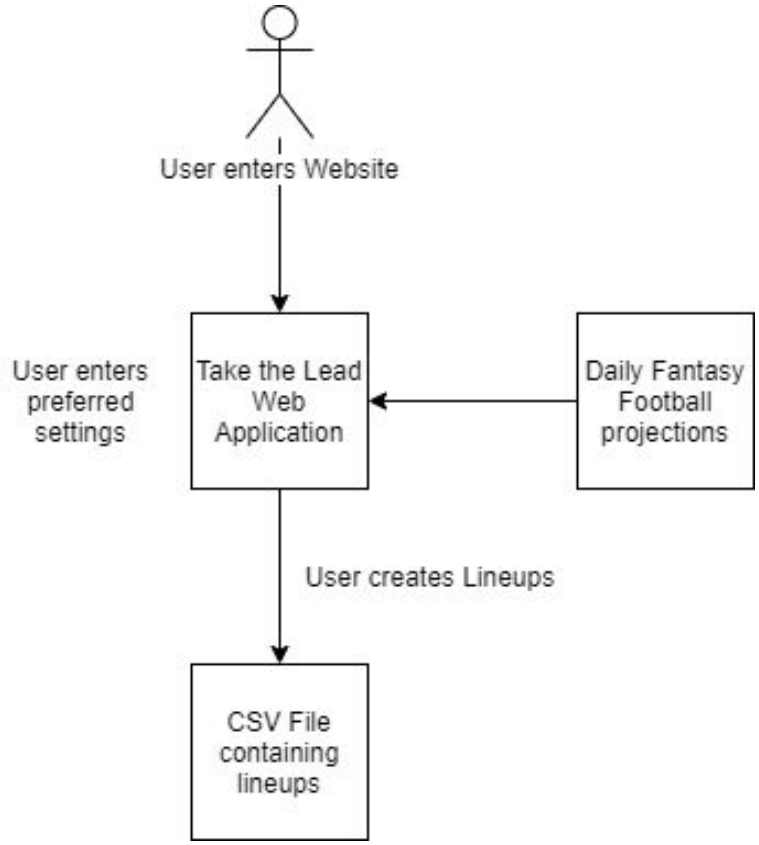
Design HW2

Purpose:

The main purpose of our project is to create a daily fantasy football optimizer that allows users to generate optimal lineups. The goal is to make something that is better than what already exists in the industry. As of now, the optimizers that exist have problems associated with them, such as a subpar algorithm or high difficulty of learning. Our end product will solve those problems, and exist as an easy to use optimizer that generates very strong lineups.

System:

