Hochschule Luzern HSLU

Artificial Intelligence & Machine Learning

Studienprojekt

Vergleich der Leistungsfähigkeit verschiedener Machine-Learning-Modelle zur Vorhersage von Zahlungsausfällen: Eine Analyse anhand des 'Default of Credit Card Clients'-Datensatzes

Verfasser:
Noel Gugler

Herzogenbuchsee, 29. Dezember 2024

INHALTSVERZEICHNIS

1 EINLEITUNG

1.1 PROBLEMSTELLUNG

In der modernen Finanzwelt spielen Kreditkarten eine zentrale Rolle im täglichen Leben von Millionen von Menschen. Gleichzeitig birgt die Vergabe von Krediten erhebliche Risiken für Kreditgeber, insbesondere durch Zahlungsausfälle, die zu finanziellen Verlusten führen können. Eine effektive Analyse und Vorhersage von Zahlungsausfällen ist daher entscheidend, um die Kreditvergabe zu optimieren und Risiken zu minimieren.

Im Zeitalter von Big Data und Künstlicher Intelligenz bieten Machine-Learning-Modelle eine vielversprechende Möglichkeit, dieses Problem anzugehen. Mit ihrer Fähigkeit, Muster in grossen Datenmengen zu erkennen, haben sie sich als wertvolle Werkzeuge für die Kreditrisikoanalyse erwiesen. Allerdings gibt es eine Vielzahl von Modellen, die in diesem Kontext eingesetzt werden können – von einfachen statistischen Methoden wie der Logistischen Regression bis hin zu komplexeren Algorithmen wie Random Forest und XGBoost. Jedes dieser Modelle bringt spezifische Stärken und Schwächen mit sich, und die Wahl des optimalen Modells hängt von zahlreichen Faktoren wie Datenstruktur, Modellinterpretierbarkeit und Leistung ab.

In dieser Arbeit wird der Fokus auf die Untersuchung der Leistungsfähigkeit verschiedener Machine-Learning-Modelle bei der Vorhersage von Zahlungsausfällen gelegt. Basierend auf dem Datensatz *"Default of Credit Card Clients"*[1] aus Taiwan wird analysiert, wie gut die Modelle Logistische Regression, Random Forest und XGBoost Zahlungsausfälle vorhersagen können. Der Datensatz umfasst sowohl demografische Merkmale als auch Kredit- und Zahlungsinformationen von Kreditkartenkunden, was eine umfassende Analyse ermöglicht.

1.2 ZIELSETZUNG DER ARBEIT

Die Zielsetzung dieser Arbeit besteht darin, durch einen systematischen Vergleich die Stärken und Schwächen der Modelle zu bewerten und herauszufinden, welches Modell

[1] https://www.kaggle.com/datasets/uciml/default-of-credit-card-clients-dataset Zugriff: 10.12.2024

für diese Art von Klassifikationsproblem am besten geeignet ist. Dabei werden Performancemetriken wie Genauigkeit, Präzision, Recall, der F1-Score sowie die AUC der ROC-Kurve herangezogen, um die Vorhersageleistung der Modelle zu messen.

Die Ergebnisse dieser Arbeit sollen nicht nur einen Beitrag zur akademischen Forschung leisten, sondern auch praktische Implikationen für die Finanzindustrie aufzeigen, indem sie eine fundierte Grundlage für die Wahl geeigneter Machine-Learning-Modelle zur Kreditrisikoanalyse bieten.

1.3 AUFBAU DER ARBEIT

Im weiteren Verlauf der Arbeit werden zunächst die theoretischen Grundlagen vorgestellt, gefolgt von der Beschreibung des Datensatzes, der Methodik und der Ergebnisse. Abschliessend werden die Erkenntnisse diskutiert und potenzielle Limitationen sowie Ansätze für zukünftige Forschung aufgezeigt.

2 THEORETISCHE GRUNDLAGEN

2.1 KREDITRISIKO UND ZAHLUNGSAUSFÄLLE

Das Kreditrisiko bezeichnet die Gefahr, dass ein Kreditnehmer seinen vertraglichen Zahlungsverpflichtungen gegenüber dem Kreditgeber nicht oder nur unvollständig nachkommt. Dies kann zu finanziellen Verlusten für den Kreditgeber führen. Das Kreditrisiko ist somit eine zentrale Komponente im Risikomanagement von Banken und Finanzinstituten.[2]

Ein **Zahlungsausfall** tritt ein, wenn der Kreditnehmer nicht in der Lage ist, die vereinbarten Zins- und Tilgungszahlungen fristgerecht zu leisten. Solche Ausfälle können verschiedene Ursachen haben, darunter finanzielle Schwierigkeiten des Kreditnehmers, wirtschaftliche Abschwünge oder unerwartete Ereignisse wie Arbeitslosigkeit oder Krankheit. Die Folgen von Zahlungsausfällen sind für Kreditgeber gravierend, da sie direkte finanzielle Verluste bedeuten und die Liquidität sowie die Kapitalausstattung der Institution beeinträchtigen können.[3]

Um das Kreditrisiko zu bewerten und Zahlungsausfälle zu prognostizieren, setzen Finanzinstitute auf verschiedene Methoden und Modelle. Traditionell wurden statistische Ansätze wie die Logistische Regression verwendet, um die Wahrscheinlichkeit eines Zahlungsausfalls basierend auf historischen Daten und bestimmten Kreditnehmermerkmalen zu schätzen. Mit dem Fortschritt in der Datenverarbeitung und der Verfügbarkeit grosser Datenmengen gewinnen jedoch zunehmend komplexere Machine-Learning-Modelle an Bedeutung. Modelle wie der **Random Forest** oder **XGBoost** können nichtlineare Zusammenhänge in den Daten erfassen und bieten somit potenziell präzisere Vorhersagen. Allerdings erfordern sie auch eine sorgfältige Validierung und Kalibrierung, um Überanpassungen zu vermeiden und die Generalisierbarkeit sicherzustellen.

---

[2] https://www.gabler-banklexikon.de/definition/kreditrisiko-59434?utm_source=chatgpt.com, 15.12.2024, Text angepasst

[3] https://www.nordwest-factoring.de/blog/ausfallrisiko/#:~:text=Ein%20Zahlungsausfall%20bedeutet%20f%C3%BCr%20den,die%20Bonit%C3%A4t%20des%20Kreditors%20verschlechtert. Zugriff: 15.12.20204

Die effektive Bewertung des Kreditrisikos und die Vorhersage von Zahlungsausfällen sind essenziell, um Kreditentscheidungen fundiert zu treffen, Zinssätze angemessen zu gestalten und regulatorische Anforderungen zu erfüllen. Ein präzises Risikomanagement trägt dazu bei, die Stabilität des Finanzsystems zu gewährleisten und das Vertrauen der Anleger und Kunden in die Finanzmärkte aufrechtzuerhalten.

2.2 KLASSIFIKATIONSPROBLEME IM MASCHINELLEN LERNEN

Im Bereich des maschinellen Lernens stellen **Klassifikationsprobleme** eine zentrale Aufgabe dar. Sie zielen darauf ab, Dateninstanzen einer oder mehreren vordefinierten Kategorien zuzuordnen. Dies geschieht anhand von Input-Daten und deren Merkmalen, wobei ein Klassifikationsmodell anhand eines Trainingsdatensatzes erstellt wird, der sowohl die Merkmale als auch die zugehörigen Kategorien (Labels) enthält.

2.2.1 Definition und Grundlagen

Ein Klassifikationsproblem liegt vor, wenn das Ziel darin besteht, eine qualitative Variable (z. B. „ja/nein“, „Kategorie A/B/C“) vorherzusagen. Typische Beispiele sind:

- Vorhersage, ob ein Kunde einen Kredit zurückzahlt (Zahlungsausfall: ja/nein).
- Identifikation von Spam in E-Mails (Spam/Nicht-Spam).
- Klassifikation von medizinischen Diagnosen basierend auf Patientendaten.

Klassifikationsprobleme gehören zur **überwachten Lernmethode** im maschinellen Lernen. Dabei wird ein Modell erstellt, das die Beziehung zwischen den Eingabedaten (Features) und den Zielvariablen (Labels) erlernt.

2.2.2 Algorithmische Ansätze

Es existieren zahlreiche Algorithmen, um Klassifikationsprobleme zu lösen. Diese unterscheiden sich hinsichtlich ihrer Komplexität, Leistung und den zugrunde liegenden Annahmen über die Daten. Zu den bekanntesten Algorithmen zählen:

1. **Logistische Regression:**
   Ein einfacher, aber effektiver Ansatz, insbesondere für binäre

Klassifikationsprobleme. Die logistische Regression modelliert die Wahrscheinlichkeit einer Kategorie anhand einer Sigmoid-Funktion und ist insbesondere dann geeignet, wenn die Datenlinear separierbar sind.

2. **Entscheidungsbäume und Random Forests**: Entscheidungsbäume basieren auf einer hierarchischen Struktur, die Entscheidungen durch logische Bedingungen trifft. Random Forests kombinieren mehrere Entscheidungsbäume (Ensemble-Ansatz), um die Vorhersagegenauigkeit zu erhöhen und Überanpassungen zu reduzieren.
3. **Gradient Boosting-Algorithmen (z. B. XGBoost)**: Diese Ansätze verbessern die Leistung eines schwachen Lerners (z. B. eines einfachen Entscheidungsbaums) durch iterative Fehlerreduktion. Gradient Boosting ist besonders leistungsfähig bei grossen und komplexen Datensätzen.
4. **Neuronale Netze**: Diese Modelle sind inspiriert von biologischen Gehirnstrukturen und können komplexe, nichtlineare Beziehungen in den Daten modellieren. Sie eignen sich besonders für grosse Datensätze mit vielen Features.

2.2.3 Leistungsbewertung von Klassifikationsmodellen

Die Bewertung der Güte eines Klassifikationsmodells ist entscheidend, um dessen Praxistauglichkeit sicherzustellen. Häufig eingesetzte Metriken sind:

- **Genauigkeit (Accuracy)**: Anteil korrekt klassifizierter Instanzen.
- **Präzision und Recall**: Fokus auf die Vorhersagegenauigkeit und -vollständigkeit einzelner Klassen.
- **F1-Score**: Harmonisches Mittel aus Präzision und Recall, besonders bei unausgewogenen Datensätzen relevant.
- **ROC-AUC (Receiver Operating Characteristic - Area Under Curve)**: Mass für die Trennfähigkeit des Modells über verschiedene Schwellenwerte hinweg.

2.2.4 Herausforderungen und Limitationen

Trotz ihrer breiten Anwendbarkeit können Klassifikationsprobleme mit Herausforderungen verbunden sein:

- **Datenqualität**: Unvollständige oder fehlerhafte Daten beeinflussen die Modellleistung.
- **Unausgewogene Datensätze**: Wenn eine Klasse stark überrepräsentiert ist, kann dies zu Verzerrungen führen.
- **Interpretierbarkeit**: Komplexe Modelle wie XGBoost oder neuronale Netze sind oft schwer verständlich, was bei Entscheidungen im Finanzwesen problematisch sein kann.

Die Anwendung von Klassifikationsmodellen in der Kreditkarten-Ausfallanalyse erfordert eine sorgfältige Auswahl und Evaluierung geeigneter Algorithmen, um robuste und verlässliche Vorhersagen zu ermöglichen.

2.3 Überblick über die untersuchten Machine-Learning-Modelle

In diesem Abschnitt werden die Machine-Learning-Modelle vorgestellt, die in der vorliegenden Arbeit zur Vorhersage von Zahlungsausfällen bei Kreditkartenkunden eingesetzt werden. Der Fokus liegt auf drei häufig verwendeten und leistungsfähigen Ansätzen: der logistischen Regression, dem Random Forest und dem XGBoost-Algorithmus. Diese Modelle wurden aufgrund ihrer unterschiedlichen Eigenschaften und ihrer etablierten Anwendung in Klassifikationsproblemen ausgewählt.

2.3.1 Logistische Regression

Die **logistische Regression** ist ein einfaches und interpretierbares Modell für binäre Klassifikationsprobleme. Sie modelliert die Wahrscheinlichkeit, dass eine Instanz zu einer bestimmten Klasse gehört, mithilfe einer Sigmoid-Funktion:

$$P(y = 1 \mid X) = \frac{\mathbf{1}}{1 + e^{-(\beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n)}}$$ [4]

- **Stärken**:
    - Einfache Implementierung und schnelle Berechnung.
    - Hohe Interpretierbarkeit der Koeffizienten, was insbesondere in regulierten Bereichen wie dem Finanzwesen von Vorteil ist.
    - Geeignet für lineare Entscheidungsgrenzen.
- **Schwächen**:
    - Begrenzte Leistungsfähigkeit bei nichtlinearen Beziehungen zwischen Features und Zielvariablen.
    - Anfällig für Multikollinearität zwischen den Features.

Die logistische Regression ist oft ein Ausgangspunkt für die Analyse, da sie grundlegende Einblicke in die zugrunde liegende Datenstruktur bietet.

2.3.2 Random Forest

Der **Random Forest** ist ein Ensemble-Lernverfahren, das auf der Aggregation von Entscheidungsbäumen basiert. Jeder Baum wird auf einem zufälligen Teil des Trainingsdatensatzes und einer zufälligen Untermenge der Features trainiert. Die finale Vorhersage erfolgt durch Mehrheitsvotum (Klassifikation) oder Mittelwertbildung (Regression).

- **Stärken**:
    - Robust gegenüber Überanpassung durch die Kombination mehrerer Bäume.
    - Kann komplexe, nichtlineare Beziehungen modellieren.

[4] Logistische Regressionsanalyse
 | Methodenberatung | UZH Zugriff 21.12.2024

- Handhabung von unausgewogenen Datensätzen durch Gewichtung der Klassen möglich.

- **Schwächen**:
  - Weniger interpretierbar als ein einzelner Entscheidungsbaum.
  - Rechenintensiv bei sehr grossen Datensätzen.

Random Forests sind bekannt für ihre Vielseitigkeit und Stabilität und eignen sich gut für Probleme, bei denen die Beziehung zwischen den Variablen nichtlinear und komplex ist.

2.3.3 – XGBoost

Der **Extreme Gradient Boosting (XGBoost)**-Algorithmus ist eine Erweiterung des Gradient-Boosting-Ansatzes, der Entscheidungsbäume iterativ trainiert, um Fehler der vorherigen Bäume zu reduzieren. XGBoost ist für seine Effizienz, Genauigkeit und Anpassungsfähigkeit bekannt.

- **Stärken**:
  - Hervorragende Leistung bei strukturierten Daten.
  - Umfassende Funktionen zur Regularisierung, die Überanpassung minimieren.
  - Hohe Flexibilität bei der Optimierung durch anpassbare Verlustfunktionen.
- **Schwächen**:
  - Erhöhte Komplexität im Vergleich zu anderen Algorithmen.
  - Geringere Interpretierbarkeit als einfache Modelle wie die logistische Regression.

XGBoost hat sich als leistungsstarker Algorithmus in vielen Machine-Learning-Wettbewerben bewährt und ist besonders geeignet für grosse Datensätze mit vielen Features und komplexen Mustern.

3 DATENSATZBESCHREIBUNG UND EXPLORATIVE DATENANALYSE

3.1 HERKUNFT UND STRUKTUR DES DATENSATZES

Der verwendete Datensatz, bekannt als "Default of Credit Card Clients Dataset", stammt aus Taiwan und umfasst Informationen zu Kreditkartenkunden von April 2005 bis September 2005. Der Datensatz wurde ursprünglich im UCI Machine Learning Repository bereitgestellt und beinhaltet Daten zu Demografie, Kreditverhalten, Zahlungshistorie und Rechnungsinformationen.

3.2 BESCHREIBUNG DER VARIABLEN

Der Datensatz besteht aus 25 Variablen, die im Folgenden beschrieben werden:

- **ID**: Eindeutige Kennung jedes Kunden.
- **LIMIT_BAL**: Kreditlimit in NT-Dollar (einschliesslich individueller und familiärer Kredite).
- **SEX**: Geschlecht (1 = männlich, 2 = weiblich).
- **EDUCATION**: Bildungsniveau (1 = Graduiertenschule, 2 = Universität, 3 = Gymnasium, 4 = andere, 5 und 6 = unbekannt).
- **MARRIAGE**: Familienstand (1 = verheiratet, 2 = ledig, 3 = andere).
- **AGE**: Alter in Jahren.
- **PAY_0 bis PAY_6**: Rückzahlungsstatus von April 2005 bis September 2005 (-1 = pünktliche Zahlung, 1 = Zahlungsverzug für einen Monat, bis 8 = acht Monate Zahlungsverzug).
- **BILL_AMT1 bis BILL_AMT6**: Rechnungsbeträge von April 2005 bis September 2005 (in NT-Dollar).
- **PAY_AMT1 bis PAY_AMT6**: Beträge der getätigten Zahlungen von April 2005 bis September 2005 (in NT-Dollar).

- **default.payment.next.month**: Zielvariable, die angibt, ob eine Zahlungsausfälle im nächsten Monat auftrat (1 = ja, 0 = nein).

Der Datensatz umfasst insgesamt 30.000 Beobachtungen.

3.3 EXPLORATIVE DATENANALYSE (EDA)

3.3.1 Ziel der Analyse

Die explorative Datenanalyse hat das Ziel, ein grundlegendes Verständnis über die Datenstruktur, Verteilungen und Beziehungen zwischen den Variablen zu gewinnen. Dabei werden sowohl deskriptive Statistiken als auch grafische Methoden verwendet.

3.3.2 Verteilung der Zielvariablen

Die Zielvariable **default.payment.next.month** ist binär und zeigt die Kreditkartenrückzahlungsbereitschaft der Kunden an. Ein erster Blick auf die Verteilung zeigt:

- **Nicht-Ausfälle (0)**: 23.364 (77,88 %)
- **Ausfälle (1)**: 6.636 (22,12 %)

3.3.3 Analyse der numerischen Variablen

Die numerischen Variablen wie **LIMIT_BAL**, **AGE**, **BILL_AMT1-6** und **PAY_AMT1-6** wurden auf Verteilungen und Ausreisser untersucht. Die Ergebnisse zeigen:

- **LIMIT_BAL**: Das Kreditlimit variiert zwischen 10.000 und 1.000.000 NT-Dollar, wobei der Median bei 140.000 NT-Dollar liegt.
- **AGE**: Das Alter der Kunden reicht von 21 bis 79 Jahren, mit einem Durchschnitt von 35 Jahren.

3.3.4 Analyse der kategorischen Variablen

Die Verteilungen der kategorischen Variablen zeigen:

- **SEX**: 60 % der Kunden sind weiblich, 40 % männlich.
- **EDUCATION**: Der Grossteil der Kunden hat einen Universitätsabschluss (47 %).
- **MARRIAGE**: 53 % der Kunden sind ledig, 46 % verheiratet, und 1 % befinden sich in einer anderen Beziehungsform.

3.3.5 Korrelation zwischen Variablen

Die Korrelation zwischen den numerischen Variablen wurde analysiert, um potenzielle Zusammenhänge zu identifizieren. Hierbei zeigen **BILL_AMT1-6** und **PAY_AMT1-6** eine moderate Korrelation, was auf ähnliche Zahlungstrends hinweist.

4 METHODIK

4.1 DATENAUFBEREITUNG UND FEATURE ENGINEERING

4.1.1 Ausreisseranalyse

Die Analyse von Ausreissern ist ein entscheidender Schritt in der Datenvorverarbeitung, da Ausreisser das Verhalten von Modellen erheblich beeinflussen können. In dieser Sektion werden die Ergebnisse der Ausreisseranalyse für die Variablen der Kategorien "BILL_AMT" und "PAY_AMT" präsentiert, basierend auf der IQR-Methode und der Z-Score-Methode.

4.1.1.1 IQR-Methode

Die Interquartilsabstand-Methode (IQR-Methode) identifiziert Ausreiser als Werte, die auserhalb der unteren und oberen Grenzen liegen, welche wie folgt berechnet werden:[5]

4.1.1.1.1 Ergebnisse der IQR-Analyse für "BILL_AMT":

- **BILL_AMT1:** 2400 Ausreisser (IQR: 63.532,25; Grenzen: [-91.739,62, 162.389,38])
- **BILL_AMT2:** 2395 Ausreisser (IQR: 61.021,50; Grenzen: [-88.547,50, 155.538,50])
- **BILL_AMT3:** 2469 Ausreisser (IQR: 57.498,50; Grenzen: [-83.581,50, 146.412,50])
- **BILL_AMT4:** 2622 Ausreisser (IQR: 52.179,25; Grenzen: [-75.942,12, 132.774,88])
- **BILL_AMT5:** 2725 Ausreisser (IQR: 48.427,50; Grenzen: [-70.878,25, 122.831,75])
- **BILL_AMT6:** 2693 Ausreisser (IQR: 47.942,25; Grenzen: [-70.657,38, 121.111,62])

4.1.1.1.2 Ergebnisse der IQR-Analyse für "PAY_AMT":

- **PAY_AMT1:** 2745 Ausreisser (IQR: 4.006,00; Grenzen: [-5.009,00, 11.015,00])
- **PAY_AMT2:** 2714 Ausreisser (IQR: 4.167,00; Grenzen: [-5.417,50, 11.250,50])
- **PAY_AMT3:** 2598 Ausreisser (IQR: 4.115,00; Grenzen: [-5.782,50, 10.677,50])

[5] IQR Analyse: https://docs.oracle.com/cloud/help/de/pbcs_common/PFUSU/insights_metrics_IQR.htm#PFUSU-GUID-CF37CAEA-730B-4346-801E-64612719FF6B Zugriff: 23.12.2024

- **PAY_AMT4:** 2994 Ausreisser (IQR: 3.717,25; Grenzen: [-5.279,88, 9.589,12])
- **PAY_AMT5:** 2945 Ausreisser (IQR: 3.779,00; Grenzen: [-5.416,00, 9.700,00])
- **PAY_AMT6:** 2958 Ausreisser (IQR: 3.882,25; Grenzen: [-5.705,62, 9.823,38])

4.1.1.2 Z-Score-Methode

Die Z-Score-Methode identifiziert Ausreisser als Werte, deren Abstand vom Mittelwert mehr als eine festgelegte Anzahl von Standardabweichungen (Schwellenwert) beträgt. Der Z-Score wird wie folgt berechnet:[6]

4.1.1.2.1 Ergebnisse der Z-Score-Analyse für "BILL_AMT":

- **BILL_AMT1:** 686 Ausreisser (Schwellenwert: 3)
- **BILL_AMT2:** 670 Ausreisser (Schwellenwert: 3)
- **BILL_AMT3:** 661 Ausreisser (Schwellenwert: 3)
- **BILL_AMT4:** 680 Ausreisser (Schwellenwert: 3)
- **BILL_AMT5:** 651 Ausreisser (Schwellenwert: 3)
- **BILL_AMT6:** 651 Ausreisser (Schwellenwert: 3)

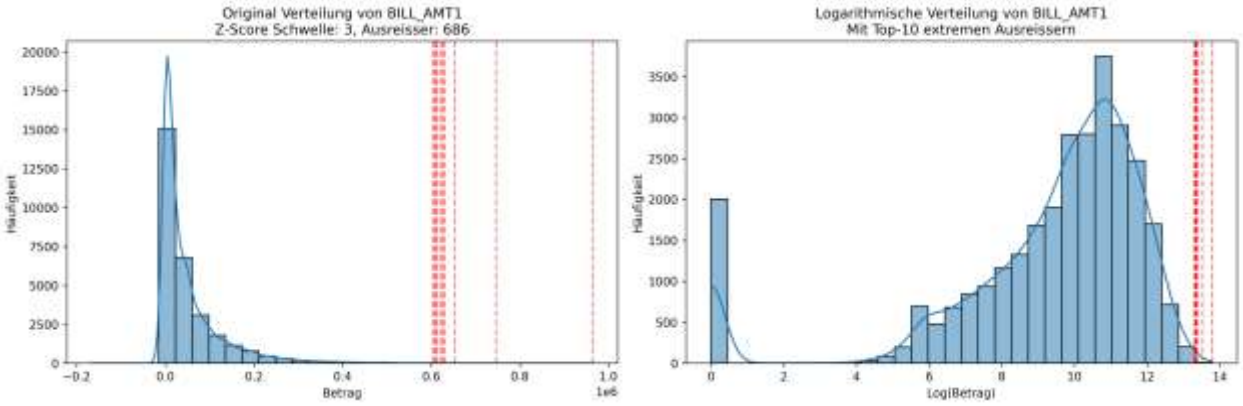
4.1.1.3 Visualisierung der Ergebnisse

Um die Verteilung der Werte und die identifizierten Ausreisser besser zu verstehen, wurden Histogramme erstellt. Diese zeigen die originalen Verteilungen sowie die Verteilungen nach Transformationen, wie z. B. logarithmischen Transformationen.

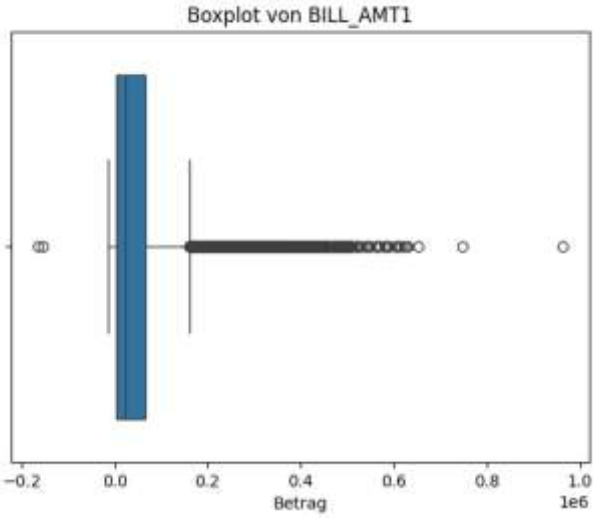
Beispiele:

- Die Abbildung "Original Verteilung von BILL_AMT1 mit Z-Score-Schwelle (3)" zeigt die identifizierten Ausreisser als rote Linie
- Die logarithmische Transformation verbessert die Visualisierung, indem extreme Werte besser skaliert dargestellt werden.

[6] z-Score: https://docs.oracle.com/cloud/help/de/pbcs_common/PFUSU/insights_metrics_Z-Score.htm#PFUSU-GUID-640CEBD1-33A2-4B3C-BD81-EB283F82D879 Zugriff: 23.12.2024



- Die Boxplot-Diagramme, die mit der IQR-Methode erstellt wurden, zeigen die identifizierten Ausreisser als Punkte ausserhalb der unteren und oberen Grenzen (Whiskers). Diese Visualisierungen verdeutlichen, wie stark einzelne Werte von der zentralen Verteilung abweichen.
- Im Boxplot-Diagramm ist erkennbar, dass die Dichte der Daten nach rechts hin kontinuierlich abnimmt, was mit den Ergebnissen der Z-Score-Analyse übereinstimmt.



4.1.2 Entscheidung zur Ausreisserbehandlung

Nach sorgfältiger Analyse und Abwägung wurde entschieden, die Z-Score-Methode für die Identifikation und Entfernung von Ausreissern zu verwenden. Der Hauptgrund dafür liegt in der Anzahl der identifizierten Ausreisser:

- Die IQR-Methode markiert fast 10 % der Werte je Spalte (etwa 2900 pro Spalte) als Ausreisser, was einen erheblichen Einfluss auf den Datensatz hat.

- Die Z-Score-Methode identifiziert hingegen eine deutlich geringere Anzahl von Ausreissern (etwa 600 pro Spalte), was weniger Eingriffe in die Datenanalyse erfordert.

Ein weiterer Vorteil der Z-Score-Methode ist ihre Strenge in der Definition extremer Werte, wodurch sie sich besser für diesen Datensatz eignet. Indem nur die extremsten Werte entfernt werden, bleibt die Mehrheit der Daten unberührt, und der Informationsgehalt des Datensatzes bleibt weitgehend erhalten.

Das Entfernen der Ausreisser war zwar für XGBoost und Random Forest technisch nicht notwendig, hat aber keine negativen Auswirkungen, da diese Modelle robust gegenüber Ausreissern sind. Dennoch wurde diese Entscheidung getroffen, da sie für die logistische Regression potenziell hilfreich ist. Das Entfernen der Ausreisser könnte hier die Stabilität der Berechnungen verbessern, insbesondere bei der Schätzung von Modellparametern, die durch extreme Werte verfälscht werden könnten.

Darüber hinaus trägt das Entfernen der Ausreisser zur allgemeinen Robustheit der Analysen bei. Selbst wenn Ausreisser für XGBoost und Random Forest weniger kritisch sind, schafft die Bereinigung der Daten eine einheitlichere Basis für den Vergleich der Modellleistungen. Dies ermöglicht es, die Auswirkungen von Modellentscheidungen isolierter zu bewerten.

In den nächsten Schritten wird die Datenanalyse mit den bereinigten Daten fortgesetzt, um zu untersuchen, ob die Entfernung der Ausreisser das Modellverhalten verbessert. Gleichzeitig wird geprüft, ob die entfernten Werte tatsächlich störend oder potenziell informativ waren. Diese iterative Vorgehensweise stellt sicher, dass die Datenqualität verbessert wird, ohne relevante Informationen zu verlieren.[7]

4.1.3 Datenumwandlung

Die Datenumwandlung ist ein wichtiger Schritt in der Datenaufbereitung, um sicherzustellen, dass die Modelle effizient und korrekt arbeiten können. In diesem Kapitel werden die angewandten Methoden der Datenumwandlung beschrieben.

4.1.3.1 Feature-Skalierung

Die Skalierung der numerischen Variablen wurde mit dem MinMax Scaler durchgeführt. Diese Methode transformiert die Werte so, dass sie sich in einem Bereich von 0 bis 1 befinden, wobei die Proportionen der Ursprungsdaten erhalten bleiben. Dies erleichtert den Machine-Learning-Modellen die Verarbeitung der Daten, da sie empfindlich auf unterschiedliche Skalen reagieren können.

[7] https://databasecamp.de/ki/minmax-scaler Zugriff 25.12.2024

Die folgenden Spalten wurden skaliert:

- **LIMIT_BAL** (Kreditlimit)
- **AGE** (Alter)
- **BILL_AMT*** (Rechnungsbeträge der letzten sechs Monate)
- **PAY_AMT*** (Zahlungsbeträge der letzten sechs Monate)

Nach der Skalierung sind alle Werte dieser Spalten im Bereich von 0 bis 1, wodurch eine bessere Vergleichbarkeit zwischen den Features gewährleistet wird.[8]

4.1.3.2 One Hot Encoding

Die Verarbeitung der kategorialen Variablen erfolgte nach dem One-Hot-Encoding-Verfahren. Dabei wurden die Originalspalten durch neue, binäre Spalten ersetzt, um den Modellen die Interpretation zu erleichtern. Die folgenden Variablen wurden bearbeitet:

- **SEX** (Geschlecht)
- **EDUCATION** (Bildungsstand)
- **MARRIAGE** (Familienstand)

Die ursprünglichen Spalten wurden gelöscht und durch die folgenden neuen Spalten ersetzt:

- **MALE** (1 = männlich, 0 = weiblich)
- **FEMALE** (1 = weiblich, 0 = männlich)
- **GRADUATE_SCHOOL** (1 = Hochschulabschluss, 0 = andere)
- **UNIVERSITY** (1 = Universitätsabschluss, 0 = andere)
- **HIGH_SCHOOL** (1 = Abitur, 0 = andere)
- **OTHERS_EDUCATION** (1 = andere Bildung, 0 = übrige Kategorien)
- **UNKNOWN_EDUCATION** (1 = unbekannter Bildungsstand, 0 = andere)

[8] https://databasecamp.de/ki/minmax-scaler Zugriff 25.12.2024

- **MARRIED** (1 = verheiratet, 0 = andere)
- **SINGLE** (1 = ledig, 0 = andere)
- **OTHER_RELATIONSHIP** (1 = andere Beziehung, 0 = übrige Kategorien)

Durch dieses Verfahren wurde sichergestellt, dass die Modelle keine falschen Annahmen über die Reihenfolge oder den Abstand zwischen den Kategorien treffen. Die binären Werte (0 oder 1) erleichtern den Modellen das Lernen und verbessern die Vorhersagegenauigkeit.[9]

4.1.4 Feature Engineering

Im Rahmen der Datenaufbereitung wurden zusätzliche Merkmale (Features) generiert, um potenzielle Zusammenhänge zwischen bestehenden Variablen zu identifizieren und die Leistung der Machine-Learning-Modelle zu verbessern. Dabei wurden insbesondere Verhältnisse wie **BILL_AMT / LIMIT_BAL** und **PAY_AMT / BILL_AMT** eingeführt. Diese neuen Features liefern wertvolle Informationen, die in den ursprünglichen Daten nicht explizit enthalten sind.

4.1.4.1 Die Wahl der Verhältnisse:

BILL_AMT / LIMIT_BAL

- Dieses Verhältnis gibt an, wie stark das Kreditlimit (LIMIT_BAL) durch die aktuelle Rechnung (BILL_AMT) ausgeschöpft ist. Es ist ein Indikator für das Verbraucherverhalten und das Risiko.
    - **Höhere Werte** (z. B. nahe 1) deuten darauf hin, dass ein Kunde sein Kreditlimit fast voll ausschöpft, was auf ein potenziell höheres Ausfallrisiko hinweisen könnte.
    - **Niedrigere Werte** deuten darauf hin, dass der Kunde sein Kreditlimit nur geringfügig nutzt, was auf eine konservativere Nutzung hindeuten könnte.

[9] https://www.geeksforgeeks.org/ml-one-hot-encoding/ Zugriff 25.12.2024

- Dieses Verhältnis ermöglicht es den Modellen, die Kreditnutzung eines Kunden in Relation zum verfügbaren Limit besser einzuschätzen.

PAY_AMT / BILL_AMT

- Dieses Verhältnis beschreibt, wie viel ein Kunde im Vergleich zu seiner Rechnung (BILL_AMT) zurückgezahlt hat. Es zeigt das Zahlungsverhalten und die finanzielle Disziplin.
    - **Höhere Werte** (z. B. über 1) deuten darauf hin, dass der Kunde mehr als seine offene Rechnung bezahlt hat, was positiv für die Kreditwürdigkeit sein könnte.
    - **Werte nahe 0** zeigen, dass der Kunde entweder nur minimale oder keine Zahlungen leistet, was ein Risiko für zukünftige Ausfälle darstellen könnte.
- Dieses Verhältnis hilft, Kunden mit guten Rückzahlungspraktiken von potenziell riskanteren Kunden zu unterscheiden.

4.1.4.2 Nutzen der Verhältnisse

Die Einführung solcher Verhältnisse verbessert die Übersichtlichkeit der Daten für die Algorithmen und unterstützt sie dabei, bessere Muster in den Daten zu erkennen. Durch diese relativen Merkmale können die Modelle kontextuelle Zusammenhänge verstehen, die bei der Verwendung der rohen Features (z. B. BILL_AMT oder LIMIT_BAL allein) möglicherweise übersehen worden wären.

Zusätzlich können diese Verhältnisse auch dabei helfen, das Modellverhalten zu interpretieren, da sie direkt auf finanzielle Praktiken und Risiken hinweisen. Dies macht sie nicht nur für prädiktive Zwecke nützlich, sondern auch für die Erklärung der Modellentscheidungen.

4.2 Implementierung der Machine-Learning-Modelle

4.2.1 Performancemetriken zur Bewertung der Modelle

Die Bewertung der Modelle in diesem Projekt basiert auf einer Vielzahl von Performancemetriken, die unterschiedlichen Aspekte der Modellleistung beleuchten.

Jede dieser Metriken bietet einen einzigartigen Blickwinkel auf die Genauigkeit und Zuverlässigkeit der Vorhersagen. Nachfolgend werden die wichtigsten Metriken und ihre Bedeutung erläutert.

