FANTASY FOOTBALL DRAFT SIMULATOR

John Crawford

Christopher Hinshaw

Jack Horgan

INTRODUCTION

For our capstone project, we decided to apply some of the Python and math skills we learned in this class to something all of us are passionate about: fantasy sports! Fantasy sports is when the average fan of any sports league (NFL, NBA, Premier League, etc.) puts together teams of individual players in that league. Individual players score fantasy points based on the statistics they accrue in-game, adding to a fantasy team's total points. In a normal fantasy league, there are usually 10-14 teams in the league, with each team facing off head-to-head against another opponent every week. Whoever's team scores the most points at the end of the week wins, and a new opponent is matched up for the next week. Based on win-loss records, a certain number of fantasy teams advance to the playoffs, where there is a bracket of head-to-heads until someone comes out as the champion. Teams are determined by a draft at the beginning of the season.

Specifically, we focused on the great sport of American football for our fantasy league. Our goal: to create an automated program that simulates fantasy football drafts. We wanted to create a draft strategy that outperformed 'autodraft', which is where the computer just takes the top projected player. This draft strategy would lead to a win-rate that is statistically significant from the expected win rate of 50%. In order to create the draft strategy, we had to define the rules of our league.

LEAGUE SETTINGS & GAME RULES

One of the main formats of fantasy football is “best-ball”. Essentially, instead of setting a starting lineup before each week and making transactions on the waiver wire, you draft a lot more players, where you don’t set a starting lineup. After the week is over and the scores come in, a starting lineup is automatically generated, maximizing the points you score with the players on your team. Similar to real sports, not every player starts for your fantasy team on any given week. There are bench players, who act as backups in case something happens to the starters (bye week, injury, poor performance, etc.). In best ball, there are no bench players, since the lineup is simply determined by which players score the most, bound to positional constraints. For our best ball league, we defined a starting lineup as containing one quarterback, two running backs, two wide receivers, one tight end, and one FLEX position (any position except quarterback). These are the positions that will be summed for the team score at the end of each week.

The conversion from in-game statistics to fantasy points differs slightly from league-to league. For our league, we outline the scoring system as this:

Statistic	**Points/unit**
Pass Yard	0.04
Interception	-1
Rush/Receiving Yard	0.1
Rush/Receiving Touchdown	6
Reception	1

Now that the league settings are defined, we need data to input into our program.

DATA

There were a couple of datasets we needed in order to generate our drafts and test their performance throughout an entire season. Firstly, we needed to quantify the value of a player to base the draft off of. We did this through season-long statistical projections. Fantasy sites such as ESPN publish these season-long projections to base their drafts off of before each season. Since we needed a full season of results to test, we couldn't use 2022 season-long projections as the season is still going.

Unfortunately, these fantasy sites don't archive their season-long projections from previous years, even though they are openly available, because they are only focused on the current season; there's not really a use for them unless you're doing a niche project such as our own. Luckily, we were able to find the season-long projections from ESPN for the 2020 season on Kaggle. Once we obtained those through a locally downloaded csv, it was easy to calculate projected fantasy points based on our scoring system to base our draft off of from there.

Secondly, we needed a dataset that contained the weekly scores of every player in order to total up the fantasy points for our lineup each week in our simulated head-to-head matchups. We found a giant, open-source data pool on Github, that contained the weekly scores for each week in each year. These following formulaic URLs, which allowed us to scrape the data from github into our program without downloading them locally. There is a separate dataset for each week, which includes the player name and their PPR (point-per-reception) fantasy score, which matches the format we are playing. Now we have everything we need to start the program.

CODE BREAKDOWN

Note: All of our code was made from scratch, and no open-source code was pulled for this project. The .ipynb file can simply be run top to bottom when using the code.

DATA IMPORT AND PREPERATION

The ESPN projections were provided in separate datasets, with each position getting its own. To start, I imported all four of these into separate Pandas dataframes, and renamed the columns to match, as there were naming discrepancies across positions. Using pandas.concat(), I was able to combine all the positional projections into one large dataset, to more easily clean the data without having to run through the same cleaning with all four separate datasets. Using our point scoring formula outlined in the league setting section, it was easy to calculate projected fantasy points by applying this formula on the pandas columns, and storing the result in a new column. The dataset also includes every player at that position on an NFL roster, so to trim it down, I removed any players who were projected for 0 fantasy points from the dataset, since we don't want our team or the auto draft to be selecting these players.

