Bachelor's Studies

Field: Quantitative Methods in Economics and Information Systems
Specialization: Methods of Decision Analysis

Aleksander Wojciech Andrzejewski
Student No.: 81689

Models of Player Evaluation in Football

Bachelor's thesis
under scientific supervision of
Michał Jakubczyk, PhD (habil.), prof. SGH
written in Institute
of Econometrics

Warsaw 2021

Contents

Introduction

Data are changing the world we know at an enormous pace. Companies invest increasingly more money in their data science departments to optimise their performance. World Economic Forum estimated that 44 zettabytes (10^{21} bytes) of data would have been collected by the end of 2020. Every day four petabytes of data are generated only from Facebook[1]. This change is visible also in the world of sports described frequently using the phrase "Moneyball" regarding to the book "Moneyball. The Art of Winning an Unfair Game" about the Oakland Athletics baseball team's 2002 Major League Baseball season, during which the club's General Manager Billy Beane came up with a system completely based on mathematics[2]. Such a change occurs in football as well. In the 2017/2018 Premier League season more than 630000 observations of event data were collected[3]. Clubs install player tracking systems in their stadiums that collect player positions every 25 ms. Thanks to the infrastructure, they are able to collect over 100000 observations of tracking data from each game. Even more sports teams acquire data scientists as it became one of the fields where teams try to outperform their rivals. For example, after being taken over by professional better Matthew Benham, who emphasised mathematical modelling in this discipline, FC Midtjylland won their first Danish Championship in history[4].

Scouting became one of the areas the clubs compete in. The possibility to scout and buy a potential superstar at a lower cost is one of the options that a team gains an edge. Not only does scouting allow to help teams make decisions about transfers, but also it can assist the coach with choosing his best-performing players. This part of club management is one of the most affected by data changing the world of sports. Multiple mathematical models were developed to evaluate players' performance during a football match. Applying appropriate measures can

[1]Desjardins, J. (2019) *How Much Data is Generated Each Day*, World Economic Forum, Retrieved from `https://www.weforum.org/agenda/2019/04/how-much-data-is-generated-each-day-cf4bddf29f/` (Accessed:1 January 2021).

[2]Lewis, M. (2003) *Moneyball. The Art of Winning an Unfair Game*, New York, W.W. Norton.

[3]Pappalardo, L., Cintia, P., Rossi, A. et al. (2019) *A Public Data Set of Spatio-Temporal Match Events in Soccer Competitions*, Nature Scientific Data, vol. 6, no. 236.

[4]Ingle, S. (2015) *How Midtjyland Took the Analytical Route Towards the Champions League*, The Guardian, Retrieved from `www.theguardian.com/football/2015/jul/27/how-fc-midtjylland-analytical-route-champions-league-brentford-matthew-benham`. (Accessed: 12 December 2020).

show us different results than a simple eye-test. However, scouts have to be aware of multiple simplifications in their calculations.

FIFA World Cup is an event that happens every four years. It is considered to be the most prestigious football competition. Scouts from clubs all over the world are present during this event. After their analyses, clubs decide to make transfers of the best performing players. For instance, Real Madrid bought James Rodriguez[5] after he was awarded the Golden Boot trophy for his performance during the 2014 FIFA World Cup[6]. The last FIFA World Cup took place in 2018 in Russia. Multiple scouts were present during this event and loads of data were collected.

In this thesis, I want to evaluate the individual players' performance during the 2018 FIFA World Cup using a data-driven approach. To do so, I will use different mathematical models proposed by various football data scientists. The first model I would like to use is the Expected Goals model, which is considered to be one of the first statistical models in football. It enables to measure the probability of scoring a goal from a given position on the pitch[7]. Secondly, I will calculate known from basketball Plus-Minus ratings, which gives a player a value given his team performance while he was on and off the field[8]. Then, I will make calculations basing on the Expected Threat model presented by blogger Karun Singh[9], who in his research depended on Markov chains models proposed by Arsenal FC Data Scientist Sarah Rudd[10]. The last model I want to use in my calculations is the Valuing Actions on Estimated Probability model, which predicts the probability scoring and conceding a goal after every action taken by the player on the pitch[11]. I will discuss each model's simplifications and choose the most appropriate one to construct my team of the best performing players at each position. Last but not least, I want to compare the results of my calculations with the journalists' teams of the tournament.

In the first chapter, I will explain the economic aspects of football clubs and the role of scouting in football club management. In the second chapter of this thesis, I will introduce models commonly used in football to evaluate players' performance. In the third chapter, I would like

---

[5]Real Madrid. (2014) *Communicado Oficial: James Rodriguez*, Retrieved from `www.realmadrid.com/noticias/2014/07/comunicado-oficial-james-rodriguez`. (Accessed: 12 December 2020).

[6]FIFA. (2014) *Adidas Golden Boot*, Retrieved from `www.fifa.com/worldcup/news/adidas-golden-boot-2336480`. (Accessed: 12 December 2020).

[7]Pollard, R., Reep C. (1997) *Measuring the Effectiveness of Playing Strategies at Soccer*, Journal of the Royal Statistical Society: Series D, vol. 46, no. 4, p. 459-582.

[8]Hvattum, L. M., Saebo, O. (2015) *Evaluating the Efficiency of the Association Football Transfer Market Using Regression Based Player Ratings*, In: NIK: Norsk Informatikkonferanse, Bibsys Open Journal Systems, 12 pages.

[9]Singh, K. (2019) *Introducing Expected Threat* Retrieved from `https://karun.in/blog/expected-threat.html` (Accessed: 2 January 2021).

[10]Rudd, S. (2011) *A Framework for Tactical Analysis and Individual Offensive Production Assessment in Soccer Using Markov Chains*, New England Symposium on Statistics in Sports.

[11]Decroos, T., Bransen, L. Van Haaren, J. Davis, J. (2019) *Actions Speak Louder than Goals: Valuing Player Actions in Soccer*, Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, p. 1851–1861.

to briefly describe the data set used in calculations, provide the results of my calculations and evaluate the results of each model. In the last chapter, I will construct the data-driven team of the tournament and compare it with an expert's one.

The main goal I set in this thesis is to research and compare how data changes the evaluation of players and how the choices of the best players differ between mathematical models and a simple eye-test. I would also show some of the simplifications and misconceptions of data and models used in the football industry than can influence the results.

My research can help various football scouts to find players who performed the best during the 2018 FIFA World Cup. Presented results and methods used in this thesis can help football managers make transfers. Knowing the players' market value, they can rank them and choose the best option for their team. My calculations can also help journalists choose the best performing players in every tournament. Football fans from all over the world will be able to check the reliability of sportswriters and discuss which player is better given a more impartial argument, which is a data-driven mathematical calculation, than their or journalist's subjective eye-test.

Chapter 1

Economics of Football Clubs

Football teams are businesses aiming to maximize their profits. The value of the European football market is estimated at €28.9 billion[1]. Revenues and expenditures come from multiple aspects of running a football club, such as broadcasting rights, kit deals, sponsorship, transfers and players' wages[2]. Successful management leads to winning trophies and financial success.

1.1. Broadcasting Rights in Football

Fans worldwide want to follow and watch games of their favourite football teams. As the fans are eager to pay money for football games transmissions, the broadcasters compete in the tender to acquire rights from leagues and federations to broadcast games. Companies sponsoring clubs also sponsor the league as their logo is portrayed on the official league graphics[3]. In Table 1.1 the biggest broadcast deals are presented.

[1]Deloitte (2020) *Annual Review of Football Finance*, Retrieved from `https://www2.deloitte.com/uk/en/pages/sports-business-group/articles/annual-review-of-football-finance.html` (Accessed: 11 April 2021).

[2]Szymanski, S. (2016) *Inside the Business of Football*, OpenLearn, Retrieved from `https://openlearn.medium.com/inside-the-business-of-football-4537acec31bb` (Accessed: 11 April 2021).

[3]Gazapo, C. (2019) *TV Rights in Football - Premier League Analysis*, Sports Business Institute Barcelona, Retrieved from `https://www.sbibarcelona.com/newsdetails/index/403`, (Accessed 14 April 2021).

Table 1.1: The Biggest Broadcast Deals of the 2020/2021 Club Season

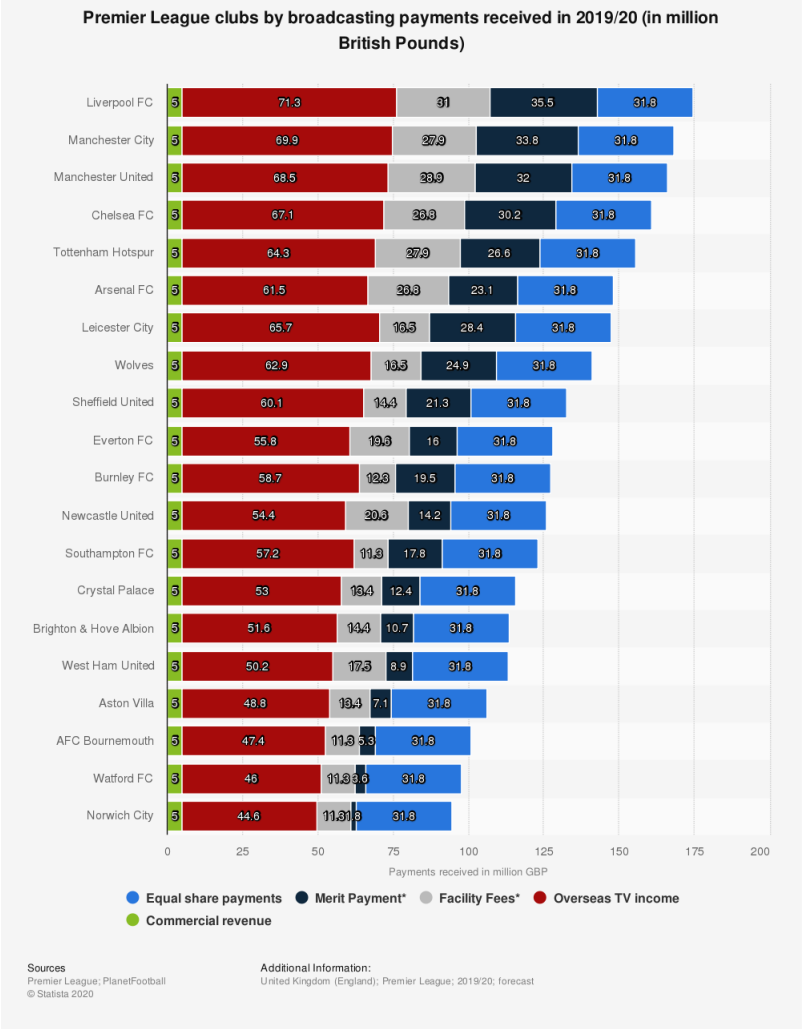
Rank	League	Country	Value [€ per season]
1	Premier League	United Kingdom	1.92B
2	Ligue 1	France	1.172B
3	Bundesliga	Germany	1.16B
4	La Liga	Spain	1.14B
5	Serie A	Italy	973M
6	Super Lig	Turkey	450M
7	Campeonato Serie A	Brazil	288M
8	English Football League	Adidas	137M
9	Primeria Liga	Portugal	126M
10	First Division A	Belgium	103M

Source: own elaboration, Wikipedia

Then, the money from the league is partially distributed between the football clubs. In Premier League, 50% of domestic broadcast revenue is split equally between the clubs (Equal Share Payment), 25% is for the final position in the table (Merit Payment). Worth mentioning is the fact that Merit Payments are the prize money of the competition. The remainder is paid for every broadcast game (Facility Fees). In the case of overseas rights money, it is shared equally between all 20 clubs[4]. The size of broadcasting payments in each category for the season 2019/2020 is presented in Figure 1.1.

[4]Gazapo, C. (2019) *TV Rights in Football - Premier League Analysis*, Sports Business Institute Barcelona, Retrieved from `https://www.sbibarcelona.com/newsdetails/index/403`, (Accessed 14 April 2021).

Figure 1.1: Premier League Clubs by Broadcasting Payments Recieved in the 2019/2020 Season



Source: Statista

Liverpool FC, the 2019/2020 Premier League Champion, received £174,6 million from broadcasting payments making the club the best earner from the broadcast revenue. The British six most popular teams, Liverpool FC, Manchester City, Manchester United, Chelsea FC, Tottenham Hotspur and Arsenal FC, were also the teams which benefited the most from the Premier League distribution system. They attract the most significant number of fans, so the Facility Fees paid to those teams are higher. Because of that, Arsenal FC, one of the most popular teams worldwide, despite finishing the 2019/2020 season in the 8^{th} place, earned more money than Leicester City, which finished 5^{th}.

For Spanish La Liga and Italian Serie A the distribution is the same as in Premier League. There are also 3 broadcast revenue split categories in France - Equal Share Payment, Merit Payment and Facility Fees, but the proportion is adequately 50%, 30%, 20%. In the case of German Bundesliga, only Equal Share Payments and Merit Payments are paid in proportion 65:35 of

total money distributed to the clubs[5].

The revenue from broadcast rights is the main source of income for the smaller clubs. For example, it amounts to 77% of total income of Spanish club Leganes. In the case of soccer giants its influence is less significant. Only 36% of Real Madrid's total revenue comes from the broadcasting rights[6]. The biggest football clubs claim that the revenue from broadcasting rights is insufficient. Therefore, in April 2021, 15 most well-known teams tried to establish a new competition called The Super League sharing €3.5 billion between them[7].

1.2. Sponsorship and Kit Deals in Football

Sponsorship means money being given usually by a company to support an organisation[8]. Companies eagerly sponsor football clubs in exchange for companies logo placement on football kits, websites, pitch banners or social media channels[9]. They may also obtain the right to name the stadium, as in the case of Bayern Munich arena called Allianz Arena after the sponsor - Allianz Insurance. The biggest sponsorship deals of the 2020/2021 football season are presented in Table 1.2.

[5]Fahran (2021) *Football*, Sportscriber, Retrieved from `https://sportscriber.com/football/` (Accessed:17 April 2021).

[6]KPMG (2019) *Broadcasting Revenue Landscape – Big Money in The "Big Five" Leagues*, Football Benchmark, Retrieved from `https://www.footballbenchmark.com/library/broadcasting_revenue_landscape_big_money_in_the_big_five_leagues` (Accessed: 24 April 2021).

[7]Diaz, J. F. (2021) *Clubs to Earn Almost Triple the Money from European Super League than Champions League*, Retrieved from `https://www.marca.com/en/football/international-football/2021/04/19/607d46efca4741ff368b4606.html` (Accessed: 24 April 2021).

[8]Cambridge Dictionary *Meaning of Sponsorship in English*, Retrieved from `https://dictionary.cambridge.org/dictionary/english/sponsorship` (Accessed: 13 April 2021).

[9]AvecSport (2019) *How to Attract a Sponsor for Your Grassrots Football Team*, Retrieved from `https://avecsport.com/blog/how-to-attract-a-sponsor-for-your-grassroots-football-team` (Accessed 14 April 2021).

Table 1.2: The Biggest Kit Sponsorship Deals of the 2020/2021 Club Season

Rank	Club	Company	Value [€ per season]
1	Manchester United	Chevrolet	71M
2	Real Madrid	Fly Emirates	70M
3	FC Barcelona	Rakuten	55M
4	Paris Saint-Germain FC	Fly Emirates	50M
5	Chelsea FC	UK-Telco Three	47M
6	Manchester City	Etihad Airways	46M
7	Bayern Munich	T-Mobile	42M
8	Liverpool	Standard Chartered	32M
9	Arsenal FC	Fly Emirates	31M
10	Tottenham FC	AIA	29M

Source: own elaboration, SportsKhabri

Companies invest the most significant amount of money in the teams known worldwide. Manchester United, FC Barcelona and Real Madrid have the most considerable number of fans in the world. Presence on the shirts during worldwide broadcast games is an advertisement as the logo on the shirt is portrayed during the whole game.

Football clubs get money not only from sponsorship but also through selling shirts. They are not sold directly by football clubs but by the sportswear firms such as Nike, Adidas or Puma. Clubs receive a small percentage (7.5% - 15%) of revenue generated through selling football jerseys. It is beneficial for them as the companies are obliged to sign kit licensing deals with the clubs worth in total €920 million per year in the five biggest European leagues - English Premier League, Spanish La Liga, German Bundesliga, Italian Serie A and French Ligue 1[10].

The size of kit licensing deals varies between the clubs. Sportswear companies make better offers to worldwide known and most successful teams. The biggest deals of the 2020/2021 club football season are presented in Table 1.3.

[10]Downs, D. (2020) *Can the £1 billion football jersey business overcome Covid-19?*, Vogue Business, Retrieved from `https://www.voguebusiness.com/companies/football-jersey-sales-premier-league-messi` (Accessed: 11 April 2021).

Table 1.3: The Biggest Kit Licensing Deals of the 2020/2021 Club Season

Rank	Club	Company	Value [€ per season]
1	FC Barcelona	Nike	155M
2	Real Madrid	Adidas	120M
3	Manchester United	Adidas	90M
4	Paris Saint-Germain FC	Nike	86M
5	Manchester City	Puma	76M
6	Arsenal FC	Adidas	70M
7	Chelsea FC	Nike	65M
8	Bayern Munich	Adidas	59M
9	Juventus	Adidas	52M
10	Liverpool FC	Nike	35M

Source: own elaboration, SportMob

The ten most extensive kit licensing deals add up to the lion's share of the total worth of these deals in the 5 biggest European leagues. The more popular the club is, the bigger offer can be placed as more fans are willing to purchase the kit of their favourite team. Real Madrid and FC Barcelona are considered the most popular clubs globally. Therefore, it is not surprising that their kit licensing deals are the greatest. Adidas and Nike are the biggest players in the football kit industry as they are responsible for 9 out of 10 biggest deals in club football.

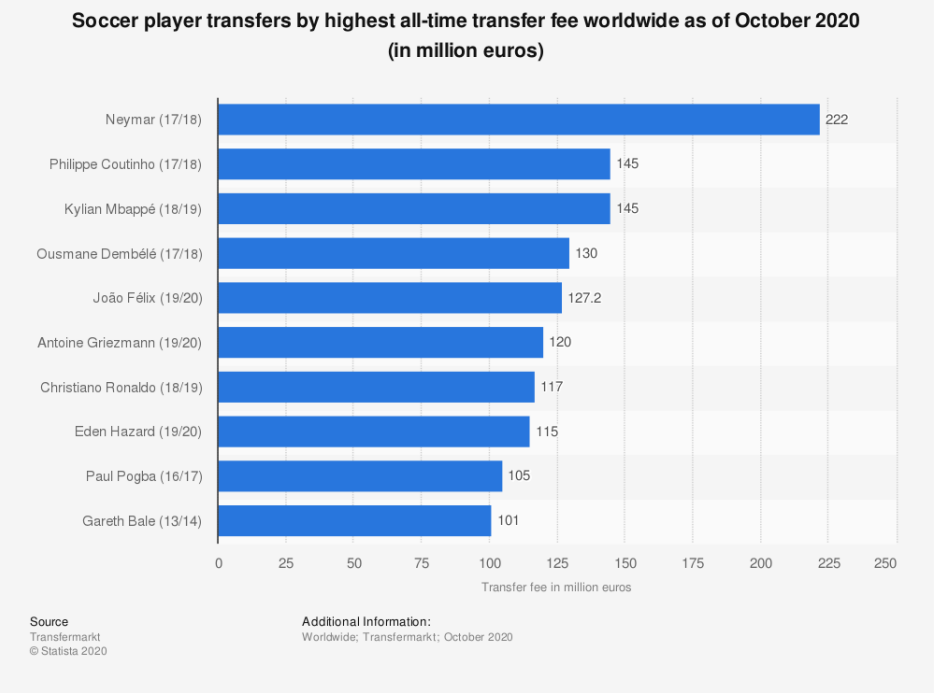
1.3. Transfer Market and Player Wages

From the money from sponsorship and broadcast rights, clubs want to acquire new footballers to achieve better results in the competitions. After a contract is signed for a given period, football players can change the club they play for in two different ways. The first one is to wait until the contract expires and the second is that another club buys a player for a previously negotiated transfer fee. Sometimes, the release clause is stated in the player's contract, which signifies the amount of money that a club has to pay for the footballer.

17007 transfers were made between 4178 clubs from 187 different football associations in 2020. Only 13.3% of the transfers were with transfer fees. The sum of these transfer fees amounted to €5.63 billion. Clubs associated in the British Football Association spent €1.4 billion, making the federation the one that spends the most on transfers, whereas clubs from the RFEF (Royal Spanish Football Federation) received the biggest number of money for players -

€634 million[11]. Chelsea FC was the club that spent the most in 2020. The total expenditure for new players amounted to €274 million, including €80 million paid to Bayer Leverkusen for Kai Havertz, making him the most expensive signing that year[12]. Despite the price being that high, it was not enough for this transfer to be placed among the top 10 most expensive transfers in football history, which are presented in Figure 1.2.

Figure 1.2: The Biggest Transfers in the Football History



Source: Statista

€222 million paid by Paris Saint-Germain to FC Barcelona made Neymar's transfer the most expensive one of all time. Thanks to the funds from Qatar Sports Investments invested in the club, the transfer was possible. The money activated the domino effect as FC Barcelona decided to buy O. Dembele from Borussia Dortmund and P. Coutinho from Liverpool FC. With the money earned, Liverpool FC bought Alisson from AS Roma and Virgil van Dijk from Southampton FC. Player's transfer from one club to another activates a series of other transfers made by teams to strengthen their squads and replace missing players.

After a transfer is made, clubs and players agree on the agreement's content. Usually, its length, player's salary and bonuses are determined. Therefore, the transfer fee is not the only expenditure the club has to make while acquiring a footballer. The best football players are among

[11]FIFA (2021) *A Review of International Football Transfers Worldwide*, Global Transfer Market Report, Retrieved from `https://www.fifa.com/who-we-are/legal/tms/tms-reports/` (Accessed: 15 April 2021).

[12]Transfermarkt (2021) *Chelsea FC - Transfers 20/21*, Retrieved from `https://www.transfermarkt.com/fc-chelsea/transfers/verein/631` (Accessed: 15 April 2021).

the world's highest-paid sportsmen. The highest salaries (before taxation) of the 2020/2021 club season are presented in Table 1.4.

Table 1.4: The Highest Player Salaries of the 2020/2021 Club Season

Rank	Player	Club	Value [€ per season]
1	L. Messi	FC Barcelona	77M
2	Neymar	Paris Saint-Germain FC	65M
3	C. Ronaldo	Juventus Turin	58M
4	K. Mbappe	Paris Saint-Germain FC	23M
4	P. Pogba	Manchester United	23M
4	A. Griezmann	FC Barcelona	23M
7	M. Salah	Liverpool FC	20M
7	R. Lewandowski	Bayern Munich	20M
7	D. de Gea	Manchester United	20M
10	G. Bale	Tottenham Hotspur	19M

Source: own elaboration, Forbes

L. Messi, Neymar and C. Ronaldo are the players who earn the most. It is not a surprise that those three footballers are considered the best in the world. They are signed by the best and most popular clubs globally that have enough money to meet their expectations. The better the player is, the more money the club offers to acquire him. However, to win competitions, teams have to sign the best performing players.

Scouting can help teams find players whom they want to buy and adequately evaluate their value on the transfer market. As in every purchase, the goal is to buy a good footballer and strengthen the squad as cheap as possible and not overpay him. The aim is to make the team as competitive as possible with the budget. Good scouting allows the team to fight for the trophies or selling the player with profit. Trophies bring the club higher revenues from broadcast rights and more fans worldwide. That leads to better recognition and increased interest from sponsors. More money from broadcast rights and sponsorship deals can be used to buy even better players, which leads to winning trophies and financial success.

Chapter 2

Player Evaluation Models Used in Football

Scouting is an integral part of successful club management. With the constant growth of importance of the data in the world, multiple player evaluation models such as Expected Goals, Plus-Minus Models, Expected Threat and Valuing Players by Estimating Probabilities were introduced to help scouts objectively evaluate players' performance.

2.1. Expected Goals Model

Expected Goals, popularly called xG, is a method of measuring the quality of scoring chance in football. The value can be interpreted as the probability of scoring a goal from a given situation. It is commonly used to evaluate player performance as it shows the expected number of goals that one should have scored during a tournament or a season[1].

R. Pollard and C. Reep (1997) built the first model predicting the shot outcome. They measured the probability of scoring a goal using the distance to the goal, the angle formed by the line connecting the shot location with the nearer goal post and the line parallel to the goal line, a binary variable with a value 1 if the shot was a first touch shot, else 0, a binary variable valued 1 if the defender was closer than 1 yard from a shot location, else 0 and last, but not the least, a binary variable signifying whether a shot was from a set piece (1) or open play (0). The dependent variable was a binary variable with a value 1 if a goal was scored, else 0. Then two logistic regression models were built - one for kicked shots and one for headers. Next, a probability of scoring a goal from each shot was estimated. Worth mentioning is the fact that penalty kicks were excluded from the model[2]. In his previous research, R. Pollard (1995) has estimated the probability of scoring a penalty at 0.8[3].