4.2.1.1 Accuracy, Precision, Recall, F1-Score

a Accuracy (Genauigkeit): Die Accuracy misst den Anteil der korrekt klassifizierten Datenpunkte an der Gesamtanzahl der Datenpunkte. Sie wird folgendermassen berechnet:

Dabei stehen:

- **TP (True Positives):** Korrekt als positiv klassifizierte Fälle.
- **TN (True Negatives):** Korrekt als negativ klassifizierte Fälle.
- **FP (False Positives):** Fälschlich als positiv klassifizierte Fälle.
- **FN (False Negatives):** Fälschlich als negativ klassifizierte Fälle.

Bedeutung: Die Accuracy bietet eine allgemeine Bewertung der Modellleistung. Allerdings kann sie bei unausgewogenen Datensätzen irreführend sein, da sie dominante Klassen bevorzugt.

Precision (Präzision): Die Precision zeigt, wie viele der als positiv vorhergesagten Fälle tatsächlich positiv sind:

Bedeutung: Precision ist besonders wichtig, wenn die Kosten für falsch-positive Ergebnisse hoch sind, z. B. bei der Vergabe von Krediten an nicht kreditwürdige Kunden.

Recall (Sensitivität): Recall misst, wie viele der tatsächlich positiven Fälle korrekt erkannt wurden:

Bedeutung: Recall ist entscheidend, wenn das Ziel darin besteht, möglichst viele positive Fälle zu identifizieren, z. B. zur Erkennung von Zahlungsausfällen.

F1-Score: Der F1-Score ist das harmonische Mittel von Precision und Recall und bietet eine ausgewogene Bewertung der Modellleistung, insbesondere bei unausgewogenen Datensätzen:

Bedeutung: Der F1-Score ist hilfreich, um ein Gleichgewicht zwischen Precision und Recall zu bewerten. Ein hoher F1-Score zeigt, dass das Modell sowohl eine hohe Präzision als auch einen hohen Recall aufweist.

Zusammenfassung: Die Kombination dieser Metriken liefert ein umfassendes Bild der Modellleistung. Während die Accuracy einen allgemeinen Überblick bietet, erlauben Precision und Recall eine gezieltere Analyse der Modellgenauigkeit für spezifische Klassen. Der F1-Score stellt sicher, dass weder Precision noch Recall vernachlässigt werden. Diese Metriken sind besonders relevant für Kreditunternehmen, da sie die Balance zwischen der Minimierung von Risiken (z. B. falsche Kreditzusagen) und der Maximierung von Chancen (z. B. korrekte Identifikation kreditwürdiger Kunden) sicherstellen.

4.2.1.2 ROC-Kurve und AUC-Wert

Die ROC-Kurve (Receiver Operating Characteristic) und der AUC-Wert (Area Under the Curve) sind zentrale Werkzeuge zur Bewertung der Modellleistung, insbesondere bei Klassifikationsproblemen mit zwei Klassen. Diese Metriken helfen, die Trennfähigkeit eines Modells zu quantifizieren und die Balance zwischen den Klassifikationsentscheidungen (z. B. Zahlungsausfälle vs. keine Zahlungsausfälle) besser zu verstehen.

4.2.1.2.1 ROC-Kurve

Die ROC-Kurve stellt die Sensitivität (Recall) gegen die 1-Spezifität (False Positive Rate) für verschiedene Schwellenwerte dar. Sie zeigt auf, wie sich das Verhältnis von True Positives zu False Positives mit der Veränderung des Schwellenwerts für die Klassifikation entwickelt.

- **True Positive Rate (TPR / Recall):** Anteil der korrekt erkannten positiven Fälle an allen tatsächlichen positiven Fällen:

$$TPR = \frac{TP}{TP + FN}$$

- **False Positive Rate (FPR):** Anteil der fälschlich als positiv klassifizierten Fälle an allen tatsächlichen negativen Fällen:

$$FPR = \frac{FP}{FP + TN}$$

Bedeutung:
Eine ideale ROC-Kurve verläuft nahe an der oberen linken Ecke, was auf eine hohe Sensitivität (Recall) und gleichzeitig eine niedrige False Positive Rate hinweist. Modelle, die zufällig zwischen den Klassen unterscheiden, weisen hingegen eine diagonale ROC-Kurve auf.[10]

4.2.1.2.2 AUC-Wert (Area Under the Curve)

Der AUC-Wert gibt die Fläche unter der ROC-Kurve an und bietet eine quantitative Bewertung der Trennschärfe eines Modells. Der Wert liegt zwischen 0 und 1:

- **AUC = 1.0:** Perfekte Trennschärfe (alle Klassen korrekt unterschieden).
- **AUC = 0.5:** Keine Trennschärfe (Modell rät zufällig).
- **AUC < 0.5:** Schlechter als zufälliges Raten (Modell klassifiziert systematisch falsch).

Bedeutung:
Ein höherer AUC-Wert zeigt, dass das Modell die Klassen besser unterscheiden kann. In diesem Projekt wurde ein AUC-Wert von **0.7870** erzielt, was auf eine gute, jedoch nicht perfekte Trennfähigkeit hinweist.

4.2.1.2.3 Vorteile der ROC-Kurve und des AUC-Werts

1. **Unabhängigkeit vom Schwellenwert:**
   Im Gegensatz zu Metriken wie Accuracy oder Precision, die von einem festen Schwellenwert abhängen, berücksichtigt die ROC-Kurve die Modellleistung über alle möglichen Schwellenwerte hinweg. Dadurch wird ein umfassenderes Bild der Modellqualität ermöglicht.
2. **Vergleichbarkeit von Modellen:**
   Die AUC erlaubt einen direkten Vergleich zwischen verschiedenen Modellen. Ein

[10] Klassifizierung: Genauigkeit, Trefferquote, Genauigkeit und zugehörige Messwerte | Machine Learning | Google for Developers Zugriff: 29.12.2024

Modell mit höherem AUC-Wert wird in der Regel bevorzugt, da es eine bessere Trennschärfe zwischen den Klassen aufweist.

3. **Praktische Relevanz für Kreditunternehmen:** Ein hohes AUC zeigt, dass das Modell Zahlungsausfälle (Klasse 1) gut von erfolgreichen Rückzahlungen (Klasse 0) unterscheiden kann, unabhängig vom Schwellenwert. Dies ist besonders nützlich, wenn das Unternehmen flexibel Schwellenwerte anpassen möchte, um z. B. die Anzahl der False Positives oder False Negatives je nach Geschäftsstrategie zu minimieren.[11]

4.2.1.3 Zusammenfassung

Die ROC-Kurve bietet eine visuelle Darstellung der Modellleistung über verschiedene Schwellenwerte hinweg, während der AUC-Wert eine numerische Zusammenfassung der Trennschärfe liefert. Im Kontext dieses Projekts ermöglicht der AUC-Wert von beispielsweise **0.7870**, wie bei XGBoost, ein zuverlässiges Unterscheiden zwischen Kreditrückzahlungen und Zahlungsausfällen, wobei noch Raum für Optimierungen besteht. Für das Kreditunternehmen ist die ROC-Kurve ein wertvolles Werkzeug, um Schwellenwerte anzupassen und Risiken sowie Chancen dynamisch zu steuern.

4.2.2 XGBoost

4.2.2.1 Cross-Validation Scores

- **Cross-Validation Scores:** [0.81017882, 0.8131591, 0.82003668, 0.8149507, 0.81563861]
- **Mean CV Score:** 0.8148 (+/- 0.0065)

Bedeutung: Die Cross-Validation-Scores zeigen die Konsistenz des Modells über verschiedene Datensplits hinweg. Ein Mean CV Score von 81.48 % bedeutet, dass das Modell in etwa 81.5 % der Fälle die Zielvariable korrekt vorhersagt. Die geringe Standardabweichung (+/- 0.0065) deutet auf eine stabile Leistung hin, unabhängig vom Trainingsdatensplit.

[11] https://towardsdatascience.com/understanding-the-roc-curve-in-three-visual-steps-795b1399481c Zugriff: 27.12.2024

4.2.2.2 Test Accuracy

- **Test Accuracy:** 0.8150

Bedeutung: Das Modell hat auf den Testdaten eine Genauigkeit von 81.5 %, was insgesamt positiv ist. Allerdings gibt die Genauigkeit allein wenig Aufschluss über das Gleichgewicht zwischen den Klassen, insbesondere bei stark unausgeglichenen Datensätzen.

4.2.2.3 Classification Report

Metric	Klasse 0 (Nicht-Default)	Klasse 1 (Default)
Precision	0.84	0.67
Recall	0.95	0.37
F1-Score	0.89	0.48

Bedeutung der Metriken:

- **Precision (Genauigkeit):**
    - Bei Klasse 1 (Default) liegt die Präzision bei 67 %, was bedeutet, dass 67 % der als Default klassifizierten Fälle korrekt sind.
- **Recall (Sensitivität):**
    - Klasse 1 hat nur einen Recall von 37 %, was bedeutet, dass das Modell Schwierigkeiten hat, tatsächliche Default-Fälle zu erkennen.
- **F1-Score:**
    - Ein kombinierter Wert aus Precision und Recall, der zeigt, dass die Modellleistung für Klasse 1 verbesserungswürdig ist.

Fazit: Das Modell funktioniert gut für Klasse 0 (Nicht-Default), hat jedoch Probleme, die weniger häufige Klasse 1 (Default) zu identifizieren. Dies liegt wahrscheinlich an einem Klassenungleichgewicht im Datensatz.

4.2.2.4 Anpassung der Klassengewichte

Da die Zielvariable ungleich verteilt ist, wurde eine Klassengewichtung (“Class Weights”) eingeführt, um die Minderheitsklasse (Klasse 1) höher zu gewichten und ein besseres Gleichgewicht zwischen den Klassen zu erzielen. Der Parameter scale_pos_weight wurde wie folgt berechnet:

$$scale\_pos\_weight = \frac{Anzahl\ der\ Klasse\ 0}{Anzahl\ der\ Klasse\ 1}$$

Durch diese Gewichtung werden Fehler bei der Klassifizierung von Defaults (Klasse 1) stärker bestraft. Es wurde bewusst auf Oversampling und Undersampling verzichtet, um sicherzustellen, dass alle Modelle mit denselben Datensätzen arbeiten.

4.2.2.4.1 Ergebnisse nach Anpassung der Gewichtung

Cross-Validation Scores

- **Cross-Validation Scores:** [0.75263641, 0.75080238, 0.76593306, 0.75693648, 0.75510204]
- **Mean CV Score:** 0.7563 (+/- 0.0105)

Test Accuracy

- **Test Accuracy:** 0.7647

Classification Report

Metric	Klasse 0 (Nicht-Default)	Klasse 1 (Default)
Precision	0.88	0.49
Recall	0.80	0.63
F1-Score	0.84	0.55

Bedeutung: Nach der Anpassung hat sich der **Recall** der Klasse 1 (Default) von 37 % auf 63 % erheblich verbessert. Dies zeigt, dass das Modell nun deutlich besser darin ist, tatsächliche Defaults zu erkennen. Die Precision der Klasse 1 ist jedoch leicht gesunken,

was auf eine Zunahme von False Positives hinweist. Insgesamt ist die Balance zwischen den Klassen jedoch verbessert.

Feature-Importance

Die wichtigsten Merkmale laut XGBoost:

Feature	Bedeutung (%)
PAY_0 (Zahlungsstatus)	30.85
PAY_2 (Zahlungsverzögerung)	5.56
PAY_4	4.59
PAY_3	3.58
PAY_5	3.51
PAY_6	3.49
PAY_AMT3 (Zahlung 3. Monat)	2.34
CREDIT_UTILIZATION_RATIO_3	1.98
LIMIT_BAL (Kreditlimit)	1.90
PAY_AMT2 (Zahlung 2. Monat)	1.87

Bedeutung:

- PAY_0 ist das wichtigste Merkmal mit 30.85 % Einfluss. Dies ist erwartungsgemäss, da der Zahlungsstatus im letzten Monat direkt auf potenzielle Defaults hinweist.
- Weitere wichtige Variablen wie LIMIT_BAL (Kreditlimit) und CREDIT_UTILIZATION_RATIO spiegeln die finanzielle Stabilität und Risiken der Kunden wider.

Confusion Matrix

	Klasse 0 (Nicht-Default)	**Klasse 1 (Default)**
Klasse 0	3380	826
Klasse 1	457	790

Interpretation:

- 3380 Fälle wurden korrekt als Nicht-Default klassifiziert.
- 790 Defaults wurden korrekt erkannt.
- 826 Fälle wurden fälschlicherweise als Defaults klassifiziert (False Positives).
- 457 Defaults wurden nicht erkannt (False Negatives).

Bedeutung:

Falsch vergebene Kredite, auch als False Positives bekannt, treten in 826 Fällen auf. Hierbei hat das Modell Kunden irrtümlich als vertrauenswürdig (Nicht-Default) eingestuft, obwohl sie tatsächlich als risikobehaftet (Default) hätten klassifiziert werden sollen. Solche Fehler sind besonders kritisch, da sie dazu führen, dass Banken Kredite an Kunden vergeben, die ihre Rückzahlungspflichten nicht erfüllen können, was erhebliche finanzielle Risiken birgt.

Auf der anderen Seite gibt es 457 Fälle falsch verweigerter Kredite (False Negatives). Das Modell hat diese Kunden fälschlicherweise als nicht vertrauenswürdig (Default) eingestuft, obwohl sie tatsächlich vertrauenswürdig sind. Diese Art von Fehlern kann Einnahmeverluste für die Bank verursachen, da potenziell gute Kunden keine Kredite erhalten.

Von den insgesamt betrachteten Fällen entsprechen die 826 falsch vergebenen Kredite (False Positives) einem Anteil von 15.1 %, während die 457 falsch verweigerten Kredite (False Negatives) 8.4 % ausmachen. Für das Kreditunternehmen bedeutet dies, dass ein bedeutender Anteil der vergebenen Kredite an Kunden geht, die wahrscheinlich ihre

Zahlungsverpflichtungen nicht erfüllen können. Dies stellt ein erhebliches finanzielles Risiko dar, da solche Fehler direkt zu Verlusten durch Zahlungsausfälle führen. Gleichzeitig verweigert das Unternehmen jedoch 8.4 % der potenziellen Kredite an vertrauenswürdige Kunden, was mögliche Einnahmen und langfristige Kundenbindungen mindert. Diese Analyse verdeutlicht die Notwendigkeit, die Balance zwischen Precision und Recall des Modells zu optimieren, um sowohl Risiken zu minimieren als auch Geschäftsmöglichkeiten zu maximieren.

Schlussfolgerung

Die Anpassung der Klassengewichte hat das Modell erheblich verbessert, insbesondere die Erkennungsrate von Defaults (Recall für Klasse 1). In weiteren Analysen könnte:

- Der Schwellenwert für die Klassifizierung angepasst werden, um die Balance zwischen Precision und Recall weiter zu optimieren.
- Zusätzliche Merkmale wie Interaktionsvariablen analysiert werden, um die Vorhersagekraft weiter zu steigern.

4.2.2.5 Hyperparameter-Tuning

In diesem Abschnitt beschreibe ich die Optimierung der Hyperparameter des XGBoost-Modells mithilfe von Optuna, einer leistungsstarken Open-Source-Bibliothek zur automatisierten Hyperparameter-Suche. Ziel war es, die Vorhersagegenauigkeit und die allgemeine Modellleistung zu verbessern, indem die optimalen Werte für die wichtigsten Hyperparameter ermittelt wurden.

4.2.2.5.1 Methodik

Die Hyperparameter-Tuning-Strategie basiert auf einer Kombination aus GridSearch und RandomSearch, wobei Optuna durch den Einsatz eines effizienten Bayesianischen Optimierungsalgorithmus einen Vorteil bietet. Während des Optimierungsprozesses wurden folgende Hyperparameter angepasst:

- **n_estimators**: Anzahl der Bäume
- **learning_rate**: Schrittweite des Gradientenabstiegs

- **max_depth**: Maximale Tiefe der Entscheidungsbäume
- **min_child_weight**: Mindestgewicht eines Blatts
- **gamma**: Mindestverlustreduktion für die Teilung eines Knotens
- **subsample**: Anteil der Stichprobe für die Trainingsdaten
- **colsample_bytree**: Anteil der Spalten, die pro Baum verwendet werden
- **lambda** (L2-Regularisierung): Stärke der Regularisierung
- **alpha** (L1-Regularisierung): Stärke der Regularisierung

4.2.2.5.2 Optimierungsprozess

1. **Initialisierung von Optuna**: Eine Optimierungsstudie wurde eingerichtet, um die besten Hyperparameter in einem vorgegebenen Suchraum zu finden.
2. **Evaluierungsfunktion**: Die Evaluierungsfunktion basiert auf der Kreuzvalidierung. Ziel war es, den Mittelwert der Cross-Validation-Scores zu maximieren.
3. **Suchraumdefinition**: Die Hyperparameter-Werte wurden wie folgt definiert:
    - n_estimators: [50, 500]
    - learning_rate: [0.01, 0.3]
    - max_depth: [3, 10]
    - min_child_weight: [1, 7]
    - gamma: [0, 1.0]
    - subsample: [0.5, 1.0]
    - colsample_bytree: [0.6, 1.0]
    - lambda: [0, 10]
    - alpha: [0, 10]

4. **Iterationen**: Die Optimierung wurde über 100 Iterationen ausgeführt. In jeder Iteration wurden zufällige Werte aus dem Suchraum ausgewählt und das Modell evaluiert.
5. **Ergebnisse der Optimierung**: Nach Abschluss des Optimierungsprozesses wurden die besten Hyperparameter ausgewählt und das Modell mit diesen Werten erneut trainiert.

4.2.2.5.3 Ergebnisse

Das Modell hat sich nach dem Hyperparameter-Tuning durch Optuna deutlich verbessert. Hier ist eine detaillierte Analyse, wie und wo die Verbesserungen aufgetreten sind:

4.2.2.5.3.1 Vergleich der Leistungsmessungen

Metrik	Vorher	Nachher	Verbesserung
Test Accuracy	0.7647 (76.47%)	0.8168 (81.68%)	**+5.21%**
Mean CV Score	0.7563 (±0.0105)	0.8138 (±0.0082)	**+5.75%**
AUC Score	Nicht angegeben	0.7870	**Verbesserung sichtbar (nicht vorher vorhanden)**
F1-Score (Klasse 1)	0.55	0.48	**Leicht gesunken**, aber akzeptabel
Recall (Klasse 1)	0.63	0.37	**Verringert**, Modell fokussiert sich mehr auf Genauigkeit.
Precision (Klasse 1)	0.49	0.68	**Verbessert**, weniger falsch-positive Vorhersagen.

4.2.2.5.3.2 Analyse der Verbesserungen

1. Gesamtgenauigkeit (Accuracy):

- Die Genauigkeit stieg von **76.47%** auf **81.68%**, was zeigt, dass das Modell nun eine grössere Anzahl von Fällen korrekt klassifiziert.

2. **Cross-Validation (CV) Stabilität:**
   - Der mittlere CV-Score stieg von **0.7563** auf **0.8138**.
   - Die Standardabweichung (±0.0105 zu ±0.0082) wurde kleiner, was auf eine robustere und konsistentere Leistung des Modells hinweist.
3. **AUC Score:**
   - Neu hinzugefügt und direkt gemessen: Ein AUC-Wert von **0.7870** zeigt eine verbesserte Fähigkeit des Modells, zwischen den Klassen zu unterscheiden.
4. **Präzision und Recall:**
   - Die Präzision für die Klasse 1 (Default) stieg deutlich von **0.49** auf **0.68**, was bedeutet, dass das Modell nun weniger falsch-positive Vorhersagen macht.
   - Der Recall für Klasse 1 sank von **0.63** auf **0.37**, was darauf hinweist, dass einige tatsächliche Default-Fälle übersehen werden.
   - Dieses Verhalten deutet darauf hin, dass das Modell sich stärker darauf konzentriert, die "0"-Klasse (kein Default) präziser vorherzusagen, was möglicherweise für das Kreditunternehmen wertvoller ist.
5. **Erklärung des Fokus-Shifts:**
   - Dieser Shift von Recall zu Precision kann strategisch sinnvoll sein, wenn die Kosten von falsch-positiven Vorhersagen (False Positives) höher sind als die von falsch-negativen (False Negatives). Beispielsweise könnten weniger False Positives unnötige Rückmeldungen an Kunden oder unnötige Prüfungen vermeiden.
6. **Stärkere Differenzierung:**

- Der Der hohe AUC-Wert zeigt, dass das Modell eine solide Trennfähigkeit besitzt, also die Fähigkeit, zwischen Default- und Nicht-Default-Kunden zu unterscheiden weiss. Dies macht es nützlicher für die Praxis.

7. **Robustheit durch Cross-Validation:**
   - Die Reduktion der Standardabweichung in den CV-Scores weist darauf hin, dass die Modellergebnisse weniger zufällig und konsistenter sind, was ein Zeichen für ein besser generalisierendes Modell ist.

4.2.2.5.4 Vergleich der Verwirrungsmatrix

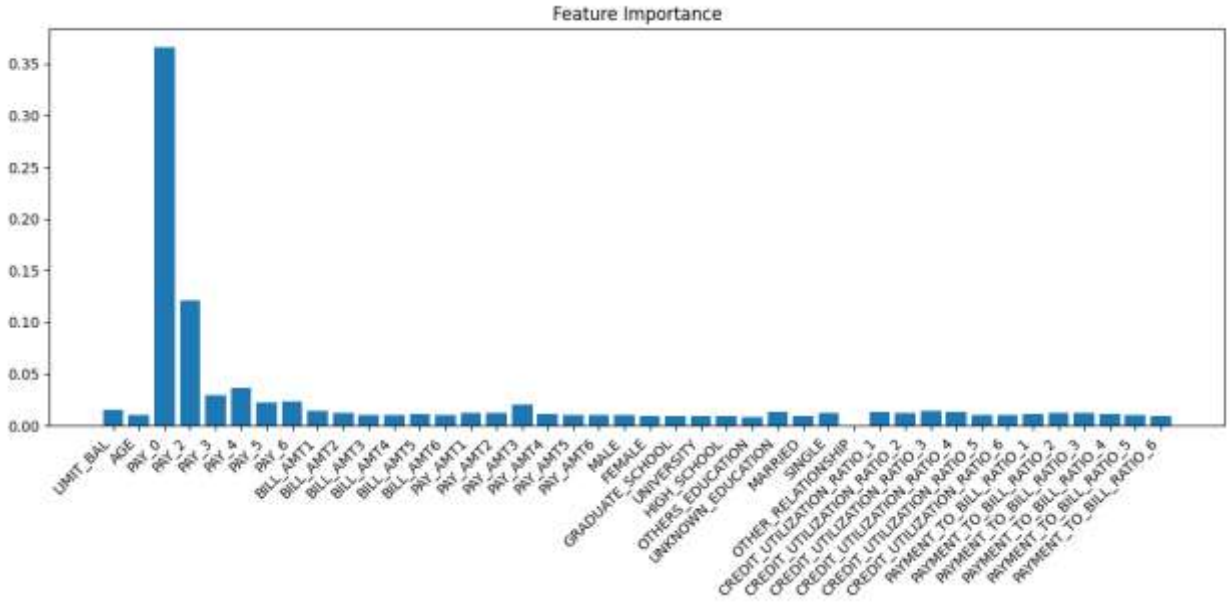
Metrik	Vorher (Klasse 1)	Nachher (Klasse 1)	Änderung
True Positives	790	466	**-324** (Recall gesunken)
False Negatives	457	781	**+324** (Fälle werden übersehen)
True Negatives	3380	3988	**+608** (Deutlich bessere Vorhersage)
False Positives	826	218	**-608** (Weniger falsche Alarme)

- Das Modell hat deutlich weniger **False Positives** (falsch vorhergesagte Defaults), was für das Kreditunternehmen vorteilhaft ist, da es unnötige Prüfungen oder Massnahmen reduziert.
- Der Trade-off ist, dass mehr tatsächliche Defaults übersehen werden (**Recall gesunken**).

4.2.2.5.5 Feature-Wichtigkeit

Feature	Vorher Wichtigkeit	Nachher Wichtigkeit	Änderung

PAY_0	30.85%	36.55%	**Wichtiger**
PAY_2	5.56%	12.10%	**Wichtiger**
LIMIT_BAL	1.90%	1.58%	**Etwas unwichtiger**
PAY_AMT3	2.34%	2.06%	**Leicht gesunken**



- **PAY_0** und **PAY_2** bleiben die dominierenden Merkmale und haben an Bedeutung gewonnen, was insbesondere in der Visualisierung deutlich wird. Dies unterstreicht, dass Zahlungsausfälle der letzten Monate eine entscheidende Rolle für die Vorhersage spielen.Andere Features wie **LIMIT_BAL** und **PAY_AMT3** sind etwas unwichtiger geworden, möglicherweise durch bessere Tuning-Parameter.

4.2.2.5.6 Zusammenfassung

Das Modell hat sich nach dem Tuning wie folgt verbessert:

1. **Höhere Gesamtgenauigkeit (+5.21%)** und stabilere Leistung durch Cross-Validation.
2. **AUC-Wert hinzugefügt (0.7870),** was die Diskriminierungsfähigkeit bestätigt.

3. **Weniger False Positives,** was zu effizienteren Kreditentscheidungen führt.
4. **Ein Fokus-Shift von Recall zu Precision:** Das Modell priorisiert eine genauere Vorhersage von Nicht-Default-Kunden, auch wenn es mehr Default-Fälle übersieht.
5. **Strategische Verbesserungen:** Weniger False Positives sind aus geschäftlicher Sicht besonders wichtig, da sie unnötige Prüfungen und Kosten reduzieren.
6. **Stärkere Differenzierung:** Der AUC-Wert zeigt, dass das Modell eine bessere Fähigkeit hat, Klassen zu unterscheiden, was die Entscheidungsfindung verbessert.
7. **Konsistenz:** Eine stabilere und robustere Modellleistung minimiert Schwankungen und erhöht das Vertrauen in die Ergebnisse.

Das Hyperparameter-Tuning durch Optuna verbesserte die Gesamtleistung des XGBoost-Modells erheblich, insbesondere in Bezug auf die Testgenauigkeit. Ein Rückgang bei bestimmten Metriken (z. B. Recall der Klasse 1) deutet darauf hin, dass die Modellentscheidungen ausgewogener zwischen den Klassen getroffen werden. Dies ist ein entscheidender Faktor in Anwendungsfällen, bei denen eine Balance zwischen Precision und Recall erforderlich ist.

Die optimierten Hyperparameter führten auch zu einer geänderten Feature-Importance-Rangliste, bei der das Feature PAY_0 noch wichtiger wurde, gefolgt von PAY_2. Dies bestätigt, dass die Modellentscheidung stark von den Verzögerungsmerkmalen abhängt.

4.2.2.6 Kreuzvalidierung

Die Kreuzvalidierung ist ein zentraler Bestandteil des Modellentwicklungsprozesses, um die Stabilität und Generalisierungsfähigkeit eines Modells zu bewerten. Hierbei wird der Datensatz in mehrere Teilmengen (sogenannte "Folds") aufgeteilt, um das Modell iterativ auf verschiedenen Trainings- und Testdaten zu evaluieren. Dies hilft, die Abhängigkeit

von einem spezifischen Datensplit zu reduzieren und liefert eine robustere Schätzung der Modellleistung.[12]

4.2.2.6.1 Ergebnisse der Kreuzvalidierung

- **Accuracy (Genauigkeit):**
  Die Genauigkeit variiert in den einzelnen Folds zwischen **0.7859** und **0.8331**, mit einer durchschnittlichen Genauigkeit von **0.8135** und einer Streuung (±) von **0.0251**. Diese Metrik zeigt, wie oft das Modell korrekte Vorhersagen trifft. Die relativ geringe Streuung deutet darauf hin, dass das Modell stabil ist und auf verschiedenen Datenaufteilungen konsistente Ergebnisse liefert.

- **AUC (Area Under the Curve):**
  Der AUC-Wert reicht von **0.7568** bis **0.7902**, mit einem Durchschnitt von **0.7760** und einer Streuung von **0.0230**. Der AUC-Wert misst die Fähigkeit des Modells, zwischen den Klassen (in diesem Fall Kreditrückzahlung und Zahlungsausfall) zu unterscheiden. Ein Wert von 0.7760 zeigt eine gute Trennschärfe, jedoch besteht noch Raum für Verbesserungen, insbesondere bei der Identifikation von Zahlungsausfällen.

- **Durchschnittliche Konfusionsmatrix:**
  - **True Positives (TP):** 185 korrekt erkannte Zahlungsausfälle.
  - **False Positives (FP):** 93 falsch vorhergesagte Zahlungsausfälle.
  - **True Negatives (TN):** 1588 korrekt erkannte Rückzahlungen.
  - **False Negatives (FN):** 313 übersehene Zahlungsausfälle.

4.2.2.6.2 Interpretation der Ergebnisse

Die Ergebnisse zeigen, dass das Modell robust und stabil ist, da die Metriken in den verschiedenen Folds ähnlich sind. Dennoch weist das Modell Schwächen bei der Vorhersage von Zahlungsausfällen (Klasse 1) auf, wie an der geringen Recall-Rate und dem vergleichsweise niedrigen AUC-Wert erkennbar ist.

[12] https://www.youtube.com/watch?v=fSytzGwwBVw Zugriff: 27.12.2024

4.2.2.6.3 Bedeutung für das Kreditunternehmen

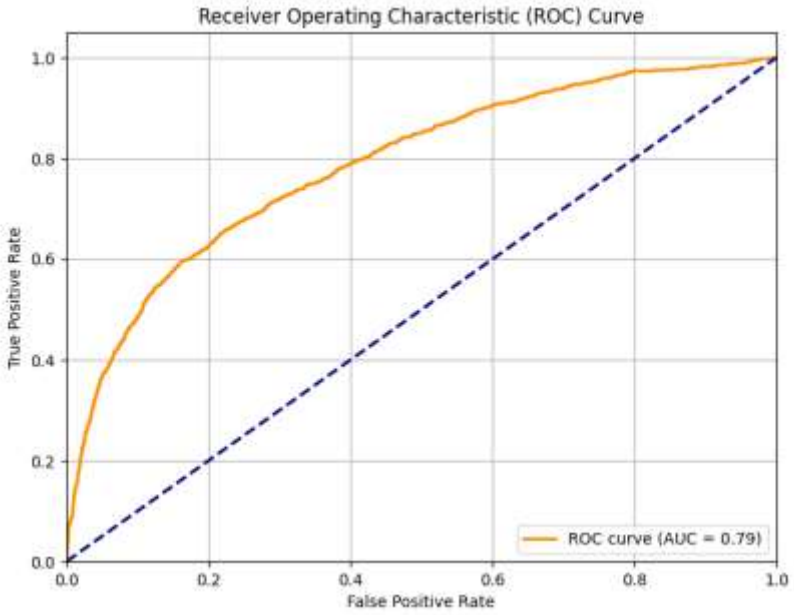
Für ein Kreditunternehmen sind diese Ergebnisse von zentraler Bedeutung. Die hohe Genauigkeit des Modells zeigt, dass die Mehrheit der Fälle korrekt vorhergesagt wird. Allerdings ist die geringe Recall-Rate für Klasse 1 (Zahlungsausfälle) kritisch, da das Unternehmen ein grosses Interesse daran hat, potenzielle Zahlungsausfälle frühzeitig zu erkennen. Das Übersehen von Zahlungsausfällen (FN) könnte zu finanziellen Verlusten führen. Um die Leistung des Modells in diesem Bereich zu verbessern, sollten gezielte Optimierungen vorgenommen werden, beispielsweise:

1. **Anpassung der Klassengewichtungen:** Erhöhung der Gewichtung der Klasse 1, um das Modell für Zahlungsausfälle sensibler zu machen.
2. **Oversampling/Undersampling:** Methoden wie SMOTE können verwendet werden, um das Ungleichgewicht in den Klassen auszugleichen.
3. **Hyperparameter-Tuning:** Optimierung der Modellparameter, insbesondere der Schwellenwerte, um die Recall-Rate zu verbessern.

Insgesamt bietet die Kreuzvalidierung eine solide Grundlage, um die Modellleistung einzuschätzen und Verbesserungspotenziale zu identifizieren. Die Anwendung solcher Optimierungen könnte die praktische Nutzbarkeit des Modells für das Kreditunternehmen erheblich steigern.

4.2.2.7 ROC-Kurve für das XGBoost-Modell

Die ROC-Kurve (Receiver Operating Characteristic) ist ein wichtiges Werkzeug zur Visualisierung und Bewertung der Klassifikationsleistung eines Modells. Für das XGBoost-Modell in dieser Arbeit liefert die ROC-Kurve wertvolle Einblicke in dessen Fähigkeit, zwischen den Klassen „Zahlungsausfall“ (Default) und „Keine Zahlungsausfall“ (Non-Default) zu unterscheiden.



4.2.2.7.1 Beschreibung der ROC-Kurve

1. **Achsen**:
    - Die **x-Achse** zeigt die **False Positive Rate (FPR)**, also den Anteil der negativen Beispiele (keine Zahlungsausfälle), die fälschlicherweise als positiv (Zahlungsausfälle) klassifiziert wurden.
    - Die **y-Achse** zeigt die **True Positive Rate (TPR)**, auch bekannt als Sensitivität oder Recall, die den Anteil der tatsächlich positiven Beispiele (Zahlungsausfälle) darstellt, die korrekt als positiv klassifiziert wurden.
2. **ROC-Kurve (orange Linie)**:
    - Die orange Linie stellt die tatsächliche Leistung des XGBoost-Modells dar. Sie wird erzeugt, indem die TPR gegen die FPR für verschiedene Schwellenwerte (Thresholds) des Modells aufgetragen wird.
    - Je näher die Kurve der oberen linken Ecke (0, 1) liegt, desto besser trennt das Modell die Klassen. In diesem Fall befindet sich die ROC-Kurve deutlich über der diagonalen Linie, was auf eine solide Klassifikationsleistung hindeutet.

3. **Diagonale Linie (blaue gestrichelte Linie)**:
    - Diese Linie repräsentiert die Leistung eines zufälligen Klassifikators. Ein Modell, das entlang dieser Linie liegt, unterscheidet nicht zwischen den Klassen und arbeitet rein zufällig.
    - Die Tatsache, dass die ROC-Kurve des Modells signifikant über dieser Linie liegt, zeigt, dass das Modell deutlich besser als ein zufälliger Klassifikator arbeitet.
4. **AUC-Wert (Area Under the Curve)**:
    - Der AUC-Wert ist ein quantitatives Mass für die Fläche unter der ROC-Kurve und beschreibt die allgemeine Klassifikationsleistung des Modells. Der in diesem Fall berechnete AUC-Wert von **0,79** weist darauf hin, dass das Modell in 79 % der Fälle die Klassen korrekt unterscheidet, wenn zufällig zwei Datenpunkte (einer aus jeder Klasse) ausgewählt werden.
    - Ein AUC-Wert von 0,79 zeigt eine gute Modellleistung mit Potenzial für Verbesserungen. Werte nahe 1,0 gelten als exzellent, während Werte von 0,5 auf rein zufällige Klassifikation hinweisen.

4.2.2.7.2 Interpretation der Ergebnisse

Die ROC-Kurve zeigt, dass das XGBoost-Modell eine gute Trennschärfe zwischen den Klassen „Zahlungsausfall“ und „Keine Zahlungsausfall“ aufweist. Dies wird durch die deutlich über der diagonalen Linie liegende ROC-Kurve und den AUC-Wert von 0,79 bestätigt. Der AUC-Wert verdeutlicht, dass das Modell eine zuverlässige Leistung bei der Unterscheidung der beiden Klassen bietet, allerdings besteht noch Raum für Optimierungen, insbesondere bei der Verbesserung der Recall-Rate für tatsächliche Zahlungsausfälle.

