Understanding Lacrosse Game Dynamics through Data

SIADS 696 Milestone II partial project results

Tony Kibling (tkibling)

Introduction

My aim in this part of the project was to apply data science techniques to better understand the game dynamics in the sport of Lacrosse and to address a large gap in the fan experience when it comes to analytical understanding. By developing these techniques, my hope is that information providers or app developers can create a richer fan experience for in-game information or post-game analysis.

Initial Focus & Related Work

Player Positions: Developing a method to enrich lacrosse data with the positions played by a given player in a given game. Currently, only the players in the starting line up receive a player position during scorekeeping, leaving the majority of players who see the field without one. This builds on techniques like that of Charlie Jackson for predicting football(soccer) player positions to predict a player's position based on their statistics for that game. Charlie only predicted two positions, whereas my model does four. To take this a step further, the same statistics can be used to cluster players based on the respective feature labels.

Approach & Primary Findings

Player Positions: For the supervised player prediction, four models were considered for evaluation (k-Nearest Neighbors, Random Forest, Naive Bayes-Gaussian, and Decision Tree). All four had a prediction accuracy greater than 70%, with the kNN and Random Forest models testing within 1.5% of each other. Ultimately, kNN with uniform weights fitted the models with the highest accuracy. While many statistical fields were considered and after a few rounds of ablation, goalie seconds, shots, faceoffs taken, and points were the features found to be the most important. For the unsupervised position clustering analysis, two models were developed (kMeans and agglomerative-Ward's). The models were evaluated for each of the nine seasons represented in the dataset. Using a dendrogram model for agglomerative clustering gave a high level view of positions, but would not break down the individual roles into their own clusters that some of the positions have, which is what I was looking for. Using PCA dimensionality reduction with kMeans clustering combined with a biplot shows which features are considered for each cluster and which features are related for each principal component. For added visibility, a Sparse PCA heat map was found to best illustrate feature importance for each season.

Dataset

The dataset used was compiled by Matt Hudson (one of the team members for the rest of this project) and is hosted on Amazon RDS in a relational dataset. The dataset is modeled as a star schema of information on NCAA Men's Division 1, 2, and 3 lacrosse from 2015 through 2023. This includes data on the Programs (e.g. Schools), Seasons, Games, Box Scores (results of a game), and Play-by-Plays (events for each game).

There are roughly 3,000 games per season across all divisions. Each game has about 500 rows of play-by-play and box score data by player. This equates to nearly 12 million rows of data in total. The data was already pre-cleaned – but required modeling for the purposes described above.

All of this data was made accessible through any Postgres client, but our team primarily used a python class built on top of `psycopg2` for easy reproducibility.

Determining Player Positions

The journey through the lacrosse data brought a will to want to learn more about the players and what positions they play. Gaining a basic understanding of player information can lead to many more studies and insights. Finding as many roles on the field as there are was not expected and it is exciting to share the findings.

Feature Engineering for Supervised Player Prediction

Since the data sources are housed in a postgresql database, it was beneficial to create local dataframes rather than connecting to the database to retrieve information from the tables stored there each time a notebook was started. Two main dataframes were created. One called box_score_df that will be used with the supervised player position prediction and the other called cluster_df that will be used with unsupervised position clustering.

The box_score_df was created by joining the box_scores and team tables from the postgresql database. Although the team information is not a feature used in the supervised model, it is informative to see which team each student plays for. While looking over the data, only one column had to be addressed. “goalie_minutes” was renamed to “goalie_seconds” due to the data being represented as values from 0 to 3600. Otherwise, the other columns look fine. See Appendix A: Table 3 for table lookup references and Appendix B: Table 4 for the list of features.

In lacrosse, there are four main positions: attacker (A), midfielder (M), defense (D), and goalie (G)[1]. Some of the main positions have specialty roles as well. For example, attackers can play the wings and “X” position and midfielders can consist of face off specialists, short stick defensive midfielder, and long stick midfielder. The “position” column contains all of these labels and more. The scorecard provided at the beginning of the game shows the starters' positions. An official scorekeeper is responsible for properly entering the position for a player for that game. However, these are manual entries and they are prone to mistakes, but not many. There are only 564 cases where the “position” field contains an incorrect acronym (CD, SR, CK), short name (MIC, FSO), or number (D08, 27, 31). Nevertheless, these mistakes were cleaned up and any label that did not make sense was changed to a blank field. All other labels that could be correlated to one of the four main categories were changed accordingly. There were 1,113 positions changed to A; 9,816 positions changed to M; 974 positions changed to D; and 19,602 positions changed to G. A table containing the most popular examples as well as the full table of changes can be found in Appendix C: Tables 5 & 6.

For the final tally, there are 1,180,290 rows of data in which 248,616 have valid positions entered that will be used for training and testing.

Player Position Prediction Methods

The intent of this supervised learning project is to predict whether a player is one of the four main positions based on their statistics for any given game. Four supervised learning techniques were used to build and test multiple machine learning models. Various ablations of features, adjusting some hyperparameters, and differing train/test splitting sizes were used. A table of the end results is shown to the right. I was not just looking for the best score on the test set, but one where the model did not overfit on the training set nor underfit with the test set.

Table 1: Various ML model results

Model	5 Fold Mean Cross Validation Score
KNN, weights = uniform	73.46 +/- 0.58
KNN, weights = distance	72.49 +/- 0.37
Random Forest	72.13 +/- 0.38
Naive Bayes - Gaussian	71.65 +/- 1.62
Decision Tree	70.88 +/- 0.47

The first model tried was K-Nearest Neighbors (KNN). This is the same model used in this supporting article[2] and will be the baseline against the three other models in this analysis. The first step to complete was to find the best nearest neighbors (n_neighbors) value to use. A small number range, 1-19, was started with. The best accuracy score in this range was okay (73.22%) but knew it could be better. Since there are a couple hundred thousand data points, a thought

[1] https://laxfarmer.com/guides/lacrosse-positions/
[2] https://charlieojackson.co.uk/python/predicting-football-positions.php

was that the best number of neighbors may be large. The range was changed to start at 10 and go to 960 by 50 and the accuracy peaked at n_neighbors=110 (73.92%). I wanted to be sure to capture the best value and expanded the next search from 60 to 210 by 10 where the top n_neighbors=180 with 73.96% accuracy. These results narrowed down the final run for each value from 170 to 190. n_neighbors=180 ended up having the best score. This analysis took several hours to complete, and the difference in accuracy between 180 and 18 is less than one percent. Whether or not that difference is worth it depends on what you are working on. A table of the results for each iteration, ranges, and scores can be found in Appendix D: Table 7.

KNN has a hyperparameter called weights. By default, this is set to 'uniform', which means that all points in each neighborhood are weighted equally. The other option is 'distance' which weighs points by the inverse of their distance. The default value was the best and the results of each are shown in Table 1 above. To continue with the sensitivity analyses, train and test splits from 60/40 to 95/5 by five were completed and accuracy scores were compared at each iteration. Five fold cross validation scores were also computed to confirm the accuracy scores. An 80/20 train/test split provided the best results.

Decision Tree was the second model that was considered for this analysis. This model was selected because it is easy to interpret and visualize the results. A decision tree learns a hierarchy of if/else questions that eventually lead to a decision. There are a few hyperparameters, but after some test runs, the default hyperparameters provided the best results. The same train and test set splits were completed, 60/40 to 95/5 by five. An 80/20 train/test split provided the best results for this model as well. One downside to this model is a propensity to overfit; and that was the case here as well. Also, this model had the lowest test scores of the four as shown in Table 1 above.

The third model considered for this analysis was Naive Bayes. Naive Bayes has three types of classifiers: Bernoulli, Multinomial, and Gaussian. Gaussian was the best fit for this exercise because of the real valued features. The Naive Bayes - Gaussian model was chosen for its simplicity and to use a model that assumes no correlation between the features to see the output. As it turns out, there is correlation between the features and that was seen with the results. Although the training and test scores were close to one another, the model produced the lowest accuracy scores on the training set and second lowest on the test set.

The fourth and final model considered for this analysis was the Random Forest model. This model was chosen mainly to compare against the Decision Tree model and because it was easy to visualize feature importances. The Random Forest model not only outperformed the Decision Tree model (a forest is better than a tree), it also outperformed the Naive Bayes model. However, it was still not as good as KNN, as shown in Table 1, and hence not used for the detailed analysis[3].

