Wide Receivers Performance in the NFL

Capstone Project 1 Final Report

Conducted under the supervision of Lucas Allen (mentor at Springboard)

Objective

The goal for this project is to help NFL's general managers to optimize decisions in the recruiting of wide receivers (WR) during the NFL draft. In particular, I provide a deep analysis of which players are best to recruit to satisfy the needs of the team based on their collegiate experience and combine numbers. The data, when complete, would include almost all statistics known of a player prior to their draft and the model would predict one's ability to perform in the professional league.
But how do we quantify **performance**? Originally, my interest was focused on a player's longevity, potentially the total number of receiving yards or touchdowns and if such a metric would meet a threshold set by the team's manager. Finding that the data would be limited, either because older players' performances are difficult to collect or because younger ones have not yet expressed their full potential, the metric chosen would eventually be **the total number of yards during the first 5 seasons.**
In fact, given that the data is limited to players drafted between 2000 and 2011, complete information during their first 5 years in the NFL can be generated. Additionally, a general recruiter is more interested in a player's performances during his first contract, which frequently lasts this amount of time.

Project summary:

This project required extensive data collecting, as I wasn't able to find a resource which explicitly contained each draftee's performance as a college player, during the combine and as a professional player. Nevertheless, I was able to use the following resources.

Pro-football-reference (and its affiliate sports-reference.com): A website which contains all imaginable sets of stats for each nfl players and which is easy to scrap. However, the website presented major difficulties in that:

- frequently, a set of data per player was missing,
- the data was spread among multiple pages,
- the data's format lacked consistency from one player to another.

Kaggle: A resource which contains the combine numbers of all draftees from 2000 to 2017, easily exportable in a csv file, but which unfortunately is filled with missing data, either by default or by error.

Finally, the project and this project report is divided into 3 main sections:

1. **the data wrangling**,
2. **the data story** which includes the inferential statistics report,
3. and **the in-depth analysis:** the formulation of a machine learning based model which aims at predicting the target value depending on the collegiate and combine performances.

Note to reader: When I started this project, I had very confidence that I would be able to capitalize on the data gathered and create a model that would predict how well a drafted player would perform in the NFL. As I've advanced in the completion of the project, it became clear that the target suffers a lot of randomness and that the feature data I collected wouldn't suffice in building a robust predictive model. To correct for that, I moved away from a linear regression problem to a classification one and conducted a **second in-depth analysis** to this project. As of this writing, I'm still exploring options that would improve the performance of this model based on the original target, which can be found in the additional section named **future directions**.

Data Wrangling

The methodology to obtain a clean data set is simple on principle:

1. Use BeautifulSoup to scrape pro-football-reference and to collect the NFL and CFB data for each player drafted between the year 2000 and 2011. Report the data in a csv file, where the rows correspond to the players and the columns are the nfl and college stats.
2. Download the combine file from Kaggle and save the data of WR from 2000 to 2011 only.
3. Merge the two by draft's year and pick or by name.
4. Clean the data.

1. The Draft table

This is the more complicated step. I use the website pro-football-reference to capture the nfl and cfb data in 3 steps. Most of this involves using the package *Beautiful Soup*. Since the tables of draftees and the players' profiles are spanned over multiple web pages, the following work is done through automated functions.

A. Generate the table of draftees 'draft table':

This table is preloaded with basic information (player's name, draft year position, college) and basic NFL stats (aggregate nfl touchdowns, aggregate yards). It is properly formatted and there's little work to do besides providing the url to the Beautiful Soup package, which scrapes the page and turns it into a Soup document. The difficulty comes down to locating and scraping the table in Soup format using the 'find_all' command, and then turning it into a dataframe. The columns represent basic sports information, but very importantly, also contain two urls:

- cfb url, which redirects to the player's college football profile
- nfl url, which redirects to the player's nfl complete profile.

Note that a moderate proportion of players have the nfl and cfb url missing. For the former, nothing we can do, but for the latter, I generate the url link by concatenating the player's first and last name as follow:
cfb url = 'https://www.sports-reference.com/cfb/players/'+*firstname*+'-'+*lastname*+'-1.html'

I then expand the draft table to accomodate for additional columns, and using a simple for loop, use the two urls to scrape for additional data. The idea is then to merge this new data in those additional columns.

B. The CFB entries

This part uses the sports-reference website. Using the provided cfb url, I reproduce the same approach as for the ‘draft table’. In turn, I’m able to generate the ‘cfb table’, a table of ‘Receiving and Rushing’ football stats at the college level.
I only worry about the overall cfb stats of a player and use an additional function to generate and merge that data into the ‘draft table’.

Issues

- Precaution is needed for players having played in multiple schools.
- There is also a substantial number of players drafted as wide receivers, but who had a different position in college. For those, the ‘Receiving and Rushing’ stats are a bit further into the web page. Unfortunately, BeautifulSoup is limited in that it cannot scrape all the tables of a web page; in Soup terms, if the table is between consecutive <a! symbols, it is hidden to the package. Therefore, for players whose table of interest isn’t rendered by Beautiful Soup, I use the webdriver.Chrome package, which allows us to capture the proper table by name recognition.
- Finally, let’s make sure that the ‘Receiving and Rushing’ table displays the data in proper order: for those with better rushing than receiving stats, I apply an additional function to generate the row of numbers in proper format

Of course, if the program fails to open the page, the function returns NULL entries.

C. The NFL entries

I reproduce the same approach as I did with ‘cfb table’ to generate ‘nfl table’, a table of receiving stats in the NFL. I once again focus on the aggregate numbers and merge the data into the corresponding row of the ‘draft table’.
Scraping this advanced nfl data is more complicated than its cfb counterpart, because of the difference in nature of each player’s nfl career. A majority end up playing as WR, but some do not play at all, some get to play very little and some are drafted as WR and yet play at a different position in their career. This abundance of cases complicated the scraping process and required multiple sets of data collecting / cleaning summarized in the following functions:

1) main0: making no assumptions whatsoever, generates the ‘nfl table’
2) main1: verifies the title of the table match with ‘Receiving and Rushing’ stats; otherwise, it uses the webdriver.Chrome package to search for other tables
3) main2: verifies the receiving yards generated are correct
4) main3: manually investigate the data generated and search for errors

Other Issues

- The aggregate row isn’t necessarily the last row of nfl table (e.g. players having played for multiple NFL teams).

- I make sure that the receiving and rushing stats are properly ordered.

Of course, if the url link provided fails to open a page, the function returns NULL entries.

2. The Combine table

This step is much easier. A csv document downloaded from the Kaggle website contained the combine data for all the players drafted between 2000 and 2017. I only need the sets of WR from the 2000 combine to 2011, included. To do so, a simple boolean conditional on the column 'Year' and the column 'Pos' can be used.
Later through the project, I realized that some of the players drafted as WR didn't enter the combine as WR. Therefore, the position conditional of the boolean method was removed.
The result is a data frame of combine numbers for wide receivers. The 'combine table' file can then be exported.

3. Put it all Together

Merging

Once the 'combine table' is generated, I add additional columns to the 'draft table' corresponding to the combine data, and match the two sets of data by year and name or by year and draft's position.

Searching

Since there is a substantial amount of players with no combine data, I use the nfl url from 'draft table' and generate missing combine data from the Pro-football-reference website using the webdriver.Chrome package. I then carefully reformat the entries to match those generated by the merging method mentioned above.

The Method Columns

Through these data collecting steps, for each player and for each set of stats (nfl, cfb or combine), we also note the generating method used every single, which is stored in additional columns of the extended data frame. These notes serve as comments for the user, to know how the data was generated and how complete it is.

The 'cfb method' column:

- success: cfb data is good
- fail: cfb data is missing

The 'nfl method' column:

- success from main, main1, main2, or main3: nfl data is good
- fail (eg link broken): no nfl data

- fail after main1: player never had 'Receiving & Rushing' stats (he was used for something else)
- fail from main2: player with mismatch between ''receiving yards' and 'nfl yards' . There shouldn't be any of those

The 'combine method':

- success from merge, search: combine data is good
- fail after search (missing table): no combine table was found on the nfl web page
- fail after search (missing link): no url was even provided

Summary

combine method	**fail after search (missing link)**	**fail after search (missing table)**	**success from merge**
cfb method			
fail (eg link broken)	8	14	31
success	10	30	300

Out of 393 data points, there is a total of 300 with complete cfb and combine data, while 31 have combine data only, 40 have cfb data only and 22 have neither.

Generating Additional Data

Since the basic nfl data only contained aggregate career numbers for each player, I figured it would be necessary to put to good use the complete nfl statistics I generated for each of them. In fact, if we're interested in performances during the first years of a player's NFL career, we would need to get the year by year data of each player. I do so using the 'nfl table' mentioned above.
I can now generate two new metrics:

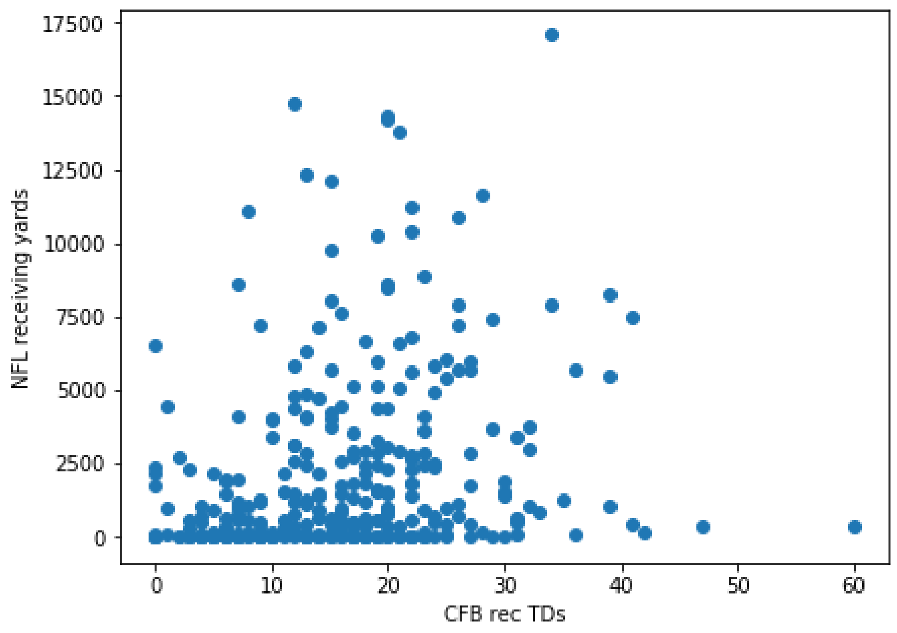
- # of seasons with yards > threshold
- # of rec yards during the first 5 seasons

4. Data Cleaning

Most of the data cleaning consisted of the following steps:
- Re-organizing the csv file to be more understable: renaming columns, changing order of columns.
- Replacing error messages with np.nan entries for missing data and with the value '0' when it is certain (when a player doesn't have any rushing stats for example).
- Catch leftover errors in the nfl and cfb data via automated functions or manually
- Carefully convert all numerical values to floats with a few exception (such as years and draft positions)

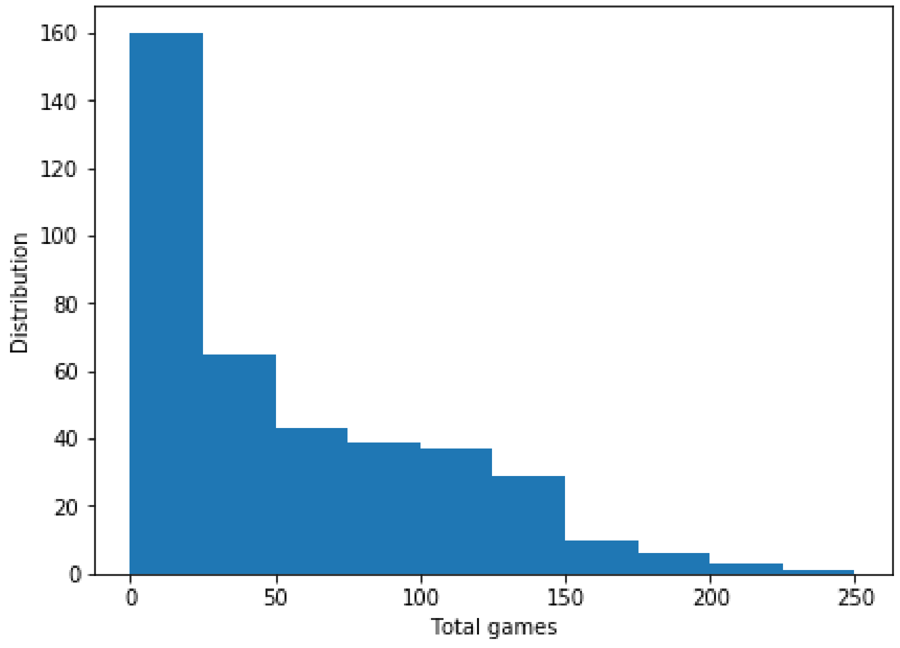
Data Story - Statistical Analysis

Now that the data frame is cleaned and saved in csv format as ‘wr_tdc_new.csv’ (‘wr_tdc_clean.csv’ missed the last 2 metrics I generated), I can finally explore some of the trends in players’ performance and derive rigorous statistical tests aimed at highlighting the impact of categories on the target metric.
In fact, I am now looking to put in evidence how the cfb and combine data impact the nfl data. Of course, the ultimate goal was to narrow it to a specific target, but to start with, I explore the more evident numbers: the number of touchdowns and the number of receiving yards.



On this scatter plot, we discern that the nfl receiving yards increase with the number of receiving touchdowns in college. But what’s striking even more is the concentration of data near the x-axis, suggesting that even some of the most performing athletes in college can get little to zero scores in the NFL.

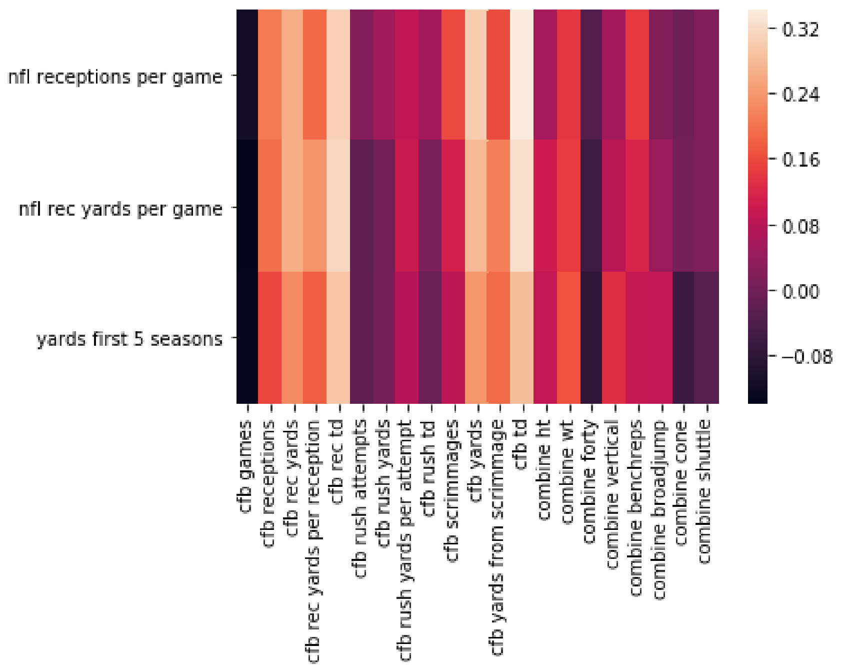
This has to do with the amount of playtime given. In fact, the histogram of number of NFL games confirms that a lot of players simply don’t ever get to play much.



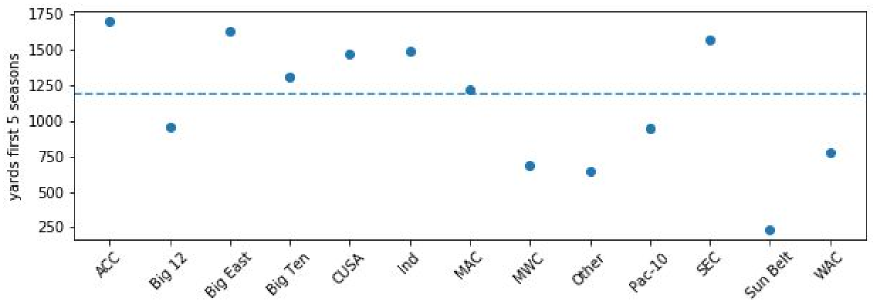
Correlation Matrix and Non-numerical Data

Given the extensive amount of variables there is (a total of 37 nfl stats and 24 cfb/combine stats), I decided to compute the correlation factor between each of the potential targets (Y_i) and the input variables (X_j): C_{ij} . Of course, this can only be done for numerical variables. I then derive the aggregate correlation factor for each nfl variable, and select those with the highest value - those that correlate the most with the cfb and combine data. 3 nfl variables are picked as the potential targets and I plot the correlation of those with the rest of the data:

- NFL receptions per game
- NFL receiving yards per game
- Number of yards during first 5 NFL seasons



I also consider the non numerical variables: the conference of origin (represented in the figure below) and the class status of a player upon entry in the draft. Results for the number of yards during the first 5 seasons are shown below.



All three potential targets present very similar results. As I've stated at the beginning of this report, for equity purposes, it is best then to choose the target $y =$'yards first 5 seasons'. The choice of an overall career metric instead would be falsified, since we're dealing with players drafted up until 2011 and who are at the midpoint of their NFL career.

Statistical Tests

I run three statistical tests to emphasize the correlation (or the lack thereof) between the target and a set of input variables. A level of confidence of $\alpha = 95\%$ is chosen for all tests.

1. Test1: College Football Conference / SEC vs Big12

I run a bootstrap inference on those two categories and I do it with respect to the mean of each of the metric y . The null hypothesis can be stated as "The sec and big12 samples are from the same distribution with respect to the mean of y" and found the p-value: p = 0.0213

2. Test2: College Football Rush Attempts

Group1 and group2 are the subgroups of players with cfb rush attempts respectively below and above the median. This time, I test for the independence of the two samples with respect to the target using a scipytest and find: p = 0.3569

3. Test3: cfb receiving touchdown

Group1 and group2 are the subgroups of players with cfb receiving touchdowns respectively below and above the mean. I follow the same procedure as test2 and find p = 0.00000060

Test1 and test3 return on average small p-values, outside the confidence interval I chose [0.025, 0.975]. This indicates that under the stated null hypothesis, obtaining a result as the one observed is very unlikely, meaning we can dismiss the hypothesis. Test2 on the contrary returns large p-values, meaning the null hypothesis is likely to be correct at least with the degree of confidence chosen, once again confirming what can be seen on the correlation heatmap.

In-Depth Analysis (1)

Despite my effort to collect and organize my data set, I must acknowledge that it is pretty challenging and most importantly... limited. For recall, my target is **y = yards first 5 seasons** and we have a total of **23 input variables**, including two non-numerical features: the conference and the class status upon entry in the draft. Unfortunately, there's only a total of 393 data points and only roughly 75% of the numerical data is complete - 300/ 393 to be exact. If you account for the discrete features, this number even drops to 273/393...

But most importantly, very noticeably, the data is very spread out. A lot of players end up having 0 play time in the NFL, either because of lack of competitiveness or because of injuries, despite very satisfying performances in college football or in the combine. For example, as I've indicated earlier in this report, $X =$ cfb_td, the variable which yet correlates the most with y, still seems to have an inconsistent impact on the player's NFL performances. For a lot of players, the two trends are positively correlated, but for others, it doesn't and $y = 0$ regardless of the value for X. This fact is precisely what makes building this predictable model particularly challenging. Hence, for this model, my strategy mostly consisted in **applying transformation to the data set** (by either reducing or augmenting the number of features or the number of rows) and by using **different linear regressors**.

The metric for model tuning

Throughout my multiple attempts at building supervised machine learning models on the different versions of the data set, I computed the coefficient of determination R^2 and very importantly, the root-mean-square deviation **RMSE,** which I looked to minimize. I derived those two metrics using a cross validation test with 10 folds and returned the average values for each of those. As I progressed in my project, it is worth notice that I also accounted for the **CV score** of the model, averaged over 5 folds.

Multiple attempts to improve Algorithm's Performance

It is worth noting that before I started all these tryouts, a total of 5 feature columns from the combine results exhibited more missing data that their counterparts: 'combine_vertical','combine_benchreps','combine_broadjump','combine_cone','combine_shuttle'

Given that those have very little importance in players' overall combine performance and that a lot of them do not attempt those combine tests, I simply got rid of these variables to reduce the dimensionality of the feature variable **X** to 18 columns.

I then proceeded to apply the **LinearRegression** regressor from Scikit-Learn on different versions of the data:

1) All the data

By considering X entirely and dropping the missing entries, the model returns a root-mean-square error averaged over 10 folds of RMSE = 1515.6908. Given that number of yards range between -1.0 to 6201.0, this was a substantial number.

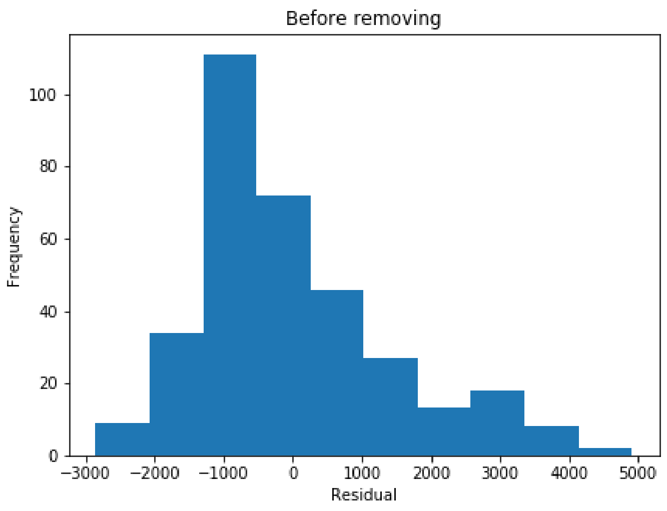
2) Reducing X to 6 essential features

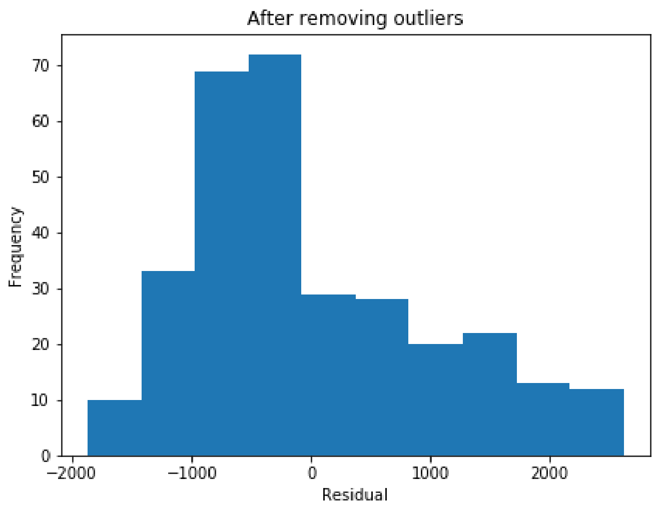
I then reduced the features variables to the 6 most essential, those that correlate the most with the target y, namely:
X' = ['cfb_rectd', 'cfb_td', 'cfb_yards', 'cfb_recyards', 'cfb_yardsfromscrimmage', 'cfb_games']

This allowed to reduce the root mean square error to RMSE = 1437.9412.

3) Removing outliers

I then decided to use the **ordinary least-squares** function of Statsmodel in order to remove some of the outrageous outliers of the model. I picked those outliers to be defined by having a residual greater than 2,500 when running a simple linear regressor on the entire data (not using the train/test split method). I additionally plotted the distribution of these residuals before and after removing those data points:





Yes, the residual distribution wasn’t nearly as spread, but most importantly, they weren’t normally distributed, indicating once again how challenging the data set is. Still, I was able to reduce that residual to: RMSE = 1363.6585 when running a cross validation test on the train/test split data.

4) Non-numerical Features

I then considered the features X' along with the non-numerical variables:
Cfb_conference, cfb_class

For the sake of the linear regression, I was able to turn those features into numerical variables using the **get_dummies** function of Pandas. But unfortunately, I did not observe a reduction in the residual.

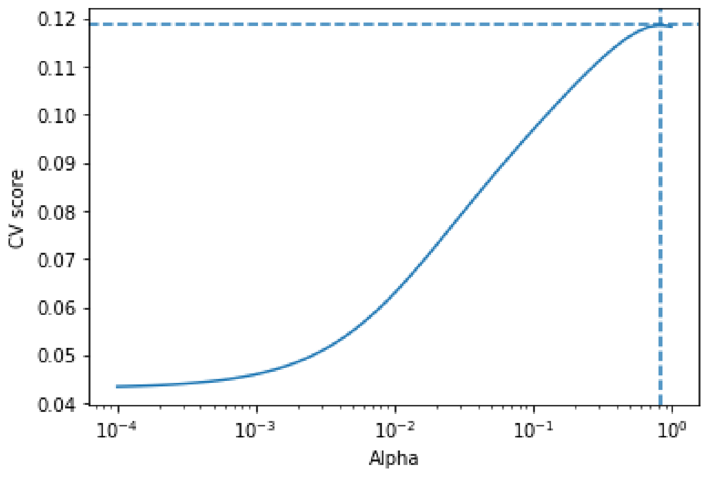
5) Applying an Imputer

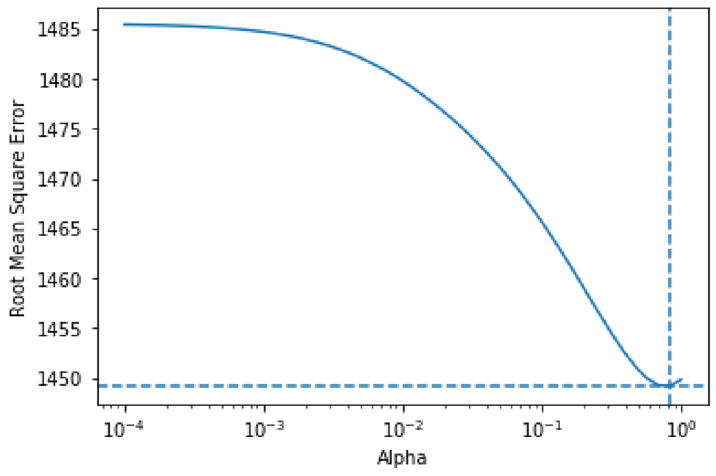
Up until now, I was removing every row which included a missing number, thus considerably reducing the total number of points. In order to handle the missing data, I then applied the imputer from **SimpleImputer** (Scikit-Learn) which would fill the feature data using a *mean value* based strategy. Doing so, I was able to find a modest RMSE = 1383.1872.

Generating variations of the data set wasn't the only objective; I also used a different regressor to build the model. The **Ridge Linear Regressor** from Scikit-Learn offered the advantage of minimizing the residual (or to improve the cv score) by tuning the α parameter of the regressor. I did so in two different circumstances, which I explain next.

6) Ridge Regressor on all data

With the original data and without applying the imputer, I found the RMSE to reach its minimum at 1449.1586 for α = 0.8286. For this same parameter, the cv score returned was then = 0.118580





7) Normalizing and Nonlinear data

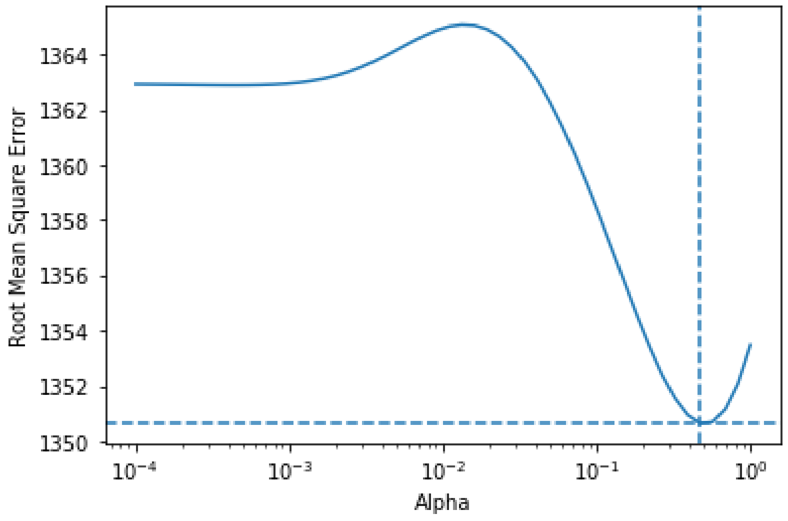
Given that some input statistics were not on the same scale (for example, cfb_td would range between 1 and 61, while cfb_recyards would between 2.0 and 4414.0), I decided to normalize all feature variable X. Unfortunately, the best cv score was identical than in the 6th attempt.

Given the number of TD's in college was the most important feature variable, I could suspect a nonlinear behavior between such metric and the target y; which is why I decided to run the linear regressor on the **square of cfb_td**, but found once again that the best cv score remained stagnant: cv_score = 0.11195 (note here, the best parameter for α was then = 0.68664)

Conclusion to 1st model

By considering all these attempts, I came up with a final algorithm and final data set, which included:
- choosing the most important features X' along with the non-numerical variables (college conference and college class)
- applying an imputer with mean value based strategy to handle the missing data
- run a ridge linear algorithm

The returned RMSE was 1350.646



As you can see, even in the best scenario, the root mean square error is still pretty high. Those aren't satisfactory results, mostly because of the challenging nature of the feature data (missing values, collinearity between input variables, zero-inflated target).
Even though a team's general manager wouldn't be particularly pleased with the predictions coming from this model, there still are lessons to be drawn from it:

- the positive correlation between the target and for example, the number of touchdowns and the number of yards in college, which at first hand, seemed evident.
- the negative correlation between the target and the number of games, highlighting that a player who is drafted as a junior rather than as a senior, tends to perform better.
- the lack of correlation between the target and most of the combine features, pointing out that the combine performances shouldn't be factored in a recruiting decision as much as the experience in college.
- the importance of the conference of origin: for example, among the power 5 conferences, the ACC and the SEC perform much better on average, compared to the Big 12.

In Depth Analysis (2)

Throughout this project, I hoped to design a predictive model based on the collegiate and combine performances, which, when fed a particular data set, would return a predicted value for the number of yards by a drafted WR during its first 5 years in the NFL. Since the first model didn't provide satisfactory predictions, I designed a complementary model, one based on **Logistic Regression** instead.

The goal now is to classify players between **flops**, those for which the target is below a certain threshold, and the others, i.e. the **successes**, implying that the new target is binary value based. Clearly, varying this threshold has a major influence on the model's performance and it could be tuned in order to provide even better satisfying levels of prediction. As of this writing though, I do not have conclusive results on that (check future directions) and have set the threshold to be 100:

- $y \leq 100 \twoheadrightarrow y_{new} = 0$ (flop)
- $y > 100 \twoheadrightarrow y_{new} = 1$ (success)

Even though I attempted multiple classifier for this study, I eventually used the features of **LogisticRegression** from Scikit-Learn to build a predictive model with respect to the new target, but using the same features data: all feature variables are included, the numerical inputs are normalized using *MinMaxScaler* (Scikit-Learn), the non-numerical variables (conference and class) are quantized using *get_dummies* (Pandas) and the missing values are handled with *SimpleImputer* (Scikit-Learn) using a mean value based strategy.

The predictive attempts obviously consisted in doing train/test splits on the existing WR data, which for recall, ranges from the draft year 2000 to 2011. While doing a cross validation test, the accuracy of the model on average was relatively satisfactory (cv_score = 0.66). Therefore, I decided to **test the model on the draft class of the year 2020.**

I generated the college and combine data of this new batch of players and built the logistic regressor entirely on the data set of WR's drafted from 2000 to 2011. Since, there's no actual NFL data for those players (as of this writing), we can't compute the accuracy of our predictive model against this new data set, but we can derive the **probability of success P** for each player using the *predict_proba* function of the regressor.
In the following tables, I display the probability P for each player. On the first table, the players are ranked by their real draft position, while on the second, they are sorted by the value of P. The first 10 entries each time are displayed:

	player	nflteam	college	round	pick	chance_of_success
0	Henry Ruggs	VEG	Alabama	1	12	0.941627
1	Jerry Jeudy	DEN	Alabama	1	15	0.876993
2	CeeDee Lamb	DAL	Oklahoma	1	17	0.902489
3	Jalen Reagor	PHI	TCU	1	21	0.887495
4	Justin Jefferson	MIN	LSU	1	22	0.902911
5	Brandon Aiyuk	SFO	Arizona St.	1	25	0.534193
6	Tee Higgins	CIN	Clemson	2	33	0.960224
7	Michael Pittman	IND	USC	2	34	0.746665
8	Laviska Shenault	JAX	Colorado	2	42	0.746760
9	KJ Hamler	DEN	Penn St.	2	46	0.638380

	player	nflteam	college	round	pick	chance_of_success
6	Tee Higgins	CIN	Clemson	2	33	0.960224
0	Henry Ruggs	VEG	Alabama	1	12	0.941627
13	Lynn Bowden	VEG	Kentucky	3	80	0.919781
4	Justin Jefferson	MIN	LSU	1	22	0.902911
2	CeeDee Lamb	DAL	Oklahoma	1	17	0.902489
3	Jalen Reagor	PHI	TCU	1	21	0.887495
10	Chase Claypool	PIT	Notre Dame	2	49	0.884979
19	Tyler Johnson	TAM	Minnesota	5	161	0.882362
1	Jerry Jeudy	DEN	Alabama	1	15	0.876993
26	Donovan Peoples-Jones	CLE	Michigan	6	187	0.875465

As we can see, the model still predicts high *‘chance_of_success’* levels for the top players of the 2020 draft, but noticeably, it also reshuffles their respective positions and points out that some may even have deserved a higher draft position. Most certainly, this is valuable information for a general manager when it comes to drafting wide receivers in the league.

Future Directions

Turning the model into a classification algorithm allowed us to make better predictions with respect to the chance of success of a player. Yet, there still is room to pursue my original idea of predicting the number of yards during the first 5 seasons in the NFL.

A potential solution that I started to explore is to follow a "**risk-severity strategy**". Because the data is heavily zero-inflated, I can build a Regressor on the successes data only OR build two separate Regressors, one for the flops and one for the successes. The building of the model would consist of:
- setting and varying a threshold value which would serve as the classifier for the target value,
- building the classifier using the train data,
- and building a regressor (or two regressors) using all the successes (and the flops) from the train data.

The predictive analysis would then include:
- applying the classifier on each row of the testing data,
- and applying the regressor depending on the predicted outcome of the classifier.