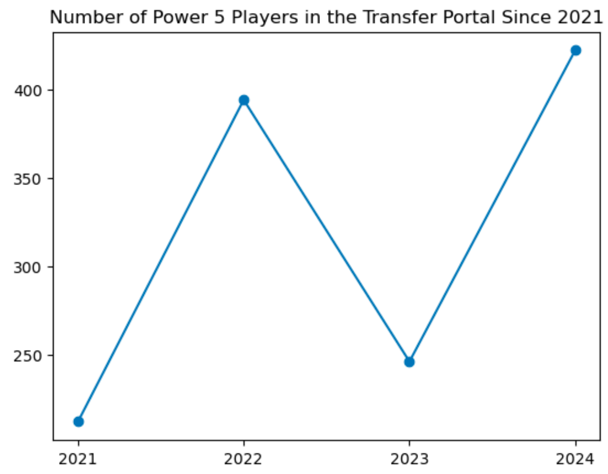
Project Statement

In recent years, the college football landscape has been dramatically impacted by the addition of the transfer portal. The transfer portal allows any college football player to transfer to a new school, with very few restrictions and, in most cases, the ability to play immediately the following season. This is a massive change, as before the implementation of the transfer portal, players had to go through a more stringent process, often having to obtain waivers to gain immediate eligibility to play at their new school. As a result of the transfer portal, schools and players are now dealing with a sort of “free agency” in college football, where every offseason teams not only have to manage the existing challenges of high school recruiting, but now must navigate the new complexities of keeping their own players from entering the portal, as well as identifying players who may enter the portal as potential additions to their program.



Since the transfer portal will continue to become an increasingly important aspect of modern college football moving forward, our team sought to build an application that utilizes available player and team statistics in conjunction with machine learning to predict the probability that a college football player will enter the transfer portal.

Methodology

Data Acquisition

We utilized CollegeFootballData.com and their API in order to obtain the necessary data about players who have and have not entered the transfer portal from Power 5 programs since 2021. We chose to limit our initial scope to Power 5 schools, which are schools that are members of the ACC, SEC, PAC12, Big Ten, or Big Twelve conferences, since much of the complete data we had access to comes from these schools. We recognize however that teams outside the Power 5 have outperformed many Power 5 schools, and certainly have the ability to compete with the top Power 5 teams. A future iteration of our project will include data from non-Power 5 schools as well.

From CollegeFootballData.com, we obtained individual player usage metrics, which quantifies on average when and how often a particular player was on the field in a given season, as well as individual player statistics such as the number of touchdowns, yards, completions, carries, receptions, sacks, etc. Additionally, we obtained team statistics for each team for a given season. Finally, we were able to obtain data on which players have entered the transfer portal since 2020.

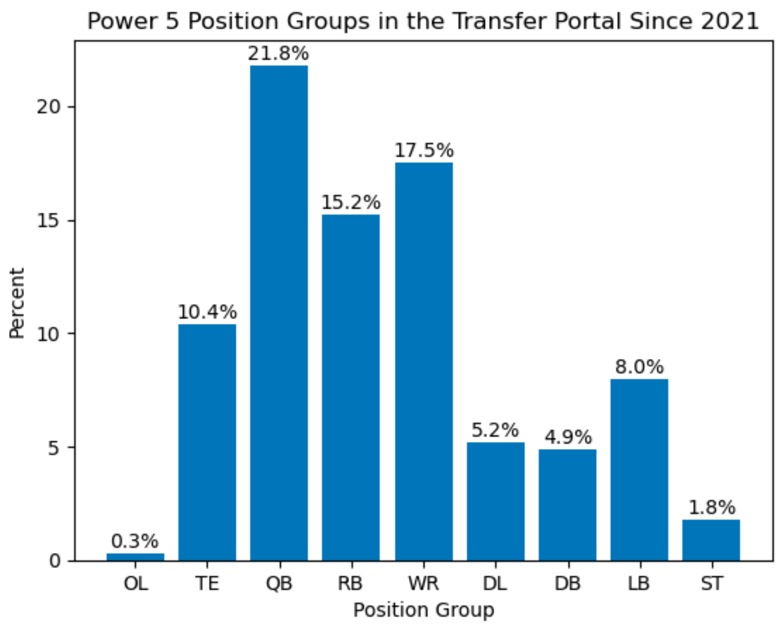
We also used a coaching dataset from Bordering on Wisdom Sports Analytics. This dataset provided information about who the coach has been for each college football team since 1987.