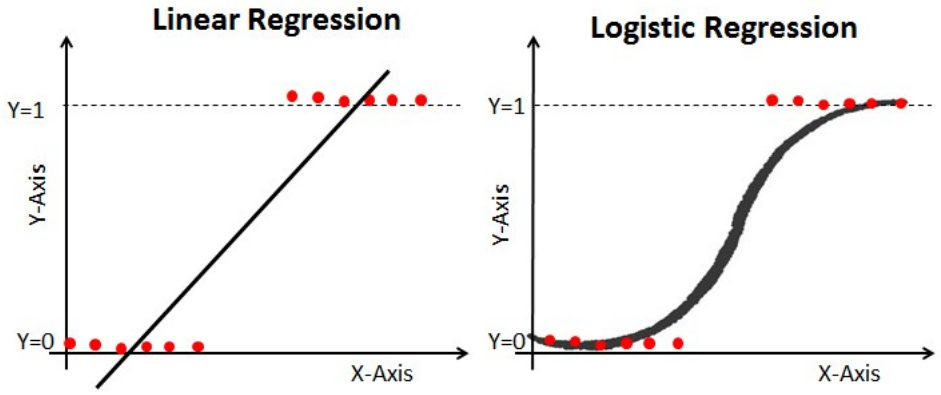
[1]OPTA *Advanced Metrics*, Retrieved from `https://www.optasports.com/services/analytics/advanced-metrics/` (Accessed: 5 March 2020).

[2]Pollard, R., Reep C. (1997) *Measuring the Effectiveness of Playing Strategies at Soccer*, Journal of the Royal Statistical Society: Series D, vol. 46, no. 4, p. 459-582.

[3]Pollard, R. (1995) *Do Long Shots Pay Off?*, Soccer Journal, vol. 40, no. 3, p. 41-43.

The logistic regression model is a model used to estimate probabilities. The classical linear regression model fails to correctly estimate that value as the predicted outcome cannot be constrained on 0-1 interval. Therefore, in a logit model, a sigmoid function is used to restrict the interval. The difference between a linear and sigmoid function is presented in Figure 2.1.

Figure 2.1: Difference in Interval Restriction between Linear and Logistic Regression



Source: https://medium.com/@ODSC/logistic-regression-with-python-ede39f8573c7

Having the regression equation

$$y_i = \beta_0 + \beta_j * X_{ij} + \epsilon_i \tag{2.1}$$

one can estimate the probability of an occurrence as[4]

$$P(Y = 1|x_1, x_2, ..., x_n) = \frac{e^{\beta_0 + \sum_{i=1}^{n} \beta_i * x_i}}{1 + e^{\beta_0 + \sum_{i=1}^{n} \beta_i * x_i}} \tag{2.2}$$

2.2. Plus-Minus Models

Plus-minus is a statistic that measures a player's impact basing on his presence and absence on the pitch during the game. Basic plus-minus is calculated using the equation

$$PM_i = PS_i - PA_i \tag{2.3}$$

where PM_i is plus-minus score of player i, PS_i signifies points scored by the team when this player was present on the pitch and PA_i points allowed during the same period.

However, such an approach has several weaknesses. It does not consider the strength of the team's opponent and an individual player's quality. Moreover, when two players of the same team play a lot together, it is difficult to distinguish their impact on the score, as their plus-minus statistic is similar[5].

[4]Greene, W. H. (2003) *Econometric Analysis*, 3^{rd} edition, New York, Pearson Education Inc.

[5]Thomas, A. C., Ventura, S., Jensen, S., Ma, S. (2013) *Competing Process Hazard Function Models for Player Ratings in Ice Hockey*, The Annals of Applied Statistics, vol 7, p. 1497–1524.

The Adjusted Plus-Minus model was firstly introduced in the context of basketball by D. Rosenbaum (2004). The games are split into periods when no substitution happens. To differentiate players impact the multiple regression model is built

$$y_i = \beta_j * X_{ij} + \epsilon_i \tag{2.4}$$

where y_i is calculated as a difference of home and away team per possession multiplied by 100 and X_{ij} is defined for each segment of game as

$$X_{ij} = \begin{cases} 1 & \text{when player j played for the home team in the segment i,} \\ -1 & \text{when player j played for the away team in the segment i,} \\ 0 & \text{when player j played did not play in the segment i.} \end{cases}$$

Using Ordinary Least Squared method the coefficients can be estimated. The intercept estimates the impact of playing at home and β_j can be interpreted as the difference between player j and the rest of the players in a database[6].

However, the number of points scored in football and basketball differs. Therefore, for the Adjusted Plus-Minus in football the dependent and independent variables are calculated differently. Independent variables X_{ij} are defined as

$$X_{ij} = \begin{cases} e^{-kt} & \text{when player j played for the home team in the segment i,} \\ -e^{-kt} & \text{when player j played for the away team in the segment i,} \\ 0 & \text{when player j played did not play in the segment i,} \end{cases}$$

where t is the age of observations and k is a parameter determining its influence. The dependent variable y_i is calculated using the formula

$$y_i = \frac{90 * (PS_i - PA_i) * e^{-kt}}{D_i} \tag{2.5}$$

where PS_i signifies number of goals scored by the home team and PA_i number of goals conceded during period i, which lasted D_i minutes when no substitution happened[7].

In his research J. Sill (2010) found the Adjusted-Plus Minus inadequate, as the results may be noisy due to overfitting and multicollinearity. Instead of using maximum likelihood minimisation, he suggested using the Tikhonov regularization model[8], minimizing the function:

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} X_{ij}\beta_j)^2 - \lambda \sum_{j=1}^{p} \beta_j^2 \tag{2.6}$$

[6]Rosenbaum, D. (2004) *Measuring How NBA Players Help Their Teams Win*, 82games, Retrieved from `http://www.82games.com/comm30.html`. (Accessed: 28 February 2021).

[7]Hvattum, L. M., Saebo, O. (2015) *Evaluating the Efficiency of the Association Football Transfer Market Using Regression Based Player Ratings*, In: NIK: Norsk Informatikkonferanse, Bibsys Open Journal Systems, 12 pages.

[8]Sill, J. (2010) *Improved NBA Adjusted +/– Using Regularization and Out-Of-Sample Testing*, Proceedings of the 2010 MIT Sloan Sports Analytics Conference.

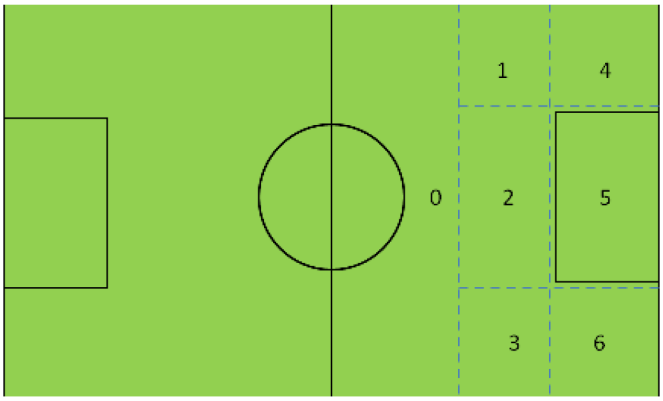
where λ is a regularization parameter helping to prevent overfitting the model with too many parameters at the expense of loosing some information. It penalises parameters from being too far away from 0[9].

According to Sill's research, such a technique, called Real Adjusted Plus-Minus, performs better than removing from the database players with low number of minutes played[10]. Using the Real Adjusted Plus-Minus as well as in the Adjusted Plus-Minus β_0 estimates the advantege of playing at home and β_j evaluates player influence - the higher the parameter is, the better the player performed[11].

2.3. Expected Threat Model

The Expected Threat model is a model based on transition probabilities between different zones on the pitch[12]. S. Rudd (2011) introduced the first framework for a model based on the transition between areas. The field was divided into 7 different zones, then 30 states concerning the defensive state were defined. The division is presented in Figure 2.2

Figure 2.2: Football Pitch Zones Divided by S. Ruud (attacking from left to right)



Source: A Framework for Tactical Analysis and Individual Offensive Production Assessment in Soccer Using Markov Chains

Additionally, this model consists of 2 additional states, which signify a goal or end of an possession. As the states are constantly changing in a game of football, a Markov chain model

[9]Bhattacharyya, S. (2018) *Ridge and Lasso Regression: L1 and L2 Regularization*, Towards Data Science, Retrieved from `https://towardsdatascience.com/ridge-and-lasso-regression-a-complete-guide-with-python-scikit-learn-e20e34bcbf0b`. (Accessed: 28 February 2021).

[10]Sill, J. (2010) *Improved NBA Adjusted +/– Using Regularization and Out-Of-Sample Testing*, Proceedings of the 2010 MIT Sloan Sports Analytics Conference.

[11]Hvattum, L. M., Saebo, O. (2019) *Modelling the Financial Contribution of Soccer Players to Their Clubs*, Journal of Sports Analytics, vol. 5, p. 23–34.

[12]Singh, K. (2019) *Introducing Expected Threat* Retrieved from `https://karun.in/blog/expected-threat.html` (Accessed: 1 March 2021).

can be built[13].

Markov chain is a sequence of random variables with probabilities of moving between states depending on the current state. The transition matrix P is a stochastic matrix with values P_{ij} signifying the transition probability p_{ij} from state i to state j. An absorbing state i is a state which satisfies the condition[14].

$$\forall_{i \neq j} p_{ij} = 0,\ p_{ii} = 1. \tag{2.7}$$

A discrete-time Markov chain is ergodic if[15]

$$\forall_{i,j} \lim_{n \to \infty} p_{ij}^n = e_j,\ \sum_{j=1} e_j = 1,\ \forall_j e_j \geq 0. \tag{2.8}$$

If a discrete Markov chain is ergodic, exists an ergodic matrix such that[16]

$$\lim_{n \to \infty} P^n = E. \tag{2.9}$$

In S. Rudd framework, states signifying a goal and a lost possession are absorbing states. The remaining elements of the transition matrix define the probability of moving from state i to state j and scoring a goal or losing the ball while in state i. Then, an ergodic matrix is calculated. A value of an action V_{ij}, which moved the ball form state i to state j, is a difference of probabilities of scoring the goal in the state j and state i[17]

$$V_{ij} = e_j - e_i. \tag{2.10}$$

The Expected Threat model uses many concepts from S. Rudd's framework. The football pitch is likewise divided into different areas and the value of an action is calculated as the difference between value of a zone where the action has finished and the zone where the action has begun. However, K. Singh (2019) applied several changes. First of all, he divided the pitch into 192 zones, 16 vertical by 12 horizontal as presented in Figure 2.3.

[13]Rudd, S. (2011) *A Framework for Tactical Analysis and Individual Offensive Production Assessment in Soccer Using Markov Chains*, New England Symposium on Statistics in Sports.

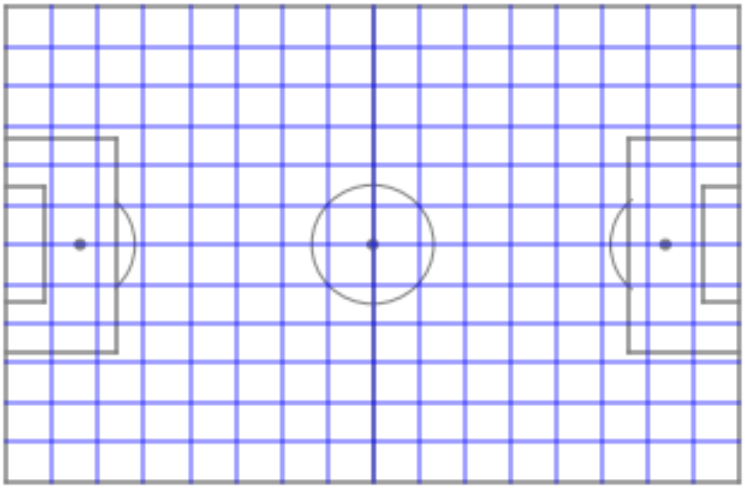
[14]Grimmett, G. R.; Stirzaker, D. R. (2001) *Probability and Random Processes*, 3^{nd} edition, Oxford University Press, Oxford.

[15]*Ibidem.*

[16]*Ibidem.*

[17]Rudd, S. (2011) *A Framework for Tactical Analysis and Individual Offensive Production Assessment in Soccer Using Markov Chains*, New England Symposium on Statistics in Sports.

Figure 2.3: Football Pitch Zones Divided by K. Singh (attacking from left to right)



Source: own elaboration, Python (matplotlib.pyplot and FCPython packages)

Moreover, the values for each area are calculated differently. At first the following attributes are calculated:

- s_i - the probability of shooting from zone i,
- m_i - the probability of moving the ball from zone i,
- g_i - the probability of scoring a goal from zone i,
- M - transition matrix consisting of moving probabilities between zones.

Then, a value of each zone is calculated according to the undermentioned equation.

$$T_i = (s_i * g_i) + (m_i * \sum_{j=1}^{192} M_{ij} * T_j). \tag{2.11}$$

To calculate the zone value, the calculations begin with $T = 0$ for each zone. Then, the formula is iteratively evaluated until it converges. Such a value can be interpreted as the probability of scoring a goal within the next n actions, if a buildup started in the given zone, where n is the number of iterations.

K. Singh's research has shown that 4-5 iterations are required so the formula converges[18] whereas T. Decroos *et al.* claims the necessary number of iterations to be 5-6[19].

After the Expected Threat value of each zone is calculated, the each action's values, which moves the ball, are computed as the difference of Expected Threat value of the ending zone and the starting one. Then players are evaluated based on this statistic. K. Singh suggested comparing players considering the cumulative change in Expected Threat of all actions throughout

[18]Singh, K. (2019) *Introducing expected threat* Retrieved from `https://karun.in/blog/expected-threat.html` (Accessed: 1 March 2021).

[19]Van Roy, M., Robberechts, P., Decroos, T., Davis, J. 2020. *Valuing On-the-Ball Actions in Soccer: A Critical Comparison of xT and VAEP*, AAAI-20 Workshop on Artifical Intelligence in Team Sports.

a season[20] whereas T. Decroos *et. al* suggested ranking them based on the same value per 90 minutes[21], that is

$$xT_{per90} = xT_{cummulative} * \frac{90}{minutes\,played}. \tag{2.12}$$

2.4. Valuing Actions by Estimating Probabilities

Valuing Actions by Estimating Probabilities (VAEP) is a framework used to evaluate the player's performance based on all events that occurred on the football pitch. To build this model, T. Decroos, L. Bransen and J. Van Haaren, J.Davis (2019) introduced SPADL (Soccer Player Action Description Language) - a language that describes actions taken by a player. Its purpose is to unify the terminology of data provided by multiple companies that optically collect it. SPADL also allows describing not the event on the pitch but by a player's action. It provides information about the start and end location and time of an action, result, a footballer who made it, his team and body part used to execute it.

After the data are converted to SPADL, two different machine learning models are built. The first one predicts the probability of a goal being scored, the second one of one being conceded within the next 10 actions. To do so, the features of 3 different categories are used. They are measured for current action and two ones preceding it.

- SPADL features - a set of features containing information received after presenting the data set using SPADL language,
- context features - variables providing information about the pace of play. To this category belong features such as the distance to the goal, an angle between a goal and an action location, the distance that the ball covered,
- game context features - current result which could define the way that team plays after scoring or conceding a goal.

After the models are implemented and probabilities are estimated, changes in those probabilities are computed as

$$\Delta P_{score} = P_s - P_{s-1},\ \Delta P_{concede} = P_s - P_{s-1} \tag{2.13}$$

where P_s describes the probability of scoring or conceding a goal in state S and P_{s-1} in the previous one. A value V of an action is computed as the difference between changes in scoring and conceding probabilities. Similarly to the values in Expected Threat, the players can be ranked and evaluated based on the cumulative value of all the actions they performed and their average impact on the game per 90 minutes.

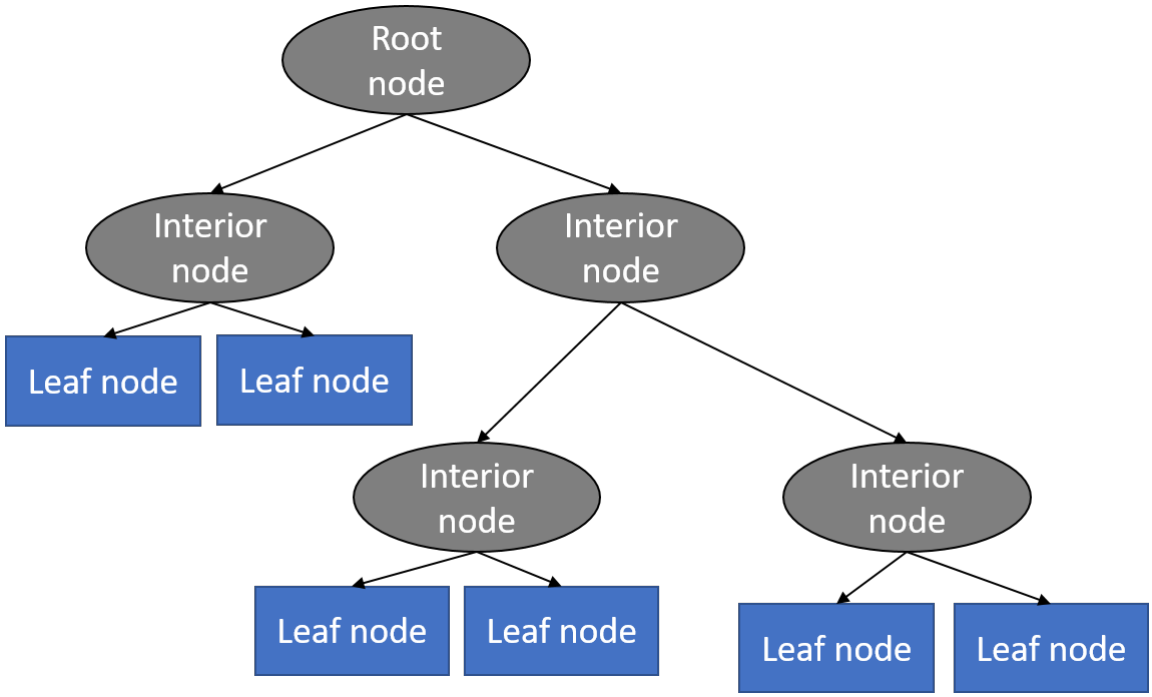
[20] Singh, K. (2019) *Introducing expected threat* Retrieved from https://karun.in/blog/expected-threat.html (Accessed: 1 March 2021)

[21] Van Roy, M., Robberechts, P., Decroos, T., Davis, J. 2020. *Valuing On-the-Ball Actions in Soccer: A Critical Comparison of xT and VAEP*, AAAI-20 Workshop on Artificial Intelligence in Team Sports.

The authors did not specify the machine learning concepts that have to be used to implement the VAEP model. Their research used multiple algorithms including logistic regression, random forest and multiple gradient tree boosting algorithms[22].

To explain what random forest is, at first, the classification tree concept needs to be introduced. A decision tree classifier is a model allowing to classify objects based on a sequence of answers. Using the training data set, the algorithm uses multiple questions in order to classify test sample. Basing on the chosen decision algorithm, a tree prune is constructed and the data are split using the feature that has the biggest information gain. Then, the procedure is repeated in each constructed in that way node as long as only leaf modes are left. Figure 2.4 presents the scheme of a classification tree.

Figure 2.4: Classification Tree Scheme



Source: https://medium.com/analytics-vidhya/a-deep-dive-to-decision-trees-6575e016d656)

Random forest is a bagging classifier built from multiple binary classification trees. The trees are created using a bootstrap sampling technique. Worth mentioning is that not all of the features are used to build those trees. Then, tree classification results are summed up and the label is assigned to a random forest in the majority voting procedure, which chooses the class that was the most frequent among decisions tree outcomes[23].

[22]Decroos, T., Bransen, L., Van Haaren, J. Davis, J. 2019. *Actions Speak Louder than Goals: Valuing Player Actions in Soccer*, Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, p. 1851–1861.

[23]Raschka, S., Mirjalili, V. 2017. *Python. Machine Learning. 2nd edition*, Packt Publishing, Birmingham.

Boosting is also a bagging technique, but instead of majority voting, it takes an iterative approach. At first, a single classification tree model is trained. Then, as long as no corrections can be made, an additional tree is trained to correct the error. Gradient boosting is a particular case of boosting, minimizing the errors using gradient descent algorithm[24]. Gradient descent is an optimization algorithm allowing to find the local minimum of a function. The values of a function are iteratively minimized in the direction opposite to gradient as it is the steepest direction[25].

The description of models used to evaluate football players contributes to better understanding of each model. Knowing how every value is computed one can understand how the achieved score can be interpreted and find the best players depending on which footballer's attribute is of the highest importance.

[24]Morde, V 2018. *XGBoost Algorithm: Long May She Reign!*, Retrieved from `https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d`. (Accessed: 9 March 2021).

[25]Chong, E. P. K, Żak, S. H., 2001. *An introduction to Optimization (2^{nd} edition)*, Josh Wiley & Sons Inc., New York.

Chapter 3

Finding the Best Players of the 2018 FIFA World Cup

Although multiple player evaluation models were built, it is not apparent which one is the most adequate to find the best performing players. To find out which one I should use to build my team of the tournament, I made calculations finding the best players using each previously described one.

3.1. Data Set Used

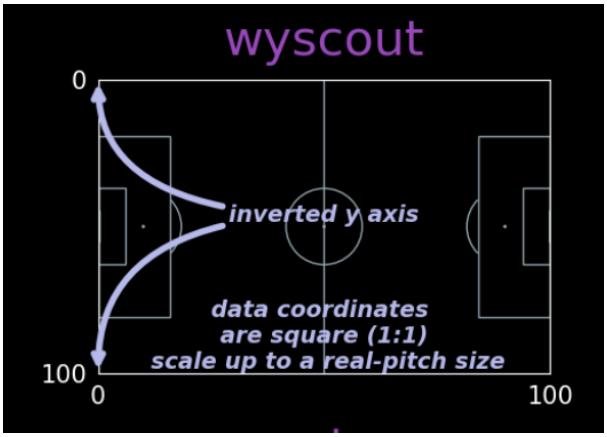
In my research I used the WyScout data set made available to the public by L. Pappalarado, P. Cintia and A. Rossi (2019). The data were collected using special software called trigger by professional video analysts[1]. It consists of 19 separate *.json* files, which can be divided into the following categories:

1. Events - 7 data sets containing all of the events that happened on the pitch in 2017/18 season in 5 biggest European leagues, that is, English Premier League (634150 observations), German Bundesliga (634150 observations), Spanish La Liga (519407 observations), Italian Serie A (647372 observations) and French Ligue 1 (632807 observations) as well as events from 2018 FIFA World Cup (101759 observations) and UEFA Euro 2016 (78140 observations). For each observation an original key (*id*) was created. Those data sets provided information about the time the event occurred (*eventSec*), (*positions*), player, who performed the action (*playerId*) and his team (*teamId*), the match and the half during when it happened (*matchId*, *matchPeriod*) as well as the event type (*eventId*, *eventName*) precised by *subEventId* and *subEventName* together with *tags*. The first two provide additional information about the free kick type, foul type, pass type, duel type or the way

[1] Pappalardo, L., Cintia, P., Rossi, A. et al. (2019) *A Public Data Set of Spatio-Temporal Match Events in Soccer Competitions.* Nature Scientific Data, vol. 6, no. 236.

the ball was lost by a team whereas the last one provides additional information on the event. For the purpose of my research 2 *tags* were of high importance - 403 for headers and 1801 for accurate passes. Moreover, this data sets provided information about event's start and end location in 100x100 Euclidean space as presented in Figure 3.1,

Figure 3.1: WyScout Pitch Coordinates



Source: https://mplsoccer.readthedocs.io

2. Matches - 7 data set containing details about results of the 1941 games from 2017/18 seasons in 5 biggest leagues, 2018 FIFA World Cup and UEFA Euro 2016. It provides information about competition (*competitionId*), season (*seasonId*), round (*roundId*) and tournament group (*groupName*), referees (*referees*), duration (*duration*), date (*date*), teams and their score (*teamsData*, *label*, *winner*), location (*venue*), each with an unique key (*wyId*),
3. Players - a data set of 3603 players who participated in the above-mentioned competitions. It provided information about player's name (*firstName*, *middleName*, *lastName*, *shortName*), nationality (*passportArea*, *birthArea*), position presented as GK - goalkeeper, DF - defender, MF - midfielder, FW - forward (*role*), current club and national team (*currentTeamId*, *currentNationalTeamId*), his stronger foot (*shortName*), date of birth (*birthDate*), as well as height and weight (*height*, *weight*),
4. Coaches - a data set of 208 football coaches with information about their name (*firstName*, *middleName*, *lastName*), nationality (*passportArea*, *birthArea*), nationality (*passportArea*, *birthArea*), date of birth (*birthDate*), coached team (*currentTeamId*) and their individual key (*wyId*),
5. Competitions - a data set providing information concerning competition name (*name)*, its format - a league or a cup (*format*), international or domestic character of a competition (*type*), location (*area*) and unique key (*wyId*),
6. Referees - a database of football referees with data on their name (*firstName*, *middleName*, *lastName*), nationality (*passportArea*, *birthArea*) and their unique key (*wyId*),

7. Teams - a data set containing information about 142 teams that took part in the competitions. name (*officialName*), *fullName*), location (*area*, *city*) and a variable stating whether team is a club or a national one (*type*) and its unique key (*wyId*).

