ANALYSIS OF NFL PLAYER VALUATION

Isaiah Bryant, Edan Eingal, Kevin Li, Steven Liao

CSCI 1951A: Spring 2017 Final Project

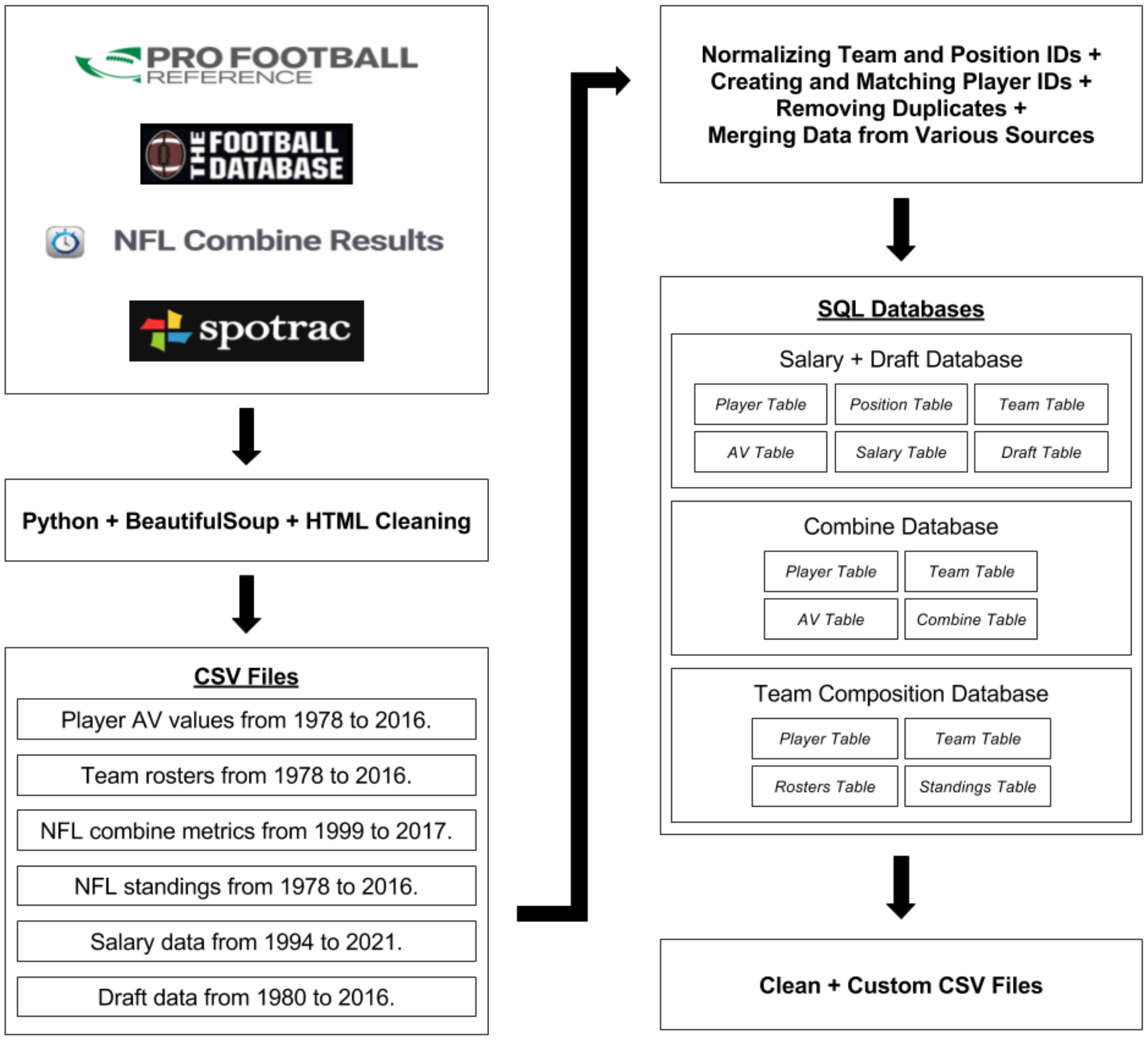
Abstract

In the National Football League (NFL), a team's general manager (GM) is tasked with building the best possible roster. The GM can acquire players through free agency (a 'free agent' is a player who is currently not contracted to any team), the draft (an annual process by which teams 'select' new players into the league, usually college players), or trades, also having to account for constraints such as roster size and salary cap (the maximum a team can spend in a year). Our goal with this project was to take on the role of a GM through data science lens. We leveraged Python's *BeautifulSoup* library to collect historical data for the NFL, including salary, draft, Approximate Value (AV), roster, draft combine, and standings data. We then divided our project into three parts: player and draft pick evaluation (Statistics), draft prospect k-nearest neighbor predictions (ML), and team composition analysis (ML). For player and draft pick evaluation, we used Pro Football Reference's AV metric to create a 'fair salary' metric for a player. We combined historical draft AV data with this metric to arrive a 'fair salary' for draft pick to value future draft picks. By comparing the 'fair salary' of draft picks to their actual predetermined salary, we were able to determine which draft picks offered the most purchasing power given salary cap restraints. Next, we implemented a k-nearest-neighbors test to determine which historical players this year's 2017 NFL Draft prospects matched most similarly to, based on NFL combine test results only. Finally, we explored optimal team composition (across positions), using position-wise summary statistics for team rosters as feature sets to predict the playoff performance of real and hypothetical teams. Our results for the player and draft pick evaluation confirmed that draft picks tend to serve as cheap talent, but offered the caveat that top draft picks may in fact be overvalued. Our draft prospect k-nearest-neighbor test was an exploratory aside, and offers value in terms of player comparison, as well as generally confirming that teams look at more than physical traits when evaluating players. Finally, our team composition analysis lent credence to the ideology that "defense wins championships," while also championing the idea that quarterbacks are crucial to team success. Limitations of our study include lack of data (the NFL only plays 16 games a year, and there is wide variance

in game outcomes), as well as the fact that there are practically infinite variables that affect a team and player's success that we couldn't entirely account for. Despite this, our results appear useful in controlled settings (player comparison on physical traits), and in some cases provide promising ground for continued exploration (draft pick evaluation, optimal team composition).

Data

Data collection and integration was a crucial aspect of our analysis, as we had to extract data from various sources, clean it, and load it into databases that would allow us to perform the analyses described throughout this report.


PRO FOOTBALL REFERENCE
THE FOOTBALL DATABASE
NFL Combine Results
spotrac
Python + BeautifulSoup + HTML Cleaning
CSV Files
Player AV values from 1978 to 2016.
Team rosters from 1978 to 2016.
NFL combine metrics from 1999 to 2017.
NFL standings from 1978 to 2016.
Salary data from 1994 to 2021.
Draft data from 1980 to 2016.
Normalizing Team and Position IDs + Creating and Matching Player IDs + Removing Duplicates + Merging Data from Various Sources
SQL Databases
Salary + Draft Database
Player Table
Position Table
Team Table
AV Table
Salary Table
Draft Table
Combine Database
Player Table
Team Table
AV Table
Combine Table
Team Composition Database
Player Table
Team Table
Rosters Table
Standings Table
Clean + Custom CSV Files

The data we extracted, the challenges we encountered during extraction, and the sources we extracted it from, are outlined below.

Appropriate Value:

A key statistic in all of our analyses is the Approximate Value (AV) of a player. AV is a metric developed by *Pro Football Reference* (pro-football-reference.com) that can be used as a proxy to describe the performance of a player at a given position over a period of one or more seasons. We obtained seasonal AV values for all players that played in the NFL from 1978 to 2016 from *Pro Football Reference.* We only obtained AV values from the 1978 season and beyond since that was the first season that the NFL had a 16 game season.

Roster:

For our team composition analysis, it was crucial to obtain roster data for every team for every season from 1978 to 2016. As with AV data, we only obtained rosters from the 1978 season and beyond since that was the first season that the NFL had a 16 game season. We obtained roster data from two different sources: *Pro Football Reference* and *The Football Database* (footballdb.com). The reason we had to obtain data from both of those sources is that the former contains information that allowed us to assign the same player ID to the same player in the roster data and the AV data (since they came from the same source) but doesn't have consistent position information, while the latter has consistent position information but does not allow us to assign player IDs accurately across the two datasets.

Draft combine:

For our draft combine analyses, it was crucial to obtain the combine metrics for every player that participated in the draft combine for as far back as we could find data. Unfortunately, the earliest year we could find combine data for was 1999, and thus our dataset only contains draft combine data from 1999 to 2017. We obtained draft combine data from two different sources: *Pro Football Reference* and *NFL Combine Results* (nflcombineresults.com). As with the roster data, the reason we had to do this is that the former contains information that would allow us to assign the same player ID to the same player in the combine data and the AV data but is missing several combine metrics, while the latter lists several more combine metrics but does not allow us to assign player IDs accurately and also is missing some players that appear in the former. Thus, merging these two datasets together required a process in which the *Pro Football Reference* combine data was looped over to assign player IDs to all of the players in the dataset, then the *NFL Combine Results* data was looped over and each player in it was assigned his player ID from the *Pro Football Reference* data and inserted into the database with all the metrics that were listed, and then the remaining players who were in the *Pro Football Reference* dataset but not in the *NFL Combine Results* dataset were inserted into the database and their missing combine metrics were listed as 0.