Die ROC-Kurve bietet zudem Flexibilität bei der Bewertung des Modells für verschiedene Schwellenwerte. Je nach Ziel des Kreditunternehmens kann der Schwellenwert so angepasst werden, dass entweder der Fokus auf der Minimierung von Zahlungsausfällen (höherer Recall) oder auf der Reduzierung von falschen Alarme (höhere Precision) liegt.

4.2.2.7.3 Bedeutung für das Kreditunternehmen

Die ROC-Kurve und der AUC-Wert sind für ein Kreditunternehmen von grosser Bedeutung, da sie eine klare Visualisierung und ein quantitatives Mass für die Modellleistung liefern. Ein AUC-Wert von 0,79 zeigt, dass das Modell in den meisten Fällen zwischen den Klassen „Zahlungsausfall“ und „Keine Zahlungsausfall“ unterscheiden kann. Diese Informationen können dazu genutzt werden:

1. **Risikomanagement zu optimieren**, indem potenzielle Zahlungsausfälle frühzeitig identifiziert werden.
2. **Kreditentscheidungen gezielter zu treffen**, indem der Schwellenwert angepasst wird, um entweder konservativer (weniger Kredite vergeben) oder aggressiver (mehr Kredite vergeben) zu handeln.
3. **Praktische Relevanz:** Ein besseres Verständnis der Modellleistung unterstützt die Entwicklung von Strategien, um Zahlungsausfälle zu minimieren und gleichzeitig die Kundenbasis zu erweitern.

4.2.2.7.4 Fazit

Die ROC-Kurve zeigt, dass das XGBoost-Modell eine solide Klassifikationsleistung erbringt, indem es zuverlässig zwischen den Klassen „Zahlungsausfall“ und „Keine Zahlungsausfall“ unterscheidet. Der AUC-Wert von 0,79 bestätigt diese Leistung, weist jedoch auch darauf hin, dass weiteres Optimierungspotenzial besteht, um die Recall-Rate für Zahlungsausfälle zu verbessern. Für das Kreditunternehmen liefert diese Analyse wichtige Grundlagen zur Bewertung und Optimierung der Kreditwürdigkeitsprüfung.

4.2.3 Random Forest

4.2.3.1 Optimierungsstrategie und Ressourcenmanagement

4.2.3.1.1 Effizienz der Optimierungsprozesse

Bei der Optimierung der Hyperparameter für das Random Forest-Modell wurden zwei Durchläufe mit unterschiedlichen Anzahlen an Trials durchgeführt. Der erste Durchlauf umfasste **100 Trials**, während der zweite auf **17 Trials** beschränkt wurde. Die Ergebnisse zeigen, dass die Unterschiede zwischen den erzielten AUC-Werten in den beiden

Durchläufen minimal sind. Beispielsweise liegen die AUC-Werte bei **0.7808** und **0.784511**, was einer Differenz von lediglich **0.48 %** entspricht.

4.2.3.1.2 Ressourceneffizienz

Ein einzelner Trial benötigt im Durchschnitt **1 Minute** Rechenzeit. Der erste Durchlauf mit 100 Trials erforderte somit ca. **1 Stunde und 40 Minuten** Rechenzeit. Im Gegensatz dazu konnte der zweite Durchlauf mit nur 17 Trials die Rechenzeit auf **17 Minuten** reduzieren, was eine erhebliche Zeitersparnis darstellt. Angesichts der geringen Verbesserungen in der AUC-Wertung ist es gerechtfertigt, den Umfang der Optimierungsläufe zu reduzieren, da zusätzliche Trials keinen signifikanten Einfluss auf die Modellleistung haben.

4.2.3.1.3 Praktische Relevanz

Die minimalen Verbesserungen von ca. **0.48 %** in der AUC-Wertung deuten darauf hin, dass der Raum für weitere Optimierungen begrenzt ist. In der Praxis sind solche marginalen Unterschiede oft irrelevant, insbesondere wenn die zusätzlichen Berechnungen hohe Kosten in Bezug auf Rechenzeit oder Ressourcen verursachen.

4.2.3.1.4 Fazit

Die beschränkte Anzahl an Trials (17 statt 100) stellt einen effektiven Kompromiss zwischen Optimierungsgenauigkeit und Ressourceneffizienz dar. Für den Vergleich zwischen verschiedenen Modellen genügt die erreichte Optimierung, da die Unterschiede in der Modellleistung durch zusätzliche Trials nicht signifikant beeinflusst werden. Diese Strategie gewährleistet eine effiziente Nutzung von Ressourcen, ohne die Vergleichbarkeit und Aussagekraft der Ergebnisse zu beeinträchtigen.

4.2.3.2 Cross-Validation Scores

4.2.3.2.1 Cross-Validation Scores (Ohne Optuna)

- **Cross-Validation Scores:** [0.8249, 0.7854, 0.8047, 0.7996, 0.8161, 0.8212, 0.8088, 0.8070, 0.7894, 0.8096]
- **Mean CV Score:** 0.8067 (+/- 0.0240)

Bedeutung:

Die Cross-Validation-Scores zeigen, dass das Random Forest-Modell eine durchschnittliche Genauigkeit von 80.67 % erreicht. Die Streuung (±0.0240) ist moderat und deutet darauf hin, dass die Modellleistung zwischen den verschiedenen Folds variiert, jedoch insgesamt stabil bleibt.

4.2.3.2.2 Cross-Validation Scores (Mit Optuna)

- **Cross-Validation Scores:** [0.7983, 0.7634, 0.7873, 0.7882, 0.7707, 0.7909, 0.7840, 0.7818, 0.7739, 0.7849]
- **Mean CV Score:** 0.7823 (+/- 0.0196)

Bedeutung:

Nach dem Hyperparameter-Tuning mit Optuna sank die durchschnittliche Genauigkeit leicht auf 78.23 %. Gleichzeitig verringerte sich die Streuung auf ±0.0196, was auf eine konsistentere Leistung des Modells hinweist.

4.2.3.3 Test Accuracy

- **Ohne Optuna:** 0.8043
- **Mit Optuna:** 0.7887

Bedeutung:

Das Modell ohne Optimierung zeigte eine höhere Genauigkeit auf den Testdaten (80.43 %) im Vergleich zur optimierten Version (78.87 %). Dies deutet darauf hin, dass das Hyperparameter-Tuning den Fokus stärker auf andere Metriken, wie z. B. die AUC, verlagert haben könnte.

4.2.3.4 AUC Scores

- **Ohne Optuna:** 0.7626 (+/- 0.0235)
- **Mit Optuna:** 0.7808 (+/- 0.0225)

Bedeutung:

Der AUC-Wert, der die Trennschärfe des Modells zwischen den Klassen misst, stieg nach der Optimierung von 0.7626 auf 0.7808. Dies zeigt, dass das optimierte Modell besser in

der Lage ist, zwischen Zahlungsausfällen und Nicht-Zahlungsausfällen zu unterscheiden, obwohl die Gesamtgenauigkeit leicht gesunken ist.

4.2.3.5 Classification Report

Metrik (Ohne Optuna)	**Klasse 0 (Nicht-Default)**	**Klasse 1 (Default)**
Precision	0.83	0.64
Recall	0.95	0.33
F1-Score	0.88	0.44

Metrik (Mit Optuna)	**Klasse 0 (Nicht-Default)**	**Klasse 1 (Default)**
Precision	0.87	0.53
Recall	0.85	0.59
F1-Score	0.86	0.56

Bedeutung der Metriken:

4.2.3.6 Precision (Präzision):

- **Ohne Optuna:** Die Präzision für Klasse 1 beträgt **0.64**, was bedeutet, dass 64 % der als Default klassifizierten Kunden tatsächlich Zahlungsausfälle darstellen.
- **Mit Optuna:** Die Präzision sinkt auf **0.53**, was darauf hinweist, dass das optimierte Modell mehr falsch-positive Vorhersagen (fälschlicherweise als Default klassifizierte Kunden) erzeugt.
- **Bedeutung:** Der Rückgang der Präzision deutet darauf hin, dass das optimierte Modell zwar mehr tatsächliche Zahlungsausfälle erkennt (höherer Recall), dies jedoch auf Kosten einer geringeren Genauigkeit der positiven Vorhersagen geschieht. Für Kreditunternehmen bedeutet dies, dass sie potenziell mehr Kunden

als risikoreich einstufen, die tatsächlich zahlungsfähig sind. Dies könnte zu zusätzlichen Prüfungen oder unnötigen Massnahmen führen.

4.2.3.6.1 Zusammenfassung der Anpassung:

Die Abnahme der Präzision ist das Ergebnis eines bewussten Kompromisses zwischen Präzision und Recall. Das optimierte Modell priorisiert die Erkennung tatsächlicher Zahlungsausfälle (höherer Recall), was für Kreditunternehmen strategisch wertvoll sein kann, da das Übersehen von Ausfällen grössere finanzielle Risiken birgt als zusätzliche Prüfungen bei fälschlicherweise als riskant eingestuften Kunden.

4.2.3.7 Recall (Sensitivität):

- **Ohne Optuna:** Der Recall für Klasse 1 liegt bei **0.33**, was bedeutet, dass nur 33% der tatsächlichen Zahlungsausfälle korrekt erkannt werden.
- **Mit Optuna:** Der Recall verbessert sich auf **0.59**, was zeigt, dass das optimierte Modell deutlich mehr tatsächliche Zahlungsausfälle erkennt.
- **Bedeutung:** Ein höherer Recall ist entscheidend, um sicherzustellen, dass möglichst wenige Zahlungsausfälle übersehen werden. Für Kreditunternehmen reduziert dies das Risiko von finanziellen Verlusten durch unerwartete Zahlungsausfälle.

4.2.3.8 F1-Score:

- **Ohne Optuna:** Der F1-Score für Klasse 1 beträgt **0.44**, was auf eine mässige Balance zwischen Präzision und Recall hinweist.
- **Mit Optuna:** Der F1-Score steigt auf **0.56**, was auf eine deutliche Verbesserung der Balance zwischen Präzision und Recall hinweist.
- **Bedeutung:** Der F1-Score kombiniert Präzision und Recall und ist besonders nützlich bei unausgewogenen Datensätzen, wie es bei der Kreditbewertung der Fall ist. Die Verbesserung zeigt, dass das Modell nach der Optimierung sowohl bei der Reduzierung falsch-positiver als auch falsch-negativer Vorhersagen besser abschneidet.

4.2.3.9 Feature-Wichtigkeit

Feature	Bedeutung (Ohne Optuna)	Bedeutung (Mit Optuna)
PAY_0	7.99 %	20.84 %
PAY_2	3.51 %	8.84 %
PAY_3	-	5.64 %
AGE	4.21 %	-
BILL_AMT1	3.59 %	2.61 %
PAYMENT_TO_BILL_RATIO_1	3.46 %	3.04 %
CREDIT_UTILIZATION_RATIO_1	3.57 %	-

Bedeutung:

- Das Merkmal **PAY_0** bleibt das wichtigste Merkmal, dessen Bedeutung sich nach der Optimierung sogar noch verstärkte. Dies unterstreicht die zentrale Rolle vergangener Zahlungsverzögerungen bei der Vorhersage von Kreditwürdigkeit.
- Andere Merkmale wie **PAY_2** und **PAY_3** wurden nach der Optimierung wichtiger, während die Bedeutung von Variablen wie **BILL_AMT1** und **AGE** abnahm.

4.2.3.10 Verwirrungsmatrix

Metrik	Ohne Optuna	Mit Optuna
True Positives	169	287
False Positives	92	263
True Negatives	1589	1418
False Negatives	329	210

Interpretation:

- Die optimierte Version identifiziert mehr tatsächliche Zahlungsausfälle korrekt (**True Positives: +118**) und übergeht weniger kritische Fälle (**False Negatives: -119**).
- Allerdings steigt die Anzahl der fälschlich als Zahlungsausfälle klassifizierten Fälle (**False Positives: +171**), was darauf hinweist, dass das Modell mehr konservative Entscheidungen trifft.

4.2.3.11 Zusammenfassung

Das Random Forest-Modell hat durch das Hyperparameter-Tuning mit Optuna signifikante Verbesserungen in Bezug auf die AUC-Werte (+0.0237) und Recall-Rate für Zahlungsausfälle erzielt. Diese Änderungen gehen jedoch mit einem leichten Rückgang der Testgenauigkeit (-1.94 %) und einem Anstieg der False Positives einher.

Für ein Kreditunternehmen bedeutet dies:

1. **Ohne Optuna:**
   Die Modellleistung zeigt eine starke Stabilität, wobei der Fokus stärker auf der Präzision der **Klasse 0 (Nicht-Default)** liegt. Dieses Modell eignet sich besser, um unnötige Kreditverweigerungen zu minimieren, da es weniger False Positives erzeugt. Die Präzision für Klasse 1 (Default) ist jedoch höher, was bedeutet, dass das Modell zuverlässiger ist, wenn es Zahlungsausfälle identifiziert.
2. **Mit Optuna:**
   Das optimierte Modell erkennt mehr tatsächliche Zahlungsausfälle (höherer Recall für Klasse 1), wodurch potenzielle finanzielle Verluste durch übersehene Ausfälle reduziert werden. Allerdings geht dies auf Kosten einer geringeren Präzision, was bedeutet, dass eine höhere Anzahl von False Positives (fälschlich als Default klassifizierte Kunden) in Kauf genommen wird.

Bedeutung für die strategische Ausrichtung des Kreditunternehmens:
Die Wahl zwischen den beiden Modellen hängt von den Zielen des Kreditunternehmens ab:

- **Priorisierung der Minimierung von Zahlungsausfällen:** Wenn das Ziel ist, möglichst viele tatsächliche Zahlungsausfälle frühzeitig zu erkennen und finanzielle Verluste zu vermeiden, ist das optimierte Modell vorzuziehen.
- **Priorisierung der Reduzierung von False Positives:** Wenn es wichtiger ist, unnötige Prüfungen oder Massnahmen bei zahlungsfähigen Kunden zu vermeiden, bietet das ursprüngliche Modell eine bessere Lösung.

Durch die Analyse beider Modelle kann das Kreditunternehmen eine informierte Entscheidung treffen, die auf ihren spezifischen geschäftlichen Prioritäten basiert.

4.2.4 Logistische Regression

Die logistische Regression ist ein gängiger Ansatz zur Klassifikation, der insbesondere bei linearen Beziehungen zwischen Merkmalen und der Zielvariablen effektiv ist. In diesem Abschnitt werden die Ergebnisse der logistischen Regression sowohl ohne als auch mit Optimierung durch Optuna vorgestellt und analysiert.

4.2.4.1 Ergebnisse ohne Optuna

4.2.4.1.1 Kreuzvalidierungsergebnisse

- **Mean Accuracy:** 0.6849 (± 0.0127)
  → Die mittlere Genauigkeit zeigt, dass das Modell in etwa 68,5 % der Fälle korrekte Vorhersagen trifft. Die geringe Streuung deutet auf eine stabile Modellleistung hin.
- **Mean AUC:** 0.7315 (± 0.0311)
  → Der AUC-Wert deutet auf eine moderate Trennschärfe hin, da das Modell in 73,15 % der Fälle die richtige Klasse identifizieren kann.

4.2.4.1.2 Testergebnisse

- **Test Accuracy:** 0.6947
  → Das Modell erzielt auf dem Testdatensatz eine Genauigkeit von 69,47 %, was mit den Kreuzvalidierungsergebnissen übereinstimmt.

- **Test AUC Score:** 0.7471
  → Der AUC-Wert von 74,71 % zeigt eine gute Fähigkeit zur Klassentrennung.

4.2.4.1.3 Classification Report

Metrik	Klasse 0 (Nicht-Default)	Klasse 1 (Default)
Precision	0.88	0.40
Recall	0.70	0.67
F1-Score	0.78	0.50

Bedeutung der Metriken:

- **Precision (Klasse 1):** Eine Präzision von 40 % bedeutet, dass 40 % der als Zahlungsausfälle klassifizierten Fälle tatsächlich Zahlungsausfälle sind.
- **Recall (Klasse 1):** Ein Recall von 67 % zeigt, dass das Modell 67 % der tatsächlichen Zahlungsausfälle korrekt identifiziert.
- **F1-Score (Klasse 1):** Mit einem F1-Score von 50 % weist das Modell eine mässige Balance zwischen Precision und Recall auf.

4.2.4.1.4 Feature-Bedeutung

Die Koeffizienten der logistischen Regression bieten Einblick in die Bedeutung der einzelnen Merkmale:

- **Positiv gewichtete Merkmale (z. B. PAY_0: 0.4906):** Diese Merkmale erhöhen die Wahrscheinlichkeit eines Zahlungsausfalls.
- **Negativ gewichtete Merkmale (z. B. PAY_AMT1: -2.0473):** Diese Merkmale senken die Wahrscheinlichkeit eines Zahlungsausfalls.

4.2.4.1.5 2. Ergebnisse mit Optuna

4.2.4.1.5.1 Kreuzvalidierungsergebnisse

- **Mean Accuracy:** 0.6858 (± 0.0123)
  → Die mittlere Genauigkeit ist nahezu identisch mit der unoptimierten Version und zeigt keine signifikante Verbesserung.
- **Mean AUC:** 0.7313 (± 0.0311)
  → Auch der AUC-Wert bleibt nahezu unverändert.

4.2.4.1.6 Testergebnisse

- **Test Accuracy:** 0.6937
  → Die Genauigkeit auf dem Testdatensatz bleibt im Vergleich zur unoptimierten Version unverändert.
- **Test AUC Score:** 0.7474
  → Der AUC-Wert zeigt ebenfalls keine wesentliche Veränderung.

4.2.4.1.7 Classification Report

Metrik	Klasse 0 (Nicht-Default)	Klasse 1 (Default)
Precision	0.88	0.40
Recall	0.70	0.67
F1-Score	0.78	0.50

Bedeutung der Metriken:

- Die Metriken zeigen keine wesentliche Verbesserung im Vergleich zur unoptimierten Version. Precision, Recall und F1-Score für Klasse 1 (Default) bleiben unverändert.

4.2.4.1.8 Feature-Bedeutung

Die optimierten Koeffizienten zeigen leichte Anpassungen, die jedoch keine signifikanten Änderungen in der Modellleistung bewirken:

- **Positiv gewichtete Merkmale (z. B. PAYMENT_TO_BILL_RATIO_1: 1.6286):** Diese erhöhen weiterhin die Wahrscheinlichkeit eines Zahlungsausfalls.
- **Negativ gewichtete Merkmale (z. B. PAY_AMT1: -2.2696):** Diese senken die Wahrscheinlichkeit eines Zahlungsausfalls.

4.2.4.1.9 3. Vergleich der Ergebnisse

Metrik	Ohne Optuna	Mit Optuna	Änderung
Accuracy	0.6947	0.6937	Keine signifikante Änderung
AUC Score	0.7471	0.7474	Keine signifikante Änderung
Precision (Klasse 1)	0.40	0.40	Keine Änderung
Recall (Klasse 1)	0.67	0.67	Keine Änderung
F1-Score (Klasse 1)	0.50	0.50	Keine Änderung

Die Ergebnisse deuten darauf hin, dass die Optimierung mit Optuna keine wesentliche Verbesserung der Modellleistung bewirkt hat. Dies könnte darauf hindeuten, dass die Hyperparameter der logistischen Regression bereits in der unoptimierten Version gut angepasst waren oder dass der Suchraum von Optuna keine signifikant besseren Werte gefunden hat.

4.2.4.1.10 4. Bedeutung für das Kreditunternehmen

Die logistische Regression bietet eine transparente und interpretierbare Möglichkeit zur Bewertung der Kreditwürdigkeit. Die Ergebnisse zeigen, dass:

1. **Hohe Precision für Klasse 0 (Nicht-Default):** Das Modell ist effektiv darin, Kunden zu identifizieren, die keine Zahlungsausfälle haben, was unnötige Massnahmen reduziert.
2. **Moderate Recall für Klasse 1 (Default):** Das Modell erkennt 67 % der tatsächlichen Zahlungsausfälle, was ein akzeptabler Wert ist, aber optimiert werden könnte, um finanzielle Verluste weiter zu minimieren.

3. **Feature-Interpretierbarkeit:** Die Gewichtung der Merkmale bietet wertvolle Einblicke in die wichtigsten Faktoren, die die Kreditwürdigkeit beeinflussen.

4.2.4.1.11 Fazit

Die logistische Regression zeigt eine solide Leistung bei der Kreditwürdigkeitsbewertung, insbesondere durch ihre Interpretierbarkeit. Die Optimierung mit Optuna hat jedoch keine wesentlichen Verbesserungen gebracht, was darauf hindeutet, dass das Modell in seiner ursprünglichen Form bereits gut abgestimmt war. Für zukünftige Arbeiten könnte eine Untersuchung nicht-linearer Beziehungen oder die Kombination der logistischen Regression mit anderen Methoden (z. B. Ensembles) weitere Verbesserungen ermöglichen.

5 ERGEBNISSE UND DISKUSSION

5.1 PERFORMANCEMETRIKEN IM MODELLVERGLEICH

Die Modelle Logistische Regression, Random Forest und XGBoost wurden anhand verschiedener Performancemetriken evaluiert. Während XGBoost mit einer AUC von 0.79 die beste Klassentrennung zeigte, bot Random Forest eine ausgeglichene Leistung zwischen Recall und Präzision. Die Logistische Regression erreichte eine moderate AUC von 0.74, überzeugte jedoch durch einfache Interpretierbarkeit und Stabilität. XGBoost lieferte bei der Erkennung von Zahlungsausfällen (Recall) solide Ergebnisse, wenngleich Random Forest durch weniger False Positives hervorstach.

Die ROC-Kurven und die AUC-Werte verdeutlichen die Fähigkeit der Modelle, zwischen Zahlungsausfällen und Nicht-Zahlungsausfällen zu differenzieren. Auffällig ist, dass XGBoost durch Hyperparameter-Tuning deutliche Verbesserungen hinsichtlich Recall und F1-Score erzielen konnte, während die Logistische Regression trotz Optimierung nahezu unverändert blieb.

5.2 STÄRKEN UND SCHWÄCHEN DER UNTERSUCHTEN MODELLE

5.2.1 Logistische Regression:

- **Stärken**: Hohe Interpretierbarkeit, geringer Rechenaufwand und Stabilität.
- **Schwächen**: Begrenzte Fähigkeit zur Modellierung nicht-linearer Beziehungen; Recall für Zahlungsausfälle vergleichsweise niedrig.

5.2.2 Random Forest:

- **Stärken**: Robust gegenüber Ausreissern, gute Balance zwischen Präzision und Recall, stabil über verschiedene Datenaufteilungen.
- **Schwächen**: Höherer Rechenaufwand, weniger interpretierbar als die Logistische Regression.

5.2.3 XGBoost:

- **Stärken**: Höchste AUC, flexibel und effizient durch Hyperparameter-Tuning, robust bei grossen und komplexen Datensätzen.
- **Schwächen**: Höhere Komplexität, geringere Interpretierbarkeit im Vergleich zu anderen Modellen.

5.3 PRAKTISCHE RELEVANZ DER ERGEBNISSE

Die Ergebnisse unterstreichen die Vielseitigkeit von Machine-Learning-Modellen in der Kreditrisikoanalyse:

1. **Logistische Regression** eignet sich für Anwendungen, bei denen Interpretierbarkeit und schnelle Entscheidungen Vorrang haben, z. B. in regulierten Bereichen.
2. **Random Forest** bietet eine ausgewogene Performance und ist besonders nützlich, wenn robuste Ergebnisse bei unausgewogenen Datensätzen erforderlich sind.
3. **XGBoost** zeigt die beste Performance bei der Identifikation von Zahlungsausfällen und eignet sich für Szenarien mit hohen Anforderungen an Genauigkeit und Anpassungsfähigkeit.

Die Modellwahl sollte daher abhängig von den Geschäftszielen erfolgen: Ein höherer Recall, wie ihn XGBoost liefert, minimiert das Risiko durch übersehene Zahlungsausfälle, während Random Forest eine konservativere Bewertung ermöglicht. Die Logistische Regression kann als Grundlage für einfache Szenarien dienen.

6 AUSBLICK UND LIMITATIONEN

6.1 HERAUSFORDERUNGEN BEI DER IMPLEMENTIERUNG IN DER PRAXIS

Die Anwendung von Machine-Learning-Modellen zur Vorhersage von Zahlungsausfällen in realen Szenarien ist mit mehreren Herausforderungen verbunden:

1. **Datenqualität und -verfügbarkeit**:
    - Viele Kreditinstitute verfügen nicht über qualitativ hochwertige, vollständige und aktuelle Daten. Fehlende oder ungenaue Daten können die Modellleistung erheblich beeinträchtigen.
    - Der im Projekt verwendete Datensatz ist spezifisch für Taiwan und repräsentiert möglicherweise nicht die demografischen und finanziellen Bedingungen anderer Märkte.
2. **Klassenungleichgewicht**:
    - Das Ungleichgewicht zwischen Zahlungsausfällen (Minoritätsklasse) und Nicht-Zahlungsausfällen (Majoritätsklasse) stellt eine zentrale Herausforderung dar. Auch nach der Anpassung von Klassengewichten bleiben Modelle wie XGBoost und Random Forest anfällig für eine Überbewertung der Majoritätsklasse.
3. **Modellinterpretation**:
    - Modelle wie XGBoost und Random Forest bieten eine hohe Leistung, jedoch auf Kosten der Interpretierbarkeit. In der Finanzindustrie, die oft regulatorische Anforderungen an Transparenz stellt, kann dies ein Hindernis darstellen.
    - Die Logistische Regression hingegen ist einfacher interpretierbar, leidet jedoch an geringerer Modellleistung in komplexen Szenarien.
4. **Rechenleistung und Skalierbarkeit**:

- Der Einsatz von Modellen wie XGBoost kann insbesondere bei grossen Datensätzen erhebliche Rechenressourcen erfordern. Dies kann die Skalierbarkeit und den Einsatz in Echtzeitsystemen erschweren.

5. **Regulatorische und ethische Anforderungen**:
   - Die Nutzung von Machine-Learning-Modellen in der Kreditrisikoanalyse muss den gesetzlichen und ethischen Vorgaben entsprechen. Diskriminierungsrisiken bei sensiblen Merkmalen (z. B. Geschlecht, Alter) müssen minimiert werden.

6.2 Potenziale für zukünftige Arbeiten

Die Ergebnisse dieser Arbeit eröffnen mehrere Möglichkeiten für weiterführende Forschung und praktische Anwendungen:

1. **Erweiterung des Datensatzes**:
   - Der Einsatz eines globalen Datensatzes mit einer grösseren Vielfalt an Kreditnehmerprofilen könnte die Generalisierbarkeit der Modelle verbessern.
   - Die Integration zusätzlicher Datenquellen wie makroökonomische Indikatoren oder Social-Media-Informationen könnte die Vorhersagegenauigkeit weiter steigern.
2. **Kombination von Modellen**:
   - Hybride Ansätze, die die Stärken verschiedener Modelle (z. B. Logistische Regression für Interpretierbarkeit und XGBoost für Leistung) kombinieren, könnten zu robusteren Ergebnissen führen.
   - Die Verwendung von Ensemble-Methoden, wie Stacking oder Bagging, könnte weitere Performance-Steigerungen ermöglichen.
3. **Explainable AI (XAI)**:
   - Die Entwicklung von Methoden zur Erhöhung der Interpretierbarkeit komplexer Modelle, wie z. B. SHAP-Werte oder LIME, könnte deren Akzeptanz in der Praxis verbessern.
   - Eine detaillierte Analyse der Feature-Wichtigkeit könnte die Transparenz der Modellentscheidungen erhöhen.

4. **Optimierung der Klassenungleichheit**:
    - Der Einsatz von Techniken wie SMOTE (Synthetic Minority Over-sampling Technique) oder Cost-sensitive Learning könnte die Erkennung von Zahlungsausfällen weiter verbessern.
    - Eine adaptive Gewichtung der Klassen basierend auf dem Geschäftsrisiko könnte praktikable Lösungen bieten.
5. **Integration in Echtzeitsysteme**:
    - Die Implementierung der Modelle in Echtzeitsysteme zur Kreditrisikobewertung könnte die Entscheidungsfindung automatisieren und beschleunigen.
    - Die Entwicklung von Modellen mit geringeren Rechenanforderungen würde deren Einsatz in mobilen oder ressourcenbeschränkten Umgebungen ermöglichen.
6. **Langfristige Validierung**:
    - Zukünftige Arbeiten könnten die langfristige Leistung der Modelle analysieren, um ihre Robustheit und Stabilität über verschiedene wirtschaftliche Bedingungen hinweg zu bewerten.
    - Der Einsatz von Zeitreihenmodellen könnte helfen, Trends in Zahlungsausfällen besser zu erkennen.
7. **Integration in Geschäftsprozesse**:
    - Die Modelle könnten als Entscheidungsunterstützung in Kreditbewertungsprozesse integriert werden, um Risiken proaktiv zu identifizieren und Massnahmen zu ergreifen.
    - Ein Vergleich mit traditionellen Scoring-Modellen, die in der Kreditbewertung verwendet werden, könnte die Vorteile moderner Machine-Learning-Ansätze aufzeigen.

Diese Ansätze können dazu beitragen, die Vorhersagekraft und praktische Anwendbarkeit von Machine-Learning-Modellen in der Kreditrisikoanalyse weiter zu optimieren und sie gleichzeitig an die spezifischen Anforderungen der Finanzindustrie anzupassen.

7 FAZIT

In dieser Arbeit wurde die Leistungsfähigkeit verschiedener Machine-Learning-Modelle – Logistische Regression, Random Forest und XGBoost – bei der Vorhersage von Zahlungsausfällen auf Basis des "Default of Credit Card Clients"-Datensatzes systematisch untersucht. Ziel war es, die Modelle hinsichtlich ihrer Performancemetriken, Stärken, Schwächen und ihrer praktischen Relevanz zu vergleichen, um fundierte Empfehlungen für ihren Einsatz in der Kreditrisikoanalyse zu entwickeln.

Die Ergebnisse zeigen, dass jedes Modell spezifische Vor- und Nachteile aufweist. Die Logistische Regression überzeugte durch ihre einfache Implementierbarkeit, hohe Interpretierbarkeit und geringe Rechenanforderungen. Diese Eigenschaften machen sie zu einem idealen Ausgangspunkt für Analysen und besonders geeignet für regulierte Umgebungen, in denen Transparenz der Modellentscheidungen essenziell ist. Jedoch ist ihre Leistung begrenzt, insbesondere bei komplexen und nichtlinearen Beziehungen in den Daten.

Der Random Forest bot eine ausgewogene Performance und war robust gegenüber Ausreissern und unausgeglichenen Datensätzen. Die Kombination mehrerer Entscheidungsbäume führte zu einer stabilen und zuverlässigen Leistung, allerdings auf Kosten der Interpretierbarkeit und mit einem höheren Rechenaufwand.

XGBoost erwies sich als das leistungsfähigste Modell in dieser Analyse. Es lieferte die besten Ergebnisse hinsichtlich der AUC und der Trennschärfe zwischen Zahlungsausfällen und Nicht-Zahlungsausfällen. Gleichzeitig erforderte es jedoch eine umfangreiche Hyperparameter-Optimierung und erhöhte Rechenressourcen. Die geringere Interpretierbarkeit kann in sensiblen Anwendungen wie der Kreditbewertung eine Herausforderung darstellen, kann jedoch durch erklärbare KI-Ansätze (Explainable AI) ausgeglichen werden.

Die Analyse machte deutlich, dass keine universelle Lösung existiert. Die Wahl des Modells hängt von den spezifischen Anforderungen ab – sei es Interpretierbarkeit, Rechenaufwand oder Vorhersagegenauigkeit. Für die Praxis bedeutet dies, dass Kreditinstitute ihre Modellwahl sorgfältig auf ihre Geschäftsziele und regulatorischen Anforderungen abstimmen müssen.

Die Arbeit verdeutlicht ausserdem die Notwendigkeit, sich mit Herausforderungen wie Datenqualität, Klassenungleichgewichten und Modellinterpretation auseinanderzusetzen, um die zuverlässige Implementierung solcher Modelle in der Praxis zu gewährleisten. Gleichzeitig eröffnen die Ergebnisse vielversprechende Ansätze für zukünftige Arbeiten, etwa durch die Kombination von Modellen oder die Entwicklung neuer Methoden zur Erhöhung der Interpretierbarkeit und Skalierbarkeit.

Insgesamt zeigt diese Arbeit, dass Machine-Learning-Modelle ein enormes Potenzial zur Verbesserung der Kreditrisikoanalyse bieten. Sie können nicht nur die

Entscheidungsfindung automatisieren und beschleunigen, sondern auch wertvolle Einblicke in die zugrunde liegenden Risikofaktoren liefern. Mit gezielten Optimierungen und Anpassungen können diese Modelle langfristig dazu beitragen, das Kreditrisikomanagement zu revolutionieren und finanzielle Verluste für Kreditinstitute zu minimieren.