Feature Importance and Ablation Results

There are 15 features that are part of this model and it is unlikely that all of them are important. One tradeoff of sticking with KNN is that it does not have a feature importance attribute to complete an "easy" ablation analysis. The sklearn library has a function called 'permutation_importance' that measures how important each feature is to the model. This was very time consuming to complete. It took almost two minutes to complete 1,000 rows. Running the entire dataset was not fruitful. Therefore, a sample of 10,000 rows was taken from the overall dataset and the number of rows for each position was based on the ratios of each position. The varying degrees of ablation helped identify the importance of the features, as shown in Table 2.

Table 2: Feature ablation for KNN model, weights=uniform

Category	Features Removed	5 Fold Mean Accuracy
all	none	73.46 +/- 0.58
important	goalie seconds, shots, faceoffs taken, points	71.86 +/- 0.75
important-ish	faceoffs taken, points goals, assists, ground balls, turnovers, shots on goal	72.96 +/- 0.64
Not that important	penalties, goalie saves, goals allowed, penalty time, caused turnovers, faceoffs won	73.19 +/- 0.58

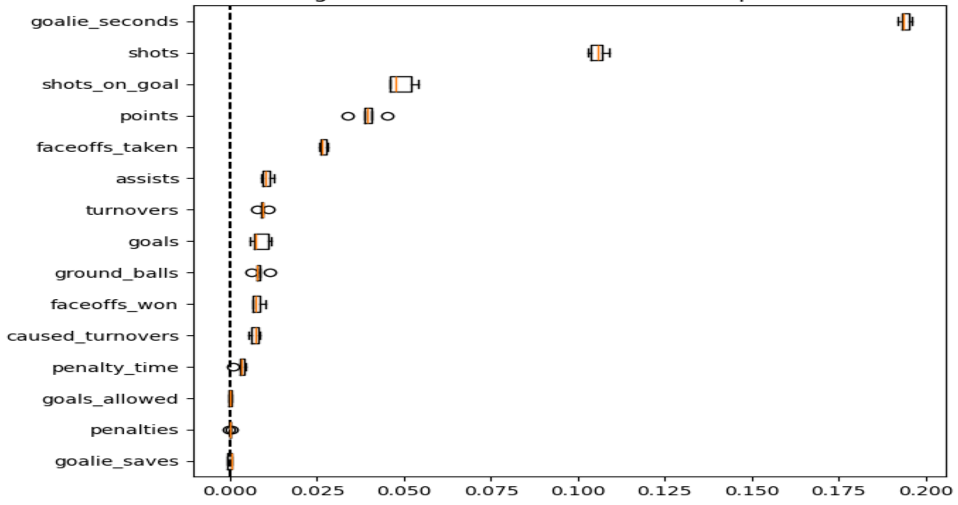
Overall, there are four features that, when removed from the model, decrease accuracy by 1.6%. These four make sense to be impactful as they are key factors in each of the statistical groupings. There are five other

[3] Appendix E has tables 11-15, which contain all of the train/test split iterations plus many of the ablations for all models.

features that have a smaller impact on accuracy, 0.5%. They are goals and assists, ground balls, turnovers, and shots on goal. Goals plus assists equals points and points is in the important category. A player can not have any shots on goal without any shots and the shots feature is also in the important category. Lastly, there are features that do not have high importance. They are penalties, penalty time, goalie saves, goals allowed, caused turnovers, and faceoffs won.

To visualize the feature importance comparisons between two models. Figure 1(below) shows the kNN permutation importance function. Additionally, a Decision Tree feature importance attribute histogram can be found in Appendix G: Figure 15. There are many similarities between the two outputs. Goalies seconds and shots are the top two features for each model.

Figure 1: kNN Feature Permutation Importance

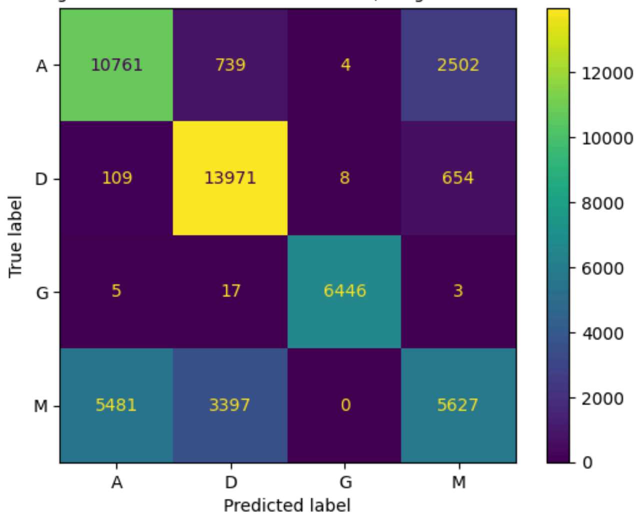


Failure Analysis

One way to visualize the quality of a prediction is with a confusion matrix. A confusion matrix is a table used to define performance of a classification algorithm. It shows true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).

An initial reaction to the results from the k-NN model is that they are not so bad. Goalies predict correctly almost all of the time at 99.6% (6446 out of 6471). That is great! Defensive players are not far behind with 94.8% predicting correctly (13971 out of 14742). Also great! However, much further behind is the prediction on Attackers at 76.8% (10761 out of 14006). This is still acceptable. Falling way behind is the ability to correctly predict Midfielders at a measly clip of 38.8% (5627 out of 14505). This is not surprising to me though. Midfielders play offense and defense, therefore, a midfielder could have stats comparable to either of these categories.

Figure 2: Confusion Matrix- kNN model, weights=uniform



Below are details about four examples where the prediction was incorrect.

- Index = 6
    - Predicted as an A but is listed as an M.

    - For this game, he had 5 goals, 3 assists, 8 points, 14 shots, 10 shots on goal, 4 ground balls, 3 turnovers.
    - These seem like stats similar to an A.
- Index = 30
    - Predicted as a D but is listed as an M.
    - For this game, he had 1 ground ball, 1 faceoff won, and 1 faceoff taken.
    - This player would likely be an M due to taking a faceoff.
    - A specialized position, known as a FOGO (face off get off), is categorized as an M and they do the faceoffs if a team has that position
- Index = 21575
    - Predicted as a G but is listed as a D.
    - For this game, he had 2 ground balls, 13 goals allowed, 16 goalies saves, and logged the entire game (3600 seconds) at G.
    - This player should predict as a G too based on goalie seconds alone.
- Index = 49637 is a fun one.
    - This player is predicted as a G but is listed as an A.
    - For this game he had 1 goal, 1 point, 3 shots, 2 shots on goal, and 3 ground balls. This person also had 7 goals allowed, 5 goalie saves, and logged half the game (1800 seconds) at G.
    - This player clearly switched roles during the game, but that is not logged in the data because it is one row per player, per game.

An improvement to address a player logging time at goalie and another position during the same game would be to tag that player with more than one position label accordingly. Even though this event likely occurs a small number of times, the model would need to be trained to this new method as well in order to account for players that have more than one position in a game. A way to potentially overcome the M inaccuracies is to keep the specialized positions in the dataset and train the model with them.

Feature Engineering for Unsupervised Player Clustering

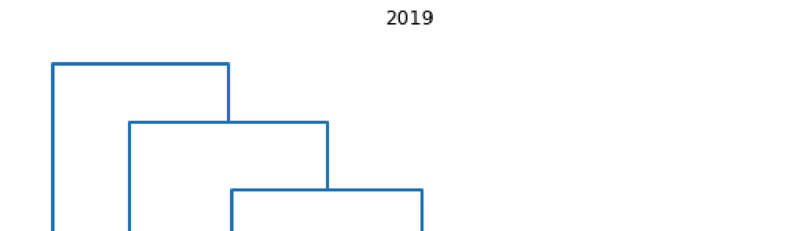
As previously mentioned, the other dataframe used for the player position analysis is called cluster_df. This dataframe was also created by joining the box_scores and team tables from the postgresql database. Two more features were brought into the dataframe, each from different tables. The season id was added from the games table and the year of that season from the seasons table. The season information will be used to explore ideas including a view of the data across seasons. Just like the other dataframe, “goalie_minutes” was renamed to “goalie_seconds” due to the data being represented from 0-3600. Additionally, “id” was changed to “gameid” to avoid duplicating the same field name when merging tables. Otherwise, the other column names look fine. The complete list of features is in Appendix B. To get the data ready for the models, all features were normalized using the standard scaler.

Player Clustering Methods

The intent of this unsupervised learning project is to identify similar groups of players and their roles on a team that may not be evident by their position using their game statistics. Two unsupervised learning techniques, one hierarchical and one partitional, were used to build machine learning models for clustering.