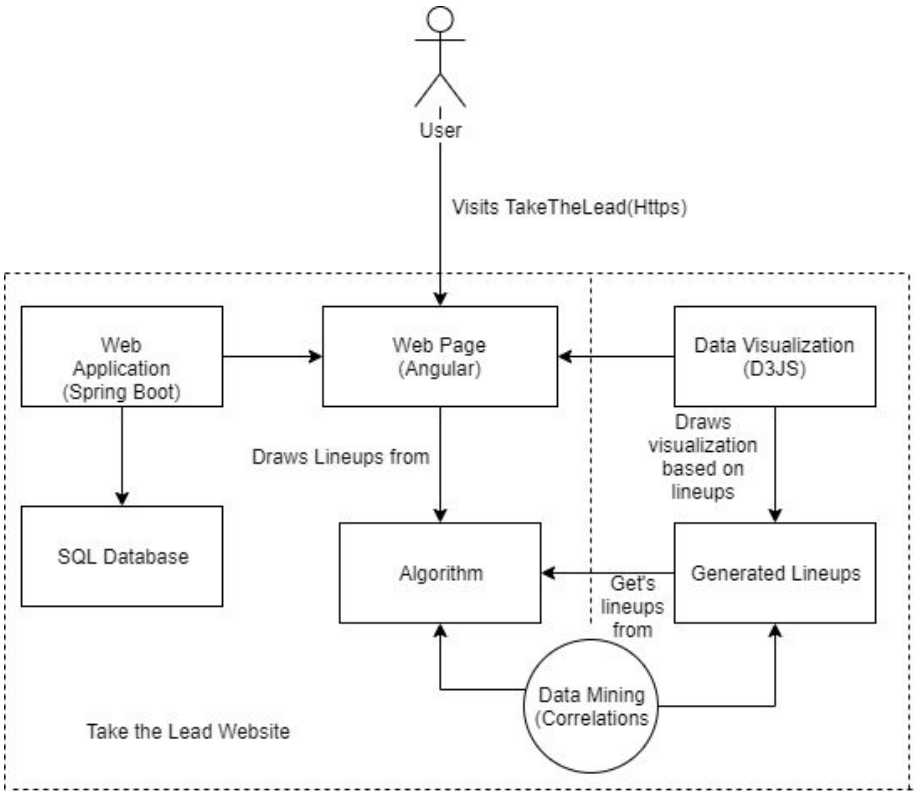


Use Cases:

The user will enter the website. On this web application, there will be imported daily fantasy projections from fantasydata.com. These projections will be factored into the lineup generation algorithm. From there, they can alter the settings to their preference, changing certain attributes such as player exposure. Once their settings are set, they hit the generate button to generate a number of lineups of their choosing. These lineups will be displayed in the interface, showing each player and the

lineup projection. Once the user is satisfied with their lineups, they can export the lineups as a csv file, that of which can be entered on draftkings. This is the high-level flow of the entire system, where the user enters the website, creates lineups, and then exports those lineups in csv format, so that they can be entered on draftkings.com. This is the overall system flow, where the user enters the website, generates lineups based on their preferences, and then exports those lineups as a csv file, so they can be entered on draftkings.com

Containers:



The user enters the HTTPs webpage at the designated URL. From there, the web page is displayed to the user. The Web Page is made in Angular, and displays to the user multiple features. These include the player pool, the settings that can be made to the optimizer, and the lineups once generated. The backend piece of the web application is made with the spring framework. The Spring boot backend will make API calls to the SQL Database, and send necessary information to the Web Page.

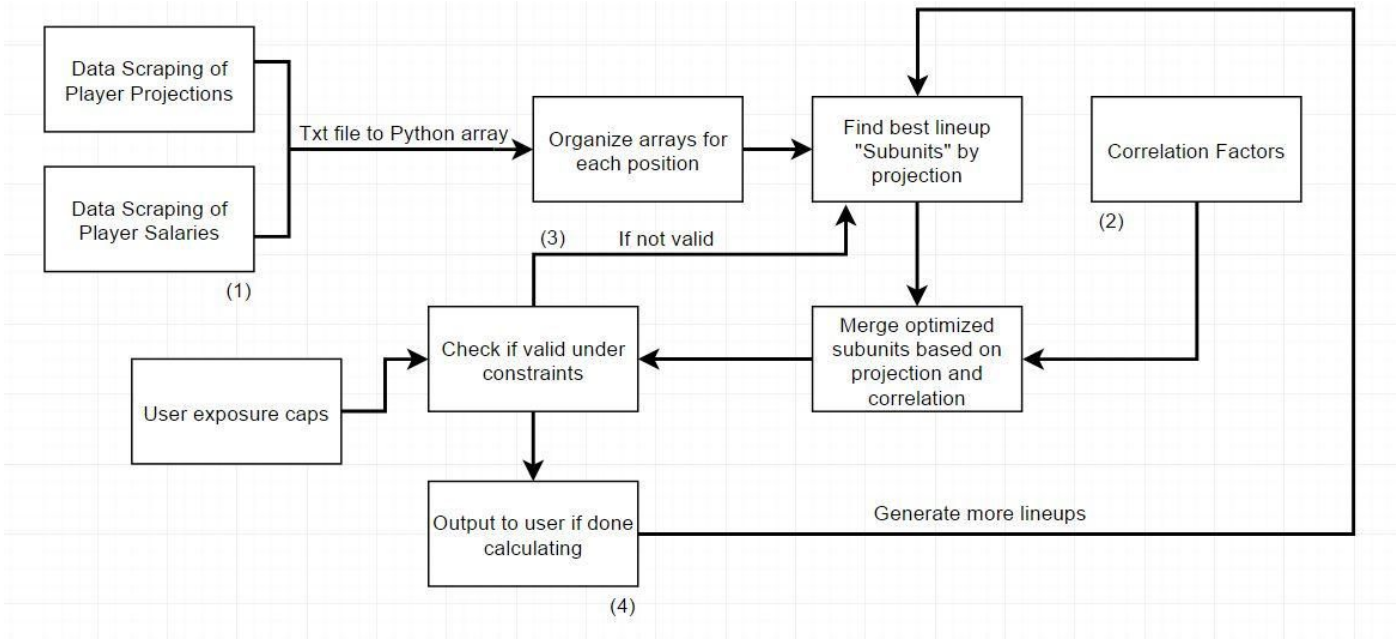
The Web Page will also get results from the optimization algorithm. This algorithm will generate lineups based on data obtained from python scripting, which occurs in the Spring backend. The algorithm generates lineups based on the settings supplied by the user, and these lineups are displayed on the angular application. This algorithm also takes in correlation data that comes from a separate data mining algorithm. This algorithm computes the correlation coefficient between pairs of players. The algorithm heavily factors in these correlations when generating lineups.

