Dokumentation von datenflanke.de

Release 0.3

Christoph Debowiak

21.06.2024

Einführung:

KAPITEL 1

Vom Scouting zum Datenscouting

1.1 Vorbemerkung

Da mich im Laufe der Zeit immer wieder Anfragen zu meinen Projekten erreichen, habe ich mich entschlossen, eine Dokumentation zu schreiben, die eines meiner Projekte und die dahinterliegenden Methoden und Techniken erläutert. Diese Dokumentation ist also einerseits für diejenigen, die sich für Fußball und Datenanalyse interessieren, und andererseits auch für jene, die sich dafür interessieren, wie so ein Projekt in der Praxis umgesetzt werden kann.

Das heißt, ich werde immer wieder auf fußballerische inhaltliche Themen eingehen, aber auch auf technische Aspekte wie die Datenbeschaffung, -aufbereitung und -speicherung etc. mit Codebeispielen.

Das hier umgesetzte Projekt dient als Beispiel! Es ist kein fertiges Produkt, sondern eher ein Prototyp. In einer realen Scoutingumgebung würden die Definitionen der Spielerqualitäten und die daraus resultierenden Bewertungen individuell auf die Bedürfnisse des Vereins angepasst werden, woraus dann auch der Wettbewerbsvorteil entstehen kann im Vergleich zu Bewertungen von Drittanbietern, die jeder Verein kaufen kann. Hier soll nur ein allgemeiner Überblick über die Möglichkeiten und Herausforderungen gegeben werden.

1.2 Einführung und persönliche Motivation

Als ehemaliger Jugendfußballer im Leistungsnachwuchszentrum und später im semi-professionellen Seniorenbereich beschäftige ich mich seit jeher intensiv mit Fußball. Während des Studiums der Sportwissenschaft und später im Masterstudium Spielanalyse habe ich mich intensiv mit der Analyse von Fußballspielen beschäftigt und vor allem mit der Frage nach einer (objektiven) Bewertung.

Bereits während des Studiums und in der Zeit danach als Spielanalyst und im Scoutingbereich habe ich festgestellt, dass die Kombination von subjektiven Eindrücken und objektiven Daten die beste Grundlage für eine fundierte Bewertung von Spielern darstellt. Eine subjektive Wahrnehmung hat jeder von uns, auch wenn wir uns dabei oft von vielen Faktoren beeinflussen lassen (siehe Biases). Die Fähigkeit, Daten zu analysieren und statistisch-mathematische Modelle zu entwickeln, um Spieler zu bewerten, ist eine Fähigkeit, die ich mir über viele Jahre angeeignet habe und hier in einem meiner Projekte veranschaulichen möchte.

Ich werde einige Aspekte erläutern, die für die Bewertung von Spielern wichtig sind. Dabei werde ich jedoch nicht alles im Detail abdecken können, da es den Rahmen dieser Dokumentation sprengen würde. Vielmehr möchte ich hier einen Leitfaden darstellen und dabei neue Anregungen für die Bewertung und deren Umsetzung mitgeben.

KAPITEL 2

3 Arten des Scoutings

Scouting ist ein grundlegender Bestandteil des Fußballs und konzentriert sich darauf, Talente zu identifizieren und zu bewerten. Es gibt verschiedene Methoden des Scoutings, die jeweils auf unterschiedliche Ziele und Kontexte zugeschnitten sind. Traditionelles Scouting stützt sich auf das geschulte Auge erfahrener Scouts, um die technischen Fähigkeiten, körperlichen Attribute und die psychologische Verfassung von Spielern zu beurteilen. Modernes Scouting integriert Datenanalysen, nutzt statistische Modelle und Videoanalysen, um Einblicke zu gewinnen, die dem bloßen Auge entgehen könnten. Gemeinsam sorgen diese Ansätze für eine umfassende Bewertung und liefern den Vereinen die notwendigen Informationen, um fundierte Entscheidungen über die Spielerrekrutierung und -entwicklung zu treffen.

Es gibt 3 verschiedene Wege, Spieler zu scouten:

- **Live-Scouting:** Beobachtung von Spielen in Echtzeit.
- **Video-Scouting:** Analyse von aufgezeichneten Spielen.
- **Daten-Scouting:** Nutzung von Daten und statistischen Methoden zur Bewertung von Spielern.

2.1 Live-Scouting

Die Vorteile des **Live-Scoutings** sind die direkte Beobachtung der Spieler und die Möglichkeit, die Interaktionen der Spieler untereinander zu beobachten. Alle Dinge, die eine Kamera nicht einfangen kann, wie die Kommunikation der Spieler, einige Laufwege ohne Ball oder die Reaktionen auf bestimmte Spielsituationen, die Körpersprache uvm.

Nachteile sind die begrenzte Anzahl von Spielen, die ein Scout live sehen kann, die subjektive Wahrnehmung, die durch unterschiedliche Biases beeinflusst wird, und die hohen Kosten, die durch Reisen und Unterkunft entstehen.

2.2 Video-Scouting

Ein großer Vorteil des **Video-Scoutings** ist die große Verfügbarkeit von Spielen bis in den Jugendbereich und selbst in exotischen ausländischen Ligen. Videos können beliebig oft angesehen werden und sind von vielen Anbietern sogar schon nach Spielaktionen vorsortiert.

Nachteile sind oft die mangelnde Qualität der Videos, die fehlende Möglichkeit, die Kameraführung zu beeinflussen und die fehlende Möglichkeit, die Interaktionen der Spieler untereinander zu beobachten. Viele Videofeeds zeigen nur Aktionen mit Ball und nicht die gesamte Spielsituation. So ist es schwierig, Spieler zu bewerten, wie sie sich ohne Ball verhalten, was immerhin 90-95% der individuellen Spielzeit ausmacht und oft ein entscheidender Unterschied zwischen einem durchschnittlichen und einem guten Spieler ist.

2.3 Daten-Scouting

Daten-Scouting ist die jüngste Entwicklung im Scouting und hat in den letzten Jahren enorm an Bedeutung gewonnen. Daten-Scouting ermöglicht es, eine große Anzahl von Spielern sehr schnell zu bewerten. Dadurch kann das Daten-Scouting einerseits als Filter dienen, um Spieler zu identifizieren, die es wert sind, genauer beobachtet zu werden (im Video- und/oder Livescouting), und andererseits als objektive Bewertungsmethode, um Spieler zu identifizieren, die vorher nicht auf dem eigenen Radar waren.

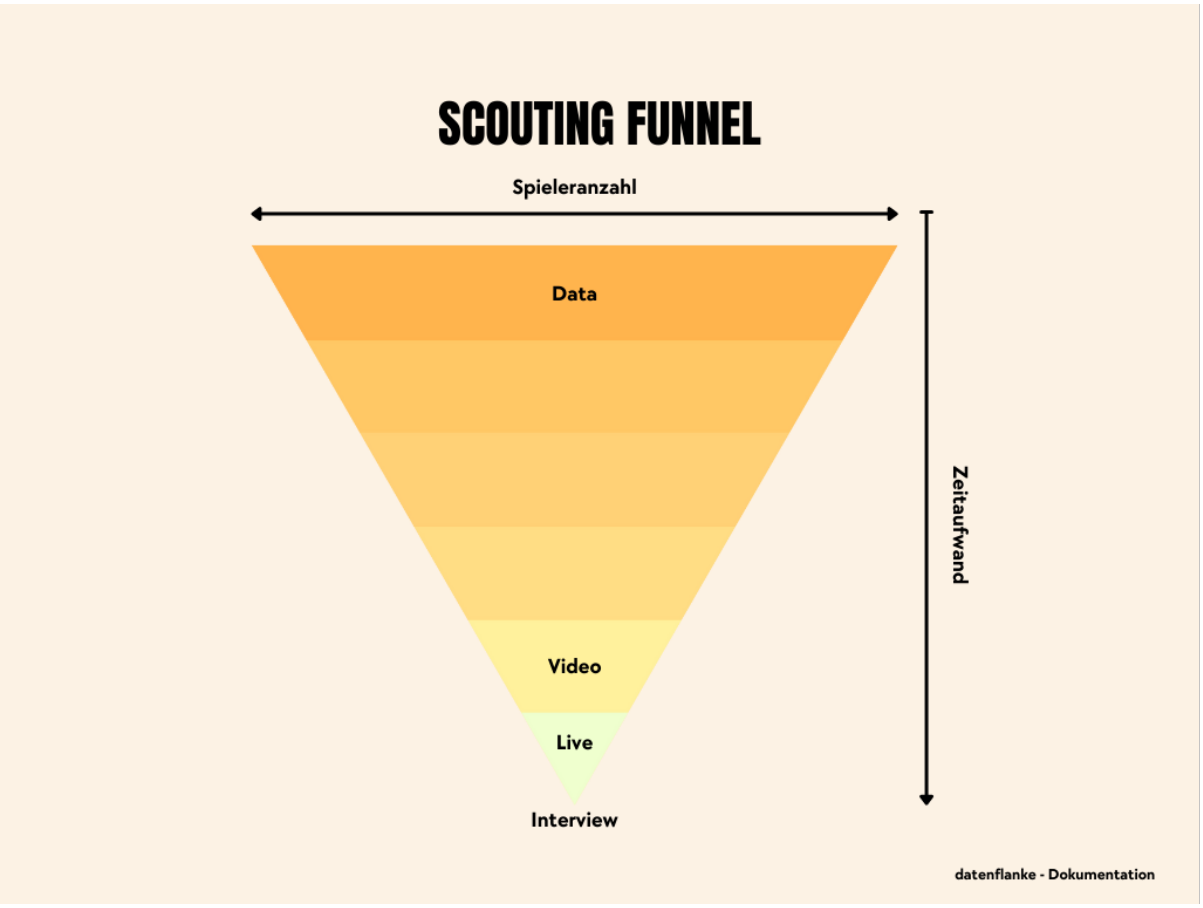
Daten-Scouting hat jedoch auch seine Nachteile. Daten können nicht alles erfassen, was im Fußball wichtig ist. Auch die Qualität der Daten spielt eine entscheidende Rolle, denn die Algorithmen sind nur so gut, wie die Daten, die ihnen zugrunde liegen. Doch auch hier gibt es mittlerweile viele Anbieter, die qualitativ hochwertige Daten anbieten und die Datenqualität wird immer besser, durch moderne Technologien und die stetige Weiterentwicklung in der Datenerfassung.

tabularytabulary

	Live-Scouting		Video
	Vorteile:		
	- Direkte Beobachtung		- Große Ver
	- Interaktionen der Spieler		- Wieder
	- Körpersprache		- Vorsortierte Spi
	Nachteile:		
	- Begrenzte Anzahl Spiele		- Mangeln
	- Subjektive Wahrnehmung		- Fehlende Kam
	- Hohe Kosten		- Kein gesamtes Spiel

2.4 Kombination der Scouting-Methoden

Ein umfassendes Scouting sollte alle drei Bereiche – Live-Scouting, Video-Scouting und Daten-Scouting – kombinieren, da jede Methode einzigartige Einblicke und Vorteile bietet: Daten-Scouting identifiziert schnell potenzielle Talente, Video-Scouting ermöglicht eine detaillierte Analyse spezifischer Spielsituationen, und Live-Scouting bietet unverzichtbare Beobachtungen der Spielerinteraktionen und -dynamiken vor Ort. Durch die Integration dieser Methoden können Entscheidungsträger eine vollständige und ausgewogene Bewertung der Spielerleistungen erhalten.


SCOUTING FUNNEL
Spieleranzahl
Data
Video
Live
Interview
Zeitaufwand
datenflanke - Dokumentation

KAPITEL 3

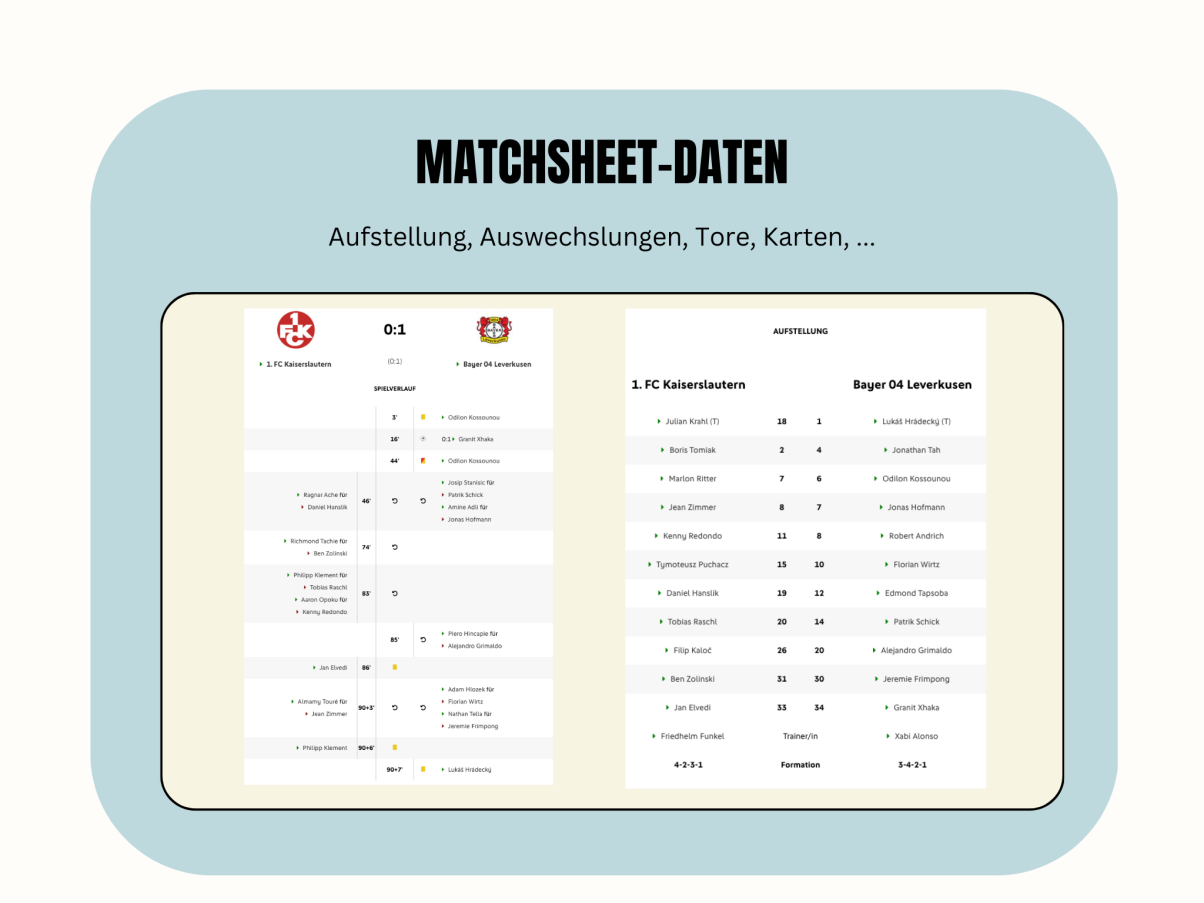
Daten im Fußball

Da nun geklärt ist, dass Datenscouting ein unverzichtbares Element im Scoutingprozess ist, stellt sich die Frage, was für Daten es überhaupt gibt und welche davon für die Bewertung von Spielern relevant sind.

- **Matchsheet-Daten:** Grundlegende Informationen zu Spielen und Spielern.
- **Event-Daten:** Details zu spezifischen Ereignissen im Spiel, wie Tore, Pässe usw.
- **Tracking-Daten:** Positionsdaten der Spieler und des Balls während des Spiels.

3.1 Matchsheet-Daten

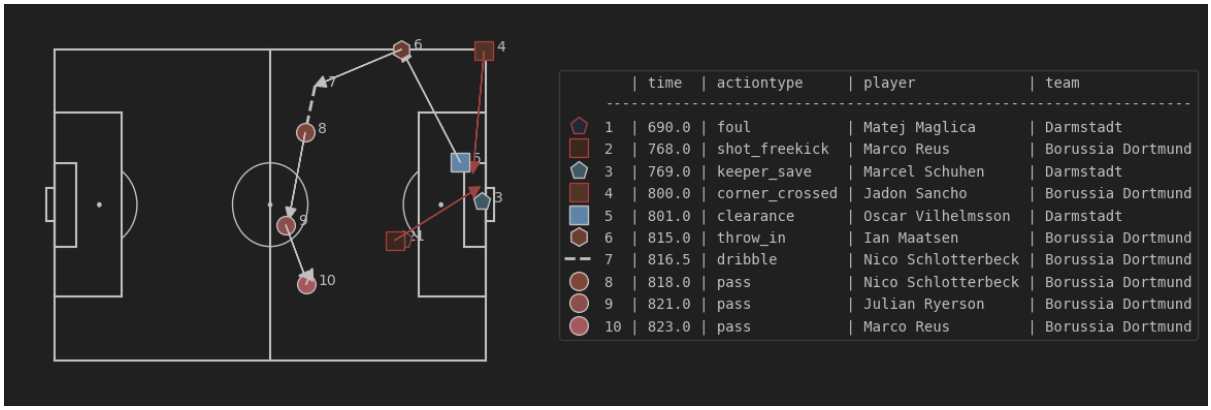
Unter **Matchsheet-Daten** versteht man in der Regel die grundlegenden Informationen zu einem Spiel, wie z.B. die Aufstellung, die Torschützen, die Karten, die Auswechslungen usw. Diese Daten sind relativ einfach zu erfassen und sind in der Regel für jedes Spiel verfügbar (transfermarkt.de, kicker.de, etc.). Es gibt unzählige Studien, die gezeigt haben, dass Matchsheet-Daten alleine nicht ausreichen, um die Leistung eines Spielers zu bewerten. Matchsheet-Daten sind wichtig, um einen Überblick über das Spiel zu bekommen, aber sie sind nicht ausreichend, um die Leistung eines Spielers zu bewerten.



3.2 Event-Daten

Event-Daten sind detaillierte Informationen zu spezifischen Ereignissen im Spiel, wie z.B. Tore, Pässe, Schüsse, Zweikämpfe usw. Diese Daten sind viel detaillierter als Matchsheet-Daten und ermöglichen eine genauere Analyse der Leistung eines Spielers. Event-Daten sind in der Regel nicht öffentlich verfügbar und müssen entweder manuell erfasst oder von spezialisierten Anbietern gekauft werden. Die bekanntesten Anbieter von Event-Daten sind Opta, StatsBomb, Wyscout, InStat, etc. Diese Anbieter erfassen die Daten in der Regel zunächst automatisiert und lassen sie dann von menschlichen Beobachtern überprüfen und korrigieren.

In der Spiel- und Gegneranalyse ist es ebenfalls üblich, die eigenen Spiele und die des kommenden Gegners sogar selbst nach den eigenen Kriterien zu taggen, um die Daten zu standardisieren und die Qualität der Analysen zu erhöhen. Event-Daten bieten den Vorteil, dass sie für eine Vielzahl an Wettbewerben verfügbar sind. Die Datengröße liegt mit circa 1600 Events (Spielaktionen) pro Spiel bei 1,3 MB pro Spiel (im JSON-Format) und knapp 400 MB pro Bundesligasaison. Der Nachteil von Event-Daten ist, dass sie nur die Aktionen erfassen, die mit dem Ball stattfinden. Laufwege und Stellungsspiel ohne Ball werden nicht erfasst.



	time	actiontype	player	team
1	690.0	foul	Matej Maglica	Darmstadt
2	768.0	shot_freekick	Marco Reus	Borussia Dortmund
3	769.0	keeper_save	Marcel Schuhen	Darmstadt
4	800.0	corner_crossed	Jadon Sancho	Borussia Dortmund
5	801.0	clearance	Oscar Vilhelmsson	Darmstadt
6	815.0	throw_in	Ian Maatsen	Borussia Dortmund
7	816.5	dribble	Nico Schlotterbeck	Borussia Dortmund
8	818.0	pass	Nico Schlotterbeck	Borussia Dortmund
9	821.0	pass	Julian Ryerson	Borussia Dortmund
10	823.0	pass	Marco Reus	Borussia Dortmund

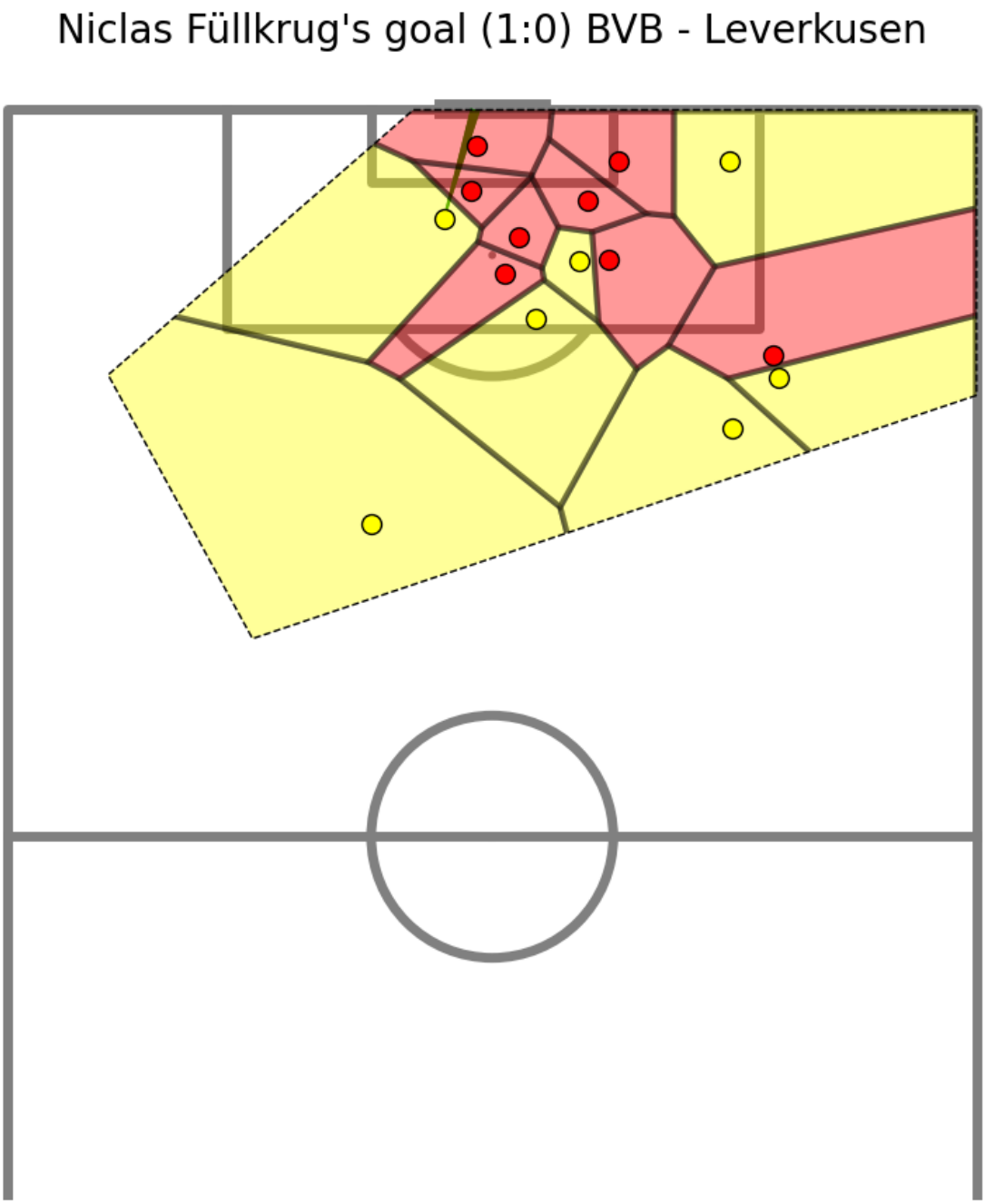
3.3 Tracking-Daten

Tracking-Daten sind Positionsdaten der Spieler und des Balls während des Spiels. Diese Daten werden in der Regel von spezialisierten Anbietern wie StatsBomb oder SecondSpectrum erfasst. Die Daten werden entweder durch Kameras oder GPS-Tracker bestimmt und bieten eine detaillierte Analyse der Bewegungen der Spieler und des Balls während des Spiels. Hiermit werden alle 22 Spieler und der Ball 25 Mal pro Sekunde getrackt.