Data Preprocessing

After collecting the necessary player and team data, we began the process of cleaning and preprocessing our data into a more organized and usable format using NumPy and pandas. Since the various data points came in separate files, for each conference and each season from 2019 to 2023, we first processed the individual data files by standardizing column names, and adjusting certain data to group statistics by player or team if necessary, and then merging the individual files together. Finally, using the same logic we merged all the individual datasets for each team and each season together to create a final comprehensive dataset containing all relevant statistics and transfer portal information for Power 5 players from 2019-2023.

Feature Engineering

Our initial dataset consisted of a wide range of features, many of which were specific to certain position groups. For example, the statistic indicating the number of interceptions that a player had in a season would only be relevant to defensive players, while the number of receptions would only be relevant for offensive players. Additionally, our initial data analysis revealed an interesting breakdown of the percentage of each position group that has been in the transfer portal since 2020. Based on this, we decided to first separate the initial dataset into multiple datasets corresponding to the following position groups: Quarterbacks (QB), Running Backs (RB), Wide Receivers/Tight Ends (WR/TE), and Defense. Due to the relatively smaller percentage of individual defensive position groups in the transfer portal and fewer distinct statistics among the defensive position groups, we decided to group all defensive positions together. Furthermore, at this time we have chosen to exclude the Offensive Line (OL) and Special Teams (ST) position groups, as we did not have enough data points for either of those positions to generate meaningful predictions.



For each position group, we narrowed down the original set of features to specific statistics that we felt were most important. For example, for the RB group we used statistics like number of carries, total rushing yards, total rushing touchdowns, etc. since those would be most relevant to the RB position, while excluding statistics that would not be applicable to the majority of running backs such as passing yards or sacks. We followed a similar approach for each position group, keeping relevant features and excluding irrelevant ones.

Using the narrowed features, we also created a number of new, slightly more informative features that we felt would provide more insight into players who enter the transfer portal.

We created a feature that highlighted what percentage of a certain team statistic could be attributed to a particular player. For example if a team had x touchdowns in a season, and a player had y touchdowns, then we could calculate the percentage of the total team touchdowns that can be attributed to the player. This provides additional information, in addition to the player usage metrics, about how much production a player was providing for their team.

Generally, it is expected that throughout a player's college career, they will gain more playing time and improve their performance year to year. It is possible that if a player is not experiencing growth, they might be inclined to enter the portal and see if they can find the opportunity to improve elsewhere. Therefore we created a feature that described whether a player's performance for a certain statistic was improved compared to the prior season.

Players often come from high school with a recruitment star ranking of 0-5, with 5 stars indicating a highly rated player. That rating alone could be a useful indicator of how likely a player is to enter the transfer portal, however we thought it would be more informative to compare a player with players of a similar star ranking. If a player is outperforming players in their class with the same ranking then it is possible that they are less likely to leave their current school, while if a player is underperforming relative to their peers they might be more likely to enter the portal. Therefore we created a feature that compared a player's performance to other players of the same class with the same star ranking.