Data visualizations will also be created, based on certain factors of the algorithm's lineups. The D3JS visualizations will take lineup data, that comes from the algorithm. This lineup data will be in csv format, and the visualizations will use that csv file. The visualizations will show how the algorithm's process worked to generate lineups, giving the user insight into how their lineups were created. The correlations supplied by data mining will also be factored into the visualizations, where a lineup decision tree can show alternatives to players in a lineup.

Components:

The Algorithm (and web-application): Joe Hallal

Figure 1.0: The flow of data scraping, algorithmic lineup generation, and correlation factors

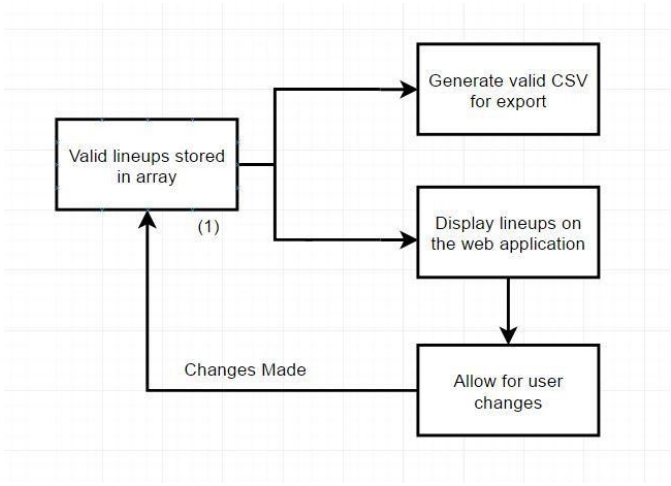


The Take the Lead algorithm is responsible for the intake of fantasy information and the successful generation of optimized lineups. It will be written in Python and is tentatively projected to not need cloud computing support. As seen by position 1 on Figure 1.0, the algorithm will receive player salary and projections from the FantasyPros website using HTML scraping. This data will be converted into a CSV. The algorithm will then divide the CSV into positional arrays. It will generate a separate array for each position and the entries will be sorted by descending salary, because this will optimize the runtime of the lineup generation. The divided arrays are passed into a combinational function that will generate all possible pairings of players in the same position. For instance, each fantasy lineup needs two running backs so the algorithm will calculate all possible combinations of 2 running backs and select the highest projected pair. It will continue this process for all the rest of the positions until each positional subset has an optimal subunit. These subunits will be stored in a separate array for each position.