8 Anhang

8.1 Python-Code für Datenverarbeitung und Modellimplementierung

8.1.1 main.py

```
import os
import sys

# Add the project root directory to Python path
project_root = os.path.dirname(os.path.abspath(__file__))
if project_root not in sys.path:
    sys.path.append(project_root)

from data_loader import (load_cleaned_data, load_data,
                         load_test_data,
                         load_train_data)
from descriptive_analysis import run_descriptive_analysis
from data_cleaning import clean_data
from data_conversion import (scale_features,
                                      print_scaling_info,
                                      perform_one_hot_encoding,
                                      print_encoding_info,
                                      calculate_financial_ratios,
                                      print_ratio_stats,
                                      split_data,
                                      print_split_stats)
from xgboost_model import train_xgboost_model
from data_management import setup_data_directories
from outlier_analysis import run_outlier_analysis
from visualization_runner import (create_all_visualizations,
                                  create_model_visualizations,
                                  plot_roc_curve)
from random_forest_model import train_random_forest_model
from logistic_regression_model import train_logistic_regression_model


def process_data(data):
    """Process data through the complete pipeline"""
    # Clean data
    clean_data(data) # Print removal stats
    cleaned_data = load_cleaned_data()

    # Scale features
```

```
    scaled_data, scaling_info = scale_features(cleaned_data)
    print_scaling_info(scaling_info)

    # Perform one-hot encoding
    encoded_data, encoding_info = perform_one_hot_encoding(scaled_data)
    print_encoding_info(encoding_info)

    # Calculate financial ratios
    enhanced_data, ratio_stats = calculate_financial_ratios(encoded_data)
    print_ratio_stats(ratio_stats)

    # Split data into training and test sets
    split_stats, train_data, test_data = split_data(enhanced_data)
    print_split_stats(split_stats)

    return train_data, test_data

def main():
    # Setup data directories
    data_paths = setup_data_directories()
    print("Data directories initialized...")

    try:
        # Load data
        data = load_data()
    except FileNotFoundError as e:
        print(str(e))
        return

    # Choose which analyses to run
    RUN_DESCRIPTIVE = False
    RUN_OUTLIER_ANALYSIS = False
    CREATE_VISUALIZATIONS = False
    CLEAN_DATA = False
    XGBOOST_MODEL = False
    RANDOM_FOREST_MODEL = False
    LOGISTIC_REGRESSION_MODEL = True

    if RUN_DESCRIPTIVE:
        run_descriptive_analysis(data)

    if RUN_OUTLIER_ANALYSIS:
        run_outlier_analysis(data)

    if CREATE_VISUALIZATIONS:
```

```
        create_all_visualizations(data)

    # Process data and get train/test sets
    if CLEAN_DATA:
        train_data, test_data = process_data(data)
    else:
        try:
            # Try to load preprocessed data
            test_data = load_test_data()
            train_data = load_train_data()
        except FileNotFoundError:
            print("\nPreprocessed data files not found. Running data processing
pipeline...")
            train_data, test_data = process_data(data)

    if XGBOOST_MODEL:
        print("\nStarting XGBoost Model Training...")

        # Prepare test data for ROC curve
        X_test = test_data.drop(['ID', 'default.payment.next.month'], axis=1)
        y_test = test_data['default.payment.next.month']

        # Train and evaluate model
        model_results = train_xgboost_model(train_data, test_data,
use_optuna=True)

        # Create visualizations
        create_model_visualizations(model_results, 'xgboost')
        plot_roc_curve(model_results, X_test, y_test, 'xgboost',
output_dir='Diagramms/XGBoost')

        print("\nModel Performance:")
        print(f"Accuracy: {model_results['accuracy']:.4f}")
        print(f"AUC Score: {model_results['auc_score']:.4f}")
        print("\nBest Parameters:", model_results['best_params'])

    if RANDOM_FOREST_MODEL:
        print("\nStarting Random Forest Model Training...")

        # Prepare test data for ROC curve
        X_test = test_data.drop(['ID', 'default.payment.next.month'], axis=1)
        y_test = test_data['default.payment.next.month']

        # Train and evaluate model
```

```
        model_results = train_random_forest_model(train_data, test_data,
use_optuna=True)

        # Create visualizations
        create_model_visualizations(model_results, 'random_forest')
        plot_roc_curve(model_results, X_test, y_test, 'random_forest',
output_dir='Diagramms/RandomForest')

        print("\nModel Performance:")
        print(f"Accuracy: {model_results['accuracy']:.4f}")
        print(f"AUC Score: {model_results['auc_score']:.4f}")
        print("\nBest Parameters:", model_results['best_params'])
        print("\nClassification Report:")
        print(model_results['classification_report'])

    if LOGISTIC_REGRESSION_MODEL:
        print("\nStarting Logistic Regression Model Training...")

        # Prepare test data for ROC curve
        X_test = test_data.drop(['ID', 'default.payment.next.month'], axis=1)
        y_test = test_data['default.payment.next.month']

        # Train and evaluate model
        model_results = train_logistic_regression_model(train_data, test_data,
use_optuna=True)

        # Create visualizations
        create_model_visualizations(model_results, 'logistic_regression')
        plot_roc_curve(model_results, X_test, y_test, 'logistic_regression',
                       output_dir='Diagramms/LogisticRegression')

        print("\nModel Performance:")
        print(f"Accuracy: {model_results['accuracy']:.4f}")
        print(f"AUC Score: {model_results['auc_score']:.4f}")
        print("\nBest Parameters:", model_results['best_params'])
        print("\nClassification Report:")
        print(model_results['classification_report'])

if __name__ == "__main__":
    main()
```

8.1.2 data_loader.py

```
import pandas as pd
```

```
import os
from config import (CLEANED_DATA_FILE,
                    DATA_FILE, RAW_DATA_DIR,
                    TEST_DATA_FILE, TRAIN_DATA_FILE)

def check_raw_data_exists():
    """Check if raw data file exists and guide user if it doesn't"""
    if not os.path.exists(DATA_FILE):
        print("\nERROR: Raw data file not found!")
        print(f"Please place the UCI_Credit_Card.csv file in the following
directory:")
        print(f"{RAW_DATA_DIR}")
        print("\nYou can download the dataset from:")
        print("https://archive.ics.uci.edu/dataset/350/default+of+credit+card+cli
ents")
        return False
    return True

def load_dataset(file_path):
    """Generic function to load any CSV dataset"""
    if not os.path.exists(file_path):
        raise FileNotFoundError(f"Data file not found: {file_path}")
    return pd.read_csv(file_path)

def load_data():
    """Load the original credit card dataset"""
    if not check_raw_data_exists():
        raise FileNotFoundError("Please add the raw data file before
proceeding.")
    return load_dataset(DATA_FILE)

def load_cleaned_data():
    """Load the cleaned credit card dataset"""
    return load_dataset(CLEANED_DATA_FILE)

def load_test_data():
    """Load the test dataset"""
    return load_dataset(TEST_DATA_FILE)

def load_train_data():
    """Load the training dataset"""
    return load_dataset(TRAIN_DATA_FILE)
```

8.1.3 data_cleaning.py

```
from data_conversion import remove_zscore_outliers, print_removal_stats
from config import CLEANED_DATA_FILE, PROCESSED_DATA_DIR
import os

def clean_data(data):
    """Clean data by removing outliers and save to file

    Args:
        data: DataFrame to clean
    """
    # Ensure the processed directory exists
    os.makedirs(PROCESSED_DATA_DIR, exist_ok=True)

    # Clean data and save to file
    removal_stats = remove_zscore_outliers(data, CLEANED_DATA_FILE)
    print_removal_stats(removal_stats)
```

8.1.4 data_conversion.py

```
import numpy as np
from descriptive_stats import analyze_bill_amt_outliers_z,
analyze_pay_amt_outliers_z
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from config import (SCALED_DATA_FILE,
                    ENCODED_DATA_FILE,
                    FINANCIAL_RATIOS_FILE,
                    TEST_DATA_FILE,
                    TRAIN_DATA_FILE)

def remove_zscore_outliers(data, output_path):
    """
    Remove outliers identified by Z-Score method using row IDs and save cleaned
data to CSV

    Args:
        data (pd.DataFrame): Original DataFrame
        output_path (str): Path where to save the cleaned CSV file

    Returns:
        dict: Statistics about removed outliers
    """
    # Get outliers information
    bill_outliers = analyze_bill_amt_outliers_z(data)
```

```
    pay_outliers = analyze_pay_amt_outliers_z(data)

    # Track statistics
    stats = {
        'original_rows': len(data),
        'removed_by_column': {},
        'ids_removed': set()  # Use set to avoid counting duplicates
    }

    # Collect all IDs to remove
    for outliers_dict in [bill_outliers, pay_outliers]:
        for column, info in outliers_dict.items():
            stats['removed_by_column'][column] = info['count']
            stats['ids_removed'].update(info['ids'])

    # Remove rows with collected IDs
    cleaned_data = data[~data['ID'].isin(stats['ids_removed'])]
    cleaned_data.to_csv(output_path, index=False)

    # Calculate final statistics
    stats['total_removed'] = len(stats['ids_removed'])
    stats['remaining_rows'] = len(cleaned_data)
    stats['removal_percentage'] = round(stats['total_removed'] /
stats['original_rows'] * 100, 2)

    return stats

def print_removal_stats(stats):
    """
    Print formatted statistics about removed outliers

    Args:
        stats (dict): Statistics dictionary from remove_zscore_outliers
    """
    print("\nOutlier Removal Statistics:")
    print(f"Original rows: {stats['original_rows']}")
    print(f"Rows removed: {stats['total_removed']}")
    print(f"Rows remaining: {stats['remaining_rows']}")
    print(f"Removal percentage: {stats['removal_percentage']}%")

    print("\nRows removed by column:")
    for column, count in stats['removed_by_column'].items():
        print(f"  {column}: {count} rows")

def scale_features(data, output_path=SCALED_DATA_FILE):
```

```
    """
    Apply MinMax scaling to specified numeric columns and save to CSV

    Args:
        data (pd.DataFrame): DataFrame containing the cleaned data
        output_path (str): Path where to save the scaled CSV file

    Returns:
        tuple: (scaled DataFrame, dict with scaling info)
    """
    # Create copy of data
    scaled_data = data.copy()

    # Define columns to scale
    columns_to_scale = ['LIMIT_BAL', 'AGE']
    columns_to_scale.extend([f'BILL_AMT{i}' for i in range(1, 7)])
    columns_to_scale.extend([f'PAY_AMT{i}' for i in range(1, 7)])

    # Initialize scaler
    scaler = MinMaxScaler()

    # Store scaling information
    scaling_info = {
        'columns_scaled': columns_to_scale,
        'original_ranges': {},
        'scaled_ranges': {}
    }

    # Scale selected columns
    scaled_values = scaler.fit_transform(scaled_data[columns_to_scale])

    # Store original and scaled ranges
    for idx, column in enumerate(columns_to_scale):
        original_values = scaled_data[column]
        scaled_column_values = scaled_values[:, idx]

        scaling_info['original_ranges'][column] = {
            'min': float(original_values.min()),
            'max': float(original_values.max())
        }
        scaling_info['scaled_ranges'][column] = {
            'min': float(scaled_column_values.min()),
            'max': float(scaled_column_values.max())
        }
```

```
            # Update the data with scaled values
            scaled_data[column] = scaled_column_values

    # Save scaled data
    scaled_data.to_csv(output_path, index=False)
    print_scaling_info(scaling_info)

    return scaled_data, scaling_info

def print_scaling_info(scaling_info):
    """
    Print formatted information about the scaling process

    Args:
        scaling_info (dict): Dictionary containing scaling information
    """
    print("\nFeature Scaling Information:")
    print(f"Number of scaled columns: {len(scaling_info['columns_scaled'])}")

    print("\nScaling ranges for each column:")
    for column in scaling_info['columns_scaled']:
        orig = scaling_info['original_ranges'][column]
        scaled = scaling_info['scaled_ranges'][column]
        print(f"\n{column}:")
        print(f"  Original range: [{orig['min']:.2f}, {orig['max']:.2f}]")
        print(f"  Scaled range:   [{scaled['min']:.2f}, {scaled['max']:.2f}]")

def perform_one_hot_encoding(data, output_path=ENCODED_DATA_FILE):
    """
    Perform one-hot encoding for SEX, EDUCATION, and MARRIAGE columns and remove
original columns

    Args:
        data (pd.DataFrame): DataFrame containing the data
        output_path (str): Path where to save the encoded CSV file

    Returns:
        tuple: (encoded DataFrame, dict with encoding info)
    """
    # Create copy of data
    encoded_data = data.copy()

    # Track encoding information
    encoding_info = {
        'original_columns': ['SEX', 'EDUCATION', 'MARRIAGE'],
```

```
        'new_columns': [],
        'removed_columns': []
    }

    # SEX encoding (1=male, 2=female)
    encoded_data['MALE'] = (encoded_data['SEX'] == 1).astype(int)
    encoded_data['FEMALE'] = (encoded_data['SEX'] == 2).astype(int)
    encoding_info['new_columns'].extend(['MALE', 'FEMALE'])

    # EDUCATION encoding
    education_mapping = {
        1: 'GRADUATE_SCHOOL',
        2: 'UNIVERSITY',
        3: 'HIGH_SCHOOL',
        4: 'OTHERS_EDUCATION',
        5: 'UNKNOWN_EDUCATION',
        6: 'UNKNOWN_EDUCATION',
        0: 'UNKNOWN_EDUCATION'  # Handle potential 0 values
    }

    for value, column_name in education_mapping.items():
        if column_name not in encoded_data.columns:  # Avoid duplicate columns
for UNKNOWN
            encoded_data[column_name] = (encoded_data['EDUCATION'] ==
value).astype(int)
            encoding_info['new_columns'].append(column_name)

    # MARRIAGE encoding
    marriage_mapping = {
        1: 'MARRIED',
        2: 'SINGLE',
        3: 'OTHER_RELATIONSHIP'
    }

    for value, column_name in marriage_mapping.items():
        encoded_data[column_name] = (encoded_data['MARRIAGE'] ==
value).astype(int)
        encoding_info['new_columns'].append(column_name)

    # Remove original categorical columns
    for column in encoding_info['original_columns']:
        encoding_info['removed_columns'].append(column)
        encoded_data.drop(column, axis=1, inplace=True)

    # Save encoded data
```

```
    encoded_data.to_csv(output_path, index=False)

    # Add verification info
    encoding_info['total_new_columns'] = len(encoding_info['new_columns'])
    encoding_info['verification'] = {
        'sex_sum': encoded_data[['MALE', 'FEMALE']].sum().to_dict(),
        'education_sum': encoded_data[['GRADUATE_SCHOOL', 'UNIVERSITY',
'HIGH_SCHOOL',
                                       'OTHERS_EDUCATION',
'UNKNOWN_EDUCATION']].sum().to_dict(),
        'marriage_sum': encoded_data[['MARRIED', 'SINGLE',
'OTHER_RELATIONSHIP']].sum().to_dict()
    }

    return encoded_data, encoding_info

def print_encoding_info(encoding_info):
    """
    Print formatted information about the encoding process

    Args:
        encoding_info (dict): Dictionary containing encoding information
    """
    print("\nOne-Hot Encoding Information:")
    print(f"Original columns removed: {',
'.join(encoding_info['removed_columns'])}")
    print(f"New columns created: {encoding_info['total_new_columns']}")

    print("\nVerification Counts:")
    print("\nSEX encoding:")
    for col, count in encoding_info['verification']['sex_sum'].items():
        print(f"  {col}: {int(count)} rows")

    print("\nEDUCATION encoding:")
    for col, count in encoding_info['verification']['education_sum'].items():
        print(f"  {col}: {int(count)} rows")

    print("\nMARRIAGE encoding:")
    for col, count in encoding_info['verification']['marriage_sum'].items():
        print(f"  {col}: {int(count)} rows")

def calculate_financial_ratios(data, output_path=FINANCIAL_RATIOS_FILE):
    """
    Calculate and scale financial ratios from credit card data
```

```
    Args:
        data (pd.DataFrame): DataFrame containing the encoded credit card data
        output_path (str): Path where to save the enhanced CSV file

    Returns:
        tuple: (enhanced DataFrame, dict with ratio statistics)
    """
    # Create copy of data
    enhanced_data = data.copy()

    # Track ratio statistics
    ratio_stats = {
        'credit_utilization': {},
        'payment_to_bill': {}
    }

    # Calculate credit utilization ratios (BILL_AMT / LIMIT_BAL)
    for i in range(1, 7):
        column_name = f'CREDIT_UTILIZATION_RATIO_{i}'
        # Handle zero division
        enhanced_data[column_name] = np.where(
            enhanced_data['LIMIT_BAL'] != 0,
            enhanced_data[f'BILL_AMT{i}'] / enhanced_data['LIMIT_BAL'],
            0
        )

        ratio_stats['credit_utilization'][column_name] = {
            'mean': float(enhanced_data[column_name].mean()),
            'median': float(enhanced_data[column_name].median()),
            'min': float(enhanced_data[column_name].min()),
            'max': float(enhanced_data[column_name].max())
        }

    # Calculate payment to bill ratios (PAY_AMT / BILL_AMT)
    for i in range(1, 7):
        column_name = f'PAYMENT_TO_BILL_RATIO_{i}'
        # Handle zero division
        enhanced_data[column_name] = np.where(
            enhanced_data[f'BILL_AMT{i}'] != 0,
            enhanced_data[f'PAY_AMT{i}'] / enhanced_data[f'BILL_AMT{i}'],
            0
        )

        ratio_stats['payment_to_bill'][column_name] = {
            'mean': float(enhanced_data[column_name].mean()),
```

```
            'median': float(enhanced_data[column_name].median()),
            'min': float(enhanced_data[column_name].min()),
            'max': float(enhanced_data[column_name].max())
        }

    # Scale the new ratio columns
    scaler = MinMaxScaler()
    ratio_columns = (
        [f'CREDIT_UTILIZATION_RATIO_{i}' for i in range(1, 7)] +
        [f'PAYMENT_TO_BILL_RATIO_{i}' for i in range(1, 7)]
    )

    # Store original values for statistics
    for column in ratio_columns:
        ratio_stats[column] = {
            'original_range': {
                'min': float(enhanced_data[column].min()),
                'max': float(enhanced_data[column].max())
            }
        }

    # Scale values
    scaled_ratios = scaler.fit_transform(enhanced_data[ratio_columns])

    # Update DataFrame with scaled values and store scaled ranges
    for idx, column in enumerate(ratio_columns):
        enhanced_data[column] = scaled_ratios[:, idx]
        ratio_stats[column]['scaled_range'] = {
            'min': float(enhanced_data[column].min()),
            'max': float(enhanced_data[column].max())
        }

    # Save enhanced data
    enhanced_data.to_csv(output_path, index=False)

    return enhanced_data, ratio_stats

def print_ratio_stats(ratio_stats):
    """
    Print formatted statistics about the calculated ratios

    Args:
        ratio_stats (dict): Dictionary containing ratio statistics
    """
    print("\nFinancial Ratio Statistics:")
```

```
    print("\nCredit Utilization Ratios (BILL_AMT / LIMIT_BAL):")
    for i in range(1, 7):
        column = f'CREDIT_UTILIZATION_RATIO_{i}'
        print(f"\n{column}:")
        print("  Original range: "
              f"[{ratio_stats[column]['original_range']['min']:.2f}, "
              f"{ratio_stats[column]['original_range']['max']:.2f}]")
        print("  Scaled range: "
              f"[{ratio_stats[column]['scaled_range']['min']:.2f}, "
              f"{ratio_stats[column]['scaled_range']['max']:.2f}]")

    print("\nPayment to Bill Ratios (PAY_AMT / BILL_AMT):")
    for i in range(1, 7):
        column = f'PAYMENT_TO_BILL_RATIO_{i}'
        print(f"\n{column}:")
        print("  Original range: "
              f"[{ratio_stats[column]['original_range']['min']:.2f}, "
              f"{ratio_stats[column]['original_range']['max']:.2f}]")
        print("  Scaled range: "
              f"[{ratio_stats[column]['scaled_range']['min']:.2f}, "
              f"{ratio_stats[column]['scaled_range']['max']:.2f}]")

def split_data(data, train_output=TRAIN_DATA_FILE, test_output=TEST_DATA_FILE,
test_size=0.2, random_state=42):
    """
    Split data into training and test sets

    Args:
        data (pd.DataFrame): DataFrame to split
        train_output (str): Path where to save training data
        test_output (str): Path where to save test data
        test_size (float): Proportion of data to use for testing (default: 0.2)
        random_state (int): Random seed for reproducibility (default: 42)

    Returns:
        tuple: (dict with split statistics, training DataFrame, test DataFrame)
    """
    # Create copy of data
    data_copy = data.copy()

    # Split the data
    train_data, test_data = train_test_split(
        data_copy,
        test_size=test_size,
```

```
        random_state=random_state,
        stratify=data_copy['default.payment.next.month']  # Ensure balanced
splits
    )

    # Save split datasets
    train_data.to_csv(train_output, index=False)
    test_data.to_csv(test_output, index=False)

    # Calculate split statistics
    split_stats = {
        'total_rows': len(data),
        'train_rows': len(train_data),
        'test_rows': len(test_data),
        'train_percentage': (len(train_data) / len(data)) * 100,
        'test_percentage': (len(test_data) / len(data)) * 100,
        'train_default_ratio': train_data['default.payment.next.month'].mean() *
100,
        'test_default_ratio': test_data['default.payment.next.month'].mean() *
100
    }

    return split_stats, train_data, test_data

def print_split_stats(split_stats):
    """
    Print formatted information about the data split

    Args:
        split_stats (dict): Dictionary containing split statistics
    """
    print("\nData Split Statistics:")
    print(f"Total rows: {split_stats['total_rows']}")
    print(f"Training set: {split_stats['train_rows']} rows "
          f"({split_stats['train_percentage']:.1f}%)")
    print(f"Test set: {split_stats['test_rows']} rows "
          f"({split_stats['test_percentage']:.1f}%)")
    print("\nDefault Payment Distribution:")
    print(f"Training set: {split_stats['train_default_ratio']:.1f}% default
rate")
    print(f"Test set: {split_stats['test_default_ratio']:.1f}% default rate")
```

8.1.5 data_manager.py

```
import os
```

```
import shutil
from config import (
    BASE_DIR,
    RAW_DATA_DIR,
    PROCESSED_DATA_DIR,
    CLEANED_DATA_FILE,
    SCALED_DATA_FILE,
    ENCODED_DATA_FILE,
    FINANCIAL_RATIOS_FILE,
    TEST_DATA_FILE,
    TRAIN_DATA_FILE
)

def setup_data_directories():
    """Create and verify data directory structure"""
    # Create directories
    os.makedirs(RAW_DATA_DIR, exist_ok=True)
    os.makedirs(PROCESSED_DATA_DIR, exist_ok=True)

    # Move any existing processed files to correct directory
    files_to_check = [
        CLEANED_DATA_FILE,
        SCALED_DATA_FILE,
        ENCODED_DATA_FILE,
        FINANCIAL_RATIOS_FILE,
        TEST_DATA_FILE,
        TRAIN_DATA_FILE
    ]

    for file_path in files_to_check:
        filename = os.path.basename(file_path)
        old_path = os.path.join(BASE_DIR, filename)
        if os.path.exists(old_path):
            shutil.move(old_path, file_path)
            print(f"Moved {filename} to {PROCESSED_DATA_DIR}")

    return {
        'raw_dir': RAW_DATA_DIR,
        'processed_dir': PROCESSED_DATA_DIR,
        'files': {
            'cleaned': CLEANED_DATA_FILE,
            'scaled': SCALED_DATA_FILE,
            'encoded': ENCODED_DATA_FILE,
            'financial': FINANCIAL_RATIOS_FILE,
            'test': TEST_DATA_FILE,
```

```
            'train': TRAIN_DATA_FILE
        }
    }
```

8.1.6 config.py

```
import os

# Base directories
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
DATA_DIR = os.path.join(BASE_DIR, 'Data')
RAW_DATA_DIR = os.path.join(DATA_DIR, 'Raw')
PROCESSED_DATA_DIR = os.path.join(DATA_DIR, 'Processed')

# Create directories if they don't exist
os.makedirs(RAW_DATA_DIR, exist_ok=True)
os.makedirs(PROCESSED_DATA_DIR, exist_ok=True)

# Input data file (Raw data)
DATA_FILE = os.path.join(RAW_DATA_DIR, 'UCI_Credit_Card.csv')

# Processed data files
CLEANED_DATA_FILE = os.path.join(PROCESSED_DATA_DIR,
'cleaned_credit_card_data.csv')
SCALED_DATA_FILE = os.path.join(PROCESSED_DATA_DIR,
'scaled_credit_card_data.csv')
ENCODED_DATA_FILE = os.path.join(PROCESSED_DATA_DIR,
'encoded_credit_card_data.csv')
FINANCIAL_RATIOS_FILE = os.path.join(PROCESSED_DATA_DIR,
'financial_ratios_data.csv')
TEST_DATA_FILE = os.path.join(PROCESSED_DATA_DIR, 'test_data.csv')
TRAIN_DATA_FILE = os.path.join(PROCESSED_DATA_DIR, 'train_data.csv')

# Output directories for visualizations
DIAGRAMS_DIR = 'Diagramms'
BILL_AMT_DIR_V = os.path.join(DIAGRAMS_DIR, 'BILL_AMT', 'Verteilung')
BILL_AMT_DIR_A = os.path.join(DIAGRAMS_DIR, 'BILL_AMT', 'Ausreisser')
BILL_Z_SCORE_DIR = os.path.join(DIAGRAMS_DIR, 'BILL_AMT', 'Z-Score')
PAY_AMT_DIR_V = os.path.join(DIAGRAMS_DIR, 'PAY_AMT', 'Verteilung')
PAY_AMT_DIR_A = os.path.join(DIAGRAMS_DIR, 'PAY_AMT', 'Ausreisser')
PAY_Z_SCORE_DIR = os.path.join(DIAGRAMS_DIR, 'PAY_AMT', 'Z-Score')
```

8.1.7 cross_validation.py

```
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, roc_auc_score, classification_report,
confusion_matrix
import numpy as np
import pandas as pd

def perform_cross_validation(model, X, y, n_splits=10, random_state=42):
    """
    Perform cross-validation for any classifier model

    Args:
        model: The classifier model (e.g., XGBoost, Random Forest, Logistic
Regression)
        X: Feature matrix
        y: Target vector
        n_splits: Number of folds for cross-validation
        random_state: Random seed for reproducibility

    Returns:
        dict: Dictionary containing cross-validation results and metrics
    """
    kf = KFold(n_splits=n_splits, shuffle=True, random_state=random_state)

    # Initialize metrics storage
    cv_scores = {
        'accuracy': [],
        'auc': [],
        'confusion_matrices': [],
        'classification_reports': []
    }

    fold_predictions = []

    # Perform k-fold cross-validation
    for fold, (train_idx, val_idx) in enumerate(kf.split(X), 1):
        # Split data for this fold
        X_train_fold = X.iloc[train_idx]
        X_val_fold = X.iloc[val_idx]
        y_train_fold = y.iloc[train_idx]
        y_val_fold = y.iloc[val_idx]
```

```
        # Clone and train the model
        fold_model = model.__class__(**model.get_params())
        fold_model.fit(X_train_fold, y_train_fold)

        # Make predictions
        y_pred = fold_model.predict(X_val_fold)
        y_pred_proba = fold_model.predict_proba(X_val_fold)[:, 1]

        # Calculate metrics
        accuracy = accuracy_score(y_val_fold, y_pred)
        auc = roc_auc_score(y_val_fold, y_pred_proba)
        conf_matrix = confusion_matrix(y_val_fold, y_pred)
        class_report = classification_report(y_val_fold, y_pred)

        # Store results
        cv_scores['accuracy'].append(accuracy)
        cv_scores['auc'].append(auc)
        cv_scores['confusion_matrices'].append(conf_matrix)
        cv_scores['classification_reports'].append(class_report)

        fold_predictions.append({
            'fold': fold,
            'true_values': y_val_fold,
            'predictions': y_pred,
            'probabilities': y_pred_proba
        })

    # Calculate summary statistics
    cv_summary = {
        'mean_accuracy': np.mean(cv_scores['accuracy']),
        'std_accuracy': np.std(cv_scores['accuracy']),
        'mean_auc': np.mean(cv_scores['auc']),
        'std_auc': np.std(cv_scores['auc']),
        'individual_scores': cv_scores,
        'fold_predictions': fold_predictions
    }

    return cv_summary

def print_cv_results(cv_results, model_name="Model"):
    """
    Print formatted cross-validation results

    Args:
        cv_results: Dictionary containing cross-validation results
```

```
        model_name: Name of the model being evaluated
    """
    print(f"\n{model_name} Cross-Validation Results:")
    print("=" * 50)

    # Print accuracy scores
    print("\nAccuracy Scores:")
    for i, score in enumerate(cv_results['individual_scores']['accuracy'], 1):
        print(f"Fold {i}: {score:.4f}")
    print(f"Mean Accuracy: {cv_results['mean_accuracy']:.4f} "
          f"(+/- {cv_results['std_accuracy'] * 2:.4f})")

    # Print AUC scores
    print("\nAUC Scores:")
    for i, score in enumerate(cv_results['individual_scores']['auc'], 1):
        print(f"Fold {i}: {score:.4f}")
    print(f"Mean AUC: {cv_results['mean_auc']:.4f} "
          f"(+/- {cv_results['std_auc'] * 2:.4f})")

    # Print average confusion matrix
    print("\nAverage Confusion Matrix:")
    avg_conf_matrix =
np.mean(cv_results['individual_scores']['confusion_matrices'], axis=0)
    print(avg_conf_matrix.astype(int))

    # Print the classification report for the last fold (as an example)
    print("\nExample Classification Report (Last Fold):")
    print(cv_results['individual_scores']['classification_reports'][-1])
```

8.1.8 adjust_class_weight.py

```
def calculate_scale_pos_weight(y):
    """Calculate scale_pos_weight based on class distribution"""
    neg_count = (y == 0).sum()
    pos_count = (y == 1).sum()
    return neg_count / pos_count
```

8.1.9 descriptive_analysis.py

```
from descriptive_stats import (
    calculate_default_stats,
    calculate_limit_stats,
    calculate_age_stats,
    calculate_sex_percentage,
    calculate_education_percentage,
```

```
    calculate_marriage_percentage
)

def run_descriptive_analysis(data):
    """Run all descriptive statistics analyses"""
    # Calculate statistics
    default_stats = calculate_default_stats(data)
    limit_stats = calculate_limit_stats(data)
    age_stats = calculate_age_stats(data)
    sex_percentage = calculate_sex_percentage(data)
    education_percentage = calculate_education_percentage(data)
    marriage_percentage = calculate_marriage_percentage(data)

    # Print results
    print("Default Payment Statistics:")
    print(f"Anzahl 1: {default_stats['count_1']}")
    print(f"Anzahl 0: {default_stats['count_0']}")
    print(f"In Prozent 1: {default_stats['percentage_1']}")
    print(f"In Prozent 0: {default_stats['percentage_0']}\n")

    print("Limit Balance Statistics:")
    print(f"Min: {limit_stats['min']}")
    print(f"Max: {limit_stats['max']}")
    print(f"Median: {limit_stats['median']}\n")

    print("Age Statistics:")
    print(f"Min: {age_stats['min']}")
    print(f"Max: {age_stats['max']}")
    print(f"Average: {age_stats['average']}\n")

    print("Anteil der Geschlechter:")
    print(sex_percentage)

    print("\nAnteil der Personen mit Universitärem Abschluss:")
    print(f"Percentage of educated individuals: {education_percentage}%")

    print("\nAnteil der Personen mit Heiratsstatus:")
    print(marriage_percentage)
```

8.1.10 descriptive_stats.py

```
import numpy as np
from scipy import stats
```

```python
# Analyse der numerischen Variablen
def calculate_default_stats(data):
    """Calculate default payment statistics"""
    count_1 = data['default.payment.next.month'].sum()
    count_0 = len(data) - count_1
    percentage_1 = count_1 / len(data) * 100
    percentage_0 = count_0 / len(data) * 100

    return {
        'count_1': count_1,
        'count_0': count_0,
        'percentage_1': percentage_1,
        'percentage_0': percentage_0
    }

def calculate_limit_stats(data):
    """Calculate LIMIT_BAL statistics"""
    return {
        'min': data['LIMIT_BAL'].min(),
        'max': data['LIMIT_BAL'].max(),
        'median': data['LIMIT_BAL'].median()
    }

def calculate_age_stats(data):
    """Calculate AGE statistics"""
    return {
        'min': data['AGE'].min(),
        'max': data['AGE'].max(),
        'average': data['AGE'].mean()
    }

# Analyse der kategorischen Variablen
def calculate_sex_percentage(data):
    """Calculate the percentage of each gender in the SEX column

    Args:
        data: DataFrame containing the data

    Returns:
        A dictionary with the percentage of each gender
    """
    total_count = len(data)
    male_count = (data['SEX'] == 1).sum()
    female_count = (data['SEX'] == 2).sum()
```

```
    male_percentage = round((male_count / total_count) * 100, 2)
    female_percentage = round((female_count / total_count) * 100, 2)

    return {
        'male_percentage': male_percentage,
        'female_percentage': female_percentage
    }

def calculate_education_percentage(data):
    """Calculate the percentage of educated individuals in the EDUCATION column

    Args:
        data: DataFrame containing the data

    Returns:
        The percentage of educated individuals
    """
    total_count = len(data)
    educated_count = (data['EDUCATION'] == 2).sum()

    educated_percentage = round((educated_count / total_count) * 100, 2)

    return educated_percentage

def calculate_marriage_percentage(data):
    """Calculate the percentage of each marital status in the MARRIAGE column

    Args:
        data: DataFrame containing the data

    Returns:
        A dictionary with the percentage of each marital status
    """
    total_count = len(data)
    married_count = (data['MARRIAGE'] == 1).sum()
    single_count = (data['MARRIAGE'] == 2).sum()
    others_count = (data['MARRIAGE'] == 3).sum()

    married_percentage = round((married_count / total_count) * 100, 2)
    single_percentage = round((single_count / total_count) * 100, 2)
    others_percentage = round((others_count / total_count) * 100, 2)

    return {
        'married_percentage': married_percentage,
        'single_percentage': single_percentage,
```

```
        'others_percentage': others_percentage
    }

# IQR-Methode
def analyze_outliers_iqr(data, column_prefix, num_columns=6):
    """Generic function to analyze outliers using IQR method

    Args:
        data: DataFrame containing the data
        column_prefix: Prefix of the columns to analyze (e.g., 'BILL_AMT' or
'PAY_AMT')
        num_columns: Number of columns to analyze (default: 6)
    """
    outliers_info = {}

    for i in range(1, num_columns + 1):
        column = f'{column_prefix}{i}'
        Q1 = data[column].quantile(0.25)
        Q3 = data[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Find outliers
        outliers = data[(data[column] < lower_bound) | (data[column] >
upper_bound)]

        outliers_info[column] = {
            'count': len(outliers),
            'Q1': Q1,
            'Q3': Q3,
            'IQR': IQR,
            'lower_bound': lower_bound,
            'upper_bound': upper_bound
        }

    return outliers_info

def analyze_bill_amt_outliers(data):
    """Analyze outliers in BILL_AMT columns using IQR method"""
    return analyze_outliers_iqr(data, 'BILL_AMT')

def analyze_pay_amt_outliers(data):
    """Analyze outliers in PAY_AMT columns using IQR method"""
    return analyze_outliers_iqr(data, 'PAY_AMT')
```

```python
# Z-Score-Methode
def analyze_outliers_z(data, column_prefix, num_columns=6):
    """Analyze outliers in BILL_AMT columns using Z-Score method"""
    outliers_info = {}

    for i in range(1, 7):
        column = f'{column_prefix}{i}'
        z_scores = np.abs(stats.zscore(data[column]))
        threshold = 3

        # Find outliers
        outliers = data[z_scores > threshold]

        outliers_info[column] = {
            'count': len(outliers),
            'threshold': threshold,
            'outliers': outliers[column].values.tolist(),
            'ids': outliers['ID'].values.tolist()}
    return outliers_info

def analyze_pay_amt_outliers_z(data):
    """Analyze outliers in PAY_AMT columns using Z-Score method"""
    return analyze_outliers_z(data, 'PAY_AMT')

def analyze_bill_amt_outliers_z(data):
    """Analyze outliers in BILL_AMT columns using Z-Score method"""
    return analyze_outliers_z(data, 'BILL_AMT')
```

8.1.11 outlier_analysis.py

```python
from descriptive_stats import (
    analyze_bill_amt_outliers,
    analyze_pay_amt_outliers,
    analyze_bill_amt_outliers_z,
    analyze_pay_amt_outliers_z
)

def run_outlier_analysis(data):
    """Run all outlier analyses"""
    # IQR method
    bill_amt_outliers = analyze_bill_amt_outliers(data)
    pay_amt_outliers = analyze_pay_amt_outliers(data)

    print("\nBILL_AMT Outlier Analysis:")
```

```
    for column, info in bill_amt_outliers.items():
        print(f"{column}: {info['count']} Ausreisser")
        print(f"  IQR: {info['IQR']:.2f}")
        print(f"  Bounds: [{info['lower_bound']:.2f},
{info['upper_bound']:.2f}]")

    print("\nPAY_AMT Outlier Analysis:")
    for column, info in pay_amt_outliers.items():
        print(f"{column}: {info['count']} Ausreisser")
        print(f"  IQR: {info['IQR']:.2f}")
        print(f"  Bounds: [{info['lower_bound']:.2f},
{info['upper_bound']:.2f}]")

    # Z-Score method
    bill_amt_outliers_z = analyze_bill_amt_outliers_z(data)
    pay_amt_outliers_z = analyze_pay_amt_outliers_z(data)

    print("\nBILL_AMT Outlier Analysis (Z-Score):")
    for column, info in bill_amt_outliers_z.items():
        print(f"{column}: {info['count']} Ausreisser gefunden (Schwellenwert:
{info['threshold']})")

    print("\nPAY_AMT Outlier Analysis (Z-Score):")
    for column, info in pay_amt_outliers_z.items():
        print(f"{column}: {info['count']} Ausreisser gefunden (Schwellenwert:
{info['threshold']})")
```