The first clustering technique used was creating a dendrogram using Ward’s model. Ward’s model works on most datasets and has the least variance. It is a type of agglomerative clustering. Agglomerative clustering is an algorithm that starts by declaring each point as its own cluster and then merges the two most similar clusters until a stopping criterion is satisfied. An effect of the algorithm is a hierarchy that reflects the order and cluster distance when assigning data points to clusters. Unfortunately, the dataset is too large to complete a dendrogram for all of the years together. Instead an analysis was completed year by year. This is an example from the 2019 season. All other years are shown in Appendix J: Figures 19 through 22.

Figure 3: Dendrogram using Ward's Aggolomerative Clustering method



As you can see, the dendrogram split the data into four clusters, much like the groups of positions (A, M, D, G) used in the previous supervised model. As I dove deeper into the results this clustering is mostly confirmed. The ‘ivl’ and ‘leave_color_list’ attributes provide insight into which cluster each player is assigned and were added to the original dataframe used to create X and y. The findings of the results in Figure 3 are: C1 represents G as all of the players have goalie stats; C2 (green) represents A and M players with offensive stats such as points and shots; C3 (red) represents mainly midfielders, specifically this cluster includes face off specialists; and C4 contains the rest and is comprised of all positions. This model gives a high level view of positions, but will not break down the individual roles into their own clusters that some of the positions have, which is what I was looking for. This could be helpful for a semi-supervised model in a future analysis though.

The second clustering technique used was K-Means. K-Means is partitional and one of the simplest and most commonly used. It tries to find cluster centers that are representative of certain regions of the data. K-Means starts by choosing ‘k’ random points as centers. It then assigns all other points to the nearest center point. The result is an updated center point. This process is repeated until the center point is unchanged. A drawback to this method is that you must specify the number of clusters (n_clusters) ahead of time. There are several ways to determine the best ‘k’ to use. Here are two of the ways that were used in this project. Both ways involve analyzing output from a range of k’s up to the quantity of features in this dataset. A table with the results of the best k’s by year are in Appendix F: Table 13.

Player Clustering Sensitivity Analysis

The first way is to look at the inertia attribute of the model. Inertia values typically start high for small values and decrease as k grows. The goal of using this is to look for the ‘elbow’ in the graph. Generally, the line starts to level off after the elbow and that is the number of clusters to use. Unfortunately, that may not always be clear when looking at the graph. K-Means has a hyperparameter called init that defaults to k-means++. “K-means++ selects initial cluster centroids using sampling based on an empirical probability distribution of the points’ contribution to the overall inertia. This technique speeds up convergence.” However, the results were preferred where init=random where observations are taken at random. The second way is to use a combination of the model’s Davies Bouldin and Calinski Harabasz scores. The idea is to match the number of clusters where the Davies Bouldin score is the lowest and Calinski Harabasz score is the highest. The graphs below illustrate inertia (Figure 4 left) and the model scores (Figure 5 right) for 2019 using init=random. Graphs comparing the hyperparameters of init=K-means++ and random for 2019 are in Appendix H: Figures 16 and 17 plus Figures 4 and 5 below.

Figure 4: Kmeans Inertia for 2019

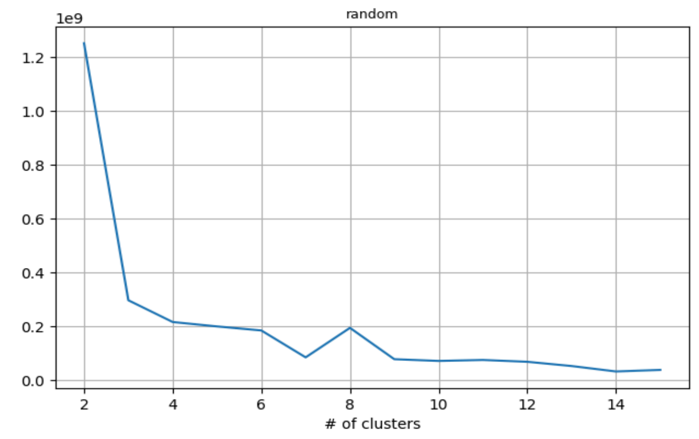
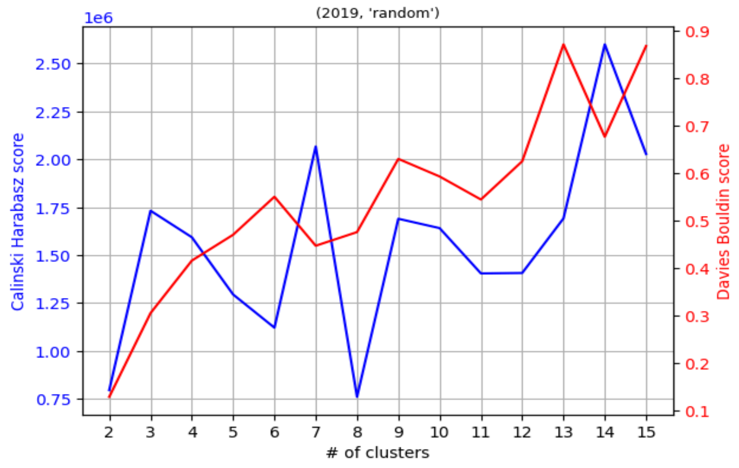
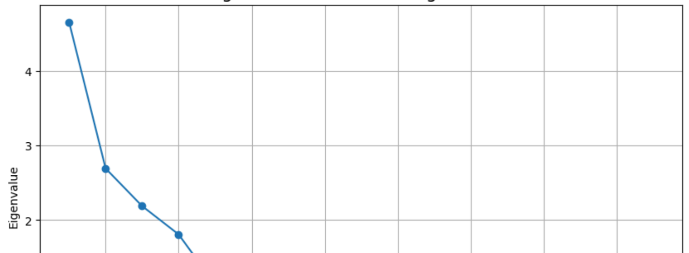


Figure 5: Where is the highest CH and the lowest BD?



Similar to the inertia graph, using init= k-means++ did not provide a clear indication of the number of clusters to use. On the other hand (or graph), init= random depicts the lowest Davies Bouldin score with the highest Calinski Harabasz score at seven clusters. See Figure 5.

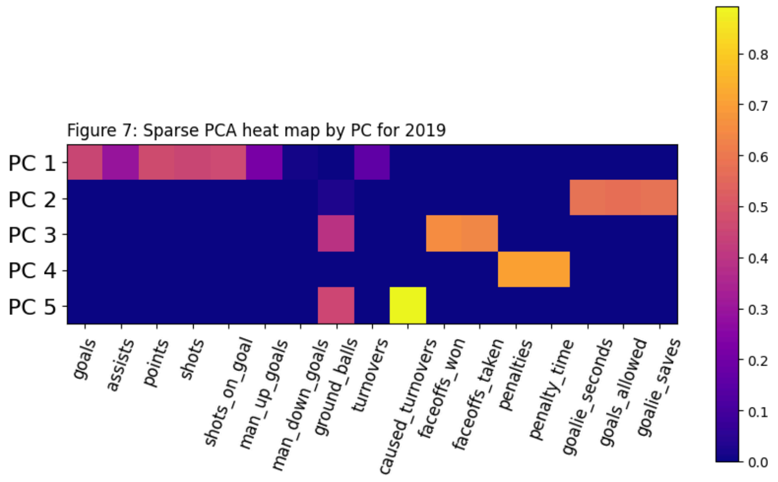
Figure 6: Scree Plot for Eigenvalues



Now that the number of clusters has been identified, it is time to create the model. Before plotting the results, the data needs to go through a principal component analysis for dimensionality reduction. Similar to the initial number of clusters, the ideal quantity of principal components to use is also an unknown that needs to be determined. The value was found using scree plots. A scree plot shows the explained variance per newly defined principal component and is measured by the percentage (Appendix I: Figure 18) or absolute value (Figure 6) of the eigenvalues. Usually the first few principal components explain most of the variance. You can use the elbow method here as well, but the most accurate way is using Kaiser's rule (keep PCs with an eigenvalue > 1). This was an easy way to determine the ideal principal components to use and can be automated for future iterations.