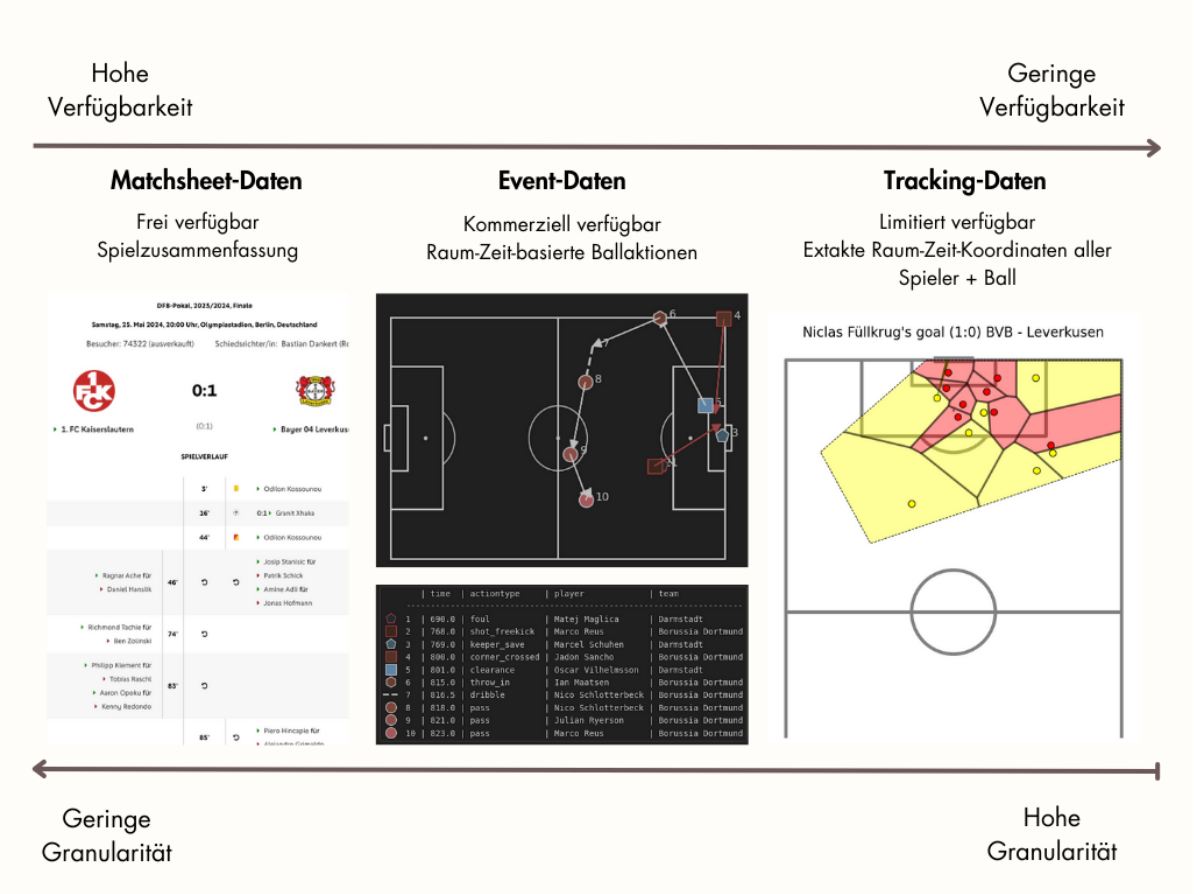
Es lassen sich eine Vielzahl von Metriken ableiten, wie z.B. Laufwege, Geschwindigkeit, Beschleunigung, Raumvorbereitung, etc. Vor allem Metriken, die ohne Ball stattfinden, sind hierbei interessant, da sie in den Event-Daten nicht erfasst werden. Die Datengröße ergibt sich aus 23 Spalten (22 Spieler + Ball) * 25 Frames * 60 Sekunden

- 90 Minuten = über 3 Millionen Datenpunkte pro Spiel. Das ergibt über 60 MB pro Spiel siehe Metrica Beispieldaten und somit um die 18 GB pro Bundesligasaison.

Der Nachteil von Tracking-Daten ist, dass sie sehr groß sind und eine spezielle Infrastruktur benötigen, um sie zu verarbeiten. Die Daten sind auch nicht ansatzweise für alle Wettbewerbe verfügbar und sind in der Regel sehr teuer.


Niclas Füllkrug's goal (1:0) BVB - Leverkusen

3.4 Daten-Vergleich



Hieraus ergibt sich schnell, dass aufgrund von Verfügbarkeit und Kosten, die meisten Analysen auf Event-Daten basieren. Tracking-Daten sind jedoch die Zukunft der Spielanalyse und werden in den nächsten Jahren immer wichtiger werden. Aus meiner Sicht haben Tracking-Daten das Potenzial, den Einfluss auf eine Bewertung maßgeblich zu verändern, wohingegen das Potenzial für Wissensvorsprung bei Event-Daten limitiert ist. Zum Beispiel könnten Spieler besser bewertet werden, die intelligente Laufwege ohne Ball machen und dadurch Räume schaffen, was in den Event-Daten nicht erfasst wird.

3.5 Erhebung von Daten

Aktuelle Tracking-Daten sind also sehr teuer und nicht für alle Wettbewerbe verfügbar. Matchsheet-Daten sind relativ einfach zu bekommen, aber nicht ausreichend für eine detaillierte Analyse. Wie werden Event-Daten erhoben?

- **Manuell:** Daten werden durch menschliche Beobachter erfasst.
- **KI-basiert:** Automatisierte Datenerfassung durch Künstliche Intelligenz.
- **Live-Daten:** Daten, die in Echtzeit erfasst werden, z.B. durch Kameras oder GPS-Tracker. Meist werden diese Daten durch spezielle Algorithmen in Eventdaten umgewandelt (z.B. Opta für die Bundesliga).
- **Post-Match-Daten:** Daten, die nach dem Spiel nochmals überprüft und korrigiert werden.

3.6 Quellen von Event-Daten

- **Kaufen:** Erwerb von Daten von spezialisierten Anbietern (Opta, StatsBomb, Wyscout, InStat, etc.).
- **Scrapen:** Automatisiertes Extrahieren von Daten aus öffentlich zugänglichen Quellen.
- **Selbst erheben:** Manuelle Erfassung von Daten Tool zur Unterstützung:.
- **Freie Daten:** Daten, die frei verfügbar sind, z.B. auf Kaggle, GitHub, etc.

Freie Daten sind oft nicht so umfangreich wie die Daten von spezialisierten Anbietern, aber sie sind ein guter Einstieg, um sich mit den Daten vertraut zu machen und erste Analysen durchzuführen. Gleichzeitig fand ich es persönlich aber immer etwas langweilig, mit alten Daten zu arbeiten, da mich die aktuellen Spiele und Spieler viel mehr interessieren. Daher habe ich mich entschieden, die Daten zu scrapen. Wie ich das gemacht habe, werde ich in eines der nächsten Kapitel erläutern.

Hier ein paar Links zu freien Event- und Tracking-Daten:

- Statsbomb Free Data: https://github.com/statsbomb
- Metrica Free Data: https://github.com/metrica-sports/sample-data
- Wyscout Free Data: https://figshare.com/collections/Soccer_match_event_dataset/4415000/5

KAPITEL 4

Kognitive Verzerrungen - Biases im Fußballscouting

Kognitive Verzerrungen (cognitive **bias**) sind systematische Fehler in der Wahrnehmung, Erinnerung oder Interpretation von Informationen. Sie können zu Fehlern in der Entscheidungsfindung führen, da sie die Art und Weise beeinflussen, wie wir Informationen verarbeiten. Kognitive Verzerrungen können aufgrund von Heuristiken, sozialen Einflüssen oder anderen Faktoren auftreten. Es ist wichtig, sich ihrer Existenz bewusst zu sein und Strategien zu entwickeln, um sie zu erkennen und zu vermeiden. Im Fußball gibt es einige kognitive Verzerrungen, die bei der Bewertung von Spielern eine Rolle spielen können. Ich möchte hier einige der wichtigsten Verzerrungen vorstellen und erläutern, wie sie sich auf die Bewertung von Spielern auswirken können. Für eine umfangreiche Auseinandersetzung mit *cognitive biases* empfehle ich das Buch Thinking, Fast and Slow von Daniel Kahneman.

4.1 Clustering Illusion

Die Clustering-Illusion bezieht sich auf die Tendenz von Menschen, Muster oder Cluster in Daten zu sehen, wo eigentlich keine signifikanten Muster existieren. Diese kognitive Verzerrung führt dazu, dass zufällige Ereignisse fälschlicherweise als bedeutungsvoll interpretiert werden, was insbesondere in Bereichen mit großen Datenmengen wie dem Finanzmarkt oder dem Sport zu Fehleinschätzungen führen kann.

Beispiel Clustering Illusion Im Fußballscouting kann die Clustering-Illusion dazu führen, dass ein Scout in der Leistung eines Spielers über einen kurzen Zeitraum fälschlicherweise ein Muster sieht und darauf basierend langfristige Vorhersagen trifft.

Beispiel: Ein Scout beobachtet einen Torwart, Spieler D, über fünf Spiele hinweg. In diesen Spielen hält Spieler D in drei Spielen hintereinander jeweils einen Elfmeter. Der Scout könnte nun aufgrund dieser kurzen Sequenz zu dem Schluss kommen, dass Spieler D eine außergewöhnliche Fähigkeit besitzt, Elfmeter zu halten, und ihn deshalb als Spezialisten auf diesem Gebiet einstufen.

Diese Einschätzung ignoriert jedoch die Möglichkeit, dass es sich bei den gehaltenen Elfmetern um ein zufälliges Ereignis handeln könnte und nicht unbedingt ein verlässlicher Indikator für die langfristigen Fähigkeiten des Torwarts ist. Ohne die Berücksichtigung weiterer Daten und Leistungen über einen längeren Zeitraum hinweg könnte die Entscheidung, Spieler D aufgrund dieser kurzen Sequenz von Ereignissen zu rekrutieren, auf der Clustering-Illusion basieren und nicht auf einer soliden Analyse seiner Gesamtleistung.

4.2 Confirmation Bias

Der Confirmation Bias, auch Bestätigungsfehler genannt, ist eine kognitive Verzerrung, bei der Menschen dazu neigen, Informationen auf eine Weise zu suchen, zu interpretieren, zu favorisieren und zu erinnern, die ihre vorherigen Überzeugungen oder Hypothesen bestätigt. Das bedeutet, dass sie eher Beweise wahrnehmen und akzeptieren, die ihre eigenen Ansichten unterstützen, während sie gleichzeitig gegenteilige Beweise ignorieren oder herunterspielen.

Beispiel Confirmation Bias Ein Scout eines Fußballvereins erhält den Auftrag, einen vielversprechenden jungen Mittelfeldspieler, nennen wir ihn Spieler B, zu beobachten, der für seine exzellenten Pässe und seine Spielintelligenz bekannt ist. Der Scout hat bereits im Vorfeld positive Berichte über Spieler B gelesen und geht daher mit der festen Überzeugung in die Spielbeobachtung, dass Spieler B ein außergewöhnliches Talent ist.

Während der Beobachtung konzentriert sich der Scout vor allem auf die Momente, in denen Spieler B seine Stärken zeigt: präzise Pässe, kluge Spielzüge und gute Positionierung. Jedes Mal, wenn Spieler B eine dieser Aktionen erfolgreich ausführt, wertet der Scout dies als Bestätigung seiner anfänglichen Einschätzung.

Gleichzeitig neigt der Scout dazu, Aspekte von Spieler B's Leistung, die nicht den Erwartungen entsprechen, weniger Gewicht zu geben oder zu übersehen. Wenn Spieler B beispielsweise in der Defensive Schwächen zeigt, schlechte Entscheidungen trifft oder in wichtigen Momenten des Spiels unsichtbar wird, könnte der Scout diese Momente als Ausnahmen betrachten oder Gründe außerhalb der Kontrolle von Spieler B dafür verantwortlich machen (z.B. schlechte Kommunikation mit den Teamkollegen oder eine unklare taktische Anweisung vom Trainer).

Durch diesen Confirmation Bias könnte der Scout am Ende einen Bericht erstellen, der die Fähigkeiten und das Potenzial von Spieler B überbewertet, während die Schwächen und Entwicklungsbereiche unterbewertet oder gar nicht erwähnt werden. Dies kann dazu führen, dass der Fußballverein eine Entscheidung auf der Basis einer verzerrten Einschätzung trifft, was langfristig negative Auswirkungen auf die Mannschaftszusammenstellung und die finanziellen Investitionen haben könnte.

4.3 Hindsight Bias

Der Hindsight Bias, auch bekannt als Rückschaufehler, beschreibt die Neigung von Menschen, nachdem ein Ereignis eingetreten ist, zu glauben, dass sie das Ergebnis vorhergesehen oder erwartet hatten. Dieser kognitive Bias kann dazu führen, dass Menschen ihre Fähigkeit, zukünftige Ereignisse vorherzusagen, überschätzen, da sie sich im Nachhinein an die Fakten so erinnern, als wären sie vorhersehbar gewesen.

Im Kontext des Fußballscoutings kann der Hindsight Bias die Bewertung von Entscheidungen im Nachhinein verzerren.

Beispiel Hindsight Bias Ein Fußballverein entscheidet sich gegen die Verpflichtung eines jungen Spielers, Spieler E, basierend auf den Bewertungen und Empfehlungen der Scouts. Einige Jahre später entwickelt sich Spieler E zu einem der führenden Spieler in seiner Position auf internationaler Ebene. Mitglieder des Vereins, einschließlich der Scouts und der Führungsebene, könnten nun rückblickend behaupten, dass sie damals bereits das Potenzial in Spieler E gesehen hatten, aber aus verschiedenen, nun als trivial angesehenen Gründen gegen seine Verpflichtung entschieden haben.

Diese retrospektive Überzeugung, dass sie das Potenzial von Spieler E „eigentlich" erkannt hatten, ignoriert die Tatsache, dass zum Zeitpunkt der ursprünglichen Entscheidung die Informationen und die Bewertung des Spielers zu einem anderen Schluss führten. Der Hindsight Bias lässt die Entscheidungsträger glauben, dass das Ergebnis (Spieler E wird ein Topspieler) vorhersehbar war, obwohl sie zum Zeitpunkt der Entscheidung nicht über die Informationen verfügten, die sie im Nachhinein als offensichtlich betrachten.

4.4 Outcome Bias

Der Outcome Bias beschreibt die Tendenz, eine Entscheidung basierend auf dem Ergebnis der Entscheidung zu bewerten, anstatt auf die Qualität der Entscheidungsfindung (zum Zeitpunkt der Entscheidung selbst). Dieser kognitive Bias kann dazu führen, dass gute Entscheidungen, die zufällig zu schlechten Ergebnissen führen, negativ bewertet werden und umgekehrt: schlechte Entscheidungen, die zufällig zu guten Ergebnissen führen, positiv bewertet werden.

Im Fußballscouting kann der Outcome Bias die Bewertung von Transfers und Talentförderung beeinflussen.

Beispiel Outcome Bias Ein Fußballverein entscheidet sich, einen relativ unbekannten Spieler, Spieler F, zu verpflichten, basierend auf einer gründlichen Analyse und positiven Bewertungen der Scouts. In seiner ersten Saison beim Verein erzielt Spieler F überraschend viele Tore und wird schnell zu einem Schlüsselspieler. Im Nachhinein wird die Entscheidung, ihn zu verpflichten, als hervorragend bewertet.

Diese positive Bewertung könnte jedoch stark vom Outcome Bias beeinflusst sein. Wenn Spieler F in seiner ersten Saison verletzungsbedingt kaum gespielt hätte oder sich nicht sofort an den neuen Verein angepasst hätte, könnten dieselben Entscheidungsträger für genau dieselbe Entscheidung kritisiert werden. In beiden Szenarien wäre die Qualität der Entscheidungsfindung dieselbe, aber das Ergebnis (und damit die Bewertung der Entscheidung) könnte stark variieren.

Der Outcome Bias kann somit zu einer verzerrten Wahrnehmung der Entscheidungsqualität führen, indem er den Fokus von der soliden Analyse und den zum Entscheidungszeitpunkt verfügbaren Informationen wegnimmt und stattdessen das Ergebnis überbewertet. Dies kann längerfristig zu einer Kultur führen, in der Glück oder Zufall statt gründlicher Analyse und Prozessqualität belohnt werden.

4.5 Self-Serving Bias

Der Self-serving Bias (Selbstwertdienliche Verzerrung) ist eine kognitive Verzerrung, bei der Menschen dazu neigen, Erfolge und positive Ereignisse ihren eigenen Fähigkeiten und Anstrengungen zuzuschreiben, während Misserfolge und negative Ereignisse externen Faktoren, wie Pech oder der Einwirkung anderer, zugeschrieben werden. Diese Verzerrung hilft Individuen, ihr Selbstwertgefühl zu schützen und zu erhöhen, kann aber auch zu einer verzerrten Wahrnehmung der eigenen Verantwortung und Leistung führen.

Beispiel Self-Serving Bias Ein Scout, nennen wir ihn Scout A, hat die Aufgabe, junge Talente für einen Fußballverein zu identifizieren. Nach umfangreicher Recherche und Beobachtung empfiehlt Scout A die Verpflichtung eines jungen Spielers, Spieler X, von dem er überzeugt ist, dass er das Potenzial hat, ein Star zu werden. In den ersten Monaten nach der Verpflichtung zeigt Spieler X herausragende Leistungen, erzielt wichtige Tore und wird schnell zu einem Schlüsselspieler für das Team.

In Interviews und internen Meetings betont Scout A wiederholt, wie seine Expertise, sein scharfes Auge für Talente und seine gründliche Analyse entscheidend für die Entdeckung und Verpflichtung von Spieler X waren. Er schreibt den Erfolg der Verpflichtung hauptsächlich seinen eigenen Fähigkeiten und seinem Urteilsvermögen zu.

Einige Zeit später empfiehlt Scout A die Verpflichtung eines weiteren Spielers, Spieler Y, der jedoch nicht die erwarteten Leistungen erbringt und Schwierigkeiten hat, sich im Team zu etablieren. In diesem Fall neigt Scout A dazu, externe Faktoren für das Scheitern der Verpflichtung verantwortlich zu machen. Er könnte argumentieren, dass Spieler Y nicht genügend Zeit bekommen hat, sich anzupassen, dass das Trainingssystem des Teams nicht zu seinen Fähigkeiten passt oder dass Verletzungen und Pech seine Entwicklung behindert haben.

Durch den Self-serving Bias schützt Scout A sein Selbstbild und seine professionelle Reputation, indem er Erfolge sich selbst zuschreibt, während er für Misserfolge externe Umstände verantwortlich macht. Obwohl dies menschlich und verständlich ist, kann es die Fähigkeit des Scouts und des Vereins beeinträchtigen, aus Fehlern zu lernen und zukünftige Scouting-Entscheidungen zu verbessern, da eine ehrliche Selbstreflexion und Bewertung der eigenen Entscheidungsprozesse möglicherweise vernachlässigt wird.

4.6 Story Bias

Der Story Bias bezieht sich auf die Tendenz von Menschen, Informationen besser zu verarbeiten und zu erinnern, wenn diese in Form einer erzählenden Geschichte präsentiert werden, im Gegensatz zu abstrakten oder zusammenhanglosen Fakten. Im Kontext des Fußballscoutings kann dieser Bias dazu führen, dass Scouts oder Entscheidungsträger Spieler aufgrund der überzeugenden Geschichten, die mit ihnen verbunden sind, anders bewerten, anstatt sich ausschließlich auf objektive Leistungsdaten zu stützen. Hier ein Beispiel:

Beispiel Story Bias Ein Scout wird beauftragt, einen Spieler, nennen wir ihn Spieler C, zu beobachten, der aus einem kleinen, wenig bekannten Verein stammt. Vor dem Spiel erfährt der Scout die inspirierende Geschichte von Spieler C: Er hat seine gesamte Jugendkarriere in unterklassigen Ligen verbracht, ohne Zugang zu den Trainingsressourcen und der Förderung, die in größeren Akademien üblich sind. Trotz dieser Hindernisse hat er sich durch außergewöhnliches Talent und harte Arbeit hervorgetan und schließlich die Aufmerksamkeit höherklassiger Vereine auf sich gezogen.

Während der Beobachtung des Spiels ist der Scout von der Geschichte von Spieler C beeindruckt und neigt daher dazu, seine Leistung auf dem Feld durch die Linse dieser erzählenden Geschichte zu sehen. Jedes Mal, wenn Spieler C eine bemerkenswerte Aktion zeigt – sei es ein geschickter Pass, eine erfolgreiche Dribbelaktion oder ein Tor –, sieht der Scout dies als weiteren Beweis für die außergewöhnliche Natur von Spieler C's Talent und seinen unerschütterlichen Willen, sich gegen alle Widrigkeiten durchzusetzen.

KAPITEL 5

Überblick der datenbasierten Bewertungsmethoden

Hier ein kurzer Überblick über die verschiedenen Methoden zur Bewertung der Spielerleistung im Fußball. Diese Übersicht ist nicht vollständig, zeigt aber meiner Meinung nach die beiden wesentlichen Bewertungsmodelle, die ohne Positionsdaten auskommen.

5.1 Top-down Bewertung

Top-down-Bewertungen, insbesondere die Plus-Minus-Modelle, messen den Einfluss eines Spielers, indem sie die Leistung des Teams mit und ohne diesen Spieler vergleichen. Diese Methode wurde ursprünglich in Sportarten wie Basketball und Eishockey eingesetzt und findet nun auch im Fußball Anwendung. Das Prinzip ist simpel: Wenn ein Spieler auf dem Feld steht, wie verändert sich dann die Tordifferenz seiner Mannschaft im Vergleich zu den Minuten, in denen er nicht spielt? Diesen Ansatz kann man noch weiter verfeinern und verbessern, aber der Ansatz bleibt derselbe.

Eine Variante dieses Modells bezieht zum Beispiel auch erwartete Tore (xG) und erwartete Punkte (xP) mit ein. Dabei wird nicht nur berücksichtigt, ob ein Tor erzielt wurde, sondern auch, wie wahrscheinlich es war, dass ein bestimmter Schuss zu einem Tor führt. Diese erweiterten Modelle nutzen verschiedene Schusskategorien und Faktoren wie die Fähigkeiten des Torhüters und die Spielsituation.

