UNI
FR
UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Freiburg, November 2025

Vergleich verschiedener Machine-Learning-Modelle unter Einbezug eines neuen Einflussfaktors anhand der Bundesliga-Saison 2024/2025

vorgelegt von:

Vor.- Nachname: Nicola Cyrill Schärer

Studentennummer: 22-207-476

E-Mail: Nicola.Schaerer@unifr.ch

Für den Bachelor of Science in Wirtschaftsinformatik

Betreuer:

Prof. Dr.Bastian Alexander Grossenbacher

Lehrstuhl AI for Data-Oriented Science

Department für informatik

Mathematisch-Naturwissenschaftliche und Medizinische Fakultät

Universität Fribourg

Inhaltsverzeichnis

1. Einleitung

1.1 Motivation und Problemstellung

Das Wetten auf Sportereignisse ist älter als der Fussball selbst. Bereits in der Antike wurde auf Wettbewerbe, wie zum Beispiel die Olympischen Spiele, gewettet. Mit der zunehmenden Popularität des Fussballs im 19. Jahrhundert, boten Buchmacher neben Pferderennen, Boxen und Rudern, auch Fussball-Wetten an. Anfangs wurden die Wetten noch direkt vor Ort abgegeben, später konnte man diese auch per Telefon, Post oder Fax platzieren und per Banküberweissung bezahlen. Aufgrund dieses umständlichen Prozesses war das Wetten auf den Fussball, nur in einem kleinen Teil der Bevölkerung verbreitet (vgl. Sportwettentest.net, o. J.).

Erst mit dem Aufkommen des Internets änderte sich dies grundlegend. Dadurch entstanden unzählige Wettbüros, die Fussballspiele zeigten, die vom öffentlichen Fernseher nicht ausgestrahlt wurden. So war es für viele Fussball begeisterte, die einzige Möglichkeit das Fussballspiel live zu verfolgen. So wurde das Platzieren von Wetten, für viele, zunehmend Teil des Fussballerlebnisses. Dadurch wurden Wettbüros immer lukrativer, was zu einem massiven Anstieg der Anzahl an Wettbüros führte. Mehr Menschen wurden so auf Sportwetten aufmerksam gemacht, so konnten die Wettbetreiber, noch mehr Kundschaft generieren (vgl. Sportwettentest.net, o. J.).

Denn Durchbruch in der breiten Bevölkerung schaffte man, jedoch erst mit dem Angebot der Online-Wetten. Online-Wetten bieten zahlreiche Vorteile gegenüber den herkömmlichen Wettanbieter, unteranderem einfache Zahlungsmethoden, jederzeitige Wettabgabe, insbesondere bei Live-Wetten wichtig, sowie eine Riesenauswahl an unterschiedlichen Wettmöglichkeiten. Die zunehmende Konkurrenz unter den Anbietern selbst, führte zu einem immensen Innovationsdruck, der sich unter anderem in der Entwicklung immer präziserer Quoten und Prognosen niederschlug, sich aber auch bei immer höheren Einzahlung- und Wett-Boni zeigte (vgl. Sportwettentest.net, o. J.).

Um konkurrenzfähig zu bleiben, greifen heutzutage viele Wettanbieter auf datengetriebene Modelle und Verfahren des maschinellen Lernens zurück. Diese Modelle ermöglichen es, auf Basis historischer Daten sowie statistischer Zusammenhänge, Vorhersagen über zukünftige Spielergebnisse zu treffen (vgl Atta Mills et al. (2024), S. 3–4). Trotz dieser Fortschritte bleibt die präzise Prognose eines Fussball Spiels eine Herausforderung, da zahlreiche schwer quantifizierbare Faktoren, wie etwa Motivation, Tagesform oder externe Einflüsse, das Spielgeschehen beeinflussen können (Pietraszewski et al., 2025).

Persönlich verfolge ich Sport schon seit meiner Kindheit. Mit etwa sieben Jahren entwickelte ich, wie viele Jungs in dem Alter auch, eine Faszination für den Fussball. Bereits in der Schulzeit begann ich, mich mit Spielanalysen und Prognosen zu beschäftigen, zunächst aus reinem Interesse am Sport, später auch im Rahmen von Sportwetten.

Nach einiger Zeit musste ich feststellen, dass selbst bei sorgfältiger Vorbereitung und der Einbeziehung vielfältiger Informationen zwar genauere Einschätzungen möglich sind, langfristig jedoch kaum ein Vorteil gegenüber renommierten Buchmachern zu erzielen ist. Diese langjährige Auseinandersetzung mit dieser Thematik hat mein Interesse an datenbasierten Vorhersagemodellen für Sportergebnisse, speziell den Fussball, geweckt und bildet somit die Grundlage für diese Arbeit.

1.2 Zielsetzung der Arbeit

Das Ziel dieser Bachelorarbeit ist es, drei unterschiedliche Prognosemodelle, die Vorhersagen zu Fussballergebnissen liefern, systematisch zu analysieren und hinsichtlich ihrer Vorhersagegenauigkeit für die Spiele der Bundesliga-Saison 2024/2025 zu vergleichen. Dabei wurden drei Modelle ausgewählt, die auf verschiedenen methodischen Ansätzen beruhen:

1. **Buchmacher-Modell**: Prognose auf Basis der niedrigsten Wettquote (höchste implizite Wahrscheinlichkeit).

2. **Logistisches Regressionsmodell**: ein klassisches statistisches Verfahren, das Wahrscheinlichkeiten für Sieg, Niederlage oder Unentschieden berechnet und gleichzeitig als einfacher Machine-Learning-Ansatz im überwachten Lernen eingesetzt werden kann

3. **Gradient Boosting Modell**: Ein Machine-Learning-Ansatz aus der Familie der Entscheidungsbäume, der im Vergleich zur logistischen Regression komplexere und nichtlineare Zusammenhänge zwischen Variablen erfassen kann.

Ergänzend wird in einem vierten Schritt untersucht, ob ein selbst entwickeltes Feature (EM-Score) die Prognoseleistung verbessern kann, indem es die Schätzung der Quoten beeinflusst.

Zur Analyse der Vorhersagegenauigkeit werden die gängigen Metriken Accuracy, Precision, Recall und F1-Score verwendet. Damit können die Modelle nicht nur in Bezug auf ihre Gesamtgenauigkeit miteinander verglichen werden, sondern auch hinsichtlich ihrer Fähigkeit, bestimmte Ergebnisarten (z. B. Heimsiege) korrekt zu prognostizieren.

Ursprünglich war vorgesehen, bestehende Modelle zu vergleichen, die auf einer grösseren Anzahl an Features beruhen. Aufgrund der eingeschränkten Datenlage (vgl. Kapitel 3) wurde dieses Ziel angepasst. Stattdessen liegt der Fokus nun auf der systematischen Evaluation eines Referenzmodells, das in mehreren Schritten angepasst und erweitert wurde. Ergänzend wird untersucht, ob bislang unberücksichtigte Einflussfaktoren identifiziert und in die Modelle integriert werden können.

Die Arbeit verfolgt somit drei zentrale Ziele:

1. Es sollten 3 unterschiedliche Vorhersagemodelle, mit Hilfe der gängigen Metriken Accuracy, Precision, Recall und F1-Score, miteinander vergleichen werden.

2. Dabei sollten mindestens 2 Modelle auf einem Machine-Learning Ansatz beruhen.

3. Es soll ein Einflussfaktor identifiziert werden, der in bestehenden Machine-Learning-Modellen bislang unberücksichtigt blieb.

1.3 Methodisches Vorgehen

Die Erstellung der Arbeit erfolgte in einem iterativen Prozess, der von einer Kombination aus theoretischer Recherche, praktischer Modellierung und kontinuierlicher Überarbeitung geprägt war.

Am Anfang stand eine Literaturrecherche, mit der die Grundlagen zu bestehenden Prognosemodellen im Fussballkontext erarbeitet wurden. Diese Recherche musste im Verlauf der Arbeit mehrfach erneut durchgeführt werden, da während der Modellierung und Analyse immer wieder neue Fragen entstanden, die eine vertiefte theoretische Fundierung erforderten.

Auf Basis der Literatur wurde anschliessend ein geeignetes Referenzmodell ausgewählt, das als Ausgangspunkt diente. Da viele bestehende Ansätze nicht direkt auf die Fragestellung übertragbar waren, wurde dieses Modell angepasst und erweitert. Der vorhandene Quellcode des Referenzmodells erwies sich dabei als hilfreich, da er eine solide Grundlage bot (vgl. Andretta, o. J.).

Die eigene Implementierung erfolgte schrittweise und wurde durch den Einsatz von ChatGPT unterstützt. Dabei nutzte ich das Tool insbesondere, um Code zu entwerfen, Fehler zu beheben und Optimierungen vorzunehmen. Der iterative Charakter zeigte sich darin, dass Zwischenergebnisse regelmässig überprüft, interpretiert und in Rücksprache mit ChatGPT angepasst wurden, bis die erzielten Resultate zufriedenstellend waren.

Parallel dazu wurde die schriftliche Ausarbeitung entwickelt. Auch hier entstand der Text zunächst in Rohfassungen, die anschliessend überarbeitet wurden. ChatGPT diente in diesem Zusammenhang als Hilfsmittel für Feedback, stilistische und grammatikalische Korrekturen sowie für die Strukturierung einzelner Textabschnitte. Die endgültige Fassung wurde stets am Leitfaden der Universität ausgerichtet, um formale Anforderungen einzuhalten.

Insgesamt zeigt dieses Vorgehen, dass die Arbeit nicht in einem linearen Prozess entstand, sondern durch wiederholte Recherche, modellbasierte Experimente, iterative Verbesserungen und den gezielten Einsatz digitaler Werkzeuge zu einem konsistenten Ergebnis geführt wurde.

1.4 Kurze Darstellung der wichtigsten Ergebnisse

Die Ergebnisse zeigen, dass datengetriebene Modelle in der Lage sind, Fussballergebnisse mit einer gewissen Genauigkeit vorherzusagen. Während die logistische Regression als interpretiertes Basismodell solide Leistungen erbrachte, konnte der Gradient-Boosting Ansatz punktuell bessere Ergebnisse erzielen.

Das neu entwickelte EM-Feature führte zu leichten Verbesserungen der Vorhersagequalität und verdeutlichte die Relevanz von Feature Engineering. Gleichzeitig wurde deutlich, dass die Verfügbarkeit geeigneter Daten eine zentrale Herausforderung darstellt.

1.5 Literaturüberblick

Die wissenschaftliche Literatur zur Vorhersage von Sportereignissen ist breit gefächert und reicht von klassischen statistischen Verfahren bis hin zu modernen Machine-Learning-Ansätzen (vgl. Christoffersson, 2023; Király & Qian, 2017; Atta Mills et al., 2024). Frühere Arbeiten zeigen, dass insbesondere die Poisson-Regression sowie maschinelle Lernverfahren wie logistische Regression oder Entscheidungsbäume häufig eingesetzt werden (vgl. Christoffersson, 2023; Király & Qian, 2017).

Darüber hinaus wird in zahlreichen Studien die Rolle von Buchmacherquoten betont, die oft sehr nah an den tatsächlichen Eintrittswahrscheinlichkeiten liegen und daher eine wichtige Benchmark darstellen (vgl. Constantinou & Fenton, 2013). Jüngere Forschungsansätze beschäftigen sich zunehmend mit der Integration zusätzlicher Einflussgrössen sowie der Anwendung komplexer Ensemble-Verfahren (vgl. Atta Mills et al., 2024; Hubáček et al., 2019; Walsh & Joshi, 2023).

1.6 Aufbau der Arbeit

Die Arbeit gliedert sich in sechs Kapitel:

- **Kapitel 1** führt in die Thematik ein, beschreibt Zielsetzung, methodisches Vorgehen und Forschungsstand.
- **Kapitel 2** bietet den theoretischen Hintergrund zu statistischen und maschinellen Lernverfahren sowie zu relevanten Einflussfaktoren im Fussball.
- **Kapitel 3** stellt die Methodik der empirischen Untersuchung vor, einschliesslich Kriterien für den Modellvergleich, Vorgehen zur Feature-Entwicklung und Datengrundlagen.
- **Kapitel 4** präsentiert die empirischen Ergebnisse der Modellanalysen.
- **Kapitel 5** diskutiert die Resultate kritisch im Hinblick auf die Forschungsfrage und methodische Herausforderungen.
- **Kapitel 6** fasst die zentralen Erkenntnisse zusammen, beantwortet die Forschungsfrage und gibt einen Ausblick auf mögliche zukünftige Forschung.

2 Theoretischer und konzeptioneller Hintergrund

2.1 Forschungsstand

Mit der steigenden Attraktivität von Sportwetten und den dadurch potenziell höheren Gewinnen der Buchmacher, insbesondere im Fussball, hat sich dieses Themenfeld zu einem komplexen Untersuchungsgebiet entwickelt (vgl. Sportwettentest.net, o. J.; Constantinou & Fenton, 2013, S. 1–2). Frühere Ansätze zur Vorhersage von Spielergebnissen basierten vor allem auf Intuition, Experteneinschätzungen oder einfachen statistischen Auswertungsmodellen (vgl. Sportwettentest.net, o. J.).

Ein klassisches statistisches Auswertungsmodell in den klassischen Sportprognosen ist die Poisson-Regression, bei der Torergebnisse als Wahrscheinlichkeitsverteilungen modelliert werden. Studien zeigen, dass sich mit diesem Verfahren Eintrittswahrscheinlichkeiten für Sieg, Unentschieden oder Niederlage realistisch berechnen lassen (vgl. Dixon & Coles, 1997, S. 5–7).

In den letzten Jahren haben sich jedoch zunehmend Methoden des maschinellen Lernens als leistungsfähige Alternativen etabliert (vgl. Atta Mills et al., 2024, S. 2-4; vgl. Alfredo & Isa, 2019). Prognosemodelle können heute auf grosse Datenmengen und leistungsfähige Rechenmethoden zurückgreifen, wodurch komplexere Verfahren zum Einsatz kommen. Verfahren wie die logistische Regression, Entscheidungsbäume oder XGBoost ermöglichen es, eine Vielzahl von Merkmalen gleichzeitig zu berücksichtigen und komplexe Muster in historischen Daten zu erkennen (vgl. Markopoulou et al., 2023, S.4-6; vgl. Atta Mills et al., 2024, S. 2-4)

(Tsokos et al., 2019, S. 1-3) zeigen in einer Vergleichsstudie, dass klassische Poisson-Modelle eine ähnliche Vorhersagequalität erreichen können, wie moderne ML-Ansätze, obwohl die methodischen Grundlagen beider Ansätze stark voneinander abweichen.

In einem weiteren Abschnitt betonen dieselben Autoren zudem die Wichtigkeit der Feature Auswahl und der Modellwahl für die Prognosequalität, was unterstreicht, dass die Leistungsfähigkeit datengetriebener Verfahren entscheidend von den verwendeten Eingangsvariablen abhängt (vgl. Tsokos et al., 2019, S. 16-18).

Neben diesen datengetriebenen Ansätzen spielen auch die Wettquoten der Buchmacher eine zentrale Rolle. Diese spiegeln nicht nur die Wahrscheinlichkeitsannahmen der Anbieter wieder, sondern entstehen heute häufig selbst auf Basis statistischer Modelle und Machine-Learning-Verfahren (vgl. Constantinou & Fenton, 2013, S. 4–8; Hubáček et al., 2019). Dabei werden historische Spieldaten, Spieler- und Teamstatistiken sowie weitere Einflussfaktoren analysiert, um Eintrittswahrscheinlichkeiten möglichst präzise zu schätzen (vgl. Király & Qian, 2017).

Zusätzlich werden Quoten an die Wettaktivität der Kunden angepasst, bedeutet also, wenn viele «Wettende» auf ein bestimmtes Ereignis setzten (z.B. Heimsieg), wird diese Quote entsprechend nach unten angepasst, um so das Risiko auszugleichen und die potenzielle Auszahlung zu verringern. Wenn auf ein anderes Resultat wenig getippt wird, wird die Quote entsprechend nach oben angepasst, um so die Aktivität dieser Wette zu erhöhen. (vgl. Liga-Zwei.de, o. J.)

Zahlreiche Untersuchungen zeigen, dass Buchmacherquoten in der Praxis häufig sehr nahe an den tatsächlichen Eintrittswahrscheinlichkeiten liegen, was sie zu einer wichtigen Benchmark für jede Art von Prognosemodell macht (vgl. Constantinou & Fenton, 2013, S. 5–9; Hubáček et al., 2019).

Insgesamt zeigt sich, dass die Prognose von Fussballergebnissen heute ein interdisziplinäres Feld darstellt, das klassische statistische Methoden, maschinelles Lernen und ökonomische Überlegungen miteinander verbindet (vgl. Atta Mills et al., 2024, S. 2-5; Pietraszewski et al., 2025, S.10-14).

2.2 Problemdefinition (Klassifikation)

Vor diesem Hintergrund ist es entscheidend, sich zunächst bewusst zu machen, welches Problem mit Hilfe eines Machine-Learning-Ansatzes gelöst werden soll. Bevor also ein entsprechendes Modell entwickelt werden kann, ist eine klare Problemanalyse notwendig. Grundsätzlich lassen sich die Fragestellungen, die man mithilfe von Machine-Learning-Modellen beantwortet, in zwei Kategorien einteilen: Regressions- und Klassifikationsprobleme (IBM, o. J.).

Regressionsprobleme zielen darauf ab, kontinuierliche Werte vorherzusagen, die auf einer Skala liegen und beliebig gross oder klein sein können, wie beispielsweise die Anzahl erzielter Tore oder der Marktwert eines Spielers (IBM, o. J.).

Klassifikationsprobleme hingegen ordnen Beobachtungen bestimmten Gruppen oder Kategorien zu, wie zum Beispiel Heimsieg, Unentschieden oder Auswärtssieg. Da in dieser Arbeit ausschliesslich die Vorhersage von Fussballergebnissen im Fokus steht, handelt es sich um ein klassisches Klassifikationsproblem (IBM, o. J.).

Im folgenden Kapitel werden daher verschiedene Machine-Learning-Methoden vorgestellt und miteinander verglichen, die speziell auf Klassifikationsaufgaben zugeschnitten sind.

2.3 Machine-Learning-Modelle

Für die Lösung von Klassifikationsproblemen stehen unterschiedliche Machine-Learning-Ansätze zur Verfügung, die sich in ihrem Anwendungsbereich, Funktionsweise, Stärken und Schwächen unterscheiden. Die folgende Übersicht basiert auf den grundlegenden Klassifikationsverfahren, wie sie in *An Introduction to Statistical Learning* beschrieben werden (James et al., 2021, Kapitel 4–8), und wird durch praktische Hinweise aus der Dokumentation von scikit-learn ergänzt (scikit-learn developers, 2024).

2.3.1 Logistische Regression

Die logistische Regression (James et al., 2021, S. 132–138; scikit-learn developers, 2024) modelliert die Wahrscheinlichkeit, dass ein bestimmtes Ereignis eintritt, indem sie die lineare Kombination der Eingangsvariablen durch die Logit-Funktion normiert. Dadurch liegen die vorhergesagten Werte immer zwischen 0 und 1, was später die Klassenzuordnung erleichtert.

Anwendungsbereich: Klassische Probleme mit binären oder mehrklassigen Zielvariablen, z. B. Vorhersage von Krankheit vs. Gesund, kreditwürdig vs. nicht kreditwürdig, oder Sieg/Unentschieden/Niederlage beim Fussball. Besonders geeignet, wenn die Beziehung zwischen den Merkmalen und dem Ziel weitgehend linear ist und die Daten überschaubar sind.

Vorteile: einfache Implementierung, gut interpretierbare Koeffizienten, liefert Wahrscheinlichkeiten als Vorhersage.

Nachteile: eingeschränkte Modellierung nichtlinearer Zusammenhänge, geringere Leistung bei komplexeren Datensätzen, empfindlich gegenüber stark korrelierten Variablen.

2.3.2 Support Vector Machine (SVM)

Die Support Vector Machine (SVM) (James et al., 2021, S. 337–353; scikit-learn developers, 2024) versucht, die Klassen durch eine optimale Trennlinie, den sogenannten Hyperplane, zu separieren. Bei linearen Problemen wird diese Linie so gewählt, dass der Abstand zwischen den Klassen maximal ist. SVMs konzentrieren sich dabei auf die Punkte, die der Trennlinie am nächsten liegen, die sogenannten Support Vectors.

Diese Punkte bestimmen die Position des Hyperplanes, während weiter entfernte Punkte keinen Einfluss haben. Dadurch werden Ausreisser im Modell weniger stark gewichtet und die Klassifikation robuster gegenüber neuen Datenpunkten.

Anwendungsbereich: SVMs eignen sich besonders für Klassifikationsprobleme mit mittelgrossen Datensätzen, bei denen die Klassen trennbar oder nur leicht überlappend sind. Typische Anwendungen finden sich in der Textklassifikation, Bild- und Spracherkennung oder in medizinischen Diagnoseaufgaben.

Vorteile: hohe Genauigkeit bei gut trennbaren Klassen, robust gegenüber Ausreissern, flexibel durch Kernel-Tricks.

Nachteile: rechenintensiv bei grossen Datensätzen, schwierige Hyperparameter-Abstimmung, geringe Interpretierbarkeit, empfindlich bei stark überlappenden Klassen.

2.3.3 Random Forest

Random Forest (James et al., 2021, S. 314–322; scikit-learn developers, 2024) ist ein Ensemble-Verfahren, das sich sowohl für Klassifikations- als auch Regressionsprobleme verwenden lässt. Da sich bei der Problemanalyse herausgestellt hat, dass es sich um ein Klassifikationsproblem handelt, wird in diesem Abschnitt der Random Forest ausschliesslich im Hinblick auf die Klassifikationsprobleme erläutert.

Beim Random Forest werden gleichzeitig verschiedene Entscheidungsbäume trainiert. Jeder Baum wird dabei anhand einer zufälligen Teilstichprobe der Daten und einer zufälligen Auswahl von Features aufgebaut. Anschliessend trifft jeder Baum eine eigene Vorhersage, die Klasse, die von den meisten Bäumen vorhergesagt wird, wird als endgültiges Ergebnis gewählt. Auf diese Weise werden zufällige Fehler einzelner Bäume ausgeglichen, und das Modell wird robuster gegenüber Ausreissern und Überanpassung.

Anwendungsbereich: Random Forest eignet sich besonders für Datensätze mit vielen Variablen, komplexen nichtlinearen Zusammenhängen oder wenn Ausreisser vorhanden sind. Typische Anwendungen finden sich in der medizinischen Diagnostik (z. B. Klassifikation von Tumorarten), Kundenklassifikation im Marketing (z. B. Segmentierung nach Kaufverhalten), oder Sportvorhersagen (z. B. Klassifikation von Spielergebnissen in Sieg, Unentschieden, Niederlage).

Vorteile: robust gegenüber Ausreissern und Störungen, gute Leistung bei nichtlinearen Zusammenhängen, weniger anfällig für Overfitting als einzelne Entscheidungsbäume, relativ einfach zu implementieren.

Nachteile: eingeschränkte Interpretierbarkeit einzelner Features, langsamer bei sehr grossen Datensätzen, höherer Speicherbedarf, Vorhersagen sind weniger transparent als bei einzelnen Entscheidungsbäumen.

2.3.4 Gradient Boosting

Gradient Boosting (James et al., 2021, S. 343–350; scikit-learn developers, 2024) ist wie Random Forest auch ein Ensemble-Verfahren, es kombiniert die Entscheidungsbäume jedoch sequenziell. Dies bedeutet das Gradient Boosting die Bäume nacheinander trainiert (im Gegensatz zu Random Forest, wo die Bäume parallel trainiert werden), wobei jeder neue Baum gezielt die Fehler der vorherigen korrigiert. Dadurch kann Gradient Boosting kleinere Unterschiede in den Daten besser erfassen und oft eine höhere Vorhersagegenauigkeit erzielen, ist aber empfindlicher gegenüber Overfitting und benötigt sorgfältige Hyperparameter-Abstimmung.

Anwendungsbereich: Gradient Boosting eignet sich insbesondere für Klassifikationsprobleme mit komplexen, nichtlinearen Zusammenhängen und deckt damit denselben Anwendungsbereich wie Random Forest ab. Besonders effektiv ist das Modell, wenn die Daten heterogen sind, also wenn die Merkmale unterschiedlich skaliert vorliegen und die Beziehungen zwischen Variablen kompliziert oder nichtlinear sind.

Vorteile: sehr hohe Vorhersagegenauigkeit, gezielte Modellierung komplexer Muster, flexibel einsetzbar bei heterogenen und nichtlinearen Datenstrukturen.

Nachteile: anfällig für Overfitting, längere Trainingszeiten, viele Hyperparameter, deren Abstimmung komplex sein kann, geringere Interpretierbarkeit einzelner Einflussgrössen.

2.3.5 K-Nearest Neighbors (KNN)

KNN (James et al., 2021, S. 126–131; scikit-learn developers, 2024) trifft Vorhersagen für einen neuen Datenpunkt (also für eine neue Prognose) auf Basis der Klassen der k nächsten Nachbarn. Die neue Beobachtung wird der Klasse zugeordnet, die unter diesen Nachbarn am häufigsten vorkommt.

Angenommen, wir wollen vorhersagen, ob ein Fussballspiel als Heimsieg, Unentschieden oder Auswärtssieg endet. Für ein neues Spiel werden die k bisherigen Spiele gesucht, die diesem Spiel statistisch am ähnlichsten sind. Wenn unter den nächsten 5 Nachbarn 3 Heimsiege, 1 Unentschieden und 1 Auswärtssieg waren, wird das neue Spiel als Heimsieg vorhergesagt.

Anwendungsbereich: KNN eignet sich besonders für Klassifikationsaufgaben mit kleinen bis mittelgrossen Datensätzen. Typische Anwendungen sind Mustererkennung, Bild- und Textklassifikation oder Empfehlungssysteme.

Vorteile: leicht verständlich, einfache Umsetzung, keine Annahmen über die Datenverteilung, intuitiv interpretierbar.

Nachteile: rechenaufwendig bei grossen Datensätzen, stark abhängig von der Skalierung der Merkmale, empfindlich gegenüber unwichtigen Variablen, kein eigenes Modell, daher eingeschränkt interpretierbar.

2.3.6 Naive Bayes

Naive Bayes (James et al., 2021, S. 135 + 138–143; scikit-learn developers, 2024) ist ein einfaches, aber effektives Klassifikationsverfahren, das auf dem Bayesschen Theorem basiert. Bei diesem Ansatz wird davon ausgegangen, dass die einzelnen Merkmale eines Datenpunkts unabhängig voneinander sind. Für jeden neuen Datenpunkt wird berechnet, wie wahrscheinlich jede mögliche Klasse ist. Die Klasse mit der höchsten Wahrscheinlichkeit wird dann als Vorhersage gewählt.

Anwendungsbereich: Naive Bayes eignet sich besonders für Probleme, bei denen viele Merkmale unabhängig voneinander sind. Typische Einsatzgebiete sind Textklassifikation (z. B. Spam-Erkennung in E-Mails), medizinische Diagnosen oder einfache Empfehlungssysteme.

Vorteile: schnelle Berechnung, gut für kleine Datensätze, liefert Wahrscheinlichkeiten als Vorhersage, leicht verständlich, robust gegenüber fehlenden Werten.

Nachteile: Annahme der Unabhängigkeit oft unrealistisch, Leistung sinkt bei stark korrelierten Features, eingeschränkte Flexibilität bei komplexen Zusammenhängen, einzelne Einflussgrössen schwer interpretierbar.

2.4 Beschreibung der ausgewählten Modelle

Im vorherigen Kapitel wurde beschreiben, wie vielfältig und komplex die Methoden zur Prognose von Sportereignissen sein können. Da es im Rahmen dieser Arbeit nicht möglich ist, alle statistischen und maschinellen Lernansätze im Detail zu vergleichen, wird der Fokus auf vier unterschiedliche Modelle gelegt, die exemplarisch verschiedene methodische Ansätze abbilden.

2.4.1 Buchmacher-Modell (Bet365)

Das erste Modell basiert nur auf den Wettquoten von Bet365. Diese Quoten spiegeln die implizite Wahrscheinlichkeitsannahmen des Wettanbieters wider, die in der Praxis auf komplexen statistischen Verfahren und maschinellen Lernmethoden beruhen (vgl. Christoffersson, 2023; Atta Mills et al., 2024, S. 2-5;). Für die vorliegende Arbeit wird das Modell vereinfacht, indem jeweils das Ergebnis mit der höchsten impliziten Wahrscheinlichkeit als Vorhersage herangezogen wird. Dieses Vorgehen macht die Quoten der Buchmacher zu einer wichtigen Benchmark, da sie sowohl auf umfangreichen Datenanalysen als auch auf Marktmechanismen beruhen (vgl. Constantinou & Fenton, 2013, 4-10; Hubáček et al., 2019).

2.4.2 Logistische Regression

Als zweites Modell wird ein echtes Machine-Learning-Modell gewählt, die Logistische Regression. Obwohl sie ihren Ursprung in der Statistik hat, wird sie in der modernen Forschung auch als Machine-Learning-Ansatz eingeordnet. Dies liegt daran, dass sie im überwachten Lernen eingesetzt wird, um datengetrieben Klassifikationsprobleme zu lösen (vgl. James et al., 2021, Kapitel 4). Damit bildet sie eine Brücke zwischen einem klassischen statistischen Verfahren und komplexeren Machine-Learning-Algorithmen.

Gerade diese Verbindung aus statistischem Ansatz und moderner Machine-Learning-Anwendung ist ein interessanter Grund, die logistische Regression für diese Arbeit zu wählen. Zudem ermöglicht sie, Wahrscheinlichkeiten für die Ergebnisse von Fussballspielen (Heimsieg, Unentschieden, Auswärtssieg) transparent zu modellieren. Gleichzeitig liefert sie leicht interpretierbare Ergebnisse. (James et al., 2021, S. 132–138)

Sie ist besonders geeignet, wenn nur wenige Einflussgrössen / Features vorliegen, wie in unserem Fall. (James et al., 2021, S. 132–138)

Damit dient sie als solide Basis, um die Leistung späterer, komplexerer Modelle mit der logistischen Regression zu vergleichen, während die wichtigsten Einflussgrössen gut verständlich bleiben.

2.4.3 Gradient Boosting

Als drittes Modell wird ein Gradient Boosting Modell gewählt. Im Gegensatz zur logistischen Regression, die nur auf einem einzelnen Modell basiert, handelt es sich hierbei um ein Ensemble-Verfahren, das mehrere Entscheidungsbäume sequenziell kombiniert (vgl. James et al., 2021, Kapitel 8). Dadurch kann Gradient Boosting komplexe und nichtlineare Zusammenhänge erfassen, die bei anderen ML-Modellen wie KNN, Naive Bayes oder SVM nur schwer darstellbar sind (vgl. James et al., 2021, Kapitel 4 + Kapitel 9).

Im Vergleich zu Random Forest, bei dem die Bäume parallel und unabhängig voneinander trainiert werden, lernt Gradient Boosting gezielt aus den Fehlern vorheriger Bäume und kann dadurch kleinere Unterschiede zwischen den Daten besser berücksichtigen, wodurch es häufig genauere Vorhersagen, als Random Forest liefert (vgl. James et al., 2021, Kapitel 8).

Für die Prognose von Fussballergebnissen eignet sich Gradient Boosting gut, da es komplexe und nichtlineare Zusammenhänge zwischen den Einflussgrössen modellieren kann und dadurch Unterschiede zwischen den Spielen sowie den möglichen Ergebnissen differenzierter abbildet (vgl. James et al., 2021, Kapitel 8; Hubáček et al., 2019; Atta Mills et al., 2024).

Während KNN stark von der Auswahl der Nachbarn abhängt, SVM bei stark überlappenden Klassen Schwierigkeiten hat und Naive Bayes auf die Annahme unabhängiger Variablen angewiesen ist (vgl. James et al., 2021, Kapitel 4 + Kapitel 9), liefert Gradient Boosting präzisere Vorhersagen und kann flexibel mit heterogenen Daten, also Daten, die auf unterschiedlichen Skalen liegen, umgehen (vgl. James et al., 2021, Kapitel 8).

Damit ergänzt das Gradient Boosting Modell, die logistische Regression ideal. Beides sind Machine-Learning-Modelle, aber während die logistische Regression eine solide, leicht interpretierbare Basis für Vorhersagen bietet (vgl. James et al., 2021, Kapitel 4), erfasst Gradient Boosting komplexere Muster und nichtlineare Zusammenhänge (vgl. James et al., 2021, Kapitel 8). Zusammen ermöglichen sie einen umfassenden Vergleich zwischen einfacheren und leistungsfähigeren ML-Ansätzen, wodurch neue Erkenntnisse für die Vorhersage von Fussballergebnissen gewonnen werden können.

2.4.4 Erweiterung durch ein eigenes Feature (EM-Score)

Neben den drei Modellen wird in einem vierten Schritt untersucht, ob ein zusätzlich selbstentwickeltes Feature, der sogenannte EM-Score, die Prognosequalität verbessern kann. Der EM-Score basiert auf einem selbst konstruierten Indikator, der zusätzliche Informationen zur Teamleistung in die Modelle einbringt, die vielleicht von den renommierten Buchmachern nicht berücksichtigt wurden. Dieses Vorgehen soll exemplarisch zeigen, welchen Einfluss Feature-Engineering, also die gezielte Auswahl und Konstruktion von Eingangsvariablen, auf die Genauigkeit und Zuverlässigkeit der Modellvorhersagen haben kann. (vgl. Christoffersson, 2023; S. 14-16)

2.5 Einflussfaktoren

Es gibt eine Vielzahl von Einflussfaktoren, die einen Einfluss auf die Vorhersage von Fussballergebnissen haben, dies wurden in der Forschung umfassend untersucht (Atta Mills et al., 2024). In diesem Kapitel werden die wichtigsten Erkenntnisse über diese Einflussfaktoren zusammengefasst.

Ein zentraler Aspekt ist der Heimvorteil, der in zahlreichen Studien nachgewiesen wurde. Mannschaften erzielen im eigenen Stadion überdurchschnittlich viele Punkte, was sowohl auf die Unterstützung der eigenen Fans, oft als „12. Mann «bezeichnet», als auch auf vertraute Spielbedingungen zurückgeführt wird (vgl. Pollard, 2006, 2008)

Darüber hinaus spielt die Teamstärke eine entscheidende Rolle. Sie wird häufig anhand von Leistungskennzahlen wie Elo-Ratings (ähnlich wie im Schach), Marktwerten oder aggregierten Spielerstatistiken bewertet (vgl. Parim et al., 2021; Moustakidis et al., 2023).

Ergänzend dazu wird auch die Form der Mannschaften berücksichtigt, also die Ergebnisse der letzten Spiele, die Hinweise auf das kurzfristige Leistungsniveau geben (vgl. Parim et al., 2021).

Ein weiterer relevanter Faktor sind individuelle Spielerleistungen. Verletzungen oder Sperren wichtiger Spieler können die Chancen auf einen Sieg erheblich beeinflussen (vgl. Moustakidis et al., 2023).

Zudem werden in modernen Analysen vermehrt Metriken wie Expected Goals eingesetzt, die anzeigen, wie hoch die erwarteten Tore eines Teams in einem Spiel waren. Dadurch lässt sich die Performance verschiedener Mannschaften besser vergleichen, da nicht nur das tatsächliche Resultat berücksichtigt wird (vgl. Parim et al., 2021; Moustakidis et al., 2023).

Auch direkte Vergleiche zwischen Mannschaften werden in der Literatur als Einflussfaktor diskutiert. Historische Begegnungen können psychologische oder taktische Muster aufzeigen, wie z. B. Angstgegner oder Lieblingsgegner (vgl. Parim et al., 2021).

Schliesslich spielen auch externe Faktoren wie Wetterbedingungen, Reisebelastung oder die Bedeutung des Spiels im Saisonverlauf (z. B. Abstiegskampf, Endphase eines Turniers) eine Rolle (vgl. Moustakidis et al., 2023).

Ausserdem zeigt sich, dass Buchmacher Unentschieden in der Regel seltener favorisieren, da dieses Ergebnis statistisch weniger häufig auftritt und schwer vorherzusagen ist (vgl. Forrest & Simmons, 2002). Deswegen liegen die Quoten für Unentschieden oft höher, da die implizite Wahrscheinlichkeit relativ niedrig ausfällt.

Insgesamt zeigt die Literatur, dass Fussballergebnisse durch ein Zusammenspiel zahlreicher Faktoren bestimmt wird, die sowohl auf Team.-, als auch auf Individualebene wirken. Für datengetriebene Modelle bedeutet dies, dass die Auswahl und Aufbereitung relevanter Variablen (Feature Engineering) entscheidend für die Prognosequalität ist (vgl. Parim et al., 2021; Moustakidis et al., 2023).

2.6 Forschungslücken und Zielsetzung

Die Literatur zu Prognosemodellen im Fussball verfolgt eine Vielzahl an Ansätzen, von klassischen statistischen Verfahren wie der Poisson-Regression bis hin zu modernen Machine-Learning-Algorithmen wie Random Forests oder neuronalen Netzen (vgl. Christoffersson, 2023; Király & Qian, 2017; Atta Mills et al., 2024). Trotzdem lassen sich folgende Forschungslücken identifizieren:

1. **Systematischer Vergleich unterschiedlicher Modelltypen:** Viele Studien betrachten entweder klassische statistische Modelle oder Machine-Learning-Ansätze. Eine systematische Untersuchung, wie unterschiedliche Modelltypen die Vorhersage von Spielergebnissen für bestimmte Saisons beeinflussen, ist bisher selten.

2. **Einfluss zusätzlicher Variablen:** Zwar werden Faktoren wie Heimvorteil, Teamstärke oder Form berücksichtigt, doch die Integration zusätzlicher, bisher nur wenig dokumentierter Einflussgrössen ist bislang nicht systematisch untersucht worden.

3. **Benchmarking mit Buchmacherquoten:** Buchmacherquoten enthalten implizit das kollektive Wissen des Marktes, werden jedoch nur selten als Vergleichsstandard für datengetriebene Modelle genutzt.

Auf Basis dieser Forschungslücken verfolgt die Arbeit folgende Ziele:

1. **Systematische Analyse:** Es werden drei unterschiedliche Modelltypen (Buchmacher, logistische Regression, Gradient Boosting) untersucht, um ihre Prognosequalität anhand gängiger Kennzahlen für die Spiele der Bundesliga-Saison 2024/2025 zu bewerten.

2. **Integration eines neuen Features (EM-Score):** Es wird geprüft, ob ein selbst entwickelter Indikator, der von den renommierten Buchmachern bisher nicht berücksichtigt wurde, die Vorhersageleistung bestehender Machine-Learning-Modelle verbessern kann.

3. **Bewertung anhand historischer Daten:** Durch die Analyse bereits gespielter Partien sowie den Vergleich mit Buchmacherquoten liefert die Arbeit Erkenntnisse über die Genauigkeit und Grenzen der untersuchten Modelle.

3 Methodik

Auf Basis der Literaturrecherche, Gesprächen mit der betreuenden Lehrperson sowie der intensiven Auseinandersetzung mit der Thematik entstand zunächst die Idee, drei unterschiedliche, bereits bestehende Modelle anhand gängiger Kennzahlen für die Saison 2022/2023 miteinander zu vergleichen.

Zudem sollte ein Einflussfaktor identifiziert werden, der von den gängigen Buchmachern bisher nicht berücksichtigt wurde.

Da der Zugriff auf bestehende Modelle eingeschränkt war, entweder standen diese nicht vollständig zur Verfügung oder wiesen Fehler auf, entschied ich mich, ein Referenzmodell schrittweise anzupassen und zu erweitern. Auf diese Weise konnten die Modelle dennoch systematisch analysiert werden.

Im Verlauf der Arbeit wurde zudem der ursprünglich geplante Vergleich der Saison 2022/2023 auf die Saison 2024/2025 verschoben. Dadurch konnte vermieden werden, dass ein Daten-Leakage entsteht, also das Modell auf bereits bekannte Ergebnisse zurückgreift (vgl. James et al., 2021, S. 204–217). Somit wurde eine realitätsnähere Abbildung gewährleistet.

Das methodische Vorgehen bestand darin, zunächst eine umfassende Literaturrecherche durchzuführen, anschliessend schrittweise Anpassungen am Referenzmodell vorzunehmen und zwei unterschiedliche Machine-Learning-Ansätze miteinander zu vergleichen. Abschliessend wurde ein eigenes Feature entwickelt, das in Kapitel 3.2 detailliert beschrieben wird.

3.1 Kriterien für den Modellvergleich

Zur systematischen Bewertung der Prognosemodelle wurden gängige Kennzahlen aus der Klassifikation verwendet. Dabei wurden ausschliesslich die Ergebnisse Heimsieg, Unentschieden und Auswärtssieg berücksichtigt, jedoch nicht die exakte Anzahl der erzielten Tore.

Eine der gängigen Kennzahlen ist der **Accuracy-Wert**, der den Anteil der korrekt vorhergesagten Spiele an allen Spielen widerspiegelt:

Accuracy = Anzahl korrekt vorhergesagten Spiele/ Gesamtanzahl der Spiele

Eine weitere Kennzahl ist der **Precision-Wert**, der den Anteil der korrekt vorhergesagten Ergebnisse innerhalb einer Kategorie (z. B. Heimsieg) an allen Vorhersagen dieser Kategorie angibt

Precision -Wert Heimsieg= korrekt vorhergesagte Heimsieg/ alle Vorhersagen Heimsieg

Der **Recall-Wert** zeigt den Anteil korrekt vorhergesagter Ergebnisse innerhalb einer Kategorie, gemessen an allen tatsächlich eingetretenen Ereignissen dieser Kategorie.

Recall-Wert = korrekt vorhergesagte Heimsieg/ Gesamtanzahl der Heimsiege

Zum Schluss wird der **F1-Score** betrachtet, der das harmonische Mittel von Precision und Recall darstellt, um so ein ausgewogenes Mass zwischen beiden Kennzahlen zu erhalten. Der F1-Score liegt zwischen 0 und 1:

- 1 bedeutet perfekte Vorhersagen, keine Fehler und alle relevanten Ereignisse wurden korrekt erkannt.
- 0 bedeutet sehr schlechte Vorhersagen, viele falsche Treffer oder viele nicht erkannte Ereignisse.

F1-Score = 2*((Precision *Recal)/ (Precision +Recall))

Diese Metriken ermöglichen nicht nur den Vergleich der Gesamtgenauigkeit der Modelle, sondern auch die Bewertung ihrer Fähigkeit, bestimmte Ergebnisarten zuverlässig vorherzusagen. Dadurch lassen sich die Modelle deutlich differenzierter analysieren (vgl. IBM, o. J.).

3.1.1 Beispiel der Berechnungen der Kriterien

Angenommen, wir haben 10 Spiele in einer Saison mit folgenden tatsächlichen Ergebnissen:

Spiel	Ergebnis	Vorhersage Model 1	Vorhersage Model 2
1	Heimsieg	Heimsieg	Heimsieg
2	Heimsieg	Heimsieg	Auswärtssieg
3	Unentschieden	Heimsieg	Heimsieg
4	Heimsieg	Heimsieg	Auswärtssieg
5	Heimsieg	Heimsieg	Unentschieden
6	Auswärtssieg	Heimsieg	Auswärtssieg
7	Heimsieg	Heimsieg	Auswärtssieg
8	Unentschieden	Heimsieg	Unentschieden
9	Heimsieg	Heimsieg	Heimsieg
10	Heimsieg	Heimsieg	Heimsieg

Richtiges Ereignis getroffen

Modell A: sagt immer „Heimsieg“ voraus.

- **Accuracy:** Das Modell trifft 7 von 10 Spielen richtig → 70 %
- **Precision Unentschieden:** 0 richtig vorhergesagte Unentschieden durch 0 Vorhersagen von einem Unentschieden → 0 %
- **Recall Unentschieden:** 0 richtig vorhergesagte Unentschieden durch 2 tatsächlich eingetretene Unentschieden → 0 %
- **F1-Score Unentschieden:** 2 * (0 * 0) / (0 + 0) = 0

Analyse: Die Accuracy allein (70 %) sieht stark aus, aber das Modell erkennt kein Unentschieden korrekt (Recall 0 %). F1-Score zeigt ein miserables Bild (0), da Precision für ein Unentschieden auch 0 % beträgt.

Modell B: sagt „Heimsieg“, „Auswärtssieg“ oder „Unentschieden“ voraus.