Something we added later was randomization to the projections. There could be some tendency our draft program falls into when using the same projected fantasy points for every draft, so in order to remove this possible confounding variable, I added slight randomization multipliers to each statistical category using numpy.random.random_sample. This random multiplier scaled from 0.925 to 1.075, so only slight changes. Everytime the program is run, a new projection set would be output to use as the basis for the draft. Another late addition was positional weighting to fantasy points. Since quarterbacks tend to be the highest projected players, the auto draft was taking them early and often, which is not truly representative of a real fantasy draft. A multiplier of 0.8 to projected quarterback fantasy points provided a better balance.

After this is all complete, we need to split the overall projection data frame back into separate positional data frames in order to calculate positional z-scores for fantasy points, which is used for our draft strategy which we will outline later. Using the .mean and .std functions, I was able to calculate these values for the fantasy points column, to input into the z-score formula. This formula would be run on the fantasy points column with the results being stored in a new column in the dataframe.

Example Projection Data-frame for QB position:

	Player	Tm	Pos	Car	RushYards	RushTD	Rec	RecYards	RecTD	PassYards	PassTD	Int	FPTS	Zscore
0	Patrick Mahomes	Chiefs	QB	58.025670	296.686210	2.836726	0.0	0.0	0.0	4709.321155	32.890386	9.676153	356.947216	1.700953
1	Lamar Jackson	Ravens	QB	154.406087	858.101587	5.620425	0.0	0.0	0.0	3211.387936	26.221255	9.912986	342.960260	1.589554
2	Kyler Murray	Cardinals	QB	88.663743	497.303392	2.936444	0.0	0.0	0.0	4002.456015	27.672299	11.489243	326.647197	1.459629
3	Deshaun Watson	Texans	QB	85.739496	537.693153	4.039694	0.0	0.0	0.0	3831.178626	26.462588	11.956530	325.148367	1.447692
4	Tom Brady	Buccaneers	QB	22.814055	27.352354	2.133119	0.0	0.0	0.0	4658.532236	30.506539	9.244651	314.656741	1.364131
5	Dak Prescott	Cowboys	QB	49.892307	248.466339	3.873570	0.0	0.0	0.0	4266.816222	25.394532	10.401697	309.937133	1.326542
6	Russell Wilson	Seahawks	QB	70.193507	425.778148	2.905879	0.0	0.0	0.0	3547.153508	29.618068	8.953416	306.018089	1.295329
7	Matt Ryan	Falcons	QB	31.431126	115.750720	0.979917	0.0	0.0	0.0	4334.075992	25.921672	11.702813	282.801486	1.110421
8	Josh Allen	Bills	QB	101.095224	506.355704	5.030467	0.0	0.0	0.0	3322.774678	19.171318	11.735287	278.679345	1.077590

WHY Z-SCORES

The Z-score calculates how many standard deviations away from the mean an observed value is from a set of data. This was used to show a comparative advantage over other players. For example, Travis Kelce, a tight end, has a z-score of 3.22, and he was projected to score around 254 points meaning his total projected points are 3.22 standard deviations away from the mean; Patrick Mahomes is a quarterback that has a z-score of 1.7 and is projected to get around 357 fantasy points. Although Kelce is going to score fewer points, he is more valuable than Mahomes because he gives a more significant comparative advantage. The draft strategy uses this to try and get the players with the greatest comparative advantage compared to the players left. It chooses players that are projected to score more than the next best alternative. In comparison to the quarterback, which was normally taken first when just using the highest projection, this strategy would often choose tight

ends or running backs first, depending on availability. The goal was to give up a few points in positions that often had a higher score to gain greater advantages over opponents in positions that often scored fewer points to have an overall advantage for the whole team.

THE DRAFT

Now that we have a projection system to base our draft off of, we can proceed to the draft process. The easiest way to run a draft in Python is through the use of classes, where each team contains its own object to store the draft information. Two variables relating to the team are stored and updated with each call to the class: 'team' and 'player_types'. To prevent a team from taking too much of any given position, the 'player_types' dictionary provides maximum values for each position. Whenever a player of a specific position is drafted, this variable updates to subtract one from the respective value in the dictionary. If one of these values hits zero, then the program will pass on drafting the player of that position and go to the next best option. The 'team' variable is a list, where the drafted player's name and position is appended for every draft selection the team makes.