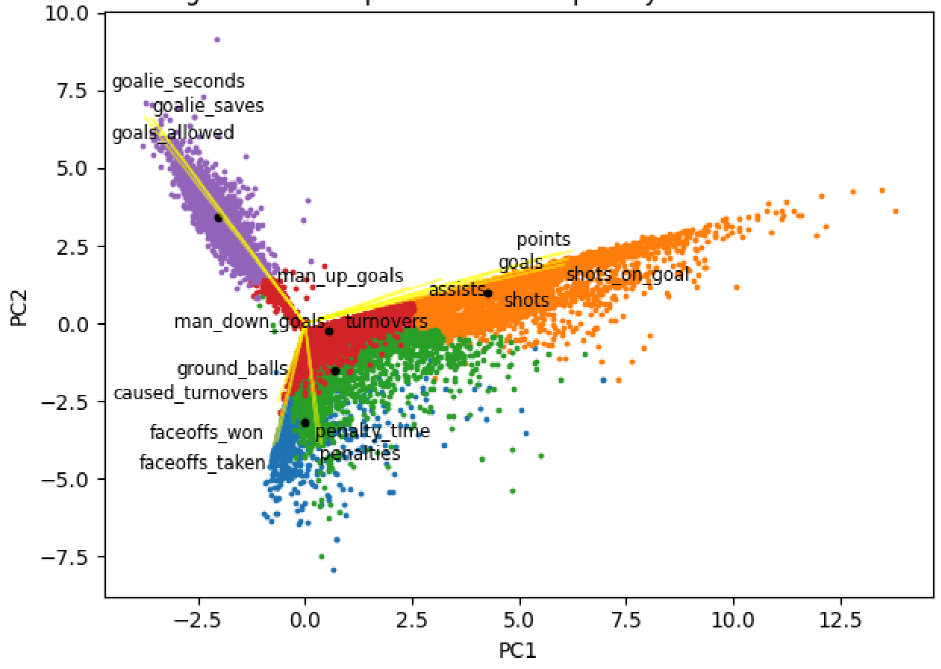
The heat map to the right (Figure 7) will help confirm that my principal component choice was correct. Showing the sparse PCA heat map is preferred as it clearly shows which features are important for each principal component. The groupings make sense too. PC1 has all of the offensive features while PC2 contains all of the goalie features. Heat maps for other years are in Appendix L: Figures 26 and 27. They mostly look the same.

Figure 7: Sparse PCA heat map by PC for 2019



Okay, I have determined the best number of clusters and confirmed the ideal principal components, let's see how all of these things come together. The visualization below (Figure 8) depicts a biplot of the five principal components labeled with the features overlaid on a scatter plot of the five position clusters. It looks pretty good!

Figure 8: PCA Biplot with Scatterplot by Cluster for 2019



Learnings and Discussion for Determining Player Positions

One of the lessons learned from the supervised player prediction is that for the KNN model, a low number of neighbors was almost as accurate as the best number of neighbors. Waiting hours for these results to see only a small improvement in accuracy was probably the only challenge.

For the unsupervised position clustering, the number of clusters for each year was expected to be the same and a higher number given all of the specialty roles. However, the k value fluctuated from five to seven clusters. The model grouped the roles together because their stats were similar.

A fun way to extend the solution would be to create a page where a user selects a player and the output would be the positions he actually played (only listed if a starter), the position predicted (would show regardless if a starter), and stats across all games played. It would be similar to searching for players on www.baseball-reference.com but with a lacrosse feel and position prediction as well.

Ethical considerations

The boxscore table that is used for both the supervised player prediction and unsupervised position clustering contains the players' names and their positions played if they started. Also, the team they play for is added to the dataframe for reference. None of that information is included in the outputs but I need to be mindful of developing reports with their names even though the statistics for each game is public data.

Another issue that arises when assigning analytics to player performance are intangible traits that may not be picked up with our methods. If decisions are made based solely on the quantitative, players who provide value on a subjective and/or immeasurable level could be disadvantaged. A final ethical consideration is the use of these models for gambling. A gambler may see a predicted win probability of 90% and assume a bet on that team is a statistical inevitability. Sports, however, are unpredictable and it is best to keep that in mind when using certain model outputs.

References

Scikit-learn 1.4.1. Scikit-learn.org

Matplotlib 3.8.3. matlotlib.org

Collins-Thompson, K. (2022). SIADS 542 Supervised Learning. Coursera. https://www.coursera.org/learn/siads542/home/

Collins-Thompson, K. (2022). SIADS 543 Unsupervised Learning. Coursera. https://www.coursera.org/learn/siads543/home/

Brownlee, J. (2020, August 20). How to Calculate Feature Importance With Python. Machine Learning Mastery. https://machinelearningmastery.com/calculate-feature-importance-with-python/

Villasante, P., Kebabci, C. Scree Plot for PCA Explained. Statistics Globe. https://statisticsglobe.com/scree-plot-pca

Jackson, C. (2020, October 25). Predicting football player positions using k-Nearest Neighbors. https://charlieojackson.co.uk/python/predicting-football-positions.php

Farmer, L. (2023, April 14). Lacrosse Positions (Explained) | Attack, Midfield, Defense, Goalie.

Appendix

Appendix A: Reference Table

Table 6: Postgresql File References

Leagues	Seasons	Programs	Teams	Rosters	Players	Games	Box Scores	Play by plays
id division	league_id							
	id		season_id			season_id		
		id						
		name	name					
				id			player_id	player_id
			id	team_id		home_team_id away_team_id	team_id	team_id
				player_id	id			
				name	name		player_name	
						id	game_id	game_id
							id	
								id

Appendix B: Features

Table 7: Features used for supervised and unsupervised models

Model(s)	Feature Name	Data type
Prediction & Clustering	position	object
Prediction & Clustering	goals	integer
Prediction & Clustering	assists	integer
Prediction & Clustering	points	integer
Prediction & Clustering	shots	integer
Prediction & Clustering	shots_on_goal	integer
Clustering	man_up_goals	integer
Clustering	man_down_goals	integer
Prediction & Clustering	ground_balls	integer
Prediction & Clustering	turnovers	integer
Prediction & Clustering	caused_turnovers	integer
Prediction & Clustering	faceoffs_won	integer
Prediction & Clustering	faceoffs_taken	integer
Prediction & Clustering	penalties	integer
Prediction & Clustering	penalty_time	integer
Prediction & Clustering	goalie_seconds	integer
Prediction & Clustering	goals_allowed	integer
Prediction & Clustering	goalie_saves	integer
Win Probability	cumulative_goals_allowed	Integer
Win Probability	home_team_flag	integer
Win Probability	minutes_left_in_game	float
Win Probability	prior_three_games_goals_scored_average	float
Win Probability	cumulative_goals_scored	integer
Win Probability	prior_three_games_goals_allowed_average	float
Win Probability	opposition_prior_three_games_goals_scored_average	float
Win Probability	opposition_prior_three_games_goals_allowed_average	float

Win Probability	win_streak	integer
Win Probability	loss_streak	integer
Win Probability	score_differential	integer
Win Probability	minutes_score_ratio	float
Win Probability	minutes_elapsed	float
Predicting Faceoff Outcomes	player_1_total_faceoffs	integer
Predicting Faceoff Outcomes	player_2_total_faceoffs	integer
Predicting Faceoff Outcomes	total_win_difference	integer
Predicting Faceoff Outcomes	win_percentage_difference	float
Predicting Faceoff Outcomes	elo_prediction	float
Predicting Faceoff Outcomes	trueskill_prediction	float
Predicting Faceoff Outcomes	pagerank_rating_difference	float
Predicting Faceoff Outcomes	time_remaining_in_game	integer
Predicting Faceoff Outcomes	time_remaining_in_quarter	integer
Predicting Faceoff Outcomes	score_difference	integer