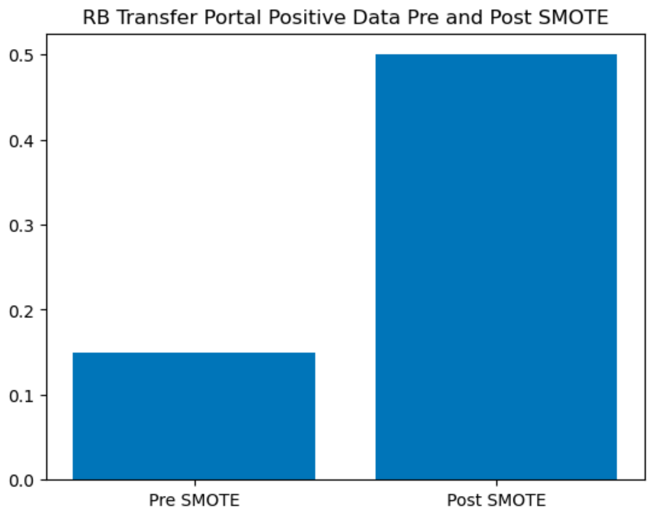
Finally, when high school players commit to play at a certain school, the head coach they will be playing under usually plays a significant role in that decision. Therefore any coaching changes might influence a player's decision to enter the transfer portal. With this in mind we utilized our coaching data set and created a feature indicating whether after a particular season, a team had a head coaching change.

Machine Learning:

Once we had our final features for each position group, our next step was to model a classifier for each group in order to predict the probability that a player from that position group will enter the transfer portal.

For each position group, we first used a ColumnTransformer pipeline which included a SimpleImputer and OneHotEncoder to fill missing values and encode the categorical features in our data.

Next, since our data was quite unbalanced, with the majority of players in our data set not entering the transfer portal, we used Synthetic Minority Oversampling Technique (SMOTE) to oversample the minority class. This is a useful strategy because when you have a data imbalance, it can result in poor model performance. In situations like ours where it is a challenge to get more data from players who have entered the transfer portal, SMOTE uses the existing data points and nearest neighbor relationships to create synthetic examples of the minority class (Malkin, C). This allows our models to be trained on balanced data, and provide better performance.



We evaluated three different classifiers for each position group: Logistic Regression, Random Forest Classifier, and Gradient Boosting Classifier. We chose to evaluate these three models since they are probabilistic models and perform well for binary classification problems. For these three models we performed extensive hyperparameter tuning to evaluate which had the highest performance for each position group.

Evaluation

There are a few factors that played into which metrics we chose to monitor when evaluating model performance. Because our dataset is heavily unbalanced, with a majority of the data points representing the negative outcome class we decided that recall was the most important metric. Using just accuracy to determine the performance of our models would be insufficient, as there would always be a reasonably high score due to how imbalanced the classes were. Because of this, we used precision, F1, and recall to guide our decision making when evaluating our models. We wanted to maximize our models' ability to correctly identify as many positive instances as possible, even if this causes an increase in false positives. While precision would be valuable to have, there were times where our models were unable to find the perfect balance we were after. Casting a broad net was more pertinent to our goals, rather than being exclusive with our positive predictions. Because of this strategy, our precision and F1 scores would in turn be decreased at times (Huilgol, P).

For the intents of our project, as long as we kept our recall and accuracy reasonably high, we felt that our models would be performing as we needed them to. However, it should also be noted that steps were taken in order to attempt to raise precision and F1 scores - such as adding additional features to certain position groups - and when possible models that possessed balance were selected. Each position group had slightly different justifications for which models were ultimately chosen, as the amount and variance in our data changed for each group.

We ran GridSearchCV to save out the top performing models, all of which were optimized for recall. We then took the precision, recall, F1 score, and accuracy of each model and compared them to each other, and to a dummy classifier that was trained and tested on the same data. All dummy classifiers were trained with the "stratified" strategy, causing it to predict with the same probability of the class distribution. Below are tables containing the results of the top models for each position group, along with the dummy classifier performance.