There are two different draft functions within this 'team' class: one for autodraft, one for our draft. The functions are almost identical, the only difference being the values they use to draft the player. As mentioned previously, our draft takes the highest z-score difference between the top two options at the position, whereas auto draft simply takes the highest projected player. The positional data frames are sorted by fantasy points already (and therefore z-score as well), so to retain the values used for drafting, we can simply take the value of the top row with the .iloc[0] command. In order to keep the player position data, these top values for each position are used as keys in a dictionary with the positional data frame as a value. After sorting the dictionary by its keys, and selecting the top value with respect to the positional constraints, this data frame can be used to store the drafted player's name and position in the 'team' variable.

After setting up this 'Team' class, these objects must be used in conjunction to form an overall draft. To do this, a list is created with 12 team objects. For loops are used to cycle through these objects in snake draft format, using the "OurDraft" function whenever the index is equal to our randomly generated draft spot, and using "AutoDraft" whenever it isn't. After cycling through this for a set number of draft rounds, all the 'team' lists for all twelve 'Team' classes can be combined together in a Pandas dataframe, where each team has its own column. This formats nicely as a draft board, with each row being the round number and each column being the players on a specific team, and also makes it easier for us to pull this data later when creating our head-to-head lineups.

LINEUP SETTING

As previously mentioned, this project is using the "best-ball" format for rules which means that the best performing players fill in our line-up; for the league, this means the best quarterback, two best running backs, two best wide receivers, best tight end, and best-remaining non-quarterback were the players that counted towards the total score that team for the week. To accomplish this, all the points for the players on each team from that week must be collected from the weekly data used. This was originally accomplished by iterating through the entire list, searching by name for each player; however, this was very inefficient and took a long time to run, so the method changed to a "merge left," which took up significantly less time to run. Once all the points had been collected, the positional players were compared to get the players needed to make the best lineup for each team.

SIMULATION AND RESULTS

SIMULATION

We can simulate how our program performs in a given number of n seasons, with each simulation we add to a win record and loss record. After n simulations we can calculate our programs' win percentage. As we increase n we will approximate closer to our program's "true" win percentage.

Below is an example with 100 simulations. Each simulation starts by giving us a random draft position. The other teams in our league draft using Auto-Draft. Which selects the best player on the team using ESPN stats.

```
n= 100 # number of simulations

num_weeks = 16 # number of weeks per simulation
win_list = np.array([]) #record wins for each season
loss_list = np.array([]) #record losses for each season

while n > 0:
    QBs, RBs, WRs, TEs = rand_projections(ffprojections)


    draft_positon = random.randint(1, 12) #randomize spot to select

    Draft = draft(draft_positon, QBs, RBs, WRs, TEs)


    opponents_list = random_opponents(draft_positon, num_weeks)
    opponents = {}
    for x, i in enumerate(opponents_list):
        opponents[x] = Draft[f'Team {i}']

    my_team_totals = []
    opponent_totals = []

    for week in range(1, num_weeks + 1):
        my_team_lineup = lineUp(Draft['My Team'], week, 'My Team')
        my_team_lineup.getLineUp(my_team_lineup.getPoints())
        my_team_totals.append(my_team_lineup.getTotal())
        opponent_lineup = lineUp(opponents[week - 1], week, f'Team {opponents_list[week - 1]}')
        opponent_lineup.getLineUp(opponent_lineup.getPoints())
        opponent_totals.append(opponent_lineup.getTotal())

    wins = 0
    losses = 0
    for week in range(num_weeks):
        if my_team_totals[week] > opponent_totals[week]:
            wins += 1
        else:
            losses += 1
    win_list = np.append(win_list, [wins])
    loss_list = np.append(loss_list, [losses])
    if n % 5 == 0:
        print(f'Simulations Left: {n}')
    n -= 1
print('Simulations Complete! win_list and loss_list can now be used')
```

RANDOMENESS

Python uses pseudo-random numbers, that is the function we use, “random.randint(1, 12)” ,generates a pseudo-random integer between 1 and 12. A pseudo random number is statistically random but is derived from a known starting point. This means it is never truly random.

Examples of how to generate pseudo random numbers are: The time in microseconds of when a code is initialized, the temperature of a given city, or the modulus of an integer divided by the square root of a prime.

RESULTS

We use for-loops to cycle through each game of the season. Using numpy arrays, we record our wins and losses from each simulation. After n simulations, we can calculate the win and loss percentages.

```
win_percentage = win_list.mean() / (win_list.mean() + loss_list.mean())
win_percentage
```

```
0.526875
```

In the above simulation, we have a win percentage of 52.69%.