After the optimized subunits have been generated, correlation factors will be incorporated into our calculations. As seen by position 2 in Figure 1.0, correlation factors are going to be weighed upon the merging of the lineup subunits into a lineup. The Quarterback will be the lynchpin to which all correlations are incorporated. After the Quarterback for the lineup is selected, we will modify the projections of the existing lineup subunits based on correlation. If the quarterback has a .2 positive correlation factor with a receiver in a lineup subunit, then that receiver will receive a corresponding boost to their projection and be more likely to be selected than he otherwise would be. After the correlation factors are applied to the rest of the player pool, the subunits will be incorporated into the lineup.

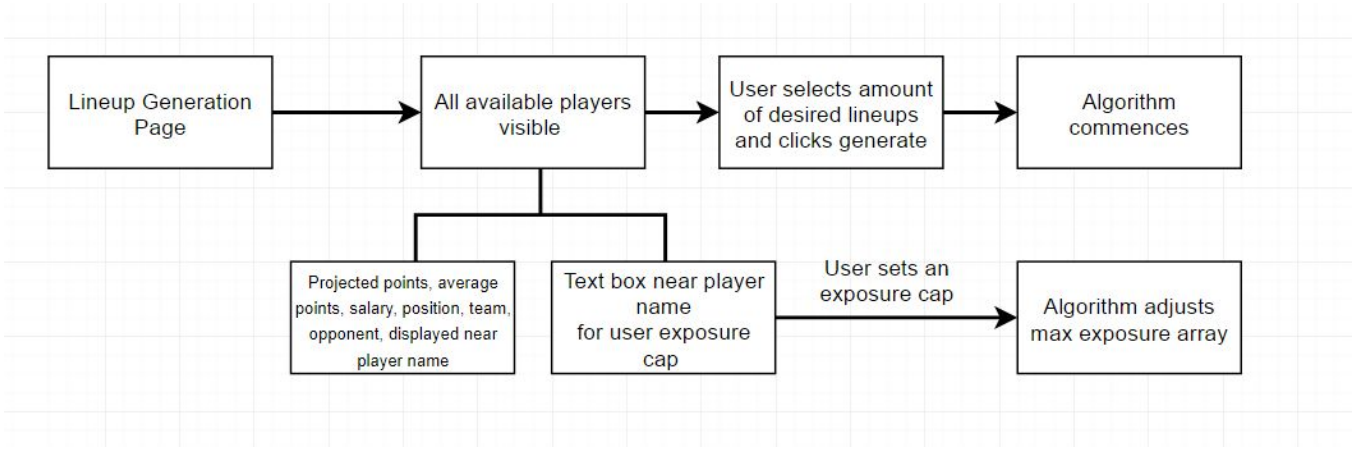
After we have built a positionally valid lineup, we will have to ensure we have met all the salary constraints. At this point if we have a valid lineup, we will exit the flow and enter it into a generated lineups array. As seen by the arrow at position 3 of Figure 1.0, if we have not met the constraints, our algorithm will recursively choose sub-units until a valid lineup is formed.

Figure 2.0 : The flow of lineups to the user



Once we have filled an array with valid lineups, we will project that information to the user. As seen by position 1 at Figure 2.0, we will have a two-dimensional array filled with valid lineup entries. We will display lineups using our Angular Web Application using a panel-based approach. Using bootstrap and D3js we will create discrete panels for each lineup and the user will be able to make changes to individual players based on their preferences. These changes will be reflected immediately in the UI and they will also be updated in the lineups array. You will be able to view the team, projected points, and salary next to the name of each player in the lineup. The lineups will appear in order of their generation. Once the user is happy with their lineups, the lineups will be changed into CSV format for export. By the click of a single button, the user will be able to receive a downloaded CSV document that can be directly uploaded to DraftKings to enter contests.

The flow of lineup generation from a user perspective



As Figure 3.0 shows, the user will be able to view all possible players in a table in the Lineup Generation Page. Each player will have relevant stats near their name that will indicate their quality of play. All players will be sortable by position and salary. There will be a text box next to each player where a user can enter a manual exposure cap. If the exposure cap is changed, the algorithm will update its "maximum-exposure array" which regulates the validity of generated lineups. Once the user is ready to create lineups, they can enter their desired number of lineups and click "generate", which begins the algorithm.