Appendix C: Changes

Table 8: Most popular player position label changes

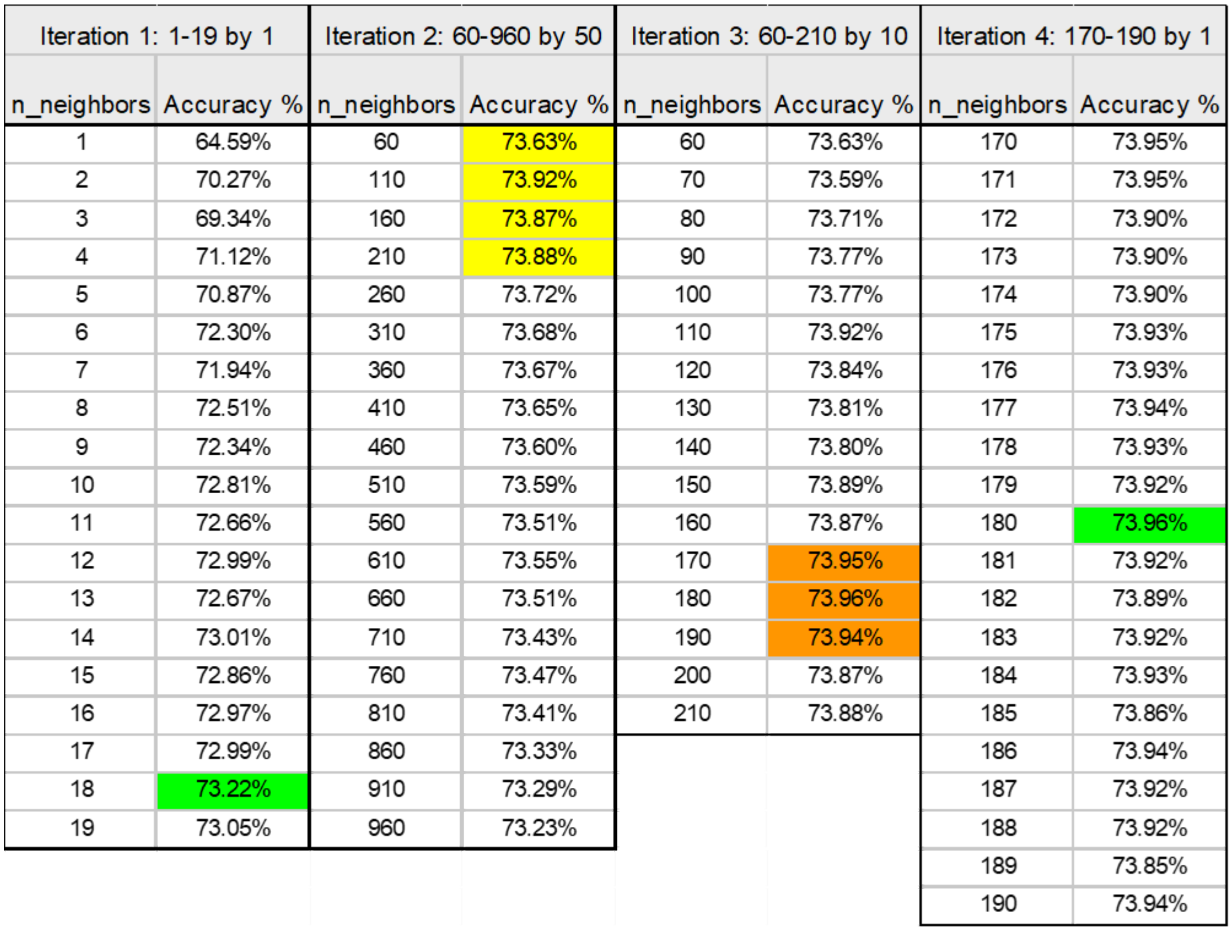
Position label (count)	What it stands for	Changed to
ATT (1035)	Attacker	A
LSM (4379)	Long Stick Midfielder	M
DEF (948)	Defense	D
GK (19576)	Goalkeeper	G

Table 9: Full table of changes for supervised player position prediction model

Position label	What they stand for	Total	Changed to
AA,AFO,AQ,AT,ATK,ATM,ATMD,ATT,FW,RW,W	Attacker, Wing	1133	A
AMF,DLMS,DLSM,DM,DMF,FA,FM,FO,FOGO,FOM,FOS,GO,LDM,LM,LMS,LPM,LS,LSM,LSMD,LSMF,MB,MD,MDM,MF,MFA,MFO,MID,MIDF,SLM,SS,SSD,SSDM,SSM	Midfielder, Long Stick, Short Stick, Face Off, Face Off Get Off	9816	M
DB,DDM,DEF,DK,DL,DLS,LP,LSD	Defense, Long Pole, Long Stick	974	D
GK,GOAL,GT	Goalkeeper	19602	G
0,1,16,2,27,3,31,35,4,45,5,6,8,9,AD,AM,AS,B,C,CD,CK,D08,DFO,DG,DST,F,F0,FK,FOR,FS,FSO,GF,ID,IH,K,L,MA,MG,MIC,MIG,MK,N,NA,NF,O,PCS,Q,S,SR	unknown	564	blank

Appendix D: KNN accuracy scores

Table 10: Supporting tables with data used for finding the best accuracy score



Iteration 1: 1-19 by 1		Iteration 2: 60-960 by 50		Iteration 3: 60-210 by 10		Iteration 4: 170-190 by 1	
n_neighbors	Accuracy %	n_neighbors	Accuracy %	n_neighbors	Accuracy %	n_neighbors	Accuracy %
1	64.59%	60	73.63%	60	73.63%	170	73.95%
2	70.27%	110	73.92%	70	73.59%	171	73.95%
3	69.34%	160	73.87%	80	73.71%	172	73.90%
4	71.12%	210	73.88%	90	73.77%	173	73.90%
5	70.87%	260	73.72%	100	73.77%	174	73.90%
6	72.30%	310	73.68%	110	73.92%	175	73.93%
7	71.94%	360	73.67%	120	73.84%	176	73.93%
8	72.51%	410	73.65%	130	73.81%	177	73.94%
9	72.34%	460	73.60%	140	73.80%	178	73.93%
10	72.81%	510	73.59%	150	73.89%	179	73.92%
11	72.66%	560	73.51%	160	73.87%	180	73.96%
12	72.99%	610	73.55%	170	73.95%	181	73.92%
13	72.67%	660	73.51%	180	73.96%	182	73.89%
14	73.01%	710	73.43%	190	73.94%	183	73.92%
15	72.86%	760	73.47%	200	73.87%	184	73.93%
16	72.97%	810	73.41%	210	73.88%	185	73.86%
17	72.99%	860	73.33%			186	73.94%
18	73.22%	910	73.29%			187	73.92%
19	73.05%	960	73.23%			188	73.92%
						189	73.85%
						190	73.94%

Appendix E: Train/Test Splits and Ablation Results

Table 11: kNN, weights = uniform models' results

Model	Training %	Training score	Test score	Train - test	Features removed
KNN, n_neighbors=180, weights=uniform	60	74.22	73.77	0.45	
KNN, n_neighbors=180, weights=uniform	60	74.25	73.84	0.41	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won
KNN, n_neighbors=180, weights=uniform	65	74.19	73.72	0.47	
KNN, n_neighbors=180, weights=uniform	65	74.22	73.73	0.49	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won
KNN, n_neighbors=180, weights=uniform	70	74.19	73.66	0.53	
KNN, n_neighbors=180, weights=uniform	70	74.18	73.63	0.55	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won
KNN, n_neighbors=180, weights=uniform	75	74.2	73.96	0.24	
KNN, n_neighbors=180, weights=uniform	75	74.16	73.81	0.35	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won
KNN, n_neighbors=180, weights=uniform	80	74.19	74.02	0.17	
KNN, n_neighbors=180, weights=uniform	80	74.16	73.92	0.24	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won
KNN, n_neighbors=180, weights=uniform	85	74.21	73.79	0.42	
KNN, n_neighbors=180, weights=uniform	85	74.15	73.78	0.37	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won
KNN, n_neighbors=180, weights=uniform	90	74.21	73.87	0.34	
KNN, n_neighbors=180, weights=uniform	90	74.18	73.55	0.63	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won
KNN, n_neighbors=180, weights=uniform	95	74.24	73.78	0.46	
KNN, n_neighbors=180, weights=uniform	95	74.24	73.42	0.82	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won

Table 12: kNN, weights = distance models' results

Model	Training %	Training score	Test score	Train - test	Features removed
KNN, n_neighbors=180, weights=distance	60	82.27	72.54	9.73	
KNN, n_neighbors=180, weights=distance	60	77.93	72.85	5.08	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won
KNN, n_neighbors=180, weights=distance	65	82.09	72.54	9.55	
KNN, n_neighbors=180, weights=distance	65	77.85	72.79	5.06	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won
KNN, n_neighbors=180, weights=distance	70	81.92	72.56	9.36	
KNN, n_neighbors=180, weights=distance	70	77.7	72.79	4.91	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won
KNN, n_neighbors=180, weights=distance	75	81.76	72.69	9.07	
KNN, n_neighbors=180, weights=distance	75	77.58	72.96	4.62	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won
KNN, n_neighbors=180, weights=distance	80	81.58	72.75	8.83	
KNN, n_neighbors=180, weights=distance	80	77.46	72.98	4.48	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won
KNN, n_neighbors=180, weights=distance	85	81.46	72.56	8.9	
KNN, n_neighbors=180, weights=distance	85	77.37	72.97	4.4	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won
KNN, n_neighbors=180, weights=distance	90	81.33	72.59	8.74	
KNN, n_neighbors=180, weights=distance	90	77.3	72.73	4.57	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won
KNN, n_neighbors=180, weights=distance	95	81.21	72.49	8.72	
KNN, n_neighbors=180, weights=distance	95	77.24	72.3	4.94	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won

Table 13: Decision Tree models' results

Model	Training %	Training score	Test score	Train - test	Features removed
Decision Tree	60	82.27	70.78	11.49	
Decision Tree	65	82.09	70.85	11.24	
Decision Tree	70	81.92	70.81	11.11	
Decision Tree	75	81.76	70.96	10.8	
Decision Tree	75	81.54	71.11	10.43	goalie saves, goals allowed, penalties, faceoffs won
Decision Tree	75	76.51	72.45	4.06	goals, assists, penalty time, shots on goal
Decision Tree	75	72.5	72.04	0.46	turnovers, caused turnovers, ground balls
Decision Tree	75	77.57	70.98	6.59	shots
Decision Tree	75	73.83	69.91	3.92	shots, shots on goal
Decision Tree	75	81.71	70.92	10.79	goalie seconds
Decision Tree	75	81.51	70.71	10.8	goalie seconds, goalie saves
Decision Tree	75	77.59	72.21	5.38	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won
Decision Tree	80	81.58	70.99	10.59	
Decision Tree	85	81.46	70.82	10.64	
Decision Tree	90	81.33	70.95	10.38	
Decision Tree	95	81.21	71.07	10.14	

Table 14: Random Forest models' results

Model	Training %	Training score	Test score	Train - test	Features removed
Random Forest	60	82.27	72.23	10.04	
Random Forest	65	82.09	72.16	9.93	
Random Forest	70	81.92	72.05	9.87	
Random Forest	75	81.76	72.23	9.53	
Random Forest	75	81.54	72.22	9.32	goalie saves, goals allowed, penalties, faceoffs won
Random Forest	75	76.51	72.7	3.81	goals, assists, penalty time, shots on goal
Random Forest	75	72.5	72.05	0.45	turnovers, caused turnovers, ground balls
Random Forest	75	77.56	71.58	5.98	shots
Random Forest	75	73.83	70.4	3.43	shots, shots on goal
Random Forest	75	81.7	72.18	9.52	goalie seconds
Random Forest	75	81.51	71.96	9.55	goalie seconds, goalie saves
Random Forest	75	77.58	72.59	4.99	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won
Random Forest	80	81.57	72.29	9.28	
Random Forest	85	81.46	72.14	9.32	
Random Forest	90	81.33	72.3	9.03	
Random Forest	95	81.2	71.97	9.23	

Table 15: Naive Bayes-Gaussian models' results

Model	Training %	Training score	Test score	Train - test	Features removed
Gaussian, Naive Bayes	60	71.59	71.54	0.05	
Gaussian, Naive Bayes	60	72.01	72	0.01	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won
Gaussian, Naive Bayes	65	71.56	71.59	-0.03	
Gaussian, Naive Bayes	65	71.98	72.05	-0.07	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won
Gaussian, Naive Bayes	70	71.65	71.47	0.18	
Gaussian, Naive Bayes	70	71.99	71.85	0.14	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won
Gaussian, Naive Bayes	75	71.61	71.58	0.03	
Gaussian, Naive Bayes	75	71.92	71.88	0.04	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won
Gaussian, Naive Bayes	80	71.59	71.69	-0.1	
Gaussian, Naive Bayes	80	71.9	71.98	-0.08	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won
Gaussian, Naive Bayes	85	71.61	71.64	-0.03	
Gaussian, Naive Bayes	85	71.95	71.9	0.05	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won
Gaussian, Naive Bayes	90	71.64	71.53	0.11	
Gaussian, Naive Bayes	90	71.95	71.85	0.1	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won
Gaussian, Naive Bayes	95	71.66	71.03	0.63	
Gaussian, Naive Bayes	95	71.97	71.29	0.68	goals, assists, penalty time, shots on goal, goalie saves, goals allowed, faceoffs won

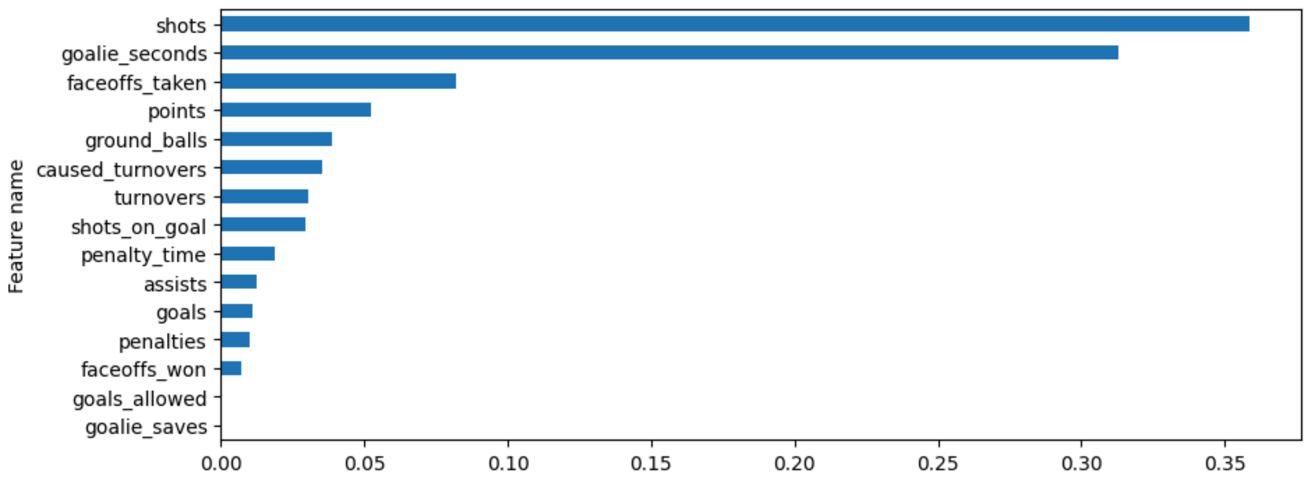
Appendix F: k's and PCs

Table 16: Best k(n_neighbors) and Principal Component values by year

Year	k's	PCs
2015	5	5
2016	6	5
2017	7	5
2018	5	5
2019	7	5
2020	5	6
2021	3	5
2022	6	5
2023	5	6

Appendix G: Decision Tree

Figure 15: Decision Tree feature importance for 2019



Appendix H: Comparing init

k-means++ vs random for inertia, Davies Bouldin, and Calinski Harabasz

Figure 16: Kmeans Inertia for 2019

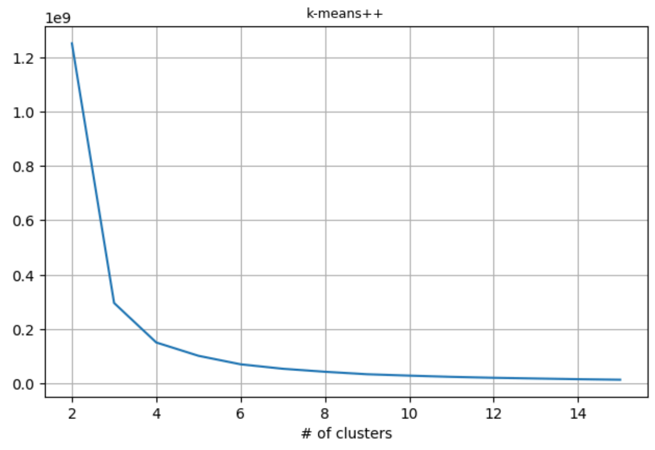


Figure 17: Where is the highest CH and the lowest BD?