Defense

	F1	Precision	Recall	Accuracy
Logistic Regression	0.307	0.191	0.789	0.793
Dummy	0.107	0.060	0.504	0.511

Quarterback

	F1	Precision	Recall	Accuracy
Random Forest	0.686	0.706	0.667	0.866
Dummy	0.328	0.232	0.555	0.500

Wide Receiver / Tight End

	F1	Precision	Recall	Accuracy
Logistic Regression	0.403	0.298	0.620	0.720
Dummy	0.194	0.128	0.400	0.495

Running Back

	F1	Precision	Recall	Accuracy
Gradient Boosting	0.600	0.632	0.571	0.889
Dummy	0.227	0.145	0.524	0.479

As shown in the tables, each model was able to achieve clear improvements over the dummy classifiers. Unfortunately, the nature of personal decisions added onto the limitations of our data caused a low floor for model performance. Regardless, through feature engineering and hyperparameter tuning, each position group was able to return favorable performance in key metrics. While our defensive model has distinctly low precision, its high recall justified its use. Defensive positions had less data overall to describe the performance of players, so prioritizing the identification of all positive cases became more important. The other models were able to maintain better balance between the performance metrics as indicated in their higher F1 scores.

We were also aware that the decision to transfer universities is an extremely personal decision that most likely can't be entirely quantified. This led our focus to also include what our probabilities looked like, in addition to which outcome class each data point was assigned. We made sure that after considering the performance metrics of our models, their probabilities also were in the right places. Both of these evaluation strategies were weighed together.

We can see in the table below the average predicted probabilities for the classes passed through each model.

Average Model Probability

	Positive Players	Negative Players
Defense	0.728	0.215
Quarterback	0.825	0.072
Wide Receiver / Tight End	0.634	0.331
Running Back	0.502	0.273

The quarterback position group was the best at distinguishing the probabilities between the two classes. This is most likely due to the quarterback position not only having the biggest share of players who enter the transfer portal, but also having a good diversity in performance metrics that are critical to team success. With the increased volume and quality of data, it makes sense that this position group has the most successful models.

It should be noted that even with low precision, the defensive position group was able to maintain appropriate probabilities between the two classes, and while the remaining two models had less impressive probability averages, their models' performance indicated that they were still able to identify the two classes at an effective rate. The increased uncertainty reflected in the probabilities is most likely due to the complicated nature of the positions' data.

Broader Impacts

Since this project deals directly with predicting what decisions a player will make, it has the potential to deeply impact the player, the players family, coaching staffs, universities, and team fan bases.

Due to the nature of this project and the application of predicting whether or not a human will take an action, this project is subject to automation bias. Automation bias occurs when humans rely on an AI system to make a decision even when the system may be incorrect (Abbasi, J and Hswen, Y). For instance, if this tool was used at the end of every college football season for every player on a team and the decision makers at the institution used the predictions to make decisions on players.

This could have a negative impact on both players and the institution. If the tool predicts a high probability of a player transferring while the player has no intention of doing so, this prediction could negatively affect the relationship between the player and school. In addition, it could have the potential to negatively impact the earning potential of that student athlete, as NIL collectives would not want to invest additional money into a player who was perceived to be leaving the school.

From a university perspective, this tool could hurt the recruitment process. If the tool is used to automate which position groups the coaching staff should focus on recruiting due to predicted position transfer. This could lead to recruitment efforts for players positions which may not actually be needed.

With this being said, the tool is not recommended to be used in any form of decision making.

Web Application Overview

Backend

Flask was chosen as the application framework to host the backend web application logic. To host the code, Heroku was chosen as the hosting platform due to its ability to auto-deploy changes pushed to GitHub branches.

Trained models were saved as *joblib* files and loaded by the Flask application at build time. Doing this prevented the need to train the models during every deployment. The models are able to be used via two endpoints on the API.

Frontend

NextJs was chosen as the application framework to host the frontend web application logic. This was done due to the team's prior experience with the framework.To host the code Vercel was chosen as the hosting platform due to its integration with NextJs and ability to auto-deploy changes pushed to GitHub branches.

Generating Predictions

The frontend web application exposed the models created in two forms, a player form and a manual submission form.