- **Accuracy:** Das Modell trifft 5 von 10 Spielen richtig → 50 %
- **Precision Unentschieden:** 1 richtig vorhergesagtes Unentschieden durch 2 Vorhersagen von einem Unentschieden → 50 %
- **Recall Unentschieden:** 1 richtig vorhergesagte Unentschieden durch 2 tatsächlich eingetretene Unentschieden → 50 %
- **F1-Score Unentschieden:** 2 * (0.5 * 0.5) / (0.5 + 0.5) = 0.5

Analyse: Die Accuracy allein (50 %) sieht mittelmässig aus, aber das Modell erkennt die Hälfte aller Unentschieden korrekt (Recall 50 %). F1-Score zeigt ein ähnliches Bild (50%), da Precision für ein Unentschieden auch 50 % beträgt.

Wenn man Modelle nur anhand des **Accuracy-Wertes** bewertet, könnte das Modell 1, das immer nur Heimsiege vorhersagt, und damit eine Accuracy von 70 % erreicht und auf den ersten Blick besser erscheinen als Modell 2. Bei genauerer Betrachtung zeigt sich jedoch, dass das Modell für andere Ergebnisarten völlig unbrauchbar ist. Deshalb sind Precision, Recall und F1-Score für die Beurteilung der Prognosequalität entscheidend (vgl. IBM, o. J.).

3.1.2 Vergleich der aggregierten F1-Scores

Damit die Modelle nicht nur anhand ihres Accuracy-Werts beurteilt werden, sondern auch die Leistung über alle Klassen hinweg betrachtet werden kann, wird ein aggregierter F1-Score gebildet. Dieser fasst die F1-Scores aller Klassen zusammen und ermöglicht so eine ganzheitliche Bewertung der Prognosequalität, ohne sich ausschliesslich auf den Accuracy-Wert zu konzentrieren.

Dazu gibt es drei gängige Möglichkeiten:

3.1.2.1 Macro-F1-Score

Der Durchschnittliche F1-Scores aller Klassen.

Beispiel: F1_Heimsieg = 0.65, F1_Unentschieden = 0.15, F1_Auswärtssieg = 0.5 Macro-F1 = (0.65 + 0.15 + 0.5) / 3 ≈ 0.433
Dieser Wert zeigt die durchschnittliche Modellleistung über alle Klassen hinweg, unabhängig von der Häufigkeit der einzelnen Klassen (vgl. KDnuggets, 2023).

3.1.2.2 Weighted-F1-Score

Dieser F1-Scores wird gebildet in dem die F1-Scores der einzelnen Klassen, entsprechend ihrer Klassenhäufigkeit in den Daten, gewichtet wird.

Beispiel: Gleiche F1-Scores wie oben. Bei 70 % Heimsiege, 10 % Unentschieden, 20 % Auswärtssiege

Weighted-F1 = (0.65 *0.7)+ (0.15*0.1) + (0.5*0.2) = 0.57

Dieser Wert berücksichtigt die tatsächliche Verteilung der Klassen und gewichtet dominante Klassen stärker (vgl. KDnuggets, 2023).

3.1.2.3 Micro-F1-Score

Hier wird nicht für jede einzelne Klasse ein F1-Score gebildet und dann aggregiert, sondern der F1-Score wird direkt über alle Klassen hinweg berechnet. Dazu werden Precision und Recall über alle Klassen zusammen bestimmt. Dieser Wert liefert eine einheitliche Kennzahl für das gesamte Modell und ist besonders sinnvoll bei unbalancierten Datensätzen, da die Gesamtleistung über alle Klassen quantifiziert wird (vgl. KDnuggets, 2023).

3.1.2.4 Auswahl des geeignetsten F1 Scores

Welche der drei Varianten für die Bewertung am meisten Sinn macht, hängt von der Zielsetzung ab. Für meine Arbeit könnte man intuitiv den Micro-F1-Score wählen, da er besonders für unbalancierte Daten geeignet ist, wie das in unserem Beispiel, mit wenigen Unentschieden, der Fall ist.

Da der Micro-F1-Score jedoch nur für das gesamte Modell berechnet werden kann und die Werte für die einzelnen Klassen nicht separat ablesbar sind, wird in dieser Arbeit der Macro-F1-Score verwendet. So können zusätzlich Informationen über die Leistung der einzelnen Klassen gewonnen werden, was in vielen Fällen sehr sinnvoll ist, um das Modell differenziert zu bewerten.

Zudem gewichtet der Macro-F1-Score, im Gegensatz zum Weighted-F1-Score**,** alle Klassen gleich, wodurch auch seltene Klassen wie das Unentschieden fair berücksichtigt werden (vgl. KDnuggets, 2023).

3.2 Vorgehen zur Identifikation eines zusätzlichen Einflussfaktors (Feature)

Ursprünglich wurde auf Basis der Saison 2022/2023 ein WM-Feature konstruiert (Anhang 4 + 5), dass die Leistungen der Spieler bei der Weltmeisterschaft berücksichtigt. Da im weiteren Verlauf der Arbeit jedoch die Saison 2024/2025 analysiert wurde, wurde das Feature entsprechend angepasst und in ein EM-Feature überführt, das die Ergebnisse der Europameisterschaft 2024 abbildet.

Neben den Variablen des Referenzmodells wurde untersucht, ob zusätzliche Einflussgrössen die Prognoseleistung verbessern können. Das Vorgehen bestand aus folgenden Schritten:

1. **Literaturrecherche:** Identifikation potenziell relevanter Variablen, die in den Quoten der Buchmacher nicht bzw. kaum berücksichtigt wurden.

2. **Feature-Konstruktion:** Entwicklung des EM-Scores. Hierbei erhielt jeder Spieler, der an der EM 2024 teilgenommen hat, einen Wert, abhängig vom Abschneiden seiner Nationalmannschaft. Entscheidend war dabei, wie weit die Mannschaft im Vergleich zu den Erwartungen kam. Für jedes Bundesligateam wurde der Wert anschliessend summiert (genaue Berechnung im Kapitel 3.2.1).
    - Die Konstruktion basiert auf der Annahme, dass Spieler nach einem erfolgreichen Turnier mit einem positiven Gefühl in die neue Saison starten.
    - Alternativ hätte man annehmen können, dass Spieler, die mehr Spiele für ihre Nationalmannschaft absolviert haben, schlechter performen, da sie weniger ausgeruht in die Bundesliga zurückkehren.
    - In dieser Arbeit wird jedoch die Annahme getroffen, dass die positiven Gefühle nach einem erfolgreichen Turnier die zusätzliche Belastung der Spieler kompensieren, da zwischen EM und Bundesliga ausreichend Erholungszeit liegt.
3. **Integration in die Modelle:** Der EM-Score allein genügt nicht, um die Prognoseleistung zu verbessern. Deshalb wurde er in Verbindung mit den Wettquoten der Buchmacher eingesetzt (vgl. Kapitel 4.4).
4. **Evaluation:** Analyse der Auswirkungen auf Accuracy, Precision, Recall und F1-Score.

3.2.1 Beispielhafte Berechnung des EM-Scores für ausgewählte Teams

Im Folgenden wird die Berechnung des EM-Scores exemplarisch für zwei Bundesligavereine dargestellt. Zunächst wird eine Tabelle erstellt (Anhang 6), die alle Nationalmannschaften auflistet, die an der EM teilgenommen haben, geordnet nach Weltranglistenplatz vor der EM 2024 (vgl. FIFA, o. J.). Anschliessend wird die Mindesterwartung jeder Mannschaft vor dem Turnier ermittelt:

- Die 4 besten Mannschaften der EM, laut Weltrangliste, sollten mindestens das Halbfinale erreichen.
- Die Plätze 5–8 sollten mindestens das Viertelfinale erreichen.
- Die Plätze 9–16 sollten mindestens das Achtelfinale erreichen.
- Für die übrigen Mannschaften wäre ein Weiterkommen in die KO-Spiele bereits eine Überraschung und übertrifft somit schon die Mindesterwartung.

Der EM-Score pro Nationalmannschaft wird anschliessend berechnet, indem 0,1 Punkte pro Runde hinzugefügt oder abgezogen werden, je nachdem, ob die Mannschaft eine Runde weiter als erwartet kam oder früher ausschied. Der EM-Sieger erhält zusätzlich 0,2 Punkte, da ein Turniersieg einen besonderen Wert für die Spieler darstellt.

Schauen wir uns das konkret für 2 Länder an. Spanien hatte laut Weltrangliste die Mindesterwartung das Viertelfinale zu erreichen, da sie jedoch bis in den Final kamen und dort sogar als Europameister vom Platz gingen, gab es 0.4 Punkte. Diese setzten sich wie folgt zusammen: 0.1 Punkt fürs Halbfinale, 0.1 Punkte für das Erreichen des Finals und 0.2 für den Sieg der Europameisterschaft.

Belgien hingegen lag vor dem Turnier auf dem 3. Weltranglistenplatz (2. bestplatziertes Team der EM) und hatte somit die Mindesterwartung, das Halbfinale zureichen. Jedoch flog man bereits im Achtelfinale raus was zu einem EM-Score von -0.2 führte, -0.1 pro Runde, die man früher rausflog. In diesem Beispiel 2*-0.1, da man ja bereits im Achtelfinale rausflog und somit das Viertel respektive Halbfinale verpasste.

Die vollständigen Werte für alle Nationalmannschaften finden sich in Anhang 6.

Anschliessend wird der EM-Score für jedes Bundesligateam berechnet, indem die Werte aller Spieler aufsummiert werden, die an der EM teilgenommen haben.

3.2.1.1 Beispiel FC Bayern München

Spieler an der EM: Joshua Kimmich, Thomas Müller, Jamal Musiala, Manuel Neuer, Leroy Sané (Deutschland), Harry Kane (England), Kingsley Coman, Dayot Upamecano (Frankreich), Matthijs de Ligt (Niederlande), Konrad Laimer (Österreich) (vgl. 90min.de, 2024).

Rechnung:

- Deutschland: 5 Spieler * 0,1 Punkte = 0,5
- England: 1 Spieler * 0,1 Punkte = 0,1
- Frankreich: 2 Spieler * 0 Punkte = 0
- Niederlande: 1 Spieler * 0,1 Punkte = 0,1
- Österreich: 1 Spieler * 0 Punkte = 0

Gesamtscore FC Bayern: 0,5 + 0,1 + 0 + 0,1 + 0 = **0,7**

3.2.1.2 Beispiel VfL Wolfsburg

Spieler an der EM: Koen Casteels, Aster Vranckx (Belgien), Joakim Mæhle, Jonas Wind (Dänemark), Lovro Majer (Kroatien), Patrick Wimmer (Österreich), Cédric Zesiger (Schweiz), Václav Černý (Tschechien) (vgl. 90min.de, 2024).

Rechnung:

- Belgien: 2 Spieler * -0,2 Punkte = -0,4
- Dänemark: 2 Spieler * 0 Punkte = 0
- Kroatien: 1 Spieler * -0,2 Punkte = -0,2
- Österreich: 1 Spieler * 0 Punkte = 0
- Schweiz: 1 Spieler * 0,1 Punkte = 0,1
- Tschechien: 1 Spieler * 0 Punkte = 0

Gesamtscore VfL Wolfsburg: -0,4 + 0 - 0,2 + 0 + 0,1 + 0 = **-0,5**

Der EM-Score für aller Bundesligateams finden sie im Anhang 7

3.3 Datengrundlage und Quellen

Für die Analyse wurden verschiedene Datenquellen genutzt, die im Literaturverzeichnis dokumentiert sind. Die Spieldaten für die Bundesliga-Saisons 2020–2025 stammen aus einem öffentlich zugänglichen ML-Modell, das sowohl Quoten als auch die tatsächlichen Spielergebnisse bereitstellt (vgl. Andretta, o. J.).

Zusätzlich wurde versucht, ein statistisches Modell auf Basis von Elo-Ratings zu integrieren. Da die hierfür verfügbaren Daten jedoch nur unvollständig vorlagen und nicht im CSV-Format bereitgestellt wurden (vgl. Bundesliga-Prognose.de, o. J.), konnte dieser Ansatz nicht weiterverfolgt werden. Eine Anfrage per E-Mail (Anhang 1) blieb unbeantwortet, weshalb ein alternatives Referenzmodell genutzt wurde.

Um die Wettquoten besser interpretieren zu können, wurde zudem eine direkte Anfrage an Bet365 gestellt, um Einblick in die Berechnungsmethoden aktueller und historischer Quoten zu erhalten. Diese Anfrage blieb jedoch unbeantwortet (Anhang 2). Daher wurden alternative, öffentlich verfügbare Quellen herangezogen (vgl. Kolibrisport, 2025). Welche Features die Buchmacher konkret nutzen, bleibt damit unklar, was plausibel ist, da die genaue Berechnung einen zentralen Bestandteil ihres Geschäftsgeheimnisses darstellt.

4 Empirische Analyse

4.1 Referenzmodell

Das in dieser Arbeit verwendete Referenzmodell (vgl. Andretta, o. J.) ist ein Machine-Learning-Modell, das darauf trainiert wurde, vorherzusagen, ob in einem zukünftigen Fussballspiel mehr oder weniger als 2.5 Tore fallen. Ein „halbes Tor“ klingt zunächst verwirrend, bedeutet hier aber, dass bei genau zwei Toren die Wette verloren und bei genau drei Toren gewonnen wird.

Die ursprünglich verwendeten Daten liegen als CSV-Dateien im Ordner **data/raw** vor. Sie umfassen Spieldaten und Wettquoten aus den europäischen Topligen, also Ligue 1 (Frankreich), Bundesliga (Deutschland, Anhang_1), La Liga (Spanien), Premier League (England) und Serie A (Italien). Für die Bundesliga befindet sich ein Auszug der entsprechenden CSV-Datei im Anhang 3, die übrigen Ligen haben eine identisch aufgebaute Datei.

Nach dem Einlesen werden die Rohdaten mittels des Skripts **data_preprocessing.py** verarbeitet. Hierbei werden zunächst verschiedene Features erzeugt. Unter anderem wird die Saisonzugehörigkeit anhand des Spieltermins bestimmt (determine_season(df['Date'])) und eine Zielvariable «Over2.5» erstellt (df["Over2.5"] = np.where(df["FTHG"] + df["FTAG"] > 2, 1, 0)) (vgl. Andretta, o. J.).

Zusätzlich werden Mittelwerte und gleitende Durchschnitte der letzten fünf Spiele berechnet, unter andrem für erzielte Tore oder die Häufigkeit von Over-2.5-Ergebnissen (Andretta, o. J.). Auf diese Weise werden kurzfristige Leistungstrends der Teams in den Datensätzen berücksichtigt.

Im Anschluss werden die Daten bereinigt, indem überflüssige Spalten wie zum Beispiel FTHG (Full Time Home Goals) entfernt werden (drop_useless_columns(df, ['FTHG','FTAG','HTHG','HTAG'])), da sie das Modell direkt auf das Spielgeschehen lenken würden und somit zu einem Informationsvorteil führen würden, der im echten Vorhersagefall nicht verfügbar ist. Zudem werden Spalten mit zu vielen Nullwerten gelöscht und die verbleibenden Zeilen, die fehlenden Werte haben, werden entfernt (handle_missing_values(df)) (vgl. Andretta, o. J.).

Die Auswahl der relevantesten Features erfolgt anschliessend über die mRMR-Methode, kombiniert mit hierarchischem Clustering (selected_features = feature_selection(df, num_features=20, clustering_threshold=0.5)). Dabei werden stark korrelierte Features erkannt, und aus jedem Cluster wird das Feature mit der höchsten Varianz beibehalten (vgl. Andretta, o. J.). Dies stellt sicher, dass nur informative, aber keine redundanten Variablen im Modell bleiben.

Beispiele für ausgewählte Features sind:

- **Spielausgang** (FTR, HTR)
- **letzte Form der Teams** (Last5HomeOver2.5Perc, Last5AwayOver2.5Perc)
- **Torstatistiken der letzten Spiele** (AvgLast5HomeGoalsScored, AvgLast5AwayGoalsScored)
- **ausgewählte Wettquoten** (B365CAHH)
- **Datum des Spiels** (Date)
- **Liga** (Div)
- **Spielpaarung** (HomeTeam, AwayTeam)

Diese Features geben dem Modell sowohl historische Leistungsdaten als auch marktbasierte Einschätzungen der Spiele, wodurch Vorhersagen deutlich realistischer werden.

Die so verarbeiteten Daten werden in einer neuen CSV-Datei im Ordner **data/processed** gespeichert und dienen als Input für das Modelltraining.

Für das Training der Machine-Learning-Modelle wird das Skript **train_models.py** verwendet. Dabei kommen mehrere Klassifikationsalgorithmen zum Einsatz, darunter Logistische Regression (LogisticRegression), Random Forest (RandomForestClassifier), XGBoost (XGBClassifier), HistGradientBoosting (HistGradientBoostingClassifier) sowie Support Vector Machines (SVC) (vgl. Andretta, o. J.).

Die Hyperparameter der Modelle werden über HalvingGridSearchCV und teilweise über BayesSearchCV optimiert. (grid_search = HalvingGridSearchCV(...)) (vgl. Andretta, o. J.). Dies erlaubt, die Parameter effizient auf Basis von Cross-Validation zu justieren, sodass jedes Modell bestmöglich an die Trainingsdaten angepasst wird, ohne zu overfitten (scikit-learn developers, 2024).

Anschliessend werden die Wahrscheinlichkeiten für die Klassen „Over 2.5“ und „Under 2.5“ von jedem Modell in einem Voting-Classifier kombiniert (voting_clf = VotingClassifier(estimators=[...], voting='soft')). In dem die Wahrscheinlichkeiten gemittelt werden, sodass nicht einfach eine Mehrheitsentscheidung getroffen wird, sondern die Stärke der Überzeugung der einzelnen Modelle in das Gesamtergebnis einfliesst. So kann der Voting-Classifier die kombinierte Wahrscheinlichkeit ausgeben und trifft basierend darauf die finale Entscheidung Over oder Under 2.5 (vgl. Andretta, o. J.).

4.2 Modell 1: Buchmacher-basierte Vorhersage

Da das Referenzmodell nicht direkt eine allgemeine Spielvorhersage liefert, sondern nur die Wahrscheinlichkeit für mehr als 2,5 Tore berechnet, muss es angepasst werden.

Als eine 1. Annäherung könnte man die Wahrscheinlichkeit der Buchmacher aus den Quoten berechnen, anschliessend, dass Ergebnis mit der höchsten Wahrscheinlichkeit als vorhersage nutzen. Alternativ kann man das Ereignis mit der kleinsten Quote (hat gleichzeitig die höchste Wahrscheinlichkeit, da (Wahrscheinlichkeit = 1/ Quote gilt) (vgl. Christoffersson, 2023, S. 9)) nehmen und als Ereignis vorhersagen.

Dazu muss man zuerst die Daten der Spielzeit 2024/2025 laden, dies erreicht man in dem man folgenden Befehl ausführt:

python scripts/data_acquisition.py --leagues D1 --seasons 2425 --raw_data_output_dir data/raw2425

Dadurch werden aus der CSV Datei D1_mergerd (D1 steht für Deutschland 1, also 1. Bundesliga) nur die Spiele geladen, die in der Saison 2024/2025 in der Bundesliga gespielt wurden. Dazu wird ein neuer Ordner raw2425, im Ordner Data erstellt. Anschliessend wird dieser Ordner respektive die Datei darin eingelesen

Bundesliga_Saison_2425 = pd.read_csv("data/raw2425/D1_merged.csv")

und die tatsächlichen Resultate extrahiert.

Tatsächliches_Resultat_2425 = Bundesliga_Saison_2425["FTR"]

Für die Vorhersage wird ermittelt, welches der drei Features Bet365H (steht für BetWin 365 Quote für Heimsieg) Bet365D (steht für BetWin 365 Quote für Unentschieden) und Bet365A (steht für BetWin 365 Quote für Auswärtssieg) den niedrigsten Wert hat und somit die höchste Wahrscheinlichkeit aufweist.

```
Bundesliga_Saison_2425["Favorit_Buchmacher"] =
Bundesliga_Saison_2425[["B365H", "B365D",
"B365A"]].idxmin(axis=1).str.extract(r"B365(.)")[0]
```

Anschliessend wird die Genauigkeit der Vorhersagen berechnet

```
accuracy_2425 = accuracy_score(Tatsächliches_Resultat_2425,
Bundesliga_Saison_2425["Favorit_Buchmacher"])
```

und ein Klassifikations-Report mit den Kennzahlen Precision, Recall, F1-Score und Support erstellt

```
report_2425 = classification_report(Tatsächliches_Resultat_2425,
Bundesliga_Saison_2425["Favorit_Buchmacher"], output_dict=True)
```

4.2.1 Visualisierung der Resultate

Zur besseren Übersicht wird der Report als Tabelle formatiert:

```
finaler_report = pd.DataFrame(report_2425).T

finaler_report = finaler_report.drop(index=[ "accuracy", "macro avg", "weighted avg"],
errors="ignore")

finaler_report = finaler_report[["precision", "recall", "f1-score", "support"]].round(2)

finaler_report.index.name = "Klasse"

finaler_report.reset_index(inplace=True)
```

Auf die detaillierte Beschreibung der Grafiken wird in diesem Kapitel verzichtet, da sie für die Zielsetzung der Arbeit nicht relevant ist. Der vollständige Code samt Erklärung ist im Anhang zu finden, ebenso die Visualisierung des Classification Reports und des Accuracy-Wertes (siehe Anhang 8, Zeilen 93–130).

4.2.1.1 Classification Report

Classification Report (Saison 2024/2025)

Klasse	precision	recall	f1-score	support
A	0.57	0.5	0.54	111.0
D	0.0	0.0	0.0	77.0
H	0.49	0.86	0.63	118.0

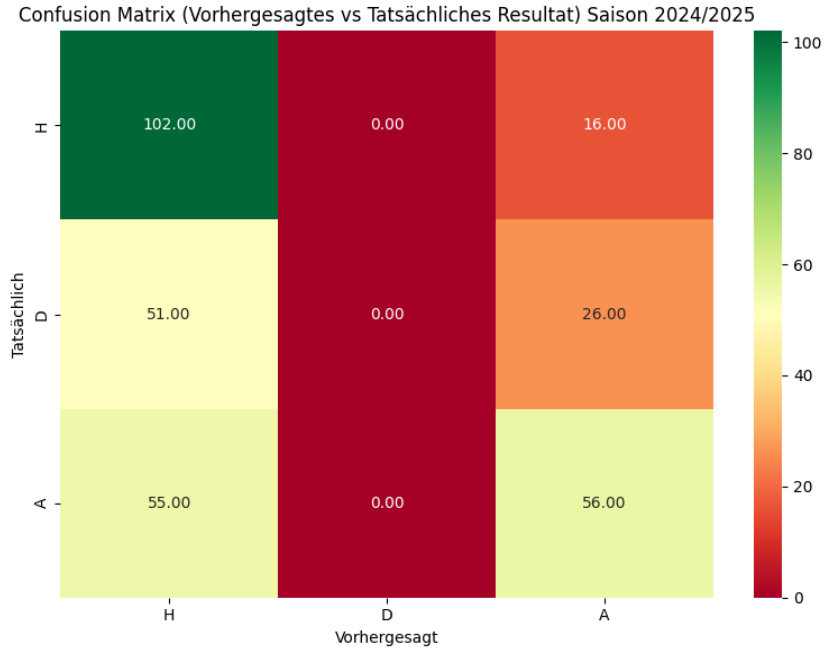
Accuracy: 0.52

Die Genauigkeit dieses Ansatzes liegt bei ungefähr 52%. Bedeutet die Buchmacher von Bet365 liegen mindestens jedes 2. richtige Resultat mit dem hervorgesagten Resultat.

Die Spalte Klasse zeigt die Resultate für die jeweiligen Klassen A (Auswärtssieg), D (Unentschieden) und H (Heimsieg). Die Spalte Precision zeigt wie viel Prozent eines hervorgesagtes Spielereignisses (z.B. Heimsieg) auch wirklich Heimsieg waren. Recall zeigt wie viel Prozent von einem tatsächlichen Spielereignisses (z.B. Heimsieg) auch so hervorgesagt wurden. Der f1-score ist eine Kombination der Werte Precision und recall, er zeigt die Balance zwischen den beiden Werten, je höher die Zahl, desto besser das Modell. Die Spalte support zeigt, wie oft ein Spielergebnis tatsächlich eingetroffen ist.

4.2.1.2 Konfusionsmatrix

Da diese Werte an sich noch wenig detailreich sind, wird im nächsten Schritt eine Konfusion-Matrix erstellt und visualisiert (siehe Anhang 8, Zeilen 45-60)

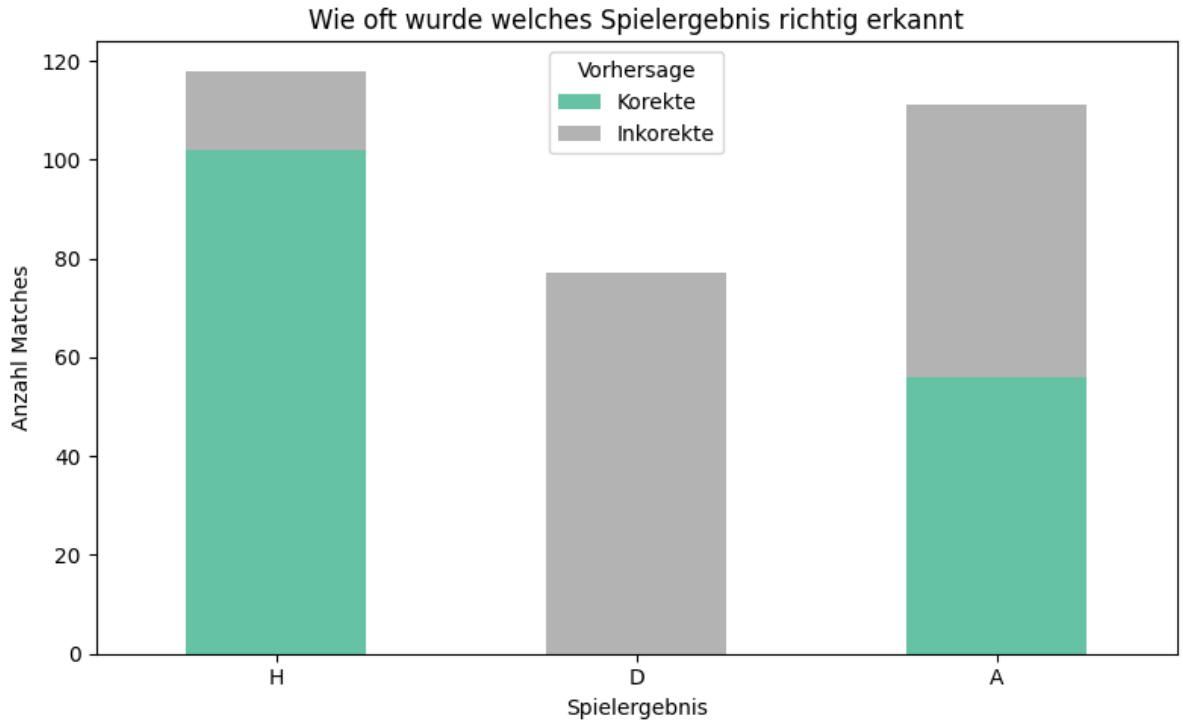


Daraus kann man ablesen welche Ergebnisse vorhergesagt wurden und welche tatsächlich eingetroffen sind. Im Grunde zeigt die Konfusion-Matrix, im Detail wie der Precision Wert zustande kommt.

Die erste Spalte zeigt wie sich die Vorhersagen für ein Heimsieg, aufteilt auf die tatsächlichen Resultate. Bedeutet also 102 Spiele die als Heimsiege vorhergesagt wurden, waren auch tatsächlich Heimsiege, 51 Spiele endeten Unentschieden, obwohl ein Heimsieg vorhergesagt wurden und bei 55 Spielen gewann die Auswärtsmannschaft. Das heisst das Modell hat insgesamt 208 Spiele (102+51+55) als Heimsiege vorhergesagt. Die anderen Spalten funktionieren analog.

4.2.1.3 Balkendiagramm der Vorhersagegenauigkeit

Zum Schluss wird noch ein Balkendiagramm erstellt und visualisiert, (siehe Anhang 8, Zeilen 68-83) das zeigt wie viele der eingetroffenen Spielereignisse auch als solches vorhergesagt wurden. Dies ist im Grunde der Recall Wert, der nun einfach und überschaubar, in einem Balkendiagramm dargestellt wird.



Während das Modell 1 lediglich die Buchmacherquoten von Bet365 für Spiel vorhersagen nutzt und somit nur ein erste Vorhersagerichtung vorgibt. Ist das Ziel dieser Arbeit einen erweiterten Ansatz, es sollen nämlich verschiedene Machine-Learning-Modelle miteinander verglichen werden, um deren Vorhersagegenauigkeit bezüglich der Fussballspiele der Bundesliga-Saison 2024/25 zu evaluieren. Modell 1 liefert dafür keine Grundlage, da es nicht um ein Machine-Learning-Modell handelt, da das Modell nicht mit historischen Daten trainiert wird. Aus diesem Grund wurde ein 2. Modell entwickelt, das auf dem Referenzmodell aufbaut.

4.3 Modell 2: Logistische Regression

Analog zum 1. Modell werden die Spiele der Bundesliga-Saison 2024/25, aus folgendem Ordner **AIFootballPredictions\data\raw\D1_merged.csv** gezogen und in einem neuen Ordner **AIFootballPredictions\data\raw2425\D1_merged.csv** gespeichert. Diese bilden die Testdaten während die Rohdaten **AIFootballPredictions\data\raw\D1_merged.csv** (ausser Spielzeit 2024/2025) als Trainingsdaten dienen.

4.3.1 Features

Nun werde verschiedene Features aus der CSV-Datei extrahiert, die später als Grundlage für das Training des Modells dienen, um so vorhersagen zumachen. Die richtige Auswahl dieser Features ist ein entscheidender Punkt (vgl. Christoffersson, 2023, S. 11-12).

4.3.1.1 Feature: Wahrscheinlichkeiten von Bet365

Als erstes Feature werden die Wahrscheinlichkeiten für die verschieden Spielausgänge berechnet. Diese ergeben sich aus den Quoten von Bet365 in den man den Kehrwert der Quote berechnet (vgl. Christoffersson, 2023, S. 9). Dies ist möglich, da die Quoten angeben, wie viel Gewinn pro eingesetztem Euro im Erfolgsfall ausgezahlt wird.

In dem Python-Code wird die Wahrscheinlichkeit folgendermassen gebildet:

```
Originle_Traningsdaten["Wahrscheinlichkeit_Heimsieg"] = 1
/Originle_Traningsdaten["B365H"]

Originle_Traningsdaten["Wahrscheinlichkeit_Unentschieden"] = 1 /
Originle_Traningsdaten["B365D"]

Originle_Traningsdaten["Wahrscheinlichkeit_Auswärtssieg"] = 1 /
Originle_Traningsdaten["B365A"]
```

Diese Wahrscheinlichkeiten sind verzerrt, da die Wettquoten der Buchmacher nie zu 100% fair sind. Die Quoten beinhalten immer eine Marge, da die Wettanbieter selbst auch was verdienen wollen. Normalerweise wird nur etwa 90% der Einsätze wieder als Gewinn ausgezahlt. Bei den wichtigen und populären Spielen steigt dieser Auszahlungsschlüssel auf 98%, da diese Spiele besser berechnet werden und so attraktiver sind für die grosse Menge zum Tippen. Man sieht auch das der Auszahlungsschlüssel kurz von den Spieltagen steigt, da die Wettanbieter mehr Informationen zur Verfügung haben und auf Basis der bereits platzierten Wetten, so genauere Quoten geben können (vgl. Wette.de, o. J.).

Unter Berücksichtigung dieses Trends, vermute ich das die Wettquoten von Bet365, die in unserer CSV-Datei liegt, einen Auszahlungsschlüssel von etwa 95% haben. Um diesen Bias zu vermeiden, werden die Wahrscheinlichkeiten normiert. Man summiert also die Wahrscheinlichkeiten aller Spielereignisse zusammen und teilt die einzelnen Spielausgänge-Wahrscheinlichkeiten (z.B. Wahrscheinlichkeit für einen Heimsieg) durch die Gesamt-Wahrscheinlichkeit.

In dem Python Code wird die normierte Wahrscheinlichkeit folgendermassen berechnet.

```
Normierte_Gesammtwahrscheinlichkeit =
Originle_Traningsdaten["Wahrscheinlichkeit_Heimsieg"] +
Originle_Traningsdaten["Wahrscheinlichkeit_Unentschieden"] +
Originle_Traningsdaten["Wahrscheinlichkeit_Auswärtssieg"]
```

(summieren der Gesamtwahrscheinlichkeit)

```
Originle_Traningsdaten["Normierte_Wahrscheinlichkeit_Heimsieg"] =
Originle_Traningsdaten["Wahrscheinlichkeit_Heimsieg"] /
Normierte_Gesammtwahrscheinlichkeit
```

(berechnen normierter Wert Heimsieg)

```
Originle_Traningsdaten["Normierte_Wahrscheinlichkeit_Unentschieden"] =
Originle_Traningsdaten["Wahrscheinlichkeit_Unentschieden"] /
Normierte_Gesammtwahrscheinlichkeit
```

(berechnen normierter Wert Unentschieden)

Originle_Traningsdaten["Normierte_Wahrscheinlichkeit_Auswärtssieg"] = Originle_Traningsdaten["Wahrscheinlichkeit_Auswärtssieg"] / Normierte_Gesammtwahrscheinlichkeit

(berechnen normierter Wert Auswärtssieg)

Diese drei normierten Wahrscheinlichkeiten bilden die ersten Features für das Modell.

4.3.1.2 Kalender-Features

Das Modell sollte sich bei der Entscheidung der Spielausgänge nicht allein auf die Wahrscheinlichkeiten der Buchmacher stützen. Da ein Modell, das ausschliesslich auf diesem Feature basiert, nur die Einschätzung der Buchmacher reproduzieren würde, anstatt eigenständig Muster in den Daten zu erkennen. Ziel ist es daher, weitere unabhängige Features einzubeziehen, die zusätzliche Information über den Spielausgang liefern.

Die Buchmacher berücksichtigen zwar in ihren Vorhersagen den Zeitlichen Aspekt, dennoch erlaubt ein explizites Kalender-Feature dem Machine-Learning-Model, die Bedeutung der Saisonphase eigenständig zu erlernen.

Aus der CSV-Datei: **AIFootballPredictions\data\raw\D1_merged.csv,** kann man den Wochentag und den Monat extrahieren. Dazu muss man zuerst die Spalte «date» der CSV-Datei, in ein passendes Datumsformat umwandeln. Gleichzeitig muss sichergestellt werden, dass das Datum in folgendem Format (TT/MM/JJJJ) interpretiert wird und das leere oder nicht gültige Werte mit «Not a Time» aufgefüllt respektive ersetzt werden.

Dies können wir in Python folgendermassen erreichen:

Originle_Traningsdaten["Datum"] = pd.to_datetime(Originle_Traningsdaten["Date"], dayfirst=True, errors="coerce")

(Datum wird aus der CSV-Datei geladen und in folgendem Format (TT/MM/JJJJ) gespeichert)

Anschliessend wird aus dem Datum der Wochentag und der Monat extrahiert, dazu bietet die Panda-Bibliothek direkte Befehle wie «.dt.weekday» und .dt.month.

Der gesamte Befehl in Python sieht folgendermassen aus:

Originle_Traningsdaten["Wochentag"] = Originle_Traningsdaten["Datum"].dt.weekday

(Wochentag wir als eine Zahl von 0-6 gespeichert)

Originle_Traningsdaten["Monat"] = Originle_Traningsdaten["Datum"].dt.month

(Monat wird als eine Zahl von 1-12 gespeichert)

4.3.1.2.1 Kalender-Feature spät_Saison

In den letzten Monaten der Saison steigt die sportliche und körperliche Belastung enorm. Mannschaften kämpfen um die Meisterschaft, die Europäischen Plätze oder gegen den Abstieg. Top Teams spielen nicht ausschliesslich deswegen, gegen Ende der Saison so stark auf, sondern auch die Breite des Kaders spielt dafür eine erhebliche Rolle. Kleine Mannschaften haben meist auch dünner besetzte Kader und können deswegen nur schlecht innerhalb der Saison rotieren, dadurch sind wichtige Spieler ende Saison erschöpft und können nicht mehr Topleistungen bringen. Zudem steigt ende Saison auch die

Verletzungsgefahr, top Temas können, dank ihres grossen und breiten Kaders, dies meist besser wegstecken als kleine Mannschaften (vgl. Draper et al., 2021; Dambroz et al., 2022).

Aus diesem Grund wäre es Interessant ein Binäres Feature zu erstellen, das berücksichtigt, ob ein Spiel spät in der Saison stattfindet. Intuitiv könnte man jetzt meinen alle Spiele, die nach dem Monat März waren, sind spät in der Saison. Dies ist aber nicht ganz korrekt, da die Saison im August/September wieder beginnt, diese Monate sind zwar nach dem März, aber sind zu Beginn der Saison. Deswegen speichert das Binäre-Feature, spät_Saison, nur für die Monate März, April und Mai eine 1 ansonsten eine 0.

Mit folgendem Befehl wird dieses Feature in Python erstellt.:

```
Originle_Traningsdaten["spät_Saison"] =
Originle_Traningsdaten["Monat"].apply(lambda m: 1 if m in [3,4,5] else 0)
```

Nun sind im Modell insgesamt neun verschiedene Features vorhanden. Diese werden im Array finale_features gespeichert, mit folgendem Befehl wird das in Python erreicht:

```
finale_features = [

    "Normierte_Wahrscheinlichkeit_Heimsieg",
    "Normierte_Wahrscheinlichkeit_Unentschieden",
    "Normierte_Wahrscheinlichkeit_Auswärtssieg",

    "Buchmacher_favorisiert_Heimsieg", "Buchmacher_favorisiert_Unentschieden",
    "Buchmacher_favorisiert_Auswärtssieg

    "Wochentag", "Monat", "spät_Saison "]
```

4.3.1.3 Binäres-Feature: Buchmacher favorisiert Spielergebnis

Ein weiteres Feature, das in das Modell integriert werden kann, ist ein Dummy-Feature, das abbildet, ob die Buchmacher den Heimsieg, ein Unentschieden oder den Auswärtssieg favorisieren. Diese binären Features nehmen den Wert 1 an, wenn die Quote für ein Spielausgang (z.B. Heimsieg) die niedrigste, unter den drei möglichen Ergebnissen (Heimsieg, Unentschieden, Auswärtssieg) aufweist. Andernfalls nimmt es den Wert 0 an.

Als erster Schritt muss man in Python die tiefste Quote der drei Spielausgänge, bestimmen, dies erreicht man analog zum ersten Model, in dem man folgende Zeile ausführt.

```
Originle_Traningsdaten["Favorit_Buchmacher"] = Originle_Traningsdaten[["B365H",
"B365D", "B365A"]].idxmin(axis=1).str.extract(r"B365(.)")[0]
```

(Bestimmung welche Quote die tiefste ist und somit die höchste Wahrscheinlichkeit hat)

```
Originle_Traningsdaten["Buchmacher_favorisiert_Heimsieg"] =
(Originle_Traningsdaten["Favorit_Buchmacher"] == "H").astype(int)
```

(Binäre Features, das zeigt, ob der Buchmacher den Heimsieg favorisiert)

```
Originle_Traningsdaten["Buchmacher_favorisiert_Unentschieden"] =
(Originle_Traningsdaten["Favorit_Buchmacher"] == "D").astype(int)
```

(Binäre Features, das zeigt, ob der Buchmacher das Unentschieden favorisiert)

```
Originle_Traningsdaten["Buchmacher_favorisiert_Auswärtssieg"] =
(Originle_Traningsdaten["Favorit_Buchmacher"] == "A").astype(int)
```

(Binäre Features, das zeigt, ob der Buchmacher den Auswärtssieg favorisiert)

Die Motivation für dieses drei Feature liegt darin, dass Buchmacher bei ihrer Quotensetzung zahlreiche Faktoren (z. B. Form der Mannschaften, Verletzungen, Heimvorteil) berücksichtigen, die in den reinen Wahrscheinlichkeiten und der CSV-Datei nicht unmittelbar sichtbar sind. Durch das Dummy-Feature wird die Einschätzung der Buchmacher in vereinfachter Form (favorisieren sie den Heimsieg oder nicht) ins Modell eingebracht. Hierdurch kann das Modell auf diese Informationen zugreifen, ohne sich ausschliesslich von den reinen Quoten der Buchmacher leiten zu lassen.

4.3.1.4 Feature Reduktion

Um zu prüfen, ob einige der ausgewählten Features redundant sind, ist es sinnvoll, ihre linearen Abhängigkeiten zu analysieren. Dafür wird eine Korrelationsmatrix berechnet, die für jedes Feature zeigt, wie stark es mit jedem anderen Feature zusammenhängt.

Eine zu hohe Korrelation (z. B. grösser als 0.8 oder kleiner als –0.8) weist darauf hin, dass zwei Features sehr ähnliche Informationen enthalten. In diesem Fall ist es Sinnvoll, eines der beiden Features zu entfernen, um das Modell schlanker und robuster zu machen (vgl. James et al., 2021, Kapitel 3.3.3)

Für die Berechnung der Korrelationsmatrix muss zuerst sichergestellt werden, dass nur Datensätze verwendet werden, die vollständig sind. Dazu werden alle Zeilen der CSV-Datei entfernt, die entweder nicht alle finalen Features oder nicht das tatsächliche Resultat beinhalten.

Dazu wird folgende Zeile in Python ausgeführt:

Vollständige_Origanle_Traningsdaten = Origanle_Traningsdaten[finale_features + ["FTR"]].dropna()

Anschliessend werden die Feature-Arrays für das Modell gebildet, in dem in Feature_Info nur die Informationen der finalen Features der Trainingsdaten gespeichert werden.

Feature_Info = Vollständige_Origanle_Traningsdaten[finale_features]

Anlog werden die Ziel-Arrays für das Modell gebildet, in dem in Vollständige_Tatsächliches_Resultat nur die Informationen der tatsächlichen Resultate der Trainingsdaten gespeichert werden.

Vollständige_Tatsächliches_Resultat = Vollständige_Origanle_Traningsdaten["FTR"]

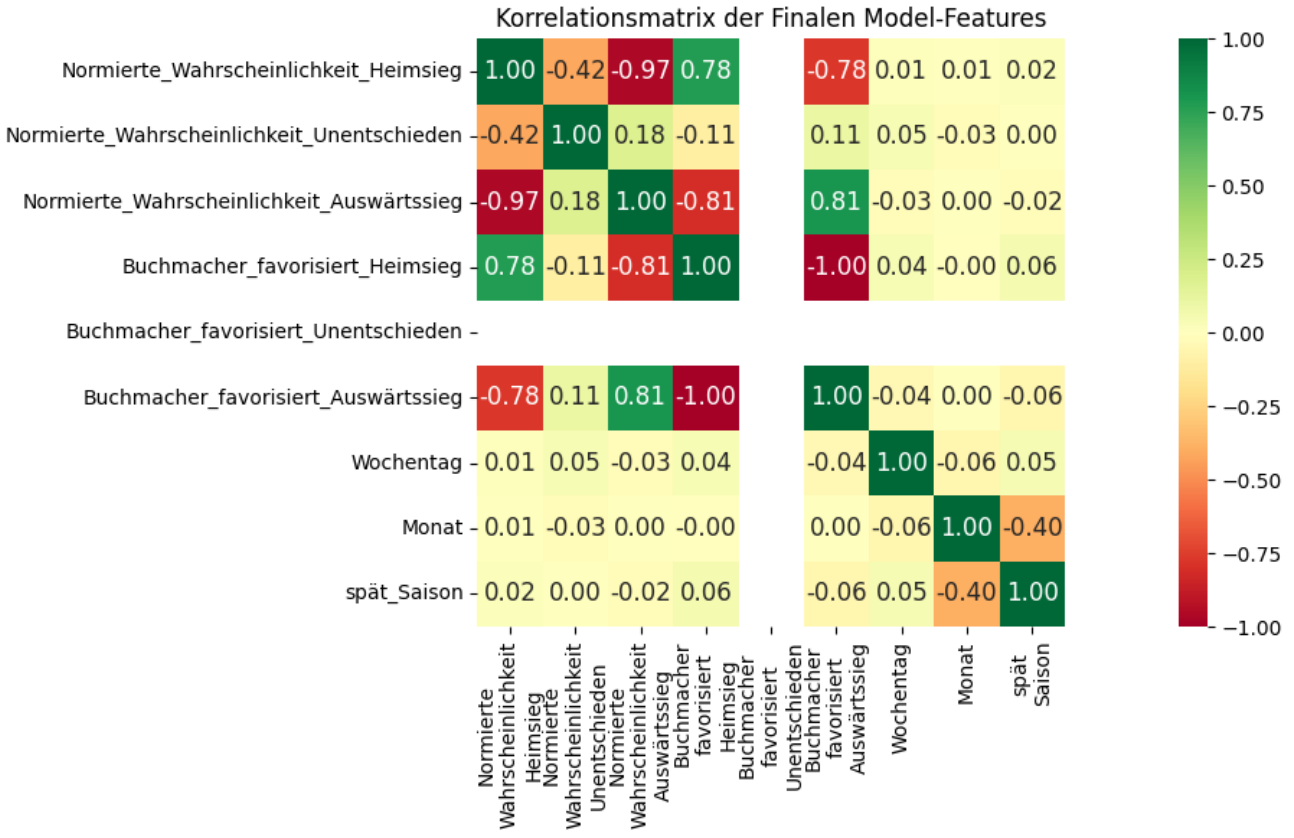
Nun wird die Korrelationsmatrix berechnet, dazu stellt die Pandas-Bibliothek folgenden Befehl bereit, «corr()». Anschliessend wird die Korrelationsmatrix visualisiert.

