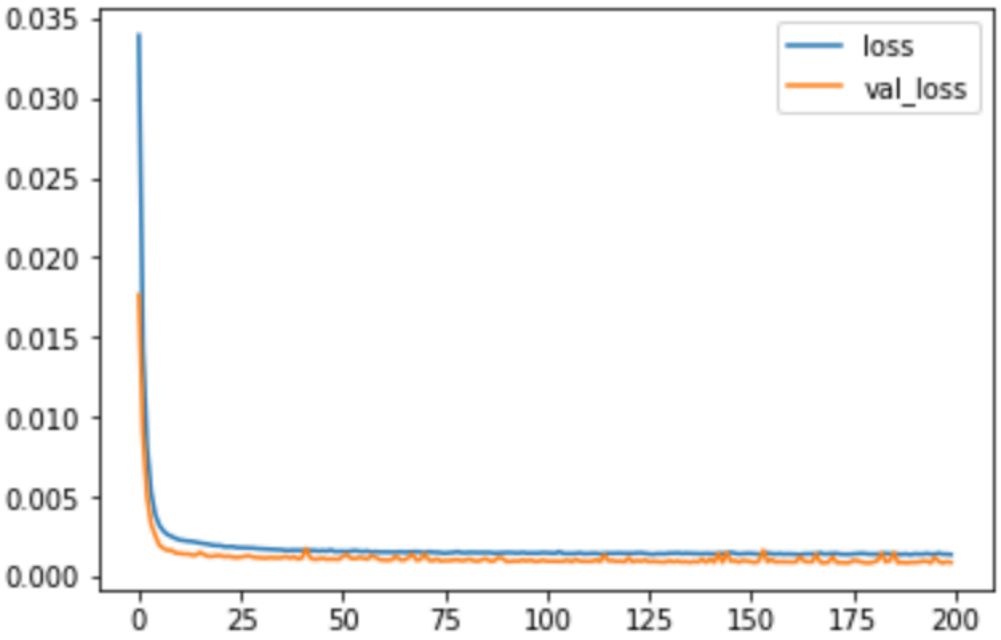
Mean Loss: 0.0018555

Mean Validation Loss: 0.00124253

Batch Size: 64

Split the difference on batch size between 10 and 11

```
model = Sequential()

model.add(Dense(116, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(232, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(116, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



model_pitching

April 30, 2020

[1]:
```
import tensorflow as tf
import pandas as pd
import numpy as np
```

[2]:
```
df = pd.read_csv('../core/output/pitchers.csv')
indexer = df.reset_index()[['index', 'retroID']].to_dict()['retroID']
y = df['Pitching'].values
```

[3]:
```
df
```

[3]:

	retroID	BAOpp	CG	SHO	IPouts	H	ER	HR	BB	SO	...	\
0	aardd001	0.2574	0	0	1011	296	160	41	183	340	...	
1	aased001	0.2508	22	5	3328	1085	468	89	457	641	...	
2	abadf001	0.2447	0	0	992	309	135	42	116	280	...	
3	abbog001	0.2786	37	5	3858	1405	627	162	352	484	...	
4	abboj001	0.2804	31	6	5022	1779	791	154	620	888	...	
...	...	...	..	...	...	...	...	...	...	...	...	
8020	zolds101	0.2700	30	5	2788	956	366	54	301	207	...	
8021	zubeb101	0.2717	23	3	2358	767	374	35	468	383	...	
8022	zumaj001	0.2286	0	0	629	169	71	18	114	210	...	
8023	zuveg101	0.2760	9	2	1927	660	253	56	203	223	...	
8024	zycht001	0.2183	0	0	218	57	22	3	34	80	...	

	IP	K/9	BB/9	HR/9	BABIP	LOB%	ERA	FIP	WAR	Pitching
0	0.062360	9.08	4.89	1.09	0.285	74.5	4.27	4.45	1.1	0.602913
1	0.205233	5.20	3.71	0.72	0.282	73.4	3.80	3.85	11.7	0.636924
2	0.061102	7.62	3.16	1.14	0.281	77.7	3.67	4.24	0.6	0.603736
3	0.237967	3.39	2.46	1.13	0.278	69.3	4.39	4.46	10.2	0.628847
4	0.309765	4.77	3.33	0.83	0.295	70.0	4.25	4.25	22.7	0.666725
...	...	...	...	...	...	...	...	...	...	...
8020	0.171925	2.00	2.91	0.52	0.267	70.7	3.54	3.80	9.3	0.630540
8021	0.145445	4.39	5.36	0.40	0.283	69.0	4.28	3.96	3.3	0.610437
8022	0.038711	9.01	4.89	0.77	0.267	78.7	3.00	3.94	2.7	0.612847
8023	0.118817	3.12	2.84	0.78	0.270	73.2	3.54	3.93	1.9	0.608497
8024	0.013360	9.91	4.21	0.37	0.293	79.1	2.72	3.22	1.1	0.611166

[8025 rows x 31 columns]

Building the Model

[4]:
```
from sklearn.model_selection import train_test_split
```

[5]:
```
X = df.drop(columns=['Pitching']).values
y = df[['retroID', 'Pitching']].values
```

When we do our train-test split, since it's random in how it splits up the data, we need to keep track of the appropriate keys (retro IDs) for each data point.

[6]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=101)
X_train_keys = np.asarray([x[0] for x in X_train])
X_train = np.asarray([x[1:] for x in X_train])
X_test_keys = np.asarray([x[0] for x in X_test])
X_test = np.asarray([x[1:] for x in X_test])
y_train_keys = np.asarray([y[0] for y in y_train])
y_train = np.asarray([y[1] for y in y_train])
y_test_keys = np.asarray([y[0] for y in y_test])
y_test = np.asarray([y[1] for y in y_test])
```

[7]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras import regularizers
```

[8]:
```
X_train.shape
```

[8]:
```
(6420, 29)
```

[9]:
```
from sklearn.preprocessing import MinMaxScaler
```

[10]:
```
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[11]:
```
def to_tensor_input(player):
    return scaler.transform(player.values.reshape(-1,29))[0]
```

[12]:
```
tensor = df.drop(columns=['retroID', 'Pitching'])
player_tensor_inputs = tensor.apply(lambda player: to_tensor_input(player),␣
 ↪axis=1)
```

[13]:
```
player_tensor_inputs
```

[13]:
```
0       [0.2574, 0.0, 0.0, 0.06256962495358337, 0.0599...
1       [0.2508, 0.06567164179104477, 0.08196721311475...
```

```
2       [0.2447, 0.0, 0.0, 0.06139373684861988, 0.0625...
3       [0.2786, 0.11044776119402985, 0.08196721311475...
4       [0.2804, 0.09253731343283582, 0.09836065573770...
                              ...
8020    [0.27, 0.08955223880597014, 0.0819672131147541...
8021    [0.2717, 0.06865671641791045, 0.04918032786885...
8022    [0.2286, 0.0, 0.0, 0.03892808515905434, 0.0342...
8023    [0.276, 0.026865671641791045, 0.03278688524590...
8024    [0.2183, 0.0, 0.0, 0.013491768783265256, 0.011...
Length: 8025, dtype: object
```

[14]:
```
tensor = pd.DataFrame(player_tensor_inputs.values.tolist())
```

[15]:
```
tensor.to_csv('../core/tensors/t_pitching.csv', index=False, float_format='%g')
```

[17]:
```
epochs = 4000
batch_size = 32
loss_param = 'mse'
optimizer_param = 'adam'
stop_monitor = 'val_loss'
stop_patience = 50
```

[18]:
```
from tensorflow.keras.callbacks import EarlyStopping
```

[19]:
```
early_stop = EarlyStopping(monitor=stop_monitor, patience=stop_patience)
```

[20]:
```
model = Sequential()

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.
 ↪0001)))
model.add(Dropout(0.5))

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.
 ↪0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```

[21]:
```
results = model.fit(x=X_train, y=y_train,
                        epochs=epochs,
                        batch_size=batch_size,
                        validation_data=(X_test, y_test),
                        callbacks=[early_stop]
                    )
```

```
Train on 6420 samples, validate on 1605 samples
Epoch 1/4000
6420/6420 [==============================] - 1s 183us/sample - loss: 0.0069 -
val_loss: 0.0035
Epoch 2/4000
6420/6420 [==============================] - 0s 66us/sample - loss: 0.0033 -
val_loss: 0.0021
Epoch 3/4000
6420/6420 [==============================] - 0s 51us/sample - loss: 0.0021 -
val_loss: 0.0013
Epoch 4/4000
6420/6420 [==============================] - 0s 49us/sample - loss: 0.0015 -
val_loss: 8.8244e-04
Epoch 5/4000
6420/6420 [==============================] - 0s 46us/sample - loss: 0.0011 -
val_loss: 6.9195e-04
Epoch 6/4000
6420/6420 [==============================] - 0s 50us/sample - loss: 9.0205e-04 -
val_loss: 5.2742e-04
Epoch 7/4000
6420/6420 [==============================] - 0s 52us/sample - loss: 7.3946e-04 -
val_loss: 4.4965e-04
Epoch 8/4000
6420/6420 [==============================] - 0s 49us/sample - loss: 6.7641e-04 -
val_loss: 4.5362e-04
Epoch 9/4000
6420/6420 [==============================] - 0s 53us/sample - loss: 6.3987e-04 -
val_loss: 3.7094e-04
Epoch 10/4000
6420/6420 [==============================] - 0s 52us/sample - loss: 5.5052e-04 -
val_loss: 3.7966e-04
Epoch 11/4000
6420/6420 [==============================] - 0s 61us/sample - loss: 5.8589e-04 -
val_loss: 3.5014e-04
Epoch 12/4000
6420/6420 [==============================] - 0s 52us/sample - loss: 5.5244e-04 -
val_loss: 3.2501e-04
Epoch 13/4000
6420/6420 [==============================] - 0s 76us/sample - loss: 5.2412e-04 -
val_loss: 3.1699e-04
Epoch 14/4000
6420/6420 [==============================] - 0s 50us/sample - loss: 5.2772e-04 -
val_loss: 3.1689e-04
Epoch 15/4000
6420/6420 [==============================] - 0s 54us/sample - loss: 5.1187e-04 -
val_loss: 3.1092e-04
Epoch 16/4000
6420/6420 [==============================] - 0s 53us/sample - loss: 4.7594e-04 -
```

```
val_loss: 3.1406e-04
Epoch 17/4000
6420/6420 [==============================] - 0s 68us/sample - loss: 4.8153e-04 -
val_loss: 2.9968e-04
Epoch 18/4000
6420/6420 [==============================] - 0s 56us/sample - loss: 4.5966e-04 -
val_loss: 2.9325e-04
Epoch 19/4000
6420/6420 [==============================] - 0s 73us/sample - loss: 5.0300e-04 -
val_loss: 2.7578e-04
Epoch 20/4000
6420/6420 [==============================] - 0s 72us/sample - loss: 4.9546e-04 -
val_loss: 3.3074e-04
Epoch 21/4000
6420/6420 [==============================] - 0s 43us/sample - loss: 4.5345e-04 -
val_loss: 3.0079e-04
Epoch 22/4000
6420/6420 [==============================] - 0s 46us/sample - loss: 4.8102e-04 -
val_loss: 2.7513e-04
Epoch 23/4000
6420/6420 [==============================] - 0s 64us/sample - loss: 4.5643e-04 -
val_loss: 2.7033e-04
Epoch 24/4000
6420/6420 [==============================] - 0s 57us/sample - loss: 4.5910e-04 -
val_loss: 2.9084e-04
Epoch 25/4000
6420/6420 [==============================] - 0s 71us/sample - loss: 4.3354e-04 -
val_loss: 2.6616e-04
Epoch 26/4000
6420/6420 [==============================] - 0s 64us/sample - loss: 4.9588e-04 -
val_loss: 2.7724e-04
Epoch 27/4000
6420/6420 [==============================] - 0s 53us/sample - loss: 4.5755e-04 -
val_loss: 2.5632e-04
Epoch 28/4000
6420/6420 [==============================] - 0s 75us/sample - loss: 4.2857e-04 -
val_loss: 2.6314e-04
Epoch 29/4000
6420/6420 [==============================] - 1s 92us/sample - loss: 4.6816e-04 -
val_loss: 2.4323e-04
Epoch 30/4000
6420/6420 [==============================] - 0s 72us/sample - loss: 4.1988e-04 -
val_loss: 2.4475e-04
Epoch 31/4000
6420/6420 [==============================] - 1s 85us/sample - loss: 4.2199e-04 -
val_loss: 2.6340e-04
Epoch 32/4000
6420/6420 [==============================] - 1s 88us/sample - loss: 4.2470e-04 -
```

```
val_loss: 2.5903e-04
Epoch 33/4000
6420/6420 [==============================] - 1s 104us/sample - loss: 4.1919e-04
- val_loss: 2.5305e-04
Epoch 34/4000
6420/6420 [==============================] - 1s 85us/sample - loss: 4.3582e-04 -
val_loss: 2.4753e-04
Epoch 35/4000
6420/6420 [==============================] - 0s 58us/sample - loss: 4.2053e-04 -
val_loss: 2.2769e-04
Epoch 36/4000
6420/6420 [==============================] - 0s 59us/sample - loss: 4.4114e-04 -
val_loss: 2.3009e-04
Epoch 37/4000
6420/6420 [==============================] - 0s 57us/sample - loss: 4.1271e-04 -
val_loss: 2.6178e-04
Epoch 38/4000
6420/6420 [==============================] - 0s 54us/sample - loss: 3.6627e-04 -
val_loss: 2.3378e-04
Epoch 39/4000
6420/6420 [==============================] - 0s 63us/sample - loss: 4.1566e-04 -
val_loss: 2.2211e-04
Epoch 40/4000
6420/6420 [==============================] - 0s 72us/sample - loss: 3.8745e-04 -
val_loss: 2.3926e-04
Epoch 41/4000
6420/6420 [==============================] - 0s 61us/sample - loss: 4.1845e-04 -
val_loss: 2.2952e-04
Epoch 42/4000
6420/6420 [==============================] - 0s 56us/sample - loss: 4.7070e-04 -
val_loss: 2.1442e-04
Epoch 43/4000
6420/6420 [==============================] - 0s 58us/sample - loss: 4.3976e-04 -
val_loss: 2.3676e-04
Epoch 44/4000
6420/6420 [==============================] - 0s 54us/sample - loss: 4.1991e-04 -
val_loss: 2.2202e-04
Epoch 45/4000
6420/6420 [==============================] - 0s 56us/sample - loss: 3.8917e-04 -
val_loss: 2.2512e-04
Epoch 46/4000
6420/6420 [==============================] - 0s 56us/sample - loss: 4.0262e-04 -
val_loss: 2.2537e-04
Epoch 47/4000
6420/6420 [==============================] - 0s 57us/sample - loss: 3.8574e-04 -
val_loss: 2.1841e-04
Epoch 48/4000
6420/6420 [==============================] - 0s 53us/sample - loss: 3.9176e-04 -
```

```
val_loss: 2.4825e-04
Epoch 49/4000
6420/6420 [==============================] - 1s 108us/sample - loss: 4.3170e-04
- val_loss: 2.1427e-04
Epoch 50/4000
6420/6420 [==============================] - 0s 62us/sample - loss: 3.8348e-04 -
val_loss: 2.2604e-04
Epoch 51/4000
6420/6420 [==============================] - 1s 116us/sample - loss: 3.8706e-04
- val_loss: 2.0489e-04
Epoch 52/4000
6420/6420 [==============================] - 0s 72us/sample - loss: 3.8088e-04 -
val_loss: 2.0127e-04
Epoch 53/4000
6420/6420 [==============================] - 0s 62us/sample - loss: 3.6288e-04 -
val_loss: 2.1046e-04
Epoch 54/4000
6420/6420 [==============================] - 0s 64us/sample - loss: 3.9881e-04 -
val_loss: 2.1971e-04
Epoch 55/4000
6420/6420 [==============================] - 0s 65us/sample - loss: 3.8779e-04 -
val_loss: 2.2584e-04
Epoch 56/4000
6420/6420 [==============================] - 0s 60us/sample - loss: 4.1901e-04 -
val_loss: 2.0682e-04
Epoch 57/4000
6420/6420 [==============================] - 0s 54us/sample - loss: 4.2708e-04 -
val_loss: 2.3885e-04
Epoch 58/4000
6420/6420 [==============================] - 0s 55us/sample - loss: 3.8187e-04 -
val_loss: 2.7590e-04
Epoch 59/4000
6420/6420 [==============================] - 0s 53us/sample - loss: 3.6134e-04 -
val_loss: 2.0648e-04
Epoch 60/4000
6420/6420 [==============================] - 0s 51us/sample - loss: 4.0795e-04 -
val_loss: 2.1653e-04
Epoch 61/4000
6420/6420 [==============================] - 0s 49us/sample - loss: 3.8923e-04 -
val_loss: 1.8850e-04
Epoch 62/4000
6420/6420 [==============================] - 1s 111us/sample - loss: 3.9704e-04
- val_loss: 2.1104e-04
Epoch 63/4000
6420/6420 [==============================] - 0s 52us/sample - loss: 3.6265e-04 -
val_loss: 1.9132e-04
Epoch 64/4000
6420/6420 [==============================] - 0s 48us/sample - loss: 4.2228e-04 -
```

```
val_loss: 1.9923e-04
Epoch 65/4000
6420/6420 [==============================] - 0s 64us/sample - loss: 4.0619e-04 -
val_loss: 1.9194e-04
Epoch 66/4000
6420/6420 [==============================] - 0s 50us/sample - loss: 3.7996e-04 -
val_loss: 1.9624e-04
Epoch 67/4000
6420/6420 [==============================] - 0s 53us/sample - loss: 3.6837e-04 -
val_loss: 2.0513e-04
Epoch 68/4000
6420/6420 [==============================] - 0s 53us/sample - loss: 3.6470e-04 -
val_loss: 1.7655e-04
Epoch 69/4000
6420/6420 [==============================] - 0s 71us/sample - loss: 3.8834e-04 -
val_loss: 1.8333e-04
Epoch 70/4000
6420/6420 [==============================] - 0s 55us/sample - loss: 4.3093e-04 -
val_loss: 2.1346e-04
Epoch 71/4000
6420/6420 [==============================] - 0s 51us/sample - loss: 3.3164e-04 -
val_loss: 1.9133e-04
Epoch 72/4000
6420/6420 [==============================] - 0s 56us/sample - loss: 4.1030e-04 -
val_loss: 1.9094e-04
Epoch 73/4000
6420/6420 [==============================] - 0s 62us/sample - loss: 3.8602e-04 -
val_loss: 1.8424e-04
Epoch 74/4000
6420/6420 [==============================] - 0s 71us/sample - loss: 3.6348e-04 -
val_loss: 2.0681e-04
Epoch 75/4000
6420/6420 [==============================] - 1s 92us/sample - loss: 3.9697e-04 -
val_loss: 2.1290e-04
Epoch 76/4000
6420/6420 [==============================] - 1s 85us/sample - loss: 3.9244e-04 -
val_loss: 1.9260e-04
Epoch 77/4000
6420/6420 [==============================] - 1s 93us/sample - loss: 4.3826e-04 -
val_loss: 2.0310e-04
Epoch 78/4000
6420/6420 [==============================] - 1s 82us/sample - loss: 3.9858e-04 -
val_loss: 2.3436e-04
Epoch 79/4000
6420/6420 [==============================] - 0s 74us/sample - loss: 4.1854e-04 -
val_loss: 1.9851e-04
Epoch 80/4000
6420/6420 [==============================] - 1s 99us/sample - loss: 3.5836e-04 -
```