Top-Down-Modelle sind nützlich, da sie auf wenigen Informationen basieren und auch auf Ligen anwendbar sind, in denen keine detaillierten Event- oder Trackingdaten erhoben werden (Niedrigere Spielklasse / Amateurbereich / Jugendbereich). Allerdings haben sie einige Einschränkungen:

- Sie können keine Rückschlüsse auf einzelne Aktionen eines Spielers ziehen. Wir wissen also nicht, warum Spieler XY ein gutes oder schlechtes Rating hat.
- Der Heimvorteil wird oft als konstanter Faktor berücksichtigt, was die Ergebnisse verfälschen kann.
- Langzeitdatensätze berücksichtigen nicht immer Veränderungen in der individuellen Spielerleistung, wie Verletzungen oder Anpassungsschwierigkeiten nach Transfers.
- Die Bewertung von Spielern, die nur kurzzeitig auf dem Feld stehen, kann ungenau sein, da die Stichprobe zu klein ist.
- Da die Spieler abhängig von der Leistung ihrer Teamkollegen bewertet werden, kann es zu Verzerrungen kommen, wenn ein Spieler in einem starken Team spielt, aber selbst nur durchschnittliche Leistungen bringt.

Einen Überblick über Plus-Minus Modelle kannst du hier finden: A comprehensive review of plus-minus ratings for evaluating individual players in team sports

Vor allem Lars Magnus Hvattum hat sich in den letzten Jahren intensiv mit dem Thema Plus-Minus-Modelle im Fußball beschäftigt und einige interessante Ansätze entwickelt. Weitere Informationen findest du auf seinem YouTube Kanal: Football Player Ratings

5.2 Bottom-up Bewertung

Die Bewertung der Spielerleistung im Fußball hat sich im Laufe der Zeit erheblich weiterentwickelt. Verschiedene Modelle und Ansätze wurden entwickelt, um die Effektivität einzelner Aktionen und Spieler zu messen. Ein zentraler Aspekt dieser Bewertung ist die sogenannte Bottom-up-Bewertung, die darauf abzielt, den Beitrag jeder einzelnen Aktion zur Gesamtleistung eines Teams zu quantifizieren. Es gibt viele verschiedene Studien, die versuchen, die Leistungsparameter für erfolgreiches Fußballspielen zu identifizieren. Beispielhaft seien hier McHale, Scarf und Folker (2012) genannt.

5.3 Bewertungsindex

McHale, Scarf und Folker (2012) entwickelten einen umfassenden Bewertungsindex, der verschiedene Subindizes umfasst. Anders als frühere Plus-Minus-Modelle berücksichtigt dieser Ansatz spezifische Spielstatistiken. Zu den einbezogenen Parametern zählen Flanken, Dribblings, Pässe, abgefangene Bälle, gelbe und rote Karten des Gegners, das Verhältnis gewonnener Zweikämpfe und Befreiungsschläge. Die Korrelationen dieser Parameter mit Torschüssen wurden berechnet, um ihre Gewichtung zu bestimmen. Zusammen mit anderen Subindizes wie Spielbeteiligung, erzielten Toren und Vorlagen sowie Spielen ohne Gegentor wurden die Spieler der Premier League 2008-2009 bewertet. Modernere Ansätze gehen jedoch weiter und berücksichtigen auch die Qualität der Aktionen, nicht nur die Quantität.

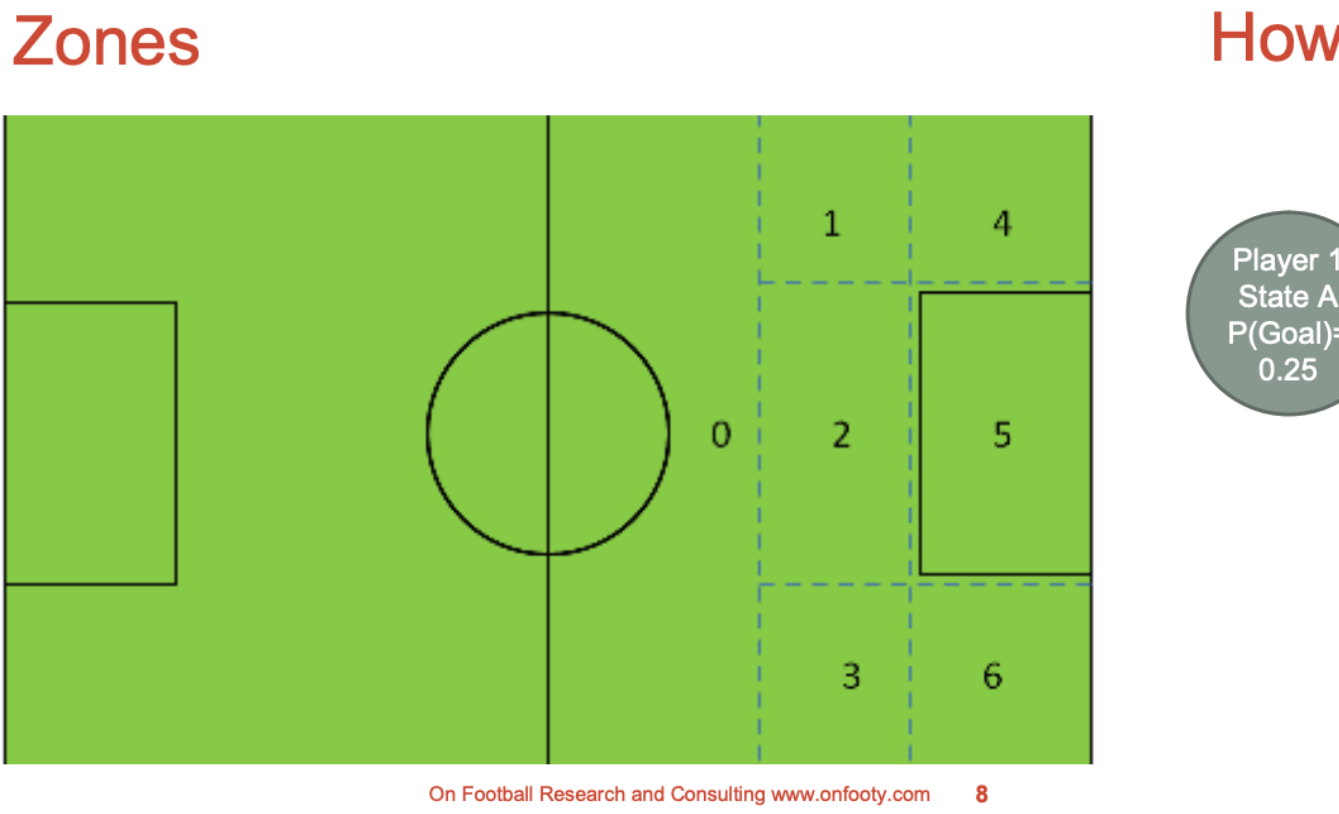
5.4 Der Einfluss einer Aktion

Der zentrale Gedanke bei der Bewertung von Aktionen ist zu verstehen, wie eine Aktion die Wahrscheinlichkeit erhöht oder verringert, dass ein Team ein Tor erzielt. Ein bekanntes Maß hierfür ist „Expected Goals“ (xG), das die Wahrscheinlichkeit eines Tors bei einem Schuss bewertet. Dieses Prinzip lässt sich auf andere Aktionen übertragen, wobei die Herausforderung darin besteht, die komplexen Zusammenhänge und den Kontext jeder Aktion zu berücksichtigen.

Für andere Aktionen, wie Pässe oder Dribblings, ist die Bewertung schwieriger. Ein Pass, der einen Stürmer in eine gute Position bringt, ist wertvoller als ein Pass in eine gut verteidigte Zone, auch wenn beide Pässe ähnliche Start- und Endkoordinaten haben können. Um diese Unterschiede zu bewerten, wurden verschiedene Modelle und Methoden entwickelt, die oft maschinelles Lernen einsetzen.

5.5 Markov-Ketten

Sarah Rudd (2011) führte einen innovativen Ansatz ein, indem sie Markov-Ketten nutzte, um die Wahrscheinlichkeit eines Tores mit bestimmten Zuständen auf dem Spielfeld zu verknüpfen (Hier ist ihr Vortrag). Sie unterteilte das Spielfeld in sechs Zonen, wobei der Ballbesitz in einer Zone als Zustand betrachtet wurde. Anhand von Eventdaten aus 123 Premier League-Spielen berechnete sie die Wahrscheinlichkeiten für Zustandswechsel, wobei Standardsituationen in zusätzliche Zustände unterteilt wurden. Das Modell berücksichtigt, dass die Wahrscheinlichkeit für ein Tor von der aktuellen Spielfeldposition abhängt und nicht von vergangenen Zuständen. Diese Methode ermöglicht es, jede Aktion eines Spielers, die eine Zustandsänderung herbeiführt, zu bewerten.



5.6 Aktionsbasiert: xGChain und xGBuildup

Lawrence (2018) berechnet xGChain und xGBuildup, indem er den Erfolg einer Sequenz aufsummiert und auf die beteiligten Spieler verteilt. xGChain (xGC) ist eine Metrik zur Bewertung von Spieleraktionen im Fußball. Es werden alle xG-Werte (Expected Goals) einer Ballbesitzkette, die mit einem Schuss enden, aufsummiert. Der Wert jeder Angriffskette wird auf alle beteiligten Spieler verteilt. Eine Ballbesitzkette ist eine Sequenz von Ereignissen, bei der das gleiche Team durchgehend die Kontrolle über den Ball hat.

Fußball ist ein dynamisches Spiel, und die Torgefahr ergibt sich unter anderem daraus, wie der Ball bewegt wird und nicht nur daraus, wo er sich befindet. (Ein lang geschlagener Ball, der sich in der Nähe des gegnerischen Tores befindet, ist noch lange keine Torgefahr.) Ein Pass hat nicht nur wegen seines Endpunkts einen bestimmten Wert, sondern auch, weil er die Verteidigung verschiebt. Flügelwechsel, die den Ball nicht nach vorne bringen, sollten trotzdem positiv bewertet werden. Ebenso sollten Rückpässe, die ein Schritt zur späteren Vorwärtsbewegung sein können, positive Werte erhalten. Durch die Betrachtung der Aktionen innerhalb einer Kette können wir diesen Kontext erfassen.

xGBuildup ist ähnlich wie xGChain, jedoch ohne die letzten beiden Aktionen (Schuss und Assist) einer Ballbesitzkette. So erhält ein Spieler für den Aufbau eines Angriffs eine Bewertung, auch wenn er nicht direkt am Schuss beteiligt war.

5.7 Positionsbasiert: xT - expected Threat

Die Basis für die Berechnung von xT (von Karun Singh) ist eine Übergangsmatrix, die auf Statistiken zu Schüssen, Toren und Aktionen (Pässe und Dribblings) über einen bestimmten historischen Zeitraum berechnet wird. Diese Matrix hilft, die Wahrscheinlichkeit eines Tores nach einer bestimmten Anzahl von Folgeaktionen zu schätzen. Jede Zelle der Matrix repräsentiert den xT-Wert für eine spezifische Zone auf dem Spielfeld, also die Wahrscheinlichkeit, ein Tor zu erzielen, wenn sich der Ball nach mehreren Aktionen in dieser Zone befindet.

Um den xT-Wert für einen Pass oder eine Bewegung des Balls von Punkt A nach Punkt B zu berechnen, zieht man den xT-Wert des Startbereichs vom xT-Wert des Zielbereichs ab. Beispiel: Ein erfolgreicher Pass vom Eckbereich in den Strafraum hat einen xT-Wert von 0.11–0.04 = 0.07. Der passgebende Spieler erhält somit 0.07 xT für diese Aktion. Die Chance ein Tor innerhalb der nächsten 5 Aktionen zu erzielen, hat sich somit um 7% erhöht.

Ein Nachteil gegenüber xGChain und xGBuildup ist, dass Rückpässe und Flügelwechsel nicht so gut oder negativ bewertet werden, da sie den Ball nicht näher ans gegnerische Tor bringen. Der Vorteil ist, dass jede Aktion bewertet wird und nicht nur Aktionen, die zu einem Schuss

KAPITEL 6

VAEP-Modell: Bewertung von Spieleraktionen

6.1 Einführung

Das VAEP-Modell (Valuing Actions by Estimating Probabilities) ist ein fortschrittliches Bewertungssystem für Fußballspieler, das entwickelt wurde, um den Wert einzelner Aktionen auf dem Spielfeld zu quantifizieren. Es bewertet sowohl offensive als auch defensive Beiträge eines Spielers und nutzt dabei maschinelles Lernen, um die Auswirkungen dieser Aktionen auf das Spiel zu analysieren.

6.2 Grundprinzipien des VAEP-Modells

Das VAEP-Modell basiert auf der Idee, dass jede Aktion auf dem Spielfeld die Wahrscheinlichkeit eines Tores oder eines Gegentores beeinflusst. Um diese Veränderungen zu messen, betrachtet das Modell die Zustände vor und nach einer Aktion und schätzt die Wahrscheinlichkeiten für Tore und Gegentore. Die Differenz dieser Wahrscheinlichkeiten gibt den Wert der Aktion an.

6.3 Was ist ein Zustand?

Ein Zustand repräsentiert die aktuelle Spielsituation mit all seinen Merkmalen. Jede Spielsituation hat dabei eine Tor- und Gegentorwahrscheinlichkeit. Die Bewertung entsteht dabei aus der Veränderung des Zustandes (und der damit verbundenen Torwahrscheinlichkeit) vor und nach einer Aktion.

6.4 Merkmale der Aktionen

Folgende Merkmale könnten wir für eine Aktion erfassen und in die Bewertung einfließen lassen:

- Art der Aktion (z.B. Pass, Schuss, Dribbling)
- Erfolgreich oder nicht erfolgreich
- Körperteil, mit dem die Aktion ausgeführt wurde
- Position auf dem Spielfeld (Start- und Endpunkt)
- Zeitpunkt der Aktion (Spielminute)
- Kontextuelle Merkmale wie die Anzahl der Tore und die Tordifferenz

Als Merkmal kann alles herangezogen werden, was an Daten verfügbar oder daraus zu berechnen ist.

6.5 Mehr Kontext

Da Fußball nicht nur aus statischen Situationen besteht, sondern sehr dynamisch ist, werden noch weitere Aktionen (meist 3) vor der zu bewertenden Aktion mit einbezogen. So kann der Kontext besser erfasst werden. Weiter wird dann geprüft, ob in den nächsten 10 Aktionen ein Tor fällt. Daraus ergibt sich der Lerneffekt des Modells. Man merkt schnell, dass wir deshalb eine Vielzahl von Aktionen brauchen, aus denen wir lernen können, da die Kombinationsmöglichkeiten der einzelnen Merkmale sehr groß sind. Mithilfe von maschinellem Lernen werden dann die Wahrscheinlichkeiten für Tore und Gegentore vor und nach jeder Aktion berechnet.

Schon mal vorab: Im maschinellen Lernen spricht man dabei von einem *Klassifikationsproblem*, da wir eine binäre Entscheidung treffen müssen: Tor oder kein Tor. Das Ergebnis der Aktion (Tor / kein Tor) ist die Zielvariable, die wir vorhersagen wollen und wird auch als *Label* bezeichnet. Die Merkmale, die wir für die Vorhersage verwenden, werden als *Features* bezeichnet.

6.6 Bewertung der Aktionen

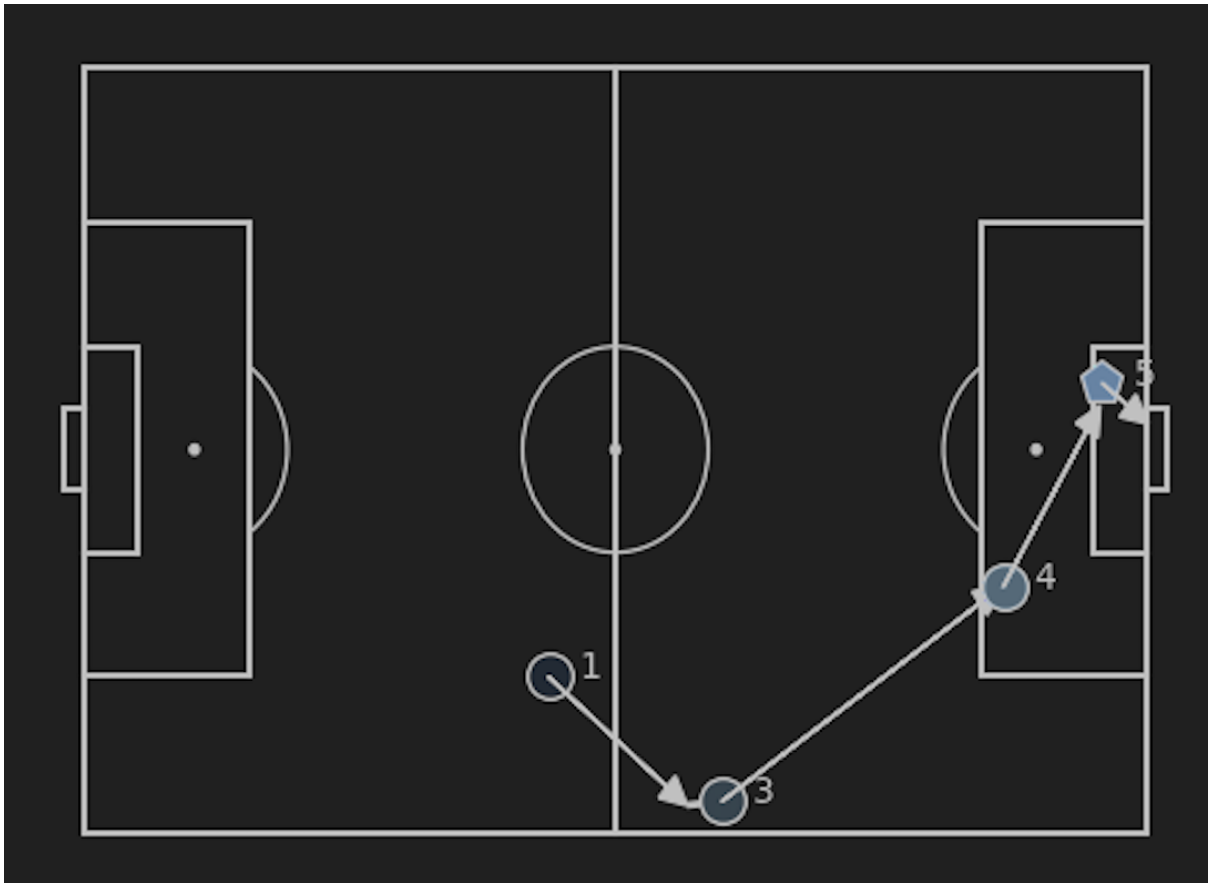
Der Wert einer Aktion wird als die Differenz der Wahrscheinlichkeiten berechnet:

- Offensiver Wert: Unterschied in der Wahrscheinlichkeit, ein Tor zu erzielen
- Defensiver Wert: Unterschied in der Wahrscheinlichkeit, ein Gegentor zu verhindern

Hier der Pass von Frimpong als Plot dargestellt (Spielrichtung im Plot für das Heimteam von links nach rechts):

Torwahrscheinlichkeit vor dem Pass: 0.0143 Torwahrscheinlichkeit nach dem Pass: 0.0817 Damit ergibt sich für den Wert der Aktion ein Wert von 0.0674, was für einen Pass >= 30m ein sehr guter Wert ist.

	actiontype	player	team	offensive_value	goal probability
1	pass	Jonas Hofmann	Leverkusen	0.004	0.0138
2	dribble	Jeremie Frimpong	Leverkusen	0.0	0.0143
3	pass	Jeremie Frimpong	Leverkusen	0.067	0.0817
4	pass	Jonas Hofmann	Leverkusen	0.298	0.3796
5	shot	Victor Boniface	Leverkusen	0.605	0.9844



6.7 Vorteile des VAEP-Modells

- Kontextuelle Bewertung: Das VAEP-Modell berücksichtigt den Kontext jeder Aktion, was zu einer genaueren und differenzierteren Bewertung führt.
- Offensive und defensive Beiträge: Es bewertet sowohl offensive als auch defensive Aktionen, wodurch eine umfassendere Analyse der Spielerleistung möglich ist.
- Anwendung maschinellen Lernens: Durch den Einsatz von maschinellem Lernen kann das Modell aus großen Datensätzen lernen und präzise Vorhersagen treffen.

6.8 Zusammenfassung

Das VAEP-Modell bietet eine detaillierte und kontextbezogene Bewertung der Spieleraktionen im Fußball. Durch die Berücksichtigung sowohl offensiver als auch defensiver Beiträge und den Einsatz fortschrittlicher Analyseverfahren liefert es wertvolle Einblicke in die Leistung von Spielern und deren Einfluss auf das Spiel.

KAPITEL 7

Datenscraping

7.1 Was ist Datenscraping?

Datenscraping ist der Prozess des automatisierten Extrahierens von Informationen aus Webseiten. In diesem Projekt verwende ich Datenscraping, um aktuelle Fußballspieldaten zu sammeln, die ich für die Bewertung der Spieler benötige.

7.2 Tools und Bibliotheken

Um Daten zu scrapen, können verschiedene Tools und Bibliotheken verwendet werden. Zu den beliebtesten gehören:

- **BeautifulSoup**: Eine Bibliothek zum Parsen von HTML- und XML-Dokumenten und zum Extrahieren von Daten daraus.
- **Selenium**: Ein Tool zur Automatisierung von Webbrowsern, das auch zum Scrapen von dynamisch geladenen Inhalten verwendet werden kann.
- **Requests**: Eine Bibliothek zum Senden von HTTP-Anfragen und zum Abrufen von Inhalten aus Webseiten.

7.3 Wer bietet überhaupt Event-Daten auf seiner Internetseite an?

Für einige andere Projekte habe ich bereits eigene Scraper geschrieben, um z.B. Spielerdaten von transfermarkt oder Match-Statistiken vom DFB. Unsere Event-Daten beziehen wir von whoscored.com. Es gab in der Vergangenheit immer auch mal andere Seiten und Anbieter (DFB MatchCenter oder Fivethirtyeight), aber Stand heute (Juni 2024) ist whoscored die beste Quelle für eine Vielzahl an Wettbewerben und Saisons. Whoscored bietet ein eigenes Matchcenter an, welches auf Basis von Eventdaten visualisiert wird. Diese Daten liegen im Quellcode der Seite vor und können mit einem Scraper ausgelesen werden. Whoscored Daten basieren auf den Daten von Opta.

7.4 Welche Wettbewerbe werden von whoscored abgedeckt?

Detailed statistics (Opta) for Premier League (England), Serie A (Italy), La Liga (Spain), Bundesliga (Germany), Ligue 1 (France), Championship (England), Eredivisie (Netherlands), Premier League (Russia), MLS (USA), Super Lig (Turkey), Serie A (Brazil) & UEFA Champions League & Europa League (https://www.whoscored.com/AboutUs)

7.5 Wie funktioniert das Scrapen von Daten nun konkret?

Das Scrapen von Daten erfolgt in mehreren Schritten:

Zuerst benötigen wir die URL des Spiels, von dem wir die Daten sammeln möchten. Diese URL wird dann an den Scraper übergeben, der die Seite herunterlädt und den HTML-Code extrahiert.