Standings:

For our team composition analysis, it was crucial to obtain standings data for every season from 1978 to 2016. As with AV and roster data, we only obtained standings from the 1978 season and beyond since that was the first season that the NFL had a 16 game season. The standings data we collected included teams' regular season records as well as whether they made the playoffs and if so, how far they advanced. The standings data came from *Pro Football Reference* and had to be minimally cleaned to ensure that the team IDs matched the IDs that we had been using for the rest of our data. Furthermore, playoff success was not included in the same table as the regular season standings and thus another section of the page had to be parsed to determine whether a team made the playoffs and how far it advanced.

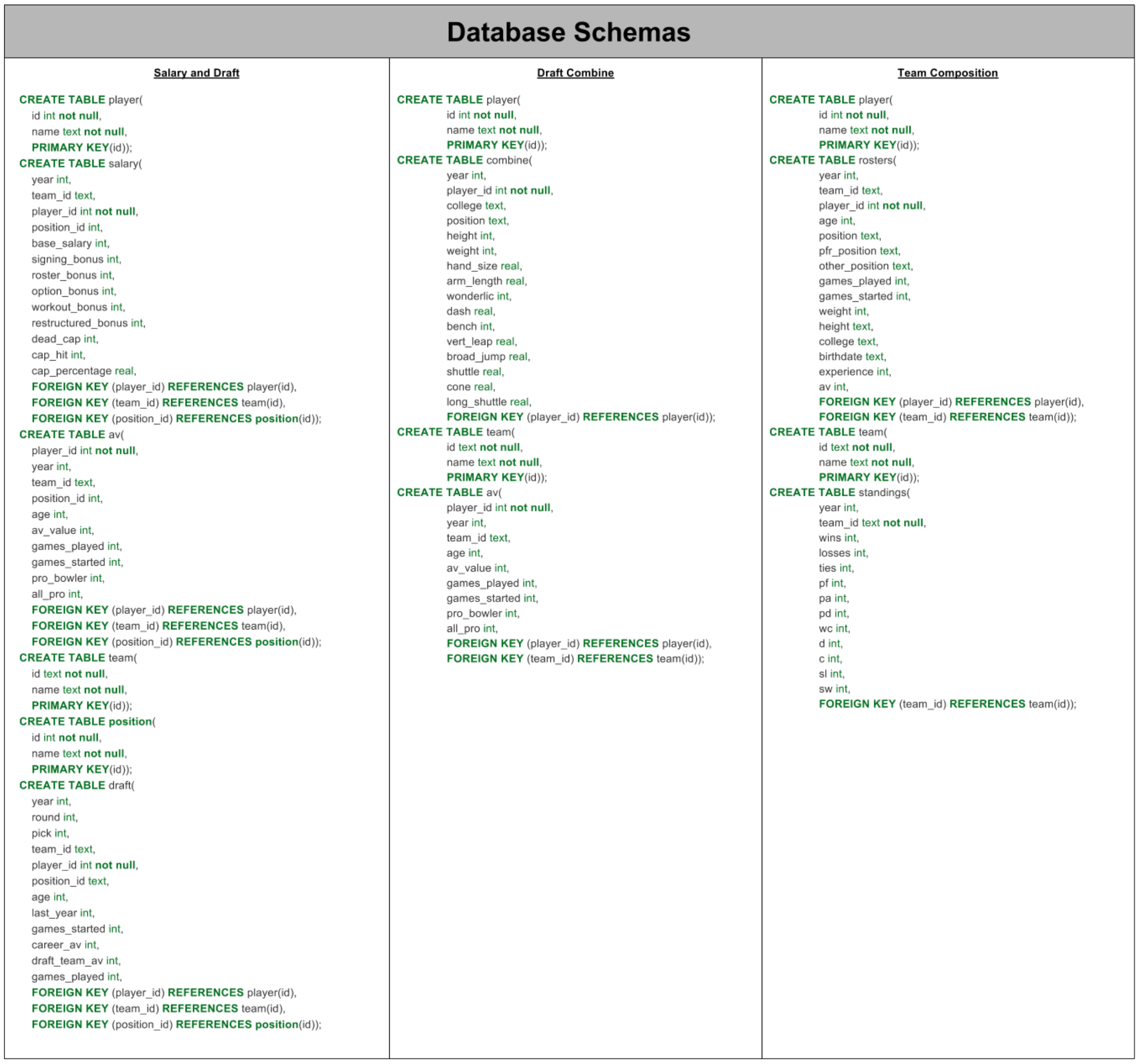
Salary:

For our salary analysis, it was crucial to obtain salary data for every season since 1994, since that was the year that the NFL salary cap was established. We had a very difficult time finding reliable salary data from before 2011, and although we were able to find some salary data as far back as 1994, any salary data that we were able to collect from before 2005 is incomplete. Still, we extracted what we could from *Spotrac* (spotrac.com), which had the most comprehensive dataset of past, present, and future NFL payrolls out of all of the websites we explored. While collecting this data, we also encountered challenges in terms of removing duplicate data - due to the fact that multiple teams have changed names since 1994 and thus the same data could be extracted twice if we didn't account for this - and converting nulls and hyphens on the website to 0s when extracting the data.

Draft:
For our draft analysis, it was crucial to obtain draft data for every season since 1980. We decided on obtaining draft data starting in 1980 since we were only interested in draft pick valuation from the start of the salary cap era, and figured that 1980 was about the earliest any player who was still playing in 1994 and making a meaningful impact would have been drafted.
We obtained the data from *Pro Football Reference*.

All of the data described above was obtained by scraping the aforementioned websites using *BeautifulSoup* scripts written in *Python* and applying HTML manipulations that allowed us to extract the correct data from each site even when it wasn't displayed in an ideal format for collection.

After scraping the websites and extracting the data, we wrote it, uncleaned, into CSV files. These CSV files were then used to create three different SQLite databases: the salary and draft database, the draft combine database, and the team composition database. We decided to separate the data into three databases since different cleaning and merging steps were required for each set of data and we felt that it might become too complicated to try and merge all of our different data sets together into one database. While transferring the data from the CSV files into the databases, we undertook various steps to clean it and ensure that all of the data in each database was consistent. Descriptions of the databases are below.



Database Schemas

Salary and Draft

```
CREATE TABLE player(
   id int not null,
   name text not null,
   PRIMARY KEY(id));
CREATE TABLE salary(
   year int,
   team_id text,
   player_id int not null,
   position_id int,
   base_salary int,
   signing_bonus int,
   roster_bonus int,
   option_bonus int,
   workout_bonus int,
   restructured_bonus int,
   dead_cap int,
   cap_hit int,
   cap_percentage real,
   FOREIGN KEY (player_id) REFERENCES player(id),
   FOREIGN KEY (team_id) REFERENCES team(id),
   FOREIGN KEY (position_id) REFERENCES position(id));
CREATE TABLE av(
   player_id int not null,
   year int,
   team_id text,
   position_id int,
   age int,
   av_value int,
   games_played int,
   games_started int,
   pro_bowler int,
   all_pro int,
   FOREIGN KEY (player_id) REFERENCES player(id),
   FOREIGN KEY (team_id) REFERENCES team(id),
   FOREIGN KEY (position_id) REFERENCES position(id));
CREATE TABLE team(
   id text not null,
   name text not null,
   PRIMARY KEY(id));
CREATE TABLE position(
   id int not null,
   name text not null,
   PRIMARY KEY(id));
CREATE TABLE draft(
   year int,
   round int,
   pick int,
   team_id text,
   player_id int not null,
   position_id text,
   age int,
   last_year int,
   games_started int,
   career_av int,
   draft_team_av int,
   games_played int,
   FOREIGN KEY (player_id) REFERENCES player(id),
   FOREIGN KEY (team_id) REFERENCES team(id),
   FOREIGN KEY (position_id) REFERENCES position(id));
```

Draft Combine

```
CREATE TABLE player(
            id int not null,
            name text not null,
            PRIMARY KEY(id));
CREATE TABLE combine(
            year int,
            player_id int not null,
            college text,
            position text,
            height int,
            weight int,
            hand_size real,
            arm_length real,
            wonderlic int,
            dash real,
            bench int,
            vert_leap real,
            broad_jump real,
            shuttle real,
            cone real,
            long_shuttle real,
            FOREIGN KEY (player_id) REFERENCES player(id));
CREATE TABLE team(
            id text not null,
            name text not null,
            PRIMARY KEY(id));
CREATE TABLE av(
            player_id int not null,
            year int,
            team_id text,
            age int,
            av_value int,
            games_played int,
            games_started int,
            pro_bowler int,
            all_pro int,
            FOREIGN KEY (player_id) REFERENCES player(id),
            FOREIGN KEY (team_id) REFERENCES team(id));
```