3.2. Calculation Results

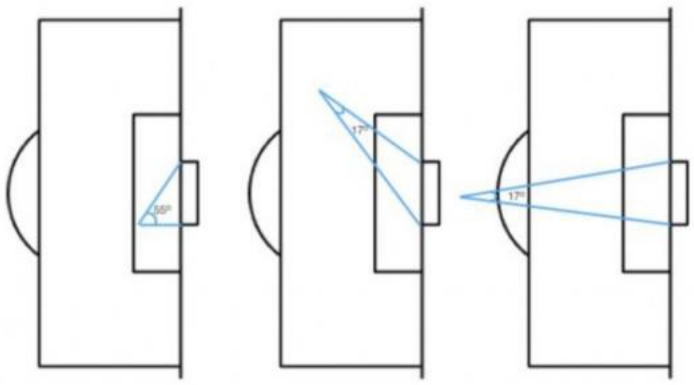
3.2.1. Expected Goals

To calculate the cumulative Expected Goals value for each player throughout the tournament, I used the WyScout data and filtered shots that happen in regulation or extra time. At first, I had to change the pitch dimensions from Wyscout 100x100 to ones corresponding to the size of football pitch - 105x68. Then, I had to invert the x coordinate, as the 0 value for x coordinates is the goal the team defends. Afterwards, I calculated the value c as the distance from a point to the vertical line through the middle of the pitch. Thus, I was able to compute the distance to the goal using Pythagoras's theorem.

I computed the angle between lines connecting the location with the goal post as presented in Figure 3.2. To do so, I used the equation presented by D. Sumpter (2017) to calculate the Expected Goals values[2]

$$\tan(Angle_i) = \frac{7.32}{x^2 + c^2 - (\frac{7.32}{2})^2}. \tag{3.1}$$

Figure 3.2: Angle Used in Expected Goals Calculations



Source: https://soccermatics.medium.com/the-geometry-of-shooting-ae7a67fdf760

[2]Sumpter, D. (2017) *The Geometry of Shooting*, Medium, Retrieved from `https://soccermatics.medium.com/the-geometry-of-shooting-ae7a67fdf760`. (Accessed: 5 March 2021).

Then, I divided the observations into 3 groups - penalties, headers and rest of the shots. I assigned to penalties the Expected Threat value of 0.8 and for the other ones I built a logistic regression models with distance and angle as independent variables using *statsmodels* package in *Python*. I used the data from 5 biggest European leagues. The histograms and descriptive statistics of variables used, that is distance to the goal and the angle, are presented in the Attachment 1. The majority of shots in the 5 most significant European leagues took place between 0 and 40 meters from the opponent's goal. The angle was within 0 and 1.5 radians with strong positive skewness. In the case of headers, the majority of them were located within 20 meters to the net. The angle was in the range from 0 up to 2 radians. Then, I estimated the model parameters. Normal shots take place further from the opponents goals than headers, but the angle between the ball's position and goal posts is lower. There are singular outliers taken about 100 meters from the opponent's net, but they are data collecting error. For example, a goal kick 103.95 meters away from opponent's net was optically counted as a shot, because the ball went from the team's goalkeeper directly into the hands of opposing one. The same applies to the headers as it is impossible to make a 100 meters long shot with ones head. Then, I estimated the logistic regression parameters, which are presented in Table 3.1.

Table 3.1: Logistic Regression Parameters

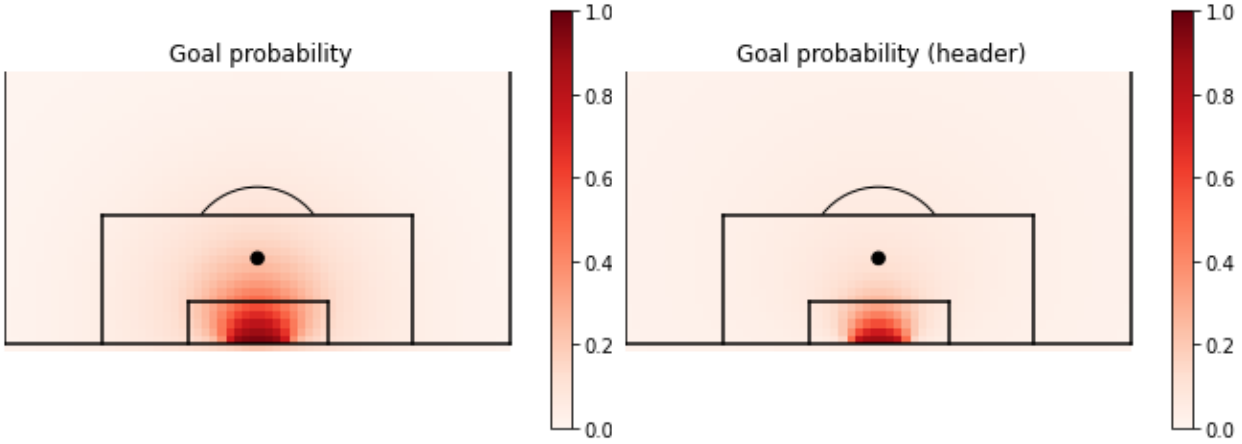
	Shots	Headers
Intercept	-1.27	-3.02
Distance	-0.09	-0.03
Angle	1.66	1.95

Source: own elaboration, Python (pandas, numpy, statsmodels packages)

Afterwards, I calculated the probability of scoring the goal from each location on the pitch from both shots and headers. The results are presented in Figure 3.3.

Figure 3.3: Goal Probability for Shots (a) and Headers (b)

(a) Goal Probability for Shots **(b)** Goal Probability for Headers



Source: own elaboration, Python (matplotlib, FCPython, pandas, numpy, statsmodels packages)

I estimated the probability of scoring a goal from each shot throughout the 2018 FIFA World Cup. To do so, I used the WyScout data from the 2018 FIFA World Cup. I compute the values of necessary variables, that is the distance to the goal and the above-mentioned angle, to estimate the Expected Threat value. The histograms and descriptive statistics are presented in the Attachment 1. Histograms have similar shapes as in the case of shots from the 5 biggest European Leagues. During the World Cup shots took place in average 0.8m closer to the goal. Headers were located slightly further away than in the club season, about 0.2m further from the goal. Then, I grouped the shots by *playerId*, summed them up and calculated the cumulative Expected Goals during the event. The results are presented in Table 3.2.

Table 3.2: Players with the Highest Cumulative Expected Goals During the 2018 FIFA World Cup

Rank	Name	Nationality	Position	xG
1	H. Kane	England	FW	4.12
2	Neymar	Brazil	GK	3.36
3	A. Griezmann	France	FW	3.27
4	M. Berg	Sweden	FW	2.94
5	E. Cavani	Uruguay	FW	2.51
6	C. Ronaldo	Portugal	FW	2.49
7	I. Perisić	Croatia	MF	2.31
8	E. Hazard	Belgium	MD	2.23
9	M. Mandzukić	Croatia	DF	2.21
10	R. Lukaku	Belgium	FW	2.05

Source: own elaboration, Python (pandas, numpy, statsmodels packages)

According to cumulative Expected Goals, Harry Kane, the top goalscorer of the event, was the best player. Evaluating players using cumulative xG statistic allowed me to rank players based only on the positions they were able to elaborate, not their ability to score goals from a given situation. If a player has higher xG than the number of goals scored, it signifies that he scored fewer goals than was expected. For example, M. Berg, ranked 4^{th} based on this statistic did not manage to score any goal during the World Cup. Moreover, players, who played the most minutes during the event have better xG as they had more time to get more shots. Therefore, I found the diagram comparing expected goals per 90 minutes and goals scored per 90 minutes adequate to make conclusions.

To calculate the number of minutes played throughout the tournament, I used the $.h5$ file prepared to convert all the events in the WyScout database to SPADL language used in the following section concerning evaluating players on their VAEP rating. However, using only *socceraction* package, I miscalculated the number of minutes played as the negative number was assigned to the ones who entered the pitch in the extra time. Moreover, the package did not manage to add 30 minutes if overtime was needed and a player was not substituted. Therefore, I had to make several changes for the games, which needed additional 30 minutes:

- added 30 minutes to the total number of minutes played for players, who played whole 120 minutes in the Croatia - England semifinal or entered the pitch during the regular time - D. Subasić, S. Vralijko, D. Lovren, D. Vida, I. Rakitić, M. Brozović, I. Perisić, J. Pickford, J. Stones, H. Maguire, D. Alli, J. Lingard, K. Trippier, H. Kane, D. Rose, M.

Rashford[3],

- added 26 minutes to the total number of minutes played for players, who entered the pitch in the extra time during the Croatia - England semifinal - V. Corluka, J. Pivarić, M. Badelj, A. Kramarić, E. Dier, J. Vardy[4]
- added 30 minutes to the total number of minutes played for players, who played whole 120 minutes in Croatia - Russia quarterfinal or entered the pitch during the regular time - I. Akinfeev, M. Fernandes, I. Kutepov, S. Ignashevich, F. Kudriashov, D. Kuziaev, R. Zobnin, I. Gazinsky, A. Erokhin, F. Smolov, D. Subasić, D. Lovren, D. Vida, I. Rakitić, L. Modrić, M. Mandzukić, A. Rebić, J. Pivarić, M. Kovacić, M. Brozović,[5],
- added 25 minutes to the total number of minutes played for players, who entered the pitch in extra time during Croatia - Russia quarterfinal - A. Dzagoev, V. Corluka[6],
- added 30 minutes to the total number of minutes played for players, who played whole 120 minutes in the Columbia - England round of 16 or entered the pitch during the regular time - D. Ospina, Y. Mina, J. Mojica, D. Sanchez, W. Barrios, J. Cuadrado, R. Falcao, M. Uribe, C. Bacca, L. Muriel, J. Pickford, J. Stones, H. Maguire, K. Trippier, J. Lingard, J. Henderson, H. Kane, E. Dier, J. Vardy[7],
- added 24 minutes to the total number of minutes played for players, who entered the pitch in extra time during the Colombia - England round of 16 - C. Zapata, D. Rose, M. Rashford[8],
- added 30 minutes to the total number of minutes played for players, who played whole 120 minutes in the Columbia - Denmark round of 16 or entered the pitch during the regular time - D. Subasić, S. Vrsaljko, D. Lovren, D. Vida, I. Rakitić, L. Modrić, A. Rebić, J. Pivarić, M. Kovacić, K. Schmeichel, S. Kjaer, J. Knudsen, M. Jorgensen, H. Dalsgaard, C. Eriksen, Y. Poulsen, L. Schone, N. Jorgensen[9],
- added 27 minutes to the total number of minutes played for players, who entered the pitch in the extra time during the Croatia - Denmark round of 16 - M. Badelj, A. Kramarić, M. Krohn-Dehli, P. Sisto[10],
- added 30 minutes to the total number of minutes played for players, who played whole

[3]FIFA. (2018) *Matches*, Retrieved from `https://www.fifa.com/worldcup/archive/russia2018/matches/match/300331522/match-lineups` (Accessed: 3 March 2021).

[4]*Ibidem.*

[5]FIFA. (2018) *Matches*, Retrieved from `https://www.fifa.com/worldcup/archive/russia2018/matches/match/300331504/match-lineups` (Accessed:3 March 2021).

[6]*Ibidem.*

[7]FIFA. (2018) *Matches*, Retrieved from `https://www.fifa.com/worldcup/archive/russia2018/matches/match/300331542/match-lineups` (Accessed:3 March 2021).

[8]*Ibidem.*

[9]FIFA. (2018) *Matches*, Retrieved from `https://www.fifa.com/worldcup/archive/russia2018/matches/match/300331498/match-lineups` (Accessed:3 March 2021).

[10]*Ibidem.*

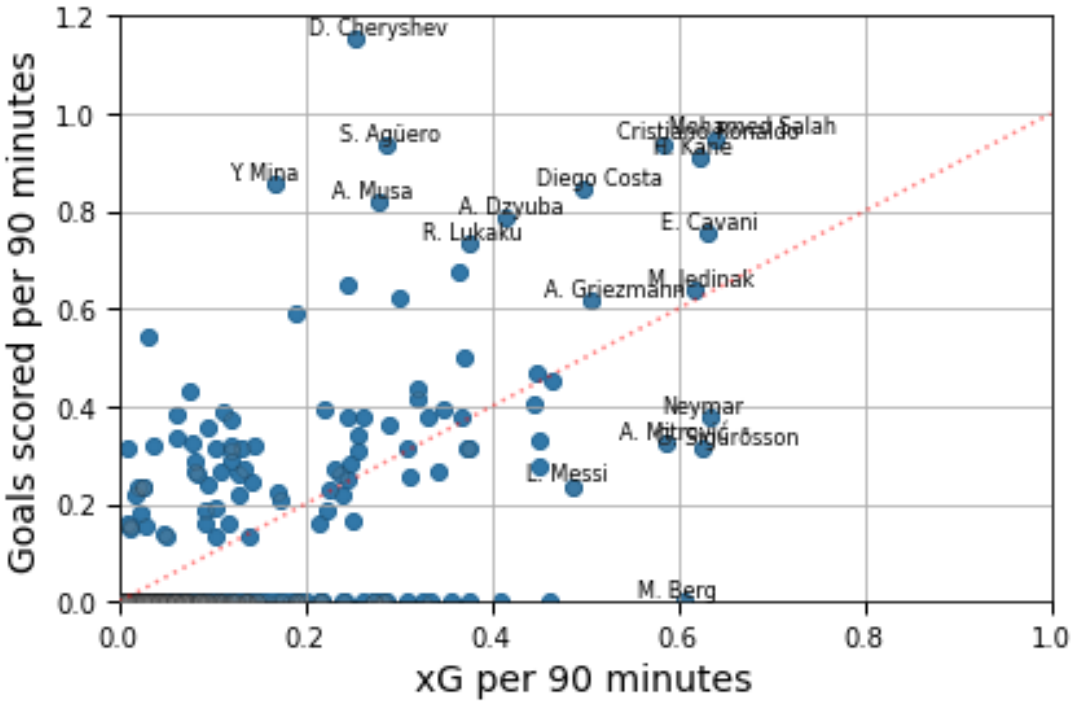
120 minutes in Spain - Russia round of 16 or entered the pitch during the regular time - D. de Gea, G. Pique, S. Ramos, J. Alba, S. Busquets, Koke, Isco, D. Carvajal, A. Iniesta, I. Aspas, I. Akinfeev, M. Fernandes, I. Kutepov, S. Ignashevich, F. Kudriashov, R. Zobnin, A. Golovin, V. Granat, D. Cheryshev, F. Smolov[11],

- added 26 minutes to the total number of minutes played for players, who entered the pitch in the extra time during the Spain - Russia round of 16 - Rodrigo, A. Erokhin[12].

I decided to leave only players who played more than 150 minutes during the World Cup. It allowed me to filter out outliers - players who entered the pitch only for 15 minutes and scored a goal from a difficult position. Setting the threshold to 150 minutes is adequate for players who played a whole game and got substituted during the second. Therefore I concluded that they had enough time to make an impact on the pitch.

I evaluated players based on their xG per 90 and goals scored per 90. Such an approach allowed me to find out players who scored goals from difficult positions as well as those who missed good situations. To do so, I constructed a diagram with xG per 90 on the x-axis and goals scored per 90 on the y-axis. Then, I drew a line for a player who scored exactly the same number of goals as was expected. The higher the player is above the line, the better finisher. Respectively, the lower, the worse. The diagram is presented in Figure 3.4.

Figure 3.4: xG per 90 and Goals Scored per 90



Source: own elaboration, Python (matplotlib, pandas, numpy packages)

[11] FIFA. (2018) *Matches*, Retrieved from `https://www.fifa.com/worldcup/archive/russia2018/matches/match/300331517/match-lineups` (Accessed:3 March 2021).

[12] *Ibidem.*

From the diagram one can deduce that D. Cheryshev was the best finisher of the FIFA 2018 World Cup, as he is the highest above the line. He managed to score 4 goals while his cumulative Expected Goals value was below 1. According to this statistic, the best players belong as well S. Aguero, Y. Mina and A. Musa. Interestingly, Lionel Messi and Neymar, considered one of the world's best footballers, had below-average scores.

The Expected Goals model allows only to measure an evaluate players based on their shot conversion. Player's impact goes beyond scoring a goal. Creating opportunities for teammates or making necessary defensive actions can be even more valuable to the final result than scoring goals. Moreover, the individual xG statistic is higher for players who are responsible for taking penalties. Therefore, those who are chosen to take a penalty kick have higher scores even though they have not contributed to getting one. However, this statistic has a wide range of football applications and is easy to understand for every fan.

3.2.2. Real Adjusted Plus-Minus

To calculate the Real Adjusted Plus-Minus for 2018 FIFA World Cup participants, I created a data set in Microsoft Excel. Based on official website[13], I collected the names of all players called to their national teams. Using official game reports[14], I applied adequate independent variables based on players' presence or absence on the pitch and the team they played for. Since the 2018 FIFA World Cup took place from 14^{th} June 2018 to 15^{th} July 2018, I decided to set the parameter t specifying the age of operations to 0, as the player's performance should not differ drastically over that short period of time. Therefore, the independent variables adopted a value -1, 0 or 1. Having parameter t = 0, I calculated the dependent variable's value using the equation (2.5) for each game period when no substitution happened. Thus, the database consisted of 736 columns corresponding to 736 players that took part in the event, calculated dependent variable and 398 observations corresponding to the number of periods without substitutions. An excerpt from the database is presented in Figure 3.5.

[13]FIFA. (2018) *Teams*, Retrieved from `https://www.fifa.com/worldcup/archive/russia2018/teams/`. (Accessed: 28 February 2021).

[14]FIFA. (2018) *Matches*, Retrieved from `https://www.fifa.com/worldcup/archive/russia2018/matches/`. (Accessed: 28 February 2021).

Figure 3.5: Excerpt of the Database

y	El Hadary	Gabr	Elmohamady	Gaber	Morsy	Hegazi	Fathy	Hamed	Mohsen	Salah
3,75										
2,25										
0										
45										
0										
20										
0		1				1	1	1	1	
0		1			1	1	1		1	
0		1			1	1	1		1	
0		1			1	1	1		1	
0		1			1	1	1			
0		1			1	1	1			
-0,166666667		1			1	1	1			
4,21875		-1				-1	-1	-1	-1	-1
0		-1				-1	-1	-1	-1	-1
-18		-1				-1	-1	-1	-1	-1

Source: own elaboration, Microsoft Excel

Then I conducted the ridge regression in *Python* using the *sklearn* package, setting the lambda parameter to 3000 as it was optimised by Hvatten and Saebo (2019)[15]. After I calculated the coefficients, I sorted them and created the ranking of players with the highest Real Adjusted Plus Minus presented in Table 3.3.

Table 3.3: Players with the Highest Real Adjusted Plus-Minus During the 2018 FIFA World Cup

Rank	Name	Nationality	Position	RAPM
1	Fagner	Brazil	DF	0.0227
2	Alisson	Brazil	GK	0.022
3	Miranda	Brazil	DF	0.022
4	T. Silva	Brazil	DF	0.022
5	Neymar	Brazil	FW	0.022
6	Casemiro	Brazil	MD	0.0213
7	T. Courtois	Belgium	GK	0.019
8	I. Rakitić	Croatia	MD	0.018
9	T. Meunier	Belgium	DF	0.018
10	R. Firmino	Brazil	FW	0.018

Source: own elaboration, Python (pandas, sklearn packages)

[15]Hvattum, L. M., Saebo, O. (2019) *Modelling the Financial Contribution of Soccer Players to Their Clubs*, Journal of Sports Analytics, vol. 5, p. 23–34.

There are several misconceptions when using this model. First of all, my calculations have shown that the model is still influenced by the team the player plays for. The highest Real Adjusted Plus-Minus value was achieved by Brazilians. Moreover, the model does not consider actions on the field, but only the difference between goals scored and allowed. Nevertheless, substitutions in football occur rarely comparing to basketball. During a football match only 3 substitutions per team are allowed, with a possibility of an additional one in the extra time if needed. Therefore, there are maximum 7 periods without substitution in one game without extra time, 9 if additional 30 minutes are necessary[16]. Nevertheless, for the matter of winning the game of football, the number of goals is the essential factor. Assessing players' performance using Real Adjusted Plus-Minus takes into consideration the duration of period without substitution. Therefore, 6 goals scored by England against Panama in 63 minutes have a lower value than 1 goal scored by Russia against Saudi Arabia in 2 minutes. As changes in football occur mostly in the second half, goals scored then have bigger influence on players' RAPM, because the D_i factor is lower. Last, but not the least, since the World Cup is played in only one country, the influence of playing at home is negligible.

3.2.3. Expected Threat

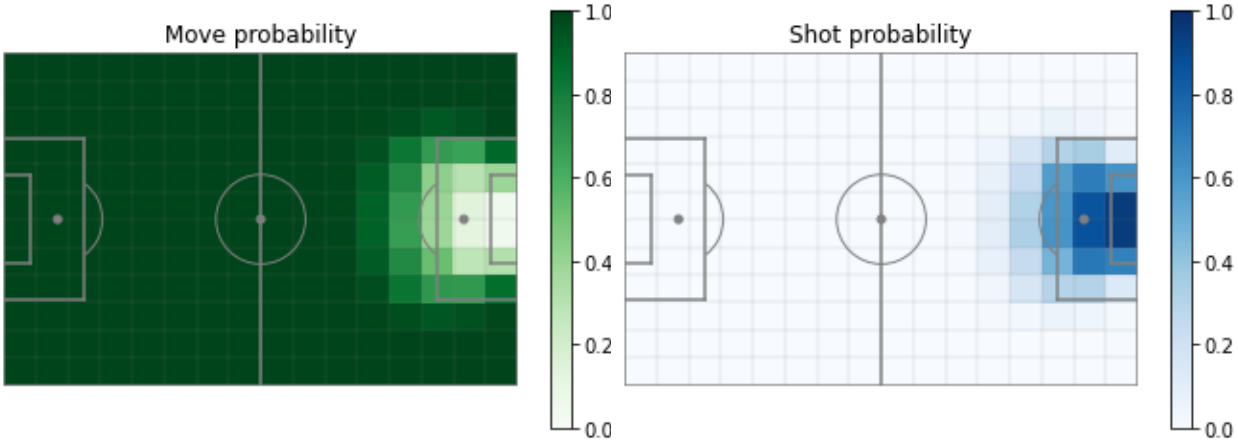
Towards calculating the values from Expected Threat model, I chose the WyScout data set consisting of all the events from the 2017/18 Premier League season. Then, I filtered the actions other than shots or passes, and ones from set pieces. I left the observations which *subEventName* was one of *'Shot', 'Free kick shot', 'Simple pass', 'High pass', 'Throw in', 'Head pass', 'Smart pass', 'Cross', 'Free kick cross', 'Corner'*. I considered actions labelled *'Shot', 'Free kick shot'* as the ones which were shots and the remainder as the ones moving the ball. Given the starting and final x and y coordinates of each action, I assigned a number from 1 to 192 corresponding to 192 zones K. Singh has divided the pitch into[17]. For shots, I decided to assign a value S as the ending location of an action. I calculated the number of shots and passes in each zone as well as their ends. Then, I estimated the shot and move probabilities for each region as the quotient of an adequate number of shots or actions moving the ball and the number of actions in the given area. The values are presented in the form of 2D histogram in Figure 3.6.

[16]Hvattum, L. M., Saebo, O. (2019) *Modelling the Financial Contribution of Soccer Players to Their Clubs*, Journal of Sports Analytics, vol. 5, p. 23–34.

[17]Singh, K. (2019) *Introducing Expected Threat* Retrieved from `https://karun.in/blog/expected-threat.html` (Accessed: 1 March 2021).

Figure 3.6: Move Probability (a) and Shot Probability (b) for Each Zone (attacking from left to right)

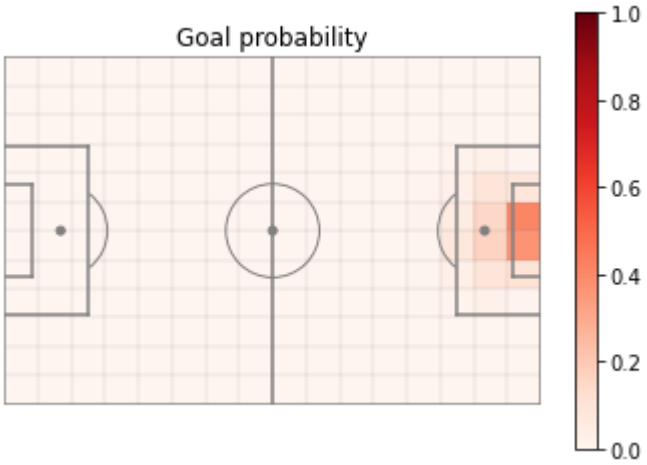
(a) Move Probability for Each Zone **(b)** Shot Probability for Each Zone



Source: own elaboration, Python (matplotlib, FCPython, pandas, numpy, packages)

I calculated the goal probability for each sector as the number of shots in a zone divided by the number of shots from that region. This probabilities are presented in Figure 3.7.

Figure 3.7: Goal Probability for Each Zone (attacking from left to right)



Source: own elaboration, Python (matplotlib, pandas, numpy packages)

Afterwards, I built a transition matrix M between zones estimating the probabilities using undermentioned equation[18] (3.2)

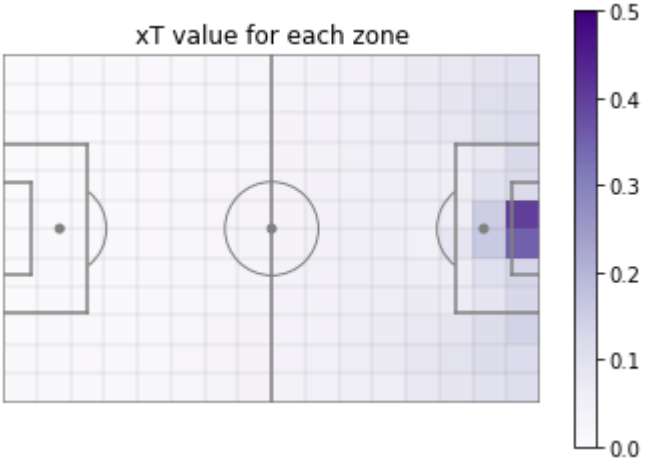
$$M_{ij} = \frac{S_{ij}}{S_i} \tag{3.2}$$

where S_{ij} signifies the number of action moving the ball from zone i to zone j and S_i the total number of moves which occurred in zone i. Later, I calculated the Expected Threat value for

[18] Sherlaw-Johnson, C., Gallivan, S., Burridge, J. (1995) *Estimating a Markov Transition Matrix from Observational Data*, The Journal of the Operational Research Society, vol. 46, no. 3, p. 405–410.

each zone. I decided to run 5 iterations to estimate those measures. The values are presented in Figure 3.8.