We can compare our program’s draft strategy vs. autodraft using a one- sample t-test.

```
p_value = stats.ttest_1samp(win_list, popmean= (num_weeks / 2)).pvalue
print(f'p_value: {p_value} \n')
if p_value > 0.05:
    print('Our Draft Strategy is not significantly better than AutoDraft!')
else:
    print('Our Draft Strategy is better than AutoDraft!')
```

```
p_value: 0.06620598690554999

Our Draft Strategy is not better than AutoDraft!
```

A note on one-sample t-tests. A t-test compares the means of two groups. In our case this is the average win rate of our draft strategy vs the win rate of the Autodraft. Our Null Hypothesis is; Our Draft strategy has a mean of wins greater than the Autodraft. Alternative Hypothesis: Our population means are not equal.

We test this to a level of significance of 5%, giving us a confidence level of 95%. A p-value of greater than 0.05 (Large p-value), implies that the population means do not differ. There is no evidence of a difference. We cannot reject the null hypothesis. A p-value of less than 0.05 (Small p-value), implies that the population means differ. There is evidence to reject the null hypothesis.

CONCLUSIONS & REFLECTIONS

LEARNING AMERICAN FOOTBALL

s an Irish exchange student, I came into the project knowing very little about American Football, except “GO GATORS!!”. Thankfully, both John and Chris took me under their wings in teaching me the rules of American Football, the different positions, and all the sport has to offer.

While brainstorming how to most efficiently draft a team, the mathematics behind the project, Z scores, Pseudo-Randomness, and T-tests, was a universal language, and sport a vessel for this language. Each method of drafting that was pitch had to be understood by someone whose knowledge of the sport was limited, namely myself. When preparing our final presentation, this groundwork of pitching our ideas paid dividends, as we had already had excessive practice in the language we were using to describe the project to our classmates, who did not have an interest in the sport.

TEACHING AMERICAN FOOTBALL

When this project started a team member was unfamiliar with american football so the group had to teach the member and make sure that everyone understood football so that there was clear communication about the statistics and ideas that were mentioned when talking about draft strategy, how points were scored, what positions were needed and other essential concepts and vocabulary used while mentioning ideas. This was for the best as it gave insight as to how best explain the concepts for the presentation, having to explain the concepts and ideas of fantasy football to someone for the first time before the presentation. These ideas were then translated over to the presentation to

make sure the entire audience would have enough of an understanding of the game so that the ideas that were being presented could be understood by anyone.

FUTURE IMPLEMENATTIONS

This project was ultimately to test a strategy for drafting and would not be used actually to draft a team; to draft a team would require a different implementation. Once a user has ultimately made the best strategy using the data from this year, the user could take the strategy and implement it into a program that would be able to take user input to remove a player from the draft that has been selected by another team to update the available players they can select from. This could be as simple as having two options such as “Drafting” or “Other Team.” The “Drafting” would run through a similar class used to draft in this project such that it has position limiters so that the user selects a full team. In the “Other Team” option, the user would simply input a player name to remove from the available players list so that the user is not trying to draft players not remaining. This project has a lot of plasticity, meaning if the user wants to use different numbers of players for the positions because the user is using different rules, the user could implement a way to change that with a different constructor potentially for the class that defines the number of quarterbacks, running backs, wide receivers, and tight ends when creating the team object. When looking for ways to improve the draft strategy, it would be recommended to put round limiters on the positions; for example, one cannot have more than one tight end until the rest of the starting positions are full. The user can also implement their own projections or use projections for another company should the individual see fit, the project can bend to many different users as it provides most importantly a framework to try and accomplish the goal within.

INDIVIDUAL CONTRIBUTIONS

The brainstorming for this project was done together to create a project outline. John mainly worked on the data preparation and revised Chris's code for the Draft and LineUp classes to use data frames instead of csv files. Chris laid the foundation of code for both of the classes used as well simulating a single draft and head-to-head matchups for the season. Jack worked with running all of the code multiple times to generate an overall win percentage, and he used a one-sample T-Test to see if the value was statistically significant. He also organized the code and grouped some of the smaller portions of code into general functions to clean up the program.

REFERENCES

"ESPN 2019 STATS AND 2020 NFL FANTASY PROJECTIONS | Kaggle." Kaggle: Your Machine Learning and Data Science Community, https://www.kaggle.com/datasets/mur418/espn-2019-stats-and-2020-nfl-fantasy-projections. Accessed 12 Dec. 2022.

fantasydatapros. "GitHub - Fantasydatapros/Data: Fantasy Football Data in the Form of CSV Files Available for Use in Pandas, R, Excel Etc." GitHub, https://github.com/fantasydatapros/data. Accessed 12 Dec. 2022.