Team Composition

```
CREATE TABLE player(
            id int not null,
            name text not null,
            PRIMARY KEY(id));
CREATE TABLE rosters(
            year int,
            team_id text,
            player_id int not null,
            age int,
            position text,
            pfr_position text,
            other_position text,
            games_played int,
            games_started int,
            weight int,
            height text,
            college text,
            birthdate text,
            experience int,
            av int,
            FOREIGN KEY (player_id) REFERENCES player(id),
            FOREIGN KEY (team_id) REFERENCES team(id));
CREATE TABLE team(
            id text not null,
            name text not null,
            PRIMARY KEY(id));
CREATE TABLE standings(
            year int,
            team_id text not null,
            wins int,
            losses int,
            ties int,
            pf int,
            pa int,
            pd int,
            wc int,
            d int,
            c int,
            sl int,
            sw int,
            FOREIGN KEY (team_id) REFERENCES team(id));
```

Cleaning the data took a while due to the fact that it originated from various sources. An early issue we had was coming up with consistent ID's for players, positions, and teams that we could store in our databases. For example, team names, and thus their abbreviations, have changed multiple times between 1993 and 2017. In fact, the team that is currently known as the Los Angeles Rams, with an abbreviation of LAR on *Pro Football Reference*, was known as the Los Angeles Rams until 1994 with an abbreviation of RAM on *Pro Football Reference* and as the St. Louis Rams between 1994 and 2016 with an abbreviation of STL on *Pro Football Reference*. Despite these different names, they still represent the same franchise, and thus we had to ensure that older team abbreviations were changed to the abbreviation currently associated with that franchise. These current abbreviations were then used throughout our databases as team IDs.

We also had to figure out a feasible way to assign player IDs to players. This was especially difficult considering various players in the NFL between 1978 and now have shared the same name - and might have even played on the same team at the same time. Thus, for each database, we decided to look at different identifying features that would allow us to assign the right player ID to players from different datasets. For example, in the salary and draft database, we used a player's draft round and pick to differentiate between players with the same name, in the draft combine database, we used a combination of a player's name, birthdate, and weight to figure out which players from the two datasets were the same and thus should receive the same player ID, and in the team composition database we used a combination of a player's name, birthdate, team, and year to achieve this goal.

Assigning the correct player ID's to entries in the salary dataset proved more difficult than we initially anticipated. After running our database loading script and printing out the entries from the salary dataset that could not be matched up to entries in the AV dataset based on a tuple of (player_name, team, year), we obtained a long list of entries that could not be matched using this tuple. We investigated the list and realized that many of the issues occurred due to name variations - this could be anything from plain misspellings to the use of a nickname in one dataset and a full name in another to varying uses of punctuation and capitalizations in names between the two datasets. We decided to fix this by lower-casing all names and removing all punctuation from them when comparing names between the two datasets. For the entries that remained after doing this, we tried to relax the standards for comparing entries between the two tables - if our script couldn't match up entries based on the aforementioned tuple, then it would try to match up on a different tuple that could plausibly link an entry from the salary table to an entry from the

AV table. Finally, we tried to change the names of the remaining entries to see if that would result in a match; we thought of various names that could go by a nickname and their reciprocals and utilized these to find additional matches.

Standardizing position IDs within each of our databases was also challenge, due to the fact that there are various positions in football, most of which can be broken up into even more specific positions. For instance, the Offensive Line position, which is abbreviated as OL, can be broken down into Offensive Guard (OG) - which can be further broken down into Right Guard (RG) and Left Guard (LG), Offensive Tackle (OT) - which can be further broken down into Right Tackle (RT) and Left Tackle (LT), and Center (C). Thus, when extracting data from various sources, we had to decide on a consistent manner to assign positions to players in our databases. We settled on a position identification system that prioritized more specific position classifications but ensured that position IDs were not too specific as to not limit the amount of data we had for each position as well as to not muddle our results due to players having played multiple positions when position groupings become too specific. For instance, we preferred Defensive End (DE) and Defensive Tackle (DT) to Defensive Linemen (DL), but also preferred Defensive Tackle to Nose Tackle (NT) due to the specificity of the latter.

Once all of the cleaned data was stored in the aforementioned databases, it was very easy to utilize SQL queries in SQLite and output the result to a CSV file in order to create custom CSV files with clean data that we could then use for our visualizations and machine learning applications.

In short, aside from the salary data, there was enough information contained within the various datasets that we extracted to allow us to pick apart the data and generate interesting insights with it.

Methodology/Results

Our analyses were divided into three broad sections: player and draft pick valuation, draft prospect k-nearest-neighbor analysis, and team composition modeling. For purposes of clarity, we will specify the methodology and results in each respective section.

Challenges

Please see 'Data' section. We ran into issues finding feasible salary data, as well as merging roster data and combine data from various sources and removing duplicates. Assigning the correct player IDs to players from different datasets, deciding on a standard set of position and team IDs.

Salary Analysis

Given the NFL salary cap (currently set for 2017 at 167 million dollars), it was crucial for us to consider salary in our player evaluation analyses. In other words, the value a player brings is not just in how ‘good’ they are, but also in how ‘cheap’ they are.

In theory, every general manager strives to maximize their return on the salary cap. They can attempt to sign players for cheap (though they are competing against a free market, in theory); however, a more common and - and dependable - method is to build a team using draft picks. Due to a fixed rookie wage scale, drafted players are believed to be paid significantly less than they would be worth on a free market.

Under this framework, we sought to develop a system to fairly value salary, players, and draft picks in conjunction to one another. How much should a player be paid? What is fair compensation for this player (and their contract) in a trade? How much value does a draft pick actually offer, if any?

To start, we needed to develop a metric for the ‘fair salary’ for a player - this is defined as the ‘maximum’ salary that a team should be willing to pay for a player in a non edge-case situation (i.e. for instance, one edge case is when a team is far below the minimum spending threshold, and then ‘overspending’ is preferable to having unspent money go wasted). In other words, by paying players below their fair salary, we are increasing the return on our salary cap!

The premise behind our fair salary metric is that the salary ratio between Player A and Player B should be equal to their talent / contribution ratio. We use Approximate Value (AV) as a proxy for talent and contribution level; Approximate Value is a metric developed by Pro Football Reference designed to measure contribution, and is comparable across positions. It is measured on a yearly basis and is useful as an objective, comparable metric for how much a player contributed to their team in a given year.

In making our fair salary metric, we made the following assumptions:

- AV contribution is linear. In other words, a player whose AV is 8 in a given year is worth twice as much as a player whose AV is 4 in a given year.
- A player with 0 AV is worth $0
- A practice squad player’s pay is negligible

Using these three assumptions, and inputting the salary cap for every year from 1994 to 2016 (ranging from 34.6 million in 1994 to 155.27 million in 2016), we were able to calculate the fair salary for each player in that given year, using the following formula:

- Let SC = total combined salary cap across all 32 teams (167 million times 32 for 2017)
- Let N = total number of players who can be on an NFL roster (53 times 32 for 2017)
- Let $AVTotal$ = Summed AV of the top N NFL players in a given season (averaged per season from 1994 -2016, probably)

Then Fair Salary is:

$$p_a v / AVTotal * SC$$

We began by using the following SQL code to compose a CSV file to be used in R.

```
sqlite3 -header -csv nfl_data.db
"SELECT m.year, player.id, player.name, m.team_id,
m.position_id, m.age, m.av_value, m.year, m.round,
m.pick, m.base_salary, m.cap_hit, m.dead_cap
FROM player,
((av LEFT OUTER JOIN salary
ON av.player_id == salary.player_id AND av.year == salary.year) k
LEFT OUTER JOIN draft ON k.player_id == draft.player_id) m
WHERE av.player_id == player.id
ORDER BY m.year ASC, m.av_value DESC" > ..\data\fair_salary.csv
```

Here is a view at the data:

```
##    year      id            name team_id position_id age av_value
## 1  1994    2403     Steve Young     SFO          QB  33       23
## 2  1994  198516      Jerry Rice     SFO          WR  32       21
## 3  1994   19893   Barry Sanders     DET          RB  26       20
## 4  1994  199145   Ricky Watters     SFO          RB  25       19
## 5  1994  199159 Aeneas Williams     ARI          CB  26       18
## 6  1994   19912     Eric Turner     CLE           S  26       18
## 7  1994  198710     Rod Woodson     PIT           S  29       18
## 8  1994 1987150      Greg Lloyd     PIT         OLB  29       18
## 9  1994 1985113    Kevin Greene     PIT         OLB  32       18
## 10 1994  199017    Emmitt Smith     DAL          RB  25       17
##    draft_year round pick    base cap_hit dead_cap
## 1        1994    NA   NA      NA      NA       NA
## 2        1994     1   16 2000000 2380000        0
## 3        1994     1    3 2575000 3080000        0
## 4        1994     2   45      NA      NA       NA
## 5        1994     3   59  750000 1200000        0
## 6        1994     1    2 1000000 1812500        0
## 7        1994     1   10      NA      NA       NA
## 8        1994     6  150      NA      NA       NA
## 9        1994     5  113      NA      NA       NA
## 10       1994     1   17 2200000 3200000        0
```

After manually adding roster size, salary cap size (34.6 million in 1994 to 167 million this year), and number of teams in the league, we were able to use the aforementioned formula to calculate Approximate Value. We chose to use 'Cap Hit' as the salary for a player; this is not how much the player is actually paid, but how much the player counts against the salary cap. These values may be different depending on contract structure. We considered 'cap hit' a better representation for our purposes because this is how much money a team would save if they cut the player.