Dann müssen wir die relevanten Daten aus dem HTML-Code extrahieren. Dazu verwenden wir die BeautifulSoup-Bibliothek, die es uns ermöglicht, den HTML-Code zu durchsuchen und die gewünschten Daten zu finden.

Aus dem HTML-Code können wir dann die benötigten Daten extrahieren und in einem geeigneten Format speichern, z.B. in einer CSV-Datei oder in eine JSON-Datei. Ich rate zu JSON-Dateien, da diese später einfach mit Python zu verarbeiten sind.

Ein Spiel sind nicht alle Spiele. Damit ich nicht jede einzelne URL von Hand eingeben muss, scrape ich vorher noch die URLs aller Spiele einer Saison und speichere sie in einer Datei ab. Diese Datei wird dann vom Scraper gelesen und so werden nach und nach die Daten für jedes Spiel gesammelt.

7.6 Herausforderungen

Da whoscored einige Maßnahmen ergriffen hat, um das Scrapen ihrer Seite zu erschweren, müssen wir auch einige Tricks anwenden, um die Daten zu erhalten:

- Setzen von Headern und Cookies, um den Zugriff auf die Seite als Mensch zu simulieren (Selenium)
- Verwendung von Zeitverzögerungen, um zu verhindern, dass die Seite den Zugriff blockiert
- Verwendung von Proxies, um den Zugriff auf die Seite zu anonymisieren und zu verhindern, dass die IP-Adresse blockiert wird

Es gibt viele Bibliotheken und Tools, die das Scrapen von Daten erleichtern, aber es erfordert immer noch ein gewisses Maß an technischem Verständnis und Erfahrung, um die Daten erfolgreich zu sammeln. Die meisten Webseiten ändern regelmäßig ihre Struktur, um das Scrapen ihrer Daten zu erschweren. Daher ist es wichtig, den Scraper regelmäßig zu überprüfen und anzupassen, um sicherzustellen, dass er weiterhin funktioniert.

Ich habe mit der Bibliothek Soccerdata gearbeitet, musste aber immer wieder Teile im Code verändern und individuell anpassen, da whoscored immer wieder minimale Änderungen vorgenommen hat und die Bibliothek bis zum neuen Release nicht mehr funktioniert hat.

7.7 Fazit

Über Monate hinweg habe ich die Daten für die Saisons 2022/23 und 2023/24 gesammelt und den Datenbestand immer wieder aktualisiert: Über 8000 Spiele aus 13 Wettbewerben machen mehr als 10 Millionen Events und schon mehr als 10 GB an (Roh-)Daten.

- Empfehlung für ein tieferes Verständnis bzgl. Datenscraping auf aktuellem Stand: John Watson Rooney

7.8 Code Beispiel

Snippet für die Anwendung von Soccerdata, um die Daten der Bundesliga aus der Saison 2023-24 zu sammeln:

```
import soccerdata as sd

league = 'GER-Bundesliga'
season = '23-24'

whoscored = sd.WhoScored(leagues=[league], seasons=season)
# load the schedule for the selected leagues and seasons and store all gameurls in a␣
↪dataframe
df = whoscored.read_schedule()

# remove all games that are not finished yet
df = df.loc[~df['url'].str.contains('Show')]
df = df.sort_values(by='date')

# iterate over all rows in the dataframe and process each game
def process_row(row):
    match_id = row['game_id']
    try:
        print(f"processing match: {match_id}")
        whoscored.read_events(match_id=match_id, force_cache=True, output_fmt='events
↪')
    except Exception as e:
        print(f"error while processing match: {match_id}")
        print(f"errordetails: {e}")

df.apply(process_row, axis=1)
```

KAPITEL 8

Dateien verarbeiten und vorbereiten

8.1 Dateien umbenennen und verschieben

Nachdem ich die Daten gesammelt habe, muss ich sie umbenennen und verschieben, um sie für die Analyse und Modellierung vorzubereiten. Die Umbenennung und Verschiebung der Dateien umfasst das Anpassen der Dateinamen und das Verschieben der Dateien in den richtigen Ordner.

Aus der Datei ‚1743694.json' in dem Ordner ‚GER-Bundesliga_2324' wird die Datei 3_232430_1743694.json. Jeder Wettbewerb hat eine eigene ID, die ich vorher bei whoscored ausgelesen habe. Ich habe danach jedem Wettbewerb in Kombination mit der Saison eine eigene Season_ID vergeben, die den nächsten Suffix bildet. So kann ich hinterher nur am Dateinamen erkennen, um welchen Wettbewerb und welche Saison es sich handelt.

Die Daten werden verschoben, damit ich die Rohdaten von den bereinigten Daten trennen kann. Eventuell möchte ich die Rohdaten noch für andere Projekte verwenden. Um den Prozess zu beschleunigen, werden bereits verarbeitete Spiele übersprungen.

8.2 Code Beispiel

Die Funktion dazu:

```
def copy_and_rename_json_files(main_folder: str, competitions_df: pd.DataFrame) ->␣
↪None:
    """
    Copy JSON files from subfolders, rename them based on competition data, and save␣
↪them to a 'data' folder.

    Args:
        main_folder (str): The main directory containing subfolders with JSON files.
        competitions_df (pd.DataFrame): DataFrame containing competition information.

    Returns:
        None
    """
    # Create a subfolder 'data' inside the main folder
    test_folder = os.path.join(main_folder, 'data')
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    os.makedirs(test_folder, exist_ok=True)

    for root, dirs, files in os.walk(main_folder):
        # Skip the 'data' folder
        if root == test_folder:
            print(f"Skipping 'data' folder: {root}")
            continue

        total_files = 0
        copied_files = 0

        for file in files:
            if file.endswith('.json'):
                total_files += 1
                source_path = os.path.join(root, file)

                # Check if the JSON file content is "null"
                try:
                    with open(source_path, 'r') as f:
                        content = json.load(f)
                        if content is None:
                            print(f"Skipping file with 'null' content: {source_path}")
                            continue
                except json.JSONDecodeError:
                    print(f"Invalid JSON in file: {source_path}")
                    continue
                except Exception as e:
                    print(f"Error reading file {source_path}: {e}")
                    continue

                # Extract match information from folder and file names
                folder_name_parts = root.split(os.path.sep)[-1].replace('-', '_').
↪split('_')
                match_id = os.path.splitext(file)[0]

                if len(folder_name_parts) == 3:
                    short_name, competition_name, season_name_short = folder_name_
↪parts
                    match_info = competitions_df[
                        (competitions_df['short_name'] == short_name) &
                        (competitions_df['competition_name'] == competition_name) &
                        (competitions_df['season_name_short'] == int(season_name_
↪short))
                        ]

                    if not match_info.empty:
                        competition_id = match_info['competition_id'].values[0]
                        season_id = match_info['season_id'].values[0]

                        new_filename = f"{competition_id}-{season_id}-{match_id}.json"
                        destination_path = os.path.join(test_folder, new_filename)

                        # Check if the file already exists in the destination folder
                        if not os.path.exists(destination_path):
                            shutil.copyfile(source_path, destination_path)
                            copied_files += 1
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                            print(f"Copied and renamed file: {source_path} ->
↪{destination_path}")
                        else:
                            print(f"File already exists: {destination_path}")

        # Display folder statistics
        print(f"Folder processed: {root}")
        print(f"Total files: {total_files}, Copied files: {copied_files}")
```

KAPITEL 9

Daten in SPADL-Format umwandeln und speichern

9.1 Was ist das SPADL-Format?

SPADL (Soccer Player Action Description Language) ist ein Standard zur Beschreibung von Fußballaktionen. Dieser Standard wurde entwickelt, um eine einheitliche und detaillierte Darstellung von Aktionen auf dem Spielfeld zu ermöglichen, was die Analyse und Vergleichbarkeit von Daten erheblich verbessert.

9.1.1 Grundprinzipien von SPADL

Einheitliche Darstellung: SPADL bietet ein konsistentes Format zur Beschreibung von Fußballaktionen, das unabhängig von der Quelle der Daten ist. Dies erleichtert die Integration und Analyse von Daten aus verschiedenen Quellen.

Detaillierte Beschreibung: Jede Aktion wird mit einer Vielzahl von Merkmalen beschrieben, darunter:

- Art der Aktion (z.B. Pass, Schuss, Dribbling)
- Ausgangs- und Endposition (x- und y-Koordinaten)
- Zeitpunkt der Aktion im Spiel (Spielminute)
- Zusätzliche Attribute wie das verwendete Körperteil (Fuß, Kopf) und der Ausgang (erfolgreich, nicht erfolgreich)

9.1.2 Struktur von SPADL-Daten

SPADL-Daten sind in Form von Tabellen organisiert, wobei jede Zeile eine einzelne Aktion darstellt und die Spalten die verschiedenen Merkmale der Aktion enthalten. Wichtige Spalten sind unter anderem:

- Game ID: Identifikation des Spiels
- Action ID: Identifikation der Aktion
- Type: Typ der Aktion (z.B. Pass, Shot)
- Result: Ergebnis der Aktion (Success, Fail)
- Start/End Coordinates: Start- und Endkoordinaten der Aktion
- Body Part: Körperteil, mit dem die Aktion ausgeführt wurde

- Time: Zeitpunkt der Aktion im Spiel

9.1.3 Vorteile von SPADL

- Interoperabilität: Durch die standardisierte Darstellung können Daten aus verschiedenen Quellen leicht kombiniert und verglichen werden.
- Detailtiefe: Die detaillierte Beschreibung ermöglicht tiefgehende Analysen und Einblicke in das Spielgeschehen.
- Flexibilität: SPADL ist flexibel genug, um eine Vielzahl von Aktionen und Szenarien abzudecken, von einfachen Pässen bis hin zu komplexen Dribblings und Schüssen.

9.2 Laden, umwandeln und speichern der Daten in einer HDF5-Datei

Aktuell befinden sich all unsere Spiele in json-Files in einem Ordner. Diese Daten müssen wir nun laden und ins SPADL Format umwandeln, um sie später für die Modellierung verwenden zu können.

Ich bediene mich einerseits der OptaLoader Klasse, um die Spiele umzuwandeln. Andererseits brauche ich meinen eigenen Code, der prüft, ob bereits Spiele umgewandelt wurden. Falls ja, dann müssen wir diese Spiele nicht noch einmal umwandeln, sondern können sie direkt aus der HDF5-Datei laden. Dies spart Zeit und Rechenleistung, wenn wir davon ausgehen, dass wir unsere Daten zum Beispiel wöchentlich aktualisieren wollen.

Am Ende speichern wir alle Spiele in einer HDF5-Datei ab, um sie später für die Modellierung zu verwenden.

9.3 HDF5

HDF5 ist ein Dateiformat, das speziell für große Datensätze entwickelt wurde. Es ermöglicht den schnellen Zugriff auf große Datenmengen und ist daher ideal für die Speicherung von SPADL-Daten geeignet.

9.4 Codebeispiel

Konkret sieht das nun so aus:

```
def load_convert_to_spadl_and_safe_all_matches_to_h5(datafolder: str, spadl_
↪datafolder: str, competitions: pd.DataFrame) -> None:
    """
    Load match data, convert it to SPADL format, and save it to an HDF5 file.

    Args:
        datafolder (str): Path to the data folder containing the event data.
        spadl_datafolder (str): Path to the folder where the SPADL data will be saved.
        competitions (pd.DataFrame): DataFrame containing competition and season␣
↪information.

    Returns:
        None
    """
    api = OptaLoader(root=datafolder, parser="whoscored")

    # Get games from all selected competitions
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    games = pd.concat([
        api.games(row.competition_id, row.season_id)
        for row in competitions.itertuples()
    ])
    print(f"Games loaded: {len(games)}")

    games_verbose = tqdm.tqdm(list(games.itertuples()), desc="Loading game data")
    teams, players = [], []
    actions = {}
    for game in games_verbose:
        # Load data
        teams.append(api.teams(game.game_id))
        players.append(api.players(game.game_id))
        events = api.events(game.game_id)

        # Convert data
        actions[game.game_id] = spadl.opta.convert_to_actions(events, game.home_team_
↪id)
    print(f"Games processed: {len(games)}")

    teams = pd.concat(teams).drop_duplicates(subset="team_id")
    players = pd.concat(players)

    # Create data folder if it doesn't exist
    if not os.path.exists(spadl_datafolder):
        os.mkdir(spadl_datafolder)
        print(f"Directory {spadl_datafolder} created.")

    spadl_h5 = os.path.join(spadl_datafolder, "spadl-opta.h5")

    # Store all SPADL data in h5-file
    with pd.HDFStore(spadl_h5) as spadlstore:
        spadlstore["competitions"] = competitions
        spadlstore["games"] = games
        spadlstore["teams"] = teams
        spadlstore["players"] = players[['player_id', 'player_name']].drop_
↪duplicates(subset='player_id')
        spadlstore["player_games"] = players[['player_id', 'game_id', 'team_id', 'is_
↪starter', 'minutes_played', 'starting_position']]
        for game_id in actions.keys():
            spadlstore[f"actions/game_{game_id}"] = actions[game_id]
```

KAPITEL 10

Machine Learning Prozess

Wie kann ich nun aber die Spielerleistung konkret berechnen? All meine Spielaktionen, die ich nun schon im SPADL-Format vorliegen habe, bekommen den Kontext der Spielsituation. Diese nennt man im Machine Learning Prozess Labels und Features.

10.1 Labels

Labels sind die Werte, die ich vorhersagen möchte. In meinem Fall sind das die Tore, die ein Spieler erzielt hat. Diese Werte sind die abhängige Variable, die ich mit meinen Features vorhersagen möchte. Um aus den bisherigen Daten zu lernen, welche Spielaktionen wahrscheinlich ein Tor ergeben, müssen die Tore in den Daten markiert werden. Die Tabelle mit meinen Spielaktionen bekommt nun zwei neue Spalten:

- `scores`: wird als True markiert, wenn in den nächsten 10 Aktionen ein Tor erzielt wird
- `concedes`: wird als True markiert, wenn in den nächsten 10 Aktionen ein Tor kassiert wird

Wieder wird darauf geachtet, ob die Spielaktion schon in einem früheren Durchgang bearbeitet wurde, um Rechenleistung zu sparen.

```
def compute_labels(spadl_h5: str, labels_h5: str) -> None:
    """
    Compute and store VAEP labels for each game in the given SPADL HDF5 file.

    Args:
        spadl_h5 (str): Path to the SPADL HDF5 file containing game actions.
        labels_h5 (str): Path to the HDF5 file where labels will be stored.

    Returns:
        None
    """
    games = pd.read_hdf(spadl_h5, "games")
    print("Number of games:", len(games))

    yfns = [lab.scores, lab.concedes, lab.goal_from_shot]

    labels = 0
    with pd.HDFStore(labels_h5) as store_labels:
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        for game in tqdm.tqdm(list(games.itertuples()), desc=f"Computing and storing␣
↪labels in {labels_h5}"):
            game_key = f"game_{game.game_id}"
            actions_key = f"actions/{game_key}"
            if game_key not in store_labels:
                labels += 1
                actions = pd.read_hdf(spadl_h5, actions_key)
                Y = pd.concat([fn(spadl.add_names(actions)) for fn in yfns], axis=1)
                store_labels.put(game_key, Y)

    print("Number of new labels:", labels)
```

10.2 Features

Features sind die Werte, die ich verwende, um die Labels vorherzusagen. In meinem Fall sind das die Spielaktionen, die ein Spieler durchgeführt hat. Diese Werte sind die unabhängigen Variablen, die ich verwende, um die abhängige Variable (die Tore) vorherzusagen.

Die Features, die ich verwenden möchte, ergeben sich aus den Informationen, die ich durch die Spielaktion erhalte. Mir stehen folgende Informationen zur Verfügung, aus denen ich Features ableiten kann:

- Zeit: In welcher Spielsekunde fand die Aktion statt?
- Ort: Wo fand die Aktion statt? Start- und Endpunkte jeder Aktion sind bekannt.
- Art der Aktion: War es ein Pass, ein Schuss, ein Dribbling oder ein Tackling?
- Resultat der Aktion: War die Aktion erfolgreich oder nicht?
- Körperteil: Mit welchem Körperteil wurde die Aktion ausgeführt?

Hieraus kann ich nun folgende Features ableiten:

- `actiontype`: Art der Aktion (Pass, Schuss, Dribbling, Tackling)
- `bodypart`: Körperteil, mit dem die Aktion ausgeführt wurde
- `result`: Ergebnis der Aktion (Success, Fail, OwnGoal, YellowCard, RedCard, Offside)
- `goals_home`: Anzahl der Tore des Heimteams nach der Aktion
- `goals_away`: Anzahl der Tore des Auswärtsteams nach der Aktion
- `goals_diff`: Differenz der Tore zwischen Heim- und Auswärtsteam nach der Aktion
- `start_x`: x-Koordinate des Startpunkts der Aktion
- `start_y`: y-Koordinate des Startpunkts der Aktion
- `end_x`: x-Koordinate des Endpunkts der Aktion
- `end_y`: y-Koordinate des Endpunkts der Aktion
- `movement`: Distanz, die der Spieler in der Aktion zurückgelegt hat
- `space_delta`: Distanz zwischen der letzten Aktion und der aktuellen Aktion
- `start_polar`: Polarkoordinaten des Startpunkts der Aktion als Tuple (Winkel, Distanz) zum gegnerischen Tor
- `end_polar`: Polarkoordinaten des Endpunkts der Aktion als Tuple (Winkel, Distanz) zum gegnerischen Tor
- `team`: Prüft, ob der Ballbesitz gewechselt hat zwischen den Aktionen
- `time`: Zeitpunkt der Aktion im Spiel
- `time_delta`: Zeitdifferenz zur letzten Aktion

Da Machine Learning-Algorithmen darauf ausgelegt sind, mit Zahlen bzw. Vektoren zu arbeiten, müssen die Features in numerische Werte umgewandelt werden. Das Verfahren, um kategorische Werte in numerische Werte umzuwandeln, wird oft als One-Hot-Encoding bezeichnet. Hierbei wird jede Kategorie in eine eigene Spalte umgewandelt und mit 0 oder 1 kodiert.

```
def compute_features(spadl_h5: str, features_h5: str) -> None:
    """
    Compute and store VAEP features for each game in the given SPADL HDF5 file.

    Args:
        spadl_h5 (str): Path to the SPADL HDF5 file containing game actions.
        features_h5 (str): Path to the HDF5 file where features will be stored.

    Returns:
        None
    """
    games = pd.read_hdf(spadl_h5, "games")
    print("Number of games:", len(games))

    xfns = [
        fs.actiontype,
        fs.actiontype_onehot,
        fs.bodypart,
        fs.bodypart_onehot,
        fs.result,
        fs.result_onehot,
        fs.goalscore,
        fs.startlocation,
        fs.endlocation,
        fs.movement,
        fs.space_delta,
        fs.startpolar,
        fs.endpolar,
        fs.team,
        fs.time,
        fs.time_delta
    ]

    features = 0
    with pd.HDFStore(features_h5) as store_features:
        for game in tqdm.tqdm(list(games.itertuples()), desc=f"Generating and storing␣
↪features in {features_h5}"):
            game_key = f"game_{game.game_id}"
            actions_key = f"actions/{game_key}"
            if game_key not in store_features:
                features += 1
                actions = pd.read_hdf(spadl_h5, actions_key)
                gamestates = fs.gamestates(spadl.add_names(actions), 3)
                gamestates = fs.play_left_to_right(gamestates, game.home_team_id)

                X = pd.concat([fn(gamestates) for fn in xfns], axis=1)
                store_features.put(game_key, X)

    print("Number of new features:", features)
```

10.3 Das Machine Learning Modell

Hier zeige ich, wie ich ein ganz neues Modell trainiere. Im späteren Verlauf, wenn immer wieder neue Spiele hinzukommen, wird das bereits trainierte Modell genutzt, um damit die neuen Spiele zu bewerten, sodass nicht immer wieder ein neues Modell trainiert werden muss.

In Kurz: Ich brauche einen Algorithmus, der die kontextbezogenen Spielaktionen in zwei Klassen vorhersagt (Es wird ein Tor erzielt / Es wird kein Tor erzielt). Wichtig ist hierbei, dass ich aus der Vorhersagewahrscheinlichkeit für jede Spielaktion die Leistung eines Spielers ableiten kann.

Was passiert nun?

1. Ich lade alle Spiele und deren Labels und Features aus der HDF5-Datei.
2. Ich teile die Spiele in Trainings- und Testdaten auf. Die Trainingsdaten werden verwendet, um das Modell zu trainieren, während die Testdaten verwendet werden, um die Leistung des Modells zu bewerten. Eine Aufteilung im Bereich 70% Trainingsdaten und 30% Testdaten ist üblich, kann aber je nach Datensatz auch variieren.

```
games = pd.read_hdf(spadl_h5, "games")
    print("nb of games:", len(games))

    # Split games into training and test set
    train_test_ratio = 0.7
    games = games.sort_values(["game_date"]).reset_index(drop=True)
    train_size = int(len(games) * train_test_ratio)
    traingames = games[:train_size]
    testgames = games[train_size:]
```

3. Ich wähle die Features aus, die ich für das Training verwenden möchte und wähle die Anzahl der Spielaktionen aus, die ein Gamestate bilden. In meinem Fall sind das drei Aktionen. Hierbei ist zu beachten, dass die Anzahl der Features und die Anzahl der Aktionen in einem Gamestate die Rechenleistung erheblich beeinflusst.

```
    # Define the feature functions
xfns = [
    fs.actiontype, fs.actiontype_onehot, fs.bodypart, fs.bodypart_onehot, fs.result,␣
↪fs.result_onehot,
    fs.goalscore, fs.startlocation, fs.endlocation, fs.movement, fs.space_delta, fs.
↪startpolar, fs.endpolar,
    fs.team, fs.time, fs.time_delta
]
nb_prev_actions = 3
Xcols = fs.feature_column_names(xfns, nb_prev_actions)
```

4. Die Trainingsdaten laden und in das Modell übergeben. Als Modell wird hier ein XGBoost-Modell verwendet.

10.3.1 XGBoost

XGBoost ist eine leistungsstarke Bibliothek für maschinelles Lernen, die auf dem Gradient Boosting-Algorithmus basiert. Es ist besonders gut geeignet für strukturierte Daten und dabei als erfolgreich erwiesen. Der Algorithmus gehört zum supervised-learning (die Daten sind bereits gelabelt) und basiert auf Entscheidungsbäumen. Die Entscheidungsbäume werden nacheinander trainiert und verbessern sich dabei ständig, indem sie die Fehler der vorherigen Bäume korrigieren. Dieser Prozess wird als Boosting bezeichnet. Dieser Algorithmus gibt uns nicht nur eine Vorhersage, ob eine Aktion ein Tor zur Folge hat oder nicht, sondern auch noch die Wahrscheinlichkeit, mit der das passiert. Aus diesen Wahrscheinlichkeitsdifferenzen kann ich dann die Leistung eines Spielers ableiten.

Die eingegebenen Hyperparameter dienen als Startpunkt. In einem weiteren Kapitel werde ich näher auf die Optimierung dieser Parameter eingehen.