```
val_loss: 2.1497e-04
Epoch 81/4000
6420/6420 [==============================] - 1s 82us/sample - loss: 3.6437e-04 -
val_loss: 2.1052e-04
Epoch 82/4000
6420/6420 [==============================] - 1s 109us/sample - loss: 4.2871e-04
- val_loss: 1.7637e-04
Epoch 83/4000
6420/6420 [==============================] - 1s 116us/sample - loss: 3.9076e-04
- val_loss: 1.8840e-04
Epoch 84/4000
6420/6420 [==============================] - 1s 124us/sample - loss: 3.9362e-04
- val_loss: 2.1731e-04
Epoch 85/4000
6420/6420 [==============================] - 1s 119us/sample - loss: 3.8175e-04
- val_loss: 2.2192e-04
Epoch 86/4000
6420/6420 [==============================] - 1s 96us/sample - loss: 3.7041e-04 -
val_loss: 1.7459e-04
Epoch 87/4000
6420/6420 [==============================] - 1s 101us/sample - loss: 3.8738e-04
- val_loss: 2.2406e-04
Epoch 88/4000
6420/6420 [==============================] - 1s 95us/sample - loss: 3.8013e-04 -
val_loss: 1.7943e-04
Epoch 89/4000
6420/6420 [==============================] - 1s 96us/sample - loss: 3.8381e-04 -
val_loss: 1.9004e-04
Epoch 90/4000
6420/6420 [==============================] - 1s 86us/sample - loss: 3.9642e-04 -
val_loss: 1.7015e-04
Epoch 91/4000
6420/6420 [==============================] - 1s 84us/sample - loss: 4.0069e-04 -
val_loss: 2.0521e-04
Epoch 92/4000
6420/6420 [==============================] - 1s 92us/sample - loss: 3.7681e-04 -
val_loss: 1.7749e-04
Epoch 93/4000
6420/6420 [==============================] - 1s 83us/sample - loss: 3.7360e-04 -
val_loss: 1.9228e-04
Epoch 94/4000
6420/6420 [==============================] - 1s 109us/sample - loss: 3.9241e-04
- val_loss: 2.1736e-04
Epoch 95/4000
6420/6420 [==============================] - 1s 91us/sample - loss: 3.7107e-04 -
val_loss: 1.9194e-04
Epoch 96/4000
6420/6420 [==============================] - 0s 77us/sample - loss: 4.1142e-04 -
```

```
val_loss: 1.7863e-04
Epoch 97/4000
6420/6420 [==============================] - 0s 69us/sample - loss: 3.7993e-04 -
val_loss: 1.7402e-04
Epoch 98/4000
6420/6420 [==============================] - 1s 82us/sample - loss: 3.7388e-04 -
val_loss: 1.9996e-04
Epoch 99/4000
6420/6420 [==============================] - 0s 64us/sample - loss: 3.8071e-04 -
val_loss: 1.6832e-04
Epoch 100/4000
6420/6420 [==============================] - 1s 80us/sample - loss: 3.7083e-04 -
val_loss: 1.7974e-04
Epoch 101/4000
6420/6420 [==============================] - 0s 49us/sample - loss: 4.1656e-04 -
val_loss: 2.1374e-04
Epoch 102/4000
6420/6420 [==============================] - 0s 66us/sample - loss: 4.1230e-04 -
val_loss: 1.9933e-04
Epoch 103/4000
6420/6420 [==============================] - 0s 54us/sample - loss: 3.9648e-04 -
val_loss: 1.8338e-04
Epoch 104/4000
6420/6420 [==============================] - 0s 47us/sample - loss: 3.6543e-04 -
val_loss: 1.8833e-04
Epoch 105/4000
6420/6420 [==============================] - 0s 45us/sample - loss: 3.7677e-04 -
val_loss: 1.6929e-04
Epoch 106/4000
6420/6420 [==============================] - 0s 44us/sample - loss: 4.1184e-04 -
val_loss: 1.8619e-04
Epoch 107/4000
6420/6420 [==============================] - 0s 46us/sample - loss: 4.0038e-04 -
val_loss: 1.7728e-04
Epoch 108/4000
6420/6420 [==============================] - 0s 44us/sample - loss: 4.0880e-04 -
val_loss: 1.7525e-04
Epoch 109/4000
6420/6420 [==============================] - 0s 49us/sample - loss: 3.5400e-04 -
val_loss: 1.7394e-04
Epoch 110/4000
6420/6420 [==============================] - 0s 49us/sample - loss: 3.7377e-04 -
val_loss: 1.9005e-04
Epoch 111/4000
6420/6420 [==============================] - 0s 46us/sample - loss: 3.5676e-04 -
val_loss: 1.5734e-04
Epoch 112/4000
6420/6420 [==============================] - 0s 45us/sample - loss: 3.6328e-04 -
```

```
val_loss: 1.9043e-04
Epoch 113/4000
6420/6420 [==============================] - 0s 49us/sample - loss: 4.0993e-04 -
val_loss: 1.8249e-04
Epoch 114/4000
6420/6420 [==============================] - 0s 45us/sample - loss: 3.7068e-04 -
val_loss: 2.0348e-04
Epoch 115/4000
6420/6420 [==============================] - 0s 51us/sample - loss: 4.1091e-04 -
val_loss: 2.0192e-04
Epoch 116/4000
6420/6420 [==============================] - 0s 47us/sample - loss: 4.1776e-04 -
val_loss: 1.8761e-04
Epoch 117/4000
6420/6420 [==============================] - 0s 56us/sample - loss: 3.5270e-04 -
val_loss: 1.8552e-04
Epoch 118/4000
6420/6420 [==============================] - 0s 70us/sample - loss: 3.8690e-04 -
val_loss: 1.7253e-04
Epoch 119/4000
6420/6420 [==============================] - 0s 67us/sample - loss: 4.0213e-04 -
val_loss: 1.8870e-04
Epoch 120/4000
6420/6420 [==============================] - 0s 72us/sample - loss: 3.5177e-04 -
val_loss: 1.6634e-04
Epoch 121/4000
6420/6420 [==============================] - 0s 77us/sample - loss: 4.2021e-04 -
val_loss: 1.7592e-04
Epoch 122/4000
6420/6420 [==============================] - 0s 60us/sample - loss: 3.9119e-04 -
val_loss: 1.6329e-04
Epoch 123/4000
6420/6420 [==============================] - 0s 63us/sample - loss: 3.4388e-04 -
val_loss: 1.7831e-04
Epoch 124/4000
6420/6420 [==============================] - 0s 66us/sample - loss: 3.8592e-04 -
val_loss: 2.1806e-04
Epoch 125/4000
6420/6420 [==============================] - 0s 57us/sample - loss: 3.6078e-04 -
val_loss: 1.8532e-04
Epoch 126/4000
6420/6420 [==============================] - 0s 51us/sample - loss: 3.8341e-04 -
val_loss: 1.6547e-04
Epoch 127/4000
6420/6420 [==============================] - 0s 55us/sample - loss: 3.5188e-04 -
val_loss: 1.5375e-04
Epoch 128/4000
6420/6420 [==============================] - 0s 55us/sample - loss: 3.9971e-04 -
```

```
val_loss: 1.8052e-04
Epoch 129/4000
6420/6420 [==============================] - 0s 45us/sample - loss: 3.9857e-04 -
val_loss: 2.4237e-04
Epoch 130/4000
6420/6420 [==============================] - 0s 47us/sample - loss: 3.5901e-04 -
val_loss: 1.5623e-04
Epoch 131/4000
6420/6420 [==============================] - 0s 47us/sample - loss: 3.7680e-04 -
val_loss: 2.1051e-04
Epoch 132/4000
6420/6420 [==============================] - 0s 45us/sample - loss: 3.7886e-04 -
val_loss: 1.5456e-04
Epoch 133/4000
6420/6420 [==============================] - 0s 50us/sample - loss: 3.8955e-04 -
val_loss: 1.8789e-04
Epoch 134/4000
6420/6420 [==============================] - 0s 46us/sample - loss: 3.5765e-04 -
val_loss: 1.6933e-04
Epoch 135/4000
6420/6420 [==============================] - 0s 44us/sample - loss: 3.7434e-04 -
val_loss: 1.7699e-04
Epoch 136/4000
6420/6420 [==============================] - 0s 45us/sample - loss: 3.8558e-04 -
val_loss: 1.6628e-04
Epoch 137/4000
6420/6420 [==============================] - 0s 45us/sample - loss: 3.9230e-04 -
val_loss: 1.6465e-04
Epoch 138/4000
6420/6420 [==============================] - 0s 47us/sample - loss: 3.8368e-04 -
val_loss: 1.7166e-04
Epoch 139/4000
6420/6420 [==============================] - 0s 50us/sample - loss: 3.5695e-04 -
val_loss: 1.7670e-04
Epoch 140/4000
6420/6420 [==============================] - 0s 44us/sample - loss: 3.7813e-04 -
val_loss: 1.8580e-04
Epoch 141/4000
6420/6420 [==============================] - 0s 43us/sample - loss: 3.7063e-04 -
val_loss: 1.9484e-04
Epoch 142/4000
6420/6420 [==============================] - 0s 44us/sample - loss: 3.8097e-04 -
val_loss: 2.1516e-04
Epoch 143/4000
6420/6420 [==============================] - 0s 46us/sample - loss: 4.0369e-04 -
val_loss: 1.8775e-04
Epoch 144/4000
6420/6420 [==============================] - 0s 44us/sample - loss: 3.8564e-04 -
```

```
val_loss: 2.3136e-04
Epoch 145/4000
6420/6420 [==============================] - 0s 45us/sample - loss: 3.7312e-04 -
val_loss: 1.7378e-04
Epoch 146/4000
6420/6420 [==============================] - 0s 43us/sample - loss: 3.6275e-04 -
val_loss: 1.7078e-04
Epoch 147/4000
6420/6420 [==============================] - 0s 51us/sample - loss: 3.9961e-04 -
val_loss: 1.7361e-04
Epoch 148/4000
6420/6420 [==============================] - 0s 59us/sample - loss: 4.0065e-04 -
val_loss: 2.1212e-04
Epoch 149/4000
6420/6420 [==============================] - 0s 63us/sample - loss: 3.6188e-04 -
val_loss: 1.9285e-04
Epoch 150/4000
6420/6420 [==============================] - 0s 72us/sample - loss: 3.9891e-04 -
val_loss: 1.7574e-04
Epoch 151/4000
6420/6420 [==============================] - 0s 61us/sample - loss: 3.5630e-04 -
val_loss: 1.8421e-04
Epoch 152/4000
6420/6420 [==============================] - 0s 71us/sample - loss: 4.4336e-04 -
val_loss: 1.7520e-04
Epoch 153/4000
6420/6420 [==============================] - 0s 72us/sample - loss: 3.7000e-04 -
val_loss: 1.6459e-04
Epoch 154/4000
6420/6420 [==============================] - 0s 65us/sample - loss: 4.0566e-04 -
val_loss: 1.6536e-04
Epoch 155/4000
6420/6420 [==============================] - 0s 73us/sample - loss: 3.6789e-04 -
val_loss: 1.7232e-04
Epoch 156/4000
6420/6420 [==============================] - 0s 65us/sample - loss: 3.6652e-04 -
val_loss: 1.8355e-04
Epoch 157/4000
6420/6420 [==============================] - 0s 49us/sample - loss: 3.6724e-04 -
val_loss: 1.8785e-04
Epoch 158/4000
6420/6420 [==============================] - 0s 76us/sample - loss: 3.9953e-04 -
val_loss: 1.8905e-04
Epoch 159/4000
6420/6420 [==============================] - 0s 65us/sample - loss: 3.7878e-04 -
val_loss: 2.1174e-04
Epoch 160/4000
6420/6420 [==============================] - 0s 61us/sample - loss: 4.0052e-04 -
```

```
val_loss: 1.9447e-04
Epoch 161/4000
6420/6420 [==============================] - 0s 47us/sample - loss: 3.7289e-04 -
val_loss: 1.7476e-04
Epoch 162/4000
6420/6420 [==============================] - 0s 48us/sample - loss: 3.9438e-04 -
val_loss: 1.6093e-04
Epoch 163/4000
6420/6420 [==============================] - 0s 49us/sample - loss: 3.8562e-04 -
val_loss: 1.6319e-04
Epoch 164/4000
6420/6420 [==============================] - 0s 44us/sample - loss: 3.6413e-04 -
val_loss: 2.0747e-04
Epoch 165/4000
6420/6420 [==============================] - 0s 45us/sample - loss: 3.7078e-04 -
val_loss: 1.5520e-04
Epoch 166/4000
6420/6420 [==============================] - 0s 47us/sample - loss: 3.3178e-04 -
val_loss: 1.5067e-04
Epoch 167/4000
6420/6420 [==============================] - 0s 45us/sample - loss: 3.6772e-04 -
val_loss: 1.7527e-04
Epoch 168/4000
6420/6420 [==============================] - 0s 45us/sample - loss: 3.7344e-04 -
val_loss: 1.8171e-04
Epoch 169/4000
6420/6420 [==============================] - 0s 43us/sample - loss: 4.1335e-04 -
val_loss: 2.5147e-04
Epoch 170/4000
6420/6420 [==============================] - 0s 51us/sample - loss: 3.9727e-04 -
val_loss: 1.7804e-04
Epoch 171/4000
6420/6420 [==============================] - 0s 46us/sample - loss: 3.9421e-04 -
val_loss: 1.5357e-04
Epoch 172/4000
6420/6420 [==============================] - 0s 45us/sample - loss: 4.0950e-04 -
val_loss: 1.6474e-04
Epoch 173/4000
6420/6420 [==============================] - 0s 43us/sample - loss: 3.6053e-04 -
val_loss: 1.7761e-04
Epoch 174/4000
6420/6420 [==============================] - 0s 45us/sample - loss: 4.1252e-04 -
val_loss: 1.9731e-04
Epoch 175/4000
6420/6420 [==============================] - 0s 43us/sample - loss: 3.8814e-04 -
val_loss: 1.4349e-04
Epoch 176/4000
6420/6420 [==============================] - 0s 43us/sample - loss: 4.3570e-04 -
```

```
val_loss: 2.0685e-04
Epoch 177/4000
6420/6420 [==============================] - 0s 52us/sample - loss: 3.8089e-04 -
val_loss: 1.7820e-04
Epoch 178/4000
6420/6420 [==============================] - 0s 57us/sample - loss: 3.5555e-04 -
val_loss: 1.4901e-04
Epoch 179/4000
6420/6420 [==============================] - 0s 53us/sample - loss: 3.9112e-04 -
val_loss: 1.8517e-04
Epoch 180/4000
6420/6420 [==============================] - 0s 61us/sample - loss: 3.7667e-04 -
val_loss: 1.5498e-04
Epoch 181/4000
6420/6420 [==============================] - 1s 80us/sample - loss: 3.8949e-04 -
val_loss: 1.9027e-04
Epoch 182/4000
6420/6420 [==============================] - 0s 72us/sample - loss: 3.8668e-04 -
val_loss: 1.5464e-04
Epoch 183/4000
6420/6420 [==============================] - 0s 65us/sample - loss: 3.9501e-04 -
val_loss: 1.6829e-04
Epoch 184/4000
6420/6420 [==============================] - 0s 63us/sample - loss: 3.9367e-04 -
val_loss: 2.2951e-04
Epoch 185/4000
6420/6420 [==============================] - 0s 66us/sample - loss: 3.7516e-04 -
val_loss: 1.6364e-04
Epoch 186/4000
6420/6420 [==============================] - 0s 74us/sample - loss: 4.0604e-04 -
val_loss: 1.6113e-04
Epoch 187/4000
6420/6420 [==============================] - 0s 59us/sample - loss: 4.4246e-04 -
val_loss: 1.6503e-04
Epoch 188/4000
6420/6420 [==============================] - 0s 56us/sample - loss: 3.9377e-04 -
val_loss: 1.9615e-04
Epoch 189/4000
6420/6420 [==============================] - 0s 51us/sample - loss: 3.9532e-04 -
val_loss: 1.9750e-04
Epoch 190/4000
6420/6420 [==============================] - 0s 46us/sample - loss: 4.0413e-04 -
val_loss: 1.5363e-04
Epoch 191/4000
6420/6420 [==============================] - 0s 50us/sample - loss: 3.8147e-04 -
val_loss: 1.7732e-04
Epoch 192/4000
6420/6420 [==============================] - 0s 46us/sample - loss: 3.6523e-04 -
```

```
val_loss: 1.6430e-04
Epoch 193/4000
6420/6420 [==============================] - 0s 49us/sample - loss: 3.8917e-04 -
val_loss: 1.7617e-04
Epoch 194/4000
6420/6420 [==============================] - 0s 48us/sample - loss: 3.7432e-04 -
val_loss: 2.2310e-04
Epoch 195/4000
6420/6420 [==============================] - 0s 54us/sample - loss: 3.8312e-04 -
val_loss: 1.6982e-04
Epoch 196/4000
6420/6420 [==============================] - 0s 51us/sample - loss: 3.8935e-04 -
val_loss: 1.4669e-04
Epoch 197/4000
6420/6420 [==============================] - 0s 63us/sample - loss: 3.6802e-04 -
val_loss: 1.4314e-04
Epoch 198/4000
6420/6420 [==============================] - 0s 71us/sample - loss: 3.6925e-04 -
val_loss: 1.5511e-04
Epoch 199/4000
6420/6420 [==============================] - 0s 64us/sample - loss: 3.8408e-04 -
val_loss: 1.7227e-04
Epoch 200/4000
6420/6420 [==============================] - 0s 70us/sample - loss: 3.7144e-04 -
val_loss: 1.6129e-04
Epoch 201/4000
6420/6420 [==============================] - 0s 50us/sample - loss: 3.7737e-04 -
val_loss: 2.0729e-04
Epoch 202/4000
6420/6420 [==============================] - 0s 49us/sample - loss: 3.5967e-04 -
val_loss: 1.5644e-04
Epoch 203/4000
6420/6420 [==============================] - 0s 47us/sample - loss: 3.7491e-04 -
val_loss: 1.5643e-04
Epoch 204/4000
6420/6420 [==============================] - 0s 46us/sample - loss: 3.8401e-04 -
val_loss: 1.5636e-04
Epoch 205/4000
6420/6420 [==============================] - 0s 49us/sample - loss: 4.1449e-04 -
val_loss: 1.5338e-04
Epoch 206/4000
6420/6420 [==============================] - 0s 53us/sample - loss: 4.1122e-04 -
val_loss: 1.6970e-04
Epoch 207/4000
6420/6420 [==============================] - 0s 65us/sample - loss: 4.2181e-04 -
val_loss: 1.5571e-04
Epoch 208/4000
6420/6420 [==============================] - 0s 77us/sample - loss: 3.7934e-04 -
```

```
val_loss: 1.4740e-04
Epoch 209/4000
6420/6420 [==============================] - ETA: 0s - loss: 3.4250e-04- ETA: 0s
- loss: 3.2 - 0s 71us/sample - loss: 3.4491e-04 - val_loss: 1.4933e-04
Epoch 210/4000
6420/6420 [==============================] - 0s 68us/sample - loss: 3.8353e-04 -
val_loss: 1.9531e-04
Epoch 211/4000
6420/6420 [==============================] - 0s 68us/sample - loss: 3.8864e-04 -
val_loss: 1.7508e-04
Epoch 212/4000
6420/6420 [==============================] - 0s 75us/sample - loss: 3.6905e-04 -
val_loss: 1.6643e-04
Epoch 213/4000
6420/6420 [==============================] - 0s 77us/sample - loss: 4.0698e-04 -
val_loss: 1.7517e-04
Epoch 214/4000
6420/6420 [==============================] - 0s 53us/sample - loss: 4.0734e-04 -
val_loss: 1.9799e-04
Epoch 215/4000
6420/6420 [==============================] - 0s 50us/sample - loss: 4.1990e-04 -
val_loss: 1.6936e-04
Epoch 216/4000
6420/6420 [==============================] - 0s 51us/sample - loss: 3.9171e-04 -
val_loss: 1.9487e-04
Epoch 217/4000
6420/6420 [==============================] - 0s 49us/sample - loss: 3.9314e-04 -
val_loss: 1.6527e-04
Epoch 218/4000
6420/6420 [==============================] - 0s 54us/sample - loss: 4.1101e-04 -
val_loss: 1.6359e-04
Epoch 219/4000
6420/6420 [==============================] - 0s 44us/sample - loss: 3.8098e-04 -
val_loss: 1.6258e-04
Epoch 220/4000
6420/6420 [==============================] - 0s 43us/sample - loss: 3.7306e-04 -
val_loss: 1.4014e-04
Epoch 221/4000
6420/6420 [==============================] - 0s 44us/sample - loss: 3.9563e-04 -
val_loss: 1.3826e-04
Epoch 222/4000
6420/6420 [==============================] - 0s 46us/sample - loss: 3.6621e-04 -
val_loss: 1.6607e-04
Epoch 223/4000
6420/6420 [==============================] - 0s 45us/sample - loss: 3.6504e-04 -
val_loss: 1.4828e-04
Epoch 224/4000
6420/6420 [==============================] - 0s 44us/sample - loss: 3.7195e-04 -
```