The player form allows a user to select or search for a player who played during the 2023 season and generate a prediction on the probability of the player transferring schools. Each option in the player form is mapped to a unique player id. When the form is submitted, the frontend application sends an HTTP POST request to the backend server with the player id in JSON format. Once the backend server receives the request, it extracts the player id from the request. The server then matches the player id to a row in a pandas DataFrame which only contains information from the 2023 season. After a series of data manipulation steps, the player information routed to one of the models based on the position of the player. In the code blocks specific to position, there is additional data manipulation specific for the position specific classifier. The player information is passed as input to a model to generate a prediction using the *predict_proba* method. The backend server responds to the frontend with a list of dictionaries, contained in each dictionary is the name of the model and the probability of the prediction.

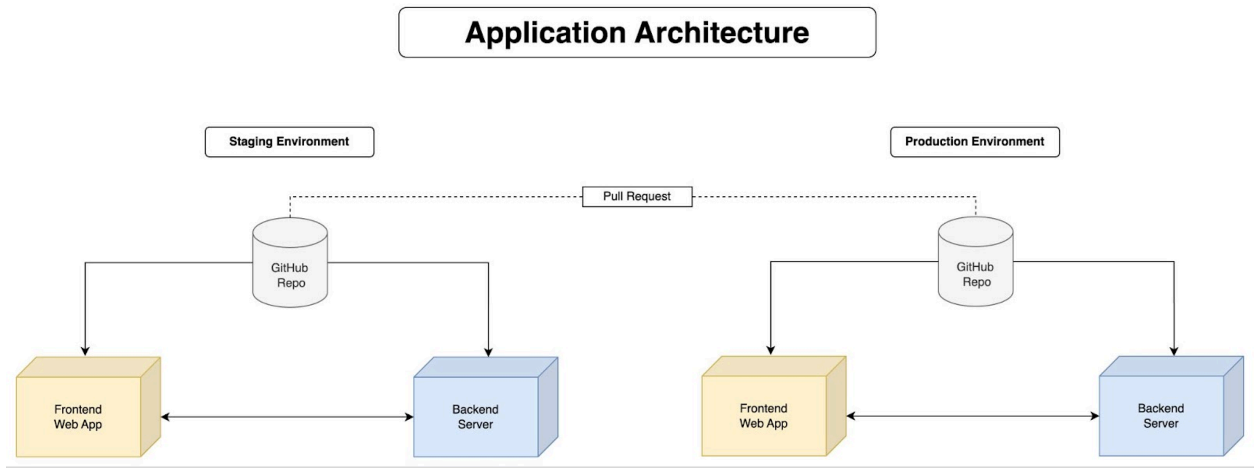
The manual submission form allows a user to manually input all the data which the model will use to generate a prediction. The request/response process works just as the player form does except that there is no matching required with an existing player.

Version Control

To collaborate, the team used a public GitHub repository.

Architecture

The application architecture consists of both a staging and production environment. The staging environment allows the team to test changes and fix bugs, before releasing the changes to the general public in production. Utilizing this architecture allowed for the team to use an iterative software development lifecycle, in which changes large and small can be delivered incrementally.



Statement of Work:

Jeffery Jones:

- Data Cleaning/Merging
- Defensive Position Groups Feature Engineering
- Defensive Position Groups Machine Learning and Evaluation

Yash Dave:

- Offensive Position Groups Feature Engineering
- Offensive Position Groups Machine Learning and Evaluation
- Data Acquisition

Raul Martinez:

- Frontend Web Application
- Backend Web Application
- Automation Pipelines and Server Hosting

Citations

Abbasi J, Hswen Y. Blind Spots, Shortcuts, and Automation Bias—Researchers Are Aiming to Improve AI Clinical Models. JAMA. 2024;331(11):903–906. doi:10.1001/jama.2023.28262

Huilgol, Purva. “Precision and Recall: Essential Metrics for Machine Learning (2024 Update).” Analytics Vidhya, 12 Apr. 2024, www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning/.

Maklin, Cory. “Synthetic Minority Over-Sampling Technique (Smote).” *Medium*, Medium, 14 May 2022, medium.com/@corymaklin/synthetic-minority-over-sampling-technique-smote-7d419696b88c.