Users and use cases:

The users for the algorithmic component of Take the Lead are going to be those who are ready to generate lineups. They will have already navigated to the lineup generation page be presented with a table of prospective players they could enter. They will have this table, exposure cap entry fields, and a generate lineups button to begin their process. The following are some of the use cases the user may engage with upon reaching the lineup generation page:

Generating a single lineup: This will return the user with a single highly projected lineup.

Generating multiple lineups: This will return the user with multiple highly projected lineups. Up to 150 lineups can be created.

Generating lineups with exposure caps: If the user seeks more variation then we will allow them to enter a maximum desired exposure for individual players via our web application. For example, if the user wants no more than 40% of their lineups to contain Tom Brady, they can enter that preference in a text box in our web application and that will be reflected by the algorithm. If the pending lineup would break any desired exposure caps, then the sub-unit will be treated as invalid and the lineup will be recalculated.

Generating lineups with invalid exposure caps: If the exposure caps are constrained too strictly it may be impossible to generate a valid lineup. The user will be given an error message.

Generating lineups with an invalid player pool: Although unlikely, it is possible that the currently selected player pool is invalid. In the case when the player pool does not allow for a valid lineup construction the user is prompted with an error and no lineups are generated.

Algorithm Timeline:

December 3: Be able to generate a single valid lineup in reasonable time using matrix manipulation, python data scraping and the base algorithm. This intakes hundreds of players' data and generates a valid lineup.

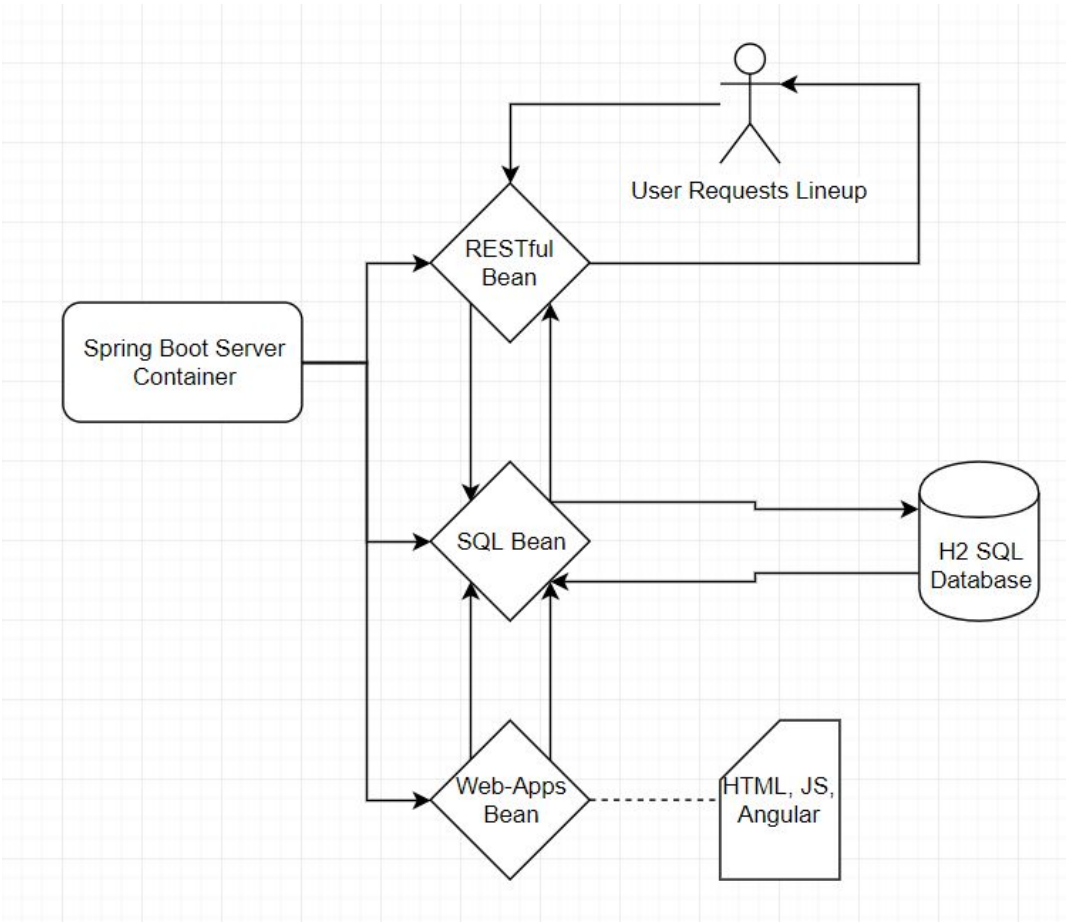
December 17: Incorporate correlation and multi-lineup generation into the algorithm. Be able to generate a variety of lineups and incorporate correlation to get unified high-upside lineups. Link the python algorithm to the front end of the application and have it run via a basic "run" button.

