Random Walks across Social Networks: The Markovian Movement of a Stochastic Soccerball

Executive Summary

Association football is the world's most popular sport (Palacios-Huerta 2004) and a multibillion dollar industry (Deloitte 2019). Statistical analysis of the sport was pioneered more than fifty years ago (Reep and Benjamin 1968), but has only recently begun to make a more widespread impact (Cintia et al. 2015). Such analysis is of potential interest to coaches and scouts working professionally within the game, journalists and media commentators explaining the game, and gamblers and bookmakers putting money on the game. In this paper I build upon previous work applying Social Network Analysis (Grund 2012) and Markov Modelling (Peña 2014) to football data, and introduce an original metric for quantifying the contribution of each player to a team's goal-scoring capability.

Introduction

After thirty years of waiting, Liverpool F.C. are once again English champions (BBC 2020). But while manager Jurgen Klopp is now guaranteed his place in footballing folklore, less well known is the name of data scientist Ian Graham, whose analysis led to Klopp being hired in the first place (Schoenfeld 2019), and whose research team is regarded as "the strongest of its kind in English football" (Austin 2020).

In this paper, I develop a rigorous account of a football match as a stochastic social network, the graph of which is generated by the movement of the ball from player to player. I then show how the adjacency matrix of a player, considered as a node in a football match's possession sequence graph, can be used to quantify his contribution to the likelihood of the team scoring, which I call *Markov Expected Goals*. Finally, I discuss the potential and pitfalls of this new metric.

Data

JSON Event-logs

The dataset is provided by WyScout, and is available from FigShare.com (Pappalardo and Massucco 2019). It includes more than three million *event-logs* for every 2017-18 match from the five major European leagues, in JSON format (see Figure 1), with the details of players, teams, and informative tags (such as the crucial 'Goal') numerically encoded. Each

event is associated with an event type (such as 'Shot', 'Pass' or 'Free Kick'), identified with a player, and includes a nested array of the locational *positions* of the event itself and its target.

▼ Figure 1


```
{
    'eventId' : 8,
    'subEventName' : Simple pass,
    'tags' : [{'id': 1801}],
    'playerId' : 262,
    'positions' : [{'y': 40, 'x': 64}, {'y': 15, 'x': 63}],
    'matchId' : 2499736,
    'eventName' : Pass,
    'teamId' : 10531,
    'matchPeriod' : 2H,
    'eventSec' : 2590.033039,
    'subEventId' : 85,
    'id' : 179988688,
}
```


Figure 1: Example of Event-Log in JSON Format

Preprocessing

I used Python to convert this into a *tidy* dataframe (Wickham 2014), which I then analyzed and visualized using R. For the sake of reproducible research (Peng 2011), all code is attached as appendices.

As part of this preprocessing stage I also labelled each event according first to its *in-play sequence*, and then its *possession sequence*. An in-play sequence ends when the ball goes out of play, as indicated by the subsequent event having the *eventName* 'Free Kick' (a category which includes throw-ins and corners, as well as actual free-kicks given for fouls or offsides); or by the final two events (the shot and the attempted save) being labelled with the *tag* 'Goal'. (Although every goal is followed by a 'free kick' from the centre of the pitch, this never seems to be included in the event logs, presumably because they are created from camera footage which focusses on the goal celebrations and tends to miss the immediate restart of play after a goal).

Intuitively, to identify a possession sequence we should just be able to iterate through the event-logs of a given in-play sequence, and note when the team of the player changes. However, since the event-logs include challenges for the ball even when they are unsuccessful, this would lead to systematically undercounting the length of possession sequences. The solution I adopt is to iterate through the event-logs for a single team within an in-play sequence, paying close attention to the location data: if the location of an event is identical to the target-location of the previous event associated with that same team, then I treat is as part of the same possession sequence, otherwise I treat it as a separate sequence.

Initial Exploration

We begin our exploration of the data by tabulating the frequency of event types (Table 1) and pass types (Table 2), confirming that the *simple pass* is football's primary event.

▼ Table 1

Table 1: Frequency of Event Types

Description	Frequency (%)
Pass	1,565,355 (51.0%)
Challenge	832,055 (27.1%)
On the Ball	242,837 (7.91%)
Free Kick	182,468 (5.94%)
Interruption	130,096 (4.24%)
Foul	47,955 (1.56%)
Shot	40,462 (1.32%)
Save attempt	16,567 (0.539%)
Offside	7,821 (0.255%)
Goalkeeper leaving line	5,779 (0.188%)
Total	3,071,395 (100.%)

▼ Table 2

Table 2: Frequency of Pass Types

	Simple pass	High pass	Head pass	Cross	Launch	Smart pass	Hand pass	Total
Frequency (%)	1,207,447 (77.1%)	123,214 (7.87%)	91,194 (5.83%)	58,634 (3.75%)	43,303 (2.77%)	28,428 (1.82%)	13,135 (0.839%)	1,565,355 (100.%)

We can then visualize the location of the event-log data points in the context of a diagrammatic football pitch, to allow for an immediate and accessible view of the events of any match (see Figures 2 & 3).

▼ Figure 2

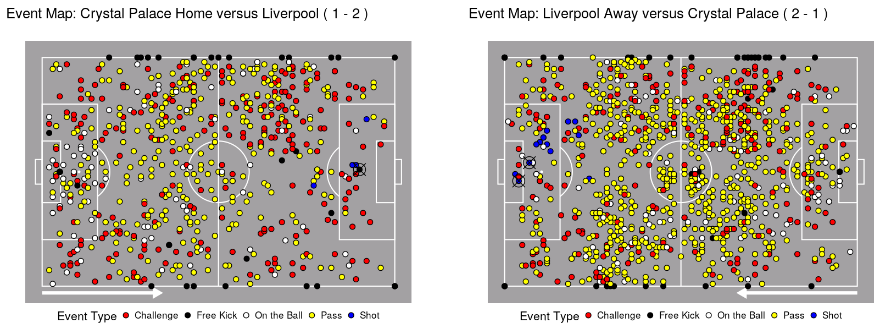


Figure 2: Events for Home and Away Team

▼ Figure 3

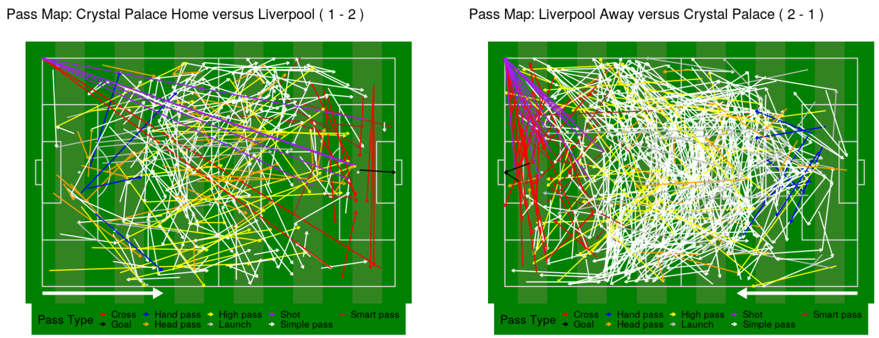


Figure 3: Passes and Shots for Home and Away Team

As well as the general overview of a full match's events, we can also consider a particular possession sequence (see Figure 4).

▼ Figure 4

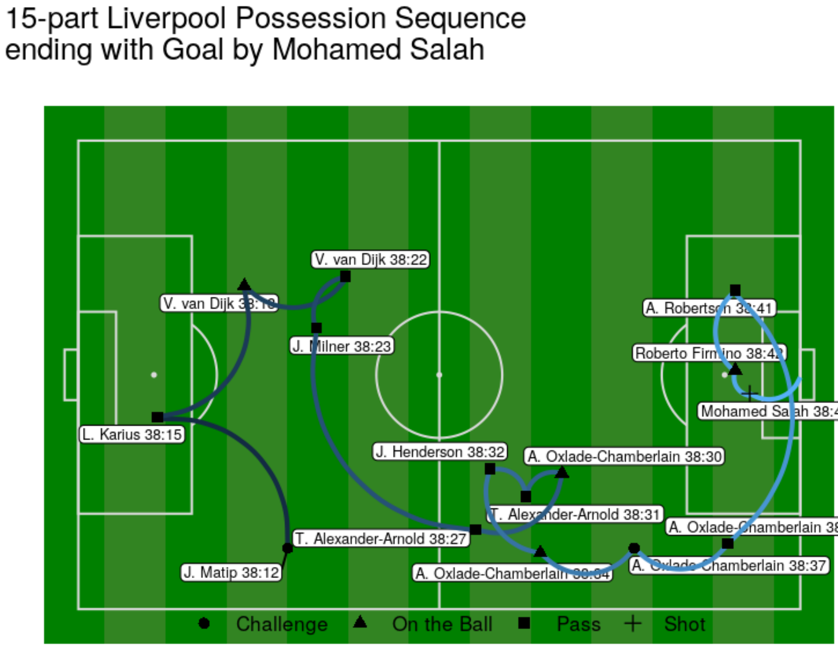


Figure 4: Pitchmap showing Possession Sequence

We can compare this to a tabulated account of all events from just before the beginning of the possession sequence until its end, and we see that our algorithm has successfully handled the case where an opposition player's challenge interrupts the sequence of events associated with the team in possession.