Korrelationsmatrix = Feature_Info.corr()

(Berechnung der Korrelationsmatrix)

```
plt.figure(figsize=(16, 6))
sns.heatmap(
        Korrelationsmatrix,
        annot=True,
        fmt=".2f",
        cmap="RdYlGn",
        square=True,
        cbar=True,
        annot_kws={"size": 12},
        xticklabels=[label.replace('_', '\n') for label in Korrelationsmatrix.columns],

)
plt.title
plt.tight_layout()
plt.savefig("Korrelationsmatrix der Finalen Model-Featurespng.png")
plt.show()
```

(Visualisierung der berechneten Korrelationsmatrix)



Die Korrelationsmatrix zeigt einige interessante Informationen. Auffällig ist zunächst, dass das Feature Buchmacher_favorisiert_Unentschieden, leer ist. Dies bedeutet, dass die Buchmacher nie ein Unentschieden favorisieren, was plausibel ist und bereits in Kapitel 2.5 beschrieben wurde.

Darüber hinaus zeigt sich, dass Buchmacher_favorisiert_Heimsieg und Buchmacher_favorisiert_Auswärtssieg perfekt negativ korrelieren, da, wenn der Buchmacher einen Heimsieg favorisiert, er nicht gleichzeitig einen Auswärtssieg favorisieren kann, und umgekehrt.

Stark korreliert sind auch die normierten Wahrscheinlichkeiten für einen Heimsieg und einen Auswärtssieg. Dies macht auch Sinn, da das Modell die Wahrscheinlichkeit eines Auswärtssiegs einfach als Gegenwahrscheinlichkeit zur Wahrscheinlichkeit eines Heimsiegs und eines Unentschiedens berechnen kann.

Auf Basis dieser Erkenntnisse, kann eine erste Feature Reduktion vorgenommen werden. Als erstes wird sicher Buchmacher_favorisiert_Unentschieden entfernt, da dieses Feature keine zusätzlichen Informationen gibt.

Zusätzlich ist auch klar, dass wir Buchmacher_favorisiert_Heimsieg oder Buchmacher_favorisiert_Auswärtssieg entfernen können, ich entscheide mich für die Entfernung von Buchmacher_favorisiert_Auswärtssieg, da dies weniger oft der Fall ist.

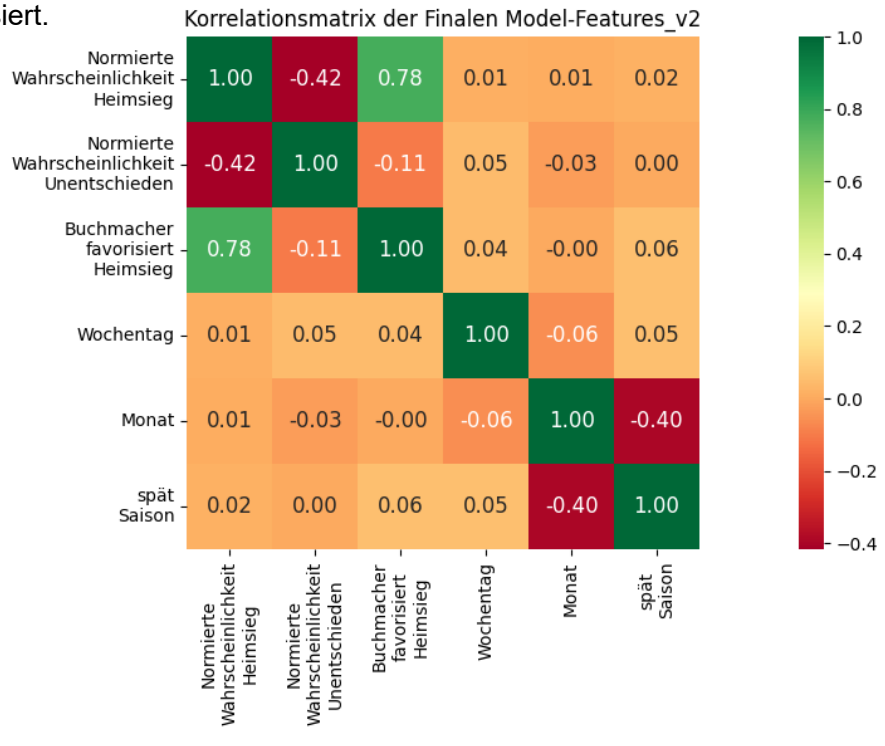
Zum Schluss wird noch die normierte Wahrscheinlichkeit eines Auswärtssieges entfernt, alternativ hätte man sich hier auch für die normierte Wahrscheinlichkeit eines Heimsiegs entscheiden können.

Nun wird in Python ein neues Array erstellt, dass nur die finalen Features, die wird behalten wollen beinhaltet.

reduzierte_finale_features = [

"Normierte_Wahrscheinlichkeit_Heimsieg",
"Normierte_Wahrscheinlichkeit_Unentschieden",
"Buchmacher_favorisiert_Heimsieg",
"Wochentag", "Monat", "spät_Saison"]

Anschliessend wird wie oben beschrieben, nochmals die Korrelationsmatrix berechnet und visualisiert.



In der Korrelationsmatrix wird sichtbar, dass es immer noch stark Korrelierende Features gibt. Insbesondere die Normierte_Wahrscheinlichkeit_Heimsieg und Buchmacher_favorisiert_Heimsieg, korrelieren stark, was aber auch logisch ist, da es sich beides auf Basis der Quoten der Buchmacher für einen Heimsieg bildet. Um sicherzustellen, dass das Modell nicht ausschliesslich auf den Wahrscheinlichkeiten der Buchmacher basiert, werden beide Features im Modell beibehalten.

4.3.2 Training des Modells

Um das Modell zu trainieren und anschliessend zu testen, bevor die Validierungsdaten der Saison 2024/2025 verwendet werden, werden die Daten in Trainings- und Testdatensätze aufgeteilt. Eine gängige Aufteilung besteht darin, 80 % der Daten für das Training zu verwenden und die verbleibenden 20 % für das Testen. Die Testdaten dienen dazu zu überprüfen, wie gut sich das Modell auf unbekannte Daten generalisieren lässt (vgl. James et al., 2021, Kapitel 5.1). Zum Schluss wird sichergestellt, dass die Verteilung der Spielausgänge (Heimsieg, Unentschieden und Auswärtssieg) in den Trainings- und Testdaten ungefähr im gleichen Verhältnis vorliegt.

Diese Aufteilung erfolgt mit folgendem Python-Befehl:

```
X_traings_daten, X_test_daten, y_traings_daten, y_test_daten = train_test_split(
    Verwendtete_Features, Vollständige_Tatsächliches_Resultat, test_size=0.2,
stratify=Vollständige_Tatsächliches_Resultat, random_state=42)
```

Bevor das Modell trainiert wird, müssen die Features normalisiert werden. Dabei werden die Werte so transformiert, dass alle Features einen Mittelwert von 0 und eine Standardabweichung von 1 aufweisen. Hierfür wird der StandardScaler aus der Bibliothek scikit-learn verwendet.

Durch diese Transformation wird sichergestellt, dass alle Features auf derselben Skala liegen und kein Merkmal aufgrund seiner Grössenordnung einen überproportionalen Einfluss auf die Resultate des Modells hat.

Die Normalisierung wird sowohl für die Trainings- als auch für die Testdaten angewendet. Dabei erfolgt die Anpassung auf Grundlage der Trainingsdaten und wird anschliessend auf die Testdaten übertragen.

Die Umsetzung in Python erfolgte wie folgt:

```
scaler = StandardScaler()

X_traings_daten_scaliert = pd.DataFrame(scaler.fit_transform(X_traings_daten),
columns=X_traings_daten.columns)

X_test_daten_scaliert = pd.DataFrame(scaler.transform(X_test_daten),
columns=X_test_daten.columns)
```

Nun folgt das eigentliche Modelltraining mithilfe eines multinomialen logistischen Regressionsmodells.

In einem 1. Ansatz wurden folgende Parameter gewählt:

- Die maximale Anzahl an Iterationen (max_iter) wurde auf 1000 gesetzt, um Konvergenz sicherzustellen.
- Die Einstellung multi_class = 'multinomial' ermöglicht die Anwendung auf mehr als zwei Klassen.
- Als Optimierungsalgorithmus wurde der lbfgs-Solver verwendet.
- Der Regularisierungsparameter (C) wurde auf 0.01 gesetzt, um Überanpassung zu vermeiden.
- Mit class_weight = "balanced" wurde eine Gewichtung der Klassen vorgenommen, um die unausgeglichene Klassenverteilung auszugleichen.

Anschliessend wird das Modell mit den normalisierten Trainingsdaten trainiert. In Python sieht die Umsetzung wie folgt aus:

log_reg = LogisticRegression(max_iter=1000, multi_class='multinomial', solver='lbfgs', C=0.01, class_weight="balanced")

log_reg.fit(X_traings_daten_scaliert, y_traings_daten)

vorhergesagte_Resultate = log_reg.predict(X_test_daten_scaliert)

4.3.3 Evaluation des Modells

Anschliessend folgt die Evaluation. Analog zum vorherigen Modell wurden erneut die Metriken Accuracy, Precision, Recall, F1-Score und Support berechnet. Die Erläuterung der einzelnen Kennzahlen erfolgte bereits in den Kapiteln 3.1 + 4.2.1.1, sodass hier lediglich die Ergebnisse dargestellt werden.

4.3.3.1 Classification Report

Die Genauigkeit für die Testdaten, dieses Modells liegt bei ungefähr **46 %**. Zwar wäre mit einem Parameter-Feintuning eine höhere Genauigkeit erreichbar (siehe Kapitel 5.2.3), dennoch zeigt sich, dass das vorherige Modell eine bessere Performance aufwies.

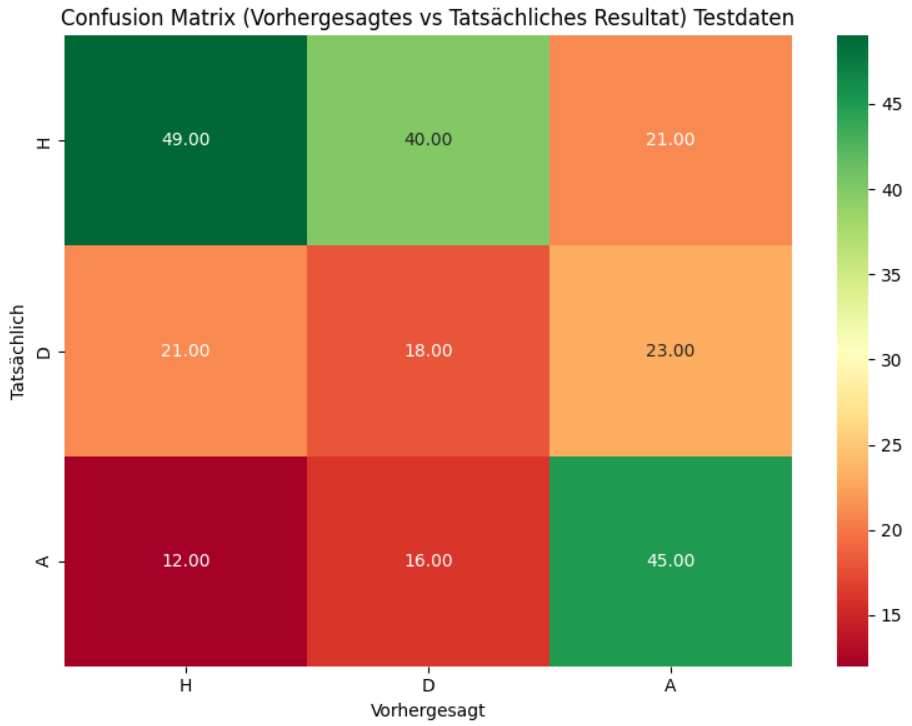
Classification Report (Saison Testdaten)

Klasse	precision	recall	f1-score	support
A	0.51	0.62	0.56	73.0
D	0.24	0.29	0.26	62.0
H	0.6	0.45	0.51	110.0
accuracy	0.46	0.46	0.46	0.46

accuracy: 0.46

Ergänzend wurde, analog zum ersten Modell, eine Konfusionsmatrix für die Testdaten erstellt, die die Verteilung der korrekten und inkorrekten Vorhersagen je Klasse im Detail veranschaulicht.

4.3.3.2 Konfusionsmatrix



4.3.3.3 Feature-Wichtigkeit

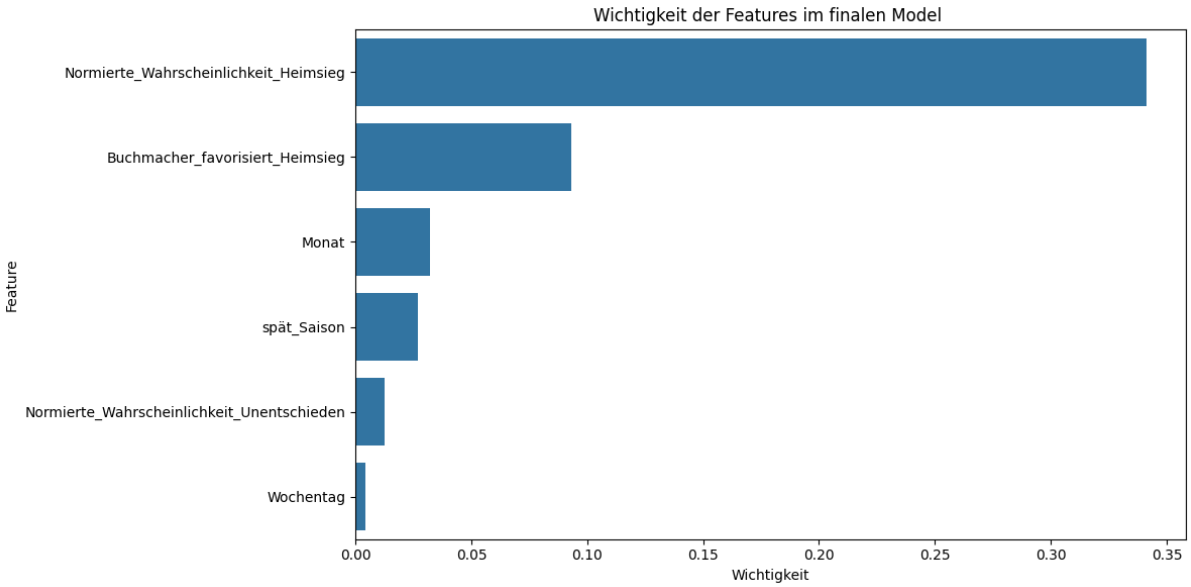
Neben der Modellgüte wird auch untersucht, welches Features den grössten Einfluss auf das Resultat des Modells hat.

Dazu wurden die Koeffizienten der Features von dem logistischen Regressionsmodells extrahiert. Damit man sowohl positive als auch negative Zusammenhänge vergleichbar darstellen kann, wurde der absolute Betrag der Koeffizienten verwendet.

In Python setzt man das folgendermassen um:

```
Koeffizienten = log_reg.coef_[0]

Feature_Wichtigkeit = pd.DataFrame({

   "Feature": X_traings_daten_scaliert.columns,

   "Wichtigkeit": np.abs(Koeffizienten)

}).sort_values(by="Wichtigkeit", ascending=False)
```

Die Feature-Wichtigkeiten wird schlussendlich in einem Balkendiagramm visualisiert, die Merkmale werden absteigend nach ihrer Wichtigkeit respektive Einfluss auf das Modell sortiert.



4.3.4. Evaluation des Modells hinsichtlich der Validierungsdaten

Im nächsten Schritt wurde das bereits trainierte Modell auf die Validierungsdaten der Saison 2024/25 angewendet. Die Methodik zur Berechnung der Wahrscheinlichkeiten, Auswahl der finalen Features sowie Erstellung der Metriken wurde bereits in den vorgegangenen Kapiteln erläutert, weshalb der Fokus hier auf der Darstellung der Ergebnisse liegt.

4.3.4.1 Classification Report

Die Accuracy des Modells ohne Parameter-Tuning beträgt **25 %**. Damit konnte das Modell nicht einmal jedes dritte Spiel korrekt vorhersagen, was unter dem Zufallserwartungswert von 33 % liegt. Auffällig ist, dass das Modell momentan ausschliesslich **Unentschieden** vorhersagt.

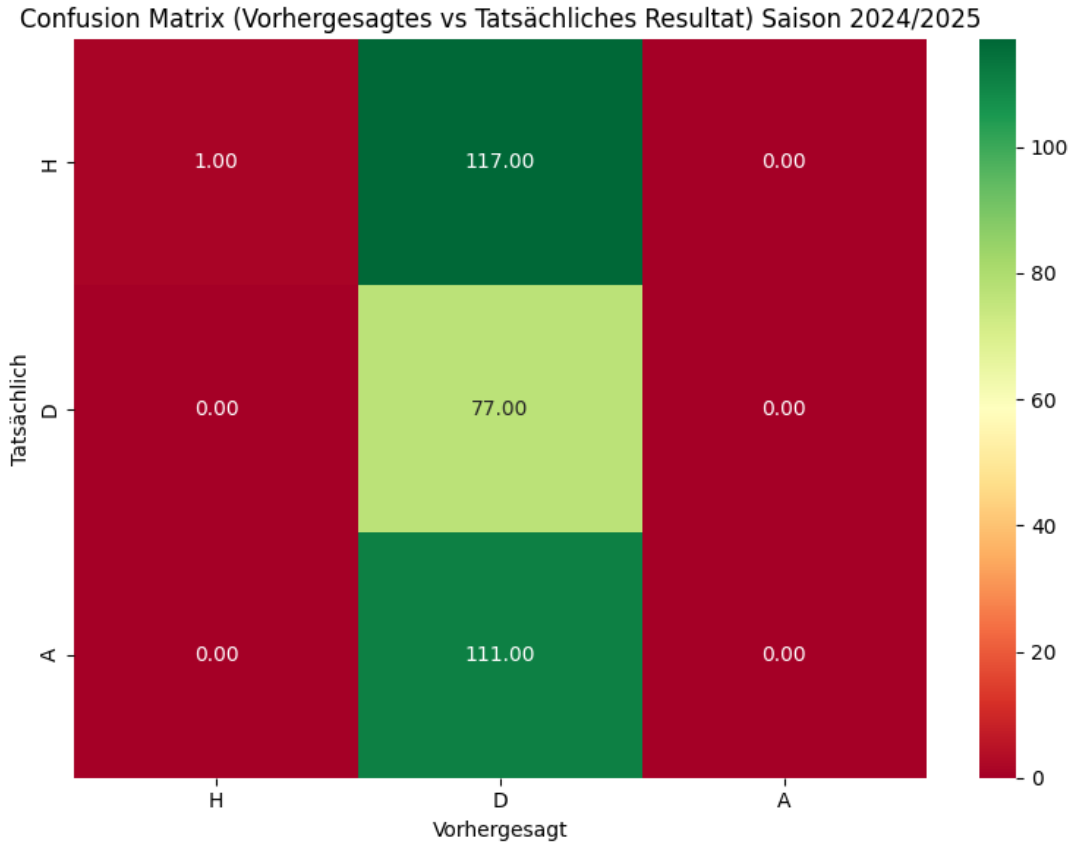
Classification Report (Saison 2024/2025)

Klasse	precision	recall	f1-score	support
A	0.0	0.0	0.0	111.0
D	0.25	1.0	0.4	77.0
H	1.0	0.01	0.02	118.0

Accuracy: 0.25

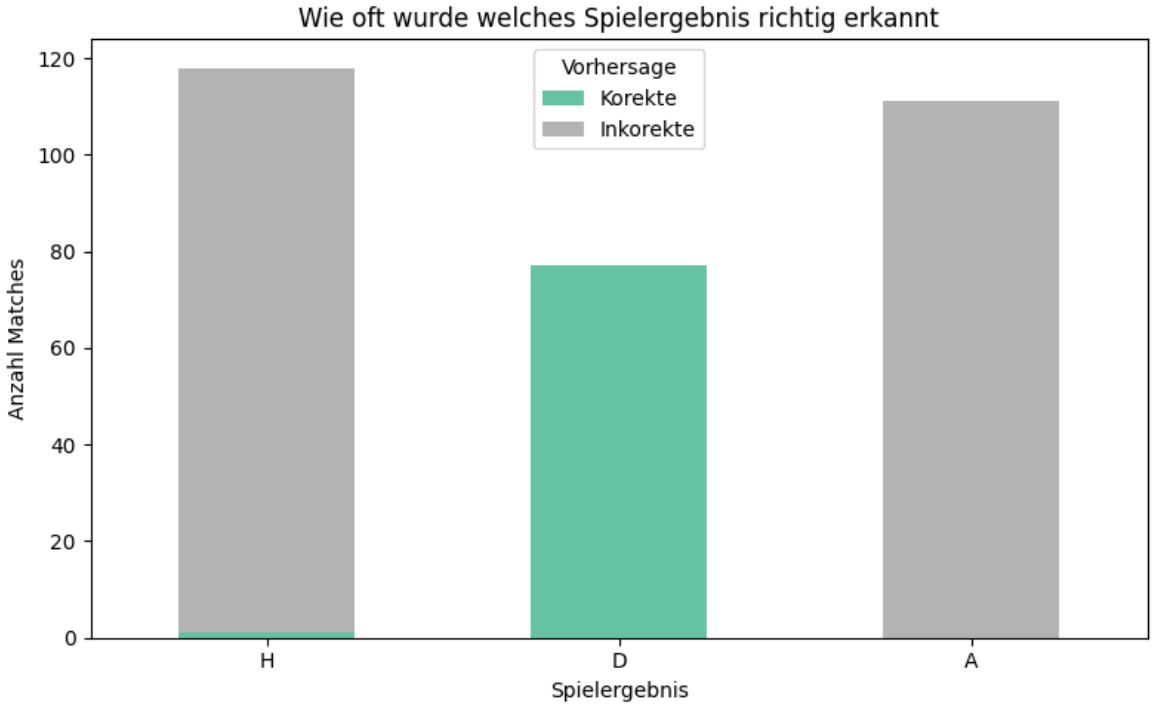
4.3.4.2 Konfusionsmatrix

Die Konfusionsmatrix verdeutlicht diese Problematik. Das Modell sagt nahezu alle Spiele als Unentschieden voraus, nur ein Spiel wird als Heimsieg klassifiziert. Dadurch werden Heimsiege und Auswärtssiege systematisch nicht korrekt vorhergesagt.



4.3.4.3 Balkendiagramm der Vorhersagegenauigkeit

Das Balkendiagramm visualisiert die Anzahl korrekt und inkorrekt vorhergesagter Ergebnisse je Klasse und bestätigt, dass ausschliesslich Unentschieden korrekt vorhergesagt wurden, während Heimsiege und Auswärtssiege vollständig fehlerhaft klassifiziert wurden.



Diese Ergebnisse deuten darauf hin, dass das Modell aktuell eine starke Verzerrung aufweist und lediglich das Unentschieden vorhersagt. Eine detaillierte Interpretation dieser Beobachtung sowie mögliche Anpassungen werden im Kapitel 5.2 und 5.3 behandelt.

4.4 Modell 2: Gradient-Boosting

Um die Vorhersagegenauigkeit ohne zusätzliches Feature-Tuning zu verbessern und alternative Modellansätze zu evaluieren, wurde zusätzlich ein Gradient-Boosting-Modell trainiert. Dabei wurden die gleichen Trainings- und Validierungsdaten sowie die zuvor ausgewählten Features verwendet. Die nachfolgenden Abschnitte beschreiben die Umsetzung und Ergebnisse dieses Modells.

Bis zum Training der Daten bleibt der Python-Code unverändert. Der einzige Unterschied besteht darin, dass anstelle des Moduls LogisticRegression das Modul GradientBoostingClassifier importiert wird, da das Modell nun auf einem Gradient-Boosting -Ansatz basiert.

Konkret wird im Python-Skript die Zeile:

from sklearn.linear_model import LogisticRegression

durch

from sklearn.ensemble import GradientBoostingClassifier

ersetzt.

Im Gegensatz zur logistischen Regression ist beim Gradient-Boosting-Modell keine Normalisierung der Features notwendig. Entscheidungsbäume und damit auch Gradient-Boosting-Modelle arbeiten mit Schwellenwerten, wodurch sie unempfindlich gegenüber unterschiedlichen Skalen der Eingabedaten sind (vgl. James et al., 2021, Kap. 8.2). Daher können die Trainings- und Testdaten direkt verwendet werden, ohne den StandardScaler einsetzten zu müssen. Dem entsprechend entfällt auch der Import dieser Bibliothek im Code ganz oben.

Das Gradient-Boosting-Modell kombiniert mehrere Entscheidungsbäume sequenziell zu einem Ensemble, wobei jeder neue Baum die Fehler der vorherigen korrigiert, um so genauere Vorhersagen zu erzielen (vgl. James et al., 2021, Kap. 8.2). Ziel ist es, im Vergleich zum logistischen Regressionsansatz bessere Ergebnisse zu erreichen. Die theoretischen Grundlagen sowie ein Vergleich zwischen den gängigen Machine-Learning Modellen wurde bereits in Kapitel 2. ausführlich behandelt.

4.4.1 Parametertuning

Um den Gradient-Boosting-Ansatz in Python umzusetzen, wird zunächst das Modell erstellt

```
Model_GradientBoosting = GradientBoostingClassifier(
  n_estimators=1000,
  learning_rate=0.001,
  max_depth=5,
  random_state=42)
```

- **n_estimators=1000** legt die Anzahl der Entscheidungsbäume im Gradient-Boosting-Modell fest. Mehr Bäume können das Modell genauer machen, da sie mehr Lernschritte und Anpassungen ermöglichen. Allerdings erhöht eine zu grosse Anzahl die Rechenzeit und kann das Training verlangsamen.

- **learning_rate=0.001** bestimmt die Schrittweite beim Anpassen der Bäume, kleinere Werte führen zu stabileren, langsameren Lernprozessen, grössere Werte zu schnellerem Lernen, können jedoch instabil sein.

- **max_depth=5** gibt die maximale Tiefe der Bäume an, zu kleine Werte können die Modellleistung begrenzen, zu tiefe Werte erhöhen das Risiko von Overfitting.

- **random_state=42** sorgt dafür, dass das Modell bei jedem Lauf die gleichen Ergebnisse liefert, damit die Vorhersagen vergleichbar und reproduzierbar sind.

Die oben gewählten Parameter sind das Resultat, eines iterativen Optimierungsprozesses. Durch ständiges Ausprobieren und Anpassen der Parameter wurde die Modellperformance nach und nach verbessert, bis eine zufriedenstellende Genauigkeit erreicht wurde.

4.4.2 Training des Modells

Anschliessend wird das Modell mit den Trainingsdaten trainiert, um so die Zusammenhänge zwischen den Eingabedaten (Features) und den Zielwerten (tatsächliche Resultate) zu erlernen:

```
Model_GradientBoosting.fit(X_traings_daten, y_traings_daten)
```

Zum Schluss wird das trainierte Modell auf die Testdaten angewendet, um Vorhersagen für die jeweiligen Spielergebnisse zu treffen:

```
vorhergesagte_Resultate = Model_GradientBoosting.predict(X_test_daten)
```

Analog zum Model Logistische Regression wird das trainierte Gradient-Boosting-Modell nun auf die Validierungsdaten der Spielzeit 2024/25 angewendet. Dabei werden die gleichen Features, wie beim Model Logistische Regression verwendet. Zudem bleibt der Ablauf zur Berechnung der Metriken Accuracy, Classification Report und Konfusionsmatrix unverändert.

Die einzige Unterscheidung im Code der beiden Modelle, liegt in der Modellbezeichnung innerhalb der Vorhersagezeile:

```
vorhergesagte_Resultate2425 = Model_GradientBoosting.predict(Feature_Info_final_2425)
```

Damit unterscheidet sich der Code nur minimal zum Model Logistische Regression:

```
(log_reg.predict(...))
```

4.4.3 Evaluation des Modells

Die Ergebnisse werden anschliessend wie gewohnt berechnet und dargestellt.

4.4.3.1 Classification Report

Zunächst erfolgt die Auswertung auf den Testdaten. Als Erstes wird die Accuracy berechnet, welche den Anteil der korrekt vorhergesagten Spiele angibt. Anschliessend wird der Classification Report erstellt, der die Kennzahlen Precision, Recall, F1-Score und Support für die Klassen Heimsieg, Unentschieden und Auswärtssieg ausweist.

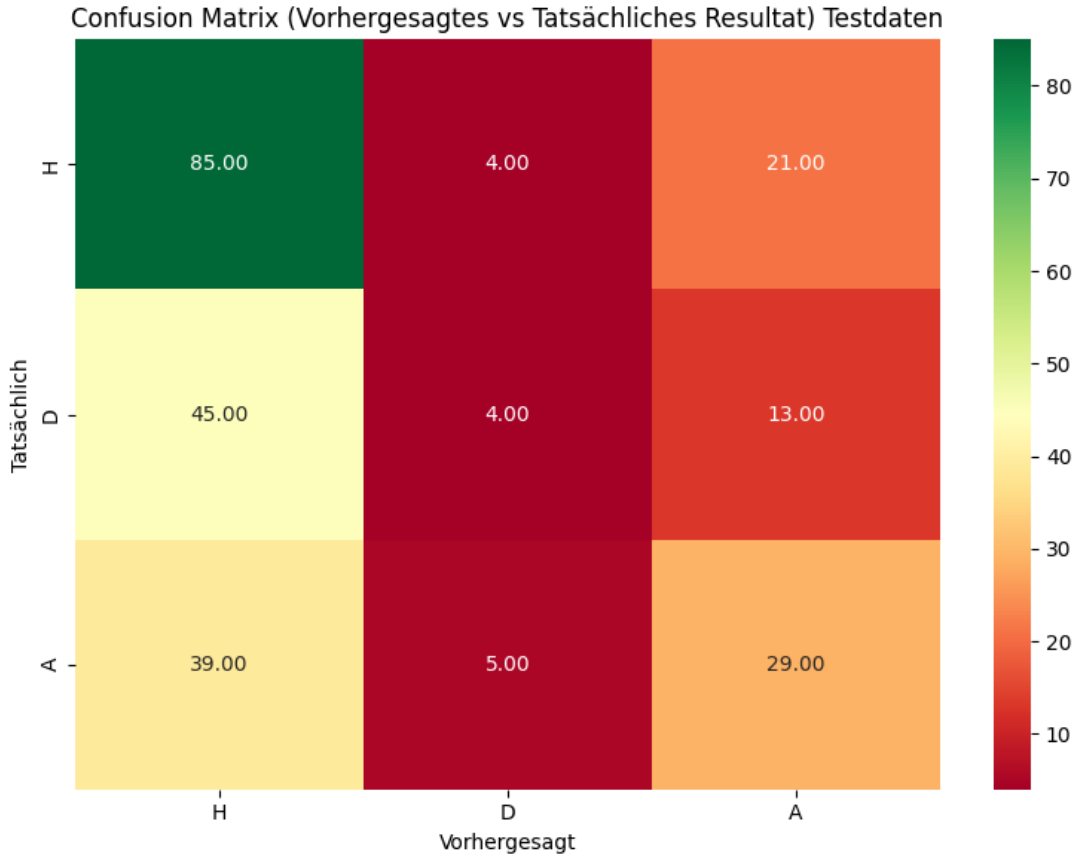
Classification Report (Saison Testdaten)

Klasse	precision	recall	f1-score	support
A	0.46	0.4	0.43	73.0
D	0.31	0.06	0.11	62.0
H	0.5	0.77	0.61	110.0
accuracy	0.48	0.48	0.48	0.48

accuracy: 0.48

4.4.3.2 Konfusionsmatrix

Danach folgt die Konfusionsmatrix, welche die Verteilung der korrekten und inkorrekten Vorhersagen verdeutlicht.



4.4.3.3 Feature-Wichtigkeit

Abschliessend wird die Feature-Wichtigkeit berechnet, um die einflussstärksten Merkmale für die Vorhersagen des Gradient-Boosting-Modells zu identifizieren.

Anders als bei der logistischen Regression, bei der dafür die Koeffizienten berechnet werden, basiert die Wichtigkeit der Features bei diesem Modell auf den «Feature Importances». Während die Koeffizienten der logistischen Regression, die Richtung (positiv/negativ) und die Stärke des Einflusses eines Features angeben, gibt es bei Gradient Boosting keine solchen Koeffizienten. Stattdessen misst das Modell die relative Bedeutung der Features basierend darauf, wie stark sie die Entscheidungen in den Entscheidungsbäumen beeinflussen.

Deshalb wird die folgende Zeile aus dem Model Logistische Regression im Python-Code angepasst:

Koeffizienten = log_reg.coef_[0]

Zu folgender

Importances = Model_GradientBoosting.feature_importances

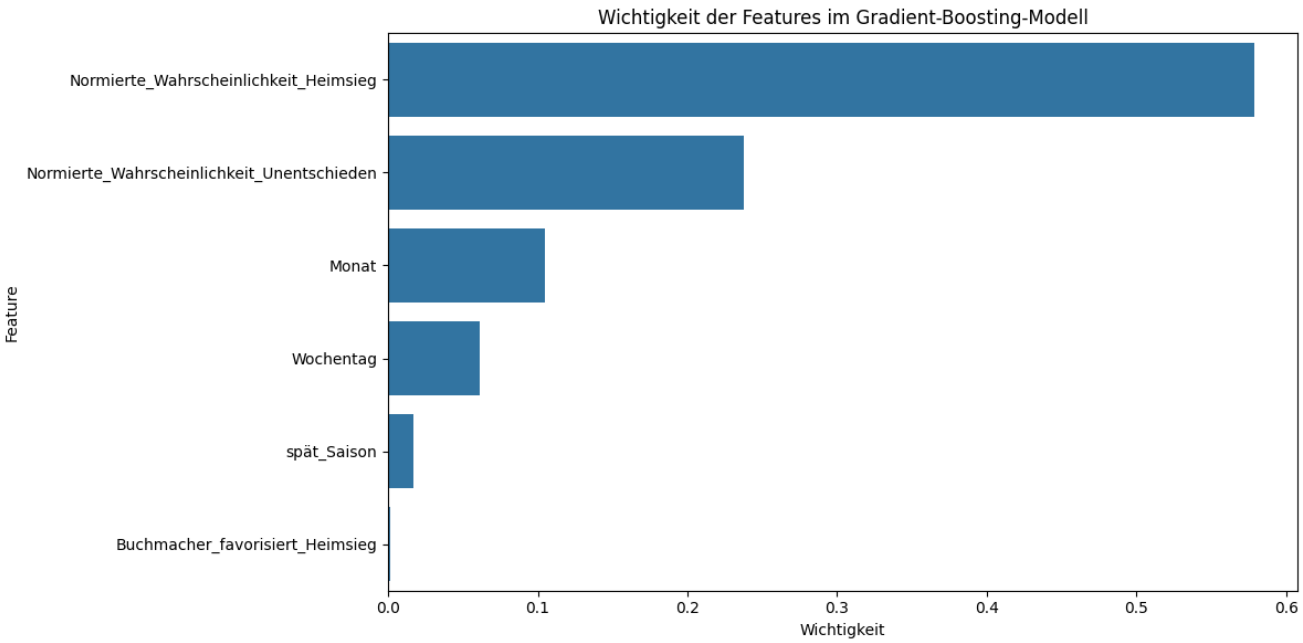
Der restliche Ablauf des Codes bleibt weitgehend unverändert. Ein kleiner Unterschied ist jedoch, dass die logistische Regression mit skalierten Daten arbeitet, während Gradient Boosting nicht normierte Daten benötigt. Daher wird bei der Erstellung des DataFrames der Zusatz «_scaliert» weggelassen:

"Feature": X_traings_daten.columns,

Ein weiterer feiner Unterschied besteht darin, dass beim Gradient-Boosting-Modell der Betrag der Werte nicht genommen werden muss, da die Importances immer positiv sind:

"Wichtigkeit": Importances

Die Werte werden wie gewohnt absteigend nach ihrer Wichtigkeit sortiert, sodass die wichtigsten Merkmale leicht identifiziert werden können. Zudem wird das Balkendiagramm analog zum Model Logistische Regression visualisiert.



4.4.4. Evaluation des Modells hinsichtlich der Validierungsdaten

4.4.4.1 Classification Report

Daraufhin werden die gleichen Schritte, also die Berechnung der Accuracy und die Erstellung des Classification Reports auch für die Validierungsdaten der Saison 2024/25 durchgeführt und visualisiert.

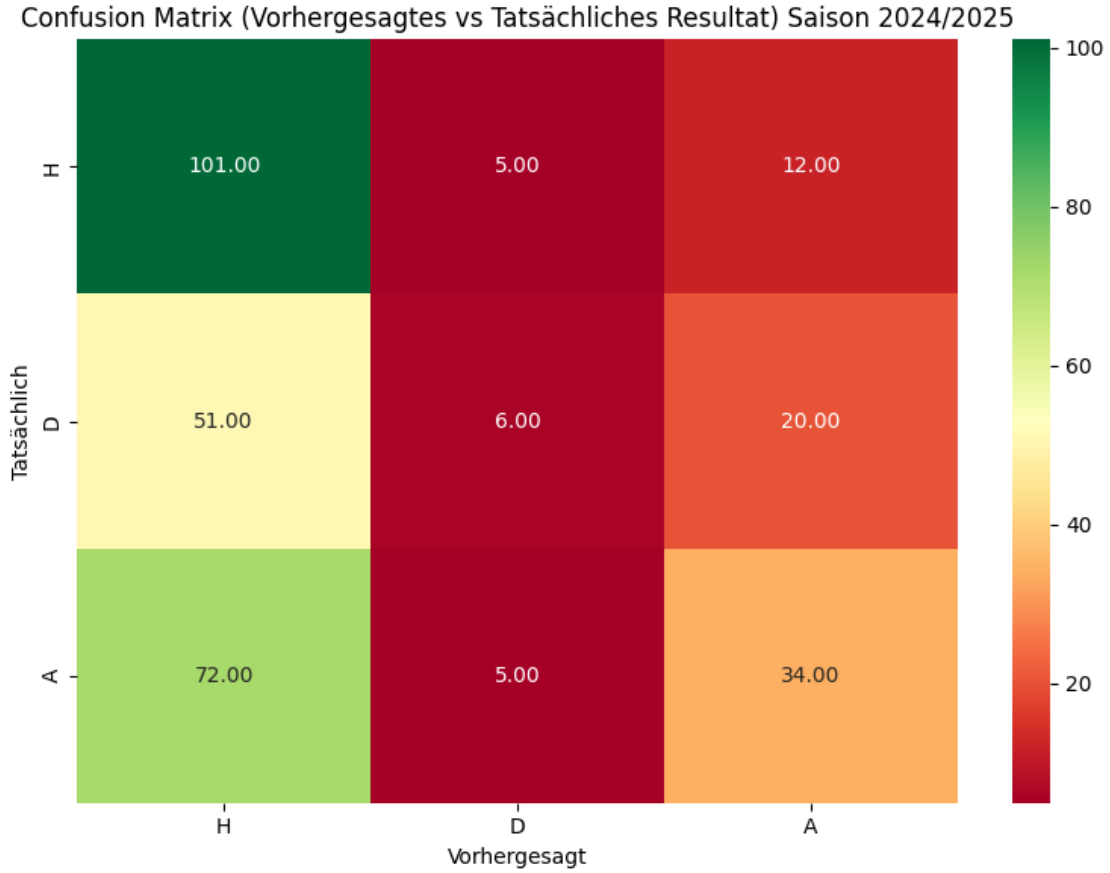
Classification Report (Saison 2024/2025)

Klasse	precision	recall	f1-score	support
A	0.52	0.31	0.38	111.0
D	0.38	0.08	0.13	77.0
H	0.45	0.86	0.59	118.0

Accuracy: 0.46

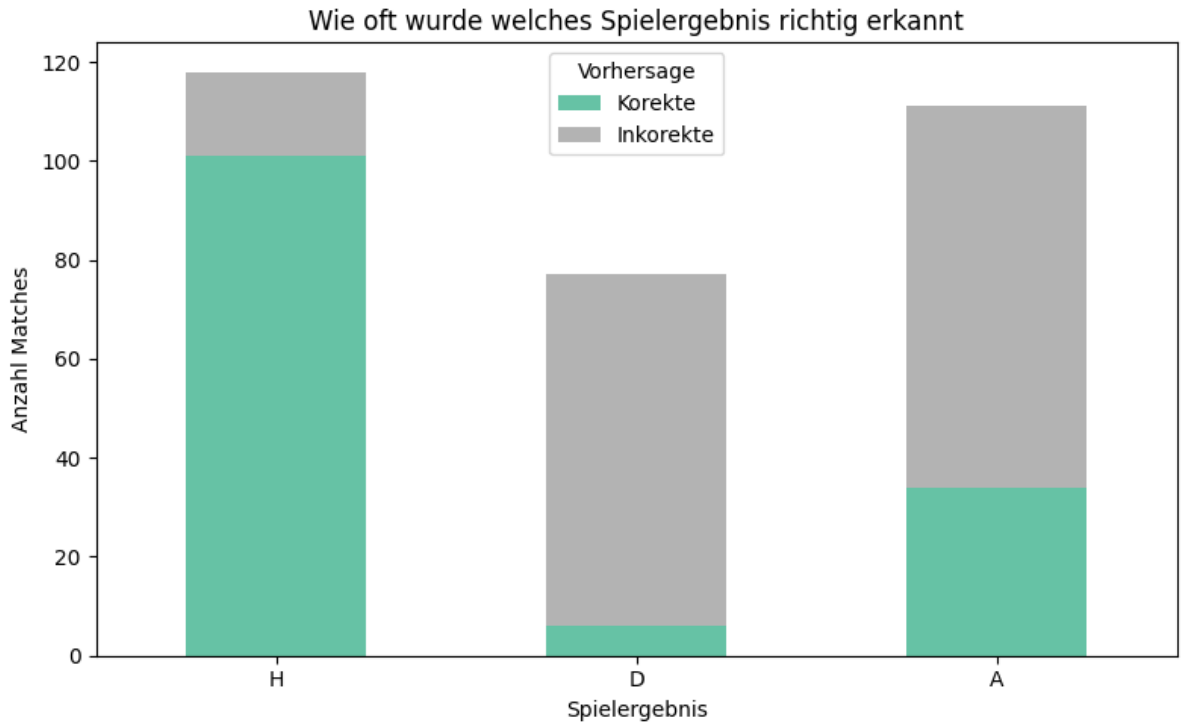
4.4.4.2 Konfusionsmatrix

Zudem wird die Berechnung und Visualisierung der Konfusionsmatrix, auch für die Validierungsdaten der Saison 2024/25 durchgeführt.



4.4.4.3 Balkendiagramm der Vorhersagegenauigkeit

Zum Abschluss wird zusätzlich ein Balkendiagramm erstellt, das die Anzahl korrekt und inkorrekt vorhergesagter Ergebnisse je Klasse zusammenfasst.



4.5 Integration eines neuen Features

Obwohl die bisher betrachteten Modelle unterschiedliche Ansätze verfolgten, beruhen ihre Vorhersagen weiterhin stark auf den Wahrscheinlichkeiten der Buchmacher. Ziel dieser Arbeit war es, zusätzliche Kriterien zu evaluieren, die von den Buchmachern bisher nicht berücksichtigt werden. Wie in Kapitel 3.2 erläutert, wurde hierfür ein EM-Score eingeführt.

Jeder Bundesliga-Verein erhält einen individuellen EM-Score, dieser sollte die summierte Form der Spieler einer Mannschaft repräsentieren. die Ermittlung wird ebenfalls in Kapitel 3.2 erläutert. Dieser Wert fliesst nicht in die Trainingsdaten ein, da diese auf Spielzeiten vor der EM 2024 basieren. Würde der EM-Score hier berücksichtigt werden, käme es zu einer klassischen Daten-Leakage.