```
val_loss: 1.6845e-04
Epoch 225/4000
6420/6420 [==============================] - 0s 51us/sample - loss: 3.6580e-04 -
val_loss: 1.7061e-04
Epoch 226/4000
6420/6420 [==============================] - 0s 48us/sample - loss: 3.5988e-04 -
val_loss: 1.4069e-04
Epoch 227/4000
6420/6420 [==============================] - 0s 44us/sample - loss: 3.8396e-04 -
val_loss: 1.6442e-04
Epoch 228/4000
6420/6420 [==============================] - 0s 45us/sample - loss: 3.6081e-04 -
val_loss: 1.6840e-04
Epoch 229/4000
6420/6420 [==============================] - 0s 44us/sample - loss: 3.8802e-04 -
val_loss: 1.6799e-04
Epoch 230/4000
6420/6420 [==============================] - 0s 67us/sample - loss: 3.9271e-04 -
val_loss: 1.7598e-04
Epoch 231/4000
6420/6420 [==============================] - 0s 66us/sample - loss: 3.9200e-04 -
val_loss: 1.7821e-04
Epoch 232/4000
6420/6420 [==============================] - 0s 50us/sample - loss: 3.9613e-04 -
val_loss: 1.6983e-04
Epoch 233/4000
6420/6420 [==============================] - 0s 47us/sample - loss: 3.8921e-04 -
val_loss: 1.6949e-04
Epoch 234/4000
6420/6420 [==============================] - 0s 53us/sample - loss: 3.9164e-04 -
val_loss: 2.3774e-04
Epoch 235/4000
6420/6420 [==============================] - 0s 55us/sample - loss: 3.6622e-04 -
val_loss: 1.5798e-04
Epoch 236/4000
6420/6420 [==============================] - 0s 61us/sample - loss: 3.6441e-04 -
val_loss: 1.9084e-04
Epoch 237/4000
6420/6420 [==============================] - 0s 69us/sample - loss: 4.0341e-04 -
val_loss: 1.8301e-04
Epoch 238/4000
6420/6420 [==============================] - 0s 73us/sample - loss: 3.6314e-04 -
val_loss: 1.7106e-04
Epoch 239/4000
6420/6420 [==============================] - 0s 78us/sample - loss: 3.7995e-04 -
val_loss: 1.7895e-04
Epoch 240/4000
6420/6420 [==============================] - 0s 78us/sample - loss: 3.8314e-04 -
```

```
val_loss: 1.5395e-04
Epoch 241/4000
6420/6420 [==============================] - 0s 70us/sample - loss: 4.5325e-04 -
val_loss: 2.7468e-04
Epoch 242/4000
6420/6420 [==============================] - 0s 66us/sample - loss: 4.1948e-04 -
val_loss: 2.0811e-04
Epoch 243/4000
6420/6420 [==============================] - 0s 51us/sample - loss: 3.5239e-04 -
val_loss: 1.4190e-04
Epoch 244/4000
6420/6420 [==============================] - 0s 46us/sample - loss: 3.8547e-04 -
val_loss: 1.4648e-04
Epoch 245/4000
6420/6420 [==============================] - 0s 49us/sample - loss: 3.8467e-04 -
val_loss: 1.5179e-04
Epoch 246/4000
6420/6420 [==============================] - 0s 44us/sample - loss: 3.8492e-04 -
val_loss: 1.5877e-04
Epoch 247/4000
6420/6420 [==============================] - 0s 43us/sample - loss: 3.8039e-04 -
val_loss: 1.7363e-04
Epoch 248/4000
6420/6420 [==============================] - 0s 51us/sample - loss: 4.1558e-04 -
val_loss: 1.5646e-04
Epoch 249/4000
6420/6420 [==============================] - 0s 43us/sample - loss: 3.7851e-04 -
val_loss: 1.9639e-04
Epoch 250/4000
6420/6420 [==============================] - 0s 44us/sample - loss: 3.9576e-04 -
val_loss: 1.6500e-04
Epoch 251/4000
6420/6420 [==============================] - 0s 47us/sample - loss: 4.1462e-04 -
val_loss: 1.8896e-04
Epoch 252/4000
6420/6420 [==============================] - 0s 43us/sample - loss: 3.6546e-04 -
val_loss: 1.4212e-04
Epoch 253/4000
6420/6420 [==============================] - 0s 43us/sample - loss: 3.9780e-04 -
val_loss: 1.5303e-04
Epoch 254/4000
6420/6420 [==============================] - 0s 48us/sample - loss: 3.9610e-04 -
val_loss: 1.6398e-04
Epoch 255/4000
6420/6420 [==============================] - 0s 46us/sample - loss: 4.1971e-04 -
val_loss: 1.6363e-04
Epoch 256/4000
6420/6420 [==============================] - 0s 43us/sample - loss: 3.6570e-04 -
```

```
val_loss: 1.5956e-04
Epoch 257/4000
6420/6420 [==============================] - 0s 42us/sample - loss: 3.4217e-04 -
val_loss: 1.4108e-04
Epoch 258/4000
6420/6420 [==============================] - 0s 43us/sample - loss: 3.8853e-04 -
val_loss: 1.5824e-04
Epoch 259/4000
6420/6420 [==============================] - 0s 53us/sample - loss: 3.8600e-04 -
val_loss: 1.6287e-04
Epoch 260/4000
6420/6420 [==============================] - 0s 46us/sample - loss: 4.1746e-04 -
val_loss: 2.1092e-04
Epoch 261/4000
6420/6420 [==============================] - 0s 53us/sample - loss: 3.8136e-04 -
val_loss: 1.4752e-04
Epoch 262/4000
6420/6420 [==============================] - 0s 55us/sample - loss: 3.4157e-04 -
val_loss: 1.8513e-04
Epoch 263/4000
6420/6420 [==============================] - 0s 50us/sample - loss: 3.7130e-04 -
val_loss: 1.6227e-04
Epoch 264/4000
6420/6420 [==============================] - 0s 64us/sample - loss: 3.8783e-04 -
val_loss: 1.4024e-04
Epoch 265/4000
6420/6420 [==============================] - 0s 67us/sample - loss: 3.7614e-04 -
val_loss: 1.4579e-04
Epoch 266/4000
6420/6420 [==============================] - 1s 78us/sample - loss: 4.1468e-04 -
val_loss: 1.4124e-04
Epoch 267/4000
6420/6420 [==============================] - 0s 65us/sample - loss: 3.9763e-04 -
val_loss: 2.0330e-04
Epoch 268/4000
6420/6420 [==============================] - 1s 91us/sample - loss: 3.7932e-04 -
val_loss: 1.8770e-04
Epoch 269/4000
6420/6420 [==============================] - 0s 69us/sample - loss: 3.4800e-04 -
val_loss: 2.0037e-04
Epoch 270/4000
6420/6420 [==============================] - 0s 67us/sample - loss: 3.9476e-04 -
val_loss: 1.8254e-04
Epoch 271/4000
6420/6420 [==============================] - 0s 68us/sample - loss: 3.5524e-04 -
val_loss: 1.5522e-04
```

[22]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                multiple                  870
_________________________________________________________________
dropout (Dropout)            multiple                  0
_________________________________________________________________
dense_1 (Dense)              multiple                  1740
_________________________________________________________________
dropout_1 (Dropout)          multiple                  0
_________________________________________________________________
dense_2 (Dense)              multiple                  59
=================================================================
Total params: 2,669
Trainable params: 2,669
Non-trainable params: 0
_________________________________________________________________
```

[23]:
```
import os
```

[24]:
```
losses = model.history.history
losses['loss'] = np.asarray(losses['loss'])
losses['val_loss'] = np.asarray(losses['val_loss'])
final_number_of_epochs = len(losses['loss'])
min_loss = losses['loss'].min()
mean_loss = losses['loss'].mean()
final_loss = losses['loss'][-1]
min_val_loss = losses['val_loss'].min()
mean_val_loss = losses['val_loss'].mean()
final_val_loss = losses['val_loss'][-1]

def get_model_summary():
    output = []
    model.summary(print_fn=lambda line: output.append(line))
    return str(output).strip('[]')

summary = get_model_summary()

record = {
    'Epochs': final_number_of_epochs,
    'Batch_Size': batch_size,
    'Loss_Func': loss_param,
    'Optimizer': optimizer_param,
    'Early_Stop_Monitor': stop_monitor,
```

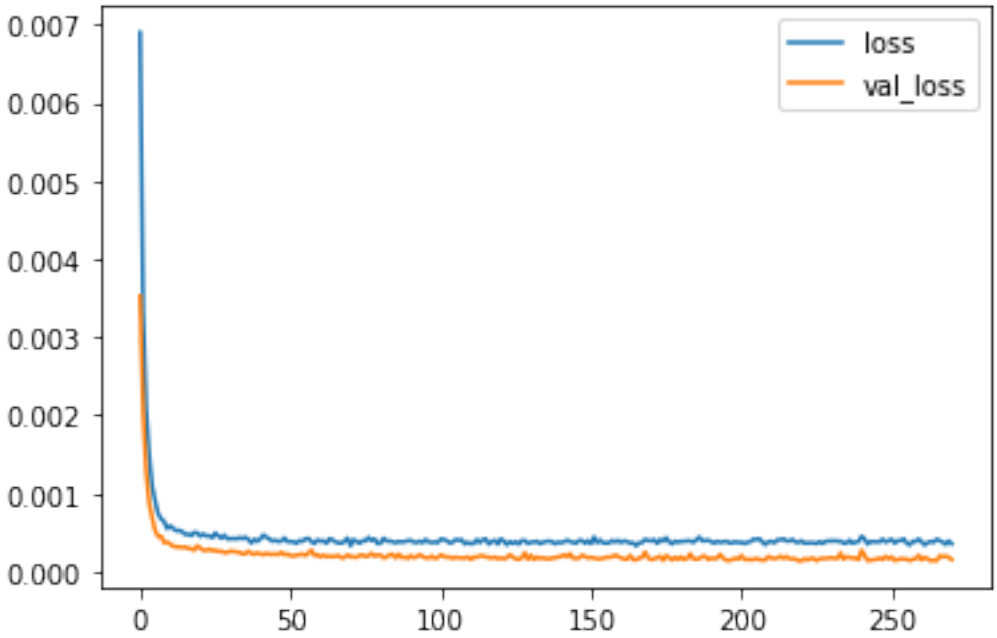
```
    'Early_Stop_Patience': stop_patience,
    'Min_Loss': min_loss,
    'Mean_Loss': mean_loss,
    'Final_Loss': final_loss,
    'Min_Val_Loss': min_val_loss,
    'Mean_Val_Loss': mean_val_loss,
    'Final_Val_Loss': final_val_loss,
    'Model': summary
}

new_data = pd.DataFrame(record, index=[0])

if os.path.exists('../core/records/pitching_results.csv'):
    df_records = pd.read_csv('../core/records/pitching_results.csv')
    df_records = df_records.append(new_data)
else:
    df_records = pd.DataFrame(new_data)

df_records.to_csv('../core/records/pitching_results.csv', index=False,␣
 ↪float_format='%g')
```

Model Evaluation

[25]:
```
losses = pd.DataFrame(model.history.history)
```

[26]:
```
losses.plot()
```

[26]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x145c796d0>
```



[27]:
```
predictions = model.predict(X_test)
predictions = [pred for sublist in predictions for pred in sublist]
```

[28]:
```
test_player_ratings = dict(zip(X_test_keys, predictions))
```

[29]:
```
player_key = df['retroID']
```

[30]:
```
player_key
```

[30]:
```
0       aardd001
1       aased001
2       abadf001
3       abbog001
4       abboj001
          ...
8020    zolds101
8021    zubeb101
8022    zumaj001
8023    zuveg101
8024    zycht001
Name: retroID, Length: 8025, dtype: object
```

[31]:
```
results = model.predict(tensor.to_numpy())
```

[32]:
```
len(results)
```

[32]:
```
8025
```

[33]:
```
results.mean()
```

[33]:
```
0.6065751
```

[34]:
```
df['Pitching'].shape
```

[34]:
```
(8025,)
```

[35]:
```
results.shape
```

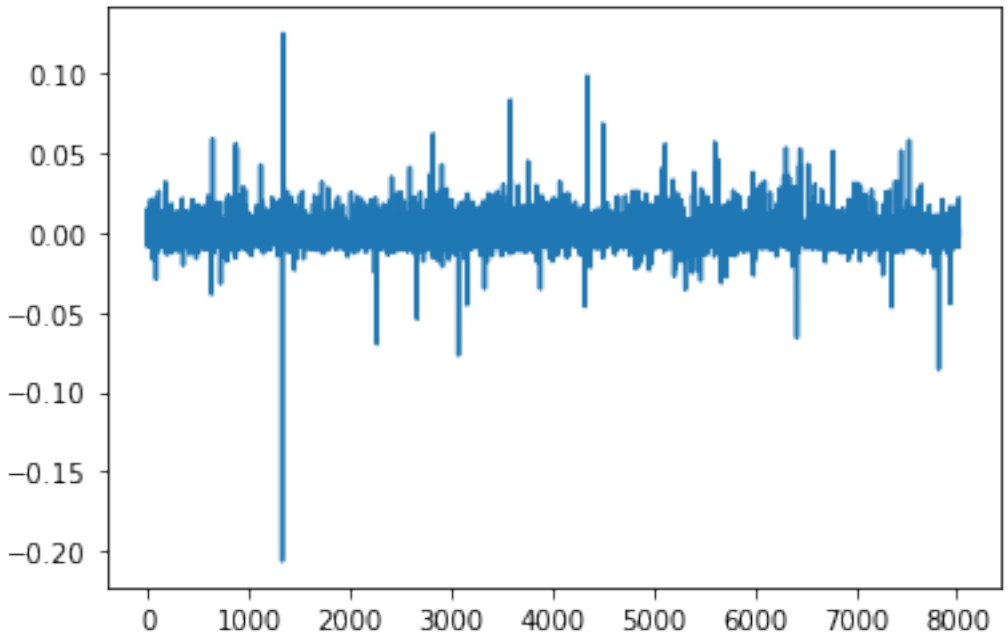
[35]:
```
(8025, 1)
```

[36]:
```
results = [pred for sublist in results for pred in sublist]
```

[37]:
```
diff = df['Pitching'] - results
```

[38]:
```
diff.plot()
```

[38]: <matplotlib.axes._subplots.AxesSubplot at 0x14684bd90>



[39]:
```
diff.mean()
```

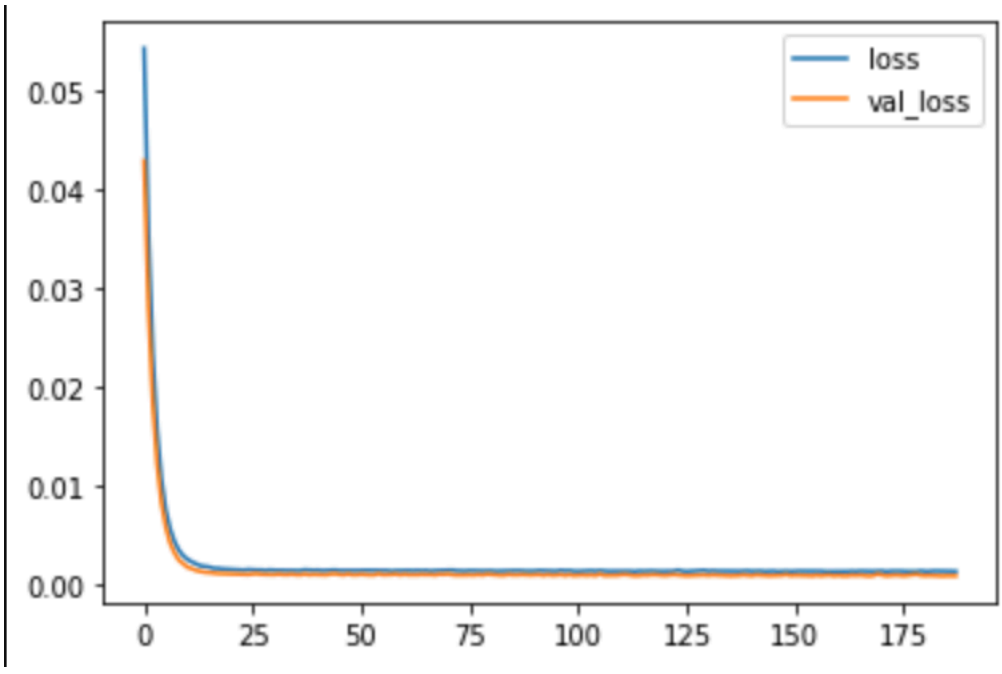
[39]: -0.0009138342898704561

[ ]:

Model 1

Mean Loss:	0.00210681
Mean Validation Loss:	0.00150978
Batch Size:	128

```
model = Sequential()

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(116, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(232, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(116, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



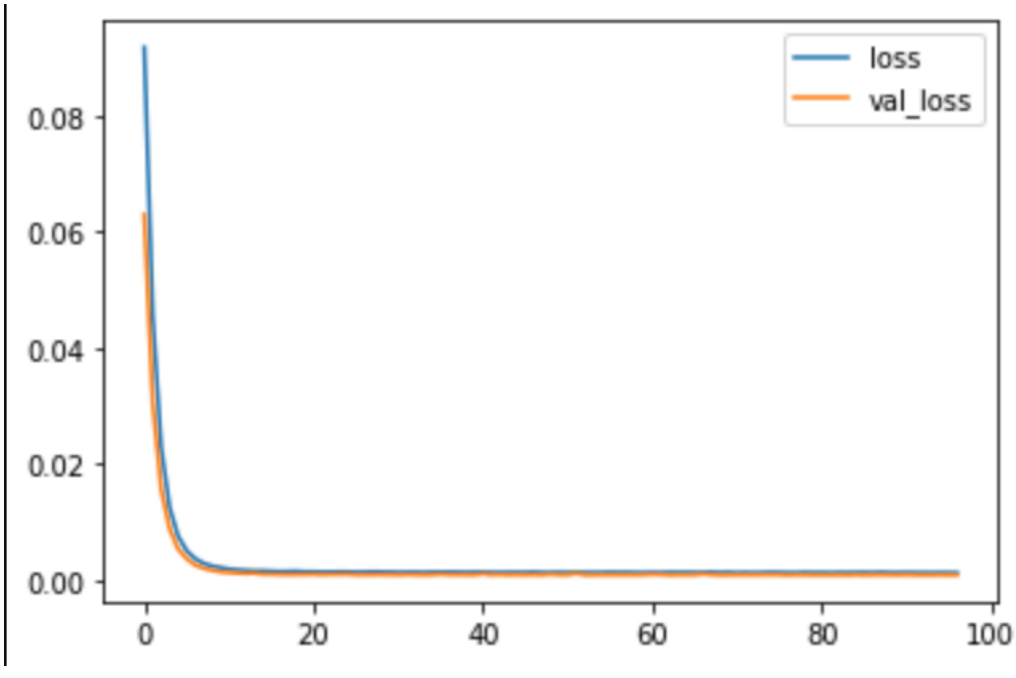
Model 2

Mean Loss: 0.00317233

Mean Validation Loss: 0.00217707

Batch Size: 128

```
model = Sequential()

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(116, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(232, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(464, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(232, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(116, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



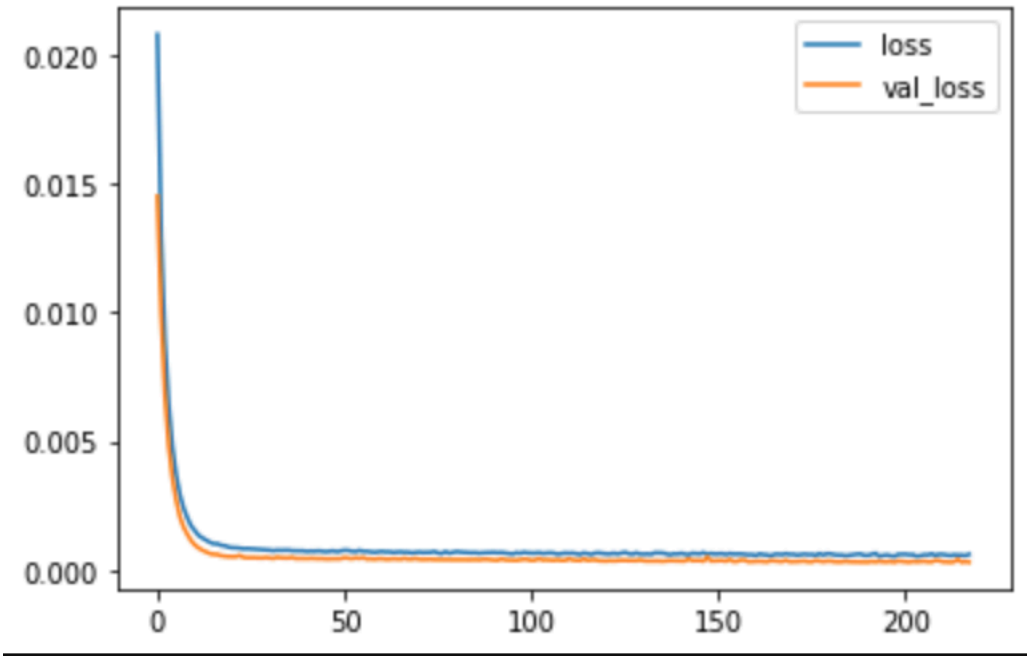
Model 3

Mean Loss: 0.000962387

Mean Validation Loss: 0.000614565

Batch Size: 128