Note that in every year in our analyses, AVTotal was simply the summed AV of all players, since the number of players with nonzero AV was always less than the number of rosterable players (N).

Some of our findings included:

- Matt Ryan, the player with the highest AV in 2016, had a fair salary of $16.13 million
- The most overpaid player between 1994 and 2016 was Tony Romo in 2016, whose fair salary was 0, but whose cap hit was over 20 million

- The most underpaid player between 1994 and 2016 was Dak Prescott, whose fair salary was 12.3 million, and whose cap hit was 546,000.

Draft Analyses

The NFL draft is considered the key pipeline for cheap talent because the contract each draft pick receives is fixed in both length and salary (length is four years, salary depends on how high the selection was), due to stipulations in the NFL's Collective Bargaining Agreement.

This is not the case for NFL free agency, where players are free to sign with any team in what resembles a free market.

If we can calculate the fair salary for each draft pick, based on historical AV data for each draft pick, we can then determine how underpaid each draft pick is (which is good for maximizing return on salary cap), if at all.

We begin by creating CSV files for our analysis in R using SQL:

```
sqlite3 -header -csv nfl_data.db 'SELECT player.id, player.name, drat.year, dra
ft.round, draft.pick, av.year, av.av_value FROM av JOIN (player LEFT JOIN draf
t ON player.id == draft.player_id) p ON player.id == av.player-id ORDER BY play
er.id, av.year;' > ../data/draft_regression.csv
```

Setup

Importing the data:

```
setwd("C:/Users/Steven/OneDrive/Documents/Brown/Junior Year/Second Semester/CSC
I1951a/cs1951a_project/data")
dr <- read.csv("draft_regression.csv")
proj_salary <- read.csv("rookie_contracts.csv")
#dr stands for draft regression
```

Data Cleaning

We clean the data, removing undrafted players, players drafted before 1994, and we fix the order of the dataset to have it sorted by draft year, draft pick, and then av_year, in that order.

Below is the first ten rows of our cleaned table:

```
##          id          name draft_year round pick year av
## 16829 19941  Dan Wilkinson       1994     1    1 1994  6
## 16830 19941  Dan Wilkinson       1994     1    1 1995  6
## 16831 19941  Dan Wilkinson       1994     1    1 1996  7
## 16832 19941  Dan Wilkinson       1994     1    1 1997  5
## 16833 19941  Dan Wilkinson       1994     1    1 1998  7
## 17324 19942 Marshall Faulk       1994     1    2 1994 16
## 17325 19942 Marshall Faulk       1994     1    2 1995 13
## 17326 19942 Marshall Faulk       1994     1    2 1996  9
## 17327 19942 Marshall Faulk       1994     1    2 1997 12
## 17328 19942 Marshall Faulk       1994     1    2 1998 18
```

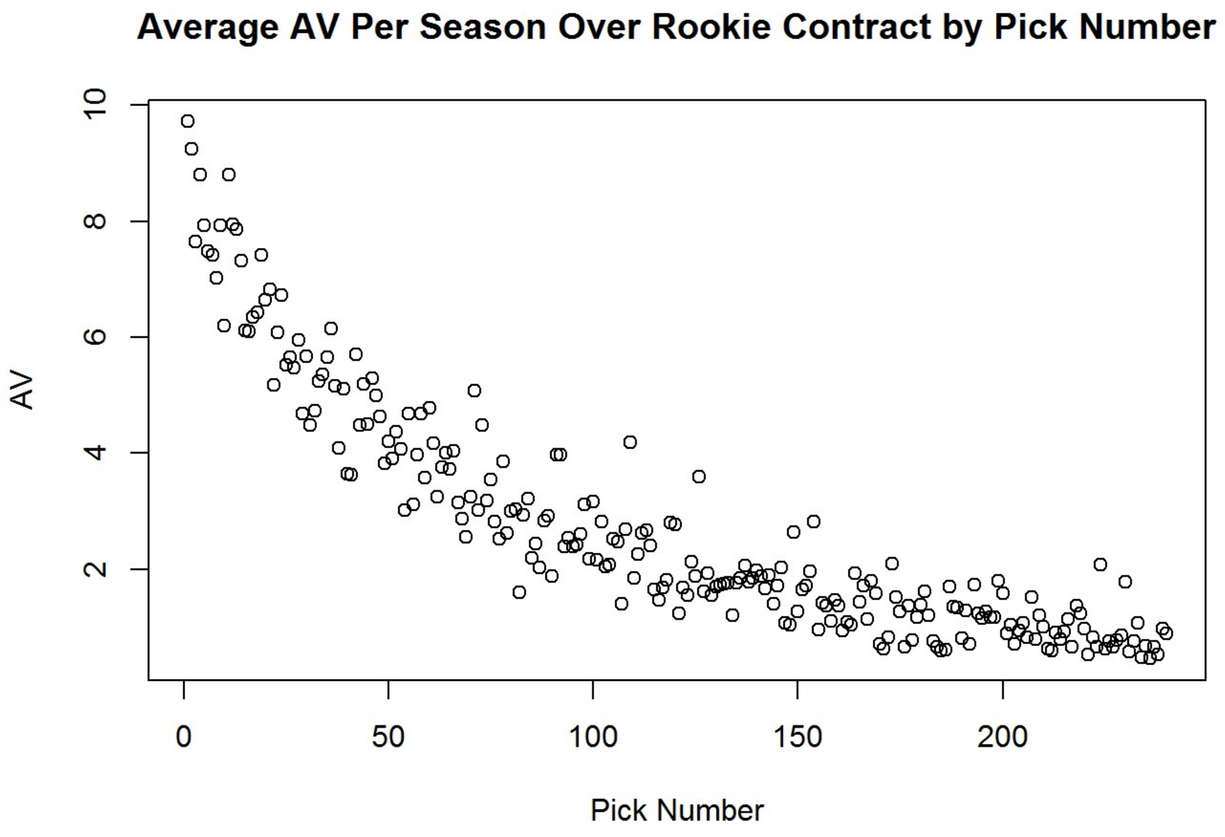
Aggregating Data into Picks Table

Our data contains draft and AV data for players from 1994 to 2016. We want our final product to be a table of all 240 picks, with their actual salary, fair salary, and ‘value above fair salary.’ 240 is the minimum number of picks in a draft between 1994 and 2016; this allows us to assume there are 23 (or are supposed to be 23) players for every pick, which makes calculations easier. Here we began constructing our Picks Table:

```
##    pick total_av average_av   log_av
## 1     1      894   9.717391 2.273917
## 2     2      850   9.239130 2.223448
## 3     3      704   7.652174 2.034990
## 4     4      809   8.793478 2.174010
## 5     5      730   7.934783 2.071256
## 6     6      688   7.478261 2.012000
## 7     7      682   7.413043 2.003241
## 8     8      646   7.021739 1.949011
## 9     9      729   7.923913 2.069885
## 10   10      570   6.195652 1.823848
## 11   11      809   8.793478 2.174010
## 12   12      731   7.945652 2.072625
## 13   13      724   7.869565 2.063003
## 14   14      673   7.315217 1.989957
## 15   15      563   6.119565 1.811491
## 16   16      561   6.097826 1.807932
## 17   17      584   6.347826 1.848112
## 18   18      592   6.434783 1.861718
## 19   19      683   7.423913 2.004706
## 20   20      611   6.641304 1.893308
## 21   21      628   6.826087 1.920752
## 22   22      476   5.173913 1.643629
## 23   23      560   6.086957 1.806148
## 24   24      619   6.728261 1.906317
## 25   25      508   5.521739 1.708693
## 26   26      520   5.652174 1.732040
## 27   27      503   5.467391 1.698802
## 28   28      547   5.945652 1.782660
## 29   29      430   4.673913 1.541997
## 30   30      522   5.673913 1.735879
## 31   31      412   4.478261 1.499235
## 32   32      435   4.728261 1.553557
## 33   33      482   5.239130 1.656156
## 34   34      492   5.347826 1.676690
## 35   35      520   5.652174 1.732040
## 36   36      566   6.152174 1.816806
## 37   37      475   5.163043 1.641526
## 38   38      376   4.086957 1.407801
## 39   39      470   5.108696 1.630944
## 40   40      335   3.641304 1.292342
## 41   41      334   3.630435 1.289352
## 42   42      525   5.706522 1.741610
## 43   43      413   4.489130 1.501659
## 44   44      477   5.184783 1.645728
## 45   45      414   4.500000 1.504077
## 46   46      487   5.293478 1.666476
## 47   47      459   4.989130 1.607262
## 48   48       426   4.630435 1.532651
## 49   49       352   3.826087 1.341843
## 50   50       386   4.195652 1.434049
```