Da die Analyse auf der vorgegebenen CSV-Datei basiert, war es nur möglich, ein bestehendes Feature anzupassen. Ich habe mich für die Normierte Wahrscheinlichkeit eines Heimsiegs entschieden, da dieses Feature den grössten Einfluss auf das trainierte Modell besitzt.

4.5.1 EM-Score

Dazu wird der individuelle EM-Score der Auswärtsmannschaft von dem individuelle EM-Score der Heimmannschaft subtrahiert.

Angenommen Bayern Munich spielt zuhause gegen Freiburg, dann würde die Rechnung wie folgt aussehen:

("Bayern Munich") 0.7 – ("Freiburg") -0.2 = 0.9

Dieser Wert wird mit 0.1 multipliziert, anschliessend wird 1 addiert. Die Summe wird dann mit der Normierten Wahrscheinlichkeit des Heimsiegs multipliziert:

In unserem Beispiel würde daraus folgen:

normierte Wahrscheinlichkeit Heimsieg*(1+0.9*0.1)

Nun sollte man durch die korrigierte Wahrscheinlichkeit des Heimsiegs eigentlich, die Wahrscheinlichkeiten aller Spielausgänge wieder normieren. Ich gehe jedoch davon aus, dass die Wahrscheinlichkeit für ein Unentschieden unverändert bleibt, dadurch passt das Modell die Wahrscheinlichkeit für einen Auswärtssieg schon automatisch an.

Konkret bedeutet, dass nur folgende Änderungen, bei der Anwendung auf die Validierungsdaten, im Python-Code gemacht werden müssen.

```
em_score = {
  "Bayern Munich": 0.7, "Leverkusen": 0.7, "Ein Frankfurt": 0.1,
  "Dortmund": 0.8, "Freiburg": -0.2, "Mainz": 0.1, "RB Leipzig": 0.4,
  "Werder Bremen": -0.1, "Stuttgart": 0.5, "M'gladbach": 0.1,
  "Wolfsburg": -0.5, "Augsburg": 0.1, "Union Berlin": -0.3,
  "St Pauli": 0.0, "Hoffenheim": 0.1, "Heidenheim": 0.0,
  "Holstein Kiel": 0.0, "Bochum": 0.0
}
```

(Zuweisung des EM-Scores für jedes Bundesliga Team)

```
Saison_2425["Heimteam"] = Saison_2425["HomeTeam"].map(em_score)
```

(Extraktion des Heimteams, aus den Validierungsdaten)

```
Saison_2425["Auswärtsteam"] = Saison_2425["AwayTeam"].map(em_score)
```

(Extraktion des Auswärtsteams, aus den Validierungsdaten)

```
Saison_2425["EM_Differenz"] = Saison_2425["Heimteam"] - Saison_2425["Auswärtsteam"]
```

(Berechnung der Differenz der beiden EM-Scores)

```
Saison_2425["Normierte_Wahrscheinlichkeit_Heimsieg"] *= (1 + 0.1 *
Saison_2425["EM_Differenz"])
```

(Anpassung des Feature Normierte_Wahrscheinlichkeit_Heimsieg)

Abgesehen von dieser Anpassung bleibt der Rest des Gradient-Boosting-Modells unverändert, heisst auch die gleichen Auswertungen (Accuracy, Report, Kofusionsmatix, Wichtigkeit der Features und das Balkendiagramm) werden berechnet und visualisiert.

4.5.2 Evaluation des Modells

Um Wiederholungen zu vermeiden, werden die entsprechenden Ergebnisse kommentarlos dargestellt.

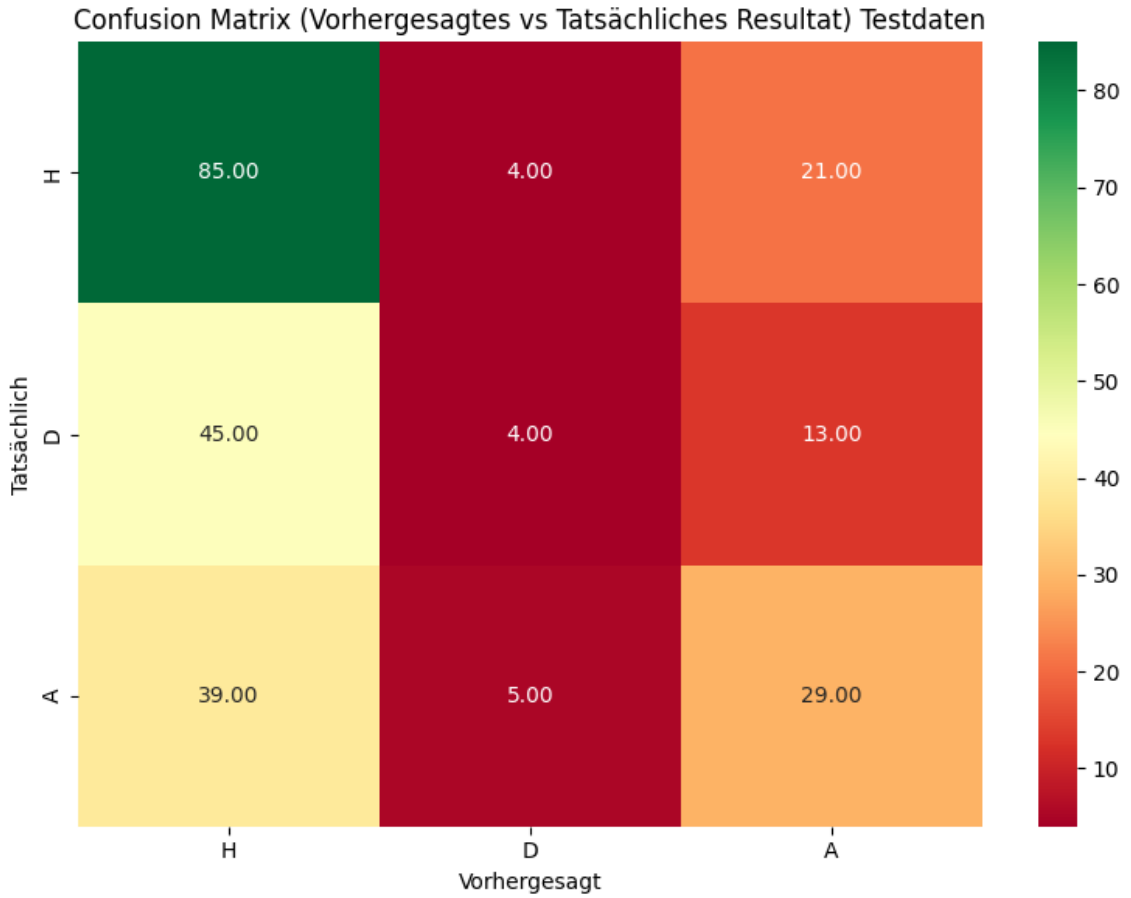
4.5.2.1Classification Report

Classification Report (Saison Testdaten)

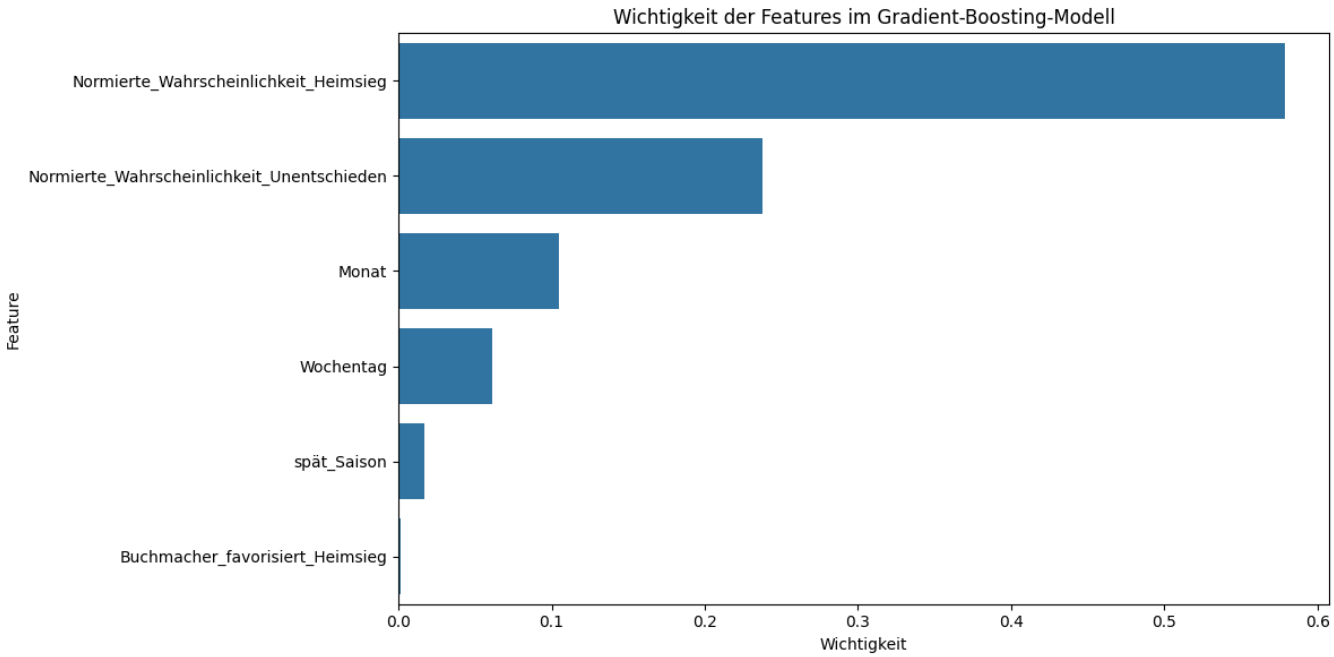
Klasse	precision	recall	f1-score	support
A	0.46	0.4	0.43	73.0
D	0.31	0.06	0.11	62.0
H	0.5	0.77	0.61	110.0
accuracy	0.48	0.48	0.48	0.48

accuracy: 0.48

4.5.2.2 Konfusionsmatrix



4.5.2.3 Feature-Wichtigkeit



4.5.3 Evaluation des Modells hinsichtlich der Validierungsdaten

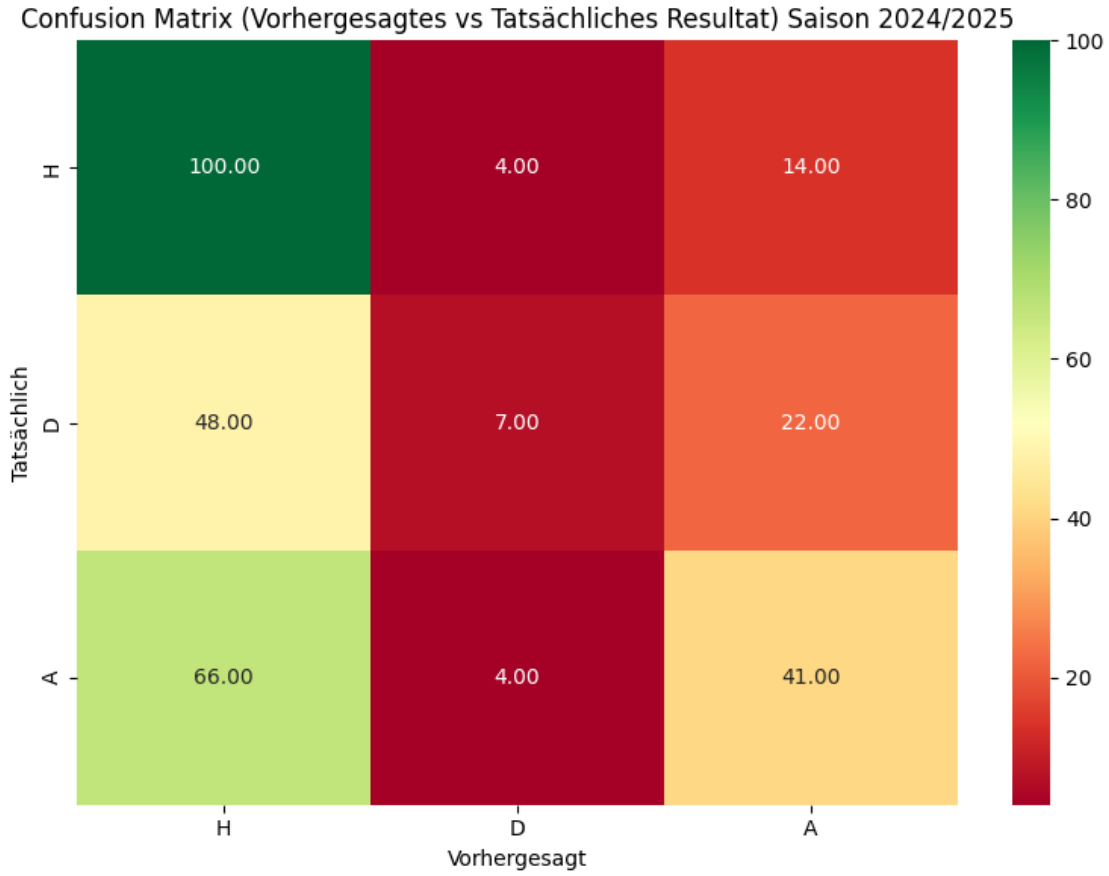
4.5.3.1 Classification Report

Classification Report (Saison 2024/2025)

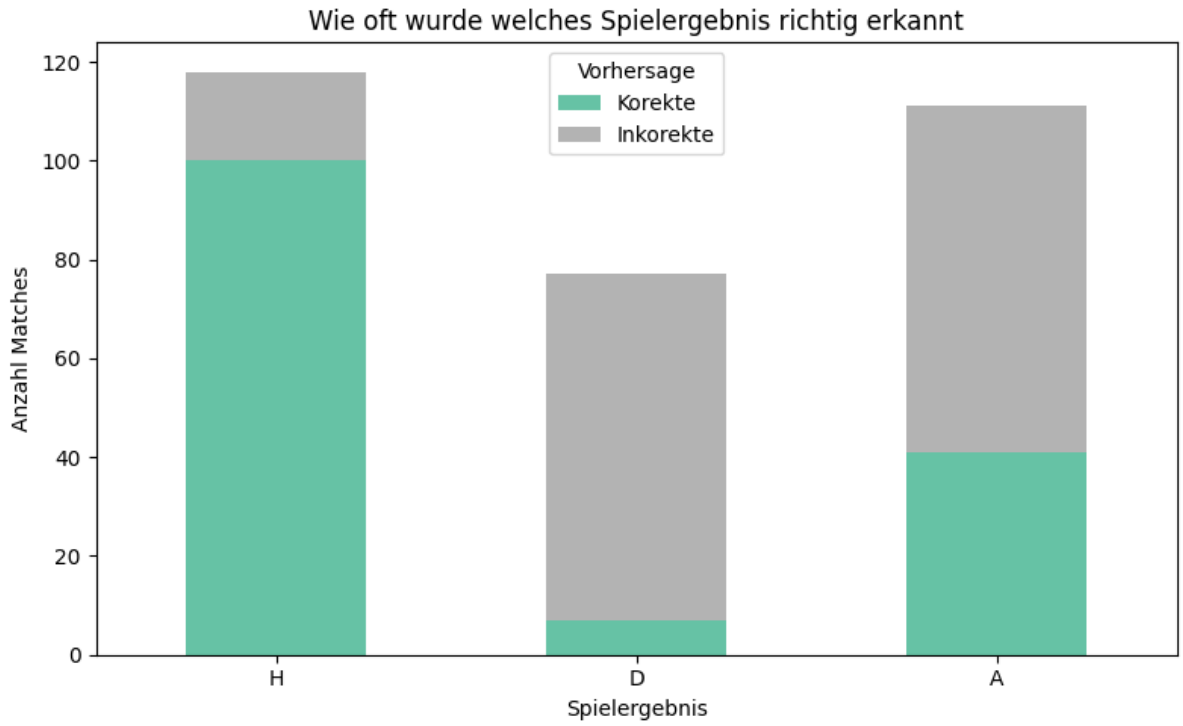
Klasse	precision	recall	f1-score	support
A	0.53	0.37	0.44	111.0
D	0.47	0.09	0.15	77.0
H	0.47	0.85	0.6	118.0

Accuracy: 0.48

4.5.3.2 Konfusionsmatrix



4.5.3.3 Balkendiagramm der Vorhersagegenauigkeit



5 Diskussion

5.1 Interpretation der Ergebnisse

Aus der empirischen Analyse geht hervor, dass sowohl traditionelle statistische Modelle als auch moderne Machine-Learning-Ansätze brauchbare Prognosen ermöglichen. Das erste Modell, das Buchmacher-Modell, dient als solide Benchmark. Die Wettquoten entstehen zunächst aus Vorhersagen statistischer und Machine-Learning-Modelle, die anschliessend unter Berücksichtigung der Marktmechanismen angepasst werden. Dadurch liefern sie naturgemäss sehr gute Vorhersagen.

Die logistische Regression liefert transparente und interpretierbare Ergebnisse, ist aber bei der Erfassung komplexer, nichtlinearer Zusammenhänge eingeschränkt. Dadurch schneidet es bei den Validierungsdaten sehr schlecht ab, durch kleine Anpassungen im Python-Code sind jedoch bessere Resultate erzielbar (vgl. Kapitel 5.2)

Der Gradient-Boosting kann komplexe Muster in den Daten erkennen und liefert insgesamt eine höhere Vorhersagegenauigkeit. Diese Prognoseleistung konnte, durch das neu eingeführte Feature den EM-Score, sogar zusätzlich verbessert werden.

Insgesamt verdeutlichen die Ergebnisse, dass die Wahl des Modells sowie die Berücksichtigung relevanter Features entscheidend für die Qualität der Vorhersagen ist.

5.2 Bewertung der Modelle

5.2.1 Das Buchmacher-Modell

Wie der Classification Report zeigt, schneidet das Modell insgesamt gut ab und weist mit 52 % die höchste Accuracy aller betrachteten Modelle auf. Die Kennzahlen Precision, Recall und F1-Score sind für Heim- und Auswärtssiege zufriedenstellend. Besonders auffällig ist der sehr hohe Recall-Wert für Heimsiege, der zeigt, dass 86 % der tatsächlichen Heimsiege korrekt vorhergesagt wurden.

Aus der Konfusionsmatrix lässt sich jedoch erkennen, dass das Modell 208 Spiele als Heimsieg klassifiziert hat. Dadurch relativiert sich der hohe Recall-Wert, da ein Modell, das eine Kategorie häufig wählt, automatisch einen höheren Recall für diese Kategorie erzielt.

Ein weiteres Problem ist, dass die restlichen 98 Spiele alle als Auswärtssiege vorhergesagt wurden. Damit wurde kein einziges Unentschieden vorhergesagt und somit auch nicht vom Modell erkannt, was die Null-Werte für Precision, Recall und F1-Score für diese Kategorie erklärt.

Dadurch wird auch die Accuracy verzerrt, denn wenn eine Kategorie die selten eintrifft, in diesem Beispiel das Unentschieden, vom Modell komplett ignoriert wird, steigt scheinbar die Genauigkeit, da die verbleibenden Vorhersagen nun immer in den häufig eintretenden Klassen liegen.

5.2.1.1 Aggregierter F1 Score

Um die Modelle auch abgesehen von ihrem Accuracy-Wert zu vergleichen, wird ein aggregierter F1-Score gebildet. In dieser Arbeit wird der Macro-F1-Score dafür verwendet, da er wie in Kapitel 3.1.2 beschrieben, die F1-Scores aller Klassen gleich gewichtet und somit auch seltene Ereignisse wie z.B. ein Unentschieden fair berücksichtigt.

0.54 (F1_Heimsieg) + 0 (F1_Unentschieden) + 0.63 (F1_Auswärtssieg) = **0.39** (F1_Gesamt)

5.2.2 Model Logistische Regression

Für die Testdaten schneidet das Modell der logistischen Regression solide ab. Der Classification Report sowie die Konfusionsmatrix zeigen, dass das Modell alle drei Spielausgänge vorhersagt und dabei eine Accuracy von 46 % erreicht. Auf den ersten Blick liegt dieser Wert unter dem des Buchmacher-Modells. Im Gegensatz dazu sagt die logistische Regression jedoch auch Unentschieden vorher, was sich in den F1-Kennzahlen positiv bemerkbar macht.

Um die beiden Modelle fair miteinander zu vergleichen, wird erneut der Macro-F1-Score gebildet.

0.56(F1_Heimsieg) + 0.26(F1_Unentschieden) + 0.51(F1_Auswärtssieg) ≈ **0.44**(F1_Gesamt)

Dieser Wert zeigt, dass die logistische Regression, über alle Klassen hinweg eine ausgewogenere Leistung erbringt als das Buchmacher-Modell, auch wenn die reine Accuracy im Vergleich niedriger ausfällt.

5.2.2.1 Feature Wichtigkeit

Das Balkendiagramm zur Feature-Wichtigkeit zeigt, dass sich das Modell in seinen Vorhersagen zu rund 35 % auf das Feature Normierte_Wahrscheinlichkeit_Heimsieg stützt. An zweiter Stelle folgt das Dummy-Feature Buchmacher_favorisiert_Heimsieg mit etwa 10 %. Danach tragen die Kalender-Features Monat und Spät_Saison mit jeweils ungefähr 5 % zur Vorhersage bei. Am wenigsten Bedeutung haben die Normierte_Wahrscheinlichkeit_Unentschieden sowie der Wochentag, die vom Modell kaum berücksichtigt werden.

Dass die Normierte_Wahrscheinlichkeit_Heimsieg den grössten Einfluss hat, ist nachvollziehbar, da Buchmacherquoten generell, und vor allem in diesem Bereich, besonders genau sind, da sie bereits selbst viele relevante Informationen bündeln (vgl. Kapitel 2.1 + 2.4.1). Auch das Feature Buchmacher_favorisiert_Heimsieg liefert einen sinnvollen Beitrag, da es direkt angibt, ob der Buchmacher den Heimsieg oder den Auswärtssieg favorisiert, eine Information, die für die Ergebnisprognosen sehr relevant sind.

Weniger stark ins Gewicht fallen dagegen die Kalender-Features Monat und Spät_Saison. Sie liefern zwar zusätzliche Informationen, ihre Aussagekraft für die Vorhersage ist jedoch gering, wodurch das Modell sich weiterhin auf die stärkeren Einflussfaktoren konzentriert. Besonders auffällig ist, dass die Normierte_Wahrscheinlichkeit_Unentschieden nur schwach genutzt wird, was sich negativ auf die Fähigkeit des Modells auswirken kann, Unentschieden korrekt vorherzusagen. Am wenigsten Bedeutung hat schliesslich der Wochentag, was plausibel ist, da dieser Faktor keinen direkten Zusammenhang mit Spielergebnissen aufweist.

5.2.2.2 Anwendung auf die Validierungsdaten

Der Classification Report zeigt, dass das Modell mit einer Accuracy von 25 % sehr schlecht abschneidet. Die Kennzahlen des Reports sowie die Konfusionsmatrix verdeutlichen, dass das Modell fast ausschliesslich Unentschieden vorhersagt. Deshalb ist der Recall für diese Klasse mit 1.00 sehr hoch, da das Modell alle Unentschieden korrekt erkennt, was logisch ist, wenn es nahezu nur Unentschieden vorhersagt.

Für die Spielausgänge Heimsieg und Auswärtssieg fällt der Recall hingegen extrem niedrig aus, was ebenfalls erklärbar ist. Denn werden diese Spielausgänge kaum oder gar nicht vorhergesagt, kann das Modell sie auch nicht korrekt erkennen, weshalb der Recall automatisch sehr niedrig ausfällt.

Die Precision-Werte spiegeln dasselbe Bild wider. Für die Klasse Heimsieg erscheint der Precision-Wert mit 1.00 sehr hoch, was jedoch trügerisch ist, da nur ein einziges Spiel als Heimsieg vorhergesagt wurde und dies korrekt war, ist zwar der Wert mathematisch sehr hoch, obwohl das Modell insgesamt kaum in der Lage ist, Heimsiege zuverlässig vorherzusagen.

Die Precision für das Unentschieden liegt bei 0.25. Da das Modell fast alle Spiele als Unentschieden klassifiziert, zeigt dieser Wert, dass etwa 25 % der Spiele tatsächlich unentschieden enden und erklärt gleichzeitig, warum die Accuracy des Modells ebenfalls bei 25 % liegt.

Für den Auswärtssieg liegt die Precision bei 0, ebenso wie bei allen anderen Kennzahlen dieser Kategorie, da kein Spiel in der Klasse Auswärtssieg vorhergesagt wurde.

Alle diese Faktoren erklären die niedrigen F1-Scores für Heimsieg und Auswärtssieg, einzig für die Klasse Unentschieden wird mit 0.4 eine akzeptable Leistung erreicht.

5.2.3 Angepasstes Logistische-Regressions-Modell

Diese auffälligen Werte, insbesondere, dass das Modell fast nur Unentschieden in den Validierungsdaten vorhersagt, lassen sich dadurch erklären, dass die Klasse Unentschieden in den Trainingsdaten am schwierigsten zu lernen war. Ohne gezieltes Klassen-Gewichtungs-Tuning hat das Modell Probleme, diese seltene Klasse korrekt zu erkennen, was sich auch an den niedrigen Kennzahlen für das Unentschieden in den Trainingsdaten zeigt.

Gleichzeitig sind die anderen Klassen leichter zu trennen oder liefern klarere Signale, sodass das Modell sie in den Trainingsdaten zuverlässiger vorhersagt, was wiederum zu höheren Kennzahlen für diese Klassen führt.

Aufgrund dieser Trainingsdynamik versucht das Modell, die Leistung auf den Validierungsdaten auszugleichen, was jedoch zu einer Überkompensation führt. Nun prognostiziert es ausschliesslich Unentschieden.

Wie bereits beschrieben, kann durch ein gezieltes Parameter-Tuning eine bessere Vorhersagegenauigkeit und vor allem ein brauchbareres Modell erreicht werden, das auch andere Spielereignisse korrekt erkennen kann.

Dazu wurde die folgende Codezeile angepasst:

log_reg = LogisticRegression(max_iter=1000, multi_class='multinomial', solver='lbfgs', C=0.01, class_weight="balanced")

zu

log_reg = LogisticRegression(max_iter=1000, multi_class='multinomial', solver='lbfgs', C=0.01, class_weight={"H": 1, "D": 0.8, "A": 1.4})

Hierdurch wird die Klassenverteilung nicht automatisch vom Modell ausgeglichen, sondern manuell vorgegeben. Wenn eine Klasse zu häufig respektive zu selten vorhergesagt wird, kann die Gewichtung entsprechend nach unten respektive oben angepasst werden. Mit Hilfe des Classification Reports und insbesondere der Konfusionsmatrix lässt sich jederzeit nachvollziehen, wie sich die Prognosen pro Klasse ändern.

Durch empirisches Testen und Anpassen der Gewichtungen konnten die Accuracy, Precision, Recall und F1-Score kontinuierlich verbessert werden, wodurch die finalen, aussagekräftigeren Ergebnisse erzielt wurden.

5.2.3.1 Trainings- und Testdaten

Zunächst wird wie gewohnt das Modell mit den Trainingsdaten trainiert und anschliessend auf den Testdaten getestet. Daraus ergibt sich folgender Classification Report.

Classification Report (Saison Testdaten)

Klasse	precision	recall	f1-score	support
A	0.49	0.71	0.58	73.0
D	0.0	0.0	0.0	62.0
H	0.59	0.74	0.65	110.0
accuracy	0.54	0.54	0.54	0.54

accuracy: 0.54

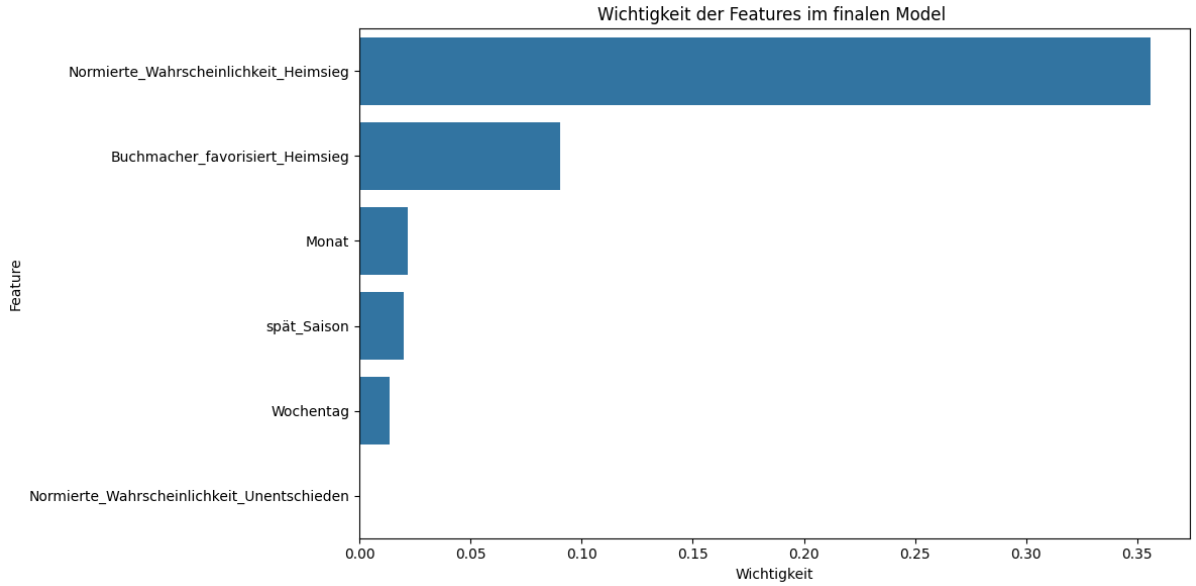
Auf den ersten Blick stellt dies eine Verbesserung zu den Vorherigen Testdaten dar, da die Accuracy von 0.46 auf 0.54 stieg.

Bei genauerer Betrachtung zeigt sich jedoch, dass diese Verbesserung vor allem dadurch zustande kommt, dass das Modell in den Testdaten kein Unentschieden vorhersagt. Dies kann daran liegen, dass die Testdaten nur eine kleinere Stichprobe enthalten, wodurch seltene Ereignisse wie Unentschieden noch seltener vertreten sind. Dadurch hat das Modell weniger Gelegenheit, diese Fälle korrekt zu erkennen.

Vergleicht man die beiden Modelle, jedoch hinsichtlich ihres Macro-F1-Scores, fällt auf, dass das Modell vor dem Parameter-Tuning (0.44) minimal besser abgeschnitten hat als das aktuelle Modell (0.41). Der Unterschied kommt vor allem zustande, da der F1-Score für die Klasse Unentschieden, beim jetzigen Modell deutlich niedriger ausfällt.

5.2.3.2 Feature Wichtigkeit

Um besser zu verstehen, warum bestimmte Spielausgänge häufiger oder seltener korrekt vorhergesagt werden, wird in diesem Abschnitt die Bedeutung der einzelnen Features für die Modellprognosen untersucht.



Im Grossen und Ganzen zeigen sich keine wesentlichen Unterschiede zum Modell ohne Parameter-Tuning. Auffällig ist jedoch, dass das Feature *Normierte_Wahrscheinlichkeit_Unentschieden* im getunten Modell keine Bedeutung mehr hat.

Dies erklärt teilweise, warum die Testdaten so schlecht für die Vorhersage von Unentschieden abschneiden, denn das Modell ignoriert das Feature, das eigentlich die relevantesten Signale für diese Klasse liefert.

5.2.3.3 Anwendung auf die Validierungsdaten

Ein anderes Bild zeigt sich beim Anwenden auf die Validierungsdaten. Hier schneidet das Modell mit Parametertuning deutlich besser ab. Nicht nur die Gesamt-Accuracy ist höher, auch die F1-Scores für Heimsieg und Auswärtssieg verbessern sich deutlich, lediglich der Wert für ein Unentschieden ist marginal schlechter als zuvor.

Aus dem Classification Report lässt sich jedoch erkennen, dass vermutlich zu viele Unentschieden vorhergesagt wurden, und zwar vor allem auf Kosten der Auswärtssiege. Dies zeigt sich insbesondere an den Recall-Werten der Klassen, Rund 60 % der tatsächlichen Unentschieden werden korrekt erkannt, während der Recall-Wert für Heim- und insbesondere für Auswärtssiege deutlich niedriger ausfällt.

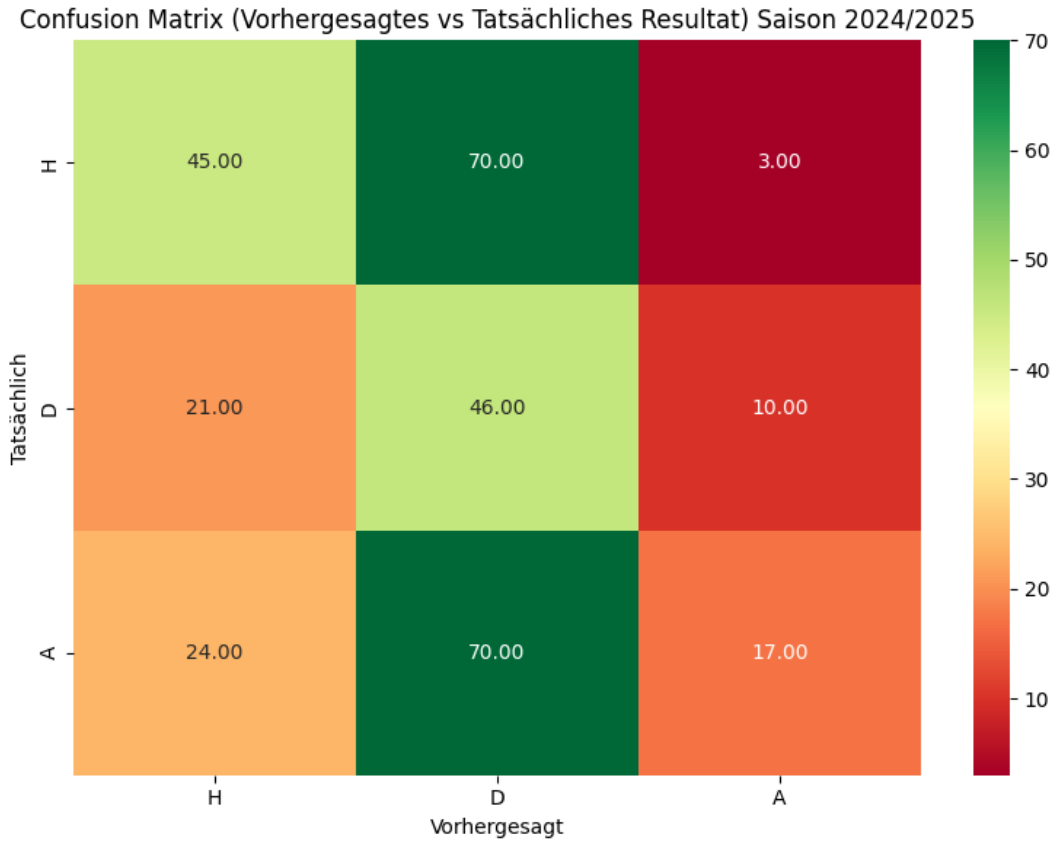
Auch der Precision-Wert bestätigt dieses Bild. Für die Klasse „Unentschieden“ ist er mit 0.25 sehr gering, was bedeutet, dass nur jede vierte Vorhersage auf ein Unentschieden tatsächlich zutrifft. Bei den anderen Klassen liegt die Precision hingegen über 50 %, sodass mehr als die Hälfte der Spiele, die als Heim- oder Auswärtssieg prognostiziert wurden, auch tatsächlich so endeten.

Classification Report (Saison 2024/2025)

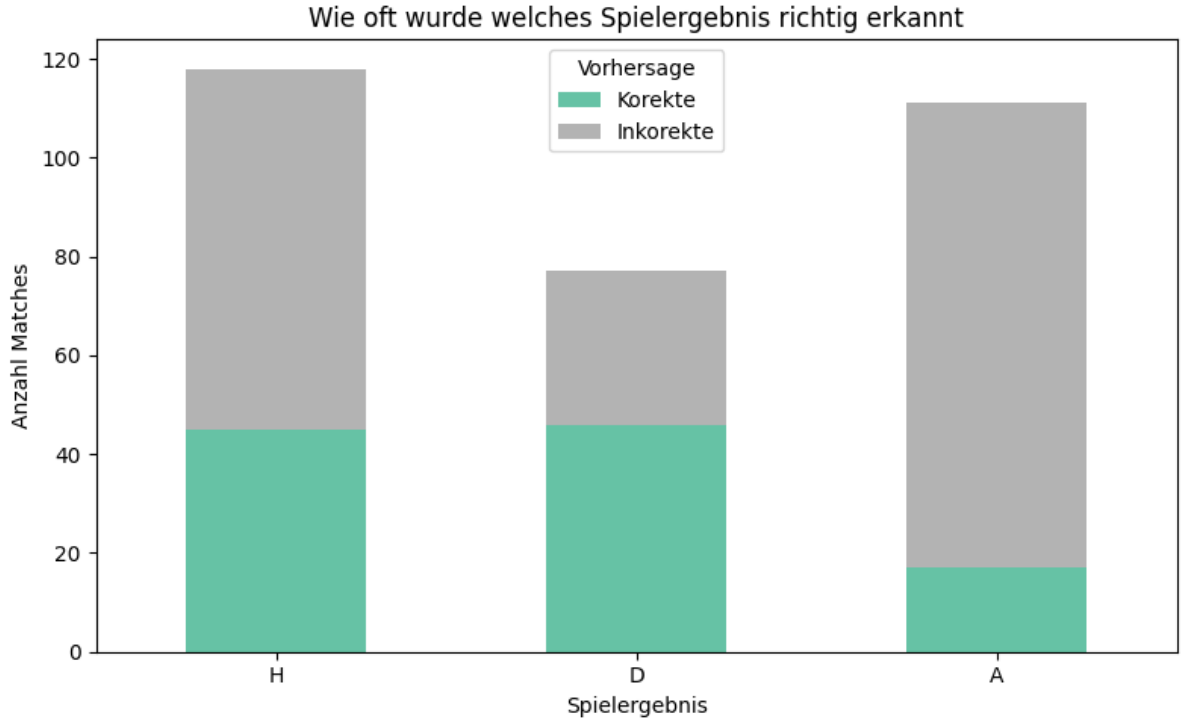
Klasse	precision	recall	f1-score	support
A	0.57	0.15	0.24	111.0
D	0.25	0.6	0.35	77.0
H	0.5	0.38	0.43	118.0

Accuracy: 0.35

Die Konfusionsmatrix verdeutlicht, was bereits im Classification Report sichtbar wurde. Das Modell deckt zwar alle Klassen ab, sagt jedoch zu viele Spiele als Unentschieden voraus und prognostiziert kaum Auswärtssiege, insgesamt nur 30 Spiele. Dies ist wenig realistisch, da Unentschieden, wie man anhand des Supports der einzelnen Klassen im Classification Report erkennt, deutlich seltener vorkommen. Mit weiterem Parametertuning könnte dies möglicherweise verbessert werden. Erste Versuche haben jedoch gezeigt, dass sich das Modell insgesamt verschlechtert, sobald dieses Ungleichgewicht reduziert wird.



Auch das Balkendiagramm bestätigt dieses Muster, Heimsiege und Unentschieden werden vergleichsweise gut erkannt, während der Auswärtssieg am schwächsten abschneidet. Was logisch ist, wenn nur insgesamt 30 Spiele als Auswärtssieg vorhergesagt werden.



5.2.3.4 Aggregierter F1-Score

Um dieses Modell mit den anderen Ansätzen fair zu vergleichen, wird erneut der Macro F1-Score berechnet.

0.43 (F1_Heimsieg) + 0.35 (F1_Unentschieden) + 0.2.4 (F1_Auswärtssieg) = **0.34** (F1_Gesamt)

5.2.4. Modell Gradient-Boosting

5.2.4.1 Trainings- und Testdaten

Das Gradient-Boosting-Modell zeigt in Bezug auf die Testdaten auf den ersten Blick eine solide Leistung. Der Classification Report weist eine Accuracy von 48 % aus, was zunächst nach einem ausgeglichenen Bild klingt. Bei genauerer Betrachtung wird jedoch deutlich, dass der Recall-Wert für die Klasse „Unentschieden“ extrem niedrig ist. Nur etwa 6 % aller tatsächlichen Unentschieden werden auch als solche erkannt. Im Gegensatz dazu ist der Recall-Wert für einen Heimsiege auffallend hoch (0,77), wodurch der F1-Score für Heimsiege entsprechend steigt, während der F1-Score für Unentschieden deutlich niedriger ausfällt.

Die Konfusionsmatrix verdeutlicht dieses Problem zusätzlich. Das Modell prognostiziert überwiegend Heimsiege (169 Spiele), während Auswärtssiege mit 63 Fällen noch im normalen Rahmen liegen. Unentschieden hingegen werden mit lediglich 13 Vorhersagen fast vollständig vernachlässigt.

Die absoluten Zahlen, der Testsaison, erscheinen auf den ersten Blick niedrig, sind aber im Kontext des gewählten Splits von 80 % Trainings- zu 20 % Testdaten nachvollziehbar. Dadurch wird der Grossteil der Spiele für das Training verwendet, sodass nur ein Fünftel für die Testphase zur Verfügung steht.

5.2.4.2 Feature Wichtigkeit

Das Balkendiagramm zur Feature-Wichtigkeit zeigt ein klares Bild, Rund 60 % der Modellvorhersagen basieren auf dem Feature Normierte_Wahrscheinlichkeit_Heimsieg. An zweiter Stelle folgt die Normierte_Wahrscheinlichkeit_Unentschieden mit etwa 30 %. Die restlichen Features haben nur eine sehr geringe Bedeutung. Dazu zählen insbesondere die Kalender-Features wie Monat oder Spät_Saison, die jeweils nur wenige Prozentpunkte beitragen. Kaum Einfluss hat dagegen das Dummy-Feature Buchmacher_favorisiert_Heimsieg, dessen Wert nahezu bei null liegt.

Im Vergleich zur logistischen Regression fällt besonders auf, dass die Bedeutung der Normierten_Wahrscheinlichkeit_Heimsieg von etwa 35 % auf rund 60 % gestiegen ist. Gleichzeitig hat das Dummy-Feature Buchmacher_favorisiert_Heimsieg, das zuvor noch rund 10 % ausmachte, nahezu keine Relevanz mehr. Stattdessen gewinnt die Normierte_Wahrscheinlichkeit_Unentschieden deutlich an Gewicht (von kaum Bedeutung auf rund 30 %).

Eine mögliche Erklärung für diese Verschiebung ist, dass Gradient-Boosting im Gegensatz zur linearen Logit-Funktion auch komplexere und nichtlineare Muster in den Daten erkennen kann. Dadurch wird das Modell in der Lage, die Quoteninformationen direkter und stärker zu nutzen, anstatt wie die logistische Regression noch auf Hilfsvariablen wie das Dummy-Feature angewiesen zu sein. Zudem reagiert Gradient-Boosting flexibler auf seltenere Ereignisse, was erklären könnte, warum die Normierte_Wahrscheinlichkeit_Unentschieden an Bedeutung gewinnt. Denn sie trägt zusätzliche Information für die schwer vorherzusagende Klasse „Unentschieden“ bei.

Dass die normierte Wahrscheinlichkeit für den Heimsieg den grössten Einfluss hat, ist in beiden Modellen erwartbar, da Quoten im Fussballbereich sehr präzise sind und bereits eine Vielzahl von relevanten Faktoren einfliessen (vgl. Kapitel 2.1 + 2.4.1)).

5.2.4.3 Anwendung auf die Validierungsdaten

Auf den Validierungsdaten zeigt das Gradient-Boosting-Modell ein sehr ähnliches Muster wie bereits bei den Testdaten. Wie zuvor werden vor allem Heimsiege vorhergesagt (224 Spiele), während Unentschieden nahezu vollständig unterrepräsentiert sind (14 Spiele). Die Accuracy liegt mit 46 % leicht unter dem Wert der Testdaten. Der F1-Wert für Heim- und Auswärtssieg ist geringfügig niedriger als bei den Testdaten, während der F1-Wert für Unentschieden mit 0,13 leicht höher ausfällt.

Die Konfusionsmatrix sowie das Balkendiagramm bestätigen dieses Bild. Heimsiege werden wieder überwiegend korrekt erkannt, während Unentschieden nach wie vor kaum vorhergesagt werden. Insgesamt zeigt sich im Vergleich zu den Testdaten ein noch extremeres Muster, da mehr Heimsiege prognostiziert werden. Die leichten Unterschiede zu den Testdaten entstehen durch die unterschiedliche Zusammensetzung der Validierungsdaten, was zu minimalen Verschiebungen in Accuracy und F1-Werten führt.