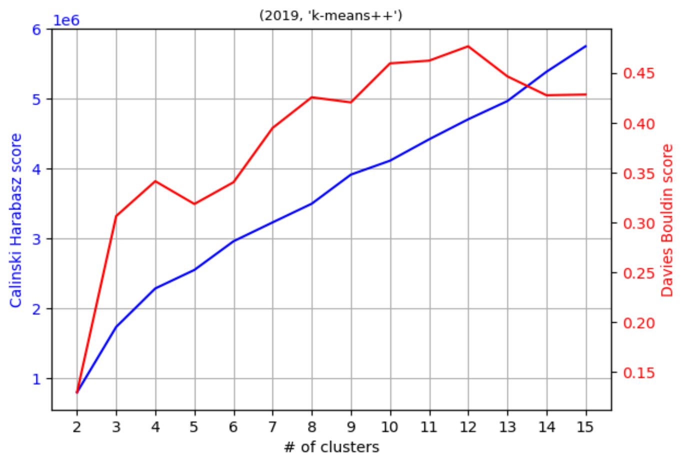


Figure 4: Kmeans Inertia for 2019

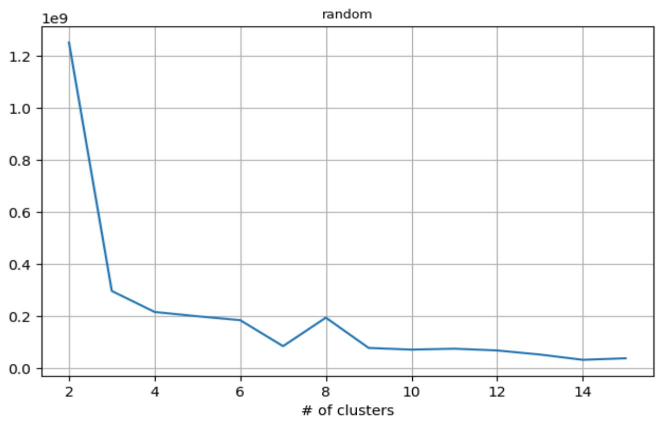
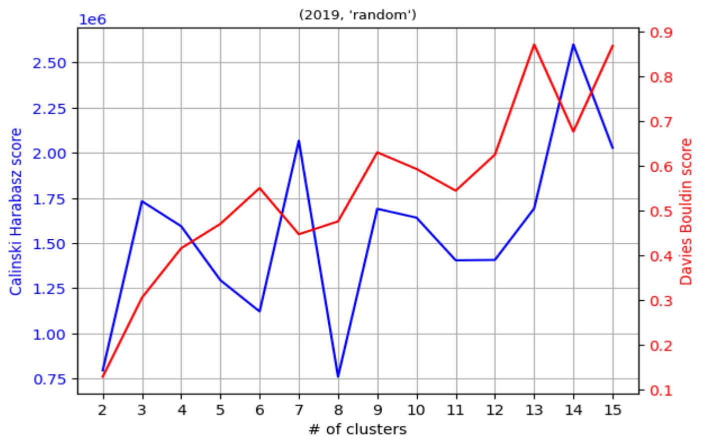
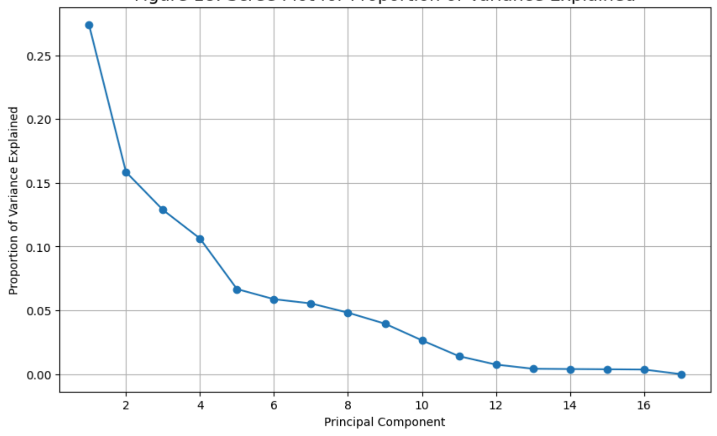


Figure 5: Where is the highest CH and the lowest BD?



Appendix I: Scree 2019

Figure 18: Scree Plot for Proportion of Variance Explained



Appendix J: Dendrogram by year

Figure 19: 2015 Dendrogram

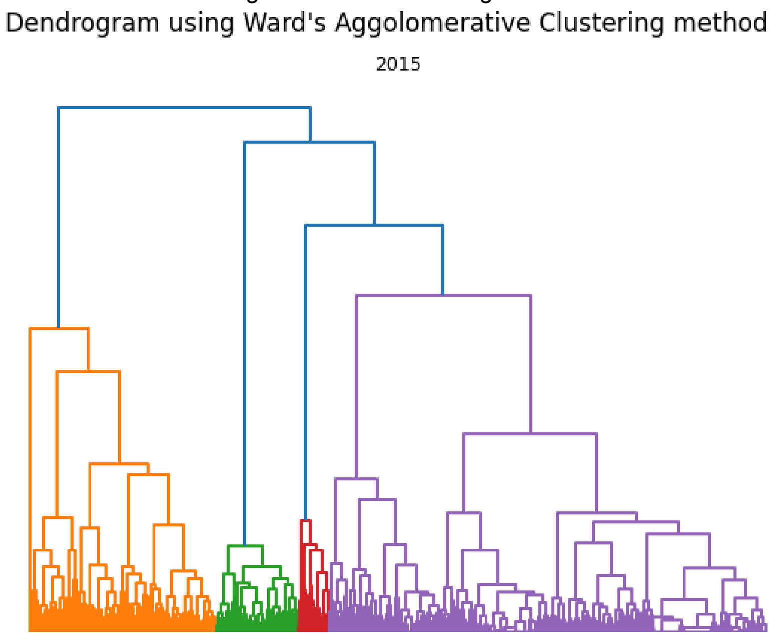


Figure 20: 2017 Dendrogram

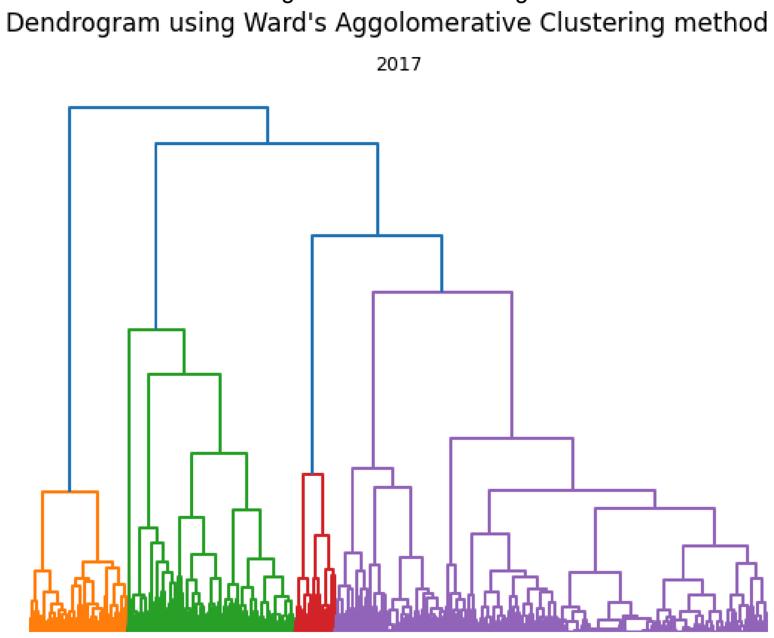


Figure 21: 2020 Dendrogram

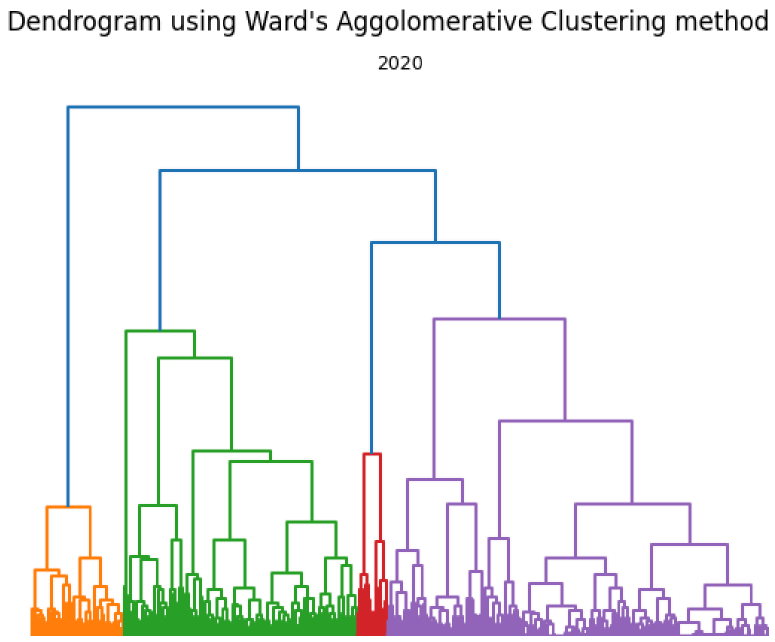
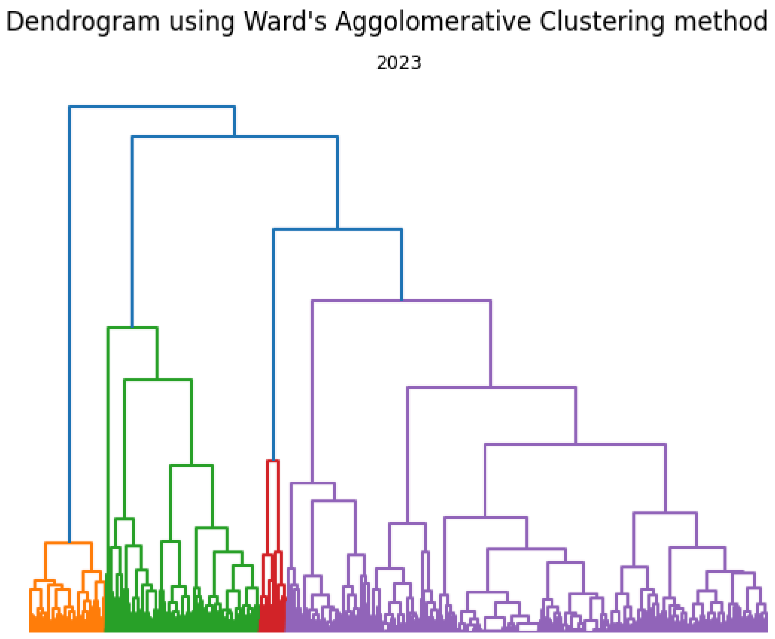


Figure 22: 2023 Dendrogram



Appendix K: Scatter Biplot by year

Figure 23: PCA Biplot with Scatterplot by Cluster for 2015

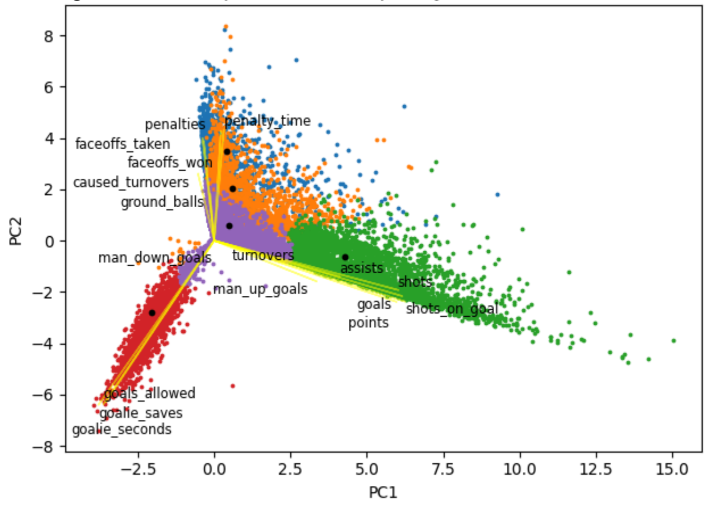


Figure 24: PCA Biplot with Scatterplot by Cluster for 2017

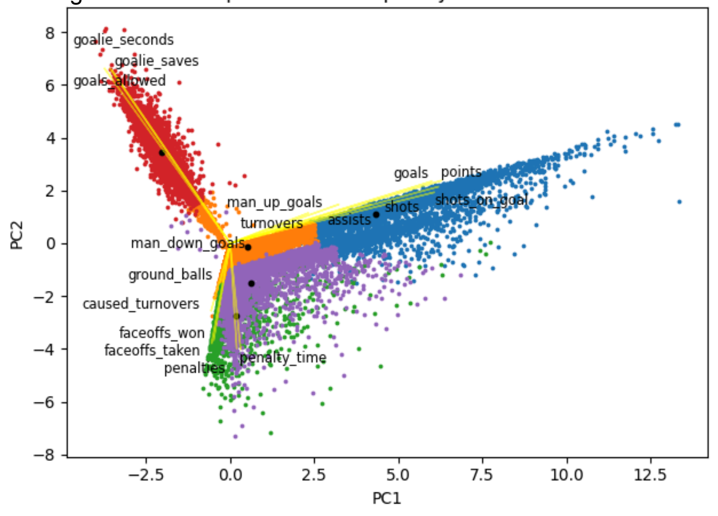
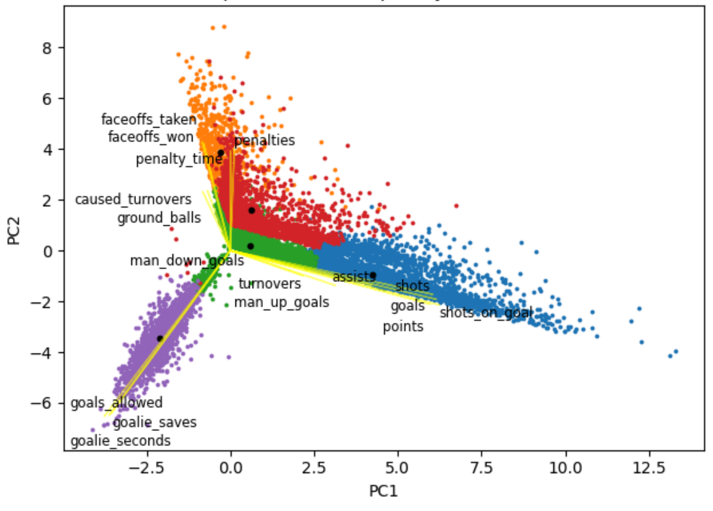


Figure 25: PCA Biplot with Scatterplot by Cluster for 2022



Appendix L: Sparse PCA heat maps by year

Figure 26: Heat map for 2015 (2016-2019 and 2021-22 looks the same)

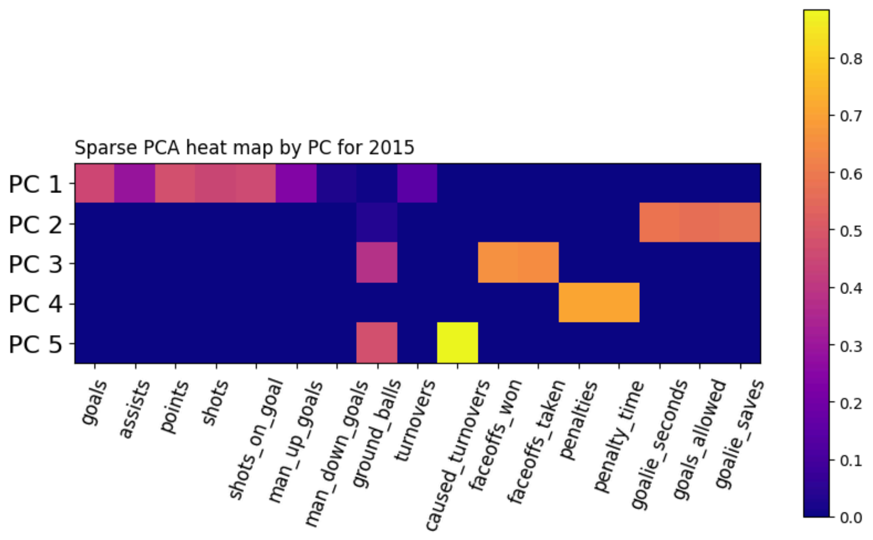
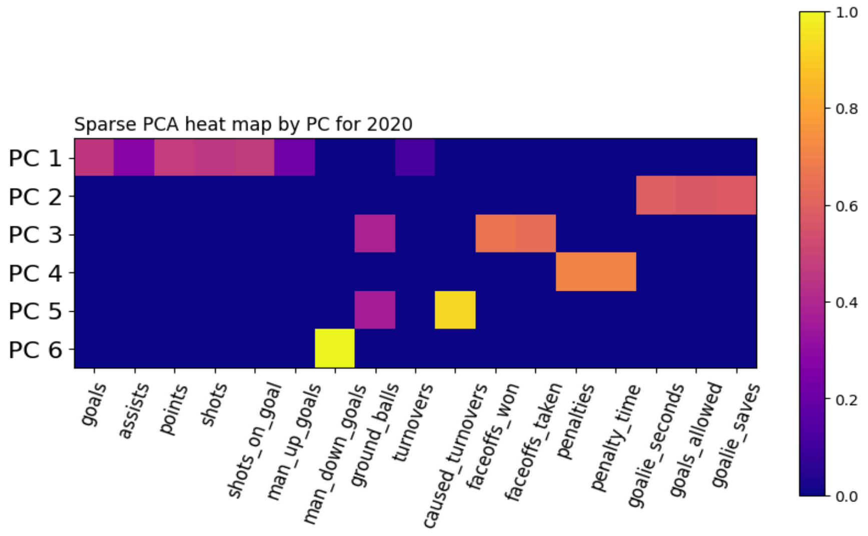


Figure 27: Heat map for 2020 (2023 looks the same)


Sparse PCA heat map by PC for 2020
PC 1
PC 2
PC 3
PC 4
PC 5
PC 6
goals
assists
points
shots
shots_on_goal
man_up_goals
man_down_goals
ground_balls
turnovers
caused_turnovers
faceoffs_won
faceoffs_taken
penalties
penalty_time
goalie_seconds
goals_allowed
goalie_saves