Regressing on our Data

First, let’s take a look at our data:

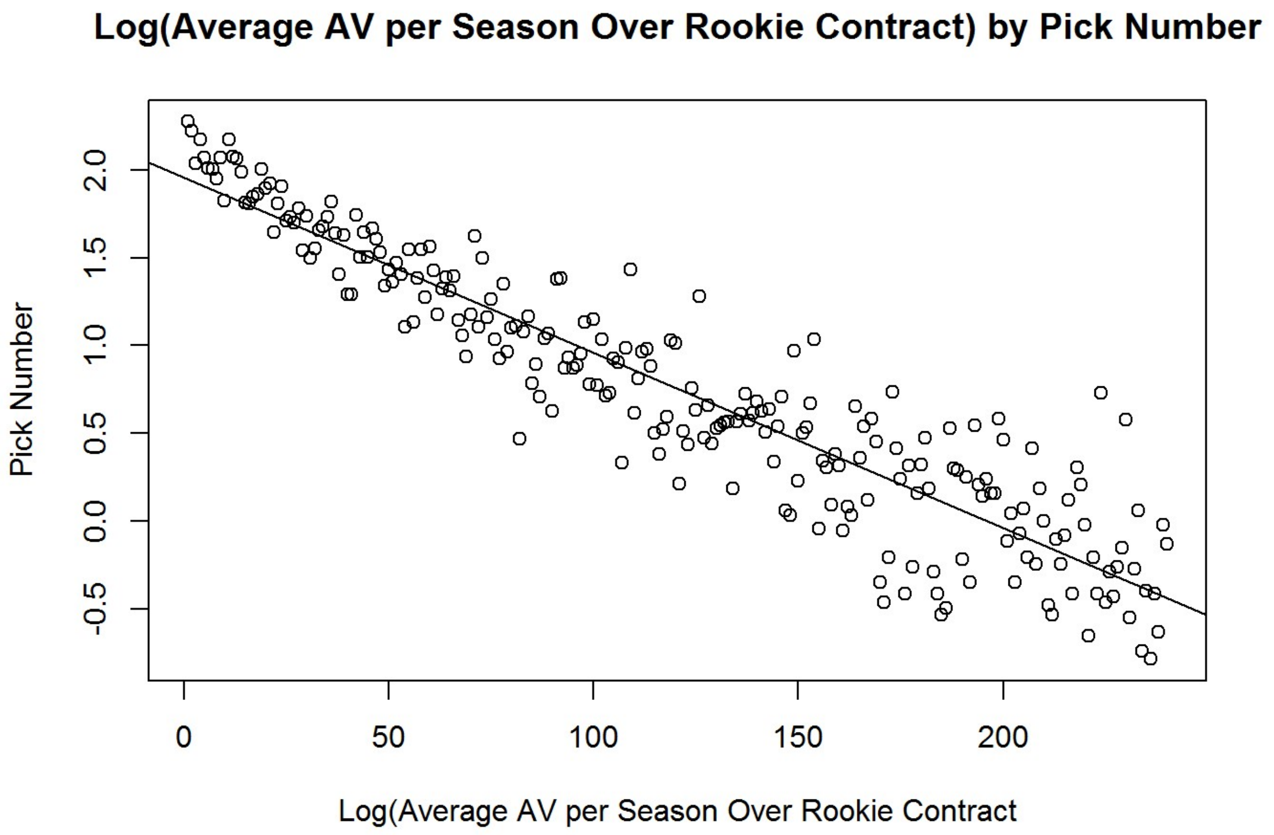


We regress our data using a logarithmic regression to best fit the data. We get the following regression formula:

$\text{av} = 7.07347e^{-0.0099652 \times \text{pick}}$

We have an R^2 of 0.8676.

```
##
## Call:
## lm(formula = log_av ~ pick, data = av_counts)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.71364 -0.16964  0.01007  0.16504  1.00634
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.9563493  0.0350705   55.78   <2e-16 ***
## pick        -0.0099652  0.0002523  -39.50   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2708 on 238 degrees of freedom
## Multiple R-squared:  0.8676, Adjusted R-squared:  0.8671
## F-statistic:  1560 on 1 and 238 DF,  p-value: < 2.2e-16
```



Fair Salary

Here we calculate the expected AV according to our regression. Using this expected AV, we then calculate the fair salary, assuming that:

1. The salary cap is that of 2017: $167,000,000
2. The salary cap stays the same over the next four years
3. The total AV in all of the next four years is approximated by the average total AV over the last five years (2012-2016), which comes out to 6559.

Assumptions #2 and #3 will almost certainly not be true but we are fairly confident any differences will be marginal; the exact differences are also difficult to predict, so we are fine making these simplifying assumptions.

Below are the fair salaries per year of the rookie contract for the top 25 picks in the draft.

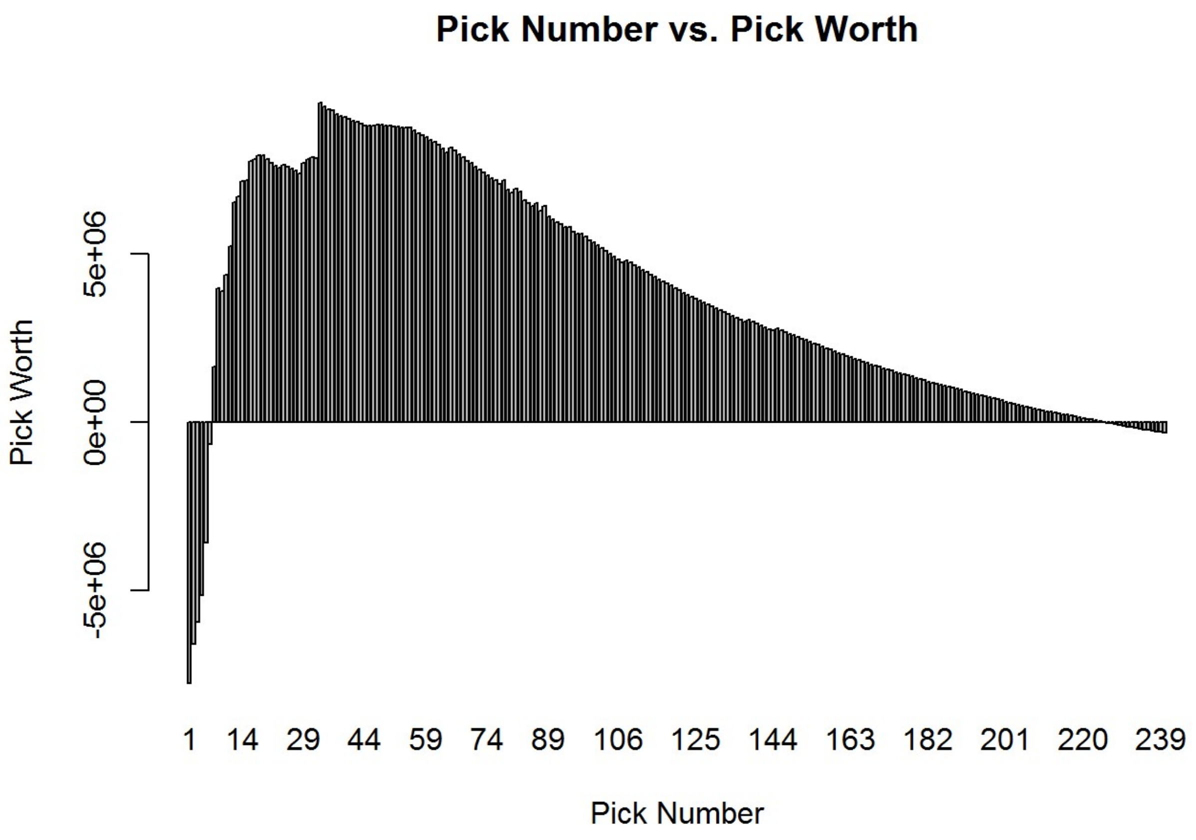
```
av_counts$expected_av = 7.073457*exp(1)^{-0.0099652*av_counts$pick}

pick_chart = av_counts[, c("pick", "expected_av")]
salary_cap = 167000000
total_av = 6559.4
num_teams = 32
pick_chart$fair_salary = pick_chart$expected_av/total_av*salary_cap*num_teams

head(pick_chart, 25)
```

```
##    pick expected_av fair_salary
## 1     1    7.003319     5705664
## 2     2    6.933876     5649089
## 3     3    6.865121     5593074
## 4     4    6.797049     5537615
## 5     5    6.729651     5482705
## 6     6    6.662922     5428340
## 7     7    6.596854     5374514
## 8     8    6.531442     5321222
## 9     9    6.466678     5268459
## 10   10    6.402556     5216218
## 11   11    6.339070     5164496
## 12   12    6.276214     5113286
## 13   13    6.213981     5062584
## 14   14    6.152365     5012385
## 15   15    6.091360     4962684
## 16   16    6.030960     4913475
## 17   17    5.971158     4864754
## 18   18    5.911950     4816517
## 19   19    5.853329     4768758
## 20   20    5.795289     4721472
## 21   21    5.737824     4674655
## 22   22    5.680930     4628303
## 23   23    5.624599     4582410
## 24   24    5.568827     4536972
## 25   25    5.513609     4491985
```