5.2.4.4 Aggregierter F1-Score

Um die Modelle fair miteinander zu vergleichen, wird wieder der Macro-F1-Score gebildet.

0.59 (F1_Heimsieg) + 0.13 (F1_Unentschieden) + 0.38 (F1_Auswärtssieg) = **0.37** (F1_Gesamt)

5.2.5 Modell Gradient-Boosting mit EM-Feature

Da das Gradient-Boosting-Modell mit dem EM-Feature identisch zu dem Modell ohne EM-Feature trainiert wird, unterscheiden sich weder die Trainingsdaten noch die Feature-Wichtigkeiten zwischen den beiden Varianten.

5.2.5.1 Anwendung auf die Validierungsdaten

Der Classification Report sowie die Konfusionsmatrix zeigen ein ähnliches Bild wie das Modell ohne EM-Feature. Dennoch ist erkennbar, dass sich jede Kennzahl minimal verbessert hat. Dadurch ist in jeder Klasse beziehungsweise jedem Spielereignis der F1-Score minimal gestiegen, was zu einer höheren Accuracy von 0.48 (zuvor 0.46) führt.

Man kann somit festhalten, dass das Modell insgesamt robuster und präziser geworden ist, ohne dass dies auf eine systematische Bevorzugung oder Benachteiligung einzelner Klassen zurückzuführen wäre.

Die Einbindung des selbst entwickelten Features verdeutlicht den entscheidenden Einfluss von Feature-Engineering auf die Prognosequalität. Selbst kleine, gezielt entwickelte Indikatoren können die Vorhersagegenauigkeit verbessern, indem sie Aspekte der Teamleistung erfassen, die in Standardvariablen (wie den Bet365-Quoten) wahrscheinlich nicht enthalten sind.

Insgesamt bestätigt sich, dass die Kombination aus Modellwahl, gezielter Anpassung und durchdachtem Feature-Engineering entscheidend für robuste und präzise Vorhersagen ist.

5.2.5.2 Aggregierter F1-Score

Um auch dieses Modell mit den anderen zu vergleichen, wird abschliessend der Macro-F1-Score berechnet.

0.6 (F1_Heimsieg) + 0.15 (F1_Unentschieden) + 0.44 (F1_Auswärtssieg) = **0.4** (F1_Gesamt)

5.3 Limitationen der Analyse

In diesem Kapitel werden die Limitationen der Analyse dargestellt, bedeutet Aspekte, die die Aussagekraft der Ergebnisse einschränken oder nicht vollständig berücksichtigt wurden.

5.3.1Datengrundlage

Die Analyse basiert ausschliesslich auf den Validierungsdaten aus einer Bundesliga-Saisons. Dadurch ergibt sich eine eingeschränkte Datenbasis, die die Stabilität der Ergebnisse beeinflussen kann, denn Fussball ist ein dynamischer Sport, bei dem sich Mannschaftsstärken, Spielstile und externe Faktoren von Saison zu Saison verändern. Somit besteht die Gefahr, dass die Ergebnisse stark saisonabhängig sind und sich nicht ohne Weiteres auf andere Spielzeiten übertragen lassen. Eine breitere Datenbasis über mehrere Jahre hinweg könnte robustere und verallgemeinerbare Ergebnisse ermöglichen.

5.3.2 Modellvereinfachungen

Im Rahmen dieser Arbeit wurden die Modelle bewusst vereinfacht, um die Analyse überschaubar zu halten. Diese Vereinfachungen betrafen vor allem die Modellarchitektur. So wurde lediglich einzelne Modelle trainiert und evaluiert. Komplexere Ansätze wie Ensemble-Methoden, bei denen mehrere Modelle separat trainiert und anschliessend über Verfahren wie einen Voting Classifier kombiniert werden, hätten potenziell zu robusteren und stabileren Vorhersagen geführt. Solche Verfahren wurden zum Beispiel im Referenzmodell erfolgreich eingesetzt.

Die Parametrisierung der Modelle erfolgte überwiegend auf Basis empirischer Analysen. Systematische Optimierungsverfahren wie Grid Search oder Random Search, die eine feinere Abstimmung der Hyperparameter ermöglichen, wurden nicht angewendet. Dadurch besteht die Möglichkeit, dass die verwendeten Einstellungen nicht die global optimalen Parameter darstellen.

Darüber hinaus stützen sich die Modelle stark auf die Wahrscheinlichkeiten der Buchmacher. Diese bieten zwar eine solide Grundlage für Vorhersagen, bilden aber nicht alle relevanten Informationen ab, die für differenziertere Prognosen notwendig wären.

Zusammenfassend erleichtern diese methodischen Vereinfachungen zwar die Nachvollziehbarkeit, verhindern jedoch, dass das Modell sein volles Potenzial ausschöpfen kann. Ein stärker datengetriebener Ansatz mit systematischer Hyperparameter-Optimierung und Ensemble-Techniken hätte hier voraussichtlich zu besseren Ergebnissen geführt.

5.3.3 Begrenzte Feature-Auswahl

Ein weiteres Problem liegt in der begrenzten Menge und Aussagekraft der verwendeten Features. Zwar wurde mit dem EM-Feature ein selbst entwickelter Indikator ergänzt, dennoch bleibt die Modellbasis stark auf den Quoten von Bet365 fokussiert. Diese bilden jedoch nur einen Teil der Spielrealität ab. Wesentliche Einflussgrössen wie detaillierte Spielerstatistiken (z. B. Expected Goals, Passquoten, Zweikampfwerte), taktische Formationen, historische Direktvergleiche oder aktuelle Belastungsfaktoren konnten nicht einbezogen werden. Die geringe Feature-Tiefe limitiert die Vorhersagefähigkeit des Modells erheblich und verhindert, dass die Komplexität von Fussballspielen adäquat abgebildet wird.

5.3.4 Generalisation

Da das Modell nur mit Spielen der Bundesliga der Saison 2020-2024 trainiert wurde, ist die Übertragbarkeit der Ergebnisse auf andere Ligen oder Wettbewerbe nur eingeschränkt möglich. Unterschiede in Spielstilen, Wettbewerbsniveaus, Kaderstrukturen oder auch in den Rahmenbedingungen der Wettmärkte können dazu führen, dass die Prognosegüte in anderen Kontexten deutlich geringer ausfällt. Eine Erweiterung auf mehrere Ligen oder internationale Wettbewerbe wäre notwendig, um die Generalisierbarkeit der Resultate belastbarer zu prüfen.

6 Konklusion

6.1 Zusammenfassung der zentralen Ergebnisse

In dieser Arbeit wurden unterschiedliche Ansätze zur Vorhersage von Fussballergebnissen untersucht und anhand ausgewählter gängigen Metriken (Accuracy, Precision, Recall, F1 Score und Micro-F1 Score) evaluiert. Dabei zeigte sich, dass die Modelle nur eingeschränkt in der Lage sind, den komplexen Verlauf von Spielen zuverlässig zu prognostizieren, jedoch deutliche Unterschiede in der Modellgüte erkennbar wurden.

Das Buchmacher-Modell erzielte mit einem Micro-F1 Score von 0.39 eine sehr solide Performance. Die **logistische Regression** stellte eine robuste Baseline dar, erreichte jedoch nur einen Wert von **0.34**. Das **Gradient-Boosting-Modell** konnte mit **0.37** leicht bessere Resultate erzielen. Durch die Integration des neu entwickelten **EM-Scores** liess sich dieser Wert steigern auf **0.40** und schliesst somit am besten von allen untersuchten Modellen ab.

Dadurch wird gezeigt, dass datengetriebene Modelle durchaus das Potenzial haben, brauchbare Prognosen zu liefern. Besonders die Erweiterung um zusätzliche Einflussfaktoren wie den EM-Score kann, wenn auch in begrenztem Umfang, die Vorhersagequalität verbessern.

Darüber hinaus wurde klar, dass die Verfügbarkeit geeigneter Daten eine zentrale Herausforderung darstellt. Mehrere ursprünglich geplante Ansätze konnten nicht umgesetzt werden, da Datensätze nicht zugänglich oder unvollständig waren. Dies machte methodische Anpassungen erforderlich, wie etwa den Wechsel von der Saison 2022/2023 zur Saison 2024/2025, wodurch ein potenzielles Daten-Leakage vermieden werden konnte.

6.2 Beantwortung der Forschungsfragen

Die Arbeit orientierte sich an drei zentralen Forschungsfragen, die in Kapitel 1.2 vorgestellt wurden und im Folgenden beantwortet werden.

6.2.1 Vergleich von Vorhersagemodellen

Der direkte Vergleich zeigte, dass keines der Modelle in allen untersuchten Bereichen überlegen war. Während die logistische Regression eine einfache, interpretierbare Baseline bildete (Micro-F1 = 0.34), konnten Gradient Boosting (0.37) und das Buchmacher-Modell (0.39) punktuell bessere Leistungen erzielen. Das erweiterte Gradient-Boosting-Modell mit EM-Feature erreichte den höchsten Wert (0.40), wobei die Unterschiede insgesamt moderat blieben.

6.2.2 Berücksichtigung von Machine-Learning-Ansätzen

Mit der logistischen Regression und dem Gradient-Boosting-Modell wurden zwei datengetriebene Verfahren untersucht. Beide zeigten, dass Machine Learning in der Lage ist, Muster in den Spieldaten zu identifizieren, wenngleich die Prognosequalität durch die begrenzte Datenbasis eingeschränkt blieb. Ohne zusätzliche Features konnten die Modelle nicht an die Performance des Buchmacher-Modells heranreichen. Dies erscheint plausibel, da das Buchmacher-Modell nicht nur auf statistischen Verfahren beruht, sondern zusätzlich die Einschätzungen und das Verhalten des Wettmarktes berücksichtigt.

6.2.3 Einführung eines neuen Einflussfaktors

Der neu entwickelte EM-Score stellte einen bislang kaum berücksichtigten Einflussfaktor dar und konnte im Gradient-Boosting-Modell zu einer Verbesserung des Micro-F1 Scores von 0.37 auf 0.40 beitragen. Auch wenn der Zugewinn gering ausfiel, bestätigt dies die Relevanz von Feature Engineering für datengetriebene Prognosemodelle im Fussballkontext.

6.3 Ausblick auf zukünftige Forschung

Für zukünftige Arbeiten ergeben sich mehrere Anknüpfungspunkte:

1. **Erweiterung der Datenbasis**: Der Zugang zu detaillierteren Spieler- und Teamstatistiken (z. B. Schüsse, Laufleistung, Passquoten) könnte die Modellqualität erheblich steigern.
2. **Weitere Modellansätze**: Weitere Modellansätze: Neben den hier getesteten Verfahren könnten auch erweiterte Varianten des Gradient Boosting, neuronale Netze oder hybride Ansätze untersucht werden
3. **Validierung über mehrere Ligen und Zeiträume**: Um die Generalisierbarkeit zu prüfen, sollten die Modelle auf andere Ligen und Saisons angewendet werden.
4. **Integration ökonomischer Aspekte**: Ein Vergleich mit den Quoten der Buchmacher bietet Potenzial für wirtschaftswissenschaftliche Betrachtungen, etwa im Hinblick auf Wettstrategien.

Darüber hinaus wäre es interessant, den EM-Score nicht nur in das Gradient-Boosting-Modell zu integrieren, sondern auch zu prüfen, ob sich durch seine Einbindung das Buchmacher-Modell weiter verbessern liesse. Diese Fragestellung konnte im Rahmen der Arbeit aus Umfangsgründen nicht vertieft werden, bietet aber Potenzial für zukünftige Untersuchungen

Insgesamt zeigt die Arbeit, dass datengetriebene Modelle wertvolle Einsichten liefern können, ihre Leistungsfähigkeit aber stark von der Datenqualität und der Auswahl geeigneter Features abhängt.

Literaturverzeichnis

Andretta, M. (o. J.), *AIFootballPredictions* [GitHub Repository]. Abgerufen am 10. Oktober 2025 von https://github.com/MauroAndretta/AIFootballPredictions

Atta Mills, E. F., Deng, Z., Zhong, Z., & Li, J. (2024), „Data-driven prediction of soccer outcomes using enhanced machine and deep learning techniques“, *Journal of Big Data*, 11, Artikel Nr. 170.

Bundesliga-Prognose.de (o. J.), „Startseite / Informationen zur Bundesliga-Prognose“. Abgerufen am 10. Oktober 2025 von https://www.bundesliga-prognose.de

Christoffersson, E. (2023), *Beating the Odds: Machine Learning Models vs. Betting Companies in Football Prediction*. Jönköping: Jönköping University (Masterarbeit).

Constantinou, A. C., & Fenton, N. E. (2013), „Profiting from arbitrage and odds biases of the European football gambling market“, *The Journal of Gambling Business and Economics*, 7(2), pp. 41–70.

Dambroz, F., Clemente, F. M., & Teoldo, I. (2022), “The effect of physical fatigue on the performance of soccer players: A systematic review“, *PLoS ONE*, 17(7), e0270099.

Dixon, M. J., & Coles, S. G. (1997), „Modelling association football scores and inefficiencies in the football betting market“, *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 46(2), pp. 265–280.

Draper, G., Wright, M., Chesterton, P., & Atkinson, G. (2021), „The tracking of internal and external training loads with next-day player-reported fatigue at different times of the season in elite soccer players“, *International Journal of Sports Science & Coaching*, 16(3), pp. 793–803.

FIFA (o. J.), „FIFA World Ranking – Men“. Abgerufen am 10. Oktober 2025 von https://inside.fifa.com/de/fifa-world-ranking/men?dateId=id14338

Forrest, D., & Simmons, R. (2002), „Outcome Uncertainty and Attendance Demand in Sport: The Case of English Soccer“, *Journal of the Royal Statistical Society: Series D (The Statistician)*, 51(2), pp. 229–241.

Hubáček, O., Sourek, G., & Zelezný, F. (2019), „Exploiting sports-betting market using machine learning“, in *Proceedings of the Conference on Machine Learning Applications*, pp. 112–120.

IBM (o. J.), „Classification versus regression“. Abgerufen am 10. Oktober 2025 von https://www.ibm.com/think/topics/classification-vs-regression

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021), *An Introduction to Statistical Learning: With Applications in R* (2nd ed.). New York: Springer.

KDnuggets (2023), „Micro, Macro & Weighted Averages of F1 Score, Clearly Explained". Abgerufen am 9. Oktober 2025 von https://www.kdnuggets.com/2023/01/micro-macro-weighted-averages-f1-score-clearly-explained.html

Király, F., & Qian, Z. (2017), „Modelling competitive sports: Bradley–Terry–Élő models for supervised and on-line learning of paired competition outcomes", *arXiv Preprint*. Abgerufen von https://arxiv.org/abs/1707.04783

Kolibrisport (2025), „Die 8 wichtigsten Faktoren bei der Sportwetten-Analyse". Abgerufen am 10. Oktober 2025 von https://kolibrisport.ch/2025/08/01/die-8-wichtigsten-faktoren-bei-der-sportwetten-analyse/

Liga-Zwei.de (o. J.), „Quoten berechnen". Abgerufen am 10. Oktober 2025 von https://www.liga-zwei.de/sportwetten/blog/quoten-berechnen

90min.de (2024), „EM-Kader 2024: Aufgebote Überblick". Abgerufen am 10. Oktober 2025 von https://www.90min.de/posts/em-kader-2024-aufgebote-uberblick

Markopoulou, C., Papageorgiou, G., & Tjortjis, C. (2024), „Diverse machine learning for forecasting goal scoring likelihood in elite football leagues", *Machine Learning and Knowledge Extraction*, 6(3), pp. 1762–1781.

Moustakidis, S., Plakias, S., Kokkotis, C., Tsatalas, T., & Tsaopoulos, D. (2023), Predicting Football Team Performance with Explainable AI: Leveraging SHAP to Identify Key Team-Level Performance Metrics", *Future Internet*, 15(5), Artikel Nr. 174.

Parim, C., Güneş, M. Ş., Büyüklü, A. H., & Yıldız, D. (2021), „Prediction of match outcomes with multivariate statistical methods for the group stage in the UEFA Champions League", *Journal of Human Kinetics*, 79(1), pp. 197–209.

Pietraszewski, P., Terbalyan, A., Roczniok, R., Maszczyk, A., Ornowski, K., Manilewska, D., Kuliś, S., Zając, A., & Gołaś, A. (2025), „The role of artificial intelligence in sports analytics: A systematic review and meta-analysis of performance trends", *Applied Sciences*, 15(13), pp. 7254.

Pollard, R. (2006), „Worldwide regional variations in home advantage in association football", *Journal of Sports Sciences*, 24, pp. 231–240.

Pollard, R. (2008), „Home advantage in football: A current review of an unsolved puzzle", *The Open Sports Sciences Journal*, 1(1), pp. 12–14.

Scikit-learn developers (2024), „Supervised Learning“. Abgerufen am 9. Oktober 2025 von https://scikit-learn.org/stable/supervised_learning.html

Sportwettentest.net (o. J.), „Sportwetten-Geschichte: Die ersten Schritte im Fußball“. Abgerufen am 10. Oktober 2025 von https://www.sportwettentest.net/sportwetten-geschichte/#Die_ersten_Schritte_im_Fußball

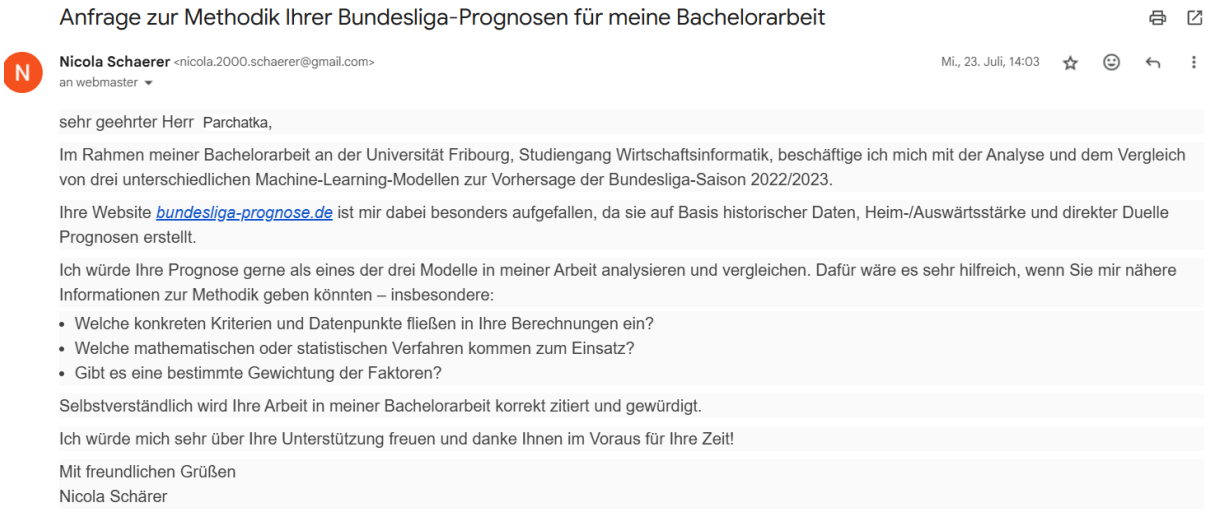
Tsokos, A., Narayanan, S., Kosmidis, I., Baio, G., Cucuringu, M., Whitaker, G., & Király, F. J. (2019), „Modeling outcomes of soccer matches“, *Machine Learning*, 108, pp. 77–95.

Walsh, C., & Joshi, A. (2023), „Machine learning for sports betting: Should model selection be based on accuracy or calibration?“. Abgerufen am 10. November 2025 von https://arxiv.org/abs/2310.19004

Wette.de (o. J.), „Sportwetten Auszahlungsquoten Erklärt: Verstehen & Gewinnen“. Abgerufen am 12. Oktober 2025 von https://www.wette.de/sportwetten/auszahlungsquoten/

Anhang

Anhang A: Informationsbeschaffung



Anfrage zur Methodik Ihrer Bundesliga-Prognosen für meine Bachelorarbeit

Nicola Schaerer <nicola.2000.schaerer@gmail.com> Mi., 23. Juli, 14:03
an webmaster

sehr geehrter Herr Parchatka,

Im Rahmen meiner Bachelorarbeit an der Universität Fribourg, Studiengang Wirtschaftsinformatik, beschäftige ich mich mit der Analyse und dem Vergleich von drei unterschiedlichen Machine-Learning-Modellen zur Vorhersage der Bundesliga-Saison 2022/2023.

Ihre Website *bundesliga-prognose.de* ist mir dabei besonders aufgefallen, da sie auf Basis historischer Daten, Heim-/Auswärtsstärke und direkter Duelle Prognosen erstellt.

Ich würde Ihre Prognose gerne als eines der drei Modelle in meiner Arbeit analysieren und vergleichen. Dafür wäre es sehr hilfreich, wenn Sie mir nähere Informationen zur Methodik geben könnten – insbesondere:

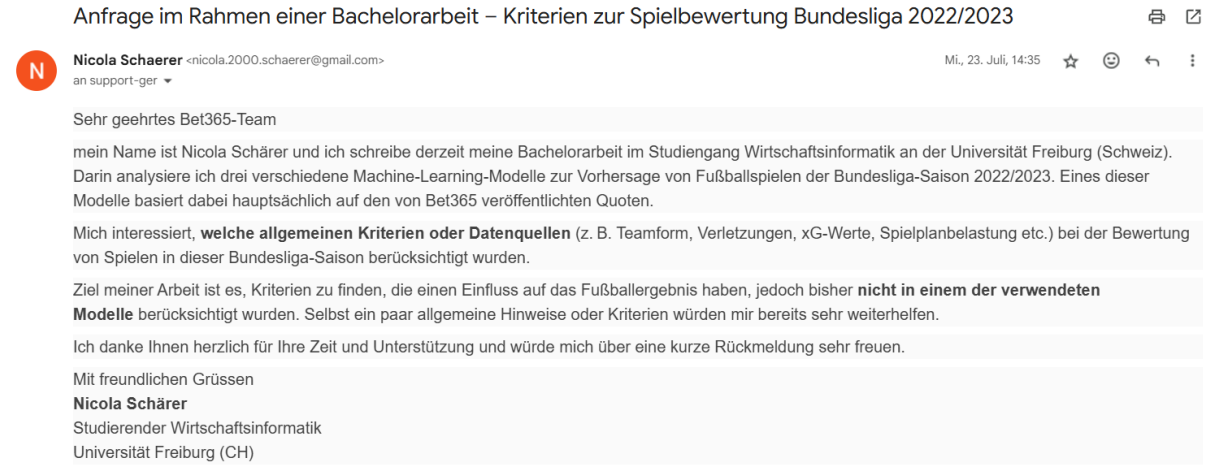
- Welche konkreten Kriterien und Datenpunkte fließen in Ihre Berechnungen ein?
- Welche mathematischen oder statistischen Verfahren kommen zum Einsatz?
- Gibt es eine bestimmte Gewichtung der Faktoren?

Selbstverständlich wird Ihre Arbeit in meiner Bachelorarbeit korrekt zitiert und gewürdigt.

Ich würde mich sehr über Ihre Unterstützung freuen und danke Ihnen im Voraus für Ihre Zeit!

Mit freundlichen Grüßen
Nicola Schärer

(Anhang 1: E-Mail vom 23.07.2025)



Anfrage im Rahmen einer Bachelorarbeit – Kriterien zur Spielbewertung Bundesliga 2022/2023

Nicola Schaerer <nicola.2000.schaerer@gmail.com> Mi., 23. Juli, 14:35
an support-ger

Sehr geehrtes Bet365-Team

mein Name ist Nicola Schärer und ich schreibe derzeit meine Bachelorarbeit im Studiengang Wirtschaftsinformatik an der Universität Freiburg (Schweiz). Darin analysiere ich drei verschiedene Machine-Learning-Modelle zur Vorhersage von Fußballspielen der Bundesliga-Saison 2022/2023. Eines dieser Modelle basiert dabei hauptsächlich auf den von Bet365 veröffentlichten Quoten.

Mich interessiert, **welche allgemeinen Kriterien oder Datenquellen** (z. B. Teamform, Verletzungen, xG-Werte, Spielplanbelastung etc.) bei der Bewertung von Spielen in dieser Bundesliga-Saison berücksichtigt wurden.

Ziel meiner Arbeit ist es, Kriterien zu finden, die einen Einfluss auf das Fußballergebnis haben, jedoch bisher **nicht in einem der verwendeten Modelle** berücksichtigt wurden. Selbst ein paar allgemeine Hinweise oder Kriterien würden mir bereits sehr weiterhelfen.

Ich danke Ihnen herzlich für Ihre Zeit und Unterstützung und würde mich über eine kurze Rückmeldung sehr freuen.

Mit freundlichen Grüssen
Nicola Schärer
Studierender Wirtschaftsinformatik
Universität Freiburg (CH)

(Anhang 2: E-Mail vom 23.07.2025)

Anhang B: Datengrundlage

Div	Date	Time	HomeTeam	AwayTeam	FTHG	FTAG	FTR	HTHG	HTAG	HTR	HS	AS	HST	AST	HF	AF	HC	AC	HY	AY	HR	AR	B365H	B365D	B365A
D1	23.08.2024	19:30	M'gladbach	Leverkusen	2	3	A	0	2	A	14	25	7	9	11	7	2	4	1	0	0	0	5.25	4.5	1.55
D1	24.08.2024	14:30	Augsburg	Werder Bremen	2	2	D	2	1	H	10	12	3	6	11	10	4	2	3	1	0	0	2.2	3.6	3.1

BWH	BWD	BWA	PSH	PSD	PSA	WHH	WHD	WHA	MaxH	MaxD	MaxA	AvgH	AvgD	AvgA	B365>2.5	B365<2.5	P>2.5	P<2.5	Max>2.5	Max<2.5	Avg>2.5	Avg<2.5	AHh	B365AHH	B365AHA
5	4.4	1.58	5.33	4.58	1.6	5	4.33	1.57	5.5	4.78	1.62	5.16	4.53	1.58	1.44	2.75	1.5	2.71	1.51	2.75	1.47	2.68	1	1.98	1.92
2.25	3.5	3	2.32	3.55	3.11	2.25	3.5	3.1	2.35	3.8	3.16	2.28	3.56	3.07	1.73	2.1	1.74	2.18	1.77	2.23	1.72	2.15	-0.25	1.99	1.91

PAHH	PAHA	MaxAHH	MaxAHA	AvgAHH	AvgAHA	B365CH	B365CD	B365CA	BWCH	BWCD	BWCA	PSCH	PSCD	PSCA	WHCH	WHCD	WHCA	MaxCH	MaxCD	MaxCA	AvgCH	AvgCD	AvgCA	B365C>2.5	B365C<2.5
1.98	1.94	2.01	1.94	1.93	1.92	4.75	4.5	1.62	4.75	4.33	1.63	4.94	4.38	1.67	5	4.4	1.57	5.6	4.62	1.67	5.09	4.46	1.6	1.44	2.75
2.02	1.89	2.02	1.95	1.96	1.9	2.45	3.6	2.7	2.45	3.5	2.7	2.57	3.61	2.77	2.45	3.6	2.7	2.57	3.69	3.16	2.46	3.54	2.82	1.67	2.2

PC>2.5	PC<2.5	MaxC>2.5	MaxC<2.5	AvgC>2.5	AvgC<2.5	AHCh	B365CAHH	B365CAHA	PCAHH	PCAHA	MaxCAHH	MaxCAHA	AvgCAHH	AvgCAHA
1.48	2.79	1.48	2.9	1.45	2.8	1	1.86	2.04	1.87	2.06	1.96	2.08	1.91	1.96
1.71	2.26	1.75	2.3	1.69	2.21	0	1.88	2.05	1.88	2.04	1.88	2.06	1.86	2.03

(Anhang 3: Ausschnitt der verwendeten CSV-Datei des Referenzmodells)

Anhang C: Berechnung des neuen Features (EM-Score)

Weltrangliste Platz vor dem Turnier	Nationalmannschaft	Mindesterwartung vor dem Turnier	Tatsächliches Endergebnis	Wm-Score
1	Brasilien	Halbfinale	Viertelfinale	-1
2	Belgien	Halbfinale	Gruppenphase	-3
3	Argentinien	Halbfinale	Sieger	3
4	Frankreich	Halbfinale	Finale	1
5	England	Viertelfinale	Viertelfinale	0
7	Spanien	Viertelfinale	Achtelfinale	-1
8	Niederlande	Viertelfinale	Viertelfinale	0
9	Portugal	Viertelfinale	Viertelfinale	0
10	Dänemark	Achtelfinale	Gruppenphase	-1
11	Deutschland	Achtelfinale	Gruppenphase	-1
12	Kroatien	Achtelfinale	Halbfinale	2
13	Mexiko	Achtelfinale	Gruppenphase	-1
14	Uruguay	Achtelfinale	Gruppenphase	-1
15	Schweiz	Achtelfinale	Achtelfinale	0
16	USA	Achtelfinale	Achtelfinale	0
18	Senegal	Achtelfinale	Achtelfinale	0
19	Wales	Gruppenphase	Gruppenphase	0
20	Iran	Gruppenphase	Gruppenphase	0
21	Serbien	Gruppenphase	Gruppenphase	0
22	Marokko	Gruppenphase	Halbfinale	3
23	Japan	Gruppenphase	Achtelfinale	1
25	Polen	Gruppenphase	Achtelfinale	1
27	Südkorea	Gruppenphase	Achtelfinale	1
29	Tunesien	Gruppenphase	Gruppenphase	0
30	Costa Rica	Gruppenphase	Gruppenphase	0
38	Australien	Gruppenphase	Achtelfinale	1
41	Kanada	Gruppenphase	Gruppenphase	0
43	Kamerun	Gruppenphase	Gruppenphase	0
44	Ecuador	Gruppenphase	Gruppenphase	0
50	Katar	Gruppenphase	Gruppenphase	0
51	Saudi-Arabien	Gruppenphase	Gruppenphase	0
61	Ghana	Gruppenphase	Gruppenphase	0
Legende				
Gruppenphase	0			
Achtelfinale	1			
Viertelfinale	2			
Halbfinale	3			
Finale	4			
Sieger	6			

(Anhang 4: Berechnung des WM-Scores pro Land)

Mannschaften	Anzahl Spieler an der WM	Wm-Score total
FC Bayern München	16	2
Borussia Dortmund	12	-12
RB Leipzig	7	-2
1. FC Union Berlin	1	-1
SC Freiburg	5	0
Bayer 04 Leverkusen	4	3
Eintracht Frankfurt	7	1
VfL Wolfsburg	3	-3
1. FSV Mainz 05	3	1
Borussia Mönchengladbach	6	1
1. FC Köln	1	0
TSG 1899 Hoffenheim	2	1
Werder Bremen	2	-1
VfL Bochum	1	1
FC Augsburg	3	1
VfB Stuttgart	2	3
FC Schalke 04	1	1
Hertha BSC	1	-1
Gesamt	77	

(Anhang 5: Berechnung des WM-Scores pro Bundesliga Team)

Weltrangliste Platz vor dem Turnier	Nationalmannschaft	Mindesterwartung vor dem Turnier	Tatsächliches Endergebnis	Em-Score
2	Frankreich	Halbfinale	Halbfinale	0
3	Belgien	Halbfinale	Achtelfinale	-0.2
4	England	Halbfinale	Finale	0.1
6	Portugal	Halbfinale	Viertelfinale	-0.1
7	Niederlande	Viertelfinale	Halbfinale	0.1
8	Spaninen	Viertelfinale	Sieger	0.4
9	Italien	Viertelfinale	Achtelfinale	-0.1
10	Kroatien	Viertelfinale	Gruppenphase	-0.2
16	Deutschland	Achtelfinale	Viertelfinale	0.1
19	Schweiz	Achtelfinale	Viertelfinale	0.1
21	Dänemark	Achtelfinale	Achtelfinale	0
22	Ukraine	Achtelfinale	Gruppenphase	-0.1
25	Österreich	Achtelfinale	Achtelfinale	0
27	Ungarn	Achtelfinale	Gruppenphase	-0.1
28	Polen	Achtelfinale	Gruppenphase	-0.1
33	Serbien	Achtelfinale	Gruppenphase	-0.1
36	Tschechien	Gruppenphase	Gruppenphase	0
39	Schottland	Gruppenphase	Gruppenphase	0
40	Türkei	Gruppenphase	Viertelfinale	0.2
46	Rumänien	Gruppenphase	Achtelfinale	0.1
48	Slowakei	Gruppenphase	Achtelfinale	0.1
57	Slowenien	Gruppenphase	Achtelfinale	0.1
66	Albanien	Gruppenphase	Gruppenphase	0
75	Georgien	Gruppenphase	Achtelfinale	0.1

(Anhang 6: Berechnung des **EM-Scores** pro Land)

Mannschaften	Anzahl Spieler an der Em	Em-Score total
FC Bayern München	10	0.7
Bayer 04 Leverkusen	10	0.7
Eintracht Frankfurt	1	0.1
Borussia Dortmund	8	0.8
SC Freiburg	4	-0.2
1. FSV Mainz 05	2	0.1
RB Leipzig	11	0.4
Werder Bremen	2	-0.1
VfB Stuttgart	5	0.5
Borussia Mönchengladbach	2	0.1
VfL Wolfsburg	8	-0.5
FC Augsburg	1	0.1
1. FC Union Berlin	3	-0.3
FC St. Pauli	0	0
TSG 1899 Hoffenheim	6	0.1
1. FC Heidenheim	0	0
Holstein Kiel	0	0
VfL Bochum	0	0
Gesamt	73	

(Anhang 7: Berechnung des **EM-Scores** pro Bundesliga Team)

Anhang D: Implementierung der ausgewählten Modelle

```
import pandas as pd
# Import der pandas-Bibliothek und verseht sie mit dem namen pd.
# pandas wird hier für die Datenanalyse und die Datenmanipulation verwendet.
# Sie ist eine leistungsstarke Bibliothek und basiert auf NumPy.
# Zudem bietet pandas verschiedene Datenstrukturen unteranderem DataFrames und
Series, dies ermöglicht mit tabellarischen und zeitbasierten Daten zu
arbeiten.

import numpy as np
# Import der numpy-Bibliothek und verseht sie mit dem namen np.
# NumPy wird hier für das wissenschaftliche Rechnen mit Python verwendet.
# Es bietet leistungsstarke Datenstrukturen wie beispielsweise Arrays, es
bietet unteranderem auch Funktionen für mathematische Operationen, lineare
Algebra und Statistik.
# numpy ist besonders effizient bei der Verarbeitung von grossen Daten und es
bildet zudem die Grundlage für viele weitere Bibliotheken, die wir verwenden
werden wie z.B. pandas, scipy oder scikit-learn.

import matplotlib.pyplot as plt
# 'matplotlib.pyplot' ist ein Modul, das für Erstellungen von Diagrammen und
Visualisierungen verwendet wird.
# Meistens braucht man es für einfache Plots wie z.B. Liniendiagramme,
Balkendiagramme oder Streudiagramme.

import seaborn as sns
# 'seaborn' basierend auf 'matplotlib' und ist eine Bibliothek, die speziell
für statistische Visualisierungen entwickelt wurde.
# Sie stellt Standarddesigns und Funktionen bereit wie z.B. Heatmaps, Boxplots
oder Korrelationsmatrizen.

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
# scikit-learn (oder sklearn) ist eine Bibliothek, die für das maschinelle
Lernen in Python verwendet wird.
# Sie stellt viele bekannte Algorithmen zur Verfügung, wie beispielsweisse für
die Klassifikation, die Regression und das Clustering.
# Sie bietet zudem Werkzeuge für Modelltraining, Vorhersage, Evaluation und
Datenvorverarbeitung an.

from sklearn.preprocessing import StandardScaler

Bundesliga_Saison_2425 = pd.read_csv("data/raw2425/D1_merged.csv")

Tatsächliches_Resultat_2425 = Bundesliga_Saison_2425["FTR"]   # echtes
Ergebnis
```

```
Bundesliga_Saison_2425["Favorit_Buchmacher"] =
Bundesliga_Saison_2425[["B365H", "B365D",
"B365A"]].idxmin(axis=1).str.extract(r"B365(.)")[0]
# Hier wird geschaut, welche der drei Optionen (Heimsieg, Unentschieden,
Auswärtssieg) die niedrigste Quote hat, und zwar für jede einzelne Spielzeile.
# Dies macht man, da man so die favorisierte Option vom Buchmacher (Bet365)
zurückerhält.
# Da ja folgende Formel gilt: Wahrscheinlichkeit = 1 / Quote, das bedeutet je
tiefer die Quote, desto höher die Wahrscheinlichkeit.
# Mit 'idxmin(axis=1)' zeigt man, den Namen der Spalte, die den kleinsten Wert
pro Zeile hat.
# Mit  folgendem Aufruf 'str.extract(r"B365(.)")' extrahiert man den
Buchstaben (H,D,A) aus dem gesamten Spaltennamen also z.B extrahiert man das H
bei 'B365H'

accuracy_2425 = accuracy_score(Tatsächliches_Resultat_2425,
Bundesliga_Saison_2425["Favorit_Buchmacher"]) # Wir messen, zuerst wie gut die
Vorhergesagten Resultate mit den tatsächlichen Resultaten übereinstimmen
(Accuracy).
report_2425 = classification_report(Tatsächliches_Resultat_2425,
Bundesliga_Saison_2425["Favorit_Buchmacher"], output_dict=True) # zusätzlich
wird ein detaillierten Bericht mit Precision, Recall und F1-Score pro Klasse
erstellt.
labels = ["H", "D","A"] # # Definiert die Labels Heimsieg, Unentschieden und
Auswärtssieg in dieser Reihenfolge. Bereite so die Labels auf Einheitlichkeit
vor.

# Confusion Matrix
Confusion_Matrix = confusion_matrix(Tatsächliches_Resultat_2425,
Bundesliga_Saison_2425["Favorit_Buchmacher"] , labels=labels)# Erstellt die
Confusion Matrix für das Modell auf Basis der Testdaten
Confusion_Matrix_final = pd.DataFrame(Confusion_Matrix, index=labels,
columns=labels) # Wandelt die Matrix in ein DataFrame um, um sie später als
Heatmap darzustellen zu können.


# Visualisierung der Confusion Matrix als Heatmap
plt.figure(figsize=(8, 6))# Gibt die grösse der Grafik vor
sns.heatmap(Confusion_Matrix_final, #erstellt eine Heatmap der Confusion
Matrix mit hilfe von Seaborn.
            annot=True, #zeigt die numerischen Werte der Confusion Matrix an.
            fmt=".2f", #rundet auf 2 Nachkommastellen
            cmap="RdYlGn")#Verseht die Heatmap mit einer Farbskala
plt.title("Confusion Matrix (Vorhergesagtes vs Tatsächliches Resultat) Saison
2024/2025")# Die Grafik wird mit einem Titel versehen
plt.xlabel("Vorhergesagt") #X-Achse wird beschriftet
plt.ylabel("Tatsächlich")#Y-Achse wird beschriftet
plt.tight_layout()# Hier wird das Layout optimiert, indem nichts abgeschnitten
wird
plt.savefig('Confusion_Matrix_2425 .png')# Speichert die Grafik
```

```
plt.show()#Zeigt die finale Grafik (Heatmap)

correct_mask = (Bundesliga_Saison_2425["Favorit_Buchmacher"] ==
Tatsächliches_Resultat_2425.values) # Balkendiagramm das die Korrekten
Vorhersagen und die falschen Vorhersagen pro Klasse zeigt
correct_labels =
Tatsächliches_Resultat_2425[correct_mask].value_counts().reindex(labels,
fill_value=0) # Zählt alle korrekten Vorhersagen pro Klasse
incorrect_labels =
Tatsächliches_Resultat_2425[~correct_mask].value_counts().reindex(labels,
fill_value=0) # Zählt alle inkorrekten Vorhersagen pro Klasse.


# Erstellt ein DataFrame für gestapelte Balken
Balkendiagramm = pd.DataFrame({
    "Korekte": correct_labels,
    "Inkorekte": incorrect_labels
}, index=labels)


# Zeichnet ein gestapeltes Balkendiagramm
Balkendiagramm.plot(kind="bar", stacked=True, figsize=(8, 5), colormap="Set2")
plt.title("Wie oft wurde welches Spielergebnis richtig erkannt")
plt.xlabel("Spielergebnis")
plt.ylabel("Anzahl Matches")
plt.xticks(rotation=0)
plt.legend(title="Vorhersage")
plt.tight_layout()
plt.savefig('Genauigkeit_2425.png')
plt.show()


finaler_report = pd.DataFrame(report_2425).T # 3. Klassifikations-Report als
Tabelle anzeigen
finaler_report = finaler_report.drop(index=[ "accuracy", "macro avg",
"weighted avg"], errors="ignore") # Entfernt zusammenfassende / Redunate
Zeilen
finaler_report = finaler_report[["precision", "recall", "f1-score",
"support"]].round(2) # Wählt alle relevanten Metriken und runde sie auf 2
Nachkommastellen.
finaler_report.index.name = "Klasse"
finaler_report.reset_index(inplace=True)

# --- Klassifikations-Report als Tabelle + Accuracy separat darstellen ---
accuracy_value = report_2425["accuracy"]

fig, ax = plt.subplots(figsize=(8, 3.5))
ax.axis('off')

# Tabelle zeichnen
```

```
table = ax.table(
    cellText=finaler_report.values,
    colLabels=finaler_report.columns,
    loc='center',
    cellLoc='center'
)

table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

# Zebra-Stil für bessere Übersicht
for i in range(len(finaler_report)):
    for j in range(len(finaler_report.columns)):
        color = "#f0f0f0" if i % 2 == 0 else "white"
        table[(i+1, j)].set_facecolor(color)

# Kopfzeile hervorheben
for j in range(len(finaler_report.columns)):
    table[(0, j)].set_facecolor("#d0d0d0")
    table[(0, j)].set_text_props(weight='bold')

# Titel
plt.title("Classification Report (Saison 2024/2025)")

# Accuracy separat fett darunter darstellen
plt.text(0.5, -0.25, f"Accuracy: {accuracy_value:.2f}",
         ha="center", va="center", fontsize=12, fontweight="bold")

plt.tight_layout()
plt.savefig("Classification_Report_2425.png", bbox_inches="tight")
plt.show()
```

(Anhang 8: Baseline-Modell)

```python
import pandas as pd
# Import der pandas-Bibliothek und verseht sie mit dem namen pd.
# pandas wird hier für die Datenanalyse und die Datenmanipulation verwendet.
# Sie ist eine leistungsstarke Bibliothek und basiert auf NumPy.
# Zudem bietet pandas verschiedene Datenstrukturen unteranderem DataFrames und
Series, dies ermöglochen mit tabellarischen und zeitbasierten Daten zu
arbeiten.

import numpy as np
# Import der numpy-Bibliothek und  verseht sie mit dem namen np.
# NumPy  wird hier für das wissenschaftliche Rechnen mit Python verwendet.
# Es bietet leistungsstarke Datenstrukturen wie beispielsweise Arrays, es
bietet unteradnderm auch Funktionen für mathematische Operationen, lineare
Algebra und Statistik.
# numpy ist besonders effizient bei der Verarbeitung von grossen Daten und es
bildet zudem die Grundlage für viele weitere Bibliotheken die wir verwenden
werden wie z.B. pandas, scipy oder scikit-learn.

import matplotlib.pyplot as plt
# 'matplotlib.pyplot' ist ein Modul das für erstellungen von Diagrammen und
visualisierungen verwendet wird.
# Meistens braucht man es für einfache Plots wie z.B. Liniendiagramme,
Balkendiagramme oder Streudiagramme.

import seaborn as sns
# 'seaborn' basierend auf 'matplotlib' und ist eine Bibliothek, die speziell
für statistische Visualisierungen entwickelt wurde.
# Sie stellt Standarddesigns und Funktionen bereit wie z.B. Heatmaps, Boxplots
oder Korrelationsmatrizen.

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
# scikit-learn (oder sklearn) ist eine Bibliothek die für das maschinelles
Lernen in Python verwendet wird.
# Sie stellt viele bekannte Algorithmen zurverfügung, wie beispielsweisse für
die Klassifikation, die Regression und das Clustering.
# Sie bietet zudem Werkzeuge für Modelltraining, Vorhersage, Evaluation und
Datenvorverarbeitung an.

from sklearn.preprocessing import StandardScaler

Bundesliga_Saison_2425 = "data/raw2425/D1_merged.csv"
# Definiert den Pfad zu den Rohdaten der Bundesliga-Saison 2024/2025.
# In diesen Daten enthalten sind vorallem, die Wettqoutten der anstehenden
Spiele der Saison 2024/2025.

Bundesliga_Saison_2021222324 = "data/raw/D1_merged.csv"
```

```
# Definiert den Pfad zur aktuellen Trainingsdatei, dise sind die gleichen
Rohdaten wie oben, einfach für die Bundesligasaison 2020/2021, 2021/2022,
2022/2023, 2023/2024
# Diese Datei wird im nächsten Schritt eingelesen.

Traningsdaten = pd.read_csv(Bundesliga_Saison_2021222324)
# Nun wird die CSV-Datei "data/raw/D1_merged.csv" mit pandas eingelesen und
speichert sie anschliessend als DataFrame.
# Dadurch verfügt der DataFrame 'df_train_raw'  nun über die tabellarischen
Daten aus der Datei "data/raw/D1_merged.csv".


# --- Vorbereitung der Daten für das Feature-Tuning ---
Originle_Traningsdaten = Traningsdaten.copy()
# Fertigt eine Kopie des ursprünglichen DataFrames an, damit die Originaldaten
nicht verändert werden.


# --- 1. Feature-Tunnig = die Buchmacher-Quoten von Bet365 ---
for col in ["B365H", "B365D", "B365A"]:
# Eine for-schleife durchläuft die drei Spalten "B365H", "B365D", "B365A" mit
den Quotten von dem Wettsanbeiter Bet365.
# Dabei steht 'B365H' für Heimsieg, 'B365D' für Unentschieden und 'B365A' für
Auswärtssieg.

    Originle_Traningsdaten[col] = pd.to_numeric(Originle_Traningsdaten[col],
errors="coerce")
# Diese Spalten enthalten eigentlich Strings, unteranderm die eingelesene
Zahlen, aus den CSV-Dateien.
# Mit 'pd.to_numeric()' wird sicher gestelltdass die Werte (die momentan zum
teil aus Strings bestehen) in echte numerische Datentypen umgewandelt werden.
# Dadurch werden verschiedene mathematische Berechnungen wie zum Beispiel
Division, Mittelwert berchnung ermöglicht.
# 'errors="coerce"' sorgt dafür, dass ungültige  Werte (z.B. leere Felder)
automatisch in 'NaN' (Not a Number) umgewandelt werden, dadurch werden sie von
pandas korrekt als fehlende Werte behandelt.

Originle_Traningsdaten["Wahrscheinlichkeit_Heimsieg"] = 1 /
Originle_Traningsdaten["B365H"]
Originle_Traningsdaten["Wahrscheinlichkeit_Unentschieden"] = 1 /
Originle_Traningsdaten["B365D"]
Originle_Traningsdaten["Wahrscheinlichkeit_Auswärtssieg"] = 1 /
Originle_Traningsdaten["B365A"]
# Die Buchmacherquoten "B365H", "B365D", "B365A" zeigen wie hoch die
Auszahlung für 1 Euro Einsatz ist wenn man das richtige Reultat tippt.
# Um nun die Wahrscheinlichkeiten daraus abzuleiten, wird der Kehrwert
berechnet
# Dazu wird die folgende Formel verwendet --> Wahrscheinlichkeit = 1 / Quote
```

```
Normierte_Gesammtwahrscheinlichkeit =
Originale_Traningsdaten["Wahrscheinlichkeit_Heimsieg"] +
Originale_Traningsdaten["Wahrscheinlichkeit_Unentschieden"] +
Originale_Traningsdaten["Wahrscheinlichkeit_Auswärtssieg"]
Originale_Traningsdaten["Normierte_Wahrscheinlichkeit_Heimsieg"] =
Originale_Traningsdaten["Wahrscheinlichkeit_Heimsieg"] /
Normierte_Gesammtwahrscheinlichkeit
Originale_Traningsdaten["Normierte_Wahrscheinlichkeit_Unentschieden"] =
Originale_Traningsdaten["Wahrscheinlichkeit_Unentschieden"] /
Normierte_Gesammtwahrscheinlichkeit
Originale_Traningsdaten["Normierte_Wahrscheinlichkeit_Auswärtssieg"] =
Originale_Traningsdaten["Wahrscheinlichkeit_Auswärtssieg"] /
Normierte_Gesammtwahrscheinlichkeit
# Hier werden die Wahrscheinlichkeiten normiert, also durch die Gesamtsumme
aller drei Wahrscheinlichkeiten geteilt, damit sie zusammen 1 ergeben.
# Dies ist davor nicht zwingend gegeben, da die Buchmacher die Quotten nicht
'fair' gestallten, da sie auch etwas verdiennen wollen.
# Nun haben wir eine echte Wahrscheinlichkeitsverteilung die von diesem
Problem bereinigt ist.
# Das Modell verwendet schlussendliche diese Wahrscheinlichkeiten als Input-
Features.

# --- 2. Feature-Tunnig = der Favorit des Buchmachers (Dummy-Feature) ---
Originale_Traningsdaten["Favorit_Buchmacher"] =
Originale_Traningsdaten[["B365H", "B365D",
"B365A"]].idxmin(axis=1).str.extract(r"B365(.)")[0]
# Hier wird geschaut, welche der drei Optionen (Heimsieg, Unentschieden,
Auswärtssieg) die niedrigste Quotte hat, und zwar für jede einzellne
Spielzeile.
# Dies macht man, da man so die favorisierte Option vom Buchmacher (Bet365)
zurückerhält.
# Da ja folgende Formel gilt: Wahrscheinlichkeit = 1 / Quote, das bedeutet je
tiefer die Quotte, desto höher die Wahrscheinlichkeit.
# Mit 'idxmin(axis=1)' zeigt man, den Namen der Spalte, die den kleinsten Wert
pro Zeile hat.
# Mit  folgendem Aufruf 'str.extract(r"B365(.)")' extrahiert man den
Buchstaben (H,D,A) aus dem gesamten Spaltennamen also z.B extrahiert man das H
bei 'B365H'

Originale_Traningsdaten["Buchmacher_favorisiert_Heimsieg"] =
(Originale_Traningsdaten["Favorit_Buchmacher"] == "H").astype(int)
Originale_Traningsdaten["Buchmacher_favorisiert_Unentschieden"] =
(Originale_Traningsdaten["Favorit_Buchmacher"] == "D").astype(int)
Originale_Traningsdaten["Buchmacher_favorisiert_Auswärtssieg"] =
(Originale_Traningsdaten["Favorit_Buchmacher"] == "A").astype(int)
# Hier werden drei verschiedene Variablen erstellt, diese zeigen, welche
Option (H,D,A) der Buchmacher favorsiert.
# Dies sind binär Variabeln die entweder zeigen, dass der Buchmacher diese
Option favorisiert oder eben halt nicht.
```

```
# Dies sind  nur Dummy-Variabeln, heisst ist ein vernachläsigbare Variable,
wenn man das ganze Modelle betrachtet.


# --- 3. Feature-Tuning: Kalenderinformationen ---

Originale_Traningsdaten["Datum"] =
pd.to_datetime(Originale_Traningsdaten["Date"], dayfirst=True, errors="coerce")
# Originale_Traningsdaten'["Date"]' konvertiert die Spalte "Date" in ein
passendes Datumsformat.
# 'dayfirst=True' sorgt dafür, dass das Datum in folgendem Format (TT/MM/JJJJ)
interpretiert wird.
# 'errors="coerce' stellt sicher, dass ungültige Datumsangaben in NaT (Not a
Time) umgewandelt werden.

Originale_Traningsdaten["Wochentag"] =
Originale_Traningsdaten["Datum"].dt.weekday
# Hier wird der Wochentag mit hilfe des Spieldatum Extrahiert es beginnt bei 0
= Montag, 1 = Dienstag,.... und endet bei 6 = Sonntag

Originale_Traningsdaten["Monat"] = Originale_Traningsdaten["Datum"].dt.month
# Hier wird der Monat mit hilfe des Spieldatum Extrahiert es beginnt bei 1 =
Januar, 2 = Februar,.... und endet bei 12 = Dezember

Originale_Traningsdaten["spät_Saison"] =
Originale_Traningsdaten["Monat"].apply(lambda m: 1 if m in [3,4,5] else 0)
# Mit 'Originale_Traningsdaten["spät_Saison"]'wird ein binäres Feature
erstellt, dies gibt auskunft, ob ein Spiel im März, April oder Mai stattfand.

Tatsächliches_Resultat = Originale_Traningsdaten["FTR"]
# Diese Variable zeigt das tatsächliche Spielergebniss, in dem es die Spalte
FTR (= full time Result) der Datei "data/raw/D1_merged.csv" aussliesst.
# Hier werden nur die folgenden Buchstabe (H,D,A) mit den Werten H (Heimsieg),
D (Unentschieden), A (Auswärtssieg) zurückgegeben.
# Die Anzahl Tore werden also vernachlässigt.

finale_features = [
    "Normierte_Wahrscheinlichkeit_Heimsieg",
"Normierte_Wahrscheinlichkeit_Unentschieden",
"Normierte_Wahrscheinlichkeit_Auswärtssieg", # Enthält die nomierten
Wahrscheinlichkeiten für die 3 Optionen
    "Buchmacher_favorisiert_Heimsieg", "Buchmacher_favorisiert_Unentschieden",
"Buchmacher_favorisiert_Auswärtssieg",# Dummy-Variablen für die Favoriten
Option des Buchmacher
    "Wochentag", "Monat", "spät_Saison" # Kalenderbasierte Features
]
# Hier werden die ausgewählten Merkmale (Features) aufgezählt, die für das
Modell verwendet werden sollen.
```

```
Vollständige_Originale_Traningsdaten = Originale_Traningsdaten[finale_features +
["FTR"]].dropna()
# Hier wird sichergestellt, dass nur vollständige Zeilen verwendet werden,
nicht vollständige Zeilen werden entfernt.

# --- Erstellung der Feature-Arrays und der Ziel-Arrays für das verwendete
Modell ---
Feature_Info = Vollständige_Originale_Traningsdaten[finale_features]
#'Feature_Info' enthält nun die Eingabedaten der ausgewählten Features.
Vollständige_Tatsächliches_Resultat =
Vollständige_Originale_Traningsdaten["FTR"] #
'Vollständige_Tatsächliches_Resultat' ist später unsere Zielvariabele.
Feature_Info.shape, Vollständige_Tatsächliches_Resultat.value_counts() #
'Feature_Info.shape' gibt die Anzahl der Zeilen (Anzahl Spiele) und der
Spalten (Anzahl Features)
# 'Vollständige_Tatsächliches_Resultat.value_counts()' zählt wie oft es
welches Resultat (Heimsieg, Unentscheiden, Auswärtsieg) gab.

# --- Korrelationsmatrix berechnen und visualisieren ---
Korrelationsmatrix = Feature_Info.corr()
# Diese Korrelationsmatrix soll zeigen, wie stark das zwei Variablen
miteinander zusammenhängen.
# Hier liegt der Korrelationswert zwischen -1 und +1
# +1 bedeudet eine perfekte positive Korrelation, wohingegen -1 eine perfekte
negative Korrelation bedeutet und 0 gar kein Zusammenhang zwischen den
Variablen bedeutet.
# Eine zu hohe Korrelation zwischen zwei Features ist nicht wünschenswert, da
sie die Komplexität des Modells erhöht, ohne dem Modell zusätzliche
Informationen zu geben.
# Hier wurde versucht Features die stark korrelieren, zu identifizieren und
anschliessend wurden redundante Merkmale entfernt,
# wie dies bei 'Bookie_Favor_A' z.B der Fall war, dort war das Feature genau
das gegenteil zu 'Bookie_Favor_H'.
# Dieser Prozess sollte Overfitting reduzieren und erhöht somit die
Interpretierbarkeit.


# --- Visualisierung der berechneten Korrelationsmatrix ---
plt.figure(figsize=(16, 6))   #gibt die Grösse der Grafik (Korrelationsmatrix)
vor
sns.heatmap(
            Korrelationsmatrix, #Ist die zuvor berechnete Korrelationsmatrix
            annot=True, #Gibt die Korrelationswerte der einzellnen Zeilen
zurück
            fmt=".2f", #Rundet auf 2 Nachkommastellen
            cmap="RdYlGn",#Gibt die Farbskala vor: (Rot, Gelb, Grün)
            square=True, #Macht das die ausgegebenen Zellen Quadratisch sind.
            cbar=True,#Zeigt eine Farblegende unter dem Diagram an
            annot_kws={"size": 12},
```

```
            xticklabels=[label.replace('_', '\n') for label in
Korrelationsmatrix.columns],

)
plt.title("Korrelationsmatrix der Finalen Model-Features")# Verseht die Grafik
mit einem Titel
plt.tight_layout()#sorgt für ein optimales Layout in dem es prüft, dass nichts
abgeschnitten wird.
plt.savefig("Korrelationsmatrix der Finalen Model-Featurespng.png")
plt.show()#Zeigt die Grafik (Korrelationsmatrix)


Korrelationsmatrix # Gibt eine ausgabe der Korrelationsmatrix als DataFrame
zurück.


# --- Auswahl der zu reduzierenden Features zur Vermeidung von Redundanzen ---
reduzierte_finale_features = [
    "Normierte_Wahrscheinlichkeit_Heimsieg",
"Normierte_Wahrscheinlichkeit_Unentschieden", #nomierte Wahrscheinlichkeiten
für Heimsieg und Unentschieden.
    "Buchmacher_favorisiert_Heimsieg", #Dummy-Variable, Buchmacher favorsiert
Heimteam
    "Wochentag", "Monat", "spät_Saison" #Kalenderinformation (Wochentag und
Monat)
]

# Erstellung eines neuen DataFrames, mit den reduzierten Merkmalen
Feature_Info_final =
Vollständige_Originle_Traningsdaten[reduzierte_finale_features]

# --- Neue Korrelationsmatrix mit reduzierten Features ---
Korrelationsmatrix = Feature_Info_final.corr()
# Berechnet die Korrelationsmatrix erneut, aber nun nur noch für die
ausgewählten Features.
# Dies sollte kontrolieren, ob zu starke Korrelationen korrekt entfernt
wurden.


# Visualisieren
plt.figure(figsize=(16, 6)) #gibt die Grösse der Grafik (Korrelationsmatrix)
vor
sns.heatmap(Korrelationsmatrix, #Ist die zuvor berechnete Korrelationsmatrix
            annot=True, #Gibt die Korrelationswerte der einzellnen Zeilen
zurück
            fmt=".2f", #Rundet auf 2 Nachkommastellen
            cmap="RdYlGn",#Gibt die Farbskala vor: (Rot, Gelb, Grün)
            square=True, #Macht das die ausgegebenen Zellen Quadratisch sind.
            cbar=True,#Zeigt eine Farblegende unter dem Diagram an
            annot_kws={"size": 12},
```

```
            xticklabels=[label.replace('_', '\n') for label in
Korrelationsmatrix.columns],
            yticklabels=[label.replace('_', '\n') for label in
Korrelationsmatrix.index]
)
plt.title("Korrelationsmatrix der Finalen Model-Features_v2")# Verseht die
Grafik mit einem Titel
plt.tight_layout()#sorgt für ein optimales Layout in dem es prüft, dass nichts
abgeschnitten wird.
plt.savefig("Korrelationsmatrix der Finalen Model-Features_v2.png")
plt.show()#Zeigt die Grafik (Korrelationsmatrix)


Korrelationsmatrix
# Gibt eine ausgabe der Korrelationsmatrix als DataFrame zurück.

# --- Nochmals eine auswahl der zu reduzierenden Features zur Vermeidung von
Redundanzen ---
#Dafür wurden nur Features ausgewählt die tatsächlich schon vor dem Spieltag
bekannt sind.
reduzierte_finale_features_v2 = [
    "Normierte_Wahrscheinlichkeit_Heimsieg",
"Normierte_Wahrscheinlichkeit_Unentschieden", #nomierte Wahrscheinlichkeiten
für Heimsieg und Unentschieden.
    "Buchmacher_favorisiert_Heimsieg", #Dummy-Variable, Buchmacher favorsiert
Heimteam
    "Wochentag", "Monat", "spät_Saison" #Kalenderinformation (Wochentag und
Monat)
]

Verwendtete_Features =
Vollständige_Originle_Traningsdaten[reduzierte_finale_features_v2]
# 'Verwendtete_Features' sind also die Informationen die wir dem Modell geben,
sodass es Vorhersagen treffen kann.
# In unserem Modell wären das z.B. die Wahrscheinlichkeiten der Quoten von
Bet365, die Wochentag oder der Monat.


# Nun teilen wir die Daten in Trainings und Testdaten auf.
X_traings_daten, X_test_daten, y_traings_daten, y_test_daten =
train_test_split(
    # 80 % der Daten werden verwendet, um das Modell zu trainieren
(X_traings_daten, y_trainngs_daten),
    Verwendtete_Features, Vollständige_Tatsächliches_Resultat, test_size=0.2,
stratify=Vollständige_Tatsächliches_Resultat, random_state=42)
    # die anderen 20 % werden später als Testdaten verwendet, also die Daten
die zeigen wie gut das Modell generalisierbar ist (X_test_daten,
y_test_daten).
```

```
    # Mit 'stratify=Vollständige_Tatsächliches_Resultat' wird sicher gestellt,
dass in den Tranings und Testdaten, ungefähr das gleiche Verhältnis von den
Resulatten (H,D,A) vorkommt.
    # Mit dem Parameter 'random_state=42' wird dafür gesorgt, dass die
Aufteilung wiederhergestellt werden kann.

# Schritt 1: Features normalisieren
scaler = StandardScaler()
X_traings_daten_scaliert = pd.DataFrame(scaler.fit_transform(X_traings_daten),
columns=X_traings_daten.columns) # Normalisierung der Traningsdaten
X_test_daten_scaliert = pd.DataFrame(scaler.transform(X_test_daten),
columns=X_test_daten.columns) # Normalisierung der Testdaten


# === Modelltraining ===
log_reg = LogisticRegression(max_iter=1000, multi_class='multinomial',
solver='lbfgs', C=0.01, class_weight="balanced")
log_reg.fit(X_traings_daten_scaliert, y_traings_daten)
vorhergesagte_Resultate = log_reg.predict(X_test_daten_scaliert)

# === Evaluation ===
accuracy = accuracy_score(y_test_daten, vorhergesagte_Resultate)
report = classification_report(y_test_daten, vorhergesagte_Resultate,
output_dict=True)
# Wir messen, zuerst wie gut die Vorhergesagten Reusltate mit den
tatsächlichen Resultaten übereinstimmen (Accuracy).
# Zusätzlich wird ein detaillierten Bericht mit Precision, Recall und F1-Score
pro Klasse erstellt.

# In DataFrame umwandeln
report = pd.DataFrame(report).T
# "accuracy" extrahieren

report = report.drop(index=[ "macro avg", "weighted avg"], errors="ignore")

# Nur relevante Spalten behalten und runden
report = report[["precision", "recall", "f1-score", "support"]].round(2)

# Bereite Tabelle für Anzeige vor
report.index.name = "Klasse"
report.reset_index(inplace=True)
report

fig, ax = plt.subplots(figsize=(8, 3.5))
ax.axis('off')

# Tabelle zeichnen
table = ax.table(
    cellText=report.values,
    colLabels=report.columns,
```

```
    loc='center',
    cellLoc='center'
)

table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

# Zebra-Stil für bessere Übersicht
for i in range(len(report)):
    for j in range(len(report.columns)):
        color = "#f0f0f0" if i % 2 == 0 else "white"
        table[(i+1, j)].set_facecolor(color)

# Kopfzeile hervorheben
for j in range(len(report.columns)):
    table[(0, j)].set_facecolor("#d0d0d0")
    table[(0, j)].set_text_props(weight='bold')

# Titel
plt.title("Classification Report (Saison Testdaten)")

# Accuracy separat fett darunter darstellen
plt.text(0.5, -0.25, f"accuracy: {accuracy:.2f}",
         ha="center", va="center", fontsize=12, fontweight="bold")

plt.tight_layout()
plt.savefig("Classification_Report_Testdaten.png")
plt.show()
# === Feature-Wichtigkeit ===

# Extrahiere die Koeffizienten und Feature-Namen
Koeffizienten = log_reg.coef_[0]

# Erstelle ein DataFrame für die Visualisierung
Feature_Wichtigkeit = pd.DataFrame({
    "Feature": X_traings_daten_scaliert.columns,
    "Wichtigkeit": np.abs(Koeffizienten) #macht das der absolute Betrag der
Koeffizienten gezeigt wird, um die wichtigkeit der Merkmale unabhängig von der
Richtung des Zusammenhangs (positiv oder negativ) darzustellen
}).sort_values(by="Wichtigkeit", ascending=False)


plt.figure(figsize=(12, 6))#Gibt die grösse der Grafik vor
sns.barplot(data=Feature_Wichtigkeit, x="Wichtigkeit", y="Feature") #erstellt
ein horizontales Balkendiagramm (mit der Wichtigkeit auf der X-Achse und den
Features auf der Y-Achse)
plt.title("Wichtigkeit der Features im finalen Model ")# Die Grafik wird mit
einem Titel versehen
```

```
plt.tight_layout()# Hier wird das Layout optimiert, indem nichts abgeschnitten
wird
plt.savefig("Feature_Wichtigkeit.png") #speichert die datei als PNG-Datei
plt.show()# Zeigt die Grafik (Balkendiagramm)

# === 2. Confusion Matrix ===
labels = ["H", "D", "A"] # Definiert die Labels Heimsieg, Unentschieden und
Auswärtssieg in dieser Reihenfolge.
Confusion_Matrix = confusion_matrix(y_test_daten, vorhergesagte_Resultate,
labels=labels) # Erstellt die Confusion Matrix für das Modell auf Basis der
Testdaten
Confusion_Matrix_final = pd.DataFrame(Confusion_Matrix, index=labels,
columns=labels) # Wandelt die Matrix in ein DataFrame um, um sie später als
Heatmap darzustellen zu können.

# Visualisierung der Confusion Matrix als Heatmap
plt.figure(figsize=(8, 6))#Gibt die grösse der Grafik vor
sns.heatmap(Confusion_Matrix_final, #erstellt eine Heatmap der Confusion
Matrix mit hilfe von Seaborn.
            annot=True, #zeigt die numerischen Werte der Confusion Matrix an.
            fmt=".2f", #rundet auf 2 Nachkommastellen
            cmap="RdYlGn")#Verseht die Heatmap mit einer Farbskala
plt.title("Confusion Matrix (Vorhergesagtes vs Tatsächliches Resultat)
Testdaten")# Die Grafik wird mit einem Titel versehen
plt.xlabel("Vorhergesagt") #X-Achse wird beschriftet
plt.ylabel("Tatsächlich")#Y-Achse wird beschriftet
plt.tight_layout()# Hier wird das Layout optimiert, indem nichts abgeschnitten
wird
plt.savefig('Confusion Matrix_modell_training.png')# Speichert die Grafik
plt.show()#Zeigt die finale Grafik (Heatmap)

# === Anwendung auf Saison 2024/25 ===
Saison_2425 = pd.read_csv(Bundesliga_Saison_2425)
Saison_2425["Datum"] = pd.to_datetime(Saison_2425["Date"], dayfirst=True,
errors="coerce")
# 'Saison_2425["Datum"]' konvertiert die Spalte "Date" in ein passendes
Datumsformat.
# 'dayfirst=True' sorgt dafür, dass das Datum in folgendem Format (TT/MM/JJJJ)
interpretiert wird.
# 'errors="coerce"' stellt sicher, dass ungültige Datumsangaben in NaT (Not a
Time) umgewandelt werden.

# Hier wird wieder die implizite Wahrscheinlichkeiten aus den Wettquoten von
Bet365 ausgerechnet --> Erklärung oben
Saison_2425["Wahrscheinlichkeit_Heimsieg"] = 1 / Saison_2425["B365H"]
#Heimsieg
Saison_2425["Wahrscheinlichkeit_Unetschieden"] = 1 / Saison_2425["B365D"]
#Unentschieden
Saison_2425["Wahrscheinlichkeit_Auswärtsieg"] = 1 / Saison_2425["B365A"]
#Auswärtsteam siegt
```

```
total_2223 = Saison_2425["Wahrscheinlichkeit_Heimsieg"] +
Saison_2425["Wahrscheinlichkeit_Unetschieden"] +
Saison_2425["Wahrscheinlichkeit_Auswärtsieg"]
# Hier werden die Wahrscheinlichkeiten aufsummiert, diese werden später
verwendet um die Wahrscheinlichkeiten pro Ereignis, zu normieren sodass alle
Wahrscheinlichkeiten zusammen 1 ergeben.
# Dies ist davor nicht zwingend gegeben, da die Buchmacher die Quotten nicht
'fair' gestallten, da sie auch etwas verdiennen wollen.

Saison_2425["Normierte_Wahrscheinlichkeit_Heimsieg"] =
Saison_2425["Wahrscheinlichkeit_Heimsieg"] / total_2223 #Wahrscheinlichkeit
für Heimsieg normiert
Saison_2425["Normierte_Wahrscheinlichkeit_Unentschieden"] =
Saison_2425["Wahrscheinlichkeit_Unetschieden"] / total_2223
#Wahrscheinlichkeit für ein Unentschieden normiert
Saison_2425["Favorit_Buchmacher"] = Saison_2425[["B365H", "B365D",
"B365A"]].idxmin(axis=1).str.extract(r"B365(.)")[0] # Ermittle, welches
Ergebnis laut Bet365 am wahrscheinlichsten ist.
Saison_2425["Buchmacher_favorisiert_Heimsieg"] =
(Saison_2425["Favorit_Buchmacher"] == "H").astype(int) # ein Binäres Feature
das zeigt ob der Favorit laut Buchmacher (Bet365) der Heimsieg ist.
Saison_2425["Wochentag"] = Saison_2425["Datum"].dt.weekday # Extrahiert
Wochentag aus dem Datum
Saison_2425["Monat"] = Saison_2425["Datum"].dt.month # Extrahiert Monat aus
dem Datum
Saison_2425["spät_Saison"] = Saison_2425["Monat"].apply(lambda m: 1 if m in
[3,4,5] else 0)

Feature_Info_final_2425 = Saison_2425[reduzierte_finale_features_v2].dropna()#
Auswahl der schlussendlichem Features für das finale Modell.
Vollständige_Tatsächliches_Resultat_2425 =
Saison_2425.loc[Feature_Info_final_2425.index, "FTR"] #Die Zielvariabel mit
den tatsächlichen Resultaten (H,D,A)


vorhergesagte_Resultate2425 = log_reg .predict(Feature_Info_final_2425) #
Vorhersage auf basis des trainierten Log_Reg-Modell
accuracy_2425 = accuracy_score(Vollständige_Tatsächliches_Resultat_2425,
vorhergesagte_Resultate2425) # Wir messen, zuerst wie gut die Vorhergesagten
Reuslate mit den tatsächlichen Resultaten übereinstimmen (Accuracy).
report_2425 = classification_report(Vollständige_Tatsächliches_Resultat_2425,
vorhergesagte_Resultate2425, output_dict=True) # Zusätzlich wird ein
detaillierten Bericht mit Precision, Recall und F1-Score pro Klasse erstellt.
labels = ["H", "D","A"] # # Definiert die Labels Heimsieg, Unentschieden und
Auswärtssieg in dieser Reihenfolge. Bereite so die Labels auf Einheitlichkeit
vor.

# Confusion Matrix
```

```
Confusion_Matrix = confusion_matrix(Vollständige_Tatsächliches_Resultat_2425, vorhergesagte_Resultate2425, labels=labels)# Erstellt die Confusion Matrix für das Modell auf Basis der Testdaten
Confusion_Matrix_final = pd.DataFrame(Confusion_Matrix, index=labels, columns=labels) # Wandelt die Matrix in ein DataFrame um, um sie später als Heatmap darzustellen zu können.

# Visualisierung der Confusion Matrix als Heatmap
plt.figure(figsize=(8, 6))# Gibt die grösse der Grafik vor
sns.heatmap(Confusion_Matrix_final, #erstellt eine Heatmap der Confusion Matrix mit hilfe von Seaborn.
            annot=True, #zeigt die numerischen Werte der Confusion Matrix an.
            fmt=".2f", #rundet auf 2 Nachkommastellen
            cmap="RdYlGn")#Verseht die Heatmap mit einer Farbskala
plt.title("Confusion Matrix (Vorhergesagtes vs Tatsächliches Resultat) Saison 2024/2025")# Die Grafik wird mit einem Titel versehen
plt.xlabel("Vorhergesagt") #X-Achse wird beschriftet
plt.ylabel("Tatsächlich")#Y-Achse wird beschriftet
plt.tight_layout()# Hier wird das Layout optimiert, indem nichts abgeschnitten wird
plt.savefig('Confusion_Matrix_2425.png')# Speichert die Grafik
plt.show()#Zeigt die finale Grafik (Heatmap)

correct_mask = (vorhergesagte_Resultate2425 == Vollständige_Tatsächliches_Resultat_2425.values) # Balkendiagramm das die Korrekten Vorhersagen und die falschen Vorhersagen pro Klasse zeigt
correct_labels = Vollständige_Tatsächliches_Resultat_2425[correct_mask].value_counts().reindex(labels, fill_value=0) # Zählt alle korrekten Vorhersagen pro Klasse
incorrect_labels = Vollständige_Tatsächliches_Resultat_2425[~correct_mask].value_counts().reindex(labels, fill_value=0) # Zählt alle inkorrekten Vorhersagen pro Klasse.


# Erstellt ein DataFrame für gestapelte Balken
Balkendiagramm = pd.DataFrame({
    "Korekte": correct_labels,
    "Inkorekte": incorrect_labels
}, index=labels)


# Zeichnet ein gestapeltes Balkendiagramm
Balkendiagramm.plot(kind="bar", stacked=True, figsize=(8, 5), colormap="Set2")
plt.title("Wie oft wurde welches Spielergebnis richtig erkannt")
plt.xlabel("Spielergebnis")
plt.ylabel("Anzahl Matches")
plt.xticks(rotation=0)
plt.legend(title="Vorhersage")
plt.tight_layout()
plt.savefig('Genauigkeit_2425.png')
```

```
plt.show()


finaler_report = pd.DataFrame(report_2425).T # 3. Klassifikations-Report als
Tabelle anzeigen
finaler_report = finaler_report.drop(index=["accuracy", "macro avg", "weighted
avg"], errors="ignore") # Entfernt zusammenfassende / Redunate Zeilen
finaler_report = finaler_report[["precision", "recall", "f1-score",
"support"]].round(2) # Wählt alle relevanten Metriken und runde sie auf 2
Nachkommastellen.


# Bereite Tabelle für Anzeige vor
finaler_report.index.name = "Klasse"
finaler_report.reset_index(inplace=True)
finaler_report

# --- Klassifikations-Report als Tabelle + Accuracy separat darstellen ---
accuracy_value = report_2425["accuracy"]

fig, ax = plt.subplots(figsize=(8, 3.5))
ax.axis('off')

# Tabelle zeichnen
table = ax.table(
    cellText=finaler_report.values,
    colLabels=finaler_report.columns,
    loc='center',
    cellLoc='center'
)

table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

# Zebra-Stil für bessere Übersicht
for i in range(len(finaler_report)):
    for j in range(len(finaler_report.columns)):
        color = "#f0f0f0" if i % 2 == 0 else "white"
        table[(i+1, j)].set_facecolor(color)

# Kopfzeile hervorheben
for j in range(len(finaler_report.columns)):
    table[(0, j)].set_facecolor("#d0d0d0")
    table[(0, j)].set_text_props(weight='bold')

# Titel
plt.title("Classification Report (Saison 2024/2025)")

# Accuracy separat fett darunter darstellen
```

```
plt.text(0.5, -0.25, f"Accuracy: {accuracy_value:.2f}",
         ha="center", va="center", fontsize=12, fontweight="bold")

plt.tight_layout()
plt.savefig("Classification_Report_2425.png")
plt.show()
```