- `n_estimators = 50`: Dieser Parameter bestimmt die Anzahl der Bäume im Modell. Eine höhere Anzahl von Bäumen kann zu einer besseren Leistung führen, aber auch die Trainingszeit erhöhen und möglicherweise zu Überanpassung (Overfitting) führen. Es ist oft nützlich, diesen Wert zu optimieren, indem man verschiedene Werte testet.
- `max_depth = 3`: Dieser Parameter legt die maximale Tiefe der einzelnen Bäume fest. Eine geringere Tiefe verhindert Überanpassung, kann jedoch die Modellleistung einschränken, wenn das Modell nicht genügend Komplexität hat, um die Datenstruktur zu erfassen. Auch hier kann eine Optimierung durch Testen verschiedener Werte hilfreich sein.
- `n_jobs = -1`: Dieser Parameter gibt an, wie viele parallele Threads zur Verfügung stehen. Der Wert -1 bedeutet, dass alle verfügbaren Kerne genutzt werden, was die Trainingszeit auf Mehrkernprozessoren verkürzen kann. Dies ist in der Regel eine gute Wahl für schnellere Trainingszeiten.
- `verbosity = 1`: Dieser Parameter steuert die Menge der Ausgaben während des Trainings. Ein Wert von 1 gibt grundlegende Informationen aus. Für detailliertere Informationen kann ein höherer Wert verwendet werden, aber das ist eher eine Präferenzfrage und hat keinen Einfluss auf die Modellleistung.

Diese Parameter bieten eine solide Grundlage für das Trainieren der Modelle, können aber je nach Datensatz und Problemstellung angepasst werden, um die beste Leistung zu erzielen.

```
# Get training data
    X_train, Y_train = getXY(traingames, Xcols)
    print("X:", list(X_train.columns))
    print("Y:", list(Y_train.columns))

    # Train classifiers
    Y_hat = pd.DataFrame()
    models = {}
    for col in list(Y_train.columns):
        model = xgboost.XGBClassifier(n_estimators=50, max_depth=3, n_jobs=-1,␣
↪verbosity=1)
        model.fit(X_train, Y_train[col])
        models[col] = model
        joblib.dump(model, f"{col}_model.pkl")
```

10.3.2 Beispiel Entscheidungsbaum

Hier noch einmal ein einfaches Beispiel, wie man sich so einen Entscheidungsbaum bezogen auf unser Thema vorstellen kann.

```
                                    actiontype
                                  /     |     |       \
                              Schuss  Pass Dribbling Tackling
                              /                           \
                    start_x > 75?                  start_x > 75?
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                    /      \                    /      \
                   Ja      Nein               Ja       Nein
                  /   \                      /   \
        bodypart = Fuß bodypart  Fuß    space_delta > 10  space_delta  10
           /           \                    /        \        /        \
      start_y < 40   start_y  40     Tor     Kein Tor   Tor      Kein Tor
        / \           / \
       Tor Kein Tor  Tor Kein Tor
```

10.4 Modell Evaluierung

Hierfür wird das Modell mit den Testdaten geladen und die Vorhersagen mit den tatsächlichen Labels verglichen. Die Genauigkeit des Modells wird anhand verschiedener Metriken bewertet.

```python
# Evaluate models
    testX, testY = getXY(testgames, Xcols)
    def evaluate(y: pd.Series, y_hat: pd.Series) -> None:
        p = sum(y) / len(y)
        base = [p] * len(y)
        brier = brier_score_loss(y, y_hat)
        print(f"  Brier score: {brier:.5f} ({brier / brier_score_loss(y, base):.5f})")
        ll = log_loss(y, y_hat)
        print(f"  log loss score: {ll:.5f} ({ll / log_loss(y, base):.5f})")
        print(f"  ROC AUC: {roc_auc_score(y, y_hat):.5f}")

    for col in testY.columns:
        Y_hat[col] = [p[1] for p in models[col].predict_proba(testX)]
        print(f"### Y: {col} ###")
        evaluate(testY[col], Y_hat[col])
```

Hier ein Beispiel für eine Modellbewertung, als ich zuletzt neue Spiele hinzugefügt habe:

```
Selecting features: 100%|██████████| 123/123 [00:00<00:00, 201.03it/s]
Selecting label: 100%|██████████| 123/123 [00:00<00:00, 467.40it/s]
### Y: scores ###
  Brier score: 0.01298 (0.84324)
  log loss score: 0.06392 (0.79390)
  ROC AUC: 0.80301
### Y: concedes ###
  Brier score: 0.00370 (0.95281)
  log loss score: 0.02218 (0.86845)
  ROC AUC: 0.78556
Loading game ids: 100%|██████████| 123/123 [00:00<00:00, 181.66it/s]
Saving predictions per game: 100%|██████████| 123/123 [00:16<00:00,  7.45it/s]
```

- **Brier-Score**: Der Brier-Score ist wie ein Fehlerpunkt. Er misst, wie nah meine Vorhersagen an den tatsächlichen Ergebnissen sind. Ein kleiner Brier-Score bedeutet, dass ich sehr gut geschätzt habe. Weitere Infos
- **Log-Loss Score**: Der Log-Loss-Score ist wie eine Strafe, die höher wird, je schlechter meine Vorhersagen sind. Je kleiner der Log-Loss-Score, desto besser meine Vorhersagen. Weitere Infos

- **ROC-AUC Wert**: Die ROC AUC-Werte zeigen eine gute Trennschärfe des Modells. Werte über 0.8 sind gut und deuten auf eine hohe Fähigkeit des Modells hin, zwischen den Klassen zu unterscheiden. Ein Wert von 1 bedeutet perfekt, und ein Wert von 0.5 bedeutet, dass es wie Raten ist.

10.5 Wahrscheinlichkeitswerte auslesen und VAEP-Werte berechnen

Das Ergebnis der Berechnungen sind nun die Wahrscheinlichkeitswerte für der Spielaktionen. Diese kann ich nun nutzen, um die VAEP-Werte für jede Aktion zu berechnen.

Zunächst einmal speichere ich die Predictions in der HDF5-Datei ab, um sie später wieder verwenden zu können. Anschließend berechne ich die VAEP-Werte für jede Aktion und speichere sie ebenfalls in der HDF5-Datei ab.

```
# Save predictions per game
    A = []
    for game_id in tqdm.tqdm(games.game_id, "Loading game ids"):
        Ai = pd.read_hdf(spadl_h5, f"actions/game_{game_id}")
        A.append(Ai[["game_id"]])
    A = pd.concat(A).reset_index(drop=True)

    grouped_predictions = pd.concat([A, Y_hat], axis=1).groupby("game_id")
    for k, df in tqdm.tqdm(grouped_predictions, desc="Saving predictions per game"):
        df = df.reset_index(drop=True)
        df[Y_hat.columns].to_hdf(predictions_h5, f"game_{int(k)}")
```

Aus den Wahrscheinlichkeitswerten können wir nun die VAEP-Werte berechnen.

- Der offensive Wert einer Aktion wird berechnet als die Differenz der Wahrscheinlichkeiten ein Tor zu erzielen vor der Aktion und nach der Aktion.
- Der defensive Wert einer Aktion wird berechnet als die Differenz der Wahrscheinlichkeit ein Gegentor zu kassieren vor der Aktion und nach der Aktion.
- Die Summe aus offensivem und defensivem Wert ergibt den VAEP-Wert einer Aktion.

Hier werden alle Aktionen geladen und mit den Spielerdaten und Spieldaten verknüpft. Anschließend werden die VAEP-Werte berechnet und in einem DataFrame gespeichert.

```
A = []
    for game in tqdm.tqdm(list(games.itertuples()), desc="Loading actions"):
        actions = pd.read_hdf(spadl_h5, f"actions/game_{game.game_id}")
        actions = (
            spadl.add_names(actions)
            .merge(players, how="left")
            .merge(teams, how="left", )
            .sort_values(["game_id", "period_id", "action_id"])
            .reset_index(drop=True)
        )
        preds = pd.read_hdf(predictions_h5, f"game_{game.game_id}")
        values = vaepformula.value(actions, preds.scores, preds.concedes)
        A.append(pd.concat([actions, preds, values], axis=1))
    A = pd.concat(A).sort_values(["game_id", "period_id", "time_seconds"]).reset_
↪index(drop=True)
```

Aus den aufsummierten Werten könnte ich nun schon eine Bewertung der Spielerleistung für einen beliebigen Zeitraum ableiten. Da Spiele mit mehr Spielminuten vermeintlich mehr Value generieren können, wird das VAEP-Rating an die Spielzeit angepasst und auf 90 Minuten normalisiert.

Für die Bundesliga 2023/24 sieht das Ergebnis für die Top 10 dann so aus (min. 1500 Spielminuten):

tabularytabulary

Spieler	Minuten	Verein	VAEP Rating
Harry Kane	3,084	Bayern	0.5787
Robin Hack	1,533	Borussia M.Gladbach	0.4839
Jamal Musiala	1,882	Bayern	0.4366
Florian Wirtz	2,501	Leverkusen	0.4248
Donyell Malen	1,928	Borussia Dortmund	0.4234
Kevin Stöger	2,820	Bochum	0.4129
Deniz Undav	2,234	Stuttgart	0.4123
Álex Grimaldo	2,929	Leverkusen	0.3971
Benjamin Sesko	1,589	RBL	0.3883
Serhou Guirassy	2,347	Stuttgart	0.3793

Diese Daten möchte ich nun für weitere Berechnungen und Analysen speichern.

KAPITEL 11

Datenbank

Während HDF5-Dateien für die Speicherung großer numerischer Datensätze und für wissenschaftliche Anwendungen nützlich sind, bieten Datenbanken umfassendere Funktionen für Datenabfragen, Integrität, Sicherheit und Skalierbarkeit. Der wichtigste Faktor ist jedoch, dass mit einer Datenbank das Projekt und die Daten von mehreren Personen gleichzeitig genutzt werden können und von überall auf der Welt darauf zugegriffen werden kann.

Ebenfalls ist es bei den Dateigrößen unmöglich eine Webanwendung zu entwickeln, die auf HDF5-Dateien basiert. Unsere aktuellen Dateien haben bisher folgende Größen:

- `spadl.h5`: 11.06 GB
- `labels.h5`: 128 MB
- `features.h5`: 8.39 GB
- `predictions.h5`: 270 MB

Bei der Datenbank handelt es sich um eine relationale Datenbank, die aus mehreren Tabellen besteht. Die Tabellen sind miteinander verknüpft und können über Abfragen (SQL) abgerufen werden. ich habe mich für MariaDB entschieden, da es eine Open-Source-Alternative zu MySQL ist und eine hohe Kompatibilität aufweist.

11.1 Datenbankstruktur

Hier möchte ich die Struktur der Datenbank anhand der Bundesligasaison 2023/24 erläutern.

11.1.1 Tabellen für alle Ligen und Saisons

Die Datenbank besteht aus folgenden Tabellen, die für alle Ligen und Saison universal sind:

- `teams`: Enthält Informationen zu den Teams. (team_id, team_name)
- `players`: Enthält Informationen zu den Spielern. (player_id, player_name)
- `result`: Enthält Informationen zu dem Ergebnis eines Events. (result_id, result_name)
- `bodypart`: Enthält Informationen zu dem Körperteil, mit dem ein Spieler den Ball berührt. (bodypart_id, bodypart_name)
- `actiontypes`: Enthält Informationen zu den Aktionen eines Spielers. (actiontype_id, actiontype_name)

- `competitions`: Enthält Informationen zu den Wettbewerben. (competition_id, competition_name, country_id, season_id, season_name, ...)

11.1.2 Tabellen für die Bundesliga-Saison 2023/24

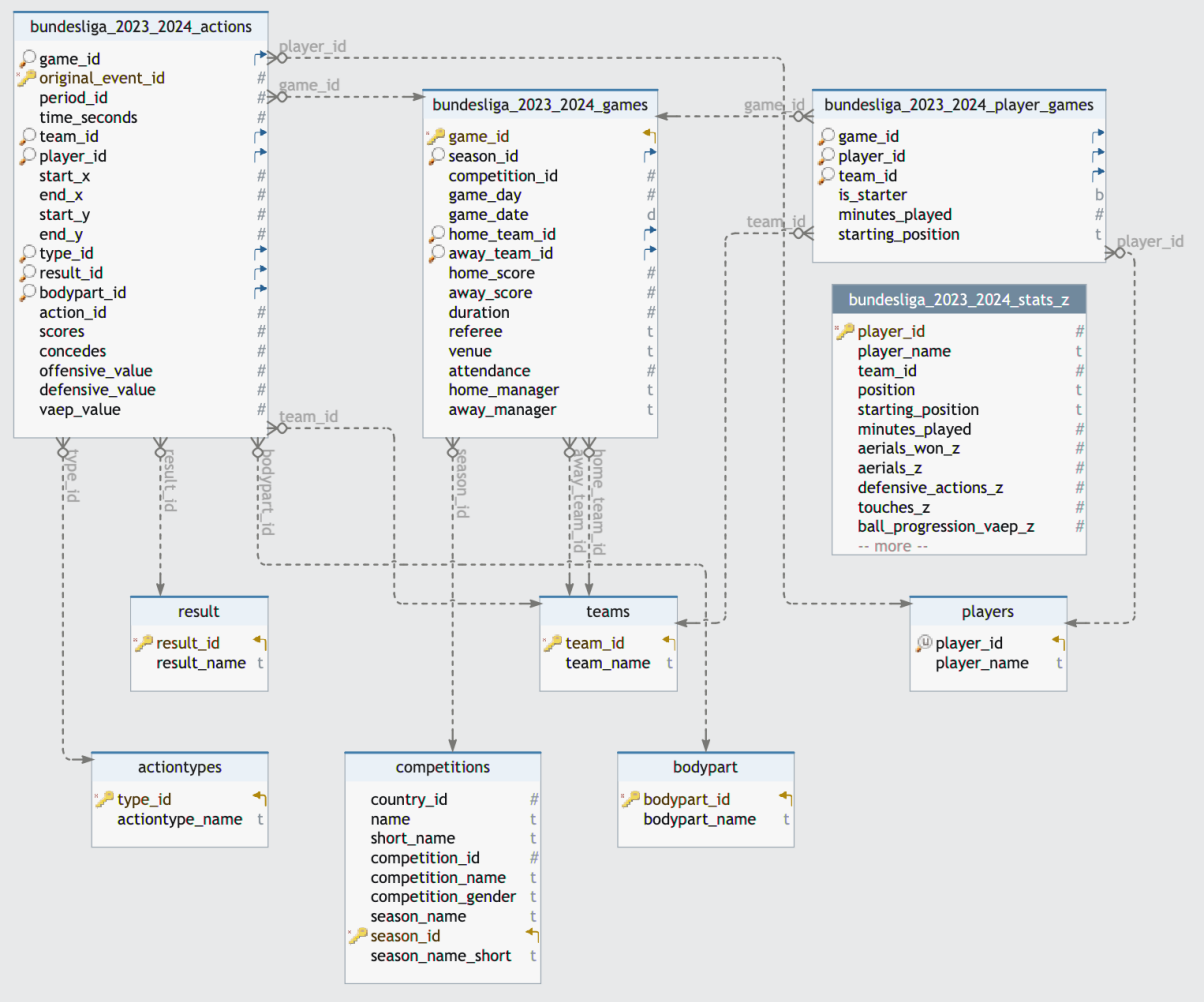
Auf diese oben genannten Tabellen wird in den folgenden Tabellen referenziert:

- `bundesliga_2023_2024_actions`: Enthält Informationen zu allen Aktionen in der Bundesliga-Saison 2023/24. (original_event_id, action_id, game_id, team_id, player_id, result_id, bodypart_id, actiontype_id, start_x, start_y, time_seconds, ...)
- `bundesliga_2023_2024_games`: Enthält Informationen zu allen Spielen in der Bundesliga-Saison 2023/24. (game_id, competition_id, season_id, game_date, home_team_id, away_team_id, home_score, away_score, ...)
- `bundesliga_2023_2024_player_games`: Enthält Informationen zu allen Spielern und den gespielten Minuten, sowie der Startposition für jedes Spiel. (game_id, player_id, team_id, is_starter, starting_position, minutes_played)
- `bundesliga_2023_2024_stats_z`: Enthält Informationen für jeden Spieler zu allen z-Scores der Basisstatistiken und Qualitäten. (player_id, team_id, minutes_played, goals, assists, shots, passes, tackles, interceptions, ...)

Aus diesen Tabellen können dann Abfragen erstellt werden, um die gewünschten Informationen zu erhalten. Aufgrund der Größe der Datenbank und der Anzahl der Tabellen, ist es wichtig, dass die Datenbank gut strukturiert ist und die Abfragen effizient sind. Viele joins können die Performance der Datenbank beeinträchtigen, weshalb ich mich teilweise für eine denormalisierte Datenbank entschieden habe (stats_z). Hieraus ergibt sich in der Praxis eine viel höhere Abfragegeschwindigkeit und der zusätzliche Speicherplatz ist in der heutigen Zeit kein Problem mehr.

11.2 Auszug aus der Datenbank

Hier visuell die Struktur der Datenbank für die Bundesliga-Saison 2023/24:



11.3 Datenbankzugriff - Werte speichern

Der Zugriff auf die Datenbank erfolgt über Python und die Bibliothek `sqlalchemy`. Hier ein Beispiel, wie auf die Datenbank zugegriffen werden kann und neue Werte gespeichert werden. Um Rechenkapazität und Zeit zu sparen, werden nur die neuen Daten in der Tabelle gespeichert. Dazu werden die Primärschlüssel der Tabellen verwendet, um zu überprüfen, ob die Daten bereits in der Datenbank vorhanden sind. Bei Tabellen ohne Primärschlüssel müsste jeder Eintrag mühselig überprüft werden, ob er bereits in der Datenbank vorhanden ist. Da ist es oft schneller alle Daten neu zu übertragen, was jedoch fehleranfälliger ist. Der Connections-String wird in eigenen Dateien gespeichert, um die Zugangsdaten nicht im Code zu speichern und wird dann mit der Funktion `create_connection_string()` geladen.

```
def write_to_sql_efficient(*args) -> None:
    """
    Writes data from a Pandas DataFrame to an SQL table, checking for duplicates␣
↪directly in the database.

    Parameters:
    - dataframe: The DataFrame to be written to the SQL table.
    - table_name: The name of the target table in the SQL database.
    - connection_string: The connection string to the SQL database.
    - primary_key_column: The name of the column to be used as the primary key for␣
↪checking duplicates.
    """
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    # Verbindung zur Datenbank herstellen mit pool_pre_ping=True
    connection_string = create_connection_string()
    engine = create_engine(connection_string, pool_pre_ping=True)

    for dataframe, table_name, primary_key_column in args:
        # SQL-Abfrage vorbereiten, um Duplikate direkt in der Datenbank zu überprüfen
        sql_query = f"SELECT {primary_key_column} FROM {table_name}"
        existing_primary_keys = pd.DataFrame(engine.connect().execute(text(sql_
↪query)))

        # Überprüfen, ob beide DataFrames nicht leer sind
        if not dataframe.empty and not existing_primary_keys.empty:
            # Nur neue Datensätze auswählen, die nicht bereits in der Datenbank␣
↪vorhanden sind
            new_data = pd.merge(dataframe, existing_primary_keys, on=primary_key_
↪column, how='left',
                                indicator=True).query('_merge == "left_only"').drop('_
↪merge', axis=1)
        else:
            new_data = dataframe

        # Wenn es neue Datensätze gibt, diese in die SQL Tabelle schreiben
        if not new_data.empty:
            new_data.to_sql(table_name, engine, index=False, if_exists='append')
            print(f"{len(new_data)} Datensätze wurden erfolgreich in die Tabelle
↪{table_name} eingefügt.")
        else:
            print("Keine neuen Datensätze zum Einfügen gefunden. Tabelle:" + table_
↪name)
```

11.4 Datenbankzugriff - Werte laden

Um Werte aus der SQl Tabelle in ein Dataframe zu laden, kann folgender Code verwendet werden:

```
def load_actions_from_sql(league: str, season: str) -> pd.DataFrame:
    """
    Loads actions from an SQL table.

    Parameters:
    - league: The league of the actions.
    - season: The season of the actions.

    Returns:
        DataFrame: A DataFrame containing the actions.
    """
    # Verbindung zur Datenbank herstellen
    # 'engine' ist ein SQLAlchemy Engine-Objekt, das für die Verbindung zur Datenbank␣
↪verwendet wird
    connection_string = create_connection_string()
    engine = create_engine(connection_string, pool_pre_ping=True)

    # Generate the table name from the league and season parameters
    table_name = f"{league}_{season}_actions"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    # Laden der existierenden Daten aus der SQL-Tabelle
    try:
        sql_query = f"SELECT * FROM {table_name}"
        existing_data = pd.DataFrame(engine.connect().execute(text(sql_query)))
    except Exception as e:
        print(f"Fehler beim Laden der Daten: {e}")
        existing_data = pd.DataFrame()
    return existing_data
```

11.5 SQL Tabellen mit Python automatisch erstellen

Da wir jede beliebe SQL Abfrage mit Python übermitteln können, ist es auch möglich die Tabellen direkt mit Python zu erstellen. Mit jeder Saison und Liga kommen 4 neue Tabellen hinzu, die erstellt werden müssen. Bei 13 Wettbewerben pro Saison kann das schnell zeitaufwendig werden.