Draft Pick Worth

Finally, we have the fair salary for each draft pick per year! We also know that NFL draft picks have relatively fixed salaries. Now, we can calculate the difference between the fair salary of a draft pick, and how much the draft pick is actually paid, to determine the draft pick's theoretical worth ('return above fair salary'). Estimated salaries for 2017 NFL draft picks are scraped from Spotrac.

```
##    pick expected_av fair_salary estimate total_fair_salary pick_worth
## 1     1    7.003319     5705664 30566250          22822657 -7743592.6
## 2     2    6.933876     5649089 29178790          22596355 -6582435.5
## 3     3    6.865121     5593074 28295878          22372296 -5923582.4
## 4     4    6.797049     5537615 27286806          22150458 -5136347.6
## 5     5    6.729651     5482705 25520944          21930821 -3590123.2
## 6     6    6.662922     5428340 22367629          21713361  -654267.9
## 7     7    6.596854     5374514 19844976          21498058  1653081.7
## 8     8    6.531442     5321222 17322323          21284889  3962566.1
## 9     9    6.466678     5268459 17196047          21073834  3877787.3
## 10   10    6.402556     5216218 16502457          20864872  4362415.2
```



Discussion and Results

Our results confirm the idea that draft picks do tend to be underpaid, and thus are great for maximizing returns on the salary cap. In addition, we find that high first round picks tend to be overpaid, and that the most underpaid pick is actually pick number 33, the top pick of the second round.

These results imply myriad insights (some not necessarily obvious), including:

- The 33rd overall pick is worth more than the 1st overall pick. Yes, you get a worse talent, but you get more bang for the buck. As a General Manager, this also means that trading down in the draft is generally a great strategy.
- In trading for Brock Osweiler this past spring, the Browns essentially traded 18 million dollars for a second round pick. A mid-second round pick is expected to only provide 8 million dollars in contributions above fair value, implying that this trade was a poor one for the Browns if they valued Brock Osweiler at 10 million dollars or less. In other words, they would have been better off spending that money in free agency.

These are impactful and meaningful insights, and provide substantial promise for future work. However, we note the following limitations (which we could also address in future work):

- The fifth year option for first round picks is not factored into the valuation. For first round picks, teams get a ‘option’ to add an additional fifth year to the player’s contract, usually at a more expensive salary than the previous four years. This past year, 23 of the 32 first round pick options were picked up. Thus, we can expect first round picks to be worth slightly more than what our model suggests. Quantifying this would be a worthy next step.
- Another flaw in our model is that we did not account for possible increases in salary cap. The salary cap has increased substantially over the past 23 years, and it is reasonable to expect it to continue to increase. Accounting for this would increase the ‘fair salary’ for the draft picks, and further increase the ‘worth’ of each draft pick. This means that our current estimates for pick worth are almost certainly conservative.
- Finally, it is important to note that Approximate Value is just as its name indicates, an ‘approximate’ value of a player’s contribution. Creating a better metric could be a sizeable undertaking but would potentially improve results.

A final caveat to note is that these draft pick valuations are only valid for future, unknown draft picks. During the actual draft, these picks should be instead valued based the players available at each selection (which, of course, is not known until usually a few picks before).

Draft Combine Analysis

Introduction

As outlined above, we constructed a metric that allows us to value both current players and draft picks. However, even if we know the predicted value of a draft pick, we still need to be able to deduce which prospects are most likely to succeed in the NFL so that we can fully maximize the value we get with said draft pick. The easiest way to compare draft prospects to current and former players, and in turn predict their potential success in the NFL, is using their draft combine metrics; after all, football is a game that relies on size, strength, and athleticism, and combine metrics are the most consistent indicator we have of the aforementioned characteristics of each player. Thus, we decided to develop an engine that would utilize unsupervised learning techniques to compare the combine metrics of current and former players to those of current draft prospects in order to determine the predicted NFL success of the prospects. Obviously this is not a fail-proof plan, as various factors go into determining the NFL potential of a prospect that we didn't account for, such as college statistics, personality, and injury history. Furthermore, we were only able to find combine data from 1999 to 2017, and thus are only able to compare draft prospect to players who participated in the combine in 1999 or later. Still, the predictions, and how we went about obtaining them, are both interesting and essential to our analysis, as they add a new dimension that allows us to not only value draft picks, but also to value the players that will be drafted with those picks.

Querying the Database

In order to acquire the training data and testing data in well-formatted CSV files that could be used to determine the NFL potential of draft prospects, we had to query our draft combine database and output the results into CSV files. The training data consists of all players that have draft combine metrics from 1999 to 2016 along with their average seasonal AV, which we computed by dividing a player's career AV by the number of years he played. This ensures that players who were recently drafted and thus don't have high career AVs but have still been productive during their time in the NFL are not undervalued in our evaluation. The test data can be anything as long as it follows the format defined by the query below. This means that you our engine can be used to find the most similar players to and predict the average seasonal AV of a single player as well as an entire draft class. All one has to do is define the training data by feeding a CSV into the engine that contains the player/s you want to analyze. The queries used to acquire each set of data are listed below.

Training data:

```
sqlite3 -header -csv combine_data.db
"select name, sum_av*1.0/count_av, combine.*
from player, combine,
    (select player_id, sum(av_value) as sum_av, count(player_id) as count_av
    from av group by player_id) as av_sum
where combine.player_id = player.id and combine.player_id = av_sum.player_id;"
> ../data/similar_players_data.csv
```

Testing data:

```
sqlite3 -header -csv combine_data.db
"select name, combine.*
from player, combine
where year = 2017 and player.id = combine.player_id;"
> ../data/combine_players_2017.csv
```

Draft Combine Nearest Neighbors

This program utilizes sklearn's k nearest neighbors algorithm in order to find the nearest neighbors to the test data that is fed into it. As described above, its training data is the draft combine metrics of all players that participated in the NFL draft combine between 1999 and 2016. It then uses these metrics to determine which of the players in the training dataset is most similar to the players being passed in as the test data by finding the total distance between the combine metrics of each player in the test data and the combine metrics of the players in the training data and returning the *k* players in the training data that are the least total distance from each player in the test data. The engine takes five flags as arguments: *training*, *test*, *k*, *pos*, and *zero*. The first two specify the csv files that contain the training data and the test data. These are optional and default to similar_players_data.csv and test_players.csv respectively. The former should almost never have to be changed, while the latter can be altered depending on which players you want to find the nearest neighbors to. The *k* flag specifies how many nearest neighbors the engine should return for each player in the test data. It defaults to 5. The *pos* flag specifies whether the engine should only return players with the same (or similar) position to that of the test player being evaluated. It can be set to True or False, and defaults to False. The *zero* flag specifies whether the engine should ignore combine metrics that are equal to 0 when finding a player's nearest neighbors, as such a value is often indicative of missing data. It can be set to True or False, and defaults to False.

Results

We ran the nearest neighbors search on the players drafted in the first round of the 2017 NFL draft to determine the former and current NFL players that are most similar to them based on draft combine metrics, and in turn predict their expected future success in the NFL. We ran it with the *pos* flag set to True in order to limit the results to players that played the same position as the player being tested. The results were interesting; in fact, the order in which the players came off the board in the first round does not align with the expected future success (in terms of average seasonal AV) predicted by our similarity. This can be explained by the fact that various other factors that we didn't account for in our model - factors such as college statistics, personality, and injury history - go into deciding the NFL potential of a prospect. Furthermore, teams tend to pick based on needs and not necessarily based on the the best players available, meaning that it is understandable that the actual draft order might not align with our predictions. Finally, it is important to note that changing the number of neighbors changes the results as well, meaning that we could obtain different predictions simply by changing the *k* value or even by setting the *zero* flag to True. However, setting *k* to 3 and *zero* to False seemed to return results that most closely aligned with the general consensus regarding the potential future success of each draft pick (with some notable exceptions such as Mitchell Trubisky and Mike Williams). The results, including the three most similar former or current players to each player drafted in the first round as well as each pick's predicted average seasonal AV, are displayed below.