Jan 14: Have the complete algorithmic flow where player data is imported into the algorithm and the algorithm can generate up to 150 lineups based on correlation and begin full testing. Tests should account for invalid lineup parameters, bad data, any number of user preferences, and any type of imported player data. Develop error cases for handling these.

Feb 14: Integrate the algorithm lineup CSV export into an easy to use Angular interface where users can seamlessly interact with the UI to get valid lineups. Add any flourishes such as being able to remove unwanted players from the player pool. Ensure linkage with data visualization and all error cases fully handled.

Server and Database: Eric Wendt

The main hub that backs our application is a Spring Boot Server written in Java. The server implements basic server functions through *Beans*. Beans can be thought of as Objects that are handled entirely through the Spring Boot interface without the need for developer instantiation. Components such as HTTP calls are contained entirely within a java class that is annotated with '@Bean' tag. Another Bean contains code to handle database calls. An outline of this process is shown below:


User Requests Lineup
RESTful Bean
Spring Boot Server Container
SQL Bean
H2 SQL Database
Web-Apps Bean
HTML, JS, Angular

The largest use-case for the database comes when generating player lineups. Users will need an account to store their lineups. A number of recent generated lineups will immediately be stored in an H2 SQL database. This is to ensure that the data persists after a browser is closed. The lineup is stored in the database before being handed off to Angular to be displayed on-screen. A typical call to the server will follow as so:

1. User will log in or sign up using a POST call. The Bean for handling RESTful calls recognizes the request and sends the information to the database Bean for verification.
2. User will construct a set of lineup parameters for their generation (i.e. player exposure, ownership, number of lineups)
3. Lineup parameters are passed as JSON file to Algorithm Handler
4. Algorithm Handler calls python scripts with arguments to generate base-lineups
5. Weekly stored-player correlations are factored into overall algorithm to produce better lineup
6. Lineup is stored in H2 database
7. Lineup is passed as JSON to Angular webpage, user is given options to view visualizations and manipulate lineups

Transferring data between Python and Java

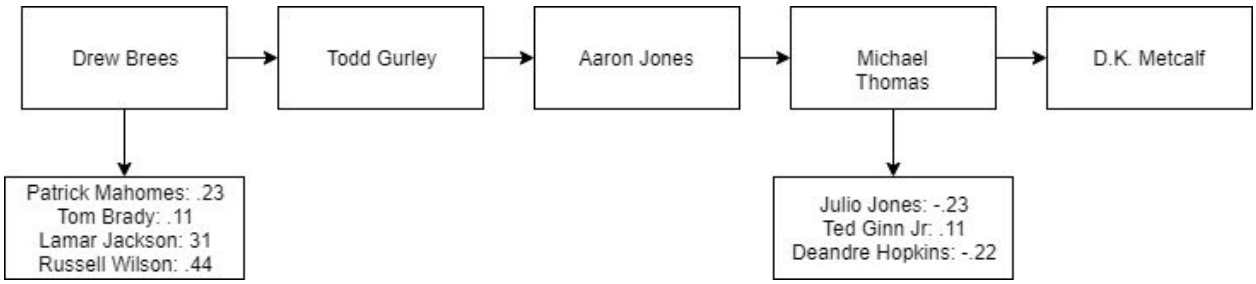
The primary method of transferring generated data between languages is through .CSV files. The files are short enough to be stored directly, and since data scraping occurs at a relatively infrequent basis, this data won't need to be stored in a database. Any data generated from making lineups *will* be stored in the sql database for speed purposes and extended querying options.