Hier als Beispiel eine Funktion um eine Tabelle zu erstellen mit den Spieleraktionen:

```
def create_actions_table(connection, prefix, season):
    table_name = f"{prefix}_{season}_actions"
    # Überprüfen, ob die Tabelle bereits existiert
    check_table_exists = f"""
            SELECT COUNT(*)
            FROM information_schema.tables
            WHERE table_schema = '{DATABASE}'
            AND table_name = '{table_name}';
            """
    result = connection.execute(text(check_table_exists)).scalar()
    if result:
        print(f"Tabelle '{table_name}' existiert bereits.")
        return

    create_sql = f"""
    CREATE TABLE IF NOT EXISTS `{table_name}` (
      `game_id` bigint(20) DEFAULT NULL,
      `original_event_id` bigint(20) NOT NULL,
      `period_id` int(20) DEFAULT NULL,
      `time_seconds` float DEFAULT NULL,
      `team_id` int(20) DEFAULT NULL,
      `player_id` int(11) DEFAULT NULL,
      `start_x` float DEFAULT NULL,
      `end_x` float DEFAULT NULL,
      `start_y` float DEFAULT NULL,
      `end_y` float DEFAULT NULL,
      `type_id` int(20) DEFAULT NULL,
      `result_id` int(20) DEFAULT NULL,
      `bodypart_id` int(20) DEFAULT NULL,
      `action_id` int(20) DEFAULT NULL,
      `scores` float DEFAULT NULL,
      `concedes` float DEFAULT NULL,
      `offensive_value` float DEFAULT NULL,
      `defensive_value` float DEFAULT NULL,
      `vaep_value` float DEFAULT NULL
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
    """
    connection.execute(text(create_sql))
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    # Füge die Indizes hinzu
    alter_sql1 = f"""
    ALTER TABLE `{table_name}`
      ADD PRIMARY KEY (`original_event_id`),
      ADD KEY `fk_game_{table_name}` (`game_id`),
      ADD KEY `fk_team_{table_name}` (`team_id`),
      ADD KEY `fk_player_{table_name}` (`player_id`),
      ADD KEY `fk_type_{table_name}` (`type_id`),
      ADD KEY `fk_result_{table_name}` (`result_id`),
      ADD KEY `fk_bodypart_{table_name}` (`bodypart_id`);
    """
    connection.execute(text(alter_sql1))

    # Füge die Fremdschlüssel hinzu, und stelle sicher, dass sie korrekt auf andere␣
↪Tabellen verweisen
    alter_sql2 = f"""
    ALTER TABLE `{table_name}`
      ADD CONSTRAINT `fk_bodypart_{table_name}` FOREIGN KEY (`bodypart_id`)␣
↪REFERENCES `bodypart` (`bodypart_id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
      ADD CONSTRAINT `fk_game_{table_name}` FOREIGN KEY (`game_id`) REFERENCES `
↪{prefix}_{season}_games` (`game_id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
      ADD CONSTRAINT `fk_player_{table_name}` FOREIGN KEY (`player_id`) REFERENCES␣
↪`players` (`player_id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
      ADD CONSTRAINT `fk_result_{table_name}` FOREIGN KEY (`result_id`) REFERENCES␣
↪`result` (`result_id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
      ADD CONSTRAINT `fk_team_{table_name}` FOREIGN KEY (`team_id`) REFERENCES␣
↪`teams` (`team_id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
      ADD CONSTRAINT `fk_type_{table_name}` FOREIGN KEY (`type_id`) REFERENCES␣
↪`actiontypes` (`type_id`) ON DELETE NO ACTION ON UPDATE NO ACTION;
    """
    connection.execute(text(alter_sql2))
    print(table_name + ' erstellt.')
```

Und schließlich die Funktion um alle Tabellen zu erstellen. Hierbei wird die Tabelle `competitions` ausgelesen und für jede Saison und Liga aus der Datei die Tabellen erstellt.

```
def create_database_tables():
    competitions = pd.read_json('../competitions.json')

    connection_string = db.db_connection.create_connection_string()

    engine = create_engine(connection_string)

    with engine.connect() as connection:
        connection.execute(text('SET FOREIGN_KEY_CHECKS=0;'))

        for index, row in competitions.iterrows():
            prefix = row['db_prefix']
            season = row['season_name'].replace("-", "_")
            create_tables(connection, prefix, season, create_vaep=False, create_
↪player_games=False, create_games=False,
                          create_actions=False, create_player_statistics_z=True)

        connection.execute(text('SET FOREIGN_KEY_CHECKS=1;'))
```

Die Einträge in der competitions.json Datei sehen dabei wie folgt aus. Hierbei wird `db_prefix` benötigt, um die

Tabellen zu erstellen.


```
[
  {
    "country_id":81,
    "name":"Germany",
    "short_name":"GER",
    "competition_id":3,
    "competition_name":"Bundesliga",
    "competition_gender":"Male",
    "season_name":"2023-2024",
    "season_id":23243,
    "season_name_short":"2324",
    "db_prefix":"bundesliga"
  }
]
```

KAPITEL 12

Etwas Statistik

Bisher habe ich jede einzelne Spielaktion bewertet und aus den summierten Spielaktionen für jeden Spieler eine Bewertung für jeden Spieler für einen bestimmten Zeitraum abgeleitet (Ein Spiel, die Hinrunde, eine Saison …). Ebenfalls kann ich den Datensatz nun einfach nach actiontype oder bodypart filtern. Selbst nach xy-Koordinaten, um zu sehen, in welchen Bereichen auf dem Spielfeld ein Spieler besser oder schlechter performt.

Für das Spielerscouting können diese rohen Daten jedoch recht umständlich sein. Ein Scout / Entscheidungsträger möchte wissen, wie ein Fußballspieler in einer bestimmten Qualität abschneiden. Um Qualitäten zu berechnen, brauche ich aber zunächst Werte, die ich miteinander vergleichen kann. Hier kommen Z-Scores ins Spiel.

12.1 Z-Scores

Z-Scores sind standardisierte Werte, die es ermöglichen, Daten miteinander zu vergleichen, die in unterschiedlichen Einheiten gemessen werden. Sie zeigen an, wie viele Standardabweichungen ein Wert von einem Mittelwert entfernt ist. Ein Z-Score von 0 bedeutet, dass der Wert genau dem Mittelwert entspricht, während ein Z-Score von 1 bedeutet, dass der Wert eine Standardabweichung über dem Mittelwert liegt.

Beispiel: Wenn ich eine Qualität berechnen möchte, die sich aus folgenden Werten zusammensetzt:

- Tore mit 50%
- Pässe in den Strafraum mit 20%
- Erfolgreiche Dribblings mit 30%

Angenommen die absoluten Werte für einen Spieler sind:

- Tore: 10
- Pässe in den Strafraum: 270
- Erfolgreiche Dribblings: 90

Dann ergibt sich daraus ein Score von 10 * 0.5 + 270 * 0.2 + 90 * 0.3 = **86**

Wenn sich die Anzahl der Tore verdoppelt ist sein Score 20 * 0.5 + 270 * 0.2 + 90 * 0.3 = **91** Verdoppelt sich die Anzahl erf. Dribblings, ist sein Score 10 * 0.5 + 270 * 0.2 + 180 * 0.3 = **113**

Hieraus ergibt sich, dass obwohl sich der Wert der Tore verdoppelt hat, der Score nur um 5 Punkte gestiegen ist. Der Score bei Verdopplung der erf. Dribblings ist jedoch um 27 Punkte gestiegen. Das würde bedeutet, dass die Qualität der Dribblings wichtiger ist als die Qualität der Tore. Wir möchten aber, dass die Tore einen höheren Einfluss haben (50% vs. 30%). Deshalb ist es notwendig die Werte in Form von z-Scores zu standardisieren.

Also berechne ich für alle Werte Z-Scores und kann dann die Werte miteinander vergleichen. Und daraus dann die Qualität berechnen.

12.2 Berechnung

Die Berechnung eines Z-Scores erfolgt in drei Schritten:

1. Berechnung des Mittelwerts der Datenreihe.
2. Berechnung der Standardabweichung der Datenreihe.
3. Berechnung des Z-Scores für jeden Wert in der Datenreihe.

12.3 Positionsabhängige Z-Scores

Achtung! Ich berechne die Z-Werte positionsabhängig. Das bedeutet, dass ich für jeden Spieler und jede Position die Z-Werte berechne. Jede Position bildet quasi eine eigene Gruppe von Daten. Die Positionen ergeben sich aus den Startposition der Spieler für jedes Spiel. Ein Spieler wird der Position zugeordnet, für die er die meisten Starteinsätze hatte. Mit genaueren Daten, die auch Positionswechsel während des Spiels berücksichtigen, könnte man die Positionen abhängig der gespielten Minuten bestimmen. Die einzelnen Positionen werden ebenfalls nochmal gruppiert in folgende übergeordnete Positionen zusammengefasst:

- Abwehrspieler
- Außenverteidiger
- Mittelfeldspieler
- Flügelspieler
- Angreifer

KAPITEL 13

Spielerqualitäten entwickeln

Bisher habe ich jede einzelne Spielaktion bewertet und aus den summierten Spielaktionen für jeden Spieler eine Bewertung für jeden Spieler für einen bestimmten Zeitraum abgeleitet (Ein Spiel, die Hinrunde, eine Saison …). Ebenfalls kann ich den Datensatz nun einfach nach actiontype oder bodypart filtern. Selbst nach xy-Koordinaten, um zu sehen, in welchen Bereichen auf dem Spielfeld ein Spieler besser oder schlechter performt.

Für das Spielerscouting können diese rohen Daten jedoch recht umständlich sein. Ein Scout möchte wissen, wie ein Fußballspieler in einer bestimmten Qualität abschneidet. Eine Qualität ist dabei eine gewichtete Zusammensetzung aus mehreren Statistiken. Und diese Statistiken lassen sich mithilfe unserer Rohdaten in der MySQL Tabelle ganz simpel berechnen.

13.1 Basisstatistiken

Unsere berechneten VAEP-Werte haben entscheidenden Anteil an der Berechnung der Basisstatistiken und somit auch großen Einfluss auf die Qualität. Die Basisstatistiken für die Berechnung der Qualitäten errechnet sich wie folgt:

Basisstatistik	Berechnungsmethode
Aerials	Zählt die Anzahl der Kopfballduelle, an denen ein Spieler teilgenommen hat.
Aerials Won	Zählt die Anzahl der gewonnenen Kopfballduelle.
Aerials Won Defensive Value	Berechnet den defensiven Wert für gewonnene Kopfballduelle.
Aerials Won Offensive Value	Berechnet den offensiven Wert für gewonnene Kopfballduelle.
Assists	Zählt die Anzahl der Assists, indem überprüft wird, ob der nächste Spielzug zu einem T
Attacking Aerials Won	Zählt die Anzahl der gewonnenen offensiven Kopfballduelle.
Attacking Aerials Won Offensive Value	Berechnet den offensiven Wert für gewonnene offensive Kopfballduelle.
Ball Progression Count	Zählt die Anzahl der erfolgreichen Vorwärtspässe.
Ball Progression VAEP	Berechnet die Summe der VAEP-Werte für erfolgreiche Vorwärtspässe.
Ball Recoveries	Zählt die Anzahl der erfolgreichen Ballrückeroberungen.
Ball Runs VAEP	Berechnet den VAEP-Wert für Ballläufe.
Box Entries	Zählt die Anzahl der Ballaktionen, die in den Strafraum führen.
Carries Offensive Value	Berechnet den offensiven Wert für Ballläufe.
Counterpressing Interceptions	Zählt die Anzahl der erfolgreichen Abfangaktionen durch Gegenpressing.
Counterpressing Recoveries	Zählt die Anzahl der erfolgreichen Ballrückeroberungen durch Gegenpressing.
Creative Passes Count	Zählt die Anzahl der kreativen Pässe, die zu einer Torschussvorlage führen.

Fortsetzung auf der nächsten Seite

Tab. 1 – Fortsetzung der vorherigen Seite

Basisstatistik	Berechnungsmethode
Crosses VAEP	Berechnet die Summe der VAEP-Werte für erfolgreiche Flanken.
Deep Completions	Zählt die Anzahl der erfolgreichen Pässe in den Strafraum.
Deep Runs VAEP	Berechnet den VAEP-Wert für tiefe Läufe im letzten Drittel.
Defensive Actions	Zählt die Anzahl der defensiven Aktionen (z.B. Tacklings, Blocken).
Defensive Actions Defensive Value	Berechnet den defensiven Wert für defensive Aktionen.
Defensive Aerials Won	Zählt die Anzahl der gewonnenen defensiven Kopfballduelle.
Defensive Aerials Won Defensive Value	Berechnet den defensiven Wert für gewonnene defensive Kopfballduelle.
Defensive Intensity	Zählt die Anzahl der erfolgreichen Defensivaktionen.
Dribbles	Zählt die Anzahl der Dribblings.
Dribbles Success	Berechnet die Erfolgsquote der Dribblings.
Dribbles VAEP	Berechnet die Summe der VAEP-Werte für Dribblings.
Goals	Zählt die Anzahl der Tore.
Goals VAEP	Berechnet die Summe der VAEP-Werte für Tore.
Headed Plays	Zählt die Anzahl der Kopfballaktionen.
High Turnovers	Zählt die Anzahl der Ballverluste im letzten Drittel.
Interceptions	Zählt die Anzahl der erfolgreichen Abfangaktionen.
Link-Up Plays Attack	Zählt die Anzahl der erfolgreichen Verbindungsspiele, die zu einem Schuss führen.
Long Ball Receptions	Zählt die Anzahl der erfolgreichen Ballannahmen bei langen Pässen.
Losses	Zählt die Anzahl der Ballverluste.
Passes	Zählt die Anzahl der erfolgreichen Pässe ins letzte Drittel des Spielfelds.
Passes into Final Third Count	Zählt die Anzahl der erfolgreichen Pässe ins letzte Drittel des Spielfelds.
Passes into Final Third VAEP	Berechnet die Summe der VAEP-Werte für erfolgreiche Pässe ins letzte Drittel des Spi
Passes in Final Third Count	Zählt die Anzahl der erfolgreichen Pässe innerhalb des letzten Drittels des Spielfelds.
Passes VAEP	Berechnet die Summe der VAEP-Werte für erfolgreiche Pässe.
Penalty Area Receptions	Zählt die Anzahl der erfolgreichen Ballannahmen im Strafraum.
Possessions Won	Zählt die Anzahl der gewonnenen Ballbesitze.
Pressure Resistance	Zählt die Anzahl der erfolgreichen Aktionen unter Druck.
Shot Conversion	Berechnet die Erfolgsquote der Schüsse.
Touches	Zählt die Anzahl der Ballkontakte.
Touches in Box	Zählt die Anzahl der Ballkontakte im Strafraum.
VAEP Buildup	Berechnet die Summe der VAEP-Werte für erfolgreiche Aufbauspielaktionen.
VAEP Created with Passes	Berechnet die Summe der VAEP-Werte für Pässe, die zu einem Tor führen.
VAEP per Shot	Berechnet den VAEP-Wert pro Schuss.
VAEP Shots	Berechnet die Summe der VAEP-Werte für Schüsse.
xA	Berechnet die Summe der erwarteten Assists (xA) für Pässe.
xG	Berechnet die erwarteten Tore (xG).
xG Created with Dribbles	Berechnet die Summe der erwarteten Tore (xG) durch Dribblings.

13.2 Qualitäten definieren

Die Definition der Qualitäten ist hierbei völlig frei. Hier kann jeder Verein seine eigenen Qualitäten definieren. In Anlehnung an typische Fußballqualitäten und dem Scoutingtool „twelve gpt“ habe ich folgende Qualitäten vordefiniert:

tabularytabulary

Qualität
Active Defense
Aerial Threat
Box Threat
Composure
Defensive Heading
Dribble
Effectiveness
Finishing
Hold-Up Play
Involvement
Intelligent Defense
Passing Quality
Poaching
Pressing
Progression
Providing Teammates
Run Quality

KAPITEL 14

Visualisierung mit Streamlit

In diesem Abschnitt werde ich die Visualisierung der Daten mit Streamlit vorstellen. Streamlit ist ein Open-Source-Framework, das es ermöglicht, interaktive Webanwendungen direkt aus Python-Scripts zu erstellen. Streamlit eignet sich für eine schnelle und einfache Visualisierung von Daten und Modellen und ist daher ideal für Prototypen und kleinere Projekte. Für größere Projekte und Anwendungen empfehle ich jedoch andere Frameworks wie Flask oder Django.

Den ganzen Code der Streamlit-Anwendung findest du in meinem GitHub-Repository / datenflanke

14.1 Installation

Die Installation ist sehr einfach. Es wird nur Python benötigt. Mit pip kann Streamlit installiert werden:

```
pip install streamlit
```

Zum Starten der Anwendung wird folgender Befehl ausgeführt:

```
streamlit run app.py
```

Das war es schon. Streamlit startet einen lokalen Server und öffnet den Standardbrowser mit der Anwendung.

14.2 Beispiel: Spielerbewertung

Die Grundlegenden Prinzipien von Streamlit kannst du in der Dokumentation nachlesen.

14.2.1 Code Spielerbewertung.py

Beispielhaft für mein Projekt möchte ich dir die Seite der Spielerbewertung zeigen:

Zuerst die Imports:

```
import streamlit as st
import utils.helpers as helpers
import utils.plots as plots
from utils.chatbot import get_player_evaluation, get_player_evaluation_german
```

Dann lade ich die Daten und zwar so, dass die Daten beim ersten Aufruf der App aus der Datenbank geladen werden und dann im Cache gespeichert werden. So ist sichergestellt, dass die Daten nicht bei jedem Aufruf der App neu geladen werden müssen.

```
# Daten laden
dataframe = helpers.preload_data()
```

Die Funktion zum Laden der Daten ist in der Datei `utils/helpers.py` definiert:

```
@st.cache_data
def preload_data():
    # Load all data from the database
    leagues = [
        "bundesliga",
        "premier_league",
        "laliga",
        "ligue_1",
        "seria_a",
        "champions_league",
        "europa_league",
        "bundesliga2",
        "eredivisie",
        "jupiler_pro_league",
        "championship",
        "liga_portugal",
        "super_lig",
    ]

    seasons = ["2022_2023", "2023_2024"]

    # Initialize an empty list to store the data
    data_list = []

    for season in stqdm.stqdm(seasons):
        for league in stqdm.stqdm(leagues, leave=False):
            data = db.dataframe_from_sql(league, season, create_connection_string())
            data_list.append({'league': league, 'season': season, 'data': data})
            print("Data preloaded for", league, season)

    # Create a new dataframe with columns league, season, and data
    preloaded_data_df = pd.DataFrame(data_list)

    return preloaded_data_df
```

Nun wird die Sidebar mit den Filteroptionen erstellt. Der Nutzer kann hier die Liga, die Saison, die Position, den Spieler und die Qualität auswählen. Über die Funktion get_data_by_league_and_season wird der Dataframe für die gewählte Liga und Saison aus dem vorgeladenen Dataframe mit allen Saisons extrahiert. Damit können die Spieler für die gewählte Liga und Saison ausgewählt werden.

```
def get_data_by_league_and_season(preloaded_data_df, league, season):
    # Filter the dataframe for the given league and season
    filtered_df = preloaded_data_df[(preloaded_data_df['league'] == league) &␣
↪(preloaded_data_df['season'] == season)]

    if not filtered_df.empty:
        # Assuming there's only one row per league-season combination
        return filtered_df.iloc[0]['data']
    else:
        # Handle case where no data is found for the given league and season
        print(f"No data found for league: {league}, season: {season}")
        return None
```

Die Sidebar wird mit den Filteroptionen erstellt:

```
# Header with logo and app name placeholder
st.sidebar.image('images/logo.jpg', use_column_width=True)

# Dictionary für Filteroptionen erstellen
leagues = helpers.create_leagues_dict_with_flags()
seasons = helpers.create_seasons_dict()
quality = helpers.create_quality_dict()

# Sidebar-Einstellungen
st.sidebar.header('Spieler auswählen:', divider=True)
selected_league_display = st.sidebar.selectbox('Wettbewerb', options=list(leagues.
↪keys()))
selected_season_display = st.sidebar.selectbox('Saison', options=list(seasons.keys()))

# Zugriff auf die tatsächlichen Werte
selected_league = leagues[selected_league_display]
selected_season = seasons[selected_season_display]

# Dataframe welcher geladen werden soll
data = helpers.get_data_by_league_and_season(dataframe, selected_league, selected_
↪season)
# Formular zur Spielerauswahl in der Seitenleiste
position = st.sidebar.selectbox('Position', ['Abwehrspieler', 'Außenverteidiger',
↪'Mittelfeldspieler', 'Flügelspieler', 'Angreifer'])
# Spieler aus Dataframe laden und Filter für Position
players =sorted(data[data['position'] == position]['player_name'])
# Auswahl in der Sidebar
player = st.sidebar.selectbox('Spieler', players)
# Auswahl der Qualität
selected_quality_display = st.sidebar.selectbox('Qualität',  options=list(quality.
↪keys()))
selected_quality = quality[selected_quality_display]
```

Die Dictionaries leagues, seasons und quality werden in der Datei `utils/helpers.py` definiert. Sie enthalten die Namen der Ligen, Saisons und Qualitäten, die in der Sidebar angezeigt werden sollen, damit nicht die internen Namen der Variabel aus der Datenbank verwendet werden müssen. Der Key ist der Name, der dem Nutzer angezeigt wird und der Value ist der interne Name, der in der Datenbank verwendet wird. Beispielhaft für die Saisons:

```
def create_seasons_dict():
    seasons = {
        "2023/24": "2023_2024",
        "2022/23": "2022_2023"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    }
    return seasons
```

Der Hauptteil der Anwendung wird mit dem Streamlit-Befehl `st.header` erstellt. Hier kommt der eigentliche Inhalt der Seite. In diesem Fall die Bewertung eines Spielers mit einem Plot und der Beschreibung der Bewertung.

```
# Main content area
st.header('Bewertung von ' + player + ' ' + selected_quality_display, divider=True)

# Variablen übergeben die geplottet werden sollen
attributes = helpers.get_attributes_details(selected_quality, position)


# Plot erstellen
chart = plots.create_player_plot(data, attributes, player, position, selected_league_
↪display, selected_season_display, selected_quality_display)
st.altair_chart(chart, use_container_width=True)


# Beschreibung der Spielerbewertung
with st.spinner('Schreibe Spielerbewertung...'):
    st.write('')
    evaluation = get_player_evaluation(player, attributes, data)
    st.success(evaluation)

    # Beschreibung der Spielerbewertung
with st.spinner('Schreibe Spielerbewertung...'):
    st.write('')
    evaluation = get_player_evaluation_german(player, attributes, data)
    st.success(evaluation)

# Footer
st.markdown('---')  # This creates a horizontal line
st.write('This Webapp was created by Christoph Debowiak - [chrisdebo @GitHub](https://
↪github.com/chrisdebo)')
```

Die Funktion get_attributes_details wird in der Datei `utils/helpers.py` definiert und gibt die Attribute zurück, die für die gewählte Qualität und Position relevant sind. Je nach Position und Qualität können die Attribute variieren. Für Angreifer sind andere Qualitäten wichtiger als für Abwehrspieler.

```
def get_attributes_summary(position) -> list:
    if position == 'Abwehrspieler':
        a = ['involvement_z', 'progression_z', 'composure_z', 'aerial_threat_z',
↪'defensive_heading_z',
             'active_defense_z', 'intelligent_defense_z']
    elif position == 'Mittelfeldspieler':
        a = ['involvement_z', 'progression_z', 'passing_quality_z', 'providing_
↪teammates_z', 'box_threat_z',
             'active_defense_z', 'intelligent_defense_z', 'effectiveness_z']
    elif position == 'Angreifer':
        a = ['involvement_z', 'pressing_z', 'run_quality_z', 'finishing_z', 'poaching_
↪z',
             'aerial_threat_z', 'providing_teammates_z', 'hold_up_play_z']
    elif position == 'Flügelspieler':
        a = ['involvement_z', 'passing_quality_z', 'providing_teammates_z', 'dribble_z
↪', 'box_threat_z',
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

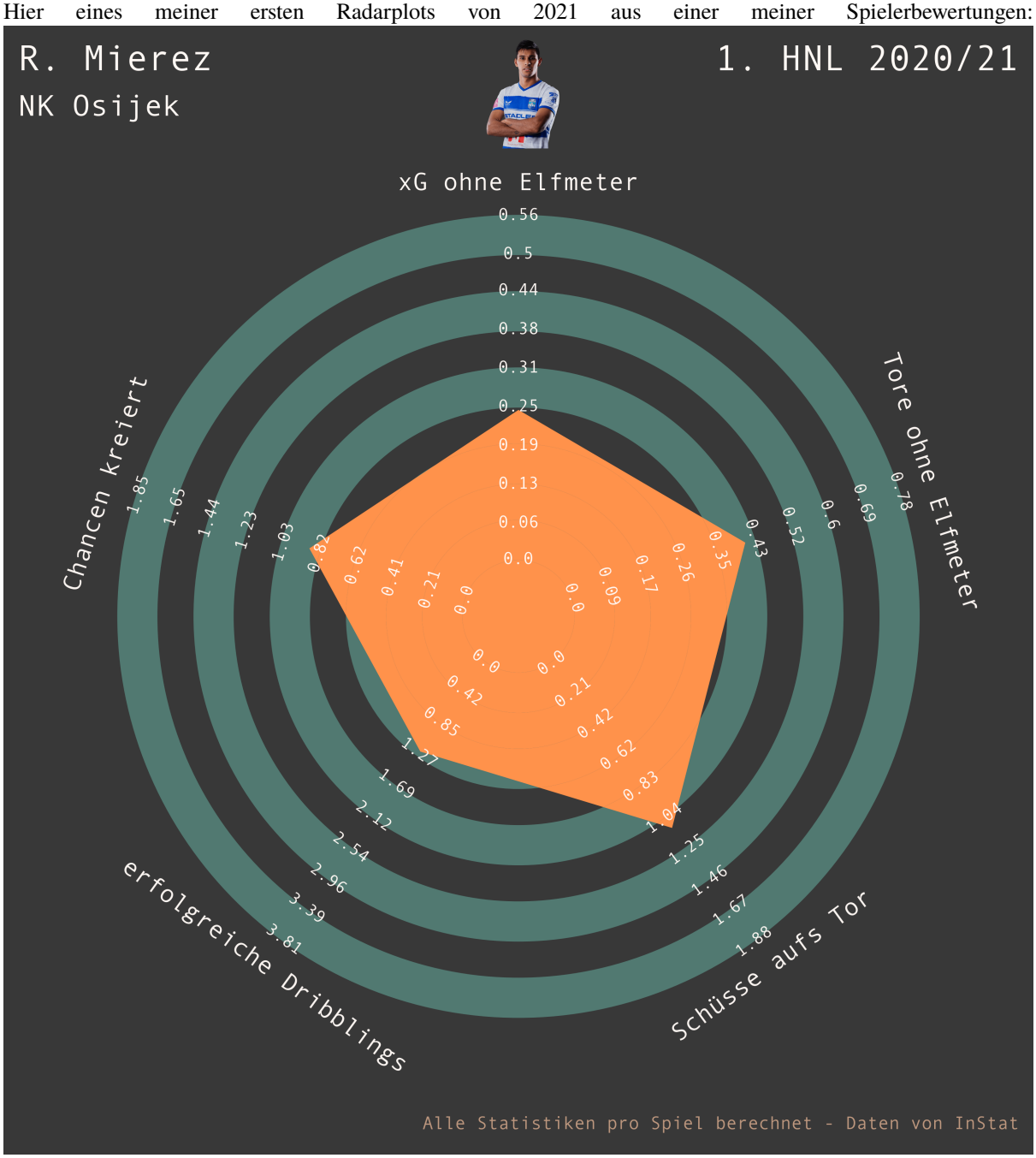
```
            'finishing_z', 'run_quality_z', 'pressing_z', 'effectiveness_z']
    elif position == 'Außenverteidiger':
        a = ['involvement_z', 'progression_z', 'passing_quality_z', 'providing_
↪teammates_z', 'run_quality_z',
            'active_defense_z', 'intelligent_defense_z']
    else:
        a = []
    return a
```

14.3 Visualisierung der Ergebnisse: Plots

Es gibt unzählige Möglichkeiten Daten zu visualisieren. Im Fußballbereich haben sich dabei Radar- und Distributionsplots als sehr beliebt erwiesen.

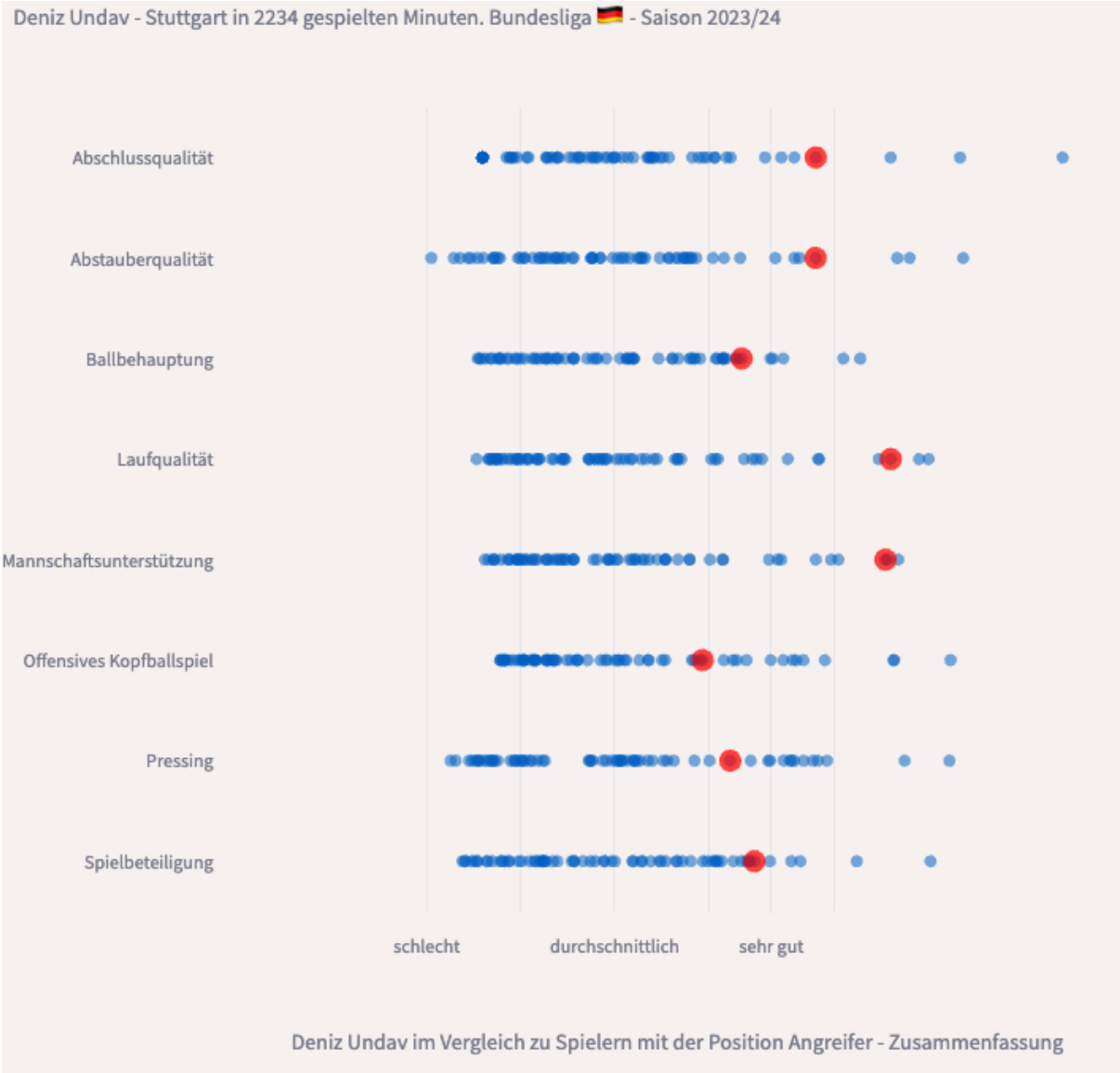
14.3.1 Radarplots

Hier eines meiner ersten Radarplots von 2021 aus einer meiner Spielerbewertungen:



14.3.2 Distributionsplots

Mittlerweile finde ich Distributionsplots (manchmal auch Verteilungsplot oder Beeplot genannt) vorteilhafter, da sie mehr Informationen auf einmal liefern. Der Nutzer kann direkt sehen, wie ein Spieler im Verhältnis zu anderen Spielern performt hat.



14.3.3 Codebeispiel Plots

Die Plots werden in der Datei `utils/plots.py` definiert. Hier ein Beispiel für Distributionplot. Die Funktion `create_player_plot` erstellt einen Plot für einen spezifischen Spieler, der die Attribute des Spielers mit anderen Spielern in der gleichen Position vergleicht. Es wird das Dataframe mit allen Daten der jeweiligen Saison und des Wettbewerbs benötigt. Es wird der Spielername, die Position, der Wettbewerb, die Saison und die Qualität benötigt, um den Plot zu erstellen. Standardmäßig wird als Qualität „Zusammenfassung“ ausgewählt, was dann positionsabhängige Qualitäten übergibt. Ausgegeben wird ein Altair-Chart-Objekt, das dann in der Streamlit-Anwendung angezeigt wird.

Zuerst werden die Labels für die X- und Y-Achse erstellt. Dann wird das Dataframe für den ausgewählten Spieler und die Spieler auf der gleichen Position gefiltert.

```
def create_player_plot(data: pd.DataFrame, attributes: list[str], player_name: str,␣
↪position: str, league: str, season: str, quality: str) -> alt.Chart:
    """
    Creates a plot for a specific player, comparing their attributes to other players␣
↪in the same position.

    Parameters:
    data (pd.DataFrame): The dataset containing player information.
    attributes (List[str]): A list of attributes to be used for evaluating the player.
    player_name (str): The name of the player.
    position (str): The position of the player.
    league (str): The league in which the player plays.
    season (str): The season for which the data is being analyzed.
    quality (str): The quality metric used for player evaluation.

    Returns:
    alt.Chart: An Altair chart object representing the player plot.
    """
    x_labels = utils.helpers.create_x_labels()
    y_labels = utils.helpers.create_y_labels()

    # Daten für den ausgewählten Spieler filtern
    player_data = data[(data['player_name'] == player_name) & (data['position'] ==␣
↪position)]

    # Daten für alle Spieler auf der gleichen Position filtern
    position_data = data[data['position'] == position]
```

Nun werden die Daten die geplottet werden sollen, in ein Dataframe geladen, da das Altair-Framework erlaubt die Daten direkt aus einem Dataframe zu plotten. Zuerst werden die Daten dafür vorbereitet. Es werden alle Attribute/Qualitäten für alle Spieler iteriert. Dabei wird der Rang des Spielers und die Gesamtanzahl der Spieler berechnet und in einer neuen Spalte gespeichert. Alle Informationen werden zuerst in der Liste plot_data_list aneinandergereiht und später in das Dataframe plot_data überführt.

```
    # Erstelle einen DataFrame für das Plotten
    plot_data_list = []
    for idx, attribute in enumerate(attributes):
        temp_data = pd.DataFrame({
            'Player': position_data['player_name'],
            'Value': position_data[attribute].values.flatten(),  # Sicherstellen,␣
↪dass die Daten 1-dimensional sind
            'Attribute': attribute,
            'Y': y_labels[attribute]  # Y-Wert als benutzerdefiniertes Label zuweisen
        })
        temp_data['Rank'] = temp_data['Value'].rank(ascending=False, method='min')
        temp_data['Total'] = len(temp_data)
        temp_data['Rank_Display'] = temp_data.apply(lambda row: f"{int(row['Rank'])}/
↪{int(row['Total'])}", axis=1)
        plot_data_list.append(temp_data)
    plot_data = pd.concat(plot_data_list)
```

Hier wird der erste Chart erzeugt mit allen Spielern und den Attributen. Der Chart wird mit einem Kreis für jeden Spieler und einem Tooltip erstellt, der die Spielerinformationen anzeigt. Die Datengrundlage ist plot_data. Jede Qualität ist dabei wie ein eigener Chart. Der z-Score wird auf der x-Achse dargestellt. Der y-Wert ist das Attribut. Der Tooltip zeigt den Spieler und den Rang an.

```
    # Erstelle den Altair-Plot
    base = alt.Chart(plot_data).mark_circle(size=60, opacity=0.5).encode(
        x=alt.X('Value:Q', scale=alt.Scale(domain=(-3, 4)),
                axis=alt.Axis(title=f"{player_name} im Vergleich zu Spielern mit der␣
↪Position {position} - {quality}",
                              titleY=75, titleAlign='center', values=list(x_labels.
↪keys()),
                              labelExpr="datum.value == -1.5 ? 'schlecht' : datum.
↪value == -0.75 ? 'unterdurchschnittlich' : datum.value == 0 ? 'durchschnittlich' :␣
↪datum.value == 0.75 ? 'gut' : datum.value == 1.25 ? 'sehr gut' : datum.value == 1.
↪75 ? 'überragend' : ''")),
        y=alt.Y('Y:N', axis=alt.Axis(
            title=f'{player_data["player_name"].values[0]} - {player_data["team_name
↪"].values[0]} in {player_data["minutes_played"].values[0]} gespielten Minuten.
↪{league} - Saison {season} ',
            titleAngle=0, titleX=100, titleY=-50, labelAngle=0, labelAlign='right',␣
↪labelLimit=175)),
        # Y-Achse mit Attributnamen und Beschriftung
        tooltip=['Player', 'Rank_Display']
    ).properties(
        width=700,
        height=85 * len(attributes)
    )
```

Nun wird der ausgewählte Spieler hervorgehoben. Dafür wird ein neuer Chart erstellt, der nur den ausgewählten Spieler hervorhebt. Der Chart wird mit einem roten Kreis erstellt und zeigt den Spieler und den Rang an.

```
    # Hervorhebung des ausgewählten Spielers
    highlight_data_list = []
    for idx, attribute in enumerate(attributes):
        temp_data = pd.DataFrame({
            'Player': [player_name],
            'Value': [player_data[attribute].values.flatten()[0]],
            'Attribute': [attribute],
            'Y': [y_labels[attribute]]
        })
        temp_data['Rank'] = \
            plot_data[(plot_data['Attribute'] == attribute) & (plot_data['Player'] ==␣
↪player_name)]['Rank'].values[0]
        temp_data['Total'] = \
            plot_data[(plot_data['Attribute'] == attribute) & (plot_data['Player'] ==␣
↪player_name)]['Total'].values[0]
        temp_data['Rank_Display'] = f"{int(temp_data['Rank'][0])}/{int(temp_data[
↪'Total'][0])}"
        highlight_data_list.append(temp_data)
    highlight_data = pd.concat(highlight_data_list)

    highlight = alt.Chart(highlight_data).mark_circle(size=200, color='red').encode(
        x='Value:Q',
        y=alt.Y('Y:N', axis=alt.Axis(labels=True)),
        tooltip=['Player', 'Rank_Display']
    )
```

Zuletzt werden die beiden Charts kombiniert und die Farben der Skalen unabhängig voneinander gesetzt.

```
    # Kombiniere den Basis-Chart und Highlight-Chart
    chart = alt.layer(base, highlight).properties(
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        #background='#f0f0f0'  # Hintergrundfarbe setzen
    ).resolve_scale(
        color='independent'
    )

    return chart
```

14.4 Large Language Models

Durch die aktuelle Entwicklung von Large Language Models wie GPT etc. ist es möglich die Daten auch direkt in einer benutzerfreundlichen Sprache auszugeben. Durch die Entwicklung von dafür generierten Prompts, wird dem Nutzer die Möglichkeit gegeben, die Daten in natürlicher Sprache zu erhalten. Diese Berichte geben einen ersten Überblick über den Spieler und können als Grundlage für weitere Analysen dienen. Für Implementierung solcher KI generierten Texte habe ich mich an der Idee von David Sumpter und seinem Projekt „twelve“ orientiert, wie er es in dem Vortrag dazu in der twelve-Community mit dem Titel „Using large language models for scouting“ vorgestellt hat.

14.4.1 Codebeispiel Textausgabe

Ich übergebe den Spielernamen, die Qualitäten die beschrieben werden sollen und das Dataframe mit den Daten. Die Funktion gibt dann eine Beschreibung des Spielers zurück als String. Wichtig ist hier, dass wir dem LLM einen vernünftigen prompt übergeben. Dieser setzt sich hier aus mehreren Informationen zusammen.

Zuerst werden alle Qualitäten des Spielers in einer Schleife durchlaufen und die Beschreibung für jede Qualität erstellt. Diese Beschreibungen werden dann in der Variable player_description gespeichert. Da das LLM mit z-Scores nicht viel anfangen kann, muss der z-Score in eine Beschreibung umgewandelt werden. Dafür wird die Funktion describe_level_german verwendet, die den z-Score in eine Beschreibung umwandelt.

```
def describe_level_german(z_score: float) -> str:
    if z_score >= 1.5:
        description = "überragend"
    elif z_score >= 1:
        description = "ausgezeichnet"
    elif z_score >= 0.5:
        description = "gut"
    elif z_score >= -0.5:
        description = "durchscnitlich"
    elif z_score >= -1:
        description = "unterdurchschnittlich"
    else:
        description = "schlecht"

    return description
```

```
def get_player_evaluation_german(player_name: str, attributes: list, data: pd.
↪DataFrame) -> str:
    """
    Generates an evaluation of a player based on their attributes and data.

    Parameters:
    player_name (str): The name of the player.
    attributes (list): A list of attributes to be used for evaluating the player.
    data (pd.DataFrame): The DataFrame containing the player data.
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    Returns:
    str: A description of the player based on their attributes and data.
    """
    try:
        player_data = data[data['player_name'] == player_name].iloc[0]
        position = player_data['position']
    except IndexError:
        return f"No data found for player {player_name}"

    description = f"Player: {player_name}\n"

    player_description = ""
    for attribute in attributes:
        z_score = player_data.get(attribute, None)
        if z_score is None:
            return f"Attribute {attribute} not found for player {player_name}"
        level = describe_level_german(z_score)
        player_description += f"Wenn es um die Fähigkeit {attribute} geht, dann ist
↪{player_name} {level}.\n"
```

Nun wird dem Chatbot eine Rolle zugewiesen:

```
messages = [
        {"role": "system", "content": f"Du bist ein Fußballscout aus Deutschland \
            Du lieferst prägnante und auf den Punkt gebrachte Zusammenfassungen von␣
↪Fußballspielern \
            basierend auf Daten. Du sprichst und benutzt für den Fußball typische␣
↪Sprache. \
            Du nutzt die Informationen aus den dir gegebenen Daten und Antworten \
            aus früheren 'user/assistant' Paaren, um Zusammenfassungen über die␣
↪Spieler zu erstellen. \
            Deine aktuelle Aufgabe besteht darin einen bestimmten Spieler auf der␣
↪Position {position} zu beschreiben."},
        {"role": "user", "content": "Was meinst du genau mit Fußball?"},
        {"role": "assistant", "content": "Ich meine die Sportart Fußball, welche in␣
↪Europa und in Deutschland die beliebteste und bekannteste Sportart ist.  \
         "}]
```

Und schließlich wird der gesamte prompt zusammengeführt und an das LLM übergeben. Die seed Variable sorgt dafür, dass die Antwort immer gleich bleibt, wenn der gleiche seed verwendet wird. Die Temperatur bestimmt, wie kreativ die Antwort ist. Je höher die Temperatur, desto kreativer die Antwort. Weitere Informationen dazu findest du in der OpenAI Dokumentation.

```
    start_prompt ="Hier findest du eine Beschreibung einiger Fähigkeiten des␣
↪Spielers:\n\n"
    end_prompt = f"\n Nutze die zur Verfügung stehenden Daten und mache eine␣
↪Zusammenfassung über den Spieler (nicht mehr als drei Sätze) und spekuliere über␣
↪die Rolle, welche dieser Spieler in einem Team haben könnte aufgrund dieser␣
↪Fähigkeiten: {', '.join(attributes)}"

    #Now ask about current player

    the_prompt = start_prompt + player_description + end_prompt
    user={"role": "user", "content": the_prompt}
    messages = messages + [user]
```

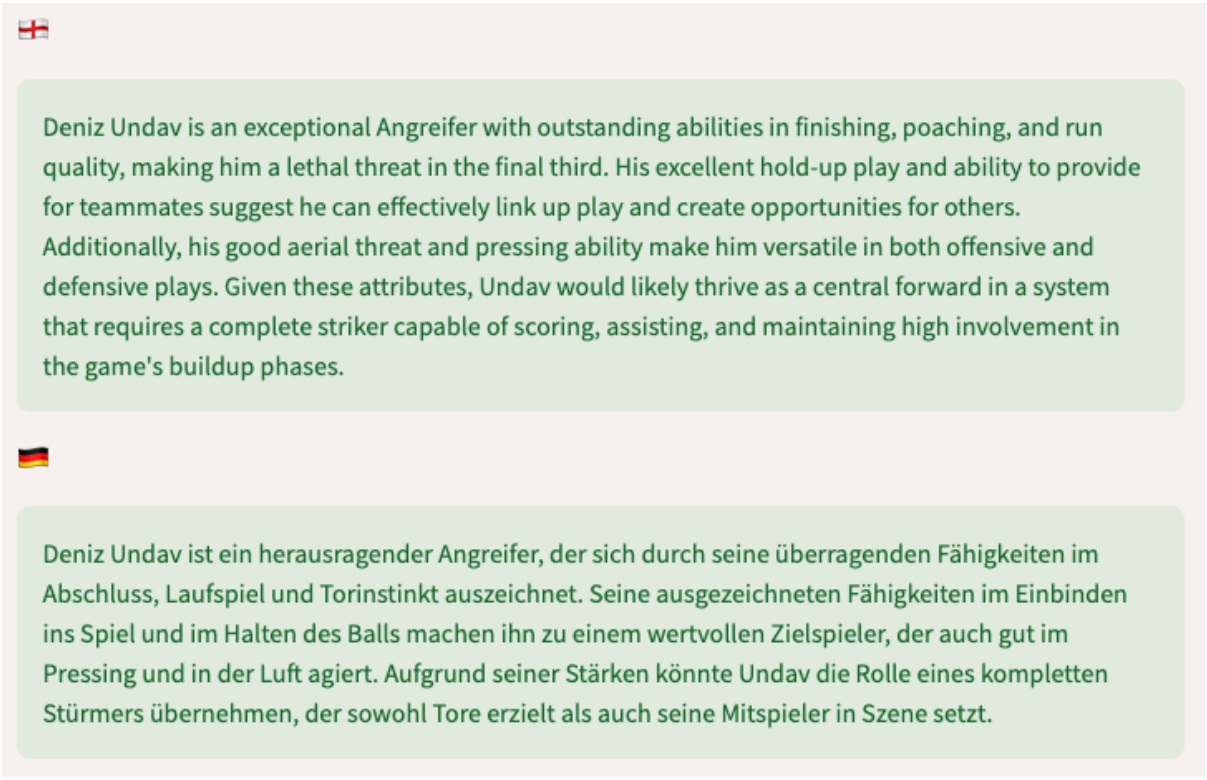
(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    response = client.chat.completions.create(
        model="gpt-4o",  # You can use other models as well
        messages=messages,
        seed=42,
        temperature=0.5
    )

    return response.choices[0].message.content
```

Als Antwort erhalten wir einen Text, der den Spieler beschreibt und die Rolle, die er in einem Team haben könnte.



KAPITEL 15

Jetzt alles online bringen

In diesem Kapitel zeige ich, wie so ein Projekt online gestellt werden kann. Der Server für dieses Projekt wird von Hetzner bereitgestellt, es ist aber natürlich auch auf jedem anderen Server möglich. Hier ist nur die Kurzanleitung. Für die Serverkonfiguration, Docker, etc. gibt es viele gute Tutorials im Internet. Das würde hier den Rahmen sprengen.