Similar Players to 2017 First Round Draft Picks

Picks 1-11:

Similar players to (Myles Garrett, DE):
(Justin Houston, 8.833333333, OLB)
(Owamagbe Odighizuwa, 0.5, DE)
(Trey Flowers, 3.5, DE)

Predicted average seasonal AV: 4.28

Similar players to (Mitchell Trubisky, QB):
(Brett Basanez, 0, QB)
(Brock Berlin, 0, QB)
(Kevin Kolb, 2.166666667, QB)

Predicted average seasonal AV: 0.72

Similar players to (Solomon Thomas, DE):
(Justin Houston, 8.833333333, OLB)
(Preston Smith, 5, DE)
(Owamagbe Odighizuwa, 0.5, DE)

Predicted average seasonal AV: 4.78

Similar players to (Leonard Fournette, RB):
(Lamar Miller, 6.4, RB)
(Trent Richardson, 6, RB)
(Kevan Barlow, 6.5, RB)

Predicted average seasonal AV: 6.3

Similar players to (Corey Davis, WR):
(Breshad Perriman, 3, WR)
(Alshon Jeffery, 7.6, WR)
(Justin Blackmon, 4.5, WR)

Predicted average seasonal AV: 5.03

Similar players to (Jamal Adams, S):
(Cyhl Quarles, 0, SS)
(Harrison Smith, 7, SS)
(Bradley McDougald, 0, FS)

Predicted average seasonal AV: 2.33

Similar players to (Mike Williams, WR):
(Arrelious Benn, 2, WR)
(David Gettis, 1.5, WR)
(Kerry Meier, 0, WR)

Predicted average seasonal AV: 1.17

Similar players to (Christian McCaffrey, RB):
(Wendell Smallwood, 3, RB)
(Mike Gillislee, 2, RB)
(Kenjon Barner, 0.666666667, RB)

Predicted average seasonal AV: 1.89

Similar players to (John Ross, WR):
(Marqise Lee, 4, WR)
(Demarcus Ayers, 0, WR)
(Joe Adams, 0, WR)

Predicted average seasonal AV: 1.33

Similar players to (Patrick Mahomes, QB):
(Dak Prescott, 16, QB)
(Pat Devlin, 0, QB)
(Chandler Harnish, 0, QB)

Predicted average seasonal AV: 5.33

Similar players to (Marshon Lattimore, CB):
(A.J. Jefferson, 2, CB)
(Janoris Jenkins, 7.8, CB)
(Chimdi Chekwa, 0.4, CB)

Predicted average seasonal AV: 3.4

Picks 12-22:

Similar players to (Deshaun Watson, QB):
(Marcus Mariota, 11, QB)
(Pat Devlin, 0, QB)
(Dak Prescott, 16, QB)

Predicted average seasonal AV: 9.0

Similar players to (Haason Reddick, DE):
(Ryan Shazier, 6.333333333, OLB)
(Darron Lee, 4, OLB)
(Kevin Pierre-Louis, 1, OLB)

Predicted average seasonal AV: 3.78

Similar players to (Derek Barnett, DE):
(Cameron Sheffield, 0.5, OLB)
(Armonty Bryant, 1, DE)
(Shaun Phillips, 5.909090909, DE)

Predicted average seasonal AV: 2.47

Similar players to (Malik Hooker, S):
(Karl Joseph, 4, SS)
(Jordan Lucas, 0, SS)
(Jeremy Cash, 0, SS)

Predicted average seasonal AV: 1.33

Similar players to (Marlon Humphrey, CB):
(Javier Arenas, 1.6, CB)
(Josh Norman, 5.8, CB)
(Micah Hyde, 4.25, CB)

Predicted average seasonal AV: 3.88

Similar players to (Jonathan Allen, DE):
(Quinton Coples, 5, DE)
(Malik Jackson, 5, DE)
(William Gholston, 4.5, DE)

Predicted average seasonal AV: 4.83

Similar players to (Adoree Jackson, CB):
(Dre Kirkpatrick, 3.8, CB)
(Damian Swann, 1, CB)
(Chimdi Chekwa, 0.4, CB)

Predicted average seasonal AV: 1.73

Similar players to (O.J. Howard, TE):
(James Hanna, 0.75, TE)
(Austin Hooper, 3, TE)
(Jordan Cameron, 2.333333333, TE)

Predicted average seasonal AV: 2.03

Similar players to (Garett Bolles, OT):
(Tanner Hawkinson, 0, OT)
(Willie Smith, 1.5, OT)
(Andrus Peat, 8, OT)

Predicted average seasonal AV: 3.17

Similar players to (Jarrad Davis, LB):
(Antonio Morrison, 3, ILB)
(Jeremiah Attaochu, 2.666666667, OLB)
(Casey Matthews, 2.5, ILB)

Predicted average seasonal AV: 2.72

Similar players to (Charles Harris, DE):
(Jacquies Smith, 2.5, DE)
(Demarcus Lawrence, 3.333333333, DE)
(Michael Buchanan, 0, DE)

Predicted average seasonal AV: 1.94

Picks 23-32:

Similar players to (Evan Engram, TE):
(Chris Gragg, 0.333333333, TE)
(Thomas Duarte, 0, TE)
(DeAngelo Peterson, 0, TE)

Predicted average seasonal AV: 0.11

Similar players to (Gareon Conley, CB):
(Kyle Fuller, 5.5, CB)
(Demetri Goodson, 0.666666667, CB)
(Curtis Marsh, 0.333333333, CB)

Predicted average seasonal AV: 2.17

Similar players to (Jabrill Peppers, LB):
(Kwon Alexander, 7, OLB)
(Sean Porter, 0, OLB)
(Darron Lee, 4, OLB)

Predicted average seasonal AV: 3.67

Similar players to (Takkarist McKinley, DE):
(Kyle Emanuel, 3.5, OLB)
(Edmond Robinson, 1, OLB)
(Von Miller, 13.33333333, OLB)

Predicted average seasonal AV: 5.94

Similar players to (TreDavious White, CB):
(Aaron Ross, 3.285714286, CB)
(Leodis McKelvin, 3.888888889, CB)
(Roc Carmichael, 0.5, CB)

Predicted average seasonal AV: 2.56

Similar players to (Taco Charlton, DE):
(Tyrone Crawford, 5.25, DE)
(Brandon Bair, 0.666666667, DE)
(Carl Nassib, 2, DE)

Predicted average seasonal AV: 2.64

Similar players to (David Njoku, TE):
(Chris Gragg, 0.333333333, TE)
(Matt Jones, 5.75, TE)
(Virgil Green, 0.833333333, TE)

Predicted average seasonal AV: 2.31

Similar players to (T.J. Watt, LB):
(Khalil Mack, 12.33333333, OLB)
(Martez Wilson, 1, ILB)
(Dontay Moch, 0, OLB)

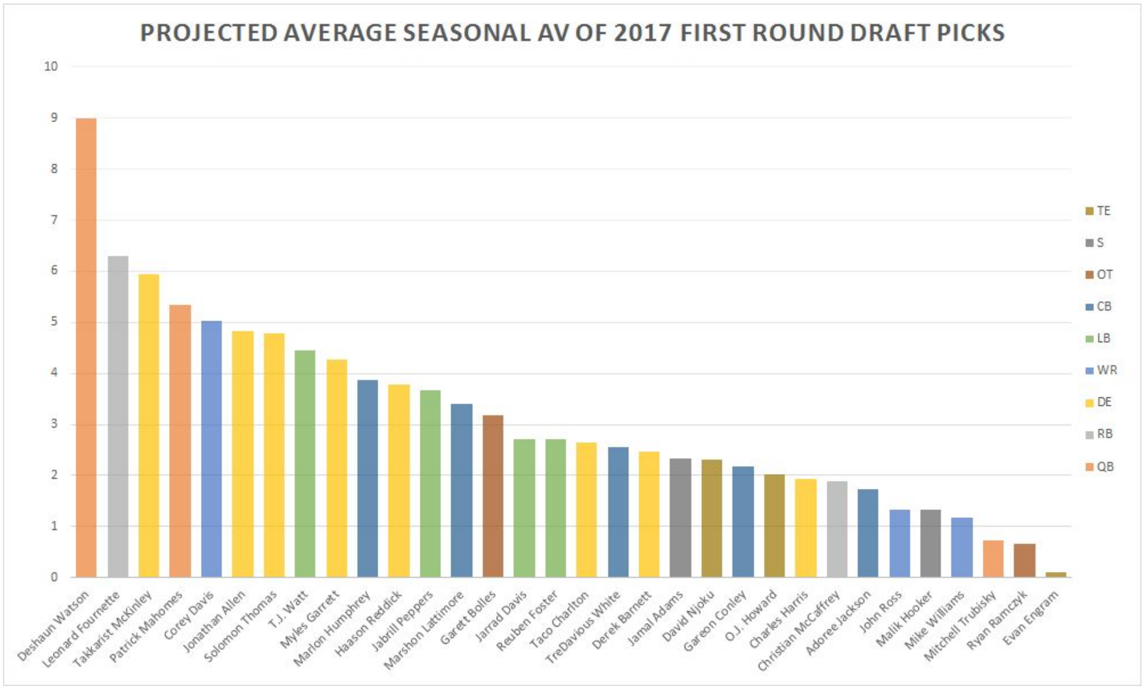
Predicted average seasonal AV: 4.44

Similar players to (Reuben Foster, LB):
(Antonio Morrison, 3, ILB)
(Casey Matthews, 2.5, ILB)
(Jeremiah Attaochu, 2.666666667, OLB)