Figure 3.8: Expected Threat Value for Each Zone (attacking from left to right)



Source: own elaboration, Python (matplotlib, pandas, numpy packages)

To evaluate players' performance during the 2018 FIFA World Cup, I used WyScout data set consisting of all events on the pitch during that event. Then I left only the actions which moved the ball and assigned a number adequate to each zone. Having the start and end area of each action, I matched the Expected Threat value to both the beginning and the finish and calculated the difference in Expected Threat.

Worth mentioning is the fact that in order to do so, I developed the basic Expected Threat model. In the primary model, only the successful actions were considered[19]. Therefore, a player was awarded a minus value for passing the ball backwards, but none value was assigned when he lost the possession. Consequently, for the Expected Threat's cumulative change, it was more beneficial to lose the ball than to pass it to the teammate. Therefore, in my calculations, I used both successful and unsuccessful actions assigning the value 0 for the ending zone if a player lost possession.

Subsequently, I grouped the events by *playerId*, summed them up and merged them with the WyScout player database on this key. Hence, I obtained the cumulative difference in Expected Threat for each player and ranked them based on this statistic. The results are presented in Table 3.4.

[19]Singh, K. (2019) *Introducing Expected Threat* Retrieved from `https://karun.in/blog/expected-threat.html` (Accessed: 1 March 2021).

Table 3.4: Players with the Highest Cumulative Change in Expected Threat During the 2018 FIFA World Cup

Rank	Name	Nationality	Position	xT
1	S. Ramos	Spain	DF	3.66
2	J. Stones	England	DF	2.55
3	T. Kroos	Germany	MF	2.21
4	G. Pique	Spain	DF	2.00
5	Miranda	Brazil	DF	1.83
6	R. Varane	France	DF	1.80
7	J. Boateng	Germany	DF	1.68
8	V. Kompany	Belgium	DF	1.48
9	I. Rakitić	Croatia	MF	1.45
10	J. Bednarek	Poland	DF	1.40

Source: own elaboration, Python (pandas, numpy packages)

Such an approach is biased by the number of minutes the player played. This number can differ significantly between players who played only in the group stage because their team was eliminated and were only able to play 270 minutes. The player, who was on the pitch the longest throughout the tournament, was J. Pickford, who played 718 minutes. Therefore, such a difference is significant because World Cup participants who managed to reach the semifinals with their teams had many more minutes on the pitch to contribute to their team's result positively, so their cumulative score was higher. Thus, I decided to calculate these value and compare rank players based on this statistic as well as make conclusions concerning the reliability of this model.

I calculated the number of minutes played using the same methodology as for the purpose of Expected Goals model. Without an appropriate calculation of minutes played it would be impossible to correctly compute the value per 90 minutes. Then, I was able to rank players based on their cumulative change in Expected Threat per 90 minutes. The results are presented in Table 3.5.

Table 3.5: Players with the Highest Cumulative Change in Expected Threat per 90 Minutes During the 2018 FIFA World Cup

Rank	Name	Nationality	Position	xT per90
1	S. Ramos	Spain	DF	0.80
2	J. Boateng	Germany	DF	0.79
3	T. Kroos	Germany	MF	0.69
4	J. Bednarek	Poland	DF	0.53
5	G. Pique	Spain	DF	0.44
6	M. Milligan	Australia	DF	0.44
7	M. Rojo	Argentina	DF	0.37
8	G. Shoji	Japan	DF	0.35
9	Miranda	Brazil	DF	0.34
10	J. Stones	England	DF	0.34

Source: own elaboration, Python (pandas, numpy packages)

9 out of 10 best players of FIFA 2018 World Cup, according to this statistic, are central defenders. Moreover, many of the attacking players scoring the largest number of goals throughout the event, such as K. Mbappe, D. Cheryshev or H. Kane, have a negative score in this statistic. Germany and Spain, both having 2 representatives in top5 in xT per 90 ranking, played using heavily possession based tactic. Every game played by one of these teams ended with ball possession higher than 65% for this side[20]. Therefore, the scores of players representing those nations, who made many passes, are the highest. Strikers are heavily penalised for passing the ball out of the penalty area, as the difference in Expected Threat between zones is relatively big - up to 0.3. Depending on the situation, sometimes, it was more beneficiary for the team to pass the ball out of the penalty area than to make an inefficient shot. Unfortunately, a the game context when a pass occurred was impossible to measure because tracking data from the 2018 FIFA World Cup are unavailable. On the other hand, defenders, who play nearer to their own goal and make short, not risky passes, get easy points for each pass as the action progresses. Moreover, the model does not consider other actions than passes to evaluate players. A simulation over zones, how the ball would progress during a football game, would never end as a goal or a shot, as the transition matrix consists only of transition probabilities between areas. Expected Threat is an appropriate model to measure a player's impact on action creation, but not the whole performance.

[20]FIFA. (2018) *Matches*, Retrieved from `https://www.fifa.com/worldcup/archive/russia2018/matches/`. (Accessed: 3 March 2021).

3.2.4. Valuing Actions by Estimating Probabilities

To evaluate players performance based on their VAEP ranking, I converted the available data into SPADL language using *socceraction* package in Python. I transformed the JSON files containing information about events and matches from the 5 most significant European leagues as well as the players database and saved it in $.h5$ file. Using the data in SPADL representation, I prepared a set of features for every action corresponding to 3 categories presented by T. Decroos, L. Bransen and J. Van Haaren and J.Davis (2019).

- SPADL features - action type, start and end location coordinates, body part used to take it and the time the action was played,
- context features - start and end distance to the goal of an action, the start and end angle between line connecting the nearer post with the location on the pitch and a parallel line to a goal line, total distance covered by a ball as well as the change in x and y coordinates,
- game context features - the event outcome

The above-mentioned values, their description and type are presented in Table 3.6, whereas their histograms and descriptive statistics can be found in the Attachment 2.

Table 3.6: Features Description

Feature name	Description	Type
type-name	action type	nominal
start-x	x coordinate of action's start (in meters)	continuous
end-x	x coordinate of action's end (in meters)	continuous
start-y	y coordinate of action's start (in meters)	continuous
end-y	y coordinate of action's end (in meters)	continuous
bodypart_name	body part used (in meters)	nominal
time_played	time when an action occurred (in seconds)	continuous
start_distance_to_goal	distance to the goal of action's start (in meters)	continuous
end_distance_to_goal	distance to the goal of action's end (in meters)	continuous
start_angle_to_goal	angle (computed as described above) of action's start	continuous
end_angle_to_goal	angle (computed as described above) of action's end	continuous
distance_covered	distance covered by the ball during an action	continuous
diff_x	difference in x coordinate	continuous
diff_y	difference in y coordinate	continuous
result_id	action's outcome	nominal

Source: own elaboration

I also prepared delayed features, that is, for each action mentioned above, variables describing 2 previous activities on the pitch. Then I created labels stating if a goal was scored and conceded within the following 10 actions.

From the covered to SPADL $.h5$ file, I created a data frame of games. I split the actions and outcomes data set into train and test data set with the features and labels prepared. The first one contained events from 5 most significant European Leagues and the second one from the 2018 FIFA World Cup. Then, I applied a gradient boosting XGBoost (Extreme Gradient Boosting) algorithm using the training data. I chose this method because multiple hardware optimisation techniques used in the *xgboost* package in Python allowed me to fit two classifiers from 2 million observations and 45 independent variables within a reasonable amount of time (about 20 minutes). This method is also considered one of the most commonly used frameworks in Machine Learning projects[21]. I chose the number of trees in the XGBoost model to be 100 and its maximal depth of a tree to be 4 as it was optimised using the grid search algorithm by authors of the VAEP model in series of *socceraction* package tutorials[22]. Grid search is an algorithm that searches through a given set of parameters for the one which performs the best according to the manually set scoring technique. In this model's case, the objective of grid search was to minimise the Brier score.

Brier score is a metric that measures the accuracy of probabilistic predictions. For classification models, a class is assigned depending on the threshold set. Usually, it is 0.5. If a probability of an occurrence exceeds 0.5, class one is assigned to observation, otherwise 0. Brier score is a statistic that allows finding models with the most accurate predictions. It is calculated as

$$BS = \frac{1}{n}\sum_{i=1}^{n}(p_i - o_i)^2 \tag{3.3}$$

where p_i is the probability of an occurrence and o_i is the observation's class. The lower the score is, the better the model estimates probabilities[23]. This metric in grid search optimisation allowed the VAEP creators to find a model that commits the lowest error, calculating the probability. After I built scoring and conceding models, I evaluated them based on the Brier score. For the first model, it amounted to 0.012 and for the second to 0.005, which are a very good scores. I also drew a calibration curve for both models. I evaluated them also using calibration curves. A calibration curve is a reliability diagram of a model. On the x axis, the observed fractions of observations are located and on the y axis, the predicted fraction of positives. A perfectly

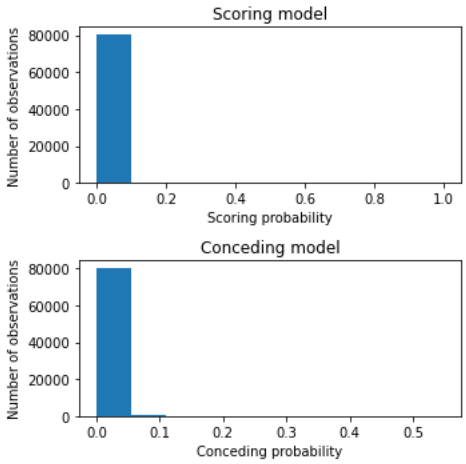
[21] Morde, V. (2018) *XGBoost Algorithm: Long May She Reign!*, Towards Data Science, Retrieved from `https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d`. (Accessed: 9 March 2021).

[22] DTAI KU Leuven (2020) *Socceraction Documentation*, Retrieved from `https://socceraction.readthedocs.io/en/latest/` (Accessed:18 February 2021).

[23] El Houndm, A. (2021) *Brier Score: Understanding Model Calibration*, Neptune Blog, Retrieved from `https://neptune.ai/blog/brier-score-and-model-calibration` (Accessed:12 March 2021).

straight line with slope = 1 would indicate perfect classifications for all probabilities[24]. They are not perfectly calibrated in the case of scoring and conceding models but still valid. The majority of observations have scoring and conceding probabilities lower than 0.1 as presented in Figure 3.9.

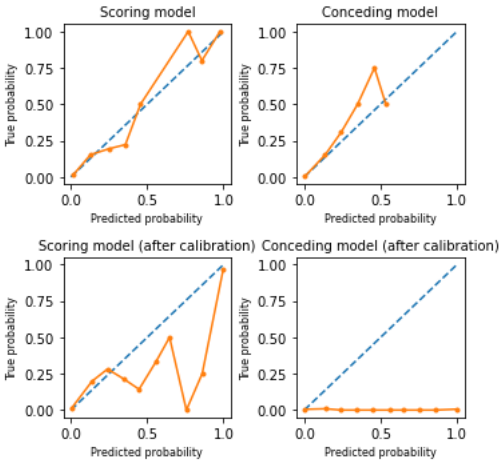
Figure 3.9: Histogram of Scoring and Conceding Probabilities



Source: own elaboration, Python (matplotlib, pandas, xgboost, socceraction packages)

Therefore the calibration of the beginning of the plot is of high importance. In the case of my curves, the beginning is nearly perfectly calibrated for both models. I tried to overcome the problem of worse calibrations for higher predicted probabilities using the calibration algorithm implemented in *sklearn* package, but it did not help with the model calibration. Calibration curves before and after calibration procedure is presented in Figure 3.10.

Figure 3.10: Calibration Curves



Source: own elaboration, Python (matplotlib, sklearn, xgboost, socceraction packages)

[24] Weishaimer, A., Palmer, T.N. (2014) *On the Reliability of Seasonal Climate Forecasts*, Journal of The Royal Society Interface, vol. 11, no. 96.

Then, using these models, I estimated the probability of scoring and conceding a goal within the next 10 actions and calculated the difference in both scoring and conceding probabilities for each action that occurred during the 2018. Later, I subtracted those values getting the value of every action. Having the value of each action I grouped them by *playerId* and summed up to get the ranking based on cumulative VAEP. The results are presented in Table 3.7.

Table 3.7: Players with the Highest Cumulative VAEP During the 2018 FIFA World Cup

Rank	Name	Nationality	Position	VAEP
1	K. Mbappe	France	FW	4.59
2	H. Kane	England	FW	3.48
3	D. Cheryshev	Russia	MF	3.24
4	Y. Mina	Colombia	DF	3.17
5	K. Trippier	England	DF	2.82
6	Neymar	Brazil	FW	2.82
7	E. Hazard	Belgium	DF	2.76
8	R. Lukaku	Belgium	FW	2.72
9	P. Coutinho	Brazil	MF	2.62
10	E. Cavani	Uruguay	FW	2.55

Source: own elaboration, Python (pandas, xgboost, socceraction packages)

As mentioned before, player comparison based on the cumulative score promotes competitors, who played more minutes, as they had more time to make valuable actions. Therefore, as in my Expected Goals and Expected Threat calculations, I calculated the value per 90 minutes using the same method as in the previous models and ranked players based on this statistic. The best players are presented in Table 3.8.

Table 3.8: Players with the Highest VAEP per 90 During the 2018 FIFA World Cup

Rank	Name	Nationality	Position	VAEP per90
1	D. Cheryshev	Russia	MF	0.94
2	Y. Mina	Colombia	DF	0.91
3	S. Aguero	Argentina	FW	0.89
4	A. Musa	Nigeria	FW	0.80
5	K. Mbappe	France	FW	0.74
6	M. Wague	Senegal	DF	0.70
7	E. Cavani	Uruguay	FW	0.64
8	A. Carrillo	Peru	MF	0.64
9	Diego Costa	Spain	FW	0.64
10	T. Kroos	Germany	MF	0.63

Source: own elaboration, Python (pandas, xgboost, socceraction packages)

According to the ranking, D. Cheryshev should be selected as the 2018 FIFA World Cup's best player. Interestingly, among the 10 best players, all scored at least 1 goal during the event. Moreover, the four best players are also the ones with the best xG per 90 to goals per 90 ratios. This approach is the most complex of all previous models, as it includes multiple events in the game of football, both from defensive and offensive perspectives. However, one has to remember that football is not only a game of actions but also of player movement which is not taken into consideration given the data set available. This model's downside is that the value is hard to interpret because of the use of complicated, black-box classifier as the XGBoost algorithm. In football analytics, the possibility of easily interpreting a value is of high importance, as the value must be understandable to coaches, who are usually former footballers without a solid mathematical background.

There are several misconceptions behind each model and different aspects of football are evaluated while using them. Therefore, the results of the calculations while using every model differ significantly from each other. Only one team of the tournament can be build to compare with the journalist's choices, so before building it, I have to choose the most appropriate model.

Chapter 4

Building the Data-Driven Team of the Tournament

To build the team of the 2018 FIFA World Cup, I decided to use VAEP per 90 minutes. Out of 4 different player evaluation methods, it is the most complex one, as it considers all of the actions taken by a player during a football match. If I had to find the tournament's best performing lineup, I would use the Real Adjusted Plus-Minus. The players who managed to produce a better goal outcome than was expected could be found using the Expected Goals model. The best creators have the highest score in the change in Expected Threat metric. The VAEP framework allows me to find the best overall player at each position. I consider evaluating players who played a different number of minutes throughout the tournament, based on the per 90 ratings more adequate. The results do not depend on the team strength and which stage of the competition the team managed to reach but indicate individual performance value.

4.1. Data-Driven Team of the Tournament

As I wanted to compare the data-driven team of the tournament with the journalist's choices, at first, I had to group players more precisely based on their position. My data set provides information only whether a player is a goalkeeper, defender, midfielder or a forward. I assume that a person who watched games of the 2018 FIFA World Cup would group players using more detailed description of a position. Using WyScout data I was not able to distinguish defenders between centre backs, right wing backs and left wing backs and midfielders offensive and defensive midfielders. Moreover, depending on the used tactics, wingers could be classified as either midfielders or forwards. Therefore, I used official game reports[1] with the tactical line-ups, which coaches have to provide to FIFA an hour before the game starts, and divided the play-

[1]FIFA. (2018) *Teams*, Retrieved from `https://www.fifa.com/worldcup/archive/russia2018/teams/`. (Accessed: 28 February 2021).

ers based on their position into following groups: goalkeepers (GK), centre backs (CB), right backs (RB), left backs (LB), central defensive midfielders (CDM), central offensive midfielders (CAM), left wingers (LW), right wingers (RW) and strikers (ST). The division can be found in Attachment 3. Players, who played in more than 1 position were classified into both categories, but could be used building the team of the tournament only in one position. I decided to rank players using the 4231 formation that consists of a goalkeeper, 2 centre backs, 1 right back, 1 left back, 2 central defensive midfielders, 1 central offensive midfielder, 1 left wing, 1 right wing and a striker. I chose this tactics as it was the most commonly used one throughout the event (21 out of 32 teams used this formation) as well as the World Cup winner France played using this set up. Then I searched for the players with the highest VAEP value at each position resulting in following team of the tournament as presented in Table 4.1.

Table 4.1: Data-Driven 2018 FIFA World Cup Team of the Tournament

Position	Name	Nationality	VAEP per90
GK	V. Stojković	Serbia	0.02
RB	M. Wague	Senegal	0.70
CB	Y. Mina	Colombia	0.91
CB	M. Jorgensen	Denmark	0.54
LB	A. Kolarov	Serbia	0.74
CDM	T. Kroos	Germany	0.63
CDM	G. Krychowiak	Poland	0.51
RW	K. Mbappe	France	0.75
CAM	P. Coutinho	Brazil	0.52
LW	D. Cheryshev	Russia	0.94
ST	S. Aguero	Argentina	0.89

Source: own elaboration, Python (pandas, xgboost, socceraction packages)

Many of the choices are controversial. There are no players from Croatia and Belgium, adequately silver and bronze medalist of the event. Moreover, Luka Modrić, who was chosen the best player of the 2018 FIFA World Cup[2], neither was ranked among the top 10 best players of the event according to any model nor could be found in the team of the tournament. Nevertheless, 5 out of 11 players, V. Stojković, M. Wague, A. Kolarov, T. Kroos and G. Krychowiak in the data-driven team of the tournament, did not manage to advance to the next round from the group stage. Such an unexpected team of the tournament comes from the fact that the metric per 90 minutes played was used. Therefore, players who had 3 outstanding games in the group stage could find a place among 11 best footballers of the World Cup. On the other hand, such an

[2]FIFA (2018) *Golden Consolation for Magical Modric*, Retrieved from: `https://www.fifa.com/worldcup/news/157-awards-piece-2986294` (Accessed:15 March 2021).

approach does not consider the strength of an opponent. Therefore for players, who played only 3 or 4 games against mediocre teams, it was easier to get higher VAEP ratings per 90 minutes than for ones like Croatia representatives, who had to beat the best teams in the world to get to the final.

4.2. Comparing My Team to the Journalist's Choices

Multiple newspaper and internet portals created their team of the tournament based on the football games they watched on TV or at the stadiums. I decided to choose The Guardian's one because the methodology behind it seemed more complex than just 11 names being written down as the best. The Guardian provided full 2018 FIFA World Cup coverage and gave ratings to the players after each game. Then, after the tournament ended, they constructed their team of the tournament using 4231 formation based on those previously given ratings.[3] The team is presented in Table 4.2.

Table 4.2: The Guardian's 2018 FIFA World Cup Team of the Tournament

Position	Name	Nationality
GK	T. Courtois	Belgium
RB	S. Vrsaljko	Croatia
CB	D. Vida	Croatia
CB	D. Godin	Uruguay
LB	L. Hernandez	France
CDM	L. Modrić	Croatia
CDM	K. de Bruyne	Belgium
RW	I. Perisić	Croatia
CAM	E. Hazard	Belgium
LW	D. Cheryshev	Russia
ST	K. Mbappe	France

Source: own elaboration, The Guardian

The team constructed based on an eye-test differs significantly from the data-driven one. There are none players from the teams that got eliminated in the round of 16 or in the group stage. In the case of my team of the tournament, there were 7 such participants. On the one hand, the eye-test is biased towards teams that managed to achieve the best places during the event. On the other hand, the data-driven approach does not consider either opponent's strength

[3]Christenson, M. (2018) *World Cup 2018: The Guardian Team of the Tournament*, The Guardian, Retrieved from `https://www.theguardian.com/football/2018/jul/16/world-cup-2018-the-guardian-team-of-the-tournament` (Accessed: 14 March 2021).

or the importance of the game. The use of VAEP per 90 metric puts more importance on individual performance and goal-scoring, as out of the 11 participants of the data-driven team, only the goalkeeper did not manage to score a goal during the event, whereas in the case of The Guardian's team, only 6 players hit the net. Goal scorers being in the majority of team members are understandable for strikers and midfielders, whose role on the pitch is to attack. However, the defenders' primary objective is not to allow goals instead of scoring them. Nevertheless, the choice of V. Stojković as the best goalkeeper is surprising, but it is due to the fact that goalkeepers rarely contribute towards scoring a goal. Their passes are hardly ever within ten passed leading to a goal. That is the reason why their VAEP is very low. Moreover, the eye-test was able to see player movement. Therefore, the authors of the team mentioned above were able to judge the player's performance based on his positioning. Movement plays a crucial role in football as it allows the whole team to gain an advantage over the rival. Player runs create open spaces for the teammate to pass and defenders positioning is important while covering the spaces to block it. Having no movement, there would be no transition between offence and defence. Without tracking players, one cannot take into consideration their ability to move between the opponent's formation, keeping his position or his ability to pressure the opposing defenders. Unfortunately, I could not measure their moves, as football tracking data are available only internally for the team members.

Conclusions and Recommendations

My research's main goals were finding the best performing players of the 2018 FIFA World Cup and comparing the results based on data analysis to the eye-test. I used 4 different player evaluation techniques - Expected Goals, Real Adjusted Plus Minus, Expected Threat and Value Added on Estimating Probabilities. I found out that Expected Goals is the most appropriate model to find the players who get the best chances and are capable of scoring more goals than expected. Denis Cheryshev was the player who surpassed the expected number of goals per 90 minutes played the most among the impactful players. Real Adjusted Plus Minus allowed me to find the most dangerous line-up of the tournament, consisting of Brazilian defender Fagner. It is based on the rotation in line-ups. Due to a small number of substitutions in football and their irregularity, the results in football cannot indicate the best players. The Expected Threat model allowed me to find out who was the event's best creator. Spanish defender Sergio Ramos was the one whose passes created the most significant danger to the opponents net. Valuing Actions on Estimating Probabilities appeared to be the most complex of all models. It considers all actions made by a player, but the value is hard to interpret. Once again, Denis Cheryshev was the best player of the event using this model. I considered valuing players score per 90 minutes played better than their cumulative score, as it is not biased towards the team's result.

The next objective I set in my research was to build the data-driven team of the tournament and assess how it differs from the subjective team based on an expert's observation. I built my team with the players with the highest VAEP value per 90 minutes and compared it with the team presented by The Guardian. From such a comparison, I managed to find the following conclusions:

1. Data-driven team differs significantly from the expert's choice. Only 2 players are present in both teams.
2. The data-driven team found the best players according to the impact their actions had on the situation on the pitch, whereas the eye-test was biased towards the team's final rank that players played for.
3. Eye-test was able to evaluate goalkeepers and defenders better than data, basing on their role on the pitch, that is, to help their team not to allow goals. That method allowed me to see how difficult was goalkeepers save or evaluate defenders positioning. The data-driven

approach was biased towards defensive players who scored goals during the event.

4. Due to the unavailability of tracking data from the 2018 FIFA World Cup, I could not evaluate player movement, which is an important part of football. In contrast, The Guardian journalists were able to do so. Using individual subjective player evaluation, one can consider more aspects of a football game.

The data-driven team successfully found players who performed the best during the event, which was visible in the transfer moves made by clubs. M. Wague was transferred to one of the best clubs in the world, FC Barcelona, after his outstanding performance during the event[4], Y. Mina was bought by Everton FC for an outstanding sum of 30 million euro[5] despite playing only 5 games during the 2017/18 season[6]. Valencia CF decided to invest their funds in Denis Cheryshev despite him not being a starter for the weaker Villareal CF side in 2017/18 season[7]. Moreover, after the amazing performance during 2018 FIFA World Cup experts claimed K. Mbappe to be the next best player in the world after C. Ronaldo and L. Messi[8].

There is no doubt that data science helps to evaluate player performance, but there is still some room for improvement left:

1. The positional aspects of game from the tracking data be included to multiple ways of evaluating players based on the actions they make on the pitch.
2. The importance of a game as well as opponent's strength parameters be calculated and included into evaluation models.
3. Training data sets consist of data from tournaments such as the FIFA World Cup and UEFA Euro as the play style can be different than in leagues.