15.1 Server

Als Server dient uns ein Hetzner Cloud Server auf dem ich mehrere Projekte hoste. Dieser Server ist ein virtueller Server, der in einem Rechenzentrum von Hetzner betrieben wird. Der Server ist über das Internet erreichbar und kann für verschiedene Anwendungen genutzt werden. Die Spezifikationen des Servers sind:

- 4 vCPUs (ARM)
- 8 GB RAM
- 40 GB SSD
- Aktuelle Kosten ca. 7€/Monat
- Debian 6

Über die Hetzner Cloud Console kann der Server verwaltet werden. Hier können wir den Server starten, stoppen, neustarten, die IP-Adresse einsehen und vieles mehr. Zuerst empfehle ich das anlegen eines SSH-Keys, um sich sicherer mit dem Server zu verbinden. (https://community.hetzner.com/tutorials/howto-ssh-key/de)

Dann verbinde ich mich über die terminal.app im Mac mit meinem Server:

```
ssh root@deine_server_ip
```

Update und Upgrade deines Systems:

```
apt-get update
apt-get upgrade -y
```

15.2 Docker

Installiere benötigte Pakete:

```
apt-get install apt-transport-https ca-certificates curl gnupg2 software-properties-
↪common -y
```

Füge den Docker GPG-Schlüssel hinzu:

```
curl -fsSL https://download.docker.com/linux/debian/gpg | apt-key add -
```

Füge das Docker-Repository hinzu:

```
echo "deb [arch=arm64] https://download.docker.com/linux/debian $(lsb_release -cs)␣
↪stable" | tee /etc/apt/sources.list.d/docker.list
```

Update die Paketliste:

```
apt-get update
```

Installiere Docker:

```
apt-get install docker-ce -y
```

Überprüfe die Docker Installation:

```
docker --version
```

Als Antwort solltest ihr so etwas sehen:

```
root@debian-4gb-nbg1-1:~# docker --version
Docker version 20.10.24+dfsg1, build 297e128
```

Lade Docker Compose für ARM herunter:

```
curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-
↪$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

Setze die Berechtigungen:

```
chmod +x /usr/local/bin/docker-compose
```

Überprüfe die Docker Compose Installation:

```
docker-compose --version
```

Als Antwort solltest ihr so etwas sehen:

```
root@debian-4gb-nbg1-1:~# docker-compose --version
docker-compose version 1.29.2, build unknown
```

15.3 Portainer (optional)

Portainer ist ein Open-Source-Tool zur Verwaltung von Docker-Containern. Es bietet eine grafische Benutzeroberfläche, mit der Benutzer Container, Images, Netzwerke und Volumes verwalten können. Portainer ist nicht notwendig, kann aber die Verwaltung von Docker-Containern erleichtern. Erstelle ein neues Docker-Volume für Portainer:

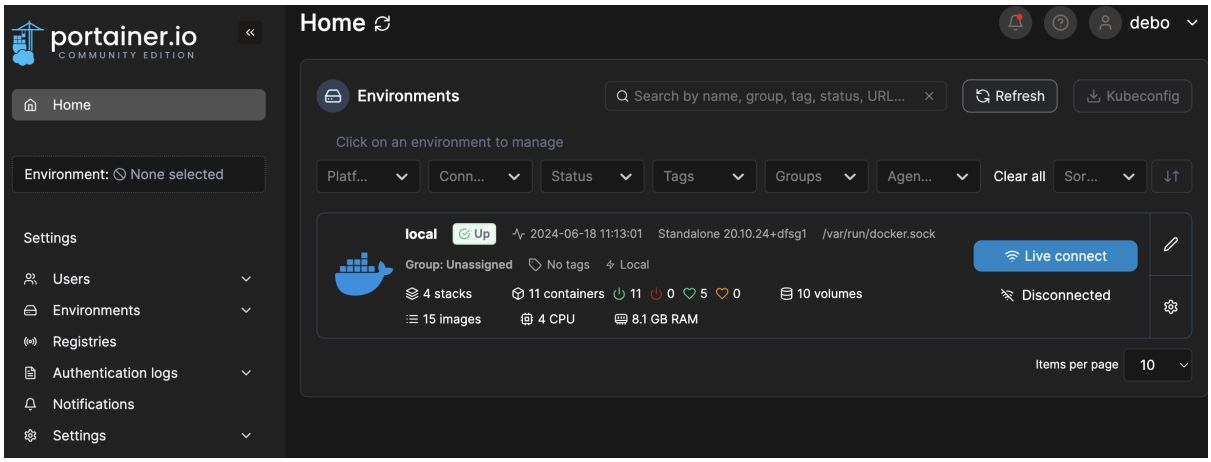
```
docker volume create portainer_data
```

Starte den Portainer-Container für ARM:

```
docker run -d -p 9000:9000 --name=portainer --restart=always -v /var/run/docker.sock:/
↪var/run/docker.sock -v portainer_data:/data portainer/portainer-ce:linux-arm
```

Bei mir sind schon einige Container installiert, bei dir sollte Portainer der einzige Container sein. Öffne Portainer in deinem Browser:

```
http://deine_server_ip:9000
```



15.4 Streamlit

Wie bekomme ich jetzt die Streamlit Anwendung auf den Server? Wir erstellen ein Docker-Image mit unserer Streamlitanwendung und laden dieses auf den Server. Dort starten wir dann den Container.

Zuerst ein Dockerfile erstellen im Anwendungsverzeichnis erstellen:

```
nano Dockerfile
```

Füge folgenden Inhalt ein:

```
FROM python:3.9.6-slim

WORKDIR /app

RUN apt-get update && apt-get install -y \
    build-essential \
    curl \
    software-properties-common \
    git \
    && rm -rf /var/lib/apt/lists/*

COPY . .
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
RUN pip3 install -r requirements.txt

# Expose Port 8501 für Streamlit
EXPOSE 8501

# Healthcheck für den Container
HEALTHCHECK CMD curl --fail http://localhost:8501/_stcore/health

# Startbefehl für Streamlit
ENTRYPOINT ["streamlit", "run", "Home.py", "--server.port=8501", "--server.address=0.
↪0.0.0"]
```

Du kannst natürlich die Python Version und die Portnummer anpassen. Achte auch darauf, dass die Datei zum starten bei mir Home.py heißt.

In der requirements.txt Datei stehen alle benötigten Python Pakete. Damit diese mit in dem Docker-Image installiert werden. Die requirements.txt Datei sollte im gleichen Verzeichnis wie das Dockerfile liegen und wird folgendermaßen erstellt:

```
pip freeze > requirements.txt
```

Deshalb empfehle ich auch immer eine virtuelle Umgebung für jedes Projekt zu erstellen, um später die benötigten Pakete zu installieren und keine Konflikte mit anderen Projekten zu bekommen.

Docker Login (Du wirst nach deinem Docker Hub Benutzernamen und Passwort gefragt. Falls nicht vorhanden, auf Docker Hub registrieren):

```
docker login
```

Nun das Docker-Image erstellen (mit buildx für verschiedene Serverarchitekturen) und mit –push direkt auf Docker Hub hochladen:

```
docker buildx build -t dein_dockername/dein_imagename --push . --no-cache
```

Dies kann ein paar Minuten dauern.

Nun auf dem Server ebenfalls in Docker einloggen:

```
docker login
```

Das Docker Image ziehen:

```
docker pull dein_dockername/dein_imagename
```

Mit:

```
docker images
```

siehst du alle images und kannst die Image ID sehen. Die brauchst du im nächsten Schritt, um den Container zu starten.

Nun den Container starten:

```
docker run -p 8501:8501 -dit --restart unless-stopped <imageid>
```

Mit

```
docker ps
```

kannst du nun deinen Container sehen.

Deine Anwendung sollte nun unter http://deine_server_ip:8501 erreichbar sein.

15.5 Proxy Manager

Jede Streamlit Anwendung wird einem bestimmten Port zugewiesen. Damit wir unserem Server eine Domain (datenflanke.de) zuweisen können und diese dann direkt die richtige Streamlit Anwendung findet, leiten wir jede Anfrage, welche über unsere Domain an den Server gesendet wird, an den richtigen Port weiter. Dafür nutzen wir den Proxy Manager nginx.

Verzeichnis erstellen:

```
mkdir -p /opt/nginx-proxy-manager
cd /opt/nginx-proxy-manager
```

Erstelle die Datei docker-compose.yml:

```
nano docker-compose.yml
```

Füge folgenden Inhalt ein und ersetze root_password und npm_password mit sicheren Passwörtern deiner Wahl:

```
version: '3'
services:
  app:
    image: 'jc21/nginx-proxy-manager:latest-armhf'
    restart: unless-stopped
    ports:
      - '80:80'
      - '81:81'
      - '443:443'
    environment:
      DB_MYSQL_HOST: "db"
      DB_MYSQL_PORT: 3306
      DB_MYSQL_USER: "npm"
      DB_MYSQL_PASSWORD: "npm_password"
      DB_MYSQL_NAME: "npm"
    volumes:
      - ./data:/data
      - ./letsencrypt:/etc/letsencrypt

  db:
    image: 'yobasystems/alpine-mariadb:latest-armhf'
    restart: unless-stopped
    environment:
      MYSQL_ROOT_PASSWORD: 'root_password'
      MYSQL_DATABASE: 'npm'
      MYSQL_USER: 'npm'
      MYSQL_PASSWORD: 'npm_password'
    volumes:
      - ./mysql:/var/lib/mysql
```

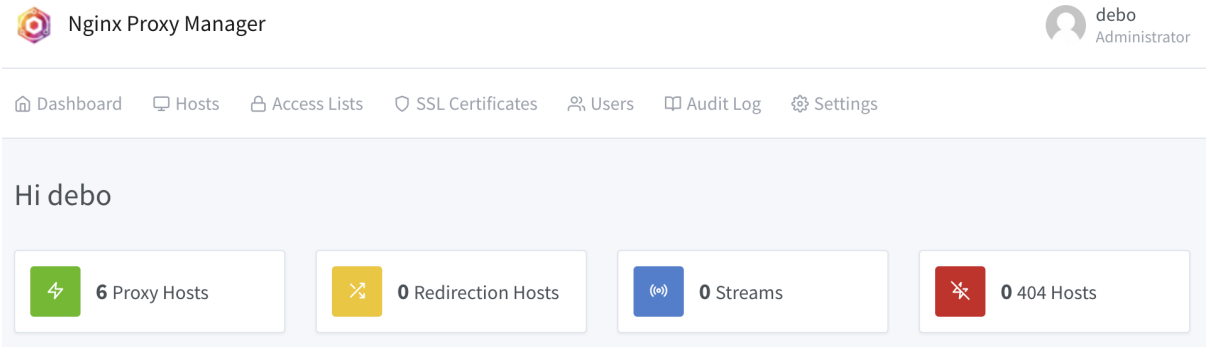
Starte die Docker-Container:

```
docker-compose up -d
```

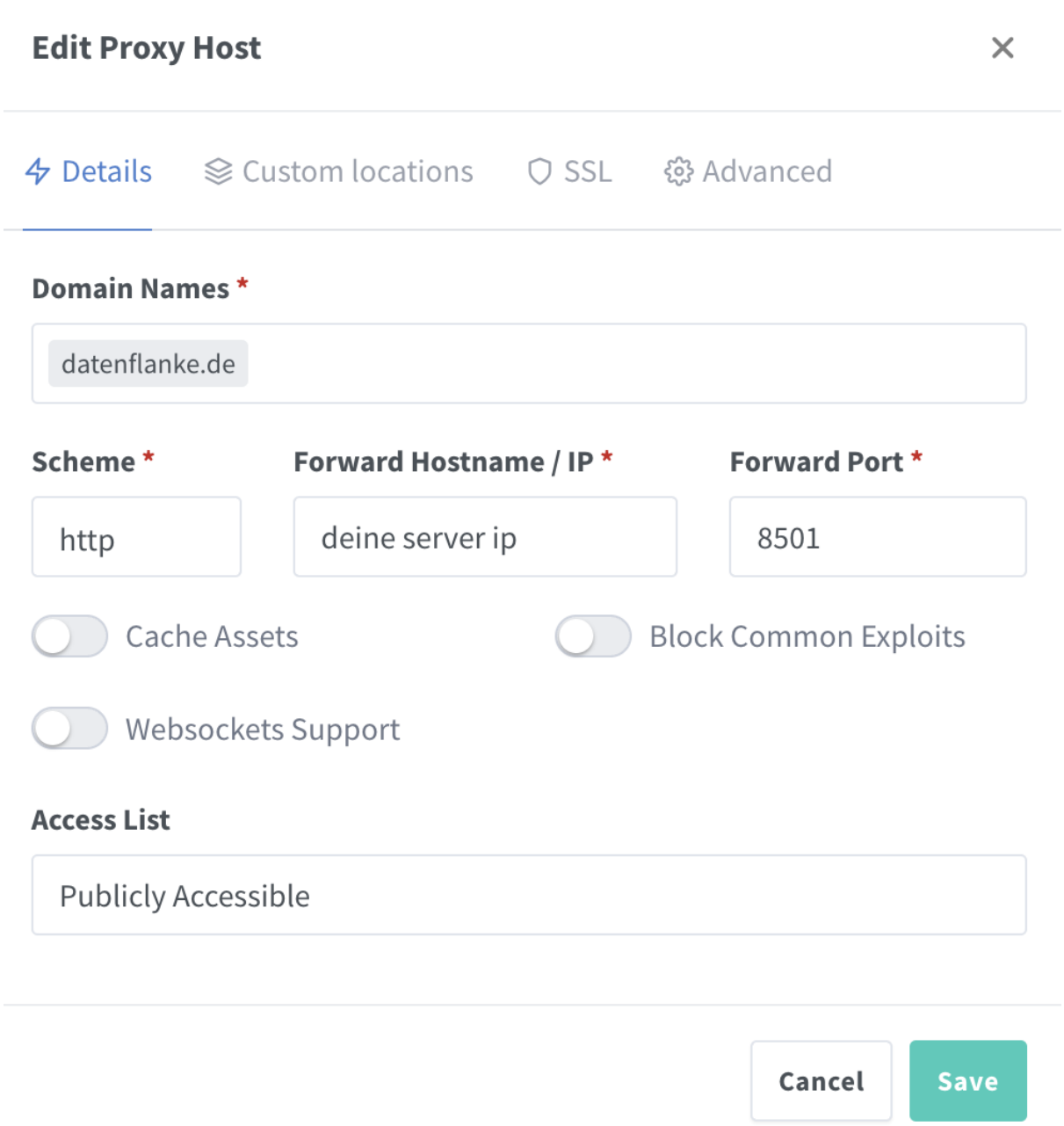
Zugriff auf das Dashboard:

```
http://deine_server_ip:81
```

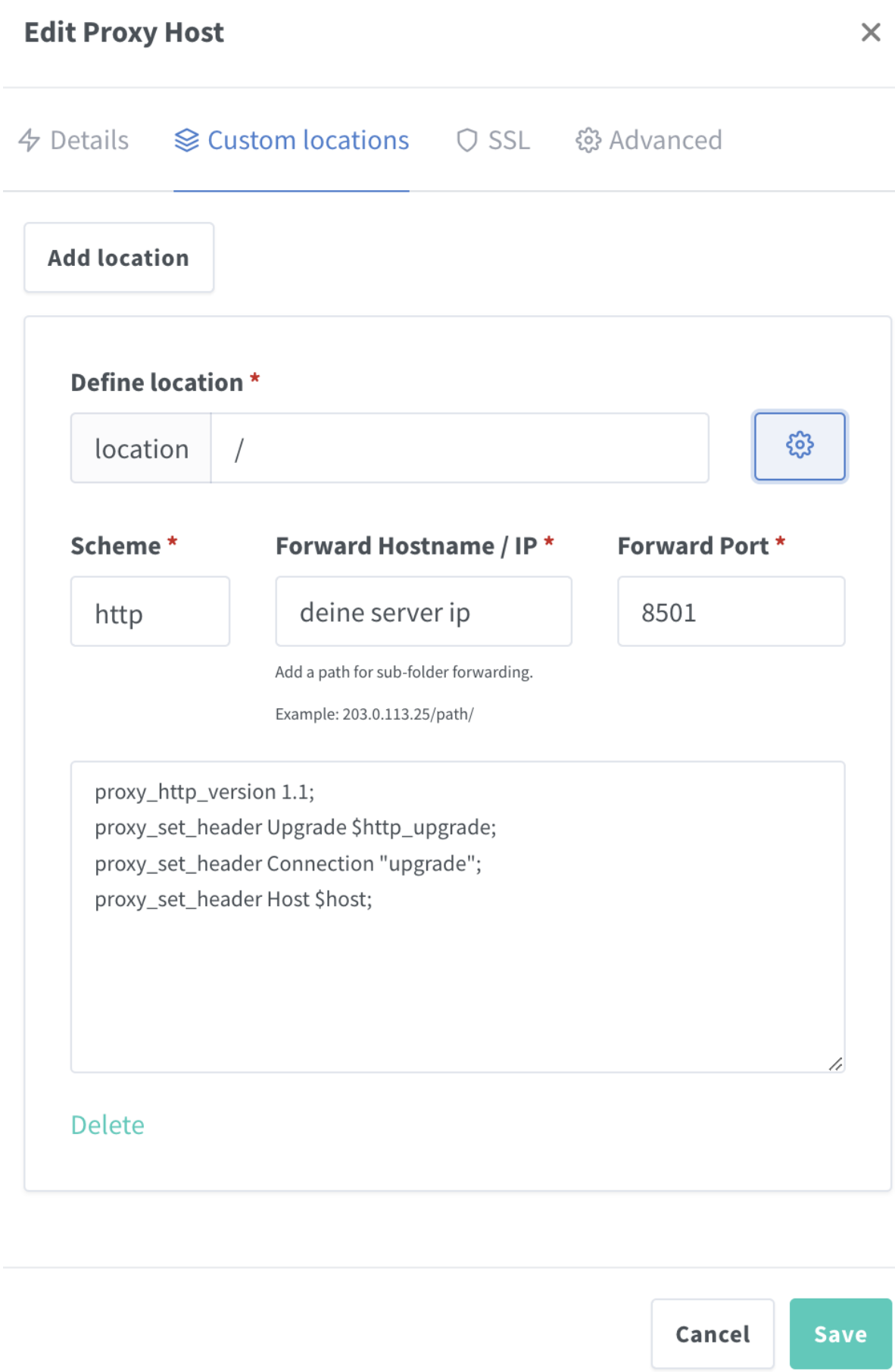
Bei mir sind schon einige Domains hinterlegt. Bei dir sollte die Liste leer sein.

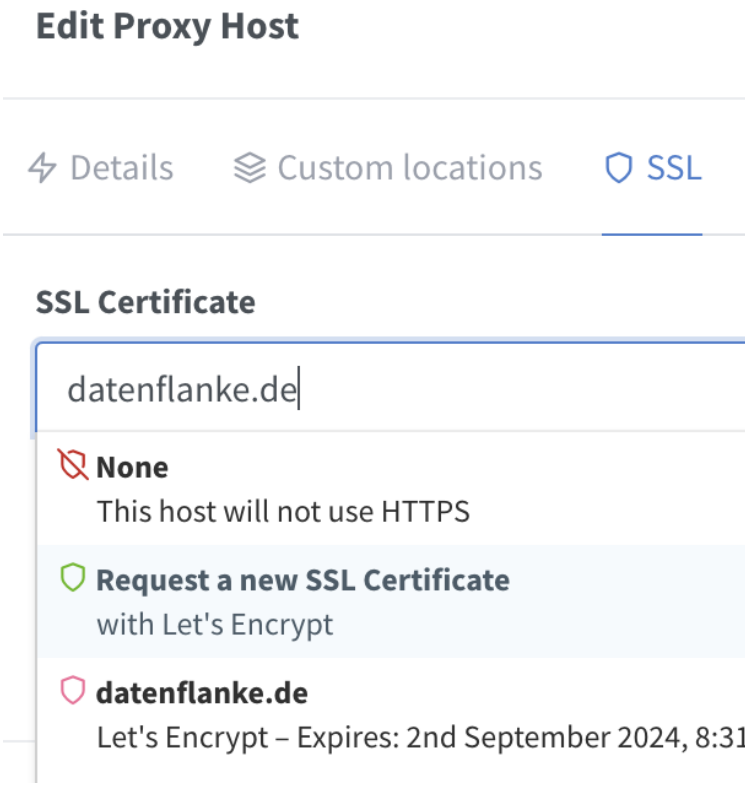


Nun kannst du einen neuen Proxy hinzufügen über Add Proxy Host im Proxy Host Menü: Der Streamlit Standardport ist 8501 - je nach dem welchen Port du beim Start des Dockercontainer benutzt hast.



Wichtig ist hier Custom locations hinzuzufügen, da die Streamlit Anwendung sonst nicht starten kann. (https://discuss.streamlit.io/t/nginx-setup-websocket-connection-error-with-a-subdomain-name/47787)


Edit Proxy Host
Details
Custom locations
SSL
Advanced
Add location
Define location *
location
/
Scheme *
Forward Hostname / IP *
Forward Port *
http
deine server ip
8501
Add a path for sub-folder forwarding.
Example: 203.0.113.25/path/
proxy_http_version 1.1;
proxy_set_header Upgrade $http_upgrade;
proxy_set_header Connection "upgrade";
proxy_set_header Host $host;
Delete
Cancel
Save



Und, falls noch nicht geschehen ein SSL Zertifikat hinzufügen:

KAPITEL 16

. . . und Ausblick

In dieser Dokumentation habe ich gezeigt, wie eine datenbasierte Spielerbewertung ablaufen kann anhand von Eventdaten. Wichtig ist hierbei, dass die Qualitäten individuell auf den Verein abgestimmt definiert werden. Erst daraus ergibt sich die Einzigartiigkeit und der Wettbewerbsvorteil gegenüber anderen Vereinen. Folgende Erweiterungen und Verbesserungen könnten in einer realen Umgebung vorgenommen werden:

- Mit den Eventdaten könnte das Modell noch mit einem eigenen expected Goals Modell erweitert werden
- Durch die Anbindung an eine Schnittstelle eines Datenanbieters könnten auch Spielerwerte wie Alter, Größe, Gewicht, Marktwert etc. in das Modell einfließen
- Aus mehreren Saisons könnten auch Entwicklungen von Spielern über die Zeit analysiert werden und so z.B. auch die Entwicklung von Talenten besser prognostiziert werden (Positionsspezifische Entwicklungskurven und/oder Qualitätskurven). Wie wird sich Spieler X in den nächsten 1-3 Jahren entwickeln?
- Durch die Analyse von Spielerwechsel in unterschiedliche Ligen, könnten Prognosen für die Anpassungsfähigkeit bestimmter Qualitäten in einer andere Ligen getroffen werden. Wie gut wäre Spieler x aus der Serie B in der Bundesliga?
- Es könnte analog ein eigenes Bewertungsmodell für Teams entwickelt werden, um damit dann zu analysieren, welche Spieler in welchem Team am besten funktionieren würden. (Teamfit)

Einige dieser Punkte habe ich bereits in anderen Projekten umgesetzt und werde sie bei Zeit in weiteren Dokumentationen erläutern.

Ich hoffe, dass ich mit dieser Dokumentation einen Einblick in die Welt des Datenscoutings geben konnte und freue mich über Feedback und Anregungen.