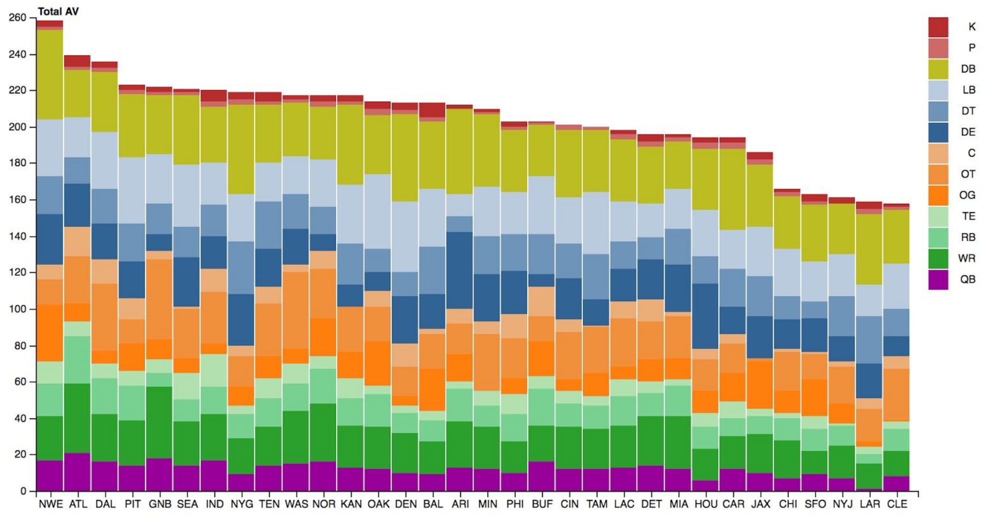
Predicted average seasonal AV: 2.72

Similar players to (Ryan Ramczyk, OT):
(Shon Coleman, 0, OT)
(Paul Cornick, 2, OT)
(Kyle Murphy, 0, OT)

Predicted average seasonal AV: 0.67



Team Composition Analysis



Our team composition analysis was done with rosters since 1978 (the year the NFL switched to a 16 game regular season). The visualization below displays the sum of the AVs of each position for every team in the 2016 season. The total AV of each team is highly correlated with its success in the 2016 season and playoffs. Balance in team composition is important; the best-performing teams receive significant contributions from all positions.

Introduction

In order to assess the value of an individual player in the context of a team's roster, it was important to understand how a given position contributes to the overall success of a team in the playoffs. In order to accomplish this we build a model that uses position-wise summary statistics for team rosters as feature sets to predict the playoff performance of real and hypothetical teams. This step represents an important extension of the aforementioned value estimations of a player because a player's performance and contribution is 'epistatic' to the composition of the team that they're placed in.

Data and Feature Engineering

We trained our model on summary statistics with respect to each position. These included the mean, sum, and max AV by position as well as the number of players for a given position. We felt that using summary statistics would reduce the dimensionality of our roster data and also provide a reasonable assessment of a team's composition. We use roster data since 1978 to provide the players and their AV's. Mean, sum and count highlight a team's strength or weakness at a given position. Including the maximum AV for a position as a feature allows our model to incorporate information about the presence of a star player. We then normalized these measurements using Z-score normalization. These features serve as input to our classification models that attempt to predict the presence of a team at a given playoff level. We trained using 5 sets of output labels: wildcard, divisional, conference, Super Bowl and Super Bowl win. Data about the standings of teams was used to create binary outcome variables corresponding to a given team's participation in a given round of the playoffs in that year. Players lacking a defined position in our raw data were assigned to a UTIL position. It was assumed that for players lacking AV values, their performance was negligible that season and could be sufficiently represented with an AV of 0. We recognize that the data are necessarily imbalanced towards greater lack of participation at each playoff level and attempt to combat this in our model training. While it would have been ideal to build a completely balanced data set, it would have left us with insufficient training data for some playoff milestones; a balanced dataset of Super Bowl winners and non participants would leave only 76 training data points. Oversampling and adjusted training weights both presented themselves as viable workarounds.

Model Selection and Validation

We elected to use scikit's implementation of a random forest classifier to handle the playoff performance classification tasks. Rather than attempting to train a multiclass classifier, we take a one-vs-all approach and train 5 individual classifiers, each corresponding to participation in a round of the playoffs. In all our tests, this never produced unrealistic predictions where a team is projected to appear in a round without being projected to appear in all previous rounds. We decided to use an RFC because of it's use out-of-the-box ease of use, it's properties as an ensemble classifier and the transparency it affords with regards to understanding the importance of each feature in the classification process. We used 2500 trees per classifier and incorporated out-of-bag error. To address the increasing class disparity for successive playoff rounds, each classifier uses balanced class weighting to upweight rare classes in the training process. This effectively increases the penalty for misclassifying rare classes in a manner inversely proportional to the class' frequency.

We validated these models using 10-fold cross validation. Mean accuracy values for each model range from ~82% to ~96%, increasing for successive playoff rounds. We recognize however, that imbalances in data can facilitate increased accuracy scores as the rarity of the minority class increases. Ideally, given additional time, we'd explore the confusion matrix for each of the classifiers to ascertain whether the classifier was accurately predicting the labels for features draw from the minority class. All things considered, the performance of the classifier predicting playoff participation was generally satisfactory. Had the classifier labeled all teams as failing to participate in the playoffs, it would achieve ~62% accuracy (simply given the distribution of classes). ~82% accuracy represented a marked improvement over the expected accuracy obtained by randomly classifying each of the teams.

Results

Examination of the feature importances for each of the models reveals several trends. Collective and star performance of a team's defensive backs and quarterback tend to be the best predictors of playoff success. Defensive player stats tend to make up the majority of the top 10 importance values at each playoff level, with the exception of the conference championship. Defensive back and linebacker performance contribute significantly to determining a team's success in the Super Bowl, suggesting that there is some truth to the adage that "defense wins championships". Future work should be completed to recursively reduce features in order to decrease the complexity of the model and reduce the overall level of overfitting that may be occurring. Perhaps the most interesting opportunity for future exploration is the continued improvement of the performance prediction engine. By modifying test rosters to reflect hypothetical trades, we can begin to use the predicted values, performance metrics and dollar values of players to evaluate the quality of potential trades for a given team.

```
Wildcard Team Validation Accuracy: 0.8238063660477453
Divisional Team Validation Accuracy: 0.8195560900557979
Conference Team Validation Accuracy: 0.8717078281806512
Superbowl Team Validation Accuracy: 0.9341428264572625
Championship Team Validation Accuracy: 0.9666445623342174

Top 10 Wildcard Team Feature Importances:
--------------------------------------------------------------------------------
DB_sum: 0.06118529384428008
QB_max: 0.05154692417350259
QB_sum: 0.04811392193327032
LB_max: 0.042639577715963134
RB_sum: 0.04128209491201473
LB_mean: 0.03758678630985892
WR_sum: 0.037031583830908266
QB_mean: 0.0329195636292424
LB_sum: 0.03258600411735213
DB_mean: 0.03174979931858905

Top 10 Divisional Team Feature Importances:
--------------------------------------------------------------------------------
QB_max: 0.05577102505565656
QB_sum: 0.05325475158559325
DB_sum: 0.05228798212971767
LB_mean: 0.04491360860380345
LB_sum: 0.04365590177306604
RB_sum: 0.04092598240819676
LB_max: 0.03636103580801604
DB_max: 0.033368863965268517
DB_mean: 0.031070953643710743
QB_mean: 0.03104839079386635

Top 10 Conference Team Feature Importances:
--------------------------------------------------------------------------------
QB_sum: 0.06020847770255992
QB_max: 0.053346690750315805
DB_sum: 0.04864805309066822
LB_mean: 0.04287819478506649
DB_max: 0.04139354035727551
RB_sum: 0.03590308685527129
LB_sum: 0.03482580221346073
RB_mean: 0.03465516327302356
LB_max: 0.034132650470536104
WR_sum: 0.030457340501381016

Top 10 Super Bowl Team Feature Importances:
--------------------------------------------------------------------------------
QB_sum: 0.06193093043447309
DB_sum: 0.05279020976609358
DB_max: 0.04430933160022431
LB_sum: 0.04394208567562981
QB_max: 0.042708966070611504
WR_sum: 0.03939386781401922
RB_mean: 0.03910866665231995
LB_mean: 0.033618108603509854
OG_max: 0.03058084579899318
RB_sum: 0.030470162543895635

Top 10 Championship Team Feature Importances:
--------------------------------------------------------------------------------
DB_sum: 0.12043620312590474
DB_max: 0.07830160918169908
LB_sum: 0.04383877819350605
QB_sum: 0.042198247452404296
WR_mean: 0.03521111232595445
DB_mean: 0.03285608855414266
WR_sum: 0.031033106518011243
RB_sum: 0.030080964153702035
LB_max: 0.0283541572919689
LB_mean: 0.02790688754661466
```

Validation results and top 10 feature importances for an example run of our playoff_classifier.py without a test roster.