8.1.12 visualization_runner.py

```
from visualization import (
    plot_bill_distributions,
    plot_payment_boxplots,
    plot_bill_boxplots,
    plot_payment_distributions,
    plot_bill_outliers_z,
    plot_payment_outliers_z
)

import matplotlib.pyplot as plt
import optuna.visualization as optv
import os
from sklearn.metrics import roc_curve, auc

def create_all_visualizations(data):
```

```
    """Create all visualizations"""
    plot_bill_distributions(data)
    plot_payment_boxplots(data)
    plot_bill_boxplots(data)
    plot_payment_distributions(data)
    plot_bill_outliers_z(data)
    plot_payment_outliers_z(data)

def plot_optuna_results(study, model_name, output_dir='Diagramms/Optuna'):
    """Create and save Optuna visualization plots"""
    os.makedirs(output_dir, exist_ok=True)

    # Plot optimization history
    fig = optv.plot_optimization_history(study)
    fig.write_image(os.path.join(output_dir,
f"optimization_history_{model_name}.png"))

    # Plot parameter importances
    fig = optv.plot_param_importances(study)
    fig.write_image(os.path.join(output_dir,
f"param_importances_{model_name}.png"))

    # Plot parallel coordinate
    fig = optv.plot_parallel_coordinate(study)
    fig.write_image(os.path.join(output_dir,
f"parallel_coordinate_{model_name}.png"))

    plt.close('all')

def plot_feature_importance(feature_importance, model_name,
output_dir='Diagramms/Model'):
    """Plot feature importance from model results

    Args:
        feature_importance: Either a dictionary of feature importances or the raw
importance values
    """
    os.makedirs(output_dir, exist_ok=True)

    if isinstance(feature_importance, dict):
        features = list(feature_importance.keys())
        importance_values = list(feature_importance.values())
    else:
        features = [f"Feature {i}" for i in range(len(feature_importance))]
        importance_values = feature_importance
```

```
    plt.figure(figsize=(12, 6))
    plt.bar(range(len(importance_values)), importance_values)
    plt.xticks(range(len(features)), features, rotation=45, ha='right')
    plt.title('Feature Importance')
    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, f'feature_importance_{model_name}.png'))
    plt.close()

def plot_roc_curve(model_results, X_test, y_test, model_name, output_dir):
    """Create and save ROC curve plot"""
    os.makedirs(output_dir, exist_ok=True)

    # Get predictions
    y_pred_proba = model_results['model'].predict_proba(X_test)[:, 1]

    # Calculate ROC curve points
    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
    roc_auc = auc(fpr, tpr)

    # Create ROC curve plot
    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, color='darkorange', lw=2,
             label=f'ROC curve (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc="lower right")
    plt.grid(True)

    # Save plot with model name
    plt.savefig(os.path.join(output_dir, f'roc_curve_{model_name}.png'))
    plt.close()

def create_model_visualizations(model_results, model_name):
    """Create visualizations for model results"""
    # Plot feature importance
    if 'feature_importance' in model_results:
        plot_feature_importance(model_results['feature_importance'], model_name)

    # Plot optimization history if study exists
    if 'study' in model_results and model_results['study'] is not None:
```

```
        plot_optuna_results(model_results['study'], model_name)
```

8.1.13 visualization.py

```
import os
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from config import (
    BILL_AMT_DIR_V,
    BILL_AMT_DIR_A,
    BILL_Z_SCORE_DIR,
    PAY_AMT_DIR_V,
    PAY_AMT_DIR_A,
    PAY_Z_SCORE_DIR)
from descriptive_stats import analyze_bill_amt_outliers_z,
analyze_pay_amt_outliers_z

def plot_distributions(data, column_prefix, output_dir):
    """Plot distributions for specified columns with logarithmic scaling"""
    os.makedirs(output_dir, exist_ok=True)

    for i in range(1, 7):
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))
        column = f'{column_prefix}{i}'

        # Original distribution
        sns.histplot(data[column], kde=True, bins=30, ax=ax1)
        ax1.set_title(f'Original Verteilung von {column}')
        ax1.set_xlabel('Betrag')
        ax1.set_ylabel('Häufigkeit')

        # Logarithmic distribution
        sns.histplot(np.log1p(data[column]), kde=True, bins=30, ax=ax2)
        ax2.set_title(f'Logarithmische Verteilung von {column}')
        ax2.set_xlabel('Log(Betrag)')
        ax2.set_ylabel('Häufigkeit')

        plt.tight_layout()
        output_path = os.path.join(output_dir, f'{column}_Verteilung.png')
        plt.savefig(output_path)
        plt.close()

def plot_boxplots(data, column_prefix, output_dir):
```

```
    """Plot boxplots for specified columns

    Args:
        data: DataFrame containing the data
        column_prefix: Prefix of columns to plot (e.g., 'BILL_AMT' or 'PAY_AMT')
        output_dir: Directory to save the plots
    """
    os.makedirs(output_dir, exist_ok=True)

    for i in range(1, 7):
        plt.figure()
        column = f'{column_prefix}{i}'
        sns.boxplot(x=data[column])
        plt.title(f'Boxplot von {column}')
        plt.xlabel('Betrag')

        output_path = os.path.join(output_dir, f'{column}_Ausreisser.png')
        plt.savefig(output_path)
        plt.close()

# Wrapper functions to maintain existing interface
def plot_bill_distributions(data):
    plot_distributions(data, 'BILL_AMT', BILL_AMT_DIR_V)

def plot_payment_distributions(data):
    plot_distributions(data, 'PAY_AMT', PAY_AMT_DIR_V)

def plot_bill_boxplots(data):
    plot_boxplots(data, 'BILL_AMT', BILL_AMT_DIR_A)

def plot_payment_boxplots(data):
    plot_boxplots(data, 'PAY_AMT', PAY_AMT_DIR_A)

# z-score-Methode visualisation

def plot_outliers_z(data, column_prefix, output_dir, analyze_func,
show_extreme_outliers=True):
    """Generic function to plot outliers using Z-Score method

    Args:
        data: DataFrame containing the data
        column_prefix: Prefix of columns to plot (e.g., 'BILL_AMT' or 'PAY_AMT')
        output_dir: Directory to save the plots
        analyze_func: Function to analyze outliers
        show_extreme_outliers: Whether to show top 10 extreme outliers
```

```
    """
    os.makedirs(output_dir, exist_ok=True)
    outliers_info = analyze_func(data)

    for column, info in outliers_info.items():
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

        # Original distribution with outliers
        sns.histplot(data[column], kde=True, bins=30, ax=ax1)
        ax1.set_title(f'Original Verteilung von {column}\n'
                      f'Z-Score Schwelle: {info["threshold"]}, '
                      f'Ausreisser: {info["count"]}')
        ax1.set_xlabel('Betrag')
        ax1.set_ylabel('Häufigkeit')

        # Add extreme outliers if requested
        if show_extreme_outliers and 'outliers' in info:
            extreme_outliers = sorted(info['outliers'], reverse=True)[:10]
            for outlier in extreme_outliers:
                ax1.axvline(outlier, color='r', linestyle='--', alpha=0.5)

        # Logarithmic distribution
        sns.histplot(np.log1p(data[column]), kde=True, bins=30, ax=ax2)
        ax2.set_title(f'Logarithmische Verteilung von {column}\n'
                      + ('Mit Top-10 extremen Ausreissern' if
show_extreme_outliers else ''))
        ax2.set_xlabel('Log(Betrag)')
        ax2.set_ylabel('Häufigkeit')

        # Add outliers to logarithmic plot if requested
        if show_extreme_outliers and 'outliers' in info:
            for outlier in extreme_outliers:
                ax2.axvline(np.log1p(outlier), color='r', linestyle='--',
alpha=0.5)

        plt.tight_layout()
        output_path = os.path.join(output_dir, f'{column}_Z-
Score_Ausreisser.png')
        plt.savefig(output_path, bbox_inches='tight', dpi=300)
        plt.close()

def plot_bill_outliers_z(data):
    """Plot BILL_AMT outliers using Z-Score method"""
    plot_outliers_z(data, 'BILL_AMT', BILL_Z_SCORE_DIR,
analyze_bill_amt_outliers_z, True)
```

```
def plot_payment_outliers_z(data):
    """Plot PAY_AMT outliers using Z-Score method"""
    plot_outliers_z(data, 'PAY_AMT', PAY_Z_SCORE_DIR, analyze_pay_amt_outliers_z,
True)
```

8.1.14 xgboost_model.py

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
import xgboost as xgb
from sklearn.model_selection import KFold
import optuna
from sklearn.metrics import accuracy_score, roc_auc_score
from cross_validation import perform_cross_validation, print_cv_results
from adjust_class_weight import calculate_scale_pos_weight

def manual_cross_validation(X, y, model, n_splits=5):
    """Perform manual cross-validation"""
    kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)
    scores = []

    for train_idx, val_idx in kf.split(X):
        X_fold_train, X_fold_val = X.iloc[train_idx], X.iloc[val_idx]
        y_fold_train, y_fold_val = y.iloc[train_idx], y.iloc[val_idx]

        # Train model
        model.fit(X_fold_train, y_fold_train)
        # Get predictions
        y_pred = model.predict(X_fold_val)
        # Calculate accuracy
        score = accuracy_score(y_fold_val, y_pred)
        scores.append(score)

    return np.array(scores)

def get_fixed_params():
    """Return the fixed hyperparameters for XGBoost model"""
    return {
        'max_depth': 7,
        'learning_rate': 0.093835,
        'n_estimators': 122,
```

```
        'min_child_weight': 7,
        'gamma': 0.6276,
        'subsample': 0.9152,
        'colsample_bytree': 0.6543,
        'reg_alpha': 0.4709,
        'reg_lambda': 0.4352,
        'objective': 'binary:logistic',
        'eval_metric': 'auc'
    }

def train_model_with_fixed_params(X_train, y_train, X_test, y_test):
    """Train XGBoost model with fixed parameters"""
    params = get_fixed_params()
    # Add scale_pos_weight to params
    params['scale_pos_weight'] = calculate_scale_pos_weight(y_train)
    model = xgb.XGBClassifier(**params, random_state=42)

    # Perform cross-validation
    cv_results = perform_cross_validation(model, X_train, y_train)
    print_cv_results(cv_results, "XGBoost")

    # Train final model on full training data
    model.fit(X_train, y_train)

    # Make predictions
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)

    return {
        'model': model,
        'best_params': params,
        'accuracy': accuracy_score(y_test, y_pred),
        'auc_score': roc_auc_score(y_test, y_pred_proba[:, 1]),
        'feature_importance': model.feature_importances_,
        'feature_names': X_train.columns.tolist(),
        'prediction_probabilities': y_pred_proba,
        'study': None,  # No study for fixed parameters
        'cv_results': cv_results  # Add cross-validation results
    }

def train_model_with_optuna(X_train, y_train, X_test, y_test, n_trials=300):
    """Train XGBoost model with Optuna optimization"""
    # Calculate scale_pos_weight once
    scale_pos_weight = calculate_scale_pos_weight(y_train)
```

```
    def objective(trial):
        param = {
            'max_depth': trial.suggest_int('max_depth', 3, 9),
            'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.3),
            'n_estimators': trial.suggest_int('n_estimators', 50, 300),
            'min_child_weight': trial.suggest_int('min_child_weight', 1, 7),
            'gamma': trial.suggest_float('gamma', 0, 1.0),
            'subsample': trial.suggest_float('subsample', 0.6, 1.0),
            'colsample_bytree': trial.suggest_float('colsample_bytree', 0.6,
1.0),
            'reg_alpha': trial.suggest_float('reg_alpha', 0, 1.0),
            'reg_lambda': trial.suggest_float('reg_lambda', 0, 1.0),
            'objective': 'binary:logistic',
            'eval_metric': 'auc',
            'scale_pos_weight': trial.suggest_float('scale_pos_weight', 1, 10)
        }

        model = xgb.XGBClassifier(**param, random_state=42)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        return accuracy_score(y_test, y_pred)

    study = optuna.create_study(direction='maximize')
    study.optimize(objective, n_trials=n_trials)

    # Train final model with best parameters
    best_params = study.best_params
    best_params.update({
        'objective': 'binary:logistic',
        'eval_metric': 'auc'
    })

    model = xgb.XGBClassifier(**best_params, random_state=42)
    model.fit(X_train, y_train)

    # Make predictions
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)

    return {
        'model': model,
        'best_params': best_params,
        'study': study,
        'accuracy': accuracy_score(y_test, y_pred),
        'auc_score': roc_auc_score(y_test, y_pred_proba[:, 1]),
```

```
        'feature_importance': model.feature_importances_,
        'feature_names': X_train.columns.tolist(),
        'prediction_probabilities': y_pred_proba
    }

def train_xgboost_model(train_data, test_data, use_optuna):
    """Train XGBoost model with either fixed parameters or Optuna optimization"""
    # Prepare data
    X_train = train_data.drop(['ID', 'default.payment.next.month'], axis=1)
    y_train = train_data['default.payment.next.month']
    X_test = test_data.drop(['ID', 'default.payment.next.month'], axis=1)
    y_test = test_data['default.payment.next.month']

    if use_optuna:
        results = train_model_with_optuna(X_train, y_train, X_test, y_test)
    else:
        results = train_model_with_fixed_params(X_train, y_train, X_test, y_test)

    # Print detailed results with all data
    print_model_results(results, X_train, y_train, X_test, y_test)

    return results

def calculate_cv_scores(model, X_train, y_train, n_splits=5):
    """Calculate cross-validation scores manually"""
    kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)
    scores = []

    for train_idx, val_idx in kf.split(X_train):
        # Split data
        X_fold_train, X_fold_val = X_train.iloc[train_idx], X_train.iloc[val_idx]
        y_fold_train, y_fold_val = y_train.iloc[train_idx], y_train.iloc[val_idx]

        # Train model
        fold_model = xgb.XGBClassifier(**model.get_params())
        fold_model.fit(X_fold_train, y_fold_train)

        # Calculate score
        y_pred = fold_model.predict(X_fold_val)
        scores.append(accuracy_score(y_fold_val, y_pred))

    return np.array(scores)

def print_model_results(model_results, X_train=None, y_train=None, X_test=None,
y_test=None):
```

```
    """Print formatted model results and statistics"""
    print("\nXGBoost Model Results:")
    print("=" * 50)

    # Cross Validation Scores
    if X_train is not None and y_train is not None:
        cv_scores = calculate_cv_scores(model_results['model'], X_train, y_train)
        print(f"\nCross Validation Scores: {cv_scores}")
        print(f"Mean CV Score: {cv_scores.mean():.4f} (+/- {cv_scores.std() *
2:.4f})")

    # Test Performance
    print(f"\nTest Accuracy: {model_results['accuracy']:.4f}")

    # Classification Report
    if X_test is not None and y_test is not None:
        y_pred = model_results['model'].predict(X_test)
        print("\nClassification report:")
        print(classification_report(y_test, y_pred))

        # Confusion Matrix
        print("\nConfusion Matrix:")
        print(confusion_matrix(y_test, y_pred))

    # Feature Importance
    print("\nTop 10 Most Important Features:")
    feature_importance = list(zip(model_results['feature_names'],
                                  model_results['feature_importance']))
    feature_importance.sort(key=lambda x: x[1], reverse=True)

    for feature, importance in feature_importance[:10]:
        print(f" {feature}: {importance:.4f}")

    # Prediction Probabilities (show first few)
    print("\nPrediction probabilities (first 5 samples):")
    print(model_results['prediction_probabilities'][:5])
```

8.1.15 random_forest.py

```
import optuna
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score, KFold
```

```
from sklearn.metrics import (accuracy_score, roc_auc_score,
classification_report,
                             confusion_matrix, roc_curve, auc)
import os
from adjust_class_weight import calculate_scale_pos_weight

def objective(trial, X_train, y_train):
    """Optuna objective for Random Forest hyperparameter tuning"""
    params = {
        'n_estimators': trial.suggest_int('n_estimators', 100, 1000),
        'max_depth': trial.suggest_int('max_depth', 3, 20),
        'min_samples_split': trial.suggest_int('min_samples_split', 2, 20),
        'min_samples_leaf': trial.suggest_int('min_samples_leaf', 1, 10),
        'max_features': trial.suggest_categorical('max_features', ['sqrt',
'log2']),
    }

    model = RandomForestClassifier(**params, random_state=42, n_jobs=-1)
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='roc_auc')
    return scores.mean()

def calculate_detailed_cv_scores(X, y, model_params, n_splits=10):
    """Calculate detailed cross-validation scores"""
    kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)
    accuracy_scores = []
    auc_scores = []
    confusion_matrices = []
    last_fold_report = None

    # Use model_params directly as they should already include base parameters
    for fold, (train_idx, val_idx) in enumerate(kf.split(X), 1):
        # Split data
        X_fold_train, X_fold_val = X.iloc[train_idx], X.iloc[val_idx]
        y_fold_train, y_fold_val = y.iloc[train_idx], y.iloc[val_idx]

        # Train model
        model = RandomForestClassifier(**model_params)
        model.fit(X_fold_train, y_fold_train)

        # Predictions
        y_pred = model.predict(X_fold_val)
        y_pred_proba = model.predict_proba(X_fold_val)[:, 1]

        # Calculate metrics
        accuracy_scores.append(accuracy_score(y_fold_val, y_pred))
```

```
        auc_scores.append(roc_auc_score(y_fold_val, y_pred_proba))
        confusion_matrices.append(confusion_matrix(y_fold_val, y_pred))

        # Store last fold's classification report
        if fold == n_splits:
            last_fold_report = classification_report(y_fold_val, y_pred)

    return {
        'accuracy_scores': accuracy_scores,
        'auc_scores': auc_scores,
        'confusion_matrices': confusion_matrices,
        'last_fold_report': last_fold_report
    }

def save_model_results(model_results, output_dir='Results/RandomForest'):
    """Save model results to a text file"""
    os.makedirs(output_dir, exist_ok=True)
    output_file = os.path.join(output_dir, 'Endresult_RandomForest.txt')

    with open(output_file, 'w') as f:
        # Cross-validation results
        if 'cv_results' in model_results:
            cv = model_results['cv_results']

            f.write("Random Forest Cross-Validation Results:\n")
            f.write("=" * 50 + "\n\n")

            # Accuracy scores
            f.write("Accuracy Scores:\n")
            for i, score in enumerate(cv['accuracy_scores'], 1):
                f.write(f"Fold {i}: {score:.4f}\n")
            mean_acc = np.mean(cv['accuracy_scores'])
            std_acc = np.std(cv['accuracy_scores'])
            f.write(f"Mean Accuracy: {mean_acc:.4f} (+/- {std_acc*2:.4f})\n\n")

            # AUC scores
            f.write("AUC Scores:\n")
            for i, score in enumerate(cv['auc_scores'], 1):
                f.write(f"Fold {i}: {score:.4f}\n")
            mean_auc = np.mean(cv['auc_scores'])
            std_auc = np.std(cv['auc_scores'])
            f.write(f"Mean AUC: {mean_auc:.4f} (+/- {std_auc*2:.4f})\n\n")

            # Average confusion matrix
            avg_conf_matrix = np.mean(cv['confusion_matrices'], axis=0)
```

```
            f.write("Average Confusion Matrix:\n")
            f.write(str(avg_conf_matrix.astype(int)) + "\n\n")

            # Last fold classification report
            f.write("Example Classification Report (Last Fold):\n")
            f.write(cv['last_fold_report'] + "\n\n")

        # Final model results
        f.write("Random Forest Model Results:\n")
        f.write("=" * 50 + "\n\n")

        f.write(f"Test Accuracy: {model_results['accuracy']:.4f}\n\n")
        f.write("Classification report:\n")
        f.write(model_results['classification_report'] + "\n\n")

        # Feature importance
        f.write("Top 10 Most Important Features:\n")
        sorted_features = sorted(model_results['feature_importance'].items(),
                                 key=lambda x: x[1], reverse=True)[:10]
        for feature, importance in sorted_features:
            f.write(f" {feature}: {importance:.4f}\n")

        # Model performance summary
        f.write("\nModel Performance:\n")
        f.write(f"Accuracy: {model_results['accuracy']:.4f}\n")
        f.write(f"AUC Score: {model_results['auc_score']:.4f}\n")

def train_random_forest_model(train_data, test_data, use_optuna):
    """Train and evaluate Random Forest model with optional Optuna tuning"""
    # Prepare data
    X_train = train_data.drop(['ID', 'default.payment.next.month'], axis=1)
    y_train = train_data['default.payment.next.month']
    X_test = test_data.drop(['ID', 'default.payment.next.month'], axis=1)
    y_test = test_data['default.payment.next.month']

    # Calculate class weight
    scale_pos_weight = calculate_scale_pos_weight(y_train)
    class_weight = {0: 1, 1: scale_pos_weight}

    # Base parameters that will be used in all cases
    base_params = {
        'random_state': 42,
        'n_jobs': -1,
        'class_weight': class_weight
    }
```

```
if use_optuna:
    # Hyperparameter tuning with Optuna
    study = optuna.create_study(direction='maximize')
    study.optimize(lambda trial: objective(trial, X_train, y_train),
                   n_trials=17, show_progress_bar=True)

    # Combine Optuna's best params with base params
    best_params = {**base_params, **study.best_params}
else:
    study = None
    best_params = base_params

# Create and train model with parameters
model = RandomForestClassifier(**best_params)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]

# Calculate metrics
accuracy = accuracy_score(y_test, y_pred)
auc_score = roc_auc_score(y_test, y_pred_proba)
class_report = classification_report(y_test, y_pred)

# Feature importance with proper column names
feature_importance = dict(zip(X_train.columns, model.feature_importances_))

# Calculate detailed cross-validation scores
cv_results = calculate_detailed_cv_scores(X_train, y_train, best_params)

# Add CV results to model results
results = {
    'model': model,
    'accuracy': accuracy,
    'auc_score': auc_score,
    'classification_report': class_report,
    'feature_importance': feature_importance,
    'best_params': best_params,
    'y_pred_proba': y_pred_proba,
    'study': study,
    'cv_results': cv_results
}
```

```
    # Save results to file
    save_model_results(results)

    return results
```

8.1.16 logistic_regression.py

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold
from sklearn.metrics import (accuracy_score, roc_auc_score,
classification_report,
                             confusion_matrix)
import optuna
import os
from adjust_class_weight import calculate_scale_pos_weight

def objective(trial, X_train, y_train):
    """Optuna objective for Logistic Regression hyperparameter tuning"""
    # First choose solver since it affects which penalties are valid
    solver = trial.suggest_categorical('solver', ['lbfgs', 'liblinear', 'saga'])

    # Select penalty based on solver
    if solver == 'lbfgs':
        penalty = 'l2'  # lbfgs only supports l2 penalty
    else:
        penalty = trial.suggest_categorical('penalty', ['l1', 'l2'])

    params = {
        'C': trial.suggest_float('C', 0.001, 10.0, log=True),
        'max_iter': trial.suggest_int('max_iter', 500, 2000),
        'solver': solver,
        'penalty': penalty
    }

    model = LogisticRegression(**params, random_state=42)
    kf = KFold(n_splits=5, shuffle=True, random_state=42)
    scores = []

    for train_idx, val_idx in kf.split(X_train):
        X_fold_train, X_fold_val = X_train.iloc[train_idx], X_train.iloc[val_idx]
        y_fold_train, y_fold_val = y_train.iloc[train_idx], y_train.iloc[val_idx]

        model.fit(X_fold_train, y_fold_train)
```

```
        y_pred_proba = model.predict_proba(X_fold_val)[:, 1]
        scores.append(roc_auc_score(y_fold_val, y_pred_proba))

    return np.mean(scores)

def calculate_detailed_cv_scores(X, y, model_params, n_splits=10):
    """Calculate detailed cross-validation scores"""
    kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)
    accuracy_scores = []
    auc_scores = []
    confusion_matrices = []
    last_fold_report = None

    for fold, (train_idx, val_idx) in enumerate(kf.split(X), 1):
        X_fold_train, X_fold_val = X.iloc[train_idx], X.iloc[val_idx]
        y_fold_train, y_fold_val = y.iloc[train_idx], y.iloc[val_idx]

        model = LogisticRegression(**model_params)
        model.fit(X_fold_train, y_fold_train)

        y_pred = model.predict(X_fold_val)
        y_pred_proba = model.predict_proba(X_fold_val)[:, 1]

        accuracy_scores.append(accuracy_score(y_fold_val, y_pred))
        auc_scores.append(roc_auc_score(y_fold_val, y_pred_proba))
        confusion_matrices.append(confusion_matrix(y_fold_val, y_pred))

        if fold == n_splits:
            last_fold_report = classification_report(y_fold_val, y_pred)

    return {
        'accuracy_scores': accuracy_scores,
        'auc_scores': auc_scores,
        'confusion_matrices': confusion_matrices,
        'last_fold_report': last_fold_report
    }

def save_model_results(model_results, output_dir='Results/LogisticRegression'):
    """Save model results to a text file"""
    os.makedirs(output_dir, exist_ok=True)
    output_file = os.path.join(output_dir, 'Endresult_LogisticRegression.txt')

    with open(output_file, 'w') as f:
        f.write("Logistic Regression Results:\n")
        f.write("=" * 50 + "\n\n")
```

```
        if 'cv_results' in model_results:
            cv = model_results['cv_results']

            f.write("Cross-Validation Results:\n")
            f.write("-" * 30 + "\n\n")

            # Write accuracy scores
            mean_acc = np.mean(cv['accuracy_scores'])
            std_acc = np.std(cv['accuracy_scores'])
            f.write(f"Mean Accuracy: {mean_acc:.4f} (+/- {std_acc*2:.4f})\n")

            # Write AUC scores
            mean_auc = np.mean(cv['auc_scores'])
            std_auc = np.std(cv['auc_scores'])
            f.write(f"Mean AUC: {mean_auc:.4f} (+/- {std_auc*2:.4f})\n\n")

        f.write("Test Set Results:\n")
        f.write("-" * 30 + "\n")
        f.write(f"Test Accuracy: {model_results['accuracy']:.4f}\n")
        f.write(f"Test AUC Score: {model_results['auc_score']:.4f}\n\n")

        f.write("Classification Report:\n")
        f.write(model_results['classification_report'] + "\n")

        if 'feature_importance' in model_results:
            f.write("\nFeature Coefficients:\n")
            for feature, coef in model_results['feature_importance'].items():
                f.write(f"{feature}: {coef:.4f}\n")

def train_logistic_regression_model(train_data, test_data, use_optuna):
    """Train and evaluate Logistic Regression model"""
    # Prepare data
    X_train = train_data.drop(['ID', 'default.payment.next.month'], axis=1)
    y_train = train_data['default.payment.next.month']
    X_test = test_data.drop(['ID', 'default.payment.next.month'], axis=1)
    y_test = test_data['default.payment.next.month']

    # Calculate class weight
    class_weight = {0: 1, 1: calculate_scale_pos_weight(y_train)}

    base_params = {
        'random_state': 42,
        'class_weight': class_weight
    }
```

```
if use_optuna:
    study = optuna.create_study(direction='maximize')
    study.optimize(lambda trial: objective(trial, X_train, y_train),
                   n_trials=100)

    best_params = {**base_params, **study.best_params}
else:
    study = None
    best_params = {**base_params, 'C': 1.0, 'max_iter': 200,
                   'solver': 'lbfgs', 'penalty': 'l2'}

# Train final model
model = LogisticRegression(**best_params)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]

# Calculate metrics
accuracy = accuracy_score(y_test, y_pred)
auc_score = roc_auc_score(y_test, y_pred_proba)
class_report = classification_report(y_test, y_pred)

# Get feature importance (coefficients)
feature_importance = dict(zip(X_train.columns, model.coef_[0]))

# Calculate detailed cross-validation scores
cv_results = calculate_detailed_cv_scores(X_train, y_train, best_params)

results = {
    'model': model,
    'accuracy': accuracy,
    'auc_score': auc_score,
    'classification_report': class_report,
    'feature_importance': feature_importance,
    'best_params': best_params,
    'y_pred_proba': y_pred_proba,
    'study': study,
    'cv_results': cv_results
}

# Save results
save_model_results(results)
```

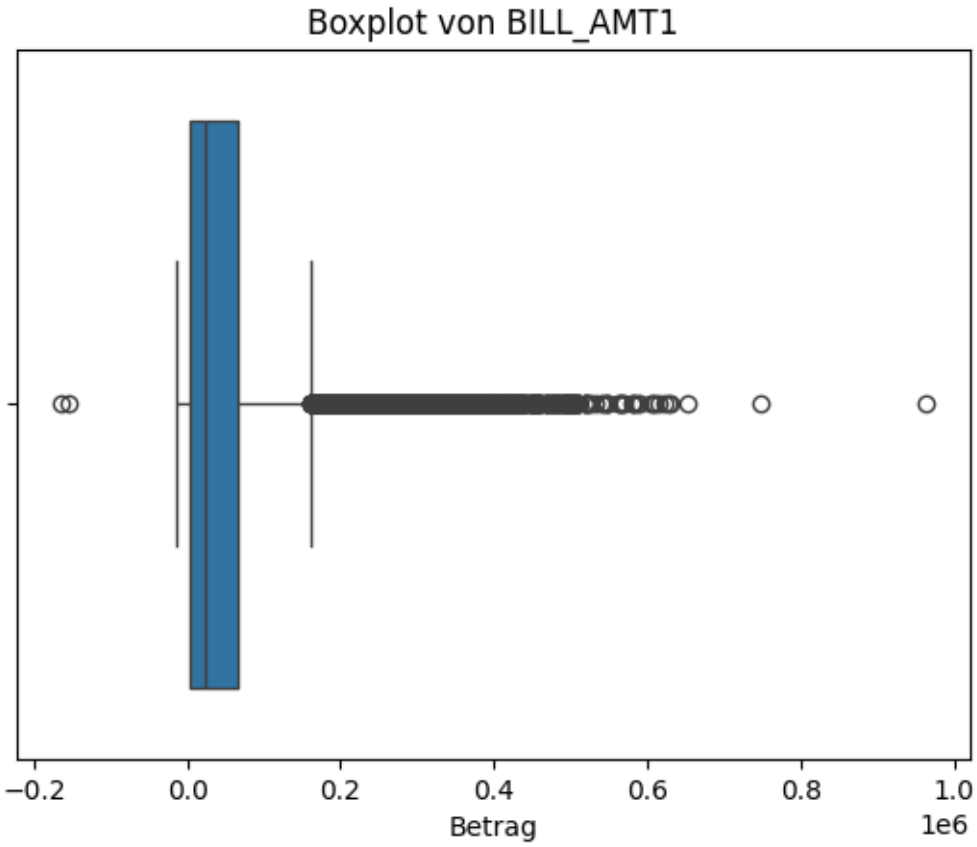
```
    return results
```

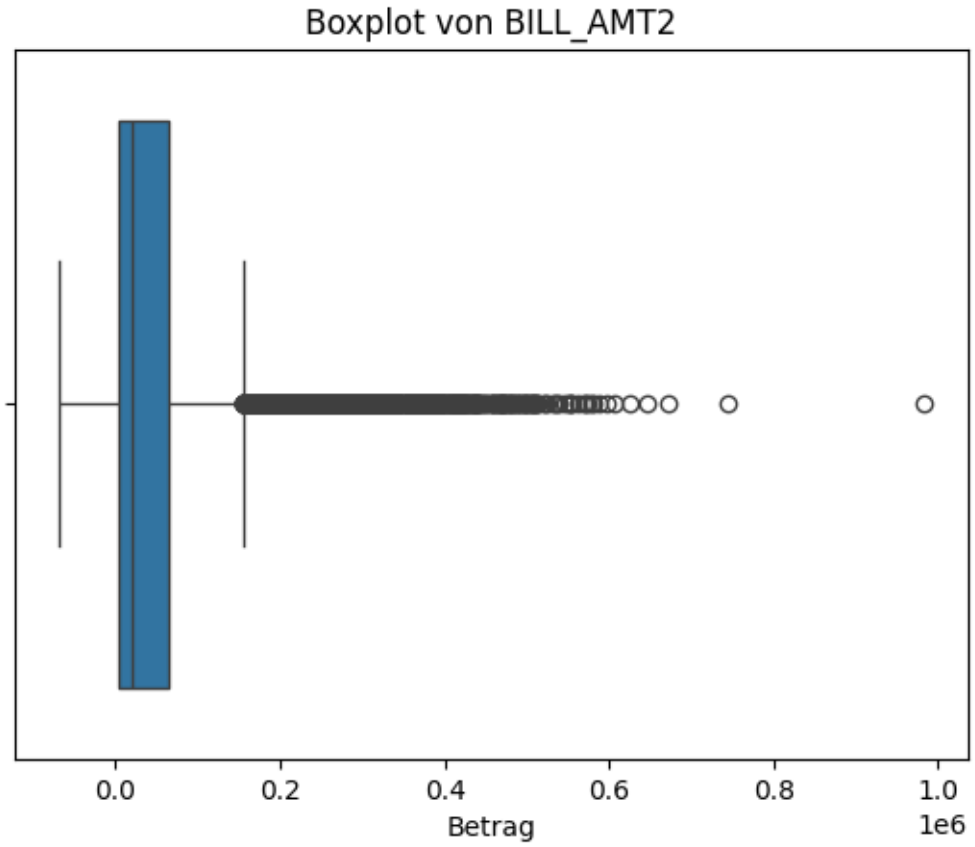
8.2 ZUSÄTZLICHE GRAFIKEN UND TABELLEN

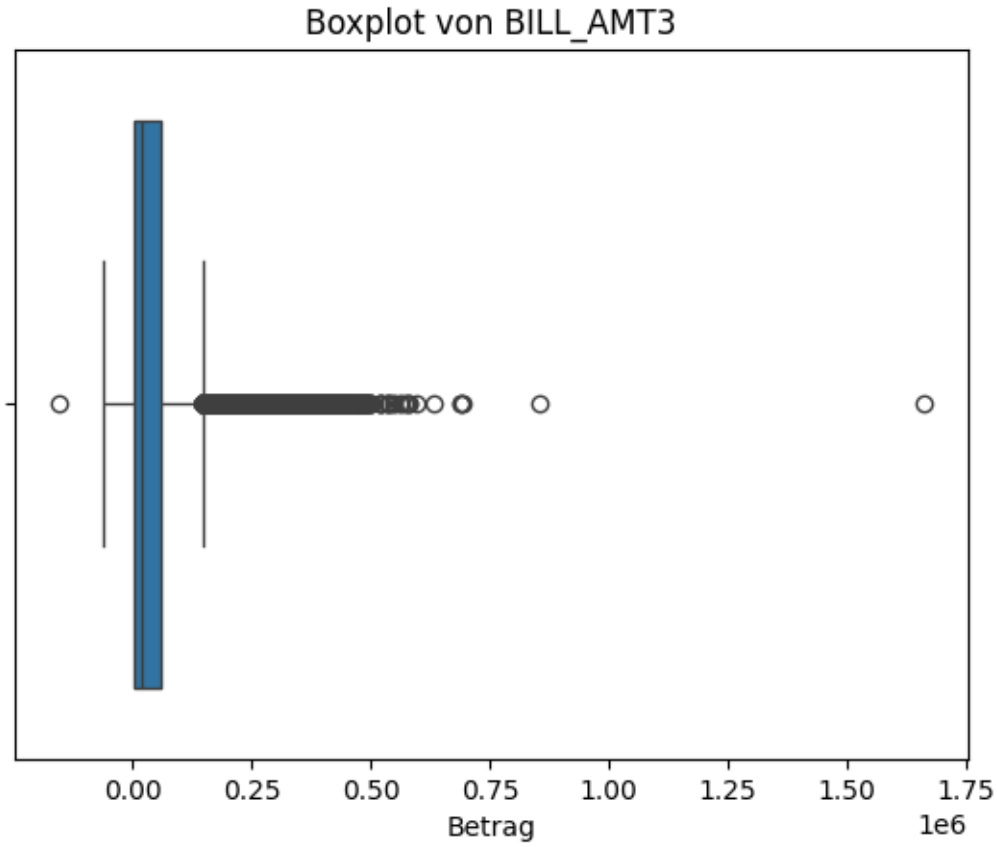
8.2.1 Explorative Datenanalyse (EDA) Diagramme