▼ Table 3

Table 3: Events of Possession Sequence

Half	Time	Team	Player	Event Description	Sequence ID
2H	38:07	Crystal Palace	W. Hennessey	Goal kick	2500032-91-CryPal-0
2H	38:12	Liverpool	J. Matip	Air duel	2500032-91-Liv-0
2H	38:12	Crystal Palace	C. Benteke	Air duel	2500032-91-CryPal-0
2H	38:15	Liverpool	L. Karius	Hand pass	2500032-91-Liv-0
2H	38:18	Liverpool	V. van Dijk	Touch	2500032-91-Liv-0
2H	38:22	Liverpool	V. van Dijk	Simple pass	2500032-91-Liv-0
2H	38:23	Liverpool	J. Milner	High pass	2500032-91-Liv-0
2H	38:27	Liverpool	T. Alexander-Arnold	Simple pass	2500032-91-Liv-0
2H	38:30	Liverpool	A. Oxlade-Chamberlain	Touch	2500032-91-Liv-0
2H	38:31	Liverpool	T. Alexander-Arnold	Simple pass	2500032-91-Liv-0
2H	38:32	Liverpool	J. Henderson	Simple pass	2500032-91-Liv-0
2H	38:34	Liverpool	A. Oxlade-Chamberlain	Acceleration	2500032-91-Liv-0
2H	38:37	Crystal Palace	W. Zaha	Ground defending duel	2500032-91-CryPal-1
2H	38:37	Liverpool	A. Oxlade-Chamberlain	Ground attacking duel	2500032-91-Liv-0
2H	38:39	Liverpool	A. Oxlade-Chamberlain	Cross	2500032-91-Liv-0
2H	38:41	Liverpool	A. Robertson	Cross	2500032-91-Liv-0
2H	38:42	Liverpool	Roberto Firmino	Touch	2500032-91-Liv-0
2H	38:43	Liverpool	Mohamed Salah	Goal	2500032-91-Liv-0

08/09/2021 Random Walks across Social Networks: The Markovian Movement of a Stochastic Soccerball

With possession sequences that end in a goal, it is generally possible to find video highlights of the events under consideration, and confirm that the data we have does correspond to what actually happened.

▼ Figure 5

Figure 5: YouTube Clip showing successful conclusion of possession sequence

---

Methodology

Graph Theoretic Foundations

Social Network Analysis (Wasserman and Faust 1994) applies the concepts of mathematical *graph theory*.

A *graph* is an ordered tuple $G = (V, E)$, consisting of a set of *nodes* (or *vertices*) $V = \{v_i\}$, and a set of *edges* $E = \{e_{ij}\}$, where the edge $e_{i,j}$ is the ordered pair (i, j) representing some connection from the *source* v_i to the *target* v_j.

Two nodes connected by an edge are said to be *adjacent* to each other. An edge e_{ii} connecting a node v_i to itself is called a *loop*; the node is then *self-adjacent*.

A graph is *undirected* if $e_{ij} \iff e_{ji}$ – otherwise it is *directed*. It is a *simple* graph if the edges are distinct and unrepeated – otherwise it is a *multigraph*.

On an undirected graph, we call the number of edges connecting a node its *degree*. In a directed graph we distinguish between the *indegree* and *outdegree*.

Given a graph G, we can describe a *walk* of length L as a sequence of adjacent (but not necessarily distinct) nodes $(v_0, \ldots, v_L)$; or, equivalently, as a sequence of edges $e_{0,1}, \ldots, e_{L-1,L}$. Conversely, given a set of walks, we can construct the (minimal) underlying graph containing all the nodes and edges involved in the walks.

A graph is *weighted* if there exists some function $w : E \mapsto \mathbb{R}$ assigning a weight value to each edge.

An unweighted multigraph can thus be viewed as a weighted simple graph by defining $w(e_{i,j}) = |\{e \in E : e = e_{i,j}\}| \in \mathbb{N}_0$, so that the weight of an edge is the number of times it is repeated.

A weighted simple graph can be uniquely described by an *adjacency matrix*:

$$M_{ij} = \begin{cases} w(e_{i,j}), & e_{i,j} \in E \\ 0, & \text{otherwise.} \end{cases}$$

Graphing a Team's Possession Network

We consider the possession sequences of a football match as *walks* of the football across a social graph in which the nodes are players (considered either individually or collectively), and the connecting edges are defined by the movement of the ball, as it travels from player to player. We can then construct the underlying graph for any particular possession sequence (as in Figure 6), or indeed for the whole game.

▼ Figure 6

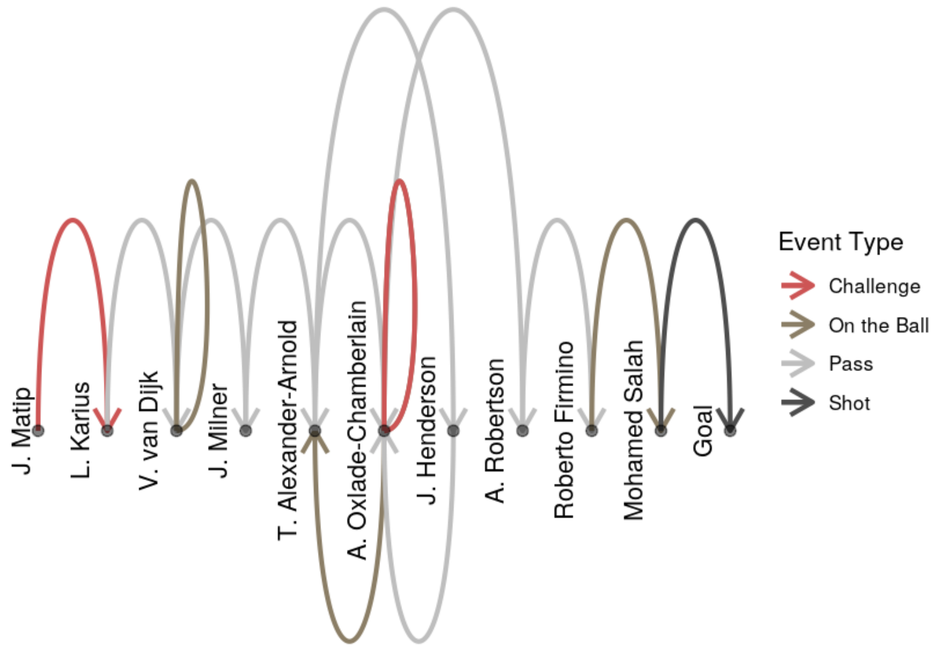
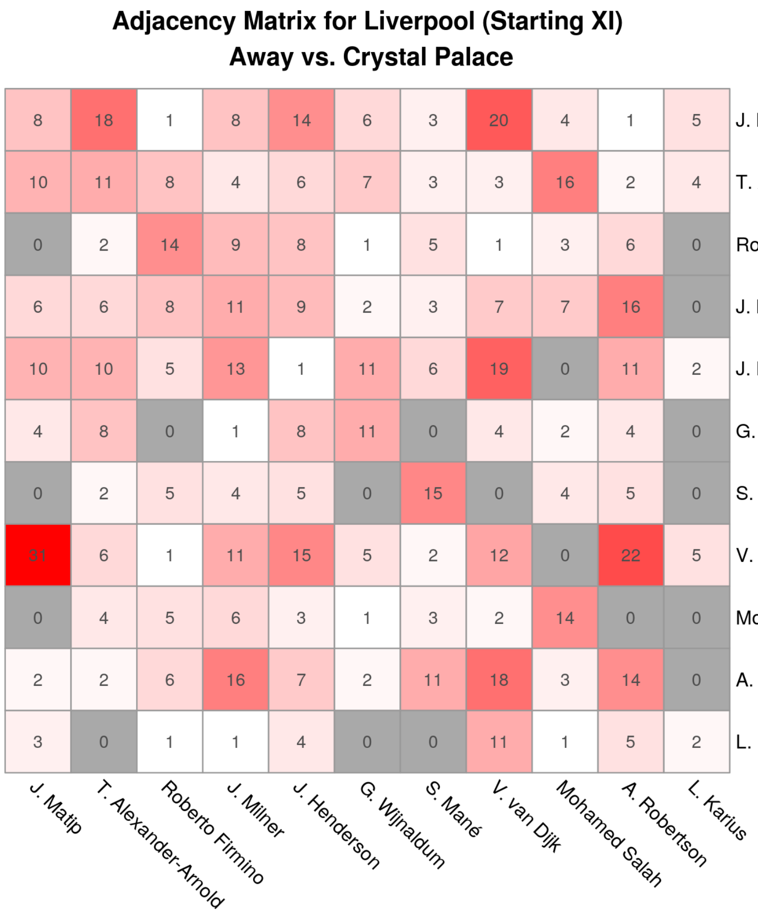


Figure 6: Graph of Possession Sequence leading to Goal

There are several different 'weights' we could assign an edge: the Euclidean distance moved by the ball; the polar angle; the time duration. However, for this particular analysis we focus simply on the network connections between players.

We do not restrict our analysis to actual passes but include touches on the ball, challenges, and shots. Regardless of the type of event, we consider any movement of the ball from player U to player V as an edge $e_{U,V}$, including the loop $e_{U,U}$ where the player dribbles with the ball. As suggested above, we let the weight of the edge $e_{U,V}$ be the number of times the ball moves from player U to player V. We can then generate the adjacency matrix for a team in a particular match.

▼ Figure 7



	J. Matip	T. Alexander-Arnold	Roberto Firmino	J. Milner	J. Henderson	G. Wijnaldum	S. Mané	V. van Dijk	Mohamed Salah	A. Robertson	L. Karius
J. Matip	8	18	1	8	14	6	3	20	4	1	5
T. Alexander-Arnold	10	11	8	4	6	7	3	3	16	2	4
Roberto Firmino	0	2	14	9	8	1	5	1	3	6	0
J. Milner	6	6	8	11	9	2	3	7	7	16	0
J. Henderson	10	10	5	13	1	11	6	19	0	11	2
G. Wijnaldum	4	8	0	1	8	11	0	4	2	4	0
S. Mané	0	2	5	4	5	0	15	0	4	5	0
V. van Dijk	31	6	1	11	15	5	2	12	0	22	5
Mohamed Salah	0	4	5	6	3	1	3	2	14	0	0
A. Robertson	2	2	6	16	7	2	11	18	3	14	0
L. Karius	3	0	1	1	4	0	0	11	1	5	2

Figure 7: Heatmap of Adjacency Matrix

It is increasingly common (Knutson 2018) to visualize a possession network by positioning each node according to the mean location of the player's touches, and varying the appearance (width, colour or transparency) of the edges to show the frequency of that pass (Figure 8). This swiftly communicates a team's formation, and gives a qualitative sense of a team's strategy.