RESTful interactions

The networking of our website uses standard procedures for fetching and modifying data. Resources are specified as such: {endpoint}/resource_name/. Query parameters are specified in typical '?' notation. Parameters we plan to have are: year desired, points scored, salary, position, team, etc.

Data Visualizations and Data Mining: Mihir Mankad

Complex data visualizations will be created using a javascript library, D3JS. This library allows us to make interactive data visualizations, ones that are more interesting than simple bar graphs and charts. These visualizations will be visible with a button that is associated with each lineup. Once clicked, a small window will pop up, showing the visualizations. This all happens on the Angular application, without needing any calls to the backend. There will be two main visualizations; a lineup decision tree and a lineup profile. The lineup decision tree will showcase the players in a single horizontal line, like the diagram below. From this, a user will be shown alternative players that have a correlation to ones currently in the lineup, and they will be able to make changes that will affect the overall lineup. These correlations will either be positive or negative, where a positive means it's a good correlation, and a negative means it is a bad one. This allows a user to make changes based on things they might dislike about a certain lineup. For example, in the image below, if a player wanted Russell Wilson instead of Drew Brees, they would click Wilson's name and he would replace Drew Brees in the lineup. This change would then generate new alternatives based on the new correlations that may exist with quarterback russell wilson.



This component will communicate with the algorithm/lineup generator, as it will take in a file from that component. It will also take correlation data dependent upon the players in the lineup. It will call a function that returns the correlation values for a given lineup.

Data Mining will be used on our data set, in order to find correlations between players. We will need to examine the entire database of players in order to calculate a correlation value. This value must be backed by enough data so that it is a strong correlation, based on years of statistical data and fantasy point data. The way these correlations will be calculated, is with the formula for finding the linear correlation coefficient. The equation, displayed below, finds a value between -1 and 1, which is a correlation coefficient. The closer the value is to 1, the stronger the correlation is, and vice versa, the closer to -1, the worse the correlation is. In our case, we will sum the players fantasy point games with the other player in the pairs fantasy point games, to find correlation between the two. This component will be making calls to the SQL database in order to use player's fantasy data. Once the correlations are

calculated for a given set, they are sent back to the database for storage, where the algorithm can then use the correlations.

$$r = \frac{n\sum xy - \left(\sum x\right)\left(\sum y\right)}{\sqrt{n\left(\sum x^2\right) - \left(\sum x\right)^2}\sqrt{n\left(\sum y^2\right) - \left(\sum y\right)^2}}$$

Since these components are smaller pieces of the project, Mihir Mankad will also be contributing to the other components, such as the angular application.

Data Visualization and Data Mining Timeline:

December 3: Will be able to generate not interactive version of visualization, from a single lineup. This lineup will be hardcoded in for now, rather than coming straight from the algorithm. At this point, we will also be able to generate correlation values for any two players, from their fantasy game history.

December 10: Will have the data visualization become interactive, where you can click a players name to replace them in the visual.

January 14: Data visualization will take lineup from the algorithm, along with corresponding correlation values generated from data mining algorithm. This visual will show alternative players with their correlation values as well. The correlation values will be automatically generated for players in the lineup if database is set up.

February 10th: At this point, the data visualization on its own will be complete, and interact with the algorithm and correlation values. A user will be able to see the lineup that was selected, as well as alternative players and their correlations.