```
model = Sequential()

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(116, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



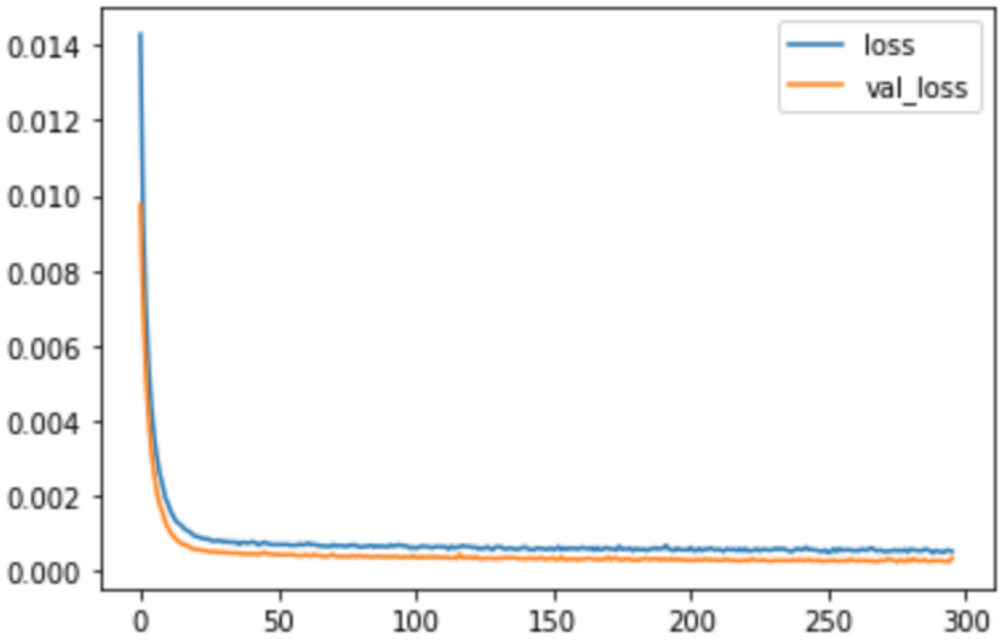
Model 4

Mean Loss: 0.000815947

Mean Validation Loss: 0.000490422

Batch Size: 128

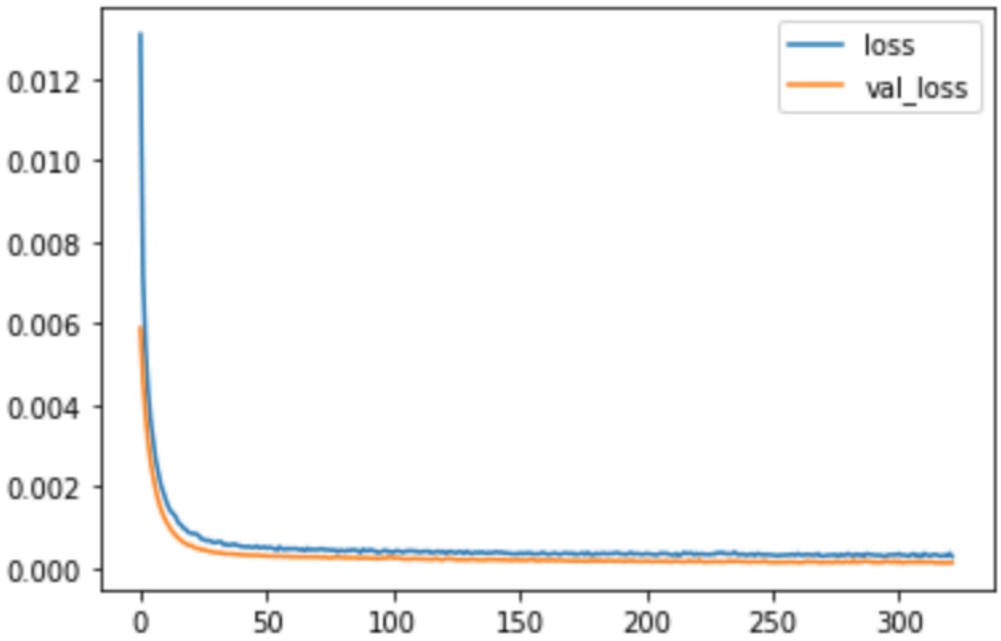
```
model = Sequential()

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



Model 5

Mean Loss:	0.000542663
Mean Validation Loss:	0.000305115
Batch Size:	128

```
model = Sequential()

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



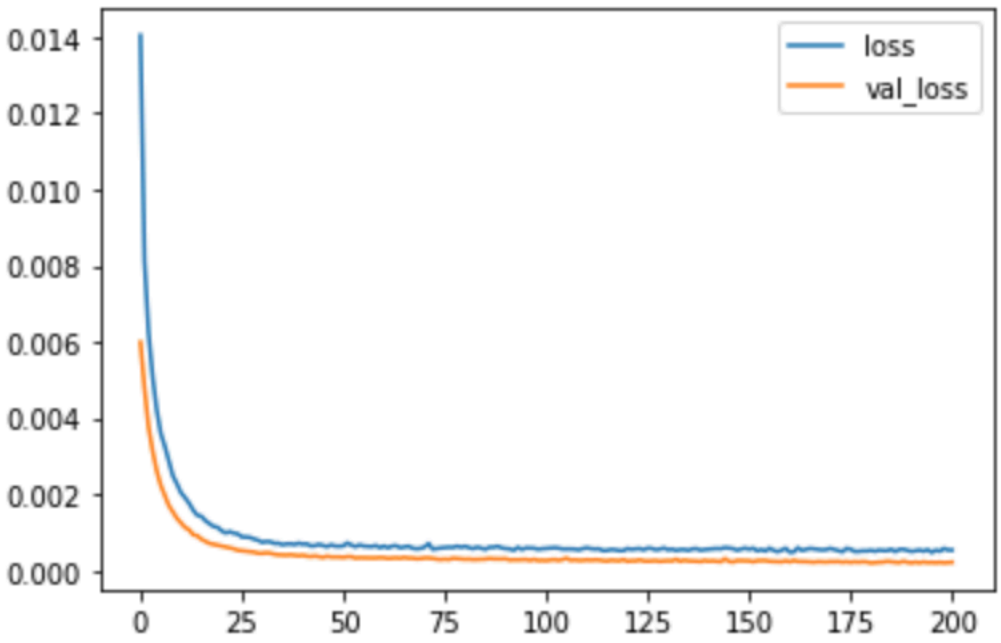
Model 6

Mean Loss: 0.00088016

Mean Validation Loss: 0.000459758

Batch Size: 128

```
model = Sequential()

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(14, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



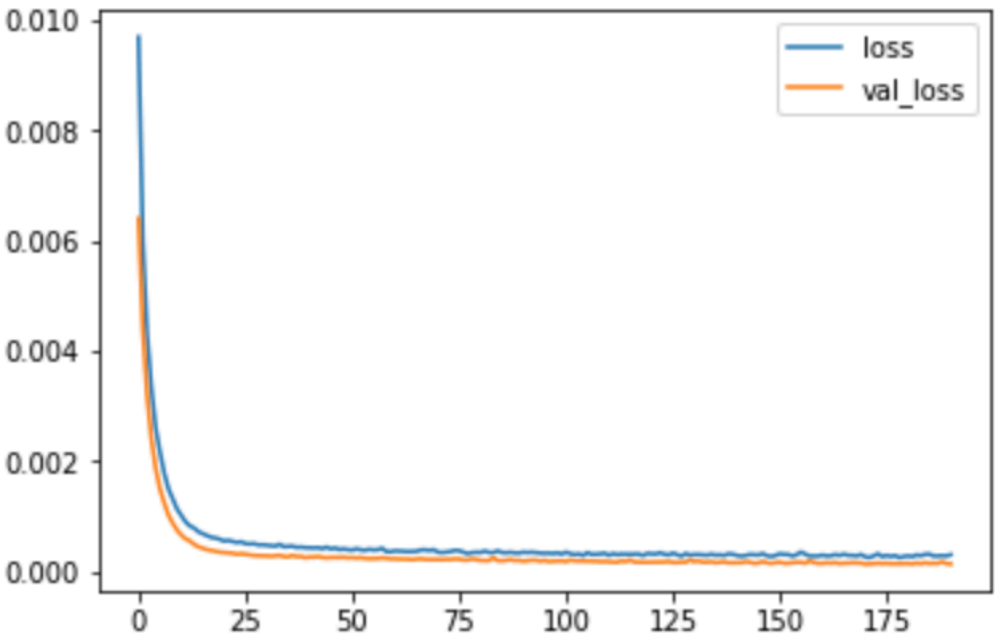
Model 7

Mean Loss: 0.000530716

Mean Validation Loss: 0.000331629

Batch Size: 128

```
model = Sequential()

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(116, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



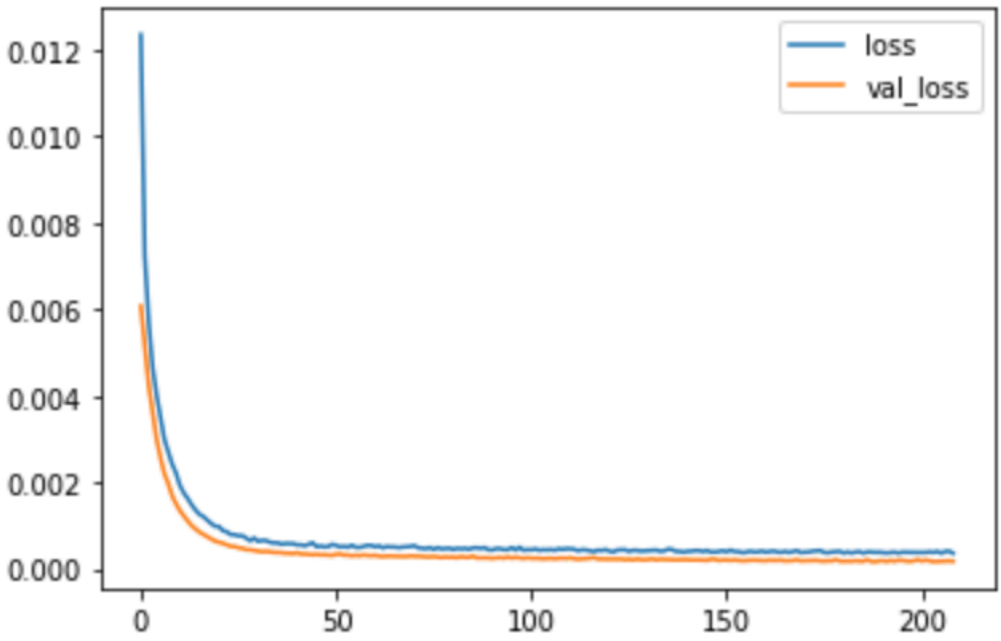
Model 8

Mean Loss: 0.000713679

Mean Validation Loss: 0.000428158

Batch Size: 128

```
model = Sequential()

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



Model 9

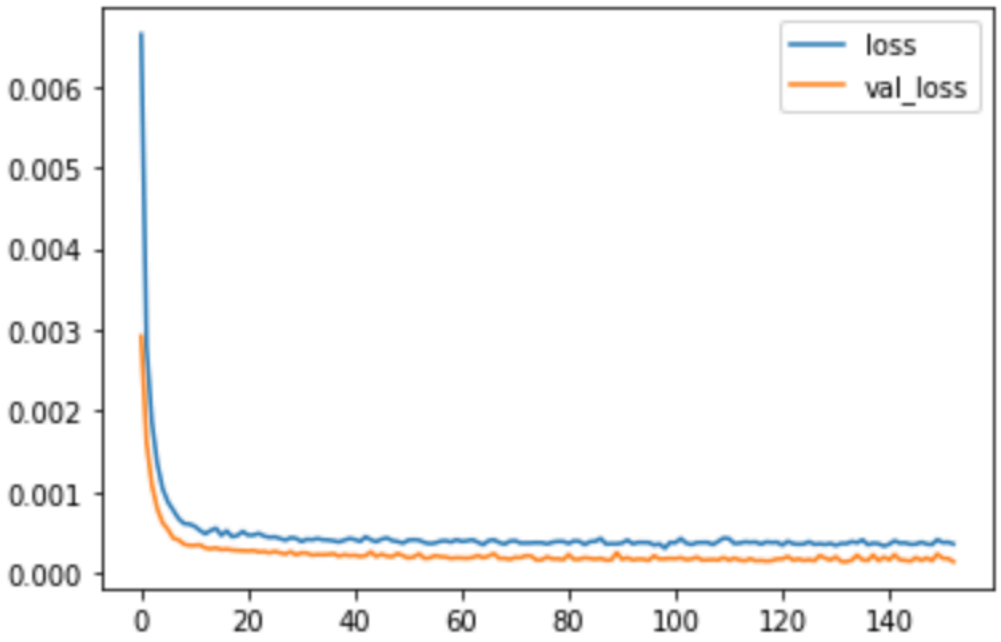
Mean Loss: 0.000486567

Mean Validation Loss: 0.000252159

Batch Size: 32

Model 5 with a much smaller batch size

```
model = Sequential()

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



Model 10

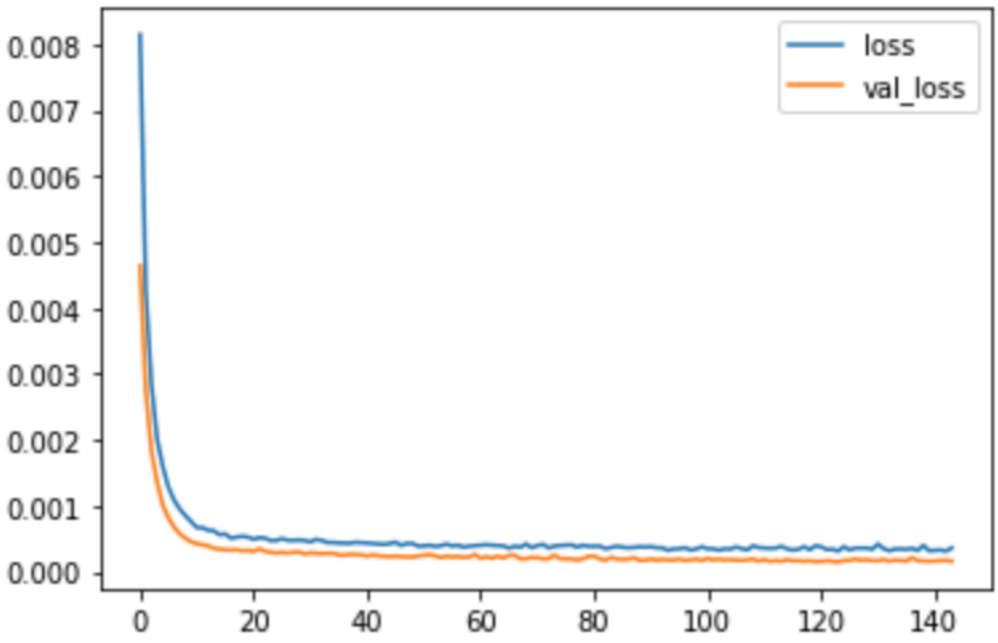
Mean Loss: 0.000540004

Mean Validation Loss: 0.000306904

Batch Size: 64

Model 5 with a smaller batch size

```
model = Sequential()

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



Model 11

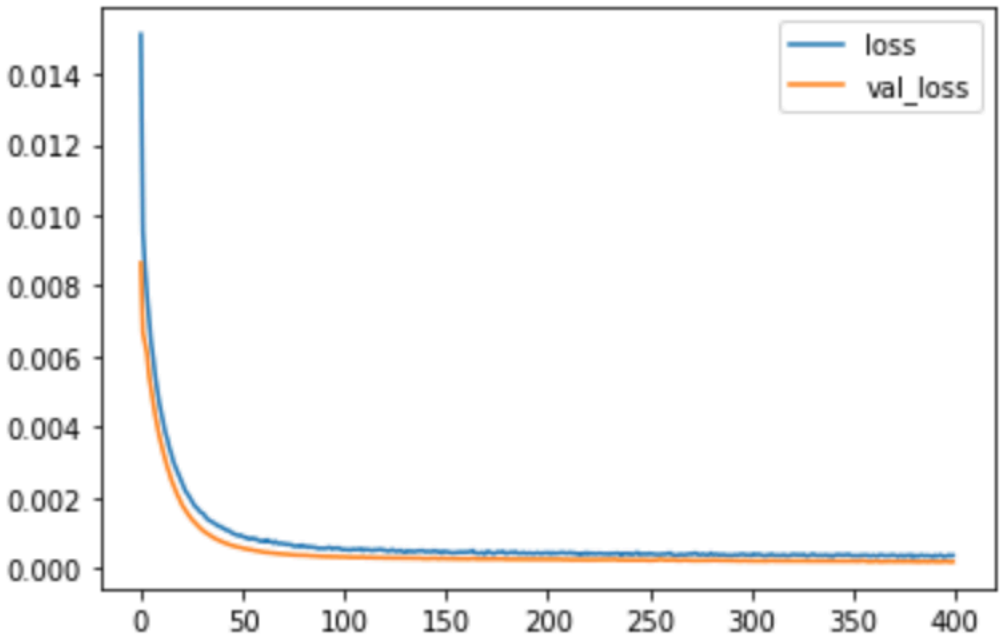
Mean Loss: 0.000767686

Mean Validation Loss: 0.000498059

Batch Size: 512

Model 5 with a larger batch size

```
model = Sequential()