In the end, one should not forget that football is an unpredictable game with a strong influence that comes from random events. The use of data can help to find out the best players, but it should be combined with the the activity of watching a game as it answers more questions than even the best model.

---

[4]FC Barcelona (2018) *Agreement with KAS Eupen for the Transfer of Moussa Wague*, Retrieved from `https://www.fcbarcelona.com/en/news/807659/agreement-with-kas-eupen-for-the-transfer-of-moussa-wague` (Accessed:14 March 2021).

[5]FC Barcelona (2018) *Agreement with Everton for the Transfer of Yerry Mina*, Retrieved from `https://www.fcbarcelona.com/en/football/first-team/news/807724/agreement-with-everton-for-the-transfer-of-yerry-mina` (Accessed:14 March 2021).

[6]Transfermarkt (2021) *Yerry Mina*, Retrieved from `https://www.transfermarkt.pl/yerry-mina/profil/spieler/289446` (Accessed: 14 March 2021).

[7]Transfermarkt (2021) *Denis Cheryshev*, Retrieved from `https://www.transfermarkt.pl/denis-cheryshev/leistungsdaten/spieler/98322/plus/0?saison=2017` (Accessed:14 March 2021).

[8]BBC (2018) *Mbappe: France World Cup Star 'Taking Crown from Messi and Ronaldo'*, Retrieved from `https://www.bbc.com/sport/football/44839768` (Accessed:14 March 2021).

Bibliography

Books and Publications

1. Chong, E. P. K, Żak, S. H. (2001) *An Introduction to Optimization*, 2^{nd} edition, New York, Josh Wiley & Sons Inc;
2. Decroos, T., Bransen, L. Van Haaren, J. Davis, J. (2019) *Actions Speak Louder than Goals: Valuing Player Actions in Soccer*, Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, p. 1851–1861;
3. Greene, W. H. (2003) *Econometric Analysis*, 3^{rd} edition, New York, Pearson Education Inc;
4. Grimmett, G. R.; Stirzaker, D. R. (2001) *Probability and Random Processes*, 3^{nd} edition, Oxford University Press, Oxford;
5. Hvattum, L. M., Saebo, O. (2015) *Evaluating the Efficiency of the Association Football Transfer Market Using Regression Based Player Ratings*, In: NIK: Norsk Informatikkonferanse, Bibsys Open Journal Systems, 12 pages;
6. Hvattum, L. M., Saebo, O. (2019) *Modelling the Financial Contribution of Soccer Players to Their Clubs*, Journal of Sports Analytics, vol. 5, p. 23–34;
7. Lewis, M. (2003) *Moneyball. The Art of Winning an Unfair Game*, New York, W.W. Norton;
8. Pappalardo, L., Cintia, P., Rossi, A. et al. (2019) *A Public Data Set of Spatio-Temporal Match Events in Soccer Competitions.* Nature Scientific Data, vol. 6, no. 236;
9. Pollard, R. (1995) *Do Long Shots Pay Off?*, Soccer Journal, vol. 40, no. 3, p. 41-43;
10. Pollard, R., Reep C. (1997) *Measuring the Effectiveness of Playing Strategies at Soccer*, Journal of the Royal Statistical Society: Series D, vol. 46, no. 4, p. 459-582;
11. Rudd, S. (2011) *A Framework for Tactical Analysis and Individual Offensive Production Assessment in Soccer Using Markov Chains*, New England Symposium on Statistics in Sports;
12. Raschka, S., Mirjalili, V. (2017) *Python. Machine Learning.*, 2^{nd} edition, Birmingham, Packt Publishing;
13. Sill, J. (2010) *Improved NBA Adjusted +/– Using Regularization and Out-Of-Sample*

Testing, Proceedings of the 2010 MIT Sloan Sports Analytics Conference;

14. Sherlaw-Johnson, C., Gallivan, S., Burridge, J. (1995) *Estimating a Markov Transition Matrix from Observational Data*, The Journal of the Operational Research Society, vol. 46, no. 3, p. 405–410;
15. Thomas, A. C., Ventura, S., Jensen, S., Ma, S. (2013) *Competing Process Hazard Function Models for Player Ratings in Ice Hockey*, The Annals of Applied Statistics, vol 7, p. 1497–1524;
16. Van Roy, M., Robberechts, P., Decroos, T., Davis, J. (2020) *Valuing On-the-Ball Actions in Soccer: A Critical Comparison of xT and VAEP*, AAAI-20 Workshop on Artifical Intelligence in Team Sports;
17. Weishaimer, A., Palmer, T.N. (2014) *On the Reliability of Seasonal Climate Forecasts*, Journal of The Royal Society Interface, vol. 11, no. 96.

Web Sources

1. AvecSport (2019) *How to Attract a Sponsor for Your Grassrots Football Team*, Retrieved from `https://avecsport.com/blog/how-to-attract-a-sponsor-for-your-grassroots-football-team` (Accessed 14 April 2021);
2. BBC (2018) *Mbappe: France World Cup Star 'Taking Crown from Messi and Ronaldo'*, Retrieved from `https://www.bbc.com/sport/football/44839768` (Accessed:14 March 2021);
3. Bhattacharyya, S. (2018) *Ridge and Lasso Regression: L1 and L2 Regularization*, Towards Data Science, Retrieved from `https://towardsdatascience.com/ridge-and-lasso-regression-a-complete-guide-with-python-scikit-learn-e20e34bcbf0b`. (Accessed: 28 February 2021);
4. Cambridge Dictionary *Meaning of Sponsorship in English*, Retrieved from `https://dictionary.cambridge.org/dictionary/english/sponsorship` (Accessed: 13 April 2021);
5. Chatterjee, S. (2019) *A Deep Dive into Decision Trees*, Medium, Retrieved from: `https://medium.com/analytics-vidhya/a-deep-dive-to-decision-trees-6575e016d656` (Accessed:17 March 2021);
6. Christenson, M. (2018) *World Cup 2018: The Guardian Team of the Tournament*, The Guardian, Retrieved from `https://www.theguardian.com/football/2018/jul/16/world-cup-2018-the-guardian-team-of-the-tournament` (Accessed: 14 March 2021);
7. Deloitte (2020) *Annual Review of Football Finance*, Retrieved from `https://www2.deloitte.com/uk/en/pages/sports-business-group/articles`

/annual-review-of-football-finance.html (Accessed: 11 April 2021);

8. Desjardins, J. (2019) *How Much Data is Generated Each Day*, World Economic Forum, Retrieved from https://www.weforum.org/agenda/2019/04/how-much-data-is-generated-each-day-cf4bddf29f/ (Accessed:1 January 2021);
9. Diaz, J. F. (2021) *Clubs to Earn Almost Triple the Money from European Super League than Champions League*, Retrieved from https://www.marca.com/en/football/international-football/2021/04/19/607d46efca4741ff368b4606.html (Accessed: 24 April 2021);
10. Downs, D. (2020) *Can the £1 billion football jersey business overcome Covid-19?*, Vogue Business, Retrieved from https://www.voguebusiness.com/companies/football-jersey-sales-premier-league-messi (Accessed: 11 April 2021);
11. DTAI KU Leuven (2020) *Socceraction Documentation*, Retrieved from https://socceraction.readthedocs.io/en/latest/ (Accessed:18 February 2021);
12. Durpagal, A., Rowlinson, A. (2021) *Mplsoccer*, Retrieved from https://mplsoccer.readthedocs.io/en/latest/ (Accessed 14 March 2021);
13. El Houndm, A. (2021) *Brier Score: Understanding Model Calibration*, Neptune Blog, Retrieved from https://neptune.ai/blog/brier-score-and-model-calibration (Accessed:12 March 2021);
14. Fahran (2021) *Football*, Sportscriber, Retrieved from https://sportscriber.com/football/ (Accessed:17 April 2021);
15. FC Barcelona (2018) *Agreement with KAS Eupen for the Transfer of Moussa Wague*, Retrieved from https://www.fcbarcelona.com/en/news/807659/agreement-with-kas-eupen-for-the-transfer-of-moussa-wague (Accessed:14 March 2021);
16. FC Barcelona (2018) *Agreement with Everton for the Transfer of Yerry Mina*, Retrieved from https://www.fcbarcelona.com/en/football/first-team/news/807724/agreement-with-everton-for-the-transfer-of-yerry-mina (Accessed:14 March 2021); item FIFA (2014) *Adidas Golden Boot*, Retrieved from www.fifa.com/worldcup/news/adidas-golden-boot-2336480. (Accessed: 12 December 2020);
17. FIFA (2018) *Golden Consolation for Magical Modric*, Retrieved from: https://www.fifa.com/worldcup/news/157-awards-piece-2986294 (Accessed:15 March 2021);
18. FIFA (2018) *Matches*, Retrieved from https://www.fifa.com/worldcup/archive/russia2018/matches (Accessed:3 March 2021);

19. FIFA (2018) *Teams*, Retrieved from https://www.fifa.com/worldcup/archive/russia2018/teams/. (Accessed: 28 February 2021);
20. FIFA (2021) *A Review of International Football Transfers Worldwide*, Global Transfer Market Report, Retrieved from https://www.fifa.com/who-we-are/legal/tms/tms-reports/ (Accessed: 15 April 2021)
21. Gazapo, C. (2019) *TV Rights in Football - Premier League Analysis*, Sports Business Institute Barcelona, Retrieved from https://www.sbibarcelona.com/newsdetails/index/403, (Accessed 14 April 2021);
22. Ingle, S. (2015) *How Midtjyland Took the Analytical Route Towards the Champions League*, The Guardian, Retrieved from www.theguardian.com/football/2015/jul/27/how-fc-midtjylland-analytical-route-champions-league-brentford-matthew-benham. (Accessed: 12 December 2020);
23. KPMG (2019) *Broadcasting Revenue Landscape – Big Money in The "Big Five" Leagues*, Football Benchmark, Retrieved from https://www.footballbenchmark.com/library/broadcasting_revenue_landscape_big_money_in_the_big_five_leagues (Accessed: 24 April 2021);
24. Lange, D. (2020) *Premier League Clubs by Broadcasting Payments Received in 2019/20*, Statista, Retrieved from https://www.statista.com/statistics/240912/broadcasting-payments-to-clubs-in-the-english-premier-league/ (Accessed: 11 April 2021);
25. Lange, D. (2020) *Soccer Player Transfers by Highest All-Time Transfer Fee Worldwide as of October 2020*, Statista, Retrieved from https://www.statista.com/statistics/263304/transfer-fees-the-10-most-expensive-transfers-in-soccer-ever/ (Accessed: 11 April 2021);
26. Morde, V. (2018) *XGBoost Algorithm: Long May She Reign!*, Towards Data Science, Retrieved from https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f999be63d. (Accessed: 9 March 2021);
27. ODSC - Open Data Science (2019) *Logistic Regression with Python*, Medium, Retrieved from: https://medium.com/@ODSC/logistic-regression-with-python-ede39f8573c7 (Accessed: 17 March 2021);
28. OPTA *Advanced Metrics*, Retrieved from https://www.optasports.com/services/analytics/advanced-metrics/ (Accessed: 5 March 2020);
29. Real Madrid. (2014) *Communicado Oficial: James Rodriguez*, Retrieved from www.realmadrid.com/noticias/2014/07/comunicado-oficial-james-rodriguez. (Accessed: 12 December 2020);
30. Rowlinson, A. (2020) *Pitch Comparison*, Mplsoccer, Retrieved from https://mpls

occer.readthedocs.io/en/latest/gallery/pitch_setup/plot_compare_pitches.html#sphx-glr-gallery-pitch-setup-plot-compare-pitches-py. (Accessed: 17 March 2021);

31. Rosenbaum, D. (2004) *Measuring How NBA Players Help Their Teams Win*, 82games, Retrieved from http://www.82games.com/comm30.html. (Accessed: 28 February 2021);
32. Settmi, C. (2020) *The World's Highest-Paid Soccer Players 2020: Messi Wins, Mbappe Rises*, Forbes, Retrieved from https://www.forbes.com/sites/christinasettimi/2020/09/14/the-worlds-highest-paid-soccer-players-2020-messi-wins-mbappe-rises/?sh=639b3c561cff (Accessed: 16 April 2021);
33. Singh, K. (2019) *Introducing Expected Threat* Retrieved from https://karun.in/blog/expected-threat.html (Accessed: 2 January 2021);
34. SportMob (2020) *Biggest Kit Deals in Football History*, Retrieved from https://sportmob.com/en/article/891087-biggest-kit-deals-in-football-history (Accessed: 11 April 2021);
35. Nishant, S. (2020) *Biggest Sponsorship Deals in Football History*, SportsKhabri, Retireved from https://sportskhabri.com/biggest-sponsorship-deals-in-football-history/ (Accessed: 14 April 2021);
36. Sumpter, D. (2017) *The Geometry of Shooting*, Medium, Retrieved from https://soccermatics.medium.com/the-geometry-of-shooting-ae7a67fdf760. (Accessed: 5 March 2021);
37. Szymanski, S. (2016) *Inside the Business of Football*, OpenLearn, Retrieved from https://openlearn.medium.com/inside-the-business-of-football-4537acec31bb (Accessed: 11 April 2021);
38. Transfermarkt (2021) *Chelsea FC - Transfers 20/21*, Retrieved from https://www.transfermarkt.com/fc-chelsea/transfers/verein/631 (Accessed: 15 April 2021);
39. Transfermarkt (2021) *Denis Cheryshev*, Retrieved from https://www.transfermarkt.pl/denis-cheryshev/leistungsdaten/spieler/98322/plus/0?saison=2017 (Accessed:14 March 2021);
40. Transfermarkt (2021) *Yerry Mina*, Retrieved from https://www.transfermarkt.pl/yerry-mina/profil/spieler/289446 (Accessed: 14 March 2021);
41. Wikipedia (2021) *List of Domestic Football League Broadcast Deals by Country*, Retrieved from https://en.wikipedia.org/wiki/List_of_domestic_football_league_broadcast_deals_by_country (Accessed: 14 March 2021).

List of Figures

List of Tables

Source Codes

Expected Goals Calculations

```
import json
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
import statsmodels.api as sm

def open_file(string):
    with open (string) as f:
        file = json.load(f)

    df = pd.DataFrame(file)
    df = df[df['subEventName'].isin(['Shot', 'Free kick shot'])]

    return df


df1 = open_file('Wyscout/events_Germany.json')
df2 = open_file('Wyscout/events_England.json')
df3 = open_file('Wyscout/events_Spain.json')
df4 = open_file('Wyscout/events_France.json')
df5 = open_file('Wyscout/events_Italy.json')


def concat_df (df1, df2, df3, df4, df5):
    frames = [df1, df2, df3, df4, df5]
    new_df = pd.concat(frames)
    return new_df

full_df = concat_df(df1, df2, df3, df4, df5)


def drop_headers(df):
   lista = []
   for i, row in df.iterrows():
       for tag in row["tags"]:
           if tag['id'] == 403:
              lista.append(i)
   return df.drop(lista)

full_df2 = drop_headers(full_df)
headers_df = full_df.drop(full_df2.index)

def penalty (string):
    with open (string) as f:
        file = json.load(f)

    df = pd.DataFrame(file)
    df = df[df['subEventName'].isin(["Penalty"])]
    df = df[df["matchPeriod"] != "P"]
    for i, row in df.iterrows():
        df.at[i,'Goal'] = 0
        for tag in row["tags"]:
            if tag['id'] == 101:
                df.at[i, "Goal"] = 1

    return df
```

```
def prepare_df_model (df):
    X_df = pd.DataFrame()
    X_df["X"] = df.positions.apply(lambda cell: 100 - cell[0]['x'])
    X_df["X"] = X_df["X"]*105/100
    X_df["C"] = df.positions.apply(lambda cell: cell[0]['y'])
    X_df["C"] = X_df["C"]*68/100
    X_df["Y"] = df.positions.apply(lambda cell: abs(cell[0]['y']-50))
    X_df["Y"] = X_df["Y"]*68 /100
    X_df["Distance"] = np.sqrt(X_df["X"]**2 + X_df["Y"]**2)
    X_df["Angle"] = np.where(np.arctan(7.32 * X_df["X"] /(X_df["X"]**2 + X_df["Y"]**2 - (7.32/2)**2)) > 0, np.arctan(7.32 *
          X_df["X"] /(X_df["X"]**2 + X_df["Y"]**2 - (7.32/2)**2)), np.arctan(7.32 * X_df["X"] /(X_df["X"]**2 + X_df["Y"]**2 -
          (7.32/2)**2)) + np.pi)
    for i, row in df.iterrows():
        X_df.at[i,'Goal'] = 0
        for tag in row["tags"]:
            if tag['id'] == 101:
                X_df.at[i, "Goal"] = 1

    return X_df

model_df = prepare_df_model(full_df2)
headers_model_df = prepare_df_model(headers_df)

def build_model (df):
    xG_model = smf.glm("Goal ~ Distance + Angle", data = df,  family = sm.families.Binomial()).fit()
    print(xG_model.summary())
    return xG_model.params

params = build_model(model_df)
params2 = build_model(headers_model_df)

all_wc = open_file("Wyscout/events_World_Cup.json")
world_cup_shots = drop_headers(all_wc)
world_cup_headers = all_wc.drop(world_cup_shots.index)
world_cup_df = prepare_df_model(world_cup_shots)
world_cup_headers_df = prepare_df_model(world_cup_headers)

def calculate_XG(df, params):
     df["xG"] = 1/(1+np.exp(-(params[0] + (params[1]*df["Distance"]) +(params[2]*df["Angle"]))))
     return df

from io import BytesIO
with open('Wyscout/players.json', 'rb') as json_file:
        players = BytesIO(json_file.read()).getvalue().decode('unicode_escape')


world_cup_df = calculate_XG(world_cup_df, params)
world_cup_headers_df = calculate_XG(world_cup_headers_df, params2)
world_cup_shots = pd.concat([world_cup_shots, world_cup_headers])
world_cup_df = pd.concat([world_cup_df, world_cup_headers_df])
df = world_cup_df.join(world_cup_shots)


players_df = pd.read_json(players)
players_df2 = pd.DataFrame()
players_df2["playerId"] = players_df["wyId"]
players_df2["role"] = players_df["role"]
players_df2["shortName"] = players_df["shortName"]

df = df.reset_index().merge(players_df2, how = "inner", on = ["playerId"]).set_index("index")
summary_df = df[["Goal", "xG", "shortName"]]



penalty_wc = penalty("Wyscout/events_World_Cup.json")
penalty_wc = penalty_wc.reset_index().merge(players_df2, how = "inner", on = ["playerId"]).set_index("index")
penalty_wc["xG"] = 0.8
penalty_wc = penalty_wc[["Goal", "xG", "shortName","playerId"]]

summary_df = pd.concat([summary_df, penalty_wc])

xG_sum = summary_df.groupby(["shortName"])["xG"].sum().sort_values(ascending = False)
goals_sum = summary_df.groupby(["shortName"])["Goal"].sum().sort_values(ascending = False)
```

```
import matplotlib.pyplot as plt
frame = { 'xG': xG_sum, 'goals': goals_sum}

summary = pd.DataFrame(frame).reset_index()

summary = pd.DataFrame(frame).reset_index()
summary["shortName"] = summary["index"]
df_games = pd.read_hdf('spadl.h5', key='games')
df_games_test = df_games[df_games["competition_id"] == 28]
df_player_games = pd.read_hdf('spadl.h5', 'player_games')
df_player_games = df_player_games[df_player_games['game_id'].isin(df_games_test['game_id'])]

#adding extra time minutes
df_player_games.loc[(df_player_games["game_id"] == 2058015) & (df_player_games["player_id"].isin([12829, 210044, 69964, 69411,
       69400, 14771])), "minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058015) & (df_player_games["
      player_id"].isin([12829, 210044, 69964, 69411, 69400, 14771]))]["minutes_played"].apply(lambda x: x+26)
df_player_games.loc[(df_player_games["game_id"] == 2058015) & (df_player_games["player_id"].isin([9380, 8653, 8945, 8717,
      13484, 10131, 7934, 69409, 25393, 135747, 3476, 69396, 69968, 14812, 397178, 8292])), "minutes_played"] =
      df_player_games[(df_player_games["game_id"] == 2058015) & (df_player_games["player_id"].isin([9380, 8653, 8945, 8717,
      13484, 10131, 7934, 69409, 25393, 135747, 3476, 69396, 69968, 14812, 397178, 8292]))]["minutes_played"].apply(lambda x:
      x+30)


df_player_games.loc[(df_player_games["game_id"] == 2058012) & (df_player_games["player_id"].isin([14771, 101590])), "
      minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058012) & (df_player_games["player_id"].isin([14771,
      101590]))]["minutes_played"].apply(lambda x: x+25)
df_player_games.loc[(df_player_games["game_id"] == 2058012) & (df_player_games["player_id"].isin([103668, 220971, 41123,
      103682, 101647, 101576, 101583, 101953, 101707, 102157, 25393, 135747, 14943, 3476, 8287, 69396, 69616, 69968, 69964,
      69404])), "minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058012) & (df_player_games["player_id"].
      isin([103668, 220971, 41123, 103682, 101647, 101576, 101583, 101953, 101707, 102157, 25393, 135747, 14943, 3476, 8287,
      69396, 69616, 69968, 69964, 69404]))]["minutes_played"].apply(lambda x: x+30)

df_player_games.loc[(df_player_games["game_id"] == 2058009) & (df_player_games["player_id"].isin([3531, 8292, 397178])), "
      minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058009) & (df_player_games["player_id"].isin([3531,
      8292, 397178]))]["minutes_played"].apply(lambda x: x+24)
df_player_games.loc[(df_player_games["game_id"] == 2058009) & (df_player_games["player_id"].isin([9380, 8653, 8945, 8717,
      7964, 10131, 7934, 210044, 12829, 246928, 25662, 257762, 91702, 256634, 20751, 3450, 37831, 91502, 20764])), "
      minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058009) & (df_player_games["player_id"].isin([9380,
      8653, 8945, 8717, 7964, 10131, 7934, 210044, 12829, 246928, 25662, 257762, 91702, 256634, 20751, 3450, 37831, 91502,
      20764]))]["minutes_played"].apply(lambda x: x+30)

df_player_games.loc[(df_player_games["game_id"] == 2058005) & (df_player_games["player_id"].isin([241945, 56025, 69411,
      69400])), "minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058005) & (df_player_games["player_id"].
      isin([241945, 56025, 69411, 69400]))]["minutes_played"].apply(lambda x: x+27)
df_player_games.loc[(df_player_games["game_id"] == 2058005) & (df_player_games["player_id"].isin([55957, 56394, 8480, 54,
      55979, 56274, 20433, 405, 15080, 69409, 25393, 135747, 3476, 8287, 69396, 69616, 69404, 69964])), "minutes_played"] =
      df_player_games[(df_player_games["game_id"] == 2058005) & (df_player_games["player_id"].isin([55957, 56394, 8480, 54,
      55979, 56274, 20433, 405, 15080, 69409, 25393, 135747, 3476, 8287, 69396, 69616, 69404, 69964]))]["minutes_played"].
      apply(lambda x: x+30)

df_player_games.loc[(df_player_games["game_id"] == 2058004) & (df_player_games["player_id"].isin([101953, 70129])), "
      minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058004) & (df_player_games["player_id"].isin([101953,
       70129]))]["minutes_played"].apply(lambda x: x+26)
df_player_games.loc[(df_player_games["game_id"] == 2058004) & (df_player_games["player_id"].isin([103668, 41123, 257800,
      103682, 101647, 101576, 101583, 101682, 4513, 101707, 7910, 3443, 3306, 3346, 3341, 3269, 3563, 3353, 4501, 3840])), "
      minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058004) & (df_player_games["player_id"].isin([103668,
       41123, 257800, 103682, 101647, 101576, 101583, 101682, 4513, 101707, 7910, 3443, 3306, 3346, 3341, 3269, 3563, 3353,
      4501, 3840]))]["minutes_played"].apply(lambda x: x+30)


minutes = df_player_games[['player_id', 'minutes_played']]
minutes = minutes.groupby('player_id').sum().reset_index()
minutes["playerId"] = minutes["player_id"]

minutes = minutes.reset_index().merge(players_df2, how = "inner", on = ["playerId"]).set_index("index")
summary = summary.merge(minutes, how = "inner", on = ["shortName"]).set_index("index")
summary = summary[summary['minutes_played']>150]
summary["xG_90"] = summary["xG"]*90/summary["minutes_played"]
summary["goals_90"] = summary["goals"]*90/summary["minutes_played"]

test = summary.loc[(summary["xG_90"] > 0.48) | (summary["goals_90"] > 0.7)]
not_test = summary.drop(test.index)
summarys = summary.reset_index()
summarys["index"].tolist
fig, ax = plt.subplots(num = 1)
```

```
ax.grid(zorder = 1)
ax.scatter(summary["xG_90"], summary["goals_90"])
for i in test.index:
    ax.scatter(summarys.loc[summarys["index"] == i]["xG_90"], summarys.loc[summarys["index"] == i]["goals_90"], edgecolors = "
         black" ,color = "grey", alpha = 0.3, lw = 0.6, zorder  =3)
    ax.text(summarys.loc[summarys["index"] == i]["xG_90"]-0.05, summarys.loc[summarys["index"] == i]["goals_90"]+0.01, str(i),
          fontsize = 8, color = "black", zorder = 2)

for i in not_test.index:
    ax.scatter(summarys.loc[summarys["index"] == i]["xG_90"], summarys.loc[summarys["index"] == i]["goals_90"],  edgecolors =
         "black" ,color = "grey", alpha = 0.2, lw = 0.6, zorder = 3)
ax.plot([0, 1], [0, 1],linestyle='dotted', color='red', alpha = 0.5, zorder = 4)
ax.tick_params(axis="x",color="black",length=5, width=1)
ax.tick_params(axis="y",color="black",length=5, width=1)
plt.xlim(0, 1)
plt.ylim(0, 1.3)
ax.set_ylabel("Goals scored per 90 minutes", color='black', fontsize = 14)
ax.set_xlabel("xG per 90 minutes", color='black', fontsize = 14)
plt.ylim((0.00,1.20))
plt.xlim((0.00,1.00))
plt.tight_layout()

pgoal_2d=np.zeros((65,65))
for x in range(65):
    for y in range(65):
        sh=dict()
        a = np.arctan(7.32 *x /(x**2 + abs(y-65/2)**2 - (7.32/2)**2))
        if a<0:
            a = np.pi + a
        sh['Angle'] = a
        sh['Distance'] = np.sqrt(x**2 + abs(y-65/2)**2)
        sh = pd.DataFrame(sh, index = [x])
        sh =  calculate_XG(sh, params)
        pgoal_2d[x,y] = sh["xG"]
import FCPython
(fig,ax) = FCPython.createGoalMouth(linecolor = "black")
pos=ax.imshow(pgoal_2d, extent=[-1,65,65,-1], aspect='auto',cmap=plt.cm.Reds,vmin=0, vmax=1)
fig.colorbar(pos, ax=ax)
ax.set_title('Goal probability')
plt.xlim((0,66))
plt.ylim((-3,35))
plt.gca().set_aspect('equal', adjustable='box')
plt.show()
fig.savefig("scoring_prob2.png", dpi=None, bbox_inches="tight")

pgoal_2d=np.zeros((65,65))
for x in range(65):
    for y in range(65):
        sh=dict()
        a = np.arctan(7.32 *x /(x**2 + abs(y-65/2)**2 - (7.32/2)**2))
        if a<0:
            a = np.pi + a
        sh['Angle'] = a
        sh['Distance'] = np.sqrt(x**2 + abs(y-65/2)**2)
        sh = pd.DataFrame(sh, index = [x])
        sh =  calculate_XG(sh, params2)
        pgoal_2d[x,y] = sh["xG"]
import FCPython
(fig,ax) = FCPython.createGoalMouth(linecolor = "black")
pos=ax.imshow(pgoal_2d, extent=[-1,65,65,-1], aspect='auto',cmap=plt.cm.Reds,vmin=0, vmax=1)
fig.colorbar(pos, ax=ax)
ax.set_title('Goal probability (header)')
plt.xlim((0,66))
plt.ylim((-3,35))
plt.gca().set_aspect('equal', adjustable='box')
plt.show()
fig.savefig("scoring_prob_head2.png", dpi=None, bbox_inches="tight")



plt.hist(model_df["Distance"], bins = 100)
plt.title("Shots - distance")
plt.savefig("grafiki_licencjat/xgshots_dist.png")
plt.show()
plt.hist(model_df["Angle"], bins = 100)
```

```
plt.title("Shots - angle")
plt.savefig("grafiki_licencjat/xgshots_ang.png")

plt.show()
plt.hist(headers_model_df["Distance"], bins = 100)
plt.title("Headers - distance")
plt.savefig("grafiki_licencjat/xghead_dist.png")

plt.show()
plt.hist(headers_model_df["Angle"], bins = 100)
plt.title("Headers - angle")
plt.savefig("grafiki_licencjat/xghead_ang.png")

plt.show()
plt.hist(world_cup_df["Distance"], bins = 100)
plt.title("Shots - distance")
plt.savefig("grafiki_licencjat/xgshots_dist_test.png")

plt.show()
plt.hist(world_cup_df["Angle"], bins = 100)
plt.title("Shots - angle")
plt.savefig("grafiki_licencjat/xgshots_ang_test.png")

plt.show()
plt.hist(world_cup_headers_df["Distance"], bins = 100)
plt.title("Headers - distance")
plt.savefig("grafiki_licencjat/xghead_dist_test.png")

plt.show()
plt.hist(world_cup_headers_df["Angle"], bins = 100)
plt.title("Headers - angle")
plt.savefig("grafiki_licencjat/xghead_ang_test.png")

plt.show()
```