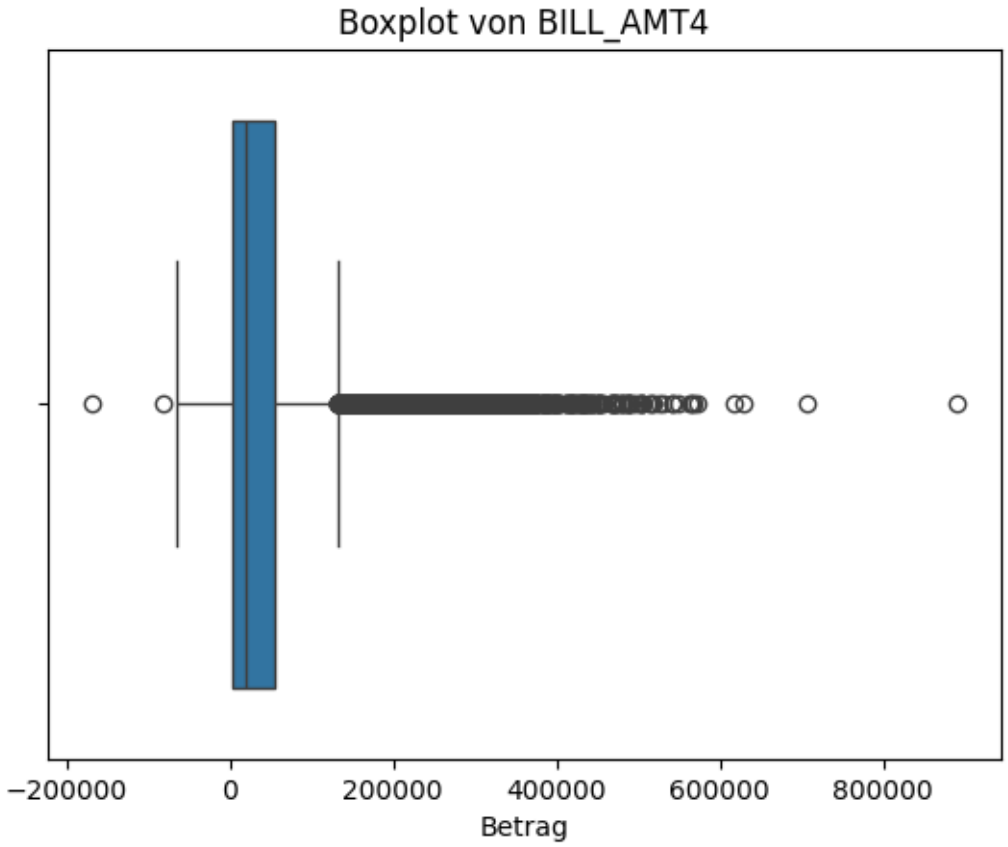
8.2.1.1 BILL_AMT

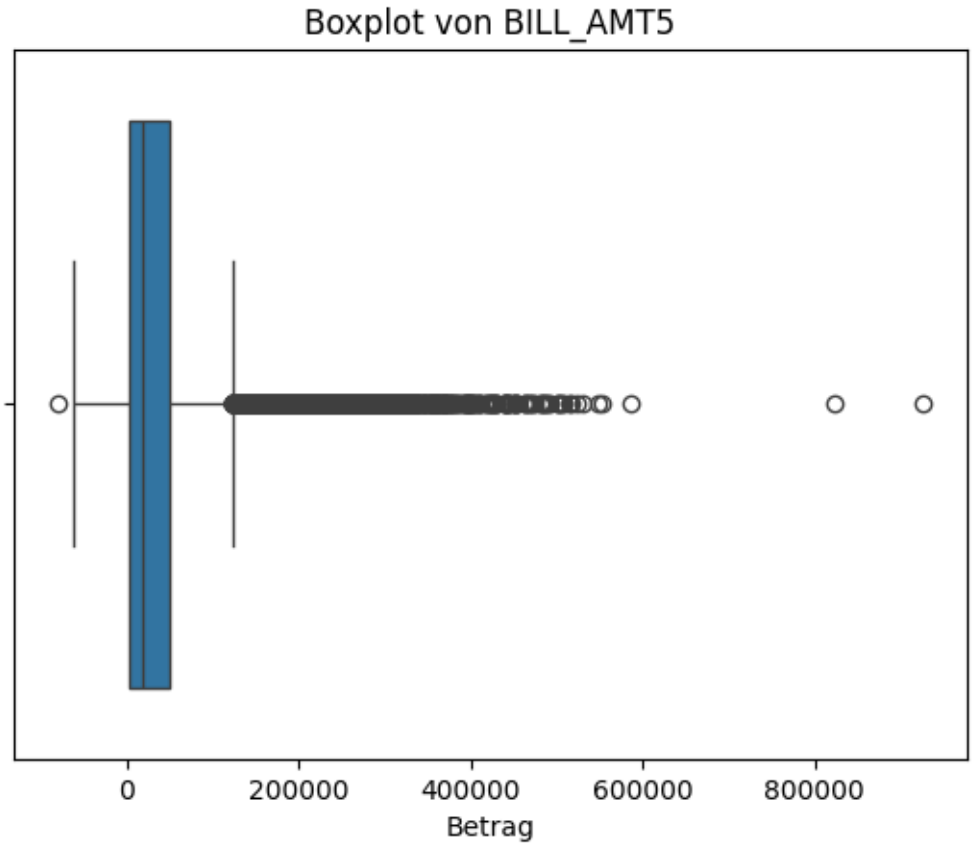
8.2.1.1.1 Ausreisser

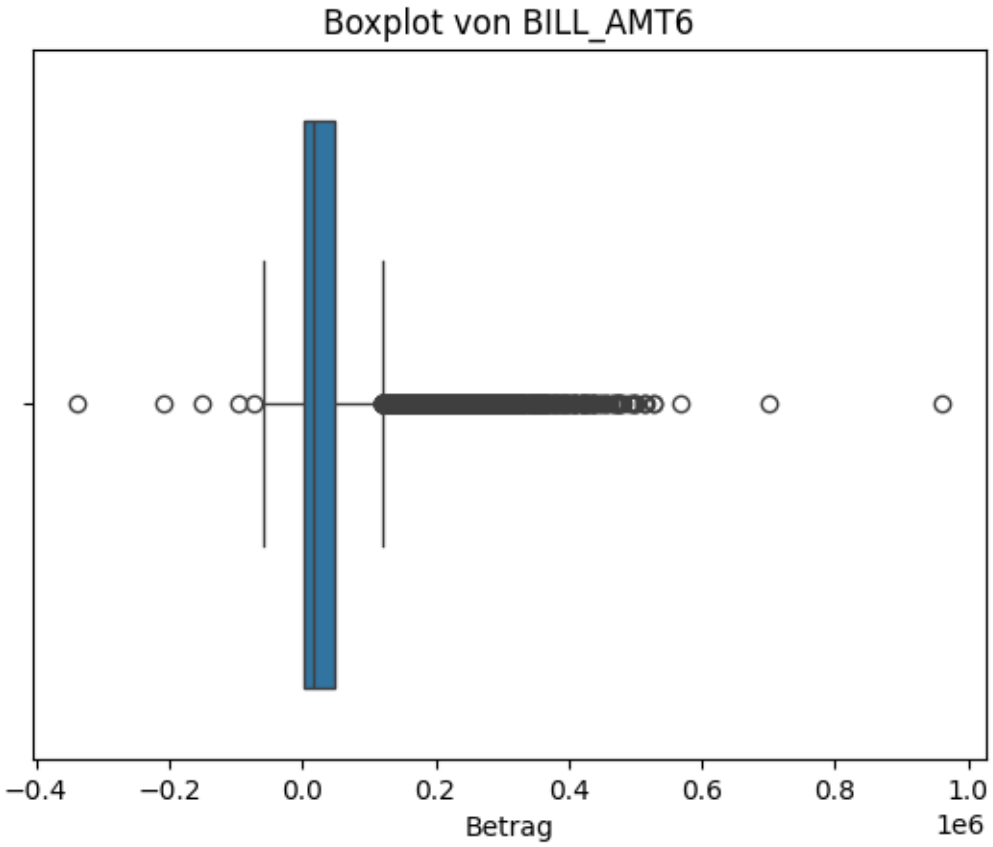

Boxplot von BILL_AMT1
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
Betrag
1e6


Boxplot von BILL_AMT2
0.0
0.2
0.4
0.6
0.8
1.0
Betrag
1e6

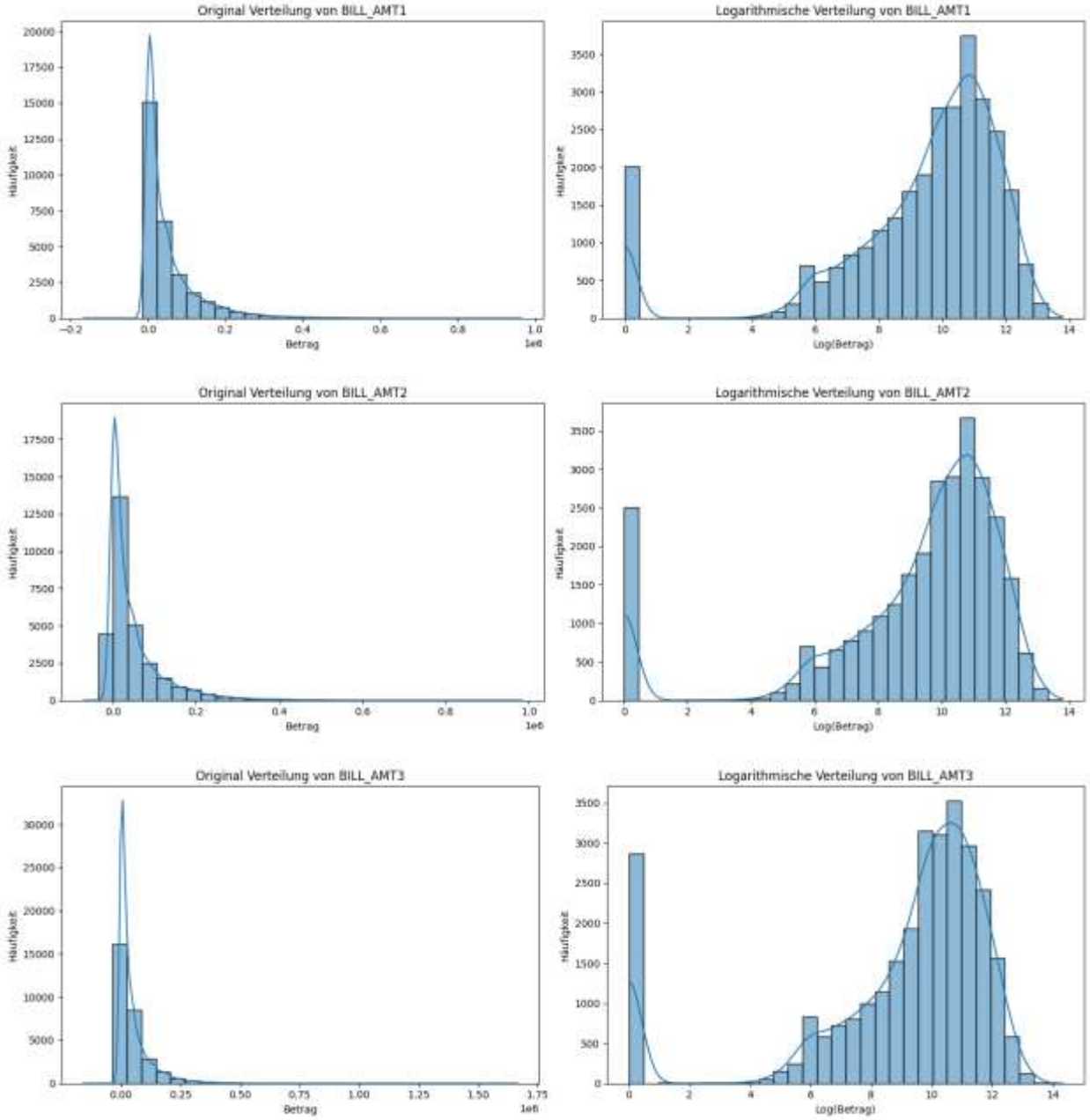

Boxplot von BILL_AMT3
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
Betrag
1e6

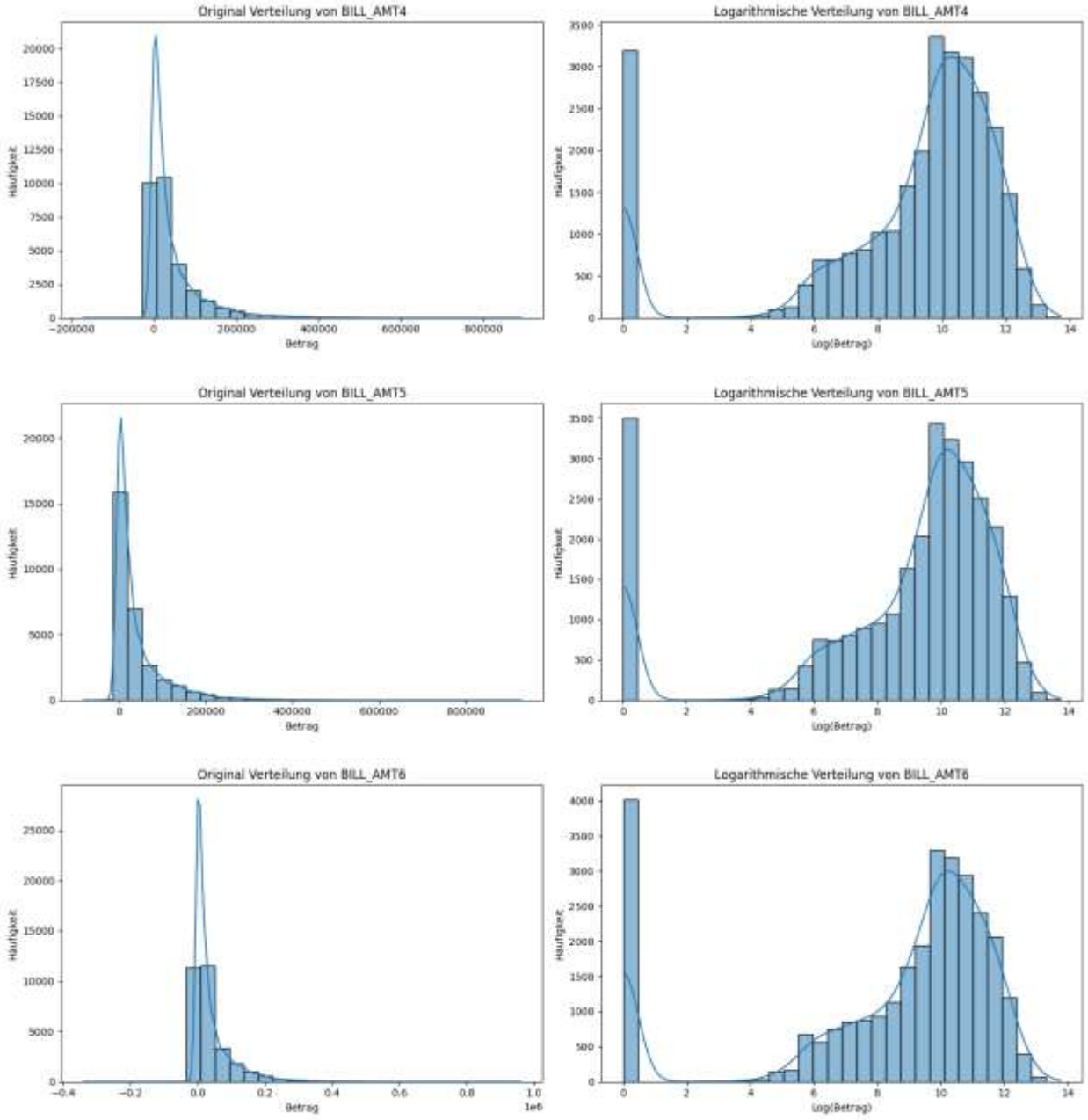

Boxplot von BILL_AMT4
−200000
0
200000
400000
600000
800000
Betrag


Boxplot von BILL_AMT5
0
200000
400000
600000
800000
Betrag

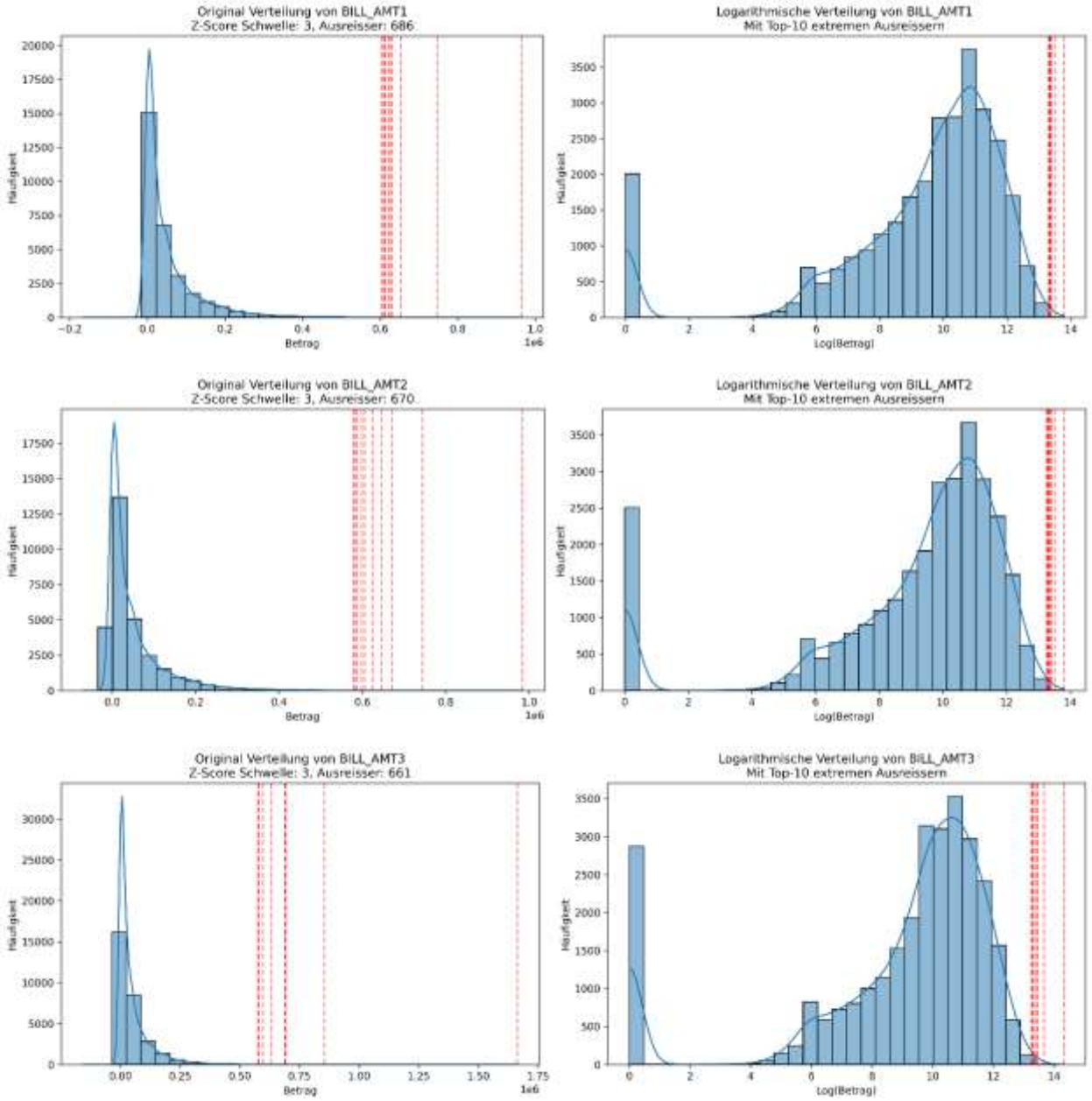

Boxplot von BILL_AMT6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
Betrag
1e6

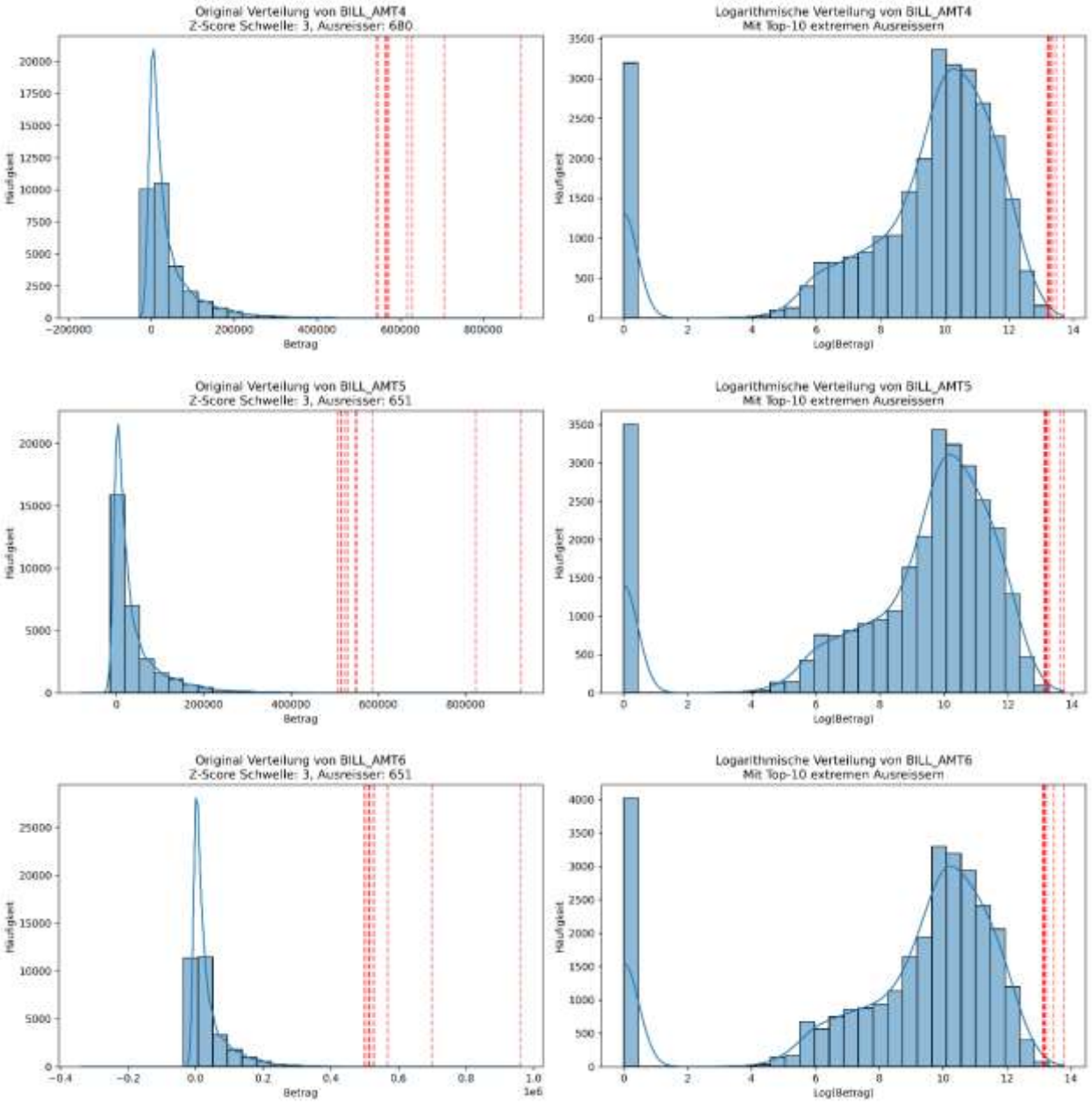
8.2.1.1.2 Verteilung




Original Verteilung von BILL_AMT4
Häufigkeit
Betrag
Logarithmische Verteilung von BILL_AMT4
Häufigkeit
Log(Betrag)
Original Verteilung von BILL_AMT5
Häufigkeit
Betrag
Logarithmische Verteilung von BILL_AMT5
Häufigkeit
Log(Betrag)
Original Verteilung von BILL_AMT6
Häufigkeit
Betrag
1e6
Logarithmische Verteilung von BILL_AMT6
Häufigkeit
Log(Betrag)

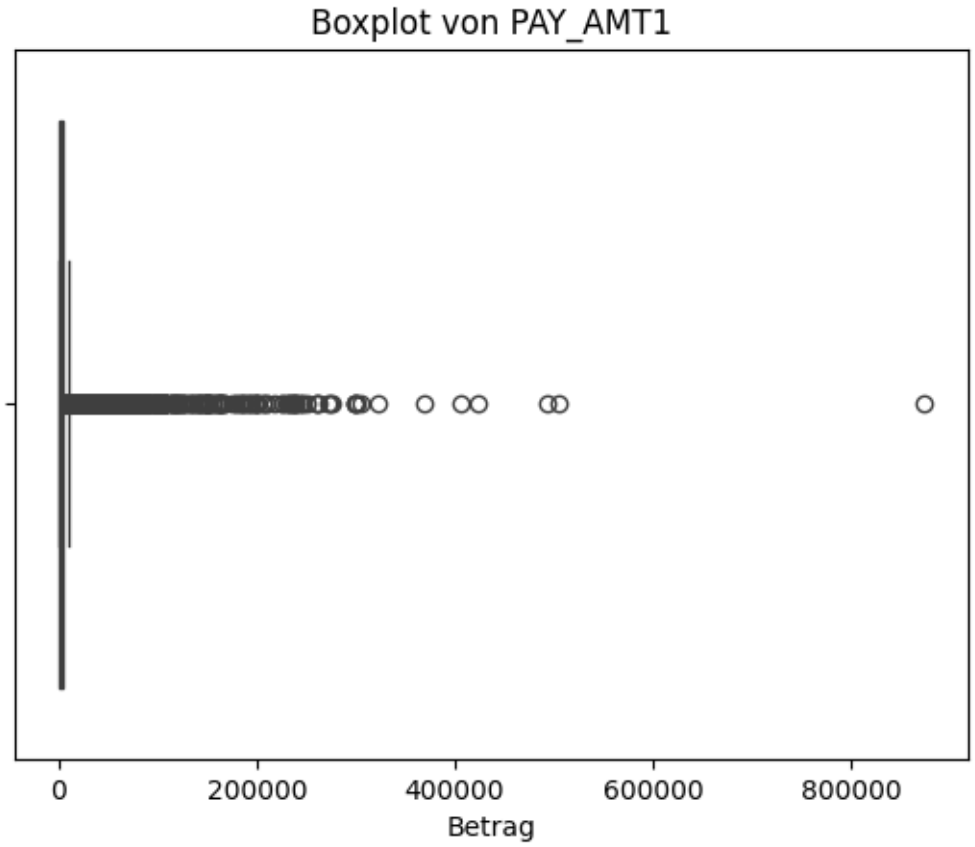
8.2.1.1.3 Z-Score

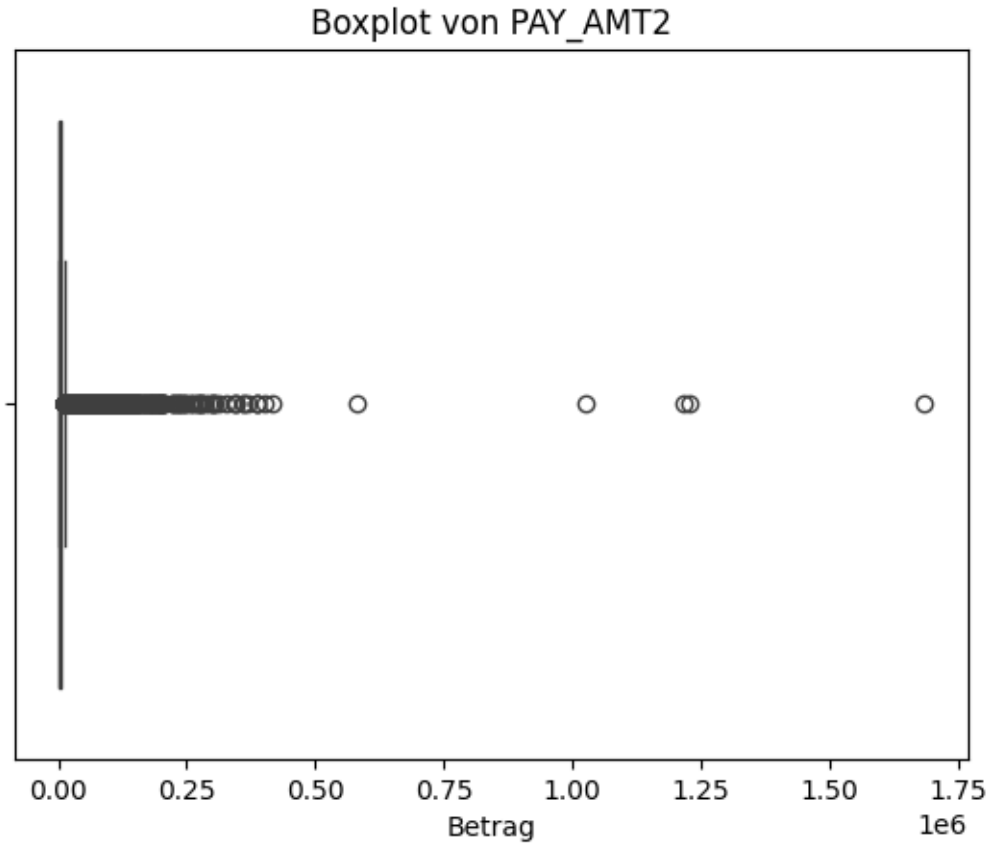


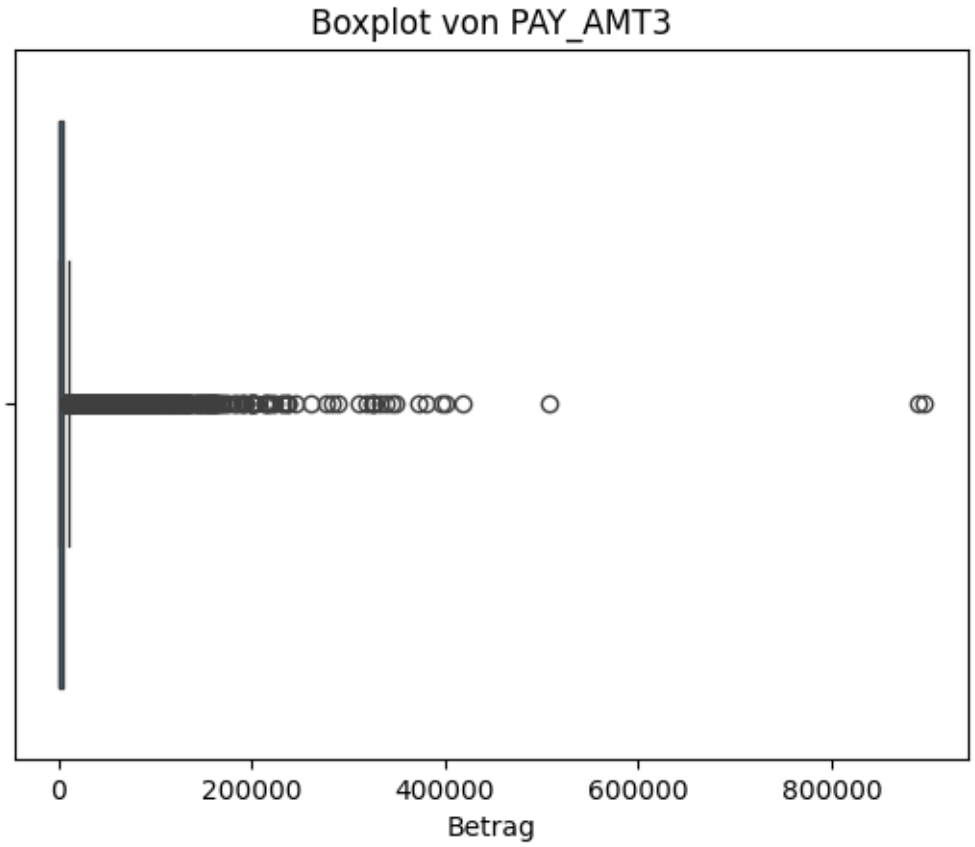

Original Verteilung von BILL_AMT4
Z-Score Schwelle: 3, Ausreisser: 680
Häufigkeit
Betrag
Logarithmische Verteilung von BILL_AMT4
Mit Top-10 extremen Ausreissern
Häufigkeit
LogBetrag
Original Verteilung von BILL_AMT5
Z-Score Schwelle: 3, Ausreisser: 651
Häufigkeit
Betrag
Logarithmische Verteilung von BILL_AMT5
Mit Top-10 extremen Ausreissern
Häufigkeit
LogBetrag
Original Verteilung von BILL_AMT6
Z-Score Schwelle: 3, Ausreisser: 651
Häufigkeit
Betrag
1e6
Logarithmische Verteilung von BILL_AMT6
Mit Top-10 extremen Ausreissern
Häufigkeit
LogBetrag