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



Model 12

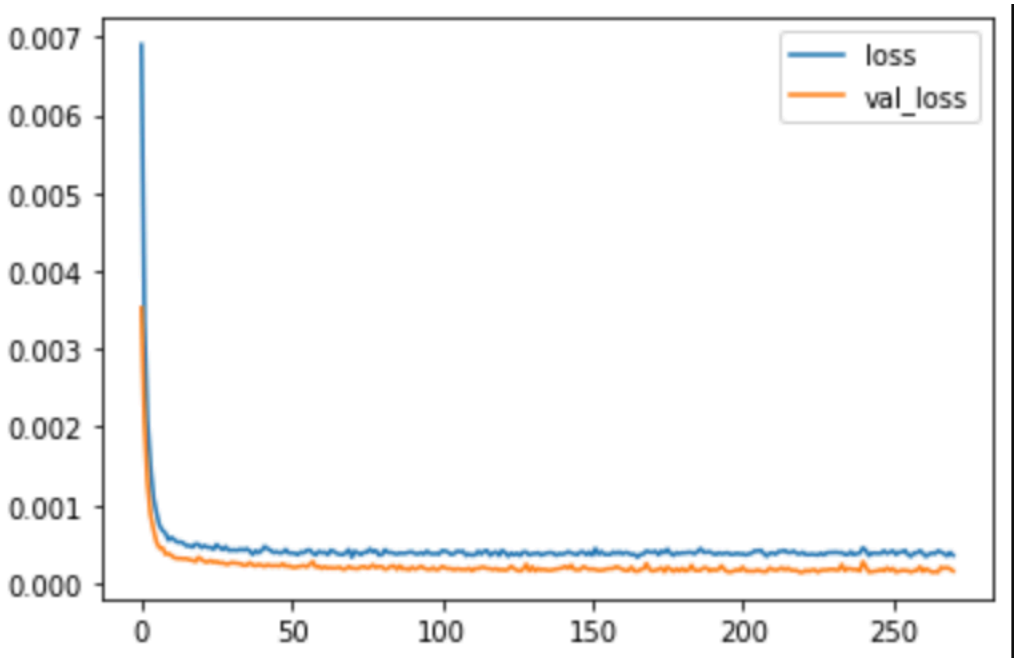
Mean Loss: 0.000448769

Mean Validation Loss: 0.000225915

Batch Size: 32

Model 9, the best so far, with more patience and epochs

```
model = Sequential()

model.add(Dense(29, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(58, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



CNN

May 17, 2020

[1]:
```
import os
import pandas as pd
import numpy as np
```

Test on a single year to ensure the method works.

[2]:
```
df = pd.read_csv('../core/tensors/games/2015.csv', header=None)
```

[3]:
```
df
```

[3]:

	0	1	2	3	4	5	6	7	8	9	...	531	\
0	0.446721	0.40	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.136311	
1	0.467213	0.40	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.212168	
2	0.409836	0.30	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.298806	
3	0.487705	0.45	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	0.185214	
4	0.508197	0.55	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.112437	
...	...	...	...	...	...	...	...	...	...	...	...	...	
2423	0.500000	0.50	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.001155	
2424	0.508197	0.50	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.041201	
2425	0.385246	0.40	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.006931	
2426	0.487705	0.30	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.033115	
2427	0.590164	0.60	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.118598	

	532	533	534	535	536	537	538	\
0	0.008721	0.140351	0.089219	0.101562	0.164557	1.0	0.164568	
1	0.007267	0.098246	0.070632	0.085938	0.065823	1.0	0.160971	
2	0.015988	0.049123	0.237918	0.414062	0.225316	1.0	0.167266	
3	0.021802	0.129825	0.052045	0.250000	0.096203	1.0	0.178058	
4	0.011628	0.056140	0.070632	0.085938	0.091139	1.0	0.163669	
...	...	...	...	...	...	...	...	
2423	0.000000	0.000000	0.007435	0.000000	0.000000	1.0	0.103417	
2424	0.000000	0.000000	0.085502	0.000000	0.020253	1.0	0.114209	
2425	0.000000	0.000000	0.022305	0.000000	0.000000	1.0	0.107014	
2426	0.000000	0.000000	0.115242	0.000000	0.002532	1.0	0.064748	
2427	0.000000	0.003509	0.271375	0.015625	0.002532	1.0	0.075540	

	539	540
0	0.116185	1.0

```
1     0.084971  1.0
2     0.124277  1.0
3     0.150867  0.0
4     0.076879  1.0
...        ...  ...
2423  0.049711  0.0
2424  0.062428  1.0
2425  0.051445  1.0
2426  0.046243  1.0
2427  0.047399  0.0

[2428 rows x 541 columns]
```

[4]:
```
game0 = list(df.iloc[0])
```

[5]:
```
len(game0)
```

[5]:
```
541
```

[6]:
```
y0 = int(game0[-1])
```

[7]:
```
y0
```

[7]:
```
1
```

[8]:
```
type(game0)
```

[8]:
```
list
```

[9]:
```
game0 = game0[:-1]
```

[10]:
```
game0 = np.reshape(game0, (18, 30, 1))
```

[11]:
```
game0.shape
```

[11]:
```
(18, 30, 1)
```

[12]:
```
game1 = list(df.iloc[1])
```

[13]:
```
y1 = int(game1[-1])
```

[14]:
```
game1 = game1[:-1]
```

[15]:
```
game1 = np.reshape(game1, (18, 30, 1))
```

[16]:
```
games = [game0, game1]
```

[17]:
```
rows = len(games)
cols = len(games[0])
fors = len(games[0][0])
last = len(games[0][0][0])
```

[18]:
```
(rows, cols, fors, last)
```

[18]: (2, 18, 30, 1)

[19]:
```
game2 = list(df.iloc[2])
y2 = int(game2[-1])
game2 = game2[:-1]
game2 = np.reshape(game2, (18, 30))
```

[20]:
```
games.append(game2)
```

[21]:
```
rows = len(games)
cols = len(games[0])
fors = len(games[0][0])
(rows, cols, fors)
```

[21]: (3, 18, 30)

We've shown that this is an effective method of getting games as 3D tables from the .csv files, as well as combining them into a list of tables. We can scale this to get a list of every game, and this will be our CNN input.

[22]:
```
df.shape
```

[22]: (2428, 541)

[23]:
```
X = []
y = []
for index in range(0, df.shape[0]):
    game = list(df.iloc[index])
    y_sample = int(game[-1])
    game = game[:-1]
    game = np.reshape(game, ([18, 30, 1]))
    X.append(game)
    y.append(y_sample)
X = np.array(X)
y = np.array(y)
```

[24]:
```
y = y[..., np.newaxis]
```

[25]:
```
X.shape
```

[25]: (2428, 18, 30, 1)

[26]:
```
y.shape
```

[26]: (2428, 1)

[27]:
```
rows = len(games)
cols = len(games[0])
fors = len(games[0][0])
last = len(games[0][0][0])
(rows, cols, fors, last)
```

[27]: (3, 18, 30, 1)

[28]:
```
len(y)
```

[28]: 2428

[29]:
```
input_shape = (18, 30, 1)
```

Now we can extrapolate this to pull all games from all years into one list. This will be the last step before running our predictive model.

[30]:
```
X = []
y = []
for year in range(1919, 2020):
    df = pd.read_csv('../core/tensors/games/{}.csv'.format(year), header=None)
    for index in range(0, df.shape[0]):
        game = list(df.iloc[index])
        y_sample = int(game[-1])
        game = game[:-1]
        game = np.reshape(game, ([18, 30, 1]))
        X.append(game)
        y.append(y_sample)
```

[31]:
```
X = np.array(X)
y = np.array(y)
```

[32]:
```
y = y[..., np.newaxis]
```

[33]:
```
rows = len(X)
cols = len(X[0])
fors = len(X[0][0])
last = len(X[0][0][0])
(rows, cols, fors, last)
```

[33]: (176687, 18, 30, 1)

[34]:
```
len(y)
```

[34]:
```
176687
```

[35]:
```
from sklearn.model_selection import train_test_split
```

[36]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[37]:
```
len(X_train)
```

[37]:
```
141349
```

[38]:
```
X_train.shape
```

[38]:
```
(141349, 18, 30, 1)
```

[39]:
```
len(X_test)
```

[39]:
```
35338
```

[40]:
```
len(y_train)
```

[40]:
```
141349
```

[41]:
```
y_train.shape
```

[41]:
```
(141349, 1)
```

[42]:
```
len(y_test)
```

[42]:
```
35338
```

[43]:
```
y_train
```

[43]:
```
array([[0],
       [0],
       [0],
       ...,
       [0],
       [1],
       [1]])
```

Building the CNN

[44]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Dropout, Flatten
```

[45]:
```
image_shape=(18, 30, 1)
```

[46]:
```
epochs = 4000
batch_size = 16
loss_param = 'binary_crossentropy'
optimizer_param = 'adam'
stop_monitor = 'val_loss'
metric = 'accuracy'
stop_patience = 20
```

[47]:
```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(1, 1),
                 input_shape=image_shape, activation='tanh'))
model.add(MaxPool2D(pool_size=(1, 1)))

model.add(Conv2D(filters=32, kernel_size=(3, 3),
                 input_shape=image_shape, activation='tanh'))
model.add(MaxPool2D(pool_size=(3, 3)))
model.add(Dropout(0.5))

model.add(Conv2D(filters=64, kernel_size=(3, 3),
                 activation='relu', padding="same"))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Conv2D(filters=128, kernel_size=(3, 3),
                 activation='relu', padding="same"))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param,
              metrics=[metric])
```

[48]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 18, 30, 32)        64
```

```
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 18, 30, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 28, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 9, 32)          0
_________________________________________________________________
dropout (Dropout)            (None, 5, 9, 32)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 5, 9, 64)          18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 4, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 2, 4, 128)         73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 1, 2, 128)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 2, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 256)               65792
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 257
=================================================================
Total params: 167,713
Trainable params: 167,713
Non-trainable params: 0
_________________________________________________________________
```

[49]:
```
from tensorflow.keras.callbacks import EarlyStopping
```

[50]:
```
early_stop = EarlyStopping(monitor=stop_monitor, patience=stop_patience)
```

[51]:
```
results = model.fit(X_train, y_train, epochs=epochs,
                            validation_data=(X_test, y_test),
                            callbacks=[early_stop]
                   )
```

```
Train on 141349 samples, validate on 35338 samples
Epoch 1/4000
141349/141349 [==============================] - 189s 1ms/sample - loss: 0.6857
- accuracy: 0.5518 - val_loss: 0.6808 - val_accuracy: 0.5633
Epoch 2/4000
141349/141349 [==============================] - 200s 1ms/sample - loss: 0.6839
- accuracy: 0.5568 - val_loss: 0.6831 - val_accuracy: 0.5560
```

```
Epoch 3/4000
141349/141349 [==============================] - 195s 1ms/sample - loss: 0.6835
- accuracy: 0.5562 - val_loss: 0.6825 - val_accuracy: 0.5595
Epoch 4/4000
141349/141349 [==============================] - 179s 1ms/sample - loss: 0.6836
- accuracy: 0.5566 - val_loss: 0.6811 - val_accuracy: 0.5591
Epoch 5/4000
141349/141349 [==============================] - 177s 1ms/sample - loss: 0.6834
- accuracy: 0.5576 - val_loss: 0.6816 - val_accuracy: 0.5570
Epoch 6/4000
141349/141349 [==============================] - 183s 1ms/sample - loss: 0.6835
- accuracy: 0.5579 - val_loss: 0.6823 - val_accuracy: 0.5572
Epoch 7/4000
141349/141349 [==============================] - 180s 1ms/sample - loss: 0.6832
- accuracy: 0.5580 - val_loss: 0.6825 - val_accuracy: 0.5627
Epoch 8/4000
141349/141349 [==============================] - 184s 1ms/sample - loss: 0.6831
- accuracy: 0.5585 - val_loss: 0.6816 - val_accuracy: 0.5615
Epoch 9/4000
141349/141349 [==============================] - 180s 1ms/sample - loss: 0.6832
- accuracy: 0.5582 - val_loss: 0.6836 - val_accuracy: 0.5626
Epoch 10/4000
141349/141349 [==============================] - 186s 1ms/sample - loss: 0.6832
- accuracy: 0.5589 - val_loss: 0.6819 - val_accuracy: 0.5623
Epoch 11/4000
141349/141349 [==============================] - 183s 1ms/sample - loss: 0.6829
- accuracy: 0.5583 - val_loss: 0.6825 - val_accuracy: 0.5622
Epoch 12/4000
141349/141349 [==============================] - 186s 1ms/sample - loss: 0.6829
- accuracy: 0.5588 - val_loss: 0.6813 - val_accuracy: 0.5596
Epoch 13/4000
141349/141349 [==============================] - 187s 1ms/sample - loss: 0.6829
- accuracy: 0.5566 - val_loss: 0.6816 - val_accuracy: 0.5620
Epoch 14/4000
141349/141349 [==============================] - 186s 1ms/sample - loss: 0.6830
- accuracy: 0.5577 - val_loss: 0.6818 - val_accuracy: 0.5612
Epoch 15/4000
141349/141349 [==============================] - 178s 1ms/sample - loss: 0.6830
- accuracy: 0.5586 - val_loss: 0.6831 - val_accuracy: 0.5614
Epoch 16/4000
141349/141349 [==============================] - 178s 1ms/sample - loss: 0.6828
- accuracy: 0.5584 - val_loss: 0.6828 - val_accuracy: 0.5634
Epoch 17/4000
141349/141349 [==============================] - 180s 1ms/sample - loss: 0.6829
- accuracy: 0.5576 - val_loss: 0.6812 - val_accuracy: 0.5614
Epoch 18/4000
141349/141349 [==============================] - 177s 1ms/sample - loss: 0.6828
- accuracy: 0.5585 - val_loss: 0.6817 - val_accuracy: 0.5595
```

```
Epoch 19/4000
141349/141349 [==============================] - 180s 1ms/sample - loss: 0.6829
- accuracy: 0.5584 - val_loss: 0.6834 - val_accuracy: 0.5610
Epoch 20/4000
141349/141349 [==============================] - 181s 1ms/sample - loss: 0.6829
- accuracy: 0.5584 - val_loss: 0.6830 - val_accuracy: 0.5531
Epoch 21/4000
141349/141349 [==============================] - 178s 1ms/sample - loss: 0.6829
- accuracy: 0.5593 - val_loss: 0.6813 - val_accuracy: 0.5588
```

[52]:
```
losses = model.history.history
losses['loss'] = np.asarray(losses['loss'])
losses['val_loss'] = np.asarray(losses['val_loss'])
final_number_of_epochs = len(losses['loss'])
min_loss = losses['loss'].min()
mean_loss = losses['loss'].mean()
final_loss = losses['loss'][-1]
min_val_loss = losses['val_loss'].min()
mean_val_loss = losses['val_loss'].mean()
final_val_loss = losses['val_loss'][-1]

def get_model_summary():
    output = []
    model.summary(print_fn=lambda line: output.append(line))
    return str(output).strip('[]')


summary = get_model_summary()

record = {
    'Epochs': final_number_of_epochs,
    'Batch_Size': batch_size,
    'Loss_Func': loss_param,
    'Optimizer': optimizer_param,
    'Early_Stop_Monitor': stop_monitor,
    'Early_Stop_Patience': stop_patience,
    'Min_Loss': min_loss,
    'Mean_Loss': mean_loss,
    'Final_Loss': final_loss,
    'Min_Val_Loss': min_val_loss,
    'Mean_Val_Loss': mean_val_loss,
    'Final_Val_Loss': final_val_loss,
    'Model': summary
}

new_data = pd.DataFrame(record, index=[0])
```

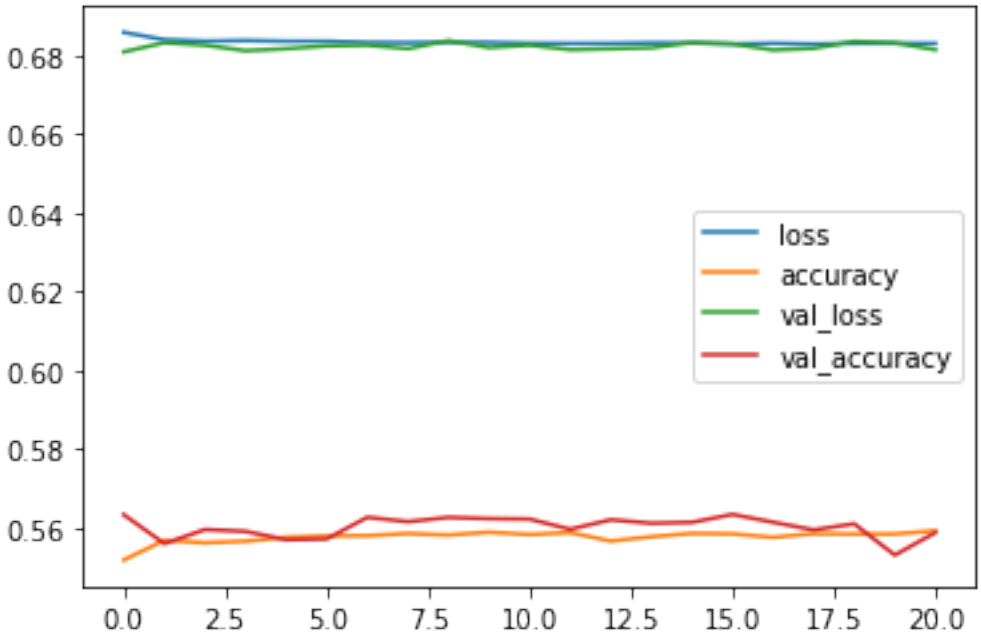
```
if os.path.exists('../core/records/game_predictions.csv'):
    df_records = pd.read_csv('../core/records/game_predictions.csv')
    df_records = df_records.append(new_data)
else:
    df_records = pd.DataFrame(new_data)

df_records.to_csv('../core/records/game_predictions.csv',
                  index=False, float_format='%g')


model.save('../core/models/model_games.h5')
```

Model Evaluation

[53]:
```
losses = pd.DataFrame(model.history.history)
```

[54]:
```
losses.plot()
```

[54]: <matplotlib.axes._subplots.AxesSubplot at 0x197e53250>

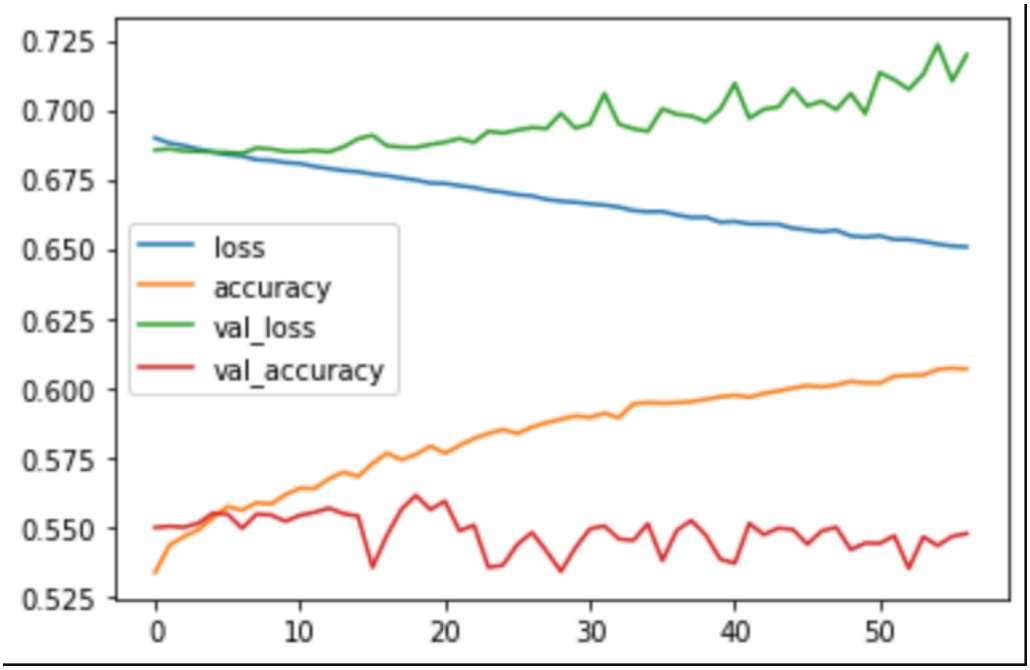


[ ]:

Model 1

Mean Loss:	0.668175
Mean Validation Loss:	0.695348
Batch Size:	32

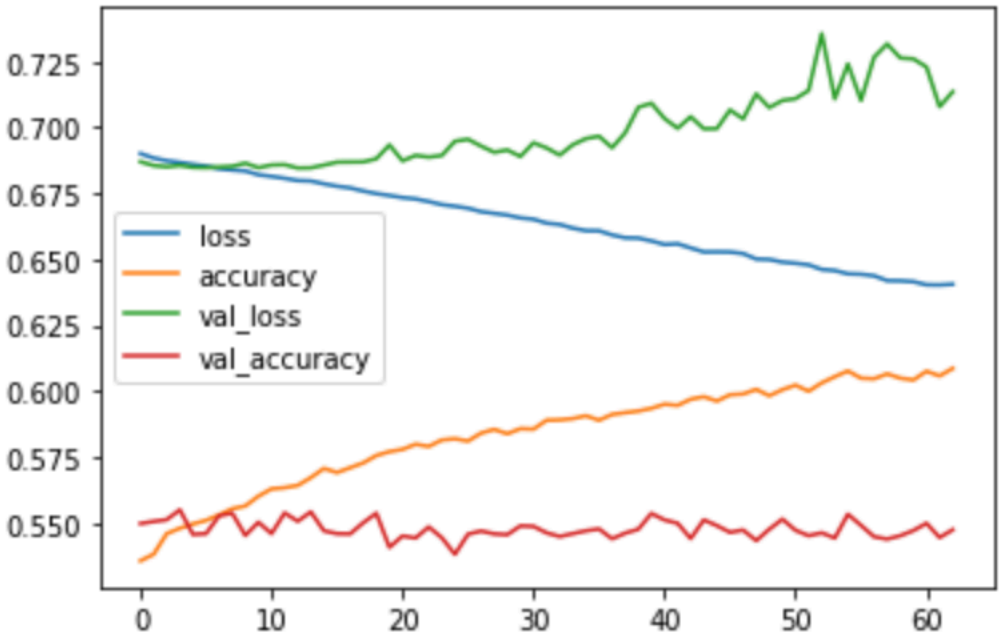
```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=image_shape, activation='relu'))
model.add(MaxPool2D(pool_size=(3, 3)))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param,
              metrics=[metric])
```



Model 2

Mean Loss:	0.664054
Mean Validation Loss:	0.69857
Batch Size:	32

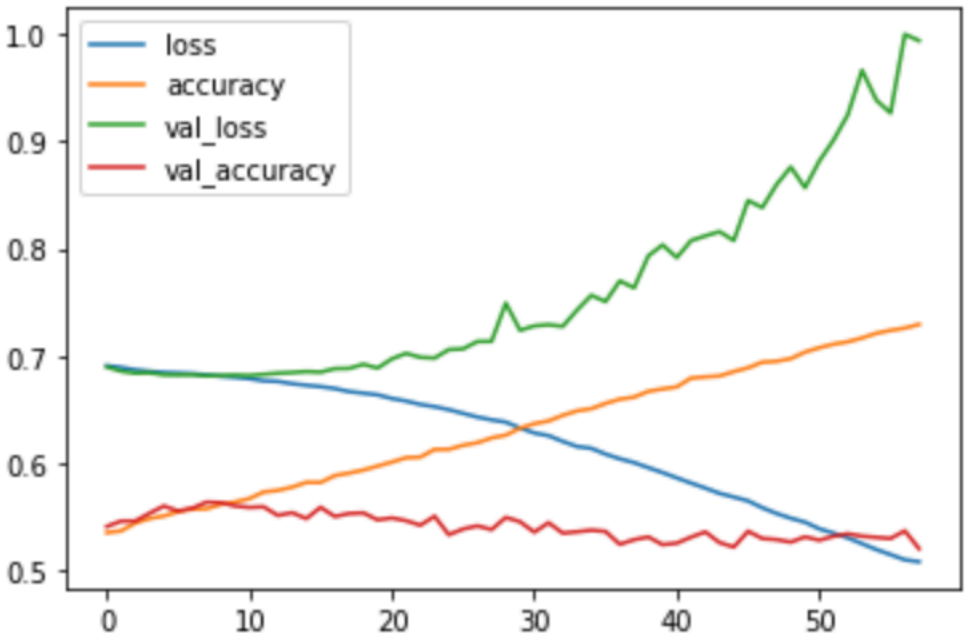
```
model = Sequential()

model.add(Conv2D(filters=64, kernel_size=(3, 3), input_shape=image_shape, activation='relu'))
model.add(MaxPool2D(pool_size=(3, 3)))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param,
              metrics=[metric])
```



Model 3

Mean Loss:	0.620608
Mean Validation Loss:	0.762239
Batch Size:	16

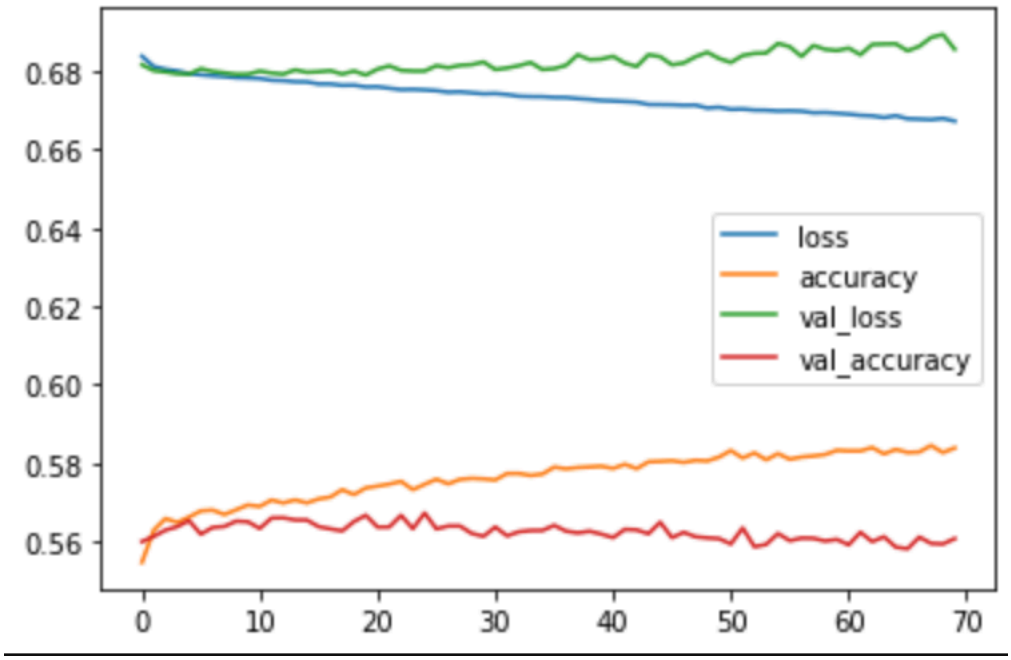
```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=image_shape, activation='relu'))
model.add(MaxPool2D(pool_size=(3, 3)))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding="same"))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding="same"))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param,
              metrics=[metric])
```



Model 4

Mean Loss:	0.673568
Mean Validation Loss:	0.682452
Batch Size:	128

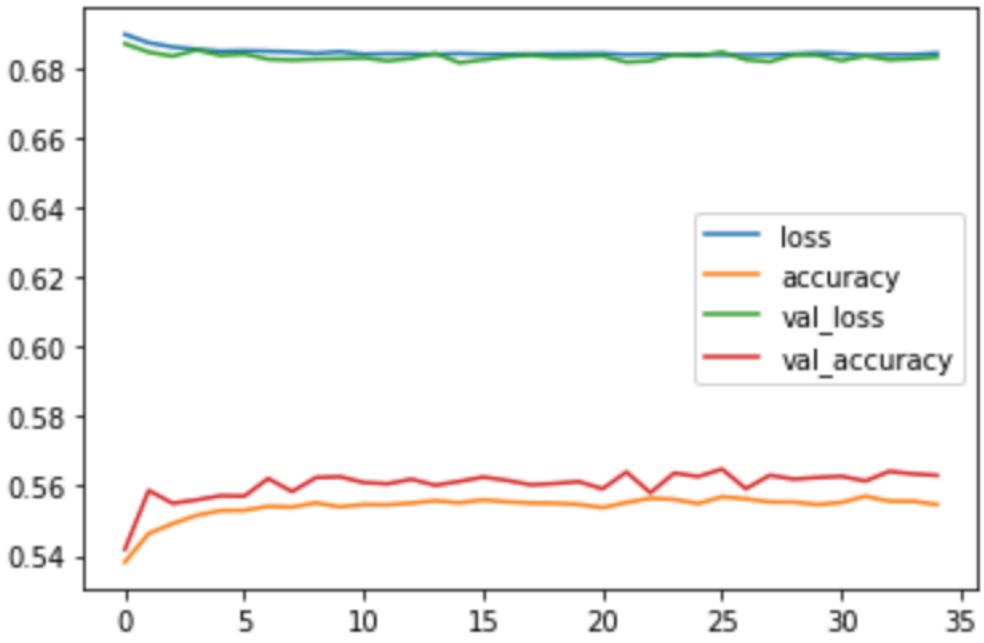
```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=image_shape, activation='relu'))
model.add(MaxPool2D(pool_size=(3, 3)))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding="same"))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param,
              metrics=[metric])
```



Model 5

Mean Loss:	0.684353
Mean Validation Loss:	0.683085
Batch Size:	128

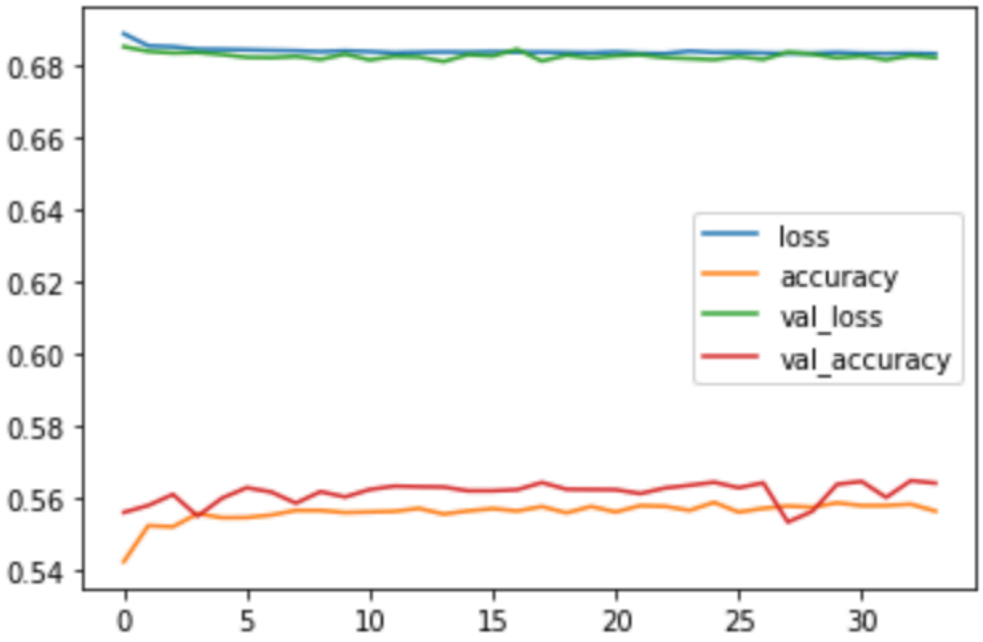
```
model = Sequential()

model.add(Conv2D(filters=8, kernel_size=(3, 3), input_shape=image_shape, activation='relu'))
model.add(MaxPool2D(pool_size=(3, 3)))
model.add(Dropout(0.5))

model.add(Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding="same"))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param,
              metrics=[metric])
```



Model 6

Mean Loss:	0.68361
Mean Validation Loss:	0.682252
Batch Size:	512

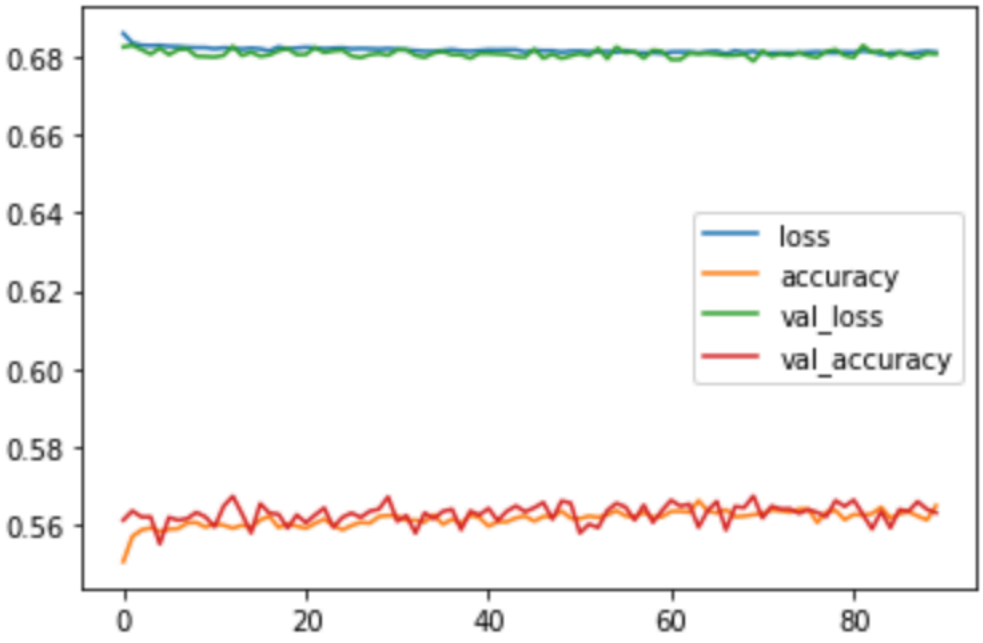
```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=image_shape, activation='relu'))
model.add(MaxPool2D(pool_size=(3, 3)))
model.add(Dropout(0.5))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding="same"))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param,
              metrics=[metric])
```



Model 7

Mean Loss:	0.681508
Mean Validation Loss:	0.68063
Batch Size:	128

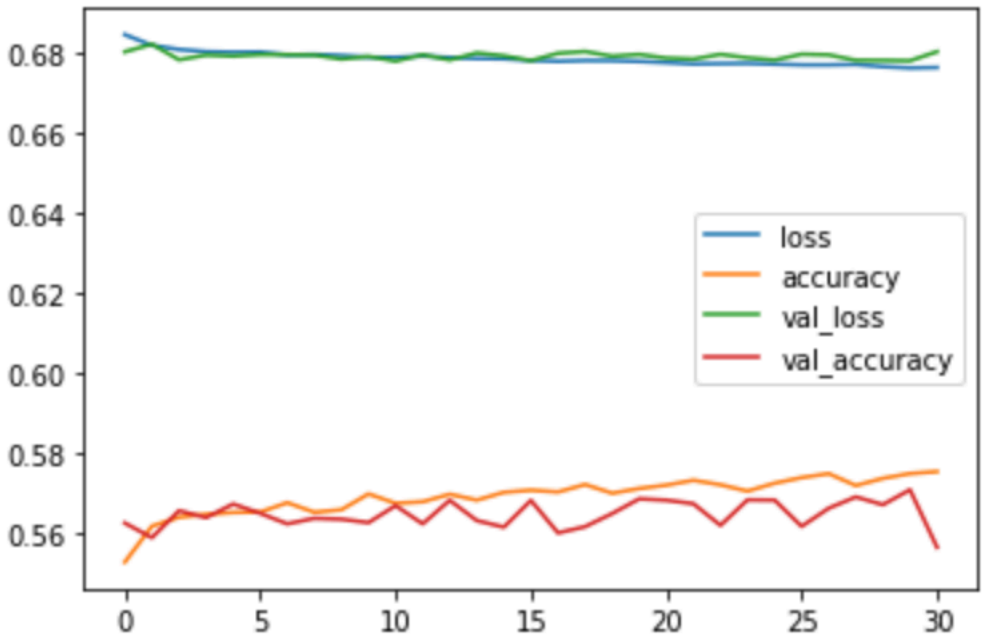
```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=image_shape, activation='tanh'))
model.add(MaxPool2D(pool_size=(3, 3)))
model.add(Dropout(0.2))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding="same"))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding="same"))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param,
              metrics=[metric])
```



Model 8

Mean Loss:	0.678727
Mean Validation Loss:	0.679328
Batch Size:	512

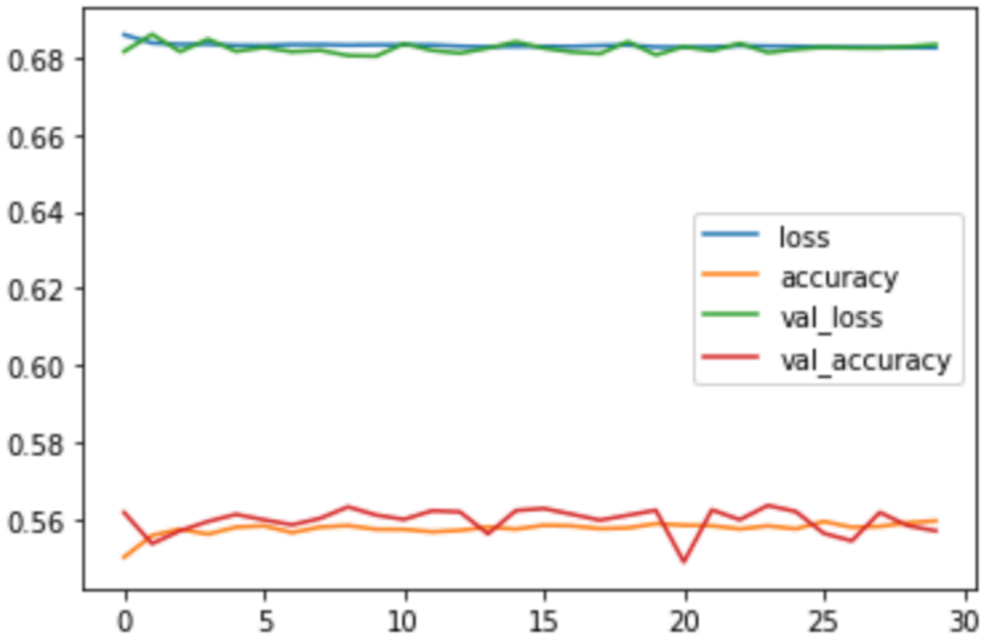
```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(1, 1), input_shape=image_shape, activation='tanh'))
model.add(MaxPool2D(pool_size=(1, 1)))

model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=image_shape,
                 activation='tanh', padding="same"))
model.add(MaxPool2D(pool_size=(3, 3)))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding="same"))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding="same"))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param,
              metrics=[metric])
```



Model 9

Mean Loss:	0.683104
Mean Validation Loss:	0.682327
Batch Size:	16

```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(1, 1),
                 input_shape=image_shape, activation='tanh'))
model.add(MaxPool2D(pool_size=(1, 1)))

model.add(Conv2D(filters=32, kernel_size=(3, 3),
                 input_shape=image_shape, activation='tanh'))
model.add(MaxPool2D(pool_size=(3, 3)))
model.add(Dropout(0.5))

model.add(Conv2D(filters=64, kernel_size=(3, 3),
                 activation='relu', padding="same"))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Conv2D(filters=128, kernel_size=(3, 3),
                 activation='relu', padding="same"))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param,
              metrics=[metric])
```



ANN

May 17, 2020

[1]:
```
import os
import joblib
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras import regularizers
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.models import Sequential
```

[2]:
```
X = []
y = []
for year in range(1919, 2020):
    df = pd.read_csv('../core/tensors/games/{}.csv'.format(year), header=None)
    for index in range(0, df.shape[0]):
        game = list(df.iloc[index])
        y_sample = int(game[-1])
        game = game[:-1]
        X.append(game)
        y.append(y_sample)
X = np.array(X)
y = np.array(y)
y = y[..., np.newaxis]
```

[3]:
```
X.shape
```

[3]:
```
(176687, 540)
```

[4]:
```
y.shape
```

[4]:
```
(176687, 1)
```

[5]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

[6]:
```
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[7]:
```
epochs = 4000
batch_size = 2048
loss_param = 'binary_crossentropy'
optimizer_param = 'adam'
stop_monitor = 'val_loss'
metric = 'accuracy'
stop_patience = 15
```

[8]:
```
model = Sequential()

model.add(Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.
 ↪0001),
                input_shape=X_train.shape[1:]))
model.add(Dropout(0.5))

model.add(Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.
 ↪0001)))
model.add(Dropout(0.5))

model.add(Dense(32, activation='relu', kernel_regularizer=regularizers.l2(0.
 ↪0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```

[9]:
```
early_stop = EarlyStopping(monitor=stop_monitor, patience=stop_patience)
```

[11]:
```
results = model.fit(x=X_train, y=y_train,
                    epochs=epochs,
                    batch_size=batch_size,
                    validation_data=(X_test, y_test),
                    callbacks=[early_stop]
                    )
```

```
Train on 141349 samples, validate on 35338 samples
Epoch 1/4000
141349/141349 [==============================] - 2s 16us/sample - loss: 0.7246 -
val_loss: 0.7154
Epoch 2/4000
141349/141349 [==============================] - 1s 10us/sample - loss: 0.7119 -
val_loss: 0.7068
```

```
Epoch 3/4000
141349/141349 [==============================] - 1s 9us/sample - loss: 0.7048 -
val_loss: 0.7016
Epoch 4/4000
141349/141349 [==============================] - 1s 9us/sample - loss: 0.7004 -
val_loss: 0.6975
Epoch 5/4000
141349/141349 [==============================] - 1s 8us/sample - loss: 0.6967 -
val_loss: 0.6958
Epoch 6/4000
141349/141349 [==============================] - 1s 8us/sample - loss: 0.6949 -
val_loss: 0.6939
Epoch 7/4000
141349/141349 [==============================] - 1s 8us/sample - loss: 0.6927 -
val_loss: 0.6912
Epoch 8/4000
141349/141349 [==============================] - 1s 9us/sample - loss: 0.6909 -
val_loss: 0.6900
Epoch 9/4000
141349/141349 [==============================] - 1s 8us/sample - loss: 0.6893 -
val_loss: 0.6904
Epoch 10/4000
141349/141349 [==============================] - 1s 9us/sample - loss: 0.6882 -
val_loss: 0.6879
Epoch 11/4000
141349/141349 [==============================] - 2s 12us/sample - loss: 0.6875 -
val_loss: 0.6886
Epoch 12/4000
141349/141349 [==============================] - 1s 10us/sample - loss: 0.6870 -
val_loss: 0.6868
Epoch 13/4000
141349/141349 [==============================] - 1s 9us/sample - loss: 0.6861 -
val_loss: 0.6882
Epoch 14/4000
141349/141349 [==============================] - 1s 9us/sample - loss: 0.6860 -
val_loss: 0.6869
Epoch 15/4000
141349/141349 [==============================] - 1s 9us/sample - loss: 0.6850 -
val_loss: 0.6865
Epoch 16/4000
141349/141349 [==============================] - 1s 9us/sample - loss: 0.6848 -
val_loss: 0.6882
Epoch 17/4000
141349/141349 [==============================] - 1s 9us/sample - loss: 0.6857 -
val_loss: 0.6868
Epoch 18/4000
141349/141349 [==============================] - 1s 10us/sample - loss: 0.6846 -
val_loss: 0.6860
```

```
Epoch 19/4000
141349/141349 [==============================] - 1s 10us/sample - loss: 0.6847 -
val_loss: 0.6848
Epoch 20/4000
141349/141349 [==============================] - 1s 10us/sample - loss: 0.6844 -
val_loss: 0.6867
Epoch 21/4000
141349/141349 [==============================] - 1s 10us/sample - loss: 0.6837 -
val_loss: 0.6853
Epoch 22/4000
141349/141349 [==============================] - 1s 10us/sample - loss: 0.6841 -
val_loss: 0.6864
Epoch 23/4000
141349/141349 [==============================] - 1s 9us/sample - loss: 0.6834 -
val_loss: 0.6858
Epoch 24/4000
141349/141349 [==============================] - 1s 9us/sample - loss: 0.6833 -
val_loss: 0.6866
Epoch 25/4000
141349/141349 [==============================] - 1s 9us/sample - loss: 0.6830 -
val_loss: 0.6851
Epoch 26/4000
141349/141349 [==============================] - 1s 10us/sample - loss: 0.6839 -
val_loss: 0.6856
Epoch 27/4000
141349/141349 [==============================] - 1s 10us/sample - loss: 0.6838 -
val_loss: 0.6867
Epoch 28/4000
141349/141349 [==============================] - 1s 10us/sample - loss: 0.6829 -
val_loss: 0.6838
Epoch 29/4000
141349/141349 [==============================] - 1s 9us/sample - loss: 0.6829 -
val_loss: 0.6850
Epoch 30/4000
141349/141349 [==============================] - 1s 9us/sample - loss: 0.6831 -
val_loss: 0.6839
Epoch 31/4000
141349/141349 [==============================] - 1s 9us/sample - loss: 0.6828 -
val_loss: 0.6857
Epoch 32/4000
141349/141349 [==============================] - 1s 9us/sample - loss: 0.6833 -
val_loss: 0.6852
Epoch 33/4000
141349/141349 [==============================] - 1s 10us/sample - loss: 0.6827 -
val_loss: 0.6862
Epoch 34/4000
141349/141349 [==============================] - 2s 13us/sample - loss: 0.6837 -
val_loss: 0.6845
```

```
Epoch 35/4000
141349/141349 [==============================] - 2s 11us/sample - loss: 0.6834 -
val_loss: 0.6870
Epoch 36/4000
141349/141349 [==============================] - 1s 9us/sample - loss: 0.6830 -
val_loss: 0.6841
Epoch 37/4000
141349/141349 [==============================] - 1s 9us/sample - loss: 0.6831 -
val_loss: 0.6840
Epoch 38/4000
141349/141349 [==============================] - 1s 9us/sample - loss: 0.6826 -
val_loss: 0.6860
Epoch 39/4000
141349/141349 [==============================] - 1s 9us/sample - loss: 0.6823 -
val_loss: 0.6848
Epoch 40/4000
141349/141349 [==============================] - 1s 9us/sample - loss: 0.6830 -
val_loss: 0.6845
Epoch 41/4000
141349/141349 [==============================] - 1s 9us/sample - loss: 0.6828 -
val_loss: 0.6863
Epoch 42/4000
141349/141349 [==============================] - 1s 10us/sample - loss: 0.6832 -
val_loss: 0.6860
Epoch 43/4000
141349/141349 [==============================] - 1s 11us/sample - loss: 0.6823 -
val_loss: 0.6864
```

[12]:
```
losses = model.history.history
losses['loss'] = np.asarray(losses['loss'])
losses['val_loss'] = np.asarray(losses['val_loss'])
final_number_of_epochs = len(losses['loss'])
min_loss = losses['loss'].min()
mean_loss = losses['loss'].mean()
final_loss = losses['loss'][-1]
min_val_loss = losses['val_loss'].min()
mean_val_loss = losses['val_loss'].mean()
final_val_loss = losses['val_loss'][-1]


def get_model_summary():
    output = []
    model.summary(print_fn=lambda line: output.append(line))
    return str(output).strip('[]')


summary = get_model_summary()
```

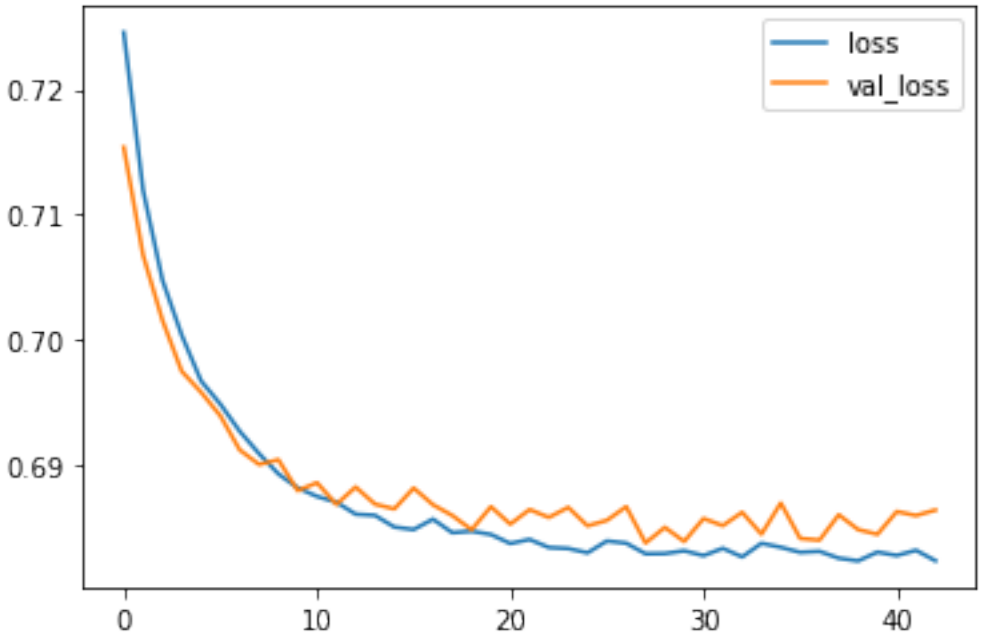
```
record = {
    'Epochs': final_number_of_epochs,
    'Batch_Size': batch_size,
    'Loss_Func': loss_param,
    'Optimizer': optimizer_param,
    'Early_Stop_Monitor': stop_monitor,
    'Early_Stop_Patience': stop_patience,
    'Min_Loss': min_loss,
    'Mean_Loss': mean_loss,
    'Final_Loss': final_loss,
    'Min_Val_Loss': min_val_loss,
    'Mean_Val_Loss': mean_val_loss,
    'Final_Val_Loss': final_val_loss,
    'Model': summary
}
```

[13]:
```
new_data = pd.DataFrame(record, index=[0])
if os.path.exists('../core/records/games_ann.csv'):
    df_records = pd.read_csv('../core/records/games_ann.csv')
    df_records = df_records.append(new_data)
else:
    df_records = pd.DataFrame(new_data)

df_records.to_csv('../core/records/games_ann.csv',
                  index=False, float_format='%g')

model.save('../core/models/games_ann.h5')
```

[14]:
```
losses = pd.DataFrame(model.history.history)
```

[15]:
```
losses.plot()
```

[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x139561a50>
```


loss
val_loss
0.72
0.71
0.70
0.69
0
10
20
30
40

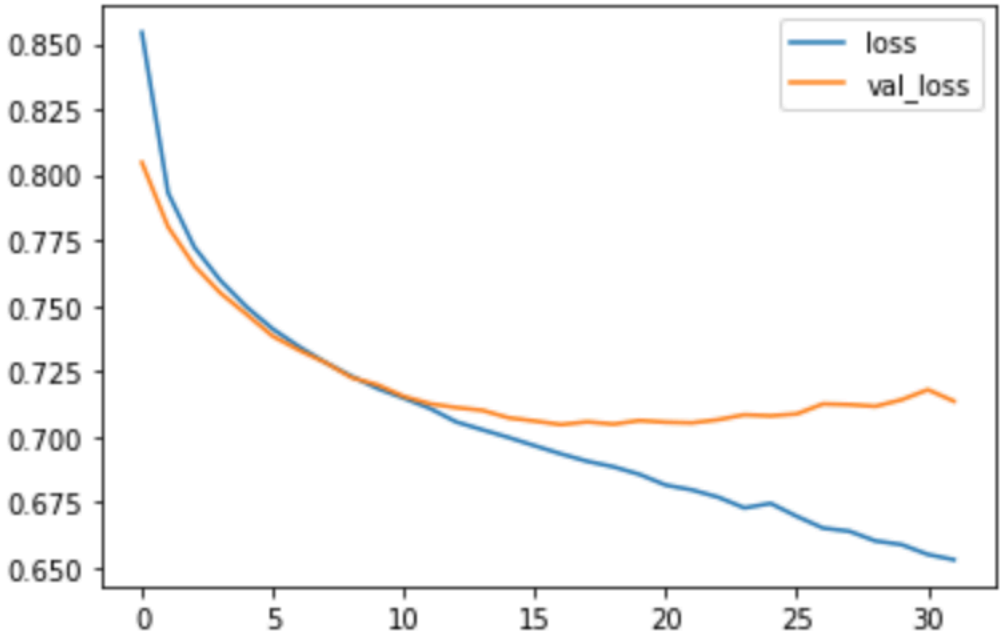
[ ]:

[ ]:

Model 1

Mean Loss:	0.705395
Mean Validation Loss:	0.721903
Batch Size:	5096

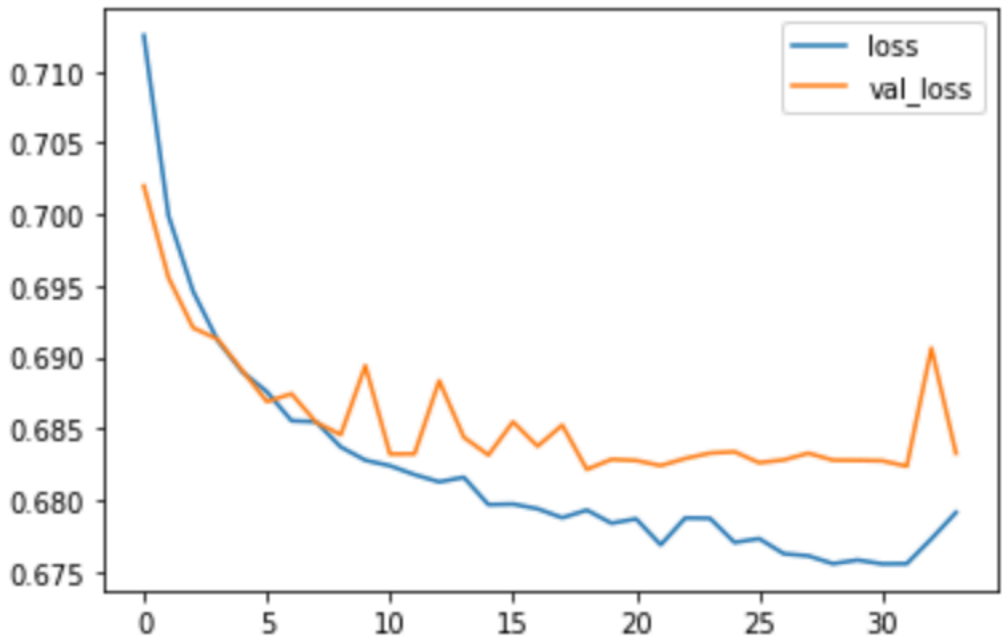
```
model = Sequential()

model.add(Dense(1024, activation='relu', kernel_regularizer=regularizers.l2(0.0001),
                input_shape=X_train.shape[1:]))
model.add(Dropout(0.2))

model.add(Dense(512, activation='relu',
                kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.2))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



Model 2

Mean Loss:	0.682143
Mean Validation Loss:	0.690175
Batch Size:	2048

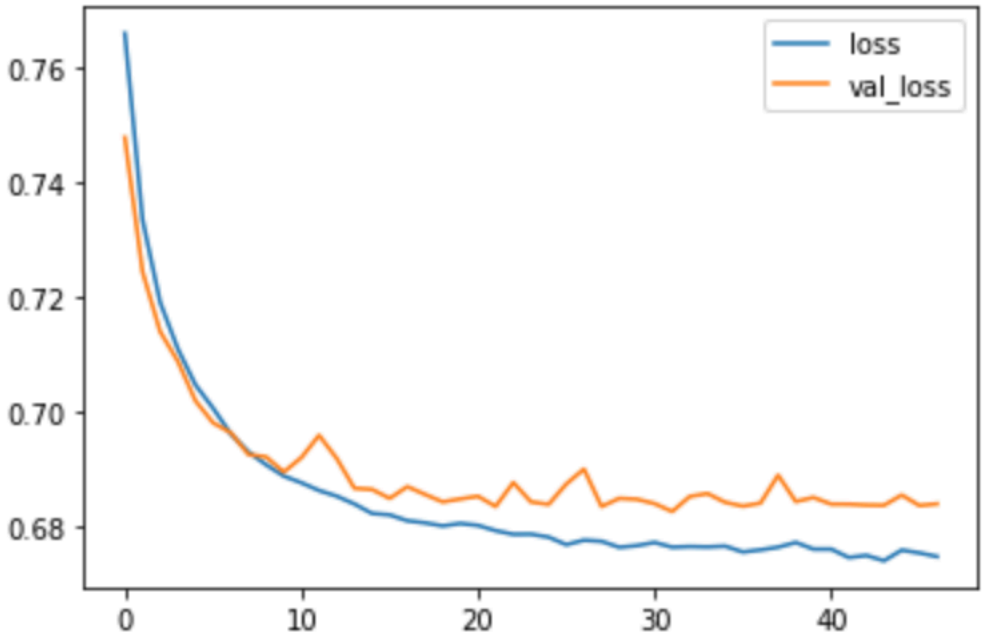
```
model = Sequential()

model.add(Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.0001),
                input_shape=X_train.shape[1:]))
model.add(Dropout(0.2))

model.add(Dense(64, activation='relu',
                kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.2))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



Model 3

Mean Loss:	0.685215
Mean Validation Loss:	0.690175
Batch Size:	2048

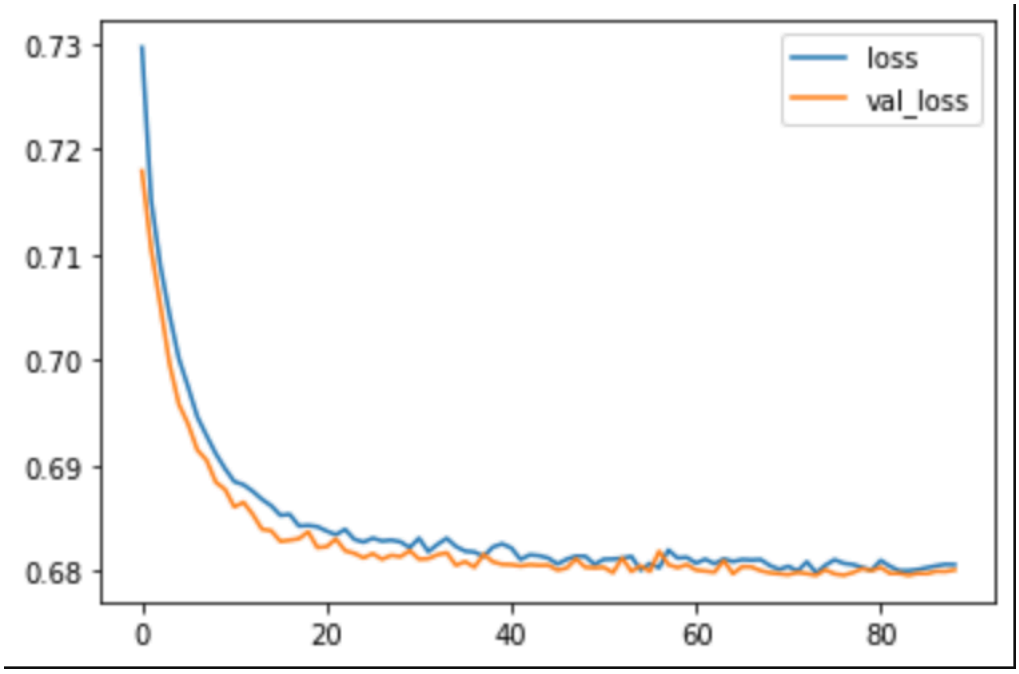
```
model = Sequential()

model.add(Dense(512, activation='relu', kernel_regularizer=regularizers.l2(0.0001),
                input_shape=X_train.shape[1:]))
model.add(Dropout(0.2))

model.add(Dense(256, activation='relu',
                kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.2))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



Model 4

Mean Loss:	0.684281
Mean Validation Loss:	0.682942
Batch Size:	2048

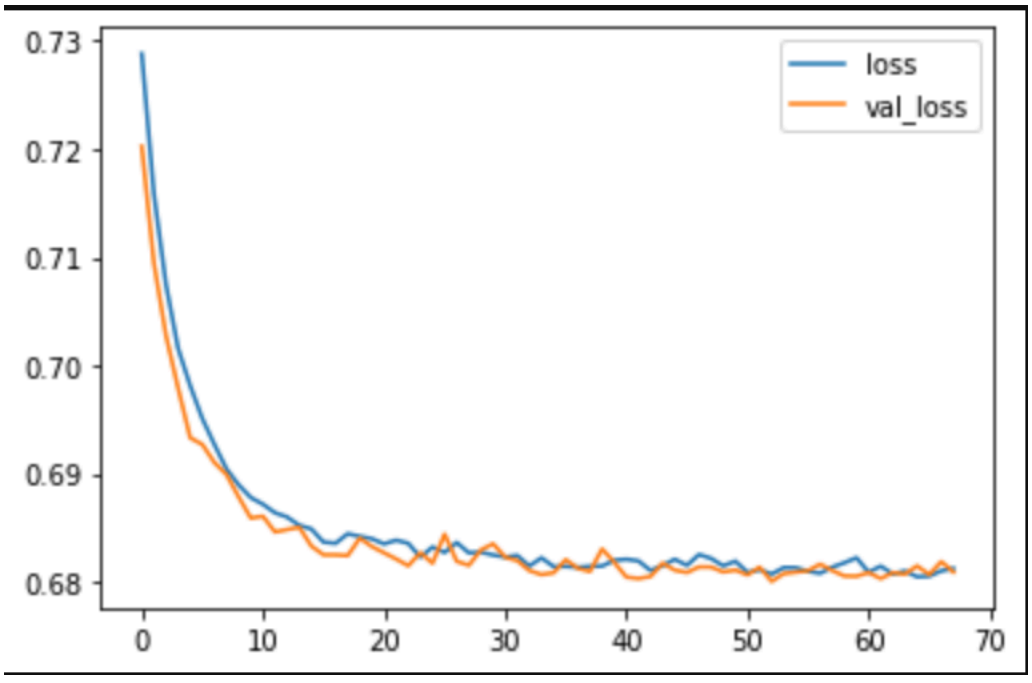
```
model = Sequential()

model.add(Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.0001),
                input_shape=X_train.shape[1:]))
model.add(Dropout(0.5))

model.add(Dense(64, activation='relu',
                kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(32, activation='relu',
                kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



Model 5

Mean Loss:	0.685052
Mean Validation Loss:	0.68408
Batch Size:	1024

```
model = Sequential()

model.add(Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.0001),
                input_shape=X_train.shape[1:]))
model.add(Dropout(0.5))

model.add(Dense(64, activation='relu',
                kernel_regularizer=regularizers.l2(0.0001)))

model.add(Dense(64, activation='relu',
                kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(32, activation='relu',
                kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```

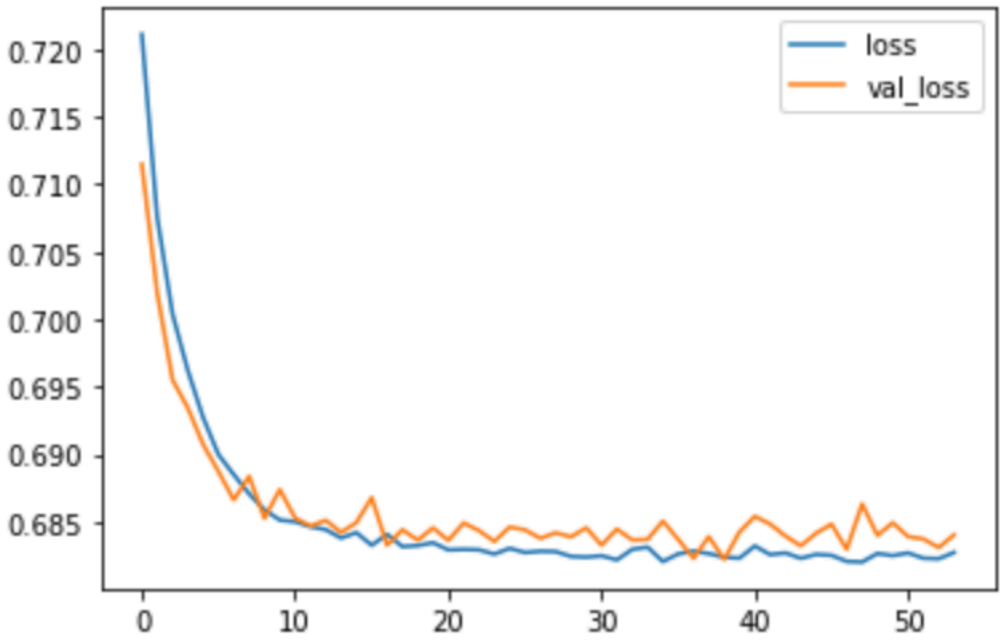


Model 6

Mean Loss:	0.685211
Mean Validation Loss:	0.685804
Batch Size:	1024

Scaling with MinMax normalization from now on

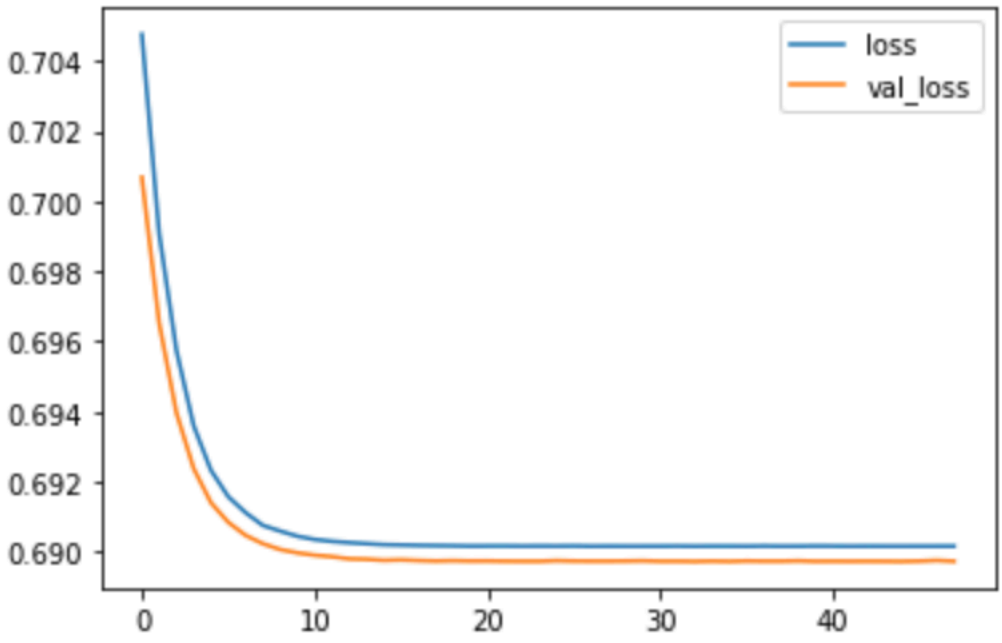
```
model = Sequential()

model.add(Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.0001),
                input_shape=X_train.shape[1:]))
model.add(Dropout(0.5))

model.add(Dense(64, activation='relu',
                kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(32, activation='relu',
                kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



Model 7

Mean Loss:	0.690971
Mean Validation Loss:	0.69035
Batch Size:	2048

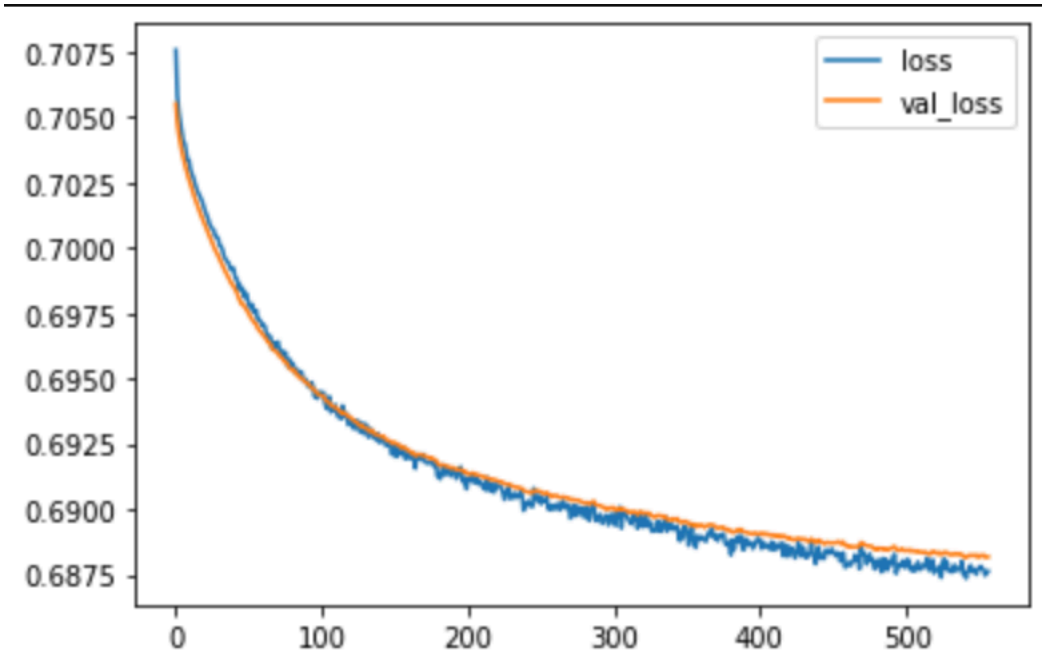
```
model = Sequential()

model.add(Dense(16, activation='relu', kernel_regularizer=regularizers.l2(0.0001),
                input_shape=X_train.shape[1:]))
model.add(Dropout(0.5))

model.add(Dense(32, activation='relu', kernel_regularizer=regularizers.l2(0.0001),
                input_shape=X_train.shape[1:]))
model.add(Dropout(0.5))

model.add(Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.0001),
                input_shape=X_train.shape[1:]))
model.add(Dropout(0.5))

model.add(Dense(32, activation='relu',
                kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(16, activation='relu',
                kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



Model 7

Mean Loss: 0.691329
Mean Validation Loss: 0.691562
Batch Size: 32
Adagrad optimization

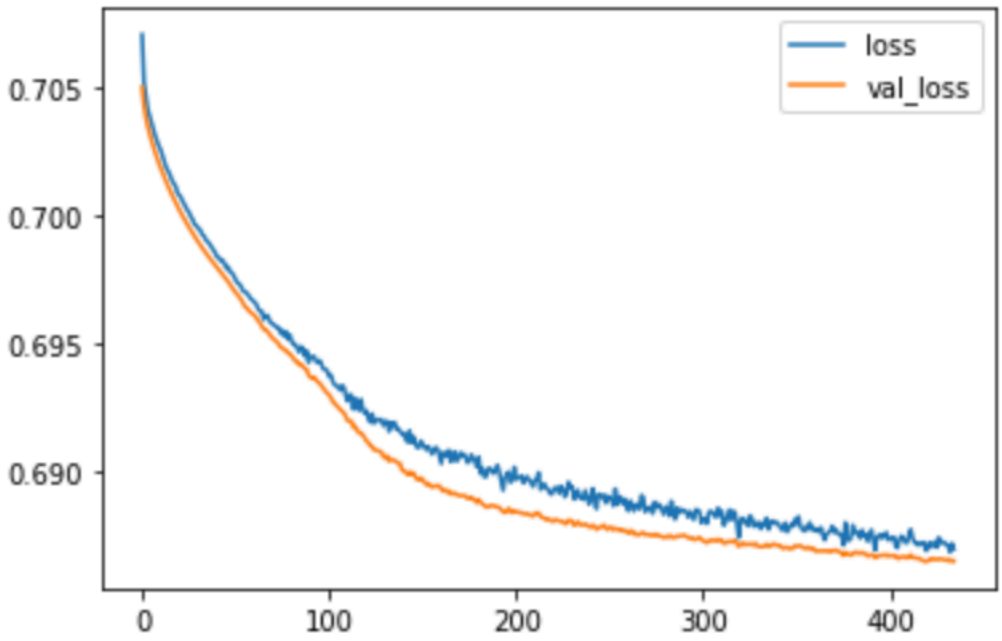
```
model = Sequential()

model.add(Dense(16, activation='relu', kernel_regularizer=regularizers.l2(0.0001),
                input_shape=X_train.shape[1:]))
model.add(Dropout(0.5))

model.add(Dense(32, activation='relu', kernel_regularizer=regularizers.l2(0.0001),
                input_shape=X_train.shape[1:]))
model.add(Dropout(0.5))

model.add(Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.0001),
                input_shape=X_train.shape[1:]))
model.add(Dropout(0.5))

model.add(Dense(32, activation='relu',
                kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(16, activation='relu',
                kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



Model 8

Mean Loss: 0.691172
Mean Validation Loss: 0.69024
Batch Size: 16
Adagrad optimization

```
model = Sequential()

model.add(Dense(16, activation='relu', kernel_regularizer=regularizers.l2(0.0001),
                input_shape=X_train.shape[1:]))
model.add(Dropout(0.5))

model.add(Dense(32, activation='relu', kernel_regularizer=regularizers.l2(0.0001),
                input_shape=X_train.shape[1:]))
model.add(Dropout(0.5))

model.add(Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.0001),
                input_shape=X_train.shape[1:]))
model.add(Dropout(0.5))

model.add(Dense(32, activation='relu',
                kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(16, activation='relu',
                kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param)
```



Acknowledgements

Thank you to Gianluca Pollastri for, in addition to serving as my advisor, teaching me almost everything I know about neural networks.

Thank you to James Loy for his insightful tutorials on building neural networks in Python, as well as his book on the same subject matter (*Neural Network Projects in Python*). Also, thank you to Kirill Eremenko and Jose Portilla at Udemy for their courses in deep learning and web development courses in Python, respectively,

Thanks to Nate Silver, Michael Lewis, Jay Boice, Bill James and Billy Beane for making data analytics in baseball what it is today and opening the sport to a new world of evaluation.

Thanks to Sean Lahman and the people at The Sean Lahman Baseball Database; thank you also to the people at Retrosheet, FanGraphs, Baseball Prospectus and MLB Advanced Media for their collection, compilation, integration and organization of all relevant baseball statistics.

Finally: thank you to Bryan Tiebert for fostering my love of the sport and Shannon Kauffman for the same, as well as her continued support in this endeavor.

Bibliography

1. Society for American Baseball Research. *The SABR Story* https://sabr.org (2019).
2. James, B. *The New Bill James Historical Baseball Abstract* Free Press, 1–1024. ISBN: 978-0743227223 (Simon and Schuster, 2010).
3. Yakovenko, N. *Machine Learning for Baseball: My Story* 2017. https://medium.com/@Moscow25/machine-learning-for-baseball-my-story-84dbe075e4b1 (2019).
4. Bezdomny, I. *How do you know if he'll be unhittable?* 2010. http://bezdomnybaseball.blogspot.com/2010/02/how-do-you-know-hes-unhittable.html (2019).
5. Bezdomny, I. *League differences in strikeout rates (a more rigorous treatment)* 2010. http://bezdomnybaseball.blogspot.com/2010/02/in-my-last-post-brian-cashman-to-stop.html (2019).
6. Tango, T. *Defensive Responsibility Spectrum (DRS)* http://www.tangotiger.net/drspectrum.html (2019).
7. Grosnick, B. *Basic Sabermetrics: Fielding Independent Pitching* 2013. https://www.beyondtheboxscore.com/2013/11/19/5121600/what-is-fip-fielding-independent-pitching-basic-sabermetrics (2019).
8. Major League Baseball. *Glossary* http://m.mlb.com/glossary (2019).
9. Fangraphs. *Fangraphs Sabermetrics Library* https://library.fangraphs.com/ (2019).
10. Slowinski, S. *WAR for Pitchers* 2012. https://library.fangraphs.com/war/calculating-war-pitchers/ (2019).
11. Tango, T. *Game Score Version 2.0* 2016. https://blogs.fangraphs.com/instagraphs/game-score-v2-0/ (2019).
12. Tango, T. *Path to Game Score 2.0* 2014. http://tangotiger.com/index.php/site/comments/path-to-game-score-2.0-part-3-of-3 (2019).
13. Boice, J. & Silver, N. *How Our MLB Predictions Work* 2019. https://fivethirtyeight.com/methodology/how-our-mlb-predictions-work/ (2019).
14. Thorn, J., Palmer, P. & Reuther, D. *The Hidden Game of Baseball* 1–440. ISBN: 9780226242484. https://www.press.uchicago.edu/ucp/books/book/chicago/H/bo19782299.html (The University of Chicago Press, Chicago, 2015).
15. Grosnick, B. *Separate but not Quite Equal: Why OPS is a Bad Statistic* 2015. https://www.beyondtheboxscore.com/2015/9/18/9329763/separate-but-not-quite-equal-why-ops-is-a-bad-statistic (2019).
16. Tango, T. *Playing the Percentages in Baseball: Why does 1.7*OPB + SLG Make Sense?* 2007. http://www.insidethebook.com/ee/index.php/site/comments/why%7B%5C_%7Ddoes%7B%5C_%7D17obpslg%7B%5C_%7Dmake%7B%5C_%7Dsense/ (2019).
17. Baseball Reference. *Batting Stats Glossary* https://www.baseball-reference.com/about/bat%7B%5C_%7Dglossary.shtml (2019).
18. Fangraphs. *Guts: wOBA and FIP Constants* 2019. https://www.fangraphs.com/guts.aspx?type=cn (2019).
19. Weinberg, N. *How to evaluate a hitter, sabermetrically* 2014. https://www.beyondtheboxscore.com/2014/5/26/5743956/sabermetrics-stats-offense-learn-sabermetrics (2019).

20. Cameron, D. *Win Values Explained: Part Three* 2008. https://blogs.fangraphs.com/explaining-win-values-part-three/ (2019).
21. Trupin, J. *An Idiot's Guide to Advanced Statistics: wOBA and wRC+* 2017. https://www.lookoutlanding.com/2017/3/7/14783982/an-idiots-guide-to-advanced-statistics-woba-and-wrc-sabermetrics (2019).
22. Baseball Prospectus. *Baseball Prospectus Glossary Details: PECOTA* https://legacy.baseballprospectus.com/glossary/index.php?search=PECOTA (2019).
23. Paine, N. *The Imperfect Pursuit of a Perfect Baseball Forecast* 2014. https://fivethirtyeight.com/features/the-imperfect-pursuit-of-a-perfect-baseball-forecast/ (2019).
24. Talsma, G. Data analysis and baseball. *The Mathematics Teacher* **92,** 738–742. https://search.proquest.com/docview/204617468/abstract/9ECBFDE1E9A849F4PQ/1?accountid=14507 (1999).
25. Yang, T. Y., Swartz, T. & East, A. L. *A Two-Stage Bayesian Model for Predicting Winners in Major League Baseball* 2004.
26. Silver, N. *Lies, Damned Lies: We are Elo?* 2006. https://www.baseballprospectus.com/news/article/5247/lies-damned-lies-we-are-elo/ (2019).
27. Paine, N. *The Best 2015 MLB Teams, According To Our New Ratings* 2015. https://fivethirtyeight.com/features/best-2015-mlb-teams-blue-jays-elo/ (2019).
28. Boice, J. *How Our MLB Predictions Work* 2019. https://fivethirtyeight.com/methodology/how-our-mlb-predictions-work/ (2019).
29. Boice, J. & Wezerek, G. *How Good Are FiveThirtyEight Forecasts?* 2019. https://projects.fivethirtyeight.com/checking-our-work/ (2019).
30. Tango, T. *True Talent Levels for Sports Leagues* 2006. http://www.insidethebook.com/ee/index.php/site/comments/true%7B%5C_%7Dtalent%7B%5C_%7Dlevels%7B%5C_%7Dfor%7B%5C_%7Dsports%7B%5C_%7Dleagues/ (2019).
31. Birnbaum, P. *Accurate Prediction and the Speed of Light* 2013. http://blog.philbirnbaum.com/2013/04/accurate-prediction-and-speed-of-light.html (2019).
32. Smith, D. *Discrepancies with Official Data* 2011. https://www.retrosheet.org/DiscrepanciesWithOffi pdf.
33. Retrosheet. *Game Logs* 2019. https://www.retrosheet.org/gamelogs/index.html (2019).
34. Retrosheet. *Fields* 2019. https://www.retrosheet.org/gamelogs/glfields.txt.
35. Lahman, S. *Lahman Baseball Database 2017 Readme* 2018. http://www.seanlahman.com/files/database/readme2017.txt.
36. Retrosheet. *Retrosheet History* https://www.retrosheet.org/history.htm (2019).
37. Fangraphs. *Fangraphs 1919-2019 Hitting Statistics* 2020. (2020).
38. Fangraphs. *Fangraphs 1919-2019 Pitching Statistics* 2020. https://www.fangraphs.com/leaders.aspx?pos=all%7B%5C&%7Dstats=pit%7B%5C&%7Dlg=all%7B%5C&%7Dqual=y%7B%5C&%7Dtype=8%7B%5C&%7Dseason=2019%7B%5C&%7Dmonth=0%7B%5C&%7Dseason1=2019%7B%5C&%7Dind=0%7B%5C&%7Dteam=%7B%5C&%7Drost=%7B%5C&%7Dage=%7B%5C&%7Dfilter=%7B%5C&%7Dplayers=%7B%5C&%7Dstartdate=%7B%5C&%7Denddate= ().
39. Nagpal, A. *L1 and L2 Regularization Methods* 2017. (2020).
40. Michael J. *Baseball Positions* 2015. https://commons.wikimedia.org/wiki/File:Baseball%7B%5C_%7Dpositions.svg.