(Anhang 9: Logistisches Regressionsmodell (ohne Korrektur))

```python
import pandas as pd
# Import der pandas-Bibliothek und verseht sie mit dem namen pd.
# pandas wird hier für die Datenanalyse und die Datenmanipulation verwendet.
# Sie ist eine leistungsstarke Bibliothek und basiert auf NumPy.
# Zudem bietet pandas verschiedene Datenstrukturen unteranderem DataFrames und
Series, dies ermöglochen mit tabellarischen und zeitbasierten Daten zu
arbeiten.

import numpy as np
# Import der numpy-Bibliothek und  verseht sie mit dem namen np.
# NumPy  wird hier für das wissenschaftliche Rechnen mit Python verwendet.
# Es bietet leistungsstarke Datenstrukturen wie beispielsweise Arrays, es
bietet unteradnderm auch Funktionen für mathematische Operationen, lineare
Algebra und Statistik.
# numpy ist besonders effizient bei der Verarbeitung von grossen Daten und es
bildet zudem die Grundlage für viele weitere Bibliotheken die wir verwenden
werden wie z.B. pandas, scipy oder scikit-learn.

import matplotlib.pyplot as plt
# 'matplotlib.pyplot' ist ein Modul das für erstellungen von Diagrammen und
visualisierungen verwendet wird.
# Meistens braucht man es für einfache Plots wie z.B. Liniendiagramme,
Balkendiagramme oder Streudiagramme.

import seaborn as sns
# 'seaborn' basierend auf 'matplotlib' und ist eine Bibliothek, die speziell
für statistische Visualisierungen entwickelt wurde.
# Sie stellt Standarddesigns und Funktionen bereit wie z.B. Heatmaps, Boxplots
oder Korrelationsmatrizen.

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
# scikit-learn (oder sklearn) ist eine Bibliothek die für das maschinelles
Lernen in Python verwendet wird.
# Sie stellt viele bekannte Algorithmen zurverfügung, wie beispielsweisse für
die Klassifikation, die Regression und das Clustering.
# Sie bietet zudem Werkzeuge für Modelltraining, Vorhersage, Evaluation und
Datenvorverarbeitung an.

from sklearn.preprocessing import StandardScaler

Bundesliga_Saison_2425 = "data/raw2425/D1_merged.csv"
# Definiert den Pfad zu den Rohdaten der Bundesliga-Saison 2024/2025.
# In diesen Daten enthalten sind vorallem, die Wettqoutten der anstehenden
Spiele der Saison 2024/2025.

Bundesliga_Saison_2021222324 = "data/raw/D1_merged.csv"
```

```
# Definiert den Pfad zur aktuellen Trainingsdatei, dise sind die gleichen
Rohdaten wie oben, einfach für die Bundesligasaison 2020/2021, 2021/2022,
2022/2023, 2023/2024
# Diese Datei wird im nächsten Schritt eingelesen.

Traningsdaten = pd.read_csv(Bundesliga_Saison_2021222324)
# Nun wird die CSV-Datei "data/raw/D1_merged.csv" mit pandas eingelesen und
speichert sie anschliessend als DataFrame.
# Dadurch verfügt der DataFrame 'df_train_raw'  nun über die tabellarischen
Daten aus der Datei "data/raw/D1_merged.csv".


# --- Vorbereitung der Daten für das Feature-Tuning ---
Originle_Traningsdaten = Traningsdaten.copy()
# Fertigt eine Kopie des ursprünglichen DataFrames an, damit die Originaldaten
nicht verändert werden.


# --- 1. Feature-Tunnig = die Buchmacher-Quoten von Bet365 ---
for col in ["B365H", "B365D", "B365A"]:
# Eine for-schleife durchläuft die drei Spalten "B365H", "B365D", "B365A" mit
den Quotten von dem Wettsanbeiter Bet365.
# Dabei steht 'B365H' für Heimsieg, 'B365D' für Unentschieden und 'B365A' für
Auswärtssieg.

    Originle_Traningsdaten[col] = pd.to_numeric(Originle_Traningsdaten[col],
errors="coerce")
# Diese Spalten enthalten eigentlich Strings, unteranderm die eingelesene
Zahlen, aus den CSV-Dateien.
# Mit 'pd.to_numeric()' wird sicher gestelltdass die Werte (die momentan zum
teil aus Strings bestehen) in echte numerische Datentypen umgewandelt werden.
# Dadurch werden verschiedene mathematische Berechnungen wie zum Beispiel
Division, Mittelwert berchnung ermöglicht.
# 'errors="coerce"' sorgt dafür, dass ungültige  Werte (z.B. leere Felder)
automatisch in 'NaN' (Not a Number) umgewandelt werden, dadurch werden sie von
pandas korrekt als fehlende Werte behandelt.

Originle_Traningsdaten["Wahrscheinlichkeit_Heimsieg"] = 1 /
Originle_Traningsdaten["B365H"]
Originle_Traningsdaten["Wahrscheinlichkeit_Unentschieden"] = 1 /
Originle_Traningsdaten["B365D"]
Originle_Traningsdaten["Wahrscheinlichkeit_Auswärtssieg"] = 1 /
Originle_Traningsdaten["B365A"]
# Die Buchmacherquoten "B365H", "B365D", "B365A" zeigen wie hoch die
Auszahlung für 1 Euro Einsatz ist wenn man das richtige Reultat tippt.
# Um nun die Wahrscheinlichkeiten daraus abzuleiten, wird der Kehrwert
berechnet
# Dazu wird die folgende Formel verwendet --> Wahrscheinlichkeit = 1 / Quote
```

```
Normierte_Gesammtwahrscheinlichkeit =
Originale_Traningsdaten["Wahrscheinlichkeit_Heimsieg"] +
Originale_Traningsdaten["Wahrscheinlichkeit_Unentschieden"] +
Originale_Traningsdaten["Wahrscheinlichkeit_Auswärtssieg"]
Originale_Traningsdaten["Normierte_Wahrscheinlichkeit_Heimsieg"] =
Originale_Traningsdaten["Wahrscheinlichkeit_Heimsieg"] /
Normierte_Gesammtwahrscheinlichkeit
Originale_Traningsdaten["Normierte_Wahrscheinlichkeit_Unentschieden"] =
Originale_Traningsdaten["Wahrscheinlichkeit_Unentschieden"] /
Normierte_Gesammtwahrscheinlichkeit
Originale_Traningsdaten["Normierte_Wahrscheinlichkeit_Auswärtssieg"] =
Originale_Traningsdaten["Wahrscheinlichkeit_Auswärtssieg"] /
Normierte_Gesammtwahrscheinlichkeit
# Hier werden die Wahrscheinlichkeiten normiert, also durch die Gesamtsumme
aller drei Wahrscheinlichkeiten geteilt, damit sie zusammen 1 ergeben.
# Dies ist davor nicht zwingend gegeben, da die Buchmacher die Quotten nicht
'fair' gestallten, da sie auch etwas verdiennen wollen.
# Nun haben wir eine echte Wahrscheinlichkeitsverteilung die von diesem
Problem bereinigt ist.
# Das Modell verwendet schlussendliche diese Wahrscheinlichkeiten als Input-
Features.

# --- 2. Feature-Tunnig = der Favorit des Buchmachers (Dummy-Feature) ---
Originale_Traningsdaten["Favorit_Buchmacher"] =
Originale_Traningsdaten[["B365H", "B365D",
"B365A"]].idxmin(axis=1).str.extract(r"B365(.)")[0]
# Hier wird geschaut, welche der drei Optionen (Heimsieg, Unentschieden,
Auswärtssieg) die niedrigste Quotte hat, und zwar für jede einzellne
Spielzeile.
# Dies macht man, da man so die favorisierte Option vom Buchmacher (Bet365)
zurückerhält.
# Da ja folgende Formel gilt: Wahrscheinlichkeit = 1 / Quote, das bedeutet je
tiefer die Quotte, desto höher die Wahrscheinlichkeit.
# Mit 'idxmin(axis=1)' zeigt man, den Namen der Spalte, die den kleinsten Wert
pro Zeile hat.
# Mit  folgendem Aufruf 'str.extract(r"B365(.)")' extrahiert man den
Buchstaben (H,D,A) aus dem gesamten Spaltennamen also z.B extrahiert man das H
bei 'B365H'

Originale_Traningsdaten["Buchmacher_favorisiert_Heimsieg"] =
(Originale_Traningsdaten["Favorit_Buchmacher"] == "H").astype(int)
Originale_Traningsdaten["Buchmacher_favorisiert_Unentschieden"] =
(Originale_Traningsdaten["Favorit_Buchmacher"] == "D").astype(int)
Originale_Traningsdaten["Buchmacher_favorisiert_Auswärtssieg"] =
(Originale_Traningsdaten["Favorit_Buchmacher"] == "A").astype(int)
# Hier werden drei verschiedene Variablen erstellt, diese zeigen, welche
Option (H,D,A) der Buchmacher favorsiert.
# Dies sind binär Variabeln die entweder zeigen, dass der Buchmacher diese
Option favorisiert oder eben halt nicht.
```

```
# Dies sind  nur Dummy-Variabeln, heisst ist ein vernachläsigbare Variable,
wenn man das ganze Modelle betrachtet.


# --- 3. Feature-Tuning: Kalenderinformationen ---

Originale_Traningsdaten["Datum"] =
pd.to_datetime(Originale_Traningsdaten["Date"], dayfirst=True, errors="coerce")
# Originale_Traningsdaten'["Date"]' konvertiert die Spalte "Date" in ein
passendes Datumsformat.
# 'dayfirst=True' sorgt dafür, dass das Datum in folgendem Format (TT/MM/JJJJ)
interpretiert wird.
# 'errors="coerce' stellt sicher, dass ungültige Datumsangaben in NaT (Not a
Time) umgewandelt werden.

Originale_Traningsdaten["Wochentag"] =
Originale_Traningsdaten["Datum"].dt.weekday
# Hier wird der Wochentag mit hilfe des Spieldatum Extrahiert es beginnt bei 0
= Montag, 1 = Dienstag,.... und endet bei 6 = Sonntag

Originale_Traningsdaten["Monat"] = Originale_Traningsdaten["Datum"].dt.month
# Hier wird der Monat mit hilfe des Spieldatum Extrahiert es beginnt bei 1 =
Januar, 2 = Februar,.... und endet bei 12 = Dezember

Originale_Traningsdaten["spät_Saison"] =
Originale_Traningsdaten["Monat"].apply(lambda m: 1 if m in [3,4,5] else 0)
# Mit 'Originale_Traningsdaten["spät_Saison"]'wird ein binäres Feature
erstellt, dies gibt auskunft, ob ein Spiel im März, April oder Mai stattfand.

Tatsächliches_Resultat = Originale_Traningsdaten["FTR"]
# Diese Variable zeigt das tatsächliche Spielergebniss, in dem es die Spalte
FTR (= full time Result) der Datei "data/raw/D1_merged.csv" aussliesst.
# Hier werden nur die folgenden Buchstabe (H,D,A) mit den Werten H (Heimsieg),
D (Unentschieden), A (Auswärtssieg) zurückgegeben.
# Die Anzahl Tore werden also vernachlässigt.

finale_features = [
    "Normierte_Wahrscheinlichkeit_Heimsieg",
"Normierte_Wahrscheinlichkeit_Unentschieden",
"Normierte_Wahrscheinlichkeit_Auswärtssieg", # Enthält die nomierten
Wahrscheinlichkeiten für die 3 Optionen
    "Buchmacher_favorisiert_Heimsieg", "Buchmacher_favorisiert_Unentschieden",
"Buchmacher_favorisiert_Auswärtssieg",# Dummy-Variablen für die Favoriten
Option des Buchmacher
    "Wochentag", "Monat", "spät_Saison" # Kalenderbasierte Features
]
# Hier werden die ausgewählten Merkmale (Features) aufgezählt, die für das
Modell verwendet werden sollen.
```

```
Vollständige_Originale_Traningsdaten = Originale_Traningsdaten[finale_features +
["FTR"]].dropna()
# Hier wird sichergestellt, dass nur vollständige Zeilen verwendet werden,
nicht vollständige Zeilen werden entfernt.

# --- Erstellung der Feature-Arrays und der Ziel-Arrays für das verwendete
Modell ---
Feature_Info = Vollständige_Originale_Traningsdaten[finale_features]
#'Feature_Info' enthält nun die Eingabedaten der ausgewählten Features.
Vollständige_Tatsächliches_Resultat =
Vollständige_Originale_Traningsdaten["FTR"] #
'Vollständige_Tatsächliches_Resultat' ist später unsere Zielvariabele.
Feature_Info.shape, Vollständige_Tatsächliches_Resultat.value_counts() #
'Feature_Info.shape' gibt die Anzahl der Zeilen (Anzahl Spiele) und der
Spalten (Anzahl Features)
# 'Vollständige_Tatsächliches_Resultat.value_counts()' zählt wie oft es
welches Resultat (Heimsieg, Unentscheiden, Auswärtsieg) gab.

# --- Korrelationsmatrix berechnen und visualisieren ---
Korrelationsmatrix = Feature_Info.corr()
# Diese Korrelationsmatrix soll zeigen, wie stark das zwei Variablen
miteinander zusammenhängen.
# Hier liegt der Korrelationswert zwischen -1 und +1
# +1 bedeudet eine perfekte positive Korrelation, wohingegen -1 eine perfekte
negative Korrelation bedeutet und 0 gar kein Zusammenhang zwischen den
Variablen bedeutet.
# Eine zu hohe Korrelation zwischen zwei Features ist nicht wünschenswert, da
sie die Komplexität des Modells erhöht, ohne dem Modell zusätzliche
Informationen zu geben.
# Hier wurde versucht Features die stark korrelieren, zu identifizieren und
anschliessend wurden redundante Merkmale entfernt,
# wie dies bei 'Bookie_Favor_A' z.B der Fall war, dort war das Feature genau
das gegenteil zu 'Bookie_Favor_H'.
# Dieser Prozess sollte Overfitting reduzieren und erhöht somit die
Interpretierbarkeit.


# --- Visualisierung der berechneten Korrelationsmatrix ---
plt.figure(figsize=(16, 6))   #gibt die Grösse der Grafik (Korrelationsmatrix)
vor
sns.heatmap(
            Korrelationsmatrix, #Ist die zuvor berechnete Korrelationsmatrix
            annot=True, #Gibt die Korrelationswerte der einzellnen Zeilen
zurück
            fmt=".2f", #Rundet auf 2 Nachkommastellen
            cmap="RdYlGn",#Gibt die Farbskala vor: (Rot, Gelb, Grün)
            square=True, #Macht das die ausgegebenen Zellen Quadratisch sind.
            cbar=True,#Zeigt eine Farblegende unter dem Diagram an
            annot_kws={"size": 12},
```

```
            xticklabels=[label.replace('_', '\n') for label in
Korrelationsmatrix.columns],

)
plt.title("Korrelationsmatrix der Finalen Model-Features")# Verseht die Grafik
mit einem Titel
plt.tight_layout()#sorgt für ein optimales Layout in dem es prüft, dass nichts
abgeschnitten wird.
plt.savefig("Korrelationsmatrix der Finalen Model-Featurespng.png")
plt.show()#Zeigt die Grafik (Korrelationsmatrix)


Korrelationsmatrix # Gibt eine ausgabe der Korrelationsmatrix als DataFrame
zurück.


# --- Auswahl der zu reduzierenden Features zur Vermeidung von Redundanzen ---
reduzierte_finale_features = [
    "Normierte_Wahrscheinlichkeit_Heimsieg",
"Normierte_Wahrscheinlichkeit_Unentschieden", #nomierte Wahrscheinlichkeiten
für Heimsieg und Unentschieden.
    "Buchmacher_favorisiert_Heimsieg", #Dummy-Variable, Buchmacher favorsiert
Heimteam
    "Wochentag", "Monat", "spät_Saison" #Kalenderinformation (Wochentag und
Monat)
]

# Erstellung eines neuen DataFrames, mit den reduzierten Merkmalen
Feature_Info_final =
Vollständige_Originle_Traningsdaten[reduzierte_finale_features]

# --- Neue Korrelationsmatrix mit reduzierten Features ---
Korrelationsmatrix = Feature_Info_final.corr()
# Berechnet die Korrelationsmatrix erneut, aber nun nur noch für die
ausgewählten Features.
# Dies sollte kontrolieren, ob zu starke Korrelationen korrekt entfernt
wurden.


# Visualisieren
plt.figure(figsize=(16, 6)) #gibt die Grösse der Grafik (Korrelationsmatrix)
vor
sns.heatmap(Korrelationsmatrix, #Ist die zuvor berechnete Korrelationsmatrix
            annot=True, #Gibt die Korrelationswerte der einzellnen Zeilen
zurück
            fmt=".2f", #Rundet auf 2 Nachkommastellen
            cmap="RdYlGn",#Gibt die Farbskala vor: (Rot, Gelb, Grün)
            square=True, #Macht das die ausgegebenen Zellen Quadratisch sind.
            cbar=True,#Zeigt eine Farblegende unter dem Diagram an
            annot_kws={"size": 12},
```

```
            xticklabels=[label.replace('_', '\n') for label in
Korrelationsmatrix.columns],
            yticklabels=[label.replace('_', '\n') for label in
Korrelationsmatrix.index]
)
plt.title("Korrelationsmatrix der Finalen Model-Features_v2")# Verseht die
Grafik mit einem Titel
plt.tight_layout()#sorgt für ein optimales Layout in dem es prüft, dass nichts
abgeschnitten wird.
plt.savefig("Korrelationsmatrix der Finalen Model-Features_v2.png")
plt.show()#Zeigt die Grafik (Korrelationsmatrix)


Korrelationsmatrix
# Gibt eine ausgabe der Korrelationsmatrix als DataFrame zurück.

# --- Nochmals eine auswahl der zu reduzierenden Features zur Vermeidung von
Redundanzen ---
#Dafür wurden nur Features ausgewählt die tatsächlich schon vor dem Spieltag
bekannt sind.
reduzierte_finale_features_v2 = [
    "Normierte_Wahrscheinlichkeit_Heimsieg",
"Normierte_Wahrscheinlichkeit_Unentschieden", #nomierte Wahrscheinlichkeiten
für Heimsieg und Unentschieden.
    "Buchmacher_favorisiert_Heimsieg", #Dummy-Variable, Buchmacher favorsiert
Heimteam
    "Wochentag", "Monat", "spät_Saison" #Kalenderinformation (Wochentag und
Monat)
]

Verwendtete_Features =
Vollständige_Originle_Traningsdaten[reduzierte_finale_features_v2]
# 'Verwendtete_Features' sind also die Informationen die wir dem Modell geben,
sodass es Vorhersagen treffen kann.
# In unserem Modell wären das z.B. die Wahrscheinlichkeiten der Quoten von
Bet365, die Wochentag oder der Monat.


# Nun teilen wir die Daten in Trainings und Testdaten auf.
X_traings_daten, X_test_daten, y_traings_daten, y_test_daten =
train_test_split(
    # 80 % der Daten werden verwendet, um das Modell zu trainieren
(X_traings_daten, y_trainngs_daten),
    Verwendtete_Features, Vollständige_Tatsächliches_Resultat, test_size=0.2,
stratify=Vollständige_Tatsächliches_Resultat, random_state=42)
    # die anderen 20 % werden später als Testdaten verwendet, also die Daten
die zeigen wie gut das Modell generalisierbar ist (X_test_daten,
y_test_daten).
```

```
    # Mit 'stratify=Vollständige_Tatsächliches_Resultat' wird sicher gestellt,
dass in den Tranings und Testdaten, ungefähr das gleiche Verhältnis von den
Resulatten (H,D,A) vorkommt.
    # Mit dem Parameter 'random_state=42' wird dafür gesorgt, dass die
Aufteilung wiederhergestellt werden kann.

# Schritt 1: Features normalisieren
scaler = StandardScaler()
X_traings_daten_scaliert = pd.DataFrame(scaler.fit_transform(X_traings_daten),
columns=X_traings_daten.columns) # Normalisierung der Traningsdaten
X_test_daten_scaliert = pd.DataFrame(scaler.transform(X_test_daten),
columns=X_test_daten.columns) # Normalisierung der Testdaten


# === Modelltraining ===
log_reg = LogisticRegression(max_iter=1000, multi_class='multinomial',
solver='lbfgs', C=0.01, class_weight={"H": 1, "D": 0.8, "A": 1.4})
log_reg.fit(X_traings_daten_scaliert, y_traings_daten)
vorhergesagte_Resultate = log_reg.predict(X_test_daten_scaliert)

# === Evaluation ===
accuracy = accuracy_score(y_test_daten, vorhergesagte_Resultate)
report = classification_report(y_test_daten, vorhergesagte_Resultate,
output_dict=True)
# Wir messen, zuerst wie gut die Vorhergesagten Reusltate mit den
tatsächlichen Resultaten übereinstimmen (Accuracy).
# Zusätzlich wird ein detaillierten Bericht mit Precision, Recall und F1-Score
pro Klasse erstellt.

# In DataFrame umwandeln
report = pd.DataFrame(report).T
# "accuracy" extrahieren

report = report.drop(index=[ "macro avg", "weighted avg"], errors="ignore")

# Nur relevante Spalten behalten und runden
report = report[["precision", "recall", "f1-score", "support"]].round(2)

# Bereite Tabelle für Anzeige vor
report.index.name = "Klasse"
report.reset_index(inplace=True)
report

fig, ax = plt.subplots(figsize=(8, 3.5))
ax.axis('off')

# Tabelle zeichnen
table = ax.table(
    cellText=report.values,
    colLabels=report.columns,
```

```
    loc='center',
    cellLoc='center'
)

table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

# Zebra-Stil für bessere Übersicht
for i in range(len(report)):
    for j in range(len(report.columns)):
        color = "#f0f0f0" if i % 2 == 0 else "white"
        table[(i+1, j)].set_facecolor(color)

# Kopfzeile hervorheben
for j in range(len(report.columns)):
    table[(0, j)].set_facecolor("#d0d0d0")
    table[(0, j)].set_text_props(weight='bold')

# Titel
plt.title("Classification Report (Saison Testdaten)")

# Accuracy separat fett darunter darstellen
plt.text(0.5, -0.25, f"accuracy: {accuracy:.2f}",
         ha="center", va="center", fontsize=12, fontweight="bold")

plt.tight_layout()
plt.savefig("Classification_Report_Testdaten.png")
plt.show()
# === Feature-Wichtigkeit ===

# Extrahiere die Koeffizienten und Feature-Namen
Koeffizienten = log_reg.coef_[0]

# Erstelle ein DataFrame für die Visualisierung
Feature_Wichtigkeit = pd.DataFrame({
    "Feature": X_traings_daten_scaliert.columns,
    "Wichtigkeit": np.abs(Koeffizienten) #macht das der absolute Betrag der
Koeffizienten gezeigt wird, um die wichtigkeit der Merkmale unabhängig von der
Richtung des Zusammenhangs (positiv oder negativ) darzustellen
}).sort_values(by="Wichtigkeit", ascending=False)


plt.figure(figsize=(12, 6))#Gibt die grösse der Grafik vor
sns.barplot(data=Feature_Wichtigkeit, x="Wichtigkeit", y="Feature") #erstellt
ein horizontales Balkendiagramm (mit der Wichtigkeit auf der X-Achse und den
Features auf der Y-Achse)
plt.title("Wichtigkeit der Features im finalen Model ")# Die Grafik wird mit
einem Titel versehen
```

```
plt.tight_layout()# Hier wird das Layout optimiert, indem nichts abgeschnitten
wird
plt.savefig("Feature_Wichtigkeit.png") #speichert die datei als PNG-Datei
plt.show()# Zeigt die Grafik (Balkendiagramm)

# === 2. Confusion Matrix ===
labels = ["H", "D", "A"] # Definiert die Labels Heimsieg, Unentschieden und
Auswärtssieg in dieser Reihenfolge.
Confusion_Matrix = confusion_matrix(y_test_daten, vorhergesagte_Resultate,
labels=labels) # Erstellt die Confusion Matrix für das Modell auf Basis der
Testdaten
Confusion_Matrix_final = pd.DataFrame(Confusion_Matrix, index=labels,
columns=labels) # Wandelt die Matrix in ein DataFrame um, um sie später als
Heatmap darzustellen zu können.

# Visualisierung der Confusion Matrix als Heatmap
plt.figure(figsize=(8, 6))#Gibt die grösse der Grafik vor
sns.heatmap(Confusion_Matrix_final, #erstellt eine Heatmap der Confusion
Matrix mit hilfe von Seaborn.
            annot=True, #zeigt die numerischen Werte der Confusion Matrix an.
            fmt=".2f", #rundet auf 2 Nachkommastellen
            cmap="RdYlGn")#Verseht die Heatmap mit einer Farbskala
plt.title("Confusion Matrix (Vorhergesagtes vs Tatsächliches Resultat)
Testdaten")# Die Grafik wird mit einem Titel versehen
plt.xlabel("Vorhergesagt") #X-Achse wird beschriftet
plt.ylabel("Tatsächlich")#Y-Achse wird beschriftet
plt.tight_layout()# Hier wird das Layout optimiert, indem nichts abgeschnitten
wird
plt.savefig('Confusion Matrix_modell_training.png')# Speichert die Grafik
plt.show()#Zeigt die finale Grafik (Heatmap)

# === Anwendung auf Saison 2024/25 ===
Saison_2425 = pd.read_csv(Bundesliga_Saison_2425)
Saison_2425["Datum"] = pd.to_datetime(Saison_2425["Date"], dayfirst=True,
errors="coerce")
# 'Saison_2425["Datum"]' konvertiert die Spalte "Date" in ein passendes
Datumsformat.
# 'dayfirst=True' sorgt dafür, dass das Datum in folgendem Format (TT/MM/JJJJ)
interpretiert wird.
# 'errors="coerce"' stellt sicher, dass ungültige Datumsangaben in NaT (Not a
Time) umgewandelt werden.

# Hier wird wieder die implizite Wahrscheinlichkeiten aus den Wettquoten von
Bet365 ausgerechnet --> Erklärung oben
Saison_2425["Wahrscheinlichkeit_Heimsieg"] = 1 / Saison_2425["B365H"]
#Heimsieg
Saison_2425["Wahrscheinlichkeit_Unetschieden"] = 1 / Saison_2425["B365D"]
#Unentschieden
Saison_2425["Wahrscheinlichkeit_Auswärtsieg"] = 1 / Saison_2425["B365A"]
#Auswärtsteam siegt
```

```
total_2223 = Saison_2425["Wahrscheinlichkeit_Heimsieg"] +
Saison_2425["Wahrscheinlichkeit_Unetschieden"] +
Saison_2425["Wahrscheinlichkeit_Auswärtsieg"]
# Hier werden die Wahrscheinlichkeiten aufsummiert, diese werden später
verwendet um die Wahrscheinlichkeiten pro Ereignis, zu normieren sodass alle
Wahrscheinlichkeiten zusammen 1 ergeben.
# Dies ist davor nicht zwingend gegeben, da die Buchmacher die Quotten nicht
'fair' gestallten, da sie auch etwas verdiennen wollen.

Saison_2425["Normierte_Wahrscheinlichkeit_Heimsieg"] =
Saison_2425["Wahrscheinlichkeit_Heimsieg"] / total_2223 #Wahrscheinlichkeit
für Heimsieg normiert
Saison_2425["Normierte_Wahrscheinlichkeit_Unentschieden"] =
Saison_2425["Wahrscheinlichkeit_Unetschieden"] / total_2223
#Wahrscheinlichkeit für ein Unentschieden normiert
Saison_2425["Favorit_Buchmacher"] = Saison_2425[["B365H", "B365D",
"B365A"]].idxmin(axis=1).str.extract(r"B365(.)")[0] # Ermittle, welches
Ergebnis laut Bet365 am wahrscheinlichsten ist.
Saison_2425["Buchmacher_favorisiert_Heimsieg"] =
(Saison_2425["Favorit_Buchmacher"] == "H").astype(int) # ein Binäres Feature
das zeigt ob der Favorit laut Buchmacher (Bet365) der Heimsieg ist.
Saison_2425["Wochentag"] = Saison_2425["Datum"].dt.weekday # Extrahiert
Wochentag aus dem Datum
Saison_2425["Monat"] = Saison_2425["Datum"].dt.month # Extrahiert Monat aus
dem Datum
Saison_2425["spät_Saison"] = Saison_2425["Monat"].apply(lambda m: 1 if m in
[3,4,5] else 0)

Feature_Info_final_2425 = Saison_2425[reduzierte_finale_features_v2].dropna()#
Auswahl der schlussendlichem Features für das finale Modell.
Vollständige_Tatsächliches_Resultat_2425 =
Saison_2425.loc[Feature_Info_final_2425.index, "FTR"] #Die Zielvariabel mit
den tatsächlichen Resultaten (H,D,A)


vorhergesagte_Resultate2425 = log_reg .predict(Feature_Info_final_2425) #
Vorhersage auf basis des trainierten Log_Reg-Modell
accuracy_2425 = accuracy_score(Vollständige_Tatsächliches_Resultat_2425,
vorhergesagte_Resultate2425) # Wir messen, zuerst wie gut die Vorhergesagten
Reuslтate mit den tatsächlichen Resultaten übereinstimmen (Accuracy).
report_2425 = classification_report(Vollständige_Tatsächliches_Resultat_2425,
vorhergesagte_Resultate2425, output_dict=True) # Zusätzlich wird ein
detaillierten Bericht mit Precision, Recall und F1-Score pro Klasse erstellt.
labels = ["H", "D","A"] # # Definiert die Labels Heimsieg, Unentschieden und
Auswärtssieg in dieser Reihenfolge. Bereite so die Labels auf Einheitlichkeit
vor.

# Confusion Matrix
```

```
Confusion_Matrix = confusion_matrix(Vollständige_Tatsächliches_Resultat_2425,
vorhergesagte_Resultate2425, labels=labels)# Erstellt die Confusion Matrix für
das Modell auf Basis der Testdaten
Confusion_Matrix_final = pd.DataFrame(Confusion_Matrix, index=labels,
columns=labels) # Wandelt die Matrix in ein DataFrame um, um sie später als
Heatmap darzustellen zu können.

# Visualisierung der Confusion Matrix als Heatmap
plt.figure(figsize=(8, 6))# Gibt die grösse der Grafik vor
sns.heatmap(Confusion_Matrix_final, #erstellt eine Heatmap der Confusion
Matrix mit hilfe von Seaborn.
            annot=True, #zeigt die numerischen Werte der Confusion Matrix an.
            fmt=".2f", #rundet auf 2 Nachkommastellen
            cmap="RdYlGn")#Verseht die Heatmap mit einer Farbskala
plt.title("Confusion Matrix (Vorhergesagtes vs Tatsächliches Resultat) Saison
2024/2025")# Die Grafik wird mit einem Titel versehen
plt.xlabel("Vorhergesagt") #X-Achse wird beschriftet
plt.ylabel("Tatsächlich")#Y-Achse wird beschriftet
plt.tight_layout()# Hier wird das Layout optimiert, indem nichts abgeschnitten
wird
plt.savefig('Confusion_Matrix_2425.png')# Speichert die Grafik
plt.show()#Zeigt die finale Grafik (Heatmap)

correct_mask = (vorhergesagte_Resultate2425 ==
Vollständige_Tatsächliches_Resultat_2425.values) # Balkendiagramm das die
Korrekten Vorhersagen und die falschen Vorhersagen pro Klasse zeigt
correct_labels =
Vollständige_Tatsächliches_Resultat_2425[correct_mask].value_counts().reindex(
labels, fill_value=0) # Zählt alle korrekten Vorhersagen pro Klasse
incorrect_labels =
Vollständige_Tatsächliches_Resultat_2425[~correct_mask].value_counts().reindex
(labels, fill_value=0) # Zählt alle inkorrekten Vorhersagen pro Klasse.


# Erstellt ein DataFrame für gestapelte Balken
Balkendiagramm = pd.DataFrame({
    "Korekte": correct_labels,
    "Inkorekte": incorrect_labels
}, index=labels)


# Zeichnet ein gestapeltes Balkendiagramm
Balkendiagramm.plot(kind="bar", stacked=True, figsize=(8, 5), colormap="Set2")
plt.title("Wie oft wurde welches Spielergebnis richtig erkannt")
plt.xlabel("Spielergebnis")
plt.ylabel("Anzahl Matches")
plt.xticks(rotation=0)
plt.legend(title="Vorhersage")
plt.tight_layout()
plt.savefig('Genauigkeit_2425.png')
```

```
plt.show()


finaler_report = pd.DataFrame(report_2425).T # 3. Klassifikations-Report als
Tabelle anzeigen
finaler_report = finaler_report.drop(index=["accuracy", "macro avg", "weighted
avg"], errors="ignore") # Entfernt zusammenfassende / Redunate Zeilen
finaler_report = finaler_report[["precision", "recall", "f1-score",
"support"]].round(2) # Wählt alle relevanten Metriken und runde sie auf 2
Nachkommastellen.


# Bereite Tabelle für Anzeige vor
finaler_report.index.name = "Klasse"
finaler_report.reset_index(inplace=True)
finaler_report

# --- Klassifikations-Report als Tabelle + Accuracy separat darstellen ---
accuracy_value = report_2425["accuracy"]

fig, ax = plt.subplots(figsize=(8, 3.5))
ax.axis('off')

# Tabelle zeichnen
table = ax.table(
    cellText=finaler_report.values,
    colLabels=finaler_report.columns,
    loc='center',
    cellLoc='center'
)

table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

# Zebra-Stil für bessere Übersicht
for i in range(len(finaler_report)):
    for j in range(len(finaler_report.columns)):
        color = "#f0f0f0" if i % 2 == 0 else "white"
        table[(i+1, j)].set_facecolor(color)

# Kopfzeile hervorheben
for j in range(len(finaler_report.columns)):
    table[(0, j)].set_facecolor("#d0d0d0")
    table[(0, j)].set_text_props(weight='bold')

# Titel
plt.title("Classification Report (Saison 2024/2025)")

# Accuracy separat fett darunter darstellen
```

```
plt.text(0.5, -0.25, f"Accuracy: {accuracy_value:.2f}",
         ha="center", va="center", fontsize=12, fontweight="bold")

plt.tight_layout()
plt.savefig("Classification_Report_2425.png")
plt.show()
```

(Anhang 10: Logistisches Regressionsmodell (mit Korrektur))

```python
import pandas as pd
# Import der pandas-Bibliothek und verseht sie mit dem namen pd.
# pandas wird hier für die Datenanalyse und die Datenmanipulation verwendet.
# Sie ist eine leistungsstarke Bibliothek und basiert auf NumPy.
# Zudem bietet pandas verschiedene Datenstrukturen unteranderem DataFrames und
Series, dies ermöglochen mit tabellarischen und zeitbasierten Daten zu
arbeiten.

import numpy as np
# Import der numpy-Bibliothek und  verseht sie mit dem namen np.
# NumPy  wird hier für das wissenschaftliche Rechnen mit Python verwendet.
# Es bietet leistungsstarke Datenstrukturen wie beispielsweise Arrays, es
bietet unteradnderm auch Funktionen für mathematische Operationen, lineare
Algebra und Statistik.
# numpy ist besonders effizient bei der Verarbeitung von grossen Daten und es
bildet zudem die Grundlage für viele weitere Bibliotheken die wir verwenden
werden wie z.B. pandas, scipy oder scikit-learn.

import matplotlib.pyplot as plt
# 'matplotlib.pyplot' ist ein Modul das für erstellungen von Diagrammen und
visualisierungen verwendet wird.
# Meistens braucht man es für einfache Plots wie z.B. Liniendiagramme,
Balkendiagramme oder Streudiagramme.

import seaborn as sns
# 'seaborn' basierend auf 'matplotlib' und ist eine Bibliothek, die speziell
für statistische Visualisierungen entwickelt wurde.
# Sie stellt Standarddesigns und Funktionen bereit wie z.B. Heatmaps, Boxplots
oder Korrelationsmatrizen.

from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
# scikit-learn (oder sklearn) ist eine Bibliothek die für das maschinelles
Lernen in Python verwendet wird.
# Sie stellt viele bekannte Algorithmen zurverfügung, wie beispielsweisse für
die Klassifikation, die Regression und das Clustering.
# Sie bietet zudem Werkzeuge für Modelltraining, Vorhersage, Evaluation und
Datenvorverarbeitung an.


Bundesliga_Saison_2425 = "data/raw2425/D1_merged.csv"
# Definiert den Pfad zu den Rohdaten der Bundesliga-Saison 2024/2025.
# In diesen Daten enthalten sind vorallem, die Wettqoutten der anstehenden
Spiele der Saison 2024/2025.

Bundesliga_Saison_2021222324 = "data/raw/D1_merged.csv"
```

```
# Definiert den Pfad zur aktuellen Trainingsdatei, dise sind die gleichen
Rohdaten wie oben, einfach für die Bundesligasaison 2020/2021, 2021/2022,
2022/2023, 2023/2024
# Diese Datei wird im nächsten Schritt eingelesen.

Traningsdaten = pd.read_csv(Bundesliga_Saison_2021222324)
# Nun wird die CSV-Datei "data/raw/D1_merged.csv" mit pandas eingelesen und
speichert sie anschliessend als DataFrame.
# Dadurch verfügt der DataFrame 'df_train_raw'  nun über die tabellarischen
Daten aus der Datei "data/raw/D1_merged.csv".


# --- Vorbereitung der Daten für das Feature-Tuning ---
Originle_Traningsdaten = Traningsdaten.copy()
# Fertigt eine Kopie des ursprünglichen DataFrames an, damit die Originaldaten
nicht verändert werden.


# --- 1. Feature-Tunnig = die Buchmacher-Quoten von Bet365 ---
for col in ["B365H", "B365D", "B365A"]:
# Eine for-schleife durchläuft die drei Spalten "B365H", "B365D", "B365A" mit
den Quotten von dem Wettsanbeiter Bet365.
# Dabei steht 'B365H' für Heimsieg, 'B365D' für Unentschieden und 'B365A' für
Auswärtssieg.

     Originle_Traningsdaten[col] = pd.to_numeric(Originle_Traningsdaten[col],
errors="coerce")
# Diese Spalten enthalten eigentlich Strings, unteranderm die eingelesene
Zahlen, aus den CSV-Dateien.
# Mit 'pd.to_numeric()' wird sicher gestelltdass die Werte (die momentan zum
teil aus Strings bestehen) in echte numerische Datentypen umgewandelt werden.
# Dadurch werden verschiedene mathematische Berechnungen wie zum Beispiel
Division, Mittelwert berchnung ermöglicht.
# 'errors="coerce"' sorgt dafür, dass ungültige  Werte (z.B. leere Felder)
automatisch in 'NaN' (Not a Number) umgewandelt werden, dadurch werden sie von
pandas korrekt als fehlende Werte behandelt.

Originle_Traningsdaten["Wahrscheinlichkeit_Heimsieg"] = 1 /
Originle_Traningsdaten["B365H"]
Originle_Traningsdaten["Wahrscheinlichkeit_Unentschieden"] = 1 /
Originle_Traningsdaten["B365D"]
Originle_Traningsdaten["Wahrscheinlichkeit_Auswärtssieg"] = 1 /
Originle_Traningsdaten["B365A"]
# Die Buchmacherquoten "B365H", "B365D", "B365A" zeigen wie hoch die
Auszahlung für 1 Euro Einsatz ist wenn man das richtige Reultat tippt.
# Um nun die Wahrscheinlichkeiten daraus abzuleiten, wird der Kehrwert
berechnet
# Dazu wird die folgende Formel verwendet --> Wahrscheinlichkeit = 1 / Quote
```

```
Normierte_Gesammtwahrscheinlichkeit =
Originale_Traningsdaten["Wahrscheinlichkeit_Heimsieg"] +
Originale_Traningsdaten["Wahrscheinlichkeit_Unentschieden"] +
Originale_Traningsdaten["Wahrscheinlichkeit_Auswärtssieg"]
Originale_Traningsdaten["Normierte_Wahrscheinlichkeit_Heimsieg"] =
Originale_Traningsdaten["Wahrscheinlichkeit_Heimsieg"] /
Normierte_Gesammtwahrscheinlichkeit
Originale_Traningsdaten["Normierte_Wahrscheinlichkeit_Unentschieden"] =
Originale_Traningsdaten["Wahrscheinlichkeit_Unentschieden"] /
Normierte_Gesammtwahrscheinlichkeit
Originale_Traningsdaten["Normierte_Wahrscheinlichkeit_Auswärtssieg"] =
Originale_Traningsdaten["Wahrscheinlichkeit_Auswärtssieg"] /
Normierte_Gesammtwahrscheinlichkeit
# Hier werden die Wahrscheinlichkeiten normiert, also durch die Gesamtsumme
aller drei Wahrscheinlichkeiten geteilt, damit sie zusammen 1 ergeben.
# Dies ist davor nicht zwingend gegeben, da die Buchmacher die Quotten nicht
'fair' gestallten, da sie auch etwas verdiennen wollen.
# Nun haben wir eine echte Wahrscheinlichkeitsverteilung die von diesem
Problem bereinigt ist.
# Das Modell verwendet schlussendliche diese Wahrscheinlichkeiten als Input-
Features.

# --- 2. Feature-Tunnig = der Favorit des Buchmachers (Dummy-Feature) ---
Originale_Traningsdaten["Favorit_Buchmacher"] =
Originale_Traningsdaten[["B365H", "B365D",
"B365A"]].idxmin(axis=1).str.extract(r"B365(.)")[0]
# Hier wird geschaut, welche der drei Optionen (Heimsieg, Unentschieden,
Auswärtssieg) die niedrigste Quotte hat, und zwar für jede einzellne
Spielzeile.
# Dies macht man, da man so die favorisierte Option vom Buchmacher (Bet365)
zurückerhält.
# Da ja folgende Formel gilt: Wahrscheinlichkeit = 1 / Quote, das bedeutet je
tiefer die Quotte, desto höher die Wahrscheinlichkeit.
# Mit 'idxmin(axis=1)' zeigt man, den Namen der Spalte, die den kleinsten Wert
pro Zeile hat.
# Mit  folgendem Aufruf 'str.extract(r"B365(.)")' extrahiert man den
Buchstaben (H,D,A) aus dem gesamten Spaltennamen also z.B extrahiert man das H
bei 'B365H'

Originale_Traningsdaten["Buchmacher_favorisiert_Heimsieg"] =
(Originale_Traningsdaten["Favorit_Buchmacher"] == "H").astype(int)
Originale_Traningsdaten["Buchmacher_favorisiert_Unentschieden"] =
(Originale_Traningsdaten["Favorit_Buchmacher"] == "D").astype(int)
Originale_Traningsdaten["Buchmacher_favorisiert_Auswärtssieg"] =
(Originale_Traningsdaten["Favorit_Buchmacher"] == "A").astype(int)
# Hier werden drei verschiedene Variablen erstellt, diese zeigen, welche
Option (H,D,A) der Buchmacher favorsiert.
# Dies sind binär Variabeln die entweder zeigen, dass der Buchmacher diese
Option favorisiert oder eben halt nicht.
```

```
# Dies sind  nur Dummy-Variabeln, heisst ist ein vernachläsigbare Variable,
wenn man das ganze Modelle betrachtet.


# --- 3. Feature-Tuning: Kalenderinformationen ---

Originale_Traningsdaten["Datum"] =
pd.to_datetime(Originale_Traningsdaten["Date"], dayfirst=True, errors="coerce")
# Originale_Traningsdaten'["Date"]' konvertiert die Spalte "Date" in ein
passendes Datumsformat.
# 'dayfirst=True' sorgt dafür, dass das Datum in folgendem Format (TT/MM/JJJJ)
interpretiert wird.
# 'errors="coerce' stellt sicher, dass ungültige Datumsangaben in NaT (Not a
Time) umgewandelt werden.

Originale_Traningsdaten["Wochentag"] =
Originale_Traningsdaten["Datum"].dt.weekday
# Hier wird der Wochentag mit hilfe des Spieldatum Extrahiert es beginnt bei 0
= Montag, 1 = Dienstag,.... und endet bei 6 = Sonntag

Originale_Traningsdaten["Monat"] = Originale_Traningsdaten["Datum"].dt.month
# Hier wird der Monat mit hilfe des Spieldatum Extrahiert es beginnt bei 1 =
Januar, 2 = Februar,.... und endet bei 12 = Dezember

Originale_Traningsdaten["spät_Saison"] =
Originale_Traningsdaten["Monat"].apply(lambda m: 1 if m in [3,4,5] else 0)
# Mit 'Originale_Traningsdaten["spät_Saison"]'wird ein binäres Feature
erstellt, dies gibt auskunft, ob ein Spiel im März, April oder Mai stattfand.

Tatsächliches_Resultat = Originale_Traningsdaten["FTR"]
# Diese Variable zeigt das tatsächliche Spielergebniss, in dem es die Spalte
FTR (= full time Result) der Datei "data/raw/D1_merged.csv" aussliesst.
# Hier werden nur die folgenden Buchstabe (H,D,A) mit den Werten H (Heimsieg),
D (Unentschieden), A (Auswärtssieg) zurückgegeben.
# Die Anzahl Tore werden also vernachlässigt.

finale_features = [
    "Normierte_Wahrscheinlichkeit_Heimsieg",
"Normierte_Wahrscheinlichkeit_Unentschieden",
"Normierte_Wahrscheinlichkeit_Auswärtssieg", # Enthält die nomierten
Wahrscheinlichkeiten für die 3 Optionen
    "Buchmacher_favorisiert_Heimsieg", "Buchmacher_favorisiert_Unentschieden",
"Buchmacher_favorisiert_Auswärtssieg",# Dummy-Variablen für die Favoriten
Option des Buchmacher
    "Wochentag", "Monat", "spät_Saison" # Kalenderbasierte Features
]
# Hier werden die ausgewählten Merkmale (Features) aufgezählt, die für das
Modell verwendet werden sollen.
```

```
Vollständige_Originale_Traningsdaten = Originale_Traningsdaten[finale_features +
["FTR"]].dropna()
# Hier wird sichergestellt, dass nur vollständige Zeilen verwendet werden,
nicht vollständige Zeilen werden entfernt.

# --- Erstellung der Feature-Arrays und der Ziel-Arrays für das verwendete
Modell ---
Feature_Info = Vollständige_Originale_Traningsdaten[finale_features]
#'Feature_Info' enthält nun die Eingabedaten der ausgewählten Features.
Vollständige_Tatsächliches_Resultat =
Vollständige_Originale_Traningsdaten["FTR"] #
'Vollständige_Tatsächliches_Resultat' ist später unsere Zielvariabele.
Feature_Info.shape, Vollständige_Tatsächliches_Resultat.value_counts() #
'Feature_Info.shape' gibt die Anzahl der Zeilen (Anzahl Spiele) und der
Spalten (Anzahl Features)
# 'Vollständige_Tatsächliches_Resultat.value_counts()' zählt wie oft es
welches Resultat (Heimsieg, Unentscheiden, Auswärtsieg) gab.

# --- Korrelationsmatrix berechnen und visualisieren ---
Korrelationsmatrix = Feature_Info.corr()
# Diese Korrelationsmatrix soll zeigen, wie stark das zwei Variablen
miteinander zusammenhängen.
# Hier liegt der Korrelationswert zwischen -1 und +1
# +1 bedeudet eine perfekte positive Korrelation, wohingegen -1 eine perfekte
negative Korrelation bedeutet und 0 gar kein Zusammenhang zwischen den
Variablen bedeutet.
# Eine zu hohe Korrelation zwischen zwei Features ist nicht wünschenswert, da
sie die Komplexität des Modells erhöht, ohne dem Modell zusätzliche
Informationen zu geben.
# Hier wurde versucht Features die stark korrelieren, zu identifizieren und
anschliessend wurden redundante Merkmale entfernt,
# wie dies bei 'Bookie_Favor_A' z.B der Fall war, dort war das Feature genau
das gegenteil zu 'Bookie_Favor_H'.
# Dieser Prozess sollte Overfitting reduzieren und erhöht somit die
Interpretierbarkeit.


# --- Visualisierung der berechneten Korrelationsmatrix ---
plt.figure(figsize=(16, 6))   #gibt die Grösse der Grafik (Korrelationsmatrix)
vor
sns.heatmap(
            Korrelationsmatrix, #Ist die zuvor berechnete Korrelationsmatrix
            annot=True, #Gibt die Korrelationswerte der einzellnen Zeilen
zurück
            fmt=".2f", #Rundet auf 2 Nachkommastellen
            cmap="RdYlGn",#Gibt die Farbskala vor: (Rot, Gelb, Grün)
            square=True, #Macht das die ausgegebenen Zellen Quadratisch sind.
            cbar=True,#Zeigt eine Farblegende unter dem Diagram an
            annot_kws={"size": 12},
```

```
            xticklabels=[label.replace('_', '\n') for label in
Korrelationsmatrix.columns],

)
plt.title("Korrelationsmatrix der Finalen Model-Features")# Verseht die Grafik
mit einem Titel
plt.tight_layout()#sorgt für ein optimales Layout in dem es prüft, dass nichts
abgeschnitten wird.
plt.savefig("Korrelationsmatrix der Finalen Model-Featurespng.png")
plt.show()#Zeigt die Grafik (Korrelationsmatrix)


Korrelationsmatrix # Gibt eine ausgabe der Korrelationsmatrix als DataFrame
zurück.


# --- Auswahl der zu reduzierenden Features zur Vermeidung von Redundanzen ---
reduzierte_finale_features = [
    "Normierte_Wahrscheinlichkeit_Heimsieg",
"Normierte_Wahrscheinlichkeit_Unentschieden", #nomierte Wahrscheinlichkeiten
für Heimsieg und Unentschieden.
    "Buchmacher_favorisiert_Heimsieg", #Dummy-Variable, Buchmacher favorsiert
Heimteam
    "Wochentag", "Monat", "spät_Saison" #Kalenderinformation (Wochentag und
Monat)
]

# Erstellung eines neuen DataFrames, mit den reduzierten Merkmalen
Feature_Info_final =
Vollständige_Originle_Traningsdaten[reduzierte_finale_features]

# --- Neue Korrelationsmatrix mit reduzierten Features ---
Korrelationsmatrix = Feature_Info_final.corr()
# Berechnet die Korrelationsmatrix erneut, aber nun nur noch für die
ausgewählten Features.
# Dies sollte kontrolieren, ob zu starke Korrelationen korrekt entfernt
wurden.


# Visualisieren
plt.figure(figsize=(16, 6)) #gibt die Grösse der Grafik (Korrelationsmatrix)
vor
sns.heatmap(Korrelationsmatrix, #Ist die zuvor berechnete Korrelationsmatrix
            annot=True, #Gibt die Korrelationswerte der einzellnen Zeilen
zurück
            fmt=".2f", #Rundet auf 2 Nachkommastellen
            cmap="RdYlGn",#Gibt die Farbskala vor: (Rot, Gelb, Grün)
            square=True, #Macht das die ausgegebenen Zellen Quadratisch sind.
            cbar=True,#Zeigt eine Farblegende unter dem Diagram an
            annot_kws={"size": 12},
```

```
            xticklabels=[label.replace('_', '\n') for label in
Korrelationsmatrix.columns],
            yticklabels=[label.replace('_', '\n') for label in
Korrelationsmatrix.index]
)
plt.title("Korrelationsmatrix der Finalen Model-Features_v2")# Verseht die
Grafik mit einem Titel
plt.tight_layout()#sorgt für ein optimales Layout in dem es prüft, dass nichts
abgeschnitten wird.
plt.savefig("Korrelationsmatrix der Finalen Model-Features_v2.png")
plt.show()#Zeigt die Grafik (Korrelationsmatrix)


Korrelationsmatrix
# Gibt eine ausgabe der Korrelationsmatrix als DataFrame zurück.

# --- Nochmals eine auswahl der zu reduzierenden Features zur Vermeidung von
Redundanzen ---
#Dafür wurden nur Features ausgewählt die tatsächlich schon vor dem Spieltag
bekannt sind.
reduzierte_finale_features_v2 = [
    "Normierte_Wahrscheinlichkeit_Heimsieg",
"Normierte_Wahrscheinlichkeit_Unentschieden", #nomierte Wahrscheinlichkeiten
für Heimsieg und Unentschieden.
    "Buchmacher_favorisiert_Heimsieg", #Dummy-Variable, Buchmacher favorsiert
Heimteam
    "Wochentag", "Monat", "spät_Saison" #Kalenderinformation (Wochentag und
Monat)
]

Verwendtete_Features =
Vollständige_Originle_Traningsdaten[reduzierte_finale_features_v2]
# 'Verwendtete_Features' sind also die Informationen die wir dem Modell geben,
sodass es Vorhersagen treffen kann.
# In unserem Modell wären das z.B. die Wahrscheinlichkeiten der Quoten von
Bet365, die Wochentag oder der Monat.

# Nun teilen wir die Daten in Trainings und Testdaten auf.
X_traings_daten, X_test_daten, y_traings_daten, y_test_daten =
train_test_split(
    # 80 % der Daten werden verwendet, um das Modell zu trainieren
(X_traings_daten, y_trainngs_daten),
    Verwendtete_Features, Vollständige_Tatsächliches_Resultat, test_size=0.2,
stratify=Vollständige_Tatsächliches_Resultat, random_state=42)
    # die anderen 20 % werden später als Testdaten verwendet, also die Daten
die zeigen wie gut das Modell generalisierbar ist (X_test_daten,
y_test_daten).
    # Mit 'stratify=y_reduced' wird sicher gestellt, dass in den Tranings und
Testdaten, ungefähr das gleiche Verhältnis von den Resulatten (H,D,A)
vorkommt.
```

```
    # Mit dem Parameter 'random_state=42' wird dafür gesorgt, dass die
Aufteilung wiederhergestellt werden kann.

Model_GradientBoosting = GradientBoostingClassifier(
    n_estimators=1000,      # Anzahl der Bäume
    learning_rate=0.001,     # Schrittweite beim Lernen (kleiner = stabiler,
grösser = schneller)
    max_depth=5,           # Tiefe der einzelnen Bäume
    random_state=42
)
# Training mit den Trainingsdaten
Model_GradientBoosting.fit(X_traings_daten, y_traings_daten)

# Vorhersage mit den Testdaten
vorhergesagte_Resultate = Model_GradientBoosting.predict(X_test_daten)

# === Evaluation ===
accuracy = accuracy_score(y_test_daten, vorhergesagte_Resultate)
report = classification_report(y_test_daten, vorhergesagte_Resultate,
output_dict=True)
# Wir messen, zuerst wie gut die Vorhergesagten Reusltate mit den
tatsächlichen Resultaten übereinstimmen (Accuracy).
# Zusätzlich wird ein detaillierten Bericht mit Precision, Recall und F1-Score
pro Klasse erstellt.

# In DataFrame umwandeln
report = pd.DataFrame(report).T
# "accuracy" extrahieren

report = report.drop(index=[ "macro avg", "weighted avg"], errors="ignore")

# Nur relevante Spalten behalten und runden
report = report[["precision", "recall", "f1-score", "support"]].round(2)

# Bereite Tabelle für Anzeige vor
report.index.name = "Klasse"
report.reset_index(inplace=True)
report

fig, ax = plt.subplots(figsize=(8, 3.5))
ax.axis('off')

# Tabelle zeichnen
table = ax.table(
    cellText=report.values,
    colLabels=report.columns,
    loc='center',
    cellLoc='center'
)
```

```
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

# Zebra-Stil für bessere Übersicht
for i in range(len(report)):
    for j in range(len(report.columns)):
        color = "#f0f0f0" if i % 2 == 0 else "white"
        table[(i+1, j)].set_facecolor(color)

# Kopfzeile hervorheben
for j in range(len(report.columns)):
    table[(0, j)].set_facecolor("#d0d0d0")
    table[(0, j)].set_text_props(weight='bold')

# Titel
plt.title("Classification Report (Saison Testdaten)")

# Accuracy separat fett darunter darstellen
plt.text(0.5, -0.25, f"accuracy: {accuracy:.2f}",
         ha="center", va="center", fontsize=12, fontweight="bold")

plt.tight_layout()
plt.savefig("Classification_Report_Testdaten.png")
plt.show()

# === 2. Confusion Matrix ===
labels = ["H", "D", "A"] # Definiert die Labels Heimsieg, Unentschieden und
Auswärtssieg in dieser Reihenfolge.
Confusion_Matrix = confusion_matrix(y_test_daten, vorhergesagte_Resultate,
labels=labels) # Erstellt die Confusion Matrix für das Modell auf Basis der
Testdaten
Confusion_Matrix_final = pd.DataFrame(Confusion_Matrix, index=labels,
columns=labels) # Wandelt die Matrix in ein DataFrame um, um sie später als
Heatmap darzustellen zu können.

# Visualisierung der Confusion Matrix als Heatmap
plt.figure(figsize=(8, 6))#Gibt die grösse der Grafik vor
sns.heatmap(Confusion_Matrix_final, #erstellt eine Heatmap der Confusion
Matrix mit hilfe von Seaborn.
            annot=True, #zeigt die numerischen Werte der Confusion Matrix an.
            fmt=".2f", #rundet auf 2 Nachkommastellen
            cmap="RdYlGn")#Verseht die Heatmap mit einer Farbskala
plt.title("Confusion Matrix (Vorhergesagtes vs Tatsächliches Resultat)
Testdaten")# Die Grafik wird mit einem Titel versehen
plt.xlabel("Vorhergesagt") #X-Achse wird beschriftet
plt.ylabel("Tatsächlich")#Y-Achse wird beschriftet
plt.tight_layout()# Hier wird das Layout optimiert, indem nichts abgeschnitten
wird
plt.savefig('Confusion Matrix_modell_training.png')# Speichert die Grafik
```

```
plt.show()#Zeigt die finale Grafik (Heatmap)

# === Anwendung auf Saison 2024/25 ===
Saison_2425 = pd.read_csv(Bundesliga_Saison_2425)
Saison_2425["Datum"] = pd.to_datetime(Saison_2425["Date"], dayfirst=True,
errors="coerce")
# 'Saison_2425["Datum"]' konvertiert die Spalte "Date" in ein passendes
Datumsformat.
# 'dayfirst=True' sorgt dafür, dass das Datum in folgendem Format (TT/MM/JJJJ)
interpretiert wird.
# 'errors="coerce"' stellt sicher, dass ungültige Datumsangaben in NaT (Not a
Time) umgewandelt werden.

# Hier wird wieder die implizite Wahrscheinlichkeiten aus den Wettquoten von
Bet365 ausgerechnet --> Erklärung oben
Saison_2425["Wahrscheinlichkeit_Heimsieg"] = 1 / Saison_2425["B365H"]
#Heimsieg
Saison_2425["Wahrscheinlichkeit_Unetschieden"] = 1 / Saison_2425["B365D"]
#Unentschieden
Saison_2425["Wahrscheinlichkeit_Auswärtsieg"] = 1 / Saison_2425["B365A"]
#Auswärtsteam siegt
total_2223 = Saison_2425["Wahrscheinlichkeit_Heimsieg"] +
Saison_2425["Wahrscheinlichkeit_Unetschieden"] +
Saison_2425["Wahrscheinlichkeit_Auswärtsieg"]
# Hier werden die Wahrscheinlichkeiten aufsummiert, diese werden später
verwendet um die Wahrscheinlichkeiten pro Ereignis, zu normieren sodass alle
Wahrscheinlichkeiten zusammen 1 ergeben.
# Dies ist davor nicht zwingend gegeben, da die Buchmacher die Quotten nicht
'fair' gestallten, da sie auch etwas verdiennen wollen.

Saison_2425["Normierte_Wahrscheinlichkeit_Heimsieg"] =
Saison_2425["Wahrscheinlichkeit_Heimsieg"] / total_2223 #Wahrscheinlichkeit
für Heimsieg normiert
Saison_2425["Normierte_Wahrscheinlichkeit_Unentschieden"] =
Saison_2425["Wahrscheinlichkeit_Unetschieden"] / total_2223
#Wahrscheinlichkeit für ein Unentschieden normiert


Saison_2425["Favorit_Buchmacher"] = Saison_2425[["B365H", "B365D",
"B365A"]].idxmin(axis=1).str.extract(r"B365(.)")[0] # Ermittle, welches
Ergebnis laut Bet365 am wahrscheinlichsten ist.
Saison_2425["Buchmacher_favorisiert_Heimsieg"] =
(Saison_2425["Favorit_Buchmacher"] == "H").astype(int) # ein Binäres Feature
das zeigt ob der Favorit laut Buchmacher (Bet365) der Heimsieg ist.
Saison_2425["Wochentag"] = Saison_2425["Datum"].dt.weekday # Extrahiert
Wochentag aus dem Datum
Saison_2425["Monat"] = Saison_2425["Datum"].dt.month # Extrahiert Monat aus
dem Datum
Saison_2425["spät_Saison"] = Saison_2425["Monat"].apply(lambda m: 1 if m in
[3,4,5] else 0)
```

```
Feature_Info_final_2425 = Saison_2425[reduzierte_finale_features_v2].dropna()#
Auswahl der schlussendlichem Features für das finale Modell.
Vollständige_Tatsächliches_Resultat_2425 =
Saison_2425.loc[Feature_Info_final_2425.index, "FTR"] #Die Zielvariabel mit
den tatsächlichen Resultaten (H,D,A)

vorhergesagte_Resultate2425 = Model_GradientBoosting
.predict(Feature_Info_final_2425) # Vorhersage auf basis des trainierten
Random-Forest-Modell
accuracy_2425 = accuracy_score(Vollständige_Tatsächliches_Resultat_2425,
vorhergesagte_Resultate2425) # Wir messen, zuerst wie gut die Vorhergesagten
Reusltate mit den tatsächlichen Resultaten übereinstimmen (Accuracy).
report_2425 = classification_report(Vollständige_Tatsächliches_Resultat_2425,
vorhergesagte_Resultate2425, output_dict=True) # Zusätzlich wird ein
detaillierten Bericht mit Precision, Recall und F1-Score pro Klasse erstellt.
labels = ["H", "D","A"] # # Definiert die Labels Heimsieg, Unentschieden und
Auswärtssieg in dieser Reihenfolge. Bereite so die Labels auf Einheitlichkeit
vor.

# Confusion Matrix
Confusion_Matrix = confusion_matrix(Vollständige_Tatsächliches_Resultat_2425,
vorhergesagte_Resultate2425, labels=labels)# Erstellt die Confusion Matrix für
das Modell auf Basis der Testdaten
Confusion_Matrix_final = pd.DataFrame(Confusion_Matrix, index=labels,
columns=labels) # Wandelt die Matrix in ein DataFrame um, um sie später als
Heatmap darzustellen zu können.

# Visualisierung der Confusion Matrix als Heatmap
plt.figure(figsize=(8, 6))# Gibt die grösse der Grafik vor
sns.heatmap(Confusion_Matrix_final, #erstellt eine Heatmap der Confusion
Matrix mit hilfe von Seaborn.
            annot=True, #zeigt die numerischen Werte der Confusion Matrix an.
            fmt=".2f", #rundet auf 2 Nachkommastellen
            cmap="RdYlGn")#Verseht die Heatmap mit einer Farbskala
plt.title("Confusion Matrix (Vorhergesagtes vs Tatsächliches Resultat) Saison
2024/2025")# Die Grafik wird mit einem Titel versehen
plt.xlabel("Vorhergesagt") #X-Achse wird beschriftet
plt.ylabel("Tatsächlich")#Y-Achse wird beschriftet
plt.tight_layout()# Hier wird das Layout optimiert, indem nichts abgeschnitten
wird
plt.savefig('Confusion_Matrix_2425.png')# Speichert die Grafik
plt.show()#Zeigt die finale Grafik (Heatmap)

correct_mask = (vorhergesagte_Resultate2425 ==
Vollständige_Tatsächliches_Resultat_2425.values) # Balkendiagramm das die
Korrekten Vorhersagen und die falschen Vorhersagen pro Klasse zeigt
correct_labels =
Vollständige_Tatsächliches_Resultat_2425[correct_mask].value_counts().reindex(
labels, fill_value=0) # Zählt alle korrekten Vorhersagen pro Klasse
```

```
incorrect_labels =
Vollständige_Tatsächliches_Resultat_2425[~correct_mask].value_counts().reindex
(labels, fill_value=0) # Zählt alle inkorrekten Vorhersagen pro Klasse.


# Erstellt ein DataFrame für gestapelte Balken
Balkendiagramm = pd.DataFrame({
    "Korekte": correct_labels,
    "Inkorekte": incorrect_labels
}, index=labels)


# Zeichnet ein gestapeltes Balkendiagramm
Balkendiagramm.plot(kind="bar", stacked=True, figsize=(8, 5), colormap="Set2")
plt.title("Wie oft wurde welches Spielergebnis richtig erkannt")
plt.xlabel("Spielergebnis")
plt.ylabel("Anzahl Matches")
plt.xticks(rotation=0)
plt.legend(title="Vorhersage")
plt.tight_layout()
plt.savefig('Genauigkeit_2425.png')
plt.show()


finaler_report = pd.DataFrame(report_2425).T # 3. Klassifikations-Report als
Tabelle anzeigen
finaler_report = finaler_report.drop(index=["accuracy", "macro avg", "weighted
avg"], errors="ignore") # Entfernt zusammenfassende / Redunate Zeilen
finaler_report = finaler_report[["precision", "recall", "f1-score",
"support"]].round(2) # Wählt alle relevanten Metriken und runde sie auf 2
Nachkommastellen.


# Bereite Tabelle für Anzeige vor
finaler_report.index.name = "Klasse"
finaler_report.reset_index(inplace=True)
finaler_report

# --- Klassifikations-Report als Tabelle + Accuracy separat darstellen ---
accuracy_value = report_2425["accuracy"]

fig, ax = plt.subplots(figsize=(8, 3.5))
ax.axis('off')

# Tabelle zeichnen
table = ax.table(
    cellText=finaler_report.values,
    colLabels=finaler_report.columns,
    loc='center',
    cellLoc='center'
```

```
)

table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

# Zebra-Stil für bessere Übersicht
for i in range(len(finaler_report)):
    for j in range(len(finaler_report.columns)):
        color = "#f0f0f0" if i % 2 == 0 else "white"
        table[(i+1, j)].set_facecolor(color)

# Kopfzeile hervorheben
for j in range(len(finaler_report.columns)):
    table[(0, j)].set_facecolor("#d0d0d0")
    table[(0, j)].set_text_props(weight='bold')

# Titel
plt.title("Classification Report (Saison 2024/2025)")

# Accuracy separat fett darunter darstellen
plt.text(0.5, -0.25, f"Accuracy: {accuracy_value:.2f}",
         ha="center", va="center", fontsize=12, fontweight="bold")

plt.tight_layout()
plt.savefig("Classification_Report_2425.png")
plt.show()
```