Real Adjusted Plus-Minus Calculations

```
import pandas as pd
df = pd.read_excel('RAPM.xlsx', engine='openpyxl')

y = df["y"]
y = y.fillna(0)
df.drop(df.columns[[0,1,2,3]], axis = 1, inplace = True)
df = df.fillna(0)
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
y = y.values
X = df.values
alpha = [3000]
params = {"alpha": alpha}
ridge = Ridge()
grid = GridSearchCV(ridge, params)
gs = grid.fit(X, y)
clf = grid.best_estimator_
clf = gs.best_estimator_
clf.fit(X, y)

rank = {"ranking": clf.coef_.T}
ranking = pd.DataFrame(data = rank, index = df.columns).sort_values("ranking", ascending = False)
```

Expected Threat Calculations

```
import pandas as pd
import numpy as np
import json
```

```
with open ("Wyscout/events_England.json") as f:
     file = json.load(f)

df = pd.DataFrame(file)
pd.unique(df["eventName"])


xT_df = df[df['subEventName'].isin(['Shot', 'Free kick shot', 'Simple pass', 'High pass', 'Throw in', 'Head pass', 'Smart pass
     ','Cross', 'Free kick cross','Corner'])]


xT_df["x0"] = xT_df.positions.apply(lambda cell: cell[0]['x']) * 105/100
xT_df["y0"] = xT_df.positions.apply(lambda cell: cell[0]['y']) * 68/100

move_df = xT_df[xT_df['subEventName'].isin(['Simple pass', 'High pass', 'Throw in', 'Head pass', 'Smart pass','Cross', 'Goal
     kick','Free kick cross','Corner'])]

move_df['x1'] = move_df.positions.apply(lambda cell: cell[1]['x']) * 105/100
move_df['y1'] = move_df.positions.apply(lambda cell: cell[1]['y']) * 68/100


shot_df = xT_df[xT_df['subEventName'].isin(['Shot', 'Free kick shot'])]
shot_df['x1'] = "S"
shot_df['y1'] = "S"

xT_df = pd.concat([move_df, shot_df])

for i, row in xT_df.iterrows():
    for j in range(0, 16):
        if row["y0"] < (1/12)*68 and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 1 +12*j
        if row["y0"] >= (1/12)*68 and row["y0"] < (2/12)*68 and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 2 +12*j
        if row["y0"] >= (2/12)*68 and row["y0"] < (3/12)*68  and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 3 +12*j
        if row["y0"] >= (3/12)*68  and row["y0"] < (4/12)*68  and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 4 +12*j
        if row["y0"] >= (4/12)*68  and row["y0"] < (5/12)*68  and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 5 +12*j
        if row["y0"] >=(5/12)*68  and row["y0"] < (6/12)*68  and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 6 +12*j
        if row["y0"] >= (6/12)*68  and row["y0"] < (7/12)*68  and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 7 +12*j
        if row["y0"] >= (7/12)*68  and row["y0"] < (8/12)*68  and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 8 +12*j
        if row["y0"] >= (8/12)*68  and row["y0"] < (9/12)*68  and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 9 +12*j
        if row["y0"] >= (9/12)*68  and row["y0"] < (10/12)*68  and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 10 +12*j
        if row["y0"] >= (10/12)*68  and row["y0"] < (11/12)*68  and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 11 +12*j
        if row["y0"] >= (11/12)*68   and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 12 +12*j




for i, row in xT_df.iterrows():
    if row["x1"] != "S":
        for j in range(0, 16):
            if row["y1"] < (1/12)*68  and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)*105:
                xT_df.at[i, "stop"] = 1 +12*j
            if row["y1"] >= (1/12)*68  and row["y1"] < (2/12)*68  and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)*105:
                xT_df.at[i, "stop"] = 2 +12*j
            if row["y1"] >= (2/12)*68  and row["y1"] < (3/12)*68  and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)*105:
                xT_df.at[i, "stop"] = 3 +12*j
            if row["y1"] >= (3/12)*68  and row["y1"] < (4/12)*68  and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)*105:
                xT_df.at[i, "stop"] = 4 +12*j
            if row["y1"] >= (4/12)*68  and row["y1"] < (5/12)*68  and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)*105:
                xT_df.at[i, "stop"] = 5 +12*j
            if row["y1"] >=(5/12)*68  and row["y1"] < (6/12)*68  and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)*105:
                xT_df.at[i, "stop"] = 6 +12*j
            if row["y1"] >= (6/12)*68  and row["y1"] < (7/12)*68  and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)*105:
                xT_df.at[i, "stop"] = 7 +12*j
            if row["y1"] >= (7/12)*68  and row["y1"] < (8/12)*68  and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)*105:
                xT_df.at[i, "stop"] = 8 +12*j
```

```
            if row["y1"] >= (8/12)*68  and row["y1"] < (9/12)*68  and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)*105:
                xT_df.at[i, "stop"] = 9 +12*j
            if row["y1"] >= (9/12)*68  and row["y1"] < (10/12)*68  and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)*105:
                xT_df.at[i, "stop"] = 10 +12*j
            if row["y1"] >= (10/12)*68  and row["y1"] < (11/12)*68  and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)
                 *105:
                xT_df.at[i, "stop"] = 11 +12*j
            if row["y1"] >= (11/12)*68   and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)*105:
                xT_df.at[i, "stop"] = 12 +12*j
    else:
       xT_df.at[i, "stop"] = 1000

delete_df = xT_df.query("x1 == 0 & y1 == 0")
xT_df = xT_df.drop(delete_df.index)


goals_df = xT_df.query("stop == 1000")
xT_df = xT_df.drop(goals_df.index)

count_starts_and_stops = xT_df.groupby(["start", "stop"])["start"].count().unstack(fill_value=0).stack()
count_starts = xT_df.groupby(["start"])["start"].count()

transition_matrix = np.zeros((192,192))

for i in range (1, 187):
    for j in range (1, 193):
        transition_matrix[i-1, j-1] = count_starts_and_stops[i, j]/count_starts[i]

for i in range (188, 193):
    for j in range (1, 193):
        transition_matrix[i-1, j-1] = count_starts_and_stops[i, j]/count_starts[i]


xT_df = pd.concat([xT_df, goals_df])
shot_starts_and_stops = xT_df.groupby(["start", "stop"])["start"].count().unstack(fill_value=0).stack()
shot_starts = xT_df.groupby(["start"])["start"].count()


shot_probability = pd.DataFrame()

for i in range (1, 193):
    shot_probability.at[i, "prob"] = shot_starts_and_stops[i, 1000]/shot_starts[i]



move_probability = 1 - shot_probability


for i, row in goals_df.iterrows():
        goals_df.at[i,'Goal'] = 0
        for tag in row["tags"]:
            if tag['id'] == 101:
                goals_df.at[i, "Goal"] = 1


goals_from_area = goals_df.groupby("start")["Goal"].sum()



goal_probability = goals_from_area/shot_starts
goal_probability = goal_probability.fillna(0)



xT = np.zeros(192)
suma = np.zeros(192)
k = 0
while k < 5:
    for i in range(0, 192):
        for j in range (0, 192):
            suma[i] = suma[i] + (transition_matrix[i, j] * xT[j])
        xT[i] = (shot_probability.iloc[i]*goal_probability.iloc[i])+(move_probability.iloc[i]*suma[i])
    suma = np.zeros(192)
    k=k+1

############################################################
```

```
with open ("Wyscout/events_World_Cup.json") as f:
     file = json.load(f)

df = pd.DataFrame(file)
pd.unique(df["eventName"])

xT_df = df[df['subEventName'].isin(['Shot', 'Free kick shot', 'Simple pass', 'High pass', 'Throw in', 'Head pass', 'Smart pass
     ','Cross', 'Free kick cross','Corner'])]


xT_df["x0"] = xT_df.positions.apply(lambda cell: cell[0]['x']) * 105/100
xT_df["y0"] = xT_df.positions.apply(lambda cell: cell[0]['y']) * 68 /100

move_df = xT_df[xT_df['subEventName'].isin(['Simple pass', 'High pass', 'Throw in', 'Head pass', 'Smart pass','Cross', 'Goal
     kick','Free kick cross','Corner'])]

#data collecting error dla id 98656, 100297
move_df = move_df.drop([98656, 100297])
move_df['x1'] = move_df.positions.apply(lambda cell: cell[1]['x']) * 105/100
move_df['y1'] = move_df.positions.apply(lambda cell: cell[1]['y']) * 68 /100


shot_df = xT_df[xT_df['subEventName'].isin(['Shot', 'Free kick shot'])]
shot_df['x1'] = "S"
shot_df['y1'] = "S"

xT_df = pd.concat([move_df, shot_df])

for i, row in xT_df.iterrows():
    for j in range(0, 16):
        if row["y0"] < (1/12)*68  and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 1 +12*j
        if row["y0"] >= (1/12)*68  and row["y0"] < (2/12)*68  and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 2 +12*j
        if row["y0"] >= (2/12)*68  and row["y0"] < (3/12)*68  and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 3 +12*j
        if row["y0"] >= (3/12)*68  and row["y0"] < (4/12)*68  and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 4 +12*j
        if row["y0"] >= (4/12)*68  and row["y0"] < (5/12)*68  and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 5 +12*j
        if row["y0"] >=(5/12)*68  and row["y0"] < (6/12)*68  and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 6 +12*j
        if row["y0"] >= (6/12)*68  and row["y0"] < (7/12)*68  and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 7 +12*j
        if row["y0"] >= (7/12)*68  and row["y0"] < (8/12)*68  and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 8 +12*j
        if row["y0"] >= (8/12)*68  and row["y0"] < (9/12)*68  and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 9 +12*j
        if row["y0"] >= (9/12)*68  and row["y0"] < (10/12)*68  and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 10 +12*j
        if row["y0"] >= (10/12)*68  and row["y0"] < (11/12)*68  and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 11 +12*j
        if row["y0"] >= (11/12)*68   and row["x0"] < ((j+1)/16)*105 and row["x0"] >= (j/16)*105:
            xT_df.at[i, "start"] = 12 +12*j




for i, row in xT_df.iterrows():
    if row["x1"] != "S":
        for j in range(0, 16):
            if row["y1"] < (1/12)*68  and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)*105:
                xT_df.at[i, "stop"] = 1 +12*j
            if row["y1"] >= (1/12)*68  and row["y1"] < (2/12)*68  and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)*105:
                xT_df.at[i, "stop"] = 2 +12*j
            if row["y1"] >= (2/12)*68  and row["y1"] < (3/12)*68  and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)*105:
                xT_df.at[i, "stop"] = 3 +12*j
            if row["y1"] >= (3/12)*68  and row["y1"] < (4/12)*68  and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)*105:
                xT_df.at[i, "stop"] = 4 +12*j
            if row["y1"] >= (4/12)*68  and row["y1"] < (5/12)*68  and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)*105:
                xT_df.at[i, "stop"] = 5 +12*j
            if row["y1"] >=(5/12)*68  and row["y1"] < (6/12)*68  and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)*105:
                xT_df.at[i, "stop"] = 6 +12*j
            if row["y1"] >= (6/12)*68  and row["y1"] < (7/12)*68  and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)*105:
                xT_df.at[i, "stop"] = 7 +12*j
```

```
            if row["y1"] >= (7/12)*68  and row["y1"] < (8/12)*68  and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)*105:
                xT_df.at[i, "stop"] = 8 +12*j
            if row["y1"] >= (8/12)*68  and row["y1"] < (9/12)*68  and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)*105:
                xT_df.at[i, "stop"] = 9 +12*j
            if row["y1"] >= (9/12)*68  and row["y1"] < (10/12)*68  and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)*105:
                xT_df.at[i, "stop"] = 10 +12*j
            if row["y1"] >= (10/12)*68  and row["y1"] < (11/12)*68  and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)
                 *105:
                xT_df.at[i, "stop"] = 11 +12*j
            if row["y1"] >= (11/12)*68   and row["x1"] < ((j+1)/16)*105 and row["x1"] >= (j/16)*105:
                xT_df.at[i, "stop"] = 12 +12*j
    else:
       xT_df.at[i, "stop"] = 1000

delete_df = xT_df.query("x1 == 0 & y1 == 0")
xT_df = xT_df.drop(delete_df.index)

move_df = xT_df[xT_df['subEventName'].isin(['Simple pass', 'High pass', 'Throw in', 'Head pass', 'Smart pass','Cross', 'Goal
      kick','Free kick cross','Corner'])]
move_df = move_df.dropna()
for i, row in move_df.iterrows():
    move_df.at[i, "xT_start"] = xT[int(row["start"]-1)]
    move_df.at[i, "xT_stop"] = xT[int(row["stop"] -1)]



for i, thepass in move_df.iterrows():
    for passtags in thepass['tags']:
        if passtags['id']==1801:
            move_df.at[i, 'Accurate'] = 1
        else:
            move_df.at[i, 'Accurate'] = 0

move_df.loc[move_df['Accurate'] == 0, 'xT_stop'] = 0


move_df["difference"] = move_df["xT_stop"] - move_df["xT_start"]

from io import BytesIO
with open('Wyscout/players.json', 'rb') as json_file:
        players = BytesIO(json_file.read()).getvalue().decode('unicode_escape')

players_df = pd.read_json(players)
players_df2 = pd.DataFrame()
players_df2["playerId"] = players_df["wyId"]
players_df2["role"] = players_df["role"]
players_df2["shortName"] = players_df["shortName"]

df = move_df.reset_index().merge(players_df2, how = "inner", on = ["playerId"]).set_index("index")
summary_df = df[["difference", "shortName", "playerId"]]

xT_sum = summary_df.groupby(["playerId"])["difference"].sum().sort_values(ascending = False).reset_index()

df_games = pd.read_hdf('spadl.h5', key='games')
df_games_test = df_games[df_games["competition_id"] == 28]
df_player_games = pd.read_hdf('spadl.h5', 'player_games')
df_player_games = df_player_games[df_player_games['game_id'].isin(df_games_test['game_id'])]

#adding extra time minutes
df_player_games.loc[(df_player_games["game_id"] == 2058015) & (df_player_games["player_id"].isin([12829, 210044, 69964, 69411,
       69400, 14771])), "minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058015) & (df_player_games["
      player_id"].isin([12829, 210044, 69964, 69411, 69400, 14771]))]["minutes_played"].apply(lambda x: x+26)
df_player_games.loc[(df_player_games["game_id"] == 2058015) & (df_player_games["player_id"].isin([9380, 8653, 8945, 8717,
      13484, 10131, 7934, 69409, 25393, 135747, 3476, 69396, 69968, 14812, 397178, 8292])), "minutes_played"] =
      df_player_games[(df_player_games["game_id"] == 2058015) & (df_player_games["player_id"].isin([9380, 8653, 8945, 8717,
      13484, 10131, 7934, 69409, 25393, 135747, 3476, 69396, 69968, 14812, 397178, 8292]))]["minutes_played"].apply(lambda x:
      x+30)


df_player_games.loc[(df_player_games["game_id"] == 2058012) & (df_player_games["player_id"].isin([14771, 101590])), "
      minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058012) & (df_player_games["player_id"].isin([14771,
      101590]))]["minutes_played"].apply(lambda x: x+25)
df_player_games.loc[(df_player_games["game_id"] == 2058012) & (df_player_games["player_id"].isin([103668, 220971, 41123,
      103682, 101647, 101576, 101583, 101953, 101707, 102157, 25393, 135747, 14943, 3476, 8287, 69396, 69616, 69968, 69964,
      69404])), "minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058012) & (df_player_games["player_id"].
      isin([103668, 220971, 41123, 103682, 101647, 101576, 101583, 101953, 101707, 102157, 25393, 135747, 14943, 3476, 8287,
```

```
      69396, 69616, 69968, 69964, 69404]))]["minutes_played"].apply(lambda x: x+30)

df_player_games.loc[(df_player_games["game_id"] == 2058009) & (df_player_games["player_id"].isin([3531, 8292, 397178])), "
      minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058009) & (df_player_games["player_id"].isin([3531,
      8292, 397178]))]["minutes_played"].apply(lambda x: x+24)
df_player_games.loc[(df_player_games["game_id"] == 2058009) & (df_player_games["player_id"].isin([9380, 8653, 8945, 8717,
      7964, 10131, 7934, 210044, 12829, 246928, 25662, 257762, 91702, 256634, 20751, 3450, 37831, 91502, 20764])), "
      minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058009) & (df_player_games["player_id"].isin([9380,
      8653, 8945, 8717, 7964, 10131, 7934, 210044, 12829, 246928, 25662, 257762, 91702, 256634, 20751, 3450, 37831, 91502,
      20764]))]["minutes_played"].apply(lambda x: x+30)

df_player_games.loc[(df_player_games["game_id"] == 2058005) & (df_player_games["player_id"].isin([241945, 56025, 69411,
      69400])), "minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058005) & (df_player_games["player_id"].
      isin([241945, 56025, 69411, 69400]))]["minutes_played"].apply(lambda x: x+27)
df_player_games.loc[(df_player_games["game_id"] == 2058005) & (df_player_games["player_id"].isin([55957, 56394, 8480, 54,
      55979, 56274, 20433, 405, 15080, 69409, 25393, 135747, 3476, 8287, 69396, 69616, 69404, 69964])), "minutes_played"] =
      df_player_games[(df_player_games["game_id"] == 2058005) & (df_player_games["player_id"].isin([55957, 56394, 8480, 54,
      55979, 56274, 20433, 405, 15080, 69409, 25393, 135747, 3476, 8287, 69396, 69616, 69404, 69964]))]["minutes_played"].
      apply(lambda x: x+30)

df_player_games.loc[(df_player_games["game_id"] == 2058004) & (df_player_games["player_id"].isin([101953, 70129])), "
      minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058004) & (df_player_games["player_id"].isin([101953,
       70129]))]["minutes_played"].apply(lambda x: x+26)
df_player_games.loc[(df_player_games["game_id"] == 2058004) & (df_player_games["player_id"].isin([103668, 41123, 257800,
      103682, 101647, 101576, 101583, 101682, 4513, 101707, 7910, 3443, 3306, 3346, 3341, 3269, 3563, 3353, 4501, 3840])), "
      minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058004) & (df_player_games["player_id"].isin([103668,
       41123, 257800, 103682, 101647, 101576, 101583, 101682, 4513, 101707, 7910, 3443, 3306, 3346, 3341, 3269, 3563, 3353,
      4501, 3840]))]["minutes_played"].apply(lambda x: x+30)

minutes = df_player_games[['player_id', 'minutes_played']]
minutes = minutes.groupby('player_id').sum().reset_index()

players_df2["player_id"] = players_df["wyId"]
minutes_df = minutes.merge(players_df2, how = "inner", on = ["player_id"])
xT_sum["player_id"] = xT_sum["playerId"]
xT_per_90 = minutes.merge(xT_sum, how = "inner", on = ["player_id"])



xT_per_90 = df = players_df2.merge(xT_per_90, how = "inner", on = ["playerId"])
summary_per_df = df[["difference", "shortName", 'minutes_played']]
summary_per_df["per90"] = summary_per_df["difference"]*90/summary_per_df['minutes_played']

summary_per_df = summary_per_df[summary_per_df['minutes_played']>150]
summary_per_df = summary_per_df.sort_values('per90', ascending=False)


pmove_2d=np.zeros((16,12))
for x in range(16):
    for y in range(12):
        pmove_2d[x, y] = move_probability["prob"][(12*(x+1) - y)-1:(12*(x+1) - y)]

pshot_2d=np.zeros((16,12))
for x in range(16):
    for y in range(12):
        pshot_2d[x, y] = shot_probability["prob"][(12*(x+1) - y)-1:(12*(x+1) - y)]

pgoal_2d=np.zeros((16,12))
for x in range(16):
    for y in range(12):
        pgoal_2d[x, y] = goal_probability[(12*(x+1) - y)]



from FCPython import createPitch
(fig,ax) = createPitch(105,68,'meters','gray')
import matplotlib.pyplot as plt
pos=ax.imshow(pgoal_2d.T, extent=[0,105,68,0], aspect='auto',cmap=plt.cm.Reds,vmin=0, vmax=1)
ax.set_title('Goal probability')
fig.colorbar(pos, ax=ax)
plt.xlim((0,105))
plt.ylim((0,68))
plt.gca().set_aspect('equal', adjustable='box')
for i in range (1, 16):
    x1, y1 = i*105/16, 0
    x2, y2 = i*105/16, 68
```

```
    ax.plot([x1, x2], [y1, y2], color = 'grey', alpha = 0.1)


for i in range (1, 12):
    x1, y1 = 0, i*68/12
    x2, y2 = 105, i*68/12
    ax.plot([x1, x2], [y1, y2], color = 'grey', alpha = 0.1)
plt.show()
fig.savefig("goal_xT.png", dpi=None, bbox_inches="tight")


(fig,ax) = createPitch(105,68,'meters','gray')
import matplotlib.pyplot as plt
pos=ax.imshow(pshot_2d.T, extent=[0,105,68,0], aspect='auto',cmap=plt.cm.Blues,vmin=0, vmax=1)
fig.colorbar(pos, ax=ax)
ax.set_title('Shot probability')
plt.xlim((0,105))
plt.ylim((0,68))
plt.gca().set_aspect('equal', adjustable='box')
for i in range (1, 16):
    x1, y1 = i*105/16, 0
    x2, y2 = i*105/16, 68
    ax.plot([x1, x2], [y1, y2], color = 'grey', alpha = 0.1)


for i in range (1, 12):
    x1, y1 = 0, i*68/12
    x2, y2 = 105, i*68/12
    ax.plot([x1, x2], [y1, y2], color = 'grey', alpha = 0.1)
plt.show()
fig.savefig("shot_probab.png", dpi=None, bbox_inches="tight")

(fig,ax) = createPitch(105,68,'meters','gray')
import matplotlib.pyplot as plt
pos=ax.imshow(pmove_2d.T, extent=[0,105,68,0], aspect='auto',cmap=plt.cm.Greens,vmin=0, vmax=1)
fig.colorbar(pos, ax=ax)
ax.set_title('Move probability')
plt.xlim((0,105))
plt.ylim((0,68))
plt.gca().set_aspect('equal', adjustable='box')
for i in range (1, 16):
    x1, y1 = i*105/16, 0
    x2, y2 = i*105/16, 68
    ax.plot([x1, x2], [y1, y2], color = 'grey', alpha = 0.1)


for i in range (1, 12):
    x1, y1 = 0, i*68/12
    x2, y2 = 105, i*68/12
    ax.plot([x1, x2], [y1, y2], color = 'grey', alpha = 0.1)
plt.show()
fig.savefig("move_probab.png", dpi=None, bbox_inches="tight")

pxT_2d=np.zeros((16,12))
for x in range(16):
    for y in range(12):
        pxT_2d[x, y] = xT[(12*(x+1) - y)-1]



(fig,ax) = createPitch(105,68,'meters','gray')
import matplotlib.pyplot as plt
pos=ax.imshow(pxT_2d.T, extent=[0,105,68,0], aspect='auto',cmap=plt.cm.Purples,vmin=0, vmax=0.5)
fig.colorbar(pos, ax=ax)
ax.set_title('xT value for each zone')
plt.xlim((0,105))
plt.ylim((0,68))
plt.gca().set_aspect('equal', adjustable='box')
for i in range (1, 16):
    x1, y1 = i*105/16, 0
    x2, y2 = i*105/16, 68
    ax.plot([x1, x2], [y1, y2], color = 'grey', alpha = 0.1)


for i in range (1, 12):
    x1, y1 = 0, i*68/12
```

```
    x2, y2 = 105, i*68/12
    ax.plot([x1, x2], [y1, y2], color = 'grey', alpha = 0.1)
plt.show()
fig.savefig("xT_probab.png", dpi=None, bbox_inches="tight")
```

Valuing Actions by Estimating Probabilities Calculations

```
from io import BytesIO
import pandas as pd  # version 1.0.3
from socceraction.spadl.wyscout import convert_to_spadl

def open_file(filename):
    with open(filename, 'rb') as json_file:
        return BytesIO(json_file.read()).getvalue().decode('unicode_escape')


teams = open_file('Wyscout/teams.json')
teams = pd.read_json(teams)
teams.to_hdf('wyscout.h5', key='teams', mode='w')

events = open_file('Wyscout/events_Germany.json')
germany = pd.read_json(events)
events = open_file('Wyscout/events_England.json')
england = pd.read_json(events)
events = open_file('Wyscout/events_Spain.json')
spain = pd.read_json(events)
events = open_file('Wyscout/events_Italy.json')
italy = pd.read_json(events)
events = open_file('Wyscout/events_France.json')
france = pd.read_json(events)
events = open_file('Wyscout/events_World_Cup.json')
world_cup = pd.read_json(events)
world_cup = world_cup.loc[world_cup["matchPeriod"] != "P"]
events = pd.concat([germany, england, spain, italy, france, world_cup])
groupby_game = events.groupby(["matchId"], as_index = False)

for i, game in groupby_game:
    game.to_hdf('wyscout.h5', key=f'events/match_{i}', mode='a')


players = open_file('Wyscout/players.json')
players = pd.read_json(players)
players.to_hdf('wyscout.h5', key='players', mode='a')

matches = open_file('Wyscout/matches_Germany.json')
germany = pd.read_json(matches)
matches = open_file('Wyscout/matches_England.json')
england = pd.read_json(matches)
matches = open_file('Wyscout/matches_Spain.json')
spain = pd.read_json(matches)
matches = open_file('Wyscout/matches_Italy.json')
italy = pd.read_json(matches)
matches = open_file('Wyscout/matches_France.json')
france = pd.read_json(matches)
matches = open_file('Wyscout/matches_World_Cup.json')
world_cup = pd.read_json(matches)


matches = pd.concat([germany, england, spain, italy, france, world_cup])
matches.to_hdf('wyscout.h5', key='matches', mode='a')

#convert to spadl
convert_to_spadl('wyscout.h5', 'spadl.h5')

import pandas as pd
import numpy as np


df_teams = pd.read_hdf('spadl.h5', key='teams')
df_players = pd.read_hdf('spadl.h5', key='players')
df_games = pd.read_hdf('spadl.h5', key='games')
```

```
def add_action_type_dummies(df_actions):
    return df_actions.merge(pd.get_dummies(df_actions['type_name']), how='left', left_index=True, right_index=True)


def time_played(df):
   df["time_played"]  = df['time_seconds'] + (df['period_id'] >= 2) * (45 * 60) + (df['period_id'] >= 3) * (15 * 60) + (df['
        period_id'] == 4) * (15 * 60)
   return df


def features_delay (df):
    features_to_delay = ['period_id', 'time_seconds', 'team_id',
                    'player_id', 'start_x', 'start_y', 'end_x', 'end_y', 'bodypart_id',
                    'type_id', 'result_id', 'type_name', 'result_name', 'bodypart_name',
                    'time_played']
    lista = []
    for i in range(0, 3):
        lista.append(df[features_to_delay].shift(i).add_suffix(f'-{i}'))
    full_df = pd.concat(lista, axis=1)
    return full_df

def same_team(df):
    for i in range(1, 3):
        df[f'team-{i}'] = df['team_id-0'] == df[f'team_id-{i}']
    return df

def invert_coordinates(df):
    for step in range(1, 3):
        df.loc[~(df[f'team-{step}']), f'start_x-{step}'] = 105 - df[f'start_x-{step}']
        df.loc[~(df[f'team-{step}']), f'start_y-{step}'] = 68 - df[f'start_y-{step}']
        df.loc[~(df[f'team-{step}']), f'end_x-{step}'] = 105 - df[f'end_x-{step}']
        df.loc[~(df[f'team-{step}']), f'end_y-{step}'] = 68 - df[f'end_y-{step}']
    return df

def location(df):
    for step in range(0, 3):
            x = 105 - df[f'start_x-{step}']
            x = 105 - df[f'end_x-{step}']
            y = abs(34 - df[f'start_y-{step}'])
            y = abs(34 - df[f'end_y-{step}'])
            df[f'start_distance_to_goal-{step}'] = np.sqrt(x ** 2 + y ** 2)
            df[f'end_distance_to_goal-{step}'] = np.sqrt(x ** 2 + y ** 2)

            df[f'start_angle_to_goal-{step}'] = np.divide(x, y, out = np.zeros_like(x), where = (y != 0))
            df[f'end_angle_to_goal-{step}'] = np.divide(x, y, out = np.zeros_like(x), where=(y != 0))

            df[f'diff_x-{step}'] = df[f'end_x-{step}'] - df[f'start_x-{step}']
            df[f'diff_y-{step}'] = df[f'end_y-{step}'] - df[f'start_y-{step}']
            df[f'distance_covered-{step}'] = np.sqrt((df[f'end_x-{step}'] - df[f'start_x-{step}']) ** 2 + (df[f'end_y-{step}']
                  - df[f'start_y-{step}']) ** 2)


def create_features_match(df):
    df_action_features = add_action_type_dummies(df)
    time_played(df_action_features)
    df_gamestate_features = features_delay(df_action_features)
    same_team(df_gamestate_features)
    invert_coordinates(df_gamestate_features)
    location(df_gamestate_features)
    return df_gamestate_features

def scores(df):

    goals = df['type_name'].str.contains('shot') & (df['result_id'] == 1)
    owngoals = df['type_name'].str.contains('shot') & (df['result_id'] == 2)
    y = pd.concat([goals, owngoals, df['team_id']], axis=1)
    y.columns = ['goal', 'owngoal', 'team_id']

    for i in range(1, 10):
        for col in ['team_id', 'goal', 'owngoal']:
            shifted = y[col].shift(-i)
            shifted[-i:] = y[col][len(y) - 1]
            y[f'{col}+{i}'] = shifted

    scores = y['goal']
```

```
    for i in range(1, 10):
        goal_scored = y[f'goal+{i}'] & (y[f'team_id+{i}'] == y['team_id'])
        own_goal_opponent = y[f'owngoal+{i}'] & (y[f'team_id+{i}'] != y['team_id'])
        scores = scores | goal_scored | own_goal_opponent

    result_df = pd.DataFrame(scores, columns=['scores'])
    return result_df

def concedes(df):

    goals = df['type_name'].str.contains('shot') & (df['result_id'] == 1)
    owngoals = df['type_name'].str.contains('shot') & (df['result_id'] == 2)
    y = pd.concat([goals, owngoals, df['team_id']], axis = 1)
    y.columns = ['goal', 'owngoal', 'team_id']

    for i in range(1, 10):
        for col in ['team_id', 'goal', 'owngoal']:
            shifted = y[col].shift(-i)
            shifted[-i:] = y[col][len(y) - 1]
            y[f'{col}+{i}'] = shifted

    concedes = y['owngoal']
    for i in range(1, 10):
        goal_opponent = y[f'goal+{i}'] & (y[f'team_id+{i}'] != y['team_id'])
        own_goal_team = y[f'owngoal+{i}'] & (y[f'team_id+{i}'] == y['team_id'])
        concedes = concedes | goal_opponent | own_goal_team

    return pd.DataFrame(concedes, columns=['concedes'])


for i, game in df_games.iterrows():
    game_id = game['game_id']
    with pd.HDFStore('spadl.h5') as spadlstore:
        df_actions = spadlstore[f'actions/game_{game_id}']
        df_actions = (
            df_actions.merge(spadlstore['actiontypes'], how='left')
            .merge(spadlstore['results'], how='left')
            .merge(spadlstore['bodyparts'], how='left')
            .reset_index(drop=True)
        )

    df_features = create_features_match(df_actions)
    df_features.to_hdf('features.h5', f'game_{game_id}')

    df_labels = pd.concat([scores(df_actions), concedes(df_actions)], axis=1)
    df_labels.to_hdf('labels.h5', f'game_{game_id}')

df_games = pd.read_hdf('spadl.h5', key='games')


dfs_features = []
for i,game in df_games.iterrows():
    game_id = game['game_id']
    df_features = pd.read_hdf('features.h5', key=f'game_{game_id}')
    df_features['game_id'] = game_id
    dfs_features.append(df_features)
df_features = pd.concat(dfs_features).reset_index(drop=True)

dfs_labels = []
for i, game in df_games.iterrows():
    game_id = game['game_id']
    df_labels = pd.read_hdf('labels.h5', key=f'game_{game_id}')
    df_labels['game_id'] = game_id
    dfs_labels.append(df_labels)
df_labels = pd.concat(dfs_labels).reset_index(drop=True)

import pandas as pd
from xgboost import XGBClassifier, plot_importance
from socceraction.vaep.formula import value

df_games = pd.read_hdf('spadl.h5', key='games')

dfs_features = []
for i, game in df_games.iterrows():
    game_id = game['game_id']
    df_features = pd.read_hdf('features.h5', key=f'game_{game_id}')
```

```
    df_features['game_id'] = game_id
    dfs_features.append(df_features)
df_features = pd.concat(dfs_features).reset_index(drop=True)

dfs_labels = []
for i, game in df_games.iterrows():
    game_id = game['game_id']
    df_labels = pd.read_hdf('labels.h5', key=f'game_{game_id}')
    df_labels['game_id'] = game_id
    dfs_labels.append(df_labels)
df_labels = pd.concat(dfs_labels).reset_index(drop=True)

df_games_train = df_games[df_games["competition_id"] != 28]
df_games_test = df_games[df_games["competition_id"] == 28]

X_train = df_features[df_features['game_id'].isin(df_games_train["game_id"])]
X_test = df_features[df_features['game_id'].isin(df_games_test["game_id"])]
y_train = df_labels[df_labels['game_id'].isin(df_games_train["game_id"])]
y_test = df_labels[df_labels['game_id'].isin(df_games_test["game_id"])]



features = ['start_x-0', 'start_y-0', 'end_x-0', 'end_y-0','type_id-0',
       'result_id-0', 'bodypart_id-0', 'time_played-0',
       'start_x-1', 'start_y-1', 'end_x-1', 'end_y-1',
       'type_id-1', 'result_id-1',
       'bodypart_id-1', 'time_played-1',
       'start_x-2', 'start_y-2',
       'end_x-2', 'end_y-2',
       'type_id-2', 'result_id-2', 'bodypart_id-2', 'time_played-2',
       'start_distance_to_goal-0', 'start_angle_to_goal-0', 'diff_x-0',
       'diff_y-0', 'distance_covered-0',
       'end_distance_to_goal-0', 'end_angle_to_goal-0',
       'start_distance_to_goal-1', 'start_angle_to_goal-1',
       'diff_x-1', 'diff_y-1', 'distance_covered-1', 'end_distance_to_goal-1', 'end_angle_to_goal-1',
       'start_distance_to_goal-2',
       'start_angle_to_goal-2', 'diff_x-2', 'diff_y-2', 'distance_covered-2',
       'end_distance_to_goal-2',
       'end_angle_to_goal-2']



scoring_model = XGBClassifier(n_estimators=100, max_depth=4, enable_categroical = True, n_jobs = -1)
scoring_model.fit(X_train[features], y_train['scores'])
plot_importance(scoring_model)

scoring_probability = scoring_model.predict_proba(X_test[features])
cont_predict = scoring_probability[:, 1]

from sklearn.calibration import calibration_curve
c1 = calibration_curve(y_test["scores"], cont_predict, n_bins = 10)


conceding_model = XGBClassifier(n_estimators=100, max_depth=4, enable_categroical = True, n_jobs = -1)
conceding_model.fit(X_train[features], y_train['concedes'])
plot_importance(conceding_model)

conceding_probability = conceding_model.predict_proba(X_test[features])
cont_predict2 = conceding_probability[:, 1]
c2 = calibration_curve(y_test["concedes"], cont_predict2, n_bins = 10)

from sklearn.metrics import brier_score_loss

brier_score_loss(y_test["scores"], cont_predict)
brier_score_loss(y_test["concedes"], cont_predict2)

scoring_predicitons = pd.DataFrame(cont_predict, index = X_test.index, columns = ["scores"])
conceding_predicitons = pd.DataFrame(cont_predict2, index = X_test.index, columns = ["concedes"])

predictions_df = pd.concat([scoring_predicitons, conceding_predicitons], axis=1).reset_index(drop = True)



df_players = pd.read_hdf('spadl.h5', key='players')
df_teams = pd.read_hdf('spadl.h5', key='teams')
```

```
dfs_actions = []
for i, row in df_games_test.iterrows():
    game_id = row['game_id']
    with pd.HDFStore('spadl.h5') as spadlstore:
        df_actions = spadlstore[f'actions/game_{game_id}']
        df_actions = (
            df_actions.merge(spadlstore['actiontypes'], how='left')
            .merge(spadlstore['results'], how='left')
            .merge(spadlstore['bodyparts'], how='left')
            .merge(spadlstore['players'], how='left')
            .merge(spadlstore['teams'], how='left')
            .reset_index()
            .rename(columns={'index': 'action_id'})
        )

    dfs_actions.append(df_actions)
df_actions = pd.concat(dfs_actions).reset_index(drop = True)


df_actions_predictions = pd.concat([df_actions, predictions_df], axis=1)
dfs_values = []
for game_id, game_predictions in df_actions_predictions.groupby('game_id'):
    df_values = value(game_predictions, game_predictions['scores'], game_predictions['concedes'])

    df_all = pd.concat([game_predictions, df_values], axis=1)
    dfs_values.append(df_all)


df_values = pd.concat(dfs_values)
df_values = df_values.sort_values(['game_id', 'period_id', 'time_seconds'])
df_values = df_values.reset_index(drop=True)



ranking = df_values[['player_id', 'team_name', 'short_name', 'vaep_value']].groupby(['player_id', 'team_name', 'short_name'])
ranking = ranking.agg(vaep_count=('vaep_value', 'count'), vaep_mean=('vaep_value', 'mean'), vaep_sum=('vaep_value', 'sum'))
ranking = ranking.sort_values('vaep_sum', ascending=False)
ranking = ranking.reset_index()



df_player_games = pd.read_hdf('spadl.h5', 'player_games')
df_player_games = df_player_games[df_player_games['game_id'].isin(df_games_test['game_id'])]

#adding extra time minutes
df_player_games.loc[(df_player_games["game_id"] == 2058015) & (df_player_games["player_id"].isin([12829, 210044, 69964, 69411,
       69400, 14771])), "minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058015) & (df_player_games["
      player_id"].isin([12829, 210044, 69964, 69411, 69400, 14771]))]["minutes_played"].apply(lambda x: x+26)
df_player_games.loc[(df_player_games["game_id"] == 2058015) & (df_player_games["player_id"].isin([9380, 8653, 8945, 8717,
      13484, 10131, 7934, 69409, 25393, 135747, 3476, 69396, 69968, 14812, 397178, 8292])), "minutes_played"] =
      df_player_games[(df_player_games["game_id"] == 2058015) & (df_player_games["player_id"].isin([9380, 8653, 8945, 8717,
      13484, 10131, 7934, 69409, 25393, 135747, 3476, 69396, 69968, 14812, 397178, 8292]))]["minutes_played"].apply(lambda x:
      x+30)


df_player_games.loc[(df_player_games["game_id"] == 2058012) & (df_player_games["player_id"].isin([14771, 101590])), "
      minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058012) & (df_player_games["player_id"].isin([14771,
      101590]))]["minutes_played"].apply(lambda x: x+25)
df_player_games.loc[(df_player_games["game_id"] == 2058012) & (df_player_games["player_id"].isin([103668, 220971, 41123,
      103682, 101647, 101576, 101583, 101953, 101707, 102157, 25393, 135747, 14943, 3476, 8287, 69396, 69616, 69968, 69964,
      69404])), "minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058012) & (df_player_games["player_id"].
      isin([103668, 220971, 41123, 103682, 101647, 101576, 101583, 101953, 101707, 102157, 25393, 135747, 14943, 3476, 8287,
      69396, 69616, 69968, 69964, 69404]))]["minutes_played"].apply(lambda x: x+30)

df_player_games.loc[(df_player_games["game_id"] == 2058009) & (df_player_games["player_id"].isin([3531, 8292, 397178])), "
      minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058009) & (df_player_games["player_id"].isin([3531,
      8292, 397178]))]["minutes_played"].apply(lambda x: x+24)
df_player_games.loc[(df_player_games["game_id"] == 2058009) & (df_player_games["player_id"].isin([9380, 8653, 8945, 8717,
      7964, 10131, 7934, 210044, 12829, 246928, 25662, 257762, 91702, 256634, 20751, 3450, 37831, 91502, 20764])), "
      minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058009) & (df_player_games["player_id"].isin([9380,
      8653, 8945, 8717, 7964, 10131, 7934, 210044, 12829, 246928, 25662, 257762, 91702, 256634, 20751, 3450, 37831, 91502,
      20764]))]["minutes_played"].apply(lambda x: x+30)

df_player_games.loc[(df_player_games["game_id"] == 2058005) & (df_player_games["player_id"].isin([241945, 56025, 69411,
```

```
      69400])), "minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058005) & (df_player_games["player_id"].
      isin([241945, 56025, 69411, 69400]))]["minutes_played"].apply(lambda x: x+27)
df_player_games.loc[(df_player_games["game_id"] == 2058005) & (df_player_games["player_id"].isin([55957, 56394, 8480, 54,
      55979, 56274, 20433, 405, 15080, 69409, 25393, 135747, 3476, 8287, 69396, 69616, 69404, 69964])), "minutes_played"] =
      df_player_games[(df_player_games["game_id"] == 2058005) & (df_player_games["player_id"].isin([55957, 56394, 8480, 54,
      55979, 56274, 20433, 405, 15080, 69409, 25393, 135747, 3476, 8287, 69396, 69616, 69404, 69964]))]["minutes_played"].
      apply(lambda x: x+30)

df_player_games.loc[(df_player_games["game_id"] == 2058004) & (df_player_games["player_id"].isin([101953, 70129])), "
      minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058004) & (df_player_games["player_id"].isin([101953,
       70129]))]["minutes_played"].apply(lambda x: x+26)
df_player_games.loc[(df_player_games["game_id"] == 2058004) & (df_player_games["player_id"].isin([103668, 41123, 257800,
      103682, 101647, 101576, 101583, 101682, 4513, 101707, 7910, 3443, 3306, 3346, 3341, 3269, 3563, 3353, 4501, 3840])), "
      minutes_played"] = df_player_games[(df_player_games["game_id"] == 2058004) & (df_player_games["player_id"].isin([103668,
       41123, 257800, 103682, 101647, 101576, 101583, 101682, 4513, 101707, 7910, 3443, 3306, 3346, 3341, 3269, 3563, 3353,
      4501, 3840]))]["minutes_played"].apply(lambda x: x+30)

minutes = df_player_games[['player_id', 'minutes_played']]
minutes = minutes.groupby('player_id').sum().reset_index()

per90 = ranking.merge(minutes)
per90['vaep_rating'] = per90['vaep_sum'] * 90 / per90['minutes_played']
per90['actions_p90'] = per90['vaep_count'] * 90 / per90['minutes_played']

per90 = per90[per90['minutes_played']>150]
per90 = per90.sort_values('vaep_rating', ascending=False)

import matplotlib.pyplot as plt
fop1, mpv1 = c1
# plot perfectly calibrated
plt.plot([0, 1], [0, 1], linestyle='--')
# plot model reliability
plt.plot(mpv1, fop1, marker='.')
plt.show()

fop2, mpv2 = c2
# plot perfectly calibrated
plt.plot([0, 1], [0, 1], linestyle='--')
# plot model reliability
plt.plot(mpv2, fop2, marker='.')
plt.show()

from sklearn.calibration import CalibratedClassifierCV
calibrated = CalibratedClassifierCV(scoring_model, method='sigmoid', cv=5)
calibrated.fit(X_train[features], y_train['scores'])
# predict probabilities
probs = calibrated.predict_proba(X_test[features])[:, 1]
# reliability diagram
fop3, mpv3 = calibration_curve(y_test['scores'], probs, n_bins=10, normalize=True)
# plot perfectly calibrated
plt.plot([0, 1], [0, 1], linestyle='--')
# plot calibrated reliability
plt.plot(mpv3, fop3, marker='.')
plt.show()

calibrated2 = CalibratedClassifierCV(conceding_model, method='sigmoid', cv=5)
calibrated2.fit(X_train[features], y_train['concedes'])
# predict probabilities
probs = calibrated.predict_proba(X_test[features])[:, 1]
# reliability diagram
fop4, mpv4 = calibration_curve(y_test['concedes'], probs, n_bins=10, normalize=True)
# plot perfectly calibrated
plt.plot([0, 1], [0, 1], linestyle='--')
# plot calibrated reliability
plt.plot(mpv4, fop4, marker='.')
plt.show()




plt.figure(figsize = (20, 20))
plt.tick_params(axis='both', which='major', labelsize=6)
fig, axs = plt.subplots(2, 2, figsize = (5, 5))
axs[0, 0].plot([0, 1], [0, 1], linestyle='--')
axs[0,0].plot(mpv1, fop1, marker='.')
```

```
axs[0, 0].set_title('Scoring model', fontsize = 10)
axs[0, 0].set_xlabel("Predicted probability", fontsize = 8)
axs[0, 0].set_ylabel("True probability", fontsize = 8)
axs[0, 1].plot([0, 1], [0, 1], linestyle='--')
axs[0,1].plot(mpv2, fop2, marker='.')
axs[0, 1].set_title('Conceding model', fontsize = 10)
axs[0, 1].set_xlabel("Predicted probability", fontsize = 8)
axs[0, 1].set_ylabel("True probability", fontsize = 8)

axs[1, 0].plot([0, 1], [0, 1], linestyle='--')


axs[1,0].plot(mpv3, fop3, marker='.')
axs[1, 0].set_title('Scoring model (after calibration)', fontsize = 10)
axs[1, 0].set_xlabel("Predicted probability", fontsize = 8)
axs[1, 0].set_ylabel("True probability", fontsize = 8)
axs[1, 1].plot([0, 1], [0, 1], linestyle='--')
axs[1,1].plot(mpv4, fop4, marker='.')
axs[1, 1].set_title('Conceding model (after calibration)', fontsize = 10)
axs[1, 1].set_xlabel("Predicted probability", fontsize = 8)
axs[1, 1].set_ylabel("True probability", fontsize = 8)
plt.tight_layout()
plt.show()

fig, axs = plt.subplots(2, 1, figsize = (5, 5))
axs[0].hist(scoring_predicitons)
axs[0].set_title("Scoring model")
axs[0].set_xlabel("Scoring probability")
axs[0].set_ylabel("Number of observations")
axs[1].hist(conceding_predicitons)
axs[1].set_title("Conceding model")
axs[1].set_xlabel("Conceding probability")
axs[1].set_ylabel("Number of observations")
plt.tight_layout()
```

```
hist_list = ['start_x-0', 'start_y-0', 'end_x-0', 'end_y-0','type_name-0',
       'result_id-0', 'bodypart_name-0', 'time_played-0',
       'start_distance_to_goal-0', 'start_angle_to_goal-0', 'diff_x-0',
       'diff_y-0', 'distance_covered-0',
       'end_distance_to_goal-0', 'end_angle_to_goal-0']
```

```
for i in hist_list:
    plt.hist(X_train[i], bins = 100)
    plt.title(i)
    plt.xticks(fontsize=12, rotation=90)
    plt.savefig("grafiki_licencjat/train" + i + ".png")
    plt.show()

for i in hist_list:
    plt.hist(X_test[i], bins = 100)
    plt.title(i)
    plt.xticks(fontsize=12, rotation=90)
    plt.savefig("grafiki_licencjat/test" + i + ".png")
    plt.show()
```

Appendix 1

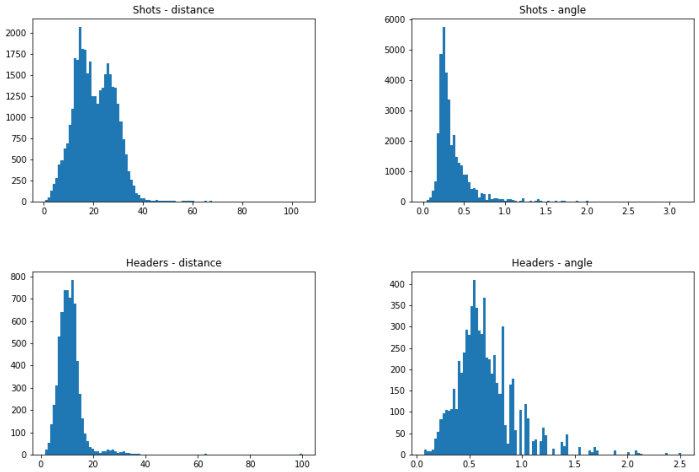
Descriptive Statistics of Logistic Regression Variables

Table A.1: Descriptive Statistics of Variables used in Expected Goals Models

	Shots - distance [m]	Shots - angle	Headers - distance [m]	Headers - angle
mean	20.54	0.36	11.05	0.64
std	8.22	0.22	5.09	0.28
min	0.68	0.02	1.72	0.07
25%	14.41	0.24	8.43	0.47
50%	19.94	0.29	10.58	0.60
75%	26.57	0.41	12.77	0.76
max	103.95	3.14	99.75	2.51

Source: own elaboration, Python (pandas, numpy, statsmodels packages)

Fig A.1: Histograms of Variables Used in the Expected Goals Models - Normal Shots and Headers



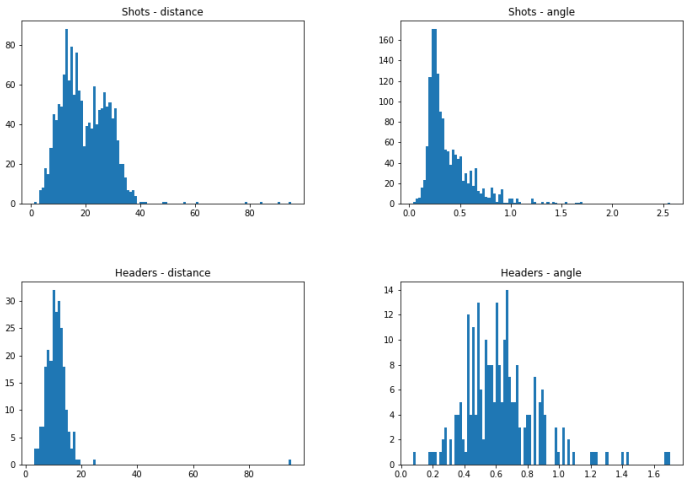
Source: own elaboration, Python (pandas, numpy, statsmodels, matplotlib packages)

Table A.2: Descriptive Statistics of Variables Used to Calculate Expected Goals Values

	Shots - distance [m]	Shots - angle	Headers - distance [m]	Headers - angle
mean	19.73	0.39	11.22	0.64
std	8.90	0.23	6.25	0.23
min	1.25	0.04	3.22	0.08
25%	12.90	0.24	8.76	0.48
50%	18.28	0.31	10.70	0.61
75%	26.49	0.47	12.80	0.73
max	94.99	2.56	94.99	1.70

Source: own elaboration, Python (pandas, numpy, statsmodels packages)

Fig A.2: Histograms of variables used to calculate the Expected Goals Value - normal shots and headers



Source: own elaboration, Python (pandas, numpy, statsmodels, matplotlib packages)

Appendix 2

Descriptive Statistics of the VAEP Models Features

Fig. A.3: Descriptive Statistics of Train Data Set

Index	start_x-0	start_y-0	end_x-0	end_y-0	time_played-0	start_distance_to_goal-0	start_angle_to_goal-0	diff_x-0	diff_y-0	distance_covered-0	end_distance_to_goal-0	end_angle_to_goal-0
count	2.32148e+06	2.32148e+06	2.32148e+06	2.32148e+06	2.32148e+06	2.32148e+06	2.32148e+06	2.32148e+06	2.32148e+06	2.32148e+06	2.32148e+06	2.32148e+06
mean	53.5538	34.1272	53.2835	34.2902	2716.74	56.3486	6.43648	-0.270314	0.163044	19.1061	56.3486	6.43648
std	25.529	20.1631	25.9204	20.1936	1613.52	24.0658	13.5531	18.6779	15.6771	15.1555	24.0658	13.5531
min	0	-0.68	0	-0.68	0.02	0	0	-105	-68	0	0	0
25%	33.6	16.32	33.6	17	1315.94	38.0196	1.60588	-8.4	-8.16	9.13854	38.0196	1.60588
50%	54.6	34	54.6	34	2718.23	54.8968	2.79129	0	0	15.9031	54.8968	2.79129
75%	73.5	51.68	73.5	51.68	4083.44	74.4178	5.12366	8.4	8.84	25.2377	74.4178	5.12366
max	105	68	105	68	6234.78	110.368	154.412	105	68	125.096	110.368	154.412

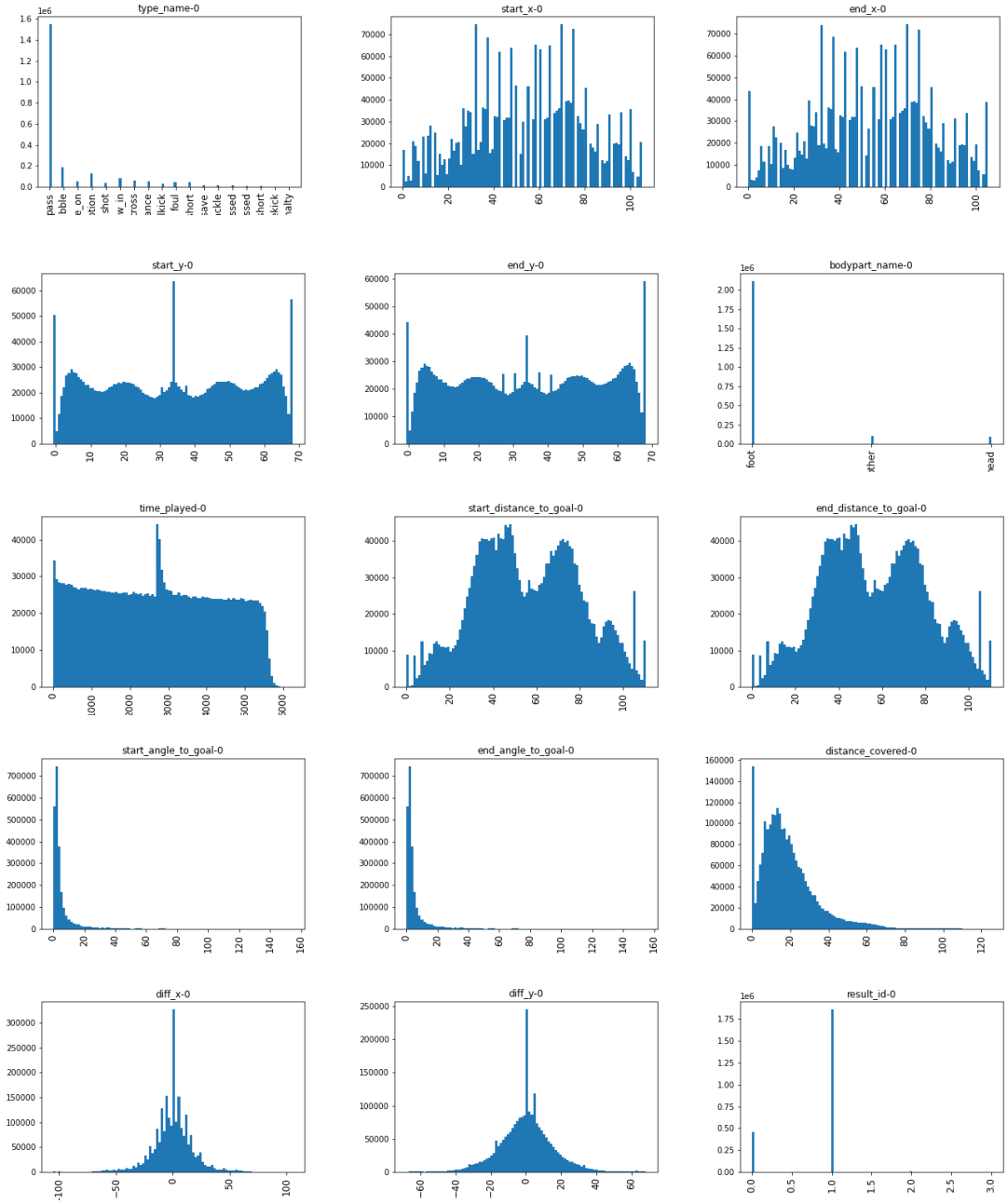
Source: own elaboration, Python (matplotlib, pandas, socceraction packages)

Fig. A.4: Descriptive Statistics of Test Data Set

Index	start_x-0	start_y-0	end_x-0	end_y-0	time_played-0	start_distance_to_goal-0	start_angle_to_goal-0	diff_x-0	diff_y-0	distance_covered-0	end_distance_to_goal-0	end_angle_to_goal-0
count	80928	80928	80928	80928	80928	80928	80928	80928	80928	80928	80928	80928
mean	53.0947	34.3141	52.9363	34.4695	2792.02	56.4875	6.50698	-0.158346	0.155382	18.3995	56.4875	6.50698
std	25.107	19.8867	25.4417	19.8723	1622.86	23.7105	13.3034	17.6212	14.9859	14.0211	23.7105	13.3034
min	0	0	0	0	0.122553	0	0	-105	-67.32	0	0	0
25%	33.6	17	33.6	17	1391.89	38.4008	1.68449	-8.4	-8.16	9.06201	38.4008	1.68449
50%	53.55	34	53.55	34	2775.24	55.65	2.85428	0	0	15.608	55.65	2.85428
75%	72.45	51.68	72.45	51.68	4181.5	74.2044	5.22059	8.4	8.84	24.389	74.2044	5.22059
max	105	68.68	105	68.68	5958.37	110.368	154.412	98.7	68	125.096	110.368	154.412

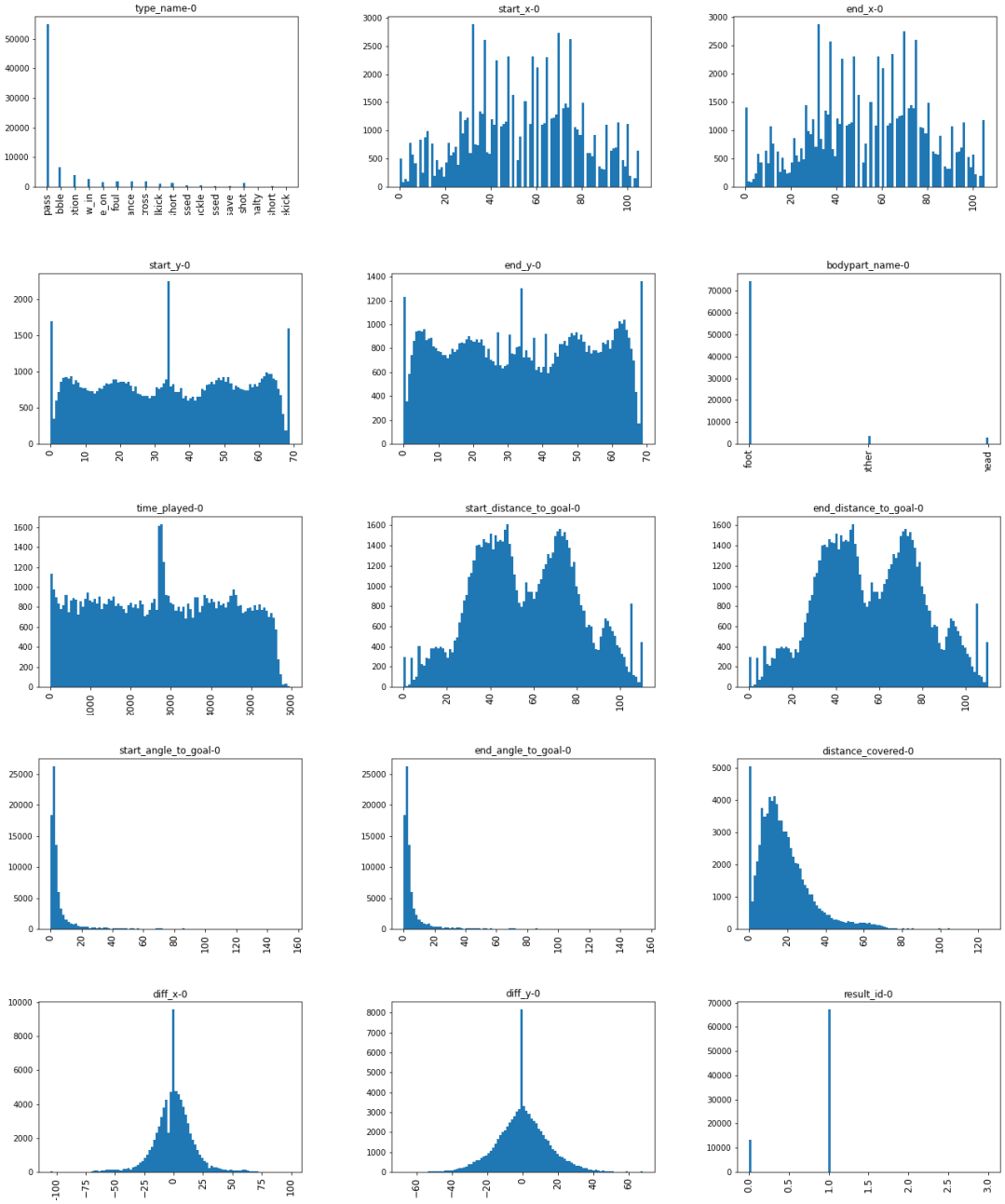
Source: own elaboration, Python (matplotlib, pandas, socceraction packages)

Fig A.4: Histograms of Variables Used to Train Scoring and Conceding Models



Source: own elaboration, Python (pandas, numpy, socceraction, matplotlib packages)

Fig A.5: Histograms of Variables Used to Calculate VAEP values



Source: own elaboration, Python (pandas, numpy, socceraction, matplotlib packages)

Appendix 3

Players' Positions during the 2018 FIFA World Cup

Goalkeepers - I. Akinfeev, A. Al.-Muaiouf, M. El-Shenawy, F. Muslera, M. El-Kajoui, A. Beiranvand, R. Patricio, D. de Gea, H. Lloris, M. Ryan, W. Caballero, H. Halldorsson, P. Gallese, K. Schmeichel, D. Subasic, F. Uzoho, K. Navas, V. Stojković, M. Neuer, G. Ochoa, Alisson, Y. Sommer, R. Olsen, H. Jo, T. Courtois, J. Penedo, M. Hassen, J. Pickford, D. Ospina, E. Kawashima, W. Szczesny, K. Ndiaye, H. Alowais, Y. Al.-Mosailem.

Right backs - M. Fernandes, M. Al-Burayk, A. Fathi, G. Valera, Cedric, Nacho, B. Pavard, J. Ridson, E. Salvio, B. Saevarsson, L. Advincula, H. Dalsgaard, S. Vrsaljko, A. Shehu, C. Gamboa, B. Ivanović, J. Kimmich, C. Salcedo, Danilo, S. Lichtsteiner, M. Lustig, Y. Lee, M. Murillo, D. Bronn, S. Arias, H. Sakai, Ł. Piszczek, M. Wague, N. Dirar, R. Rezaeian, Fagner, D. Carvajal, A. Rukavina, E. Alvarez, A. Machado, B. Bereszyński, R. Pereira.

Centre backs - I. Kutepov, Os. Hawsawi, Om. Hawsawi, A. Gabr, A. Hegazy, M. Abdelshafy, J. Gimenez, D. Godin, M. Benatia, G. Saiss, A. Hakimi, M. Pouraliganji, R. Cheshmi, R. Rezaeian, Pepe, J. Fonte, G. Pique, S. Ramos, R. Varane, S. Umtiti, T. Sainsbury, M. Milligan, N. Otamendi, M. Rojo, K. Arnason, R. Sigurdsson, A. Rodriguez, C. Ramos, S. Kjaer, A. Christensen, D. Lovren, D. Vida, L. Balogun, W. Ekong, J. Acosta, G. Gonzalez, O. Duarte, N. Milenković, D. Tosić, J. Boateng, M. Hummels, H. Ayala, H. Moreno, Thiago Silva, Marquinhos, M. Akanji, F. Schaer, A. Granqvist, P. Jansson, H. Jang, Y. Kim, J. Verthongen, D. Boyata, T. Alderweirerld, R. Torres, F. Escobar, S. Ben Youssef, Y. Meriah, K. Walker, J. Stones, H. Maguire, D. Sanchez, O. Murillo, M. Yoshida, G. Shoji, T. Cionek, M. Pazdan, K. Koulibaly, S. Sane, M. de Costa, A. Al-Bulayhi, N. Tagliafico, G. Mercado, K. Omeruo, A. Rudiger, J. Bednarek, Y. Mina, M. Hawsawi, A. Santamaria, M. Veljković, Y. Yun, N. Sule, G. Cahill, P. Jones, L. Dendoncker, T. Vermaelen, L. Ovalle, T. Makino, K. Glik, S. Ignashevich, F. Kudriashov, M. Jorgensen, F. Luis, C. Salcedo, V. Kompany, V. Lindelof, Miranda.

Left backs - Y. Zhirkov, Y. Al-Shahrani, M. Abdelshafy, M. Caceres, R. Guerreiro, J. Alba, L. Hernandez, A. Behich, N. Tagliafico, H. Magnusson, M. Trauco, J. Larsen, I. Strinić, B. Idowu, F. Calvo, A. Koralov, M. Plattenhardt, J. Gallardo, Marcelo, R. Rodriguez, L. Augustinsson, J. Park, E. Davis, A. Maaloul, J. Mojica, Y. Nagatomo, M. Rybus, Y. Sabaly, A. Hakimi,

E. Haji Safi, B. Oviedo, M. Kim, J. Hector, C. Hong, A. Jędrzejczyk, D. Laxalt, F. Kudriashov.

Central defensive midfielders - R. Zobnin, I. Gazinsky, S. Al-Faraj, T. Al-Jassam, A. Otayf, T. Hemed, M. Elneny, R. Betancur, M. Vecino, K. El-Ahmadi, H. Ziyach, E. Haji Safi, O. Ebrahimi, W. Carvalho, J. Moutinho, Koke, S. Busquets, P. Pogba, C. Tolisso, N. Kante, M. Jedinak, A. Mooy, J. Mascherano, L. Biglia, E. Hallfredson, A. Gunnarsson, R. Tapia, Y. Yotun, W. Kvist, T. Delaney, L. Modrić, I. Rakitić, W. Ndidi, O. Etebo, C. Borges, D. Guzman, N. Matić, L. Milivojević, S. Khedira, T. Kroos, H. Herrera, A. Guadrado, Casemiro, Paulinho, V. Behrami, G. Xhaka, S. Larsson, A. Ekdal, J. Lee, S. Ki, J. Koo, K. de Bruyne, A. Witsel, A. Godoy, A. Cooper, G. Gomez, A. Badri, E. Skhiri, F. Sassi, J. Henderson, C. Sanchez, J. Lerma, M. Hasebe, G. Shibasaki, G. Krychowiak' P. Zieliński, I. Gueye, A. Ndiaye, S. Ezatolahi, L. Schone, P. Aquino, E. Perez, M. Brozović, S. Ju, S. Rudy, P. Ndiaye, J. Góralski, A. Aguilar, W. Barrios, A. Ateef, H. Al-Moqahwi, A. Jahanbakhsh, Ad. Silva, S. Milinković-Savić, H. Jang, W. Jung, E. Dier, Y. Tielemans, M. Dembele, H. Yamagutchi, E. Banega, L. Torreira, D. Kuziaev, A. Christensen, R. Marquez, M. Fellaini.

Central attacking midfielders - A. Dzagoev, Abdalla Said, G. Guedes, Isco, T. Rogić, L. Messi, G. Sigurdsson, C. Cueva, C. Eriksen, A. Kramarić, J. Mikel, S. Milinković-Savić, M. Ozil, C. Vela, P. Coutinho, B. Dzemaili, J. Lingaard, D. Alli, J. Quintero, S. Kagawa, A. Milik, Y. Belhanda, M. Taremi, A. Griezmann, L. Modrić, J. Lee, M. Reus, R. Loftus-Cheek, A. Ljajić, F. Delph, P. Zieliński.

Left wingers - A. Golovin, Y. Al-Shehri, Trezeguet, G. de Arrascaeta, A. Harit, Y. Belhanda, M. Shojaei, V. Amiri, B. Fernandes, A. Iniesta, A. Griezmann, R. Kruse, A. di Maria, B. Bjarnason, E. Flores, P. Sisto, I. Perisić, A. Iwobi, B. Ruiz, A. Ljajić, J. Draxler, H. Lozano, Neymar, S. Zuber, E. Forsberg, H. Min, Y. Carrasco, E. Hazard, J. L. Rodriguez, N. Sliti, A. Young, J. Izquierdo, T. Inui, K. Grosicki, I. Sarr, D. Cheryshev, J. Mario, H. Zyiech, C. Rodriguez, B. Matuidi, M. Acuna, B. Idowu, F. Kostić, H. Hwang, S. Mane, M. Rybus, D. Kownacki, J. Rodriguez, B. Embolo, S. Moon, M. Reus, T. Hazard, M. Fellaini, T. Ukami, R. Kurzawa, M. Asensio, M. Braithwaite, N. Chadli.

Right wingers - A. Samedov, S. Al-Dawsari, A. Warda, N. Nandez, M. Boussoufa, N. Amrabat, K. Ansarifard, A. Jahanbakhsh, B. Silva, D. Silva, O. Dembele, M. Leckie, M. Meza, J. Gudmundsson, A. Carrillo, Y. Poulsen, A. Rebić, V. Moses, J. Venegas, D. Tadić, T. Muller, M. Layun, Willian, X. Shaqiri, V. Klaesson, H. Hwang, T. Meunier, D. Mertens, E. Barcenas, F. Ben Youssef, K. Trippier, J. Cuadrado, G. Haraguchi, J. Błaszczykowski, S. Mane, M. Salah, C. Sanchez, H. Bahbri, K. Mbappe, L. Messi, E. Salvio, S. Mun, I. Sarr, B. Bereszyński, P. Zieliński, R. Quaresma, J. Lee, L. Goretzka, T. Alexander-Arnold, N. Chadli, A. Januzaj, G. Sakai, K. Grosicki, C. Vela, K. De Bruyne.

Strikers - F. Smolov, M. Al-Sahlawi, M. Mohsen, L. Suarez, E. Cavani, A. El-Kaabi, S. Azmoun, C. Ronaldo, D. Costa, K. Mbappe, A. Nabbout, S. Aguero, A. Finnbogason, J. Farfan,

N. Jorgensen, M. Mandzukić, O. Ighalo, M. Urena, A. Mitrović, T. Werner, J. Hernandez, G. Jesus, H. Seferović, O. Toivonen, M. Berg, Sh. Kim, R. Lukaku, B. Perez, W. Khazri, H. Kane, R. Sterling, R. Falcao, Y. Osako, R. Lewandowski, M. Diouf, M. Niang, A. Dzyuba, G. Guedes, K. Boutaib, F. Al-Muwallad, O. Giroud, P. Guerreiro, A. Musa, H. Min, M. Berg, An. Silva, T. Jurić, M. Gavranović, J. Koo, M. Rashford, J. Vardy, M. Batshuayi, Y. Muto, A. Cornelius, J. Drmić, C. Stuani.

Abstract

The aim of this thesis was to find the best players of the 2018 FIFA World Cup using a data-driven approach, build the team of the tournament based on data analysis and compare how such a team differs from the subjective one constructed by a football specialist. The use of football data in data scouting influence a team's financial and sports success. Therefore, mathematical models of player evaluation such as Expected Goals model, Plus-Minus Models, Expected Threat and Valuing Actions by Estimating Probabilities were introduced. Basing on these models and using the WyScout data set consisting of events from 5 biggest European Leagues and the 2018 FIFA World Cup, I calculated in Python environment appropriate values and found the best players of the event. I built the team of the tournament using the VAEP per 90 minutes value as the results considered the highest number of actions. The team differed from the journalist choices, as the data-driven team found the players who made the most significant impact with their actions but lacked the evaluation of their movement. In contrast, the eye-test evaluated goalkeepers and defenders considering more appropriate actions for those positions than only scoring goals and was able to track player movement, but was biased towards teams that achieved the best place.