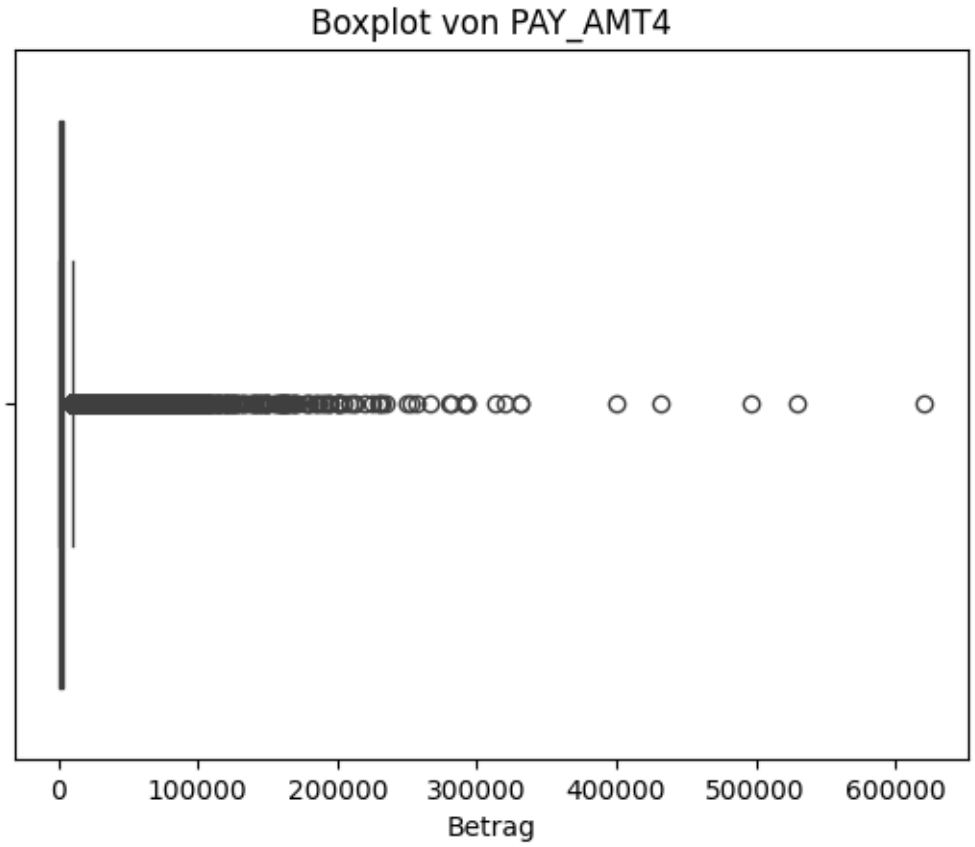
8.2.1.2 PAY_AMT

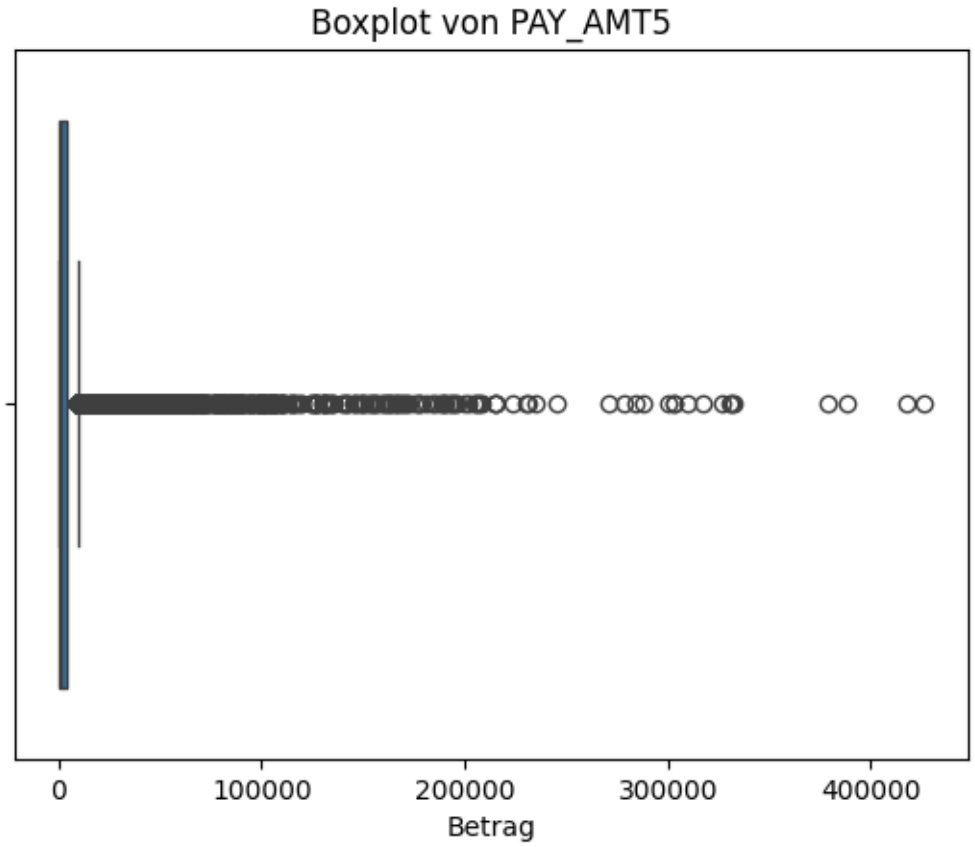
8.2.1.2.1 PAY_AMT

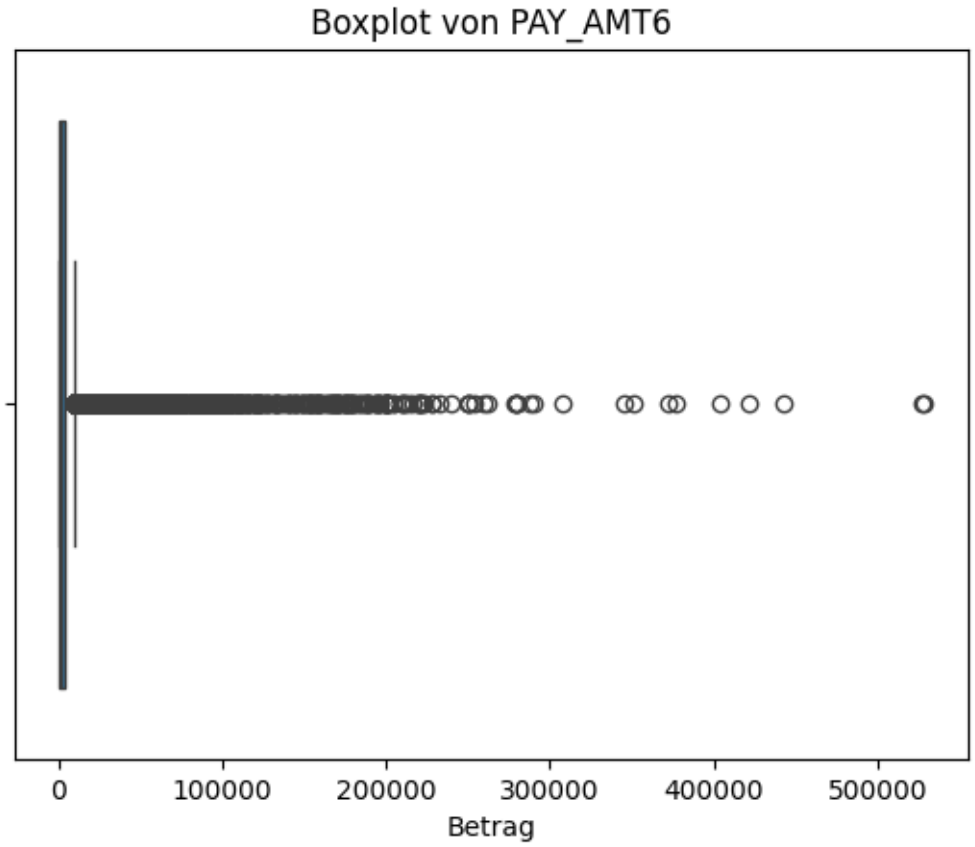

Boxplot von PAY_AMT1
0
200000
400000
600000
800000
Betrag


Boxplot von PAY_AMT2
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
Betrag
1e6

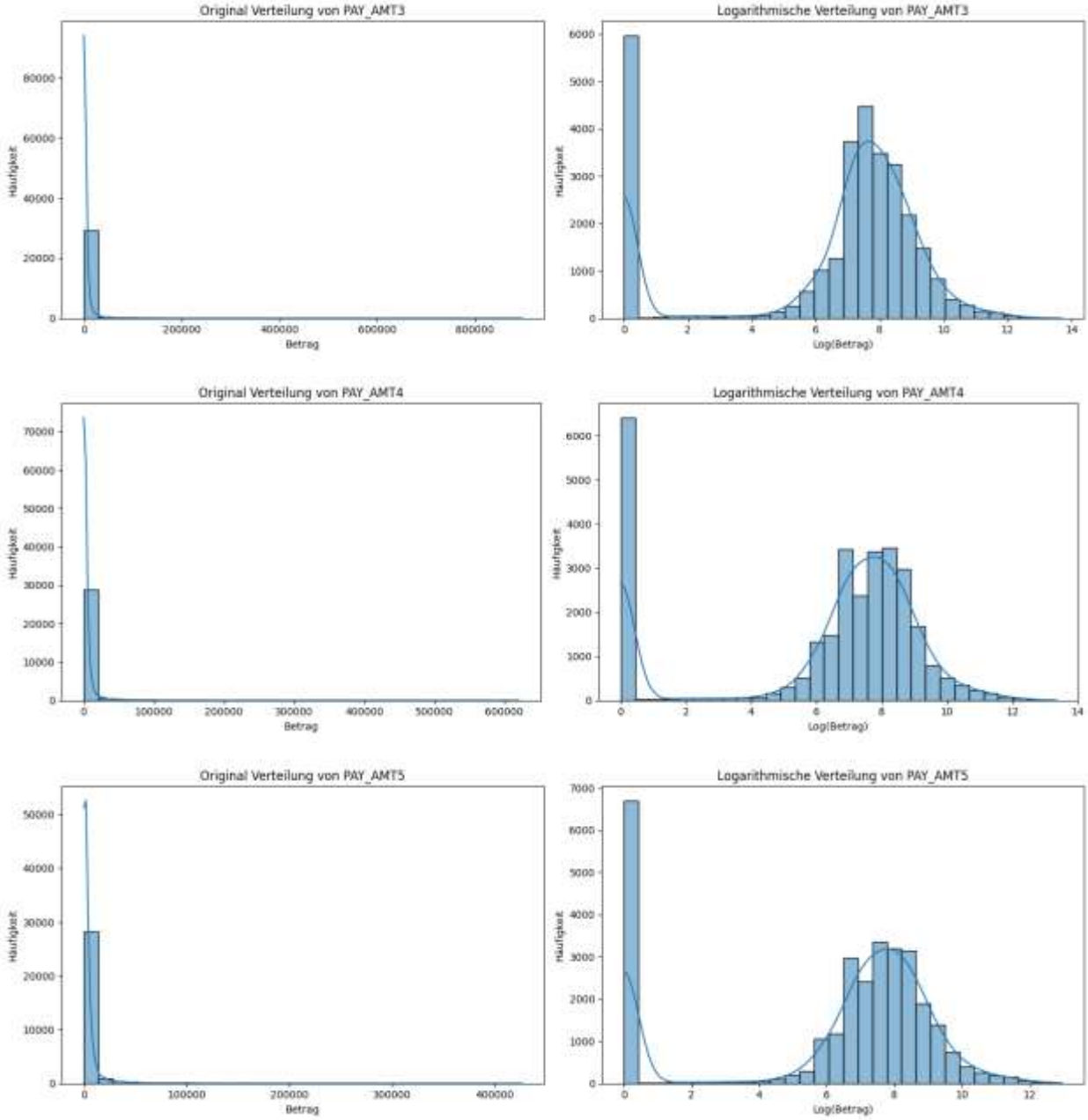

Boxplot von PAY_AMT3
0
200000
400000
600000
800000
Betrag

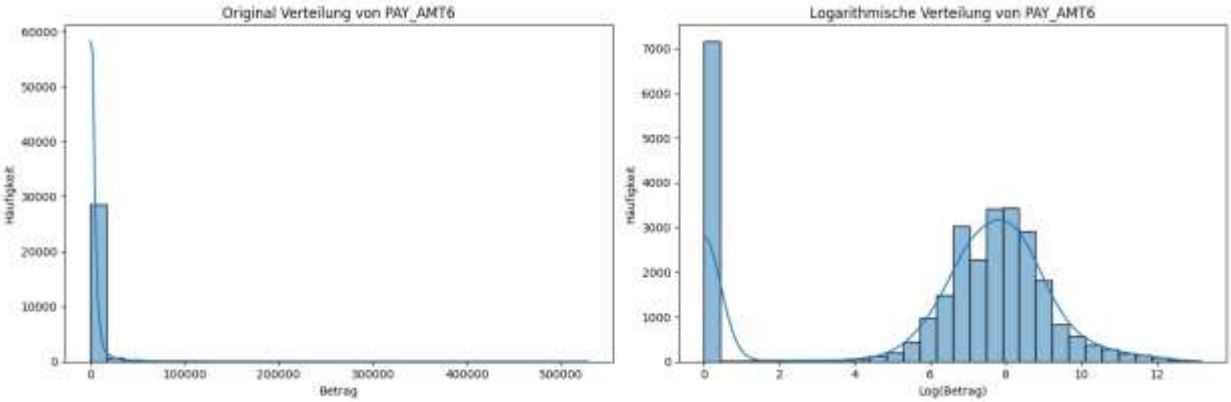

Boxplot von PAY_AMT4
0
100000
200000
300000
400000
500000
600000
Betrag

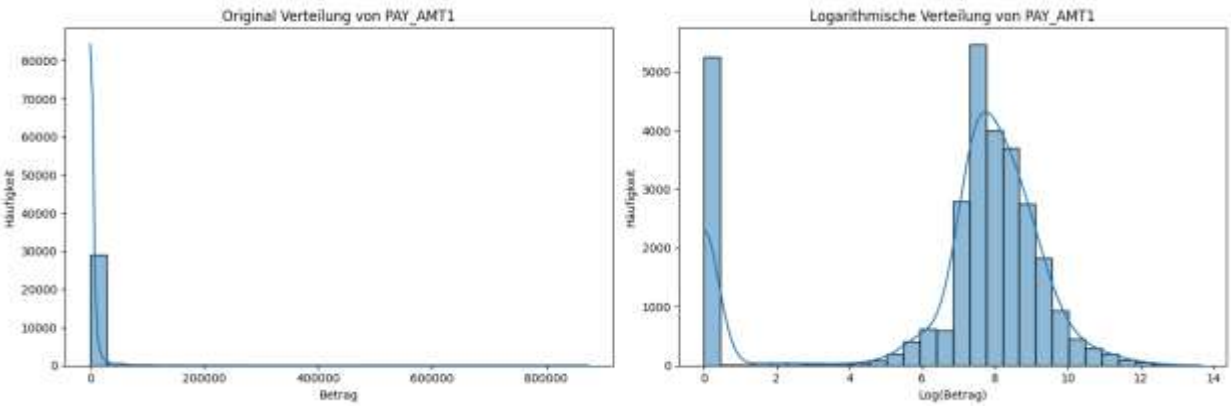

Boxplot von PAY_AMT5
0
100000
200000
300000
400000
Betrag

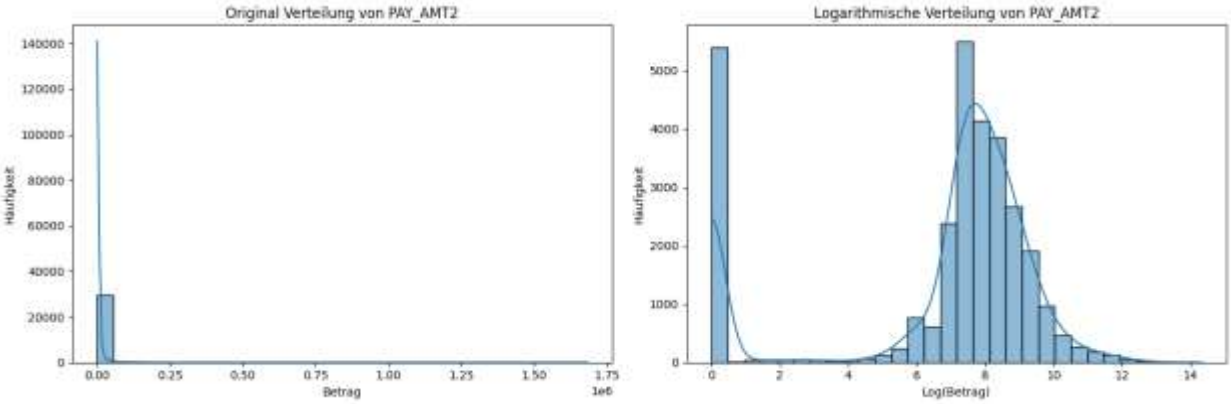

Boxplot von PAY_AMT6
0
100000
200000
300000
400000
500000
Betrag

8.2.1.2.2 Verteilung

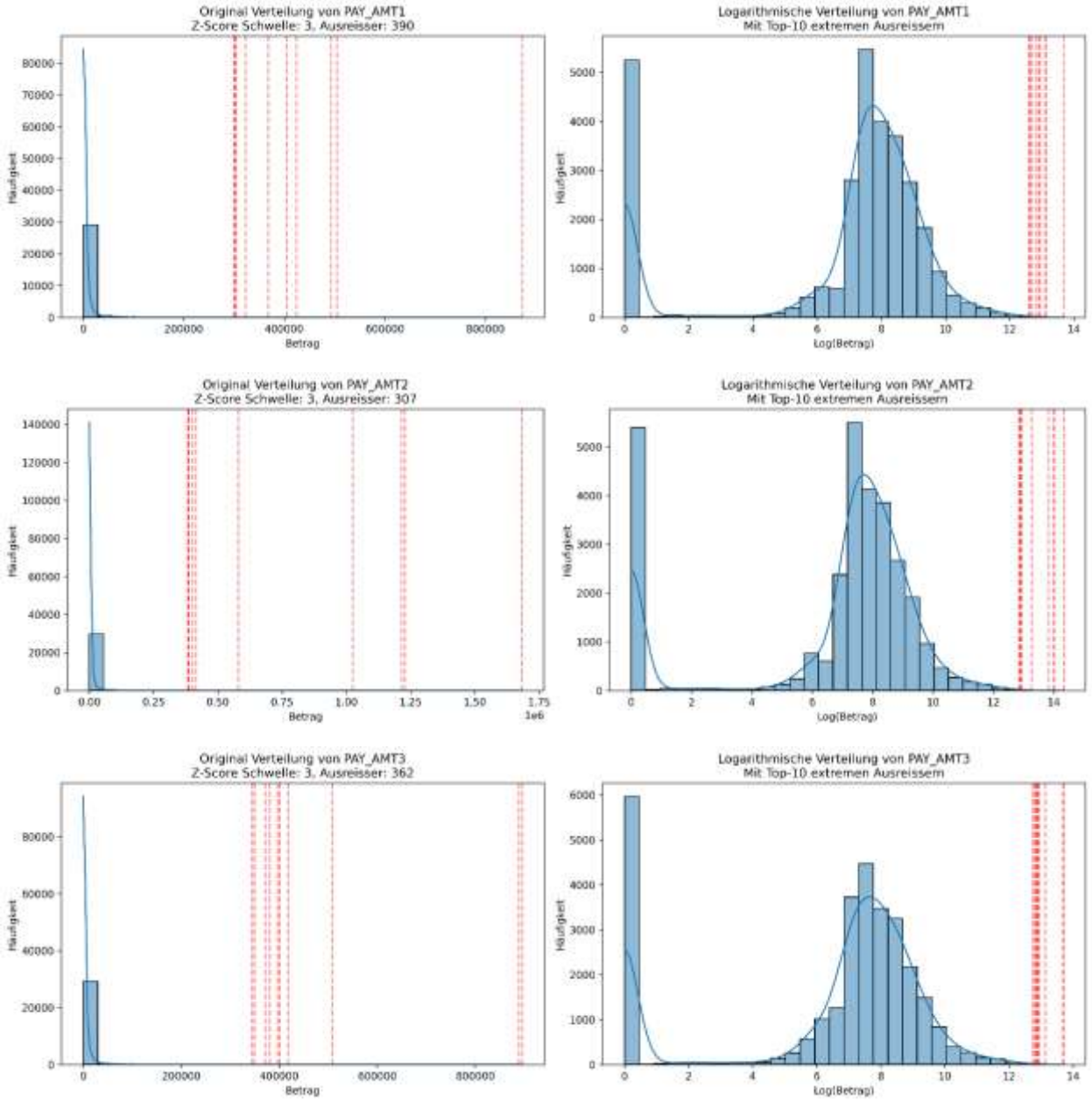


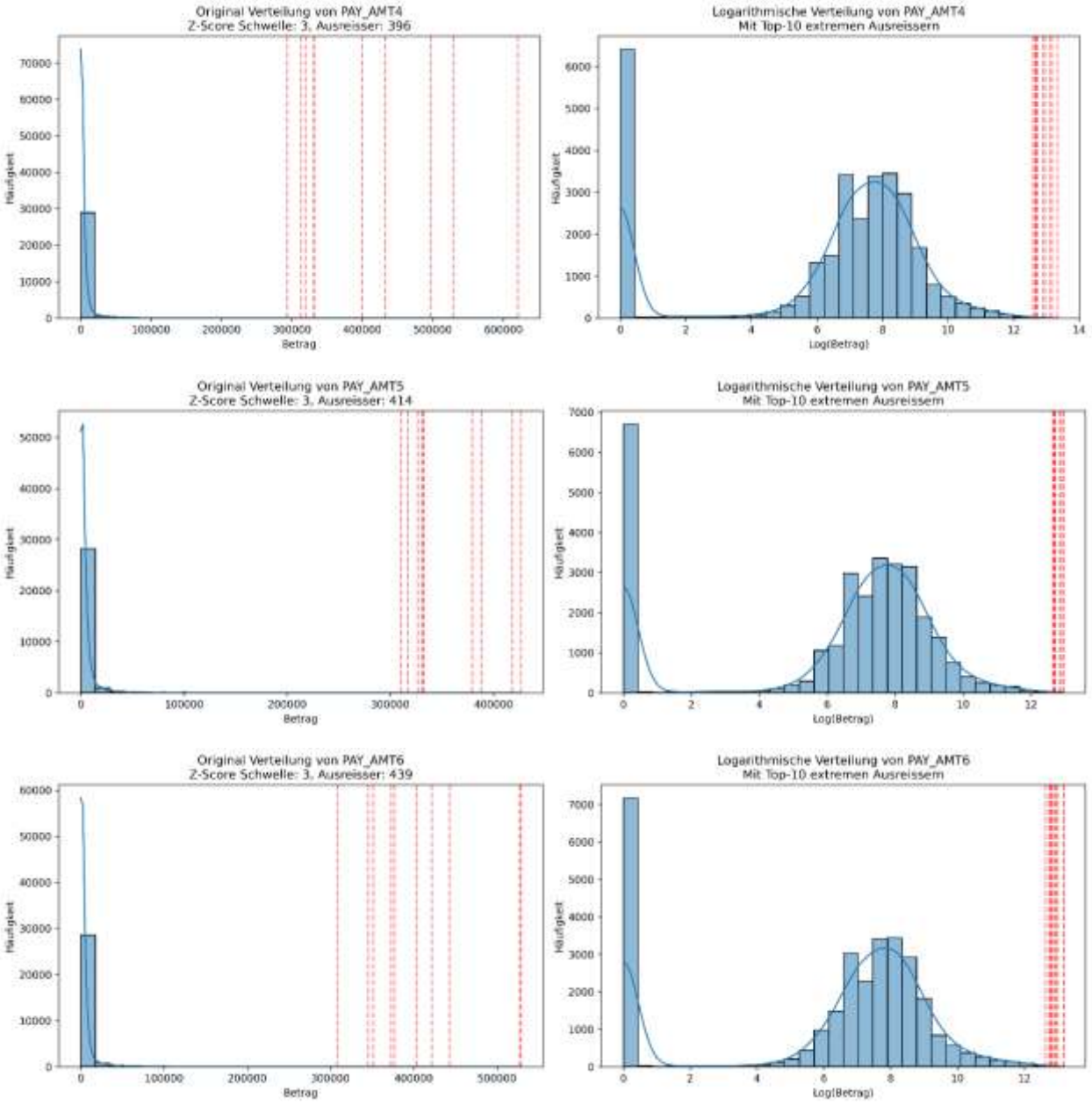

Original Verteilung von PAY_AMT6
Häufigkeit
Betrag
Logarithmische Verteilung von PAY_AMT6
Häufigkeit
Log(Betrag)


Original Verteilung von PAY_AMT1
Häufigkeit
Betrag
Logarithmische Verteilung von PAY_AMT1
Häufigkeit
Log(Betrag)

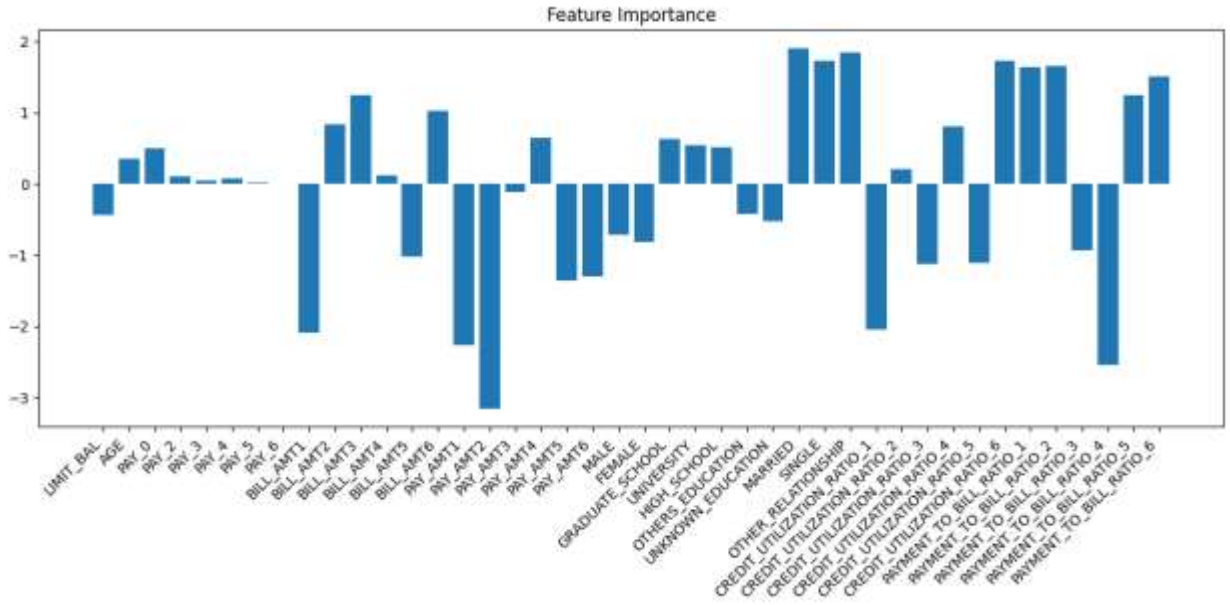

Original Verteilung von PAY_AMT2
Häufigkeit
Betrag
1e6
Logarithmische Verteilung von PAY_AMT2
Häufigkeit
Log(Betrag)

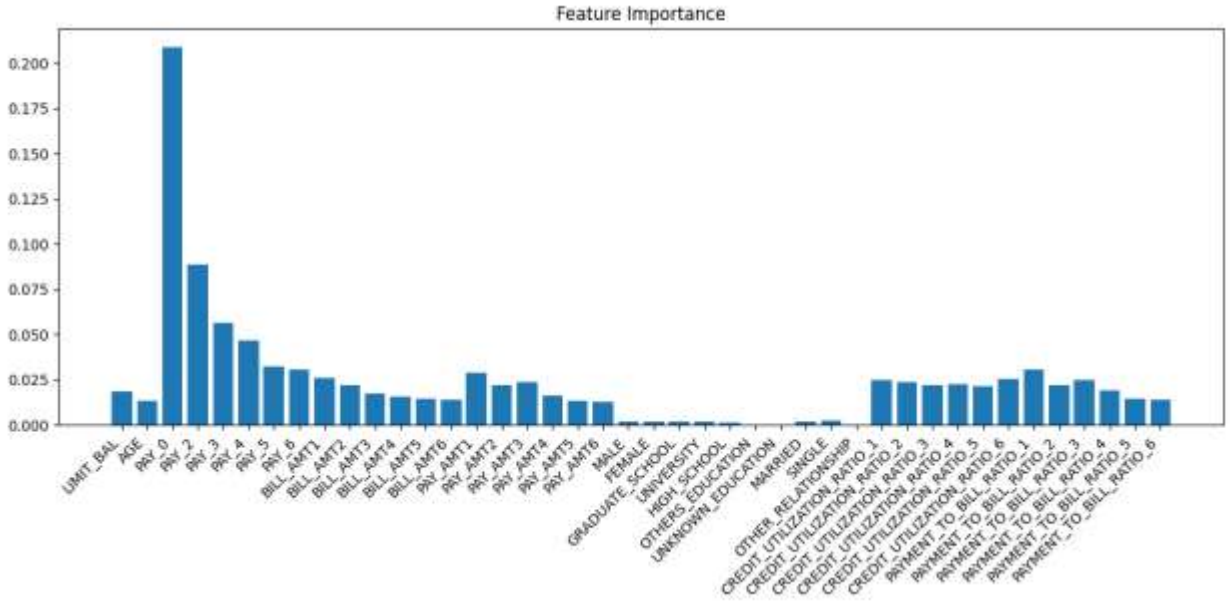
8.2.1.2.3 Z-Score

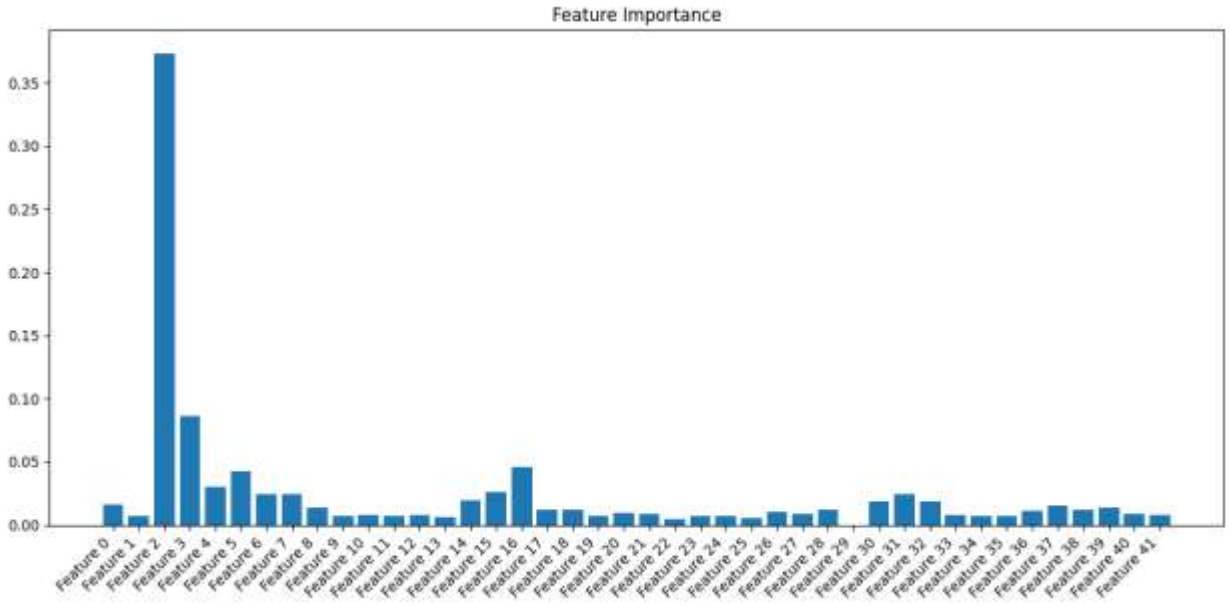



Original Verteilung von PAY_AMT4
Z-Score Schwelle: 3, Ausreisser: 396
Häufigkeit
Betrag
Logarithmische Verteilung von PAY_AMT4
Mit Top-10 extremen Ausreissern
Häufigkeit
Log(Betrag)
Original Verteilung von PAY_AMT5
Z-Score Schwelle: 3, Ausreisser: 414
Häufigkeit
Betrag
Logarithmische Verteilung von PAY_AMT5
Mit Top-10 extremen Ausreissern
Häufigkeit
Log(Betrag)
Original Verteilung von PAY_AMT6
Z-Score Schwelle: 3, Ausreisser: 439
Häufigkeit
Betrag
Logarithmische Verteilung von PAY_AMT6
Mit Top-10 extremen Ausreissern
Häufigkeit
Log(Betrag)

8.2.2 Feauture importance

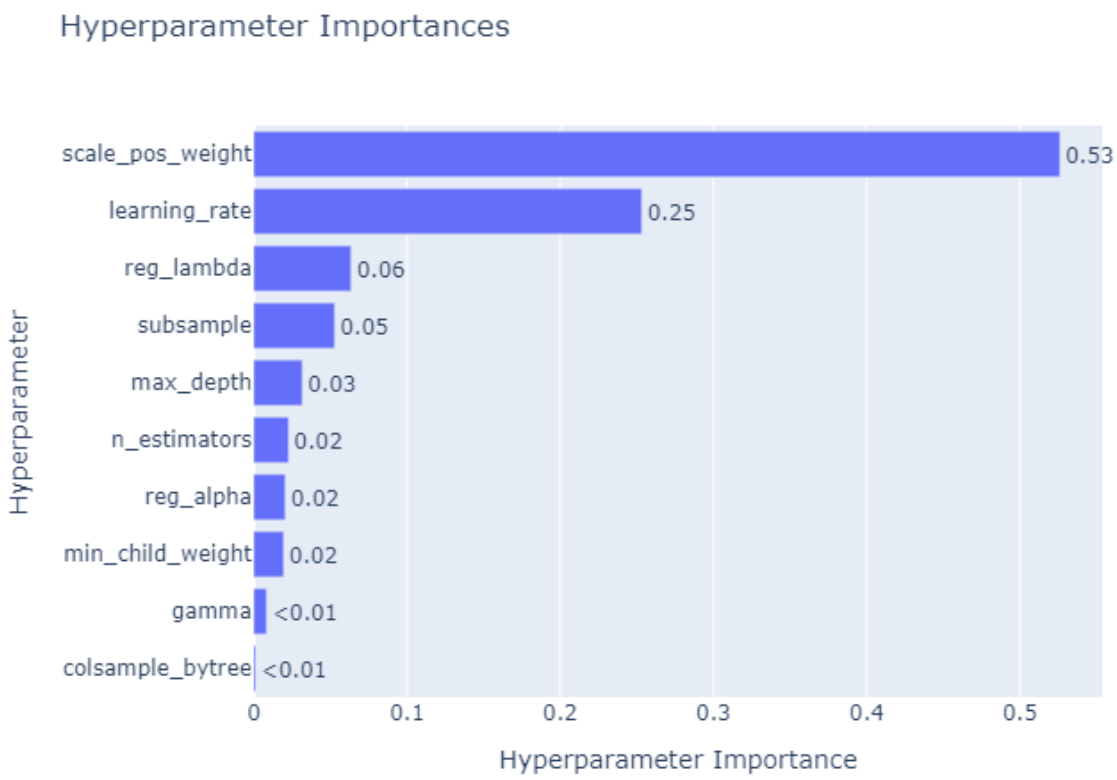

Feature Importance
2
1
0
−1
−2
−3
LIMIT_BAL
AGE
PAY_0
PAY_2
PAY_3
PAY_4
PAY_5
PAY_6
BILL_AMT1
BILL_AMT2
BILL_AMT3
BILL_AMT4
BILL_AMT5
BILL_AMT6
PAY_AMT1
PAY_AMT2
PAY_AMT3
PAY_AMT4
PAY_AMT5
PAY_AMT6
MALE
FEMALE
GRADUATE_SCHOOL
UNIVERSITY
HIGH_SCHOOL
OTHERS_EDUCATION
UNKNOWN_EDUCATION
MARRIED
SINGLE
OTHER_RELATIONSHIP
CREDIT_UTILIZATION_RATIO_1
CREDIT_UTILIZATION_RATIO_2
CREDIT_UTILIZATION_RATIO_3
CREDIT_UTILIZATION_RATIO_4
CREDIT_UTILIZATION_RATIO_5
CREDIT_UTILIZATION_RATIO_6
PAYMENT_TO_BILL_RATIO_1
PAYMENT_TO_BILL_RATIO_2
PAYMENT_TO_BILL_RATIO_3
PAYMENT_TO_BILL_RATIO_4
PAYMENT_TO_BILL_RATIO_5
PAYMENT_TO_BILL_RATIO_6