(Anhang 11: Gradient-Boosting-Modell (ohne EM-Feature))

```python
import pandas as pd
# Import der pandas-Bibliothek und verseht sie mit dem namen pd.
# pandas wird hier für die Datenanalyse und die Datenmanipulation verwendet.
# Sie ist eine leistungsstarke Bibliothek und basiert auf NumPy.
# Zudem bietet pandas verschiedene Datenstrukturen unteranderem DataFrames und
Series, dies ermöglochen mit tabellarischen und zeitbasierten Daten zu
arbeiten.

import numpy as np
# Import der numpy-Bibliothek und  verseht sie mit dem namen np.
# NumPy  wird hier für das wissenschaftliche Rechnen mit Python verwendet.
# Es bietet leistungsstarke Datenstrukturen wie beispielsweise Arrays, es
bietet unteradnderm auch Funktionen für mathematische Operationen, lineare
Algebra und Statistik.
# numpy ist besonders effizient bei der Verarbeitung von grossen Daten und es
bildet zudem die Grundlage für viele weitere Bibliotheken die wir verwenden
werden wie z.B. pandas, scipy oder scikit-learn.

import matplotlib.pyplot as plt
# 'matplotlib.pyplot' ist ein Modul das für erstellungen von Diagrammen und
visualisierungen verwendet wird.
# Meistens braucht man es für einfache Plots wie z.B. Liniendiagramme,
Balkendiagramme oder Streudiagramme.

import seaborn as sns
# 'seaborn' basierend auf 'matplotlib' und ist eine Bibliothek, die speziell
für statistische Visualisierungen entwickelt wurde.
# Sie stellt Standarddesigns und Funktionen bereit wie z.B. Heatmaps, Boxplots
oder Korrelationsmatrizen.

from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
# scikit-learn (oder sklearn) ist eine Bibliothek die für das maschinelles
Lernen in Python verwendet wird.
# Sie stellt viele bekannte Algorithmen zurverfügung, wie beispielsweisse für
die Klassifikation, die Regression und das Clustering.
# Sie bietet zudem Werkzeuge für Modelltraining, Vorhersage, Evaluation und
Datenvorverarbeitung an.


Bundesliga_Saison_2425 = "data/raw2425/D1_merged.csv"
# Definiert den Pfad zu den Rohdaten der Bundesliga-Saison 2024/2025.
# In diesen Daten enthalten sind vorallem, die Wettqoutten der anstehenden
Spiele der Saison 2024/2025.

Bundesliga_Saison_2021222324 = "data/raw/D1_merged.csv"
```

```
# Definiert den Pfad zur aktuellen Trainingsdatei, dise sind die gleichen
Rohdaten wie oben, einfach für die Bundesligasaison 2020/2021, 2021/2022,
2022/2023, 2023/2024
# Diese Datei wird im nächsten Schritt eingelesen.

Traningsdaten = pd.read_csv(Bundesliga_Saison_2021222324)
# Nun wird die CSV-Datei "data/raw/D1_merged.csv" mit pandas eingelesen und
speichert sie anschliessend als DataFrame.
# Dadurch verfügt der DataFrame 'df_train_raw'  nun über die tabellarischen
Daten aus der Datei "data/raw/D1_merged.csv".


# --- Vorbereitung der Daten für das Feature-Tuning ---
Originle_Traningsdaten = Traningsdaten.copy()
# Fertigt eine Kopie des ursprünglichen DataFrames an, damit die Originaldaten
nicht verändert werden.


# --- 1. Feature-Tunnig = die Buchmacher-Quoten von Bet365 ---
for col in ["B365H", "B365D", "B365A"]:
# Eine for-schleife durchläuft die drei Spalten "B365H", "B365D", "B365A" mit
den Quotten von dem Wettsanbeiter Bet365.
# Dabei steht 'B365H' für Heimsieg, 'B365D' für Unentschieden und 'B365A' für
Auswärtssieg.

    Originle_Traningsdaten[col] = pd.to_numeric(Originle_Traningsdaten[col],
errors="coerce")
# Diese Spalten enthalten eigentlich Strings, unteranderm die eingelesene
Zahlen, aus den CSV-Dateien.
# Mit 'pd.to_numeric()' wird sicher gestelltdass die Werte (die momentan zum
teil aus Strings bestehen) in echte numerische Datentypen umgewandelt werden.
# Dadurch werden verschiedene mathematische Berechnungen wie zum Beispiel
Division, Mittelwert berchnung ermöglicht.
# 'errors="coerce"' sorgt dafür, dass ungültige  Werte (z.B. leere Felder)
automatisch in 'NaN' (Not a Number) umgewandelt werden, dadurch werden sie von
pandas korrekt als fehlende Werte behandelt.

Originle_Traningsdaten["Wahrscheinlichkeit_Heimsieg"] = 1 /
Originle_Traningsdaten["B365H"]
Originle_Traningsdaten["Wahrscheinlichkeit_Unentschieden"] = 1 /
Originle_Traningsdaten["B365D"]
Originle_Traningsdaten["Wahrscheinlichkeit_Auswärtssieg"] = 1 /
Originle_Traningsdaten["B365A"]
# Die Buchmacherquoten "B365H", "B365D", "B365A" zeigen wie hoch die
Auszahlung für 1 Euro Einsatz ist wenn man das richtige Reultat tippt.
# Um nun die Wahrscheinlichkeiten daraus abzuleiten, wird der Kehrwert
berechnet
# Dazu wird die folgende Formel verwendet --> Wahrscheinlichkeit = 1 / Quote
```

```
Normierte_Gesammtwahrscheinlichkeit =
Originale_Traningsdaten["Wahrscheinlichkeit_Heimsieg"] +
Originale_Traningsdaten["Wahrscheinlichkeit_Unentschieden"] +
Originale_Traningsdaten["Wahrscheinlichkeit_Auswärtssieg"]
Originale_Traningsdaten["Normierte_Wahrscheinlichkeit_Heimsieg"] =
Originale_Traningsdaten["Wahrscheinlichkeit_Heimsieg"] /
Normierte_Gesammtwahrscheinlichkeit
Originale_Traningsdaten["Normierte_Wahrscheinlichkeit_Unentschieden"] =
Originale_Traningsdaten["Wahrscheinlichkeit_Unentschieden"] /
Normierte_Gesammtwahrscheinlichkeit
Originale_Traningsdaten["Normierte_Wahrscheinlichkeit_Auswärtssieg"] =
Originale_Traningsdaten["Wahrscheinlichkeit_Auswärtssieg"] /
Normierte_Gesammtwahrscheinlichkeit
# Hier werden die Wahrscheinlichkeiten normiert, also durch die Gesamtsumme
aller drei Wahrscheinlichkeiten geteilt, damit sie zusammen 1 ergeben.
# Dies ist davor nicht zwingend gegeben, da die Buchmacher die Quotten nicht
'fair' gestallten, da sie auch etwas verdiennen wollen.
# Nun haben wir eine echte Wahrscheinlichkeitsverteilung die von diesem
Problem bereinigt ist.
# Das Modell verwendet schlussendliche diese Wahrscheinlichkeiten als Input-
Features.

# --- 2. Feature-Tunnig = der Favorit des Buchmachers (Dummy-Feature) ---
Originale_Traningsdaten["Favorit_Buchmacher"] =
Originale_Traningsdaten[["B365H", "B365D",
"B365A"]].idxmin(axis=1).str.extract(r"B365(.)")[0]
# Hier wird geschaut, welche der drei Optionen (Heimsieg, Unentschieden,
Auswärtssieg) die niedrigste Quotte hat, und zwar für jede einzellne
Spielzeile.
# Dies macht man, da man so die favorisierte Option vom Buchmacher (Bet365)
zurückerhält.
# Da ja folgende Formel gilt: Wahrscheinlichkeit = 1 / Quote, das bedeutet je
tiefer die Quotte, desto höher die Wahrscheinlichkeit.
# Mit 'idxmin(axis=1)' zeigt man, den Namen der Spalte, die den kleinsten Wert
pro Zeile hat.
# Mit  folgendem Aufruf 'str.extract(r"B365(.)")' extrahiert man den
Buchstaben (H,D,A) aus dem gesamten Spaltennamen also z.B extrahiert man das H
bei 'B365H'

Originale_Traningsdaten["Buchmacher_favorisiert_Heimsieg"] =
(Originale_Traningsdaten["Favorit_Buchmacher"] == "H").astype(int)
Originale_Traningsdaten["Buchmacher_favorisiert_Unentschieden"] =
(Originale_Traningsdaten["Favorit_Buchmacher"] == "D").astype(int)
Originale_Traningsdaten["Buchmacher_favorisiert_Auswärtssieg"] =
(Originale_Traningsdaten["Favorit_Buchmacher"] == "A").astype(int)
# Hier werden drei verschiedene Variablen erstellt, diese zeigen, welche
Option (H,D,A) der Buchmacher favorsiert.
# Dies sind binär Variabeln die entweder zeigen, dass der Buchmacher diese
Option favorisiert oder eben halt nicht.
```

```
# Dies sind  nur Dummy-Variabeln, heisst ist ein vernachläsigbare Variable,
wenn man das ganze Modelle betrachtet.


# --- 3. Feature-Tuning: Kalenderinformationen ---

Originale_Traningsdaten["Datum"] =
pd.to_datetime(Originale_Traningsdaten["Date"], dayfirst=True, errors="coerce")
# Originale_Traningsdaten'["Date"]' konvertiert die Spalte "Date" in ein
passendes Datumsformat.
# 'dayfirst=True' sorgt dafür, dass das Datum in folgendem Format (TT/MM/JJJJ)
interpretiert wird.
# 'errors="coerce' stellt sicher, dass ungültige Datumsangaben in NaT (Not a
Time) umgewandelt werden.

Originale_Traningsdaten["Wochentag"] =
Originale_Traningsdaten["Datum"].dt.weekday
# Hier wird der Wochentag mit hilfe des Spieldatum Extrahiert es beginnt bei 0
= Montag, 1 = Dienstag,.... und endet bei 6 = Sonntag

Originale_Traningsdaten["Monat"] = Originale_Traningsdaten["Datum"].dt.month
# Hier wird der Monat mit hilfe des Spieldatum Extrahiert es beginnt bei 1 =
Januar, 2 = Februar,.... und endet bei 12 = Dezember

Originale_Traningsdaten["spät_Saison"] =
Originale_Traningsdaten["Monat"].apply(lambda m: 1 if m in [3,4,5] else 0)
# Mit 'Originale_Traningsdaten["spät_Saison"]'wird ein binäres Feature
erstellt, dies gibt auskunft, ob ein Spiel im März, April oder Mai stattfand.

Tatsächliches_Resultat = Originale_Traningsdaten["FTR"]
# Diese Variable zeigt das tatsächliche Spielergebniss, in dem es die Spalte
FTR (= full time Result) der Datei "data/raw/D1_merged.csv" aussliesst.
# Hier werden nur die folgenden Buchstabe (H,D,A) mit den Werten H (Heimsieg),
D (Unentschieden), A (Auswärtssieg) zurückgegeben.
# Die Anzahl Tore werden also vernachlässigt.

finale_features = [
    "Normierte_Wahrscheinlichkeit_Heimsieg",
"Normierte_Wahrscheinlichkeit_Unentschieden",
"Normierte_Wahrscheinlichkeit_Auswärtssieg", # Enthält die nomierten
Wahrscheinlichkeiten für die 3 Optionen
    "Buchmacher_favorisiert_Heimsieg", "Buchmacher_favorisiert_Unentschieden",
"Buchmacher_favorisiert_Auswärtssieg",# Dummy-Variablen für die Favoriten
Option des Buchmacher
    "Wochentag", "Monat", "spät_Saison" # Kalenderbasierte Features
]
# Hier werden die ausgewählten Merkmale (Features) aufgezählt, die für das
Modell verwendet werden sollen.
```

```
Vollständige_Originale_Traningsdaten = Originale_Traningsdaten[finale_features +
["FTR"]].dropna()
# Hier wird sichergestellt, dass nur vollständige Zeilen verwendet werden,
nicht vollständige Zeilen werden entfernt.

# --- Erstellung der Feature-Arrays und der Ziel-Arrays für das verwendete
Modell ---
Feature_Info = Vollständige_Originale_Traningsdaten[finale_features]
#'Feature_Info' enthält nun die Eingabedaten der ausgewählten Features.
Vollständige_Tatsächliches_Resultat =
Vollständige_Originale_Traningsdaten["FTR"] #
'Vollständige_Tatsächliches_Resultat' ist später unsere Zielvariabele.
Feature_Info.shape, Vollständige_Tatsächliches_Resultat.value_counts() #
'Feature_Info.shape' gibt die Anzahl der Zeilen (Anzahl Spiele) und der
Spalten (Anzahl Features)
# 'Vollständige_Tatsächliches_Resultat.value_counts()' zählt wie oft es
welches Resultat (Heimsieg, Unentscheiden, Auswärtsieg) gab.

# --- Korrelationsmatrix berechnen und visualisieren ---
Korrelationsmatrix = Feature_Info.corr()
# Diese Korrelationsmatrix soll zeigen, wie stark das zwei Variablen
miteinander zusammenhängen.
# Hier liegt der Korrelationswert zwischen -1 und +1
# +1 bedeudet eine perfekte positive Korrelation, wohingegen -1 eine perfekte
negative Korrelation bedeutet und 0 gar kein Zusammenhang zwischen den
Variablen bedeutet.
# Eine zu hohe Korrelation zwischen zwei Features ist nicht wünschenswert, da
sie die Komplexität des Modells erhöht, ohne dem Modell zusätzliche
Informationen zu geben.
# Hier wurde versucht Features die stark korrelieren, zu identifizieren und
anschliessend wurden redundante Merkmale entfernt,
# wie dies bei 'Bookie_Favor_A' z.B der Fall war, dort war das Feature genau
das gegenteil zu 'Bookie_Favor_H'.
# Dieser Prozess sollte Overfitting reduzieren und erhöht somit die
Interpretierbarkeit.


# --- Visualisierung der berechneten Korrelationsmatrix ---
plt.figure(figsize=(16, 6))   #gibt die Grösse der Grafik (Korrelationsmatrix)
vor
sns.heatmap(
            Korrelationsmatrix, #Ist die zuvor berechnete Korrelationsmatrix
            annot=True, #Gibt die Korrelationswerte der einzellnen Zeilen
zurück
            fmt=".2f", #Rundet auf 2 Nachkommastellen
            cmap="RdYlGn",#Gibt die Farbskala vor: (Rot, Gelb, Grün)
            square=True, #Macht das die ausgegebenen Zellen Quadratisch sind.
            cbar=True,#Zeigt eine Farblegende unter dem Diagram an
            annot_kws={"size": 12},
```

```
            xticklabels=[label.replace('_', '\n') for label in
Korrelationsmatrix.columns],

)
plt.title("Korrelationsmatrix der Finalen Model-Features")# Verseht die Grafik
mit einem Titel
plt.tight_layout()#sorgt für ein optimales Layout in dem es prüft, dass nichts
abgeschnitten wird.
plt.savefig("Korrelationsmatrix der Finalen Model-Featurespng.png")
plt.show()#Zeigt die Grafik (Korrelationsmatrix)


Korrelationsmatrix # Gibt eine ausgabe der Korrelationsmatrix als DataFrame
zurück.


# --- Auswahl der zu reduzierenden Features zur Vermeidung von Redundanzen ---
reduzierte_finale_features = [
    "Normierte_Wahrscheinlichkeit_Heimsieg",
"Normierte_Wahrscheinlichkeit_Unentschieden", #nomierte Wahrscheinlichkeiten
für Heimsieg und Unentschieden.
    "Buchmacher_favorisiert_Heimsieg", #Dummy-Variable, Buchmacher favorsiert
Heimteam
    "Wochentag", "Monat", "spät_Saison" #Kalenderinformation (Wochentag und
Monat)
]

# Erstellung eines neuen DataFrames, mit den reduzierten Merkmalen
Feature_Info_final =
Vollständige_Originle_Traningsdaten[reduzierte_finale_features]

# --- Neue Korrelationsmatrix mit reduzierten Features ---
Korrelationsmatrix = Feature_Info_final.corr()
# Berechnet die Korrelationsmatrix erneut, aber nun nur noch für die
ausgewählten Features.
# Dies sollte kontrolieren, ob zu starke Korrelationen korrekt entfernt
wurden.


# Visualisieren
plt.figure(figsize=(16, 6)) #gibt die Grösse der Grafik (Korrelationsmatrix)
vor
sns.heatmap(Korrelationsmatrix, #Ist die zuvor berechnete Korrelationsmatrix
            annot=True, #Gibt die Korrelationswerte der einzellnen Zeilen
zurück
            fmt=".2f", #Rundet auf 2 Nachkommastellen
            cmap="RdYlGn",#Gibt die Farbskala vor: (Rot, Gelb, Grün)
            square=True, #Macht das die ausgegebenen Zellen Quadratisch sind.
            cbar=True,#Zeigt eine Farblegende unter dem Diagram an
            annot_kws={"size": 12},
```

```
            xticklabels=[label.replace('_', '\n') for label in
Korrelationsmatrix.columns],
            yticklabels=[label.replace('_', '\n') for label in
Korrelationsmatrix.index]
)
plt.title("Korrelationsmatrix der Finalen Model-Features_v2")# Verseht die
Grafik mit einem Titel
plt.tight_layout()#sorgt für ein optimales Layout in dem es prüft, dass nichts
abgeschnitten wird.
plt.savefig("Korrelationsmatrix der Finalen Model-Features_v2.png")
plt.show()#Zeigt die Grafik (Korrelationsmatrix)


Korrelationsmatrix
# Gibt eine ausgabe der Korrelationsmatrix als DataFrame zurück.

# --- Nochmals eine auswahl der zu reduzierenden Features zur Vermeidung von
Redundanzen ---
#Dafür wurden nur Features ausgewählt die tatsächlich schon vor dem Spieltag
bekannt sind.
reduzierte_finale_features_v2 = [
    "Normierte_Wahrscheinlichkeit_Heimsieg",
"Normierte_Wahrscheinlichkeit_Unentschieden", #nomierte Wahrscheinlichkeiten
für Heimsieg und Unentschieden.
    "Buchmacher_favorisiert_Heimsieg", #Dummy-Variable, Buchmacher favorsiert
Heimteam
    "Wochentag", "Monat", "spät_Saison" #Kalenderinformation (Wochentag und
Monat)
]

Verwendtete_Features =
Vollständige_Originle_Traningsdaten[reduzierte_finale_features_v2]
# 'Verwendtete_Features' sind also die Informationen die wir dem Modell geben,
sodass es Vorhersagen treffen kann.
# In unserem Modell wären das z.B. die Wahrscheinlichkeiten der Quoten von
Bet365, die Wochentag oder der Monat.

# Nun teilen wir die Daten in Trainings und Testdaten auf.
X_traings_daten, X_test_daten, y_traings_daten, y_test_daten =
train_test_split(
    # 80 % der Daten werden verwendet, um das Modell zu trainieren
(X_traings_daten, y_trainngs_daten),
    Verwendtete_Features, Vollständige_Tatsächliches_Resultat, test_size=0.2,
stratify=Vollständige_Tatsächliches_Resultat, random_state=42)
    # die anderen 20 % werden später als Testdaten verwendet, also die Daten
die zeigen wie gut das Modell generalisierbar ist (X_test_daten,
y_test_daten).
    # Mit 'stratify=y_reduced' wird sicher gestellt, dass in den Tranings und
Testdaten, ungefähr das gleiche Verhältnis von den Resulatten (H,D,A)
vorkommt.
```

```
    # Mit dem Parameter 'random_state=42' wird dafür gesorgt, dass die
Aufteilung wiederhergestellt werden kann.

Model_GradientBoosting = GradientBoostingClassifier(
    n_estimators=1000,      # Anzahl der Bäume
    learning_rate=0.001,     # Schrittweite beim Lernen (kleiner = stabiler,
grösser = schneller)
    max_depth=5,           # Tiefe der einzelnen Bäume
    random_state=42
)
# Training mit den Trainingsdaten
Model_GradientBoosting.fit(X_traings_daten, y_traings_daten)

# Vorhersage mit den Testdaten
vorhergesagte_Resultate = Model_GradientBoosting.predict(X_test_daten)

# === Evaluation ===
accuracy = accuracy_score(y_test_daten, vorhergesagte_Resultate)
report = classification_report(y_test_daten, vorhergesagte_Resultate,
output_dict=True)
# Wir messen, zuerst wie gut die Vorhergesagten Reusltate mit den
tatsächlichen Resultaten übereinstimmen (Accuracy).
# Zusätzlich wird ein detaillierten Bericht mit Precision, Recall und F1-Score
pro Klasse erstellt.

# In DataFrame umwandeln
report = pd.DataFrame(report).T
# "accuracy" extrahieren

report = report.drop(index=[ "macro avg", "weighted avg"], errors="ignore")

# Nur relevante Spalten behalten und runden
report = report[["precision", "recall", "f1-score", "support"]].round(2)

# Bereite Tabelle für Anzeige vor
report.index.name = "Klasse"
report.reset_index(inplace=True)
report

fig, ax = plt.subplots(figsize=(8, 3.5))
ax.axis('off')

# Tabelle zeichnen
table = ax.table(
    cellText=report.values,
    colLabels=report.columns,
    loc='center',
    cellLoc='center'
)
```

```
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

# Zebra-Stil für bessere Übersicht
for i in range(len(report)):
    for j in range(len(report.columns)):
        color = "#f0f0f0" if i % 2 == 0 else "white"
        table[(i+1, j)].set_facecolor(color)

# Kopfzeile hervorheben
for j in range(len(report.columns)):
    table[(0, j)].set_facecolor("#d0d0d0")
    table[(0, j)].set_text_props(weight='bold')

# Titel
plt.title("Classification Report (Saison Testdaten)")

# Accuracy separat fett darunter darstellen
plt.text(0.5, -0.25, f"accuracy: {accuracy:.2f}",
         ha="center", va="center", fontsize=12, fontweight="bold")

plt.tight_layout()
plt.savefig("Classification_Report_Testdaten.png")
plt.show()

### === Feature-Wichtigkeit für Gradient Boosting ===

# Extrahiere die Feature Importances
Importances = Model_GradientBoosting.feature_importances_

# Erstelle ein DataFrame für die Visualisierung
Feature_Wichtigkeit = pd.DataFrame({
    "Feature": X_traings_daten.columns,  # oder X_traings_daten_scaliert,
falls skaliert
    "Wichtigkeit": Importances
}).sort_values(by="Wichtigkeit", ascending=False)

# Visualisierung
plt.figure(figsize=(12, 6))
sns.barplot(data=Feature_Wichtigkeit, x="Wichtigkeit", y="Feature")
plt.title("Wichtigkeit der Features im Gradient-Boosting-Modell")
plt.tight_layout()
plt.savefig("Feature_Wichtigkeit_GradientBoosting.png")
plt.show()

# === 2. Confusion Matrix ===
labels = ["H", "D", "A"] # Definiert die Labels Heimsieg, Unentschieden und
Auswärtssieg in dieser Reihenfolge.
```

```
Confusion_Matrix = confusion_matrix(y_test_daten, vorhergesagte_Resultate,
labels=labels) # Erstellt die Confusion Matrix für das Modell auf Basis der
Testdaten
Confusion_Matrix_final = pd.DataFrame(Confusion_Matrix, index=labels,
columns=labels) # Wandelt die Matrix in ein DataFrame um, um sie später als
Heatmap darzustellen zu können.

# Visualisierung der Confusion Matrix als Heatmap
plt.figure(figsize=(8, 6))#Gibt die grösse der Grafik vor
sns.heatmap(Confusion_Matrix_final, #erstellt eine Heatmap der Confusion
Matrix mit hilfe von Seaborn.
            annot=True, #zeigt die numerischen Werte der Confusion Matrix an.
            fmt=".2f", #rundet auf 2 Nachkommastellen
            cmap="RdYlGn")#Verseht die Heatmap mit einer Farbskala
plt.title("Confusion Matrix (Vorhergesagtes vs Tatsächliches Resultat)
Testdaten")# Die Grafik wird mit einem Titel versehen
plt.xlabel("Vorhergesagt") #X-Achse wird beschriftet
plt.ylabel("Tatsächlich")#Y-Achse wird beschriftet
plt.tight_layout()# Hier wird das Layout optimiert, indem nichts abgeschnitten
wird
plt.savefig('Confusion Matrix_modell_training.png')# Speichert die Grafik
plt.show()#Zeigt die finale Grafik (Heatmap)

# === Anwendung auf Saison 2024/25 ===
Saison_2425 = pd.read_csv(Bundesliga_Saison_2425)
Saison_2425["Datum"] = pd.to_datetime(Saison_2425["Date"], dayfirst=True,
errors="coerce")
# 'Saison_2425["Datum"]' konvertiert die Spalte "Date" in ein passendes
Datumsformat.
# 'dayfirst=True' sorgt dafür, dass das Datum in folgendem Format (TT/MM/JJJJ)
interpretiert wird.
# 'errors="coerce"' stellt sicher, dass ungültige Datumsangaben in NaT (Not a
Time) umgewandelt werden.

# Hier wird wieder die implizite Wahrscheinlichkeiten aus den Wettquoten von
Bet365 ausgerechnet --> Erklärung oben
Saison_2425["Wahrscheinlichkeit_Heimsieg"] = 1 / Saison_2425["B365H"]
#Heimsieg
Saison_2425["Wahrscheinlichkeit_Unetschieden"] = 1 / Saison_2425["B365D"]
#Unentschieden
Saison_2425["Wahrscheinlichkeit_Auswärtsieg"] = 1 / Saison_2425["B365A"]
#Auswärtsteam siegt
total_2223 = Saison_2425["Wahrscheinlichkeit_Heimsieg"] +
Saison_2425["Wahrscheinlichkeit_Unetschieden"] +
Saison_2425["Wahrscheinlichkeit_Auswärtsieg"]
# Hier werden die Wahrscheinlichkeiten aufsummiert, diese werden später
verwendet um die Wahrscheinlichkeiten pro Ereignis, zu normieren sodass alle
Wahrscheinlichkeiten zusammen 1 ergeben.
# Dies ist davor nicht zwingend gegeben, da die Buchmacher die Quotten nicht
'fair' gestallten, da sie auch etwas verdiennen wollen.
```

```
Saison_2425["Normierte_Wahrscheinlichkeit_Heimsieg"] =
Saison_2425["Wahrscheinlichkeit_Heimsieg"] / total_2223 #Wahrscheinlichkeit
für Heimsieg normiert
Saison_2425["Normierte_Wahrscheinlichkeit_Unentschieden"] =
Saison_2425["Wahrscheinlichkeit_Unetschieden"] / total_2223
#Wahrscheinlichkeit für ein Unentschieden normiert

em_score = {
    "Bayern Munich": 0.7, "Leverkusen": 0.7, "Ein Frankfurt": 0.1,
    "Dortmund": 0.8, "Freiburg": -0.2, "Mainz": 0.1, "RB Leipzig": 0.4,
    "Werder Bremen": -0.1, "Stuttgart": 0.5, "M'gladbach": 0.1,
    "Wolfsburg": -0.5, "Augsburg": 0.1, "Union Berlin": -0.3,
    "St Pauli": 0.0, "Hoffenheim": 0.1, "Heidenheim": 0.0,
    "Holstein Kiel": 0.0, "Bochum": 0.0
}
Saison_2425["Heimteam"] = Saison_2425["HomeTeam"].map(em_score)
Saison_2425["Auswärtsteam"] = Saison_2425["AwayTeam"].map(em_score)
Saison_2425["EM_Differenz"] = Saison_2425["Heimteam"] -
Saison_2425["Auswärtsteam"]
Saison_2425["Normierte_Wahrscheinlichkeit_Heimsieg"] *=  (1 + 0.1 *
Saison_2425["EM_Differenz"])

Saison_2425["Favorit_Buchmacher"] = Saison_2425[["B365H", "B365D",
"B365A"]].idxmin(axis=1).str.extract(r"B365(.)")[0] # Ermittle, welches
Ergebnis laut Bet365 am wahrscheinlichsten ist.
Saison_2425["Buchmacher_favorisiert_Heimsieg"] =
(Saison_2425["Favorit_Buchmacher"] == "H").astype(int) # ein Binäres Feature
das zeigt ob der Favorit laut Buchmacher (Bet365) der Heimsieg ist.
Saison_2425["Wochentag"] = Saison_2425["Datum"].dt.weekday # Extrahiert
Wochentag aus dem Datum
Saison_2425["Monat"] = Saison_2425["Datum"].dt.month # Extrahiert Monat aus
dem Datum
Saison_2425["spät_Saison"] = Saison_2425["Monat"].apply(lambda m: 1 if m in
[3,4,5] else 0)

Feature_Info_final_2425 = Saison_2425[reduzierte_finale_features_v2].dropna()#
Auswahl der schlussendlichem Features für das finale Modell.
Vollständige_Tatsächliches_Resultat_2425 =
Saison_2425.loc[Feature_Info_final_2425.index, "FTR"] #Die Zielvariabel mit
den tatsächlichen Resultaten (H,D,A)

vorhergesagte_Resultate2425 = Model_GradientBoosting
.predict(Feature_Info_final_2425) # Vorhersage auf basis des trainierten
Random-Forest-Modell
accuracy_2425 = accuracy_score(Vollständige_Tatsächliches_Resultat_2425,
vorhergesagte_Resultate2425) # Wir messen, zuerst wie gut die Vorhergesagten
Reusltate mit den tatsächlichen Resultaten übereinstimmen (Accuracy).
```

```
report_2425 = classification_report(Vollständige_Tatsächliches_Resultat_2425,
vorhergesagte_Resultate2425, output_dict=True) # Zusätzlich wird ein
detaillierten Bericht mit Precision, Recall und F1-Score pro Klasse erstellt.
labels = ["H", "D","A"] # # Definiert die Labels Heimsieg, Unentschieden und
Auswärtssieg in dieser Reihenfolge. Bereite so die Labels auf Einheitlichkeit
vor.

# Confusion Matrix
Confusion_Matrix = confusion_matrix(Vollständige_Tatsächliches_Resultat_2425,
vorhergesagte_Resultate2425, labels=labels)# Erstellt die Confusion Matrix für
das Modell auf Basis der Testdaten
Confusion_Matrix_final = pd.DataFrame(Confusion_Matrix, index=labels,
columns=labels) # Wandelt die Matrix in ein DataFrame um, um sie später als
Heatmap darzustellen zu können.

# Visualisierung der Confusion Matrix als Heatmap
plt.figure(figsize=(8, 6))# Gibt die grösse der Grafik vor
sns.heatmap(Confusion_Matrix_final, #erstellt eine Heatmap der Confusion
Matrix mit hilfe von Seaborn.
            annot=True, #zeigt die numerischen Werte der Confusion Matrix an.
            fmt=".2f", #rundet auf 2 Nachkommastellen
            cmap="RdYlGn")#Verseht die Heatmap mit einer Farbskala
plt.title("Confusion Matrix (Vorhergesagtes vs Tatsächliches Resultat) Saison
2024/2025")# Die Grafik wird mit einem Titel versehen
plt.xlabel("Vorhergesagt") #X-Achse wird beschriftet
plt.ylabel("Tatsächlich")#Y-Achse wird beschriftet
plt.tight_layout()# Hier wird das Layout optimiert, indem nichts abgeschnitten
wird
plt.savefig('Confusion_Matrix_2425.png')# Speichert die Grafik
plt.show()#Zeigt die finale Grafik (Heatmap)

correct_mask = (vorhergesagte_Resultate2425 ==
Vollständige_Tatsächliches_Resultat_2425.values) # Balkendiagramm das die
Korrekten Vorhersagen und die falschen Vorhersagen pro Klasse zeigt
correct_labels =
Vollständige_Tatsächliches_Resultat_2425[correct_mask].value_counts().reindex(
labels, fill_value=0) # Zählt alle korrekten Vorhersagen pro Klasse
incorrect_labels =
Vollständige_Tatsächliches_Resultat_2425[~correct_mask].value_counts().reindex
(labels, fill_value=0) # Zählt alle inkorrekten Vorhersagen pro Klasse.


# Erstellt ein DataFrame für gestapelte Balken
Balkendiagramm = pd.DataFrame({
    "Korekte": correct_labels,
    "Inkorekte": incorrect_labels
}, index=labels)


# Zeichnet ein gestapeltes Balkendiagramm
```

```
Balkendiagramm.plot(kind="bar", stacked=True, figsize=(8, 5), colormap="Set2")
plt.title("Wie oft wurde welches Spielergebnis richtig erkannt")
plt.xlabel("Spielergebnis")
plt.ylabel("Anzahl Matches")
plt.xticks(rotation=0)
plt.legend(title="Vorhersage")
plt.tight_layout()
plt.savefig('Genauigkeit_2425.png')
plt.show()


finaler_report = pd.DataFrame(report_2425).T # 3. Klassifikations-Report als
Tabelle anzeigen
finaler_report = finaler_report.drop(index=["accuracy", "macro avg", "weighted
avg"], errors="ignore") # Entfernt zusammenfassende / Redunate Zeilen
finaler_report = finaler_report[["precision", "recall", "f1-score",
"support"]].round(2) # Wählt alle relevanten Metriken und runde sie auf 2
Nachkommastellen.


# Bereite Tabelle für Anzeige vor
finaler_report.index.name = "Klasse"
finaler_report.reset_index(inplace=True)
finaler_report

# --- Klassifikations-Report als Tabelle + Accuracy separat darstellen ---
accuracy_value = report_2425["accuracy"]

fig, ax = plt.subplots(figsize=(8, 3.5))
ax.axis('off')

# Tabelle zeichnen
table = ax.table(
    cellText=finaler_report.values,
    colLabels=finaler_report.columns,
    loc='center',
    cellLoc='center'
)

table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

# Zebra-Stil für bessere Übersicht
for i in range(len(finaler_report)):
    for j in range(len(finaler_report.columns)):
        color = "#f0f0f0" if i % 2 == 0 else "white"
        table[(i+1, j)].set_facecolor(color)

# Kopfzeile hervorheben
```

```
for j in range(len(finaler_report.columns)):
    table[(0, j)].set_facecolor("#d0d0d0")
    table[(0, j)].set_text_props(weight='bold')

# Titel
plt.title("Classification Report (Saison 2024/2025)")

# Accuracy separat fett darunter darstellen
plt.text(0.5, -0.25, f"Accuracy: {accuracy_value:.2f}",
         ha="center", va="center", fontsize=12, fontweight="bold")

plt.tight_layout()
plt.savefig("Classification_Report_2425.png")
plt.show()
```

(Anhang 12: Gradient-Boosting-Modell (mit EM-Feature))

UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Faculté des sciences économiques et sociales
Wirtschafts- und sozialwissenschaftliche Fakultät
Boulevard de Pérolles 90
CH-1700 Fribourg

E R K L Ä R U N G

Ich bestätige mit meiner Unterschrift, dass ich die Arbeit persönlich erstellt und dabei nur die aufgeführten Quellen und Hilfsmittel verwendet sowie wörtliche Zitate und Paraphrasen als solche gekennzeichnet habe.

Es ist mir bekannt, dass andernfalls die Fakultät gemäss der Entscheidung des Fakultätsrats vom 09.11.2004 das Recht hat, den auf Grund dieser Arbeit verliehenen Titel zu entziehen.

Ich erkläre hiermit weiterhin, dass diese Arbeit bzw. Teile daraus noch nicht in dieser Form an anderer Stelle als Prüfungsleistung eingereicht worden sind, gemäss der Entscheidung des Fakultätsrats vom 18.11.2013.

Freiburg, den 10.11.2025

(Unterschrift)