▼ Figure 8

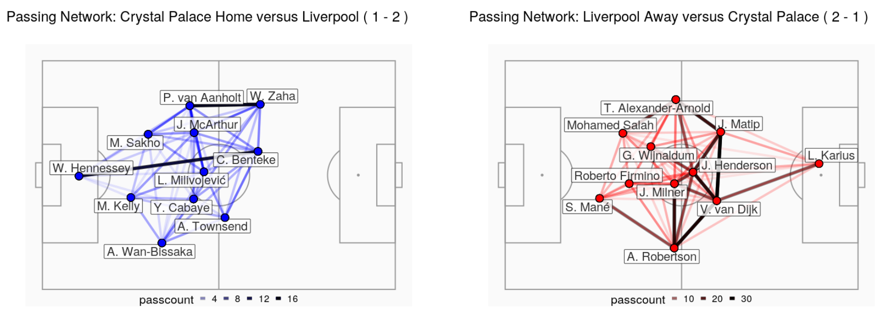


Figure 8: Possession Networks for Home and Away Team

Probabilistic Walks on a Graph

To consider the team's ball-possession in its dynamic development, rather than merely as a static structure, we also need some definitions from *probability theory* (Serfozo 2009).

A *sample space* $\Omega = \{\omega_i\}$ is the set of possible *outcomes* of an *observation*. If we are observing who has the ball, then the sample space is the set of players on the pitch. An *event* F is some subset of Ω, and the *event space* $\mathcal{F}$ is the set of subsets of Ω. In particular, $\Omega \subset \mathcal{F}$.

To illustrate the difference between the sample space and the event space, we could observe whether the *home team* is in possession of the ball. The outcome of observing who has the ball will not be a whole team, but merely some particular player ω_x, whom we could call h_x if he is on the home team, or a_x if he is on the away team. So our sample space $\Omega = \{\omega_i\} = \{h_i\} \cup \{a_i\}$, and the *event* that the home team has possession $H = \{h_i\} \in \mathcal{F}$.

A *probability* $\mathbb{P}$ is then a function $\mathbb{P} : \mathcal{F} \mapsto [0, 1] \in \mathbb{R}$, such that $\sum_{\omega_i \in \Omega} \mathbb{P}(\omega_i) = 1$ and $\mathbb{P}(\emptyset) = 0$. We can also talk about the *conditional probability* of an event X given another event Y, which we write $\mathbb{P}(X|Y)$.

A *random variable* is a function $X : \Omega \mapsto S$. If $S \subseteq \mathbb{R}$ then we also have the *expectation* $\mathbb{E}[X]$ and *variance* $\mathrm{Var}[X]$. Again, to illustrate what we mean – the number of goals in the match is an integer-valued random variable, and since $\mathbb{Z} \subset \mathbb{R}$ we could talk about the expectation and variation of the number of goals; but this is not the case for the question of which player has the ball at a particular time, though this also is a random variable.

A *stochastic process* is an indexed set (that is, a *sequence*) of random variables, where the index commonly refers to points in time $t \in T$, which may be considered discretely (so $t \in \mathbb{N}_0$) or continuously (so $t \in \mathbb{R}_{\geq 0}$). The range S of a stochastic process $\{X(t, \omega) : t \in T, \omega \in \Omega\}$ is called its *state space*, and the value X_t is its *state* at time t.

A stochastic process has the *Markov property* if $\forall t \geq 0, \mathbb{P}(X_{t+1} = s|X_0, ..., X_t) = \mathbb{P}(X_{t+1} = s|X_t)$; this means that future states are dependent only on the present state, regardless of the previous history of past states. Such a process can be called a Markov process (Norris 1998), and we call the probability $\mathbb{P}(X_{t+1} = s_j|X_t = s_i)$ the *transition probability* p_{ij} .

A Markov process is *time-homogenous* if the transition probabilities stay constant through time, ie. $\mathbb{P}(X_{n+1} = s|X_n) = \mathbb{P}(X_{m+1} = x|X_m) \forall m, n \in T$. If S is countable, then the probability distribution of such a process can be described by its initial *distribution vector* $\boldsymbol{\alpha_0}$ and its *transition matrix* $\mathbf{P}$, where each ith row is made up of the transition probabilities p_{ij} and $\sum_j p_{ij} = 1 \forall i$. Specifically, after k transitions we have the distribution $\boldsymbol{\alpha_k} = \boldsymbol{\alpha_0} \mathbf{P}^k$.

If a Markov process develops in discrete time and takes a values from a discrete state space, then we call it a *Markov chain* and can straightforwardly draw it as a weighted simple graph by treating its transition matrix as an adjacency matrix. Conversely, we can transform an adjacency matrix $\mathbf{M}$ into a transition matrix $\mathbf{P}$ by defining the elements so that each row sums to 1 as required:

$$p_{ij} := \frac{m_{ij}}{\sum_j m_{ij}}$$

We can then picture the development of a Markov process as a *random walk* on the graph in which the nodes V of the graph are the state space S of the process, and the weights of the edges are the transition probabilities.

▼ Figure 9

08/09/2021 Random Walks across Social Networks: The Markovian Movement of a Stochastic Soccerball

Transition Matrix for Liverpool Away vs. Crystal Palace

J. Matip	T. Alexander-Arnold	Roberto Firmino	J. Milner	J. Henderson	G. Wijnaldum	S. Mané	V. van Dijk	Mohamed Salah	A. Robertson	L. Karius	A. Oxlade-Chamberlain	A. Lallana	D. Lovren	Opposition	Goal	
0.07	0.17	0.01	0.07	0.13	0.06	0.03	0.19	0.04	0.01	0.05	0.02	0.00	0.02	0.14	0.00	J. Matip
0.08	0.09	0.07	0.03	0.05	0.06	0.03	0.03	0.13	0.02	0.03	0.03	0.01	0.00	0.34	0.00	T. Alexander-Arnold
0.00	0.03	0.18	0.12	0.10	0.01	0.06	0.01	0.04	0.08	0.00	0.04	0.00	0.01	0.31	0.00	Roberto Firmino
0.06	0.06	0.08	0.10	0.08	0.02	0.03	0.07	0.07	0.15	0.00	0.02	0.00	0.01	0.26	0.00	J. Milner
0.09	0.09	0.05	0.12	0.01	0.10	0.06	0.17	0.00	0.10	0.02	0.02	0.00	0.01	0.17	0.00	J. Henderson
0.07	0.15	0.00	0.02	0.15	0.20	0.00	0.07	0.04	0.07	0.00	0.00	0.00	0.00	0.24	0.00	G. Wijnaldum
0.00	0.04	0.09	0.07	0.09	0.00	0.26	0.00	0.07	0.09	0.00	0.00	0.00	0.00	0.28	0.02	S. Mané
0.24	0.05	0.01	0.09	0.12	0.04	0.02	0.09	0.00	0.17	0.04	0.00	0.00	0.04	0.10	0.00	V. van Dijk
0.00	0.06	0.08	0.10	0.05	0.02	0.05	0.03	0.22	0.00	0.00	0.05	0.00	0.00	0.33	0.02	Mohamed Salah
0.02	0.02	0.05	0.14	0.06	0.02	0.09	0.15	0.03	0.12	0.00	0.04	0.00	0.00	0.26	0.00	A. Robertson
0.08	0.00	0.03	0.03	0.10	0.00	0.00	0.28	0.03	0.13	0.05	0.00	0.00	0.03	0.26	0.00	L. Karius
0.03	0.10	0.10	0.14	0.07	0.00	0.00	0.00	0.03	0.07	0.00	0.10	0.00	0.03	0.31	0.00	A. Oxlade-Chamberlain
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.50	0.00	0.00	0.00	0.00	0.50	0.00	A. Lallana
0.09	0.04	0.04	0.00	0.09	0.00	0.00	0.30	0.00	0.04	0.04	0.04	0.00	0.04	0.26	0.00	D. Lovren
0.07	0.09	0.03	0.04	0.05	0.01	0.01	0.00	0.02	0.05	0.03	0.01	0.00	0.01	0.59	0.00	Opposition
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	Goal

Figure 9: Heatmap of Transition Matrix

Calculating 'Markov Expected Goals'

In Figure 9 we show the heatmap of the Transition Matrix derived from the Adjacency Matrix shown in Figure 7. There we showed only the adjacency counts for the starting eleven, but now we include the subsitutes, as well as treating the 'Opposition' collectively as a single node. Most significantly, we now include a 'Goal' as a node. In particular, in the terms of Markov theory, we treat it as an *absorbing state* which loops back on itself with certain probability. This allows us to calculate the expected *hitting time* of a goal, given that any particular player has the ball.

The hitting time τ_s of some state $s \in S$ is given by $\tau_s := min\{t \geq 0 \colon X_t = s\}$, the earliest time at which the process reaches that state.

Given an initial distribution $\boldsymbol{\alpha_0}$, we know that after k transitions, the state $\boldsymbol{\alpha_k} = \boldsymbol{\alpha_0} \mathbf{P}^k$.

We can define the *state vector* $\boldsymbol{\delta_i}$ of any particular state s_i as

$$\delta_{ij} := \begin{cases} 1, & i = j \\ 0, & \text{otherwise.} \end{cases}$$

The probability $\mathbb{P}(X_k = s_i)$ is then given by the dot product $\langle \boldsymbol{\alpha_k}, \boldsymbol{\delta_i} \rangle$.

If s_i is an absorbing state, then the probability that it is reached *for the first time* is given by

$$\pi_k := \langle \boldsymbol{\alpha_k}, \boldsymbol{\delta_i} \rangle - \langle \boldsymbol{\alpha_{k-1}}, \boldsymbol{\delta_i} \rangle.$$

If the expectation of the hitting time is finite it can then be approximated directly by iterating up to some large K:

$$\mathbb{E}[\tau_s] = \sum_{k=0}^{\infty} k\pi_k \approx \sum_{k=0}^{K} k\pi_k$$

Thus we can quantify, for each player on the team, the expected number of events there will be before the team scores a goal. We invert this to give a measure of the expected number of goals; and standardize it by the team's average number of events per game. We then scale this by the team's mean values, so that we can make a fair comparison across teams to see which individual players contribute the most to their teams' goalscoring chances. We call this measure *Markov Expected Goals* (MXG).

Results

Applying our method to the cumulative season's transition matrix for every team in the dataset, we find that Inter's Icardi is the player who contributed most to the chance of his team scoring (considering only players with degree of at least 1000). His MXG score is almost three times higher than that of Messi, who only just makes it into the Top Twenty (Table 4).

It is instructive to consider Messi's score in order to understand what a high MXG score implies. Of the twenty, Messi has the highest number of goals, the highest number of loops, and the highest (out-)degree, but the very strength of his degree score means that his percentage of goals per degree (which correlates most strongly with MXG) is the lowest.

▼ Table 4

Table 4: Top Twenty Players in European Leagues by MXG Score (2017-18)

Player	Team	MXG	Degree	Loops (% Degree)	Goals (% Degree)
M. Icardi	Internazionale	34.1	1,005	152 (15.1%)	29 (2.89%)
E. Cavani	PSG	30.5	1,232	293 (23.8%)	28 (2.27%)
R. Lewandowski	Bayern München	23.5	1,330	256 (19.2%)	29 (2.18%)
S. Agüero	Manchester City	21.9	1,276	261 (20.5%)	21 (1.65%)
H. Kane	Tottenham Hotspur	20.9	1,804	396 (22.0%)	29 (1.61%)
C. Immobile	Lazio	20.7	1,526	255 (16.7%)	29 (1.90%)
Cristiano Ronaldo	Real Madrid	20.2	1,489	266 (17.9%)	26 (1.75%)
Mohamed Salah	Liverpool	19.3	2,089	425 (20.3%)	32 (1.53%)
Mariano Díaz	Olympique Lyonnais	17.3	1,157	231 (20.0%)	18 (1.56%)
J. Vardy	Leicester City	16.4	1,249	194 (15.5%)	20 (1.60%)
C. Bacca	Villarreal	14.6	1,095	178 (16.3%)	15 (1.37%)
F. Quagliarella	Sampdoria	14.4	1,419	235 (16.6%)	19 (1.34%)
C. Stuani	Girona	14.1	1,416	248 (17.5%)	21 (1.48%)
L. Suárez	Barcelona	14.1	1,929	274 (14.2%)	25 (1.30%)
R. Falcao	Monaco	13.3	1,231	230 (18.7%)	18 (1.46%)
Maxi Gómez	Celta de Vigo	13.1	1,528	260 (17.0%)	18 (1.18%)
Gabriel Jesus	Manchester City	12.9	1,137	179 (15.7%)	13 (1.14%)
M. Balotelli	Nice	12.9	1,514	252 (16.6%)	18 (1.19%)
G. Bale	Real Madrid	12.2	1,361	232 (17.0%)	16 (1.18%)
L. Messi	Barcelona	12.0	3,225	671 (20.8%)	34 (1.05%)

Conclusion

We have here focussed on the attacking contribution a player makes towards his team's goal-scoring capability, but we could equally consider the defensive aspect by letting the absorbing state be the possibility that the opposition score.

One apparent flaw of this analysis is that it pays no attention to the *spatial* dynamics of the game. But it would be fairly straightforward to develop the analysis articulated here to also take into account a player's location on the pitch when he plays the ball. Rather than merely representing the players, the nodes' state space could instead be the Cartesian product of players with some set of discrete areas of the pitch, say $\{\Delta, \Phi, \Xi\}$ representing the Defensive Half, Attacking Half, and Attacking Penalty Area respectively. One could simply add the spatial indicator to the player name at the preprocessing stage, and the rest of the analysis would then proceed as described here.

But the major weakness is one inherent in the data itself – although the event-log dataset offers a wealth of information when compared to the manual *notational analysis* first developed by Charles Reep in the 1950s (Pollard 2002), a convincing analysis of football will only really be possible with data that includes the positions of off-the-ball players, as well as on-the-ball events (Tavares 2017). Such data can be captured (eg. Pettersen et al. 2014), but is not yet widely (at least openly) available.

Appendix 1: Social Network Analysis in R

▼ Code

```
library(plyr) # data manipulation
library(dplyr) # data manipulation
library(igraph) # for graph/network analysis
library(ggraph) # graph visualization in the style of ggplot
library(netrankr) # network centrality
library(ggplot2) # for plotting visualizations
library(ggforce) # adds functionality to ggplot2, eg. geom_circle
library(ggrepel) # adds 'repellent' non-overlapping labels to ggplot
library(pheatmap) # prettier heat maps
library(RColorBrewer) # color palettes
library(docstring) # allows ?help to display function descriptions analogous to Pythonic docstrings
```

```
LoadAndScaleData <- function(national_league) {
   #' Load preprocessed event logs for given national league.

   events <- read.csv(paste0('../data/processed/', national_league, '.csv'))

   x_scale <- 105
   y_scale <- 68

   events$location_x <- events$location_x/100 * x_scale
   events$target_x <- events$target_x/100 * x_scale

   events$location_y <- y_scale - (events$location_y/100 * y_scale)
   events$target_y <- y_scale - (events$target_y/100 * y_scale)

   return(events)
}
```



```
LeagueResults <- function(league){
   #' Test data integrity by calculating results and table for given national league.

   teams <- sort(unique(league$team))


   league_table <- matrix(0, nrow=length(teams), ncol=9)
   rownames(league_table) <- teams
   colnames(league_table) <- c('Pos','Pld','W','D','L','GF','GA','GD','Pts')

   for (m in unique(league$matchId)){

       match <- league[league$matchId==m,]
       home_team <- unique(match[match$home_or_away=='Home',]$team)
       away_team <- unique(match[match$home_or_away=='Away',]$team)
       home_goals <- nrow(match[match$to_team=='Home Goal',])
       away_goals <- nrow(match[match$to_team=='Away Goal',])


       if (home_goals > away_goals){
           league_table[home_team,'W'] <- league_table[home_team,'W'] + 1
           league_table[away_team,'L'] <- league_table[away_team,'L'] + 1
       } else if (away_goals > home_goals){
           league_table[home_team,'L'] <- league_table[home_team,'L'] + 1
           league_table[away_team,'W'] <- league_table[away_team,'W'] + 1
       } else if (home_goals == away_goals) {
           league_table[home_team,'D'] <- league_table[home_team,'D'] + 1
           league_table[away_team,'D'] <- league_table[away_team,'D'] + 1
       }

       league_table[home_team,'GF'] <- league_table[home_team,'GF'] + home_goals
       league_table[away_team,'GF'] <- league_table[away_team,'GF'] + away_goals
       league_table[home_team,'GA'] <- league_table[home_team,'GA'] + away_goals
       league_table[away_team,'GA'] <- league_table[away_team,'GA'] + home_goals

       for (team in c(home_team, away_team)){
           league_table[team,'GD'] <- league_table[team,'GF'] - league_table[team,'GA']
           league_table[team,'Pts'] <- 3*league_table[team,'W'] + 1*league_table[team,'D']
           league_table[team,'Pld'] <- league_table[team,'Pld'] + 1
       }

   }

   league_table <- league_table[order(-league_table[,'Pts']),]

   league_table[,'Pos'] <- 1:20

   return(league_table)
}
```

```
DrawPitch <- function(lengthPitch=105, widthPitch=68, arrow=c("none", "r", "l"), theme=c("light", "dark", "grey", "grass")) {
    #' Draw regulation football pitch with penalty areas and centre circle.
    #' Adapted from https://github.com/JoGall/soccermatics

  # define colours by theme
  if(theme[1] == "grass") {
    fill1 <- "#008000"
    fill2 <- "#328422"
    colPitch <- "grey85"
    arrowCol <- "white"
    colText <- "white"
  } else if(theme[1] == "light") {
    fill1 <- "grey98"
    fill2 <- "grey98"
    colPitch <- "grey60"
    arrowCol = "black"
    colText <- "black"
  } else if(theme[1] %in% c("grey", "gray")) {
    fill1 <- "#A3A1A3"
    fill2 <- "#A3A1A3"
    colPitch <- "white"
    arrowCol <- "white"
    colText <- "black"
  } else if(theme[1] == "dark") {
    fill1 <- "#1C1F26"
    fill2 <- "#1C1F26"
    colPitch <- "white"
    arrowCol <- "white"
    colText <- "white"
  } else if(theme[1] == "blank") {
    fill1 <- "white"
    fill2 <- "white"
    colPitch <- "white"
    arrowCol <- "black"
    colText <- "black"
  }
  lwd <- 0.5

  border <- c(5, 5, 5, 5)

  # mowed grass lines
  lines <- (lengthPitch + border[2] + border[4]) / 13
  boxes <- data.frame(start = lines * 0:12 - border[4], end = lines * 1:13 - border[2])[seq(2, 12, 2),]

  # draw pitch
  p <- ggplot() +
    # background
    geom_rect(aes(xmin = -border[4], xmax = lengthPitch + border[2], ymin = -border[3], ymax = widthPitch + border[1]), fill =
fill1) +
    # mowed pitch lines
    geom_rect(data = boxes, aes(xmin = start, xmax = end, ymin = -border[3], ymax = widthPitch + border[1]), fill = fill2) +
    # perimeter line
    geom_rect(aes(xmin = 0, xmax = lengthPitch, ymin = 0, ymax = widthPitch), fill = NA, col = colPitch, lwd = lwd) +
    # centre circle
    geom_circle(aes(x0 = lengthPitch/2, y0 = widthPitch/2, r = 9.15), col = colPitch, lwd = lwd) +
    # kick off spot
```



```
    geom_circle(aes(x0 = lengthPitch/2, y0 = widthPitch/2, r = 0.25), fill = colPitch, col = colPitch, lwd = lwd) +
    # halfway line
    geom_segment(aes(x = lengthPitch/2, y = 0, xend = lengthPitch/2, yend = widthPitch), col = colPitch, lwd = lwd) +
    # penalty arcs
    geom_arc(aes(x0= 11, y0 = widthPitch/2, r = 9.15, start = pi/2 + 0.9259284, end = pi/2 - 0.9259284), col = colPitch, lwd =
lwd) +
    geom_arc(aes(x0 = lengthPitch - 11, y0 = widthPitch/2, r = 9.15, start = pi/2*3 - 0.9259284, end = pi/2*3 + 0.9259284),
             col = colPitch, lwd = lwd) +
    # penalty areas
    geom_rect(aes(xmin = 0, xmax = 16.5, ymin = widthPitch/2 - 20.15, ymax = widthPitch/2 + 20.15), fill = NA, col = colPitch,
lwd = lwd) +
    geom_rect(aes(xmin = lengthPitch - 16.5, xmax = lengthPitch, ymin = widthPitch/2 - 20.15, ymax = widthPitch/2 + 20.15),
              fill = NA, col = colPitch, lwd = lwd) +
    # penalty spots
    geom_circle(aes(x0 = 11, y0 = widthPitch/2, r = 0.25), fill = colPitch, col = colPitch, lwd = lwd) +
    geom_circle(aes(x0 = lengthPitch - 11, y0 = widthPitch/2, r = 0.25), fill = colPitch, col = colPitch, lwd = lwd) +
    # six yard boxes
    geom_rect(aes(xmin = 0, xmax = 5.5, ymin = (widthPitch/2) - 9.16, ymax = (widthPitch/2) + 9.16), fill = NA, col = colPitch,
lwd = lwd) +
    geom_rect(aes(xmin = lengthPitch - 5.5, xmax = lengthPitch, ymin = (widthPitch/2) - 9.16, ymax = (widthPitch/2) + 9.16),
              fill = NA, col = colPitch, lwd = lwd) +
    # goals
    geom_rect(aes(xmin = -2, xmax = 0, ymin = (widthPitch/2) - 3.66, ymax = (widthPitch/2) + 3.66), fill = NA, col = colPitch,
lwd = lwd) +
    geom_rect(aes(xmin = lengthPitch, xmax = lengthPitch + 2, ymin = (widthPitch/2) - 3.66, ymax = (widthPitch/2) + 3.66),
              fill = NA, col = colPitch, lwd = lwd) +
    coord_fixed() +
    theme(rect = element_blank(),
          line = element_blank(),
          axis.text = element_blank(),
          axis.title = element_blank())

  # add arrow
  if(arrow[1] == "r") {
    p <- p +
      geom_segment(aes(x = 0, y = -2, xend = lengthPitch / 3, yend = -2),
                   colour = arrowCol, size = 1.5, arrow = arrow(length = unit(0.2, "cm"), type="closed"), linejoin='mitre')
  } else if(arrow[1] == "l") {
    p <- p +
      geom_segment(aes(x = lengthPitch, y = -2, xend = lengthPitch / 3 * 2, yend = -2),
                   colour = arrowCol, size = 1.5, arrow = arrow(length = unit(0.2, "cm"), type="closed"), linejoin='mitre')
  }

  return(p)
}
```

```
ShowMatchEvents <- function(events, match_id, team_name, home_or_away='Home', flip=F) {
    #' Visualize events on pitch for given match and team.

    # select match
    game <- events[events$matchId==match_id,]
    goals <- game[game$subEventName=='Goal',]

    # note teams
    teams <- unique(game$team)

    # limit events to team specified by name or home/away
    if(!missing(team_name)){
        game <- game[game$team==team_name,]
        home_or_away <- unique(game$home_or_away)
    } else {
        game <- game[game$home_or_away==home_or_away,]
        team_name <- unique(game$team)
    }

    # note opposition
    opposition <- teams[teams!=team_name]

    # get score
    team_score <- table(goals$team)[team_name]
    opposition_score <- table(goals$team)[opposition]

    # flip coordinates if desired
    if (flip==T) {
        game$location_x <- 105 - game$location_x
        game$location_y <- 68 - game$location_y
        direction_of_play = 'l'
    } else {
        direction_of_play = 'r'
    }

    # limit attention to main events
    game_events <- game[game$eventName == 'Pass'|
                    game$eventName == 'Shot' |
                    game$eventName == 'On the Ball' |
                    game$eventName == 'Challenge' |
                    game$eventName == 'Free Kick',]

    # draw pitch
    p <- DrawPitch(theme='grey', arrow=direction_of_play) +
        geom_point(data = game_events,
               aes(location_x , location_y, fill=eventName, shape=eventName), pch=21, alpha=1, size=2 ) +
        geom_point(data=game_events[game_events$subEventName=='Goal',],
                   aes(location_x, location_y), shape=13, size=5) +
#         geom_label_repel(data=game_events[game_events$subEventName=='Goal',],
#                          aes(location_x, location_y, label = paste0(source,'(',matchPeriod,' ',time,')')), label.padding=0.1,
size=2.3, alpha=1) +
        theme(legend.direction='horizontal', legend.position=c(0.5,0)) +
        scale_fill_manual(values=c("red", "black", 'white', "yellow", 'blue'), name='Event Type') +
        ggtitle(paste('Event Map:', team_name,home_or_away,'versus',opposition,'(',team_score,'-',opposition_score,')'))

    p$figname <- paste0('EventMap',home_or_away)
```



```
    return(p)

}
```

```
ShowPassesAndShots <- function(events, match_id, team_name, home_or_away='Home', flip=F) {
   #' Visualize passes on pitch for given match and team.

   # select match
   game <- events[events$matchId==match_id,]
   goals <- game[game$subEventName=='Goal',]

   # note teams
   teams <- unique(game$team)

   # limit events to team specified by name or home/away
   if(!missing(team_name)){
       game <- game[game$team==team_name,]
       home_or_away <- unique(game$home_or_away)
   } else {
       game <- game[game$home_or_away==home_or_away,]
       team_name <- unique(game$team)
   }

   # note opposition
   opposition <- teams[teams!=team_name]

   # get score
   team_score <- table(goals$team)[team_name]
   opposition_score <- table(goals$team)[opposition]

   # flip coordinates if desired
   if (flip==T) {
       game$location_x <- 105 - game$location_x
       game$target_x <- 105 - game$target_x
       game$location_y <- 68 - game$location_y
       game$target_y <- 68 - game$target_y
       direction_of_play = 'l'
   } else {
       direction_of_play = 'r'
   }

   # limit attention to passes and shots
   passes <- game[game$eventName == 'Pass' & game$team == team_name,]
   shots <- game[(game$eventName=='Shot' | game$subEventName=='Goal') & game$team==team_name,]

   # draw pitch
   p <- DrawPitch(theme='grass', arrow=direction_of_play) +
       geom_segment(data=na.exclude(passes),
                    aes(x=location_x, y=location_y, xend=target_x, yend=target_y, color=subEventName),
                    alpha=1, arrow = arrow(length = unit(0.1,"cm"))) +
#        geom_label_repel(data=game[game$subEventName=='Goal',],
#                     aes(location_x, location_y, label = paste0(source,'(',matchPeriod,' ',time,')')), label.padding=0.1,
size=2.3, alpha=1) +
       geom_segment(data=na.exclude(shots),
                aes(x=location_x, y=location_y, xend=target_x, yend=target_y, color=subEventName),
                alpha=1, arrow = arrow(length = unit(0.1,"cm"))) +
       theme(legend.position=c(0.5,-0.01), legend.direction='horizontal',
             legend.background=element_rect(fill='#008000', linetype='solid')) +
       scale_color_manual(values=c("red", "black", 'blue', 'orange', "yellow", 'grey', 'purple', 'white','brown'), name='Pass
Type') +
```



```
       ggtitle(paste('Pass Map:',team_name,home_or_away,'versus',opposition,'(',team_score,'-',opposition_score,')'))

   p$figname <- paste0('PassMap',home_or_away)

   return(p)
}
```

```
GoalSeqs <- function(events, match_id) {
   match <- events[events$matchId==match_id,]
   goals <- match[match$subEventName=='Goal',]
   goal_seqs <- unique(goals$possession)
   return(goal_seqs)
}
```

```
TabulateSequence <- function(events, possession_sequence) {
   # Return a markdown table starting from the row before a possession sequence begins, and continuing until it ends.

   start <- min(events[events$possession == possession_sequence,]$X)
   stop <- max(events[events$possession == possession_sequence,]$X) + 1
   table <- events[start:stop,]
   df <- data.frame(table$matchPeriod, table$time, table$team, table$source, table$subEventName)
   names(df) <- c('Half', 'Time', 'Team', 'Event Description', 'Player')

   return(df)
}
```

```
SequenceOnPitch <- function(events, possession_sequence){
   #' Draw Possession Sequence on Pitch

   data <- events[events$possession == possession_sequence,]

   p <- DrawPitch(theme='grass') +
       geom_label_repel(data = data,
                        aes(location_x, location_y, label = paste(source,time)),
                        label.padding=0.1, size=2.3, alpha=1) +
       geom_curve(data = data,
                  aes(x = location_x, xend = target_x,
                      y = location_y, yend = target_y*.99, col = X),
                  show.legend=FALSE, size=1, alpha = 1) +
       geom_point(data = data,
                  aes(location_x , location_y, shape=eventName), size=2) +
       ggtitle(paste0(nrow(data),'-part ', unique(data$team)[1],
                      ' Possession Sequence \nending with ',
                      data[nrow(data),]$subEventName, ' by ',
                      data[nrow(data),]$source))  +
       theme(legend.position=c(0.5,0.08),
             legend.direction='horizontal',
             legend.title=element_blank())

   return(p)
}
```

```
SequenceGraph <- function(events, possession_sequence){
   #' Return Graph of Possession Sequence

   data <- events[events$possession == possession_sequence,]
   data <- data[data$source!='' & data$target!='nan',]
   nodes <- unique(c(as.character(data$source),as.character(data$target)))
   edges <- data.frame(data$source, data$target)
   g <- graph_from_data_frame(d=edges, vertices=nodes, directed=TRUE)
   g$id <- possession_sequence
   g$data <- data
   g$team <- as.character(unique(data$team))

   return(g)
}
```

```
VisualizeGraph <- function(possession_sequence_graph) {
   #' Visualize Possession Sequence as Linear Graph with Looping Edges

   g <- possession_sequence_graph
   data <- possession_sequence_graph$data

   visualization <- ggraph(g, 'linear') +
       geom_edge_arc(aes(color=data$eventName),
                     arrow=arrow(length=unit(4,'mm')),
                     fold=F,
                    width=1) +
       geom_edge_loop(aes(color=data$eventName),
                     width=1) +
       geom_node_point(color='black',
                       size=2,
                       alpha=0.5) +
       geom_node_text(aes(label = name),
                       repel=T,
                       angle=90, hjust=2, ) +
       scale_edge_colour_manual(
           values=c('indianred3', 'wheat4', 'grey', 'grey30',
                   'red','blue','green','orange','purple','brown','pink'
                   ),
           name='Event Type') +
       theme_void()

   return(visualization)

}
```



```
PassNetwork <- function(events, match_id, team_name, team_colour='red', home_or_away='Home', flip=F, lower_threshold=1,
high_threshold=10) {
   #' Draw Pass Map of First XI with nodes placed on mean (x,y) pitch-coordinates.

   game <- events[events$matchId == match_id,]
   goals <- game[game$subEventName=='Goal',]

   # note teams
   teams <- unique(game$team)

   # limit events to team specified by name or home/away
   if(!missing(team_name)){
       game <- game[game$team==team_name,]
       home_or_away <- unique(game$home_or_away)
   } else {
       game <- game[game$home_or_away==home_or_away,]
       team_name <- unique(game$team)
   }

   # note opposition
   opposition <- teams[teams!=team_name]

   # get score
   team_score <- table(goals$team)[team_name]
   opposition_score <- table(goals$team)[opposition]



   firstXI <- game[game$FirstXI == 'True',]
   mean_positions <- firstXI[firstXI$location_x>0 & firstXI$location_y>0 & firstXI$location_x<105 & firstXI$location_y<68,] %>%
       group_by(team, matchId, source) %>%
         dplyr::summarise(x_mean = mean(location_x), y_mean = mean(location_y)) %>%
         ungroup() %>%
         mutate(team = as.factor(team), id = as.factor(matchId)) %>%
         as.data.frame()



   pass_counts <- ddply(data.frame(game$source, game$target),.(game.source,game.target),nrow)
   names(pass_counts) <- c('source','target','passcount')
   step1 <- merge(mean_positions, pass_counts, by='source')
   step2 <- step1[,c(1,4,5,7,8)]
   names(step2)[2:3] <- c('source_x','source_y')
   names(mean_positions)[3] <- 'target'
   step3 <- merge(mean_positions, step2, by='target')
   team <- step3[step3$team==team_name,]

   if (flip==T) {
       team$source_x <- 105 - team$source_x
       team$x_mean <- 105 - team$x_mean
       team$source_y <- 68 - team$source_y
       team$y_mean <- 68 - team$y_mean
       mean_positions$x_mean <- 105 - mean_positions$x_mean
       mean_positions$y_mean <- 68 - mean_positions$y_mean
   }

   p <- (DrawPitch() +
       geom_segment(data=team[team$passcount>=lower_threshold,],
```

```
                     size=1, colour=team_colour,
                     aes(x=source_x, y=source_y,
                         xend=x_mean, yend=y_mean, alpha=passcount)) +
        geom_segment(data=team[team$passcount>=high_threshold,],
                     size=1.5, colour='black',
                     aes(x=source_x, y=source_y,
                         xend=x_mean, yend=y_mean, alpha=passcount)) +
        geom_label_repel(data = mean_positions[mean_positions$team==team_name,],
                         aes(x_mean, y_mean, label = target),
                         label.padding=0.5, size=4, alpha=0.8) +
        geom_point(data=team, aes(x_mean, y_mean,),
                   fill=team_colour, colour='black', pch=21, size=3) +
        ggtitle(paste('Passing Network:',team_name,home_or_away,
                      'versus',opposition,'(',team_score,'-',opposition_score,')')) +
        theme(legend.position=c(0.5,0.07), legend.direction='horizontal'))

    p$figname <- paste0('PassingNetwork',home_or_away)

    return(p)
}
```



```
GameGraph <- function(events, match_id, team_name, home_or_away='Home') {
   #' Return Possession Graph for given Match and Team.

   game <- events[events$matchId == match_id,]
   game <- game[game$source!='' & game$target!='' & game$target!='nan',]

    # note teams
   teams <- unique(game$team)

   # limit events to team specified by name or home/away
   if(!missing(team_name)){
       team_game <- game[game$team==team_name,]
       home_or_away <- unique(team_game$home_or_away)
   } else {
       team_game <- game[game$home_or_away==home_or_away,]
       team_name <- as.character(unique(team_game$team))
   }

   opposition_game <- game[game$team != team_name,]
   opposition_team <- as.character(unique(opposition_game$team))

   team_nodes <- unique(c(as.character(team_game$source),as.character(team_game$target)))
   team_edges <- data.frame(team_game$source, team_game$target)
   team_graph <- graph_from_data_frame(d=team_edges, vertices=team_nodes, directed=TRUE)
   team_adj <- as.matrix(team_graph[])

   possession_seqs <- unique(team_game$possession)

   seq_start <- c()

   for (pseq in possession_seqs){
       if (sum(team_game$possession==pseq)>1){
           player <- as.character(team_game[team_game$possession==pseq,][1,]$source)
           if (player!='' & player!='nan'){
               seq_start <- append(seq_start, player)
           }
       }
   }

   start_counts <- table(as.factor(seq_start))

   for (i in (2:(length(start_counts)-1))){
       team_adj[nrow(team_adj)-1,][names(start_counts[i])] <- as.numeric(start_counts[i])
   }

   opposition_status <- as.character(unique(opposition_game$home_or_away))
   opp_to_opp <- as.numeric(summary(game[game$team != team_name,]$to_team)[opposition_status])

   opp_nodes <- unique(c(as.character(opposition_game$source),as.character(opposition_game$target)))
   opp_edges <- data.frame(opposition_game$source, opposition_game$target)
   opp_graph <- graph_from_data_frame(d=opp_edges, vertices=opp_nodes, directed=TRUE)
   opp_adj <- as.matrix(opp_graph[])

   opp_to_opp <- sum(opp_adj[,ncol(opp_adj)-1])
   team_adj[(nrow(team_adj)-1),(ncol(team_adj)-1)] <- opp_to_opp
```

```
    team_status <- unique(team_game$home_or_away)
    opposition_own_goals <- as.numeric(nrow(game[game$home_or_away==opposition_status &
game$to_team==paste(team_status,'Goal'),]))
    team_adj[(nrow(team_adj)-1),(ncol(team_adj))] <- opposition_own_goals

    team_graph <- graph_from_adjacency_matrix(team_adj, mode='directed')

    team_graph$team <- team_name
    team_graph$opposition <- opposition_team
    team_graph$status <- home_or_away

    return(team_graph)
}
```

```
AdjacencyMatrix <- function(graph){
    #' Return Adjacency Matrix for Graph

    return(as.matrix(graph[]))
}
```

```
TransitionMatrix <- function(graph){
    #' Return Transition Matrix for Graph considered as time-homogeneous Markov Process

    matrix <- as.matrix(graph[])

    for (i in 1:nrow(matrix)){
        matrix[i,] <- matrix[i,]/sum(matrix[i,])
    }

    matrix[nrow(matrix),ncol(matrix)] <- 1
    matrix[nrow(matrix),] <- rep(0, ncol(matrix))

    # treat 'Goal' as absorbtion state
    matrix[nrow(matrix),ncol(matrix)] <- 1

    return(matrix)
}
```

```
MatrixHeatMap <- function(matrix, color, number_format, title){
    #' Show Matrix as HeatMap

    hm <- pheatmap(matrix,color=color,
                   cluster_rows=F,cluster_cols=F,legend=F,
                    display_numbers=T,number_format=number_format,
                   fontsize_number=9,angle_col='315',
                    main=title)

    return(hm)
}
```



```
ExpectedScoringTime <- function(transition_matrix){
    #' Calculate expected scoring time based on transition matrix.

    max_steps <- 10000
    hitting_times <- rep(0,(ncol(transition_matrix)-2))

    for (i in (1:(ncol(transition_matrix)-2))){

        state_probabilities <- matrix(NA,
                                      nrow=max_steps+1,
                                      ncol=ncol(transition_matrix),
                    dimnames=list(0:max_steps,(ncol(transition_matrix)-1):0))
        vector <- rep(0,ncol(transition_matrix))

        vector[i] <- 1
        state_probabilities[1,] <- vector

        for ( kk in 1:max_steps ) {
            state_probabilities[kk+1,] <- t(transition_matrix)%*%state_probabilities[kk,]
        }

        probs <- diff(state_probabilities[,ncol(transition_matrix)])
        hitting_time <- sum(probs*seq_along(probs))

        hitting_times[i] <- hitting_time
    }

    names(hitting_times) <- names(transition_matrix[1,])[1:(nrow(transition_matrix)-2)]

    return(hitting_times)

}
```

```
SeasonAdjacencyMatrix <- function(events, team) {
    #' Return Adjacency Matrix for Team's full season.

    players <- unique(events[events$team==team & events$source!='' & events$target!='',]$source)
    n <- length(players) + 2
    squad_adj <- matrix(0,nrow=n,ncol=n)
    rownames(squad_adj) <- c(as.character(players), 'Opposition', 'Goal')
    colnames(squad_adj) <- rownames(squad_adj)

    games <- unique(events[events$team==team,]$matchId)

    for (game in games){

        # get graph and adjacency matrix for game
        game_graph <- GameGraph(events, game, team)
        game_adj <- AdjacencyMatrix(game_graph)

        # add values of game adjacency matrix to full-season matrix
        for (source in rownames(game_adj)){
            for (target in colnames(game_adj)){
                squad_adj[source,target] <- squad_adj[source,target] + game_adj[source,target]
            }
        }
    }

    return(squad_adj)
}
```

```
MarkovExpectedGoals <- function(events, season_adjacency_matrix) {
    #' Calculate MXG for each player from team's Season Adjacency Matrix.

    season_graph <- graph_from_adjacency_matrix(season_adjacency_matrix)
    season_transition <- TransitionMatrix(season_graph)
    seasonXST <- ExpectedScoringTime(season_transition)
    seasonMXG <- 1/seasonXST
    stdize <- seasonMXG/(mean(seasonMXG))
    avg_events_per_game <- sum(season_adjacency_matrix)/38
    scale <- stdize * avg_events_per_game - avg_events_per_game

    return(scale)
}
```


08/09/2021 Random Walks across Social Networks: The Markovian Movement of a Stochastic Soccerball

```
MXGTable <- function(events, team){
    #' Return dataframe with Degree, Loops, Goals, and MXG for each player in team.

    Team <- team
    m <- SeasonAdjacencyMatrix(events,Team)
    Degree <- rowSums(m)[1:(nrow(m)-2)]
    Loops <- diag(m)[1:(nrow(m)-2)]
    Goals <- m[,ncol(m)][1:(nrow(m)-2)]
    Player <- rownames(m)[1:(nrow(m)-2)]
    MXG <- MarkovExpectedGoals(events, m)[1:(nrow(m)-2)]
    df <- data.frame(Player, Team, MXG, Goals, Loops, Degree)
    rownames(df) <- c()

    return(df)
}
```

```
CalculateMXG <- function() {
    #' Get MXG and other stats for each player in all leagues.

    mxgtable_df <- data.frame(Player = character(),
                    Team = character(),
                    MXG = double(),
                    Goals = integer(),
                    Loops = integer(),
                    Degree = integer())

    leagues = c('England','France','Germany','Italy','Spain')
    for (league in leagues){
        events <- LoadAndScaleData(league)
        for (team in unique(events$team)){
            mxgtable_df <- rbind(mxgtable_df, MXGTable(events,team))

        }
    }

    write.csv(mxgtable_df[order(-mxgtable_df[,'MXG']),], file='../tables/MXG.csv', row.names=F)

}
```

```
TallyEventTypes <- function(events, eventType=F){
    #' Return dataframe with tallies for specified type.

    if (eventType!=F){
        events <- events[events$eventName==eventType,]
        df <- data.frame(table(events$subEventName))
        df <- df[df$Freq>0,]
    } else {
        df <- data.frame(table(events$eventName))
    }

    df <- df[order(-df$Freq),]
    colnames(df)[1:2] <- c('Description','Frequency')

    df$Description <- as.character(df$Description)

    total <- sum(df$Frequency)
    df[nrow(df)+1,] <- c('**Total**',total)
    df$Frequency <- as.numeric(df$Frequency)

    df$Pct <- df$Frequency / total * 100
    rownames(df) <- c()
    colnames(df)[3] <- '%'

    df[,3] <- formatC(signif(df[,3],digits=3), digits=3,format="fg", flag="#")

    df[,'Frequency (%)'] <- paste0(as.character(
                            prettyNum(df$Frequency, big.mark=",", scientific=F)),
                           ' (', as.character(df[,3]),'%)')

    df <- df[,c(1,4)]

    return(df)
}
```

```
ExploratoryTallies <- function(){
    #' Get exploratory tallies.



    events <- LoadAndScaleData('England')

    leagues = c('England','France','Germany','Italy','Spain')
    for (league in leagues[2:5]){
        events <- rbind(events, LoadAndScaleData(league))
    }

    types <- TallyEventTypes(events)
    passes <- TallyEventTypes(events,'Pass')
    shots <- TallyEventTypes(events,'Shot')

    write.csv(types,'../tables/EventTypeTally.csv',row.names=F)
    write.csv(passes,'../tables/PassTally.csv',row.names=F)
    write.csv(shots,'../tables/ShotTally.csv',row.names=F)

}
```


08/09/2021 Random Walks across Social Networks: The Markovian Movement of a Stochastic Soccerball

```
SavePairedFigures <- function(league, match_id){
    #' Generate and Save Paired Figures.

    events <- LoadAndScaleData(league)

    EventMapHome <- ShowMatchEvents(events, match_id)
    EventMapAway <- ShowMatchEvents(events, match_id, home_or_away='Away', flip=T)

    PassMapHome <- ShowPassesAndShots(events, match_id)
    PassMapAway <- ShowPassesAndShots(events, match_id, home_or_away='Away', flip=T)

    PassingNetworkHome <- PassNetwork(events, match_id, team_colour='blue')
    PassingNetworkAway <- PassNetwork(events, match_id, home_or_away='Away',
                                      team_colour='red',flip=T)

    pairedFigures <- list(EventMapHome, EventMapAway,
                          PassMapHome, PassMapAway,
                          PassingNetworkHome, PassingNetworkAway)

    for (fig in pairedFigures){
        png(filename=paste0('../figures/',fig$figname,'.png'),
            width=800,
            height=650,
            units='px',
           pointsize=4,
           res=140)
        print(fig)
        dev.off()

    }
}
```

```
CountSequences <- function(events) {
    #' Count Possession Sequences and return DataFrame.

    poss_seqs <- data.frame(table(events$possession))
    df_psq <- data.frame(table(poss_seqs$Freq))
    goals_seqs <- events[events$subEventName == 'Goal',]$possession
    goals_seqs_events <- events[events$possession %in% goals_seqs,]
    df_gpsq0 <- data.frame(table(goals_seqs_events$possession))
    df_gpsq <- df_gpsq0[df_gpsq$Freq>0,]
    df_g <- data.frame(table(df_gpsq$Freq))

    m <- rbind(1:10, df_psq$Freq[1:10], df_g$Freq[1:10])
    mdf <- data.frame(m)
    rownames(mdf) <- c('Sequence Length', 'Frequency', 'Goal Scoring')

    total_goals <- sum(df_g$Freq)
    total_seqs <- sum(df_psq$Freq)
    final <- c('**Total**', total_seqs, total_goals)
    mdf <- cbind(mdf,final)
    colnames(mdf) <- c()

    df <- t(mdf)
    df[,2] <- prettyNum(df[,2], big.mark=",", scientific=F)
    df[,3] <- prettyNum(df[,3], big.mark=",", scientific=F)
    return(df)
}
```

```
PossessionSequence <- function(league, sequence){
    #' Generate Figures and Table for given Possession Sequence.

    events <- LoadAndScaleData(league)
    write.csv(TabulateSequence(events, sequence),'../tables/PossessionSequence.csv',row.names=F)
    png(filename=paste0('../figures/PossessionSequence.png'),
        width=800,
        height=600,
        units='px',
       pointsize=12,
       res=140)
    print(SequenceOnPitch(events, sequence))
    dev.off()

    g <- SequenceGraph(events, sequence)
    png(filename=paste0('../figures/PossessionGraph.png'),
        width=800,
        height=600,
        units='px',
       pointsize=12,
       res=140)
    print(VisualizeGraph(g))
    dev.off()
}
```



```
Top20MXG <- function(){
    #' Save CSV with top 20 players by MXG score.

    d <- read.csv('../tables/MXG.csv')
    top <- d[d$Degree>1000,]
    rownames(top) <- c()
    top['Loops/Degree'] <- top$Loops/top$Degree
    top['Goals/Degree'] <- top$Goals/(top$Degree)
    t <- top[c('Player','Team','Degree','Goals',
               'Goals/Degree','Loops','Loops/Degree','MXG')]
    write.csv(t[1:20,], '../tables/2.csv',row.names=F)
}
```

```
SaveMatrixHeatmaps <- function(league, match_id, team_name){
    #' Save Heatmaps for Adjacency and Transition Matrices.

    events <- LoadAndScaleData(league)
    g <- GameGraph(events, match_id, team_name)
    adj_m <- AdjacencyMatrix(g)
    tr_m <- TransitionMatrix(g)

    matrix_cols <- c('darkgrey',colorRampPalette(c('white','red'))(35))
    adj_m_hm <- MatrixHeatMap(adj_m[1:11,1:11], color=matrix_cols, number_format='%.0f',
                  title=paste('Adjacency Matrix for',g$team,'(Starting XI)\n',
                              g$status,'vs.',g$opposition))
    adj_m_hm$type <- 'Adjacency'

    tr_m_hm <- MatrixHeatMap(tr_m, color=matrix_cols[2:36], number_format='%.2f',
                  title=paste('Transition Matrix for',g$team,g$status,'vs.',g$opposition))
    tr_m_hm$type <- 'Transition'

    hm_vec <- list(adj_m_hm, tr_m_hm)

    for (hm in hm_vec){
        png(filename=paste0('../figures/',hm$type,'MatrixAway.png'),
            width=1600,
            height=1600,
            units='px',
           pointsize=12,
           res=240)

        print(hm)

        dev.off()
    }

}
```

```
GenerateTablesAndFigures <- function(league, match_id, possession_seq, team){
    #' Generate Tables and Figures for Paper.

    ExploratoryTallies()
    SavePairedFigures(league, match_id)
    PossessionSequence(league, possession_seq)
    SaveMatrixHeatmaps(league, match_id, team)
    CalculateMXG()
    Top20MXG()
}
```

```
GenerateTablesAndFigures('England','2500032','2500032-91-Liv-0','Liverpool')
```

Appendix 2: Preprocessing in Python

▼ Code

```
import pandas as pd
import matplotlib.pyplot as plt
import json
import numpy as np

import warnings
warnings.filterwarnings('ignore')
```

```
tag_df = pd.read_csv('../data/raw/tagcodes.csv')
tag_dict = dict(zip(tag_df.Tag, tag_df.Description))
```

```
def load_json(file):
    "Load raw JSON data."
    with open(f'../data/raw/{file}.json','r') as f:
        return json.loads(f.read().replace('\\\\','\\').encode('utf-8'))
```



```
def preprocess(national_league):
    "Create processed CSV from raw JSON for WyScout national league event-logs."

    # load JSON event-logs
    events = load_json(f'events_{national_league}')

    # create dataframe from JSON event-logs
    ev_df = pd.DataFrame(events)

    # decode tags
    for i in range(6):
        ev_df[f'tag{i}'] = ev_df['tags'].str[i].str['id'].map(tag_dict).str.lower()

    # decode player ids
    players = load_json('players')
    players_dict = dict(zip(pd.DataFrame(players).wyId,pd.DataFrame(players).shortName))
    ev_df['player'] = ev_df['playerId'].map(players_dict)

    # decode team ids
    teams = load_json('teams')
    teams_dict = dict(zip(pd.DataFrame(teams).wyId,pd.DataFrame(teams).name))
    ev_df['team'] = ev_df['teamId'].map(teams_dict)

    # unpack location and target
    ev_df['location'] = ev_df['positions'].str[0]
    ev_df['target'] = ev_df['positions'].str[1]

    # create simpler dataframe
    e_df =
ev_df[['matchId','eventSec','matchPeriod','team','location','target','player','eventName','subEventName','tag0','tag1','tag2','ta
g3','tag4','tag5']]
    e_df['notes'] = e_df[['tag0','tag1','tag2','tag3','tag4','tag5']].values.tolist()
    e_df['tags'] = e_df['notes'].astype(str)
    e_df['accurate'] = e_df.tags.str.contains('accurate')
    e_df['accurate'] = np.where(e_df.tags.str.contains('not accurate'), 0, e_df.accurate)
    e_df.drop(columns=['tag0','tag1','tag2','tag3','tag4','tag5','notes'],inplace=True)

    # label in_play sequences
    play = 0
    plays = []
    next_action = []
    match_play = []
    matchId = e_df.iloc[0].matchId
    for i in range(len(e_df)):
        if i>1:
            row_minus2 = row_minus1
        if i>0:
            row_minus1 = row
        row = e_df.iloc[i]
        if i == 0:
            row_minus1 = row
            row_minus2 = row

        if row.eventName == 'Free Kick':
            # when the ball goes 'out of play', the game resumes with a 'free kick' (here includes Throw-Ins)
            play += 1
```

```
        elif 'goal' in row_minus2.tags and 'goal' in row_minus1.tags:
            # (there are generally two events marked goal -- the shot and the attempted save)
            play +=1

        if row.matchId == matchId:
            pass
        else:
            play = 0
            matchId = row.matchId
        plays.append(play)
        match_play.append(f'{matchId}-{play}')

    e_df['in_play'] = pd.Series(plays)
    e_df['match_play'] = pd.Series(match_play)

    # label possession sequences
    ps_sequence = np.full(len(e_df),'0000000000000000000000000000000000')
    played_to = np.full(len(e_df), '                                    ')
    to_team = np.full(len(e_df), '                                        ')

    for match in e_df.matchId.unique():
        match_df = e_df.loc[e_df.matchId == match]
        for team in match_df.team.unique():
            for play in match_df.in_play.unique():
                team_play_df = match_df.loc[match_df.in_play == play].loc[match_df.team == team]
                possession_sequence = 0
                for i in range(len(team_play_df)):
                    if i>0:
                        previous_row = row

                    row = team_play_df.iloc[i]


                    if i > 0:
                        if row.location != target:
                            possession_sequence += 1
                            played_to[previous_row.name] = 'Opposition'
                            to_team[previous_row.name] = 'Opposition'
                        else:
                            played_to[previous_row.name] = row.player
                            to_team[previous_row.name] = row.team
                    team_name = team.split(' ')
                    team_abbr = ''
                    for word in team_name:
                        team_abbr = team_abbr + word[:3]
                    ps_sequence[row.name] = f'{match}-{play}-{team_abbr}-{possession_sequence}'
                    target = row.target

    e_df['possession'] = ps_sequence
    e_df['played_to'] = played_to
    e_df['to_team'] = to_team

    e_df['played_to'] = np.where(e_df['played_to'] == '                                    ', 'Opposition', e_df['played_to'])
    e_df['to_team'] = np.where(e_df['to_team'] == '                                        ', 'Opposition', e_df['to_team'])

    # unpack x and y coorinates
    for foo in ('location','target'):
```



```
        for goo in ('x','y'):
            e_df[f'{foo}_{goo}'] = e_df[foo].str[goo]

    # change target coordinates of goal to inside goal
    e_df['target_x'] = np.where((e_df.tags.str.contains("'goal'")) & (e_df.eventName != 'Save attempt'),
                                100, e_df.target_x )
    e_df['target_y'] = np.where((e_df.tags.str.contains("'goal'")) & (e_df.eventName != 'Save attempt'),
                                50, e_df.target_y )

    e_df['subEventName'] = np.where((e_df.tags.str.contains("'goal'")) & (e_df.eventName != 'Save attempt'),
                                'Goal', e_df.subEventName )

    e_df['eventName'] = np.where(e_df['eventName'] == 'Others on the ball', 'On the Ball', e_df['eventName'])

    e_df['eventName'] = np.where(e_df['eventName'] == 'Duel', 'Challenge', e_df['eventName'])

    e_df['played_to'] = np.where(e_df.subEventName == 'Goal', 'Goal', e_df.played_to)
    e_df['to_team'] = np.where(e_df.subEventName == 'Goal', 'Goal', e_df.to_team)

    # change location coordinates of goal kick
    e_df['location_x'] = np.where((e_df.subEventName == 'Goal kick'), 5, e_df.location_x)
    e_df['location_y'] = np.where((e_df.subEventName == 'Goal kick'), 50, e_df.location_y)

    # add time in minutes and seconds
    e_df['minute'] = e_df.eventSec//60
    e_df.minute = e_df.minute.astype(int)
    e_df['seconds'] = e_df.eventSec % 60
    e_df.seconds = e_df.seconds.astype(int)
    e_df['time'] = e_df.minute.astype(str).str.zfill(2) + ':' + e_df.seconds.astype(str).str.zfill(2)
    e_df.drop(columns=['minute','seconds'],inplace=True)

    e_df.drop(columns=['location','target'], inplace=True)
    e_df.rename(columns={'player':'source','played_to':'target'},inplace=True)

    # include starting lineup data
    matches = load_json(f'matches_{national_league}')
    match_df = pd.DataFrame(matches)
    match_lineups = {}
    for match in match_df.wyId.unique():
        match_lineups[match] = {}
        for team in match_df.loc[match_df.wyId == match].teamsData.values[0].keys():
            match_lineups[match][teams_dict[int(team)]] = list(pd.Series(pd.DataFrame(match_df.loc[
                match_df.wyId ==
match].teamsData.str[str(team)].str['formation'].str['lineup'].values[0]).playerId.values).map(players_dict))

    def firstXI(row):
        return row.source in match_lineups[row.matchId][row.team]

    e_df['FirstXI'] = e_df.apply(firstXI, axis=1)

    # include home_or_away data
    h_a_map = {}
    for match in match_df.wyId.unique():
        h_a_map[match] = {}
        for team in match_df.loc[match_df.wyId == match].teamsData.values[0].keys():
            status = match_df.loc[match_df.wyId==match].teamsData.str[str(team)].str['side'].values[0].capitalize()
            h_a_map[match][team] = status
```

```
    def home_or_away(row):
        return h_a_map[row.matchId][str(row.teamId)]

    e_df['home_or_away'] = ev_df.apply(home_or_away, axis=1)
    e_df['to_team'] = np.where(e_df.to_team == e_df.team, e_df.home_or_away, e_df.to_team)
    opposition = {'Home':'Away','Away':'Home'}
    e_df['to_team'] = np.where(e_df.to_team == 'Opposition', e_df.home_or_away.map(opposition), e_df.to_team)
    e_df['to_team'] = np.where(e_df.tags.str.contains('own goal'), 'Own Goal', e_df.to_team)
    whose_goal = {'Home':'Home Goal', 'Away':'Away Goal'}
    e_df['to_team'] = np.where(e_df.to_team == 'Goal', e_df.home_or_away.map(whose_goal), e_df.to_team)
    whose_own_goal = {'Home':'Away Goal', 'Away':'Home Goal'}
    e_df['to_team'] = np.where(e_df.to_team == 'Own Goal', e_df.home_or_away.map(whose_own_goal), e_df.to_team)

    e_df.drop(columns=['tags','in_play'],inplace=True)

    e_df.to_csv(f'../data/processed/{national_league}FootballLeague2017-18.csv')
```

```
national_leagues = ('England','France','Germany','Italy','Spain')
```

```
if __name__ == '__main__':
    for league in national_leagues:
        preprocess(league)
```

Bibliography

Austin, S. 2020. Ian Graham: How Liverpool integrate data, analysis and coaching. *Training Ground Guru*.

BBC. 2020. Liverpool win Premier League: Reds' 30-year wait for top-flight title ends. *BBC Sport*.

Cintia, P.; Giannotti, F.; Pappalardo, L.; Pedreschi, D.; Malvaldi, M. 2015. The harsh rule of the goals: Data-driven performance indicators for football teams. *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*: 1–10.

Deloitte. 2019. *European football market worth EUR 28.4 billion as Premier League clubs lead the way to record revenues.*.

Grund, T.U. 2012. Network structure and team performance: The case of English Premier League soccer teams. *Social Networks* 34: 682–690.

Knutson, T. 2018. Explaining xGChain Passing Networks. *StatsBomb*.

Norris, J.R. 1998. *Markov Chains*. Cambridge University Press, Cambridge,.

Palacios-Huerta, I. 2004. Structural changes during a century of the world's most popular sport. *Statistical Methods & Applications* 13.

Pappalardo, L.; Massucco, E. 2019. Soccer Match Event Dataset..

Peng, R.D. 2011. Reproducible research in computational science. *Science* 334: 1226–1227.

08/09/2021 Random Walks across Social Networks: The Markovian Movement of a Stochastic Soccerball

Peña, J.L. 2014. A Markovian model for association football possession and its outcomes. *ArXiv:1403.7993 [math, stat]*.

Pettersen, S.A.; Halvorsen, P.; Johansen, D.; Johansen, H.; Berg-Johansen, V.; Gaddam, V.R.; et al. 2014. Soccer video and player position dataset. *Proceedings of the 5th ACM Multimedia Systems Conference on - MMSys '14*: 18–23.

Pollard, R. 2002. Charles Reep (1904-2002): Pioneer of notational and performance analysis in football. *Journal of Sports Sciences* 20: 853–855.

Reep, C.; Benjamin, B. 1968. Skill and Chance in Association Football. *Journal of the Royal Statistical Society. Series A (General)* 131: 581.

Schoenfeld, B. 2019. How Data (and Some Breathtaking Soccer) Brought Liverpool to the Cusp of Glory. *The New York Times*.

Serfozo, R. 2009. *Basics of applied stochastic processes*. Springer Science & Business Media,.

Tavares, R. 2017. Why We Need Positional Data. *StatsBomb*.

Wasserman, S.; Faust, K. 1994. *Social Network Analysis: Methods and Applications*. Cambridge University Press,.

Wickham, H. 2014. Tidy Data. *Journal of Statistical Software* 59: 1–23.