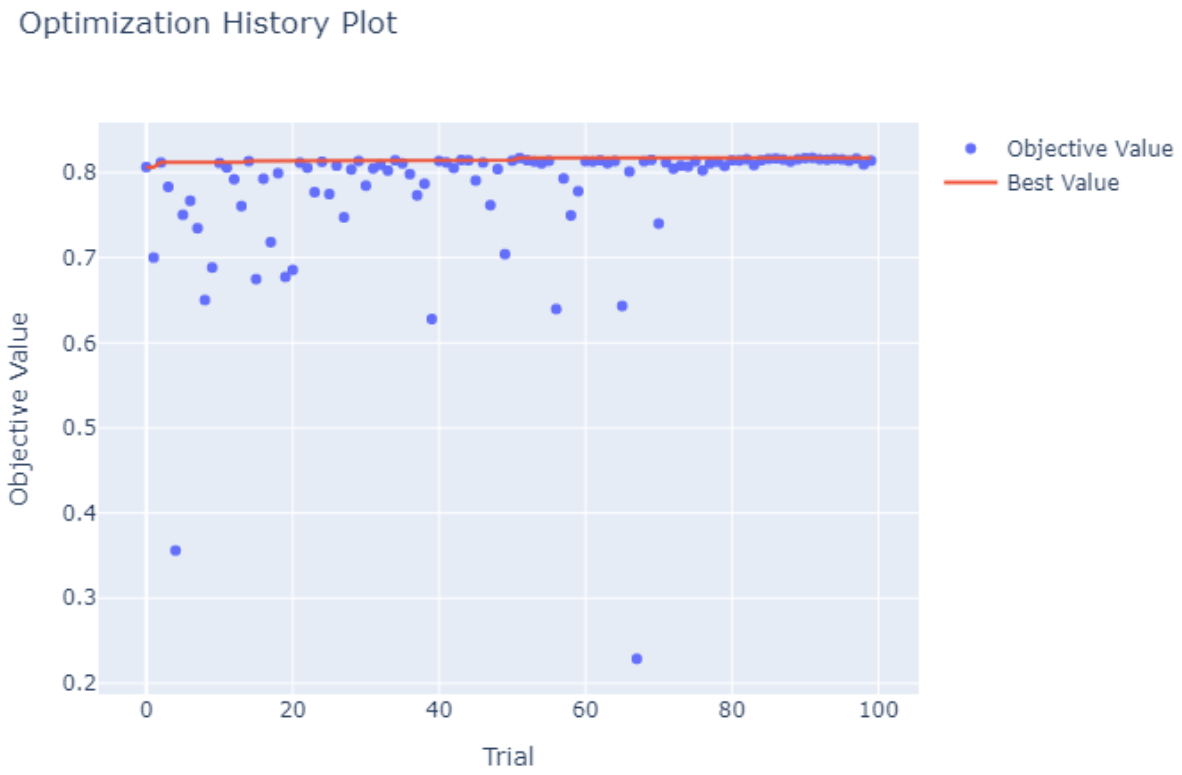

Feature Importance
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
LIMIT_BAL
AGE
PAY_0
PAY_2
PAY_3
PAY_4
PAY_5
PAY_6
BILL_AMT1
BILL_AMT2
BILL_AMT3
BILL_AMT4
BILL_AMT5
BILL_AMT6
PAY_AMT1
PAY_AMT2
PAY_AMT3
PAY_AMT4
PAY_AMT5
PAY_AMT6
MALE
FEMALE
GRADUATE_SCHOOL
UNIVERSITY
HIGH_SCHOOL
OTHERS_EDUCATION
UNKNOWN_EDUCATION
MARRIED
SINGLE
OTHER_RELATIONSHIP
CREDIT_UTILIZATION_RATIO_1
CREDIT_UTILIZATION_RATIO_2
CREDIT_UTILIZATION_RATIO_3
CREDIT_UTILIZATION_RATIO_4
CREDIT_UTILIZATION_RATIO_5
CREDIT_UTILIZATION_RATIO_6
PAYMENT_TO_BILL_RATIO_1
PAYMENT_TO_BILL_RATIO_2
PAYMENT_TO_BILL_RATIO_3
PAYMENT_TO_BILL_RATIO_4
PAYMENT_TO_BILL_RATIO_5
PAYMENT_TO_BILL_RATIO_6

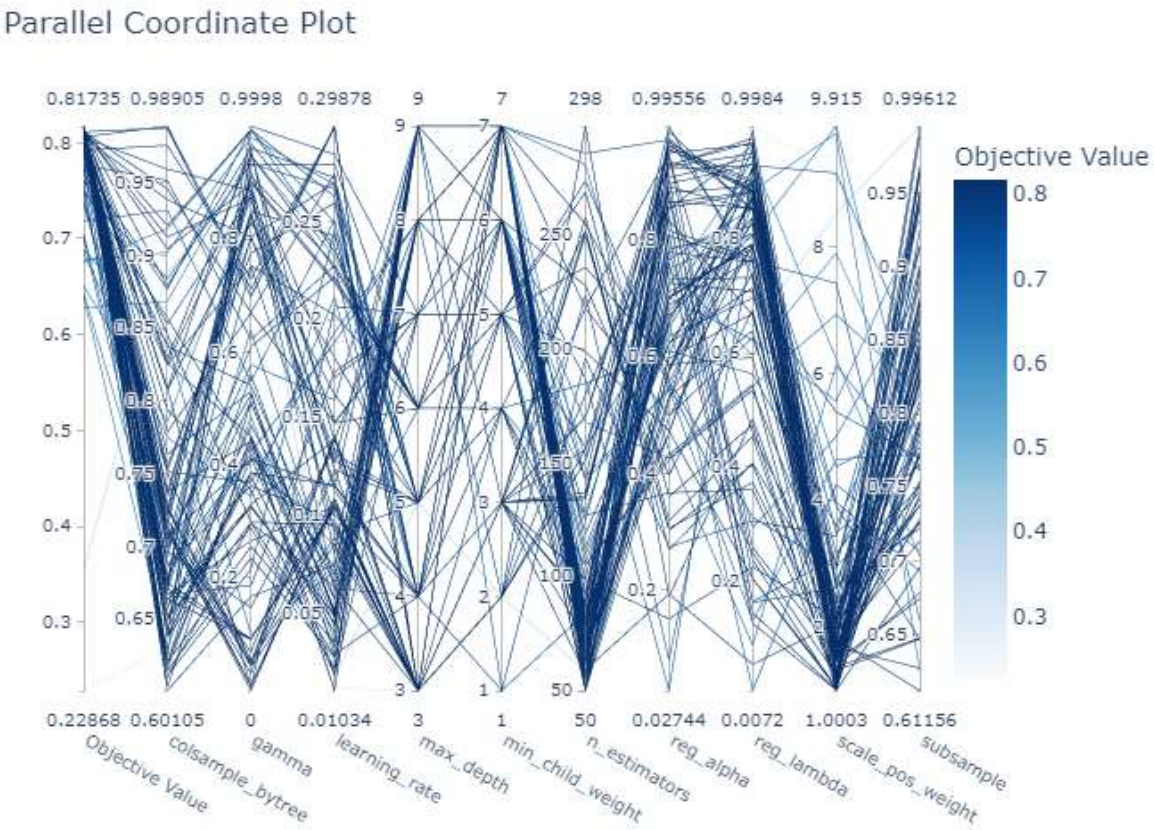

Feature Importance
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Feature 0
Feature 1
Feature 2
Feature 3
Feature 4
Feature 5
Feature 6
Feature 7
Feature 8
Feature 9
Feature 10
Feature 11
Feature 12
Feature 13
Feature 14
Feature 15
Feature 16
Feature 17
Feature 18
Feature 19
Feature 20
Feature 21
Feature 22
Feature 23
Feature 24
Feature 25
Feature 26
Feature 27
Feature 28
Feature 29
Feature 30
Feature 31
Feature 32
Feature 33
Feature 34
Feature 35
Feature 36
Feature 37
Feature 38
Feature 39
Feature 40
Feature 41

8.2.3 Optuna

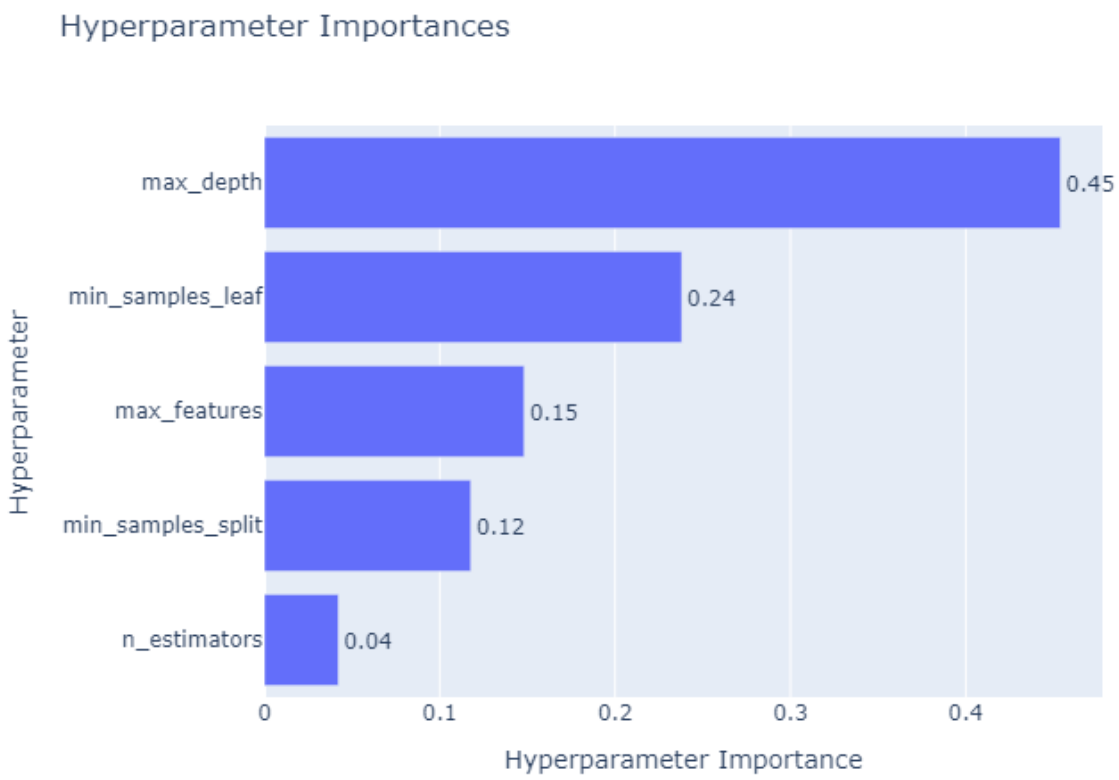
8.2.3.1 XGBoost

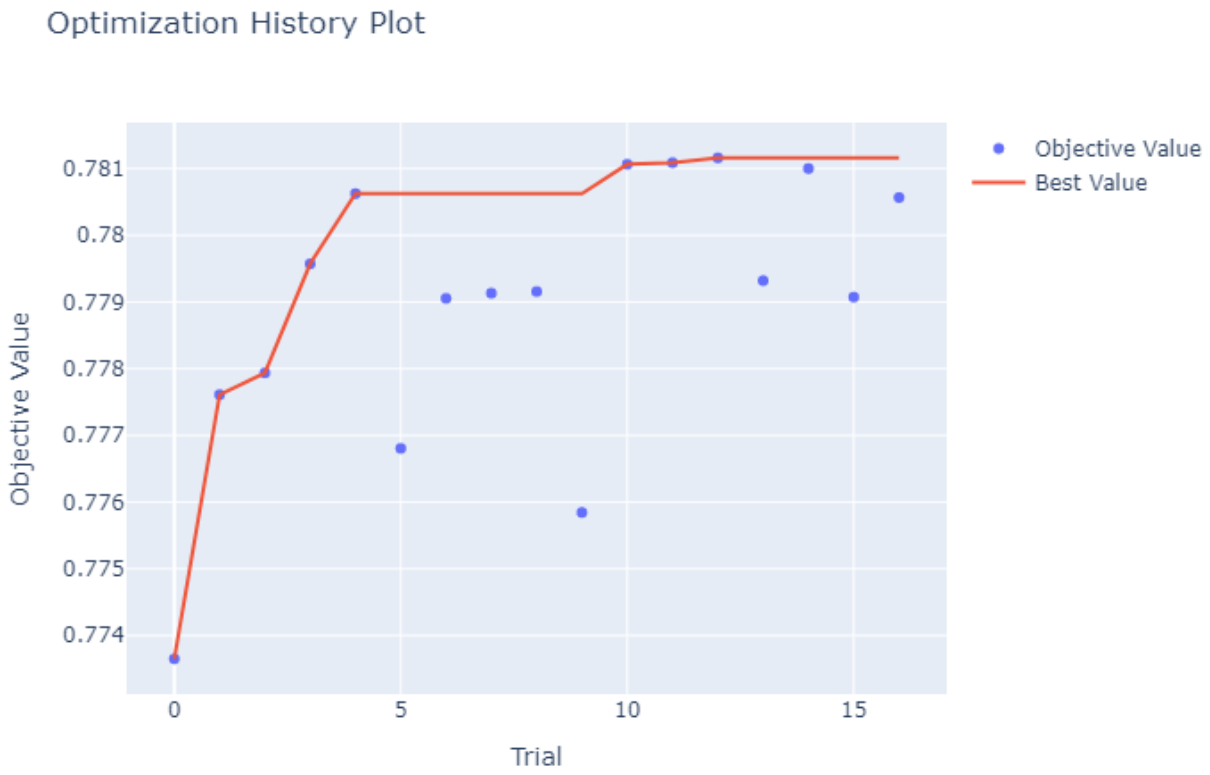

Hyperparameter Importances
scale_pos_weight
learning_rate
reg_lambda
subsample
max_depth
n_estimators
reg_alpha
min_child_weight
gamma
colsample_bytree
0.53
0.25
0.06
0.05
0.03
0.02
0.02
0.02
<0.01
<0.01
Hyperparameter
0
0.1
0.2
0.3
0.4
0.5
Hyperparameter Importance

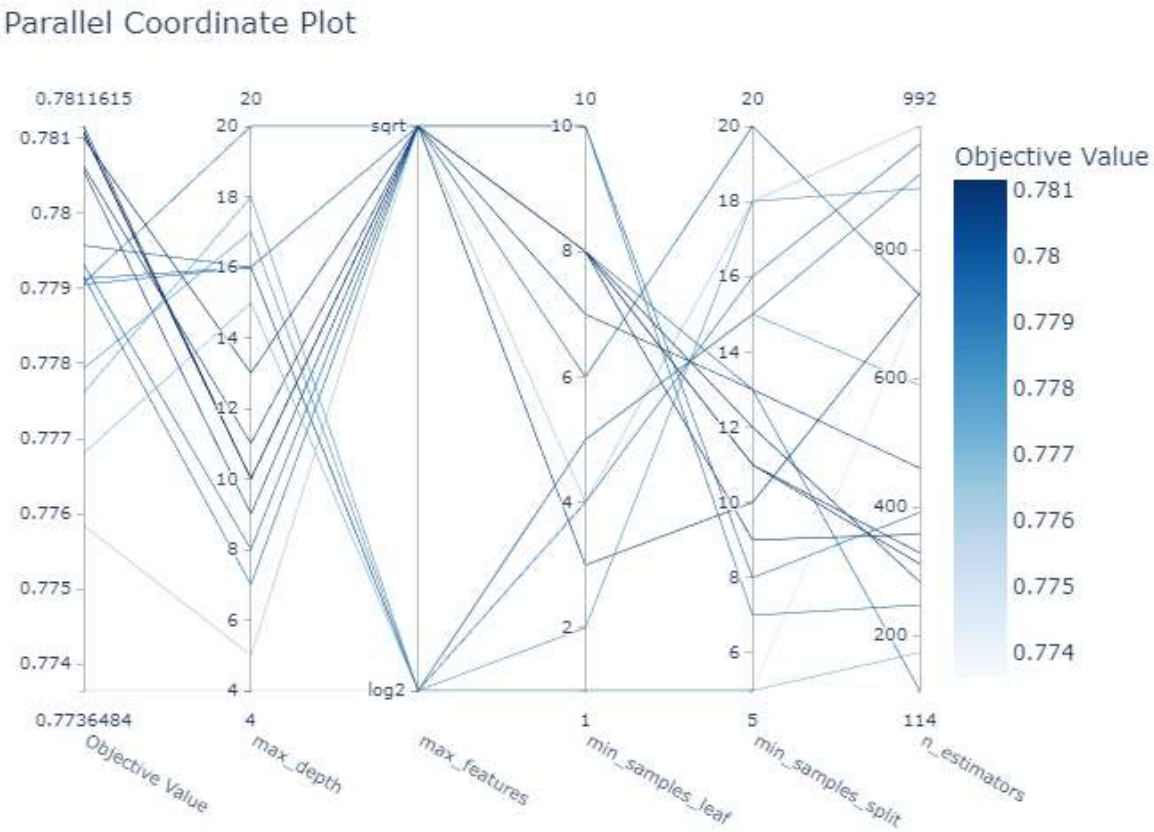

Optimization History Plot
Objective Value
Best Value
Objective Value
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
20
40
60
80
100
Trial


Parallel Coordinate Plot
Objective Value
0.81735 0.98905 0.9998 0.29878 9 7 298 0.99556 0.9984 9.915 0.99612
0.22868 0.60105 0 0.01034 3 1 50 0.02744 0.0072 1.0003 0.61156
Objective Value
colsample_bytree
gamma
learning_rate
max_depth
min_child_weight
n_estimators
reg_alpha
reg_lambda
scale_pos_weight
subsample

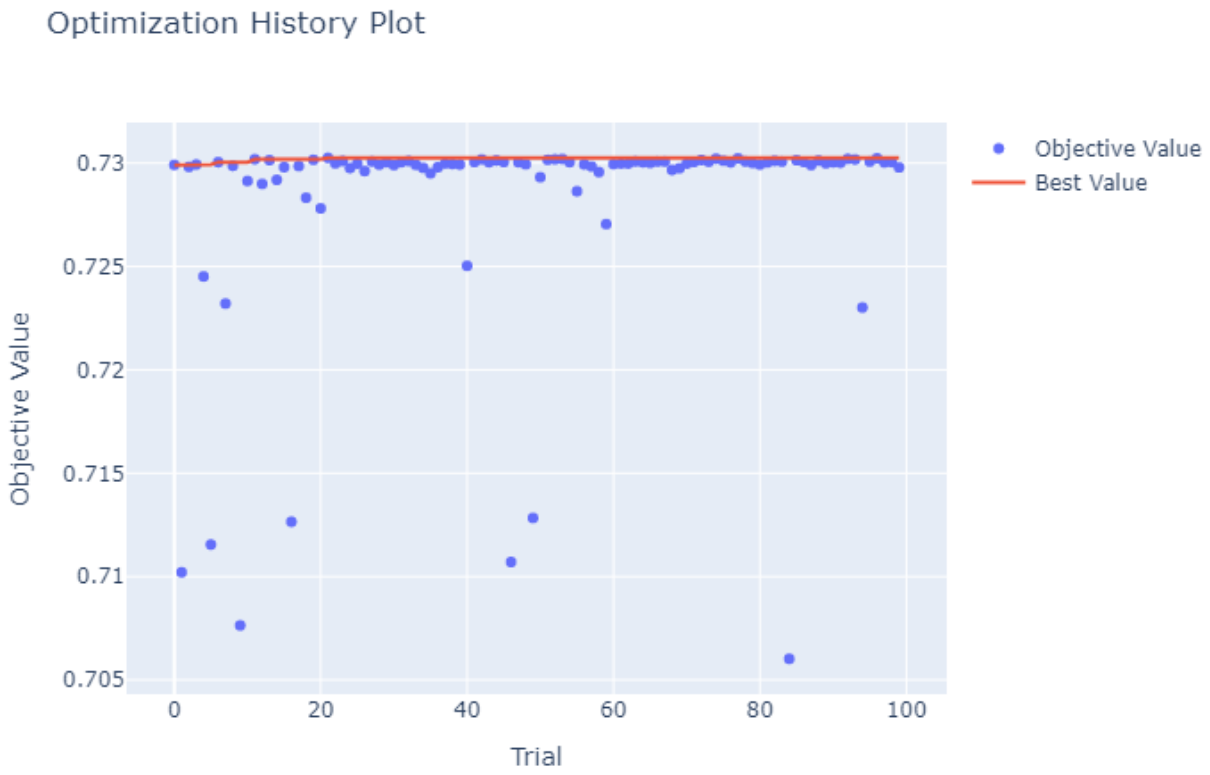
8.2.3.2 Random Forest

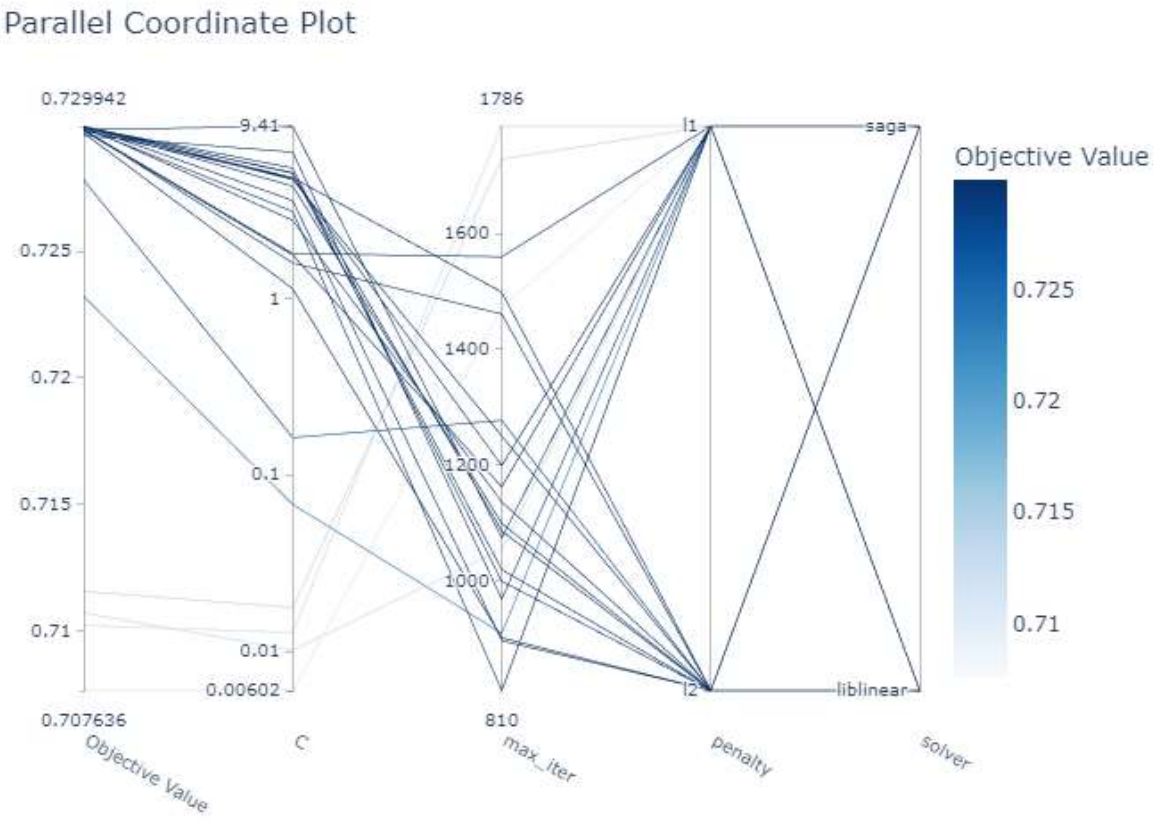

Hyperparameter Importances
max_depth
min_samples_leaf
max_features
min_samples_split
n_estimators
0.45
0.24
0.15
0.12
0.04
Hyperparameter
0
0.1
0.2
0.3
0.4
Hyperparameter Importance

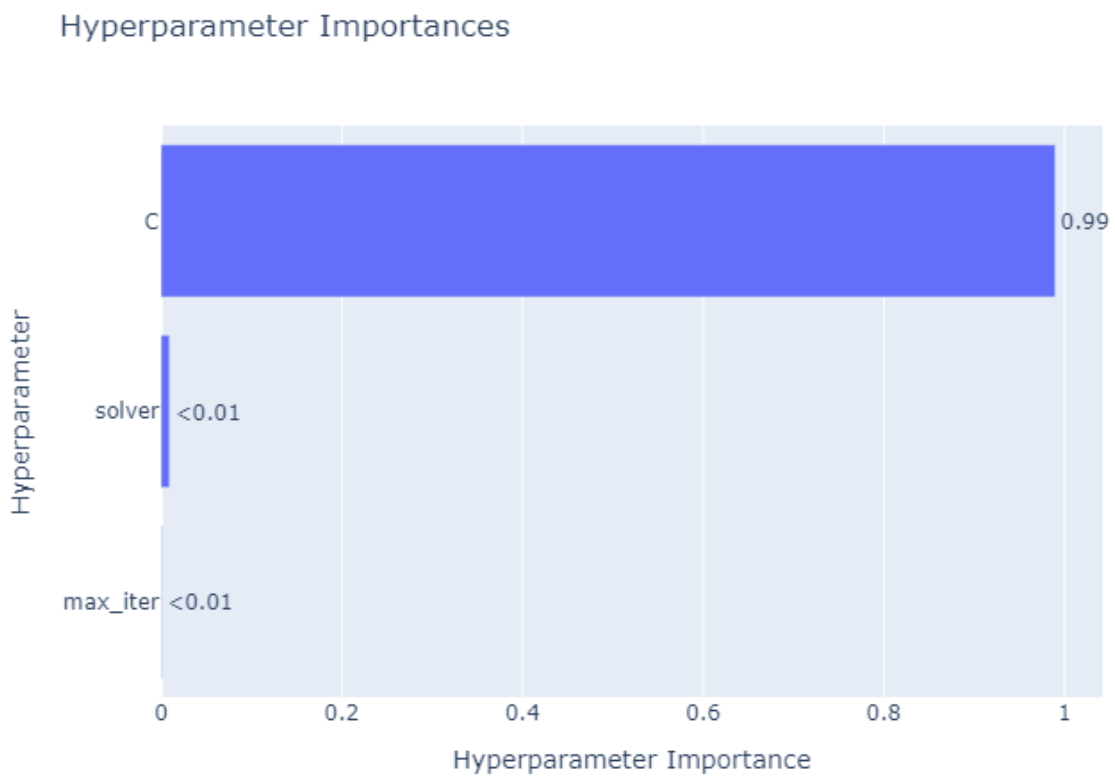

Optimization History Plot
Objective Value
Best Value
0.781
0.78
0.779
0.778
0.777
0.776
0.775
0.774
Objective Value
0
5
10
15
Trial


Parallel Coordinate Plot
Objective Value
0.781
0.78
0.779
0.778
0.777
0.776
0.775
0.774
0.7811615
0.7736484
20
4
sqrt
log2
10
1
20
5
992
114
800
600
400
200
max_depth
max_features
min_samples_leaf
min_samples_split
n_estimators

8.2.3.3 Logistische Regression

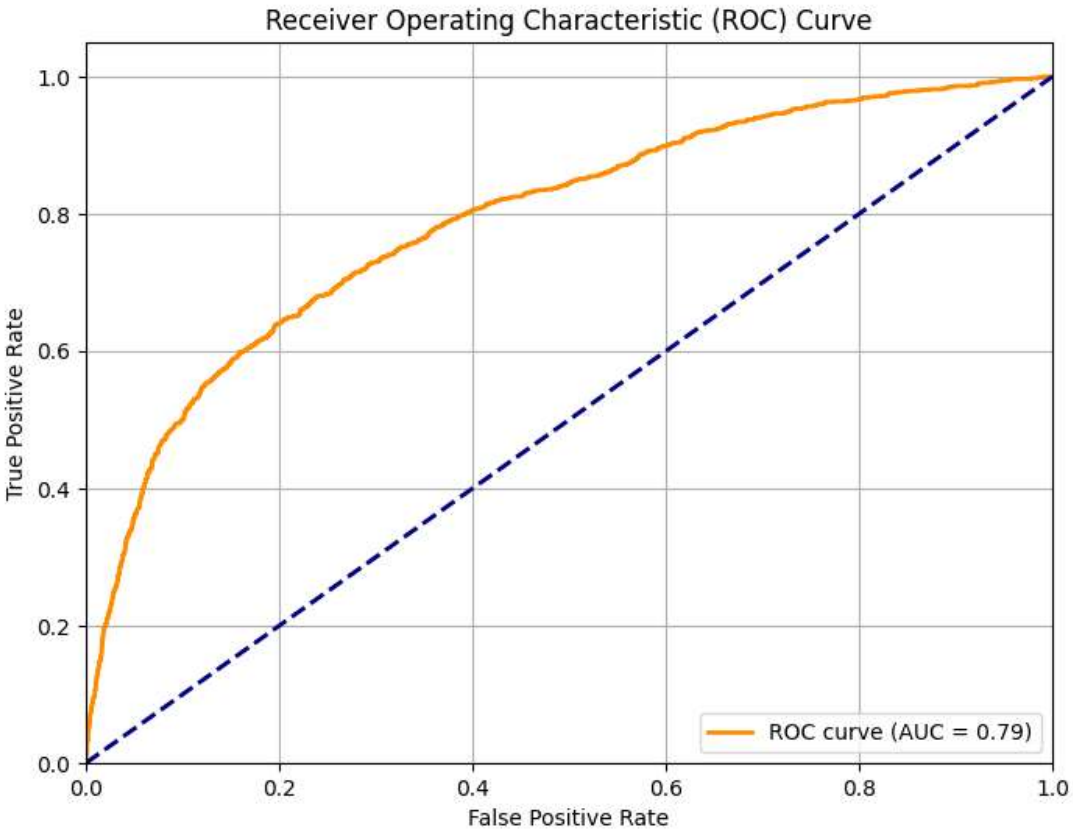

Optimization History Plot
Objective Value
Best Value
0.73
0.725
0.72
0.715
0.71
0.705
Objective Value
0
20
40
60
80
100
Trial


Parallel Coordinate Plot
0.729942
0.725
0.72
0.715
0.71
0.707636
Objective Value
9.41
1
0.1
0.01
0.00602
C
1786
1600
1400
1200
1000
810
max_iter
l1
l2
penalty
saga
liblinear
solver
Objective Value
0.725
0.72
0.715
0.71

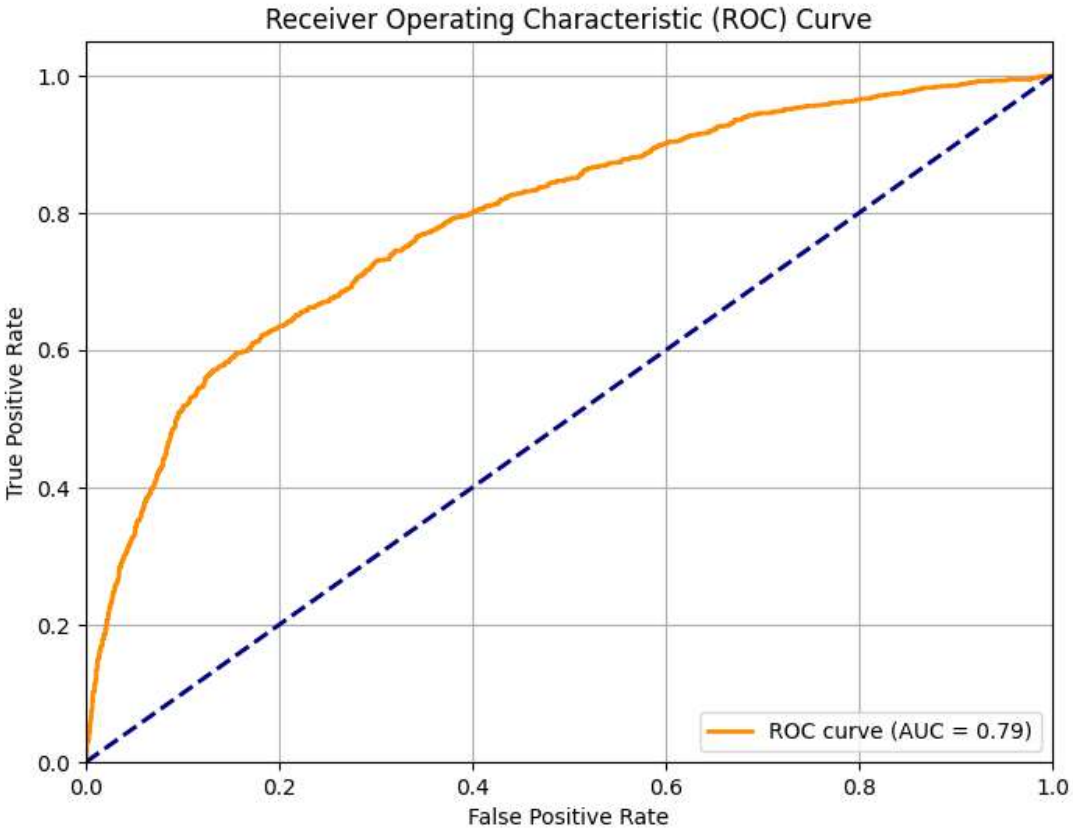

Hyperparameter Importances
C
solver
max_iter
0.99
<0.01
<0.01
Hyperparameter
0
0.2
0.4
0.6
0.8
1
Hyperparameter Importance

8.2.4 ROC-Kurve

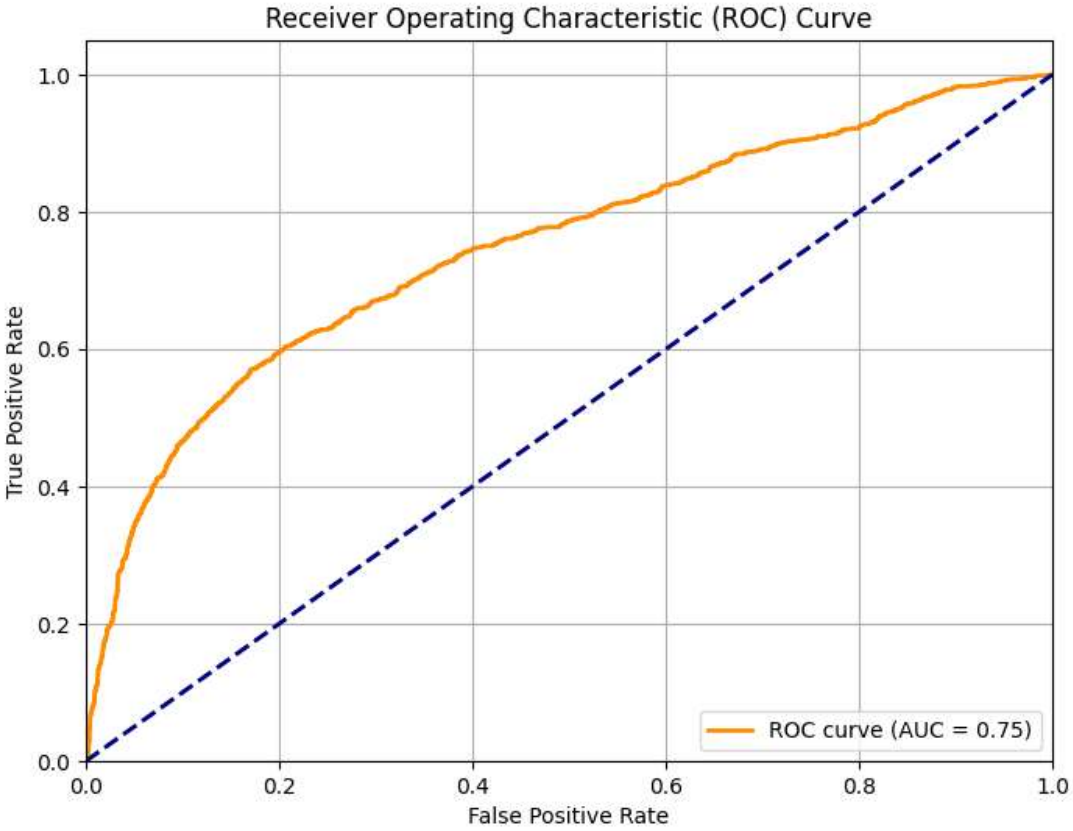
8.2.4.1 XGBoost



8.2.4.2 Random Forest



8.2.5 Logistische Regression



9 SCHLUSSERKLÄRUNG

„Hiermit versichere ich, dass die vorliegende Arbeit selbstständig angefertigt wurde“

Noel Gugler: